UNIVERSIDADE DE LISBOA
Faculdade de Medicina

Thesis Title

Thesis Subtitle

Nuno Daniel Saraiva Agostinho

Orientador: Prof. Doutor Nuno Luís Barbosa Morais

Documento provisório
Tese especialmente elaborada para obtenção do grau de Doutor em
Ciências Biomédicas, Ramo da Biologia Computacional

2021

Contents

Resumo

Summary

Abstract goes here

Acknowledgements

List of Figures

List of Tables

Chapter 1

Introduction

This is the story of my PhD, a personal journey like no other I have faced before. For you to follow my story, we first have to rewind back to a few years ago. Billions and billions of years ago.

Once upon a time there was a violent, harsh and unwelcoming planet among countless others. Earth was lifeless. But as millions of years went by, it started being home to a complex recipe whose special sauce is still kept secret: the primordial soup. These were the perfect conditions for a young, 500-million-year-old planet to brew life.

And what is life? Although this question is not easy to answer, living organisms as we know them are complex, carbon-based systems composed of nucleic acids, proteins, carbohydrates and lipids. Together with some smaller molecules, these molecules are known as biomolecules (biological molecules) and are crucial for the survival of living organisms.

Amongst biomolecules, two are of particular importance to my tale/story: proteins and nucleic acids. Proteins have many important functions in an organism, including catalysing chemical reactions (enzymes), signalling cellular processes (hormones) and playing a role in the immune system (antigens), among many others. To generate these proteins, the deoxyribonucleic acid (DNA) stores genetic data, a blueprint required to generate proteins.

But life is so complex. How did this all started from the primordial soup? One possible mechanism for the origin of life is based on the idea of an RNA world, where self-replicating RNA proliferated long before DNA and proteins. RNA can both store genetic information (like DNA) and catalyse life-critical chemical reactions (like proteins). DNA and protein may have appeared later as better suited for storing information and catalysing reactions, respectively, leaving RNA in-between.

1.1 Alternative splicing

1.1.1 What it is?

1.1.2 In disease context

1.2 Transcriptomics

1.2.1 RNA-seq

1.3 Bioinformatic apps

The good, the bad and the ugly.

What is the importance of good user interface/experience?

Reprodutibility

Code optimisation

Benchmarking

Maintained codebase

GitHub

Docker

1.3.1 Making big data accessible

What is missing? Make it easier to access big data.

Chapter 2

Objectives

- psichomics: alternative splicing quantification, analysis and visualisation

- cTRAP: identification of candidate causal perturbations from differential gene expression data

App server

PanAShé

Chapter 3

Materials and Methods

3.1 R programming language

3.1.1 Shiny

Interactive plots via highcharter

3.1.2 Bioconductor

Packages in Bioconductor can only depend on CRAN or Bioconductor packages

Two releases per year

3.2 Datasets

TCGA GTEX SRA/recount2 CMap

Alternative splicing annotation

Alternative splicing quantification

3.3 Data analyses

Gene expression normalisation and filtering

Differential gene expression

Differential alternative splicing

PCA + survival analyses

3.4 Software development

GitHub

3.4.1 Documentation

Function documentation

pkgdown

Vignettes / tutorials

3.4.2 Unit testing

3.4.3 Continuous testing

GitHub Actions

Docker + Docker Hub + Docker Compose

Nextflow

3.4.4 Benchmarking

3.5 Computers

nmorais workstation

Lobito workstation

Lobo - iMM computing cluster

App server (VM inside Lobo)

Chapter 4

psichomics

After finishing the first year of my Masters in Informatics, I was looking for a challenge: I wanted to apply everything I had learned to biology as part of my thesis. However, it was not clear to me how to do it.

While looking for computational biology groups and their projects, I found out about Nuno Morais lab, a research group focused on using computational biology methods to better understand alternative splicing in disease. I wanted to make sure that the project I would be developing would have a strong component in informatics. So I went to personally talk with Nuno Morais about the gaps in the field and how to mitigate them. Nuno immediately mentioned the need for graphical, interactive tools to allow non-experts to analyse and visualise splicing from big datasets. I loved the idea and started exploring ways of going from concept to reality.

After toying with multiple frameworks and programming languages, I decided to stick with the R statistical language and the Shiny web app framework. Shiny allows to develop web apps using R and helped immensely in kick-starting what would be later known as psichomics.

The tool was first made available in 2016 via Bioconductor. When released, the tool was focused on quantifying, analysing and visualising alternative splicing in TCGA. As the time went by, more and more functionality was added. I feel like it took until 2021 for the tool to fully realise its potential: when it finally became available via our app server.

Nowadays, psichomics allows to analyse gene expression and alternative splicing based on user-provided or public transcriptomic data, including The Cancer Genome Atlas (TCGA) (Cancer Genome Atlas Research Network et al., 2013), The Genotype-Tissue Expression (GTEx) project [1] and recount2 [2]. Following an invitation from Springer Methods, I also prepared a book chapter on using psichomics to analyse alternative splicing in stem cell differentiation [3].

Following typical user requests, I added built-in support for analysing non-human data, including new alternative splicing annotations for 14 species (including mouse,

Table 4.1: Major released features of psichomics

Version	Release date	Major features
1.0.0	18 Oct 2016	Alternative splicing quantification and analysis from TCGA data
1.0.8	18 Feb 2017	Load GTEx data
1.4.0	31 Oct 2017	Analyse gene expression data from GTex and TCGA
1.6.1	5 Jul 2018	Load SRA data via recount2 and improved user-owned data
1.12.1	29 Jan 2020	Diagram of alternative splicing events
1.14.2	11 Aug 2020	improved support for loading more data formats, including VAST-TOOLS output
1.18.6	4 Oct 2021	Support for ShinyProxy

fruit fly, frog, and Arabidopsis thaliana). These annotations are based on those provided by the alternative splicing quantification tool VAST-TOOLS [4, 5]. Other improvements include visual diagrams for intuitive representation of alternative splicing events and support for loading VAST-TOOLS output tables, thus allowing to analyse intron retention events quantified by VAST-TOOLS.

4.1 Methods article / stem cells

One day, I received an email from an editor of Springer Methods asking me to contribute a chapter for a new edition of a book of protocols. I forwarded the email to Nuno Morais stating that I was almost certain that it was not spam and whether we should accept the offer.

4.2 Docker

4.3 GitHub Actions

4.4 Feedback

It is wonderful to see that the work I put into psichomics is appreciated based on feedback received via GitHub and email. psichomics is still used nowadays based on citations from recent published articles (Ling et al., 2020; Baeza-Centurion et al., 2020; Birladeanu et al., 2021). In the lab, we can also track visitors of psichomics' documentation via Google Analytics to better understand our users (e.g., what pages they visit the most).

Chapter 5

cTRAP: identification of candidate causal perturbations from differential gene expression data

During a 2017 lab retreat in Madeira, we focused our attention to what were the objectives of the lab and what we could provide to the community. One of the ideas seemed easy to do: comparing a custom differential gene expression against a large database of differential expression profiles. The idea was already been discussed in the original paper of CMap and implemented in their online tool at `clue.io`, but there were issues with their implementation.

The Connectivity Map or CMap [6] is a repository of transcriptomic signatures for thousands of genetic (gene overexpression or knockout) and pharmacological perturbations tested in human cancer cell lines. We developed cTRAP (`bioconductor.org/packages/cTRAP`), an R/Bioconductor package to compare user-provided differential gene expression profiles with those from CMap, allowing to infer putative candidate molecular causes for the observed differences, as well as compounds that may promote or revert them. The comparisons are made based on correlation and gene set enrichment [7] approaches.

