Workshop Overview - DO NOT DELETE

Title: Data Science Platform Administration with RStudio Team

Who: Alex Gold

Abstract: In this workshop, you'll learn to use the capabilities of RStudio Team to enable your organization's R and Python users, including topics like package and environment management, performance and scaling, external data connections, and integrating RStudio Team with CI/CD pipelines.

RStudio

Wifi connection details

Network: conf22

Password: together!

RStudio

Remember there's another copy of this slide for stragglers around slide 14.

9:00-10:30
Next up:
Intro - Alex
Getting a server - Lisa
Deployment Architecture - Sam E

Logistics

Gender Neutral Bathrooms: By the National Harbor Rooms
Meditation/Prayer Room: National Harbor 9, 8-5 M-Th
Lactation Room: Potomac Dressing Room; 8-5 M-Th
Hotel also has dedicated room behind reception.

Please respect buttons!
Red lanyards = please do not photograph

RStudio

- Announce anything that the resort requires us to
- Also highlight emergency exits
- And anything else that seems important for the safety of attendees.

Logistics, continued…

Download the conference app – RStudio Conf

RStudio

Food

Breakfast 7:30-9

Lunch 12:30-1:30 Mon/Tues

RStudio

COVID

RStudio requires that you wear a mask that fully covers your mouth and nose at all times in all public spaces. We strongly recommend that you use a correctly fitted N95, KN95, or similar particulate filtering mask; we will have a limited supply available upon request.

RStudio

Photo by Gayatri Malhotra on Unsplash

Code of Conduct

Everyone who comes to learn and enjoy the experience should feel welcome at rstudio::conf. RStudio is committed to providing a professional, friendly and safe environment for all participants at its events, regardless of gender, sexual orientation, disability, race, ethnicity, religion, national origin or other protected class.

This code of conduct outlines the expectations for all participants, including attendees, sponsors, speakers, vendors, media, exhibitors, and volunteers. RStudio will actively enforce this code of conduct throughout rstudio::conf.

conf@rstudio.com 844-448-1212

RStudio

https://www.rstudio.com/conference/2022/2022-conf-code-of-conduct/

RStudio

Introduce yourself - Who will be our presenters and TAs - introduce them to the room. Also take some time to introduce **Solutions Engineering** and explain what we do

Introduce yourself to your neighbours!

Howdy neighbour!

RStudio

Look left, look right, introduce yourself to your neighbours.

Introducing the workshop

RStudio

What you'll learn

How to use RStudio's Pro Products to administer data science at scale.

RStudio

We'll be talking about R and RStudio's pro products, maybe with a little Python thrown in here and there for good measure.
We'll cover installing and managing extension packages, the problems folks face when they're writing or maintaining environments running production R code.
Later on we'll talk about some of the "how and why"s of the way that R works and the way RStudio's pro products have been designed to help your business overcome these challenges.
Ultimately we want you to go away from this workshop with a better understanding of the tools, your users and more confidently able to support these tools in your own enterprise settings.

Today: Managing Environments

- R
- R Packages
- System Dependencies
- RStudio Package Manager
- RStudio Package Manager + RStudio Connect

RStudio

I'm not going to read all that, but it's from the conference web page that describes the workshop.
We'll be talking about R and RStudio's pro products, maybe with a little Python thrown in here and there for good measure.
We'll cover installing and managing extension packages, the problems folks face when they're writing or maintaining environments running production R code.
Later on we'll talk about some of the "how and why"s of the way that R works and the way RStudio's pro products have been designed to help your business overcome these challenges.
Ultimately we want you to go away from this workshop with a better understanding of the tools, your users and more confidently able to support these tools in your own enterprise settings.

Tomorrow: Production

- RStudio Connect Publishing
- Data Connections
- Scaling RStudio Products
- CI/CD + Robust Deployments

RStudio

We'll be talking about R and RStudio's pro products, maybe with a little Python thrown in here and there for good measure.
We'll cover installing and managing extension packages, the problems folks face when they're writing or maintaining environments running production R code.
Later on we'll talk about some of the "how and why"s of the way that R works and the way RStudio's pro products have been designed to help your business overcome these challenges.
Ultimately we want you to go away from this workshop with a better understanding of the tools, your users and more confidently able to support these tools in your own enterprise settings.

General Schedule

Time	What
09:00 - 10:30	👨‍🏭 Work
10:30 - 11:00	☕ Coffee
11:00 - 12:30	⚙️ Work
12:30 - 13:30	🍱 Lunch
13:30 - 15:00	🏭 Work
15:00 - 15:30	☕ Coffee
15:30 - 17:00	👷 Work

RStudio

Stickies

I'm lost/need help

I'm done and ready to move along

Put them up on the back of your laptop screen.

RStudio

What is R ?

A **free/open source** software environment for **statistical computing and graphics**.

Created in 1995 by statisticians.

Successor to S.

Controlled by R Core + R Foundation.

RStudio

What is RStudio ?

RStudio was founded by JJ Allaire in 2009.

Initial focus was on delivering a new IDE for R, but soon was complemented by the release of Shiny, RMarkdown, …

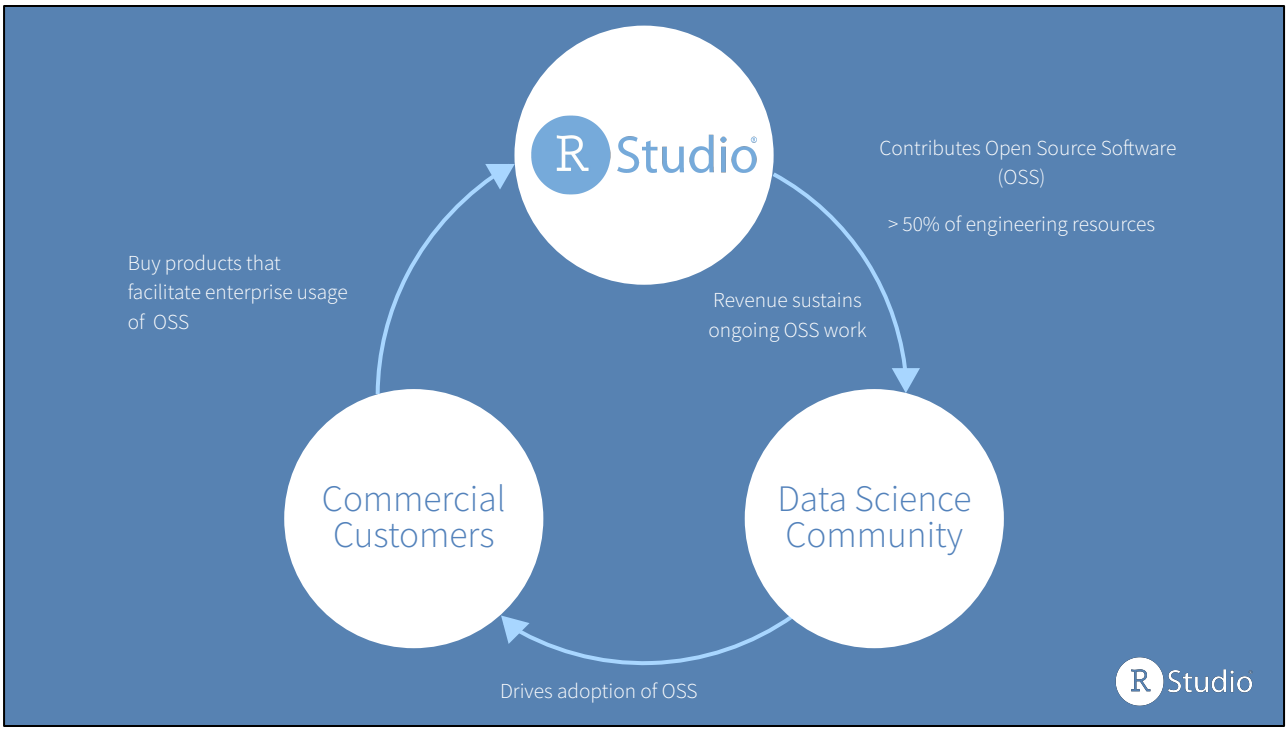
Commercial products (RSW, RSC, RSPM, ShinyApps.io, RStudio.cloud) - but all of them have free versions/plans available, too (now and in the future).

50 % of the revenue is spent on open source development

2020 PBC (Public Benefit Corporation)

Certified B Corporation

RStudio



For more info:

- Certified B Corp Annual Report: https://rstudio.com/about/pbc-report/
- What Makes RStudio Different: https://rstudio.com/about/what-makes-rstudio-different/
- Open source projects: https://rstudio.com/about/rstudio-open-source-packages/

(Engineering comprises ~50% of RStudio organization)

First task: get on Discord

1. Go to Discord server (email w/ invite sent last week).
2. Click ✅ once you've read Code of Conduct.
3. You'll have access to ALL workshop channels.
4. Go to #💻-ds-admin and add a comment with your favorite donut 🍩 flavor.

Use the stickies!
discord.gg/FRxNvG7Kp9

RStudio

Getting to know your workshop environment

- Wifi information
- Accessing the environment
- Logging in
- Looking around and familiarise yourself with the tools.

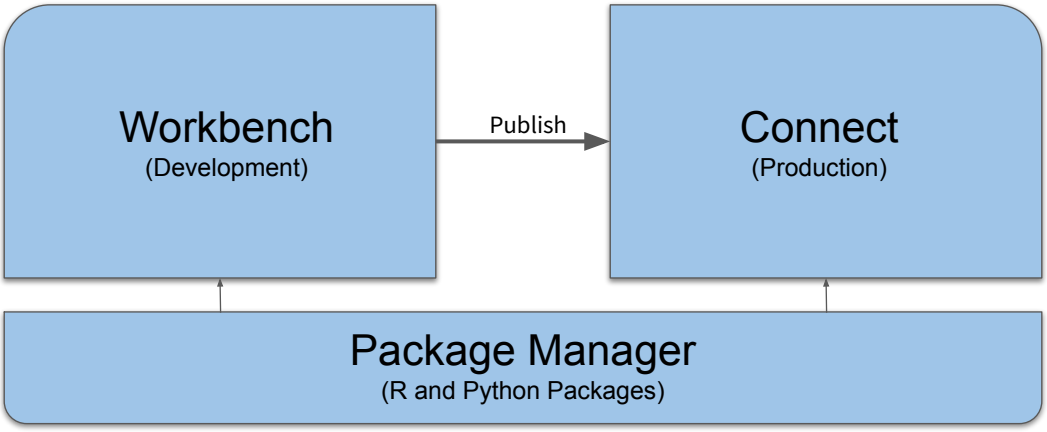
RStudio

Lisa

Your workshop environment

Lisa

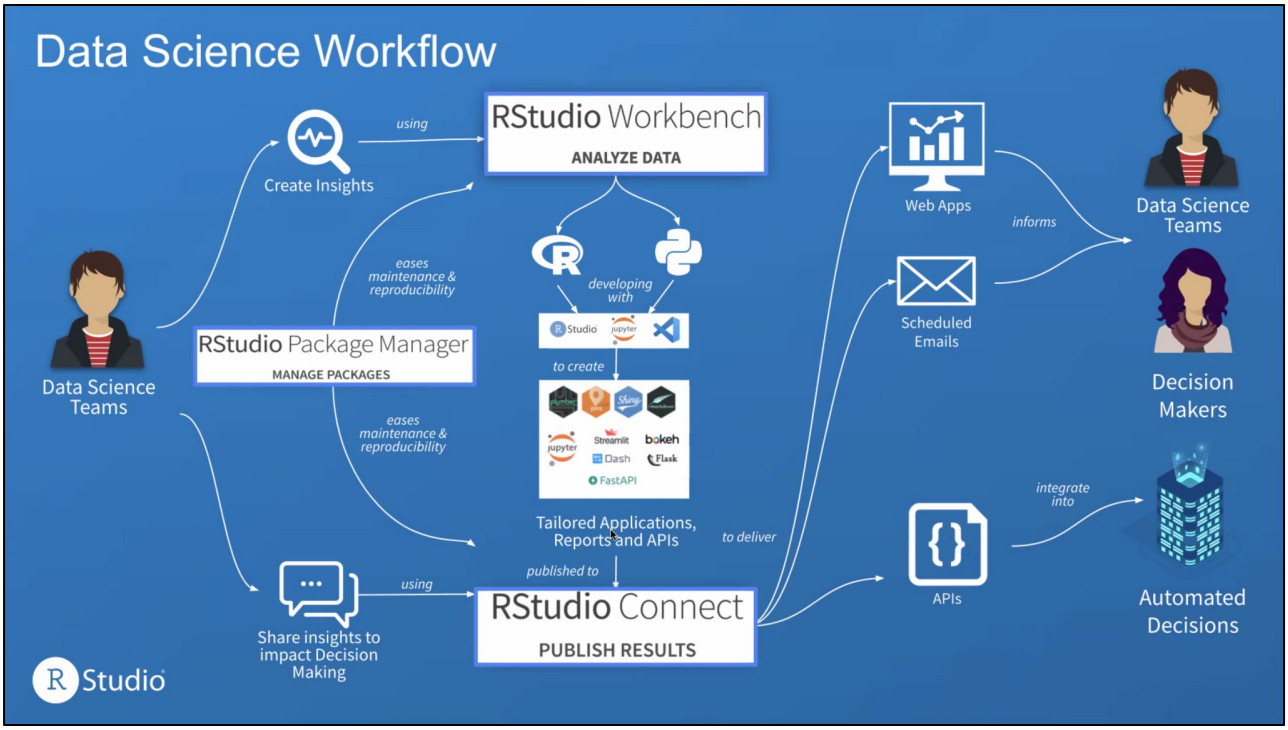
RStudio's Pro Products



RStudio

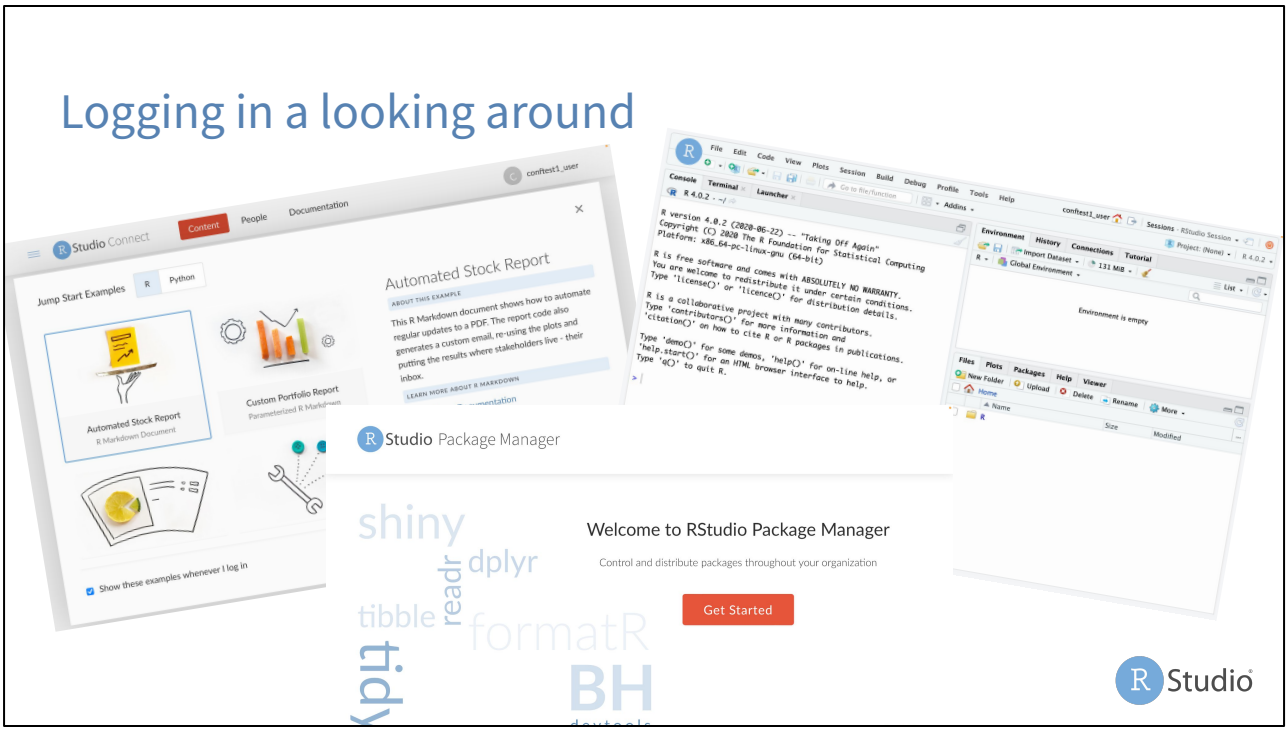
At a very high level…
What is each and what does it do?

- **Workbench** is your development environment - It's where your users will write their code - both data science and data product
- **Connect** is your publishing platform where your long lived assets like data-centric applications, APIs, scheduled reports and so on will live. For publishing rather than development
- **Package Manager** supports the other products by providing an on-prem one stop shop from which add-on packages can be obtained.



Talk through what's on the slide.

- Who is the audience for each product?
- Workbench is for developers/data scientists
- Connect is primarily for non-technical consumers of data products written by developers
    - As well as interactive "apps" Connect can also host scheduled content and can be configured to also send emails with the output (push rather than pull)
    - Can also host APIs, so it's also good for machine-to-machine comms
- Package Manager's main consumer is "code", it provides language dependencies (packages) into the code execution environment, whether that's interactively in RSW or non-interactively in RSC



Instructor will also do this live and show people around

Link: https://rstd.io/class
Current ID: conftest2

Log in to Workbench and Connect and have a look around.
Have a look at the RSPM landing page.

Server: t3.xlarge
Best practice deploy on separate servers (so each server can be optimized, etc) but for simplicity deploying on one.

We need to see:

- Claiming server process
- Workbench
  - Log in
  - Landing page
  - RStudio IDE
  - Point out that in a full installation, other IDEs like Jupyter lab/notebooks and VSCode are available, but since we're focussing on R right now, they're omitted here.
- Connect
  - Log in

    - Landing page - with a note to say that it fills up with content as things get deployed to it.
- RSPM
    - Just the default landing page for now - we'll come back and finish setting that up later.

Wifi connection and workshop servers

Network: conf22

Password: together!

discord.gg/rstudioconf2022

Visit: https://rstd.io/class

Workshop Identifier: ds-admin

RStudio

Make sure everyone understands how to get connected to the WiFi and then heads over the the URL above to claim their server.
ID for workshop: ds-admin

Explain what the server is and what it will be used for.

Remind people that it will exist only for the duration of the workshop, after which it will be decommissioned.

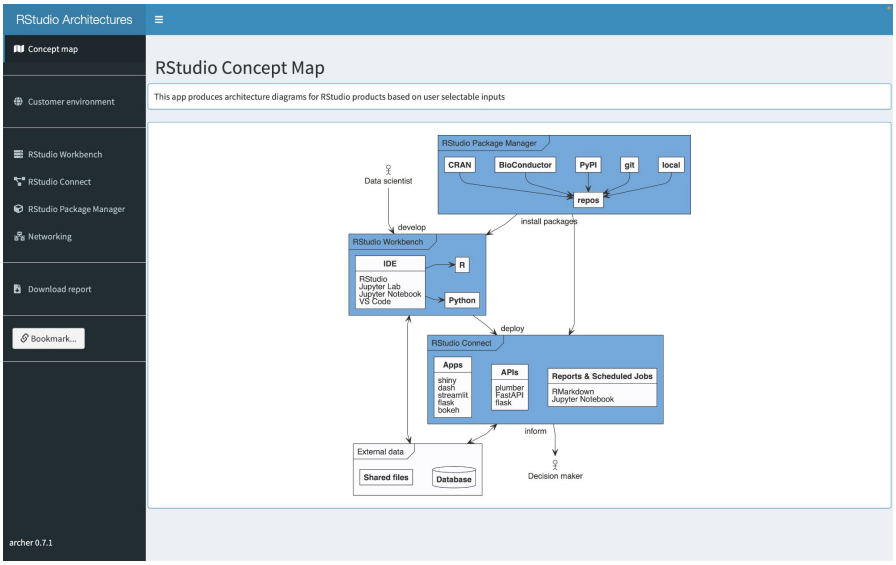
Once you see the landing page go ahead and put a green sticky up. If you want to log in to workbench and connect to explore go ahead. If you run into any issues red sticky.

