Kubernetes Logging

Table of Contents

What are the different types of logs in a Kubernetes cluster, and what security-relevant information does each provide?

Kubernetes generates multiple log streams providing comprehensive visibility into cluster operations, security events, and application behavior. Understanding these log types is essential for effective security monitoring.

Kubernetes log categories:

1. Control Plane Logs (most security-critical):

kube-apiserver logs:

- **What it logs** - all API requests to the cluster, authentication and authorization events, admission controller decisions, resource creation/modification/deletion, API access patterns.
- **Security value** - who accessed what resources and when, unauthorized access attempts, privilege escalation attempts, suspicious API usage patterns, compliance audit trail.

Example log entry:

```
{
  "kind": "Event",
  "apiVersion": "audit.k8s.io/v1",
  "level": "Metadata",
  "auditID": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "stage": "ResponseComplete",
```

```
  "requestURI": "/api/v1/namespaces/production/secrets",
  "verb": "list",
  "user": {
    "username": "alice@example.com",
    "groups": ["developers", "system:authenticated"]
  },
  "sourceIPs": ["203.0.113.45"],
  "userAgent": "kubectl/v1.28.0",
  "objectRef": {
    "resource": "secrets",
    "namespace": "production",
    "apiVersion": "v1"
  },
  "responseStatus": {
    "code": 200
  },
  "requestReceivedTimestamp": "2024-01-20T10:30:00.123456Z",
  "stageTimestamp": "2024-01-20T10:30:00.234567Z"
}
```

Security events to monitor:

- Secret access (ConfigMaps, Secrets)
- Role/RoleBinding modifications
- Exec into pods
- Port-forward connections
- Privileged pod creation
- ServiceAccount token creation
- Admission webhook denials

kube-controller-manager logs:

- **What it logs** - controller operations (ReplicaSet, Deployment reconciliation), garbage collection events, node lifecycle events, persistent volume operations.
- **Security value** - unauthorized controller operations, resource quota violations, suspicious resource creation patterns, node compromise indicators.

kube-scheduler logs:

- **What it logs** - pod scheduling decisions, node affinity/anti-affinity, resource allocation, scheduling failures.
- **Security value** - unusual scheduling patterns, attempts to schedule on specific nodes, resource exhaustion attacks.

etcd logs:

- **What it logs** - distributed key-value store operations, cluster state changes, backup/restore

operations.

- **Security value** - direct etcd access (should be none except kube-apiserver), data corruption attempts, unauthorized state modifications.

cloud-controller-manager logs (cloud-specific):

- **What it logs** - cloud provider interactions, load balancer provisioning, node lifecycle in cloud, persistent volume provisioning.
- **Security value** - unauthorized cloud resource creation, suspicious load balancer configurations.

2. Node-level logs:

kubelet logs:

- **What it logs** - pod lifecycle on the node (starting, stopping), image pulls, container runtime interactions, volume mounts, health checks.
- **Security value** - privileged container creation, hostPath volume usage, suspicious image pulls, failed pod starts, resource exhaustion on node.

Example kubelet log:

```
I0120 10:30:00.123456 kubelet.go:1234] Creating pod: production/webapp-abc123
W0120 10:30:01.234567 kubelet.go:5678] Pod production/webapp-abc123 attempted to mount
hostPath /etc, blocked by admission
E0120 10:30:02.345678 kubelet.go:9012] Failed to pull image
"malicious.registry.com/backdoor:latest": unauthorized
```

kube-proxy logs:

- **What it logs** - network proxy operations, service endpoint updates, iptables/IPVS rule changes.
- **Security value** - network policy bypasses, suspicious service access, port scanning detection.

Container runtime logs (containerd, Docker, CRI-O):

- **What it logs** - container lifecycle events, image operations, runtime errors.
- **Security value** - container escape attempts, runtime vulnerabilities exploited, image integrity violations.

3. Application logs:

Pod/Container logs (stdout/stderr):

- **What it logs** - application-specific logging, business logic events, errors and exceptions.
- **Security value** - application-level attacks (SQL injection, XSS), authentication failures, suspicious user behavior, data access patterns.

Sidecar logs:

- **What it logs** - service mesh traffic (Istio, Linkerd), logging agents (Fluentd, Fluent Bit), security scanning (Falco).

4. Audit logs (critical for security):

Kubernetes Audit Logs:

- **What they capture** - comprehensive audit trail of all API server interactions, configurable levels (None, Metadata, Request, RequestResponse), policy-driven (what to log).
- **Security value** - complete forensic record, compliance requirements (PCI-DSS, HIPAA, SOC 2), incident investigation, insider threat detection.

Audit policy levels:

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
  # Log all secret access at RequestResponse level (full payload)
  - level: RequestResponse
    resources:
      - group: ""
        resources: ["secrets"]

  # Log all authentication/authorization at Metadata level
  - level: Metadata
    omitStages:
      - RequestReceived
    verbs: ["create", "update", "patch", "delete"]

  # Don't log read-only requests to non-sensitive resources
  - level: None
    verbs: ["get", "list", "watch"]
    resources:
      - group: ""
        resources: ["configmaps", "endpoints"]

  # Log everything else at Request level (no response body)
  - level: Request
```

5. Cluster add-on logs:

DNS logs (CoreDNS):

- **What they log** - DNS queries and responses, cache behavior, query failures.
- **Security value** - DNS tunneling detection, C2 communication, data exfiltration via DNS, malicious domain access.

Ingress controller logs:

- **What they log** - HTTP/HTTPS requests, TLS handshakes, routing decisions, rate limiting.
- **Security value** - web attacks (OWASP Top 10), DDoS attempts, suspicious user agents, geographic anomalies.

Network policy logs:

- **What they log** - allowed/denied connections, policy violations.
- **Security value** - lateral movement attempts, unauthorized service access, network reconnaissance.

Log security priorities:

Highest priority (must monitor):

1. kube-apiserver audit logs (all API access)
2. Secret/ConfigMap access
3. Role/RoleBinding changes
4. Privileged pod creation
5. Exec/port-forward sessions

High priority:

1. Authentication failures
2. Admission webhook denials
3. Image pull failures
4. Suspicious network connections
5. Node kubelet errors

Medium priority:

1. Application errors
2. DNS queries
3. Ingress logs
4. Resource quota violations

Storage considerations:

- Control plane logs: 1-5 GB/day per cluster
- Node logs: 500 MB - 2 GB/day per node
- Application logs: Varies widely (1-100 GB/day)
- Audit logs with RequestResponse: 10-50 GB/day (high verbosity)

Retention recommendations:

- Hot storage (immediate query): 30-90 days

- Warm storage (archive query): 1 year
- Cold storage (compliance): 7 years (regulatory requirement)

Best practices:

- Enable audit logging on all clusters (production mandatory)
- Use Metadata level minimum (RequestResponse for secrets)
- Separate control plane from application logs
- Implement log aggregation (don't rely on node storage)
- Encrypt logs at rest and in transit
- Implement log integrity protection (immutable storage)
- Regular log review and anomaly detection
- Automated alerting on security events

Understanding Kubernetes log types enables comprehensive security monitoring - control plane logs provide cluster-level visibility while node and application logs reveal runtime security events.

What are Kubernetes audit logs, how do you configure them, and what audit policy should you implement for security monitoring?

Kubernetes audit logs provide chronological record of all API server activities, essential for security monitoring, compliance, and forensic investigations.

Audit log fundamentals:

What audit logs capture:

- Every request to the kube-apiserver
- Who made the request (user, ServiceAccount, system component)
- What action was requested (verb: get, create, delete, etc.)
- Which resource was targeted
- When the request occurred
- Source IP and user agent
- Request payload (configurable)
- Response status and body (configurable)

Audit stages (lifecycle of request):

