



# Amazon Elasticsearch Service

Fully managed search and analytics service.

---

**Have your front end and monitor it too!**

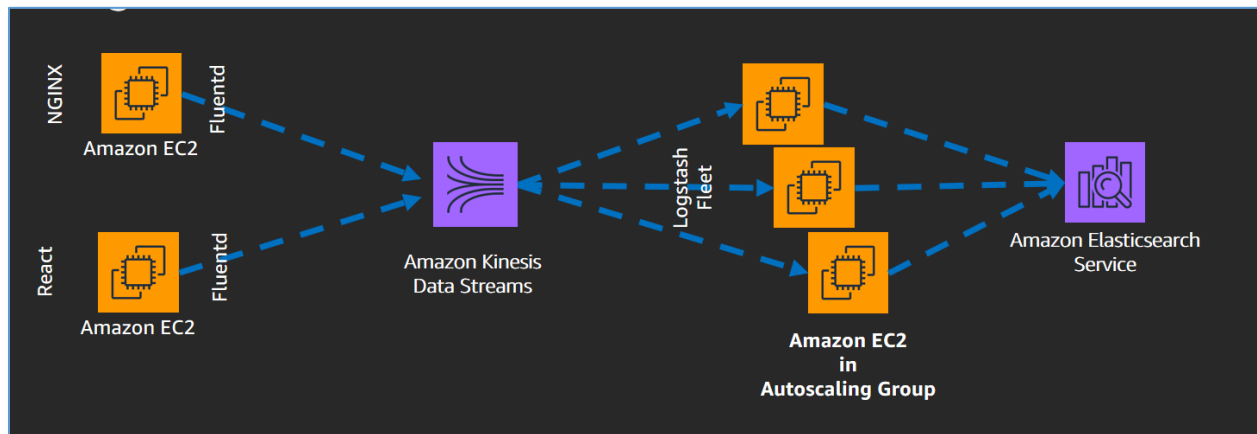
**Lab 2 Instructions**

## Contents

Lab 2 Overview.....	3
Launch the “bootcamp-aes-lab2” nested stack.....	4
Navigate to the AWS CloudFormation console .....	4
Provide the AWS CloudFormation S3 URL .....	5
Populate the needed parameters.....	5
Deploy the stack.....	7
Log consumer aggregation configuration (Logstash).....	8
Logstash configuration.....	8
Review the configuration for Metricbeat for the Logstash process .....	10
Review the configuration for Kinesis for the Logstash process .....	11
Start Logstash and monitor the startup log .....	14
Review the Metricbeat configuration .....	15
Install Metricbeat agent from the preconfigured YUM repo .....	16
Start Metricbeat.....	16
Log producer configuration (Fluentd) – React Apache Server.....	20
Source section.....	22
Filter section .....	23
Explore the ec2_metadata filter .....	23
Explore the record_transformer filter .....	23
Match section .....	24
Start the Fluentd agent (react server) .....	24
Validate you are receiving logs in Kibana .....	26
Log producer configuration (Fluentd) – Kibana Proxy - NGINX Server .....	26
Source section.....	28
Filter section .....	28
Start the Fluentd agent (proxy server).....	28
Validate you are receiving logs in Kibana .....	30

## Lab 2 Overview

In this lab, you configure Fluentd, Metricbeat and Logstash so that you can write events and metrics to your Amazon Elasticsearch Service domain. Events are written directly or indirectly the domain using Amazon Kinesis Data Streams or through direct signed requests over HTTPS to the domain.



Once the logs persist in the domain, you will review the data and the indexes created from your actions.

This lab is an exercise in configuration. Please take the time to understand the configurations. The details contained in this architecture are hard to find in one blog post or vendor documentation. Setting up this type of pipeline is hard and time consuming. It is not a simple task to put this architecture together without understanding why specific components surface in the final implementation. The lab guide discusses the integration topics in detail. Feel free to skip over those details if you time is a concern.

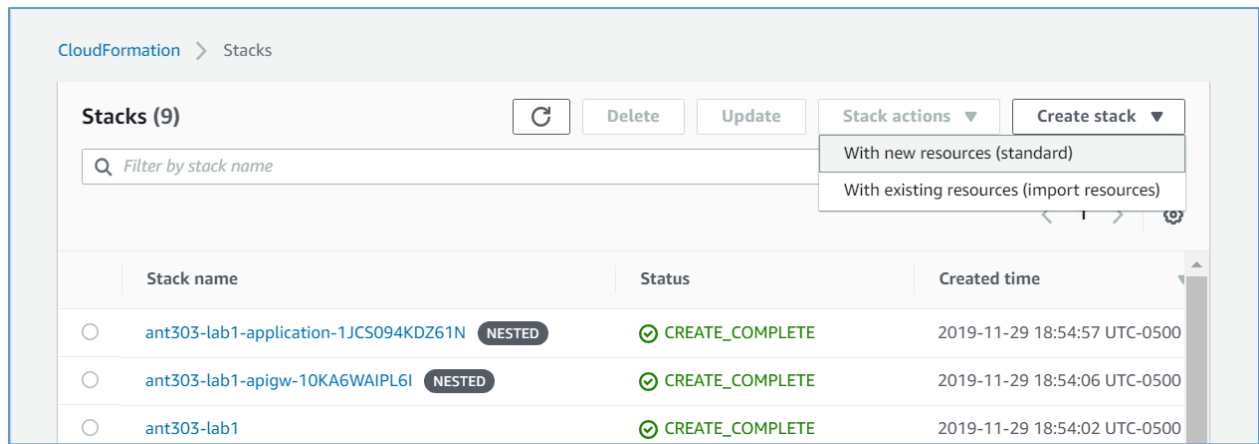
## Launch the “bootcamp-aes-lab2” nested stack

This is the final AWS CloudFormation stack for the bootcamp. This stack creates a fleet of Logstash instances on Amazon EC2. The instances are behind an auto-scaling group. Auto-scaling groups have two benefits:

- 1) They enable you to adjust the Logstash fleet size based on volume of traffic.
- 2) They promote self-healing. If the size of your fleet is set to three and one fails, auto-scaling will replace that instance so that your fleet remains healthy and preserves throughput.

## Navigate to the AWS CloudFormation console

As you have done in the last lab, navigate to the “Create stack” button in the console.



Provide the AWS CloudFormation S3 URL

Paste in the following link (mac users beware PDF documents sometimes do tricky things with URLs that have dashes):

<https://search-sa-log-solutions.s3-us-east-2.amazonaws.com/fluentd-kinesis-logstash/templates/json/bootcamp-aes-lab2>

Make sure all dashes are present. Click the next button when you are done.

**Prepare template**  
Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.

☒ Template is ready ☐ Use a sample template ☐ Create template in Designer

**Specify template**  
A template is a JSON or YAML file that describes your stack's resources and properties.

**Template source**  
Selecting a template generator or Amazon S3 URL where it will be stored

☒ Amazon S3 URL ☐ Upload a template file

Amazon S3 URL  
https://search-sa-log-solutions.s3-us-east-2.amazonaws.com/fluentd-kinesis-logstash/templates/json/bootcamp-aes-lab2

Amazon S3 template URL

S3 URL: Will be generated when URL is provided

View in Designer

Cancel **Next**

Populate the needed parameters

- Stack Name – **ant303-lab2**
- KickoffStackName – **ant303**
- OperatorEmail – your email address (not captured and used for anything other than giving you notifications for instance events via auto scaling).

**Stack name**

Stack name

ant303-lab2

Stack name can include letters (A-Z and a-z), numbers (0-9), and dashes (-).

**Parameters**

Parameters are defined in your template and allow you to input custom values when you create or update a stack.

**KickoffStackName**

Name of an active CloudFormation stack that contains the kickoff stack resources.

ant303

**OperatorEmail**

Email address to notify if there are any scaling operations

kt@b.com

Cancel Previous **Next**

Click next. On the new screen select Stack creation options and disable rollback. Click next.

**▼ Stack creation options**

**Rollback on failure**

Specifies whether the stack should be rolled back if stack creation fails.

☐ Enabled

☒ Disabled

**Timeout**

The number of minutes before a stack creation times out.

Minutes

**Termination protection**

Prevents the stack from being accidentally deleted. Once created, you can update this through stack actions.

☒ Disabled

☐ Enabled

Cancel Previous **Next**

Review the deployment. Select the approval checkboxes for IAM and then click on the "Create stack" button.