The associated manuscript (of which I am a co-first and co-corresponding author) is in preparation for submission to an international peer-reviewed scientific journal.

5.1 Background

5.2 cTRAP analyses

From a vector of user-provided differential expression results (e.g. t-statistic values) with respective gene symbols, cTRAP can return a ranked list of similar CMap perturbations or predict targeting drugs. Moreover, cTRAP can also analyse the enrichment of drug sets in a ranked vector of compounds to identify common compound characteristics.

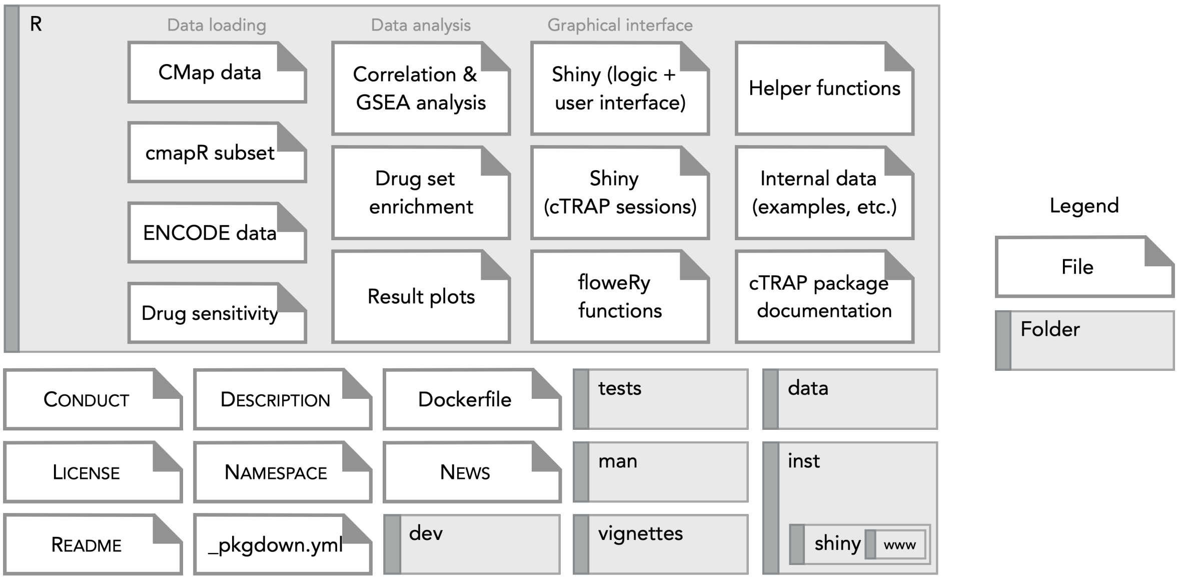


Figure 5.1: Visual representation of cTRAP's file structure. As usual in an R package, the R folder contains the scripts with package functions and data. `dev` is a non-standard folder for an R package and is only used to store supporting scripts related with cTRAP (e.g. test analyses and benchmarks); contents of the `dev` folder are not included when building the R package.

5.2.1 Ranking of similar CMap perturbations

CMap is a repository of transcriptomic signatures of thousands of genetic and pharmacological perturbations in human cancer cell lines. These perturbations can be categorised into gene knockdown, gene over-expression and compounds. Available perturbation types and respective conditions can be enquired in cTRAP with the function `getCMapConditions()`, which will download and load CMap perturbation information into R. Afterwards, the function `filterCMapMetadata()` can be used to download and filter the metadata related to the data to use in downstream analyses based on perturbations types, cell lines, dosages and time points. This information is passed to `prepareCMapPerturbations()` to download CMap differential expression profiles z-scores (GCTX file) and gene and compound information and load the filtered data only. Note that the GCTX file size is 21GB and we recommend to download it directly from GEO GSE92742's Level 5 data link (`ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE92nnn/GSE92742/suppl/GSE92742_Broad_LINCS_Level5_COMPZ.MODZ_n473647x12328.gctx.gz`).

After comparing differential expression z-scores from select CMap perturbations against user-provided differential expression results, `rankSimilarPerturbations()` returns a table with ranked CMap perturbations and their respective correlation coefficients and GSEA scores. Lower ranks indicate perturbations whose differential expression profiles are more similar to the user-provided data, i.e. CMap perturbations that potentially mimic the user-provided transcriptomic changes, whereas higher ranks define perturbations that may revert those changes.

To rank CMap perturbations, cTRAP performs Spearman's and Pearson's correlations

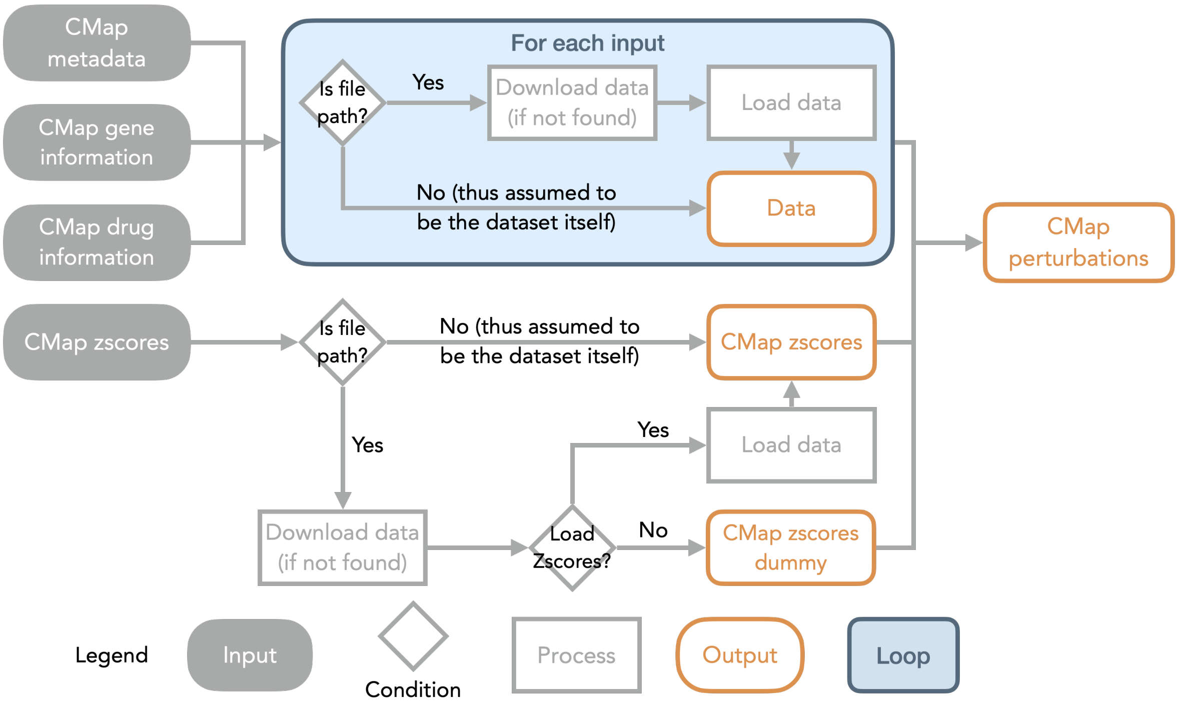


Figure 5.2: Loading data from CMap perturbations.

between the user-provided statistics for differential expression and values from CMap perturbations, and calculates a GSEA-based score (all three methods are run by default). For each method, the similarity scores are averaged across multiple cell lines (when available) and the averages are then used to rank CMap perturbations. By default, results for individual cell lines are provided for informative purposes (e.g. to check the heterogeneity of response across cell lines) but not used when ranking. The different ranking scores are combined into one final rank product, finally used to rank the CMap perturbations.

