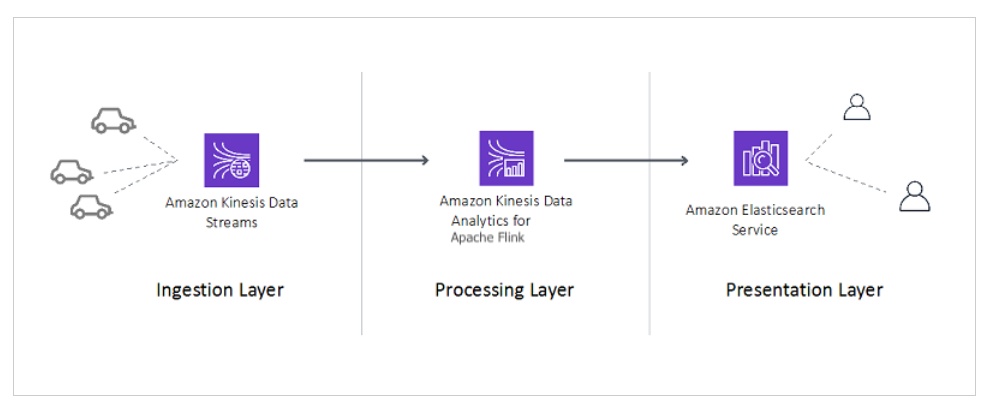
**Implement a Real-time Data Processing Pipeline with Kinesis.**

In real-time stream processing, it becomes critical to collect, process, and analyze high-velocity real-time data to provide timely insights and react quickly to new information. Streaming data velocity could be unpredictable, and volume could spike based on user demand at a given time of day. Real-time analysis needs to handle the data spike, because any delay in processing streaming data can cause a significant impact on the desired outcome. The value of insights and actions reduces over time, whereas real-time analysis can substantially improve the effectiveness of the analytics application.

**Architecture**

As shown in the following architecture diagram, every taxi in the fleet is capturing information about completed trips. The tracked information includes the pickup and drop-off locations, number of passengers, and generated revenue. This information is ingested into [Amazon Kinesis Data Streams](https://aws.amazon.com/kinesis/data-streams/) as a simple JSON blob. The application reads the timestamp attribute of the stored events and replays them as if they occurred in real time. From there, the data is processed and analyzed by a Flink application, which is deployed to Kinesis Data Analytics for Apache Flink.

The application identifies areas that are currently requesting a high number of taxi rides. The derived insights are finally persisted into [Amazon Elasticsearch Service](https://aws.amazon.com/elasticsearch-service/), where they can be accessed and visualized using [Kibana](https://aws.amazon.com/elasticsearch-service/the-elk-stack/kibana/).



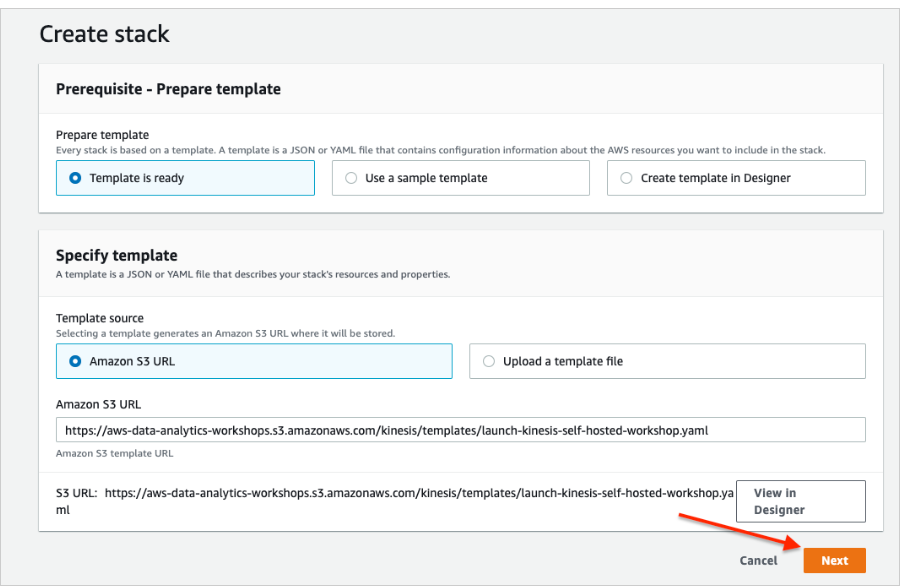
**Deploy the real-time streaming and analysis workload**

