Moving Projects into Production (Learnings from working at a Tech Startup)

Christopher Collins

September 2019

Glovo

About Me

- Data Scientist at Glovo (since April 2019), working in the Customer Intelligence & Insights team
- Gravitated towards Data Science whilst trying to predict football games in university
- First job as a Data Scientist in the UK, using R, SQL and Tableau
- Moved to Glovo after 2 years, now I’m primarily using Python & SQL, but still try to keep up with R

Aims of the talk?

- Main focus on ‘productionising’ projects and good coding practices
- What kind of projects? - Any projects designed to do something, with more of a focus on projects that output data regularly and have a high business impact
- Productionising? Many factors, will get onto that!
- Lots to cover.. Presentation = high-level overview + lots of guides/articles/examples

R isn’t for production?

- Everyone benefits from projects that are robust, well written and can be automated (if required), even if the project is a simple report.

Reality

Team Members Intern Team You

R isn’t for production?

Making projects production ready

To me, a production level project is one that is:

1. **Readable**
2. **Robust**
3. **Version Controlled**
4. **Modular**
5. **Standalone**
6. **Automatable**
7. **Documented**

Making projects production ready

To me, a production level project is one that is:

1. **Readable** - Easy to understand, consistent structure
2. **Robust** - Hard to break, easy to fix
3. **Version Controlled** - Easy to track changes, easy for others to collaborate
4. ~~**Modular** - Broken down into small, manageable pieces~~
5. ~~**Standalone** - Can run on other systems without issues~~ https://www.docker.com/
6. **Automatable** - Does not require someone to ‘press play’ on a regular basis
7. ~~**Documented** - Everything required to run/maintain the project in one place~~

Example

Take a .csv stored online, and host it as a Dash web app.

Download Data

```r
# Extract data from a given url (the url must point to a .csv file)
download_data <- function(url) {
  # Download data from url
  output_table <- read.csv(url, stringsAsFactors = FALSE)
  # Perform some operations
  output_table <- do_something(output_table)
  # Return the output
  return(output_table)
}
```

Host Web App

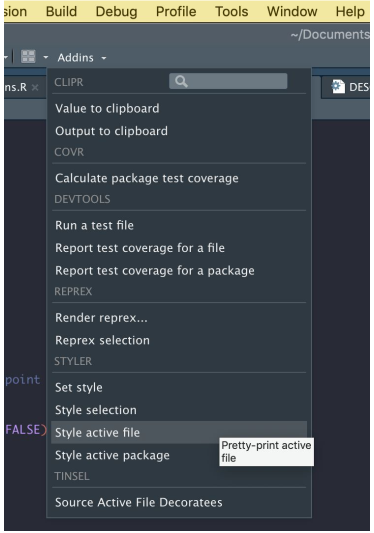
```r
create_dash_server <- function(input_table) {
  # Create an app object
  app <- Dash$new()
  # Define columns used in data table
  columns <- lapply(colnames(input_table),
         function(colName) {list(id = colName, name = colName)}
         )
  # Define the layout (including the table)
  app$layout(
    dashDataTable(
      id = "table",
      columns = columns,
      data = df_to_list(input_table),
      # Enable sorting
      sort_action = 'native',
      # Enable filtering
      filter_action = 'native',
    )
  )
  # Start the server
  app$run_server() # Default port 8050
}
```

Readability - Style Guides

“Good coding style is like correct punctuation: you can manage without it, butitsuremakesthingseasiertoread”

https://style.tidyverse.org/ <- Tidyverse Style Guide (recommended)

https://google.github.io/styleguide/Rguide.html <- Google’s R style guide



Useful Packages (for Tidyverse Style Guide)

Lintr (passive) - Automatic checking that you are conforming to the style guide while coding
Styler (aggressive) - Restyle your code automatically

```
# Extract data from a given url (the url must point to a .csv file)
download_data <- function(
  url) {
  # Download data from url
  fb_data <-  read.csv(     url, stringsAsFactors = FALSE)
  #   Perform some operations
  output_table <- do_something(fb_data
                               )

  # Return the result
  return(     output_table)
}
```

```
# Extract data from a given url (the url must point to a .csv file)
download_data <- function(url) {
  # Download data from url
  fb_data <-   read.csv(url, stringsAsFactors = FALSE)
  #  Perform some operations
  output_table <- do_something(fb_data)
  # Return the result
  return(output_table)
}
```

