aws COMMUNITY DAY

IoT: End to End. Anomaly detection, Dashboarding and Notificationsystem

Gernot Glawe & Malte Walkowiak | 09-09-2019

Community Day 2019 Sponsors

REPLY

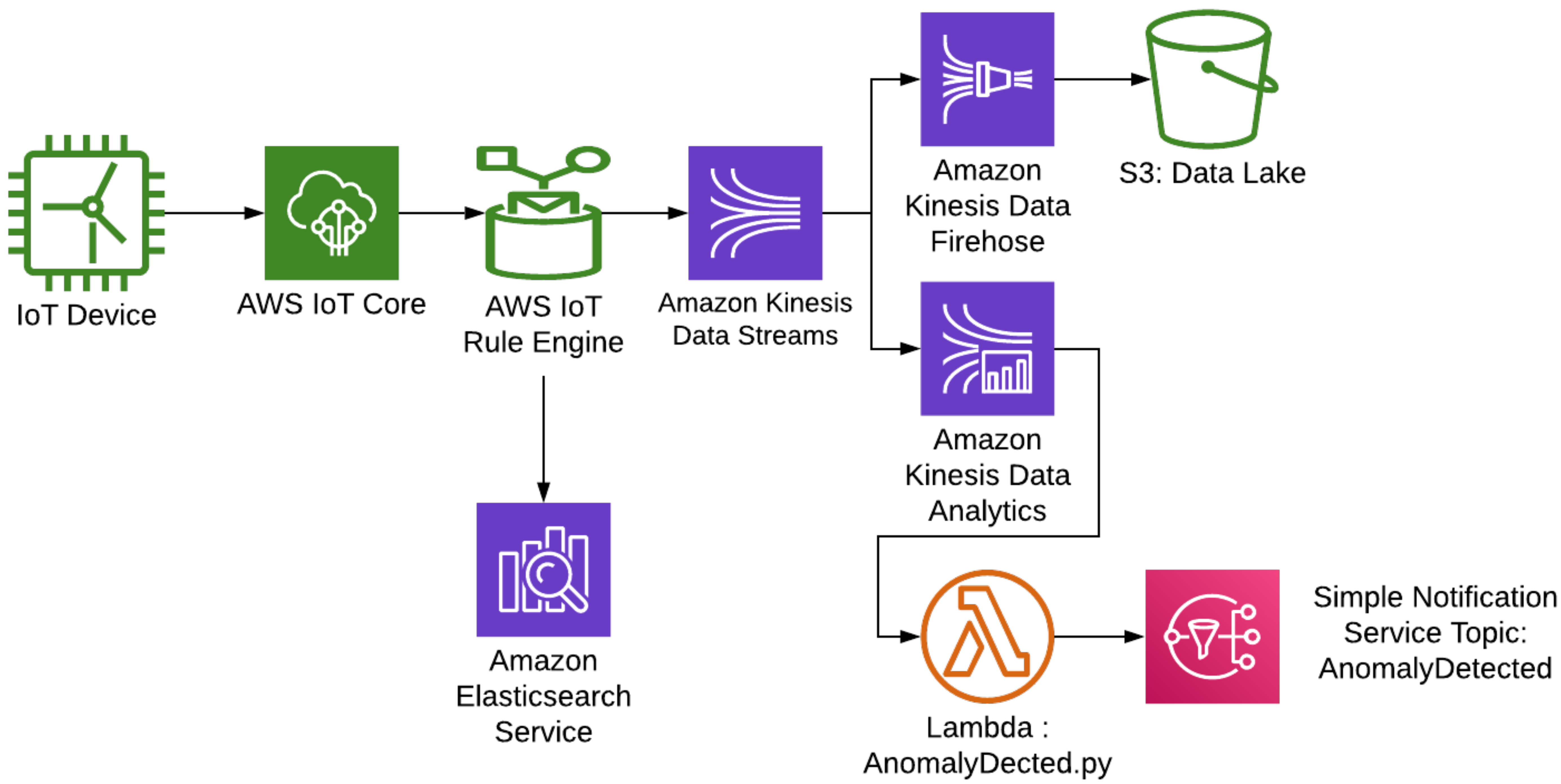
globaldatanet

smaato

Team Internet
Ideas. Change. Markets.

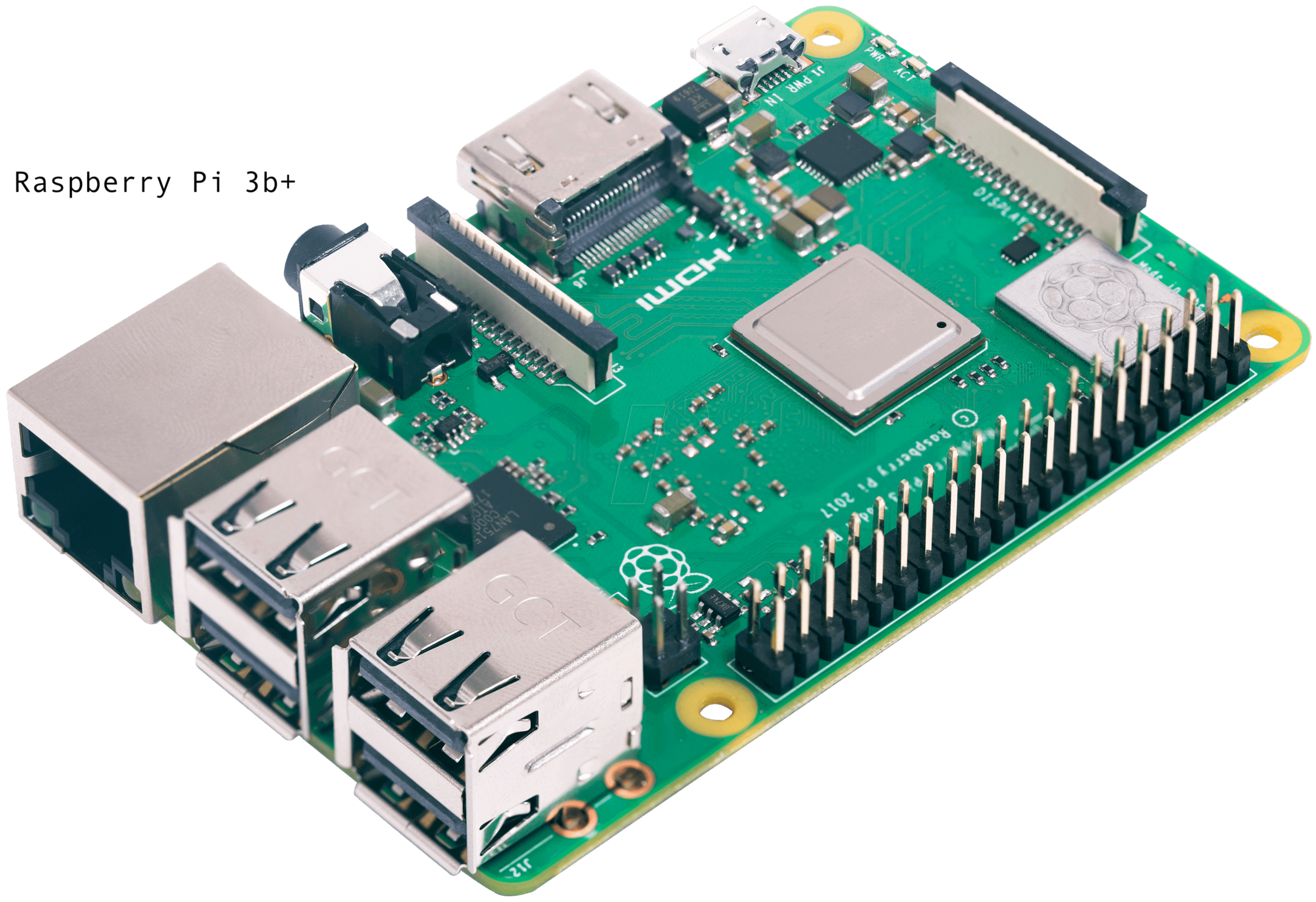
Agenda

- First Milestone: Internet of Things on Edge and AWS
- Second Milestone: Analyze your Data on the Fly
- Third Milestone: Build a Data Repository aka Data Lake
- Fourth Milestone: We need Pictures!
- Final thoughts:
  - Do we have a scalable solution?
  - Try it your self! Cloudformation vs. CDK
- Discussion


