Approaches for containerized scientific workflows in cloud environments with applications in life science

Ola Spjuth[1,*], Marco Capuccini[1,2], Matteo Carone[1], Anders Larsson[3], Wesley Schaal[1], Jon Ander Novella[1], Oliver Stein[1], Morgan Ekmefjord[1], Paolo Di Tommaso[4], Evan Floden[4], Cedric Notredame[4], Pablo Moreno[5], Payam Emami Khoonsari[6], Stephanie Herman[1,6], Kim Kultima[6], and Samuel Lampa[1]

[1]Department of Pharmaceutical Biosciences and Science for Life Laboratory, Uppsala University, Sweden
[2]Department of Information Technology, Uppsala University, Sweden
[3]National Bioinformatics Infrastructure Sweden, Uppsala University, Sweden
[4]Centre for Genomic Regulation (CRG), The Barcelona Institute for Science and Technology, Barcelona, Spain
[5]European Molecular Biology Laboratory, European Bioinformatics Institute (EMBL-EBI), Cambridge, United Kingdom
[6]Department of Medical Sciences, Clinical Chemistry, Uppsala University, Sweden
[*]Corresponding author: ola.spjuth@farmbio.uu.se

January 30, 2020

Abstract

Containers are gaining popularity in life science research as they provide a solution for encompassing dependencies of provisioned tools, simplify software installations for end users and offer a form of isolation between processes. Scientific workflows are ideal for chaining containers into data analysis pipelines to aid in creating reproducible analyses. In this manuscript we review a number of approaches to using containers as implemented in the workflow tools Nextflow, Galaxy, Pachyderm, Argo, Kubeflow, Luigi and SciPipe, when deployed in cloud environments. A particular focus is placed on the workflow tool's interaction with the Kubernetes container orchestration framework.

1 Introduction

The life sciences have become data-intensive, driven largely by the massive increase in throughput and resolution of molecular data generating technologies. Massively parallel sequencing (also known as next-generation sequencing or NGS) is where the largest increase has been seen in recent years, but other domains are also increasing dramatically, including proteomics, metabolomics, systems biology and biological imaging [1, 2, 3]. Consequently, the need for computational and storage resources has continued to grow, but the focus has also changed towards the downstream steps of biological experiments and the need to carry out efficient and reproducible data analysis. While high-performance computing (HPC) and high-throughput computing (HTC) clusters remain the main e-infrastructure resource used in biological data analysis, cloud computing is emerging as an appealing alternative where, in the infrastructure-as-a-service (IaaS) case, scientists are able to spawn virtual instances or infrastructure on demand to facilitate their analysis, after which the resources are released [4, 5]. One area which has traditionally often caused headaches for users is the installation of software tools and their inclusion into workflow tools or other computing frameworks [6]. This is an area where cloud resources provide an advantage over HPC, in that scientists are not dependent on system administrators to install software, but can handle the installation themselves. However, soft-

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 31 January 2020 doi:10.20944/preprints202001.0378.v1

ware for biological analyses can be quite challenging to install due to sometimes complex dependencies. Virtual machines (VMs) offer the benefit of instantiating ready-made environments with data and software, including all dependencies for specific tasks or analyses and constitute a big step towards making computational reproducibility easier and more realistic to achieve in daily work. VMs form the backbone of traditional cloud-based infrastructures. Virtual machine images (VMIs), however, can easily become large and take considerable time to instantiate, transfer or rebuild when software in the image needs to be updated. Such images can easily be shared between users and can be deposited and indexed by one of the available VMI catalogs [7].

1.1 Containers

Software container technology has emerged as a complement to virtual machines. While they offer a bit less isolation between processes, they are on the other hand more lightweight and easy to share; the operating system kernel makes them much faster to launch and terminate [8]. Containers encompass all dependencies of the provisioned tools and greatly simplify software installations for end users. The most widely used containerization solution is Docker (www.docker.com), but Singularity [9], uDocker [10], and Shifter [11] are recent alternatives that prevent users from running containers with root privileges, addressing the most common security issues when deploying containers in multi-tenant computing clusters such as on high-performance computing (HPC) clusters. Docker containers are commonly shared via Docker Hub [12]. There are also initiatives for standardizing containers in the life sciences such as BioContainers [13]. Containers have seen an increased uptake in the life sciences, both for delivering software tools and for facilitating data analysis in various ways [14, 15, 16, 17, 18].

When running more than just a few containers, an orchestration system is needed to coordinate and manage their execution and handle issues related to e.g. load balancing, health checks and scaling. Kubernetes [19] has over the last couple of years rose to become the de facto standard container orchestration system, but Docker Swarm [20] and Apache Mesos [21] are other well-known systems. A key objective for these systems is to allow the users to treat a cluster of compute nodes as a single deployment target and handle the packaging of containers on compute nodes behind the scenes.

1.2 Scientific workflows

While orchestration tools such as Kubernetes enable scientists to run many containers in a distributed environment (such as a virtual infrastructure on a cloud provider), the problem remains how to orchestrate the scheduling and dependencies between containers and being able to chain (define data dependencies between) them into an analysis pipeline. This is where scientific workflow management systems (WMSs) can be very useful; in fact, it can be difficult to carry out more complex analyses without such a system. Traditionally in data-intensive bioinformatics, an important task for WMSs has been to support analyses consisting of several components by running a set of command-line tools in a sequential fashion, commonly on a computer cluster. The main benefits of using a WMS include: i) making automation of multi-step computations easier to create, more robust and easy to change ii) providing more transparency as to what the pipeline does through its more high-level workflow description and better reporting and visualization facilities, and iii) providing more reliable policies to handle transient or persistent error conditions, including strategies to recover from interrupted computations while re-using any partially finished data.

