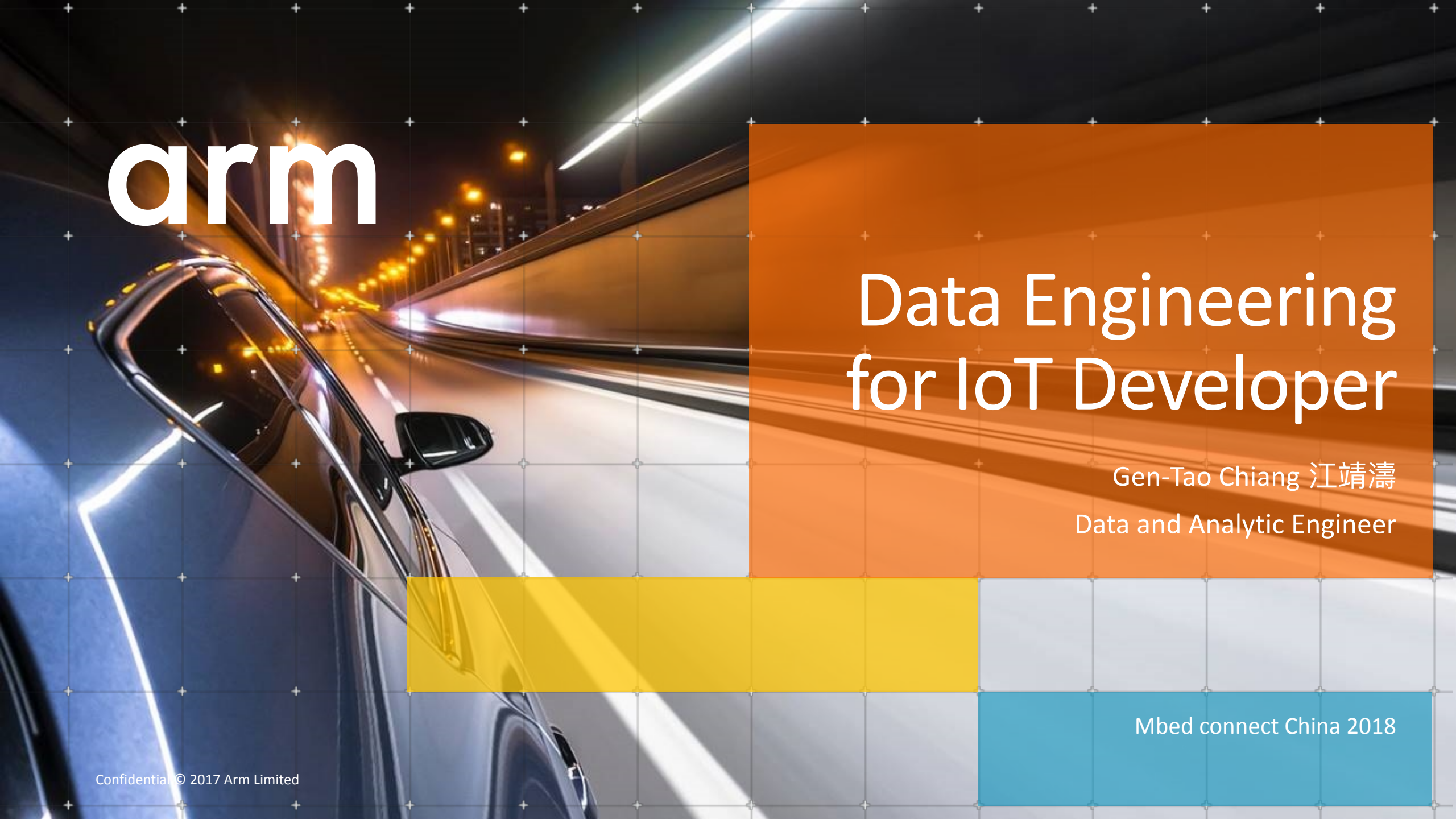


Data Engineering for IoT Developer

Gen-Tao Chiang 江靖濤

Data and Analytic Engineer

Mbed connect China 2018

Outline

- The importance of sending your data somewhere.
- How a setup would look like which could get billions of records in.
- Data Science Africa use case.
- Arm use case.

Where to process data

Edge

- Analytics at the edge means part of the data processing is done relatively close to the sensor.
- Scenario: If a machine must respond immediately to its own metrics, environmental sensor data or some combination of those.

Where to process data

Cloud

- Larger variety of data
- adding historical data to streaming data.
- Analyze all the outputs from all the devices.
- For those tasks require high volume of data or it is process intensive tasks.