IoT Device
AWS IoT Core
AWS IoT
Rule Engine
Amazon Kinesis
Data Streams
Amazon
Kinesis Data
Firehose
S3: Data Lake
Amazon
Kinesis Data
Analytics
Amazon
Elasticsearch
Service
Lambda :
AnomalyDected.py
Simple Notification
Service Topic:
AnomalyDetected

First Milestone: Internet of Things on Edge

Raspberry Pi 3b+



First Milestone: Internet of Things on Edge

```python
import RPi.GPIO as GPIO
from AWSIoTPythonSDK.MQTTLib import \
    AWSIoTMQTTClient  # Import from AWS-IoT Library

# SetUp MQTT Client
myMQTTClient = AWSIoTMQTTClient("new_Client")
myMQTTClient.configureEndpoint(
    "awvkkbwtsweza-ats.iot.eu-central-1.amazonaws.com", 8883)

myMQTTClient.configureCredentials(
    "/home/pi/certs/root-ca-cert.pem",
    "/home/pi/certs/a62755dc63-private.pem.key",
    "/home/pi/certs/a62755dc63-certificate.pem.crt"
)
```

First Milestone: Internet of Things on Edge

```python
# Define RPM function
def fell(n):
    global t, freq, rpm
    dt = time.time() - t
    if dt < 0.01:
        return  # reject spuriously short pulses
    freq = 1 / dt
    rpm = (freq / 2) * 60
    t = time.time()

# Publish Data
while True:
    data = {
        'TIME': datetime.now().isoformat(), "FANID": "ABC123456",
        "FREQ": int(round(freq, 0)), "RPM": int(round(rpm, 0))}
    data_out = json.dumps(data)
    myMQTTClient.publish("IoT_TOPIC/recieve", data_out, 0)
    time.sleep(1)
```

First Milestone: Internet of Things on AWS

- **AWS IoT Device SDKs**
  - Connection, Authentication, and exchange messages with AWS IoT Core using the MQTT, HTTP, or WebSockets protocols.
  - Android, Arduino, iOS, C, C++, Java, JavaScript and Python

First Milestone: Internet of Things on AWS

- **Device Gateway** serves as the entry point for IoT devices connecting to AWS, is fully managed service and scales automatically.
- **Message Broker** is a high throughput pub/sub message broker that securely transmits messages to and from all of your IoT devices and applications with low latency.

First Milestone: Internet of Things on AWS

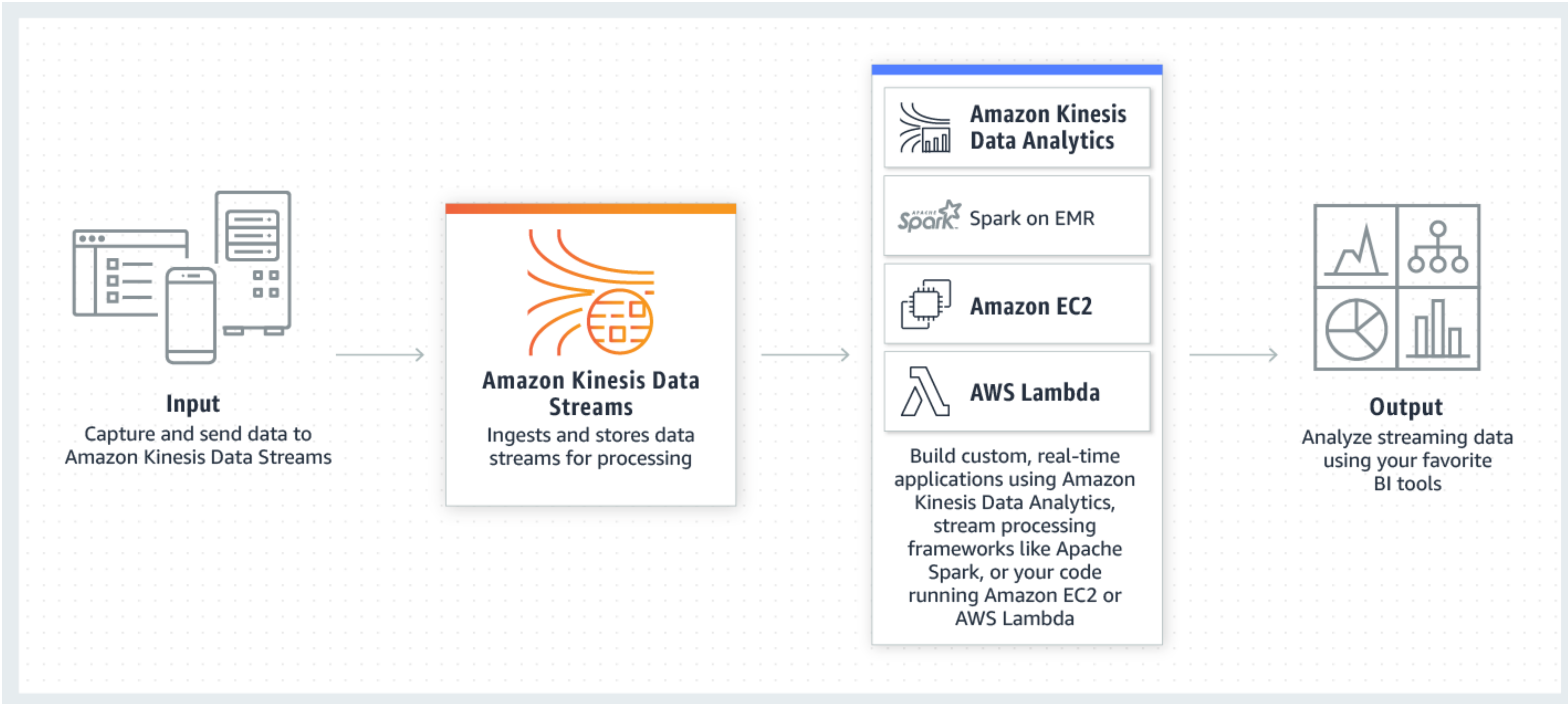
- **Registry** establishes an identity for devices. It assigns a unique identity to each device.
- **Device Shadow** makes it easier to build applications that interact with your devices by providing always available REST APIs.
- **Rules Engine** makes it possible to build IoT applications that gather, process, analyze and act on data generated by connected devices at global scale without having to manage any infrastructure.

Demo!

