Centralized Container Logging with Fluent Bit

by Michael Hausenblas | on 09 JUL 2019 | in [Amazon Athena](https://aws.amazon.com/blogs/opensource/category/analytics/amazon-athena/), [Amazon CloudWatch](https://aws.amazon.com/blogs/opensource/category/management-tools/amazon-cloudwatch/), [Amazon EC2 Container Registry](https://aws.amazon.com/blogs/opensource/category/compute/amazon-ec2-container-registry/), [Amazon Elastic Container Service](https://aws.amazon.com/blogs/opensource/category/compute/amazon-elastic-container-service/), [Amazon Elastic Kubernetes Service](https://aws.amazon.com/blogs/opensource/category/compute/amazon-kubernetes-service/), [Amazon Kinesis](https://aws.amazon.com/blogs/opensource/category/analytics/amazon-kinesis/), [Amazon Simple Storage Service (S3)](https://aws.amazon.com/blogs/opensource/category/storage/amazon-simple-storage-services-s3/), [Open Source](https://aws.amazon.com/blogs/opensource/category/open-source/) | [Permalink](https://aws.amazon.com/blogs/opensource/centralized-container-logging-fluent-bit/) | [Comments](https://aws.amazon.com/blogs/opensource/centralized-container-logging-fluent-bit/#Comments) | [Share](https://aws.amazon.com/blogs/opensource/centralized-container-logging-fluent-bit/)

*September 8, 2021: Amazon Elasticsearch Service has been renamed to Amazon OpenSearch Service.*[*Visit the website to learn more*](https://aws.amazon.com/opensearch-service/)*.*

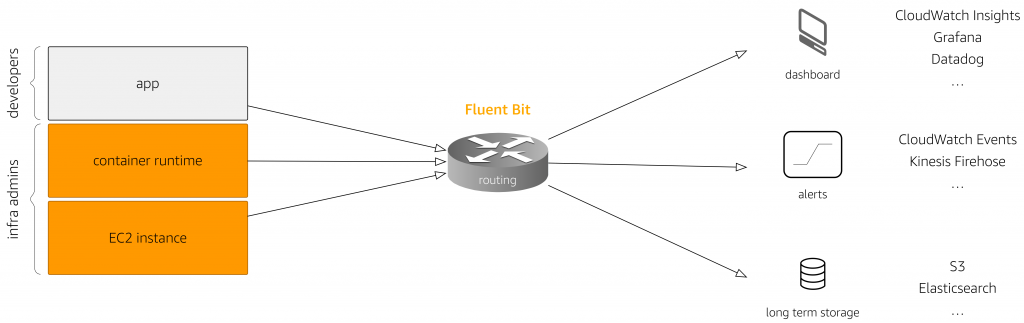
*by Wesley Pettit and Michael Hausenblas*

AWS is built for builders. Builders are always looking for ways to optimize, and this applies to application logging. Not all logs are of equal importance. Some require real-time analytics, others simply need to be stored long-term so that they can be analyzed if needed. It’s therefore critical to be able to easily route your logs to a wide variety of tools for storage and analytics provided by AWS and its partners.

That’s why we are supporting [Fluent Bit](https://fluentbit.io/) to help create an easy extension point for streaming logs from containerized applications to AWS’ and partners’ solutions for log retention and analytics. With the newly-launched Fluent Bit plugin for AWS container image, you can route logs to [Amazon CloudWatch](http://aws.amazon.com/cloudwatch) and [Amazon Kinesis Data Firehose](https://aws.amazon.com/kinesis/data-firehose/) destinations (which include Amazon S3, [Amazon Elasticsearch Service](http://aws.amazon.com/elasticsearch-service), and [Amazon Redshift](http://aws.amazon.com/redshift)). In this post we will show you the Fluent Bit plugin in action on both Amazon ECS and EKS clusters. You might also want to check out the tutorial on the [basics of Fluentd and the Kinesis Firehose](https://aws.amazon.com/blogs/compute/building-a-scalable-log-solution-aggregator-with-aws-fargate-fluentd-and-amazon-kinesis-data-firehose/), if you’re not familiar with the tooling itself, as well as review the relevant issues in the [AWS containers roadmap](https://github.com/aws/containers-roadmap/projects/1), especially [#10](https://github.com/aws/containers-roadmap/issues/10) and [#66](https://github.com/aws/containers-roadmap/issues/66).

