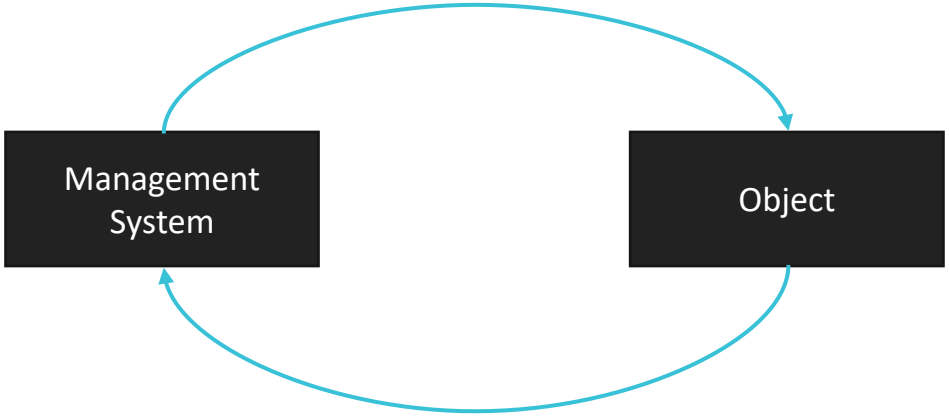
CI/CD

Monitoring

Overview

General Concept

Lifecycle management



Service Level

An **SLI** is a service level *indicator*—a carefully defined quantitative measure of some aspect of the level of service that is provided:
- request latency
- system throughput
- availability
- etc.

An **SLO** is a service level objective: a target value or range of values for a service level that is measured by an SLI. A natural structure for SLOs is thus SLI ≤ target, or lower bound ≤ SLI ≤ upper bound.

SLAs are service level agreements: an explicit or implicit contract with your users that includes consequences of meeting (or missing) the SLOs they contain.

Definitions

- Monitoring

Collecting, processing, aggregating, and displaying real-time quantitative data about a system, such as query counts and types, error counts and types, processing times, and server lifetimes.

- White-box monitoring

Monitoring based on metrics exposed by the internals of the system, including logs, interfaces like the Java Virtual Machine Profiling Interface, or an HTTP handler that emits internal statistics.

- Black-box monitoring

Testing externally visible behavior as a user would see it.

- Dashboard

An application (usually web-based) that provides a summary view of a service's core metrics. A dashboard may have filters, selectors, and so on, but is prebuilt to expose the metrics most important to its users. The dashboard might also display team information such as ticket queue length, a list of high-priority bugs, the current on-call engineer for a given area of responsibility, or recent pushes.

Definitions

- Alert

A notification intended to be read by a human and that is pushed to a system such as a bug or ticket queue, an email alias, or a pager. Respectively, these alerts are classified as *tickets*, *email alerts*, and *pages*.

- Root cause

A defect in a software or human system that, if repaired, instills confidence that this event won't happen again in the same way. A given incident might have multiple root causes: for example, perhaps it was caused by a combination of insufficient process automation, software that crashed on bogus input, *and* insufficient testing of the script used to generate the configuration. Each of these factors might stand alone as a root cause, and each should be repaired.

- Node and Machine

Used interchangeably to indicate a single instance of a running kernel in either a physical server, virtual machine, or container. There might be multiple *services* worth monitoring on a single machine. The services may either be:

- Related to each other: for example, a caching server and a web server
- Unrelated services sharing hardware: for example, a code repository and a master for a configuration system like Puppet or Chef

Why monitor?

- Analyzing long-term trends
- Comparing over time or experiment groups
- Alerting
- Building dashboards
- Conducting *ad hoc* retrospective analysis (i.e., debugging)

