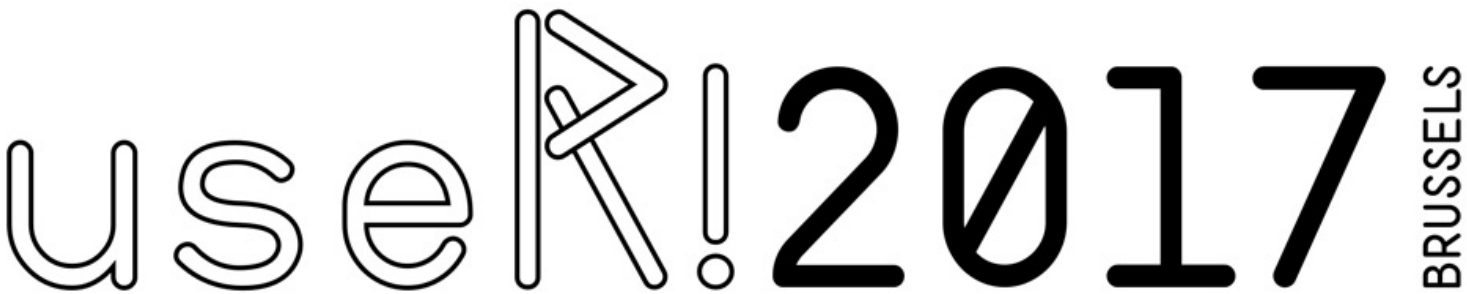
Stream processing with R in AWS

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

Gergely Daroczi

@daroczig

July 05, 2017

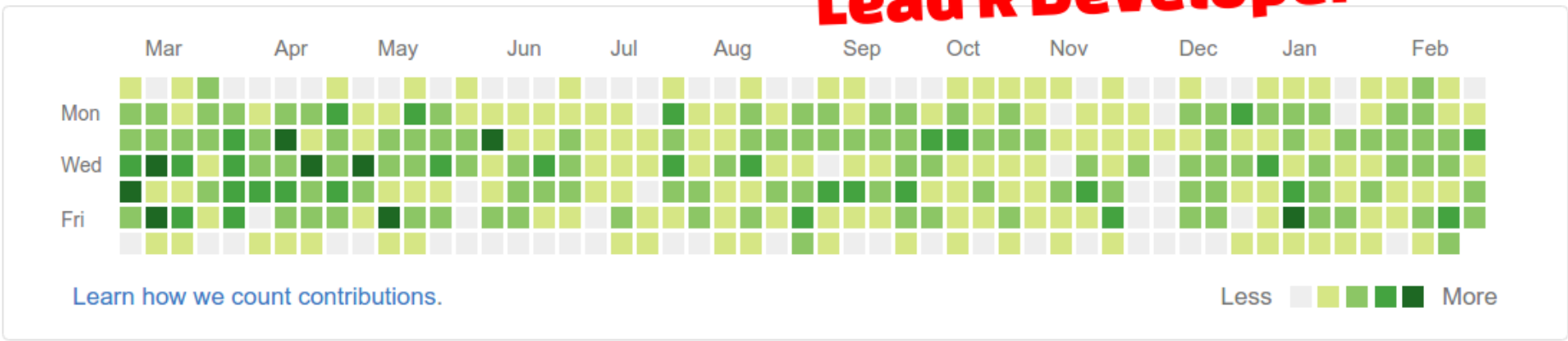


About me

useR!2017

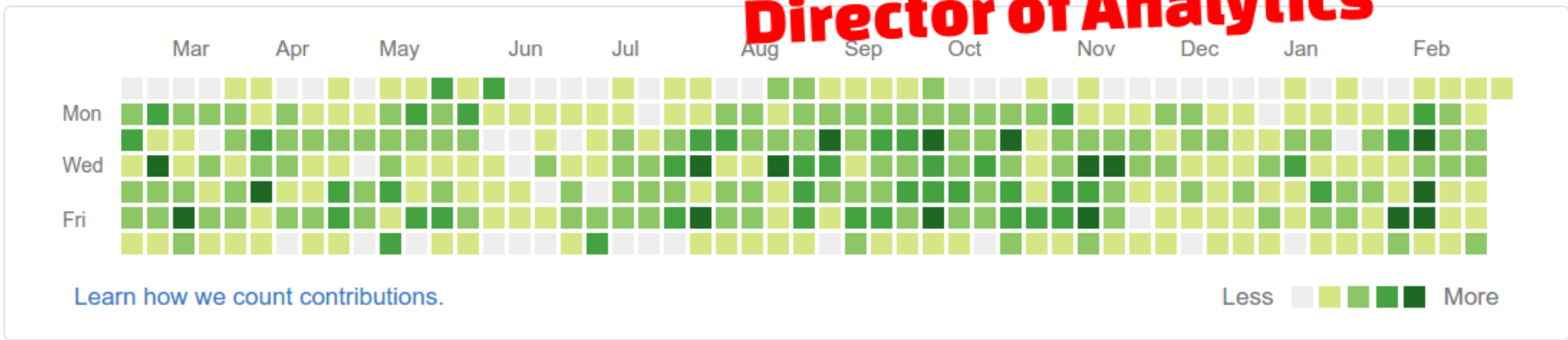
3,633 contributions in the year before last

Lead R Developer



3,470 contributions in the last year

Director of Analytics



About me

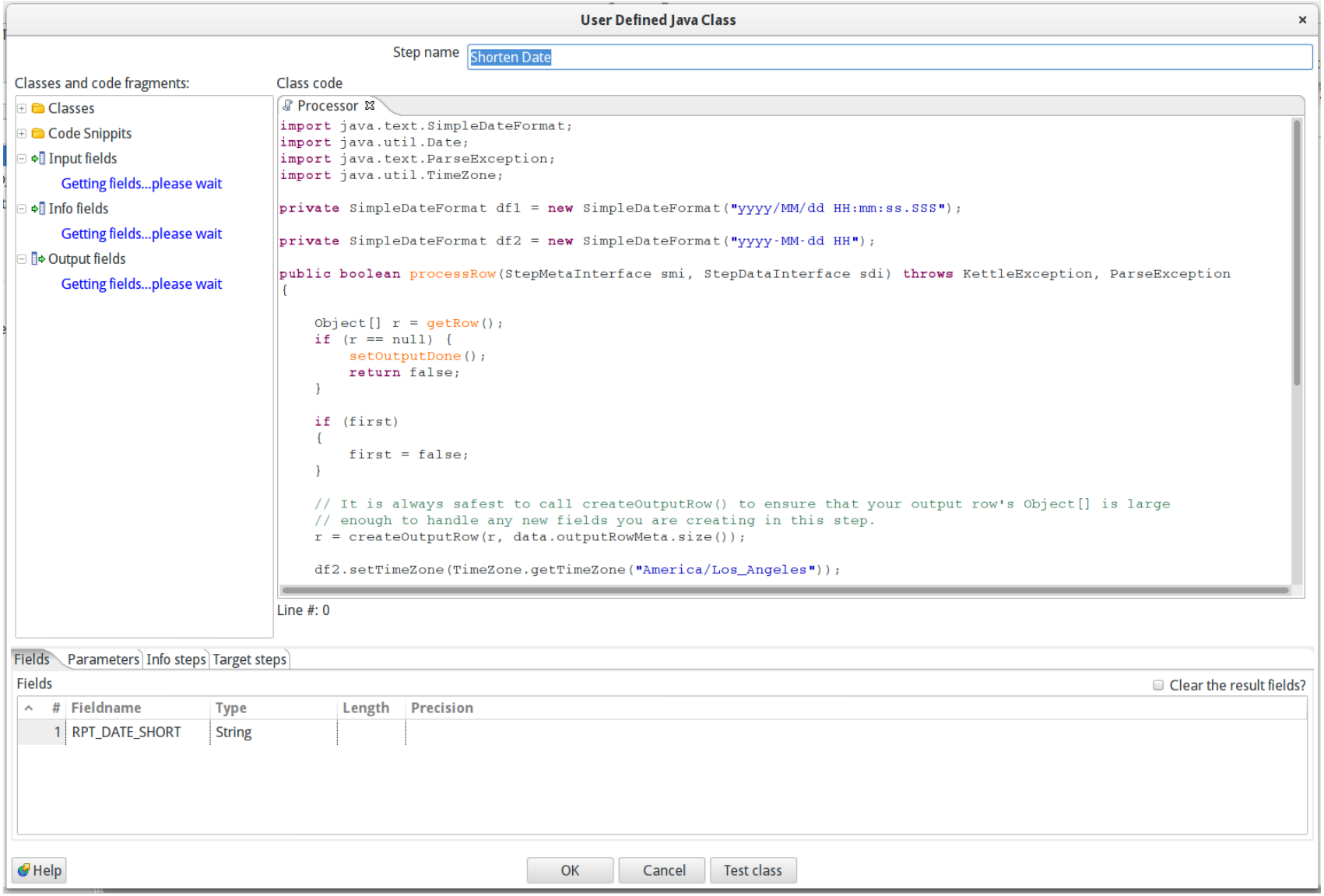
About me

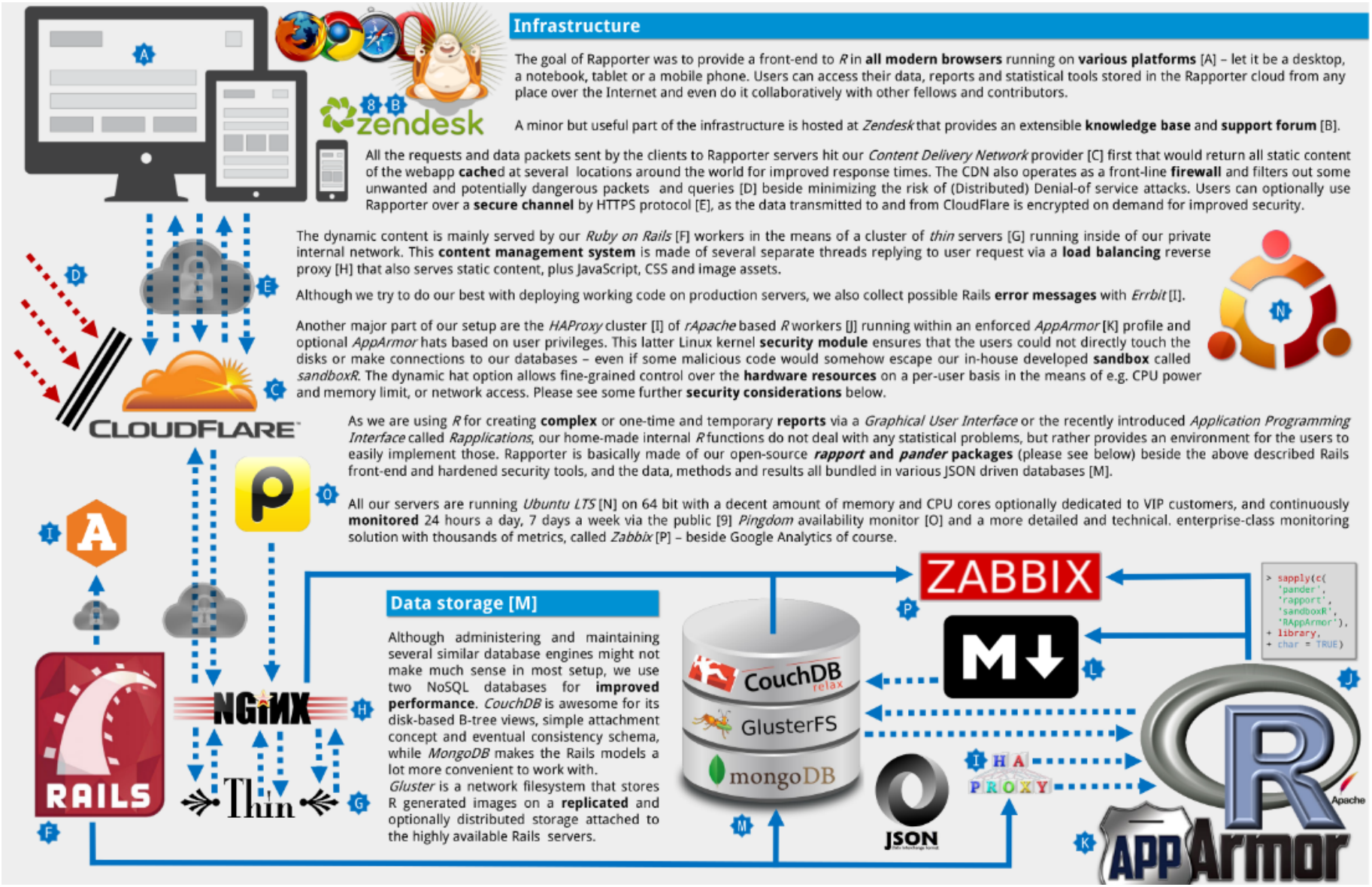
SYSTEM1

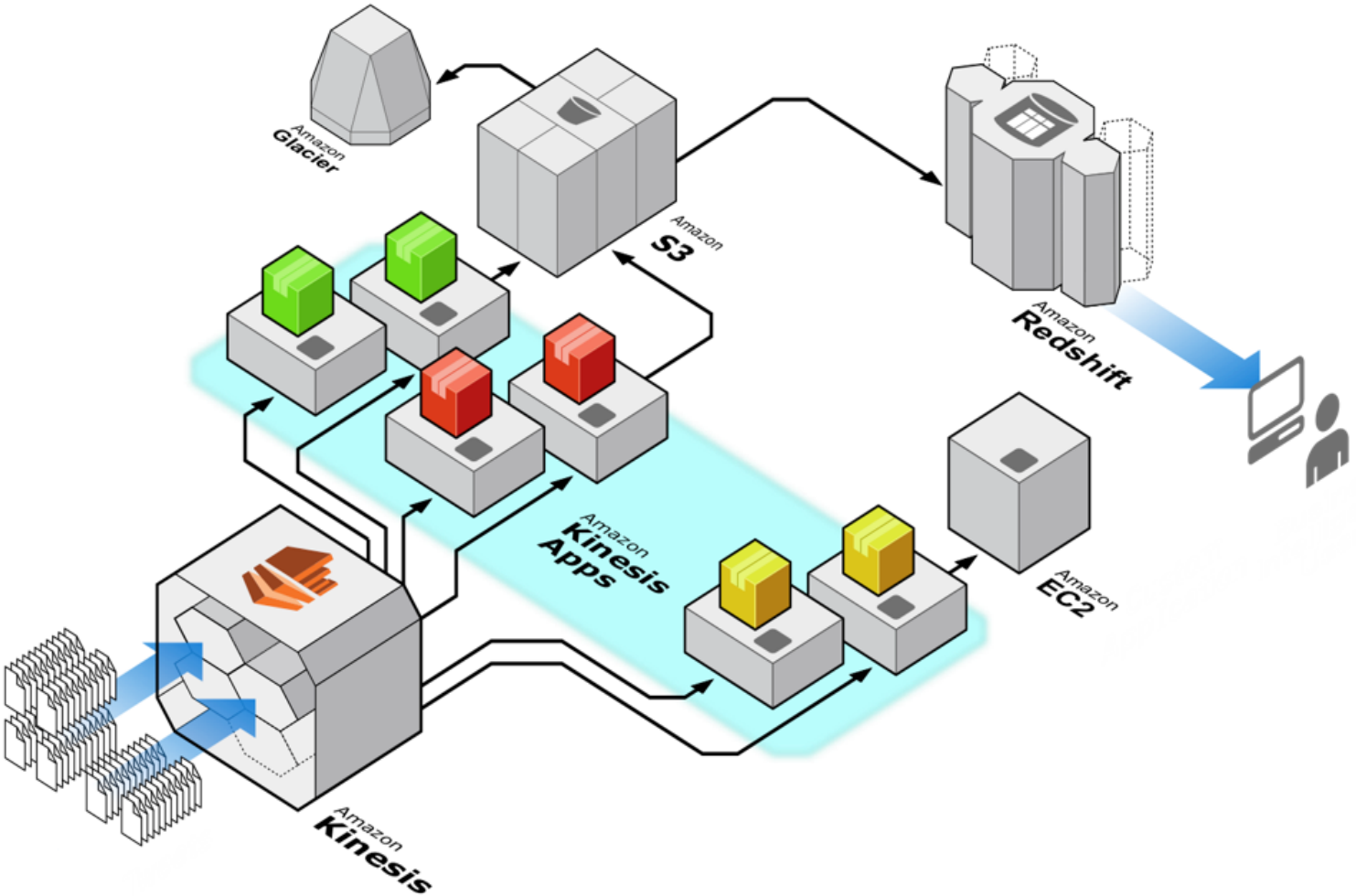
About me

Stream Processing . . . Why R?

