I. INTRODUCTION

Digital methods have become commonplace in modern science. Their popularity is amplified by the recent establishment of the so-called data-driven science, which is expected to provide scientific insights by answering concrete scientific questions. Other researchers might, nevertheless, benefit from using it to solve similar problems or for the validation of the results by applying the software to a different set of data [1]. Sharing software solutions to facilitate such sharing.

II. DISTRIBUTED INFRASTRUCTURE

Forschungszentrum Jülich GmbH, Jülich, Germany

Jedrzej Rybicki and Benedikt von St. Vieth

Sharing services (e.g. new data analysis tools) between users and also touch on the problem of sharing services between infrastructures, for instance to facilitate cross-disciplinary exchange. Our goal was to increase the sustainability of the developed research software by implementing in a project is only understandable and deployable by the authors. In this paper we will address the problem of sharing services (e.g. new data analysis tools) beyond the rigid set of services towards flexible Software-as-a-Service (SaaS) solutions. We share the initial experiences presented.

Unfortunately, it happens often that the software implemented in a project is only understandable and deployable by the authors. In this paper we will address the problem of sharing services (e.g. new data analysis tools) beyond the rigid set of services towards flexible Software-as-a-Service (SaaS) solutions. We share the initial experiences presented.

I. INTRODUCTION

Digital methods have become commonplace in modern science. Their popularity is amplified by the recent establishment of the so-called data-driven science, which is expected to provide scientific insights by answering concrete scientific questions. Other researchers might, nevertheless, benefit from using it to solve similar problems or for the validation of the results by applying the software to a different set of data [1]. Sharing software solutions to facilitate such sharing.

II. DISTRIBUTED INFRASTRUCTURE

Forschungszentrum Jülich GmbH, Jülich, Germany

Jedrzej Rybicki and Benedikt von St. Vieth

Sharing services (e.g. new data analysis tools) between users and also touch on the problem of sharing services between infrastructures, for instance to facilitate cross-disciplinary exchange. Our goal was to increase the sustainability of the developed research software by implementing in a project is only understandable and deployable by the authors. In this paper we will address the problem of sharing services (e.g. new data analysis tools) beyond the rigid set of services towards flexible Software-as-a-Service (SaaS) solutions. We share the initial experiences presented.

Unfortunately, it happens often that the software implemented in a project is only understandable and deployable by the authors. In this paper we will address the problem of sharing services (e.g. new data analysis tools) beyond the rigid set of services towards flexible Software-as-a-Service (SaaS) solutions. We share the initial experiences presented.

I. INTRODUCTION

Digital methods have become commonplace in modern science. Their popularity is amplified by the recent establishment of the so-called data-driven science, which is expected to provide scientific insights by answering concrete scientific questions. Other researchers might, nevertheless, benefit from using it to solve similar problems or for the validation of the results by applying the software to a different set of data [1]. Sharing software solutions to facilitate such sharing.

II. DISTRIBUTED INFRASTRUCTURE

Forschungszentrum Jülich GmbH, Jülich, Germany

Jedrzej Rybicki and Benedikt von St. Vieth

Sharing services (e.g. new data analysis tools) between users and also touch on the problem of sharing services between infrastructures, for instance to facilitate cross-disciplinary exchange. Our goal was to increase the sustainability of the developed research software by implementing in a project is only understandable and deployable by the authors. In this paper we will address the problem of sharing services (e.g. new data analysis tools) beyond the rigid set of services towards flexible Software-as-a-Service (SaaS) solutions. We share the initial experiences presented.

Unfortunately, it happens often that the software implemented in a project is only understandable and deployable by the authors. In this paper we will address the problem of sharing services (e.g. new data analysis tools) beyond the rigid set of services towards flexible Software-as-a-Service (SaaS) solutions. We share the initial experiences presented.
able to audit them to verify if they are correct, the deployment descriptions can be used to build a dynamic Software-as-a-Service solution, which allows interdisciplinarity, bears one more use case for sharing Infrastructures. This use case reinforces the previously mentioned requirement of making software deployment without repeating the installation process.

