

# Stream processing with R in AWS

AWR, AWR.KMS, AWR.Kinesis (R packages) used in ECS

Gergely Daroczi

@daroczig

July 05, 2017



3,633 contributions in the year before last

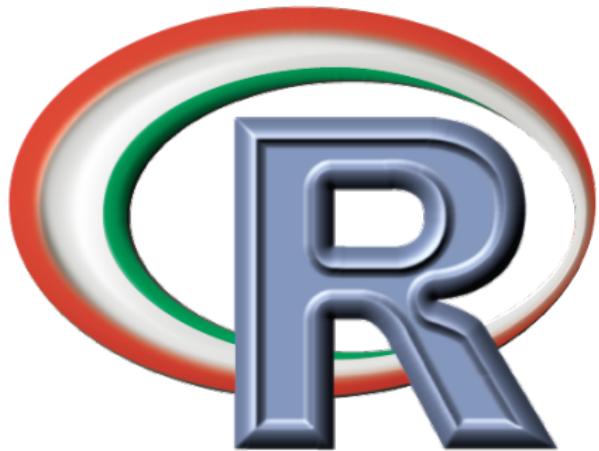
## Lead R Developer



3,470 contributions in the last year

## Director of Analytics





**rapporter**



**CARD.COM**



*rapporTer*



CARD.COM

SYSTEM1



# Stream Processing ... Why R?

useR!2017

User Defined Java Class

Step name: ShortenDate

Classes and code fragments:

- Classes
- Code Snippets
- Input fields
- Getting fields...please wait
- Info fields
- Getting fields...please wait
- Output fields
- Getting fields...please wait

Class code

```
Processor
import java.text.SimpleDateFormat;
import java.util.Date;
import java.text.ParseException;
import java.util.TimeZone;

private SimpleDateFormat df1 = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss.SSS");

private SimpleDateFormat df2 = new SimpleDateFormat("yyyy-MM-dd HH");

public boolean processRow(StepMetaInterface smi, StepDataInterface sdi) throws KettleException, ParseException
{
    Object[] r = getRow();
    if (r == null) {
        setOutputDone();
        return false;
    }

    if (first)
    {
        first = false;
    }

    // It is always safest to call createOutputRow() to ensure that your output row's Object[] is large
    // enough to handle any new fields you are creating in this step.
    r = createOutputRow(r, data.outputRowMeta.size());

    df2.setTimeZone(TimeZone.getTimeZone("America/Los_Angeles"));

    Line #: 0
}
```

Line #: 0

Fields Parameters Info steps Target steps

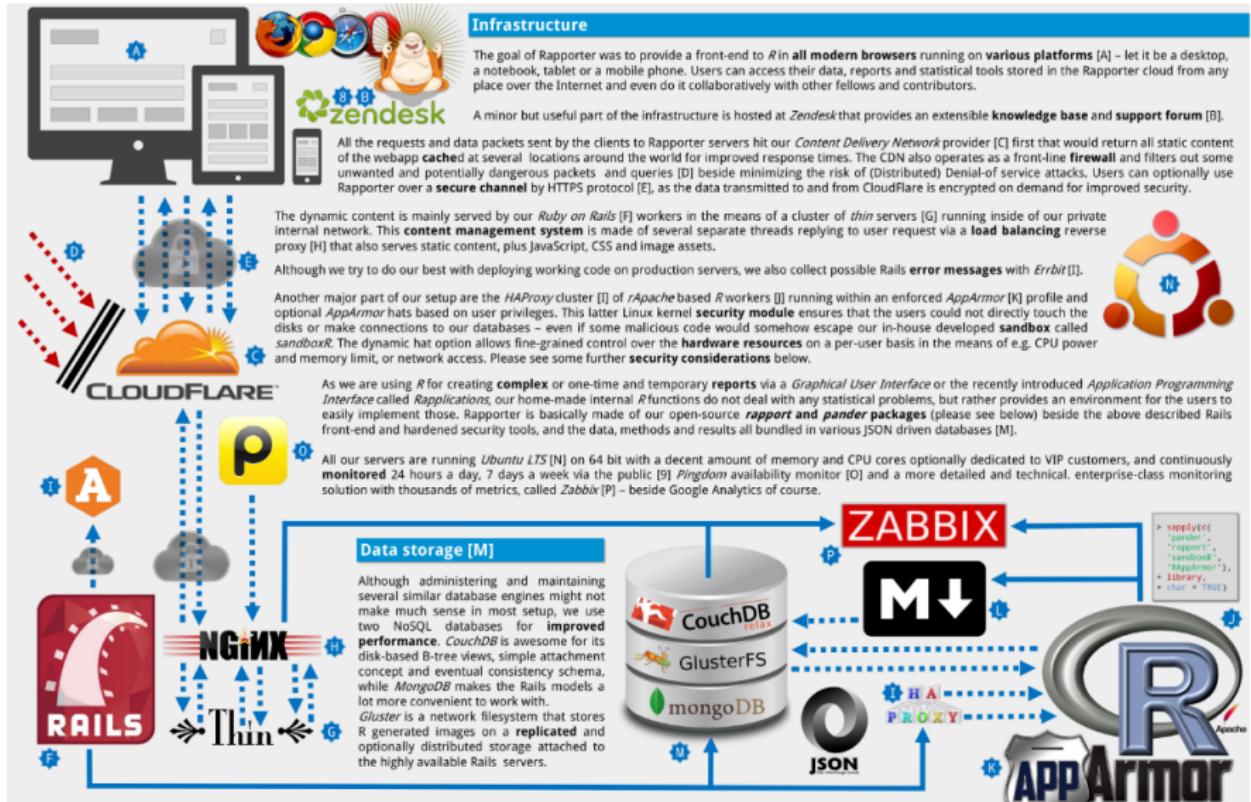
Fields  Clear the result fields?

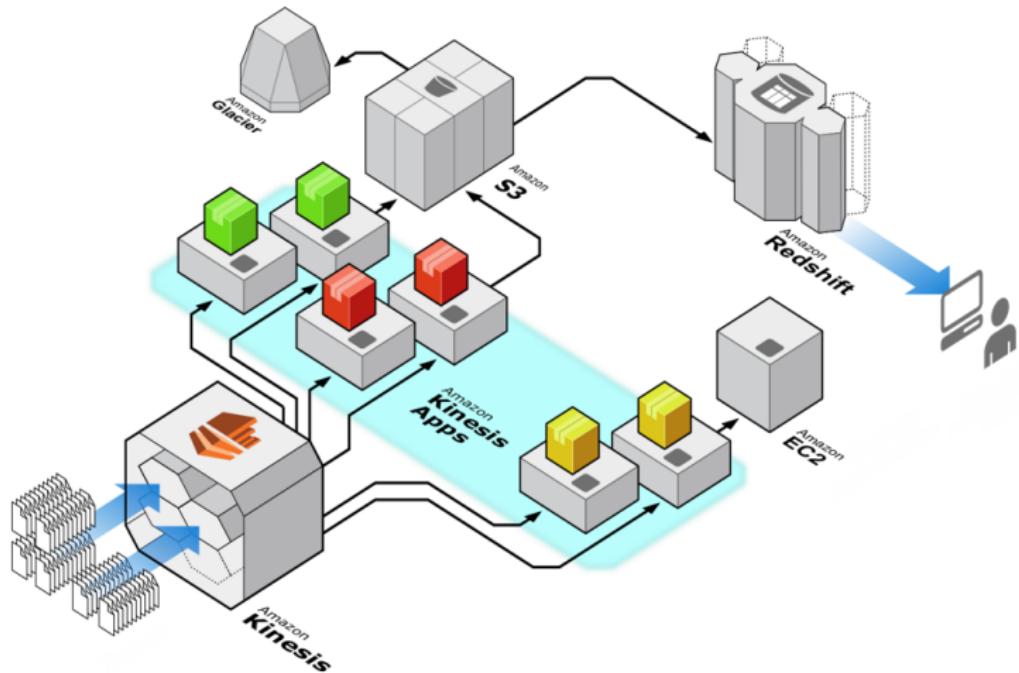
#	Fieldname	Type	Length	Precision
1	RPT_DATE_SHORT	String		

Help OK Cancel Test class

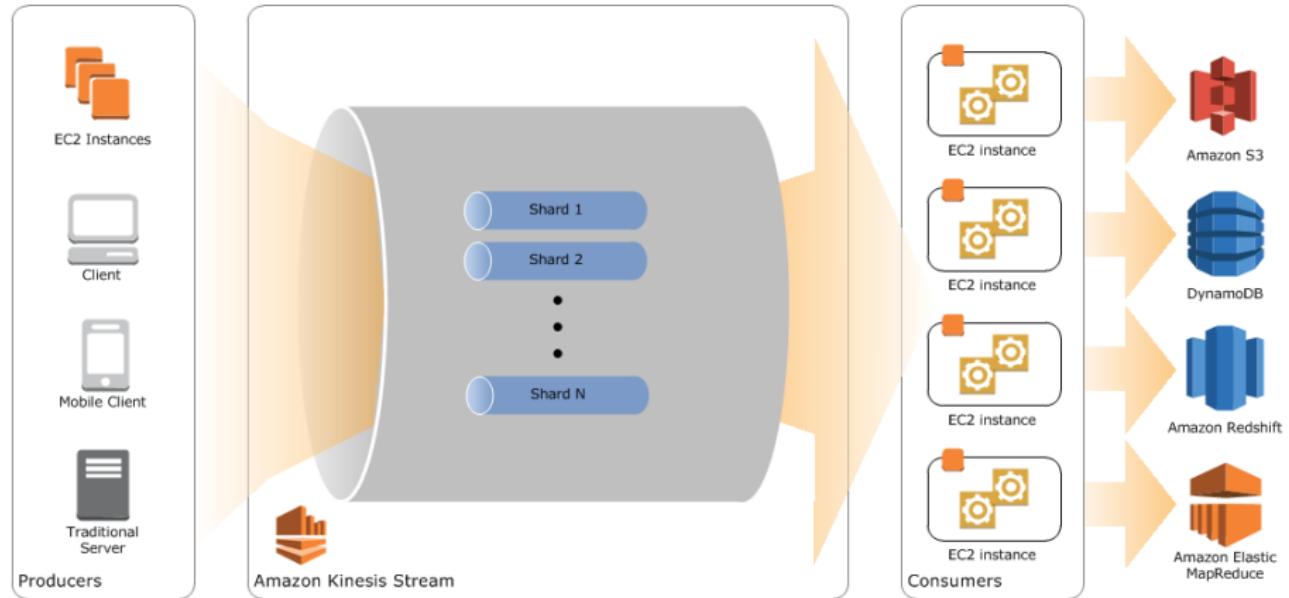
Stream Processing ... Why AWS?