The Four Golden Signals

- Latency
- Traffic
- Errors
- Saturation

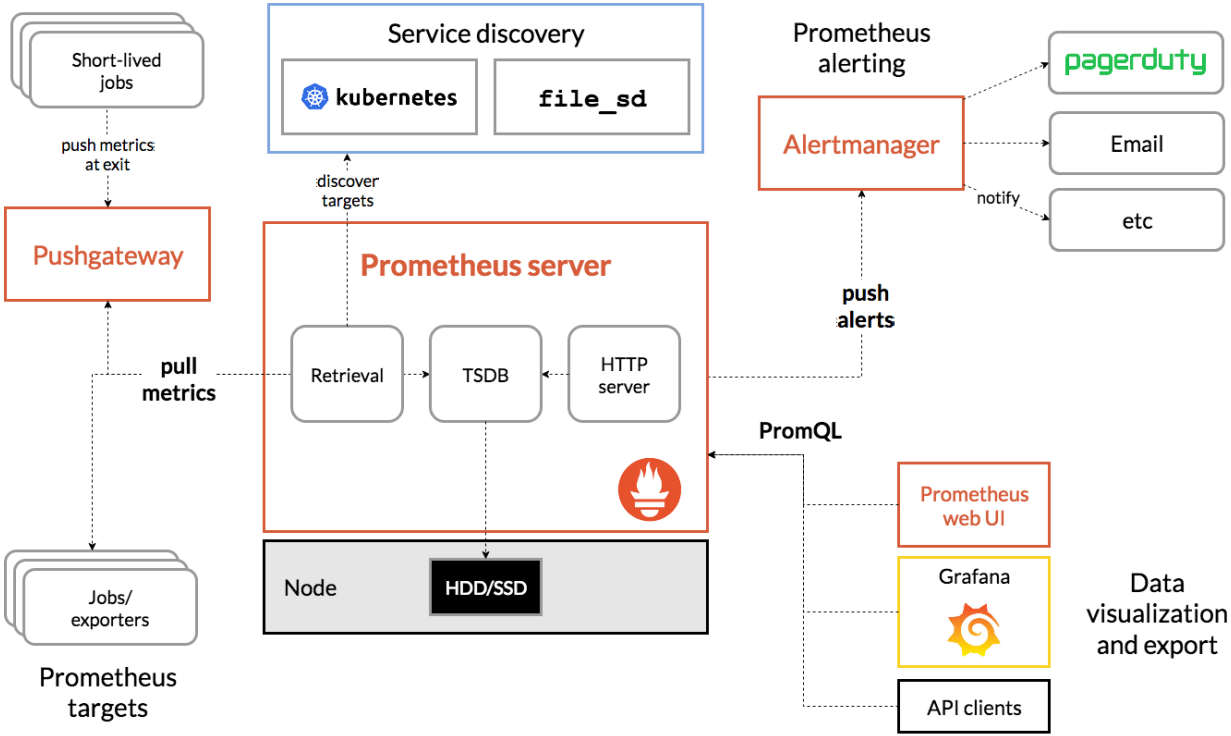
Prometheus

/pɹəˈmiːθi.əs/

Time series database

A time series database (TSDB) is a software system that is optimized for storing and serving time series through associated pairs of time(s) and value(s). In some fields, time series may be called profiles, curves, traces or trends.

Architecture



Terms and definitions

Target
A target is the definition of an object to scrape. For example, what labels to apply, any authentication required to connect, or other information that defines how the scrape will occur.

