

Security Monitoring in the Cloud
Session #2

Container Security & Observability (Docker + Kubernetes on GCP)

Jan 2026

Key Outcomes

- Assess and mitigate risks in container images.
- Monitor and analyze Kubernetes audit logs.
- Detect pod-level privilege escalation and misconfigured RBAC.
- Understand observability in GKE environments.

Agenda

01

Understand containers risks:

Image & Build Stage, Runtime Security

02

Understand Kubernetes risks:

K8s architecture, network security, RBAC misconfiguration,

03

Containers and K8s monitoring:

Container and Node-Level, K8s and GKE, Network monitoring

04

Attack Scenarios & Detection:

Pod attack, supply chain attack, RBAC attack

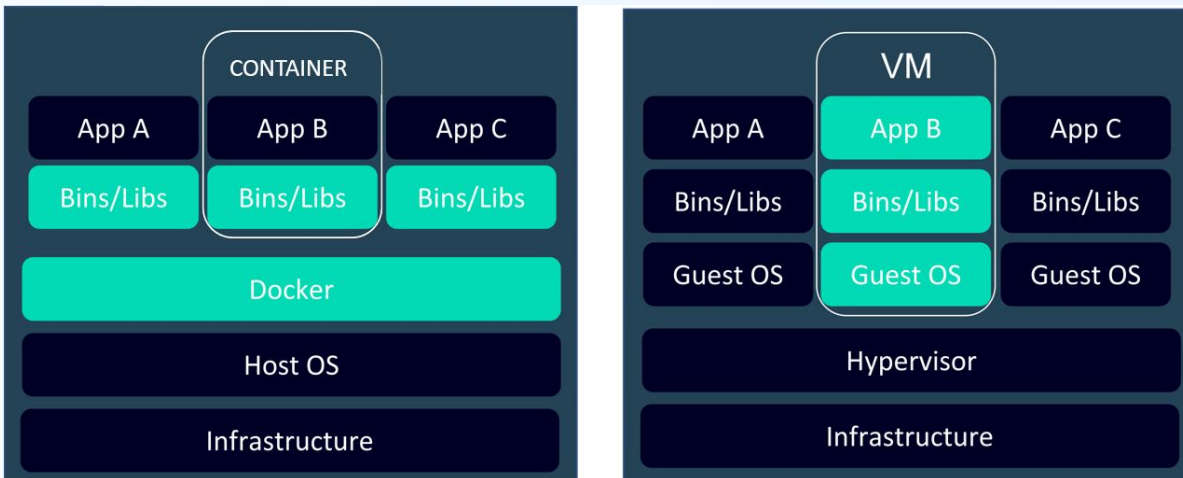
1. Containers



Containers Security Risks

1.0.0

Containers refresher



A container is:

- A process running in a restricted environment but no no hypervisor isolation
- With its own filesystem layer but shares the host kernel with other containers
- Using Linux kernel features:
 - Namespaces (process, network, mount isolation)
 - cgroups (resource limits)

Container-inherent risks:



- A kernel vulnerability affects all containers on that host
- Misconfigured containers can give host-level access
- Attackers can pivot from one pod/container to another



