Session 1.3 – Scalability and Portability, Nextflow and Singularity

Miguel Juliá

BU-ISCIII

Unidades Comunes Científico Técnicas – SGSAFI-ISCIII

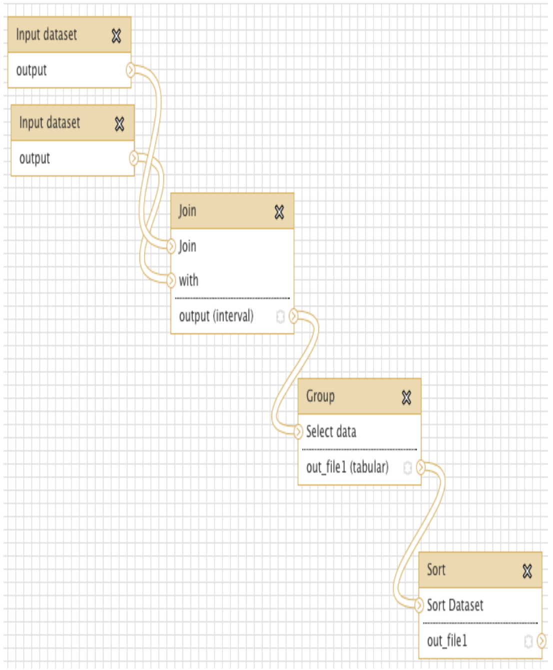
05-09 Noviembre 2018, 1ª Edición
Programa Formación Continua, ISCIII

Index

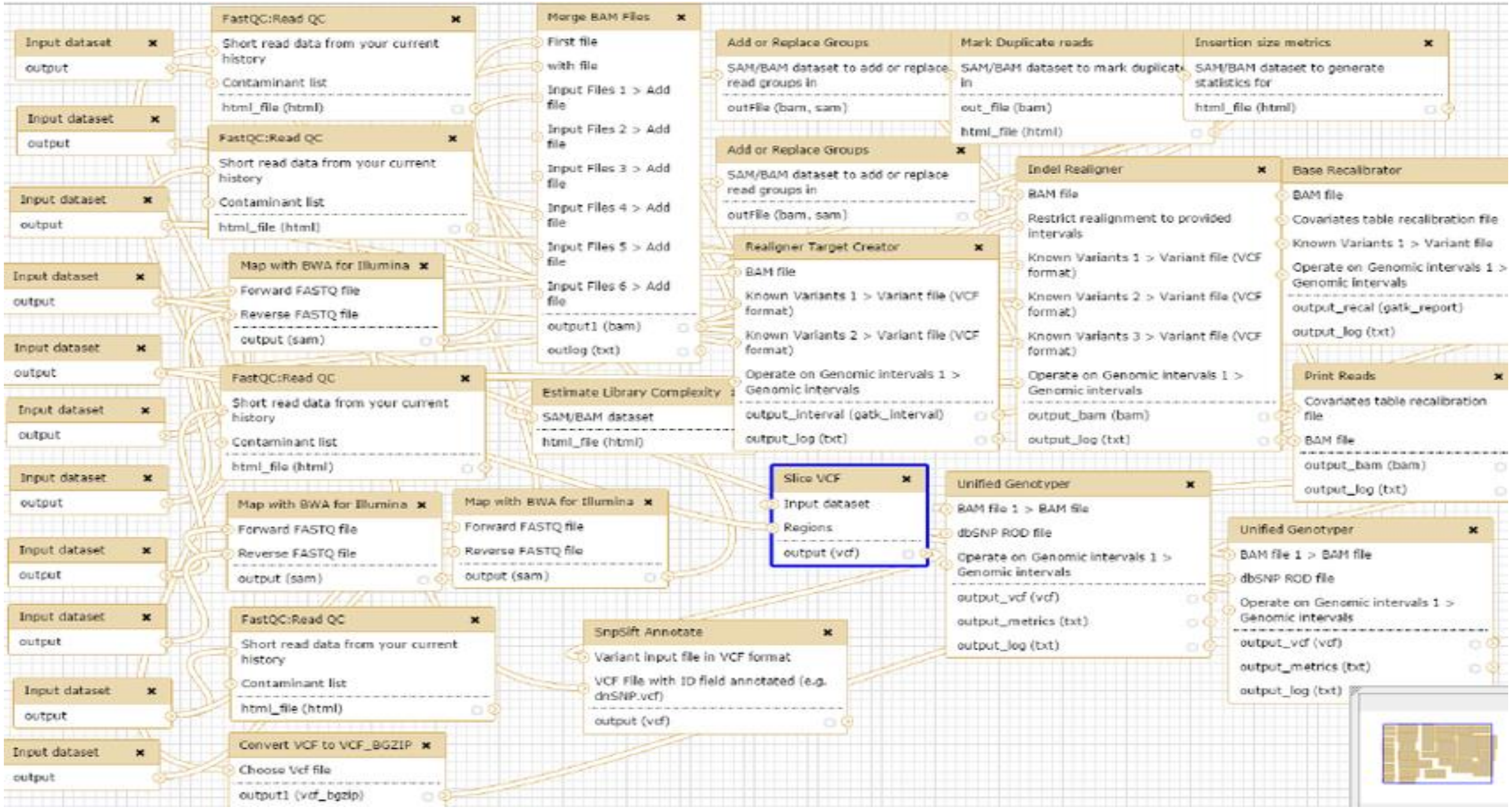
Scalability and Portability, Nextflow and Singularity:

Workflows I

- Bioinformatic analyses invariably involve shepherding files through a series of transformations, called a **pipeline** or a **workflow.**
- These transformations are done by executable **command line software** written for Unix-compatible operating systems.
- They need to be **reproducible, easy to maintain, portable and scalable.**



Workflows II



Input dataset
output
FastQC:Read QC
Short read data from your current history
Contaminant list
html_file (html)
Merge BAM Files
First file
with file
Input Files 1 > Add file
Input Files 2 > Add file
Input Files 3 > Add file
Input Files 4 > Add file
Input Files 5 > Add file
Input Files 6 > Add file
output1 (bam)
outlog (txt)
Add or Replace Groups
SAM/BAM dataset to add or replace read groups in
outFile (bam, sam)
Mark Duplicate reads
SAM/BAM dataset to mark duplicates in
out_file (bam)
html_file (html)
Insertion size metrics
SAM/BAM dataset to generate statistics for
html_file (html)
Map with BWA for Illumina
Forward FASTQ file
Reverse FASTQ file
output (sam)
Indel Realigner
BAM file
Restrict realignment to provided intervals
Known Variants 1 > Variant file (VCF format)
Known Variants 2 > Variant file (VCF format)
Known Variants 3 > Variant file (VCF format)
Operate on Genomic intervals 1 > Genomic intervals
output_bam (bam)
output_log (txt)
Base Recalibrator
BAM file
Covariates table recalibration file
Known Variants 1 > Variant file
Operate on Genomic intervals 1 > Genomic intervals
output_recal (gatk_report)
output_log (txt)
Realigner Target Creator
BAM file
Known Variants 1 > Variant file (VCF format)
Known Variants 2 > Variant file (VCF format)
Operate on Genomic intervals 1 > Genomic intervals
output_interval (gatk_interval)
output_log (txt)
Estimate Library Complexity
SAM/BAM dataset
html_file (html)
Print Reads
Covariates table recalibration file
BAM file
output_bam (bam)
output_log (txt)
Slice VCF
Input dataset
Regions
output (vcf)
Unified Genotyper
BAM file 1 > BAM file
dbSNP ROD file
Operate on Genomic intervals 1 > Genomic intervals
output_vcf (vcf)
output_metrics (txt)
output_log (txt)
SnpSift Annotate
Variant input file in VCF format
VCF File with ID field annotated (e.g. dnSNP.vcf)
output (vcf)
Convert VCF to VCF_BGZIP
Choose Vcf file
output1 (vcf_bgzip)

HPC infrastructure

Machine	OS	Software	CPU	RAM	Storage
Workstation (x5)	Centos 6.9	/opt(*)	4 cores	32 Gb	4 TB
Bioinfo01 (1 node)		/opt(*)	16 cores	120 Gb	500 Gb
HPC (16 nodes)		/opt(*)	320 cores	8 TB	500 Gb

