Contents

1 kinesis-example-scala-consumer

1.1 KinesisConsumerApp.scala

```
1 /*
2 * Copyright (c) 2013-2014 Snowplow Analytics Ltd. with significant
3 * portions copyright 2012-2014 Amazon.
4 * All rights reserved.
5 *
```

```
6  * This program is licensed to you under the Apache License Version 2.0,
7  * and you may not use this file except in compliance with the Apache
8  * License Version 2.0.
9  * You may obtain a copy of the Apache License Version 2.0 at
10 * http://www.apache.org/licenses/LICENSE-2.0.
11 *
12 * Unless required by applicable law or agreed to in writing,
13 * software distributed under the Apache License Version 2.0 is distributed
14 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
15 * either express or implied.
16 *
17 * See the Apache License Version 2.0 for the specific language
18 * governing permissions and limitations there under.
19 */
20
21 package com.snowplowanalytics.kinesis.consumer
22
23 // Config
24 import com.typesafe.config.{Config, ConfigFactory}
25
26 // Argot
27 import org.clapper.argot.ArgotParser
28
29 // Java
30 import java.io.{File,FileInputStream,IOException}
31 import java.net.InetAddress
32 import java.util.{Properties,UUID}
33
34 // Amazon.
35 import com.amazonaws.AmazonClientException
36 import com.amazonaws.auth.{
37   AWSCredentials,
38   BasicAWSCredentials,
39   AWSCredentialsProvider,
40   ClasspathPropertiesFileCredentialsProvider
41 }
42 import com.amazonaws.auth.InstanceProfileCredentialsProvider
43 import com.amazonaws.services.kinesis.AmazonKinesisClient
44 import com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorFactory
45 import com.amazonaws.services.kinesis.clientlibrary.lib.worker.{
46   InitialPositionInStream,
47   KinesisClientLibConfiguration,
48   Worker
49 }
50 import com.amazonaws.services.kinesis.metrics.impl.NullMetricsFactory
51
52
53 class KinesisConsumerConfig(config: Config) {
54   private val consumer = config.resolve.getConfig("consumer")
55
56   private val aws = consumer.getConfig("aws")
```

```
57    val accessKey = aws.getString("access-key")
58    val secretKey = aws.getString("secret-key")
59
60    private val stream = consumer.getConfig("stream")
61    val streamName = stream.getString("stream-name")
62    val appName = stream.getString("app-name")
63    println(appName)
64    val initialPosition = stream.getString("initial-position")
65    val streamDataType = stream.getString("data-type")
66    val streamEndpoint = stream.getString("endpoint")
67  }
68
69  object KinesisConsumerApp extends App {
70    val parser = new ArgotParser(
71      programName = generated.Settings.name,
72      compactUsage = true,
73      preUsage = Some("%s: Version %s. Copyright (c) 2013, %s.".format(
74        generated.Settings.name,
75        generated.Settings.version,
76        generated.Settings.organization)
77      )
78    )
79
80    // Optional config argument
81    val config = parser.option[Config](
82        List("config"), "filename", """
83          |Configuration file. Defaults to \"resources/default.conf\"
84          |(within .jar) if not set""".stripMargin) {
85      (c, opt) =>
86        val file = new File(c)
87        if (file.exists) {
88          ConfigFactory.parseFile(file)
89        } else {
90          parser.usage("Configuration file \"%s\" does not exist".format(c))
91          ConfigFactory.empty()
92        }
93    }
94
95    parser.parse(args)
96    val kinesisConsumerConfig = new KinesisConsumerConfig(
97      config.value.getOrElse(ConfigFactory.load("default"))
98    )
99
100   val workerId = InetAddress.getLocalHost().getCanonicalHostName() +
101     ":" + UUID.randomUUID()
102   println("Using workerId: " + workerId)
103
104   val kinesisCredentials = createKinesisCredentials(
105     kinesisConsumerConfig.accessKey,
106     kinesisConsumerConfig.secretKey
107   )
```

```
108   val kinesisClientLibConfiguration = new KinesisClientLibConfiguration(
109     kinesisConsumerConfig.appName,
110     kinesisConsumerConfig.streamName,
111     kinesisCredentials,
112     workerId
113   ).withInitialPositionInStream(
114     InitialPositionInStream.valueOf(kinesisConsumerConfig.initialPosition)
115   )
116
117   println(s"Running: ${kinesisConsumerConfig.appName}.")
118   println(s"Processing stream: ${kinesisConsumerConfig.streamName}")
119
120   val recordProcessorFactory = new RecordProcessorFactory(kinesisConsumerConfig)
121   val worker = new Worker(
122     recordProcessorFactory,
123     kinesisClientLibConfiguration,
124     new NullMetricsFactory()
125   )
126
127   worker.run()
128
129   private def createKinesisCredentials(accessKey: String, secretKey: String):
130       AWSCredentialsProvider =
131     if (isCpf(accessKey) && isCpf(secretKey)) {
132       new ClasspathPropertiesFileCredentialsProvider()
133     } else if (isCpf(accessKey) || isCpf(secretKey)) {
134       throw new RuntimeException(
135         "access-key and secret-key must both be set to 'cpf', or neither"
136       )
137     } else {
138       new BasicAWSCredentialsProvider(
139         new BasicAWSCredentials(accessKey, secretKey)
140       )
141     }
142   private def isCpf(key: String): Boolean = (key == "cpf")
143 }
144
145 class BasicAWSCredentialsProvider(basic: BasicAWSCredentials) extends
146     AWSCredentialsProvider{
147   @Override def getCredentials: AWSCredentials = basic
148   @Override def refresh = {}
149 }
```

1.2 RecordProcessorFactory.scala

```
1  /*
2  * Copyright (c) 2013-2014 Snowplow Analytics Ltd. with significant
3  * portions copyright 2012-2014 Amazon.
4  * All rights reserved.
```

```
5  *
6  * This program is licensed to you under the Apache License Version 2.0,
7  * and you may not use this file except in compliance with the Apache
8  * License Version 2.0.
9  * You may obtain a copy of the Apache License Version 2.0 at
10 * http://www.apache.org/licenses/LICENSE-2.0.
11 *
12 * Unless required by applicable law or agreed to in writing,
13 * software distributed under the Apache License Version 2.0 is distributed
14 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
15 * either express or implied.
16 *
17 * See the Apache License Version 2.0 for the specific language
18 * governing permissions and limitations there under.
19 */
20
21 package com.snowplowanalytics.kinesis.consumer
22
23 import com.amazonaws.services.kinesis.clientlibrary.interfaces.{
24   IRecordProcessor,
25   IRecordProcessorFactory
26 }
27
28 class RecordProcessorFactory(config: KinesisConsumerConfig)
29     extends IRecordProcessorFactory {
30   @Override
31   def createProcessor: IRecordProcessor = {
32     return new RecordProcessor(config);
33   }
34 }
```

