Complex Simulation Workflows in Containerized High-Performance Environment

Vladimr Visnovsky[1]

Viktoria Spisakova[1]

Vojtech Spiwok[2]

Jana Hozzova[1]

Jaroslav Olha[1]

Dalibor Trapl[2]

Lukas Hejtmanek[1]

Ales Krenek[1z]

[1] Institute of Computer Science, Masaryk University, Czech Republic

[2] Department of Biochemistry and Microbiology, University of Chemistry and Technology Prague, Czech Republic

[z] ljocha@ics.muni.cz

Acknowledgements: This work was supported by the Grant Agency of the Czech Republic, grant no. 19-16857S. Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

Abstract

Cutting-edge research involving in-silico simulations of-ten requires to use many heterogeneous software tools made by different developers, resulting in complex, custom-made pipelines of various scripts and programs. Such pipelines are nearly impossible to be reproduced by other research groups, jeopardizing both quality and acceptance of such research results.

Starting with two in-house use cases in computational chemistry, we identified a common pattern applicable for other applications as well, and we designed and implemented a solution based on

 doi: 10.18278/jpcs.7.2.4

Jupyter notebooks to drive the simulation, Docker containers to package all soft-ware dependencies, and Kubernetes execution environment to run several cooperating containers which build up the whole application.

Keywords: workflow, Jupyter notebook, Docker, Kubernetes, reproducibility, protein folding, molecular force field

Flujos de trabajo de simulación complejos en entornos de alto rendimiento en contenedores

Resumen

La investigación de vanguardia que involucra simulaciones in-silico a menudo requiere el uso de muchas herramientas de software heterogéneas creadas por diferentes desarrolladores, lo que da como resultado canalizaciones complejas y personalizadas de varios scripts y programas. Dichos proyectos son casi imposibles de reproducir por otros grupos de investigación, lo que pone en peligro tanto la calidad como la aceptación de dichos resultados de investigación.

Comenzando con dos casos de uso internos en química computacional, identificamos un patrón común aplicable también para otras aplicaciones, y diseñamos e implementamos una solución basada en portátiles Jupyter para impulsar la simulación, contenedores Docker para empaquetar todas las dependencias de software, y el entorno de ejecución de Kubernetes para ejecutar varios contenedores cooperativos que construyen toda la aplicación.

Palabras Clave: flujo de trabajo, Jupyter notebook, Docker, Kubernetes, reproducibilidad, plegamiento de proteínas, campo de fuerza molecular

容器化高性能环境中的复杂模拟工作流程

摘要

关于in-silico 计算模拟的最新研究经常要求使用许多异质的、由不同开发者设计的软件工具，这造成一系列有关不同脚本和程序的复杂定制系统。这类系统几乎不可能由其他研

究团队进行再生产，因此破坏了这类研究结果的质量和接受度。

我们通过计算化学中的两个内部用例，识别了一个适用于其他程序的通用模式，并且我们设计和执行了一个解决方案，该方案基于Jupyter notebooks（驱动模拟），Docker容器（包装所有软件依赖），以及Kubernetes执行环境（驱动建立起整个应用过程的合作容器）。

关键词：工作流程，Jupyter notebook，Docker，Kubernetes，再现性，蛋白质折，分子力场

Introduction

The nature of modern in-silico simulations requires the use of complex pipelines of programs, scripts, and intermediate results. This makes it virtually impossible to reproduce even moderately complex computational experiments for the purpose of validation or extension of existing research.

This known problem is gradually addressed by development of formats and applications such as the Jupyter Notebook (Kluyver et al., 2016) which allow for the exact replication of code pipelines. Container technologies such as Docker (Merkel,2014) can be used to fully package a computing environment, including specific software versions and all of their dependencies, although with growing complexity, the container images become very large and rather difficult to manage.

Besides the issue of reproducibility, another common problem is that complex simulation pipelines tend to have uneven demands on hardware resources, i.e, some parts of the pipeline require much more computational power than others, potentially causing idling and inefficiency during the less intensive parts. A cloud-based service could solve this problem, since it can dynamically allocate more resources for the more intensive parts of the pipeline, and then free the resources for use by other applications; however, the typical overhead associated with spawning or reconfiguring entire virtual machines is rather prohibitive. On the contrary, the Kubernetes platform (Hightower et al., 2017), which provides an environment for running multiple containers and man-aging their mutual interactions, appears to be promising for these purposes.

