NGS Analysis

learn.gencore.bio.nyu.edu

Nextflow + Containerization

Presented by: Tobias Schraink and Stephen Kelly

March 28, 2019

Prerequiste

GitHub: BADAS Nextflow workshop

Prerequisite

- Java 8 or later
- Nextflow
- Conda
- Gitclone the repo into a directory in your linux environment

Nextflow hands-on

During this tutorial you will implement a proof of concept of a RNA-Seq pipeline which:

1. Indexes a trascriptome file.
2. Performs quality controls
3. Performs quantification.
4. Create a MultiqQC report.

Step 1 – define the pipeline parameters

The script `script1.nf` defines the pipeline input parameters. Run it by using the following command:

```
nextflow run script1.nf
```

Try to specify a different input parameter, for example:

```
nextflow run script1.nf --reads this/and/that
```

Exercise 1.1

Modify the `script1.nf` adding a fourth parameter named `outdir` and set it to a default path that will be used as the pipeline output directory.

Exercise 1.2

Modify the `script1.nf` to print all the pipeline parameters by using a single `println` command and a multiline stringstatement.

Tip: see an example here.

Recap

In this step you have learned:

1. How to define parameters in your pipeline script
2. How to pass parameters by using the command line
3. The use of `$var` and `${var}` variable placeholders
4. How to use multiline strings

Step 2 – Create transcriptome index file

Nextflow allows the execution of any command or user script by using a `process` definition.

A process is defined by providing three main declarations: the process inputs, the process outputs and finally the command script.

The second example adds the `index` process. Open it to see how the process is defined.

It takes the transcriptome file as input and creates the transcriptome index by using the `salmon` tool.

Note how the input declaration defines a `transcriptome` variable in the process context that it is used in the command script to reference that file in the Salmon command line.

Try to run it by using the command:

```
nextflow run script2.nf
```

Exercise 2.1

Print the output of the `index_ch` channel by using the println operator (do not confuse it with the `println` statement seen previously).

Exercise 2.3

Use the command `tree -a work` to see how Nextflow organises the process work directory.

Recap

In this step you have learned:

1. How to define a process executing a custom command
2. How process inputs are declared
3. How process outputs are declared
4. How to access the number of available CPUs
5. How to print the content of a channel

Step 3 – Collect read files by pairs

This step shows how to match *read* files into pairs, so they can be mapped by *Salmon*.

Edit the script `script3.nf` and add the following statement as the last line:

```
read_pairs_ch.println()
```

Save it and execute it with the following command:

```
nextflow run script3.nf
```

It will print an output similar to the one shown below:

```
[ggal_gut, [/../data/ggal/gut_1.fq, /../data/ggal/gut_2.fq]]
```

The above example shows how the `read_pairs_ch` channel emits tuples composed by two elements, where the first is the read pair prefix and the second is a list representing the actual files.

Try it again specifying different read files by using a glob pattern:

```
nextflow run script3.nf --reads 'data/ggal/*_{1,2}.fq'
```

Exercise 3.1

Use the set operator in place of = assignment to define the `read_pairs_ch` channel.

Exercise 3.2

Use the ifEmpty operator to check if the `read_pairs_ch` contains at least an item.

Recap

In this step you have learned:

1. How to use `fromFilePairs` to handle read pair files

2. How to use the `set` operator to define a new channel variable
3. How to use the `ifEmpty` operator to check if a channel is empty

Step 4 – Perform expression quantification

The script `script4.nf` adds the `quantification` process.

In this script note as the `index_ch` channel, declared as output in the `index` process, is now used as a channel in the input section.

Also note as the second input is declared as a `set` composed by two elements: the `pair_id` and the `reads` in order to match the structure of the items emitted by the `read_pairs_ch` channel.

Execute it by using the following command:

```
nextflow run script4.nf
```

You will see the execution of a `quantication` process.

Execute it again adding the `-resume` option as shown below:

```
nextflow run script4.nf -resume
```

The `-resume` option skips the execution of any step that has been processed in a previous execution.

Try to execute it with more read files as shown below:

```
nextflow run script4.nf -resume --reads 'data/ggal/*_{1,2}.fq'
```

You will notice that the `quantification` process is executed more than one time.

Nextflow parallelizes the execution of your pipeline simply by providing multiple input data to your script.

Exercise 4.1

Add a tag directive to the `quantification` process to provide a more readable execution log .

Exercise 4.2

Add a publishDir directive to the `quantification` process to store the process results into a directory of your choice.

