Using dCache Storage Events in NextCloud and On-Site Experiments

HEPiX Autumn Conference 2022

Tim Moeller, Tigran Mkrtchyan, Peter van der Reest, Kilian Schwarz, Christian Voß,
Umeå, November 3rd 2022

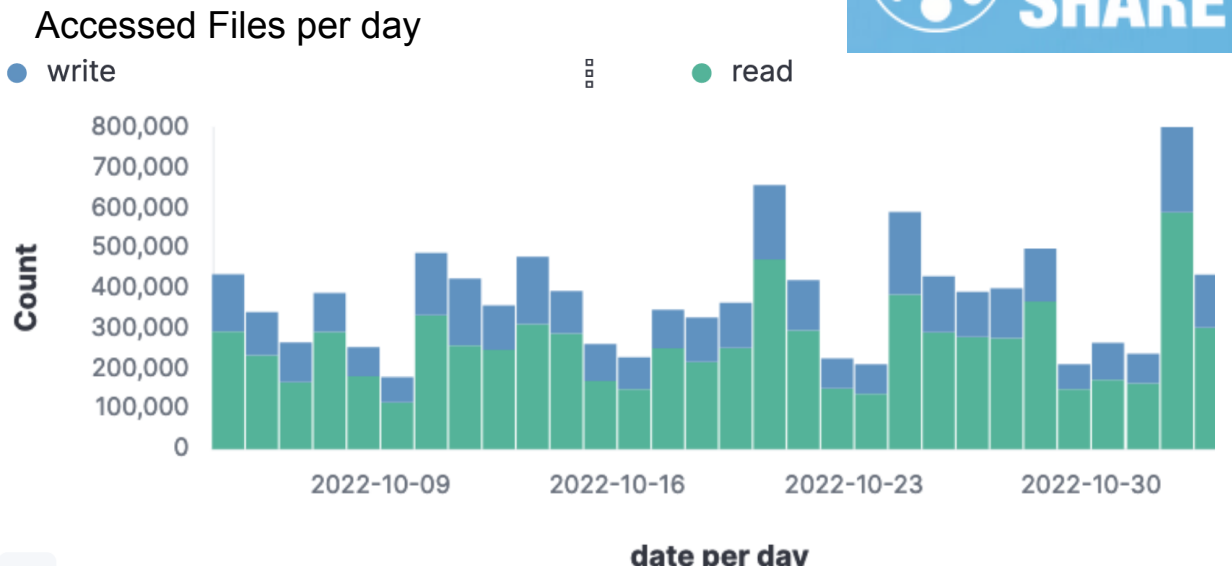
HELMHOLTZ

DESY.

Motivation

Why do We Need Storage Events

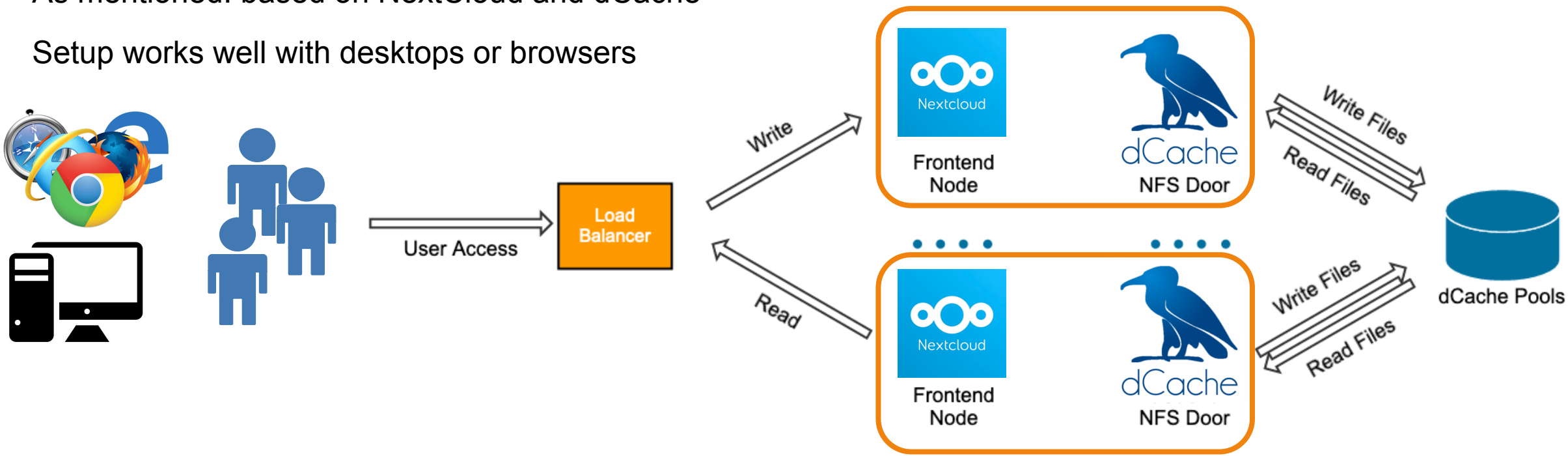
- Sync&Share service based on NextCloud increases in popularity at DESY
- Running regular dCache and NextCloud versions
- See increased pure scientific use-cases
  “Take a look at my training results from today”
- Find a way to integrate into our cluster infrastructure
- Deployment of NextCloud Sync-Client suboptimal on a shell based environment
  → Desktop design (GUI heavy)/CLI limited functionality
  → Tedious configuration in SSH login shell
- Cause unnecessary traffic and space usage (syncing your talks/reports to the cluster)
- Make use of our backend storage: dCache



