GCP-Specific Questions

What is Google Cloud Identity and Access Management (IAM)?

Google Cloud IAM is GCP's unified access control system managing who can do what on which resources.

Core concepts:

- **Members** (identities) include:
    - Google accounts (individual users)
    - Service accounts (applications and VMs)
    - Google Groups (collections of users)
    - Google Workspace domains (entire organization)
    - Cloud Identity domains (GCP-only organizations without Workspace)
- **Roles** are collections of permissions defining what actions members can perform. GCP has three role types:
    - **Primitive roles** (Owner, Editor, Viewer - broad, legacy roles with extensive permissions across all services, generally avoid these).
    - **Predefined roles** (curated by Google for specific services like `roles/compute.instanceAdmin` or `roles/storage.objectViewer` - follow least privilege).
    - **Custom roles** (organization-defined roles with specific permissions for unique requirements).
- **Permissions** are granular access controls in format `service.resource.verb` like `compute.instances.delete` or `storage.objects.get`.
- **Resources** are GCP entities (projects, instances, buckets) organized hierarchically: Organization → Folders → Projects → Resources.
- **Policy** is the binding of members to roles on specific resources defining who has what access.

How it works: IAM policies are set at any level of the resource hierarchy and inherited downward - policy set at organization level applies to all folders, projects, and resources below. You grant roles to members at appropriate hierarchy level:

```
gcloud projects add-iam-policy-binding PROJECT_ID --member="user:[email protected]"
--role="roles/viewer"
```

Multiple policies combine with union of permissions (least restrictive wins unless explicitly denied).

Key differences from AWS IAM:

- GCP IAM is resource-centric (permissions attached to resources) vs. AWS's identity-centric approach (permissions attached to identities).

- No separate groups concept (uses Google Groups).
- Simpler policy structure (no complex JSON).
- Inheritance through resource hierarchy (powerful but requires careful planning).

Service accounts are special account type for applications:

- Automatically created for many GCP services.
- Can be used as identity for VMs and applications.
- Have their own keys for authentication outside GCP.
- Should follow least privilege strictly.

Best practices:

- Use predefined roles over primitive roles.
- Grant roles at lowest necessary hierarchy level.
- Use groups instead of individual users for easier management.
- Regularly audit IAM policies with Policy Analyzer.
- Use service accounts for application access not user accounts.
- Enable Cloud Audit Logs tracking all IAM changes.
- Implement Organization Policy constraints enforcing IAM standards.
- Rotate service account keys regularly (or use Workload Identity eliminating keys).

Conditions allow context-aware access: time-based access (only during business hours), resource-based conditions (specific resource attributes), and IP-based restrictions. Example policy with condition: role granted only if request from corporate IP range and during weekdays.

IAM is foundational to GCP security - misconfigurations here expose entire cloud environment.

Explain the role of Google Cloud Security Command Center in GCP.

Security Command Center (SCC) is GCP's centralized security and risk management platform providing visibility, threat detection, and compliance monitoring across GCP resources.

Core capabilities:

- **Asset discovery and inventory** - automatically discovers all GCP resources across organization (compute instances, storage buckets, databases, IAM policies), maintains complete asset inventory with metadata and configurations, tracks asset changes over time, and provides centralized view across projects and folders.
- **Vulnerability detection** - integrates with Web Security Scanner finding vulnerabilities in App Engine, GCE, and GKE web applications, identifies common vulnerabilities like XSS, CSRF, mixed content, and provides CVE information for OS and application packages.
- **Threat detection** - analyzes logs and configurations detecting anomalous activity: suspicious API calls, cryptocurrency mining, data exfiltration attempts, malware detection on VMs, and unauthorized access patterns. Uses machine learning establishing baselines and detecting deviations.
- **Compliance monitoring** - built-in compliance dashboards for standards (PCI DSS, HIPAA, ISO 27001, CIS GCP Benchmarks), continuous compliance checking against security standards, identifies non-compliant resources with remediation guidance, and generates compliance reports for audits.
- **Security findings management** - aggregates findings from multiple sources (SCC built-in detectors, Event Threat Detection, Container Threat Detection, Web Security Scanner, third-party integrations), prioritizes findings by severity and exploitability, provides detailed finding information with remediation steps, and tracks finding lifecycle from detection to resolution.

Tiers:

- **Standard tier** (free) - asset discovery and inventory, Security Health Analytics for misconfigurations, Web Security Scanner (limited scans), and basic compliance dashboards.
- **Premium tier** (paid) - Event Threat Detection analyzing Cloud Logging for threats, Container Threat Detection for GKE, continuous exports to BigQuery or Pub/Sub, premium compliance dashboards and reporting, integration with third-party SIEM and SOAR tools, and advanced threat detection capabilities.

Security Health Analytics - automatically detects misconfigurations: public storage buckets, overly permissive IAM bindings, disabled audit logging, weak firewall rules, unencrypted resources, and SSL certificate issues. Runs continuously checking resources against security best practices.

Integration and automation: Findings exported to Pub/Sub enabling real-time notifications and automated response, BigQuery exports for analysis and trending, integration with Cloud Functions for automated remediation, SIEM integration sending findings to Splunk, Chronicle, or other SIEM, and Security Command Center API for programmatic access.

Use cases: Security teams use SCC as single pane of glass for security posture across all GCP projects, compliance teams generate audit reports showing alignment with regulatory requirements, incident response teams investigate findings and track remediation, and DevOps teams receive notifications about misconfigurations for quick fixes.

Example workflow: SCC detects public Cloud Storage bucket → generates HIGH severity finding → exports to Pub/Sub → Cloud Function triggered → Function applies bucket policy blocking public access → Function updates finding status to remediated → Security team notified.

Limitations: SCC focuses on GCP-native resources (limited visibility into applications running on GCP), findings require tuning to reduce false positives, and premium tier required for advanced detection capabilities.

SCC is essential for maintaining security visibility in GCP environments, especially in organizations with many projects where manual monitoring is impractical.

How can you secure Google Kubernetes Engine (GKE) clusters?

Securing GKE requires controls at multiple layers.

