SYSTEM1

Managing Database Credentials and Connections in R:

an Easy and Secure Approach with 'dbr'

Gergely Daroczi

@daroczig

July 12, 2018

About me

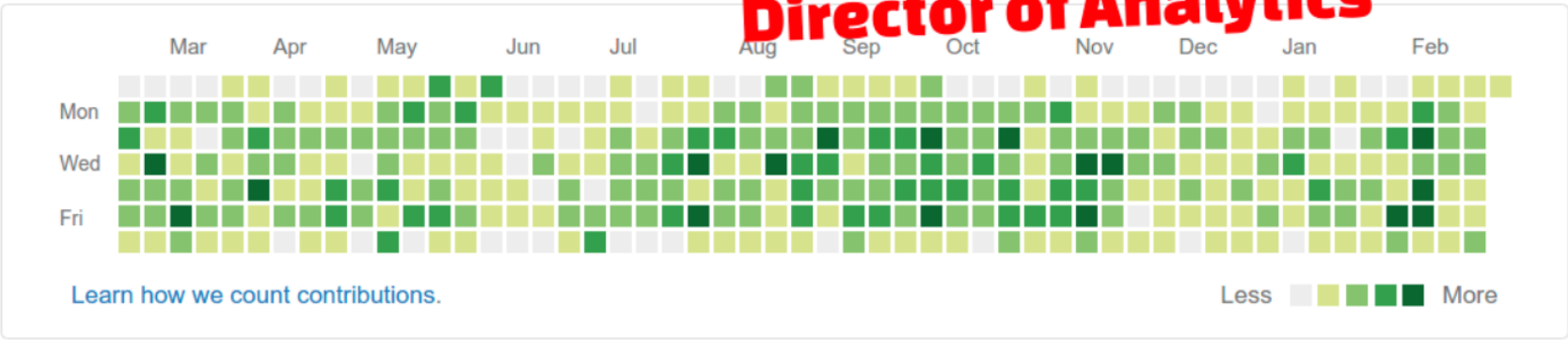
SYSTEM1

3,633 contributions in 2015

Lead R Developer

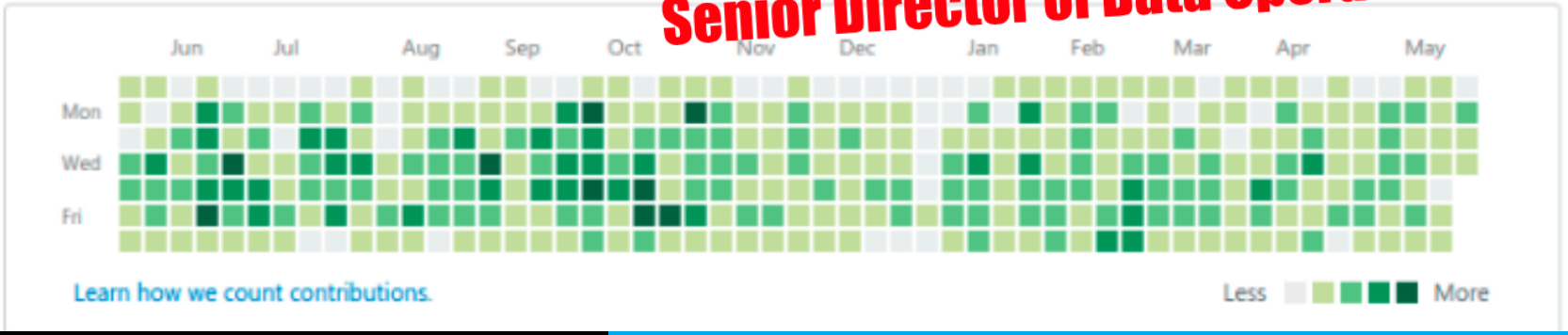
3,470 contributions in 2016

Director of Analytics

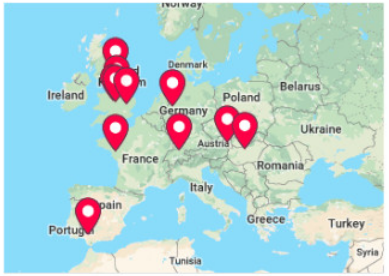


4,591 contributions in 2017

Senior Director of Data Operations



About me



Members	Groups	Countries
20,169	35	17

A brief visual summary on the first #satRdays conf with almost 200 regs from 19 countries and 170 #rstats attendees:

5:52 AM - 6 Sep 2016 from Hungary

Save the date!

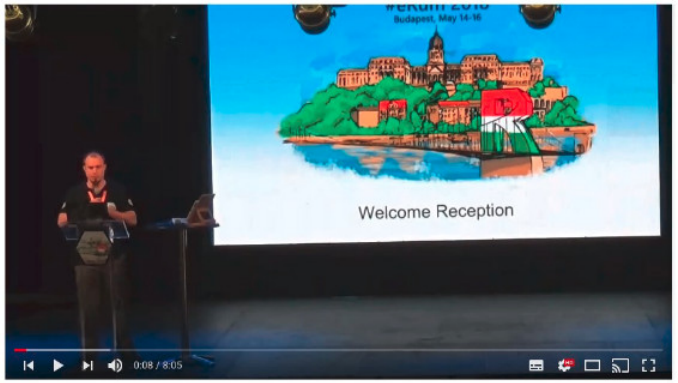
- May 14-16, 2018
- Budapest, Hungary
- ~400-500 #rstats folks
- from €50 (early-bird student reg
- 2018.erum.io

Almost all #erum2018 talk slides and video recordings are now available at 2018.erum.io/#talk-abstracts -- thanks a lot for all our #rstats speakers and the great crowd and lovely R community we had in Budapest in mid-May!

I'm extremely proud of the #rstats speaker lineup for #erum2018 -- looking forward to having you all here in a month!

erum2018 @erum2018
The final program of #erum2018 is now online at 2018.erum.io/#schedule

The recent updates include all the #rstats keynote, invited,...



eRum 2018 - May 14 - Welcome

About me

SYSTEM1

Gergely Daróczi @daroczig · Jul 2

Packing for the "bring your own lanyard" @useR2018_conf, and found quite a good number of lanyards (~10), t-shirts (~30), canvas bags (~10) & hex stickers (~100) left over from #erum2018 -- looking forward to distributing these #rstats swags at #user2018. Ping me if interested 🤓

SYSTEM1

Example MySQL query from R (with 3 potential problems)

```
## connect to the database
library(DBI)
con <- dbConnect(
    RMySQL::MySQL(),
    dbname = "shinydemo",
    host = "shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com",
    username = "guest",
    password = "guest")

## run a query
query <- dbSendQuery(
    con,
    "SELECT Continent, COUNT(DISTINCT(Region)) FROM Country GROUP BY Continent")
res <- dbFetch(query)
dbClearResult(query)

## say good bye
dbDisconnect(con)
```

SYSTEM1

Example MySQL query from R doing AI (with 3 potential problems)

```
## connect to the database
library(DBI)
con <- dbConnect(
    RMySQL::MySQL(),
    dbname = "shinydemo",
    host = "shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com",
    username = "guest",
    password = "guest")

## run a query
query <- dbSendQuery(
    con,
    "SELECT CASE WHEN Continent = 'foo' THEN 'bar' ELSE 'foo' END FROM Country GROU
res <- dbFetch(query)
dbClearResult(query)

## say good bye
dbDisconnect(con)
```

Motivation

SYSTEM1

'When I woke up the next morning, I had four emails and a missed phone call from Amazon AWS – something about 140 servers running on my AWS account, mining Bitcoin.' Andrew Hoffman

Source: Dev put AWS keys on Github. Then BAD THINGS happened

Motivation

SYSTEM1

'When I woke up the next morning, I had four emails and a missed phone call from Amazon AWS – something about 140 servers running on my AWS account, mining Bitcoin.' Andrew Hoffman

Source: Dev put AWS keys on Github. Then BAD THINGS happened

```
SELECT
  TO_HEX(CODE_POINTS_TO_BYTES([0xac, num2, num3, num4])) AS nonce
FROM
  UNNEST(GENERATE_ARRAY(0, 255)) num2,
  UNNEST(GENERATE_ARRAY(0, 255)) num3,
  UNNEST(GENERATE_ARRAY(0, 255)) num4
WHERE
TO_HEX(REVERSE(SHA256(SHA256(CONCAT(FROM_HEX(
  '000000204a4ef98461ee26898076e6a2cfc7c764d02b5f8d6708320000000000000000000f99f5c4d
```

Source: How to mine Bitcoin on Google's BigQuery

SYSTEM1

Using a pre-configured Data Source Name

```
con <- DBI::dbConnect(odbc::odbc(), dsn = "shinydemo")
```

SYSTEM1

Using a pre-configured Data Source Name

```
con <- DBI::dbConnect(odbc::odbc(), dsn = "shinydemo")
```

But we still need someone to set up / deploy configuration.