This section uses an [AWS CloudFormation](http://aws.amazon.com/cloudformation) template to build a producer client program that sends NYC taxi trip data to our Kinesis data stream. The template creates the following resources:

* An S3 bucket to house the data resources.
* A new Kinesis data stream that we use to stream a dataset of NYC taxi trips.
* Amazon ElasticSearch cluster with Kibana integration for displaying dashboard information.
* A build pipeline and [AWS CodeBuild](http://aws.amazon.com/codebuild) project along with sources for a Flink Kinesis connector application.
* An EC2 instance for running a Flink application to replay data onto the data stream. An Elastic IP is provisioned for the EC2 instance to allow SSH access.
* A Java application hosted on the EC2 instance, which loads data from the EC2 instance.
* A Kinesis data analytics application to continuously monitor and analyze data from the connected data stream and run the Apache Flink 1.11 application.
* The necessary [AWS Identity and Access Management](http://aws.amazon.com/iam) (IAM) service roles, security groups, and policies to run and communicate between the resources you create.
* An [Amazon CloudWatch](http://aws.amazon.com/cloudwatch) alarm when the application falls behind based on the millisBehindLatest metric.

To provision these resources, complete the following steps:

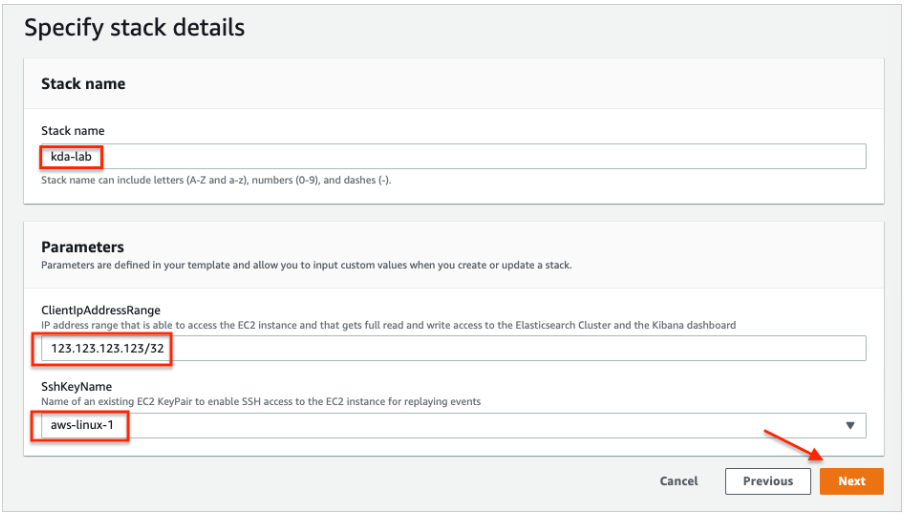
1. Choose [Launch Stack](https://console.aws.amazon.com/cloudformation/home?region=us-east-1#/stacks/create/template?stackName=kinesis-workshop&templateURL=https://aws-bigdata-blog.s3.amazonaws.com/artifacts/kds-kpu-optimization/kinesis-analytics-taxi-consumer.yml) (right click) and open a new tab to run the CloudFormation template. Make sure to choose us-east-1 ( N. Virginia) region.
2. Choose Next.



1. For Stack name, enter a name.
2. For ClientipAddressRange, enter the IP address from which the Kibana dashboard is accessed.

Use [https://checkup.amazon.com](https://checkup.amazon.com/) to find IP and add /32 at the end.

