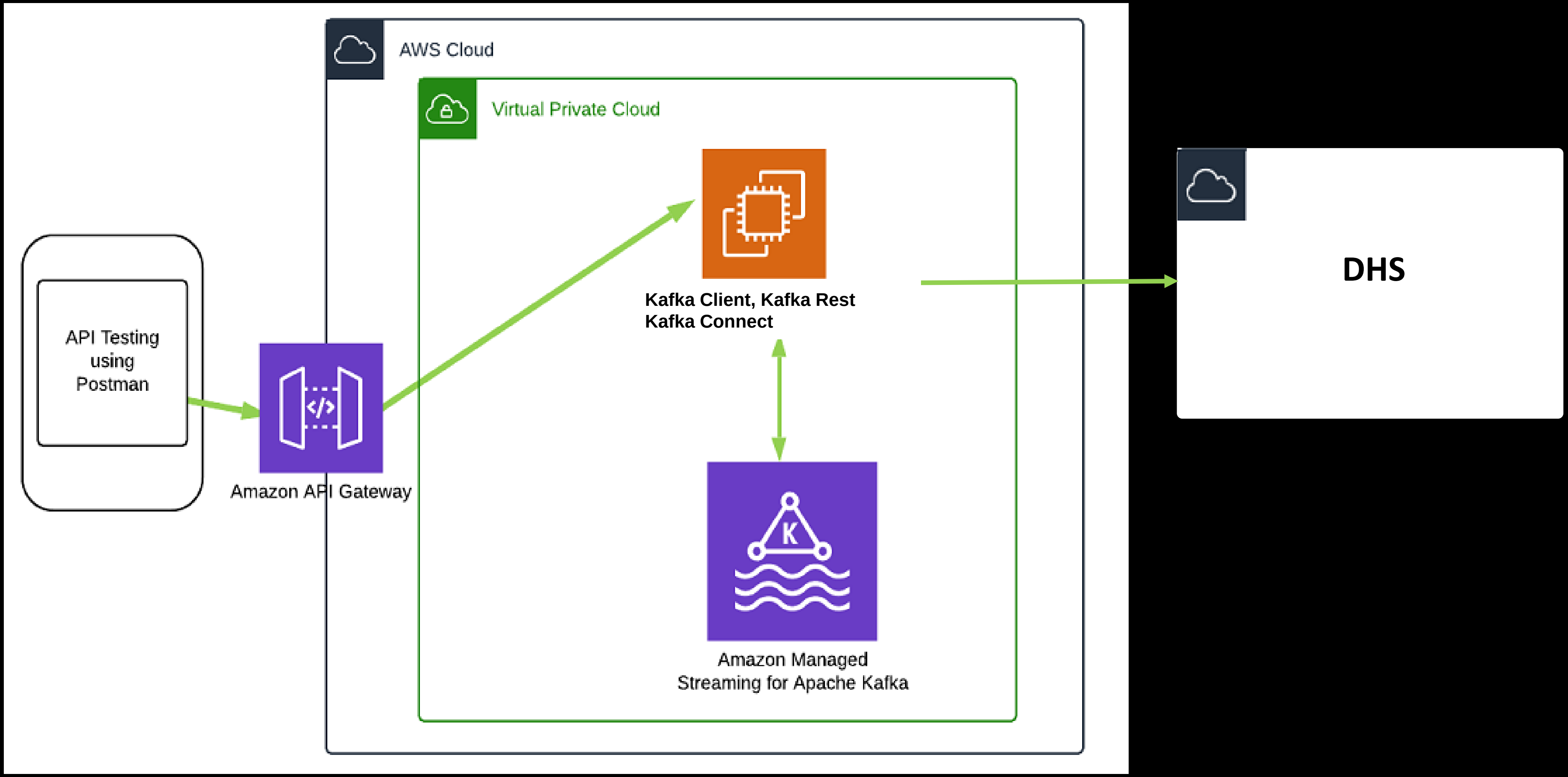
Setting Up MarkLogic Kafka Connector for AWS MSK

Objective: The target architecture



Step 0 – Pre-requisite (DHS & Connector)

- Provision a DHS environment.
  - For the purposes of this material a DHS with public endpoints is assumed.

Any reachable MarkLogic environment can be used. In this demonstration a DHS environment is assumed to be provisioned.

- Download or build the marklogic kafka connector jar from https://github.com/marklogic-community/kafka-marklogic-connector

- The subsequent steps assume knowledge in working with AWS services through AWS console and AWS CLI

Step 1 – Pre-requisite *(Infrastructure for MSK)*

1. Create a VPC
   a. https://docs.aws.amazon.com/msk/latest/developerguide/create-vpc.html
2. Enable HA and Fault Tolerance (more subnets in other AZs)
   a. https://docs.aws.amazon.com/msk/latest/developerguide/add-subnets.html

