PROCEEDINGS OF SCIENCE

Building a scalable, interactive and event-driven computing platform in multi-cloud environments with dCache

Michael Schuh*
DESY
E-mail: michael.schuh@desy.de

Patrick Fuhrmann†
DESY
E-mail: patrick.fuhrmann@desy.de

Paul Millar
DESY, dCache
E-mail: paul.millar@desy.de

Tigran Mkrtchyan
DESY, dCache
E-mail: tigran.mkrtchyan@desy.de

This paper about the dCache storage platform and use cases for one of its latest features - storage events is based on three talks given at the ISGC 2019, the International Symposium on Grids and Clouds in Taipei, Taiwan from April 1-5 2019. Building on a deployment in a multi-cloud environment, where dCache acts as a write-through and read-through cache with one single, distributed namespace, we introduce the reader to dCache storage events in both tracks: message queuing and server sent events. Turning to broader use case scenarios in cloud native applications, we address the role of dCache in the EOSCpilot project Photon and Neutron Science Demonstrator, a deployment of Function-as-a-Service platforms for both, interactive and event-driven scientific computing.

International Symposium on Grids & Clouds 2019, ISGC2019
31st March - 5th April, 2019
Academia Sinica, Taipei, Taiwan

*Speaker.
†Speaker.

© Copyright owned by the author(s) under the terms of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License (CC BY-NC-ND 4.0).

PoS(ISGC2019)007

1. Introduction

This paper about the dCache storage platform in event-driven and interactive use-cases in multi cloud environments is organized in three sections, which reflect the topics of three talks given by DESY IT scientists at the ISGC 2019, the International Symposium on Grids and Clouds in Taipei, Taiwan from April 1-5 2019. Firstly, we present the role of dCache as a write-through and read-through cache with one single, distributed namespace in federated cloud environments. In the second section, we introduce the reader to dCache storage events in both tracks: message queuing and server sent events. We address how dynamic processing of storage events provides users with additional capabilities for monitoring and automation. Turning to broader use case scenarios in cloud native applications in the third section, we present findings from the EOSCpilot project Photon and Neutron Science Demonstrator about interactive and event-driven applications of functions as-a-service in scientific computing. We conclude that dCache storage events enable users to execute codes fully automated in response to incoming data, e.g. data written by sensors or an experiment, or data staged from tape. Finally, we conclude with some comments on the current state of interoperability of the demonstrated scientific software with cloud native environments and discuss potential future work.

2. dCache in a multi-cloud environment

For over a decade commercial providers offer their computer resources as public clouds. With a couple of clicks one can build and run a full data center without any local infrastructure. While this is a very attractive approach, high costs and/or legal aspects force sites to run their own infrastructure, often as a private cloud, though. However, such local resources may be insufficient if user demand cannot be foreseen in advance.

One option to cope with such peak loads, known as cloud bursting [1], is to build a hybrid cloud by combining the local instance with external resources or public clouds. Although this model works, it has downsides. Despite the ease with which additional CPU resources can be integrated into local infrastructure, the data is typically only available from the primary site. Due to network latency, jobs running on external CPU resources may be less efficient than jobs with local data access. Moreover, remote data access may produce greater network usage, especially when the same data is requested from multiple jobs. This inefficient network access can increase overall operational costs or result in network bandwidth starvation.

For years, dCache.ORG has provided robust software, called dCache, that is used at more than 80 universities and research institutes around the world, allowing these sites to provide reliable storage services for the WLCG experiments and many other scientific communities. Thanks to its microservice-like architecture, dCache scales horizontally with increasing number of nodes and can provide the desired storage space with the required data throughput. As all dCache components communicate with each other by sending messages over the TCP connection, a single instance of dCache can be as small as a single computer, including a Raspberry-Pi, and as big as thousands of hosts providing tenths of petabytes of data spread over multiple geographical locations [2].