```
RequestReceived → ResponseStarted → ResponseComplete → Panic
```

- **RequestReceived**: Audit event generated as soon as request received, before any processing
- **ResponseStarted**: For long-running requests (watch), logged when headers sent but before body
- **ResponseComplete**: After response body sent
- **Panic**: Generated when request handler panicked

Audit levels (what to log):

- **None**: Don't log
- **Metadata**: Log request metadata (user, timestamp, resource, verb) but not request/response bodies
- **Request**: Log metadata and request body, not response
- **RequestResponse**: Log everything (metadata, request, and response bodies)

Configuring audit logging:

Method 1: kube-apiserver flags (most common):

```
# /etc/kubernetes/manifests/kube-apiserver.yaml (static pod)
apiVersion: v1
kind: Pod
metadata:
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - name: kube-apiserver
    image: registry.k8s.io/kube-apiserver:v1.28.0
    command:
    - kube-apiserver
    # Audit configuration
    - --audit-policy-file=/etc/kubernetes/audit-policy.yaml
    - --audit-log-path=/var/log/kubernetes/audit/audit.log
    - --audit-log-maxage=30          # Retain 30 days
    - --audit-log-maxbackup=10       # Keep 10 rotated files
    - --audit-log-maxsize=100        # Rotate at 100MB
    # Dynamic backend (webhook for real-time)
    - --audit-webhook-config-file=/etc/kubernetes/audit-webhook-config.yaml
    - --audit-webhook-initial-backoff=10s
    volumeMounts:
    - name: audit-policy
      mountPath: /etc/kubernetes/audit-policy.yaml
      readOnly: true
    - name: audit-log
      mountPath: /var/log/kubernetes/audit/
```

```
    - name: audit-webhook
      mountPath: /etc/kubernetes/audit-webhook-config.yaml
      readOnly: true
  volumes:
  - name: audit-policy
    hostPath:
      path: /etc/kubernetes/audit-policy.yaml
      type: File
  - name: audit-log
    hostPath:
      path: /var/log/kubernetes/audit/
      type: DirectoryOrCreate
  - name: audit-webhook
    hostPath:
      path: /etc/kubernetes/audit-webhook-config.yaml
      type: File
```

Security-focused audit policy:

```
# /etc/kubernetes/audit-policy.yaml
apiVersion: audit.k8s.io/v1
kind: Policy
omitStages:
  - RequestReceived  # Don't log until request processed
rules:
  # ============================================
  # CRITICAL: Full logging for security resources
  # ============================================

  # Secrets - log everything (RequestResponse level)
  - level: RequestResponse
    resources:
      - group: ""
        resources: ["secrets"]
    omitManagedFields: true  # Reduce noise

  # ServiceAccounts and tokens
  - level: RequestResponse
    resources:
      - group: ""
        resources: ["serviceaccounts", "serviceaccounts/token"]

  # RBAC resources (roles, bindings, etc.)
  - level: RequestResponse
    resources:
      - group: "rbac.authorization.k8s.io"
        resources: ["roles", "rolebindings", "clusterroles", "clusterrolebindings"]

  # Security-sensitive pod operations
  - level: RequestResponse
```

```
    verbs: ["create", "update", "patch"]
    resources:
      - group: ""
        resources: ["pods", "pods/exec", "pods/portforward", "pods/proxy"]

  # Network policies
  - level: RequestResponse
    resources:
      - group: "networking.k8s.io"
        resources: ["networkpolicies"]

  # Pod security policies (deprecated but still used)
  - level: RequestResponse
    resources:
      - group: "policy"
        resources: ["podsecuritypolicies"]

  # Admission webhooks (can bypass security)
  - level: RequestResponse
    resources:
      - group: "admissionregistration.k8s.io"
        resources: ["mutatingwebhookconfigurations",
"validatingwebhookconfigurations"]

  # ================================================
  # HIGH: Metadata for important operations
  # ================================================

  # ConfigMaps (may contain sensitive data)
  - level: Metadata
    resources:
      - group: ""
        resources: ["configmaps"]

  # Persistent volumes (data access)
  - level: Metadata
    resources:
      - group: ""
        resources: ["persistentvolumes", "persistentvolumeclaims"]

  # Nodes (infrastructure)
  - level: Metadata
    resources:
      - group: ""
        resources: ["nodes"]

  # All creates, updates, deletes
  - level: Metadata
    verbs: ["create", "update", "patch", "delete"]

  # ================================================
```

```
  # MEDIUM: Request level for write operations
  # ============================================

  # Deployments, StatefulSets, DaemonSets
  - level: Request
    verbs: ["create", "update", "patch", "delete"]
    resources:
      - group: "apps"
        resources: ["deployments", "statefulsets", "daemonsets", "replicasets"]

  # Jobs and CronJobs
  - level: Request
    verbs: ["create", "update", "patch", "delete"]
    resources:
      - group: "batch"
        resources: ["jobs", "cronjobs"]

  # ============================================
  # LOW: Metadata for reads, None for noise
  # ============================================

  # Read-only operations on non-sensitive resources
  - level: Metadata
    verbs: ["get", "list", "watch"]
    resources:
      - group: ""
        resources: ["endpoints", "services", "namespaces"]

  # Exclude high-volume, low-value requests
  - level: None
    users: ["system:kube-proxy"]
    verbs: ["watch"]
    resources:
      - group: ""
        resources: ["endpoints", "services"]

  - level: None
    users: ["system:kube-controller-manager"]
    verbs: ["get", "update"]
    namespaces: ["kube-system"]

  - level: None
    users: ["kubelet"]
    verbs: ["get"]
    resources:
      - group: ""
        resources: ["nodes"]

  # System components health checks
  - level: None
    userGroups: ["system:nodes"]
```

```
    verbs: ["get"]
    resources:
      - group: ""
        resources: ["nodes", "nodes/status"]

  # Exclude read-only URLs
  - level: None
    nonResourceURLs:
      - /healthz*
      - /version
      - /swagger*

  # ============================================
  # DEFAULT: Catch-all for everything else
  # ============================================

  - level: Metadata
```

Webhook backend for real-time streaming:

```
# /etc/kubernetes/audit-webhook-config.yaml
apiVersion: v1
kind: Config
clusters:
- name: audit-webhook
  cluster:
    server: https://audit-collector.monitoring.svc.cluster.local:8443/api/v1/audit
    certificate-authority: /etc/kubernetes/pki/ca.crt
contexts:
- name: default
  context:
    cluster: audit-webhook
    user: audit-apiserver
current-context: default
users:
- name: audit-apiserver
  user:
    client-certificate: /etc/kubernetes/pki/audit-client.crt
    client-key: /etc/kubernetes/pki/audit-client.key
```

Managed Kubernetes audit configuration:

AKS (Azure Kubernetes Service):

```
# Enable diagnostic settings
az aks update \
  --resource-group myResourceGroup \
  --name myAKSCluster \
  --enable-azure-rbac \
```

```
  --enable-audit-logs

# Configure diagnostic settings to send to Log Analytics
az monitor diagnostic-settings create \
  --resource /subscriptions/{sub-
id}/resourceGroups/myResourceGroup/providers/Microsoft.ContainerService/managedCluster
s/myAKSCluster \
  --name audit-logs-to-sentinel \
  --workspace /subscriptions/{sub-
id}/resourceGroups/myResourceGroup/providers/Microsoft.OperationalInsights/workspaces/
sentinel-workspace \
  --logs '[
    {
      "category": "kube-audit",
      "enabled": true,
      "retentionPolicy": {
        "enabled": true,
        "days": 90
      }
    },
    {
      "category": "kube-audit-admin",
      "enabled": true,
      "retentionPolicy": {
        "enabled": true,
        "days": 90
      }
    }
  ]'
```

EKS (Amazon Elastic Kubernetes Service):

```
# Enable control plane logging
aws eks update-cluster-config \
  --name my-cluster \
  --logging '{
    "clusterLogging": [{
      "types": ["api", "audit", "authenticator", "controllerManager", "scheduler"],
      "enabled": true
    }]
  }'

# Logs automatically sent to CloudWatch Logs
# Group: /aws/eks/my-cluster/cluster
```

GKE (Google Kubernetes Engine):

```
# Enable audit logging (enabled by default)
gcloud container clusters update my-cluster \
```

```
  --enable-cloud-logging \
  --logging=SYSTEM,WORKLOAD,API

# Logs automatically sent to Cloud Logging
# Can be exported to Cloud Storage, BigQuery, or Pub/Sub
```

Security monitoring queries (examples for analysis):

Detect secret access:

```
// Audit log query
{
  "objectRef.resource": "secrets",
  "verb": ["get", "list"],
  "user.username": {"$ne": "system:kube-controller-manager"}
}
```

Detect privilege escalation:

```
{
  "objectRef.resource": ["clusterrolebindings", "rolebindings"],
  "verb": ["create", "update", "patch"],
  "requestObject.roleRef.name": {"$in": ["cluster-admin", "admin"]}
}
```

Detect exec into pods:

```
{
  "objectRef.resource": "pods",
  "objectRef.subresource": "exec",
  "verb": "create",
  "responseStatus.code": 101  // Switching Protocols (successful)
}
```

Best practices:

- Start with conservative policy (Metadata level)
- Gradually increase to RequestResponse for critical resources
- Exclude high-volume, low-value events (health checks)
- Monitor audit log volume and adjust policy
- Use webhook backend for real-time SIEM integration
- Retain logs for compliance period (typically 1-7 years)
- Encrypt audit logs at rest
- Implement log integrity protection

- Regular review and tuning of audit policy
- Test policy changes in non-production first
- Document all policy decisions and exemptions

Proper audit configuration provides comprehensive security visibility while managing log volume and storage costs - essential for detecting threats, investigating incidents, and meeting compliance requirements.

How do you integrate Kubernetes logs (including control plane logs) with Microsoft Sentinel without using Grafana?

Integrating Kubernetes logs with Sentinel provides centralized security monitoring, correlation with other data sources, and advanced threat detection using Sentinel's analytics capabilities.

Integration architecture options:

Option 1: Azure Monitor Container Insights (AKS-native, recommended):

```
AKS Cluster → Container Insights Agent → Log Analytics Workspace → Sentinel
```

Option 2: Fluent Bit/Fluentd (flexible, multi-cloud):

```
K8s Cluster → Fluent Bit DaemonSet → Log Analytics → Sentinel
```

Option 3: Azure Event Hub (high-throughput):

```
K8s Cluster → Fluent Bit → Event Hub → Stream Analytics → Sentinel
```

Option 4: Direct API ingestion:

```
K8s Cluster → Custom Exporter → Data Collection API → Sentinel
```

Detailed implementation - Option 1: Azure Monitor Container Insights (AKS):

Enable Container Insights:

```
# Enable on existing AKS cluster
az aks enable-addons \
  --resource-group myResourceGroup \
  --name myAKSCluster \
  --addons monitoring \
```

```
  --workspace-resource-id /subscriptions/{sub-id}/resourceGroups/sentinel-
rg/providers/Microsoft.OperationalInsights/workspaces/sentinel-workspace

# Verify agent deployment
kubectl get ds omsagent -n kube-system
kubectl get deployment omsagent-rs -n kube-system
```

Configure data collection:

```
# ConfigMap for Container Insights
apiVersion: v1
kind: ConfigMap
metadata:
  name: container-azm-ms-agentconfig
  namespace: kube-system
data:
  schema-version: v1
  config-version: ver1

  # Log collection settings
  log-data-collection-settings: |-
    [log_collection_settings]
       [log_collection_settings.stdout]
          enabled = true
          exclude_namespaces = ["kube-system", "kube-public"]

       [log_collection_settings.stderr]
          enabled = true
          exclude_namespaces = ["kube-system"]

       [log_collection_settings.env_var]
          enabled = true

       [log_collection_settings.enrich_container_logs]
          enabled = true

  # Prometheus scraping (optional)
  prometheus-data-collection-settings: |-
    [prometheus_data_collection_settings.cluster]
        interval = "1m"
        monitor_kubernetes_pods = true

    [prometheus_data_collection_settings.node]
        interval = "1m"
```

Enable control plane logs:

```
# Enable diagnostic settings for control plane
az monitor diagnostic-settings create \
```

```
  --resource /subscriptions/{sub-
id}/resourceGroups/myResourceGroup/providers/Microsoft.ContainerService/managedCluster
s/myAKSCluster \
  --name control-plane-logs \
  --workspace /subscriptions/{sub-id}/resourceGroups/sentinel-
rg/providers/Microsoft.OperationalInsights/workspaces/sentinel-workspace \
  --logs '[
    {
      "category": "kube-apiserver",
      "enabled": true
    },
    {
      "category": "kube-audit",
      "enabled": true
    },
    {
      "category": "kube-audit-admin",
      "enabled": true
    },
    {
      "category": "kube-controller-manager",
      "enabled": true
    },
    {
      "category": "kube-scheduler",
      "enabled": true
    },
    {
      "category": "cluster-autoscaler",
      "enabled": true
    },
    {
      "category": "cloud-controller-manager",
      "enabled": true
    },
    {
      "category": "guard",
      "enabled": true
    },
    {
      "category": "csi-azuredisk-controller",
      "enabled": true
    },
    {
      "category": "csi-azurefile-controller",
      "enabled": true
    }
  ]' \
  --metrics '[
    {
      "category": "AllMetrics",
```

