Rucio Concepts

and principles

Rob Gardner, Benedikt Riedel
University of Chicago

Mario Lassnig
CERN

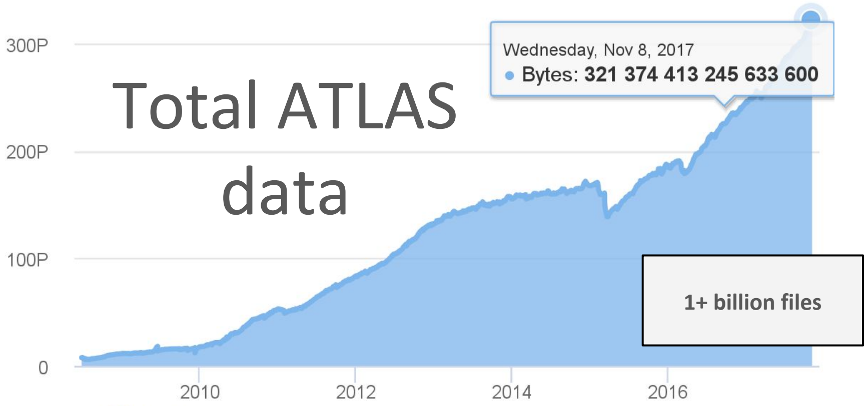
Open Science Grid Blueprint
December 8, 2017

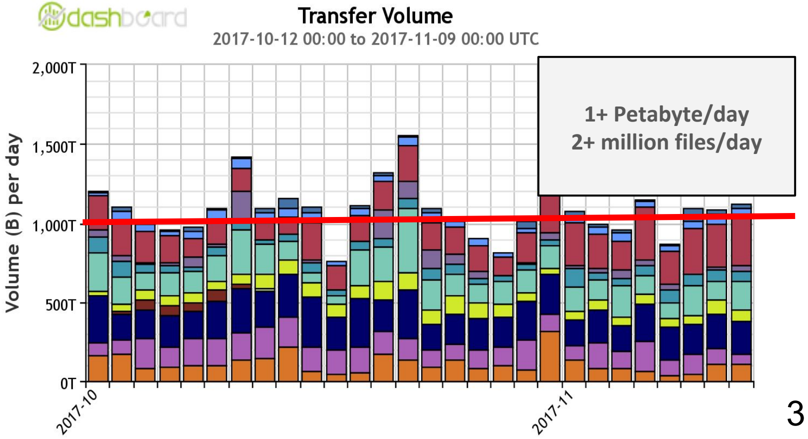
This talk

- These slides are a compendium of individual topics relevant for input to further discussion today
- special thanks to Mario Lassnig who provided the vast majority of input

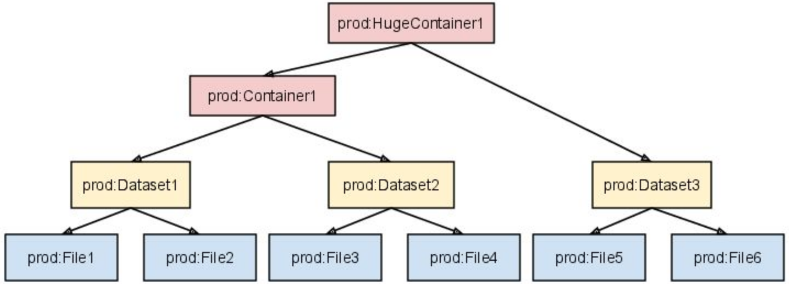
Rucio in a nutshell

- Main functionalities
  - Discovery, Location, Transfer, Deletion
  - Quota, Permission, Consistency
  - Monitoring, Analytics
  - Can enforce computing models
- Integration with workload management
- Automation of operations
- Enables heterogeneous data management
  - No vendor/product lock-in
  - Able to follow the market