One of the key aspects of dCache's design is the separation of files' data and metadata such as name, size and checksum. By using a unique identifier for each file that is independent from the

PoS(ISGC2019)007

file's name and location, dCache splits a file's metadata and data. This unique identifier is used by dCache internally when a file is addressed. By using this additional level of indirection, new data servers can be added and removed dynamically within existing deployments. To reduce operation overhead, dCache can dynamically make use of newly available data servers and automatically redirect clients coming from a remote site to a locally available resource. This is achieved by each data server connected to the system using an internal discovery mechanism to locate the Pool Manager, a component that is responsible for client request distribution within the system. Data servers that present the same tags are grouped together dynamically into pool groups. Based on the client's IP address, the file's path and data flow direction (read or write), a desired pool group is selected. When data is not available within the given pool group, it is replicated from elsewhere. This flexible data placement mechanism ensures that when a file is requested from a remote location it will be replicated to the (network-topology wise) nearest storage node. Once data is available locally, all access will be served from the local copy. Moreover, data produced at the remote site can be written into local store for further local use and later pushed to the main site. Such a setup provides a transparent read-through and write-back cache behaviour for the end-user application with a minimal operation overhead.

In addition to on-demand replication, dCache supports yet another two data replication modes: a manual replication, where data managers can pre-place data at specified locations, and an automatic replication, which is typically used for data resilience. The latter can be configured to guarantee data availability at different geographic locations.

The network-topology-aware data replication on demand is well suited for the cloud bursting model. When CPU resources are allocated at the external resource provider, an additional dCache data server is allocated. The dynamic pool group configuration will make use of this storage and start to replicate requested data to the remote location. However, in case such an additional data server cannot be added to the system, or this storage becomes unavailable, client applications will continue to work, albeit with reduced efficiency.

For muti-node deployments it is recommended to run dCache in High Availability (HA) mode. This is a deployment where a single node crash does not affect the overall system availability. However, as a distributed system we cannot guarantee network reliability all the time. As Brewer's CAP Theorem [3] says: in case of network partitioning a distributed system cannot guarantee consistency and availability at the same time. dCache is not an exception in this sense. Thus, in case of network partitioning, dCache chooses consistency over availability. Although such behavior is expected for typical deployments, in a hybrid cloud setup, where some nodes play a role of data caches, we may prefer to provide access to the locally available cached data even if the main storage system is not accessible. This is a subject of forthcoming development activity within the dCache project.

3. dCache storage events

In this section we discuss storage events: a powerful, new way for clients to interact with storage.

PoS(ISGC2019)007

3.1 Introduction to storage events

Traditionally, clients interact with a storage service by issuing direct commands; for example, by opening a file or requesting a file is deleted. The service's response indicates whether this is successful and, depending on the command, some additional information; for example, a request to open a file returns a handle that represents the file's open state if the user is allowed to read that file, while a delete request returns whether the delete operation was successful.

This request-response pattern is also repeated for potentially long-running activity (e.g., staging media transitions or calculating a checksum), where a client requests the service starts an activity and is provided with some token that represents this activity. The client may later use this token when enquiring whether this activity has completed. To learn if the activity has completed, the client must request the current status of the activity repeatedly: polling.

Although this request-response pattern is almost universally adopted, there are several limitations with it.

First, only the client making the request knows that a particular command was issued. In many scenarios, other services should be updated when data is uploaded, deleted, or when there are other changes to the storage system. A common work-around for this problem is to use a special client that informs the other services when it makes a change. This introduces new problems, as it requires all changes to be made with this special client. As the underlying problem is not solved, this work-around is fragile: changes that do not come through the client (e.g., admin intervention) may still cause inconsistencies.

Second, polling to discover when a long-running activity has completed does not scale. The number of queries the client must make increases with the number of long-running activities. Each query uses resources to obtain the current status, resulting in some upper limit on the number of concurrent activities a service may sustain, simply to answer all the client current-status requests.

Storage events is a new way of interacting with storage. In contrast to the traditional request-response pattern, the storage system generates events when something happens, whether this is from client interaction (e.g., a file has been uploaded, renamed, or deleted), from internal subsystems (e.g., a file has been staged back from tape) or from admin intervention (e.g., storage nodes being taken offline). A client subscribes to some subset of events, those triggered by transitions the client is interested in discovering. These events are delivered to the client asynchronously, without the client polling. As storage events do not require polling, the client learns of changes very quickly. It also scales well, since the subscriptions typically require very few resources.