useR!2017

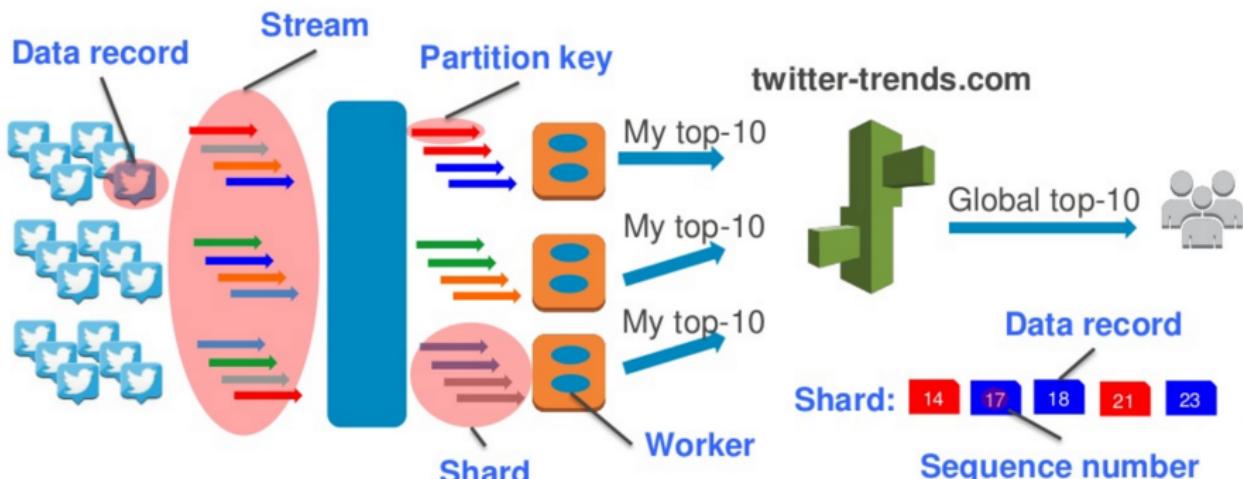




Source: Kinesis Product Details



Source: Kinesis Developer Guide



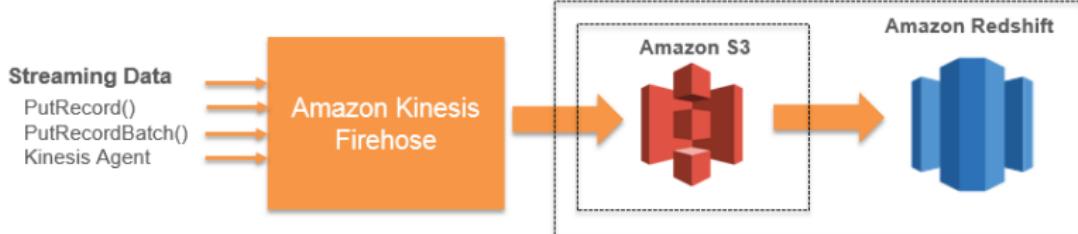
Source: AWS re:Invent 2013

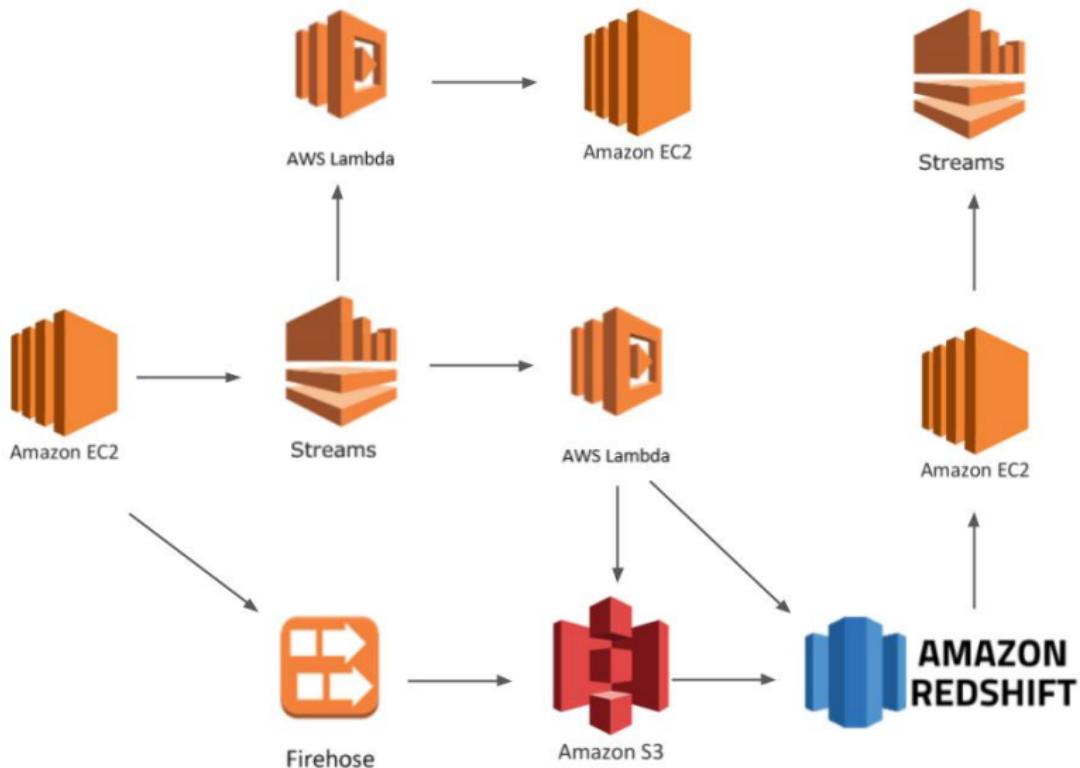


*Capture & submit  
streaming data to Firehose*

*Firehose loads streaming data  
continuously into S3 and Redshift*

*Analyze streaming data using your favorite BI tools*





Writing data to the stream:

- Amazon Kinesis Streams API, SDK
- Amazon Kinesis Producer Library (KPL) from Java
- flume-kinesis
- Amazon Kinesis Agent

Reading data from the stream:

- Amazon Kinesis Streams API, SDK
- Amazon Kinesis Client Library (KCL) from Java, Node.js, .NET, Python, Ruby

Managing streams:

- Amazon Kinesis Streams API (!)

```
> library(rJava)
> .jinit(classpath = list.files('~/Projects/AWR/inst/java/', full.names = TRUE))

> kc <- .jnew('com.amazonaws.services.kinesis.AmazonKinesisClient')
> kc$setEndpoint('kinesis.us-west-2.amazonaws.com', 'kinesis', 'us-west-2')

> sir <- .jnew('com.amazonaws.services.kinesis.model.GetShardIteratorRequest')
> sir$setStreamName('test_kinesis')
> sir$setShardId(.jnew('java/lang/String', '0'))
> sir$setShardIteratorType('TRIM_HORIZON')
> iterator <- kc$getShardIterator(sir)$getShardIterator()

> grr <- .jnew('com.amazonaws.services.kinesis.model.GetRecordsRequest')
> grr$setShardIterator(iterator)
> kc$getRecords(grr)$getRecords()
[1] "Java-Object{[SequenceNumber: 495628941604494443321533463710843135723243616650,
ApproximateArrivalTimestamp: Tue Jun 14 09:40:19 CEST 2016,
Data: java.nio.HeapByteBuffer[pos=0 lim=6 cap=6],PartitionKey: 42]}"

> sapply(kc$getRecords(grr)$getRecords(),
+         function(x)
+             rawToChar(x$getData()$array()))
[1] "foobar"
```

Let's merge two shards:

```
> ms <- .jnew('com.amazonaws.services.kinesis.model.MergeShardsRequest')
> ms$setShardToMerge('shardId-000000000000')
> ms$setAdjacentShardToMerge('shardId-000000000001')
> ms$setStreamName('test_kinesis')
> kc$mergeShards(ms)
```

What do we have now?

```
> kc$describeStream(StreamName = 'test_kinesis')$getStreamDescription()$getShards()
[1] "Java-Object{[
{ShardId: shardId-000000000000,HashKeyRange: {StartingHashKey: 0,EndingHashKey: 170
SequenceNumberRange: {
StartingSequenceNumber: 49562894160427143586954815717376297430913467927668719618,
EndingSequenceNumber: 49562894160438293959554081028945856364232263390243848194}},
{ShardId: shardId-000000000001,HashKeyRange: {StartingHashKey: 17014118346046923173
SequenceNumberRange: {
StartingSequenceNumber: 4956289416044944332153346340517833149186116289174700050,
EndingSequenceNumber: 49562894160460594704752611652087392082504911751749828626}},
{ShardId: shardId-000000000002,
ParentShardId: shardId-000000000000,
AdjacentParentShardId: shardId-000000000001,
HashKeyRange: {StartingHashKey: 0,EndingHashKey: 3402823669209384634633746074317682
SequenceNumberRange: {StartingSequenceNumber: 495629049914976730997049243472701952
Gergely Daroczi (@daroczig) Stream processing using AWR gitlab.com/cardcorp/AWR 15 / 62
```

- An *easy-to-use* programming model for processing data

```
java -cp amazon-kinesis-client-1.7.3.jar \
com.amazonaws.services.kinesis.multilang.MultiLangDaemon \
app.properties
```

- *Scalable* and *fault-tolerant* processing (checkpointing via DynamoDB)
- Logging and metrics in CloudWatch
- The **MultiLangDaemon** spawns processes written in any language, communication happens via JSON messages sent over stdin/stdout
- Only a few events/methods to care about in the consumer application:
  - ① initialize
  - ② processRecords
  - ③ checkpoint
  - ④ shutdown

