INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

NGS4Cloud: Cloud-based NGS Data Processing

Alexandre Almeida

João Forja

Project and Seminary

BSc in computer science and computer engineering

Final Report

Supervisors: Cátia Vaz, ISEL
José Simão, ISEL
Alexandre P. Francisco, IST

September 2016

Instituto Superior de Engenharia de Lisboa
BSc in computer science and computer engineering

NGS4Cloud
Cloud-based Data Processing

40640 Alexandre Teixeira de Almeida
41087 João Nuno de Gonçalves Forja

Supervisors: Cátia Vez
José Simão
Alexandre P. Francisco

Final report written for Project and Seminary
from the Bsc in computer science and computer engineering
Summer semester 2015/2016

September de 2016

Abstract

Next-Generation Sequencing (NGS) technologies are greatly increasing the amount of genomic computer data, revolutionizing the biosciences field leading to the development of more complex NGS data analysis workflows (Shuster, 2008). These workflows, also known as pipelines[1], consist of running and refining a series of intertwined computational analysis and visualization tasks on large amounts of data. These pipelines involve the use of multiple software tools and data resources in a staged fashion, with the output of one tool being passed as input to the next one.

Due to the complexity of configuring and parametrizing pipelines, the use of NGS data analysis techniques is not an easy task for a user without IT knowledge. Moreover, knowing input data can be as much as terabytes and petabytes, pipelines execution require, in general, a great amount of computational resources.

We developed NGS4Cloud, a system that offers less IT proficient users a solution to easily produce shareable and reusable pipeline descriptions and execute the pipelines on the cloud.

NGS4Cloud integrates NGSPipes (Dantas and Fleitas, 2015) domain-specific language and tool meta-data repository libraries, introducing minor modifications to these, in order to keep pipelines easy to describe, and adds support for multi-core execution and parallel execution of tools along with data partitioning.

The developed solution in the context of this project integrates an application to schedule and execute pipelines in powerful cloud environments recurring to distributed systems and cluster computing state of art technologies like Docker and Mesos.

[1] In the context of biosciences, a pipeline is not necessarily sequential.

Resumo

As tecnologias *Next-Generation Sequencing* (NGS) estão a criar um aumento na quantidade de dados computacionais genómicos, revolucionando o campo das biociências e levando ao desenvolvimento de *workflows* de análise de dados NGS mais complexas (Shuster, 2008). Estes *workflows*, também conhecidos por *pipelines*[2], consistem em executar uma série de tarefas computacionais de análise e visualização de grandes volumes de dados. Estes *pipelines* envolvem a utilização de múltiplas ferramentas de *software* e dados de forma estruturada, com o *output* de uma ferramenta sendo passado como *input* da próxima.

Devido à complexidade de configurar e parametrizar *pipelines*, a utilização de ténicas de análise de dados NGS não é uma tarefa fácil para utilizadores com pouca experiência informática. Para além disso, sabendo que os ficheiros de dados de *input* podem ter dimensões tão grandes quanto *terabytes* ou até *petabytes*, a execução de *pipelines* requer, geralmente, grandes quantidades de recursos computacionais.

Desenvolvemos o NGS4Cloud, uma solução para utilizadores menos proficientes na área informática facilmente produzirem e partilharem descrições de *pipelines* e executar *pipelines* em ambiente *cloud.*

O NGS4Cloud integra bibliotecas da solução NGSPipes, nomeadamente as da linguagem específica de domínio e do repositório de `meta-dados` das ferramentas do NGSPipes (Dantas and Fleitas, 2015). Embora tenha sido mantida a simplicidade da descrição de *pipelines*, foram introduzidas pequenas modificações com vista a suportar a execução *multi-core* e paralela de ferramentas, juntamente com suporte para partição de dados.

A solução desenvolvida no contexto deste projeto integra uma aplicação que agenda e executa *pipelines* em ambientes de exeução poderosos como a *cloud*, recor-

[2] No contexto das biociências, um *pipeline* não necessariamente sequencial.

rendo a diversas tecnologias do momento na àrea de sistemas distribuídos como o Docker e o Mesos.

Table of contents

List of Figures

List of Tables

List Of Symbols

NGS	Next-Generation Sequencing
DSL	Domain-specific Language
TMR	Tool Meta-data Repository
IR	Intermediate Representation

1 Introduction

Next-Generation Sequencing (NGS) technologies are greatly increasing the amount of genomic computer data, revolutionizing the biosciences field and leading to the development of more complex NGS data analysis workflows (Shuster, 2008). These workflows, also known as pipelines, consist of running and refining a series of intertwined computational analysis and visualization tasks on large amounts of data. These pipelines involve the use of multiple software tools and data resources in a staged fashion, with the output of one tool being passed as input to the next one.

Due to the complexity of configuring and parametrizing pipelines, the use of NGS Data Analysis techniques is not an easy task for a user without IT knowledge. Moreover, knowing input data can be as much as terabytes and petabytes, pipelines execution require, in general, a great amount of computational resources.

NGSPipes framework is devised to solve the first issue, allowing to easily design and use pipelines, without users need to configure, install and manage tools, servers and complex workflow management systems (Dantas and Fleitas, 2015). In the context of NGSPipes, a pipeline is a set of commands of NGS data-processing tools chained by input and output. NGSPipes offers a DSL (domain-specific language), to describe pipelines. The DSL's syntax provides primitives to specify the sequence through which each tool command is executed, to specify arguments, and to chain commands' inputs and outputs. To know each tools' properties and commands, NGSPipes uses tools' meta-data which is saved in a user-provided repository.

However, at the present, NGSPipes operates only in a single machine, which will most likely lack the necessary resources to execute the pipeline in a reasonable period of time, if at all. Even if users do have access to a powerful machine, there are still problems that need to be solved in order to execute a pipeline as efficiently as possible: NGSPipes does not support parallel execution of pipeline tasks, nor splitting input data of tools's commands into multiple fragments to be processed in parallel.

Cloud technologies are sought as a solution to solve the lack of resources of an average computer, a feature that will substantially improve NGSPipes solution, providing users with big clusters of powerful machines to run pipelines more efficiently (Armbrust et al., 2010). The solution we developed, NGS4Cloud, integrates NGSPipes with the capability of running pipelines in a remote cluster. It also extends it with parallel execution of pipeline tasks, in either different cores of the same machine or different machines, as well as data partitioning. On top of that, it also allows the multi-core execution of tools' commands that support it. NGS4Cloud tries to achieve these goals while standing to the usability standards of NGSPipes, which aim to keep pipelines easy to use by non-expert IT users. We presented the NGS4Cloud (Forja et al., 2016) solution at the INForum[1] 2016 edition.

To implement NGS4Cloud, we developed an execution engine that analyses the pipeline and builds a graph of tasks. This graph of tasks reflects the dependency among tasks and allows to infer what can be executed in parallel and what can only be executed in serial. From it, the engine deploys the tasks in a cluster of machines governed by the Mesos's batch job scheduling framework Chronos.

With the automatic inference of dependencies between tasks of the pipeline, we allow the parallel execution of a pipeline while keeping the same standards regarding the simplicity of designing a pipeline.

1.1 Outline

This work is is divided in 8 chapters.

In chapter 2 we will describe the domain of our problem in more depth and our approach to solve it.

In chapter 3 we will analyse a case study, and explain how to use each component of NGS4Cloud.

In chapter 4 we will introduce the technologies that support NGS4Cloud.

In chapter 5 we will overview NGS4Cloud, namely its components and how the goals pointed out in chapter 2 will be reached with this framework.

In chapter 6 we will discuss implementation details about each of the components that compose NGS4Cloud, their functionalities and their interactions.

[1] http://www.inforum.org.pt/, last visited 21/09/2016

In chapter 7 we show and analyse the results of the execution of three different pipelines.

In chapter 8 we will do an overview about what we think should be the future of NGS4Cloud.

2 Problem Description

DNA sequencing is used in multiple fields like molecular biology, evolutionary biology, medicine and forensics. There has been a big increase on the amount of data produced by DNA sequencing since the appearance of Next-Generation Sequencing methods, creating the need to further develop techniques and programs to analyse the data. Due to that, a plurality of tools have appeared to process DNA sequences, or NGS data. These tools are usually executed in structured data workflows and the output of one is the input of others, forming what is usually called in this field a pipeline[1]. Summing up, in the context of NGS data processing, a pipeline is a set of intertwined computational analysis and visualization tasks on large amounts of data.

NGS data analysis techniques raise some challenges since, depending on the tools and the amount of data to be processed, the execution of a pipeline may take days or even weeks; the design and parameterization of a pipeline and setup of tools are not easy tasks for users with little or no IT experience, like many biologists; and the ability to reproduce a pipeline is rather valuable.

We think a good application should address the before mentioned challenges by attending to the following objectives:

1. Make the design and parametrization of a pipeline user friendly and easy for non-IT experts;
2. Facilitate the reproducibility of a pipeline;
3. Run the pipelines efficiently, making use of a cluster of powerful machines and parallelization techniques.

NGSPipes(Dantas and Fleitas, 2015) is a framework to easily design and use pipelines, relying on state of the art cloud technologies to execute them without users need to configure, install and manage tools, servers and complex pipeline

[1] In the context of biosciences, a pipeline is not necessarily sequential.

systems. NGSPipes fulfils the first two objectives, but fails to accomplish the third. NGSPipes does not fully explore the cloud services and remote execution to the maximum, nor the parallelization of a pipeline and its tasks[2], as right now it only allows to execute pipelines on a single machine, running its tasks sequentially, one at a time.

In order to allow a user to abstract from the necessary resources to execute a pipeline and for NGSPipes to be able to validate a given description of a pipeline, there is a component called tool `meta-data` repository. The *tool `meta-data` repository* is a repository which contains information such as: the computational resources needed to execute a tool, the commands a tool contains, the arguments of each command, and the docker image (see section 4.1) in which the tool is installed. Since the setup of a tool can be a hard task, NGSPipes uses docker images to setup and deploy tools.

The tools used to process NGS data are, in essence, batch jobs executors. They are usually developed by software engineers connected to the biology field. There is not a main entity when it comes to developing NGS data processing tools and no sort of standards for the tools are implemented, resulting in a wide variety of tools that operate and behave differently. Due to this variety there isn't much interoperability between them and it becomes difficult to find patterns. Each tool has different commands that receive different arguments and need different amounts of computational resources to execute. Therefore, NGSPipes lays the responsibility of describing each tool used in the pipeline to the users, by requiring them to specify each tool's information and save it in the tool meta-data repository. Notice that users that fill in tool meta-data information are usually tech-savvy, like bioinformaticians, since the descriptions of tools can be used independently by multiple users in different contexts.

To facilitate the design and parametrization of a pipeline, NGSPipes offers a DSL that allows users to produce files containing a pipeline description, that can be executed by NGSPipes engine in any machine that runs it. Together with inputs being easily changed, this makes pipelines reproducible. Notice that the users that produce `.pipes` files using the DSL may not have IT skills, unlike the ones producing the tools' descriptions.

[2] In the context of a pipeline, a task is the execution of a tool's command.

2.1 NGS4Cloud Solution

In order to fulfil all the objectives listed above, including the ones not achieved by NGSPipes solution, we have developed *NGS4Cloud*. To fulfil the first two objectives we are integrating NGSPipes and making use of its DSL and tool meta-data repository. To achieve the third objective we treat remote execution in a cluster of machines and task parallelization as separate concepts.

To explore the remote execution of pipelines in a cluster of machines, we are using Mesos(Ignazio, 2016) (see section 4.2) which includes the following features: allows to easily manage clusters of machines assuring an efficient use of resources; provides fault tolerance and high availability; and has support for Docker(Nickoloff, 2016) (see section 4.1), the same technology used by NGSPipes to setup and deploy tools. Mesos also decouples NGS4Cloud from a specific cloud service. Since pipeline tasks are batch jobs, we are using the Chronos framework (see section 4.3), a batch job scheduler framework that runs on top of Mesos and offers a REST API for scheduling jobs with dependencies. The cluster also includes a distributed file system which hosts the working directory for the execution of the pipeline, allowing all cluster machines to access the files being produced. This solution allows to solve the problem of transferring the necessary files, to execute a tool, between machines. NGS4Cloud system includes a software component, the Monitor (see section 6.4), responsible for reading a JSON file containing a description of the tasks and its dependencies, and using Chronos to schedule jobs corresponding to the tasks.

NGS4Cloud system does not include a cluster and this must be previously set up. However, it does seek to use the cluster resources efficiently and offer an autonomous process for installing tools, uploading inputs and downloading the outputs. NGS4Cloud requires the cluster to be properly configured to run the pipeline (see section 6.3). Concerns like guaranteeing that the cluster has the necessary resources to execute the pipeline or giving priority of task execution to a machine which already has the necessary inputs to execute the task in memory, are not part of our project. Our only preoccupation regarding these aspects is that NGS4Cloud does not need modifications when working against different clusters, as long as they fulfil the specified requirements (see section 6.3).

Regarding the parallelization, we cannot forget that pipelines must remain simple to design and use, therefore we try to create a high abstraction over the parallelization, making this feature as transparent as possible to users, so it does not have a negative impact in usability by non-IT experts. We split parallelization in three levels: multi-core execution of tool commands that support it; parallel execution of independent tasks of a pipeline; and data partitioning in order to process the fragments in parallel executions of the same tool command.

The optimal partitioning of the input files for parallel execution is not solved by NGS4Cloud. This responsibility is entirely delegated to the users when they describe the pipeline using the DSL. If a user decides to split a file, then it will be partitioned as requested. Other ideas of parallelization like streaming data between tools to allow tasks to process different parts of the stream in parallel are not taken into account, since most NGS data processing tools do not support it.

To support the multi-core execution of a command, we modified the tools' meta-data to indicate if a tool command supports multi-core execution by specifying how many cores it can use. Also, to support the parallelization of independent tool commands we automatically infer from the pipeline description dependencies based on the outputs and inputs of commands. Moreover, to support the data partitioning we added a field to the tool meta-data that identifies whether a tool is a unit processing tool or used to split, join, or list files.

Our solution also includes a software component, the analyser (see section 6.1), responsible for interpreting a pipeline description and cross it with the tool meta-data repository, applying the necessary transformations due to data partitioning, and producing tasks which correspond to each command and a DAG (directed acyclic graph) representing the dependencies between tasks. The vertices of the DAG are the tasks, while the edges represent the dependencies. Posteriorly, the DAG is converted into a JSON object which is saved on a file: the same JSON file that the Monitor reads. This file is called intermediate representation (see section 5.2.5) whose benefits are explained on the chapter about the architecture (see chapter 5).

3

Case Study

In this chapter we will present a pipeline, that will serve as case study throughout the report. Its purpose is to introduce the biological context to the reader, while navigating through the NGS4Cloud system and its components. We believe that by the end of the report, the user will be able to use and understand NGS4Cloud different components and structure. This case study is written assuming this is the first time the user is executing NGS4Cloud, therefore there are steps which are only needed to be performed because we assume this is the first execution. All `.pipes` files, tool `meta-data` repository contents, intermediate representation `.json` files, the analyser and monitor `.jar` files and the virtual images used can be found in the NGS4Cloud DvD besides any mentioned URL.

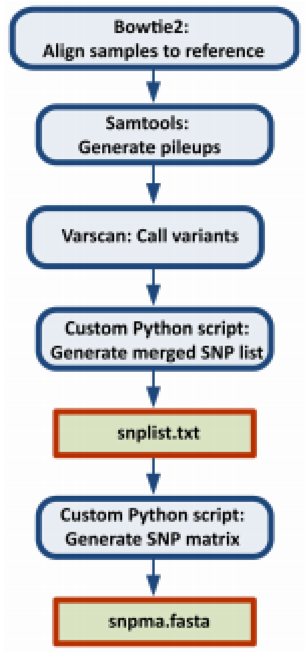


Figure 3.1. Pipeline representation of the case study

A specific use of NGS data in public health is the determination of the relationship between samples potentially associated with a foodborne pathogen outbreak. This relationship can be determined from the phylogenetic analysis of a DNA sequence alignment containing only variable positions, which we refer to as a SNP matrix. The applications of such a matrix include inferring a phylogeny for systematic studies and determining within traceback investigations whether a clinical sample is significantly different from environmental/product samples.

