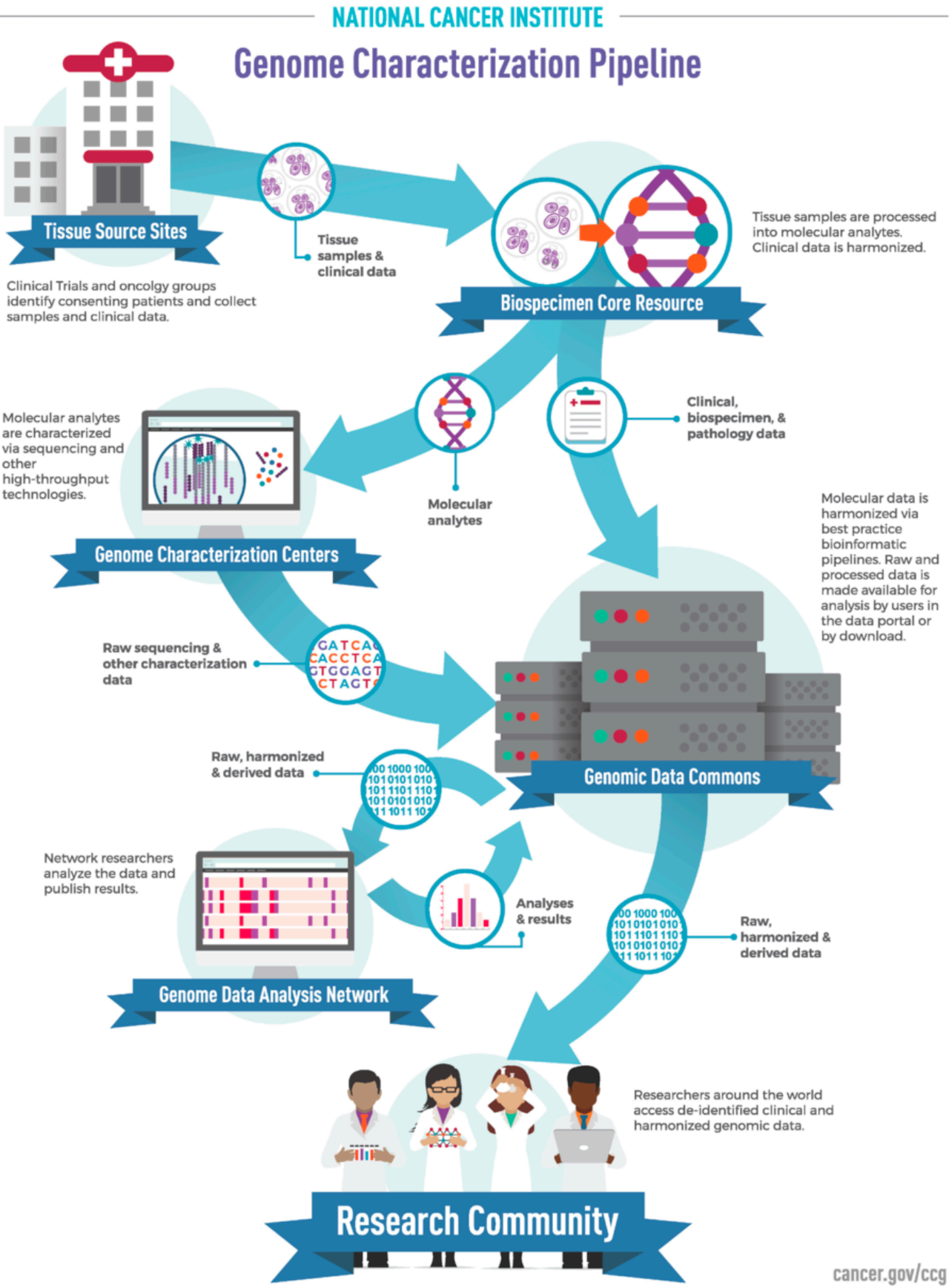
Nextflow: Distribuible escalable, y reproducible computación cientifica.

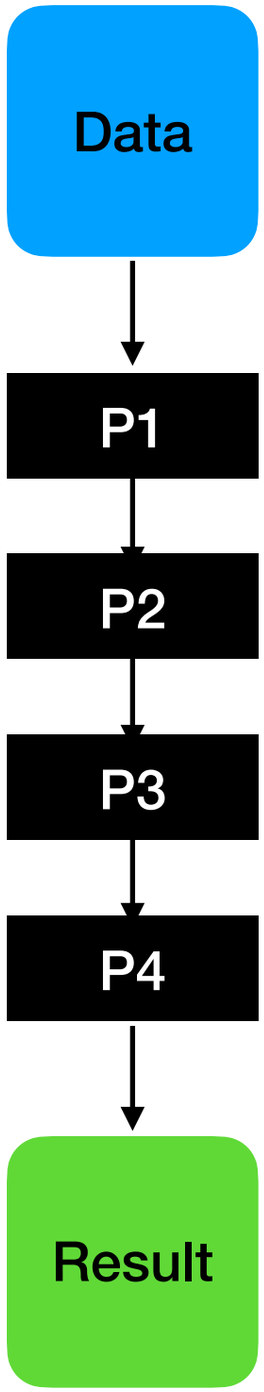
Alex Di Genova

11/10/2024

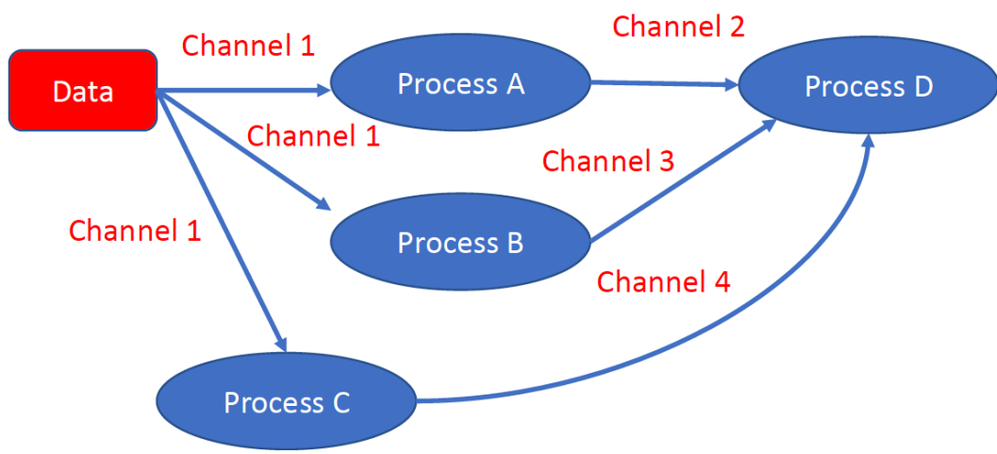


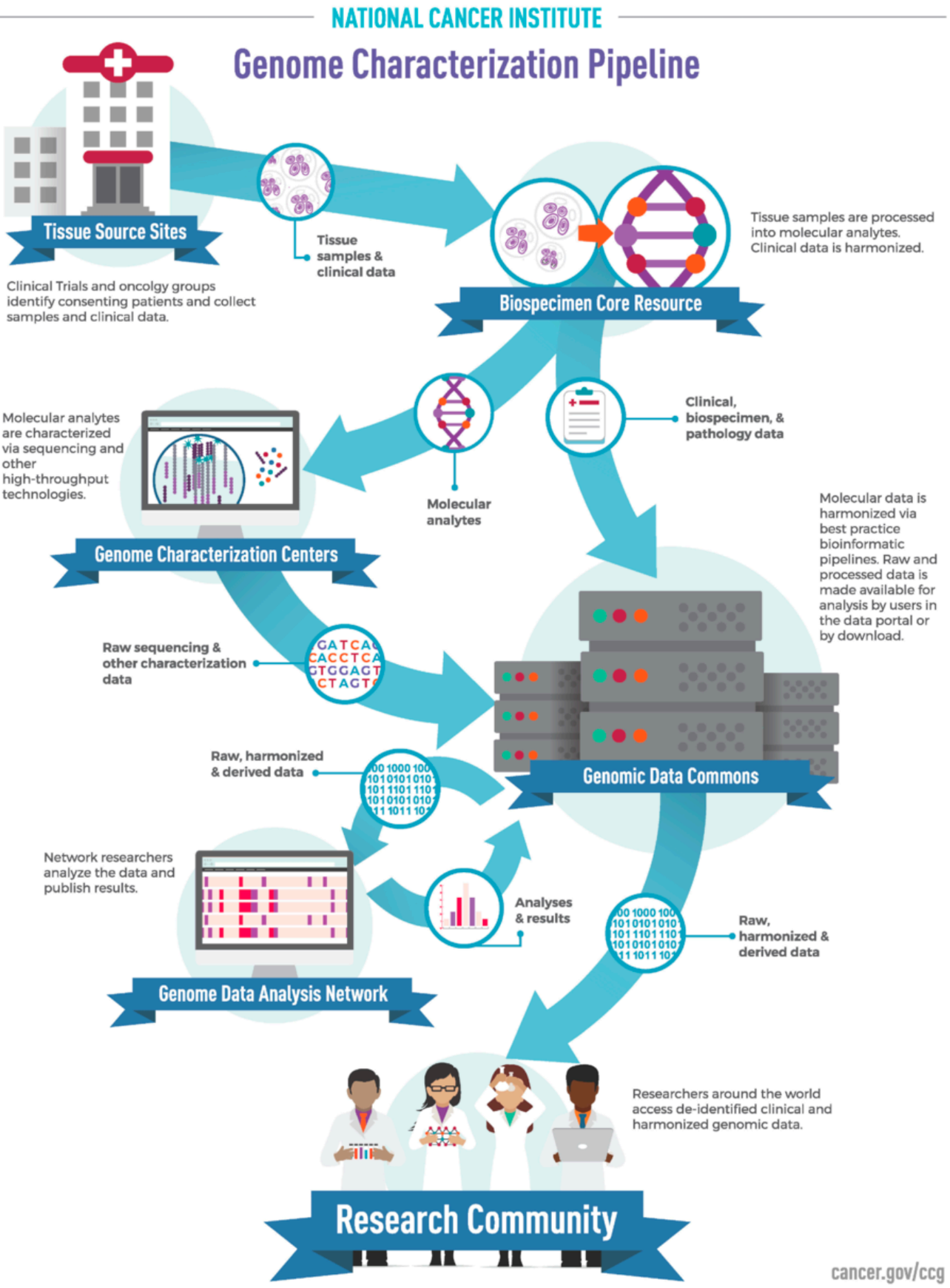
Nextflow

Genómica



Workflow





Nextflow

Genómica

Requisitos de pipelines para proyectos genómicos a gran escala

- **Reproducible**: los resultados genómicos deben ser totalmente reproducibles
- **Escalable**: Facil ejecucion en cluster HPC o cloud.
- **Portátil**: puede ejecutarse en varias infraestructuras (diferente sistema operativo, cloud)
- **Manejar la heterogeneidad**: funciona con dependencias de software en conflicto y varios requisitos de recursos.

Pipelines para datos genómicos

Solución utilizar un lenguaje de dominio especifico (DSL)

Un lenguaje específico de dominio (DSL) es un lenguaje de programación especializado en un dominio de aplicación particular (pipelines).