## ① initialize:

- Perform initialization steps
- Write “status” message to indicate you are done
- Begin reading line from STDIN to receive next action

## ② processRecords:

- Perform processing tasks (you may write a checkpoint message at any time)
- Write “status” message to STDOUT to indicate you are done.
- Begin reading line from STDIN to receive next action

## ③ shutdown:

- Perform shutdown tasks (you may write a checkpoint message at any time)
- Write “status” message to STDOUT to indicate you are done.
- Begin reading line from STDIN to receive next action

## ④ checkpoint:

- Decide whether to checkpoint again based on whether there is an error or not.

```
#!/usr/bin/r -i

while (TRUE) {

  ## read and parse JSON messages
  line <- fromJSON(readLines(n = 1))

  ## nothing to do unless we receive records to process
  if (line$action == 'processRecords') {

    ## process each record
    lapply(line$records, function(r) {

      business_logic(fromJSON(rawToChar(base64_dec(r$data))))
      cat(toJSON(list(action = 'checkpoint', checkpoint = r$sequenceNumber)))

    })
  }

  ## return response in JSON
  cat(toJSON(list(action = 'status', responseFor = line$action)))
}

}
```

```
#!/usr/bin/r -i

while (TRUE) {

  ## read and parse JSON messages
  line <- fromJSON(readLines(n = 1))

  ## nothing to do unless we receive records to process
  if (line$action == 'processRecords') {

    ## process each record
    lapply(line$records, function(r) {

      business_logic(fromJSON(rawToChar(base64_dec(r$data))))
      cat(toJSON(list(action = 'checkpoint', checkpoint = r$sequenceNumber)))

    })
  }

  ## return response in JSON
  cat(toJSON(list(action = 'status', responseFor = line$action)))
}

}
```

```
> install.packages('AWR.Kinesis')
also installing the dependency 'AWR'

trying URL 'https://cloud.r-project.org/src/contrib/AWR_1.11.89.tar.gz'
Content type 'application/x-gzip' length 3125 bytes

trying URL 'https://cloud.r-project.org/src/contrib/AWR.Kinesis_1.7.3.tar.gz'
Content type 'application/x-gzip' length 3091459 bytes (2.9 MB)

* installing *source* package 'AWR' ...
** testing if installed package can be loaded
trying URL 'https://gitlab.com/cardcorp/AWR/repository/archive.zip?ref=1.11.89'
downloaded 58.9 MB
* DONE (AWR)

* installing *source* package 'AWR.Kinesis' ...
* DONE (AWR.Kinesis)
```

Business logic coded in R (demo\_app.R):

```
library(AWR.Kinesis)
kinesis_consumer(processRecords = function(records) {
  flog.info(jsonlite::toJSON(records))
})
```

Business logic coded in R (demo\_app.R):

```
library(AWR.Kinesis)
kinesis_consumer(processRecords = function(records) {
  flog.info(jsonlite::toJSON(records))
})
```

## Note

This is not something you should run in RStudio.

Business logic coded in R (demo\_app.R):

```
library(AWR.Kinesis)
kinesis_consumer(processRecords = function(records) {
  flog.info(jsonlite::toJSON(records))
})
```

Config file for the MultiLangDaemon (demo\_app.properties):

```
executableName = ./demo_app.R
streamName = demo_stream
applicationName = demo_app
```

Start the MultiLangDaemon:

```
/usr/bin/java -cp AWR/java/*:AWR.Kinesis/java/*:./ \
  com.amazonaws.services.kinesis.multilang.MultiLangDaemon \
  ./demo_app.properties
```

```
library(futile.logger)
library(AWR.Kinesis)

kinesis_consumer(
  initialize      = function()
    flog.info('Hello'),
  processRecords = function(records)
    flog.info(paste('Received', nrow(records), 'records from Kinesis')),
  shutdown        = function()
    flog.info('Bye'),
  updater         = list(
    list(1, function()
      flog.info('Updating some data every minute')),
    list(1/60*10, function()
      flog.info(paste(
        'This is a high frequency updater call',
        'running every 10 seconds')))),
  checkpointing = 1,
  logfile = '/logs/logger.log')
```

## Note

In theory you could, but this is not something you should run in RStudio.

- ① Create a Kinesis Stream
- ② Create an IAM user with DynamoDB and Kinesis permissions
- ③ Write data to the Stream
- ④ Run the MultiLangDaemon referencing the properties file

## Note

In theory you could, but this is not something you should run in RStudio.

- ① Create a Kinesis Stream
- ② Create an IAM user with DynamoDB and Kinesis permissions
- ③ Write data to the Stream
- ④ Run the MultiLangDaemon referencing the properties file



The screenshot shows the Amazon Kinesis service page in the AWS Management Console. The URL is <https://console.aws.amazon.com/kinesis/home?region=us-east-1#/>. The page features a central logo with a bar chart icon and the text "Amazon Kinesis". Below it, a sub-headline reads: "Amazon Kinesis services make it easier to work with real-time streaming data in the AWS Cloud." Three main service icons are displayed: "Amazon Kinesis Firehose" (represented by a red firehose icon), "Amazon Kinesis Analytics" (represented by a magnifying glass over a SQL icon), and "Amazon Kinesis Streams" (represented by a green valve icon). Each service has a brief description, a "Go to [Service]" button, and a "Learn more about [Service]" link. At the bottom, there's a footer with links to documentation and forums.

Amazon Kinesis

Amazon Kinesis services make it easier to work with real-time streaming data in the AWS Cloud.

**Amazon Kinesis Firehose**

Continuously deliver streaming data to Amazon S3, Amazon Redshift, and Amazon Elasticsearch Service.

[Go to Firehose](#)

[Learn more about Firehose](#)

**Amazon Kinesis Analytics**

Analyze streaming data from Amazon Kinesis Firehose and Amazon Kinesis Streams in real-time using SQL.

[Go to Analytics](#)

[Learn more about Analytics](#)

**Amazon Kinesis Streams**

Collect and stream data for ordered, replayable, real-time processing.

[Go to Streams](#)

[Learn more about Streams](#)

Amazon Kinesis documentation and support

[Firehose documentation](#) | [Analytics documentation](#) | [Streams documentation](#) | [Forums](#)

[Feedback](#) | [English](#)

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. [Privacy Policy](#) | [Terms of Use](#)

Amazon Kinesis Stream

https://console.aws.amazon.com/kinesis/home?region=us-east-1#/streams/create

Gergely Daróczki N. Virginia Support

## Create stream

Streams

Firehose

Analytics

Stream name\* test-AWR

### Shards

A shard is a unit of throughput capacity. Each shard ingests up to 1MB/sec and 1000 records/sec, and emits up to 2MB/sec. To accommodate for higher or lower throughput, the number of shards can be modified after the stream is created using the API. [Learn more](#)

Producers → Stream → Consumers

Estimate the number of shards you'll need

Number of shards\* 1

You can provision up to 48 more shards before hitting your account limit of 50.  
[Learn more or request a shard limit increase for this account](#)

Total stream capacity Values are calculated based on the number of shards entered above.

