ARCHITECTURE DESIGN DOCUMENT

Healthcare Analytics on Heart Disease Data

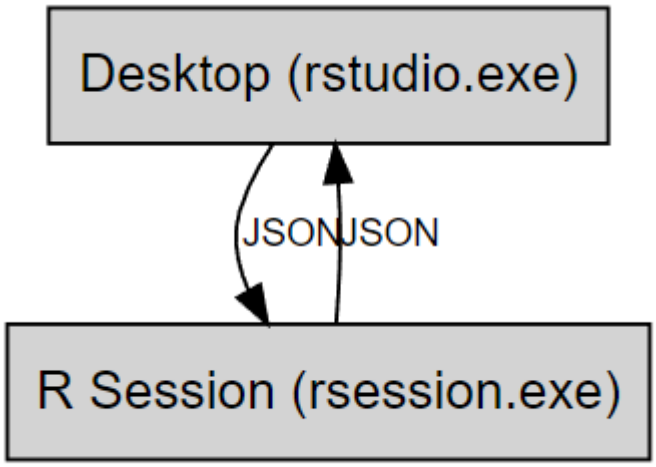
WRITTEN BY :

Atmik Pandey

Overview

RStudio Desktop

RStudio is at its heart a client-server application. Let's look at RStudio Desktop first as it's the simplest configuration, consisting only of a single client and a single server. Even though it appears to the user a single application, these two processes communicate over a socket using JSON.



rstudio.exe

This process functions much like a web browser. And in fact, it very nearly *is* a web browser. On Windows (`rstudio.exe`) and Linux (`rstudio`), the desktop executable is little more than a Qt window frame with some menus and a `QtWebKit` web browser control. On macOS, the desktop frame is written in Cocoa for performance reasons, but the architecture is otherwise identical.

rsession.exe

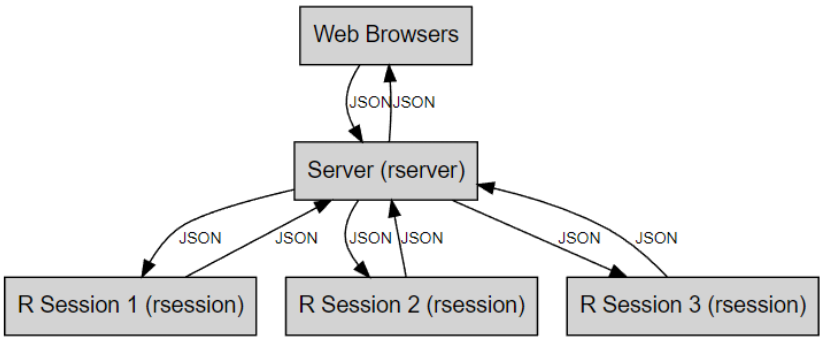
This is the process containing the actual R session. It isn't quite the same as starting R at the command line as it doesn't run the same binary you get when you type `R` at the console. Instead, it loads R as a library – a DLL on windows, and a shared object file on Linux and macOS – and calls the library when needed to evaluate R code, process console input, and so on.

The R session process acts much like a web server in the desktop configuration, but we'll avoid referring to it as the "server" for reasons which will soon become clear.

RStudio Server

The open-source version of RStudio Server is much like RStudio Desktop, except that a web browser functions as the client, and there can be multiple sessions (one per

user). It also introduces a new process called `rserver`, which is responsible for routing traffic from web browsers to sessions.



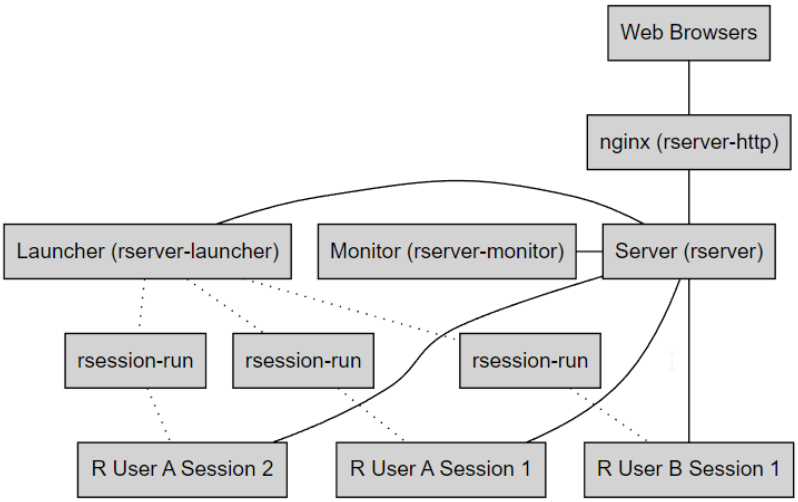
The communication between the web browser and the server happens over the network, and the communication between the server and the individual R sessions happens over a Unix domain socket.

The `rserver` process is also responsible for serving the login page and starting sessions. When a user visits the RStudio URL, it will check to see if that user already has a session running, and:

1. If they have a session running, redirect them to that session.
2. If not, start a new session (a new `rsession`), and redirect them to the new session.

RStudio Server Pro

RStudio Server Pro introduces some more moving parts. In RStudio Server Pro (hereafter "RSP"), there can be multiple sessions for each user. Sessions are also launched differently, in order to support features like PAM sessions.



rserver-launcher

This is the process which actually launches sessions. It receives launch requests from the `rserver` process, and runs the `rsession-run` script to start the session.

rserver-http

RSP includes a bundled copy of Nginx, which is used as a reverse proxy to feed HTTP traffic into the main binary. The Nginx configuration is dynamically generated each time RStudio Server starts.

rserver-monitor

The "monitor" process is responsible for RSP's metrics and monitoring features.