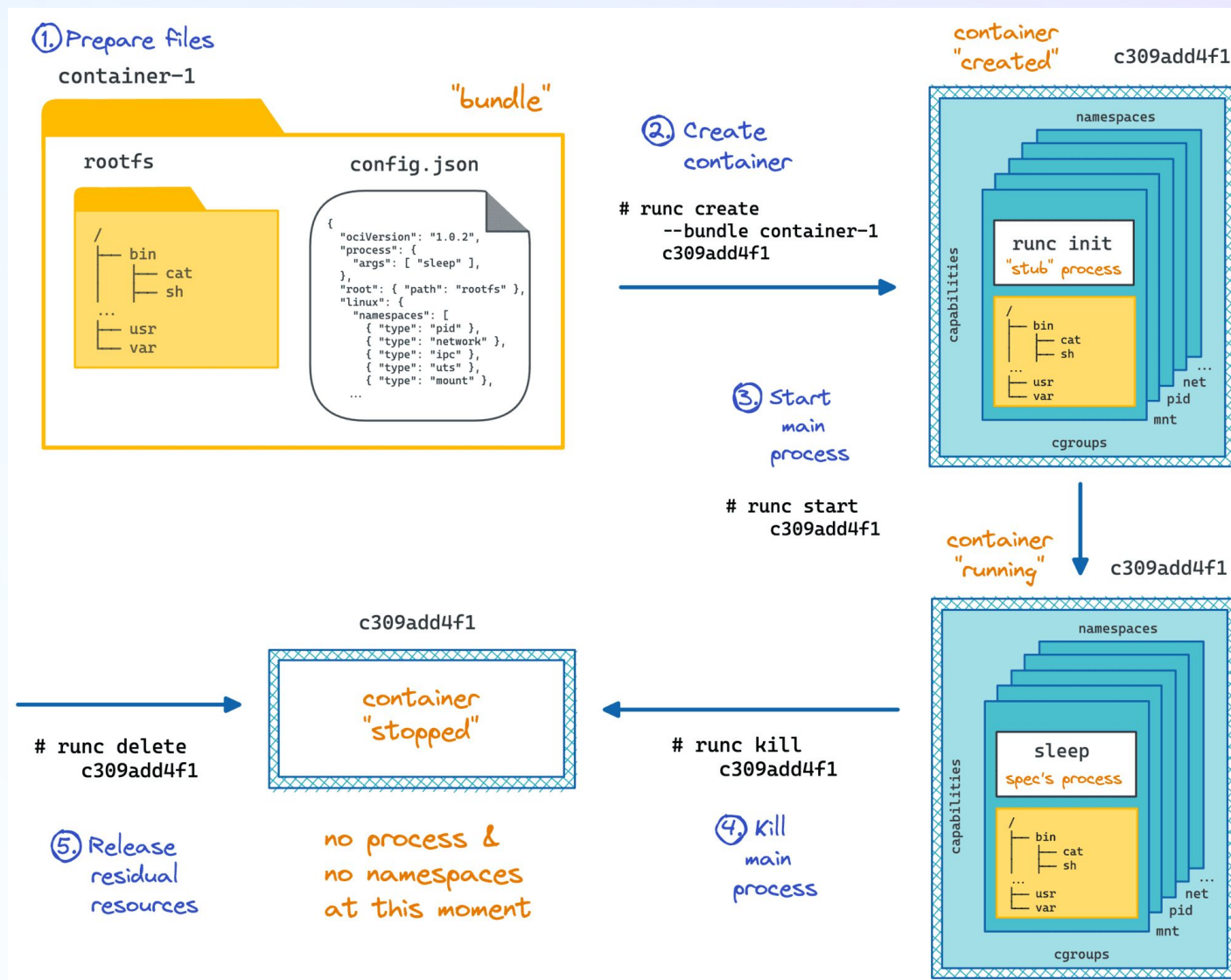
Usage model risks:

- Ephemeral → vanish quickly, making forensic evidence fragile
- Portable → vulnerabilities travel with images
- Highly Automated → fast propagation + Reuse Amplifies Impact + CI/CD servers are overly permissive

Containers Security Risks

1.0.1

Containers Lifecycle



Containers Security Risks

1.1.0

Image & Build Stage



Image Vulnerabilities

- Outdated OS libraries
- Vulnerable dependencies (Python, Node, Go modules)
- Misconfigured build artifacts (compilers, package managers, build metadata)
- Tools/binaries unnecessary for production (curl, bash)



Secrets in Images

Common mistakes include embedding:

- API keys in Dockerfile ENV
- Plaintext secrets in /app/config.json
- SSH keys in the image layer history



Supply Chain Risks

Developers unknowingly introduce malicious code via:

- Public base images (Docker Hub)
- Compromised dependencies
- CI/CD credential misuse



Security Monitoring should detect:

- Vulnerability scans with critical CVEs
- Unsigned or unverified images
- Deployment of unscanned images to GKE