- **Nextflow**: Basado en Groovy (java); Desarrollado en el Centro de regulacion del genoma (Barcelona, España).

nature biotechnology

Explore content · About the journal · Publish with us

nature › nature biotechnology › correspondence › article

Published: 11 April 2017

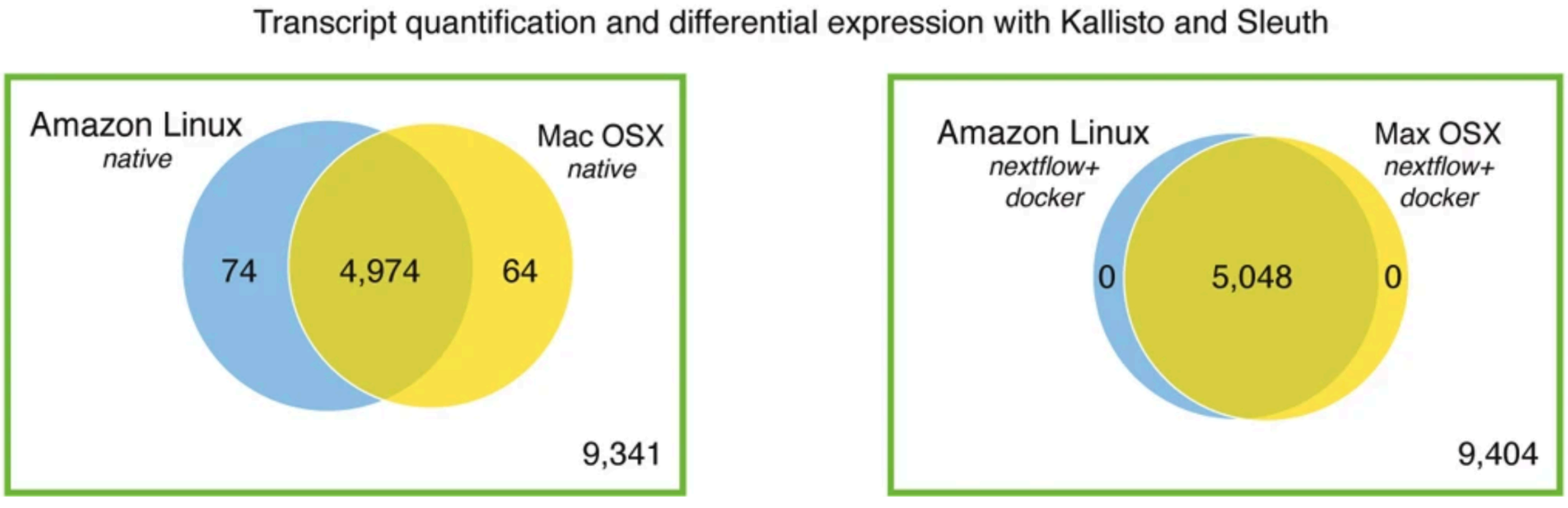
Nextflow enables reproducible computational workflows

Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo & Cedric Notredame

Nature Biotechnology **35**, 316–319 (2017) | Cite this article

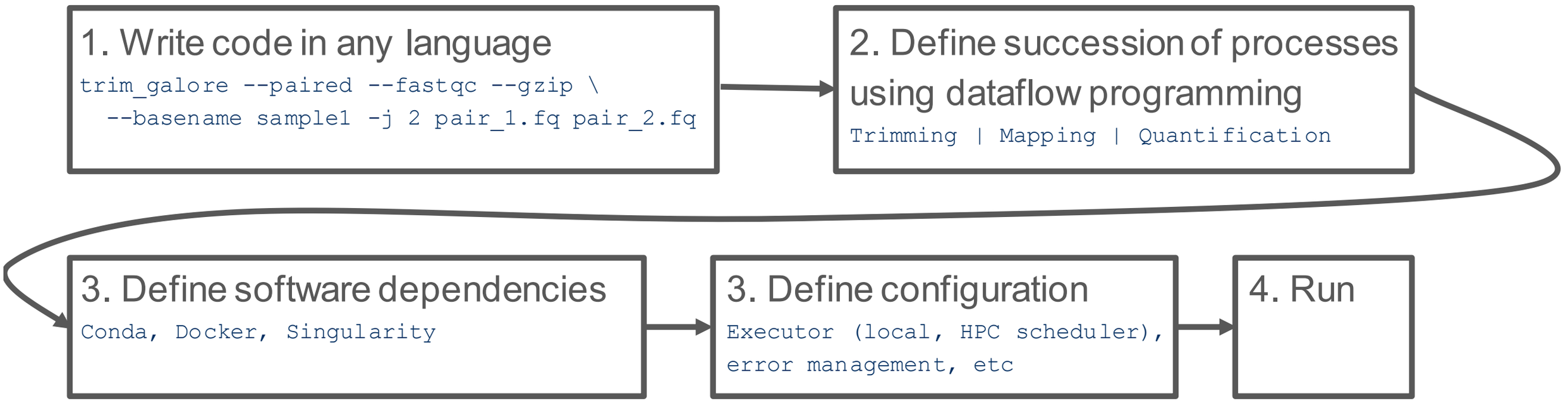
14k Accesses | **590** Citations | **122** Altmetric | Metrics

- **c**



Nextflow

Diseño y caracteristicas

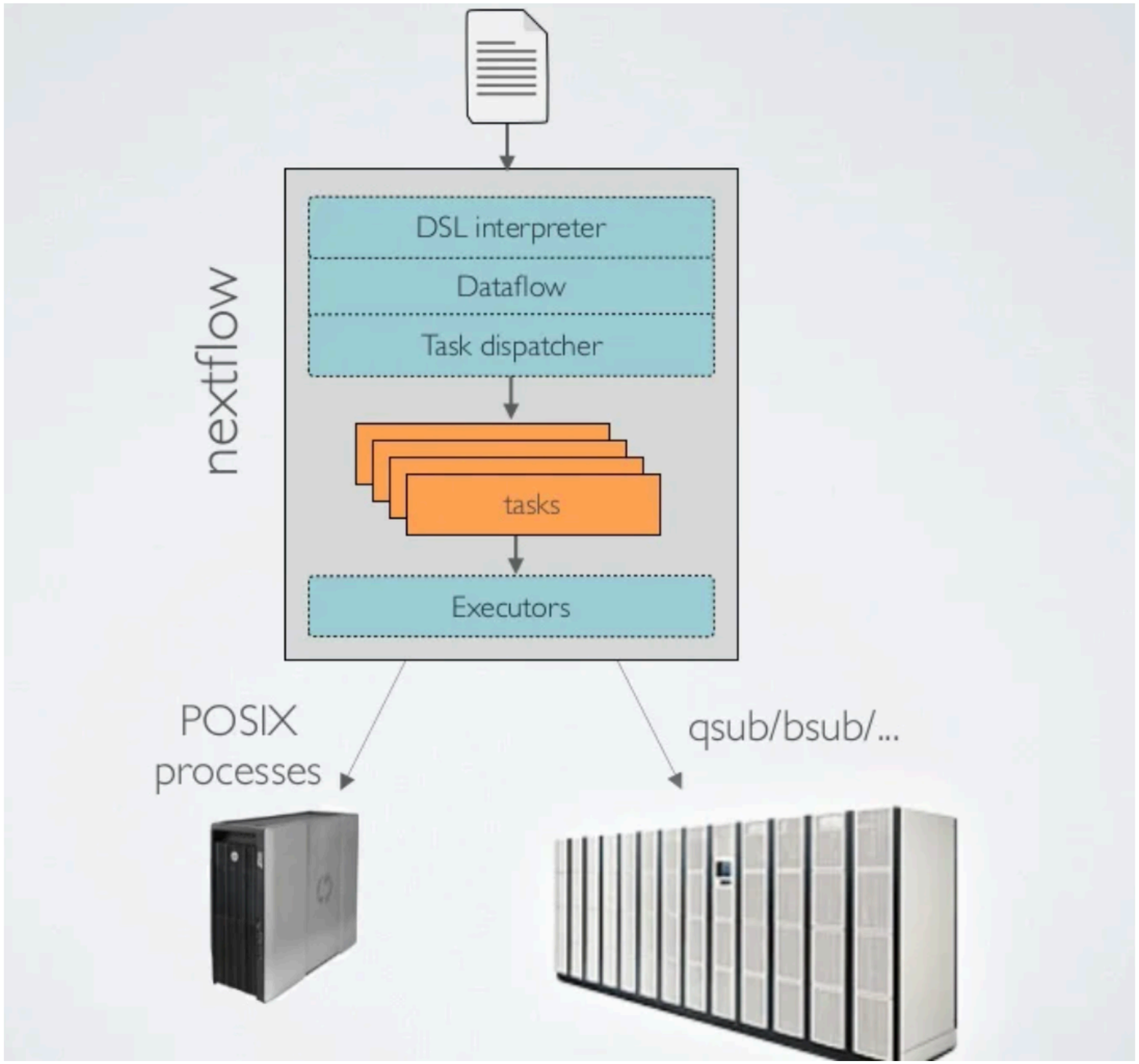


Lenguaje de pipeline con paralelización implícita y programación de tareas:

- El mismo conjunto de tareas se aplica a todos los archivos de entrada
- Las tareas se ejecutan automáticamente en el orden correcto dadas sus entradas y salidas
- Maneja automáticamente el envío de tareas a distintos sistemas administradores de recursos.