The potential users of research software from different disciplines will have to invest some time to understand the software. The time should not be increased if the documentation is seldom up-to-date and does not cover all the corner cases encountered in the process described above is quite laborious, let us now reflect on how software reusing is typically achieved nowadays. In the first step the scientific software is only useful if it can be applied to a machine running somewhere in the e-Infrastructure. The process is conducted either locally on the researcher's machine or (in a more modern fashion) on a remote environment, additional libraries, and actual software. The context of all, images are black boxes: one has to trust the creator of the image.

One way to create trustworthy and reproducible virtual machines (VM) is by making the user understand how to install, and deploy the required software, dependencies, and operating system. This results in a high overhead in terms of both time and effort. The software is only usable and result in the need for a re-installation. Since the documentation is seldom up-to-date, the further problem with the required software, dependencies, and operating system results in a high overhead in terms of both time and effort. The software is only useful if it can be applied to usable and result in the need for a re-installation. Since the documentation is seldom up-to-date, the high development pace of cooperative science leads to the need for periodic updates of the image: for instance to apply new security patches. The further problem with images is, that they are infrastructure-specific: they include a complete software stack comprised of operating system and virtual machines suffer, however, under hacking or destruction by malicious third parties. Virtual images do not include some malicious software or are not misconfigured so that running instances can be easily hacked or destroyed by malicious third parties.

Clouds include their own image repositories but moving images between the infrastructures i.e. between different clouds, remains an unsolved problem. A problem with contextualization solution provided in the particular cloud. When an image is created and uploaded to a cloud image repository it can be instantiated by other users to obtain running version of the application. The later is often subsumed under the term: contextualization. When an image is created and uploaded to a cloud image repository it can be instantiated by other users to obtain running version of the application. When an image is created and uploaded to a cloud image repository it can be instantiated by other users to obtain running version of the application. The later is often subsumed under the term: contextualization. When an image is created and uploaded to a cloud image repository it can be instantiated by other users to obtain running version of the application. When an image is created and uploaded to a cloud image repository it can be instantiated by other users to obtain running version of the application. When an image is created and uploaded to a cloud image repository it can be instantiated by other users to obtain running version of the application. When an image is created and uploaded to a cloud image repository it can be instantiated by other users to obtain running version of the application. When an image is created and uploaded to a cloud image repository it can be instantiated by other users to obtain running version of the application. This results in some overhead in terms of both time and effort.

One example for this is the deployment of a web platform, for instance a software often used for by virtual machines while accounting for special properties of the given cloud. The later is often subsumed under the term: contextualization. When an image is created and uploaded to a cloud image repository it can be instantiated by other users to obtain running version of the application. This results in some overhead in terms of both time and effort. The software is only useful if it can be applied to usable and result in the need for a re-installation. Since the documentation is seldom up-to-date, the high development pace of cooperative science leads to the need for periodic updates of the image: for instance to apply new security patches. The further problem with images is, that they are infrastructure-specific: they include a complete software stack comprised of operating system and virtual machines suffer, however, under hacking or destruction by malicious third parties. Virtual images do not include some malicious software or are not misconfigured so that running instances can be easily hacked or destroyed by malicious third parties. Clouds include their own image repositories but moving images between the infrastructures i.e. between different clouds, remains an unsolved problem. A problem with contextualization solution provided in the particular cloud.

Let us now reflect on how software reusing is typically achieved nowadays. In the first step the scientific software is only useful if it can be applied to usable and result in the need for a re-installation. Since the documentation is seldom up-to-date, the high development pace of cooperative science leads to the need for periodic updates of the image: for instance to apply new security patches. The further problem with images is, that they are infrastructure-specific: they include a complete software stack comprised of operating system and virtual machines suffer, however, under hacking or destruction by malicious third parties. Virtual images do not include some malicious software or are not misconfigured so that running instances can be easily hacked or destroyed by malicious third parties. Clouds include their own image repositories but moving images between the infrastructures i.e. between different clouds, remains an unsolved problem. A problem with contextualization solution provided in the particular cloud. When an image is created and uploaded to a cloud image repository it can be instantiated by other users to obtain running version of the application. When an image is created and uploaded to a cloud image repository it can be instantiated by other users to obtain running version of the application. When an image is created and uploaded to a cloud image repository it can be instantiated by other users to obtain running version of the application. When an image is created and uploaded to a cloud image repository it can be instantiated by other users to obtain running version of the application. When an image is created and uploaded to a cloud image repository it can be instantiated by other users to obtain running version of the application. When an image is created and uploaded to a cloud image repository it can be instantiated by other users to obtain running version of the application. This results in some overhead in terms of both time and effort.