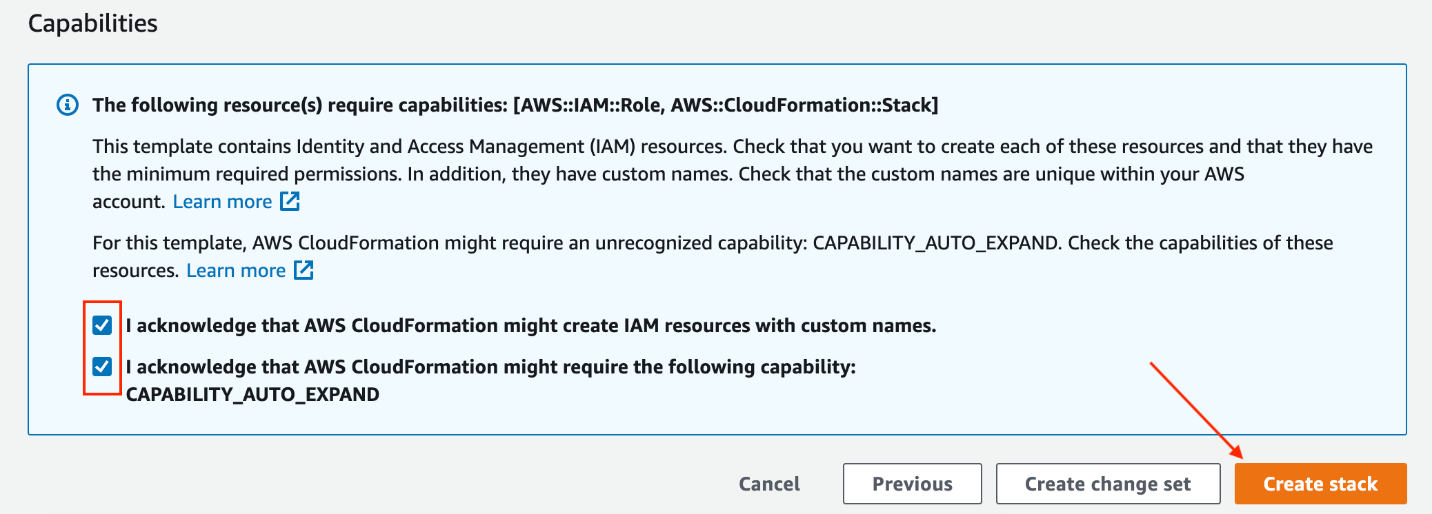
1. For SshKeyName¸ enter the SSH key name for the Linux-based EC2 instance you created.
2. Choose Next.



7.Select I acknowledge that AWS CloudFormation might create IAM resources with custom names and I acknowledge that AWS CloudFormation might require the following capability: CAPABILITY\_AUTO\_EXPAND.

8.Choose Create stack.

9.Wait until the CloudFormation template has been successfully created.



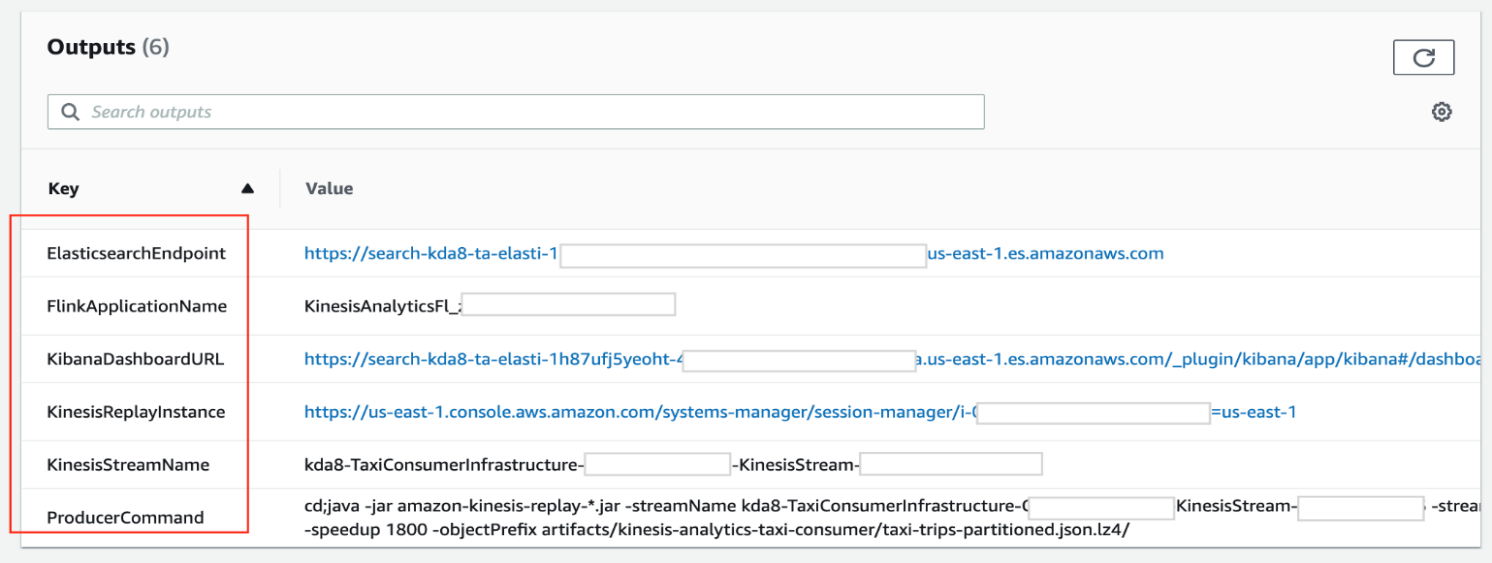
**Connect to the new EC2 instance and run the producer client program**

