EXPERIENCE IN ORGANIZING FLEXIBLE ACCESS TO REMOTE COMPUTING RESOURCES FROM JUPYTERLAB ENVIRONMENT USING TECHNOLOGIES OF EVEREST AND TEMPLET PROJECTS

S. Vostokin[1,a], S. Popov[1], O. Sukhoroslov[2,3]

[1] *Samara National Research University*

[2] *Institute for Information Transmission Problems of the Russian Academy of Sciences*

[3] *HSE University*

E-mail: [a] easts@mail.ru

The paper describes the experience of building distributed web applications based on the interactive computing technologies of the Jupyter project. The new architecture of such applications is proposed, considering the possibility of deploying a Jupyter notebook server separately from computing resources, and the possibility to interact with several computing resources simultaneously. These features are implemented using the Everest platform for resource integration and the Templet SDK for accessing the platform from Jupyter notebooks. Two examples of computing and data processing applications built on this architecture are discussed. The proposed solutions are designed to automate resource-intensive computing activities in scientific and research projects.

Keywords: interactive computing, Project Jupyter, many-task application, distributed computing

Sergei Vostokin, Stefan Popov, Oleg Sukhoroslov

1. Introduction

New tools for automating workflows in the fields of data science, scientific computing, and machine learning are under active development. One of the significant advances is the interactive web-based development environment JupyterLab. JupyterLab allows one to quickly create a convenient multi-window web interface for a distributed application that runs from a browser and does not require local installation. However, the following problem exists: scientific computing applications require not only a rich user interface but flexible access to a wide range of computing resources. The standard solution of the problem – deploying JupyterLab where the computation is done – doesn't work in the two important cases: (a) there is no technical feasibility of such deployment; (b) a distributed application needs to work with several resources at the same time. In the article, we present a solution that covers these use cases, an alternative to commercial cloud solutions such as Google Colab, Yandex DataSphere, JetBrains Datalore, which are also based on Project Jupyter [1].

2. Method for building the distributed application

In the solution, we use simple, affordable, but resource-limited JupyterLab deployment options. The first one is the public cloud deployment based on MyBynder.org public service. The second one is the virtual machine deployment in a private cloud powered by The Littlest JupyterHub. In both variants of deployment, the JupyterLab server implements the interface (in the form of a Jupyter notebook) and starts the orchestrator.

We implement an orchestrator for managing tasks in many-task applications using the Templet SDK [2]. This software development kit is a research project of Samara National Research University. The orchestrator implements a variant of the actor model designed to manage tasks in many-task applications.

The orchestrator accesses the Everest platform through the REST protocol to execute tasks. The Everest platform [3,4] was developed by the Institute for Information Transmission Problems of the Russian Academy of Sciences to manage the execution of tasks on remote computing resources. Everest server allows users to attach their resources and define resource access policy; distributes application and launches tasks across resources; and returns the results of tasks to the orchestrator, which generates the following tasks in accordance with the calculation logic.

3. Components of the distributed application

A more detailed view of the solution is shown on Fig. 1. The figure shows the components of a distributed application and their deployment. The application has three parts. The first part is deployed on JupyterLab in the docker container by using MyBynder.org (or directly on The Littlest JupyterHub server). This part contains a Jupyter notebook to define application workflow; an orchestrator to dynamically form DAG of tasks; and the Templet runtime library that uses libcurl to communicate with the Everest platform. The second part is the Everest platform server. On the platform server, we have created a special Everest application and a code to be run on resources. The Everest application's main function is to check the validity of the REST request and to map the request to the command line. The third part is the Everest resource agent which is installed and running on each resource used in the computation. The resource agent deploys code from the Everest application onto the resource and invokes it using the command line.

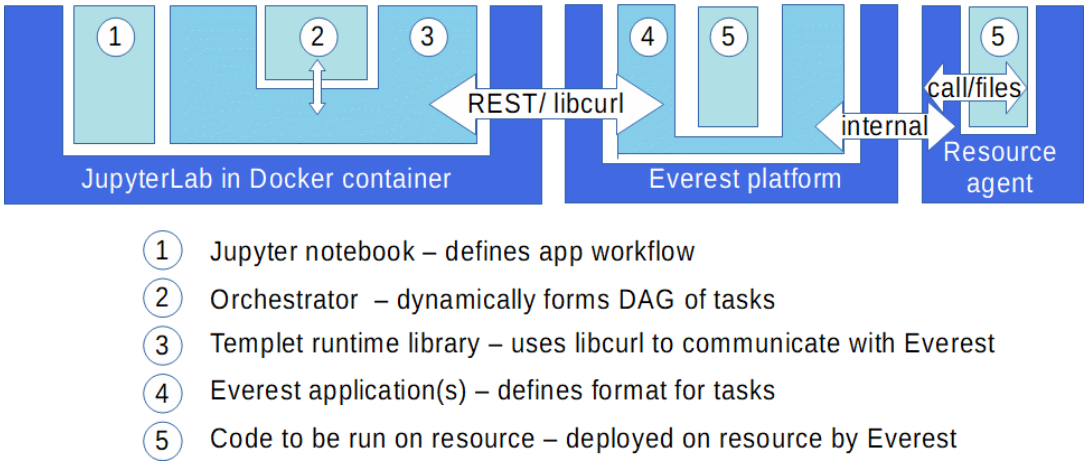

Figure 1. Application architecture

The Fig.2 shows the architecture of the application in terms of its deployment. The deployment starts with the registration of computing resources of the application on the Everest platform and obtaining access tokens for agent programs through the Everest web interface (step 1).

The next (step 2) is the installation of application components. This installation is performed through the web interface of the Everest platform.

