Monitoring Large-Scale dCache Installations with Kafka streams

27th Conference on Computing in High Energy and Nuclear Physics

Felix Christians, Sandro Grizzo, Jürgen Hannappel, Thomas Hartmann, Tigran Mkrtchyan, Marina Sahakyan, Christian Sperl, Alexander Trautsch, Christian Voß

Krakow, 21st October 2024

HELMHOLTZ

DESY.

Mass-Storage for Different Communities

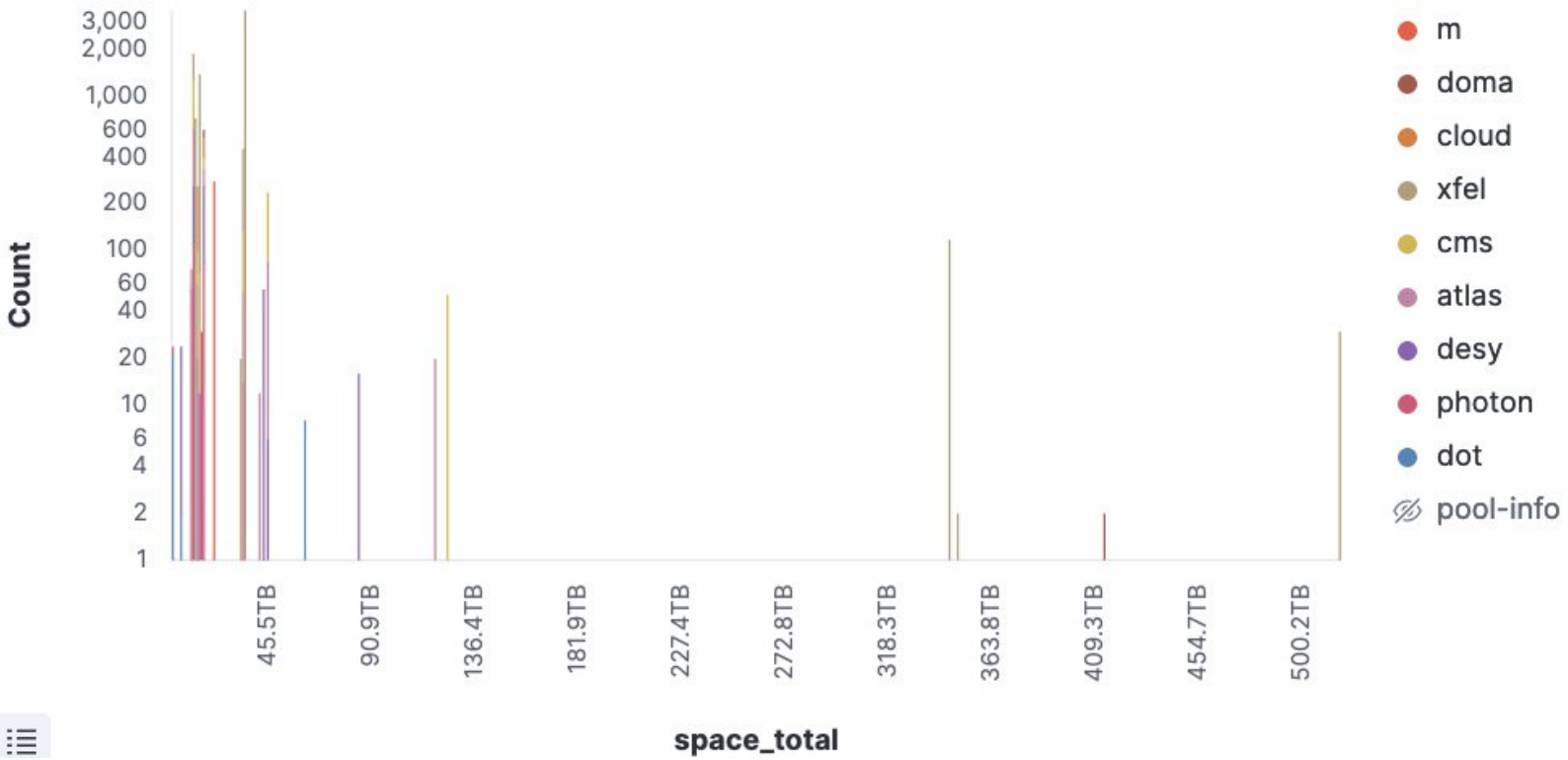
dCache as Central Mass Storage

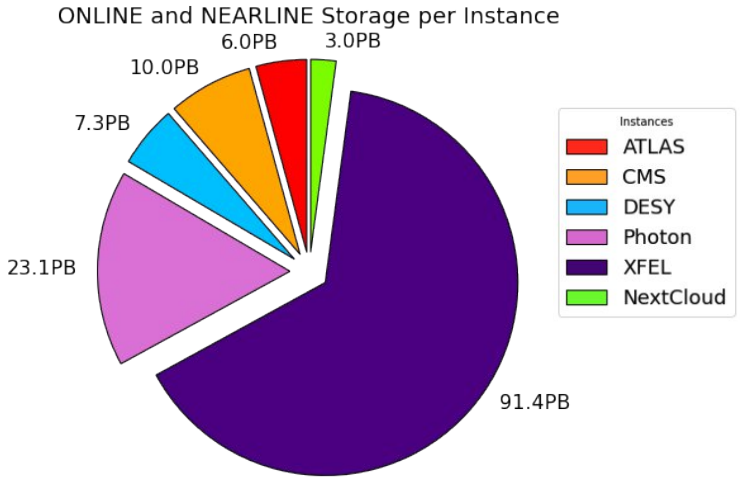
- Central element in overall storage strategy
- Collaborative development under open source licence by
  - DESY
  - Fermilab
  - Nordic E-Infrastructure Collaboration (inoffically NDGF)
- Particle Physics in general
  - In production at 9 of 13 WLCG Tier-1 centres
  - In use at over 60 Tier-2 sites world wide
  - 75% of all remote LHC data stored on dCache
  - In addition: Tevatron and HERA data
  - All smaller DESY experiments store data in dCache
- Photon Science
  - Raw-Data for all DESY light sources
  - Long-term archival
  - Large cache for machine division