Recap

In this step you have learned:

1. How to connect two processes by using the channel declarations
2. How to resume the script execution skipping already already computed steps
3. How to use the `tag` directive to provide a more readable execution output
4. How to use the `publishDir` to store a process results in a path of your choice

Step 5 – Quality control

This step implements a quality control of your input reads. The inputs are the same read pairs which are provided to the `quantification` steps

You can run it by using the following command:

```
nextflow run script5.nf -resume
```

The script will report the following error message:

```
Channel `read_pairs_ch` has been used twice as an input by process `fastqc` and
process `quantification`
```

Exercise 5.1

Modify the creation of the `read_pairs_ch` channel by using a into operator in place of a `set`.

Tip: see an example here.

Recap

In this step you have learned:

1. How to use the `into` operator to create multiple copies of the same channel

Step 6 – MultiQC report

This step collect the outputs from the `quantification` and `fastqc` steps to create a final report by using the MultiQC tool.

Execute the script with the following command:

```
nextflow run script6.nf -resume --reads 'data/ggal/*_{1,2}.fq'
```

It creates the final report in the `results` folder in the current work directory.

In this script note the use of the mix and collect operators chained together to get all the outputs of the `quantification`and `fastqc` process as a single input.

Recap

In this step you have learned:

1. How to collect many outputs to a single input with the `collect` operator
2. How to `mix` two channels in a single channel
3. How to chain two or more operators togethers

Step 7 – Handle completion event

This step shows how to execute an action when the pipeline completes the execution.

Note that Nextflow processes define the execution of *asynchronous* tasks i.e. they are not executed one after another as they are written in the pipeline script as it would happen in a common *imperative* programming language.

The script uses the `workflow.onComplete` event handler to print a confirmation message when the script completes.

Try to run it by using the following command:

```
nextflow run script7.nf -resume --reads 'data/ggal/*_{1,2}.fq'
```

Step 8 – Custom scripts

Real world pipelines use a lot of custom user scripts (BASH, R, Python, etc). Nextflow allows you to use and manage all these scripts in consistent manner. Simply put them in a directory named `bin` in the pipeline project root. They will be automatically added to the pipeline execution `PATH`.

For example, create a file named `fastqc.sh` with the following content:

```
#!/bin/bash
set -e
set -u

sample_id=${1}
reads=${2}

mkdir fastqc_${sample_id}_logs
fastqc -o fastqc_${sample_id}_logs -f fastq -q ${reads}
```

Save it, grant the execute permission and move it in the `bin` directory as shown below:

```
chmod +x fastqc.sh
mkdir -p bin
mv fastqc.sh bin
```

Then, open the `script7.nf` file and replace the `fastqc` process' script with the following code:

```
  script:
    """
    fastqc.sh "$sample_id" "$reads"
    """
```

Run it as before:

```
nextflow run script7.nf -resume --reads 'data/ggal/*_{1,2}.fq'
```

Recap

In this step you have learned:

1. How to write or use existing custom script in your Nextflow pipeline.
2. How to avoid the use of absolute paths having your scripts in the `bin/` project folder.

Step 9 – Executors

Real world genomic application can spawn the execution of thousands of jobs. In this scenario a batch scheduler is commonly used to deploy a pipeline in a computing cluster, allowing the execution of many jobs in parallel across many computing nodes.

Nextflow has built-in support for most common used batch schedulers such as Univa Grid Engine and SLURM between the others.

To run your pipeline with a batch scheduler modify the `nextflow.config` file specifying the target executor and the required computing resources if needed. For example:

```
process.executor = 'slurm'
process.queue = 'short'
process.memory = '10 GB'
process.time = '30 min'
process.cpus = 8
```

The above configuration specify the use of the SLURM batch scheduler to run the jobs spawned by your pipeline script. Then it specifies to use the `short` queue (partition), 10 gigabyte of memory and 8 cpus per job, and each job can run for no more than 30 minutes.

Note: the pipeline must be executed in a shared file system accessible to all the computing nodes.

Exercise 9.1

Print the head of the `.command.run` script generated by Nextflow in the task work directory and verify it contains the SLURM `#SBATCH` directives for the requested resources.

Exercise 9.2

Modify the configuration file to specify different resource request for the `quantification` process.

Tip: see the process documentation for an example.

Recap

In this step you have learned:

1. How to deploy a pipeline in a computing cluster.
2. How to specify different computing resources for different pipeline processes.

Step 10 – Use configuration profiles

The Nextflow configuration file can be organised in different profiles to allow the specification of separate settings depending on the target execution environment.