Broadly speaking, there are two approaches to deliver events: direct delivery and brokered. In direct delivery, the events client connects directly with the storage system and subscribes to events there. In a brokered delivery, the events client connects to some broker external to the storage system and subscribes to topics. A topic is a group of related events. By subscribing to topics, a client may have courser-grain control compared to direct delivery

Data catalogues, where a service maintains a list of data stored on dCache, is an example where storage events are useful. By subscribing to storage events, catalogues learn of all changes in the storage, while simultaneously allowing users to use any supported client for data transfer and management.

Staging files back from tape is another example. To be as efficient as possible, tape systems

PoS(ISGC2019)007

should avoid unloading a tape from a tape drive if that tape contains additional data to be staged. To increase the likelihood that all desired data is read from a tape in one go, the storage system should be presented with as many as possible of the files to be staged. Polling limits the number of files present in a single request. This in turn limits the size of the list of files to be staged.

A similar approach is available in commercial cloud storage systems; for example, Amazon provides Lambda and Google's cloud platform provides Cloud Functions. Both systems allow users to define computational work that is tied to events, with a canonical example being thumbnail image generation when a photo is uploaded.

Event driven work-flows are also well supported by open-source software. Apache Storm and Apache Spark provide analytical frameworks for events, while projects like OpenWhisk and Kubeless provide more general computational frameworks. Apache Kafka and Apache Nifi provide the infrastructure for routing events from the source to the destination.

3.2 Storage events in dCache

dCache provides two ways for clients to receive events: either using Apache Kafka or via W3C's Server-Sent Events (SSE) protocol. The two event delivery mechanisms have different trade-offs with different intended audiences: they complement each other rather than providing alternatives. Currently, these two event delivery mechanisms each provide a non-overlapping set of possible events: some types of event are only available via Apache Kafka while other event types are only available via SSE.

Apache Kafka is a de facto industry standard software package for building pipelines that distribute events from services generating events to those services consuming them. The main benefit of Kafka is that it is widely adopted, with good integration with many open-source projects. dCache provides the possibility of sending certain dCache-internal events over Kafka. This is currently the "billing" events: which record data transfers (a file being uploaded or downloaded, internal replication) and deletions (removal in the namespace or a file's data being removed from storage media). These events currently lack any security model: a client that can see one billing event can see all billing events.

W3C SSE protocol is a standard mechanism to deliver events to clients based on the HTTP protocol. The client must authenticate to receive events and to manage subscriptions, and can authenticate with any of the many authentication schemes that dCache supports. Currently SSE supports an inotify-like protocol. Inotify is an API provided by the Linux kernel that allows clients to discover when changes are made to the filesystem. The dCache implementation provides an implementation of this interface, using SSE to deliver inotify events. As with the Linux inotify implementation, dCache's inotify has an authorisation model where clients can only see events if they are authorised to do so. The main benefits of SSE/inotify are the built-in security and a more complete set of events.

3.3 Use-cases and demonstrators

One of the software developments during the EU-funded INDIGO-DataCloud project is the INDIGO Orchestrator. This allows clients to specify a desired analysis platform using the TOSCA standard language. The Orchestator has a brokering feature where it selects the best available Platform-as-a-Service (PaaS) site and orchestrates the desired services at that site.

PoS(ISGC2019)007

Within the follow-on eXtreme-DataCloud (XDC) project, the INDIGO Orchestrator is extended to support dynamic data processing. To achieve this, the Orchestator uses dCache's inotify support, as delivered by the SSE interface, to learn when new data is available. It then contacts a data management tool (Rucio) to ensure that new data is available at storage co-located with the PaaS infrastructure and processes that data automatically as it arrives.

Once the new data has been processed, Rucio takes care that it is moved off to external storage, leaving sufficient capacity for subsequent new data.

3.4 Storage events demonstration

During the talk, a live demonstration was given that illustrated some of the benefits of storage events. In this demo, a video was taken of the audience with the speaker's smart phone. This video contained deliberately introduced camera shake. The video was uploaded from the phone into dCache, triggering a storage event delivered to Kafka describing the new file.

