**Prerequisites**

**Read this** [**https://aws.amazon.com/premiumsupport/knowledge-center/kinesis-kafka-connector-msk/**](https://aws.amazon.com/premiumsupport/knowledge-center/kinesis-kafka-connector-msk/)

Create vpc endpoints for glue and firehose in the same subnets as the msk cluster and msk connector.

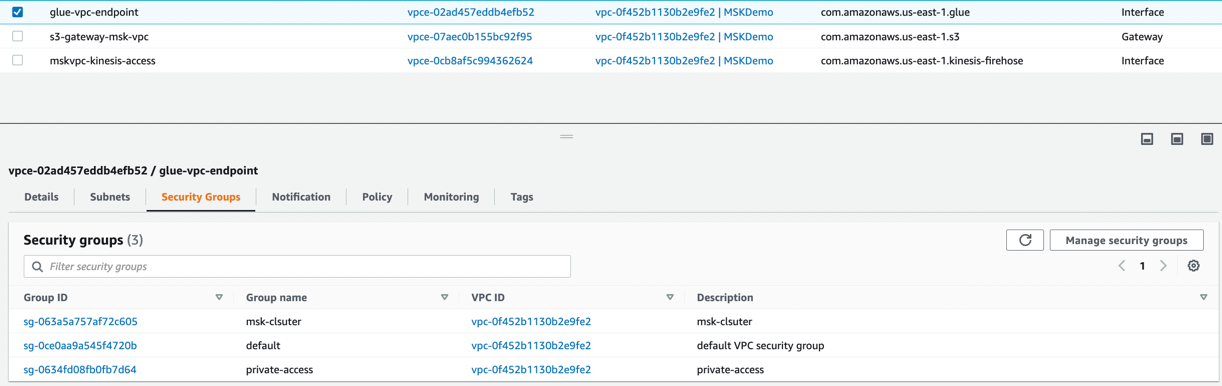
1-Setup the MSK Cluster.

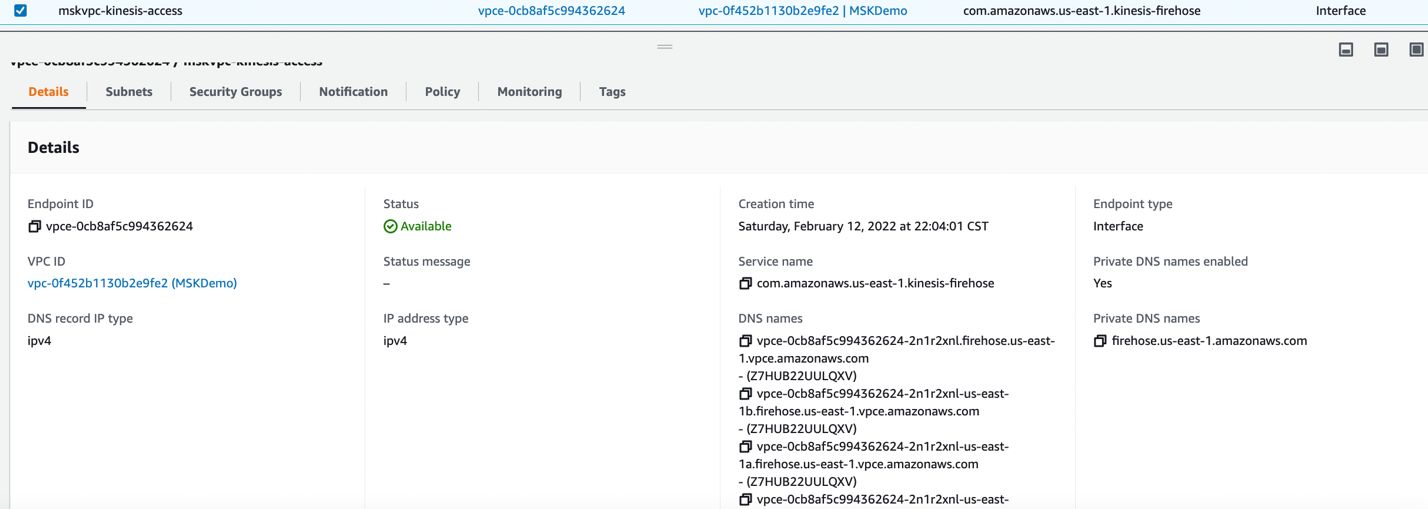
2-Deploy lambda function first.

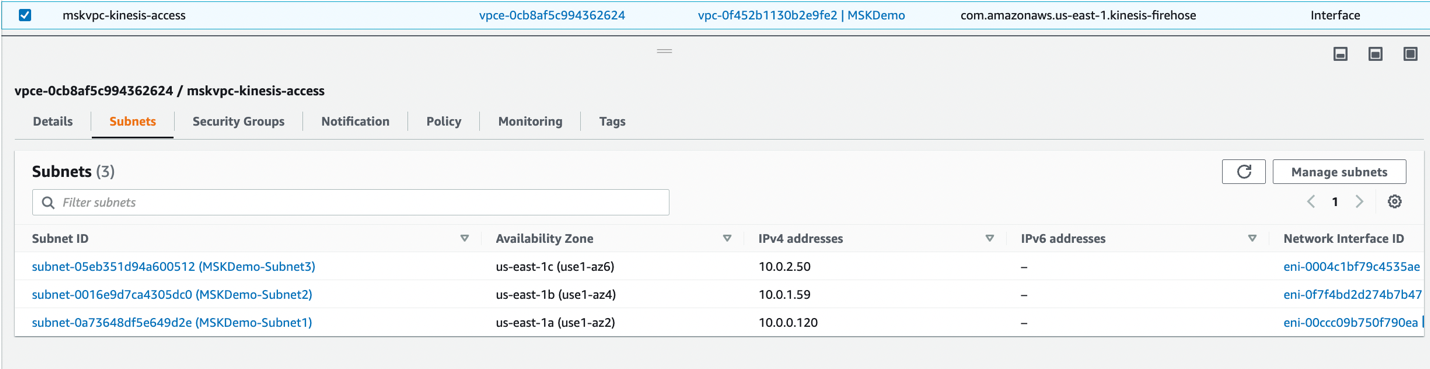
3-Setup the Firehose Delivery stream with the lambda

4-Deploy the custom plugin.