The analysis of next-generation sequence (NGS) data is often a fragmented step-wise process. For example, multiple pieces of software are typically needed to map NGS reads, extract variant sites, and construct a DNA sequence matrix containing only single nucleotide polymorphisms (i.e., a SNP matrix) for a set of individuals.

This case study is a pipeline which combines all the steps necessary to construct a reference-based SNP matrix from an NGS sample data set.The pipeline starts with the mapping of NGS reads to a reference genome using Bowtie2[1], then it continues with the processing of those mapping (BAM) files using SAMtools[2], identification of variant sites using VarScan[3], and ends with the production of a SNP matrix using custom Python scripts[4] (calling of SNPs at each variant site, combining the SNPs into a SNP matrix).

In order to execute the pipeline we will have to do the following actions:

1. Add the description of tools and the URI to the docker image where it is installed, to the tool meta-data repository. This only needs to be done if they are not already in the tool meta-data repository;
2. Make a description of the pipeline using the DSL;
3. Use the analyser application to produce the intermediate representation of the pipeline;
4. Set up the cluster where the pipeline will be executed. This is only done in the first execution, the cluster does not need to be reconfigured according to the pipeline it is executing;
5. Use the monitor application to launch the remote execution of the pipeline by providing it the intermediate representation of the pipeline;
6. Use the monitor to download the pipeline output(s).

The execution of each of these steps will be described throughout the chapter.

[1] http://bowtie-bio.sourceforge.net/bowtie2/index.shtml, last visited 18/09/2016
[2] http://samtools.sourceforge.net/, last visited on 18/09/2016
[3] http://varscan.sourceforge.net/, last visited on 18/09/2016
[4] http://snp-pipeline.readthedocs.io/en/latest/, last visited on 18/09/2016

In section 3.1 we will add the tools used by the pipeline to the tool meta-data repository while exploring its structure.

In section 3.2 we will introduce the DSL, its language primitives and how they can be used.

In section 3.3 we will cover the analyser. The analyser is used to convert the pipeline description into the intermediate representation.

In section 3.4 we will inspect the intermediate representation. We will also explain the mapping from the pipeline description using the DSL to the information contained in the intermediate representation.

In section 3.5 we will expose the requirements a cluster needs to fulfil in order to be able to execute the pipeline. We will also explain how a user can execute NGS4Cloud using a virtual machine for experimentation purposes.

In section 3.6 we will review the monitor architecture and see how it executes the last two actions that will finish our case study. We will also see that the monitor is capable of interacting with the cluster in order to know the current state of execution of the pipeline.

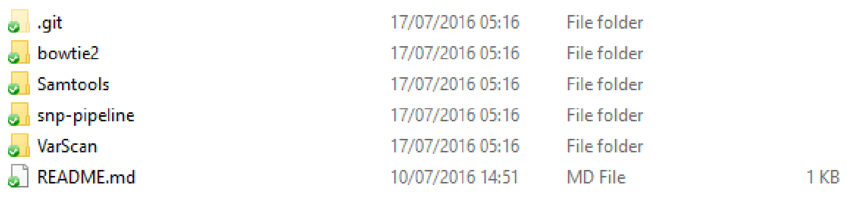
All used resources and files can be found in the DvD provided with the report.

3.1 Tool Meta-data Repository

Information regarding the tool meta-data repository, or TMR, can be found in subsection 4.4.1 and subsection 5.2.1.

Before users start using the analyser to process a pipeline description and produce an intermediate representation, they must first prepare the tool meta-data repository with the information regarding the tools used in the pipeline. As mentioned through the report, this repository is a hierarchical directory system, containing folders with the same name as each tool they contain, and can be stored locally or in a public GitHub repository. Each folder contains the meta-data for the respective tool. For the purpose of the execution of the before mentioned pipeline, we will create a GitHub repository containing meta-data for the following tools: Bowtie2, Samtools, VarScan, and snp-pipeline (See figure 3.1). In the next sections we will also depict a version of the pipeline that also uses the Listing tool (see section 6.1). Remember to make sure the repository is public as the analyser will need to access it.

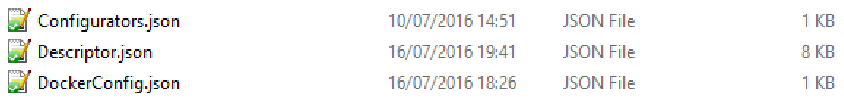
Each folder of a tool should contain at least three files: A `Configurators.json` file, a `Descriptor.json`, and a config file per entry in the configurators file.



.git	17/07/2016 05:16	File folder	
bowtie2	17/07/2016 05:16	File folder	
Samtools	17/07/2016 05:16	File folder	
snp-pipeline	17/07/2016 05:16	File folder	
VarScan	17/07/2016 05:16	File folder	
README.md	10/07/2016 14:51	MD File	1 KB

Figure 3.2. An example of a tool meta-data repository root directory.

Since we are using docker configurators for every tool, every folder will have a `DockerConfig.json` file. Every tool folder should look like figure 3.1.



Configurators.json	10/07/2016 14:51	JSON File	1 KB
Descriptor.json	16/07/2016 19:41	JSON File	8 KB
DockerConfig.json	16/07/2016 18:26	JSON File	1 KB

Figure 3.3. An example of a tool meta-data repository tool directory.

For every tool, we will use a configurators file with a single entry for Docker, and each docker configurator file will be equal except for the docker image URI, which changes from tool to tool. A configurators file should look like this:

```
{
  "configuratorsFileName": [
    "DockerConfig"
  ]
}
```

Figure 3.4. What a configurators file should look like.

and a docker configurator file like this:

```
{
  "name" : "DockerConfig",

  "builder" : "Docker",

  "uri" : "jnforja/samtools",

  "setup" : [
   "sudo apt-get install -y docker.io"
  ]
}
```

Figure 3.5. What a docker configurator file should look like.

Notice that you need docker images with the tools installed. We already provide the necessary docker images to execute the pipeline, and the building and sharing of docker (Nickoloff, 2016) images is not covered in this report. The `uri` property should point to the docker image of the tool.

The tool meta-data repository is intended to be setup by users with IT expertise and knowledge of pipeline design, specifically the tools used. The `Descriptors.json` file must be filled with information regarding the tool corresponding to the folder name. We will go through the steps of creating the descriptor file for the Samtools tool and one of its command. First, we will fill some tool information. The properties `name`, `author`, `description`, `documentation` and `setup`. Notice that only the name is required as the others are optional. Then, we will write the property `toolType` with `unit` as value, as Samtools is a unit Processing tool and we want the analyser to process it like one. The resources properties, `requiredMemory`, `recommendedCpus`, `recommendedDiskSpace` with values corresponding to the Samtools tool. Unlike bioinformaticians, we do not have much experience using Samtools or knowledge about the tool, however, these values do not need to be absolutely precise: Mesos allows for overestimation of the RAM used, although the extra RAM will be idle when the command is being executed; Mesos allows users to specify the minimum number of cpus to execute a task; and Mesos ignores the disk space necessary using as much as it needs. So we will say the required memory is 4GB, the minimum cpus is 1, and recommended disk space is 1MB. We know 4GB RAM is enough for this pipeline. The information about the computational resources to execute a tool should be specified by the tool developer.

We will add an array `commands` where each object represents a command of the respective tool. We will add the command `samtools mpileup`. To do this we add its name ”mpileup”, the command string without arguments ”samtools mpileup”, and a description of the command ”Generate VCF, BCF or pileup for one or multiple BAM files.”. A property `priority` with value 1.

The `arguments` property is an array where each object represents an argument of the command. Those objects each have 4 properties: `name`, `argumentType`, `isRequired`, and `description`. We will exemplify with the arguments of the command ”samtools mpileup” used in this pipeline. The command string we want to use is

```
samtools mpileup --fasta-ref <reference.fasta> <input.bam file> --output <output pileup file>".
```

We will fill strictly the necessary arguments to execute the command as described above. We go through the samtools documentation[5] to the mpileup command and find the arguments `"--fasta-ref"`, `"--output"`, and the `in.bam`. For each argu-

[5] http://www.htslib.org/doc/samtools.html, last visited on 17/07/2016

ment we add an object to the arguments array with the before mentioned properties. Notice how the first two arguments are a name-value pair, unlike the third which is only a value. The name of the name-value pair arguments needs to be the same as it appears in the command string. In this case, `"--fasta-ref"` and `"--output"`. The argument corresponding to the input can have any name. We will use the name input. We will fill the descriptions accordingly to the documentation, set the `isRequired` property false for every argument except the input, and the argument type to file for the three arguments.

Next we will need to fill the inputs and outputs arguments. In the case of the ”mpileup” command, all arguments correspond to either input or output. The argument ”–output” corresponds to an output, so we create an output object to insert in the outputs array. That object contains 5 properties. One for the name of the output `name`, one for the name of the matching argument `argumentName` in case there is one, which there is: ”–output”, one for the output type, one for the default value, and one for the description. The same goes for input entries, except that instead of the property `outputType`, it is `inputType`. All inputs and outputs types are `file_dependent` as they either consume or generate a file.

The inputs and outputs properties are important for the analyser to be able to generate the dependencies between tasks appropriately by combining inputs with outputs.

After filling in the tool and command information, the descriptor should look like this:

```
{
  "name": "Samtools",
  "author": "Heng Li from the Sanger Institute wrote the original C version of samtools. Bob
      Handsaker from the Broad Institute implemented the BGZF library. James Bonfield from the
      Sanger Institute developed the CRAM implementation. John Marshall and Petr Danecek
      contribute to the source code and various people from the 1000 Genomes Project have
      contributed to the SAM format specification.",
  "version": "1.3.1",
  "description": "Samtools is a set of utilities that manipulate alignments in the BAM format. It
      imports from and exports to the SAM (Sequence Alignment/Map) format, does sorting, merging
      and indexing, and allows to retrieve reads in any regions swiftly. Samtools is designed to
      work on a stream. It regards an input file ‘-’ as the standard input (stdin) and an output
      file ‘-’ as the standard output (stdout). Several commands can thus be combined with Unix
      pipes. Samtools always output warning and error messages to the standard error output
      (stderr). Samtools is also able to open a BAM (not SAM) file on a remote FTP or HTTP server
      if the BAM file name starts with ‘ftp://’ or ‘http://’. Samtools checks the current working
      directory for the index file and will download the index upon absence. Samtools does not
      retrieve the entire alignment file unless it is asked to do so.",
  "documentation": [
    "http://www.htslib.org/doc/samtools.html"
  ],
  "setup": [
  ],
  "toolType": "unit",
  "requiredMemory": 4096,
  "recommendedCpus": 1,
  "recommendedDiskSpace": 1,
  "commands": [
    {
      "name": "mpileup",
      "command": "samtools mpileup",
      "description": "Generate VCF, BCF or pileup for one or multiple BAM files. Alignment records
           are grouped by sample (SM) identifiers in @RG header lines. If sample identifiers are
           absent, each input file is regarded as one sample.",
      "priority": 1,
      "argumentsComposer": "Samtools",
      "arguments": [
        {
          "name": "input",
          "argumentType": "string",
          "isRequired": "true",
          "description": "BAM input files."
        },
        {
          "name": "--fasta-ref",
          "argumentType": "file",
          "isRequired": "false",
          "description": "The faidx-indexed reference file in the FASTA format. The file can be
               optionally compressed by bgzip. [null]"
        },
        {
          "name": "--output",
          "argumentType": "file",
          "isRequired": "false",
          "description": "Write pileup or VCF/BCF output to FILE, rather than the default of
               standard output.(The same short option is used for both --open-prob and --output. If
               -o’s argument contains any non-digit characters other than a leading + or - sign, it
               is interpreted as --output. Usually the filename extension will take care of this,
               but to write to an entirely numeric filename use -o ./123 or --output 123.)"
        }
      ],
      "outputs": [
        {
          "name": "--output",
          "description": "Write pileup or VCF/BCF output to FILE, rather than the default of
               standard output. (The same short option is used for both --open-prob and --output. If
               -o’s argument contains any non-digit characters other than a leading + or - sign, it
               is interpreted as --output. Usually the filename extension will take care of this,
               but to write to an entirely numeric filename use -o ./123 or --output 123.)",
          "outputType": "file_dependent",
          "value": "",
          "argument_name": "--output"
        }
      ],
      "inputs": [
        {
          "name": "input",
          "description": "BAM input files",
          "inputType": "file_dependent",
          "value": "",
          "argument_name": "input"
        },
        {
          "name": "--fasta-ref",
          "description": "The faidx-indexed reference file in the FASTA format. The file can be
               optionally compressed by bgzip. [null]",
          "inputType": "file_dependent",
          "argument_name": "--fasta-ref",
          "value": ""
        }
      ]
    }
  ]
}
```

A GitHub repository containing the folders for the 4 tools already appropriately filled with the purpose of executing this pipeline can be found in `https://github.com/Vacalexis/ngs4cloudcasestudytools`. Besides the 4 unit processing tools, you can also find there the three data partitioning tools: split, join and listing. These are necessary if users intend to use data partitioning features of the NGS4Cloud framework, which we will be using.

3.2 Domain-specific Language

One of the objectives of NGS4Cloud is the ability to easily share pipelines. These can be shared using `.pipes` files which are then processed by the analyser into intermediate representations. The .pipes file contains the description of a pipeline, its tools, its commands, and its arguments. In the following, we will show how to add a tool and a command and later on redirect you to the NGSPipes DSL github wiki where there is a more detailed guide on the DSL and also provide the link for the full description of the case study pipeline. The inputs used can be found in `https://github.com/CFSAN-Biostatistics/snp-pipeline`.

First of all we start a .pipes file by declaring a pipeline using the `Pipeline` primitive and stating the type of repository and the URL of the repository we will use. We will use the repository linked above. It should look something like figure 3.6.

Then we step through the description of a tool command. To do that we first use the `tool` primitive to describe the tool and the type of configurator we will use. We will pick the tool ”samtools” and the docker configurator. The description file should look like figure 3.7.

```
Pipeline "GitHub" "https://github.com/Vacalexis/ngs4cloudcasestudytools" {
}
```

Figure 3.6. Example of a Pipeline primitive usage.

```
Pipeline "GitHub" "https://github.com/Vacalexis/ngs4cloudcasestudytools" {
  tool "Samtools" "DockerConfig" {
  }
}
```

Figure 3.7. Example of a tool primitive usage.

We will look into the steps of adding two commands chain between each other using the `command`, `argument`, and `chain` primitives.

First we add a command stating its name and encapsulate its arguments in brackets. Each argument must contain its name and value. We chain arguments from a command with an argument of other command using the `chain` primitive. We will add the commands "samtools sort" and "samtools mpileup" and chain the output "-o" of the "sort" command with the input "input" of the "mpileup" command. It should look like figure 3.8.

```
Pipeline "GitHub" "https://github.com/Vacalexis/ngs4cloudcasestudytools" {
  tool "Samtools" "DockerConfig" {
   command "sort" {
     argument "-o" "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads1.sorted.bam"
     argument "input" "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads1.unsorted.bam"
   }

    command "mpileup" {
     argument "--fasta-ref"
          "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reference/lambda_virus.fasta"
     chain "input" "-o"
     argument "--output" "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads1.pileup"
   }
  }
}
```

Figure 3.8. command, argument and chain primitives usage.

