FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Reproducible and Privacy-preserving Analyses for Next-Generation Sequencing data

Mark Timothy Vasconcelos Meehan

Mestrado em Engenharia Informática e Computação

Supervisor: Prof. João Correia Lopes

Co-Supervisor: Prof. Artur Rocha

July 30, 2022

Reproducible and Privacy-preserving Analyses for Next-Generation Sequencing data

Mark Timothy Vasconcelos Meehan

Mestrado em Engenharia Informática e Computação

July 30, 2022

Abstract

The analysis of health research data has been growing in relevance and has proven to be very useful across many fields of scientific study. Some highly complex data analysis methodologies, named High-Throughput Technologies (HTT), have been developed to process extremely extensive volumes of patient data. One of the most relevant and standard HTT is Next Generation Sequencing (NGS). Through NGS, researchers can perform very detailed introspections across various biological applications, including sequencing the entire human genome.

To employ such data analysis technologies, researchers resort to bioinformatics pipelines, which are computational pipelines consisting of algorithms capable of extensively processing large datasets of sequencing data. Such pipelines usually require vast volumes of patient data, which is very sensitive and requires special safeguards and procedures to be put in place. Therefore, these data must be analysed in a privacy-preserving manner, which proves to be especially difficult when dealing with extensive datasets that may require execution across a cluster of machines.

A crucial property of bioinformatics pipelines is their reproducibility since they must constantly output relevant results across different builds. Furthermore, reproducibility empowers collaboration since researchers can share bioinformatics pipelines while achieving consistent results. At the moment, this is one of the principal liabilities that bioinformatics pipelines showcase.

This work focuses on studying and improving current frameworks for developing and analysing reproducible bioinformatics pipelines. An actual bioinformatics pipeline will be implemented in DolphinNext, which is considered the most advanced, complete and portable bioinformatics pipeline execution and distribution platform. A set of reproducibility features of the platform are evaluated, including execution output consistency, configuration overhead, portability and shareability. Besides that, different versions of a bioinformatics pipeline are implemented by changing execution-specific parameters. By analysing the similarity of different outputs, important conclusions are drawn from the impact that small changes in pipeline configuration can have on final output results.

From the carried work, it is clear that DolphinNext is a complete and stable platform that can provide accurate and consistent analysis of bioinformatics pipelines. Creating revisions will small changes in a given pipeline is supported natively and proves to be extremely useful in developing and maturing pipelines. Reproducibility seemed to be, overall, ensured by DolphinNext.

Studying a bioinformatics pipeline's determinism and consistency is crucial for understanding its reproducibility. Therefore, in future work, comparability tools should be supported natively across different pipeline execution and distribution platforms.

Keywords: Next-Generation Sequencing, High-throughput Technologies, bioinformatics pipelines, FAIR principles, Domain-Specific Languages, reproducibility, replicability.

Resumo

A análise dos dados de investigação em saúde tem vindo a crescer em relevância e provou ser muito útil em muitos campos de estudo científico. Algumas metodologias de análise de dados altamente complexas, chamadas *High-Throughput Technologies* (HTT) - têm sido desenvolvidas para processar volumes extremamente extensos de dados de pacientes. Uma das HTT mais relevantes e correntes é a Next Generation Sequencing (NGS). Através da NGS, os investigadores podem realizar introspecções muito detalhadas através de várias aplicações biológicas, incluindo o sequenciamento de todo o genoma humano.

Para aplicar tais tecnologias de análise de dados, os investigadores recorrem a *bioinformatics pipelines*, que são *pipelines* computacionais constituídas por algoritmos capazes de processar extensivamente grandes conjuntos de dados de sequenciação. Tais *pipelines* requerem normalmente grandes volumes de dados de pacientes, o que é muito sensível e requer salvaguardas e procedimentos especiais a serem postos em prática. Por conseguinte, estes dados devem ser analisados de forma a preservar a privacidade, o que se revela especialmente difícil quando se trata de conjuntos de dados extensos que podem exigir a execução distribuída através de um conjunto de máquinas.

Uma propriedade crucial das *bioinformatics pipelines* é a sua reprodutibilidade, uma vez que devem produzir constantemente resultados relevantes ao longo de diferentes execuções. Além disso, a reprodutibilidade permite a colaboração, uma vez que os investigadores podem partilhar *bioinformatics pipelines* alcançando resultados consistentes. Neste momento, esta é uma das principais responsabilidades que as *bioinformatics pipelines* apresentam.

Este trabalho concentra-se no estudo e melhoria das estruturas actuais para o desenvolvimento e análise de *bioinformatics pipelines* reproduzíveis. Uma *bioinformatics pipelines* real será implementada no DolphinNext, que é considerada a mais avançada, completa e portátil plataforma de execução e distribuição de *bioinformatics pipelines*. É avaliado um conjunto de características de reprodutibilidade da plataforma, incluindo a consistência da execução, a sobrecarga de configuração, a portabilidade e a capacidade e facilidade de partilha. Além disso, são implementadas diferentes versões de uma *bioinformatics pipelines* através da alteração de parâmetros específicos de execução. Ao analisar a semelhança dos diferentes *outputs*, são tiradas importantes conclusões do impacto que pequenas alterações na configuração da *pipelines* podem ter nos resultados finais.

Do trabalho realizado, fica claro que o DolphinNext é uma plataforma completa e estável que pode fornecer uma análise precisa e consistente das *bioinformatics pipelines*. A criação de revisões irá apoiar pequenas alterações numa determinada *pipeline* e revela-se extremamente útil no desenvolvimento e na maturação de *pipelines*. A reprodutibilidade parece ser, de um modo geral, assegurada pelo DolphinNext.

O estudo do determinismo e consistência de uma *bioinformatics pipelines* é crucial para compreender a sua reprodutibilidade. Portanto, no trabalho futuro, diferentes ferramentas de comparabilidade devem ser implementadas nativamente através de diferentes plataformas de execução e

distribuição de *pipelines*.

Keywords: *Next-Generation Sequencing*, *High-throughput Technologies*, *bioinformatics pipelines*, principios FAIR, *Domain-Specific Languages*, reprodutibilidade, replicabilidade.

Acknowledgements

Firstly, I would like to thank my supervisor Prof. João Correia Lopes and my Co-Supervisor Prof. Artur Rocha for teaching, supporting and guiding me, always closely, throughout my work for the last few months.

I would also like to extend my gratitude to Prof. Gur Yaari and to the researchers at the *Gur Yaari lab* for providing me with the necessary infrastructure for my work to be successfully developed, as well as supporting me so closely throughout. Likewise, to the researchers at INESCTEC who worked closely with me for all the support I had.

Finally, since this document marks the end of a chapter that lasted five long years of higher education, I would like to give a word of appreciation to some important people in my life during this time.

To my parents, Helena and Denis, for putting everything you ever had (and didn't have) into educating me.

To my grandparents, Carolina and António, for shaping me as a person since I was born.

To Luísa and Paulo, for always making me feel like my family is bigger than my household.

To Michael, Carolina, Marta and Francisca for being by my side since we were babies.

To Inês, for such great memories, adventures, travels, laughter and for sticking with me when the going got tough. I could not have asked for better company through it all.

To my fellow graduates, for all the late-night study sessions, the unhealthy number of coffees and, above all, great moments that are too many to mention. It was a hell of a ride and the best I ever had. See you around!

Author

"Knowledge is knowing that a tomato is a fruit,
wisdom is knowing not to put it in a fruit salad."

Brian O'Driscoll

Contents

List of Figures

List of Tables

Listings

Abbreviations

FAIR	Findable, Accessible, Interoperable, Reusable
DSL	Domain-Specific Languages
CI	Continuous Integration
HPC	High-Performance Computing
UI	User Interface
API	Application Programming Interface
AIRR	Adaptive Immune Receptor Repertoire
AIRR-seq	Adaptive Immune Receptor Repertoire sequencing
MiAIRR	Minimal information Adaptive Immune Receptor Repertoires
NGS	Next-Generation Sequencing
HTT	High-Throughput Technologies
BCR	B-Cell Repertoire
TCR	T-Cell Repertoire
ML	Machine Learning
CLI	Command Line Interface
CPU	Central Processing Unit

Chapter 1

Introduction

Over the past decade, health data has become increasingly more available across many areas of scientific study. With technological and scientific advancements in data collection methodologies, researchers are seeing increasing potential in Omics data at a molecular level, which can now be studied in unprecedented detail through novel emerging technologies.

1.1 Context

With the technological advancements made during the last decade, Omics data has become ever more prevalent and significant across many areas of study. Advances in biology and computation have been exponential, providing unprecedented scientific understanding. A new science field which looks to study biology through informatics and computer-based methodologies, named bioinformatics, has been created from the pillars of biology and computer engineering.

In recent years, new patient data analysis technologies have been created, which provide a framework for analyzing large volumes of highly complex data named High-Throughput Technologies (HTT) [12]. Next-Generation Sequencing (NGS) is one of the most widely and commonly used High-Throughput technologies since it is capable of sequencing entire genomes and, therefore, offers a deep understanding of molecular data [31].

These methodologies created, in consequence, a demand for specially fine-tuned workflows for data analysis, which are called bioinformatics pipelines [22]. These pipelines handle the vast datasets collected through HTT and are capable of performing very detailed analyses of patient data across a chain of interlinked data process tasks [8].

1.2 Motivation

The evolution and maturing of NGS and other HTT resulted in highly complex bioinformatics pipelines. Besides that, the sensitive nature of health research data, alongside the endless efforts

toward preserving patient data privacy, created a surge in demand for shareable computational pipelines across researchers [33].

The complex nature of bioinformatics pipelines poses a hurdle in ensuring their reproducibility in that they execute a set of instructions and algorithms imported from specific packages and dependencies. The long list of dependencies of a pipeline, which often needs a particular version of each dependency to function correctly, complicates sharing a bioinformatics pipeline. Creating a framework that abstracts the users from the complex dependency configurations of each pipeline would ensure that pipelines are reproducible and easily executable by other researchers [33].

Patient data preservation should remain a core cause for concern in bioinformatics pipelines. When dealing with extremely extensive datasets, the analysis may only be possible through a pooled co-analysis of data from different studies [7]. However, this approach raises some ethical issues and is currently contested. Facilitating the reproducibility of a pipeline will also contribute to preserving the privacy of patient data since researchers can opt to share the pipeline on its own without the need to share data. This also explains another benefit of sharing data analysis tools: different researchers can use other datasets for the same bioinformatics pipeline, which then leads to new findings and insights since the work is cross-examined [5].

By facilitating the reproducibility and shareability of bioinformatics pipelines, researchers can more easily compare results, share analysis frameworks, and generally gain better insights into patient data. This will, in turn, result in better medical research findings and discoveries.

1.3 General Goals

This dissertation's primary goal will be to study the best available platforms for developing, executing and sharing reproducible pipelines currently available. A previously created bioinformatics pipeline will be implemented in the most well-fitted reproducibility platform so that relevant conclusions can be drawn from the developed study.

Concretely, a set of exploratory work will be performed when implementing bioinformatics pipelines: the pipeline's shareability (since it is a core factor for reproducibility and replicability of bioinformatics pipelines), the ease of execution, overhead of configuration, and quality of output information.

Finally, a set of proposals for further improving current reproducibility platforms will be constructed to contribute to the current state of the art in reproducibility frameworks.

1.4 Document Structure

The structure of the remaining chapters of the document is next described.

Chapter 2, *Background*, aims at explaining key concepts related to the work implemented in this dissertation.

Chapter 3, *State of The Art*, describes the related work already carried out in the scope of the work implemented in this dissertation.

Chapter 4, *Proposed Solution*, explicitly defines the solution that this dissertation proposes to implement.

Chapter 5, *Implementation*, details the implementation performed.

Chapter 6, *Results*, described the obtained results.

Chapter 7, *Conclusion And Future Work*, provides a conclusion for the developed work and proposes future work to be carried out.

Chapter 2

Background

This chapter provides the necessary background to help understand the context of bioinformatics and a description of the leading technologies relevant to the scope of work being developed.

2.1 FAIR Principles

The complex nature of health research data collection and analysis creates a need for data reuse. An essential step toward ensuring that scientific studies can be conducted accordingly is to secure a good framework for data management, not only during the course of research but also for future studies, ensuring the ease of access to information [32]. This way, the information's life span is increased, which leads to data being reused by future studies. Besides data access, other factors such as data retrievability and evaluation should also be facilitated. In fact, these improvements should be made to all components of the scientific workflow, that is, to all steps ranging from data collection, data analysis itself (which falls heavily into the topics next described in this chapter) and data storage.

Due to significant data collection and analysis advancements, science has become increasingly data-intensive. In order to cope with said advancements, information should be readily and easily available to humans and machines "in the discovery of, access to, and integration and analysis of, task-appropriate scientific data and other scholarly digital objects" [32].

A set of four principles was made to solve these issues and facilitate the reuse of information universally. These four principles are commonly known as FAIR principles and are the following: Findability, Accessibility, Interoperability and Reusability.

Findability: To ensure that humans and computers can quickly find data, both data and metadata should contain an assigned unique identifier so that they can be indexed correctly. Data should be described through extensive metadata. Metadata should reference the data it describes.

Accessibility: Data and metadata should be easily accessible through their unique identifier via a protocol that should be open, free, universal and disclose proper authentication and authorisation mechanisms. Even when data is not accessible, metadata should still be.

Interoperability: Data and metadata should be easily accessed and used across different platforms, meaning they should use a broadly applicable language.

Reusability: Data and metadata should be described in detail, be released with a clear and accessible data usage licence and, finally, meet the respective community standards.

These four principles should be taken into account from the early stages of a data management workflow and work as guidelines for use throughout its lifecycle. They are purposefully broad and don't specify any technologies or requirements so that they can be used broadly, reaching most of the scientific community.

2.2 Omics Data

The analysis of health research data has been growing in relevance and has proven to be very useful across many fields of scientific study. These data usually encompasses both biological and molecular features, from various scientific domains — such as genomic and immunologic data (among other fields of science, which generally possess the "omic" suffix) — and is generally referred to as Omics data.

Omics data have associated Omics analysis methodologies, which are very recent and are still in a maturing stage. Until recent years there were technological limitations to the analysis of Omics data, since it is extremely detailed and rich in information, not only relative to the sequencing data itself but also related to the factors which need to be analysed in order to extract the most out of the data. As an example, genomics is one of the main areas of Omics data, and constitutes the study of the entire genome of an organism, as well as the relationships that the genes hold between them. Obviously, studying the entire DNA of a species is extremely extensive work, and requires special novel methodologies to be done effectively.

However, given the appropriate analysis framework, infrastructure and methodologies, analysing Omics data is extremely promising. Even though it is still at a relatively early stage, analysing Omics data has already proven to be extremely useful in detecting pathologies, for instance, and gaining a general better understanding of the human body, across many different specific fields. It has become clear that understanding the genome of an individual gives important insights into how that individual functions.

2.3 Next-Generation Sequencing

Some highly complex data analysis methodologies, in the form of HTT, have been developed to process health research data. One of the most relevant and standard HTT is NGS, a technology capable of sequencing the entire human genome in a single day, with reduced costs [19]. Through NGS, researchers can perform very detailed introspections across various biological applications,

including the study of the response of the human genome to infection or pathological conditions such as autoimmunity and cancer [17]. As these technologies mature, new research and diagnosis perspectives have arisen, bringing very bright perspectives to human medicine.

However, the rapid expansion of NGS applications also calls for further advancements in data storage, analysis and visualisation frameworks. Specifically, NGS data analysis is most often made in the form of bioinformatics pipelines, which will be further described in Section 2.5. In recent years, with the fast advancements in High Throughput Technologies, such computational pipelines have become even more relevant in processing the large volumes of molecular data that these technologies require.

Finally, it is essential to note that the analysis of NGS data usually requires extensive volumes of patient data, which is very sensitive and requires special safeguards and procedures to be put in place. Therefore, these data must be analysed in a way that preserves their privacy, which proves to be especially difficult when dealing with very extensive datasets. For this reason, it's essential to apply the FAIR principles described in Section 2.1 to such analysis to preserve patient data privacy to the fullest.

2.4 The AIRR Community

There has been a remarkable increase in antibody and T cell receptor sequencing data in recent years, mainly due to the tending lowering DNA sequencing costs [10]. The AIRR (Adaptive Immune Receptor Repertoire) community emerged in 2015 due to such an increase in data collection and evolution of NGS techniques. This research group focuses on leveraging NGS sequencing to study antibody, B-cell and T-cell repertoires, through implementations that follow the FAIR principles [26]. Such data is commonly named AIRR-seq data and consists of a subset of NGS data. The AIRR community establishes a set of data and metadata standards, reference implementation tools, and a collection of practices to ensure the correct gathering, storage and analysis of AIRR-seq data [3].

2.4.1 AIRR Data Model

One of the main goals of the AIRR Community is to establish a set of standard practices for the main stages of data analysis (collection, processing and sharing). By creating such best practices, the collaboration between laboratories and researchers is greatly facilitated, as the data is stored in a standardised format. Therefore, comparative and integrative analysis of AIRR-seq data is a fundamental goal of the community and a critical step towards reproducibility.

To create relevant and accurate standards for AIRR-seq data, a set of minimal required metadata for describing a dataset was established (MiAIRR), which describes the study design and the type of information represented in such data [10]. By ensuring that different AIRR-seq datasets follow this model, it is possible to perform cross-examination on them since they follow the same data standard. This proves to be extremely valuable since it increases collaboration and allows for data sharing between pipelines and applications.

In order to tackle these issues, the standards created by the AIRR community consist mainly of data models, schema applications, file formats, and application programming interfaces (APIs) to promote interoperability and reusability of AIRR-seq data [10]. Here, the focus will be on the developed data models since they can represent not only metadata related to the collected data but also some information relating to the data analysis.

The MiAIRR standard defines six blocks of data elements that represent the most broader objects of AIRR-seq datasets:

- Study, Subject and Diagnosis
- Sample Collection
- Sample Processing and Sequencing
- Raw Sequences
- Data Processing
- Processed Sequences with Annotations

Even though this standard is an instrumental step towards reproducibility, it could be expanded into defining the analysis step more in detail. In Section 4.1, this model will be slightly modified and developed to represent the data analysis stage in greater detail.

2.5 Bioinformatics pipelines

To analyse Omics data, researchers resort to computational pipelines, which are fine-tuned towards bioinformatics computations, called bioinformatics pipelines. These pipelines contain a set of algorithms that can process vast datasets and analyse them very detailed and complexly. With the rise of HTT, bioinformatics pipelines became even more relevant in analysing health research data since the complex nature of NGS data requires well-built and efficient computational analysis pipelines to produce consistent and relevant results.

In order to execute efficiently and in a scalable manner, bioinformatics pipelines should be developed in a modularized way, exemplified in Figure 2.1. A user interacts directly with the pipeline through a User Interface layer, which usually constitutes a web application or CLI tool. The specific software tools used in the pipeline are orchestrated by a pipeline specification workflow, which a pipeline or a script can implement. Finally, the respective computations must be executed in any given execution environment, which usually consists of a local workstation, cluster of nodes, HPC or Cloud machine [6].

Even though bioinformatics pipelines are customisable depending on laboratory and research needs, the foundations are standard across mostly all use cases. The main steps and workflow of a bioinformatics pipeline are shown in 2.2.