useR!2017

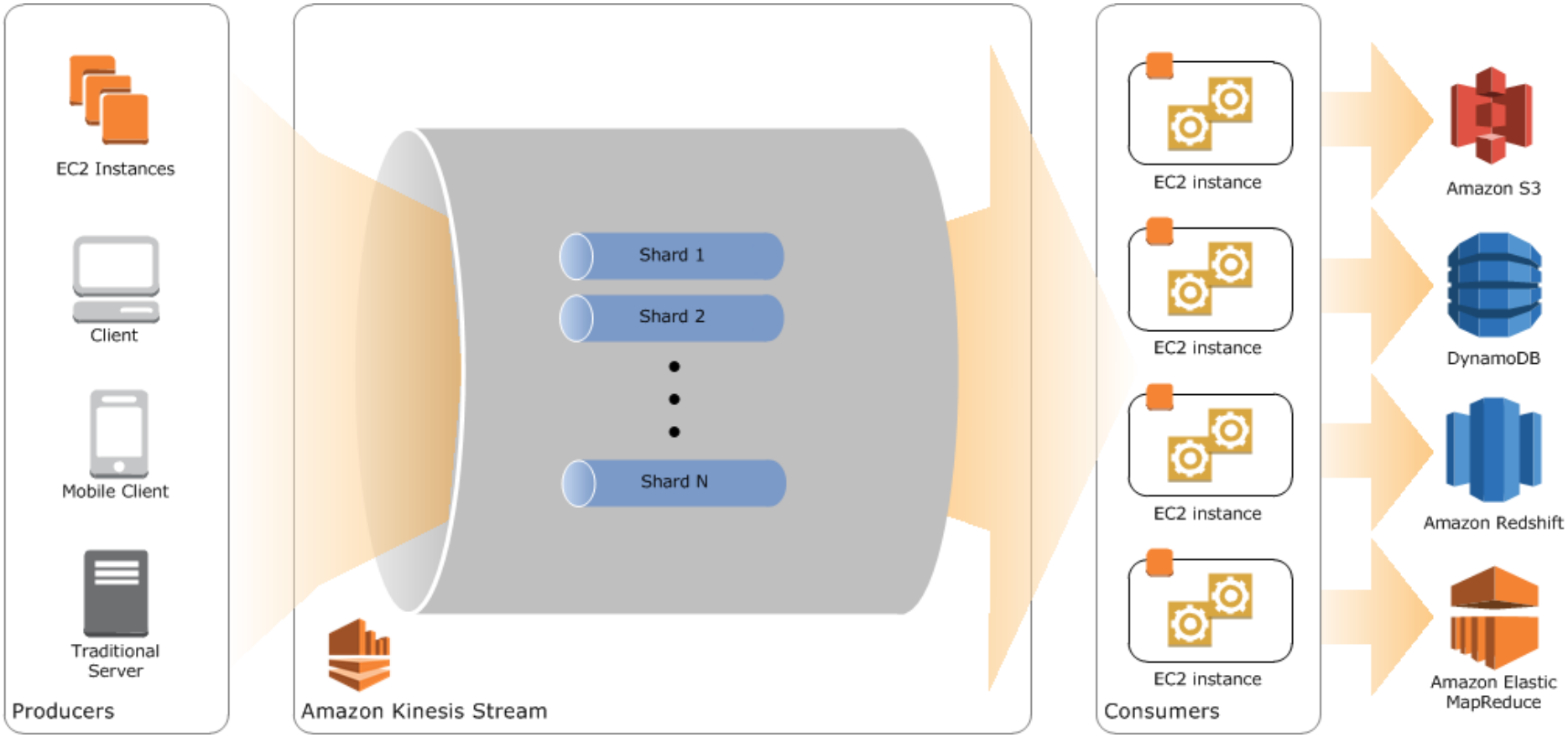



Infrastructure
The goal of Rapporter was to provide a front-end to R in all modern browsers running on various platforms [A] – let it be a desktop, a notebook, tablet or a mobile phone. Users can access their data, reports and statistical tools stored in the Rapporter cloud from any place over the Internet and even do it collaboratively with other fellows and contributors.
A minor but useful part of the infrastructure is hosted at Zendesk that provides an extensible knowledge base and support forum [B].
All the requests and data packets sent by the clients to Rapporter servers hit our Content Delivery Network provider [C] first that would return all static content of the webapp cached at several locations around the world for improved response times. The CDN also operates as a front-line firewall and filters out some unwanted and potentially dangerous packets and queries [D] beside minimizing the risk of (Distributed) Denial-of service attacks. Users can optionally use Rapporter over a secure channel by HTTPS protocol [E], as the data transmitted to and from CloudFlare is encrypted on demand for improved security.
The dynamic content is mainly served by our Ruby on Rails [F] workers in the means of a cluster of thin servers [G] running inside of our private internal network. This content management system is made of several separate threads replying to user request via a load balancing reverse proxy [H] that also serves static content, plus JavaScript, CSS and image assets.
Although we try to do our best with deploying working code on production servers, we also collect possible Rails error messages with Errbit [I].
Another major part of our setup are the HAProxy cluster [I] of rApache based R workers [J] running within an enforced AppArmor [K] profile and optional AppArmor hats based on user privileges. This latter Linux kernel security module ensures that the users could not directly touch the disks or make connections to our databases – even if some malicious code would somehow escape our in-house developed sandbox called sandboxR. The dynamic hat option allows fine-grained control over the hardware resources on a per-user basis in the means of e.g. CPU power and memory limit, or network access. Please see some further security considerations below.
As we are using R for creating complex or one-time and temporary reports via a Graphical User Interface or the recently introduced Application Programming Interface called Rapplications, our home-made internal R functions do not deal with any statistical problems, but rather provides an environment for the users to easily implement those. Rapporter is basically made of our open-source rapport and pander packages (please see below) beside the above described Rails front-end and hardened security tools, and the data, methods and results all bundled in various JSON driven databases [M].
All our servers are running Ubuntu LTS [N] on 64 bit with a decent amount of memory and CPU cores optionally dedicated to VIP customers, and continuously monitored 24 hours a day, 7 days a week via the public [9] Pingdom availability monitor [O] and a more detailed and technical, enterprise-class monitoring solution with thousands of metrics, called Zabbix [P] – beside Google Analytics of course.
Data storage [M]
Although administering and maintaining several similar database engines might not make much sense in most setup, we use two NoSQL databases for improved performance. CouchDB is awesome for its disk-based B-tree views, simple attachment concept and eventual consistency schema, while MongoDB makes the Rails models a lot more convenient to work with.
Gluster is a network filesystem that stores R generated images on a replicated and optionally distributed storage attached to the highly available Rails servers.
CLOUDFLARE
zendesk
ZABBIX
CouchDB
GlusterFS
mongoDB
NGINX
Thin
RAILS
JSON
HAPROXY
AppArmor
Apache
> sapply(c('pander', 'rapport', 'sandboxR', 'RAppArmor'), + library, + char = TRUE)



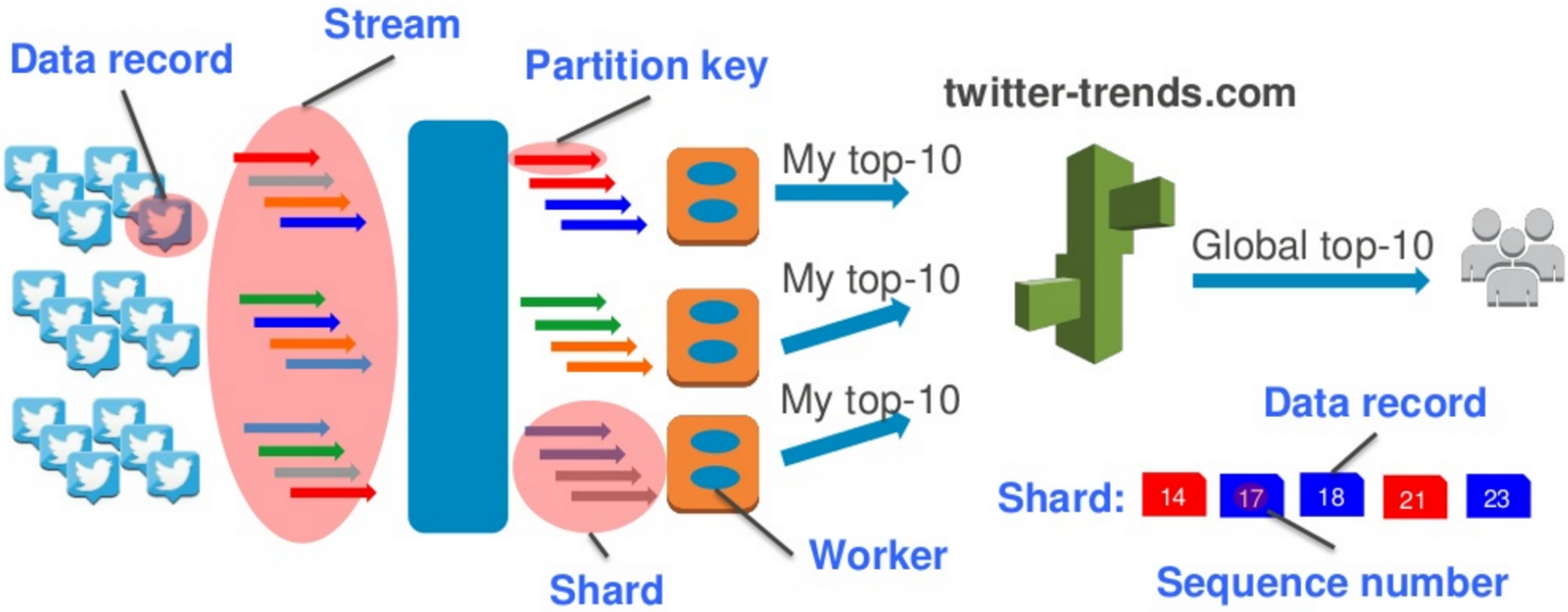
Source: Kinesis Product Details

Intro to Amazon Kinesis Streams



Source: Kinesis Developer Guide

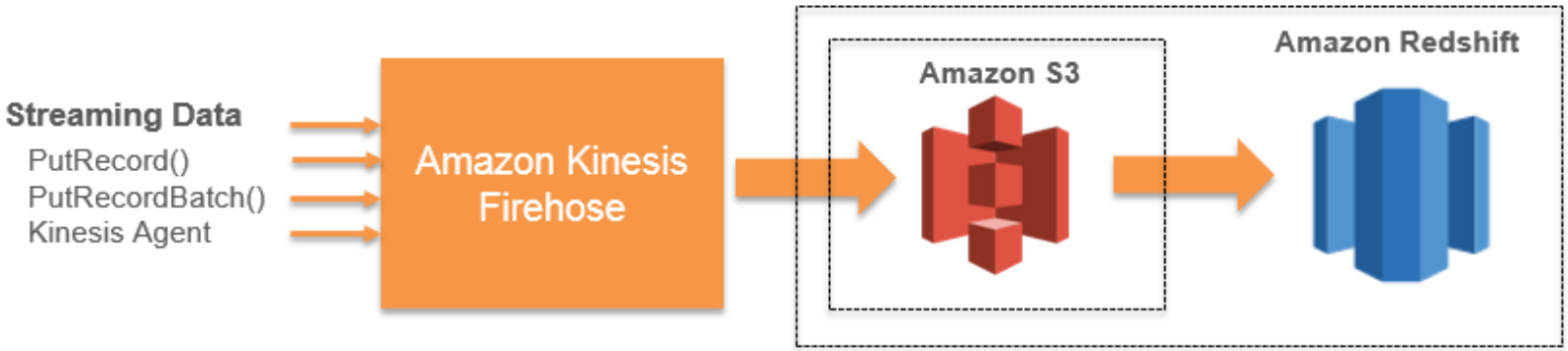
Intro to Amazon Kinesis Shards



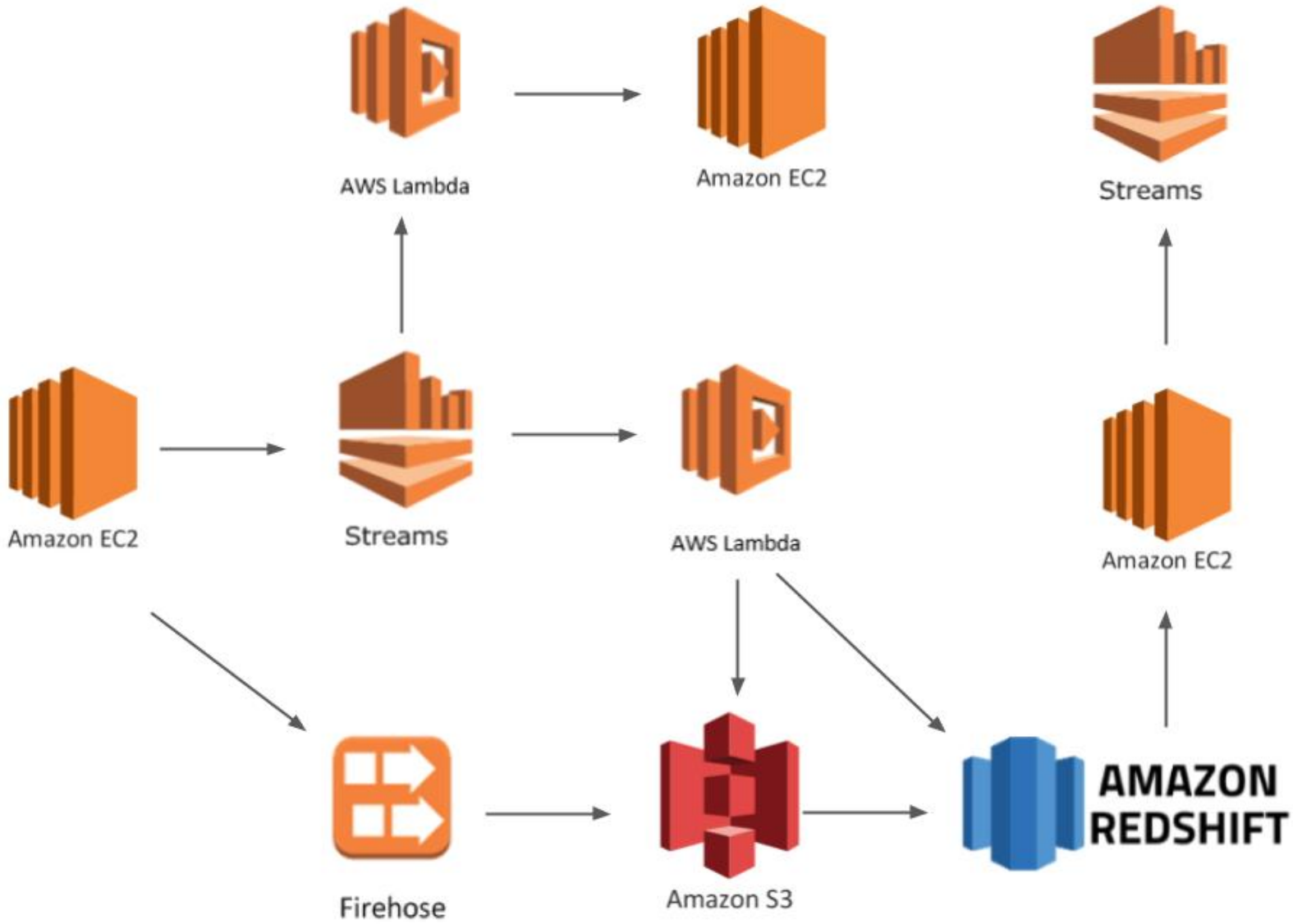
Source: AWS re:Invent 2013

A Very Deep Learning


S3
FIREHOSE
REDSHIFT
Capture & submit streaming data to Firehose
Firehose loads streaming data continuously into S3 and Redshift
Analyze streaming data using your favorite BI tools


Streaming Data
PutRecord()
PutRecordBatch()
Kinesis Agent
Amazon Kinesis Firehose
Amazon S3
Amazon Redshift

S4: Multiple Dispatch



How to Communicate with Kinesis

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 (!)