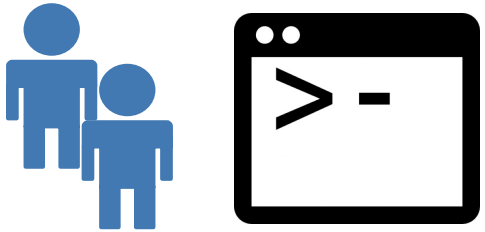
DESY Sync&Share

Service Overview

- As mentioned: based on NextCloud and dCache
- Setup works well with desktops or browsers



- How to fit in access from compute clusters?



How to Access Sync&Share without Using NextCloud

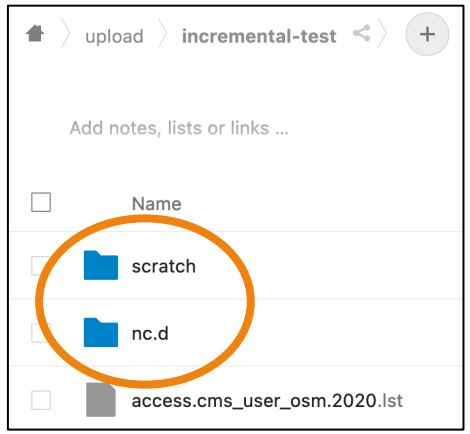
Use dCache directly

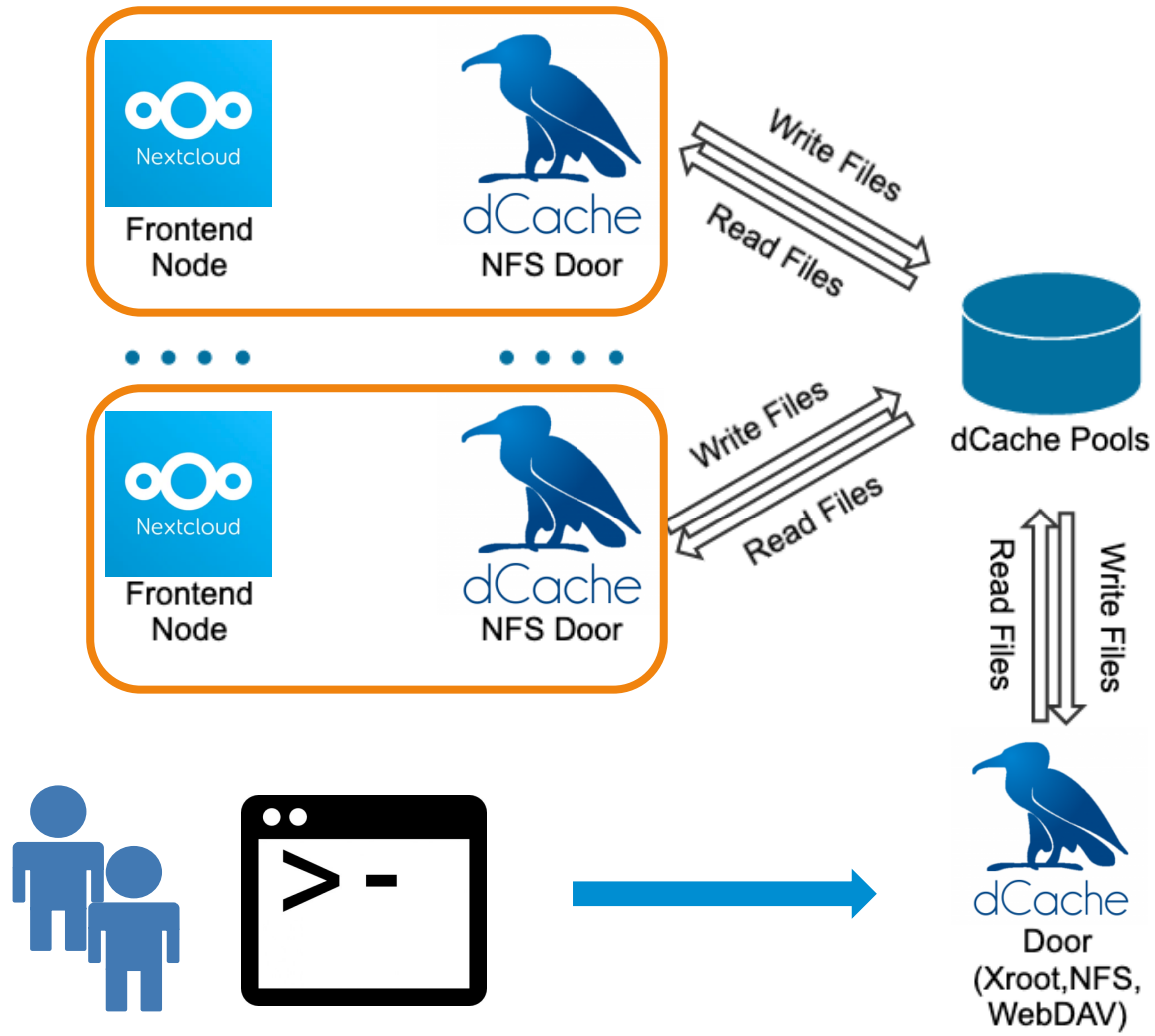
- Instead of using the sync-client → write/read directly from dCache
- Naively: Introduce regular dCache endpoints