More on the Domain-Specific Language, or DSL, in subsection 4.4.2 and subsection 5.2.2. You can also find info in the NGSPipes DSL GitHub repository wiki [6]. You can find the full `.pipes` file in annex D.

3.3 Analyser

The analyser only has one functionality which is to produce a `.json` IR file from the `.pipes` description file.

[6] https://github.com/ngspipes/dsl/wiki, last visited on 18/07/2016

In figure 3.9 is depicted how to execute the analyser console application with the command:

```
java -jar analyser.jar analyser -pipes pipeline.pipes -ir ir.json -input
    https://github.com/CFSAN-Biostatistics/snp-pipeline/archive/master.zip
```

Figure 3.9. Case study analyser command.

This command uses the pipeline description file passed after `-pipes`, stores it in the file passed after `-ir`, and uses the inputs in the URL passed in `-input`. The `analyse` command supports an option to state which outputs the user wants to download, by passing the option `-outputs output1 [ | output2 [ | output3 [ | ... ]`. When not specified, the analyser will specify the download of every possible pipeline execution output.

The IR produced by the analyser, using the tool meta-data repository and the .pipes file built for the case study, can be found in annex E.

The analyser is thoroughly described in section 6.1.

3.4 Intermediate Representation

The Intermediate Representation (IR) is an important abstraction in our solution. Although the analyser succeeds most times in inferring the dependencies between tasks and partitioning the data adequately, sometimes a wrongly described tool meta-data or wrongly programmed argument composer may induce the production of a bad intermediate representation. We are not responsible for these user specific components, so we suggest users to take a glance at the IR once this is produced to confirm everything is correct, in order to avoid errors in the execution of the pipeline.

By looking at the full `.pipes` file (appendix D), we can see that we are using a listing tool in order to apply a set of four commands (SAMtools and VarScan) to four different files. By looking at the IR file, we can see that indeed, four tasks were generated per SAMtools command and VarScan command. We can also see the `chain` primitive was replaced with the outputs of the chained tool correctly. Everything worked out good. Although we have confidence in the analyser to skip inspecting the intermediate representation looking for errors before passing it to the monitor, it's still a good practice to do it if we're going to execute a pipeline that takes a long time in order to reduce the risk of it failing midways due to a poorly written pipeline description or tool `meta-data` repository with incorrect information, or even a mistake of the analyser.

More importantly, the Intermediate Representation contains each exact command that is going to be executed, its dependencies and the computational resources that will be requested to the executing environment.

To know more about the Intermediate Representation, or IR, read section 6.2.

3.5 Cluster

As already said, to execute NGS4Cloud it is necessary a cluster which makes available a Chronos endpoint. However, for experimental purposes, users can simulate a cluster using a virtual machine. We have created a virtual machine image that can be used to execute the pipeline of this case study, as it doesn't require a big amount of computational resources.

The virtual image is in the NGS4Cloud DvD and is compressed into a `.zip` file with the name NGS4CloudExecutionEnvironment`.zip`. To emulate the virtual image it is necessary 4GB of RAM and 1 CPU. We should extract the content, and emulate the image using a virtualization software which supports vmdk[7] files, like VMware[8] or VirtualBox[9].

Having launched the image we wait for the graphical environment and login with the following credentials:

User: ngs4cloud

Pass: cloud123

Then open a terminal window and run `/sbin/ifconfig`. From that command we should get the ip where the virtual machine is operating.

Then we can open a browser on the host operating system and connect to the launched virtual machine at port 4400. If everything was done correctly, we should now see the homepage of Chronos.

3.6 Monitor

Now that we have a Chronos service running it's time to execute the Monitor component.

[7] https://en.wikipedia.org/wiki/VMDK, last visited on 14/09/2016
[8] http://www.vmware.com/products/player/playerpro-evaluation.html, last visited on 14/09/2016
[9] https://www.virtualbox.org/, last visited on 21/09/2016

We will start by creating a directory called Monitor and moving the file monitor.`jar`, which is in the NGS4Cloud DvD, to there.

After downloading the `.jar`, create a file called configs and open it. As depicted in figure 3.10 it is necessary to specify some configurations.

```
SSH\_HOST = "ip of the virtual machine"
SSH\_PORT = 22
SSH\_USER = ngs4cloud
SSH\_PASS = cloud123
CHRONOS\_HOST = "ip of the virtual machine"
CHRONOS\_PORT = 4400
PIPELINE\_OWNER = example@example.com
CLUSTER\_SHARED\_DIR\_PATH = /home/ngs4cloud/pipes
WGET\_DOCKER\_IMAGE = jnforja/wget
P7ZIP\_DOCKER\_IMAGE = jnforja/7zip
```

Figure 3.10. Example of configs file.

Now in the monitor directory we will create another directory called repo and add the following entry to the configs file:

EXECUTION_TRACKER_REPO_PATH = ”The path of the repo directory”

We will save the configs file and close it.

To finish configuring the monitor we just add a environment variable named NGS4_CLOUD_MONITOR_CONFIGS with the path to the configs file as value.

We will open a terminal and execute monitor.jar. A message explaining all the commands the monitor can execute should appear. Move the ir.json file produced in the previous steps of this case study to the Monitor directory and execute the command depicted in figure 3.11.

```
java -jar monitor.jar launch ir.json
```

Figure 3.11. Case study monitor launch pipeline command.

We see in the console messages regarding the upload of the input file. Once the upload is finished a message like this will appear ”ID: 1”, this is the ID attributed to the launched pipeline.

Now to check the state of the pipeline execution we will use the command from figure 3.12.

```
java -jar monitor.jar status 1
```

Figure 3.12. Case study monitor status command.

Notice that ”1” is the id of pipeline, which was given in the previous step. If the pipeline has finished executing this message will appear ”The pipeline execution has finished with success.”

When the pipeline execution has finished we will type the command pictured in figure 3.13, to download its outputs.

```
java -jar monitor.jar outputs 1 .
```

Figure 3.13. Case study monitor outputs command.

This will download the outputs of the pipeline to the directory where the monitor is being executed. The downloaded outputs are the ones specified in the intermediate representation in annex E.

4

Related Technologies

In this chapter we introduce the technologies that support NGS4Cloud.

4.1 Docker

Docker (Nickoloff, 2016) is an open-source project which wraps and extends Linux containers technology to create a complete solution for the creation and distribution of containers. The Docker platform provides a vast number of commands to conveniently manipulate containers.

A *container* is an isolated, yet interactive, environment configured with all the dependencies necessary to execute an application. The use of containers brings advantages such as:

- Having little to no overhead compared to running an application natively, as it interacts directly with the host OS kernel and no layer exists between the application running and the OS;
- Providing high portability since the application runs in the environment provided by the container; bugs related to runtime environment configurations will almost certainly not occur;
- Running dozens of containers at the same time, thanks to their lightweight nature;
- Executing an application by downloading the container and running it, avoiding going through possible complex installations and setup.

To easily configure the virtual environment that the container hosts, Docker provides Docker images. *Images* are snapshots of all the necessary tools and files to execute an application. Containers can be started from images, the same way virtual machines run snapshots.

To effortlessly distribute images, Docker provides *registries*. These are public or private stores where users may upload or download images. Docker provides a cloud-based registry service called DockerHub (Nickoloff, 2016).

We use Docker as an alternative for software containerization because it is the most well known and actively developed and supported in the area. Many frameworks already support it or are starting to support it.

4.2 Mesos

Apache Mesos (Ignazio, 2016) is a distributed, high availability, fault tolerant system that allows for multitenancy with the support of containers technologies like Linux containers and Docker. Mesos architecture has three main entities: *masters*, *slaves*, and *frameworks*.

A Mesos cluster can have one or more masters, responsible for managing the computational resources provided by the slaves. Using Zookeeper (Ignazio, 2016), Mesos implements a leader election technique to ensure fault tolerance: if the leading master fails, other master will be ready to replace it.

Slaves are the cluster machines where tasks will be executed. After launching a slave, it uses Zookeeper to register with the current leading master and advertises its available computational resources, such as CPU, memory and disk. A slave can be configured with the list of resources it shall advertise. If omitted, it will automatically offer all the available resources.

A framework is a Mesos application responsible for scheduling and executing user provided tasks in the cluster. It can be split into two components: the *scheduler* and *executors*. Using Zookeeper, the scheduler detects the leading master and proceeds to registering with it. The scheduler then starts receiving computational resource offers from the leading master. If it has pending tasks to be executed, prompted by users, the scheduler chooses a suitable resource offer and launches an executor on the respective slave to run the tasks. Executors are process containers launched in slaves and are responsible for running user submitted tasks. The built-in Mesos executors offer the possibility to launch tasks from shell scripts or docker containers.

We chose Mesos since it is widely used and stable, it supports Docker, and it is open-source. It is used by big companies like Twitter and Ebay. Mesos also allows to develop custom frameworks to work on top of it.

4.3 Chronos

Chronos[1] is a Mesos framework specialized in scheduling batch jobs. Similarly to Mesos, it uses Zookeeper to ensure fault tolerance. It is possible to schedule command line jobs and docker jobs through a user interface or through a REST API. Jobs can be scheduled to run on a specified date and in repeating intervals, with the option to specify the frequency at which the job is meant to rerun. Chronos also allows job dependency scheduling.

We use Chronos because we need a batch-job scheduler that runs on top of Mesos to schedule the pipeline tasks and supports Docker. To our knowledge, there was no other batch-job scheduler that met our requisites.

4.4 NGSPipes

In these subsections we explore the components of *NGSPipes* that are used in the NGS4Cloud solution. In subsections 5.2.1 and 5.2.2 we explain how we modified or extended them to meet the new requirements mentioned in chapter 2.

4.4.1 Tool Meta-data and Repository

Bearing in mind the diversified nature of the NGS data processing tools and that fact that biologists need to use specific tools for different tasks, NGSPipes manages to treat tools like plug-ins. The description of these plug-ins is known as *tool meta-data.* This is stored in a hierarchical directory system organised repository, known as *tool meta-data repository.* This repository may be local or remote. Using a remote repository allows us to access the data on the repository independently of the machine we are using, and also gives us a backup of the data. The repository has a file `Tools.json` with an array of the tools' names contained in the repository, and a folder *per* tool, named after the stored tool. Each folder contains the tool meta-data, which has to supply the following information:

- A descriptor object, kept in a JSON file named `Descriptor.json` which contains information about the tool, its commands, and its commands' arguments.
- A JSON file named `Configurators.json` which contains an array of configurators for the tool. A configurator is an object that holds the basic information to setup the tool.

[1] https://mesos.github.io/chronos/, last visited on 2016/04/26

- One file per configurator in the `Configurators` array. The file name corresponds to the configurator name plus the extension `.json`. In the case of Docker, the configurator contains the name of the docker image, and an array with the setup commands. The docker image name maps to an image stored in DockerHub registry, being easily distributed by downloading from any machine that needs to run the image.

Schemas for the `.json` files mentioned above can be found in the NGSPipes tool meta-data wiki[2].

Information from new tools can be added to the tool meta-data repository, and new tool images can de added to DockerHub. By recurring to tool meta-data, biologists can use almost any tool as long its specification is added to the tool meta-data repository they use.

4.4.2 Domain-specific Language

In this subsection we look into the *domain-specific language* used to describe pipelines. A `.pipes` file is a file whose content is a description of a pipeline. The description is done using the DSL. The DSL was created to offer users a way to describe and parametrize pipelines. It also promotes reproducibility since parameters are easily changed and descriptions can be stored and shared with a single file. The tool meta-data and the DSL, when combined together, provide programmers with a mechanism to validate the descriptions of the pipelines by matching the description parameter values with the tool meta-data information, regarding tool names, command names, argument names, and argument types.

The DSL has 5 primitives: *Pipeline*, *tool*, *command*, *argument* and *chain*.

Pipeline

Since a pipeline is composed by the execution of one or more tools, it must be defined the tools' repository, i.e., all the information necessary with respect to the available tools. To define this repository in the pipeline it is necessary to identify not only where it is stored, but also the type of storage (locally or remotely, like GitHub) to know how to process that information. Formally, the pipeline must follow the following grammar:

```
Pipeline: 'Pipeline' repositoryType repositoryLocation '{' (tool)+ '}';
```

[2] https://github.com/ngspipes/tools/wiki, last visited on 2016/06/11

where `(tool)+` represents that a pipeline is composed by the execution of one or more tools (notice that, as will be further explained, the tool execution may include the execution of one or more commands).

tool

Each tool is specified in the pipeline by its name, its configuration file name (without extension), and by the set of commands within the tool that will be executed within this pipeline. The tool primitive follows the grammar:

```
tool: 'tool' toolName configuratorFileName '{' (command)+ '}'
```

where `(command+)` represents the commands of a tool to be executed. One or more commands can be specified here and may even be specified repeated commands with different arguments. A tool and its commands can be specified more than once in a pipeline. A file with the name `configuratorFileName` must exist in the tool repository, passed in the *Pipeline* primitive (parameter `repositoryLocation`). Within the tool information, a configuration file named `configuratorFileName`, with JSON Format, must also exist. This file must define a JSON object with the `build` property set as `configuratorFileName`. With this information together with the repository information, the environment for executing the tool's commands is specified.

command

As mentioned before, there may exist a set of commands within the tool that should be executed within a pipeline. For executing each command, it is necessary to identify its name, which is unique in the tool context and to set the values for each required parameters (optional parameters may not be specified). We refer to the command parameters in NGS4Cloud language as arguments, since we only specify in the pipeline the parameters for which we have values to set. The command specification must follow the grammar:

```
command: 'command' commandName '{' (argument | chain)+ '}'
```

where `(argument | chain)+` represents that there may exist a list of arguments within this command as well as a list of chains. As mentioned before, a command can be executed more than once in a pipeline, and in that case it must be described the amount of times it is to be executed, always passing the desired arguments and chains.

argument

The argument definition has the following syntax:

```
argument: 'argument' argumentName argumentValue
```

chain

The *chain* primitive allows to set an argument of a command with the produced output of other command. It links outputs to inputs. Sometimes the produced output is returned as files with names given internally by the command (identified in each tool-meta-data through property `outputs`). Alternatively the output files name may be given explicitly as an argument to the command. In both situations, it is common that other commands use these output files to continue processing the pipeline. The primitive *chain* has a simplified version, which can be used when the output is from the previous stated command in the pipeline specification. In this case, we only specify the name of the output file to chain with the given argument. Another version of the primitive allows the user to chain between different tools commands that are now followed. Here the user must state the tool name, the command name, the output and argument names. A last version of the primitive *chain* is when the name of the tool can be omitted, but it is necessary to specify the name of the command, of the argument and also the output. This applies to cases where the chain occurs between two commands of the same tool. Thus, the chain specification must follow the grammar:

```
chain: 'chain' argumentName ( ( toolName )? commandName)? outputName
```

Examples of pipeline descriptions can be found in the NGSPipes DSL wiki[3].

[3] https://github.com/ngspipes/dsl/wiki, last visited on 11/06/2016

5

Architecture

In this chapter we describe each component of the system, how they interact, and which technologies support them.

5.1 Overview

Figure 5.1 is a diagram that shows the system's main components and their interactions. There are five components:

1. Tool meta-data repository;
2. Domain-specific language;
3. Analyser;
4. Intermediate representation;
5. Monitor.

The interaction begins when a user provides a file with a pipeline's description, specified using the NGS4Cloud DSL, to the system. The analyser inspects the given file and, using the information stored in the tool meta-data repository, produces the instructions to execute the pipeline, along with the required computational resources for its execution. These instructions are then written to a file and locally stored, giving origin to an intermediate representation. Having the analyser concluded its job, the monitor is launched. It converts the intermediate representation into jobs' descriptions readable by Chronos and, using Chronos's REST API, schedules them for execution. Having launched the pipeline, the user can now query the system to know its current state. This results in a series of requests from the Monitor to Chronos. When the pipeline finishes the execution, the user can make a request to the monitor to download its outputs.