Now We Need an R Client!

useR!2017

```
> 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: 49562894160449444332153346371084313572324361665
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"
```

Managing Shards via the Java SDK

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: 170141183460469231731
SequenceNumberRange: {
StartingSequenceNumber: 49562894160449444332153346340517833149186116289174700050,
EndingSequenceNumber: 49562894160460594704752611652087392082504911751749828626}},
{ShardId: shardId-000000000002,
ParentShardId: shardId-000000000000,
AdjacentParentShardId: shardId-000000000001,
HashKeyRange: {StartingHashKey: 0,EndingHashKey: 340282366920938463463374607431768
SequenceNumberRange: {StartingSequenceNumber: 49562904991497673099704924344727019521
```

Amazon Kinesis Client Library

- 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

Messages from the KCL

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.

R Script Interacting with KCL

useR!2017

```
#!/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)))

}
```

R Script Interacting with KCL

useR!2017

```
#!/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)))

}
```

Get rid of the bugs and the boilerplate

useR!2017

```
> 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)
```

Add content to the boilerplate

useR!2017

Business logic coded in R (demo_app.R):

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

Add content to the boilerplate

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.

Add content to the boilerplate

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
```

"Advanced" AWR.Kinesis features

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')
```

Let's run it locally!

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

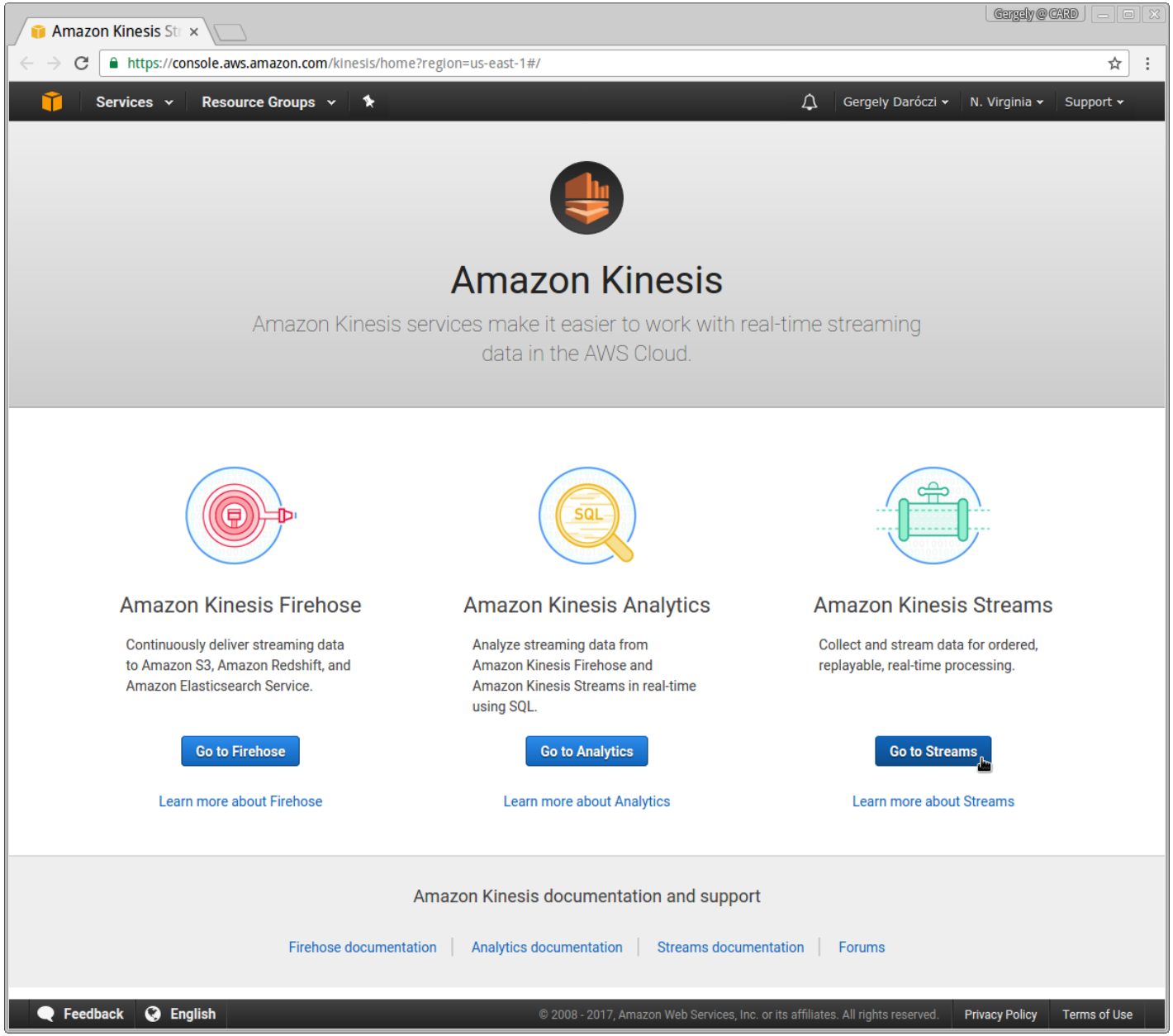
Let's run it locally!

Note

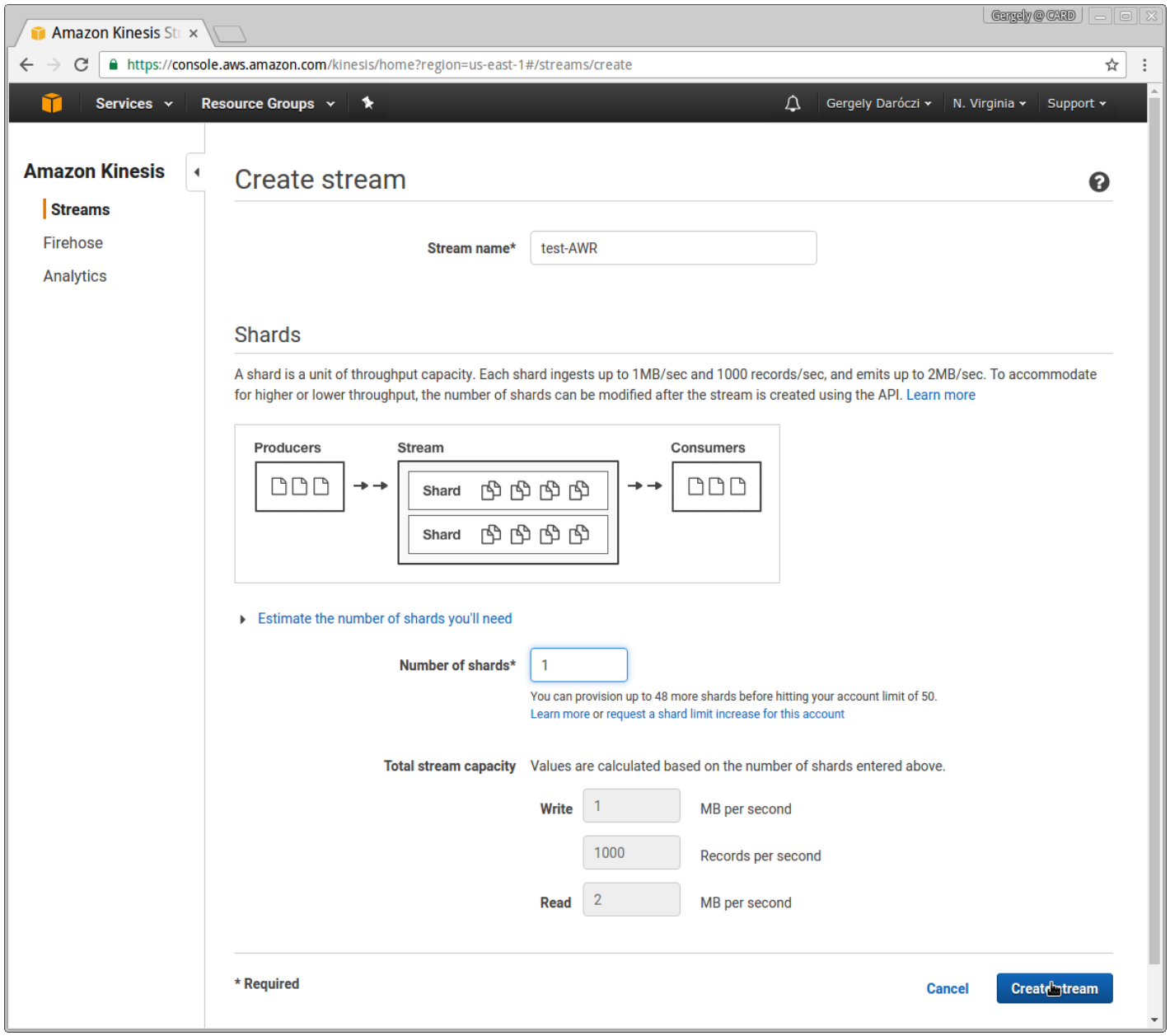
In theory you could, but this is not something you should run in RStudio.

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

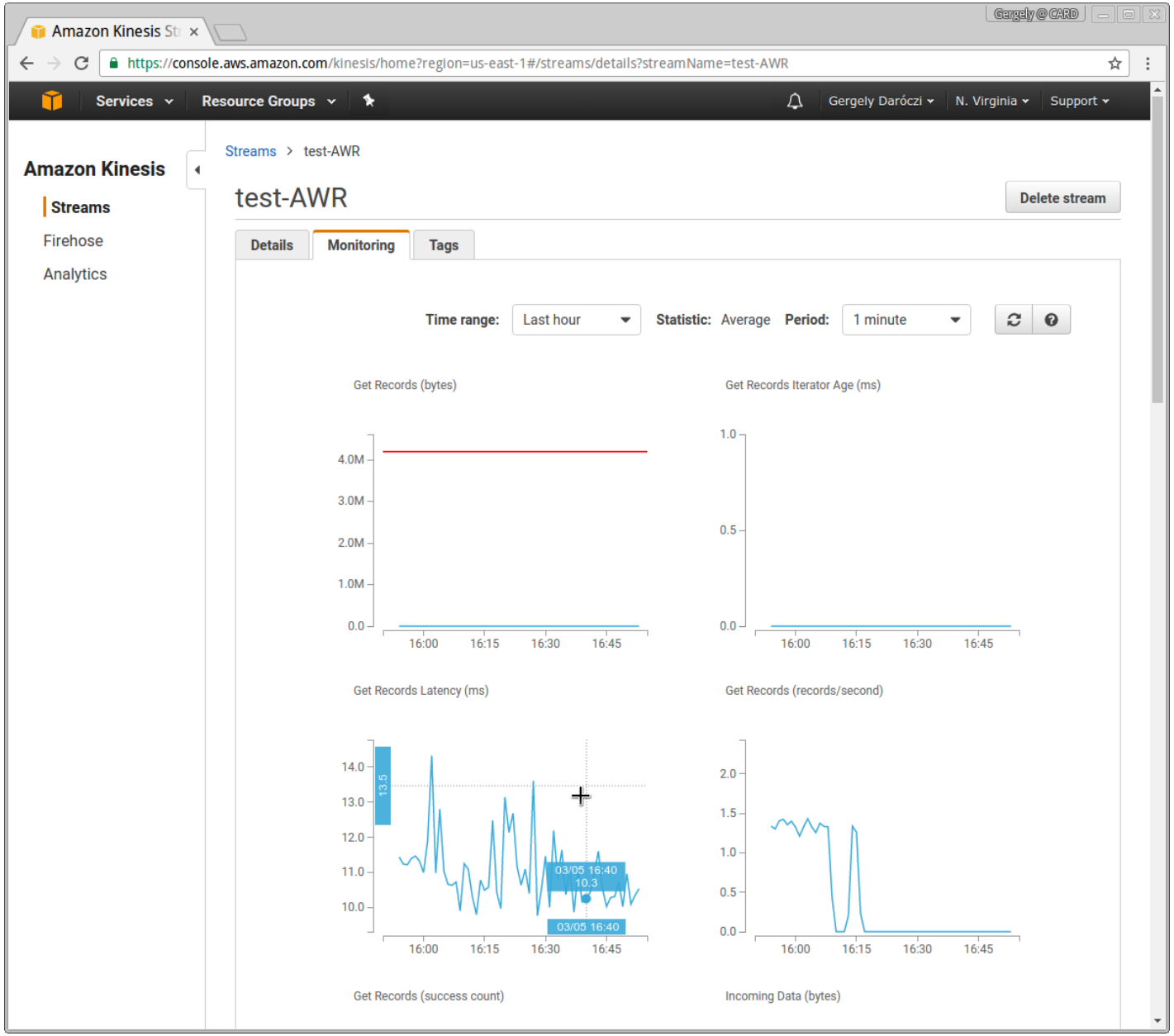
Create a Kinesis Stream



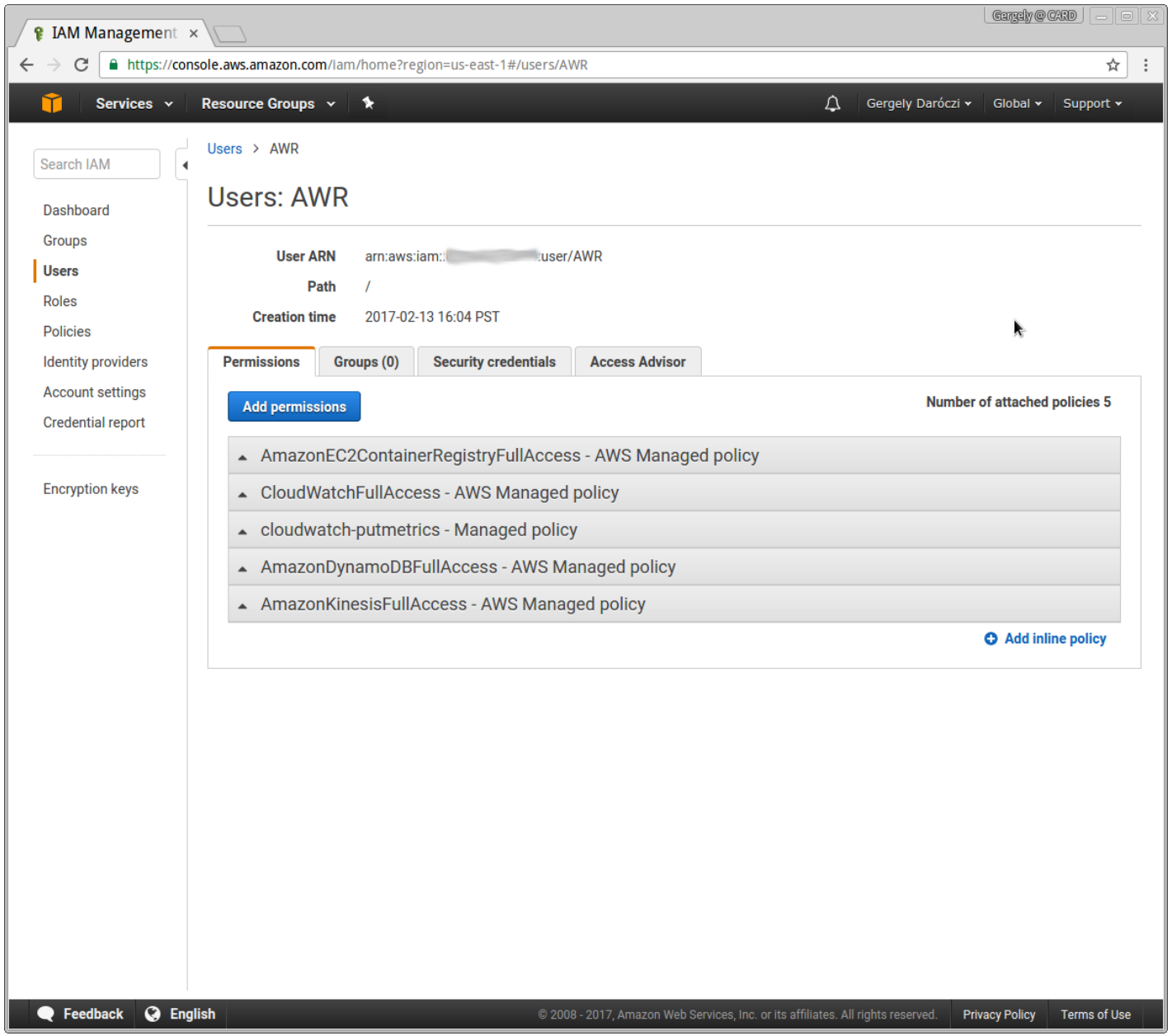
Create a Kinesis Stream



Check the Kinesis Stream



Create an IAM user



Write Data to the Stream from R

useR!2017