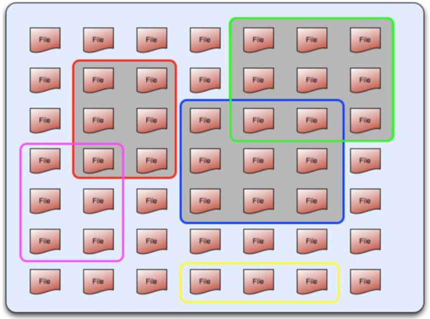




Namespace handling



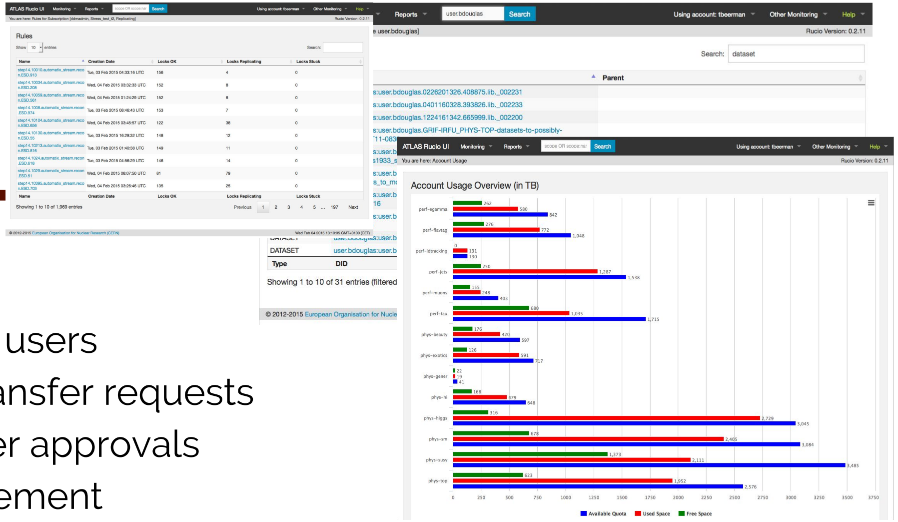
- Smallest addressable unit is the file
- Files can be grouped into datasets
- Datasets can be grouped into containers
- Names are partitioned by scopes
    - To distinguish users, groups and activities
    - Accounts map to users/groups/activities
- Multiple data ownership across accounts
- Large set of available metadata, e.g.
    - Data management: size, checksums, creation times, access times, …
    - Physics: run identification, derivations, events, …
    - …



Declarative data management

- Express what you want, not how you want it
  - *e.g., "3 copies of this dataset, distributed evenly across two continents, with 1 copy on TAPE"*
  - Rules can be dynamically added and removed by all users, some pending authorisation
  - Evaluation engine resolves all rules and tries to satisfy them by with transfers/deletions
- Replication rules
  - Lock data against deletion in particular places for a given lifetime or pin
  - Primary replicas have indefinite lifetime rules
  - Secondary replicas are dynamically created replicas based on traced usage and their access popularity
- Subscriptions
  - Automatically generate rules for newly registered data matching a set of filters/metadata
  - e.g., spread *project=data17_13TeV* and *data_type=AOD* evenly across *T1s*

Monitoring



- RucioUI
  - Provides several views for different types of users
  - Normal users: Data discovery and details, transfer requests
  - Site admins: Quota management and transfer approvals
  - Admin: Account / Identity / Storage management
- Monitoring
  - Internal system health monitoring (Graphite / Grafana)
  - Transfer / Staging / Deletion monitoring using industry-stranding architectures (ActiveMQ / Kafka / Spark / HDFS / ElasticSearch / InfluxDB / Grafana)
- Analytics
  - Periodic full database dumps to Hadoop (pilot traces, transfer events, …)
  - Used studies, e.g., transfer time estimation which is now already in a pre-production stage

Third party copy

- Rucio provides a generic transfertool API
  - `submit_transfers()`, `query_transfer_status()`, `cancel_transfers()`, …
  - Independent of underlying transfer service
  - Asynchronous interface to any potential third-party tool

- Currently only available implementation of transfertool API is FTS3
  - Additional notification channel via ActiveMQ for instant acknowledgments
  - Potential to include GlobusOnline for improved HPC data transfers

- FTS3 Deployment
  - CERN Pilot, CERN Production, RAL Production, BNL Production
  - We distribute our transfers across all FTS3 servers based on file destination
    - (We also have one dedicated for OSG use in production)

Topology

- Storage systems are abstracted as *Rucio Storage Elements (RSEs)*
  - Logical definition, not a software stack
  - Mapping between activities, hostnames, protocols, ports, paths, sites, …
  - Define priorities between protocols and numerical distances between sites
  - Can be tagged with metadata for grouping
  - Files on RSEs are stored deterministically via hash function
    - Can be overridden (e.g., useful for Tier-0, TAPE, fixed data output experiments, …)
- Rucio's topology can exist standalone outside an information catalogue
  - However, for a non-trivial amount of sites this can quickly become infeasible
    - We suggest to have a flexible way of describing resources
  - For ATLAS, we use AGIS (ATLAS Grid Information System) and sync to Rucio via Nagios
  - AGIS is now evolving into generic CRIC (Computing Resource Information Catalogue)

