EPiC Series in Computing

Volume 78, 2021, Pages 98–107

Proceedings of the European University Information Systems Conference 2021

Moving the sciebo Sync and Share Cloud Service to State-of-the-Art Cloud Technology

Holger Angenent, Marcel Wunderlich, Raimund Vogl
Westfälische Wilhelms-Universität Münster, WWU IT, Germany
holger.angenent@uni-muenster.de, wunderlich@uni-muenster.de,
rvogl@uni-muenster.de

Abstract

The campuscloud sciebo is a widely used service for higher education in the federal state of North Rhine-Westphalia (NRW) in compliance with German data protection legislation. The project started in February 2015. Currently, there are over 200.000 registered users. Modernizing the infrastructure for future demands is one of the current major challenges of sciebo. Due to evolved technology, a modernisation of the setup became necessary. Not only for scalability, but also for the closer integration with other services, the shift to a Kubernetes-based platform was an obvious step. Today, the modernized setup consists of two synchronously mirrored server sites, each with a 5 PB net capacity Spectrum Scale file system and two Kubernetes clusters running ownCloud software.

1 Project History

The campuscloud sciebo – reported at EUNIS before (Rudolph et al., 2016; Vogl et al., 2015) – started in early 2015 after a conceptual design phase of two years. Today, the number of registered users has grown to over 200,000 (see predictions in Vogl, Angenent, Rudolph, Stieglitz, & Meske, 2016). The main drivers of the project were and still are the need for a secure file syncing and sharing solution and large on premise data storage capacities for researchers.

An evaluation of available software solutions in 2013/2014 led to the decision for ownCloud, because it was covering all features required for the project. At the time, ownCloud was uniquely positioned as an open source solution. In the meantime, however, the fork Nextcloud and other open source solutions appeared on the market. For several reasons like the growing focus of ownCloud on stability, their roadmap and the effort of a migration, sciebo stayed with ownCloud.

For the first project phase of five years (2015-2020), a consortium was formed as a legal framework for the operation of sciebo. The consortium as the operator of the sciebo service was replaced by the University of Muenster in the course of updating the contract framework for GDPR, now being the only supplier of the service, with all other participating institutions acting as customers.

S. Bolis, J.-F. Desnos, L. Merakos and R. Vogl (eds.), EUNIS 2021 (EPiC Series in Computing, vol. 78), pp. 98–107

End users now conclude the terms of service no longer with their own institution, but directly with the University of Muenster, which enables greater flexibility and coherence of usage terms.

Some effort was necessary to enable all participating universities to join the Shibboleth-based authentication and authorization infrastructure, operated by German research network (DFN-AAI), which is required for the envisioned self-enrolment portal. During the first project phase this portal had been extended with additional features as the invitation for external guest users and the creation of so-called project boxes that offer larger accounts for those with higher storage demand.

The centre of sciebo is a classical LAMP-stack which runs ownCloud. Currently, there are three hosting sites, one at the University of Bonn, one at the University Duisburg-Essen and one at the University of Muenster. Each site consists of an IBM Spectrum Scale cluster for storage, a MariaDB Galera cluster, a fleet of webservers and assorted services for session persistence. There are some central services that are hosted in Münster and that are accessible from all sites, such as the user authentication and the integrated OnlyOffice service for online office document collaboration.

Abiding the KISS principle, deployment and management happens mainly via Ansible and ssh, avoiding complex interactions as much as possible. This has yielded in a very reliable and stable deployment.

2 sciebo ng – The Next Generation of sciebo

The industry has moved forward since the initial conception of sciebo some seven years ago, and new best practices have found their way into production: Version controlled configurations and deployments, review processes, automated testing and vulnerability scans – things that once have been wishful thinking are becoming requirements, not for developers only, but also for service operators. An ever increasing demand for feature velocity from our own users as well as the quest for more interoperability in the scientific community pushes the limits in adaptability and maintainability of the old deployment. With a central service such as file storage, usability and user experience depend not only on availability, but also on low latency. This requires modern monitoring solutions, which not only tell the services' health status, but allow more quantified measurements.

These factors require a more effective deployment strategy, standardized interfaces and APIs exposing operational functionality not only to operators, but also developers of associated services and automation tools. There are several technology stacks enabling this move, among which Kubernetes stands out as a comparatively lightweight solution: Building containers with dockerfiles comes natural and is far more transparent than the provisioning of virtual machines with other solutions. As containers are just Linux processes, it is still possible to gain introspection in the behaviour of involved components with familiar tools.

