This guide explains, in depth, how to access and retain logs produced by Airflow components and Spark jobs running on Amazon EKS **without relying on the Airflow Web UI**. You will learn:

* Where every relevant log lives (pod stdout, kubelet, node, control plane)
* How to retrieve logs interactively and non-interactively (kubectl, kubectl cp, kubectl exec)
* How to deploy logging DaemonSets (Fluent Bit/Promtail) to collect and forward logs to CloudWatch, S3, OpenSearch or Loki
* How to capture Spark driver & executor logs reliably for later debugging (even when pods terminate)
* How to correlate logs across systems (pod -> node -> cloud provider logs)

Target audience: platform engineers, SREs, DevOps, and data engineers who run Airflow + Spark on EKS.

**High-level logging architecture on EKS**

1. **Application stdout/stderr:** Best practice: write logs to stdout/stderr so the container runtime forwards them to node-level files. These are what kubectl logs reads.
2. **Node-level logs:** The container runtime (containerd/docker) writes per-pod container logs to files (commonly /var/log/containers or journald). Kubelet also writes logs and events.
3. **DaemonSet collectors:** Fluent Bit, Fluentd, or Promtail run as DaemonSets and tail container log files or read journald, adding Kubernetes metadata and forwarding logs to chosen sinks.
4. **Sinks / storages:** CloudWatch Logs, OpenSearch, S3, Grafana Loki. These are where logs are stored long-term and searched.
5. **Cloud provider logs:** CloudTrail, EKS control plane logs, and VPC Flow Logs provide additional telemetry about API calls and network flows.

**Where logs originate (sources) — exhaustive list**

* **Airflow components** (if deployed in k8s):
  + Scheduler pod(s) — scheduling decisions, DAG parsing, scheduler heartbeats.
  + Webserver — limited runtime logs (but UI not available in our scenario).
  + Workers / K8sExecutor pods / Celery workers — DAG task stdout/stderr.
  + Triggerer, Flower, metadata DB sidecars.
* **Spark-on-Kubernetes components:**
  + Spark driver pod — main program logs.
  + Spark executor pods — task-level logs.
  + Spark history server (if deployed) — event logs and UI.
* **Kubernetes control plane & node logs:**
  + kubelet logs, container runtime logs (containerd), systemd/journal logs, kernel logs.
  + Kubernetes events (resource creation/failure). kubectl describe pod is helpful.
* **Cluster-level services:**
  + CNI plugin logs (aws-vpc-cni), kube-proxy, CoreDNS.
* **Cloud provider logs:**
  + CloudWatch (if enabled), CloudTrail logs, EKS control plane logs, VPC Flow Logs.

**Kubernetes-native ways to read logs**

**Basic kubectl usage**

* Get pods in namespace:

kubectl get pods -n airflow

* Tail logs of a pod (default container):

kubectl logs -n <ns> <pod-name> -f

* Tail a specific container in a pod:

kubectl logs -n <ns> <pod-name> -c <container-name> -f

* Show previous (crashed) container logs:

kubectl logs -n <ns> <pod-name> -c <container-name> --previous

* Since and tailing:

kubectl logs -n <ns> <pod> --since=2h --tail=500

**Find pods created by a DAG run or job**

Many people annotate pods with labels. Example commands to find pods for a run:

kubectl get pods -n airflow -l "dag\_id=my\_dag" --sort-by=.metadata.creationTimestamp

kubectl get pods -n airflow -l "owner=myuser,dag\_id=my\_dag"

If no labels exist, use kubectl get pods -n airflow and inspect START and AGE plus kubectl describe pod to link to a DAG run.

**kubectl describe pod — events and reasons**

kubectl describe pod -n <ns> <pod-name>

This shows recent events (image pull failures, OOMKilled, CrashLoopBackOff) — essential when a pod terminates quickly and logs disappear.

**Multi-pod tailing tools (more ergonomic)**

* **stern** — tail logs from multiple pods matching a label selector, adds pod/container prefix.
* **kubetail** — similar to stern but implemented in bash.

Example stern usage:

stern -n airflow "my-dag-.\*" --since 10m

**Node and kubelet logs (host-level)**

When a pod is scheduled, the kubelet on the node manages the container and writes logs. Access node logs via SSH (if you have access) or via CloudWatch (if you configured node log shipping).

Typical paths:

* container logs: /var/log/containers/\*.log
* pod logs: /var/log/pods/<pod-uid>/\*
* kubelet: journal (journalctl -u kubelet) or /var/log/messages//var/log/syslog depending on AMI.