2 shared data storage disk boxes: 70TB + 250 TB

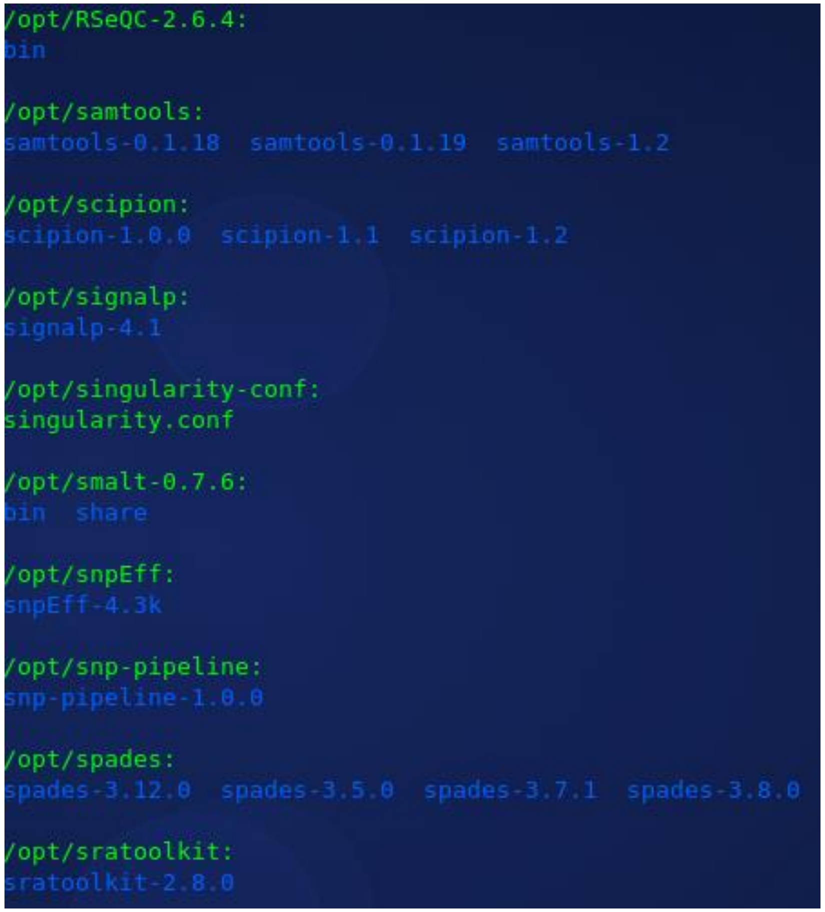
VMs, ISCIII's Windows personal terminals, personal laptops mobile platforms, cloud computing platforms, cloud storage, remote services, …

Environments I

- An **environment** in a combination of hardware and software in a computer.
- For a perfect **reproducibility**, workflows have to be executed in the same environment.
- Porting an environment across infrastructures, and even between clonal machines, can be an **overwhelming task.**
- Some software versions (especially development versions and un-published software) may be **difficult to obtain or eventually taken down.**

Environments II

- Multiple software versions may coexist in the same machine.
- Environmental variables.
- PATHs may change.
- Input data and folder structure.
- Other OS and hardware peculiarities.



```
/opt/RSeQC-2.6.4:
bin

/opt/samtools:
samtools-0.1.18  samtools-0.1.19  samtools-1.2

/opt/scipion:
scipion-1.0.0  scipion-1.1  scipion-1.2

/opt/signalp:
signalp-4.1

/opt/singularity-conf:
singularity.conf

/opt/smalt-0.7.6:
bin  share

/opt/snpEff:
snpEff-4.3k

/opt/snp-pipeline:
snp-pipeline-1.0.0

/opt/spades:
spades-3.12.0  spades-3.5.0  spades-3.7.1  spades-3.8.0

/opt/sratoolkit:
sratoolkit-2.8.0
```