**Introduction to log routing**

Conceptually, log routing in a containerized setup such as Amazon ECS or EKS looks like this:



On the left-hand side of above diagram, the log sources are depicted (starting at the bottom):

1. The *host and control plane level* is made up of EC2 instances, hosting your containers. These instances may or may not be accessible directly by you. For example, for containers running on Fargate, you will not see instances in your EC2 console. On this level you’d also expect logs originating from the [EKS control plane](https://docs.aws.amazon.com/eks/latest/userguide/control-plane-logs.html), managed by AWS.
2. The *container runtime level*commonly includes logs generated by the Docker engine, such as the [agent logs in ECS](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/logs.html). These logs are usually most useful to people in infrastructure admin roles, but can also assist developers in troubleshooting situations.
3. The *application level* is where the user code runs. This level generates application-specific logs, such as a log entry on the outcome of an operation in your own app, or the app logs from off-the-shelf application components such as NGINX.

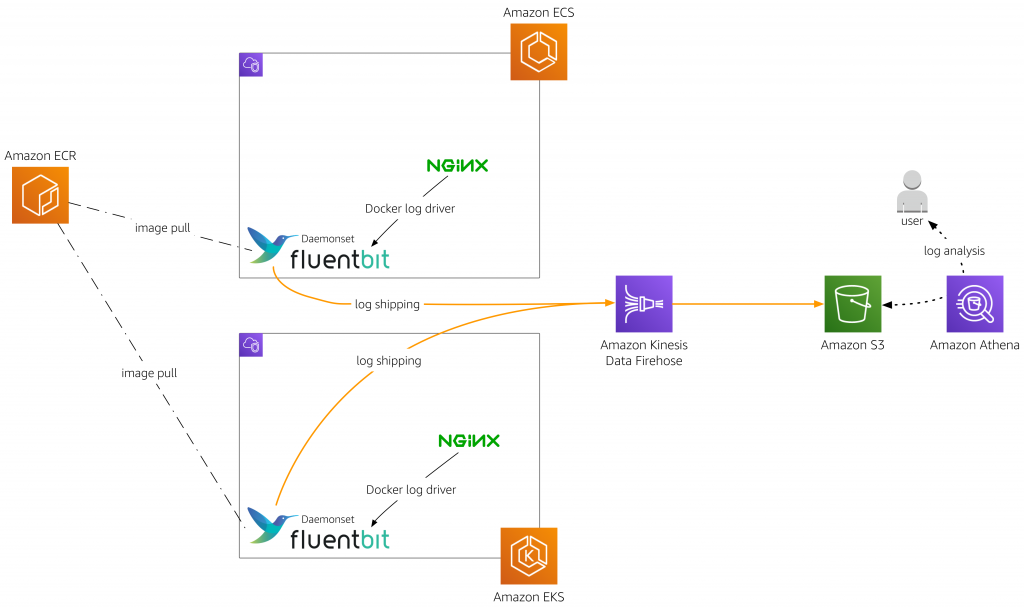
Next comes the routing component: this is Fluent Bit. It takes care of reading logs from all sources and routing log records to various destinations, also known as log sinks. This routing component needs to run somewhere, for example as a sidecar in a Kubernetes pod / ECS task, or as a host-level daemon set.

The downstream log sinks consume logs for different purposes and audiences. These include a number of use cases, from log analysis to compliance (requiring that logs be stored for a given retention period), alerting when a human user needs to be notified of an event, and dashboard logs that provide a collection of (real-time) graphs to help human users absorb the overall state of the system at a glance.

With these basics out of the way, let’s now look at a concrete use case: centralized logging of a multi-cluster app using Fluent Bit. All the container definitions and configurations ace available in the [Amazon ECS Fluent Bit Daemon Service GitHub repo](https://github.com/aws-samples/amazon-ecs-fluent-bit-daemon-service).

