elastic | O11y
GLOBAL 2021

Data Streams & Pipelines

Observing with the Elastic Stack

Pete Hampton
Data Engineer, Elastic Security

You are here

01 Intro

02 Data Engineering

03 Case Study 1 – Batch

04 Case Study 2 – Streaming

05 Advanced Usage

Who am I?

- 👋 **Name**: Pete Hampton
- ☀️ **Live**: Belfast, N. Ireland
- 🤖 **PhD**: Artificial Intelligence
- 🧐 **Work**: Data processing & Distributed systems

Who are you?

- 🤓 Developer / DevOps / Data Scientist
- 😎 Builds / Operates pipelines
- 😇 Elastic Stack rookie

Mission

The goals for this presentation

- Discuss modern challenges in Data Engineering
- Elastic Stack - Get started / enhance your usage
- Share anecdotes, use cases, and recommendations

Intro

Pipelines? 🧐 Streams? 🧐

Pipelines

Definition for this presentation

- Typically, batch / scheduled based
- Can be slow(er) to deliver insights
- Moving data around on demand
- Think Airflow, Luigi, & other workflow/schedule systems

Streams

Definition for this presentation

- Not talking about low latency (<100µs) or hard real-time (<20ms)
- Think fast data & *near* real-time
- Processing of data as it received (Polled or Pushed)
- Think tech *Flink, Kafka, Spark Streaming, RabbitMQ*, etc

Data Engineering

Systems, Services & Infrastructure

“Data” engineers design and build pipelines that transform and transport data into a format wherein, by the time it reaches the Data Scientists or other end users, it is in a highly usable state.

https://quanthub.com/what-is-data-engineering/

Data Engineering

- *Find and make data usable for products, systems and stakeholders*

- Use Cases
  - Big Data Ingest
  - Business Intelligence
  - Contextual product enhancements, eg Recommender Systems

- Rapidly evolving
  - Monoliths -> SOA -> Microservices -> Cloud Native -> Low Code -> No Code
  - New tools and programming languages come and go
  - Industry adopting streaming approaches more and more

Data Engineering

Day to day struggles

- Taming Large / Complex data infrastructures and integrations
- Few or no engineers understand the entire data plane
- Development slows down as things get more complex
- Data processing can continue after it no longer yields value

Integration Points

Systems and Services that connect

- Databases / Data Warehouses
- Cloud Providers
- Edge Services
- 3rd Party APIs
- And much more

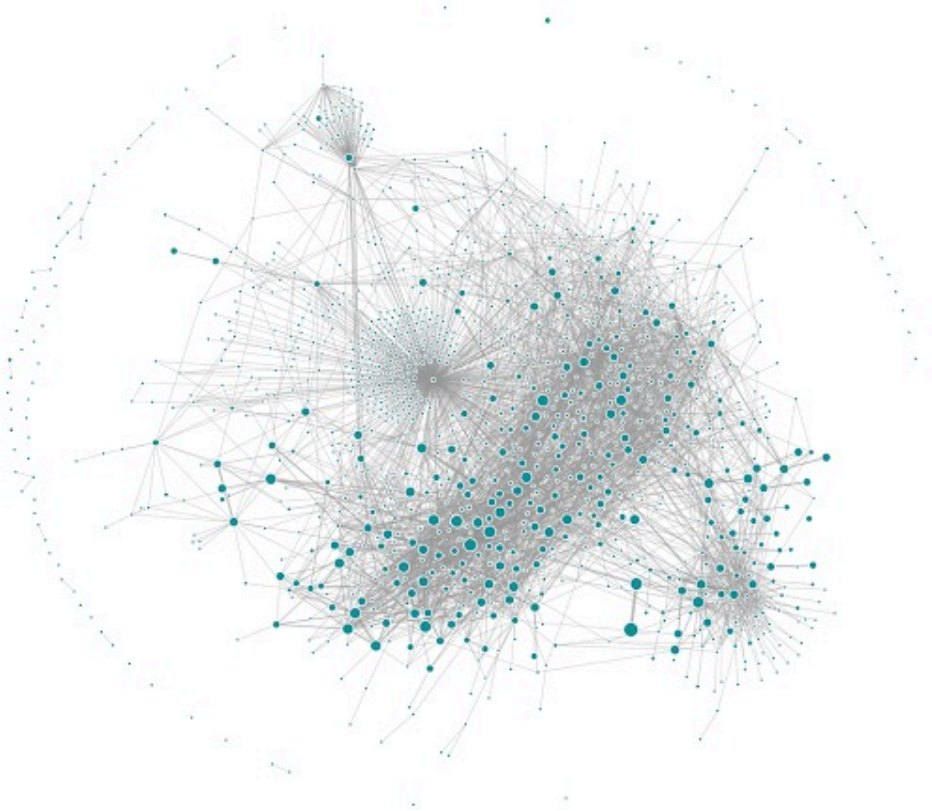
What makes it so complex?

- Too many tools, clouds, SaSS
- Expertise comes and goes
- No or scattered system of record
- Synchronization is a tricky business
- Scaling Batch / ETL / Lambda architectures are difficult

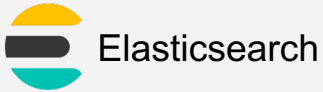
Uber Service Map

~2018

- 2k+ critical microservices
- *“engineers had to work through around 50 services across 12 different teams in order to investigate the root cause of the problem.”*



Ref: https://eng.uber.com/microservice-architecture/




```
~
→ curl localhost:9200 | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   552  100   552    0     0   107k      0 --:--:-- --:--:-- --:--:--  134k
{
  "name": "Petes-MacBook-Pro.local",
  "cluster_name": "elasticsearch",
  "cluster_uuid": "pHen4Y7oRoOO-sU0zGbkzg",
  "version": {
    "number": "7.14.0",
    "build_flavor": "default",
    "build_type": "tar",
    "build_hash": "dd5a0a2acaa2045ff9624f3729fc8a6f40835aa1",
    "build_date": "2021-07-29T20:49:32.864135063Z",
    "build_snapshot": false,
    "lucene_version": "8.9.0",
    "minimum_wire_compatibility_version": "6.8.0",
    "minimum_index_compatibility_version": "6.0.0-beta1"
  },
  "tagline": "You Know, for Search"
}

~
→
```

Elasticsearch

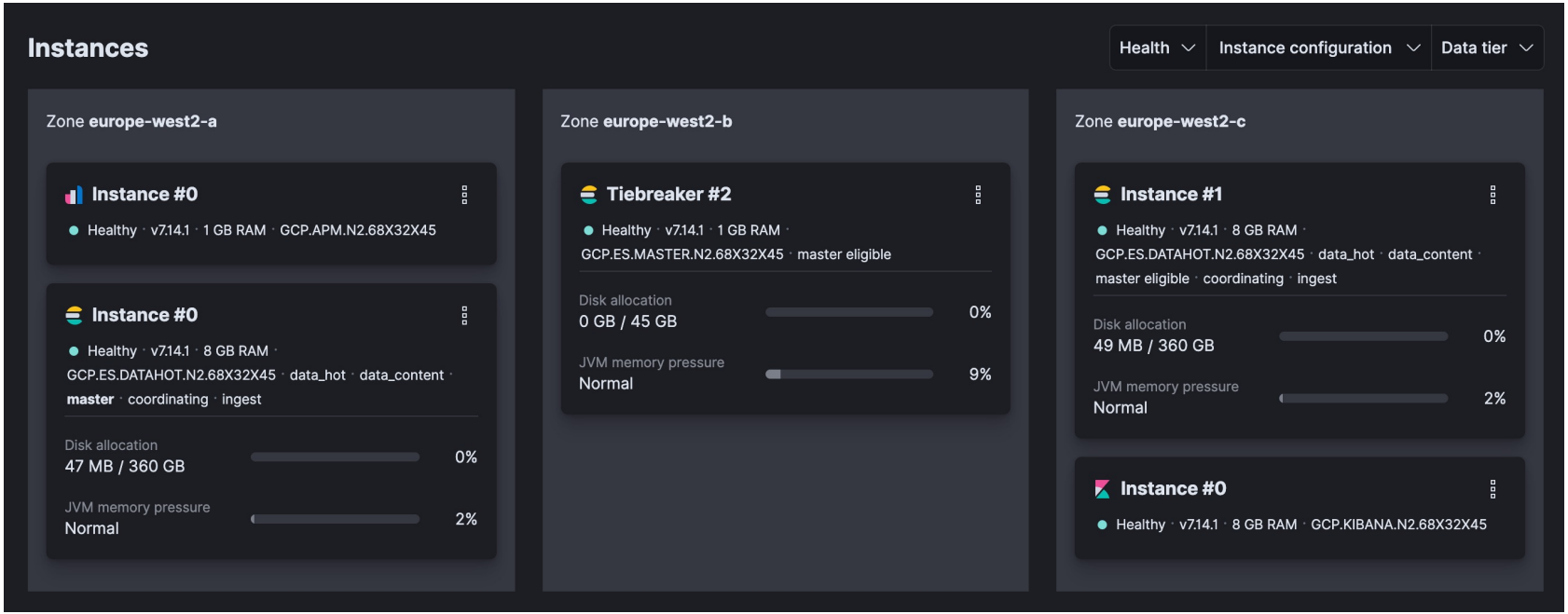
Handy components for Data Engineers

- Data Ingest Pipelines
- Data Streams
- Index Lifecycle Management (ILM)
- Data Tiers
- Painless scripts
- …SQL, EQL