Step 2 – Create MSK Cluster

1. Create the MSK Cluster
    a. https://docs.aws.amazon.com/msk/latest/developerguide/create-cluster.html

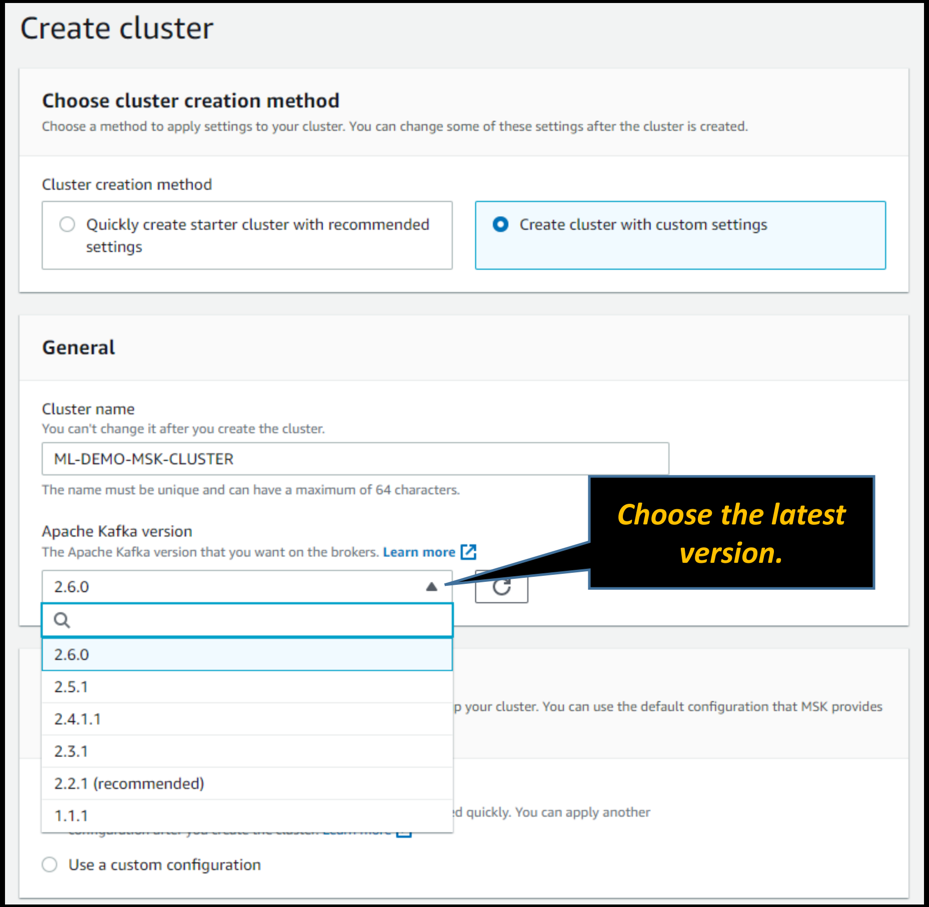
If you are just trying out the infrastructure and low side on cost, choose a smaller instance type, 2 broker nodes and TLS_PLAINTEXT for client Broker and smaller EBS volume . Check the following slides on a manual creation of MSK Cluster.

Note that MSK Cluster, once created will start incurring cost per hour. There is no real 'on-demand' pricing. The cost has two parts

- *For the MSK Cluster*
- *For the storage.*

Alternate Step 2 – Create MSK Cluster Manually

Step 2' – Create MSK Cluster (Manual Process)



Create cluster

Choose cluster creation method

Choose a method to apply settings to your cluster. You can change some of these settings after the cluster is created.

Cluster creation method

( ) Quickly create starter cluster with recommended settings [✓ Create cluster with custom settings]

General

Cluster name

You can't change it after you create the cluster.

ML-DEMO-MSK-CLUSTER

The name must be unique and can have a maximum of 64 characters.

Apache Kafka version

The Apache Kafka version that you want on the brokers. Learn more

2.6.0

- 2.6.0
- 2.5.1
- 2.4.1.1
- 2.3.1
- 2.2.1 (recommended)
- 1.1.1

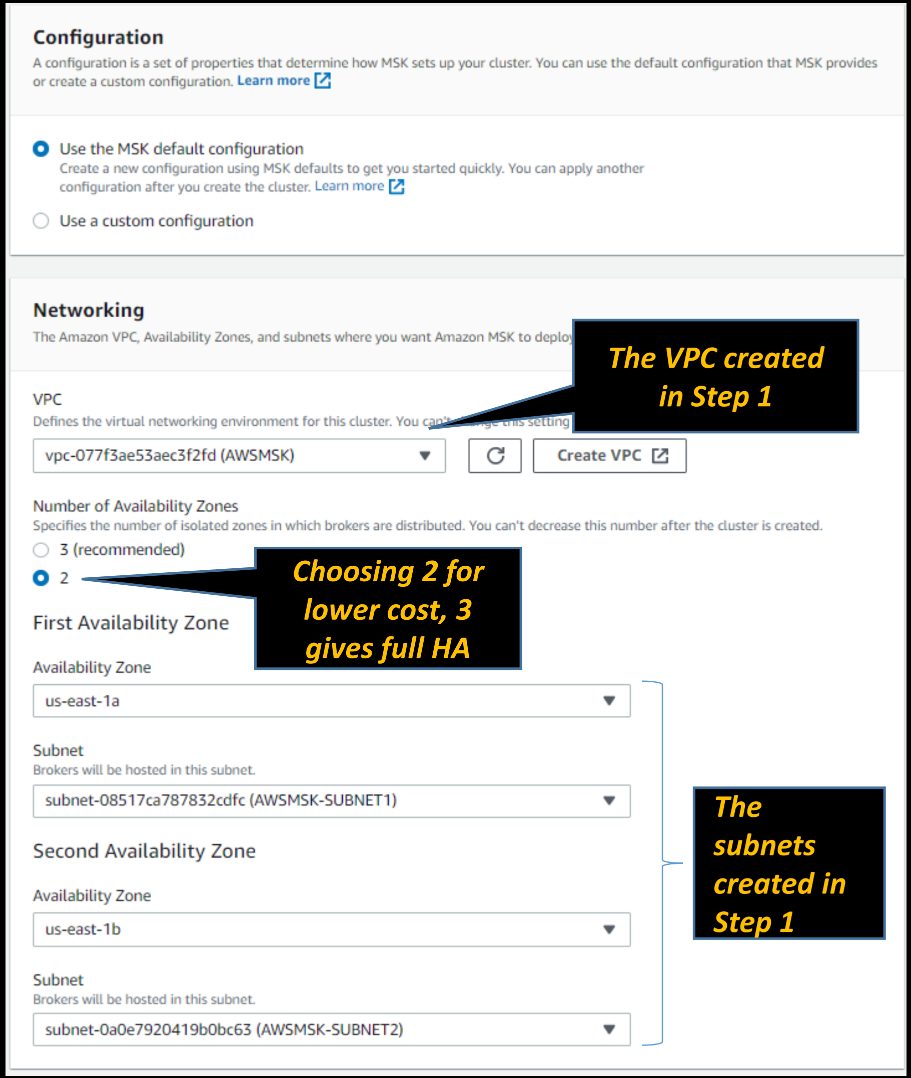
p your cluster. You can use the default configuration that MSK provides

