ABSTRACTsß

In order to reduce the risk of a collision between engines and provide a more modern service experience both for our customers and traffic controllers, we are deploying a solution that addresses these items.

This paper presents the design and architecture of the proposed solution to track our engine location/telemetry in real-time, manage our asset inventory, and track cargo volume and profit.

**1. INTRODUCTION**

COMPONENTS

Kinesis Streams

Kinesis Firehose

S3

ElasticSearch

DynamoDB

Athena

Middleware

WorkSpaces

Traffic Controller Dashboard

**Kinesis Streams**

Kinesis streams is a flexible data streaming solution that would be capable of handling the desired rate of accuracy of the sensor data.

**Kinesis Firehose**

Kinesis Firehose is a fully managed service for delivering real-time streaming data to destinations such as Amazon Simple Storage Service (Amazon S3), Amazon Redshift, or Amazon Elasticsearch Service (Amazon ES).

**S3**

S3 is a cloud-based object store based on a partition/key-style hierarchy. We will use this to store sensor data, and exported order data for future analysis.

**ElasticSearch**

ElasticSearch is a robust data indexing and search service. We will be using this to house our asset inventory project.

**DynamoDB**

DynamoDB is a highly scalable, and consistent NoSQL engine. We will be utilizing DynamoDB to track aggregate telemetry from the sensor data.

**Athena**

Athena is a BI platform that can ingest data from S3. We will be processing BI to enlighten management to the status of cargo volume and profitability.

**Middleware**

In order to facilitate the cohesion between all of the service platforms, we will utilize our own middleware applications to manage the kinesis stream data between Kinesis Streams/Firehose, DynamoDB, and the Traffic Controller Dashboard.

**WorkSpaces**

In order to easily deploy, manage, and maintain our desktop deployments we will use WorkSpaces to host the Traffic Controller Dashboard.

**Traffic Controller Dashboard**

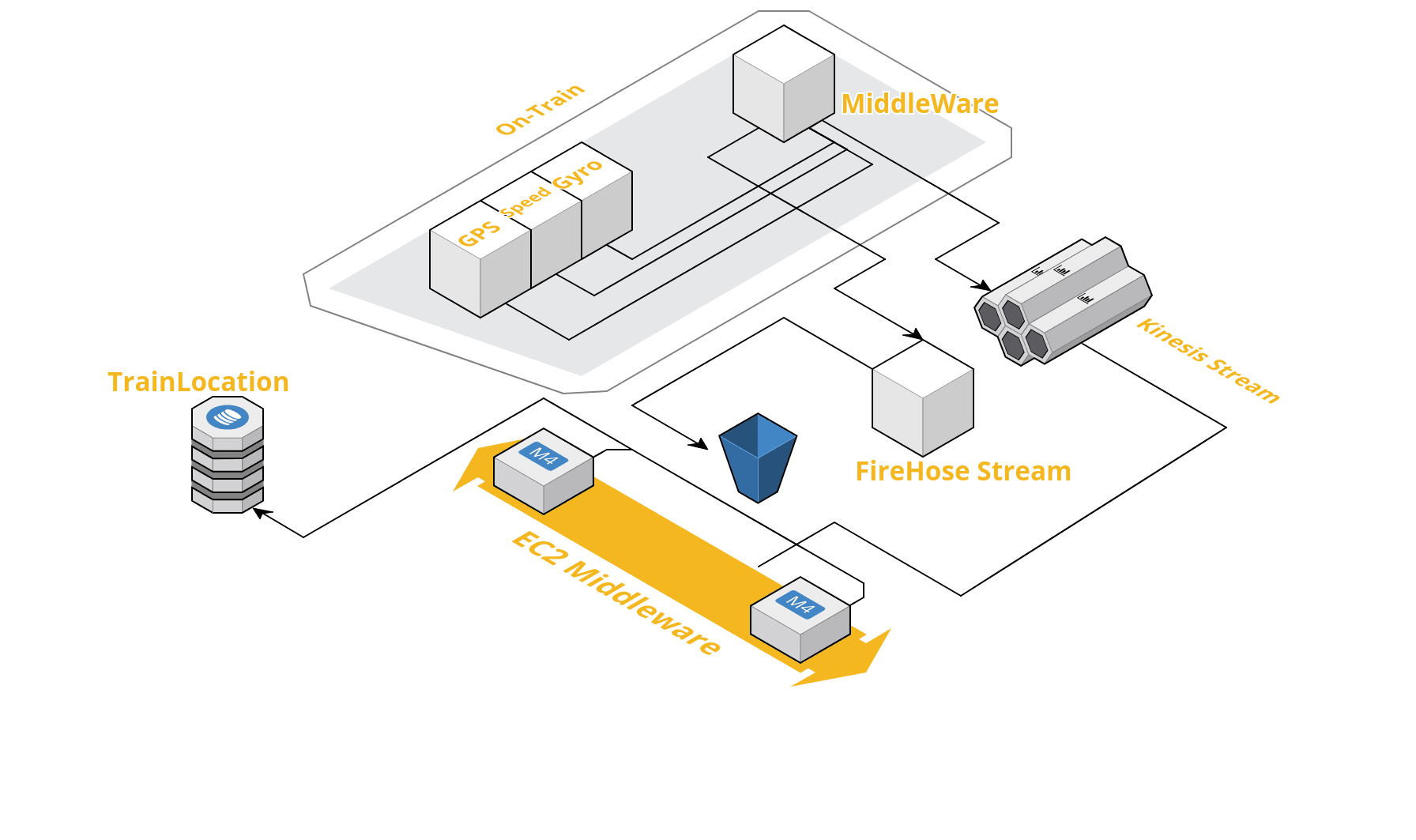
The Traffic Controller Dashboard will be a simple .NET application running on desktops deployed in the WorkSpaces environment.