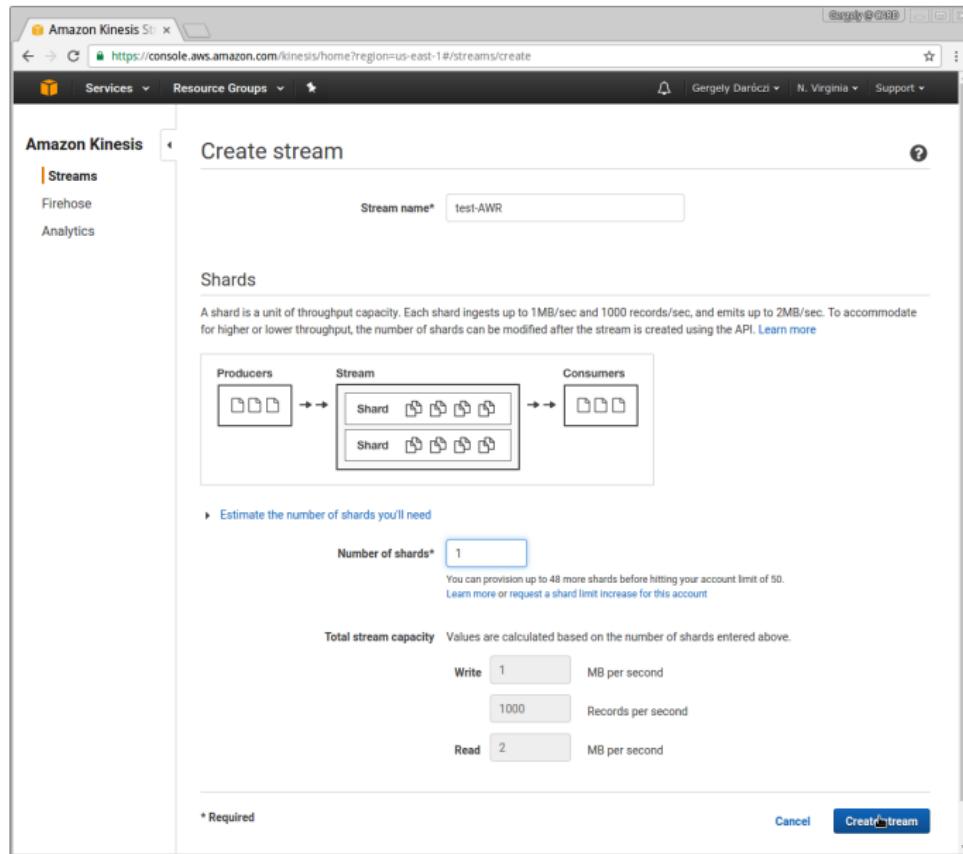
Write 1 MB per second

1000 Records per second

Read 2 MB per second

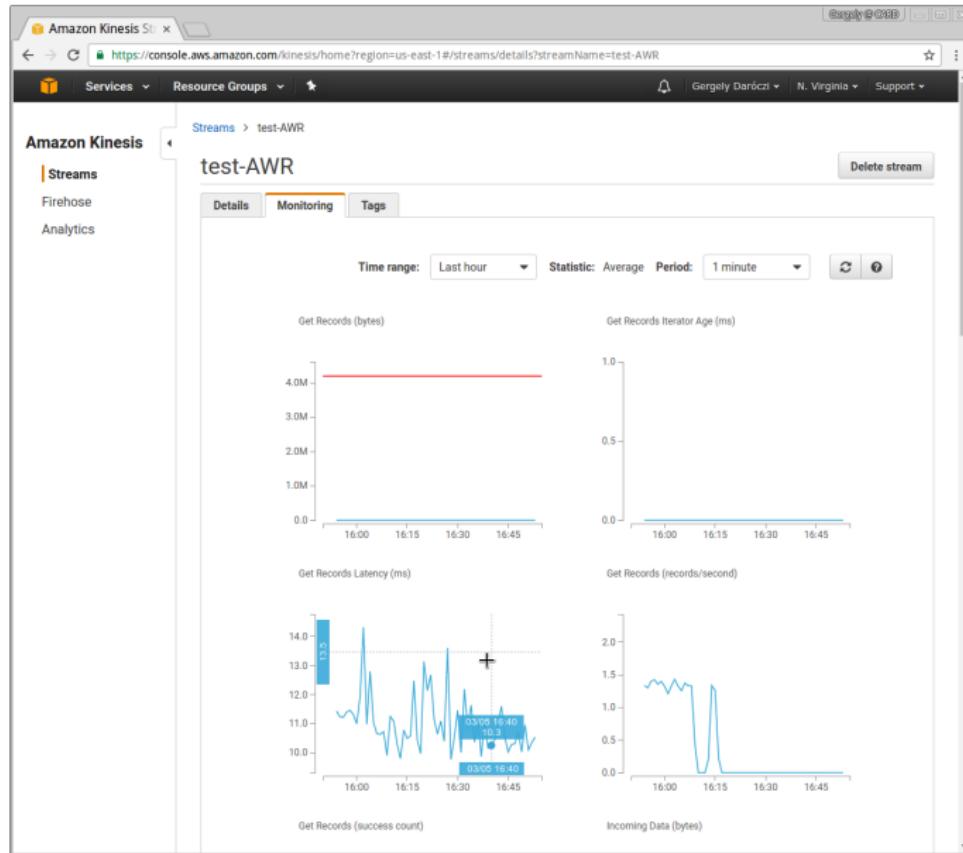
\* Required

Cancel Create stream



## Check the Kinesis Stream

useR!2017



# Create an IAM user

useR!2017

The screenshot shows the AWS IAM Management console interface. The left sidebar navigation bar includes links for Dashboard, Groups, Users (which is selected), Roles, Policies, Identity providers, Account settings, and Credential report. The main content area displays the details for a user named "AWR". The user's ARN is listed as "arn:aws:iam::[REDACTED]user/AWR", with a path of "/". The creation time is noted as "2017-02-13 16:04 PST". Below this, there are tabs for "Permissions", "Groups (0)", "Security credentials", and "Access Advisor". The "Permissions" tab is active, showing five attached policies: "AmazonEC2ContainerRegistryFullAccess - AWS Managed policy", "CloudWatchFullAccess - AWS Managed policy", "cloudwatch-putmetrics - Managed policy", "AmazonDynamoDBFullAccess - AWS Managed policy", and "AmazonKinesisFullAccess - AWS Managed policy". A button labeled "Add inline policy" is visible at the bottom of this section. The footer of the page includes links for Feedback, English, and other AWS services like S3, Lambda, and CloudWatch.

```
library(rJava)
.jcall("java/lang/System", "S", "setProperty", "aws.profile", "personal")

library(AWR.Kinesis)
library(jsonlite)
library(futile.logger)
library(nycflights13)
while (TRUE) {

  ## pick a ~car~flight
  flight <- flights[sample(1:nrow(flights), 1), ]

  ## prr <- .jnew('com.amazonaws.services.kinesis.model.PutRecordRequest')
  ## prr$setStreamName('test1')
  ## prr$setData(J('java.nio.ByteBuffer')$wrap(.jbyte(charToRaw(toJSON(car)))))
  ## prr$setPartitionKey(rownames(car))
  ## kc$putRecord(prr)

  res <- kinesis_put_record(stream = 'test-AWR', region = 'us-east-1',
                             data = toJSON(flight), partitionKey = flight$dest)
  flog.info(paste('Pushed a new flight to Kinesis:', res$sequenceNumber))

}
```

# Write Data to the Stream from R

useR!2017

```
library(futile.logger)
library(AWR.Kinesis)

kinesis_consumer(
  initialize      = function()
    flog.info('Hello'),
  processRecords = function(records)
    flog.info(paste('Received', nrow(records), 'records from Kinesis')),
  shutdown        = function()
    flog.info('Bye'),
  updater         = list(
    list(1, function()
      flog.info('Updating some data every minute')),
    list(1/60*10, function()
      flog.info(paste(
        'This is a high frequency updater call',
        'running every 10 seconds')))),
  checkpointing = 1,
  logfile = '/logs/logger.log')
```

# Running the MultiLangDaemon locally

useR!2017

```
Terminix: Default
1 / 1 ▾ + ⌂ ⌂

1: daroczig@gergely-CARD: ~/Projects/card-rocker/r-kinesis-example/files ~ master ? export AWS_PROFILE=personal
~/Projects/card-rocker/r-kinesis-example/files ~ master ? /usr/bin/java -cp \
"/usr/local/lib/R/site-library/AWR/java":~/usr/local/lib/R/site-library/AWR.Kinesis/java/*:." \
com.amazonaws.services.kinesis.multilang.MultilangDaemon ./app.properties
Mar 05, 2017 5:32:33 PM com.amazonaws.services.kinesis.clientlibrary.config.KinesisClientLibConfigurator getConfiguration
INFO: Value of workerId is not provided in the properties. WorkerId is automatically assigned as:
Mar 05, 2017 5:32:34 PM com.amazonaws.services.kinesis.clientlibrary.config.KinesisClientLibConfigurator withProperty
INFO: Successfully set property regionName with value us-east-1
Mar 05, 2017 5:32:34 PM com.amazonaws.services.kinesis.multilang.MultilangDaemonConfig buildExecutorService
INFO: Using a cached thread pool.
Mar 05, 2017 5:32:34 PM com.amazonaws.services.kinesis.multilang.MultilangDaemonConfig <init>
INFO: Running AWR-demo-app to process stream test-AWR with executable /app/app.R
Mar 05, 2017 5:32:34 PM com.amazonaws.services.kinesis.multilang.MultilangDaemonConfig prepare
INFO: Using workerId:
Mar 05, 2017 5:32:34 PM com.amazonaws.services.kinesis.multilang.MultilangDaemonConfig prepare
INFO: Using credentials with access key id:
Mar 05, 2017 5:32:34 PM com.amazonaws.services.kinesis.multilang.MultilangDaemonConfig prepare
INFO: MultiLangDaemon is adding the following fields to the User Agent: amazon-kinesis-client-library-java-1.7.3 amazon-kinesis-multi-lang-dae
mon-1.0.1 R /app/app.R
Mar 05, 2017 5:32:34 PM com.amazonaws.services.kinesis.leases.impl.LeaseCoordinator <init>
INFO: With failover time 10000 ms and epsilon 25 ms, LeaseCoordinator will renew leases every 3308 ms, takeleases every 20050 ms, process maxi
mum of 2147483647 leases and steal 1 lease(s) at a time.
Mar 05, 2017 5:32:34 PM com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker initialize
INFO: Initialization attempt 1
Mar 05, 2017 5:32:34 PM com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker initialize
INFO: Initializing LeaseCoordinator
Mar 05, 2017 5:32:35 PM com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker initialize
INFO: Syncing Kinesis shard info
Mar 05, 2017 5:32:36 PM com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker initialize
INFO: Starting LeaseCoordinator
Mar 05, 2017 5:32:36 PM com.amazonaws.services.kinesis.leases.impl.LeaseTaker computeLeasesToTake
INFO: Worker [REDACTED] needed 2 leases but none were expired, so it will steal lease shardId-000000000002 from befs
4447-3adb-444f-8dc6-67504e5c86ef
Mar 05, 2017 5:32:36 PM com.amazonaws.services.kinesis.leases.impl.LeaseTaker computeLeasesToTake
INFO: Worker [REDACTED] saw 3 total leases, 0 available leases, 2 workers. Target is 2 leases, I have 0 leases, I wi
ll take 1 leases
Mar 05, 2017 5:32:36 PM com.amazonaws.services.kinesis.leases.impl.LeaseTaker takeLeases
INFO: Worker [REDACTED] successfully took 1 leases: shardId-000000000002
Mar 05, 2017 5:32:46 PM com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker run
INFO: Initialization complete. Starting worker loop.
Mar 05, 2017 5:32:46 PM com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker infoForce
INFO: Created new shardConsumer for : ShardInfo [shardId=shardId-000000000002, concurrencyToken=
shardIds=[shardId-000000000000], checkpoint={SequenceNumber: TRIM_HORIZON, SubsequenceNumber: 0}]
Mar 05, 2017 5:32:46 PM com.amazonaws.services.kinesis.clientlibrary.lib.worker.BlockOnParentShardTask call
INFO: No need to block on parents [shardId-000000000000] of shard shardId-000000000002
Mar 05, 2017 5:32:47 PM com.amazonaws.services.kinesis.clientlibrary.lib.worker.KinesisDataFetcher initialize
```