```
      "enabled": true
    }
  ]'
```

Query control plane logs in Sentinel:

```
// kube-apiserver audit logs
AzureDiagnostics
| where Category == "kube-audit" or Category == "kube-audit-admin"
| where TimeGenerated > ago(24h)
| extend AuditLog = parse_json(log_s)
| extend
    User = tostring(AuditLog.user.username),
    Verb = tostring(AuditLog.verb),
    Resource = tostring(AuditLog.objectRef.resource),
    Namespace = tostring(AuditLog.objectRef.namespace),
    Name = tostring(AuditLog.objectRef.name),
    ResponseCode = toint(AuditLog.responseStatus.code),
    SourceIP = tostring(AuditLog.sourceIPs[0])
| project TimeGenerated, User, Verb, Resource, Namespace, Name, ResponseCode, SourceIP
| order by TimeGenerated desc

// kube-controller-manager logs
AzureDiagnostics
| where Category == "kube-controller-manager"
| where TimeGenerated > ago(1h)
| project TimeGenerated, log_s

// kube-scheduler logs
AzureDiagnostics
| where Category == "kube-scheduler"
| where TimeGenerated > ago(1h)
| project TimeGenerated, log_s
```

Detailed implementation - Option 2: Fluent Bit (works on any K8s):

Deploy Fluent Bit DaemonSet:

```
apiVersion: v1
kind: Namespace
metadata:
  name: logging
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: fluent-bit
  namespace: logging
---
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: fluent-bit-read
rules:
- apiGroups: [""]
  resources:
    - namespaces
    - pods
    - nodes
  verbs: ["get", "list", "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: fluent-bit-read
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: fluent-bit-read
subjects:
- kind: ServiceAccount
  name: fluent-bit
  namespace: logging
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: fluent-bit-config
  namespace: logging
data:
  fluent-bit.conf: |
    [SERVICE]
        Flush         5
        Daemon        Off
        Log_Level     info
        Parsers_File  parsers.conf

    # Tail container logs
    [INPUT]
        Name              tail
        Path              /var/log/containers/*.log
        Parser            docker
        Tag               kube.*
        Refresh_Interval  5
        Mem_Buf_Limit     50MB
        Skip_Long_Lines   On

    # Tail control plane logs (if accessible)
    [INPUT]
        Name              tail
```

```
        Path              /var/log/kube-apiserver-audit.log
        Parser            json
        Tag               audit.kube-apiserver
        Mem_Buf_Limit     100MB

    # Kubernetes metadata enrichment
    [FILTER]
        Name                kubernetes
        Match               kube.*
        Kube_URL            https://kubernetes.default.svc:443
        Kube_CA_File        /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        Kube_Token_File     /var/run/secrets/kubernetes.io/serviceaccount/token
        Kube_Tag_Prefix     kube.var.log.containers.
        Merge_Log           On
        Keep_Log            Off
        K8S-Logging.Parser  On
        K8S-Logging.Exclude On

    # Add cluster name
    [FILTER]
        Name    modify
        Match   *
        Add     cluster_name production-cluster-01
        Add     environment production

    # Output to Azure Log Analytics
    [OUTPUT]
        Name                azure
        Match               *
        Customer_ID         ${WORKSPACE_ID}
        Shared_Key          ${WORKSPACE_KEY}
        Log_Type            KubernetesLogs
        Time_Generated      true

  parsers.conf: |
    [PARSER]
        Name        docker
        Format      json
        Time_Key    time
        Time_Format %Y-%m-%dT%H:%M:%S.%L%z

    [PARSER]
        Name        json
        Format      json
        Time_Key    timestamp
        Time_Format %Y-%m-%dT%H:%M:%S.%L%z
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluent-bit
```

```
  namespace: logging
  labels:
    app: fluent-bit
spec:
  selector:
    matchLabels:
      app: fluent-bit
  template:
    metadata:
      labels:
        app: fluent-bit
    spec:
      serviceAccountName: fluent-bit
      containers:
      - name: fluent-bit
        image: fluent/fluent-bit:2.1.0
        env:
        - name: WORKSPACE_ID
          valueFrom:
            secretKeyRef:
              name: log-analytics-secret
              key: workspace-id
        - name: WORKSPACE_KEY
          valueFrom:
            secretKeyRef:
              name: log-analytics-secret
              key: workspace-key
        resources:
          limits:
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 100Mi
        volumeMounts:
        - name: varlog
          mountPath: /var/log
          readOnly: true
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
        - name: fluent-bit-config
          mountPath: /fluent-bit/etc/
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
      - name: fluent-bit-config
        configMap:
```

```
          name: fluent-bit-config
```

Create Log Analytics secret:

```
# Get workspace credentials
WORKSPACE_ID=$(az monitor log-analytics workspace show \
  --resource-group sentinel-rg \
  --workspace-name sentinel-workspace \
  --query customerId -o tsv)

WORKSPACE_KEY=$(az monitor log-analytics workspace get-shared-keys \
  --resource-group sentinel-rg \
  --workspace-name sentinel-workspace \
  --query primarySharedKey -o tsv)

# Create Kubernetes secret
kubectl create secret generic log-analytics-secret \
  --from-literal=workspace-id=$WORKSPACE_ID \
  --from-literal=workspace-key=$WORKSPACE_KEY \
  --namespace logging
```

Detailed implementation - Option 3: Control Plane Log Collection (self-managed clusters):

Collect audit logs with sidecar:

```
# If running kube-apiserver as static pod
apiVersion: v1
kind: Pod
metadata:
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  # Main kube-apiserver container
  - name: kube-apiserver
    image: registry.k8s.io/kube-apiserver:v1.28.0
    command:
    - kube-apiserver
    - --audit-log-path=/var/log/kubernetes/audit.log
    - --audit-policy-file=/etc/kubernetes/audit-policy.yaml
    volumeMounts:
    - name: audit-log
      mountPath: /var/log/kubernetes

  # Sidecar to ship logs
  - name: audit-log-shipper
    image: fluent/fluent-bit:2.1.0
    env:
    - name: WORKSPACE_ID
```

```
      valueFrom:
        secretKeyRef:
          name: log-analytics-secret
          key: workspace-id
    - name: WORKSPACE_KEY
      valueFrom:
        secretKeyRef:
          name: log-analytics-secret
          key: workspace-key
    volumeMounts:
    - name: audit-log
      mountPath: /var/log/kubernetes
      readOnly: true
    - name: fluent-bit-audit-config
      mountPath: /fluent-bit/etc/

  volumes:
  - name: audit-log
    hostPath:
      path: /var/log/kubernetes
      type: DirectoryOrCreate
  - name: fluent-bit-audit-config
    configMap:
      name: fluent-bit-audit-config
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: fluent-bit-audit-config
  namespace: kube-system
data:
  fluent-bit.conf: |
    [SERVICE]
        Flush         5
        Log_Level     info

