Building a Logging Pipeline with AWS

You use logging, right?

```
2018-03-12 11:30:28,243 [http-bio-9966-exec-1] DEBUG ClinicServiceImpl - findOwnerById(1)
2018-03-12 11:30:44,148 [http-bio-9966-exec-4] DEBUG ClinicServiceImpl - findOwnerById(1)
2018-03-12 11:30:44,154 [http-bio-9966-exec-4] DEBUG ClinicServiceImpl - findPetTypes()
2018-03-12 11:31:14,972 [http-bio-9966-exec-2] DEBUG ClinicServiceImpl - findOwnerById(1)
2018-03-12 11:31:14,977 [http-bio-9966-exec-2] DEBUG ClinicServiceImpl - findPetTypes()
2018-03-12 11:31:14,984 [http-bio-9966-exec-2] DEBUG ClinicServiceImpl - findPetTypes()
2018-03-12 11:31:14,990 [http-bio-9966-exec-2] DEBUG ClinicServiceImpl - savePet(pumpkin cheeks)
2018-03-12 11:31:15,029 [http-bio-9966-exec-2] DEBUG ClinicServiceImpl - findOwnerById(1)
2018-03-12 11:34:16,300 [http-bio-9966-exec-1] DEBUG ClinicServiceImpl - findOwnerById(19)
2018-03-12 11:34:16,449 [http-bio-9966-exec-1] WARN  ClinicServiceImpl - findOwnerById: invalid ID 19
```

The problems with local logging

Have to go to the machine to see logs (or retrieve with scp/rsync)

Limited ability to extract information (grep is your only friend)

No correlation of events from multiple sources

Logfiles disappear when machine shuts down

Options for Centralized Logging

Syslog

The logging daemon that comes with Linux.

Processes send UDP messages to a daemon (either local or remote).

Pros:

- Library support for most major languages (or roll your own)
- Can integrate with many third-party applications

Cons:

- Limited ability to customize output: everything goes in the “message”
- Need an agent/configuration on each machine
- Sending messages requires a context switch

AWS Log Agents

A tool provided by AWS Labs: monitors specified logfiles and sends their contents to an AWS service (CloudWatch Logs or Kinesis).

Pros:

- A (relatively) easy way to capture local files for centralized logging

Cons:

- Must install/configure agent on each server
- No built-in support for associating instance information with logging events
- Each line of the file is sent as a separate logging event

Logstash / Elasticsearch / Kibana

The standard “roll your own” solution: Logstash parses logfiles and sends them to Elasticsearch, Kibana provides analytics.

Pros:

- Can find pre-built parsers for standard logging formats

Cons:

- Must install/configure agent
- “Roll your own” means you deploy and maintain

External Service Providers

Several companies provide Log Management as a Service: they ingest your logs, store them in a search engine, and provide you a variety of ways to analyze.

Pros:

- Nothing to manage locally (other than feeding the logs to the service)
- Can ingest many different formats
- Wide variety of analytics tools, including machine learning

Cons:

- Cost ratchets up quickly depending on volume and storage duration
- Not all plans support archives