**Centralized logging in action: multi-cluster log analysis**

To show Fluent Bit in action, we will perform a multi-cluster log analysis across both an Amazon ECS and an Amazon EKS cluster, with Fluent Bit deployed and configured as daemon sets. The application-level logs generated by NGINX apps running in each cluster is captured by Fluent Bit and streamed via [Amazon Kinesis Data Firehose](https://docs.aws.amazon.com/firehose/latest/dev/what-is-this-service.html) to Amazon S3, where we can query them using [Amazon Athena](https://aws.amazon.com/athena/):



**Setup for Amazon ECS**

[Create an ECS on EC2](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-cli-tutorial-ec2.html) cluster with the following user data—in our case, in a file called enable-fluent-log-driver.sh ([source](https://github.com/aws-samples/amazon-ecs-fluent-bit-daemon-service/blob/master/ecs/enable-fluent-log-driver.sh))—to enable the Fluentd log driver in the ECS agent:

#!/bin/bash

echo "ECS\_AVAILABLE\_LOGGING\_DRIVERS=[\"awslogs\",\"fluentd\"]" >> /etc/ecs/ecs.config

Bash

For example, we created the ECS on EC2 cluster like so; this step assumes that you have the [ECS CLI](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ECS_CLI.html) installed:

$ ecs-cli up \

--size 2 \

--instance-type t2.medium \

--extra-user-data enable-fluent-log-driver.sh \

--keypair fluent-bit-demo-key \

--capability-iam \

--cluster-config fluent-bit-demo

Bash

Next, we need to build a container image containing the Fluent Bit configuration. We’ll do that by creating a Dockerfile ([source](https://github.com/aws-samples/amazon-ecs-fluent-bit-daemon-service/blob/master/ecs/Dockerfile)) with the following content:

FROM amazon/aws-for-fluent-bit:1.2.0

ADD fluent-bit.conf /fluent-bit/etc/

ADD parsers.conf /fluent-bit/etc/

NOTE Counter to good security practice, the USER is not defined, making it run as root. This is intentionally done so, because Fluent Bit currently requires to run as root.

The above Dockerfile in turn depends on two configuration files:

* the fluent-bit.conf file ([source](https://github.com/aws-samples/amazon-ecs-fluent-bit-daemon-service/blob/master/ecs/fluent-bit.conf)) defining the routing to the Firehose delivery stream, and
* the parsers.conf file ([source](https://github.com/aws-samples/amazon-ecs-fluent-bit-daemon-service/blob/master/ecs/parsers.conf)), defining the NGINX log parsing.

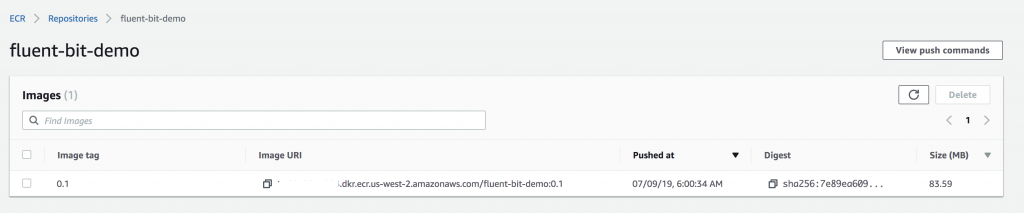
Now, we’ll build our custom container image and push it to an ECR repository called fluent-bit-demo:

$ docker build --tag fluent-bit-demo:0.1 .

$ ecs-cli push fluent-bit-demo:0.1

Bash

Verify that your custom log routing image build and push was successful by visiting the ECR console; you should see something like this:



We’re now in a position to launch an [ECS service with daemon scheduling strategy](https://github.com/aws-samples/amazon-ecs-fluent-bit-daemon-service/blob/master/ecs/ecs-fluent-bit-daemonset.yml) to deploy our custom-configured Fluent Bit into our cluster, using the above container image:

$ aws cloudformation deploy \

--template-file ecs-fluent-bit-daemonset.yml \

--stack-name ecs-fluent-bit-daemon-service \

--parameter-overrides \

EnvironmentName=fluentbit-daemon-service \

DockerImage=XXXXXXXXXXXX.dkr.ecr.us-west-2.amazonaws.com/fluent-bit-demo:0.1 \

Cluster=fluent-bit-demo \

--region $(aws configure get region) \

--capabilities CAPABILITY\_NAMED\_IAM

Bash

In the ECS console you should now see something like this:

A screenshot of a computer

Description automatically generated

Now we can launch an ECS service, running NGINX, based on following [task definition](https://github.com/aws-samples/amazon-ecs-fluent-bit-daemon-service/blob/master/ecs/nginx-task-definition.json):

{

"taskDefinition": {

"taskDefinitionArn": "arn:aws:ecs:us-west-2:XXXXXXXXXXXX:task-definition/nginx:1",

"containerDefinitions": [

{

"name": "nginx",

"image": "nginx:1.17",

"memory": 100,

"essential": true,

"portMappings": [

{

"hostPort": 80,

"protocol": "tcp",

"containerPort": 80

}

],

"logConfiguration": {

"logDriver": "fluentd",

"options": {

"fluentd-address": "unix:///var/run/fluent.sock",

"tag": "logs-from-nginx"

}

}

}

],

"family": "nginx"

}

}

JSON

After creating the above task definition, you should now see the following in your ECS console:

A screenshot of a computer

Description automatically generated

And now we can launch the ECS service based on above task definition:

$ aws ecs create-service \

--cluster fluent-bit-demo \

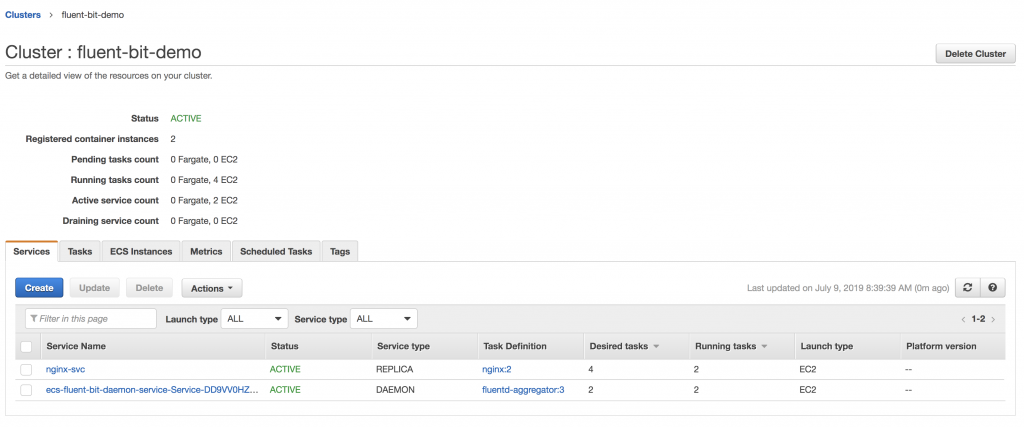
--service-name nginx-svc \

--task-definition nginx:1 \

--desired-count 1

Bash

If everything worked out, you should see something like the following in the ECS console:



With this, we’ve set up the ECS part. Now we configure the same setup on our Kubernetes cluster running on Amazon EKS.

**Setup for Amazon EKS**

Create an Amazon EKS cluster named fluent-bit-demo using eksctl, as shown in the [EKS docs](https://docs.aws.amazon.com/eks/latest/userguide/eksctl.html), and then create a policy file called eks-fluent-bit-daemonset-policy.json ([source](https://github.com/aws-samples/amazon-ecs-fluent-bit-daemon-service/blob/master/eks/eks-fluent-bit-daemonset-policy.json)) with the following content:

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"firehose:PutRecordBatch"

],

"Resource": "\*"

},

{

"Effect": "Allow",

"Action": "logs:PutLogEvents",

"Resource": "arn:aws:logs:\*:\*:log-group:\*:\*:\*"

},

{

"Effect": "Allow",

"Action": [

"logs:CreateLogStream",

"logs:DescribeLogStreams",

"logs:PutLogEvents"

],

"Resource": "arn:aws:logs:\*:\*:log-group:\*"

},

{

"Effect": "Allow",

"Action": "logs:CreateLogGroup",

"Resource": "\*"

}

]

}

JSON

To attach this policy file to the EKS on EC2 worker nodes, execute the following sequence:

$ STACK\_NAME=$(eksctl get nodegroup --cluster fluent-bit-demo -o json | jq -r '.[].StackName')

$ INSTANCE\_PROFILE\_ARN=$(aws cloudformation describe-stacks --stack-name $STACK\_NAME | jq -r '.Stacks[].Outputs[] | select(.OutputKey=="InstanceProfileARN") | .OutputValue')

$ ROLE\_NAME=$(aws cloudformation describe-stacks --stack-name $STACK\_NAME | jq -r '.Stacks[].Outputs[] | select(.OutputKey=="InstanceRoleARN") | .OutputValue' | cut -f2 -d/)

$ aws iam put-role-policy \

--role-name $ROLE\_NAME \

--policy-name FluentBit-DS \

--policy-document file://eks-fluent-bit-daemonset-policy.json

Bash

And now we move on to defining the Kubernetes RBAC settings – that is, the service account the Fluent Bit pods will be using along with the role and role binding.

