Logging Architecture

Application logs can help you understand what is happening inside your application. The logs are particularly useful for debugging problems and monitoring cluster activity.

Most modern applications have some kind of logging mechanism. Likewise, container engines are designed to support logging.

The easiest and most adopted logging method for containerized applications is writing to standard output and standard error streams.

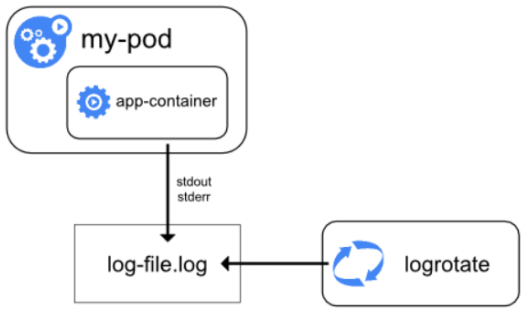
However, the native functionality provided by a container engine or runtime is usually not enough for a complete logging solution.For example, you may want to access your application's logs if a container crashes, a pod gets evicted, or a node dies.

In a cluster, logs should have a separate storage and lifecycle independent of nodes, pods, or containers. This concept is called cluster-level logging.

Cluster-level logging architectures require a separate backend to store, analyze, and query logs. Kubernetes does not provide a native storage solution for log data. Instead, there are many logging solutions that integrate with Kubernetes.

How nodes handle container logs

A container runtime handles and redirects any output generated to a containerized application's `stdout` and `stderr` streams.



Different container runtimes implement this in different ways; however, the integration with the kubelet is standardized as the *CRI logging format*.By default, if a container restarts, the kubelet keeps one terminated container with its logs. If a pod is evicted from the node, all corresponding containers are also evicted, along with their logs.
The kubelet makes logs available to clients via a special feature of the Kubernetes API. The usual way to access this is by running `kubectl logs`.

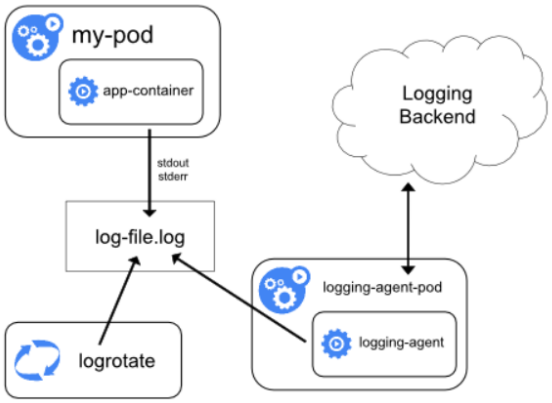
You can configure the kubelet to rotate logs automatically.If you configure rotation, the kubelet is responsible for rotating container logs and managing the logging directory structure. The kubelet sends this information to the container runtime (using CRI), and the runtime writes the container logs to the given location.

You can configure two kubelet configuration settings, containerLogMaxSize and containerLogMaxFiles, using the kubelet configuration file. These settings let you configure the maximum size for each log file and the maximum number of files allowed for each container respectively.
When you run kubectl logs the kubelet on the node handles the request and reads directly from the log file. The kubelet returns the content of the log file.

Only the contents of the latest log file are available through kubectl logs. For example, if a Pod writes 40 MiB of logs and the kubelet rotates logs after 10 MiB, running kubectl logs returns at most 10MiB of data.

Cluster-level logging architectures

a.Using a node logging agent



You can implement cluster-level logging by including a node-level logging agent on each node. The logging agent is a dedicated tool that exposes logs or pushes logs to a backend. Commonly, the logging agent is a container that has access to a directory with log files from all of the application containers on that node.

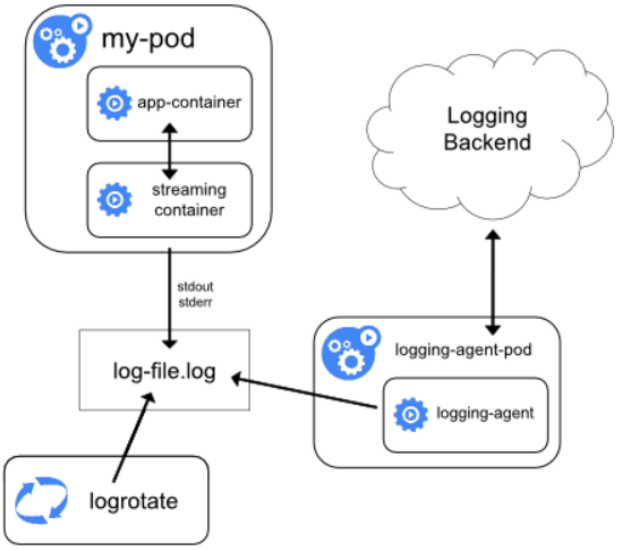
Because the logging agent must run on every node, it is recommended to run the agent as a DaemonSet. Node-level logging creates only one agent per node and doesn't require any changes to the applications running on the node.

Containers write to stdout and stderr, but with no agreed format. A node-level agent collects these logs and forwards them for aggregation.

b.Using a sidecar container with the logging agent

You can use a sidecar container in one of the following ways:

Streaming sidecar container

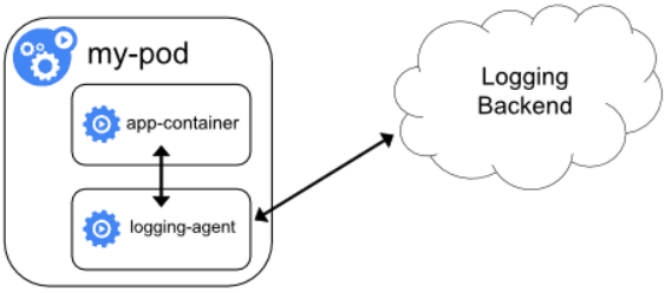


Here sidecar containers write to their own stdout and stderr streams. In this approach you can take advantage of the kubelet and the logging agent that already run on each node

The sidecar containers read logs from a file, a socket, or journald. Each sidecar container prints a log to its own stdout or stderr stream.

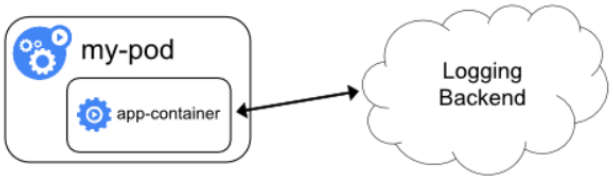
This approach allows you to separate several log streams from different parts of your application, some of which can lack support for writing to stdout or stderr. The logic behind redirecting logs is minimal, so it's not a significant overhead. Additionally, because stdout and stderr are handled by the kubelet, you can use built-in tools like kubectl logs.

Sidecar container with a logging agent



If the node-level logging agent is not flexible enough for your situation, you can create a sidecar container with a separate logging agent that you have configured specifically to run with your application.
Using a logging agent in a sidecar container can lead to significant resource consumption. Moreover, you won't be able to access those logs using kubectl logs because they are not controlled by the kubelet.

c.Exposing logs directly from the application



Cluster-logging that exposes or pushes logs directly from every application is outside the scope of Kubernetes.