In this section, we connect to our new EC2 instance and run the producer client program.

1. On the AWS CloudFormation console, choose the parent in which the stack was deployed.
2. In the Outputs section, locate the [AWS Systems Manager Session Manager](https://docs.aws.amazon.com/systems-manager/latest/userguide/session-manager.html) URL

for KinesisReplayInstance.

1. Choose the Session Manager URL to connect to the EC2 instance.



1. After the connection has been established, start ingesting events into the Kinesis data stream by running the JAR file that was downloaded to the EC2 instance.

The command with pre-populated parameters is available in the Outputs section of the CloudFormation template for ProducerCommand.

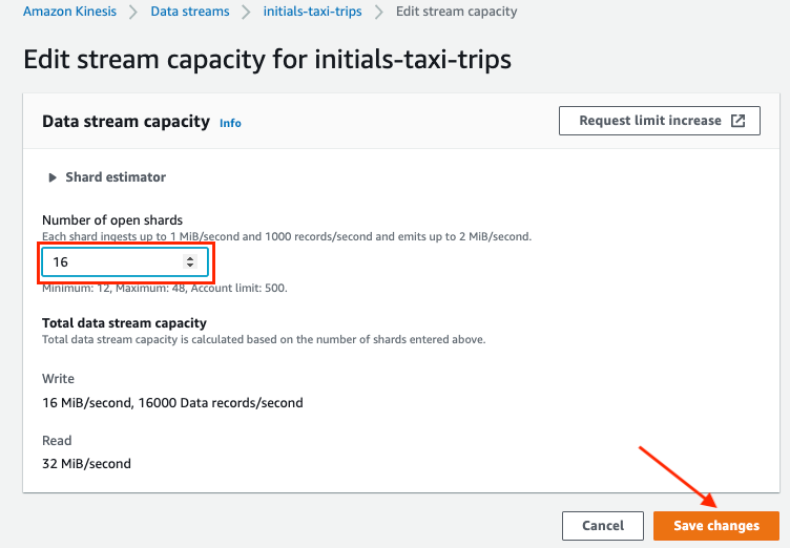
1. On the Kinesis Data Streams console, go to your data stream.
2. On the Monitoring tab, locate the Incoming Data – Sum
3. AWS CloudFormation automatically grants access to the IP address provided during stack creation. However, if you encounter access issues in the Kibana dashboard, modify your Amazon ES domain’s access policy and change your local IP address on the Amazon ES console.
4. To change your IP address, find and choose the Amazon ES domain that you provisioned. On the **Actions**menu, choose **Modify access policy**.

**Scale the Kinesis data stream to adapt to a spiky data stream.**

**The Kinesis data stream was deliberately under-provisioned so that the Kinesis Replay Java application is completely saturating the data stream. When you closely inspect the output of the Java application, you can notice that the replay lag is continuously increasing. This means that the producer can’t ingest events as quickly as required according to the specified speedup parameter.**

We then observe how the Kinesis Data Analytics for Apache Flink application automatically scales to adapt to the increased throughput.

1. On the Kinesis Data Streams console, navigate to the stream you created.
2. In the Shards section, choose Edit.
3. Double the throughput of the Kinesis stream by changing the number of open shards to 16.
4. Choose Save changes to confirm the changes.