## Deploy the stack

### Capabilities

**i** The following resource(s) require capabilities: [AWS::CloudFormation::Stack]

This template contains Identity and Access Management (IAM) resources. Check that you want to create each of these resources and that they have the minimum required permissions. In addition, they have custom names. Check that the custom names are unique within your AWS account. [Learn more](#)

For this template, AWS CloudFormation might require an unrecognized capability: CAPABILITY\_AUTO\_EXPAND. Check the capabilities of these resources.

- ☒ I acknowledge that AWS CloudFormation might create IAM resources with custom names.
- ☒ I acknowledge that AWS CloudFormation might require the following capability: CAPABILITY\_AUTO\_EXPAND

Cancel Previous Create change set **Create stack**

Periodically monitor the deployment. The circle with arrows in the middle of the screen gives you access to refresh the periodic cycle.

CloudFormation > Stacks

Stacks (11)

Filter by stack name

Active View nested

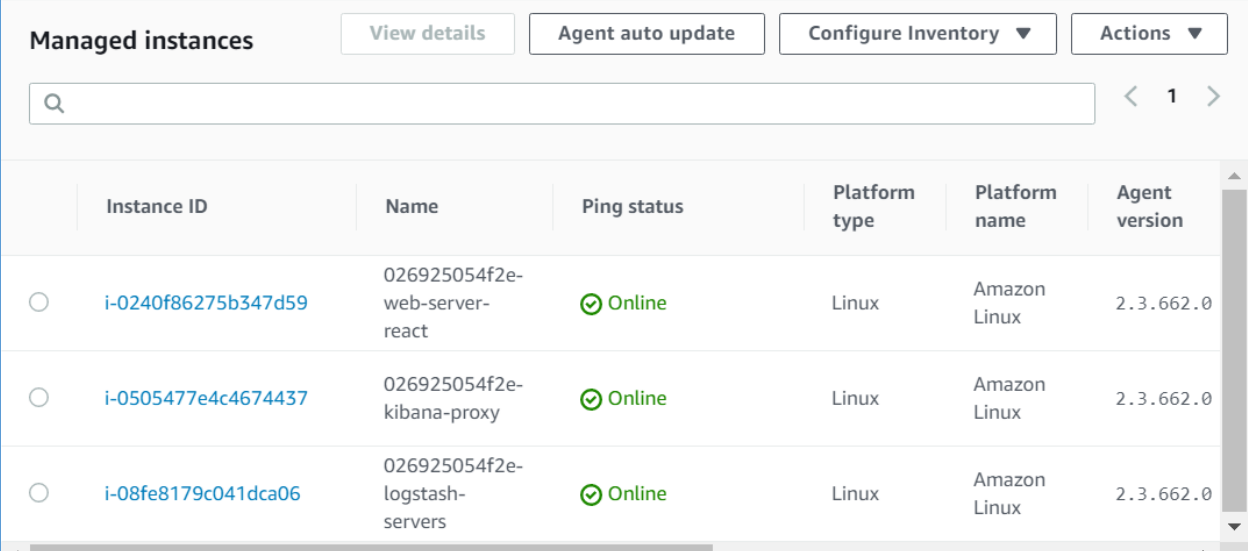
< 1 >

Stack name	Status	Created time
<input type="radio"/> ant303-lab2-logstash-1THK5EH2L9ZKU <b>NESTED</b>	<b>i</b> CREATE_IN_PROGRESS	2019-11-29 23:46:26 UTC-0500
<input type="radio"/> ant303-lab2	<b>i</b> CREATE_IN_PROGRESS	2019-11-29 23:46:21 UTC-0500
<input type="radio"/> ant303-lab1-application-1JCS094KDZ61N <b>NESTED</b>	<b>✓</b> CREATE_COMPLETE	2019-11-29 18:54:57 UTC-0500
<input type="radio"/> ant303-lab1-anjour-10KA6WAIPL6L <b>NESTED</b>	<b>✓</b> CREATE_COMPLETE	2019-11-29 18:54:06 UTC-0500

Once the deployment is complete, you can start configuring the ingestion layer.

## Log consumer aggregation configuration (Logstash)

By now, you can navigate to your AWS Systems Manager console and observe that you have three running instances for your log analytics pipeline. You will see a similar screen.



Managed instances						
<div>View details Agent auto update Configure Inventory Actions</div>						
<div>Q &lt; 1 &gt;</div>						
	Instance ID	Name	Ping status	Platform type	Platform name	Agent version
<input type="radio"/>	<a href="#">i-0240f86275b347d59</a>	026925054f2e-web-server-react	<span>Online</span>	Linux	Amazon Linux	2.3.662.0
<input type="radio"/>	<a href="#">i-0505477e4c4674437</a>	026925054f2e-kibana-proxy	<span>Online</span>	Linux	Amazon Linux	2.3.662.0
<input type="radio"/>	<a href="#">i-08fe8179c041dca06</a>	026925054f2e-logstash-servers	<span>Online</span>	Linux	Amazon Linux	2.3.662.0

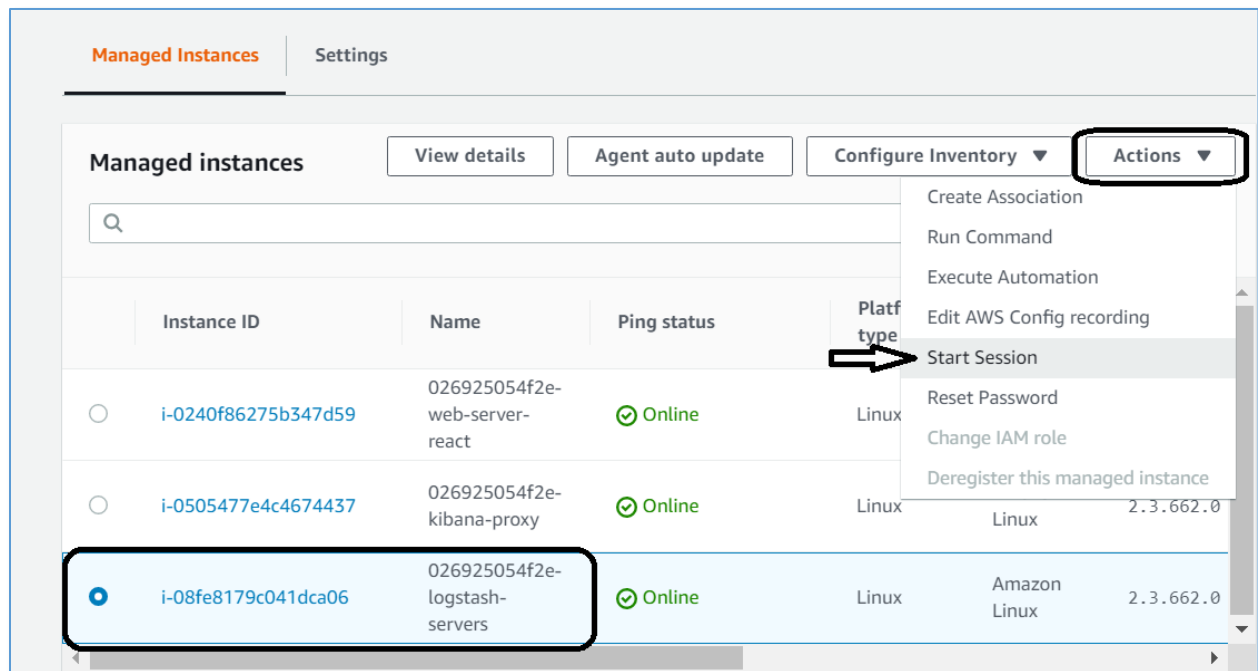
## Logstash configuration

Logstash is built on top of a JVM and it employs a plugin strategy. The plugins address a wide variety of things from data input, filtering, transformation and routing data to a set of destinations. Data sources for Logstash in this workshop are:

- Metricbeat agent – the Logstash server has a configuration to receive events from port 5044. Metricbeat provides system details for the Logstash servers like:
  - CPU utilization
  - File system details
  - Memory
  - Load on the system
  - In addition, more...
- Amazon Kinesis Data Streams – buffer for incoming data. If Amazon ES needs to limit requests, you do not suffer consequences of lost data.
  - Takes in multi-tenant data (more than one application is using the Amazon Elasticsearch Service domain).
  - Data can survive for up to 7 days based on configuration settings.
  - Clients like Logstash are able to parse disparate data formats and route them to the correct index.



In AWS Systems Manager, start a host session for the active Logstash server.