Where to process data

Hybrid

- Often only part of the processing is done on the edge. We really need do both cloud and the edge. The right answer for many companies is to process some of the information on the edge and transfer the results to the cloud.
- Edge, GDPR (avoid transfer personal information).
- storage: save storage cost
- Network: save bandwidth
- DSA, Kenya, there is not enough bandwidth and devices need to be deployed at rural locations where has no reliable internet connections to send batches of data or streaming data.
- No matter what scenario, we need to transfer data to cloud and need a data pipeline.

arm

Data Engineering

Common Tools

- SQL : MySQL, PostgreSQL, Redshift, Hive
- NoSQL: MongoDB, ElasticSearch, Cassandra, InfluxDB, Treasure Data
- Big data solutions: Hadoop, Spark
- Programming: Python, Scala
- Notebook: Jupyter, zeppelin
- DevOps, or Cluster Management: Docker, Jenkins, Kubernetes, Yarn, Mesos, Terraform.
- Cloud Computing: AWS, Azure
- Workflow management, Logstash, Luigi, GLUE
- Message queue: Kafka, RabbitMQ, Kinesis.
- Business Intelligent and Visualization: Splunk, Tableau, Kibana, D3, Superset..etc.

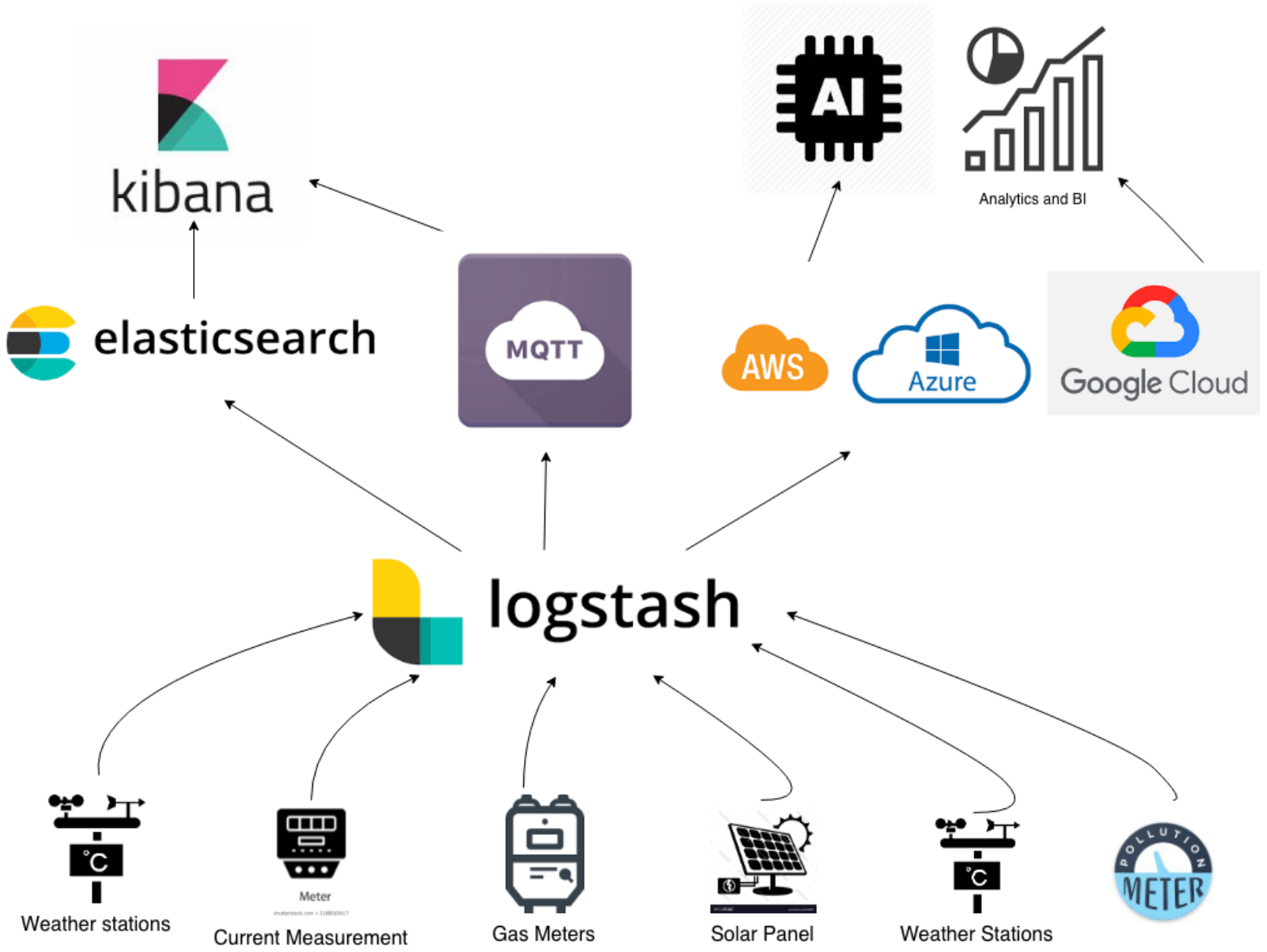
Architecture

ELK

- Elasticsearch is a search engine and provides a distributed storage and an HTTP web interface and schema-free JSON documents.
- Logstash is an open source, server-side data processing pipeline that ingests data from a multitude of sources simultaneously, transforms it, and then send it to storage.
- Kibana is a open source data visualization plugin for Elasticsearch. It provides visualization capabilities on top of the content indexed on an Elasticsearch cluster.