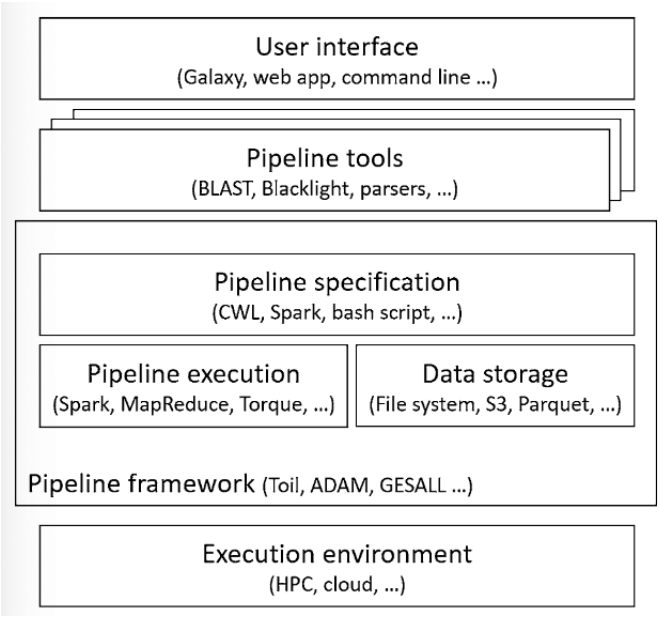


Figure 2.1: Bioinformatics pipeline framework. From [6]

Due to bioinformatics pipelines' complex nature, where multiple tools are used to execute specific computations, it can become challenging to keep track of all needed dependencies and perform the necessary installations in the execution environment. For each processing step of a pipeline, there are usually many different tools which can be implemented. Previous benchmarking studies for the most widely used bioinformatics tools (such as *IgBLAST* and *MiXCR*) have been made. In such studies, researchers concluded that even seemingly small changes, such as updating the reference germline sets used in the respective tools, can significantly change annotation output [27]. Besides that, another issue is that some bioinformatics pipelines implement Machine Learning (ML) algorithms, which are practical analysis frameworks for AIRR-seq data, that make pipelines non-deterministic and cause outputs to vary across different executions [21]. For this reason, to ensure pipeline reproducibility, it is vital to both fully document bioinformatics pipelines in a standardized way and have access to a comparability framework that accurately detects output similarity.

Since pipelines can be parallelised and executed by many nodes, they can be executed across a cluster of machines. In this situation, it is necessary to configure the execution environment accordingly across various machines, representing a significant overhead and making maintenance extraordinarily laborious and complex. Therefore, a solution for reproducibility of pipeline execution is crucial for efficient NGS data analysis.

Collaborative work is a foundation of science, and only through collaboration is it possible to perform studies on highly complex areas such as genomics. This has become more evident in recent years, with the maturing of High-throughput technologies such as NGS, that increase the level of complexity of the topics that researchers can study. Therefore, collaboration has become increasingly important, and there is often a need to perform comparative studies of bioinformatics

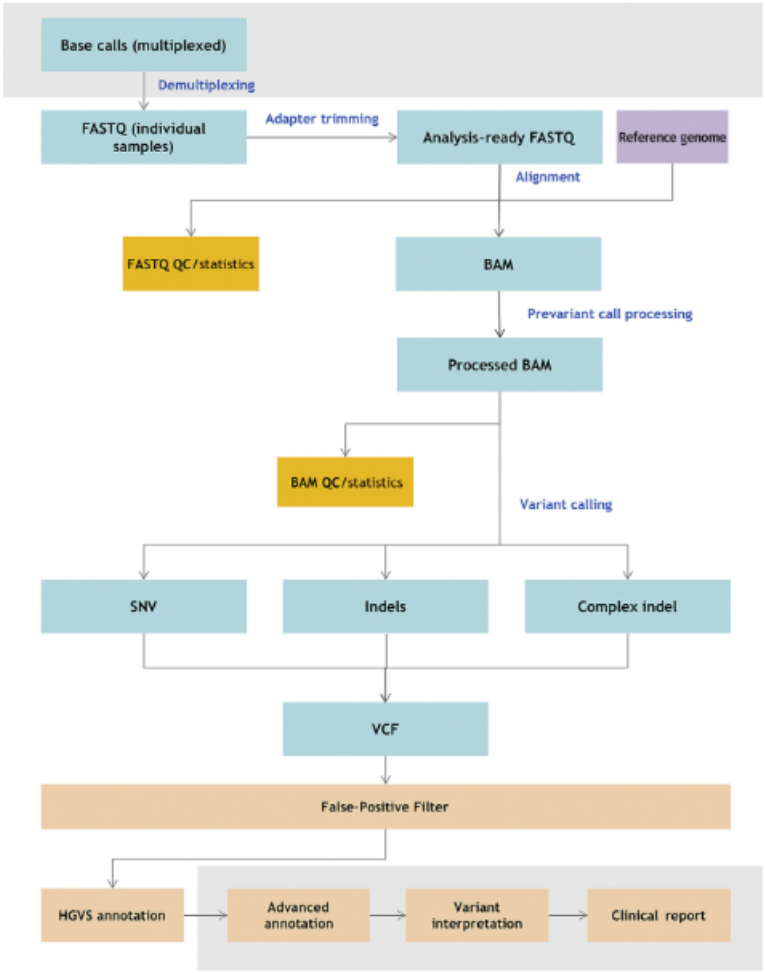


Figure 2.2: NGS bioinformatics pipeline. From [25]

pipelines. Different datasets, each hosting their own set of standards and formats, need to be analysed to achieve this [11]. The lack of a general standard for bioinformatics pipelines is an obstacle to the current need for collaborative studies. Besides that, the vast volumes of data to be consumed also pose a challenge from the computational perspective.

To solve this issue, scaling solutions need to be provided. Vertical scaling (increasing machine power) could be a simple solution. However, the high cost it brings represents a mishap. Besides that, vertical scaling generally does not constitute an optimised solution since there are many independent processes to be executed in most bioinformatics pipelines. Therefore, horizontal scaling offers a solution based on parallelisation and efficient task orchestration. Even though computational nodes should still be powerful, having many nodes is the main improvement that can be made. This way, researchers can execute separate, independent processes on different nodes, released after executing a given process entirely, so that they can execute another. However, execution environments still need to be correctly configured. Container technologies, which will be next described, can be a solution to such an issue since they offer the ability to pre-configuring portable execution environments with all necessary dependencies.

Figure 2.3 showcases a typical workflow of computational bioinformatics, from data collection to results interpretability. In the first stage, most commonly known as the experimental phase, physical samples are collected. Then, they are sequenced through specific technologies, which nowadays usually consist of High-Throughput technologies such as NGS, originating raw data stored in files of a specific format, usually *fastq*[1]. A bioinformatics pipeline's workflow then takes the raw data and generates meaningful information through a sequence of processing steps that usually implement specific software tools. Finally, the interpretable outputs are analysed to entice conclusions from the study [13].

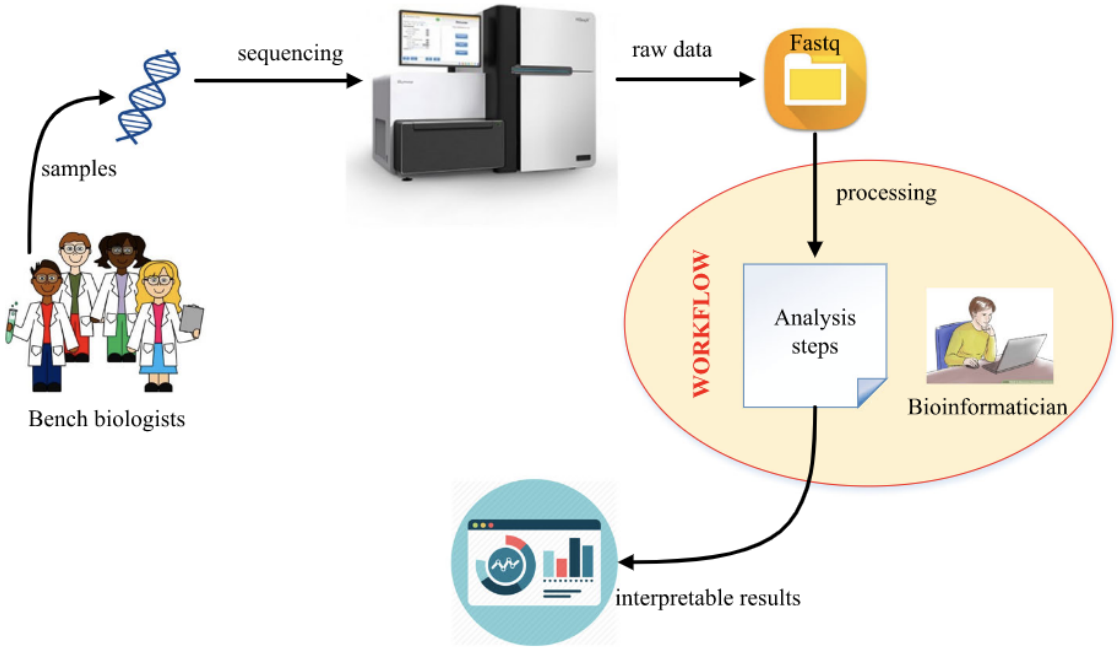


Figure 2.3: Bioinformatics computing workflow. From [13]

2.5.1 The *fastq* file format

Bioinformatics pipelines usually take in sequencing read data in the form of *fastq* files, which combines both the sequences and associated base quality scores [4]. This file type is very specific, and consists of a large set of reads, which each span four lines. Figure 2.4 exemplifies a single read of a *fastq* file. The first line represents an identifier for the specific read, followed by a line containing the sequence identifier name attributed to the read by the sequencer (machine which generated the NGS data). The following line simply contains a delimiter character. Finally, the fourth line contains the Phred quality score for the bases represented on the second line. This score represents the error probability associated with each base through chromatogram analysis [1], and is calculated by the following logarithmic Equation 2.1:

$$Q = -10\log_{10} p \tag{2.1}$$

[1] https://thesequencingcenter.com/knowledge-base/fastq-files/

Figure 2.4: Composition of a fastq read. From https://old.abmgood.com/

Where Q represents the Phred score and p is the probability of a single base to have been incorrectly called.

2.6 Containerization

During the last decade, due to the exponential growth of software tools and platforms, software applications have become increasingly more complex. It has become significantly more challenging to execute and deploy software across different machines due to the large set of dependencies and software tools required in most modern applications. The overhead of installing and managing all the dependencies and tools to successfully deploy an application has been increasing, and performing successful deployments in plain Virtual Machines poses a difficult challenge. Besides that, developing and deploying software in variable infrastructure also poses a significant challenge, which is further noticeable in scientific workflows composed of multiple tasks requiring complex and specific tools and packages ([20]). In response to these issues, an open platform called Docker[2] was created. The main goal of this platform was to provide a portable and lightweight platform for building and deploying software applications through a technology called containerisation. Containerisation refers to the ability to create simple "virtual machines", commonly called containers, which contain an operating system and the necessary dependencies required to run a specific application. For this reason, if configured correctly, containers represent, in essence, light virtual machines.

In particular, Docker has three main components: Docker images, Docker registries and Docker containers. Docker images are, in essence, templates which specify all the requirements to build a Docker container. They may contain source code, requirements, external tools and other required components. Images have a layered structure, meaning a given image can build over another image, adding further requirements. Docker images can be shared in registries, which can be private or public. Docker hosts a public registry, Docker hub, where users can freely pull images from the registry or, in the alternative, push their images to the registry. The open nature of image registries ensures that images are accessible and that users can use preexisting images, building on top of them if needed. Finally, Docker containers are isolated, virtualized environments defined by docker images. By having been pre-configured by the corresponding Docker image, containers

[2] https://www.docker.com/

should contain all the necessary configurations for an application to be executed as expected in the deployment host machine.

Docker containers have a very significant advantage over conventional virtual machines when it comes to deploying multiple instances. On a relatively weak machine, up to 50 requests can be made simultaneously to docker containers, as opposed to conventional virtual machines, which are limited to around ten instances [23]. Therefore multiple active containers can be deployed. As we can see in figure 2.5, three containers are deployed in the host Operating System. Each container has resource allocation requirements and constraints, and they are fully isolated, even though they share the same host machine. Docker uses an extra layer between the host Operating System and the higher layer in which the applications are executed, called the Docker engine [23].

Docker is also nicely integrated with other orchestration and cloud technologies, which is extremely useful for application development. By joining the capabilities of different technologies, they can be used to their full extent and further improve computational and workflow constraints that are otherwise challenging to overcome.

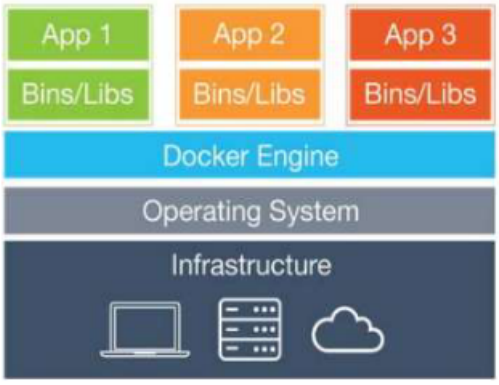


Figure 2.5: Docker container architecture. From [23]

2.7 Orchestration

With the breakthrough of NGS, the extensive volume of highly detailed AIRR-seq data created a need for open and scalable bioinformatics pipelines, which could serve such demanding computational requirements [2]. When deploying or executing a program requiring much computational power, a network of computers or nodes is often required for efficiency. However, when creating a cluster of computers that share executions and results, there is often a need for a framework that manages all of the nodes and specifies requirements to function as expected. The management of containers is often referred to as orchestration, and some technologies currently offer such service. Besides Docker, there is another platform, named Kubernetes[3], that provides fine-tuned orchestration of Docker containers and is currently the industry standard. By resorting to both technologies,

[3] https://kubernetes.io/

it is possible to not only execute a program across a cluster of isolated and optimised containers but also orchestrate their executions.

For this reason, containerization, orchestration and also cloud computing are very relevant for bioinformatics and, even though some work has been developed in the sense of leveraging these technologies in bioinformatics pipelines, there is still a lot of potential and further work that can be done to improve this.

Chapter 3

State of the art

This chapter describes the current state of the topics related to the work being carried out in this dissertation. It builds on the concepts presented and described in the previous chapter and provides an overview of the related work that has been carried out in bioinformatics and, more specifically, the reproducibility of bioinformatics pipelines.

3.1 Nextflow

A crucial property of bioinformatics pipelines is their reproducibility since they must constantly output relevant results across different builds. Furthermore, they need to be replicable across different environments. This is important so that the data analysis can happen in a distributed fashion, across multiple processing nodes, and because it facilitates collaborative studies since researchers can execute pipelines in any given environment.

Currently, this is one of the central liabilities that bioinformatics pipelines showcase. Workflow managers are platforms which focus on simplifying the implementation and execution of bioinformatics pipelines while providing further features which enhance reproducibility, useability, task scheduling optimization, and cost-efficiency, among others [33]. Besides that, Domain-Specific Languages (DSL) are often used to describe these pipelines [18]. However, some issues arise from this approach (namely their reduced versatility in aligning with different types of pipelines). As a solution for bioinformatics pipeline standardization, Domain-Specific Languages (DSLs) are commonly viewed as a good solution. They offer a simple framework for declaring a pipeline's workflow, stating the input and output of each processing step, as well as managing the most efficient task orchestration when necessary at runtime.

More recently, Nextflow[1] was created to provide a DSL very well-tailored to the context of bioinformatics. Over the years, it has become the industry standard as the DSL to define bioinformatics pipelines. It offers native containerization and orchestration functionalities, integration

[1] https://www.nextflow.io/

with cloud providers and other useful features.

Nextflow is a workflow framework which aims to facilitate the development of portable and reproducible pipelines. The framework implements the dataflow paradigm, in which tasks are fully isolated and can execute in a stateless manner, which allows for distributed executions of pipelines across various execution environments [28]. It is, in essence, a pipeline development and execution framework that leverages containerization technologies to ensure reproducibility without compromising on scaleability [35]).

In complex genomics analysis pipelines, multiple tools and software packages are implemented. Maintaining working versions of many packages across a single analysis pipeline brings many challenges. Since bioinformatics pipelines are experimental, version changes of implemented tools are frequent and can raise reproducibility issues [30]. Packaging the correctly versioned dependencies for all necessary software packages is a crucial step towards reproducibility, and container technologies are a reliable and tested solution for such issues. This approach also facilitates the packaging and distribution of bioinformatics pipelines, contributing to collaboration across many fields of study.

3.1.1 Docker integration with Nextflow

As mentioned, Nextflow natively supports the execution of pipelines in a containerised environment through Docker. Besides that, Nextflow also supports containerisation through Singularity[2], another container platform focused on providing reproducible scientific computations [16]. It is possible to declare a Docker image that will serve as the execution environment for a given pipeline when deployed as a container. The used images can be fully configured by researchers and packaged specifically for a single pipeline or be developed more broadly so that they can be reused across pipelines. Images can be stored in an image hub, such as *Docker hub*, or be built in a local workstation.

Furthermore, it is possible to develop images for specific pipeline processes instead of serving all the processes present in a pipeline in a single container. Nextflow supports a variety of configurations to be made in a *nextflow.config* file, including specifying which image should be used for each processing step, which is exemplified in Listing 3.1

[2] https://docs.sylabs.io/guides/3.5/user-guide/introduction.html

```
process {
    withName:process1 {
        container = 'image_name_1'
    }
    withName:process2 {
        container = 'image_name_2'
    }
}
docker {
    enabled = true
}
```

Listing 3.1: Example *nextflow.config* file

The researcher must choose which approach to follow when selecting or developing Docker images for pipelines. Some pipelines, which may be executed in resource-restricted environments, or across a cluster of computational nodes, may be better suited for a multi-image approach, where each process is executed in a very well-tailored image. On the other hand, very complex pipelines or pipelines executed in a single container may be better suited for a single-image approach, which is also simpler to develop. Even though it may be heavy, this single image can package all necessary software tools the pipelines use.

The integration with Docker and Singularity is an excellent step toward achieving pipeline reproducibility since it allows for executing pipelines in a truly isolated and contained environment. Nevertheless, given the extreme computational effort required by most bioinformatics pipelines, the computational performance of Docker containers should be evaluated. Some benchmarking studies found that Docker containers had a negligible performance overhead in executing medium or long execution tasks (the vastly more common bioinformatics tasks) [30]. For this reason, it is found that minor the performance loss of executing bioinformatics pipelines in containerised environments is significantly offset by the reproducibility gains of creating very controlled self-contained execution environments.

3.1.2 Orchestration in Nextflow

The use of software containers, such as the container technology provided by Docker, generates the need for managing and organising them, particularly in large-scale systems. Orchestration technologies offer a solution for container management, and Kubernetes has been the industry standard for highly scalable container orchestration.

The extensive computational efforts that bioinformatics requires mean that they are an excellent potential application for orchestration, in the sense that the computational effort is usually distributed across many nodes of a network. Kubernetes allows for declaring a set of variables and configuring a cluster of computational nodes to be deployed. These deployed computations nodes act as workers, and a Master node can also be deployed to manage them, following a master-slave

architecture. This is a very well-tested architecture for general computational workflows supported by Nextflow [20].

The nodes launched and managed by Kubernetes are executing Docker images that can be pulled from a container registry without any additional configuration. Figure 3.1 displays the general architecture of deploying a pipeline across a cluster of Docker containers. The Nextflow driver pulls the application workflow and the needed Docker images and orchestrates the task allocation across a cluster of computational nodes [35]. In essence, the orchestration features that Nextflow offers to build on top of its container features, providing a way of configuring the deployment of nodes executing images. The configuration of the Kubernetes cluster can also be made in the *nextflow.config* file.