Containers Security Risks

Runtime Security Risks

1.2.0

Common Misconfigurations:

- Privileged Containers
 - `securityContext: privileged: true`
=disable seccomp + relax SELinux/AppArmor
 - Access host devices
 - Modify kernel parameters
 - Load kernel modules
 - Escape to the host
- Dangerous Linux Capabilities
 - NET_ADMIN → modify network interfaces
 - SYS_PTRACE → inspect other processes
 - SYS_ADMIN → near-root privilege on host
- HostPath Mounts -> gives direct access to the host filesystem



Security monitoring should detect:

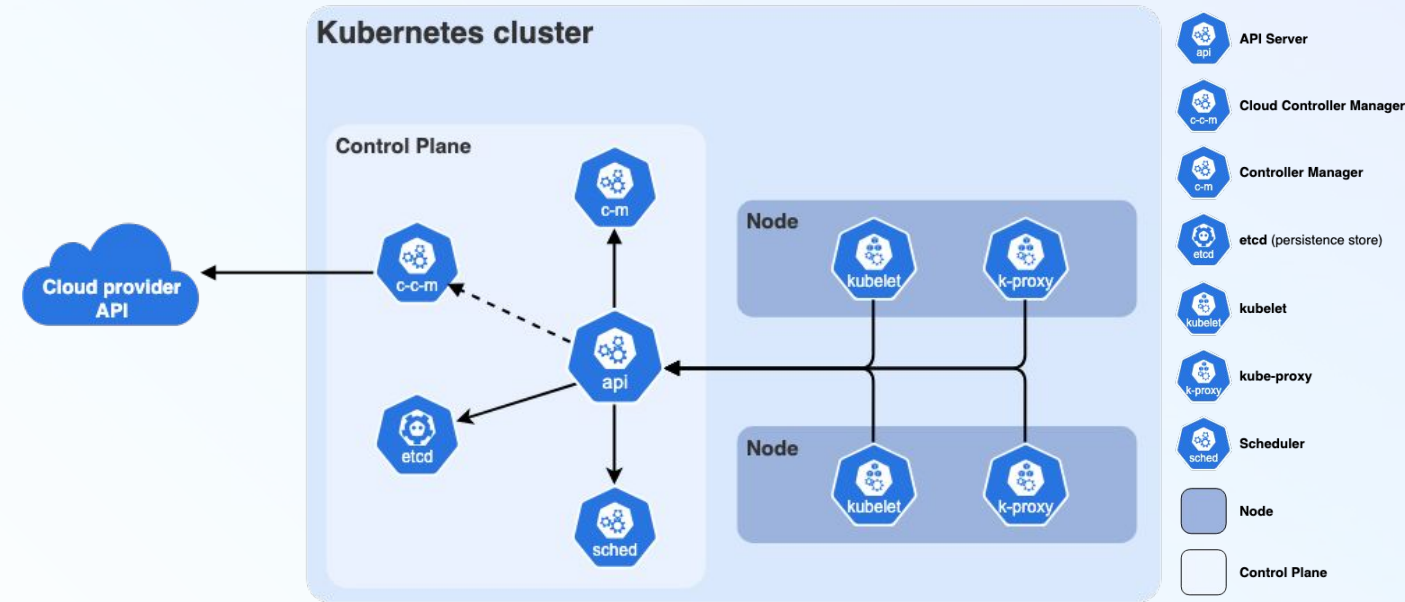
- Unexpected processes running inside containers
- Execution of tools like bash, sh, curl, wget, nmap
- Elevated capabilities (`mount / umount = CAP_SYS_ADMIN, ip link = CAP_NET_ADMIN, etc.`)
- Outbound connections to suspicious IPs
- Container restarts/crashes (DoS or exploitation attempts)

2. Kubernetes

Kubernetes Security Risks

2.0.0

Kubernetes refresher



Security capabilities:

- Namespace:
 - Logical grouping of workloads
 - But not a security boundary; RBAC must enforce isolation
- Security Contexts(SYS_ADMIN, SYS_NET, etc)
- Service Accounts:
 - Identities used by pods
 - Often over-permissioned
 - Mapped to GCP IAM using Workload Identity
- RBAC:
 - Defines who can:
 - create pods
 - exec into containers
 - modify roles
 - access secrets

Inherited Security implications:

- API server → single point of failure
- etcd → holds secrets → must be encrypted
- Controller or scheduler compromise → cluster-wide takeover

Kubernetes Security Risks

Manifests



```
metadata:
  name: bad-ns
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: bad-sa
  namespace: bad-ns
---
# Danger: binds the service account to cluster-admin
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: bad-sa-cluster-admin
subjects:
- kind: ServiceAccount
  name: bad-sa
  namespace: bad-ns
roleRef:
  kind: ClusterRole
  name: cluster-admin # <-- API admin
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: v1
kind: Pod
metadata:
  name: dangerous-pod
  namespace: bad-ns
spec:
  serviceAccountName: bad-sa
  containers:
  - name: toolbox
    image: busybox
    command: ["sh", "-c", "sleep 3600"]
    securityContext:
      privileged: false
      capabilities:
        add: ["SYS_ADMIN"] # <-- Node admin
```

2.0.1



```
metadata:
  namespace: good-ns
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list"] # <-- No deployments, nodes or secrets
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: bind-app-sa-pod-reader
  namespace: good-ns
subjects:
- kind: ServiceAccount
  name: app-sa
  namespace: good-ns
roleRef:
  kind: Role
  name: pod-reader # <-- API read
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: v1
kind: Pod
metadata:
  name: minimal-app
  namespace: good-ns
spec:
  serviceAccountName: app-sa
  containers:
  - name: app
    image: busybox
    command: ["sh", "-c", "sleep 3600"]
    securityContext:
      runAsNonRoot: true
      runAsUser: 1000
      allowPrivilegeEscalation: false # <-- no sudo
      capabilities:
        drop: ["ALL"] # <-- drop all capabilities
```

Kubernetes Security Risks

2.1.0

Common Threats



RBAC Misconfigurations

- cluster-admin granted to developers
- Service accounts with wildcard permissions
- Ability to create pods/exec or modify roles



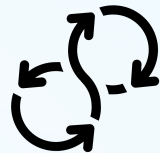
Pod Privilege Escalation

- Risky permissions: privileged: true, hostPID: true, hostNetwork: true, hostPath mounting root filesystem, SYS_ADMIN capability



Pod Escape / Runtime Attacks

- Break kernel isolation via vulnerabilities (Dirty COW, RunC bug)
- Attach to Docker socket (/var/run/docker.sock or containerd.sock) in CI/CD
- Load kernel modules
- Access host file system



Supply Chain Attacks

- Malicious or compromised container images
- Tampered base images
- Compromised CI/CD pipeline
- Dependency poisoning



Lateral Movement

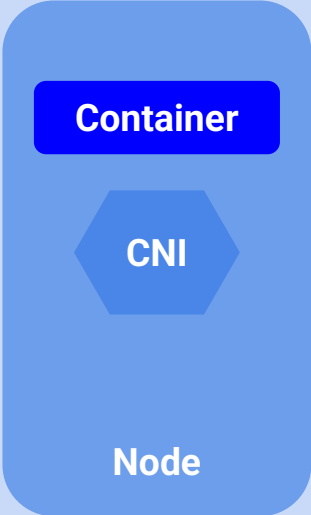

- kubectl exec into another pod
- Abuse service account token to call Kubernetes API
- Pivot into databases, internal microservices, Cloud APIs, etc