Drop links into the discord chat

RStudio Deployment Architectures

Runs until 10:30
Sam E

RStudio Deployment Architectures



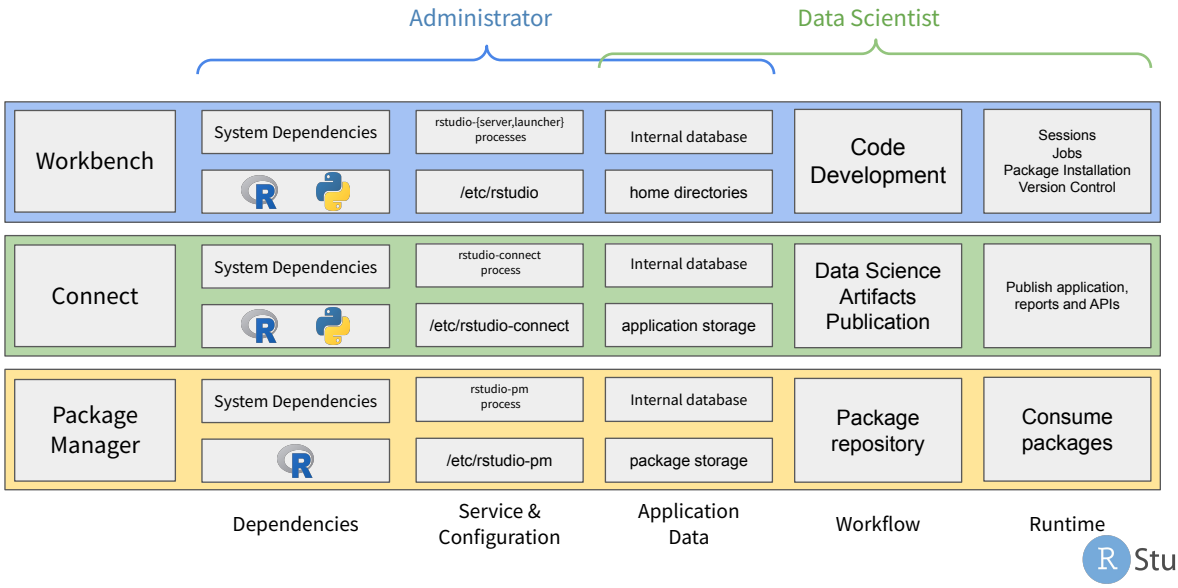
Introduce the archer app and explain the process. https://connect.rstudioservices.com/target-architecture/. Then go through at least one potential architecture call.

Discuss:

- Introduce the architecture call - When do we do them? What are they for?
- Go through the architecture call process asking the attendees to shout out options from their own environments
- Discuss some of the options
- Basically a big multi-user architecture call
- Wide variety of disparate architectures
- Pro & cons of particular architectures
- Specifics around things like OS choice (eg. why is RHEL 7 not the best choice?)
- Cloud vs on-prem
- Launcher
- Do at least one simple architecture and one complex one
- Could we end this with a test of some sort?

10:30 - 10:00
Next speaker: Michael Mayer - R & R package installation

Photo by Nathan Dumlao on Unsplash


Administrator
Data Scientist
Workbench
System Dependencies
rstudio-{server,launcher} processes
Internal database
Code Development
Sessions
Jobs
Package Installation
Version Control
/etc/rstudio
home directories
Connect
System Dependencies
rstudio-connect process
Internal database
Data Science Artifacts Publication
Publish application, reports and APIs
/etc/rstudio-connect
application storage
Package Manager
System Dependencies
rstudio-pm process
Internal database
Package repository
Consume packages
/etc/rstudio-pm
package storage
Dependencies
Service & Configuration
Application Data
Workflow
Runtime
RStudio

R & R Package Installation

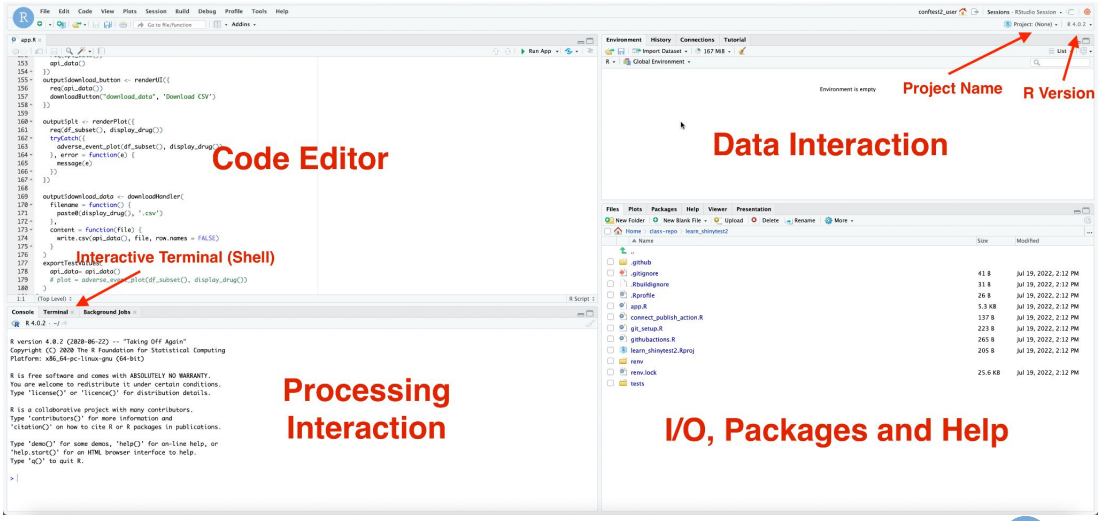
R's standard library contains the things you'd expect of a programming language, e.g. filesystem and environment operations, as well as tools for matrix math, statistics, and graphics.

For additional functionality, packages are installed from the Comprehensive R Archive Network (CRAN), a network of mirror sites that host R source code, extension packages, documentation,
and versions of the language.

CRAN is maintained by a small number of R community members, who perform manual reviews
of most submitted packages. You can [read more about the process packages go through before
they are accepted to CRAN here](https://r-pkgs.org/release.html). Importantly, CRAN submission does not constitute a security review

R users generally access that functionality via the `install.packages()` command.

IDE Introduction



RStudio®

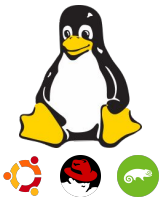
What is R ?

GPL v 2/3

A **free/open source** software environment

for **statistical computing and graphics**.

It compiles and runs on a wide variety of **UNIX platforms, Windows and MacOS**.



RStudio

The anatomy of a typical R installation

RStudio

Let's install the latest version of R

Following the instruction from https://docs.rstudio.com/resources/install-r/

1. Define the R version you would like to install (latest = 4.2.1)
2. Identify the operating system you would like to install R onto
3. Download R software
4. Install
5. Create Symbolic links if you want to make this R version available as default

Hands-On Exercise 1+2

Installing and loading a package

R commands are usually issued in the context of a running R session:

```
> install.packages('rlog')
```

NB: The “Packages” Pane in RStudio IDE can be used for easy package installation as well.

In the R session, or in an R script or app a user will use the “library()” function to load a package.

```
> library(rlog)
```

RStudio

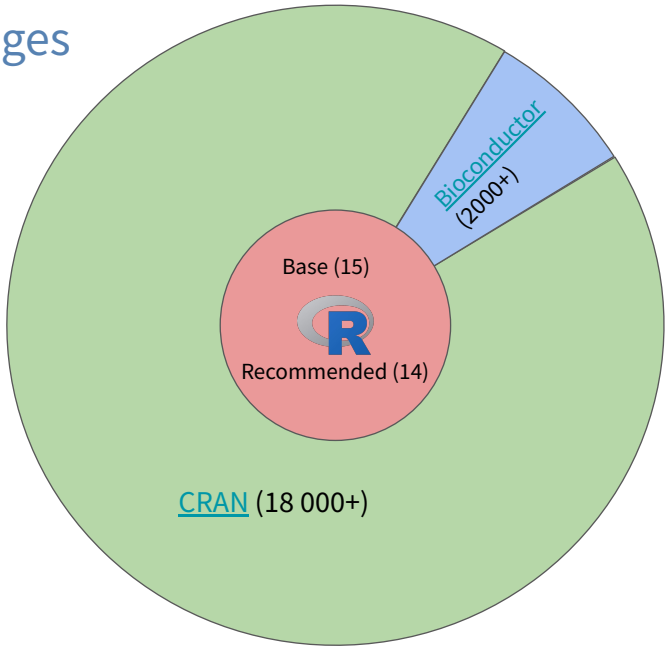
Notice the default prompt for an R session is the greater-than symbol, >

In the command line example, what’s in between the double-quotes matches what you’d run in the R console

Note that we’re using {rlog} here because it has no external dependencies and is written in pure R, so there’s no code to compile

EXERCISE Ask attendees to install the library, using the R console in the IDE

Where do packages come from ?



- GitHub
- R-Universe
- Self-hosted
- Others

RStudio

Where do packages go when they're installed?

In R, this is called a “library” and there are 3 important library types to be aware of:

- System (or “site”)
- User
- Project

Where the packages go depends in part on who executes the install

- Installation performed by **root** will install the packages to the **system** library.
    - Sometimes also called a “site” library
- Installation performed bv a user will install packages to a user library (in the user's’ home directory)

Often, the best way to organise your package library, is to have one per-project that the user is working on.

That way, each project’s dependencies will be isolated from one another.

This is most commonly done with the open-source package “renv”.

An important factor for the admin to remember, is that in R, the user has control over where they store packages.
The user library location is configurable by the end user, though it’s rare that a user would do so outside of the context of a project library

Photo by Devon Divine on Unsplash

.libPaths()

Usually two library paths are available, a system library path and a user library path

```
> .libPaths()
[1] "/home/conftest2_user/R/x86_64-pc-linux-gnu-library/4.2" "/opt/R/4.2.1/lib/R/library"
```

User library (R_LIBS_USER)

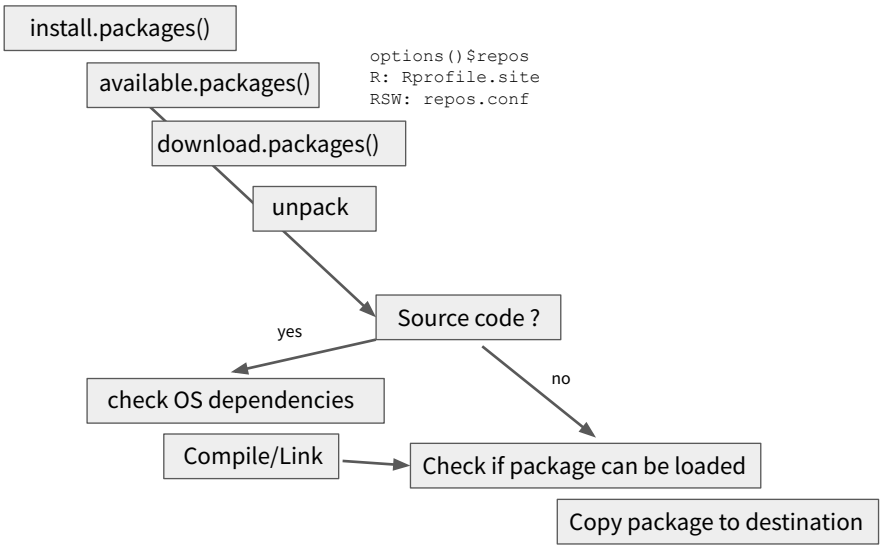
System/Site library (R_LIBS_SITE)

R will (by default)

- Install a package in to the first directory given by .libPaths() it can write to
- Load a package from the first directory it finds a package with that name

RStudio

High Level Process for Package Installation




```
options()$repos
R: Rprofile.site
RSW: repos.conf
```

Hands-On Exercise 3-5

```
.libPaths()
```

RStudio

Environment management

RStudio

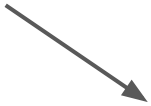
Runs until 12:30
Michael Mayer

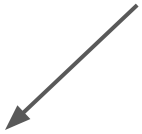
CRAN

- Packages added very frequently
- Unless package specified, compatible with all R versions
- Time-based snapshots provided via MRAN (and RSPM)

Bioconductor

- Formal Releases (2 versions per x.y release of R)
- Given BioC Release only compatible with single R version





Time-based snapshots of CRAN + Bioconductor = Win !

RStudio

Package Dependencies

There are two main types of package dependencies:

- Packages that depend on other packages
- Packages that depend on other software

This second kind are referred to as system dependencies.

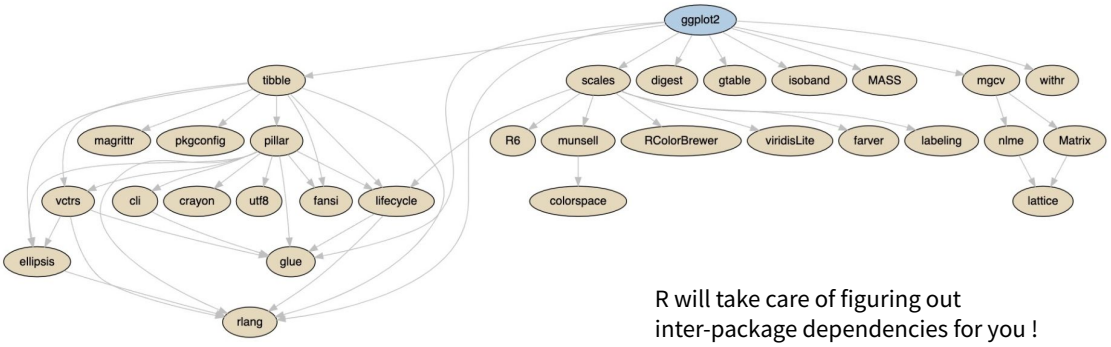
RStudio

Talk about how R handles regular dependencies

Contrast that with system dependencies where other software must first be installed to the OS before the package will run

Note that it's this second kind that trips admins up, because install.packages() handles packages that depend on packages, but system deps - because they must be installed to the OS - are the purview of whoever is responsible for system administration

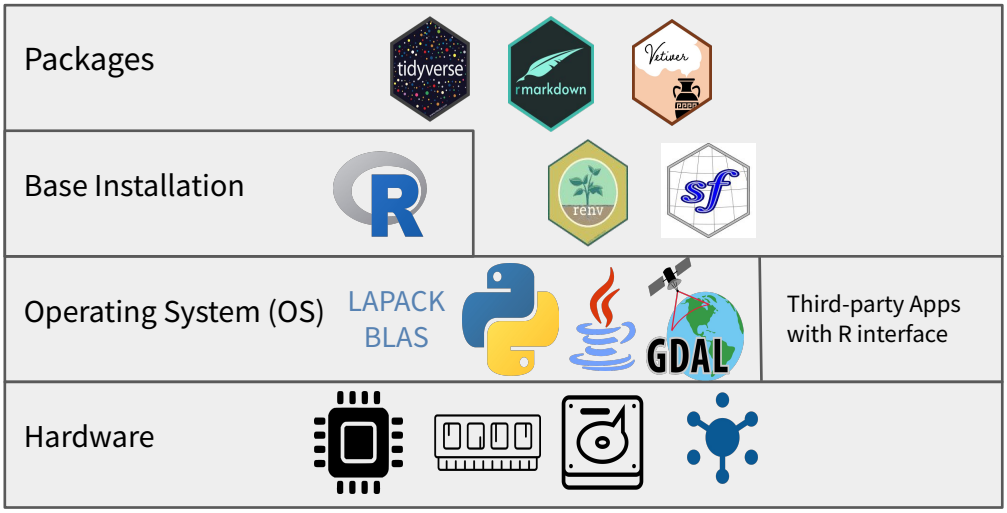
Inter-package dependencies



R will take care of figuring out
inter-package dependencies for you !

RStudio

Dependency Layers for R packages



RStudio

OS dependencies

Many R packages depend on other software provided by the operating system or third-party software providers.

If software (OS or third-party) is missing, package installation will fail until you install the dependencies

Hands-On Exercise 6

RStudio

OS package dependencies in Enterprise grade OS

Enterprise OS have a Stable API/ABI for 10+ years with each major release

- GNU Compiler version In RHEL7 is GCC 4.8.5, but the latest version available upstream is GCC 12.1
- System provided Python is python 3.6.8 while latest upstream is 3.10

Solutions:

- Use Software Collections when in RHEL/CentOS land
- Rely on parallel installed software versions (e.g. Python) when in Ubuntu land
- Use 3rd party software, e.g. (Mini)conda for Python
- Run your own application stack (Environment modules/EasyBuild/Spack)
- Use container solutions (e.g. Kubernetes, Apptainer/Singularity)
- Additional OS Software repositories

R libraries (repeat)

- System (or “site”)
- User
- Project

RStudio

System library

Pros

- Provides a base environment of working packages
- good for locked down and more static environments

Cons

- burden is on the Admin
- challenging to upgrade safely
- can result in deployment challenges during environment discovery
- difficult to scale

RStudio

Installing packages to the system library should be a last resort

RStudio recommend avoiding this approach if at all possible.

User libraries

Pros

- The user is responsible for the packages that they need
- The user can install and update packages as required

Cons

- Can be challenging to upgrade safely
- Can result in deployment challenges during environment discovery
- “Dependency Hell” - CRAN Snapshots may be useful.

RStudio

With great power comes great responsibility!

In this model, the user is free to install and upgrade at will.

However, they must maintain their own user library and ensure all the required code works properly with it.

Remember: upgrading a package for a new project, will also upgrade that package for the old project and if there are breaking changes in the package, the user will now have to update the old package to make it work again.

Project libraries

Pros

- isolation of the environment from other environments on your system
- enumeration of libraries used within the project, discovered from your code
- Simpler automation
- Can be as simple as defining an additional `.libPaths()`

Cons

- some cognitive overhead

RStudio

This one is RStudio's best practice recommendation
When package libraries are explicitly tied to a project the developer is responsible for for ensuring that the packages contained in that library work as expected with project

Pros:

- isolation of the environment from other environments on your system
  - Different projects all have their own libraries, so upgrade a package in one, does not break a different project
- enumeration of libraries used within the project, discovered from your code
  - It's very simple to understand what packages are used in a given project, which can be useful for code audits and so on later
- Simpler automation
  - If your project knows exactly what it's own library should contain, it's more straightforward to to bring it into a CI/CD workflow for testing or deployment while having your pipeline build out the library in a way which should just-work

Cons:

- some cognitive overhead
  - Isolating package libraries like this is something that R developers often come to later, so it might require some rethinking of existing workflows.

- It can also be hard to port these more formal techniques on to developers who are accustomed to a more free-wheeling approach to their development processes

renv

The `renv` package helps you create reproducible environments for your R projects.

