Chapter 3

Neuroimaging Workflows in the Cloud

Tara Madhyastha

Abstract

Analysis of large neuroimaging datasets requires scalable computing power and storage, plus methods for secure collaboration and for reproducibility. The application of cloud computing can address many of these requirements, providing a very flexible model that is generally far less expensive than a lab trying to purchase the most computer equipment they would ever need. This chapter describes how researchers can change the way that they traditionally run neuroimaging workflows in order to leverage cloud-computing capabilities. It describes various considerations and options related to cloud-based neuroimaging analyses, including cost models and architectures. Next, using data from the AOMIC-PIOP2 project hosted on Open-NEURO, it shows how to use Nextflow to create a very simple skull stripping and tissue segmentation workflow using FSL's *bet* and *fast* programs installed on a local computer. Nextflow allows scalability from a laptop to a cluster to cloud-native services with no code changes.

Key words Neuroimaging, Cloud computing, Parallelization, Virtual machines, Containers

1 Introduction

Several drivers of modern neuroimaging research demand more scalable computing power, more storage, secure collaboration, and reproducibility. Recent papers have highlighted problems with small sample sizes and the need for greater numbers of subjects in studies [1]. At the same time, the Human Connectome Project has demonstrated the importance of higher resolution data both for structural and functional analysis, through gains in alignment and in more precise connectivity analysis [2]. Because it is often difficult to recruit sufficient subjects from specialized populations at a single site, and scanner throughput is limited, these forces necessitate multisite studies and sharing of data among researchers. Finally, reproducibility is of critical importance to the field, as workflows are incredibly complex, affected by subtle differences in operating systems and software package versions, and in our ongoing learnings about how different preprocessing steps may change or bias results [3].

Robert Whelan and Hervé Lemaître (eds.), *Methods for Analyzing Large Neuroimaging Datasets*, Neuromethods, vol. 218, https://doi.org/10.1007/978-1-0716-4260-3_3, © The Author(s) 2025

Cloud computing has several characteristics that directly address these drivers. Cloud computing offers virtually unlimited resources on-demand. Using cloud computing makes it possible to address statistical problems not only by scaling to analyze a larger number of subjects and higher resolution data, but by enabling more accurate statistical methods that are infeasible on a desktop or small cluster (such as permutation-based methods for correction of multiple comparisons). Further, cloud computing makes available a wide range of processor types and system architectures. This enables algorithms that can be accelerated through commodity Graphics Processing Units (GPU) [*see* Glossary] or perhaps through Field-Programmable Gate Arrays (FPGAs) [*see* Glossary] to take advantage of this hardware just for the duration of execution, without capital investment.

Storage is also scalable. Object-based storage [*see* Glossary] is an excellent fit for secure storage of raw and preprocessed images with fine-grained access control and auditing capabilities and a global footprint; these capabilities are a foundation for any large-scale data repository.

Finally, a fundamental characteristic of cloud computing is the ability to save computing infrastructure as code. This allows researchers to relaunch not just software pipelines on a new machine, but entire computational environments together with the operating system(s) and application libraries as code. This makes it easy to reproduce computations at scale with relatively low effort and technical knowledge.

These characteristics not only meet the demands of modern neuroimaging research, but they create new possibilities to develop massively parallel algorithms and analytical tools that leverage specialized hardware. However, cloud computing has very different cost models and architectures than the traditional on-premise computing resources that have been shaping the development of neuroimaging workflows. To leverage these cloud capabilities, researchers must change the way that they traditionally run neuroimaging workflows. This chapter describes these differences, and strategies for how researchers can leverage them.

2 Cloud Fundamentals

All cloud computing providers (e.g., Google Cloud Platform, Amazon Web Services, Microsoft Azure, Oracle Cloud Infrastructure, Alibaba Cloud) share some common characteristics and infrastructure services. All of the platforms provide "infrastructure as a service", which means that you can purchase virtual computers, disks, storage, and networking and assemble them to create computing architectures that are similar to computers you can build on-premises (e.g., workstations, high-performance clusters) but with the added benefit that because these are virtual components, you can create and manipulate them programmatically. This is called "infrastructure as code". We describe some features of infrastructure as a service that are generally common across providers, and their implications. However, when working with any specific provider, it is important to understand how the details of their implementation and the features of any cloud software that you use can impact how you work.