For sciebo ng (the technological next generation rehaul) we devised a Kubernetes platform that serves as a thin layer between the hardware and the application itself. Building upon the existing expertise in the team and playing well along with the IBM Spectrum Scale storage, the bare metal nodes are provisioned via xcat. Ansible is used to deploy the necessary tools for Kubernetes, like kubeadm, a container runtime and some minor adjustments, which might change during cluster lifetime. Kubernetes itself was installed with kubeadm. Inside the Kubernetes cluster, Prometheus operator, rook-ceph and NGinx-ingresses and HAProxy-loadbalancer were deployed for monitoring. This allows bootstraping a whole site quickly and might even serve as a starting point for a generalized administrative platform of webservices or as addition on existing hardware.

The long term vision of ownCloud of delivering a stable and scalable product with their next generation ownCloud Infinite Scale (oCIS) still is best suited for the future of sciebo. After the general availability of oCIS, sciebo ng will be migrated to this software stack which will give the performance of sciebo ng a big boost.

2.1 Cloud Native as a Challenge

A well-known buzzword is “cloud-native”, a term summarizing several principles which make an application suitable to be run on Kubernetes. Those are principles as statelessness, microservice architecture and rollback capabilities. These were of no concern or even diametral to deployment goals (e.g. avoiding microservice sprawl) at the time of sciebo’s first inception. OwnCloud (and consequently its fork Nextcloud) were not conceptualized as cloud-native services. Because of this, we had to put considerable effort into getting started in this new environment. In some instances, this even was beneficial for the first-generation-sciebo-deployment. For example, the ephemeral nature of Kubernetes pods made errors in session persistence apparent, whereas the stability of the existing deployment hid those bugs from our view.

Now, we have reduced the toil of setting up an ownCloud running on multiple servers with its own database cluster to a handful of kubectl and helm commands. We run two physically separate sites in Muenster, with the IBM Spectrum Scale filesystem mirrored synchronously. Moving an ownCloud installation from one site to the other is as fast as waiting for DNS caches to flush.

3 General problems solved

We first want to discuss the general, abstract challenges and problems involved and solved with the deployment of sciebo on Kubernetes and then see how this plays out in concreto.

End-users often require new features. The common pace for releasing new features is set by big tech companies. Even though awareness for data sovereignty has increased, there often is just a short time window before users switch to commercial cloud solutions. Since cloud services rely on network effects to be useful and accepted, this is a situation we want to avoid.

3.1 The Need for Higher Automation

Additional features often do not require changes to the original application, but collocation of additional services. This, of course, breaks with homogeneity and requires a more dynamic approach to managing services. Scalability is another issue which requires to step away from very static deployments. Moving away from vertically scaled applications usually involves clustering and sharding, which also leads to an increase in services to be managed. Both of these forces are also reflected in the general evolution of software, so that in general collections of services have become preferred over monolithic applications.

On the internal side, there are two main driving forces. One is the need for higher automation. Shell scripting works fine for deployment of long living homogeneous services, however, more dynamic environments involving heterogeneous services with shorter life-cycles require unified APIs to program against, so that services can be notified and monitoring can be kept up to date.

The other force is more difficult and more important. It is based on the observation of Joel Spoelsky, who stated that distributed version control systems (DVCS) have been the "the biggest advance in software development technology in the [past] ten years" (Spoelsky, 2010).

No other technology has enabled collaboration to a comparable extent. The adoption of DVCS and associated practices for deployment of systems is generally accepted as best practice. There is an acknowledged need for a single source of truth for configuration and package versions, as well as for a way to keep colleagues informed about these versions and configuration changes. Beneath the surface of these very obvious benefits, there also is a need for continuous development of the systems being run. Not only is it necessary to continually perform site reliability engineering to make sure errors do not occur twice, or at least will be caught by monitoring, but one also has to proactively work on the infrastructure to keep it ready for what there is to come. Despite the general consensus on

the validity and necessity of these practices, often subsumed under umbrella terms like DevOps, the actual change in methodology is often met with resistance.

While there are certainly some factors which are more related to the culture amongst system operators, we have found a very practical reason for this resistance: More traditional deployment of systems via shell access, scripts and package managers just does not play along well with version control and modern workflows. Configuration directories themselves are subject to change by operating system updates, so tracking these and one's own changes to specific configurations is cumbersome and error prone. Furthermore, actual packaging in most distributions is not easily managed via versioned files, but requires usage of specific tools.