In this paper, we introduce a deployment setup to address all of these problems. The workflow of the computational experiments is implemented as a Jupyter note-book strictly, which runs in a container with all the necessary lightweight dependencies already pre-installed. The notebook container

itself runs in Kubernetes, and it can spawn other jobs (e.g., requesting a node with more CPU cores and RAM temporarily) to run heavyweight dependencies in separate containers. The notebook container itself can be started either by the user manually, or by a simple web front-end application on behalf of the user, bringing additional convenience. The whole setup ensures the reproducibility of the computation (its sequence and software used is controlled strictly), while leveraging the flexibility of resource allocation, as well as making maintenance easier by isolating self-contained software packages in separate containers. We demonstrate a practical usage of the setup on two pilot use cases from computational chemistry.

Related Work

The Binder project (Project Jupyter et al., 2018) addresses the problem of long-term preservation and reproducibility of Jupyter Notebooks. It takes care of storing the versioned notebook in a suitable repository (Git) together with precise information on the environment (base operating system, versioned dependencies etc.). However, it does not handle resource allocation, and it assumes the underlying environment to execute a single Docker container with the notebook.

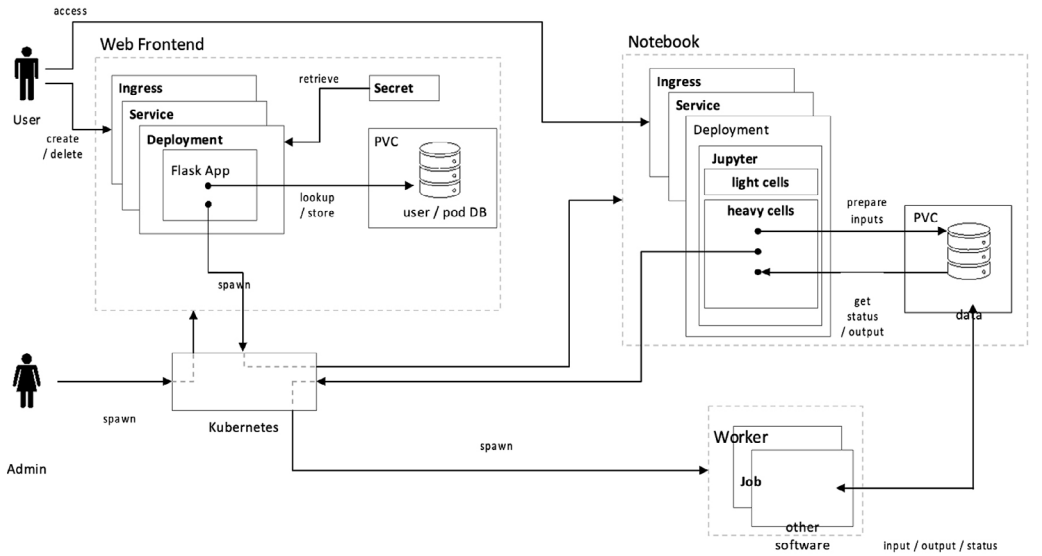

Figure 1. Principal interactions and dataflows among the user, web front end, Kubernetes, notebook and worker containers.

Within the European Open Science Cloud (EOSC) ecosystem, Jupyter Notebooks are one option to access compute resources[1]. The user can specify the required resources, but the allocation is static and it is limited to a

1 https://nootebooks.egi.eu

single container. Recently, the Galaxy project (Afgan et al., 2018) introduced Jupyter Notebook as an interactive environment (Coraor et al., 2021) partially linked to the Galaxy native mechanisms of tracking data provenance. However, the design favors the use of a notebook as a final visualization and analysis step; there are no specific means to control the computation.

Technical Solution

The interactions of all the principal components of the proposed deployment setup are shown in Fig. 1. The application's life starts with the service administrator's action: deploying the application web frontend in Kubernetes and exposing it to the users. The application, implemented in Flask (Grinberg, 2018), performs the following tasks: it authenticates the user, looks up whether there is a running Jupyter notebook belonging to this user, spawns it if not, redirects the user to the notebook endpoint, and deletes the notebook deployment if it is not needed anymore. The frontend is composed of the usual stack of Kubernetes components: Ingress to expose the frontend at a predefined public URL (including a pre-assigned DNS name and a TLS certificate), Service to manage the opened port and to link the Ingress to the deployment, and Deployment which contains the actual Docker container with the frontend application. The mapping between user identities and the running notebooks is maintained in a trivial database which resides in another Kubernetes resource, Persistent Volume Claim (PVC) attached to the Deployment. The choice of user authentication mechanism is arbitrary, just a unique identification of the user is required. We prefer to use the ELIXIR authentication service (Linden et al., 2018).