Architecture

ELK



arm

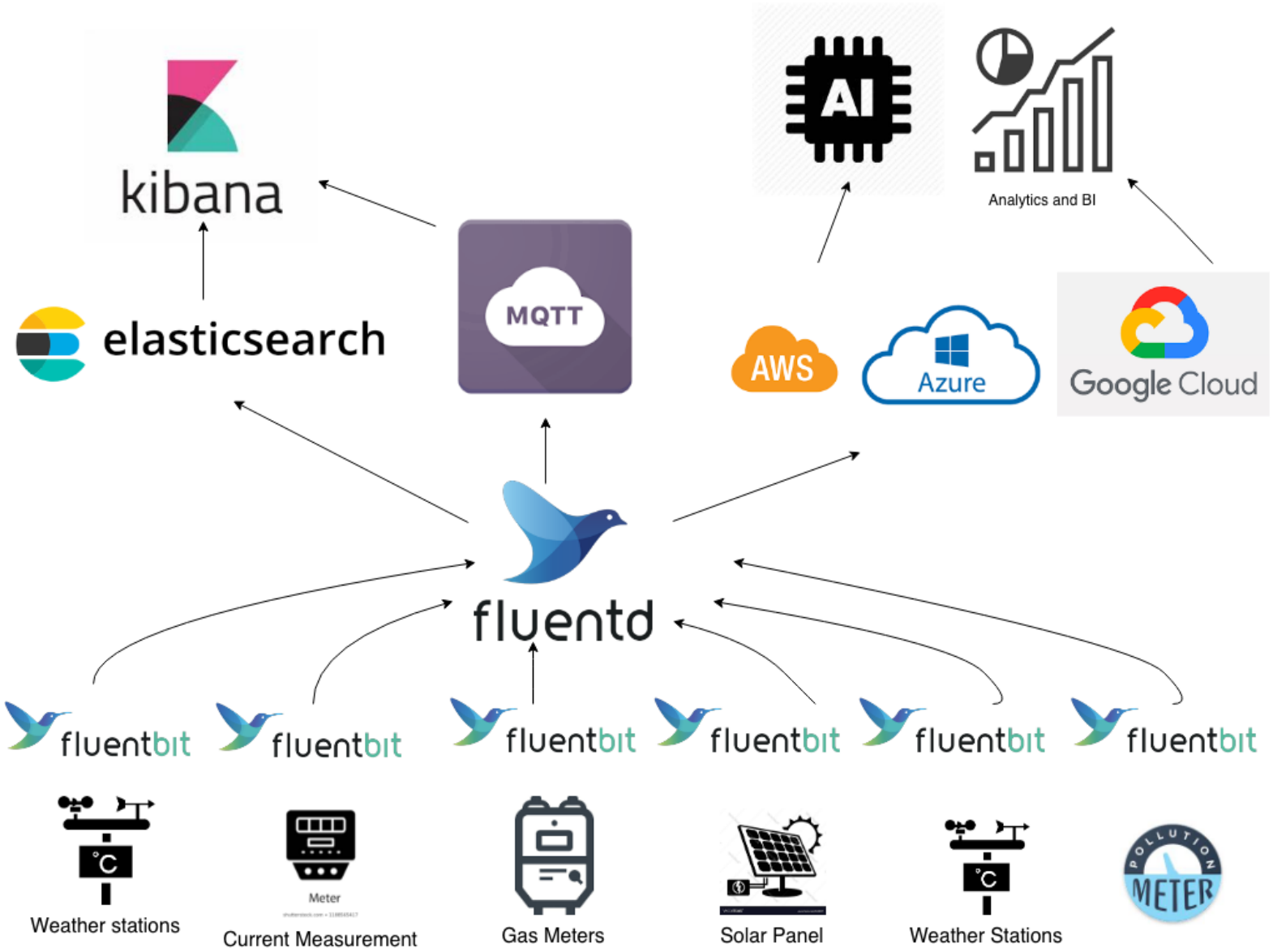
Architecture

Arm Treasure Data > EFK

- Fluent-bit: Fluent Bit is a Data Forwarder for Linux, Embedded Linux, OSX and BSD family operating systems. It's part of the Fluentd Ecosystem. Fluent Bit allows collection of information from different sources, buffering and dispatching them to different outputs such as Fluentd, Elasticsearch, or any HTTP end-point within others. It's fully supported on x86_64, x86 and ARM architectures.
- Fleuntd: Fluentd collects events from various data sources and writes them to files, RDBMS, NoSQL, IaaS, SaaS, Hadoop and so on.