5-Deploy the MSK Connector based off the custom plugin







**MSK Cluster Setup**

MSK is setup in public subnet.

Producer and consumer (java) setup on local and connectivity with AWS VPC setup via AWS Managed VPN and Tunnelblick software.

Code Repo link -> git clone <https://PiyaliPersonal@bitbucket.org/microservices_poc_cd/msk_producer_consumer_avro.git>

**MSK connector configuration**

MSK connector setup in public subnet so that it can connect to the advertised listeners of MSK which is always a private address.

MSK Connecter connects to glue schema registry as well for registering avro schema.

MSK Connector code : - git clone <https://PiyaliPersonal@bitbucket.org/microservices_poc_cd/msk-firehose-connector-avro.git>

**MSK connector configuration setup**

connector.class=com.amazon.kinesis.kafka.FirehoseSinkConnector

value.converter.schemaAutoRegistrationEnabled=true

key.converter.endpoint=**https://vpce-02ad457eddb4efb52-1tjj7rpe.glue.us-east-1.vpce.amazonaws.com**

tasks.max=1

key.converter.region=us-east-1

value.converter.endpoint=**https://vpce-02ad457eddb4efb52-1tjj7rpe.glue.us-east-1.vpce.amazonaws.com**

key.converter.schemaName=**KAFKA-GLUEREGISTRY-SAMPLE-TOPIC**

value.converter.timeToLiveMillis=3600000

value.converter.avroRecordType=GENERIC\_RECORD

key.converter.timeToLiveMillis=3600000

value.converter.schemaName=**KAFKA-GLUEREGISTRY-SAMPLE-TOPIC**

batchSizeInBytes=3670016

value.converter.compressionType=NONE

kinesisEndpoint=**https://vpce-0cb8af5c994362624-2n1r2xnl.firehose.us-east-1.vpce.amazonaws.com**

key.converter.cacheSize=100

key.converter.avroRecordType=GENERIC\_RECORD

value.converter=**com.amazonaws.services.schemaregistry.kafkaconnect.AWSKafkaAvroConverter**

key.converter=**com.amazonaws.services.schemaregistry.kafkaconnect.AWSKafkaAvroConverter**