An AWS-Centric Alternative

Kinesis / Firehose / AWS Elasticsearch / Kibana

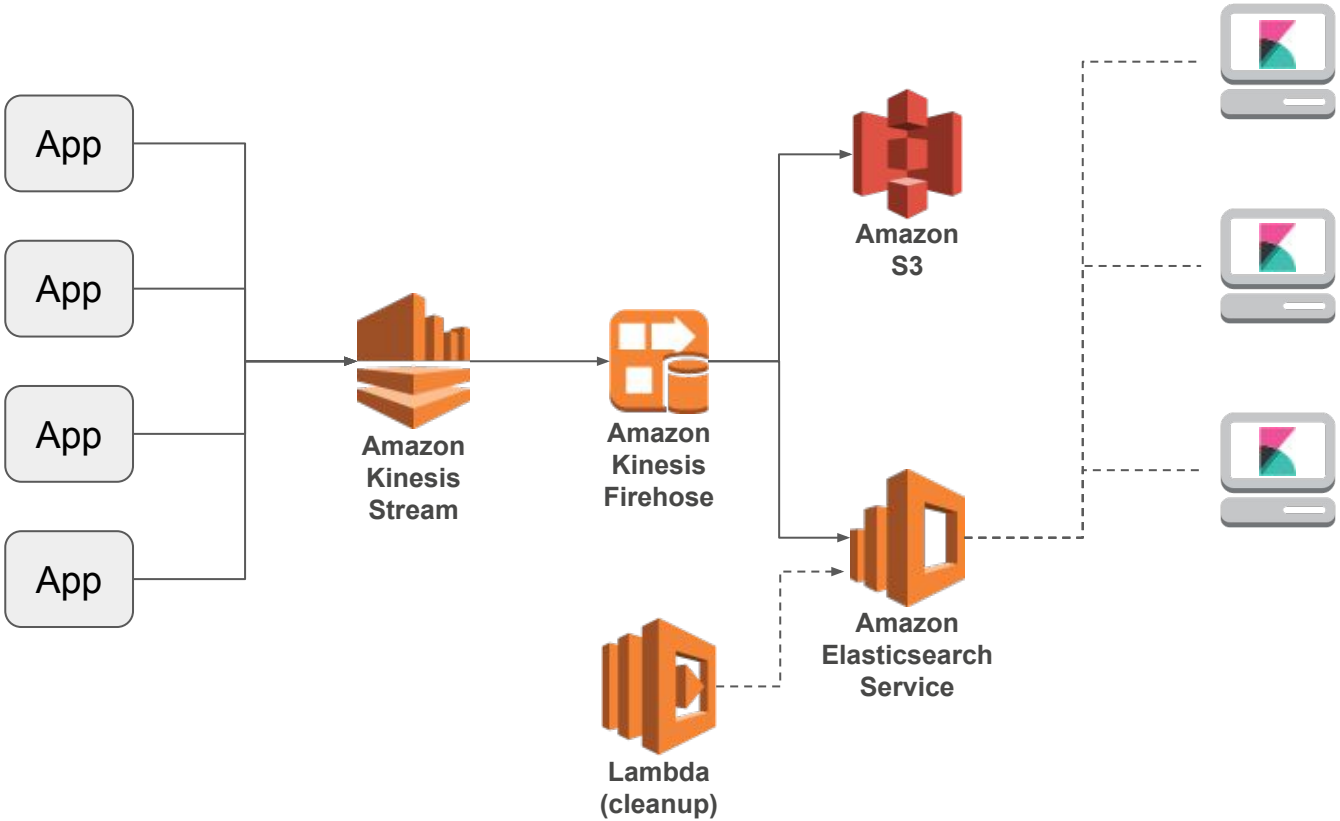
A homegrown solution that leverages several AWS services.

Pros:

- Amazon manages the components -- It Just Works™
- Scalable: configure each component to match volumes
- Can use Lambda functions to parse arbitrary messages
- Easy to archive messages (on S3)
- Kinesis stream can service multiple destinations

Cons:

- More expensive than fully-managed solutions at small scales
- Still some management required (cleanup of old indexes)
- Must provide logging events to Elasticsearch in JSON


App
App
App
App
Amazon
Kinesis
Stream
Amazon
Kinesis
Firehose
Amazon
S3
Amazon
Elasticsearch
Service
Lambda
(cleanup)

Sizing and Costs

Description	Unit Cost	Monthly Cost
Kinesis Streams		
8 shards	0.02 / hour	87.84
100 MM payload units/day (est)	0.014 / MM units	42.70
Kinesis Firehose		
100 MM records/day (est)	0.029 / MM records	88.45
S3 Backup		
1 GB/day IA (est)	0.013 / GB / Month	0.38
1 GB/day Standard (est)	0.023 / GB / Month	0.70
Elasticsearch Cluster		
6 m4.large instance	0.15 / hour	663.19
512 GB storage/node (SSD)	0.135 / GB / Month	414.72
Total		1,297.98