Key design principles

- Horizontal scalability of servers and services
- Data streams
    - Stateless API — serve each request independently
    - Servers can handle arbitrary length responses (e.g., list 1 billion files)
- Work sharding
    - All daemons share their work-queues
    - Algorithm for work selection independent of length of workqueue!
    - Elastic and fail-safe
        - If one service goes down (e.g, node failure) others take over automatically, no need to reconfigure or restart
- Fault-tolerance
    - Fail hard and early, but keep running and retry once up

Rucio daemons and operations

- 10 daemons
  - Minimum 2 daemons required
    - Rule evaluation daemon, Transfer handling daemon
  - All others give extra functionality and can be enabled as required
    - Deletion, Rebalancing, Popularity, Tracing, Messaging, …

- Sites do not run any Rucio services — they only need to operate storage

- ATLAS DDM Central Team operates 320+PB on 120 sites with <2 FTE!
  - Due to all the automations that Rucio daemons provide

Known Rucio limits

- Backend database performance
  - Scaling tests up to LHC Run-3 expectations showed no problems on CERN Oracle instance
  - Want to do more scaling tests with MariaDB and PostgreSQL
- Single-node limit for rule evaluation
  - 8 GB of RAM can serve a single rule with max 500'000 files
  - This limitation is currently being addressed
- Automated deployment of nodes due to load
  - Datacenter issue
  - Currently requires operator to bring up new nodes
  - Want to automate this based on internal system performance metrics