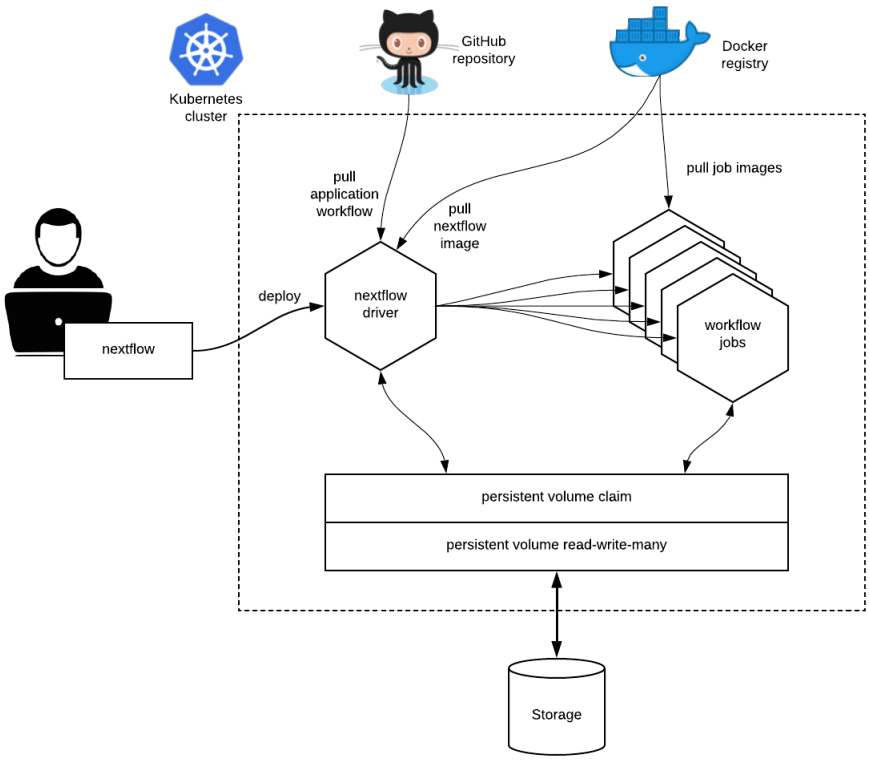


Figure 3.1: Nextflow kubernetes architecture. From https://www.nextflow.io/docs

3.2 The *nf-core* platform

As the bioinformatics research community understood the value of reproducibility in NGS data analysis, some platforms have emerged as solutions for such issues. More notably, *nf-core* is a community-driven framework for developing fully portable and reproducible bioinformatics pipelines [5].

The *nf-core* platform aims to standardise bioinformatics pipelines by providing a framework for defining guidelines, code standardisation and comprehensive documentation. Since it is a fully

Table 3.1: *nf-core* guidelines for bioinformatics pipelines. From [5]

Importance	**Guideline**	**Comment**
Requirement	Built using Nextflow	NA
Requirement	Have an MIT license	NA
Requirement	Software bundled using Docker / Singularity	NA
Requirement	Continuous integration testing	Pipeline tests run on Travis CI with minimal test dataset.
Requirement	Pass nf-corelint tests	Lint tests run on Travis CI with each update
Requirement	Stable release tags	GitHub releases are used with an associated changelog, and DOI using Zenodo
Requirement	Common pipeline structure and usage	NA
Requirement	Run in a single command	Pipelines should not be split into multiple sub-pipelines
Requirement	Excellent documentation	Pipeline specific documentation, in addition to documentation on the main nf-core website.
Requirement	A responsible point of contact / GitHub username	NA
Recommendation	Software bundled using bioconda	Software should be packaged for bioconda if not already available
Recommendation	Require as little input metadata as possible	Optional pipeline steps to convert file types and to build reference indexes.
Recommendation	Optimized output file formats	Standard file formats where possible, including CRAM.
Recommendation	Explicit support for running in cloud environments	NA
Recommendation	Benchmarks from running on cloud environments	NA

open platform, it incentivises discussion and collaboration from the community members [5]. The proposed guidelines and respective descriptions are described in Table 3.1.

By offering extensive documentation and pre-configured templates, *nf-core* facilitates the creation of bioinformatics pipelines from scratch. There are a set of community-built and maintained pipelines that researchers can freely use.

Since all *nf-core* pipelines are implemented Nextflow, they can be deployed across a vast set of containerised environments, including workstations, HPCs, or on the Cloud.

3.3 Continuous Integration/Continuous Delivery

Besides the Cloud support that Nextflow offers, there are no native Continuous Integration (CI) solutions natively integrated with the framework. Usually, the complex nature of bioinformatics pipelines poses an obstacle to creating a CI solution. Besides, many CI features are already somewhat provided by Nextflow, such as task scheduling and parallel execution. Previously to Nextflow's release, many bioinformatics pipelines were fully developed, deployed and executed in CI platforms, such as Jenkins[3].

However, CI offers many benefits and improvements to a workflow, especially in a collaborative environment. With the current efforts to create reproducible and shareable pipelines, CI could be another vector for leveraging shareable pipelines. Configuring automated tests would be an elementary example of the benefits of creating a CI framework in bioinformatics pipelines. In fact, with the increasing complexity of bioinformatics pipelines concerning data volume and many analysis tools available, the need for such automation has been increasingly evident.

Nevertheless, there is a margin for improvement in integrating bioinformatics pipelines in a CI environment, and this possibility should be explored.

3.4 DolphinNext

In modern high-throughput sequencing analysis pipelines, dozens of workflow processes usually use multiple analysis tools specific to each processing step, which represents a very significant hurdle in reproducing different executions. Soon, researchers understood the value behind the ability to visually represent an analysis pipeline, from the upper-level representation of its basic structure and process communications to the specification of scripts, tools and dependencies. In the initial stage, some platforms, such as Galaxy, were developed to provide a platform for a visual representation of bioinformatics pipelines. Specifically, in Galaxy, "Users upload data to a central server and apply a diverse, heterogeneous set of programs through a standardised user interface" [34]. Even though creating such platforms was a crucial leap toward visual representation in pipelines, which is a vital pillar of reproducibility, there were some clear limitations concerning the platforms' capabilities.

In the context of bioinformatics, there is an evident intersection of scientific domains. Multiple steps across many fields of study are involved in analysing NGS data. Researchers must have a deep understanding of biological concepts and good software development abilities. This makes sharing pipelines frustratingly delicate since researchers often do not have the necessary specific knowledge to perform a given analysis. Therefore, there is a clear need for a framework that removes the overhead of analysing a pipeline, so a less knowledgeable user can fully and easily reproduce and develop an understanding of any given pipeline. There have already been some

[3] https://www.jenkins.io/

efforts to facilitate the analysis of NGS data at different levels including the development of user-friendly frameworks for scalable analysis of NGS data [29]. However, there are still improvement opportunities in this context.

To tackle the difficulties of developing, analysing and executing complex bioinformatics pipelines, DolphinNext[4]. was created. It proposes an end-to-end platform, where an inexperienced user can develop a pipeline in its whole, from the designing to the testing and reporting stage.

With the recent advancements in HPC centres and cloud computing, it has become increasingly crucial that computational pipelines are portable and executable across different environments. Otherwise, the excessive overhead of manually configuring an environment for a specific computing platform is not sustainable. By leveraging some of the very useful containerisation and orchestration features of Nextflow, DolphinNext allows for highly portable and reproducible pipelines to be created, developed and executed. This is achieved through abstraction since Nextflow uses the many Central Processing Unit (CPU) cores available to execute tasks in parallel accordingly, without needing specific configuration by the user.

Even though Nextflow provides all these great features, it "can get unwieldy when pipelines become complex, and maintaining them becomes taxing" [34]. Pipeline steps can look like black boxes where a researcher cannot comprehend its inner workings or even which tools are being executed. DolphinNext is a user-friendly platform aiming to increase the reproducibility and portability of analysis pipelines. By providing a powerful yet simple User Interface (UI), understanding and changing pipelines is no longer limited to technically proficient users. Besides that, pipelines can be extremely well described and documented, with every process, used tool, and communication channel explicit and documented.

DolphinNext is also very versatile in the sense that it supports execution across multiple environments, in physical workstations or the cloud, through the Nextflow layer. This is important because an analysis pipeline can be executed in the environment where the data is located, which means that the data does not need to be transferred across systems or locations. Besides the noticeable efficiency gains, this also has benefits regarding preserving data privacy. This is possible since DolphinNext is, in essence, a wrapper platform that builds on top of Nextflow and leverages all of its features while possessing other vital capabilities. In summary, as shown in Figure 3.2, Nextflow offers execution of pipelines through HPCs, Cloud providers or workstations, where pipelines can be executed in containerised environments (through Docker or Singularity). However, DolphinNext serves a layer on top of that, providing a *Pipeline Builder* UI, a Job monitoring page, a revisioning system for pipelines and processes, and modular development. Besides that, pipelines can easily be shared and reproduced, together with a set of reports that are also custom generated.

By sharing pipelines, researchers can not only collaborate with others but also get inspiration and novel views of analysis processes, methodologies or implementations of tools. Since most of the most common analysis tools are shared between laboratories and bioinformatics pipelines, processing chunks are also commonly very similar or even identical. Therefore, by modularising

[4] https://dolphinnext.readthedocs.io/en/latest/

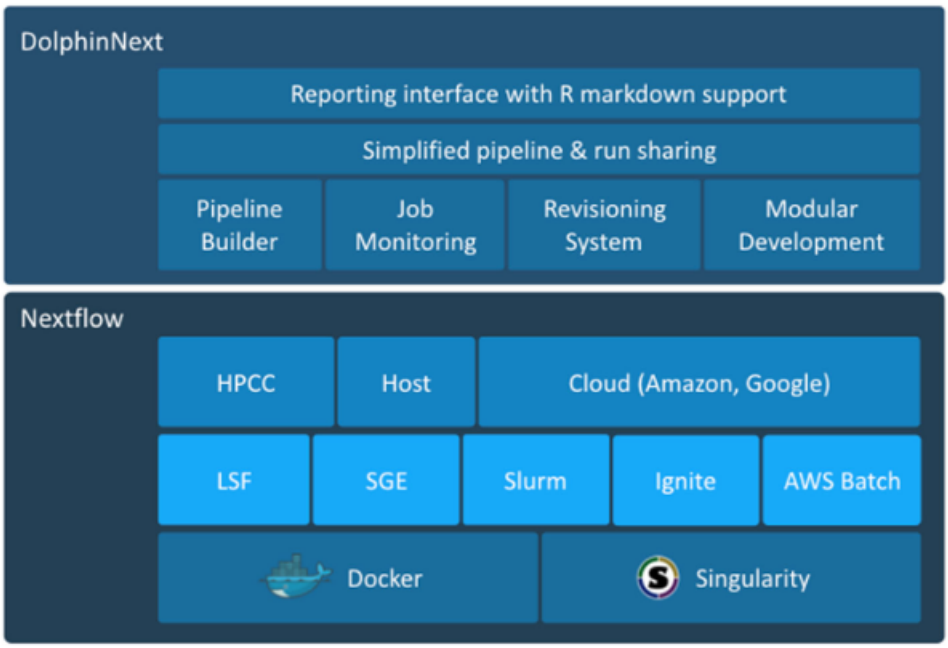


Figure 3.2: DolphinNext architecture. From [34]

pipeline processes and ensuring the shareability of analysis blocks, researchers can benefit significantly from collaborative work. For this reason, DolphinNext supports creating modules which can be reused across pipelines.

In order to tackle the various current main concerns regarding reproducibility and shareability of bioinformatics pipelines, DolphinNext is composed of different modules, or parts, which focus on different components of the platform.

3.4.1 *Pipeline Executor* module

The *Pipeline Executor* module focuses on the actual execution of pipelines in a specific and previously configured environment. In order to provide a practical and efficient executor module, DolphinNext focuses on the current main bottlenecks and limitations to bioinformatics pipeline execution. Perhaps the most common in-demand feature is in managing an execution's workflow, that is, controlling the current status of a pipeline's execution (such as pausing, resuming and cancelling executions). This is a widespread scenario since various errors can be thrown when a pipeline is executed, from permission errors (relatively common when a pipeline is executed across multiple environments) to specific computational errors from the used analysis tools. Given the extreme computational effort required by most large bioinformatics pipelines, restarting execution from scratch every time an error occurs would be very costly in time and resources.

Most execution frameworks do not have unique mechanisms to account for these issues. However, Nextflow has a well-tailored resuming functionality, allowing any pipeline to be resumed after an errored state. Since DolphinNext pipelines are translated into Nextflow code, which is

then executed through the Nextflow driver, it is essential that DolphinNext can offer such resuming mechanisms through the UI and then perform the necessary instructions at the Nextflow level. Therefore, DolphinNext provides a user interface where users can monitor execution status, resume executions when errors are thrown, and change specific parameters, which may solve issues causing the pipeline error. Regarding computational availability, users can also configure allocated resources according to current needs so that the execution occurs as smoothly as possible.

Finally, this module also provides access to execution logs (exemplified in Figure 3.3), which are necessary for execution inspecting and troubleshooting. Besides that, since pipeline execution happens at the Nextflow layer, the Nextflow-generated reports can also be easily accessed. This is very useful since the information displayed gives excellent insights into the pipeline's computational costs and also documents the processes it contains. By analysing this data, users can locate possible computational bottlenecks and further improve the pipeline's computational efficiency.

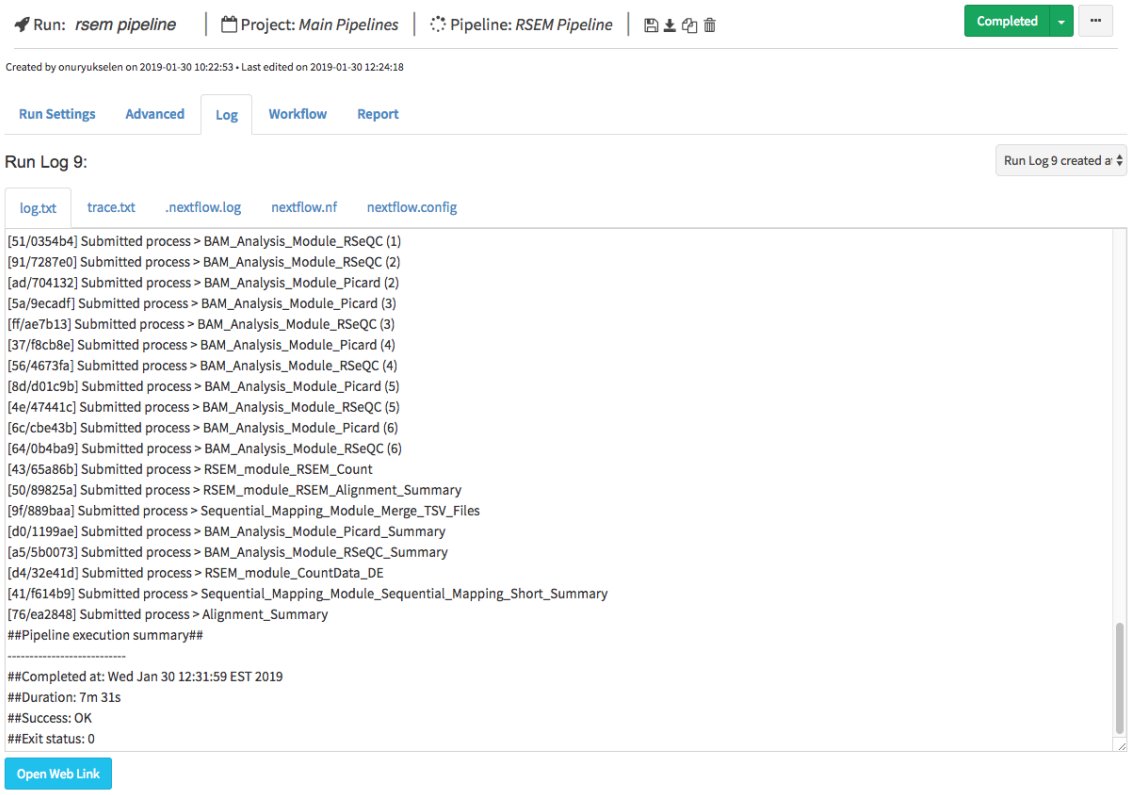


Figure 3.3: DolphinNext execution logs. From https://dolphinnext.readthedocs.io

3.4.2 *Profile* module

Commonly, a user may want to configure multiple execution environments according to computational needs, budget, availability, or other factors. For these reasons, DolphinNext contains a *Profile* Module where users can specify and configure their various execution environments across different platforms (Cloud, HPC, or even a local workstation). In Figure 3.4, the execution environment page is shown. In this configuration, a user has added a Host environment, which can be

an HPC or a workstation, and an Amazon environment, which constitutes an EC2 instance where pipelines can be executed. When a pipeline is executed, such configurations are transcribed into the Nextflow layer, which means that DolphinNext encapsulates the configured environments into Nextflow when a workflow is executed.

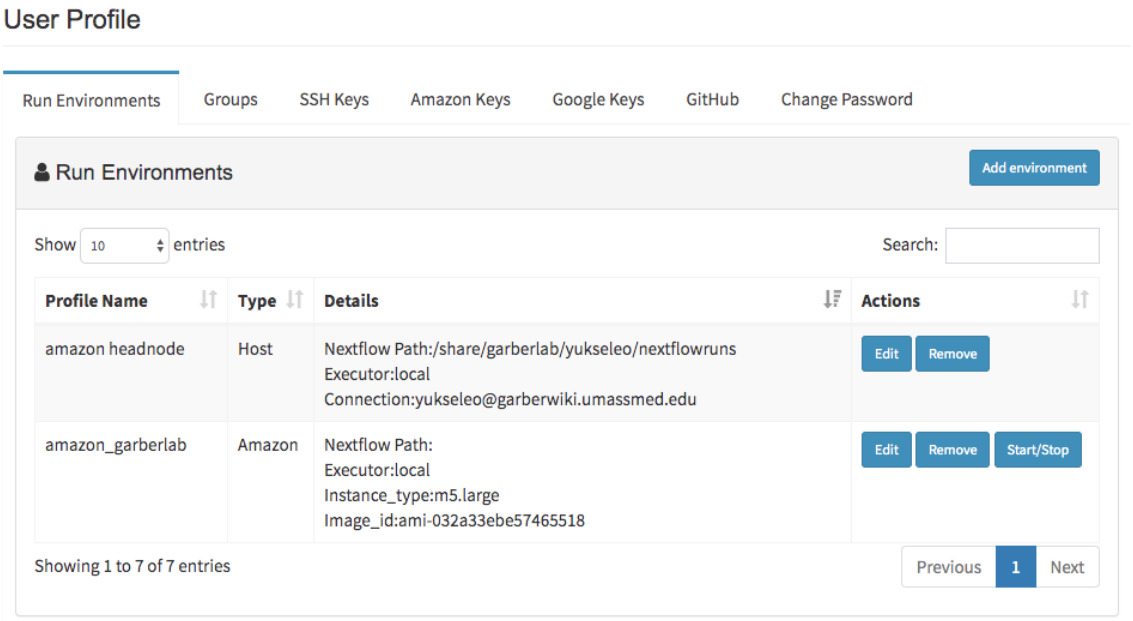


Figure 3.4: DolphinNext execution environments. From https://dolphinnext.readthedocs.io

Evidently, by providing profiling, DolphinNext also handles user authentication through various methods. By combining profiling with the configured execution environments, users with administrative rights can configure authorization policies, which is an important step beyond authentication. This allows an administrator to configure access rights to specific execution platforms, which is extremely useful in a collaborative environment. In Figure 3.5, the groups configuration page is displayed. Here, users can be added to groups with specific access and execution rights. As long as a given user has permissions, it is possible to add and remove users from groups and create and delete groups as a whole. Once again, this also simplifies execution platform configuration and allows inexperienced users to use such platforms on a single page easily.