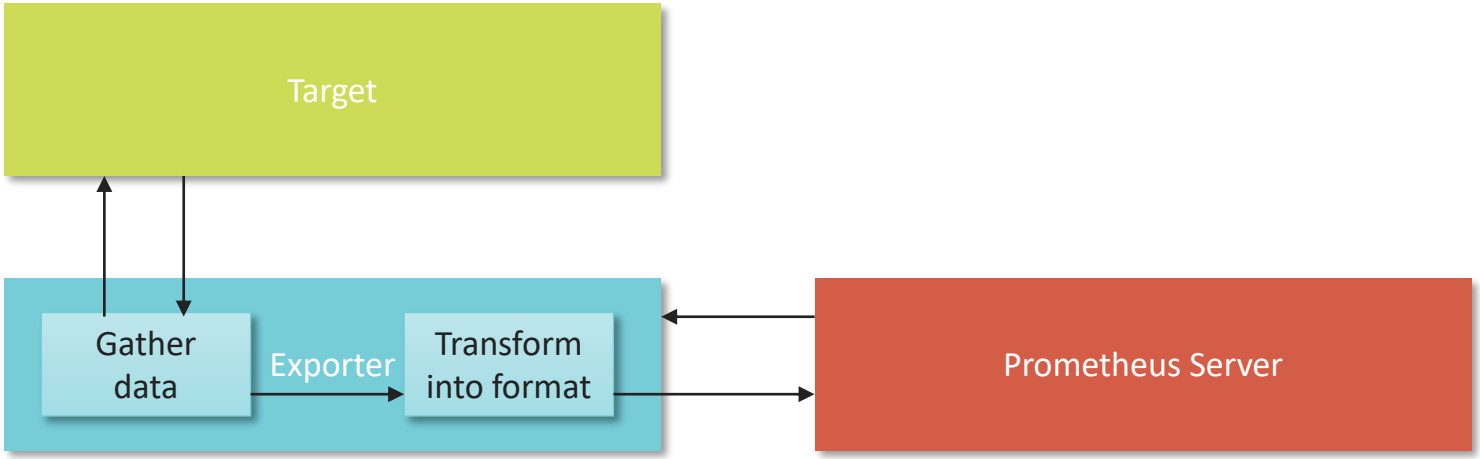
Instance
An instance is a label that uniquely identifies a target in a job.
`For ex.: 127.0.0.1:9090, 192.168.99.103:10250, etc.`

Job
A collection of targets with the same purpose, for example monitoring a group of like processes replicated for scalability or reliability, is called a job.
For ex.:

```
job=“api-server“:
          instance1: 192.168.99.103:8443
          instance2: 192.168.99.104:8443
```

Sample
A sample is a single value at a point in time in a time series.
In Prometheus, each sample consists of a float64 value and a millisecond-precision timestamp.
For ex.: `prometheus_http_requests_total{code="200",handler="/metrics"} 44`

Exporter


Target
Gather data
Exporter
Transform into format
Prometheus Server

Prometheus - Configuration

```
/prometheus $ cat /etc/prometheus/prometheus.yml
# my global config
global:
  scrape_interval:     15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: 'prometheus'

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
    - targets: ['localhost:9090']
```

PromQL – prometheus query language

In Prometheus's expression language, an expression or sub-expression can evaluate to one of four types:

•**Instant vector** - a set of time series containing a single sample for each time series, all sharing the same timestamp

```
For ex.: http_requests_total
```

•**Range vector** - a set of time series containing a range of data points over time for each time series

For ex.: http_requests_total{job="prometheus"}[5m]

•**Scalar** - a simple numeric floating point value

For ex.: 2396.01

•**String** - a simple string value; currently unused

For ex.: i_am_a_string

Recording rules

Recording rules allow you to precompute frequently needed or computationally expensive expressions and save their result as a new set of time series. Querying the precomputed result will then often be much faster than executing the original expression every time it is needed. This is especially useful for dashboards, which need to query the same expression repeatedly every time they refresh.

```
groups:
  - name: my-rule
    rules:
    - record: job:http_inprogress_requests:sum
      expr: sum by (job) (http_inprogress_requests)
```

Data structure

http://<Prometheus_ip>:9090/metrics

```
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 4.2874e-05
go_gc_duration_seconds{quantile="0.25"} 5.6063e-05
go_gc_duration_seconds{quantile="0.5"} 7.9763e-05
go_gc_duration_seconds{quantile="0.75"} 0.000209804
go_gc_duration_seconds{quantile="1"} 0.040763561
go_gc_duration_seconds_sum 0.196046782
go_gc_duration_seconds_count 228
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 233
```

Grafana

https://grafana.com/grafana/dashboards

Document-oriented database

A document-oriented database is a specific kind of database that works on the principle of dealing with 'documents’ rather than strictly defined tables of information.

The document-oriented database plays an important role is aggregating data from documents and getting them into a searchable, organized form.

