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 protein). 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

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 have 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 in Portugal 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 replied that there is a 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 (The GTEx Consortium, 2013) and recount2 (Collado-Torres, 2017). Following an invitation from Springer Methods, I also prepared a book chapter on using psichomics to analyse alternative splicing in stem cell differentiation (Saraiva-Agostinho & Barbosa-Morais, 2020).

Table 4.1: Major released features of psichomics

Version	Release data	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

Following typical user requests, I added built-in support for analysing non-human data, including new alternative splicing annotations for 14 species (including mouse, fruit fly, frog, and Arabidopsis thaliana). These annotations are based on those provided by the alternative splicing quantification tool VAST-TOOLS (Irimia et al., 2014; Tapial et al., 2017). 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 for a chapter for a new edition of a book of protocols. I immediately sent an email to Nuno Morais stating that I was almost certain that this email was not spam.

4.2 GitHub Actions

4.3 Docker

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 lab retreat to Madeira in 2017, we focused our attention to what were the objectives of the lab and what we can 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 hinted in the original paper of CMap and usable through their online tool at `https://clue.io`, but there were issues with their implementation.

The Connectivity Map or CMap (Subramanian et al., 2017) 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 (`https://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 (Subramanian et al., 2005) approaches.

The most recent feature of cTRAP is its visual interface, allowing users to interactively explore their results. 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.

– Functions to deal with dummy/mock object

5.1 Benchmarking + code/memory optimisation

5.2 Graphical interface

cTRAP has one modular interface that allows to

This was an experiment that combines using explicit R commands via an helpful graphical interface.

5.2.1 Web server support

In order to increase its usefulness to the scientific community, we made cTRAP available online[1] with a single interface to perform all the analyses and plots in 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 simultaneously, for instance. To avoid this, long-running tasks are performed in the background. But this also meant that the users would need get their results back afterwards. And thus the idea of sessions was born.

In practical terms, cTRAP sessions are folders that are created in a directory based on a random, unique string of numbers and letters, henceforth denominated as a *token*. Each folder holds all the data for a session and either the associated token can be kept or the session data can be downloaded as a single RDS file. All data changes performed in the scope of the current session are saved into the respective folder.

When going to cTRAP in the next visit, the user is greeted with a welcome screen that allows to create a new session or load a previous one by either giving a token or a RDS file. The RDS file has another advantage: it ensures the user can open the 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 be inaccessible.

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.

However, cTRAP can also run the session feature without Celery/Flower support. It works similarly, but the tasks are run in the same R process (as usual in an R/Shiny

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

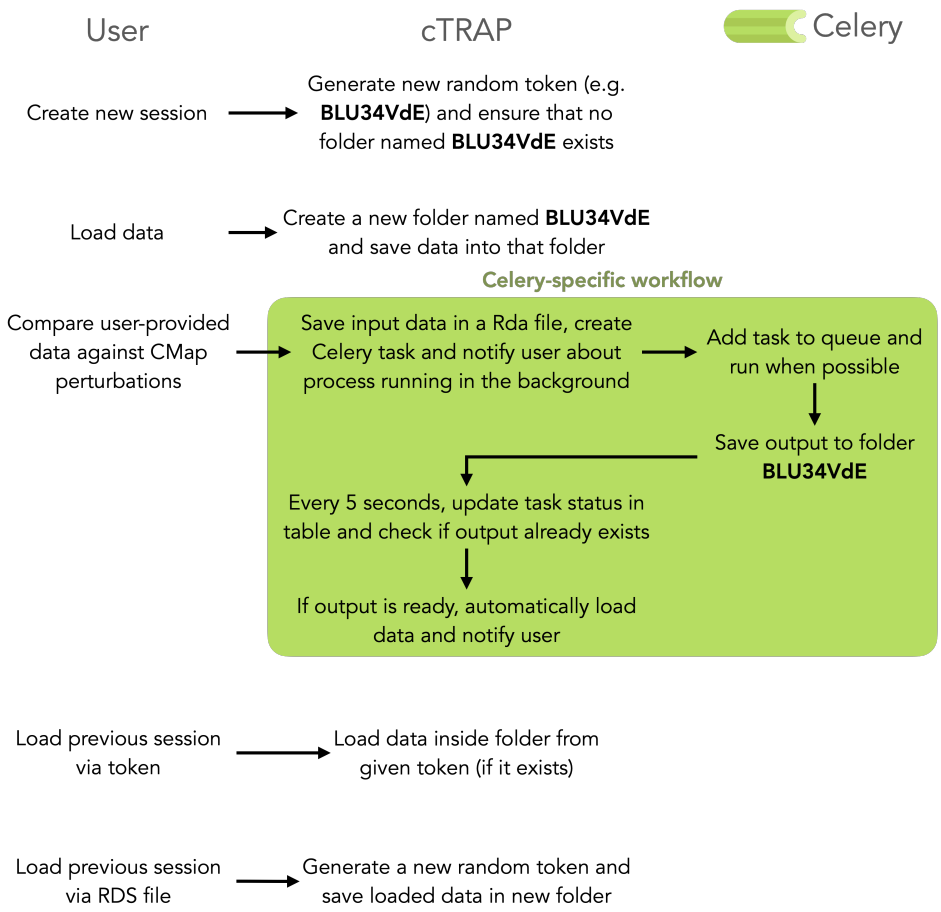

Figure 5.1: Interaction between the user, cTRAP and Celery.