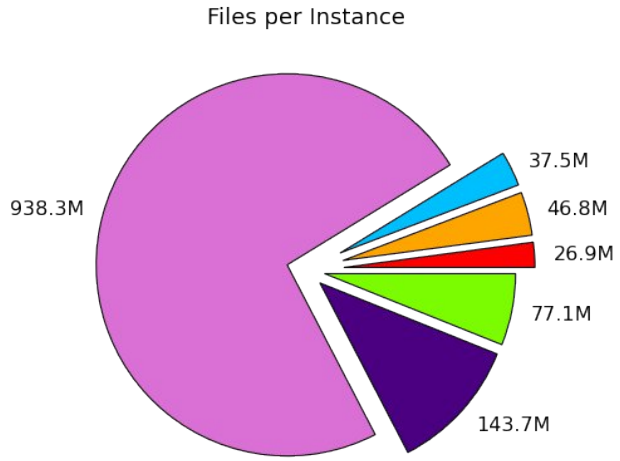
Overview of our Instances at DESY-HH

Number of Hosts and Stored Data

- Eight large dCache instances with about 700 pool Nodes and 5500 pools
- One pre-production instance for dCache Operations for testing and development



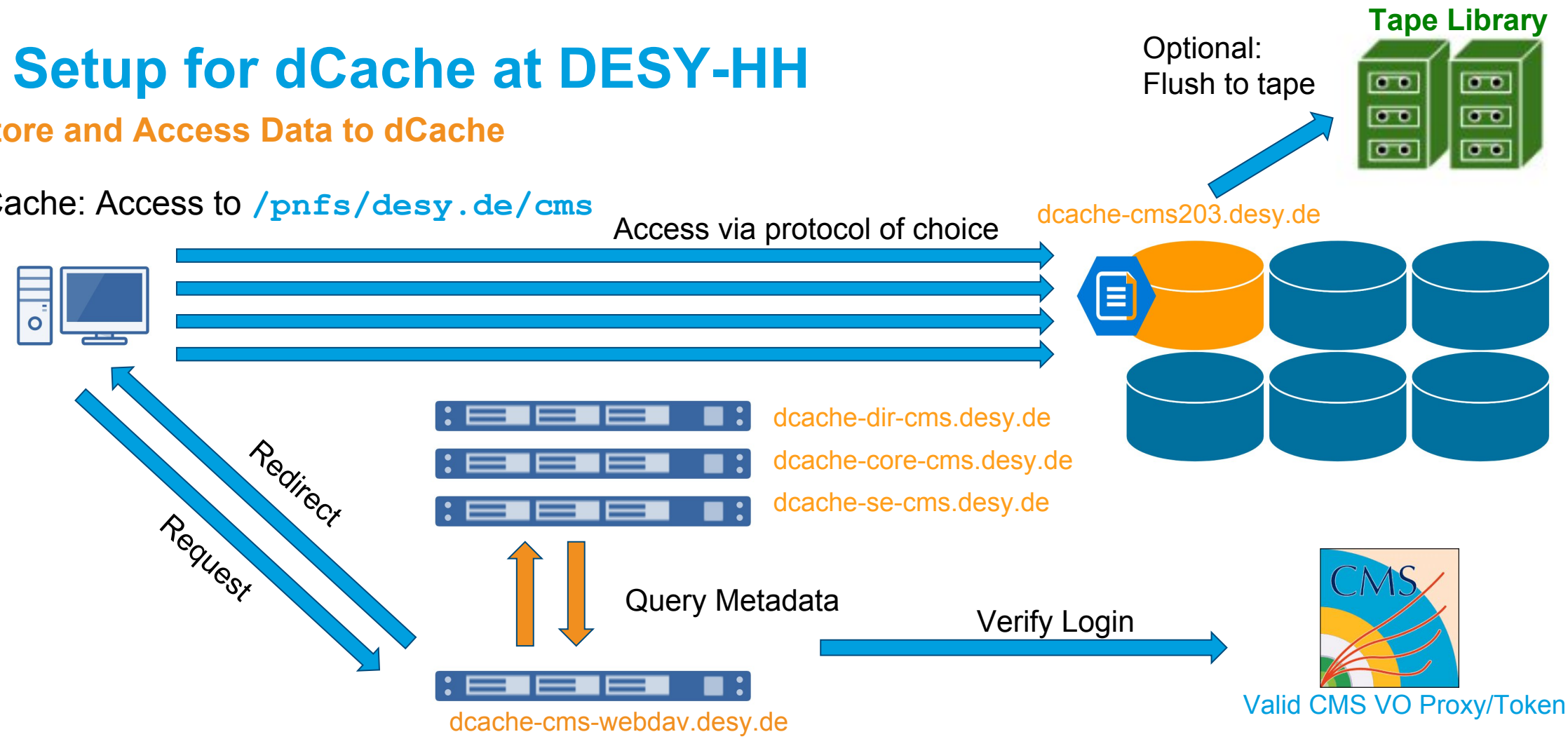




Basic Setup for dCache at DESY-HH

How to Store and Access Data to dCache

- Use dCache: Access to `/pnfs/desy.de/cms`



- Files are stored as a single copy on a storage server (replication used only used for Sync&Share/HIFIS)
- Storage nodes currently are RAID-6 setups with multiple virtual disks mapping to dCache pools
- Pool nodes are very heterogenic ➜ 200TiB 14 disk systems to 700TiB 96 disk systems, current: 520TiB 28 disk systems
- Pool size and pool number very heterogenic ➜ 10 small pools per node (DESY dCache) to single 520TiB pools for XFEL