Nextflow

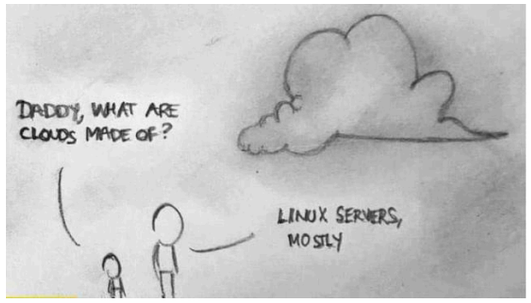
Independiente de la plataforma

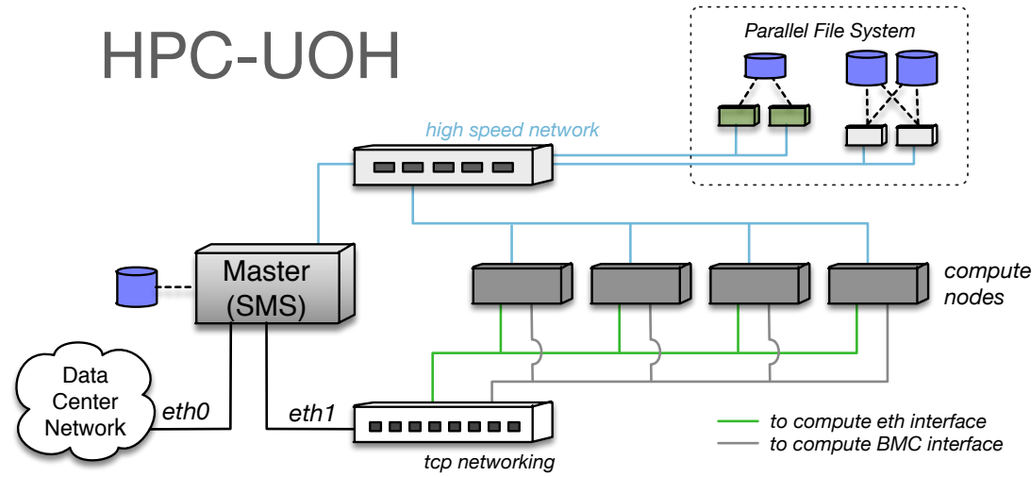


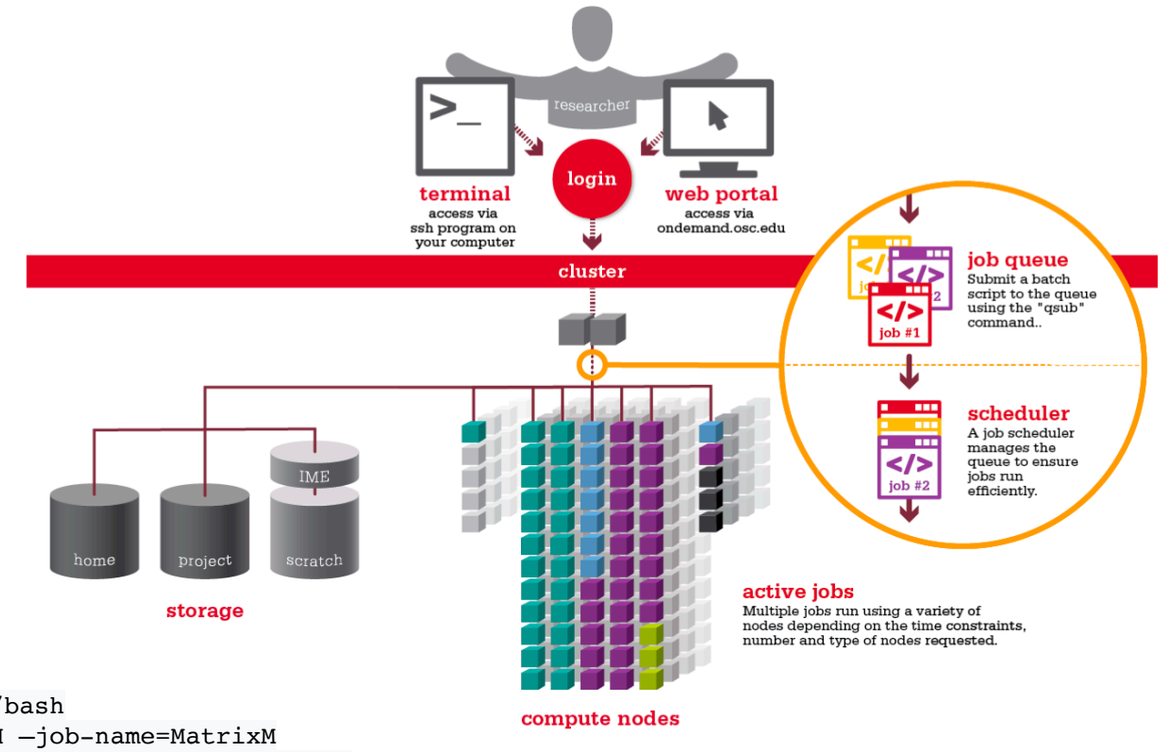
Nextflow

Local, cluster, cloud

- Manejadores de recursos:
  - **Local** (Ejecuta los procesos en la computadora local)
  - **SLURM** (https://slurm.schedmd.com/documentation.html)
  - **LSF**(https://en.wikipedia.org/wiki/IBM_Spectrum_LSF)
  - **Moab** (https://en.wikipedia.org/wiki/Moab_Cluster_Suite)
  - **OAR** (https://oar.imag.fr/)
  - **PBS/Torque** (http://en.wikipedia.org/wiki/Portable_Batch_System)
  - **SGE** (http://gridscheduler.sourceforge.net/)
  - **Clouds**:
    - **AWS Batch** (https://aws.amazon.com/batch/)
    - **Azure Batch** (https://azure.microsoft.com/en-us/services/batch/)
    - **Google Cloud Batch** (https://cloud.google.com/batch)
    - **Google Life Sciences** (https://cloud.google.com/life-sciences)







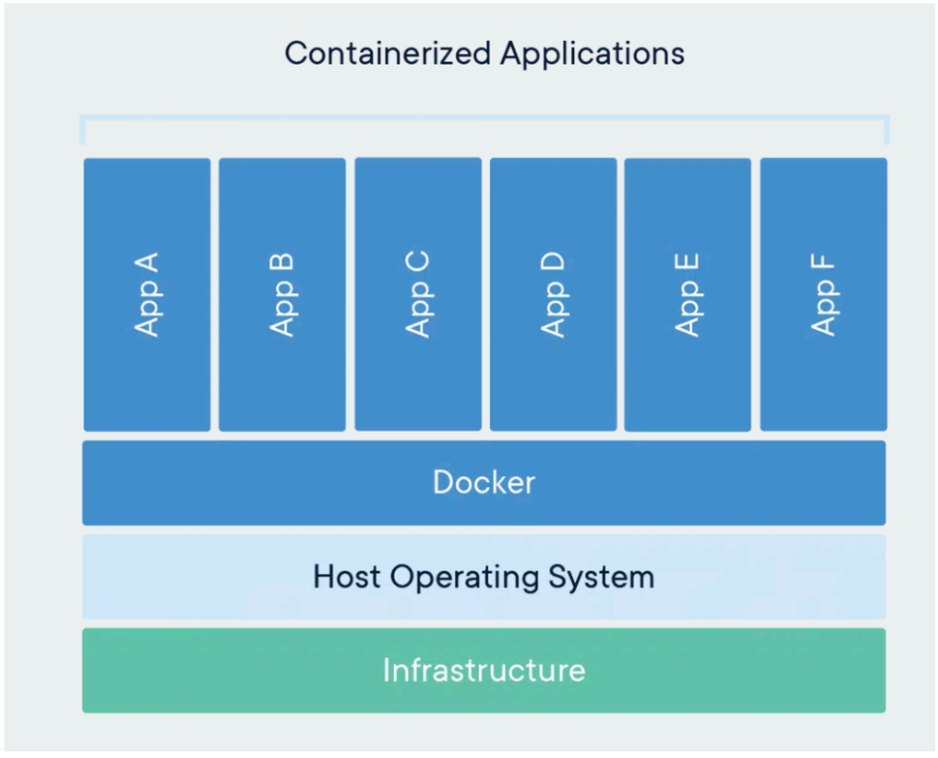

```
#!/bin/bash
#SBATCH –job-name=MatrixM
#SBATCH --output=LAMMPS_%j.out
#SBATCH --mail-type=END,FAIL
#SBATCH –mail-user=alex.digenova@uoh.cl
#SBATCH --nodes=4                # Number of nodes
#SBATCH --ntasks=8               # Number of MPI ranks
#SBATCH --ntasks-per-node=2      # Number of MPI ranks per node
#SBATCH --ntasks-per-socket=1    # Number of tasks per processor socket on the node
#SBATCH --cpus-per-task=8        # Number of OpenMP threads for each MPI process/rank
#SBATCH --mem-per-cpu=2000mb     # Per processor memory request
#SBATCH --time=4-00:00:00        # Walltime in hh:mm:ss or d-hh:mm:ss

## bash code
date;hostname;pwd
module load intel/2018 openmpi/3.1.0
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
srun --mpi=pmi_v1 /path/to/app/matrix_multiplication
date
```

Nexflow

Contenedores (Docker/singularity)

- Un contenedor es una unidad estándar de software que empaqueta el código y todas sus dependencias, por lo que la aplicación se ejecuta de forma rápida y fiable de un entorno informático a otro.

- Una imagen de contenedor de Docker es un paquete de software ligero, independiente y ejecutable que incluye todo lo necesario para ejecutar una aplicación: código, binarios, herramientas del sistema, bibliotecas del sistema y configuración.




```
################ BASE IMAGE ####################
FROM mambaorg/micromamba:0.15.3
##site to test docker configuration files
################ METADATA ######################
LABEL base_image="mambaorg/micromamba"
LABEL version="0.15.3"
LABEL software="k-count-nf"
LABEL software.version="1.1"
LABEL about.summary="Container image containing all requirements for k-count-nf"
LABEL about.home="https://github.com/digenoma-lab/k-count-nf"
LABEL about.documentation="https://github.com/digenoma-lab/k-count-nf/README.md"
LABEL about.license_file="https://github.com/digenoma-lab/k-count-nf/LICENSE.txt"
LABEL about.license="MIT"

################ MAINTAINER ####################
MAINTAINER Alex Di Genova <digenova@gmail.com>
################ INSTALLATION ####################
USER root
#the next command install the ps command needed by nexflow to collect run metrics
RUN apt-get update && apt-get install -y procps
USER micromamba
COPY --chown=micromamba:micromamba environment.yml /tmp/environment.yml
RUN micromamba create -y -n k-count -f /tmp/environment.yml && \
    micromamba clean --all --yes
ENV PATH /opt/conda/envs/k-count/bin:$PATH
```

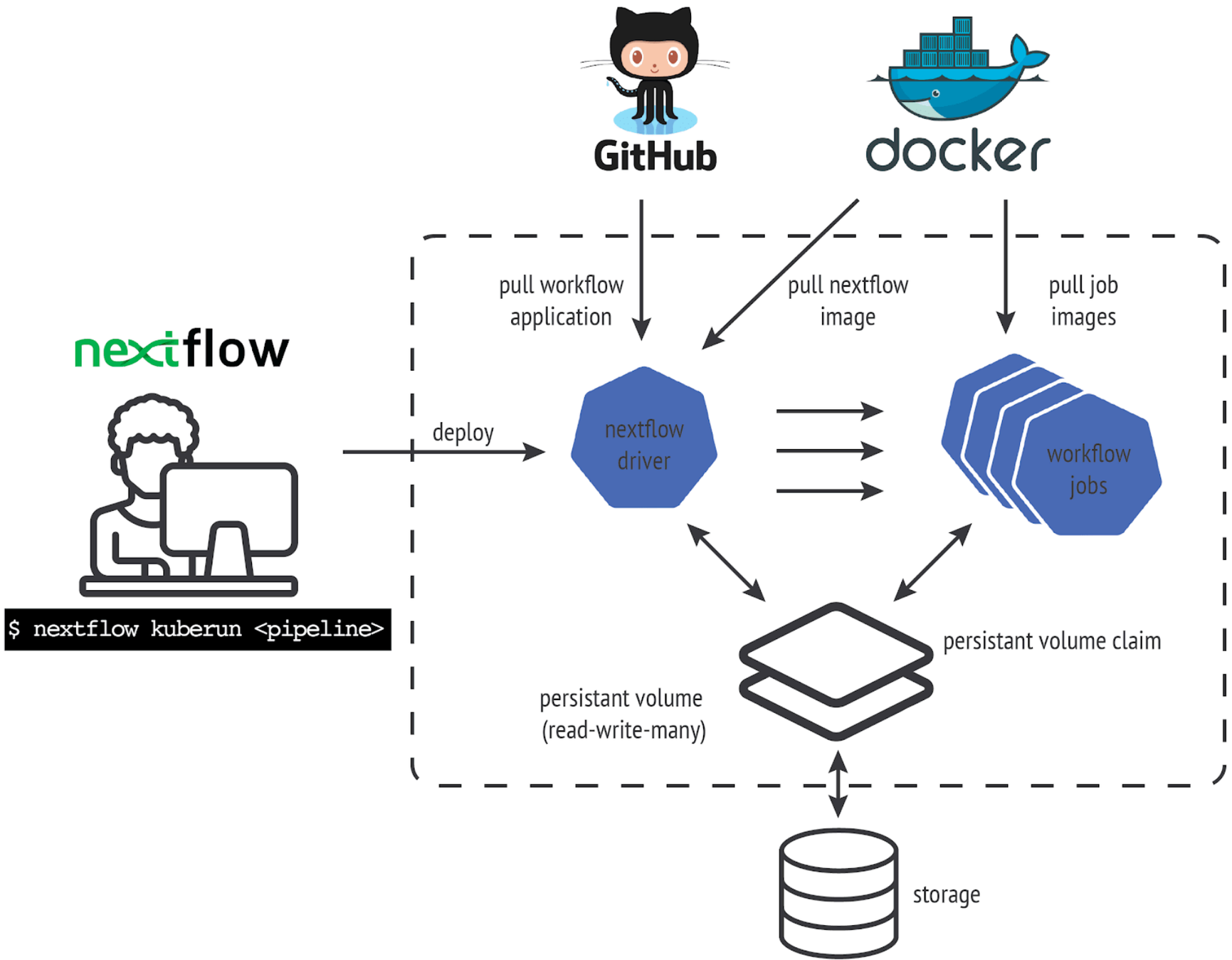
```
name: k-count

channels:

  - conda-forge

  - bioconda

  - defaults

dependencies:

  - kmc=3.1.1rc1

  - genomescope2=2.0
```

