# **Auditing**

In a Kubernetes cluster, audit logs are very useful for tracing and tracking activities and changes to different cluster resources. By enabling auditing (since it’s not enabled by default), you will be able to know who did what, and when.

The cluster audits the activities generated by users, by applications that use the Kubernetes API, and by the control plane itself, auditing allows cluster administrators to answer the following questions:

* what happened?
* when did it happen?
* who initiated it?
* on what did it happen?
* where was it observed?
* from where was it initiated?
* to where was it going?

When a request comes to the api server, and the lifecycle of the request through the API server depending on the time in that whole lifecycle you can decide when you want to log things. Each request can be recorded with an associated *stage*. The defined stages are:

* RequestReceived - The stage for events generated as soon as the audit handler receives the request, and before it is delegated down the handler chain.
* ResponseStarted - Once the response headers are sent, but before the response body is sent. This stage is only generated for long-running requests (e.g. watch).
* ResponseComplete - The response body has been completed and no more bytes will be sent.
* Panic - Events generated when a panic occurred.

The audit logging feature increases the memory consumption of the API server because some context required for auditing is stored for each request. Memory consumption depends on the audit logging configuration.

What should be recorded? You can specify the level of logging based on the following criteria. This is defined in the audit policy configuration file.

* None - don't log events that match this rule.
* Metadata - log request metadata (requesting user, timestamp, resource, verb, etc.) but not request or response body.
* Request - log event metadata and request body but not response body. This does not apply for non-resource requests.
* RequestResponse - log event metadata, request and response bodies. This does not apply for non-resource requests.

## Audit policy

## Audit policy defines rules about what events should be recorded and what data they should include. The audit policy object structure is defined in the [audit.k8s.io API group](https://kubernetes.io/docs/reference/config-api/apiserver-audit.v1/#audit-k8s-io-v1-Policy). When an event is processed, it's compared against the list of rules in order

You can pass a file with the policy to kube-apiserver using the --audit-policy-file flag. If the flag is omitted, no events are logged. Note that the rules field **must** be provided in the audit policy file. A policy with no (0) rules is treated as illegal.

Below is an example audit policy file:

**apiVersion**: audit.k8s.io/v1 *# This is required.*

**kind**: Policy

*# Don't generate audit events for all requests in RequestReceived stage.*

**omitStages**:

- "RequestReceived"

**rules**:

*# Log pod changes at RequestResponse level*

- **level**: RequestResponse

**resources**:

- **group**: ""

*# Resource "pods" doesn't match requests to any subresource of pods,*

*# which is consistent with the RBAC policy.*

**resources**: ["pods"]

*# Log "pods/log", "pods/status" at Metadata level*

- **level**: Metadata

**resources**:

- **group**: ""

**resources**: ["pods/log", "pods/status"]

*# Don't log requests to a configmap called "controller-leader"*

- **level**: None

**resources**:

- **group**: ""

**resources**: ["configmaps"]

**resourceNames**: ["controller-leader"]

*# Don't log watch requests by the "system:kube-proxy" on endpoints or services*

- **level**: None

**users**: ["system:kube-proxy"]

**verbs**: ["watch"]

**resources**:

- **group**: "" *# core API group*

**resources**: ["endpoints", "services"]

*# Don't log authenticated requests to certain non-resource URL paths.*

- **level**: None

**userGroups**: ["system:authenticated"]

**nonResourceURLs**:

- "/api\*" *# Wildcard matching.*

- "/version"

*# Log the request body of configmap changes in kube-system.*

- **level**: Request

**resources**:

- **group**: "" *# core API group*

**resources**: ["configmaps"]

*# This rule only applies to resources in the "kube-system" namespace.*

*# The empty string "" can be used to select non-namespaced resources.*

**namespaces**: ["kube-system"]

*# Log configmap and secret changes in all other namespaces at the Metadata level.*