d quickly. You can apply another

( ) Use a custom configuration

Choose the latest version.

It would be safer to choose the recommended kafka version option.



Configuration

A configuration is a set of properties that determine how MSK sets up your cluster. You can use the default configuration that MSK provides or create a custom configuration. Learn more

[✓ Use the MSK default configuration]
Create a new configuration using MSK defaults to get you started quickly. You can apply another configuration after you create the cluster. Learn more

( ) Use a custom configuration

Networking

The Amazon VPC, Availability Zones, and subnets where you want Amazon MSK to deploy

VPC

Defines the virtual networking environment for this cluster. You can't change this setting

vpc-077f3ae53aec3f2fd (AWSMSK)

Create VPC

Number of Availability Zones

Specifies the number of isolated zones in which brokers are distributed. You can't decrease this number after the cluster is created.

( ) 3 (recommended)
[✓ 2]

First Availability Zone

Availability Zone: us-east-1a

Subnet
Brokers will be hosted in this subnet.
subnet-08517ca787832cdfc (AWSMSK-SUBNET1)

Second Availability Zone

Availability Zone: us-east-1b

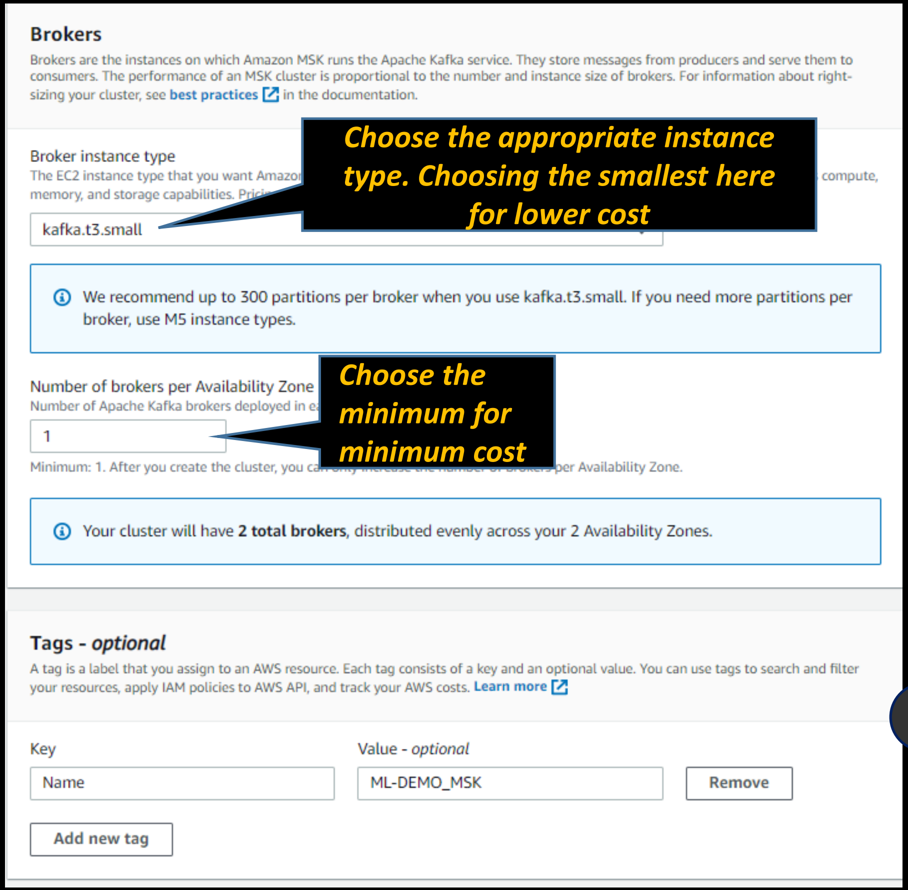
Subnet
Brokers will be hosted in this subnet.
subnet-0a0e7920419b0bc63 (AWSMSK-SUBNET2)

The VPC created in Step 1

Choosing 2 for lower cost, 3 gives full HA

The subnets created in Step 1

Step 2’ – Create MSK Cluster (Manual Process)



Brokers

Brokers are the instances on which Amazon MSK runs the Apache Kafka service. They store messages from producers and serve them to consumers. The performance of an MSK cluster is proportional to the number and instance size of brokers. For information about right-sizing your cluster, see best practices in the documentation.

Broker instance type

The EC2 instance type that you want Amazo … compute, memory, and storage capabilities. Prici

kafka.t3.small

Choose the appropriate instance type. Choosing the smallest here for lower cost

We recommend up to 300 partitions per broker when you use kafka.t3.small. If you need more partitions per broker, use M5 instance types.