Classical Monitoring for dCache at DESY-HH

Probes and Log-Files

- Split monitoring between three groups
  - Monitoring of the storage hardware: Operational team in the computing centre (based on tools provided by the hardware manufacturer, `Icinga`, `Grafana`)
  - Monitoring of the Linux services: Linux team for services like `SSH` or `Puppet` (Tools like `Icinga` and ELK stack)
  - **Monitoring of dCache services: dCache Operations Team**

Monitoring dCache:

- Log files from dCache services (transitioned to `SystemD` with dCache 6.2)
- Information available in the dCache admin-domain/REST-API
- Billing files for active traffic (replaced by a Kafka-stream in dCache 4.2)
- Access-log still stored in files (non-transfer access)

Multiple Sources all with different Characteristics to Collect into a Single View

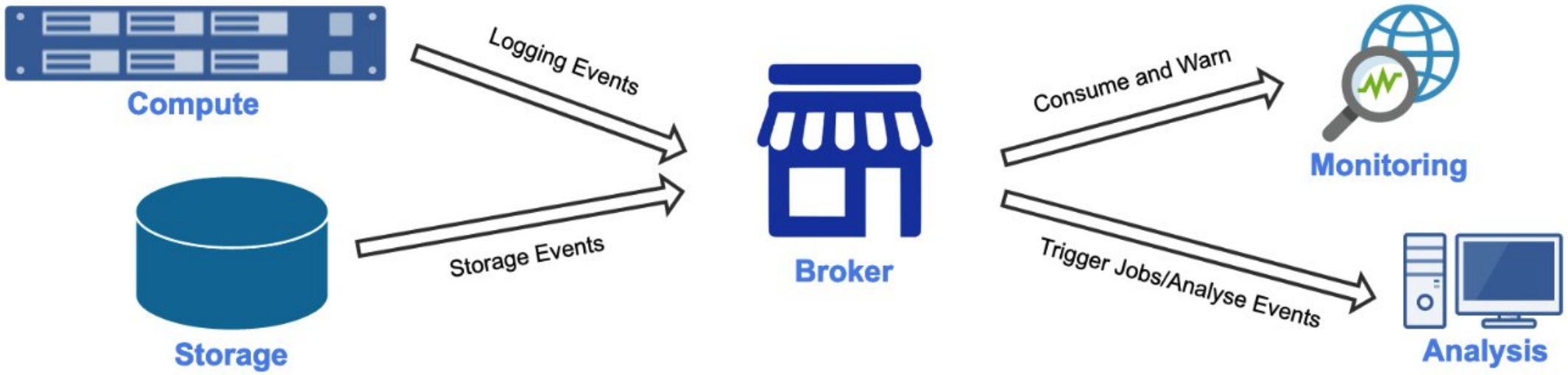
Monitoring for dCache Using Kafka

Introduction to Kafka

- Kafka is a message broker system allowing services to send and receive messages
- Central element is a message broker ensuring load balancing and resilience
- Widely used tool in the IT industry and thus scales very well
  ➜ We produce about 10-20M messages a day; our brokers have a load of 0.05
- Wide usage means a lot of other tools speak Kafka natively
- APIs well supported through C/C++, Python and more

kafka

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



From dCache to Analysis

Connecting dCache with Monitoring Infrastructure

- Apache Kafka common IT tool → a lot of services can subscribe and consume Kafka messages

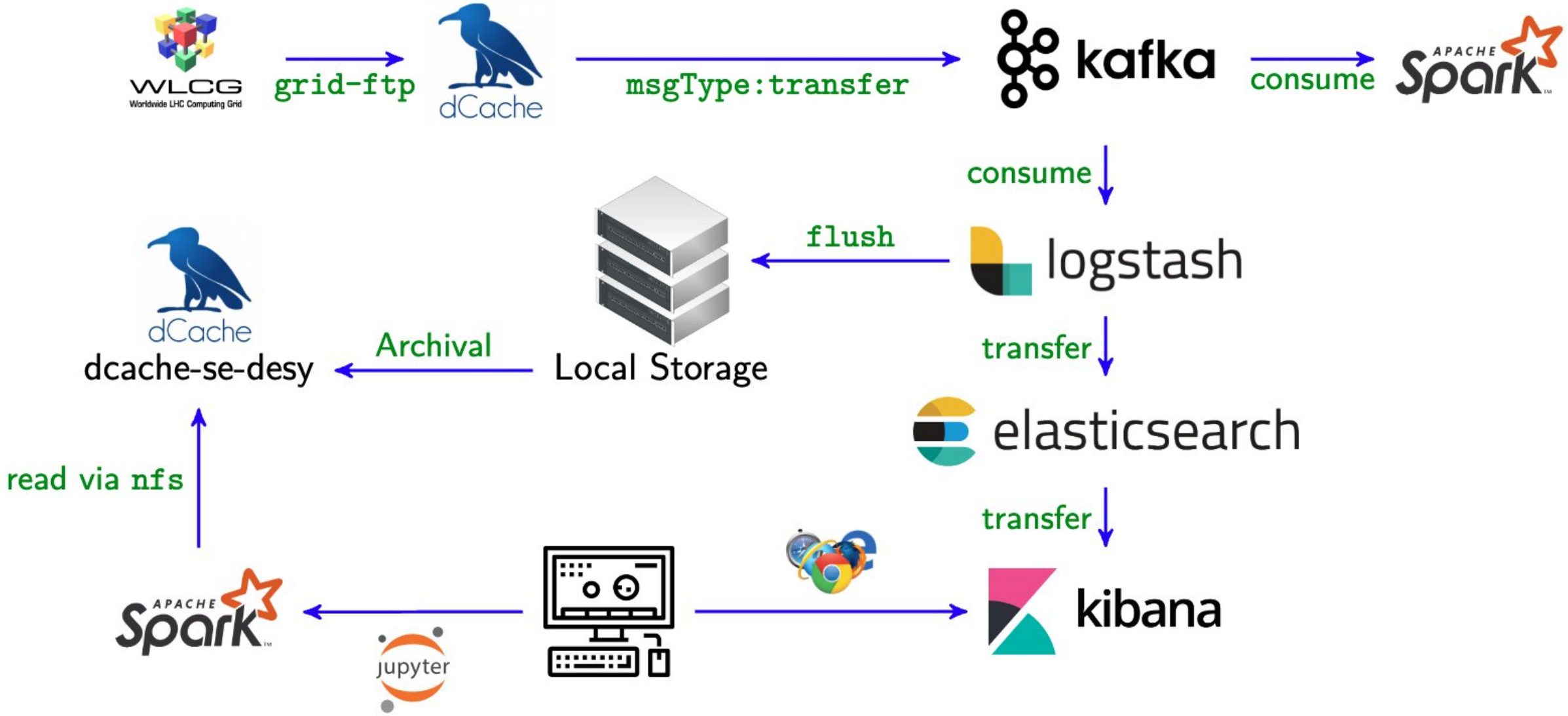
- Beats can easily push JournalD, Syslog, local files, … into Kafka
- DESY-IT provides dedicated ELK infrastructure
- Logstash and ElasticSearch accepts Kafka streams → direct in ingest and visualisation with Elastic tool box (classic monitoring)