value.converter.cacheSize=100

topics=**KAFKA-GLUEREGISTRY-SAMPLE-TOPIC**

batch=false

value.converter.registry.name=**default-registry**

deliveryStream=**PUT-AVRO-DYNAMIC-PQT**

value.converter.region=us-east-1

key.converter.registry.name=default-registry

**key.converter.schemaAutoRegistrationEnabled=true**

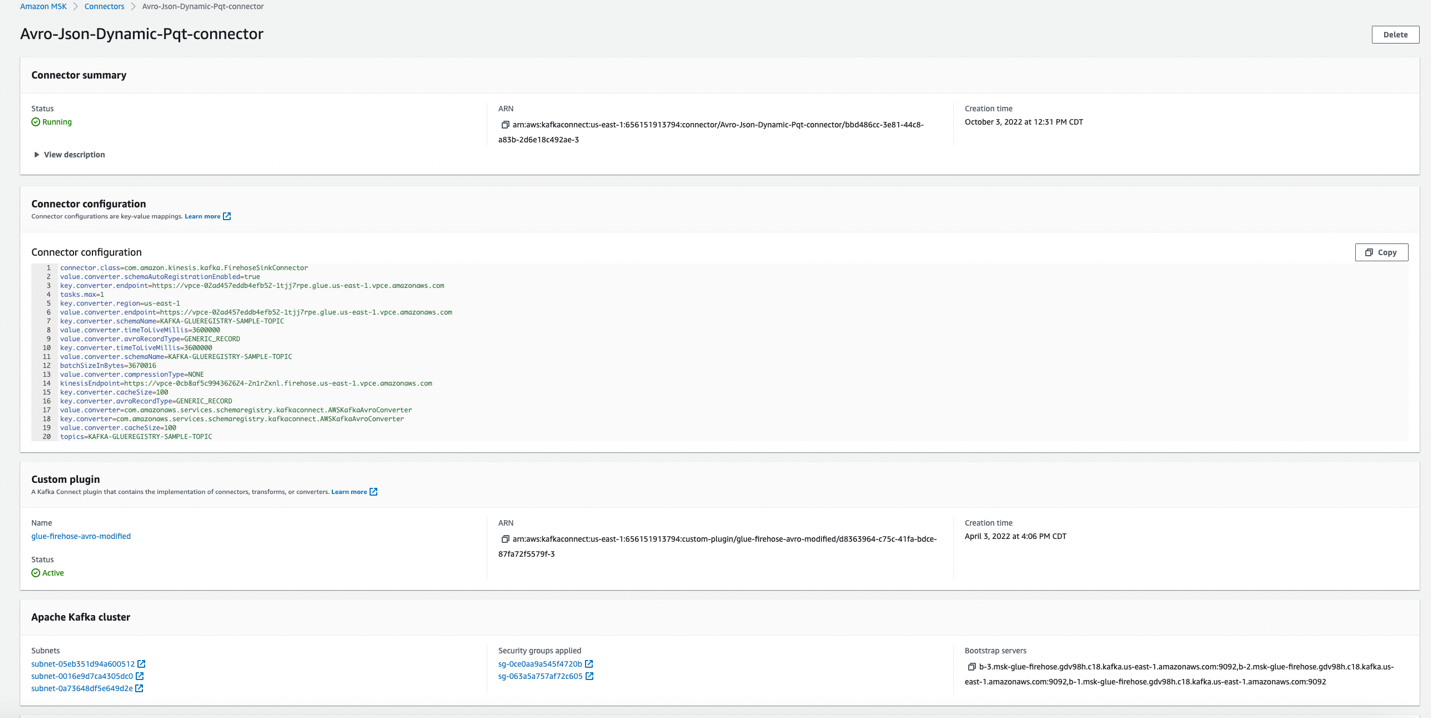
key.converter.compressionType=NONE

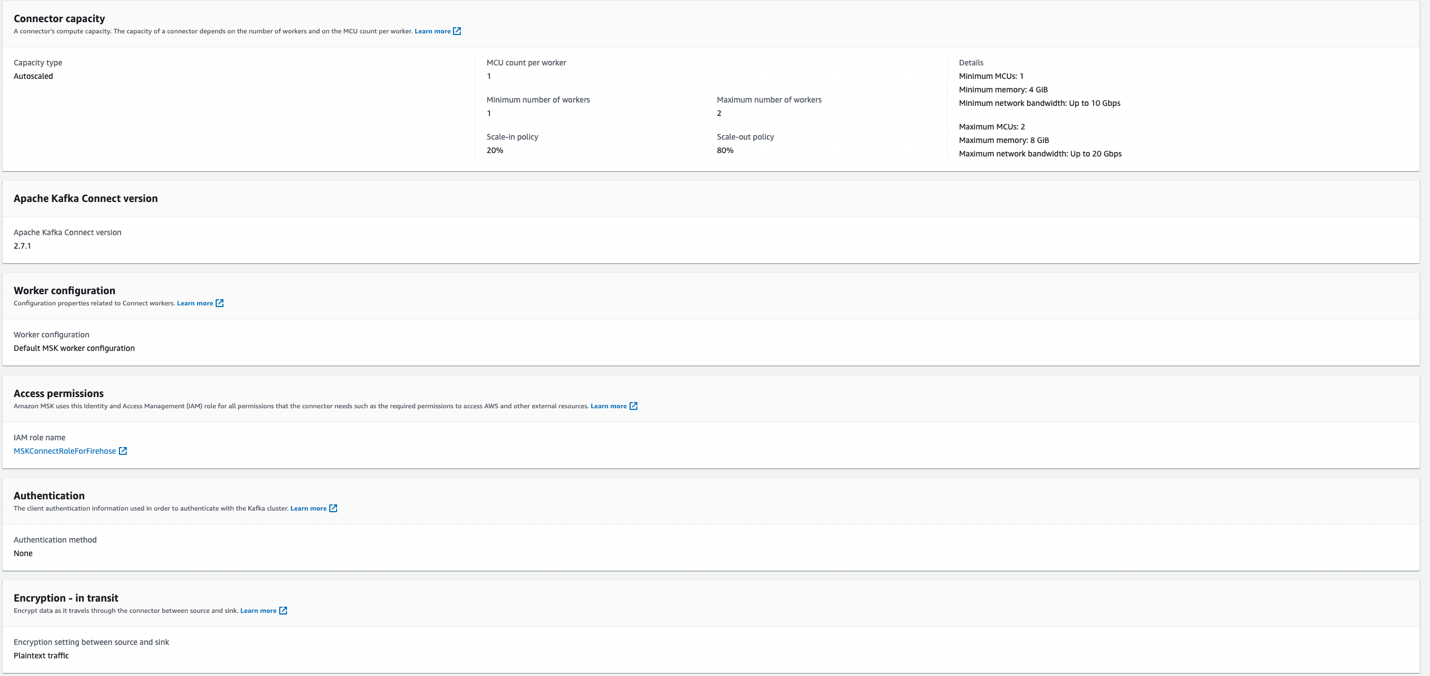
region=us-east-1

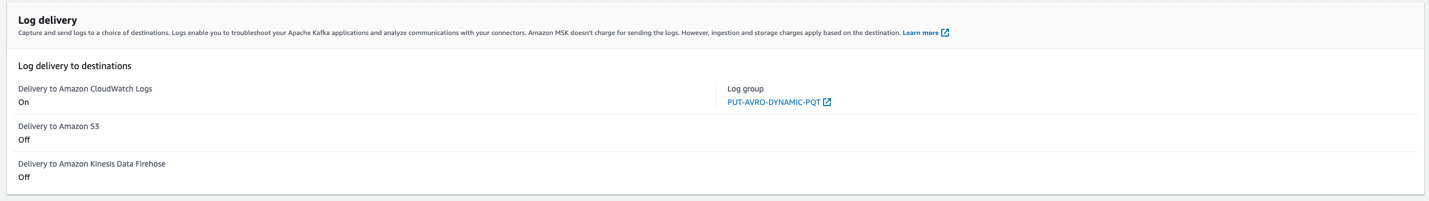
batchSize=1

**How to deploy connector code** -> create java jar file by mvn clean package . Upload jar file in s3 bucket.

Connector Screenshots – Setup autoscaling, plaintext,loggroup.



