# Part 1 - Deployment

# Learning about data pipelines

This session will be all about the components of data pipelines. Specifically, we will be using the following components for our pipeline today:

- Extractor: Filebeat, Metricbeat, and Packetbeat
- Buffer: Apache Kafka
- Transformer/Loader: Apache NiFi
- Database: OpenSearch

However, to demonstrate the pipeline working, we need an environment to gather data from. So, we will be re-deploying the 5G core network and MonArch monitoring system from yesterday.

# Re-deploying the 5G Core

To begin, we will use `git clone` to fetch the source code of our data pipeline and re-create the entire Kubernetes setup we had deployed yesterday.

> Tip: Triple-click to select a whole line in a code block. To paste in a terminal, use the shortcut Ctrl-Shift-V instead of Ctrl-V.

```
cd ~
git clone https://github.com/hautonjt/data-pipeline
cd data-pipeline
./day1.sh
```

We will be extracting logs from every Kubernetes container using Filebeat, memory and CPU usage of the containers and hosts using Metricbeat, and network connection information using Packetbeat. We will also demonstrate how to extend Metricbeat to extract metrics from the Prometheus exporters in Monarch as well.

# How do we create a data pipeline?

Generally, in production we want to deploy a data pipeline in the following order:

1. Database
2. Buffer
3. Transformer/Loader
4. Extractors

The database and buffer are deployed first as they only receive events and do not send them. Once both the database and buffer have been set up, connecting them with a transformer/loader is trivial. Extractors are started last to prevent a build-up of events at the buffer while waiting for the transformer/loader to initialize.

# How do we create a data pipeline? (2)

However, for testing, it is easier to deploy the pipeline like this:

1. Database
2. Buffer
3. Extractors
4. Transformers/Loaders

By deploying the pipeline in this order, the extractors will send a bunch of events into the buffer first, which will allow you to inspect the structure of each event. This makes it much easier to check the events' schema as well as verify whether the information needed by the transformer is available. We will deploy our pipeline using this order.

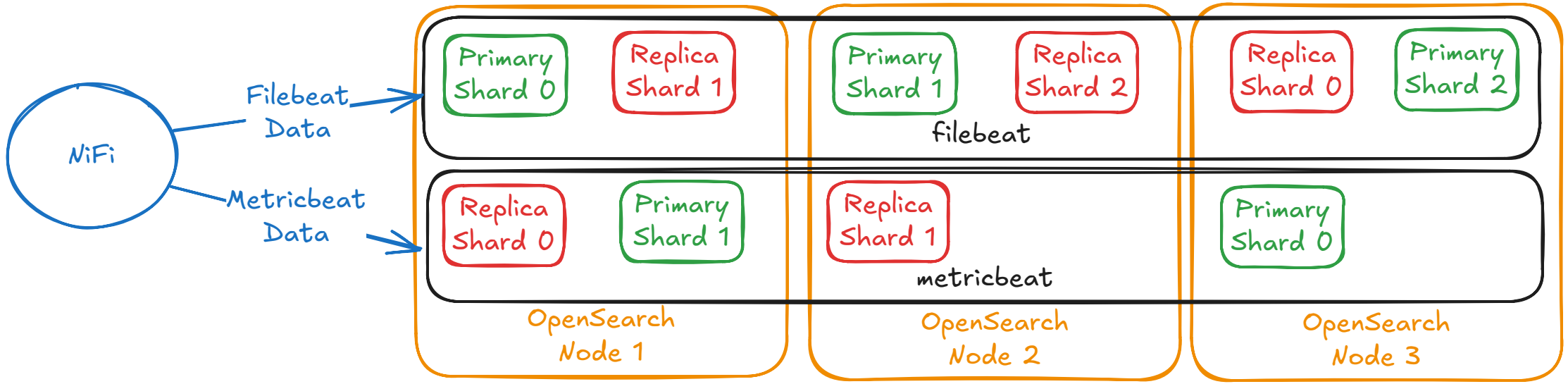
# What is OpenSearch?

As previously mentioned, we'll be using OpenSearch for the database of our pipeline. OpenSearch is a **fork** of ElasticSearch developed by Amazon. It was forked because the company developing ElasticSearch, Elastic NV, changed its license to restrict commercial usage.

