Common Architectures

Rahul Ramaka

SDE-I Intern
AWS DevOpsGuru London

Background

Customers of Amazon DevOpsGuru can select which CloudFormation stacks to monitor.

We refer to a group of related resources within a CloudFormation as an AppGroup and each AppGroup has an architecture.

By identifying common architectures used by our customers we will be able to better prioritise support for new proactive insights and improved recommendations etc.

Presentation Structure

This presentation aims to answer two main questions:

- How do we identify and group together similar architectures?
- How do we integrate these results with HitRate?

Commonality ranking algorithm

Week 1-5

Feature Extraction

What are some useful features we can extract from an AWS architecture/AppGroup by which we can categorise them?

- Resources (type, cardinality, monitored or unmonitored)
- Connections between resources (direction and amount of data flow)

Let’s build a dataset!

Cloudformation resource types *https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html*

DevOpsGuru Pricing: *https://aws.amazon.com/devops-guru/pricing/*

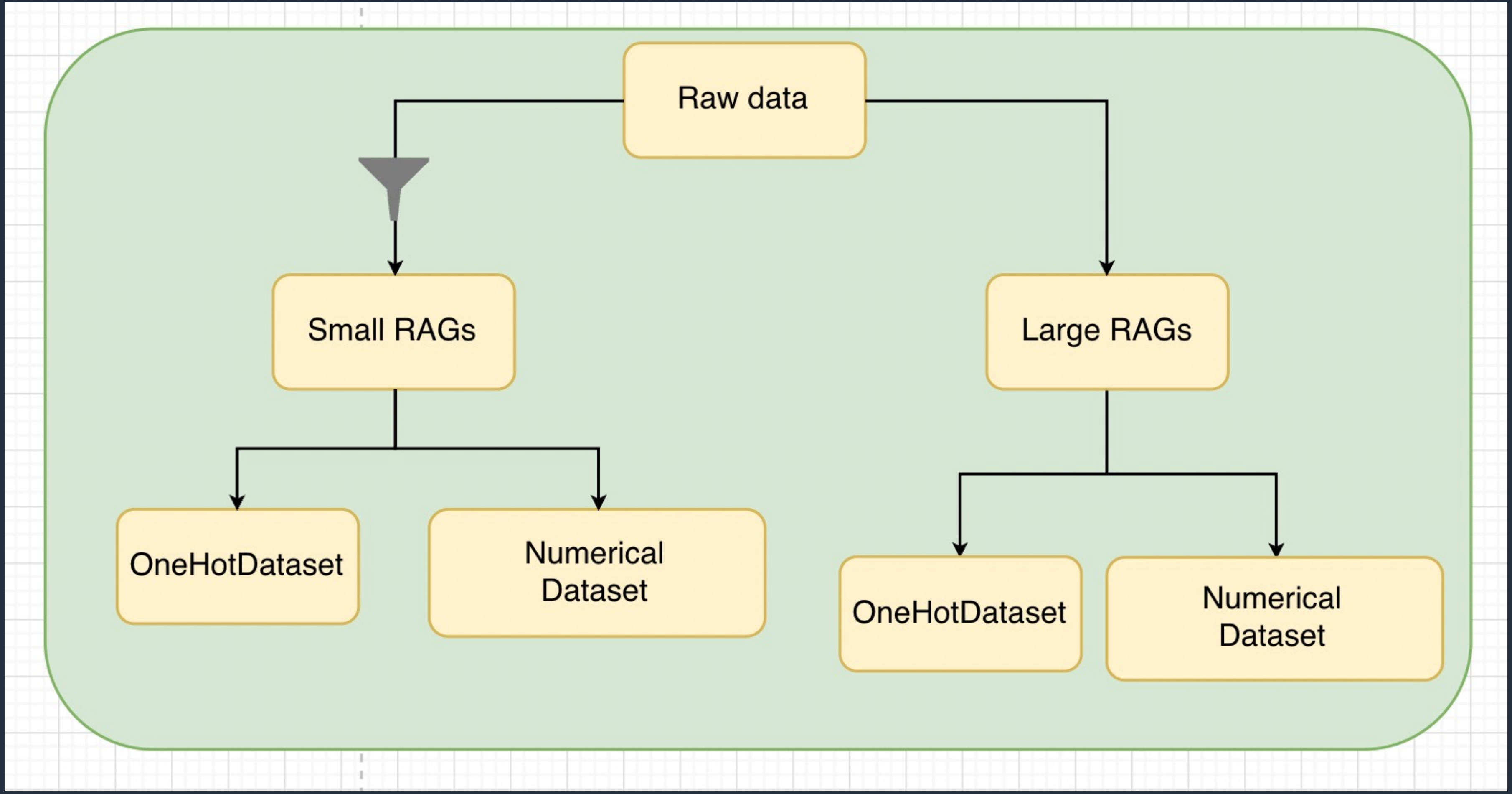
Feature Extraction (Numerical or One-hot Encoding)

```
>vec = FE("ff900585-5479-4edf-b4b1-fd3ac149fc71": {
        "AWS::ApiGateway::Account": 1,
        "AWS::ApiGateway::RestApi": 5,
        "AWS::ApiGateway::Stage": 1,
        "AWS::Lambda::Function": 1
})

[0,0,0,0,1,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,5,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

>vec.shape
(1x38)
```