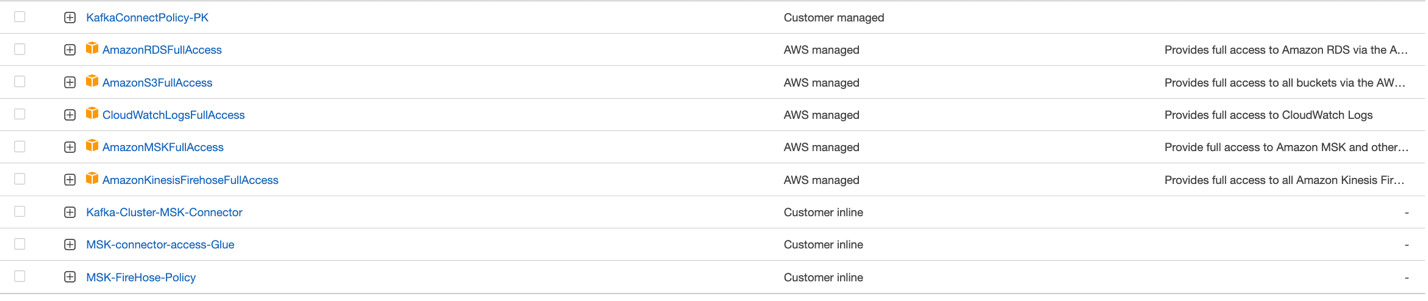




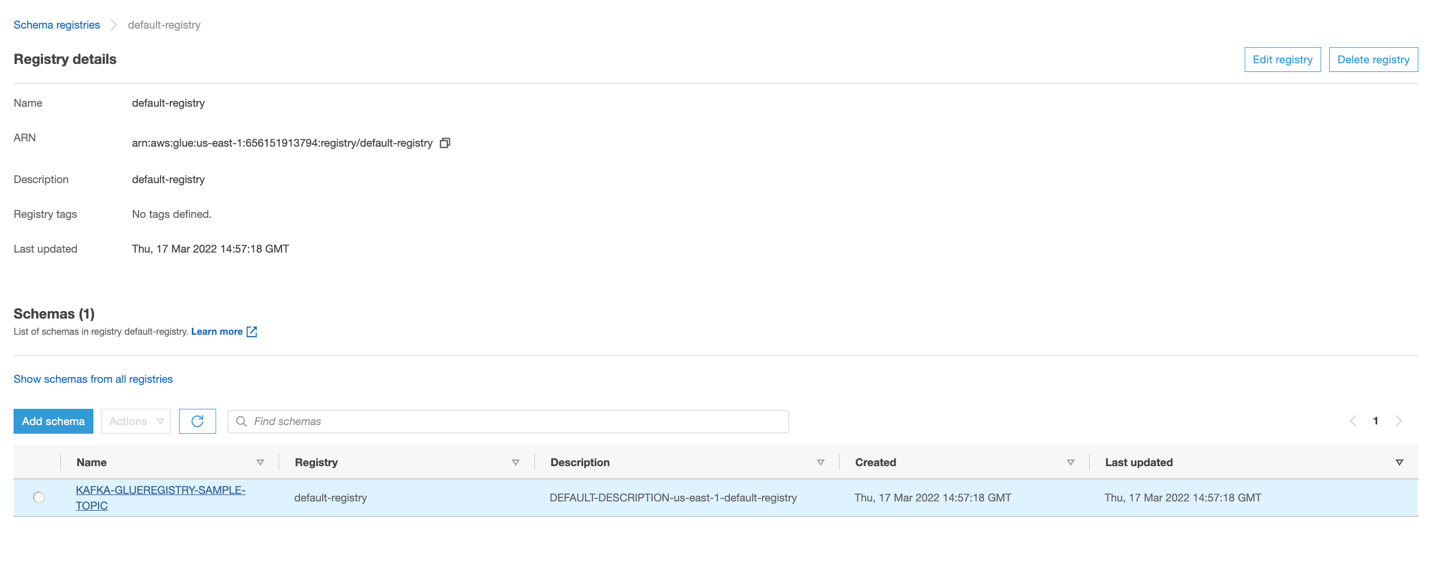
IAM role for MSK Connector

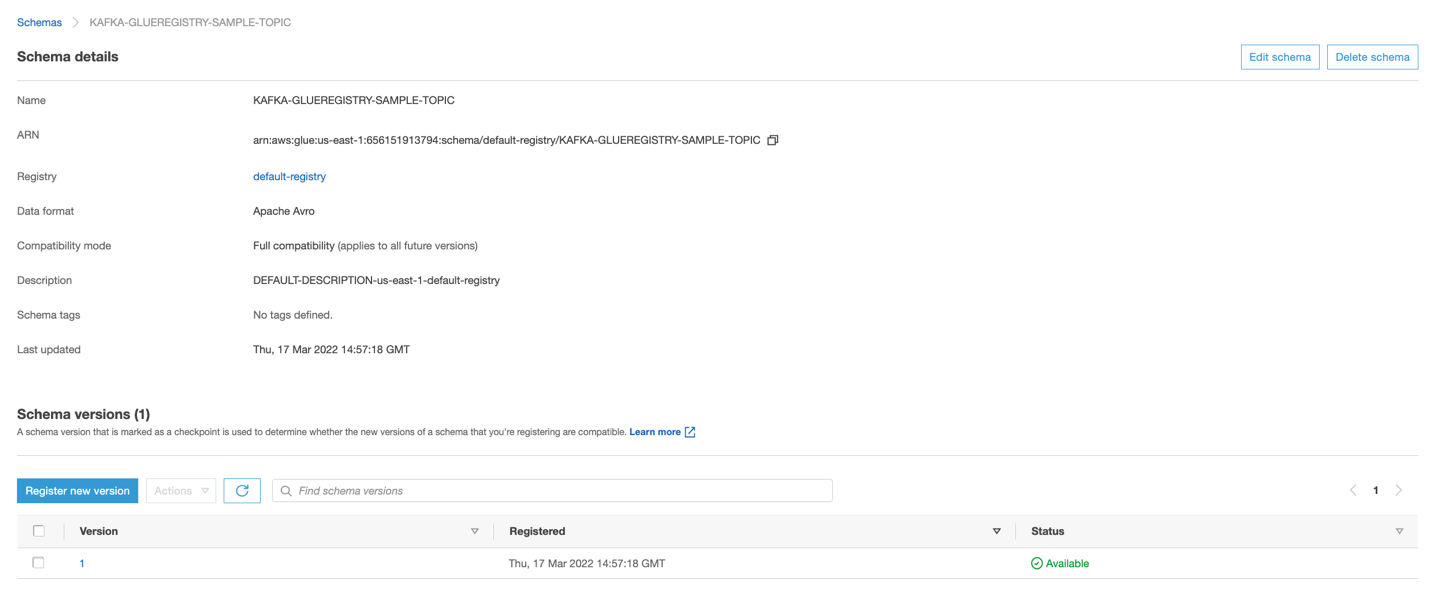
<https://aws.amazon.com/premiumsupport