Grafana

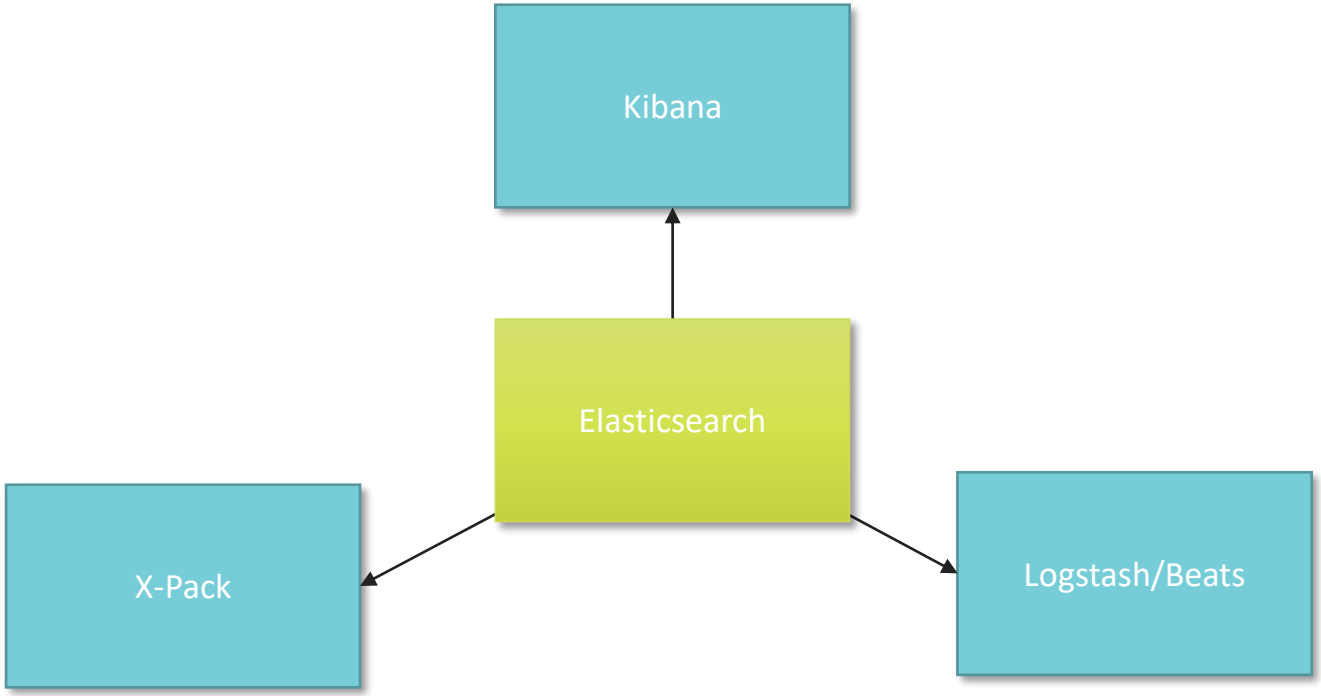
Grafana allows you to query, visualize, alert on and understand your metrics no matter where they are stored. Create, explore, and share dashboards with your team and foster a data driven culture:

- **Visualize:** Fast and flexible client side graphs with a multitude of options. Panel plugins for many different way to visualize metrics and logs.
- **Dynamic Dashboards:** Create dynamic & reusable dashboards with template variables that appear as dropdowns at the top of the dashboard.
- **Explore Metrics:** Explore your data through ad-hoc queries and dynamic drilldown. Split view and compare different time ranges, queries and data sources side by side.
- **Explore Logs:** Experience the magic of switching from metrics to logs with preserved label filters. Quickly search through all your logs or streaming them live.
- **Alerting:** Visually define alert rules for your most important metrics. Grafana will continuously evaluate and send notifications to systems like Slack, PagerDuty, VictorOps, OpsGenie.
- **Mixed Data Sources:** Mix different data sources in the same graph! You can specify a data source on a per-query basis. This works for even custom datasources.