A dynamic agent reacted to this storage event by checking the file matched configurable criteria and generating pre-authorised URLs that allowed a computing infrastructure to download the new file and upload multiple derived data. These pre-authorised URLs were included in a JSON object, sent to Kafka on a separate task topic.

A scalable computing infrastructure was deployed as a Function-as-a-Service (FaaS) resource. The OpenWhisk platform was chosen as the implementation, which provides good integration with Apache Kafka. For this demo, the OpenWhisk cluster was configured to listen to the task topic. When it received the JSON object from the dynamic agent, it dispatched the task to a container that was configured to download the video, use ffmpeg's vid.stab plugin to apply a two-pass "de-shake" filter, creating a version of the video with reduced camera shake. A subsequent ffmpeg operation generated a split-screen video, combining the left-half of the original video with the right-half of the deshaked video. These two artefacts (the full deshaked video and the split-screen video) were uploaded to dCache using the pre-authorised URLs.

This demonstration provides a trivial example of a more general model, where advanced computation may be triggered by uploading data into dCache. This powerful model is explored further in the next section, along with interactive applications of FaaS computing.

This ability to trigger computational work when data is uploaded is, in turn, only one example of more general integration possibilities, where external systems react to changes within dCache. Other uses include service synchronisation, where some external service provides an enhanced (domain-specific) view of stored data. Such services must be made aware of any changes to that data (available new data, data deleted, etc). Storage events provide a powerful way to integrate such services.

4. Function-as-a-Service as use-case for dCache storage events

The Photon and Neutron science community (PaN) is pushing frontiers with ground breaking research and technologies in molecular imaging at the atomic level. State of the art Photon and Neutron sources, like the European XFEL [4] and the European Spallation Source (ESS) [5] will create hundreds of Petabytes of data per year, challenging established data processing strategies. Leveraging cloud computing methodologies to provide innovative flexible and scalable storage and

PoS(ISGC2019)007

compute services, the project Photon and Neutron Science Demonstrator (Pan SD), in the European Open Science Cloud for Research Pilot (EOSCpilot) [6] [1], covers the entire data life cycle from experiment control to long term archival. A particular focus was given to FAIR [7] access to large volumes of scientific data and the general re-usability of research artefacts. The reproducibility of methods and results is achieved with an integrated approach that bundles publications, data, workflows and functions implemented in containers. Building on cloud native solution building blocks deployed on commodity hardware and on dCache as storage backend, a service oriented architecture is presented that enables scientists to develop micro-services and make them available for interactive and event-driven scientific analysis workflows. Services discussed in this section, have been made publicly accessible through the EOSC portal [8] for demonstration and testing.

During this project, DESY has advanced with a CEPH-backed OpenStack on-premise cloud infrastructure, from PoC to the integration with the EGI FedCloud. Key design principles are a fully functional auto-scaling infrastructure, taking advantage of the programmability of software defined resources, self-healing features and service discovery. These objectives are widely achieved, leveraging Platform-as-a-Service (PaaS) strategies, deploying Kubernetes and Docker with OpenStack as Infrastructure-as-a-Service (IaaS) layer.

In the following chapter, we will briefly introduce the solution building blocks, comprising input from the GitLab [10], Jupyter [9] and OpenWhisk [11] projects as shown in figure 1, and then present two use case models for interactive usage of containerized functions as-a-service in Jupyter Notebooks. We will then show how dCache storage events seamlessly enable event-driven automation with the same codes on the same architecture. We conclude with lessons learned from deploying codes from photon and neutron science to the described platform.

