Getting Started with Amazon Kinesis Data Streams

The information in this section helps you get started using Amazon Kinesis Data Streams. If you are new to Kinesis Data Streams, start by becoming familiar with the concepts and terminology presented in Amazon Kinesis Data Streams Terminology and Concepts.

This section shows you how to perform basic Amazon Kinesis Data Streams operations using the AWS Command Line Interface. You will learn fundamental Kinesis Data Streams data flow principles and the steps necessary to put and get data from an Kinesis data stream.

Topics

- Install and Configure the AWS CLI
- Perform Basic Kinesis Data Stream Operations Using the AWS CLI

For CLI access, you need an access key ID and a secret access key. Use temporary credentials instead of long-term access keys when possible. Temporary credentials include an access key ID, a secret access key, and a security token that indicates when the credentials expire. For more information, see Best practices for managing AWS access keys in the *AWS General Reference*.

You can find detailed step-by-step IAM and security key set up instructions at Create an IAM User.

In this section, the specific commands discussed are given verbatim, except where specific values are necessarily different for each run. Also, the examples are using the US West (Oregon) region, but the steps in this section work in any of the regions where Kinesis Data Streams is supported.

Install and Configure the AWS CLI

Install AWS CLI

For detailed steps on how to install the AWS CLI for Windows and for Linux, OS X, and Unix operating systems, see Installing the AWS CLI.

Use the following command to list available options and services:

```
aws help
```

You will be using the Kinesis Data Streams service, so you can review the AWS CLI subcommands related to Kinesis Data Streams using the following command:

```
aws kinesis help
```

This command results in output that includes the available Kinesis Data Streams commands:

```
AVAILABLE COMMANDS

        o add-tags-to-stream

        o create-stream

        o delete-stream

        o describe-stream

        o get-records
```

o get-shard-iterator

o help

o list-streams

o list-tags-for-stream

o merge-shards

o put-record

o put-records

o remove-tags-from-stream

o split-shard

o wait

This command list corresponds to the Kinesis Data Streams API documented in the Amazon Kinesis Service API Reference. For example, the create-stream command corresponds to the CreateStream API action.

The AWS CLI is now successfully installed, but not configured. This is shown in the next section.

Configure AWS CLI

For general use, the aws configure command is the fastest way to set up your AWS CLI installation. For more information, see Configuring the AWS CLI.

Perform Basic Kinesis Data Stream Operations Using the AWS CLI

This section describes basic use of a Kinesis data stream from the command line using the AWS CLI. Be sure you are familiar with the concepts discussed in Amazon Kinesis Data Streams Terminology and Concepts.

Note

After you create a stream, your account incurs nominal charges for Kinesis Data Streams usage because Kinesis Data Streams is not eligible for the AWS free tier. When you are finished with this tutorial, delete your AWS resources to stop incurring charges. For more information, see Step 4: Clean Up.

Topics

Step 1: Create a Stream

Your first step is to create a stream and verify that it was successfully created. Use the following command to create a stream named "Foo":

```
aws kinesis create-stream --stream-name Foo
```

Next, issue the following command to check on the stream's creation progress:

```
aws kinesis describe-stream-summary --stream-name Foo
```

You should get output that is similar to the following example:

```
{
   "StreamDescriptionSummary": {
      "StreamName": "Foo",
      "StreamARN": "arn:aws:kinesis:us-west-2:123456789012:stream/Foo",
      "StreamStatus": "CREATING",
      "RetentionPeriodHours": 48,
      "StreamCreationTimestamp": 1572297168.0,
      "EnhancedMonitoring": [
         {
```

```
            "ShardLevelMetrics": []
        }
    ],
    "EncryptionType": "NONE",
    "OpenShardCount": 3,
    "ConsumerCount": 0
  }
}
```

In this example, the stream has a status CREATING, which means it is not quite ready to use. Check again in a few moments, and you should see output similar to the following example:

```
{
  "StreamDescriptionSummary": {
    "StreamName": "Foo",
    "StreamARN": "arn:aws:kinesis:us-west-2:123456789012:stream/Foo",
    "StreamStatus": "ACTIVE",
    "RetentionPeriodHours": 48,
    "StreamCreationTimestamp": 1572297168.0,
    "EnhancedMonitoring": [
        {
            "ShardLevelMetrics": []
```

```
        }
      ],
      "EncryptionType": "NONE",
      "OpenShardCount": 3,
      "ConsumerCount": 0
    }
}
```

There is information in this output that you don't need to be concerned about for this tutorial. The main thing for now is "StreamStatus": "ACTIVE", which tells you that the stream is ready to be used, and the information on the single shard that you requested. You can also verify the existence of your new stream by using the list-streams command, as shown here:

```
aws kinesis list-streams
```