Number of brokers per Availability Zone

Number of Apache Kafka brokers deployed in e

1

Choose the minimum for minimum cost

Minimum: 1. After you create the cluster, you ca … per Availability Zone.

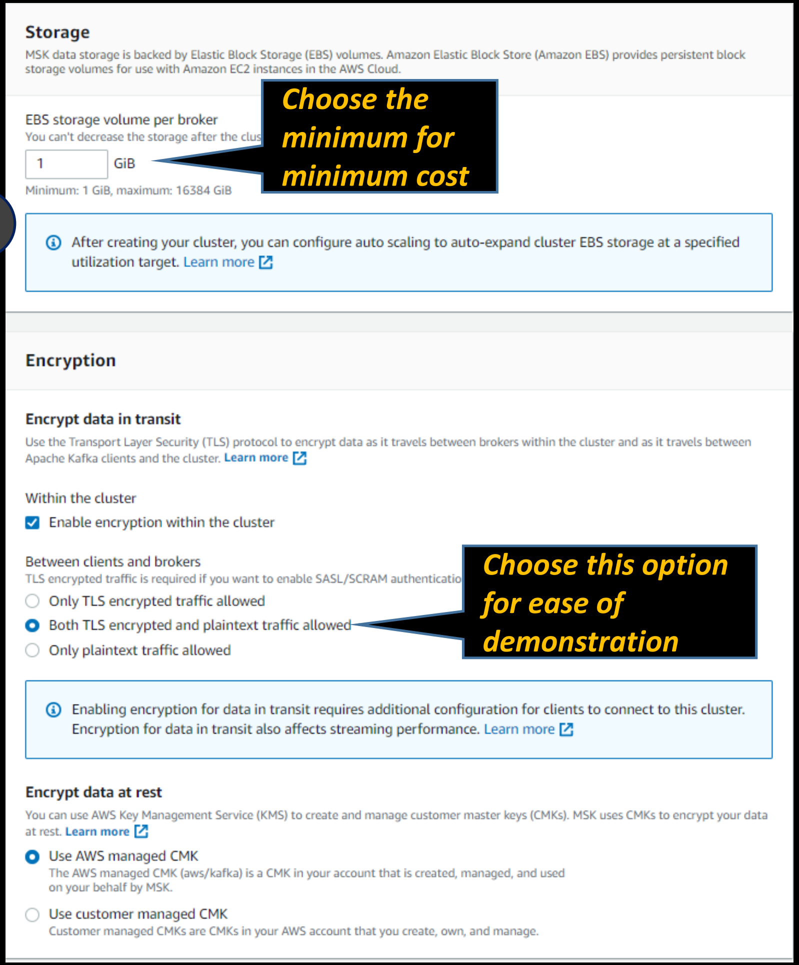
Your cluster will have **2 total brokers**, distributed evenly across your 2 Availability Zones.

Tags - *optional*

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources, apply IAM policies to AWS API, and track your AWS costs. Learn more

Key	Value - *optional*
Name	ML-DEMO_MSK

The configurations mentioned here is for a low-cost PoC. For production, use better instance type, brokers, storage and encryption



Storage

MSK data storage is backed by Elastic Block Storage (EBS) volumes. Amazon Elastic Block Store (Amazon EBS) provides persistent block storage volumes for use with Amazon EC2 instances in the AWS Cloud.

EBS storage volume per broker

You can't decrease the storage after the clus

1 GiB

Choose the minimum for minimum cost

Minimum: 1 GiB, maximum: 16384 GiB

After creating your cluster, you can configure auto scaling to auto-expand cluster EBS storage at a specified utilization target. Learn more

Encryption

Encrypt data in transit

Use the Transport Layer Security (TLS) protocol to encrypt data as it travels between brokers within the cluster and as it travels between Apache Kafka clients and the cluster. Learn more

Within the cluster

☑ Enable encryption within the cluster

Between clients and brokers

TLS encrypted traffic is required if you want to enable SASL/SCRAM authenticatio

○ Only TLS encrypted traffic allowed

● Both TLS encrypted and plaintext traffic allowed

○ Only plaintext traffic allowed

Choose this option for ease of demonstration

Enabling encryption for data in transit requires additional configuration for clients to connect to this cluster. Encryption for data in transit also affects streaming performance. Learn more