/knowledge-center/kinesis-kafka-connector-msk/>

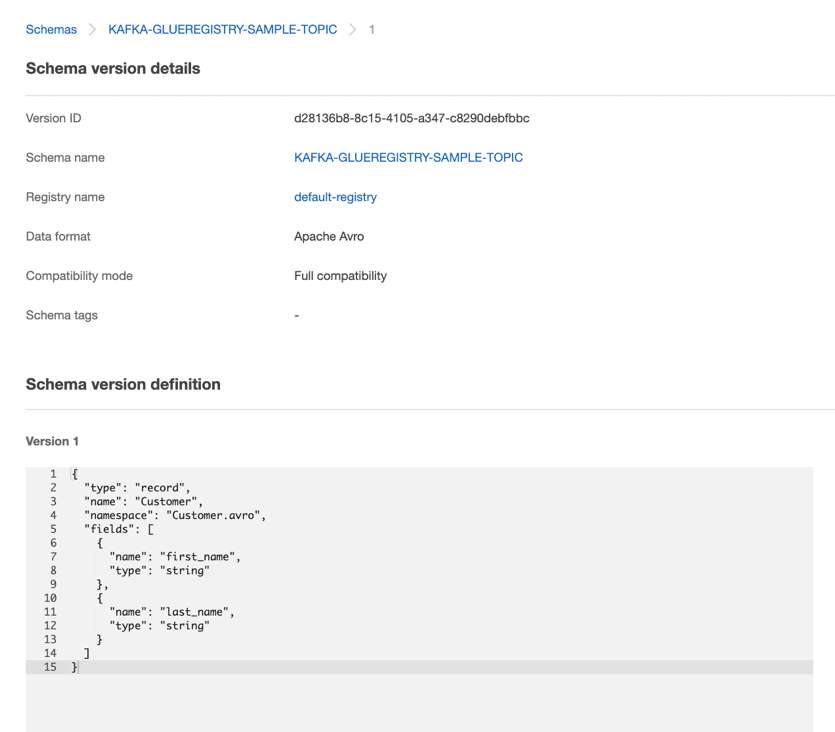
I provided MSKConnect policies, MSKCluster full access, glue full access, firehose full access