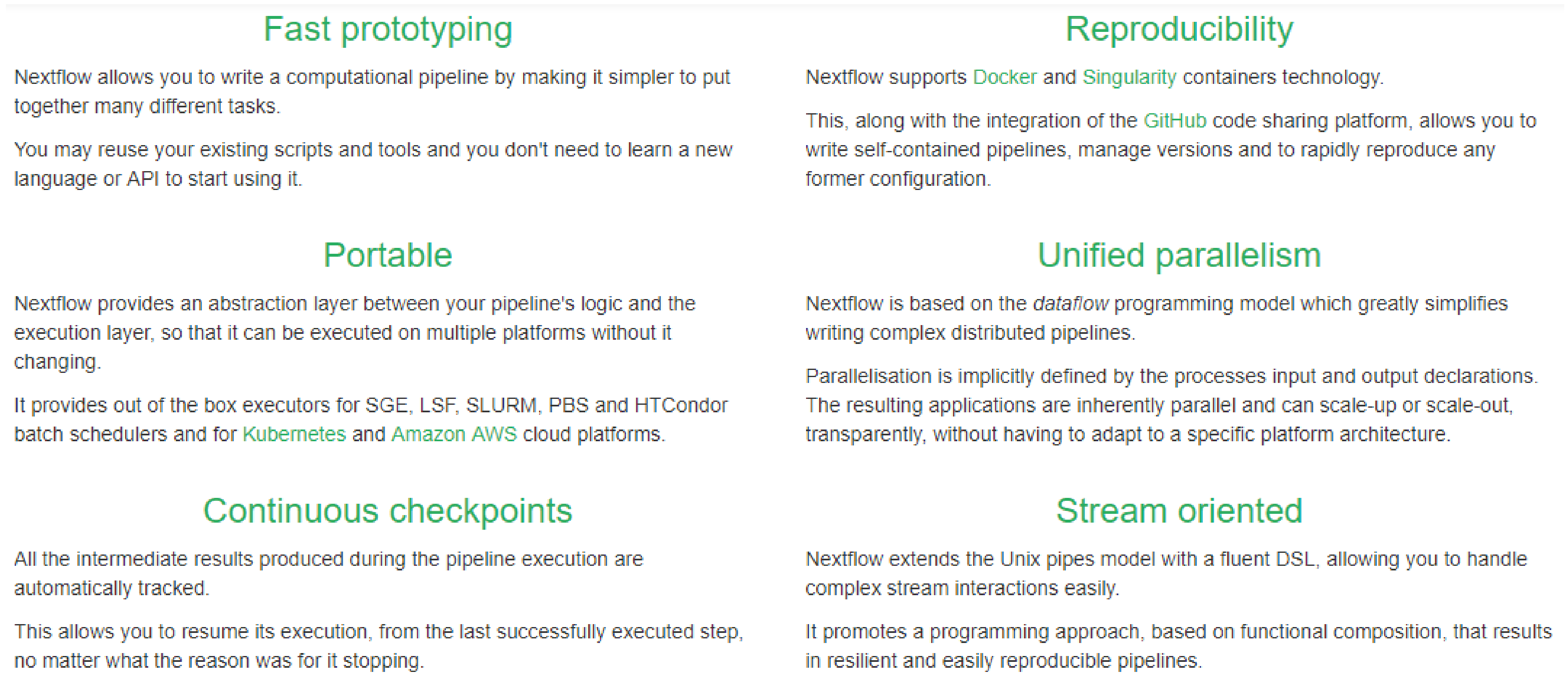
Pipeline frameworks

- Basic scripting for automating scripts is inefficient, difficult to port and maintain, and does not include support for **dependencies** or **reentrancy**.
- High-throughput bioinformatic analyses increasingly rely on **pipeline frameworks** to process sequence and metadata.
- Modern implementations of these frameworks differ on three key dimensions: using an implicit or explicit syntax, using a configuration, convention or class-based design paradigm and offering a command line or workbench interface.
- Choosing the right framework depends on both your lab developers and users needs.

Nextflow I

- **Nextflow** is a DSL for parallel and scalable computational pipelines.
- It enables **scalable and reproducible scientific workflows** using software **containers.**
- It **allows the adaptation of pipelines** written in the most common scripting languages.
- Its fluent **DSL** simplifies the implementation and the deployment of complex parallel and reactive workflows on clouds and clusters.

Nextflow II



Fast prototyping

Nextflow allows you to write a computational pipeline by making it simpler to put together many different tasks.

You may reuse your existing scripts and tools and you don't need to learn a new language or API to start using it.

Reproducibility

Nextflow supports Docker and Singularity containers technology.

This, along with the integration of the GitHub code sharing platform, allows you to write self-contained pipelines, manage versions and to rapidly reproduce any former configuration.

Portable

Nextflow provides an abstraction layer between your pipeline's logic and the execution layer, so that it can be executed on multiple platforms without it changing.

It provides out of the box executors for SGE, LSF, SLURM, PBS and HTCondor batch schedulers and for Kubernetes and Amazon AWS cloud platforms.

Unified parallelism

Nextflow is based on the *dataflow* programming model which greatly simplifies writing complex distributed pipelines.

Parallelisation is implicitly defined by the processes input and output declarations. The resulting applications are inherently parallel and can scale-up or scale-out, transparently, without having to adapt to a specific platform architecture.

Continuous checkpoints

All the intermediate results produced during the pipeline execution are automatically tracked.

