

Controlled Remote Usage of Private Shared Resources via Docker and NoVNC

Daniil Malevanniy(✉), Olga Sedova, and Oleg Iakushkin

Saint-Petersburg State University, 7/9 Universitetskaya nab.,
St. Petersburg 199034, Russia
rukmarr@gmail.com, o.yakushkin@spbu.ru

Abstract. Shared usage of computational power in scientific projects is often associated with organizational problems, since different users may need different software environments, and used resources may have no public access. The usage of the majority of existing environment managers is often not suitable for inexperienced users participating in such projects, as well as access to shared resources only through public gateways. This paper describes a method of organizing public remote access to an isolated working environment on shared resources based on containerization technology and a system for the remote call of console applications via the web interface. Combined usage of these systems allows inexperienced users to use shared computational resources with full effectivity.

Keywords: Remote access · Shared resources · Containerization

1 Introduction

The need for computing resources in scientific projects can be high even in areas far from Computer Science. Depending on the scale of the project, it may require the use of a GRID network or voluntary computing [1], or it will be limited to just leasing resources in cloud services. But, in any case, there is a need to provide end users with access to the shared resources used. This raises several problems.

Despite the fact that Linux was originally created as a multi-user operating system, the free use of one machine by several users is associated with certain inconveniences. The possibility of uncontrolled changes to the working environment by users, in particular, installing new software or changing versions of an existing one, can break the workflow of other users, but on the other hand, when the administrator controls the environment, users are often limited in their choice of software used.

In addition, it should be borne in mind that in research projects, despite their possible complexity, users who do not have extensive experience with Linux

This research was partially supported by Russian Foundation for Basic Research grant (project no. 17-29-04288).

© Springer Nature Switzerland AG 2019
S. Misra et al. (Eds.): ICCSA 2019, LNCS 11622, pp. 782–791, 2019.
https://doi.org/10.1007/978-3-030-24305-0_59

and the command-line interfaces can participate. However, a large proportion of software used in such projects is available only in Linux-like environments and in the form of console applications [10–12].

Other problems may include strict firewall policies regarding incoming connections and the absence of the public IP address of the private machines of educational institutions acting as computational resources. This greatly complicates access to such resources for users, especially untrained.

This paper describes a possible solution to the problems described above. The created system provides users with two interfaces: a full system GUI and a simplified web-interface for launching console programs. The working environments of each user are isolated from one another, and there is no restriction to user permissions in them. In addition, we describe how to provide public access to the work environment in the case when it deployed on private resources.

2 Approach

The goal of this work is to implement a system for remote use of shared resources, which, on the one hand, will allow experienced users to fully manage their work environment, and on the other hand, will provide their less advanced colleagues with a convenient interface for working with console applications.

Also, in order to ensure the safety and efficiency of data storage, the system should provide centralized file storage located on a separate machine, containing both public and private user files. Access to this repository should be possible from both available interfaces.

Another important requirement for the system is the ability to work with several host machines, including those located behind the firewall in private networks. At the same time, the way for a user to access their work environment should not depend on the machine on which it is deployed.

For the administrator, the system should provide control over users and their work environments and at the same time automate the process of adding new hosts or users to the system.

3 Results

To isolate users and give them full control over their working environment, the Docker containerization platform is used [2]. When working in the Docker container, the user has no restrictions but works in isolation from other users. This allows them to fully control their environment, without side-effects in the other ones.

However, the only user interface of most Docker images is a virtual terminal, which is unsuitable for use by an untrained user. To access the graphical interface of the system, the TigerVNC vncserver [3] is used, which provides remote access to the virtual display running inside the container. On the client side, the NoVNC JavaScript library [4] is used, allowing users to connect to the VNC server in a browser window with WebSocket protocol (Fig. 1).