Second Milestone: Ingest and Analyze your Data on the Fly and make a notification if needed

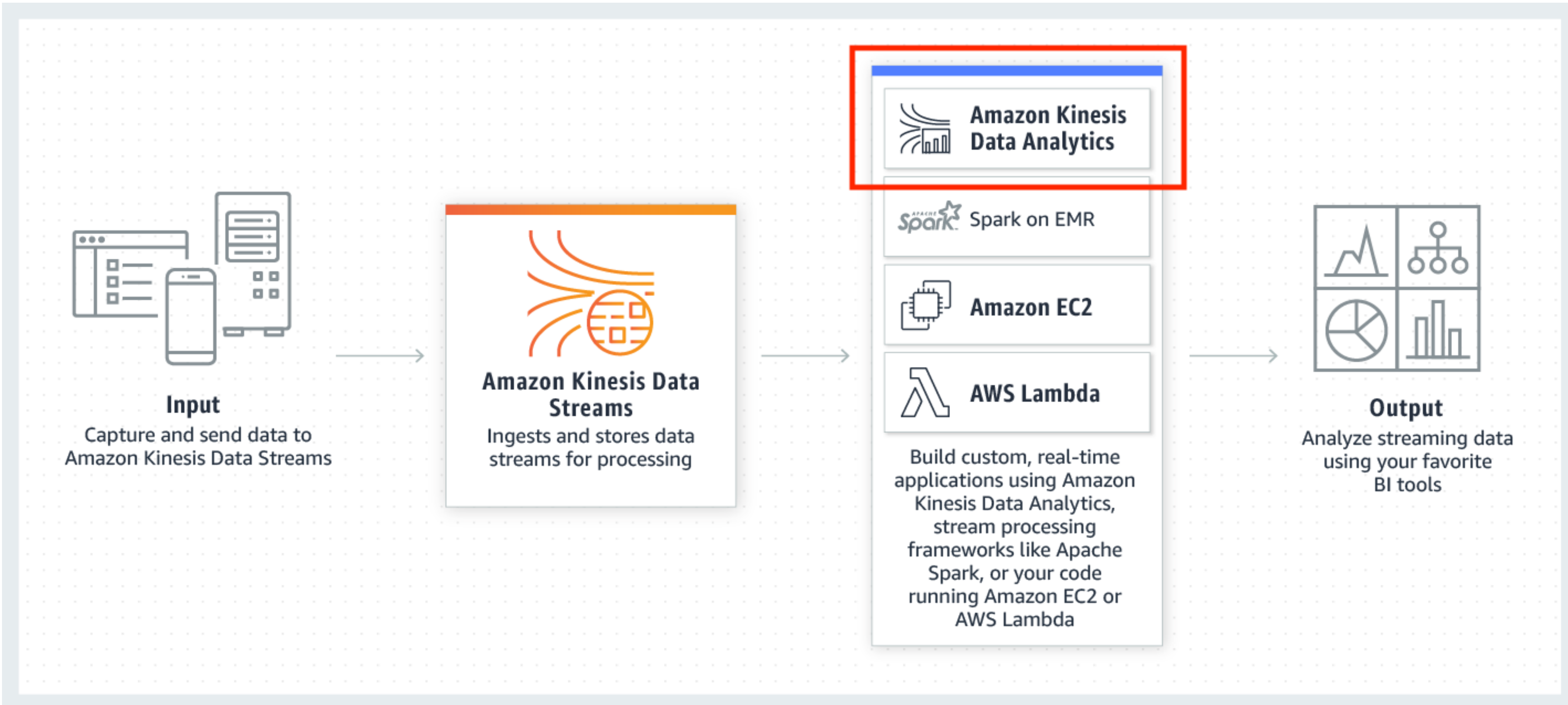
- Kinesis Data Streams
- Kinesis Data Analytics
- Simple Notification Services
- AWS Lambda

Second Milestone: Ingest and Analyze your Data on the Fly and make a notification if needed



* A data producer is an application that typically emits data records as they are generated to a Kinesis data stream.
* A data consumer is a distributed Kinesis application or AWS service retrieving data from all shards in a stream as it is generated.
* A shard is the base throughput unit of an Amazon Kinesis data stream.
* A data stream is a logical grouping of shards. There are no bounds on the number of shards within a data stream (request a limit increase if you need more).

Second Milestone: Ingest and Analyze your Data on the Fly and make a notification if needed



* Support for Standard SQL with an interactive editor to build SQL queries using streaming data operations like sliding time-window averages.
* Amazon Kinesis Data Analytics provides an easy-to-use schema editor to discover and edit the structure of the input data. The wizard automatically recognizes standard data formats such as JSON and CSV. It infers the structure of the input data to create a baseline schema, which you can further refine using the schema editor.
* Pre-built Stream Processing Templates

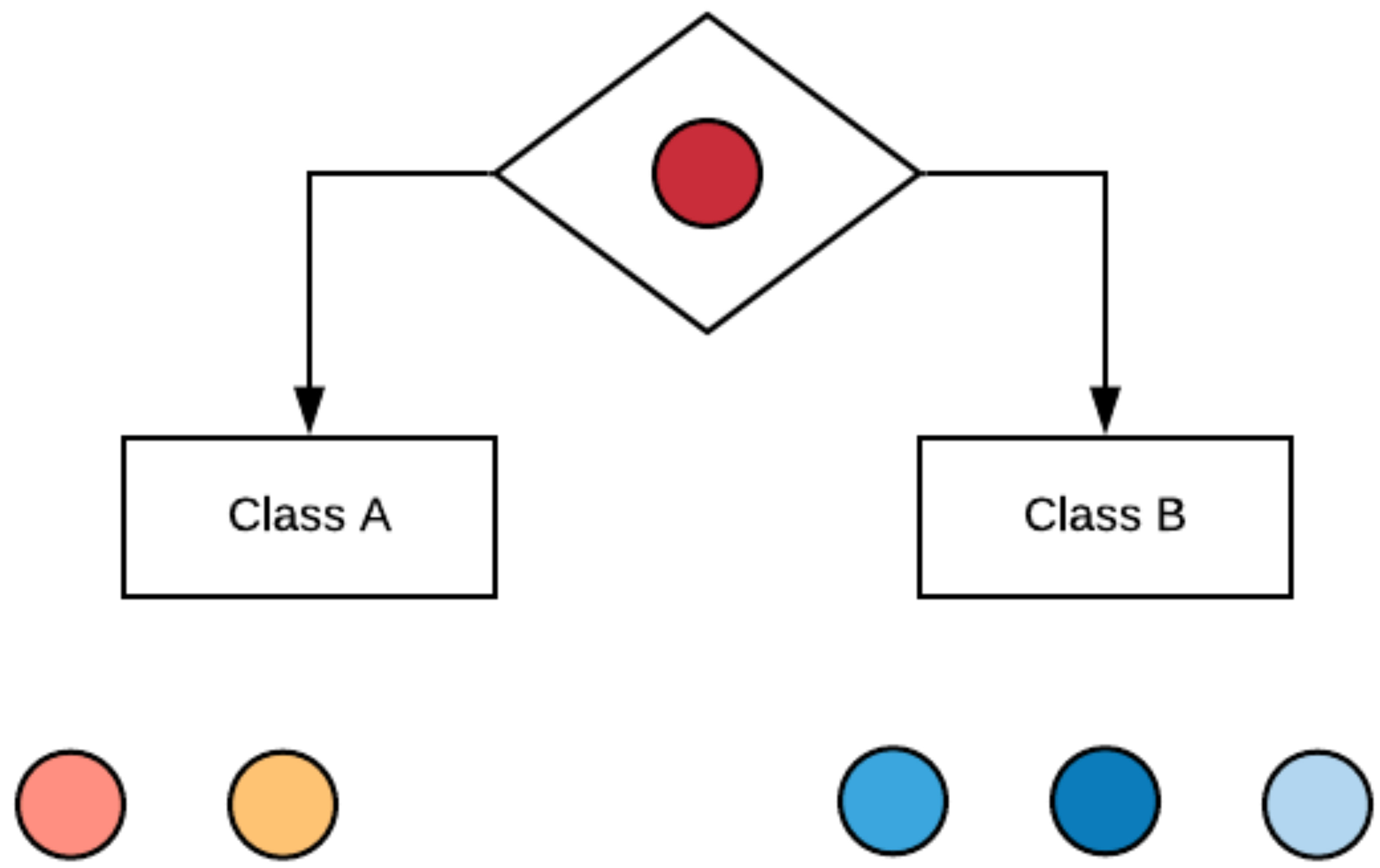
Second Milestone: Ingest and Analyze your Data on the Fly and make a notification if needed

```
--Creates a temporary stream.
CREATE OR REPLACE STREAM "TEMP_STREAM" (
            "FREQ"        INTEGER,
            "RPM"        INTEGER,
            "FANID"         varchar(20),
            "ANOMALY_SCORE"    DOUBLE);

--Creates another stream for application output.
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
            "FREQ"        INTEGER,
            "RPM"        INTEGER,
            "FANID"         varchar(20),
            "ANOMALY_SCORE"    DOUBLE);
```