2.1 Pay-as-You-Go

A key characteristic of cloud providers is that you pay for the time that you use the infrastructure, rather than purchasing hardware upfront as you would in an on-premise lab (although there are often ways to pay in advance or reserve computing infrastructure to accommodate the bursty nature of grant funding). There are benefits and risks to this payment model. The strongest benefit is that it means that researchers have access to vast or specialized computing resources for short durations of time. This is a very flexible model and is generally far less expensive than a lab trying to purchase the most computer equipment they would ever need. Researchers can use the computers they need as their demands change and problems take them in new directions. The risk is that there is often no link between computing and research funding, so it is easy to run out of money if you cannot easily track and bound your spend. An e-commerce website that scales up to meet holiday demand spends more money on infrastructure in proportion to additional sales; however, a researcher who is able to obtain results faster on a large cluster does not necessarily obtain proportionately more funding. Thus, a critical implication of the pay-as-you-go cost model is that it rewards efficiency, a point we will discuss in terms of implications to neuroimaging workflows.

2.2 Computing (Virtual Machines)

Computing time is available in many forms from cloud providers. A basic characteristic is that it is possible to provision many different configurations of virtual computers, with different memory, CPU (Central Processing Unit) architectures and cores, and local storage footprints. The cost of virtual computers is closely related to the memory, storage, and compute resources that they have. Therefore, while there is no penalty to running a code on a dedicated on-premise workstation that only uses one core out of 32, to do so on a virtual cloud computer would be a waste of money. On the cloud, one selects a virtual computer with just enough resources to run an application efficiently. This can be a difficult optimization problem because it means selecting a virtual computer that can complete a job (perhaps with varying data) at the lowest cost in a reasonable time. A corollary of this is that when designing workflows, from the start, one should separate out components that have significantly different computational demands. Common workflows within popular neuroimaging packages such as Analysis of Functional Neuroimages (AFNI), FreeSurfer, FMRIB Software Library (FSL), and Statistical Parametric Mapping (SPM) are usually scripts that run different programs for specific processes. Some, such as probabilistic tractography, may be highly parallelizable and benefit from GPU acceleration. Others, such as an independent components analysis, may require a large amount of memory. Finally, simple image mathematical or transformational operations may be I/O bound (input/output bound; i.e., the time constraint in the workflow is the time taken to request the data, rather than the time needed to actually process the data). When these types of processes are run in a single script on a virtual computer, the computer will need to have sufficient resources to accommodate the highest demands from any process. From a cloud cost perspective, this is wasteful, because you are (for example) paying for a GPU when it is sitting idle, or paying for additional memory when it is not being used.

Another important concept across cloud providers is that of the "spot" or "preemptable" market. This is when extra cloud capacity is provided at a substantial discount, with some caveat that it can be reclaimed if needed, or after a specific amount of time (depending on provider). When a computer is reclaimed, it shuts down and any work that has not been saved to disk is lost. This is an excellent opportunity to obtain cloud computing resources at a fraction of the regular costs, but to take advantage of this, workflows must be written to save their state periodically (or when given a reclamation warning, if available) and restart from where they have left off.

2.3 Services, Serverless, and Containers

Cloud providers also offer services that are built on top of their infrastructure platform. For example, many business applications require a database, which is relatively complicated to manage. Offering a database as a managed service means that you can take care of such things as replicating and backing up the database and patching the underlying operating system on which it runs. These services incur an additional cost over the actual infrastructure costs but save on human time (from personnel that are often hard to hire) to manage the virtual servers. It is easier to create services that represent common IT or business functions than it is to create services for researchers; many services that seem appropriate for neuroimaging workflows such as Artificial Intelligence (AI), Machine-Learning (ML) services lock one into a particular cloud vendor, may leverage proprietary algorithms, and may not be reproducible. These concepts are not as important to a business analyst who is applying a machine-learning algorithm to their sales data to predict the impact of a specific marketing campaign. The analyst does not worry about sharing or reproducing the code, and if the results are easier to get from a service, that saves time and effort. However, taken to the limit, arbitrary services can run on computing infrastructure without the user having to actually start a virtual machine, secure the operating system, and perpetually make sure it is up to date. This abstraction is called "serverless computing", and it is another important cloud abstraction that also has implications for how one designs neuroimaging workflows.