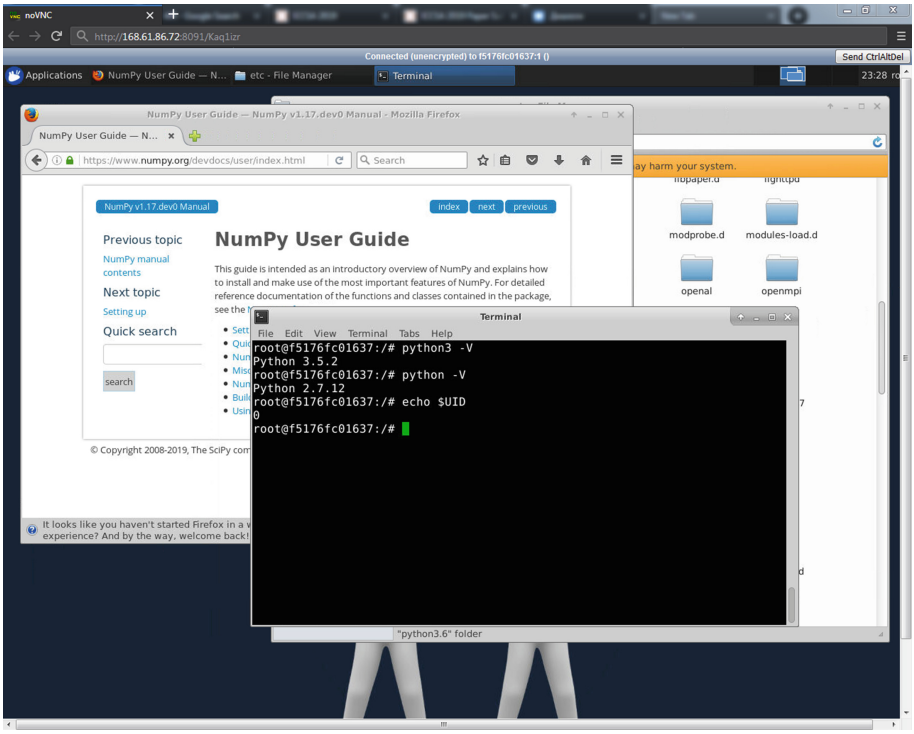

Fig. 1. System GUI, provided by NoVNC library in the browser window

If users of the system need to work with console programs, using the graphical interface of the system does not make the workflow easier. To provide inexperienced users with a convenient interface in such cases, a system was developed that generates a web-interface with the same functionality as the console program, based on a command line template and implemented as a web server running in a user container. To organize the execution of programs, the Celery task queue is used. Tasks sent to it by the user interface can be performed both in their containers, where they completely control the environment or on other nodes.

If manipulating large data is necessary when working on shared resources, it may be inconvenient to store it in a container. To organize the central storage of user data, a file system was created on the basis of the libfuse library, using Celery tasks to manipulate files located on a separate machine. Using Celery allows not only to mount the file system in a container but also to integrate it into the web interface, used to run console commands, which further simplifies the use of the system by users of different levels of experience.

To use both the system GUI provided by VNC server working in the container and the simplified web interface, public access to certain container TCP ports is required. In our case, the publication of ports on the servers used for