Second Milestone: Ingest and Analyze your Data on the Fly and make a notification if needed

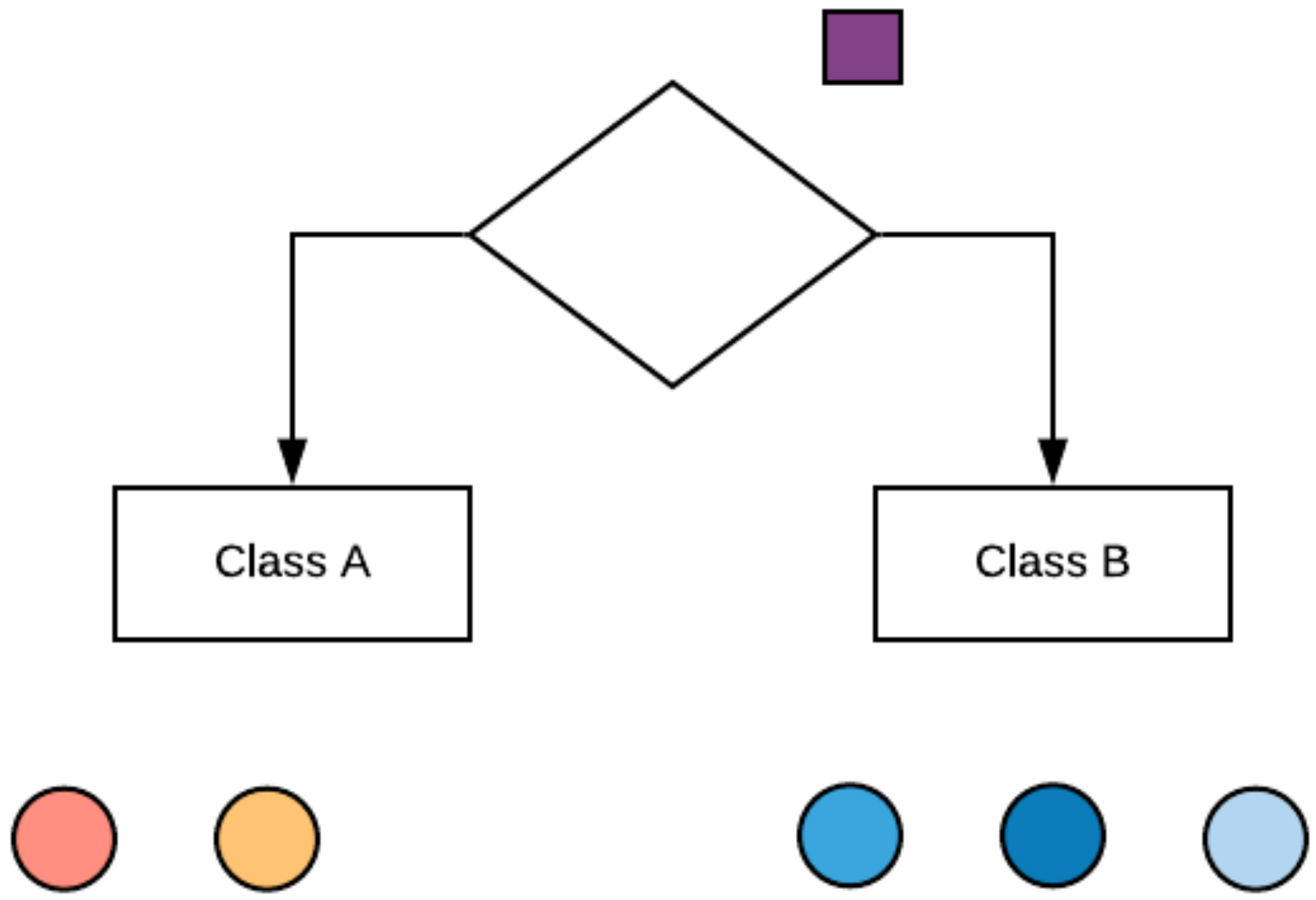
```
-- Compute an anomaly score for each record in the input stream
-- using Random Cut Forest
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
   INSERT INTO "TEMP_STREAM"
      SELECT STREAM "FREQ", "RPM", "FANID", ANOMALY_SCORE
      FROM TABLE(RANDOM_CUT_FOREST(
              CURSOR(SELECT STREAM * FROM "SOURCE_SQL_STREAM_001")));

CREATE OR REPLACE PUMP "OUTPUT_PUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM * FROM "TEMP_STREAM"
      WHERE ANOMALY_SCORE > 6 -- To keep things simple
      ORDER BY FLOOR("TEMP_STREAM".ROWTIME TO SECOND), ANOMALY_SCORE DESC;
```

Random Cut Forest Key Concepts



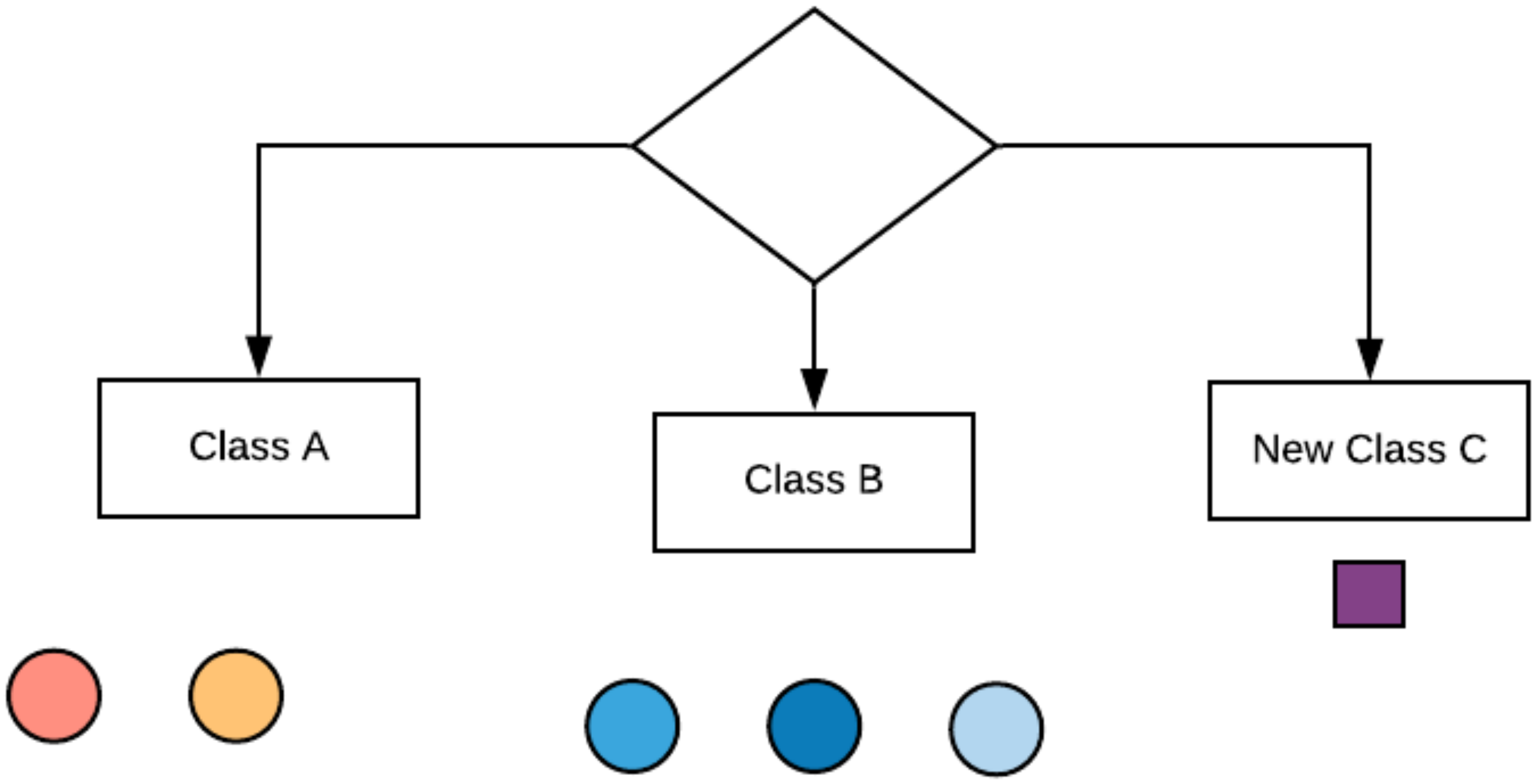
* Random Cut Forest (RCF) is an unsupervised algorithm for detecting anomalous data points within a dataset.
* These are observations which diverge from otherwise well-structured or patterned data. Anomalies can manifest as unexpected spikes in time series data, breaks in periodicity, or unclassifiable data points. They are easy to describe in that, when viewed in a plot, they are often easily distinguishable from the "regular" data.
* Including these anomalies in a dataset can drastically increase the complexity of a machine learning task since the "regular" data can often be described with a simple model.