Feature Extraction



Similarity Measures

We're dealing with vector spaces over natural numbers. To express similarity, we need a distance function (or *norm* or *inertia*) which output a smaller distance for similar vectors. Possible options include:

- Hamming distance (*L0* norm) – discrete metric not distance
- Manhattan distance (*L1* norm)
- Euclidean distance (*L2* norm)
- Chebyshev distance (*L-infinity* norm)
- Minkowski distance (L-p norm)
- Cosine distance

Lp spaces: https://en.wikipedia.org/wiki/Lp_space

Similarity Measures

It is theoretically possible to prove that higher norm metrics (higher values of p) lose meaning and provide a poorer contrast when we increase the number of dimensions (features) of our data. This is also known as the *curse of dimensionality*.

On the Surprising Behavior of Distance Metrics in High Dimensional Space - Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim

Building a clustering model

We used centroid-based clustering where each cluster is represented by a central vector.

Goal: *find the k cluster centers and assign the objects to the nearest cluster center, such that distance function from the cluster to the objects is minimized.*

A particularly well known approximate method which does this is Lloyd's algorithm also known as *k*-means algorithm.

K-means vs k-median vs k-modes vs k-medioids

The loss functions and their corresponding clustering algorithms are shown below:

Euclidean distance -> K-means

Manhattan distance - > K-medians

Hamming distance -> K-modes

Minkowski distance -> K-mediods

PCA to the rescue!

Multiple dimensions are hard to think in, impossible to visualize.

Hence to improve the efficiency and accuracy of mining task on high dimensional data, we use dimensionality reduction methods such as Principal Component Analysis (PCA) which *increase* interpretability but at the same time *minimize* information loss.

Finding such new variables, the principal components, reduces to solving an eigenvalue/eigenvector problem.

PCA to the rescue!

```
{"ff900585-5479-4edf-b4b1-fd3ac149fc71": {
        "AWS::ApiGateway::Account": 1,
        "AWS::ApiGateway::RestApi": 5,
        "AWS::ApiGateway::Stage": 1,
        "AWS::Lambda::Function": 1
},{"3c5dc579-fdf8-4a01-9bef-8f8dcdcad0c4": {
        "AWS::S3::Bucket": 10,
        "AWS::Lambda::Function": 1
}}
```