Robustness

- **Making code easy to fix** - Testing and logging
- **Why tests?**
  - You can never think of everything that could go wrong
  - Pinpoint exactly where an issue occured
  - Add new features without fear
- **Types of tests:**
  - **“Offline Testing”** - Unit test, Integration Test, System Test
  - **“Run-Time Testing”** - Assert statements
- Great guide on unit testing:
  [Unit testing with test_that]
  [Integration testing and more]

“Input, Execute, and Assert”

R: testthat

tests/testthat/

Python: unittest, nose

```
# Extract data from a given url (the url must point to a .csv file)
download_data <- function(url) {
  # Download data from url
  output_table <- read.csv(url, stringsAsFactors = FALSE)
  # Perform some operations
  output_table <- do_something(output_table)
  # Return the output
  return(output_table)
}
```

```
# UNIT TESTS
## Test that do_something() works correctly on dummy data
test_that("do_something() returns expected output", {
  expect_equal(ds_dummy_output, do_something(dummy_input))
})
## Test that the url exists
test_that("url exists", {
  expect_true(url.exists(url))
})

# INTEGRATION TEST
## Test that the download data function works completely
test_that("download_data returns expected output", {
  expect_equal(dd_dummy_output, download_data(url))
})
```

Use the url of something that won’t change!

Runtime Testing

- Use in addition to other tests to increase robustness
- **Assertthat package** - Clean assert statements with custom messages

```
# Extract data from a given url (the url must point to a .csv file)
download_data <- function(url) {
  # Download data from url
  output_table <- read.csv(url, stringsAsFactors = FALSE)
  # Perform some operations
  output_table <- do_something(output_table)
  # Return the output
  return(output_table)
}
```

```
download_data <- function(url) {
  # Test that the input url is the correct format
  assert_that(url.exists(url),
              msg="Url does not exist, aborting")
  # Download data from url
  output_table <- read.csv(url, stringsAsFactors = FALSE)
  # Perform some operations
  output_table <- do_something(output_table)
  # Test that the number of rows in the dataFrame is not less than 1
  assert_that(nrow(output_table) > 1,
              msg="Rows of output_table less than 1 after
                  returning from do_something(), aborting.")
  # Return the output
  return(output_table)
}
```

Ditching print() for loggers

- Walk users through your code as it runs, with varying levels of detail.
- You want detailed messages for debugging, but not see them every time you run the project
- Save messages to a file (log file) for post-execution analysis.
- Good logging can save HOURS of debugging

```
INFO:root:Function get_churn_metrics completed in 96 seconds
INFO:root:Running get_latest_churn_rate()
INFO:root:Loading existing file and appending new data if applicable
INFO:root:Rows to be added: 0
File /Users/chriscollins/Projects/TEST/customer-churn/data/churn_perfo
and will not be overwritten, so obtained results will be lost.
INFO:root:Function get_latest_churn_rate completed in 10 seconds
INFO:root:Running get_normal_churn_rate()
INFO:root:Running complete refresh of data
INFO:root:Function get_normal_churn_rate completed in 43 seconds
INFO:root:Running get_confusion_matrix_data()
INFO:root:Running complete refresh of data
```

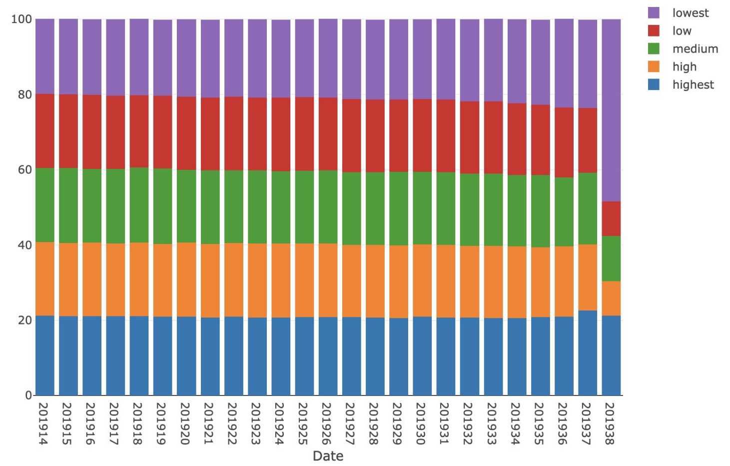
R Packages: logging, logger
Python: logger

