Published in Proceedings of the 17th IEEE International Symposium on Parallel and Distributed Computing (ISPDC 2018), Geneva, Switzerland, 25-28 June 2018, which should be cited to refer to this work. DOI: 10.1109/ISPDC2018.2018.00022

Distributed container-based evaluation platform for private/large datasets

Ivan Eggel
HES-SO
Institute of Information Systems
TechnoArk 3, 3960 Sierre, Switzerland
Email: ivan.eggel@hevs.ch

Roger Schaer
HES-SO
Institute of Information Systems
TechnoArk 3, 3960 Sierre, Switzerland
Email: roger.schaer@hevs.ch

Henning Müller
HES-SO
Institute of Information Systems
TechnoArk 3, 3960 Sierre, Switzerland
Email: henning.mueller@hevs.ch

***Abstract*—The rise of big data and artificial intelligence techniques such as deep learning has lead to an exponential increase in stored data in various fields, including medical imaging, genetics and financial trading. Sharing these increasing amounts of data for research is challenging, as privacy risks increase with the increased size of data. Physically moving very large datasets to researchers is inconvenient, as download or sending physical hard disks are not optimal. Research on sensitive data is often not possible, as sharing is not legal. The popularity of container-based technologies such as Docker has revolutionized the way applications are deployed, due to their self-sufficient, light-weight and portable nature. In this paper, we propose a novel distributed platform using containers for simple execution and evaluation of research applications on the data owner's infrastructure, bringing the algorithms to the data. This approach avoids the cumbersome transfer of large datasets and can help circumventing problems linked to non-shareable data by providing a sandboxed execution environment with read-only access to the data. At no point the data leave the data owner's site, giving researchers access to their evaluation results, only, and not the data themselves. The presented proof-of-concept confirms the feasibility of a distributed container-based evaluation platform for large and/or sensitive data. This has several advantages, including execution of code instead of submission of result files and availability of otherwise inaccessible data. The container architecture allows for minimal computational overhead, no software dependency management on the infrastructure, distributed runtime environment and isolation of processes from the underlying host system. A version addressing various identified architectural and security-related challenges has the potential to be deployed in a production setting and therefore allows researchers to gain insights from previously inaccessible data. One goal is to target hospitals with increasingly strong local infrastructure for storage and computation, needed for artificial intelligence based decision support (genetics and imaging).**

I. Introduction

Over the past few years, the cheap data storage and strong computation power in connection with artificial intelligence techniques such as deep learning have lead to an exponential increase in produced data in various fields such as health or the finance industry [1]. For instance, whole slide histopathology imaging allows to digitize biopsies with a resolution in the gigapixel range (dimensions of approximately 100'000 x 100'000 pixels) and a required disk space of several GigaBytes (GBs) per image. Whole Slide Imaging (WSI) is increasingly used also for clinical routine and many computerised WSI analysis methods are currently being developed to reduce the workload of pathologists. Thus, in the coming years hospitals are expected to produce an increasing number of WSIs with work flows becoming entirely digital and leading to important storage requirements [2], [3], [4]. Making such big data available for research has gained momentum in the medical data decision support field, as large-scale analysis has led to major research contributions and understanding of complex physical or biomedical phenomena and structures [5].

Big data analysis will likely play an even more important role as a driving force in medicine than in other industries. Digital medicine is now a reality, as the complexity of data, for example genomics, proteomics, metabolomics etc. can not be mastered manually anymore. Thus, a principal challenge for preclinical and clinical research is to get access to sufficiently high quality, informative data of large size [6], so modern artificial intelligence techniques such as deep learning can be trained and subsequently used to show performance [7].

Sharing large datasets of several TeraBytes (TBs) is a challenge on its own. Usually, it requires hard disks to be sent around through the postal service or providing the data on the Internet via a download mechanism, which can potentially take several weeks or even months, strongly depending on the available bandwidth. On the client side, researchers require high storage capacity, as well as a strong computing infrastructure [8], which is not always the case for Universities in lower resource countries, creating a strong digital divide. Beyond this, more dynamic datasets (quickly changing data items) can currently not be used efficiently, as the time to prepare a test collection is too long and when distributed the data are already outdated in some cases [9].

Another aspect that can make it extremely difficult to distribute datasets is the involvement of confidential data, for example of medical nature, which often can not leave the data owner's closed environment (e.g. a hospital). There is no conventional way of sharing such data with external researchers, except via an ethics approval and usually informed consent by the patient, followed by anonymization/pseudonimization [9], which is not trivial in case of very large data sets. As a consequence, researchers can be slowed by administrative procedures or a lack of data, meaning that the scientific discoveries are sometimes equally slowed.

Kaggle[1], who claims to be the world's largest community of data scientists, statisticians, and machine learning engineers, provides researchers with a platform for the evaluation of their machine-learning related algorithms on specific datasets provided by challenge organizers. It follows the traditional approach where participants download the dataset and then submit the results obtained by running their algorithms, usually in the form of a Comma-Separated Values (CSV) file. With this approach, the necessity of moving data around remains and this may become insurmountable when very large datasets are involved. DrivenData[2] is another platform following a similar strategy, where participants of competitions submit results files directly. Anjos et al. propose BEAT[3], a web-based platform for Open Science that allows not only submitting and evaluating results, but also defining modular toolchains and executing code directly on the platform [10]. The main limitation of the platform is that only a Python processing backend is currently available, therefore restricting the usage to researchers familiar with this language and its libraries. Moreover, the platform is oriented towards code and data sharing, limiting its applicability for usage with confidential data.