Containers [*see* Glossary] are an important information technology development that have made strong inroads into neuroimaging, and simultaneously spurred the popularity of serverless computing. A container consists of an entire runtime environment: an application, plus all its dependencies, libraries and other binaries, and configuration files needed to run it, bundled into one package. A container is similar to a virtual machine but is lighter weight—multiple containers can run on a machine and share the underlying operating system. This allows multiple packages to be run efficiently within containers on the same underlying machine, without worrying about differences in operating system distributions or dependencies. Because the details of the underlying compute infrastructure are not critical, one does not need to manage a server simply to run a container. It is sufficient merely to specify what resources (GPU, memory, cores) a container needs to execute and what broad platform (Linux, Windows) and a cloud service can run it without requiring you to provision and maintain the underlying infrastructure. An additional advantage of containers is that—unlike virtual machines—files describing their contents can be written and kept under *version control* [*see* Glossary]. This makes it possible to recreate an environment from scratch to reproduce it, rather than merely interrogating it to find out what is in it.

Containers share a key characteristic with virtual machines: the ability to package up code and dependencies into a self-contained unit. For this reason, containers have been adopted by neuroimaging researchers [4] to encapsulate entire complex workflows bound together using Python code (e.g. Nipype [5]). Two popular containers are FMRIPrep [6] and MRIQC [7] (*see* Chapter 8). The idea of using Python both to connect workflow components that may be different applications and to program novel algorithms or in-line transformations within the same piece of code is appealing and simple. However, a major problem with this design is that different stages in such workflows require different computing resources, and so cannot take advantage of highly efficient cloud-based container services to execute them. In contrast, bioinformatics workflows often consist of smaller discrete containerized applications that are bound together with a dedicated workflow description language. Each step of the workflow can be accompanied by a specification of what resources are needed to run it. These workflow characteristics make it possible to cost-efficiently use many cloud services as well as on-premise resources, and this cross-discipline experience is starting to have an impact on neuroimaging workflows.

2.4 Security and Collaboration

An area where cloud architectures excel is in enabling global collaboration. Instead of copying data from one site to another, it is possible to architect secure research environments where researchers can come to access data and compute on it. Cloud providers can certify that their services adhere to compliance standards (such as Federal Risk and Authorization Management Program, General Data Protection Regulation) and architectures built on these services inherit these controls.

Depending on the sensitivity of the data, it can be controlled in many ways, ranging from restricting or enforcing access and maintaining an audit trail, to limiting exactly what flows in and out of a secure environment. The ability to protect data while maintaining environments where researchers anywhere can collaborate as though they were in the same lab is an enormous strength of cloud computing. Large datasets remain and are secured in one place while researchers come to the data to work on it. In particular, cloud computing enables large-scale collaborative studies and fine-grained control over dissemination of data with different levels of protected health information.

In most infrastructure-as-service platforms, security of data and the infrastructure is up to the user to configure. In a research context, this configuration would typically be designed and provided by research IT staff, leveraging third party products to avoid reinventing infrastructure where sensible.

2.5 Architectural Considerations

Cloud abstractions have been developed to serve business applications, which have different characteristics than research applications. For example, business applications are often critical; if an e-commerce website failed it could result in huge loss of revenue and damage to the company reputation. In contrast, if a research pipeline fails sporadically, it can generally be restarted without serious repercussions. Designing for high availability means creating architectures that build in failover (i.e., switching to a redundant system) to multiple independent data centers. They must also be elastic, so that if a single compute element becomes overwhelmed, it can be replicated to accommodate the bursty load. Often these characteristics are not important in a neuroimaging research context.

Cloud infrastructure components have specific service level agreements: *availability* (can you get to them), *reliability* (do they work correctly), and *durability* (is your data intact). These are normally much higher than what can be provided by on-premise components. For example, the default S3 object storage on AWS will maintain several copies of your data in distinct data centers that are unlikely to fail for the same cause unless there is a large geographic disaster. This will probably be safer than network attached storage in a lab machine room, so your backup plan may look different. It is important to reassess strategies for maintaining availability and durability of your workflows and data in the context of the cloud.