The GSEA-based score is calculated via the following steps:

1. Sort genes from the user-provided differential expression statistics;
2. Define the top 150 (by default) and bottom 150 (by default) genes as two sets
3. For each CMap perturbation, sort genes by their differential expression z-scores and calculate the Weighted Connectivity Score (WTCS) (1) based on the GSEA enrichment scores for the two sets.

As an example, for a CMap perturbation with a similar differential expression profile to user's input, we expect to find higher enrichment of the top gene set in the most up-regulated genes and higher enrichment of the bottom gene set in the most down-regulated genes.

To minimise RAM usage, `prepareCMapPerturbations()` downloads the CMap's perturbation differential expression z-scores GCTX file (if not previously downloaded) and returns its path without loading the file contents. `rankSimilarPerturbations()` then loads a chunk of 1GB or lower from the GCTX file, compares the differential expression z-scores from that chunk against user-provided data and proceeds to loading and comparing the z-scores from the next chunk.

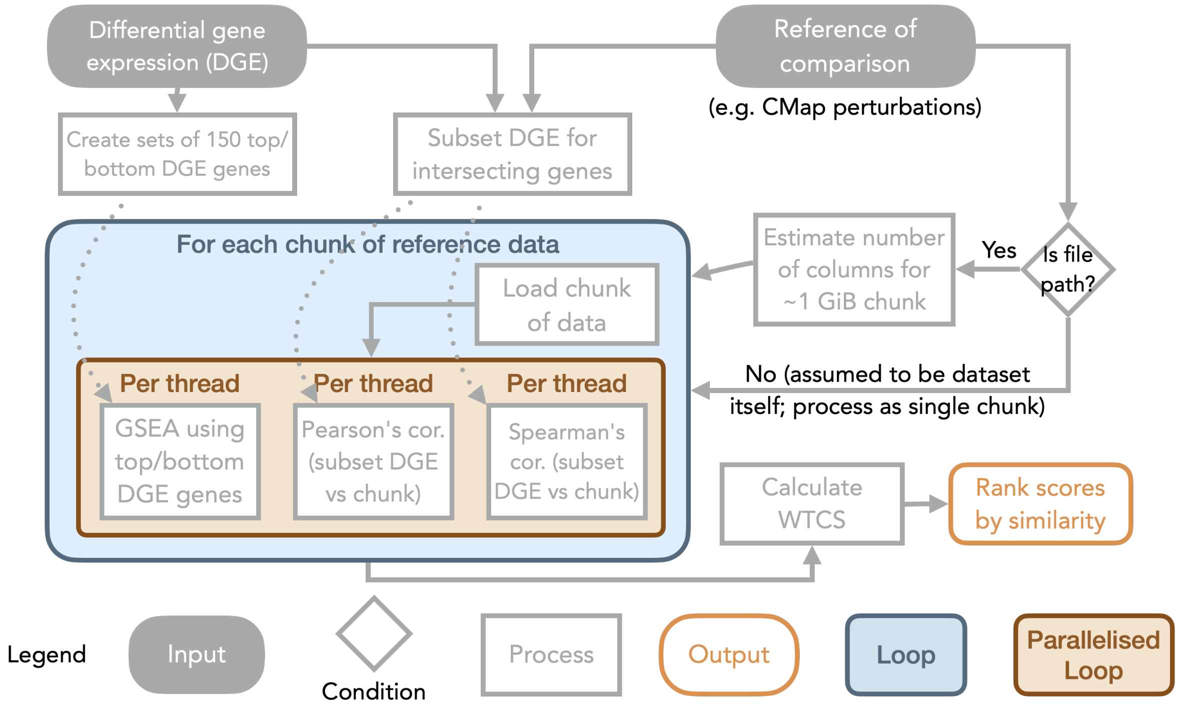

Differential gene expression (DGE)
Reference of comparison
(e.g. CMap perturbations)
Create sets of 150 top/ bottom DGE genes
Subset DGE for intersecting genes
For each chunk of reference data
Load chunk of data
Estimate number of columns for ~1 GiB chunk
Yes
Is file path?
No (assumed to be dataset itself; process as single chunk)
Per thread
GSEA using top/bottom DGE genes
Per thread
Pearson's cor. (subset DGE vs chunk)
Per thread
Spearman's cor. (subset DGE vs chunk)
Calculate WTCS
Rank scores by similarity
Legend
Input
Condition
Process
Output
Loop
Parallelised Loop

Figure 5.3: cTRAP analysis workflow

The ranked list from `rankSimilarPerturbations()` can be plotted using `plot()`, showing a list of all results ordered by a given score or either a scatterplot or GSEA plot for a predicted targeting drug.

5.2.2 Prediction of targeting drugs

Gene expression and drug activity data across multiple cell lines are available from NCI-60 [8], Cancer Therapeutics Response Portal (CTRP) 2.1 [9] and Genomics of Drug Sensitivity in Cancer (GDSC) 7 [10]. For each source, the internal function `prepareExpressionDrugSensitivityAssociation()` performs the following steps:

1. Download all the necessary data depending on given source;
2. Perform Spearman's correlation (by default) between the expression of each gene against the sensitivity of intersecting cell lines to each drug;
3. Generate a matrix with the correlation coefficients per gene and drug; and
4. Prepare metadata for downstream analyses, including gene, compound and cell line information from each source.

A higher correlation coefficient for a given gene and drug suggests a gene whose higher expression is associated with higher drug sensitivity across multiple cell lines. As this process can take multiple hours to finish for all sources, the resulting objects were stored online for each aforementioned source and can be listed with `listExpressionDrugSensitivityAssociation()` and downloaded and loaded into R using `loadExpressionDrugSensitivityAssociation()`.

To identify compounds that could target the phenotype associated with user-provided differential expression profiles, we use `predictTargetingDrugs()` with user-provided differential expression results and a correlation matrix of gene expression and drug sensitivity as input. The correlation coefficients between gene expression and drug sensitivity for each drug are compared against user-provided differential expression results by Spearman's and Pearson's correlation and GSEA-based scores (as performed when ranking CMap perturbations, results from comparison methods are ranked and then those rankings are finally used to calculate the rank product's rank). `predictTargetingDrugs()` returns a table with ranked predicted targeting drugs and their respective correlation coefficients and GSEA scores. A lower rank comprise drugs that may target phenotypes similar to the user-provided differential expression profile.

The resulting object can be plotted with `plot()`, showing a list of all results ordered by a given score or either a scatterplot or GSEA plot for a predicted targeting drug.

To compare the results from predicted targeting drugs and CMap perturbations that may mimic or revert the observed phenotype, we can use the function `plotTargetingDrugsVSsimilarPerturbations()`. For the available compound identifiers in the metadata pertaining from the different datasets (e.g. compound name, Broad ID, PubChem CID and SMILES), the function will automatically select the identifiers with higher number of matching values between the two datasets, unless the identifiers are defined. A scatterplot is then plotted using, by default, the rank product's rank of targeting drugs in one axis and the rank product's rank of similar perturbations in the other.