Examples:

# If you can SSH into node

sudo journalctl -u kubelet -n 500

sudo ls -l /var/log/containers | tail

sudo cat /var/log/containers/<pod>-<container>-<namespace>.log

If nodes are managed and you cannot SSH, ensure DaemonSet uploading node logs to CloudWatch or S3.

**Container runtime and file locations**

Kubernetes typically relies on the container runtime to write container logs:

* containerd: logs under /var/log/containers and /var/log/pods or in journald depending on distro.
* older Docker setups: /var/lib/docker/containers/<container-id>/\*.log.

DaemonSets should be configured to read from /var/log/containers and enrich entries with labels.

**Application logs (Airflow & Spark) — patterns and useful configuration**

**Airflow best practices**

* Configure Airflow to emit structured logs to stdout (JSON preferable). In airflow.cfg or via Helm values, set logging to console:
  + enable\_task\_log\_to\_stdout = True (or equivalent setting depending on chart/Helm/operator)
* Ensure workers don't write important logs only to ephemeral disk inside pods — instead forward to stdout.

**Spark on Kubernetes**

To retain Spark logs even after pods die, configure Spark's event logging and a persistent event log store:

* spark.eventLog.enabled=true
* spark.eventLog.dir=s3a://my-bucket/spark-events/ (or an HDFS/MinIO path)

Also set driver/executor log forwarding:

* Set spark.driver.log.dfsDir or spark.kubernetes.driverEnv variables to forward logs to S3.
* Use the Spark History Server pointed at the same spark.eventLog.dir for UI and historical troubleshooting.

**Example:** pass --conf spark.eventLog.dir=s3a://my-bucket/spark-events when submitting.

Capturing full stdout/stderr for driver and executors is done by: 1) kubectl logs, 2) Collectors that tail /var/log/containers, 3) configuring Spark to write application logs to S3.

**Deploying log collection: Fluent Bit / Fluentd / Promtail — examples**

**Fluent Bit (lightweight) — DaemonSet outline**

Fluent Bit runs on every node and tails /var/log/containers/\*.log. It enriches logs with Kubernetes metadata through the kubernetes filter and then forwards to sinks.

Minimal Fluent Bit YAML (snippet):

*apiVersion: v1*

*kind: ConfigMap*

*metadata:*

*name: fluent-bit-config*

*namespace: logging*

*data:*

*fluent-bit.conf: |*

*[SERVICE]*

*Flush 1*

*Daemon Off*

*Log\_Level info*

*[INPUT]*

*Name tail*

*Path /var/log/containers/\*.log*

*Parser docker*

*Tag kube.\**

*[FILTER]*

*Name kubernetes*

*Match kube.\**

*Kube\_URL https://kubernetes.default.svc:443*

*[OUTPUT]*

*Name cloudwatch*

*Match kube.\**

*region us-east-1*

Then run Fluent Bit as a DaemonSet mounting /var/log and /var/lib/docker/containers (if needed) and the Kubernetes service account token.

**Promtail + Loki**

Promtail scrapes logs and labels them for Loki (Grafana). Promtail uses a pipeline\_stages config to parse and extract fields from logs (timestamps, log level, trace ids).

**Enrichment and parsing**

* Use the kubernetes filter to pull namespace, pod, container, labels and annotations into each record.
* Use parsers for structured log formats (JSON, nginx, syslog). If Airflow emits JSON logs, pick a JSON parser to index fields in the sink.

**Shipping logs to destinations (details)**

**CloudWatch Logs (AWS)**

* Fluent Bit output plugin available to send logs to CloudWatch Logs. Configure log group names like /eks/<cluster>/<namespace>/<pod>.
* Advantages: integrated with AWS IAM and CloudWatch Logs Insights for queries.
* Cost: pay per ingestion & storage.

Example Fluent Bit output config to CloudWatch (snippet):

*[OUTPUT]*

*Name cloudwatch*

*Match kube.\**

*region us-east-1*

*log\_group\_name eks-cluster-logs*

*log\_stream\_prefix from-fluent-bit-*

**S3 (archival)**

* Use Fluent Bit s3 plugin or write to an intermediate (Kinesis/Firehose) to deliver to S3 in compressed, partitioned files for long retention and audit.
* Good for cheap long-term retention and compliance.

**OpenSearch / Elasticsearch**

* Use Fluent Bit/Fluentd output to OpenSearch. Create indices by namespace and date. Useful when you need full-text search and dashboards (Kibana/OpenSearch Dashboards).

**Loki (Grafana)**

* Good for cost-efficient log storage with label-based queries. Works well with Promtail.

**Searching and filtering logs — examples**

**CloudWatch Logs Insights example**

Query for errors in Airflow worker logs (CloudWatch Insights syntax):

*fields @timestamp, @message*

*| filter @message like /ERROR/ or @message like /Traceb*