Using containerized components as the nodes in a scientific workflow has the advantage of adding isolation between processes and the complete encapsulation of tool dependencies within a container, and reduces the burden to administrate the host system where the workflow is scheduled to run. This means the system can be tested on a local computer and executed on remote servers and clusters without modification, using the exact same containers. Naturally, this opens up for execution on virtual infrastructures, even those provisioned on-demand with features such as auto-scaling. Apart from greatly simplifying access to the needed distributed compute resources, a positive side-effect is that the entire analysis becomes portable and reproducible.

Data ingestion and access to reference data con-

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 31 January 2020 doi:10.20944/preprints202001.0378.v1

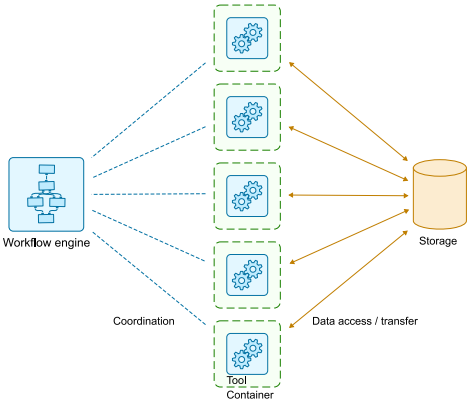

Figure 1: Overview of a containerized workflow with a workflow engine managing the dependency graph and scheduling of the containers encompassing the tools to run. The tools pass data between each other by reading and writing to a shared storage system.

stitute a key step when using cloud-based resources. When spawning a virtual infrastructure, data used in the analysis either needs to be available on storage connected to the compute nodes, or ingested into a locally provisioned file system. At the end of the analysis, the resulting data needs to either be put into persistent storage on the cloud or downloaded to local storage. However, the overhead with this can also be turned around. If data resides on storage connected to a cloud infrastructure, it is possible to simply move the workflow and containers there and "bring compute to the data".

2 Workflow systems

Due to their simplicity, containers are supported out of the box by most workflow tools. However, the approach for interacting with containers differs, especially when using virtual infrastructures on cloud resources. This section describes the approaches taken by a set of workflow management systems to interacting with containers in cloud environments, with varying degree of uptake in the life science community.

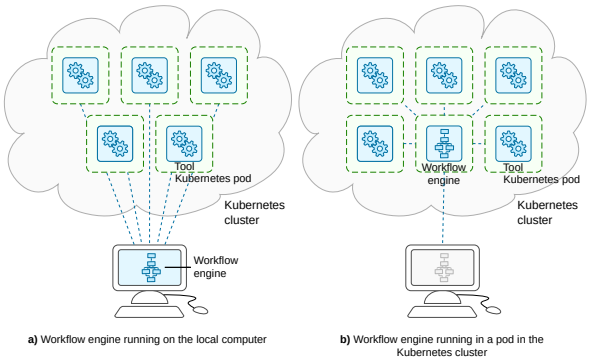

Figure 2: Comparison between two approaches for how to run the workflow engine. It can run either a) on the user's computer, or b) in a pod in the kubernetes (abbreviated "k8s" in the image) cluster.

2.1 Nextflow

Nextflow [22] is a workflow framework that aims to ease the implementation of scientific workflows in a portable and reproducible manner across heterogeneous computing platforms and to enable the smooth migration of these applications to the cloud. The framework is based on the dataflow paradigm, a functional/reactive programming model in which tasks are isolated from each other and are expected to be executed in a stateless manner. This paradigm simplifies the deployment of complex workflows in distributed computing environments and matches the immutable execution model of containers.

Nextflow provides built-in support for the most used container runtimes i.e. Docker, Singularity, Shifter and uDocker (though the last two are undocumented because of still being in an experimental state). The use of containers is defined in a declarative manner i.e. by specifying the container image that a workflow needs to use for the task executions. Once annotated in this way, the Nextflow runtime takes care of transparently running each task in its own container instance while mounting the required input and output files as needed. This approach allows a user that needs to target the requirements of a specific execution platform to quickly switch from one containerization technology to another with just a few changes in the workflow configuration file (e.g. the same application can be deployed in an HPC cluster using

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 31 January 2020 doi:10.20944/preprints202001.0378.v1

Singularity or in the cloud with Docker).

Great flexibility is given for container image configuration. Nextflow supports both the image-per-workflow pattern (the same image for all executed tasks) and the image-per-task pattern (each task uses a different container image). The container image used by a task can even be specified at runtime by a given input configuration.

Nextflow allows the deployment of containerized workloads in a cloud-native manner using different strategies, as briefly discussed in the following paragraphs.

Cloud-managed computing service: in this approach workflow tasks are submitted to a cloud provider batch execution service. The containerization and auto-scaling are delegated to the cloud service. The workflow inputs/outputs and intermediate result files need to be stored using the provider object storage service. Nextflow adapts the task definition, resource requests and container image to the cloud provider format, submitting the corresponding API requests and properly staging the task data. Currently, Nextflow has built-in support for the AWS Batch service and the Google Genomics Pipelines service.