Cluster configuration:

- **Private clusters** - create clusters with private endpoints where nodes have only private IPs, master endpoint accessible only from authorized networks or via Cloud VPN/Interconnect, and `--enable-private-nodes` and `--enable-private-endpoint` flags during creation. This prevents direct internet access to cluster.
- **Workload Identity** - use instead of service account keys: enables Kubernetes service accounts to act as GCP service accounts, eliminates need to manage and distribute service account keys, pods automatically get credentials for GCP APIs, and configured with `--workload-pool=PROJECT_ID.svc.id.goog`.
- **Shielded GKE Nodes** - enable for secure boot and integrity monitoring: `--enable-shielded-nodes` provides verifiable node identity, protects against rootkits and bootkits, and uses Secure Boot and vTPM.

Network security:

- **Network policies** - enable Calico or GKE native network policies: control pod-to-pod traffic with Kubernetes NetworkPolicy resources, default deny all traffic then allow specific required communications, and micro-segmentation within cluster. Example: allow only frontend pods to communicate with backend pods on specific ports.
- **Private Google Access** - enable for nodes to access GCP APIs without public IPs, traffic stays on Google's network not internet.
- **Authorized networks** - restrict cluster control plane access to specific IP ranges: `--enable-master-authorized-networks --master-authorized-networks=CIDR_RANGE`, prevents unauthorized access to Kubernetes API server.
- **Binary Authorization** - enforce only verified container images can be deployed:

  ```
  gcloud container clusters update CLUSTER --enable-binauthz
  ```

  Integrates with Container Analysis checking vulnerabilities before deployment, requires images signed by trusted authorities, and prevents deployment of unsigned or vulnerable images.

Container security:

- **Container-Optimized OS** - use GCP's hardened OS for GKE nodes: minimal attack surface with only essential packages, automatic security updates, and read-only root filesystem.
- **Vulnerability scanning** - enable Container Analysis automatically scanning images pushed to Container Registry/Artifact Registry, identifies CVEs in base images and application dependencies, blocks deployment of images with critical vulnerabilities via Binary

Authorization, and continuous scanning detects new vulnerabilities in existing images.

- **Pod Security Standards** - enforce pod security policies: run containers as non-root user, drop unnecessary Linux capabilities, use read-only root filesystem, disable privilege escalation, and restrict volume types. Implement via PodSecurityPolicy (deprecated) or Pod Security Standards (replacement).

Secrets management:

- **Use Secrets, not ConfigMaps** - store sensitive data in Kubernetes Secrets not ConfigMaps, encrypt Secrets at rest with application-layer encryption, and integrate with Secret Manager for additional protection.
- **Workload Identity for secrets** - applications use Workload Identity accessing GCP Secret Manager, eliminates secrets stored in cluster, and automatic rotation without pod restarts.

Access control:

- **RBAC** - implement granular role-based access control: create service accounts for each application with minimal permissions, use RoleBindings limiting access to specific namespaces, avoid cluster-admin role except for administrators, and regularly audit RBAC policies.
- **GKE IAM integration** - combine Kubernetes RBAC with GCP IAM: GCP IAM controls who can access cluster (get cluster credentials), Kubernetes RBAC controls what they can do inside cluster.

Monitoring and logging:

- **Cloud Logging** - enable GKE logging sending cluster logs to Cloud Logging: audit logs tracking administrative actions, system logs from nodes, and application logs from containers.
- **Cloud Monitoring** - collect metrics detecting anomalies, alert on suspicious activity (unusual API calls, failed authentication), and monitor resource usage for cryptocurrency mining.
- **Audit logging** - enable Kubernetes audit logs capturing all API server requests, track who accessed what resources, and detect unauthorized access attempts.

Additional hardening:

- **Disable legacy endpoints** - remove legacy ABAC and basic authentication, disable legacy metadata API v1, and use only supported authentication methods.
- **Automatic upgrades and repairs** - enable auto-upgrade for nodes ensuring latest security patches: `--enable-autoupgrade`, and auto-repair detecting and replacing unhealthy nodes: `--enable-autorepair`.
- **Resource quotas and limits** - implement ResourceQuotas preventing resource exhaustion, LimitRanges ensuring containers specify resource requests/limits, and PodDisruptionBudgets for availability.

Compliance:

- **GKE Sandbox** - use gVisor for additional isolation running untrusted workloads, provides defense in depth against container breakout, and enabled per-node pool.

- **CIS Benchmarks** - configure clusters following CIS Kubernetes Benchmark recommendations, Security Command Center checks compliance, and remediate findings.

Example secure GKE cluster creation:

```
gcloud container clusters create secure-cluster \
  --enable-private-nodes \
  --enable-private-endpoint \
  --master-ipv4-cidr 172.16.0.0/28 \
  --enable-ip-alias \
  --enable-master-authorized-networks \
  --master-authorized-networks=CORPORATE_CIDR \
  --enable-shielded-nodes \
  --enable-autorepair \
  --enable-autoupgrade \
  --workload-pool=PROJECT_ID.svc.id.goog \
  --enable-binauthz \
  --enable-stackdriver-kubernetes \
  --addons=HttpLoadBalancing,HorizontalPodAutoscaling,NetworkPolicy \
  --network-policy \
  --zone us-central1-a
```

This comprehensive approach creates defense in depth for GKE clusters protecting against common attack vectors.

Describe how Google Cloud Armor helps protect applications running on GCP.

Cloud Armor is GCP's DDoS protection and web application firewall (WAF) service defending applications from attacks.

Core capabilities:

- **DDoS protection** - automatic protection against network and protocol layer attacks (L3/L4): absorbs volumetric attacks leveraging Google's global infrastructure, protects against SYN floods, UDP amplification, ICMP floods, and provides always-on protection without configuration. Adaptive protection (premium tier) uses machine learning detecting and mitigating application-layer DDoS attacks automatically.
- **WAF functionality** - protects against OWASP Top 10 vulnerabilities at L7: SQL injection, cross-site scripting (XSS), remote code execution, and local file inclusion. Configurable security policies with custom rules matching request attributes (IP, headers, geography, request path) and pre-configured rules for common attack patterns (Google-managed ModSecurity Core Rule Set compatible rules).
- **Rate limiting** - prevents abuse and brute force attacks: rate limits by client IP, per-session, or custom criteria, configurable actions (deny, throttle, redirect), and protects APIs from excessive requests.
- **Geofencing** - block or allow traffic based on geography: allow only specific countries accessing application, block regions with no legitimate users, and comply with data residency requirements.