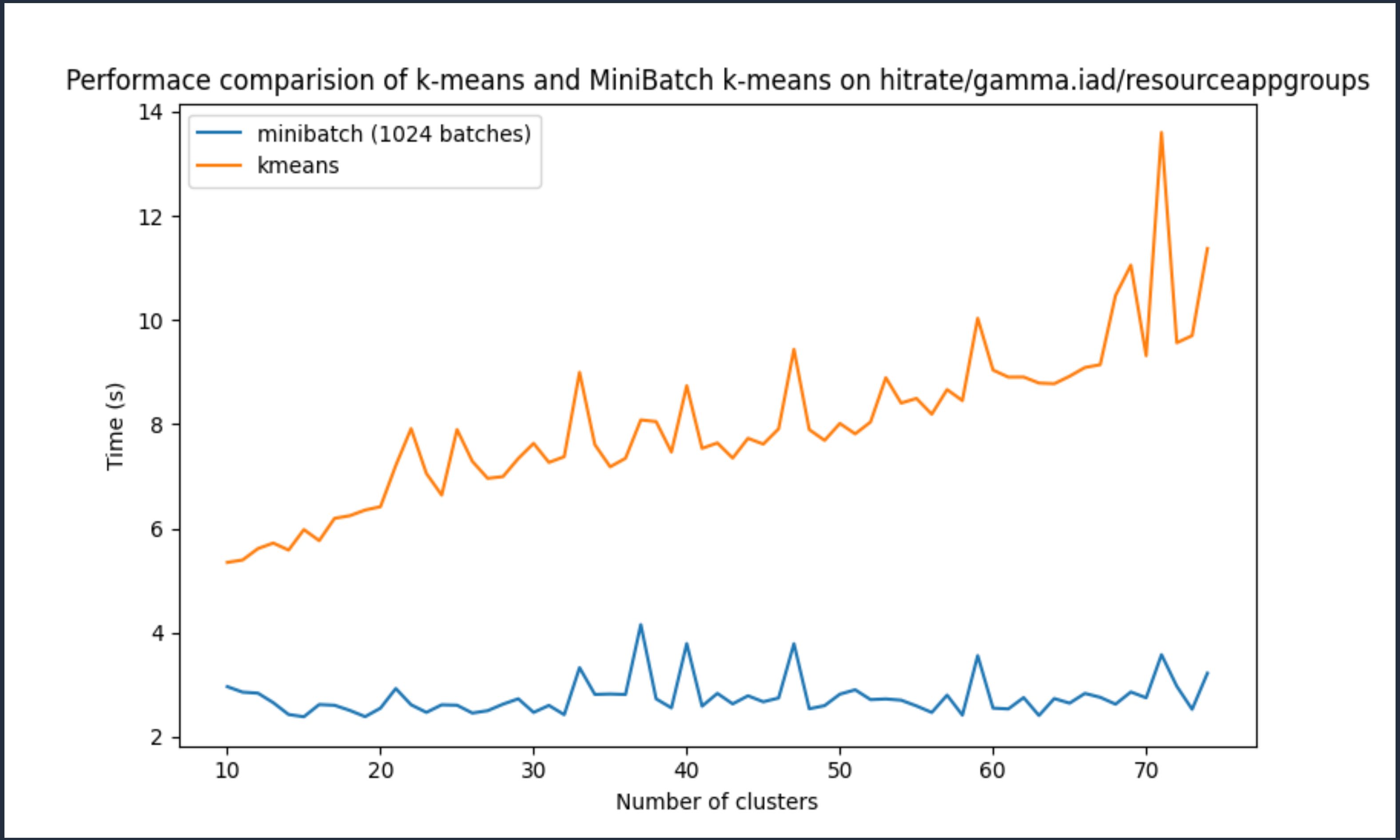
~>

```
[[0,0,0,0,1,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,5,0,0,0,0,0,0,0,0,0,0,0,0,0],
 [0,0,0,0,1,0,0,0,0,10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]]
```

~>

```
[[ 0.16114089,  0.32491763, -0.51388278],
[-0.3903814 , -0.18516744,  0.64447917]]
```

Scalable clustering



https://github.com/pyutils/line_profiler

Web-Scale K-Means Clustering

Evaluating clusters

We will be evaluating the cluster algorithm results for MiniBatch k-means using internal validation techniques listed below:

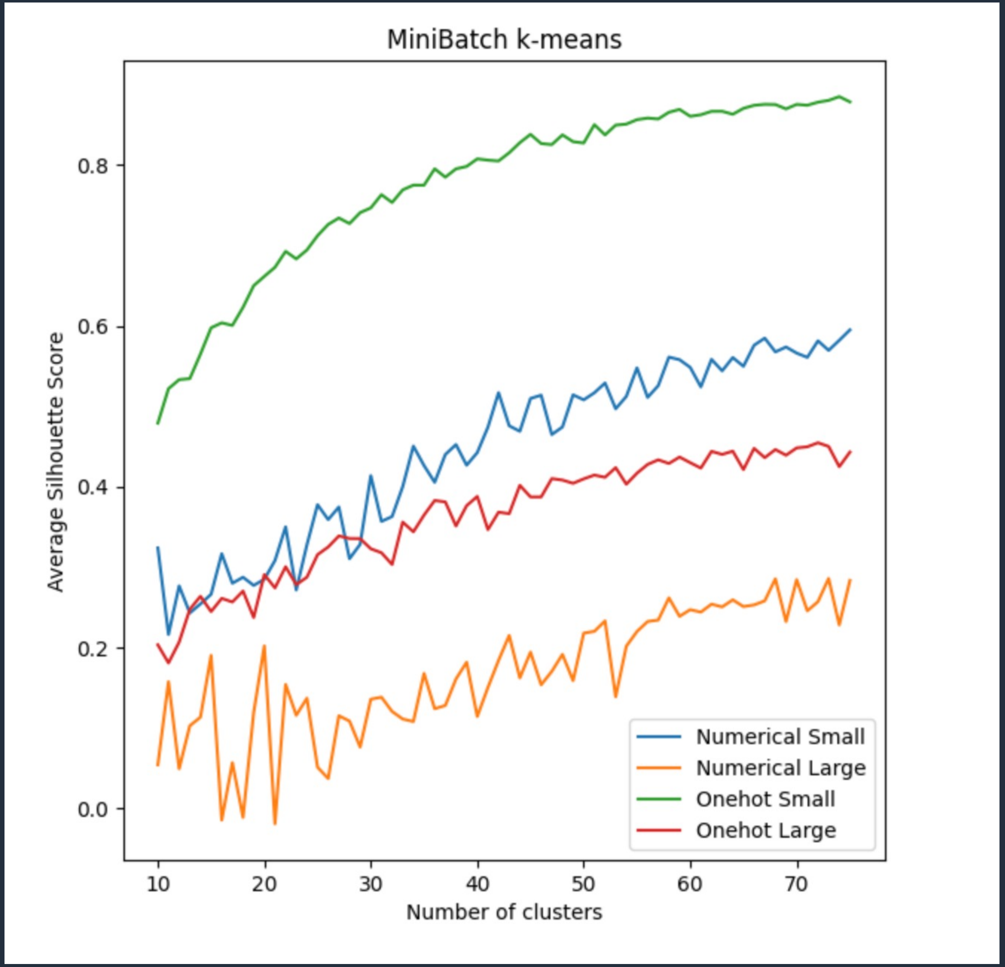
- Silhoutte Score
- Calinski-Harabasz Index
- Davies-Bouldin Index

All of these scores measure cluster cohesion and separation based on rules set by nominal paper below.