First create the service account fluent-bit (this is what we will later use in the daemon set) by executing kubectl create sa fluent-bit.

Next, define the role and binding in a file named eks-fluent-bit-daemonset-rbac.yaml([source](https://github.com/aws-samples/amazon-ecs-fluent-bit-daemon-service/blob/master/eks/eks-fluent-bit-daemonset-rbac.yaml)):

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRole

metadata:

name: pod-log-reader

rules:

- apiGroups: [""]

resources:

- namespaces

- pods

verbs: ["get", "list", "watch"]

---

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRoleBinding

metadata:

name: pod-log-crb

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: pod-log-reader

subjects:

- kind: ServiceAccount

name: fluent-bit

namespace: default

YAML

Now, in order to make the access permissions for the Fluent Bit plugin effective, you create the role and role binding, defined above, by executing the command kubectl apply -f eks-fluent-bit-daemonset-rbac.yaml.

In contrast to the ECS case, where we backed the configuration into a custom image, in our Kubernetes setup we’re using a [config map](https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/) to define the log parsing and routing for the Fluent Bit plugin. For this, use a file called eks-fluent-bit-configmap.yaml([source](https://github.com/aws-samples/amazon-ecs-fluent-bit-daemon-service/blob/master/eks/eks-fluent-bit-configmap.yaml)) with the following content:

apiVersion: v1

kind: ConfigMap

metadata:

name: fluent-bit-config

labels:

app.kubernetes.io/name: fluentbit

data:

fluent-bit.conf: |

[SERVICE]

Parsers\_File parsers.conf

[INPUT]

Name tail

Tag kube.\*

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

Parser docker

DB /var/log/flb\_kube.db

Mem\_Buf\_Limit 5MB

Skip\_Long\_Lines On

Refresh\_Interval 10

[FILTER]

Name parser

Match \*\*

Parser nginx

Key\_Name log

[OUTPUT]

Name firehose

Match \*\*

delivery\_stream eks-stream

region us-west-2

parsers.conf: |

[PARSER]

Name nginx

Format regex

Regex ^(?<remote>[^ ]\*) (?<host>[^ ]\*) (?<user>[^ ]\*) \[(?<time>[^\]]\*)\] "(?<method>\S+)(?: +(?<path>[^\"]\*?)(?: +\S\*)?)?" (?<code>[^ ]\*) (?<size>[^ ]\*)(?: "(?<referer>[^\"]\*)" "(?<agent>[^\"]\*)")? \"-\"$

Time\_Key time

Time\_Format %d/%b/%Y:%H:%M:%S %z

YAML

Create this config map by executing the command kubectl apply -f eks-fluent-bit-configmap.yaml and then define the Kubernetes Daemonset (using said config map) in a file called eks-fluent-bit-daemonset.yaml ([source](https://github.com/aws-samples/amazon-ecs-fluent-bit-daemon-service/blob/master/eks/eks-fluent-bit-daemonset.yaml)) with below content:

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: fluentbit

labels:

app.kubernetes.io/name: fluentbit

spec:

selector:

matchLabels:

name: fluentbit

template:

metadata:

labels:

name: fluentbit

spec:

serviceAccountName: fluent-bit

containers:

- name: aws-for-fluent-bit

image: amazon/aws-for-fluent-bit:1.2.0

volumeMounts:

- name: varlog

mountPath: /var/log

- name: varlibdockercontainers

mountPath: /var/lib/docker/containers

readOnly: true

- name: fluent-bit-config

mountPath: /fluent-bit/etc/

- name: mnt

mountPath: /mnt

readOnly: true

resources:

limits:

memory: 500Mi

requests:

cpu: 500m

memory: 100Mi

volumes:

- name: varlog

hostPath:

path: /var/log

- name: varlibdockercontainers

hostPath:

path: /var/lib/docker/containers

- name: fluent-bit-config

configMap:

name: fluent-bit-config

- name: mnt

hostPath:

path: /mnt

YAML

Finally, launch the Fluent Bit daemonset by executing kubectl apply -f eks-fluent-bit-daemonset.yaml and verify the Fluent Bit daemonset by peeking into the logs like so:

$ kubectl logs ds/fluentbit

Found 3 pods, using pod/fluentbit-9zszm

Fluent Bit v1.1.3

Copyright (C) Treasure Data

[2019/07/08 13:44:54] [ info] [storage] initializing...

[2019/07/08 13:44:54] [ info] [storage] in-memory

[2019/07/08 13:44:54] [ info] [storage] normal synchronization mode, checksum disabled

[2019/07/08 13:44:54] [ info] [engine] started (pid=1)

[2019/07/08 13:44:54] [ info] [in\_fw] listening on unix:///var/run/fluent.sock

...

[2019/07/08 13:44:55] [ info] [sp] stream processor started

Bash

Next, deploy the following [NGINX app](https://github.com/aws-samples/amazon-ecs-fluent-bit-daemon-service/blob/master/eks/eks-nginx-app.yaml) via kubectl apply -f eks-nginx-app.yaml:

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx

labels:

app.kubernetes.io/name: nginx

spec:

replicas: 4

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.17

ports:

- containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

name: nginx

labels:

app: nginx

spec:

ports:

- port: 80

targetPort: 80

selector:

app: nginx

YAML

With that, we’re done setting up the log sources and routing. Now let’s move on to actually doing something with all the log data we’re collecting from the NGINX containers running in ECS and EKS: we will perform a centralized analysis of the logs.

**Log analysis across clusters**

The goal is to do a log analysis of the NGINX containers running in the ECS and EKS clusters. For this, we’re using [Amazon Athena](http://aws.amazon.com/athena), which allows us to interactively query the service log data from Amazon S3 using SQL. Before we can query the data in S3, however, we need to get the log data there.

Remember that in the Fluent Bit configurations for ECS and EKS (above) we set the output to delivery\_stream xxx-stream. That’s an Amazon [Kinesis Firehose](https://docs.aws.amazon.com/firehose/latest/dev/what-is-this-service.html) delivery stream, and we first have to create it, for ECS and EKS.

First, set up the access control part by defining a policy that effective allows Firehose to write to S3. To do this, we need to create a new IAM Role with two policy files. First, firehose-policy.json([source](https://github.com/aws-samples/amazon-ecs-fluent-bit-daemon-service/blob/master/log-analysis/firehose-policy.json)):

{

"Version": "2012-10-17",

"Statement": {

"Effect": "Allow",

"Principal": {

"Service": "firehose.amazonaws.com"

},

"Action": "sts:AssumeRole"

}

}

JSON

Second, in the [firehose-delivery-policy.json](https://gist.github.com/mhausenblas/3e533e493dd1b6851c7a29748bc9611e)policy file ([source](https://github.com/aws-samples/amazon-ecs-fluent-bit-daemon-service/blob/master/log-analysis/firehose-delivery-policy.json)), replace the XXXXXXXXXXXX with your own account ID (if you’re unsure what it is, you can get the account ID  by executing aws sts get-caller-identity --output text --query 'Account'). Also, in the S3 section, replace mh9-firelens-demo with your own bucket name.

Now we can create the firehose\_delivery\_role to use for both the ECS and the EKS delivery streams:

$ aws iam create-role \

--role-name firehose\_delivery\_role \

--assume-role-policy-document file://firehose-policy.json

Bash

From the resulting JSON output of the above command, note down the role ARN, which will be something in the form of arn:aws:iam::XXXXXXXXXXXXX:role/firehose\_delivery\_role. We will use this soon to create the delivery stream, but before that can happen we have to put in place the policy defined in the firehose-delivery-policy.json:

$ aws iam put-role-policy \

--role-name firehose\_delivery\_role \

--policy-name firehose-fluentbit-s3-streaming \

--policy-document file://firehose-delivery-policy.json

Bash

Now create the ECS delivery stream:

$ aws firehose create-delivery-stream \

--delivery-stream-name ecs-stream \

--delivery-stream-type DirectPut \

--s3-destination-configuration \

RoleARN=arn:aws:iam::XXXXXXXXXXXX:role/example\_firehose\_delivery\_role,\

BucketARN="arn:aws:s3:::mh9-firelens-demo",\

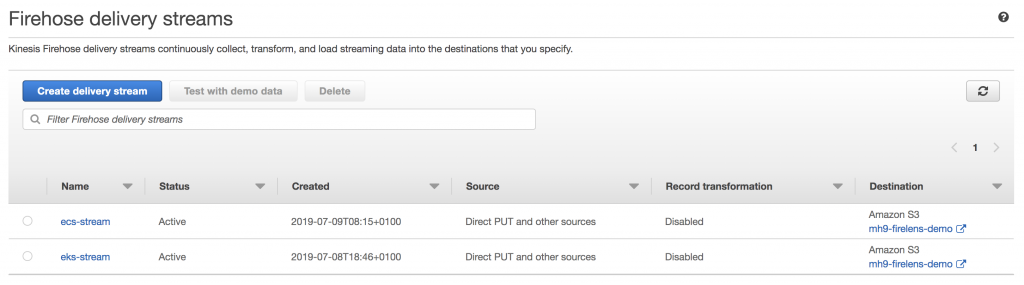
Prefix=ecs

Bash

NOTE The spacing in above command matters: RoleARN etc. must be on one line without spaces.

Now we have to repeat the above for the EKS delivery stream, re-using the role created in the first step. (In other words, you only need to repeat the aws firehose create-delivery-stream command replacing ecs-stream with eks-streamand Prefix=ecs with Prefix=eks.)

It will take a couple of minutes for the delivery streams to be created and active. When you see something like the following, you’re ready to move on to the next step:



We now need to generate some load for the NGINX containers running in ECS and EKS. You can grab the load generator files for [ECS](https://github.com/aws-samples/amazon-ecs-fluent-bit-daemon-service/blob/master/ecs/load-gen-ecs.sh) and [EKS](https://github.com/aws-samples/amazon-ecs-fluent-bit-daemon-service/blob/master/eks/load-gen-eks.sh) and execute the commands below; this will curl the respective NGINX services every two seconds (executing in the background), until you kill the scripts:

$ ./load-gen-ecs.sh &

$ ./load-gen-eks.sh &

Bash

Now that we have some log data from the NGINX webservers, we can query the log entries in S3 from Athena. For this, we first have to create tables for ECS and EKS, telling Athena about the schema we’re using (here shown for the [ECS log data](https://github.com/aws-samples/amazon-ecs-fluent-bit-daemon-service/blob/master/log-analysis/create-ecs-table.sql)and the same applies for [EKS](https://github.com/aws-samples/amazon-ecs-fluent-bit-daemon-service/blob/master/log-analysis/create-eks-table.sql)):

CREATE EXTERNAL TABLE fluentbit\_ecs (

agent string,

code string,

host string,

method string,

path string,

referer string,

remote string,

size string,

user string

)

ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'

LOCATION 's3://mh9-firelens-demo/ecs2019/'

SQL

NOTE Amazon Athena does not import or ingest data; it queries the data directly in S3. So, as log data arrives from the NGINX containers via Fluent Bit and the Firehose delivery stream in the S3 bucket, it is available for you to query using Athena.

Next create a [consolidated view](https://github.com/aws-samples/amazon-ecs-fluent-bit-daemon-service/blob/master/log-analysis/create-consolidated-view.sql) of both the ECS and EKS log entries with the following SQL statement:

CREATE OR REPLACE VIEW "fluentbit\_consolidated" AS

SELECT \* , 'ECS' as source

FROM fluentbit\_ecs

UNION

SELECT \* , 'EKS' as source

FROM fluentbit\_eks

SQL

This allows us to merge the two tables (using the same schema) and add an additional column that flags the source, ECS or EKS. We can now perform a SQL query to figure out who the [top 10 users](https://github.com/aws-samples/amazon-ecs-fluent-bit-daemon-service/blob/master/log-analysis/select-top-10.sql) of our NGINX services are, across the two clusters:

SELECT source,

remote AS IP,

count(remote) AS num\_requests

FROM fluentbit\_consolidated

GROUP BY remote, source

ORDER BY num\_requests DESC LIMIT 10

SQL

This yields something like the following result:

A screenshot of a computer

Description automatically generated

That’s it! You’ve successfully set up the Fluent Bit plugin and used it across two different managed AWS container environments (ECS and EKS) to perform log analytics.