How it works: Cloud Armor policies attach to GCP load balancers (HTTP(S) Load Balancer, SSL Proxy, TCP Proxy), traffic flows through load balancer inspected by Cloud Armor before reaching backend, rules evaluated in priority order (lower number = higher priority), and actions taken based on rule match (allow, deny with specific response code, throttle, or redirect).

Security policy structure: Policies contain ordered rules, each rule has priority (0-2147483647), match condition (IP address, region, headers, path, expression language for complex conditions), and action (allow, deny-403, deny-404, deny-502, throttle, or rate-based-ban). Default rule (lowest priority) handles traffic not matching other rules.

Pre-configured WAF rules: Google-managed rule sets based on ModSecurity CRS: `sqli-stable` for SQL injection protection, `xss-stable` for cross-site scripting, `lfi-stable` for local file inclusion, `rce-stable` for remote code execution, and `scannerdetection-stable` for security scanner detection. Enable with `--enable-managed-protection-tier=CA_STANDARD` or `CA_PREMIUM`.

Custom rules examples:

- Block specific IP ranges:

  ```
  gcloud compute security-policies rules create 1000 --security-policy=my-policy
  --src-ip-ranges=192.0.2.0/24 --action=deny-403
  ```

- Allow only specific countries:

  ```
  gcloud compute security-policies rules create 2000 --security-policy=my-policy
  --expression="origin.region_code in ['US', 'CA']" --action=allow
  ```

- Rate limit per IP:

  ```
  gcloud compute security-policies rules create 3000 --security-policy=my-policy
  --expression="true" --action=throttle --rate-limit-threshold-count=100 --rate-limit
  -threshold-interval-sec=60
  ```

Advanced features:

- **Adaptive Protection** (premium tier) - ML-based anomaly detection: learns normal traffic patterns, detects volumetric attacks automatically, and generates protection rules dynamically.
- **Named IP lists** - maintain reusable IP allowlists/blocklists: update centrally affecting multiple rules, and useful for known malicious IPs or trusted partners.
- **Custom expressions** - powerful rule matching using Common Expression Language (CEL): match based on request headers, cookies, query parameters, request method, user agent, and complex boolean logic. Example: `request.headers['user-agent'].contains('bot') && origin.region_code != 'US'`.

Integration and monitoring: Policies integrate with Cloud Logging logging all requests, accepted and blocked traffic visibility, Security Command Center showing Cloud Armor findings and policy compliance, and Cloud Monitoring metrics on request rates, blocked requests, and rule matches.

Use case workflow: Global e-commerce site using Cloud Armor: base policy denies all traffic by default, allow traffic from customer regions (US, EU, Asia), rate limit per IP preventing brute force (100 req/min), enable SQL injection and XSS protection rules, allow specific IPs for administrative access, and log all denied requests for analysis. During DDoS attack: Adaptive Protection detects unusual traffic spike, analyzes attack pattern automatically, generates dynamic protection rules, and mitigates attack while allowing legitimate traffic.

Best practices: Start with pre-configured rules in monitoring mode observing impacts, tune rules based on false positives before enforcing, implement allow-lists for known good actors (CDNs, monitoring services), regularly review logs identifying attack patterns, test policies in staging before production, and combine with load balancer health checks for comprehensive protection.

Limitations: Only works with GCP load balancers (can't protect resources not behind load balancer), rules evaluated in order (careful priority planning needed), and some advanced features require premium tier.

Cloud Armor provides robust protection against common web attacks and DDoS, essential for production GCP applications.

What are Google Cloud Key Management Service (KMS) and Cloud HSM?

Cloud KMS is GCP's managed service for creating, managing, and using cryptographic keys.

Key capabilities: manages symmetric and asymmetric keys for encryption, signing, and authentication; integrates with GCP services for automatic encryption (GCS, BigQuery, Compute Engine); supports external key management (BYOK - bring your own key); provides hardware-backed key protection; enables automatic and on-demand key rotation; and offers fine-grained IAM access control per key.

Key hierarchy: Keys organized in keyring→key→key version structure.

- **Keyrings** are organizational containers grouping related keys, have specific GCP location (regional or global), and cannot be deleted (permanent).
- **Keys** are logical containers for key versions, have purpose (encryption/decryption, signing, MAC), and have protection level (software, hardware HSM, external).
- **Key versions** are actual cryptographic material, multiple versions exist per key (for rotation), primary version used for encryption/signing, and old versions retained for decryption/verification.

Key types:

- **Symmetric encryption keys** - single key for encryption and decryption, AES-256-GCM algorithm, most common for data encryption.
- **Asymmetric encryption keys** - public/private key pairs, RSA or EC algorithms, used for encrypting data sent to you.
- **Asymmetric signing keys** - public/private key pairs, RSA, EC, or Ed25519 algorithms, used for digital signatures.
- **MAC signing keys** - symmetric keys for message authentication codes, HMAC algorithm.

Protection levels:

- **Software** - keys stored in Google's software infrastructure, FIPS 140-2 validated at boundaries, lowest cost option, and adequate for most use cases.
- **HSM** - keys stored in FIPS 140-2 Level 3 certified hardware security modules (Cloud HSM), higher assurance of key protection, keys never leave HSM in plaintext, and required for certain compliance scenarios.
- **External** - keys stored outside Google Cloud in external key manager, you control key material and lifecycle, GCP calls external manager for crypto operations, and maximum control for regulatory requirements.

Key rotation: Automatic rotation for symmetric encryption keys (default 90 days, configurable), creates new key version automatically, new encryptions use new version, old versions retained for decryption, and manual rotation for asymmetric keys (create new version explicitly).

Cloud HSM specifically: Cloud HSM is FIPS 140-2 Level 3 certified hardware security module integrated with Cloud KMS.

Key features: cryptographic operations occur in dedicated hardware, keys generated and stored only in HSM, keys never exist in plaintext outside HSM, physically tamper-evident devices, and meets strictest security requirements.