Encrypt data at rest

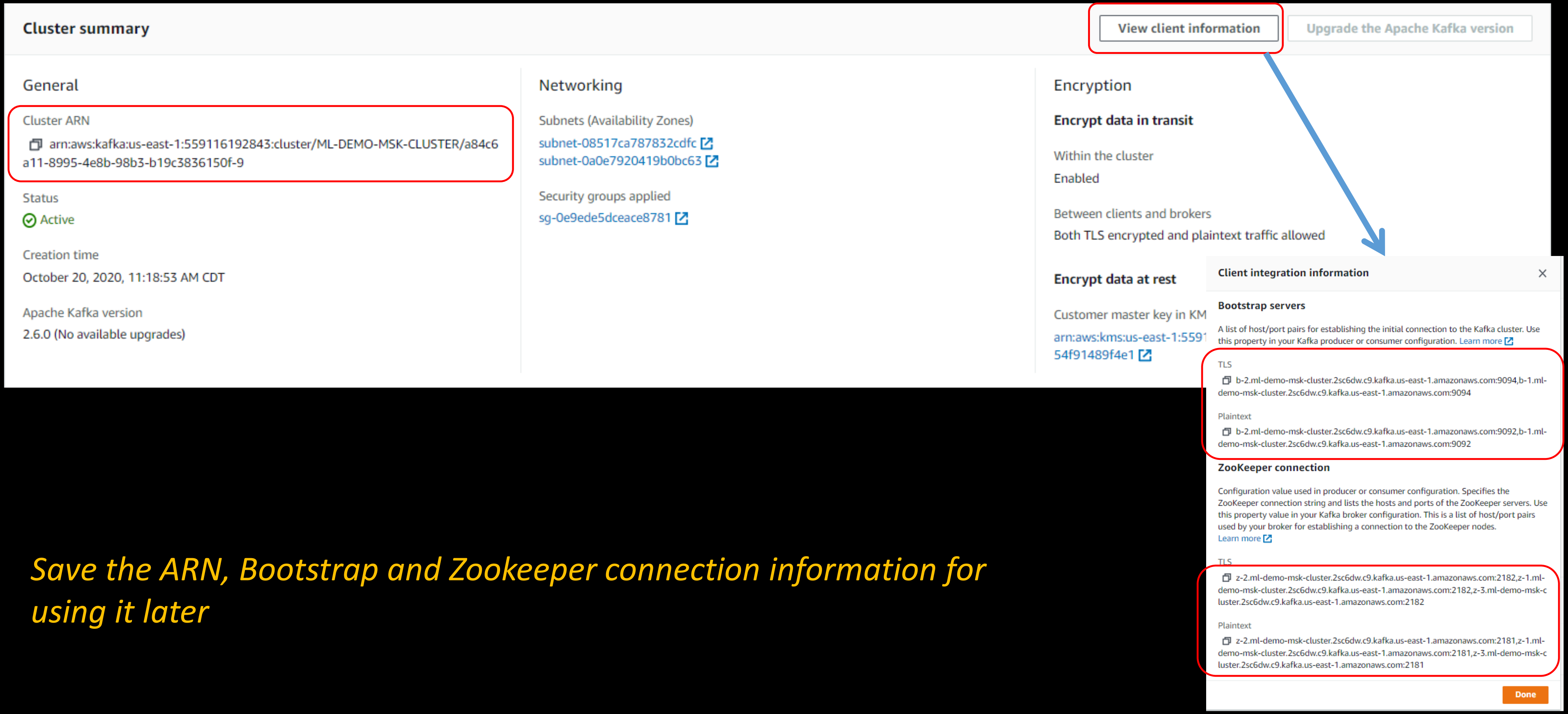
You can use AWS Key Management Service (KMS) to create and manage customer master keys (CMKs). MSK uses CMKs to encrypt your data at rest. Learn more

● Use AWS managed CMK
The AWS managed CMK (aws/kafka) is a CMK in your account that is created, managed, and used on your behalf by MSK.

○ Use customer managed CMK
Customer managed CMKs are CMKs in your AWS account that you create, own, and manage.

Leave all other attributes with default values

Step 2' – Create MSK Cluster (Manual Process)



Cluster summary

General

Cluster ARN

arn:aws:kafka:us-east-1:559116192843:cluster/ML-DEMO-MSK-CLUSTER/a84c6a11-8995-4e8b-98b3-b19c3836150f-9

Status

Active

Creation time

October 20, 2020, 11:18:53 AM CDT

Apache Kafka version

2.6.0 (No available upgrades)

Networking

Subnets (Availability Zones)

subnet-08517ca787832cdfc
subnet-0a0e7920419b0bc63

Security groups applied

sg-0e9ede5dceace8781

Encryption

Encrypt data in transit

Within the cluster

Enabled

Between clients and brokers

Both TLS encrypted and plaintext traffic allowed

Encrypt data at rest

Customer master key in KM

arn:aws:kms:us-east-1:5591
54f91489f4e1

Client integration information

Bootstrap servers

A list of host/port pairs for establishing the initial connection to the Kafka cluster. Use this property in your Kafka producer or consumer configuration. Learn more

TLS

b-2.ml-demo-msk-cluster.2sc6dw.c9.kafka.us-east-1.amazonaws.com:9094,b-1.ml-demo-msk-cluster.2sc6dw.c9.kafka.us-east-1.amazonaws.com:9094

Plaintext

b-2.ml-demo-msk-cluster.2sc6dw.c9.kafka.us-east-1.amazonaws.com:9092,b-1.ml-demo-msk-cluster.2sc6dw.c9.kafka.us-east-1.amazonaws.com:9092

ZooKeeper connection

Configuration value used in producer or consumer configuration. Specifies the ZooKeeper connection string and lists the hosts and ports of the ZooKeeper servers. Use this property value in your Kafka broker configuration. This is a list of host/port pairs used by your broker for establishing a connection to the ZooKeeper nodes. Learn more

TLS

z-2.ml-demo-msk-cluster.2sc6dw.c9.kafka.us-east-1.amazonaws.com:2182,z-1.ml-demo-msk-cluster.2sc6dw.c9.kafka.us-east-1.amazonaws.com:2182,z-3.ml-demo-msk-cluster.2sc6dw.c9.kafka.us-east-1.amazonaws.com:2182

Plaintext

z-2.ml-demo-msk-cluster.2sc6dw.c9.kafka.us-east-1.amazonaws.com:2181,z-1.ml-demo-msk-cluster.2sc6dw.c9.kafka.us-east-1.amazonaws.com:2181,z-3.ml-demo-msk-cluster.2sc6dw.c9.kafka.us-east-1.amazonaws.com:2181

Save the ARN, Bootstrap and Zookeeper connection information for using it later

Step 3 – Create Client Machine

- Why a client machine is needed ?