Once the session has launched, switch users to root and execute the following sequence of commands:

- `sudo su - root`
- `cd /etc/logstash`
- `cat pipelines.yml`

Observe similar output:

```
sh-4.2$ sudo su - root
[root@ip-10-1-6-149 ~]# cd /etc/logstash/
[root@ip-10-1-6-149 logstash]# ls -al
total 52
drwxr-xr-x  3 root root  181 Nov 30 04:51 .
drwxr-xr-x 83 root root 8192 Nov 30 12:03 ..
drwxrwxr-x  2 root root   47 Nov 30 04:51 conf.d
-rw-r--r--  1 root root 2019 Sep 27 10:25 jvm.options
-rw-r--r--  1 root root 5043 Sep 27 10:25 log4j2.properties
-rw-r--r--  1 root root  342 Sep 27 10:25 logstash-sample.conf
-rw-r--r--  1 root root 8236 Nov 30 04:50 logstash.yml
-rw-r--r--  1 root root  152 Nov 30 04:51 pipelines.yml
-rw-r--r--  1 root root  285 Sep 27 10:25 pipelines.yml.bak
-rw-----  1 root root 1696 Sep 27 10:25 startup.options
[root@ip-10-1-6-149 logstash]# cat pipelines.yml
- pipeline.id: metricbeat
  path.config: "/etc/logstash/conf.d/metricbeat.cfg"
- pipeline.id: kinesis
  path.config: "/etc/logstash/conf.d/kinesis.cfg"
[root@ip-10-1-6-149 logstash]#
```

Logstash has a function called [pipelines](#). Pipelines give you the ability to define multiple flows for the different log sources in your solution. You have two log sources:

- Metricbeat data – this data comes from the local host so you can monitor performance on the Logstash node to ensure they are configured to performance needs
- Amazon Kinesis Data Streams – this data is on one stream (2 Kinesis shards) holding two different event sources:
  - Access logs from the Kibana proxy server, in NGINX format, enhanced with instance metadata.
  - Access logs from the React web server, in Apache format, enhanced with instance metadata.

Run the next set of commands to review the two pipelines:

- **sudo su - root**
- **cd /etc/logstash**

```
[root@ip-10-1-6-149 logstash]# cd conf.d/
[root@ip-10-1-6-149 conf.d]# ls -al
total 8
drwxrwxr-x 2 root root 47 Nov 30 04:51 .
drwxr-xr-x 3 root root 181 Nov 30 04:51 ..
-rw-r--r-- 1 root root 832 Nov 30 04:51 kinesis.cfg
-rw-r--r-- 1 root root 294 Nov 30 04:51 metricbeat.cfg
[root@ip-10-1-6-149 conf.d]#
```

You will see two configurations, one for each pipeline.

[Review the configuration for Metricbeat for the Logstash process](#)

Run the following command to view the metricbeat.cfg

- **cat metricbeat.cfg**

```
[root@ip-10-1-6-149 conf.d]# cat metricbeat.cfg
input {
  beats {
    port => 5044
  }
}
output {
  amazon_es {
    hosts => ["vpc-es-lab-026925054f2e-5ktmz7pklga74holuc2wjvounq.eu-central-1.es.amazonaws.com"]
    region => ["eu-central-1"]
    index => "%{[@metadata][beat]}-%{[@metadata][version]}-%{+YYYY.MM.dd}"
  }
}
[root@ip-10-1-6-149 conf.d]#
```

As you review the configuration file, you see two sections called:

- input – defines the source
- output – defines the sink

In this case, Metricbeat itself has a configuration file that posts data on port 5044. When Logstash starts up, it will create a listener on port 5044 so that it can receive events from the Metricbeat process. This is the input for the one pipeline.

The output is the Amazon Elasticsearch Service and prefilled with the necessary endpoint details. As you review the configuration, note that the output plugin labeling is “amazon\_es” and not “elasticsearch”. There are two popular plugins found in the Logstash ecosystem when it comes to writing to Elasticsearch. The “amazon\_es” plugin signs requests with Sigv4. The “elasticsearch” plugin does not. This plugin (“amazon\_es”) is specifically required to sign requests to the domain if you use IAM to control access. It reads instance metadata to pull the credential secret key and access key need to sign requests.

In summary, the configuration will read Metricbeat data from a local Metricbeat instance over port 5044. It will receive the events in JSON format and write them to your domain. By using the “beats” ecosystem, the mappings are populated in the for the Metricbeat index automatically when the pipeline deploys.

[Review the configuration for Kinesis for the Logstash process](#)

The Kinesis pipeline is a bit more robust. Two different types of logs come from Kinesis once the end-to-end pipeline is setup. These are:

- Access logs generated by NGINX for the Kibana proxy – they use the common log format but don’t populate same fields as Apache access logs
- Access logs generated by Apache / React serving – they also use the common log format but again, different fields

Fluentd also populates additional fields such as instance id, VPC id, and other AWS specific metadata. The format you receive deviates from the common log format and comes in as a JSON structure.

View the configuration and look at what each section does by using the less command:

- **less kinesis.cfg**

```
[root@ip-10-1-6-149 conf.d]# less kinesis.cfg
```

### *Input section*

The input section uses a Kinesis input plugin for reading events from the Amazon Kinesis Data Stream. This plugin also uses Sigv4 signed requests and uses instance profile metadata to get the secret key and access key for signing. The AWS CloudFormation template for the Logstash instances pre-populate the necessary details for the stream and the region.

```
input {
  kinesis {
    kinesis_stream_name => "log-delivery-026925054f2e"
    type => "kinesis"
    region => "eu-central-1"
    application_name => "026925054f2e"
  }
}
```

### *Filter sections*

The filter sections pull out and work with data.

```
filter {
  json {
    source => "message"
  }
}
filter {
  if ([message] =~ /^.*HealthChecker.*/) {
    drop {}
  }
}
```

The first filter pulls the message from the stream. This message is already in JSON format from the Fluentd agents running on the web and proxy server. Here is an example of each message:

Log from web server:

```
{"host":"44.224.22.196","user":null,"method":"GET","path":"/","code":200,"size":431,"referer":null,"agent":"AWS Security Scanner","hostname":"06c1d1eeaccc-web-servers-filebeat","instance_id":"i-08b69473828395754","instance_type":"t2.large","private_ip":"10.1.5.197","az":"eu-west-1b","vpc_id":"vpc-07e4284eafdd552b2","ami_id":"ami-0ce71448843cb18a1","account_id":"123456789012","event_group":"imdb"}
```

Log from NGINX proxy server:

```
{"remote":"75.67.145.147","host":"-","user":"-","method":"POST","path":"/_plugin/kibana/api/console/proxy?path=_aliases&method=GET","code":"200","size":"585","referer":"https://34.247.200.105/_plugin/kibana/app/kibana","agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64)"}
```

```
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.97
Safari/537.36", "http_x_forwarded_for": "-", "hostname": "06c1d1eeaccc-kibana-
proxy", "instance_id": "i-
01falflaf4c44f81c7", "instance_type": "t2.large", "private_ip": "10.1.0.150", "az":
"eu-west-1a", "vpc_id": "vpc-07e4284eafdd552b2", "ami_id": "ami-
0ce71448843cb18a1", "account_id": "123456789012", "event_group": "proxy"}
```

Both are similar, however, there is enough variance that you can break out the logs into separate indexes to have solid mappings. Both come in as the “message” payload from the stream. The JSON filter pulls out that detail and breaks the JSON document into individual fields.

The next filter looks at the message and sees if there are any health check messages from the Application Load Balancer. A large majority of the logs are health check events. These are not adding any value for our workshop goals and it is much easier to eliminate the documents before they arrive in the domain. This will also cut down on long term costs for the domain because more data = more storage = more cost.

#### *Output section*

The output section does two things.

- It applies a condition expression on the “event\_group” field to route the data to separate indexes. The “event\_group” is populated by Fluentd further up the ingest stream and can be one of “proxy” or “imdb”.
- It then uses the “amazon\_es” plugin to write to the Amazon Elasticsearch Service. This plugin, as mentioned before, signs the requests using Sigv4. It uses the instance profile metadata to get secret key and access key data for the request signing. Note that each section writes to a separate index post fixed with the year, month and day for rolling log files.

```
output {
  if [event_group] == "proxy" {
    amazon_es {
      hosts => ["vpc-es-lab-026925054f2e-5ktmz7pklga74holuc2wjvounq.eu-
central-1.es.amazonaws.com"]
      region => ["eu-central-1"]
      index => "proxy-%{+YYYY.MM.dd}"
    }
  }
  else {
    amazon_es {
      hosts => ["vpc-es-lab-026925054f2e-5ktmz7pklga74holuc2wjvounq.eu-
central-1.es.amazonaws.com"]
      region => ["eu-central-1"]
      index => "webapp-%{+YYYY.MM.dd}"
    }
  }
}
```

Start Logstash and monitor the startup log

Now you can start Logstash and begin consuming the first set of events. Since the Amazon Kinesis stream will not have any data yet, you will only see Metricbeat data in the cluster.