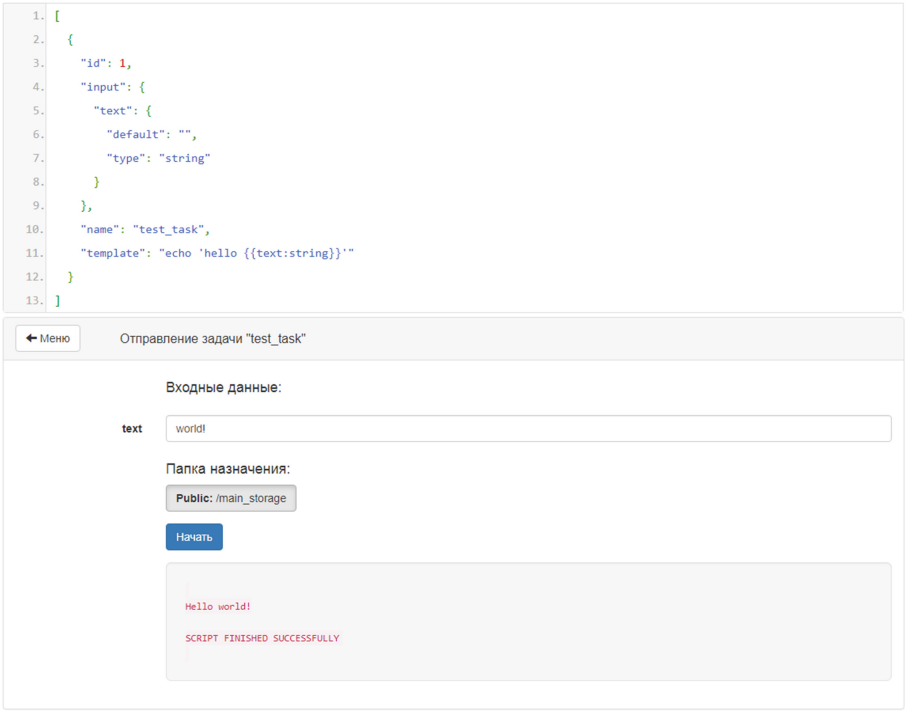

Fig. 2. Task definition, describing template for console program execution and web-interface, generated for this task

container hosting was prohibited, and the servers themselves were operating in the private network. When working with computing resources closed from external connections, often the only way to provide public access to TCP ports is reverse tunneling. This technique uses outbound connections to transfer data in the opposite direction and requires fine-tuning of the firewall to intercept (Fig. 2).

To organize the operation of reverse tunnels, the "Fast Reverse Proxy" [5] was used, which provides the client and server parts necessary for the organization of the TCP-over-TCP reverse tunnels. It allows you to open and control in detail the external access to services deployed on private resources that are closed by the firewall.

However, each tunnel requires a separate port on a public machine, which degrades the scalability of the system because of the need to create a separate set of tunnels for the services of each user. Another technique, reverse proxy, can be used to solve this problem. It is used to provide access to several web services under the guise of one, working, on the one hand, as a server responding to client requests, and, on the other, as a client, sending proxied requests to services (Figs. 3, 4, 5 and 6).