**Glue Shema registry screen shots (automatically created when the connector code runs )**





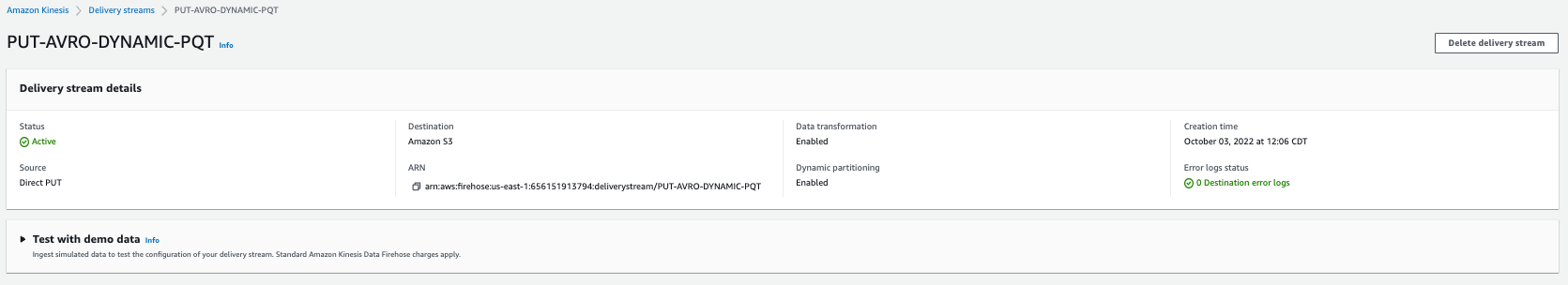


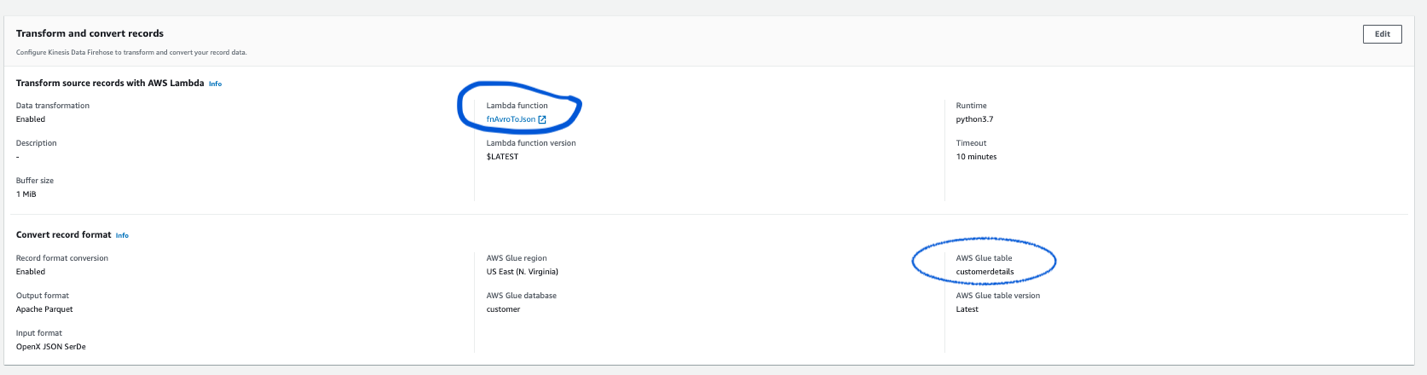
**Firehose Setup**

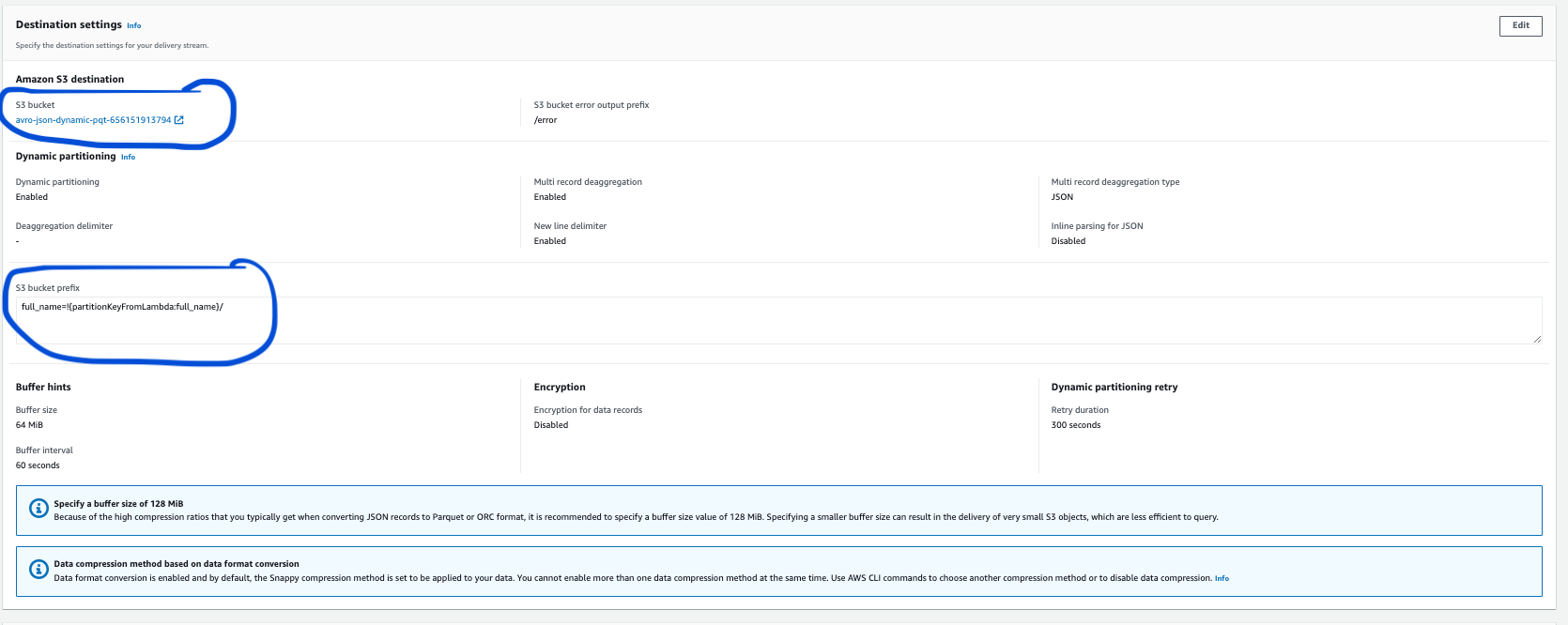
Delivery stream name **PUT-AVRO-DYNAMIC-PQT**

