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], Paolo Di Tommaso[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

Abstract

Containers are gaining popularity in life science research as they encompass all dependencies of provisioned tools and simplifies software installations for end users, as well as offering a form of isolation between processes. Scientific workflows are ideal to chain containers into data analysis pipelines to sustain reproducible science. In this manuscript we review the different approaches to use containers inside the workflow tools Nextflow, Galaxy, Pachyderm, 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.

Introduction

Life science has 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 [1–3]. Consequently, the need for computational and storage resources continues to grow, but 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 sustain their analysis, after which the resources are released [4,5]. Another important advantage of cloud resources over HPC is that the scientists are not dependent on system administrators to install software, but this can be handled by themselves. However, software 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 computational reproducibility. VMs form the backbone in traditional cloud-based infrastructures, however, virtual machine images (VMIs) 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 catalogues [6].

Containers

Software container technology has emerged as a complement to virtual machines, offering a bit less isolation between processes and they are more lightweight and easy to share; the operating system kernel makes them much faster to launch and terminate [7]. Containers encompass all dependencies of the provisioned tools and greatly simplifies software installations for end users. The most widely used containerization solution is Docker (www.docker.com), but Singularity [8], uDocker [9], and Shifter [10] are recent alternatives that prevent users to run containers with root privileges, addressing most commons security issues when deploying contains in a multi-tenant computing clusters such as on on high-performance computing (HPC) clusters. Docker containers are commonly shared via Docker Hub (https://hub.docker.com). There are also initiatives for standardizing containers in the life sciences such as BioContainers [11]. Containers have seen an increasing uptake in the life sciences, both for delivering software tools and for facilitating data analysis in various ways [12–16].

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 (http://kubernetes.io) has over the last couple of years rose to become the most popular container orchestration system, but Docker Swarm (https://docs.docker.com/engine/swarm/) and Mesos (https://mesosphere.com/) 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.

Scientific Workflows

While orchestration tools such as Kubernetes enable scientists to run a multitude of 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 and 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 and including strategies to recover partial computations.

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 administer the system where the workflow is scheduled to run. This means the system can be tested on a local computer and executed on a remote server and cluster 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 autoscaling. 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 constitute 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, 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 you have data residing on a storage connected to a cloud infrastructure, you are able to simply move the workflow and containers there and “bring compute to the data”.

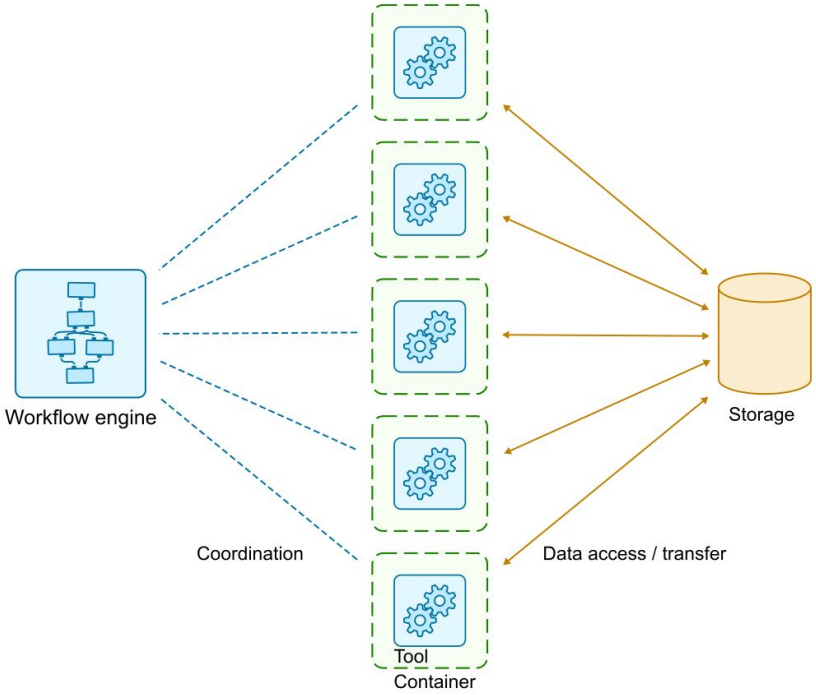

Figure 1: Overview of a containerized workflow with a workflow engine managing the dependency graph and scheduling the containers encompassing the tools to run. The tools pass data between each other by reading and writing to a shared storage system.