Random Cut Forest Key Concepts



* The main idea behind the RCF algorithm is to create a forest of trees where each tree is obtained using a partition of a sample of the training data. * For example, a random sample of the input data is first determined. The random sample is then partitioned according to the number of trees in the forest. Each tree is given such a partition and organizes that subset of points into a k-d tree.

Random Cut Forest Key Concepts



* The anomaly score assigned to a data point by the tree is defined as the expected change in complexity of the tree as a result adding that point to the tree; which, in approximation, is inversely proportional to the resulting depth of the point in the tree.
* The random cut forest assigns an anomaly score by computing the average score from each constituent tree and scaling the result with respect to the sample size.

Second Milestone: Ingest and Analyze your Data on the Fly and make a notification if needed

- Simple Notification Service
  - Message Filtering
  - Message fanout
  - Encrypted topics
  - Mobile notifications

Second Milestone: Ingest and Analyze your Data on the Fly and make a notification if needed

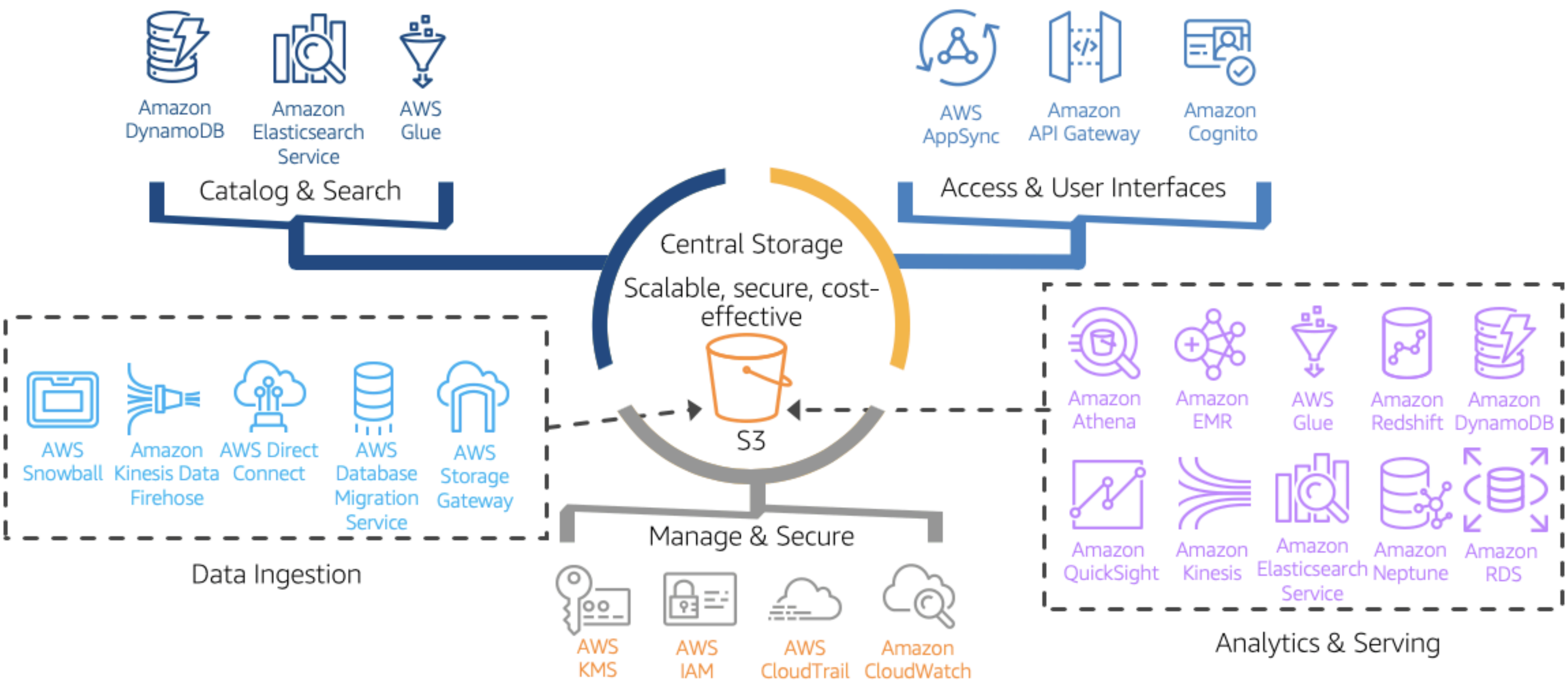
- AWS Lambda
  - Run your code in a event-driven manner
  - Call back-end services for your applications
  - Completely automated administration
  - High Availabillity and Fault Tolerance by default
  - Automatic scaling
  - With AWS Lambda you pay only for the requests served and the compute time required to run your code.

```python
sns_client = boto3.client('sns')

def lambda_handler(event, context):
    print("Received event: " + json.dumps(event, indent=2))
    for record in event['Records']:
        # Kinesis data is base64 encoded so decode here
        payload = base64.b64decode(record['kinesis']['data'])
        msg = json.loads(payload)
        email_msg = 'Anomaly detected on CPU-FAN with the ID ' + msg['FANID']
        print("Decoded payload: ")
        print(email_msg)
        print("SNS topic ARN: " + os.environ['SnsTopicArn'])
        print("Sending anomaly information to SNS topic")
        response = sns_client.publish(
            TopicArn=os.environ['SnsTopicArn'],
            Message=email_msg,
            Subject='Anomaly Detected!',
            MessageStructure='string',
        )
```

Anomaly detected on CPU-FAN with the ID ABC123456

Third Milestone: Build a Data Repository aka Data Lake

- Simple Storage Service
  - object-store
  - extremly durable and high avaible by default
  - pre-serverless service


Amazon DynamoDB
Amazon Elasticsearch Service
AWS Glue
Catalog & Search
AWS AppSync
Amazon API Gateway
Amazon Cognito
Access & User Interfaces
Central Storage
Scalable, secure, cost-effective
S3
AWS Snowball
Amazon Kinesis Data Firehose
AWS Direct Connect
AWS Database Migration Service
AWS Storage Gateway
Data Ingestion
Manage & Secure
AWS KMS
AWS IAM
AWS CloudTrail
Amazon CloudWatch
Amazon Athena
Amazon EMR
AWS Glue
Amazon Redshift
Amazon DynamoDB
Amazon QuickSight
Amazon Kinesis
Amazon Elasticsearch Service
Amazon Neptune
Amazon RDS
Analytics & Serving

Third Milestone: Build a Data Repository aka Data Lake

- Kinesis Firehose
  - Scales up and down automatically
  - Support for Built-In Data Format Conversion (to Parquet, ORC)
  - Can be used to invoke AWS Lambda (Data Preparation)
  - Data can be automatically encrypted after it is uploaded to the destination.
  - Builds up your Data Lake

Demo!