Due to its common ancestry, many open-source tools that work with ElasticSearch also work with OpenSearch, including NiFi.

# How does OpenSearch Work?

Data stored in OpenSearch using indexes. Each index consists of a number of primary shards, which allow data to be spread across multiple machines. These shards can be configured with replicas to provided high availability.



# Deploying the database

First, let's deploy the OpenSearch cluster. Run the following:

```
cd ~/data-pipeline
./deploy-opensearch.sh
```

> Note: you will need to type your password (`user`) for this script as it uses `sudo` internally

This script does the following:

- Create a highly available OpenSearch cluster of 3 pods
- Provision an instance of OpenSearch Dashboards
- Configure certificate-based authentication

# Switching namespaces

All of the containers we will be deploying in this session will be assigned to a new namespace named `datapipeline`.

Since we will be interacting with the `datapipeline` namespace the most in this lab, we can switch the namespace that `kubectl` is connected to by running:

```
kubectl config set-context --current --namespace=datapipeline
```

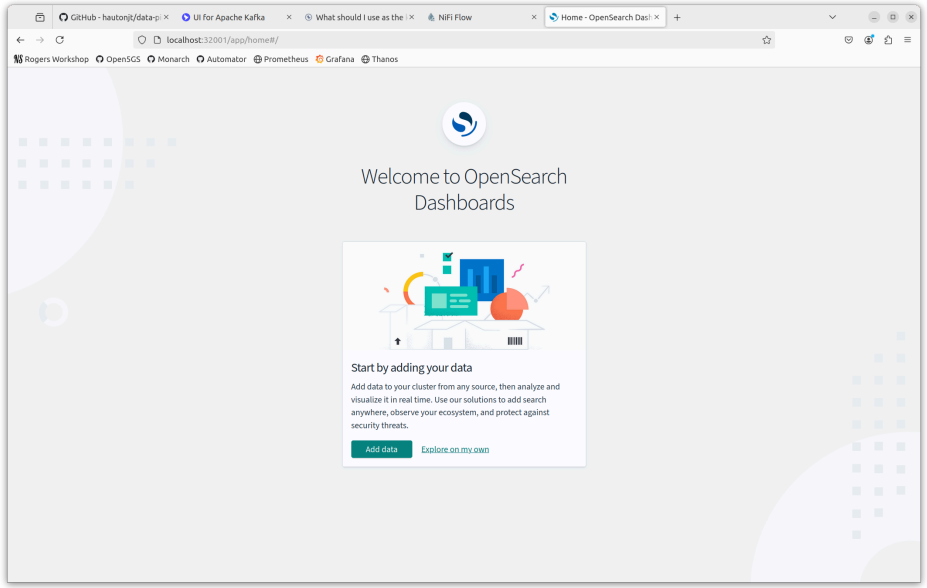
> Tip: if OpenSearch is still deploying, you can also run this in a new terminal window

Now, if you run a kubectl command, such as, `kubectl get pods`, it will run that command in the context of the `datapipeline` namespace by default.

# Access OpenSearch Dashboards

> Tip: Click links with your scroll wheel (middle click) to open links in a new tab. Alternatively, right click to choose if a link should open in a new tab or window.

To access the dashboards, navigate to http://localhost:32001. Both the username and password to log in are set to the value `admin`. If you reach the screen below, then OpenSearch has been deployed successfully and you may close it. We will return to OpenSearch Dashboards later on.