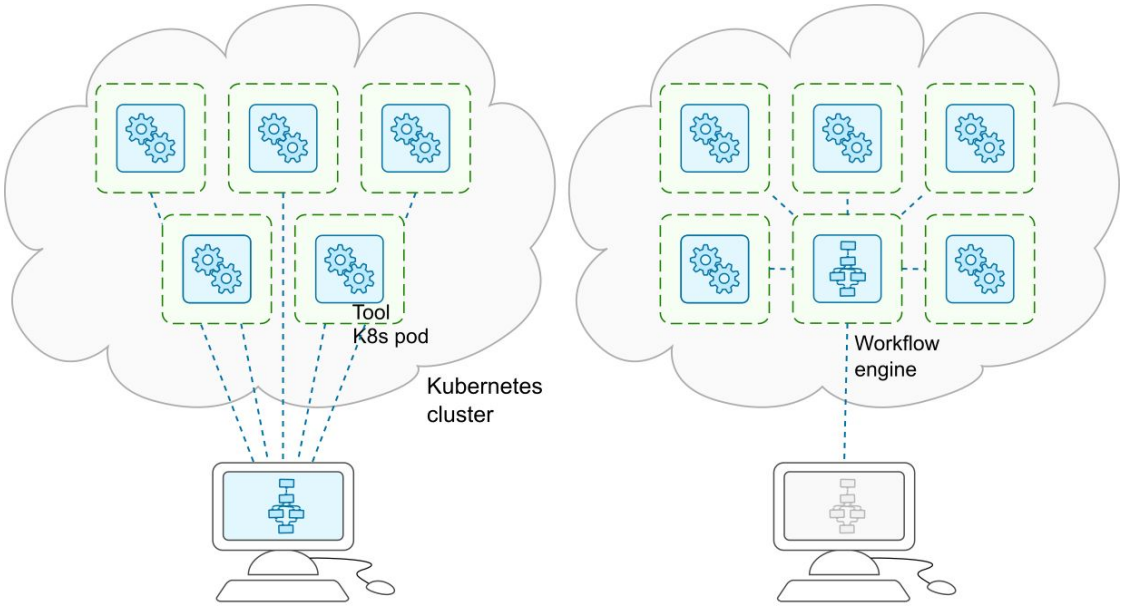

Figure 2: Comparison between two approaches on how to run the workflow engine. It can either a) run in a on the user's computer, or b) in a pod in the kubernetes cluster.

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 work with containers in cloud environments.

Nextflow
Nextflow [17] 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 by each other and are expected to be executed in a stateless manner. This paradigm simplifies the deployment of complex workflows in a distributed computing environment 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 still in experimental state). The usage of containers is defined in a declarative manner i.e. by specifying the container image that a workflow need 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, 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 containerisation technology to another with just a few changes in the workflow configuration file (eg. the same application can be deployed in a HPC cluster using Singularity or in the cloud with Docker).

Great flexibility is given on container image configuration. Nextflow supports either image-per-workflow pattern (the same image for all executed tasks) and 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 containerised workloads in a cloud-native manner using different strategies, 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 containerisation and auto-scaling is 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 additional support for similar services from other providers such as Azure Batch is planned.

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 optimises the overall computing cost when applicable. At time of writing, this feature supports the use of the AWS EC2 cloud service.