A client machine is required to create topics, to run the connectors or rest proxy. The managed service MSK does not support managed connectors. For this purpose, a EC2 instance (in same VPC as MSK or a different VPC) needs to be created and install Apache Kafka or Confluent Kafka. In this presentation, using both are demonstrated.

Follow the below link to create the client machine.

https://docs.aws.amazon.com/msk/latest/developerguide/create-client-machine.html

<I created a Elastic IP and associated to the EC2 instance so that I get a permanent IP address / DNS name>

Steps 4-6: Setting up Apache Kafka and MarkLogic connector

Step 4 – Install Java and Kafka in client machine

- https://docs.aws.amazon.com/msk/latest/developerguide/create-topic.html
- Ensure that the kafka version installed is the version that was chosen while creating MSK
- Create topics *perf-queue* and *test-dlq* using command

```
bin/kafka-topics.sh --create --zookeeper <zookeeper
connection string> --replication-factor 2 --partitions 1 -
-topic test-dlq
```

```
bin/kafka-topics.sh --create --zookeeper <zookeeper
connection string> --replication-factor 2 --partitions 1 -
-topic perf-queue
```

Step 5 – Set up the MarkLogic connector

1. Copy the connector jar to `libs` directory under kafka installation.
2. Copy the `marklogic-connect-standalone.properties` and `marklogic-sink.properties` to the `config` directory under kafka installation.
3. Update below properties in `marklogic-connect-standalone.properties` with the broker information from MSK cluster.

   `consumer.bootstrap.servers & bootstrap.servers`

   Use the PLAINTEXT connection string if SSL is not required. In production TLS might be required.
4. Update `plugin.path` in marklogic-connect-standalone.properties with the `libs` directory.

   Example: `plugin.path=/home/ec2-user/kafka_2.12-2.6.0/libs`
5. Update the `marklogic-sink.properties` with the DHS information
6. Update the `key.converter` and `value.converter` with the appropriate converter classes

   Example:

   ```
   key.converter=org.apache.kafka.connect.json.JsonConverter
   value.converter=org.apache.kafka.connect.json.JsonConverter
   ```

Step 6: Validate the connector

- Start the connector

  ```
  bin/connect-standalone.sh config/marklogic-connect-
  standalone.properties config/marklogic-sink.properties >
  /dev/null 2>&1 &
  ```

  <In Production the connector has to be started as a service and logs redirected to a log file>

- Produce some messages

  ```
  bin/kafka-console-producer.sh --broker-list <broker
  connection string> --topic perf-queue
  ```

- Check that the messages are inserted in DHS (MarkLogic) according to the configuration in the sink properties file.

Alternate Steps 4-6: Setting up Confluent Kafka and MarkLogic connector

Step 5’ – Setup Confluent Kafka

Why Confluent Kafka ?

- Confluent Kafka has connect, control-center, kafka-rest modules for connector and managing MSK through REST APIs.
- Additionally, confluent kafka has built-in support for AVRO, PROTOBUF, JSON WITH SCHEMA messages.

Beginning with Confluent Platform 6.0, connectors are no longer packaged natively with Confluent Platform

Step 5’ – Setup Confluent Kafka and connector

1. Copy the connector jar to `share/java/kafka-connect-marklogic` directory under confluent installation.
2. Copy the `marklogic-connect-standalone.properties` and `marklogic-sink.properties` to the `etc` directory under confluent installation.
3. Update below properties in `marklogic-connect-standalone.properties` with the broker information from MSK cluster.

   ```
   consumer.bootstrap.servers, bootstrap.servers
   ```

   Use the PLAINTEXT connection string if SSL is not required. In production TLS might be required.
4. Update `plugin.path` in `marklogic-connect-standalone.properties` with the `libs` directory.

   Example: `plugin.path=/home/ec2-user/confluent-6.0.0/share/java/kafka-connect-marklogic/`
5. Update the `marklogic-sink.properties` with the DHS information
6. Update the `key.converter` and `value.converter` with the appropriate converter classes

   Example:

   ```
   key.converter=org.apache.kafka.connect.json.JsonConverter
   value.converter=org.apache.kafka.connect.json.JsonConverter
   ```

Step 6’: Validate the connector

- Start the connector

```
bin/connect-standalone etc/marklogic-connect-
standalone.properties etc/marklogic-sink.properties >
/dev/null 2>&1 &
```

  <In Production the connector has to be started as a service and logs redirected to a log file>

- Produce some messages

```
bin/kafka-console-producer.sh --broker-list <broker
connection string> --topic perf-queue
```

- Check that the messages are inserted in DHS (MarkLogic) according to the configuration in the sink properties file.

Step 7 : Set up confluent kafka-rest

- Update `etc/kafka-rest/kafka-rest.properties` for below values

  ```
  zookeeper.connect=<The PLAINTEXT zookeeper connection string of MSK>
  bootstrap.servers=<The PLAINTEXT bootstrap connection string of MSK>
  listeners=http://0.0.0.0:8082,https://0.0.0.0:8085
  ```

  In production configuration, you might be using the TLS connection strings. In that case you need to configure `security.protocol` and `ssl.truststore.location`, `ssl.truststore.password` properties.Please refer the confluent documentation.