5.2.3 Drug descriptor set enrichment analysis

We computed drug descriptors (e.g. molecular weight and number of aromatic rings) for compounds from CMap and NCI-60. The dimensionality of these descriptors can either be 3D (i.e. descriptors calculated based on three-dimensional compound characteristics) or 2D. These datasets are downloaded and loaded into R using `loadDrugDescriptors()`.

Next, we created sets of descriptors via `prepareDrugSets()`. By default, the function creates a maximum of 15 sets per drug descriptor. For each alphanumeric descriptor, one set is created per unique value of that descriptor. Alphanumeric descriptors containing more than 15 unique values (by default) will be discarded. For numerical descriptors, `prepareDrugSets()` internally uses the `binr::bins()` function to create evenly-distributed bins of drug descriptors, where each set contains a minimum number of points equal to the number of non-missing values divided by the number of maximum sets (15 by default) divided by a constant (5 by default).

By using `analyseDrugSetEnrichment()`, we analysed the enrichment of the created drug descriptor sets in a named numeric vector or an object returned from `rankSimilarPerturbations()` – only if run against CMap compound perturbations – or `predictTargetingDrugs()`. The enrichment analysis is internally performed based on GSEA using `fgsea::fgsea()`.

The resulting object can be plotted with `plot()`, showing a list of all results ordered by a given score or either a scatterplot or GSEA plot for a predicted targeting drug.

5.3 Benchmarking + code/memory optimisation

We measured elapsed time using R's `Sys.time()` immediately before and after ranking similar CMap perturbations, predicting targeting drugs (using NCI60 expression and drug sensitivity association, the most time-consuming option) and performing drug set enrichment analysis using cTRAP 1.8.1 (296f9b21). As input, we used the t-statistics for the differential expression between EIF4G1 knockdown versus control based on ENCODE gene expression data from cell line HepG2 (the diffExprStat object in the cTRAP package; running `?cTRAP::diffExprStat` shows the R commands to obtain this object).

We measured the heap memory usage of cTRAP 1.8.1 (296f9b21) along time by running R in debug mode with the heaptrack 1.0.0 profiler. heaptrack tracks and logs all calls to the core memory allocation functions via `LD_PRELOAD` and respective backtraces. For R to work properly with heaptrack, the file `/usr/bin/R` was edited: all lines of the last *if* statement were commented out with the exception of

```
exec ${debugger} ${debugger_args} "${R_binary}" ${args} "${@}"
```

Afterwards, we benchmarked R scripts with:

```
R -d heaptrack -f ${Rscript} --args ${Rscript_args}
```

All benchmarks were run in a workstation running Ubuntu 18.04.5 LTS with 768 GB of RAM memory and 72 cores (Intel Xeon Gold 6254 CPU @ 3.10GHz).

5.4 Graphical interface

The most recent feature of cTRAP is its visual interface that allows users to interactively perform most features of cTRAP via the web browser. The graphical interface was modularly built and as an experiment that combines using explicit R commands with an helpful graphical interface. Simply put, there are 5 interface functions:

- `launchDiffExprLoader()` to load differential expression data. Returns a differential expression object that can be used in cTRAP analyses.
- `launchCMapDataLoader()` to explore and load CMap data by type of perturbation, cell types, time points and dosages. Returns filtered CMap data based on the user's selection.
- `launchMetadataViewer()` to check metadata of given cTRAP objects.
- `launchResultPlotter()` to view and plot cTRAP results given as input.

- `launchDrugSetEnrichmentAnalyser()` to analyse drug set enrichment and visualize respective results.

Like usual R functions, these graphical interfaces functions accept input and may return output and can thus be intertwined with R code, allowing to easily reproduce cTRAP analyses. For instance:

```r
# Launch differential expression loading interface to select knockdown
# data from ENCODE (pre-filtered for HepG2 cell line and EIF4G1 gene)
diffExpr <- launchDiffExprLoader(cellLine="HepG2", gene="EIF4G1")

# This command does the following:
# 1. Download ENCODE's HepG2 data for EIF4G1 knockdown and controls
# 2. Perform DGE between EIF4G1 knockdown vs. control
# 3. Return resulting t-statistics by gene

# Load CMap knockdown data in HepG2
cmapKD <- launchCMapDataLoader(
    cellLine="HepG2",
    perturbationType="Consensus signature from shRNAs targeting the same gene")
# Load CMap compound data in HepG2
cmapCompounds <- launchCMapDataLoader(cellLine="HepG2",
                                      perturbationType="Compound")
# Load all CMap data in HepG2
cmapPerts <- launchCMapDataLoader(cellLine="HepG2")

# View metadata of all resulting CMap data objects
launchMetadataViewer(cmapKD, cmapCompounds, cmapPerts)

# Rank similar perturbations ------------------------------------------
compareKD        <- rankSimilarPerturbations(diffExpr, cmapKD)
compareCompounds <- rankSimilarPerturbations(diffExpr, cmapCompounds)
comparePerts     <- rankSimilarPerturbations(diffExpr, cmapPerts)



launchResultPlotter(compareCompounds, compareKD, comparePerts)

# Predict targeting drugs ---------------------------------------------
listExpressionDrugSensitivityAssociation()
assocMatrix <- listExpressionDrugSensitivityAssociation()[[1]]
assoc       <- loadExpressionDrugSensitivityAssociation(assocMatrix)
predicted   <- predictTargetingDrugs(diffExpr, assoc)
launchResultPlotter(predicted)

# Plot targeting drugs vs similar perturbations -----------------------
launchResultPlotter(predicted, compareCompounds)
```

```r
# Analyse drug set enrichment -------------------------------------------
descriptors <- loadDrugDescriptors("NCI60", "3D")
drugSets    <- prepareDrugSets(descriptors)

launchDrugSetEnrichmentAnalyser(drugSets, compareCompounds)
launchDrugSetEnrichmentAnalyser(drugSets, predicted)
```

In order to increase its usefulness to the scientific community, we made cTRAP available online[1] with a single interface to provide all the features of the aforementioned functions, as well as perform all analyses, via a sixth interface function: `cTRAP()`. A clear question arrived with such strategy: how to deal with long-running tasks? The way R/Shiny is built, an entire cTRAP section would be kept online and consuming useful resources, but this would not properly scale for multiple users using heavy memory resources simultaneously. To avoid this, long-running tasks should be put in a queue and performed in the background. But this also meant that the users would need get their results back once finished calculating. And thus the idea of using user sessions was born.

5.4.1 User sessions

When visiting cTRAP, a user is greeted with a welcome screen that allows to create a new session or restore a previous one (Figure 5.4). If the user creates a new session, a random string of numbers and letters is created, henceforth denominated as *token*. The token will be the name of the folder storing user data in the web server and thus cTRAP ensures that token is unique and no other folder is currently using it (Figure 5.5).

If the user loads data to their session, a new folder is named after the session token (Figure 5.5). Any changes performed and new datasets appended to the session data are immediately saved to the session folder. The common cTRAP data available across sessions (e.g. the big 21GB CMap perturbations z-scores file) can be made available in a folder accessible to all sessions, thus skipping the step of downloading and preparing the data.[2]

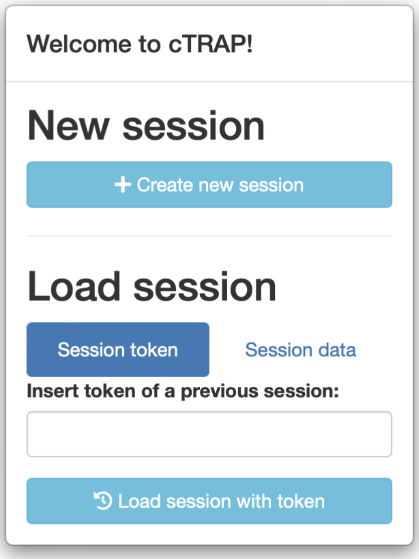


Figure 5.4: Welcome screen.