Cloud unmanaged computing service: when using this deployment strategy Nextflow takes care of the provisioning of a set of cloud VM instances in which it automatically sets up its own clustering engine for the distributed execution of the workflow. The VM instances only require the availability of a Docker runtime and the Java virtual machine. Workflow tasks exchange data via a shared file system or by using a cloud object storage service. The workflow execution is managed by Nextflow which also handles the cluster auto-scaling i.e. adds or removes VM instances on-demand to adapt the actual needs of a workload at any point in time and optimizes the overall computing cost when applicable. At the time of writing, this feature supports the use of the AWS EC2 cloud service and Google Compute Engine service.

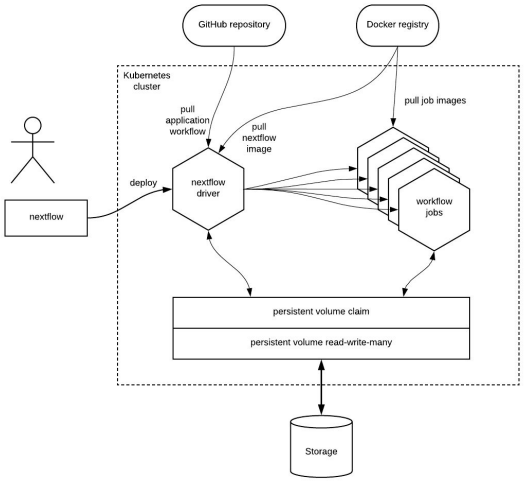

Figure 3: Overview of a containerized workflow with a workflow engine managing the dependency graph and scheduling of the containers encompassing the tools to run. The tools pass data between each other by reading and writing to a shared storage system.

2.2 Galaxy

Galaxy (Galaxy, RRID:SCR_006281) is a data-analysis workflow platform that has probably the most active development community in the field of workflow environments in Bioinformatics. This has enabled it to persist as an active project and to continue to evolve since 2005, enabling reproducible research through active channels for sharing tools, workflows and datasets [23]. Galaxy is currently used by tens of thousands of scientists across the globe [24]. Besides offering a modern user interface (UI) for end users, Galaxy is accessible through its REST API or through the Bioblend Python binding (that talks to this REST API).

For execution versatility, Galaxy has adapters to offload jobs in a wide variety of systems, from local Docker containers to different batch systems. In Galaxy, there is a complete separation of concerns between tools, workflows and execution environments, meaning that a workflow definition is not bound to a particular execution environment. Within the last five years Galaxy gained the ability to launch on Amazon AWS through the CloudMan Launcher Galaxy sub-project [25]. This setup follows a relative VM centric approach, which has natural limitations for scaling in the number of tools included as they all need to be provisioned inside the VM used for the provisioned cluster.

More recently, through efforts within the PhenoMeNal H2020 Project [26], Galaxy gained the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 31 January 2020 doi:10.20944/preprints202001.0378.v1

ability to deploy inside Kubernetes in an automated fashion through the use of Helm Charts [27]. The Galaxy-stable Helm chart makes use of community maintained Galaxy container images (Docker-Galaxy-stable), which are used on other container orchestrators, and include support for SSH file transfer protocol (SFTP) access, database backends and even deployment/compatibility of HPC scheduling systems like Condor or SLURM for hybrid cloud setups. The Kubernetes job runner, also contributed to the Galaxy community by the PhenoMeNal H2020 Project, allows Galaxy to offload jobs to Kubernetes, either when Galaxy runs inside the container orchestrator or from the outside (provided that the shared file system is accessible to both the container orchestrator and Galaxy). The Kubernetes runner for Galaxy will take care of resource requirement settings, recovery of jobs in Kubernetes if there is a restart, auto-upgrading of resource utilization, Persistent Volume Claim mounting both for Galaxy and in each job's pod, as well as adequate supplementary groups handling for network filesystem (NFS) and other shared file systems. This integration requires the use of a shared file system, which needs to be mountable in Kubernetes in a read-write-many configuration, as both the main Galaxy pod and the job pods need to mount it at the same time (through the PV/PVC abstraction). The Kubernetes runner for Galaxy adheres to the principles of failsafeness and redundancy commonly considered in cloud computing, providing ways to recover from jobs being lost by cluster nodes going down in the middle of an execution.

The Kubernetes runner for Galaxy can benefit from explicit tool-to-containers mapping through Galaxy dynamic destinations, or take advantage of dynamic bioconda-to-containers mapping thanks to built-in functionality in Galaxy for this purpose. This means that if a Galaxy tool declares a Bioconda package as the dependency to be resolved, it will bring an automatically built container for that Bioconda package [28] from BioContainers [13].

The Galaxy-Kubernetes integration, which includes all the aspects related to Kubernetes mentioned in the previous paragraphs, has been battle tested by the PhenoMeNal H2020 Project, having its public instance received in the order of tens of thousands of jobs during the past 2 years, having tens of deployments in production environments and possibly hundreds of deployments in development settings. This integration has been tested locally on Minikube, on OpenStack deployments, on Google Cloud Platform (GCP) and Amazon Web Services (AWS). It is expected that in the next few years the Galaxy-Kubernetes integration is embraced in the areas of Proteomics and Transcriptomics. The CloudMan Galaxy launcher is being considered to be migrated to a more containerized deployment scheme based on the Galaxy-Kubernetes setup.