- **service logstash start**
- **less /var/log/logstash/logstash-plain.log**

```
[root@ip-10-1-6-149 conf.d]# service logstash start
Redirecting to /bin/systemctl start logstash.service
[root@ip-10-1-6-149 conf.d]# less /var/log/logstash/logstash-plain.log
```

Page down to the bottom of the file. There should be no errors. You can see that Logstash loaded the configurations.

```
[2019-11-30T14:52:13,737][INFO ][logstash.outputs.elasticsearch][kinesis]
Installing amazon_es template to /_template/logstash
[2019-11-30T14:52:13,743][INFO ][logstash.outputs.elasticsearch][metricbeat]
Installing amazon_es template to /_template/Logstash
```

The next set of events are the different plugins starting listeners and connections. You can see elasticsearch connection detail.

```
[2019-11-30T14:52:13,748][WARN ][logstash.outputs.elasticsearch][kinesis]
Restored connection to ES instance {:url=>"https://vpc-es-lab-026925054f2e-
5ktmz7pklga74holuc2wjvounq.eu-central-1.es.amazonaws.com:443/"}
[2019-11-30T14:52:13,791][INFO ][logstash.outputs.elasticsearch][kinesis] ES
Output version determined {:es_version=>7}
[2019-11-30T14:52:13,792][WARN ][logstash.outputs.elasticsearch][kinesis]
Detected a 6.x and above cluster: the `type` event field won't be used to
determine the document _type {:es_version=>7}
[2019-11-30T14:52:13,812][INFO ][logstash.outputs.elasticsearch][kinesis] New
Elasticsearch output {:class=>"LogStash::Outputs::ElasticSearch",
:hosts=>["//vpc-es-lab-026925054f2e-5ktmz7pklga74holuc2wjvounq.eu-central-
1.es.amazonaws.com"]}
[2019-11-30T14:52:13,813][INFO ][logstash.outputs.elasticsearch][kinesis]
Using mapping template from {:path=>nil}
[2019-11-30T14:52:13,821][INFO ][logstash.outputs.elasticsearch][kinesis]
Attempting to install template {:manage_template=>{"template"=>"logstash-*",
"version"=>60002, "settings"=>{"index.refresh_interval"=>"5s",
"number_of_shards"=>1},
"mappings"=>{"dynamic_templates"=>[{"message_field"=>{"path_match"=>"message"
, "match_mapping_type"=>"string", "mapping"=>{"type"=>"text",
"norms"=>false}}}, {"string_fields"=>{"match"=>"*",
"match_mapping_type"=>"string", "mapping"=>{"type"=>"text", "norms"=>false,
"fields"=>{"keyword"=>{"type"=>"keyword", "ignore_above"=>256}}}}]},
"properties"=>{"@timestamp"=>{"type"=>"date"},
"@version"=>{"type"=>"keyword"}, "geoip"=>{"dynamic"=>true,
"properties"=>{"ip"=>{"type"=>"ip"}, "location"=>{"type"=>"geo_point"},
"latitude"=>{"type"=>"half_float"}, "longitude"=>{"type"=>"half_float"}}}}}}
```

```
[2019-11-30T14:52:13,876][INFO ][logstash.outputs.elasticsearch][kinesis]
Installing amazon_es template to /_template/logstash
[2019-11-30T14:52:14,021][WARN
][org.logstash.instrument.metrics.gauge.LazyDelegatingGauge][kinesis] A gauge
metric of an unknown type (org.jruby.RubyArray) has been create for key:
cluster_uuids. This may result in invalid serialization. It is recommended
to log an issue to the responsible developer/development team.
[2019-11-30T14:52:14,024][INFO ][logstash.javapipeline ][kinesis] Starting
pipeline {:pipeline_id=>"kinesis", "pipeline.workers"=>2,
"pipeline.batch.size"=>125, "pipeline.batch.delay"=>50,
"pipeline.max_inflight"=>250, :thread=>"#<Thread:0x2c171f0a run>"}
[2019-11-30T14:52:14,384][INFO ][logstash.javapipeline ][kinesis] Pipeline
started {"pipeline.id"=>"kinesis"}
[2019-11-30T14:52:14,927][INFO ][logstash.inputs.beats ][metricbeat] Beats
inputs: Starting input listener {:address=>"0.0.0.0:5044"}
[2019-11-30T14:52:14,960][INFO ][logstash.javapipeline ][metricbeat]
Pipeline started {"pipeline.id"=>"metricbeat"}
```

The final set of events summarize the pipelines that are running and the establishment of the listener for Metricbeat data.

```
[2019-11-30T14:52:15,189][INFO ][logstash.agent ] Pipelines running
{:count=>2, :running_pipelines=>[:kinesis, :metricbeat],
:non_running_pipelines=>[]}
[2019-11-30T14:52:15,236][INFO ][org.logstash.beats.Server][metricbeat]
Starting server on port: 5044
[2019-11-30T14:52:15,778][INFO ][logstash.agent ] Successfully
started Logstash API endpoint {:port=>9600}
```

You now have a functional Logstash deployment! We need some data now!

[Review the Metricbeat configuration](#)

Now that Logstash is up and running, we can enable the Metricbeat agent. Review the configuration for Metricbeat by issuing the following command. This configuration is created by AWS CloudFormation template for the Logstash stack.

- **less /etc/metricbeat/metricbeat.yml**

```
[root@ip-10-1-6-149 conf.d]# cat /etc/metricbeat/metricbeat.yml
metricbeat.modules:
- module: system
  metricsets:
    - cpu
    - load
    - filesystem
    - fsstat
    - memory
    - network
    - process
  enabled: true
  period: 10s
  processes: ['..*']
```

```
name: "logstash-server"
output.logstash:
  hosts: ["127.0.0.1:5044"]
logging.to_syslog: true
[root@ip-10-1-6-149 conf.d]#
```

As with the Logstash configuration, there is a concept of inputs and outputs. In this case, Metricbeat is a purpose built, lightweight agent that has specific monitoring modules for input of data for things like cpu, memory, and network data. You enable these modules by adding them to the metricset configuration.

On the output side of things, you configure the Logstash output to write to port 5044. Keep in mind; you configured Logstash in the previous section so it could listen for events written by Metricbeat.

[Install Metricbeat agent from the preconfigured YUM repo](#)

Since the CloudFormation template did not preinstall the Metricbeat agent, you need to install it by issuing the following command:

- **yum install -y metricbeat**

```
[root@ip-10-1-6-149 conf.d]# yum install -y metricbeat
...
Installing : metricbeat-7.4.2-1.x86_64
1/1
warning: /etc/metricbeat/metricbeat.yml created as
/etc/metricbeat/metricbeat.yml.rpmnew
Verifying : metricbeat-7.4.2-1.x86_64
1/1

Installed:
  metricbeat.x86_64 0:7.4.2-1

Complete!
[root@ip-10-1-6-149 conf.d]#
```

You should see the “Complete!” message once the agent installs.