While using cTRAP, the user can create a new session, load a previous session via a token or a RDS file at any time. When using the token, cTRAP loads the contents of the folder named after the token – if no such folder exists, it will warn the user (Figure 5.5). In case the user uploads a RDS file, cTRAP will create a new session and load the contents of the RDS file as the data session (Figure 5.5). Using an RDS file ensures the user can open the

[1] More information in chapter 6: **CompBio app server**

[2] This is how cTRAP is configured in our web server.

data in an R session in their local computer given that this RDS file is simply a list of all the datasets available in the session or even use this file in a local version of cTRAP. Moreover, as sessions start to accumulate in our server, they may be removed from the system and thus may become inaccessible[3].

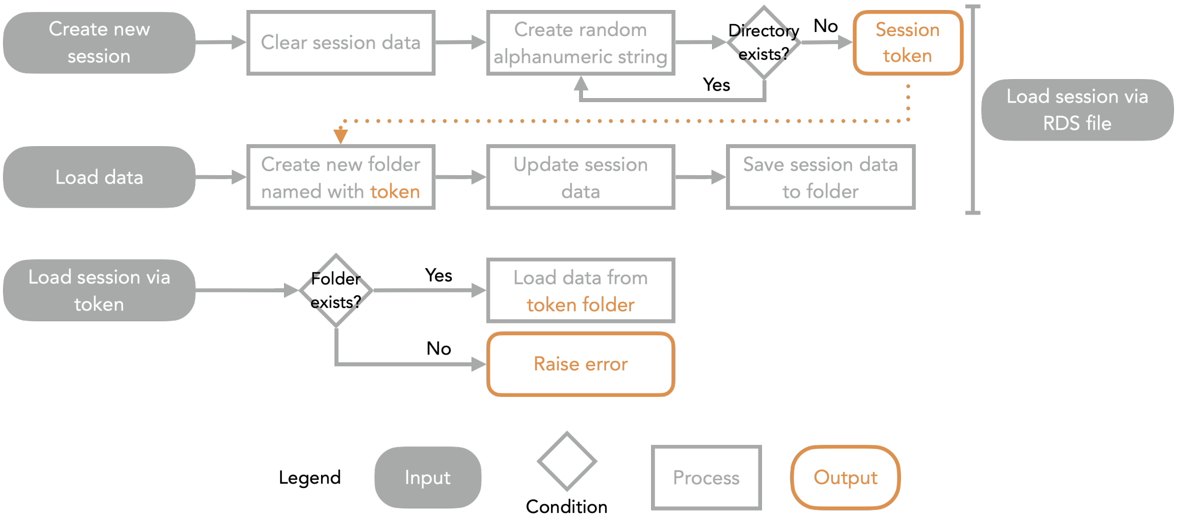


Figure 5.5: User session workflow. cTRAP allows to create new session or load a previous one via token or RDS file.

5.4.2 Background tasks

To run tasks in the background, I decided to use Celery, a task queue manager written in Python, and Flower, a Celery monitoring app that also provides a useful RESTful API to work with Celery. Flower makes it easier to send tasks to Celery via HTTP methods, facilitating the communication between cTRAP and Celery. To make use of Flower in R, I created floweRy, an R package to help create the commands used in the Flower API more easily.

If Celery/Flower support is not available, cTRAP can run tasks in the same R process (as usual in an R/Shiny app) so the user has to wait for the long-running tasks to finish before proceeding with interacting with the app. Another limitation is that if cTRAP times out or is shut down, the running processes will stop.

During the whole process, the user interface shows tables with the status of each job from the user. When the process finishes running, the data is automatically loaded and a notification in cTRAP alerts the user that the data is now loaded and ready to visualise.

[3] We have also considered implementing a 14-day expiration date for user sessions. This is not currently in place.

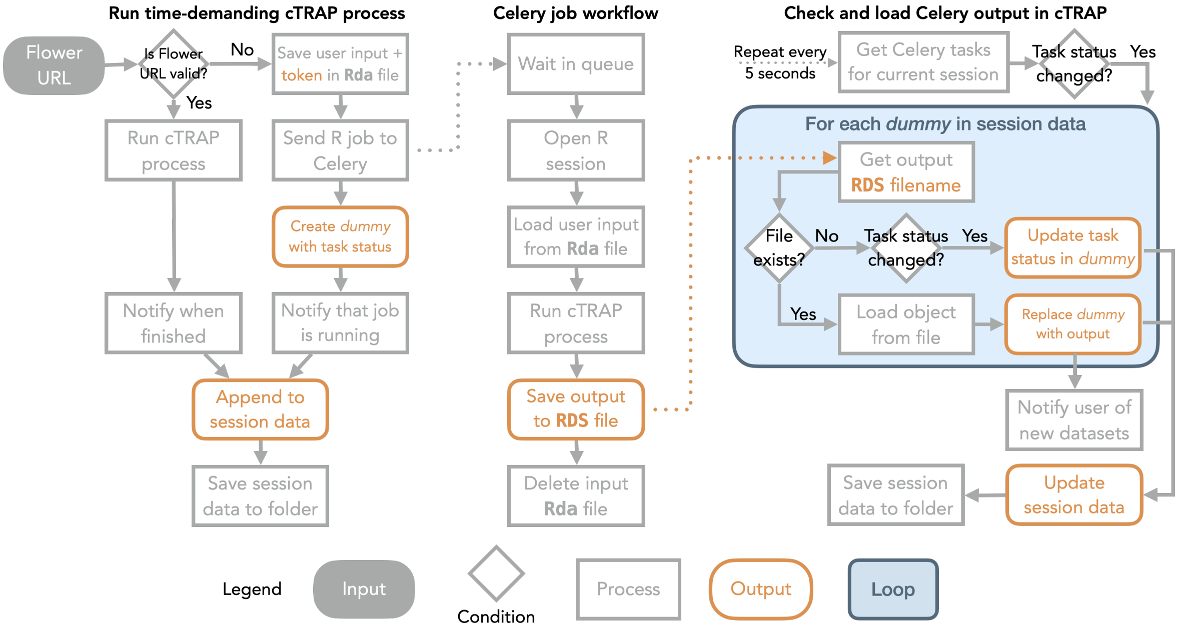


Figure 5.6: cTRAP process running in Celery. Time-demanding cTRAP processes can be run in the background using Celery/Flower. While running in Celery, the output of the cTRAP process is saved to the folder associated with the token of the user's session. When that specific session is active, all finished files are automatically loaded as part of the data session and the user is notified.