It is necessary to bridge this gap between the DVCS and the servers themselves. We distinguish three ways to accomplish this, according to the ambient infrastructure required:

1 tools deploying software and configuration to servers, like Ansible, Puppet and Chef,

2 container orchestration tools, like Kubernetes, Airship and Nomad,

3 tool deploying machines that are leveraging infrastructure as a service (IaaS) platform, such as Terraform or Pulumi.

At sciebo, we already have used Ansible for the deployment of the first iteration of the platform to great success. However, it is too static to deal with a multitude of services. Broadly speaking, the first category of tools does help only with deployment but does not serve as a control plane for running services.

As Kubernetes has become the best supported and most widely adopted solution for container orchestration over multiple machines, we here find a very suitable solution to the general problems described above. The wide adoption opens up the possibility of adding new services without too much customization. Moreover, it does not only allow a very dynamic deployment, but serves as an operating system for the whole cluster, enabling us to gain insight and control in an automatable way. Finally, it not only bridges the gap between the DVCS and the running system, but even opens up possibilities for automated testing and deployments.

4 Specific problems solved and architectural overview

We now delve deeper in the specifics of the sciebo ng architecture. The first part outlines building a Kubernetes platform from the bird's eye view. The second part deals with some more specific approaches to standard problems. In general, we found a lot of introductory tutorials. However, there was very little information on how to put it all together. Consequently, we assume that readers will be able to find information on the basic terminology such as helm charts, resources and individually mentioned tools, if necessary, on their own.

Following the analogy of Kubernetes being an operating system for the whole cluster, what we are operating might be described as a Kubernetes from scratch, similar to Linux from scratch, which clearly is distinct from preconfectioned distributions. Since then, several Kubernetes distributions came into existence, and it is to be expected that these at some point will supplant the process of building one's own cluster. However, to this day, automating the cluster setup around the kubeadm tool is still the industrial standard for on-premises-deployments.

4.1 Low level deployment

As mentioned before, at the lowest level we deploy our bare metal nodes with xcat, because it is the supported way to install nodes with IBM Spectrum Scale storage attached. We can also build upon man-decades of experience with this tool. Networking has to be done at this level, too, as the proprietary drivers necessary to run our 100 Gbit/s network do not integrate as nicely as open source drivers with the common Linux tools. This stage of deployment is rather static. We achieved our main goal of reproducibility of node installations. One can view this as layer zero, as it is not necessary in setups less dependent on exotic hardware and software.

A schematic view of the hardware and network setup can be found in figure 1. Identical hardware is distributed to two server sites for optimal redundancy. On each site, a 5 PB of Spectrum Scale storage, frontend nodes for the Kubernetes cluster, a loadbalancer and an administrative node serve as the basis for the setup. The frontend nodes can access the storage via a 100 Gbit/s Inifiniband RDMA

