PRO TIPS

from RStudio Customer Success

RStudio

Creating Efficient Workflows with `pins`

When your Workflow is Clunky

Like that one piece of furniture in your living room that you *need*, but you just can't find a spot for, we often struggle to find a home for the mid-process artifacts of a data analysis.

Think about some of your workflows. Are you:

- Using `read.csv()` to bring in data that is emailed to you after every update?
- Saving `.Rds` or `.RData` objects to be called later?
- Sharing your model or data across multiple apps?
- Redeploying your app every time the supporting data is updated?

A "yes" to any of the above indicates a solid use case for `pins`! Here at RStudio, we developed `pins` to make discovering, caching, and sharing resources simpler, all to promote efficient data workflows.

What are Pins?

Just like you'd pin a note (or a photo of your BFF) to a physical cork board, `pins` lets you pin an R or Python object to virtual board where you and others access it. The virtual board can be on RStudio Connect, S3, Google Cloud, or even your website, to name a few.

Pins are best suited for objects around a few hundred megabytes. Often they're made of lightweight or ephemeral data, and they may be relied upon by multiple assets.

Once pinned, it's much easier to share internal datasets across multiple assets or across your team. If you're using RStudio Connect, you can be sure that only the right people can access your pins with the RStudio Connect access controls you're already used to.

In this Pro Tip, you'll learn how to pin and retrieve a resource on RStudio Connect and how to schedule updates to pins so downstream analyses always stay current without redeployment.

For a follow-along accompaniment to this Pro Tip with real data see: https://github.com/rstudio/CS_protips/blob/master/example_pins/

Requirements

To utilize `pins` with Connect make sure you:

1. Are a Publisher or Administrator on your Connect server.[1]
2. Have RStudio Connect v 1.7.8 or higher.[2]

Prework

API keys will let the RStudio IDE communicate with Connect on our behalf, acting as our credentials. The steps below will save your credentials for future work, so these are one-time only steps:

1. Create an API key from RStudio Connect[3] Give this key any name you like, such as `CONNECT_API_KEY` and be certain to copy the value to your clipboard.
2. Return to the RStudio IDE and save your API key as a system environment variable in your .Rprofile file:
   a. In the Console, enter `usethis::edit_r_profile()` to open your .Rprofile for editing
   b. In the .RProfile, insert `Sys.setenv( "CONNECT_API_KEY" = "paste key value")`
3. For convenience, save your RStudio Connect server address as a system environment variable in your .RProfile as well. Example: `Sys.setenv("CONNECT_SERVER" = "https://your-server-address.com/")`
4. Save and close the file
5. Restart R (shift + cmd + F10)

If you're using git, it's a good idea to add your .RProfile to your .gitignore, perhaps with `usethis::use_git_ignore()`, to ensure you're not publishing your API key to your version control system.

Install `pins`. The `pins` package is available on CRAN, and was at version 0.4.0 at the time of this writing. Install `pins` with `install.packages("pins")`.

Now let's get cooking!

[1] Viewers can only retrieve pins

[2] If you're not yet at this version but are keen to get started with `pins`, talk to your R Admin and refer to https://docs.rstudio.com/connect/admin/server-management/#upgrading

[3] See: https://docs.rstudio.com/connect/user/api-keys/

PRO TIPS

from RStudio Customer Success

RStudio

Pinning to RStudio Connect

The first step in using pins is to "Register" the board. This just means you're identifying a location where you can store resources.

Register Connect as the board with the following:

```
#Register RStudio Connect
pins::board_register(
    "rsconnect",
    server = Sys.getenv("CONNECT_SERVER"),
    key = Sys.getenv("CONNECT_API_KEY"))
```

Now it's time to pin your first resource. Select an object and **pin to the rsconnect board with:**

```
my_data <- iris #for example
#pin object
pins::pin(my_data,
    name = "my_data",
    description = "A lovely pin!",
    board = "rsconnect")
```

The `board` argument tells the `pins` package that the destination is RStudio Connect. You'd change this argument if you were bound for other destinations.

With those two commands, you've created your first pin! Congratulations! Be sure to adjust the **Access Settings** on your pin in the RStudio Connect content dashboard if you want to share this pin with others.

Retrieving a Pin From Connect

If you head over to Connect, you'll notice your pin has some useful header information for pin retrieval. So let's copy that code into our analysis and **retrieve the pin**.

Replace the `# Retrieve Pin` code section below with the sample from your own pin:

```
# Register RStudio Connect
pins::board_register("rsconnect",
                key = Sys.getenv("CONNECT_API_KEY"),
                server = Sys.getenv("CONNECT_SERVER"))

# Retrieve Pin
my_data <- pin_get("your_username/my_data",
                   board = "rsconnect")

# View the pin locally
head(my_data)
```

Schedule Updates to your Pin

Up to this point, you've found a home for your mid-process artifacts and learned how to share them as pins. Now it's time to put your pins on an update schedule and bask in glory as your analyses automatically refer to the most current data without requiring redeployment.

To do this, create an R Markdown document that pulls your data, does any needed processing, and then creates your pin on RStudio Connect. This will be a supporting ETL (extract, transform, and load) file in your pipeline. Publish this R Markdown document to Connect. Then, – *whoa, wait, wha?!…*

Were you greeted with a failure to deploy and/or very angry looking error message when you tried to publish your ETL? This is because the environment variables in your code (`CONNECT_API_KEY` and `CONNECT_SERVER`) are not saved on the Connect server; they're only sitting in your .Rprofile in the RStudio IDE.

Don't worry, it's easy to address: In the RStudio Connect content settings panel for your R Markdown ETL, select the **Vars** option and **create environment variables for `CONNECT_API_KEY` and `CONNECT_SERVER`** - values will appear in plain text until you hit **Save**.

Click the **Refresh Report** button (*the button!* don't just refresh your browser window) to re-render your document with the new environment variables in place. Then, go ahead… do a little dance to celebrate! You've published an ETL document that publishes a pin to Connect!

To finish this little gem, click the **Schedule** button and establish a schedule for your ETL (and resulting pin) to refresh. Now you can point your customer-facing data analysis at this pin to always have a fresh source of data behind it. Dazzling!

Where Do I Go From Here?

At this point, you know what a pin is, whether pins will be useful for your workflow, and how to implement them. What next?

Go try `pins` on your own!

- If you want more hands-on advice with data you can play with yourself, work through our detailed pinning example here: https://github.com/rstudio/CS_protips/blob/master/example_pins/example_pins.Rmd
- Looking for inspiration? See this content collection that uses a pinned model and datasets as part of a pipeline to support a Shiny app in Production. The underlying data in the pin is refreshed on a schedule, keeping the Shiny app current: https://solutions.rstudio.com/tour/bike_predict/.
- We also have a ton of resources available for pins right at the `pins` website here: http://pins.rstudio.com/.
- Any issues? Let us know here: https://github.com/rstudio/pins/issues.

Last but not least, let us know how you get on with pins! Reach out to your Customer Success Representative, or send a note to us at sales@rstudio.com.