```
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

Minibuf-1

File Edit Options Buffers Tools Minibuf Help

```
> library(rJava)
> .jcall("java/lang/System", "S", "setProperty", "aws.profile", "personal")
> library(AWR.Kinesis); library(jsonlite); library(futile.logger); library(nycflights13)
> while (TRUE) {
+     flight <- flights[sample(1:nrow(flights), 1), ]
+     res <- kinesis_put_record('test-AWR', region = 'us-east-1', data = toJSON(flight),
+                               partitionKey = flight$dest)
+     flog.info(paste('Pushed a new flight to Kinesis:', res$sequenceNumber))
+ }
INFO [2017-03-06 21:47:16] Pushed a new flight to Kinesis: 49571082550613725758991014186133813750336290428328869938
INFO [2017-03-06 21:47:17] Pushed a new flight to Kinesis: 49571082550613725758991014186380434617537674848688406578
INFO [2017-03-06 21:47:17] Pushed a new flight to Kinesis: 49571082550613725758991014186610130523264454391882580018
INFO [2017-03-06 21:47:17] Pushed a new flight to Kinesis: 49571082550613725758991014186870049574481599664444407858
INFO [2017-03-06 21:47:18] Pushed a new flight to Kinesis: 49571082550613725758991014187125132922420286489026887730
INFO [2017-03-06 21:47:18] Pushed a new flight to Kinesis: 49571082550613725758991014187394723380194348794986364978
INFO [2017-03-06 21:47:19] Pushed a new flight to Kinesis: 49571082550613725758991014187636508544117274698647076914
INFO [2017-03-06 21:47:19] Pushed a new flight to Kinesis: 49571082550613725758991014187857741969106751837618307122
INFO [2017-03-06 21:47:20] Pushed a new flight to Kinesis: 49571082550613725758991014188075348616637385157784895538
INFO [2017-03-06 21:47:20] Pushed a new flight to Kinesis: 49571082550613725758991014188250642860481506388117291058
INFO [2017-03-06 21:47:21] Pushed a new flight to Kinesis: 49571082550613725758991014188505726208420193143980294194
INFO [2017-03-06 21:47:21] Pushed a new flight to Kinesis: 49571082550613725758991014188814002292421923652249845810
INFO [2017-03-06 21:47:21] Pushed a new flight to Kinesis: 49571082550613725758991014189015892904297566724425777202
INFO [2017-03-06 21:47:22] Pushed a new flight to Kinesis: 49571082550613725758991014189266140548957795032309432370
INFO [2017-03-06 21:47:22] Pushed a new flight to Kinesis: 49571082550613725758991014189516388193618023271473610802
INFO [2017-03-06 21:47:23] Pushed a new flight to Kinesis: 49571082550613725758991014189794441132129388119094984754
INFO [2017-03-06 21:47:23] Pushed a new flight to Kinesis: 49571082550613725758991014190015674557118865258066214962
INFO [2017-03-06 21:47:24] Pushed a new flight to Kinesis: 49571082550613725758991014190342084528414815135236882482
INFO [2017-03-06 21:47:24] Pushed a new flight to Kinesis: 49571082550613725758991014190545184066110072905306996786
INFO [2017-03-06 21:47:25] Pushed a new flight to Kinesis: 49571082550613725758991014190772462120197623190151757874
INFO [2017-03-06 21:47:25] Pushed a new flight to Kinesis: 49571082550613725758991014190965890251335963858104746034
INFO [2017-03-06 21:47:25] Pushed a new flight to Kinesis: 49571082550613725758991014191170198714850836257349566514
INFO [2017-03-06 21:47:26] Pushed a new flight to Kinesis: 49571082550613725758991014191397476768938386542194327602
INFO [2017-03-06 21:47:26] Pushed a new flight to Kinesis: 49571082550613725758991014191629590526304395412457390130
INFO [2017-03-06 21:47:27] Pushed a new flight to Kinesis: 49571082550613725758991014191910061316454989380989222962
INFO [2017-03-06 21:47:27] Pushed a new flight to Kinesis: 49571082550613725758991014192179651774229051686948700210
INFO [2017-03-06 21:47:28] Pushed a new flight to Kinesis: 49571082550613725758991014192373079905367392423621165106
INFO [2017-03-06 21:47:28] Pushed a new flight to Kinesis: 49571082550613725758991014192595522256176484191767101490
INFO [2017-03-06 21:47:28] Pushed a new flight to Kinesis: 49571082550613725758991014192772025425840220119993679922
INFO [2017-03-06 21:47:29] Pushed a new flight to Kinesis: 49571082550613725758991014192981169592633550967217848370
INFO [2017-03-06 21:47:29] Pushed a new flight to Kinesis: 49571082550613725758991014193206029795081871993713197106
INFO [2017-03-06 21:47:30] Pushed a new flight to Kinesis: 49571082550613725758991014193413965036055588280482136114
INFO [2017-03-06 21:47:30] Pushed a new flight to Kinesis: 49571082550613725758991014193609811018833158206784536626
INFO [2017-03-06 21:47:30] Pushed a new flight to Kinesis: 49571082550613725758991014193818955185626489122728181810
INFO [2017-03-06 21:47:31] Pushed a new flight to Kinesis: 49571082550613725758991014194043815388074810149223530546
INFO [2017-03-06 21:47:31] Pushed a new flight to Kinesis: 49571082550613725758991014194267466664703516546544173106
INFO [2017-03-06 21:47:32] Pushed a new flight to Kinesis: 49571082550613725758991014194544310677395266696271364146
INFO [2017-03-06 21:47:32] Pushed a new flight to Kinesis: 49571082550613725758991014194752245918368982914320826418
INFO [2017-03-06 21:47:33] Pushed a new flight to Kinesis: 49571082550613725758991014195169325326136030048313933874
INFO [2017-03-06 21:47:34] Pushed a new flight to Kinesis: 49571082550613725758991014195478810335957375117038714930
INFO [2017-03-06 21:47:34] Pushed a new flight to Kinesis: 49571082550613725758991014195897098669544036880206528562
INFO [2017-03-06 21:47:35] Pushed a new flight to Kinesis: 49571082550613725758991014196216255085922989823289590 26
INFO [2017-03-06 21:47:35] Pushed a new flight to Kinesis: 49571082550613725758991014196406056439601795831477305394
INFO [2017-03-06 21:47:36] Pushed a new flight to Kinesis: 49571082550613725758991014196589813164183219466032644146
INFO [2017-03-06 21:47:36] Pushed a new flight to Kinesis: 49571082550613725758991014196811046589172696673723351090
INFO [2017-03-06 21:47:36] Pushed a new flight to Kinesis: 49571082550613725758991014197006892571950266600025751602
INFO [2017-03-06 21:47:37] Pushed a new flight to Kinesis: 49571082550613725758991014197254722364971265580840517682
INFO [2017-03-06 21:47:37] Pushed a new flight to Kinesis: 49571082550613725758991014197475955789960742857250701362
INFO [2017-03-06 21:47:38] Pushed a new flight to Kinesis: 49571082550613725758991014197724994508801356467240173618
INFO [2017-03-06 21:47:38] Pushed a new flight to Kinesis: 49571082550613725758991014197965570746904667741726179378
INFO [2017-03-06 21:47:39] Pushed a new flight to Kinesis: 49571082550613725758991014198145700694027247488757399602
INFO [2017-03-06 21:47:39] Pushed a new flight to Kinesis: 49571082550613725758991014198412873300162080536367464498
INFO [2017-03-06 21:47:40] Pushed a new flight to Kinesis: 49571082550613725758991014198640151354249630889931702322
INFO [2017-03-06 21:47:41] Pushed a new flight to Kinesis: 49571082550613725758991014199353417587822262171727822898
INFO [2017-03-06 21:47:41] Pushed a new flight to Kinesis: 49571082550613725758991014199624216971415939106862006322
 * 12M U:  R-kinesis-putrecord  iESS [R]: run                                                    109137 :  0  99%
M-x butterfly
```