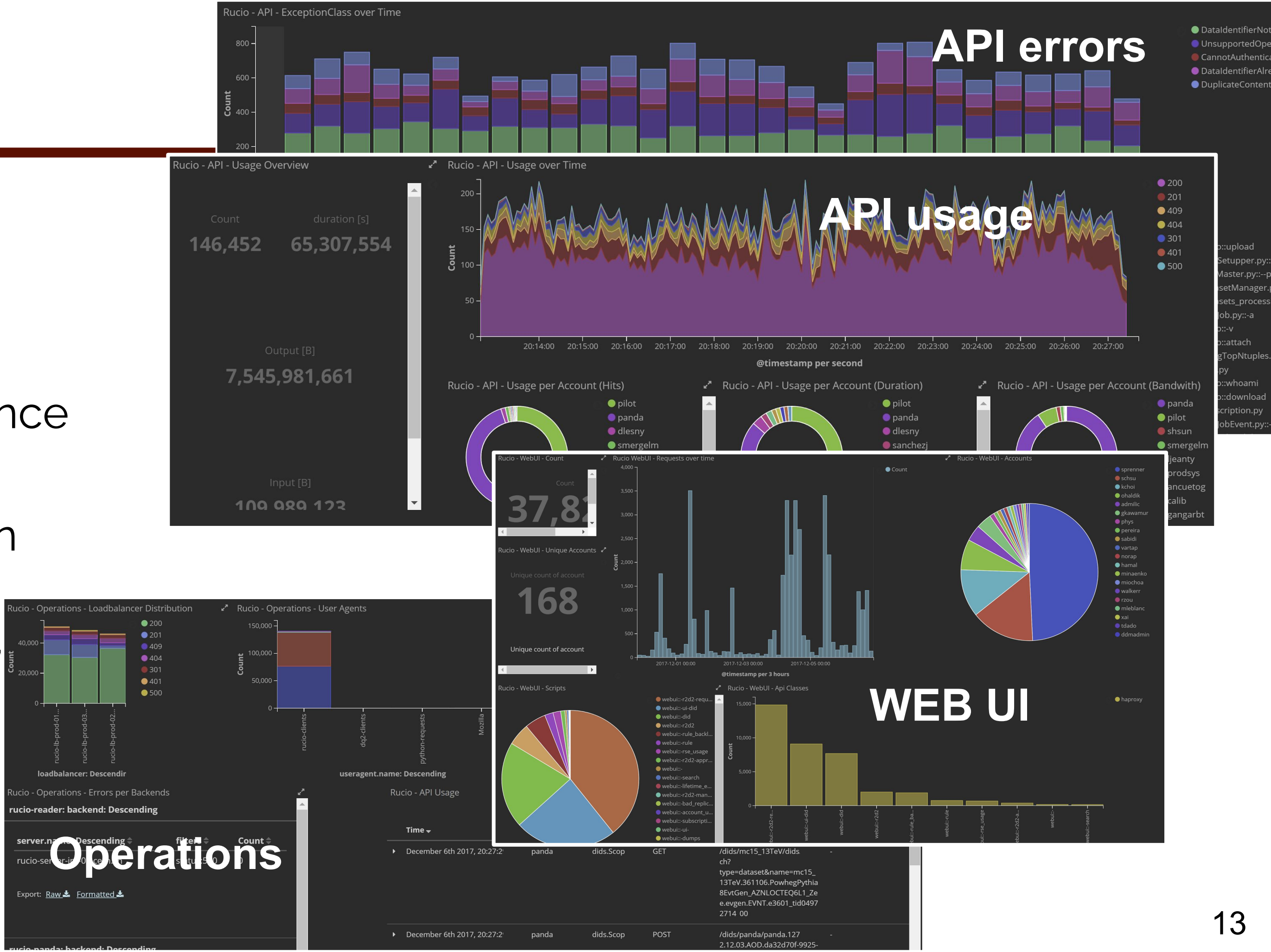
Rucio dependences

- Python 2.7
  - Major parts already Python3 compatible
- Multiple database support
  - Object-relational mapper
  - SQLite, MySQL/MariaDB, PostgreSQL, Oracle
- File Transfer service
  - FTS3