# Running the MultiLangDaemon locally

useR!2017

```
1: ec2-user@ip-10-0-1-100 logs]$ head -n 44 logger.log
INFO [2017-03-05 03:35:23] Starting R Kinesis Consumer application
INFO [2017-03-05 03:35:23 UTC] shardId-000000000000 Start of initialize
INFO [2017-03-05 03:35:23 UTC] shardId-000000000000 Hello
INFO [2017-03-05 03:35:23 UTC] shardId-000000000000 End of initialize
INFO [2017-03-05 03:35:23 UTC] shardId-000000000000 Received 2 records from Kinesis
INFO [2017-03-05 03:35:24 UTC] shardId-000000000000 Received 3 records from Kinesis
INFO [2017-03-05 03:35:25 UTC] shardId-000000000000 Received 3 records from Kinesis
INFO [2017-03-05 03:35:26 UTC] shardId-000000000000 Received 2 records from Kinesis
INFO [2017-03-05 03:35:27 UTC] shardId-000000000000 Received 3 records from Kinesis
INFO [2017-03-05 03:35:28 UTC] shardId-000000000000 Received 2 records from Kinesis
INFO [2017-03-05 03:35:29 UTC] shardId-000000000000 Received 2 records from Kinesis
INFO [2017-03-05 03:35:30 UTC] shardId-000000000000 Received 3 records from Kinesis
INFO [2017-03-05 03:35:31 UTC] shardId-000000000000 Received 3 records from Kinesis
INFO [2017-03-05 03:35:32 UTC] shardId-000000000000 Received 2 records from Kinesis
INFO [2017-03-05 03:35:33 UTC] shardId-000000000000 Received 2 records from Kinesis
INFO [2017-03-05 03:35:33 UTC] shardId-000000000000 This is a high frequency updater call running every 10 seconds
INFO [2017-03-05 03:35:34 UTC] shardId-000000000000 Received 3 records from Kinesis
INFO [2017-03-05 03:35:35 UTC] shardId-000000000000 Received 2 records from Kinesis
INFO [2017-03-05 03:35:36 UTC] shardId-000000000000 Received 3 records from Kinesis
INFO [2017-03-05 03:35:37 UTC] shardId-000000000000 Received 3 records from Kinesis
INFO [2017-03-05 03:35:38 UTC] shardId-000000000000 Received 2 records from Kinesis
INFO [2017-03-05 03:35:39 UTC] shardId-000000000000 Received 2 records from Kinesis
INFO [2017-03-05 03:35:40 UTC] shardId-000000000000 Received 3 records from Kinesis
INFO [2017-03-05 03:35:41 UTC] shardId-000000000000 Received 2 records from Kinesis
INFO [2017-03-05 03:35:42 UTC] shardId-000000000000 Received 3 records from Kinesis
INFO [2017-03-05 03:35:43 UTC] shardId-000000000000 Received 2 records from Kinesis
INFO [2017-03-05 03:35:43 UTC] shardId-000000000000 This is a high frequency updater call running every 10 seconds
INFO [2017-03-05 03:35:44 UTC] shardId-000000000000 Received 3 records from Kinesis
INFO [2017-03-05 03:35:45 UTC] shardId-000000000000 Received 3 records from Kinesis
INFO [2017-03-05 03:35:46 UTC] shardId-000000000000 Received 2 records from Kinesis
INFO [2017-03-05 03:35:47 UTC] shardId-000000000000 Received 3 records from Kinesis
INFO [2017-03-05 03:35:48 UTC] shardId-000000000000 Received 2 records from Kinesis
INFO [2017-03-05 03:35:49 UTC] shardId-000000000000 Received 3 records from Kinesis
INFO [2017-03-05 03:35:50 UTC] shardId-000000000000 Received 2 records from Kinesis
INFO [2017-03-05 03:35:51 UTC] shardId-000000000000 Received 3 records from Kinesis
INFO [2017-03-05 03:35:52 UTC] shardId-000000000000 Received 2 records from Kinesis
INFO [2017-03-05 03:35:53 UTC] shardId-000000000000 Received 3 records from Kinesis
INFO [2017-03-05 03:35:54 UTC] shardId-000000000000 Received 2 records from Kinesis
INFO [2017-03-05 03:35:54 UTC] shardId-000000000000 This is a high frequency updater call running every 10 seconds
INFO [2017-03-05 03:35:55 UTC] shardId-000000000000 Received 3 records from Kinesis
INFO [2017-03-05 03:35:56 UTC] shardId-000000000000 Received 2 records from Kinesis
INFO [2017-03-05 03:35:57 UTC] shardId-000000000000 Received 3 records from Kinesis
INFO [2017-03-05 03:35:58 UTC] shardId-000000000000 Received 2 records from Kinesis
INFO [2017-03-05 03:35:59 UTC] shardId-000000000000 Received 3 records from Kinesis
[ec2-user@ip-10-0-1-100 logs]$
```

## ① Dockerize your Kinesis Consumer:

- Java
- R
- AWR, AWR.Kinesis packages
- app.R
- app.properties
- startup command

## ② Put it on Docker Hub

## ③ Run as a EC2 Container Service Task:

- Create an ECS cluster
- Create ECS Task Role
- Create a Task definition
- Run it (as a service)

# Dockerize your Kinesis Consumer

useR!2017

The screenshot shows a GitHub repository page for `card-rocker/Dockerfile`. The URL is <https://github.com/cardcorp/card-rocker/blob/master/r-kinesis/Dockerfile>. The repository has 13 watchers, 6 stars, and 2 forks. The Dockerfile is named `r-kinesis / Dockerfile` and was last updated by `daroczig` with the commit message `break long lines` on `3bf49ac` 26 seconds ago. It has 1 contributor. The Dockerfile content is as follows:

```
FROM cardcorp/r-aws-java-pandoc:latest
MAINTAINER Gergely Daroczi <gergely.daroczi@card.com>

## Install extra AWR packages
RUN install2r --error \
  AWR.KMS \
  AWR.Kinesis \
  && rm -rf /tmp/downloaded_packages/ /tmp/*.rds

## Run MultiLangDaemon on /app
ENTRYPOINT ["/usr/bin/java", \
  "-cp", \
  "/usr/local/lib/R/site-library/AWR/java/*:/usr/local/lib/R/site-library/AWR.Kinesis/java/*", \
  "com.amazonaws.services.kinesis.multilang.MultiLangDaemon"]

## Override this if the consumer app is mounted elsewhere or the config file has a different name
CMD ["app/app.properties"]
```

# Dockerize your Kinesis Consumer

useR!2017

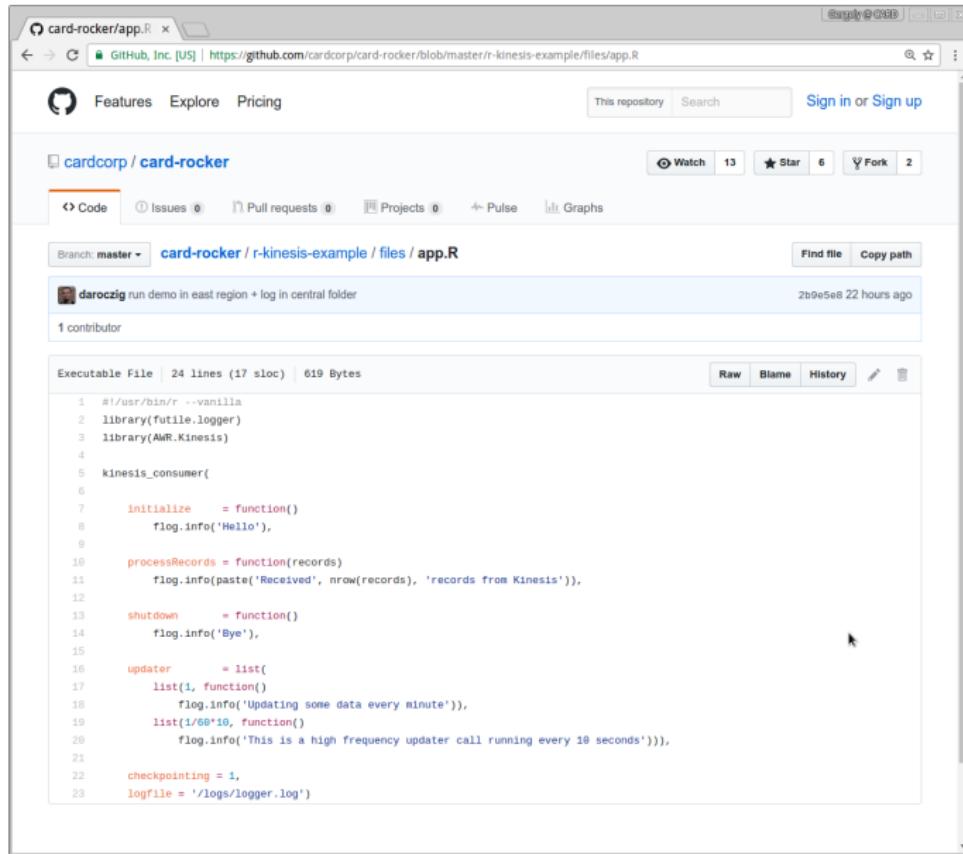
The screenshot shows a GitHub repository page for 'card-rocker/Dockerfile'. The repository has 13 watchers, 6 stars, and 2 forks. The 'Code' tab is selected, showing a Dockerfile. The file was last updated by 'daroczig' a day ago. The Dockerfile content is as follows:

```
FROM cardcorp/r-kinesis:latest
MAINTAINER Gergely Daroczi <gergely.daroczi@card.com>
## Add consumer
COPY files /app
```

At the bottom of the page, there are links for GitHub, Inc. (Terms, Privacy, Security, Status, Help), Contact GitHub, API, Training, Shop, Blog, and About.

# Dockerize your Kinesis Consumer

useR!2017



The screenshot shows a GitHub browser interface. The URL is <https://github.com/cardcorp/card-rocker/blob/master/r-kinesis-example/files/app.R>. The repository name is `cardcorp / card-rocker`. The branch is `master`. The file is `app.R`. The code content is as follows:

```
#!/usr/bin/r --vanilla
library(futile.logger)
library(AWR.Kinesis)

kinesis_consumer(
  initialize = function()
    flog.info('Hello'),

  processRecords = function(records)
    flog.info(paste('Received', nrow(records), 'records from Kinesis')),

  shutdown = function()
    flog.info('Bye'),

  updater = list(
    list(1, function()
      flog.info('Updating some data every minute')),
    list(1/60*10, function()
      flog.info('This is a high frequency updater call running every 10 seconds'))),
  checkpointing = 1,
  logfile = '/logs/logger.log')
```

# Dockerize your Kinesis Consumer

useR!2017

The screenshot shows a GitHub repository page for 'card-rocker/app.R'. The page includes navigation links for Features, Explore, Pricing, and Sign in or Sign up. It displays repository statistics: Watch (13), Star (6), Fork (2). The code file 'app.properties' is shown, containing the following content:

```
1 executableName = /app/app.R
2 streamName = test-AWR
3 applicationName = AWR-demo-app
4 AWSCredentialsProvider = DefaultAWSCredentialsProviderChain
5 processingLanguage = R
6 regionName = us-east-1
```

At the bottom, there are links for Raw, Blame, History, and a copy icon. The footer of the GitHub page includes links for Terms, Privacy, Security, Status, Help, Contact GitHub, API, Training, Shop, Blog, and About.