    [INPUT]
        Name              tail
        Path              /var/log/kubernetes/audit.log
        Parser            json
        Tag               audit.kube-apiserver
        Refresh_Interval  5
        Mem_Buf_Limit     100MB

    [FILTER]
        Name    modify
        Match   *
        Add     log_type kube-audit
        Add     cluster_name my-cluster

    [OUTPUT]
```

```
        Name                azure
        Match               *
        Customer_ID         ${WORKSPACE_ID}
        Shared_Key          ${WORKSPACE_KEY}
        Log_Type            KubeAudit
```

Query Kubernetes logs in Sentinel:

```
// Container logs
KubernetesLogs_CL
| where TimeGenerated > ago(1h)
| where Namespace_s == "production"
| where ContainerName_s == "webapp"
| project TimeGenerated, Computer, PodName_s, ContainerName_s, LogEntry_s
| order by TimeGenerated desc

// Audit logs
KubeAudit_CL
| where TimeGenerated > ago(24h)
| extend Audit = parse_json(log_s)
| extend
    User = tostring(Audit.user.username),
    Verb = tostring(Audit.verb),
    Resource = tostring(Audit.objectRef.resource),
    ResponseCode = toint(Audit.responseStatus.code)
| where Resource == "secrets"
| project TimeGenerated, User, Verb, Resource, ResponseCode
```

Sentinel Analytics Rules (K8s-specific):

```
// Detect privileged pod creation
KubeAudit_CL
| where TimeGenerated > ago(5m)
| extend Audit = parse_json(log_s)
| where tostring(Audit.verb) == "create"
| where tostring(Audit.objectRef.resource) == "pods"
| extend RequestObj = parse_json(tostring(Audit.requestObject))
| where RequestObj.spec.containers[0].securityContext.privileged == true
| project
    TimeGenerated,
    User = tostring(Audit.user.username),
    Namespace = tostring(Audit.objectRef.namespace),
    PodName = tostring(Audit.objectRef.name),
    SourceIP = tostring(Audit.sourceIPs[0])
```

Best practices for Kubernetes-Sentinel integration covered monitoring control plane, node, and application logs with proper enrichment, filtering, and security-focused analytics.

How do you integrate Kubernetes logs with Microsoft Sentinel using Grafana Loki as an intermediary, and what are the advantages of this approach?

Using Grafana Loki as a log aggregation layer before Sentinel provides additional benefits for log processing, cost optimization, and flexible querying while maintaining comprehensive security monitoring.

Architecture with Loki:

```
K8s Cluster → Promtail/Fluentd → Loki → Logstash/Custom Exporter → Sentinel
                                   ↓
                              Grafana (Optional visualization)
```

Why use Loki as intermediary:

Advantages:

- **Cost optimization** - filter and aggregate logs before sending to Sentinel (reduce ingestion costs), compress and deduplicate logs, selective forwarding of security-relevant events
- **Query flexibility** - LogQL for initial analysis and triage, Grafana dashboards for operational teams, KQL in Sentinel for security analytics
- **Buffer and resilience** - Loki acts as buffer during Sentinel outages, replay capability for missed events, local retention for fast queries
- **Multi-destination** - same logs to Sentinel (security) and other systems (operations), different retention policies per destination
- **Label-based filtering** - efficient filtering using Loki labels before expensive queries, reduce data sent to Sentinel

Detailed implementation:

Step 1: Deploy Loki in Kubernetes:

```
# Using Helm (recommended)
# Add Grafana Helm repo
helm repo add grafana https://grafana.github.io/helm-charts
helm repo update

# Create namespace
kubectl create namespace monitoring

# Install Loki with distributed mode for production
```

```
cat <<EOF > loki-values.yaml
loki:
  auth_enabled: false

  commonConfig:
    replication_factor: 3

  storage:
    type: 's3'
    bucketNames:
      chunks: loki-chunks
      ruler: loki-ruler
      admin: loki-admin
    s3:
      endpoint: s3.us-east-1.amazonaws.com
      region: us-east-1
      secretAccessKey: ${AWS_SECRET_KEY}
      accessKeyId: ${AWS_ACCESS_KEY}
      s3ForcePathStyle: false
      insecure: false

  schemaConfig:
    configs:
      - from: 2024-01-01
        store: tsdb
        object_store: s3
        schema: v12
        index:
          prefix: index_
          period: 24h

  limits_config:
    retention_period: 30d
    ingestion_rate_mb: 50
    ingestion_burst_size_mb: 100
    max_query_length: 30d
    max_query_parallelism: 32

# Resource allocation
read:
  replicas: 3
  resources:
    limits:
      cpu: 2
      memory: 4Gi
    requests:
      cpu: 1
      memory: 2Gi

write:
  replicas: 3
```

```
  resources:
    limits:
      cpu: 2
      memory: 4Gi
    requests:
      cpu: 1
      memory: 2Gi

backend:
  replicas: 3
  resources:
    limits:
      cpu: 2
      memory: 4Gi
    requests:
      cpu: 1
      memory: 2Gi

gateway:
  enabled: true
  replicas: 2
  resources:
    limits:
      cpu: 1
      memory: 1Gi
    requests:
      cpu: 500m
      memory: 512Mi

# Security
serviceAccount:
  create: true
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::123456789012:role/loki-storage-role
EOF

helm install loki grafana/loki-distributed -f loki-values.yaml -n monitoring
```

Step 2: Deploy Promtail to ship logs to Loki:

```
cat <<EOF > promtail-values.yaml
config:
  clients:
    - url: http://loki-gateway.monitoring.svc.cluster.local/loki/api/v1/push

  snippets:
    scrapeConfigs: |
      # Scrape pod logs
      - job_name: kubernetes-pods
        pipeline_stages:
```

```
      - cri: {}
      - json:
          expressions:
            stream: stream
      - labels:
          stream:
    kubernetes_sd_configs:
      - role: pod
    relabel_configs:
      # Only scrape pods with logging enabled
      - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
        action: keep
        regex: true

      # Add namespace label
      - source_labels: [__meta_kubernetes_namespace]
        target_label: namespace

      # Add pod name
      - source_labels: [__meta_kubernetes_pod_name]
        target_label: pod

      # Add container name
      - source_labels: [__meta_kubernetes_pod_container_name]
        target_label: container

      # Add node name
      - source_labels: [__meta_kubernetes_pod_node_name]
        target_label: node

      # Add security labels for filtering
      - source_labels: [__meta_kubernetes_pod_label_security_monitoring]
        target_label: security_monitored

      # Add environment
      - source_labels: [__meta_kubernetes_namespace]
        target_label: environment
        regex: (.+)
        replacement: $1

  # Scrape control plane logs (if accessible)
  - job_name: kubernetes-control-plane
    static_configs:
      - targets:
          - localhost
        labels:
          job: control-plane
          __path__: /var/log/kube-apiserver-audit.log
    pipeline_stages:
      - json:
          expressions:
```