**2. TECHNICAL.**

**Kinesis Streams**

Figure 1.1 outlines the topology of the Kinesis solution.

Fig 1.1



The on-train middleware component will sample sensor data every 2.5 seconds, and will aggregate two records into a put records API call every 5 seconds. The data will be split into shards based on TrainID. The GPS, Speed, and Gyro data will be concatenated into one string blob separated by pipes as a delimiter.

1. Put records[1]
2. Get shard iterator[2]
3. Get records[3]

The EC2-hosted middleware component will extract the sensor data, grab the shard iterator(LATEST) for each train and update the TrainLocation DDB table with the latest information. The application will continually grab the next shard iterator from the GetRecords requests.

**ElasticSearch**

The ElasticSearch cluster will be launched with the following configuration:

\*\*\*\*

Instance Count: 6

Instance Type: m4.large

Dedicated Master Instance Type: m4.large

Enable Dedicated Master: Yes

Dedicated Master Instance Count: 3

Enable Zone Awareness: Yes

Storage Type: EBS

EBS Volume Type: General Purpose (SSD)

EBS Volume Size: 50GB

Daily Snapshots at 00:00 local time which will be managed by Amazon.com

Daily Snapshots at 3:00 local time which will be managed by Choo Choo

\*\*\*\*

This configuration allows for a large search capacity. In the future, if we need more capacity we can simply restore a snapshot of the asset inventory to a new and larger cluster.

We will utilize zone awareness to increase the availability of the catalog. We are also taking snapshots on top of Amazon’s schedule so that we can manage our own snapshots.

In order to capture the data regarding these engines, I believe a data entry team would need to be established to generate the data.

***Schema***

For the mapping of the documents, I have outlined the following mappings for each example, reference included json files for each document and mapping example.

Engine Types: Engine\_Doc.json,, Engine\_Mapping.json

Carriage Types: Carriage\_Doc.json, Carriage\_Mapping.json

Rail Types: Rail\_Doc.json, Rail\_Mapping.json

**DynamoDB**

TableName: TrainLocation

HashKey: TrainID

Item Properties: Arriving location, departing location, current GPS, current speed, current heading.

Provisioned WCU: 300

Provisioned RCU: 300

I believe 300 WCU/RCU will be adequate for this application at project live date. One RCU represents one strongly consistent read per second, so that means we can read location data for 600 trains simultaneously. The trains will be grouped into two staggered batches so that only 300 trains check in at a given second.

There’s no need to backup this table up because all of the contained data is transient and can be replaced within seconds.

**Athena**

The Athena table will be created from order data stored on S3. Basic addition will be done for cargo and profit numbers.

**Middleware**

There will be two middleware components to the full solution. The On-Train middleware and the On-EC2 middleware.  
  
***On-Train***

There will be a small server running CentOS on the train. The server will host an application that:

1. Captures raw sensor data every 2.5 seconds. It will then aggregate two samples into each record for averaging by the Ec2 middleware.
2. In the event that internet is unavailable, the server will cache data locally up to 24 hours. Once connection resumes, cached records will only be placed into the Firehose stream. The latest available sensor data will begin to be pushed to Kinesis.
3. Host a text chat with train operator to be used in absence of other communication methods in an emergency.