Shameless Plug: log4j-aws-appenders

https://github.com/kdgregory/log4j-aws-appenders

Appender library for Log4J 1.2

- Supports Cloudwatch Logs, Kinesis, Simple Notification Service (SNS)

Produces JSON output

- No parsing needed for Elasticsearch ingest

Adds additional metadata to logging events

- Instance metadata (hostname, …)
- User-defined metadata (application name, …)

Sample Output


```
{
    "hostname": "ithilien",
    "level": "DEBUG",
    "logger": "org.springframework.samples.petclinic.service.ClinicServiceImpl",
    "message": "findOwnerByLastName(franklin)",
    "processId": "5971",
    "tags": {
        "appName": "Example",
        "deployment": "prod"
    },
    "thread": "http-bio-9966-exec-4",
    "timestamp": "2018-03-12T15:41:31.564Z"
}
```

Configuration

```
log4j.appender.kinesis=com.kdgregory.log4j.aws.KinesisAppender

log4j.appender.kinesis.streamName=LoggingPipeline

log4j.appender.kinesis.layout=com.kdgregory.log4j.aws.JsonLayout

log4j.appender.kinesis.layout.tags=appName=Example,deployment={env:ENV}

log4j.appender.kinesis.layout.enableHostname=true

log4j.appender.kinesis.layout.enableLocation=false
```

Example: Spark Application

Same code, running on multiple nodes

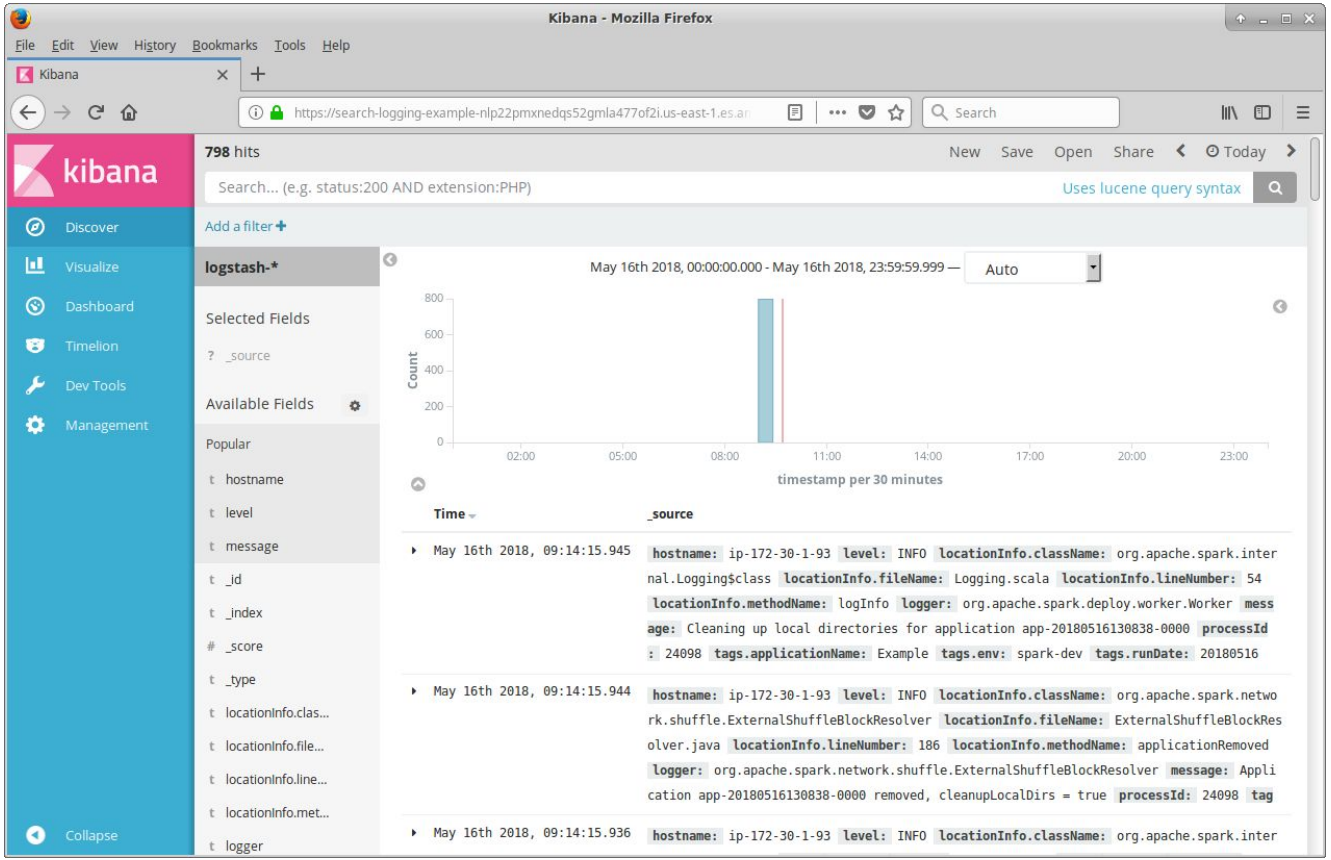
- Where was a particular task run?
- Do we have a “hot” node?
- How much progress are we making (for large jobs)?