Multiple initiatives have tackled the problem of running evaluations on undistributable data. The main outcome, summarized under the Evaluation-as-a-Service (EaaS) paradigm, has the idea of keeping the data in a central place and allowing access to researchers in the form of an Application Programming Interface (API), Virtual Machine (VM) or other possibilities to ship executables [11], [12], [13]. As an example, the FP7 EU Project Visual Concept Extraction Challenge in Radiology (VISCERAL)[4] developed a cloud-based framework for experimentation on large medical datasets where participants would implement solutions to a specific task on VMs located in a cloud environment, effectively bringing the algorithms to the data. The organizers then ran the evaluation on the results produced by the participants [14], [15], [16]. Similarly to VISCERAL, Testbed for Information Retrieval Algorithms (TIRA)[5] is also a platform that allows participants to run their algorithms in VMs and submit their results for subsequent evaluation [17]. However, VMs are considered heavyweight as they include a full copy of an Operating System (OS) and other necessary binaries and libraries in order to run the code used for the analysis, along with emulating all hardware components of a computer and introducing overheads in accessing computational, network or storage resources.

Recently, container-based technologies such as Docker[6] have become increasingly popular, mainly in the software developer community and the Development and Operations (DevOps) software engineering culture. A Docker container is an instance of a container image that can be described as a lightweight, self-sufficient executable package of a piece of software that contains everything needed to run an application, including code, runtime and dependencies. This ensures that a container always runs the exact same way regardless of the execution environment. Compared to VMs, containers do not emulate the underlying hardware with a hypervisor but run directly as an isolated process on the same OS kernel as the host OS. This greatly reduces overhead, such as system set-up, disk space and boot up time, which makes them highly portable [18], [19]. In terms of performance, a container only adds negligible overhead compared to native execution [20], [21], [22], [23]. Originally, containers were mainly used for software development and deployment, however there is also a trend that tends to shift thinking away from traditional patterns towards more creative ways of how to further exploit the container technology in various domains, such as facilitating reproducibility in research [24], [25].

Efforts towards container-based evaluation have recently been made by CodaLab[7]. CodaLab Competitions is an open source framework for running competitions that also involves code submission. As soon as a participant submits code, it is run in a Docker container and a scoring program provided by the organizers evaluates the results produced by the executed code. A disadvantage of this approach, making CodaLab not a highly flexible solution, is the predefined container image in which the participant's code is run, not allowing for an entirely free choice of programming languages and technologies by the participant (rather the organizer's choice). Another downside of CodaLab is that datasets need to be uploaded to a cloud, making this solution possibly incompatible with confidential data.

In order to foster maximum usage of large and/or confidential data, two main aspects need to be addressed. First, the data must remain inside the data owner's site. Second, a portable and lightweight solution allowing external researchers to have their models evaluated on these data needs to be provided. To achieve this, the researchers' algorithms are executed directly on the data owner's internal infrastructure, bringing the algorithm to the data, therefore making them only visible to the algorithm itself and not any human.

In this paper we present a proof-of-concept of a distributed container-based evaluation platform for private/large datasets based on Docker with swarm mode (cluster). The prototype enables linking a Docker image (containing their analysis code) to an internal dataset and specifying constraints on the type of worker node in the cluster the users' code should be run on. After the execution of the container, the users have the possibility to have their results evaluated.

II. Methods

A. Platform architecture

The architecture of the proposed prototype consists of the following main components:

[1]http://kaggle.com/, as of 22.02.2018
[2]http://drivendata.org/, as of 22.02.2018
[3]http://beat-eu.org/platform/, as of 22.02.2018
[4]http://visceral.eu/, as of 22.02.2018
[5]http://tira.io/, as of 23.02.2018
[6]http://docker.com/, as of 22.02.2018
[7]http://codalab.org/, as of 06.02.2018

- A frontend user interface based on the open source Angular[8] web application platform that allows building scalable applications for desktops and mobile devices.
- A MySQL[9] database for storing all data related to participants, datasets, evaluation instances and other metadata linked to the platform.
- A Java-based RESTful backend linking the frontend to the database, developed using the Spring[10] application framework and its Spring Boot[11] solution for rapid application development and deployment.
- A Docker Swarm[12] cluster allowing execution of computation instances in a distributed and configurable manner. The native "Swarm mode" feature introduced in Docker v1.12.0 was used to create a cluster of Docker computation nodes.
- A Spring-based REpresentational State Transfer (REST) service allowing interaction with the Docker Swarm cluster. This service is tasked with calling the Docker Engine API[13] to enable starting, stopping and monitoring Docker instances within the Docker swarm. An existing library[14] was used to simplify communication with the Docker Engine API.

The choice of technologies was based on the need to quickly develop a robust and flexible platform. See Figure 1 for an overview of the platform architecture.