3 Where Traditional Neuroimaging Workflows Fall Short

There are a variety of packages commonly used in neuroimaging, such as AFNI, FSL, FreeSurfer, SPM, Advanced Normalization Tools (ANTs) [8], HCP Informatics Infrastructure [9], that contain programs and graphical user interfaces (GUI) [*see* Glossary] to make it possible to process neuroimaging data. There are many steps involved in taking structural or functional imaging data from raw DICOM (digital imaging and communications in medicine) format to processed images and group results; the details of these steps are not the topic of this paper. However, our main focus in the context of cloud computing is how they are connected together.

There are different schools of thought on the flexibility one should have to "mix and match" different algorithms (embodied in programs) from different packages. Some neuroimaging researchers prefer to stick to workflows created within a package, to make sure that subtle differences in how individual programs work do not cause errors. In this case, the different programs are typically connected using scripts provided by the packages that govern their execution. Another school of thought suggests that some packages are better at some capabilities than others, so they mix and match programs from different packages in a single workflow.

Regardless of which approach is used, one characteristic of neuroimaging workflows is that they can take advantage of parallelism, often at multiple levels. Different brains can be preprocessed at the same time (for example, on different cores within a cluster). This is called *coarse-grained parallelism* [*see* Glossary]. Often analysis of a single image can be parallelized (for example, probabilistic tractography lends itself well to fine-grain parallelism). At the level of course-grained parallelism, several approaches have been used to take advantage of multiple cores or a cluster where available. Several workflows (e.g., many scripts involving multiple tools in FSL, FreeSurfer) can automatically submit independent jobs to a cluster where available. Some tools can readily take advantage of multiple cores where available. Although parallelism may exist under the hood, the general approach is to abstract that from users within a single package so that they do not need to worry about how their code is being parallelized. Figure 1 shows an illustration of how this may be accomplished. To the user, processing appears to be conducted by a single tool that executes on a traditional computer architecture (a single node, or a cluster).