Elastic stack

Elasticsearch Filebeat Kibana

The Elastic Stack



Elasticsearch

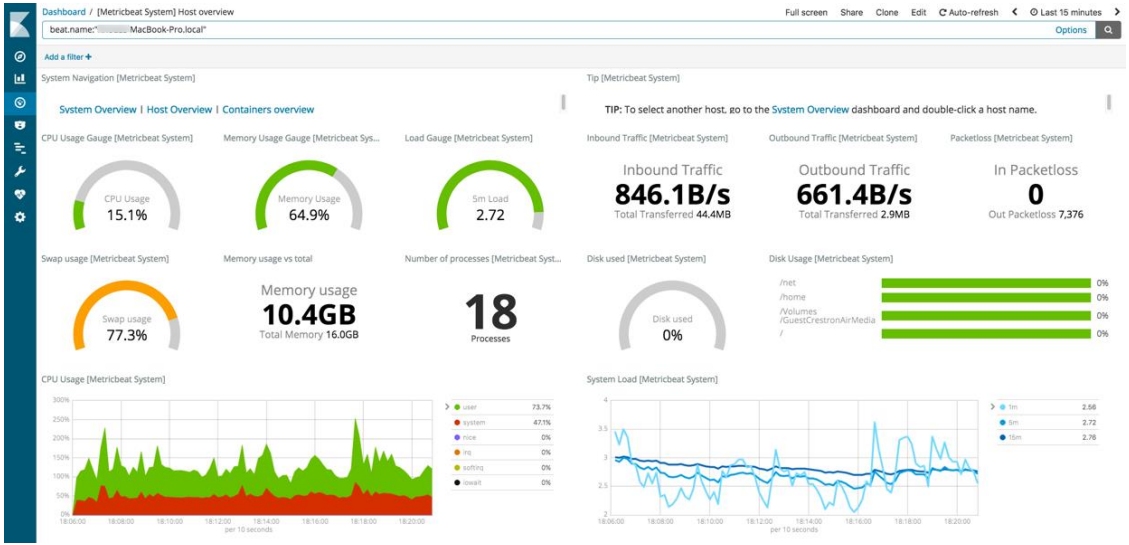
Elasticsearch is a real-time, distributed storage, search, and analytics engine. It can be used for many purposes, but one context where it excels is indexing streams of semi-structured data, such as logs or decoded network packets.| Elasticsearch is the central component of the Elastic Stack, a set of open source tools for data ingestion, enrichment, storage, analysis, and visualization.

The speed and scalability of Elasticsearch and its ability to index many types of content mean that it can be used for a number of use cases:

- Application search
- Website search
- Enterprise search
- Logging and log analytics
- Infrastructure metrics and container monitoring
- Application performance monitoring
- Geospatial data analysis and visualization
- Security analytics
- Business analytics

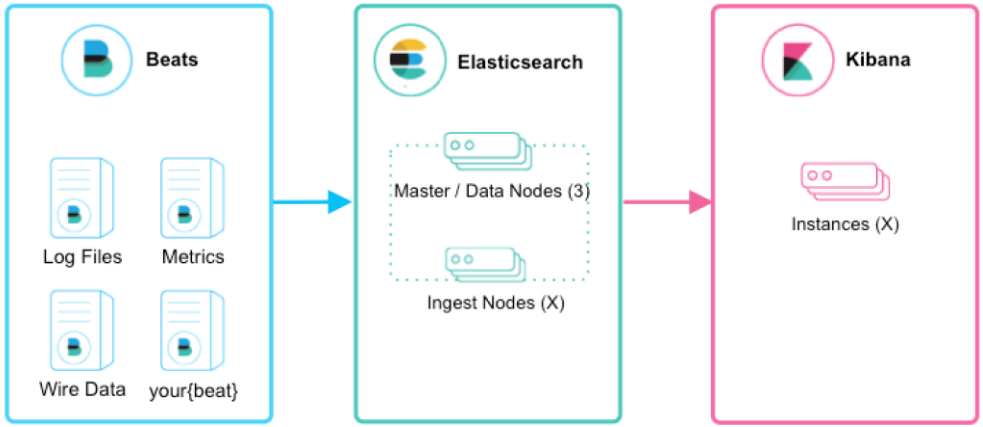
Kibana

Kibana is an open source analytics and visualization platform designed to work with Elasticsearch. You use Kibana to search, view, and interact with data stored in Elasticsearch indices. You can easily perform advanced data analysis and visualize your data in a variety of charts, tables, and maps.
Kibana is a data visualization and management tool for Elasticsearch that provides real-time histograms, line graphs, pie charts, and maps. Kibana also includes advanced applications such as Canvas, which allows users to create custom dynamic infographics based on their data, and Elastic Maps for visualizing geospatial data.