Benefit of Plugging Kafka in Between?

- Uniform data format and infrastructure behind Kafka; only the producer needs to be provided
- Kafka allows **push** rather then **pull** monitoring as e.g. with Icinga Probes
- Have operations triggered on Kafka messages → Listen passively rather than search actively
- Often have well known signatures in billing/logging → introduce an automatic response (alarm or fix)
- Have a number of lightweight python based consumers in operation listening for specific patterns
- Complex analysis frameworks can consume Kafka events at scale using a sliding time window

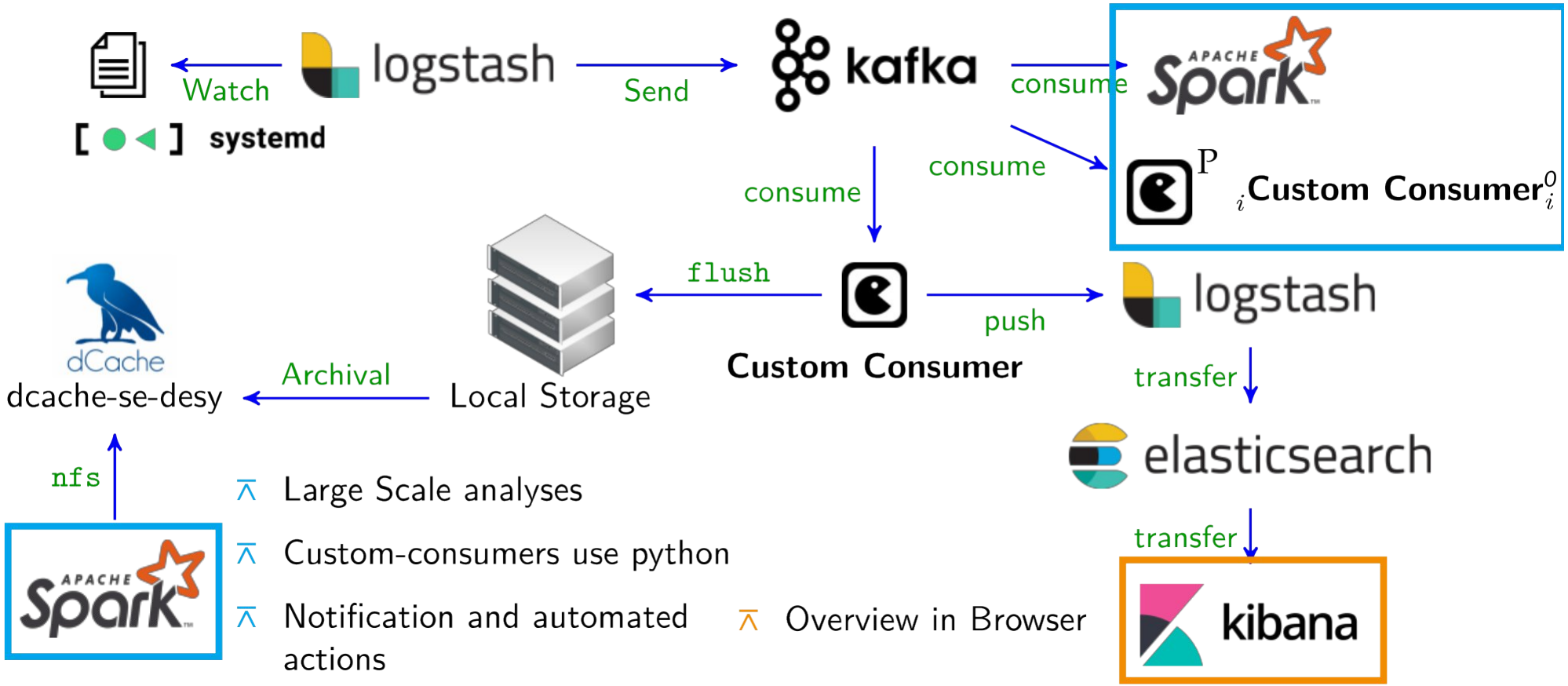
Step 1: Kafka Billing Stream Workflow

Native Kafka Event Stream Provided by dCache



Step 2: dCache Logging Workflow

Example for Loggings Stream



Example for Custom Consumers

Alarming if Pools Become Disabled

- Apache Kafka Logging-Stream listens to all logging streams
- Check each message if the message contains e.g. ‘pool mode changed to disabled’
- Generate an email to admins and enter entry to database ➜ Trigger automated restarts of dCache pools
- Notify if pool recovers and remove entry in database