Output:

```
{
  "StreamNames": [
    "Foo"
  ]
}
```

Step 2: Put a Record

Now that you have an active stream, you are ready to put some data. For this tutorial, you will use the simplest possible command, put-record, which puts a single data record containing the text "testdata" into the stream:

```
aws kinesis put-record --stream-name Foo --partition-key 123 --data testdata
```

This command, if successful, will result in output similar to the following example:

```
{

   "ShardId": "shardId-000000000000",

   "SequenceNumber":
"495469866831355442865074579363216256757001924711567851 54"

}
```

Congratulations, you just added data to a stream! Next you will see how to get data out of the stream.

Step 3: Get the Record

GetShardIterator

Before you can get data from the stream you need to obtain the shard iterator for the shard you are interested in. A shard iterator represents the position of the stream and shard from which the consumer (get-record command in this case) will read. You'll use the get-shard-iterator command, as follows:

```
aws kinesis get-shard-iterator --shard-id shardId-000000000000 --shard-iterator-type TRIM_HORIZON --stream-name Foo
```

Recall that the aws kinesis commands have a Kinesis Data Streams API behind them, so if you are curious about any of the parameters shown, you can read about

them in the GetShardIterator API reference topic. Successful execution will result in output similar to the following example (scroll horizontally to see the entire output):

```
{

   "ShardIterator":
"AAAAAAAAAAHSywljv0zEgPX4NyKdZ5wryMzP9yALs8NeKbUjp1IxtZs1Sp+KEd9I6AJ
9ZG4lNR1EMi+9Md/nHvtLyxpfhEzYvkTZ4D9DQVz/mBYWRO6OTZRKnW9gd+efGN2a
HFdkH1rJl4BL9Wyrk+ghYG22D2T1Da2EyNSH1+LAbK33gQweTJADBdyMwlo5r6PqcP
2dzhg="

}
```

The long string of seemingly random characters is the shard iterator (yours will be different). You will need to copy/paste the shard iterator into the get command, shown next. Shard iterators have a valid lifetime of 300 seconds, which should be enough time for you to copy/paste the shard iterator into the next command. Notice you will need to remove any newlines from your shard iterator before pasting to the next command. If you get an error message that the shard iterator is no longer valid, simply execute the get-shard-iterator command again.

GetRecords

The get-records command gets data from the stream, and it resolves to a call to GetRecords in the Kinesis Data Streams API. The shard iterator specifies the position in the shard from which you want to start reading data records sequentially. If there are no records available in the portion of the shard that the iterator points to, GetRecords returns an empty list. Note that it might take multiple calls to get to a portion of the shard that contains records.

In the following example of the get-records command (scroll horizontally to see the entire command):

```
aws kinesis get-records --shard-iterator
AAAAAAAAAAHSywljv0zEgPX4NyKdZ5wryMzP9yALs8NeKbUjp1IxtZs1Sp+KEd9I6AJ9
ZG4lNR1EMi+9Md/nHvtLyxpfhEzYvkTZ4D9DQVz/mBYWRO6OTZRKnW9gd+efGN2aH
```

```
FdkH1rJl4BL9Wyrk+ghYG22D2T1Da2EyNSH1+LAbK33gQweTJADBdyMwlo5r6PqcP2
dzhg=
```

If you are running this tutorial from a Unix-type command processor such as bash, you can automate the acquisition of the shard iterator using a nested command, like this (scroll horizontally to see the entire command):

```
SHARD_ITERATOR=$(aws kinesis get-shard-iterator --shard-id shardId-
000000000000 --shard-iterator-type TRIM_HORIZON --stream-name Foo --query
'ShardIterator')



aws kinesis get-records --shard-iterator $SHARD_ITERATOR
```

If you are running this tutorial from a system that supports PowerShell, you can automate acquisition of the shard iterator using a command such as this (scroll horizontally to see the entire command):

```
aws kinesis get-records --shard-iterator ((aws kinesis get-shard-iterator --shard-id
shardId-000000000000 --shard-iterator-type TRIM_HORIZON --stream-name
Foo).split('"')[4])
```