SYSTEM1

Loading MySQL configuration from the keyring

```
con <- DBI::dbConnect(
    conf$drv,
    dbname = conf$dbname,
    host = conf$host,
    username = keyring::key_get("my_username"),
    password = keyring::key_get("my_password"))
```

SYSTEM1

Loading MySQL configuration from the keyring

```
con <- DBI::dbConnect(
    conf$drv,
    dbname = conf$dbname,
    host = conf$host,
    username = keyring::key_get("my_username"),
    password = keyring::key_get("my_password"))
```

Great for the single-desktop R user, but how to make use of it on a remote server?

SYSTEM1

Using a MySQL configuration file

```
con <- dbConnect(RMySQL::MySQL(), group = "shinydemo")
```

SYSTEM1

Using a MySQL configuration file

```
con <- dbConnect(RMySQL::MySQL(), group = "shinydemo")
```

But we still need to set up `~/.my.cnf`:

```
[shinydemo]
user=guest
password=guest
database=shinydemo
host=shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com
```

SYSTEM1

Loading MySQL configuration from more general, custom files

```
secret_api_token <- readRDS('secrets/amp_token_v201610.RData')
```

SYSTEM1

Loading MySQL configuration from more general, custom files

```
secret_api_token <- readRDS('secrets/amp_token_v201610.RData')

mysql_user <- readRDS('secrets/mysql_user.RData')
mysql_pass <- readRDS('secrets/mysql_pass.RData')

library(DBI)
con <- dbConnect(
    RMySQL::MySQL(),
    dbname = "shinydemo",
    host = "shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com",
    username = mysql_user,
    password = mysql_pass)
```

SYSTEM1

Loading MySQL configuration from more general, custom files

```
secret_api_token <- readRDS('secrets/amp_token_v201610.RData')

mysql_user <- readRDS('secrets/mysql_user.RData')
mysql_pass <- readRDS('secrets/mysql_pass.RData')

library(DBI)
con <- dbConnect(
    RMySQL::MySQL(),
    dbname = "shinydemo",
    host = "shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com",
    username = mysql_user,
    password = mysql_pass)
```

Again, how to get those *unencrypted* `RData` files to the server?

SYSTEM1

Loading MySQL configuration from encrypted custom files

```
library(secret); try(local_key(), silent = TRUE)

vault <- file.path(tempdir(), ".vault"); dir.create(vault); create_vault(vault)

key_dir <- file.path(system.file(package = "secret"), "user_keys")
alice_public_key <- file.path(key_dir, "alice.pub")
alice_private_key <- file.path(key_dir, "alice.pem")
add_user("alice", alice_public_key, vault = vault)

secret_to_keep <- c(password = "guest", username = "guest")
add_secret("secret_one", secret_to_keep, users = "alice", vault = vault)

secrets <- get_secret("secret_one", key = alice_private_key, vault = vault)
mysql_user <- secrets$my_username; mysql_pass <- secrets$my_password

con <- DBI::dbConnect(
    RMySQL::MySQL(), dbname = "shinydemo",
    host = "shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com",
    username = mysql_user, password = mysql_pass)
```

SYSTEM1

Loading MySQL configuration from encrypted custom files

```
library(secret); try(local_key(), silent = TRUE)

vault <- file.path(tempdir(), ".vault"); dir.create(vault); create_vault(vault)

key_dir <- file.path(system.file(package = "secret"), "user_keys")
alice_public_key <- file.path(key_dir, "alice.pub")
alice_private_key <- file.path(key_dir, "alice.pem")
add_user("alice", alice_public_key, vault = vault)

secret_to_keep <- c(password = "guest", username = "guest")
add_secret("secret_one", secret_to_keep, users = "alice", vault = vault)

secrets <- get_secret("secret_one", key = alice_private_key, vault = vault)
mysql_user <- secrets$my_username; mysql_pass <- secrets$my_password

con <- DBI::dbConnect(
    RMySQL::MySQL(), dbname = "shinydemo",
    host = "shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com",
    username = mysql_user, password = mysql_pass)
```

But how to get the private key to a new server?

SYSTEM1

Loading MySQL configuration from global options

```
library(DBI)
con <- dbConnect(
    RMySQL::MySQL(),
    dbname = "shinydemo",
    host = "shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com",
    username = getOption("my_username"),
    password = getOption("my_password"))
```

SYSTEM1

Loading MySQL configuration from global options

```
library(DBI)
con <- dbConnect(
    RMySQL::MySQL(),
    dbname = "shinydemo",
    host = "shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com",
    username = getOption("my_username"),
    password = getOption("my_password"))
```

Again, how to set those env vars on the server?

```
$ cat .Rprofile
options('my_username' = 'guest')
options('my_password' = 'guest')
```

SYSTEM1

Loading MySQL configuration from environment variables

```
library(DBI)
con <- dbConnect(
    RMySQL::MySQL(),
    dbname = "shinydemo",
    host = "shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com",
    username = Sys.getenv("my_username"),
    password = Sys.getenv("my_password"))
```

SYSTEM1

Loading MySQL configuration from environment variables

```
library(DBI)
con <- dbConnect(
    RMySQL::MySQL(),
    dbname = "shinydemo",
    host = "shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com",
    username = Sys.getenv("my_username"),
    password = Sys.getenv("my_password"))
```

Again, how to set those global options on the server?

```
$ cat .Renviron
my_username = "guest"
my_password = "guest"

$ docker run --env my_username=guest --env my_password=guest my_docker_image ...

$ aws ecs register-task-definition --container-definitions ...
```

SYSTEM1

Loading MySQL configuration from config files

```
conf <- config::get("shinydemo")
con  <- DBI::dbConnect(
    RMySQL::MySQL(),
    dbname = conf$dbname,
    host = conf$host,
    username = conf$username,
    password = conf$password)
```

With the below YAML config:

```
default:
  shinydemo:
    host: shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com
    username: guest
    password: guest
    dbname: shinydemo
```

SYSTEM1

Loading MySQL configuration from config files

```
conf <- config::get("shinydemo")
con  <- DBI::dbConnect(
    conf$drv,
    dbname = conf$dbname,
    host = conf$host,
    username = conf$username,
    password = conf$password)
```

With the below YAML config:

```
default:
  shinydemo:
    drv: !expr RMySQL::MySQL()
    host: shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com
    username: guest
    password: guest
    dbname: shinydemo
```

SYSTEM1

Loading MySQL configuration from config files

```
conf <- config::get("shinydemo")
do.call(DBI::dbConnect, conf)
```

With the below YAML config:

```
default:
  shinydemo:
    drv: !expr RMySQL::MySQL()
    host: shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com
    username: guest
    password: guest
    dbname: shinydemo
```