After running and setting Windows 7 virtual machines in the corporate cloud of Samara University is performed (step 3). It includes installing agent programs on them using the access tokens and verifying the activity of agent programs through the web interface on the Everest platform.

Then, at step 4, if necessary, the user uploads data to a file server in the corporate cloud of Samara University. This can be performed through one of the virtual machines.

At step 5 the user launches the application orchestrator from the GitHub code repository via the web interface. This action automatically activates the Binder service (step 6) to build a docker container with the application orchestrator running in the JupyterLab environment.

Finally, the Binder deploys the docker container in the Google Cloud and returns the link to the web interface of the orchestrator to the web terminal of the application user (step 7). After that, the user launches the orchestrator via the web interface and starts processing (step 8).

During the processing, the application orchestrator sends commands to launch the next tasks to the Everest platform server and polls the status of previously launched tasks (step 9). At the same time, the Everest platform server distributes tasks for execution to free virtual machines through resource agent programs (step 10).

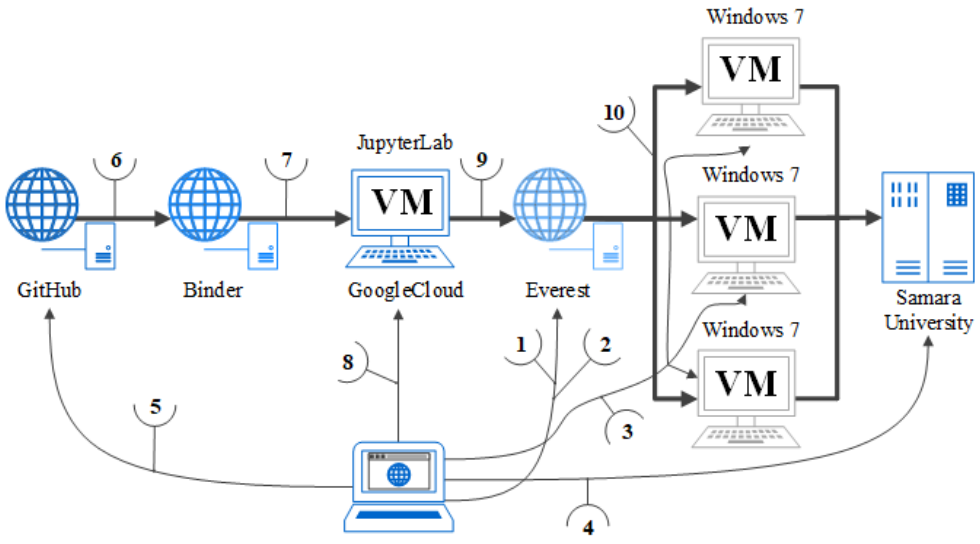

Figure 2. Deployment procedure

4. Sample applications: compute-intensive and data-intensive cases

We have developed two examples to demonstrate the practical use of the described distributed application architecture.

The first example relates to the area of heterogeneous compute-intensive applications [5]. The application is used to study dynamical systems based on the calculation of Lyapunov exponents. The practical purpose of the application is to find the parameters of a dynamical system at which chaotic behavior occurs. The purposes of using our architecture in this example are: to hide the heterogeneous nature of the components of the application written in C ++ and the Maple system language from the end-user; to enable all corporate licenses of Samara University at the same time to parallelize the parametric scanning process; to flexibly customize the scanning process through the JupyterLab web interface. The application implements the "bag of tasks" algorithmic skeleton.

The second example relates to the area of data-intensive applications [6]. The application is used to build a frequency dictionary for the Twitter microblogs. The practical purpose of such an application is to track the dynamics of the vocabulary to learn the focus of public attention in the subject area of interest. The purposes of using our architecture in this example are: to show the possibility of non-dedicated computing resources utilization in a data processing task; to implement processing based on a complex graph of task dependencies, generated programmatically using the Templet SDK. The application implements the algorithmic skeleton called "asynchronous round-robin tournament" [7].

5. Conclusion

We have implemented a distributed application architecture that allows one to work via the JupyterLab web interface without local installation, deploy JupyterLab separately from computing resources, and run complex workflow scenarios involving parallel computing on multiple resources.

As a potential future optimization, to minimize dependency on the JupyterLab deployment method, we plan to implement the JupyterLab session as an Everest job that can be launched via special Everest application.

References

[1] Project Jupyter. Available at: https://jupyter.org. (accessed 14.09.2021)

[2] The Templet Project. Available at: https://github.com/the-templet-project. (accessed 14.09.2021)

[3] The Everest Project. Available at: http://everest.distcomp.org. (accessed 14.09.2021)

[4] Sukhoroslov, O. Volkov, S. Afanasiev, A. A Web-Based Platform for Publication and Distributed Execution of Computing Applications // 14th International Symposium on Parallel and Distributed Computing (ISPDC). IEEE, 2015, pp. 175-184.

[5] Popov, S.N. Vostokin, S.V. Doroshin, A.V. Dynamical systems analysis using many task interactive cloud computing // Journal of Physics: Conference Series, 2020, vol. 1694, issue 1.

[6] Vostokin, S.V. Bobyleva, I.V. Implementation of frequency analysis of twitter microblogging in a hybrid cloud based on the Binder, Everest platform and the Samara University virtual desktop service // CEUR Workshop Proceedings, 2020, vol. 2667, pp. 162-165.

[7] Vostokin, S.V. Bobyleva, I.V. Asynchronous round-robin tournament algorithms for many-task data processing applications // International Journal of Open Information Technologies, ISSN: 2307-8162, vol. 8, no. 4, 2020.