Use cases: financial services requiring HSM for PCI DSS, healthcare protecting PHI under HIPAA, government workloads requiring FIPS 140-2 Level 3, and any high-security environment where software protection insufficient.

Creating HSM-protected key:

```
gcloud kms keys create my-hsm-key --location=us-east1 --keyring=my-keyring
--purpose=encryption --protection-level=hsm --rotation-period=90d --next-rotation
-time=2026-04-01T00:00:00Z
```

Encryption context and additional authenticated data (AAD): Cloud KMS supports AAD in encryption operations adding context to encryption:

```
gcloud kms encrypt
--key=projects/PROJECT/locations/LOCATION/keyRings/RING/cryptoKeys/KEY --plaintext
-file=plaintext.txt --ciphertext-file=ciphertext.enc --additional-authenticated
-data="context=production,app=payroll"
```

Decryption requires providing same AAD preventing ciphertext use in wrong context.

Access control: IAM permissions control key access: `roles/cloudkms.cryptoKeyEncrypterDecrypter` for encryption/decryption, `roles/cloudkms.admin` for key management, and `roles/cloudkms.viewer` for read-only access. Grant at keyring or individual key level, use service accounts for application access, and separate encryption from decryption permissions when possible.

Integration with GCP services: BigQuery encrypts tables with KMS keys, Compute Engine encrypts disks with customer-managed keys, Cloud Storage uses KMS for bucket encryption, Cloud SQL supports customer-managed encryption keys, and GKE secrets encrypted with KMS.

Monitoring and audit: Cloud Audit Logs tracks all KMS API calls, Cloud Monitoring provides key usage metrics, and export logs to BigQuery for analysis.

External Key Manager (EKM): For organizations requiring key material outside GCP: keys remain in your external key management system, GCP calls external system for cryptographic operations, you maintain complete control over key lifecycle, and useful for regulatory requirements demanding on-premises key control.

Best practices: Use HSM protection for highly sensitive data, enable automatic rotation for encryption keys, implement least privilege IAM on keys, separate keys by environment and data classification, enable audit logging for all key operations, regularly review key access and usage, test key rotation procedures, and maintain disaster recovery for key material.

Comparison to AWS KMS: Similar concepts but different terminology (Cloud KMS has keyrings vs AWS's key aliases), Cloud KMS offers external key management more directly, both provide HSM-backed protection, and GCP's global keyrings can be useful for multi-region applications.

Cloud KMS and Cloud HSM provide enterprise-grade key management essential for regulatory compliance and data protection in GCP.

How does Google Cloud Logging and Monitoring assist in security?

Cloud Logging (formerly Stackdriver Logging) and Cloud Monitoring (formerly Stackdriver Monitoring) provide observability for security operations.

Cloud Logging for security:

- **Audit logs** - automatically capture security-relevant events:
    - Admin Activity logs track administrative actions (free, always enabled, cannot disable).
    - Data Access logs track data reads/writes (not enabled by default, can be expensive).
    - System Event logs track GCP system events.
    - Policy Denied logs track when access denied due to security policy.
    - Access logs show who did what, when, from where, and result.
- **Log types for security**: VPC Flow Logs capturing network traffic for anomaly detection, Firewall Rules logs showing allowed/denied connections, Load Balancer logs tracking requests to applications, GKE audit logs for Kubernetes API activity, and Cloud SQL audit logs for database access.
- **Log analysis**: Logs Explorer provides advanced filtering and querying: filter by resource type, severity, time range, and specific fields. Example query finding failed authentication attempts: `protoPayload.authenticationInfo.principalEmail="*" AND protoPayload.status.code="7" AND resource.type="gce_instance"`.
- **Create log-based metrics** converting log entries to metrics enabling alerting: metric counts failed login attempts, alarm triggers on threshold, and integrates with Cloud Monitoring for notifications.
- **Log sinks and export**: Route logs to destinations for long-term storage and analysis: Cloud Storage for archival (encrypted, cheap long-term storage), BigQuery for SQL analysis and correlation, Pub/Sub for real-time processing and SIEM integration, and other GCP projects for centralization. Configure exclusion filters reducing costs by filtering out non-security-relevant logs.

Cloud Monitoring for security:

- **Security metrics** - monitor security-relevant signals: failed authentication attempts (sudden spike indicates brute force), unusual API calls (may indicate compromised credentials), resource usage anomalies (cryptocurrency mining), network traffic patterns (data exfiltration), and error rates (application attacks causing failures).
- **Alerting policies** - define conditions triggering alerts: metric threshold (CPU usage > 90%), log-based alerts (specific log patterns), uptime checks failing, and complex conditions with multiple metrics. Notification channels include email, SMS, Slack, PagerDuty, webhooks for automation.
- **Example security alerts**: Alert when new compute instances created in production (unauthorized resource creation), IAM policy changes in production projects, authentication failures exceed threshold, VPC firewall rules modified, and Cloud Storage bucket made public.

- **Dashboards** - visualize security posture: create dashboards showing security metrics over time, track trends identifying improving or degrading security, and use pre-built dashboards for common scenarios or create custom.

Security-specific monitoring workflows:

- **Unauthorized access detection**: Cloud Logging captures access denied events → log-based metric counts denials → Cloud Monitoring alert triggers on spike → notification to security team → investigation in Logs Explorer reviewing full context.
- **Anomaly detection**: Establish baseline of normal behavior (API call rates, resource creation patterns) → Cloud Monitoring detects deviation from baseline → alert on anomalous activity → automated response or investigation.
- **Compliance monitoring**: Track compliance metrics (encryption enabled on resources, MFA usage rates, password policy compliance) → dashboard shows compliance posture → alerts on compliance violations.

Integration with Security Command Center: Cloud Logging audit logs fed into Security Command Center for threat detection, Event Threat Detection analyzes logs identifying threats, findings appear in SCC with context from logs, and combined view of security findings and supporting log evidence.

SIEM integration: Export logs to external SIEM via Pub/Sub: configure log sink routing to Pub/Sub topic, SIEM subscribes to topic receiving logs in real-time, correlation rules in SIEM detect sophisticated attacks, and unified view of GCP and on-premises security.