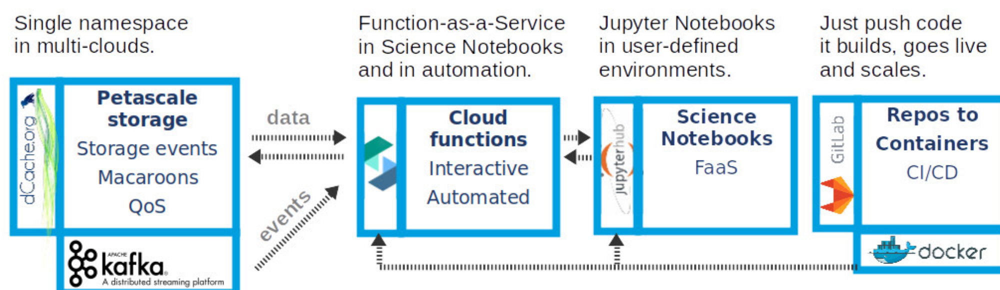

Figure 1: Solution architecture and data flow

4.1 Solution building blocks

At the center of the user-perspective, we make the Jupyter Notebook available as a service for users in the European Open Science Cloud, accessible through Jupyter Hub, deployed via Zero-to-Jupyter-Hub (z2jh) Helm charts [12]. We integrate with federated AAI using EGI Check-in [13] as OpenID Connect (OIDC) proxy, which is interoperable with the Shibboleth Identity Provider

[1]The European Open Science Cloud for Research pilot was funded by the European Commission, DG Research & Innovation under contract no. 739563

PoS(ISGC2019)007

(SAML) at DESY. Offering 10 GB persistent storage per single user server, we integrate also OpenStack Cinder and create volumes for new users automatically. We allow users to set up the environments in which they run their Jupyter Server by building Docker Containers from GitLab repositories, through pipelines executed by GitLab's CI/CD Runners (Continuous Integration / Continuous Delivery).

While we still manually scale the Kubernetes Cluster for the Jupyter Hub itself, we have achieved full auto-scaling for CI/CD jobs that build and deploy these user environments. For this, we deployed GitLab based on OpenStack HEAT templates, offering auto-scaling GitLab Runners configured for Docker+Machine executor and OpenStack Driver. Users find a template project with re-usable, yet customizable, CI/CD configuration (.gitlab-ci.yml) based on Docker-in-Docker (dind) approach. A local pull-through Docker registry mirrors the Docker Hub and a S3 Cloud Storage enhance the re-usability of artefacts and layers between CI/CD stages and save exectution time. The cache is implemented based on MinIO, but is foreseen to be replaced by dCache itself in future releases. Submitting user environments as code into version control supports re-producability, which is completed by checksums for CI/CD pipelines and container builds, allowing to track changes in environments and to roll back to a certain version and configuration by refering to that specific revision. Users can share built containers on Docker Hub or other external registries. As means to revisit codes that are continuously deployed on the cloud platform, they also push to the projects own registry in the same namespace as their GitLab project, which allows the deployment of GitLab's access token management in sync with the projects user and group settings.

Approaching the main topic of this section and one of the key features of the EOSCpilot PaN platform, we describe how the same CI/CD strategy is applied to publish functions running on user-defined software stacks as micro-services. We have evaluated the integration of the projects Kubernetes Fission, OpenFaas and OpenWhisk, which all have in common, that users can run containers as functions as-a-service. In this chapter, we will discuss the current version, in which we chose to integrate Apache OpenWhisk. The project is designed to deploy functions directly from code (down to one single file) in runtime environments which are provided for a rich set of programming languages. The functions can then be called via a simple HTTPS service, either directly per URL (curl, HTTP clients) or with the project-provided client. Atomic function calls can be done synchronously, asynchronously and also sequenced conditionally.

Users at photon and neutron facilities typically work with highly specialized scientific environments, hence we need to include libraries, frameworks etc. to the runtime containers, a concept named blackbox images in OpenWhisk. The most simple implementation layout for a cloud function using a blackbox image is a GitLab project that comprises a Dockerfile defining the environment, and a single file that provides the function to run on top of it. This is complemented by the re-usable, customizable CI/CD definition file, which allows continuous integration of the function by building the Docker Container, deploying it to a test namespace on OpenWhisk, performing functional tests, and then releasing a Docker Container in the project's registry as well as deploying it to the given production name space. OpenWhisk admins can distribute the needed authentication keys via CI/CD variables in GitLab, with fine grained control over their visibility based on projects, users and groups. This release strategy also allows to pull and run the functions from federated resources, e.g. in the EGI FedCloud or the European Open Science Cloud.