When the user opens the notebook for the first time, the frontend application deploys a small set of Kubernetes resources forming a functional computational setup. A new PVC is created and attached to the newly-created Deployment to store its working data. The notebook's Ingress resource exposes the notebook on a dynamically assigned DNS name unique for the user. The container starts from an image with the base Jupyter software stack, as well as lightweight dependencies required for the specific application (for example, in our pilot use cases, libraries to read/write chemical le formats and to manipulate molecular structures).

Lightweight computation steps are executed directly in the notebook container. On the other hand, there are steps which require more hardware resources (CPU cores, memory, GPU) and additional complex software stacks (molecular dynamics or quantum chemical simulations in our use cases). We prefer to run these steps in separate containers to allocate the resources only when needed, and to keep more complex pieces of software separate images to streamline their maintenance. The notebook prepares inputs for the heavy computation and stores them on the notebook's PVC. Then, running a heavy cell deploys a Kubernetes Job. It requests the appropriate hardware re-

sources, starts a container from an image containing the necessary software, and mounts the notebook's PVC to access inputs and to store out-puts.

When the user revisits the web frontend again, the application finds the notebook endpoint in its database. If it is still active (different applications may set different notebook lifetimes), the user is redirected there. Otherwise, a new deployment is started and the database is updated. An experienced user can also bypass the web frontend, spawning the payload containers directly.

Pilot Use Cases

Protein folding space exploration

As we reported recently (Krenek et al., 2020), we can use synthetic, artificially generated "landmarks" of a protein to sample the multidimensional landscape of all possible 3D shapes of the protein, and train a neural network model to calculate a non-linear, low-dimensional embedding of this space. Such an embedding can be used to generate a bias potential of the molecule that "pushes" its simulation run in Gro-macs (Abraham et al., 2015) towards not-yet-explored areas of the landscape. Consequently, we can explore biologically interesting behavior of the molecule in significantly shorter simulation time than before.

The main steps of the simulation include the generation of random landmarks, energy minimization, generation of low-dimensional embedding, neural network training, simulation preparation and execution, and the visualization of results. The minimization, preparation and simulation steps require Gromacs, with the simulation step being the most demanding, taking hours or even days to complete (depending on the molecule). Almost all of the steps (with the exception of NN training and visualization) allow for some level of parallelism; in particular, the main simulation run in Gromacs can be highly parallel, even taking advantage of GPUs.

Table 1 summarizes the main steps of the simulation: the implementation in Jupyter notebook[2] contains more than 50 code cells. Typically, these steps result in visualizations, and the user may tune some of the simulation parameters and repeat certain stages. The table also il-lustrates the heterogeneity of the resource requirements. In this case, we managed to install all but one of the dependencies (Gromacs) in a single container image of rea-sonable size (2.4 GB) which can be built fairly quickly. This is complemented by a dedicated image (400 MB) with Gromacs including extensions for biased simulations (Tribello et al., 2014), where building includes compilation and it runs for up to an hour.

Molecular force field correction

Molecular dynamics simulations rely on so-called force elds, which contain specific parameters that determine the properties of molecules. These parameters are well-tuned for biopolymers

2 https://github.com/ljocha/chicken-and-egg/

such as proteins (Lindor-Larsen et al., 2012), but struggle to faithfully reproduce the behavior of smaller organic molecules due to their high chemical diversity. Therefore, the force elds need to be reparametrized for each such molecule.

We provide a pipeline that automatizes this process. It generates landmark structures, and it calculates the correction between accurate quantum mechanics and the less accurate force eld (Trapl et al., 2021). The deployment is available publicly.[3]

Table 2 shows the main steps needed to calculate the molecular force eld correction. Each step is divided into its own set of logically grouped Jupyter notebook cells. Most of these steps can be customized by the user: parameters of calculations can be changed accordingly. Visualizations are provided for the results after multiple steps, and the user can decide whether to change the parameters and repeat the step or continue the computation.

The table also illustrates the needed software. The majority of steps need Gromacs, which uses the same Docker container image as the Protein folding space exploration use case. Another software we need is Orca (Neese et al., 2020), which is containerized in Docker as well (3.4 GB). The rest is installed directly, and no other container is needed.

Conclusions

We identified a common setup pattern in our work on simulations in computational chemistry, and implemented the workflow as a Jupyter Notebook wrapped in a container with all of its dependencies, and running it in Kubernetes. The more demanding steps of the pipeline were isolated in other container images and spawned when needed. The whole solution is interfaced with a simple web front-end. We demonstrated this workflow on two pilot use cases, allowing us to complement our own publications with instant access to completely reproducible implementations of all the experiments. This solution is generic for any applications which follow a similar pattern.

