BioLizard

Welcome to the

NEXTFLOW WORKSHOP

Tuur Muyldermans: tuur.muyldermans@lizard.bio
Steff Taelman: steff.taelman@lizard.bio

Welcome to the nextflow workshop

Objectives:

- Understand Nextflow syntax
- Understand workflow pipelines
- Write simple pipelines yourself!

Results pre-workshop survey.

Schedule

Schedule day 1:

- 9:30 - 11:00 - session
- 11:00 - 11:15 - break
- 11:15 - 12:45 - session
- 12:45 - 13:45 - lunch
- 13:45 - 15:15 - session
- 15:15 - 15:30 - break
- 15:30 - 17:00 - session

Schedule

Schedule day 2:

- 9:30 - 13:00 - project

Course materials

- Clone from GitHub:

```
git clone https://github.com/vibbits/nextflow-workshop.git
```

- Structure of course materials

Overview:

- Introduction
- Basic concepts: processes, channels and operators
- Creating our first Nextflow script(s)
- Managing configurations: parameters, portability, execution
- Creating reports

1. Building blocks

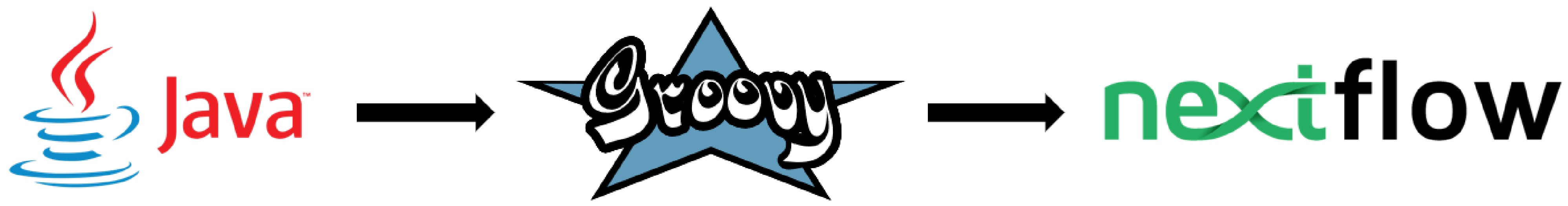
In the first chapter we will elaborate on how Nextflow is designed, its advantages and disadvantages, the basic components, etc.

Bash scripts

```
#!/bin/bash

...

# Download each fasta read sequence file into the directory
for file in $LIST; do
    echo "Downloading $file"
        wget -P ../data -np ${rawdatalink}/$file
done

...
```

Workflow managers

> **Nextflow** is a reactive workflow framework and a programming Domain Specific Language that eases the writing of data-intensive computational pipelines.



Alternatives: [link]()

Why Nextflow?

- Parallelization
- Scalability
- Portability
- Reproducible
- Continuous checkpoints
- Modularity
- Community

Why not?

- Syntax of the Groovy language, yet another language
- Flexibility also comes with cost of complexity
- Nitpicking details in failure of scripts

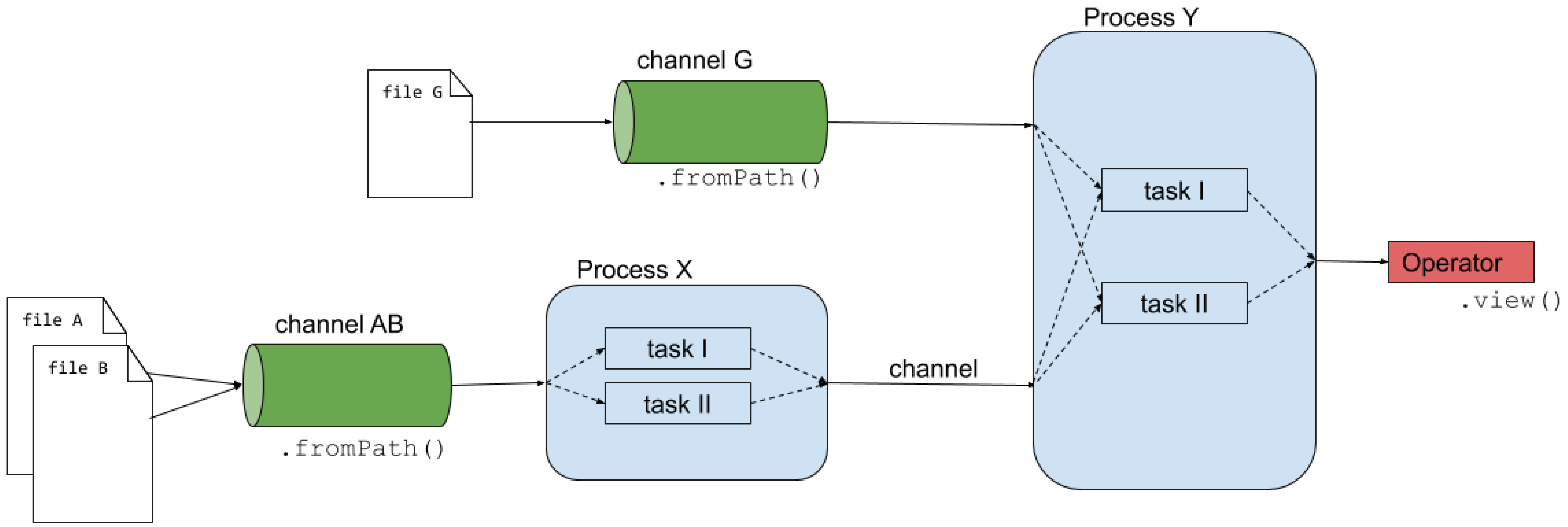
Interact with Nextflow:

```
nextflow [options] COMMAND [arg...]
```