[Start Metricbeat](#)

Start the Metricbeat agent by running the following commands:

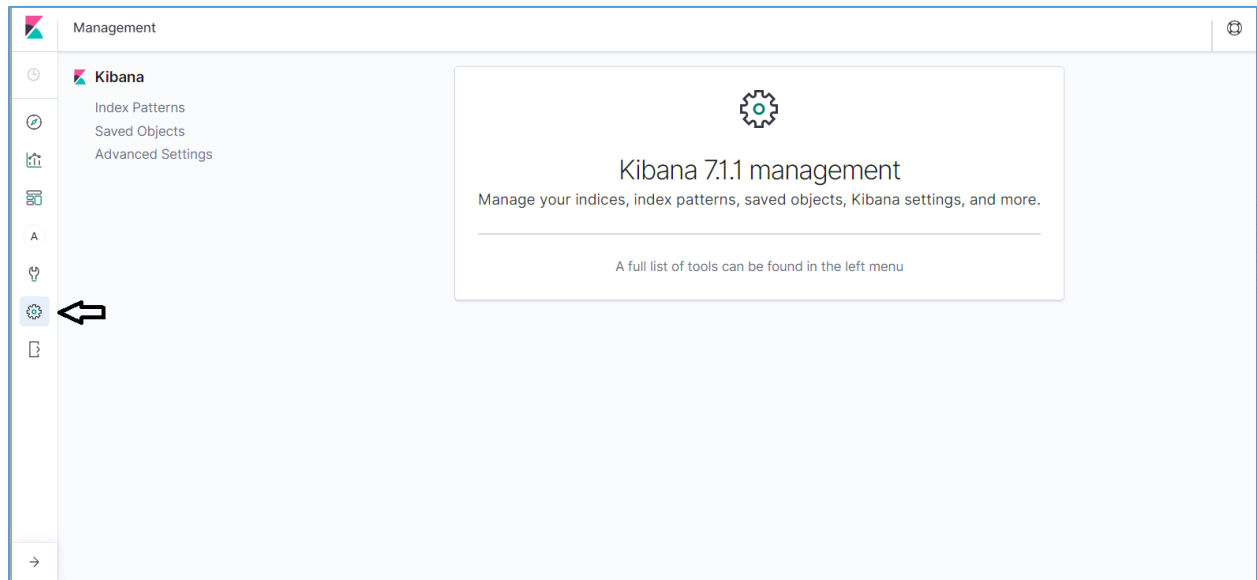
- **service metricbeat start**

```
[root@ip-10-1-6-149 conf.d]# service metricbeat start
Starting metricbeat (via systemctl):
[root@ip-10-1-6-149 conf.d]#
```

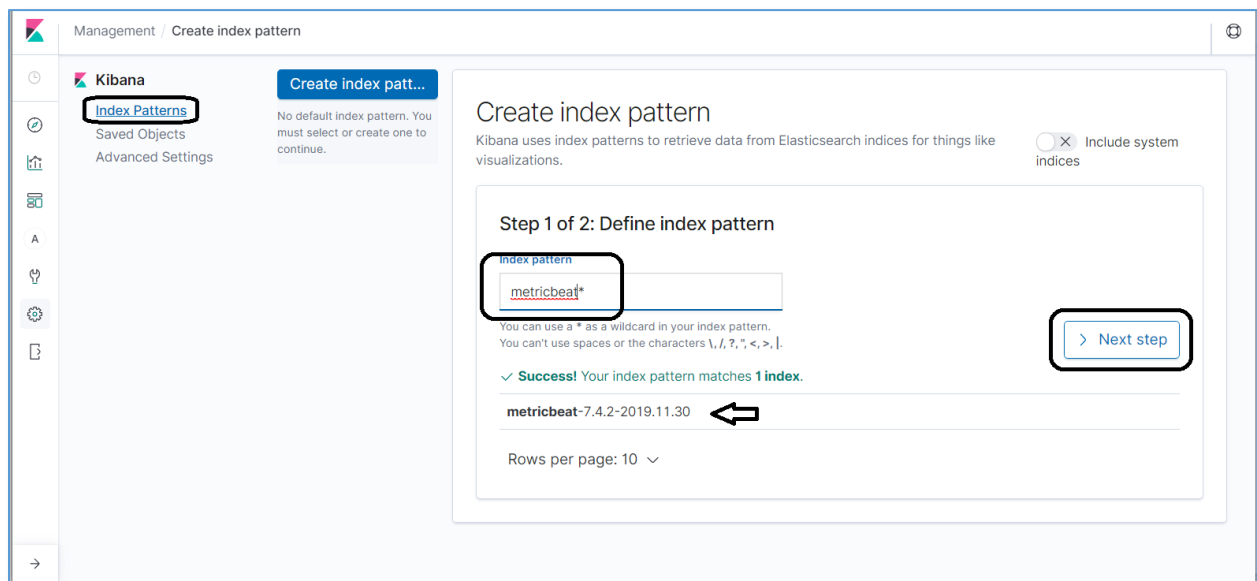
[ OK ]



The process is writing events to Logstash, which in turn, is writing events to Elasticsearch. Let us validate the data in Kibana with the Dev Tools plugin. Navigate to the Kibana console and click on the Settings plugin. You are going to configure an index pattern for Metricbeat.



Click on Index Patterns to create a new pattern.



You will see the newly created metricbeat-7.\*.\* index. Type in metricbeat\* and click “Next step”.

Create index pat...

No default index pattern. You must select or create one to continue.

## Create index pattern

Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.

☐ Include system indices

### Step 2 of 2: Configure settings

You've defined **metricbeat\*** as your index pattern. Now you can specify some settings before we create it.

Time Filter field name

Refresh

@timestamp

system.process.cpu.start\_time

I don't want to use the Time Filter

Show advanced options

[< Back](#)

Create index pattern

Select the **@timestamp** field and click on “Create index pattern”. Your Kibana instance will start scanning the available records and apply default mappings to the data. In the last part of this workshop, you will import prebuilt dashboards for visualizing the Metricbeat data.

Management / metricbeat\*

Kibana

Create index pat...

★ metricbeat\*

Discover

Patterns

Visualize

Dashboard

Settings

Help

metricbeat\*

Time Filter field name: @timestamp

This page lists every field in the **metricbeat\*** index and the field's associated core type as recorded by Elasticsearch. To change a field type, use the Elasticsearch [Mapping API](#)

Fields (127)

Scripted fields (0)

Source filters (0)

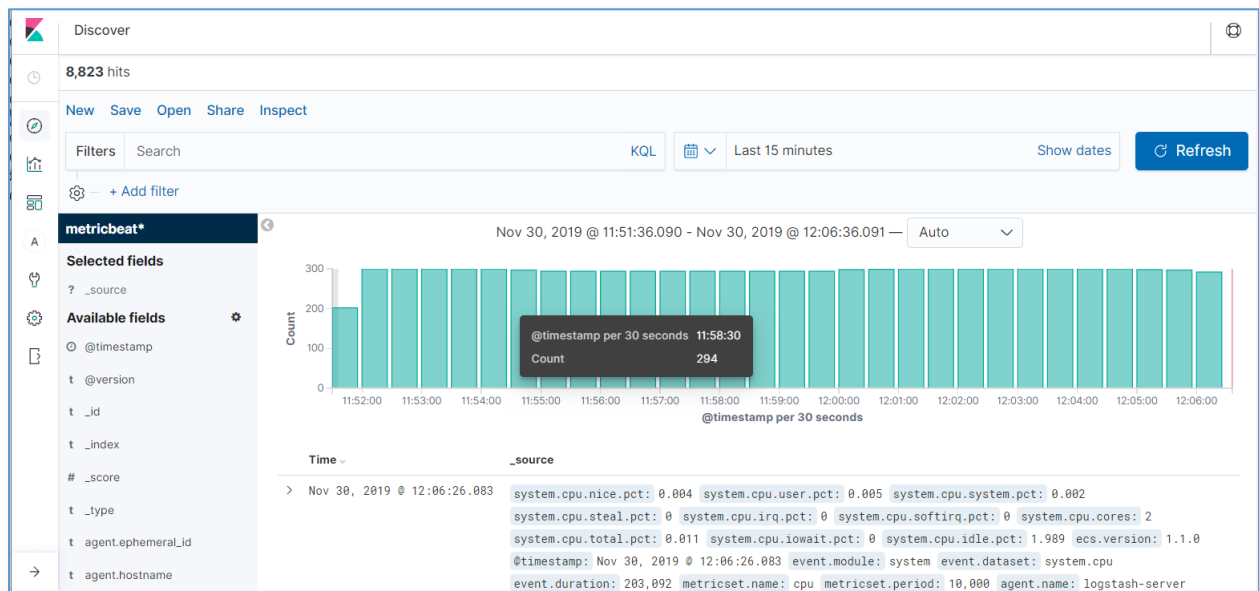
Q Filter

All field types

Name	Type	Format	Searchable	Aggregat...	Excluded
@timestamp	date		●	●	
@version	string		●		