1.3 RecordProcessor.scala

```
1  /*
2  * Copyright (c) 2013-2014 Snowplow Analytics Ltd. with significant
3  * portions copyright 2012-2014 Amazon.
4  * All rights reserved.
5  *
6  * This program is licensed to you under the Apache License Version 2.0,
7  * and you may not use this file except in compliance with the Apache
8  * License Version 2.0.
9  * You may obtain a copy of the Apache License Version 2.0 at
10 * http://www.apache.org/licenses/LICENSE-2.0.
11 *
12 * Unless required by applicable law or agreed to in writing,
13 * software distributed under the Apache License Version 2.0 is distributed
14 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
15 * either express or implied.
16 *
```

```
17  * See the Apache License Version 2.0 for the specific language
18  * governing permissions and limitations there under.
19  */
20
21 package com.snowplowanalytics.kinesis.consumer
22
23 import java.util.List
24
25 import com.amazonaws.services.kinesis.clientlibrary.exceptions.{
26   InvalidStateException,
27   ShutdownException,
28   ThrottlingException
29 }
30 import com.amazonaws.services.kinesis.clientlibrary.interfaces.{
31   IRecordProcessor,
32   IRecordProcessorCheckpointer
33 }
34 import com.amazonaws.services.kinesis.clientlibrary.types.ShutdownReason
35 import com.amazonaws.services.kinesis.model.Record
36
37 import scala.util.control.Breaks._
38 import scala.collection.JavaConversions._
39
40 // Thrift.
41 import org.apache.thrift.TDeserializer
42
43 class RecordProcessor(config: KinesisConsumerConfig)
44     extends IRecordProcessor {
45   private val thriftDeserializer = new TDeserializer()
46
47   private var kinesisShardId: String = _
48   private var nextCheckpointTimeInMillis: Long = _
49
50   // Backoff and retry settings.
51   private val BACKOFF_TIME_IN_MILLIS = 3000L
52   private val NUM_RETRIES = 10
53   private val CHECKPOINT_INTERVAL_MILLIS = 1000L
54
55   @Override
56   def initialize(shardId: String) = {
57     println("Initializing record processor for shard: " + shardId)
58     this.kinesisShardId = shardId
59   }
60
61   private val printData: (Array[Byte] => Unit) =
62     if (config.streamDataType == "string") printDataString
63     else if (config.streamDataType == "thrift") printDataThrift
64     else throw new RuntimeException(
65         "data-type configuration must be 'string' or 'thrift'.")
66
67   @Override
```

```
68    def processRecords(records: List[Record],
69        checkpointer: IRecordProcessorCheckpointer) = {
70      println(s"Processing ${records.size} records from $kinesisShardId")
71      processRecordsWithRetries(records)
72
73      if (System.currentTimeMillis() > nextCheckpointTimeInMillis) {
74        checkpoint(checkpointer)
75        nextCheckpointTimeInMillis =
76          System.currentTimeMillis + CHECKPOINT_INTERVAL_MILLIS
77      }
78    }
79
80    private def processRecordsWithRetries(records: List[Record]) = {
81      for (record <- records) {
82        try {
83          println(s"Sequence number: ${record.getSequenceNumber}")
84          printData(record.getData.array)
85          println(s"Partition key: ${record.getPartitionKey}")
86        } catch {
87          case t: Throwable =>
88            println(s"Caught throwable while processing record $record")
89            println(t)
90        }
91      }
92    }
93
94    private def printDataString(data: Array[Byte]) =
95      println("data: " + new String(data))
96
97    private def printDataThrift(data: Array[Byte]) = {
98      var deserializedData: generated.StreamData = new generated.StreamData()
99      this.synchronized {
100       thriftDeserializer.deserialize(deserializedData, data)
101     }
102     println("data: " + deserializedData.toString)
103   }
104
105   @Override
106   def shutdown(checkpointer: IRecordProcessorCheckpointer,
107       reason: ShutdownReason) = {
108     println(s"Shutting down record processor for shard: $kinesisShardId")
109     if (reason == ShutdownReason.TERMINATE) {
110       checkpoint(checkpointer)
111     }
112   }
113
114   private def checkpoint(checkpointer: IRecordProcessorCheckpointer) = {
115     println(s"Checkpointing shard $kinesisShardId")
116     breakable { for (i <- 0 to NUM_RETRIES-1) {
117       try {
118         checkpointer.checkpoint()
```

```
119           break
120         } catch {
121           case se: ShutdownException =>
122             println("Caught shutdown exception, skipping checkpoint.", se)
123           case e: ThrottlingException =>
124             if (i >= (NUM_RETRIES - 1)) {
125               println(s"Checkpoint failed after ${i+1} attempts.", e)
126             } else {
127               println(s"Transient issue when checkpointing - attempt ${i+1} of "
128                 + NUM_RETRIES, e)
129             }
130           case e: InvalidStateException =>
131             println("Cannot save checkpoint to the DynamoDB table used by " +
132               "the Amazon Kinesis Client Library.", e)
133         }
134         Thread.sleep(BACKOFF_TIME_IN_MILLIS)
135       }
136     } }
137 }
```

2 scala-stream-collector

2.1 ScalaCollectorApp.scala

```
1  /*
2   * Copyright (c) 2013-2014 Snowplow Analytics Ltd. All rights reserved.
3   *
4   * This program is licensed to you under the Apache License Version 2.0, and
5   * you may not use this file except in compliance with the Apache License
6   * Version 2.0.  You may obtain a copy of the Apache License Version 2.0 at
7   * http://www.apache.org/licenses/LICENSE-2.0.
8   *
9   * Unless required by applicable law or agreed to in writing, software
10  * distributed under the Apache License Version 2.0 is distributed on an "AS
11  * IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
12  * implied.  See the Apache License Version 2.0 for the specific language
13  * governing permissions and limitations there under.
14  */
15
16 package com.snowplowanalytics.snowplow.collectors.scalastream
17
18 // Snowplow
19 import sinks._
20
21 // Akka and Spray
22 import akka.actor.{ActorSystem, Props}
23 import akka.io.IO
24 import spray.can.Http
```

```
25
26 // Java
27 import java.io.File
28
29 // Argot
30 import org.clapper.argot._
31
32 // Config
33 import com.typesafe.config.{ConfigFactory, Config, ConfigException}
34
35 // Logging.
36 import org.slf4j.LoggerFactory
37
38 // Main entry point of the Scala collector.
39 object ScalaCollector extends App {
40   lazy val log = LoggerFactory.getLogger(getClass())
41   import log.{error, debug, info, trace}
42
43   import ArgotConverters._ // Argument specifications
44
45   val parser = new ArgotParser(
46     programName = generated.Settings.name,
47     compactUsage = true,
48     preUsage = Some("%s: Version %s. Copyright (c) 2013, %s.".format(
49       generated.Settings.name,
50       generated.Settings.version,
51       generated.Settings.organization)
52     )
53   )
54
55   // Optional config argument
56   val config = parser.option[Config](List("config"), "filename",
57     "Configuration file. Defaults to \"resources/application.conf\" " +
58       "(within .jar) if not set") { (c, opt) =>
59     val file = new File(c)
60     if (file.exists) {
61       ConfigFactory.parseFile(file)
62     } else {
63       parser.usage("Configuration file \"%s\" does not exist".format(c))
64       ConfigFactory.empty()
65     }
66   }
67   parser.parse(args)
68
69   val rawConf = config.value.getOrElse(ConfigFactory.load("application"))
70   implicit val system = ActorSystem.create("scala-stream-collector", rawConf)
71   val collectorConfig = new CollectorConfig(rawConf)
72   val sink = collectorConfig.sinkEnabled match {
73     case Sink.Kinesis => new KinesisSink(collectorConfig)
74     case Sink.Stdout => new StdoutSink
75   }
```

```
76
77    // The handler actor replies to incoming HttpRequests.
78    val handler = system.actorOf(
79      Props(classOf[CollectorServiceActor], collectorConfig, sink),
80      name = "handler"
81    )
82
83    IO(Http) ! Http.Bind(handler,
84      interface=collectorConfig.interface, port=collectorConfig.port)
85  }
86  // Return Options from the configuration.
87  object Helper {
88    implicit class RichConfig(val underlying: Config) extends AnyVal {
89      def getOptionalString(path: String): Option[String] = try {
90        Some(underlying.getString(path))
91      } catch {
92        case e: ConfigException.Missing => None
93      }
94    }
95  }
96
97  // Instead of comparing strings and validating every time
98  // the sink is accessed, validate the string here and
99  // store this enumeration.
100 object Sink extends Enumeration {
101   type Sink = Value
102   val Kinesis, Stdout, Test = Value
103 }
104
105 // Rigidly load the configuration file here to error when
106 // the collector process starts rather than later.
107 class CollectorConfig(config: Config) {
108   import Helper.RichConfig
109
110   private val collector = config.getConfig("collector")
111   val interface = collector.getString("interface")
112   val port = collector.getInt("port")
113   val production = collector.getBoolean("production")
114
115   private val p3p = collector.getConfig("p3p")
116   val p3pPolicyRef = p3p.getString("policyref")
117   val p3pCP = p3p.getString("CP")
118
119   private val cookie = collector.getConfig("cookie")
120   val cookieExpiration = cookie.getMilliseconds("expiration")
121   var cookieDomain = cookie.getOptionalString("domain")
122
123   private val sink = collector.getConfig("sink")
124   val sinkEnabled = sink.getString("enabled") match {
125     case "kinesis" => Sink.Kinesis
126     case "stdout" => Sink.Stdout
```

```
127     case "test" => Sink.Test
128     case _ => throw new RuntimeException("collector.sink.enabled unknown.")
129   }
130
131   private val kinesis = sink.getConfig("kinesis")
132   private val aws = kinesis.getConfig("aws")
133   val awsAccessKey = aws.getString("access-key")
134   val awsSecretKey = aws.getString("secret-key")
135   private val stream = kinesis.getConfig("stream")
136   val streamName = stream.getString("name")
137   val streamSize = stream.getInt("size")
138 }
```