https://hub.docker.com/
https://biocontainers.pro/


```
singularity pull biocontainers/fastqc:v0.11.9_cv8
 singularity shell fastqc_v0.11.9_cv8.sif
```

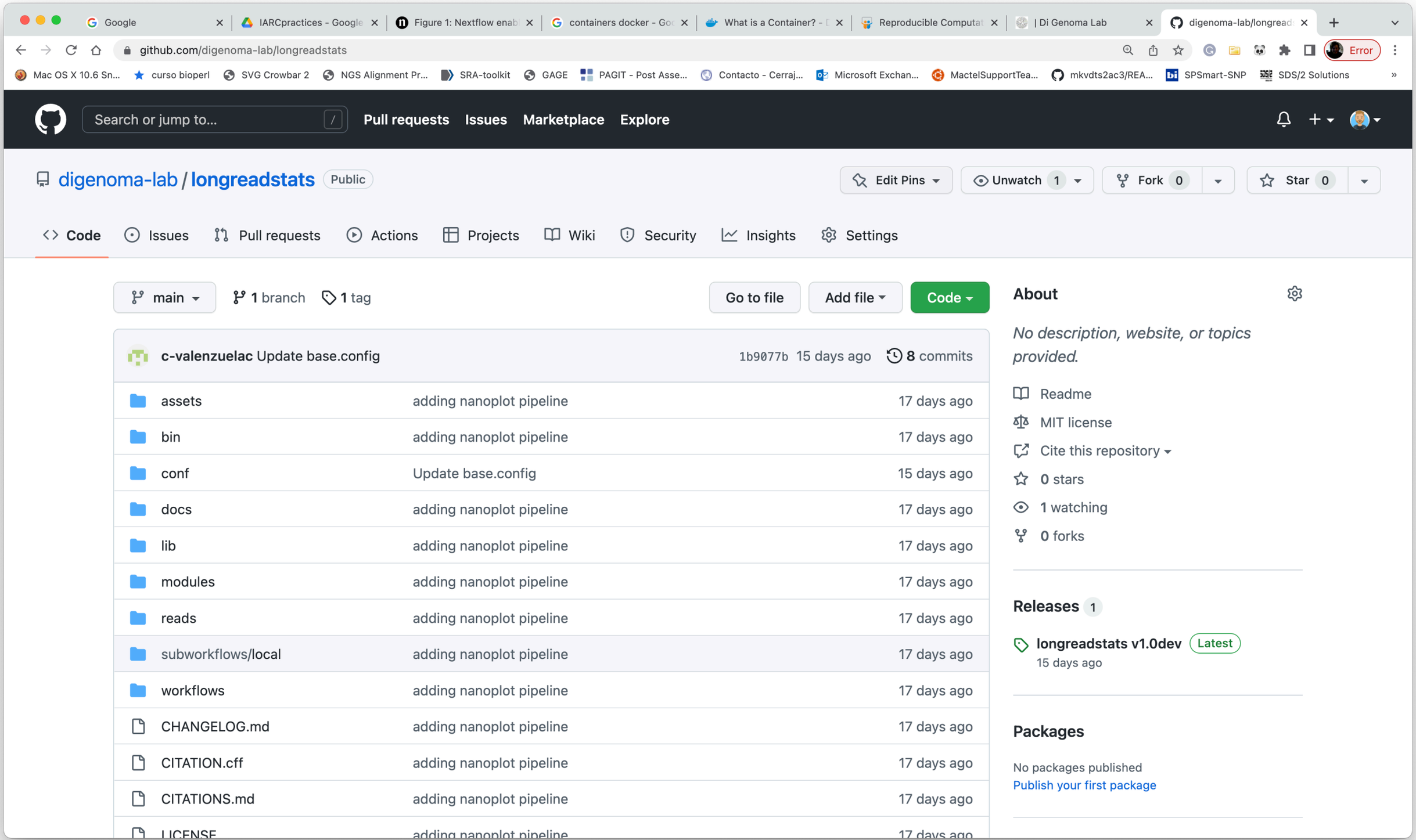
Nextflow

Tecnología



GitHub

Nextflow pipeline



https://github.com/digenoma-lab/longreadstats

Nextflow

Corriendo un pipeline

nextflow run digenoma-lab/longreadstats -profile <singularity> --input samplesheet.csv

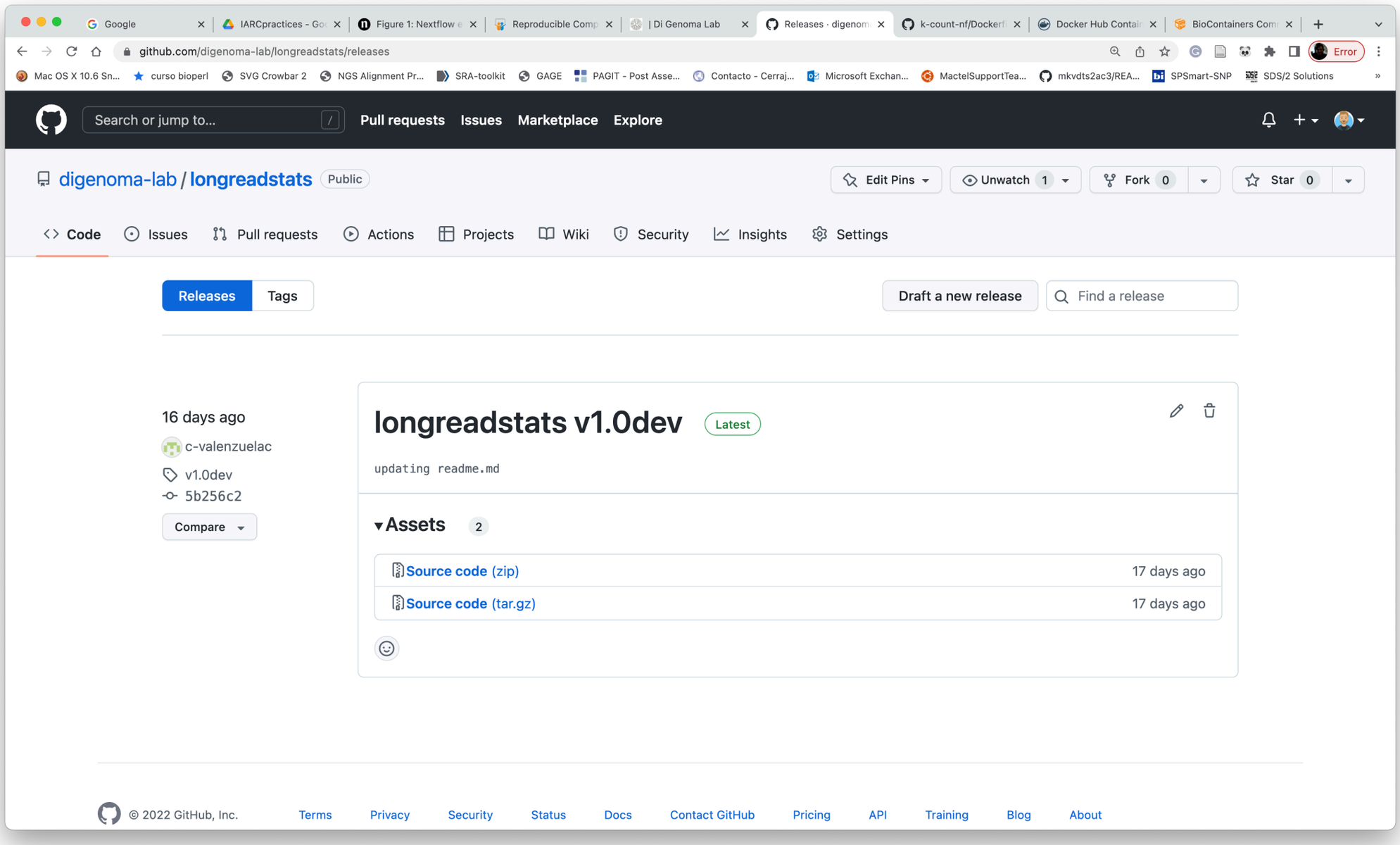
Nombre del pipeline: ruta a un script de *nextflow script*, un ruta a un directorio *con un archivo de configuracion de nextflow que indica un script nextflow*, o un repositorio en *github* (ejemplo, https://github.com/digenoma-lab/longreadstats)

Opciones de nextflow (“-”): por ejemplo, perfiles definiendo el uso de contenerdores o el software local.

```
sample,fastq_1,fastq_2
A01,/home/adigenova/digenoma-lab-longreadstats/reads/tmp1.fastq.gz,
A02,/home/adigenova/digenoma-lab-longreadstats/reads/tmp2.fastq.gz,
A03,/home/adigenova/digenoma-lab-longreadstats/reads/tmp3.fastq.gz,
```

Github releases

nextflow run digenoma-lab/longreadstats **-r v1.0dev** —profile <singularity> --input samplesheet.csv



Nextflow

Convirtiendo la pesadilla de las dependencias en…

Para correr un pipeline usualmente debemos instalar muchos programas:

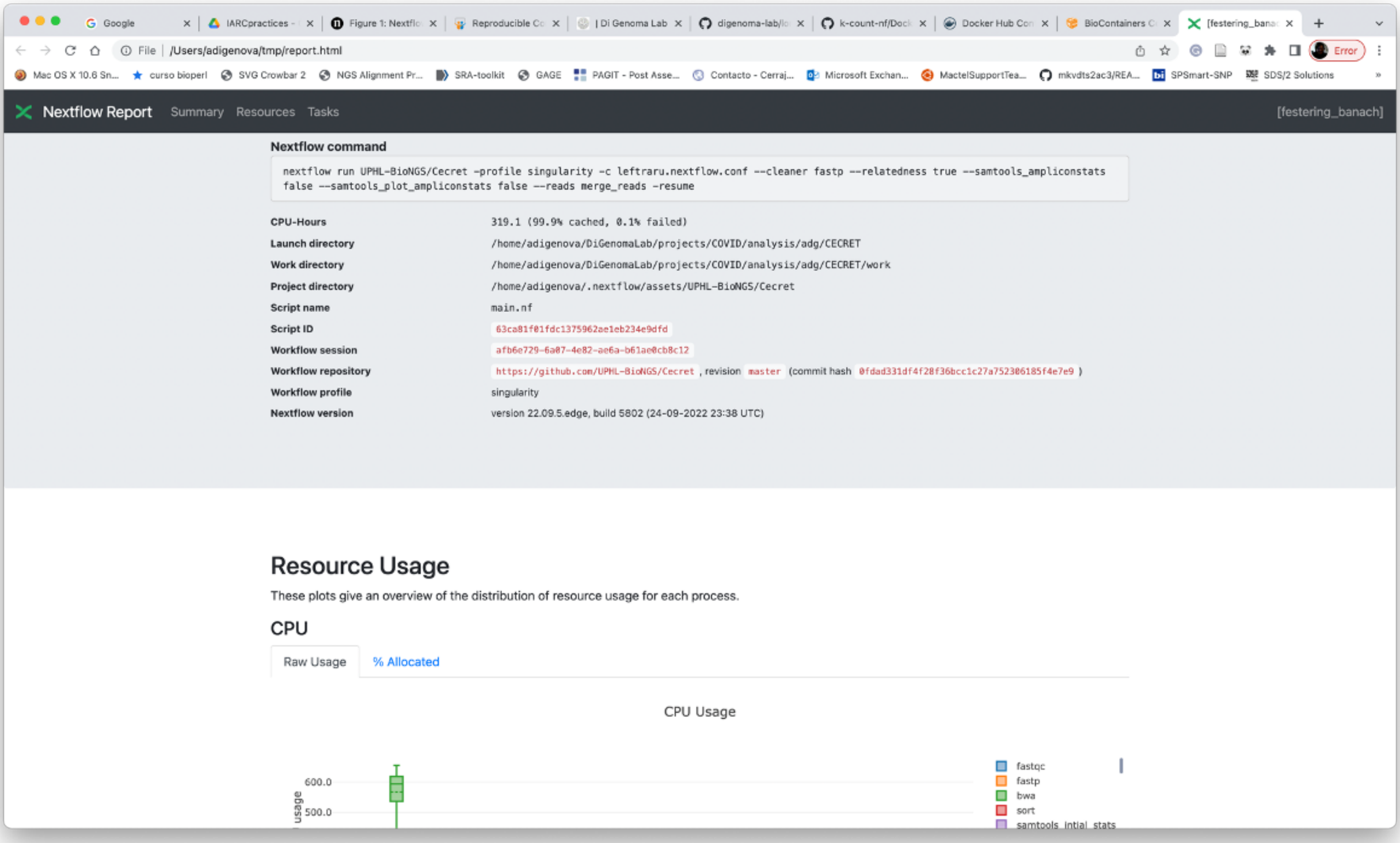
gatk4, fastqc, trim-galore, samtools, star, rseqc, ldc, sambamba, samblaster, multiqc, htseq, R with ggplot2, gsalib, reshape package….

….. Un gran sueño.

nextflow run digenoma-lab/longreadstats **-r v1.0dev** —profile <singularity> --input samplesheet.csv