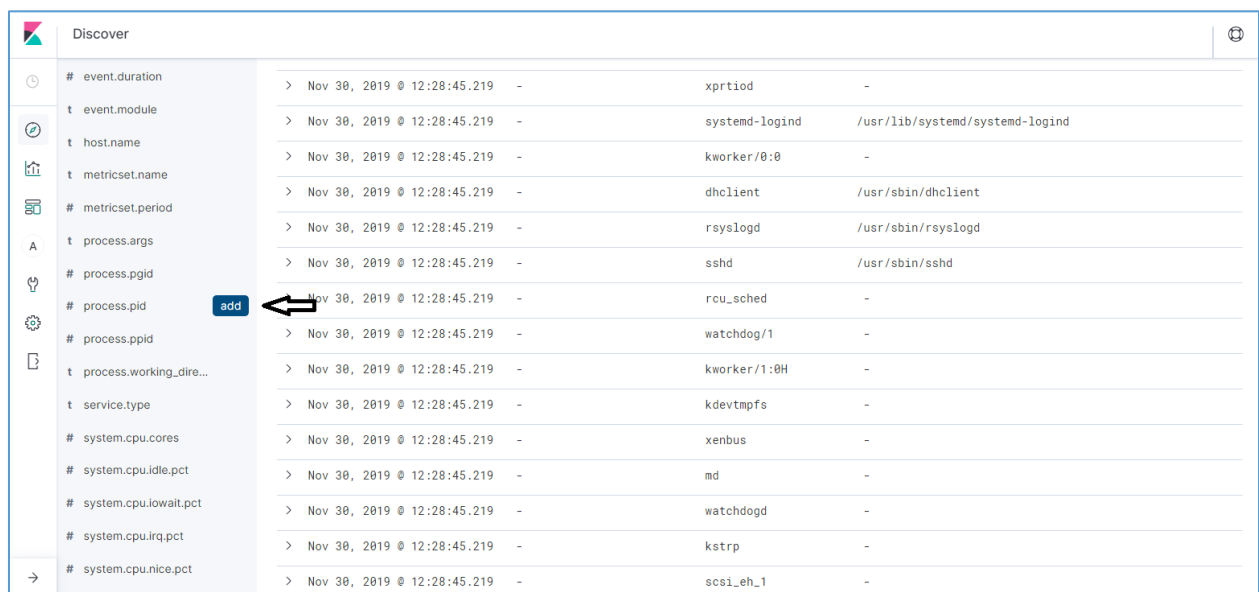
Next, navigate to the Discover plugin on the left hand menu bar.

Once in the Discover plugin, you can start exploring the data. There is a field selection menu on the left hand side to help you navigate the data.

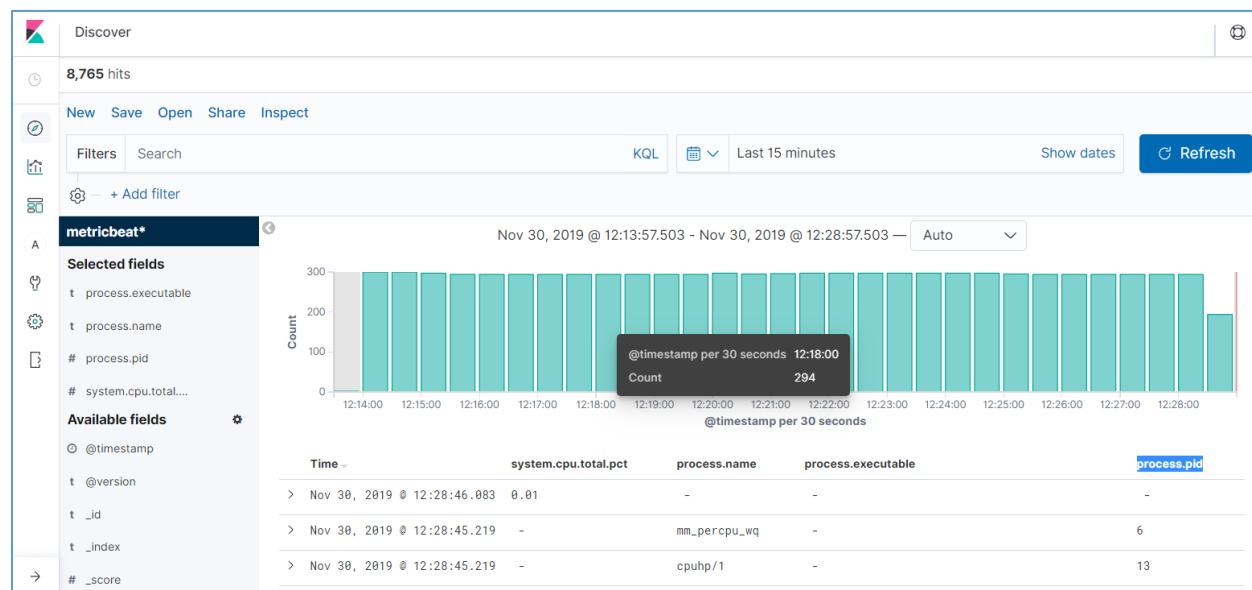


Select the following fields from the field navigator to narrow down the data you view for each document:

- system.cpu.total.pct
- process.name
- process.executable
- process.pid



Scroll back up to the top once you have added the fields. You will see a bunch of bars that represent time slices of data. In this case, it is a 30-second window of data from Metricbeat sliced into buckets. Using your mouse pointer, click on one of the bars to narrow the scope of the data you view.



As you can see, the Discover plugin helps you explore the data to find fields that might be relevant to the operation and monitoring of your entire solution.

Now that you have a flow of data coming in from Logstash, the Amazon Kinesis Data Stream is sitting idle. It needs some data! Let us go to the React application web server, enable Fluentd to write data to the Amazon Kinesis Data Stream so that Logstash can pick it up, and send it to Elasticsearch.

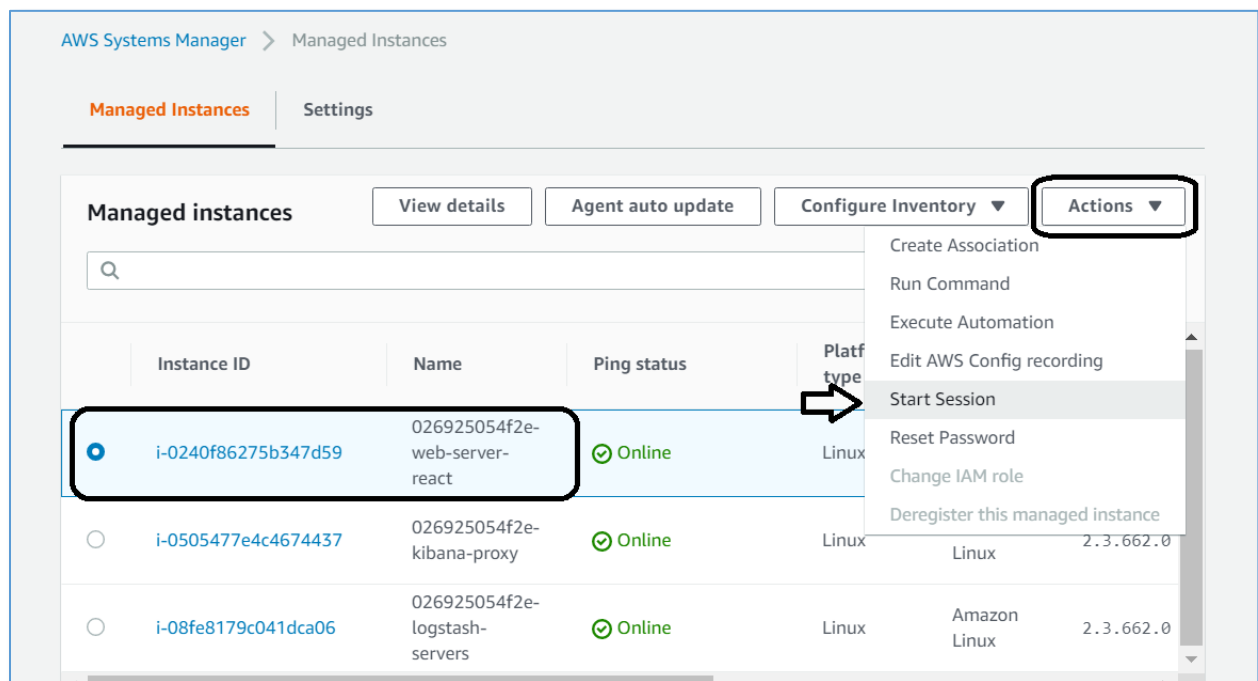
## Log producer configuration (Fluentd) – React Apache Server

Since the React framework serves from an Apache web server, data called “access logs” writes to the local file system. Apache creates both error logs and access logs. You are going to mine data from the access logs so that you can visualize things like most popular queries, the different type of HTTP method calls like GET, PUT and POST, and the volume of data returned in each HTTP call.