# Deploy Kafka

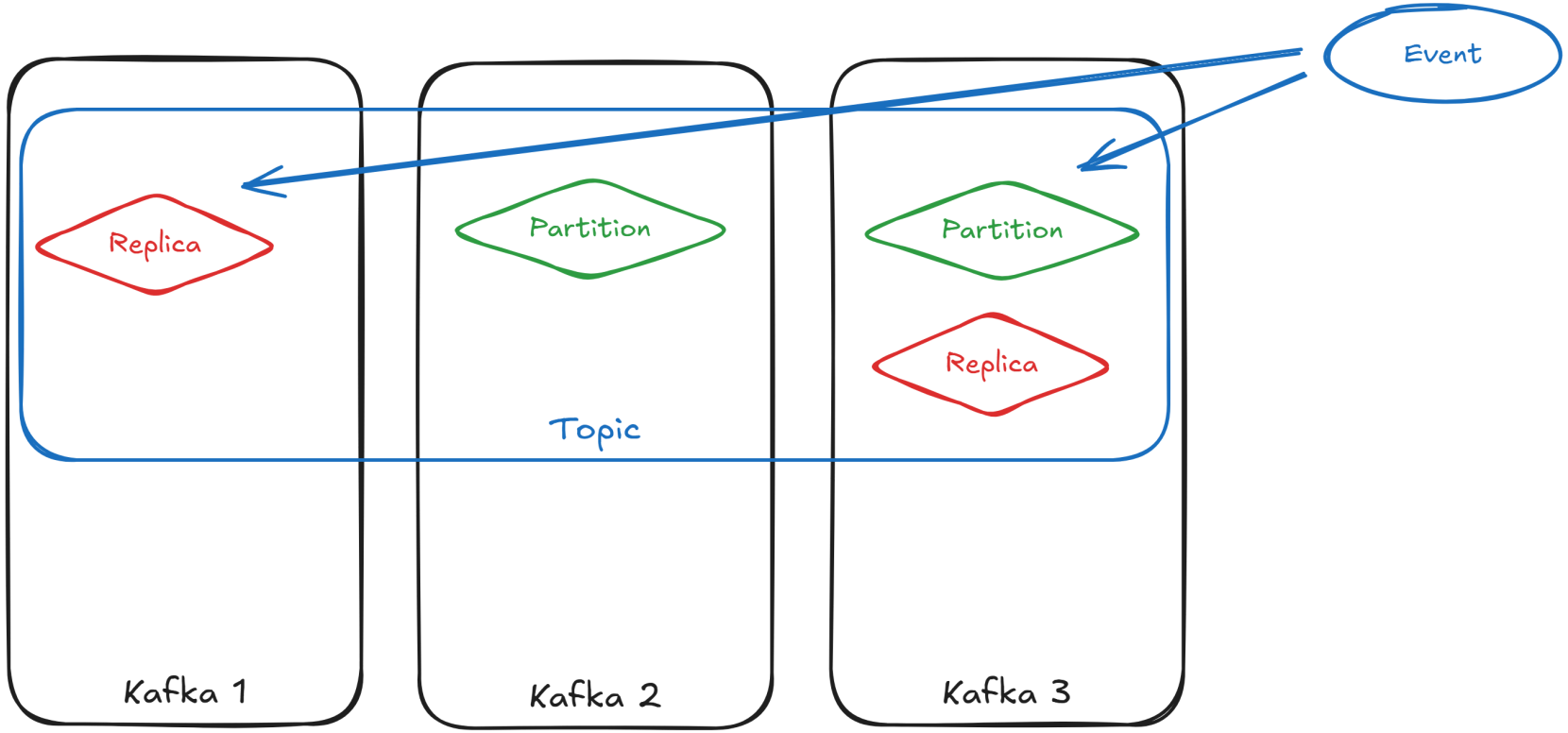
Now that OpenSearch is configured, we can install Apache Kafka. To deploy Kafka, ensure you are in the `data-pipeline` directory, and run:

```
./deploy-kafka.sh
```

This deploys both a Kafka cluster and Kafka-UI, which can be used to configure Kafka using a GUI.

# What is Kafka?

Kafka is a distributed event store. Events in Kafka are mainly divided into *topics*, which are essentially categories of events. Topics are divided into partitions. These partitions allow events in a topic to be distributed across different instances of Kafka, and replicated partitions can provide fault-tolerance.



# What is Kafka? (2)

Topics can be configured to have any number of partitions. The higher number of partitions, the more distributed your data becomes, at the cost of some overhead. Data redundancy and availability is configured using the replication factor, and the number of in-sync replicas.

### Replication factor

Replication factor controls the number of copies of each partition. Having a replication factor of 1 means only 1 copy of each partition, meaning no redundancy. Higher replication factors mean that data from a partition can still be available as long as one copy is still present.