For the sake of this tutorial modify the `nextflow.config` as shown below:

```
profiles {
  standard {
    process.container = 'nextflow/rnaseq-nf'
    docker.enabled = true
  }

  cluster {
    process.executor = 'slurm'
    process.queue = 'short'
    process.memory = '10 GB'
    process.time = '30 min'
    process.cpus = 8
  }
}
```

The above configuration defines two profiles: `standard` and `cluster`. The name of the profile to use can be specified when running the pipeline script by using the `-profile` option. For example:

```
nextflow run script7.nf -profile cluster
```

The profile `standard` is used by default if no other profile is specified by the user.

Recap

In this step you have learned:

1. How to organise your pipeline configuration in separate profiles

Step 11 – Run a pipeline from a GitHub repository

Nextflow allows the execution of a pipeline project directly from a GitHub repository (or similar services eg. BitBucket and GitLab).

This simplifies the sharing and the deployment of complex projects and tracking changes in a consistent manner.

For the sake of this tutorial consider the example project published in the following URL:

https://github.com/nextflow-io/hello

You can run it by specifying the project name as shown below:

```
nextflow run nextflow-io/hello
```

It automatically downloads it and store in the `$HOME/.nextflow` folder.

Use the command `info` to show the project information, e.g.:

```
nextflow info nextflow-io/hello
```

Nextflow allows the execution of a specific *revision* of your project by using the `-r` command line option. For Example:

```
nextflow run -r v1.2 nextflow-io/hello
```

Revision are defined by using Git tags or branches defined in the project repository.

This allows a precise control of the changes in your project files and dependencies over time.

Step 12 (bonus) – Deposit your pipeline scripts in a GitHub repository

Create a new repository in your GitHub account and upload there the pipeline scripts of this tutorial. Then execute it specifying its name on the Nextflow command line.

Tip: see the documentation for further details.

Docker hands-on

Get practice with basic Docker commands to pull, run and build your own containers.

A container is a ready-to-run Linux environment which can be executed in an isolated manner from the hosting system. It has own copy of the file system, processes space, memory management, etc.

Containers are a Linux feature known as *Control Groups* or Ccgroups introduced with kernel 2.6.

Docker adds to this concept an handy management tool to build, run and share container images.

These images can be uploaded and published in a centralised repository know as Docker Hub, or hosted by other parties like for example Quay.

Step 1 – Run a container

Run a container is easy as using the following command:

```
docker run <container-name>
```

For example:

```
docker run hello-world
```

Step 2 – Pull a container

The pull command allows you to download a Docker image without running it. For example:

```
docker pull debian:wheezy
```

The above command download a Debian Linux image.

Step 3 – Run a container in interactive mode

Launching a BASH shell in the container allows you to operate in an interactive mode in the containerised operating system. For example:

```
docker run -it debian:wheezy bash
```

Once launched the container you wil noticed that’s running as root (!). Use the usual commands to navigate in the file system.

To exit from the container, stop the BASH session with the exit command.

Step 4 – Your first Dockerfile

Docker images are created by using a so called `Dockerfile` i.e. a simple text file containing a list of commands to be executed to assemble and configure the image with the software packages required.

In this step you will create a Docker image containing the Samtools tool.

Warning: the Docker build process automatically copies all files that are located in the current directory to the Docker daemon in order to create the image. This can take a lot of time when big/many files exist. For this reason it's important to *always* work in a directory containing only the files you really need to include in your Docker image. Alternatively you can use the `.dockerignore` file to select the path to exclude from the build.

Then use your favourite editor eg. `vim` to create a file named `Dockerfile` and copy the following content:

```
FROM debian:wheezy

MAINTAINER <your name>

RUN apt-get update && apt-get install -y curl cowsay

ENV PATH=$PATH:/usr/games/
```

When done save the file.

Step 5 – Build the image

Build the Docker image by using the following command:

```
docker build -t my-image .
```

Note: don’t miss the dot in the above command. When it completes, verify that the image has been created listing all available images:

```
docker images
```

You can try your new container by running this command:

```
docker run my-image cowsay Hello Docker!
```

Step 6 – Add a software package to the image

Add the Salmon package to the Docker image by adding to the `Dockerfile` the following snippet:

```
RUN curl -sSL https://github.com/COMBINE-
lab/salmon/releases/download/v0.8.2/Salmon-0.8.2_linux_x86_64.tar.gz | tar xz \
 && mv /Salmon-*/bin/* /usr/bin/ \
 && mv /Salmon-*/lib/* /usr/lib/
```

Save the file and build again the image with the same command as before:

```
docker build -t my-image .
```

You will notice that it creates a new Docker image with the same name *but* with a different image ID.

Step 7 – Run Salmon in the container

Check that everything is fine running Salmon in the container as shown below:

```
docker run my-image salmon --version
```

You can even launch a container in an interactive mode by using the following command:

```
docker run -it my-image bash
```

Use the `exit` command to terminate the interactive session.