3. Monitoring

Monitoring for Containers and Kubernetes

3.0.0

Monitoring Layers

K8S Cluster	What	Why	Source
 <p>Container</p> <p>CNI</p> <p>Node</p>	stdout/stderr, process activity, restarts	Detect runtime abuse, reverse shells, miners	GKE Container Logs, runtime tools (Cloud Logging, Datadog, etc.)
	Pod-to-pod, pod-to-external flows	Detect scanning, C2, exfiltration	VPC Flow Logs, GKE Dataplane logs (kubelet, container runtime, node OS, CNI)
	CPU/mem/network, kernel events	Detect mining, DoS, escape attempts	Cloud Logging, Agent for node logs (Kernel-level activity)
 <p>K8S API</p> <p>CCM</p> <p>Control Plane</p>	kubectl exec, RoleBindings, pod creations, secret access	Detect admin abuse, RBAC escalation, pod compromise	Kubernetes Audit Logs
	IAM, GKE API calls	Detect cluster creation, config changes, key creation	Cloud Audit Logs (gke.googleapis.com, container API)

Monitoring for Containers and Kubernetes

3.0.1

Telemetry Sources

Telemetry Source	Security Insights	Example Usage
K8s Audit Logs	All API requests, RBAC changes, pod exec	Detect RBAC abuse, secret access
Container Logs	App activity, process output	Detect reverse shells, mining output
Node Logs	Kernel and system activity	Detect container escape attempts
Runtime Sensors (DD/SCC)	Syscall-level events	Detect malicious processes inside containers
VPC Flow Logs	Network traffic meta	Detect scanning, lateral movement, exfil
Dataplane v2 Logs	Pod-to-pod traffic	Detect internal reconnaissance
Cloud Audit Logs	Cluster creation, IAM events	Detect out-of-band admin actions
Artifact Analysis	Vulnerability scanning results	Detect malicious or outdated images
Cloud Monitoring Metrics	CPU, Mem, Restarts	Detect mining or DoS behavior
Cloud DNS Logs	Domain queries from pods	Detect C2 / DNS tunneling

Monitoring for Containers and Kubernetes

3.1.0

[minimum] Logs for Containers (GKE)

Application Layer

Contain:

- Application logs (INFO/ERROR/etc.)
- Stack traces
- Authentication events (for in-app auth)

Useful for:

- Application-level attacks
- SQL injection traces
- Abuse of business logic

Example Logs Explorer Query:

```
resource.type="k8s_container"  
resource.labels.pod_name="web-frontend-123"
```

Node Layer

Contain:

- OOM kills (out of memory)
- Kernel warnings
- System daemon logs

Useful for:

- Crashes due to exploitation attempts
- Kernel anomalies on nodes
- Frequent restarts (crash-looping pods)

Example Logs Explorer Query:

```
resource.type="k8s_cluster"  
jsonPayload.reason="BackOff"
```

Monitoring for Containers and Kubernetes

3.2.0

Kubernetes Audit Logs (Control Plane)

- The single source of truth for:
 - User actions (kubectl, CI/CD pipelines)
 - RBAC changes (RoleBindings, ClusterRoleBindings)
 - Pod exec / attach / port-forward events
 - Secret access
 - Workload creation/modification

Key events to watch

- create / patch / update (Roles, Pods, secrets, Service Accounts)
- get / list on Secrets
- exec on Pods (via kubectl exec)
- portforward, attach (remote access to pods)

Example — Detect kubectl exec into a Pod:

- Audit events for kubectl exec usually have:
- verb: "create"
- objectRef.resource: "pods"
- subresource: "exec"

```
resource.type="k8s_cluster"  
protoPayload.@type="type.googleapis.com/google  
.cloud.audit.AuditLog"  
protoPayload.objectRef.resource="pods"  
protoPayload.objectRef.subresource="exec"
```

4. Common Security Threats

Common Security Threats

4.0.0

Unauthorized Kubectl Exec Into a Pod

Attack Path:

Attacker gains credentials (developer account or service account token) and gains interactive shell inside pod → installs tools → pivots internally..

Detection:

Exec into production namespace / Pod generating unusual container logs after exec.