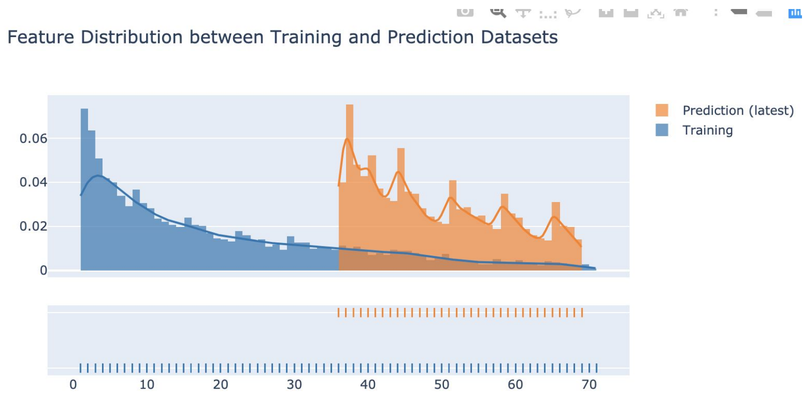
```r
# Extract data from a given url (the url must point to a .csv file)
download_data <- function(url) {
  log_info(paste("Executing download_data()"))
  # Start timing
  tic()
  # Test that the input url is the correct format
  assert_that(is.character(url),
              msg=paste0("Url is ",class(url),", not character, aborting."))
  # Download data from url
  output_table <- read.csv(url, stringsAsFactors = FALSE)
  # Log dimensions of downloaded table (for debugging)
  log_debug(paste0("Table downloaded. Downloaded table has ",
                   nrow(output_table)," rows and ", ncol(output_table), " columns."))
  # Test that the number of rows in the dataFrame is not less than 1
  assert_that(nrow(output_table) > 1,
              msg="Rows of downloaded table less than 1, aborting.")
  # Perform some operations
  output_table <- do_something(output_table)
  # Log dimensions of edited table (for debugging)
  log_debug(paste0("After applying do_something(), output_table has ",
                   nrow(output_table)," rows and ", ncol(output_table), " columns."))
  # Test that the number of rows in the dataFrame is not less than 1
  assert_that(nrow(output_table) > 1,
              msg="Rows of output_table less than 1 after returning from do_something(), aborting.")
  # Log the time taken to run the function
  run_time <- toc(quiet = TRUE)
  log_info(paste("download_data() completed in",
                 as.character(round(run_time$toc - run_time$tic, 3)),
                 "seconds"))
  # Return the output
  return(output_table)
}
```