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.

Most data used for the analyses in cTRAP are the same across sessions with the exception of the differential expression data. In the implementation of our app server, the common cTRAP data is available in a folder accessible to all sessions, thus skipping the step of downloading and preparing the data.

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.

Chapter 6

CompBio app server

Since I started building psichomics, I wanted to make it accessible as an online web app, allowing users to have the latest version at their fingerprints, without having to worry about installing and updating R, Bioconductor, psichomics and all their dependencies. Following the first release of psichomics in Bioconductor 5 years ago, that vision finally came true.

One of our lab's ambitious goals is to develop interactive visual tools to explore biological data intuitive enough to be used by everyone, no matter their computational background. To make that a reality, I set up the CompBio app server, a Linux virtual machine that hosts multiple Shiny apps, including psichomics and cTRAP, as well as dashboards from my lab colleagues. The app server is publicly accessible at `https://compbio.imm.medicina.ulisboa.pt`.

CompBio is built using Docker Compose, a program that manages multiple Docker containers simultaneously, and is composed by the following intercommunicating web services (Figure 6.1: **App server architecture**):

- **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.

[1] More information in subsection 5.2.1: **Web server support**

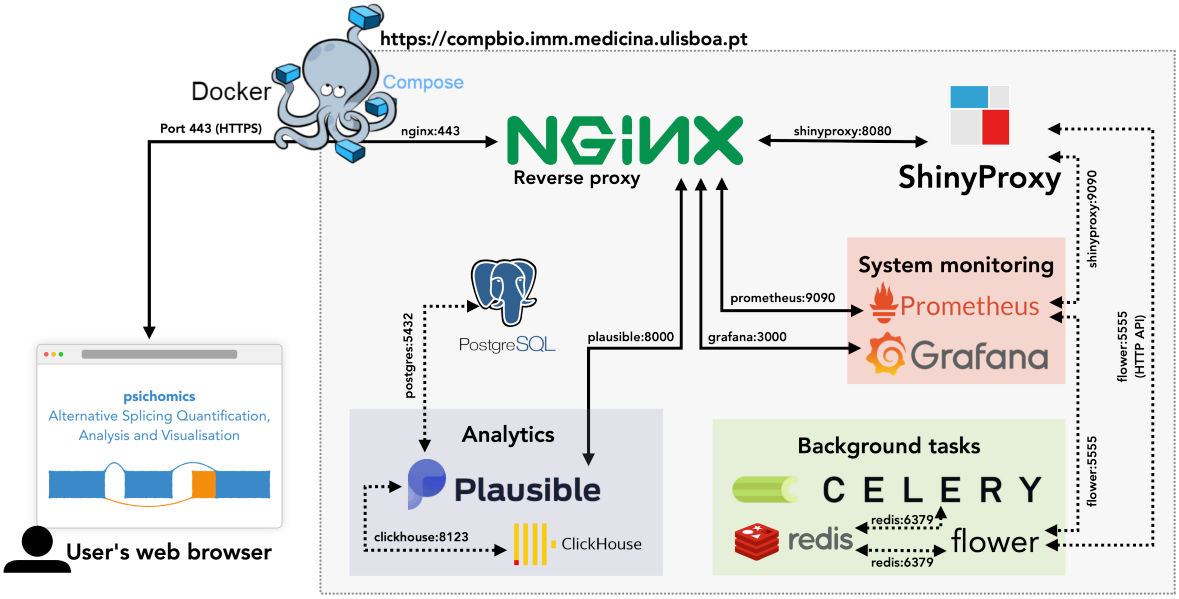

Figure 6.1: **App server architecture is based on Docker Compose.** All services are provided via Docker images and communicate with each other via a Docker-created network using the name of the service and a specific port (e.g. Nginx communicates with ShinyProxy via `shinyproxy:8080`). The groups (analytics, system monitoring and background tasks) are strictly conceptual.

- **RStudio Web** to run R sessions and test features (not used in production).

6.1 Docker Compose

In Docker Compose, 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 created when starting all services based on a Dockerfile. 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`) can be used to configure the applications. The configuration of each service depends on the specific application and can be found either in `docker-compose.yml` or in the local directory. For organisation purposes, the local directory was organised by folders named after each service, where each folder stores files (e.g. Dockerfile, configuration and data) associated with the respective application (Figure 6.2: **App server's file structure**).

Data from containers are preserved in volumes to avoid data loss when restarting services. Docker volumes are mounted when starting the `docker-compose.yml` project.