Architecture

EFK



arm

Fluentd

https://www.fluentd.org/plugins/all

Input / Output plugins:

Collect events from sources or send events to destinations

Certified	Download	Name	Author	About	Version
✓	2523659	elasticsearch	diogo, pitr	Elasticsearch output plugin for Fluent event collector	2.11.10
✓	1593331	secure-forward	TAGOMORI Satoshi	Message forwarding over SSL with authentication	0.4.5
✓	1281137	forest	TAGOMORI Satoshi	create sub-plugin dynamically per tags, with template configuration and parameters	0.3.3
✓	1199132	record-reformer	Naotoshi Seo	Fluentd plugin to add or replace fields of a event record	0.9.1
✓	1163002	s3	Sadayuki Furuhashi, Masahiro Nakagawa	Amazon S3 output plugin for Fluentd event collector	1.1.6
✓	1052169	kinesis	Amazon Web Services	Fluentd output plugin that sends events to Amazon Kinesis.	2.1.1
	585858	google-cloud	Stackdriver Agents Team	Fluentd plugins for the Stackdriver Logging API, which will make logs viewable in the Stackdriver Logs Viewer and can optionally store them in Google Cloud Storage and/or BigQuery. This is an official Google Ruby gem.	0.6.25.1
	575857	systemd	Ed Robinson	This is a fluentd input plugin. It reads logs from the systemd journal.	1.0.1
✓	573403	kafka	Hidemasa Togashi, Masahiro Nakagawa	Fluentd plugin for Apache Kafka > 0.8	0.7.9

arm

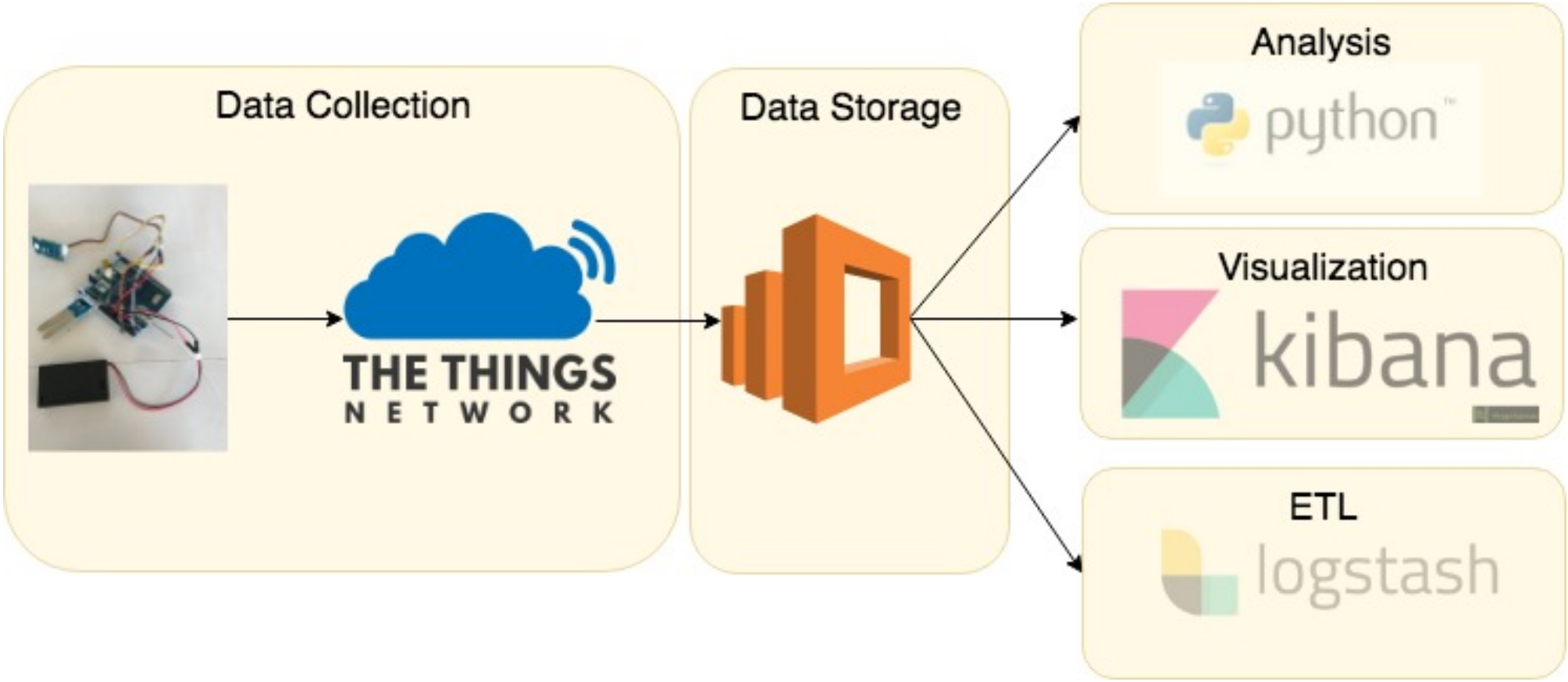
Data Science Africa (DSA) Use Case

- DSA aim to create a hub in the network of data science researchers across Africa.
- DSA 2018 is sponsored by Arm, Amazon, Google DeepMind, Voyage, and Qualcomm.
- Its an opportunity to **show a IoT system solving a real-world problem.** Air quality monitoring and greenhouse monitoring are aligned with "Connected spaces" with a **lasting deployment generating reference data for data scientists.**