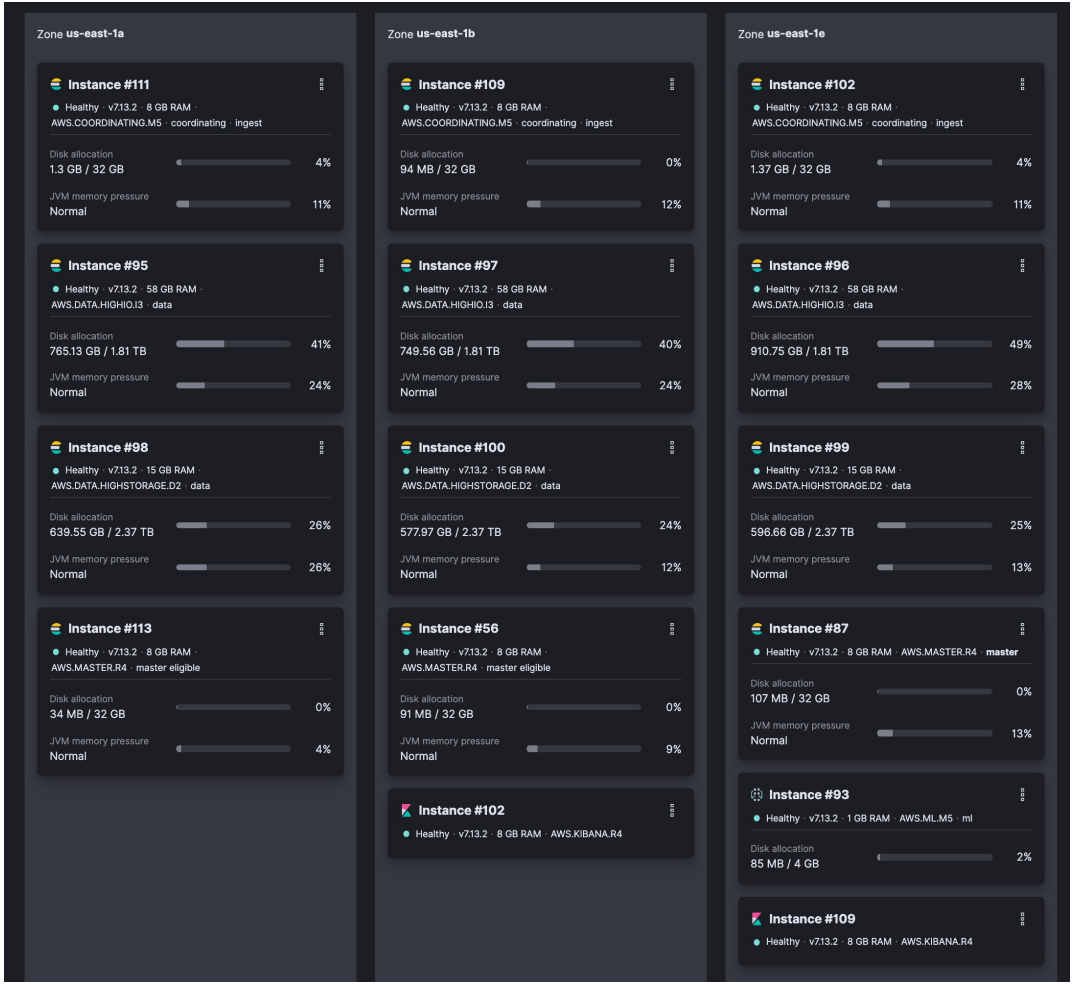
Stack Management

Topologies for pipeline observability

- Official Terraform provider available for Cloud deployments
- If you aren't using *Elasticsearch as a Service*, consider ECK/ECE
- A couple of effective topologies (hardware profiles)


Instances
Health
Instance configuration
Data tier
Zone europe-west2-a
Instance #0
Healthy · v7.14.1 · 1 GB RAM · GCP.APM.N2.68X32X45
Instance #0
Healthy · v7.14.1 · 8 GB RAM · GCP.ES.DATAHOT.N2.68X32X45 · data_hot · data_content · master · coordinating · ingest
Disk allocation
47 MB / 360 GB
0%
JVM memory pressure
Normal
2%
Zone europe-west2-b
Tiebreaker #2
Healthy · v7.14.1 · 1 GB RAM · GCP.ES.MASTER.N2.68X32X45 · master eligible
Disk allocation
0 GB / 45 GB
0%
JVM memory pressure
Normal
9%
Zone europe-west2-c
Instance #1
Healthy · v7.14.1 · 8 GB RAM · GCP.ES.DATAHOT.N2.68X32X45 · data_hot · data_content · master eligible · coordinating · ingest
Disk allocation
49 MB / 360 GB
0%
JVM memory pressure
Normal
2%
Instance #0
Healthy · v7.14.1 · 8 GB RAM · GCP.KIBANA.N2.68X32X45

elastic


Zone us-east-1a
Instance #111
Healthy · v7.13.2 · 8 GB RAM · AWS.COORDINATING.M5 · coordinating · ingest
Disk allocation
1.3 GB / 32 GB
4%
JVM memory pressure
Normal
11%
Instance #95
Healthy · v7.13.2 · 58 GB RAM · AWS.DATA.HIGHIO.I3 · data
Disk allocation
765.13 GB / 1.81 TB
41%
JVM memory pressure
Normal
24%
Instance #98
Healthy · v7.13.2 · 15 GB RAM · AWS.DATA.HIGHSTORAGE.D2 · data
Disk allocation
639.55 GB / 2.37 TB
26%
JVM memory pressure
Normal
26%
Instance #113
Healthy · v7.13.2 · 8 GB RAM · AWS.MASTER.R4 · master eligible
Disk allocation
34 MB / 32 GB
0%
JVM memory pressure
Normal
4%
Zone us-east-1b
Instance #109
Healthy · v7.13.2 · 8 GB RAM · AWS.COORDINATING.M5 · coordinating · ingest
Disk allocation
94 MB / 32 GB
0%
JVM memory pressure
Normal
12%
Instance #97
Healthy · v7.13.2 · 58 GB RAM · AWS.DATA.HIGHIO.I3 · data
Disk allocation
749.56 GB / 1.81 TB
40%
JVM memory pressure
Normal
24%
Instance #100
Healthy · v7.13.2 · 15 GB RAM · AWS.DATA.HIGHSTORAGE.D2 · data
Disk allocation
577.97 GB / 2.37 TB
24%
JVM memory pressure
Normal
12%
Instance #56
Healthy · v7.13.2 · 8 GB RAM · AWS.MASTER.R4 · master eligible
Disk allocation
91 MB / 32 GB
0%
JVM memory pressure
Normal
9%
Instance #102
Healthy · v7.13.2 · 8 GB RAM · AWS.KIBANA.R4
Zone us-east-1e
Instance #102
Healthy · v7.13.2 · 8 GB RAM · AWS.COORDINATING.M5 · coordinating · ingest
Disk allocation
1.37 GB / 32 GB
4%
JVM memory pressure
Normal
11%
Instance #96
Healthy · v7.13.2 · 58 GB RAM · AWS.DATA.HIGHIO.I3 · data
Disk allocation
910.75 GB / 1.81 TB
49%
JVM memory pressure
Normal
28%
Instance #99
Healthy · v7.13.2 · 15 GB RAM · AWS.DATA.HIGHSTORAGE.D2 · data
Disk allocation
596.66 GB / 2.37 TB
25%
JVM memory pressure
Normal
13%
Instance #87
Healthy · v7.13.2 · 8 GB RAM · AWS.MASTER.R4 · master
Disk allocation
107 MB / 32 GB
0%
JVM memory pressure
Normal
13%
Instance #93
Healthy · v7.13.2 · 1 GB RAM · AWS.ML.M5 · ml
Disk allocation
85 MB / 4 GB
2%
Instance #109
Healthy · v7.13.2 · 8 GB RAM · AWS.KIBANA.R4

