PRO TIPS

from RStudio Customer Success

Creating Efficient Workflows with `pins`

When your workflow is clunky

Creating and sustaining 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 with bumps or stall outs along the road. 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 the our supporting objects just don't have a logical home.

Think about one of your workflows:

- Are you using `read.csv()` to bring in a data file that your colleague emails to you after every update?
- Do you work with ephemeral data that frequently has to be updated to use the latest version?
- Do you need to redeploy 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: they're a way to store and then remotely access R and Python objects, just like you'd pin a note (or a photo of your BFF) to a (virtual) cork board. 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`? Typically,`pins` are just a few hundred megabytes in size, lightweight or ephemeral data, or are for things relied on by multiple assets.

Once you've `pin`ned 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. Secondly, `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 the next section, you'll learn how to `pin` your first resource to RStudio Connect.

Requirements

To `pin` to Connect, make sure you match these requirements:

1. Be a Publisher or Administrator role on your Connect server

2. Be running RStudio Connect v 1.7.8 or higher.[1]

Walk me through my first `pin`

You can think of this process in four main steps: First, there is some prep work to ensure the `pin`ning process goes as smoothly as possible. Second, you get your R session to connect with your board. Third, you `pin` a resource. Finally, you can access resources that you or others have `pin`ned.

Let's walk through each step using RStudio Connect as an example.

[1] 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 if you are at an earlier version

PRO TIPS

from RStudio Customer Success

Prework

Before you can get down to business, you'll need to:

1. Create an API key from RStudio Connect.[2] Give this key any name you like, such as `CONNECT_API_KEY` and be certain to copy the value to your clipboard.
2. Save your API key as a system environment variable in your RStudio IDE in your .Renviron file.[3] Example: `Sys.setenv("CONNECT_API_KEY" = "key value that you copied to your clipboard in Step 1")` 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 to ensure you're not publishing it to GitHub.
3. For convenience, save your RStudio Connect server address as a system environment variable as well. Example: `Sys.setenv("CONNECT_SERVER" = "https://your-server-address.com/")`
4. Tell knittr to exit the knitting process if these environment variables don't exist by placing this code at the top of your document. This will save you from getting scary error messages that will stop your content from deploying or rendering altogether. It's not necessary, but it will be helpful later on. `{ r warning} 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() }}`
5. Install the `pins` package. Example: `install.packages("pins")`

That's all you need to get started!

[2]It's not hard. See: https://docs.rstudio.com/connect/user/api-keys/
[3]API keys are unique means of identifying you as a user to a system – in this case when we use this key, it is to tell Connect who you are so you have permission to publish

Register the Board

The first step is to "Register" the board. Don't let the vocabulary throw you. This just means you're identifying a location where you can store resources. You're registering, or recognizing, the board location within your current RStudio session.

Add the following to your script:

```r
pins::board_register(
    "rsconnect",
    server = "CONNECT_SERVER",
    key = Sys.getenv("CONNECT_API_KEY")
    )
```

Pin your Object

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

```r
pin(my_data, #your object
    name = "my_data",
    description = "Super cool data set",
    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.

Retriving a pin

And now for the fourth step, you can retrieve a `pin`.

If you head on over to Connect, you'll notice your `pin` has some useful header information. You can copy that code into your analysis, then assign the retrieved `pin` as a variable.

```r
# Register RStudio Connect
library(pins)
board_register("rsconnect",
 server = "https://yourconnectserver.example.com")
# Retrieve Pin
your_data <- pin_get("yourdata",
                     board = "rsconnect")
#Check out the pin
head(your_data)
```

The first time you publish content that calls a `pin`, bear in mind that your environmnet variables will not be set on the Connect server. If you used the Friendly Fail message above, your content will deploy and you'll just see the custom friendly error message saying "You must set the CONNECT_API_KEY environment variable." If you don't use the Friendly Fail, you will get a message saying "[Connect] Message: 'Invalid API key, the API key is empty.' "

Regardless, it's easy to take the next necessary step: put CONNECT_API_KEY in environment variables in Connect and refresh.

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

You have now `pin`ned and retrieved a `pin`!

Where do I go for help?

At this point, you know what a `pin` is, whether `pins` will be useful for your workflow, and how to get started with your first pin. What next?

If you want more hands-on advice with data you can play with yourself, we have a downloadable example that accompanies this Pro Tip with real data, see: https://github.com/rstudio/CS_protips/blob/master/example_pins/example_pins.Rmd

Go try `pins` on your own! 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! Get in touch with the Customer Success team at sales@rstudio.com.