Read Access

- Straight forward
- NextCloud namespace reflected 1 to 1 in dCache
- Configures namespace to carry correct `uid:gid`

```
[vossc@naf-it01] vossc $ rclone lsd cloud-dcache:/vossc\/files/upl
oad/incremental-test
          -1 2022-11-01 13:35:20        -1 accesslist_per_month
          -1 2022-11-01 13:35:17        -1 accesslist_per_protocol
          -1 2022-11-01 13:19:57        -1 nc.d
          -1 2022-11-01 13:19:57        -1 scratch
```





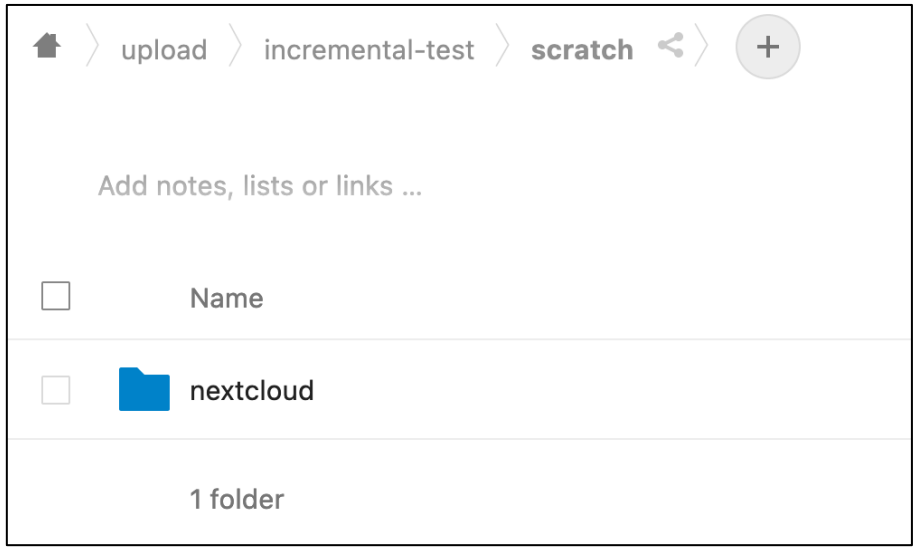
Writing through dCache to Sync&Share

View from Command Line Clients

Write Access

- Should be straight forward as well
- NextCloud namespace reflected 1 to 1 in dCache
- Configures namespace to carry correct `uid:gid`
- Use our client to write a file

```
[vossc@naf-it01] tmp $ rclone copy --max-depth 1 --include "dcache-8.1.6-1.noarch.rpm" . cloud-dcache:/vossc/files/upload/incremental-test/scratch/
[vossc@naf-it01] tmp $ rclone ls --max-depth 1 cloud-dcache:/vossc/files/upload/incremental-test/scratch
129278835 dcache-8.1.6-1.noarch.rpm
```



The View from NextCloud

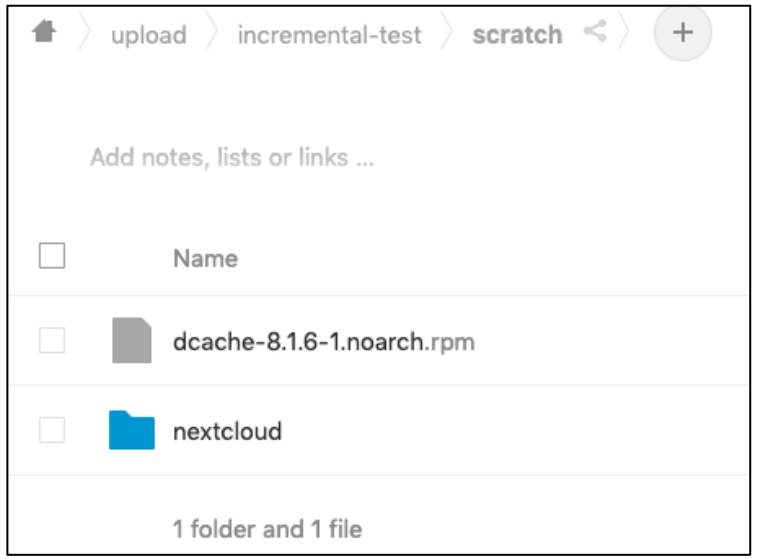
How to Register Files

What Happened?