```
                user: user.username
                verb: verb
                resource: objectRef.resource
                namespace: objectRef.namespace
                responseCode: responseStatus.code
          - labels:
              user:
              verb:
              resource:
              namespace:
          - match:
              selector: '{job="control-plane"}'
              stages:
                - static_labels:
                    log_type: audit
                    security_critical: "true"

# DaemonSet for node coverage
daemonSet:
  enabled: true

# Security
serviceAccount:
  create: true

# Resources
resources:
  limits:
    cpu: 200m
    memory: 256Mi
  requests:
    cpu: 100m
    memory: 128Mi

# Volume mounts for log access
extraVolumes:
  - name: audit-logs
    hostPath:
      path: /var/log/kubernetes
      type: DirectoryOrCreate

extraVolumeMounts:
  - name: audit-logs
    mountPath: /var/log/kubernetes
    readOnly: true
EOF

helm install promtail grafana/promtail -f promtail-values.yaml -n monitoring
```

Step 3: Create Loki to Sentinel exporter:

```
# Custom Python service to query Loki and forward to Sentinel
import requests
import time
import json
import hashlib
from datetime import datetime, timedelta
from azure.monitor.ingestion import LogsIngestionClient
from azure.identity import DefaultAzureCredential

# Configuration
LOKI_URL = "http://loki-gateway.monitoring.svc.cluster.local:80"
SENTINEL_DCE = "https://dce-production.eastus-1.ingest.monitor.azure.com"
SENTINEL_DCR_ID = "dcr-xxxxxxxxxxxx"
SENTINEL_STREAM = "Custom-KubernetesLogs_CL"

# Initialize Sentinel client
credential = DefaultAzureCredential()
sentinel_client = LogsIngestionClient(
    endpoint=SENTINEL_DCE,
    credential=credential
)

# Track last processed timestamp
last_processed = {}

def query_loki(query, start_time, end_time):
    """Query Loki for logs"""
    params = {
        'query': query,
        'start': int(start_time.timestamp() * 1e9),  # nanoseconds
        'end': int(end_time.timestamp() * 1e9),
        'limit': 5000
    }

    response = requests.get(
        f"{LOKI_URL}/loki/api/v1/query_range",
        params=params
    )

    if response.status_code == 200:
        return response.json()['data']['result']
    else:
        raise Exception(f"Loki query failed: {response.text}")

def transform_to_sentinel_format(loki_logs):
    """Transform Loki logs to Sentinel schema"""
    sentinel_logs = []

    for stream in loki_logs:
        labels = stream['stream']
```

```
        for entry in stream['values']:
            timestamp_ns, log_line = entry
            timestamp = datetime.fromtimestamp(int(timestamp_ns) / 1e9)

            # Parse JSON log if possible
            try:
                log_data = json.loads(log_line)
            except:
                log_data = {'message': log_line}

            sentinel_log = {
                'TimeGenerated': timestamp.isoformat() + 'Z',
                'Cluster': labels.get('cluster', 'unknown'),
                'Namespace': labels.get('namespace', ''),
                'Pod': labels.get('pod', ''),
                'Container': labels.get('container', ''),
                'Node': labels.get('node', ''),
                'LogLevel': log_data.get('level', 'INFO'),
                'Message': log_data.get('message', log_line),
                'Labels': json.dumps(labels),
                'LogData': json.dumps(log_data)
            }

            sentinel_logs.append(sentinel_log)

    return sentinel_logs

def forward_to_sentinel(logs, log_type):
    """Send logs to Sentinel"""
    if not logs:
        return

    try:
        sentinel_client.upload(
            rule_id=SENTINEL_DCR_ID,
            stream_name=SENTINEL_STREAM,
            logs=logs
        )
        print(f"Forwarded {len(logs)} {log_type} logs to Sentinel")
    except Exception as e:
        print(f"Failed to forward logs: {str(e)}")

def process_security_logs():
    """Process security-critical logs from Loki"""
    end_time = datetime.utcnow()
    start_time = last_processed.get('security', end_time - timedelta(minutes=5))

    # Query 1: Audit logs
    audit_query = '{log_type="audit", security_critical="true"}'
    audit_logs = query_loki(audit_query, start_time, end_time)
```

```
    audit_sentinel = transform_to_sentinel_format(audit_logs)
    forward_to_sentinel(audit_sentinel, 'audit')

    # Query 2: Security-monitored pods
    security_query = '{security_monitored="true"} |= "error" or "failed" or
"unauthorized" or "forbidden"'
    security_logs = query_loki(security_query, start_time, end_time)
    security_sentinel = transform_to_sentinel_format(security_logs)
    forward_to_sentinel(security_sentinel, 'security')

    # Query 3: Privileged container logs
    privileged_query = '{namespace=~"production|staging"} | json | privileged="true"'
    privileged_logs = query_loki(privileged_query, start_time, end_time)
    privileged_sentinel = transform_to_sentinel_format(privileged_logs)
    forward_to_sentinel(privileged_sentinel, 'privileged')

    last_processed['security'] = end_time

def process_error_logs():
    """Process error logs from Loki"""
    end_time = datetime.utcnow()
    start_time = last_processed.get('errors', end_time - timedelta(minutes=5))

    # Query for errors across all pods
    error_query = '{namespace!~"kube-system|kube-public"} |~
"(?i)error|exception|fatal"'
    error_logs = query_loki(error_query, start_time, end_time)
    error_sentinel = transform_to_sentinel_format(error_logs)
    forward_to_sentinel(error_sentinel, 'errors')

    last_processed['errors'] = end_time

def main():
    """Main processing loop"""
    print("Starting Loki to Sentinel forwarder...")

    while True:
        try:
            # Process different log types
            process_security_logs()
            process_error_logs()

            # Sleep before next iteration
            time.sleep(60)  # Run every minute

        except Exception as e:
            print(f"Error in processing loop: {str(e)}")
            time.sleep(60)

if __name__ == "__main__":
```

```
    main()
```

Deploy exporter as Kubernetes Deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: loki-sentinel-exporter
  namespace: monitoring
spec:
  replicas: 2
  selector:
    matchLabels:
      app: loki-sentinel-exporter
  template:
    metadata:
      labels:
        app: loki-sentinel-exporter
    spec:
      serviceAccountName: loki-sentinel-exporter
      containers:
      - name: exporter
        image: myregistry.azurecr.io/loki-sentinel-exporter:latest
        env:
        - name: LOKI_URL
          value: "http://loki-gateway.monitoring.svc.cluster.local:80"
        - name: AZURE_CLIENT_ID
          valueFrom:
            secretKeyRef:
              name: sentinel-credentials
              key: client-id
        - name: AZURE_TENANT_ID
          valueFrom:
            secretKeyRef:
              name: sentinel-credentials
              key: tenant-id
        - name: AZURE_CLIENT_SECRET
          valueFrom:
            secretKeyRef:
              name: sentinel-credentials
              key: client-secret
        resources:
          limits:
            cpu: 500m
            memory: 512Mi
          requests:
            cpu: 200m
            memory: 256Mi
---
apiVersion: v1
```

```
kind: ServiceAccount
metadata:
  name: loki-sentinel-exporter
  namespace: monitoring
  annotations:
    azure.workload.identity/client-id: "your-managed-identity-client-id"
```