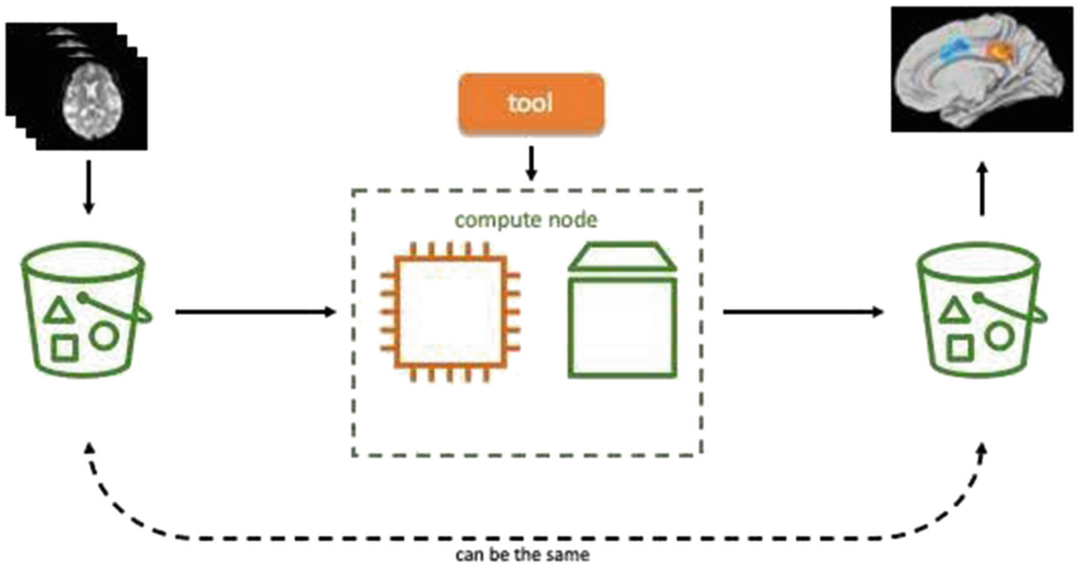

Fig. 1 A single tool approach to cloud-based workflow

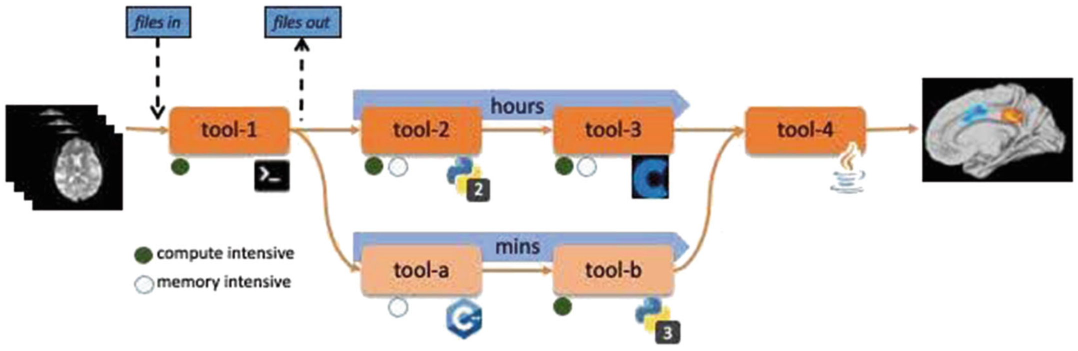

Fig. 2 An example of "mix and match" workflow

Another approach to take advantage of parallelism is used more in the "mix and match" scenario. It is common to express the course-grained parallelism of a workflow as a directed acyclic graph (DAG) [*see* Glossary], with dependencies clearly spelled out. This tells an execution engine which programs need to be run before others. Where dependencies do not exist between programs, they can all be run simultaneously. A dependency graph can also provide information about conditions for successful step completion, so if the workflow is interrupted, it can be restarted without replication of work. A typical sequential programming script does not express this dependency information, and so is an inefficient way to structure a workflow from a performance perspective. Figure 2 shows an example of a "mix and match" workflow. Each tool has different software and hardware requirements, and some steps take minutes and some take hours. However, the dependencies between tools and their requirements (even if they are all from the same neuroimaging package and designed to work together) is articulated so that they can automatically be parallelized.

Two programs that have been used to describe parallel workflows are GNU Make [10] and Nipype [5] [*see* Resources]. GNU Make has the advantage that it is a robust and relatively simple tool and can be used to connect scripts written in multiple different languages. Its main disadvantage is that it is well grounded in traditional UNIX conventions (such as files, standard input and output), and has its own unique syntax. For this reason, it is difficult to describe dependencies on very complex output directory structures, and researchers need to learn Make syntax. Nipype has gained a lot of traction in the neuroimaging community as a Python-based library that can be used to express dependencies among neuroimaging workflows, and is used by several other packages. It has the strong advantage that researchers can work exclusively in Python, and the Nipype wrappers handle the complexity of neuroimaging outputs. The disadvantage is that components need to be wrapped and as an open-source project, changes can cause problems with backward compatibility.

Both GNU Make and Nipype approaches fail to leverage modern aspects of cloud computing. To take full advantage of serverless and elastic computing, we would want to be able to separate the compute needs of each component that makes up a workflow, and express dependencies among them. We would want to be able to execute containers as well as code. To take advantage of cloud storage we would want to be able to natively indicate that our input and output data live on object storage in the cloud. And to avoid lock-in to a specific cloud platform or architecture, we would want to be able to run the same workflow on on-premise architectures and multiple cloud platforms.

4 Futureproofing Neuroimaging Workflows

To leverage the capabilities of cloud computing in neuroimaging workflows, enabling scaling to larger, more compute-intensive datasets, we need to rethink how to write workflows. Below are some basic principles and how to implement them.

4.1 Portability and Reproducibility

Researchers will have access to different computing resources at different points in their careers, and so must not be locked into a specific cloud platform or architecture. Moreover, their colleagues, who may need to be able to reproduce their work, cannot be expected to have access to the same platforms or architectures.

Note
To improve portability and reliability of neuroimaging workflows. First, avoid using proprietary algorithms embedded in cloud services (e.g., AI/ML services) that impact research results. At best, these can lock researchers into a specific vendor that they or their colleagues lose access to in the future. At worst, services can change or be retired, making it impossible to reproduce results. Second, stick to the "lowest common denominator" of services when using cloud computing for running workflows. For example, virtual machines, object storage, and container execution platforms are available on all platforms and have on-premise analogs. Failure to do so makes it difficult or impossible to change platforms.

4.2 Workflow and Serverless

To take advantage of serverless cloud computing and spot market offerings, it is critical to structure your applications from the start to be parallelizable, architecture-aware, and fault tolerant.