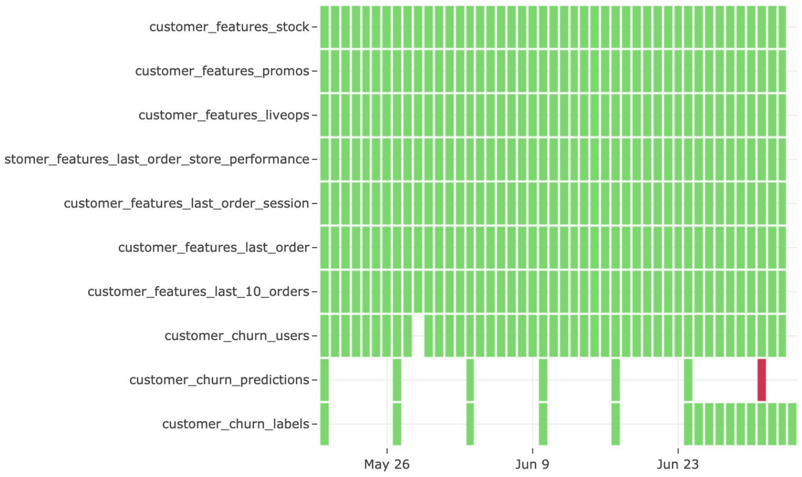
```
INFO [2019-09-24 13:39:59] Executing download_data()
INFO [2019-09-24 13:39:59] Executing download_data()
DEBUG [2019-09-24 13:40:00] Table downloaded. Downloaded table has 60 rows and 106 columns.
INFO [2019-09-24 13:40:00] Executing do_something()
INFO [2019-09-24 13:40:00] do_something() completed in 0.043 seconds
DEBUG [2019-09-24 13:40:00] After applying do_something(), output_table has 60 rows and 106 columns.
INFO [2019-09-24 13:40:00] download_data() completed in 0.483 seconds
```

Model performance dashboards

- Help you to find the cause of problems much faster.
- Check that model inputs and outputs are as expected.
- Check that features follows the same distributions as they did when the model was trained.
- Check for duplicates

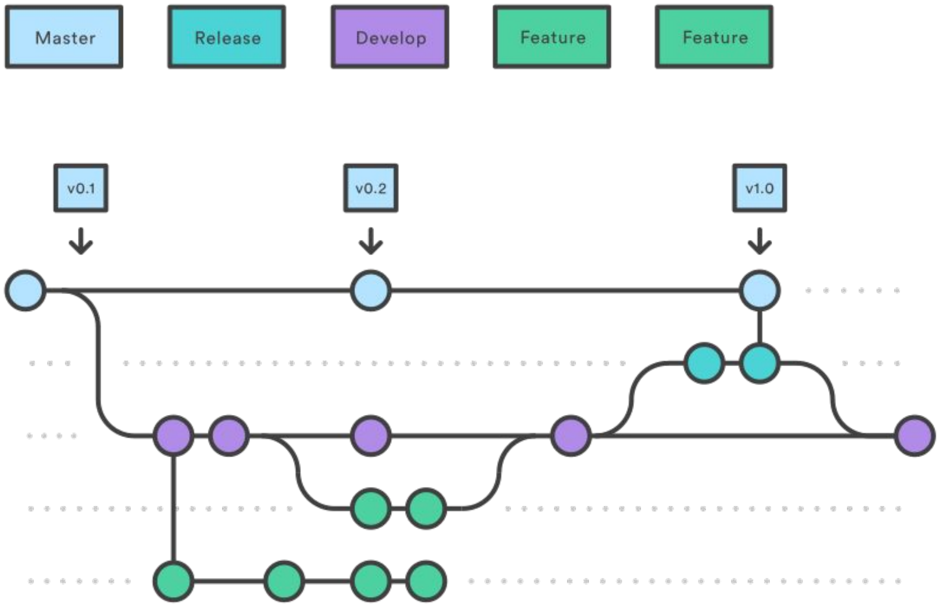






Version Control (Using GitHub as an example)

- GitHub lets you run multiple versions (branches) of a project in parallel, allowing you to develop/test changes without affecting the original (master) branch.
- Testing all changes on a development branch minimises the chance of error (essential for anything high-impact).
- Packages can be installed by GitHub (super handy!
- Promotes good coding practices (especially if other team members check your code!)
- Free private repos for up to 4 contributors (5 for BitBucket)