One way to create trustworthy and reproducible virtual machines is by making the user understand how to install, and deploy the required software, dependencies, and operating system. This results in a high overhead in terms of both time and effort. The software is only useful if it can be applied to usable and result in the need for a re-installation. Since the documentation is seldom up-to-date, the high development pace of cooperative science leads to the need for periodic updates of the image: for instance to apply new security patches. The further problem with images is, that they are infrastructure-specific: they include a complete software stack comprised of operating system and virtual machines suffer, however, under hacking or destruction by malicious third parties. Virtual images do not include some malicious software or are not misconfigured so that running instances can be easily hacked or destroyed by malicious third parties.
argue that there will be no need to manage the setup new instance will be spun off. Researcher will not configuration changes in one instance, for the modified quickly, and cheaply create new instances. Previous results and they will be discarded afterwards. We management when it would be possible to simply, be willing to spend e.g. to verify the researcher's hypothesis instances of given services. These hand, will be rather used to create quickly disposable changes are then automatically loaded on all relevant machines. The scientific software sharing, on the other efficient sharing of scientific software. In this section we machines. They constitute the perfect basis for an infrastructure. For this purpose, they allow for the configuration of software running in a distributed way. It lowers the overhead in terms of both cgroups namespaces and others to isolate independent application containers. The latter feature permits rudimentary provenance technologies (and usage). Applications running with Docker are called containers, they are created from images. Images should public ones, like the system kernel. It lowers the overhead in terms of both solutions, Docker images do not include a guest operating system, but it has problems with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of the Listing 1 show how to with the fact that the same software is available with the different package, path, and service naming. To prepare deployment descriptions for all the supported platforms. The first lines of theListing 1. Puppet example for deploying Apache httpd, A - BASED VIRTUALIZATION

III. Container -ba ed I All A i on
Infrastructures like DARIAH-DE provide resources which the researchers can use for their projects. We present our prototype solution which runs on low-level compute resources already available in the DARIAH-DE infrastructure. Our goal was a system which provides the researchers with an instance of the software she requires in just one request through a web interface. This service will offer all the benefits of the solution, this service will offer all the benefits of the SaaS approach (i.e., very low effort is required to use scientific software shared by other researchers) whilst require long-lasting software runs. This is hard to perform on the local researcher's machine. Fortunately, e-Infrastructures like DARIAH-DE provide resources which are not always the best option. One reason for remote deployments is efficiency: remote instances can run close to the data they are interested in, pulling and starting requires just one command.

The workflow described above can be repeated multiple times. Software is made available. Moreover, the roles can be distributed or handed-over. Both repositories allow for multiple admins. Hence it is not required to have just one registration of organizations. Such organizations can have additional information about the software. The developer can describe the typical applications or provide a link to a more extensive documentation.

Software developed in one part of DARIAH might be passed over to other national efforts. There is also an effort, DARIAH-DE [2], where the described work was done. On Fig. 1 we present how such a software sharing life cycle is triggered. Figure 1. Proposed software sharing life cycle

### IV. IMPLEMENTING ME A HOSTING D

The process of making scientific software available requires long-lasting software runs. This is hard to perform on the local researcher's machine. Fortunately, e-Infrastructures like DARIAH-DE provide resources which are not always the best option. One reason for remote deployments is efficiency: remote instances can run close to the data they are interested in, pulling and starting requires just one command.
Fig. 2 depicts the architecture of the implemented dynamic Software-as-a-Service. It includes lots of components loosely coupled with each other by a messaging bus. Messaging is used for exchanging information between components in an asynchronous way. In production we use an amqp-based product (RabbitMQ [14]). Let us briefly discuss the roles and responsibilities of the single elements of our architecture.

The Facade is the entity facing the end-users. It is an entry point from which the commands (like create a new instance of a given software product) are issued. The Facade is stateless and it is possible to run multiple Facades. They can offer interfaces of different types, e.g.: an html-based for browser access or a json-based programmatic API.

The Facade has access to the messaging bus and a read-only access to the Store. Store is a system-wide cache with information about:

1. types of service registered in the system,
2. instances running in the system.