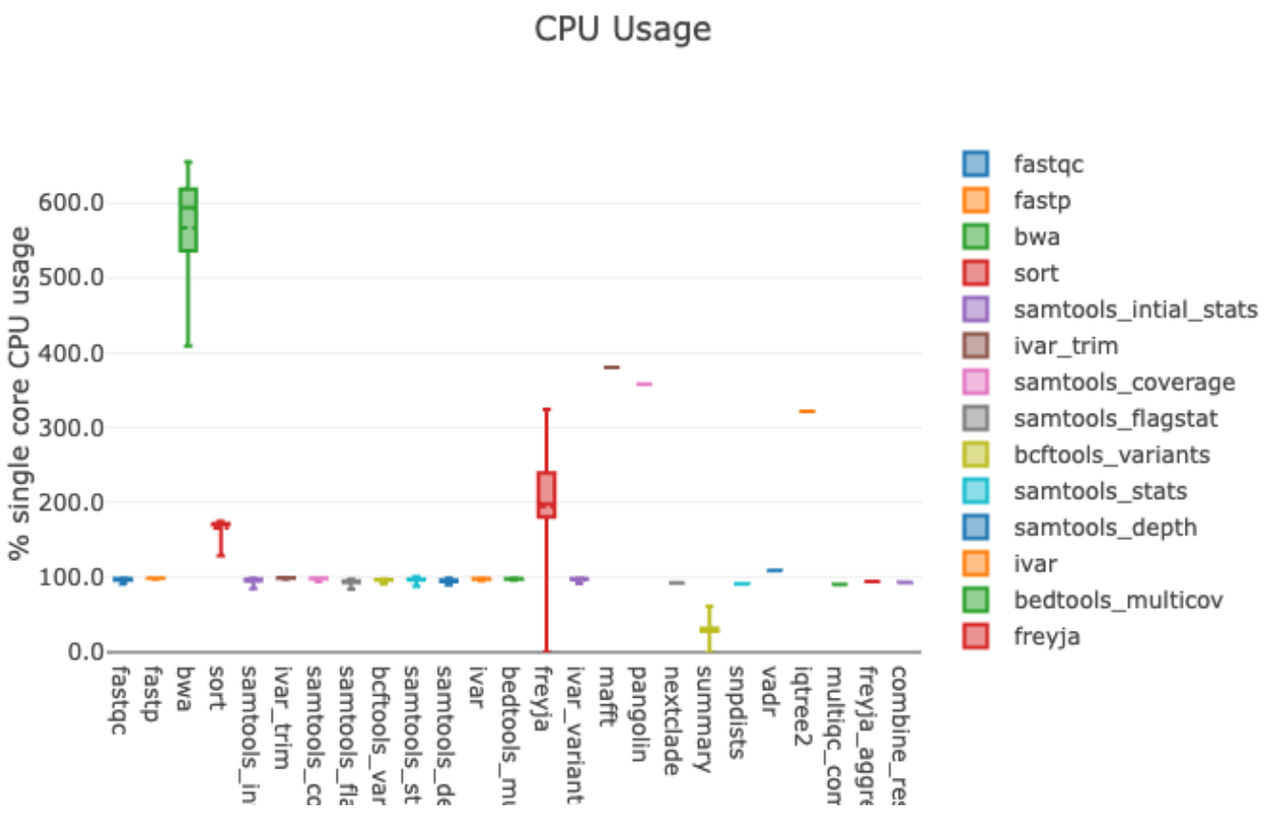
Nextflow

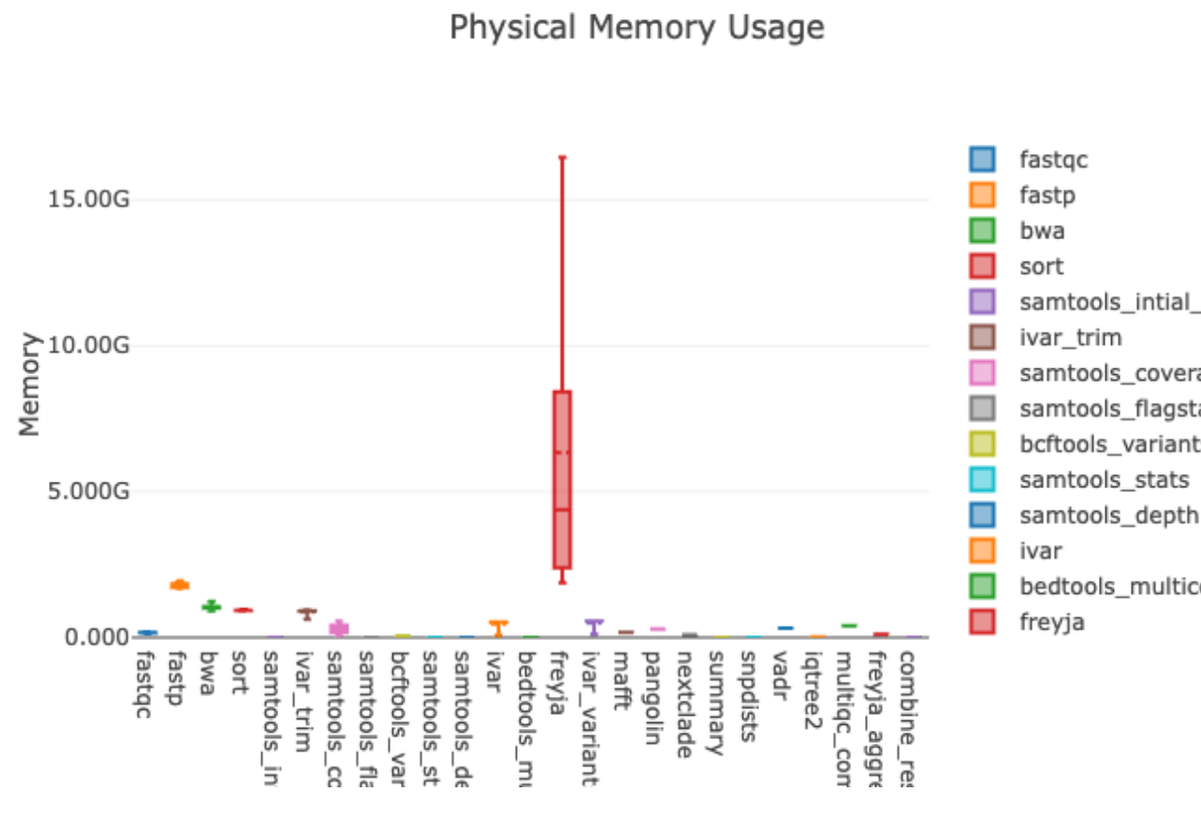
Reportes

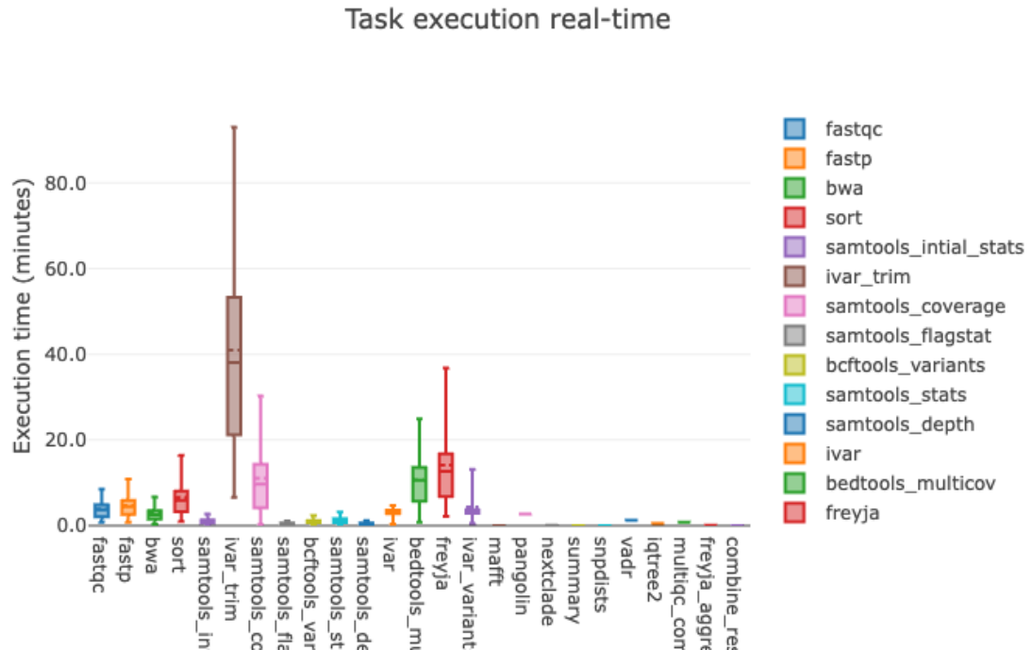


Nextflow

Reportes

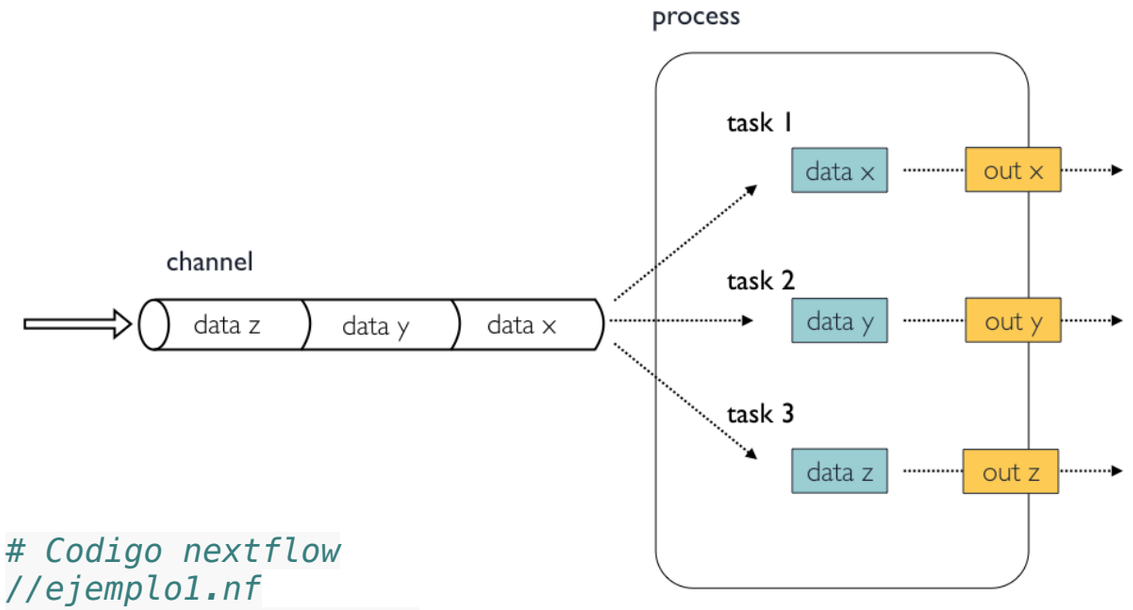






Nextflow

Bases de programación



- Un script de Nextflow se crea uniendo diferentes **procesos**. Cada proceso se puede escribir en cualquier lenguaje de comandos que pueda ejecutar la plataforma Linux (Bash, Perl, Ruby, Python, etc.).

- Los **procesos** se ejecutan de forma independiente y están aislados entre sí, es decir, no comparten un estado común (de escritura). La única forma de comunicación es a través de colas FIFO asíncronas, llamadas **canales** en Nextflow.

- Cualquier proceso puede definir uno o más **canales** como entrada y salida. La interacción entre estos procesos y, en última instancia, el propio flujo de ejecución del pipeline, se define implícitamente mediante estas declaraciones de entrada y salida.

```
# Codigo nextflow
//ejemplo1.nf
nextflow.enable.dsl=2

process NUM_LINEAS {
    input:
    path read
    script:
    """
    printf '${read} '
    gunzip -c ${read} | wc -l
    """
}

reads_ch = Channel.fromPath( 'reads/ref*.fq.gz')

workflow {
  NUMLINES(reads_ch)
}

# Bash
 nextflow run ejemplo1.nf -process.echo

# output
[cd/77af6d] process > NUMLINES (1) [100%] 6 of 6 ✔
ref1_1.fq.gz 58708
ref3_2.fq.gz 52592
ref2_2.fq.gz 81720
ref2_1.fq.gz 81720
ref3_1.fq.gz 52592
ref1_2.fq.gz 58708
```

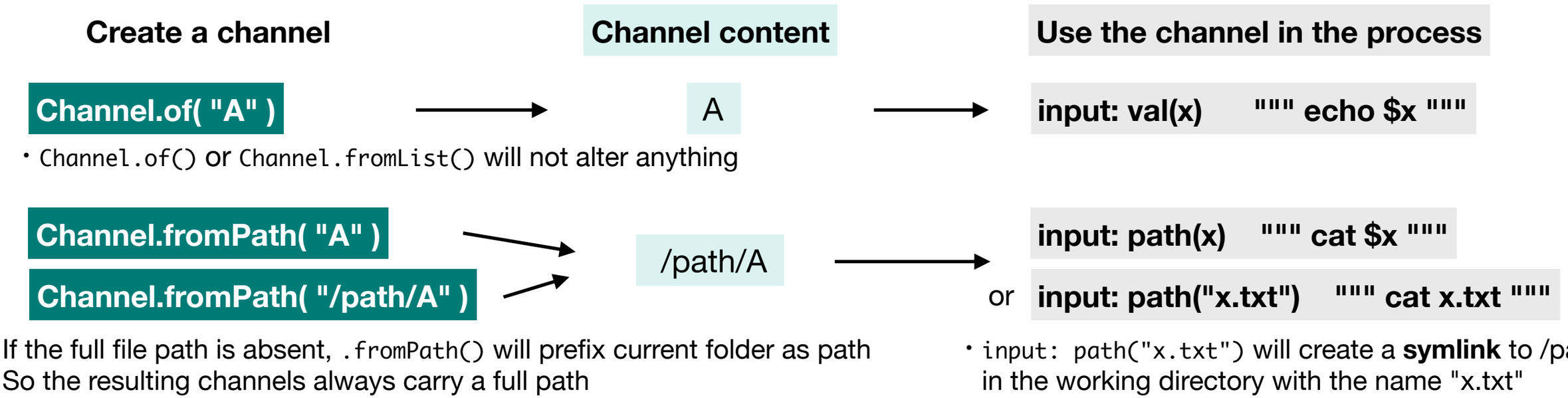
Nextflow

Primer pipeline

Procesos

```
//pipeline_01.nf
nextflow.enable.dsl=2

process INDEX {
    input:
      path transcriptome
    output:
      path 'index'
    script:
      """
      salmon index -t $transcriptome -i index
      """
}

 process QUANT {
    input:
      each  path(index)
      tuple(val(pair_id), path(reads))
    output:
      path pair_id
    script:
      """
      salmon quant --threads $task.cpus --libType=U \
          -I $index \
          -1 ${reads[0]} -2 ${reads[1]} -o $pair_id
      """
}

//Definición del pipeline
workflow {
     //canales de entrada
    transcriptome_ch =
     channel.fromPath('transcriptome/*.fa.gz',checkIfExists: true)
    read_pairs_ch =
     channel.fromFilePairs('reads/*_{1,2}.fq.gz',checkIfExists: true)
    //el proceso INDEX toma el calar transcriptome_ch
    index_ch = INDEX(transcriptome_ch)
    //el proceso QUANT toma dos canales como argumentos: Indice y las lecturas
    QUANT( index_ch, read_pairs_ch ).view()
}
```

Channels



- `Channel.of()` or `Channel.fromList()` will not alter anything

- If the full file path is absent, `.fromPath()` will prefix current folder as path
- So the resulting channels always carry a full path

- `input: path("x.txt")` will create a **symlink** to /path/A in the working directory with the name "x.txt"

Channels

Input text file content

A
B
C

Create a channel

```
Channel.fromPath( "file.txt" )
    .splitText { it.strip( ) }
```

- `.splitText` will read in the file.txt
- `it.strip()` will remove the blank items

Channel content

A →
B →
C →

Use the channel in the process

input: val(x)

- 1 channel, 3 items
- 3 parallel executions of process

```
Channel.fromPath( "file.txt" )
    .splitText { it.strip( ) }
    .map { it -> file(it) }
```

- `file()` adds current folder as path unless there is already full path in the item

/path/A →
/path/B →
/path/C →

input: path(x)

- 1 channel, 3 items
- 3 parallel executions of process

A	/path/A.bam	/path/A.bam.bai
B	/path/B.bam	/path/B.bam.bai
C	/path/C.bam	/path/C.bam.bai

```
Channel.fromPath( "file.tsv" )
    .splitCsv( sep: "\t" )
    .map { row -> [ row[0], file(row[1]), file(row[2]) ] }
```

[A, /path/A.bam, /path/A.bam.bai] →
[B, /path/B.bam, /path/B.bam.bai] →
[C, /path/C.bam, /path/C.bam.bai] →

- `.map{ }` is very useful to select columns and specify channel structure. Here it converts the row to a tuple