Time: 07:51:05 (+0000) - 2024.06.06, Stale entries: 1779, Host: christif@naf-it01.desy.de

Pool Domain	Pool Group	Error Status	Error Type	Error Message	Date of latest Occurrence
dcache-xfel497-01	xfel	unresolved	PoolDisabled	[] Pool mode changed to disabled(fetch,store,stage,p2p-client,p2p-server,dead): Shutdown	2024-06-06 07:49:55.404000+00:00
dcache-xfel498-01	xfel	unresolved	PoolDisabled	[] Pool mode changed to disabled(fetch,store,stage,p2p-client,p2p-server,dead): Shutdown	2024-06-06 07:50:06.872000+00:00
dcache-xfel500-01	xfel	unresolved	PoolDisabled	[] Pool mode changed to disabled(fetch,store,stage,p2p-client,p2p-server,dead): Shutdown	2024-06-06 07:50:30.024000+00:00
dcache-xfel499-01	xfel	unresolved	PoolDisabled	[] Pool mode changed to disabled(fetch,store,stage,p2p-client,p2p-server,dead): Shutdown	2024-06-06 07:50:18.443000+00:00

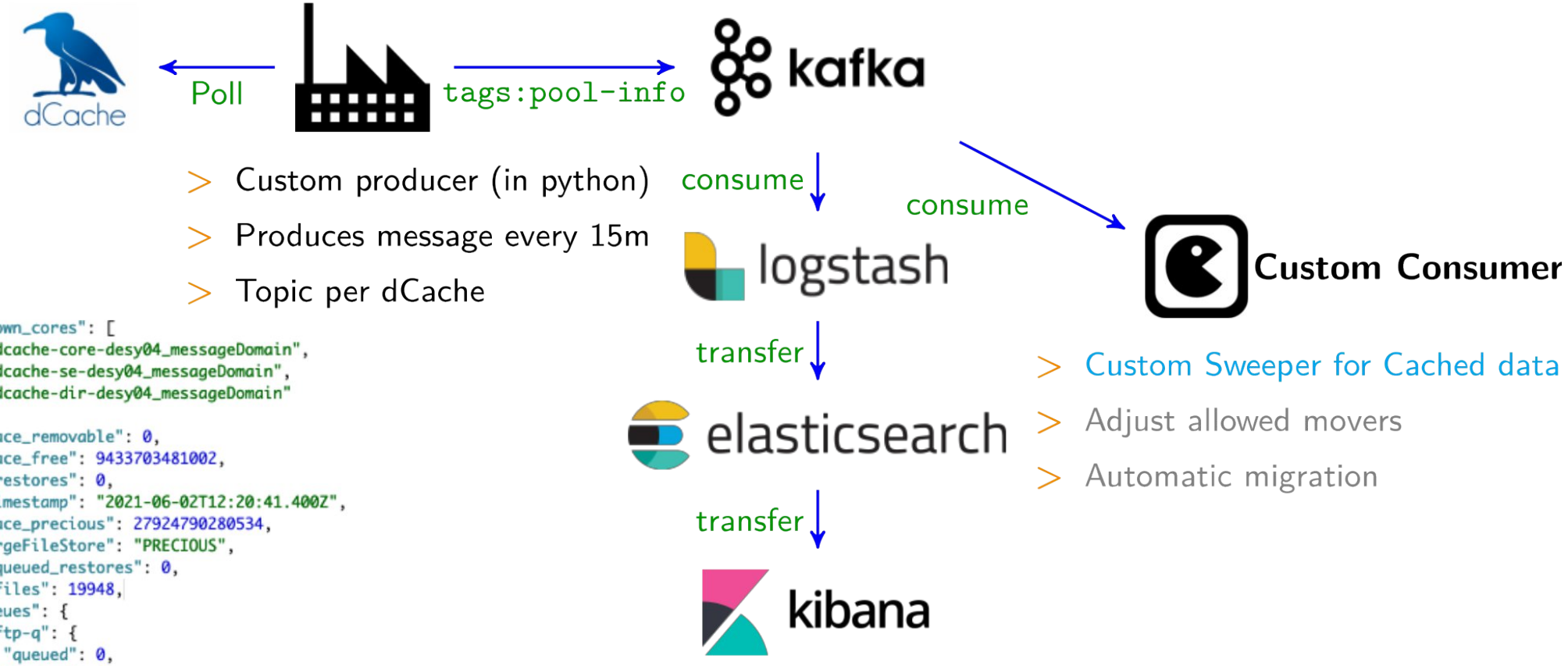
At most 300 entries are shown. Refer to the database to get all entries.