We documented our recommendations to use git from the single user servers and showed, how

PoS(ISGC2019)007

nbgitpuller links [14] can be used to grant access to notebooks, which are published in git, and which will be automatically synced to the user's Jupyter Server, for users following those links. A related technology, which we are planning to implement is the Jupyter BinderHub [15], which provides another way to define the Jupyter Server environment, in which the published notebooks should run. At the time of writing the missing user authentication in current releases remains a blocking issue, but we expect this feature to be available soon. Almost the same functionality can be achieved, by allowing users to choose from customized server environments, which are shown at login time. This requires that users can identify interoperable environments for a given notebook, but we are working with the Jupyter Project on a solution that also eradicates this last manual step in the next release.

4.2 Interactive use of functions as a service

The focus on collaboration in the European Open Science Cloud and this science demonstrator project which brought together service providers and scientists from photon and neutron science, stressed the demand to enable collaborative editing of science notebooks as well as sharing of functions to use in different workflows. Like the Jupyter Project, the scientific platform presented in this section allows sites to build on a broad spectrum of programming languages, hence distribution of functions with means of a given language (e.g. pip for python) can only serve a matching subset of notebooks/workflows. In a live demonstration during our talk at the ISGC, we showed that provisioning of codes as services is an interesting approach to solve this problem.

We imported a compact python class providing convenience wrappers for a subset of the OpenWhisk and dCache APIs. However, advanced users still have the choice to use either the APIs directly in their own code or use the OpenWhisk Go client binary, which is installed in the Jupyter Server Single User Environments. This is independent of the kernel on which the notebook runs. Another elegant way to call functions from notebooks are Jupyter Widgets, which provide many features for interactive user workflows, where parameters for function calls can be selected from drop down lists or configured and validated with the help of the various input widgets such as the slider widget. Remote function execution can also be initated by mouse click, in the easiest layout just by pressing a button. More advanced page impressions are also possible, such as selecting features in a plot or an embedded image.

The first example we presented was a very simple micro-service to work on reference data sets from the Linac Coherent Light Source (LCLS) [16] photon science facility that were published as set cxidb-21 by the Coherent X-ray Imaging Data Bank (CXIDB) [17, 18], a website dedicated to the goal of archiving data from Coherent X-ray Imaging (CXI) experiments and making them available to improve the reproducibility of results and to enable new research based on previous experiments. For this science demonstrator, we copied the data to the dCache storage backend hosted in the DESY data-center. The data is publicly accessible, however, the following examples for micro services work with the dCache API and WebDAV doors, for which they can make use of Macaroons, to access data protected by ACLs. This safe way of delegating access rights allows stateless container based functions to read and write data and to perform other operations to manage the respective namespaces.

In many realistic use cases for science notebooks in large scale data driven research, some method of compiling lists of files will be performed, which serve as input for further processing.

PoS(ISGC2019)007

Often, this contains nested loops and selection cuts. Repeatedly including related code blocks in science notebooks does not leverage full re-usability, but instead adds potential for undetected errors and distracts the reader from the main workflow. They can be extracted and easily implemented as a simple service, which allows to share the method, to reduce the amount of codes in user facing notebooks and to focus on other steps of the respective workflow. On a button press, the information is retreived from a FaaS execution as serialized JSON, which is directly deserialized to a Python object in the interactive user session. In this example, the result was a dictionary holding a list of links to the selected data and an integer number giving the length of that list. This dictionary may be used as input for microservices, that process the list of files e.g. to start a batch job or to call a function for each link and finally perform a map reduce operation on the respective outputs.