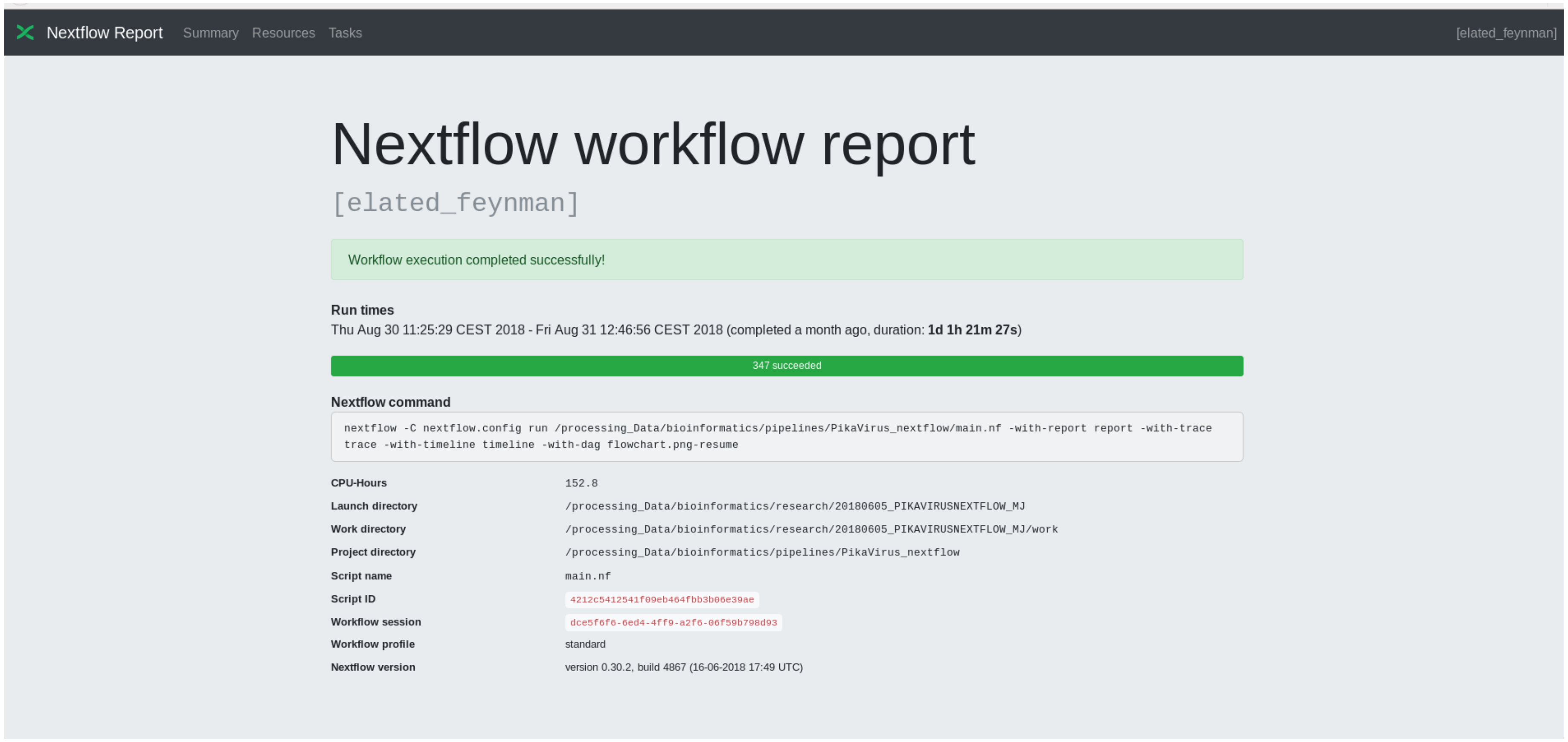
This allows you to resume its execution, from the last successfully executed step, no matter what the reason was for it stopping.

Stream oriented

Nextflow extends the Unix pipes model with a fluent DSL, allowing you to handle complex stream interactions easily.

It promotes a programming approach, based on functional composition, that results in resilient and easily reproducible pipelines.

Nextflow III



Nextflow Report Summary Resources Tasks [elated_feynman]

Nextflow workflow report

[elated_feynman]

Workflow execution completed successfully!

Run times

Thu Aug 30 11:25:29 CEST 2018 - Fri Aug 31 12:46:56 CEST 2018 (completed a month ago, duration: **1d 1h 21m 27s**)