# What is Kafka? (3)

## Min in-sync replicas

Each event must be present in at least *min in-sync replicas* before being successfully written. This minimizes the probability of data loss, but also means that if fewer than that many replicas of a partition are present, a partition could no longer be written to.

## Data persistence

Unlike message queues like RabbitMQ, data stored in Kafka topics are not deleted when events are processed. Deletion is instead controlled by a data retention setting. This means that if an issue occurs in the transformer causing events to be corrupted, events can be re-ingested from the same Kafka topic before retention expires. Kafka consumers need to keep track of what events it has read from a topic.

# Configuring Kafka

**1. Accessing Kafka-UI**

Once the deployment is done, the UI is accessible at http://localhost:32000.

**2. Create filebeat topic**

On the left sidebar, click *Topics*, then on the top right, select "Add a Topic". Fill out the following settings:

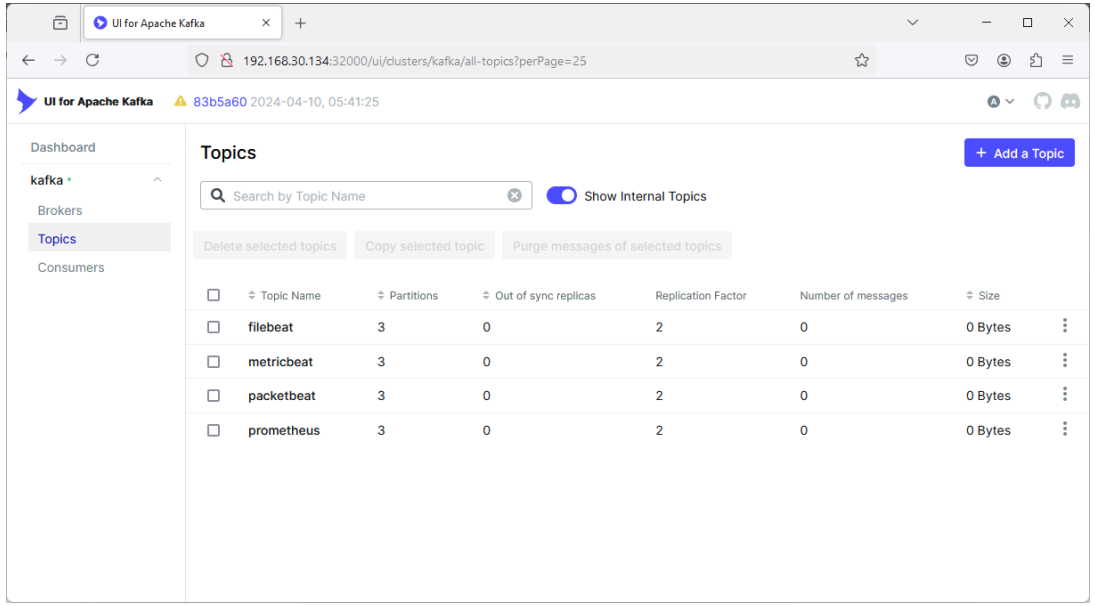
- Topic Name: `filebeat`
- Number of partitions: `3`
- Min in-sync replicas: `2`
- Replication factor: `2`
- Time to retain data: `1 day` (select the "1 day" box below the input)

Then select "Create".

# Configuring Kafka (2)

## 3. Mini-Exercise

Repeat the prior steps with the topics "metricbeat", "packetbeat", and "prometheus". All other settings should remain the same. Your final screen topics screen should look something like this:




> Hint: Click on the *Topics* button in the sidebar again after creating a topic to access the "Add a Topic" button quickly.

# Configuring Beats Agents

Now, we will be deploying our Filebeat, Metricbeat, and Packetbeat agents. Out of the box, Filebeat, Metricbeat, and Packetbeat all support Kubernetes natively and can gather data from Kubernetes automatically, but we also want to gather metrics from our custom Prometheus exporters in Monarch.

To do this, we will need to explicitly configure Metricbeat to find the Prometheus endpoints.

# Configuring Beats Agents (2)

Open the `data-pipeline` folder in VS Code. You can do this by typing:

```
code .
```

In the `beats` folder, there is a file called `metricbeat-prometheus.yaml`.