- **level**: Metadata

**resources**:

- **group**: "" *# core API group*

**resources**: ["secrets", "configmaps"]

*# Log all other resources in core and extensions at the Request level.*

- **level**: Request

**resources**:

- **group**: "" *# core API group*

- **group**: "extensions" *# Version of group should NOT be included.*

*# A catch-all rule to log all other requests at the Metadata level.*

- **level**: Metadata

*# Long-running requests like watches that fall under this rule will not*

*# generate an audit event in RequestReceived.*

**omitStages**:

- "RequestReceived"

You can use a minimal audit policy file to log all requests at the Metadata level:

*# Log all requests at the Metadata level.*

**apiVersion**: audit.k8s.io/v1

**kind**: Policy

**rules**:

- **level**: Metadata

## Audit backends

Audit backends persist audit events to an external storage. Out of the box, the kube-apiserver provides two backends:

* Log backend, which writes events into the filesystem
* Webhook backend, which sends events to an external HTTP API

### **Log backend**

The log backend writes audit events to a file in [JSONlines](https://jsonlines.org/) format. You can configure the log audit backend using the following kube-apiserver flags:

* --audit-log-path specifies the log file path that log backend uses to write audit events. Not specifying this flag disables log backend. - means standard out
* --audit-log-maxage defined the maximum number of days to retain old audit log files
* --audit-log-maxbackup defines the maximum number of audit log files to retain
* --audit-log-maxsize defines the maximum size in megabytes of the audit log file before it gets rotated

If your cluster's control plane runs the kube-apiserver as a Pod, remember to mount the hostPath to the location of the policy file and log file, so that audit records are persisted. For example:

--audit-policy-file=/etc/kubernetes/audit-policy.yaml **\**

--audit-log-path=/var/log/kubernetes/audit/audit.log

then mount the volumes:

**...**

**volumeMounts**:

- **mountPath**: /etc/kubernetes/audit-policy.yaml

**name**: audit

**readOnly**: **true**

- **mountPath**: /var/log/kubernetes/audit/

**name**: audit-log

**readOnly**: **false**

and finally configure the hostPath:

**...**

**volumes**:

- **name**: audit

**hostPath**:

**path**: /etc/kubernetes/audit-policy.yaml

**type**: File

- **name**: audit-log

**hostPath**:

**path**: /var/log/kubernetes/audit/

**type**: DirectoryOrCreate

### **Webhook backend**

The webhook audit backend sends audit events to a remote web API, which is assumed to be a form of the Kubernetes API, including means of authentication. You can configure a webhook audit backend using the following kube-apiserver flags:

* --audit-webhook-config-file specifies the path to a file with a webhook configuration. The webhook configuration is effectively a specialized [kubeconfig](https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters).
* --audit-webhook-initial-backoff specifies the amount of time to wait after the first failed request before retrying. Subsequent requests are retried with exponential backoff.

The webhook config file uses the kubeconfig format to specify the remote address of the service and credentials used to connect to it.

### **Event batching**

So let's go a little deeper we figured out that these logs go to a back-end, now we can also give more details or we can also tell k8s how we want all of this information to go that's defined by batching.

Both log and webhook backends support batching. Using webhook as an example, here's the list of available flags. To get the same flag for log backend, replace webhook with log in the flag name. By default, batching is enabled in webhook and disabled in log.

* --audit-webhook-mode defines the buffering strategy. One of the following:
  + batch - buffer events and asynchronously process them in batches. This is the default.
  + blocking - block API server responses on processing each individual event.
  + blocking-strict - Same as blocking, but when there is a failure during audit logging at the RequestReceived stage, the whole request to the kube-apiserver fails.

The following flags are used only in the batch mode:

* --audit-webhook-batch-buffer-size defines the number of events to buffer before batching. If the rate of incoming events overflows the buffer, events are dropped.
* --audit-webhook-batch-max-size defines the maximum number of events in one batch.
* --audit-webhook-batch-max-wait defines the maximum amount of time to wait before unconditionally batching events in the queue.
* --audit-webhook-batch-throttle-qps defines the maximum average number of batches generated per second.
* --audit-webhook-batch-throttle-burst defines the maximum number of batches generated at the same moment if the allowed QPS was underutilized previously.