The screenshot shows a browser window displaying the Docker Hub interface for the repository `cardcorp/r-kinesis-example`. The page title is "PUBLIC | AUTOMATED BUILD". Below the title, it says "Last pushed: a day ago". There are tabs for "Repo Info", "Tags", "Dockerfile", and "Build Details", with "Build Details" being the active tab. A table lists eight build entries:

Status	Actions	Tag	Created	Last Updated
Building	<button>Cancel</button>	latest	3 minutes ago	a minute ago
✓ Success		latest	a day ago	a day ago
✓ Success		latest	a day ago	a day ago
✓ Success		latest	a day ago	a day ago
✓ Success		latest	a day ago	a day ago
✓ Success		latest	a day ago	a day ago
✓ Success		latest	a day ago	a day ago

To the right of the table, there is a sidebar titled "Source Repository" which shows the URL `cardcorp/card-rocker`.

The screenshot shows the 'Create Cluster' wizard in the AWS Management Console. The left sidebar shows 'Amazon ECS' and 'Clusters' selected. The main area has a heading 'Create Cluster' with a sub-instruction: 'When you run tasks using Amazon ECS, you place them on a cluster, which is a logical grouping of EC2 instances. This wizard will guide you through the process to [create a cluster](#). You will name your cluster, and then configure the container instances that your tasks can be placed on, the security group for your container instances to use, and the IAM role to associate with your container instances so that they can make calls to the AWS APIs on your behalf.' Below this, there are several input fields:

- Cluster name\***: AWR-test
- Create an empty cluster
- EC2 Instance type\***: t2.medium
- Number of instances\***: 1
- EC2 Ami Id\***: amzn-ami-2016.09.1-amazon-ecs-optimized [ami-b2df2ca4]
- EBS storage (GiB)\***: 22
- Key pair**: (dropdown menu)

A note below the key pair field states: 'You will not be able to SSH into your EC2 instances without a key pair. You can create a new key pair in the [EC2 console](#)'.

**Networking**

Configure the VPC for your container instances to use. A VPC is an isolated portion of the AWS cloud populated by AWS objects, such as Amazon EC2 instances. You can choose an existing VPC, or create a new one with this wizard.

- VPC**: Create a new vpc
- CIDR Block**: 10.0.0.0/16

# Create ECS Task Role

useR!2017

The screenshot shows the AWS IAM Management console with the URL <https://console.aws.amazon.com/iam/home?region=us-east-1#/roles>. The page title is "Create Role". The main section is titled "Set Role Name" with the sub-instruction "Enter a role name. You cannot edit the role name after the role is created." A "Role Name" input field contains "AWR-ECS". Below it is a note: "Maximum 64 characters. Use alphanumeric and '-' characters". At the bottom right are "Cancel" and "Next Step" buttons.

IAM Management

Services Resource Groups

Gergely Daróczki Global Support

Create Role

Step 1 : Set Role Name

Step 2 : Select Role Type

Step 3 : Establish Trust

Step 4 : Attach Policy

Step 5 : Review

Set Role Name

Enter a role name. You cannot edit the role name after the role is created.

Role Name  Maximum 64 characters. Use alphanumeric and '-' characters

Cancel Next Step

# Create ECS Task Role

useR!2017

The screenshot shows the AWS IAM Management console with the URL <https://console.aws.amazon.com/iam/home?region=us-east-1#roles>. The page is titled "Create Role" and is on "Step 2 : Select Role Type". A sidebar on the left lists steps: Step 1 : Set Role Name, Step 2 : Select Role Type (which is active), Step 3 : Establish Trust, Step 4 : Attach Policy, and Step 5 : Review.

The main content area is titled "Select Role Type" and contains a list of AWS Service Roles:

- Amazon EC2 Role for EC2 Container Service**: Role to allow EC2 instances in an Amazon ECS cluster to access Amazon ECS.
- Amazon EC2 Container Service Role**: Allows ECS to create and manage AWS resources on your behalf.
- Amazon EC2 Container Service Task Role**: Allows ECS tasks to call AWS services on your behalf.
- Amazon EC2 Spot Fleet Role**: Role to Allow EC2 Spot Fleet to request and terminate Spot Instances on your behalf.
- Amazon Elastic MapReduce**: Role to allow EMR to access other AWS services such as EC2 on your behalf.
- Role for Cross-Account Access** (radio button selected)
- Role for Identity Provider Access** (radio button unselected)

At the bottom right are buttons for "Cancel", "Previous", and "Next Step".

# Create ECS Task Role

useR!2017

IAM Management > Services > Resource Groups

https://console.aws.amazon.com/iam/home?region=us-east-1#roles

Create Role

Step 1 : Set Role Name

Step 2 : Select Role Type

Step 3 : Establish Trust

**Step 4 : Attach Policy**

Step 5 : Review

Attach Policy

Select one or more policies to attach. Each role can have up to 10 policies attached.

Filter: Policy Type ▾ kinesis Showing 7 results

	Policy Name	Attached Entities	Creation Time	Edited Time
<input type="checkbox"/>	AmazonKinesisFullAccess	1	2015-02-06 10:40 PST	2015-02-06 10:40 PST
<input checked="" type="checkbox"/>	AmazonKinesisReadOnlyA...	1	2015-02-06 10:40 PST	2015-02-06 10:40 PST
<input type="checkbox"/>	AmazonKinesisAnalyticsF...	0	2016-09-21 12:01 PST	2016-09-21 12:01 PST
<input type="checkbox"/>	AmazonKinesisAnalyticsR...	0	2016-09-21 11:16 PST	2016-09-21 11:16 PST
<input type="checkbox"/>	AmazonKinesisFirehoseFu...	0	2015-10-07 11:45 PST	2015-10-07 11:45 PST
<input type="checkbox"/>	AmazonKinesisFirehoseR...	0	2015-10-07 11:43 PST	2015-10-07 11:43 PST
<input type="checkbox"/>	AWSLambdaKinesisExecu...	0	2015-04-09 08:14 PST	2015-04-09 08:14 PST

Cancel Previous Next Step

# Create a Task definition

useR!2017

The screenshot shows the 'Create a Task Definition' page in the AWS ECS console. The left sidebar shows 'Amazon ECS' selected under 'Task Definitions'. The main form has the following fields:

- Task Definition Name:** AWR-logger
- Task Role:** AWR-ECS (selected from a dropdown)
- Network Mode:** Bridge
- Constraint:** A section explaining constraints for filtering placement strategies.
- Type:** A table with one row showing 'Add constraint'.
- Container Definitions:** A table with columns: Container Name, Image, Hard/Soft memory limits (MB), Essential. It shows 'No results'.
- Volumes:** A table with columns: Name, Source Path. It shows 'No results'.
- Buttons:** 'Add container', 'Add volume', 'Configure via JSON'.

# Create a Task definition

useR!2017

The screenshot shows the 'Create a Task Definition' dialog box over a background of the AWS ECS console. The dialog is titled 'Add volume' and contains fields for 'Name\*' (set to 'logs') and 'Source path' (set to '/logs'). A note at the bottom left says '\*Required'. At the bottom right are 'Cancel' and 'Add' buttons. Below the dialog, the ECS console shows sections for 'Type' (with 'Add constraint' selected), 'Container Definitions' (with 'Add container' button), 'Volumes' (with 'Add volume' button), and 'Configure via JSON'.

A task definition for your cluster

Add volume

Name\* logs

Source path /logs

\*Required

Cancel Add

Constraint

Constraints allow you to filter the instances used for your placement strategies using built-in or custom attributes. The scheduler first filters the instances that match the constraints and then applies the placement strategy to place the task.

Type

Add constraint

Container Definitions

Add container

Container Name	Image	Hard/Soft memory limits (MB)	Essential
No results			

Volumes

Name	Source Path
No results	

Add volume

Configure via JSON

# Create a Task definition

useR!2017

The screenshot shows the 'Create a Task Definition' dialog box on the AWS ECS console. The left sidebar lists 'Amazon ECS', 'Clusters', 'Task Definitions' (which is selected), and 'Repositories'. The main area is titled 'Add container' under 'Standard' configuration.

**Container name\***: logger

**Image\***: cardcorp/r-kinesis-example:latest

**Memory Limits (MB)\***: Hard limit 512

**Type**: Add constraint

**Port mappings**: Host port, Container port, Protocol: tcp

**Advanced container configuration**: ENVIRONMENT

**CPU units**: (input field)

**Essential**: checked

A tooltip for CPU units states: "The number of cpu units to reserve for the container. A container instance has 1.024 cpu units for every CPU core."

\* Required

Cancel Add

# Create a Task definition

useR!2017

The screenshot shows the 'Create a Task Definition' wizard in the AWS ECS console. The left sidebar lists 'Amazon ECS', 'Clusters', 'Task Definitions' (which is selected), and 'Repositories'. The main area is titled 'Create a Task Definition' with the sub-instruction: 'A task definition specifies the resources required for your containers to run.' Below this, there's a 'Constraint' section and a 'Type' section with an 'Add constraint' button.

The central part of the screen is the 'Add container' configuration. It includes fields for 'IP address' and 'Add extra host'. The 'STORAGE AND LOGGING' section contains:

- 'Read only root file system': A checkbox that is unchecked.
- 'Mount points': A section where a new mount point is being added. The source volume is 'logs', the container path is '/logs', and it is set to 'Read only'. A tooltip for this section states: 'When this parameter is true, the container is given read-only access to its root file system.'
- 'Volumes from': A section where a new volume is being added. The source container is 'Source container' and it is set to 'Read only'. A tooltip for this section states: 'This is used to share volumes between multiple containers in the same task definition.'
- 'Log configuration': A section with 'Log driver' set to '<none>', 'Log options', and 'Key Value' fields.

The 'SECURITY' section is partially visible at the bottom.

At the bottom right of the configuration pane are 'Cancel' and 'Add' buttons.