2.2 CollectorService.scala

```
1  /*
2   * Copyright (c) 2013-2014 Snowplow Analytics Ltd. All rights reserved.
3   *
4   * This program is licensed to you under the Apache License Version 2.0, and
5   * you may not use this file except in compliance with the Apache License
6   * Version 2.0.  You may obtain a copy of the Apache License Version 2.0 at
7   * http://www.apache.org/licenses/LICENSE-2.0.
8   *
9   * Unless required by applicable law or agreed to in writing, software
10  * distributed under the Apache License Version 2.0 is distributed on an "AS
11  * IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
12  * implied.  See the Apache License Version 2.0 for the specific language
13  * governing permissions and limitations there under.
14  */
15
16 package com.snowplowanalytics.snowplow.collectors.scalastream
17
18 // Snowplow
19 import sinks._
20
21 // Akka
22 import akka.actor.{Actor, ActorRefFactory}
23 import akka.pattern.ask
24 import akka.util.Timeout
25
26 // Spray
27 import spray.http.{Uri, Timedout, HttpRequest}
28 import spray.routing.HttpService
29
30 // Scala
31 import scala.concurrent.duration._
32
33 // Actor accepting Http requests for the Scala collector.
34 class CollectorServiceActor(collectorConfig: CollectorConfig,
```

```
35      sink: AbstractSink) extends Actor with HttpService {
36    implicit val timeout: Timeout = 1.second // For the actor 'asks'
37    def actorRefFactory = context
38
39    // Deletage responses (content and storing) to the ResponseHandler.
40    private val responseHandler = new ResponseHandler(collectorConfig, sink)
41
42    // Use CollectorService so the same route can be accessed differently
43    // in the testing framework.
44    private val collectorService = new CollectorService(responseHandler, context)
45
46    // Message loop for the Spray service.
47    def receive = handleTimeouts orElse runRoute(collectorService.collectorRoute)
48
49    def handleTimeouts: Receive = {
50      case Timedout(_) => sender ! responseHandler.timeout
51    }
52  }
53
54  // Store the route in CollectorService to be accessed from
55  // both CollectorServiceActor and from the testing framework.
56  class CollectorService(
57      responseHandler: ResponseHandler,
58      context: ActorRefFactory) extends HttpService {
59    def actorRefFactory = context
60    val collectorRoute = {
61      get {
62        path("i") {
63          optionalCookie("sp") { reqCookie =>
64            optionalHeaderValueByName("User-Agent") { userAgent =>
65              optionalHeaderValueByName("Referer") { refererURI =>
66                headerValueByName("Raw-Request-URI") { rawRequest =>
67                  hostName { host =>
68                    clientIP { ip =>
69                      requestInstance{ request =>
70                        complete(
71                          responseHandler.cookie(
72                            Option(Uri(rawRequest).query.toString).filter(
73                              _.trim.nonEmpty
74                            ).getOrElse(null),
75                            reqCookie,
76                            userAgent,
77                            host,
78                            ip.toString,
79                            request,
80                            refererURI
81                          )._1
82                        )
83                      }
84                    }
85                  }
```

```
86                  }
87                }
88              }
89            }
90          }
91        }~
92        complete(responseHandler.notFound)
93      }
94    }
```

2.3 ResponseHandler.scala

```
1  /*
2   * Copyright (c) 2013-2014 Snowplow Analytics Ltd. All rights reserved.
3   *
4   * This program is licensed to you under the Apache License Version 2.0, and
5   * you may not use this file except in compliance with the Apache License
6   * Version 2.0.  You may obtain a copy of the Apache License Version 2.0 at
7   * http://www.apache.org/licenses/LICENSE-2.0.
8   *
9   * Unless required by applicable law or agreed to in writing, software
10  * distributed under the Apache License Version 2.0 is distributed on an "AS
11  * IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
12  * implied.  See the Apache License Version 2.0 for the specific language
13  * governing permissions and limitations there under.
14  */
15
16 package com.snowplowanalytics.snowplow.collectors
17 package scalastream
18
19 // Snowplow
20 import generated._
21 import thrift._
22 import sinks._
23
24 // Java
25 import java.nio.ByteBuffer
26 import java.util.UUID
27
28 // Apache commons
29 import org.apache.commons.codec.binary.Base64
30
31 // Spray
32 import spray.http.{DateTime, HttpRequest, HttpResponse, HttpEntity, HttpCookie}
33 import spray.http.HttpHeaders.{
34   `Set-Cookie`,
35   `Remote-Address`,
36   `Raw-Request-URI`,
37   RawHeader
```

```
38 }
39 import spray.http.MediaTypes.`image/gif`
40
41 // Typesafe config
42 import com.typesafe.config.Config
43
44 // Java conversions
45 import scala.collection.JavaConversions._
46
47 // Contains an invisible pixel to return for `/i` requests.
48 object ResponseHandler {
49   val pixel = Base64.decodeBase64(
50     "R0lGODlhAQABAAAAACH5BAEKAAEALAAAAAABAAEAAAICTAEAOw=="
51   )
52 }
53
54 // Receive requests and store data into an output sink.
55 class ResponseHandler(config: CollectorConfig, sink: AbstractSink) {
56   // When `/i` is requested, this is called and stores an event in the
57   // Kinisis sink and returns an invisible pixel with a cookie.
58   def cookie(queryParams: String, requestCookie: Option[HttpCookie],
59       userAgent: Option[String], hostname: String, ip: String,
60       request: HttpRequest, refererUri: Option[String]):
61       (HttpResponse, Array[Byte]) = {
62     // Use the same UUID if the request cookie contains `sp`.
63     val networkUserId: String =
64       if (requestCookie.isDefined) requestCookie.get.content
65       else UUID.randomUUID.toString()
66
67     // Construct an event object from the request.
68     val timestamp: Long = System.currentTimeMillis
69
70     val payload = new TrackerPayload(
71       PayloadProtocol.Http,
72       PayloadFormat.HttpGet,
73       queryParams
74     )
75
76     val event = new SnowplowRawEvent(
77       timestamp,
78       s"${generated.Settings.shortName}-${generated.Settings.version}-${config.sinkEnabled}",
79       "UTF-8",
80       ip
81     )
82
83     event.payload = payload
84     event.hostname = hostname
85     if (userAgent.isDefined) event.userAgent = userAgent.get
86     if (refererUri.isDefined) event.refererUri = refererUri.get
87     event.headers = request.headers.flatMap {
88       case _: `Remote-Address` | _: `Raw-Request-URI` => None
```

```
89        case other => Some(other.toString)
90      }
91      event.networkUserId = networkUserId
92
93      // Only the test sink responds with the serialized object.
94      val sinkResponse = sink.storeRawEvent(event, ip)
95
96      // Build the HTTP response.
97      val responseCookie = HttpCookie(
98        "sp", networkUserId,
99        expires=Some(DateTime.now+config.cookieExpiration),
100       domain=config.cookieDomain
101     )
102     val policyRef = config.p3pPolicyRef
103     val CP = config.p3pCP
104     val headers = List(
105       RawHeader("P3P", s"""policyref="${policyRef}", CP="${CP}""""),
106       `Set-Cookie`(responseCookie)
107     )
108     val httpResponse = HttpResponse(
109       entity = HttpEntity(`image/gif`, ResponseHandler.pixel)
110     ).withHeaders(headers)
111     (httpResponse, sinkResponse)
112   }
113
114   def notFound = HttpResponse(status = 404, entity = "404 Not found")
115   def timeout = HttpResponse(status = 500, entity = s"Request timed out.")
116 }
```