arm

DSA Use Case

Greenhouse and Air quality



arm

DSA Use Case

Data Storage: Elasticsearch : Create Index and Mapping

```
curl -X PUT "localhost:9200/greenhouse" -H 'Content-Type: application/json' -d'
{
   "settings" : {
      "number_of_shards" : 1
   },
   "mappings": {
    "type1": {
     "properties": {
      "analog_in_4" : { "type": "float"  },
      "relative_humidity_3":   { "type": "float"  },
      "temperature_2":    { "type": "float"  },
      "app_id":     { "type": "text" },
      "dev_id":   { "type": "text"  },
      "time":  { "type":   "date", "format": "strict_date_optional_time" },
……..……..
```


arm

DSA Use Case

Data Storage: Elasticsearch : List Index

curl -X GET "https://search-dsa-fx7s43inji63qg3oxxuziwjxa4.us-west-2.es.amazonaws.com:443/_cat/indices?v"

health status index uuid pri rep docs.count docs.deleted store.size pri.store.size

yellow open airquality 46OHSD-ASzG3cLWGW2A4Rw 1 1 0 0 230b 230b

yellow open greenhouse2 SLJ7oNS2SqKT-6qkgkKRmA 1 1 2552 0 704.7kb 704.7kb

green open .kibana pn-pPxQVQRO5gxe2C0iyug 1 0 7 1 44.5kb 44.5kb

arm

DSA Use Case

Data Storage: Indexing Data from TTN to Elasticsearch

```
import ttn
from elasticsearch import Elasticsearch

app_id = "dsa2018-greenhouse"
access_key = ”ttn-key"
index = "greenhouse"
doctype = "type1"
eshost = "localhost:9200”

es = Elasticsearch(
    [eshost],
    scheme="http",
    port=9200,
)
```


arm

DSA Use Case

Data Storage: Indexing Data from TTN to Elasticsearch

```
def uplink_callback(msg, client):
    print("Received uplink from ", msg.dev_id)
    print("type of msg",type(msg))
    doc = {
        "_op_type": "index",
        "_index": index,
        "_type": doctype,
        "app_id": msg.app_id,
        "dev_id": msg.dev_id,
        "hardware_serial": msg.hardware_serial,
        "counter":msg.counter,

        "analog_in_4" : analog_in_4,
        "relative_humidity_3": relative_humidity_3,
        "temperature_2" : temperature_2
     }
```


arm

DSA Use Case

Data Storage: Indexing Data from TTN to Elasticsearch

```
def write2es(documents):
    helpers.bulk(es, documents, index=index, doc_type=doctype)
    print("reset docs to 0")
    global docs
    docs = []


if len(docs) == 2:
   print("call write2es")
   write2es(docs)
```


arm

DSA Use Case

Exploring Data: Kibana

Time	app_id	dev_id	gtw_id	gtw_time	relative_humidity_3	time	hardware_serial
May 24th 2018, 11:46:39.163	dsa2018	simulated-device-1	eui-0242ca0000fba14f	May 24th 2018, 11:46:39.148	1.5	May 24th 2018, 11:46:39.163	00EF87326A4CC685
May 24th 2018, 11:46:33.383	dsa2018	simulated-device-1	eui-0242ca0000fba14f	May 24th 2018, 11:46:33.376	2	May 24th 2018, 11:46:33.383	00EF87326A4CC685
May 24th 2018, 11:46:27.126	dsa2018	simulated-device-1	eui-0242ca0000fba14f	May 24th 2018, 11:46:27.117	1	May 24th 2018, 11:46:27.126	00EF87326A4CC685
May 24th 2018, 11:46:20.806	dsa2018	simulated-device-1	eui-0242ca0000fba14f	May 24th 2018, 11:46:20.754	2	May 24th 2018, 11:46:20.806	00EF87326A4CC685
May 24th 2018, 11:46:07.661	dsa2018	simulated-device-1	eui-0242ca0000fba14f	May 24th 2018, 11:46:07.438	1.5	May 24th 2018, 11:46:07.661	00EF87326A4CC685
May 24th 2018, 11:44:30.892	dsa2018	simulated-device-1	eui-0242ca0000fba14f	May 24th 2018, 11:44:30.884	1.5	May 24th 2018, 11:44:30.892	00EF87326A4CC685
May 22nd 2018, 16:28:57.551	dsa2018	simulated-device-1	eui-0242ca0000fba14f	May 22nd 2018, 16:28:57.541	0	May 22nd 2018, 16:28:57.551	00EF87326A4CC685
May 22nd 2018, 16:28:46.696	dsa2018	simulated-device-1	eui-0242ca0000	May 22nd 2018, 16:28:46.687	0	May 22nd 2018,	00EF87326A4CC685

arm

Exploring Data

Pvthon notebook

In [34]:
```
from elasticsearch import Elasticsearch
from elasticsearch import helpers
from pandas import DataFrame, Series
import pandas as pd
import matplotlib.pyplot as plt
from pandas.io.json import json_normalize
```

In [35]:
```
eshost = "https://search-dsa-fx7s43inji63qg3oxxuziwjxa4.us-west-2.es.amazonaws.com"
```

In [36]:
```
es = Elasticsearch(
    [eshost],
    scheme="https",
    port=443,
)
```

In [81]:
```
# call ES to return all docs, using size to define the return size of documents
res = es.search(index="greenhouse2", doc_type="type1", body={"query": {"match_all": {}}}, size = 2000)
# How many documents
print("%d documents found" % res['hits']['total'])
```

```
2092 documents found
```