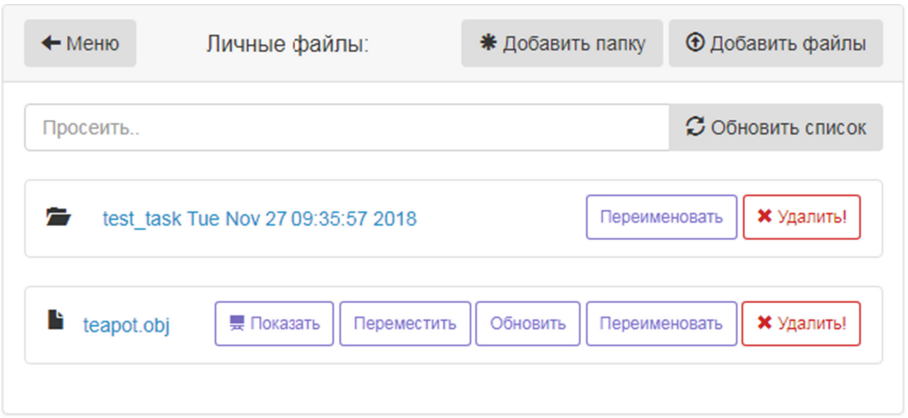

Fig. 3. Overview of a folder located in the shared storage in the web interface

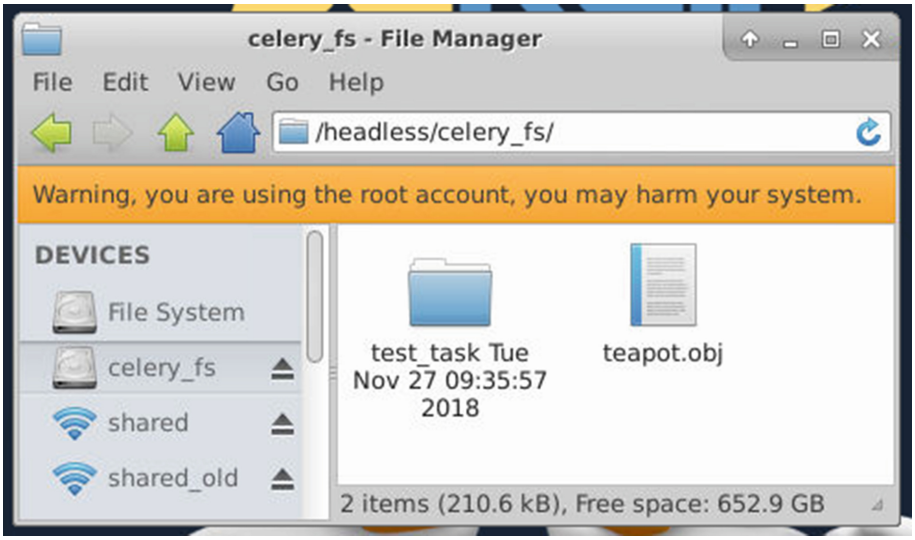

Fig. 4. Overview of the same folder in GUI

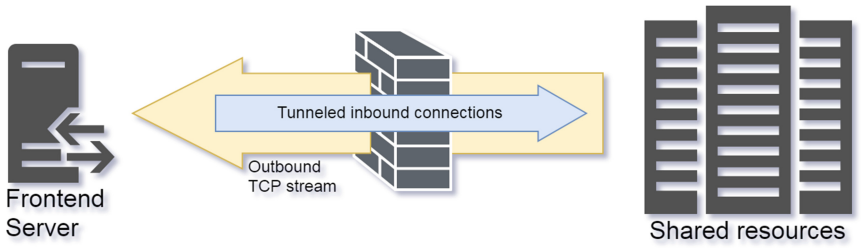

Fig. 5. Reverse tunnel workflow diagram

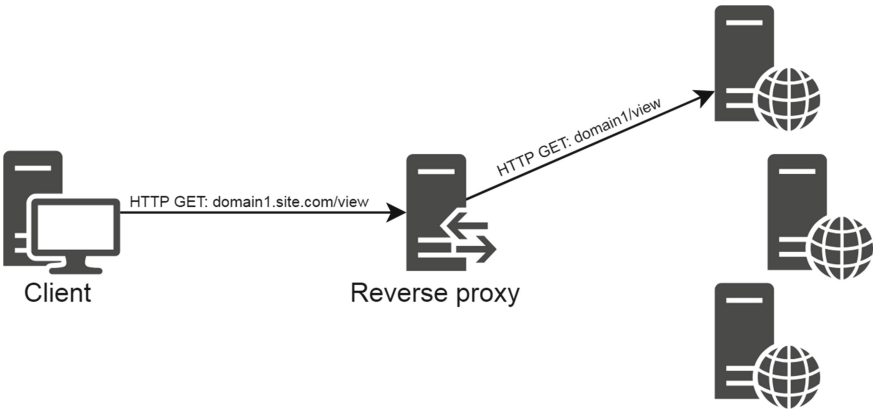

Fig. 6. Reverse proxy workflow diagram

Traefik [6] was chosen as the reverse proxy server. It allows its users to redirect HTTP and HTTPS requests and WebSocket connections based on their HTTP headers and supports the dynamic redefinition of request routing logic. To minimize the number of TCP ports required for the operation of services through a reverse tunnel, slightly different use of this technology is needed. Instead of redirecting to other machines, the reverse proxy will serve as an entry point for all requests passing through the reverse tunnel and redirect them to services running on the same machine through a loopback interface unregulated by the firewall.