The second example illustrates a use case where data needs to be transferred and processed, in this case to visualize the content of a HDF5 file, a detector readout in serial x-ray crstallography. The service is implemented by running the program hdfsee, which is a part of the CrystFEL framework [19]. For particular implementations, users can write their own programs compiled against the C library libcrstfel. However, the framework provides a lot of functionality and its tools can operate on data from a broad spectrum of experiments, detectors and related file creation modules. This FaaS deployment works without downloading data to the users' single notebook server, where it would still fit in this case, but will no longer fit well for growing file sizes, which we observe for data sets obtained with latest instruments e.g. at the European XFEL. Note, that in a multi-cloud environment, the Jupyter Server may run in a different data center, than the storage element and transfering the data set via Wide Area Network should be avoided. The approach we discuss here deploys the function to process the data on compute elements in the same data center as the storage elements and data is streamed in the local high bandwidth network. The result (in this case an image only approx. 2 MB in size) is then written back to dCache, from where it can be shared, accessed or embedded in the notebook or other webviews. The HTTP response for the function execution contains the links to the files created. This response, as well as the logs created during the function run, will also be stored in the activation database managed by the FaaS service.

This strategy is particular suited for short running functions performing data reduction, validation or visualization. The maximum function runtime defaults to 5 minutes in the OpenWhisk standard configuration, the maximum size of data directly returned with the HTTP response defaults to 1 MB. It is also possible to integrate other architectures, e.g. trigger batch processing by submitting into nearby HTC clusters, thereby providing dedicated batch processing pipelines as a service. Newest developments of those platforms, for instance as presented in the HTCondor workshop at the ISCG also allow interactive processes, which would allow to connect from the Jupyter Notebook, e.g. to display intermediate results. A similar behaviour in containerized micro-services can be reached by implementing an API inside the container and exposing the endpoint during run time.

4.3 Running functions in response to storage events

In the demonstrated setup, the dynamic processing tool [20] works with the Kafka Stream API to map Kakfa messages published by the dCache storage backend into subqueues. It filters the messages by information like the upload path, uploading user or source IP and matches file names to a regular expression. Messages are re-published on dedicated topics on the same or a

PoS(ISGC2019)007

different Kafka broker, where they can be secured with respective authentication, authorization and certificates to support multi tenancy in cloud environments.

All FaaS platforms evaluated and operated for this project provide connectors to Kafka brokers, that consume message queues accessible by server endpoint and topic. Incoming messages in the FaaS cluster will fire a trigger, which can be mapped by rules to one or more functions or sequences of functions. Each function again, can be mapped by more than one rule. The trigger invocations, as well as the function invocations are tracked in the activation database. It is possible to replay whole queues, e.g. to repeat data processing steps with updated codes or configurations.

An important design goal of the presented microservices is to enable the use of exactly the same function both, interactively and fully automated. The data visualization example can be triggered via a Kafka message, and can automate the creation of previews or thumbnails which can be used to display a dynamic web page representing a data set. The simple and generic layout for such a data processing micro service expects two links as function arguments, one download link to read data for processing, and one upload link to the output directory to upload results. They can be provided by the user in an interactive session as well as by the Dynamic Processing tool in automation pipelines. In both use cases, embedded macaroons delegate read and write access in a secure way, allowing to restrict their use to the IP range of the FaaS cluster and add an expiration date.

In the eXtreme-DataCloud (XDC) project, DESY demonstrates that event-driven code execution as a service adds a flexible building block to smart data placement strategies, enforcing machine actionable Data Management Plans. For this, the FaaS system interacts with rule-based data management engines and file transfer systems, e.g. to create replicas of data sets with respect to data locality and Quality of Service for storage. On data ingestion, files can be copied to cloud storage elements, which act as buffers next to strong clusters of compute elements that perform Function-as-a-Service pipelines and update data placement rules on success. This automatically deregisters files in the buffer next to the compute clusters, and triggers their distribution to offline storage and long term archival.

Another appealing use case is re-staging data from tape after archival, which often may include a waiting time in the order of minutes or even hours. The implementation of data processing or batch processing starters as a service allows to trigger following steps in the workflow fully automated as soon as the data is available on disk. Also in the other direction of QoS changes, the archival of data from disk to tape, event-driven code execution is applicable in some use cases, performing data validation to assert that only cross checked data sets are effectively moved to long term archival.