In [83]:
```
# read data to df
#data = res['hits']['hits']
#df_all = pd.concat(map(pd.DataFrame.from_dict, data), axis=1)['_source'].T

df = json_normalize(res['hits']['hits'])
```

In [93]:
```
# we only interested in time, temperature, and humidity
df_greenhouse = df[['_source.time','_source.dev_id','_source.temperature_2','_source.relative_humidity_3','_source.anal
df_greenhouse.columns = ['time','dev_id','temperature','humidity','moisture']
```

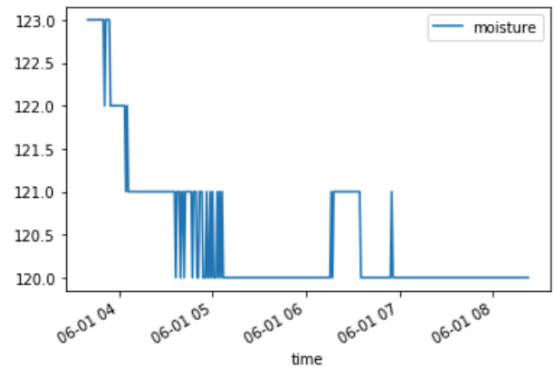
In [94]:
```
# sorting based on time
df_greenhouse.sort_values(by='time')
```

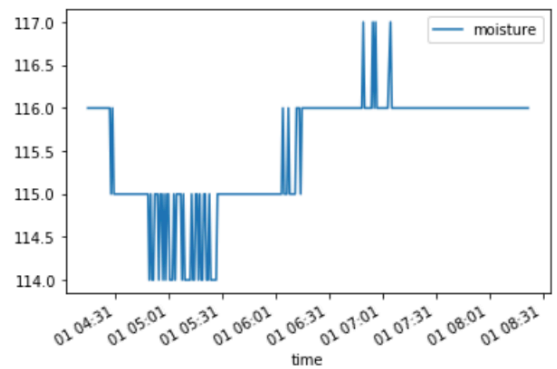
Out[94]:

	time	dev_id	temperature	humidity	moisture
14	2018-05-31 15:00:29.581456053	dsa-greenhouse2	20.0	88.0	43
15	2018-05-31 15:00:36.216661414	dsa-greenhouse4	20.0	91.0	133
8	2018-05-31 15:00:46.499420401	dsa-greenhouse5	20.0	98.0	13
9	2018-05-31 15:00:50.667582301	dsa-greenhouse10	18.0	90.0	134
10	2018-05-31 15:00:54.760346157	dsa-greenhouse3	20.0	83.0	119
11	2018-05-31 15:01:09.847542100	dsa-greenhouse2	20.0	89.0	43
12	2018-05-31 15:01:22.339151924	dsa-greenhouse4	NaN	NaN	134
13	2018-05-31 15:01:26.665107355	dsa-greenhouse5	20.0	99.0	13
2	2018-05-31 15:01:35.046254320	dsa-greenhouse3	20.0	83.0	119
3	2018-05-31 15:01:36.795762345	dsa-greenhouse10	NaN	NaN	134
18	2018-05-31 15:01:43.703890716	dsa-greenhouse8	19.0	91.0	136
19	2018-05-31 15:01:48.117988568	dsa-greenhouse7	20.0	76.0	118
16	2018-05-31 15:01:50.080199449	dsa-greenhouse2	20.0	90.0	42
17	2018-05-31 15:02:02.744816394	dsa-greenhouse4	20.0	87.0	134
4	2018-05-31 15:02:09.942532474	dsa-greenhouse5	20.0	93.0	14
5	2018-05-31 15:02:21.119441039	dsa-greenhouse3	NaN	NaN	119
6	2018-05-31 15:02:23.080372312	dsa-greenhouse10	NaN	NaN	134
7	2018-05-31 15:02:23.955006274	dsa-greenhouse8	19.0	96.0	136
0	2018-05-31 15:02:33.362836098	dsa-greenhouse2	20.0	84.0	42
1	2018-05-31 15:02:48.867791907	dsa-greenhouse4	NaN	NaN	134
28	2018-05-31 15:02:53.222401607	dsa-greenhouse5	20.0	94.0	14
29	2018-05-31 15:03:01.539049391	dsa-greenhouse3	20.0	85.0	119
24	2018-05-31 15:03:14.234715588	dsa-greenhouse1	19.0	90.0	93
25	2018-05-31 15:03:16.640250494	dsa-greenhouse2	20.0	82.0	42
30	2018-05-31 15:03:29.275184207	dsa-greenhouse4	20.0	87.0	134
31	2018-05-31 15:03:33.468794977	dsa-greenhouse5	20.0	99.0	14
34	2018-05-31 15:03:41.788677843	dsa-greenhouse3	20.0	86.0	119
35	2018-05-31 15:03:44.456512310	dsa-greenhouse8	19.0	96.0	137
26	2018-05-31 15:03:49.769686520	dsa-greenhouse10	18.0	90.0	134
27	2018-05-31 15:03:54.485569501	dsa-greenhouse1	19.0	93.0	93
...	...	...	...	...	...