3 http://pmcvff-correction.cerit-sc.cz/

Table 1. Essential steps of protein folding simulation. Exact values depend on the simulated protein, short steps are within a few minutes, medium less than an hour, long can take days; medium parallelism can leverage up to 10 cores, high can be dozens (depending on the protein size)

Step	Duration	Parallelism	GPU	Gromacs
Random landmarks	Short	Medium	No	No
Energy minimization	Medium	Medium	No	Yes
Embedding	Short	Medium	No	No
Train NN	Short	No	No	No
Prepare simulation	Medium	Medium	Possible	Yes
Full simulation	Long	High	Yes	Yes
Visualize results	Short	no	no	no

Table 2. Main steps of molecular force eld correction pipeline. Exact values depend on the simulated molecule, short steps take minutes to complete, medium up to an hour, long steps take hours to days.

Step	Duration	Parallelism	GPU	External Software
Molecule visualization	Short	No	No	No
Energy minimization	Medium	No	Possible	Gromacs
Trajectory generation	Medium	Medium	Possible	Gromacs
Configurations clustering	Short	No	No	Gromacs
Train NN	Medium	No	No	No
Energy evaluation	Long	High	No	Orca
Energy minimization	Medium	No	Possible	Gromacs
Visualize results	Short	No	No	No

References

Abraham, M.J., Murtola, T., Schulz, R., Páll, S., Smith, J.C., Hess, B., & Lindahl, E. (2015). GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. SoftwareX, 1(2), 19-25. ISSN 2352-7110. doi: 10.1016/j.softx. 2015.06.001.

Coraor, N., Gladman, S., Rasche, H., & Bretaudeau, A. (2021). Galaxy Interactive Tools. Galaxy training. https://training.galaxyproject.org/training-material/topics/admin/tutorials/ interactive-tools/tutorial.html.

Grinberg, M. (2018). Flask web development: developing web applications with python. O'Reilly Media, Inc.

Hightower, K., Burns, B., & Beda, J. (2017). Kubernetes: Up and Running Dive into the Future of Infrastructure (1st. ed.). O'Reilly Media, Inc.

Jalili, V., Afgan, E., Gu, Q., Clements, D., Blankenberg, D., Goecks, J., Taylor, J., & Nekrutenko, A. (2018). The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. Nucleic Acids Research, 46(W1), W537-W544. ISSN 0305-1048. doi:10.1093/nar/gky379.

Kluyver T., Ragan-Kelly, B., Perez, F., & Granger, B. (2016). Jupyter Notebooks-a publishing format for reproducible computational workflows. In F. Loizides & B. Schmidt (Eds), Positioning and Power in Academic Publishing: Players, Agents and Agendas (pp. 87-90). IOS Press.

Krenek A., Hozzova , J., Olha, J., Trapl, D., & Spiwok, V., (2020). Exploring Protein Folding Space with Neural Network Guided Simulations. In MODELLING AND SIMULATION 2020. EUROSIS-ETI. ISBN 978-94-92859-12-9, 305{309.

Linden, M., Prochazka, M., Lappalainen, I., Bucik, D., Vyskocil, P., Kuba, M., Silén, S., Belmann, P., Sczyrba, A., Newhouse, S., Matyska, L., & Nyrönen, T. (2018). Common ELIXIR Service for Researcher Authentication and Authorisation. F1000Research. doi:10.12688/ f1000research.15161.1.

Lindor-Larsen, K., Maragakis, P., Piana, S., Eastwood, M., Dror, R.O., & Shaw, D.E. (2012). Systematic Validation of Protein Force Fields Against Experimental Data. PLoS One, 7(2). doi:10.1371/journal.pone.0032131.

Merkel, D. (2014, May). Docker: Lightweight Linux containers for consistent development and deployment. Linux Journal.

Neese, F, Wennmohs, F., Becker, U., & Riplinger, C. (2020). The ORCA quantum chemistry program package. J Chem Phys, 152(22). doi: 10.1063/5.0004608.

Project Jupyter, Forde, J., Bussonnier, M., & Freeman, J. (2018). Binder 2.0 - Reproducible, Interactive,

Sharable Environments for Science at Scale. Proceedings of the 17th Python in Science Conference, 113-120. doi:10.25080/Majora-4af1f417-011.

Trapl, D., Krupicka, M. Visnovsky,, V., Hozzova , J., Olha, J., Krenek, A., & Spiwok, V. (2021). Property map collective variable as a useful tool for force field correction. J Chem Inf Model. Submitted.

Tribello, G.A., Bonomi, M., Branduardi, D., Camilloni, C., & Bussi, G. (2014). PLUMED 2: New feathers for an old bird. Computer Physics Communications, 185(2), 604-613. DOI:10.1016/j.cpc.2013.09.018.