2.4 sinks/KinesisSink.scala

```
1  /*
2   * Copyright (c) 2013-2014 Snowplow Analytics Ltd. All rights reserved.
3   *
4   * This program is licensed to you under the Apache License Version 2.0,
5   * and you may not use this file except in compliance with the Apache License Version
        2.0.
6   * You may obtain a copy of the Apache License Version 2.0 at
        http://www.apache.org/licenses/LICENSE-2.0.
7   *
8   * Unless required by applicable law or agreed to in writing,
9   * software distributed under the Apache License Version 2.0 is distributed on an
10  * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
        implied.
11  * See the Apache License Version 2.0 for the specific language governing permissions
        and limitations there under.
12  */
13
14 package com.snowplowanalytics.snowplow.collectors
```

```
15 package scalastream
16 package sinks
17
18 // Snowplow
19 import scalastream._
20 import thrift.SnowplowRawEvent
21
22 // Java
23 import java.nio.ByteBuffer
24
25 // Amazon
26 import com.amazonaws.AmazonServiceException
27 import com.amazonaws.auth.{
28   BasicAWSCredentials,
29   ClasspathPropertiesFileCredentialsProvider
30 }
31
32 // Scalazon (for Kinesis interaction)
33 import io.github.cloudify.scala.aws.kinesis.Client
34 import io.github.cloudify.scala.aws.kinesis.Client.ImplicitExecution._
35 import io.github.cloudify.scala.aws.kinesis.Definitions.{
36   Stream,
37   PutResult,
38   Record
39 }
40 import io.github.cloudify.scala.aws.kinesis.KinesisDsl._
41
42 // Config
43 import com.typesafe.config.Config
44
45 // Concurrent libraries
46 import scala.concurrent.{Future,Await,TimeoutException}
47 import scala.concurrent.ExecutionContext.Implicits.global
48 import scala.concurrent.duration._
49
50 // Logging
51 import org.slf4j.LoggerFactory
52
53 // Mutable data structures
54 import scala.collection.mutable.StringBuilder
55 import scala.collection.mutable.MutableList
56
57 /**
58  * Kinesis Sink for the Scala collector.
59  */
60 class KinesisSink(config: CollectorConfig) extends AbstractSink {
61   private lazy val log = LoggerFactory.getLogger(getClass())
62   import log.{error, debug, info, trace}
63
64   // Create a Kinesis client for stream interactions.
65   private implicit val kinesis = createKinesisClient
```

```
66
67    // The output stream for enriched events.
68    private val enrichedStream = createAndLoadStream()
69
70    // Checks if a stream exists.
71    def streamExists(name: String, timeout: Int = 60): Boolean = {
72      val streamListFuture = for {
73        s <- Kinesis.streams.list
74      } yield s
75      val streamList: Iterable[String] =
76        Await.result(streamListFuture, Duration(timeout, SECONDS))
77      for (streamStr <- streamList) {
78        if (streamStr == name) {
79          info(s"Stream $name exists.")
80          return true
81        }
82      }
83
84      info(s"Stream $name doesn't exist.")
85      false
86    }
87
88    // Creates a new stream if one doesn't exist.
89    def createAndLoadStream(timeout: Int = 60): Stream = {
90      val name = config.streamName
91      val size = config.streamSize
92
93      if (streamExists(name)) {
94        Kinesis.stream(name)
95      } else {
96        info(s"Creating stream $name of size $size.")
97        val createStream = for {
98          s <- Kinesis.streams.create(name)
99        } yield s
100
101       try {
102         val stream = Await.result(createStream, Duration(timeout, SECONDS))
103
104         info(s"Successfully created stream $name. Waiting until it's active.")
105         Await.result(stream.waitActive.retrying(timeout),
106           Duration(timeout, SECONDS))
107
108         info(s"Stream $name active.")
109
110         stream
111       } catch {
112         case _: TimeoutException =>
113           throw new RuntimeException("Error: Timed out.")
114       }
115     }
116   }
```

```
117
118   /**
119    * Creates a new Kinesis client from provided AWS access key and secret
120    * key. If both are set to "cpf", then authenticate using the classpath
121    * properties file.
122    *
123    * @return the initialized AmazonKinesisClient
124    */
125   private def createKinesisClient: Client = {
126     val accessKey = config.awsAccessKey
127     val secretKey = config.awsSecretKey
128     if (isCpf(accessKey) && isCpf(secretKey)) {
129       Client.fromCredentials(new ClasspathPropertiesFileCredentialsProvider())
130     } else if (isCpf(accessKey) || isCpf(secretKey)) {
131       throw new RuntimeException("access-key and secret-key must both be set to 'cpf',
              or neither of them")
132     } else {
133       Client.fromCredentials(accessKey, secretKey)
134     }
135   }
136
137   def storeRawEvent(event: SnowplowRawEvent, key: String): Array[Byte] = {
138     info(s"Writing Thrift record to Kinesis: ${event.toString}")
139     val putData = for {
140       p <- enrichedStream.put(
141         ByteBuffer.wrap(serializeEvent(event)),
142         key
143       )
144     } yield p
145     val result = Await.result(putData, Duration(60, SECONDS))
146     info(s"Writing successful.")
147     info(s"  + ShardId: ${result.shardId}")
148     info(s"  + SequenceNumber: ${result.sequenceNumber}")
149     null
150   }
151
152   /**
153    * Is the access/secret key set to the special value "cpf" i.e. use
154    * the classpath properties file for credentials.
155    *
156    * @param key The key to check
157    * @return true if key is cpf, false otherwise
158    */
159   private def isCpf(key: String): Boolean = (key == "cpf")
160 }
```

2.5 sinks/StdoutSink.scala

```
1 /*
```

```
2  * Copyright (c) 2013-2014 Snowplow Analytics Ltd.
3  * All rights reserved.
4  *
5  * This program is licensed to you under the Apache License Version 2.0,
6  * and you may not use this file except in compliance with the Apache
7  * License Version 2.0.
8  * You may obtain a copy of the Apache License Version 2.0 at
9  * http://www.apache.org/licenses/LICENSE-2.0.
10 *
11 * Unless required by applicable law or agreed to in writing,
12 * software distributed under the Apache License Version 2.0 is distributed
13 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
14 * either express or implied.
15 *
16 * See the Apache License Version 2.0 for the specific language
17 * governing permissions and limitations there under.
18 */
19
20 package com.snowplowanalytics.snowplow.collectors
21 package scalastream
22 package sinks
23
24 // Snowplow
25 import scalastream._
26 import thrift.SnowplowRawEvent
27
28 import java.nio.ByteBuffer
29 import org.apache.thrift.TSerializer
30 import com.typesafe.config.Config
31 import org.apache.commons.codec.binary.Base64
32
33 class StdoutSink extends AbstractSink {
34   // Print a Base64-encoded event.
35   def storeRawEvent(event: SnowplowRawEvent, key: String) = {
36     println(Base64.encodeBase64String(serializeEvent(event)))
37     null
38   }
39 }
```