The successful result of the get-records command will request records from your stream for the shard that you specified when you obtained the shard iterator, as in the following example (scroll horizontally to see the entire output):

```
{

  "Records":[ {

    "Data":"dGVzdGRhdGE=",

    "PartitionKey":"123”,

    "ApproximateArrivalTimestamp": 1.441215410867E9,


"SequenceNumber":"49544985256907370027570885864065577703022652638596
431874"
```

```
 } ],

 "MillisBehindLatest":24000,


"NextShardIterator":"AAAAAAAAAAEDOW3ugseWPE4503kqN1yN1UaodY8unE0sYslM
UmC6lX9hlig5+t4RtZM0/tALfiI4QGjunVgJvQsjxjh2aLyxaAaPr+LaoENQ7eVs4EdYXgK
yThTZGPcca2fVXYJWL3yafv9dsDwsYVedI66dbMZFC8rPMWc797zxQkv4pSKvPOZvrUI
udb8UkH3VMzx58Is="

}
```


Note that get-records is described above as a *request*, which means you may receive zero or more records even if there are records in your stream, and any records returned may not represent all the records currently in your stream. This is perfectly normal, and production code will simply poll the stream for records at appropriate intervals (this polling speed will vary depending on your specific application design requirements).

The first thing you'll likely notice about your record in this part of the tutorial is that the data appears to be garbage –; it's not the clear text testdata we sent. This is due to the way put-record uses Base64 encoding to allow you to send binary data. However, the Kinesis Data Streams support in the AWS CLI does not provide Base64 *decoding* because Base64 decoding to raw binary content printed to stdout can lead to undesired behavior and potential security issues on certain platforms and terminals. If you use a Base64 decoder (for example, https://www.base64decode.org/) to manually decode dGVzdGRhdGE= you will see that it is, in fact, testdata. This is sufficient for the sake of this tutorial because, in practice, the AWS CLI is rarely used to consume data, but more often to monitor the state of the stream and obtain information, as shown previously (describe-stream and list-streams). Future tutorials will show you how to build production-quality consumer applications using the Kinesis Client Library (KCL), where Base64 is taken care of for you. For more information about the KCL, see Developing Custom Consumers with Shared Throughput Using KCL.

It's not always the case that get-records will return all records in the stream/shard specified. When that happens, use the NextShardIterator from the last result to get the next set of records. So if more data were being put into the stream (the normal

situation in production applications), you could keep polling for data using get-records each time. However, if you do not call get-records using the next shard iterator within the 300 second shard iterator lifetime, you will get an error message, and you will need to use the get-shard-iterator command to get a fresh shard iterator.

Also provided in this output is MillisBehindLatest, which is the number of milliseconds the GetRecords operation's response is from the tip of the stream, indicating how far behind current time the consumer is. A value of zero indicates record processing is caught up, and there are no new records to process at this moment. In the case of this tutorial, you may see a number that's quite large if you've been taking time to read along as you go. That's not a problem, by default, data records stay in a stream for 24 hours waiting for you to retrieve them. This time frame is called the retention period and it is configurable up to 365 days.

Note that a successful get-records result will always have a NextShardIterator even if there are no more records currently in the stream. This is a polling model that assumes a producer is potentially putting more records into the stream at any given time. Although you can write your own polling routines, if you use the previously mentioned KCL for developing consumer applications, this polling is taken care of for you.

If you call get-records until there are no more records in the stream and shard you are pulling from, you will see output with empty records similar to the following example (scroll horizontally to see the entire output):


```
{

  "Records": [],

  "NextShardIterator":
"AAAAAAAAAAGCJ5jzQNjmdhO6B/YDIDE56jmZmrmMA/r1WjoHXC/kPJXc1rckt3TFL5
5dENfe5meNgdkyCRpUPGzJpMgYHaJ53C3nCAjQ6s7ZupjXeJGoUFs5oCuFwhP+Wul/E
hyNeSs5DYXLSSC5XCapmCAYGFjYER69QSdQjxMmBPE/hiybFDi5qtkT6/PsZNz6kFoqt
Dk="

}
```

Step 4: Clean Up

Finally, you'll want to delete your stream to free up resources and avoid unintended charges to your account, as previously noted. Do this in practice any time you have created a stream and will not be using it because charges accrue per stream whether you are putting and getting data with it or not. The clean-up command is simple:

```
aws kinesis delete-stream --stream-name Foo
```

Success results in no output, so you might want to use describe-stream to check on deletion progress:

```
aws kinesis describe-stream-summary --stream-name Foo
```

If you execute this command immediately after the delete command, you will likely see output part of which is similar to the following example:

```
{
   "StreamDescriptionSummary": {
      "StreamName": "samplestream",
      "StreamARN": "arn:aws:kinesis:us-west-
2:123456789012:stream/samplestream",
      "StreamStatus": "ACTIVE",
```

After the stream is fully deleted, describe-stream will result in a "not found" error:

```
A client error (ResourceNotFoundException) occurred when calling the
DescribeStreamSummary operation:

Stream Foo under account 123456789012
```