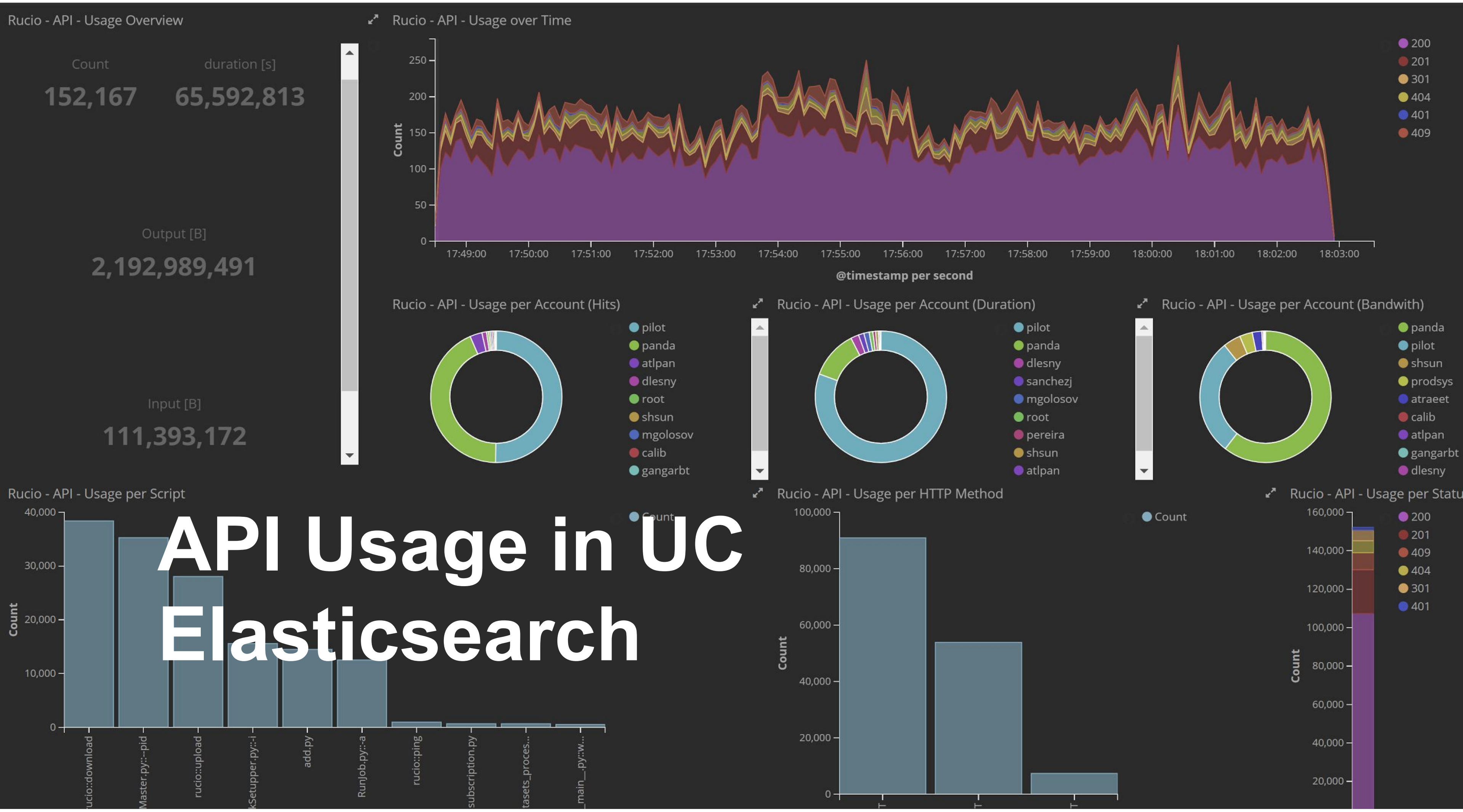
Monitoring

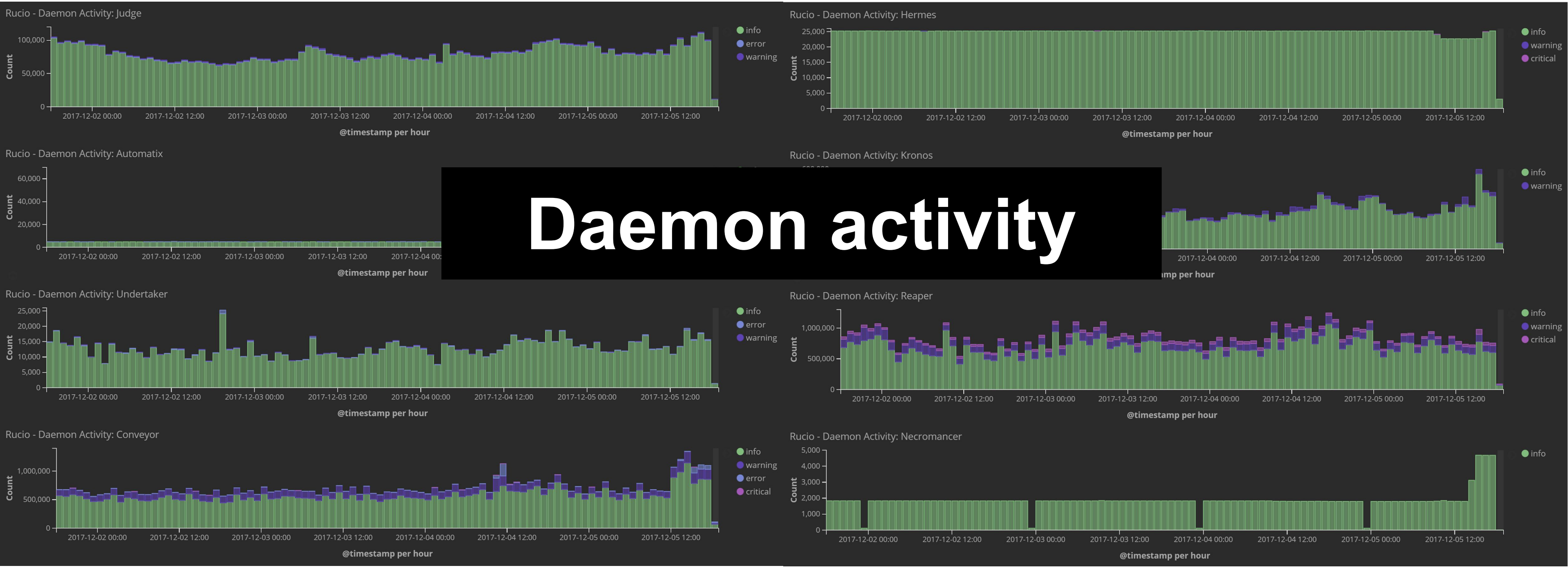
Rucio

- All the DDM data dumped to HDFS once a day.
- All the traces kept in Hadoop and ES
- Internal monitoring Grafana



API Usage in UC Elasticsearch





Daemon activity

- Judge - replication rule engine
- Automatix - generates fake data and upload it on a RSE
- Conveyer - handles requests for data transfers
- Undertaker - obsoleting data identifiers with expired lifetime
- Hermes - delivers messages to an asynchronous broker
- Kronos - consumes tracer messages and updates replica last access time accordingly
- Reaper - deletion of the expired data replicas
- Necromancer - tries to repair erroneous rules, by selecting different replica destinations
- Transmogrifier - is responsible to apply subscriptions and to generate replication rules