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

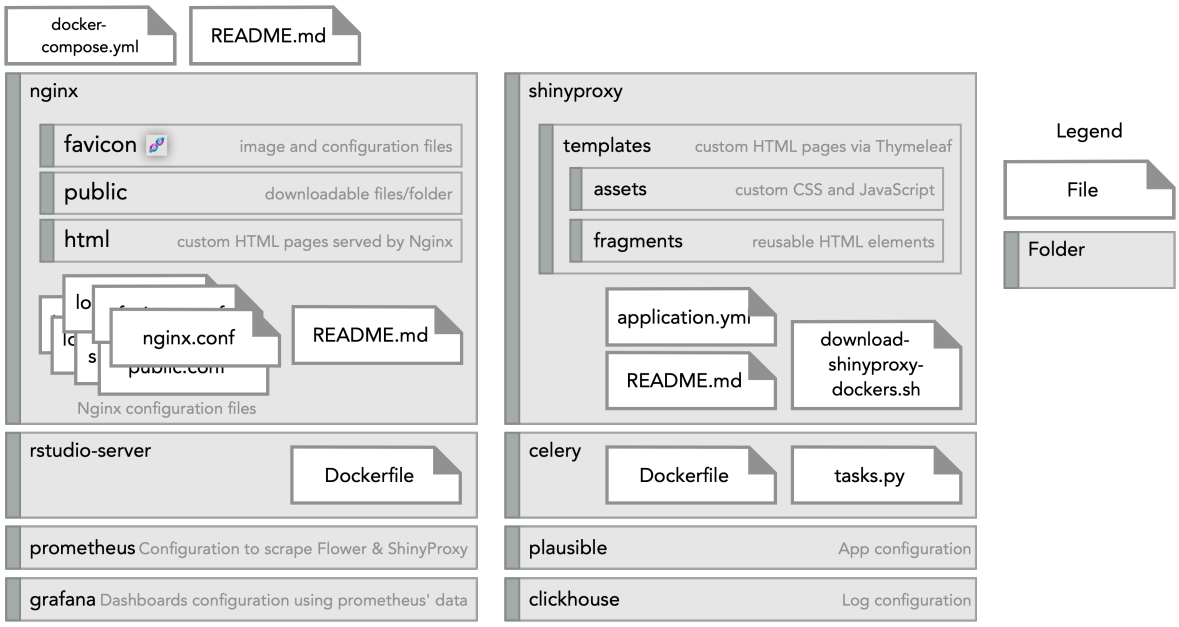

Figure 6.2: **App server's file structure.** Visual representation of the file/directory structure of the CompBio app server. Each folder contains files associated with a specific service. Folders `rstudio-server` and `celery` contain Dockerfiles for building Docker images of the respective services.

Although Docker containers destroy all data within, 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 available Dockerfiles, download Docker images from Docker Hub and start up every service in detached mode: `docker-compose up -d --build`.

Multiple `docker-compose` commands allow to manage the services. For instance, it is possible to restart single services (without affecting other programs), 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 `https://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 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); more information at `https://shinyproxy.io/documentation/ui/#using-multiple-instances-of-an-app`.

6.2.2 Progress bar when opening apps

When ShinyProxy is loading an app, a spinning wheel is shown as a loading indicator. This may work fine for apps that take 5 seconds to load, but anything longer may give the impression that there is something wrong with loading the webpage. To avoid that feeling, I added a progress bar that fills up with time to replace the spinning wheel.

However, if the progress bar fills up too fast, it will not have the desired effect. By default, the progress bar takes 5 seconds to fill (as most apps take that much time to launch in ShinyProxy), but the time is customisable for specific apps. For instance, psichomics usually takes 20 seconds to start, whereas cTRAP takes 15 seconds. When the app is ready, the progress bar fades into the app, no matter the progress shown.

To create this progress bar, `shinyproxy/templates/app.html` was edited to remove the spinning wheel and to include an empty progress bar. The progress bar's

width is changed from 0% to 100% a millisecond afterwards. To make this transition smooth (instead of filling the bar directly), the progress bar is given two classes for each app: classes `progress-bar` and a custom `progress-{appID}`, where `appID` is a specific app identifier. The CSS file (`shinyproxy/templates/assets/shinyproxy.css`) has a rule for class `progress-bar` to perform a 5-second ease in-and-out animation when changing the width of the progress bar. This rule can be superseded by a custom `progress-{appID}` with a different time interval of choice. In the case of psichomics, the CSS file includes a custom rule for its progress bar's 20-second transition: `.progress-bar { transition: width 20s ease-in-out; }`.

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

6.4 Plausible

6.5 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. 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 from Docker Hub and edit the versions of the Docker images used in `docker-compose.yml` accordingly. Afterwards, the app server services are restarted with `docker-compose down && docker-compose up -d --build`. Another advantage of using Docker Compose is that if something goes wrong with the updated Docker images, we simply need to edit `docker-compose.yml`, indicate a previous version of the software to run and restart the services.

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

Appendix A

Appendix