The Docker swarm is composed of 5 nodes, including the swarm leader (that also acts as a worker) and 4 additional worker nodes. The nodes are all servers located within the infrastructure of our research institute and possess different characteristics:

- the number of Central Processing Unit (CPU) cores varies between 12 and 48
- the amount of Random-Access Memory (RAM) varies between 128GB and 1024GB
- ne of the worker nodes is equipped with 2 NVIDIA Tesla K80 Graphical Processing Unit (GPU) Accelerators, and another has an NVIDIA GRID K1 graphics board

Since the nodes are heterogeneous, descriptive labels are added to them in order to allow users of the platform to define constraints to choose a specific type of node ("high memory", "high CPU", "GPU-enabled", etc.) for the execution of their computation instances.

Leveraging the NVIDIA GPUs installed in two of the used servers is achieved through the use of the "nvidia-docker" utility[15] that allows accessing the NVIDIA hardware and Compute Unified Device Architecture (CUDA) drivers installed on the host OS from within Docker containers.

[8]http://angular.io/, as of 02.02.2018
[9]http://mysql.com/, as of 02.02.2018
[10]http://spring.io/, as of 02.02.2018
[11]http://projects.spring.io/spring-boot/, as of 05.02.2018
[12]http://docs.docker.com/engine/swarm/, as of 05.02.2018
[13]http://docs.docker.com/develop/sdk/, as of 05.02.2018
[14]http://github.com/spotify/docker-client/, as of 06.02.2018
[15]http://github.com/NVIDIA/nvidia-docker/, as of 13.02.2018

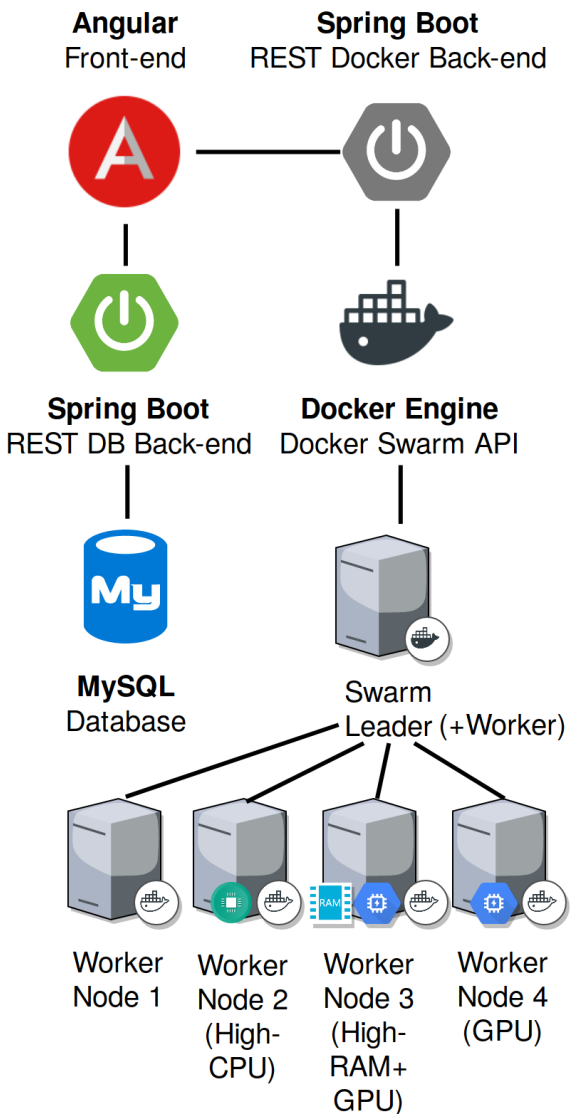

Fig. 1. Overview of the architecture of the developed platform.

Finally, to allow streaming of log messages from Docker containers to the frontend interface, the HyperText Markup Language 5 (HTML5) WebSocket API[16] was used in conjunction with a JavaScript library using the Simple Text-Oriented Messaging Protocol (STOMP)[17] protocol on the client side, while the server side was developed using the Spring framework's integrated WebSocket support. The server polls the Docker Swarm leader for new log messages every second and pushes the responses to an in-memory message broker which the client can subscribe to.

B. Computational task

A sample computational task was prepared in order to test and evaluate the platform. The task consists of a Python script

[16]http://websocket.org/aboutwebsocket.html/, as of 14.02.2018
[17]http://stomp.github.io/, as of 14.02.2018

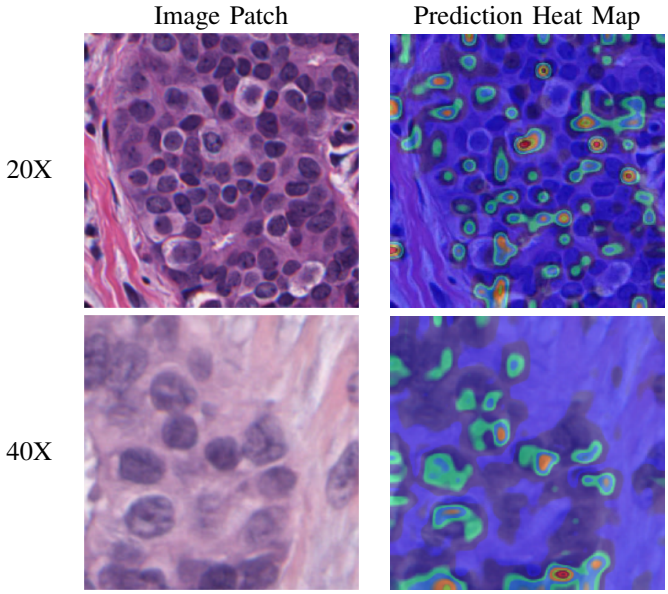

Fig. 2. Illustration of the computational task used for testing the platform. In the left column are two patches from a histopathology image at different magnification levels (20X and 40X). In the right column are the corresponding heat maps generated by the model trained in the computational task, which predict the average area of nuclei within the patch.