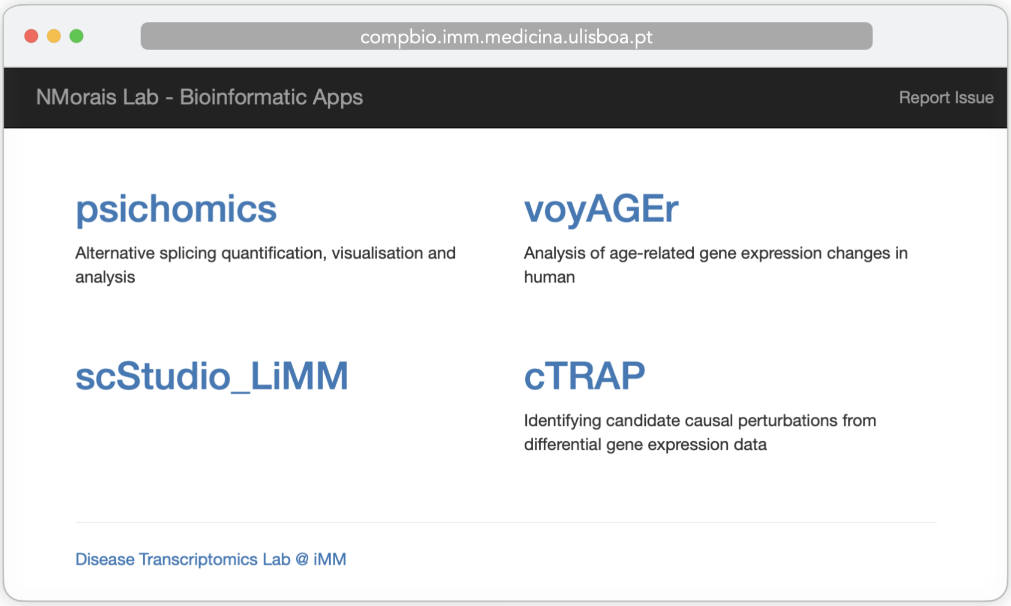
Chapter 6

CompBio app server

Since I started building psichomics, I wanted the program to be publicly available as an online web app, providing users the most up-to-date version at their fingerprints, without having to install, update and manage different versions of R, Bioconductor, psichomics and all their dependencies. Five years after the first Bioconductor release of psichomics in 2016, that vision finally came true.

One of our lab's ambitious goals is to develop interactive visual tools to assist in exploring biological data. These tools need to be intuitive and user-friendly enough to be used by everyone, no matter their computational background. To turn that dream into reality, I set up the CompBio app server, a Linux virtual machine running in iMM computing cluster that hosts psichomics, cTRAP and other Shiny apps from my lab colleagues. The server is accessible at `compbio.imm.medicina.ulisboa.pt` (Figure 6.1) and its code is available at `github.com/nuno-agostinho/compbio-app-server`.

CompBio is built using Docker Compose, a program to manage multiple Docker containers



compbio.imm.medicina.ulisboa.pt

NMorais Lab - Bioinformatic Apps

Report Issue

psichomics

Alternative splicing quantification, visualisation and analysis

voyAGEr

Analysis of age-related gene expression changes in human

scStudio_LiMM

cTRAP

Identifying candidate causal perturbations from differential gene expression data

Disease Transcriptomics Lab @ iMM

Figure 6.1: CompBio's homepage screenshot. List of hosted web apps (11 Nov 2021).

simultaneously, and includes the following services (Figure 6.2):

- **ShinyProxy** to serve web apps in R/Shiny and Python.
- **Nginx** as a reverse proxy, serves as an intermediary between the user requests and the server. Nginx is responsible to return what is shown to the user, to ensure HTTPS traffic is encrypted via SSL certificates, serve publicly available files and show a custom error page if ShinyProxy is not responding (e.g. temporarily down or overloaded).
- **Celery**, **Redis** and **Flower** to run background tasks.[1]
- **Plausible**, **PostgreSQL** and **ClickHouse** for website analytics (i.e. track visitor metrics).
- **Prometheus** and **Grafana** to register and monitor server resources.
- **RStudio Web** to run R sessions and test features (not used in production).

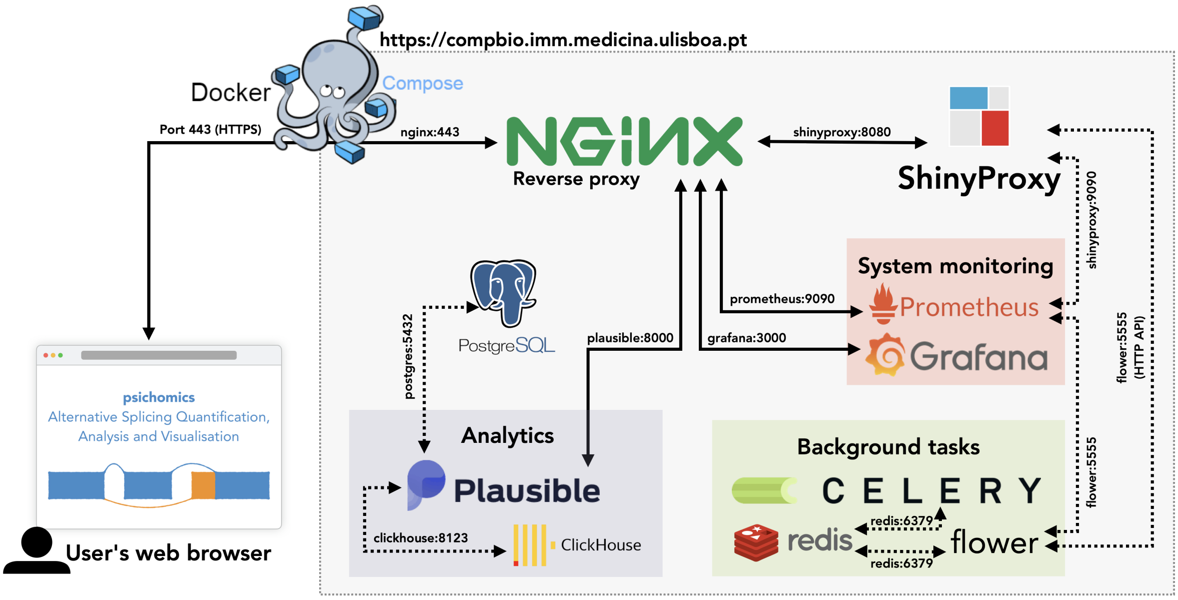


Figure 6.2: App server architecture is based on Docker Compose. All services are provided via Docker images and communicate with each other via a Docker-created network using the name of the service and a specific port (e.g. Nginx communicates with ShinyProxy via `shinyproxy:8080`). The groups (analytics, system monitoring and background tasks) are strictly conceptual.

6.1 Docker Compose

Experimenting different programs while managing their manifold dependencies to develop an healthy web server is like an intricate ballet where all finely-coordinated dancers interplay for an astounding performance. This can be as difficult as it sounds: a wrong move can affect

[1] More information in subsection 5.4.1: **User sessions**

the whole show. After all, each program/dependency has its own requirements and some may be a distress to (un)install. Moreover, when the server is online, errors may arise due to configuration changes (such as new app updates), requiring a fast rollback to minimise server downtime. That can be achieved if the programs are self-contained and modular such as when using Docker. But how to coordinate several Docker containers to beautifully perform the Swan Lake?

With Docker Compose, multiple applications are run isolated from others in their own Docker containers, allowing to easily update or replace them without affecting other system components. All services spawned in Docker Compose are Docker images, either pre-created (e.g. official Docker images from Docker Hub) or built before starting up all services (in this case, a Dockerfile is required to the create the Docker images). Docker Compose allows to quickly play and swap programs: the services present in the server were selected after trying out many other combinations of alternative apps. The modularity of Docker Compose allows to easily test new system components and update software versions.

CompBio can run in any Linux[2] machine with only Docker and Docker Compose installed, thus making the setup easily portable across different computers and requiring minimal user intervention. A single file (`docker-compose.yml`) contains the main configuration of each application in the server, and extra configuration files may be available in the local directory. For organisation purposes, the project is organised by folders named after each service, where each folder stores files (e.g. Dockerfile, configuration and data) associated with the respective application (Figure 6.3).