E.g.:

```
nextflow run hello
```

2. Basic concepts



```
#!/usr/bin/env nextflow
nextflow.enable.dsl=2

// Creating channels
numbers_ch = Channel.from(1,2,3)
strings_ch = Channel.from('a','b')

// Defining the process that is executed
process valuesToFile {
    input:
    val nums
    val strs

    output:
    path 'result.txt'

    """
    echo $nums and $strs > result.txt
    """
}

// Running a workflow with the defined processes
workflow{
    valuesToFile(numbers_ch, strings_ch)
}
```

2.1 Channels

```
# Channel consisting of strings
strings_ch = Channel.from('This', 'is', 'a', 'channel')

# Channel consisting of a single file
file_ch = Channel.fromPath('data/sequencefile.fastq')

# Channel consisting of multiple files by using a wildcard *
multfiles_ch = Channel.fromPath('data/*.fastq')
```

Further reading: Nextflow's documentation.

2.2 Operators

- `collect`: e.g. when using a channel consisting of multiple independent files (e.g. fastq-files)

```
Channel
    .from( 1, 2, 3, 4 )
    .collect()
    .view()

# outputs
[1,2,3,4]
```

Further reading: [Nextflow's documentation]

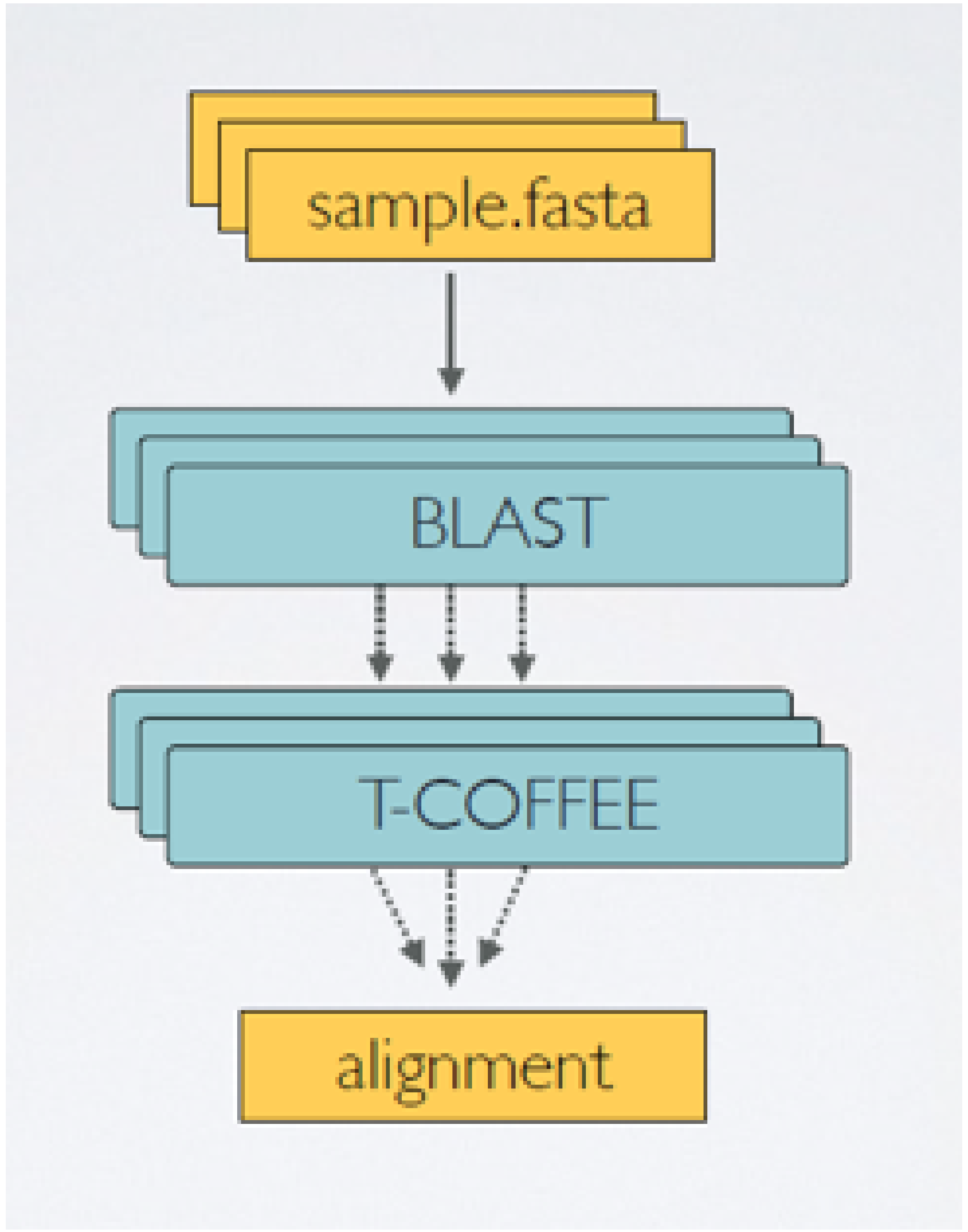
- `mix`: e.g. when assembling items from multiple channels into one channel for a next process (e.g. multiqc)

```
c1 = Channel.from( 1,2,3 )
c2 = Channel.from( 'a','b' )
c3 = Channel.from( 'z' )

c1 .mix(c2,c3)

# outputs
1
2
3
'a'
'b'
'z'
```

2.3 Processes

```
process < name > {
   [ directives ]

   input:
    < process inputs >

   output:
    < process outputs >

   when:
    < condition >

   [script|shell|exec]:
   < user script to be executed >
}
```