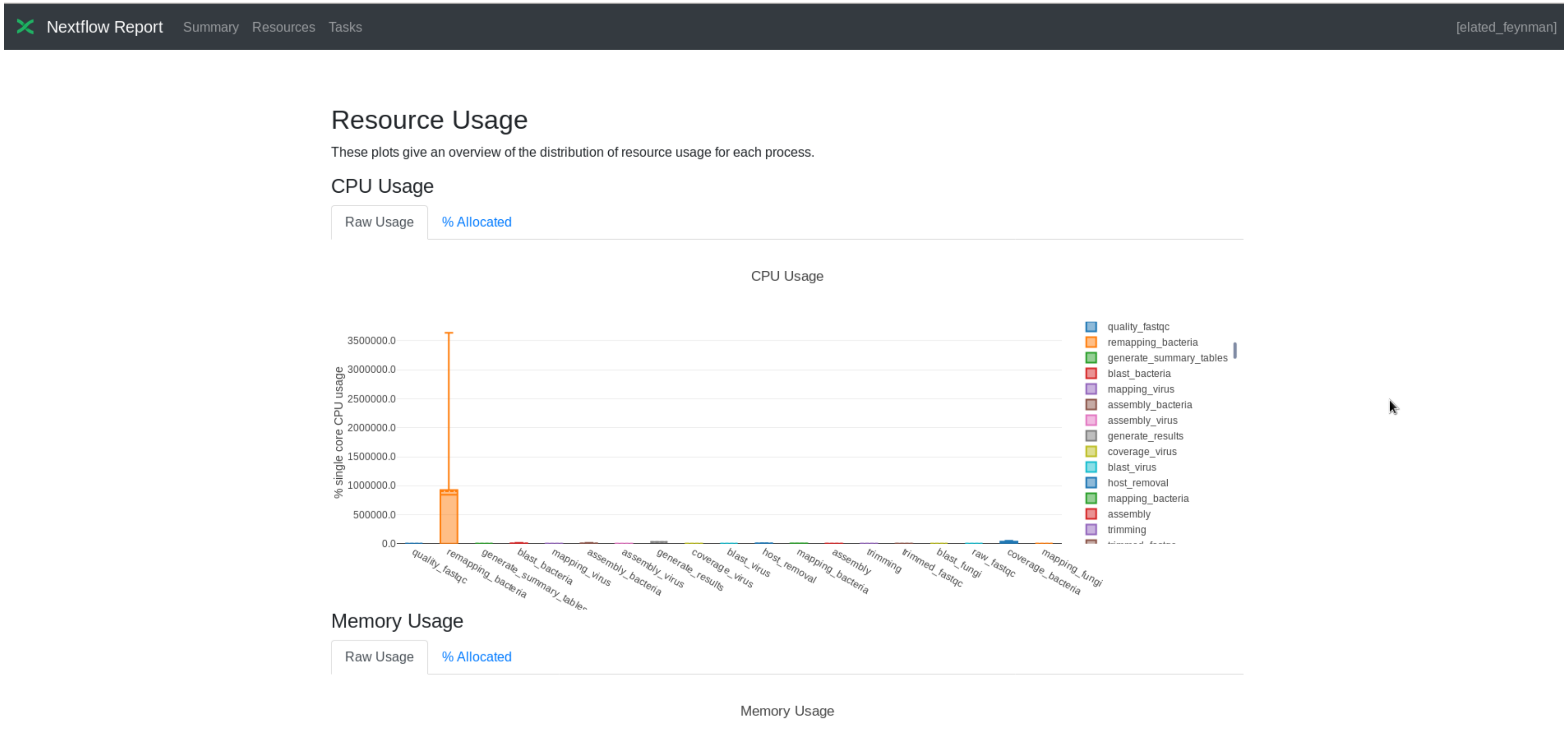
347 succeeded

Nextflow command

```
nextflow -C nextflow.config run /processing_Data/bioinformatics/pipelines/PikaVirus_nextflow/main.nf -with-report report -with-trace
trace -with-timeline timeline -with-dag flowchart.png-resume
```

CPU-Hours	152.8
Launch directory	/processing_Data/bioinformatics/research/20180605_PIKAVIRUSNEXTFLOW_MJ
Work directory	/processing_Data/bioinformatics/research/20180605_PIKAVIRUSNEXTFLOW_MJ/work
Project directory	/processing_Data/bioinformatics/pipelines/PikaVirus_nextflow
Script name	main.nf
Script ID	4212c5412541f09eb464fbb3b06e39ae
Workflow session	dce5f6f6-6ed4-4ff9-a2f6-06f59b798d93
Workflow profile	standard
Nextflow version	version 0.30.2, build 4867 (16-06-2018 17:49 UTC)

Nextflow IV



Nextflow Report   Summary   Resources   Tasks   [elated_feynman]

Resource Usage

These plots give an overview of the distribution of resource usage for each process.