Evaluation Metrics for Unsupervised Learning Algorithms

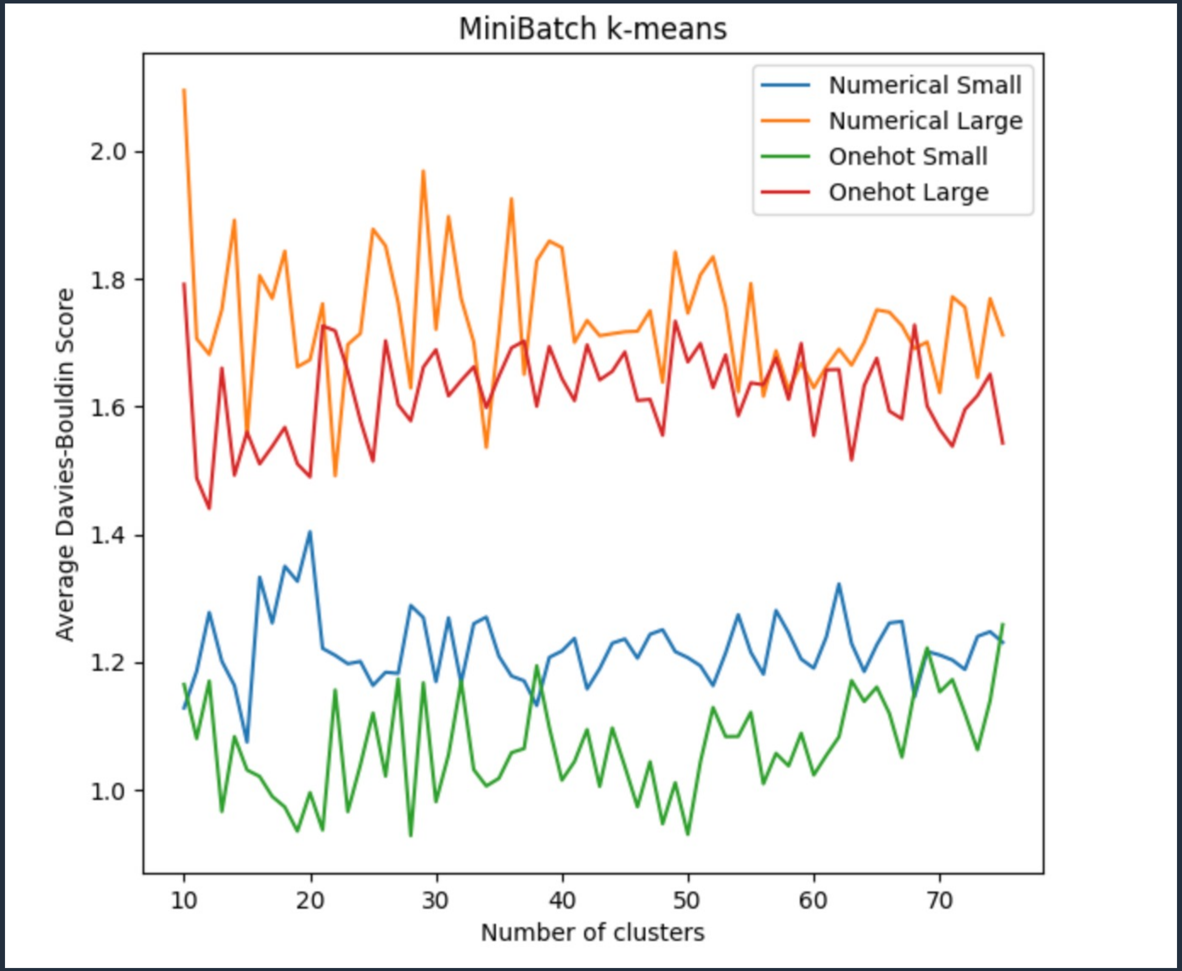
Evaluating clusters 2

Higher Silhouette score is better



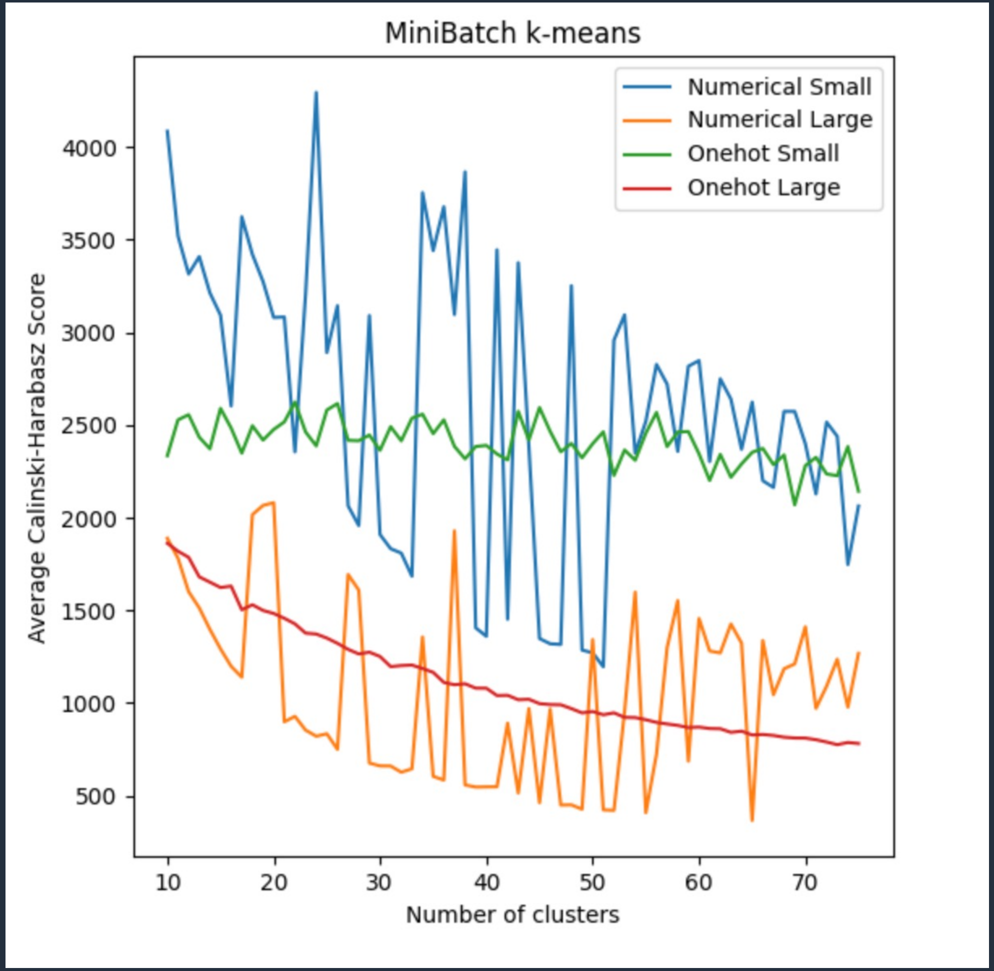
Evaluating clusters 3

Lower Davies-Bouldin Index is better



Evaluating clusters 4

Higher Calinski-Harabasz Index is better



Evaluating clusters

Using a grid-search over 3 evaluation metrics and clusters ranging from [10,70) , it was found that:

- k = [40-46, 54] gives the highest average scores
- Low-dimensional data (small RAGs) performs well with OneHot encoding
- High-dimensional data (large RAGs) performs well with Numerical encoding

Cloud-based unsupervised learning

Week 5-9

Building blocks

Step 1: Ingest AppGroup data from HitRate

Step 2: Preprocess data to create feature vectors and run PCA

Step 3: Generate minibatch k-Means clustering model

Step 4: Deploy model to a HTTP/boto3-queryable endpoint

SageMaker Pipelines

From the AWS SageMaker page:

"SageMaker Pipelines takes care of everything from data preprocessing using DataWrangler workflows, training, model evaluation, versioning using Model Registry and deployment using an endpoint. Combined with Amazon SageMaker Studio, the first fully integrated development environment (IDE), they both come at no additional charge to the Studio Notebooks."