Beats

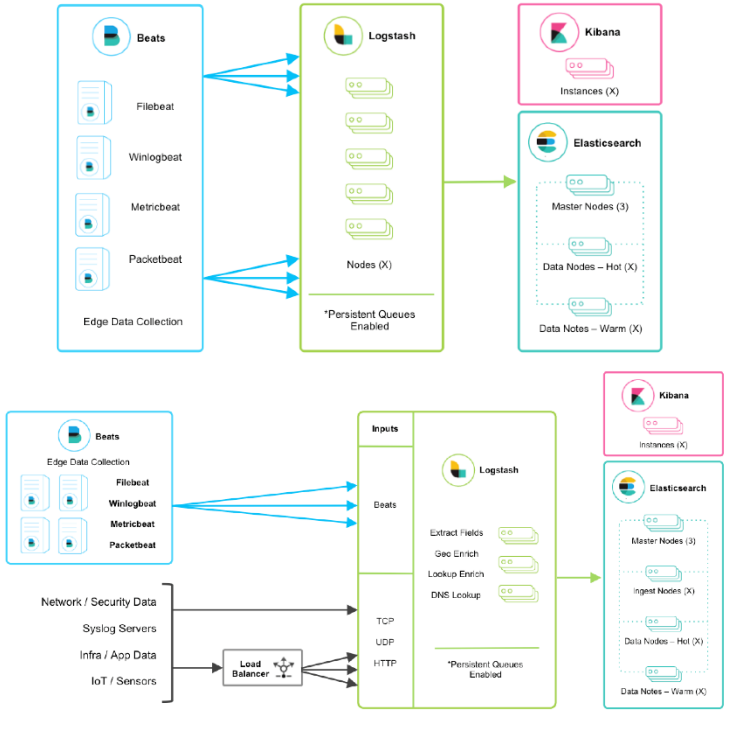
The Beats are open source data shippers that you install as agents on your servers to send operational data to Elasticsearch. Beats can send data directly to Elasticsearch or via Logstash, where you can further process and enhance the data.
Beats run across thousands of edge host servers, collecting, tailing, and shipping logs to Logstash. Logstash serves as the centralized streaming engine for data unification and enrichment. The Beats input plugin exposes a secure, acknowledgement-based endpoint for Beats to send data to Logstash.



Logstash

Logstash is a free and open server-side data processing pipeline that ingests data from a multitude of sources, transforms it, and then sends it to your favorite "stash."

```
input {
   ...
}

filter {
   ...
}

output {
   ...
}
```



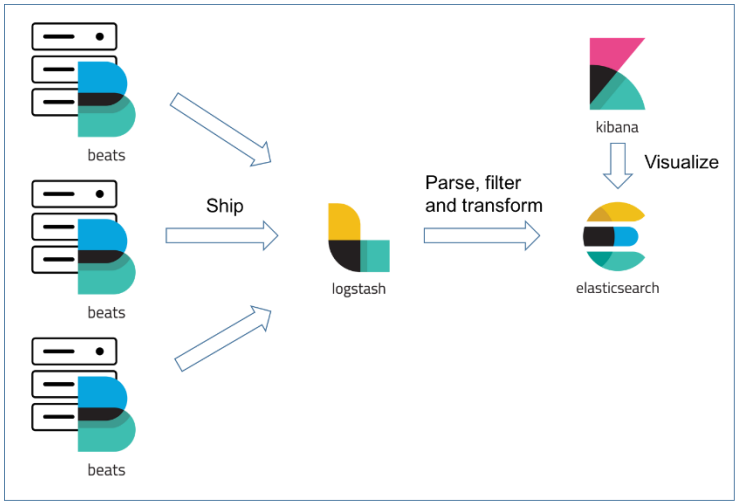
logstash.conf

```
input {
  file {
    path => "/tmp/access_log"
    start_position => "beginning"
  }
}

filter {
  grok {
      match => { "message" => "%{COMBINEDAPACHELOG}" }
    }
  date {
    match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
  }
}

output {
  elasticsearch {
    hosts => ["localhost:9200"]
  }
  stdout { codec => rubydebug }
}
```

Filebeat

Logstash acts as an aggregator — pulling data from various sources before pushing it down the pipeline, usually into Elasticsearch but also into a buffering component in larger production environments. It's worth mentioning that the latest version of Logstash also includes support for persistent queues when storing message queues on disk.