Cloud native clustering: a recent Nextflow development is the built-in support for Kubernetes, a cloud-native clustering and containerisation orchestration technology. When deploying a workflow execution with this strategy, Nextflow executes a pod running the main workflow driver application which in turn orchestrates and submits the execution of workflow tasks as separate pods. Nextflow requires the use of a Kubernetes persistent volume supporting the *read-write-many* access mode to share the input and output data across tasks. The associated *volume claim* is automatically mounted in each pod launched by Nextflow.

Nextflow provides a mature and flexible technology to support the deployment of containerised workloads across clusters and clouds in a portable manner. Future works will be focused on extending the support to cloud providers other than Amazon and the optimisation of workload deployment with the implementation of concepts such as data-affinity and co-location.

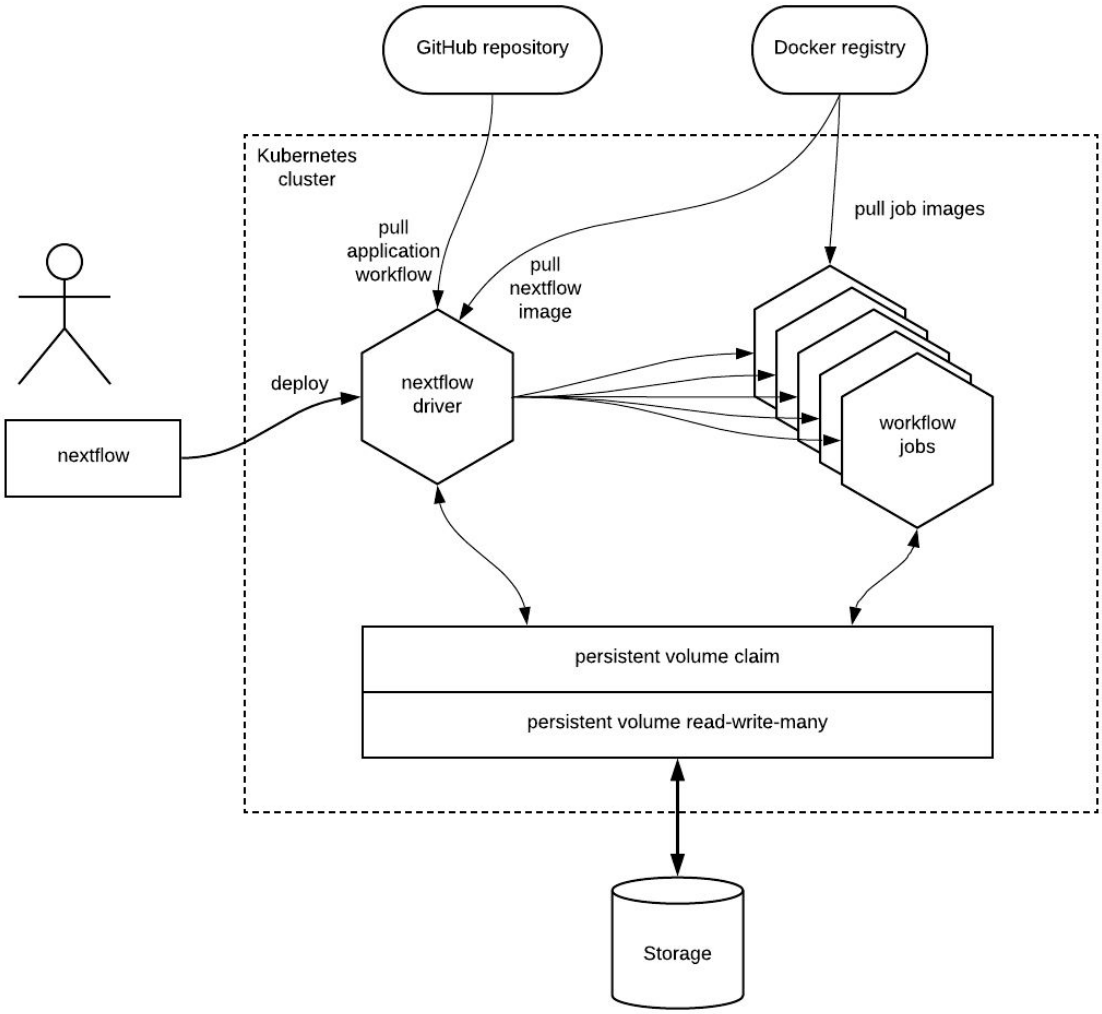

Figure 3: Nextflow workflow automated deployment and execution into a Kubernetes cluster.