Authentication

PAM

When PAM authentication is used, RStudio uses a helper binary called `rserver-pam` to validate the username and password. This binary is launched with the username as an argument; the password is sent from RStudio to the `rserver-pam` executable over standard input, after which `rserver-pam` passes it on to PAM itself. RStudio uses `rserver-pam`'s exit code to determine whether the authentication was successful.

It's not possible to run `rserver-pam` directly, but RStudio comes with a `pamtester` utility which can be used to invoke it for testing purposes.

Sessions

In the open source version of RStudio, PAM is used only when the user signs in. However, in many environments, PAM modules also need to run to initialize the R session; for instance, to set environment variables, or issue a Kerberos ticket for use with the session. RStudio Server Pro makes this possible by delegating session launches to a dedicated binary named `rserver-launcher`.

You can disable PAM sessions by using `auth-pam-sessions-enabled=0` in `rserver.conf`. If you do this, you'll notice that `rserver-launcher` is not used at all; instead, Rstudio will launch sessions directly.

Session Management

One of the primary functions performed by RStudio Server is the creation and management of R sessions. Understanding everything that happens when a session is started and how its lifetime is managed is invaluable in troubleshooting!

Starting an R Session

R itself performs a large number of tasks when it starts up, many of which can run arbitrary code. They are all documented in one of R's most useful help pages, R: Initialization at the Start of a Session. In RStudio Desktop and RStudio Server, there is little else executed when a session starts up.

RStudio Server Pro

As we've already seen, however, RStudio Server Pro launches sessions a little differently.

1. Prior to launching the process, it sets environment variables appropriate to the R version. This is what makes it possible for it to support multiple R installations simultaneously.
2. Instead of launching the `rsession` process directly, it uses the `rserver-launcher` process, which is responsible for doing PAM session management (documented in the admin guide).
3. The sessions are launched using an `rsession-run` script. This script runs `/etc/rstudio/rsession-profile`, then uses `bash` to launch the session, with the result that startup actions performed by `bash` will be run prior to starting the session.
4. Finally, the session itself starts, with all of the R Startup behavior described above.

Command line R

One of the first questions to ask when experiencing trouble in RStudio is whether the same problem exists when running R at the command line. This helps to ascertain whether the problem actually has anything to do with RStudio, which it often does not.

R at the command line follows the R documentation as regards to startup, but it's important to note that typing R in your console actually invokes a script – try `which R` to see where the script lives. This script is responsible for setting variables such as `R_HOME`. RStudio does *not* invoke this script, so customizations to the script, which are often performed by distributions such as MRO, are not reflected in RStudio.

Suspend and Resume

State Folders

RStudio has three main levels of state storage: per-user, per-project, and per-session.

Type	Example
Per-User	Preferred color scheme and pane layout.
Per-Project	Indentation and tab settings.
Per-Session	Open files, environment contents.

There are two folders used by RStudio to save this state. One is `.rstudio`, which is stored in the user's home directory, and the other is `.Rproj.user`, which is stored in the top level of a project.

Both store session-specific data, but `.rstudio` serves two purposes: first, it stores all the data that is user-specific, but not project-specific, such as global settings. Second, it doubles as a `.Rproj.user` folder when there is no specific project open. For this reason, many of the folders that exist in `.rstudio` also exist in `.RProj.user`., but the reverse is not true.

.rstudio

File/Folder	Contents
`client-state/`	The state of the IDE's user interface, such as the current size of the panels.
`history_database`	History of R console commands and when they were executed.

File/Folder	Contents
`projects_settings/`	A list of unique IDs for every project, and which to load at startup.
`monitored/`	Global user preferences, saved settings, snippets, and MRU lists (recently o
`rversion-settings/`	Default R versions for the user and each of their projects.
`unsaved-notebooks/`	Cached output for R Markdown notebooks that haven't been saved.
`notebooks/`	Cached output for R Markdown notebooks that have been saved.

Again, `.rstudio` is also a project folder, so it also contains all the files and folders described in `.Rproj.user` below.

.Rproj.user

Because there is only one `.Rproj.user` folder but potentially several different users for the project, each user has their own folder beneath `.Rproj.user`. The folder is named with the user's ID, which can be found as `contextIdentifier="XXXXXXXX"` in `.rstudio/monitored/user-settings`. For this reason, removing the `.rstudio` folder has the side effect of losing all your state in each project – not because that state is actually stored in `.rstudio`, but because the key used to look it up is stored there.

Each user's folder inside `.Rproj.user` contains the following contents:

File/Folder	Contents
`console`	The scrollback buffer, and state, of each open terminal.
`saved_source_makers`	Source markers (lint, syntax errors, etc.) for open files.
`sessions/`	Session-specific data.
`sessions/graphics/`	Plots, past and present.
`sessions/active/`	List of active sessions and session-specific data for each.

File/Folder	Contents
`sessions/session-routes/`	Specifies the RSP instance associated with a session.
`session-persistent-state`	Which browser is connected to the session, and whether it cra
`pcs/`	The state of the IDE's user interface.
`sources/`	Source database: contents of open files. Was called `sdb` prior
`sources/per/t/`	Contents of files with filenames ("titled").
`sources/per/u/`	Contents of files that have never been saved ("untitled").
`sources/prop/`	Properties/preferences associated with files.
`viewer-cache/`	The data for any open data viewer tabs.
`viewer_history`	A list of past locations (URLs) for the Viewer tab.

There are also a handful of files in `.Rproj.user` which aren't user-specific. These are stored directly beneath `.Rproj.user` in a folder called `shared`.

File/Folder	Contents
`notebooks/paths`	A list of unique IDs for each notebook in the project.
`notebooks/`	Per-user notebook data, such as saved notebook output.
`users`	A list of all the users who currently have the project open