3.4.3 *Pipeline Builder* module

Regarding Nextflow, it is clear that the main barrier to pipeline reproducibility and shareability is that it requires learning the provided DSL to develop the pipeline as a whole. The increase in complexity of bioinformatics pipelines can become a significant issue since it complicates pipeline maintenance, iteration and collaboration.

The *Pipeline Builder* module aims to tackle this issue directly by providing a user interface in which users can entirely create, change and maintain pipelines from scratch. Through this feature, users can create processes, the building blocks of a pipeline in which a particular set

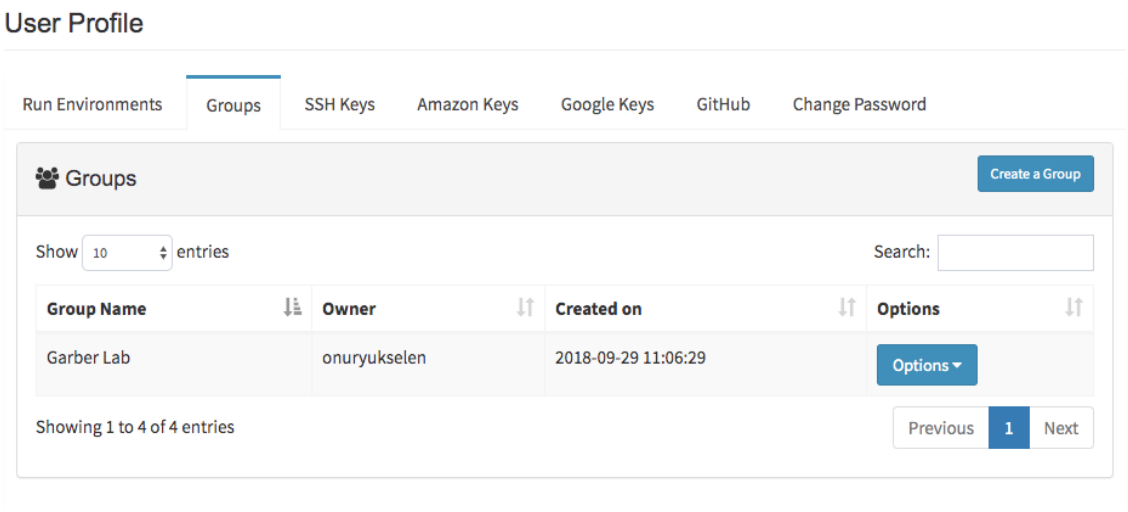


Figure 3.5: DolphinNext user groups. From https://dolphinnext.readthedocs.io

of self-contained computations are made. Processes can have a set of inputs and outputs, representing attributes (such as variables or files) or channels that allow for communication with other processes. These channels serve as data streams and are a very efficient and intuitive way of passing information through processes. Users can also select input and output parameters of a given pipeline with a specific type associated. This allows for type checking when two given processes are connected, ensuring a channel between processes can only be built if the types are consistent.

Users can insert specific information in the UI when creating a process. For instance, a set of inputs and outputs can be specified according to existing parameters (which are also configurable). Further configurations can be made to create inputs and outputs, such as declaring them as optional and creating operators that will be applied to perform transformations in the data. Another fundamental block of a process is the script, which represents the set of computations to be executed. DolphinNext supports scripts to be written in a variety of languages — Shell scripting, Groovy, Python, Perl, Ruby and R. This way, it is possible to store all the scripts of a specific pipeline in a given location and then execute any single or multiple files from a Shell script specified in a pipeline, allowing for very clean scripts to be created and eases development. Figure 3.6 displays the page for configuring the mentioned blocks of a process.

Since DolphinNext uses Nextflow's computational framework, the computations performed are highly efficient. Besides leveraging parallel execution whenever possible, "whenever two processes are connected, a dependency is implicitly defined whereby a process that consumes the output of another only runs once this output is generated" [34].

When creating a pipeline, users can select from a set of previously created processes and add them to a new pipeline accordingly, through drag and drop functionalities. The *Pipeline Builder* module also offers a revisioning system which is extremely useful since it allows for creating slightly different versions of pipelines and tracking changes across them. For instance, different revisions of a pipeline could be created for different pipeline implementations that use different versions of a specific tool. Besides creating different versions, users can also edit, delete or even

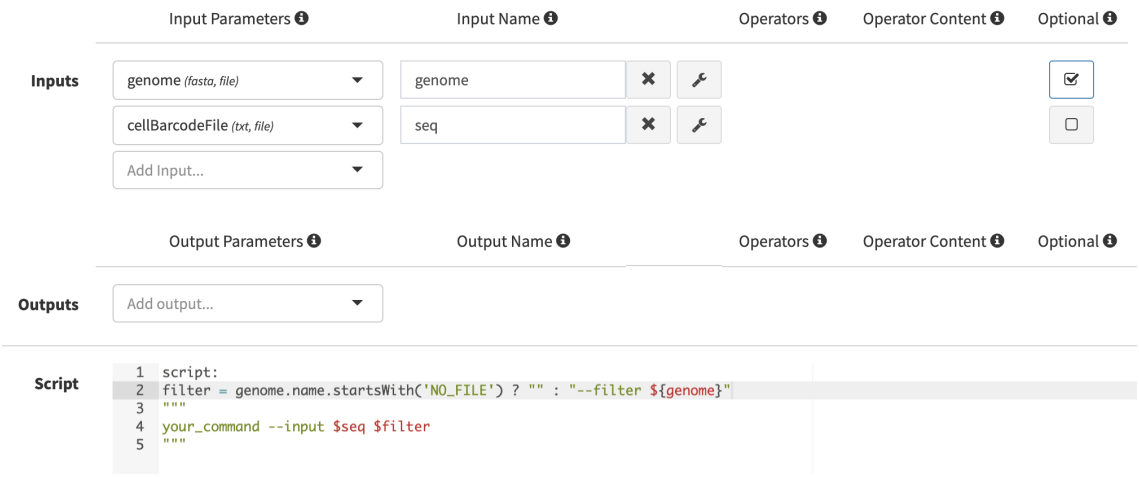


Figure 3.6: Dolphinnext configuration page for a specific process. From https://dolphinnext.readthedocs.io

share them with other researchers. For this reason, it is also possible to efficiently execute different versions of a pipeline and evaluate the difference in results, enhancing pipeline comparability. Finally, this module can generate a PDF document that displays the created workflow, which facilitates documenting a pipeline.

Another great feature this module provides is native integration with version control systems, namely Github[5]. This way, pipelines can be more easily shared across researchers and laboratories. However, a DolphinNext pipeline can also be downloaded in a Nextflow format and then imported or exported for sharing pipelines easily if the more common version control is not an option. Besides that, it also seemingly integrates with CI/CD frameworks, namely Travis-ci[6], a primary CI platform for automated testing. Given the complexity of bioinformatics pipelines, this is a convenient feature since it automates testing in created pipelines.

Figure 3.7 showcases what a straightforward process created in this module could be. The larger circle represents a *Build_index* process, which expects one input (represented by the small red circle on the process's outer ring) and an output (represented by the small blue circle on the process's outer ring). Input and output objects can also be present in a workflow, where inputs are yellow circles and outputs are green circles. This colouring scheme is shared across the entire module to allow for better visualisation. If a given input or output in a workflow is of the same type as the expected input or output of a specific process, they can be linked, and a channel is created. Figure 3.8 shows an example of such type verification. This is a useful feature since it ensures that pipelines can remain consistent during development, avoiding possible inconsistency issues at runtime.

In Figure 3.9, an RNA-sequencing pipeline, a public pipeline from DolphinNext, is displayed. This pipeline was built through the *Pipeline Builder* module and is a good demonstration of the core components of this module. The larger blue circles represent modules, which are a set of

[5] https://github.com/
[6] https://travis-ci.org/

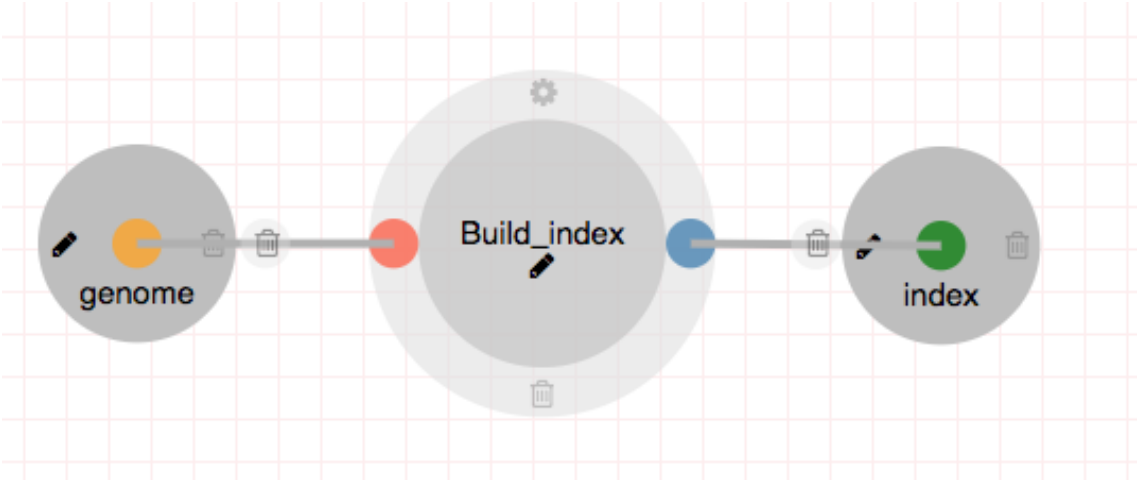


Figure 3.7: Simple Dolphinnext process with an input and output objects. From https://dolphinnext.readthedocs.io

related processes that execute a specific computational step of a pipeline. They can be reused across pipelines and may have inputs (smaller red circles on module ring) and outputs (smaller blue circles on module ring) specified, which have a specific type. Processes of the pipeline are the medium-sized grey rings, which also have inputs and outputs associated. Finally, the smaller circles represent inputs (in yellow) or outputs (in green) of a process or module. Inputs can be inserted into the pipeline execution page (either by writing the actual values or selecting paths to input files). In contrast, outputs can be visualized in many ways, as will be described in Section 3.4.4.

3.4.4 *Reporting* module

The generated results from pipeline executions must be readable and promptly ready for display to the user. For this reason, DolphinNext comprises a module dedicated to generating clear and informative reports, which can be flexible to fit a user's needs.

A convenient feature that this module has to offer is that DolphinNext allows users to visualise interim data, that is, data generated from intermediate steps of a pipeline. Such interim data is usually unnecessary in the overall reporting of a pipeline and is only significant for inter-process communication. However, in many cases, some helpful analyses can be made by analysing such data. The reporting module offers a set of formats where data can be visualised. The primary reporting files which are generated are:

- *timeline.html*: indicates the execution times of each process of the executed pipeline, as displayed in Figure 3.10
- *trace.txt*: indicates more specific execution information for each task of the pipeline, including execution status and resource usage, as displayed in Figure 3.11
- *.nextflow.log*: constitutes the logging file that Nextflow provides, as displayed in Figure 3.12

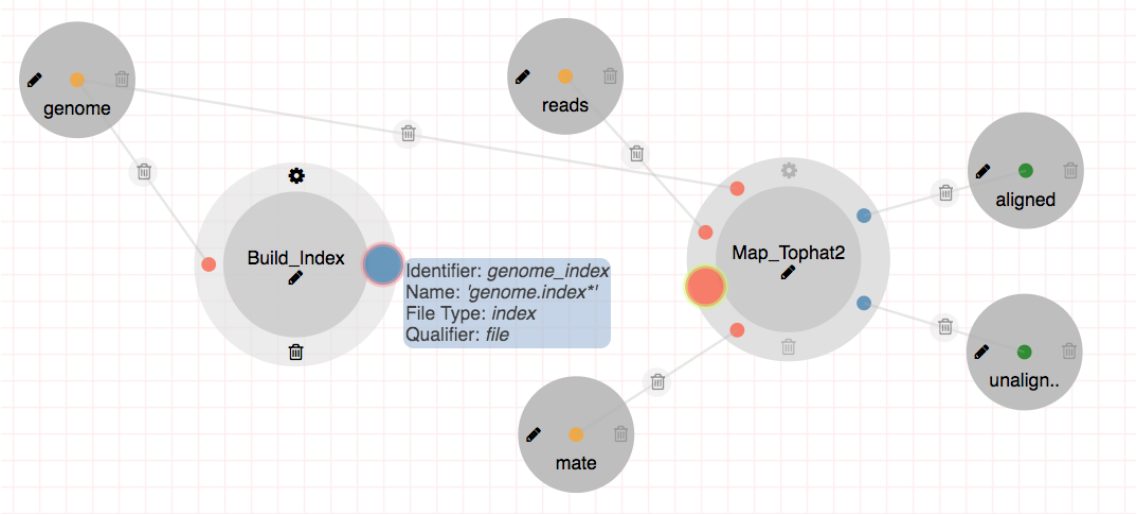


Figure 3.8: Dolphinnext attribute typing verification. From https://dolphinnext.readthedocs.io

- *nextflow.nf*: constitutes the Nextflow file that DolphinNext generates from the *Pipeline Builder* module, which is in turn executed by the Nextflow driver, as displayed in Figure 3.13
- *nextflow.config*: represents the config file that is generated from the different configurations inserted by the user when creating a pipeline, as displayed in Figure 3.14

Besides these standard output files, users can generate different reports from a pipeline execution, which will all be displayed in the Reports tab of the pipeline execution page, as shown in Figure 3.15. These can come in various formats, such as R-Markdown (Figure 3.16), Datatables (Figure 3.17), HTML (Figure 3.18) or PDF (Figure 3.19).

3.4.5 DolphinNext API

DolphinNext also provides an API for fetching existing executions and creating new ones. Requests must contain a bearer that identifies existing users to be valid. The entire API specification will not be given. However, a sample request for fetching all of the runs for an authenticated user is described in Listing 3.2. An example response to such a request is displayed in Listing 3.3.

```
1   $ curl -X GET 'https://dolphinnext.umassmed.edu/api/service.php?data=getRuns' \
2 -H 'Authorization: Bearer <your-access-token>'
```

Listing 3.2: Example Dolphinnext API request

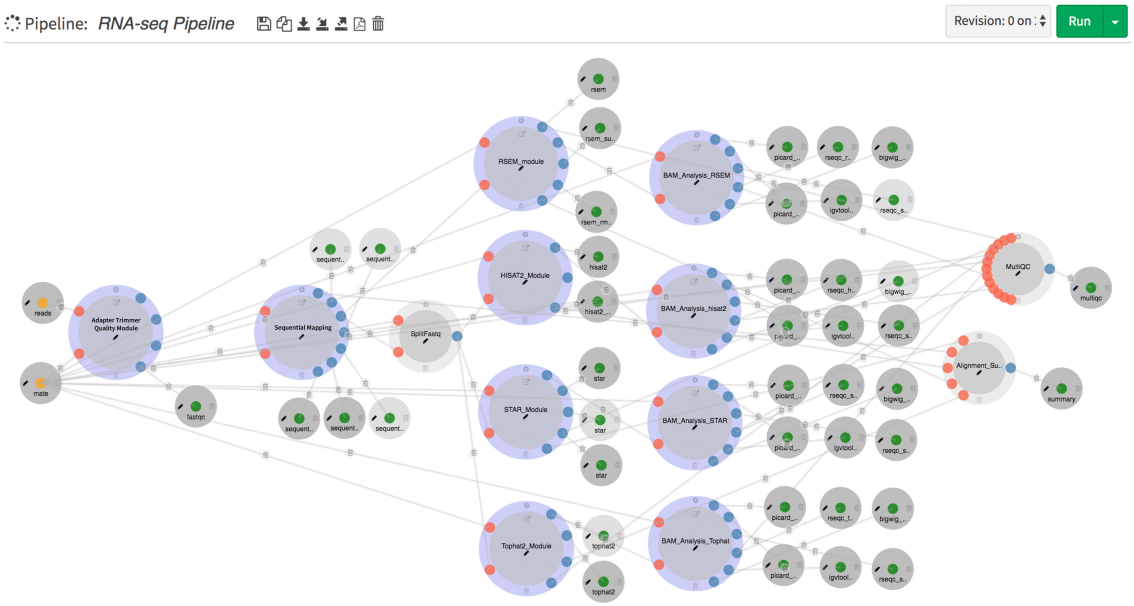


Figure 3.9: DolphinNext RNA-seq pipeline. From https://dolphinnext.readthedocs.io

```
{
    "status": "success",
    "data": {
        "data": [
            {
                "name": "Build index (amazon)",
                "_id": "234"
            },
            {
                "name": "UMIextract",
                "_id": "249"
            },
            {
                "name": "test",
                "_id": "250"
            }
        ]
    }
}
```