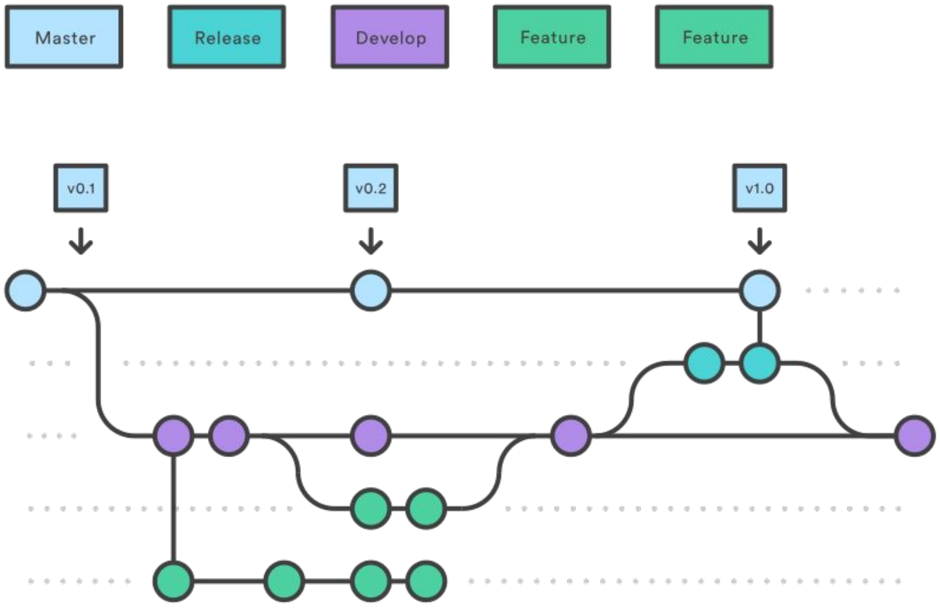


Version Control (Using GitHub as an example)

Why GitHub has helped me develop as a data scientist:

- Knowing other people will be reading my code helps me focus on making it more understandable.
- Reading other people's code makes me realise the importance of style guides
- Branching lets me work on lots of experimental features without affecting the project operation
- Easier to figure out where bugs were introduced

Great guide to get started:
https://guides.github.com/introduction/flow/

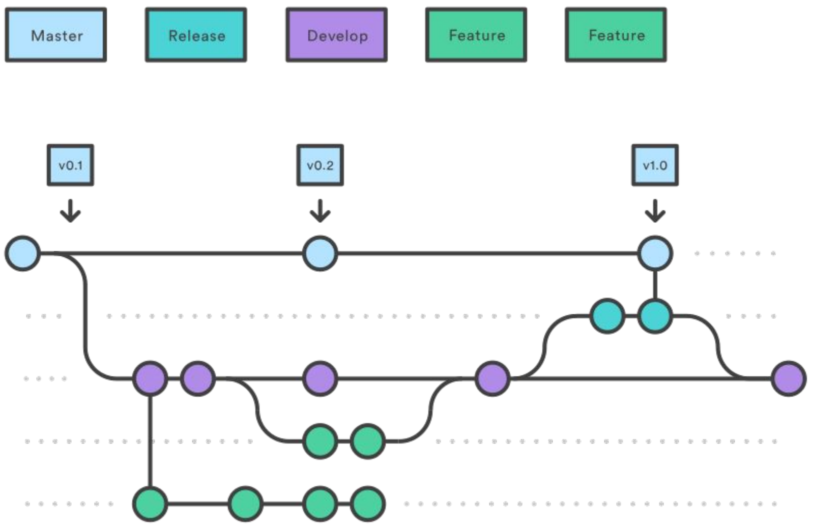


Version Control - Pre-commit Hooks

- **Bonus:** You can set up hooks, which perform operations on your code when before/after you commit changes to an online branch (other options available).

- Examples include:
  - **Code formatting**: Formats all of your code to adhere to a particular style guide)
  - **Linting**: Searches your code for potential run-time errors
  - Check that all files contain valid R code
  - Check that packages are ordered alphabetically

- (Guide for implementing pre-commit hooks in R using GitHub): https://github.com/lorenzwalthert/pre-commit-hooks



Pre-commit works for any language!