2.3 Pachyderm

Pachyderm [29] is a large-scale data processing tool built natively on top of Kubernetes. This platform offers: i) a data management system based on Git semantics and ii) a workflow system for creating distributed and reproducible pipelines based on application containers. In order to create workflows with Pachyderm, users simply need to supply a JSON pipeline specification including a Docker image, an entrypoint command to execute in the user containers and one or more data input(s). Afterwards, Pachyderm will ensure that the corresponding pods are created in Kubernetes, and will share the input data across them and collect the corresponding outputs. Thanks to using Kubernetes for container orchestration, Pachyderm is able to, among other things: i) optimize cluster resource utilization, ii) run seamlessly on multiple cloud and on-premise environments, iii) self-heal pipeline jobs and related resources.

This workflow tool provides an easy mechanism to distribute computations over a collection of containers. In other words, it offers similar capabilities to frameworks such as Apache Spark, but replaces MapReduce syntax with legacy code. More specifically, Pachyderm is able to distribute workloads over collections of containers by partitioning the data into minimal data units of computation called "datums". The contents of these datums are defined by glob patterns such as "/" or "/*", which instruct Pachyderm how to process the input data: all together as a single datum, or in isolation (as separate datums). Users can define the number of pipeline workers to process these datums, which will be processed one at a time in isolation. When the datums have been successfully processed the Pachyderm daemon gathers the results correspond-

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 31 January 2020 doi:10.20944/preprints202001.0378.v1

ing to each datum, combines them and versions the complete output of the pipeline. The scheduling of pipeline workers is determined by the resource utilization status of the workers nodes and the resource requests that are made for each pipeline, allowing for efficiently distributing workloads. Recently, work to integrate and demonstrate Pachyderm in bioinformatics was published [30].

2.4 Luigi

Luigi [31] is an open-source Python module for defining and running batch-like workflows. The system is strongly oriented towards Big Data, and it comes as a convenience tool to pipe jobs from well-established analytics frameworks (e.g. Hadoop, Spark, Pig and Hive). Hence, instead of trying to substitute any of the available analytics systems, Luigi provides the means to plug together jobs from different frameworks in a single batch-oriented workflow. In doing so, Luigi seamlessly handles many boilerplate operations such as: dependency resolution, visualization, failure tolerance and atomic file system operations.

The Kubernetes integration was developed by the PhenoMeNal consortium, and it is now part of the official Luigi project and thus being supported and maintained by the community. The integration enables to include pipeline processing steps as Kubernetes Jobs, which represent batch-like application containers that run until command completion. Hence, Luigi mainly uses Kubernetes as a cluster-resource manager where multiple parallel jobs can be run concurrently. To this extent, it is important to point out that Luigi is not meant to be used as a substitute for parallel processing frameworks (such as Spark and Hadoop), as it has limited support for concurrent tasks. In fact, in our experience Luigi can handle up to 300 parallel tasks before it starts to break down, or 64 parallel tasks if running on a single node. A project to adapt Luigi for scientific applications in general and bioinformatics in particular resulted in a tool named SciLuigi [32].

We benchmarked Luigi on Kubernetes by reproducing a large-scale metabolomics study [33]. The analysis was carried out on a cloud-based Kubernetes cluster, provisioned via KubeNow and hosted by EMBL-EBI, showing good scalability up to 40 concurrent jobs [34]. Workflow definition, and instructions to reproduce the experiment, are publicly available on GitHub [35].

2.5 SciPipe

SciPipe [36] (SciPipe, RRID:SCR_017086) is a workflow library based on Flow-based programming principles, which enables building workflows from component libraries of predefined workflow components. SciPipe is implemented as a programming library in Go, which enables using the full power of the Go programming language to define workflows, as well as to compile workflows to executable files, for maximum ease of deployment in cloud scenarios.

SciPipe has been used to orchestrate machine learning pipelines to build predictive models for off-target binding in pharmacology [37]. SciPipe currently provides early support for container based workloads via Kubernetes, implemented through the official Kubernetes Go client library [38]. The Go client library enables transparent access to a Kubernetes cluster regardless of whether the SciPipe workflow runs outside the cluster or inside, in a pod. When running inside the cluster, it automatically detects cluster settings. When not inside a cluster, the connection to a cluster can be configured by setting the appropriate environment variables. The Kubernetes integration in SciPipe is demonstrated through the implementation of a use case workflow consisting of a set of data preparation steps for mass-spectrometry data, using the OpenMS software suite [39]. It contains a Go file (With the .go extension) with the workflow, and a Kubernetes job-spec file in YAML format (with the .yml extension) for starting SciPipe inside a Kubernetes cluster is together with the workflow [40].

Singularity containers [9] can already today be used from SciPipe on a simple level, by configuring the appropriate shell commands when creating new processes through the shell-command process constructor (the scipipe.NewProc() function in SciPipe). Development of a more integrated support for Singularity containers is planned.

