PRO TIPS

from RStudio Customer Success

RStudio

Creating Efficient Workflows with `pins`

When your Workflow is Clunky

A finished analysis or a data product in production often requires supporting artifacts. Whether those artifacts are data, a model, a plot, or any other object, the way we integrate those artifacts into our workflow can make the difference between a streamlined process and one that is clunky. Like that one piece of furniture in your living room that you *need*, but you just can't find a good place to stick it, sometimes we struggle to find a logical home for these artifacts.

Think about some of your workflows. Are you:

- Using `read.csv()` to bring in data that is emailed to you after every update?
- Saving `.R` 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?

If you answered yes to any of the above, you've got a solid use case for `pins`! Here at RStudio, we developed `pins` to make the process of discovering, caching, and sharing resources simpler, all with the ultimate aim of helping you create efficient data workflows.

What are `pins`?

`pins` are pretty much what they sound like: just like you'd pin a note (or a photo of your BFF) to a physical cork board, pin an R or Python object to virtual board so that you and others can share and access that object. The cork board can be on RStudio Connect, S3, Google Cloud, or even your website, to name a few.

Got an idea for an asset that might make a great `pin`? `pins` are best suited for objects that are just a few hundred megabytes in size. Often they are made of lightweight or ephemeral data, and they may be relied upon by multiple assets.

Once you've pinned a resource, other folks can discover them. This is great for two reasons: First, `pins` on sites like Kaggle let you search well-known data repositories that are guaranteed to contain datasets, instead of searching on the internet and hoping to find what you need. Second, `pins` also enhance the discoverability of internally-created datasets with the same access controls you're accustomed to for any content published to Connect, which is great for sharing data and learning more about what your colleagues are up to.

In this Pro Tip, you'll learn how to `pin` your first resource to RStudio Connect, how to retrieve a `pin`, and how to deploy an asset to Connect that includes a `pin` as a supporting artifact.

For hands-on advice for each step in this process, follow along with a complete example with real data that accompanies this Pro Tip: https://github.com/rstudio/CS_protips/blob/master/example_pins/example_pins.Rmd

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

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] You can retrieve a `pin` with only a Viewer role

[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

PRO TIPS

from RStudio Customer Success

RStudio

1. Create an API key from RStudio Connect (See: https://docs.rstudio.com/connect/user/api-keys/) 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 file, insert `Sys.setenv("CONNECT_API_KEY" = "key value from your clipboard")`.
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)

Note, saving your key as a system environment variable is a best practice. This lets you call your key without explicitly hard coding the value into your scripts. It's also a good idea to put it in your .gitignore file (or equivalent) to ensure you're not publishing it to your version control platform.

Now we're ready to get cooking!

Walk me Through my First `pin`

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")`.

The first step in using `pins` is to "Register" the board. This means you're identifying a location where you can store resources. You're registering, or recognizing, the location of the virtual cork board within your current RStudio session.

Register RStudio Connect as the board with the following script:

```
#Register RStudio Connect
pins::board_register(
    "rsconnect",
    server = "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:**

```
#pin object
pins::pin(my_data,
    name = "my_data",
    description = "A lovely pin!",
    board = "rsconnect")
```

The `name` argument will be the name of the content when deployed on Connect. The `description` should help you and others know what the pin is. And the specific `board` argument of `rsconnect` 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 pinned your first asset. 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 RStudio 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/pin", board = "rsconnect")

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

Include `pins` in a Deployed Asset

`pins` shine when they are a part of a workflow or pipeline supporting an app, model, or report. The last step in `pin` mastery

PRO TIPS

from RStudio Customer Success

is to deploy an asset to RStudio Connect that uses a `pin`. Heads up: this section is less about `pins` and more about environment variables, but you'll be a master of both by the time we are through.

Use your new `pin` creating and retrieving skills to call a pinned object to your content of choice to deploy to Connect. Now try to deploy your content - don't be surprised if you got a failure to deploy and/or very angry looking error message. This is because the environment variables in your code (recall: `CONNECT_API_KEY` and `CONNECT_SERVER` from above) are not saved on the Connect server; they're only sitting in your .Rprofile in the RStudio IDE. The specific error and where it presents may vary depending on they type of asset you're trying to deploy (R Markdown, Shiny, plumber API, etc.), but the punchline is the same.

Regardless of where and how your error presents, it's easy to address: In the RStudio Connect content settings panel for the asset you are trying to deploy, 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**.

That environment variable that you specify doesn't have to be the same API key that was used when the pin was created - for example, User A might create the pin, and User B might pull it into a Shiny app. API keys are different per user, and are just a way of telling Connect that you're allowed to `pin` resources.

Click the **Refresh Report** button 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 asset that calls a pin!

A Special Tip for R Markdown Deployments

If you're trying to deploy a R Markdown document that calls a pin (or more generally, uses environment variables), make life a little friendlier for future you and any colleague that might be running your code. **Insert the following code at the top of any R Markdown document that includes environment variables that will be deployed to Connect:**

```
# Check for system environment variables
knitr::opts_chunk$set(echo = TRUE)
if(Sys.getenv('CONNECT_SERVER') == '')
{ "<h4>ERROR: You must set the
  CONNECT_SERVER environment variable</h4>\n" }
if(Sys.getenv('CONNECT_API_KEY') == '')
{ "<h4>ERROR: You must set the
  CONNECT_API_KEY environment variable</h4>\n" }
if(Sys.getenv('CONNECT_API_KEY') == '' ||
   Sys.getenv('CONNECT_SERVER') == '')
  { knitr::knit_exit() }
```

The code above will tell `knitr` to stop at that point in the rendering and exit the knitting process if these environment variables don't exist. By placing this code at the top of your document, you'll have a friendly reminder to set your missing environment variables rather than a red and angry failed deployment message.

Where do I go From Here?

At this point, you know what a `pin` is, whether `pins` will be useful for your workflow, and all the basic steps for working with `pins`. 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, *no re-deployments necessary.*
- We also have a ton of resources available for `pins` right at the `pins` website here: http://pins.rstudio.com/.
- Check out the source code here https://github.com/rstudio/pins.
- 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.