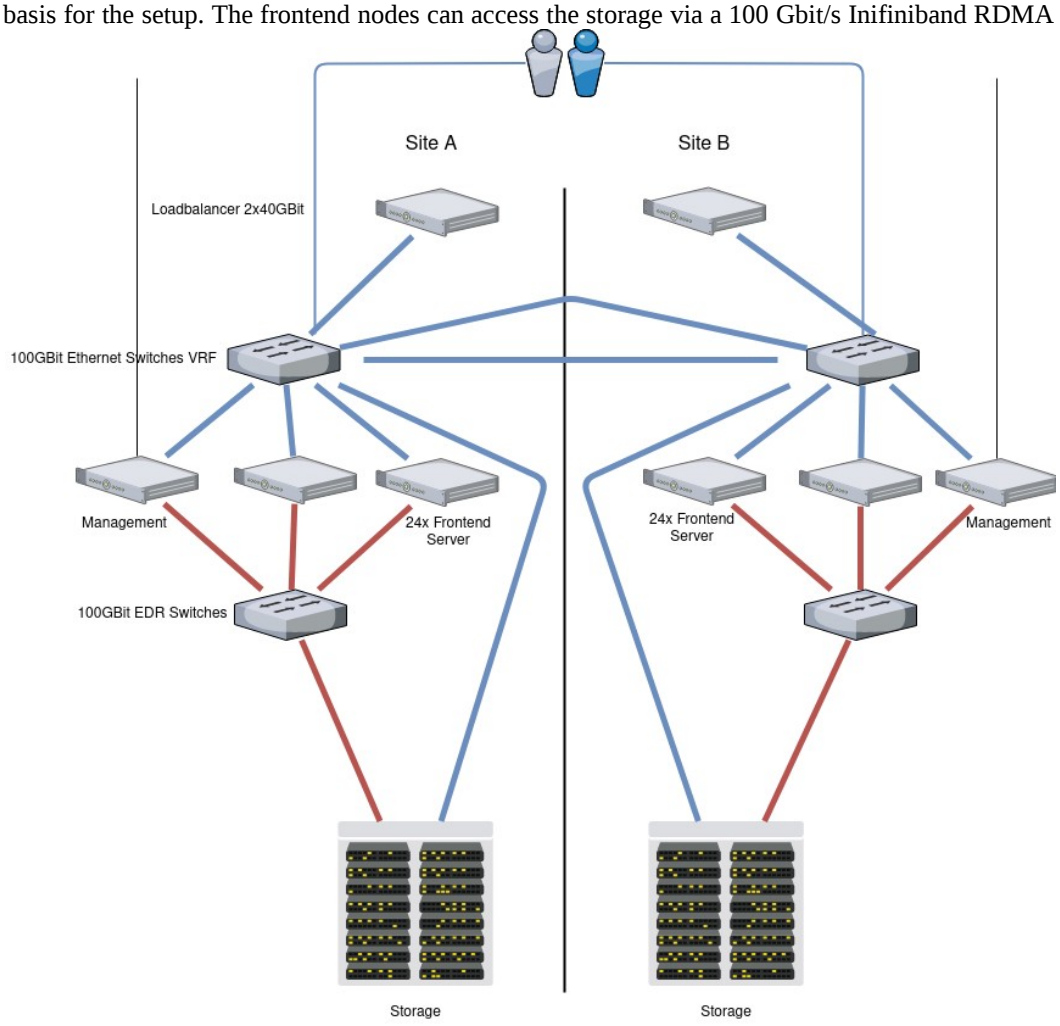

Figure 1: Hardware setup: Identical hardware in two separate server rooms aims at maximal redundancy. Each site has a 5 PB spectrum scale storage that is coupled via a cross-site 2x100G ethernet connection. Inside of each site, a 100 G Infiniband connection is used for fast RDMA storage access. 24 servers on each site are hosting the Kubernetes clusters. Ingress of connections is done via a pair of loadbalancers. Two management servers serve as entry point for administrative work. All fast network connections (100G ethernet and Infiniband) are redundant connections with switch pairs.

network, while the applications communication and inter-site communication is performed via a 100 Gbit/s Ethernet.

However, at the inception of the platform, most open source storage solutions did not have the comfort and performance we are already used to from IBM Spectrum Scale and usually lacked the ability to synchronously replicate across two sites. The closest competitor was and still is EOS, the file system developed by CERN. However, the complexity of EOS was not manageable for us at the time, since it involves running many different services, whereas the IBM Spectrum Scale is run as an appliance. In light of the operational burdens that proprietary software brings with itself, and the enabling power of Kubernetes in running a multitude of services, this will certainly be a point of thorough consideration for future generations of the sciebo platform. Especially as metadata performance in multi-petabyte scenarios necessitate certain architectural changes down the road that might impact the comfortable appliance character.

The next and rather first layer of our platform are OS-level package installations for all tools required. This layer is deployed via Ansible. Among the standard toolbox we like to have on systems, and some volume provisioning with lvm2, this encompassed specifically the container runtime engine, kubeadm, kubelet, as well as keepalived to for high availability of the Kubernetes control plane and the internet facing loadbalancers. Furthermore, we purge the systems from packages that interfere with operations, such as network manager and firewall. The container runtime engine cri-o, kubeadm, kubelet and kernel are version pinned. This allows us to keep the base installation of the nodes updated in a controlled matter, without fearing to break the system due to incompatible versions. In this state we then can install the Kubernetes base with kubeadm.

The second and certainly most interesting layer is what we would consider our Kubernetes distribution. It consists of the ingress network, volume provisioning for assorted services and monitoring of the whole setup. When viewing Kubernetes as the operating system of the cluster, the previous layer just gave us a way to manage processes across several computers. However, processes without networking, storage and monitoring are rather useless and uninteresting in most instances.