Thus, connection to containers running on private host machines is possible through a single public server. However, access to each of the hosts is open through separate public TCP port of the corresponding FRP tunnel. To unify access to containers running on different machines, these public ports are hidden beside a single reverse proxy Traefik, operating according to the scheme described earlier. This allows users to access their work environment from a single URL, regardless of exact deployment location.

It is easy to see that manual control of all components of the system of such complexity would take an unreasonably long time. To automate this process, an admin panel and a system for sending configurations to system nodes were implemented. In order to simplify the user authorization logic, users are divided into groups defining the containers they use and the tasks available in the web interface. The admin panel, deployed on a public site, allows administrators to manage groups, user containers, add or remove system nodes. The broadcasting system starts and stops the containers on nodes and updates the logic of the routing of Traefik requests if it is necessary (Fig. 7).

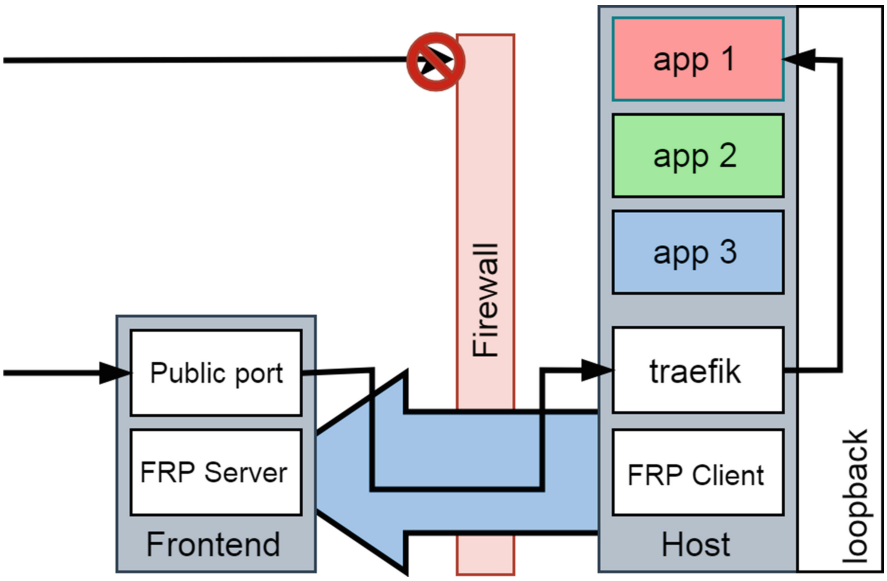

Fig. 7. Publishing system diagram

4 Evaluation

The described system of remote access to shared resources is based on a combination of several technologies: containerization using Docker, remote execution of tasks through Celery and bypassing firewalls using reverse tunneling. The advantages and disadvantages of the constructed solution arise from the features of these technologies and the method of their use in the system.

As an example, the combination of reverse proxy and reverse tunneling affects the speed of message transmission between the client and the server, compared to the use of a separate tunnel for each server, but allows more efficient use of available TCP ports on a public machine and thereby improves the scalability of the system.

The use of containerization technology reduces the overhead required to isolate the user environment, compared to virtual machines. Choosing Docker as a containerization platform allows usage of an extensive library of Docker images for container creation, but limits the ability of users to work with the kernel [7] and further slows down message transfer between the client and the server working in the container [8] (Fig. 8).