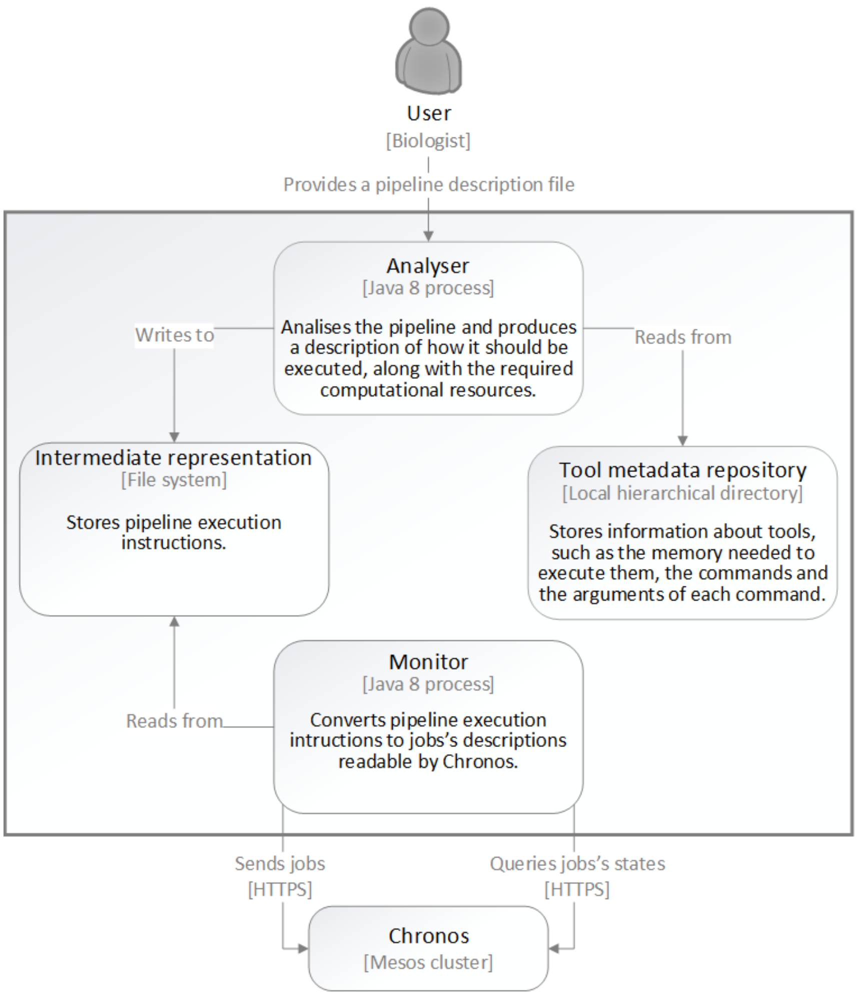

User
[Biologist]
Provides a pipeline description file
Analyser
[Java 8 process]
Analises the pipeline and produces a description of how it should be executed, along with the required computational resources.
Writes to
Reads from
Intermediate representation
[File system]
Stores pipeline execution instructions.
Tool metadata repository
[Local hierarchical directory]
Stores information about tools, such as the memory needed to execute them, the commands and the arguments of each command.
Monitor
[Java 8 process]
Converts pipeline execution intructions to jobs's descriptions readable by Chronos.
Reads from
Sends jobs
[HTTPS]
Queries jobs's states
[HTTPS]
Chronos
[Mesos cluster]

Figure 5.1. Architecture container diagram

5.2 Components

In the following subsections we take a brief look at the components of the NGS4Cloud solution.

5.2.1 Tool Meta-data Repository

In NGS4Cloud we adopted NGSPipes solution and modified it to suit the new needs. We now require users to specify in the tool meta-data information regarding the tool type, the recommended computational resources and input arguments of each command. The tool type is necessary as NGS4Cloud has four different tool types which are data processing tools, listing tools, splitting tools, and joining tools, while NGSPipes had only the first. The new tool types were introduced to support data partitioning. In section 6.1 tool types will be more thoroughly explained. The computational resources information was added as this information is important to help running each command in a machine that supports its computational resources needs. This includes multi-core execution of tool's commands that support it. Input arguments were added to help the analyser inferring dependencies between pipeline tasks. This too will be better explored in section 6.1.

As of now, NGS4Cloud has only support for repositories stored locally or in GitHub. If users want to use other storages, they will have to extend the code to support it.

The schemas for the updated JSON files to match NGS4Cloud modifications to the tool meta-data mentioned above can be found in appendix C.

5.2.2 Domain-specific Language

The DSL library is a component reused from NGSPipes and no syntax modifications were done to it or primitives added.

In section 6.1, we will explain how, although we did not modify the syntax of the DSL from NGSPipes to NGS4Cloud, we did change how it is processed. We will explain how it is processed and transformed into an intermediate representation and the motivation for that.

Since there were no syntax modifications, applications made in the context of NGSPipes that work with the DSL, like editors, can also be used in the context of NGS4Cloud.

5.2.3 Analyser

It's the analyser who is tasked with mapping the pipeline description into an intermediate representation, which has all the information needed for the remote execution of the pipeline.

The analyser receives a pipeline description and validates it against the information stored in the tool meta-data repository. Then it associates to each tool of the pipeline the resources required for its execution with information which is also stored in the tool meta-data repository. The analyser examines the description and translates it in memory java objects applying the necessary transformations corresponding to the use of data partitioning tools. After the analysis of the pipeline's description it then proceeds to create a directed acyclic graph of tasks based on the dependencies between tools' commands. The representation of these tasks is stored in a file. The analyser is developed in Java8 as it reuses components from NGSPipes's parser which is written in Java.

5.2.4 Monitor

The Monitor parses a pipeline execution instructions file and produces the corresponding jobs description that Chronos understands. The monitor then schedules jobs using the respective descriptions. Besides scheduling jobs, the monitor also queries Chronos to check their progression state and allows the user to download the pipeline's final outputs. We implemented the Monitor in Java8.

Mesos and Chronos

Aside from characteristics like fault tolerance and high availability, Mesos provides support for Docker containers and is currently the target of plenty of development efforts. Having support for Docker is an essential feature for our system since the unit processing tools setup is achieved through Docker images: Docker enables tools' automatic setup. By accompanying Mesos development, NGS4Cloud may benefit from future Mesos updates. From a set of Mesos frameworks, Chronos is the only one we found focused on batch job scheduling which offers Docker support, job dependency scheduling and an API.

5.2.5 Intermediate Representation

Unlike Chronos, there are batch job scheduling frameworks that don't offer an API, but instead offer a library to schedule jobs. These libraries are not necessarily written in Java, the analyser's development language. In order to decouple the Monitor from the Java programming language and the NGS4Cloud solution from Chronos, thus allowing for a wider range of scheduling frameworks to be explored, we established an intermediate representation in JSON format that is stored in a text file.

5.3 Data Model

The main entity of NGS4Cloud is the pipeline. A pipeline, in our domain problem, can be seen as a directed acyclic graph (DAG) where the vertexes are the tools' commands and the edges the input dependencies between them. To help representing a pipeline in NGS4Cloud we created the abstraction task. Task represents a tool command, a list of tasks represents a pipeline. A task contains the following properties:

- id

- dockerImage
- command
- mem
- disk
- cpus
- parents

The `id` property is an integer which identifies a task in the current pipeline. The `dockerImage` property is the name of the docker image stored in DockerHub which will be used to set up a tool, which will be executed on the cluster. The `command` property is the command line instruction which will cause the execution of the tool. The `mem` property is the necessary RAM, in MB, to execute the command. The `disk` property is the necessary disk space, in MB, to execute the command. The `cpus` property is the number of necessary cores to execute the command, and it can be fractional. The `parents` property is the representation of the dependencies between tasks; it is a list which contains the id of all the tasks that a task depends upon.

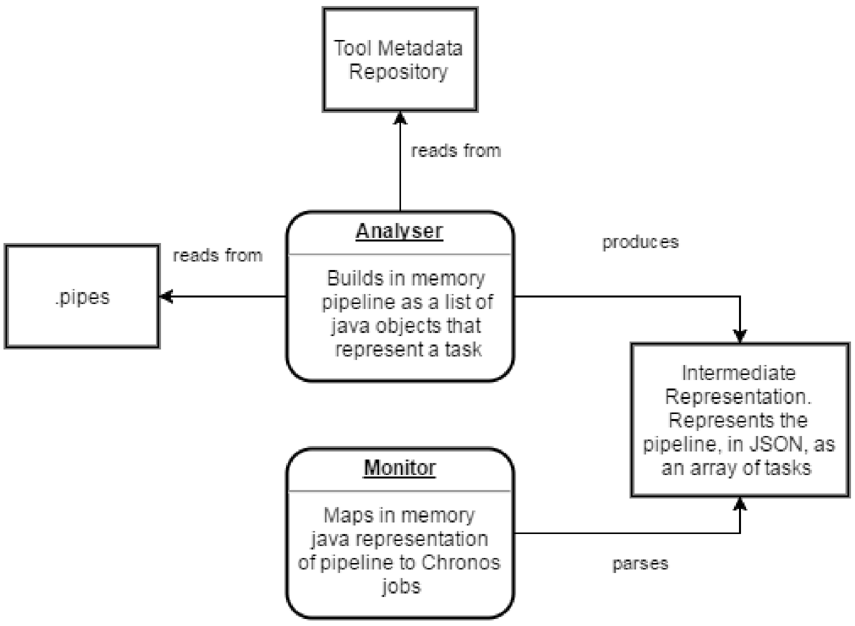

Tool Metadata Repository
reads from
Analyser
Builds in memory pipeline as a list of java objects that represent a task
reads from
.pipes
produces
Intermediate Representation. Represents the pipeline, in JSON, as an array of tasks
Monitor
Maps in memory java representation of pipeline to Chronos jobs
parses

Figure 5.2. Pipeline abstraction flow diagram

Figure 5.2 expresses all the transformations the pipeline representation suffers during the execution of NGS4Cloud. The pipeline representation, built on tasks, is firstly created in the analyser with the information present in the tool meta-data repository and in the `.pipes` file. The pipeline exists in the analyser as a list of java objects where each represents a task. These objects are then used to write the intermediate representation, in JSON. In the intermediate representation the pipeline is represented by a property named `tasks`, which is an array. Each object of the array `tasks` represents a task. The monitor will parse the intermediate representation, and then the pipeline will again be represented as a list of java objects, where each object represents a task. Having the tasks in memory, the monitor will map each one of them to a Chronos Job and send them to be processed.

5.4 Conclusion

We believe we've achieved a highly modular architecture where each component is loosely coupled with each other, and therefore easily replaced. This makes our system more testable, extensible and adaptable. Scalability is also a quality due to the use of Mesos (Ignazio, 2016).

6

Implementation

In this chapter we describe in more detail each component of NGS4Cloud, how they interact, their functionalities and implementation details.

6.1 Analyser

In this section we will describe the *analyser*, in particular:

- How it processes the pipeline description file;
- How it uses the tool meta-data repository;
- How it handles data partitioning;
- How it determines the dependencies between pipeline tasks.

The analyser is the component that parses and processes the `.pipes` files and creates a file holding the intermediate representation that contains the instructions and information, like resources, necessary to run the pipeline. Using the pipeline description and the tool's meta-data repository to validate and enrich the pipeline description, the analyser produces an intermediate representation containing detailed instructions and information on how to execute the pipeline.

The analyser is a command line application with a single functionality, namely to produce an intermediate representation JSON file from a pipeline description file, with extension `.pipes`. For each command it will produce one or more tasks which will be specified in the intermediate representation. It receives as a parameter the file name of the `.pipes` file, the URL of the input file, and, optionally, the relative paths of the output files the users want to have access to, once the pipeline is done running. The input URL can point to a compressed file on the file system or a remote located compressed file.

If the user specified outputs, using the information passed in the pipeline description and present in the tool meta-data repository, the analyser can verify that the specified outputs will be produced when the pipeline is executed.

To parse the pipeline description, the NGSPipes parser is reused. The analyser generates in-memory representations of Java objects that resemble the pipeline. The information of these objects is validated using the tool meta-data repository to check the correctness of the different tools, commands and arguments, preventing the user from inserting incorrect information and producing erroneous pipelines.

Additionally, the analyser fetches, from the tool meta-data repository, the information regarding each command's recommended computational resources, the tools' configurators and the tool type.

Combining the tool meta-data repository information about command inputs and outputs from commands and the value of the corresponding arguments, presented in the newly generated Java pipeline representation, the analyser also infers dependencies between tasks. For instance, if the output of command A is the input of command B, then command A depends on command B, and thus B must be executed only after A. These dependencies are represented in the intermediate representation through the tasks' property `parents`.

Each tasks' `command` property string is then composed from the information contained in the Java objects correspondent to the command and its arguments. Since not every command's arguments are composed (or concatenated) in the same way, the tool meta-data repository contains information regarding the tool argument composer. Our library supports some common arguments composer and more can be added.

Data partitioning is solved in the analyser. Initially, we planned to automatize the data partitioning but figured that an optimal, generic way, to partition the data is not trivial in the DNA sequencing and NGS data context, since there is a plethora of different ways to split the different file formats. Instead, we give the users the option to break the data how they find appropriate. We decided to allow different types of tools besides the unit processing tools: the *splitting* tools, the *listing* tools, and the *joining* tools. The DSL applies to these tools just like it applies to unit processing tools. They are, however, processed differently by the analyser, depending on their tool type, identified in the tool meta-data.

Unlike unit processing tools, where each command is mapped into one task, a splitting command within a splitting tool generates one task corresponding to the splitting of the file plus N tasks per command, where N is the number of partitions of the file whereas each task processes a partition of the file.

Data partitioning allows users to work with and process multiple files having to specify each command only once, while treating the files like a single file. It means that when users split a file in ten, for instance, they do not have to include the same tool ten times in the pipeline for every partition: the analyser will do that for them.

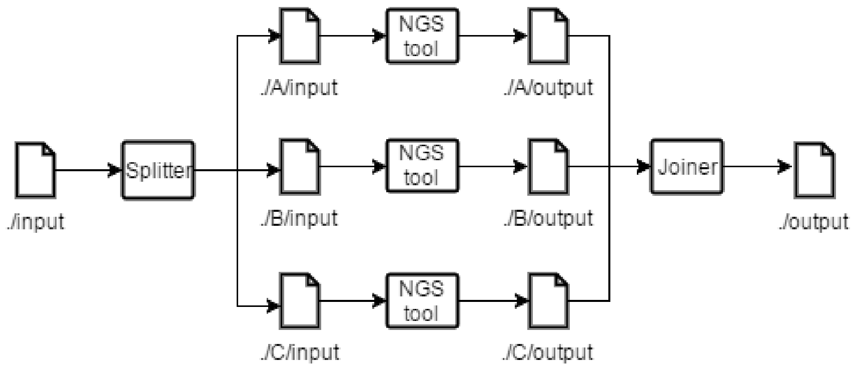


Figure 6.1. Splitting and Joining tools example

Figure 6.1 shows how a file named `input` is split originating three different files with the same name, stored in directories with different names.

For each partition the analyser will generate a directory where it stores the file partition with the same name it had before being partitioned. For every command specified in the pipeline description that uses the partitioned file, it is generated a task where the input path (partitioned file's path) is concatenated with the name of the directory where the partition of the file is stored. Multiple directories are created to avoid name collision between files generated.

Joining tools generate a task to join the partitions with the name of the input, that are stored in analyser generated directories (through either splitting or listing) corresponding to that input. Commands whose input depend on the join output will no longer have their tasks multiplied per partition.

In figure 6.2 it is depicted an example where a user wants to process different files of the same type, using the same tools, without having to specify each command more than once. Listing tools move and rename the files to match the same pattern as the splitting tools. After files are listed, they can be treated as one, as if it had been a split.