MITRE: TA0002 - Execution (T1609 - Container Administration Command, T1059 - Command Execution)

GCP Logs Explorer

```
resource.type="k8s_cluster"  
protoPayload.@type="type.googleapis.com/google.cloud.audit.AuditLog"  
protoPayload.methodName=~"exec"
```

SCC Events examples

[Suspicious Exec or Attach to a System Pod](#)

Common Security Threats

4.2.0

RBAC Privilege Escalation

Attack Path:

Attacker compromises a service account or developer user, creates a new ClusterRoleBinding, gains full cluster control or deploys malicious pods, steals secrets, or pivots to cloud APIs

Detection:

Creation of RoleBinding or ClusterRoleBinding or RBAC changes in production namespaces
Privileged roles assigned to unusual identities

MITRE: T1098 - Account Manipulation, T1068 - Privilege Escalation

GCP Logs Explorer

```
resource.type="k8s_cluster"
resource.labels.cluster_name="lab-test-cluster"
protoPayload.@type="type.googleapis.com/google.cloud.audit.AuditLog"
(
  protoPayload.objectRef.resource="clusterrolebindings"
  OR protoPayload.objectRef.resource="rolebindings"
  OR protoPayload.request.objectRef.resource="clusterrolebindings"
  OR protoPayload.request.objectRef.resource="rolebindings"
)
protoPayload.methodName=~"create"
protoPayload.responseStatus.code=0
```

SCC Events examples

[Changes to sensitive
Kubernetes RBAC objects
ClusterRoleBinding to
Privileged Role](#)

Common Security Threats

4.3.0

Privileged Pod or HostFilesystem Mount

Attack Path:

Attacker creates a malicious privileged pod, accesses host filesystem, extracts credentials, takes over node or uses node as pivot to entire cluster or cloud API

Detection:

Overly privileged pod created (privileged: true, capabilities.add=*, hostPath mount, etc)

Node filesystem writes from pod or unusual process in pod.

MITRE: T1611 - Escape to Host, T1203 - Exploitation for Privilege Escalation

GCP Logs Explorer

```
resource.type="k8s_cluster"  
resource.labels.cluster_name="lab-test-cluster"  
protoPayload.methodName="io.k8s.core.v1.pods.create"  
protoPayload.request.spec =~ "privileged.*true"
```

SCC Events examples

[Launch of privileged
Kubernetes container](#)

Common Security Threats

Malicious or Vulnerable Container Images

4.4.0

Attack Path:

Developer pulls and deploys a vulnerable container image.

Detection:

High CPU usage, node crashes or OOM events

Suspicious outbound domains

Execution of unexpected binaries

MITRE: T1204 – User Execution

GCP Logs Explorer

```
resource.type="k8s_container"
resource.labels.cluster_name="lab-test-cluster"
(
  textPayload=~"bash -i" OR textPayload=~"nc -" OR
  textPayload=~"curl http://" OR
  textPayload=~"wget http://"
)
```

SCC Events examples

[Suspicious Kubernetes
Container Names](#)

Common Security Threats

4.5.0

Compromised Pod

Attack Path:

Pod compromised via app vulnerability. The attacker uses pod SA token to call Kubernetes API and pivots to cloud resources

Detection:

Pod making outbound calls to metadata server

Pod reaching many internal IPs on many ports

Pod contacting high-risk external IPs

MITRE: T1552 - Credentials in Files / Metadata Access

GCP Logs Explorer

```
resource.type="k8s_container"  
resource.labels.cluster_name="lab-test-cluster"  
(  
  textPayload =~ "metadata\\.google\\.\\.internal"  
  OR textPayload =~ "169\\.\\.254\\.\\.169\\.\\.254"  
  OR textPayload =~ "computeMetadata/v1"  
)
```

SCC Events examples

[Local Reconnaissance Tool Execution](#)

Question ?

The background of the slide is composed of several large, overlapping triangles in various shades of purple, blue, and magenta. The triangles are arranged in a way that creates a sense of depth and geometric complexity. The colors range from deep indigo to bright magenta, with some areas appearing darker due to the overlap.

Additional Resources

1. [Kubernetes Security](#)
2. [GKE CIS Benchmark](#)

Hands-on time after a 10min break !



Detect reconnaissance activity within GKE

https://github.com/0x74696D/security_monitoring_tp