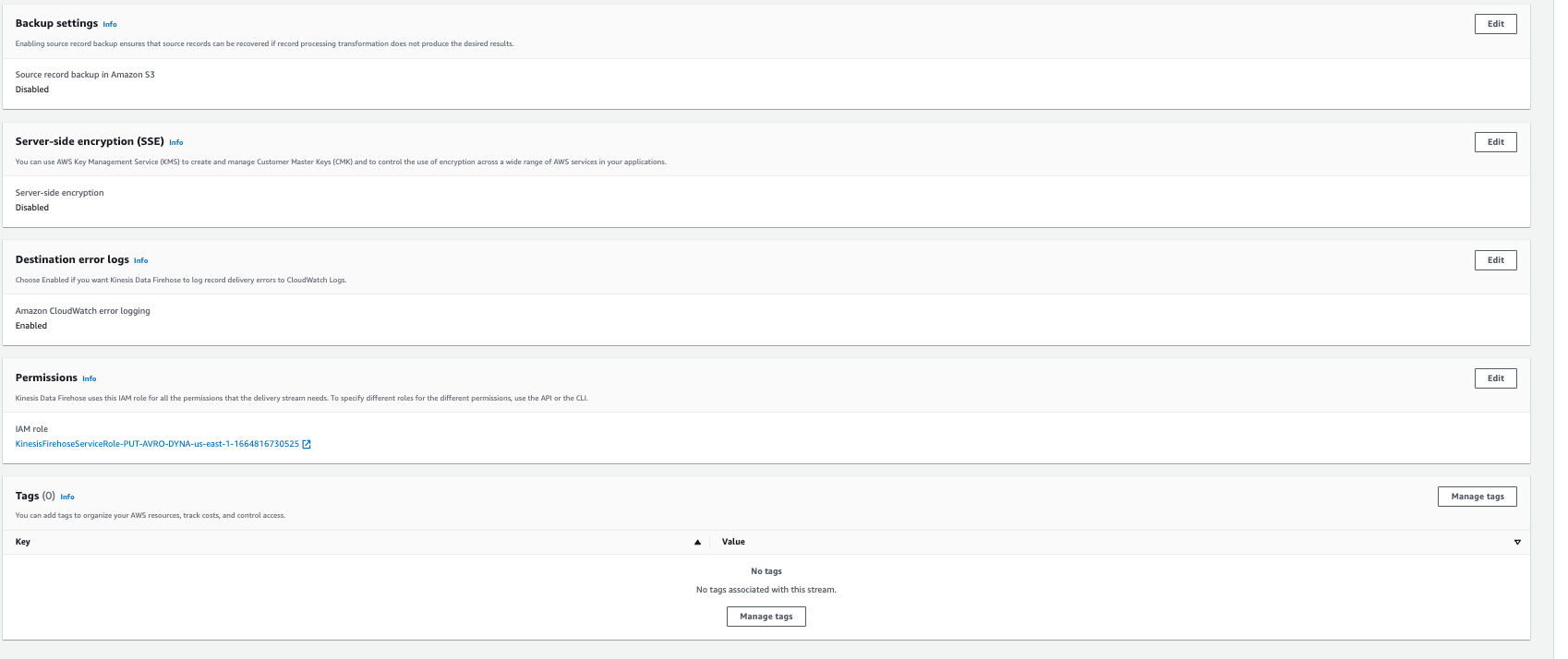
**Description-**The firehose stream receives data in avro format and calls lambda function which converts data to json , adds a new field full\_name into the data field, add full\_name as a partition key and puts it back into the firehose delivery stream. If I don’t want full\_name field to be added as a column field (attribute) in the actual parquet file , there is no need to add it to the glue table which KDF will use for source mapping.

This glue table is separate from the glue schema registry used by the connector. The glue schema table is used when you want to convert json data to ORC/Parquet format and is applied on the incoming source data in KDF. Incoming source data in this case is data that is coming back to KDF after being transformed by the lambda.