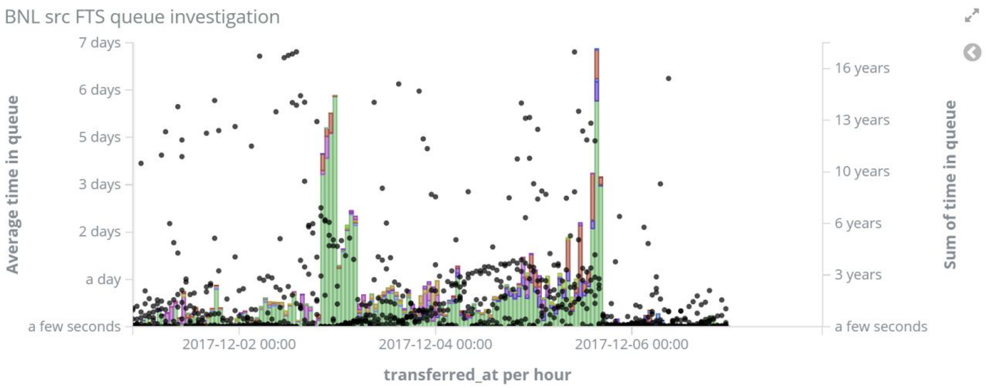
Understanding and optimizing FTS usage

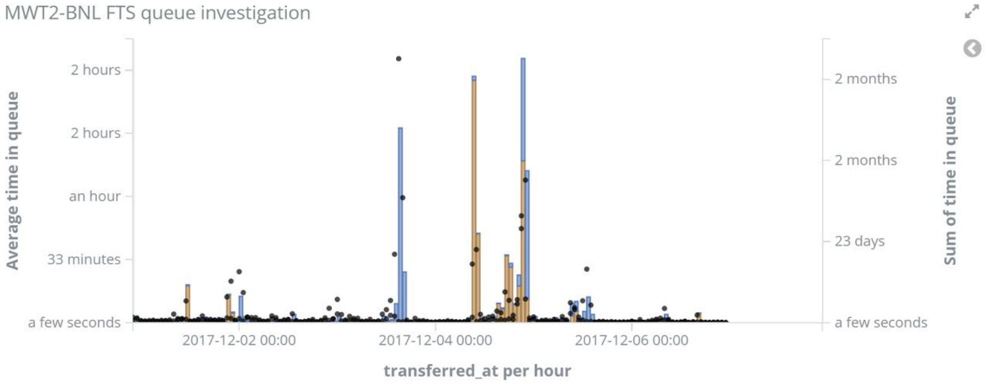
Requires a lot of different data sources:

- Rucio (detailed log on transactions)
- FTS (optimizer settings, reasons behind decisions)
- Sites storage load (from summing up all the traffic)
- Network (PerfSONAR)

For the first time we have all the information and can do detailed analysis, even simulations of how system would behave with different settings.

We found a lot of space for improvement.





ATLAS Statistics

- ~1 billion active files
- ~2 billion archived files
- ~15M datasets/containers
- 840 storage endpoints
- 340 PB storage almost full
- 1.5 PB/day transferred, peaks up to 2.5 PB/day
- 2 PB/day deleted

XENON1T Statistics

- > 1.2M Files
- ~16k Datasets
- 9 storage endpoints
- 1887.5 TB of available storage
- 854.1 TB of available storage used
- Adding 1.3 TB per day, 200+ files per hour
- > 115 GB per hour transferred

AMS Statistics

- ~1M Files
- ~50k Datasets
- 9 storage endpoints
- ~2 PB of available storage
- ~1.5 PB of available storage used

Comparison with similar systems

- PhEDEx
- Globus
  - Can serve as alternative to FTS3 data transport but entirely different set of management principles
- DynaFed, EOS Federation, Xroot Federation
  - Inter-cluster shared filesystem
  - Dynamic discovery of data
  - Can be used as RSEs

Rucio vocabulary

- DID (Data IDentifier)
  - File
  - Dataset
  - Container
- Scope
  - DID namespace partition
- RSE (Rucio Storage Element)
  - Topology description of a storage endpoint
- Rules
  - Declarative mapping of DIDs to RSEs
- Subscription
  - Automatic generation of rules

References

- Code https://github.com/rucio/rucio
- Web https://rucio.cern.ch/
- Docker https://hob.docker.com/r/rucio
- Support https://rucio.slack.com/
- Mail rucio-dev@cern.ch