- **Isolated** - Uses a project specific library to isolate from other projects with conflicting package versions
- **Portable & Reproducible** - Code together with the `renv` metadata (`renv.lock`)

Typical tasks

- `renv::init()` - will check existing source code, install any missing R packages and add all information to `renv.lock`
- `renv::restore()` - will restore packages according to existing `renv.lock`
- `renv::install()` - will install given package and update `renv.lock`
- `renv::snapshot()` - updates `renv.lock` with the currently used versions of R packages

Hands-on exercise 7

RStudio

A note about GitHub.com

RStudio

Stickies

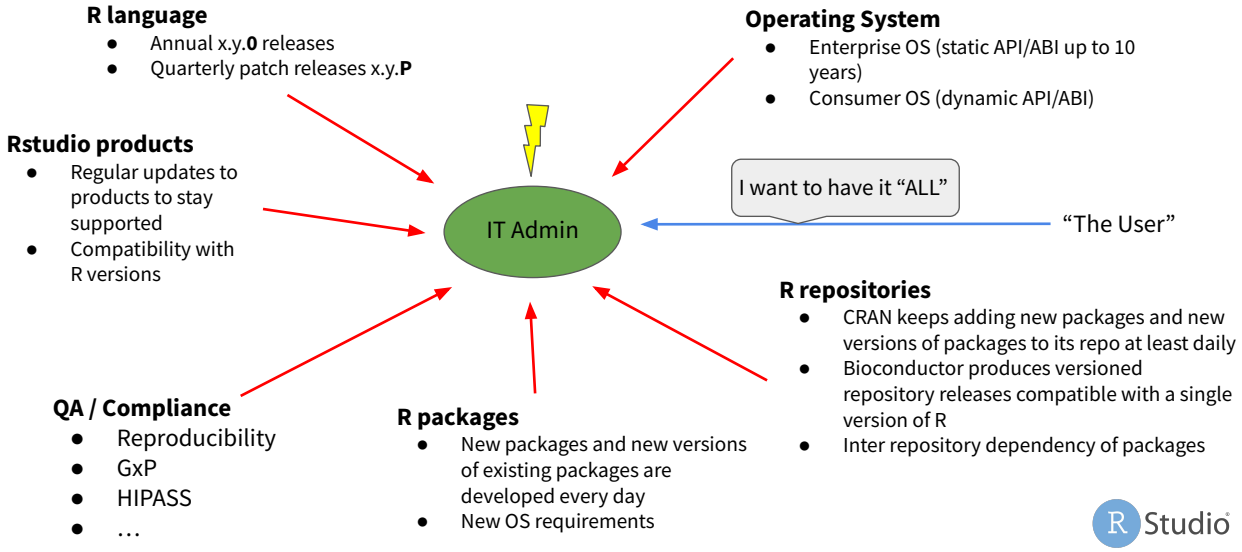
One thing that was boring/repetitive/too fast/unnecessary/unclear

Most useful thing to you from this morning

Back from lunch at 1:30

RStudio

Anatomy of an RStudio/R environment



RStudio

Package Manager

RStudio

Shannon
Runs until 15:00

How do your users
get their packages today?

How is that working for everyone?

RStudio

Think, Pair, Share - let's talk about what we do today, are there any places where packages cause problems on the admin side? On the user side?

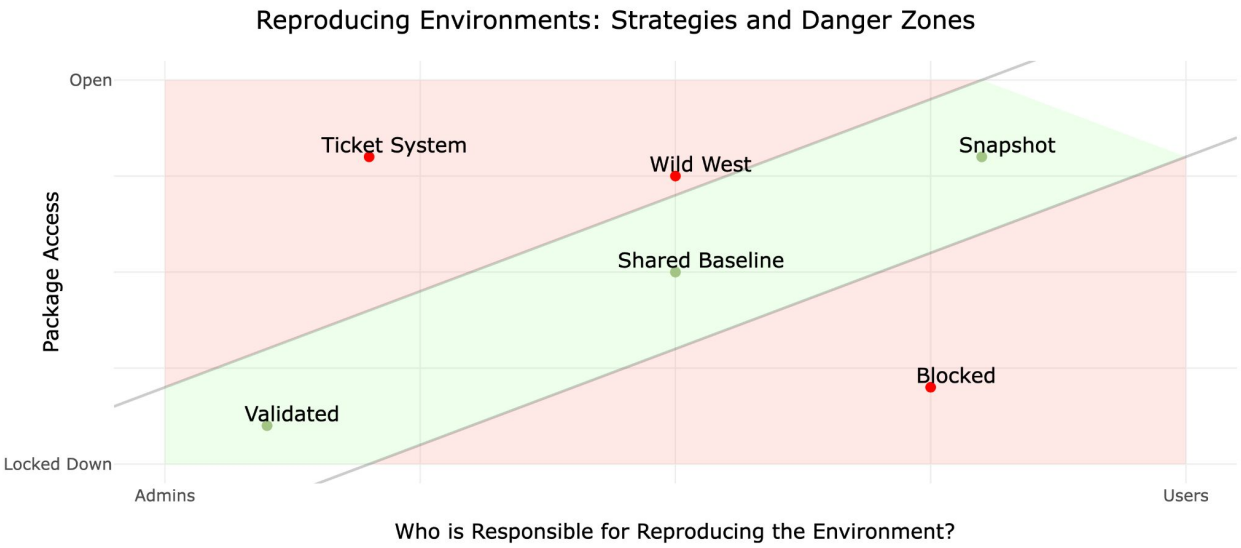
RStudio Package Manager

RSPM empowers users to access packages and reproduce environments while giving IT control and visibility into package use.

Photo by Maksym Kaharlytskyi on Unsplash

RStudio

Package management strategies



IS THIS USEFUL HERE, OR AN UNNECESSARY DISTRACTION?

So far, we've talked about what R does when users install packages and where those packages end up going

This diagram outlines the ways in which users and admins can most successfully work together.

Taken from: https://environments.rstudio.com/reproduce.html

The three strategies are outlined in detail:

Snapshot and Restore
Shared Baseline
Validated
In addition to these three strategies, the strategy map above details a set of danger zones, areas where "who" is in control and "what" can be installed are mis-aligned to create painful environments that can not be reliably recreated. Identifying if you're in a danger zone can help you identify a "nearby" strategy to move towards.

Wild West

The wild west scenario occurs when users are given free reign to install packages with no strategy for reproducing package environments.

RECOMMENDATIONS:

If you are a single data scientist, or in a team of experienced data scientists, consider moving to the snapshot and restore strategy.

If you are working with a group of newer users, consider working with IT to setup the shared baseline strategy. Be careful not to slip into the ticket system scenario, which occurs if you ask IT to restrict the system without teaching them how to manage shared baselines. It might make sense to use the shared baseline strategy by default, and allow experienced users to step into the snapshot strategy.

Ticket System

The ticket system scenario occurs when administrators are involved in package installation, but they do not have a strategy for ensuring consistent and safe package updates; for example:

A user wants a new package installed, so they submit a ticket to have the package added
An admin receives the ticket, and manually installs the new package into the system library
This scenario is problematic because it encourages partial upgrades, is often slow, and still results in broken environments!

RECOMMENDATION

If your organization requires admin involvement for practical reasons, (e.g. you're working on offline server), consider adopting the shared baseline strategy.

If your organization requires admin involvement for strategic reasons (e.g. you have concerns about package licenses), consider adopting the validation strategy.

Blocked

The blocked scenario occurs when servers are locked down, but there is no strategy

in place for R package access. This strategy often leads R users to “backdoor” approaches to package access, such as manually copying over installed packages.

In this scenario, it is important for R users to level-set with IT on why R packages are essential to successful data science work. You may need to refer to the validation section of the site or the section on picking packages, both of which help explain where packages come from and address issues around trust.

Come to this discussion prepared to advocate for either the shared baseline or validation strategy. It may also help your admin team to know that there are supported products, like RStudio Package Manager, designed to help them help you!

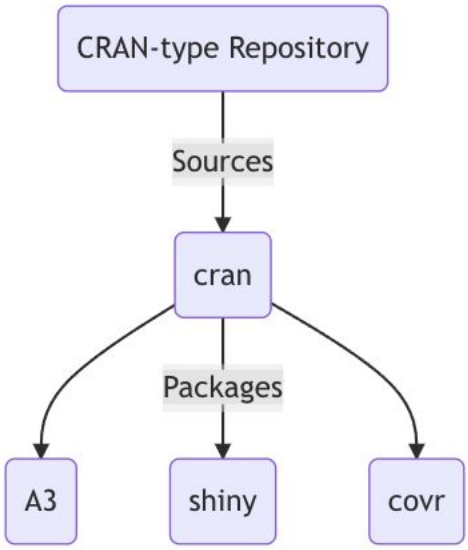
Why you might want RStudio Package Manager:

Photo by Cayetano Gil on Unsplash

1. You want a single on-prem location for all installs

More reasons to come…

RStudio



Repository - the primary vehicle for organizing and distributing packages. These are views of the data that your client (R, BiocManager, pip, etc.) uses to find and install packages.

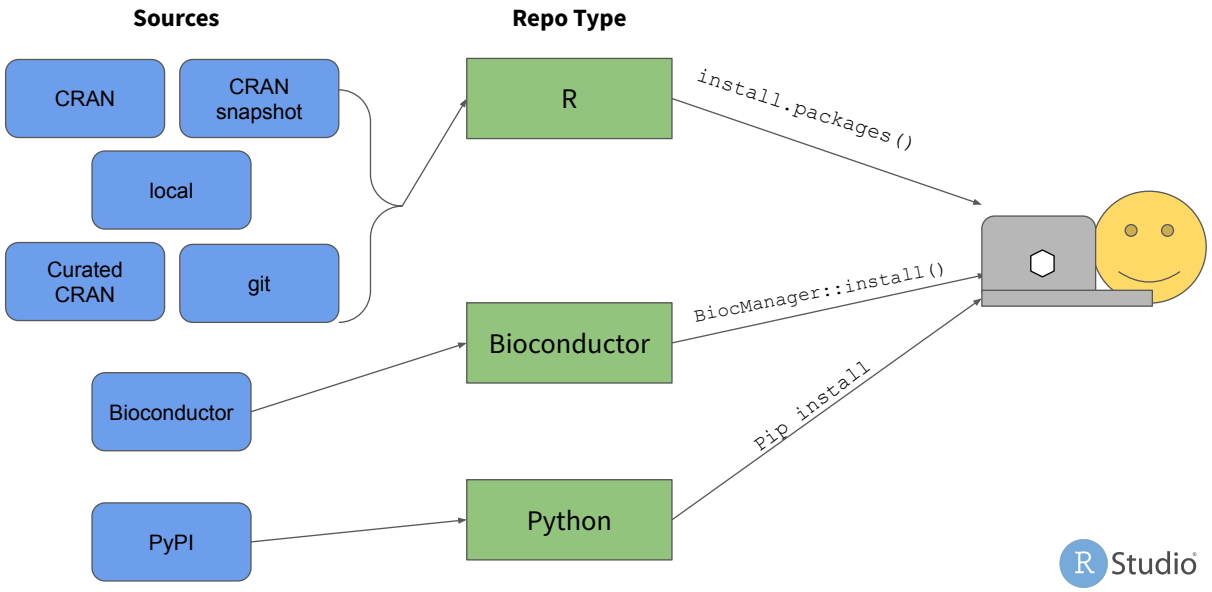
Source - Sources are collections of files from a single location (e.g. CRAN or PyPI).

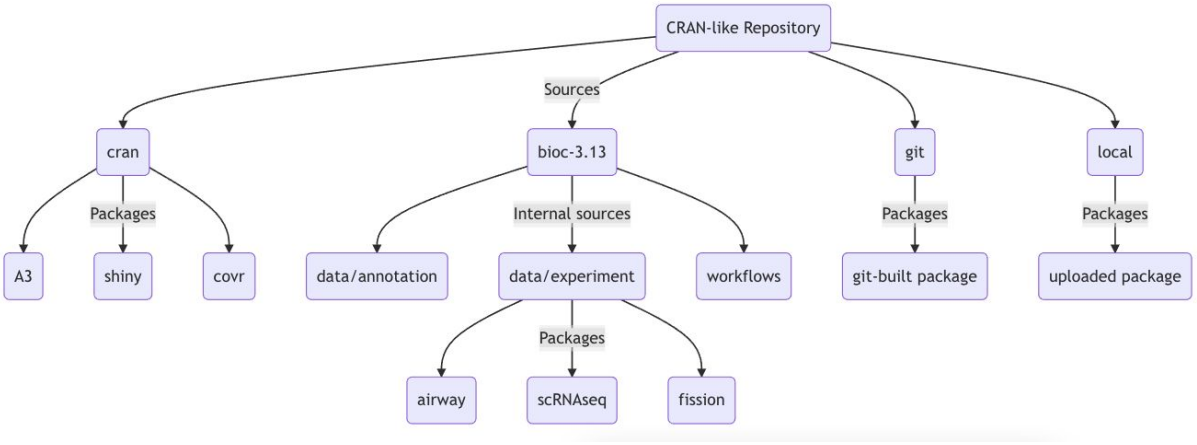
RStudio

Repos:

- RSPM doesn't create repos on disk
- Can have more than one source

RStudio Package Manager




CRAN-like Repository
Sources
cran
bioc-3.13
git
local
Packages
Internal sources
Packages
Packages
A3
shiny
covr
data/annotation
data/experiment
workflows
git-built package
uploaded package
Packages
airway
scRNAseq
fission

RStudio

Diagram your current or ideal package management strategy

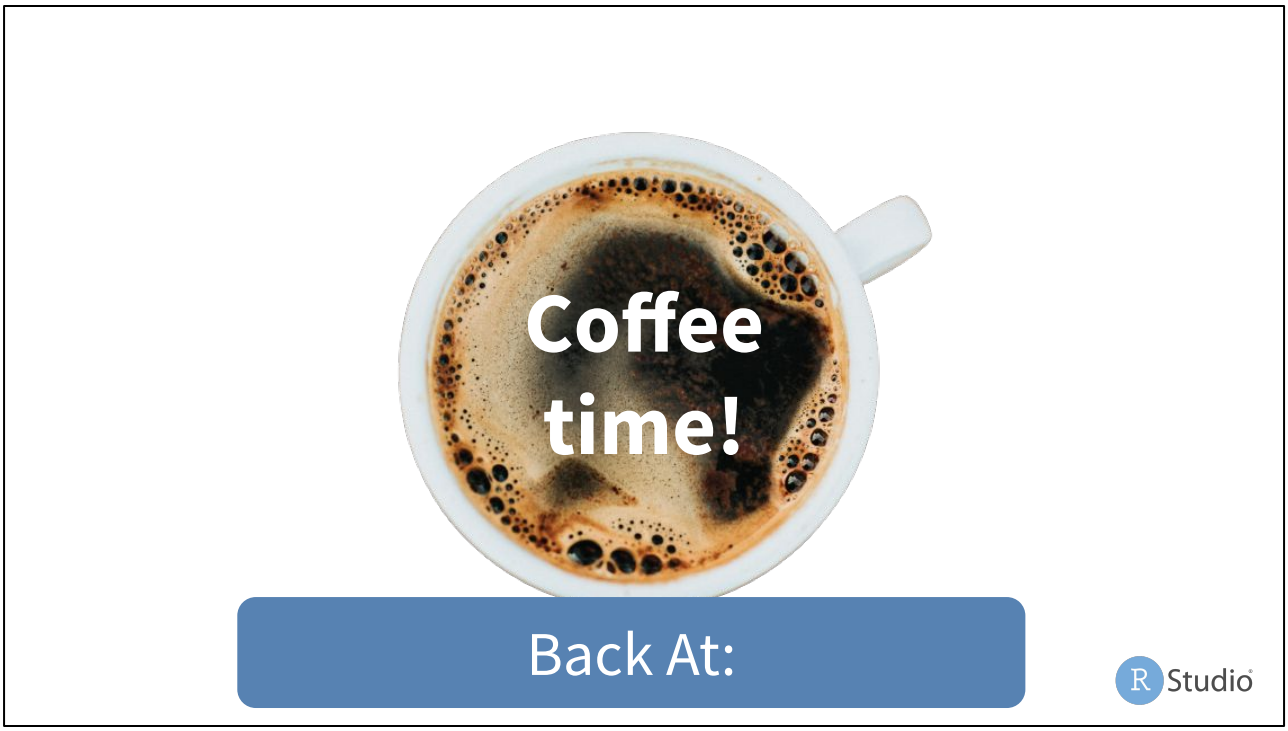
- What are your sources?
- How many repos do you have?
- What are the repo types?
- What are the connections between sources and repos?

RStudio

Configure a CRAN repo on RSPM

Let's run through the commands in `01_rspm.sh` to configure our first repo!

RStudio



15:00 - 15:30

Next up is Shannon

Photo by Nathan Dumlao on Unsplash

Setting a default repo in RStudio Workbench

File	Level	Use
`rsession.conf`	Whole Server	Configuring single repo server-wide
`repos.conf`	Whole Server	Configuring multiple repos-server wide
`Rprofile.site` or `Renviron.site`	Version of R	Configuring different repos per R version

RStudio

Set RStudio Package Manager as our default repo

Let's run through the commands in `02_rspm.sh` to configure!

RStudio

R package binaries

```
> install.packages("curl")
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
trying URL
'https://packagemanager.rstudio.com/cran/__linux__/focal/latest/src/contrib/curl_4.3.2.tar.gz'
Content type 'binary/octet-stream' length 905296 bytes (884 KB)
==================================================
downloaded 884 KB
* installing *binary* package ‘curl’ ...
* DONE (curl)
The downloaded source packages are in
   ‘/tmp/RtmpYE8pch/downloaded_packages’
```

RStudio

- RSPM serves pre-compiled package binaries for Linux and Windows
  - Linux binaries is something that CRAN does not do
  - CRAN only provide source code versions of packages to Linux users, and R tries to compile them itself when they're downloaded
- Since almost all scientific and technical computing happens on Linux, having Linux binaries available for a variety of popular Linux distros is a big deal
- And for packages where there are system-level shared libraries are required at *build* time, this:
  - simplifies configuring your development and deployment environments, since they no longer need to compile packages before they can execute R code
  - significantly reduces the time required to install a large number of packages, as is typical of some data science workflows in R

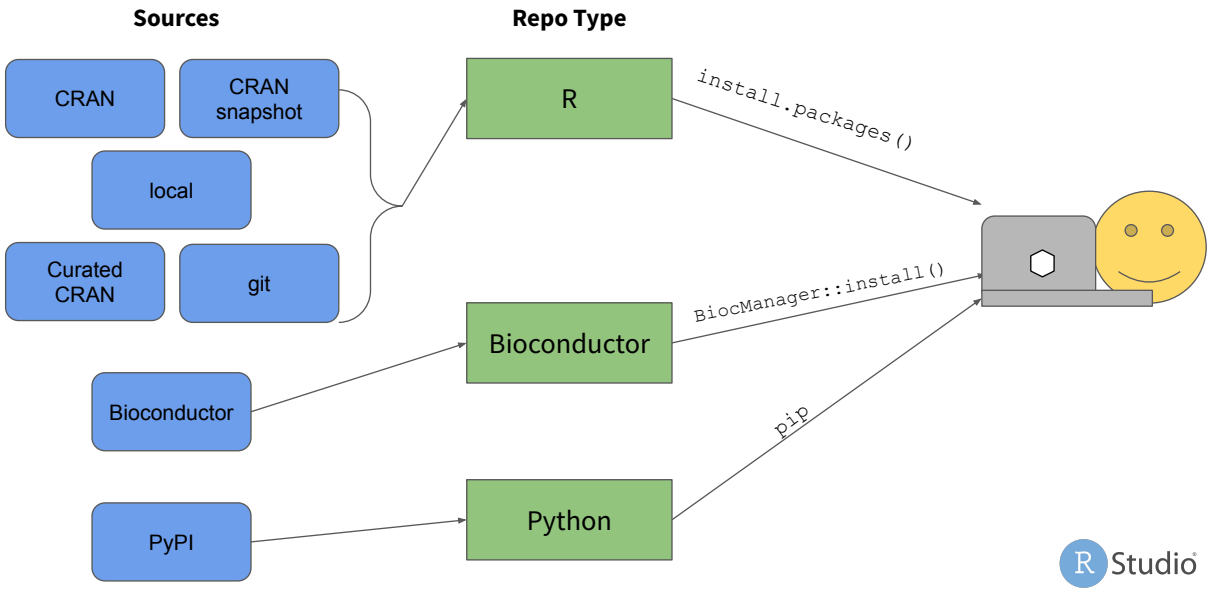
Run-time dependencies

```
* installing *binary* package ‘sf’ ...
* DONE (sf)
The downloaded source packages are in
   ‘/tmp/RtmpDwQ58g/downloaded_packages’
> library(sf)
Error: package or namespace load failed for ‘sf’ in dyn.load(file, DLLpath = DLLpath, ...):
 unable to load shared object '/usr/local/lib/R/site-library/units/libs/units.so':
  libudunits2.so.0: cannot open shared object file: No such file or directory
```

RStudio

Unfortunately, because run-time dependencies exist at the operating system level, it’s not possible for package manager to “solve” this problem, however, this is a much smaller proportion of packages than those which have build-time dependencies.