Listing 3.3: Example Dolphinnext API response

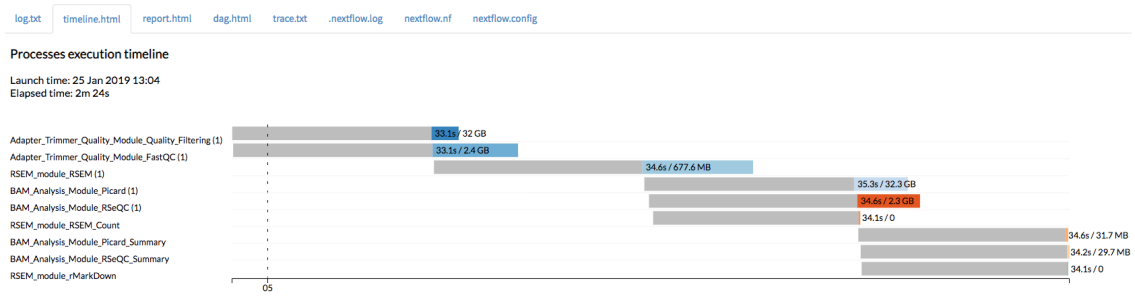


Figure 3.10: *timeline.html* file. From https://dolphinnext.readthedocs.io

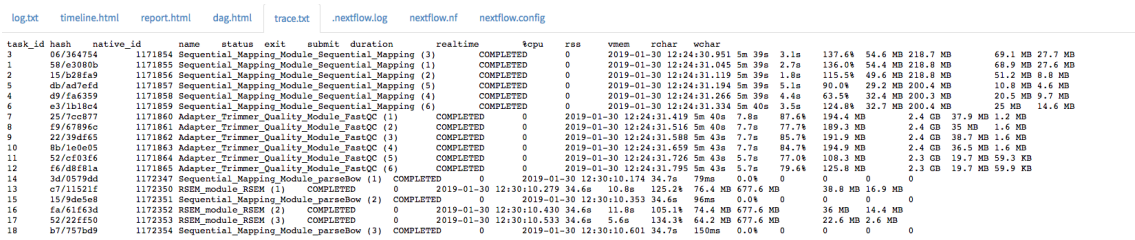


Figure 3.11: *trace.txt* file. From https://dolphinnext.readthedocs.io

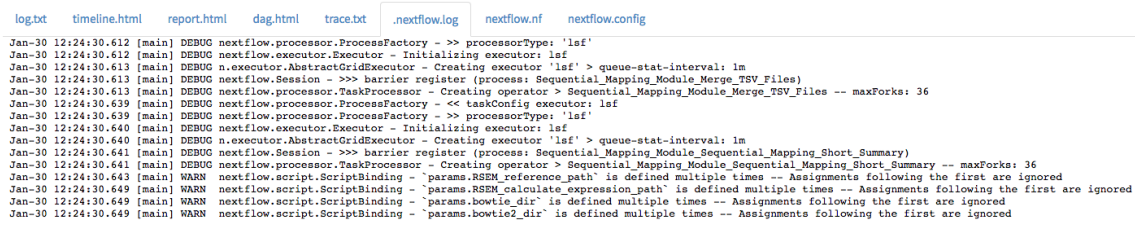


Figure 3.12: *.nextflow.log* file. From https://dolphinnext.readthedocs.io

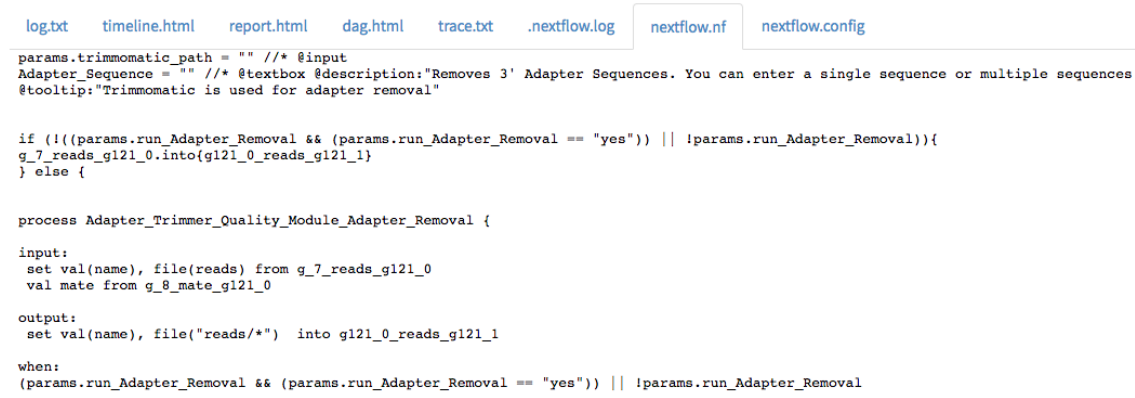


Figure 3.13: *nextflow.nf* file. From https://dolphinnext.readthedocs.io

Figure 3.14: *nextflow.config* file. From https://dolphinnext.readthedocs.io

PROCESS	PUBLISHED DIRECTORY	VIEW FORMAT
Multiqc	Multiqc	HTML
Alignment Summary	Summary	Table
Rsem Module	Rsem Summary	Table
Rsem Module	Rsem Summary	DE-Browser
Rsem Module	Rsem Rmarkdown	R-Markdown

Figure 3.15: Generated reports from an execution. From https://dolphinnext.readthedocs.io

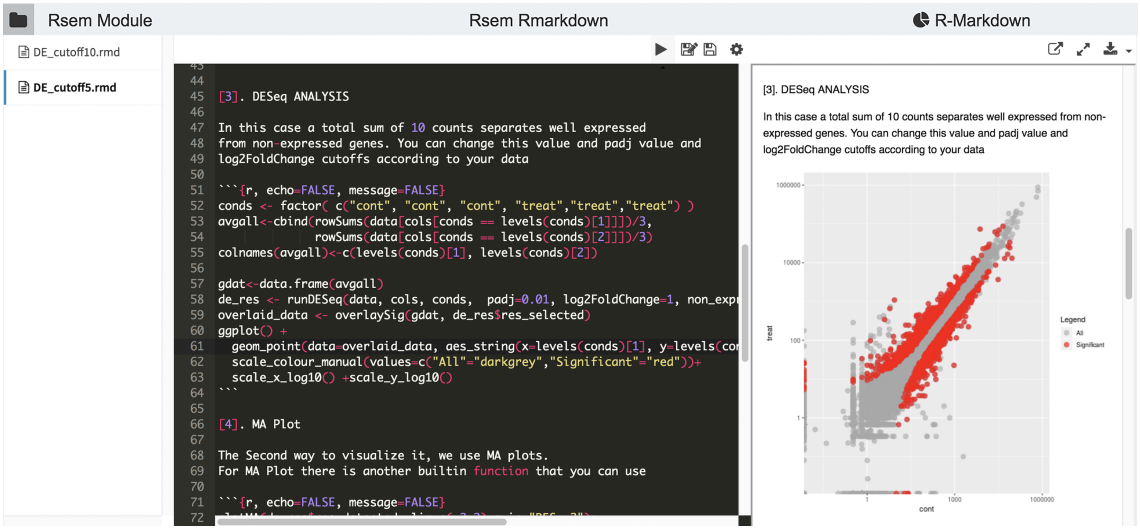
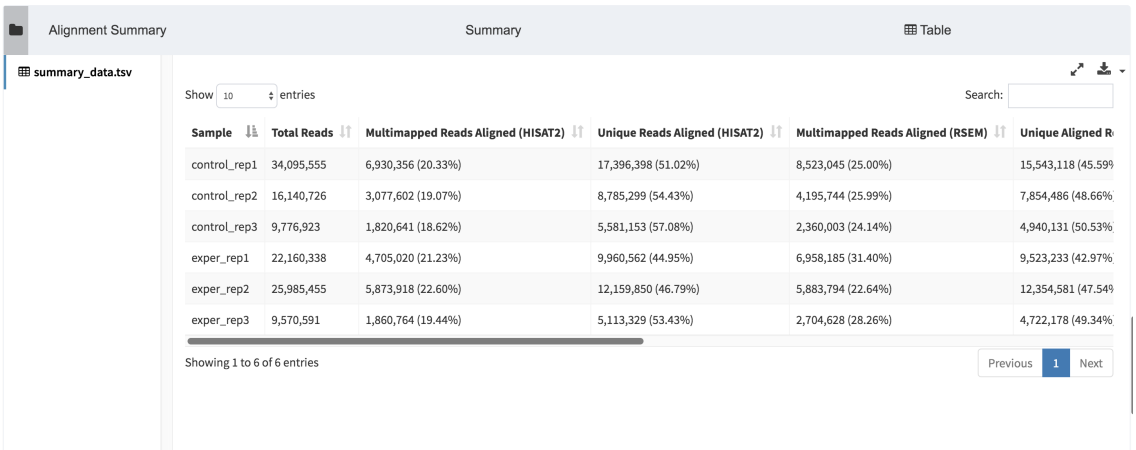


Figure 3.16: Generated R-Markdown report. From https://dolphinnext.readthedocs.io



Sample	Total Reads	Multimapped Reads Aligned (HISAT2)	Unique Reads Aligned (HISAT2)	Multimapped Reads Aligned (RSEM)	Unique Aligned R
control_rep1	34,095,555	6,930,356 (20.33%)	17,396,398 (51.02%)	8,523,045 (25.00%)	15,543,118 (45.59%
control_rep2	16,140,726	3,077,602 (19.07%)	8,785,299 (54.43%)	4,195,744 (25.99%)	7,854,486 (48.66%
control_rep3	9,776,923	1,820,641 (18.62%)	5,581,153 (57.08%)	2,360,003 (24.14%)	4,940,131 (50.53%
exper_rep1	22,160,338	4,705,020 (21.23%)	9,960,562 (44.95%)	6,958,185 (31.40%)	9,523,233 (42.97%
exper_rep2	25,985,455	5,873,918 (22.60%)	12,159,850 (46.79%)	5,883,794 (22.64%)	12,354,581 (47.54%
exper_rep3	9,570,591	1,860,764 (19.44%)	5,113,329 (53.43%)	2,704,628 (28.26%)	4,722,178 (49.34%

Figure 3.17: Generated Datatables report. From https://dolphinnext.readthedocs.io

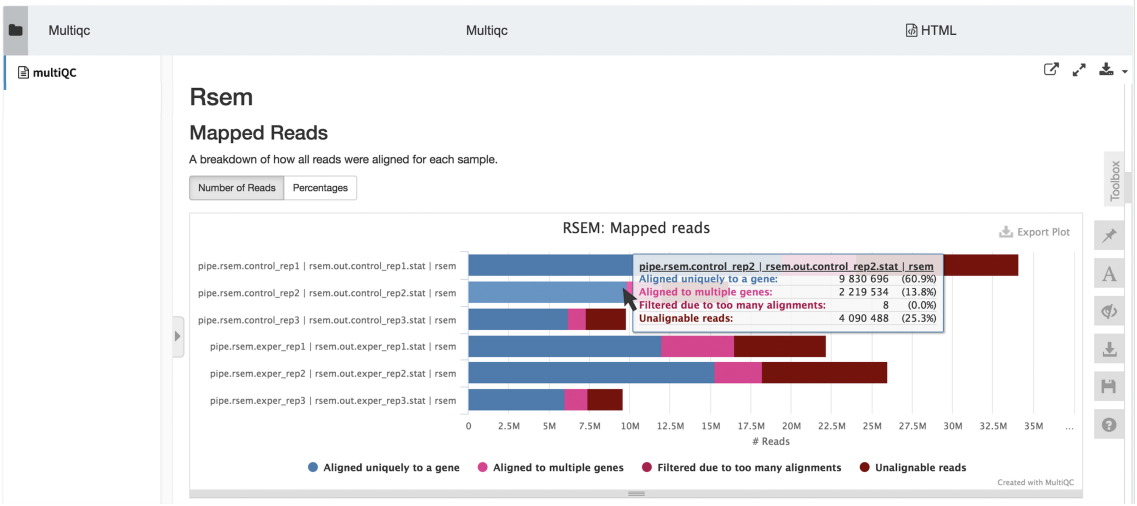


Figure 3.18: Generated HTML report. From https://dolphinnext.readthedocs.io

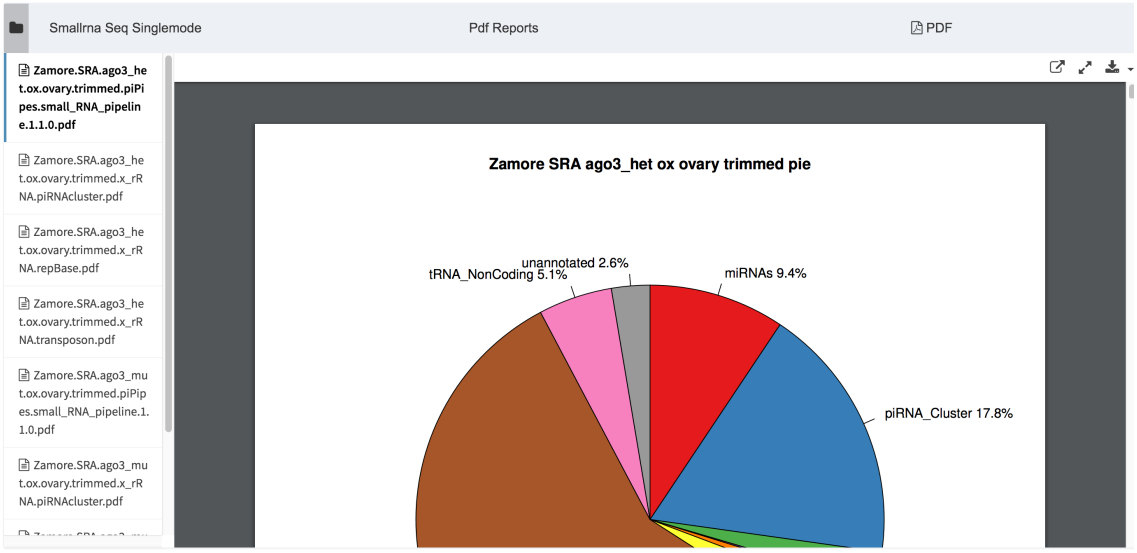


Figure 3.19: Generated PDF report. From https://dolphinnext.readthedocs.io

Chapter 4

Proposed Solution: A Platform for Reproducible Pipelines

In order to fully understand the current limitations of the leading frameworks offering reproducible bioinformatics pipelines, it is crucial to recreate an execution of an external pipeline and analyse the main issues and missing features in said platforms. This proposal consists of reproducing a preprocessing pipeline in DolphinNext, in a local workstation, recreating a real-world use case in which a pipeline is shared between researchers. When performing said implementation, improvements and solutions for some present issues will also be proposed to further contribute to the capabilities of DolphinNext. In Chapter 5 the details of this implementation will be further described.

4.1 Reproducibility model

In Section 2.4.1, the AIRR Data Model was introduced. This model explains how data should be collected and documented, mainly at an experiment level. However, with the increasing complexity of bioinformatics pipelines, extending the standardisation and best practices established in the data analysis domain has become increasingly important. To this effect, a complete data analysis reproducibility model is proposed in Appendix A.

This model is heavily based on the MiAIRR Data Model regarding the data descriptive entities. However, it was extended to cover and detail the most crucial analysis objects so that the analysis execution process can be better standardised. Since Nextflow has become increasingly relevant and familiar for analysing NGS data, the entities described in this model are also based on some of the most crucial Nextflow components. This ensures that researchers can more easily document their pipelines and implement the standardization model directly through Nextflow pipelines.

Table 4.1 briefly describes all of the entities represented in the proposed model present in Appendix A.

Table 4.1: Reproducibility model classes description

Entity	**Description**
Subject	Information about the study cohorts and individual subjects, including species, sex, age, and ancestry
Repertoire	Represents a set of samples collected from a given individual, at a specific point in time
Study	Information about the experimental study design, including the title of the study, laboratory contact information, funding, and linked publications
Species	Group of individuals that carry reproducibility capabilities between them
GermlineDatabase	Described the database in which germline information is present, which is often used in specific analysis tools
Sample	Information about the origin and expected composition of the biological sample
WorkflowInput	Defines the expected input of a workflow as a whole
UMI	Constitutes an index that provides error correction and increased accuracy during sequencing
Read	Sequence of base pairs from a DNA fragment
Primer	Describes the primer used in a given database collection
ComputationalPrimer	Computational information that has the effect of a primer, in the data analysis stage
Workflow	Represents a complete bioinformatics pipeline
WorkflowOutput	Represents the expected output of a workflow as a whole
ProcessingReport	Generated report for a pipeline execution
File	Represents a single file which can have any extension
Configuration	Describes a configuration instance, which can be a file or attribute
Process	Single self-contained computational block
Executor	Execution environment
Directive	Optional settings for executing a process
Condition	Optional condition which must be satisfied for a process to be executed
Script	Computational block which is executed
Command	Individual instruction to be executed in a scrips
Tool	Software program which is used with a specific purpose
ProcessInput	Expected input of a process
Process output	Expected output of a process
Channel	Data stream where information can be carried between processes
Operator	Methods for filtering or transforming information exchanged between processes
DataProcessing	Information about the raw sequencing processing
ReadingTechnique	Technique used in the collection of Reads

4.2 Current limitations in Nextflow

With the release of Nextflow, researchers were given a framework for creating very reproducible pipelines in fully containerised environments. This was a significant step toward reproducibility since a pipeline could be executed across different machines without any previous configurations required. However, in reality, many other factors contribute to the reproducibility of a pipeline which may not be directly associated with its execution. Even if a pipeline is fully modularised and packaged into a container (which can very quickly be deployed without any further configuration) it is vital that the processes, tools, and parameters are well described and documented, given that different releases of a single tool can lead to reproducibility unnoticeable issues [15]. The fact that an entire pipeline is encapsulated into a containerised environment can be counter-productive since it hinders the ability to describe it accurately. Nextflow processes, workflows or even complete pipelines can commonly feel like black boxes to researchers trying to reproduce them. Currently, this is one of the main concerns regarding the reproducibility of pipelines that Nextflow does not address.

The specific Nextflow DSL syntax, despite being extremely efficient and valuable, has a learning curve. Besides being time-consuming, the main concern is that researchers unfamiliar with the technology cannot fully understand implemented pipelines, even if they can easily be fully reproduced. Besides, the fact that pipelines are built through the DSL means that maintaining them or even performing small changes can be difficult. This is also a significant issue since pipelines often suffer minor changes and iterations, or even significant changes in processing steps or, more specifically, the tools used. Difficulties in maintenance can hinder the quality of the analysis being made. The lack of a graphical interface is another main downside in useability since pipelines require extensive documenting, which proves to be difficult and time-consuming since various tools and dependencies are used.

Therefore, the concept of reproducibility should be extended to encapsulate the ability to be not only duplicated but also understood by an external researcher. This is of extreme importance since it dramatically facilitates collaboration between laboratories and researchers in the sense that researchers can benefit from other tools or analysis methodologies being used by others. By facilitating the exchange of such information, it is expected that new analyses will be performed across different datasets and in novel applications, which can improve the studies being made.

For these reasons, there is a clear need for a platform that addresses the main limitations of Nextflow while leveraging all of its main features concerning computational efficiency and reproducibility. DolphinNext can be a practical option since it acts as a layer on top of Nextflow, offering features that tackle some of the issues described above. In order to understand if DolphinNext is fit to serve as a solution for such needs, a set of requirements for a platform which offers fully reproducible creation, management and execution of bioinformatics pipelines is described in the next section.

4.3 Requirements of a Platform for Reproducible Pipeline

After pointing out the current limitations of data analysis frameworks, it is important to understand the possible optimizations and improvements in pipeline reproducibility. To this effect, a set of research questions which should be answered can be defined:

- **RQ1**: *How can a bioinformatics pipeline be described?*
- **RQ2**: *What factors truly contribute to a pipeline's reproducibility?*
- **RQ3**: *How can containerization and orchestration be leveraged to improve reproducibility?*
- **RQ4**: *How can continuous integration tools be leveraged to improve workflow management?*
- **RQ5**: *How viable is cloud computing in bioinformatics analysis, both in computational capabilities and cost?*
- **RQ6**: *How shareable can pipelines be even in a completely portable environment?*
- **RQ7**: *How can we compare executions and calculate output similarity?*

There are currently numerous frameworks and platforms which offer reproducible pipeline execution. To better understand the current limitations these platforms contain, it is essential to collect a set of features that these platforms should offer.

In order to improve shareability and implement an authorisation later, platforms must offer access control so pipelines can be shared and modified as expected. To this effect, a set of actors representing the platform's different users can be defined, given their expertise level, domain and responsibility. By creating these different roles and user profiling, user permissions can be delimited to a fine grain, easing collaboration and building safety mechanisms into the pipelines. A set of users of a reproducibility platform is proposed in Table 4.2.

In order to fully understand the required features of the reproducibility platform, a set of user stories can also be defined, as shown in Table 4.3. Even though it is hard to summarize the required features of such a complex platform in a single table, the proposed user stories can represent the core features which focus on describing and documenting bioinformatics pipelines, as well as facilitating collaboration between researchers wishing to execute different bioinformatics pipelines across a variety of environments.

It is evident that the requirements do not share equal levels of importance. To this effect, the Priority score of the User Stories is a good way of evaluating how crucial they are, given that some are not as significant as others.