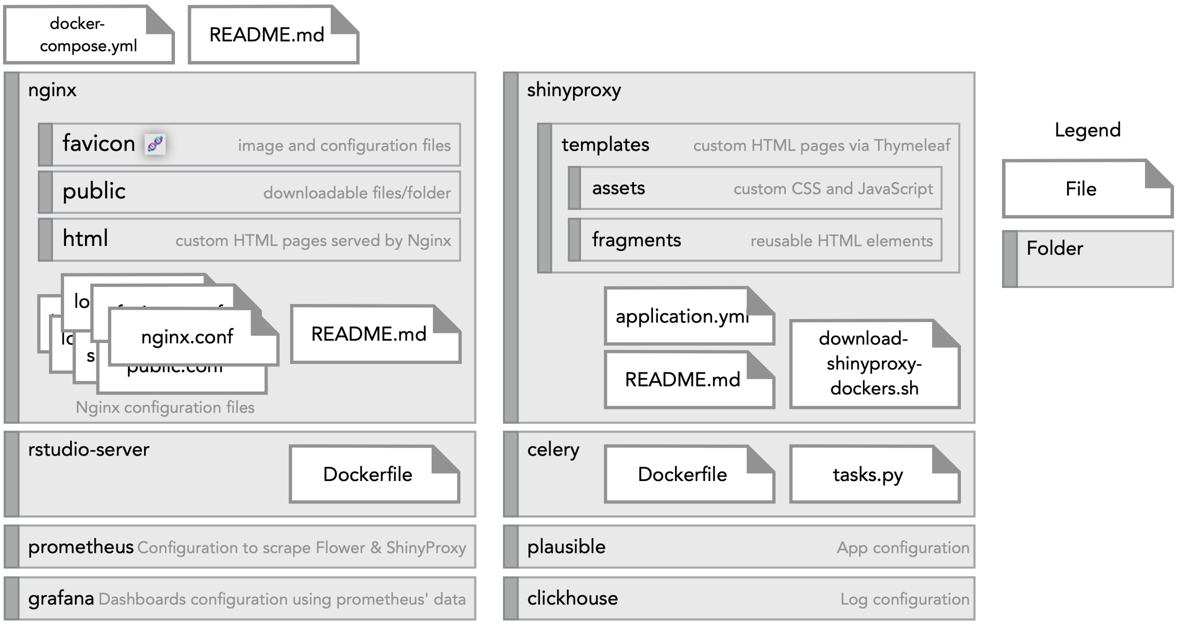

docker-compose.yml
README.md
nginx
favicon
image and configuration files
public
downloadable files/folder
html
custom HTML pages served by Nginx
nginx.conf
README.md
Nginx configuration files
shinyproxy
templates
custom HTML pages via Thymeleaf
assets
custom CSS and JavaScript
fragments
reusable HTML elements
application.yml
README.md
download-shinyproxy-dockers.sh
Legend
File
Folder
rstudio-server
Dockerfile
celery
Dockerfile
tasks.py
prometheus
Configuration to scrape Flower & ShinyProxy
plausible
App configuration
grafana
Dashboards configuration using prometheus' data
clickhouse
Log configuration

Figure 6.3: Visual representation of the file structure of the CompBio app server. Each folder contains files associated with a specific service. Folders `rstudio-server` and `celery` contain Dockerfiles for building custom Docker images of the respective services.

Although data from Docker containers are only available temporary after the container

[2] Some services may require a different configuration to run in other operative systems.

is stopped, important files are preserved in Docker volumes to avoid data loss when restarting services. Docker volumes are mounted when starting the `docker-compose.yml` project. Although data from Docker is temporary, specific directories are mounted in Docker volumes to be stored in the long-term (such as databases).

A single command is enough to build Docker images from Dockerfiles, download Docker images from Docker Hub and start up every service in detached mode[3].

Multiple `docker-compose` commands allow to manage the services. For instance, it is possible to restart single services without affecting other programs. This is specially useful when altering the configuration of a service. However, changes to `docker-compose.yml` are only applied after restarting all services by shutting down all services with `docker-compose down`.

6.2 ShinyProxy

ShinyProxy is an open-source program that deploys R/Shiny and Python apps via Docker. When a user starts an app, ShinyProxy creates a new Docker container exclusively for that user. The containers are automatically terminated 30 minutes (by default) after the last user interaction.

Adding new apps to the system is as simple as pulling the Docker image of the app in the server and adding them to the ShinyProxy configuration file.

6.2.1 Features

ShinyProxy offers multiple built-in features, including:

- **Usage statistics:** many ShinyProxy metrics (including app usage time, app failures and user numbers) are collected with Prometheus and visualised using Grafana.

- **App recovery:** when restarting ShinyProxy, ShinyProxy-initiated Docker containers continue running in the background and are attached once ShinyProxy finishes loading, minimising issues related with server maintenance. The apps will be unavailable while ShinyProxy is not running. For more information, please read `shinyproxy.io/documentation/app-recovery`.

- **User authentication:** authentication with multiple methods, including social login via GitHub, LinkedIn, Google, etc. However, user authentication requires all visitors to login before continuing. As we prefer users to be able to anonymously access our apps, this feature is currently disabled.

- **User sessions:** user data can be stored in user-specific folders. As the sessions are only accessible when the Docker container is already attached to the volumes, this allows for complete isolation from other user folders. However, this feature works best with

[3] `docker-compose up -d --build`

user authentication enabled (otherwise, random identifiers are used for each visitor and requires custom logic to load data between computers).

- **Multiple app instances:** users can open and manage multiple app instances simultaneously (not currently enabled in the app server); for more information, please visit `shinyproxy.io/documentation/ui/#using-multiple-instances-of-an-app`.

6.2.2 Progress bar when loading apps

When ShinyProxy is loading an app, a spinning wheel is usually shown as a loading indicator. For apps that take more than 10 seconds to load (e.g. psichomics and cTRAP), this may give the feeling that the website is not working, as the perceived time taken to load is larger than elapsed time. To avoid that, the spinning wheel was replaced with a progress bar that provides users with a time estimate for app loading (Figure 6.4). The progress bar fills based on time alone.

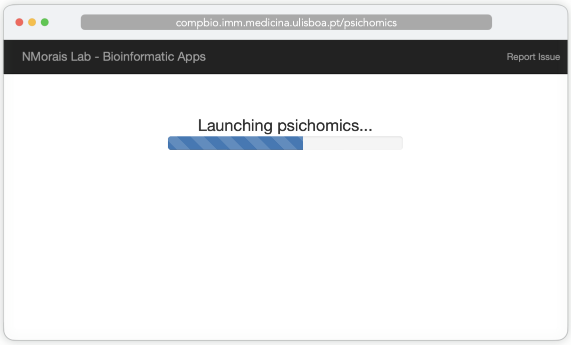

compbio.imm.medicina.ulisboa.pt/psichomics
NMorais Lab - Bioinformatic Apps
Report Issue
Launching psichomics...

Figure 6.4: Progress bar displayed while psichomics loads (11 Nov 2021).

By default, the progress bar takes 5 seconds to fill (as sample Shiny apps take that much to launch in ShinyProxy), but the time is customisable for specific apps by editing a specific app in `shinyproxy/application.yml` and adding a `template-properties.start-up` parameter. For instance, psichomics takes 20 seconds to fully load the progress bar (i.e. `template-properties.start-up: 20s`), whereas cTRAP takes 15 seconds. When the app is loaded (regardless of the progress shown), the progress bar fades out.

