Introduction

This document introduces Kubernetes and the available logging options. This document is provided “as is”. It is highly recommended to practice before trying these instructions in production!

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services that facilitates declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

Kubernetes Overview

Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure no downtime. For example, if a container goes down, another container must start. It is easier if a system handled this behaviour.

Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of scaling and failover for your application, provides deployment patterns, and more.

For example: Kubernetes can easily manage a canary deployment for your system. A canary deployment, or canary release, is a deployment pattern that allows you to roll out new code/features to a subset of users as an initial test.

Kubernetes provides you with:

- **Service discovery and load balancing** Kubernetes can expose a container using the DNS name or its own IP address. If traffic to a container is high, Kubernetes can load balance and distribute the network traffic so that the deployment is stable.
- **Storage orchestration** Kubernetes allows you to automatically mount a storage system of your choice, such as local storage, public cloud providers, and more.
- **Automated rollouts and rollbacks** You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.
- **Automatic bin packing** You provide Kubernetes with a cluster of nodes that it can use to run containerized tasks. You tell Kubernetes how much CPU and memory (RAM) each container needs. Kubernetes can fit containers onto your nodes to make the best use of your resources.
- **Self-healing** Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.
- **Secret and configuration management** Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.
- **Batch execution** In addition to services, Kubernetes can manage your batch and CI workloads, replacing containers that fail, if desired.
- **Horizontal scaling** Scale your application up and down with a simple command, with a UI, or automatically based on CPU usage.
- **IPv4/IPv6 dual-stack** Allocation of IPv4 and IPv6 addresses to Pods and Services
- **Designed for extensibility** Add features to your Kubernetes cluster without changing upstream source code.

Kubernetes Log Types

In Kubernetes clusters, logging is crucial for monitoring, debugging, and maintaining the health of applications and the cluster itself. There are several types of logs you might encounter, each serving different purposes:

Container Logs

- **Application Logs**: Generated by the applications running inside your containers. These logs can provide insights into the application's behavior, errors, and performance metrics.
- **Standard Output and Error (stdout/stderr)**: Kubernetes captures the output from the standard output and standard error streams of the containers. These logs are accessible via kubectl logs <pod-name> and are often used for debugging and monitoring.

Kubernetes System Logs

- **Kubelet Logs**: Logs from the Kubelet, which is the agent running on each worker node. It manages container runtime, node status, and communication with the Kubernetes control plane.
- **Kube-apiserver Logs**: Logs from the Kubernetes API server, which is responsible for handling API requests from clients and maintaining the cluster state.
- **Kube-controller-manager Logs**: Logs from the controller manager, which runs controllers responsible for managing the state of the cluster, such as replication controllers and deployments.
- **Kube-scheduler Logs**: Logs from the scheduler, which is responsible for assigning Pods to worker nodes based on resource availability and other constraints.
- **etcd Logs**: Logs from /etcd, the key-value store used by Kubernetes to store all cluster data and configuration.

Infrastructure Logs

- **Node Logs**: System logs from the underlying nodes (operating system and hardware logs). These logs can provide insights into issues at the node level.
- **Networking Logs**: Logs related to networking components like CNI (Container Network Interface) plugins or network policies, which are important for diagnosing network issues.

Audit Logs

- **Kubernetes Audit Logs**: Record all API requests made to the Kubernetes API server. These logs provide details about what actions were taken, by whom, and when, and are useful for security auditing and troubleshooting.

Cluster-level Logs

- **Control Plane Logs**: Collectively refers to logs from the Kubernetes API server, controller manager, and scheduler. These logs are critical for understanding the operations and health of the Kubernetes control plane.
- **Scheduler Logs**: Provides information on scheduling decisions made by the Kubernetes scheduler.

Custom and Sidecar Logs

- **Sidecar Containers**: Containers running alongside your main application containers within the same Pod. They often handle logging or monitoring tasks and generate their own logs.
- **Custom Logs**: Logs generated by custom logging agents or tools that you deploy within your cluster for specific purposes, such as log aggregation or processing.

Kubernetes Audit Logs

Kubernetes audit logs help maintain security, compliance, and operational integrity within a Kubernetes cluster. They provide a detailed record of all actions and events, offering insight into who accessed the system, what actions were performed, and when they occurred.