elastic

Case Study 1

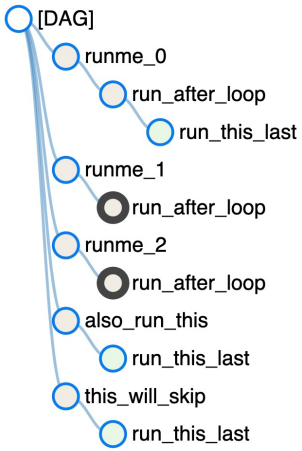
Batch

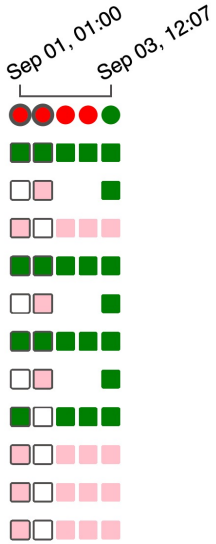
Common Pattern

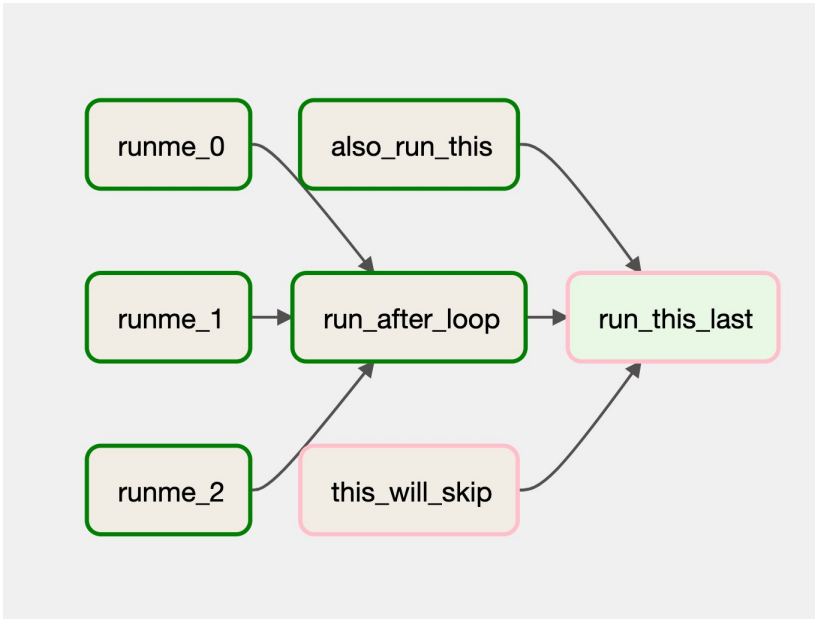
Ingestion to Storage

Batch Architecture

Simple



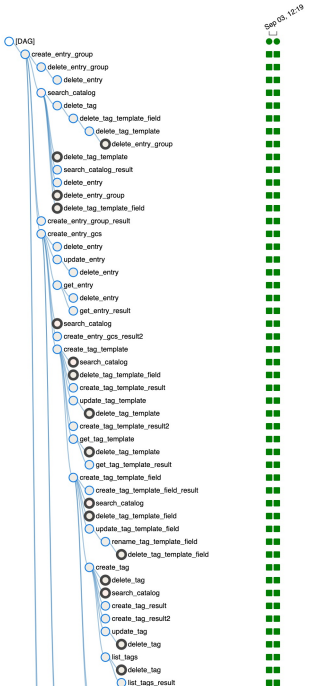


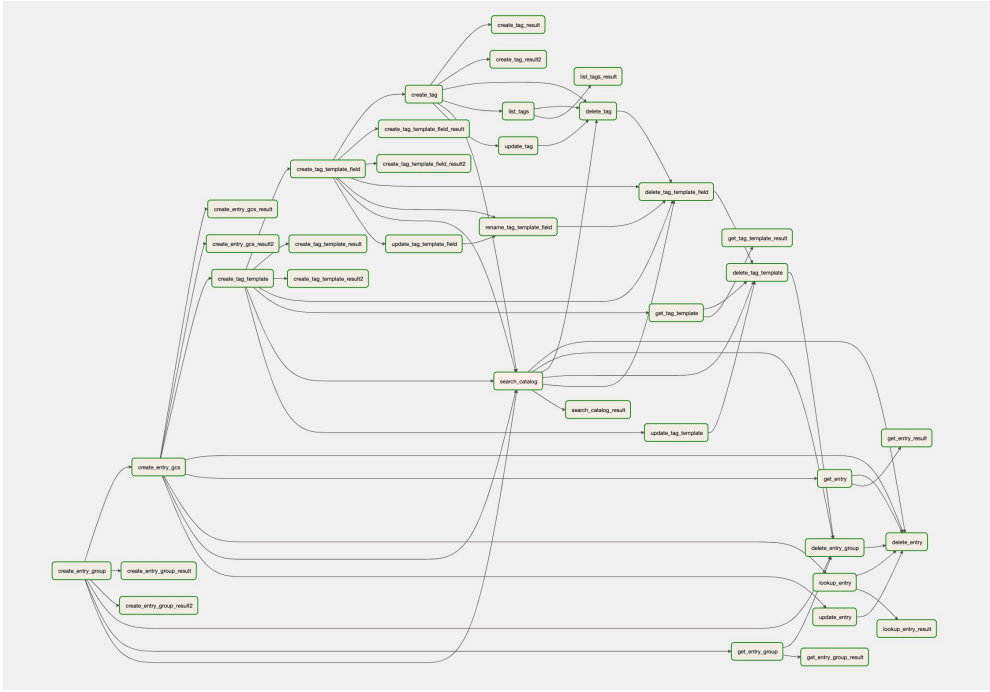


elastic

Batch Example

Not so simple





elastic

Logs?