- We have produced dark data for NextCloud → need to perform a scan

```
[root@it-sas-management ~]# sudo -u apache php /var/www/nextcloud/occ files:scan --path "vossc/files/upload/incremental-test/scratch"
Starting scan for user 1 out of 1 (vossc)
+---------+-------+--------------+
| Folders | Files | Elapsed time |
+---------+-------+--------------+
| 45      | 1     | 00:00:01     |
+---------+-------+--------------+
```

- After the scan the file is visible:



How can we automate this?

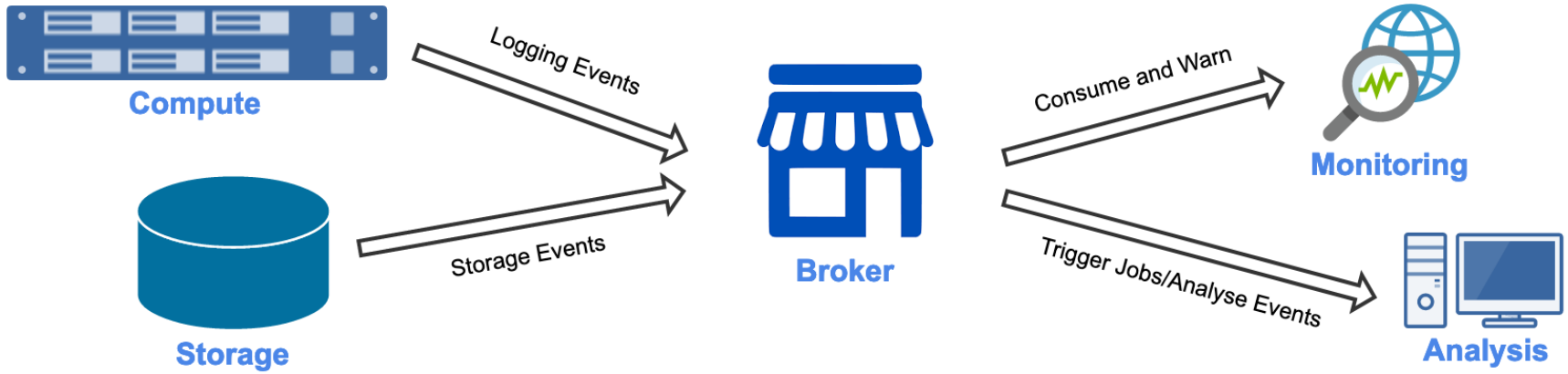
- Cronjob per day/hour → latency too large
- All 3k user directories? → time intensive
- Use dCache storage events

Messages and Events

Arbitrary Definitions for Upcoming Slides

- Event act similar announcements to interested consumers (marketplace)
- Not directed to a specific endpoint (e.g. sending a direct call to a specific service)