1. While the service doubles the number of shards and therefore the throughput of the stream, examine the metrics of the data stream on the Monitoring.

## Calibrate KPUs

Currently, Kinesis Data Analytics scales your application solely based on the underlying CPU usage. However, because not all applications are CPU bound, depending on your needs, you may want to use a different mechanism for sizing your application. In this section, we demonstrate how you can use the millisBehindLatest metric (available when consuming data from a Kinesis data stream) to responsively size your Kinesis data analytics application.

Kinesis Data Analytics provisions capacity in the form of Amazon Kinesis Processing Units (KPUs). One KPU provides you with 1 vCPU and 4 GB memory. The default limit for KPUs for your application is 32. You can also request an increase to this limit in [**AWS Service Limits**](https://docs.aws.amazon.com/general/latest/gr/aws_service_limits.html)**.**

Kinesis Data Analytics calculates the KPUs needed to run your application as Parallelism/ParallelismPerKPU.

The following example request for the [CreateApplication](https://docs.aws.amazon.com/kinesisanalytics/latest/apiv2/API_CreateApplication.html" \t "_blank) action sets parallelism when you create an application.

{

"ApplicationName": "string",

"RuntimeEnvironment":"FLINK-1\_11",

"ServiceExecutionRole":"arn:aws:iam::123456789123:role/myrole",

"ApplicationConfiguration": {

"ApplicationCodeConfiguration":{

"CodeContent":{

"S3ContentLocation":{

"BucketARN":"arn:aws:s3:::mybucket",

"FileKey":"myflink.jar",

"ObjectVersion":"AbCdEfGhIjKlMnOpQrStUvWxYz12345"

}

},

"CodeContentType":"ZIPFILE"

},

"FlinkApplicationConfiguration": {

"ParallelismConfiguration": {

"AutoScalingEnabled": "true",

"ConfigurationType": "CUSTOM",

"Parallelism": 4,

"ParallelismPerKPU": 4

}

}

}

}

For more examples and instructions for using request blocks with API actions, see [Kinesis Data Analytics API Example Code](https://docs.aws.amazon.com/kinesisanalytics/latest/java/api-examples.html).

The following example request for the [UpdateApplication](https://docs.aws.amazon.com/kinesisanalytics/latest/apiv2/API_UpdateApplication.html" \t "_blank) action sets parallelism for an existing application:

{

"ApplicationName": "MyApplication",

"CurrentApplicationVersionId": 4,

"ApplicationConfigurationUpdate": {

"FlinkApplicationConfigurationUpdate": {

"ParallelismConfigurationUpdate": {

"AutoScalingEnabledUpdate": "true",

"ConfigurationTypeUpdate": "CUSTOM",

"ParallelismPerKPUUpdate": 4,

"ParallelismUpdate": 4

}

}

}

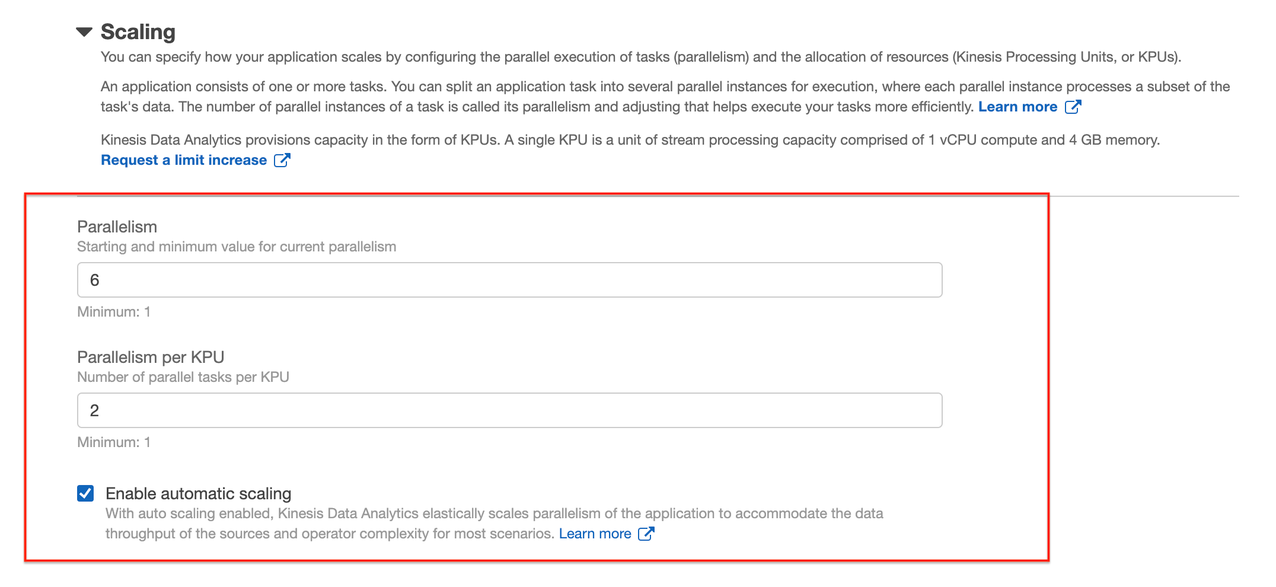
}

**Scale the Kinesis Data Analytics for Apache Flink application**

Kinesis Data Analytics natively supports auto scaling. After few minutes, you can see the effect of the auto scaling activities in the metrics. The millisBehindLatest metric starts to decrease until it reaches zero when the processing has caught up with the tip of the Kinesis data stream.

You can calibrate the scaling operation based on your application needs by adjusting the KPU.

1. On the Kinesis Data Analytics console, navigate to your application.
2. Under Scaling, choose Configure.
3. Adjust Parallelism to 6 and Parallelism per KPU to 2.
4. Choose Update.



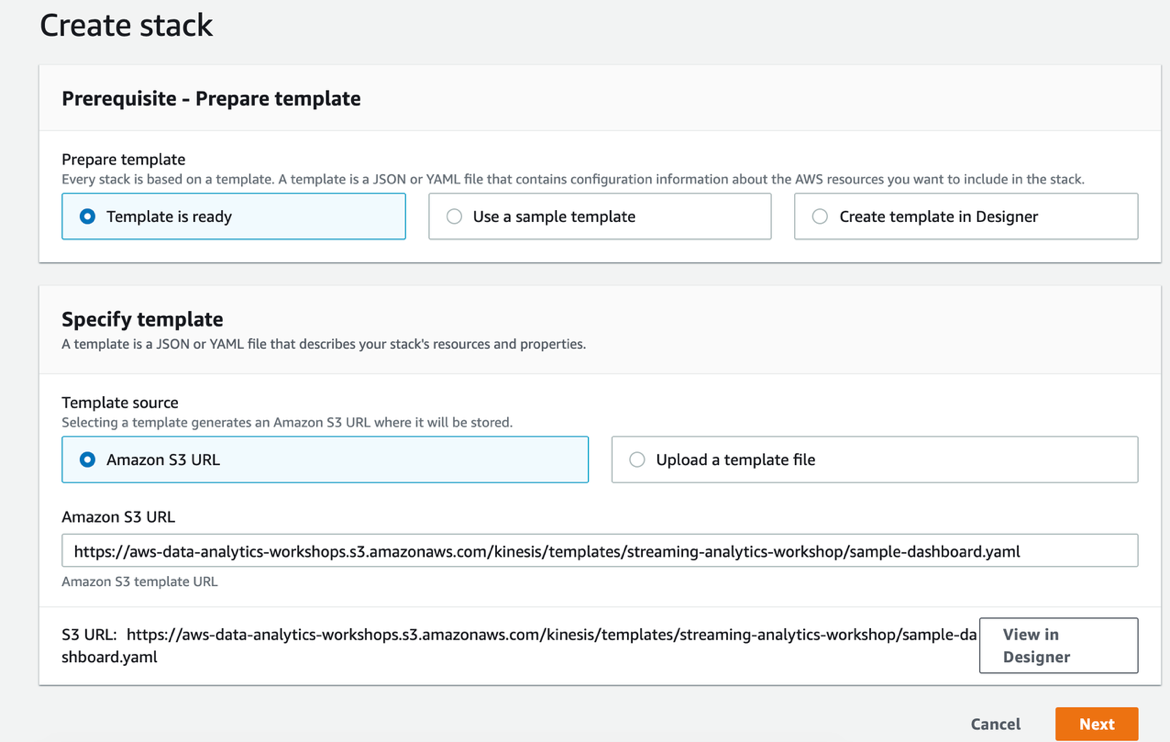
The CloudFormation template includes the following components:

* An advanced [Amazon CloudWatch](http://aws.amazon.com/cloudwatch) dashboard
* Two CloudWatch alarms for scaling your Kinesis Data Analytics for Apache Flink application
* Two CloudWatch alarms for scaling your Kinesis Data Streams
* Accompanying auto scaling policy actions in these alarms
* An [Amazon API Gateway](https://aws.amazon.com/api-gateway) endpoint for triggering [AWS Lambda](http://aws.amazon.com/lambda)
* A Lambda function responsible for handling the scale-in and scale-out functions

These components work in tandem to monitor the metrics configured in the CloudWatch alarm and respond to metrics accordingly.

To provision your resources, complete the following steps:

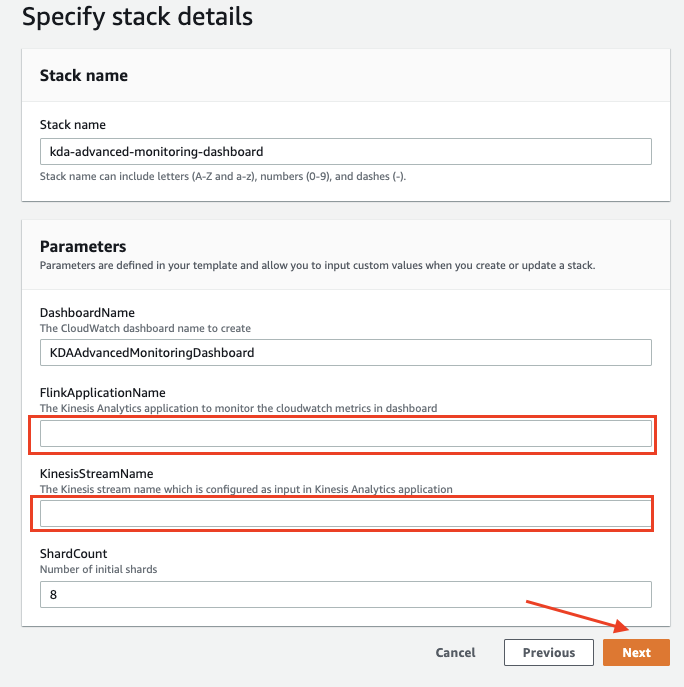
1. Choose [Launch Stack](https://console.aws.amazon.com/cloudformation/home?region=us-east-1#/stacks/create/template?stackName=kinesis-workshop-scaling&templateURL=https://aws-bigdata-blog.s3.amazonaws.com/artifacts/kds-kpu-optimization/kinesis-monitoring-scaling.yml) (right-click) to open a new tab to run the CloudFormation template. Make sure to choose us-east-1 (N. Virginia) region.
2. Choose Next.



1. For FlinkApplicationName, enter the name of your application.
2. For KinesisStreamName, enter the name of your data stream.
3. Make sure ShardCount is same as the current shard count of Kinesis Data Streams created in [Part 1](https://aws.amazon.com/blogs/big-data/part1-build-and-optimize-a-real-time-stream-processing-pipeline-with-amazon-kinesis-data-analytics-for-apache-flink/).

This information is available on the Outputs tab of the CloudFormation stack detailed in [Part 1](https://aws.amazon.com/blogs/big-data/part1-build-and-optimize-a-real-time-stream-processing-pipeline-with-amazon-kinesis-data-analytics-for-apache-flink/).