# Run the ECS Task

useR!2017

The screenshot shows the AWS ECS console interface. The left sidebar navigation includes 'Amazon ECS', 'Clusters', 'Task Definitions' (which is selected and highlighted in orange), and 'Repositories'. The main content area displays the 'Task Definition Name : AWR-logger' page. At the top, there is a breadcrumb navigation: 'Task Definitions > AWR-logger > status > ACTIVE'. Below this, a message says 'Select a revision for more details'. On the left, there is a 'Actions' dropdown menu with options: 'Create new revision', 'Status: Active (Inactive)', 'Filter in this page', and a 'Run Task' option which is currently selected and highlighted in blue. The main table lists one task definition entry:

Task Definition Name	Status
AWR-logger:1	Active

At the bottom of the table, there are pagination controls ('1-1 / Page size 50') and a refresh button. The status bar at the bottom of the browser window shows 'Gergely Daróczki' and 'N. Virginia'.

# Run the ECS Task

useR!2017

The screenshot shows the AWS CloudWatch Stream Processing using Amazon ECS (AWS Lambda) interface. The top navigation bar includes 'Amazon EC2 Control' and the URL <https://console.aws.amazon.com/ecs/home?region=us-east-1#/clusters/AWR/tasks>. The user is Gergely Daróczki from N. Virginia.

The left sidebar menu has 'Clusters' selected under 'Amazon ECS'. The main content area shows the 'Cluster : AWR' page with the following details:

- Status: ACTIVE
- Registered container instances: 1
- Pending tasks count: 0
- Running tasks count: 1

Below this, there are tabs for 'Services', 'Tasks' (which is selected), 'ECS Instances', and 'Metrics'. A 'Run new Task' button is available. The 'Last updated on March 5, 2017 6:05:01 PM (0m ago)' message is displayed next to a refresh icon.

The 'Desired task status' dropdown is set to 'Running'. A 'Filter in this page' input field and a 'Page size' dropdown (set to 50) are also present.

A table lists the running task:

Task	Task Definition	Group	Container Inst...	Last status	Desired status	Started By
0c9f224a-7808...	AWR-logger:1	family:AWR-log...	27b4935-70e2...	RUNNING	RUNNING	

The bottom footer contains the URL <https://console.aws.amazon.com/ecs/home?region=...>, a copyright notice (© 2008-2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.), and links for 'Privacy Policy' and 'Terms of Use'.

# Run the ECS Task

useR!2017

```
1: ec2-user@... ~]$ ssh -i ~/.ssh/... ec2-user@...
Last login: Mon Mar  6 02:05:29 2017 from ...
[ec2-user@... ~]$ ls
Amazon ECS-Optimized Amazon Linux AMI 2016.09.f
For documentation visit, http://aws.amazon.com/documentation/ecs
4 package(s) needed for security, out of 6 available
Run "sudo yum update" to apply all updates.
[ec2-user@... ~]$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS          NAMES
b59981414dda        cardcorp/r-kinesis-example:latest   "/usr/bin/java -cp /u"   22 hours ago      Up 22 hours         e
cs-AWR-logger-1-logger-92ca888bd5d681e59d01   amazon/amazon-ecs-agent:latest    "/agent"           23 hours ago      Up 23 hours         e
Zed65c5a3752        cs-agent
[ec2-user@... ~]$ pgrep app.R
29435
[ec2-user@... ~]$ head -n 10 /logs/logger.log
INFO [2017-03-05 03:35:23] Starting R Kinesis Consumer application
INFO [2017-03-05 03:35:23 UTC] shardId-000000000000 Start of initialize
INFO [2017-03-05 03:35:23 UTC] shardId-000000000000 Hello
INFO [2017-03-05 03:35:23 UTC] shardId-000000000000 End of initialize
INFO [2017-03-05 03:35:23 UTC] shardId-000000000000 Received 2 records from Kinesis
INFO [2017-03-05 03:35:24 UTC] shardId-000000000000 Received 3 records from Kinesis
INFO [2017-03-05 03:35:25 UTC] shardId-000000000000 Received 3 records from Kinesis
INFO [2017-03-05 03:35:26 UTC] shardId-000000000000 Received 2 records from Kinesis
INFO [2017-03-05 03:35:27 UTC] shardId-000000000000 Received 3 records from Kinesis
INFO [2017-03-05 03:35:28 UTC] shardId-000000000000 Received 2 records from Kinesis
[ec2-user@... ~]$
```

# Scaling the Kinesis Consumer up

useR!2017

```
1 / 1 + ↻ Terminix: Default
1: ec2-user@ip-172-31-10-15: ~
INFO [2017-03-05 03:43:01 UTC] shardId-000000000000 Received 1 records from Kinesis
INFO [2017-03-05 03:43:01 UTC] shardId-000000000000 This is a high frequency updater call running every 10 seconds
INFO [2017-03-05 03:43:09 UTC] shardId-000000000000 Shutting down
INFO [2017-03-05 03:43:09 UTC] shardId-000000000000 Bye
INFO [2017-03-05 03:43:15] Starting R Kinesis Consumer application
INFO [2017-03-05 03:43:15 UTC] shardId-000000000002 Start of initialize
INFO [2017-03-05 03:43:15 UTC] shardId-000000000002 Hello
INFO [2017-03-05 03:43:15 UTC] shardId-000000000002 End of initialize
INFO [2017-03-05 03:43:15] Starting R Kinesis Consumer application
INFO [2017-03-05 03:43:16 UTC] shardId-000000000001 Start of initialize
INFO [2017-03-05 03:43:16 UTC] shardId-000000000001 Hello
INFO [2017-03-05 03:43:16 UTC] shardId-000000000001 End of initialize
INFO [2017-03-05 03:44:32 UTC] shardId-000000000002 Received 1 records from Kinesis
INFO [2017-03-05 03:44:32 UTC] shardId-000000000002 Updating some data every minute
INFO [2017-03-05 03:44:32 UTC] shardId-000000000002 This is a high frequency updater call running every 10 seconds
INFO [2017-03-05 03:44:33 UTC] shardId-000000000002 Received 3 records from Kinesis
INFO [2017-03-05 03:44:34 UTC] shardId-000000000002 Received 1 records from Kinesis
INFO [2017-03-05 03:44:35 UTC] shardId-000000000002 Received 1 records from Kinesis
INFO [2017-03-05 03:44:36 UTC] shardId-000000000001 Received 2 records from Kinesis
INFO [2017-03-05 03:44:36 UTC] shardId-000000000002 Received 1 records from Kinesis
INFO [2017-03-05 03:44:36 UTC] shardId-000000000001 Updating some data every minute
INFO [2017-03-05 03:44:36 UTC] shardId-000000000001 This is a high frequency updater call running every 10 seconds
INFO [2017-03-05 03:44:37 UTC] shardId-000000000001 Received 1 records from Kinesis
INFO [2017-03-05 03:44:38 UTC] shardId-000000000001 Received 1 records from Kinesis
INFO [2017-03-05 03:44:39 UTC] shardId-000000000002 Received 1 records from Kinesis
INFO [2017-03-05 03:44:39 UTC] shardId-000000000001 Received 2 records from Kinesis
INFO [2017-03-05 03:44:40 UTC] shardId-000000000001 Received 1 records from Kinesis
INFO [2017-03-05 03:44:40 UTC] shardId-000000000002 Received 1 records from Kinesis
INFO [2017-03-05 03:44:41 UTC] shardId-000000000001 Received 2 records from Kinesis
INFO [2017-03-05 03:44:42 UTC] shardId-000000000001 Received 2 records from Kinesis
INFO [2017-03-05 03:44:43 UTC] shardId-000000000002 Received 2 records from Kinesis
INFO [2017-03-05 03:44:43 UTC] shardId-000000000002 This is a high frequency updater call running every 10 seconds
INFO [2017-03-05 03:44:44 UTC] shardId-000000000002 Received 1 Records from Kinesis
INFO [2017-03-05 03:44:45 UTC] shardId-000000000002 Received 2 records from Kinesis
INFO [2017-03-05 03:44:45 UTC] shardId-000000000001 Received 1 records from Kinesis
INFO [2017-03-05 03:44:46 UTC] shardId-000000000002 Received 1 records from Kinesis
INFO [2017-03-05 03:44:46 UTC] shardId-000000000001 Received 1 records from Kinesis
INFO [2017-03-05 03:44:47 UTC] shardId-000000000001 Received 1 records from Kinesis
INFO [2017-03-05 03:44:47 UTC] shardId-000000000001 This is a high frequency updater call running every 10 seconds
INFO [2017-03-05 03:44:47 UTC] shardId-000000000002 Received 1 records from Kinesis
INFO [2017-03-05 03:44:48 UTC] shardId-000000000001 Received 1 records from Kinesis
INFO [2017-03-05 03:44:48 UTC] shardId-000000000002 Received 1 records from Kinesis
INFO [2017-03-05 03:44:49 UTC] shardId-000000000001 Received 1 records from Kinesis
INFO [2017-03-05 03:44:49 UTC] shardId-000000000002 Received 1 records from Kinesis
INFO [2017-03-05 03:44:50 UTC] shardId-000000000001 Received 1 records from Kinesis
```

Nice example project, but ...