1. Separate time-consuming and resource-intensive components from other parts of the workflow. Distribute these components as containers so that they can be run unmodified on cloud services.
2. Create workflows from components within the same package or across packages by using a cloud-native workflow description language that can be easily ported to on-premises clusters and cloud-native architectures across multiple platforms. As cloud computing grows more prevalent, more software for workflows (e.g., Nextflow, snakemake [*see* Resources]) is written to make code portable across platforms.
3. Avoid loops to process multiple subjects, and instead use a workflow description language, parallel job submission, or multiple cores (e.g., via GNU parallel [11]) to run them.
4. Write long-running applications so that they checkpoint their work and can resume upon interruption. Choose a workflow description language that permits resuming a workflow after it has been interrupted or when a step needs to be modified.

4.3 Data Management

Storage options in the cloud are incredibly powerful but form a model that is more complex than most on-premises storage systems and cost models. Cloud object storage is scalable and highly reliable. However, pricing for object storage is typically based on the size of the data stored, the storage tier (how readily accessible and/or available is the data), and access charges. There are also potentially charges for data egress from the cloud or between regions. Entire machines can be saved along with everything necessary to create an analysis. Finally, object storage is not suitable as a file system; to use data on object storage one needs to stage it to a file system. To use these features cost-effectively requires thinking about data management at the start of an analysis.

1. Save virtual machines, containers, code and data together at the end of a completed analysis so that you can reproduce the analysis.
2. Use directory and file naming conventions consistently so that you can create automatic rules for creating versions of objects for backup, deleting versions of objects, moving objects to less expensive tiers of storage, or archiving objects.
3. Automatically migrate important objects to lower-cost archival storage when appropriate.
4. Save data products for completed analyses when the cost to store them for an appropriate timeframe is less than the cost to reproduce them, and delete them otherwise.
5. Write workflows to copy data from object storage to file system storage and write back results to minimize the size of a working disk.

5 Step-by-Step Example: Nextflow and AWS

Nextflow is a workflow description language that has these characteristics and is the basis of Tractoflow [12] and several other workflows from the Sherbrooke Connectivity Imaging lab (https://scil.usherbrooke.ca). Nextflow is well-established in bioinformatics workflows, which share a lot in common with neuroimaging workflows, but because of larger data sizes and sharing requirements have migrated to the cloud earlier. Key characteristics of Nextflow are the ability to take advantage of cloud native storage and batch computing services to execute containers, an extremely flexible language to describe expected inputs and outputs, and the ability to configure multiple engines. A language such as Nextflow allows scalability from a laptop to a cluster to cloud native services with no code changes. Other bioinformatics workflow systems such as snakemake [13] and WDL/CWL [14] share similar characteristics with Nextflow, and there is some effort to introduce these into the neuroimaging community. AWS is a major cloud provider, and may be familiar to neuroimagers because data from the Human Connectome Project and OpenNeuro are stored on AWS S3 object storage.

In this example, we use data from the AOMIC-PIOP2 project [15] hosted on OpenNEURO (https://openneuro.org/datasets/ds002790) and Nextflow to create a very simple skull stripping and tissue segmentation workflow using FSL's bet and fast programs installed on a local computer. We will stage data from where it is stored on AWS S3 object storage.

To follow along with this example, you will need a Linux or MacOS terminal environment (commands assume bash), and you will need to have FSL installed (if you do not use FSL, feel free to substitute any other simple neuroimaging commands that you prefer).

5.1 Installing Nextflow

Installation of Nextflow is very simple (see directions for most recent information). You must have a recent version of Java installed, which you can check by typing:

```
java -version
```

If you do not have Java installed then click here (https://www.java.com/en/download/help/download_options.html) for more installation instructions.

Then, install the Nextflow software:

```
curl -s https://get.nextflow.io | bash
```

This will create the executable program called nextflow in your current working directory. You can test that this program works by running a canned workflow.

```
./nextflow run hello
```

If everything works, move the nextflow program to your ~/bin directory and add this directory to your path in your .bashrc so it will be there every time you log in:

```
mkdir -p ~/bin
mv nextflow ~/bin
cat << EOF >> ~/.bashrc
export PATH=$PATH:~/bin/nextflow
EOF
source ~/.bashrc
```

Note
Create a configuration file. The files in the AOMIC-PIOP2 repository are on S3, and require no specific permissions, but if you have not configured your environment with valid AWS credentials, you will obtain an error when Nextflow attempts

(continued)

to stage the S3 files locally. To get around this, you can create a file called nextflow.config with the following contents.

```
aws {
 client {
 anonymous='true'
 }
 }
```

This tells Nextflow that the request will not be authenticated, so you do not need any credentials.