- Executed independently
- Isolated from any other process
- FIFO queues



Examples in the course materials:

- `valuesToFile`
- `fastqc`
- `salmon_quant`
- `trimmomatic`

FIFO-principle

```
nextflow run 02-basic-consepts/fifo.nf
```

```
N E X T F L O W  ~  version 20.07.1
Launching `02-basic-concepts/fifo.nf` [nauseous_mahavira] - revision: a71d904cf6
[-        ] process > whosfirst [  0%] 0 of 2
This is job number 6
This is job number 3
This is job number 7
This is job number 8
...
This is job number 2
This is job number 9
executor >  local (10)
[4b/aff57f] process > whosfirst (10) [100%] 10 of 10
```

A script, as part of the process, can be written in any language (bash, Python, Perl, Ruby, etc.). This allows to add self-written scripts in the pipeline.

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

process python {

    """
    #!/usr/bin/python3

    firstWord = 'hello'
    secondWord = 'folks'
    print(f'{firstWord} {secondWord}')
    """
}
```

2.4 Workflows

```
workflow{
    trimmomatic(reads)
}
```

```
workflow {
  trimmomatic(reads)
  fastqc_trim(trimmomatic.out.trim_fq)
}
```

2.5 Executing pipelines

```
nextflow run firstscript.nf
```

```
N E X T F L O W  ~  version 20.07.1
Launching `02-basic-concepts/firstscript.nf` [elegant_curie] - revision: 9f886cc00a
executor >  local (2)
executor >  local (2)
[5e/195314] process > valuesToFile (2) [100%] 2 of 2 ✓
results file: /path/to/work/51/7023ee62af2cb4fdd9ef654265506a/result.txt
results file: /path/to/work/5e/195314955591a705e5af3c3ed0bd5a/result.txt
```

Besides the output, also a bunch of hidden `.command.*` files are present:

```
-.command.begin*
-.command.err*
-.command.log*
-.command.out*
-.command.run*
-.command.sh*
-.exitcode*
```

Parameters

- Nextflow related parameters
    - Predefined in Nextflow's language
    - Set with single dash and define something about the execution:

      ```
      nextflow -bg -resume -with-report -work-dir <pipeline.nf>
      ```

- Pipeline parameters:
    - Manually and specific created for a given pipeline
    - `params.reads = ''` in the pipeline script or config file
    - Overwritten on runtime with double dashes

      ```
      nextflow run <pipeline.nf> --reads 'read.fq'
      ```

Publicly available pipelines

- Some curated nextflow pipelines are available on awesome-nextflow.
- Pipelines from the nf-core community.
- Pipelines from WorkflowHub (this is a currently ongoing effort).

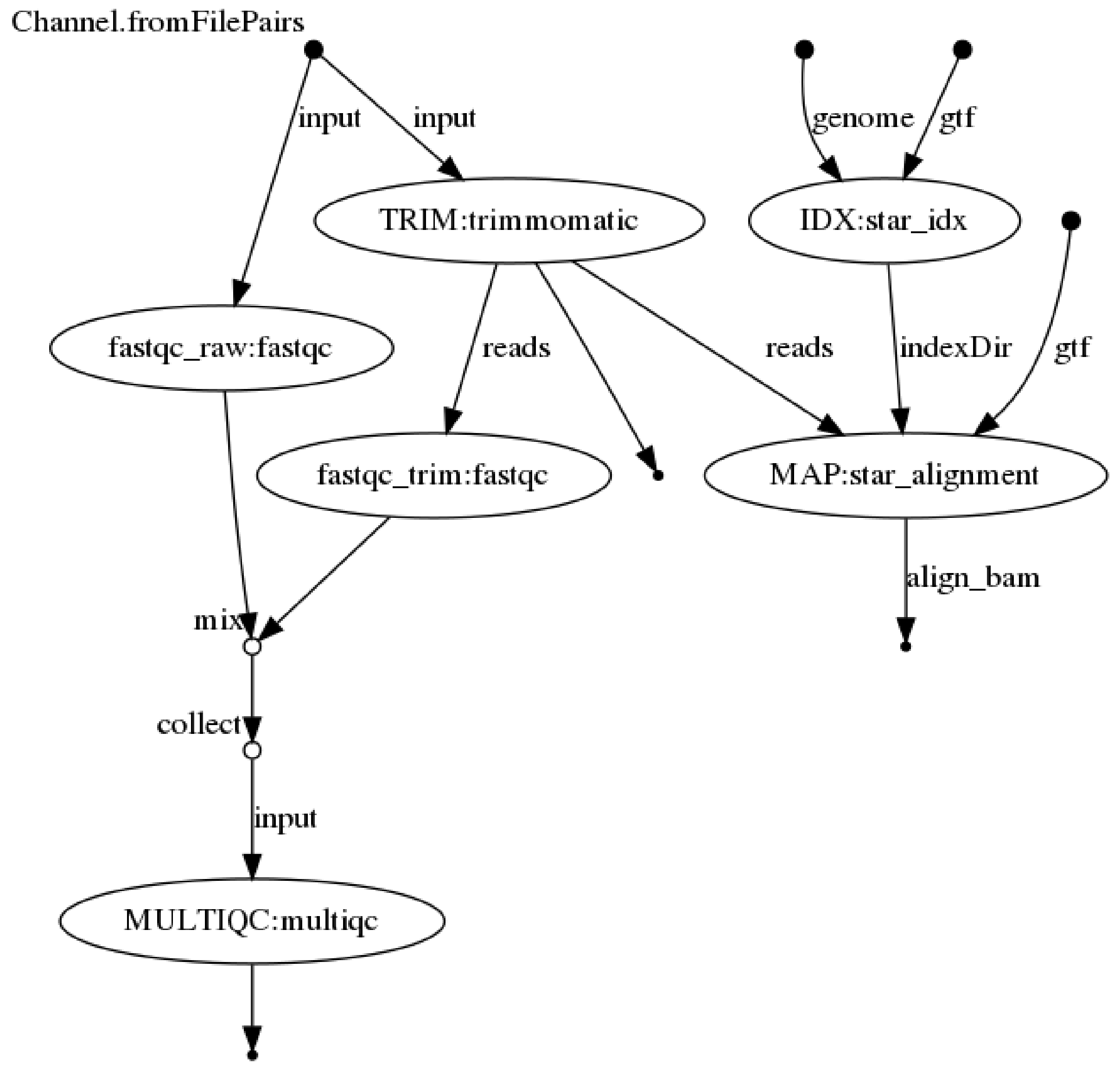
Importing a pipeline