This Kinesis app is being run

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

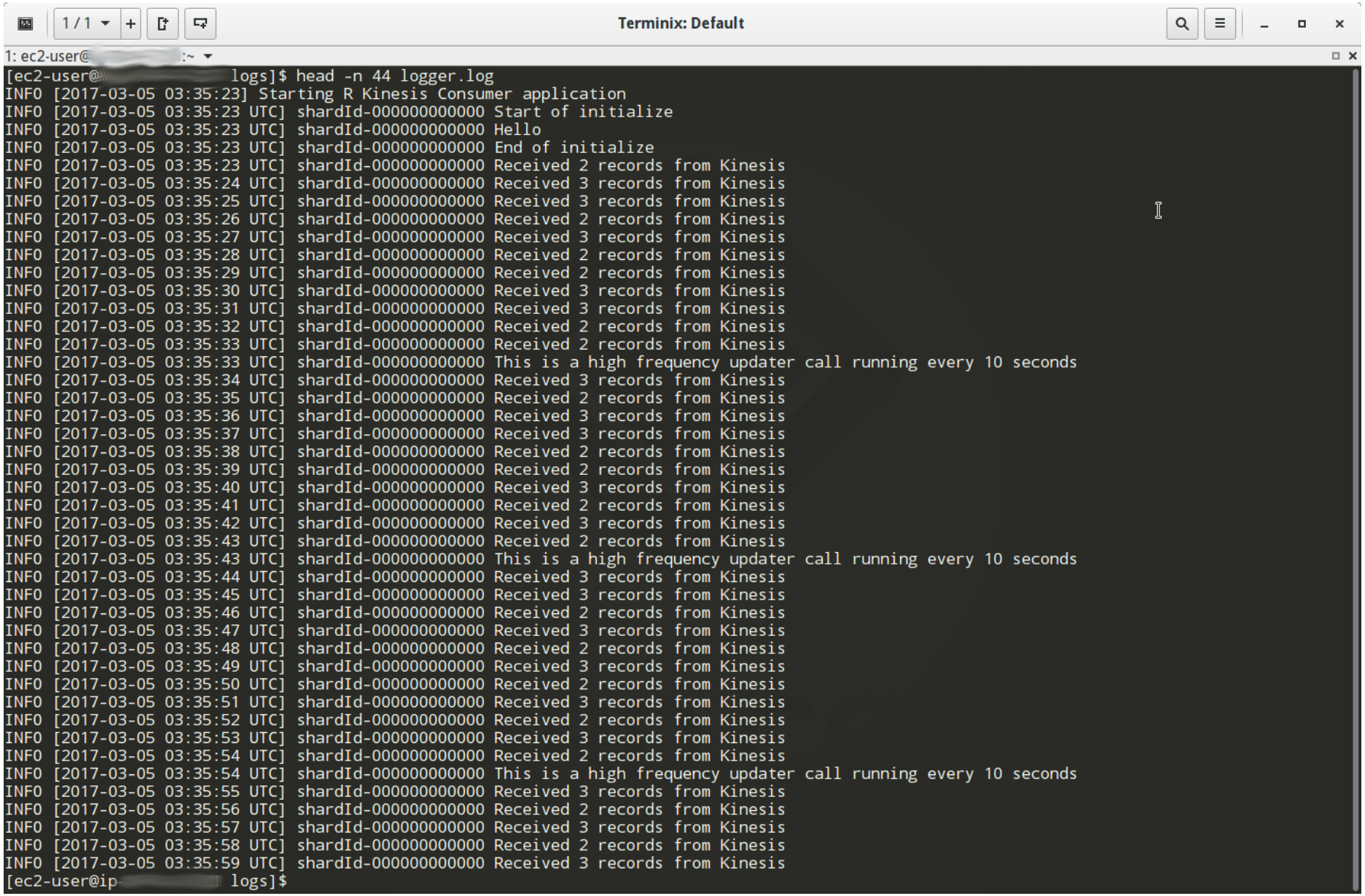
Terminix: Default

1: daroczig@gergely-CARD: ~/Projects/card-rocker/r-kinesis-example/files

```
~/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: [redacted]
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: [redacted]
Mar 05, 2017 5:32:34 PM com.amazonaws.services.kinesis.multilang.MultiLangDaemonConfig prepare
INFO: Using credentials with access key id: [redacted]
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 bef5
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: Worke[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=[redacted], parentSh
ardIds=[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

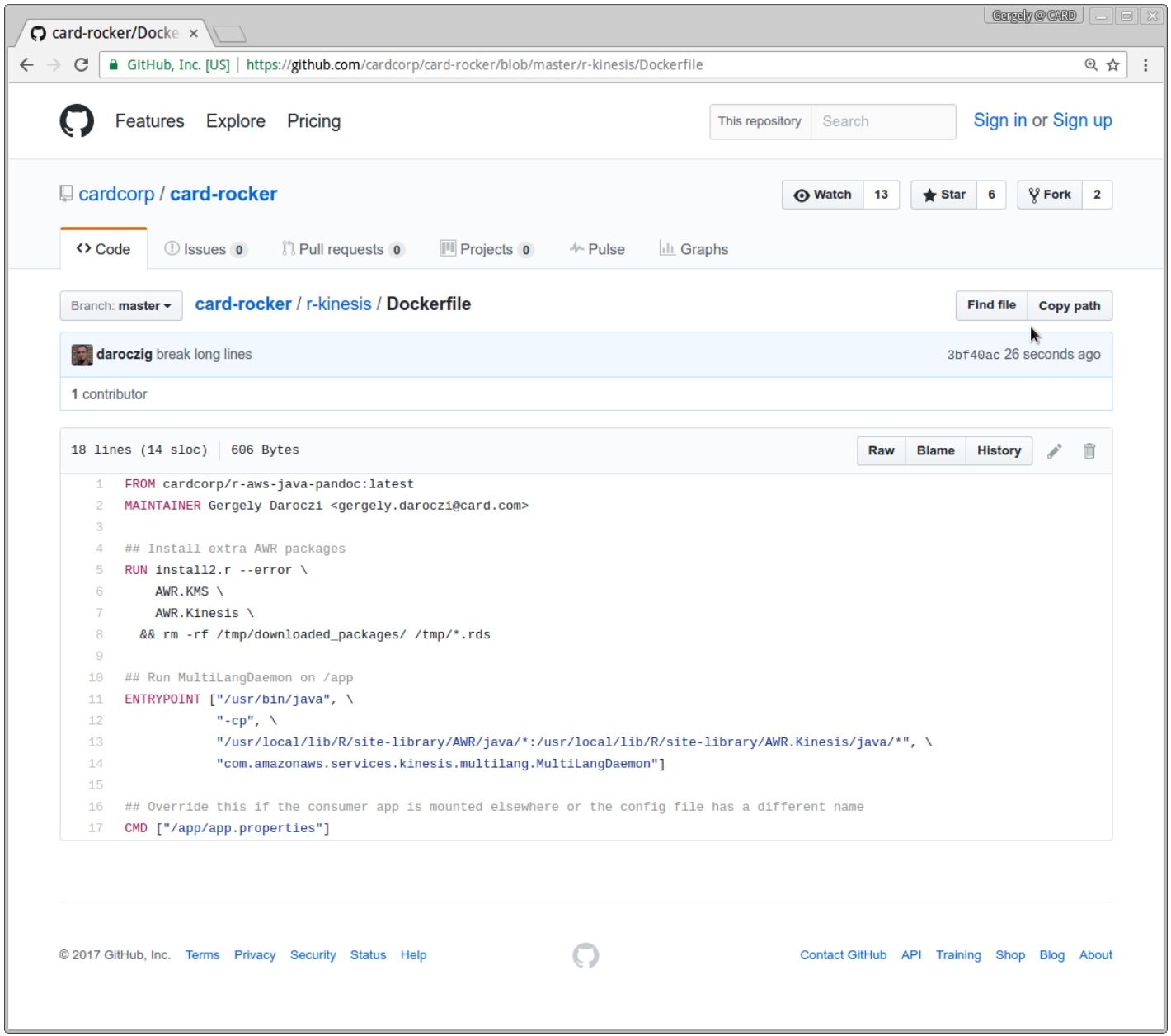
useR!2017



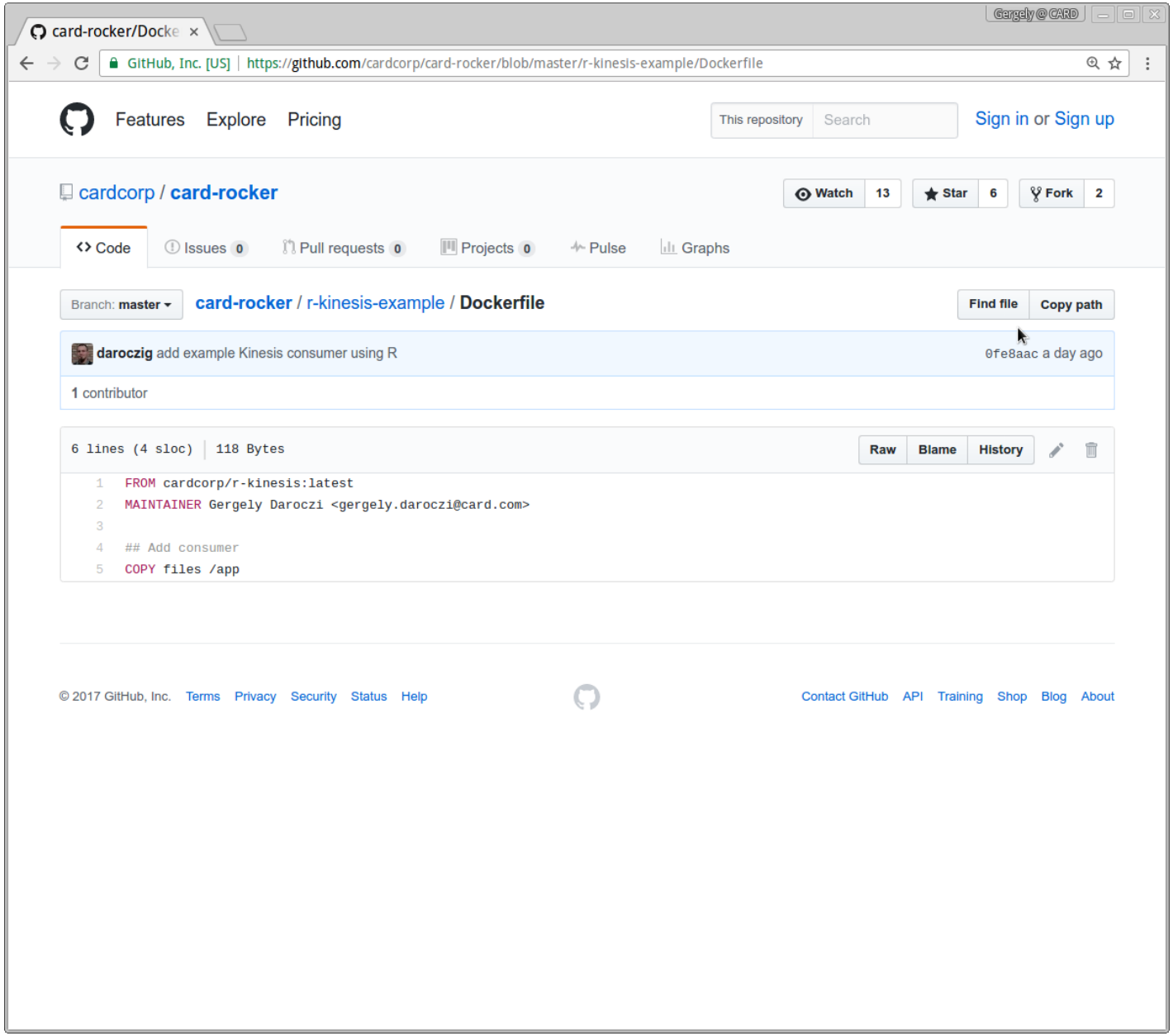