While Kubernetes monitoring tools focus on performance and resource management, audit logs primarily serve security, compliance, and operational integrity objectives. By leveraging a Kubernetes audit log, administrators can detect and investigate security breaches, troubleshoot issues, track configuration changes, and ensure adherence to regulatory requirements — ultimately enhancing the overall reliability and trustworthiness of the Kubernetes environment.

Security and Compliance Considerations

The Kubernetes ecosystem is a dynamic one. Containerized applications are constantly being deployed, scaled, and updated. Kubernetes audit logs play a pivotal role in meeting various security and compliance requirements such as:

- Health Insurance Portability and Accountability Act (HIPAA)
- Payment Card Industry Data Security Standard (PCI DSS)
- General Data Protection Regulation (GDPR)
- National Institute of Standards and Technology (NIST) Cybersecurity Framework

How Kubernetes Audit Logs Help You Meet These Standards

Here are examples of how they contribute to meeting compliance standards.

- **Visibility and Accountability:** Kubernetes audit logs provide visibility into system activities, enabling organizations to track user actions, resource accesses, and configuration changes. This transparency fosters accountability and ensures adherence to security policies and regulatory requirements.
- **Forensic Analysis and Incident Response:** In the event of a security breach or compliance violation, Kubernetes audit logs serve as a vital source of information for forensic analysis and incident response. By following the sequence of events recorded in the logs, security teams can identify the root cause of incidents, assess the extent of damage, and take appropriate remedial actions.
- **Continuous Monitoring and Alerting:** Automated monitoring of Kubernetes audit logs allows you to detect suspicious activities and anomalies in real time. By configuring alerts based on predefined security policies and thresholds, deviations from normal behavior can be promptly identified, investigated, and mitigated. This serves to greatly reduce the risk of security breaches and compliance failures.
- **Evidence for Audits and Compliance Reporting:** During audits and compliance assessments, Kubernetes audit logs serve as concrete evidence of security controls implementation and adherence to regulatory requirements. Presenting comprehensive and well-maintained audit trails demonstrate your commitment to maintaining a secure and compliant Kubernetes environment.

Understanding Kubernetes Audit Logs

Kubernetes audit logs capture a comprehensive record of all activities and events occurring within the cluster, including user actions, API requests, and configuration changes. These logs are invaluable for security, compliance, and troubleshooting purposes, enabling administrators to track changes, detect unauthorized access or actions, investigate incidents, and maintain regulatory compliance.

Events Captured in Kubernetes Audit Logs

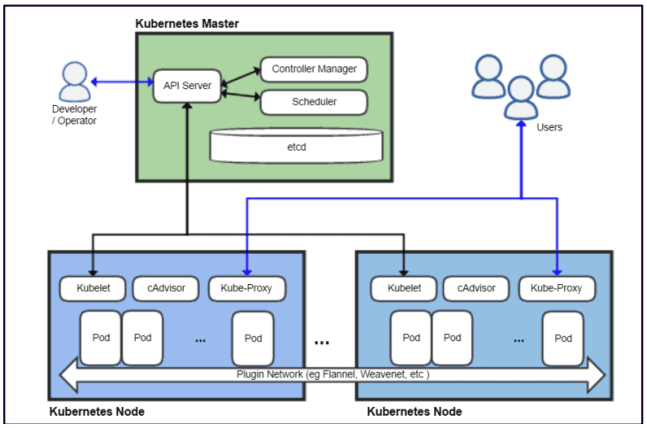
Kubernetes audit logs offer administrators valuable insights into the activities occurring within their clusters, facilitating security monitoring, compliance enforcement, troubleshooting, and incident response efforts.

Various types of events are captured to provide a comprehensive record of activities within the cluster. Some common types of events include:

- **Resource Access:** These events track actions related to accessing Kubernetes resources such as pods, services, configmaps, secrets, and persistent volumes. Examples include requests to create, read, update, or delete these resources.
- **Authentication and Authorization:** Events related to user authentication and authorization are logged, including successful and failed login attempts, token authentication, role-based access control (RBAC) decisions, and authorization failures.
- **API Server Requests:** Kubernetes API server requests are documented, including API calls to list resources, watch for changes, create or update objects, and perform operations like scaling deployments or updating configurations.
- **Policy Enforcement:** Events related to the enforcement of network policies, pod security policies, and other Kubernetes security policies are captured. This includes actions taken by admission controllers to validate and enforce policies.
- **Cluster-Level Operations:** Events involving cluster-wide operations such as node management, pod scheduling, namespace creation, role binding updates, and cluster role assignments are recorded to provide visibility into administrative activities.
- **Audit Configuration Changes:** Changes to the audit logging configuration itself, such as modifications to audit policy rules, log storage settings, or audit sink configurations, are also tracked to clarify changes to the auditing setup.