Development

AWS CDK

- IaaC tool which provisions infrastructure through AWS Cloudformation
- Used to build CapstoneHitrateMLCDK

PicaPica

- Internal tool which offers CodeCommit Replicas that makes source code in GitFarm available in AWS for deployment.
- Used to mirror CapstoneCommonArchitecturesSageMakerPipeline which defines the SageMaker Pipeline

SageMaker Python SDK & SageMaker Local Mode

- Allows local testing of SageMaker Pipeline using custom images loaded in Docker containers

Inference pipeline

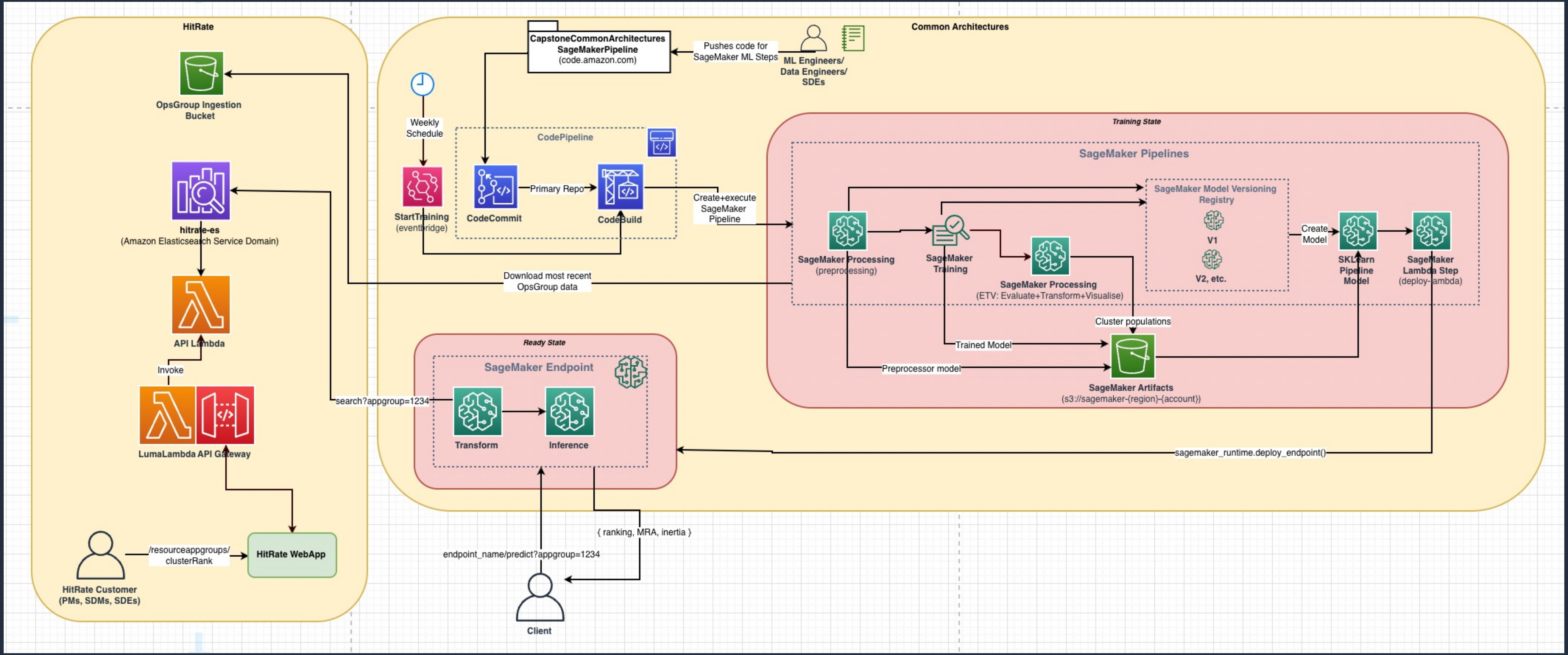
Week 9-10

SageMaker Endpoint

Although 40% more expensive than their EC2 equivalent, SageMaker Endpoint instances were picked due to their easy integration with SageMaker Pipeline. An alternative ApiGateway-> Lambda+EFS pattern was considered but rejected since we would be exiting the SageMaker ecosystem.

Next Steps talks about reducing costs of our SageMaker Endpoint.