2.6 CollectorServiceSpec.scala

```
1 /*
2  * Copyright (c) 2013-2014 Snowplow Analytics Ltd. All rights reserved.
3  *
4  * This program is licensed to you under the Apache License Version 2.0, and
5  * you may not use this file except in compliance with the Apache License
6  * Version 2.0.  You may obtain a copy of the Apache License Version 2.0 at
7  * http://www.apache.org/licenses/LICENSE-2.0.
8  *
```

```
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the Apache License Version 2.0 is distributed on an "AS
 * IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
 * implied.  See the Apache License Version 2.0 for the specific language
 * governing permissions and limitations there under.
 */

package com.snowplowanalytics.snowplow.collectors
package scalastream

// Snowplow
import sinks._
import thrift.{
  PayloadProtocol,
  PayloadFormat,
  SnowplowRawEvent
}

// Akka
import akka.actor.{ActorSystem, Props}

// specs2 and spray testing libraries
import org.specs2.matcher.AnyMatchers
import org.specs2.mutable.Specification
import org.specs2.specification.{Scope,Fragments}
import spray.testkit.Specs2RouteTest

// Spray
import spray.http.{DateTime,HttpHeader,HttpRequest,HttpCookie}
import spray.http.HttpHeaders.{
  Cookie,
  `Set-Cookie`,
  `Remote-Address`,
  `Raw-Request-URI`
}

// Config
import com.typesafe.config.{ConfigFactory,Config,ConfigException}

// Thrift
import org.apache.thrift.TDeserializer

// Scala
import scala.collection.mutable.MutableList

class CollectorServiceSpec extends Specification with Specs2RouteTest with
      AnyMatchers {
    val testConf: Config = ConfigFactory.parseString("""
collector {
  interface = "0.0.0.0"
  port = 8080
```

```
60
61    production = true
62
63    p3p {
64      policyref = "/w3c/p3p.xml"
65      CP = "NOI DSP COR NID PSA OUR IND COM NAV STA"
66    }
67
68    cookie {
69      expiration = 365 days
70      domain = "test-domain.com"
71    }
72
73    sink {
74      enabled = "test"
75
76      kinesis {
77        aws {
78          access-key: "cpf"
79          secret-key: "cpf"
80        }
81        stream {
82          name: "snowplow_collector_example"
83          size: 1
84        }
85      }
86    }
87  }
88  """)
89    val collectorConfig = new CollectorConfig(testConf)
90    val sink = new TestSink
91    val responseHandler = new ResponseHandler(collectorConfig, sink)
92    val collectorService = new CollectorService(responseHandler, system)
93    val thriftDeserializer = new TDeserializer
94
95    // By default, spray will always add Remote-Address to every request
96    // when running with the `spray.can.server.remote-address-header`
97    // option. However, the testing does not read this option and a
98    // remote address always needs to be set.
99    def CollectorGet(uri: String, cookie: Option[`HttpCookie`] = None,
100       remoteAddr: String = "127.0.0.1") = {
101     val headers: MutableList[HttpHeader] =
102       MutableList(`Remote-Address`(remoteAddr),`Raw-Request-URI`(uri))
103     if (cookie.isDefined) headers += `Cookie`(cookie.get)
104     Get(uri).withHeaders(headers.toList)
105   }
106
107   "Snowplow's Scala collector" should {
108     "return an invisible pixel." in {
109       CollectorGet("/i") ~> collectorService.collectorRoute ~> check {
110         responseAs[Array[Byte]] === ResponseHandler.pixel
```

```
111       }
112     }
113     "return a cookie expiring at the correct time." in {
114       CollectorGet("/i") ~> collectorService.collectorRoute ~> check {
115         headers must not be empty
116
117         val httpCookies: List[HttpCookie] = headers.collect {
118           case `Set-Cookie`(hc) => hc
119         }
120         httpCookies must not be empty
121
122         // Assume we only return a single cookie.
123         // If the collector is modified to return multiple cookies,
124         // this will need to be changed.
125         val httpCookie = httpCookies(0)
126
127         httpCookie.name must be("sp")
128         httpCookie.domain must beSome
129         httpCookie.domain.get must be(collectorConfig.cookieDomain.get)
130         httpCookie.expires must beSome
131         val expiration = httpCookie.expires.get
132         val offset = expiration.clicks - collectorConfig.cookieExpiration -
133           DateTime.now.clicks
134         offset.asInstanceOf[Int] must beCloseTo(0, 2000) // 1000 ms window.
135       }
136     }
137     "return the same cookie as passed in." in {
138       CollectorGet("/i", Some(HttpCookie("sp", "UUID_Test"))) ~>
139           collectorService.collectorRoute ~> check {
140         val httpCookies: List[HttpCookie] = headers.collect {
141           case `Set-Cookie`(hc) => hc
142         }
143         // Assume we only return a single cookie.
144         // If the collector is modified to return multiple cookies,
145         // this will need to be changed.
146         val httpCookie = httpCookies(0)
147
148         httpCookie.content must beEqualTo("UUID_Test")
149       }
150     }
151     "return a P3P header." in {
152       CollectorGet("/i") ~> collectorService.collectorRoute ~> check {
153         val p3pHeaders = headers.filter {
154           h => h.name.equals("P3P")
155         }
156         p3pHeaders.size must beEqualTo(1)
157         val p3pHeader = p3pHeaders(0)
158
159         val policyRef = collectorConfig.p3pPolicyRef
160         val CP = collectorConfig.p3pCP
161         p3pHeader.value must beEqualTo(
```

```
162             s"""policyref="${policyRef}", CP="${CP}"""")
163         }
164       }
165       "store the expected event as a serialized Thrift object in the enabled sink" in {
166         val payloadData = "param1=val1&param2=val2"
167         val storedRecordBytes = responseHandler.cookie(payloadData, None,
168           None, "localhost", "127.0.0.1", new HttpRequest(), None)._2
169
170         val storedEvent = new SnowplowRawEvent
171         this.synchronized {
172           thriftDeserializer.deserialize(storedEvent, storedRecordBytes)
173         }
174
175         storedEvent.timestamp must beCloseTo(DateTime.now.clicks, 1000)
176         storedEvent.encoding must beEqualTo("UTF-8")
177         storedEvent.ipAddress must beEqualTo("127.0.0.1")
178         storedEvent.payload.protocol must beEqualTo(PayloadProtocol.Http)
179         storedEvent.payload.format must beEqualTo(PayloadFormat.HttpGet)
180         storedEvent.payload.data must beEqualTo(payloadData)
181       }
182     }
183 }
```

3 scala-kinesis-enrich

3.1 KinesisEnrichApp.scala

```
1  /*
2  * Copyright (c) 2013-2014 Snowplow Analytics Ltd.
3  * All rights reserved.
4  *
5  * This program is licensed to you under the Apache License Version 2.0,
6  * and you may not use this file except in compliance with the Apache
7  * License Version 2.0.
8  * You may obtain a copy of the Apache License Version 2.0 at
9  * http://www.apache.org/licenses/LICENSE-2.0.
10 *
11 * Unless required by applicable law or agreed to in writing,
12 * software distributed under the Apache License Version 2.0 is distributed
13 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
14 * either express or implied.
15 *
16 * See the Apache License Version 2.0 for the specific language
17 * governing permissions and limitations there under.
18 */
19
20 package com.snowplowanalytics.snowplow.enrich.kinesis
21
```

```
22 // Snowplow
23 import sources._
24 import sinks._
25
26 // Config
27 import com.typesafe.config.{Config,ConfigFactory}
28
29 // Argot
30 import org.clapper.argot.ArgotParser
31
32 // Java
33 import java.io.File
34
35 // The enrichment process takes input SnowplowRawEvent objects from
36 // an input source out outputs enriched objects to a sink,
37 // as defined in the following enumerations.
38 object Source extends Enumeration {
39   type Source = Value
40   val Kinesis, Stdin, Test = Value
41 }
42 object Sink extends Enumeration {
43   type Sink = Value
44   val Kinesis, Stdouterr, Test = Value
45 }
46
47 // The main entry point of the Scala Kinesis Enricher.
48 object KinesisEnrichApp extends App {
49   val parser = new ArgotParser(
50     programName = generated.Settings.name,
51     compactUsage = true,
52     preUsage = Some("%s: Version %s. Copyright (c) 2013, %s.".format(
53       generated.Settings.name,
54       generated.Settings.version,
55       generated.Settings.organization)
56     )
57   )
58
59   // Optional config argument
60   val config = parser.option[Config](
61       List("config"), "filename", """
62         |Configuration file. Defaults to \"resources/default.conf\"
63         |(within .jar) if not set""".stripMargin) {
64     (c, opt) =>
65       val file = new File(c)
66       if (file.exists) {
67         ConfigFactory.parseFile(file)
68       } else {
69         parser.usage("Configuration file \"%s\" does not exist".format(c))
70         ConfigFactory.empty()
71       }
72   }
```

```
73
74    parser.parse(args)
75    val kinesisEnrichConfig = new KinesisEnrichConfig(
76      config.value.getOrElse(ConfigFactory.load("default"))
77    )
78
79    val source = kinesisEnrichConfig.source match {
80      case Source.Kinesis => new KinesisSource(kinesisEnrichConfig)
81      case Source.Stdin => new StdinSource(kinesisEnrichConfig)
82    }
83    source.run
84  }
85
86  // Rigidly load the configuration file here to error when
87  // the enrichment process starts rather than later.
88  class KinesisEnrichConfig(config: Config) {
89    private val enrich = config.resolve.getConfig("enrich")
90
91    val source = enrich.getString("source") match {
92      case "kinesis" => Source.Kinesis
93      case "stdin" => Source.Stdin
94      case "test" => Source.Test
95      case _ => throw new RuntimeException("enrich.source unknown.")
96    }
97
98    val sink = enrich.getString("sink") match {
99      case "kinesis" => Sink.Kinesis
100     case "stdouterr" => Sink.Stdouterr
101     case "test" => Sink.Test
102     case _ => throw new RuntimeException("enrich.sink unknown.")
103   }
104
105   private val aws = enrich.getConfig("aws")
106   val accessKey = aws.getString("access-key")
107   val secretKey = aws.getString("secret-key")
108
109   private val streams = enrich.getConfig("streams")
110
111   private val inStreams = streams.getConfig("in")
112   val rawInStream = inStreams.getString("raw")
113
114   private val outStreams = streams.getConfig("out")
115   val enrichedOutStream = outStreams.getString("enriched")
116   val enrichedOutStreamShards = outStreams.getInt("enriched_shards")
117   val badOutStream = outStreams.getString("bad")
118   val badOutStreamShards = outStreams.getInt("bad_shards")
119
120   val appName = streams.getString("app-name")
121
122   val initialPosition = streams.getString("initial-position")
123   val streamEndpoint = streams.getString("endpoint")
```

```
124
125   private val enrichments = enrich.getConfig("enrichments")
126   private val geoIp = enrichments.getConfig("geo_ip")
127   val geoIpEnabled = geoIp.getBoolean("enabled")
128   val maxmindFile = new File(geoIp.getString("maxmind_file"))
129
130   private val anonIp = enrichments.getConfig("anon_ip")
131   val anonIpEnabled = anonIp.getBoolean("enabled")
132   val anonOctets = anonIp.getInt("anon_octets")
133 }
```