- Start the kafka-rest server as below

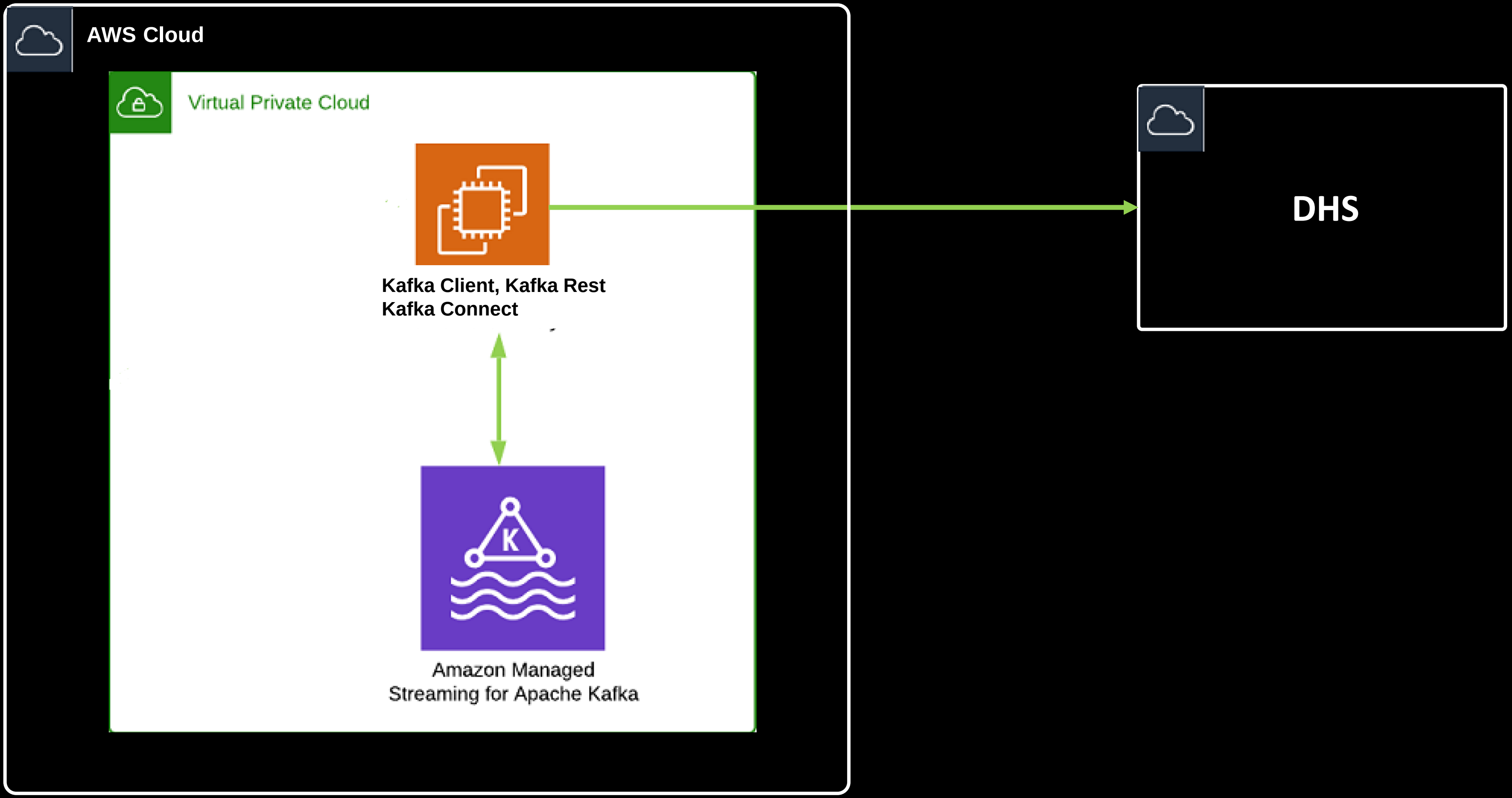
  ```
  bin/kafka-rest-start etc/kafka-rest/kafka-rest.properties > /dev/null
  2>&1 &
  ```

- Produce a message using a API client like curl, Postman or SoapUI

  ```
  curl -v -X POST -i -H "Content-Type: application/vnd.kafka.json.v2+json"
  -H "Accept: application/vnd.kafka.v2+json, application/vnd.kafka+json,
  application/json" --data '{"records": [{"key": "somekey","value":
  {"foo": "bar"}}]}' http://<publicDNSof the EC2
  instance>:8082/topics/perf-queue
  ```