Architecture Design


HitRate
OpsGroup Ingestion Bucket
hitrate-es
(Amazon Elasticsearch Service Domain)
API Lambda
Invoke
LumaLambda API Gateway
HitRate WebApp
/resourceappgroups/clusterRank
HitRate Customer (PMs, SDMs, SDEs)
search?appgroup=1234
Download most recent OpsGroup data
Common Architectures
CapstoneCommonArchitectures SageMakerPipeline (code.amazon.com)
Pushes code for SageMaker ML Steps
ML Engineers/ Data Engineers/ SDEs
Weekly Schedule
StartTraining (eventbridge)
CodePipeline
CodeCommit
Primary Repo
CodeBuild
Create+execute SageMaker Pipeline
Training State
SageMaker Pipelines
SageMaker Processing (preprocessing)
SageMaker Training
SageMaker Processing (ETV: Evaluate+Transform+Visualise)
SageMaker Model Versioning Registry
V1
V2, etc.
Create Model
SKLearn Pipeline Model
SageMaker Lambda Step (deploy-lambda)
Cluster populations
Trained Model
Preprocessor model
SageMaker Artifacts (s3://sagemaker-(region)-(account))
Ready State
SageMaker Endpoint
Transform
Inference
sagemaker_runtime.deploy_endpoint()
{ ranking, MRA, inertia }
endpoint_name/predict?appgroup=1234
Client

Estimated Cost of Common Architectures

Region	Instance Type	Hourly Cost($)	Hours used per month	Monthly Cost($)
Processing	ml.t3.medium	$0.05	0.67	<$0.05
Training	ml.m5.large	$0.115	0.08	<$0.115
Inference	ml.t2.medium	$0.06	720	$40.32
Sum				$40.52

Demo

FAQs

FAQ 1

How do we know if the clusters are accurate?

Ans: We've manually verified the cluster accuracy based on the inertia of samples in cluster population. It is also a starting point to create a labelled dataset of common architectural patterns. This would be the clean, validated, golden dataset which now shifts the problem from an unsupervised learning approach to supervised learning approach like k-nearest neighbors which knows what architectural patterns to look for in AppGroups.

FAQ 2

How do we extend this solution to graphs?

Ans: We can use the same classification techniques as we used in Common Architectures but the similarity measures are different. The definition may vary from simple like Isomorphism which identifies a bijection between two edge sets, or if there exists a an invertible matrix which can prove two adjacency matrices (graphs) are similar.

Other, more established methods include Jaccard similarity, Edit distance, Subgraph matching, Graph kernels etc.

FAQ 3

Have you considered any non-ML approaches?

Ans: Locally-sensitive hashing was initially considered too and is still a viable solution once the dataset we're dealing with gets bigger and the velocity with which results are requested gets faster.

- LSH utilizes approximate nearest neighbors algorithms in the form of hash functions.
- Similar items are hashed into the same buckets with a high probability and thereby clustering in effect.

Next Steps

Next Step 1

Invoking endpoint API with Hitrate webapp

Next Step 2

Moving to SageMaker Serverless Inference

- The estimated costs for the Common Architectures by making the SageMaker Endpoint instances autoscalable based on traffic patterns.
- Serverless Inference is an example which deals with spiky traffic.
- There were some implementation issues that came with this which couldn’t be tackled due to time constraints.

Next Step 3

Elbow method

- Currently, *k* (number of clusters) is pre-determined based on grid-search using evaluation results seen before.
- We can make this automatic

Next Step 4

K-medioids and K-modes clustering

- We used PCA combined with k-means to tackle the curse of dimensionality
- However, there are (efficient) implementations of partitioning clustering algorithms using smaller norm distance functions for example FasterPAM and CLARA
- We could explore even fractional norms to further heal from curse of dimensionality

Fast and eager -medoids clustering: runtime improvement of the PAM, CLARA, and CLARANS algorithms

Fast k-medoids Clustering in Rust and Python

Acknowledgments

Thank you!