SYSTEM1

Loading MySQL configuration from config files

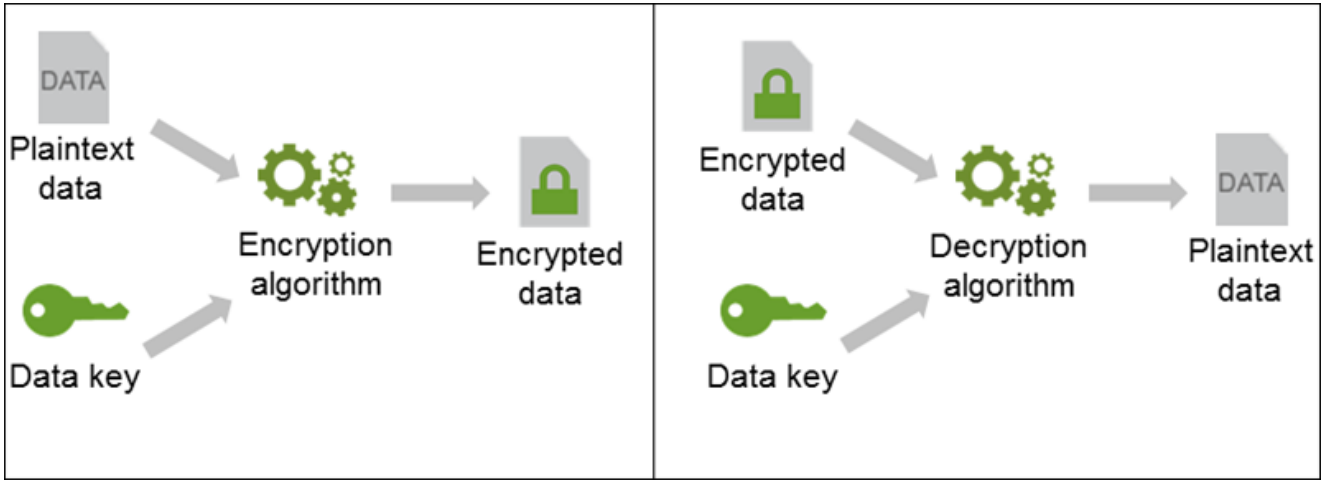
```
conf <- config::get("shinydemo")
do.call(DBI::dbConnect, conf)
```

With the below YAML config:

```
default:
  shinydemo:
    drv: !expr RMySQL::MySQL()
    host: shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com
    username: guest
    password: guest
    dbname: shinydemo
```

But again, we have to get the YAML file to the server :/

Amazon KMS



Source: AWS Encryption SDK

How to deploy the private key to the server?

Using the Default Credential Provider Chain

SYSTEM1

How to deploy the private key to the server?

You cannot.

Using the Default Credential Provider Chain

SYSTEM1

How to deploy the private key to the server?

You cannot.

How to grant access to KMS keys?

- Environment variables: via `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`
- The default credential profiles file: via `~/.aws/credentials`
- Amazon ECS container credentials: via `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI`
- Instance profile credentials: via http://169.254.169.254/latest/meta-data/

Amazon KMS from R: AWR.KMS

SYSTEM1

Current AWR.KMS Features

SYSTEM1

- 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"
```

Current AWR.KMS Features

SYSTEM1

- encrypt up to 4 KB of arbitrary data:

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

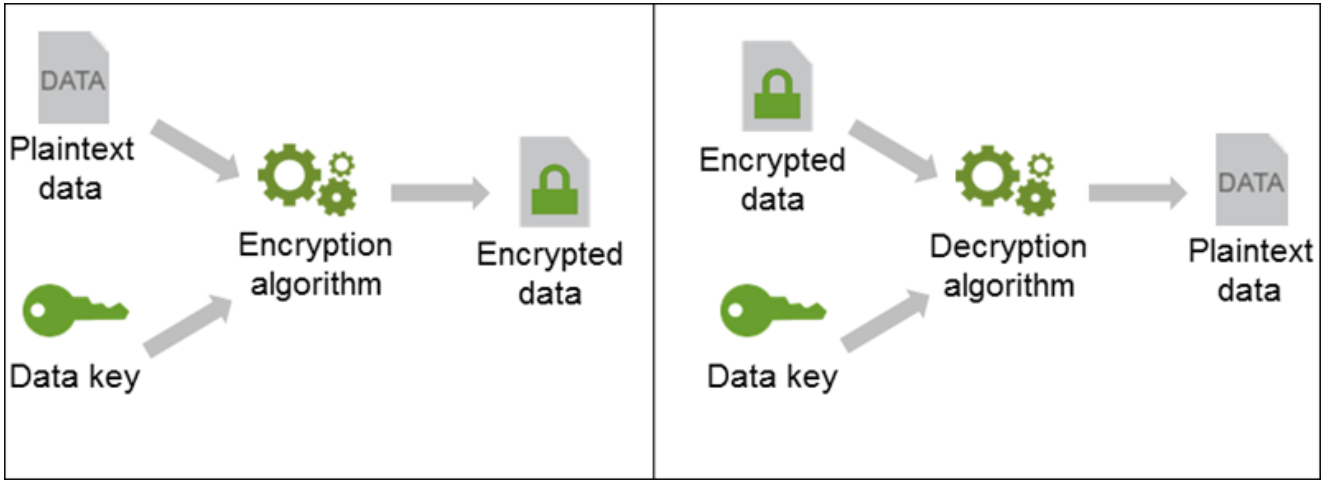
- decrypt such Base-64 encoded ciphertext back to plaintext:

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

- generate a data encryption key:

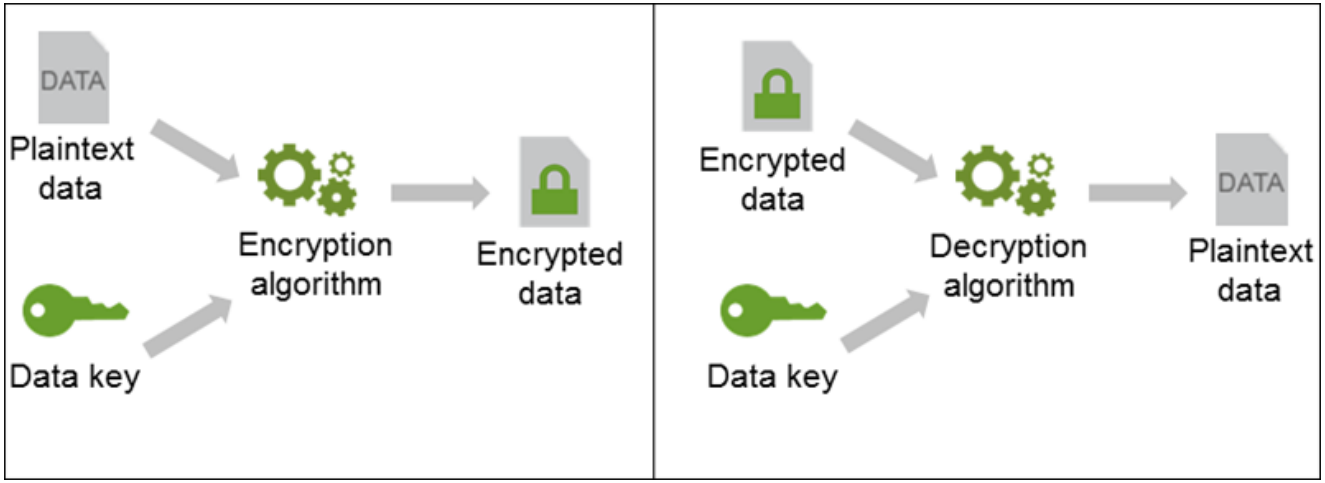
```
> kms_generate_data_key('alias/mykey')
$cipher
[1] "Base-64 encoded, encrypted data encryption key"
$key
[1] "alias/mykey"
$text
[1] 00 01 10 11 00 01 10 11 ...
```

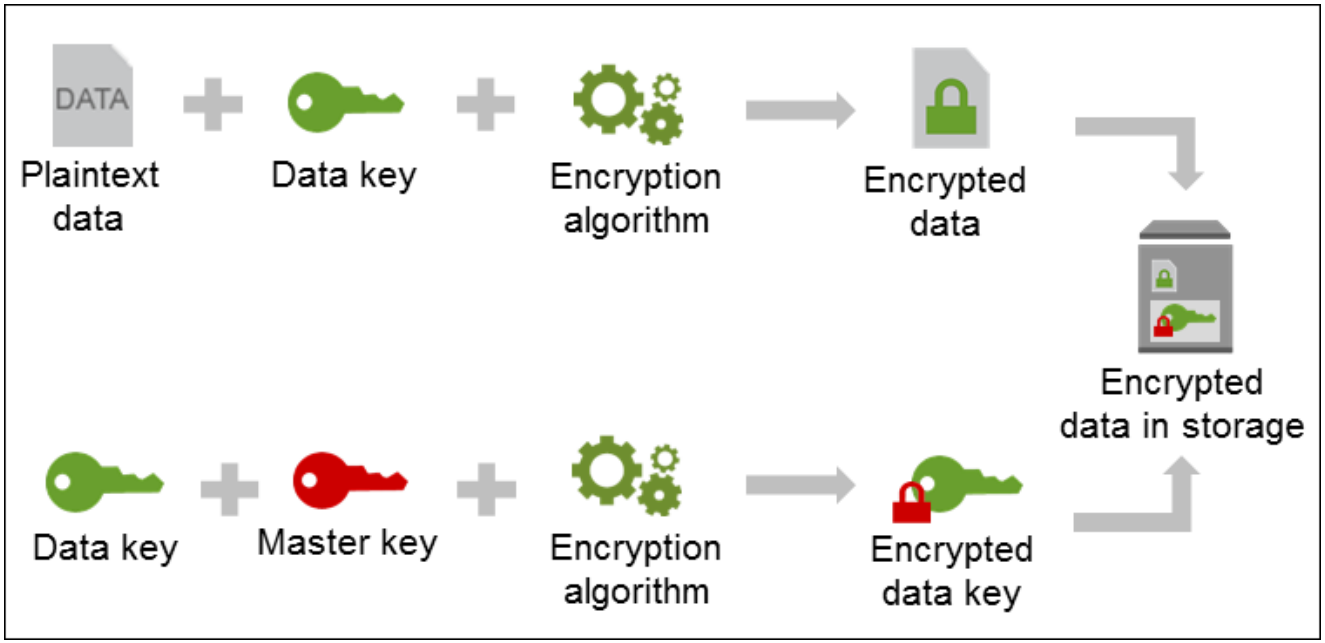
Amazon KMS



Source: AWS Encryption SDK

Amazon KMS





Source: AWS Encryption SDK

Encrypting Data Larger Than 4 KB?

SYSTEM1

