--------------------------

Thank you, Peter.

Hello everyone.

---------------------------

So, a bit of presentation before we get going with the main topic.

I’m Bruno.

I was first introduced to R on Emacs during my actuarial studies.

I did not touch the language again until 2014, when a colleague showed me RStudio.

What a game changer.

Combine that with the availability of packages like data.table and shiny and I could now replace a bunch of proprietary softwares with a single ecosystem.

So, this is what I’ve been doing for the past 7 years.

---------------------------

Everything I will talk about today should be reviewed in the context of what you are trying to achieve.

Before you start making decisions, you should do a basic analysis of what your code will be used for.

Let’s call this the Availability / Scale / Impact analysis. ASI for short.

Availability means how much downtime can you afford when something goes wrong with your code. The more availability required the more reliable your code must be. It also means more testing and more engineering effort.

Scale relates to the resources needed for your code to do its job. It’s not the same to setup a dashboard for a couple executives than to provide a public dashboard for tweet glocalization tracking worldwide that gets a million plus visits a day. The larger the scale, the more complex the problem becomes. Things like cache, load balancing, redundancy must be considered when writing the code.

Impact is simply an assessment of the number of people that will end up relying on your code. If it is only yourself or your company, you can cut yourself some slack in object naming decision or function call definition. If you are writing a library like curl, you cannot make breaking changes without affecting a very large number of other packages.

So low availability, low scale, low impact, just make the code work and do not stress too much about it. If any of those metrics increase, I strongly advice you follow best practices from the R community.

---------------------------

What follows are those best practices I recommend for an average ASI.

Use a version control system like git, hosted somewhere available to all interested parties.

Think github, gitlab, bitbucket. Learn the workflow and how to use git.

It is a lot easier to track changes and who made them this way.

Some platforms let you execute actions triggered by the change made to the code. It simplifies your integration and deployment pipeline when you are more mature with these tools.

Package code to make maintenance easier for yourself and others.

It might take you a bit of time to learn how to create an R package but trust me, the benefits are worth it.

You get :

* A somewhat rigid file structure, share across all R packages
* Package dependencies management
* Integrated unit test
* You can run the same checks that CRAN uses, see 
* It forces you to document your code
* It makes your code easily sharable

It’s the single best decision you can make on your path to reliability.

Test your code with testthat usethis::use\_testthat()

You are doing it anyway when you are writing functions, just create unit tests as part of your development process. Try different inputs, keep a trace of your experimentation.

Learn how to debug your code and log meaningful information that will help you understand failure in production.

For debugging, I use a combination of debug() or browser() if I want to stop at a particular point during execution. recover() to navigate through the call stack and trace() with edit=TRUE to hot swap the code in a function, even those from other packages.

Another big part is how you should write your code. Let’s get interactive.

Use a style guide. A style guide is some convention on how code should be written, indentation, naming, that kind of stuff. I use the tidyverse one. The more code decision you can push to a common reference, the more time you have to make your code do what you it to do. There is a package named lintr integrated in R to auto indent code.

For next tricks, I think it’s best if I switch to RStudio

1. Microbench for performance critical code.

2. Use argument names in function call with more than one argument.

3. Subset with [[ instead of $.

4. Replace is.na + == with %in% when operating on data frame columns.

5. Use an environment to store global variables.

6. Check for NA, NULL, length one all at once with isTRUE.

7. Consider any and all on logical vectors.

8. Know the difference between &/| and &&/||.

9. You can call names<- and other assignment function directly.

10. Substitute library with a quiet requireNamespace.

11. Delete temporary files using on.exit expression.

12. Exploit attributes with attr and attributes.

13. :: and ::: let’s you explore the objects in a package

Try to measure how much resources your code will use once deployed. Every library you load increase the amount of memory needed to run your code. I use either the task manager on Windows, top on linux to monitor memory and compute usage. There is more to this topic but learning about the interaction between software and hardware will lead you on the right path.

The three most common pattern of R deployment are

Interactive web app (shiny)

REST API (plumber, opencpu, RestRserve)

Scheduled tasks (cronr, taskscheduleR)

They all present different challenges and the selection of a deployment host should be based on which pattern you are using.

This is getting into engineering stuff. If you are interested in this stuff, shoot me question later.

Here is a checklist for code in production.

Version control

R Package

Unit tests

Code style

Pass checks

Performance requirement

Resources measurement

Deployment strategy

Load testing

And here a list of references to help you on your way of putting more R code in production.

Using git with R

happygitwithr.com

R packages

r-pkgs.org

Tidyverse style guide

https://style.tidyverse.org/

R debugging and setup up your dev environment

<https://www.youtube.com/watch?v=r7oBeEyN2jQ>

<https://rstats.wtf/>

More and more industries are realizing the benefits of using tools built around R and I’m there to support them during their transition.

Feel free to contact me at bruno@boostao.ca or through Linkedin if you have any questions.

Thanks for your time.