Legend:

unresolved	The issue persists
resolved	The issue has been resolved
stale	The issue has persisted for multiple days. The entry must be manually reviewed and pruned
finalized	The entry will be deleted momentarily
temporary	The issue has no associated resolved-messageand will be treated as self-resolving
oddity	The issue is treated as self-resolving and doesn't warrant an email notification on its own
permanent	The issue will not be deleted and must be removed manually.
no-delay	This issue will always produce a notification

Step 3: Custom Data Stream

Collect Data for Each Pool Every 15min




```
"known_cores": [
  "dcache-core-desy04_messageDomain",
  "dcache-se-desy04_messageDomain",
  "dcache-dir-desy04_messageDomain"
],
"space_removable": 0,
"space_free": 9433703481002,
"n_restores": 0,
"@timestamp": "2021-06-02T12:20:41.400Z",
"space_precious": 27924790280534,
"LargeFileStore": "PRECIOUS",
"n_queued_restores": 0,
"n_files": 19948,
"queues": {
  "ftp-q": {
    "queued": 0,
    "active": 0
  },
```


- Pool plot shown earlier based on this stream
- Also archived on dCache → allows forensics on which pool belonged where in the past

dCache Data Analytics

How to Process Large amounts of Data

- Apache Kafka Streams are archived to dCache ➜ Can be analysed at scale using e.g. Apache Spark on our National Analysis Facility
- Use Spark to analyse up to 20 TiB of our billing archive ➜ Treasure trove for ML (collaborate with BNL)

Spark Master at spark://htc-it02.desy.de:3000

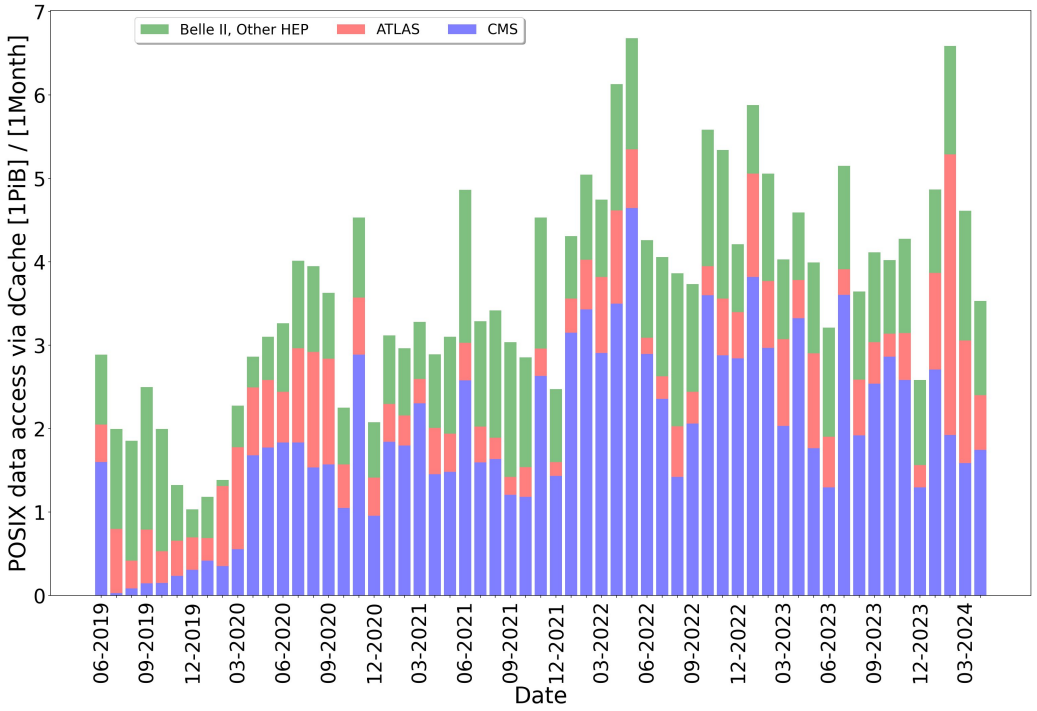
URL: spark://htc-it02.desy.de:3000
Alive Workers: 28
Cores in use: 112 Total, 112 Used
Memory in use: 448.0 GiB Total, 448.0 GiB Used
Resources in use:
Applications: 1 Running, 1 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (28)

Worker Id	Address	State	Cores	Memory	Resources
worker-20240606144752-batch1554.desy.de-4048	batch1554.desy.de:4048	ALIVE	4 (4 Used)	16.0 GiB (16.0 GiB Used)	
worker-20240606144752-batch1554.desy.de-4058	batch1554.desy.de:4058	ALIVE	4 (4 Used)	16.0 GiB (16.0 GiB Used)	
worker-20240606144752-batch1554.desy.de-4063	batch1554.desy.de:4063	ALIVE	4 (4 Used)	16.0 GiB (16.0 GiB Used)	
worker-20240606144752-batch1554.desy.de-4068	batch1554.desy.de:4068	ALIVE	4 (4 Used)	16.0 GiB (16.0 GiB Used)	

- Investigating on how to use Spark as a Kafka consumer
- New APIs not really easy to set up
- Apply training to sliding window