```r
## 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 128 bits (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))
```

Encrypting Data Larger Than 4 KB?

SYSTEM1

```
AWR.KMS::kms_encrypt_file
AWR.KMS::kms_decrypt_file
```

Example config file using KMS

```
> AWR.KMS::kms_encrypt('guest', key = 'alias/gergely-test')
[1] "AQICAHiMkU2ZNbL+kRcQoM3wGpuLb8HbIKjM9VcEGt72rZV2SAGxXb00/k152quzRzCV+n6r\nAAAA
```

Example config file using KMS

SYSTEM1

```
> AWR.KMS::kms_encrypt('guest', key = 'alias/gergely-test')
[1] "AQICAHiMkU2ZNbL+kRcQoM3wGpuLb8HbIKjM9VcEGt72rZV2SAGxXb00/k152quzRzCV+n6r\nAAAA
```

```
> db_config_encrypt_secret('guest', key = 'alias/gergely-test')
AQICAHiMkU2ZNbL+kRcQoM3wGpuLb8HbIKjM9VcEGt72rZV2SAE6IQVMFPyj9JBP7cEgf9oT
AAAAYzBhBgkqhkiG9w0BBwagVDBSAgEAME0GCSqGSIb3DQEHATAeBglghkgBZQMEAS4wEQQM
Q8zMzSSMTX0UzT0dAgEQgCBlwaYQyO29zKbtIBuQtSHBWxqgyu49/lUQKZn8CCwmyQ==

> db_config_encrypt_secret(
+   secret = 'shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com',
+   key = 'alias/gergely-test')
AQICAHiMkU2ZNbL+kRcQoM3wGpuLb8HbIKjM9VcEGt72rZV2SAEXX7aTXvtsf91BzgoiiIDh
AAAAlDCBkQYJKoZIhvcNAQcGoIGDMIGAAgEAMHsGCSqGSIb3DQEHATAeBglghkgBZQMEAS4w
EQQMgVoMPjgAi+S7i7cvAgEQgE5X4dnyt/Tl0+PiX/yjzdC2wYl+tWzvHnApAhIahQroK+VJ
8OQEQse/s/VE6n2gHPuXe4c/9lK9Od6e1aR8+YZCflyOA5F2sWFz6+hU5XI=
```

Example config file using KMS

SYSTEM1

```
shinydemo:
  drv: !expr RMySQL::MySQL()
  host: !kms |
    AQICAHiMkU2ZNbL+kRcQoM3wGpuLb8HbIKjM9VcEGt72rZV2SAEXX7aTXvtsf91BzgoiiIDh
    AAAAlDCBkQYJKoZIhvcNAQcGoIGDMIGAAgEAMHsGCSqGSIb3DQEHATAeBglghkgBZQMEAS4w
    EQQMgVoMPjgAi+S7i7cvAgEQgE5X4dnyt/Tl0+PiX/yjzdC2wYl+tWzvHnApAhIahQroK+VJ
    8OQEQse/s/VE6n2gHPuXe4c/9lK9Od6e1aR8+YZCflyOA5F2sWFz6+hU5XI=
  username: !kms |
    AQICAHiMkU2ZNbL+kRcQoM3wGpuLb8HbIKjM9VcEGt72rZV2SAE6IQVMFPyj9JBP7cEgf9oT
    AAAAYzBhBgkqhkiG9w0BBwagVDBSAgEAME0GCSqGSIb3DQEHATAeBglghkgBZQMEAS4wEQQM
    Q8zMzSSMTX0UzT0dAgEQgCBlwaYQyO29zKbtIBuQtSHBWxqgyu49/lUQKZn8CCwmyQ==
  password: !kms |
    AQICAHiMkU2ZNbL+kRcQoM3wGpuLb8HbIKjM9VcEGt72rZV2SAE6IQVMFPyj9JBP7cEgf9oT
    AAAAYzBhBgkqhkiG9w0BBwagVDBSAgEAME0GCSqGSIb3DQEHATAeBglghkgBZQMEAS4wEQQM
    Q8zMzSSMTX0UzT0dAgEQgCBlwaYQyO29zKbtIBuQtSHBWxqgyu49/lUQKZn8CCwmyQ==
  dbname: shinydemo
```

Parsing our example config file using KMS

SYSTEM1

```
db_config <- function(db, db_config_path = getOption('db_config_path') {

    if (is.function(db_config_path)) {
        db_config_path <- db_config_path()
    }

    if (!file.exists(db_config_path)) {
        stop(paste('DB config file not found at', db_config_path))
    }

    ## parse config file
    db_secrets <- yaml.load_file(
        db_config_path,
        ## add KMS classes
        handlers = list('kms' = function(x) structure(x, class = c('kms'))))

    hasName(db_secrets, db) || stop('Database ', db, ' not found, check ', db_config

    flog.debug('Looking up config for {db}')

    ## hit KMS with each base64-encoded cipher-text (if any) and decrypt
    rapply(db_secrets[[db]], kms_decrypt, classes = 'kms', how = 'replace')

})
```

Example config file using KMS

SYSTEM1

```
> options('db_config_path' = ...)
> db_config('shinydemo')
$drv
<MySQLDriver>

$host
[1] "shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com"

$username
[1] "guest"

$password
[1] "guest"