using the Keras[18] Deep Learning library with the TensorFlow[19] backend in order to train a model for automatically detecting the scale of histopathology image patches from the biomedical literature. It is inspired by [26] and works by predicting the average area occupied by cell nuclei within the patch to determine its magnification level. The code runs both on CPU-only systems, as well as hosts configured for GPU computation. Examples of input patches at two different magnification levels (20X and 40X) and the corresponding heat maps generated by the trained model are shown in Figure 2. The training process is performed on $\sim 23'000$ sample patches and a validation of the model is performed on $\sim 6'500$ samples. The training code was simplified to allow reasonable run times when using CPU computation: All parameters are fixed (no hyperparameter optimization is performed) and the number of epochs was limited to 5.

III. Results

The developed prototype is a basic proof-of-concept of a distributed container-based evaluation platform for private/large datasets. This section gives a detailed view on the proposed system and presents the results obtained from the comparison of execution times of a sample algorithm executed in a container and in a native environment on different architectures.

A. Platform description

In order to make the entire application as portable and extensible as possible, all platform components were deployed as micro services in separate Docker containers. The Angular web client is the only point of interaction between the researcher and the internal backend components such as the database and the REST APIs.

1) Instance creation: On the home page of the web interface, the user is presented with a list of available datasets, including brief descriptions. As soon as a dataset is selected, the user gets redirected to a detailed view containing all necessary information on how to conduct the analysis, e.g. a list of contained files and their respective location. The next step for the researcher is to create a Docker image containing the analysis code, as well as all dependencies needed to run the executable. In order to minimize the entry barriers for researches unfamiliar with Docker, we provide a basic image they can extend, however it is possible (and often needed) to create a Docker image from scratch. Currently, a user has to upload their image to the public Docker Hub[20], a cloud-based registry service that is able to host Docker images and that supports discovery and automatic download of images from within a local Docker client. As soon as the image is pushed to the registry, the user can create a new so-called instance. In the context of our prototype, an instance represents an entity linked to a dataset, a Docker image and an execution environment. Creating an instance consists of the provision of a name, the Docker image that should be run in a container, the dataset to use, and finally the execution node type, with the latter specifying the type of machine within the Docker cluster environment. For instance, thanks to the distributed Docker environment, users have the choice to run their code on a high-CPU node for computation intensive or high-RAM for memory demanding tasks. In addition, we also provide a GPU option, ideal for deep learning scenarios, which greatly benefit from the massively parallel architecture. Docker in swarm mode also acts as a load balancer, trying to equally distribute the load among the different worker nodes and to reschedule a node's task on another node as soon as it becomes unavailable.

2) Instance execution: The creation of an instance does not imply direct execution of the code, it rather creates the necessary entries in the backend database in order for the system to know how to run the container. Within the instance-detail view (see Figure 3), the researcher is able to run the instance, factually meaning the execution of the Docker container within the Docker cluster via the call of the REST Docker backend, which in turn communicates with the Docker Engine API. As soon as the container is launched, the user is provided with the current state of the container, e.g. "RUNNING" if the code is being executed, "FAILED" when the container exited with an error code or "COMPLETE" in case the code has completed successfully. This helps the user to get minimal feedback and therefore take necessary measures if something went wrong. The interface also supports the display of a running container's "stdout" and "stderr" streams by printing them to the log console (implemented via an HTML5 WebSocket). However, this feature must not be enabled for highly confidential data

[18] http://keras.io/, as of 14.02.2018
[19] http://tensorflow.org/, as of 14.02.2018
[20] http://hub.docker.com/, as of 22.02.2018