Best practices: Enable audit logs for all services storing sensitive data, configure log sinks for long-term retention (7+ years for compliance), implement least privilege on logs (encrypt logs at rest, restrict access to security team), create alerts for high-priority security events, regularly review logs and alerts tuning for false positives, use labels and resource hierarchies organizing logs by project/environment, monitor log ingestion and export for gaps, and automate log analysis with BigQuery or Cloud Functions.

Cost optimization: Audit logs can be expensive at scale, exclude non-security-relevant logs from sinks, sample high-volume logs when full fidelity unnecessary, use lifecycle policies transitioning old logs to cheaper storage, and monitor logging costs against security value.

Example log-based metric for security:

```
gcloud logging metrics create failed_auth_attempts \
  --description="Count of failed authentication attempts" \
  --log-filter='protoPayload.status.code="7" AND resource.type="gce_instance"' \
  --value-extractor='EXTRACT(protoPayload.authenticationInfo.principalEmail)'
```

Then create alert:

```
gcloud alpha monitoring policies create --notification-channels=CHANNEL_ID --display
-name="High failed auth attempts" --condition-threshold-value=10 --condition-threshold
-duration=300s --condition
```

```
-filter='metric.type="logging.googleapis.com/user/failed_auth_attempts"'
```

Cloud Logging and Monitoring transform GCP from black box to transparent environment enabling proactive threat detection and incident response.

How do you enable VPC Service Controls in GCP, and why is it important?

VPC Service Controls creates security perimeters around GCP resources preventing data exfiltration.

What it does: Defines security perimeters restricting which GCP services can be accessed from where, prevents data exfiltration even with compromised credentials, enforces context-aware access based on client attributes (IP address, device security status), and protects against accidental or malicious data exposure.

Why it's important:

- **Data exfiltration prevention** - even if attacker compromises GCP credentials, VPC Service Controls prevents them from copying data to attacker-controlled project or external location.
- **Compliance** - many regulations require preventing unauthorized data transfer (HIPAA, GDPR, financial regulations), VPC Service Controls provides technical control demonstrating compliance.
- **Defense in depth** - complements IAM providing network-level protection even if IAM misconfigured.
- **Insider threat mitigation** - prevents malicious insiders from exfiltrating data to personal projects.

How it works: Service perimeter is virtual boundary around GCP resources (projects, VPCs), resources inside perimeter can communicate freely, communications crossing perimeter boundary are restricted by policy, requests from outside perimeter to protected resources are blocked, and requests from inside perimeter to outside are controlled (can be blocked or allowed with conditions).

Enabling VPC Service Controls:

- **Step 1: Enable Access Context Manager API**:

```
gcloud services enable accesscontextmanager.googleapis.com
```

- **Step 2: Create access policy** (organization-level container for perimeters):

```
gcloud access-context-manager policies create --organization=ORG_ID
--title="Production Security Policy"
```

- **Step 3: Define access levels** (optional, for conditional access): Access levels specify client attributes required for access like IP ranges, device policy requirements, or region.

```
gcloud access-context-manager levels create CorporateNetwork --policy=POLICY_ID
```

```
--basic-level-spec=corporate_ips.yaml
```

Where YAML specifies allowed IP ranges.

- **Step 4: Create service perimeter**:

  ```
  gcloud access-context-manager perimeters create ProductionPerimeter
  --policy=POLICY_ID --resources=projects/PROJECT_NUMBER --restricted
  -services=storage.googleapis.com,bigquery.googleapis.com --access
  -levels=accessPolicies/POLICY_ID/accessLevels/CorporateNetwork
  ```

  This creates perimeter protecting specified projects, restricting Cloud Storage and BigQuery access, requiring corporate network for access.

Perimeter configuration options:

- **Regular perimeter** - hard enforcement immediately, fully blocks unauthorized access.
- **Perimeter bridge** - allows communication between two perimeters, useful for shared services across security zones.
- **Dry run mode** - test perimeter without enforcement logging what would be blocked, analyze logs understanding impact, then enforce after validation.

Ingress and egress policies: Control traffic crossing perimeter boundary.

- **Ingress** - traffic entering perimeter from outside: define which external sources can access perimeter resources, specify identity and access level requirements.
- **Egress** - traffic leaving perimeter to outside: control which external services perimeter resources can access, prevent data exfiltration to unauthorized destinations.

Example egress policy allowing access only to specific external project:

```
egressPolicies:
- egressFrom:
    identities:
    - serviceAccount:[email protected]
  egressTo:
    resources:
    - projects/123456789  # Allowed destination project
    operations:
    - serviceName: storage.googleapis.com
      methodSelectors:
      - method: google.storage.objects.create
```

Protected services: VPC Service Controls supports many GCP services: Cloud Storage, BigQuery, Cloud SQL, Bigtable, Pub/Sub, Cloud Functions, Cloud Run, Secret Manager, and AI Platform. Comprehensive list: https://cloud.google.com/vpc-service-controls/docs/supported-products.

Access levels for conditional access: IP-based: allow only from corporate IP ranges, Device policy: require device meets security standards (managed, encrypted, updated), Geographic: allow only from specific regions, and Combine conditions with AND/OR logic.

Monitoring and troubleshooting: VPC Service Controls logs all boundary violations to Cloud Logging: filter for `protoPayload.metadata.vpcServiceControlsUniqueId`, logs show source, destination, and reason for denial, and analyze logs identifying legitimate traffic needing policy adjustment. Cloud Monitoring alerts on perimeter violations: sudden spike indicates attack or misconfiguration.

Common use cases:

- **Healthcare PHI protection** - create perimeter around projects storing patient data, restrict Cloud Storage and BigQuery access to perimeter only, require access from approved networks with device compliance, and prevent PHI export to unauthorized locations.
- **Financial data isolation** - separate perimeters for development and production, production perimeter blocks access from dev environments, egress policies prevent production data copying to dev projects.
- **Multi-region data residency** - create geographic perimeters enforcing data sovereignty, EU customer data stays in EU perimeter, US data in US perimeter, prevent cross-region data transfer.