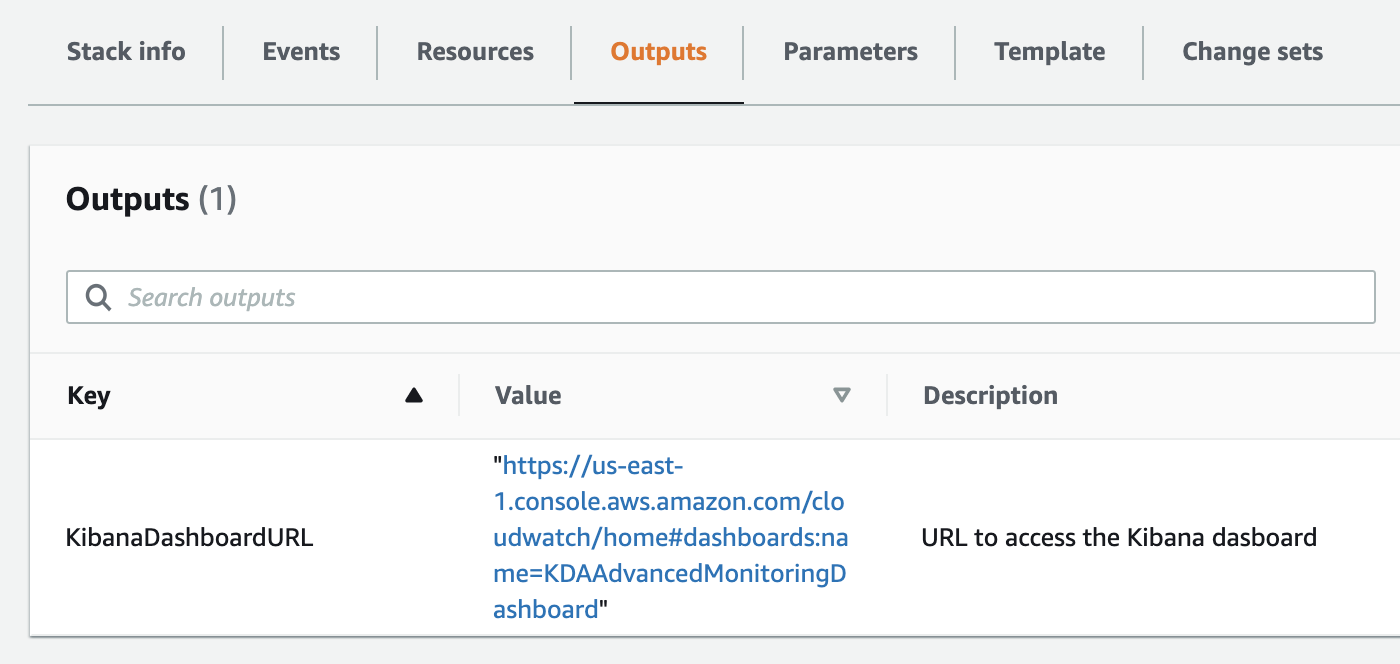
1. Choose Next.



1. Follow the remaining instructions and choose Create stack.

This dashboard should take a few minutes to launch.

1. When the stack is complete, go to the Outputs tab of the stack on the AWS CloudFormation console and choose the dashboard link.



Currently, the settings are tuned to the max throughput per KPU, which is ideal for a production workload. Let’s tune this setting down to a lower value to more quickly see results.

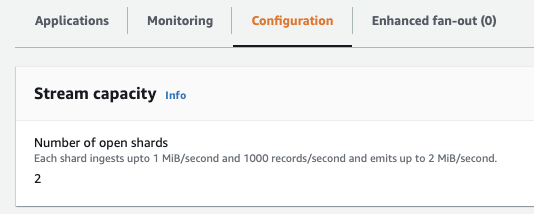
1. On the CloudWatch console, choose Alarms in the navigation pane.
2. Choose the alarm KDAScaleOutAlam.

The alarm has been preconfigured for you in CloudWatch to listen for specific metrics breaching a threshold. Optionally, you can adjust the alarm to trigger scale-out or scale-in events as needed.

1. On the Actions menu, choose Edit.
2. In the Conditions section, adjust the threshold value as needed.

5.Choose Update alarm.

On the Configuration tab of your data stream, observe the scaling operation of allocated shards.



**Conclusion**

In this post, you built a reliable, scalable, and highly available advanced scaling mechanism for streaming applications based on Kinesis Data Analytics for Apache Flink and Kinesis Data Streams. The post also discussed how to auto scale your applications based on a metric other than CPU utilization and explored ways to extend observability of the application with advanced monitoring and error handling. This solution was largely enabled by using managed services, so you didn’t need to spend time provisioning and configuring the underlying infrastructure.

You should now have a good understanding of how to build, monitor and auto scale a real-time streaming application using [Amazon Kinesis](https://aws.amazon.com/kinesis/). You can also calibrate various components based on your application needs and volume of data by applying advanced monitoring and scaling techniques.