Application logging can be buried in framework logging

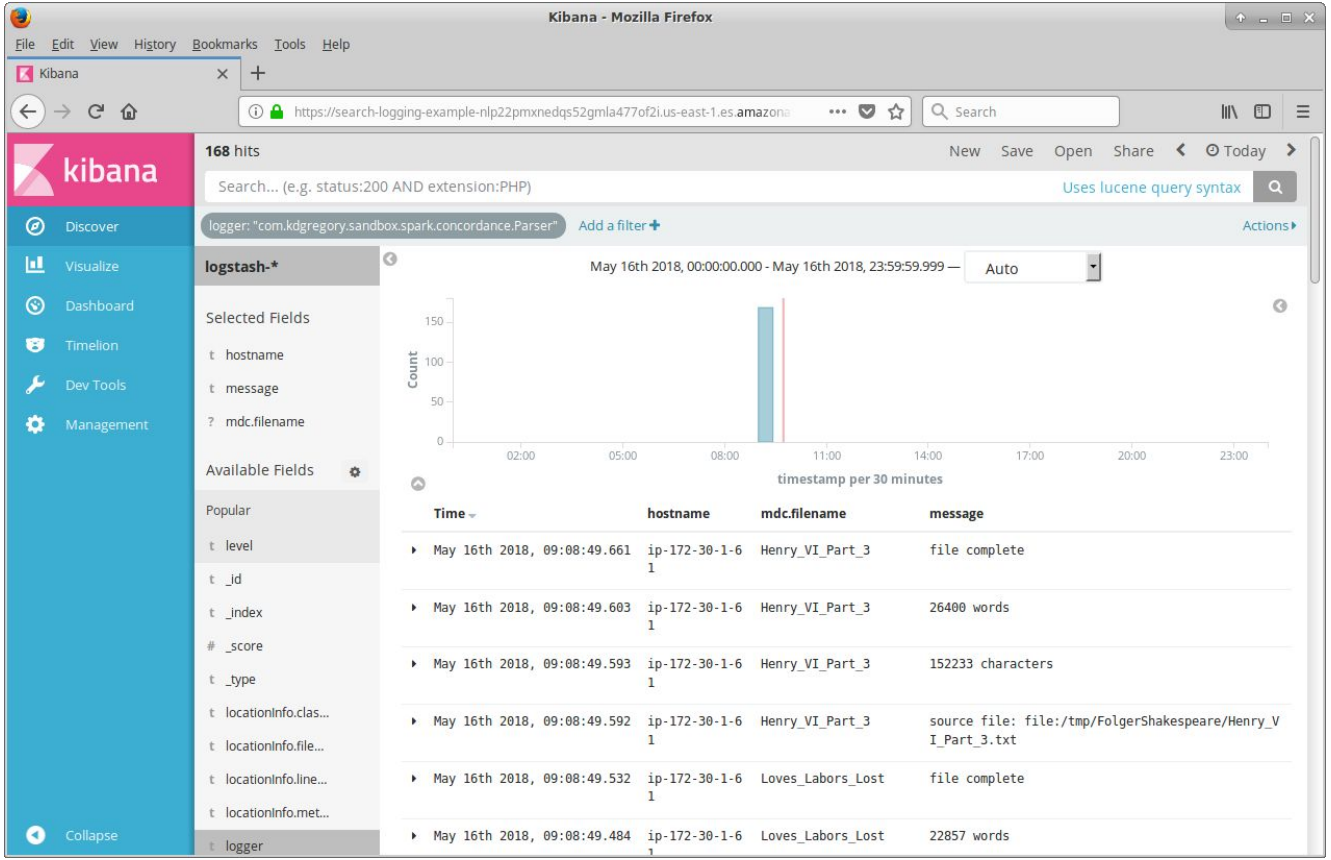
Leverages MDC

- Messages include application name, ID
- Parser adds name of current file

Kibana: Default View



Kibana: After Configuration



Monitoring and Management

Can use AWS CloudWatch metrics to track operation

- How many records are ingested per day
- Kinesis throttling
- How much space is available in Elasticsearch cluster

Need to delete old Elasticsearch indexes

- http://blog.kdgregory.com/2018/02/cleaning-up-aws-elasticsearch-indexes.html
- Decision: how many days of indexes to keep

Load Balancer Logs

Example

```
2018-04-30T12:40:34.740628Z
```