Let's run it in AWS!

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

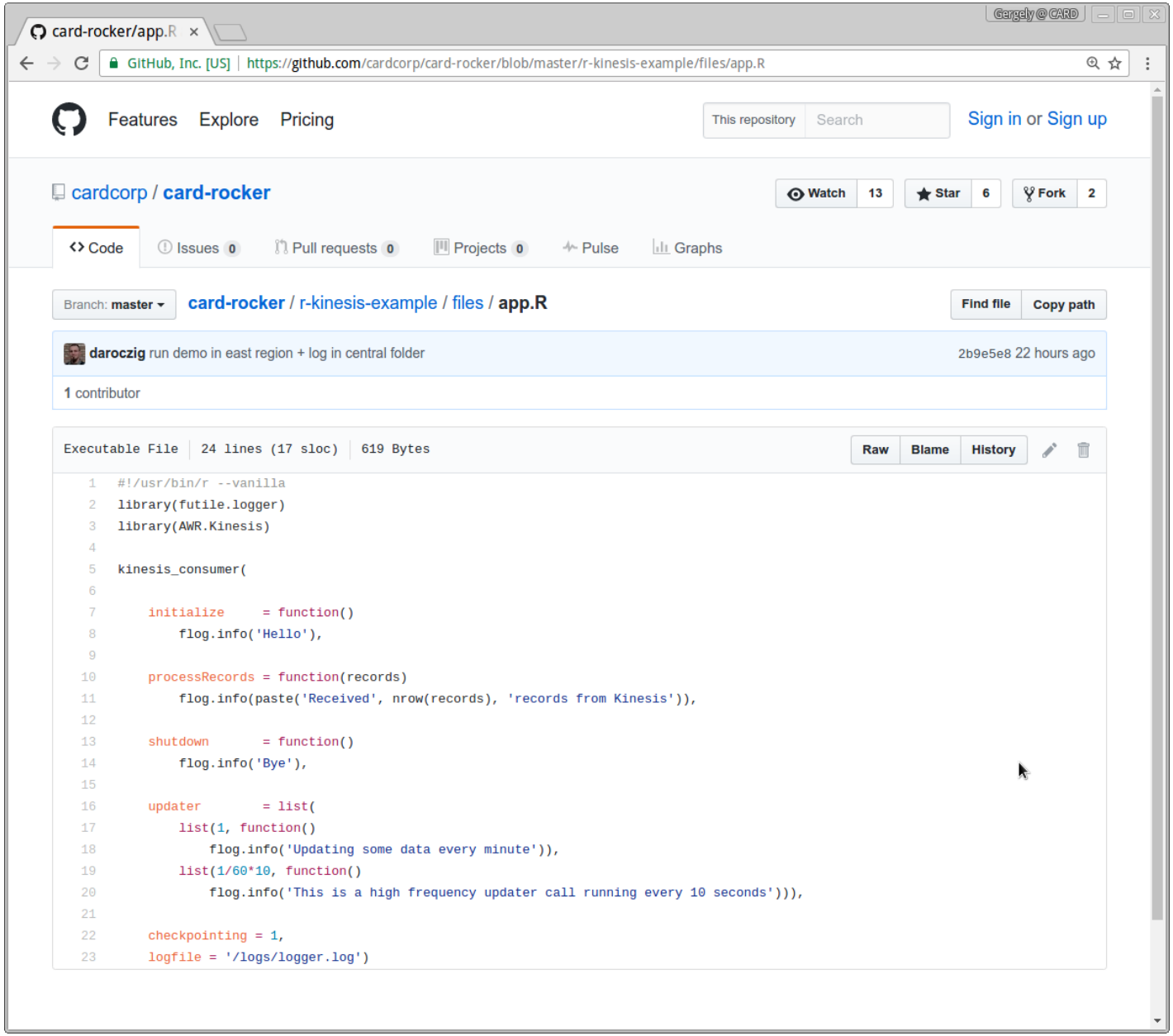


Dockerize your Kinesis Consumer



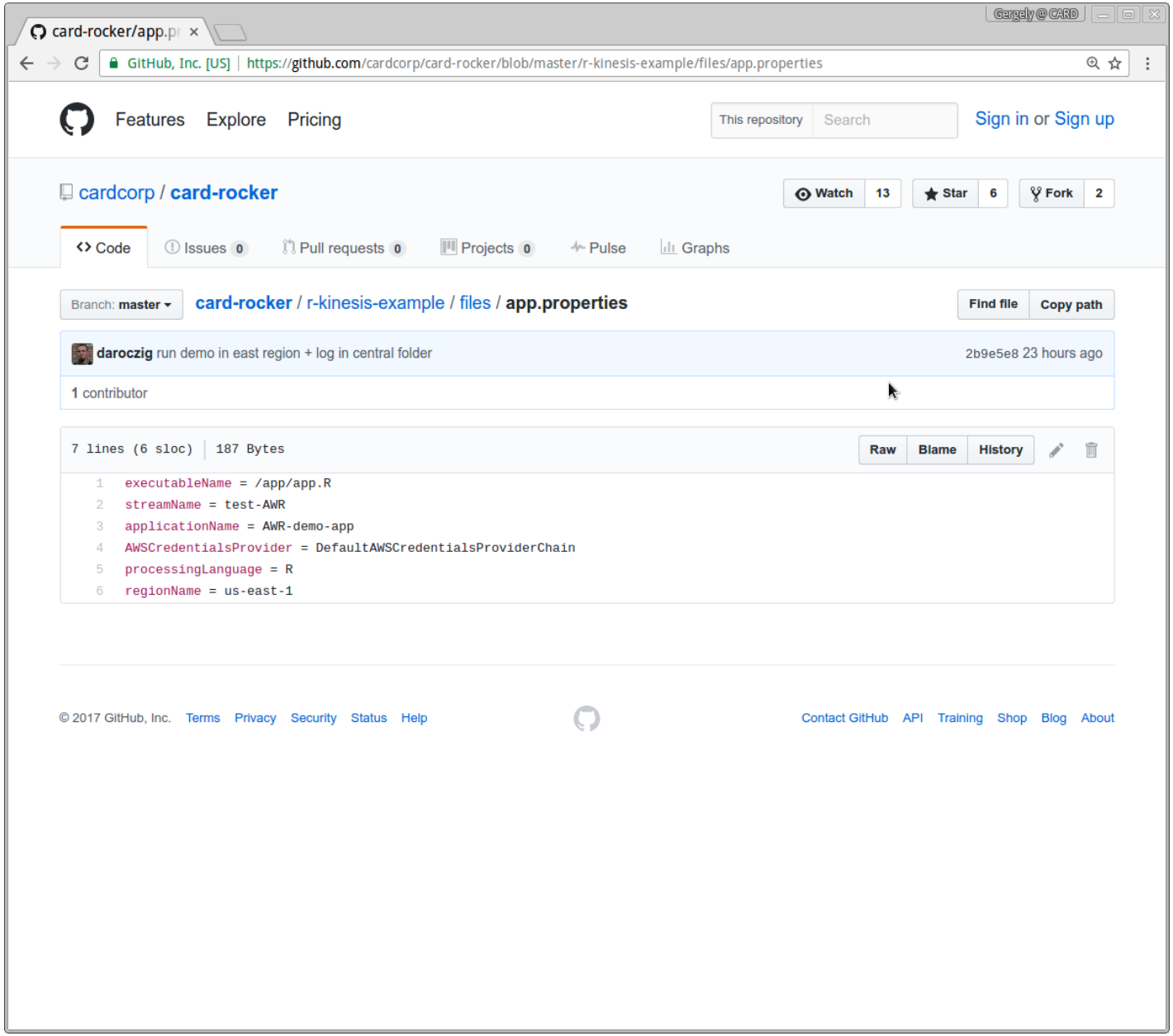
Dockerize your Kinesis Consumer

useR!2017

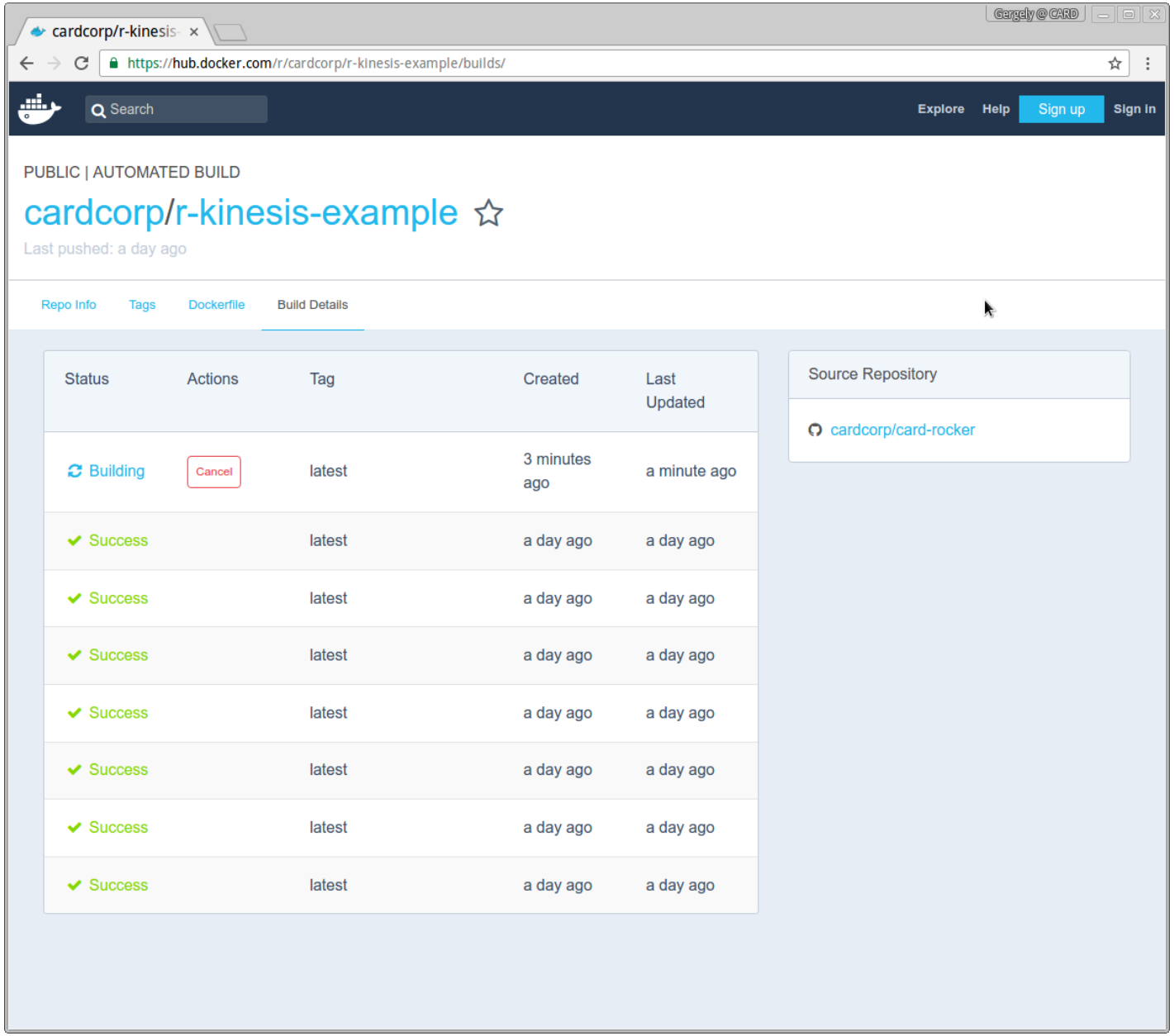


Dockerize your Kinesis Consumer

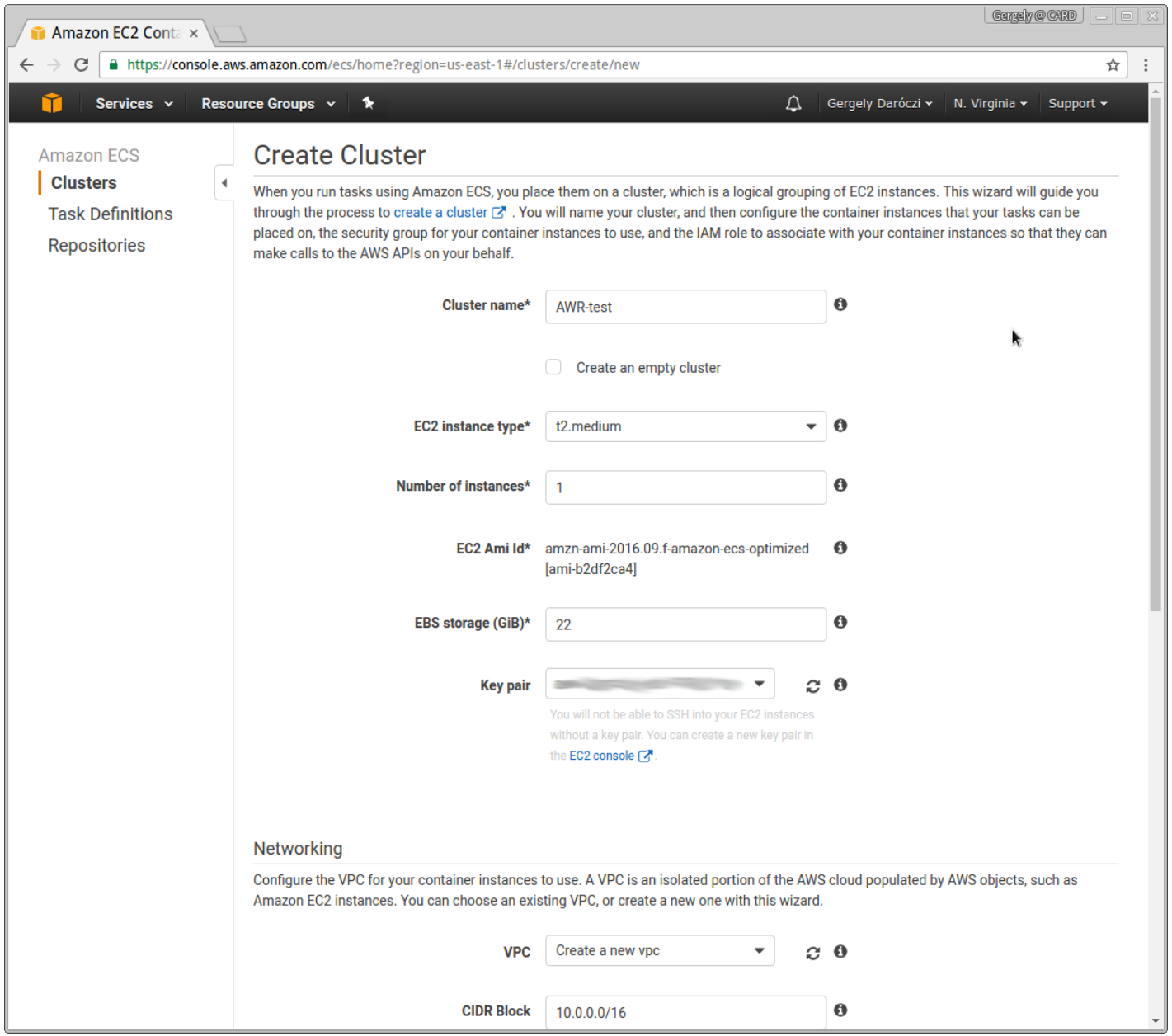
useR!2017



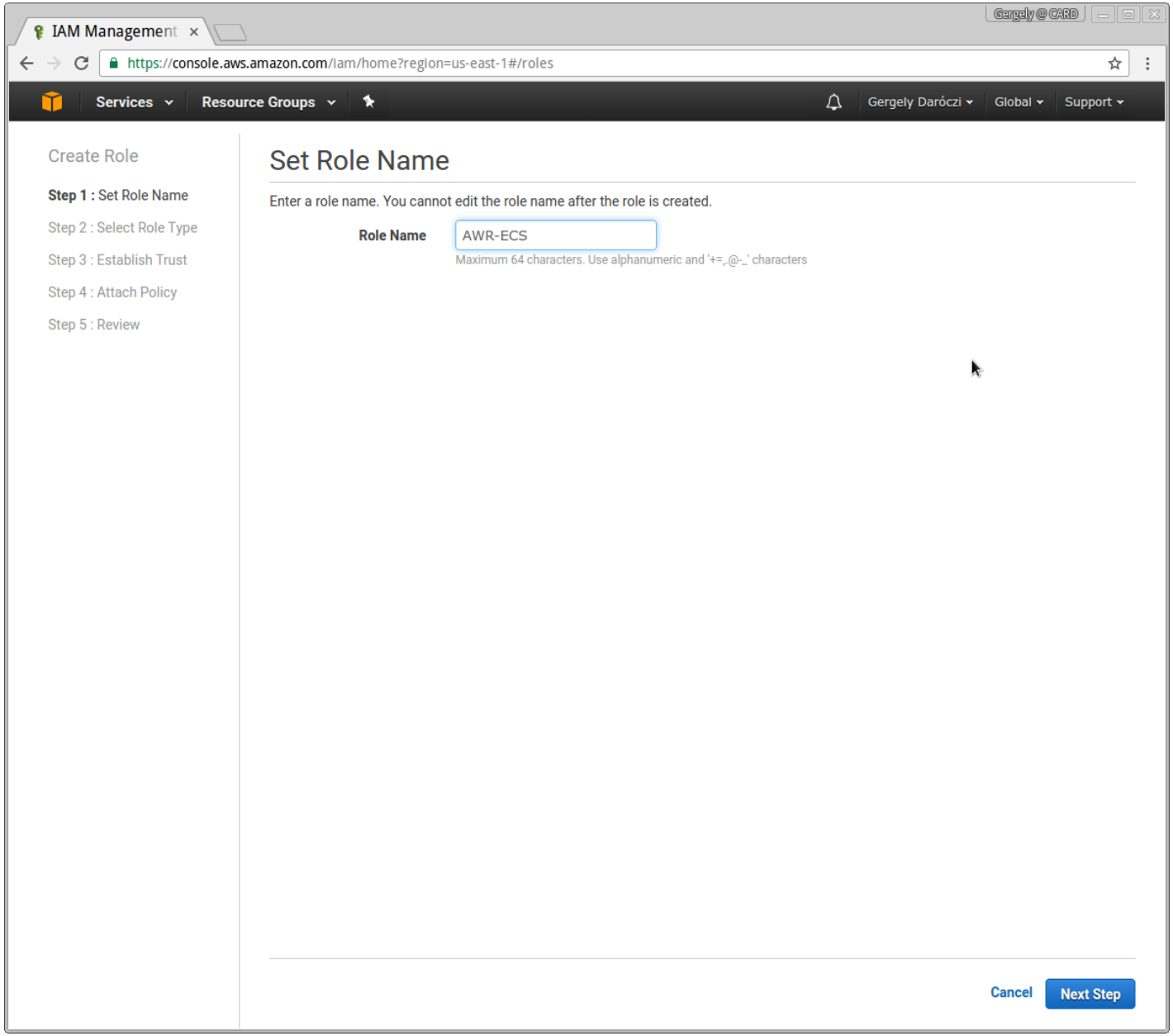
Put it on Docker Hub



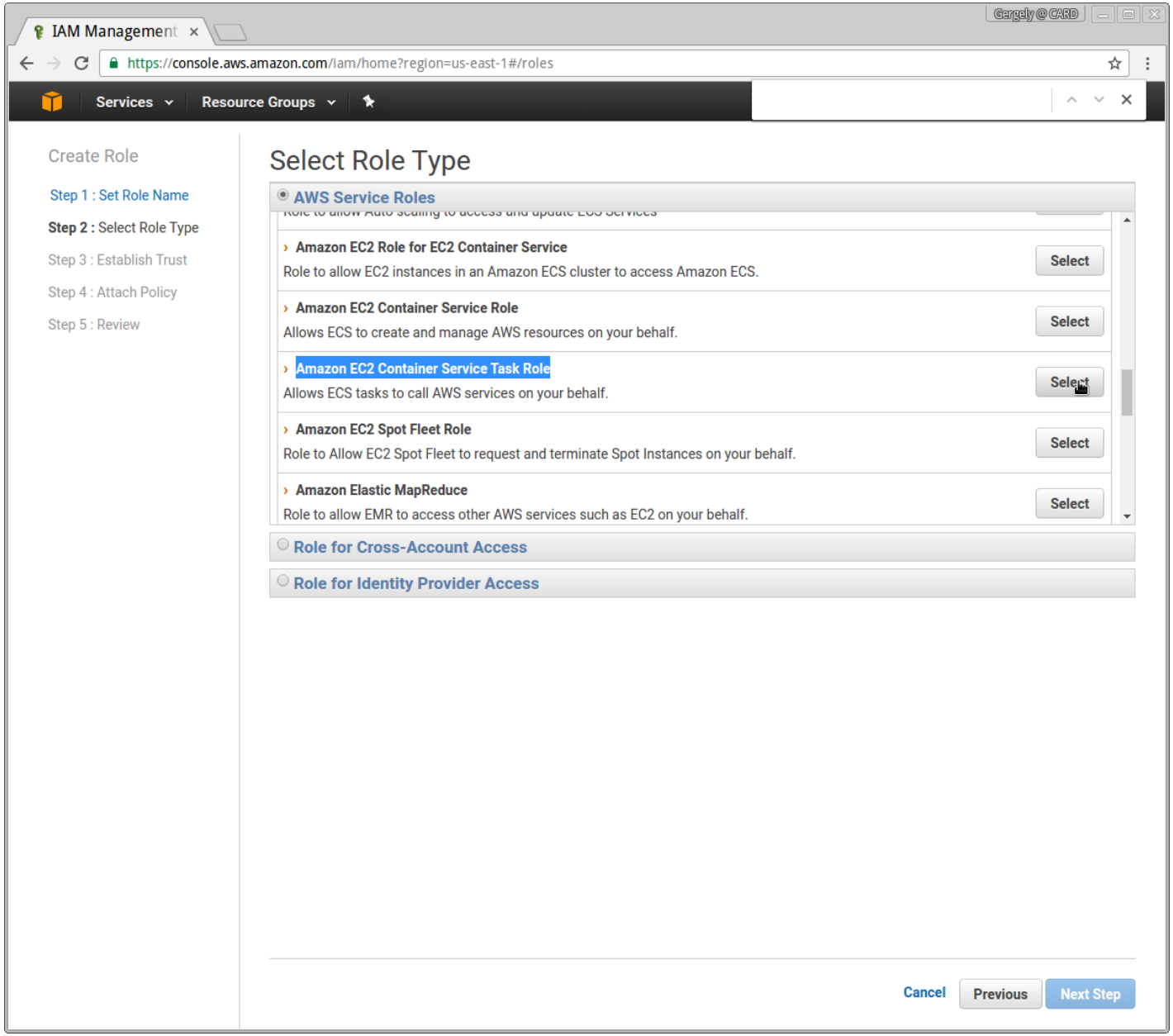
Create an ECS cluster



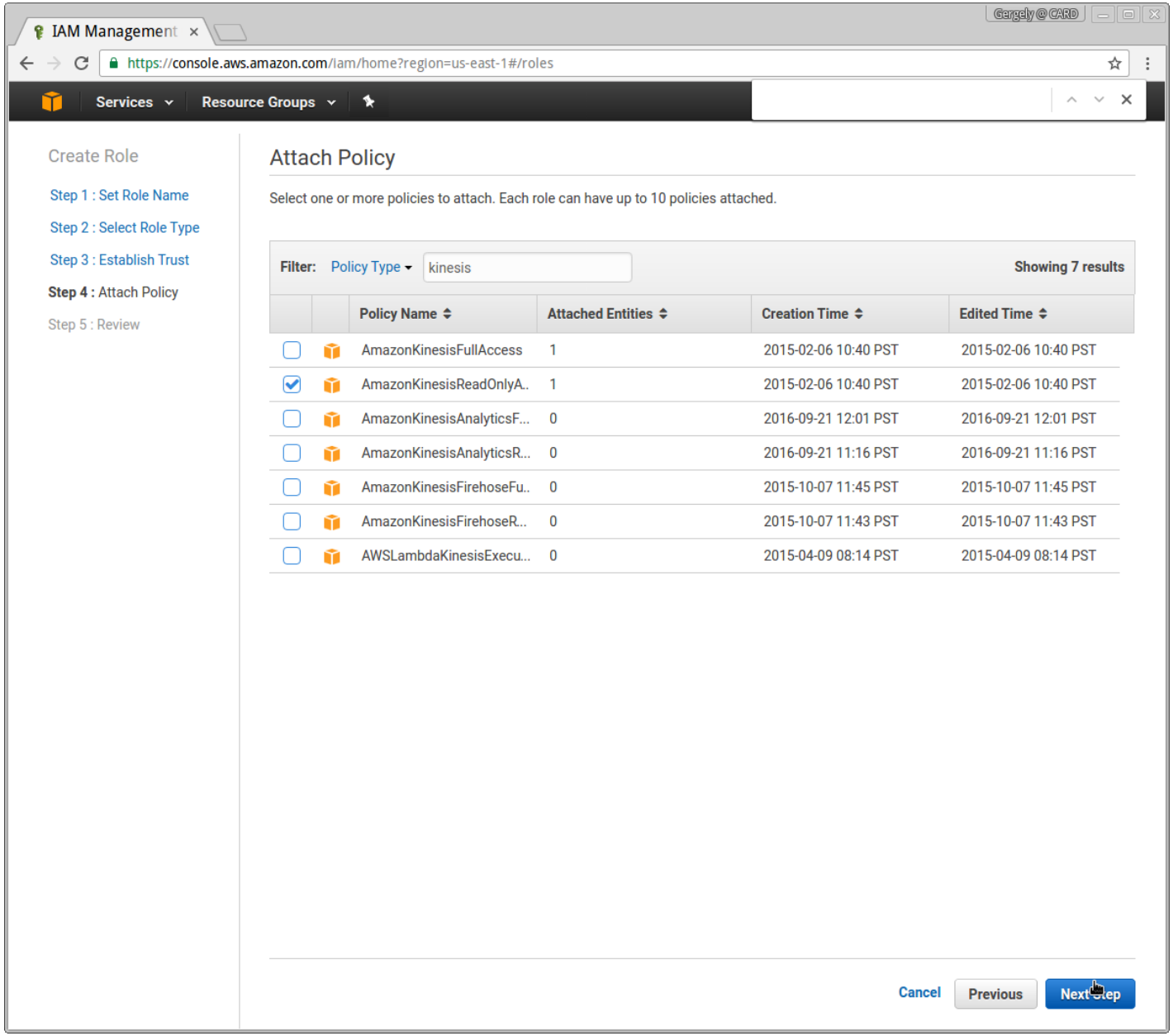
Create ECS Task Role



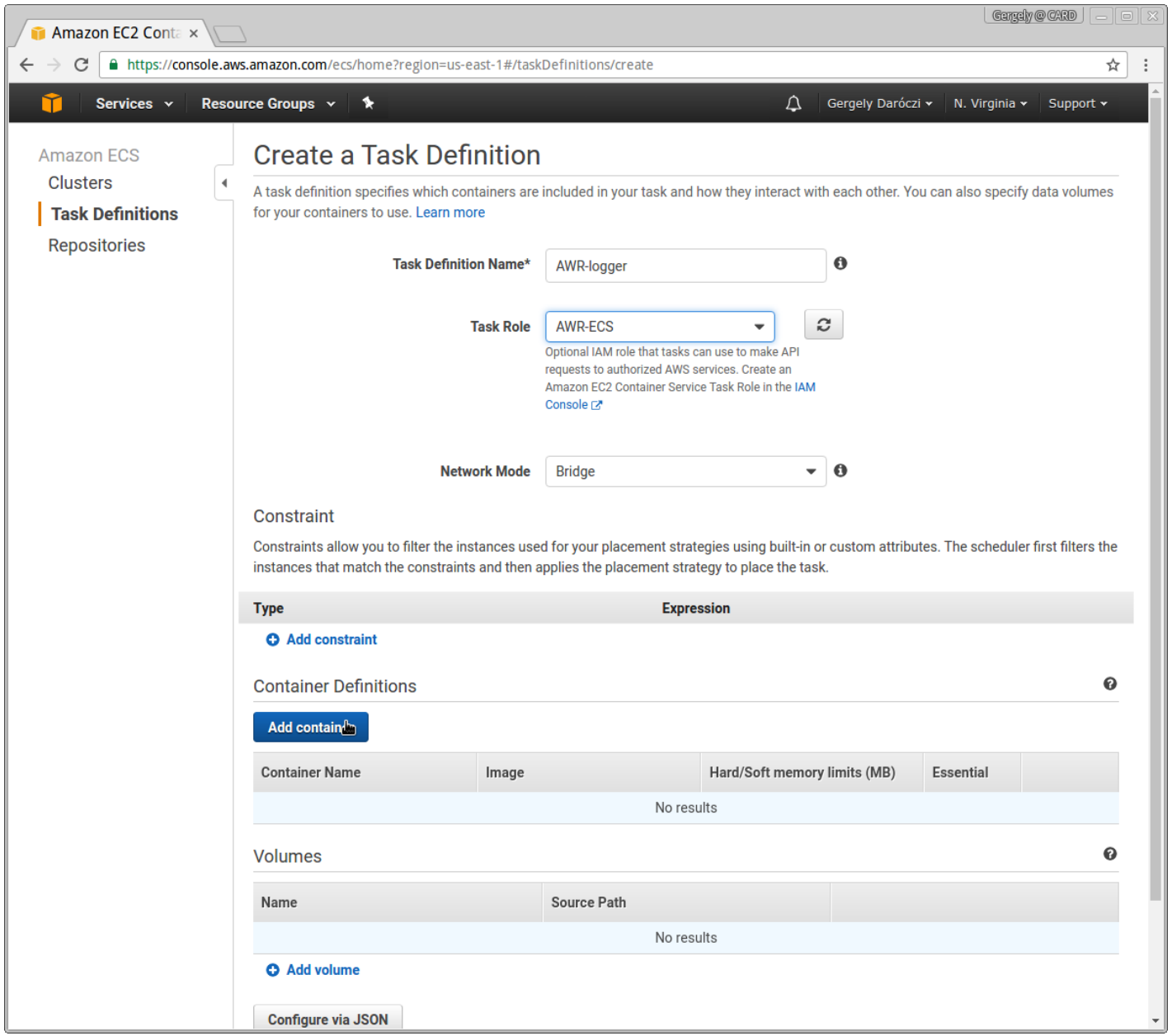
Create ECS Task Role



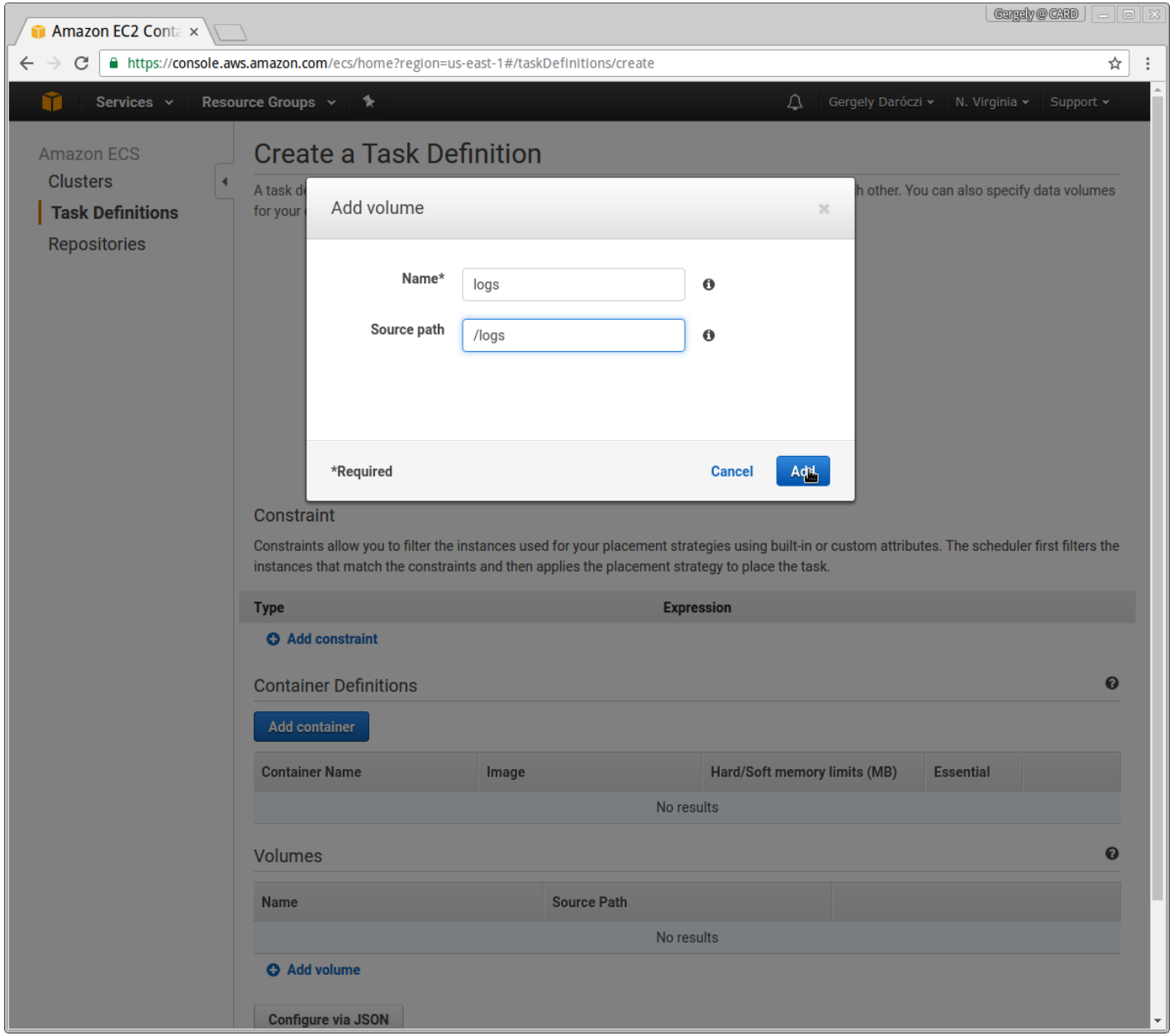
Create ECS Task Role



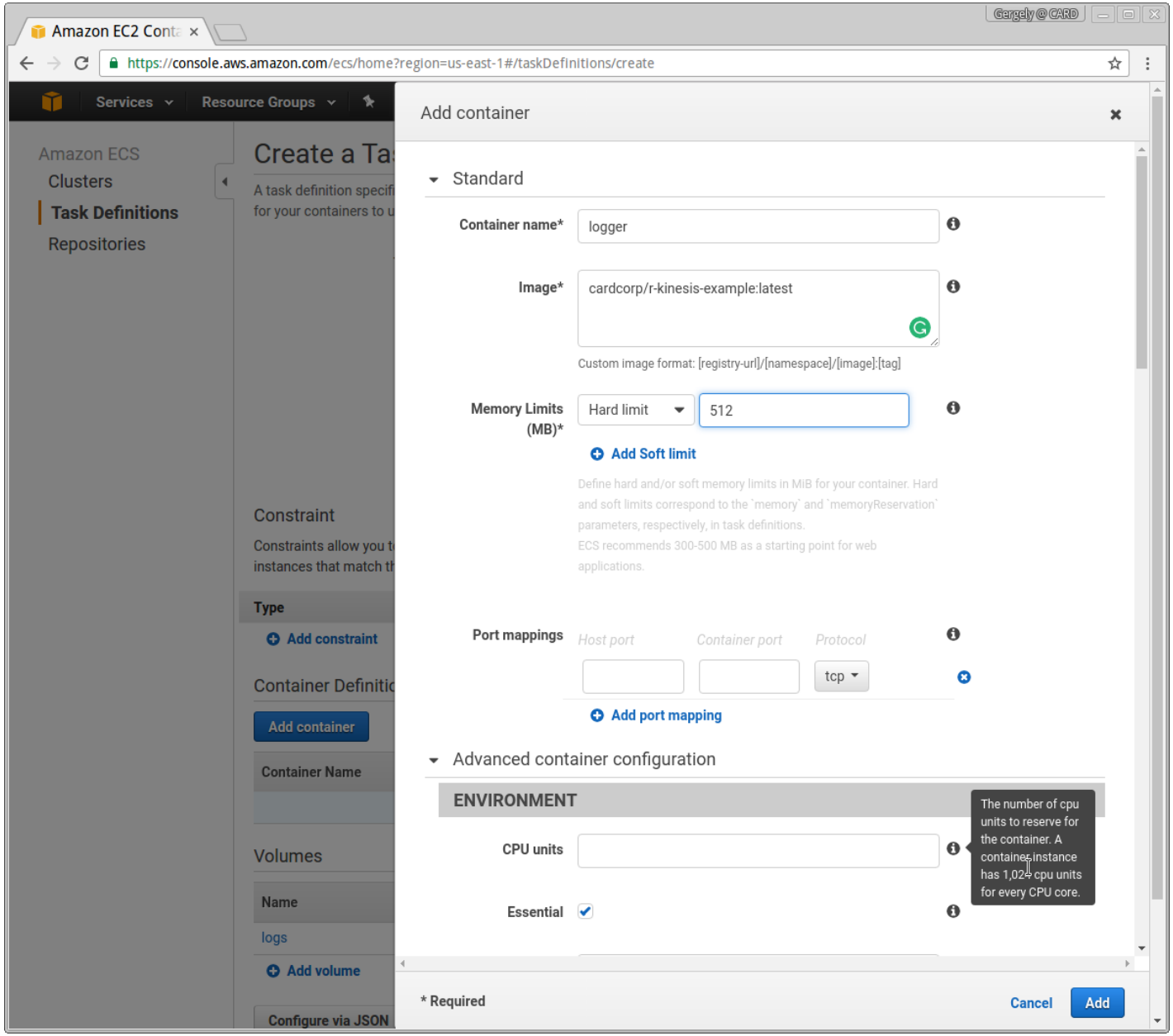
Create a Task definition



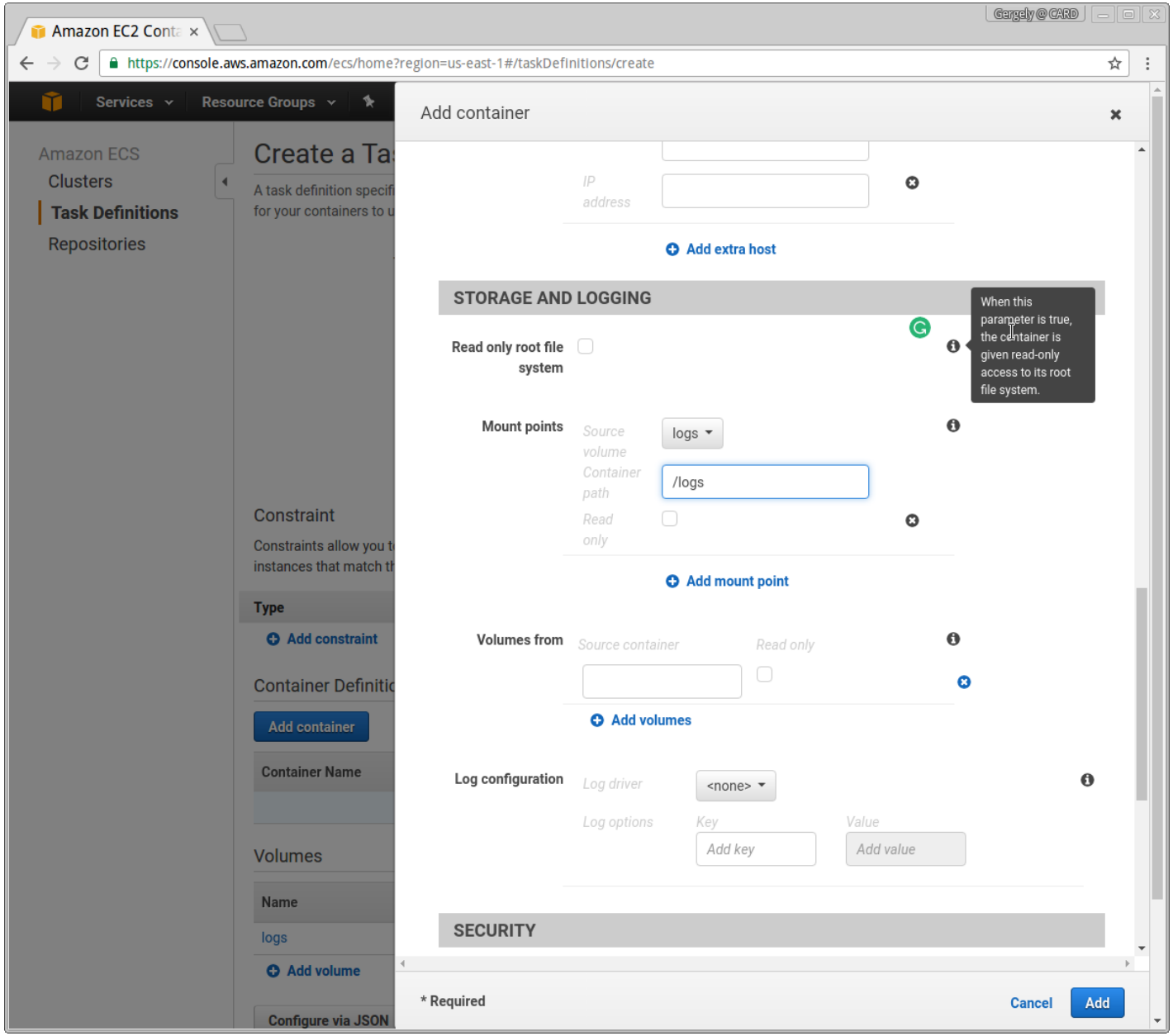
Create a Task definition