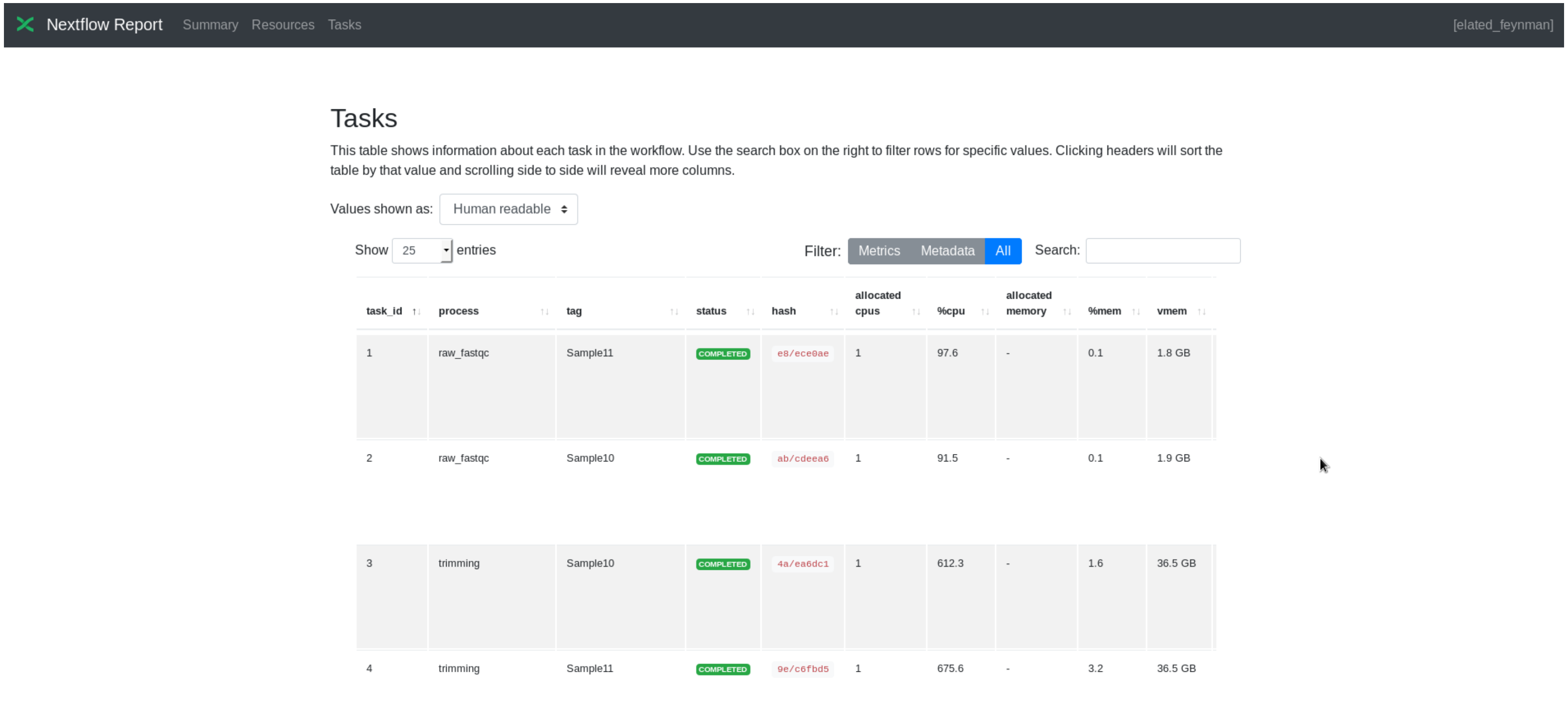
CPU Usage

Raw Usage   % Allocated

Memory Usage

Raw Usage   % Allocated

Nextflow V



Nextflow Report · Summary · Resources · Tasks · [elated_feynman]

Tasks

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

Values shown as: Human readable

Show 25 entries · Filter: Metrics · Metadata · All · Search:

task_id	process	tag	status	hash	allocated cpus	%cpu	allocated memory	%mem	vmem
1	raw_fastqc	Sample11	COMPLETED	e8/ece0ae	1	97.6	-	0.1	1.8 GB
2	raw_fastqc	Sample10	COMPLETED	ab/cdeea6	1	91.5	-	0.1	1.9 GB
3	trimming	Sample10	COMPLETED	4a/ea6dc1	1	612.3	-	1.6	36.5 GB
4	trimming	Sample11	COMPLETED	9e/c6fbd5	1	675.6	-	3.2	36.5 GB

Containers I

Linux containers is a generic term for an implementation of operating system-level virtualization for the Linux operating system.

Containers allow us to **port** pipelines and **replicate** their exact execution environments across different hardware.

Currently, a number of such implementations exist, and they are all based on the **virtualization, isolation, and resource management mechanisms provided by the Linux kernel.**

Containers II

Singularity is a free, cross-platform and open-source computer program that performs operating-system-level virtualization.