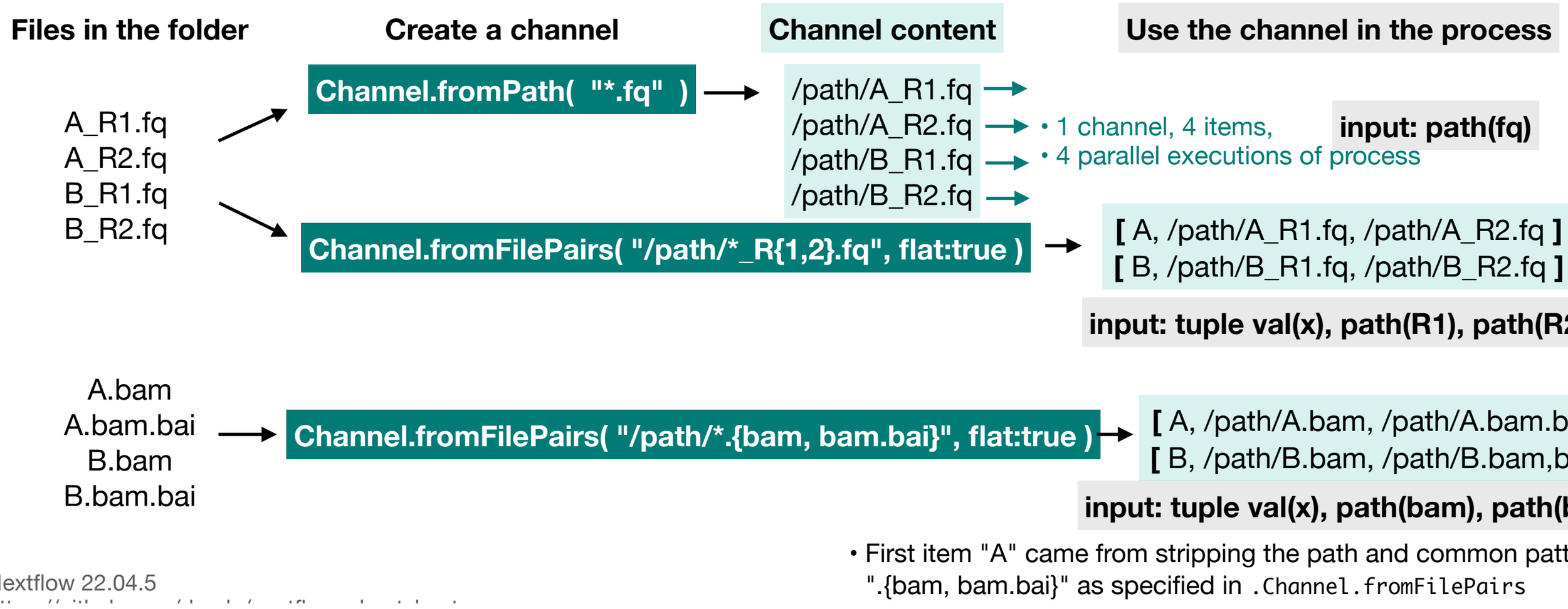
input: tuple val(x), path(bam), path(bai)

strain	bam
A	A.bam
B	B.bam
C	C.bam

```
Channel.fromPath( "file.tsv" )
    .splitCsv( header:true, sep: "\t" )
    .map { row ->
                if ( params.bam_path != "" ) {
                    row.bam = "${params.bam_path}/${row.bam}"
                    }
            [ row.strain, file("${row.bam}"), file("${row.bam}.bai") ] }
```

- A path is added with params.bam_path
- If "params.bam_path" doesn't add a full path, `file( )` will.
- `[ x, y ]` is the same as `tuple( x, y )`

Channels


Files in the folder
Create a channel
Channel content
Use the channel in the process
A_R1.fq
A_R2.fq
B_R1.fq
B_R2.fq
Channel.fromPath("*.fq")
/path/A_R1.fq
/path/A_R2.fq
/path/B_R1.fq
/path/B_R2.fq
• 1 channel, 4 items,
• 4 parallel executions of process
input: path(fq)
Channel.fromFilePairs("/path/*_R{1,2}.fq", flat:true)
[A, /path/A_R1.fq, /path/A_R2.fq]
[B, /path/B_R1.fq, /path/B_R2.fq]
input: tuple val(x), path(R1), path(R2)
A.bam
A.bam.bai
B.bam
B.bam.bai
Channel.fromFilePairs("/path/*.{bam, bam.bai}", flat:true)
[A, /path/A.bam, /path/A.bam.bai]
[B, /path/B.bam, /path/B.bam,bai]
input: tuple val(x), path(bam), path(bai)
• First item "A" came from stripping the path and common pattern ".{bam, bam.bai}" as specified in .Channel.fromFilePairs

Channels

examples of 1 channel:

Channel.of("A.fa", "B.fa", "C.fa" **)**

A.fa →
B.fa →
C.fa →

- **Each item (row) has 1 execution of the proces**
- Here is 1 channel with 3 items
- 3 parallel executions of the process

Channel.of(["A.fa", "B.fa", "C.fa" **])**

[A.fa, B.fa, C.fa] →

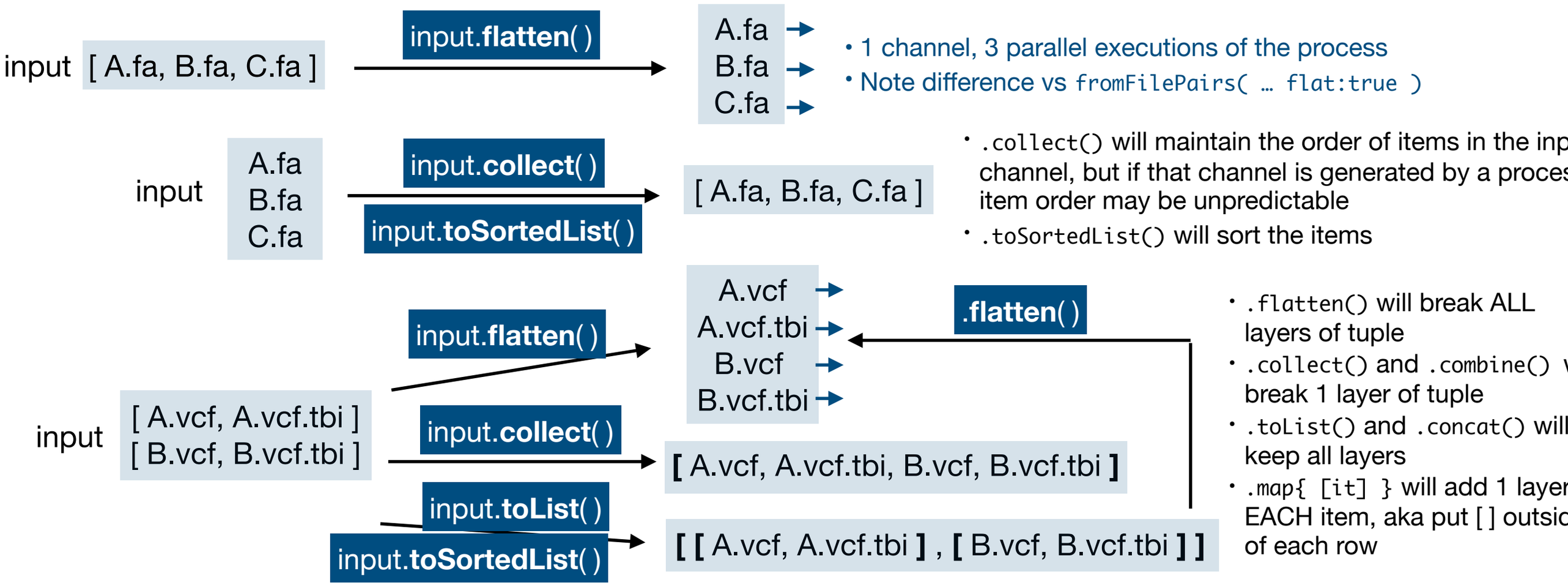
- 1 channel, 1 item, 1 execution

Channel.of(["I.vcf", "I.vcf.tbi" **] ,**
["II.vcf", "II.vcf.tbi" **] ,**
["III.vcf", "III.vcf.tbi" **])**

[I.vcf, I.vcf.tbi] →
[II.vcf, II.vcf.tbi] →
[III.vcf, III.vcf.tbi] →

- 1 channel, 3 items
- 3 parallel executions of process
- [ ] indicates a “set” “tuple” “ArrayList”

Channels


input
[A.fa, B.fa, C.fa]
input.flatten( )
A.fa
B.fa
C.fa
• 1 channel, 3 parallel executions of the process
• Note difference vs fromFilePairs(… flat:true)
input
A.fa
B.fa
C.fa
input.collect( )
input.toSortedList( )
[A.fa, B.fa, C.fa]
• .collect() will maintain the order of items in the inp
channel, but if that channel is generated by a proces
item order may be unpredictable
• .toSortedList() will sort the items
input.flatten( )
input
[A.vcf, A.vcf.tbi]
[B.vcf, B.vcf.tbi]
input.collect( )
input.toList( )
input.toSortedList( )
A.vcf
A.vcf.tbi
B.vcf
B.vcf.tbi
.flatten( )
[A.vcf, A.vcf.tbi, B.vcf, B.vcf.tbi]
[[A.vcf, A.vcf.tbi] , [B.vcf, B.vcf.tbi]]
• .flatten() will break ALL
layers of tuple
• .collect() and .combine()
break 1 layer of tuple
• .toList() and .concat() will
keep all layers
• .map{ [it] } will add 1 layer
EACH item, aka put [ ] outsi
of each row

Channels

vcf [I.vcf]
index [I.vcf.tbi]

vcf.**combine**(index) → [I.vcf, I.vcf.tbi]

- Note how `.combine()` works when there are multiple items in each input channels (see bottom section)

vcf1 [I.vcf, I.vcf.tbi]
vcf2 [II.vcf, II.vcf.tbi]

vcf1.**combine**(vcf2) → [I.vcf, I.vcf.tbi, II.vcf, II.vcf.tbi]

vcf1.**concat**(vcf2) → [I.vcf, I.vcf.tbi] [II.vcf, II.vcf.tbi]

.collect( )

- `.concat()` and then `.collect()` sometimes gives the same result as `.combine()`, but they are much more versatile for diverse input types

vcf [I.vcf, I.vcf,tbi] [II.vcf, II.vcf,tbi]
contig contigs.txt

vcf.**concat**(contig) → [I.vcf, I.vcf,tbi] [II.vcf, II.vcf,tbi] contigs.txt

.collect( ) → [I.vcf, I.vcf.tbi, II.vcf, II.vcf.tbi, contigs.txt]

- Can use **input: path("*")** to pass all files to the process

vcf1 [I.vcf, I.vcf.tbi]

vcf1.**map { [it] }** → [[I.vcf, I.vcf.tbi]]

.combine(contig) → [[I.vcf, I.vcf.tbi], contigs.txt]

- This is especially useful when number of items in vcf1 could vary. In process: **input: tuple path("*"), path(contig)**

vcf A.vcf B.vcf
contig chr1 chr2

vcf.**combine**(contig) → [A.vcf, chr1] [A.vcf, chr2] [B.vcf, chr1] [B.vcf, chr2]

- All combinations

input1 [A, 1] [B, 2]
input2 [A, I] [B, m] [B, n]

- counting starts from 0

input1.**combine**(input2,**by:0**) → [A, 1, I] [B, 2, m] [B, 2, n

input1.**join**(input2) → [A, 1, I] [B, 2, m

input2.**collect { it[1] }** → [I, m, n]

- "**it**" represents an item, which is
- `join()` will keep only the first match

Nextflow patterns

```
process foo {
  debug true
  input:
  path x

  script:
  """
  echo your_command --input $x
  """
}

workflow {
  foo("$baseDir/data/reads/*_1.fq.gz")
}
```

Process per file path

```
process foo {
  debug true

  input:
  tuple val(sampleId), file(reads)

  script:
  """
  echo your_command --sample $sampleId --reads $reads
  """
}

workflow {
  Channel.fromFilePairs("$baseDir/data/reads/*_{1,2}.fq.gz", checkIfExists:true) \
    | foo
}
```