In [108]:
```
df_greenhouse_0601.groupby('dev_id').plot(x='time', y='moisture')
```

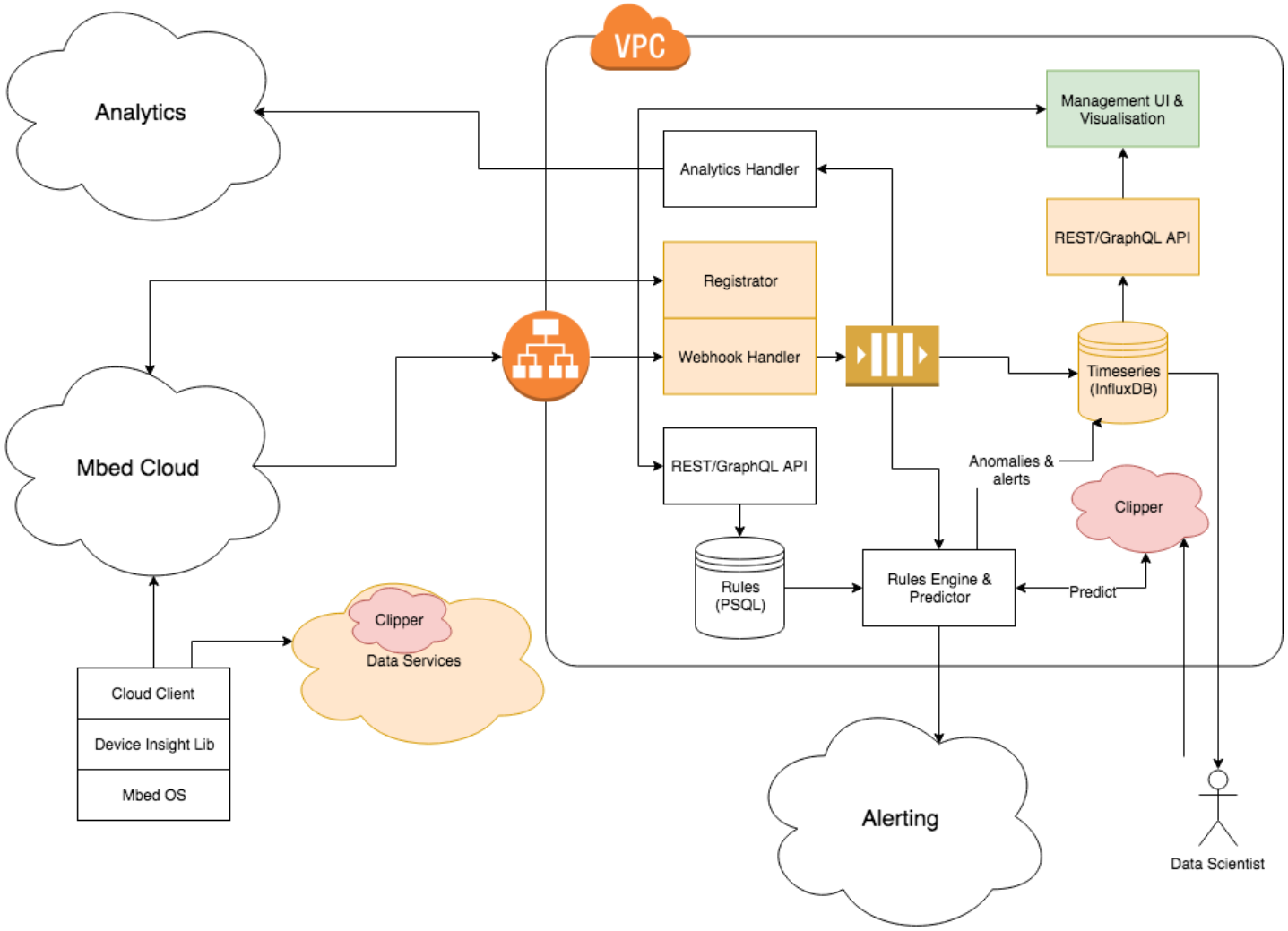
Out[108]:
```
dev_id
dsa-greenhouse10    AxesSubplot(0.125,0.2;0.775x0.68)
dsa-greenhouse4     AxesSubplot(0.125,0.2;0.775x0.68)
dtype: object
```





Arm Use Case:

Device Insight



arm

Device Insight and Cloud Infrastructure

Cloud Sandbox to EKS

SandBox : Using mbed SaaS Platform Tool (splatt) developed by the ISG platform team to deploy a Kubernetes cluster and components including all the Mbed Cloud components and Webhook Handler, influxDB…etc.

EKS (Amazon Elastic Container Service for Kubernetes)

- Webhook Handler
- Timeseries DB
- Visualization Tool (Grafana)
- Clipper Cluster
- ML models

Clipper: Deploying ML models to production

University of California at Berkeley RISE Lab

- Deploying trained machine-learning models into production today is an ad-hoc and labor-intensive process. This creates an enormous impediment to building and maintaining user-facing applications that incorporate machine-learning.
- Clipper is designed to simplify this process by decoupling applications that consume predictions from trained models that produce predictions.
- Clipper is a prediction serving system that sits between user-facing applications and a wide range of commonly used machine learning models and frameworks.