The content of the cache is updated by the Store Updaters. These entities subscribe for information about creation of new instances and registration of new types. Upon reception of such messages, Store Updaters write the information in the Store. It should be stressed that the content of the Store itself can be lost as it will be periodically republished. The Store is solely used to speed-up the response times.

The Workers embody the main functionality of the system. Those are the entities which can manage instances of given service types (e.g. instances of eXist or neo4j databases). Each Worker upon its creation (and later periodically) informs other components about its capabilities (i.e. supported service type) and its current state (i.e. managed instances). Workers receive commands from users (via Facade) through the messaging system.

The critical part of our system is the communication between the components. It is dynamic, asynchronous, and subject-based. We require the possibility to change the system capabilities during runtime. In particular, to add new software to the portfolio of the offered Software-as-a-Service. This is done by adding (or removing) Workers. To become active in the system, Worker needs access to the messaging bus. The incoming Worker publishes its "specialty" (i.e. instance type that it can provide) and subscribes for messages in this instance-type-queue (like create new instance, delete instance). As can be seen on Fig. 2, for each instance type supported by the system, there is a separate queue. When a user requires a new instance of a given type it sends a message to the respective queue (via Facade). The message is routed to the proper Worker, it creates an instance and sends a response (via messaging) with details required for connecting to the just-created instance. Such details typically include IP address, protocol and port, user name and password, etc. Workers republish their state periodically (i.e. instance types that they support and state of managed instances). Each worker has a dedicated queue for the incoming commands and all workers share a common response queue (the info queue on Fig. 2).

We have created a prototype of the dynamic Software-as-a-Service called DARIAH Meta Hosting. The implementation was done in Python and uses a set of frameworks and well-known products: Flask for the front end, MongoDB as storage, and RabbitMQ for messaging. We offer two interfaces: one is a human-friendly web site, the other is a programmatic json-based API. We envision the later one to be used to script automatic deployments of software e.g., in data processing workflows.

One feature of our design was the division of the system into small, autonomous parts connected by messaging. The advantage of such a design was the ability to change the parts independently, for instance to scale up. We use Docker for deploying single parts and the Docker-based orchestration tool docker-compose [15] to create test deployments including all the parts required. Docker-Compose is also able to scale out the system by spinning off additional instances. In this self-test, the solution we use for the software sharing was successfully applied for the software we have implemented. Our prototype is deployed on a OpenStack Infrastructure-as-a-Service cloud which we offer in DARIAH-DE. So far the complete system is placed in one data center but it should be possible to extend it beyond this one entity. This would only require to make the messaging accessible from the outside and deploy additional workers in different data centers.

For our prototype we have implemented a generic Docker worker. The deployment of Docker-based instances is usually conducted in the same fashion, Figure 2.
regardless of what software is contained in the image. Hence the Worker is initialized with a name of the Docker image containing the software it will be responsible for. We create an instance of Worker for each software package offered in the dynamic Software-as-a-Service. To scale out the system it is also possible to have multiple workers offering the same type of software. The requests will then be divided among the workers in a Round-robin fashion, allowing for load balancing.

At this stage it should be stressed, that the proposed architecture is extensible. The Workers abstract the technology used for creating the instances. It would be perfectly possible to write a worker which would use Puppet-based deployment descriptions or any other technology that might come around in the future.

V. CONCLUSION

The motivation for this paper was the challenge of scientific software sharing. A solution for that problem is crucial for the further development of reproducible data-driven science. We have argued that the problem is many fold. It involves both technical and social challenges. In this paper we have focused on the former ones and firstly reviewed the technologies which could be used to leverage the software sharing. Subsequently we have presented a workflow for sharing software in the context of e-Infrastructures like DARIAH-DE which ensures high grade of trustworthiness. In the second part of our paper we have shown how the presented software sharing solution can be used to implement a dynamic Software-as-a-Service system. It enables a quick and easy deployment of the software on IaaS resources available in the e-Infrastructure. The proposed architecture is extensible and we have also shared some experiences gained during the implementation and operation of a working prototype.

ACKNOWLEDGMENT

The work has been supported by DARIAH-DE, which is partially funded by the German Federal Ministry of Education and Research (BMBF), fund number 01 G1110A-M.

REFERENCES