ADMON
Anomaly Detection for MONIT data

ML Workshop

Nikolay Tsvetkov

10.03.2023

Anomaly Detection

- **ML based technique for detecting data pattern anomalies**
- **Very useful for monitoring data**
  - Applicable on time-series metrics and/or logs
  - Allow correlation of different datasets
  - Help to identify misbehaviours
  - Decrease the reaction time

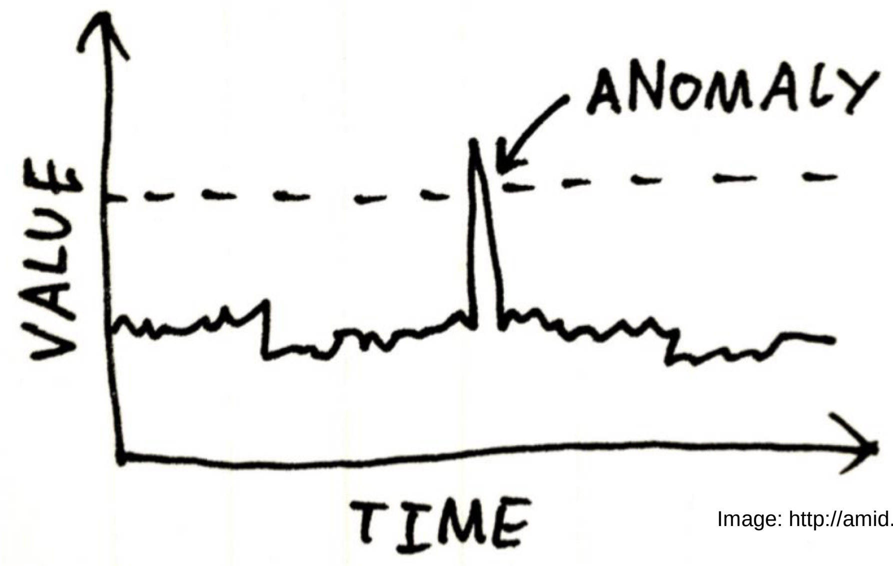


Image: http://amid.fish

<u>Leads to prevented failures and improved reliability!</u>

ADMON Motivation and Objectives

Improve the monitoring experience

- Allow users and Service Managers to detect and prevent outages as early as possible
- Improve the performance and stability of services by improving their monitoring