4.4 Scientific frameworks integration in serverless applications

The EOSCpilot PaN SD worked with the model cases CrystFEL (Photon Science) and Mantid (Neutron Science), which both usually process large volume data sets. An important consideration with the photon and neutron community is that a majority of users expects functionality to be delivered through directed workflow management tools and graphical user interfaces (GUIs). The implementation of the discussed hdfsee micro service also serves to illustrate an important limitation for the integration of scientific frameworks in cloud native applications. The original implementation is done in C and GTK3, where the framework supplied solution is bound directly to

PoS(ISGC2019)007

the GTK classes and no strict separation into backend (reading into a pixel buffer, producing a PNG file) and frontend (display image in GUI, provide menu entry to export image) is implemented. To make this original code work in a stateless container, whithout display and user interactions, we run the tools with a virtual pixel buffer (xvfb) and a virtual user input to simulate keystrokes (xdotool).

Factoring in benefits of rapid development of serverless deployments, framework authors would ideally take the effort of separate core from GUI codes to make them accessible in headless compute environments. For cloud native applications, a co-development approach adds strong competences on the provider side, with respect to enhanced scalability and facilitated configuration management as well as integration with event-driven use cases.

We also see a strong trend towards the reimplementation of visualization codes directly in Widgets in Jupyter Notebooks, to make them available as genuine embedded elements which adds value and anhances user experience in many cases. However, forking the code for Jupyter Widgets should be carefully justified, weighting the maintenance effort for future developments and evaluating the stronger interoperabiltiy of the FaaS approach.

In many cases, where users re-invoke previously completed data analysis procedures, the function-wise organization scheme, together with the strong revision based implementation, that versions data, software and containers build from it, promises better caching approaches than concurrent HTC activation strategies. The exact environment, arguments passed to the functions, input read and output created can be accessed from the activation database and backend storage later-on.

Further ongoing research includes the routing of event-driven containerized processes to FaaS based clusters, taking into account data locality. Another objective is the integration with existing computing pipelines HPC and HTC clusters, running jobs with consistent data profiles as a service, using FaaS to compile submit start scripts. An interesting emerging feature here is the ability to snapshot running containers, and move them between infrastructures on behalf of the user.

References

[1] Rouse, Margaret. *Definition: Cloudbursting* SearchCloudComputing.com (2011).

[2] G. Behrmann, P. Fuhrmann, M. Grønager, and J. Kleist *A distributed storage system with dcache*, *Journal of Physics Conference Series* **vol. 119**, (p. 062014), (2008).

[3] Seth Gilbert and Nancy Lynch, *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services*, *ACM SIGACT News*, **Volume 33 Issue 2**, pg. 5159. doi:10.1145/564585.564601 (2002).

[4] *European XFEL - European X-ray Free Electron Laser* https://www.xfel.eu

[5] *ESS - European Spallation Source* https://europeanspallationsource.se/

[6] *principle findings from the EOSCpilot project* https://www.eoscpilot.eu/sites/default/files/eoscpilot_principal_findings.pdf

[7] *FAIR Guiding Principles for scientific data management and stewardship* http://www.nature.com/articles/sdata201618

[8] *EOSC portal* https://www.eosc-portal.eu

[9] *Project Jupyter* https://jupyter.org

PoS(ISGC2019)007

[10] *GitLab* https://gitlab.com/

[11] *OpenWhisk* https://openwhisk.apache.org/

[12] *Zero to JupyterHub with Kubernetes* https://zero-to-jupyterhub.readthedocs.io/en/latest/

[13] *EGI Check-in* https://wiki.egi.eu/wiki/AAI

[14] *nbgitpuller* https://github.com/jupyterhub/nbgitpuller

[15] *BinderHub* https://binderhub.readthedocs.io/en/latest/

[16] *LCLS - Linac Coherent Light Source* https://lcls.slac.stanford.edu/

[17] *CXIDB - Coherent X-ray Imaging Data Bank),* http://www.cxidb.org/

[18] *CXIDB-21 Data Set* **doi:** 10.11577/1169541, www.cxidb.org/id-21.html

[19] *CrystFEL: a software suite for snapshot serial crystallography* http://www.desy.de/~twhite/crystfel/

[20] *Dynamic Data* https://github.com/paulmillar/dynamic-processing

PoS(ISGC2019)007