While the splitting tools are used to partition data files and apply the same command or set of commands to each partition, the listing tools allow users to

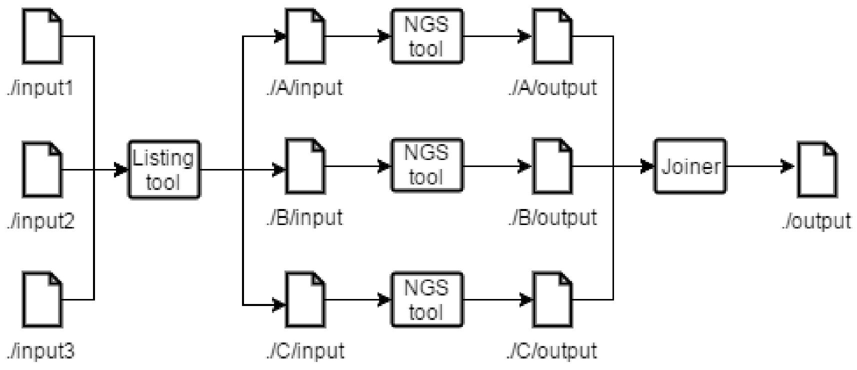


Figure 6.2. Listing and Joining tools example

apply the same command or set of commands to a list of specified files. Listing tools generate a task to move the files to the newly generated directories and change the files names to match the name used in the `.pipes` file. Listing tools generate the same tasks as splitting tools on commands that depend from them. The listing tools purpose is to allow a user to provide multiple files and treat them as one in the `.pipes` file.

The way data partitioning and dependencies inference is implemented, allows users to benefit from parallelization without adding complexity to the DSL and the process of writing a `.pipes` file.

The analyser also skims all the outputs that will be produced in order to create a list of the directories that have to exist for the swift and correct execution of the pipeline. These directories are stored in an array on the intermediate representation, under the `directories` property.

With the current version of the analyser, we have achieved a solution that allows users to split data and that allows to infer a topological graph from task dependencies, enabling the parallelization of the pipeline execution, without increasing the DSL complexity.

6.2 Intermediate Representation

In this section we explore the intermediate representation and explain what each property represents, how it is produced from a pipeline description and how it should be used by a Monitor.

As mentioned before, the *intermediate representation* was created to decouple the analyser from the Monitor at the language level. This means that we can develop the analyser in a different programming language than the Monitor. More importantly, it allows to use different job schedulers without modifying the analyser at all. Currently, we use the Chronos framework to schedule jobs on the cluster. The Chronos framework makes available a REST API, which means that we can interact with the framework without depending on a specific language. However, there are other Mesos's frameworks and there might be in the future more Mesos's frameworks that do not provide a REST API and, instead, provide a client library to schedule the jobs, which are not necessarily written in Java.

The intermediate representation is a JSON object, stored in a JSON format file. It follows the schema found in appendix A. The properties of the IR object are:

- Input
- Tasks
- Directories
- Outputs

Input

The `input` property is an URL which points to a compressed file which contains all the inputs necessary to execute the pipeline. This compressed file must be able to be decompressed by p7zip (see section 6.4). The URL can point to either a file stored remotely or in the user's computer.

Tasks

The `tasks` property is an array of tasks. Each task corresponds to the execution of a tool command, whether it is a unit processing tool, a splitting tool, a joining tool, or a listing tool. The *task* entity is the representation of a task in the context of a pipeline. It has information about the tool command to be executed, like the required resources and the docker image to setup the tool, as well as information about the execution itself, like which tasks it depends upon. Its properties are the following:

- `id` - unique inside a pipeline, identifies a task;
- `dockerImage` - string which identifies where the tool is installed. Is relevant to setup the tool;
- `command` - will cause the execution of the command with the necessary inputs. Is a string;
- `mem` - is an integer which specifies the amount of RAM, in MB, needed to run the command;

- `disk` - is an integer which specifies the amount of disk, in MB, needed to run the command;
- `cpus` - is a decimal which specifies the number of cpus needed to run the command;
- `parents` - array which contains the ids of all the tasks from which the task depends.

Directories

The `directories` property is an array of paths that contains every single directory that is necessary to exist in the cluster's file system to properly execute the pipeline. This is necessary to take the responsibility of analysing the command and determining which directories should be created, during the execution of the tasks, in the execution environment. The values present in this array are inferred by the analyser from the description of the pipeline and the meta-data of tools.

Outputs

The `outputs` property consists on an array of strings and each one identifies which files the user desires to download after the execution of the pipeline. The values contained in the array are passed by the user in the analyser's command line.

The intermediate representation is a format to describe pipelines. It has no explicit dependencies towards the NGS4Cloud context and can be used in other projects that need to represent a pipeline. It can be created via mechanisms other than the analyser's parsing and processing of a `.pipes` file.

6.3 Cluster

In this section we will cover the current structure of the cluster used for the remote execution of the pipeline.

When building the cluster structure our goals were to make efficient use of all its available resources, to guarantee we could execute independent pipeline tasks, in parallel, and to ensure it had support for Docker in order to allow an easy installation of any tool used in the pipeline.

As already said in subsection 5.2.4, to meet the goals set for the cluster we used the frameworks Mesos and Chronos. Mesos guarantees that the resources of our cluster will be efficiently used and it has Docker support. Chronos is a Mesos framework built for the execution of batch job which has support for dependent jobs.

Although Mesos has support for Docker, it is not included in its distribution. Thus all Mesos-Slaves of the cluster must have Docker installed.

However, the execution of the pipeline in a distributed environment brought a problem regarding the access of all machines to the files being produced during its execution. To solve this problem we integrated in the cluster a distributed file system to which all the Mesos-Slaves must have access. This way, all Mesos-Slaves can execute any task of the pipeline without having to do any special procedure to have access to the file they are going to process. Currently, the distributed file system configured in the cluster must follow a protocol similar to the Network File System Protocol (NFS)[1], such that it allows access to the distributed file system similarly to a local storage, thus allowing us to use a task's tool command with no special concerns regarding the fact that its inputs and its produced outputs are stored in a distributed file system.

Since the pipeline input files can be stored in the user's machine, we need to upload them to the cluster's shared directory in order to be able to execute the pipeline. To achieve this, the cluster must provide a machine capable of being accessed through Secure Shell (SSH)[2] so that the input files can be uploaded. The SSH connection is also necessary to download the outputs of the pipeline.

As already mentioned, the cluster must have Chronos installed. However, besides having Chronos installed, an endpoint to the Chronos REST API must be made available so that NGS4Cloud can schedule the pipeline tasks to be executed.

A cluster used by NGS4Cloud must provide:

1. An endpoint to Chronos REST API;
2. Support for the execution of Docker jobs on all Mesos-Slaves;
3. A NFS accessible on all Mesos-Slaves;
4. SSH access to a cluster machine that can interact with the cluster's NFS;

The use of Mesos and Chronos, besides allowing us to achieve the goals described in the beginning of this chapter, also decouples our solution from any concrete cloud service like Microsoft Azure or Amazon Cloud Services. Since Mesos and Chronos are currently being actively developed, we believe that NGS4Cloud will be able to take advantage of improvements to these technologies. It is also important to notice

[1] https://tools.ietf.org/html/rfc1094, last visited on 05/06/2016
[2] https://tools.ietf.org/html/rfc4253, last visited on 05/06/2016

that Mesos and Chronos provide features that could improve NGS4Cloud in ways that are not contemplated in this project. As an example, although data locality is not considered when executing a pipeline, Mesos offers a system called `constraints` that enables the locality of a file to be taken into account when choosing the cluster machine to execute the pipeline task.

6.4 Monitor

In this section we explore the software component monitor, namely its functionalities, the multiple interactions it has with the cluster, and how they are supported.

The `monitor` is a command line application which has three functionalities:

1. Launch the remote execution of the pipeline from an intermediate representation;
2. Consult the current status of the launched pipeline;
3. Download the outputs produced by the pipeline.

To support the last two functionalities, the monitor uses persistent storage to save the information about the Chronos Jobs that constitute a pipeline execution and what output files should be downloaded once the execution is finished.

6.4.1 Launch pipeline execution

To launch the execution of the pipeline a user can run the following command:

```
java -jar Monitor.jar launch ir.json
```

The `ir.json` file used on the command above is a file containing the intermediate representation of the pipeline to be launched.

The process to launch the pipeline starts by parsing the given intermediate representation into java objects that will contain the information stored in the intermediate representation.

After parsing the intermediate representation, it follows the creation of a working directory, in the distributed file system of the cluster, for the execution of the pipeline. The creation of this directory will depend on the location of the input file. If the input is stored locally, it will be created through an SSH connection to a cluster's machine; if the input is stored remotely, it will be created through a Chronos Job.

Having created the pipeline's work directory. it follows the uploading of the input file. From the analysis of the intermediate representation the monitor determine if the input is stored locally or remotely. This is possible because `input` is always an

URL, which allows us to infer, from the scheme, the input file location. If the input file is locally stored, an SSH access to a machine of the cluster will be made in order to upload it. When it's remotely stored, its download to the cluster will be made through a Chronos Job.

Having parsed the intermediate representation and uploaded the input file, if it is locally stored, the monitor proceeds to launch the Chronos Jobs. The Chronos Jobs launched can be divided in three different types: setup jobs, jobs from tasks, and clean up jobs. When the pipeline input is remotely located there will be two setup jobs. The first job will create the base directory for the execution of the pipeline and will then use the application `wget`[3] to download the input to the cluster. The second job will decompress the input file just downloaded, using the application `p7zip`[4], and will create the rest of the necessary directories to execute the pipeline. If the pipeline input is located in the user computer, only the setup job regarding the decompression of the input and the creation of the pipeline work directories will be launched.

Notice that these setup jobs are always executed using a docker container. This was necessary in order to execute the applications `wget` and `p7zip` without having to previously install them on the cluster. Although we provide docker images with theses applications installed on them, the user can change these docker images in the monitor configurations (see subsection 6.4.4). We choose `wget` over its alternative `curl`, because `wget` is simpler to use than `curl` when it comes to downloading files from an URL. We choose `p7zip` because it's the decompresser we found which supports more formats.

All the jobs are launched using Chronos REST API. The jobs from tasks are jobs originated from the tasks described in the intermediate representation, which will only be executed after the setup jobs are concluded. The cleanup jobs will be executed after all the jobs from tasks have been executed. Cleanup jobs perform two tasks, first they reduce the permissions necessary to access the pipeline outputs, then they launch a Chronos Job that will delete all the files and directories regarding the executed pipeline 7 days after its execution has finished. Changing the access permission on the pipeline outputs is necessary because outputs produced from docker jobs can only be accessed by the root user. This means that without changing the access permission to allow non-root users to access the files, only a root user would be able to download the outputs of the pipeline.

[3] https://www.gnu.org/software/wget/, last visited on 2016/09/13
[4] http://p7zip.sourceforge.net/, last visited on 2016/09/13

After launching the pipeline execution, an identifier is generated. The identifier is then saved to persistent storage. It is associated with the name of the Chronos Jobs that compose the pipeline, the output files that should be downloaded to the user's machine once the execution is finished, and the pipeline working directory. The output files that should be downloaded are specified in the intermediate representation.

This command returns the identifier of the pipeline execution.

6.4.2 Consult pipeline status

To consult a pipeline progression a user can run the following command:

```
java -jar Monitor.jar status id
```

This command receives an `id` which identifies a pipeline execution.

Using the id, the monitor accesses the persistent storage and fetches the name of all the Chronos jobs associated with it. Then a request to list all the jobs current status is made to the Chronos REST API. The information retrieved from the request is then matched with the information fetched from the persistent storage in order to know which jobs belong to the pipeline execution specified with the id.

If any of the Chronos Jobs that constitute the pipeline failed, this command will return an error message. In case all the jobs are finished, a success message is returned. When the execution is still in progress and no error has occurred, a message indicating how many jobs have been finished and how many are in total is presented.

6.4.3 Download pipeline outputs

To download the outputs of the pipeline a user can run the following command:

```
java -jar Monitor.jar outputs id
```

This command receives an id which identifies a pipeline execution.

First an access to the persistent storage is made to retrieve all the stored information regarding a pipeline execution. In the same way that is done when consulting the pipeline's progress status, a request to the Chronos REST API will be made, and its result will be matched with the data stored in the persistent storage to determine if the pipeline execution has already finished. If the execution is yet to be concluded, a warning message, stating that the execution of the pipeline is not concluded and the outputs cannot be downloaded, is returned. When the pipeline has finished executing, an SSH connection is established to a cluster machine, which has access to the distributed file system, and the specified outputs will be downloaded to the directory in which the user has executed this command.

6.4.4 Persistent storage

To save the information regarding the execution of a pipeline, we are using a flat file database. With this kind of database, a user doesn't need to install any database service and it's as efficient as an access to disk.

Since there is no concurrency in the access to the database, no complex query needs to be carried out and its data is immutable, the use of a flat file database is appropriate.

6.4.5 Configurations

The monitor configurations are done through the environment variable `NGS4_CLOUD_MONITOR_CONFIGS`, whose value points to a file in which all the configurations are specified.

The configuration is a `.txt` file containing name-value pairs separated by an equals sign. This file includes information regarding the SSH access to the cluster, the path to a directory of the cluster's distributed file system, and the Chronos endpoint. When writing the configuration file the following properties are required:

- `SSH_HOST`;
- `SSH_PORT`;
- `SSH_USER`;
- `CHRONOS_HOST`;
- `CHRONOS_PORT`;
- `CLUSTER_SHARED_DIR_PATH`;
- `EXECUTION_TRACKER_REPO_PATH`;
- `PIPELINE_OWNER`;
- `WGET_DOCKER_IMAGE`;
- `P7ZIP_DOCKER_IMAGE`.

`SSH_HOST` and `SSH_PORT` define the endpoint to access a machine of the cluster, via SSH, which has access to the directory belonging to the distributed file system of the cluster. `SSH_USER` defines the user that wants to establish an SSH connection to the cluster's machine. `CHRONOS_HOST` and `CHRONOS_PORT` define the endpoint to access Chronos. `CLUSTER_SHARED_DIR_PATH` is the path to a directory of the distributed file system. `EXECUTION_TRACKER_REPO_PATH` is the path to a directory of the machine in which the monitor will be executed. This directory contains the information regarding the execution of pipelines. `PIPELINE_OWNER` is the email address of the person responsible for the execution of the pipeline. When the pipeline execution has finished, an email will be sent to this address. Notice that this will only happen if the cluster is configured to do so. `WGET_DOCKER_IMAGE` is the name of a docker image that has `wget` installed in it. This image will be used in the setup of the pipeline execution. If a user does not have access to a docker image with `wget` installed he can use the image `jnforja/wget`. `P7ZIP_DOCKER_IMAGE` is the name of a docker image that has `p7zip` installed in it. This image will be used in the setup of the pipeline execution. If a user does not have access to a docker image with `p7zip` installed he can use the image `jnforja/7zip`.

The reason for `WGET_DOCKER_IMAGE` and `P7ZIP_DOCKER_IMAGE` being configurable is because we didn't want to have to recompile and redeploy Monitor every time an update to `wget` or to `p7zip` is made. With this approach when new docker images with updated versions of these applications are made, we only have to change these configurations.

To establish an SSH connection using password authentication, one should define the property `SSH_PASS`. To establish an SSH connection using public key authentication one must define the following properties:

- `SSH_PRIVATE_KEY_PATH`;
- `SSH_PRIVATE_KEY_PASS`;
- `SSH_PUBLIC_KEY_PATH`.

`SSH_PRIVATE_KEY_PATH` is the path to the file with the private key; the password associated with the private key is `SSH_PRIVATE_KEY_PASS`; and `SSH_PUBLIC_KEY_PATH` is the path to the file with the public key.