3.2 sinks/ISink.scala

```
1  /*
2   * Copyright (c) 2013-2014 Snowplow Analytics Ltd.
3   * All rights reserved.
4   *
5   * This program is licensed to you under the Apache License Version 2.0,
6   * and you may not use this file except in compliance with the Apache
7   * License Version 2.0.
8   * You may obtain a copy of the Apache License Version 2.0 at
9   * http://www.apache.org/licenses/LICENSE-2.0.
10  *
11  * Unless required by applicable law or agreed to in writing,
12  * software distributed under the Apache License Version 2.0 is distributed
13  * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
14  * either express or implied.
15  *
16  * See the Apache License Version 2.0 for the specific language
17  * governing permissions and limitations there under.
18  */
19
20 package com.snowplowanalytics.snowplow.enrich
21 package kinesis.sinks
22
23 // Snowplow
24 import common.outputs.CanonicalOutput
25
26 // Amazon
27 import com.amazonaws.auth._
28
29 // Define an interface for all sinks to use to store events.
30 trait ISink {
31   def storeCanonicalOutput(bytes: String, key: String)
32 }
```

3.3 sinks/KinesisSink.scala

```
1  /*
2  * Copyright (c) 2013-2014 Snowplow Analytics Ltd.
3  * All rights reserved.
4  *
5  * This program is licensed to you under the Apache License Version 2.0,
6  * and you may not use this file except in compliance with the Apache
7  * License Version 2.0.
8  * You may obtain a copy of the Apache License Version 2.0 at
9  * http://www.apache.org/licenses/LICENSE-2.0.
10 *
11 * Unless required by applicable law or agreed to in writing,
12 * software distributed under the Apache License Version 2.0 is distributed
13 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
14 * either express or implied.
15 *
16 * See the Apache License Version 2.0 for the specific language
17 * governing permissions and limitations there under.
18 */
19
20 package com.snowplowanalytics.snowplow.enrich
21 package kinesis
22 package sinks
23
24 // Snowplow
25 import com.snowplowanalytics.snowplow.collectors.thrift._
26 import common.outputs.CanonicalOutput
27
28 // Java
29 import java.nio.ByteBuffer
30
31 // Amazon
32 import com.amazonaws.AmazonServiceException
33 import com.amazonaws.auth.AWSCredentialsProvider
34
35 // Scalazon (for Kinesis interaction)
36 import io.github.cloudify.scala.aws.kinesis.Client
37 import io.github.cloudify.scala.aws.kinesis.Client.ImplicitExecution._
38 import io.github.cloudify.scala.aws.kinesis.Definitions.{
39   Stream,
40   PutResult,
41   Record
42 }
43 import io.github.cloudify.scala.aws.kinesis.KinesisDsl._
44
45 // Config
46 import com.typesafe.config.Config
47
48 // Concurrent libraries.
49 import scala.concurrent.{Future,Await,TimeoutException}
50 import scala.concurrent.ExecutionContext.Implicits.global
51 import scala.concurrent.duration._
```

```
52
53 // Logging.
54 import org.slf4j.LoggerFactory
55
56 // Kinesis Sink for Scala enrichment.
57 class KinesisSink(provider: AWSCredentialsProvider,
58     config: KinesisEnrichConfig) extends ISink {
59   private lazy val log = LoggerFactory.getLogger(getClass())
60   import log.{error, debug, info, trace}
61
62   // Create a Kinesis client for stream interactions.
63   private implicit val kinesis = Client.fromCredentials(provider)
64
65   // The output stream for enriched events.
66   private val enrichedStream = createAndLoadStream()
67
68   // Checks if a stream exists.
69   def streamExists(name: String, timeout: Int = 60): Boolean = {
70     val streamListFuture = for {
71       s <- Kinesis.streams.list
72     } yield s
73     val streamList: Iterable[String] =
74       Await.result(streamListFuture, Duration(timeout, SECONDS))
75     for (streamStr <- streamList) {
76       if (streamStr == name) {
77         info(s"Stream $name exists.")
78         return true
79       }
80     }
81
82     info(s"Stream $name doesn't exist.")
83     false
84   }
85
86   // Creates a new stream if one doesn't exist.
87   def createAndLoadStream(timeout: Int = 60): Stream = {
88     val name = config.enrichedOutStream
89     val size = config.enrichedOutStreamShards
90     if (streamExists(name)) {
91       Kinesis.stream(name)
92     } else {
93       info(s"Creating stream $name of size $size.")
94       val createStream = for {
95         s <- Kinesis.streams.create(name)
96       } yield s
97
98       try {
99         val stream = Await.result(createStream, Duration(timeout, SECONDS))
100
101        info(s"Successfully created stream $name. Waiting until it's active.")
102        Await.result(stream.waitActive.retrying(timeout),
```

```
103             Duration(timeout, SECONDS))
104
105           info(s"Stream $name active.")
106
107           stream
108         } catch {
109           case _: TimeoutException =>
110             throw new RuntimeException("Error: Timed out.")
111         }
112       }
113     }
114
115     // Store successfully validated events in tab delimited form
116     // to the enriched output stream.
117     def storeCanonicalOutput(tabDelimCanonicalOutput: String, key: String) = {
118       val putData = for {
119         p <- enrichedStream.put(
120           ByteBuffer.wrap(tabDelimCanonicalOutput.getBytes),
121           key
122         )
123       } yield p
124       val result = Await.result(putData, Duration(60, SECONDS))
125       info(s"Writing successful.")
126       info(s"  + ShardId: ${result.shardId}")
127       info(s"  + SequenceNumber: ${result.sequenceNumber}")
128     }
129 }
```

3.4 sinks/StdouterrSink.scala

```
1  /*
2   * Copyright (c) 2013-2014 Snowplow Analytics Ltd.
3   * All rights reserved.
4   *
5   * This program is licensed to you under the Apache License Version 2.0,
6   * and you may not use this file except in compliance with the Apache
7   * License Version 2.0.
8   * You may obtain a copy of the Apache License Version 2.0 at
9   * http://www.apache.org/licenses/LICENSE-2.0.
10  *
11  * Unless required by applicable law or agreed to in writing,
12  * software distributed under the Apache License Version 2.0 is distributed
13  * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
14  * either express or implied.
15  *
16  * See the Apache License Version 2.0 for the specific language
17  * governing permissions and limitations there under.
18  */
19
```

```
20 package com.snowplowanalytics.snowplow.enrich.kinesis
21 package sinks
22
23 // Snowplow events.
24 import com.snowplowanalytics.snowplow.collectors.thrift._
25
26 /**
27  * Stdouterr Sink for Scala enrichment.
28  */
29 class StdouterrSink extends ISink {
30   def storeCanonicalOutput(bytes: String, key: String) {
31     println(bytes)
32   }
33 }
```