- I might want to avoid publishing my Consumer on Docker Hub
- I might want to avoid publishing my code on GitHub
- I might want to avoid committing credentials etc to the repo

Problems:

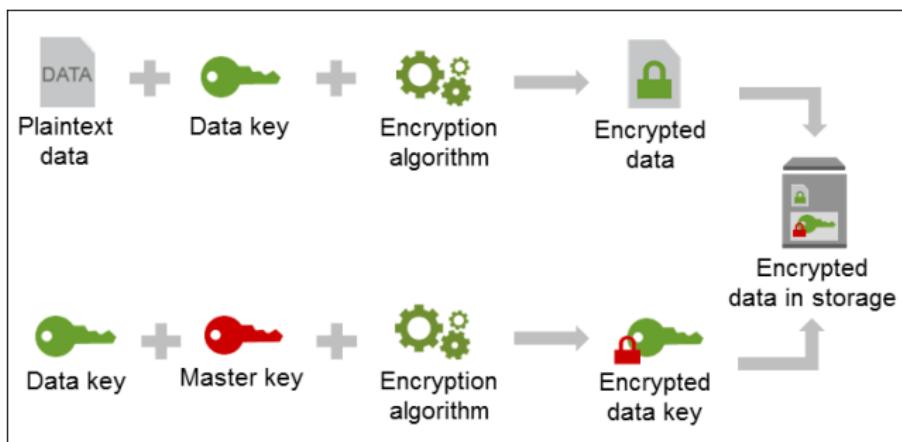
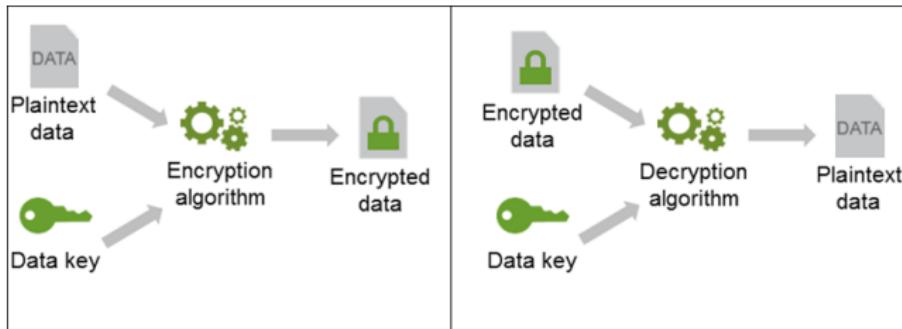
- How to store credentials in the Docker images?
- Where to store the Docker images?

Nice example project, but ...

- I might want to avoid publishing my Consumer on Docker Hub
- I might want to avoid publishing my code on GitHub
- I might want to avoid committing credentials etc to the repo

Problems:

- How to store credentials in the Docker images? **KMS**
- Where to store the Docker images? **ECR**



Source: AWS Encryption SDK

- encrypt up to 4 KB of arbitrary data:

```
> library(AWR.KMS)
> kms_encrypt('alias/mykey', 'foobar')
[1] "Base-64 encoded ciphertext"
```

- decrypt such Base-64 encoded ciphertext back to plaintext:

```
> kms_decrypt('Base-64 encoded ciphertext')
[1] "foobar"
```

- generate a data encryption key:

```
> kms_generate_data_key('alias/mykey')
$cipher
[1] "Base-64 encoded, encrypted data encryption key"
$key
[1] "alias/mykey"
$text
[1] 00 01 10 11 00 01 10 11 ...
```

```
## let's say we want to encrypt the mtcars dataset stored in JSON
library(jsonlite)
data <- toJSON(mtcars)

## generate a 256-bit data encryption key (that's supported by digest::AES)
library(AWR.KMS)
key <- kms_generate_data_key('alias/mykey', byte = 32L)

## convert the JSON to raw so that we can use that with digest::AES
raw <- charToRaw(data)
## the text length must be a multiple of 16 bytes
## https://github.com/sdoyen/r_password_crypt/blob/master/crypt.R
raw <- c(raw, as.raw(rep(0, 16 - length(raw) %% 16)))

## encrypt the raw object with the new key + digest::AES
## the resulting text and the encrypted key can be stored on disk
library(digest)
aes <- AES(key$text)
base64_enc(aes$encrypt(raw))

## decrypt the above returned ciphertext using the decrypted key
rawToChar(aes$decrypt(base64_dec(...), raw = TRUE))
```

# Example “Production” Consumer App

useR!2017

```
library(AWR.Kinesis); library(jsonlite); library(AWR.KMS); library(futile.logger); flog.threshold(DEBUG)

kinesis_consumer(
  initialize      = function() {
    flog.info('Decrypting Redis hostname via KMS')
    host <- kms_decrypt('AQECAHiiz4GEPFQLL9AAON5TY/1DR5euQQScpXQU9iYTn+u... ')
    flog.info('Connecting to Redis')
    library(rredis); redisConnect(host = host)
    flog.info('Connected to Redis')
  },
  processRecords = function(records) {
    flog.info(paste('Received', nrow(records), 'records from Kinesis'))
    for (record in records$data) {
      flight <- fromJSON(record)$dest
      if (!is.null(flight)) {
        flog.debug(paste('Adding +1 to', flight))
        redisIncr(sprintf('flight:%s', flight))
      } else {
        flog.error('Flight destination not found')
      }
    }
  },
  updaters = list(
    list(1/6, function() {
      flog.info('Checking overall counters')
      flights <- redisKeys('flight:*')
      for (flight in flights) {
        flog.debug(paste('Found', redisGet(flight), sub('^flight:', '', flight)))
      }
    })),
  logfile = '/logs/redis.log')
```

## Dockerfile:

```
FROM cardcorp/r-kinesis:latest
MAINTAINER Gergely Daroczi <gergely.daroczi@card.com>

## Install R package to interact with Redis
RUN install2.r --error rredis && rm -rf /tmp/downloaded_packages/ /tmp/*.rds

## Add consumer
COPY files /app
```

## Build and push to ECR:

```
docker build -t cardcorp/r-kinesis-secret .
`aws ecr get-login --region us-east-1`
docker tag -f cardcorp/r-kinesis-secret:latest \
    ***.dkr.ecr.us-east-1.amazonaws.com/cardcorp/r-kinesis-secret:latest
docker push ***.dkr.ecr.us-east-1.amazonaws.com/cardcorp/r-kinesis-secret:latest
```

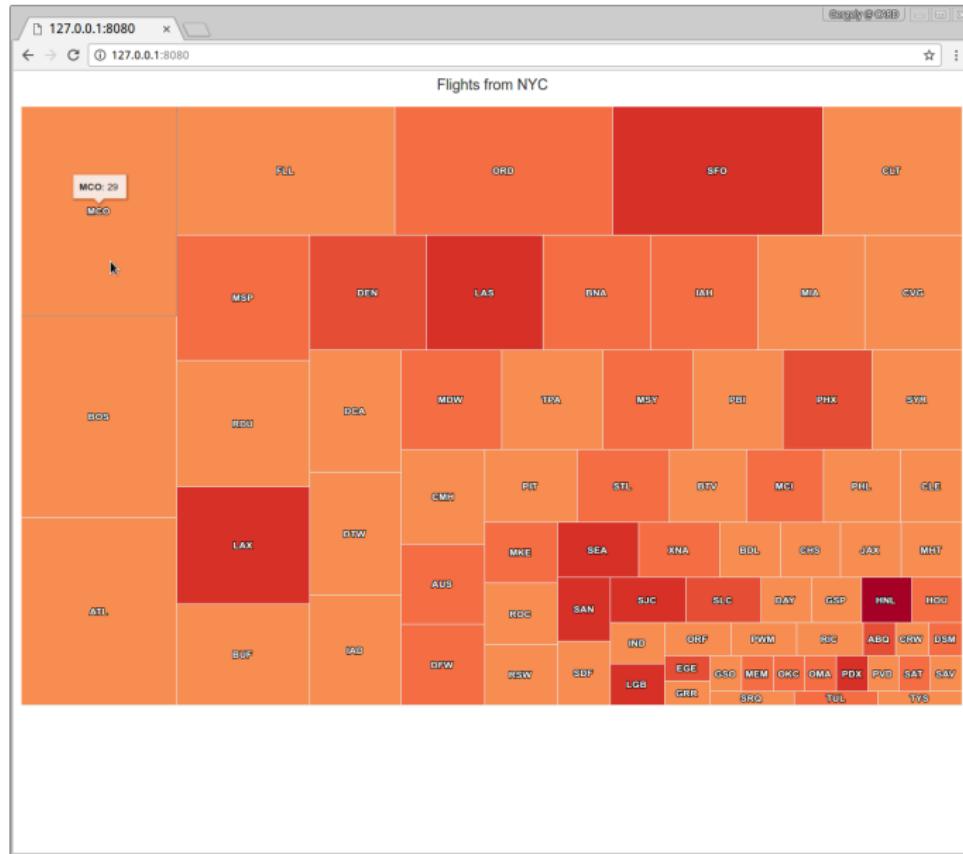
```
library(treemap);library(highcharter);library(nycflights13)
library(rredis);redisConnect(host = '***', port = '***')

ui      <- shinyUI(highchartOutput('treemap', height = '800px'))
server <- shinyServer(function(input, output, session) {

  destinations <- reactive({
    reactiveTimer(2000)()
    flights <- redisMGet(redisKeys('flight:*'))
    flights <- data.frame(faa = sub('^flight:', '', names(flights)),
                           N    = as.numeric(flights))
    merge(flights, airports, by = 'faa')
  })

  output$treemap <- renderHighchart({
    tm <- treemap(destinations(), index = c('faa'),
                  vSize = 'N', vColor = 'tz',
                  type = 'value', draw = FALSE)
    hc_title(hctreemap(tm, animation = FALSE), text = 'Flights from NYC')
  })
}

shinyApp(ui = ui, server = server)
```



- AWR repo:
  - 9.7 GB
  - 388 tags/versions
  - GitLab + CI + drat

```
install.packages('AWR', repos = 'https://cardcorp.gitlab.io/AWR')
```

- AWR repo:
  - 9.7 GB
  - 388 tags/versions
  - GitLab + CI + drat

```
install.packages('AWR', repos = 'https://cardcorp.gitlab.io/AWR')
```

- Submitted to CRAN on
  - 2016-12-05

- AWR repo:
  - 9.7 GB
  - 388 tags/versions
  - GitLab + CI + drat

```
install.packages('AWR', repos = 'https://cardcorp.gitlab.io/AWR')
```

- Submitted to CRAN on
  - 2016-12-05
  - 2017-01-09
  - 2017-01-10
  - 2017-01-11
  - 2017-01-11
  - 2017-01-13

- AWR repo:
  - 9.7 GB
  - 388 tags/versions
  - GitLab + CI + drat

```
install.packages('AWR', repos = 'https://cardcorp.gitlab.io/AWR')
```

- Submitted to CRAN on
  - 2016-12-05
  - 2017-01-09
  - 2017-01-10
  - 2017-01-11
  - 2017-01-11
  - 2017-01-13
- Release cycle: 2 minor, ~125 patch versions in the past 12 months
- CI



```
> library(rJava)
> kc <- .jnew('com.amazonaws.services.s3.AmazonS3Client')
> kc$getS3AccountOwner()$getDisplayName()
[1] "foobar"
```



Because "S"  
is so 1992.