The configuration file must have the information necessary to do one type of authenticated SSH access to the cluster. Unauthenticated SSH accesses are not supported. An example configurations file can be found in appendix B.

7

Experimental Evaluation

In order to test the solution, we ran some pipelines on an OpenStack cluster made available by INCD (http://www.incd.pt). By the time of these tests[1], this cluster offered 16 GB RAM and 8 CPUs.

The descriptions and intermediate representation files of the two pipelines we used can be found in the appendices (see appendices D and F). The input size of the first pipeline is 18.3MB, and the input size of the second pipeline is 14MB.

We performed sequential executions and parallel executions of the pipelines. As expected, we obtained relatively different times for the two different patterns of executions. The table down below lists the different execution patterns (sequential vs parallel) for each pipeline and the average time of values obtained for 10 executions of each pipeline and pattern combinations.

Pipeline	Sequential Execution (s)	Parallel Execution (s)
snppipeline	125.6	77.3
blastnpipeline	160.3	44.3

Table 7.1. Sequential and parallel cxecution comparision for different pipelines.

Even with small inputs such as the ones used, the difference between sequential and parallel execution is obvious, with the parallel being faster.

We have not taken times for the execution of the analyser and processing of descriptions of pipelines as these are supposed to be swiftly analysed originating the intermediate representation files that are used as input in the monitor.

[1] 17/09/2016

8 Final Remarks

With the development and completion of the project, we noticed a couple modifications and additions that we would like to remark.

Firstly, we believe that in order to take best advantage of the resources made available by the machines used to process the pipeline, a more detailed and more precise method of evaluating the resources used by each task of the pipeline is required, as relying on pre-established values by a technician that configures the tool meta-data repository is not adequate. Particularly because these vary depending on the different inputs and options used when executing the tool command. In our opinion, it would be rather interesting to develop (or use, if such contraption already exists) a system that is able to test a tool in order to determine, for example, an equation for the necessary memory, disk and cpus, depending on the commands' arguments. This way Mesos would allocate its slaves and distribute the resources more efficiently.

Secondly, in our solution we do not use a custom Mesos framework as the development of such piece of software did not seem suitable for the time period available for our Bachelor's project. However, we are aware that using a custom Mesos framework dedicated to executing pipelines in a Mesos cluster would help improving performance and distributing resources by contemplating certain aspects like data locality and using advanced features provided by Mesos, only accessible in case we develop a custom framework[1].

Thirdly, it would be interesting to execute tasks from the same pipeline both locally and on the cloud. This way, tasks that do not require more computational resources than the normal personal computer makes available, could be executed on the users' personal computers where the monitor is running, saving the monetary resources that using the cloud may cost.

[1] http://mesos.apache.org/documentation/latest/, last visited on 17/07/2016

In fourth place, we think that the analyser generating a graphical representation of the intermediate representation, describing each task in a block and dependencies between commands with pointing arrows would be very helpful in letting users understand the pipeline generated by the analyser and sharing with other people what they are doing.

Lastly, we think that the next step for the monitor would be for it to merely require access to a cluster with a Chronos endpoint, totally decoupling the NGS4Cloud solution from the cluster used and its configuration. In order to achieve that, we would have to come up with a solution that enables us to execute the pipeline without requiring a distributed file system configured on the cluster and a way of uploading and downloading files from the cluster to the local machine without the need to establish an SSH connection with the cluster.

References

Bowtie2. http://bowtie-bio.sourceforge.net/bowtie2/index.shtml, 2016. [Online, accessed 2016/09/18].

Chronos. https://mesos.github.io/chronos/, 2016. [Online, accessed 2016/04/26].

Rfc1094 - nfs. https://tools.ietf.org/html/rfc1094, 2016. [Online, accessed 2016/06/05].

Ngspipes github. http://ngspipes.github.io/, 2016. [Online, accessed 2016/06/11].

Samtools. http://samtools.sourceforge.net/, 2016. [Online, accessed 2016/09/18].

Rfc4253 - ssh. https://tools.ietf.org/html/rfc4253, 2016. [Online, accessed 2016/06/05].

Samtools documentation. http://samtools.sourceforge.net/, 2016. [Online, accessed 2016/09/18].

Snp-pipeline. http://snp-pipeline.readthedocs.io/en/latest/, 2016. [Online, accessed 2016/09/18].

Vmware. http://www.vmware.com/products/player/playerpro-evaluation.html, 2016. [Online, accessed 2016/09/14].

Varscan. http://varscan.sourceforge.net/,lastvisitedon18/09/2016, 2016. [Online, accessed 2016/09/18].

Virtualbox. https://www.virtualbox.org/, 2016. [Online, accessed 2016/09/21].

Wikipedia vmdk. https://en.wikipedia.org/wiki/VMDK/, 2016. [Online, accessed 2016/09/14].

p7zip. http://p7zip.sourceforge.net/, 2016. [Online, accessed 2016/09/13].

wget. https://www.gnu.org/software/wget/, 2016. [Online, accessed 2016/09/13].

M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, Apr. 2010. ISSN 0001-0782. doi: 10.1145/1721654.1721672. URL http://doi.acm.org/10.1145/1721654.1721672. 2

B. Dantas and C. Fleitas. Infraestrutura de suporte à execução de fluxos de trabalho para bioinformática. Diploma Thesis, Instituto Superior de Engenharia de Lisboa, 2015. i, iii, 1, 5

J. Forja, A. Almeida, A. P. Francisco, J. Simão, and C. Vaz. NGS4Cloud: Cloud-based NGS Data Processing INForum, September 2016. 2

D. Greenberg. *Building applications on Mesos.* O'Reilly, Sebastopol, 2016. ISBN 149192652X.

R. Ignazio. *Mesos in Action.* Manning Pubns Co, City, 2016. ISBN 1617292923. 7, 24, 34

J. Nickoloff. *Docker in action.* Manning Publications, Shelter Island, NY, 2016. ISBN 1633430235. 7, 13, 23, 24

S. C. Shuster. Next-generation sequencing transforms today's biology. *Nature Methods*, 5(1):16–18, Jan. 2008. i, iii, 1

Appendices

A Intermediate Representation Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "inputs": {
     "type": "array",
     "items": {
       "type": "object",
      "properties": {
        "src": {"type": "string", "format":"uri"},
        "filesDests": {
          "type": "array",
         "items":{
           "type": "object",
           "properties": {
             "file": {"type": "string", "minLength": 1},
            "dest": {"type": "string", "minLength": 0}
           },
           "required": ["file", "dest"]
         }
        }
      }
     }
   },
   "outputs": {
     "type": "array",
     "items": { "type": "string" }
   }
   "tasks": {
     "type": "array",
     "items": {
       "type": "object",
      "properties":{
        "id":{"type":"integer"},
        "dockerImage":{"type":"string", "minLength": 1},
        "command":{"type":"string", "minLength": 1},
        "mem": {"type":"integer"},
        "disk": {"type":"integer"},
        "cpus": {"type":"integer"},
        "parents": {
          "type": "array",
         "items":{
           "type": "integer"
         }
        }
      },
        "required": ["id", "dockerImage", "command", "mem", "disk", "cpus", "parents"]
     }
   },
   "directories": {
     "type": "array",
     "items": { "type": "string" }
   }
  },
   "required": ["inputs", "outputs", "tasks", "directories"]
}
```

B Monitor Configuration File

```
SSH_HOST = 127.0.0.1
SSH_PORT = 22
SSH_USER = ngs4cloud
SSH_PRIVATE_KEY_PATH = C:\Users\NGS4Cloud\privkey.ppk
SSH_PRIVATE_KEY_PASS = privkeyPass123
SSH_PUBLIC_KEY_PATH = C:\Users\NGS4Cloud\pubkey
CHRONOS_HOST = 127.0.0.1
CHRONOS_PORT = 8080
CLUST_SHARED_DIR_PATH = /shared
WGET_DOCKER_IMAGE = jnforja/wget
P7ZIP_DOCKER_IMAGE = jnforja/7zip
```

C TMR Descriptor Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "name": {
      "type": "string"
    },
    "author": {
      "type": "string"
    },
    "version": {
      "type": "string"
    },
    "description": {
      "type": "string"
    },
    "documentation": {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "setup": {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "toolType": {
      "type": "string",
      "enum": [
        "Unit",
        "splitting",
        "joining",
        "listing"
      ]
    },
    "requiredMemory": {
      "type": "integer"
    },
    "recommendedCpus": {
      "type": "integer"
    },
    "recommendedDiskSpace": {
      "type": "integer"
    },
    "commands": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "name": {
            "type": "string"
        },
        "command": {
          "type": "string"
        },
        "description": {
          "type": "string"
        },
        "priority": {
          "type": "integer"
        },
        "argumentsComposer": {
          "type": "string"
        },
        "arguments": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "name": {
                "type": "string"
              },
              "argumentType": {
                "type": "string",
                "enum": [
                  "int",
                  "file",
                  "string",
                  "double",
                  "directory"
                ],
                "isRequired": {
                  "type": "string",
                  "enum": [
                    "true",
                    "false"
                  ]
                }
              },
              "description": {
                "type": "string"
              }
            },
            "required": [
              "name",
              "argumentType",
              "isRequired",
              "description"
            ]
          }
        },
        "outputs": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
```

```
          "name": {
            "type": "string"
          },
          "description": {
            "type": "string"
          },
          "outputType": {
            "type": "string",
            "enum": [
              "directory_dependent",
              "file_dependent",
              "independent"
            ]
          },
          "argument_name": {
            "type": "string"
          },
          "value": {
            "type": "string"
          }
        },
        "required": [
          "name",
          "description",
          "outputType",
          "argument_name",
          "value"
        ]
      }
    },
    "inputs": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "name": {
            "type": "string"
          },
          "description": {
            "type": "string"
          },
          "inputType": {
            "type": "string",
            "enum": [
              "directory_dependent",
              "file_dependent",
              "independent"
            ]
          }
        },
        "required": [
          "name",
          "description",
          "inputType",
          "argument_name",
                "value"
              ]
            }
          }
        },
        "required": [
          "name",
          "command",
          "description",
          "priority",
          "argumentsComposer",
          "arguments",
          "outputs",
          "inputs"
        ]
      }
    }
  },
  "required": [
    "name",
    "author",
    "version",
    "description",
    "documentation",
    "setup",
    "toolType",
    "requiredMemory",
    "recommendedDiskSpace",
    "recommendedCpus",
    "commands"
  ]
}
```

D Case Study Snp-pipeline Pipeline Description

```
Pipeline "Github" "https://github.com/Vacalexis/tools" {

   tool "snp-pipeline" "DockerConfig" {
      command "create_sample_dirs" {
         argument "-d" "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/*"
         argument "--output"
              "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/sampleDirectories.txt"
      }
   }

   tool "Bowtie2" "DockerConfig" {
      command "bowtie2-build" {
         argument "reference_in"
              "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reference/lambda_virus.fasta"
         argument "bt2_base" "reference"
      }
      command "bowtie2" {
            argument "-p" "1"
         argument "-q" "-q"
         argument "-x" "reference"
         argument "-1"
              "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample1/sample1_1.fastq"
          argument "-2"
               "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample1/sample1_2.fastq"
         argument "-S" "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads1.sam"
      }
      command "bowtie2" {
         argument "-p" "1"
         argument "-q" "-q"
         argument "-x" "reference"
         argument "-1"
              "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample2/sample2_1.fastq"
         argument "-2"
              "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample2/sample2_2.fastq"
         argument "-S" "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads2.sam"
      }
      command "bowtie2" {
         argument "-p" "1"
         argument "-q" "-q"
         argument "-x" "reference"
         argument "-1"
              "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample3/sample3_1.fastq"
         argument "-2"
              "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample3/sample3_2.fastq"
         argument "-S" "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads3.sam"
      }
      command "bowtie2" {
         argument "-p" "1"
         argument "-q" "-q"
         argument "-x" "reference"
         argument "-1"
              "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample4/sample4_1.fastq"
      argument "-2"
           "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample4/sample4_2.fastq"
      argument "-S" "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads4.sam"
   }

}
tool "Listing" "DockerConfig" {
   command "startListing" {
      argument "referenceName" "reads.sam"
      argument "filesList" "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads1.sam
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads2.sam
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads3.sam
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads4.sam"
   }
}
tool "Samtools" "DockerConfig" {

      command "view" {
      argument "-b" "-b"
      argument "-S" "-S"
      argument "-F" "4"
      argument "-o" "reads.unsorted.bam"
      argument "input" "reads.sam"
   }
      command "sort" {
      argument "-o" "reads.sorted.bam"
      argument "input" "reads.unsorted.bam"
   }
   command "mpileup" {
      argument "--fasta-ref"
           "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reference/lambda_virus.fasta"
      argument "input" "reads.sorted.bam"
      argument "--output" "reads.pileup"
   }
}
tool "VarScan" "DockerConfig" {
   command "mpileup2snp" {
      argument "mpileupFile" "reads.pileup"
      argument "--min-var-freq" "0.90"
      argument "--output-vcf" "1"
      argument "output" "var.flt.vcf"
   }
}
tool "Listing" "DockerConfig" {
   command "stopListing" {
      argument "referenceName" "var.flt.vcf"
      argument "destinationFiles"
           "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample1/var.flt.vcf
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample2/var.flt.vcf
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample3/var.flt.vcf
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample4/var.flt.vcf"
   }
}
tool "snp-pipeline" "DockerConfig" {
   command "create_snp_list" {
```

```
            argument "--vcfname" "var.flt.vcf"
            argument "--output" "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snplist.txt"
            argument "sampleDirsFile"
                 "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/sampleDirectories.txt"
        }
    }
    tool "Listing" "DockerConfig" {
        command "restartListing" {
            argument "referenceName" "reads.pileup"
        }
    }
    tool "snp-pipeline" "DockerConfig" {
        command "call_consensus" {
            argument "--snpListFile"
                 "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snplist.txt"
            argument "--output" "consensus.fasta"
            argument "--vcfFileName" "consensus.vcf "
            argument "allPileupFile" "reads.pileup"
        }
    }
    tool "Listing" "DockerConfig" {
        command "stopListing" {
            argument "referenceName" "consensus.fasta"
            argument "destinationFiles"
                 "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample1/consensus.fasta
                 snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample2/consensus.fasta
                 snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample3/consensus.fasta
                 snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample4/consensus.fasta"
        }
    }
    tool "snp-pipeline" "DockerConfig" {
        command "create_snp_matrix" {
            argument "sampleDirsFile"
                 "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/sampleDirectories.txt"
            argument "--consFileName" "consensus.fasta"
            argument "--output" "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snpma.fasta"
        }
    }
}
```