Kubernetes Architecture

Kubernetes cluster consists of a single node or multiple nodes. In Kubernetes, if you need to run a single container, the container will be created inside a pod. A pod can have a single container or multiple containers.



Kubernetes Cluster Auditing

Control plane manages the worker nodes, and the worker nodes host the pods. The API server is a component of the Kubernetes control plane that exposes the Kubernetes API. The core component of the control plane is the kube-apiserver. The kube-apiserver is a REST based front end service for all the external users, nodes, and services to interact with each other. Every command executed in the Kubernetes environment will be processed by the kube-apiserver. The Kubernetes API server is responsible for authentication and authorization.

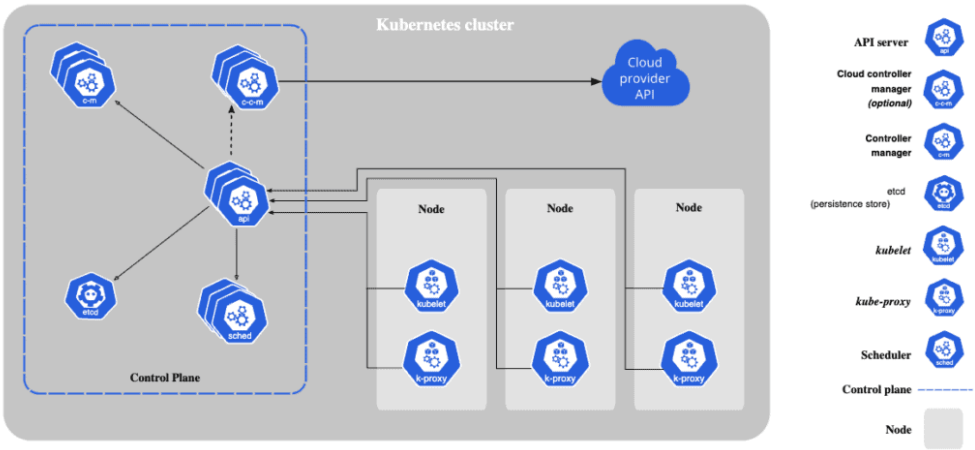


Figure 3: Source: https://kubernetes.io/docs/concepts/overview/components/

Kubernetes API Audit Logs

Kubernetes API records all the API requests made to kube-apiserver by the users and the Kubernetes internal services as well. The audit log provides a lot of information such as source IP address, time of the request, user info, and request and response information. Audit logs trace abnormalities in the cluster environment such as failed login attempts or any malicious activity.

Audit Logging Stages

The kube-apiserver allows us to capture the logs at various stages of a request sent to it. This includes the events at the metadata stage, request, and response bodies as well. Kubernetes allows us to define the stages which we intend to capture. The following are the allowed stages in the Kubernetes audit logging framework:

- **RequestReceived**: As the name suggests, this stage captures the generated events as soon as the audit handler receives the request.
- **ResponseStarted**: In this stage, collects the events once the response headers are sent, but just before the response body is sent.
- **ResponseComplete**: This stage collects the events after the response body is sent completely.
- **Panic**: Events collected whenever the API server panics.

There are lots of calls made to the API server, and we need a mechanism to filter out the events based on our requirements. Kubernetes auditing provides yet another feature for this very reason — the level field in the policy configuration.

Audit Log Levels

The level field in the rules list defines what properties of an event are recorded. An important aspect of audit logging in Kubernetes is, whenever an event is processed *it is matched against the rules defined in the config file in order.* The first rule sets the audit level of logging the event. Kubernetes provides the following audit levels while defining the audit configuration.

- **None**: This disables logging of any event that matches the rule.
- **Metadata**: Logs request metadata (requesting user/userGroup, timestamp, resource/subresource, verb, status, etc.) but not request or response bodies.
- **Request**: This level records the event metadata and request body but does not log the response body.
- **RequestResponse**: This level is more verbose than the other levels since this level logs the Metadata, request, and response bodies.