$dbname
[1] "shinydemo"
```

Motivation (again)

SYSTEM1

Example MySQL query from R

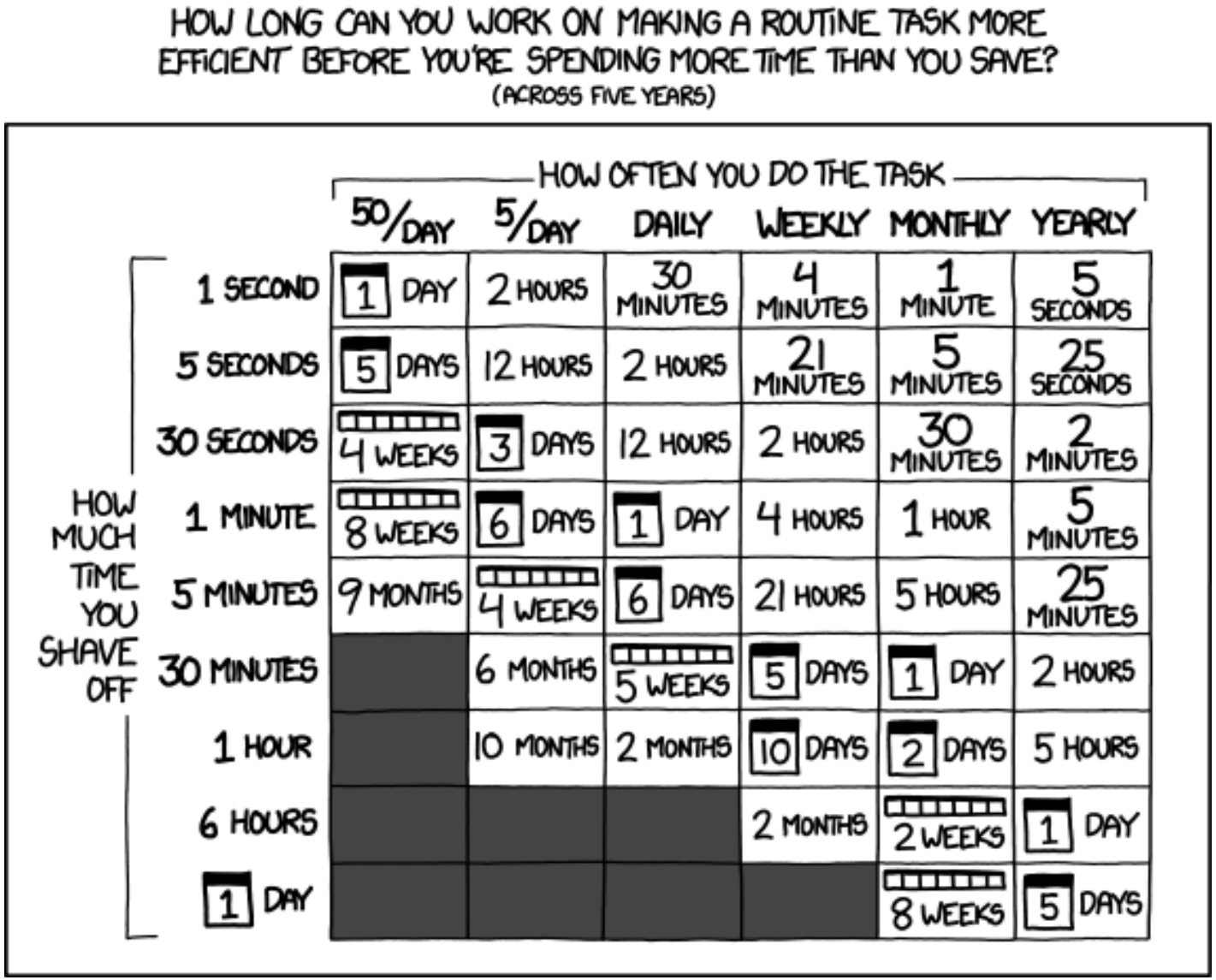
```
## connect to the database
library(DBI)
con <- dbConnect(
    RMySQL::MySQL(),
    dbname = "shinydemo",
    host = "shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com",
    username = "guest",
    password = "guest")

## run a query
query <- dbSendQuery(
    con,
    "SELECT Continent, COUNT(DISTINCT(Region)) FROM Country GROUP BY Continent")
res <- dbFetch(query)
dbClearResult(query)

## say good bye
dbDisconnect(con)
```

SYSTEM1

Example MySQL query from R using YAML config and KMS

```
## connect to the database
library(DBI)
con <- do.call(dbConnect, db_config('shinydemo'))

## run a query
query <- dbSendQuery(
    con,
    "SELECT Continent, COUNT(DISTINCT(Region)) FROM Country GROUP BY Continent")
res <- dbFetch(query)
dbClearResult(query)

## say good bye
dbDisconnect(con)
```

SYSTEM1



Source: xkcd/1205

A simple wrapper to connect AND query

SYSTEM1

```
db_query <- function(sql, db) {

    db <- db_connect(db)

    on.exit({
        db_close(db)
    })

    dbGetQuery(db, sql)

}
```

A simple wrapper to connect AND query

SYSTEM1

```
db_query <- function(sql, db, ...) {

    if (!is.object(db)) {
        db <- db_connect(db, ...)
        on.exit({
          db_close(db)
        })
    }

    assert_attr(db, 'db')
    assert_string(sql)

    flog.info("Executing:**********")
    flog.info(sql)
    flog.info("********************")

    start <- Sys.time()
    result_set <- dbGetQuery(db, sql)
    time_to_exec <- Sys.time() - start

    flog.info("Finished in %s returning %s rows",
              format(time_to_exec, digits = 4),
              nrow(result_set))

    attr(result_set, 'when') <- start
    attr(result_set, 'db') <- attr(db, 'db')
    attr(result_set, 'time_to_exec') <- time_to_exec
    attr(result_set, 'statement') <-  sql

    result_set

}
```

db_query example

SYSTEM1

Bundled `db_config.yml` coming with the `dbr` package:

```
sqlite:
  drv: !expr RSQLite::SQLite()
  dbname: !expr tempfile()
```

db_query example

SYSTEM1

Bundled `db_config.yml` coming with the `dbr` package:

```
sqlite:
  drv: !expr RSQLite::SQLite()
  dbname: !expr tempfile()
```

```
> db_query('SELECT 42', 'sqlite')

INFO [2018-07-12 02:46:25] Connecting to sqlite
INFO [2018-07-12 02:46:25] Executing:**********
INFO [2018-07-12 02:46:25] SELECT 42
INFO [2018-07-12 02:46:25] ********************
INFO [2018-07-12 02:46:25] Finished in 0.001331 secs returning 1 rows
INFO [2018-07-12 02:46:25] Closing connection to sqlite
  42
1 42
```

db_query features: attributes

SYSTEM1

```
> str(db_query('SELECT 42', 'sqlite'))

INFO [2018-07-11 17:07:12] Connecting to sqlite
INFO [2018-07-11 17:07:12] Executing:**********
INFO [2018-07-11 17:07:12] SELECT 42
INFO [2018-07-11 17:07:12] ********************
INFO [2018-07-11 17:07:12] Finished in 0.0007429 secs returning 1 rows
INFO [2018-07-11 17:07:12] Closing connection to sqlite

'data.frame':   1 obs. of  1 variable:
 $ 42: int 42
 - attr(*, "when")= POSIXct, format: "2018-07-11 17:07:12"
 - attr(*, "db")= chr "sqlite"
 - attr(*, "time_to_exec")=Class 'difftime'  atomic [1:1] 0.000743
  .. ..- attr(*, "units")= chr "secs"
 - attr(*, "statement")= chr "SELECT 42"
```

db_query features: making use of attributes