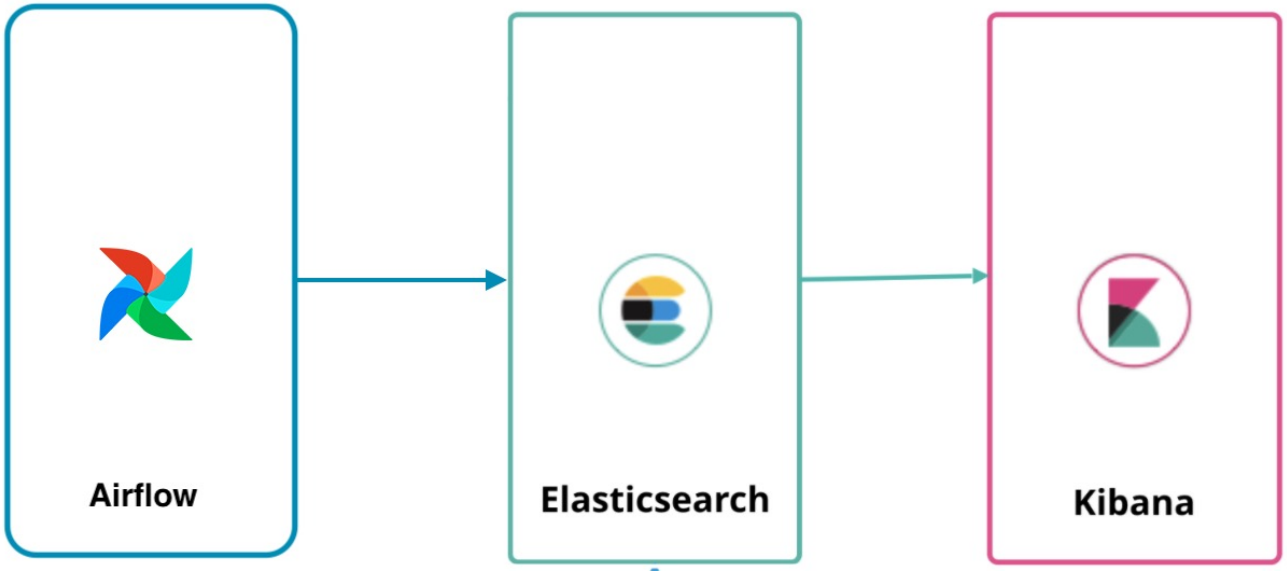
First line of defense

- Collect them if you can

- How to ship though? 🤔

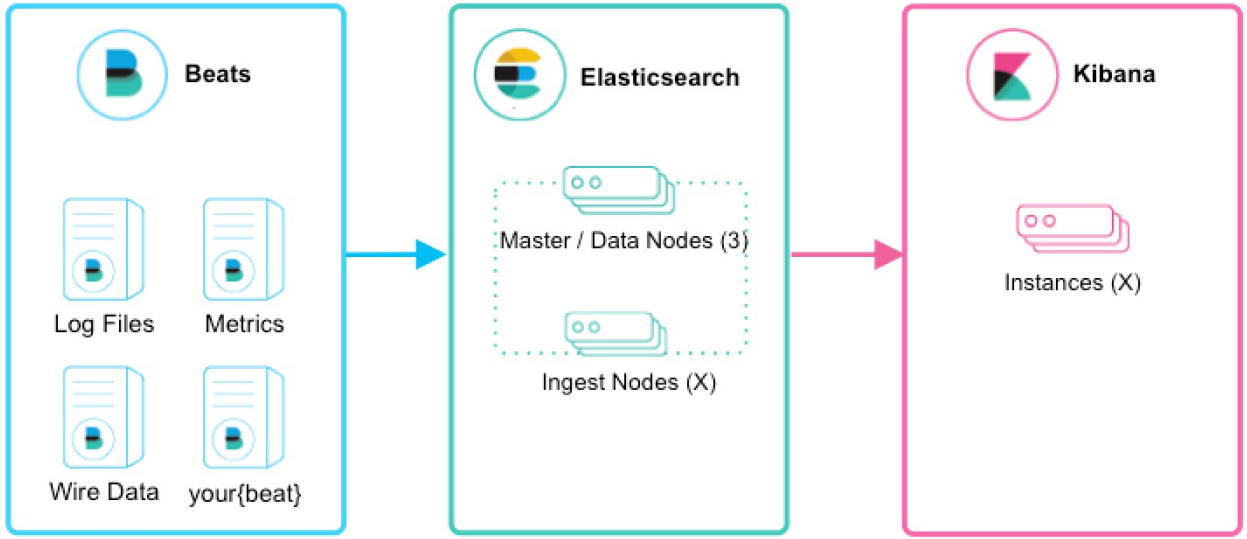
Basic Setup

Orchestrator to Elasticsearch



Basic Setup

Beats to Elasticsearch



Logs

Get the most out of your logs

- More isn't always better
- Structure logs and make consistent across services
- Build in contextualization
- Develop a schema, or adopt an existing one (https://github.com/elastic/ecs)

```
abstract class Event(
    exception: Exception? = null,
    throwable: Throwable? = null,
    val level: LogLevel = LogLevel.DEBUG,
    val message: String? = null,
    val reason: String? = null,
    val statusCode: String? = null,
    val warning: String? = null
) {
    val application: String? = applicationLocal.get()
    val caller: String = Thread.currentThread().name
    val correlationId: String? = contextualLocal.get()
    val event: String = this.javaClass.simpleName
    val exceptionMessage: String? = exception?.message
    val thread: String = Thread.currentThread().name
    val time: Long = System.currentTimeMillis()
}
```

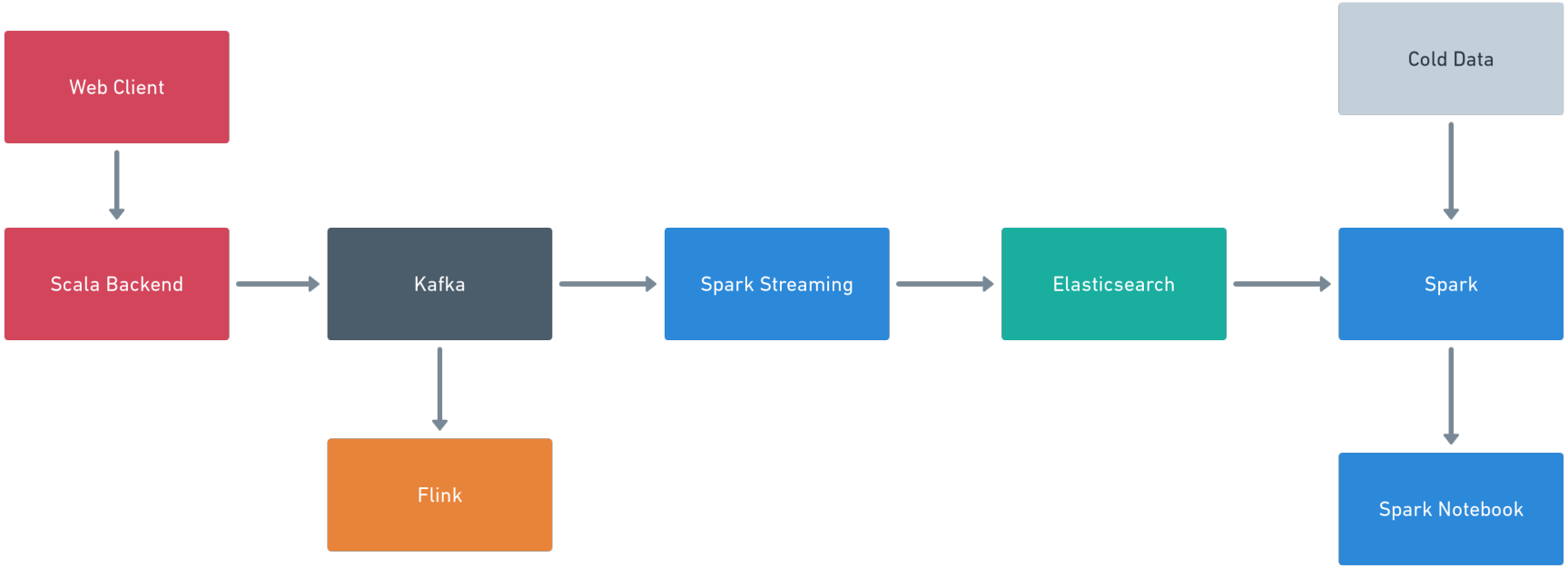
```
class BadRequestReceived(message: String) : Event(
    level = LogLevel.ERROR,
    message = message
)

class CleaningUpAsyncWorker(name: String) : Event(
    level = LogLevel.INFO,
    message = "cleaning up async worker $name"
)
```