```
nextflow [run/pull/clone] nextflow-io/rnaseq-nf
```

```
nextflow run nextflow-io/rnaseq-nf -r [v1.2/master/98ffd10a76]
nextflow pull nextflow-io/rnaseq-nf
nextflow clone nextflow-io/rnaseq-nf target-dir
```

More information: here

3. Creating our first pipeline



4. Managing configurations

- Configuration files (parameters, labels, etc.)
- Portability & reproducibility
- Executors

Configuration files

- Pipeline configuration properties are defined in `nextflow.config`
- Technical parameters:
  - Executor, CPUs, memory, containers etc.
- Pipeline specific parameters:
  - Input files, process related parameters (e.g. trimmomatic or STAR)
- Make pipelines more modular.

The parameters can be defined with `params.<name> = <value>` or join them all in one long list as such:

```
params.reads = "$launchDir/data/*{1,2}.fq.gz"

params.refdir = "/path/to/references"
params.genome = "${refdir}/Drosophila_melanogaster.BDGP6.dna.fa"
params.gtf = "${refdir}/Drosophila_melanogaster.BDGP6.85.sample.gtf"
```

```
// Define project parameters needed for running the pipeline
params {
    // General parameters
    projdir = "/path/to/data"
    refdir = "/path/to/references"
    outdir = "/path/to/data-analysis"

    // Reference genome and annotation files
    genome = "${refdir}/Drosophila_melanogaster.BDGP6.dna.fa"
    gtf = "${refdir}/Drosophila_melanogaster.BDGP6.85.sample.gtf"

    // Input parameters
    reads = "${projdir}/*{1,2}.fq.gz"

    ...
}
```

- Separate analysis parameteres in a separate file

```
includeConfig "/path/to/params.config"
```

Technical parameters (local executor):

```
process {
    memory='1G'
    cpus='1'
}
```

```
// Define technical resources below:
process {
    withLabel: 'low' {
        memory='1G'
        cpus='1'
        time='6h'
    }
    withLabel: 'med' {
        memory='2G'
        cpus='2'
    }
    withLabel: 'high' {
        memory = '8G'
        cpus='8'
    }
}
```

Executors (example):

Schedulers

GRID ENGINE
Sun Grid Engine (SGE)

slurm
workload manager

LSF
Platform Load Sharing Facility

PBS Works™
Portable Batch System

HTCondor
High Throughput Computing

Cloud platforms

kubernetes

amazon web services™

Google Cloud Platform

Portability & reproducibility

- Support for Conda, Docker & Singularity

How to execute pipelines with containers:

- Run pipeline with Docker container:

```
nextflow run example.nf -with-docker [docker image]
```

- Or add the following to `nextflow.config`-file:

```
process.container = 'vibbioinfocore/analysispipeline:latest'
docker.enabled = true
```

Note: to set the correct user- and group-settings:
`docker.runOptions = '-u \$(id -u):\$(id -g)'`

What could go wrong in this situation?

A more modular approach:
Define the containers in the process directives:

```
process quality-control {
    container 'biocontainers/fastqc:v0.11.9_cv7'

    ...
}
```

- Run pipeline with Singularity image:

```
nextflow run example.nf -with-singularity [singularity-image-file]
```

Or extend `nextflow.config`-file with:

```
singularity.cacheDir = "/path/to/singularity" // centralised caching directory
process.container = 'singularity.img'         // define the image
singularity.enabled = true                    // enable running with singularity
```

Profiles allow to execute a pipeline with a number of parameters defined in `profiles`:

- Locally with conda:

```
nextflow run main.nf -profile standard,conda
```

- Locally with docker:

```
nextflow run main.nf -profile standard,docker
```

- On Microsoft Azure with Docker:

```
nextflow run main.nf -profile azure,docker
```

5. Creating reports

- Workflow report (html): `-with-report`
- DAG: visualization of the pipeline: `-with-dag <filename.PNG>`
- Tower: `-with-tower`

Questions

Further reading & references:

- Nextflow's official documentation (link)
- Reach out to the community on Gitter (link)
- Curated collection of patterns (link)
- Workshop focused on DSL2 developed by CRG Bioinformatics Core (link)
- Tutorial exercises (DSL1) developed by Seqera (link)
- Curated ready-to-use analysis pipelines by NF-core (link)
- Model example pipeline on Variant Calling Analysis with NGS RNA-Seq data developed by CRG (link)