Step 8 – File system mounts

Create an genome index file by running Salmon in the container.

Try to run Bowtie in the container with the following command:

```
docker run my-image \
  salmon index -t $PWD/data/ggal/transcriptome.fa -i index
```

The above command fails because Salmon cannot access the input file.

This happens because the container runs in a complete separate file system and it cannot access the hosting file system by default.

You will need to use the `--volume` command line option to mount the input file(s) eg.

```
docker run --volume $PWD/data/ggal/transcriptome.fa:/transcriptome.fa my-image \
  salmon index -t /transcriptome.fa -i index
```

An easier way is to mount a parent directory to an identical one in the container, this allows you to use the same path when running it in the container eg.

```
docker run --volume $HOME:$HOME --workdir $PWD my-image \
  salmon index -t $PWD/data/ggal/transcriptome.fa -i index
```

Step 9 – Upload the container in the Docker Hub (bonus)

Publish your container in the Docker Hub to share it with other people.

Create an account in the https://hub.docker.com web site. Then from your shell terminal run the following command, entering the user name and password you specified registering in the Hub:

```
docker login
```

Tag the image with your Docker user name account:

```
docker tag my-image <user-name>/my-image
```

Finally push it to the Docker Hub:

```
docker push <user-name>/my-image
```

After that anyone will be able to download it by using the command:

```
docker pull <user-name>/my-image
```

Note how after a pull and push operation, Docker prints the container digest number e.g.

```
Digest: sha256:aeacbd7ea1154f263cda972a96920fb228b2033544c2641476350b9317dab266
Status: Downloaded newer image for nextflow/rnaseq-nf:latest
```

This is a unique and immutable identifier that can be used to reference container image in a univocally manner. For example:

```
docker pull nextflow/rnaseq-
nf@sha256:aeacbd7ea1154f263cda972a96920fb228b2033544c2641476350b9317dab266
```

Singularity

Singularity is container runtime designed to work in HPC data center, where the usage of Docker is generally not allowed due to security constraints.

Singularity implements the container execution model similarly to Docker however using a complete different implementation design.

A Singularity container image is archived in a plain file that can be stored in shared file system and accessed by many computing nodes managed by a batch scheduler.

Create a Singularity images

Singularity images are created using a `Singularityfile` in similar manner to Docker, though using a different syntax.

```
Bootstrap: docker
From: debian:wheezy

%environment
export PATH=$PATH:/usr/games/


%labels
AUTHOR <your name>

%post

apt-get update && apt-get install -y locales-all curl cowsay
curl -sSL https://github.com/COMBINE-lab/salmon/releases/download/v0.8.2/Salmon-
0.8.2_linux_x86_64.tar.gz | tar xz \
 && mv /Salmon-*/bin/* /usr/bin/ \
 && mv /Salmon-*/lib/* /usr/lib/
```

Once you have save the `Singularity` file. Create the image with these commands:

```
singularity create my-image.img
sudo singularity bootstrap my-image.img Singularityfile
```

Singularity requires two commands to build an image, the creates the file image and allocate the required space on the storage. The second build the real container image.

Note: the `bootstrap` command requires sudo permissions.

Running a container

Once done, you can run your container with the following command

```
singularity exec my-image.img cowsay Hello Singularity
```

By using the `shell` command you can enter in the container in interactive mode. For example:

```
singularity shell my-image.img
```

Import an Docker image

An easier way to create Singularity container without requiring sudo permission and bootsting the containers interoperability is to import a Docker image container pulling it directly from a Docker registry. For example:

```
singularity pull docker://debian:wheezy
```

The above command automatically download the Debian Docker image and converts it to a Singularity image store in the current directory with the name `debian-wheezy.img`.

Run a Nextflow script using a Singularity container

Nextflow allows the transparent usage of Singularity containers as easy as with Docker ones.

It only requires to enable the use of Singularity engine in place of Docker in the Nextflow configuration file.

To run your previous script with Singularity add the following profile in the `nextflow.config` file in the `$HOME/hack17-course`directory:

```
profiles {

  foo {
    process.container = 'docker://nextflow/rnaseq-nf'
    singularity.enabled = true
    singularity.cacheDir = "$PWD"
  }
}
```

The above configuration instructs nextflow to use Singularity engine to run your script processes. The container is pulled from the Docker registry and cached in the current directory to be used for further runs.

Try to run the script as shown below:

```
nextflow run script7.nf -profile foo
```

Note: Nextflow will pull the container image automatically, it will require a few seconds depending the network connection speed.

April 12, 2019 Eric Borenstein