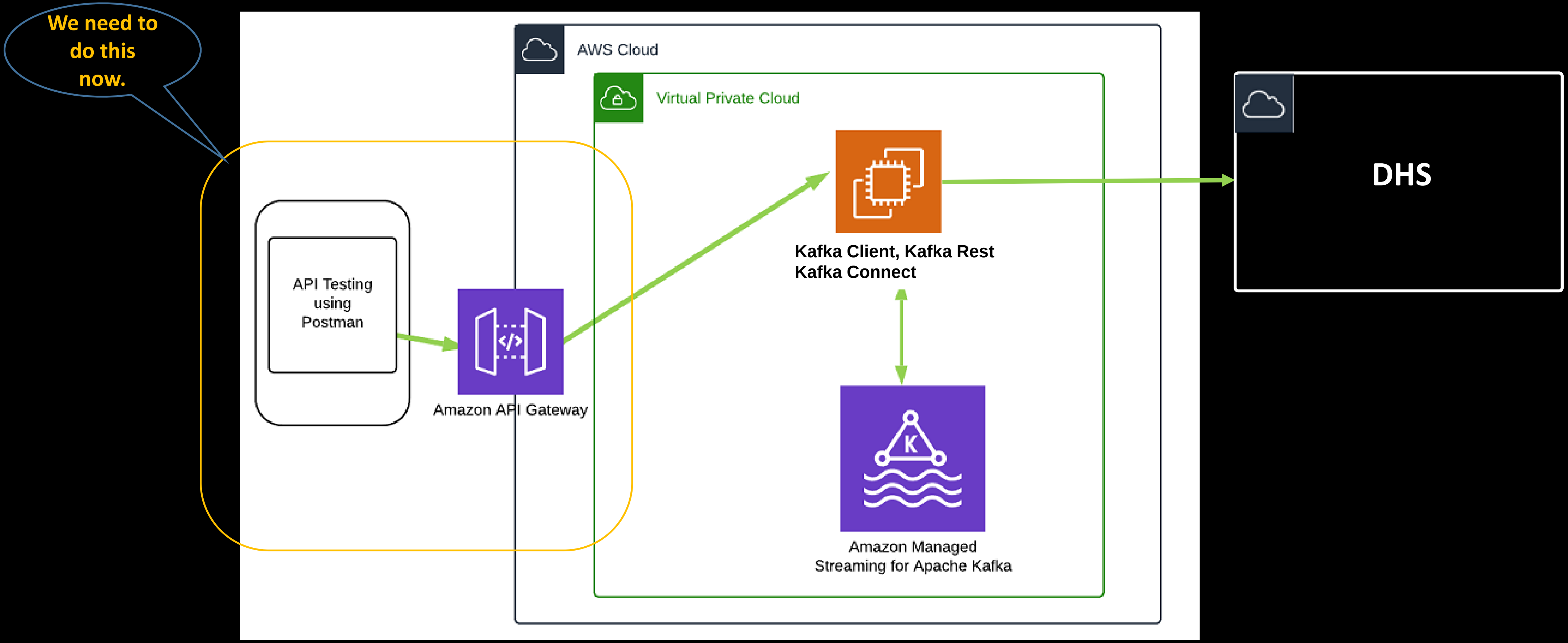
- Check the availability of the message in DHS
  - `Note that the connector is already running when you are in this step`

Reviewing the configuration till now



The above configuration can further improved by having an API gateway or proxy server.

The target architecture with API Gateway


We need to do this now.
AWS Cloud
Virtual Private Cloud
API Testing using Postman
Amazon API Gateway
Kafka Client, Kafka Rest Kafka Connect
Amazon Managed Streaming for Apache Kafka
DHS

Deploy an API in the AWS API Gateway

1. On the API Gateway console, choose **Create API**.
2. For API type, choose **REST API**.
3. Choose **Build**.
4. Choose **New API**.
5. For API Name, enter a name (for example, amazonmsk-restapi).
6. As an optional step, for Description, enter a brief description.
7. Choose **Create API**. The next step is to create a child resource.
8. Under **Resources**, choose a parent resource item.
9. Under **Actions**, choose **Create Resource**. The **New Child Resource** pane opens.
10. Select **Configure** as proxy resource.
11. For **Resource Name**, enter `proxy`.
12. For **Resource Path**, enter `/{proxy+}`.
13. Select **Enable API Gateway CORS**.
14. Choose **Create Resource**. After you create the resource, the Create Method window opens.
15. For **Integration type**, select **HTTP Proxy**.
16. For **Endpoint URL**, enter an HTTP backend resource URL. For example, `http://<KafkaClientDNS>:8082/{proxy}`
17. Use the default settings for the remaining fields.
18. Choose **Save**.
19. Choose the API you just created.
20. Under **Actions**, choose **Deploy API**.
21. For **Deployment** stage, choose **New Stage**.
22. For Stage name, enter the stage name (for example, dev, test, or prod).
23. Choose **Deploy**.
24. Record the **Invoke URL** after you have deployed the API.
25. Your external Kafka REST Proxy, which was exposed through API Gateway, now looks like `https://YourAPIGWInvoleURL/<stage>/topics/perf-queue` You use this URL in testing.

Additionally, make sure that the Kafka Client EC2 is having TCP ports opened so that we can test from the API testing machine.

Final Testing

- Ensure that kafka-rest and the connector is running
- From a API testing tool like SoapUI or Postman perform the below testing

URL `POST https://<invokeURLfromAPIGateway>/<stage>/topics/perf-queue`

Headers:
```
Accept: application/vnd.kafka.v2+json, application/vnd.kafka+json, application/json
      Content-Type: application/vnd.kafka.json.v2+json
```

Body:
```
{
     "records": [
       {
         "value": {
         "Customer": {
             "id" : "C001",
             "name" : "Jon"
         }
         }
       }
     ]
   }
```

- Check in DHS for the availability of the message.
- AVRO, PROTOBUF and JSON WITH SCHEMA messages can be validated by making appropriate changes to the connector properties. Pls refer the connector documentation for configuring the connector for these messages. You will need to start the schema-registry service also for handling messages with schema.

Chargeable components the infrastructure

Below are the minimum AWS components that will be charged for the infrastructure used in this presentation.

Managed Streaming Service for Apache Kafka (MSK)

- Amazon Managed Streaming for Apache Kafka RunBroker
- Amazon Managed Streaming for Apache Kafka RunVolume

Elastic Compute Cloud

- Amazon Elastic Compute Cloud running Linux/UNIX
- EBS
- Elastic IP Addresses (Optional)

API Gateway

- Amazon API Gateway ApiGatewayRequest

Data Transfer

- *Out Bytes

DHS

- MarkLogic Data Hub Service (DHS)

There will be more chargeable components when a production configuration is set up. The above are components charged for the PoC.