When you’re done, don’t forget to delete the respective workloads, including the Kubernetes NGINX service (which in turn removes the load balancer), and tear down the EKS and ECS clusters, destroying the containers with it. Last but not least, you will want to clean up the Kinesis delivery streams and the S3 bucket with the log data.

Looking ahead, we are also working on a feature to further simplify installing and configuring fluent bit plugins on AWS Fargate, Amazon ECS, and Amazon EKS. You can follow this feature via the [Issue 10](https://github.com/aws/containers-roadmap/issues/10) of our AWS container roadmap.

**Notes on performance and next steps**

To get a better feeling for the performance, we performed a benchmarking test to compare the above Fluent Bit plugin with the Fluentd [CloudWatch](https://github.com/fluent-plugins-nursery/fluent-plugin-cloudwatch-logs) and [Kinesis Firehose](https://github.com/awslabs/aws-fluent-plugin-kinesis) plugins. All our tests were performed on a [c5.9xlarge EC2 instance.](https://aws.amazon.com/ec2/instance-types/c5/) Here are the results:

CloudWatch Plugins: Fluentd vs Fluent Bit

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Log Lines Per second | Data Out | Fluentd CPU | Fluent Bit CPU | Fluentd Memory | Fluent Bit Memory |
| 100 | 25 KB/s | 0.013 vCPU | 0.003 vCPU | 146 MB | 27 MB |
| 1000 | 250 KB/s | 0.103 vCPU | 0.03 vCPU | 303 MB | 44 MB |
| 10000 | 2.5 MB/s | 1.03 vCPU | 0.19 vCPU | 376 MB | 65 MB |

Our tests show that the Fluent Bit plugin is more resource-efficient than Fluentd. On average, Fluentd uses over four times the CPU and six times the memory of the Fluent Bit plugin.

Kinesis Firehose Plugins: Fluentd vs Fluent Bit

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Log Lines Per second | Data Out | Fluentd CPU | Fluent Bit CPU | Fluentd Memory | Fluent Bit Memory |
| 100 | 25 KB/s | 0.006 vCPU | 0.003 vCPU | 84 MB | 27 MB |
| 1000 | 250 KB/s | 0.073 vCPU | 0.033 vCPU | 102 MB | 37 MB |
| 10000 | 2.5 MB/s | 0.86 vCPU | 0.13 vCPU | 438 MB | 55 MB |

In this benchmark, on average Fluentd uses over three times the CPU and four times the memory than the Fluent Bit plugin consumes. Keep in mind that this data does not represent a guarantee; your footprint may differ. However, the above data points suggest that the Fluent Bit plugin is significantly more efficient than Fluentd.

**Next Steps**

We’re excited for you to try this out on your own clusters. Let us know if something doesn’t work the way you expect, and also please share your insights on performance/footprint as well as use cases. Please leave comments on [the issue in GitHub](https://github.com/aws/containers-roadmap/issues/10), or open an issue on the [AWS containers roadmap](https://github.com/aws/containers-roadmap/projects/1) on GitHub.



Wesley Pettit

Software developer in the AWS container service team.

TAGS: [containers](https://aws.amazon.com/blogs/opensource/tag/containers/), [fluent-bit](https://aws.amazon.com/blogs/opensource/tag/fluent-bit/), [fluentd](https://aws.amazon.com/blogs/opensource/tag/fluentd/), [kubernetes](https://aws.amazon.com/blogs/opensource/tag/kubernetes/), [logging](https://aws.amazon.com/blogs/opensource/tag/logging/), [logs](https://aws.amazon.com/blogs/opensource/tag/logs/), [open source](https://aws.amazon.com/blogs/opensource/tag/open-source/)



Michael Hausenblas

Michael works in the AWS open source observability service team where he is a Solution Engineering Lead and owns the AWS Distro for OpenTelemetry (ADOT) from the product side.