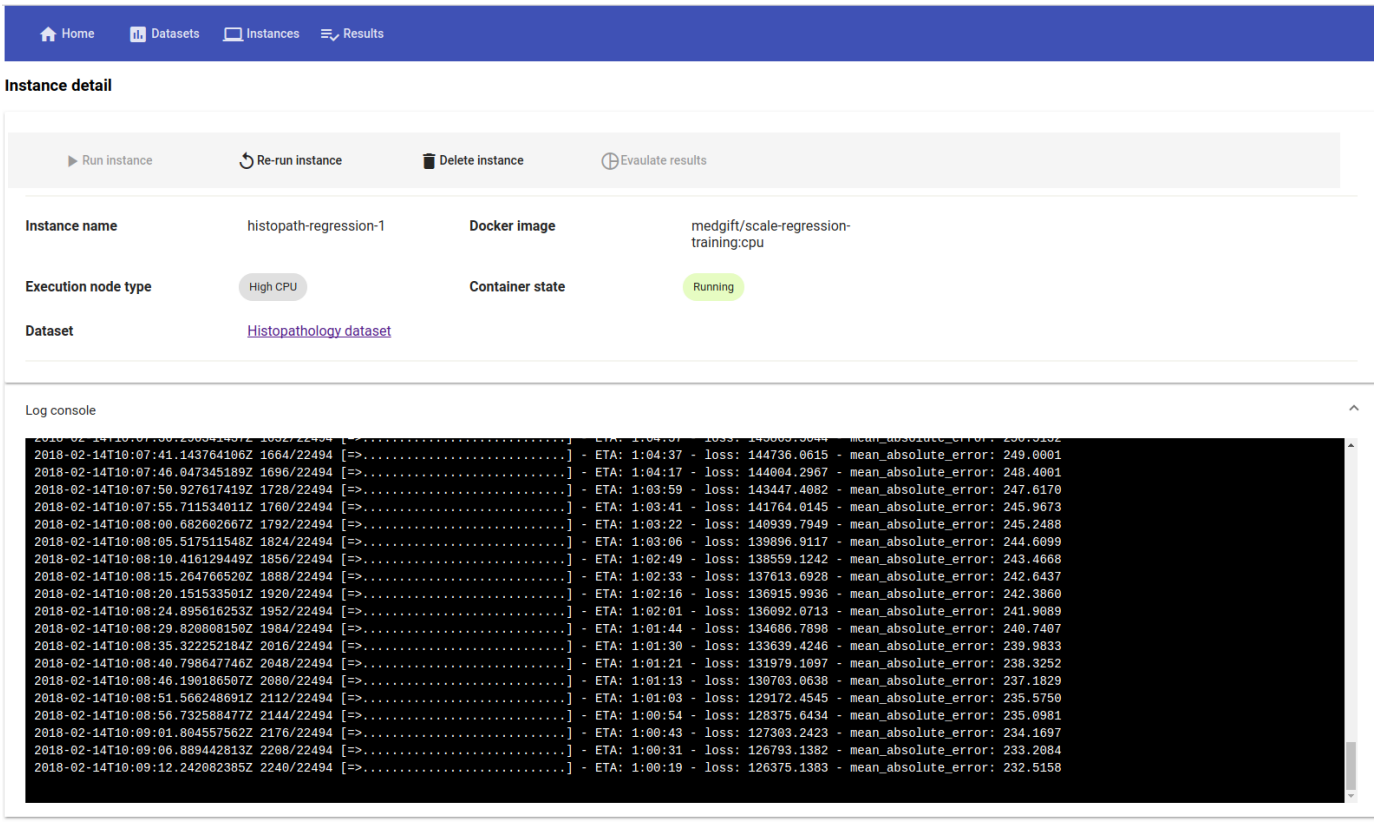

Fig. 3. Instance-detail view of the platform user interface.

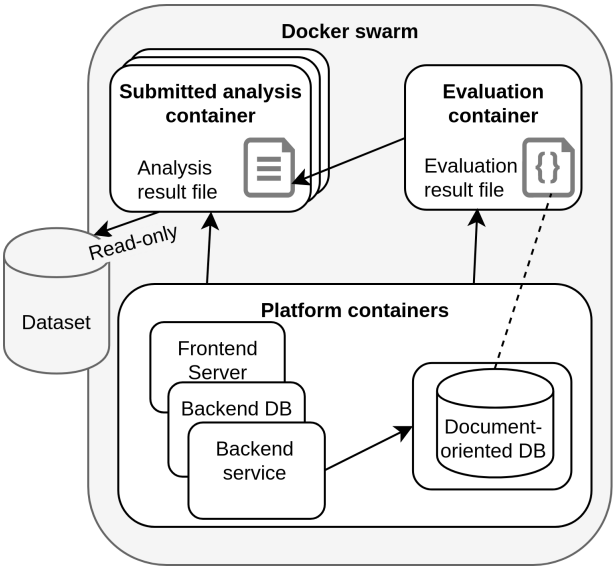

Fig. 4. Interaction of platform-, analysis- and evaluation containers.

as the user's code could transfer sensitive information to the output streams.

3) Dataset access: In order to correctly deploy an analysis container, it is essential to link it to the respective dataset, since by default Docker containers are not allowed to access the host file system. To achieve such an access, the container is assigned a read-only bind-mount from the corresponding dataset directory on the host to the "/data" directory inside the container. With this approach, users have access to the dataset by referring to the "/data" base path in their code. Internally, all datasets are located on Network-Attached Storage (NAS) drives with Network File System (NFS) access from all worker nodes in our code but architectures can slightly vary.

4) Node execution type: The user's choice of the node execution type is implemented via the placement constraints provided by Docker's swarm mode. Worker nodes in the cluster are labelled with specific metadata, such as "node.type=highcpu", designating the type of machine. Upon deployment of the container, the placement constraint is passed as an argument, making Docker take care of which node the container is placed on. In case no appropriate nodes exist, the deployment fails, which is necessary as often the analysis code is optimized for a certain architecture. In order to make placement constraints work for containers requiring GPUs, a workaround had to be supplied by adding the the GPU device IDs to the default Docker runtime in the Docker engine configuration file on the GPU nodes. Additionally, activating the NVIDIA wrapper for Docker swarm mode via a configuration file was necessary to allow running GPU-enabled code in our Docker cluster. Thanks to these changes, containers demanding GPUs can be deployed by providing

a specific ”generic resource” argument to the Docker Engine API.