The first ingredient for this is volume provisioning. When processes are allowed to spawn on any system in a cluster and are short-lived, there has to be an automated way to provide volumes for them. In principle it could be possible to use the IBM Spectrum Scale, which is mounted on every node, but this has several considerable drawbacks, mostly related to making sure each process has only access to specific designed areas in an automated fashion and enforcing quotas. Usually, whenever one deploys a service which also needs some persistent storage space, a persistent volume claim (PVC) is created along the way. These can be consumed by volume provisioners, which then make sure such an appropriate volume exists and will be mounted to the pods. Such provisioners exist for a multitude of different storage backends. However, most of them come with several limitations, making them only third grade choice. A canonical choice are ceph and rook-ceph, which can be viewed as a ceph-distribution designed for Kubernetes. We install rook-ceph using the community supported helm charts.

This has two benefits for us: First of all, we have a running ceph cluster, including monitoring endpoints and provisioners, up in a matter of minutes. Second, by it running in the cluster, we do not have to deal with version pinning on the base OS, which is somewhat painful due to the restrictions placed on us by the proprietary hardware stack. As can be seen in figure 2, this means that all frontend nodes have two storage areas available, the ceph storage and Spectrum Scale.

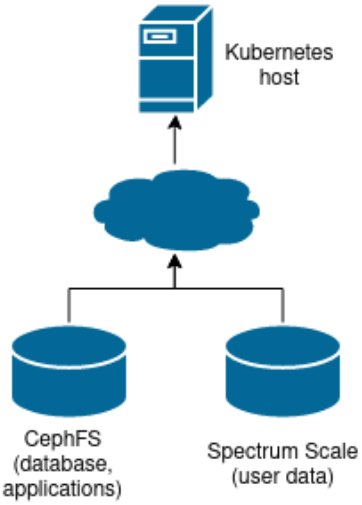

Figure 2: Storages mounted on each Kubernetes. The Spectrum Scale holds the user data while the application data and databases reside in the even faster CephFS

The second ingredient to this layer is the network ingress. This layer does leave the most scope for design and is inherently tied back to the ambient network of the cluster. The general problem being solved here is the translation of hostnames into routes leading to the correct backend pod. This job is done by so called ingress controllers, which ingest ingress resources and are basically reverse proxies that perform name-based virtual hosting. However, a step not found often in documentation, is that traffic has to get to these ingress controllers in some way in the first place. In the simplest form, this could be achieved by scheduling the ingress controller pods on a node on the network edge, which can be accessed from the outside. Since on the old platform, TLS termination was distributed across all servers of the webserver fleet and now is being concentrated on the ingress nodes, we decided to put these behind a simple HAProxy loadbalancer, which then runs in Kubernetes on the edge node and is redundant via keepalived on the node. As we use NGinx as ingress controller, we had to do some config tuning for the open file descriptors with the number of cores of the node. This leads to frequent crashes on more generously equipped servers with a double digit number of cores.

4.2 Monitoring

The third and last ingredient to this layer is the monitoring stack. Here we simply rely on the Prometheus operator community helm charts as time series backend, Loki for logs and Grafana for displaying both, as well as alerting via various channels. Just as PVC resources are used to connect storages to pods and ingress resources are used to route traffic from the outside world to backend pods, ServiceMonitor resources are used to expose metrics to Prometheus. There are huge operational benefits associated with this monitoring setup. By using Prometheus and Grafana, it is easy to create new dashboards and persist them in source control. More importantly, we can not only collect and view metrics, but with Prometheus query language it is possible to create derived metrics, which often give a lot more insight than just the raw numbers. Finally, metrics and logs are now unified. When collaborating with other people on problems across sites, there is no more searching for certain information on different systems, but queries for specific metrics and even derived metrics work across sites. In particular, almost all open source projects already expose most of their metrics in a easily ingested way.

This concludes the rather general layer two of our setup. Based upon a thin unified setup of the base install, kubeadm for cluster installation and the community driven projects rook-ceph and Prometheus operator, it is rather straightforward to setup a working Kubernetes cluster at any scale. The main difficulty is then fitting the ingress infrastructure into the ambient network and to the use case. Find an overview of the setup in figure 3.

4.3 Application Software deployment

To complete our setup, we now discuss layer three, the actual deployment of sciebo, as well as some of the more practical challenges we encountered when deploying ownCloud in a clustered setup on Kubernetes.