Case Study 2

Streaming

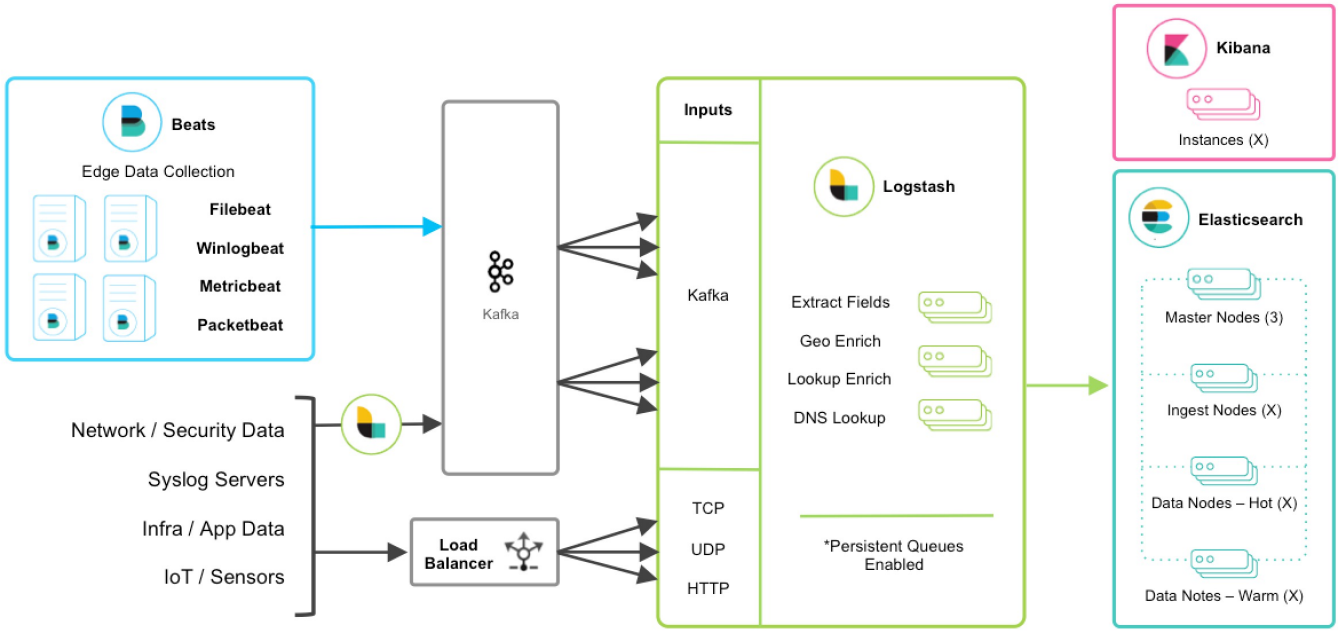
Example Streaming Architecture



elastic

Advanced Ingestion Setup

Additional On / Off Stack infrastructure

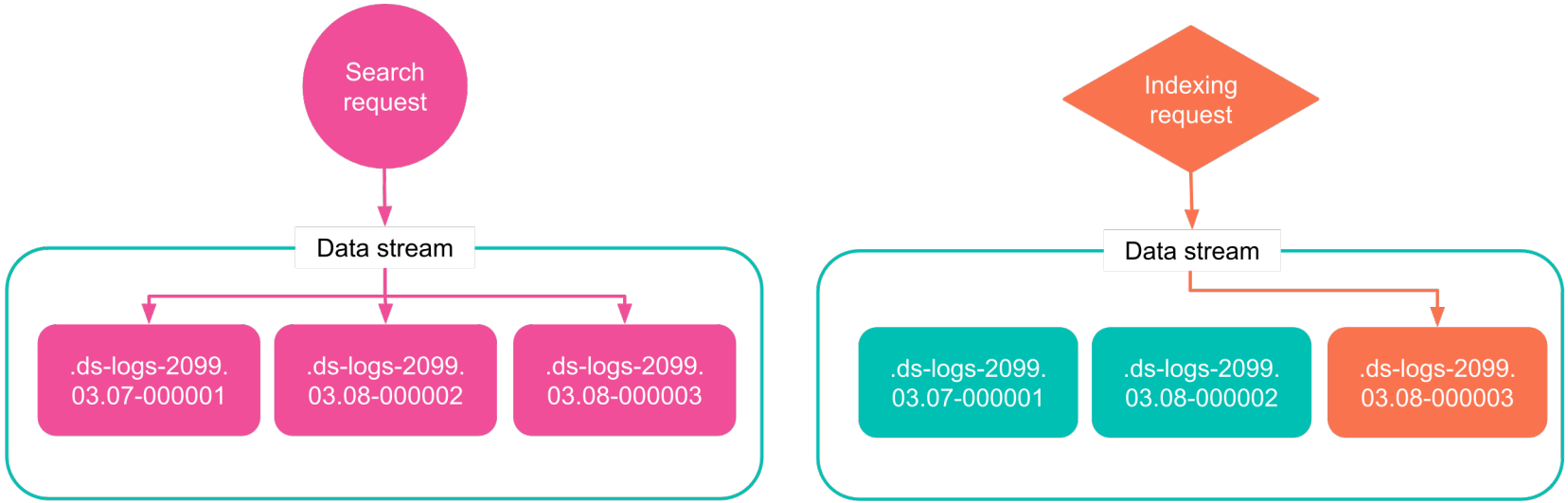


elastic

Elasticsearch Data Streams

- Append only
- Great for logs, metrics and other continuously generated data
- Can use Index Lifecycle Management (ILM) to automate index management
- ILM can help reduce costs significantly