Fourth Milestone: We need Pictures!

- Kibana on Elasticsearch Service
  - Fully managed: You can deploy a production-ready Elasticsearch cluster in minutes
  - Steps: creating a new domain and specifiing the number of instances
  - Get direct access to the Elasticsearch APIs to load, query, analyze data, and manage indices.
  - Easy data ingestion: You can easily ingest structured and unstructured data into your Amazon Elasticsearch domain with Logstash
  - Kibana is automatically deployed with your Amazon Elasticsearch Service domain.

Demo!

Final thoughts: Do we have a scalable solution?

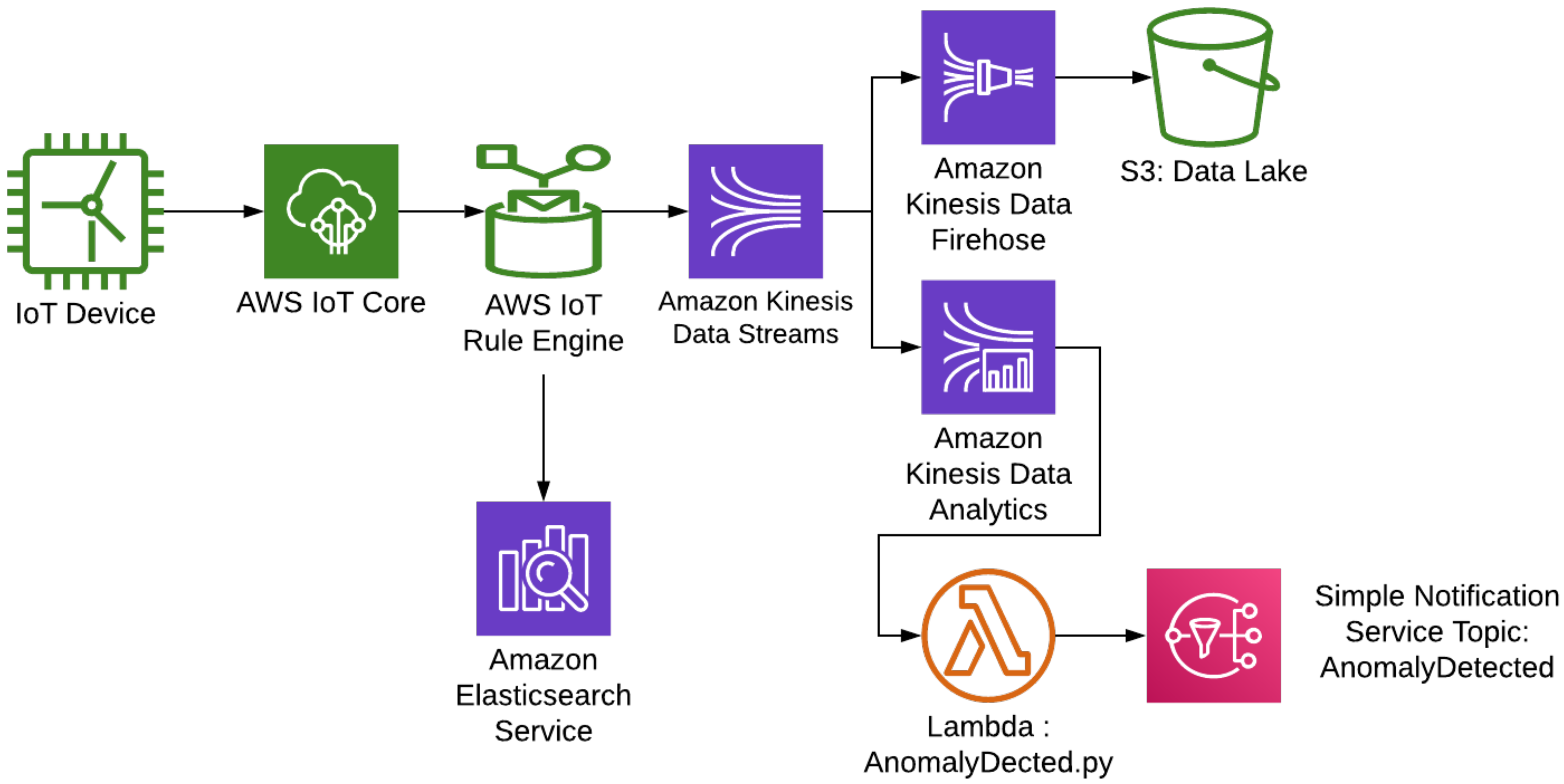
Automated Deployment

- Logical Architecture
- CloudFormation complexity
- CDK architecture simplifies

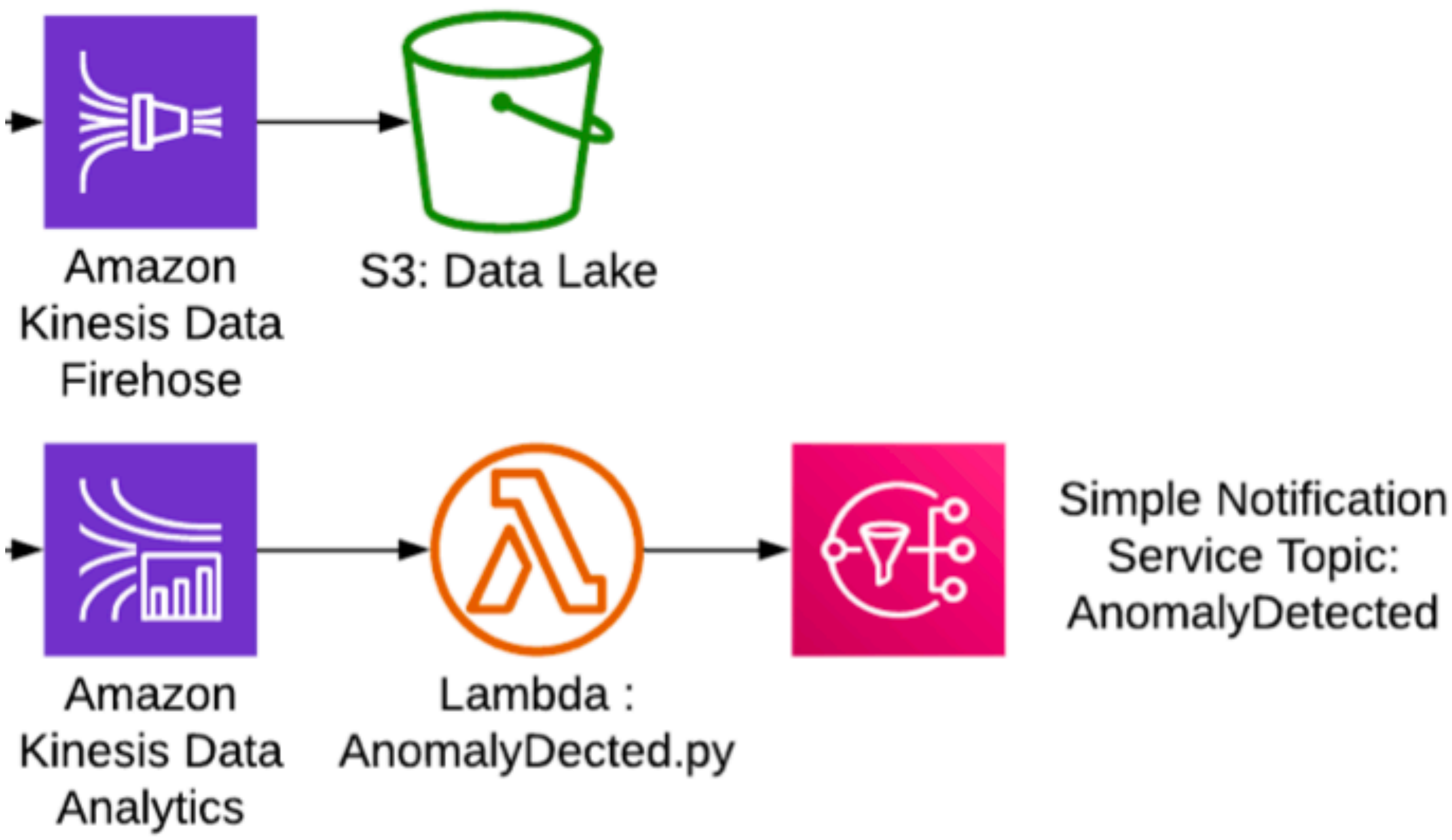
Logical Architecture is a blueprint

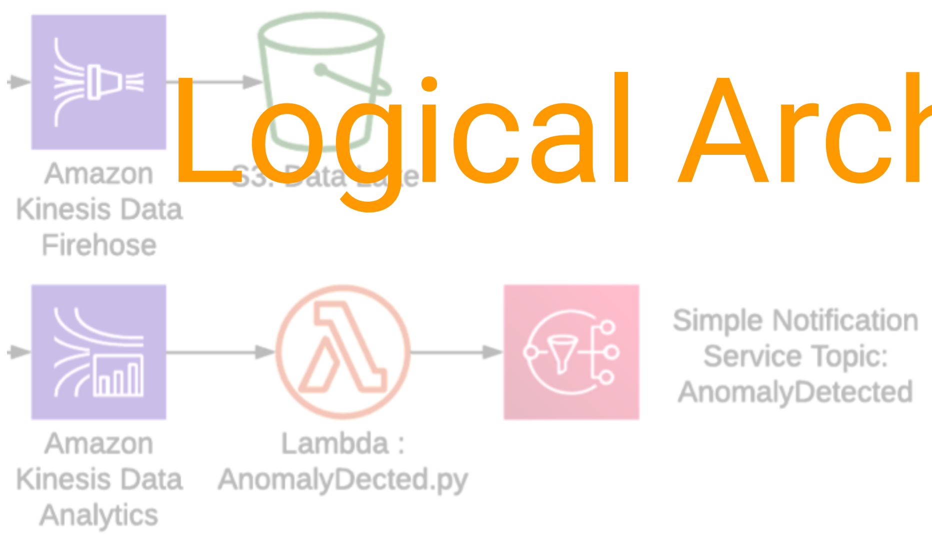
- reduces complexity
- simplifies
- ignores something
- is something to talk about