Galaxy

Galaxy is a data-analysis workflow platform that has probably the most active development community in the field among workflow environments in Bioinformatics, enabling it to persist as an active project and continue to evolve since 2005, enabling reproducible research through active channels to share tools, workflows and datasets [18]. Galaxy is currently used by tens of thousands of scientists across the globe [19]. Besides offering a modern 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 5 years Galaxy gained the ability to be launched on Amazon AWS through the CloudMan Launcher Galaxy sub-project [20]. This setup follows a relative VM centric approach, which has natural limitations to scale 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 (http://phenomenal-h2020.eu/), Galaxy gained the ability to be deployed inside Kubernetes in an automated fashion through the use of Helm Charts. The galaxy-stable helm chart make use of community maintained Galaxy container images (docker-galaxy-stable), which are used on other container orchestrators, and include support for sftp access, database backend and even deployment/compatibility of HPCS systems like Condor or Slurm for hybrid cloud setups. The Kubernetes job runner, contributed to the Galaxy community as well 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 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 for Galaxy and in each job's pod as well as adequate supplementary groups handling for 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 various jobs 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 of 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 [21] from BioContainers [11].

The Galaxy-Kubernetes integration, which includes all the aspects related to Kubernetes mentioned on the previous paragraphs, has been battle tested by the PhenoMeNal H2020 Project, having its public instance received in the order of tenths of thousands jobs during the past 2 years, having tenths of deployment in the production environments and possibly various hundreds of deployments in the development setting. 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 considering to migrate to a more containerized deployment scheme based on the Galaxy-Kubernetes setup.

Pachyderm

Pachyderm (http://pachyderm.io/) 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) optimise cluster resource utilisation, 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 promises what a framework like Apache Spark does, but replacing 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 acknowledge 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, pachd then gathers the results corresponding to each datum, combines them together, and versions the complete output of the pipeline. The scheduling of pipeline workers is determined by the resource utilisation status of the workers nodes and the resource requests that are made for each pipeline, making it able to efficiently distribute workloads. Recently, work to integrate and demonstrate Pachyderm in bioinformatics was published [22].

Luigi

Luigi (https://github.com/spotify/luigi) 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. A project to adapt Luigi for scientific applications in general and bioinformatics in particular resulted in a tool named SciLuigi [23].

We benchmarked Luigi on Kubernetes by reproducing a large-scale metabolomics study (https://www.ebi.ac.uk/metabolights/MTBLS233). 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 [24]. Workflow definition, and instructions to reproduce the experiment, are publically available on GitHub (https://github.com/phnmnl/MTBLS233-Jupyter).

SciPipe
SciPipe [25] currently supports container based workloads via Kubernetes, implemented through the official kubernetes Go client library [26]. The Go client library enables transparent access to a Kubernetes cluster regardless of if the SciPipe workflow itself runs outside the cluster or inside it 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 [27]. 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 [28].

Singularity containers [8] 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.

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, with the exception of Pachyderm that only supports containers as means of processing.

Workflow tools either work with containers instead of command-line tools, using ‘docker run’, and then take 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, but 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 the interact with its API; SciPipe and Pachyderm interact via

the Go API whereas Luigi, Nextflow and Galaxy communicate over 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 what level of flexibility they allow for consuming 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 are an important concept. Kubernetes is designed to restart pods that fail, which is not particularly useful is there is an error reported from the containerized tool. The cloud paradigm includes fault-tolerance, and the tools differ here in their strategies. Galaxy, Luigi, and SciPipe uses Kubernetes Jobs, which allows restarting the pods *N* times before reporting as failure. Nextflow offers the possibility to automatically tune the job, e.g. to increase the available memory for a job before reporting it as failed.