Elasticsearch Data Streams



elastic

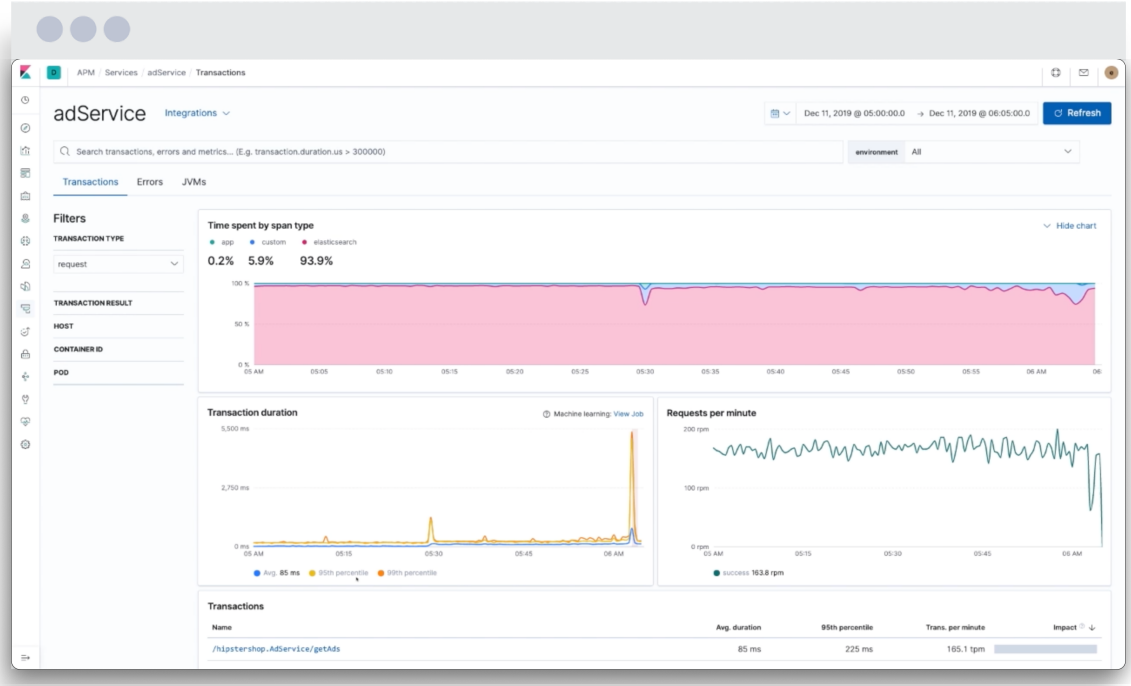
Homebrew Stream Processing

- Some teams don’t wish to use *Big Data* tooling
- Bring your own code / tools

Advanced Stack Usage

APM & Machine Learning

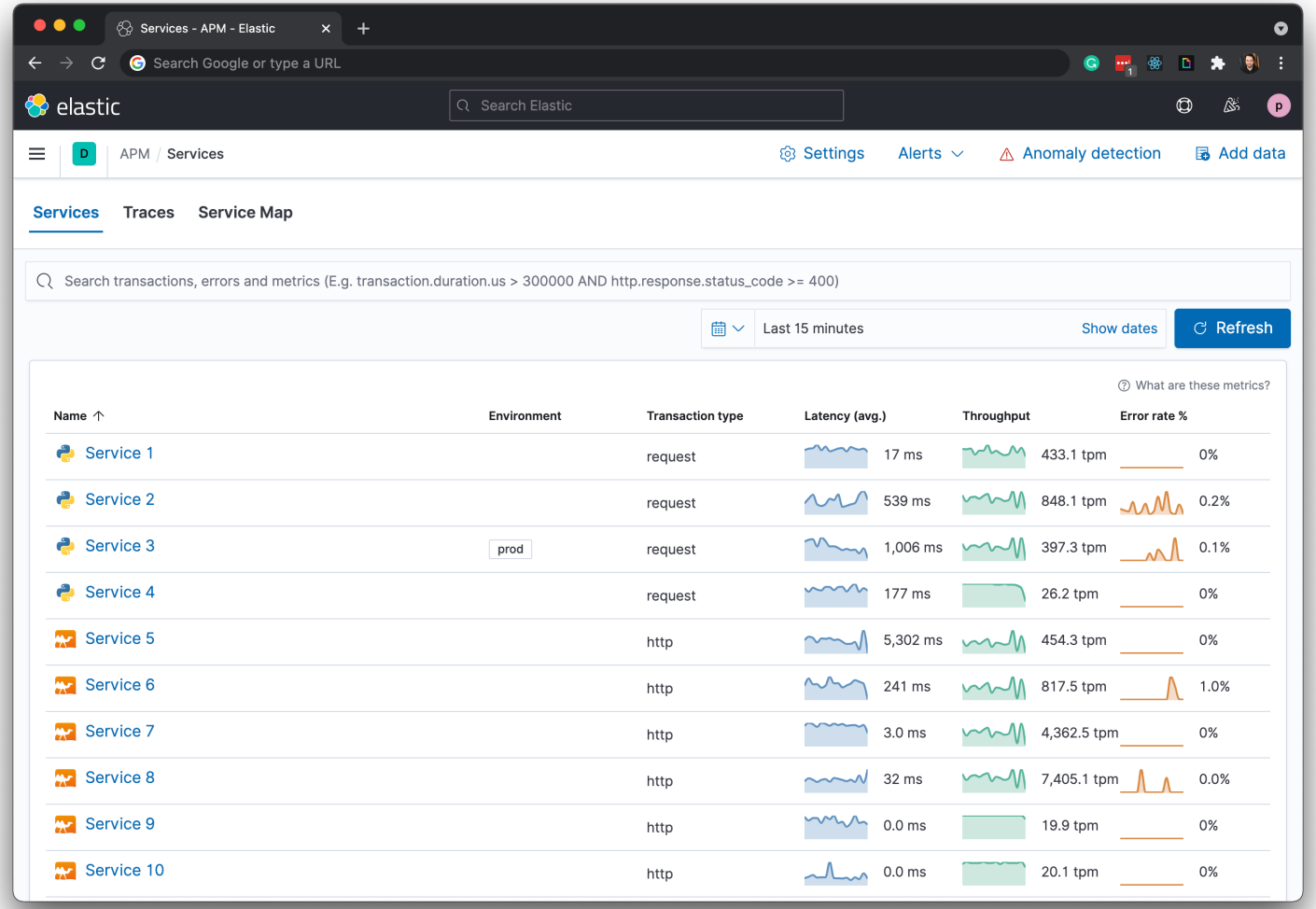
Elastic APM

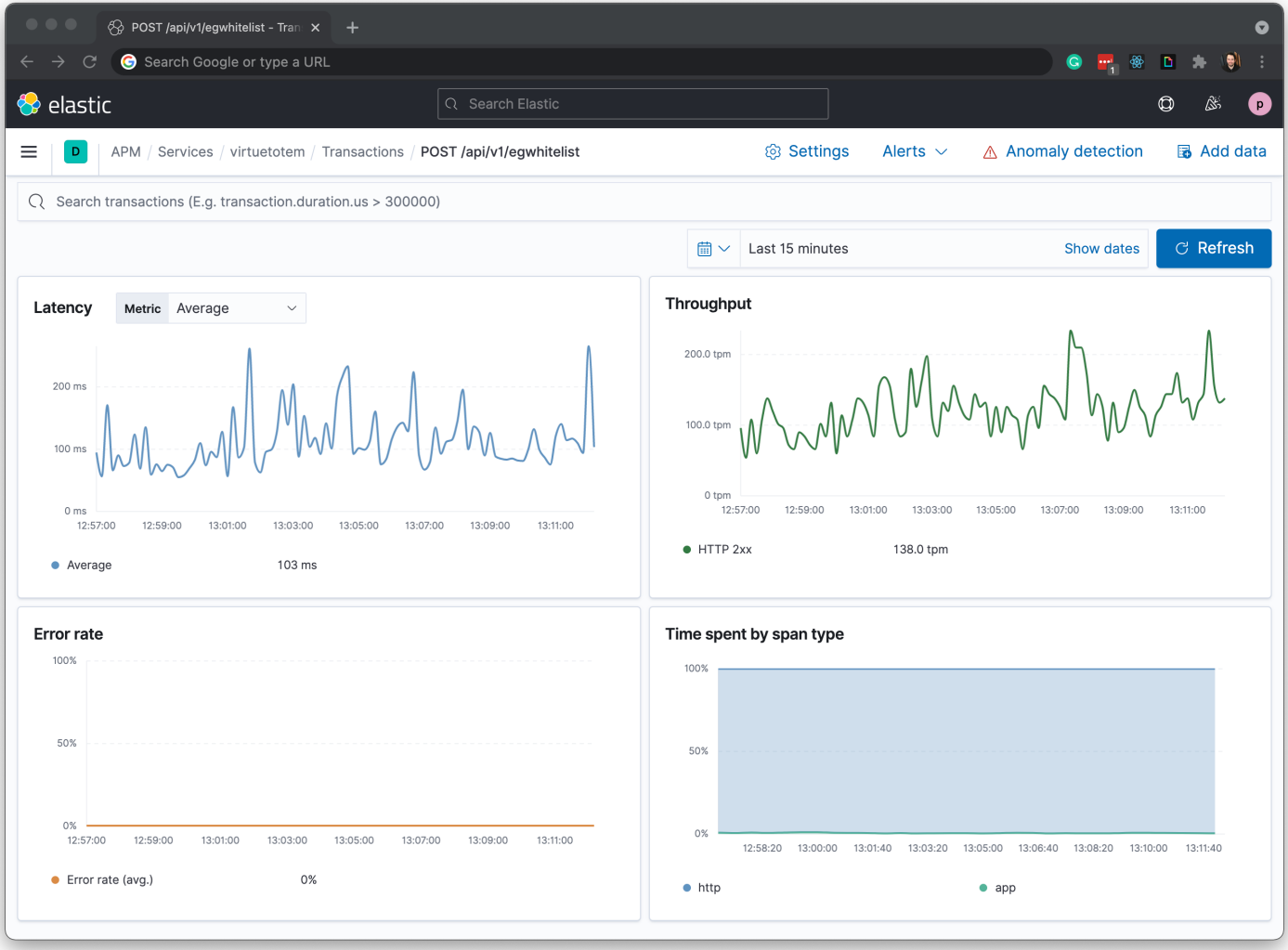

APM / Services / adService / Transactions
adService
Integrations
Dec 11, 2019 @ 05:00:00.0
Dec 11, 2019 @ 06:05:00.0
Refresh
Search transactions, errors and metrics... (E.g. transaction.duration.us > 300000)
environment
All
Transactions
Errors
JVMs
Filters
TRANSACTION TYPE
request
TRANSACTION RESULT
HOST
CONTAINER ID
POD
Time spent by span type
Hide chart
app
custom
elasticsearch
0.2%
5.9%
93.9%
Transaction duration
Machine learning: View Job
Requests per minute
Transactions
Name
Avg. duration
95th percentile
Trans. per minute
Impact
/hipstershop.AdService/getAds
85 ms
225 ms
165.1 tpm

Elastic APM

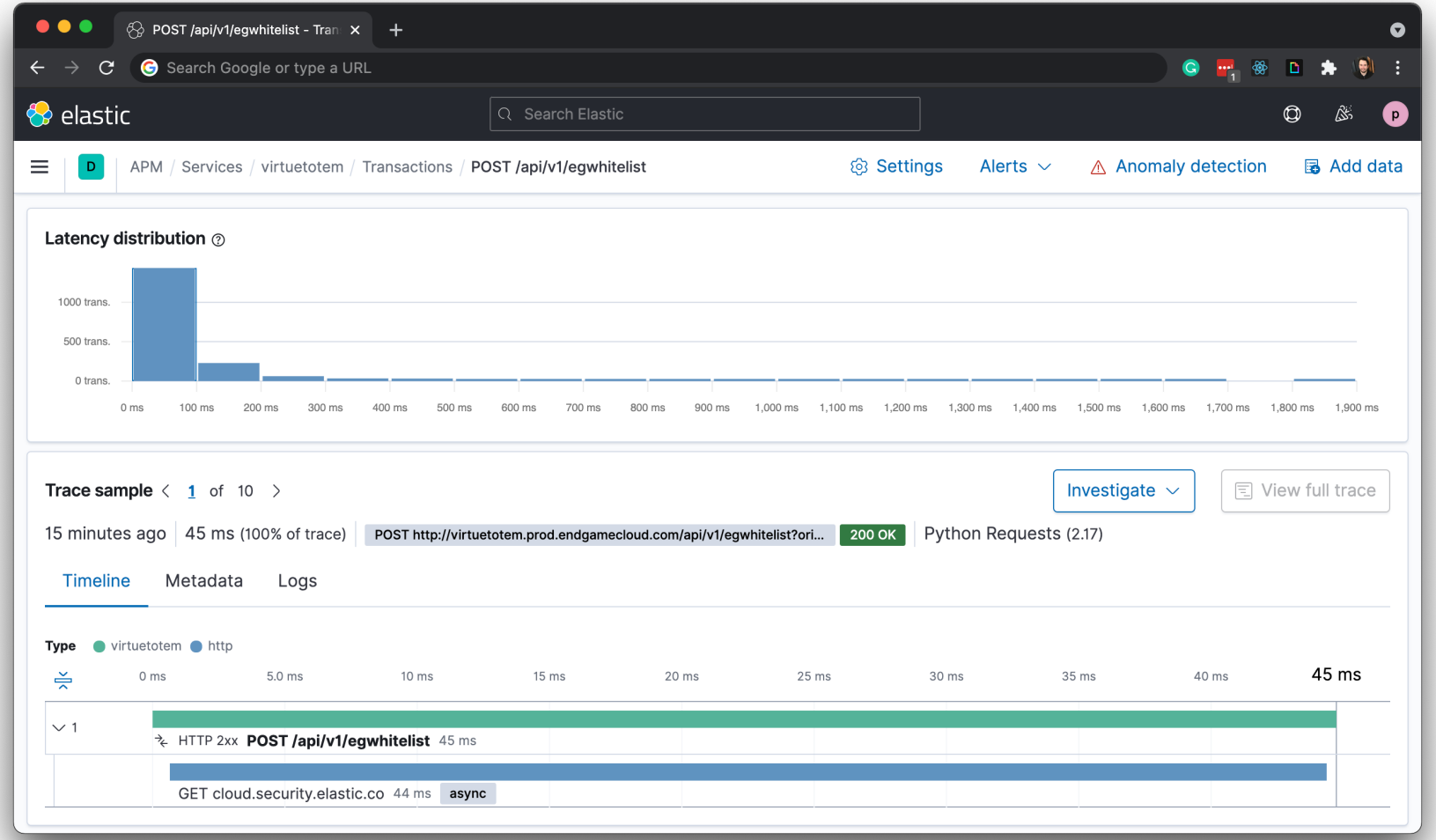
Profile pipeline and stream processing components