E Case Study Snp-pipeline Pipeline Intermediate Representation

```
{
    "input":"https://github.com/CFSAN-Biostatistics/snp-pipeline/archive/master.zip",
    "tasks":[
        {
            "id":1,
            "dockerImage":"vacalexis/snp-pipeline",
            "command":"ls -d snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/* >
                 snp-pipeline-master/snppipeline/data/lambdaVirusInputs/sampleDirectories.txt",
            "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[

    ]
  },
  {
    "id":2,
    "dockerImage":"tobiasschroff/bowtie2:v2",
    "command":"bowtie2-build
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reference/lambda_virus.fasta
        reference ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[

    ]
  },
  {
    "id":3,
    "dockerImage":"tobiasschroff/bowtie2:v2",
    "command":"bowtie2 -p 1 -q -x reference -1
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample1/sample1_1.fastq
        -2
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample1/sample1_2.fastq
        -S snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads1.sam ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      2
    ]
  },
  {
    "id":4,
    "dockerImage":"tobiasschroff/bowtie2:v2",
    "command":"bowtie2 -p 1 -q -x reference -1
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample2/sample2_1.fastq
        -2
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample2/sample2_2.fastq
        -S snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads2.sam ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      2
    ]
  },
  {
    "id":5,
    "dockerImage":"tobiasschroff/bowtie2:v2",
    "command":"bowtie2 -p 1 -q -x reference -1
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample3/sample3_1.fastq
        -2
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample3/sample3_2.fastq
        -S snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads3.sam ",
   "mem":2048,
   "disk":1,
   "cpus":1,
   "parents":[
      2
   ]
},
{
   "id":6,
   "dockerImage":"tobiasschroff/bowtie2:v2",
   "command":"bowtie2 -p 1 -q -x reference -1
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample4/sample4_1.fastq
        -2
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample4/sample4_2.fastq
        -S snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads4.sam ",
   "mem":2048,
   "disk":1,
   "cpus":1,
   "parents":[
      2
   ]
},
{
   "id":7,
   "dockerImage":"vacalexis/split",
   "command":"/Split/startListing.sh reads.sam
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads1.sam
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads2.sam
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads3.sam
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads4.sam ",
   "mem":1024,
   "disk":1,
   "cpus":1,
   "parents":[
      3,
      4,
      5,
      6
   ]
},
{
   "id":8,
   "dockerImage":"jnforja/samtools",
   "command":"samtools view -b -S -F 4 -o ngs4clouddir1/reads.unsorted.bam
        ngs4clouddir1/reads.sam ",
   "mem":4096,
   "disk":1,
   "cpus":1,
   "parents":[
      7
   ]
},
{
      "id":9,
      "dockerImage":"jnforja/samtools",
      "command":"samtools view -b -S -F 4 -o ngs4clouddir2/reads.unsorted.bam
           ngs4clouddir2/reads.sam ",
      "mem":4096,
      "disk":1,
      "cpus":1,
      "parents":[
         7
      ]
   },
   {
      "id":10,
      "dockerImage":"jnforja/samtools",
      "command":"samtools view -b -S -F 4 -o ngs4clouddir3/reads.unsorted.bam
           ngs4clouddir3/reads.sam ",
      "mem":4096,
      "disk":1,
      "cpus":1,
      "parents":[
         7
      ]
   },
   {
      "id":11,
      "dockerImage":"jnforja/samtools",
      "command":"samtools view -b -S -F 4 -o ngs4clouddir4/reads.unsorted.bam
           ngs4clouddir4/reads.sam ",
      "mem":4096,
      "disk":1,
      "cpus":1,
      "parents":[
         7
      ]
   },
   {
      "id":12,
      "dockerImage":"jnforja/samtools",
      "command":"samtools sort -o ngs4clouddir1/reads.sorted.bam ngs4clouddir1/reads.unsorted.bam
           ",
      "mem":4096,
      "disk":1,
      "cpus":1,
      "parents":[
         8
      ]
   },
   {
      "id":13,
      "dockerImage":"jnforja/samtools",
      "command":"samtools sort -o ngs4clouddir2/reads.sorted.bam ngs4clouddir2/reads.unsorted.bam
           ",
      "mem":4096,
      "disk":1,
      "cpus":1,
  "parents":[
    9
  ]
},
{
  "id":14,
  "dockerImage":"jnforja/samtools",
  "command":"samtools sort -o ngs4clouddir3/reads.sorted.bam ngs4clouddir3/reads.unsorted.bam
      ",
  "mem":4096,
  "disk":1,
  "cpus":1,
  "parents":[
    10
  ]
},
{
  "id":15,
  "dockerImage":"jnforja/samtools",
  "command":"samtools sort -o ngs4clouddir4/reads.sorted.bam ngs4clouddir4/reads.unsorted.bam
      ",
  "mem":4096,
  "disk":1,
  "cpus":1,
  "parents":[
    11
  ]
},
{
  "id":16,
  "dockerImage":"jnforja/samtools",
  "command":"samtools mpileup --fasta-ref
      snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reference/lambda_virus.fasta
      ngs4clouddir1/reads.sorted.bam --output ngs4clouddir1/reads.pileup ",
  "mem":4096,
  "disk":1,
  "cpus":1,
  "parents":[
    12
  ]
},
{
  "id":17,
  "dockerImage":"jnforja/samtools",
  "command":"samtools mpileup --fasta-ref
      snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reference/lambda_virus.fasta
      ngs4clouddir2/reads.sorted.bam --output ngs4clouddir2/reads.pileup ",
  "mem":4096,
  "disk":1,
  "cpus":1,
  "parents":[
    13
  ]
},
{
    "id":18,
    "dockerImage":"jnforja/samtools",
    "command":"samtools mpileup --fasta-ref
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reference/lambda_virus.fasta
        ngs4clouddir3/reads.sorted.bam --output ngs4clouddir3/reads.pileup ",
    "mem":4096,
    "disk":1,
    "cpus":1,
    "parents":[
      14
    ]
  },
  {
    "id":19,
    "dockerImage":"jnforja/samtools",
    "command":"samtools mpileup --fasta-ref
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reference/lambda_virus.fasta
        ngs4clouddir4/reads.sorted.bam --output ngs4clouddir4/reads.pileup ",
    "mem":4096,
    "disk":1,
    "cpus":1,
    "parents":[
      15
    ]
  },
  {
    "id":20,
    "dockerImage":"djordjeklisic/sbg-varscan2:v1",
    "command":"java -jar /VarScan2.3.7.jar mpileup2snp ngs4clouddir1/reads.pileup
        --min-var-freq 0.90 --output-vcf 1 >> ngs4clouddir1/var.flt.vcf ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      16
    ]
  },
  {
    "id":21,
    "dockerImage":"djordjeklisic/sbg-varscan2:v1",
    "command":"java -jar /VarScan2.3.7.jar mpileup2snp ngs4clouddir2/reads.pileup
        --min-var-freq 0.90 --output-vcf 1 >> ngs4clouddir2/var.flt.vcf ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      17
    ]
  },
  {
    "id":22,
    "dockerImage":"djordjeklisic/sbg-varscan2:v1",
    "command":"java -jar /VarScan2.3.7.jar mpileup2snp ngs4clouddir3/reads.pileup
        --min-var-freq 0.90 --output-vcf 1 >> ngs4clouddir3/var.flt.vcf ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      18
    ]
  },
  {
    "id":23,
    "dockerImage":"djordjeklisic/sbg-varscan2:v1",
    "command":"java -jar /VarScan2.3.7.jar mpileup2snp ngs4clouddir4/reads.pileup
        --min-var-freq 0.90 --output-vcf 1 >> ngs4clouddir4/var.flt.vcf ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      19
    ]
  },
  {
    "id":24,
    "dockerImage":"vacalexis/split",
    "command":"/Split/stopListing.sh var.flt.vcf
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample1/var.flt.vcf
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample2/var.flt.vcf
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample3/var.flt.vcf
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample4/var.flt.vcf ",
    "mem":1024,
    "disk":1,
    "cpus":1,
    "parents":[
      20,
      21,
      22,
      23
    ]
  },
  {
    "id":25,
    "dockerImage":"vacalexis/snp-pipeline",
    "command":"create_snp_list.py --vcfname var.flt.vcf --output
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snplist.txt
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/sampleDirectories.txt ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      1,
      24
    ]
  },
  {
    "id":26,
    "dockerImage":"vacalexis/snp-pipeline",
    "command":"call_consensus.py --snpListFile
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snplist.txt --output
        ngs4clouddir1/consensus.fasta --vcfFileName consensus.vcf ngs4clouddir1/reads.pileup ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      16,
      25
    ]
  },
  {
    "id":27,
    "dockerImage":"vacalexis/snp-pipeline",
    "command":"call_consensus.py --snpListFile
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snplist.txt --output
        ngs4clouddir2/consensus.fasta --vcfFileName consensus.vcf ngs4clouddir2/reads.pileup ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      17,
      25
    ]
  },
  {
    "id":28,
    "dockerImage":"vacalexis/snp-pipeline",
    "command":"call_consensus.py --snpListFile
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snplist.txt --output
        ngs4clouddir3/consensus.fasta --vcfFileName consensus.vcf ngs4clouddir3/reads.pileup ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      18,
      25
    ]
  },
  {
    "id":29,
    "dockerImage":"vacalexis/snp-pipeline",
    "command":"call_consensus.py --snpListFile
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snplist.txt --output
        ngs4clouddir4/consensus.fasta --vcfFileName consensus.vcf ngs4clouddir4/reads.pileup ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      19,
      25
    ]
  },
  {
    "id":30,
      "dockerImage":"vacalexis/split",
      "command":"/Split/stopListing.sh consensus.fasta
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample1/consensus.fasta
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample2/consensus.fasta
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample3/consensus.fasta
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample4/consensus.fasta
           ",
      "mem":1024,
      "disk":1,
      "cpus":1,
      "parents":[
        26,
        27,
        28,
        29
      ]
    },
    {
      "id":31,
      "dockerImage":"vacalexis/snp-pipeline",
      "command":"create_snp_matrix.py
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/sampleDirectories.txt
           --consFileName consensus.fasta --output
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snpma.fasta ",
      "mem":2048,
      "disk":1,
      "cpus":1,
      "parents":[
        1,
        30
      ]
    }
  ],
  "outputs":[
    "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/sampleDirectories.txt",
    "reference",
    "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads1.sam",
    "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads2.sam",
    "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads3.sam",
    "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads4.sam",
    "ngs4clouddir1/reads.sam",
    "ngs4clouddir2/reads.sam",
    "ngs4clouddir3/reads.sam",
    "ngs4clouddir4/reads.sam",
    "ngs4clouddir1/reads.unsorted.bam",
    "ngs4clouddir2/reads.unsorted.bam",
    "ngs4clouddir3/reads.unsorted.bam",
    "ngs4clouddir4/reads.unsorted.bam",
    "ngs4clouddir1/reads.sorted.bam",
    "ngs4clouddir2/reads.sorted.bam",
    "ngs4clouddir3/reads.sorted.bam",
    "ngs4clouddir4/reads.sorted.bam",
    "ngs4clouddir1/reads.pileup",
    "ngs4clouddir2/reads.pileup",
    "ngs4clouddir3/reads.pileup",
    "ngs4clouddir4/reads.pileup",
    "ngs4clouddir1/var.flt.vcf",
    "ngs4clouddir2/var.flt.vcf",
    "ngs4clouddir3/var.flt.vcf",
    "ngs4clouddir4/var.flt.vcf",
    "var.flt.vcf",
    "var.flt.vcf",
    "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snplist.txt",
    "ngs4clouddir1/consensus.fasta",
    "ngs4clouddir2/consensus.fasta",
    "ngs4clouddir3/consensus.fasta",
    "ngs4clouddir4/consensus.fasta",
    "consensus.fasta",
    "consensus.fasta",
    "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snpma.fasta"
  ],
  "directories":[
    "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reference",
    "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample1",
    "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample2",
    "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample3",
    "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample4",
    "snp-pipeline-master/snppipeline/data/lambdaVirusInputs",
    "ngs4clouddir1",
    "ngs4clouddir2",
    "ngs4clouddir3",
    "ngs4clouddir4"
  ]
}
```

F 49 Combinations Blastn Pipeline Description

```
Pipeline "Github" "https://github.com/Vacalexis/tools" {

   tool "snp-pipeline" "DockerConfig" {
      command "create_sample_dirs" {
         argument "-d" "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/*"
         argument "--output"
              "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/sampleDirectories.txt"
      }
   }

   tool "Bowtie2" "DockerConfig" {
      command "bowtie2-build" {
         argument "reference_in"
              "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reference/lambda_virus.fasta"
         argument "bt2_base" "reference"
      }
      command "bowtie2" {
            argument "-p" "1"
         argument "-q" "-q"
         argument "-x" "reference"
         argument "-1"
              "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample1/sample1_1.fastq"
          argument "-2"
               "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample1/sample1_2.fastq"
         argument "-S" "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads1.sam"
      }
      command "bowtie2" {
         argument "-p" "1"
         argument "-q" "-q"
         argument "-x" "reference"
         argument "-1"
              "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample2/sample2_1.fastq"
         argument "-2"
              "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample2/sample2_2.fastq"
         argument "-S" "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads2.sam"
      }
      command "bowtie2" {
         argument "-p" "1"
         argument "-q" "-q"
         argument "-x" "reference"
         argument "-1"
              "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample3/sample3_1.fastq"
         argument "-2"
              "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample3/sample3_2.fastq"
         argument "-S" "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads3.sam"
      }
      command "bowtie2" {
         argument "-p" "1"
         argument "-q" "-q"
         argument "-x" "reference"
         argument "-1"
              "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample4/sample4_1.fastq"
      argument "-2"
           "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample4/sample4_2.fastq"
      argument "-S" "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads4.sam"
   }

}
tool "Listing" "DockerConfig" {
   command "startListing" {
      argument "referenceName" "reads.sam"
      argument "filesList" "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads1.sam
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads2.sam
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads3.sam
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads4.sam"
   }
}
tool "Samtools" "DockerConfig" {

      command "view" {
      argument "-b" "-b"
      argument "-S" "-S"
      argument "-F" "4"
      argument "-o" "reads.unsorted.bam"
      argument "input" "reads.sam"
   }
      command "sort" {
      argument "-o" "reads.sorted.bam"
      argument "input" "reads.unsorted.bam"
   }
   command "mpileup" {
      argument "--fasta-ref"
           "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reference/lambda_virus.fasta"
      argument "input" "reads.sorted.bam"
      argument "--output" "reads.pileup"
   }
}
tool "VarScan" "DockerConfig" {
   command "mpileup2snp" {
      argument "mpileupFile" "reads.pileup"
      argument "--min-var-freq" "0.90"
      argument "--output-vcf" "1"
      argument "output" "var.flt.vcf"
   }
}
tool "Listing" "DockerConfig" {
   command "stopListing" {
      argument "referenceName" "var.flt.vcf"
      argument "destinationFiles"
           "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample1/var.flt.vcf
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample2/var.flt.vcf
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample3/var.flt.vcf
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample4/var.flt.vcf"
   }
}
tool "snp-pipeline" "DockerConfig" {
   command "create_snp_list" {
      argument "--vcfname" "var.flt.vcf"
      argument "--output" "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snplist.txt"
      argument "sampleDirsFile"
          "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/sampleDirectories.txt"
    }
  }
  tool "Listing" "DockerConfig" {
    command "restartListing" {
      argument "referenceName" "reads.pileup"
    }
  }
  tool "snp-pipeline" "DockerConfig" {
    command "call_consensus" {
      argument "--snpListFile"
          "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snplist.txt"
      argument "--output" "consensus.fasta"
      argument "--vcfFileName" "consensus.vcf "
      argument "allPileupFile" "reads.pileup"
    }
  }
  tool "Listing" "DockerConfig" {
    command "stopListing" {
      argument "referenceName" "consensus.fasta"
      argument "destinationFiles"
          "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample1/consensus.fasta
          snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample2/consensus.fasta
          snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample3/consensus.fasta
          snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample4/consensus.fasta"
    }
  }
  tool "snp-pipeline" "DockerConfig" {
    command "create_snp_matrix" {
      argument "sampleDirsFile"
          "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/sampleDirectories.txt"
      argument "--consFileName" "consensus.fasta"
      argument "--output" "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snpma.fasta"
    }
  }
}
```