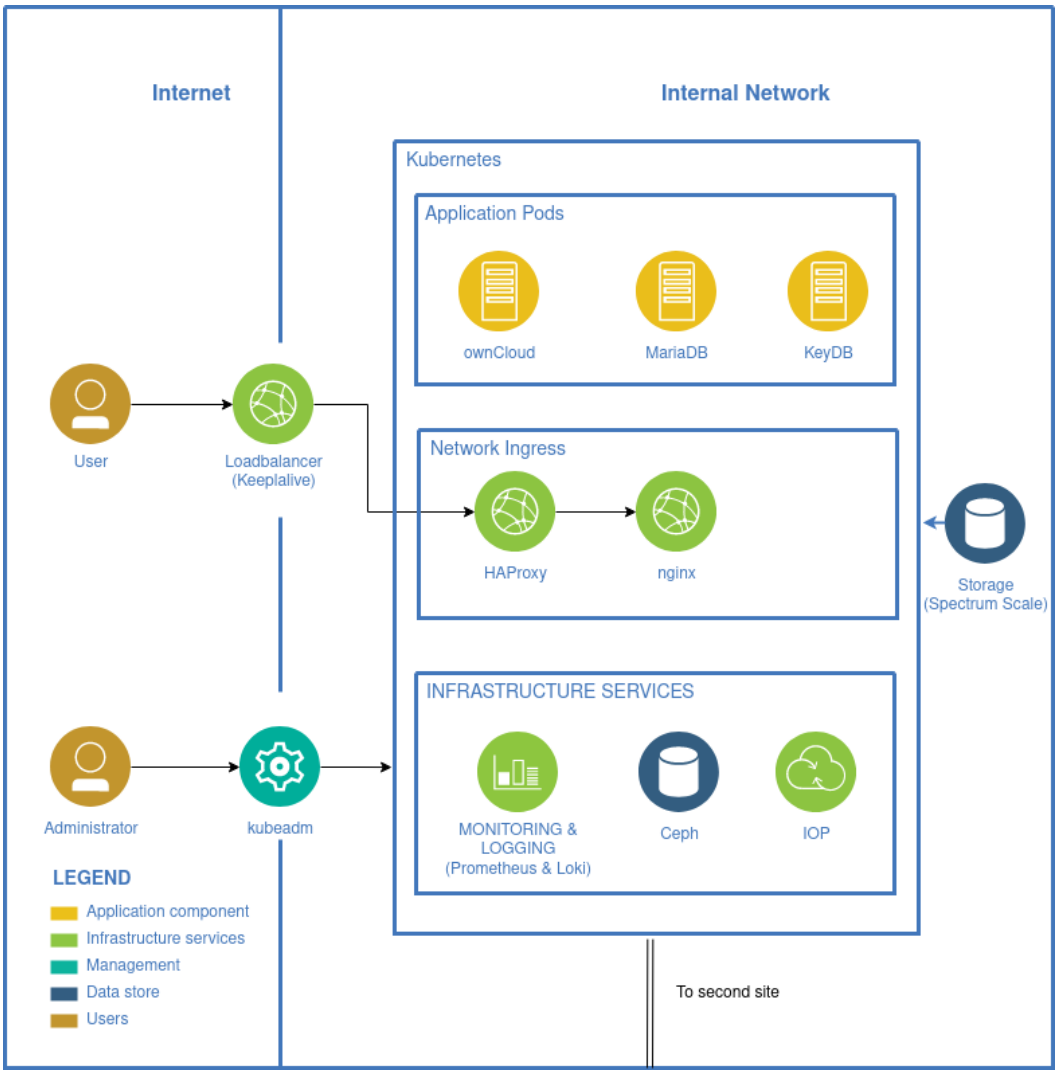

Figure 3: Logical view of the sciebo plattform. User input is directed through keepalive, HAProxy and nginx. Each instance (e.g. university) has its own application pod and can access the central Spectrum Scale storage. The IOP component (interoperability platform) serves as a connector for collaboration functions of the CS3MESH4EOSC project.

A general problem we often encountered were dockerisms in many container images, such as expecting to be root or having write access to arbitrary locations in the file system of the container. This definitely improved with the increased adoption of Kubernetes, superior container runtime engines and more focus on security in the community, but still happens every now and then and is worth mentioning. As a consequence, there are several container images which we created on our own. In general, this process is rather easy; instead of typing the corresponding commands into the terminal, one puts them into a dockerfile. Instead of putting configs in there directly, one later uses config maps. This makes it easier to follow the path of certain configs, too, which is sometimes difficult in very docker centric images, where config parameters are passed through several layers of entrypoint scripts, environment variables and such.