4.4 Methodology

After establishing these requirements, it is clear that DolphinNext provides a solution for most of the current reproducibility issues, mainly through its Pipeline Builder module. However, it is still

Table 4.2: Actors for pipeline reproducibility platform

Identifier	**Name**	**Job**	**Notes**
A1	System developer	Designs the entire system	Should be a knowledgeable software engineer
A2	Process developer	Builds individual processes or modules which can then be combined into a pipeline	Not necessarily an engineer, but probably should have some related knowledge. Must have an understanding of the tools being implemented in the created process
A3	Pipeline developer	Combines processes into a pipeline	Must understand the expected data analyses as a whole. Should not need to be a software engineer. Should understand what the various parameters and tools compute
A4	Pipeline user	Executes the pipeline. Creates a complete pipeline and respective configuration which can be simply executed	Should use an interactive interface and understand what the pipeline input parameters must be.
A5	System admin	Set up and configure pipeline running environments	Includes interfacing with Cloud providers, clusters, and others.
A6	Pipeline investigator (1)	Replicating previous work	Extension of A4
A7	Pipeline investigator (2)	Comparing outputs of pipelines	Extension of A3

Table 4.3: User Stories for pipeline reproducibility platform

Identifier	Name	Priority	Description
US101	Create process	High	As a User, I want to be able to create a new process, in order to perform self-contained and well-defined operations
US102	Create channel	High	As a User, I want to be able to create a new channel, in order to establish communication lines between processes
US103	Create input	High	As a User, I want to be able to create a new input, in order to declare an expected process input
US104	Create output	High	As a User, I want to be able to create a new output, in order to declare an expected process input
US105	Create script	High	As a User, I want to be able to create a new script, in order to perform a given computation.)
US106	Connect script to process	High	As a User, I want to be able to connect a script to a process, in order to execute a set of commands in such process
US107	Connect input to process	High	As a User, I want to be able to connect an input to a process, in order to feed data into processes
US108	Connect output to process	High	As a User, I want to be able to connect an output to a process, in order to retrieve data from processes
US109	Connect channel to process	High	As a User, I want to be able to connect a channel to a process, in order to establish communication between processes
US110	Save workflow to JSON	Medium	As a User, I want to be able to save a current workflow to a JSON file, in order to store the pipeline information
US111	Generate workflow graph	Low	As a User, I want to be able to generate a workflow graph from a JSON file, in order to visually analyse a pipeline

a recent platform which needs to be thoroughly tested with real-world bioinformatics pipelines since it has some limitations.

Implementing a reproducibility platform similar to DolphinNext, with features in the established requirements and missing from DolphinNext, would be extremely challenging and time-consuming. Since the platform is open-source, it would be a significantly better option to contribute to it or even implement the changes in a private repository and serve the improved version on a proprietary server.

For that reason, the work described in Chapter 5 will focus on implementing a bioinformatics pipeline in DolphinNext, while documenting the main hurdles of using the platform.

4.4.1 Preprocessing pipeline

In order to thoroughly test the DolphinNext platform, it is essential to implement an objective analysis pipeline and achieve consistent and reliable results, as would be possible through a plain Nextflow execution. Therefore, a bioinformatics pipeline written in the Nextflow DSL will be transcribed into DolphinNext and built through the *Pipeline Builder* module. This pipeline was written by the *Gur Yaari lab*[1] and is, in essence, a preprocessing pipeline. Its main goal is to preprocess the input data to transform it into a better format for alignment. In this implementation, the input data will be *fastq* files containing Covid samples from a given individual.

The *fastq* files which are input to the preprocess pipelines go through some transformations employed by some bioinformatics tools. Firstly, *pRESTO*[2], which is a tool for processing raw reads from high-throughput sequencing of B-Cell Repertoires (BCR) and T-Cell Repertoires (TCR) [9]. At a later stage, the *IgBLAST* tool is implemented to further analyse BCR and TCR sequences. *IgBLAST* leverages the BLAST search algorithm, a specialised algorithm that forms alignments from short common patterns in sequences [14]. Furthermore, it reports the germline V, D and J gene matches in a given sequence.

The preprocessing pipeline should be represented fully in the *Pipeline Builder* module, including every process, channel, input and output. Further, and perhaps most importantly, every analysis tool should also be implemented, considering the correct version of each software package.

Since a single pipeline will be executed in a completely different framework, this implementation will be a practical test for the reproducibility features offered by DolphinNext.

4.4.2 Comparing outputs of different pipeline executions

Some bioinformatics analysis tools introduce a certain level of non-determinism into a pipeline since their outputs can vary across executions. A pipeline's reproducibility can be an issue since the results must be consistent across different executions. Therefore, it is crucial to evaluate the consistency of a pipeline's execution across different builds.

[1] http://yaarilab.com/
[2] https://presto.readthedocs.io/

Besides that, understanding the impact that some changes in specific steps of a bioinformatics pipeline carry is crucial for its evolution, maturing and fine-tuning. Understanding how the results are affected by specific execution parameter changes, software tool version upgrades or any other changes is crucial in developing relevant and valuable bioinformatics pipelines.

For these reasons, tracking and comparing output results across different builds is crucial to get more reliable and accurate results. In order to study the output similarity of the outputs of two different DolphinNext pipeline versions, two different revisions of the preprocessing pipeline will be implemented. The different versions and their description will be discussed in Section 5.2. Finally, the results obtained from the analysis of both pipeline versions across both similarity analysis tools will be discussed in Section 6.2.

In this similarity study, the main objective is to analyse the output consistency of DolphinNext executions and to implement a common use-case of bioinformatics pipelines, in which a researcher wishes to understand the impact of a change in the pipeline output.

Chapter 5

Implementation

This chapter will present an implementation of a preprocessing pipeline in DolphinNext. The pipeline should be built fully through the Pipeline Builder module and executed in a containerised environment on a local workstation, representing a common use case for sharing and reproducing a bioinformatics pipeline.

5.1 DolphinNext implementation

In order to test the usability and relevance of DolphinNext in the reproducibility context, it is crucial to implement an actual pipeline in the platform and perform a deep analysis of the development, execution and reporting process. Besides that, it is also important to analyse different executions and achieve as consistent and reliable results as possible through a plain Nextflow execution. Therefore, the preprocessing pipeline described in Section 4.4.1 will be fully implemented in the platform through the **Pipeline Builder** module and accordingly executed and analysed.

5.1.1 Local Server deployment and pipeline execution framework

To test the portability of DolphinNext and its execution of pipelines through Docker in a local workstation, a DolphinNext local server will be deployed to a single machine, where multiple executions of the preprocessing pipeline will be performed. DolphinNext offers a pre-configured Docker image [1] which can build and deploy a local server of the platform. In order to execute a dockerised pipeline in a local workstation that is executing a DolphinNext server, some necessary configurations are necessary since Nextflow does not seem to support this feature currently. However, it supports out-of-the-box pipeline executions through Singularity, which raises some questions as to why Docker is not supported without further configurations.

As mentioned previously, this execution framework will be deployed in a local workstation, which must have Docker installed. All the containers, volumes and images necessary for this

[1] https://github.com/UMMS-Biocore/dolphinnext-studio/blob/master/Dockerfile

framework will share the same docker daemon. Since Nextflow launches Docker containers for the different computing processes of a pipeline (not necessarily in a 1-to-1 fashion), there will be a given number of containers being deployed (not all at once) according to Nextflow's job scheduler. These computational processes will be executed in containers generated from pre-configured Docker images, which should contain all the necessary dependencies for the pipelines to execute as expected. Note that it is also possible to specify different Docker images for different processes. However, this seems to be an unnecessary overhead for the relatively small pipeline which is being implemented. Some platforms, such as imcantation[2], offer previously configured Docker and Singularity images that support the most common tools and dependencies in bioinformatics pipelines.

Volumes must be configured for information to flow between the host filesystem and the containers. In this model, there are three proposed volumes to be mounted. The first one is a DIND (Docker In Docker) volume, which is necessary for a container to launch another child container. This is because a Docker container launched from another uses the parent's Docker daemon. Therefore, it is necessary to mount a volume from the Host's daemon socket into the child container. Besides that, the DolphinNext Database should also be mounted into a volume to ensure data preservation after containers are terminated. This means that user accounts, pipelines, processes, environments and groups can be reused. Finally, another volume can be mounted in order to transfer specific data for a pipeline into the pipeline containers. This ensures that data can be preserved in a privacy-preserving manner and is only accessed by pipeline-specific containers. Figure 5.1 displays the architecture for executing bioinformatics pipelines through the DolphinNext Local Server, following a DIND configuration.

The Docker stack created for developing the DolphinNext Local Server, capable of executing dockerised bioinformatics pipelines through DIND, is presented in Appendix B. The stack consists of a *docker-compose* service (consult Section B.2) which deploys a container from a preconfigured *Dockerfile* (consult Section B.1).

5.1.2 Workflow

The preprocessing pipeline was, in its original Nextflow implementation, comprised of 13 processes which follow a linear sequence. This pipeline receives, as input, *Chain* and *Subject* variables, which can be defined in the *Pipeline Builder* UI. Through these inputs, the *get_source* process fetches the corresponding reads that should be used in the pipeline. At the end of the workflow, a set of *tab* files comprise the output of the pipeline. This set of files contain specific information regarding the sequence alignments found and can be displayed in a very explicit *Datatable* format. Figure 5.2 displays this workflow created.

Among the 13 specific processes, some processes are intrinsically related and can be encapsulated into modules. This can very easily be implemented without needing to start from scratch.

[2] https://hub.docker.com/r/immcantation/suite/

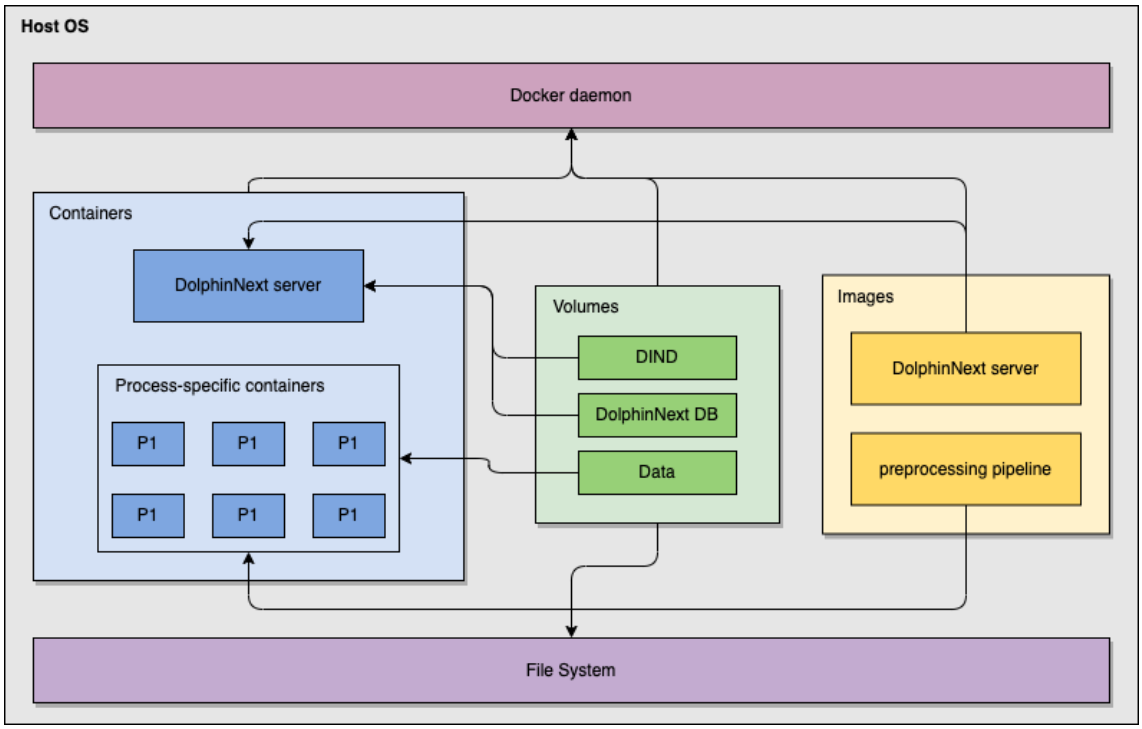


Figure 5.1: Pipeline execution framework for Docker deployments in local workstations

In order to keep both implementations, it is possible to copy the pipeline created previously into a new one, which will then be Modularized, as described in Figure 5.3.

5.2 Experimental pipeline revisions

Given the experimental nature of bioinformatics pipelines, DolphinNext must support the creation of different revisions for a single pipeline. That way, a pipeline's core infrastructure and configurations are preserved, while more minor changes can be employed (for instance, updating the version of a specific tool).

In order to test such capabilities, a new revision of the preprocessing pipeline was developed. In such revision, an execution-specific parameter for the pRESTO[3] assembling step was changed, specifically, the MAX_ERROR parameter. This value represents the maximum error rate for a valid assembly. Therefore, by increasing the value of this parameter, it is expected that the number of sequences in the output is more extensive, even if only by a minor factor.

When creating a new revision with the updated execution parameters, it was possible to duplicate the previous preprocessing pipeline so that all the previous configurations were not lost. The following implemented changes were then tracked across the new revision, even though the pre-configured execution environments could be shared across revisions. The output files from both versions were always kept separately and never contaminated, which is essential.

A short description of the two versions are described in Table 5.1.

[3] https://presto.readthedocs.io/

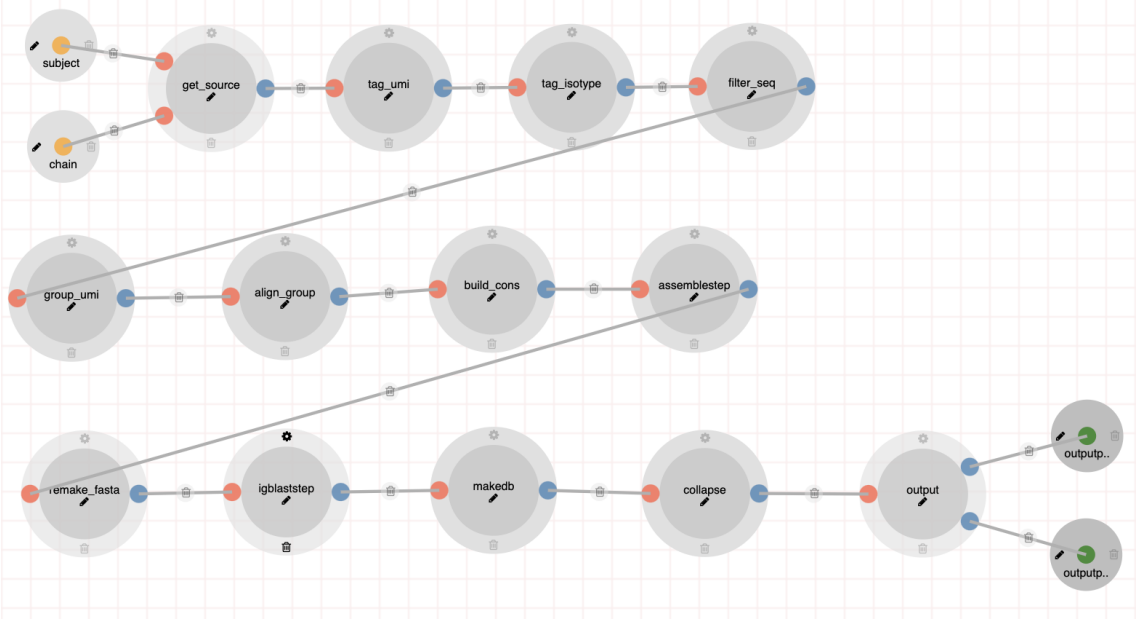


Figure 5.2: First implementation of the preprocessing workflow

In order to analyse the similarity of execution output of both pipeline versions, a set of tools will be used. Firstly, CompAIRR is a very appropriate tool in the context of the preprocessing pipeline since it is well-tailored for AIRR-seq data. Specifically, it is a command-line tool for calculating AIRR intersections and, in essence, studying the similarity of repertoires [24]. By combining different analysis methods across different vectors, it is possible to perform very detailed similarity calculations across output files and develop a good understanding of execution output consistency across a different build of a single pipeline or even of the results given by different versions of a pipeline.

Another tool, developed by the *Gur Yaari lab*, will also be used for analysing the similarity of different outputs. Even though this tool is in active development and a prototype phase, it is mature enough to help provide some conclusions. The comparison tool takes in two *csv* files and performs some operations to understand their similarity level. Firstly, the number of common attributes in the files is calculated, as well as the number of extra attributes in each one (the number of attributes only present in one file).

In Section 6.2, the results of these comparative studies will be analysed.

Table 5.1: Preprocessing pipeline versions

Pipeline Version	**Description**
preprocessing_V1	Original provided pipeline, without any modifications. In this version, $MAX\_ERROR = 0.3$
preprocessing_V2	Modified variant of preprocessing_V1. In this version, $MAX\_ERROR = 0.6$

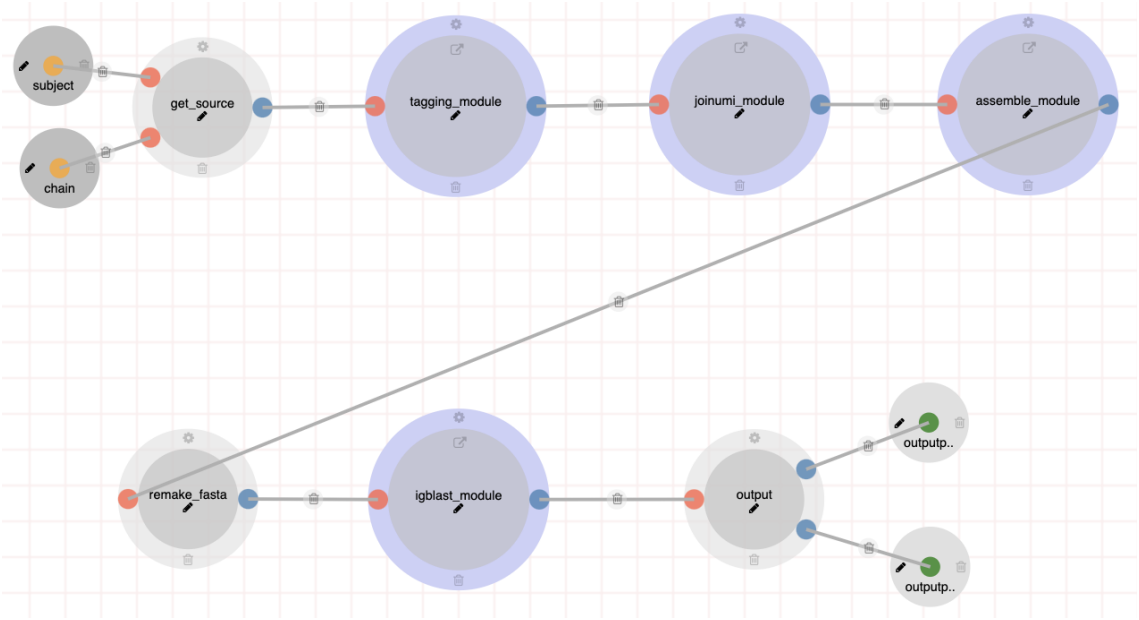


Figure 5.3: Modularized implementation of the preprocessing workflow

5.3 Versioning tool

When executing bioinformatics pipelines in containers originating from pre-configured Docker images, it can become challenging to keep track of the different versions of the used dependencies. This can hinder pipeline reproducibility since tracking versions is a crucial step toward creating well-defined and documented pipelines and ensuring their correct execution.

For this reason, there is clear value in developing a tool that outputs versions of a set of common bioinformatics tools. This tool can then be integrated into DolphinNext or any other workflow execution platform as a process. This way, the versions of all the tools are always fetched during each execution and can be outputted to a file or generated report. An implementation of such a platform is proposed in Appendix C, which currently supports *USEARCH*[4] and *MUSCLE*[5], which are tools used in the preprocessing pipeline being used in the current study. However, this can easily be extended into supporting any other bioinformatics tools.