6.2.3 Custom HTML pages

6.3 Nginx

Nginx is a reverse proxy, i.e. an intermediary that decides what is shown to the user depending on the URL visited.

Nginx is also used for ensuring encrypted HTTPS traffic via SSL certificates. SSL certificates are handed by the IT team at iMM and we only need to point Nginx to the correct location of those certificates. SSL certificates include three separate parts: the site certificate, intermediate certificates, and the private key.

A public folder is available via Nginx.

In case ShinyProxy is down, Nginx will serve a custom error page stating that the server is down probably because of ShinyProxy. This is informative enough to end-users that know they should wait to refresh the page in a moment and also to admins that will understand that ShinyProxy is temporarily down (this can happen because of multiple reasons, such as a restart of the service or overloading).

6.4 Background tasks

6.5 Website analytics

Plausible is an open-source, privacy-focused web analytics tool that collects traffic metrics for multiple websites and provides them via an interactive dashboard. CompBio runs the self-hosted version of Plausible. All of Plausible metrics (e.g., visitor numbers, total page views and session duration) are anonymously aggregated without cookies, thus avoiding individual tracing.

Using the self-hosted version of Plausible guarantees that the user data tracked is done locally in the server. Plausible also protects user privacy by making their data hard to individually trace and by complying with current privacy laws (GDPR, CCPA and PECR). This is in stark contrast with Google Analytics.

6.6 Resource monitoring

Prometheus monitors server resources. Graphana is used to visualise the metrics collected by Prometheus.

6.6.1 Celery

6.6.2 ShinyProxy

6.6.3 Nginx

Nginx requires further configuration to be monitored. Currently, it is not monitored.

6.6.4 System

6.7 Server maintenance

CompBio is a web server that hosts Shiny applications and is publicly accessible by everyone online. This makes our server a target for potential security attacks. In order to mitigate such vulnerabilities, it is crucial to update user-facing programs (Docker, Docker Compose, Nginx and ShinyProxy), while components that are not directly available to end-users should be updated when possible. As updates may contain breaking changes that hamper website functionality, it is recommended to read change logs related to new software versions to pinpoint potential issues before updating.

Updates to Docker and Docker Compose need to be performed by an administrator using Linux's `apt-get` command[4]. Docker images of the server (including Nginx and ShinyProxy), on the other hand, require a user in the `docker` group to pull the latest Docker images

[4] sudo apt-get update && sudo apt-get upgrade

from Docker Hub and edit the versions of the Docker images used in `docker-compose.yml` accordingly. Afterwards, the app server services need to be restart to apply changes[5]. Another advantage of using Docker Compose: if something goes wrong with the updated Docker images, simply revert `docker-compose.yml` to a previous working state and restart all the services.

[5] While inside the project folder: `docker-compose down && docker-compose up -d --build`

Chapter 7

PanAShé

In a collaborative lab effort, we are also developing a Nextflow pipeline to process raw RNA sequencing data from TCGA (Cancer Genome Atlas Research Network et al., 2013) and GTEx (The GTEx Consortium, 2013) in order to provide processed gene expression and alternative splicing data from samples from multiple normal and diseased tissues. The aims of this project extend those of recount2 (Collado-Torres, 2017) and include alternative splicing analysis, as well as a complementary dashboard to help users explore the data in these data sources. We are also considering integrating the data from this project in psichomics in lieu of the limited processed data from the public sources for TCGA and GTEx.

The Nextflow pipeline we are working on is based on Docker images for portability and reproducibility. This means that only Docker and Nextflow are required to be installed in the computer running the pipeline. We intend to write a peer-reviewed article regarding this project, as well as share our scripts and processed data with the scientific community.

Chapter 8

Discussion

Looking back, I fought my biggest opponents of all time during my PhD. Some days were bright as the sun, others were dark as the night. Some days were spent alongside my friends, others alongside my shadow alone. Some were full of victories, others full of self-doubt.

I fought against and defeated many (software) bugs, yet they keep swarming around me. No matter the quality of the code, no matter the amount of unit testing, no matter the time squatting each pesky bug – as long as there is code, there will be bugs.

Another boss I struggled with was time: hard to reach as it never stays still. Deadlines are a motivation to strive for and also a cause of distress. It is easy to let oneself be swallowed by each tic, tac, tic, tac. The only way to deal with time is with self-negotiation, a skill otherwise known as time management. I am still learning how to do it efficiently, while doing my best to have moments for everything each day: moments of work, moments of sleep, moments of food and moments of life.

But the biggest enemy of them all was one I knew too well since birth. It was myself: my fears, my anxieties, my insecurities. To this day, I am still learning how to cope with all of them. I believe that, maybe one day, I will be able to convince myself to finally fight alongside me. I can only hope.

Bibliography

[1] Lonsdale J, Thomas J, Salvatore M, Phillips R, Lo E, Shad S, et al. The Genotype-Tissue Expression (GTEx) project. 2013;45(6):580–585. doi:10.1038/ng.2653.

[2] Collado-Torres L, Nellore A, Kammers K, Ellis SE, Taub MA, Hansen KD, et al. Reproducible RNA-seq analysis using recount2. Nature biotechnology. 2017;35(4):319–321. doi:10.1038/nbt.3838.

[3] Saraiva-Agostinho N, Barbosa-Morais NL. In: Interactive Alternative Splicing Analysis of Human Stem Cells Using psichomics. Springer US; 2020. p. 179–205. Available from: https://doi.org/10.1007/978-1-0716-0301-7_10.

[4] Irimia M, Weatheritt RJ, Ellis JD, Parikshak NN, Gonatopoulos-Pournatzis T, Babor M, et al. A Highly Conserved Program of Neuronal Microexons Is Misregulated in Autistic Brains. 2014;159(7):1511–1523. doi:10.1016/j.cell.2014.11.035.

[5] Tapial J, Ha KCH, Sterne-Weiler T, Gohr A, Braunschweig U, Hermoso-Pulido A, et al. An atlas of alternative splicing profiles and functional associations reveals new regulatory programs and genes that simultaneously express multiple major isoforms. 2017;27(10):1759–1768. doi:10.1101/gr.220962.117.

[6] Subramanian A, Narayan R, Corsello SM, Peck DD, Natoli TE, Lu X, et al. A Next Generation Connectivity Map: L1000 Platform and the First 1,000,000 Profiles. Cell. 2017;171(6):1437–1452.e17. doi:https://doi.org/10.1016/j.cell.2017.10.049.

[7] Subramanian A, Tamayo P, Mootha VK, Mukherjee S, Ebert BL, Gillette MA, et al. Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles. PNAS; Proceedings of the National Academy of Sciences. 2005;102(43):15545–15550.

[8] Shoemaker RH. The NCI60 human tumour cell line anticancer drug screen. 2006;6(10):813–823. doi:10.1038/nrc1951.

[9] Seashore-Ludlow B, Rees MG, Cheah JH, Cokol M, Price EV, Coletti ME, et al. Harnessing Connectivity in a Large-Scale Small-Molecule Sensitivity Dataset. 2015;5(11):1210–1223. doi:10.1158/2159-8290.cd-15-0235.

[10] Yang W, Soares J, Greninger P, Edelman EJ, Lightfoot H, Forbes S, et al. Genomics of Drug Sensitivity in Cancer (GDSC): a resource for therapeutic biomarker discovery in cancer cells. 2012;41(D1):D955–D961. doi:10.1093/nar/gks1111.

Appendix A

Appendix