2.6 Argo workflow

Argo workflow [41] originates from a corporate application with the need of a competent continuous

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 31 January 2020 doi:10.20944/preprints202001.0378.v1

integration / continuous deployment (CI/CD) solution but matured into a generalized workflow engine. It was open sourced by Intuit and is now adopted across several industries and companies among them life science and biopharma. A big differentiator of Argo is that no domain specific language knowledge is required to build workflows. The workflows are instead constructed by a clearly defined declarative approach manifested as Kubernetes custom resource definitions (CRDs), which is a templated approach to enable composability and extendability to the original Kubernetes resource orchestration. Argo is often referred to as a cloud native workflow orchestration tool and is designed for container native execution without the overhead of other non native container solutions. Argo provides a robust way to declare and run workflows as arguments and artifact objects can be declared as input and output and containers run as state transitions merely passing objects and arguments through the pipeline. In each workflow step the state is stored in redundant etcd [42] memory for resilience and reproducibility. Argo is complemented by an artifacts concept that enables data and parameter versioning through complementing Argo with an s3 compatible storage backend. Furthermore Argo events enables signals and events for triggering workflows on webhooks, resource changes, GitHub or Docker image changes etc. enabling additional automation scenarios.

2.7 Kubeflow

Kubeflow [43] is another open-source platform for Kubernetes, specifically aimed at building and running machine learning workflows and is developed by Google. As a framework, Kubeflow is intended to provide a set of tools for various tasks in the process of machine learning workflows, from data preparation and training to model serving and maintenance. Many of the tools are existing tools with a user base in the industry, such as Jupyter Notebook and Seldon, while others are customized wrappers or endpoints to frameworks like TensorFlow, PyTorch.

In particular, Kubeflow provides a workflow tool called Kubeflow Pipelines, which allows users to create workflows of varying complexity, chaining together series of container operations. The tool is based on the workflow engine Argo, which has been integrated with the Kubeflow ecosystem with a different UI than the original Argo UI. In essence, a user declares a workflow in the Kubeflow Pipelines software development kit (SDK), defining the different steps in terms of container operations, i.e. what container to use for the step, Kubernetes resource allocations, commands to execute in the container and so on. The user also declares the order of these container operations and any desired input parameters to the workflow, and from this definition a pipeline can be compiled. This pipeline is a portable configuration which in theory can be uploaded and used in any Kubeflow environment. With the pipeline deployed in Kubeflow, a user can start a run where available pipeline parameters can be provided and the progress of the run tracked in the Kubeflow Pipelines UI. Here, logs of the container's standard output, output artifacts and various Kubernetes information can be tracked for each step, along with a visualization of the workflow graph.

As an evaluation of the Kubeflow Pipelines system, a simple pipeline was developed, based on previous work [44] where a convolutional neural network is trained to "... predict cell mechanisms of action in response to chemical perturbations...", based on a dataset of cell microscopy images from the Broad Bioimage Benchmark Collection. The pipeline covers the following steps of the machine learning process: data pre-processing, model training, model evaluation and model serving preparation. The result is a workflow that handles the entire process end-to-end, building a servable machine learning model and publishing this as a Docker container that can be deployed for serving predictions. The pipeline source code, while partly tailored to a specific environment, is available in [45].

3 Discussion

The tools covered in this study have taken slightly different approaches to work with containers, with different implications. Most tools are designed to work with general command-line tools while also supporting containers. except Pachyderm, Argo and Kubeflow which are Kubernetes-native and only support containers as means of processing.

Workflow tools can almost always work with containers instead of command-line tools, using the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 31 January 2020 doi:10.20944/preprints202001.0378.v1

'docker run' command, and taking advantage of an external scheduling system such as a batch scheduler (e.g. SLURM). When deployed in cloud environments, Kubernetes is the most widely used orchestration framework by the tools, though Galaxy also supports Docker Swarm as well as provides a convenient mapping from the Conda package manager to containers. For Kubernetes, tools differ in the way they interact with its API; Pachyderm, Argo, Kubeflow Pipelines and SciPipe interact via the Go API whereas Nextflow, Galaxy and Luigi communicate via the REST API. While these APIs are not identical, the API designs are very similar in terms of data structures and in effect allow for the same level of control. In more detail, the structure of the JSON documents sent to the REST API is closely matched by the hierarchic structure of the 'spec' structs in the Go API. The workflow tools also differ somewhat in the level of flexibility they allow for utilizing the API features, where Luigi is the most versatile as it allows for passing a raw JSON string with full flexibility but also requires the most skills from workflow authors.

Error management for containerized workflows is an important concept. Kubernetes is designed to restart pods that fail, which is not particularly useful if there is an error reported from the containerized tool. The cloud paradigm includes fault-tolerance, and the tools differ in their strategies. Galaxy, Luigi and SciPipe use Kubernetes Jobs, which allows restarting the pods N times before reporting as failure. Argo and Kubeflow Pipelines run workflow steps as pods that are governed by Argo's custom Kubernetes resource workflow; as such errors for individual containers are handled as for normal Kubernetes pods, including error messages detailing reasons for failure etc. Nextflow offers the possibility to automatically tune the job, e.g. to increase the available memory for a job before reporting it as failed.

The way in which workflow systems schedule containers on Kubernetes is one factor where they differ from each other. Nextflow implements an actor-like model to handle job requests. Jobs are submitted in a manner similar to an internal job queue. Then a separate thread submits a request to the Kubernetes cluster for each of them and periodically checks their status until the job completion is detected. Galaxy has two handlers iterating over all running jobs. It keeps no separate threads or processes per Kubernetes job. Pachyderm makes direct use of Kubernetes scheduling and uses etcd for storing metadata for data versioning and status of all jobs. It is also the only system that uses the Kubernetes feature of "parallel jobs". Argo and Kubeflow Pipelines similarly utilize etcd for storing metadata of workflows and executions, and employ a workflow controller component to schedule containers based on the state of deployed workflows. Luigi keeps a thread open and continuously polls the job status. SciPipe keeps a lightweight thread (go-routine) alive for each created job, as the Go API call for creating jobs do block the calling go-routine until the job finishes or is stopped for other reasons.