Best practices: Start with dry run mode understanding impact before enforcement, monitor logs during dry run adjusting policies, use separate perimeters for different security zones (dev, staging, prod), implement least privilege in ingress/egress policies, combine with IAM for defense in depth, regularly review perimeter membership ensuring appropriate projects included, document exceptions and business justifications, test emergency access procedures (break-glass scenarios), and integrate with Security Command Center for compliance monitoring.

Limitations: Some GCP services not yet supported, policy changes can take time to propagate, overly restrictive policies can break legitimate workflows, and careful planning needed to avoid operational disruption.

VPC Service Controls is powerful control for high-security GCP environments preventing data exfiltration that IAM alone cannot stop. Essential for compliance in regulated industries.

Explain the concept of Identity-Aware Proxy (IAP) in GCP.

Identity-Aware Proxy (IAP) is GCP's zero-trust access control service enabling identity and context-aware application access without VPN.

- **Core concept**: Instead of network perimeter security (VPN), IAP authenticates and authorizes each request based on user identity and context, sits between users and applications verifying identity before granting access, works at application layer (HTTP/HTTPS) not network layer, and eliminates need for bastion hosts or VPNs for application access.
- **How it works**: User requests application URL (https://app.example.com) → request goes to Google Front End (GFE) → IAP checks if user authenticated, if not, redirects to Google/SAML identity provider for authentication → user authenticates providing credentials → IAP receives identity token → IAP checks authorization (does user have iap.httpsResourceAccessor role for this resource?) → if authorized, IAP proxies request to backend adding signed headers with user identity → backend receives request with verified identity information → backend can make access decisions based on user identity.
- **Key benefits**:
  - **Zero-trust security** - no implicit trust based on network, every request authenticated and authorized regardless of source, protects against lateral movement after perimeter breach.
  - **No VPN required** - employees access internal applications from any location without VPN, simplifies remote work, reduces VPN infrastructure costs.
  - **Centralized access control** - manage application access through IAM not application-specific configs, consistent access control across all applications, easy to grant/revoke access.
  - **User context** - applications receive verified user identity enabling user-specific authorization, audit trails show which user performed which action.

Enabling IAP:

- **Step 1: Configure OAuth consent screen** - GCP Console → APIs & Services → OAuth consent screen, configure app name, support email, authorized domains.
- **Step 2: Enable IAP for resource** - for App Engine/Cloud Run: enable IAP in console or via gcloud, for Compute Engine/GKE behind load balancer: configure backend service with IAP.

  ```
  gcloud compute backend-services update BACKEND_SERVICE --iap=enabled --global
  ```

- **Step 3: Configure IAM permissions** - grant `roles/iap.httpsResourceAccessor` to users/groups who should access application:

  ```
  gcloud projects add-iam-policy-binding PROJECT_ID --member="user:[email protected]"
  --role="roles/iap.httpsResourceAccessor" --condition=None
  ```

  Can scope to specific backend services for granular control.

- **Step 4: Application receives identity** - IAP adds headers to requests: `X-Goog-Authenticated-User-Email` contains user email, `X-Goog-Authenticated-User-ID` contains unique user ID, and `X-Goog-IAP-JWT-Assertion` contains signed JWT with claims. Application validates JWT ensuring request actually came through IAP (prevents bypass).

JWT validation in application:

```
from google.auth.transport import requests
from google.oauth2 import id_token

def validate_iap_jwt(iap_jwt, expected_audience):
    try:
        decoded_jwt = id_token.verify_token(
            iap_jwt,
            requests.Request(),
            audience=expected_audience,
            certs_url='https://www.gstatic.com/iap/verify/public_key'
        )
        return decoded_jwt
    except Exception as e:
        return None

# In request handler:
iap_jwt = request.headers.get('X-Goog-IAP-JWT-Assertion')
expected_audience = '/projects/PROJECT_NUMBER/apps/PROJECT_ID'
decoded_jwt = validate_iap_jwt(iap_jwt, expected_audience)
if decoded_jwt:
    user_email = decoded_jwt['email']
    # User is authenticated via IAP
```

Access levels and conditional access: Combine IAP with Access Context Manager for conditional access: require corporate IP range, managed device compliance, geographic restrictions, time-based access. Create access level defining conditions, apply to IAP via access policy binding.

Programmatic access (service-to-service): IAP supports service accounts for automated access: service account authenticates using OAuth 2.0, obtains IAP token, includes token in requests to IAP-protected resource. Useful for CI/CD, monitoring tools, or microservices.

Monitoring and logging: Cloud Logging captures IAP access logs showing authentication attempts, authorization decisions, accessed resources, user identities, and denial reasons. Cloud Monitoring alerts on authorization failures (spike indicates attack or misconfiguration).

Use cases:

- **Internal admin tools** - HR dashboard, internal analytics tools, admin panels accessible to employees without VPN.
- **Partner access** - provide external partners access to specific applications without full VPN access, granular control per partner organization.
- **Contractor access** - temporary access to applications for contractors, easily grant/revoke

through IAM, no need to manage separate accounts.

- **Multi-tenant SaaS** - use IAP for tenant isolation, each tenant organization gets access to only their resources.

Best practices: Always validate JWT in application (don't trust headers alone), use signed headers preventing header spoofing, implement least privilege IAM bindings (grant access only to specific users/groups), combine with Access Context Manager for enhanced security, enable Cloud Audit Logs tracking all access, use service accounts for programmatic access, not user credentials, test access thoroughly before full deployment, provide alternative access method for emergencies (break-glass), monitor for bypass attempts (direct backend access), and educate users on authentication flow.

Limitations: Only works for HTTP/HTTPS applications (no TCP/UDP), requires applications behind GCP load balancer or App Engine/Cloud Run, adds latency due to authentication checks (usually <100ms), and OAuth consent screen may confuse some users.

Comparison to VPN: VPN provides network access, IAP provides application access, VPN is all-or-nothing, IAP is granular per-application, VPN trusts network, IAP requires authentication per request, VPN requires client software, IAP works in browser.

IAP represents modern zero-trust approach to application access, more secure and user-friendly than traditional VPN for many use cases.

What is the purpose of Google Cloud Security Scanner?

Web Security Scanner is GCP's automated vulnerability scanning service for web applications.