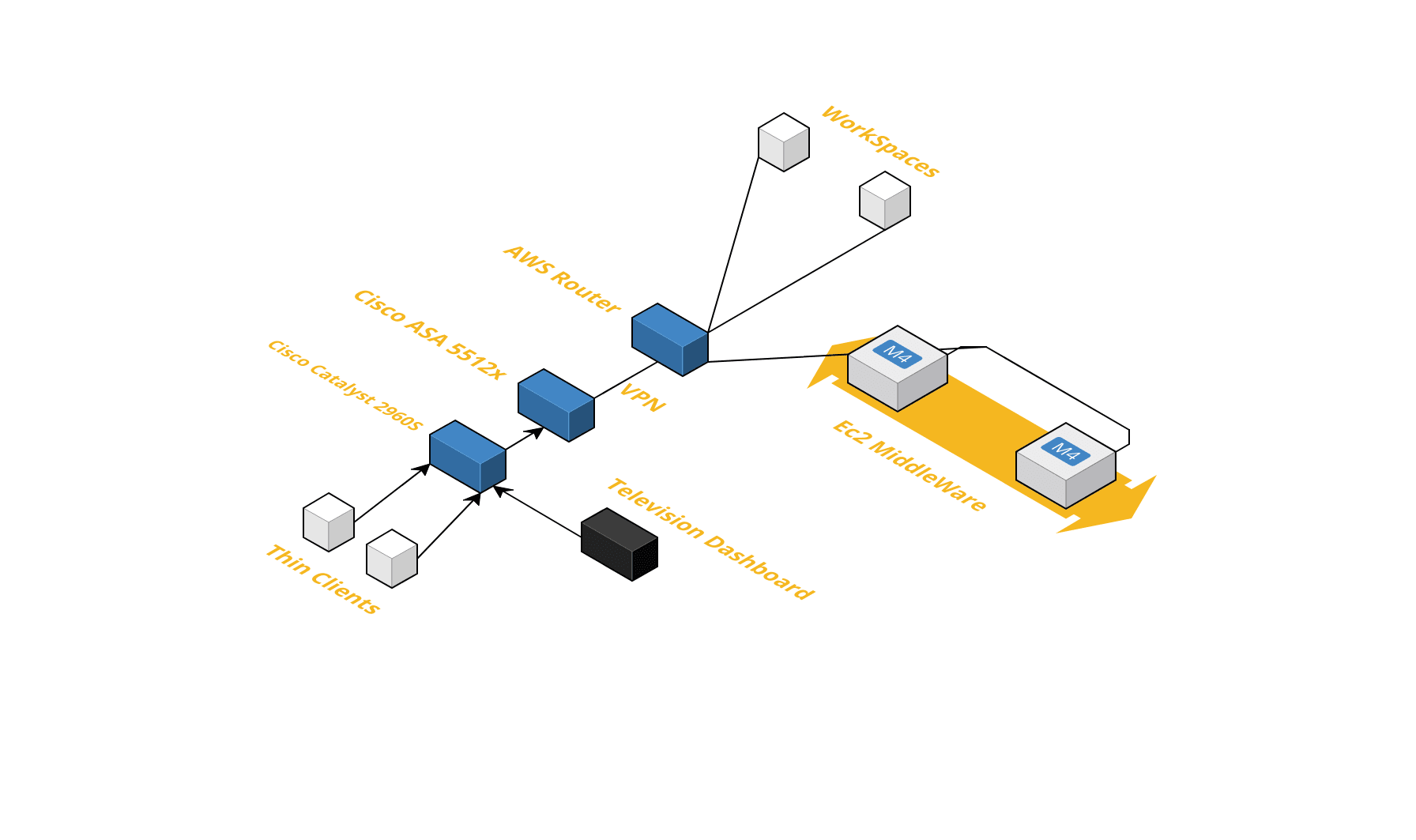
***On-EC2***

1. Process records from Kinesis stream. Average GPS/Speed/Gyro data between the two data points in each record.
2. Update the DynamoDB items with the latest properties from the Kinesis stream.
3. Query current database deployment for arriving and departing locations, then update DynamoDB table.
4. Provided SNS warnings when any issues are detecting within the solution.

**WorkSpaces**

The WorkSpaces configuration is pretty straight-forward. A performance WorkSpace will be provisioned for each user. The solution will connect to a Managed AD, configured with a trust to the on-premise domain. We will manage employee data backups via WorkDocs as well. Fig 2.1 shows the topology of the TCD dashboard.

Fig 2.1



**Traffic Controller Dashboard**

The traffic controller dashboard will be run on the WorkSpaces themselves. The .NET application will display a map, then pull data from DynamoDB to update the display of trains on the dashboard. There will also be a dashboard on the Tv.

[1]http://docs.aws.amazon.com/kinesis/latest/APIReference/API\_PutRecords.html

[2]http://docs.aws.amazon.com/kinesis/latest/APIReference/API\_GetShardIterator.html

[3]http://docs.aws.amazon.com/kinesis/latest/APIReference/API\_GetRecords.html