- Check service and infrastructure uptime and health
- Service Maps & Distributed tracing
- Monitor TLS Cert expiry
- Alert when things go wrong (& create cases!)
- Data Engineering friendly
  - Supports integrations such as Kafka, RabbitMQ, …
  - Programming language support – Java, Python, Go, …
  - Community support for OCaml, Erlang and others


Services - APM - Elastic
Search Google or type a URL
elastic
Search Elastic
APM / Services
Settings
Alerts
Anomaly detection
Add data
Services
Traces
Service Map
Search transactions, errors and metrics (E.g. transaction.duration.us > 300000 AND http.response.status_code >= 400)
Last 15 minutes
Show dates
Refresh
What are these metrics?
Name
Environment
Transaction type
Latency (avg.)
Throughput
Error rate %
Service 1 request 17 ms 433.1 tpm 0%
Service 2 request 539 ms 848.1 tpm 0.2%
Service 3 prod request 1,006 ms 397.3 tpm 0.1%
Service 4 request 177 ms 26.2 tpm 0%
Service 5 http 5,302 ms 454.3 tpm 0%
Service 6 http 241 ms 817.5 tpm 1.0%
Service 7 http 3.0 ms 4,362.5 tpm 0%
Service 8 http 32 ms 7,405.1 tpm 0.0%
Service 9 http 0.0 ms 19.9 tpm 0%
Service 10 http 0.0 ms 20.1 tpm 0%

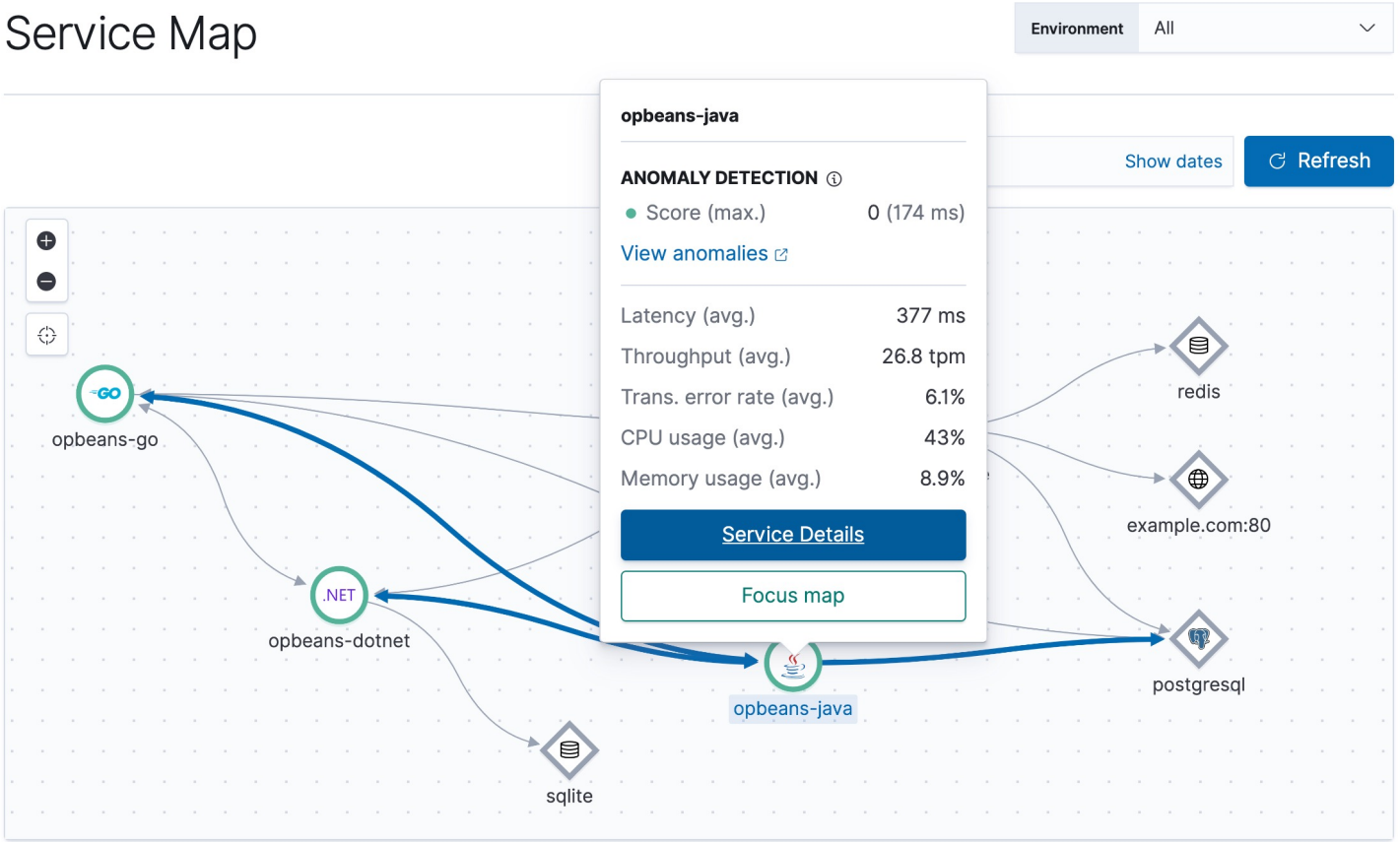

POST /api/v1/egwhitelist - Tran
Search Google or type a URL
elastic
Search Elastic
APM / Services / virtuetotem / Transactions / POST /api/v1/egwhitelist
Settings
Alerts
Anomaly detection
Add data
Search transactions (E.g. transaction.duration.us > 300000)
Last 15 minutes
Show dates
Refresh
Latency
Metric
Average
200 ms
100 ms
0 ms
12:57:00
12:59:00
13:01:00
13:03:00
13:05:00
13:07:00
13:09:00
13:11:00
Average
103 ms
Throughput
200.0 tpm
100.0 tpm
0 tpm
HTTP 2xx
138.0 tpm
Error rate
100%
50%
0%
Error rate (avg.)
0%
Time spent by span type
12:58:20
13:00:00
13:01:40
13:03:20
13:05:00
13:06:40
13:08:20
13:10:00
13:11:40
http
app

elastic


POST /api/v1/egwhitelist - Tran
Search Google or type a URL
elastic
Search Elastic
APM / Services / virtuetotem / Transactions / POST /api/v1/egwhitelist
Settings
Alerts
Anomaly detection
Add data
Latency distribution
1000 trans.
500 trans.
0 trans.
0 ms 100 ms 200 ms 300 ms 400 ms 500 ms 600 ms 700 ms 800 ms 900 ms 1,000 ms 1,100 ms 1,200 ms 1,300 ms 1,400 ms 1,500 ms 1,600 ms 1,700 ms 1,800 ms 1,900 ms
Trace sample 1 of 10
Investigate
View full trace
15 minutes ago
45 ms (100% of trace)
POST http://virtuetotem.prod.endgamecloud.com/api/v1/egwhitelist?ori...
200 OK
Python Requests (2.17)
Timeline
Metadata
Logs
Type virtuetotem http
0 ms 5.0 ms 10 ms 15 ms 20 ms 25 ms 30 ms 35 ms 40 ms 45 ms
1
HTTP 2xx POST /api/v1/egwhitelist 45 ms
GET cloud.security.elastic.co 44 ms
async