- **Message Producer – Broker – Consumer Model**



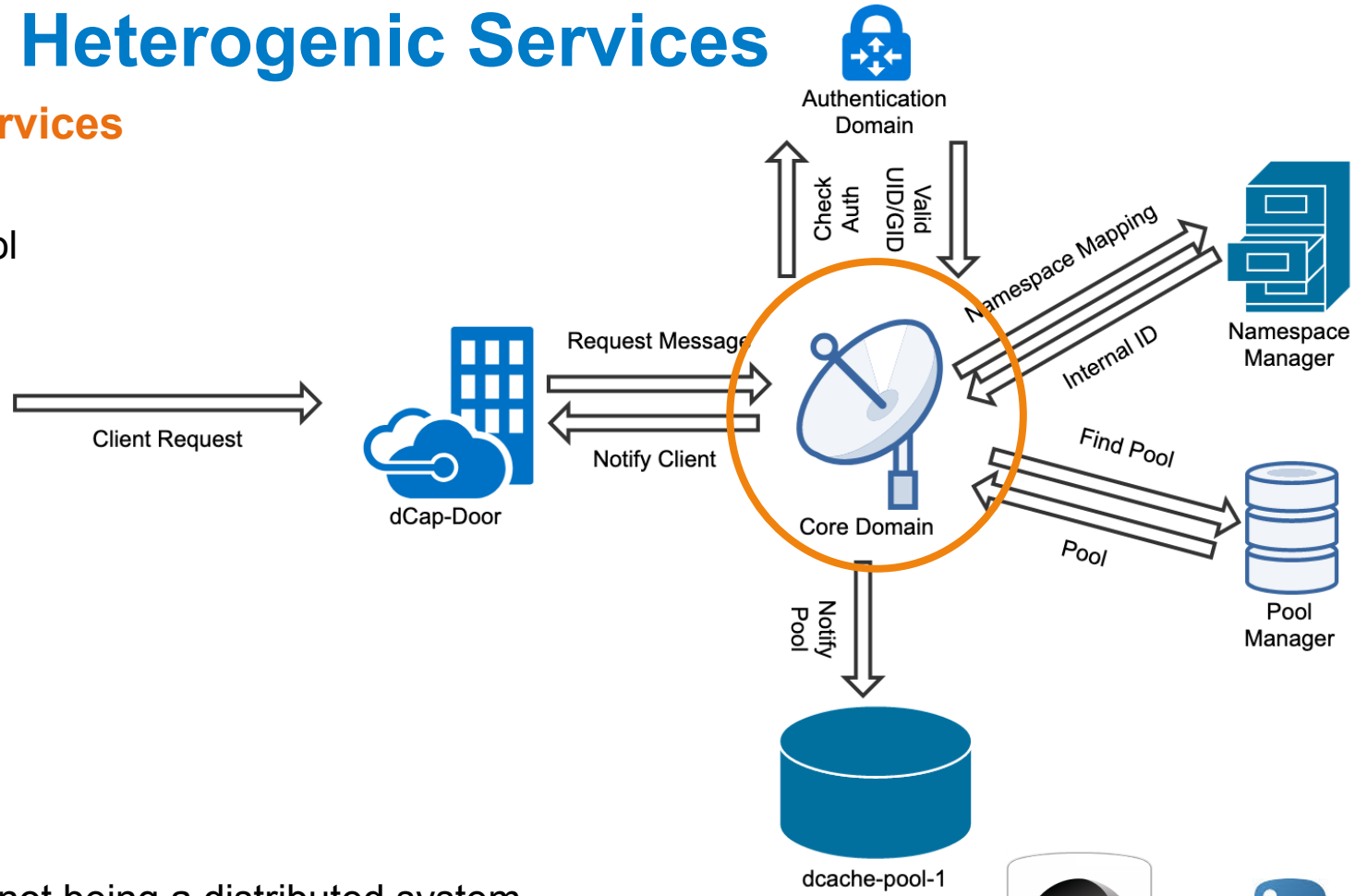
Define two categories of events

- Event itself contains all data: e.g. **Logging Events**
  → Consumer itself can perform data analysis
- Event points to data: e.g. **Storage Events**
  → Consumer needs to interface with the data or tools analysing the data

Communication between Heterogenic Services

Connecting Different and Independent Services

- Based on Micro-Services (1 service 1 task)
  - `Doors` – supporting each a different protocol
  - `Heads` – pool selection, Namespace, Auth
  - `Pools` – data storage and server



- Central Domain in dCache handling messages
- Usage of native Java object serialisation (JOS)
- Message hub internal to dCache

How can we apply this to our workflows

- Often employ a collection of 3rd party software:
  - 3rd party tools act as micro-services despite not being a distributed system
  - No message service → Employ specialised messaging tools
  - No unified internal messaging format → Use universal standards e.g. JSON
  - Make workflow message aware → Use universal bindings to messaging tools

python™

kafka

Make Our Sync&Share Event-Aware

Trigger the NextCloud File-Scan whenever a File is Written

- Connecting the dots → use events to register each new file automatically

Message Producer

- Offers a message stream by default
  → no need to develop a message producer
- Message stream is Kafka based

Notify

File written

Message Broker

kafka

- Widely used tool in industry
- Comes with bindings for all common languages
- A lot of tool can connect directly to Kafka
- Cope with high data rates
- Retains memory about last consumed messages