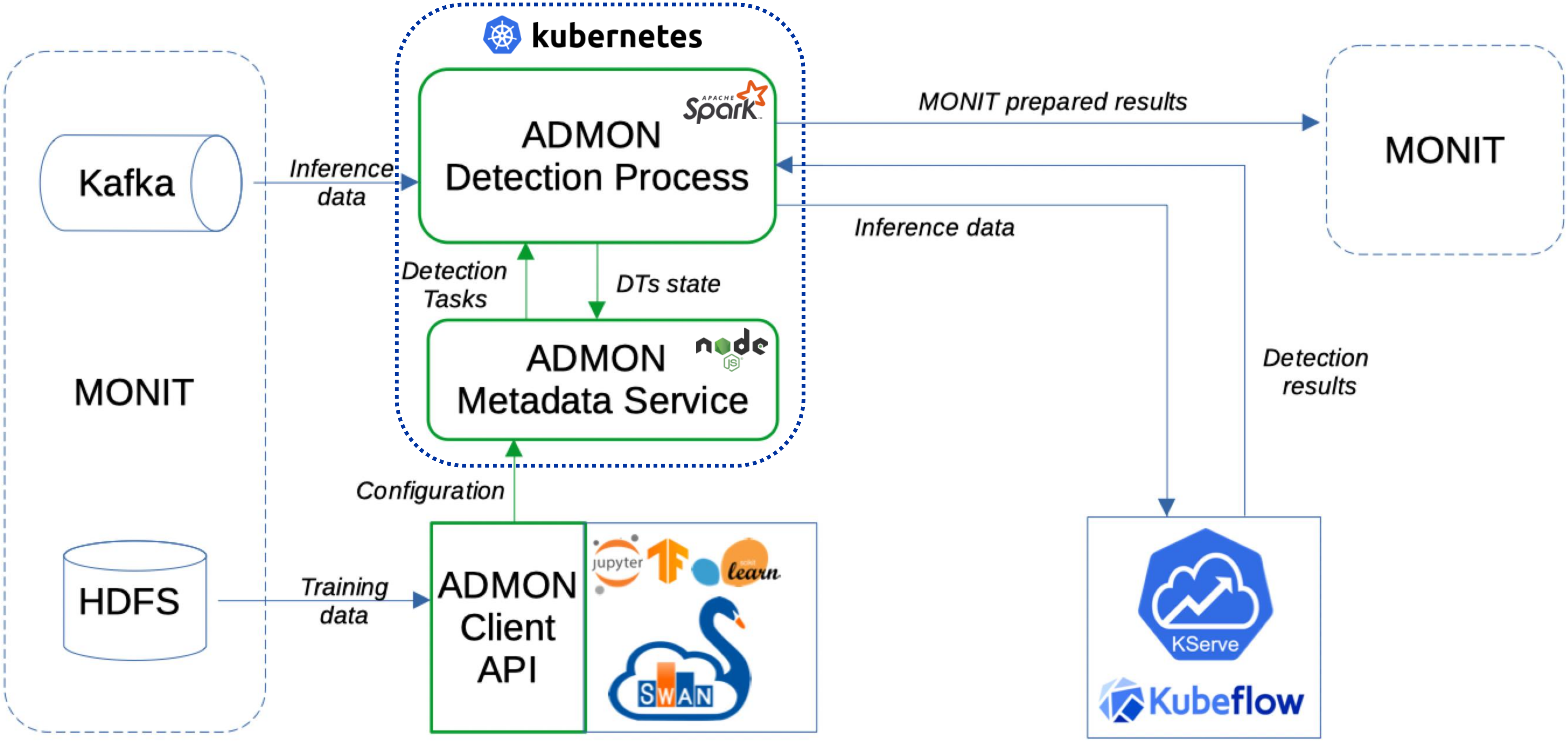
Make AD widely accessible

- Provide common infrastructure for processing AD models for IT Monitoring (MONIT) data
- Simplify the access to fresh IT Monitoring data and generate results on recent events
- Export the AD results to the IT Monitoring infrastructure

Consolidate ongoing work and efforts within IT / WLCG

- Integrate already available tools and services provided within CERN IT
- Reduce the overhead of building and maintaining custom ML infrastructure and tools
- Set ground for sharing know-how on already developed and proven algorithms and models

Architecture



Input Data Processing

1. Filter, Select, Rename

E.g.: filter for type 'msg'

time	col a	host	type
0	2	b	msg
1	1	a	msg
3	3	a	msg
3	4	b	msg
5	2	a	msg
6	4	b	msg
7	6	a	msg

time	col b	host	type
0	3	a	msg
1	7	b	msg
2	3	a	msg
4	3	b	msg
5	9	b	msg
7	3	a	msg
7	5	b	msg

2. Join over time window

E.g.: Window of size 5 with average

time	col a	host
0	2	a
0	3	b
5	4	a
5	4	b

time	col b	host
0	3	a
0	5	b
5	3	a
5	7	b

time	col a	col b	host
0	2	3	a
0	3	5	b
5	4	3	a
5	4	7	b

3. JSON payload

E.g.: Payload of host "a"

```
{
   metadata: {
      host: a
   },
   data: [
      {
         time: 0,
         col a: 2,
         col b: 3
      }, {
         time: 5,
         col a: 4,
         col b: 3
      }
   ]
}
```