Process per file pairs

Nextflow patterns

Table 1

sampleId	read1	read2
FC816RLABXX	110101_I315_FC816RLABXX_L1_HUMrutRGXDIAAPE_1	110101_I315_FC816RLABXX_L1_HUMrutRGXDIAAPE_2
C812MWABXX	110105_I186_FC812MWABXX_L8_HUMrutRGVDIABPE_1	110105_I186_FC812MWABXX_L8_HUMrutRGVDIABPE_
FC81DE8ABXX	110121_I288_FC81DE8ABXX_L3_HUMrutRGXDIAAPE_1	110121_I288_FC81DE8ABXX_L3_HUMrutRGXDIAAPE_2
FC81DB5ABXX	110122_I329_FC81DB5ABXX_L6_HUMrutRGVDIAAPE_1	110122_I329_FC81DB5ABXX_L6_HUMrutRGVDIAAPE_2
FC819P0ABXX	110128_I481_FC819P0ABXX_L5_HUMrutRGWDIAAPE_1	110128_I481_FC819P0ABXX_L5_HUMrutRGWDIAAPE_2

```
params.index = "$baseDir/data/index.csv"

process foo {
    debug true
    input:
    tuple val(sampleId), file(read1), file(read2)

    script:
    """
    echo your_command --sample $sampleId --reads $read1 $read2
    """
}

workflow {
    Channel.fromPath(params.index) \
        | splitCsv(header:true) \
        | map { row-> tuple(row.sampleId, file(row.read1), file(row.read2)) } \
        | foo
}
```

Nextflow patterns

```
process foo {
  input:
  path x
  output:
  path 'file.fq'
  script:
  """
  < $x zcat > file.fq
  """
}

process bar {
  debug true
  input:
  path '*.fq'
  script:
  """
  cat *.fq | head -n 50
  """
}

workflow {
  Channel.fromPath("$baseDir/data/reads/*_1.fq.gz", checkIfExists: true) \
    | foo \
    | collect \
    | bar
}
```

Process all outputs altogether

Real example

Nextflow

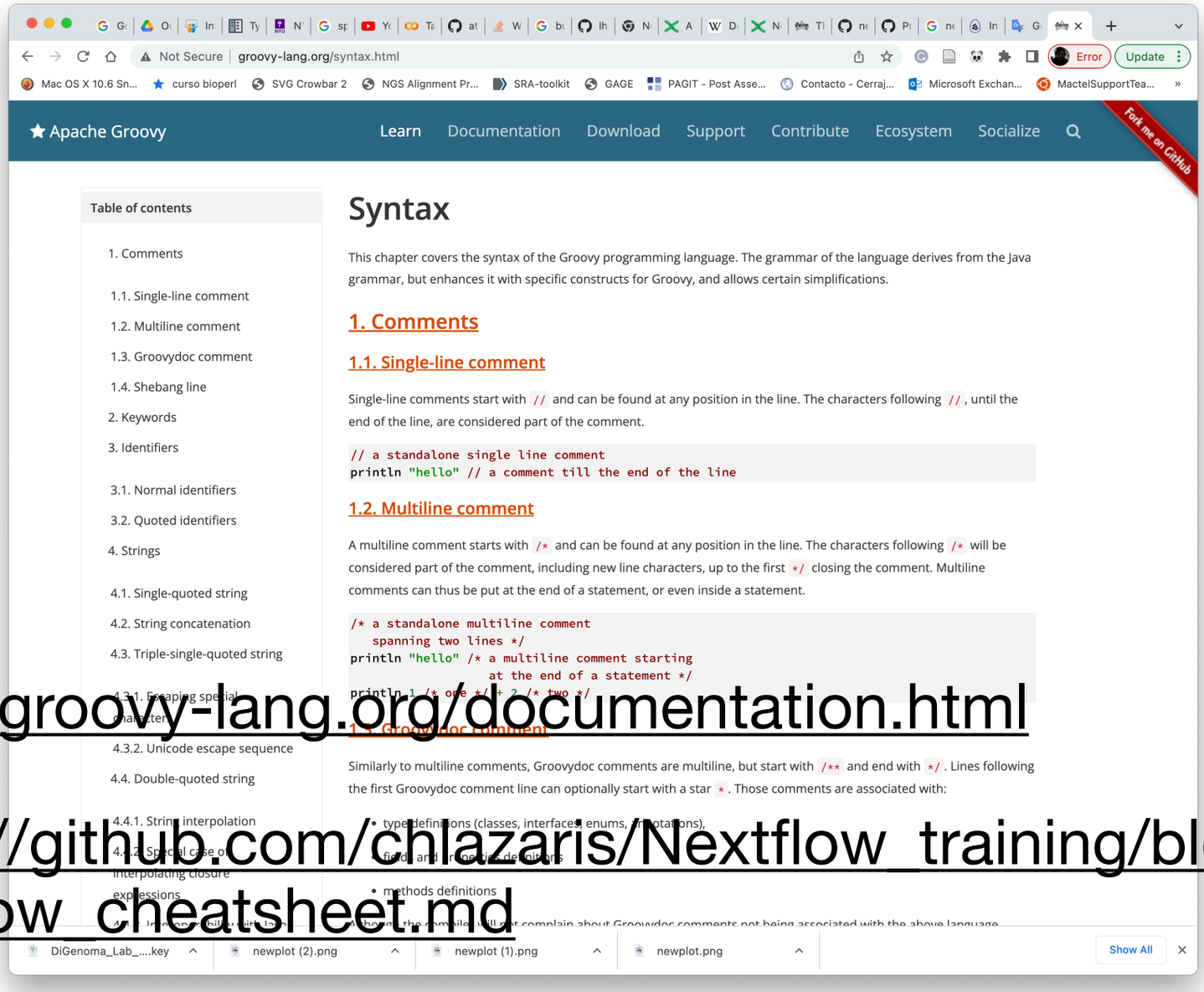
- BRCA pipeline
- https://github.com/digenoma-lab/BRCA/blob/main/

```
#!/usr/bin/env nextflow

nextflow.enable.dsl = 2
//check some variables before execution

//loading scripts from modules
include {FASTQC} from './modules/fastqc'
include {BWAMEM} from './modules/bwamem'
include {MERGEB} from './modules/mergeb'
include {ELPREP} from './modules/elprep'
include {SAMTOOLS} from './modules/samtools'
include {QUALIMAP} from './modules/qualimap'
include {B2C} from './modules/b2c'
include {STRELKA_ONESAMPLE} from './modules/strelka'
include {STRELKA_POOL} from './modules/strelka'
include {BCFTOOLS_FILTER; BCFTOOLS_FILTER as BF} from './modules/bcftools'
include {ANNOVAR} from './modules/annovar'
```

Nextflow

Groovy

- Apache Groovy es un lenguaje de programación orientado a objetos compatible con la sintaxis de Java para la plataforma Java.



- http://groovy-lang.org/documentation.html
- https://github.com/chlazaris/Nextflow_training/blob/main/nextflow_cheatsheet.md
- https://www.nextflow.io/docs/latest/script.html

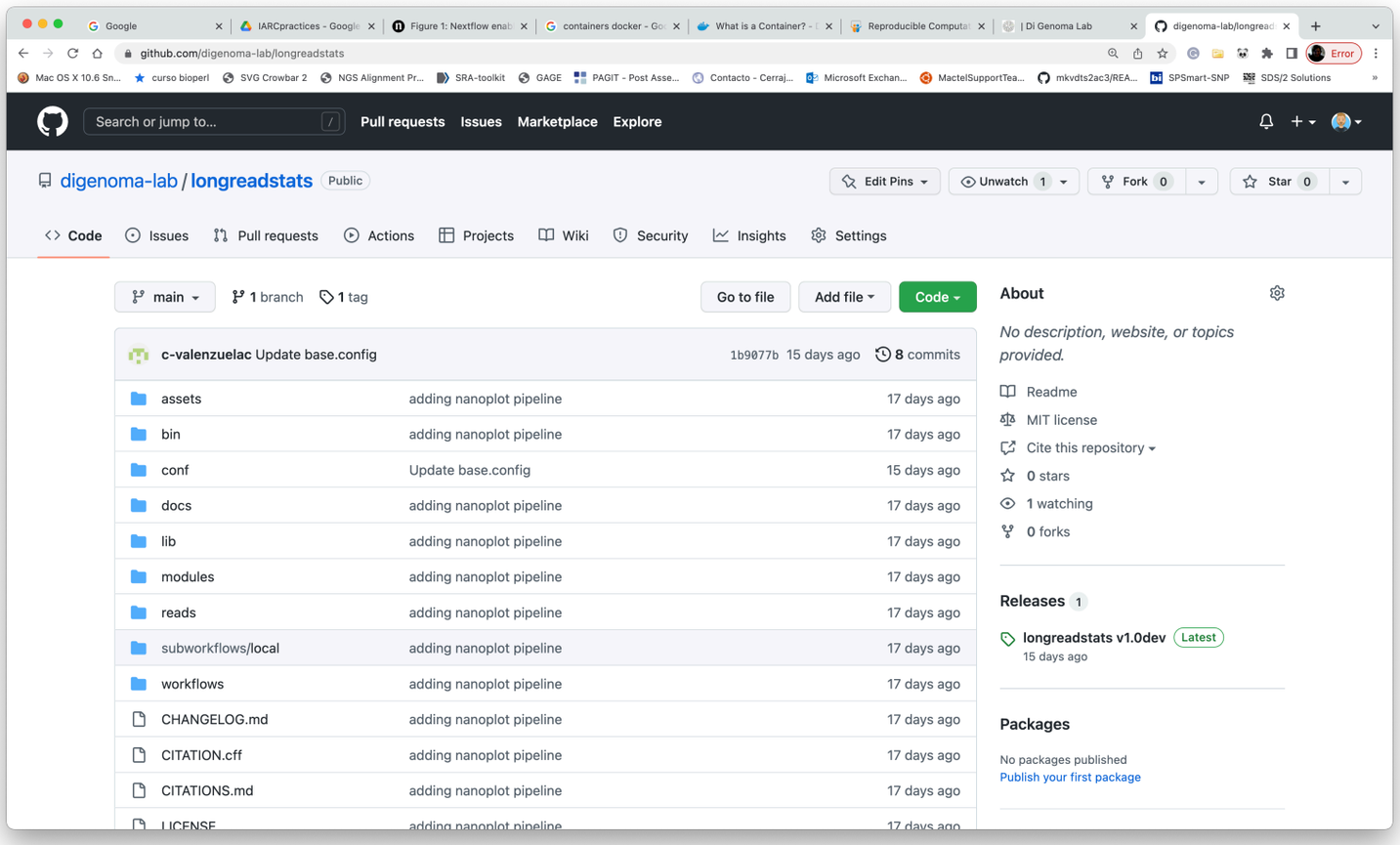
Nextflow

Creando pipelines con el estandar nf-core

```
nf-core create -n long-read-stats  -d "pipeline for computing long-read statistics" -a "Alex Di Genova"
```

```
nf-core modules list remote
```

```
nf-core modules install nanoplot
```




```
[adigenova@leftraru1 ~]$ nf-core modules list remote
```

```
nf-core/tools version 2.5.1 - https://nf-co.re
There is a new version of nf-core/tools available! (2.6
```

```
INFO     Modules available from nf-core/modules (master):
```

Module Name
nf-core/abacas
nf-core/abricate/run
nf-core/abricate/summary
nf-core/adapterremoval
nf-core/adapterremovalfixprefix
nf-core/agrvate
nf-core/allelecounter
nf-core/ampir
nf-core/amplify/predict
nf-core/amps
nf-core/amrfinderplus/run
nf-core/amrfinderplus/update
nf-core/angsd/docounts
nf-core/antismash/antismashlite
nf-core/antismash/antismashlitedownloaddatabases
nf-core/aria2
nf-core/ariba/getref
nf-core/ariba/run

Preguntas?