Purpose: Automatically identifies common web vulnerabilities in App Engine, Compute Engine, and GKE applications: cross-site scripting (XSS), Flash injection, mixed content (HTTP resources on HTTPS pages), outdated or insecure libraries, and clear text passwords. Helps developers find security issues before attackers do, integrates into CI/CD for continuous security testing, and complements manual security testing.

How it works: Scanner crawls application starting from seed URLs, follows links discovering application structure, submits forms with test payloads, and analyzes responses detecting vulnerability indicators. Uses GoogleBot user-agent (customizable), respects robots.txt restrictions, and throttles requests preventing application disruption.

Scan types:

- **Managed scan** (easy setup) - point scanner at App Engine or Compute Engine application, configure authentication if needed, scanner automatically crawls and tests.
- **Custom scan** (advanced) - specify seed URLs manually, configure authentication (Google account, custom login), set crawl scope and exclusions, and adjust scan aggressiveness.

Authentication support: For applications requiring login: Google account authentication (OAuth), custom login (provide credentials), or IAP-protected applications. Scanner authenticates before scanning testing authenticated portions of application.

Vulnerability detection:

- **Cross-site scripting (XSS)** - reflected XSS (input reflected in response), stored XSS (malicious input stored and displayed), DOM-based XSS potential.
- **Mixed content** - HTTP resources loaded on HTTPS pages creating security warnings and downgrade attacks.
- **Outdated libraries** - detects known vulnerable JavaScript libraries (old jQuery, Angular versions).
- **Clear text passwords** - password fields without HTTPS.
- **Flash injection** - vulnerable Flash content.

Findings and remediation: Scan results show vulnerabilities with severity (High, Medium, Low), affected URLs and parameters, proof-of-concept demonstrating vulnerability, remediation guidance explaining how to fix, and CWE/OWASP references for context. Integrate findings with Security Command Center for centralized vulnerability management.

Limitations: Doesn't test for all vulnerability types (no SQL injection, authentication flaws, business logic issues), may generate false positives requiring validation, can't test stateful or complex multi-step workflows thoroughly, limited to HTTP/HTTPS applications, and shouldn't replace manual penetration testing.

Best practices: Run scans regularly (weekly or on every deployment), integrate with CI/CD failing builds on high-severity findings, test in staging before production scans, validate findings (automated scanners have false positives), combine with other security testing (SAST, DAST, manual pentesting), track remediation progress over time, and exclude sensitive endpoints from scanning if needed.

Integration with CI/CD:

```
# In Cloud Build pipeline
steps:
- name: 'gcr.io/cloud-builders/gcloud'
  args:
    - 'alpha'
    - 'web-security-scanner'
    - 'scans'
    - 'create'
    - '--starting-urls=https://staging.example.com'
    - '--max-qps=5'
- name: 'gcr.io/cloud-builders/gcloud'
  args:
    - 'alpha'
    - 'web-security-scanner'
    - 'scans'
    - 'list'
    - '--filter=scanRunState:FINISHED'
    - '--format=value(findingCount)'
  id: 'check-findings'
# Fail build if findings exceed threshold
```

Web Security Scanner provides baseline automated vulnerability testing, valuable for catching common issues but should be part of comprehensive security testing strategy including SAST, dependency scanning, and manual testing.

How can you secure data stored in Google Cloud Storage?

Securing Cloud Storage requires multiple layers of controls.

Access control:

- **IAM policies** - grant permissions at bucket or object level: `roles/storage.objectViewer` for read access, `roles/storage.objectAdmin` for full object control, and grant to specific users, groups, or service accounts following least privilege.
- **Uniform bucket-level access** - recommended over ACLs: simplifies permission management using only IAM, disables object ACLs and bucket ACLs, enforced with `gsutil uniformbucketlevelaccess set on gs://BUCKET`.
- **Access Control Lists (ACLs)** - legacy, finer-grained control per object: useful for specific use cases (public website content), generally prefer IAM for easier management.
- **Signed URLs and signed policy documents** - provide time-limited access without authentication: generate signed URL for specific object with expiration, user can access via URL without GCP credentials, useful for temporary sharing or client uploads.

Encryption:

- **Encryption at rest** (default) - all data encrypted automatically: Google-managed encryption keys (default, no configuration needed), customer-managed encryption keys (CMEK) with Cloud KMS for key control, or customer-supplied encryption keys (CSEK) where you provide keys per request. Enable CMEK: `gsutil kms encryption -k projects/PROJECT/locations/LOCATION/keyRings/RING/cryptoKeys/KEY gs://BUCKET`.
- **Encryption in transit** - HTTPS enforced for API access, use bucket policy requiring TLS.

Object Lifecycle Management: Automatically transition objects to cheaper storage classes or delete: move old objects to Nearline/Coldline/Archive reducing costs, delete temporary data after retention period, comply with data retention policies.

Versioning: Enable object versioning preventing accidental deletion/overwrite: `gsutil versioning set on gs://BUCKET`. Deleted objects retained as noncurrent versions, restore previous versions if needed, combine with lifecycle rules managing version retention.

Object retention and holds:

- **Bucket Lock** (retention policy lock) - immutable retention preventing deletion: set retention period (e.g., 7 years for compliance), lock policy making it permanent, and objects cannot be deleted until retention expires even by bucket owner.
- **Holds** - temporary locks on objects: event-based holds for legal/compliance reasons, temporary holds for ongoing investigations, release holds when appropriate.

Audit logging: Enable Cloud Audit Logs tracking bucket and object access: Data Access logs capture who accessed which objects when (not enabled by default, can be expensive), Admin Activity logs track configuration changes, analyze logs for unauthorized access or suspicious patterns.

Public access prevention:

- **Block public access** - organization policy preventing public exposure:

  ```
  gcloud org-policies set-policy public_access_prevention.yaml` where policy denies
  `allUsers` and `allAuthenticatedUsers
  ```

- **Bucket-level controls** - use IAM conditions preventing public access grants.

DLP integration: Cloud Data Loss Prevention scans buckets for sensitive data: PII, credit cards, API keys, or custom patterns. Creates findings showing what sensitive data exists where, helps classify data for appropriate protection.

VPC Service Controls: Place buckets inside service perimeter preventing data exfiltration: even with stolen credentials, data can't be copied outside perimeter.