Since there are several customizations to our ownCloud installation, we also build these containers by hand and ultimately use helm charts to deploy them. This still is an area of ongoing development for us, as ownCloud itself still shows some properties that were manageable in more traditional setups, such as writing in the configuration files or writing out logs to a file and not to stdout. The former can be solved with a combination of initContainers and emptyDir volumes, the latter by accompanying ownCloud with a fluentbit sidecar container, which ingests the log file from an emptyDir volume, enriches it with metadata information and then passes it along to stdout, ready to be picked up by the log parser Loki. The user data itself is stored in the IBM Spectrum Scale filesystem, which is mounted as a hostPath volume.

A large problem with multiple services and applications in sciebo's first generation setup was the management of long running cronjobs on multiple machines. This often only worked via a large number of hacks, file locks and dedicating a whole machine to cronjobs only. With Kubernetes, all these things just come for free and it is immediately visible whether or not cronjobs worked as expected.

The most critical part of any ownCloud deployment is a MariaDB Galera cluster. Here we rely on the openstack-helm-infra helmcharts, which so far needed minimal tuning only. These use ceph rbd block storage, which, as it is provisioned from fast local NVMe drives connected by 100 Gbit/s network, are obscenely fast. These helm charts implement the writer-reader logic with NGinx ingress servers, and consequently, we had to restrict the worker processes of these to a small number again.

Currently, we can spin up a new ownCloud instance, including an associated database server, ingress routing, TLS certificates and a session cache with less than ten commands. The next bottleneck to extinguish is developing an automated build and deployment process of our containers. Here, the main challenge we are currently working on, are ownCloud updates, as these require changes to the database when new software or application versions are being deployed.

5 Conclusion

At the moment, we are quite happy with the setup and despite not yet being finished with the migration, we can already benefit from some of the comfort features. The ability to automate processes is increasingly empowering and takes away a lot of the intimidation that comes from large diagrams of interacting microservices. This will become very relevant once oCIS is moving into production readiness, as well as the exciting possibilities with connectors for several powerful applications under the umbrella of the CS3MESH4EOSC project.

We find that Kubernetes as a thin layer between bare metal (or virtual servers) and application is worth the effort even for rather simple setups, especially as the community is moving more and more towards separate services. In a sense, the move from monoliths to microservices is a move back to the

old UNIX philosophy of "do one thing and do it well", but this simplicity is only represented on the level of Kubernetes and not below on the base system.
With the upcoming challenges of edge computing, especially bringing data closer to users, we expect a mix of both containerization and IaaS solution to be necessary in order to utilize state wide clouds such as developed in the NRW RDI project.

References

Vogl, R., Angenent, H., Rudolph, D., Thoring, A., Schild, C., Stieglitz, S., & Meske, C. (2015). sciebo — theCampuscloud for NRW. In: *21st EUNIS Congress, Dundee, Schottland*, p. 15-26.

Rudolph, D., Vogl, R., Angenent, A., Thoring, A., Wilmer, A., & Schild, S. (2016). Challenging Dropbox: The Adoption and Usage of New Cloud Storage Service "sciebo" at German Universities. In: *EUNIS 2016: Crossroads where the past meets the future. EUNIS 22nd Annual Congress, Book of Proceedings*. Thessaloniki, p. 19-29.

Vogl, R., Angenent H., Rudolph, D., Stieglitz S., & Meske, C. (2016). Predictions on Service Adoption and Utilization Meet Reality. First Results from the sciebo (science box) Project. In: Zaphiris, P., Iannou, A. (Eds.): *Learning and Collaboration Technologies. Third International Conference, LCT 2016, Held as Part of HCI International 2016, Toronto, ON, Canada, July 17-22, 2016, Proceedings.* Springer International Publishing, p. 639-649. doi: 10.1007/978-3-319-39483-1_58.

Spoelsky, Joel, https://www.joelonsoftware.com/2010/03/17/distributed-version-control-is-here-to-stay-baby/

Voronkov, A. (2014). Keynote talk: EasyChair. In *Proceeedings of the 29th ACM/IEEE International Conference on Automated Software Engineering* (pp. 3-4). ACM.

Wikipedia. (n.d.). *EasyChair.* Retrieved from Wikipedia: https://en.wikipedia.org/wiki/EasyChair