Consume &

Register File

Message Consumer

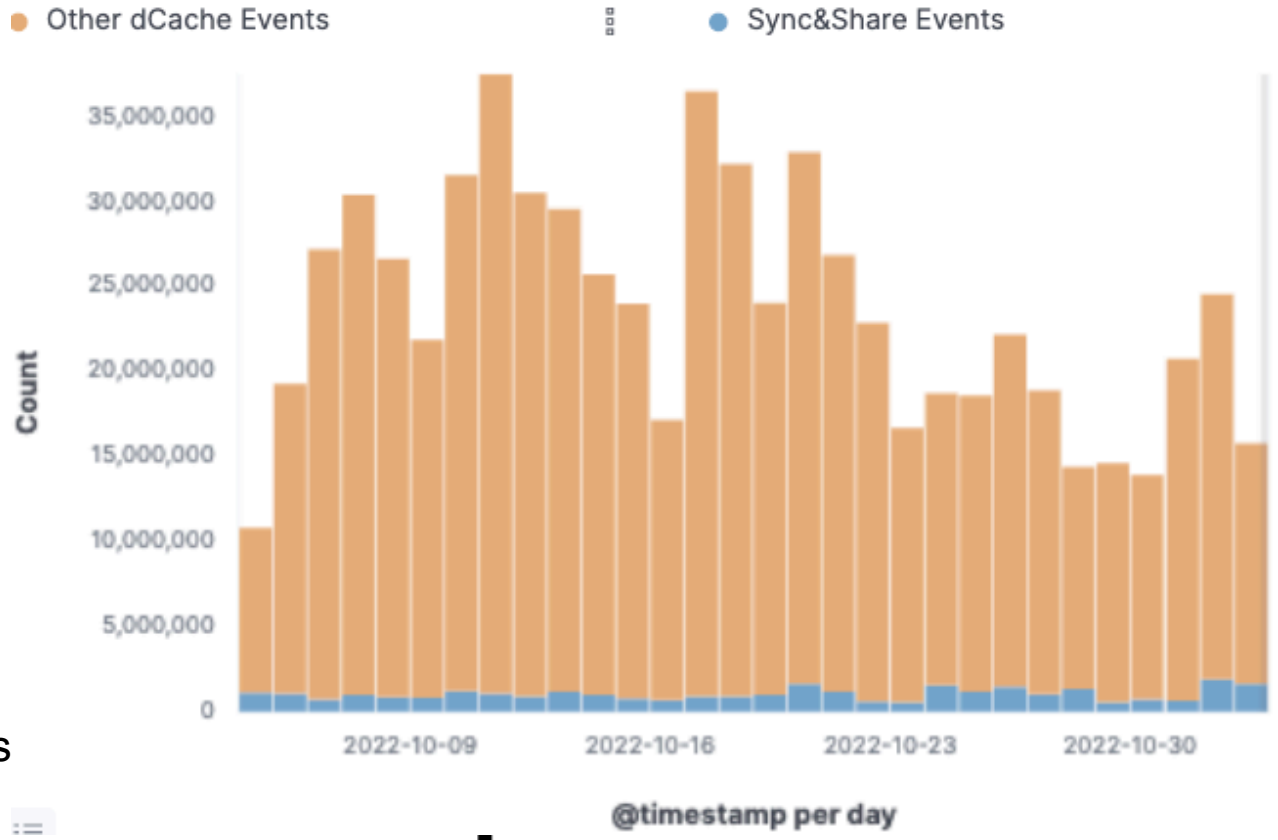
Nextcloud

- Unfortunately not message aware
  → need to develop a Kafka consumer to register files in NextCloud

Custom Consumer

dCache Operations Kafka Infrastructure

Number of Storage Events per Day



- Cluster of 5 nodes, incl. their Zookeeper cluster
- Average 15M events per day(small S&S contribution)
- Use one topic per instance
  → reduces number events we need to analyse

- Strong support from industry software
- Established as message broker with prominent customers
- Monitoring tools easily pluggable with Kafka
- Data analysis platforms can consume and analyse streams directly ⟶ Online data analysis
- Open API for Kafka for custom producers/consumers

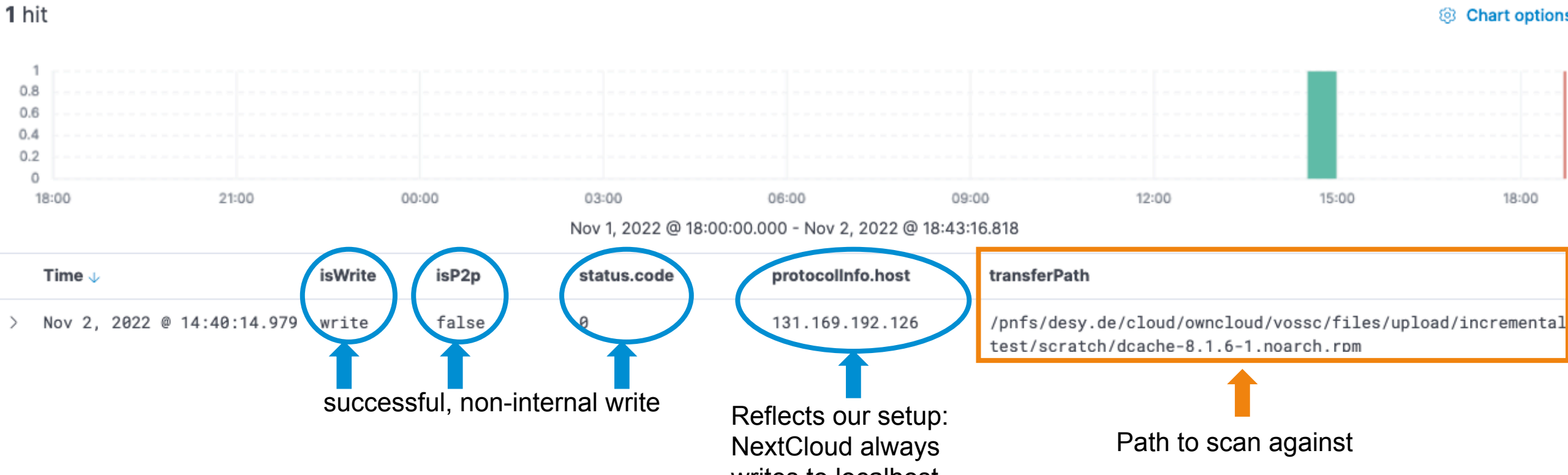
Custom Consumer

Custom producer

How to Find the Uploads in the Event Stream

Use a Custom Consumer

- Almost all accesses are made through NextCloud → allows to write a simple filter



successful, non-internal write

Reflects our setup: NextCloud always writes to localhost

Path to scan against

Adding the Consumer

Use Kafka Bindings to Trigger the Scans

python™

- NextCloud cannot evaluate the storage events → write a custom consumer
- Use the python bindings (easy to install via pip)
- Depends on Kafka-wrapper classes written by us
- Pluggable Kafka-consumer using a object of type `DOTKafkaMessageAnalyser`
- Deploy as SystemD-service on management node (due to necessary permissions)