Find the section that looks like this:

```
    metricbeat.modules:
      - module: prometheus
        period: 5s
        hosts: ["nssdc-kube-state-metrics.monarch.svc:8080"]
        metricsets: ["collector"]
        metrics_path: /metrics

      - module: prometheus
        period: 5s
        hosts: ["kpi-calculator-service.monarch.svc:9000"]
        metricsets: ["collector"]
        metrics_path: /metrics
```

# Configuring Beats Agents (3)

**Mini Exercise:** This is the section for configuring endpoints for Prometheus collectors. We want to add the metrics endpoint from Monarch to gather slice monitoring metrics.

Open the Prometheus GUI deployed by Monarch, and view the list of endpoint URLs.

Add the AMF, SMF, and UPF endpoints on that page to the Metricbeat configmap by replacing the `<insert AMF/SMF/UPF collector URL>` placeholder text. Ensure that the **port** is included in the URL, but the `/metrics` subpath is removed.

Note that two endpoints have already been configured for Metricbeat as examples so you can see what the format should look like.

> Note: answers are in the ~/data-pipeline/lab/metricbeat-prometheus.yaml file if you are stuck.

# Deploy Beats

Now that all the Prometheus endpoints are configured, we can deploy Filebeat, Metricbeat, and Packetbeat all at once. To do this, run:

```
./deploy-beats.sh
```

Filebeat, Metricbeat, and Packetbeat all produce events from different types of information.

**Filebeat** reads informations mainly from log files and JSON HTTP endpoints. Filebeat has native integration with Kubernetes, enabling it to read the logs of all running containers, even ones deployed after Filebeat. For advanced use cases, Filebeat can also use Kubernetes annotations to separately parse the logs of certain containers, if necessary.

# Deploy Beats (2)

**Metricbeat** reads metrics, typically resource usage, from the system. It also has native integrations with various metrics providers. Notably, this includes Prometheus, allowing us to integrate with Monarch. It also integrates with kube-state-metrics, which Monarch installs.

**Packetbeat** captures connection information from all network interfaces present in the current running machine. This includes ingress traffic, egress traffic, and network traffic between pods.

Beats are designed to be deployed together, and have a unified schema. Thus, one advantage of using Beats is not having to worry about normalizing the schema in the transformer.

# Checking Output

If you check the Kafka UI at http://localhost:32000, you should be able to see messages in all the topics if you navigate to Topics on the left menu. Within each individual topic, you can view the messages within the topic by selecting the "Messages" tab. If you do not see messages, please ask for help.

# NiFi

Finally, let's deploy NiFi. To do this, run:

```
./deploy-nifi.sh
```

> Note: you will need to type your password (`user`) for this script as it uses `sudo` to copy the geoip database to a system directory.

This script does the following:

- deploys a NiFi cluster of 3 nodes
- deploys a Zookeeper cluster of 3 nodes (used by NiFi for maintaining configuration and cluster information)

Warning: this deployment takes a long time to complete.

# What is NiFi?

NiFi is a versatile distributed data processor. NiFi allows you to create data processing graphs that can allow you to visually see how data will be processed.

# How does NiFi work?

NiFi operates using the concept of FlowFiles, which is just a container that can hold any data along with some attributes, which are essentially metadata. This means that FlowFiles inherently do not have any structure at all.

Processors act on FlowFiles and transforms them in some way. There are a vast array of processors that cater to almost any use case, and for use cases that built-in processors are incapable of handling, NiFi also supports calling external scripts. You can integrate ML with NiFi directly using processors instead of interfacing with Kafka.

# How does NiFi work? (2)

Since FlowFiles have no structure, typically services need to be specified so the processor knows how to parse the FlowFile and what format to output it in. For example, the `JsonTreeReader` service reads the FlowFile and parses it into one or more JSON object for processing. The `JsonRecordSetWriter` writes an array of JSON objects as its output.

Services are not limited to just parsing FlowFiles, they also provide services such as SSL authentication, OpenSearch integration, caching, lookups, and more.

# Next Steps

**Congratulations!**

You have successfully completed the following:

- Deployed all the components of a data pipeline
- Configured Apache Kafka and Beats
- Customized Metricbeat configuration to pull metrics from Prometheus

**What's Next?**

Configure NiFi and visualizations in Part 2.