```
> res <- db_query('SELECT CURRENT_TIMESTAMP AS time, 42 AS everything', 'sqlite')
INFO [2018-07-12 02:44:57] Connecting to sqlite
INFO [2018-07-12 02:44:57] Executing:**********
INFO [2018-07-12 02:44:57] SELECT CURRENT_TIMESTAMP AS time, 42 AS everything
INFO [2018-07-12 02:44:57] ******************
INFO [2018-07-12 02:44:57] Finished in 0.0007801 secs returning 1 rows
INFO [2018-07-12 02:44:57] Closing connection to sqlite

> res
                 time everything
1 2018-07-12 00:44:57         42
```

db_query features: making use of attributes

```
> res <- db_query('SELECT CURRENT_TIMESTAMP AS time, 42 AS everything', 'sqlite')
INFO [2018-07-12 02:44:57] Connecting to sqlite
INFO [2018-07-12 02:44:57] Executing:**********
INFO [2018-07-12 02:44:57] SELECT CURRENT_TIMESTAMP AS time, 42 AS everything
INFO [2018-07-12 02:44:57] ********************
INFO [2018-07-12 02:44:57] Finished in 0.0007801 secs returning 1 rows
INFO [2018-07-12 02:44:57] Closing connection to sqlite

> res
                 time everything
1 2018-07-12 00:44:57         42
```

```
> db_refresh(res)
INFO [2018-07-12 02:48:20] Connecting to sqlite
INFO [2018-07-12 02:48:20] Executing:**********
INFO [2018-07-12 02:48:20] SELECT CURRENT_TIMESTAMP AS time, 42 AS everything
INFO [2018-07-12 02:48:20] ********************
INFO [2018-07-12 02:48:20] Finished in 0.0005436 secs returning 1 rows
INFO [2018-07-12 02:48:20] Closing connection to sqlite

time everything
1 2018-07-12 00:48:20         42
```

db_query features: connections

SYSTEM1

```
> db_query('SELECT 42', 'sqlite')
INFO [2018-07-12 02:46:25] Connecting to sqlite
INFO [2018-07-12 02:46:25] Executing:**********
INFO [2018-07-12 02:46:25] SELECT 42
INFO [2018-07-12 02:46:25] ********************
INFO [2018-07-12 02:46:25] Finished in 0.001331 secs returning 1 rows
INFO [2018-07-12 02:46:25] Closing connection to sqlite
```

db_query features: connections

SYSTEM1

```
> db_query('SELECT 42', 'sqlite')
INFO [2018-07-12 02:46:25] Connecting to sqlite
INFO [2018-07-12 02:46:25] Executing:**********
INFO [2018-07-12 02:46:25] SELECT 42
INFO [2018-07-12 02:46:25] ********************
INFO [2018-07-12 02:46:25] Finished in 0.001331 secs returning 1 rows
INFO [2018-07-12 02:46:25] Closing connection to sqlite
```

Reusing connections:

```
> con <- db_connect('sqlite')
INFO [2018-07-12 02:54:17] Connecting to sqlite
> db_query('SELECT 42', con)
INFO [2018-07-12 02:53:48] Executing:**********
INFO [2018-07-12 02:53:48] SELECT 42
INFO [2018-07-12 02:53:48] ********************
INFO [2018-07-12 02:53:48] Finished in 0.0009012 secs returning 1 rows
```

db_query features: connections

SYSTEM1

```
> con <- db_connect('sqlite')
INFO [2018-07-12 02:58:00] Connecting to sqlite
> db_query('SELECT 42', 'sqlite')
INFO [2018-07-12 02:58:07] Connecting to sqlite
INFO [2018-07-12 02:58:07] Executing:**********
INFO [2018-07-12 02:58:07] SELECT 42
INFO [2018-07-12 02:58:07] ********************
INFO [2018-07-12 02:58:07] Finished in 0.0007277 secs returning 1 rows
INFO [2018-07-12 02:58:07] Closing connection to sqlite
```

db_query features: connections

SYSTEM1

```
> con <- db_connect('sqlite')
INFO [2018-07-12 02:58:00] Connecting to sqlite
> db_query('SELECT 42', 'sqlite')
INFO [2018-07-12 02:58:07] Connecting to sqlite
INFO [2018-07-12 02:58:07] Executing:**********
INFO [2018-07-12 02:58:07] SELECT 42
INFO [2018-07-12 02:58:07] ********************
INFO [2018-07-12 02:58:07] Finished in 0.0007277 secs returning 1 rows
INFO [2018-07-12 02:58:07] Closing connection to sqlite
```

```
> con <- db_connect('sqlite', cache = TRUE)
INFO [2018-07-12 02:59:31] Connecting to sqlite
> db_query('SELECT 42', 'sqlite')
INFO [2018-07-12 02:59:33] Executing:**********
INFO [2018-07-12 02:59:33] SELECT 42
INFO [2018-07-12 02:59:33] ********************
INFO [2018-07-12 02:59:33] Finished in 0.0009344 secs returning 1 rows
```

db_query features: connections

SYSTEM1

```
> con <- db_connect('sqlite')
INFO [2018-07-12 02:58:00] Connecting to sqlite
> db_query('SELECT 42', 'sqlite')
INFO [2018-07-12 02:58:07] Connecting to sqlite
INFO [2018-07-12 02:58:07] Executing:**********
INFO [2018-07-12 02:58:07] SELECT 42
INFO [2018-07-12 02:58:07] ********************
INFO [2018-07-12 02:58:07] Finished in 0.0007277 secs returning 1 rows
INFO [2018-07-12 02:58:07] Closing connection to sqlite
```

```
> con <- db_connect('sqlite', cache = TRUE)
INFO [2018-07-12 02:59:31] Connecting to sqlite
> db_query('SELECT 42', 'sqlite')
INFO [2018-07-12 02:59:33] Executing:**********
INFO [2018-07-12 02:59:33] SELECT 42
INFO [2018-07-12 02:59:33] ********************
INFO [2018-07-12 02:59:33] Finished in 0.0009344 secs returning 1 rows
```

```
> db_close(db_connect('sqlite', cache = FALSE))
INFO [2018-07-12 02:59:55] Connecting to sqlite
INFO [2018-07-12 02:59:55] Closing connection to sqlite
```

db_query features: forking

SYSTEM1

```
> system.time(db_config('shinydemo'))
   user  system elapsed
  3.359   0.092   5.236
```

db_query features: forking

SYSTEM1

```
> system.time(db_config('shinydemo'))
   user  system elapsed
  3.359   0.092   5.236
```

```
> system.time(db_config('shinydemo'))
   user  system elapsed
  0.001   0.000   0.001
```

db_query features: forking

SYSTEM1

```
> system.time(db_config('shinydemo'))
   user  system elapsed
  3.359   0.092   5.236
```

```
> system.time(db_config('shinydemo'))
   user  system elapsed
  0.001   0.000   0.001
```

It's not caching the connection this time, only the credentials.

db_query features: forking

SYSTEM1

```
> system.time(db_config('shinydemo'))
   user  system elapsed
  3.359   0.092   5.236
```

```
> system.time(db_config('shinydemo'))
   user  system elapsed
  0.001   0.000   0.001
```

It's not caching the connection this time, only the credentials.