Filebeat, and the other members of the Beats family, acts as a lightweight agent deployed on the edge host, pumping data into Logstash for aggregation, filtering and enrichment.



ELK - Configuration

```
[root@minikube filebeat]# cat /etc/filebeat.yml
filebeat.inputs:
- type: container
  paths:
    - /var/log/containers/*.log
  processors:
    - add_kubernetes_metadata:
        host: ${NODE_NAME}
        matchers:
        - logs_path:
            logs_path: "/var/log/containers/"

processors:
  - add_cloud_metadata:
  - add_host_metadata:

cloud.id: ${ELASTIC_CLOUD_ID}
cloud.auth: ${ELASTIC_CLOUD_AUTH}

setup.kibana:
  host:
"['${KIBANA_HOST:elasticsearch}:${KIBANA_PORT:5601}']"
  protocol: "https"
  ssl.verification_mode: none
```

```
output.elasticsearch:
  hosts: ['${ELASTICSEARCH_HOST:elasticsearch}:${ELASTICSEARCH_PORT:9200}']
  username: ${ELASTICSEARCH_USERNAME}
  password: ${ELASTICSEARCH_PASSWORD}
  protocol: "https"
  ssl.verification_mode: none
  indices:
    - index: "kube-system-%{[agent.version]}-%{+yyyy.MM.dd}"
      when.contains:
        kubernetes.namespace: "kube-system"
    - index: "kube-graph-%{[agent.version]}-%{+yyyy.MM.dd}"
      when.contains:
        kubernetes.namespace: "kube-graph"
    - index: "kube-logging-%{[agent.version]}-%{+yyyy.MM.dd}"
      when.contains:
        kubernetes.namespace: "kube-logging"
    - index: "elastic-system-%{[agent.version]}-%{+yyyy.MM.dd}"
      when.contains:
        kubernetes.namespace: "elastic-system"
```

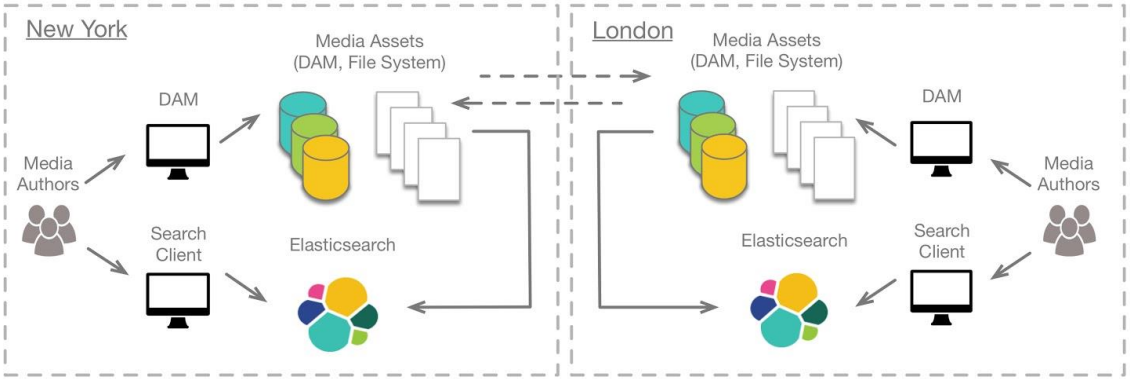
Mappings

Mapping is the process of defining how a document, and the fields it contains, are stored and indexed. For instance, use mappings to define:

- which string fields should be treated as full text fields.
- which fields contain numbers, dates, or geolocations.
- the format of date values.
- custom rules to control the mapping for dynamically added fields.

```
curl –XPUT 127.0.0.1:9200/movies –d '
{
    "mappings": {
        "properties": {
            "year": {"type": "date"}
            }
        }
}'
```

ELK – design notes



Potential Architectures:

- Single Shared Elasticsearch Cluster Distributed Across Data Centers
- Independent Elasticsearch Clusters Searchable by Tribe Nodes
- Independent Elasticsearch Clusters and A Shared Kafka Cluster
- Independent Elasticsearch and Kafka Clusters