Execute for every event

Filter selection

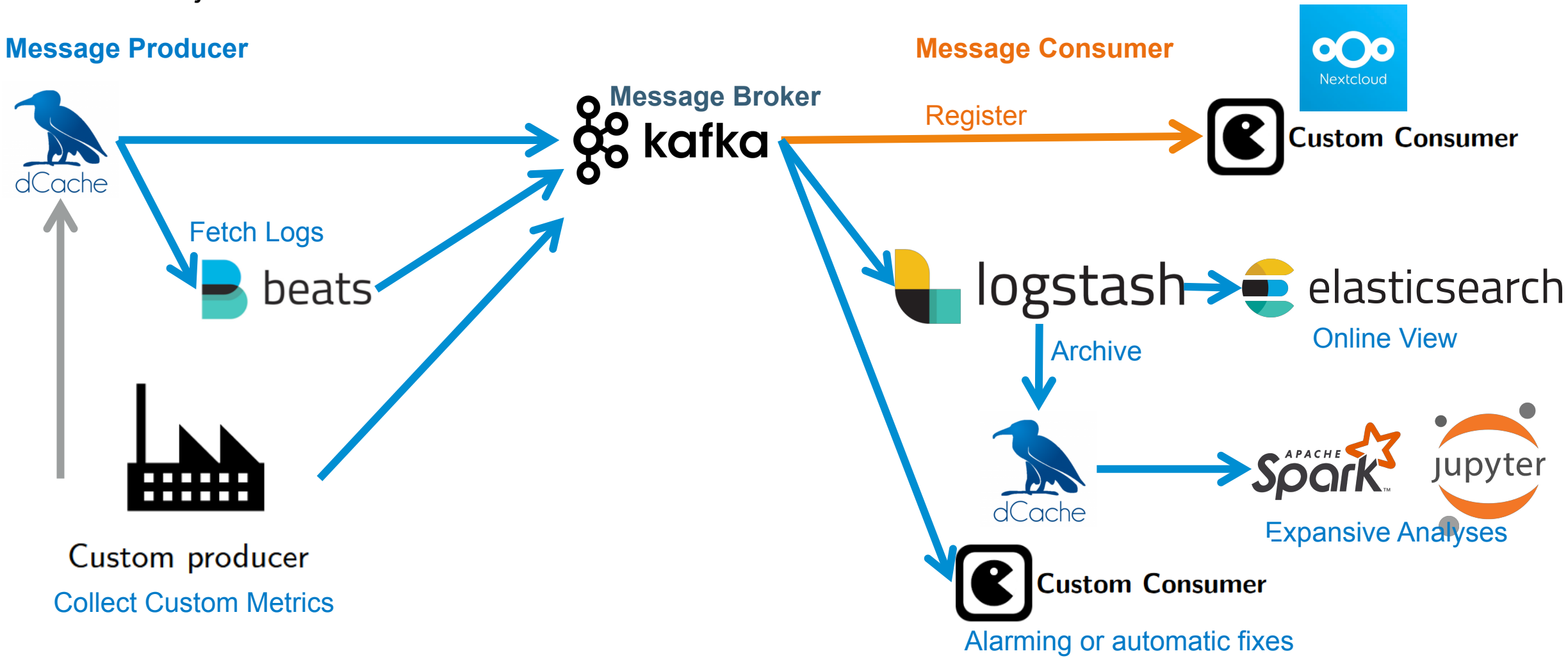
Trigger the scan

```
class ScanOnArrival(DOTKafkaMessageAnalyser):
    def __init__(self, instance = '', listenPath = '/', verbose = False) :
        self.instance = instance
        self.listenPath = listenPath
        self.verbose = verbose

    def execute_shell(self, command=''):
        systemcall = subprocess.Popen(command, stdout=subprocess.PIPE, stderr=subprocess.PIPE, shell=True)
        log, logerr = systemcall.communicate()
        return log

    def execute(self, message):
        message_data = message.value

        if message_data.get('msgType', None) == 'request' and \
           message_data.get('client',None) != '127.0.0.1' and \
           message_data.get('moverInfo',{}).get('isP2p', None) == False and \
           message_data.get('moverInfo', {}).get('isWrite', None) == 'write' and \
           message_data.get('moverInfo', {}).get('status',{}).get('code', None) == 0 and \
           self.listenPath in  message_data.get('transferPath',None):
            print("File: {} - written from {}; requires file-scan in NextCloud)".format(message_data.get('transferPath',None),
                                                                                        message_data.get('client', None)
                                                                                        ))
            file_scan_cmd = 'sudo -u apache php /var/www/nextcloud/occ files:scan --path "{}"'.format(scan_path)
            scan_log = self.execute_shell(command=file_scan_cmd)

            if self.verbose :
                print(scan_log)
```