Step 4: Query and analyze in Sentinel:

```
// Query forwarded Kubernetes logs
KubernetesLogs_CL
| where TimeGenerated > ago(1h)
| extend Labels = parse_json(Labels_s)
| extend LogData = parse_json(LogData_s)
| where Namespace_s == "production"
| project
    TimeGenerated,
    Cluster_s,
    Namespace_s,
    Pod_s,
    Container_s,
    LogLevel_s,
    Message_s,
    Labels,
    LogData
| order by TimeGenerated desc

// Detect suspicious activity
KubernetesLogs_CL
| where TimeGenerated > ago(5m)
| where Message_s contains "error" or Message_s contains "failed"
| extend Labels = parse_json(Labels_s)
| where Labels.security_monitored == "true"
| summarize
    ErrorCount = count(),
    Pods = make_set(Pod_s)
    by Namespace_s, bin(TimeGenerated, 5m)
| where ErrorCount > 10
```

Advanced filtering with Loki:

```
# Loki configuration for selective forwarding
# Only forward security-relevant logs to reduce Sentinel costs

limits_config:
  # Retention in Loki (cheap storage)
  retention_period: 30d

  # Rate limits
```

```
  ingestion_rate_mb: 50
  ingestion_burst_size_mb: 100

# Compactor for cost optimization
compactor:
  working_directory: /data/compactor
  shared_store: s3
  compaction_interval: 10m
  retention_enabled: true
  retention_delete_delay: 2h
  retention_delete_worker_count: 150

# Query for selective export
# LogQL query in exporter:
# {namespace=~"production|staging", security_monitored="true"}
#   |= "error" or "unauthorized" or "exec" or "secret"
# Only these logs sent to Sentinel
```

Cost comparison:

```
Without Loki (direct to Sentinel):
- All logs: 500 GB/day
- Sentinel ingestion: 500 GB x $2.30/GB = $1,150/day = $34,500/month

With Loki (filtered):
- All logs to Loki: 500 GB/day
- Loki storage (S3): 500 GB x $0.023/GB = $11.50/day = $345/month
- Filtered to Sentinel: 50 GB/day (10% security-critical)
- Sentinel ingestion: 50 GB x $2.30/GB = $115/day = $3,450/month
- Total: $3,795/month

Savings: $30,705/month (89% reduction)
```

Benefits summary:

- **Cost optimization** - 80-90% reduction in Sentinel ingestion costs
- **Query performance** - Fast queries in Loki for operational issues
- **Flexibility** - LogQL for developers, KQL for security team
- **Resilience** - Loki buffers during Sentinel outages
- **Multi-tenant** - Different teams query Loki, security uses Sentinel

Best practices:

- Use Loki labels efficiently (avoid high cardinality)
- Implement retention policies (7 days hot, 30 days warm in Loki)
- Filter aggressively before Sentinel (security-critical only)

- Monitor exporter performance and lag
- Implement alerting for export failures
- Regular review of forwarding rules
- Test query performance in both systems
- Document which logs go where

Using Loki as intermediary provides cost-effective log management while maintaining comprehensive security monitoring in Sentinel - ideal for large-scale Kubernetes deployments where full log ingestion to SIEM would be prohibitively expensive.

What are the key security events you should detect and alert on from Kubernetes control plane logs in Sentinel?

Effective security monitoring requires detection rules targeting specific attack patterns and policy violations visible in control plane logs.

Critical detection scenarios:

1. Unauthorized API access attempts:

```
// Detect failed authentication attempts
AzureDiagnostics
| where Category == "kube-audit"
| where TimeGenerated > ago(5m)
| extend Audit = parse_json(log_s)
| extend
    User = tostring(Audit.user.username),
    ResponseCode = toint(Audit.responseStatus.code),
    Reason = tostring(Audit.responseStatus.reason),
    SourceIP = tostring(Audit.sourceIPs[0])
| where ResponseCode in (401, 403)  // Unauthorized or Forbidden
| summarize
    FailedAttempts = count(),
    Resources = make_set(tostring(Audit.objectRef.resource)),
    FirstAttempt = min(TimeGenerated),
    LastAttempt = max(TimeGenerated)
    by User, SourceIP, Reason
| where FailedAttempts > 5
| extend Severity = case(
    FailedAttempts > 20, "High",
    FailedAttempts > 10, "Medium",
    "Low"
)
```

2. Secret access monitoring:

```
// Alert on secret enumeration or mass access
AzureDiagnostics
| where Category == "kube-audit"
| where TimeGenerated > ago(10m)
| extend Audit = parse_json(log_s)
| where tostring(Audit.objectRef.resource) == "secrets"
| where tostring(Audit.verb) in ("get", "list")
| extend
    User = tostring(Audit.user.username),
    Namespace = tostring(Audit.objectRef.namespace),
    SecretName = tostring(Audit.objectRef.name),
    SourceIP = tostring(Audit.sourceIPs[0])
| summarize
    UniqueSecrets = dcount(SecretName),
    Namespaces = make_set(Namespace),
    Secrets = make_set(SecretName)
    by User, SourceIP, bin(TimeGenerated, 5m)
| where UniqueSecrets > 10  // Accessed >10 secrets in 5 min
| extend
    AlertTitle = strcat("Secret Enumeration: ", User, " accessed ", UniqueSecrets, "
secrets"),
    Severity = "High"
```

3. Privilege escalation detection:

```
// Detect creation of privileged roles or bindings
AzureDiagnostics
| where Category == "kube-audit"
| where TimeGenerated > ago(5m)
| extend Audit = parse_json(log_s)
| where tostring(Audit.objectRef.resource) in ("clusterrolebindings", "rolebindings")
| where tostring(Audit.verb) in ("create", "update", "patch")
| extend
    User = tostring(Audit.user.username),
    BindingName = tostring(Audit.objectRef.name),
    RequestObj = parse_json(tostring(Audit.requestObject))
| extend RoleName = tostring(RequestObj.roleRef.name)
| where RoleName in ("cluster-admin", "admin", "edit")  // Privileged roles
| project
    TimeGenerated,
    User,
    BindingName,
    RoleName,
    Namespace = tostring(Audit.objectRef.namespace),
    Subjects = RequestObj.subjects,
    SourceIP = tostring(Audit.sourceIPs[0])
| extend
```

```
    AlertTitle = strcat("Privilege Escalation: ", User, " granted ", RoleName, " to ",
tostring(Subjects[0].name)),
    Severity = "Critical"
```