5) Evaluation: The main concept for the automatic evaluation part is shown in Figure 4. One requirement is that the code running in the container must write the analysis results into a specific format to a predefined file path (e.g. ”/results.csv”). The second precondition is that each dataset must be linked to an existing Docker image containing the code for the evaluation of the analysis results produced for that dataset. This concept empowers the result evaluation to run in a container in the swarm and thus ensures high encapsulation and portability of the evaluation component. This also makes it possible to easily add or replace an evaluation method. Furthermore, evaluation worker nodes can be appointed by labelling them accordingly. Upon deployment of the evaluation container a designated evaluation node is then assigned as execution environment by passing a special placement constraint as an argument. Consequently, as soon as the container has finished running the analysis code on the dataset, the user can launch the evaluation, which in turn triggers the deployment of a new container derived from the predefined evaluation image. However, in order to get the results into the evaluating container, they first need to be copied from the user's completed container. This is a cleaner and more secure solution than the direct writing of the results into a host mapped directory by the code. After the provision of the result file, the evaluation container is able to run its code against those results and finally produce the evaluation scores. Those in turn will be written into a JavaScript Object Notation (JSON) file inside a host mapped directory linked to a NAS drive. Going from there, the system transfers the file contents to a document-oriented database (such as MongoDB[21]) and finally incorporates the evaluation scores into the user's instance-detail view.

B. Containers vs. standard execution

In order to analyze the potential overhead of using the Docker Swarm platform instead of directly executing a computational task (CPU or GPU-based) on a host, an experiment was run to compare the runtime of a given task on the 5 cluster nodes, once using direct execution and a second time using the developed platform, through the Docker Swarm cluster. The results are shown in Figure 5. Experiments on CPU nodes were run using the default multi-threading configuration of Keras and TensorFlow. For the GPU runs, a single GPU was used in both cases (Tesla K80 and GRID K1 nodes) for computation.

IV. Conclusions

This paper presents a proof-of-concept for a distributed container-based evaluation platform for private/large datasets, as they occur in medical data analysis. We describe the problems solved, analyze container execution times, discuss the limitations of the current solution and give an outlook to future developments.

[21]http://mongodb.com/, as of 22.02.2018

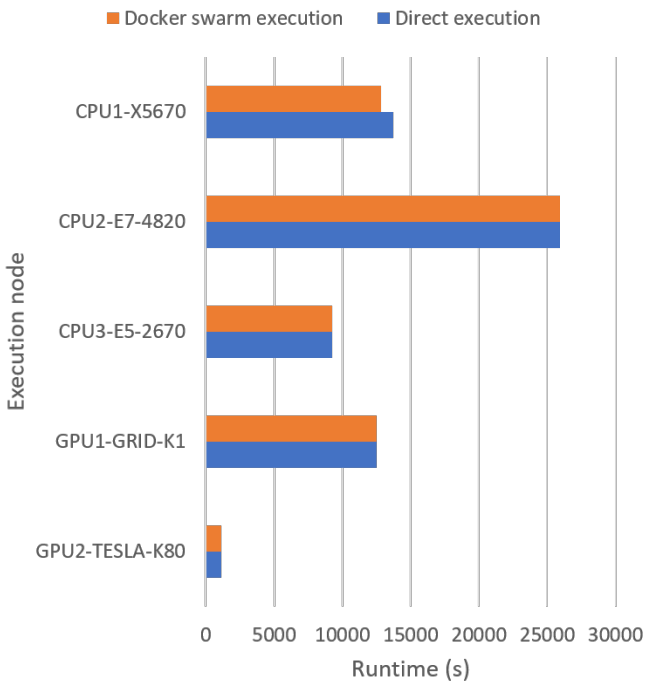

Fig. 5. Comparison of run times between direct execution of a Deep Learning computation task and execution through the platform using Docker Swarm.

A. Problems solved

Users are able to create their own Docker images that are then run as containers analysing confidential data directly on the data owner's infrastructure. On the one hand this avoids the distribution of large data and all the inconveniences this causes. On the other hand, only the executed code inside the container comes into contact with the dataset. The researchers are granted access to evaluation results alone, implying that at no point in time they see the data themselves first-hand. The distributed architecture of the system based on Docker's swarm mode allows the application of advanced concepts such as load balancing amongst worker nodes, as well as container placement constraints in order to define on what kind of node the code should be run. This opens up the opportunity to deploy containers that rely on specific hardware and thus also allows the execution of GPU-oriented code, which is often becoming a requirement due to the rising popularity of deep learning techniques. Another benefit of providing an evaluation platform that supports the submission of containers is the free choice of programming languages and frameworks by researchers, effectively avoiding tight coupling of a specific technology with the ability to perform analysis, as it would be the case with direct code submission. The ”algorithm-to-the-data” paradigm shift, in combination with the use of software containers represents a novelty that has the potential to make large/confidential data more easily accessible for research, particularly in the medical domain.

B. Analysis of containers vs. standard execution

The comparison of execution times between a standard direct execution and an execution using Docker showed no significant difference between both options in terms of runtime. The minor differences in runtime between the standard and Docker executions can be attributed to the varying global system load on the computation nodes. Docker therefore does not introduce any overhead for a typical CPU- or GPU-intensive task. It is interesting to note that a CPU-based execution on a host with 2 Xeon® E5-2670 v2 processors was faster than an execution on a system using an NVIDIA GRID K1 graphics board for computation, most probably due to the limited number of CUDA computation cores available in one GRID K1 GPU.

C. Limitations and proposed improvements

Despite many benefits of the proposed approach, we would also like to address a few shortcomings and introduce possible solutions.