The Larger Picture – Connection to Other dCache Consumer

Context to Other Active Consumers of dCache Storage Events

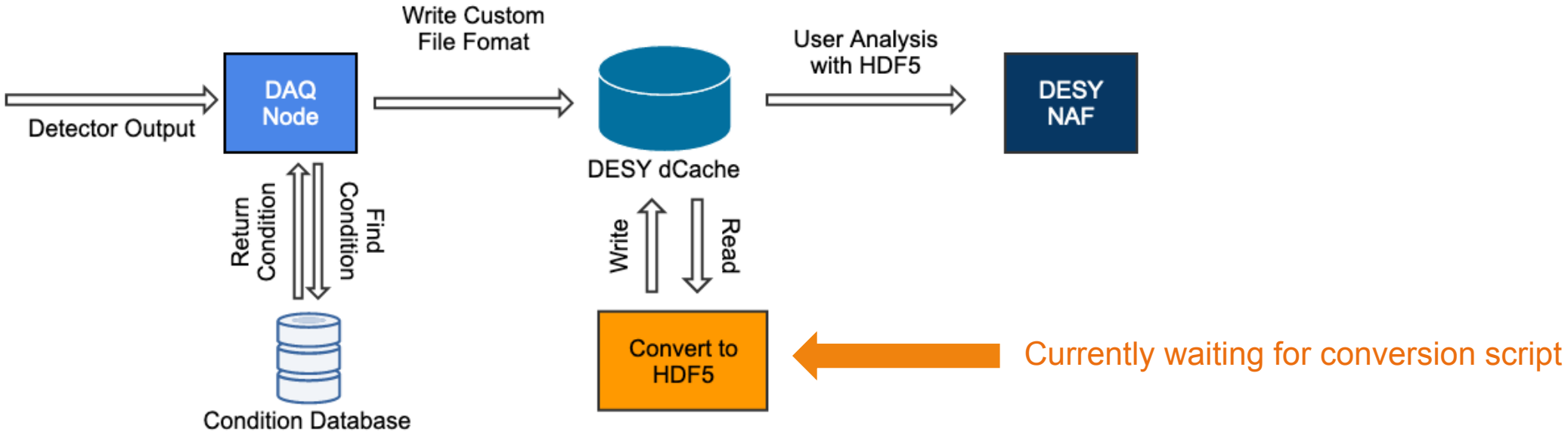
- NextCloud just another in our dCache infrastructure



Example for Similar Use case – ALPS II

Converting Custom Data Format to Default HDF5

- Next step: would other tasks profit from a similar workflow
- Existing similar work using Kafka events for PETRA 3 but method and Ansatz different to our solution
- Local experiment searching for Axions → write custom format but aim for HDF5 for user analysis



- New complication: permission to read/write data files → exists dedicated user to use together with NFS/POSIX ACLs
- Run regular batch jobs doing the conversion: ALPS scientists have limited experience there
- Our idea: put a similar service to NextCloud into production using NAF resources (details still to be discussed)

Scale-Out – Cloud Based Ansatz

Use Modern Tools Instead of Dedicated SystemD Service

- Described setup sounds familiar? → sounds like AWS Lambda; suited for Function as a Service workflow
- Our consumers are
  - Python Scripts
  - Containerised easily for a Kubernetes based deployment
  - Serverless: take Kafka message → manipulate/analyse → push on to next service
- Showstoppers:
  - Overall lack of experience in our dCache operations group
  - Some applications are not always serverless
    → NextCloud consumer needs be run on a NextCloud no
    → ALPS II use case needs permissions to r/w to dCache (currently based on NFS)
- Colleagues investigated these problems in the past: https://doi.org/10.5281/zenodo.3599569
- Handling of access tokens to dCache, HTCondor and FaaS service itself proved difficult
- Follow up on these in the future

Scale-Out – Spark and Workflow Management Systems

Managing Compute Intensive Consumer/Integration into Larger Infrastructure

- Currently our consumers are not CPU intensive → might change in the future
- ALPS II conversion might offer insight if we need more dedicated scale-out into our clusters

Possible Scale-out: Apache Spark

- Spark ships with Kafka streaming build in → collect 1k messages and then compute each in the cluster
- Adopted Spark in 2018 for our data analysis tasks; migrated certain Spark-jobs successfully as HTCondor Jobs on NAF
- Offer a start to think about dealing with CPU intensive tasks

Wider Picture: Make Use of Workflow Management Systems

- Instead of off-loading to Spark, why not offload to the batch systems on NAF directly
- Rather home-brewed setup → should easily be integrated into a more general service
- DESY lacks a dedicated Workflow Management System for us to integrate → might change in the future

Summary

Using dCache Kafka Storage Events with external Consumers

- Migrated DESY Sync&Share to regular dCache and NextCloud releases early in 2022
- Begin integrating NexCloud into our regular scientific infrastructure using dCache storage events and access protocols
- Build on our setup and development using Kafka messages for logging, accounting, monitoring and analytics for dCache

- Find our Workflows adaptable for certain scientific use cases for our local experiments
- Work remains to merge this into a modern setup using container-based setups like Kubernetes
- Merge our effort into a Workflow Management System once available

Thank you