Automation - EC2 (AWS)

What is EC2? - Cloud computing that allows you to rent small servers and run code. Advantages:

- Always running, very small chance of downtime as the server is maintained by amazon.
- Pay for what you use
- Easy to set up R-Studio Server! (link below)
- Expose ports to local network to let others view your web apps

Setting up an EC2 server (Amazon Guide)
Setting up EC2 & R Studio

EC2

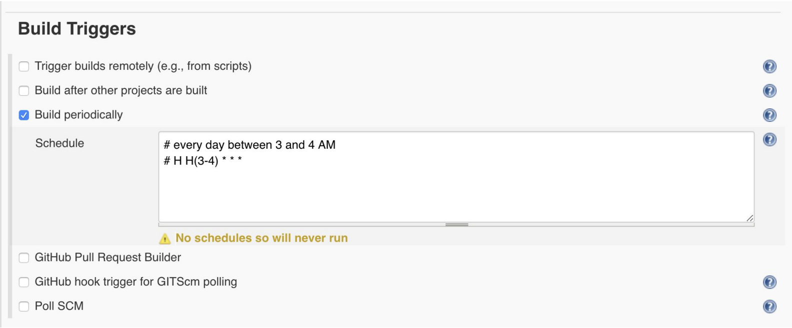
Run multiple projects simultaneously using ‘screen’ inside EC2
https://linuxize.com/post/how-to-use-linux-screen/

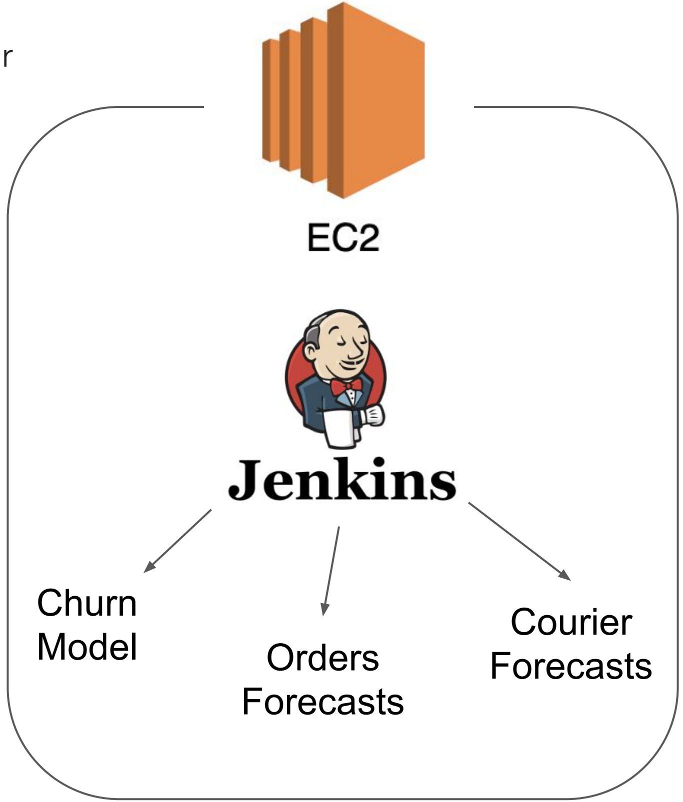
Automation - Job Scheduling with Jenkins and EC2

What is Jenkins? - Open-source job scheduling tool that works for almost any combination of languages and repositories.
Advantages:

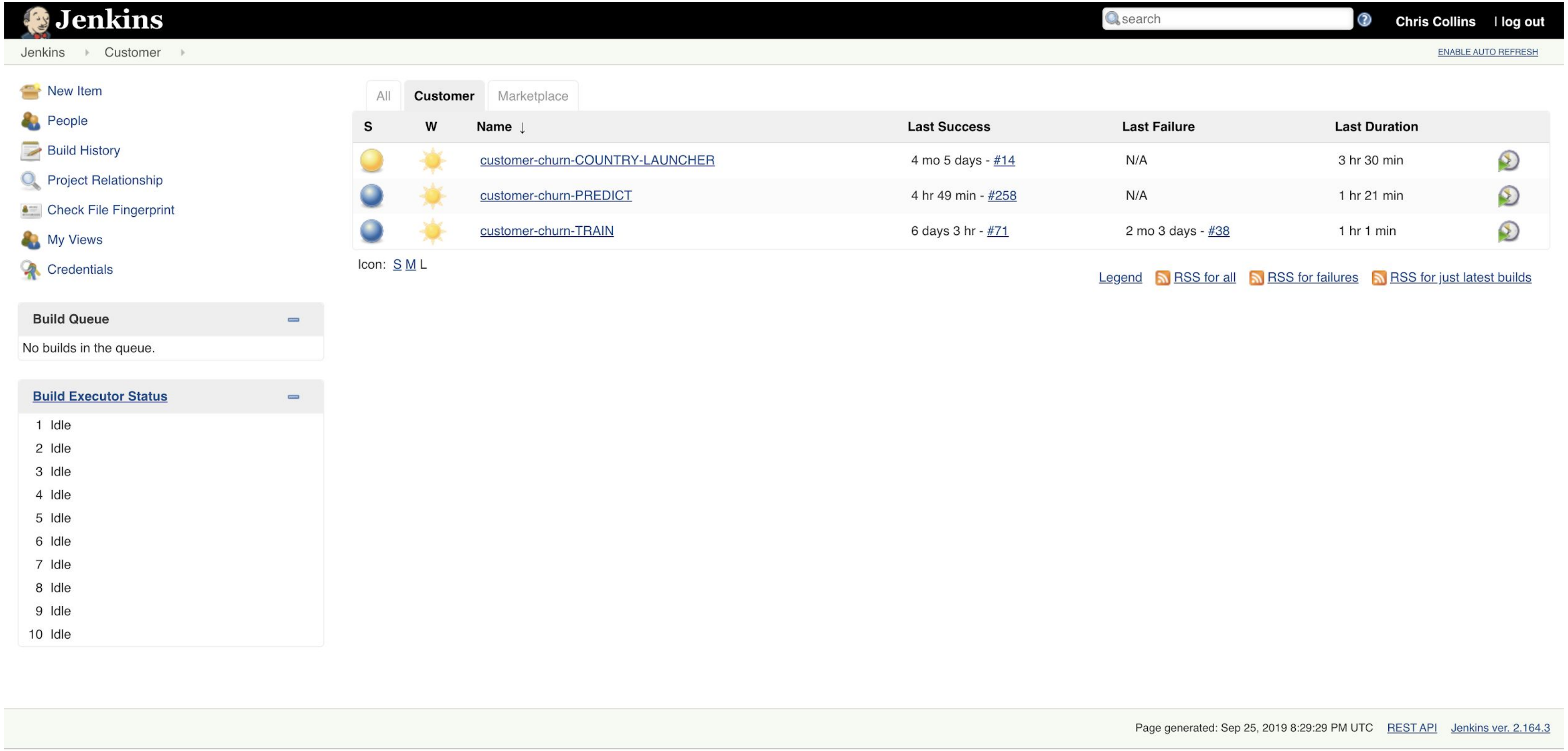
- Scheduled execution of projects
- Connect Jenkins and GitHub through web-hooks to trigger build actions
- Email alerts if a build fails

[Jenkins: The Definitive Guide (400 page book)](#)
[Running Jenkins on EC2](#)





Jenkins Home Dashboard



Thanks!

https://github.com/qemtek/footballTableDemo

Modification to EC2 setup code to get R Studio to work (if it dosent first time)

Paste this code into the ‘Advanced Details’ section when you create the EC2 instance, it worked for me :)

```
#!/bin/bash
#install R
yum install -y R

#install RStudio-Server 1.0.153 (2017-07-20)
wget https://download2.rstudio.org/rstudio-server-rhel-1.0.153-x86_64.rpm
yum install -y --nogpgcheck rstudio-server-rhel-1.0.153-x86_64.rpm
yum install libxml2-devel
yum install libcurl-devel
yum install openssl-devel

rm rstudio-server-rhel-1.0.153-x86_64.rpm

#install shiny and shiny-server (2017-08-25)
R -e "install.packages('shiny', repos='http://cran.rstudio.com/')"
wget https://download2.rstudio.org/server/centos6/x86_64/rstudio-server-rhel-1.2.5001-x86_64.rpm
sudo yum install rstudio-server-rhel-1.2.5001-x86_64.rpm
Rm  rstudio-server-rhel-1.2.5001-x86_64.rpm

#add user(s)
```