For this workshop’s goals, I have specifically chosen Fluentd as the log collector of choice. Why? AWS has many internal agents and one of those is the [Kinesis Agent](#). However, you are going to enrich the data. Both the Kinesis Agent and the Fluentd agent move files and they do it with efficiency and at scale. However, Fluentd provides a rich set of utilities to transform and munge the data before it even gets to Logstash. You could manipulate the data in Logstash, however, you burn CPU cycles in the Logstash

processes. As a design preference in this particular use case, Fluentd provides enrichment at the edge. You get to use the distributed compute of all your logging instances to do the dirty work. That gives you the freedom to optimize the performance to the Logstash deployment and not worry about pegging the JVM for massive transformations. Logstash then becomes a simple pass through framework.

Now that you understand the reason for the choice, let us get started by opening up a session on the React web server. Navigate to AWS Systems Manager and begin a session on the React server.



Once the console renders in your browser, click in the web page and type the following commands:

- `sudo su - root`
- `cd /etc/td-agent/`
- `ls -al`
- `less td-agent.conf`

You will see output similar to what shows below:

```
sh-4.2$ sudo su - root
[root@ip-10-1-6-222 ~]# cd /etc/td-agent/
[root@ip-10-1-6-222 td-agent]# ls -al
total 20
drwxr-xr-x  3 root    root      61 Nov 30 00:00 .
drwxr-xr-x 90 root    root     8192 Nov 30 18:06 ..
```

```
drwxr-xr-x  2 root      root          6 Nov 29 23:58 plugin
-rw-r--r--  1 td-agent td-agent 1020 Nov 30 00:00 td-agent.conf
-rw-r--r--  1 root      root        2381 Nov 29 23:58 td-agent.old
[root@ip-10-1-6-222 td-agent]# less td-agent.conf
```

Now let us review the configuration to understand what is happening.

### Source section

The source section does exactly what it says – it defines your sources. For the monitoring of the web application, you are interested in the access logs. Access logs will give you details about where requests are coming from, what requests are hacking attempts, and what searches are the hottest so you could possibly enable targeted advertising to generate more revenue.

The web application has a tool that rotates logs periodically and it organizes those logs by day. The web server creates a new file each day for those day's events. Since the pattern is "YYYY-mm" – "2019-11" for the directory name and 2020 is shortly arriving, you need to use the [tail](#) functionality for Fluentd. You define the pattern and Fluentd will tail all files in that directory until they delete. Fluentd provides to a "position file" or `pos_file` setting that is a file of pointers to the last line read in a file. As new data arrives, the pointer advances. If Fluentd is shut down, it can be restarted and not have to read the same data again because it knows where it left off.

The [parse](#) directive tells the plugin how to pull out the data (apache2 format, nginx format, etc). This will read the file based on a regex pattern and transform them as a JSON document.

The tag directive gives you a way in which filters can include or exclude different log types in the same configuration file.

```
<source>
  @type tail
  path /var/log/httpd/%Y/*-website.log
  pos_file /var/log/td-agent/apache.log.pos
  tag webserver.access.log
  <parse>
    @type apache2
  </parse>
</source>
```

## Filter section

The [filter](#) sections allow you to add data, change data and even remove data to your final output. You can install custom plugins or use the out of the box plugins that are prepackaged with Fluentd.

### Explore the ec2\_metadata filter

The [ec2\\_metadata](#) filter uses the instance profile and metadata to extract specific details about the AWS deployment. This filter adds fields to the access log data. You must have an instance profile deployed on the EC2 instance to get access to this data and you need to have the appropriate permissions for the plugin to make calls to informational APIs on AWS.

```
<filter webserver.access.log>
  @type ec2_metadata

  metadata_refresh_seconds 300 # Optional, default 300 seconds

  <record>
    hostname      ${tagset_name}
    instance_id   ${instance_id}
    instance_type ${instance_type}
    private_ip    ${private_ip}
    az            ${availability_zone}
    vpc_id        ${vpc_id}
    ami_id        ${image_id}
    account_id    ${account_id}
  </record>
</filter>
```

### Explore the record\_transformer filter

The [record\\_transformer](#) filter applies a new field to the access log data. This field gives Logstash a specific keyword so that it can filter events coming from the Kinesis Data Stream. For the React web server, you will use a key called “event\_group” and associate a value of “imdb” for the entry. Review the Logstash setup if you want to understand how the conditional filtering works for sending to different indexes.

```
<filter webserver.access.log>
  @type record_transformer
  <record>
    event_group imdb
  </record>
</filter>
```

## Match section

The match section gives you the functionality of routing output to different destinations. In this case, you are sending data to an Amazon Kinesis Data Stream. The [kinesis\\_streams](#) plugin uses the instance profile and metadata to sign requests using Sigv4. The EC2 instance has an IAM policy for access to the Amazon Kinesis Data Stream. The values for the stream name and the region populate automatically in the CloudFormation process.

```
<match webserver.access.log>
  # plugin type
  @type kinesis_streams

  # your kinesis stream name
  stream_name log-delivery-026925054f2e
  # AWS region
  region eu-central-1

  <buffer>
    # Frequency of ingestion
    flush_interval 5s
    # Parallelism of ingestion
    flush_thread_count 8
  </buffer>
</match>
```

## Start the Fluentd agent (react server)

[Fluentd](#) is an open source product created and managed by a company called [Treasure Data, Inc.](#) The agent that controls the startup is the td-agent. Navigate to your console and execute the following commands:

- **chown -R root:td-agent /var/log/httpd**
- **chmod -R 777 /var/log/httpd/**
- **service td-agent start**
- **less /var/log/td-agent/td-agent.log**

```
[root@ip-10-1-6-222 td-agent]# service td-agent start
Starting td-agent (via systemctl): [ OK ]
[root@ip-10-1-6-222 td-agent]# less /var/log/td-agent/td-agent.log
```

Observe similar output in the logs:

```
2019-11-30 19:47:26 +0000 [info]: parsing config file is succeeded
path="/etc/td-agent/td-agent.conf"
2019-11-30 19:47:27 +0000 [info]: using configuration file: <ROOT>
  <source>
...
</match>
```



```
</ROOT>
2019-11-30 19:47:27 +0000 [info]: starting fluentd-1.7.0 pid=4494
ruby="2.4.6"
2019-11-30 19:47:27 +0000 [info]: spawn command to main:  cmdline=["/opt/td-
agent/embedded/bin/ruby", "-Eascii-8bit:ascii-8bit", "/opt/td-
agent/embedded/bin/fluentd", "--log", "/var/log/td-agent/td-agent.log", "--
daemon", "/var/run/td-agent/td-agent.pid", "--under-supervisor"]
2019-11-30 19:47:27 +0000 [info]: gem 'fluent-plugin-ec2-metadata' version
'0.1.2'
2019-11-30 19:47:27 +0000 [info]: gem 'fluent-plugin-elasticsearch' version
'3.5.4'
2019-11-30 19:47:27 +0000 [info]: gem 'fluent-plugin-kafka' version '0.11.1'
2019-11-30 19:47:27 +0000 [info]: gem 'fluent-plugin-kinesis' version '3.2.0'
2019-11-30 19:47:27 +0000 [info]: gem 'fluent-plugin-prometheus' version
'1.5.0'
2019-11-30 19:47:27 +0000 [info]: gem 'fluent-plugin-record-modifier' version
'2.0.1'
2019-11-30 19:47:27 +0000 [info]: gem 'fluent-plugin-rewrite-tag-filter'
version '2.2.0'
2019-11-30 19:47:27 +0000 [info]: gem 'fluent-plugin-s3' version '1.1.11'
2019-11-30 19:47:27 +0000 [info]: gem 'fluent-plugin-td' version '1.0.0'
2019-11-30 19:47:27 +0000 [info]: gem 'fluent-plugin-td-monitoring' version
'0.2.4'
2019-11-30 19:47:27 +0000 [info]: gem 'fluent-plugin-webhdfs' version '1.2.4'
2019-11-30 19:47:27 +0000 [info]: gem 'fluentd' version '1.7.0'
2019-11-30 19:47:27 +0000 [info]: adding filter
pattern="webserver.access.log" type="ec2_metadata"
2019-11-30 19:47:28 +0000 [info]: adding filter
pattern="webserver.access.log" type="record_transformer"
2019-11-30 19:47:28 +0000 [info]: adding match pattern="webserver.access.log"
type="kinesis_streams"
2019-11-30 19:47:28 +0000 [info]: adding source type="tail"
2019-11-30 19:47:28 +0000 [info]: #0 starting fluentd worker pid=4517
ppid=4512 worker=0
2019-11-30 19:47:28 +0000 [info]: #0 fluentd worker is now running worker=0
```

Validate you are receiving logs in Kibana

Navigate to your Kibana console. Go to the Dev Tools plugin and run the following command:

- `GET _cat/indices`

Observe similar output and that an index prefixed with proxy exists.