```
> db_config_invalidate_cache()
INFO [2018-07-12 03:03:38] Invalidating cache on already loaded DB config(s)
```

db_query features: forking

SYSTEM1

```
> parallel::mclapply(1:16, function(i) db_query("SELECT 42", "sqlite"), mc.cores = 8
INFO [2018-07-12 03:05:04] Connecting to sqlite
INFO [2018-07-12 03:05:04] Executing:**********
INFO [2018-07-12 03:05:04] SELECT 42
INFO [2018-07-12 03:05:04] Connecting to sqlite
INFO [2018-07-12 03:05:04] ********************
INFO [2018-07-12 03:05:04] Executing:**********
INFO [2018-07-12 03:05:04] Finished in 0.001053 secs returning 1 rows
INFO [2018-07-12 03:05:04] SELECT 42
INFO [2018-07-12 03:05:04] Closing connection to sqlite
INFO [2018-07-12 03:05:04] ********************
INFO [2018-07-12 03:05:04] Connecting to sqlite
INFO [2018-07-12 03:05:04] Connecting to sqlite
INFO [2018-07-12 03:05:04] Executing:**********
INFO [2018-07-12 03:05:04] Finished in 0.005117 secs returning 1 rows
INFO [2018-07-12 03:05:04] SELECT 42
INFO [2018-07-12 03:05:04] Closing connection to sqlite
INFO [2018-07-12 03:05:04] ********************
INFO [2018-07-12 03:05:04] Connecting to sqlite
INFO [2018-07-12 03:05:04] Executing:**********
INFO [2018-07-12 03:05:04] Connecting to sqlite
INFO [2018-07-12 03:05:04] Finished in 0.003497 secs returning 1 rows
INFO [2018-07-12 03:05:04] SELECT 42
INFO [2018-07-12 03:05:04] ********************
```

Simple benchmarks

SYSTEM1

```
> devnull <- db_config('sqlite')
> devnull <- db_config('shinydemo')
> sql <- 'SELECT CURRENT_TIMESTAMP, 42'

> library(microbenchmark)
> sqlite <- function() db_query(sql, 'sqlite')
> mysql  <- function() db_query(sql, 'shinydemo')

> microbenchmark(sqlite(), mysql(), times = 10)
Unit: milliseconds
     expr        min         lq       mean     median         uq       max
 sqlite()   10.69874   12.13927   17.94149   14.96143   26.23301   29.6122
  mysql() 1066.12648 1124.29127 1221.90519 1196.29284 1214.73232 1687.4414
```

Extending DBI to be able to use db_query

SYSTEM1

AWR.Athena: 'AWS' Athena 'DBI' Wrapper

'RJDBC' based 'DBI' driver to Amazon Athena, which is an interactive query service to analyze data in Amazon 'S3' using standard 'SQL'.

Version:	1.1.0-1
Imports:	methods, RJDBC, rJava
Published:	2017-11-19
Author:	Neal Fultz, Gergely Daróczi
Maintainer:	Neal Fultz <nfultz at gmail.com>
License:	AGPL-3
NeedsCompilation:	no
Materials:	README
CRAN checks:	AWR.Athena results

Downloads:

Reference manual:	AWR.Athena.pdf
Package source:	AWR.Athena_1.1.0-1.tar.gz
Windows binaries:	r-devel: AWR.Athena_1.1.0-1.zip, r-release: AWR.Athena_1.1.0-1.zip, r-oldrel: AWR.Athena_1.1.0-1.zip
OS X binaries:	r-release: AWR.Athena_1.1.0-1.tgz, r-oldrel: AWR.Athena_1.1.0-1.tgz
Old sources:	AWR.Athena archive

Extending DBI to be able to use db_query

SYSTEM1

Branch: master ▾ AWR.Athena / R / athena.R

Find file | Copy path

nfultz Fixing docs for R CMD check 1c47b89 on Nov 7, 2017

2 contributors

124 lines (110 sloc) | 3.82 KB

Raw | Blame | History

```r

#' Athena driver class.
#'
#' @keywords internal
#' @export
#' @import RJDBC
#' @import methods
#' @importClassesFrom RJDBC JDBCDriver
setClass("AthenaDriver", contains = "JDBCDriver")

#' Athena DBI wrapper
#'
#' @export
Athena <- function() {
  new("AthenaDriver")
}

#' Constructor of AthenaDriver
#'
#' @name AthenaDriver
#' @rdname AthenaDriver-class
setMethod(initialize, "AthenaDriver",
   function(.Object, ...)
{
    # passed to parent builder, than unboxed, yuck
    # should ping RJDBC maintainers, and have them implement initialize methods instead
    jdbc <- JDBC(driverClass="com.amazonaws.athena.jdbc.AthenaDriver",
                 identifier.quote="'")

```

Extending DBI to be able to use db_query

SYSTEM1

Branch: master ▾ AWR.Snowflake / R / zzz.R

Find file | Copy path

daroczig create the java folder as there's no placeholder there — fd4d1aa on Nov 2, 2017

2 contributors

46 lines (34 sloc) | 1.55 KB

Raw | Blame | History

```r
#' @importFrom utils packageVersion download.file
#' @importFrom rJava .jpackage
.onLoad <- function(libname, pkgname) {

    ## path to the java folder
    path <- paste0(system.file('', package = pkgname), 'java')
    if (!file.exists(path)) {
        dir.create(path)
    }

    ## path to the JDBC driver
    file <- sprintf('snowflake-jdbc-%s.jar', packageVersion(pkgname))
    path <- file.path(path, file)

    ## check if the jar is available and install if needed (on first load)
    if (!file.exists(path)) {

        url <- file.path(
            'https://repo1.maven.org/maven2/net/snowflake/snowflake-jdbc',
            packageVersion(pkgname), file)

        ## download the jar file from Maven
        try(download.file(url = url, destfile = path, mode = 'wb'),
            silent = TRUE)

    }

    ## add the RJDBC driver and the log4j properties file to classpath
    rJava::.jpackage(pkgname, lib.loc = libname)
```

A more complex benchmark

SYSTEM1

Transaction Processing Performance Council (TPC) Benchmarks ™ 10 TB version:

```
with v1 as(
  select i_category, i_brand, cc_name, d_year, d_moy,
        sum(cs_sales_price) sum_sales,
        avg(sum(cs_sales_price)) over
          (partition by i_category, i_brand,
                     cc_name, d_year)
          avg_monthly_sales,
        rank() over
          (partition by i_category, i_brand,
                     cc_name
           order by d_year, d_moy) rn
  from tpcds_sf10tcl.item, tpcds_sf10tcl.catalog_sales, tpcds_sf10tcl.date_dim, tpcds_sf10tcl.call_center
  where cs_item_sk = i_item_sk and
       cs_sold_date_sk = d_date_sk and
       cc_call_center_sk= cs_call_center_sk and
       (
         d_year = {year} or
         ( d_year = {year}-1 and d_moy =12) or
         ( d_year = {year}+1 and d_moy =1)
       )
  group by i_category, i_brand,
          cc_name , d_year, d_moy),
v2 as(
  select v1.i_category ,v1.d_year, v1.d_moy ,v1.avg_monthly_sales
        ,v1.sum_sales, v1_lag.sum_sales psum, v1_lead.sum_sales nsum
  from v1, v1 v1_lag, v1 v1_lead
  where v1.i_category = v1_lag.i_category and
       v1.i_category = v1_lead.i_category and
       v1.i_brand = v1_lag.i_brand and
       v1.i_brand = v1_lead.i_brand and
       v1.cc_name = v1_lag.cc_name and
```

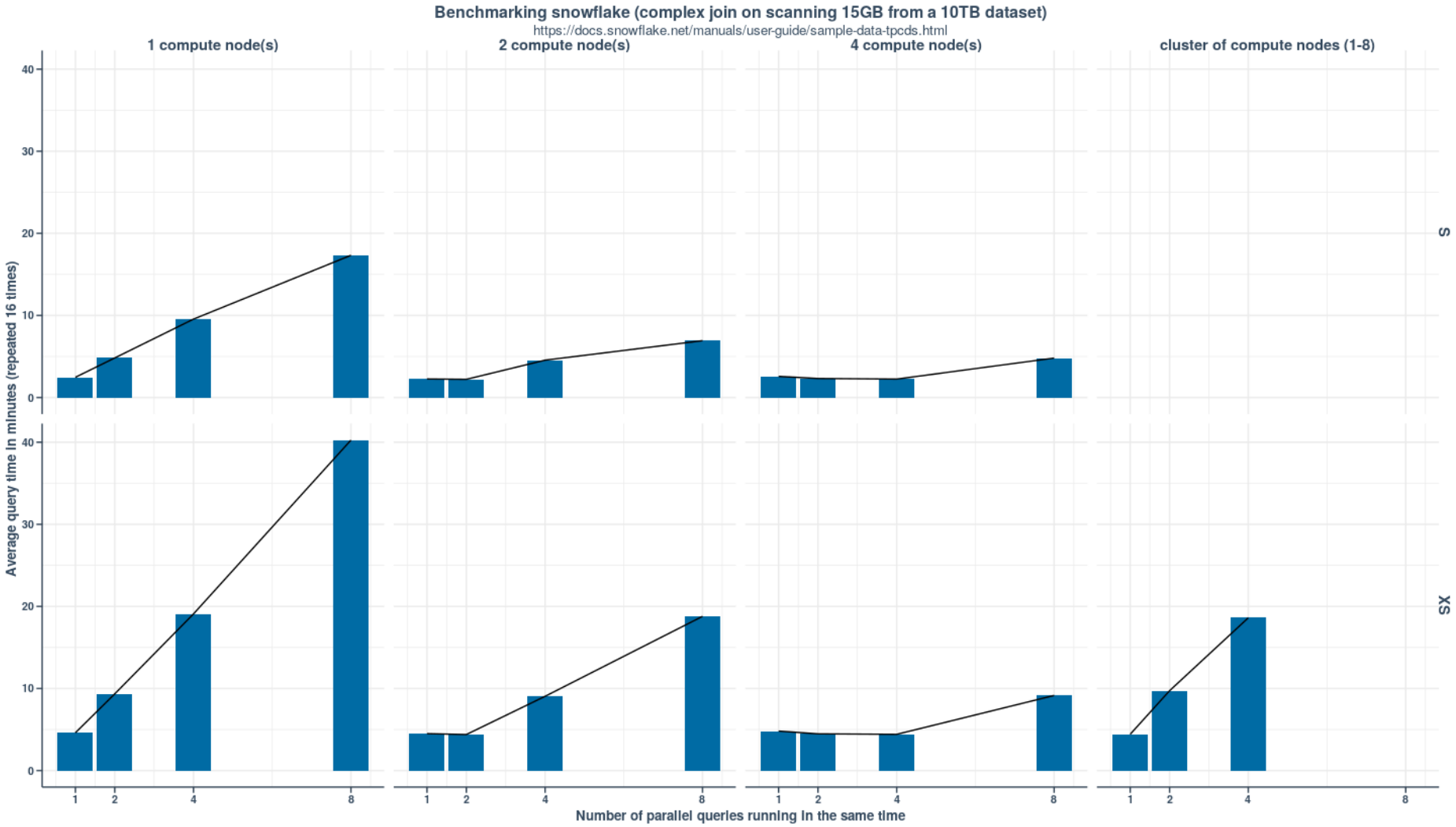
A more complex benchmark

SYSTEM1