RStudio Package Manager



Let’s create a Bioconductor repo

Open up `03_rspm.sh` to configure!

RStudio

Why you might want RStudio Package Manager:

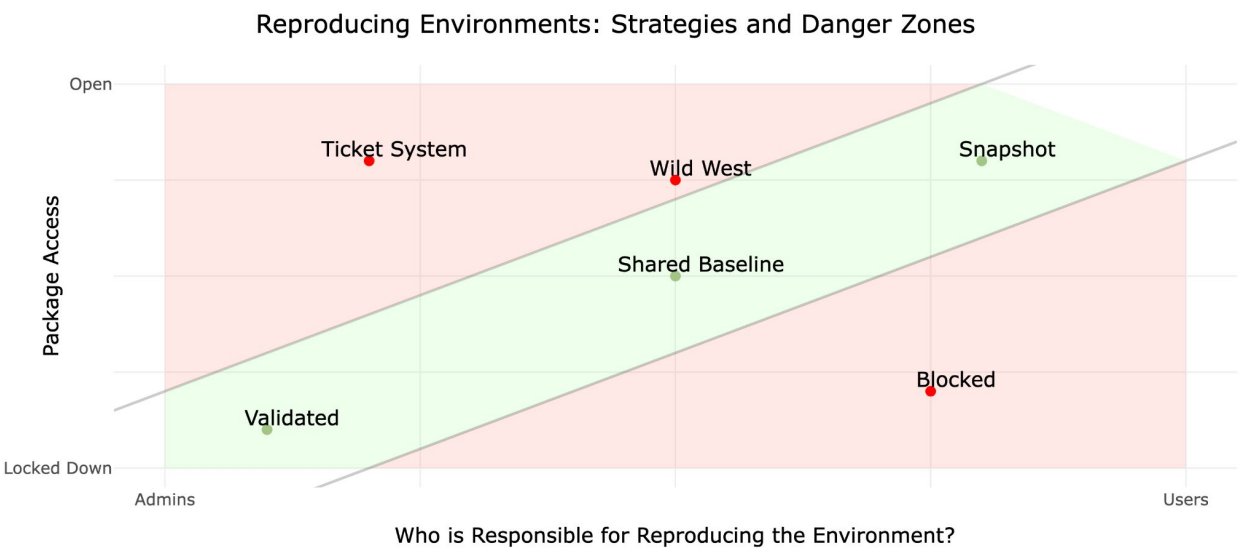
1. You want a single known source for all installs
2. You want to curate CRAN subsets

More reasons to come…

Photo by Cayetano Gil on Unsplash

RStudio

Package management strategies



Discuss validation / curated cran

Let’s create a curated repos

Return to 03_rspm.sh to configure!

RStudio

Why you might want RStudio Package Manager:

Photo by Cayetano Gil on Unsplash

1. You want a single known source for all installs
2. You want to curate CRAN subsets
3. You need to serve internally developed packages

More reasons to come…

RStudio

We're going to add a

How does RSPM work with Connect

When content is published to Connect, a manifest file is used that contains a snapshot of the developers environment.

It contains a record of:

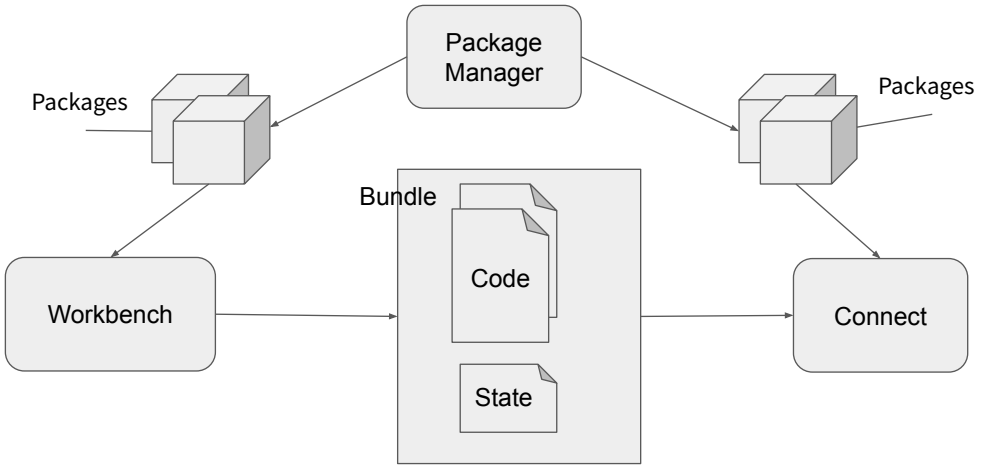
- The R version used
- The packages used
- The package versions used
- As well as some information about where the packages came from

RStudio

We'll create a manifest file later, but for now it's probably enough to know that the manifest is somewhat similar to a Python requirements.txt created when running pip freeze.
Remember this manifest and resulting bundle only contain a record of the users environment, not a copy of it.
This avoids problems that may be introduced where users are developing on one platform, say MacOS and then publishing to a Linux based system.

There's a nice one here (https://github.com/sellorm/bass_model/blob/main/manifest.json) that's pretty easy to underststand.

How Package Manager Supports Connect



Recap:

- Package Manager makes open-source and internal packages available inside your network
- Workbench is where data science and development take place
- Connect is where long-lived content is published and shared

Workflow

- Workbench users install packages from Package Manager to support the code they're writing
- When the users go to publish their code the publishing process creates a snapshot of the development environment state
  - This includes the R version
  - Packages and package versions
- The state and code are bundled together and transferred to Connect
- Connect unpacks the bundle
- It inspects the manifest.json (the state) and builds a local isolated environment using the same R version and packages
- It then runs the users code.

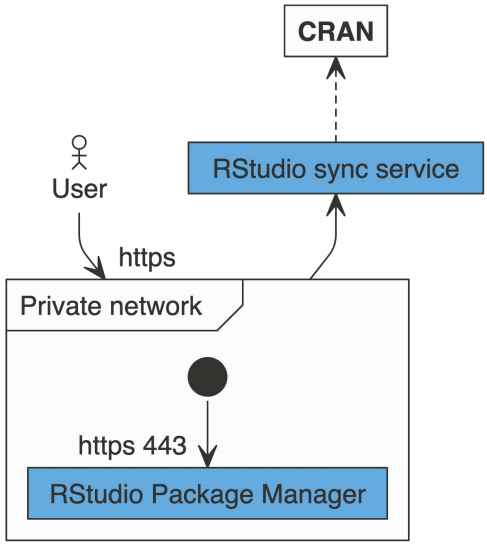
Why you might want RStudio Package Manager:

1. You want a single known source for all installs
2. You want to curate CRAN subsets
3. You need to serve internally developed packages
4. You are in an offline environment

Photo by Cayetano Gil on Unsplash

RStudio

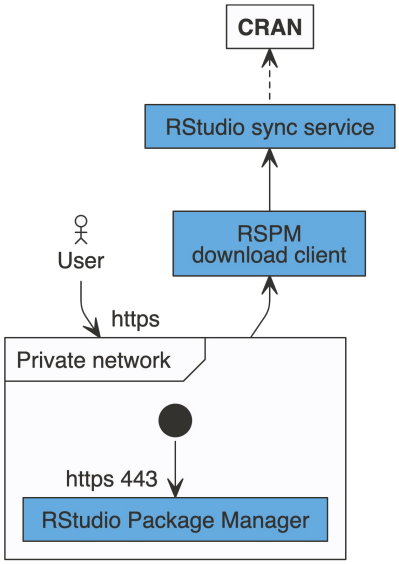
Configurations: networking (online)



RStudio

in online mode package manager can reach the sync service and keeps itself up to date
The sync service is an API, provided by RStudio that provides all of the snapshot data and packages for both R and Python.
Package Manager uses the API to retrieve metadata and packages on behalf of users.
By default, RSPM will check the sync service for new metadata updates daily and will retrieve packages for users on-demand, however, those packages are cached locally by RSPM so that subsequent requests for the same package are served from the cache, rather than re-fetched from the sync service

configurations: networking (offline)



RStudio

Note that while this configuration is possible, it's far from optimal due to the additional administrative burden of managing updates

RSPM ships with a command line tool that can retrieve data from the sync API and dump it all out to disk.
An admin would need to take this tool to a host that is online, use it to fetch all of the available data (this can be many gigabytes) and then transfer all of that to the RSPM machine.
Once that's done, there are additional commands to ingest that into package manager.

RECAP:

How do your users
get their packages today?

How is that working for everyone?

RStudio

Think, Pair, Share - let's talk about what we do today and

RECAP:

How do your users
get their packages today?

How is that working for everyone?

RStudio

Think, Pair, Share - let's talk about what we do today and

This is not a 1 person job!

Admin

- Configure repositories
- Sets default repository for development environment.

User

- Installs packages to development environment
- Snapshots package requirements for each project

RStudio

End of Day 1

Back At: 9 am tomorrow!

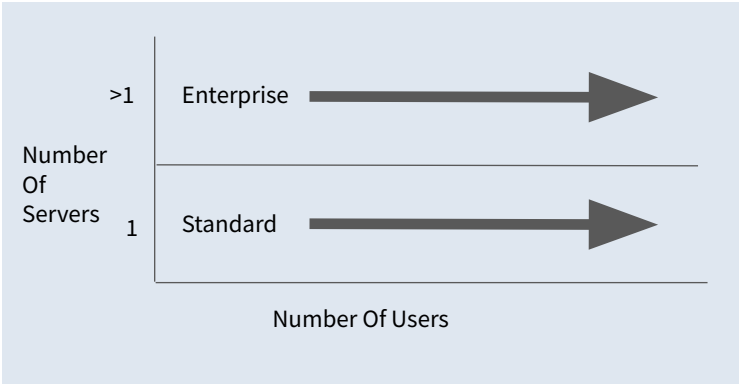
RStudio

Are you in the right workshop?

RStudio Team Licensing

3 Pro Products: RStudio Workbench, RStudio Connect, RStudio Package Manager

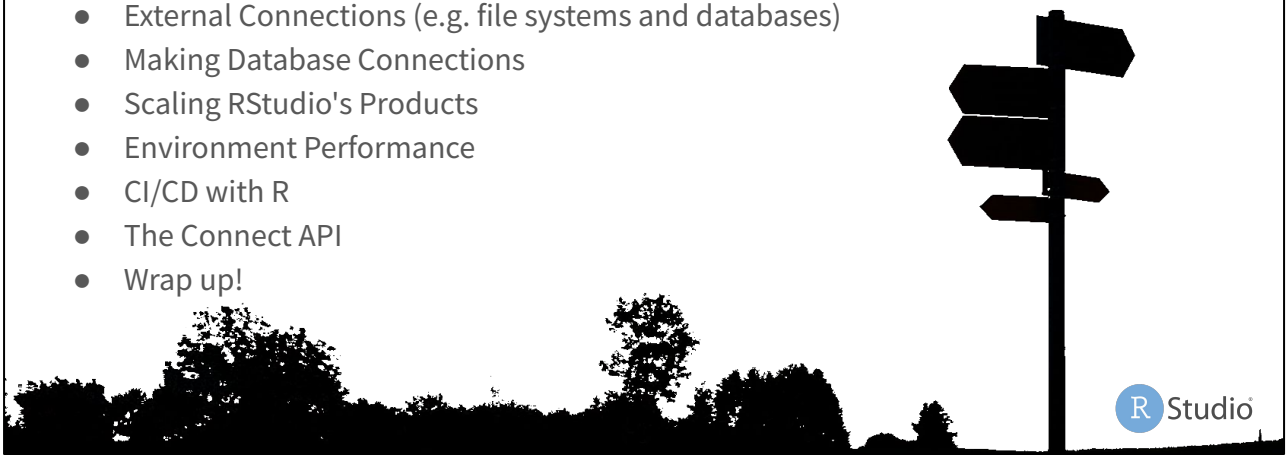
Licensed separately or as a bundle: RStudio Team



RStudio

Topics for Day 2:

- Recap day 1 and Connect publishing breakdown (this bit!)
- Connect R content types (including app design)
- External Connections (e.g. file systems and databases)
- Making Database Connections
- Scaling RStudio's Products
- Environment Performance
- CI/CD with R
- The Connect API
- Wrap up!



Next up: Connect publishing - Sam E

Photo by Jan Huber on Unsplash

Connect publishing breakdown

RStudio

Overview

In this section we will discuss the different ways you can publish content to Connect!

How do I deploy content to Connect?

(1) Push-button or interactive

(2) Git backed

(3) Programmatic

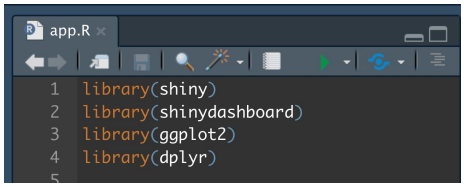
RStudio

Intro

- How do you take your app from Workbench and deploy it to Connect?
- In general, there are three ways you can deploy content to Connect:
  - (1) Push-button or interactive - from the RStudio IDE or Jupyter Notebook you can deploy to Connect using the GUI.
  - (2) Git backed - Connect will import your code from git and deploy.
  - (3) Programmatic - Deploy from the command line, R console/script, or using the Connect API.
- Each method has their pros/cons and valid use cases.
- To learn more read the RStudio IDE - RStudio Connect: User Guide docs.

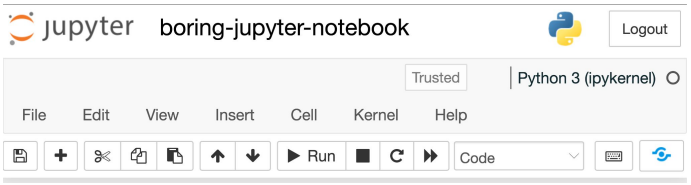
(1) **Push** // Push button deployment

RStudio IDE



💡Add Connect as a publishing account in the RStudio IDE.

Jupyter Notebook



💡Use an API key to enable publishing from Jupyter Notebooks.

RStudio

Overview

- The push button deploy is the most simple / user friendly deployment options.
- It allows users to push their code directly from the IDE to Connect.

RStudio IDE

Demo

- Deploy this shiny app.
- For any content that is deployable the publish icon will appear.

Configuration

- In order to publish from the RStudio IDE to Connect you need to configure RStudio IDE.
- *Tools > Global Options > Publishing > Connect… > RStudio Connect*
  - Enter the public URL of your server and hit enter
  - You will be prompted to authenticate
  - Authenticate, and then the RStudio IDE should confirm you have verified your account. Click “Connect Account”
  - Tip! If you are not sure what the URL is you can find it in Connect by going to: *Content > Publish > Shiny App > Next > Next > Next > Copy the URL*

- For further learning read the Publishing from the RStudio IDE docs.

Jupyter Notebook

- Jupyter notebook also has support for push button deployment.
- We will not be covering during this session, but I have left detailed speaker notes on this slide for anyone who wants to learn more.

Things to be aware of

- Push button deploy **works for** all **R content in the RStudio IDE** and **Jupyter Notebooks**.

- There is **NO push button deploy**
  - for **python** content other than Jupyter Notebooks.
  - for **Jupyter Lab** or **code-server**.

APPENDIX

Jupyter Notebook

Demo

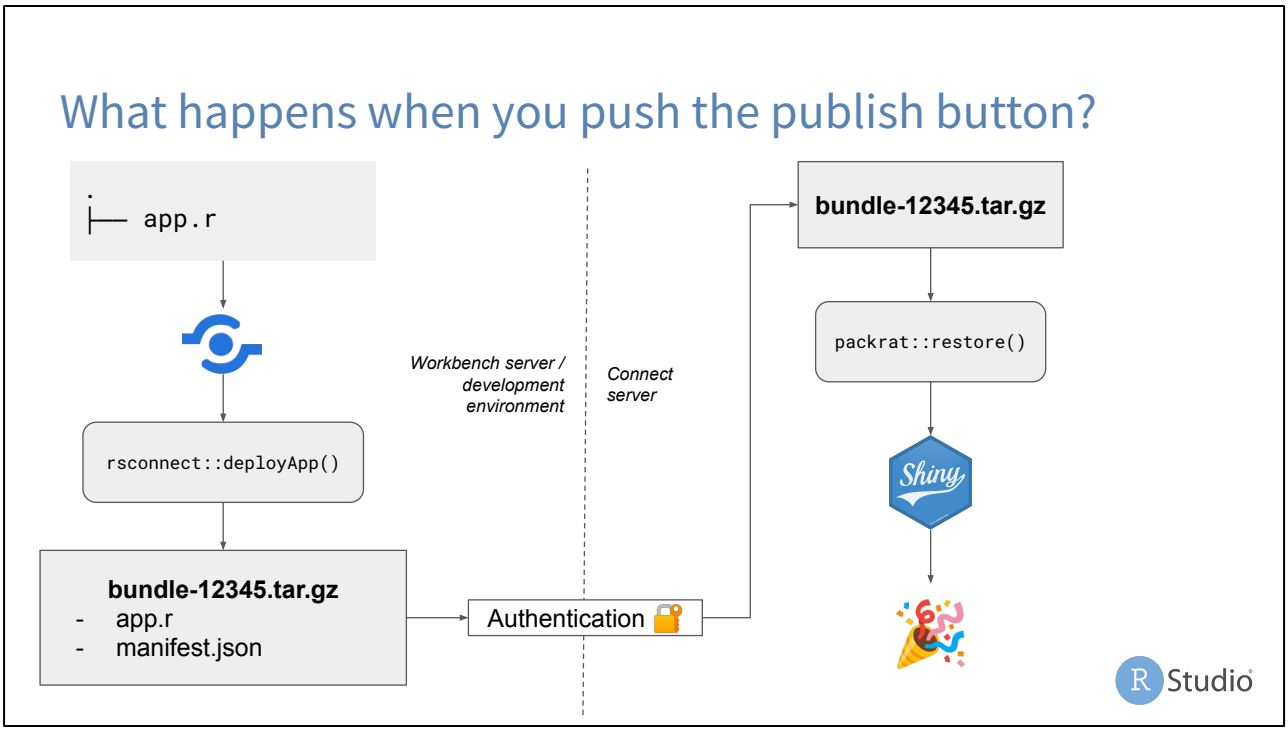
- Open workbench, launch a Jupyter Notebook session, and create a new empty notebook.
- Make sure that you have rsconnect-python installed in your environment. You need this to publish. For example, I like to always use a virtual environment

```
# set up a virtual environment
/opt/python/3.10.4/bin/python -m venv .virtualenvs/py3104
source .virtualenvs/py3104/bin/activate
python -m  pip install --upgrade pip wheel setuptools
python -m  pip install rsconnect-python
# register the venv as kernel
python -m  pip install ipykernel
python -m ipykernel install --user --name ~/.virtualenvs/py3104
```

- Click on the blue publishing button > Publish to RStudio Connect
- Should I publish with source code?
  - If you publish the source code the code will be executed on the Connect server. This allows you to re-run and schedule the content.
  - In most cases, I think you should publish the source code, but if you want to publish content that will not be re-run you can choose to publish the finished document only.

Configuration

- Click on the blue publishing button > Publish to RStudio Connect
    - Server Address -> Same as the configuration for the RStudio IDE
    - API key -> paste an API key that you can obtain from the Connect UI. To get learn more about obtaining an API key read the API Keys - RStudio Connect: User Guide docs.
    - Server Name -> Give the server a nickname
- For further learning read the rsconnect-jupyter User Guide docs.



Overview

This is a high level overview of what happens after you use push button deploy in the RStudio IDE. It can be helpful to know a little bit about what is happening behind the scenes if you ever need to debug a failed deploy.