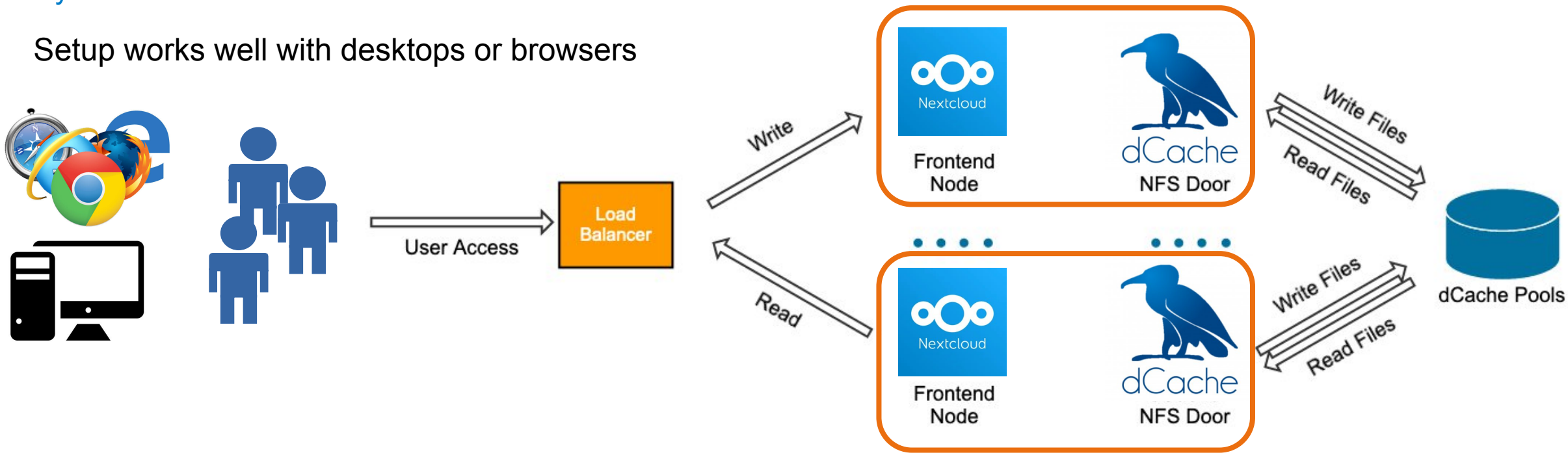
```
Analysing cms billing for 2019-06
+----------------+
|Data            |
+----------------+
|1801640603785806|
+----------------+

Analysing cms billing for 2019-07
+--------------+
|Data          |
+--------------+
|34751126730552|
+--------------+
```



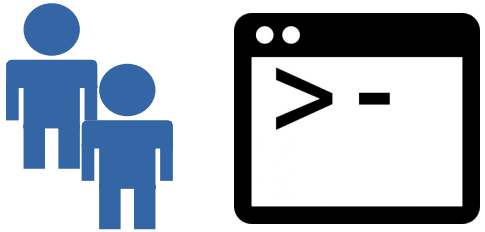
Improving Other Serviced: Example DESY Sync&Share

Service Overview

- Sync&Share service based on NextCloud and dCache
- Setup works well with desktops or browsers



- How to fit in access from compute clusters?



Issue: Files need to be registered in NextCloud

- Write data through dCache produces dark data in NextCloud
- Possible to do file-scans to register but cron-jobs and full scans insufficient
- Use dCache related Kafka events

Make Our Sync&Share Event-Aware

Trigger the NextCloud File-Scan whenever a File is Written

Message Producer

dCache

Notify

File written, (re)moved, Directory created

Message Broker

kafka

Consume &

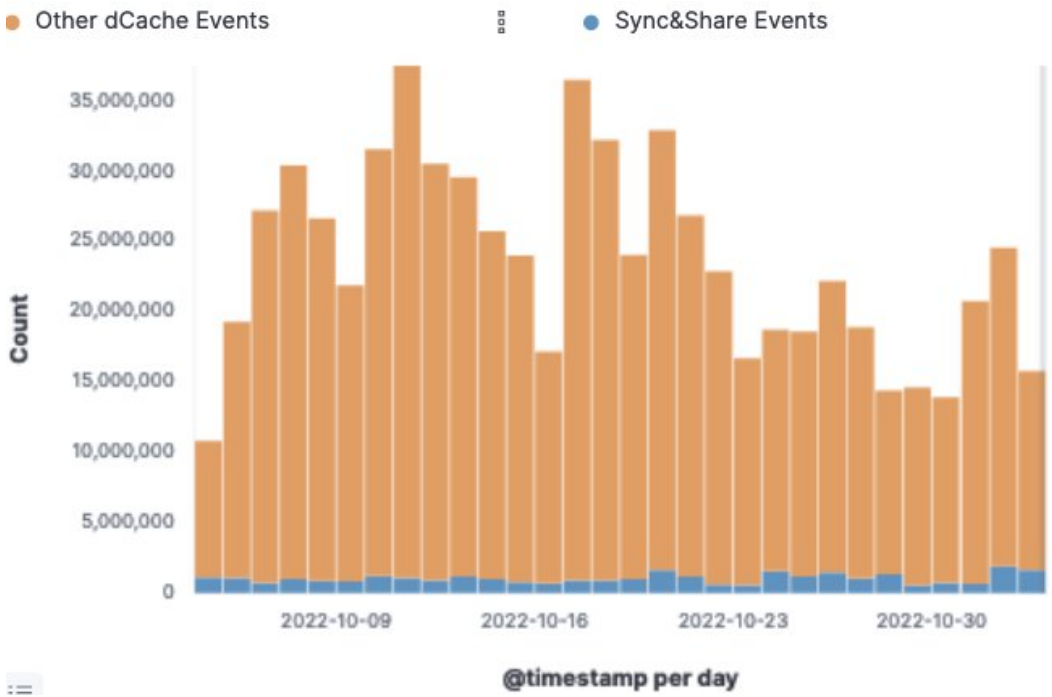
Scan dCache

Message Consumer

Nextcloud

- Offers a message stream by default
  → no need to develop a message producer
- Message stream is Kafka based
- But the Billing stream is not sufficient → need access logs
- Access logs pushed into Kafka through file-beat