```
warehouse_type <- 'S'
warehouses <- 2
threads <- 4

con1 <- db_connect('snowflake', warehouse = paste0(warehouse_type, 1))
con2 <- db_connect('snowflake', warehouse = paste0(warehouse_type, 2))
con3 <- db_connect('snowflake', warehouse = paste0(warehouse_type, 3))
con4 <- db_connect('snowflake', warehouse = paste0(warehouse_type, 4))

mclapply(1:16, function(i, warehouse, threads) {
    timing <- system.time(db_query(
        query,
        db = get(paste0('con', i %% warehouses + 1)),
        year = 1999))
    data.table(
        warehouse = warehouse,
        threads = threads,
        time = timing[[3]])
}, mc.cores = threads, threads = threads)

for (i in 1:4) {
    db_close(get(paste0('con', i %% 4 + 1)))
}
```

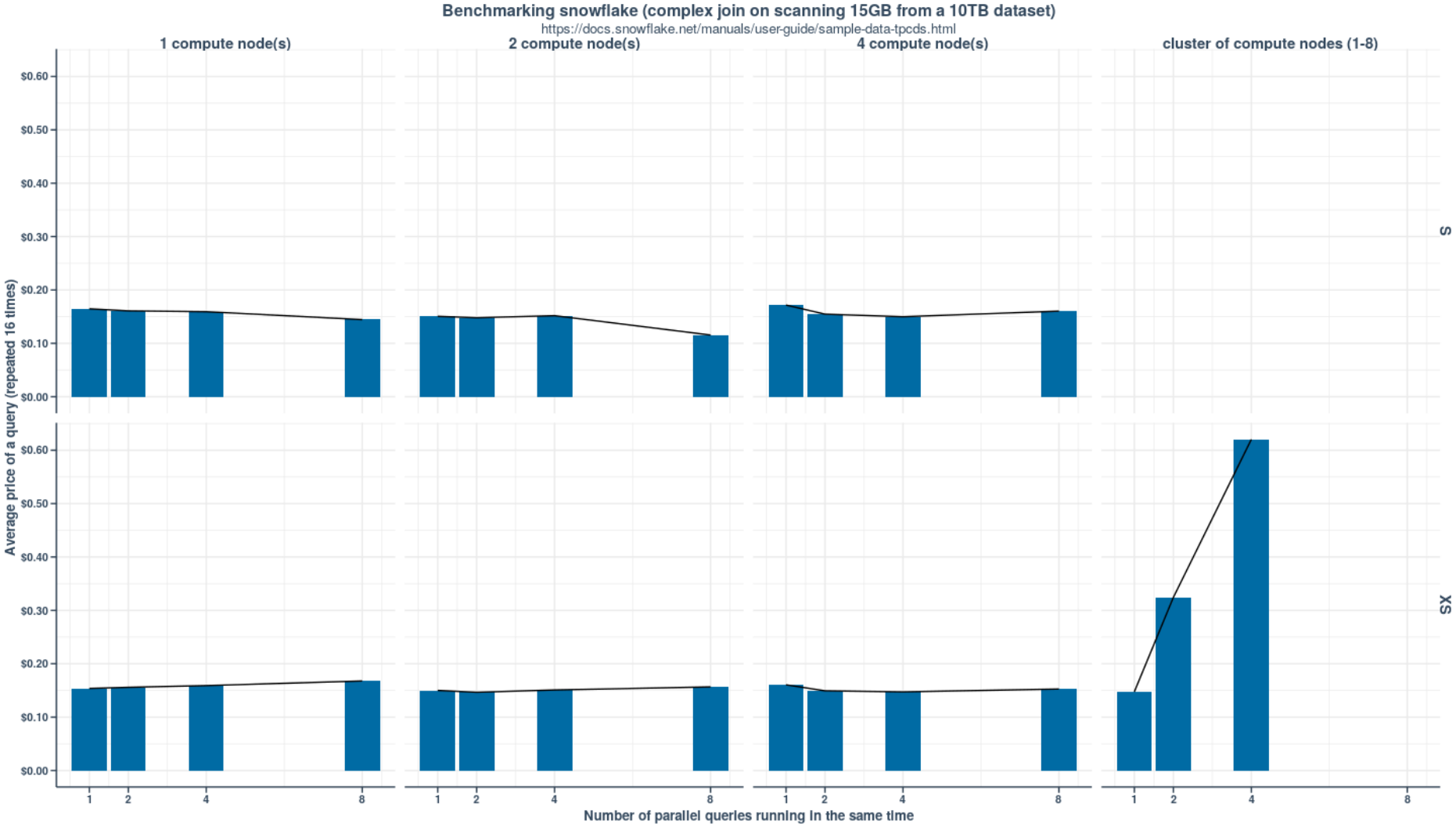
A more complex benchmark

SYSTEM1



A more complex benchmark

SYSTEM1



What's Next?

SYSTEM1

- optional connection pooling with `rstudio/pool`
- PR to consolidate `db_config` with `rstudio/config`
- open-source `kmsdata` method to store encrypted R objects in YAML
- add more YAML methods to decrypt data, eg outside of AWS
- open-source `glue` extension and make use of `glue_sql`
- bump on `zatonovo/futile.logger/pull/73` for adding `glue` support over `sprintf` when logging
- open-source SQL inserts and upserts
- return object (being `data.table` internally due to efficiency, eg setting `attr`, but left it as `data.frame` here – will make it configurable)

http://github.com/daroczig/dbr