Logical Architecture

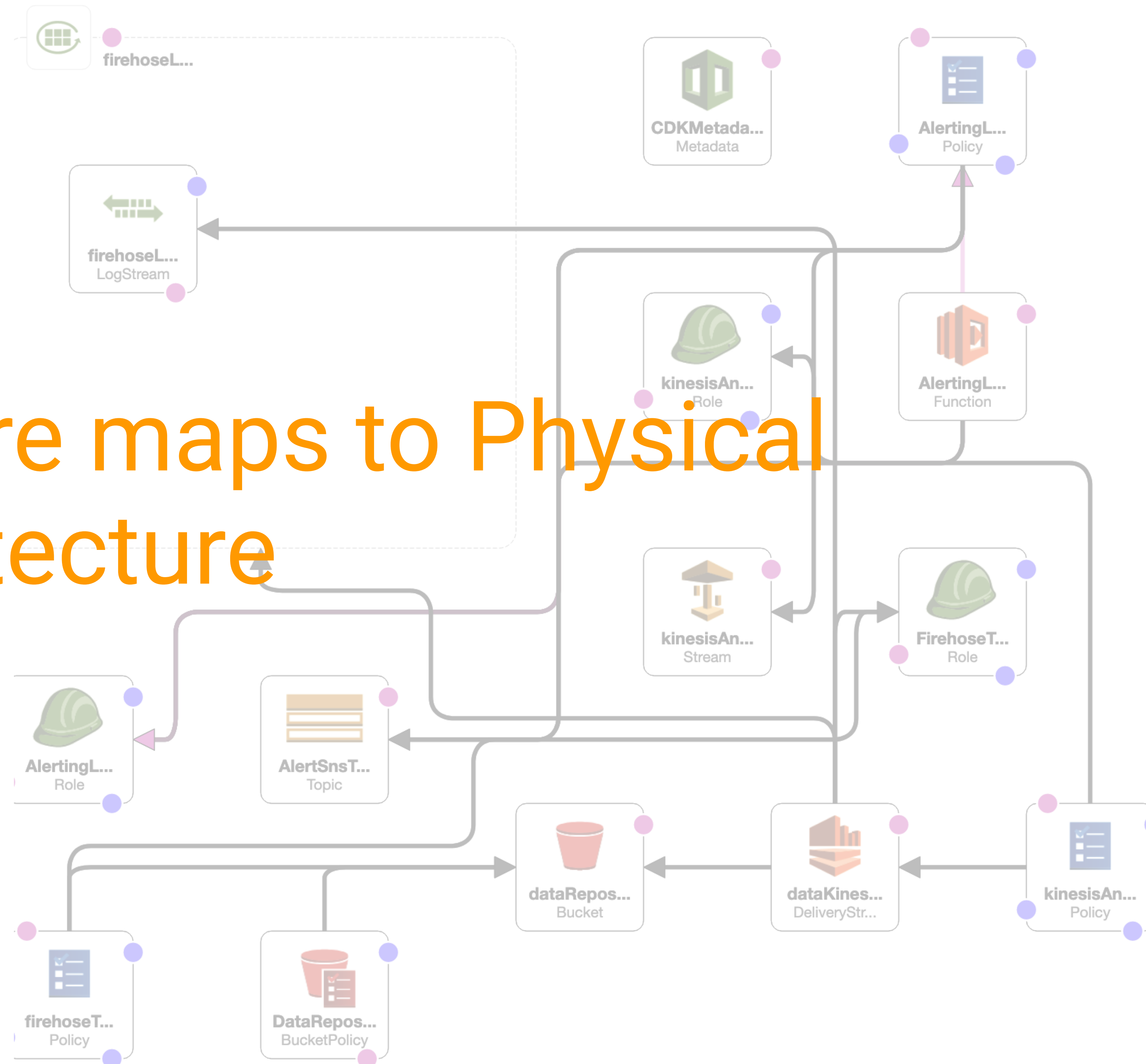


Logical Architecture of Moving Parts

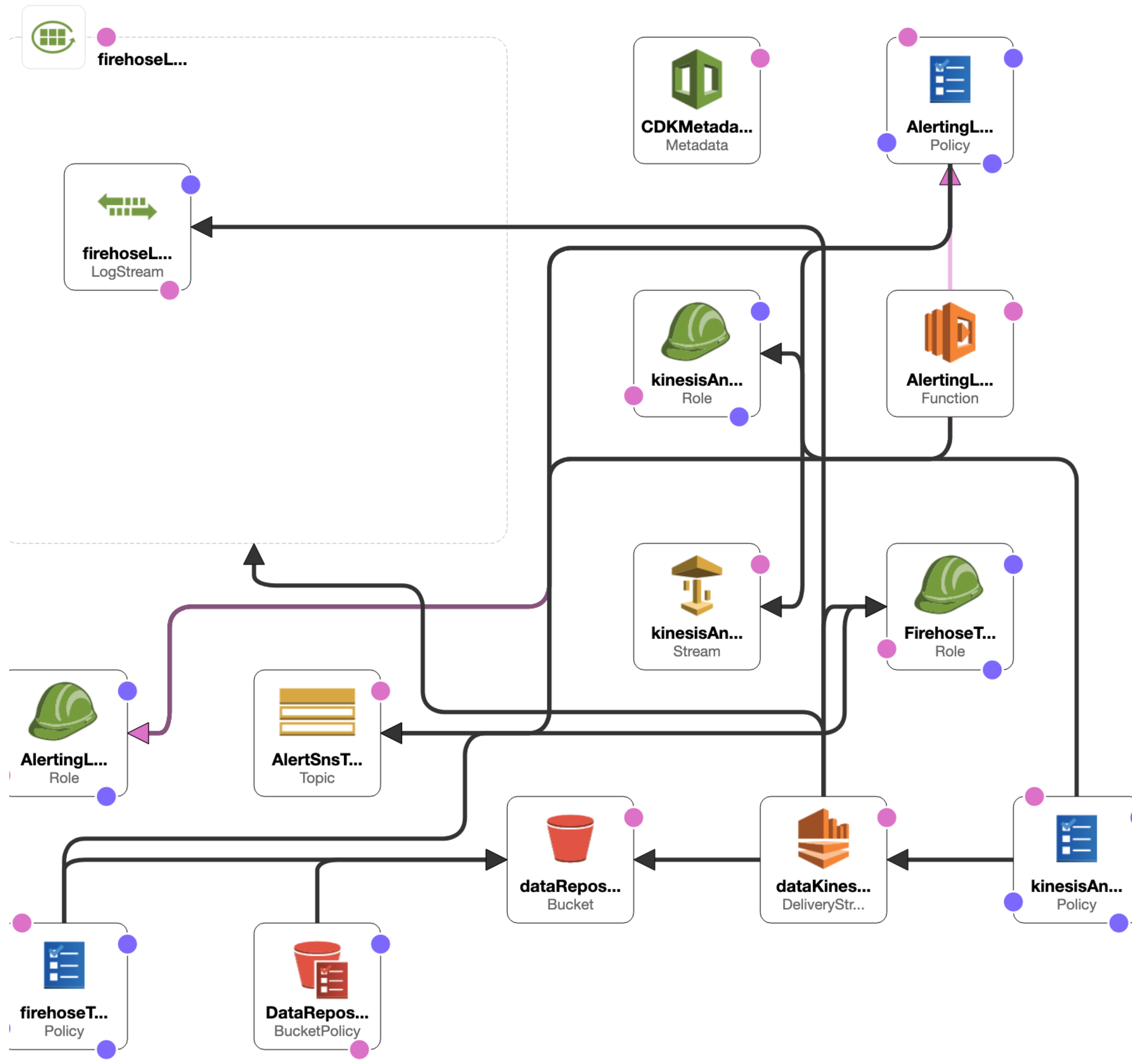




Logical Architecture maps to Physical Architecture

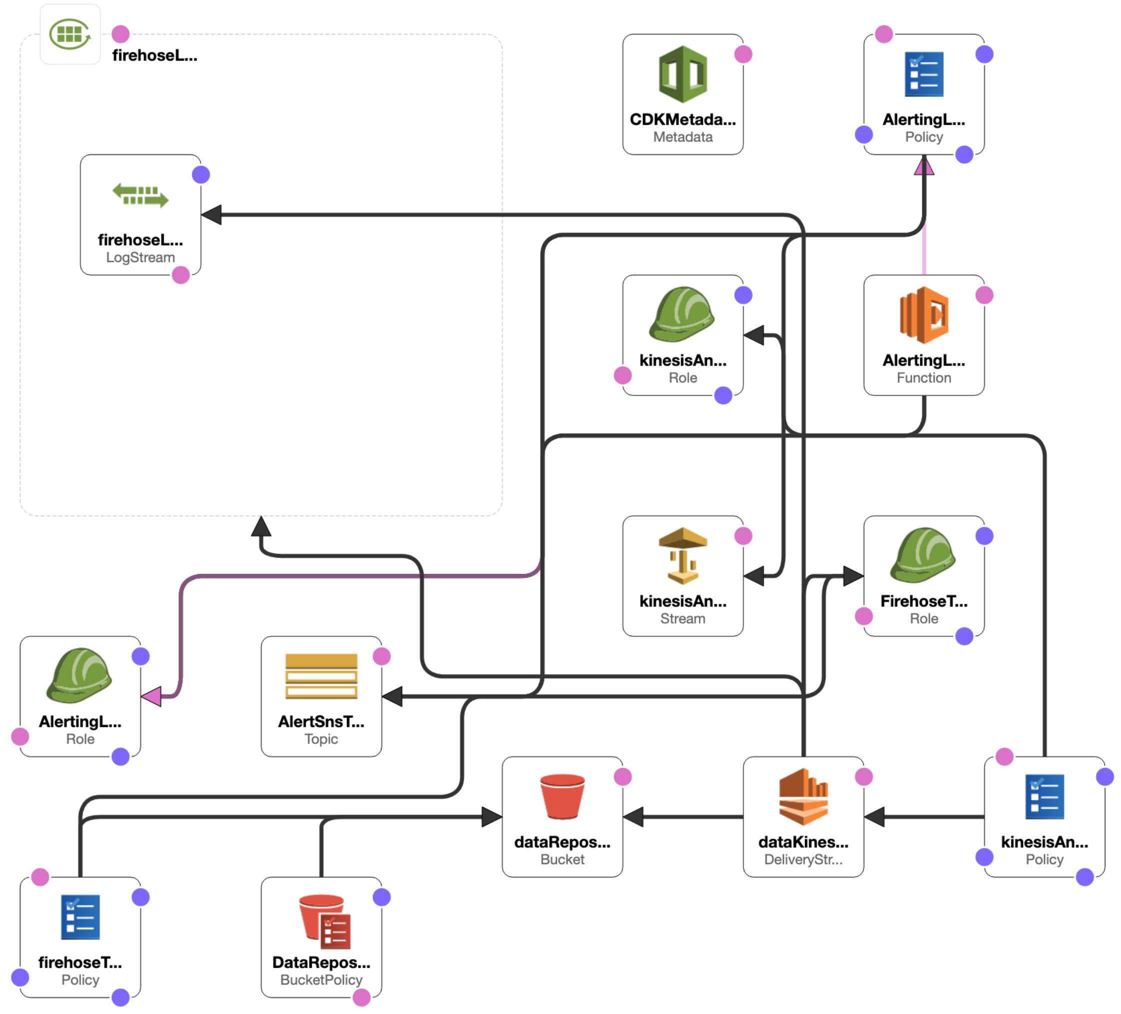


Physical Architecture of Moving Parts



Physical architecure: CloudFormation complexity

- Referencing
- Explicit roles
- More Parts
- …



Template complexity

```
# Referencing
   Resource: !Join [ "", [ "arn:aws:firehose:",
    !Ref "AWS::Region", ":", !Ref "AWS::AccountId", ":deliverystream/", !Ref DataKinesisFirehose ] ]

# Explicit roles
    Effect: "Allow"
    Action:
    - "sns:GetTopicAttributes"
    - "sns:ListTopics"
    - "sns:Publish"
    Resource: !Ref AlertSnsTopic
```

CDK simplification Referencing

```
Resource: !Join [ "", [ "arn:aws:firehose:",
!Ref "AWS::Region", ":", !Ref "AWS::AccountId", ":deliverystream/", !Ref DataKinesisFirehose ] ]
```

becomes more understandable

```
resources: [dataKinesisFirehose.ref],
```

CDK simplification Explicit roles

Physical view

```
Effect: "Allow"
Action:
- "sns:GetTopicAttributes"
- "sns:ListTopics"
- "sns:Publish"
Resource: !Ref AlertSnsTopic
```

becomes (more) logical view

```
alertingTopic.grantPublish(analyticsLambda);
```

Migration from CloudFormation to CDK

- took 2 1/2 hours
- 270 lines of code to 150 loc
- created a more logical representation

but

- you need code snippets
- some new paradigm to learn

Appendix

CDK Code Snippets

- AWS: `https://github.com/aws-samples/aws-cdk-examples`
- tecRacer: `https://github.com/tecracer/cdk-templates`

Demos

- Malte Walkowiak: `https://github.com/Zirkonium88/AWS`