The development and integration of this tool into the preprocessing pipeline further improves its reproducibility and documentation, which is an important feature that could also be supported by DolphinNext or any other platforms.

[4] https://bio.tools/usearch
[5] https://bio.tools/muscle

Chapter 6

Results

After completing the implementations described in Chapter 5, it is vital to study the obtained results. Firstly, the reproducibility of the pipeline executions performed will be analysed, providing an insight into how DolphinNext can further assist researchers in duplicating executions across machines and laboratories. Besides that, a deeper insight into the comparability of different pipelines will also be made. This is especially relevant since pipeline reproducibility and comparability are related, and providing an accurate and efficient comparability framework can also improve and further test a pipeline's reproducibility.

6.1 Reproducibility of the preprocessing pipeline

After studying different reproducibility platforms, it is clear that DolphinNext is the most complete. The fact that it supports every stage from a pipeline's creation to its maintenance, versioning and distribution is advantageous and relevant for the current state of bioinformatics analysis. The work implemented in this document shows DolphinNext's importance and proves its capabilities in actual pipeline development and analysis. In order to analyse the results obtained from the preprocessing pipeline implementation, some specific aspects of the implementation can be reviewed in greater detail.

6.1.1 Pipeline versioning

Different versions of the preprocessing bioinformatics pipeline were implemented by making slight changes in the original pipeline. The first approach was transcribing a Nextflow pipeline into DolphinNext from scratch through the *Pipeline Builder* module. Even though this first implementation was fully functional, it was not modularised, which meant that blocks of processes which belong to the same processing stage were not grouped. By modularising related processes, larger computational blocks were created, improving readability and allowing for future reusability across other pipelines. For these reasons, a second, modularised pipeline was created.

Given the complex nature of bioinformatics tools, coupled with the extensive number of tools being implemented, it can be difficult to fine-tune their specific execution parameters to better fit the user's needs. To evaluate the overhead of making such changes in DolphinNext, another version of the preprocessing pipeline was implemented. This third version pipeline had a change in the assembly step, where the maximum error rate for an assembly to be considered valid was increased. By implementing such change, it was expected that the number of sequences in the output.

In Section 6.1.2, the execution outputs across the different implemented versions will be evaluated.

6.1.2 Output consistency

All the different executions of the preprocessing pipeline were performed in Docker containers on a local workstation, through the Local Server framework proposed in Section 5.1.1.

A deeper look into the generated Nextflow code showed that is is not very readable and can be slightly confusing for inexperienced users. However, since DolphinNext provides an abstraction layer over Nextflow, this does not represent a significant issue. Since the Nextflow DSL is extensive, there are multiple different ways of implementing a specific bioinformatics pipeline which will output the same results. However, DolphinNext seems to generate reliable and well-formatted code, which is crucial to ensure that pipelines execute as expected. Even though the original Nextflow code for the preprocessing pipeline is different to the code generated by DolphinNext, the output results were an exact match across multiple different execution. This indicates that the preprocessing pipeline is deterministic, which was to be expected since it is deployed in Docker containers and does not implement ML tasks which may be non-deterministic.

Both the non-modularised and the modularised versions of the preprocessing pipeline provided equivalent output files, which were also an exact match.

The third version of the pipeline, with the change in the allowed error rate in preprocessing tasks, also had expected outputs, given that there were a greater number of sequences compared to the first and second implementations of the pipeline.

6.1.3 Generated reports

In general, the execution reports generated by DolphinNext and Nextflow were very useful for better understanding the limitations of the implemented pipeline.

By understanding the computational strain that the different processes put on the execution machine, researchers can identify possible bottlenecks and change the execution machines to better fit the computational requirements accordingly. Figure 6.1 demonstrates the Resource Usage section of the Nextflow report, where the *igblaststep* process was found to be very computationally expensive.

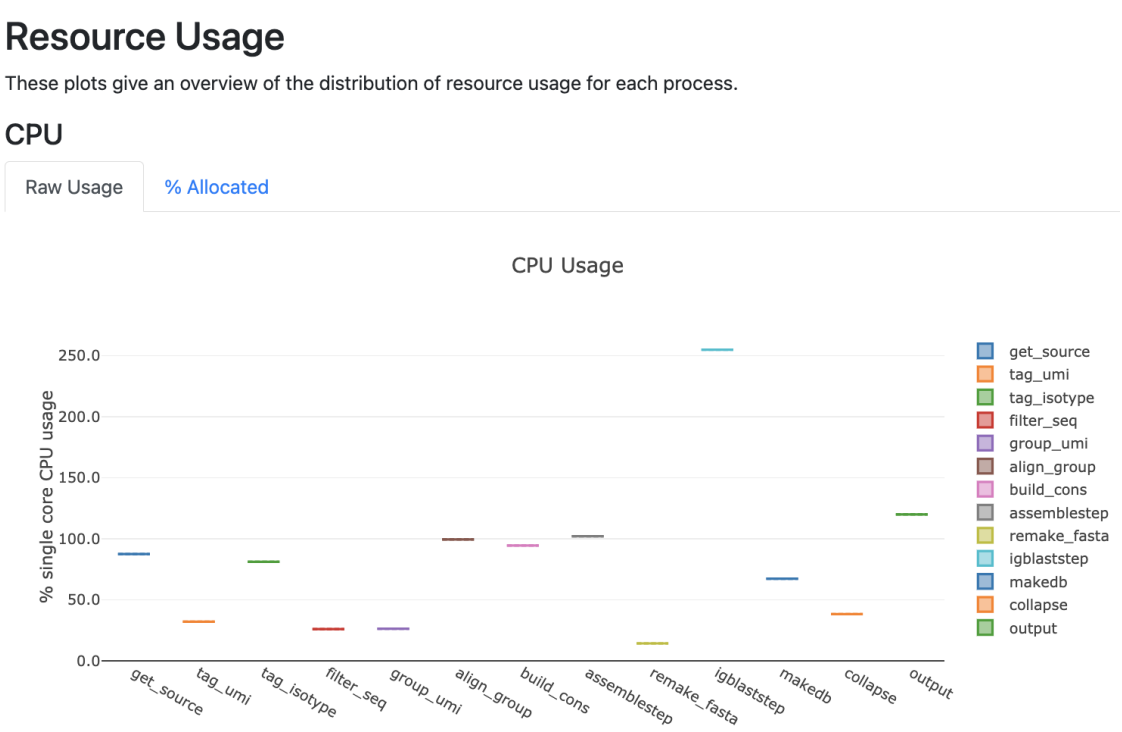


Figure 6.1: Resource usage

Similar conclusions can be drown from analysing the execution times of each process. In Figure 6.2, the execution times of each process are shown. It is evident that the assembly module is executing in over 17 hours, which is extremely long.

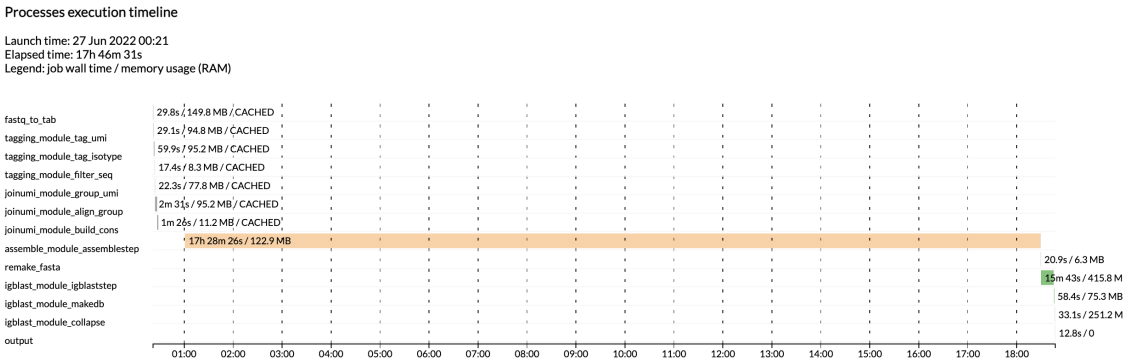
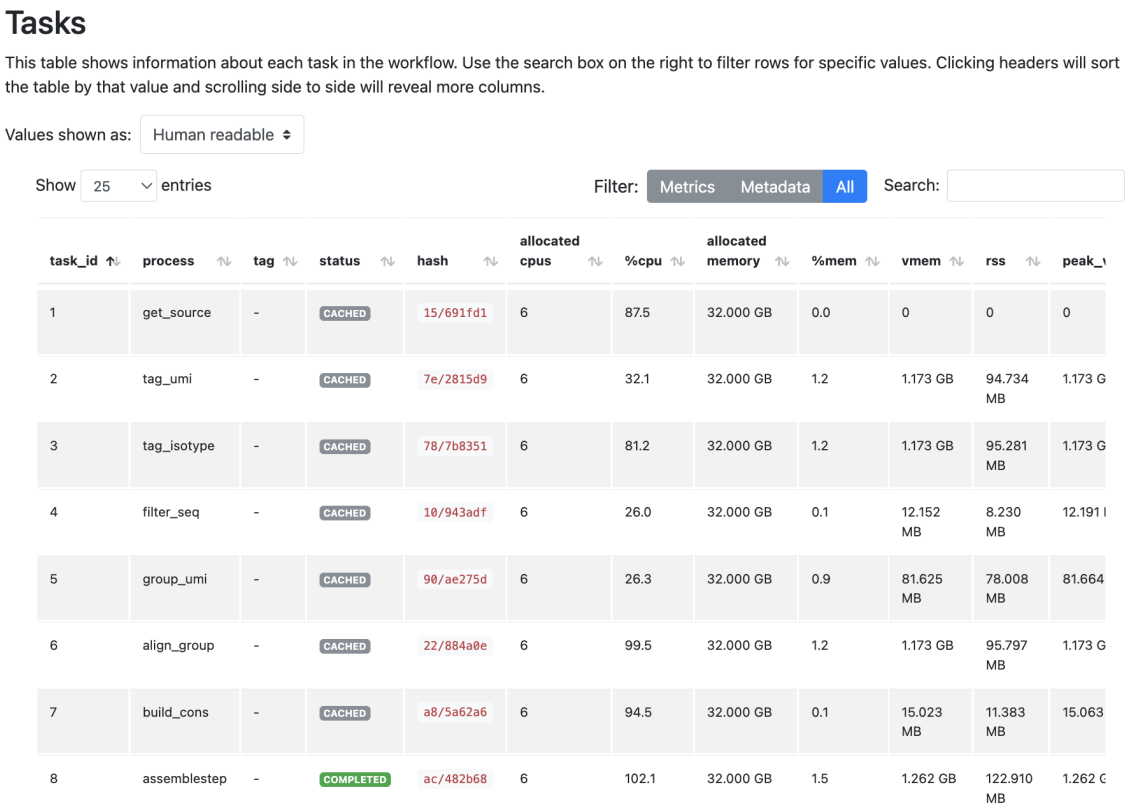


Figure 6.2: Process execution timeline

Finally, a Tasks tab is also displayed in the generated execution reports. This report compiles different information into a selected format, which in Figure 6.3 is displayed in a *Human readable* format.

Simply from analysing these three small reports, some important conclusions were made with regards to the pipeline's efficiency and current bottlenecks. This showcases the value of the reports generated for each execution. By having different reports for each version of a single pipeline,



Tasks

This table shows information about each task in the workflow. Use the search box on the right to filter rows for specific values. Clicking headers will sort the table by that value and scrolling side to side will reveal more columns.

Values shown as: Human readable

Show 25 entries Filter: Metrics Metadata All Search:

task_id	process	tag	status	hash	allocated cpus	%cpu	allocated memory	%mem	vmem	rss	peak_
1	get_source	-	CACHED	15/691fd1	6	87.5	32.000 GB	0.0	0	0	0
2	tag_umi	-	CACHED	7e/2815d9	6	32.1	32.000 GB	1.2	1.173 GB	94.734 MB	1.173 G
3	tag_isotype	-	CACHED	78/7b8351	6	81.2	32.000 GB	1.2	1.173 GB	95.281 MB	1.173 G
4	filter_seq	-	CACHED	10/943adf	6	26.0	32.000 GB	0.1	12.152 MB	8.230 MB	12.191
5	group_umi	-	CACHED	90/ae275d	6	26.3	32.000 GB	0.9	81.625 MB	78.008 MB	81.664
6	align_group	-	CACHED	22/884a0e	6	99.5	32.000 GB	1.2	1.173 GB	95.797 MB	1.173 G
7	build_cons	-	CACHED	a8/5a62a6	6	94.5	32.000 GB	0.1	15.023 MB	11.383 MB	15.063
8	assemblestep	-	COMPLETED	ac/482b68	6	102.1	32.000 GB	1.5	1.262 GB	122.910 MB	1.262 G

Figure 6.3: Execution tasks

researchers can very easily analyse the impact that implemented changes have on performance, as well as generated outputs.

6.2 Analysing similarity levels of different pipeline outputs

Through CompAIRR, the overlap of outputs of the different pipeline versions can be analysed.

Since the *preprocessing_v1* and *preprocessing_v2* versions are very similar, the overlap is expected to be large. Since in *preprocessing_v2* $MAX\_ERROR = 0.6$, the number of sequences in this version is expected to be larger than on *preprocessing_v1*.

Some of the findings from execution of the comparison tools suggested in Section 5.2 are described in Table 6.1.

By analysing the results, it is clear that they are consistent and expected.

Table 6.1: Preprocessing pipeline comparison results

Pipeline Version	**Attributes**	**Sequences**
preprocessing_V1	6	3607
preprocessing_V2	6	3609

Firstly, the number of attributes in both output sets is consistent (6). Since no change was made in the selected output format, it proves that DolphinNext provided consistent executions of different pipeline versions without issues in the output format. Since the output data is expected to follow a specific format (in this case, the AIRR format), this is crucial because it allows for the correct examination and interpretation of output data.

Secondly, the number of sequences in both outputs shows a slight difference (3607 in *preprocessing_V1* and 3909 in *preprocessing_V2*). Once again, this is expected since the MAX_ERROR execution parameter was increased in *preprocessing_V2*, which is expected to increase the number of sequences aligned. Even though the increase is relatively low, this is due to the small data samples used in this specific analysis. It is expected that, with larger samples, this value of the difference in sequences would be much more significant. Nevertheless, this result is expected and once again showcases that it is possible to draw interesting conclusions from executing different versions of a pipeline, providing valuable insights into how the pipeline outputs change when such changes are implemented.

The results presented here serve as a validation for the specific change implemented. However, many other conclusions can be made by executing CompAIRR, specific to the AIRR data being analysed. Even though the tool has some minor limitations (mainly by assuming that the data present in the output files is always clean and correct), it is a potent and efficient tool.

By analysing such results, it is clear that comparability tools can offer a deep understanding of the impact of a given set of changes in a bioinformatics pipeline. Even though the difference in implementation of both pipeline versions was minor, analysing the similarity of outputs between versions validates the implemented changes and helps to develop an understanding of the possible improvements to be made.

Chapter 7

Conclusion and Future Work

After studying the current state of reproducibility frameworks in bioinformatics pipelines, it is clear that, even though there are multiple solutions in the community, there is not a single platform that can flawlessly offer reproducible analysis of pipelines. However, DolphinNext has shown to be the most comprehensive platform, which can guide a user through most stages of a pipeline's lifetime. In Chapter 5, a real use-case of pipeline reproducibility was implemented, in which a pipeline was shared across laboratories, to be replicated across a new environment in the hands of an unacquainted user. In order to consolidate the work implemented, it is important to draw some conclusions from the results and establish a roadmap for future work so that the work can be further developed.

7.1 Conclusion

The present dissertation provides a detailed study of the current state of bioinformatics, specifically the applications of bioinformatics pipelines and their current limitations. A general overview of the context and motivation behind this dissertation was given to support the relevance of the proposed work. After describing the background required to understand the work presented, the state of the art and its current limitations has been thoroughly reviewed. In order to accurately propose a concrete solution, a set of requirements and a methodology were defined, in which a reproducibility model was defined. Lastly, a bioinformatics pipeline was implemented in a local DolphinNext server, where a framework for a DIND configuration was constructed. Different versions of the bioinformatics pipeline were constructed, and the respective outputs were compared through an output similarity tool.

It is essential to highlight the importance of the development of platforms which focus on offering solutions for pipeline reproducibility. Even though this is a relatively recent necessity, triggered by the advancements in NGS techniques, the fact that so many platforms are emerging to tackle this issue is a testament to its importance.

DolphinNext is a highly feature-rich platform which covers all stages of a pipeline's lifetime. However, there are some limitations concerning its functionalities. The fact that the pipeline can not be executed through Docker in the same machine where the server is deployed is a setback which could be solved.

Besides that, an embedded comparability analysis similar to the one described in Chapter 5.2 would be very useful to study the determinism of pipelines or even the effect that changes in a pipeline can have on its output. Comparing outputs is an accurate way of determining how reproducible a pipeline is; therefore, this feature could be added to the platform.

Finally, even though DolphinNext is an open-source platform, it seems that it is somewhat inactive. Given its importance, it would be great if there were more contributions from the bioinformatics community to the project. Nevertheless, it is still under development and is well supported.

In conclusion, the studied work described in this document was instrumental since it provided a reasonably detailed view into the current state of bioinformatics, the specific technologies and tools currently used to perform bioinformatics analysis and, finally, the main limitations and areas of improvement. Moving forward, a great foundation is established for the specific work to be implemented in this dissertation.

7.2 Future work

Even though it is possible to follow a very capable and effective bioinformatics analysis workflow by using DolphinNext only, it is also evident that there are some limitations in its current state.

Firstly, even though it is an excellent feature that a DolphinNext server can be deployed from a Docker image, it can only be genuinely portable once it can handle pipeline execution through Docker. The fact that, currently, it is necessary to make installations and further configurations to the DolphinNext server container is a hurdle for users wishing to execute pipelines in the same machine where the server is deployed. This is an actual use case that currently is not supported, which is problematic since it represents the lowest-effort configuration for researchers to implement. Even though the reproducibility framework suggested in Section 5.1.1 is a solution for this issue, it created some unnecessary overhead which could be dismissed by supporting this feature with DolphinNext.

DolphinNext is very capable in the domain of reporting execution outputs. However, as described in Section 5.2, comparing different outputs of a single pipeline or across different pipelines can also be extremely useful. For this reason, it would be of value to offer the functionality of comparing selected outputs in DolphinNext. Since all the pipeline execution outputs are stored in the platform's database, it would be intuitive to have the ability to select a set of output files and compare them. The comparability of outputs could be analysed through methods similar to those suggested in Section 5.2. This would offer an extensive analysis through a variety of methods and domains, also making use of community-developed tools which are highly efficient and mature, such as CompAIRR. However, it should support not only AIRR-formatted outputs and, therefore, no assumptions regarding the output format should also be made. The results of such similarity

analysis could then be stored accordingly, where they could be displayed to users through various reports. Similar metrics could also be displayed through the graphical modules already in DolphinNext. All these features would bring tremendous value to the platform in that they would provide further insights into the effects that given changes have on a pipeline's outputs, as well as give a better understanding of a pipeline's determinism.

Finally, even though DolphinNext provides logging functionalities, they are somewhat restricted and not very extensive. Integration with logging platforms could be supported, providing filtering and searching functionalities in the present and past logs. Even though this is a particular feature, logging is crucial for troubleshooting and, in general, querying and understanding the operability of a pipeline.