beats logstash



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

Custom Consumer

Adding the Consumer

Use Kafka Bindings to Trigger the Scans

- 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)
```

Summary

dCache Operations at DESY-HH

- DESY-HH offers about 150PiB of disk space on 700 nodes with 5500 pools

- Our monitoring is all based aroung Kafka and ElasticSearch/Kibana
- We use Kafka to monitor
  - dCache transfers, (re)stores to tape, removal of files from pools
  - dCache internal logging for all domains
  - Poll dCache for additional data to receive a fuller picture

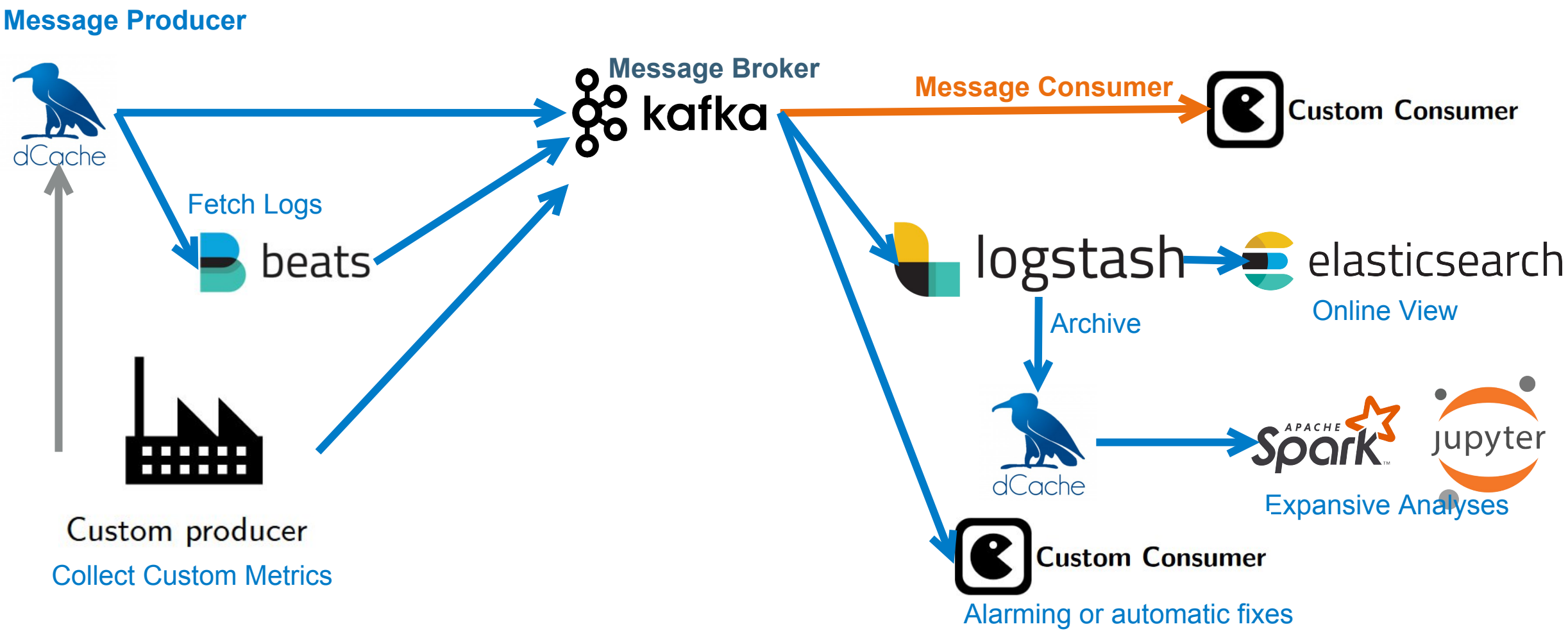
- Use archived Kafka data as source for extensive data analysis
- Our tool of choice for data analysis is Apache Spark

- We lack manpower to experiment with the full AI/ML ICBM-Toolsets
- Teamed up with BNL: They have the ML expertise, we have the infrastructure for easy data aquisition

Thank you, any Questions

The Larger Picture – Connecting to Other dCache Consumers

Context to Other Active Consumers of dCache Storage Events



Example for Custom Consumers

Alarming if Pools Become Disabled

- Apache Kafka Logging-Stream listens to all logging streams
- Check each message if the message contains e.g. ‘pool mode changed to disabled’
- Generate an email to admins and enter entry to database
- Notify if pool recovers and remove entry in database

Time: 07:51:05 (+0000) - 2024.06.06, Stale entries: 1779, Host: christif@naf-it01.desy.de

Pool Domain	Pool Group	Error Status	Error Type	Error Message	Date of latest Occurrence
dcache-xfel497-01	xfel	unresolved	PoolDisabled	[] Pool mode changed to disabled(fetch,store,stage,p2p-client,p2p-server,dead): Shutdown	2024-06-06 07:49:55.404000+00:00
dcache-xfel498-01	xfel	unresolved	PoolDisabled	[] Pool mode changed to disabled(fetch,store,stage,p2p-client,p2p-server,dead): Shutdown	2024-06-06 07:50:06.872000+00:00
dcache-xfel500-01	xfel	unresolved	PoolDisabled	[] Pool mode changed to disabled(fetch,store,stage,p2p-client,p2p-server,dead): Shutdown	2024-06-06 07:50:30.024000+00:00
dcache-xfel499-01	xfel	unresolved	PoolDisabled	[] Pool mode changed to disabled(fetch,store,stage,p2p-client,p2p-server,dead): Shutdown	2024-06-06 07:50:18.443000+00:00

At most 300 entries are shown. Refer to the database to get all entries.

Legend:

unresolved	The issue persists
resolved	The issue has been resolved
stale	The issue has persisted for multiple days. The entry must be manually reviewed and pruned
finalized	The entry will be deleted momentarily
temporary	The issue has no associated resolved-messageand will be treated as self-resolving
oddity	The issue is treated as self-resolving and doesn't warrant an email notification on its own
permanent	The issue will not be deleted and must be removed manually.
no-delay	This issue will always produce a notification

Developed by Felix Christians

Custom Kafka Consumers

An Example for dCache Kafka Data

- Apache Kafka Stream-API allows an easy development e.g. in Python
- Kafka JSON structures makes handling with data structures convenient and easy
- Python makes also further actions easy to develop down to employing ML/AI methods

Sync&Share File Registration

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)
```