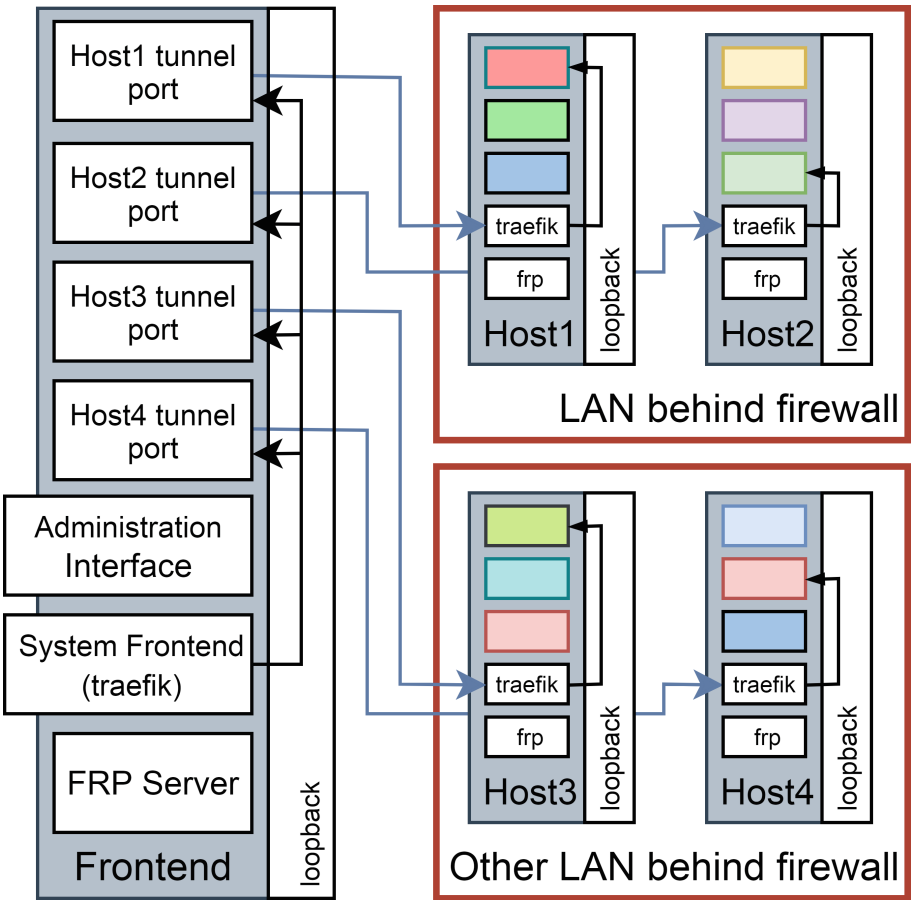

Fig. 8. Complete system structure overview

The usage of the Celery task queue to organize work with console applications allows users to choose execution environment but affects performance compared to the direct invocation of applications by the web server that provides the user interface. On the other hand, this difference is not noticeable in comparison with the total running time of the majority of applications. In addition, Celery makes it easier to monitor user activity, allows administrators to control access to task queues, and provides the opportunity for the future development of the system for remote execution of applications (Fig. 9).