ADMON API

1.Create *SourceConfig*

```python
from admonapi import SourceConfig

sc_gled = SourceConfig(
    "collectd", "raw", "monitoring",
    select=["timestamp", "value", "host"],
    filter_expression="topic=='xrootd_raw_gled'",
    rename={"value": "gled_value"}
)
sc_alice = SourceConfig(
    "collectd", "raw", "monitoring",
    select=["timestamp", "value", "host"],
    filter_expression="topic=='xrootd_raw_alice'",
    rename={"value": "alice_value"}
)
```

2.Create *InputDataConfig*

```python
from admonapi import InputDataConfig

input_data_config = InputDataConfig(
    source_configs = [sc_gled, sc_alice],
    agg_interval_seconds = 300,
    agg_method = "avg",
    group_by = ["host"]
)
```

3.Load data from HDFS

```python
from admonapi import DataSource

data_source = DataSource(spark, input_data_config)
data_frame = data_source.read_hdfs(start_timestamp, end_timestamp)
```

Result document


```
{
    {
  "metadata": {
    "host": "monit-kafkay-1182c933d6.cern.ch",
    "agg_interval_seconds": "300",
    "agg_method": "avg"
  },
  "data": [
    {"timestamp": 1652169600000, "gled_value": 1404320.02, "alice_value": 16440.78},
    {"timestamp": 1652169900000, "gled_value": 1189512.86, "alice_value":    35.42},
    {"timestamp": 1652170200000, "gled_value":   11370.35, "alice_value":    42.97},
    {"timestamp": 1652170500000, "gled_value":   10336.88, "alice_value":    35.98},
    {"timestamp": 1652170800000, "gled_value":   11548.64, "alice_value":  1297.48},
    {"timestamp": 1652171100000, "gled_value":   11200.05, "alice_value":    33.63},
    {"timestamp": 1652171400000, "gled_value":   11147.68, "alice_value":    48.03},
    {"timestamp": 1652171700000, "gled_value":   13309.60, "alice_value":    33.78},
    {"timestamp": 1652172000000, "gled_value":   12769.24, "alice_value":    48.07},
    {"timestamp": 1652172300000, "gled_value":   13392.86, "alice_value":    33.79},
    {"timestamp": 1652172600000, "gled_value":   13040.87, "alice_value":    47.94},
    {"timestamp": 1652172900000, "gled_value":   14044.85, "alice_value":    33.49},
  ]
}
```


Users can use these data for developing their model

- The same schema will be received for model inference
- Further transformation functions can be applied by the user

ADMON API

4.Create *Project* and *DetectionEntity*

```
project = Project.create(
    title="XRootD Anomaly Detection",
    project_url="https://admon.docs.cern.ch",
    description="Anomaly detection on XRootD data based on correlation.",
    is_private=False,
    egroup="admon-dev"
)

de = DetectionEntity.create(
    project=project,
    title="XRootD Anomaly Detection for 1 hour intervals",
    interval_minutes=60,
    sliding_interval_minutes=15,
    input_data_config=idc,
    inference_model="xrootd-model",
    inference_namespace="admon-dev",
    monit_producer="admon",
    monit_label: {"admon_entity": "xrootd_with_join"}
)
```

ADMON Docs: (https://admon.docs.cern.ch/)

Final step: Create *InferenceService* with *Transformer* in Kubeflow

- Build Docker image containing your transformation functions
- Train and store prediction model in S3

Project summary

- **Simplifies feature engineering and developing AD models**
  - Integrates the MONIT HDFS storage through Python API in SWAN
  - Provides aggregation of multiple data sources into a single dataset
- **Automates the model inferencing using the provided configuration**
  - Removes the effort of developing and maintaining own ML pipelines
  - Based on standard IT tools (SWAN, Kubeflow, IT Monitoring)
- **Applies on fresh MONIT data and sends results back to MONIT**
  - Allows earlier detection of potential problems
- **Scalable infrastructure able to cover more load in case of demand**
  - Spark based process running in Kubernetes cluster
- **Standard API allows sharing configurations between Service Managers**
- **Project has been completed and ready-to-use infrastructure is archived**

Thank you !

Q & A

home.cern