After calling rsconnect::deployApp() a few things happen behind the scenes:

- A manifest.json file is created (more on that later with git backed deploy)
- A tar file (or tarball) is created named bundle-xxx.tar.gz. This tarbal is sent to connect behind the scenes.
- After authentication is successful the tarbal is sent to the Connect server.
- Connect then uses packrat to rebuild the environment.

Tip

In settings > publishing > check “Show diagnostic information when publishing” for more verbose output

Rsconnect directory

What is the ***rsconnect* directory** that gets created after I publish?

- This directory contains **meta-data** about your published content.
- The RStudio IDE uses the data in this directory to remember where you

- have published the content.
- When you clear deployment history in the RStudio IDE UI it deletes this directory.
- You should **not delete this content or edit it by hand**. If you delete it by mistake, you can restore it from version control. If you do not have it in version control you can attempt to re-create it by hand, getting the content GUID from Connect. However this is likekly to be error prone and is really a last resort.
- The RStudio IDE uses this content to know when to create a new content vs. update an existing one.
- Should you c**ommit this directory to version control** so that you can consistently publish your content no matter which device you are on.

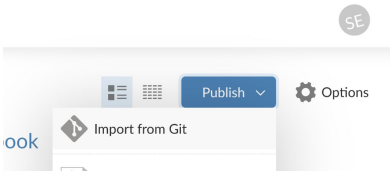
Python

Also talk in broad terms about the equivalent python flow if time permits. Main different is a requirements.txt is used in addition to the manifest.json file.

(2) **Git-backed** // Git-Backed Publishing

Rstudio Connect can publish content from a git source:

1. Create a *manifest.json* file
2. Commit and push code to a git repo
3. Connect will monitor and redeploy when changes are detected



🌐 R example: app/manifest.json
🌐 Python example: app/manifest.json

💡 *Publishing a **private** repo? Update* `/etc/rstudio-connect/rstudio-connect.gcfg` *with git credentials (docs).*

RStudio

Overview

Connect supports deploying content from any valid git source (it can be any git provider, e.g. GitHub, GitLab, AzureDevOps, etc.)

- Why do git backed deploy?
  - Automatically push the latest changes to production
  - Deploy multiple branches for testing (e.g. dev branch and main branch)

- Demo
  - Demo how to perform a git backed deploy using demo-shiny-penguins.

- Before doing a git backed deploy you need to write a manifest.json file.
  - This file documents all of the required application dependencies.
  - In R: `rsconnect::writeManifest()`
  - In Python: `rsconnect write-manifeset <type>`

- Auth and private git repos:
  - The current support for authenticating with private Git repositories is limited to a single set of credentials which must be configured on the Connect server.
  - will work with any authentication based on HTTP basic auth.
  - Use a service account that can be an owner of all the repos that need access.

    - See the admin guide and the user guide for details.

- R vs. Python
    - The *manifest.json* file for R lists all of the dependencies in the manifest.json file.
    - The *manifest.json* file for Python references the *requirements.txt* file. It takes a hash of the file so that it knows when it changes.
    - In Python you can choose to write your *requirements.txt* by hand, or let rsconnect generate it for you

- How it works
    - At a pre-defined interval a git fetch will be performed.
    - The decision to trigger a new deployment is made after a periodic fetch of the source repository, and will only occur if the specified branch and path prefix have changed since the last fetched commit.
    - You can customize the frequency in which RStudio Connect checks git using the `Git.PollingFrequency` configuration setting.

(3) **Programmatic** - Programmatic based deployment

There are several tools that allow you to programmatically deploy content to Connect:

- **R**
  - rsconnect package
  - connectapi package (just went live on CRAN 🎉)
- **Python**
  - rsconnect-python CLI tool
- **Other**
  - RStudio Connect API (the most powerful)

Combine these deployment tools with tools like GitHub Actions, GitLab CI, Azure Pipelines, etc. to create flexible CI/CD pipelines.

RStudio

Overview

Programmatic deploy is the most flexible of all the options. For purposes of time we will not demo this now, but we will dive deeper into programmatic deployment later with Shannon.

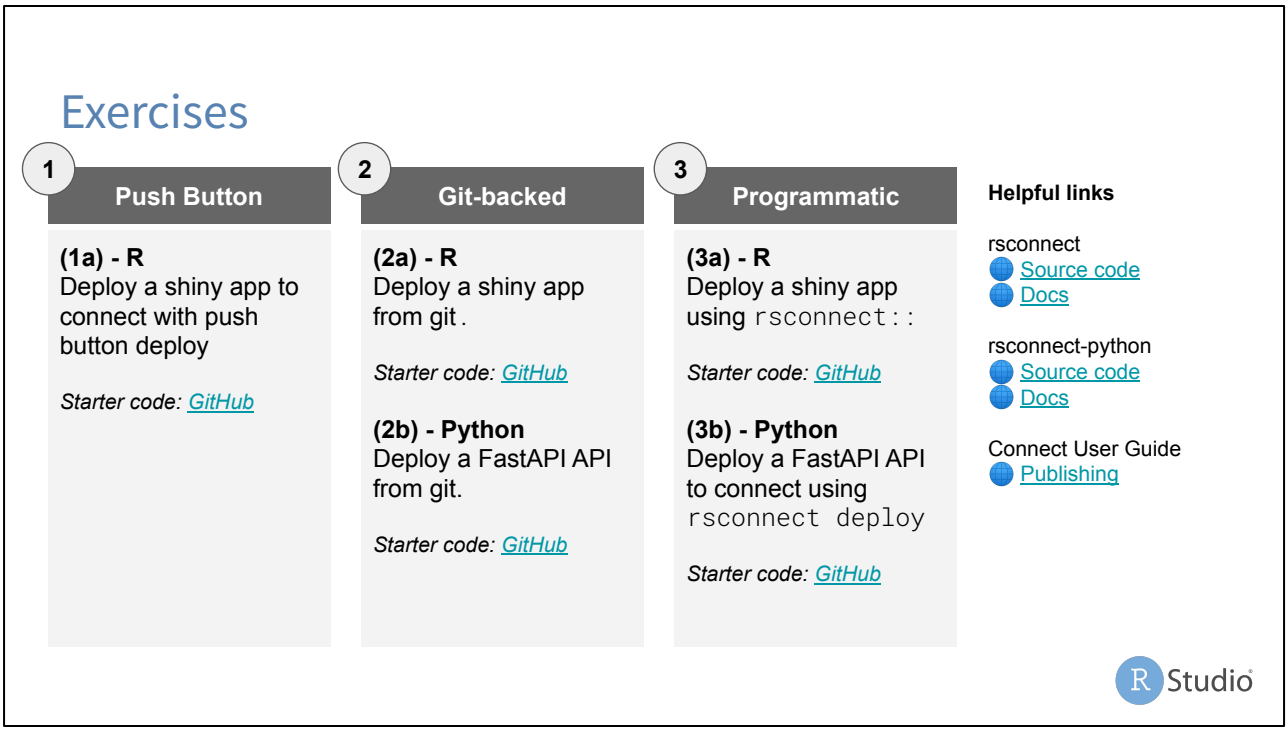
- You can…
  - Combine with other tools such as GitHub Actions or Azure DevOps
  - Connect can deploy content into Connect.
  - Deploy content from a script running on your computer or another server.

- For R content you can use the ***rsconnect*** (docs | source code) and/or ***connectapi*** (docs | source code) package.
- For Python you can use the ***rsconnect-python*** (docs | source code) packages.
- Or you can use the ***RStudio Connect API*** (docs) from any language.

- **Demo**
  - Demo push button deploy using the same file from the earlier demos
  - Note that for it to work we need to have set up our Connect account already

- **Auth**

- When using programmatic based deployment outside of the RStudio you must authenticate using **API keys**.
- Users can obtain API keys directly from the Connect UI.
- Refer to the API Keys section of the user guide for more details.



Instructions

- Each participant should do the activity on their own computer.
- Practise deploying a shiny app using each of the 3 deployment methods
- The python content (2b) and (3b) are optional.
- (1) Push button:
    - Click the link to access the start code
    - Copy and paste the code into a new app.R file on your Workbench Server
    - Publish to Connect
- (2) Git backed:
    - Using the app.R file from number 1, us the rsconnect package to generate a manifest.json file.
    - If you have time, publish your two files to a github repo
    - Then, deploy the app to Connect using git backed deploy
    - If you are short on time, or do not want to publish the files to GitHub you can use the starter code provided and publish from that URL
- (3) Using the same app.R file from number deploy to connect with a programmatic deploy

Effective App Design on Connect

RStudio

Now that we know how to deploy content to Connect, let's talk about…

- What kind of content we can deploy on Connect
- How we can optimize the performance of that Content through good design patterns and configuration

What can you publish to Connect?

This same data is presented in an easy to digest table (but without the nice logos!) on the next slide.

What can you publish to Connect continued?

	R	Python	Both
App frameworks	Shiny	Dash, Streamlit, Bokeh	/
API frameworks	Plumber	Flask, FastAPI	/
Reports	RMarkdown	Jupyter Notebooks	Quarto
Data Storage	/	/	Pins
ML-Ops	/	/	Vetiver

RStudio

App frameworks

- These are tools that let you build rich interactive web-based experiences for your users
- Web-app building tools

API frameworks

- These allow you to write APIs that can be hosted on Connect
- What the APIs do is up to the person who writes them
- Of course Connect itself also has an API, but Connect also allows you to host your own

Reports

- Reports are a special type of content in that they can be scheduled and outputs can be emailed
- These can be used to generate static (or nearly static) reports that are updated on a schedule and then hosted on Connect itself as well as having the output emailed to users.
- One non-obvious use-case for this facility is running ETL and ETL-like jobs that need to be run on a schedule and have the results of the process emailed to developers

Data Storage

- Pins is an open-source data-storage package for R and Python
- As well as other storage options (including local storage and cloud storage

- backends) it can use Connect as a storage location
- Pinned data can be read back out using the same powerful access controls that Connect uses for all content types
- Pins will also version your data, allowing you to retrieve previous versions as necessary

ML-Ops

- Vetiver is RStudio's open-source ML-Ops framework for both R and Python
- It allows users to train, test and publish their models and support for Connect is built right in

Effective App Design

We will cover the following topics of effective app design:

- Monolithic vs. composable apps
- Data
- Runtime settings
- Benchmarking & profiling
- Testing

RStudio

Effective App Design - Monolithic vs. Composable

There are two main approaches for structuring your content:

	Monolithic	**Composable**
What is it?	One unit of content is deployed to Connect.	Two or more units of content are deployed to Connect that work together.

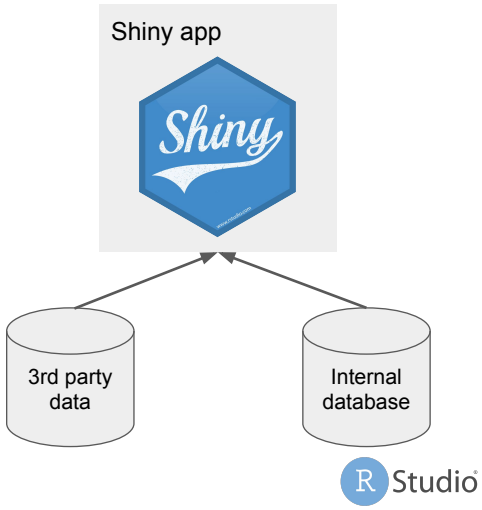
RStudio

Overview

- Explain the difference between monolith and composable
- What does this mean in the context of Connect?

Effective App Design - Monolithic Content

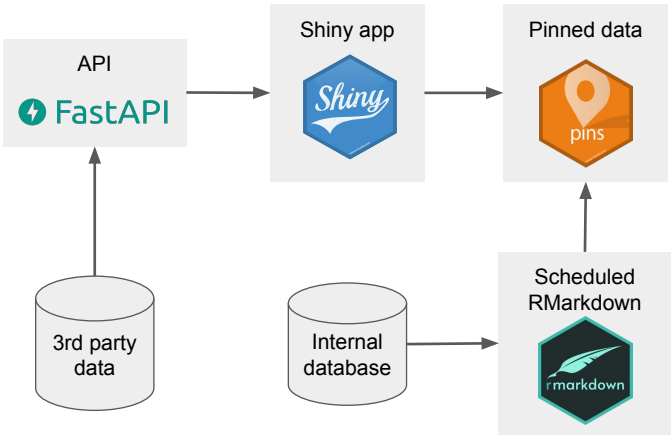
- Can have the exact same feature set as a composable app
- All code and business logic lives in the same code base



- These sort of applications are typically where users start on their app development journey.
- The “getting started” docs for most frameworks will have people build this sort of app and for many users, there’s never any need to think about performance beyond that.
- You can go a very long way with this sort of application, including many optimisations which would allow this sort of app to scale effectively.
- For example, here are some well known monoliths in the wild:
  - Pandas
  - Stackoverflow (listen to hansleminutes podcast #847)
- What are some signs that your monolithic app strategy may not be working anymore?

Effective App Design - Composable Content

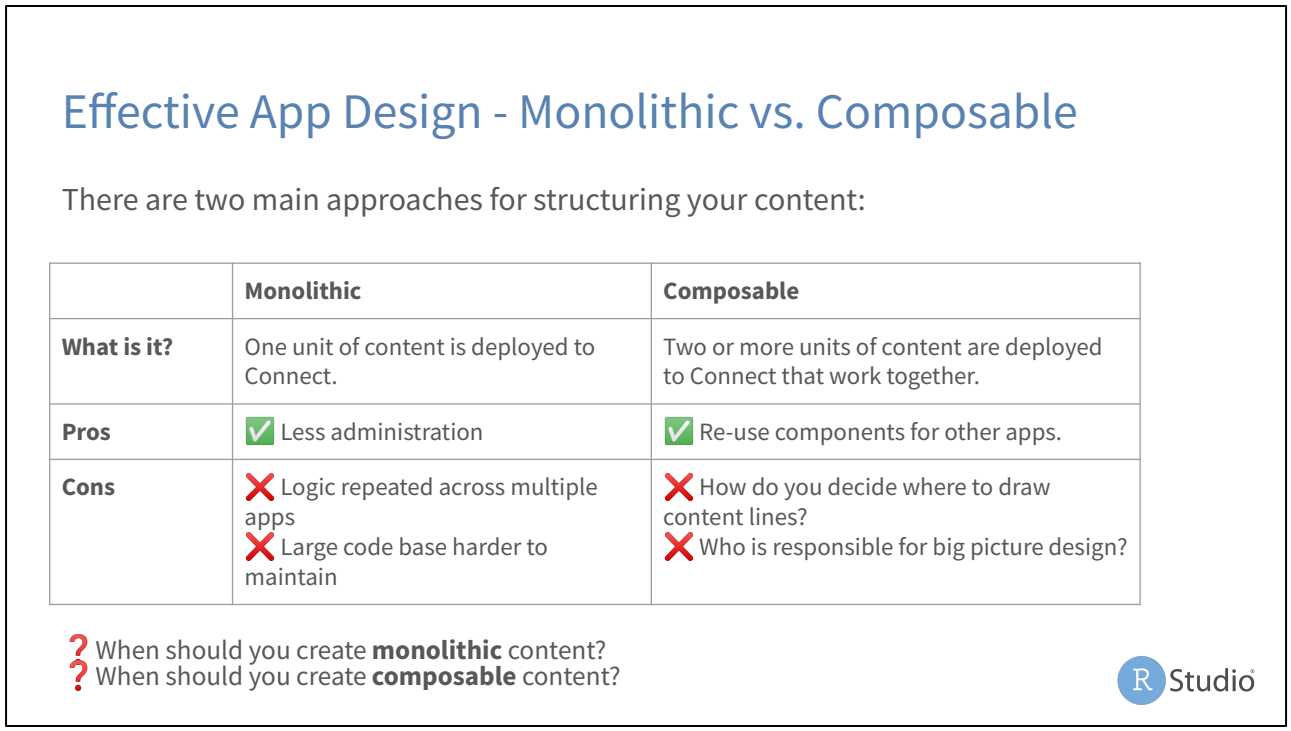
- Connect hosts many different types of content.
- Content can talk to each other, expanding the possibilities of what you can build.
- Use the right tool for the right job.



RStudio

Overview

- Composable apps combine together multiple apps to create one complete picture.
- They can help reduce work by reusing certain components across multiple apps.
- Example of composable app in the wild:
  - Tidyverse
  - Bike Predict example



Overview

- Discuss the content on the slide
- Other considerations:
    - Think about how the server is managed and designed. What workload types it the server optimized for?

Questions

- At what point do you take a monolith application and break it out into a composable app?
- When should you create a monolithic app?
- When should you create a composable app?

Conclusion

- Key takeaway - there is no right answer. You need to decide based on what works best for your app.
- In general the advice would be to start small, try not to over-engineer, but be mindful of how data flows through your app and consider - even in the early stages - what options there may be to modularise later.
- A working app that's slow but is easy to maintain, can sometimes be better than a fast app that's difficult to understand and maintain.
- Think about your use case, what your users really need and what you have the appetite to build and maintain

Effective App Design - Data

Data science focused apps often work with data. Common tasks are:

- Data loading (reading data from a database, flat file, etc.).
- Data crunching (e.g. joins, aggregations, model training).

If poorly implemented these tasks can often result in performance issues.

RStudio

Overview

- Data is a critical part of many data science focused applications.
- Consider:
  - Does your app use data?
  - How much data?
  - What is it doing to the data?
- If poorly designed, data can often be the bottleneck to your app.

Loading data

- In order to use data, you need to load it into memory.
- It may be OK to wait 10 seconds to read data in an interactive R session, but it is not OK in an app!

Crunching data

- Both R and Python can be slow compared to other tools like SQL.
- Both R and Python can potentially use a lot of memory.

Effective App Design - Data Toolbox

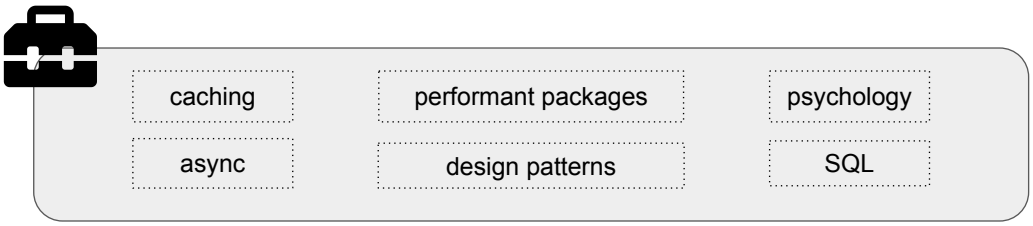
❓ What tools can we use to improve our apps performance with data?

- Database
- Not loading data into R process / server memory until last moment
- Caching (issues with pre-caching)
- Pre-processing through R markdown (snowflake -> pin -> shiny app()
- API endpoint that is called from shiny app

RStudio

Effective App Design - Data Toolbox

❓ What tools can we use to improve our apps performance with data?



RStudio

Overview

- Review each section and ask the class for example of tools.