3.5 sources/AbstractSource.scala

```
1  /*
2  * Copyright (c) 2013-2014 Snowplow Analytics Ltd.
3  * All rights reserved.
4  *
5  * This program is licensed to you under the Apache License Version 2.0,
6  * and you may not use this file except in compliance with the Apache
7  * License Version 2.0.
8  * You may obtain a copy of the Apache License Version 2.0 at
9  * http://www.apache.org/licenses/LICENSE-2.0.
10 *
11 * Unless required by applicable law or agreed to in writing,
12 * software distributed under the Apache License Version 2.0 is distributed
13 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
14 * either express or implied.
15 *
16 * See the Apache License Version 2.0 for the specific language
17 * governing permissions and limitations there under.
18 */
19
20 package com.snowplowanalytics.snowplow.enrich
21 package kinesis
22 package sources
23
24 // Snowplow
25 import sinks._
26 import com.snowplowanalytics.maxmind.geoip.IpGeo
27 import common.outputs.CanonicalOutput
28 import common.inputs.ThriftLoader
29 import common.MaybeCanonicalInput
30 import common.outputs.CanonicalOutput
31 import common.enrichments.EnrichmentManager
32 import common.enrichments.PrivacyEnrichments.AnonOctets
```

```
33
34 // Amazon
35 import com.amazonaws.auth._
36
37 abstract class AbstractSource(config: KinesisEnrichConfig) {
38   def run
39
40   /**
41    * Fields in our CanonicalOutput which are discarded for legacy
42    * Redshift space reasons
43    */
44   private val DiscardedFields = Array("page_url", "page_referrer")
45
46   // Initialize a kinesis provider to use with a Kinesis source or sink.
47   protected val kinesisProvider = createKinesisProvider
48
49   // Initialize the sink to output enriched events to.
50   protected val sink: ISink = config.sink match {
51     case Sink.Kinesis => new KinesisSink(kinesisProvider, config)
52     case Sink.Stdouterr => new StdouterrSink
53     case Sink.Test => null
54   }
55
56   // Iterate through an enriched CanonicalOutput object and tab separate
57   // the fields to a string.
58   def tabSeparateCanonicalOutput(output: CanonicalOutput): String = {
59     output.getClass.getDeclaredFields
60     .filter { field =>
61       !DiscardedFields.contains(field.getName)
62     }
63     .map{ field =>
64       field.setAccessible(true)
65       Option(field.get(output)).getOrElse("")
66     }.mkString("\t")
67   }
68
69   // Helper method to enrich an event.
70   def enrichEvent(binaryData: Array[Byte]): String = {
71     val canonicalInput = ThriftLoader.toCanonicalInput(
72       new String(binaryData.map(_.toChar))
73     )
74
75     (canonicalInput.toValidationNel) map { (ci: MaybeCanonicalInput) =>
76       if (ci.isDefined) {
77         val ipGeo = new IpGeo(
78           dbFile = config.maxmindFile,
79           memCache = false,
80           lruCache = 20000
81         )
82         val anonOctets =
83           if (!config.anonIpEnabled || config.anonOctets == 0) {
```

```
84              AnonOctets.None
85            } else {
86              AnonOctets(config.anonOctets)
87            }
88          val canonicalOutput = EnrichmentManager.enrichEvent(
89            ipGeo,
90            s"kinesis-${generated.Settings.version}",
91            anonOctets,
92            ci.get
93          )
94          (canonicalOutput.toValidationNel) map { (co: CanonicalOutput) =>
95            val ts = tabSeparateCanonicalOutput(co)
96            if (config.sink != Sink.Test) {
97              sink.storeCanonicalOutput(ts, co.user_ipaddress)
98            } else {
99              return ts
100           }
101           // TODO: Store bad event if canonical output not validated.
102         }
103       } else {
104         // CanonicalInput is None: do nothing
105       }
106       // TODO: Store bad event if canonical input not validated.
107     }
108     return null
109   }
110
111   // Initialize a Kinesis provider with the given credentials.
112   private def createKinesisProvider(): AWSCredentialsProvider = {
113     val a = config.accessKey
114     val s = config.secretKey
115     if (isCpf(a) && isCpf(s)) {
116         new ClasspathPropertiesFileCredentialsProvider()
117     } else if (isCpf(a) || isCpf(s)) {
118       throw new RuntimeException(
119         "access-key and secret-key must both be set to 'cpf', or neither"
120       )
121     } else {
122       new BasicAWSCredentialsProvider(
123         new BasicAWSCredentials(a, s)
124       )
125     }
126   }
127   private def isCpf(key: String): Boolean = (key == "cpf")
128
129   // Wrap BasicAWSCredential objects.
130   class BasicAWSCredentialsProvider(basic: BasicAWSCredentials) extends
131       AWSCredentialsProvider{
132     @Override def getCredentials: AWSCredentials = basic
133     @Override def refresh = {}
134   }
```

```
135 }
```

3.6 sources/KinesisSource.scala

```
1  /*
2  * Copyright (c) 2013-2014 Snowplow Analytics Ltd.
3  * All rights reserved.
4  *
5  * This program is licensed to you under the Apache License Version 2.0,
6  * and you may not use this file except in compliance with the Apache
7  * License Version 2.0.
8  * You may obtain a copy of the Apache License Version 2.0 at
9  * http://www.apache.org/licenses/LICENSE-2.0.
10 *
11 * Unless required by applicable law or agreed to in writing,
12 * software distributed under the Apache License Version 2.0 is distributed
13 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
14 * either express or implied.
15 *
16 * See the Apache License Version 2.0 for the specific language
17 * governing permissions and limitations there under.
18 */
19
20 package com.snowplowanalytics.snowplow
21 package enrich
22 package kinesis
23 package sources
24
25 // Snowplow events and enrichment
26 import sinks._
27 import collectors.thrift.{
28   SnowplowRawEvent,
29   TrackerPayload => ThriftTrackerPayload,
30   PayloadProtocol,
31   PayloadFormat
32 }
33
34 // Java
35 import java.io.{FileInputStream,IOException}
36 import java.net.InetAddress
37 import java.nio.ByteBuffer
38 import java.util.{List,UUID}
39
40 // Amazon
41 import com.amazonaws.auth._
42 import com.amazonaws.AmazonClientException
43 import com.amazonaws.services.kinesis.AmazonKinesisClient
44 import com.amazonaws.services.kinesis.clientlibrary.interfaces._
45 import com.amazonaws.services.kinesis.clientlibrary.exceptions._
```

```
46 import com.amazonaws.services.kinesis.clientlibrary.lib.worker._
47 import com.amazonaws.services.kinesis.clientlibrary.types.ShutdownReason
48 import com.amazonaws.services.kinesis.metrics.impl.NullMetricsFactory
49 import com.amazonaws.services.kinesis.model.Record
50
51 // Scala
52 import scala.util.control.Breaks._
53 import scala.collection.JavaConversions._
54
55 // Thrift
56 import org.apache.thrift.TDeserializer
57
58
59 // Overarching class to read events from Kinesis.
60 class KinesisSource(config: KinesisEnrichConfig)
61     extends AbstractSource(config) {
62   def run {
63     val workerId = InetAddress.getLocalHost().getCanonicalHostName() +
64       ":" + UUID.randomUUID()
65     println("Using workerId: " + workerId)
66
67     val kinesisClientLibConfiguration = new KinesisClientLibConfiguration(
68       config.appName,
69       config.rawInStream,
70       kinesisProvider,
71       workerId
72     ).withInitialPositionInStream(
73       InitialPositionInStream.valueOf(config.initialPosition)
74     )
75
76     println(s"Running: ${config.appName}.")
77     println(s"Processing raw input stream: ${config.rawInStream}")
78
79
80     val rawEventProcessorFactory = new RawEventProcessorFactory(
81       config,
82       sink
83     )
84     val worker = new Worker(
85       rawEventProcessorFactory,
86       kinesisClientLibConfiguration,
87       new NullMetricsFactory()
88     )
89
90     worker.run()
91   }
92
93   // Factory needed by the Amazon Kinesis Consumer library to
94   // create a processor.
95   class RawEventProcessorFactory(config: KinesisEnrichConfig, sink: ISink)
96       extends IRecordProcessorFactory {
```

```
97      @Override
98      def createProcessor: IRecordProcessor = {
99        return new RawEventProcessor(config, sink);
100     }
101   }
102
103   // Process events from a Kinesis stream.
104   class RawEventProcessor(config: KinesisEnrichConfig, sink: ISink)
105       extends IRecordProcessor {
106     private val thriftDeserializer = new TDeserializer()
107
108     private var kinesisShardId: String = _
109     private var nextCheckpointTimeInMillis: Long = _
110
111     // Backoff and retry settings.
112     private val BACKOFF_TIME_IN_MILLIS = 3000L
113     private val NUM_RETRIES = 10
114     private val CHECKPOINT_INTERVAL_MILLIS = 1000L
115
116     @Override
117     def initialize(shardId: String) = {
118       println("Initializing record processor for shard: " + shardId)
119       this.kinesisShardId = shardId
120     }
121
122     @Override
123     def processRecords(records: List[Record],
124         checkpointer: IRecordProcessorCheckpointer) = {
125       println(s"Processing ${records.size} records from $kinesisShardId")
126       processRecordsWithRetries(records)
127
128       if (System.currentTimeMillis() > nextCheckpointTimeInMillis) {
129         checkpoint(checkpointer)
130         nextCheckpointTimeInMillis =
131           System.currentTimeMillis + CHECKPOINT_INTERVAL_MILLIS
132       }
133     }
134
135
136     private def processRecordsWithRetries(records: List[Record]) = {
137       for (record <- records) {
138         try {
139           println(s"Sequence number: ${record.getSequenceNumber}")
140           println(s"Partition key: ${record.getPartitionKey}")
141           enrichEvent(record.getData.array)
142         } catch {
143           case t: Throwable =>
144             println(s"Caught throwable while processing record $record")
145             println(t)
146         }
147       }
```

```
148     }
149
150     @Override
151     def shutdown(checkpointer: IRecordProcessorCheckpointer,
152         reason: ShutdownReason) = {
153       println(s"Shutting down record processor for shard: $kinesisShardId")
154       if (reason == ShutdownReason.TERMINATE) {
155         checkpoint(checkpointer)
156       }
157     }
158
159     private def checkpoint(checkpointer: IRecordProcessorCheckpointer) = {
160       println(s"Checkpointing shard $kinesisShardId")
161       breakable {
162         for (i <- 0 to NUM_RETRIES-1) {
163           try {
164             checkpointer.checkpoint()
165             break
166           } catch {
167             case se: ShutdownException =>
168               println("Caught shutdown exception, skipping checkpoint.", se)
169             case e: ThrottlingException =>
170               if (i >= (NUM_RETRIES - 1)) {
171                 println(s"Checkpoint failed after ${i+1} attempts.", e)
172               } else {
173                 println(s"Transient issue when checkpointing - attempt ${i+1} of "
174                   + NUM_RETRIES, e)
175               }
176             case e: InvalidStateException =>
177               println("Cannot save checkpoint to the DynamoDB table used by " +
178                 "the Amazon Kinesis Client Library.", e)
179           }
180           Thread.sleep(BACKOFF_TIME_IN_MILLIS)
181         }
182       }
183     }
184   }
185 }
```