Best practices: Enable uniform bucket-level access simplifying permission management, use customer-managed encryption keys for sensitive data, enable versioning on buckets with important data, implement lifecycle policies for data retention, enable audit logging for security monitoring, block public access at organization level, regularly scan for sensitive data with DLP, use signed URLs for temporary external access, apply least privilege IAM policies, combine multiple controls for defense in depth, monitor bucket access patterns for anomalies, test disaster recovery from backups, and document data classification and protection requirements.

Example secure bucket configuration:

```
# Create bucket with security controls
gsutil mb -l us-central1 -b on gs://secure-data-bucket

# Enable uniform bucket-level access
gsutil uniformbucketlevelaccess set on gs://secure-data-bucket

# Enable versioning
gsutil versioning set on gs://secure-data-bucket

# Set CMEK encryption
gsutil kms encryption \
  -k projects/PROJECT/locations/us-central1/keyRings/ring/cryptoKeys/key \
  gs://secure-data-bucket

# Set retention policy (30 days)
gsutil retention set 30d gs://secure-data-bucket

# Lock retention policy (careful - permanent!)
# gsutil retention lock gs://secure-data-bucket

# Set lifecycle rule
```

```
gsutil lifecycle set lifecycle.json gs://secure-data-bucket
```

Where `lifecycle.json` might transition old objects:

```
{
  "lifecycle": {
    "rule": [
      {
        "action": {"type": "SetStorageClass", "storageClass": "NEARLINE"},
        "condition": {"age": 30}
      },
      {
        "action": {"type": "Delete"},
        "condition": {"age": 365, "isLive": false}
      }
    ]
  }
}
```

This comprehensive approach protects Cloud Storage data from unauthorized access, accidental deletion, and exfiltration.

What measures would you put in place to ensure its security?

I'd implement comprehensive security across all GKE layers.

Cluster architecture: Create **private GKE cluster** with nodes in private subnets having only private IPs (`--enable-private-nodes`), master endpoint accessible only from authorized networks (`--enable-master-authorized-networks --master-authorized-networks=CORP_CIDR`), and use Workload Identity eliminating service account keys (`--workload-pool=PROJECT.svc.id.goog`). Enable **Shielded GKE Nodes** for secure boot and integrity monitoring (`--enable-shielded-nodes`) protecting against rootkits.

Network security: Implement **Network Policies** with Calico for pod-to-pod micro-segmentation, default deny all traffic then allow specific pod communications, separate namespaces by security zone (frontend, backend, data), and restrict egress to only required external services. Enable **Private Google Access** for nodes to reach GCP APIs without public IPs. Configure **Cloud Armor** on load balancers protecting ingress with WAF rules and DDoS protection. Use **GKE Dataplane V2** for improved network security and observability.

Access control: Implement strict **RBAC** creating service accounts per application with minimal permissions, RoleBindings scoped to namespaces not cluster-wide, avoiding cluster-admin except for administrators, and regular RBAC audits removing unused permissions. Integrate **GCP IAM** controlling who can get cluster credentials (`container.clusters.get`) separately from Kubernetes RBAC controlling in-cluster actions. Enable **Binary Authorization** preventing deployment of unsigned or vulnerable images, require images signed by trusted CI/CD pipeline, integrate with Container Analysis for vulnerability checks before deployment.

Container security: Use **Container-Optimized OS** as node OS providing minimal attack surface and automatic updates. Enable **Container Scanning** in Artifact Registry automatically scanning pushed images for CVEs. Implement **Pod Security Standards** enforcing containers run as non-root, drop unnecessary capabilities, use read-only root filesystem, and disable privilege escalation.

Secrets management: Store secrets in **Kubernetes Secrets** with etcd encryption enabled, use **Secret Manager** for sensitive secrets accessed via Workload Identity, never hardcode secrets in container images or ConfigMaps, and enable automatic secret rotation where possible.

Monitoring and logging: Enable **GKE Audit Logging** capturing all API server requests, **Cloud Logging** collecting container logs, node logs, and cluster events, configure **Cloud Monitoring** with alerts on suspicious activity (unauthorized API calls, unusual resource usage), and integrate with **Security Command Center** for centralized security visibility.

Vulnerability management: Enable **automatic node upgrades** (`--enable-autoupgrade`) ensuring latest security patches, **automatic node repair** (`--enable-autorepair`) replacing unhealthy nodes, scan clusters against **CIS Kubernetes Benchmark** remediating findings, and regularly update application dependencies patching CVEs.

Compliance: Enable **GKE Sandbox** (gVisor) for running untrusted workloads with additional isolation, implement **Pod Security Policies** or **Pod Security Standards** enforcing security baselines, maintain **audit trails** meeting compliance requirements, and regular security assessments and penetration testing.

Example secure cluster creation:

```
gcloud container clusters create production-cluster \
  --zone us-central1-a \
  --enable-private-nodes \
  --enable-private-endpoint \
  --master-ipv4-cidr 172.16.0.0/28 \
  --enable-ip-alias \
  --network=prod-vpc \
  --subnetwork=gke-subnet \
  --enable-master-authorized-networks \
  --master-authorized-networks=CORPORATE_CIDR \
  --enable-shielded-nodes \
  --shielded-secure-boot \
  --shielded-integrity-monitoring \
  --workload-pool=PROJECT_ID.svc.id.goog \
  --enable-binauthz \
  --enable-autorepair \
  --enable-autoupgrade \
  --enable-stackdriver-kubernetes \
  --logging=SYSTEM,WORKLOAD \
  --monitoring=SYSTEM,WORKLOAD \
  --addons=HorizontalPodAutoscaling,HttpLoadBalancing,NetworkPolicy \
  --enable-network-policy \
  --enable-intra-node-visibility \
  --maintenance-window-start=2026-01-20T03:00:00Z \
  --maintenance-window-duration=4h \
  --release-channel=regular \
  --image-type=COS_CONTAINERD \
  --machine-type=n2-standard-4 \
  --disk-type=pd-ssd \
  --disk-size=100 \
  --enable-autoscaling \
  --min-nodes=3 \
  --max-nodes=10
```

This creates production-grade secure GKE cluster with defense in depth protecting against common Kubernetes attack vectors.