Handling many concurrent jobs can become a problem in Luigi/SciLuigi, where a separate Python process is started for each job which imposes a practical limit on the number of concurrent jobs; in the authors' experience 64 is a rule of thumb for an upper limit if running on a single node, or else 300. Going above this tends to result in HTTP timeouts since workers are talking to the central scheduler via HTTP calls over the network.

So what are the main differences between the systems when running containers in cloud environments? Galaxy has a graphical user interface and might be more attractive for users with less expertise in programming/scripting. Workflows in Argo can be submitted and controlled both from multi platform cli's as well as monitored through a graphical user interface but can also be controlled by other Kubernetes cluster resources since workflows are standard custom resource description (CRD) resources. Kubeflow also provides a GUI as the main method for running pipelines, in addition to being a very portable platform in Kubernetes environments. Kubeflow Pipelines does however have a somewhat steep learning curve as the Python-based domain-specific language (DSL) requires a good grasp of Kubernetes concepts. Nextflow is arguably the most versatile system for working with containers, being able to operate either on cloud or HPC batch schedulers. Pachyderm, Argo and Kubeflow are built specifically for running on Kubernetes and Pachyderm has a data-versioning file system built-in. Argo and Kubeflow Pipelines support artifacts in workflows and versioning of input and output artifacts through interfacing with an optional S3 storage backend. Luigi/SciLuigi sup-

ports many data sources out of the box, including HDFS, S3, etc. and integrates seamlessly with Apache Hadoop and Spark. SciPipe implements an agile workflow programming API that smoothly integrates with the fast growing Go programming language ecosystem.

We would like to end with a word of caution regarding the use of containers. While containers have many advantages from a technical perspective and alleviates many of the problems with dependency management, it also comes with implications as it can make tools more opaque and discourage accessing and inspecting the inner workings of containers. This could potentially lead to misuse or lack of understanding of what a specific tool does. Thus, when wiring together containers in a scientific workflow, proper care needs to be taken that the observed output matches what would be expected from using the tool directly [46]. Also for this reason we strongly suggest to follow community best practices for packaging and containerization of bioinformatics software [47].

4 Declarations

4.1 List of abbreviations

- API: Application programming interface
- AWS: Amazon web services
- CI/CD: Continuous integration / continuous deployment
- CRD: Custom resource description
- DSL: Domain-specific language
- GCP: Google cloud platform
- GUI: Graphical user interface
- HPC: High-performance computing
- HTC: High-throughput computing
- NFS: Network file system
- NGS: Next-generation sequencing
- PV: Persistent volume
- PVC: Persistent volume claim
- SDK: Software development kit
- SFTP: SSH file transfer protocol
- UI: User interface
- VM: Virtual machine
- VMI: Virtual machine image
- WMS: Workflow management system

4.2 Funding

This research was supported by The European Commission's Horizon 2020 programme funded under grant agreement number 654241 (PhenoMeNal) and grant agreement number 731075 (OpenRiskNet), the Swedish Foundation for Strategic Research, the Swedish Research Council FORMAS, the Swedish e-Science Research Centre (SeRC), Åke Wiberg Foundation, and the Nordic e-Infrastructure Collaboration (NeIC) via the Glenna2 project. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

4.3 Author's Contributions

OS and SL coordinated the project. All authors contributed with experiences on containers and workflow systems as well as manuscript preparation.

4.4 Competing Interests

PDT and EF are founders of Seqera Labs, a company based in Barcelona, Spain, offering commercial support for the (open source) Nextflow software. SL is involved in RIL Partner AB, a Sweden based company offering commercial support for the (open source) SciPipe software.

References

[1] Vivien Marx. Biology: The big challenges of big data. *Nature*, 498(7453):255–260, Jun 2013.

[2] Bertil Schmidt and Andreas Hildebrandt. Next-generation sequencing: big data meets high performance computing. *Drug discovery today*, 22(4):712–717, Apr 2017.

[3] Mike May. Big data, big picture: Metabolomics meets systems biology. *Science*, 356(6338):646–648, 2017.

[4] Vivien Marx. Genomics in the clouds. *Nature Methods*, 10(10):941–945, September 2013.

[5] Nadia Drake. How to catch a cloud. *Nature*, 522(7554):115–116, June 2015.

[6] Björn A Grüning, Samuel Lampa, Marc Vaudel, and Daniel Blankenberg. Software engineering for scientific big data analysis. *GigaScience*, 8(5), May 2019.

[7] Martin Dahlö, Frédéric Haziza, Aleksi Kallio, Eija Korpelainen, Erik Bongcam-Rudloff, and Ola Spjuth. BioImg.org: A Catalog of Virtual Machine Images for the Life Sciences. *Bioinformatics and Biology Insights*, 9:BBI.S28636, 2015.

[8] Andrew Silver. Software simplified. *Nature*, 546(7656):173–174, June 2017.