⇐ timestamp when request received

```
example
```
⇐ load balancer name

```
173.49.123.456:58798 172.30.1.119:80
```
⇐ source and destination IPs

```
0.000037 0.001039 0.000046
```
⇐ elapsed time for ELB, back-end service

```
200 200 0 43862
```
⇐ HTTP status codes, request/response size

```
"GET http://example-1886477337.us-east-1.elb.amazonaws.com:80/index.php?page=scm.git
HTTP/1.1"
```
⇐ request

```
"curl/7.47.0"
```
⇐ user agent

```
- -
```
⇐ SSL cipher/protocol

Features

Every request is logged, whether or not successful

Logs are written to S3, organized by date/time

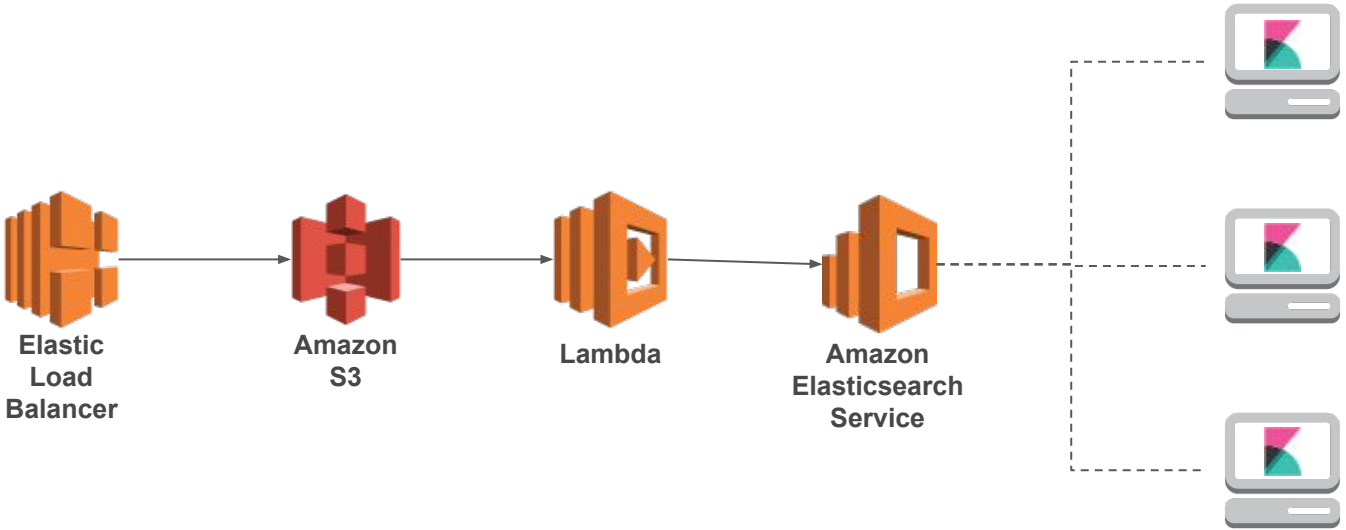
- Can choose whether to write every hour or every five minutes