4. Exec/Port-forward sessions:

```
// Monitor interactive access to containers
AzureDiagnostics
| where Category == "kube-audit"
| where TimeGenerated > ago(5m)
| extend Audit = parse_json(log_s)
| where tostring(Audit.objectRef.subresource) in ("exec", "portforward", "attach")
| where tostring(Audit.verb) == "create"
| where toint(Audit.responseStatus.code) == 101  // Switching Protocols (successful)
| extend
    User = tostring(Audit.user.username),
    PodName = tostring(Audit.objectRef.name),
    Namespace = tostring(Audit.objectRef.namespace),
    Subresource = tostring(Audit.objectRef.subresource),
    SourceIP = tostring(Audit.sourceIPs[0])
| project
    TimeGenerated,
    User,
    Namespace,
    PodName,
    Subresource,
    SourceIP
| extend
    AlertTitle = strcat("Container Access: ", User, " executed ", Subresource, " on ",
Namespace, "/", PodName),
    Severity = case(
        Namespace in ("kube-system", "production"), "High",
        "Medium"
    )
```

5. Privileged container creation:

```
// Detect creation of containers with elevated privileges
AzureDiagnostics
| where Category == "kube-audit"
| where TimeGenerated > ago(5m)
| extend Audit = parse_json(log_s)
| where tostring(Audit.objectRef.resource) == "pods"
| where tostring(Audit.verb) == "create"
| extend RequestObj = parse_json(tostring(Audit.requestObject))
| mv-expand Container = RequestObj.spec.containers
| extend SecurityContext = Container.securityContext
| where
    SecurityContext.privileged == true
```

```
    or SecurityContext.capabilities.add has "SYS_ADMIN"
    or SecurityContext.capabilities.add has "NET_ADMIN"
    or Container.image startswith "/"  // hostPath mount
| project
    TimeGenerated,
    User = tostring(Audit.user.username),
    Namespace = tostring(Audit.objectRef.namespace),
    PodName = tostring(Audit.objectRef.name),
    ContainerName = tostring(Container.name),
    Image = tostring(Container.image),
    Privileged = SecurityContext.privileged,
    Capabilities = SecurityContext.capabilities,
    SourceIP = tostring(Audit.sourceIPs[0])
| extend
    AlertTitle = strcat("Privileged Container: ", User, " created privileged pod ",
Namespace, "/", PodName),
    Severity = "High"
```

6. Admission controller bypasses:

```
// Detect modifications to admission controllers
AzureDiagnostics
| where Category == "kube-audit"
| where TimeGenerated > ago(5m)
| extend Audit = parse_json(log_s)
| where tostring(Audit.objectRef.resource) in ("mutatingwebhookconfigurations",
"validatingwebhookconfigurations")
| where tostring(Audit.verb) in ("update", "patch", "delete")
| extend
    User = tostring(Audit.user.username),
    WebhookName = tostring(Audit.objectRef.name),
    Action = tostring(Audit.verb)
| project
    TimeGenerated,
    User,
    WebhookName,
    Action,
    SourceIP = tostring(Audit.sourceIPs[0])
| extend
    AlertTitle = strcat("Security Control Modified: ", User, " ", Action, " admission
webhook ", WebhookName),
    Severity = "Critical"
```

7. Suspicious API usage patterns:

```
// Detect unusual API call patterns (reconnaissance)
AzureDiagnostics
| where Category == "kube-audit"
| where TimeGenerated > ago(10m)
```

```
| extend Audit = parse_json(log_s)
| extend
    User = tostring(Audit.user.username),
    Verb = tostring(Audit.verb),
    Resource = tostring(Audit.objectRef.resource)
| where Verb in ("list", "get")  // Read operations
| summarize
    UniqueResources = dcount(Resource),
    Resources = make_set(Resource),
    RequestCount = count()
    by User, bin(TimeGenerated, 5m)
| where UniqueResources > 15  // Accessed >15 different resource types
| extend
    AlertTitle = strcat("API Reconnaissance: ", User, " queried ", UniqueResources, "
resource types"),
    Severity = "Medium"
```

8. After-hours activity:

```
// Detect administrative actions outside business hours
let BusinessHours = dynamic(["09", "10", "11", "12", "13", "14", "15", "16", "17"]);
AzureDiagnostics
| where Category == "kube-audit"
| where TimeGenerated > ago(1h)
| extend Audit = parse_json(log_s)
| extend Hour = format_datetime(TimeGenerated, "HH")
| where Hour !in (BusinessHours)
| where tostring(Audit.verb) in ("create", "update", "patch", "delete")
| where tostring(Audit.objectRef.resource) in ("secrets", "rolebindings",
"clusterrolebindings", "pods")
| extend
    User = tostring(Audit.user.username),
    Resource = tostring(Audit.objectRef.resource),
    Action = tostring(Audit.verb),
    SourceIP = tostring(Audit.sourceIPs[0])
| where User !startswith "system:"  // Exclude system accounts
| project
    TimeGenerated,
    Hour,
    User,
    Resource,
    Action,
    SourceIP
| extend
    AlertTitle = strcat("After-Hours Activity: ", User, " ", Action, " ", Resource, "
at ", Hour, ":00"),
    Severity = "Medium"
```

9. Anonymous or unauthenticated access:

```
// Detect anonymous API access attempts
AzureDiagnostics
| where Category == "kube-audit"
| where TimeGenerated > ago(5m)
| extend Audit = parse_json(log_s)
| extend User = tostring(Audit.user.username)
| where User in ("system:anonymous", "system:unauthenticated")
| where toint(Audit.responseStatus.code) == 200  // Successful
| extend
    Resource = tostring(Audit.objectRef.resource),
    Verb = tostring(Audit.verb),
    SourceIP = tostring(Audit.sourceIPs[0])
| project
    TimeGenerated,
    User,
    Resource,
    Verb,
    SourceIP
| extend
    AlertTitle = "Anonymous API Access Allowed",
    Severity = "High"
```

10. ServiceAccount token theft detection:

```
// Detect ServiceAccount tokens used from unexpected locations
let KnownPodIPs = (
    // Build table of known pod IPs (from Container Insights)
    KubePodInventory
    | where TimeGenerated > ago(1h)
    | summarize by PodIp, ServiceName
);
AzureDiagnostics
| where Category == "kube-audit"
| where TimeGenerated > ago(5m)
| extend Audit = parse_json(log_s)
| extend User = tostring(Audit.user.username)
| where User startswith "system:serviceaccount:"
| extend SourceIP = tostring(Audit.sourceIPs[0])
| join kind=leftanti (KnownPodIPs) on $left.SourceIP == $right.PodIp
| where SourceIP !startswith "10."  // Not from cluster network
| project
    TimeGenerated,
    ServiceAccount = User,
    SourceIP,
    Resource = tostring(Audit.objectRef.resource),
    Verb = tostring(Audit.verb)
| extend
    AlertTitle = strcat("ServiceAccount Token Misuse: ", ServiceAccount, " used from
external IP ", SourceIP),
```

```
    Severity = "Critical"
```

Best practices for detection rules:

- Tune thresholds based on baseline (reduce false positives)
- Include context in alerts (user, IP, resource, action)
- Severity based on risk (secrets > configmaps)
- Exclude expected system activity (kube-controller-manager, kubelet)
- Alert aggregation (don't spam on every event)
- Include remediation guidance in alerts
- Regular review and tuning of rules
- Test rules against historical data
- Document rule logic and expected false positive rate

These detection rules transform raw audit logs into actionable security intelligence, enabling rapid threat detection and response in Kubernetes environments.