Caching

- Allows you to save R / Python objects so they do not need to be re-created.
- This reduces the overall computing power required by your application.
- Example: caching with Shiny in RStudio Connect
- Tools:
  - Shiny: Using caching in Shiny to maximize performance
  - Streamlit: Optimize performance with st.cache
  - Dash: Memoization

Performant Packages

- Not all packages are created equal. Some packages can work with data much faster than others.
- Choose a package that is right for the job.
- Consider the trade-off between development time and data crunching time (e.g. data table will manipulate my data much quicker, but I can code the logic much quicker in dplyr).
- Example packages:
  - R
    - dplyr -> dbplyr / dtplyr
    - datatable
    - vroom

        - arrow
    - Python
        - ibis
        - arrow

Psychology

- What can you do make the app feel faster or more performant?
- Ideas:
    - Show loading spinner
    - Sample data (e.g. do you need to plot every point?)
    - Paging

Async

- What is async?
    - Both R and Python are single threaded. Despite being very fast, they can only do one thing at a time.
    - In the context of web apps, much time is often spent waiting (e.g. on a database, on a http request).
    - This time spent waiting is basically time wasted, the Python / R process is doing nothing except waiting.
    - Async frees up this waiting time so that R and Python can go do something else while they are waiting.
    - Learn more here: https://fastapi.tiangolo.com/async/
- Tools
    - FastAPI in python is very easy to get started with asynchronous
    - In R use the promises package to enable async
- Watch this awesome talk about speeding up API and web app performance: https://www.rstudio.com/resources/rstudioglobal-2021/plumber-and-future-async-web-apis/

Design patterns

- Pre-aggregate or crunch the data outside of the app.
- Keep N number of processes running.
- Consider the order of your operations (e.g. do not filter after a group by)
- Do all heavy compute when app starts up one time (e.g. the app is slow to start, but quick after that).
- Store your data in the “right format” (e.g. CSVs are faster to read than RDS)
- Choose the right tool for the right job (e.g. FastAPI vs. plumber)

SQL

- Can we perform the data crunching in SQL instead of R / python?
- Use tools like dbplyr or ibis to avoid writing SQL code!
- Ralf will cover in more detail later today.

Effective App Design - Data Sources

❓ What are the pros and cons of using each data sources?

- In app data
- Pins
- SQL
- Files on the local system / server
- Files on the cloud
- API

RStudio

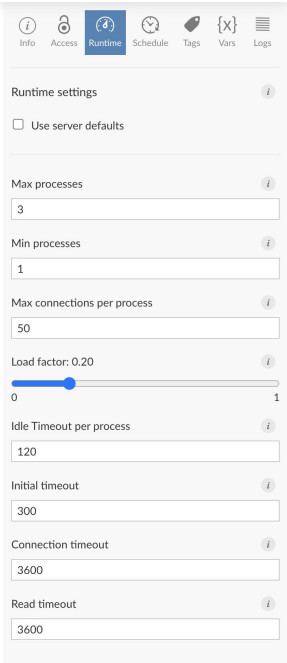
Exercise

For each data source ask the audience each question in turn and then discuss their answers

Effective App Design - Runtime Settings

Connect allows you to finetune your apps runtime settings (read the docs!)

- **Max processes**: The max number of processes / node
- **Min processes**: The min number of processes / node
- **Max connections per process**: The max number of client connections allowed to an individual process.
- **Load factor**: When to spawn a new process



Suggested reading: Scaling and Performance Tuning in RStudio Connect
Simulate runtime settings and events: Shimmer Shiny Sizing

Overview

- RStudio comes with several built in tools / settings to optimize the performance of your app.
- What is a process? What is a connection?
    - When you run a shiny app on your local computer it will run one process.
    - Each R and Python app will take at least one process.
    - That single process can server more than one user though… this is where connections come in!
    - Analogy:
        - One process is like a single chef
        - One connection is a customer
        - They can only do one thing at a time, but they can context switch
        - E.g. while the pasta is boiling the chef can prepare a salad
        - The number of customers the chef is serving is the number connections
    - Often, the process is just listening, waiting for something to change (e.g. for you to push a button).
    - We can take advantage of this by serving multiple users with the same process.
- Review each setting.
- For future reference read:

- https://docs.rstudio.com/connect/user/content-settings/#content-runtime
- Watch this talk: https://www.rstudio.com/resources/rstudioconf-2018/scaling-shiny/

Exercises

For each use case, what would your suggested runtime settings be?

1 Mission critical API

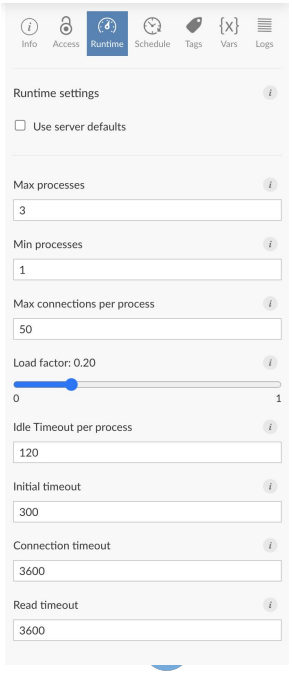
Roo's Doggy Day Care hosts a services for customers to send voice notes to their client. It relies on an API hosted on Connect. Customers expect the experience to be quick and seamless. On rainy days more than 100 hundred customers will send notes.

2 Simple Shiny App

Bodie's Burger Truck has a shiny app they use to analyze sales. The app connects to a SQL database where the data is stored and uses plotly to create interactive visualizations. The app is used by the two main investors to monitor performance.

3 Prediction API

Where is the Pow? hosts an API that predicts when and where the best powder days for skiers and snowboards will be. When queried the API loads a Microsoft's Turing NLG model which is VERY large (20 billion parameters…). The results of the model are then returned to users in JSON format.



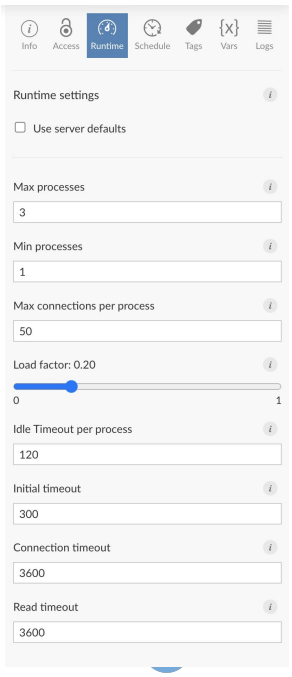
Exercise

- Talk each example through with the class
- Ask for suggestions and discuss the responses

Effective App Design - Runtime Settings

❓ What are some general rules of thumb for each setting?

- Max processes
- Min processes
- Max connections per process
- Load factor



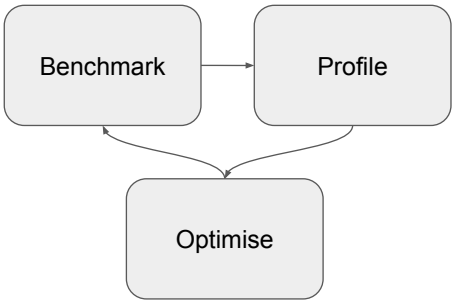
Reference: https://docs.rstudio.com/connect/user/content-settings/#content-runtime

- **Max processes**: Pick a value that will support the expected number of concurrent users or requests.
- **Min processes**: Pick a number close to Max processes if your application or API pre-loads a large amount of data to be shared by every user. Pick a small number to minimize the amount of memory consumed.
- **Max connections per process**: Pick a small number if your content involves heavy computation. Pick a larger number if your content shares data between users. (e.g. Pick a large number if your content takes a long time to load, but after loading is very fast.)
- **Load factor**: Pick a small number if your content loads quickly but involves expensive computation. Pick a number closer to 1 if your content loads slowly, but after loading is fast OR if you want to minimize the amount of memory.

Effective App Design - Benchmarking & Profiling

Why is my app SLOW!??

I don't know, but a profiling tool can tell you (maybe).



Benchmarking / profiling tools
Shiny: shinyloadtest / profvis
Plumber: profvis
Dash: Dash Dev Tools / pyinstrument
Streamlit / Bokeh / Flask / FastAPI: pyinstrument

Read this: https://mastering-shiny.org/performance.html
Watch this: Shiny in production: Principles, practices, and tools

RStudio

Overview

- Profiling and optimization are great, but without a bench mark it is very difficult to know if the changes you make actually are effective.
- This is why benchmarking is so important. In order to assess performance, we need to measure it and track it over time.
- We do not have time in this workshop to dive into benchmarking and profiling.
- Key takeaway is to to remember to benchmark and track performance over time.

Effective App Design - Testing

Unit tests: use to verify individual parts of the app are correct.

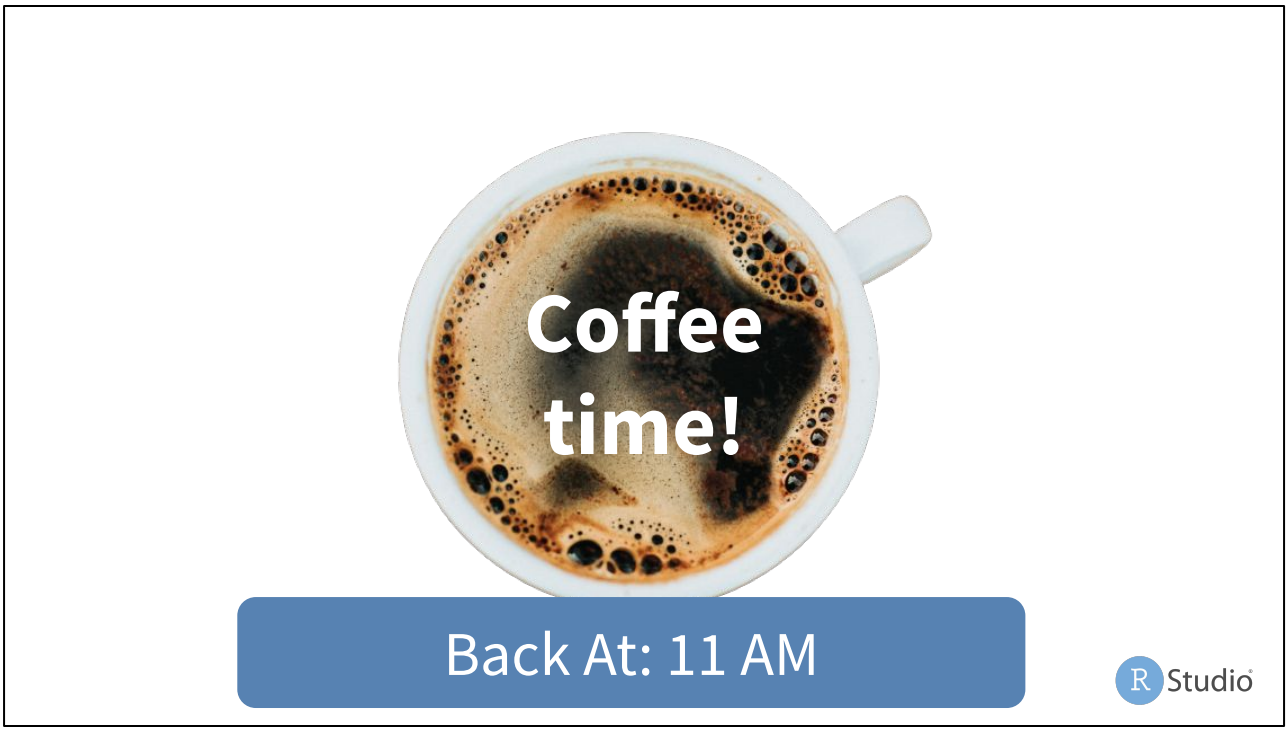
- testthat
- pytest
- shinytest
- Dash Testing

Integration testing: use to verify that different components of the app work together correctly.

User acceptance testing (UAT): use to validate app with end users.

- Google sheets
- Azure DevOps
- GitHub issues
- Etc.

RStudio



Next up: Ralf - External Connections

Photo by Nathan Dumlao on Unsplash

External connections

RStudio

Ralf

External connections used as data sources

- Files in the file system
  - Local or shared files
  - Different file formats
- Databases
  - Relational databases
  - Analytics databases
  - Document storage
- Other
  - APIs
  - Blob storage (e.g. S3)
  - Data lakes & Big data

RStudio

Remember, everything listed here is a concern of the language - in this case R - and not something that any of the pro products will particularly help you with.
Where there are helpers - for instance when loading a dataset in the IDE - all it's really doing automatically writing and running some R code for you.

RStudio Workbench & RStudio Connect

External connections are handled by R or Python, never by Workbench or Connect itself
“You can connect to any data source from Workbench and Connect to which you can connect to from R or Python (running on Linux).”
Two integration levels:

- Configuration at OS level
- R or Python package

Main difference between Workbench and Connect is typically authentication

- Interactive authentication is typically only possible on Workbench
- Service accounts with credentials in environment variables are an often used pattern on Connect

RStudio

Files

- Local files are easy to use but need to be included in the deployment to Connect
- Network shares can be mounted on all servers via NFS
  - Global mount
  - Authorization via Unix file access rights
- Integration with Windows shares via SMB/CIFS possible
  - Global or user-specific mounting
  - Typically access rights correspond to **mounting** user
  - Exception: multiuser mount with additional authentication token (e.g. Kerberos)

RStudio

File formats

- Spreadsheet programs
  - CSV, TSV, XLS(X)
  - {readr}, {readxl}
- Statistical software
  - SPSS, Stata, SAS
  - {haven}
- Large datasets and fast operations
  - Feather, Parquet
  - {feather}, {arrow}

RStudio

Exercise

- Directory `~/class-repo/data/data` contains `nycflights13::flights` dataset
    - Load the data files using suitable packages and compare with original dataset
- Test app/document in Workbench that uses data
- Deploy app/document to Connect using the same data
- In general (but not here) the location for data files will differ on Workbench and Connect. How could one handle that efficiently?

RStudio

Databases

- No “one size fits all”
  - File based (SQLite, DuckDB)
  - RDBMS (PostgreSQL, MariaDB, MSSQL, …)
  - NoSQL and document stores (MongoDB, Cassandra, …)
- Handled in greater detail in next section.

R Studio

Databases are access either via standard Linux ODBC drivers, or via custom R packages.

This likely means that there may be a few more niche databases, where access from R is difficult or impossible

Photo by Maurice Garlet on Unsplash

Other Data Sources

- There are many other potential data sources.
- Anything that can be used from R or Python (on Linux) can also be used in Workbench and Connect.
- Two examples with more details
  - API
  - AWS S3

RStudio

Example: Bikesharing API

- The R Markdown document `api-example.Rmd` contains code from the bikeshare example to retrieve and process data from an API
- Test on Workbench
- Deploy to Connect and put on a schedule

RStudio

- People can also play with the Rmd to inspect other parts of the API

Example: Amazon S3

- read/write data using {aws.s3}, {botor} or {paws} packages
- {aws.s3} is native R but needs to be kept in sync with AWS changes
- {botor} wraps AWS's boto3 library but requires Python + {reticulate}
- {paws} is a native R implementation of a large number of AWS' APIs
- Authentication:
    - IAM tokens or default role

RStudio

None of these packages are maintained or controlled by RStudio - they're all open source efforts

Making database connections

RStudio

Ralf

Database connections in R

- {DBI} package provides a consistent interface independent of the database
  - Driver packages for individual databases
  - odbc package as generic driver
- Example:

```
con <- DBI::dbConnect(RPostgres::Postgres(),
                      dbname = 'DATABASE_NAME',
                      host = 'HOST',
                      port = 5432,
                      user = rstudioapi::askForPassword("Database user"),
                      password = rstudioapi::askForPassword("Database password"))
```

RStudio

ODBC

- ODBC specifies an API independent of the database and **also** independent of R
- ODBC drivers are available for most database systems making integration simpler
- The {odbc} package provides a generic way to use any installed ODBC driver
- Example:

```
con <- DBI::dbConnect(odbc::odbc(),
                      Driver   = "[your driver's name]",
                      Server   = "[your server's path]",
                      Database = "[your database's name]",
                      UID      = rstudioapi::askForPassword("Database user"),
                      PWD      = rstudioapi::askForPassword("Database password"),
                      Port     = 5432)
```

RStudio

ODBC Drivers

- ODBC drivers are libraries that translate the ODBC interface to the DB specific functions
- Location and additional attributes are defined in `/etc/odbcinst.ini`
- Example:

```
[PostgreSQL]
Driver = /opt/rstudio-drivers/postgresql/bin/lib/libpostgresqlodbc_sb64.so
Version = 1.6.0
Installer = RStudio ODBC Installer
```

RStudio

Data Source Name (DSN)

- DSNs combine the parameters needed to do the connection
- Configured in `/etc/odbc.ini` or `~/.odbc.ini`
- Use `isql <DSN> [username [password]]` to test before trying them in R
- Example:

```
[MyPostgresDB]
Driver   = "PostgreSQL"
Server   = "[your server's path]"
Database = "[your database's name]"
Port     = 5432
```

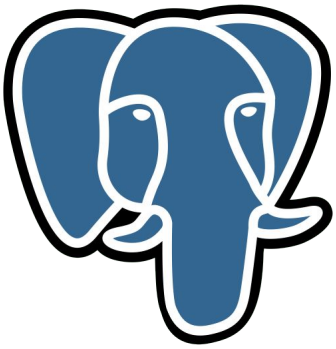
```
con <- DBI::dbConnect(odbc::odbc(),
         DSN = "MyPostgresDB",
         UID = rstudioapi::askForPassword("DB user"),
         PWD = rstudioapi::askForPassword("DB password"))
```

- DSN can point to different databases on Workbench (DEV) and Connect (PROD) simplifying code promotion

RStudio

Exercises: Connect to a PostgreSQL instance

- … using DBI and RPostgres
- … using DBI and odbc referencing the ODBC driver
- Inspect the DSNs configured in `/etc/odbc.ini`
- Test DSN using isql
- … using {DBI} and {odbc} referencing the DSN
- Do some simple stuff with the connection
    - Read a table into a data frame
    - Do a simple SQL query



RStudio

Connection details from /etc/odbc.ini:

```
[Postgres Dev]
driver = PostgreSQL
server = localhost
uid = rstudio_dev
pwd = dev_user
port = 5432
database = postgres

[Postgres Prod]
driver = PostgreSQL
server = localhost
uid = rstudio_prod
pwd = prod_user
port = 5432
database = postgres
```

dbplyr

- {dplyr} is used to write R processing pipelines to select, filter and modify data for in-memory data frames
- {dbplyr} allows you to treat database tables like in-memory data frames transparently translating the R code to SQL as needed
- This allows for operations on data that are too large to fit into memory.
- Move the operation to the data, not the data to the operation!

RStudio

Exercises

- Walk through the examples from https://dbplyr.tidyverse.org/articles/dbplyr.html in `dbplyr-example.Rmd`
- Also try to run the SQL created by show_query() via dbGetQuery()
  - Same results?
- Add collect() at the top of the pipeline
  - Effect on memory consumption in R?
  - Effect on execution time?

RStudio

The sample code produces some warnings. Can one improve the pipeline to remove these warnings?

Authentication: Username & Password

- Interactively providing username and password works in RStudio Workbench but not in RStudio Connect
- Putting the password into the source code is no option!
- Good alternatives:
  - Configure a DSN that includes the credentials
  - Get credentials from environment variables with Sys.getenv()
    - Environment variables can be configured in Connect on an app specific level

RStudio

Authentication: Kerberos

- Often databases use Kerberos tickets for authentication
  - For example MS SQL Server using “integrated security”
- On Workbench a Kerberos ticket can be created when PAM authentication is used.
- On Connect the code execution is important
  - Similar to Workbench when `CurrentUserExecution` is used with PAM authentication
  - For code executed by a service accounts (all APIs and documents!) one needs to programmatically create the ticket and keep it up-to-date
    - `cron` or `k5start` together with a keytab file

RStudio

Authentication: Other

- IAM roles
  - Global IAM roles can be assigned to the machines running Workbench or Connect
  - User-specific IAM roles are possible in Workbench together with the Kubernetes Launcher when the templating functionality is used
- Other tokens etc.
  - No established patterns

RStudio

Exercises

- Create an artifact deployable to Connect that needs a DB connection
- `rstudioapi::askForPassword()` works for interactive execution on Workbench but not on Connect
- Use username and password from environment variables
- Use a DSN with configured passwords
- In both cases differentiate between DEV settings on Workbench and PROD settings on Connect

RStudio

Fast track: Use dbplyr-example.Rmd and alter how the connection is created

Next up: Scaling RStudio products - Monanshi

Photo by Luisa Brimble on Unsplash

Scaling RStudio's products

Monanshi

Breakout Session

- What are the signs you've experienced as an administrator or user that has motivated you to think about scaling your architecture?
- What do you need to know before you can scale your architecture?

RStudio

…By scaling RStudio's products.