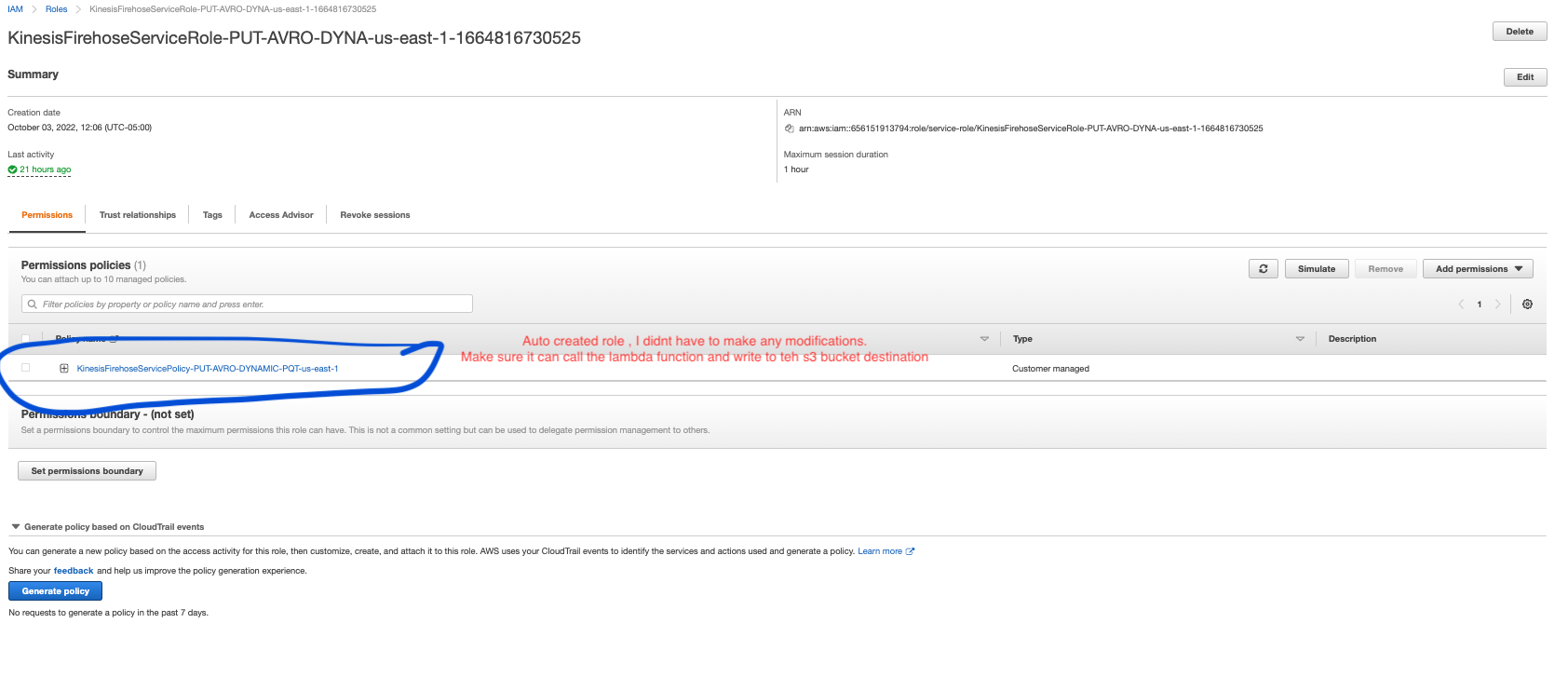
****



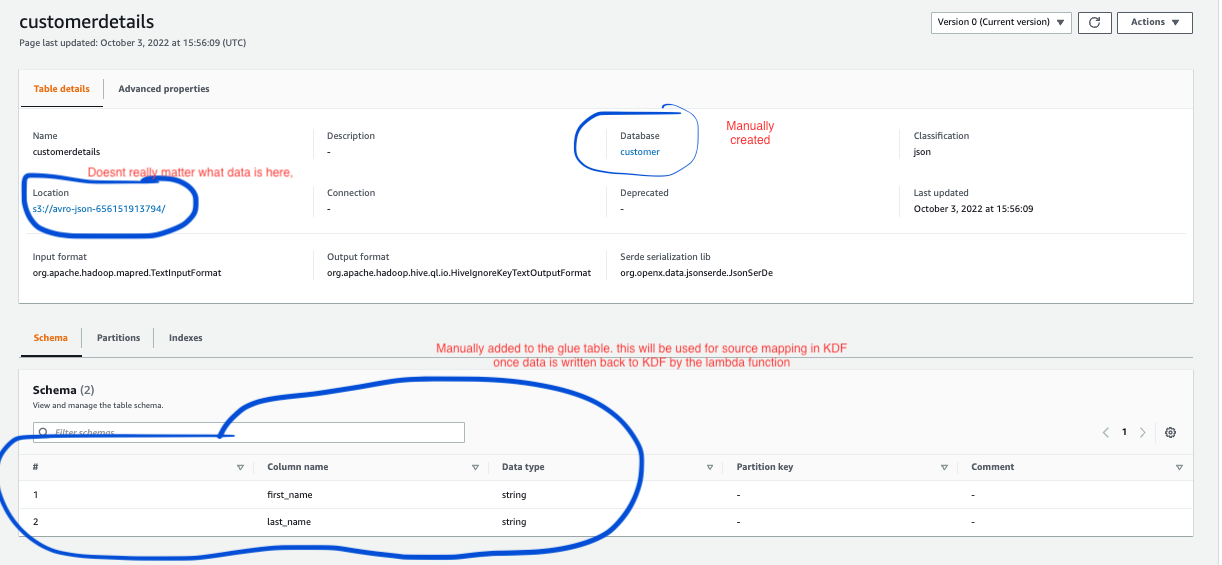




IAM Role for Firehose



Glue table settings for source mapping that will be called inside KDF after return from lambda



**Lambda Code**

Code Location - git clone <https://PiyaliPersonal@bitbucket.org/microservices_poc_cd/firehose-lambda-dynamic-avro-to-json.git>

How to test -Select template as firehose and create a sample event with base64 encoded data

{

"invocationId": "invocationIdExample",

"deliveryStreamArn": "arn:aws:kinesis:EXAMPLE",

"region": "us-east-1",

"records": [

{

"recordId": "49546986683135544286507457936321625675700192471156785154",

"approximateArrivalTimestamp": 1495072949453,

"data": "'T2JqAQQWYXZyby5zY2hlbWH6AnsidHlwZSI6InJlY29yZCIsIm5hbWUiOiJDdXN0b21lciIsIm5hbWVzcGFjZSI6IkN1c3RvbWVyLmF2cm8iLCJmaWVsZHMiOlt7Im5hbWUiOiJmaXJzdF9uYW1lIiwidHlwZSI6InN0cmluZyJ9LHsibmFtZSI6Imxhc3RfbmFtZSIsInR5cGUiOiJzdHJpbmcifV0sImNvbm5lY3QubmFtZSI6IkN1c3RvbWVyLmF2cm8uQ3VzdG9tZXIifRRhdnJvLmNvZGVjCG51bGwAxgOHt3Rlr1xYX3GToa8rNgIaBkFkYRBMb3ZlbGFjZcYDh7d0Za9cWF9xk6GvKzY='"

}

]

}

It is easy to get base64 encoded data by having this startement. I got it working by making sure connector connects to firehose and firehose sends data to lamabda and then modified lambda code to get the base 64 dump which then helped me to unit test the lambda.

>>print(firehose\_record\_input['data'])

**Architecture:**