Clipper Architecture

Applications

NETFLIX

Predict ↕ **RPC/REST Interface** ↕ Observe

Clipper

theano

Dato Create

KeystoneML

Caffe

TensorFlow

scikit learn

dmlc mxnet

KALDI

Clipper Demo

Damon Civin (DSA 2018)

- Pretrained model
- Deploy model using Clipper
- Define the scale of model
- Query the model

Pretrained model

Sciket-learn Iris example

```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import itertools
import numpy as np

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

# import some data to play with
iris = datasets.load_iris()
X = iris.data
y = iris.target
class_names = iris.target_names

# Split the data into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# Run classifier, using a model that is too regularized (C too low) to see
# the impact on the results
classifier = svm.SVC(kernel='linear', C=0.1)
y_pred = classifier.fit(X_train, y_train).predict(X_test)
```

Deploy the model using Clipper

Start clipper cluster

```
from clipper_admin import ClipperConnection, DockerContainerManager
```

```
clipper_conn.stop_all()
```

```
18-07-09:18:59:31 INFO     [clipper_admin.py:1258] Stopped all Clipper cluster and all model containers
```

```
clipper_conn = ClipperConnection(DockerContainerManager())
```

```
# Start Clipper. Running this command for the first time will
# download several Docker containers, so it may take some time.
try:
    clipper_conn.start_clipper()
except:
    print("Clipper already running")
    cliiper_conn.connect()
```

```
18-07-09:19:00:08 INFO     [docker_container_manager.py:119] Starting managed Redis instance in Docker
18-07-09:19:00:11 INFO     [clipper_admin.py:126] Clipper is running
```


arm

Deploy the model using Clipper

Register application

```
1
2  # Register an application called "hello_world". This will create
3  # a prediction REST endpoint at http://localhost:1337/iris/predict
4  clipper_conn.register_application(name="iris",
5                                    input_type="doubles",
6                                    default_output="-1.0",
7                                    slo_micros=1000000)
8
9  # Inspect Clipper to see the registered apps
10 clipper_conn.get_all_apps()
11
```

```
18-07-09:19:03:00 INFO     [clipper_admin.py:201] Application iris was successfully registered
```

```
['iris']
```


arm

Deploy the model using Clipper

Define the function serve the model

```
# Define a function that serves our trained iris model

# Note that the prediction function takes a list of feature vectors as
# input and returns a list of strings.
def iris_predict(xs):
    print(xs)
    ret = class_names[classifier.predict(xs)]
    print("ret is", ret)
    return ret
```

Deploy the model using Clipper

Deploy the model using python deployer

```
# Deploy the "iris_predict" function as a model. Notice that the application and model
# must have the same input type.
python_deployer.deploy_python_closure(clipper_conn,
                                      name="iris-model",
                                      version=1,
                                      input_type="doubles",
                                      func=iris_predict,
                                      num_replicas=3,
                                      pkgs_to_install=["scikit-learn","scipy"])
```

```
18-07-09:23:49:41 INFO     [deployer_utils.py:44] Saving function to /tmp/clipper/tmpc3h26o7v
18-07-09:23:49:41 INFO     [deployer_utils.py:54] Serialized and supplied predict function
18-07-09:23:49:41 INFO     [python.py:192] Python closure saved
18-07-09:23:49:41 INFO     [python.py:206] Using Python 3.6 base image
18-07-09:23:49:41 INFO     [clipper_admin.py:452] Building model Docker image with model data from /tmp/clipper/tmpc3
h26o7v
```

```
clipper_conn.cm.get_num_replicas(name="iris-model", version='1')
```

```
3
```


arm

Deploy the model using Clipper

Link model to application

```
# Tell Clipper to route requests for the "iris" application to the "iris-model"
clipper_conn.link_model_to_app(app_name="iris", model_name="iris-model")

# Your iris application is now ready to serve predictions
```

```
18-07-09:19:08:20 INFO     [clipper_admin.py:263] Model iris-model is now linked to application iris
```


arm

Deploy the model using Clipper

Query the model

```
q1 = [4.3, 2.0,1.0, 0.1]
q2 = [5.84, 3.05,3.76, 1.2]
q3 = [7.9, 4.4,6.9, 2.5]
```

```
import requests, json, numpy as np
headers = {"Content-type": "application/json"}
```

```
requests.post("http://localhost:1337/iris/predict",
              headers=headers,
              data=json.dumps({"input": q3})).json()
```

```
{'default': False, 'output': 'virginica', 'query_id': 6}
```

```
requests.post("http://localhost:1337/iris/predict",
              headers=headers,
              data=json.dumps({"input": q2})).json()
```

```
{'default': False, 'output': 'versicolor', 'query_id': 7}
```

Asante!

Thank You!

Danke!

Merci!

谢谢!

ありがとう!

Gracias!

Kiitos!

감사합니다

धन्यवाद