elastic

Service Map



elastic

Machine Learning

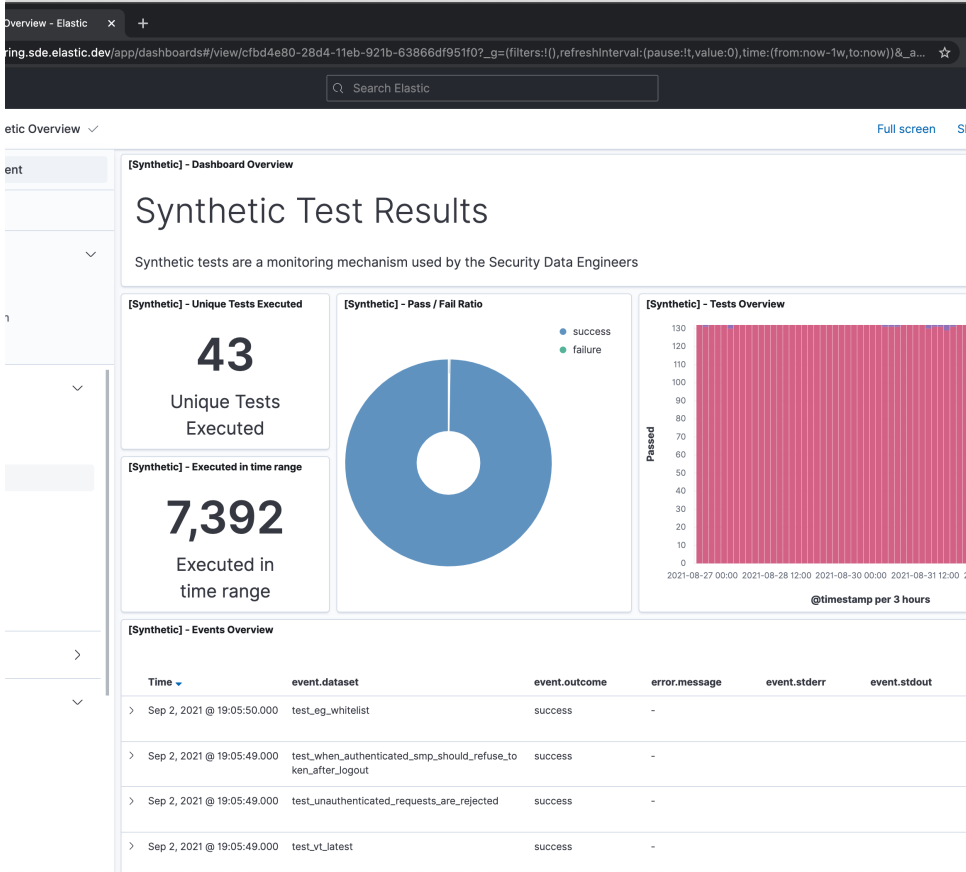
Unsupervised analysis of APM Data

- Highlights anomalies
- Deep insight into performance of individual service + trace
- Spot issues before they happen

Capturing Tests

Monitoring Synthetic events

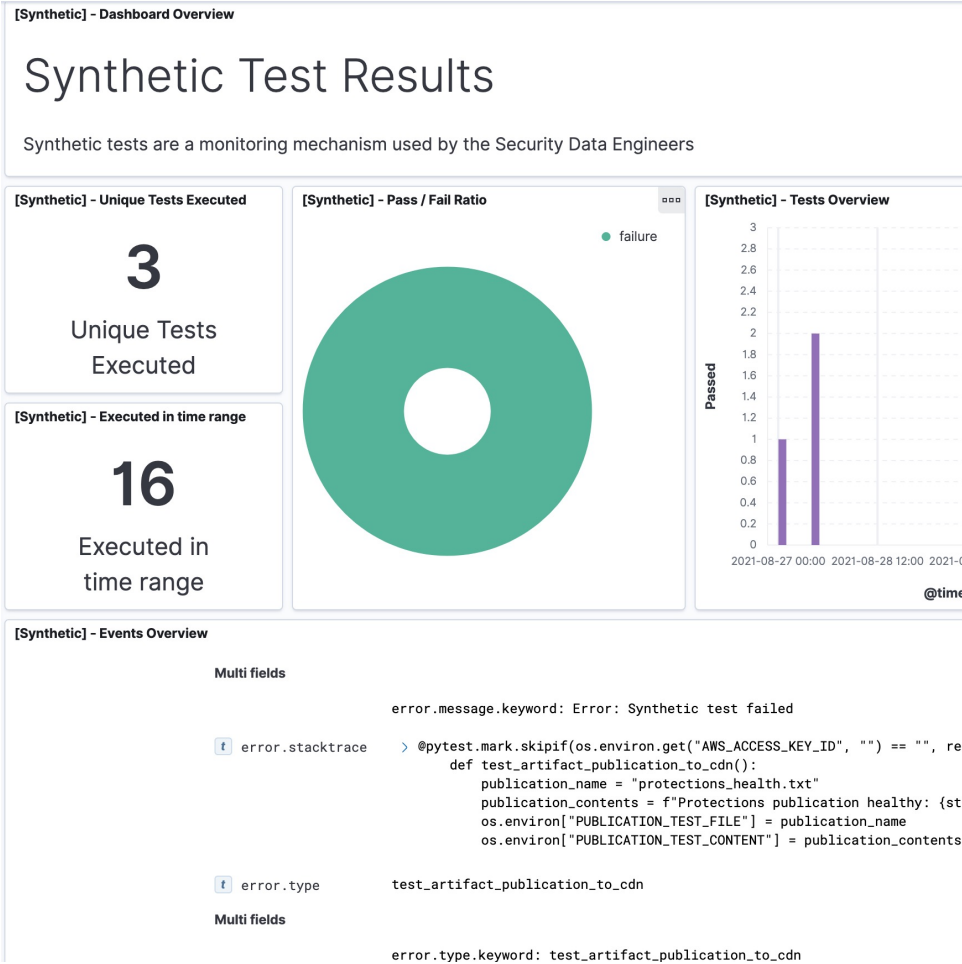
- Run tests on a temporal cadence
- Insert test output into Elasticsearch
- Create simple visualizations and dashboards



Components Fail

Context matters

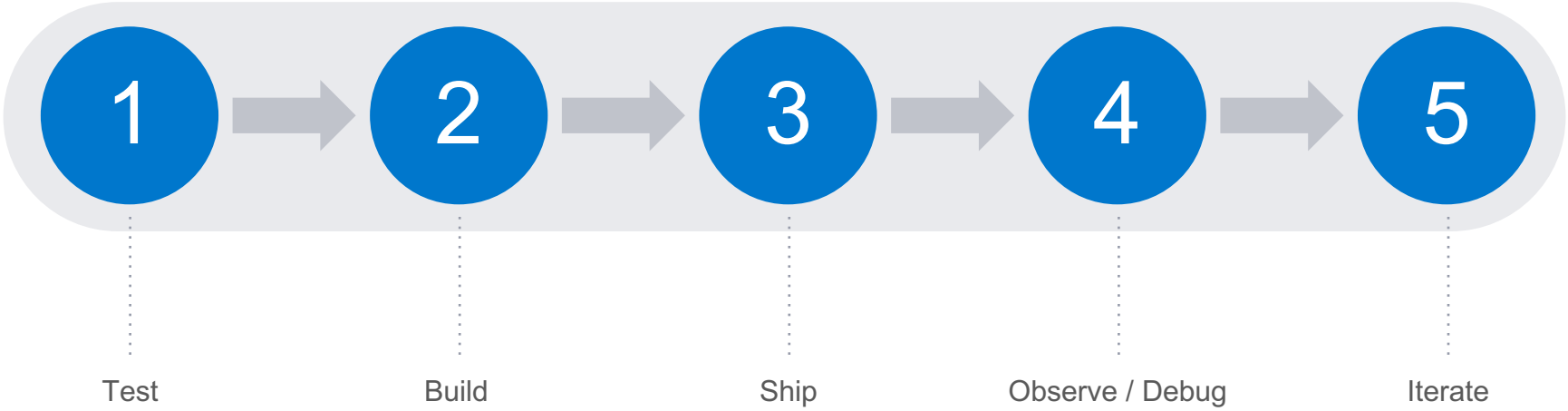
- Determine trends in services not meeting our SLAs



elastic

Summary

Feedback loops are key



Takeaways

- Elastic Stack helps operate data workloads
- Understand data integration and processing
- Use APM & ML for enhanced observability
- Lots of great integrations available
  (Public Cloud, JMS, JDBC, Vault, Zookeeper, …)

elastic | ON
GLOBAL 2021

Thanks!

https://github.com/pjhampton