[9] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. Singularity: Scientific containers for mobility of compute. *PLOS ONE*, 12(5):e0177459, May 2017.

[10] Jorge Gomes, Emanuele Bagnaschi, Isabel Campos, Mario David, Luís Alves, João Martins, João Pina, Alvaro López-García, and Pablo Orviz. Enabling rootless linux containers in multi-user environments: The udocker tool. *Computer Physics Communications*, 232:84–97, November 2018.

[11] Richard Shane Canon and Doug Jacobsen. Shifter: containers for hpc. *Proceedings of the Cray User Group*, 2016.

[12] Docker Hub. Accessed 20 Jan 2020.

[13] Felipe da Veiga Leprevost, Björn A Grüning, Saulo Alves Aflitos, Hannes L Röst, Julian Uszkoreit, Harald Barsnes, Marc Vaudel, Pablo Moreno, Laurent Gatto, Jonas Weber, Mingze Bai, Rafael C Jimenez, Timo Sachsenberg, Julianus Pfeuffer, Roberto Vera Alvarez, Johannes Griss, Alexey I Nesvizhskii, and Yasset Perez-Riverol. BioContainers: an open-source and community-driven framework for software standardization. *Bioinformatics*, 33(16):2580–2582, March 2017.

[14] Reem Almugbel, Ling-Hong Hung, Jiaming Hu, Abeer Almutairy, Nicole Ortogero, Yashaswi Tamta, and Ka Yee Yeung. Reproducible bioconductor workflows using browser-based interactive notebooks and containers. *Journal of the American Medical Informatics Association*, 25(1):4–12, October 2017.

[15] Heru Suhartanto, Agung P Pasaribu, Muhammad F Siddiq, Muhammad I Fadhila, Muhammad H Hilman, and Arry Yanuar. A preliminary study on shifting from virtual machine to docker container for insilico drug discovery in the cloud. *International Journal of Technology*, 8(4):611, July 2017.

[16] Ling-Hong Hung, Daniel Kristiyanto, Sung Bong Lee, and Ka Yee Yeung. GUIdock: Using docker containers with a common graphics user interface to address the reproducibility of research. *PLOS ONE*, 11(4):e0152686, April 2016.

[17] Baekdoo Kim, Thahmina Ali, Carlos Lijeron, Enis Afgan, and Konstantinos Krampis. Bio-docklets: virtualization containers for single-step execution of NGS pipelines. *GigaScience*, 6(8), June 2017.

[18] Wade L Schulz, Thomas Durant, Alexa J Siddon, and Richard Torres. Use of application containers and workflows for genomic data analysis. *Journal of Pathology Informatics*, 7(1):53, 2016.

[19] Kubernetes Project. Accessed 20 Jan 2020.

[20] Docker Swarm. Accessed 20 Jan 2020.

[21] Apache Mesos. Accessed 20 Jan 2020.

[22] Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. Nextflow enables reproducible computational workflows. *Nature Biotechnology*, 35(4):316–319, April 2017.

[23] Daniel Blankenberg, Gregory Von Kuster, Emil Bouvier, Dannon Baker, Enis Afgan, Nicholas Stoler, James Taylor, and Anton Nekrutenko and. Dissemination of scientific software with galaxy ToolShed. *Genome Biology*, 15(2):403, 2014.

[24] Enis Afgan, Dannon Baker, Bérénice Batut, Marius van den Beek, Dave Bouvier, Martin Čech, John Chilton, Dave Clements, Nate Coraor, Björn A Grüning, Aysam Guerler, Jennifer Hillman-Jackson, Saskia Hiltemann, Vahid Jalili, Helena Rasche, Nicola Soranzo, Jeremy Goecks, James Taylor, Anton Nekrutenko, and Daniel Blankenberg. The galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. *Nucleic Acids Research*, 46(W1):W537–W544, May 2018.

[25] C. Sloggett, N. Goonasekera, and E. Afgan. BioBlend: automating pipeline analyses within galaxy and CloudMan. *Bioinformatics*, 29(13):1685–1686, April 2013.

[26] Kristian Peters, James Bradbury, Sven Bergmann, Marco Capuccini, Marta Cascante, Pedro de Atauri, Timothy M D Ebbels, Carles Foguet, Robert Glen, Alejandra Gonzalez-Beltran, Ulrich L Günther, Evangelos Handakas, Thomas Hankemeier, Kenneth Haug, Stephanie Herman, Petr Holub, Massimiliano Izzo, Daniel Jacob, David Johnson, Fabien Jourdan, Namrata Kale, Ibrahim Karaman, Bita Khalili, Payam Emami Khonsari, Kim Kultima, Samuel Lampa, Anders Larsson, Christian Ludwig, Pablo Moreno, Steffen Neumann, Jon Ander Novella, Claire O'Donovan, Jake T M Pearce, Alina Peluso, Marco Enrico Piras, Luca Pireddu, Michelle A C Reed, Philippe Rocca-Serra, Pierrick Roger, Antonio Rosato, Rico Rueedi, Christoph Ruttkies, Noureddin Sadawi, Reza M Salek, Susanna-Assunta Sansone, Vitaly Selivanov, Ola Spjuth, Daniel Schober, Etienne A Thévenot, Mattia Tomasoni, Merlijn van Rijswijk, Michael van Vliet, Mark R Viant, Ralf J M Weber, Gianluigi Zanetti, and Christoph Steinbeck. PhenoMeNal: processing and analysis of metabolomics data in the cloud. *GigaScience*, 8(2), 12 2018. giy149.