1) Image repository: A current weakness of the system is to rely on Docker Hub as an image hosting platform. Docker images of non-paying customers are implicitly made publicly available. However, researchers might not be willing to publicly share their code and do not necessarily want to pay a fee in order to keep their images private. On the other hand, the implementation for the system to access these private images on Docker Hub risks not to be trivial and might even need to internally store the users' credentials for Docker Hub. A better solution would be the introduction of an internal private registry server, including user authentication allowing users to push their images. This keeps the images (and thus the contained code) hidden from the outside world but enables usage by the internal platform.

2) Quotas / Scheduling: Considering the possibility of a large number of participants using the system and depending on the available worker nodes, the Docker swarm cluster can become overloaded by running analysis containers. To counteract this, the platform can establish quotas on memory and CPU usage. This is already natively supported in Docker and also at the same time allows the use of a scheduler that leaves a container in a 'PENDING' state as long as insufficient resources are available in the cluster.

3) Security implications: Letting researchers run their code in a sandboxed environment on confidential data may be seen as a safe approach compared to execution in a traditional setting. However, in order to guarantee the privacy and safety of private datasets, several potential security risks and loopholes bust be carefully studied and addressed accordingly.

- In order to prohibit the users' code to send out sensitive information about the dataset, all incoming and outgoing network traffic not concerning internal Docker swarm communication must be blocked. This can be achieved by using separate networks for swarm management related traffic and application related traffic, where the latter would then be completely disabled for the containers running the analysis code.
- Since multiple Docker containers may share the same Linux kernel on a given worker node in the swarm, it is crucial that kernel security patches are applied as soon as they become available. This reduces the likelihood of a compromised system due to arbitrary container code execution.
- The security guidelines and best practises published by Docker must be followed accurately and the version of Docker should be kept to the latest stable version, in order to minimize Docker-internal security risks.
- There needs to be a procedure to make participants sign a legally binding End User Agreement in which they recognize the confidentiality of the data and agree not to misuse the data.

4) Usage of Docker images: Requiring users of the system to develop their own Docker images might lower the usage potential of the proposed platform, as learning the concepts of container technology translates to an important learning curve. Despite this fact, we believe that using containers is a highly portable way of code deployment and in this case it might justify the means. The large number of free Docker images available on Docker Hub can be reused and built upon, greatly reducing the effort needed to create a custom image containing the analysis code.

D. Future developments

The proof-of-concept presented in this paper has built a solid foundation for further improvements towards a more mature solution regarding distributed container-based evaluation systems for confidential data. In case all aforementioned improvements, notably the ones concerning security, can be reliably improved, a prototype could be deployed and tested in a staging setting and then moved to a real-life test-bed. After that, a deployment in production environments such as a hospital is considered, possibly also on distributed data that are stored in several institutions. This however needs solid engineering effort on a high expertise level and requires external security audits. Additionally, legal and political barriers are possible. An open source release of the project can help build a stable/mature version and at the same time reveal if there is enough general interest in the topic.

ACKNOWLEDGMENT

This work was partly supported by the RCSO project CaDEnCe 80566 of the HES-SO.

REFERENCES

[1] J. Wood, T. Andersson, A. Bachem, C. Best, F. Genova, D. R. Lopez, W. Los, M. Marinucci, L. Romary, H. Van de Sompel *et al.*, "Riding the wave: How Europe can gain from the rising tide of scientific data," *European Union*, 2010.

[2] M. N. Gurcan, L. E. Boucheron, A. Can, A. Madabhushi, N. M. Rajpoot, and B. Yener, "Histopathological image analysis: A review," *IEEE Reviews in Biomedical Engineering*, vol. 2, pp. 147–171, 2009.

[3] O. Jimenez-del-Toro, S. Otálora, M. Andersson, K. Eurén, M. Hedlund, M. Rousson, H. Müller, and M. Atzori, "Analysis of histopathology images: From traditional machine learning to deep learning," in *Biomedical Texture Analysis: Fundamentals, Applications, Tools, and Challenges Ahead*, A. Depeursinge, O. S. Al-Kadi, and J. R. Mitchell, Eds. Elsevier, 2017.

[4] L. Pantanowitz, P. N. Valenstein, A. J. Evans, K. J. Kaplan, J. D. Pfeifer, D. C. Wilbur, L. C. Collins, and T. J. Colgan, "Review of the current state of whole slide imaging in pathology," *Journal of Pathology Informatics*, vol. 2, 2011.

[5] M. P. Washburn, D. Wolters, and J. R. Yates, "Large–scale analysis of the yeast proteome by multidimensional protein identification technology," *Nature Biotechnology*, vol. 19, no. 3, pp. 242–247, 2001.

[6] C. Auffray, R. Balling, I. Barroso, L. Bencze, M. Benson, J. Bergeron, E. Bernal-Delgado, N. Blomberg, C. Bock, A. Conesa, S. Del Signore, C. Delogne, P. Devilee, A. Di Meglio, M. Eijkemans, P. Flicek, N. Graf, V. Grimm, H.-J. Guchelaar, Y.-K. Guo, I. G. Gut, A. Hanbury, S. Hanif, R.-D. Hilgers, A. Honrado, D. R. Hose, J. Houwing-Duistermaat, T. Hubbard, S. H. Janacek, H. Karanikas, T. Kievits, M. Kohler, A. Kremer, J. Lanfear, T. Lengauer, E. Maes, T. Meert, W. Müller, D. Nickel, P. Oledzki, B. Pedersen, M. Petkovic, K. Pliakos, M. Rattray, J. R. I Màs, R. Schneider, T. Sengstag, X. Serra-Picamal, W. Spek, L. A. I. Vaas, O. van Batenburg, M. Vandelaer, P. Varnai, P. Villoslada, J. A. Vizcaíno, J. P. M. Wubbe, and G. Zanetti, "Making sense of big data in health research: Towards an EU action plan," *Genome Medicine*, vol. 8, no. 1, p. 71, 2016.