3.7 sources/StdinSource.scala

```
1  /*
2  * Copyright (c) 2013-2014 Snowplow Analytics Ltd.
3  * All rights reserved.
4  *
5  * This program is licensed to you under the Apache License Version 2.0,
6  * and you may not use this file except in compliance with the Apache
7  * License Version 2.0.
8  * You may obtain a copy of the Apache License Version 2.0 at
```

```
 * http://www.apache.org/licenses/LICENSE-2.0.
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the Apache License Version 2.0 is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
 * either express or implied.
 *
 * See the Apache License Version 2.0 for the specific language
 * governing permissions and limitations there under.
 */

package com.snowplowanalytics.snowplow.enrich.kinesis
package sources

// Java
import java.util.List
import java.nio.ByteBuffer

// Scala
import scala.io
import scala.util.control.Breaks._
import scala.collection.JavaConversions._

// Thrift
import org.apache.thrift.TDeserializer

// Apache commons
import org.apache.commons.codec.binary.Base64

// Decode Base64 Thrift objects from stdin.
class StdinSource(config: KinesisEnrichConfig)
    extends AbstractSource(config) {
  def run = {
    for (ln <- io.Source.stdin.getLines) {
      val bytes = Base64.decodeBase64(ln)
      enrichEvent(bytes)
    }
  }
}
```

3.8 sources/TestSource.scala

```
/*
 * Copyright (c) 2013-2014 Snowplow Analytics Ltd.
 * All rights reserved.
 *
 * This program is licensed to you under the Apache License Version 2.0,
 * and you may not use this file except in compliance with the Apache
 * License Version 2.0.
```

```
 8  * You may obtain a copy of the Apache License Version 2.0 at
 9  * http://www.apache.org/licenses/LICENSE-2.0.
10  *
11  * Unless required by applicable law or agreed to in writing,
12  * software distributed under the Apache License Version 2.0 is distributed
13  * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
14  * either express or implied.
15  *
16  * See the Apache License Version 2.0 for the specific language
17  * governing permissions and limitations there under.
18  */
19
20 package com.snowplowanalytics.snowplow.enrich.kinesis
21 package sources
22
23 // Java
24 import java.util.List
25 import java.nio.ByteBuffer
26
27 // Scala
28 import scala.io
29 import scala.util.control.Breaks._
30 import scala.collection.JavaConversions._
31
32 // Thrift
33 import org.apache.thrift.TDeserializer
34
35 // Apache commons
36 import org.apache.commons.codec.binary.Base64
37
38 // Allow the testing framework to enrich events using the
39 // same methods from AbstractSource as the other sources.
40 class TestSource(config: KinesisEnrichConfig)
41     extends AbstractSource(config) {
42   def run = {
43     throw new RuntimeException("'run' should not be called on TestSource.")
44   }
45
46   def enrich(bytes: Array[Byte]): String = {
47     enrichEvent(bytes)
48   }
49 }
```

3.9 KinesisEnrichSpec.scala

```
1 /*
2  * Copyright (c) 2013-2014 Snowplow Analytics Ltd. All rights reserved.
3  *
4  * This program is licensed to you under the Apache License Version 2.0, and
```

```
5   * you may not use this file except in compliance with the Apache License
6   * Version 2.0.  You may obtain a copy of the Apache License Version 2.0 at
7   * http://www.apache.org/licenses/LICENSE-2.0.
8   *
9   * Unless required by applicable law or agreed to in writing, software
10  * distributed under the Apache License Version 2.0 is distributed on an "AS
11  * IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
12  * implied.  See the Apache License Version 2.0 for the specific language
13  * governing permissions and limitations there under.
14  */
15
16 package com.snowplowanalytics.snowplow
17 package enrich.kinesis
18
19 // Snowplow
20 import sources._
21 import collectors.thrift.{
22   PayloadProtocol,
23   PayloadFormat,
24   SnowplowRawEvent
25 }
26
27 // Commons Codec
28 import org.apache.commons.codec.binary.Base64
29
30 // specs2 testing libraries
31 import org.specs2.matcher.AnyMatchers
32 import org.specs2.mutable.Specification
33 import org.specs2.execute.Result
34 import org.specs2.specification.{Scope,Fragments}
35 import org.specs2.scalaz.ValidationMatchers
36
37 // Config
38 import com.typesafe.config.{ConfigFactory,Config,ConfigException}
39
40 // Thrift
41 import org.apache.thrift.TDeserializer
42
43 class KinesisEnrichSpec extends Specification with AnyMatchers {
44   val config = new KinesisEnrichConfig(ConfigFactory.parseString("""
45 enrich {
46   source = "test"
47   sink= "test"
48
49   aws {
50     access-key: "cpf"
51     secret-key: "cpf"
52   }
53
54   streams {
55     in: {
```

```
56        raw: "SnowplowRaw"
57      }
58      out: {
59        enriched: "SnowplowEnriched"
60        enriched_shards: 1 # Number of shards to use if created.
61        bad: "SnowplowBad" # Not used until #463
62        bad_shards: 1 # Number of shards to use if created.
63      }
64      app-name: SnowplowKinesisEnrich-${enrich.streams.in.raw}
65      initial-position = "TRIM_HORIZON"
66      endpoint: "https://kinesis.us-east-1.amazonaws.com"
67    }
68    enrichments {
69      geo_ip: {
70        enabled: true # false not yet suported
71        maxmind_file: "/tmp/GeoLiteCity.dat"
72      }
73      anon_ip: {
74        enabled: true
75        anon_octets: 1 # Or 2, 3 or 4. 0 is same as enabled: false
76      }
77    }
78  }
79  """))
80
81    val testSource = new TestSource(config)
82
83    "Snowplow's Kinesis enricher" should {
84      "enrich a valid SnowplowRawEvent." in {
85        val eventBytes =
              Base64.decodeBase64("CgABAAABQ5iGqAYLABQAAAAQc3NjLTAuMC4xLVN0ZG91dAsAHgAAAAVVVEYtOA
86        val enrichedEvent = testSource.enrich(eventBytes).split("\t")
87        val expected = Array[String](
88          "", "",
89          "2014-01-16 00:49:58.278",
90          "", "",
91          "com.snowplowanalytics",
92          "[0-9a-z-]*",
93          "", "",
94          "ssc-0.0.1-Stdout",
95          "kinesis-0.0.1-common-0.2.0-SNAPSHOT",
96          "",
97          "127.0.0.x",
98          "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",
99          "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",
100         "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",
101         "", "", "", "", "", "",
102         "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
              Chrome/31.0.1650.63 Safari/537.36",
103         "Chrome 31",
104         "Chrome",
```

```
105           "31.0.1650.63",
106           "Browser",
107           "WEBKIT",
108           "", "", "", "", "", "", "", "", "", "", "", "", "",
109           "Linux",
110           "Linux",
111           "Other",
112           "",
113           "Computer",
114           "0"
115         )
116         enrichedEvent.size must beEqualTo(expected.size)
117         Result.unit(
118           (0 to expected.size-1) foreach { i =>
119             if (i == 6) {
120               enrichedEvent(i) must beMatching(expected(i).r)
121             } else {
122               enrichedEvent(i) must beEqualTo(expected(i))
123             }
124           }
125         )
126       }
127     }
128 }
```