[27] Pablo Moreno, Luca Pireddu, Pierrick Roger, Nuwan Goonasekera, Enis Afgan, Marius van den Beek, Sijin He, Anders Larsson, Daniel Schober, Christoph Ruttkies, David Johnson, Philippe Rocca-Serra, Ralf JM Weber, Björn Gruening, Reza M Salek, Namrata Kale, Yasset Perez-Riverol, Irene Papatheodorou, Ola Spjuth, and Steffen Neumann. Galaxy-kubernetes integration: scaling bioinformatics workflows in the cloud. *bioRxiv*, December 2018.

[28] Björn Grüning, Ryan Dale, Andreas Sjödin, Brad A. Chapman, Jillian Rowe, Christopher H. Tomkins-Tinch, Renan Valieris, Johannes Köster, and The BioConda Team. Bioconda: sustainable and comprehensive software distribution for the life sciences. *Nature Methods*, 15(7):475–476, July 2018.

[29] Pachyderm. Accessed 20 Jan 2020.

[30] Jon Ander Novella, Payam Emami Khoonsari, Stephanie Herman, Daniel Whitenack, Marco Capuccini, Joachim Burman, Kim Kultima, and Ola Spjuth. Container-based bioinformatics with pachyderm. *Bioinformatics*, 35(5):839–846, August 2018.

[31] Luigi. Accessed 20 Jan 2020.

[32] Samuel Lampa, Jonathan Alvarsson, and Ola Spjuth. Towards agile large-scale predictive modelling in drug discovery with flow-based programming design principles. *Journal of Cheminformatics*, 8(1), November 2016.

[33] Christina Ranninger, Lukas Schmidt, Marc Rurik, Alice Limonciel, Paul Jennings, Oliver Kohlbacher, and Christian Huber. MT-BLS233: Improving global feature detectabilities through scan range splitting for untargeted metabolomics by high-performance liquid chromatography-Orbitrap mass spectrometry. Accessed 20 Jan 2020.

[34] Marco Capuccini, Anders Larsson, Matteo Carone, Jon Ander Novella, Noureddin Sadawi, Jianliang Gao, Salman Toor, and Ola Spjuth. On-demand virtual research environments using microservices. *PeerJ Computer Science*, 5:e232, November 2019.

[35] MTBLS233 with PhenoMeNal Jupyter. Accessed 20 Jan 2020.

[36] Samuel Lampa, Martin Dahlö, Jonathan Alvarsson, and Ola Spjuth. SciPipe: A workflow library for agile development of complex and dynamic bioinformatics pipelines. *GigaScience*, 8(5), April 2019.

[37] Samuel Lampa, Jonathan Alvarsson, Staffan Arvidsson Mc Shane, Arvid Berg, Ernst Ahlberg, and Ola Spjuth. Predicting off-target binding profiles with confidence using conformal prediction. *Frontiers in Pharmacology*, 9, November 2018.

[38] The Kubernetes Community (2016). Go client for kubernetes. Accessed 16 Jan 2018.

[39] Hannes L Röst, Timo Sachsenberg, Stephan Aiche, Chris Bielow, Hendrik Weisser, Fabian Aicheler, Sandro Andreotti, Hans-Christian Ehrlich, Petra Gutenbrunner, Erhan Kenar, Xiao Liang, Sven Nahnsen, Lars Nilse, Julianus Pfeuffer, George Rosenberger, Marc Rurik, Uwe Schmitt, Johannes Veit, Mathias Walzer, David Wojnar, Witold E Wolski, Oliver Schilling, Jyoti S Choudhary, Lars Malmström, Ruedi Aebersold, Knut Reinert, and Oliver Kohlbacher. OpenMS: a flexible open-source software platform for mass spectrometry data analysis. *Nature Methods*, 13(9):741–748, August 2016.

[40] Samuel Lampa. OpenMS SciPipe workflow example. OpenMS SciPipe workflow example. Accessed 16 Jan 2018.

[41] The Argo Authors (2019). Argo project. Accessed 14 Oct 2019.

[42] The Etcd Community (2019). Distributed reliable key-value store for the most critical data of a distributed system. Accessed 14 Oct 2019.

[43] Kubeflow. Accessed 20 Jan 2020.

[44] Alexander Kensert, Philip J. Harrison, and Ola Spjuth. Transfer learning with deep convolutional neural networks for classifying cellular morphological changes. *SLAS DISCOVERY: Advancing Life Sciences R&D*, 24(4):466–475, January 2019.

[45] Alexander Kensert. CNN example pipeline. Accessed 20 Jan 2020.

[46] Konrad Hinsen. Verifiability in computer-aided research: the role of digital scientific notations at the human-computer interface. *PeerJ Computer Science*, 4:e158, July 2018.

[47] B Gruening, O Sallou, P Moreno, F da Veiga Leprevost, H Ménager, D Søndergaard, H Röst, T Sachsenberg, B O'Connor, F Madeira, V Dominguez Del Angel, MR Crusoe, S Varma, D Blankenberg, RC Jimenez, null null, and Y Perez-Riverol. Recommendations for the packaging and containerizing of bioinformatics software [version 2; peer review: 2 approved, 1 approved with reservations]. *F1000Research*, 7(742), 2019.