[7] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[8] H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi, "Big Data and Its Technical Challenges," *Communications of the ACM*, vol. 57, pp. 86–94, 2014.

[9] A. Hanbury, H. Müller, and G. Langs, Eds., *Cloud–Based Benchmarking of Medical Image Analysis*. Springer International Publishing, 2017, vol. 6.

[10] A. Anjos, L. El-Shafey, and S. Marcel, "BEAT: An Open-Source Web-Based Open-Science Platform," *arXiv:1704.02319 [cs]*, 2017.

[11] F. Hopfgartner, A. Hanbury, H. Müller, N. Kando, S. Mercer, J. Kalpathy-Cramer, M. Potthast, T. Gollub, A. Krithara, J. Lin, K. Balog, and I. Eggel, "Report on the evaluation–as–a–service (eaas) expert workshop," *ACM SIGIR Forum*, vol. 49, no. 1, pp. 57–65, 2015.

[12] A. Hanbury, H. Müller, K. Balog, T. Brodt, G. V. Cormack, I. Eggel, T. Gollub, F. Hopfgartner, J. Kalpathy-Cramer, N. Kando, A. Krithara, J. Lin, S. Mercer, and M. Potthast, "Evaluation–as–a–service: Overview and outlook," *ArXiv*, vol. 1512.07454, 2015.

[13] J. Lin and M. Efron, "Evaluation As a Service for Information Retrieval," *SIGIR Forum*, vol. 47, no. 2, pp. 8–14, 2013.

[14] A. Hanbury, H. Müller, G. Langs, and B. H. Menze, "Cloud–based evaluation framework for big data," in *Future Internet Assembly (FIA) book 2013*, ser. Springer LNCS, A. Galis and A. Gavras, Eds. Springer Berlin Heidelberg, 2014, pp. 104–114.

[15] A. Hanbury, H. Müller, G. Langs, M. A. Weber, B. H. Menze, and T. S. Fernandez, "Bringing the algorithms to the data: cloud–based benchmarking for medical image analysis," in *CLEF conference*, ser. Springer Lecture Notes in Computer Science, 2012.

[16] O. Jimenez-del-Toro, H. Müller, M. Krenn, K. Gruenberg, A. A. Taha, M. Winterstein, I. Eggel, A. Foncubierta-Rodríguez, O. Goksel, A. Jakab, G. Kontokotsios, G. Langs, B. Menze, T. Salas Fernandez, R. Schaer, A. Walleyo, M.-A. Weber, Y. Dicente Cid, T. Gass, M. Heinrich, F. Jia, F. Kahl, R. Kechichian, D. Mai, A. B. Spanier, G. Vincent, C. Wang, D. Wyeth, and A. Hanbury, "Cloud–based evaluation of anatomical structure segmentation and landmark detection algorithms: VISCERAL Anatomy Benchmarks," *IEEE Transactions on Medical Imaging*, vol. 35, no. 11, pp. 2459–2475, 2016.

[17] T. Gollub, B. Stein, S. Burrows, and D. Hoppe, "TIRA: Configuring, Executing, and Disseminating Information Retrieval Experiments," in *2012 23rd International Workshop on Database and Expert Systems Applications*, 2012, pp. 151–155.

[18] Z. Li, M. Kihl, Q. Lu, and J. A. Andersson, "Performance Overhead Comparison between Hypervisor and Container Based Virtualization," 2017, pp. 955–962.

[19] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2015, pp. 171–172.

[20] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. F. D. Rose, "Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments," in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, Feb. 2013, pp. 233–240.

[21] P. D. Tommaso, E. Palumbo, M. Chatzou, P. Prieto, M. L. Heuer, and C. Notredame, "The impact of Docker containers on the performance of genomic pipelines," *PeerJ*, vol. 3, 2015.

[22] J.-W. Park and J. Hahm, "Container-based cluster management platform for distributed computing," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 2015, p. 34.

[23] C. Arango, R. Dernat, and J. Sanabria, "Performance evaluation of container-based virtualization for high performance computing environments," *arXiv preprint arXiv:1709.10140*, 2017.

[24] J. Cito and H. C. Gall, "Using docker containers to improve reproducibility in software engineering research," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, 2016, pp. 906–907.

[25] C. Stelly and V. Roussev, "SCARF: A container-based approach to cloud-scale digital forensic processing," *Digital Investigation*, vol. 22, pp. S39 – S47, 2017.

[26] S. Otálora, O. Perdomo, M. Atzori, M. Andresson, M. Hedlund, and H. Müller, "Determining the scale of image patches using a deep learning approach," in *IEEE 15th International Symposium on Biomedical Imaging*, ser. ISBI 2018. IEEE, Apr. 2018.