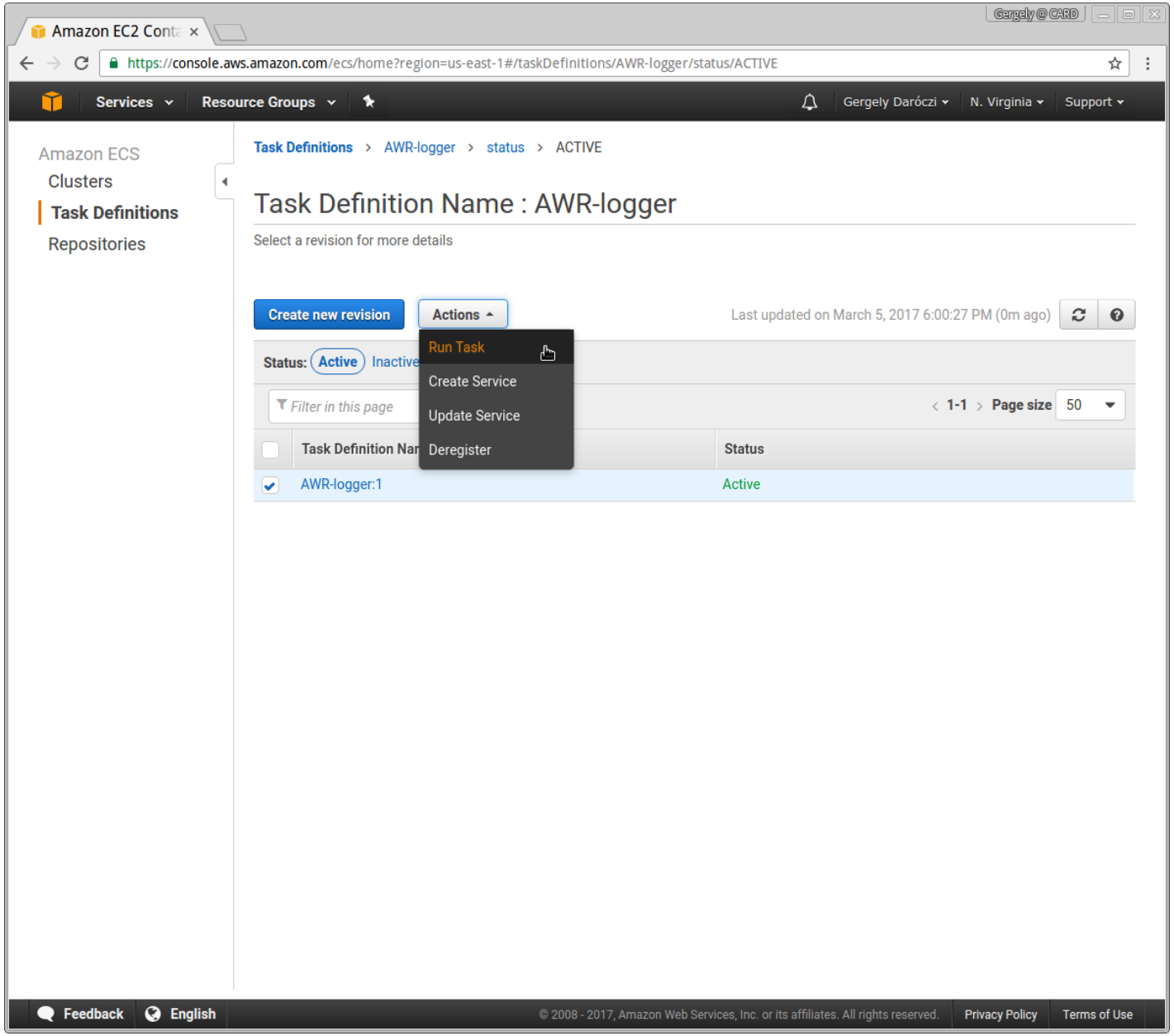
Create a Task definition



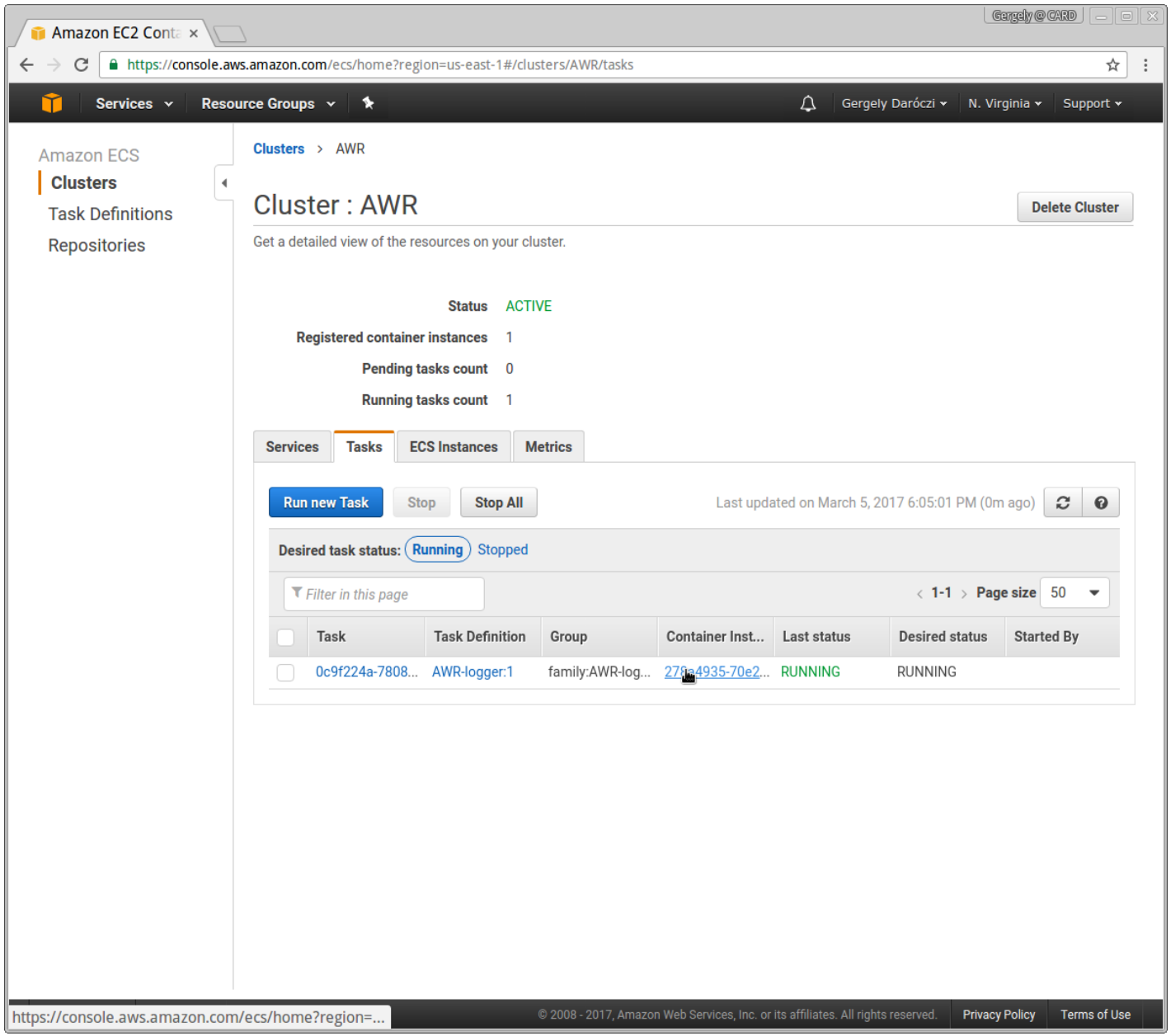
Create a Task definition



Run the ECS Task

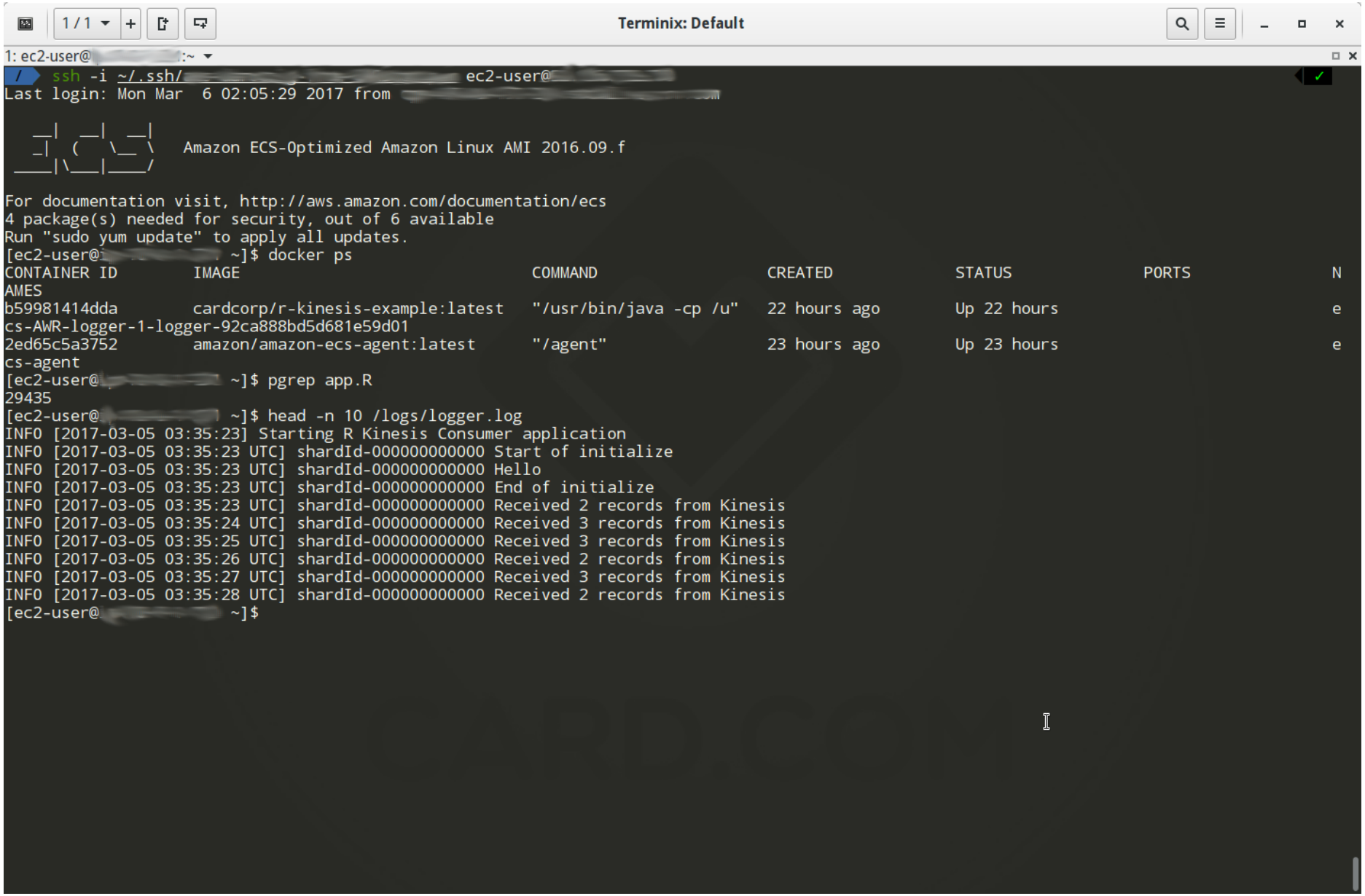


Run the ECS Task

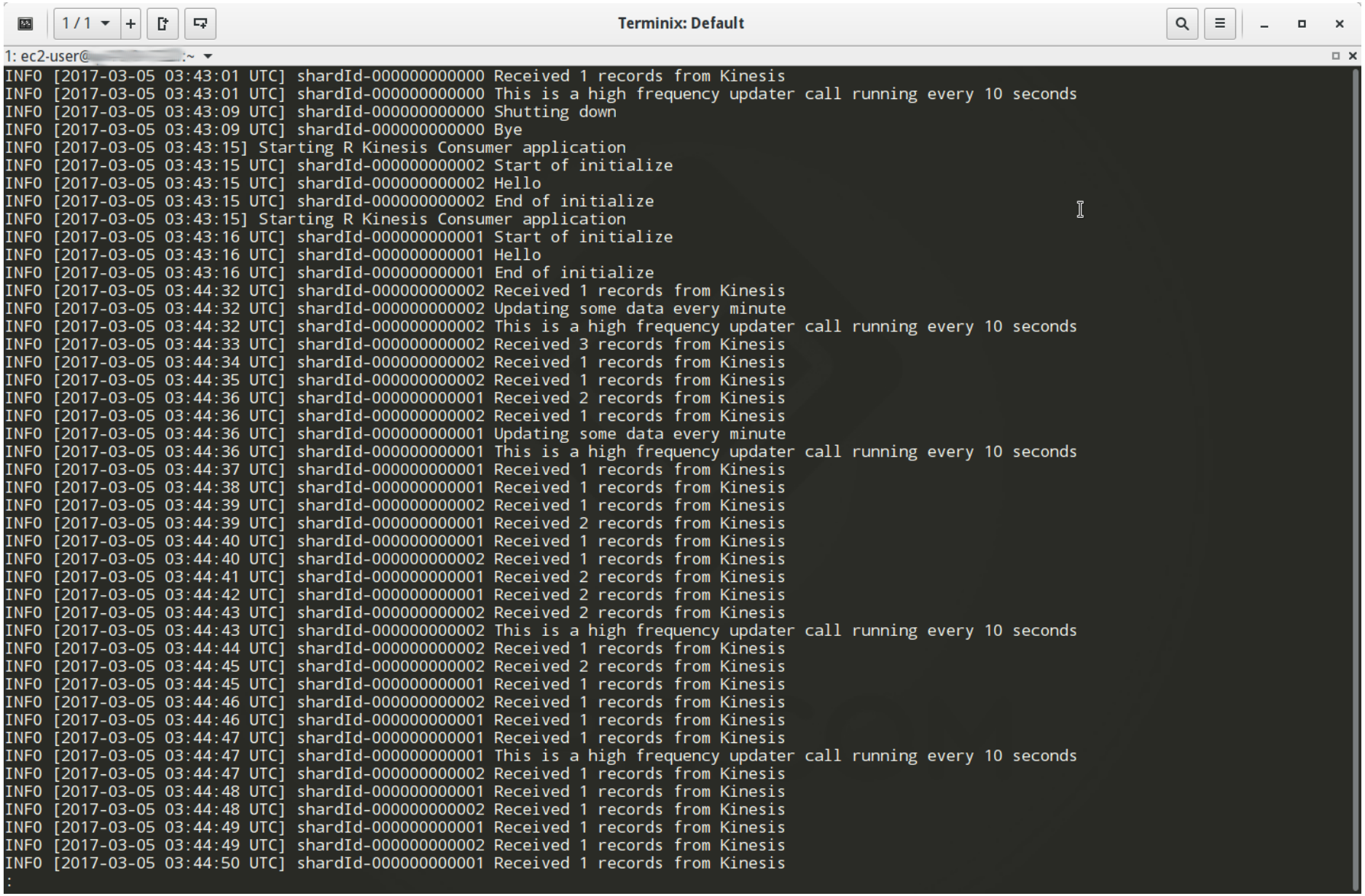


Run the ECS Task

useR!2017



Scaling the Kinesis Consumer up



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 commiting credentials etc to the repo

Problems:

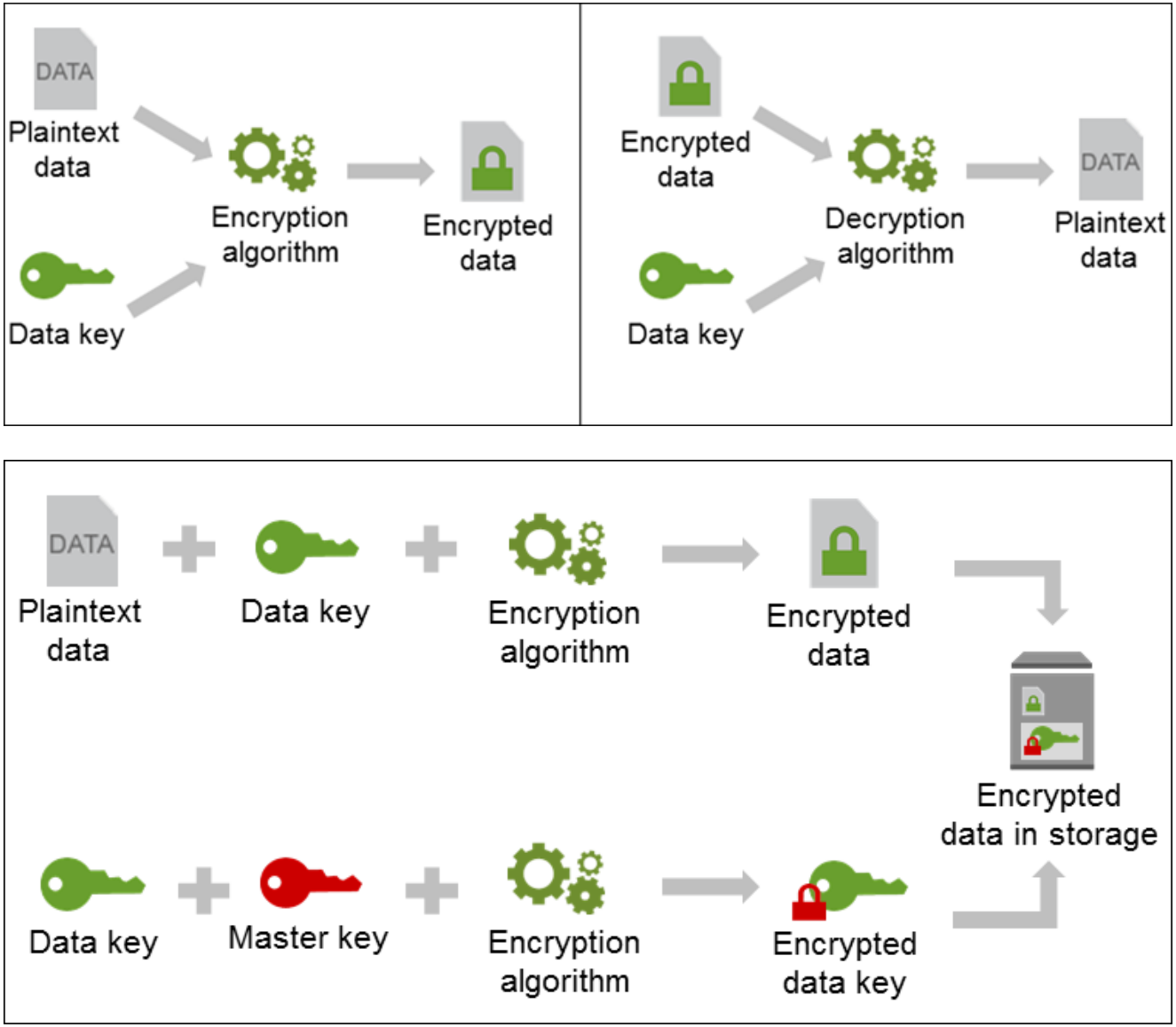
- How to store credentials in the Docker images?
- Where to store the Docker images?

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 commiting 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

Current AWR.KMS Features

- 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_encrypt('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 ...
```