How the workflow systems schedule containers on Kubernetes is one factor where they differ from each other. 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. Nextflow implement an *actor*-like model to handle job requests. They are submitted in a parallel manner to an internal jobs queue, then a separate thread submits a request to the Kubernetes cluster for each of them and periodically checking their status util the job completion is detected. Pachyderm makes direct use of Kubernetes scheduling and uses etcp 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". Luigi keeps a thread open and continuously polls the job status. Galaxy has 2 handlers iterating over all running jobs, it keeps no separate threads or processes per Kubernetes job.

Handling many concurrent jobs can become a problem in Luigi/SciLuigi, where a separate Python process is started for each job and, 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. 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. Nextflow is arguably the most versatile systems for working with containers being able to operator either on cloud and HPC batch schedulers. Pachyderm is build specifically for running on Kubernetes and has a data versioning file system built-in. Luigi/SciLuigi supports 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'd 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 makes it can make tools more opaque and discourage to access and inspect the inner workings of containers. This could potentially lead to misuse, and when wiring together containers in a scientific workflow proper care need to be taken that the observed output matches what would be expected from using the tool directly [29]. Also for this reason we strongly suggest to follow community best practices [30].

Acknowledgements

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) and Åke Wiberg Foundation. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Conflicts of interest

The authors declare no conflicts of interest.

Bibliography

1. Marx V (2013) Biology: The big challenges of big data. Nature 498: 255–260. doi:10.1038/498255a.
2. Schmidt B, Hildebrandt A (2017) Next-generation sequencing: big data meets high performance computing. Drug Discov Today 22: 712–717. doi:10.1016/j.drudis.2017.01.014.
3. May M (2017) Big data, big picture: Metabolomics meets systems biology. Science 356: 646–648. doi:10.1126/science.356.6338.646.
4. Marx V (2013) Genomics in the clouds. Nat Methods 10: 941–945. doi:10.1038/nmeth.2654.
5. Drake N (2015) How to catch a cloud. Nature 522: 115–116. doi:10.1038/522115a.
6. Dahlö M, Haziza F, Kallio A, Korpelainen E, Bongcam-Rudloff E, et al. (2015) Bioimg.org: A catalog of virtual machine images for the life sciences. Bioinform Biol Insights 9:

125–128. doi:10.4137/BBI.S28636.