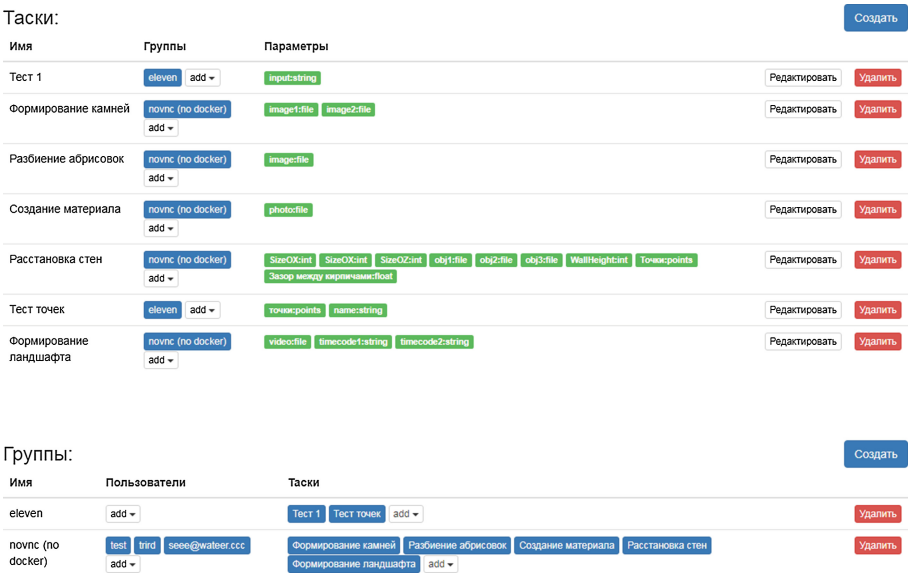

Fig. 9. System administration interface

5 Conclusion

The implemented system allows simplifying the process of remote use of shared computing resources by a large number of users. It is built taking into account the possibility of using distributed resources working in a private network without the possibility of establishing incoming connections. The system only requires shared computing resources to have a connection to the Internet and a single node with a public IP address.

Further development of the system can be associated with the use of alternative containerization platforms. Due to security risks, Docker usage may not be allowed on high-performance computing resources used in research projects. As an alternative containerization platform, Singularity is the most popular choice [9]. However, the use of network software in Singularity is inconvenient, since it does not manage container networks. Overcoming these difficulties and adapting the system to use Singularity can be the next step in its development.

Regardless of the choice of containerization platform used, both Docker and Singularity allow you to limit computing resources, such as the percentage of CPU or RAM available to applications running in the container. Integrating the management of these constraints into the administration interface would allow centralized control of users' resource usage. This feature may appear with future system updates.

Other system changes may be related to the publication system. In particular, the separation of host machines into private and public, without tunneling in the second case, will improve the speed of connection with services running on public machines.

Usage of this system in research projects makes remote use of shared resources easier for end users. For experienced users, direct use of the system GUI is possible, and for those less familiar with the console, a web interface that implements the functionality of a predefined set of console applications. Simplification of the workflow increases the efficiency of the use of computing resources in scientific projects.

Acknowledgments. This research was partially supported by the Russian Foundation for Basic Research grants (projects no. 17-29-04288). The authors would like to acknowledge the Reviewers for the valuable recommendations that helped in the improvement of this paper.

References

1. Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D.: SETI@ home: an experiment in public-resource computing. Commun. ACM **45**(11), 56–61 (2002)
2. Merkel, D.: Docker: lightweight linux containers for consistent development and deployment. Linux J. **2014**(239), 2 (2014)
3. TigerVNC vncserver (2019). https://tigervnc.org/doc/vncserver.html
4. Martin, J., Mannehed, S., Astrand, P., Ross, S.: noVNC: HTML5 VNC Client (2015)
5. fatedier/frp: A fast reverse proxy to help you expose a local server behind a NAT or firewall to the internet (2019). https://github.com/fatedier/frp
6. Traefik: The cloud native edge router (2019). https://traefik.io/
7. Bui, T.: Analysis of docker security. arXiv preprint arXiv:1501.02967 (2015)
8. Felter, W., et al.: An updated performance comparison of virtual machines and linux containers. In: 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE (2015)
9. Kurtzer, G.M., Sochat, V., Bauer, M.W.: Singularity: scientific containers for mobility of compute. PLoS ONE **12**(5), e0177459 (2017)
10. Iakushkin, O.O., Malevanniy, D.M., Degtyarev, A.B., Selivanov, D.A., Fatkina, A.I.: Texture generation for archaeological reconstructions (2018)
11. Iakushkin, O., Selivanov, D., Tazieva, L., Fatkina, A., Grishkin, V., Uteshev, A.: 3D reconstruction of landscape models and archaeological objects based on photo and video materials. In: Gervasi, O., et al. (eds.) ICCSA 2018. LNCS, vol. 10963, pp. 160–169. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95171-3_14
12. Iakushkin, O., Fatkina, A., Plaksin, V., Sedova, O., Degtyarev, A., Uteshev, A.: Reconstruction of stone walls in form of polygonal meshes from archaeological studies. In: Gervasi, O., et al. (eds.) ICCSA 2018. LNCS, vol. 10963, pp. 136–148. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95171-3_12