The road towards reproducibility and data preservation in bioinformatics pipelines is never-ending. After advancements in NGS technologies, the research community understood the apparent need to tackle such issues. As a response, some platforms have been developed to improve these characteristics of bioinformatics pipelines. However, they are primarily novel implementations that still have not been thoroughly reviewed and should therefore be further analysed and developed, considering the proposed improvements.

References

[1] Eliseu Binneck, Adriane Wendland, Agda Maria Rodrigues Morales, Álvaro Manoel Rodrigues Almeida, Carlos Alberto Arrabal Arias, Cesar Augusto da Silveira, João Flávio Veloso Silva, José Renato Bouças Farias, Juliana C Molina, Julio César Pedroso, et al. A visual system for DNA sequencing quality control.

[2] Bryan Briney and Dennis R Burton. Massively scalable genetic analysis of antibody repertoires. *BioRxiv*, page 447813, 2018.

[3] Scott Christley, Ademar Aguiar, George Blanck, Felix Breden, Syed Ahmad Chan Bukhari, Christian E. Busse, Jerome Jaglale, Srilakshmy L. Harikrishnan, Uri Laserson, Bjoern Peters, Artur Rocha, Chaim A. Schramm, Sarah Taylor, Jason Anthony Vander Heiden, Bojan Zimonja, Corey T. Watson, Brian Corrie, and Lindsay G. Cowell. The ADC API: A web API for the programmatic query of the AIRR data commons. *Frontiers in Big Data*, 3, 6 2020.

[4] Peter J.A. Cock, Christopher J. Fields, Naohisa Goto, Michael L. Heuer, and Peter M. Rice. The sanger fastq file format for sequences with quality scores, and the solexa/illumina fastq variants. *Nucleic Acids Research*, 38:1767–1771, 12 2009.

[5] Philip A. Ewels, Alexander Peltzer, Sven Fillinger, Johannes Alneberg, Harshil Patel, Andreas Wilm, Maxime Ulysse Garcia, Paolo Di Tommaso, and Sven Nahnsen. Nf-core: Community curated bioinformatics pipelines. *bioRxiv*, 2019.

[6] Bjørn Fjukstad and Lars Ailo Bongo. A review of scalable bioinformatics pipelines. *Data Science and Engineering*, 2:245–251, 9 2017.

[7] Amadou Gaye, Yannick Marcon, Julia Isaeva, Philippe Laflamme, Andrew Turner, Elinor M. Jones, Joel Minion, Andrew W. Boyd, Christopher J. Newby, Marja Liisa Nuotio, Rebecca Wilson, Oliver Butters, Barnaby Murtagh, Ipek Demir, Dany Doiron, Lisette Giepmans, Susan E. Wallace, Isabelle Budin-ljøsne, Carsten Oliver schmidt, Paolo Boffetta, Mathieu Boniol, Maria Bota, Kim W. Carter, Nick Deklerk, Chris Dibben, Richard W. Francis, Tero Hiekkalinna, Kristian Hveem, Kirsti Kvaløy, Sean Millar, Ivan J. Perry, Annette Peters, Catherine M. Phillips, Frank Popham, Gillian Raab, Eva Reischl, Nuala Sheehan, Melanie Waldenberger, Markus Perola, Edwin Van den heuvel, John Macleod, Bartha M. Knoppers, Ronald P. Stolk, Isabel Fortier, Jennifer R. Harris, Bruce H.R. Woffenbuttel, Madeleine J. Murtagh, Vincent Ferretti, and Paul R. Burton. Datashield: Taking the analysis to the data, not the data to the analysis. *International Journal of Epidemiology*, 43:1929–1944, 12 2014.

[8] Youngmahn Han. Bioworks: A workflow system for automation of bioinformatics analysis processes. pages 76–81. IEEE Computer Society, 2011.

[9] Jason A. Vander Heiden, Gur Yaari, Mohamed Uduman, Joel N.H. Stern, Kevin C. O'connor, David A. Hafler, Francois Vigneault, and Steven H. Kleinstein. Presto: A toolkit for processing high-throughput sequencing raw reads of lymphocyte receptor repertoires. *Bioinformatics*, 30:1930–1932, 7 2014.

[10] Jason Anthony Vander Heiden, Susanna Marquez, Nishanth Marthandan, Syed Ahmad Chan Bukhari, Christian E. Busse, Brian Corrie, Uri Hershberg, Steven H. Kleinstein, Frederick A. Matsen IV, Duncan K. Ralph, Aaron M. Rosenfeld, Chaim A. Schramm, Scott Christley, and Uri Laserson. Airr community standardized representations for annotated immune repertoires. *Frontiers in Immunology*, 9, 9 2018.

[11] Shawn Hoon, Kiran Kumar Ratnapu, Jer Ming Chia, Balamurugan Kumarasamy, Xiao Juguang, Michele Clamp, Arne Stabenau, Simon Potter, Laura Clarke, and Elia Stupka. Biopipe: A flexible framework for protocol-based bioinformatics analysis. *Genome Research*, 13:1904–1915, 8 2003.

[12] Saad F Idris, Saif S Ahmad, Michael A Scott, George S Vassiliou, and James Hadfield. The role of high-throughput technologies in clinical cancer genomics. *Expert review of molecular diagnostics*, 13(2):167–181, 2013.

[13] Sehrish Kanwal, Farah Zaib Khan, Andrew Lonie, and Richard O. Sinnott. Investigating reproducibility and tracking provenance - a genomic workflow case study. *BMC Bioinformatics*, 18, 7 2017.

[14] Ian. Korf, Mark. Yandell, and Joseph. Bedell. *Basic Local Alignment Search Tool (BLAST)*.

[15] Neha Kulkarni, Luca Alessandrì, Riccardo Panero, Maddalena Arigoni, Martina Olivero, Giulio Ferrero, Francesca Cordero, Marco Beccuti, and Raffaele A. Calogero. Reproducible bioinformatics project: A community for reproducible bioinformatics analysis pipelines. *BMC Bioinformatics*, 19, 10 2018.

[16] Gregory M Kurtzer, Vanessa Sochat, and Michael W Bauer. Singularity: Scientific containers for mobility of compute. *PloS one*, 12(5):e0177459, 2017.

[17] William D. Lees and Adrian J. Shepherd. *Studying antibody repertoires with next-generation sequencing*, volume 1526, pages 257–270. Humana Press Inc., 2017.

[18] Jeremy Leipzig. A review of bioinformatic pipeline frameworks. *Briefings in bioinformatics*, 18(3):530–536, 2017.

[19] Dale Muzzey, Eric A. Evans, and Caroline Lieber. Understanding the basics of ngs: From mechanism to variant calling. *Current Genetic Medicine Reports*, 3:158–165, 12 2015.

[20] Michal Orzechowski, Bartosz Balis, Krystian Pawlik, Maciej Pawlik, and Maciej Malawski. Transparent deployment of scientific workflows across clouds-kubernetes approach. *Proceedings - 11th IEEE/ACM International Conference on Utility and Cloud Computing Companion, UCC Companion 2018*, pages 9–10, 1 2019.

[21] Milena Pavlović, Lonneke Scheffer, Keshav Motwani, Chakravarthi Kanduri, Radmila Kompova, Nikolay Vazov, Knut Waagan, Fabian L M Bernal, Alexandre Almeida Costa, Brian Corrie, Rahmad Akbar, Ghadi S Al Hajj, Gabriel Balaban, Todd M Brusko, Maria Chernigovskaya, Scott Christley, Lindsay G Cowell, Robert Frank, Ivar Grytten, Sveinung Gundersen, Ingrid Hobaek Haff, Sepp Hochreiter, Eivind Hovig, Ping-Han Hsieh, Günter

Klambauer, Marieke L Kuijjer, Christin Lund-Andersen, Antonio Martini, Thomas Minotto, Johan Pensar, Knut Rand, Enrico Riccardi, Philippe A Robert, Artur Rocha, Andrei Slabodkin, Igor Snapkov, Ludvig M Sollid, Dmytro Titov, Cédric R Weber, Michael Widrich, Gur Yaari, Victor Greiff, and Geir Kjetil Sandve. immuneML: an ecosystem for machine learning analysis of adaptive immune receptor repertoires.

[22] Rute Pereira, Jorge Oliveira, and Mário Sousa. Bioinformatics and computational tools for next-generation sequencing analysis in clinical genetics. *Journal of clinical medicine*, 9(1):132, 2020.

[23] Babak Bashari Rad, Harrison John Bhatti, and Mohammad Ahmadi. An introduction to docker and analysis of its performance. *IJCSNS International Journal of Computer Science and Network Security*, 17, 2017.

[24] Torbjørn Rognes, Lonneke Scheffer, Victor Greiff, and Geir Kjetil Sandve. Compairr: ultrafast comparison of adaptive immune receptor repertoires by exact and approximate sequence matching.

[25] Somak Roy, Christopher Coldren, Arivarasan Karunamurthy, Nefize S. Kip, Eric W. Klee, Stephen E. Lincoln, Annette Leon, Mrudula Pullambhatla, Robyn L. Temple-Smolkin, Karl V. Voelkerding, Chen Wang, and Alexis B. Carter. Standards and guidelines for validating next-generation sequencing bioinformatics pipelines: A joint recommendation of the association for molecular pathology and the college of american pathologists. *Journal of Molecular Diagnostics*, 20:4–27, 1 2018.

[26] Jamie K. Scott and Felix Breden. The adaptive immune receptor repertoire community as a model for fair stewardship of big immunology data, 12 2020.

[27] Erand Smakaj, Lmar Babrak, Mats Ohlin, Mikhail Shugay, Bryan Briney, Deniz Tosoni, Christopher Galli, Vendi Grobelsek, Igor D'Angelo, Branden Olson, Sai Reddy, Victor Greiff, Johannes Trück, Susanna Marquez, William Lees, and Enkelejda Miho. Benchmarking immunoinformatic tools for the analysis of antibody repertoire sequences. *Bioinformatics*, 36:1731–1739, 2020.

[28] Ola Spjuth, Marco Capuccini, Matteo Carone, Anders Larsson, Wesley Schaal, Jon Ander Novella, Oliver Stein, Morgan Ekmefjord, Paolo Di Tommaso, Evan Floden, Cedric Notredame, Pablo Moreno, Payam Emami Khoonsari, Stephanie Herman, Kim Kultima, and Samuel Lampa. Approaches for containerized scientific workflows in cloud environments with applications in life science. 2020.

[29] Morris A. Swertz, Martijn Dijkstra, Tomasz Adamusiak, Joeri K. van der Velde, Alexandros Kanterakis, Erik T. Roos, Joris Lops, Gudmundur A. Thorisson, Danny Arends, George Byelas, Juha Muilu, Anthony J. Brookes, Engbert O. De Brock, Ritsert C. Jansen, and Helen Parkinson. The molgenis toolkit: Rapid prototyping of biosoftware at the push of a button. *BMC Bioinformatics*, 11, 12 2010.

[30] Paolo Di Tommaso, Emilio Palumbo, Maria Chatzou, Pablo Prieto, Michael L. Heuer, and Cedric Notredame. The impact of docker containers on the performance of genomic pipelines. *PeerJ*, 2015, 2015.

[31] Rucha M Wadapurkar and Renu Vyas. Computational analysis of next generation sequencing data and its applications in clinical oncology. *Informatics in Medicine Unlocked*, 11:75–82, 2018.

[32] Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J.G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A.C. t Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan Van Der Lei, Erik Van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. Comment: The FAIR guiding principles for scientific data management and stewardship. *Scientific Data*, 3, 3 2016.

[33] Laura Wratten, Andreas Wilm, and Jonathan Göke. Reproducible, scalable, and shareable analysis pipelines with bioinformatics workflow managers. *Nature Methods*, 18:1161–1168, 10 2021.

[34] Onur Yukselen, Osman Turkyilmaz, Ahmet Rasit Ozturk, Manuel Garber, and Alper Kucukural. DolphinNext: A distributed data processing platform for high throughput genomics. *BMC Genomics*, 21, 4 2020.

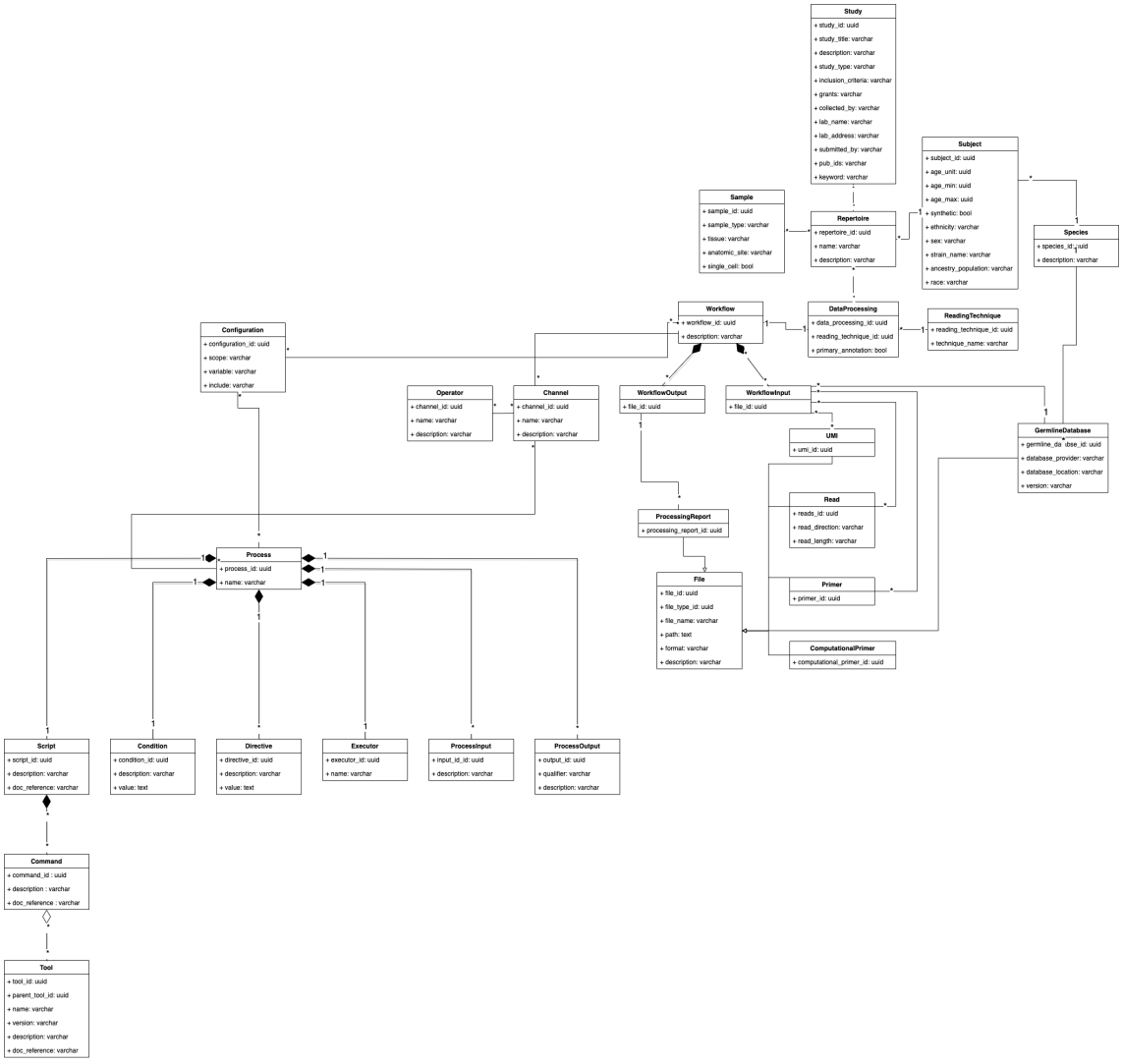
[35] Junjun Zhang, Joachim Baran, A. Cros, Jonathan M. Guberman, Syed Haider, Jack Hsu, Yong Liang, Elena Rivkin, Jianxin Wang, Brett Whitty, Marie Wong-Erasmus, Long Yao, and Arek Kasprzyk. Nextflow enables reproducible computational workflows. *Database*, 2011, 2011.

Appendix A

Reproducibility model

In order to better understand the complete context of most bioinformatics pipelines, a fairly comprehensive model was developed, detailing the main components of pipelines and how they interact. This model is presented in Figure A.

Figure A.1: Reproducibility model



Appendix B

Docker stack for DolphinNext Local Server DIND execution

Here, a simple Docker stack is presented as a complement to the DolphinNext Local Server Docker image.

Listing B.1 contains the source code for the created Docker image, which build on top of the provided DolphinNext Local Server image by installing Docker with the necessary permissions. This way, the deployed container can deploy other child containers, where a pipeline's computations will be executed.

On the other hand, listing B.2 contains the source code for deploying the *compose* stack containing the necessary volume mounts, as well as building and deploying the *Dockerfile* presented in Listing B.1.

```
1  FROM ummsbiocore/dolphinnext-studio:latest
2
3  RUN sudo apt install apt-transport-https ca-certificates curl software-properties-
       common
4  RUN curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
5  RUN sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/
       ubuntu `lsb_release -cs` test"
6  RUN sudo apt update
7  RUN sudo apt install docker-ce -y
8  RUN sudo chmod 666 /var/run/docker.sock
9
10 CMD [ "startup" ]
```

Listing B.1: *Dockerfile*

```
services:
  dolphinnext-server:
    privileged: true
    build: .
    ports:
      - "8080:80"
    volumes:
      - ${DN_PATH}:${DN_PATH}
      - ${DN_PATH}:/export
      - /var/run/docker.sock:/var/run/docker.sock
```

Listing B.2: *docker-compose* file

Appendix C

Dependency versioning tool

Here, the source code for the developed versioning tool is presented in Listing C.1.

This tool is written in *Python3* and imports three packages, *argparse*, *os* and *re*.

Currently, this versioning tool supports *MUSCLE* and *USEARCH*, but can easily be extended to support other common bioinformatics tools.

```
import argparse
import os
import re

RE_PATTERN = '(\d+(?:\.\d+)+)'
INSTALATION_PATHS = {'MUSCLE': '/usr/bin/muscle','USEARCH': '/usr/bin/usearch'}

def printVersion(tool, version):
    print("Tool: %s\tVersion: %s" % (tool, version))

def setupParser():
    parser = argparse.ArgumentParser()

    parser.add_argument('--IGBLAST', action='append_const',
                        dest='tools',
                        const='IGBLAST',
                        default=[],
                        help='Get IGBLAST version')

    parser.add_argument('--MUSCLE', action='append_const',
                        dest='tools',
                        const='MUSCLE',
                        default=[],
                        help='Get MUSCLE version')

    parser.add_argument('--USEARCH', action='append_const',
                        dest='tools',
                        const='USEARCH',
                        default=[],
                        help='Get USEARCH version')

    parser.add_argument('--list', action='version',
                        version='Supported tools:\tigblast, usearch, muscle')

    return parser.parse_args().tools

def getVersions(tools):
    for tool in tools:
        if tool == 'USEARCH':
            stream = os.popen(INSTALATION_PATHS['USEARCH'], '--version')
            output = stream.read()
            version = re.search(RE_PATTERN, output).group(0)
            printVersion(tool, version)
        if tool == 'MUSCLE':
            stream = os.popen(INSTALATION_PATHS['MUSCLE'], '-version')
            output = stream.read()
            version = re.search(RE_PATTERN, output).group(0)
            printVersion(tool, version)

if __name__ == "__main__":
    tools = setupParser()
    getVersions(tools)
```

Listing C.1: Versioning Tool