G 49 Combinations Blastn Pipeline Intermediate Representation

```
{
  "input":"https://github.com/CFSAN-Biostatistics/snp-pipeline/archive/master.zip",
  "tasks":[
    {
      "id":1,
      "dockerImage":"vacalexis/snp-pipeline",
      "command":"ls -d snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/* >
          snp-pipeline-master/snppipeline/data/lambdaVirusInputs/sampleDirectories.txt",
      "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[

    ]
  },
  {
    "id":2,
    "dockerImage":"tobiasschroff/bowtie2:v2",
    "command":"bowtie2-build
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reference/lambda_virus.fasta
        reference ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[

    ]
  },
  {
    "id":3,
    "dockerImage":"tobiasschroff/bowtie2:v2",
    "command":"bowtie2 -p 1 -q -x reference -1
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample1/sample1_1.fastq
        -2
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample1/sample1_2.fastq
        -S snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads1.sam ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      2
    ]
  },
  {
    "id":4,
    "dockerImage":"tobiasschroff/bowtie2:v2",
    "command":"bowtie2 -p 1 -q -x reference -1
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample2/sample2_1.fastq
        -2
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample2/sample2_2.fastq
        -S snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads2.sam ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      2
    ]
  },
  {
    "id":5,
    "dockerImage":"tobiasschroff/bowtie2:v2",
    "command":"bowtie2 -p 1 -q -x reference -1
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample3/sample3_1.fastq
        -2
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample3/sample3_2.fastq
        -S snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads3.sam ",
   "mem":2048,
   "disk":1,
   "cpus":1,
   "parents":[
      2
   ]
},
{
   "id":6,
   "dockerImage":"tobiasschroff/bowtie2:v2",
   "command":"bowtie2 -p 1 -q -x reference -1
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample4/sample4_1.fastq
        -2
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample4/sample4_2.fastq
        -S snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads4.sam ",
   "mem":2048,
   "disk":1,
   "cpus":1,
   "parents":[
      2
   ]
},
{
   "id":7,
   "dockerImage":"vacalexis/split",
   "command":"/Split/startListing.sh reads.sam
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads1.sam
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads2.sam
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads3.sam
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads4.sam ",
   "mem":1024,
   "disk":1,
   "cpus":1,
   "parents":[
      3,
      4,
      5,
      6
   ]
},
{
   "id":8,
   "dockerImage":"jnforja/samtools",
   "command":"samtools view -b -S -F 4 -o ngs4clouddir1/reads.unsorted.bam
        ngs4clouddir1/reads.sam ",
   "mem":4096,
   "disk":1,
   "cpus":1,
   "parents":[
      7
   ]
},
{
    "id":9,
    "dockerImage":"jnforja/samtools",
    "command":"samtools view -b -S -F 4 -o ngs4clouddir2/reads.unsorted.bam
        ngs4clouddir2/reads.sam ",
    "mem":4096,
    "disk":1,
    "cpus":1,
    "parents":[
      7
    ]
  },
  {
    "id":10,
    "dockerImage":"jnforja/samtools",
    "command":"samtools view -b -S -F 4 -o ngs4clouddir3/reads.unsorted.bam
        ngs4clouddir3/reads.sam ",
    "mem":4096,
    "disk":1,
    "cpus":1,
    "parents":[
      7
    ]
  },
  {
    "id":11,
    "dockerImage":"jnforja/samtools",
    "command":"samtools view -b -S -F 4 -o ngs4clouddir4/reads.unsorted.bam
        ngs4clouddir4/reads.sam ",
    "mem":4096,
    "disk":1,
    "cpus":1,
    "parents":[
      7
    ]
  },
  {
    "id":12,
    "dockerImage":"jnforja/samtools",
    "command":"samtools sort -o ngs4clouddir1/reads.sorted.bam ngs4clouddir1/reads.unsorted.bam
        ",
    "mem":4096,
    "disk":1,
    "cpus":1,
    "parents":[
      8
    ]
  },
  {
    "id":13,
    "dockerImage":"jnforja/samtools",
    "command":"samtools sort -o ngs4clouddir2/reads.sorted.bam ngs4clouddir2/reads.unsorted.bam
        ",
    "mem":4096,
    "disk":1,
    "cpus":1,
    "parents":[
      9
    ]
  },
  {
    "id":14,
    "dockerImage":"jnforja/samtools",
    "command":"samtools sort -o ngs4clouddir3/reads.sorted.bam ngs4clouddir3/reads.unsorted.bam
        ",
    "mem":4096,
    "disk":1,
    "cpus":1,
    "parents":[
      10
    ]
  },
  {
    "id":15,
    "dockerImage":"jnforja/samtools",
    "command":"samtools sort -o ngs4clouddir4/reads.sorted.bam ngs4clouddir4/reads.unsorted.bam
        ",
    "mem":4096,
    "disk":1,
    "cpus":1,
    "parents":[
      11
    ]
  },
  {
    "id":16,
    "dockerImage":"jnforja/samtools",
    "command":"samtools mpileup --fasta-ref
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reference/lambda_virus.fasta
        ngs4clouddir1/reads.sorted.bam --output ngs4clouddir1/reads.pileup ",
    "mem":4096,
    "disk":1,
    "cpus":1,
    "parents":[
      12
    ]
  },
  {
    "id":17,
    "dockerImage":"jnforja/samtools",
    "command":"samtools mpileup --fasta-ref
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reference/lambda_virus.fasta
        ngs4clouddir2/reads.sorted.bam --output ngs4clouddir2/reads.pileup ",
    "mem":4096,
    "disk":1,
    "cpus":1,
    "parents":[
      13
    ]
  },
  {
    "id":18,
    "dockerImage":"jnforja/samtools",
    "command":"samtools mpileup --fasta-ref
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reference/lambda_virus.fasta
        ngs4clouddir3/reads.sorted.bam --output ngs4clouddir3/reads.pileup ",
    "mem":4096,
    "disk":1,
    "cpus":1,
    "parents":[
      14
    ]
  },
  {
    "id":19,
    "dockerImage":"jnforja/samtools",
    "command":"samtools mpileup --fasta-ref
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reference/lambda_virus.fasta
        ngs4clouddir4/reads.sorted.bam --output ngs4clouddir4/reads.pileup ",
    "mem":4096,
    "disk":1,
    "cpus":1,
    "parents":[
      15
    ]
  },
  {
    "id":20,
    "dockerImage":"djordjeklisic/sbg-varscan2:v1",
    "command":"java -jar VarScan.jar mpileup2snp ngs4clouddir1/reads.pileup --min-var-freq 0.90
        --output-vcf 1 >> ngs4clouddir1/var.flt.vcf ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      16
    ]
  },
  {
    "id":21,
    "dockerImage":"djordjeklisic/sbg-varscan2:v1",
    "command":"java -jar VarScan.jar mpileup2snp ngs4clouddir2/reads.pileup --min-var-freq 0.90
        --output-vcf 1 >> ngs4clouddir2/var.flt.vcf ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      17
    ]
  },
  {
    "id":22,
    "dockerImage":"djordjeklisic/sbg-varscan2:v1",
    "command":"java -jar VarScan.jar mpileup2snp ngs4clouddir3/reads.pileup --min-var-freq 0.90
        --output-vcf 1 >> ngs4clouddir3/var.flt.vcf ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      18
    ]
  },
  {
    "id":23,
    "dockerImage":"djordjeklisic/sbg-varscan2:v1",
    "command":"java -jar VarScan.jar mpileup2snp ngs4clouddir4/reads.pileup --min-var-freq 0.90
         --output-vcf 1 >> ngs4clouddir4/var.flt.vcf ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      19
    ]
  },
  {
    "id":24,
    "dockerImage":"vacalexis/split",
    "command":"/Split/stopListing.sh var.flt.vcf
         snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample1/var.flt.vcf
         snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample2/var.flt.vcf
         snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample3/var.flt.vcf
         snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample4/var.flt.vcf ",
    "mem":1024,
    "disk":1,
    "cpus":1,
    "parents":[
      20,
      21,
      22,
      23
    ]
  },
  {
    "id":25,
    "dockerImage":"vacalexis/snp-pipeline",
    "command":"create_snp_list.py --vcfname var.flt.vcf --output
         snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snplist.txt
         snp-pipeline-master/snppipeline/data/lambdaVirusInputs/sampleDirectories.txt ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      1,
      24
    ]
  },
  {
    "id":26,
    "dockerImage":"vacalexis/snp-pipeline",
    "command":"call_consensus.py --snpListFile
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snplist.txt --output
        ngs4clouddir1/consensus.fasta --vcfFileName consensus.vcf ngs4clouddir1/reads.pileup ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      16,
      25
    ]
  },
  {
    "id":27,
    "dockerImage":"vacalexis/snp-pipeline",
    "command":"call_consensus.py --snpListFile
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snplist.txt --output
        ngs4clouddir2/consensus.fasta --vcfFileName consensus.vcf ngs4clouddir2/reads.pileup ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      17,
      25
    ]
  },
  {
    "id":28,
    "dockerImage":"vacalexis/snp-pipeline",
    "command":"call_consensus.py --snpListFile
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snplist.txt --output
        ngs4clouddir3/consensus.fasta --vcfFileName consensus.vcf ngs4clouddir3/reads.pileup ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      18,
      25
    ]
  },
  {
    "id":29,
    "dockerImage":"vacalexis/snp-pipeline",
    "command":"call_consensus.py --snpListFile
        snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snplist.txt --output
        ngs4clouddir4/consensus.fasta --vcfFileName consensus.vcf ngs4clouddir4/reads.pileup ",
    "mem":2048,
    "disk":1,
    "cpus":1,
    "parents":[
      19,
      25
    ]
  },
  {
    "id":30,
      "dockerImage":"vacalexis/split",
      "command":"/Split/stopListing.sh consensus.fasta
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample1/consensus.fasta
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample2/consensus.fasta
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample3/consensus.fasta
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample4/consensus.fasta
           ",
      "mem":1024,
      "disk":1,
      "cpus":1,
      "parents":[
         26,
         27,
         28,
         29
      ]
   },
   {
      "id":31,
      "dockerImage":"vacalexis/snp-pipeline",
      "command":"create_snp_matrix.py
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/sampleDirectories.txt
           --consFileName consensus.fasta --output
           snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snpma.fasta ",
      "mem":2048,
      "disk":1,
      "cpus":1,
      "parents":[
         1,
         30
      ]
   }
],
"outputs":[
   "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/sampleDirectories.txt",
   "reference",
   "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads1.sam",
   "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads2.sam",
   "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads3.sam",
   "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reads4.sam",
   "ngs4clouddir1/reads.sam",
   "ngs4clouddir2/reads.sam",
   "ngs4clouddir3/reads.sam",
   "ngs4clouddir4/reads.sam",
   "ngs4clouddir1/reads.unsorted.bam",
   "ngs4clouddir2/reads.unsorted.bam",
   "ngs4clouddir3/reads.unsorted.bam",
   "ngs4clouddir4/reads.unsorted.bam",
   "ngs4clouddir1/reads.sorted.bam",
   "ngs4clouddir2/reads.sorted.bam",
   "ngs4clouddir3/reads.sorted.bam",
   "ngs4clouddir4/reads.sorted.bam",
   "ngs4clouddir1/reads.pileup",
   "ngs4clouddir2/reads.pileup",
   "ngs4clouddir3/reads.pileup",
    "ngs4clouddir4/reads.pileup",
    "ngs4clouddir1/var.flt.vcf",
    "ngs4clouddir2/var.flt.vcf",
    "ngs4clouddir3/var.flt.vcf",
    "ngs4clouddir4/var.flt.vcf",
    "var.flt.vcf",
    "var.flt.vcf",
    "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snplist.txt",
    "ngs4clouddir1/consensus.fasta",
    "ngs4clouddir2/consensus.fasta",
    "ngs4clouddir3/consensus.fasta",
    "ngs4clouddir4/consensus.fasta",
    "consensus.fasta",
    "consensus.fasta",
    "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/snpma.fasta"
  ],
  "directories":[
    "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/reference",
    "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample1",
    "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample2",
    "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample3",
    "snp-pipeline-master/snppipeline/data/lambdaVirusInputs/samples/sample4",
    "snp-pipeline-master/snppipeline/data/lambdaVirusInputs",
    "ngs4clouddir1",
    "ngs4clouddir2",
    "ngs4clouddir3",
    "ngs4clouddir4"
  ]
}
```

H NGS Pipes Pipeline Description

```
Pipeline "Github" "https://github.com/Vacalexis/tools" {
  tool "Trimmomatic" "DockerConfig" {
    command "trimmomatic" {
      argument "mode" "SE"
      argument "quality" "-phred33"
      argument "inputFile" "ERR406040.fastq"
      argument "outputFile" "ERR406040.filtered.fastq"
      argument "fastaWithAdaptersEtc" "adapters/TruSeq3-SE.fa"
      argument "seed mismatches" "2"
      argument "palindrome clip threshold" "30"
      argument "simple clip threshold" "10"
      argument "windowSize" "4"
      argument "requiredQuality" "15"
      argument "leading quality" "3"
      argument "trailing quality" "3"
      argument "minlen length" "36"
    }
 }
  tool "Velvet" "DockerConfig" {
    command "velveth" {
      argument "output_directory" "velvetdir"
      argument "hash_length" "21"
      argument "file_format" "-fastq"
      chain "filename" "outputFile"
    }
    command "velvetg" {
      argument "output_directory" "velvetdir"
      argument "-cov_cutoff" "5"
    }
  }
  tool "Blast" "DockerConfig" {
    command "makeblastdb" {
      argument "-dbtype" "prot"
      argument "-out" "allrefs"
      argument "-title" "allrefs"
      argument "-in" "allrefs.fna.pro"
    }
    command "blastx" {
      chain "-db" "-out"
      chain "-query" "Velvet" "velvetg" "contigs_fa"
      argument "-out" "blast"
    }
  }
}
```

I NGS Pipes Pipeline Intermediate Representation

```
{
   "inputs":[
      {
```

```
      "src":"file:///C:/Users/Jo%C3%A3o%20Forja/Documents/Isel/6%20Semestre/PFC/inputs/",
      "filesDests":[
        {
          "file":"allrefs.fna.pro",
          "dest":"allrefs.fna.pro"
        },
        {
          "file":"ERR406040.fastq",
          "dest":"ERR406040.fastq"
        }
      ]
    }
  ],
  "tasks":[
    {
      "id":1,
      "dockerImage":"ngspipes/trimmomatic0.33",
      "command":"java -jar /trimmomatic-0.33.jar SE -phred33 ERR406040.fastq
          ERR406040.filtered.fastq ILLUMINACLIP:adapters/TruSeq3-SE.fa:2:30:10
          SLIDINGWINDOW:4:15 LEADING:3 TRAILING:3 MINLEN:36 ",
      "mem":1024,
      "disk":1,
      "cpus":1,
      "parents":[

      ]
    },
    {
      "id":2,
      "dockerImage":"ngspipes/velvet0.7",
      "command":"velveth velvetdir 21 -fastq ERR406040.filtered.fastq ",
      "mem":12288,
      "disk":1,
      "cpus":1,
      "parents":[
        1
      ]
    },
    {
      "id":3,
      "dockerImage":"ngspipes/velvet0.7",
      "command":"velvetg velvetdir -cov_cutoff 5 ",
      "mem":12288,
      "disk":1,
      "cpus":1,
      "parents":[
        2
      ]
    },
    {
      "id":4,
      "dockerImage":"simonalpha/ncbi-blast-docker",
      "command":"makeblastdb -dbtype=prot -out=allrefs -title=allrefs -in=allrefs.fna.pro ",
      "mem":2048,
      "disk":1,
      "cpus":1,
      "parents":[

      ]
    },
    {
      "id":5,
      "dockerImage":"simonalpha/ncbi-blast-docker",
      "command":"blastx -out=blast.out -db=allrefs -query=velvetdir/contigs.fa ",
      "mem":2048,
      "disk":1,
      "cpus":1,
      "parents":[
        3,
        4
      ]
    }
  ],
  "outputs":[
    "/velvetdir/contigs.fa"
  ],
  "directories":[
    "adapters",
    "velvetdir"
  ]
}
```