7. Silver A (2017) Software simplified. Nature 546: 173–174. doi:10.1038/546173a.
8. Kurtzer GM, Sochat V, Bauer MW (2017) Singularity: Scientific containers for mobility of compute. PLoS ONE 12: e0177459. doi:10.1371/journal.pone.0177459.
9. Gomes J, Campos I, Bagnaschi E, David M, Alves L, et al. (2017) Enabling rootless Linux Containers in multi-user environments: the udocker tool.
10. Canon S, Jacobsen D (2016) Shifter: Containers for HPC. Cray User Group.
11. da Veiga Leprevost F, Grüning BA, Alves Aflitos S, Röst HL, Uszkoreit J, et al. (2017) BioContainers: an open-source and community-driven framework for software standardization. Bioinformatics 33: 2580–2582. doi:10.1093/bioinformatics/btx192.
12. Almugbel R, Hung L-H, Hu J, Almutairy A, Ortogero N, et al. (2018) Reproducible Bioconductor workflows using browser-based interactive notebooks and containers. J Am Med Inform Assoc 25: 4–12. doi:10.1093/jamia/ocx120.
13. Suhartanto H, Pasaribu AP, Siddiq MF, Fadhila MI, Hilman MH, et al. (2017) A Preliminary Study on Shifting from Virtual Machine to Docker Container for Insilico Drug Discovery in the Cloud. IJTech 8: 611. doi:10.14716/ijtech.v8i4.9478.
14. Hung L-H, Kristiyanto D, Lee SB, Yeung KY (2016) GUIdock: Using Docker Containers with a Common Graphics User Interface to Address the Reproducibility of Research. PLoS ONE 11: e0152686. doi:10.1371/journal.pone.0152686.
15. Kim B, Ali T, Lijeron C, Afgan E, Krampis K (2017) Bio-Docklets: virtualization containers for single-step execution of NGS pipelines. Gigascience 6: 1–7. doi:10.1093/gigascience/gix048.
16. Schulz WL, Durant TJS, Siddon AJ, Torres R (2016) Use of application containers and workflows for genomic data analysis. J Pathol Inform 7: 53. doi:10.4103/2153-3539.197197.
17. Di Tommaso P, Chatzou M, Floden EW, Barja PP, Palumbo E, et al. (2017) Nextflow enables reproducible computational workflows. Nat Biotechnol 35: 316–319. doi:10.1038/nbt.3820.
18. Blankenberg D, Von Kuster G, Bouvier E, Baker D, Afgan E, et al. (2014) Dissemination of scientific software with Galaxy ToolShed. Genome Biol 15: 403. doi:10.1186/gb4161.
19. Afgan E, Baker D, Batut B, van den Beek M, Bouvier D, et al. (2018) The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. Nucleic Acids Res 46: W537–W544. doi:10.1093/nar/gky379.
20. Sloggett C, Goonasekera N, Afgan E (2013) BioBlend: automating pipeline analyses within Galaxy and CloudMan. Bioinformatics 29: 1685–1686. doi:10.1093/bioinformatics/btt199.
21. Grüning B, Dale R, Sjödin A, Chapman BA, Rowe J, et al. (2018) Bioconda: sustainable

and comprehensive software distribution for the life sciences. Nat Methods 15: 475–476. doi:10.1038/s41592-018-0046-7.

22. Novella JA, Khoonsari PE, Herman S, Whitenack D, Capuccini M, et al. (2018) Container-based bioinformatics with Pachyderm. Bioinformatics. doi:10.1093/bioinformatics/bty699.

23. Lampa S, Alvarsson J, Spjuth O (2016) Towards agile large-scale predictive modelling in drug discovery with flow-based programming design principles. J Cheminform 8: 67. doi:10.1186/s13321-016-0179-6.

24. Capuccini M, Larsson A, Carone M, Novella J, Sadawi N, et al. (2018) KubeNow: an On-Demand Cloud-Agnostic Platform for Microservices-Based Research Environments. arxiv.

25. Lampa S, Dahlö M, Alvarsson J, Spjuth O (2018) SciPipe - A workflow library for agile development of complex and dynamic bioinformatics pipelines. BioRxiv. doi:10.1101/380808.

26. The Kubernetes Community (2016) Go client for Kubernetes. Go client for Kubernetes. Available: https://github.com/kubernetes/client-go. Accessed 16 January 2018.

27. Röst HL, Sachsenberg T, Aiche S, Bielow C, Weisser H, et al. (2016) OpenMS: a flexible open-source software platform for mass spectrometry data analysis. Nat Methods 13: 741–748. doi:10.1038/nmeth.3959.

28. Lampa S (2017) OpenMS SciPipe Workflow Example. OpenMS SciPipe Workflow Example. Available: https://github.com/scipipe/scipipe/tree/feature/k8s-support/examples/k8s_openms. Accessed 16 January 2018.

29. Hinsen K (2018) Verifiability in computer-aided research: the role of digital scientific notations at the human-computer interface. PeerJ Computer Science 4: e158. doi:10.7717/peerj-cs.158.

30. Gruening B, Sallou O, Moreno P, da Veiga Leprevost F, Ménager H, et al. (2018) Recommendations for the packaging and containerizing of bioinformatics software [version 1; referees: 2 approved with reservations]. F1000Res 7: 742. doi:10.12688/f1000research.15140.1.