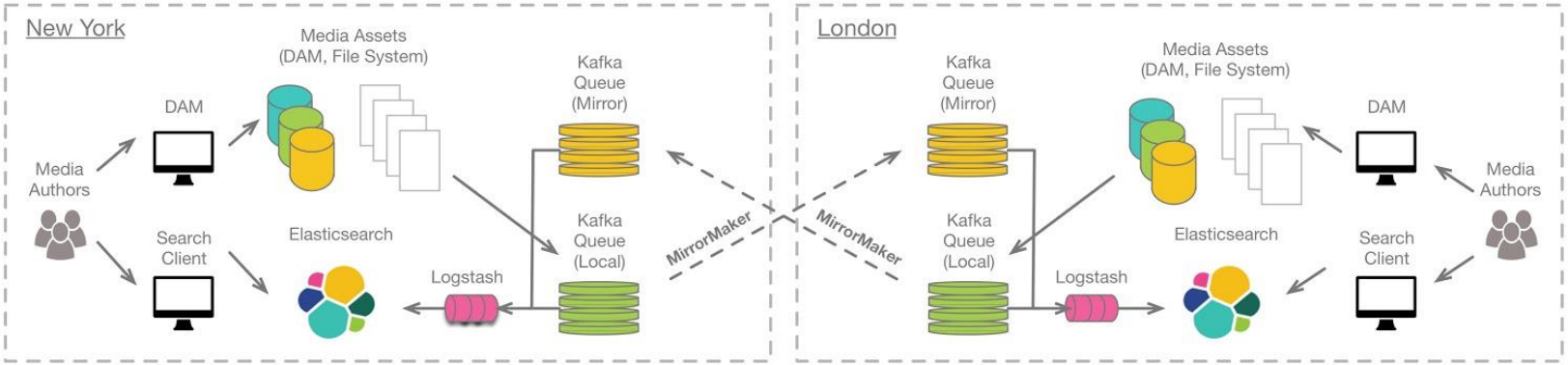
Kafka

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.

- Apache Kafka
- Opensource stream processing platform
- High throughput, low latency
- Publish/subscribe
- Process streams
- Store streams

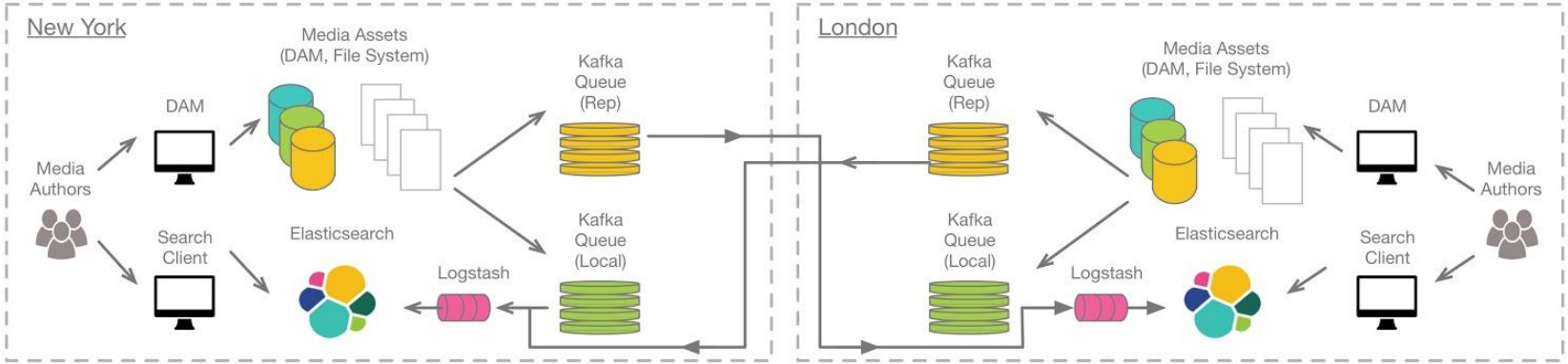
ELK – design notes

Independent Elasticsearch and Kafka Clusters



ELK – design notes

Independent Elasticsearch and Kafka Clusters



THANK YOU