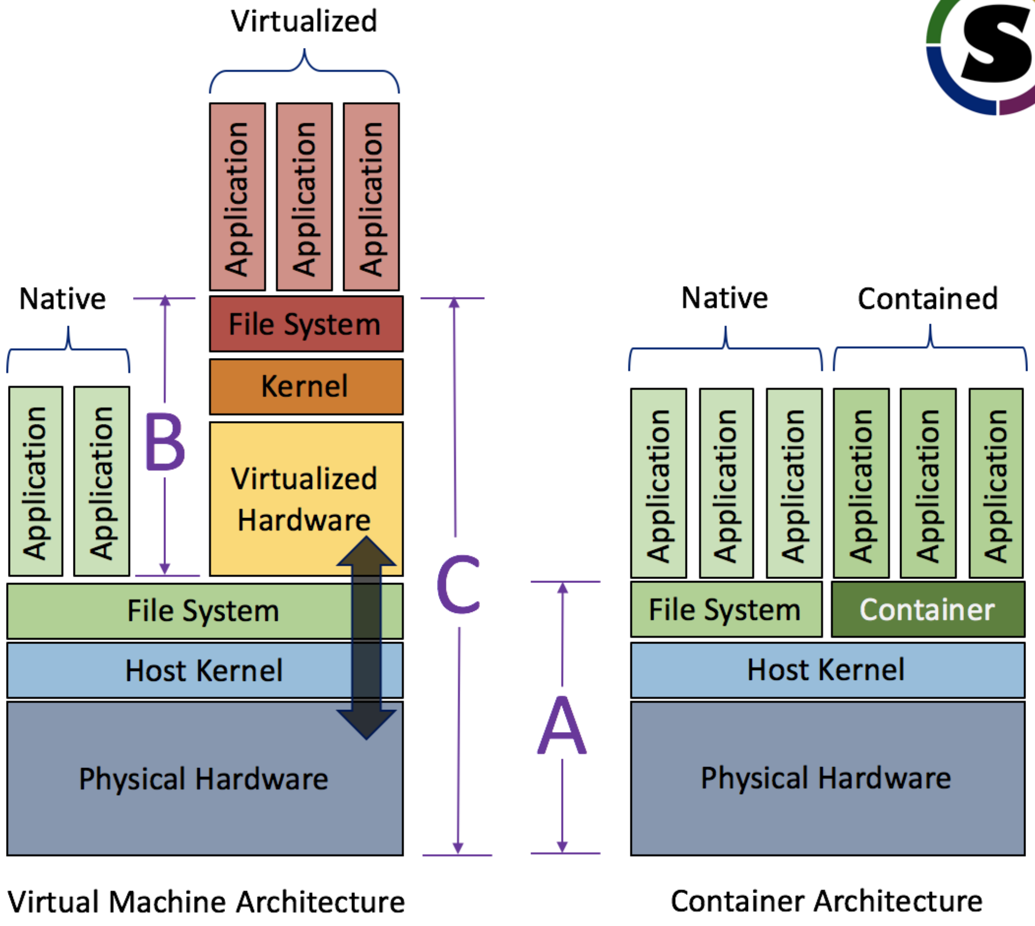
One of the main uses of Singularity is to bring containers and **reproducibility to scientific computing** and the HPC world.

While Docker is broadly used, Singularity is **fully compatible with Docker**, plus Singularity does **not require root permissions** to be executed.

Singularity I

VIRTUAL MACHINES VS. CONTAINERS

- Applications running within a container will always be "closer" to the physical hardware
    - Notice how close to native a container behaves
- Applications running through a virtual machine will always have multiple levels of indirection
- The container's proximity to the physical hardware equates to less overhead, higher performance and lower latency



Singularity II

Singularity image runs on the same level as the OS, directly above the kernel, and can access all hardware in the machine.

Not needing to virtualise the hardware and run a kernel again makes this kind of virtualisation really effective.

Filesystem is shared, and some paths are automatically mounted (/tmp and /home), while the others are optional.

Files of the host system can be created, modified and deleted from the image in the mounted folders.

Scientific File System I

Containers may become black boxes. To avoid this scenario, we use the **Scientific File System (SCIF)**.

This filesystem is **well documented and free**.

SCIF software allows us to create **reproducible images** from publishable scripts (recipes).

SCIF can be **integrated with Singularity** to provide a description of the software and data in the image, and even execute it without manually entering into it.

Scientific File System II

All **installed software** in SCIF is modular and is located in:

/scif/apps/

Despite the software version loaded by default in PATH, others may be available inside the main folder.

Data is stores in

/scif/data/

Results I

Before the implementation of these combination of framework, software and guidelines, executing a workflow in an HPC environment consisted in the following steps:

- Loading input data
- Asking sysadmin to install dependencies
- Load references
- Estimate and book computational resources
- Manually execute each step of the pipeline, or automate it with a script
- Wait with no control over the process status until finished

Results II

Now a simple command works out of the box in any machine:

```
    nextflow run //buisciii/main.nf –profile sigularity
```

Plus, it give us:

- Dependency and computing automatization
- Resource usage statistics
- Easy to share and maintain
- Reproducibility and re-entrancy
- Transparency

Thanks for your attention!

And this is only the tip of the iceberg...
Check this if you wanna know what’s really going under the hood:

https://github.com/BU-ISCIII