Multiple tools available to analyze

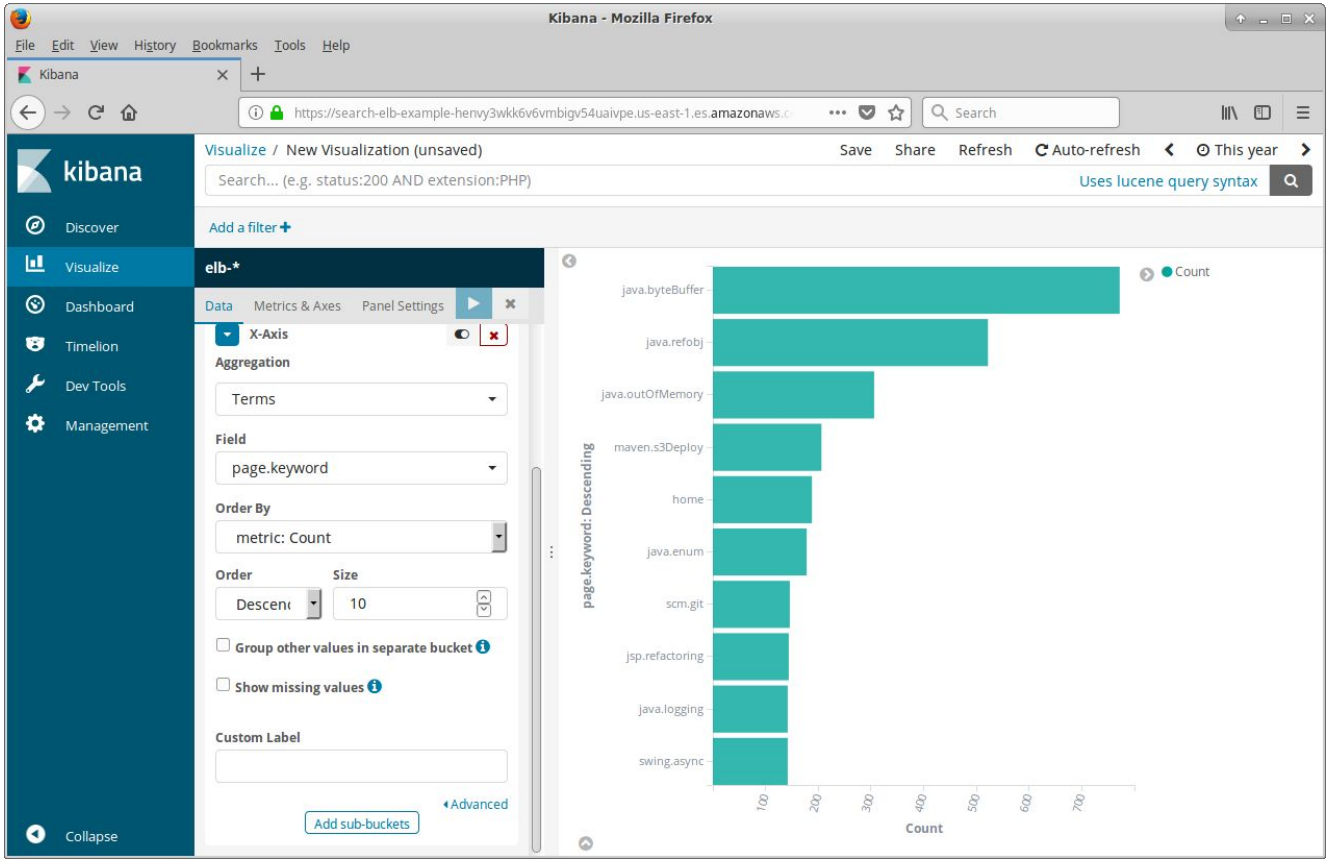
- Simple queries via AWS Athena
- Plugins for Logstash, Loggly, Splunk, …