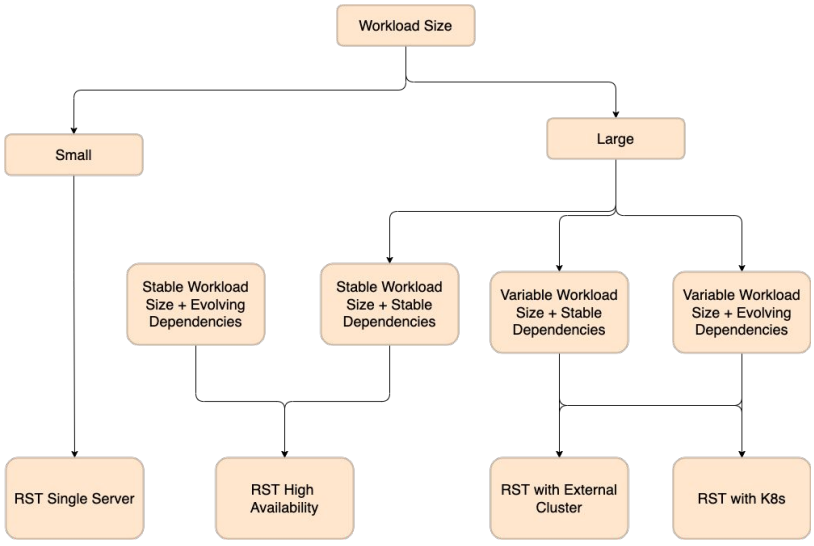
As your Data Science team grows or the team's workloads grow, you may find your initial architecture is not able to handle the load properly.

There are 3 options to scale RStudio Products:

- Clustering RStudio products
- Using RStudio with an external cluster such as Slurm or Kubernetes
- Running RStudio entirely within Kubernetes

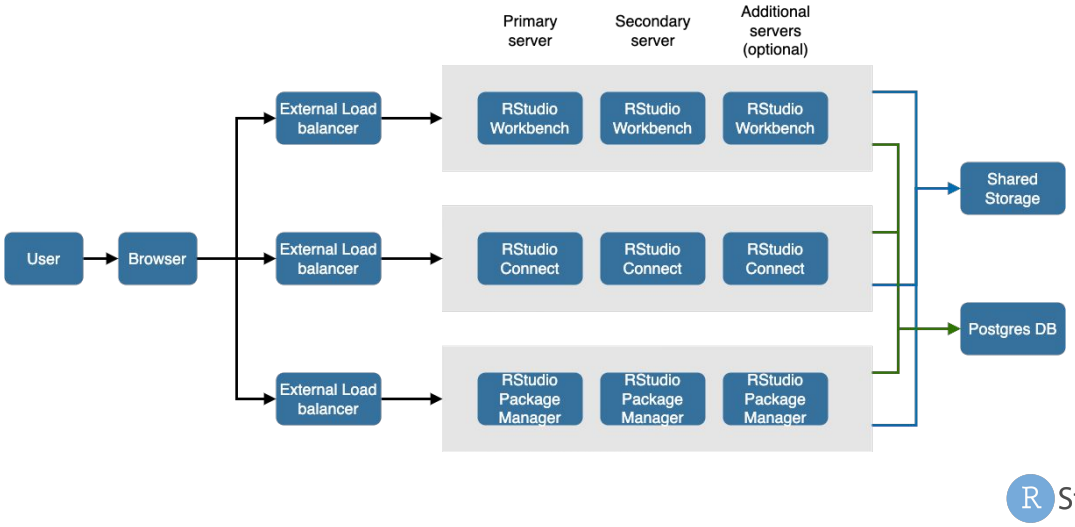
Photo by Hello I'm Nik on Unsplash

Scaling Decision Tree



RStudio

High Availability (HA)

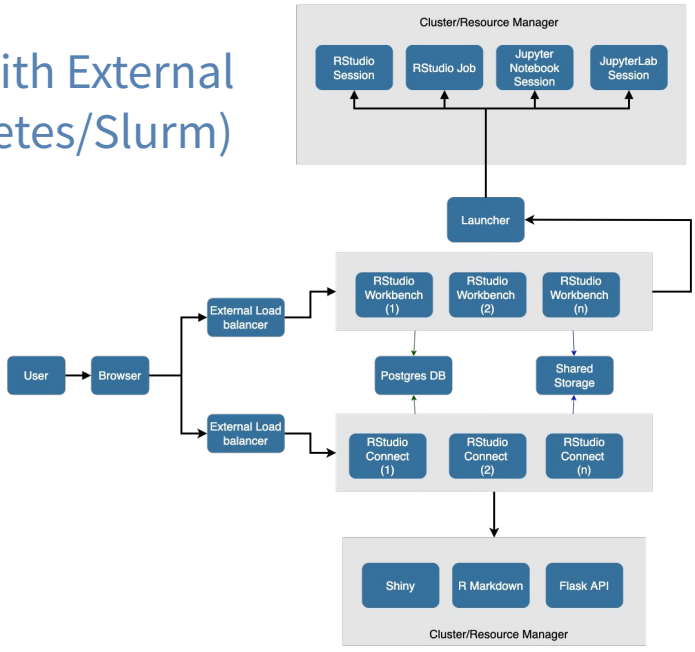


RStudio

- Load balancing to provide additional computational resources to end users
- High availability to provide redundancy
- Potential to use an autoscaler to add additional servers

Talk through in as much detail as you can

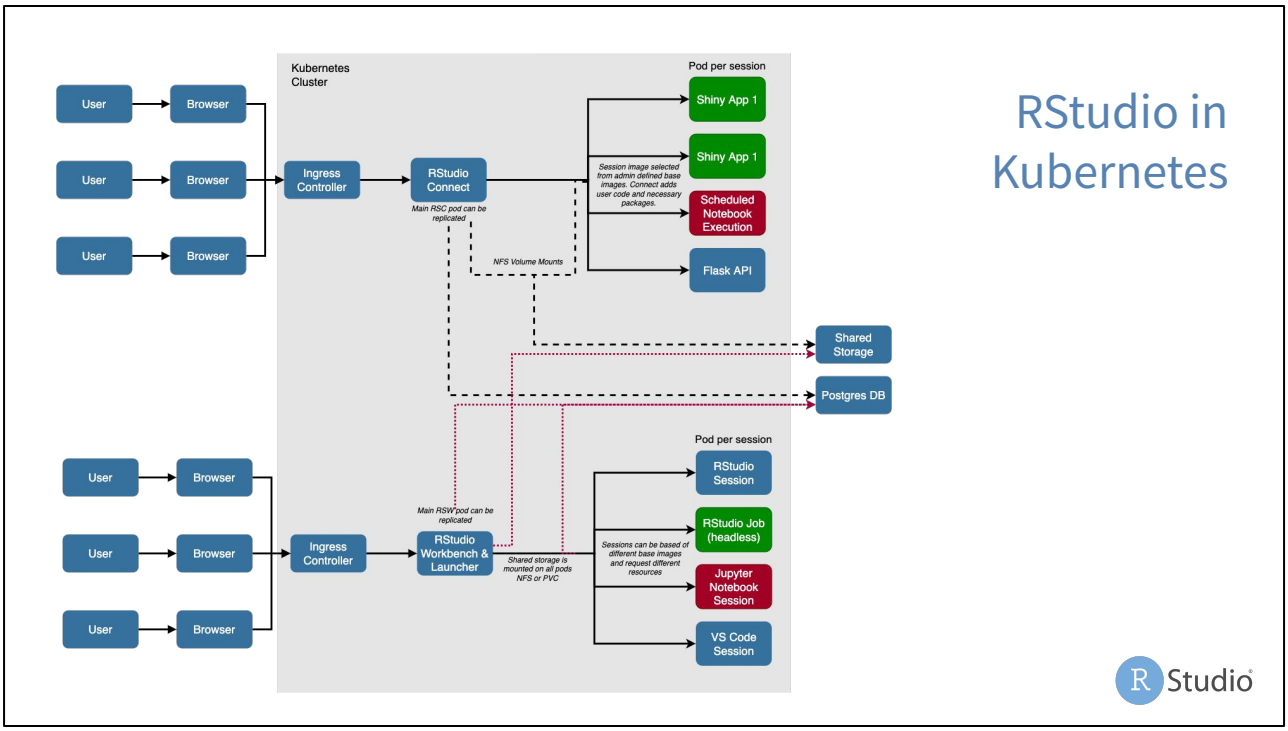
RStudio Team with External Cluster (Kubernetes/Slurm)



RStudio

Allows users to tap into compute resources without needing to maintain additional servers specifically for that purpose

Talk through in as much detail as you can - talk about SLURM here



- User sessions and jobs can run in isolated pods, with different base images
- Installation is managed in Kubernetes with Helm
- Configure number of replicas for high availability environments
- Talk about parallel cluster here :)

Talk through in as much detail as you can

Exercise: An Application in Production

RStudio

A data scientist has built an application and your mission is now to put it in….

RStudio

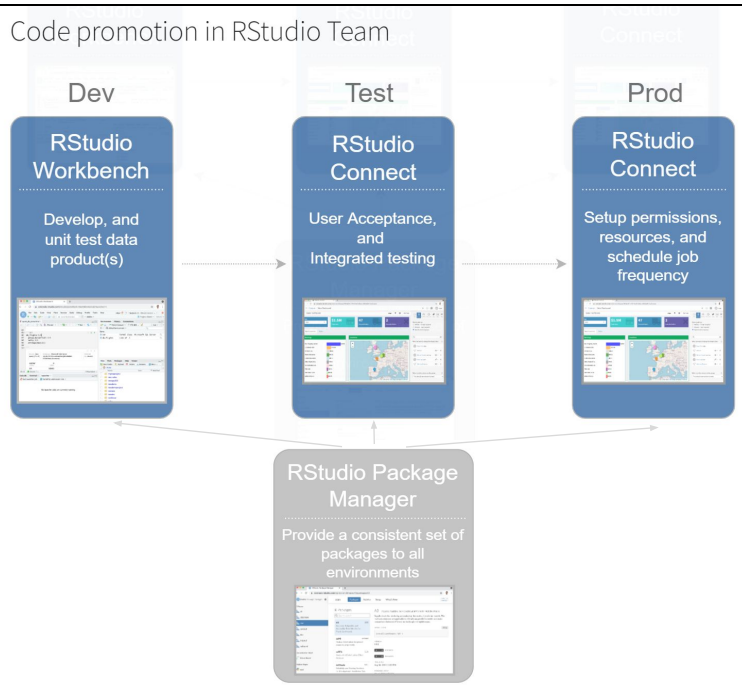
What does production mean?

Hopefully not:

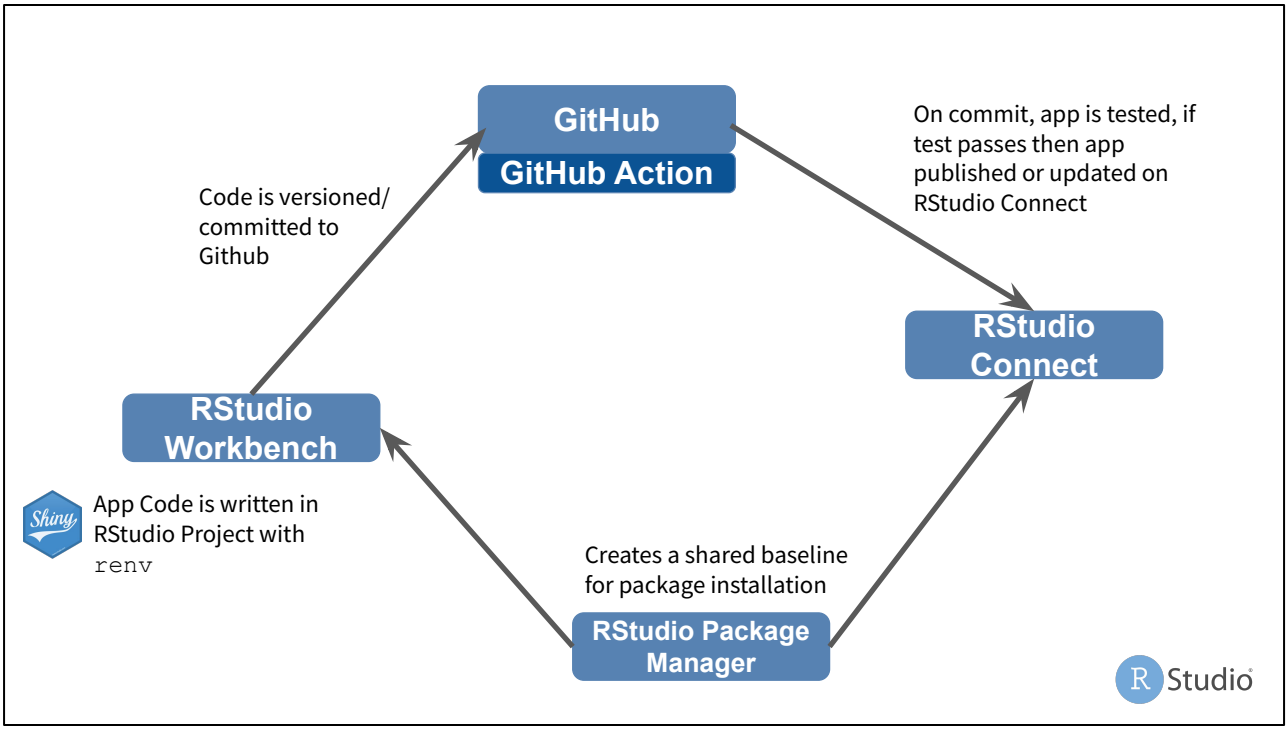
Instead we want:

1. **Reproducibility (including Versioning)**
2. **Testing**
3. **Continuous Integration / Continuous Deployment**

RStudio



For more details check out: https://solutions.rstudio.com/data-science-admin/code-promotion/


GitHub
GitHub Action
Code is versioned/ committed to Github
On commit, app is tested, if test passes then app published or updated on RStudio Connect
RStudio Connect
RStudio Workbench
App Code is written in RStudio Project with renv
Creates a shared baseline for package installation
RStudio Package Manager
RStudio

Workshop Exercises Outline

1. **Reproducibility (including Versioning)**
    - Set up the provided project files on the Workbench server provided by RStudio SolEng, loading the reproducible environment provided environment using renv, and setting up git/github version control using usethis.
2. **Testing**
    - Run the provided tests using shinytest2 based on the testthat workflow and check the outputs. Record a new test using the interactive record function.
3. **Continuous Integration/Continuous Deployment**
    - Set up and run our first github actions workflow for automated deployment to the Connect server when code is pushed to the git repository.
4. **Putting it all together**
    - Set up and run a github actions workflow for automated deployment to the Connect server after all tests are run and passed when code is pushed to the git repository.

RStudio

Lisa

1. Reproducibility (including Versioning)

RStudio

Reproduce the Environment.

Anatomy of the Project

Files Conf Workshop 2022

```
|   app.R
|   renv.lock

\---tests
    |   testthat.R
    |
    \---testthat
        |   setup.R
        |   test-shinytest2.R
        |

\---.github
    \---workflows
        |   connect-publish.yaml
        |   test-actions.yaml
        |
```

1. **Reproducibility (including Versioning)**
    - Using git, usethis, and reproducible environments using renv.
2. **Testing**
    - Using shinytest2 based on the testthat workflow.
3. **Continuous Integration/Continuous Deployment**
    - Automation using github actions and various community built action scripts to simplify the process such as the actions written by the r-lib team.

RStudio

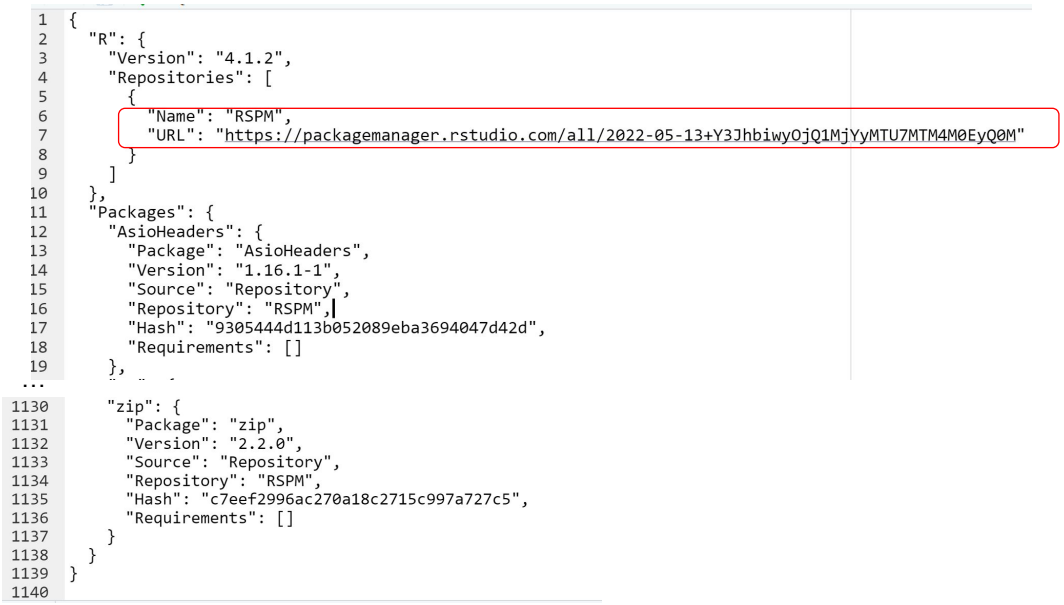
Bolded is what is provided initially

Will update this with final repo, can add a slide on RStudio Projects if we want to cover that (its mentioned in env performance outline)

Lisa note (remove before workshop): Project diagram outlining the files that will be included (app.r, renv.lock, .github, tests) - used https://stackoverflow.com/questions/9518646/tree-view-of-a-directory-folder-in-windows#:~:text=In%20the%20Windows%20command%20prompt,select%20%22Open%20command%20window%20here%22 for generating tree

renv.lock

Lisa

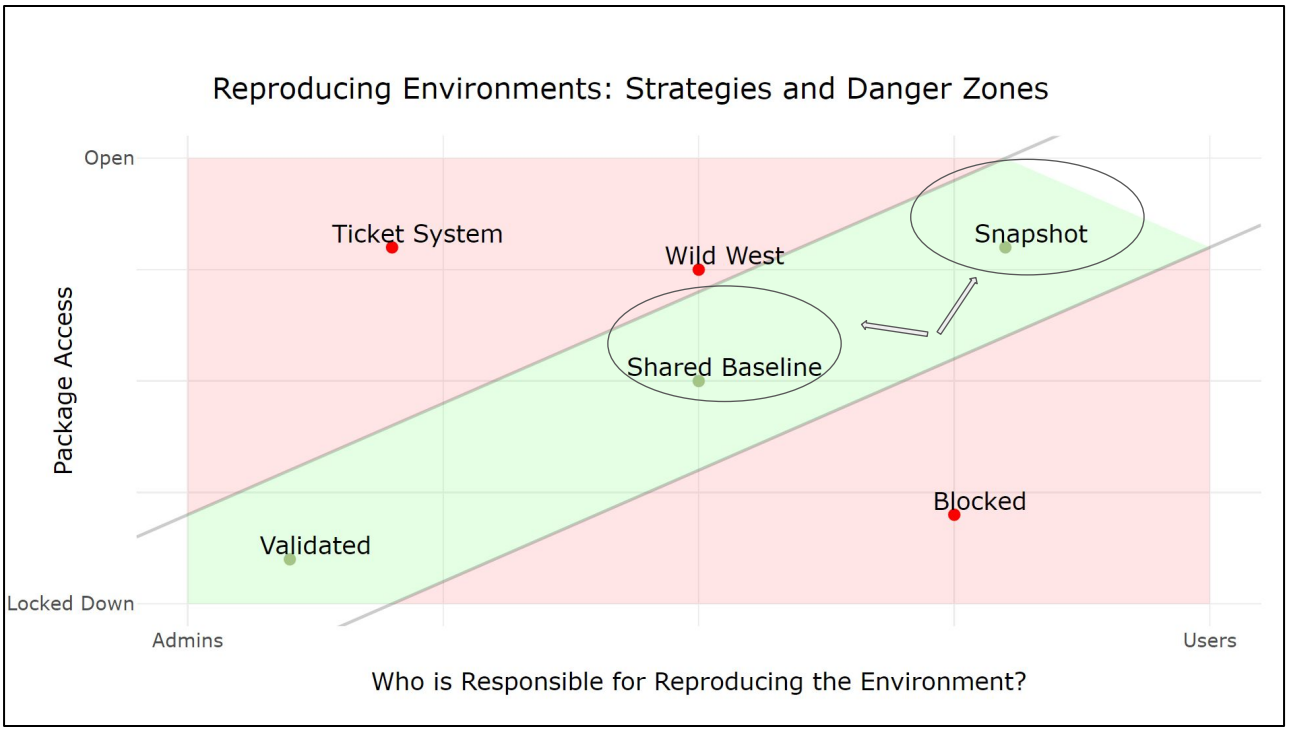


```
{
  "R": {
    "Version": "4.1.2",
    "Repositories": [
      {
        "Name": "RSPM",
        "URL": "https://packagemanager.rstudio.com/all/2022-05-13+Y3JhbiwyOjQ1MjYyMTU7MTM4M0EyQ0M"
      }
    ]
  },
  "Packages": {
    "AsioHeaders": {
      "Package": "AsioHeaders",
      "Version": "1.16.1-1",
      "Source": "Repository",
      "Repository": "RSPM",
      "Hash": "9305444d113b052089eba3694047d42d",
      "Requirements": []
    },
...
    "zip": {
      "Package": "zip",
      "Version": "2.2.0",
      "Source": "Repository",
      "Repository": "RSPM",
      "Hash": "c7eef2996ac270a18c2715c997a727c5",
      "Requirements": []
    }
  }
}
```

RStudio

Both strategies can be brought together - exploring the renv.lock file we can see that in our environment snapshot it is using package manager for using managed and preferred package versions

Screenshot of inside the renv file showing that its coming from a snapshot in rspm

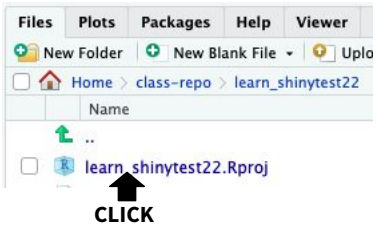


I just really like this figure

Exercise 1: Reproducible Project Setup

Reproducing the developer's environment:

1. Open `learn_shinytest2` and start the RStudio Project



2. Restore the project library:

```
library(renv)
renv::restore()
```

3. Open `app.R`, run App!

Versioning

4. Run code in `git_setup.R`

RStudio

Create manifest and publish using git

Trevor

2. Testing

RStudio

2. Establish testing for your Shiny app

Trevor

Why automated testing?