Encrypting Data Larger Than 4 KB?

useR!2017

```
## 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/lDR5euQQScpXQU9iYTn+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')
            }
        }
    },
    updater = 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')
```

Private Docker Image

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
```

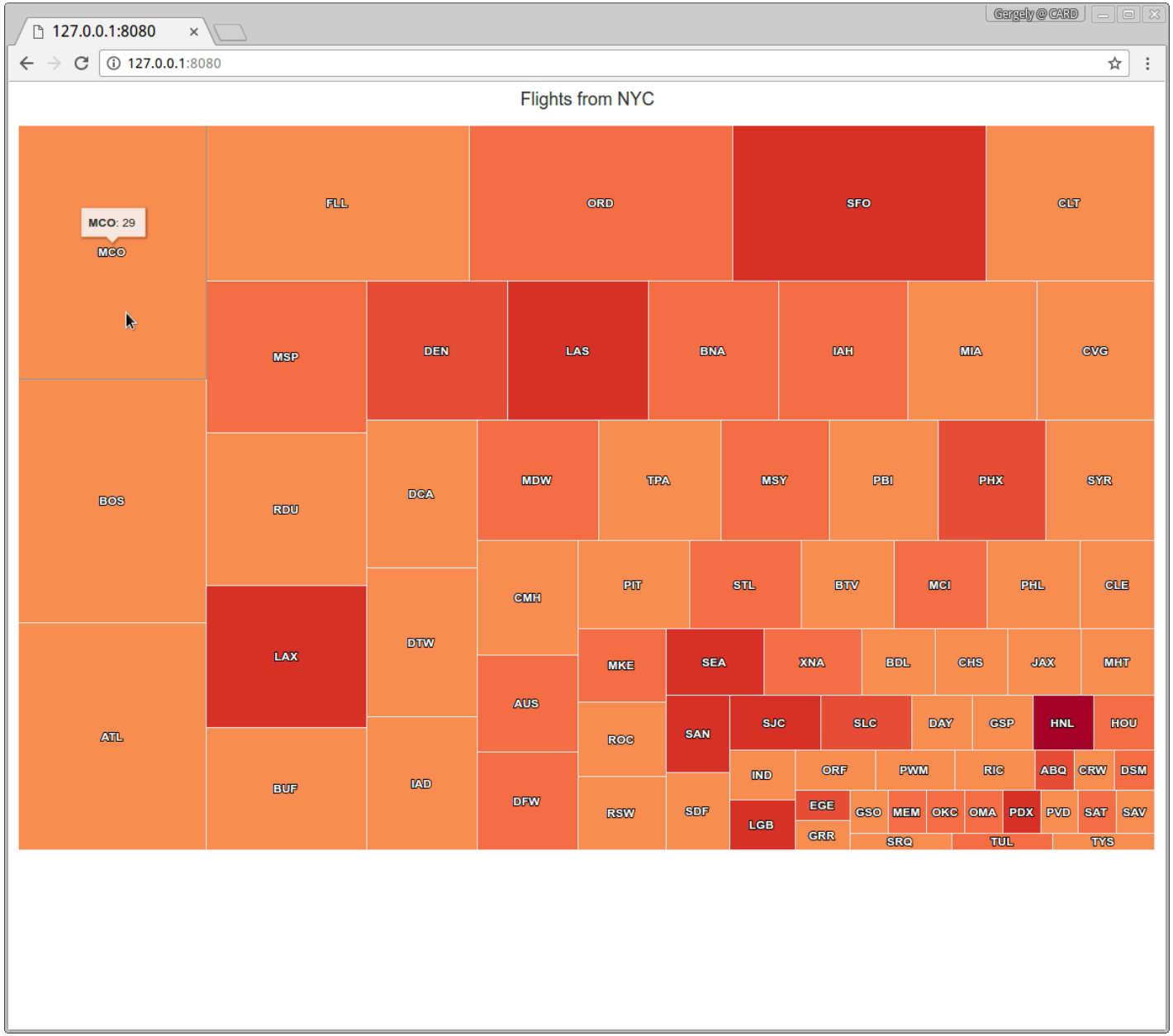
Build and push to ECR:

```
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
```

Shiny Dashboard


```
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)
```

Shiny Dashboard



Technical Details

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

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

Technical Details

- 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

Technical Details

useR!2017

- 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

Technical Details

- 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

What's Next?


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


AWR
Because "S"
is so 1992.