5.2 Create a Small Workflow

Add the following code in a text file: *script.nf.*

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

nextflow.enable.dsl=1

t1 = Channel.of(["0001_bet.nii.gz", "s3://openneuro.org/
ds002790/sub-0001/anat/sub-0001_T1w.nii.gz"], ["0002_bet.nii.
gz", "s3://openneuro.org/ds002790/sub-0002/anat/sub-0002_T1w.
nii.gz"])

/* perform skull stripping */
process skull_strip {
 input:
 tuple val(bet), path(t1) from t1

 output:
 path(bet) into betout
"""
/home/ubuntu/fsl/bin/bet $t1 $bet
"""
}

process fast {
 input:
 path bet from betout

 output:
 path '*_bet_*' into fastout

 """
 /home/ubuntu/fsl/bin/fast $bet
 """
}

fastout
.flatMap()
 .subscribe{ println "File: ${it.name}" }
```

This workflow has two processes. The first, skull_strip executes the bet skull stripping command, and the second executes the fast tissue segmentation command. The execution order is determined by the inputs and outputs. The skull_strip process takes inputs from a set of tuples (anatomical T1-weighted files on S3, and the friendly name we would like to give to the skull stripped images). The second, fast, executes the fast tissue parcellation command using the output from skull_strip.

We have specified multiple files as input. Unlike a script, each process runs independently and in a separate working directory. This enables us to run all the processes as quickly as possible—as soon as prerequisites have completed—without worrying about files with the same name being overwritten. Nevertheless, here we show how we can pass in friendly file names to help keep our outputs straight.

5.3 Run the Workflow

To run the workflow, type:

```
nextflow run script.nf
```

This command will run the workflow locally, which is great for testing, and publish the output in the directory work. Note that each process is stored separately, so file names do not conflict with each other. There are many settings you can use to move output to other directories.

5.4 To Infinity: Going Cloud Native

A workflow system such as Nextflow covers the basics of portable neuroimaging workflow that can scale. By defining processes separately from each other and clearly specifying the parallel structure of the steps and the input data, you now have the capability to run at scale and use scalable object storage effectively. This is the most important precursor to neuroimaging in the cloud. When resources are limited, there is little to be gained by structuring your workflow in this way; a script that cannot exploit parallel computing will be slow if there is no extra capacity to be had. This will allow you to effectively use even an autoscaling cluster in the cloud, or a multi-core server. To leverage serverless computing and optimize your use of resources further, you will need to replace each process with a container and describe the compute resources necessary to run each step. With this work done, you can move to a serverless container-based platform.

References

1. Marek S, Tervo-Clemmens B, Calabro FJ, Montez DF, Kay BP, Hatoum AS, Donohue MR, Foran W, Miller RL, Hendrickson TJ, Malone SM, Kandala S, Feczko E, Miranda-Dominguez O, Graham AM, Earl EA, Perrone AJ, Cordova M, Doyle O, Moore LA, Conan GM, Uriarte J, Snider K, Lynch BJ, Wilgenbusch JC, Pengo T, Tam A, Chen J, Newbold DJ, Zheng A, Seider NA, Van AN, Metoki A, Chauvin RJ, Laumann TO, Greene DJ, Petersen SE, Garavan H, Thompson WK, Nichols TE, Yeo BTT, Barch DM, Luna B, Fair DA, Dosenbach NUF (2022) Reproducible brain-wide association studies require thousands of individuals. Nature 603:654–660. https://doi.org/10.1038/s41586-022-04492-9
2. Elam JS, Glasser MF, Harms MP, Sotiropoulos SN, Andersson JLR, Burgess GC, Curtiss SW, Oostenveld R, Larson-Prior LJ, Schoffelen J-M, Hodge MR, Cler EA, Marcus DM, Barch DM, Yacoub E, Smith SM, Ugurbil K, Van Essen DC (2021) The human connectome project: a retrospective. NeuroImage 244: 118543. https://doi.org/10.1016/j.neuroimage.2021.118543
3. Poldrack RA, Baker CI, Durnez J, Gorgolewski KJ, Matthews PM, Munafò MR, Nichols TE, Poline J-B, Vul E, Yarkoni T (2017) Scanning the horizon: towards transparent and reproducible neuroimaging research. Nat Rev Neurosci 18:115–126. https://doi.org/10.1038/nrn.2016.167
4. Gorgolewski KJ, Auer T, Calhoun VD, Craddock RC, Das S, Duff EP, Flandin G, Ghosh SS, Glatard T, Halchenko YO, Handwerker DA, Hanke M, Keator D, Li X, Michael Z, Maumet C, Nichols BN, Nichols TE, Pellman J, Poline J-B, Rokem A, Schaefer G, Sochat V, Triplett W, Turner JA, Varoquaux G, Poldrack RA (2016) The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments. Sci Data 3:160044. https://doi.org/10.1038/sdata.2016.44
5. Gorgolewski K, Burns CD, Madison C, Clark D, Halchenko YO, Waskom ML, Ghosh SS (2011) Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in python. Front Neuroinformatics 5. https://doi.org/10.3389/fninf.2011.00013
6. Esteban O, Markiewicz CJ, Blair RW, Moodie CA, Isik AI, Erramuzpe A, Kent JD, Goncalves M, DuPre E, Snyder M, Oya H, Ghosh SS, Wright J, Durnez J, Poldrack RA, Gorgolewski KJ (2019) fMRIPrep: a robust preprocessing pipeline for functional MRI. Nat Methods 16:111–116. https://doi.org/10.1038/s41592-018-0235-4
7. Esteban O, Birman D, Schaer M, Koyejo OO, Poldrack RA, Gorgolewski KJ (2017) MRIQC: advancing the automatic prediction of image quality in MRI from unseen sites. PLOS ONE 12:e0184661. https://doi.org/10.1371/journal.pone.0184661
8. Avants B, Tustison NJ, Song G (2009) Advanced normalization tools: V1.0. Insight J. https://doi.org/10.54294/uvnhin
9. Marcus D, Harwell J, Olsen T, Hodge M, Glasser M, Prior F, Jenkinson M, Laumann T, Curtiss S, Van Essen D (2011) Informatics and data mining tools and strategies for the human connectome project. Front Neuroinformatics 5. https://doi.org/10.3389/fninf.2011.00004
10. Askren MK, McAllister-Day TK, Koh N, Mestre Z, Dines JN, Korman BA, Melhorn SJ, Peterson DJ, Peverill M, Qin X, Rane SD, Reilly MA, Reiter MA, Sambrook KA, Woelfer KA, Grabowski TJ, Madhyastha TM (2016) Using make for reproducible and parallel neuroimaging workflow and quality-assurance. Front Neuroinform 10:2. https://doi.org/10.3389/fninf.2016.00002
11. Tange O (2018) GNU parallel 2018. Ole Tange
12. Theaud G, Houde J-C, Boré A, Rheault F, Morency F, Descoteaux M (2020) TractoFlow: a robust, efficient and reproducible diffusion MRI pipeline leveraging Nextflow & Singularity. NeuroImage 218:116889. https://doi.org/10.1016/j.neuroimage.2020.116889
13. Mölder F, Jablonski KP, Letcher B, Hall MB, Tomkins-Tinch CH, Sochat V, Forster J, Lee S, Twardziok SO, Kanitz A, Wilm A, Holtgrewe M, Rahmann S, Nahnsen S, Köster J (2021) Sustainable data analysis with Snakemake. F1000Research 10:33. https://doi.org/10.12688/f1000research.29032.2
14. Amstutz P, Crusoe MR, Tijanić N, Chapman B, Chilton J, Heuer M, Kartashov A, Leehr D, Ménager H, Nedeljkovich M, Scales M, Soiland-Reyes S, Stojanovic L (2016) Common workflow language, v1.0. 5921760 Bytes
15. Snoek L, van der Miesen MM, Beemsterboer T, van der Leij A, Eigenhuis A, Steven Scholte H (2021) The Amsterdam Open MRI Collection, a set of multimodal MRI datasets for individual difference analyses. Sci Data 8:85. https://doi.org/10.1038/s41597-021-00870-6