- Your app can be broken with package changes, code updates, changes to external data sources.
- Manual testing may be time consuming, difficult to do consistently or comprehensively.
- Automating tests can ensure you find out about problems quickly.
- Your app is important!

RStudio

shinytest2

Streamlined toolkit for unit testing Shiny applications and seamlessly integrating with the testthat framework.

2 methods for creating tests:

- Interactively with record_test()
- Programmatically with use_shinytest2_test()

Run tests with run_app()

RStudio

You can create tests interactively and programmatically. This relies on the testthat framework.

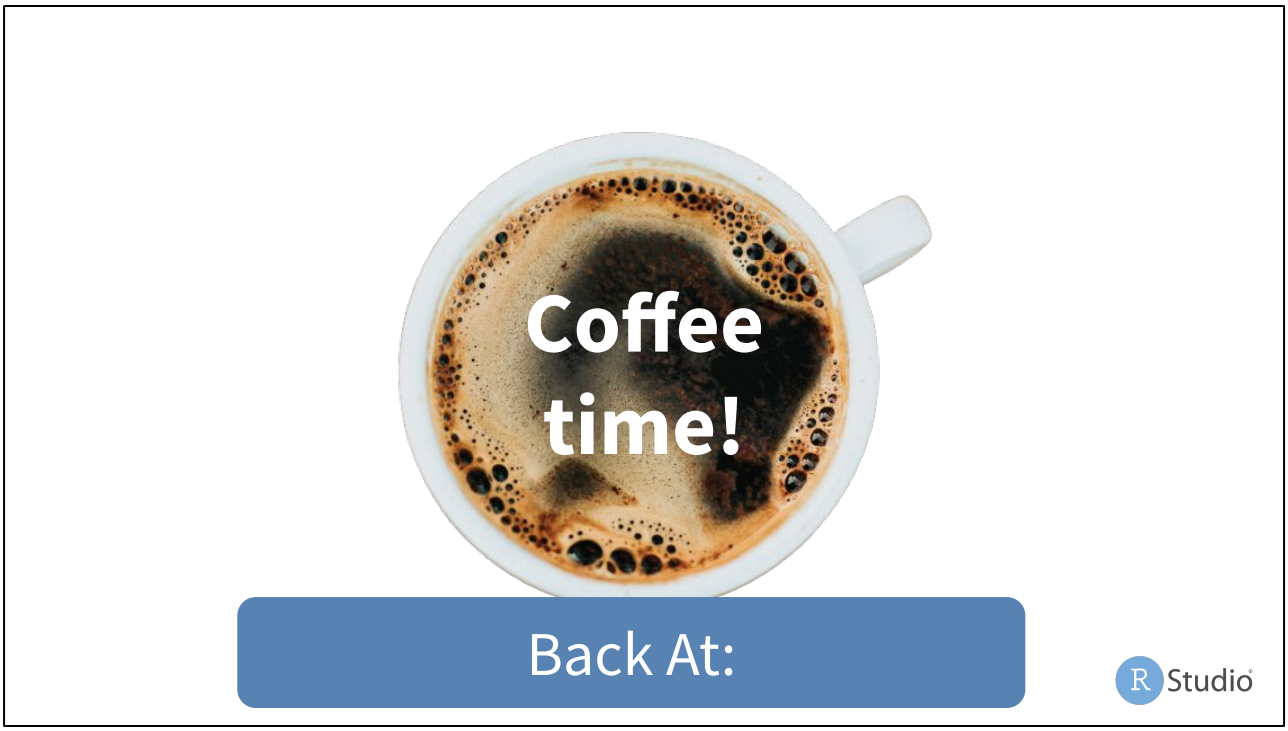
Exercise 2 (a): Interactive

1. Load `shinytest2`:
   ```
   library(shinytest2)
   ```
2. To create a new test run `record_test()`
3. Interact with your app, setting inputs and recording expected outputs.
4. Save test and exit.
5. Make changes and run test with `test_app()`

RStudio

Getting started type steps. Plan will be 1) basic demo of shinytest2 on the app. 2) participants practice by creating a simple test, making change, testing again 3) we demo more complex / prebuilt shinytest that will be in gha

Ideas:
In code bundle actually have some tests commented out. Users will run it and see that it runs. More advanced ones will be commented out, then have them run it again.



Next up is: Putting an application into Production - Shannon

Photo by Nathan Dumlao on Unsplash

Exercise 2 (b): Programmatic

1. Uncomment the sample tests in tests/testthat/test-shinytest2.R
2. Run tests with `test_app()`
3. Look at the .json and .png files in the _snaps directory to see the expectations based on app$expected_values()

RStudio

Getting started type steps. Plan will be 1) basic demo of shinytest2 on the app. 2) participants practice by creating a simple test, making change, testing again 3) we demo more complex / prebuilt shinytest that will be in gha

Ideas:
In code bundle actually have some tests commented out. Users will run it and see that it runs. More advanced ones will be commented out, then have them run it again.

Use a graphical viewer for highlighting differences (install.packages("diffviewer")then testthat::snapshot_review("shinytest2"))

3. Continuous Integration / Continuous Deployment (CI/CD)

RStudio

3. Deploy to Connect with CI/CD*

What are CI/CD & GitHub Actions?

CI/CD is a method to frequently deliver apps to the end users by introducing automation into the stages of app development.

GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline. You can create workflows that build and test every pull request to your repository, or deploy merged pull requests to production.

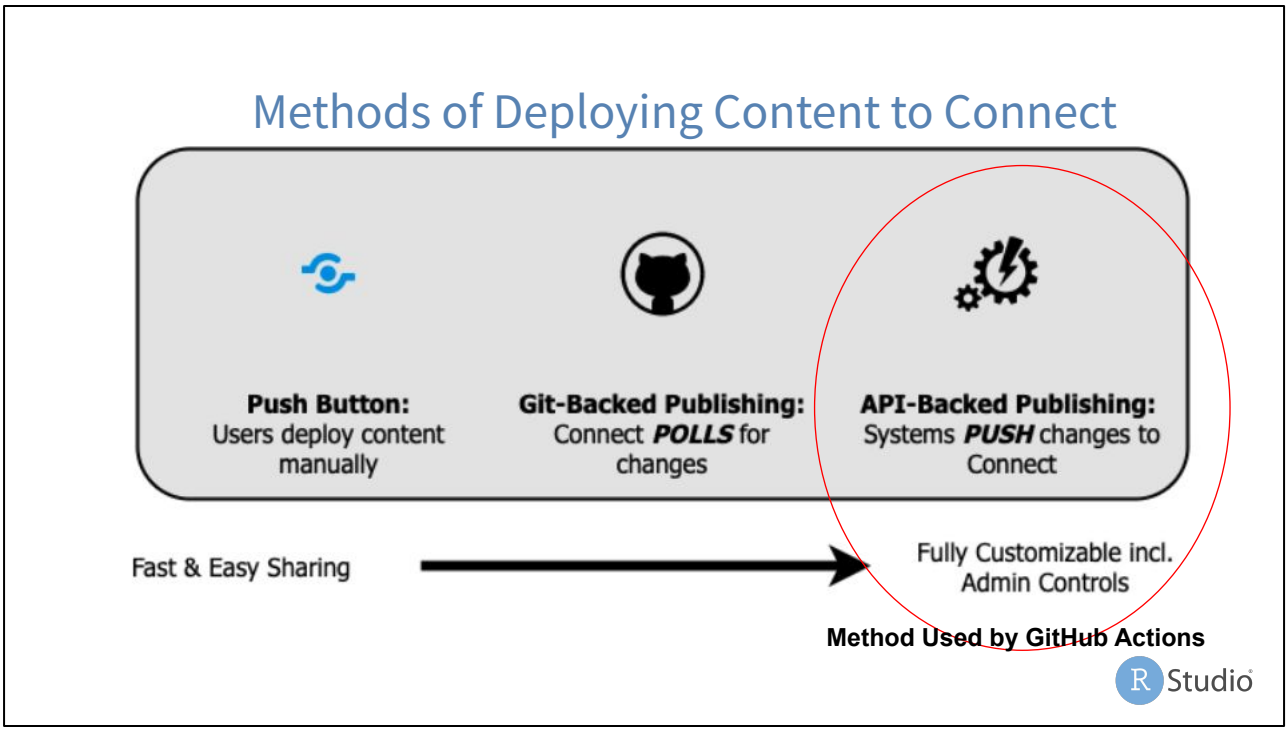
- For more information see: https://solutions.rstudio.com/data-science-admin/deploy/github-actions/

Our CI/CD pipeline = RStudio Connect API + GitHub Actions*

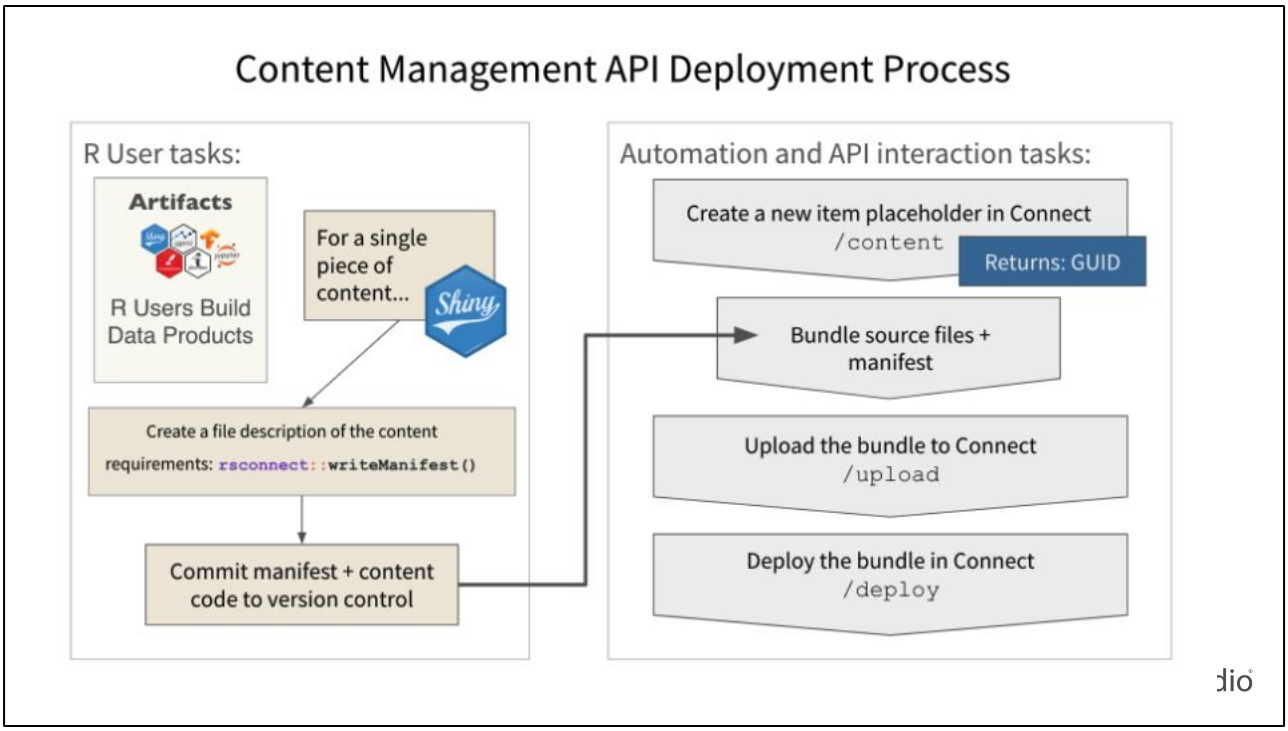
*Could also be done with other CI Services (e.g. Azure DevOps, Jenkins, etc.)

RStudio

This slide might need to be split into multiple slides


Methods of Deploying Content to Connect
Push Button:
Users deploy content manually
Git-Backed Publishing:
Connect POLLS for changes
API-Backed Publishing:
Systems PUSH changes to Connect
Fast & Easy Sharing
Fully Customizable incl. Admin Controls
Method Used by GitHub Actions
RStudio

What is CI/CD



What actually happens when you deploy to connect?

Exercise 3: Basic GitHub Action Deployment (Setup)

`Setup GitHub Credentials, YAML Template and Connect manifest`

1. Run the commands in `connect_publish_action.R`
2. Update the script to have your Connect Server URL
3. Add your Connect API key to your github repo secrets
4. Name the secret: CONNECT_API_KEY
5. Go to Actions and rerun last commit

RStudio

Probably will need to be split into at least two slides
Git secrets and Connect API
Recipe yaml setup

4. Putting it all together

RStudio

3. Deploy to Connect with CI/CD*

Exercise 4: Putting it all together (Add Dependency)

Edit `.github/workflows/connect-publish.yaml`

1. Add an additional job `test-app`, the details of which will go in `.github/workflows/test-actions.yaml` (created next page)

2. Add `needs: [test-app]` in the `connect-publish` job so `test-app` is run first. Otherwise by default both jobs would be run in parallel.

```yaml
name: connect-publish
on:
  push:
    branches: [master]

jobs:
  test-app:
    uses: ./.github/workflows/test-actions.yaml
  connect-publish:
    name: connect-publish
    needs: [test-app]
    runs-on: macOS-latest
    steps:
      - uses: actions/checkout@v2
      - name: Publish Connect content
        uses: rstudio/actions/connect-publish@main
        with:
          url: connect_url_here.com
          api-key: ${{ secrets.CONNECT_API_KEY }}
          access-type: acl
          dir: |
            .:/shiny-app-url
```

RStudio

Exercise 4: Putting it all together (Create Testing Job)

Create `.github/workflows/test-actions.yaml`

1. `on:` is set to `pull_request` only, so this job will run anytime there is a pull request.
2. `workflow_call:` is included, this allows `test-app` to be referenceable by `.github/workflows/connect-publish.yaml`
3. Pandoc and an R environment using RSPM are setup
4. The renv project is activated and `renv::restore()` is run
5. `shinytest2::test_app()` is run

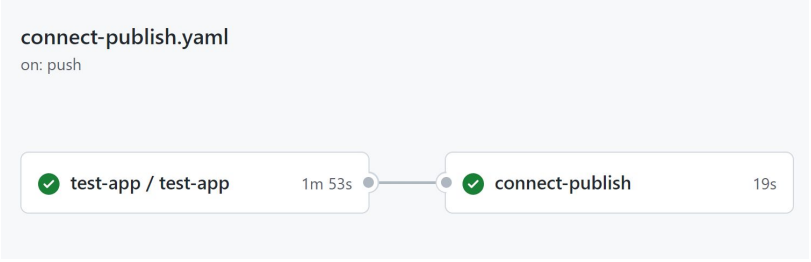
```yaml
on:
  pull_request:  1
    branches: [master]
  workflow_call:  2

name: test-app

jobs:
  test-app:
    name: test-app
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: r-lib/actions/setup-pandoc@v2
      - uses: r-lib/actions/setup-r@v2  3
        with:
          r-version: release
          use-public-rspm: true

      - name: Remove `.Rprofile`
        shell: bash
        run: |
          rm .Rprofile
      - uses: r-lib/actions/setup-renv@v2  4

      - name: Test app
        uses: rstudio/shinytest2/actions/test-app@main  5
```

RStudio

Exercise 4: Putting it all together (Github Action)

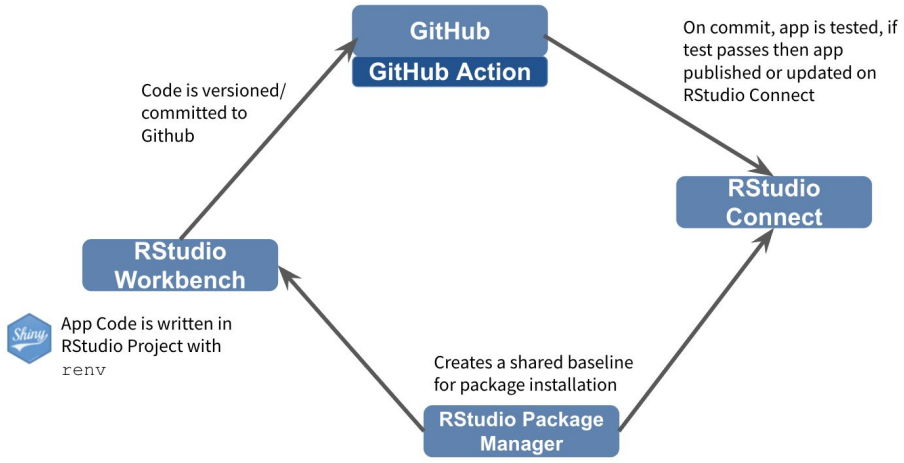
Test the GitHub Action by committing and pushing to GitHub

From the GitHub Actions UI a diagram like the one below will show the Actions and their progress.



RStudio

Summary



RStudio

Workshop ends

Thanks everyone!
Please share your feedback at:
https://rstd.io/r_admin_feedback
https://rstd.io/conf-workshop-survey

RStudio

Thanks
But also, please help us improve this workshop by filling in the feedback request. This one is specifically for us to work on improving the course, but you'll get another one in your email from RStudio and we'd really appreciate you filling in that one when it arrives too!