Home-grown solution: Lambda to parse logs, write to Elasticsearch

- Triggered every time a logfile is written to S3
- Will extract details from URL


Elastic
Load
Balancer
Amazon
S3
Lambda
Amazon
Elasticsearch
Service

Kibana: Top 10 Pages



CloudWatch Metrics

Overview

Time-series dimensioned metrics

- Most AWS services have a set of predefined metrics
- Example: free storage space by database instance

May be used to trigger alarms

- Which in turn can trigger auto-scaling

Applications can write custom metrics

- Example: request processing time by URL

Custom metrics are charged per metric, per month

- A “metric” is a specific combination of dimensions

Effective Logging Techniques

Use the Right Log Level

ERROR: major problems, system may not continue to function

- These should be very infrequent, may be important enough to wake people up

WARN: bad or unexpected data, should be corrected/investigated

- Useful for planning “maintenance” projects

INFO: general progress reports

- Can be the basis for analytics

DEBUG: detailed progress reports or data dumps

- Probably *don’t* want these to go to Elasticsearch

Use the Mapped Diagnostic Context

A per-thread map of data that is attached to all log output

Useful to track all calls to fulfill a service request

- Endpoint name
- Invoking user
- Request ID
- …

Can be a quick-and-dirty way to expose metrics

Use Tracer Bullets for Micro-Services

A generated tag (eg: UUID) used to track requests through micro-services

- Create when request first enters application
- Add to request headers to propagate through system
- Application Load Balancer will generate automatically

Log with Mapped Diagnostic Context (MDC)

You Aren’t Restricted To One Logger

Loggers are named by strings, classname is just a convention

Use primary logger for information relevant to debugging, secondary loggers for special cases

- Operational statistics
- Critical errors
- Anything that you don’t want polluting your main log

Don’t Store Logger in Static Variable

Source of copy-paste errors

Provides misleading information when inheritance involved

Things that do logging shouldn’t be worried about instance size

Centralization Enables New Habits

Save/rerun useful queries

- Errors in last 24 hours
- Distribution of X by instance

Visualize your data

Notify developers as errors happen

- SNS -> Slack -> you
- Only useful if you’re not overwhelmed by messages

Deep analysis

- Use AWS Athena to analyze logs over time

For More Information

Building a Logging Pipeline

http://www.kdgregory.com/index.php?page=aws.loggingPipeline

Kibana

https://www.elastic.co/guide/en/kibana/current/index.html

Amazon services

Kinesis Streams: https://docs.aws.amazon.com/streams/latest/dev/introduction.html
Kinesis Firehose: https://docs.aws.amazon.com/firehose/latest/dev/what-is-this-service.html
Elasticsearch: https://docs.aws.amazon.com/elasticsearch-service/latest/developerguide/what-is-amazon-elasticsearch-service.html