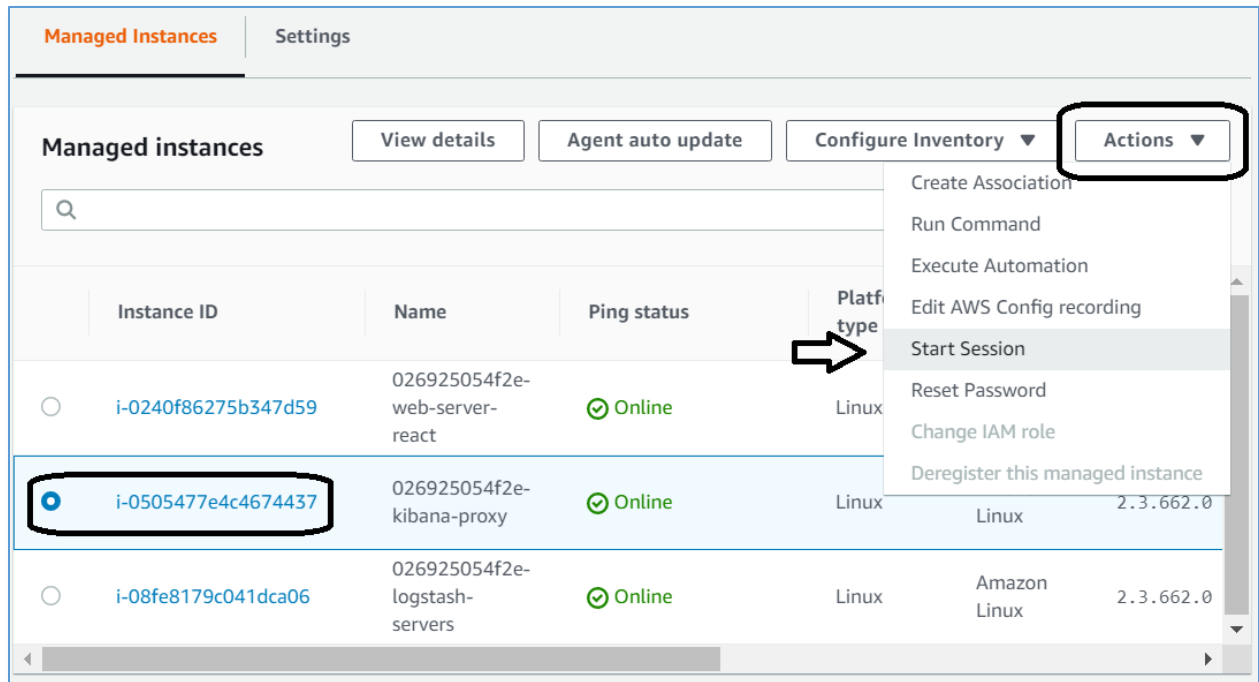


You now have data for the proxy servers. Next, you need to configure the Kibana proxy NGINX server to produce logs.

## Log producer configuration (Fluentd) – Kibana Proxy - NGINX Server

Since the proxy server brokers Kibana requests, you want to capture the access logs generated by NGINX. Just as seen with the web application Apache servers, you will tail a file, enhance the data and then

Navigate to AWS Systems Manager and begin a session on the proxy server.



Once the console renders in your browser, click in the web page and type the following commands:

- `sudo su - root`
- `cd /etc/td-agent/`
- `ls -al`
- `less td-agent.conf`

You will see output similar to what shows below:

```
sh-4.2$ sudo su - root
Last login: Sat Nov 30 20:26:55 UTC 2019 on pts/0
[root@ip-10-1-0-150 ~]# cd /etc/td-agent/
[root@ip-10-1-0-150 td-agent]# ls -al
total 20
drwxr-xr-x  3 root    root      61 Nov 30 20:05 .
drwxr-xr-x 86 root    root     8192 Nov 29 18:27 ..
drwxr-xr-x  2 root    root        6 Nov 29 15:30 plugin
-rw-r--r--  1 td-agent td-agent 1004 Nov 30 20:05 td-agent.conf
-rw-r--r--  1 root     root     2381 Nov 29 15:30 td-agent.old
[root@ip-10-1-0-150 td-agent]# less td-agent.conf
```

The noticeable difference in this configuration compared to the web server are the following sections.

### Source section

Compared to the web server, the proxy server reads from NGINX access logs. The configuration varies to read a different directory and use a different parser – the [nginx](#) parser.

```
<source>
  @type tail
  path /var/log/nginx/access.log
  pos_file /var/log/td-agent/access.log.pos
  tag proxy.nginx.access
  <parse>
    @type nginx
  </parse>
</source>
```

### Filter section

The only difference between the web server and the proxy server in this section is the `event_group` is “proxy” instead of “imdb”.

```
<filter proxy.nginx.access>
  @type record_transformer
  <record>
    event_group proxy
  </record>
</filter>
```

### Start the Fluentd agent (proxy server)

Navigate to your console and execute the following commands in the session manager:

- **service td-agent start**
- **less /var/log/td-agent/td-agent.log**

```
[root@ip-10-1-0-150 td-agent]# service td-agent start
Starting td-agent (via systemctl): [ OK ]
[root@ip-10-1-0-150 td-agent]# less /var/log/td-agent/td-agent.log
```

Observe similar output in the logs:

```
019-12-01 01:09:51 +0000 [info]: starting fluentd-1.7.0 pid=17110
ruby="2.4.6"
2019-12-01 01:09:51 +0000 [info]: spawn command to main: cmdline=["/opt/td-agent/embedded/bin/ruby", "-Eascii-8bit:ascii-8bit", "/opt/td-agent/embedded/bin/fluentd", "--log", "/var/log/td-agent/td-agent.log", "--daemon", "/var/run/td-agent/td-agent.pid", "--under-supervisor"]
2019-12-01 01:09:51 +0000 [info]: gem 'fluent-plugin-ec2-metadata' version '0.1.2'
2019-12-01 01:09:51 +0000 [info]: gem 'fluent-plugin-elasticsearch' version '3.5.4'
```

```
2019-12-01 01:09:51 +0000 [info]: gem 'fluent-plugin-kafka' version '0.11.1'
2019-12-01 01:09:51 +0000 [info]: gem 'fluent-plugin-kinesis' version '3.2.0'
2019-12-01 01:09:51 +0000 [info]: gem 'fluent-plugin-prometheus' version
'1.5.0'
2019-12-01 01:09:51 +0000 [info]: gem 'fluent-plugin-record-modifier' version
'2.0.1'
2019-12-01 01:09:51 +0000 [info]: gem 'fluent-plugin-rewrite-tag-filter'
version '2.2.0'
2019-12-01 01:09:51 +0000 [info]: gem 'fluent-plugin-s3' version '1.1.11'
2019-12-01 01:09:51 +0000 [info]: gem 'fluent-plugin-td' version '1.0.0'
2019-12-01 01:09:51 +0000 [info]: gem 'fluent-plugin-td-monitoring' version
'0.2.4'
2019-12-01 01:09:51 +0000 [info]: gem 'fluent-plugin-webhdfs' version '1.2.4'
2019-12-01 01:09:51 +0000 [info]: gem 'fluentd' version '1.7.0'
2019-12-01 01:09:51 +0000 [info]: adding filter pattern="proxy.nginx.access"
type="ec2_metadata"
2019-12-01 01:09:52 +0000 [info]: adding filter pattern="proxy.nginx.access"
type="record_transformer"
2019-12-01 01:09:52 +0000 [info]: adding match pattern="proxy.nginx.access"
type="kinesis_streams"
2019-12-01 01:09:52 +0000 [info]: adding source type="tail"
2019-12-01 01:09:52 +0000 [info]: #0 starting fluentd worker pid=17133
ppid=17128 worker=0
2019-12-01 01:09:52 +0000 [info]: #0 following tail of
/var/log/nginx/access.log
2019-12-01 01:09:52 +0000 [info]: #0 fluentd worker is now running worker=0
2019-12-01 01:10:05 +0000 [info]: #0 disable filter chain optimization
because [Fluent::Plugin::RecordTransformerFilter] uses `#filter_stream`
method.
```

Validate you are receiving logs in Kibana

Navigate to your Kibana console. Go to the Dev Tools plugin and run the following command:

- `GET _cat/indices`

Observe similar output and that an index prefixed with proxy exists.



Jump to [Lab 3](#).