Moving Rucio to Production in Kubernetes

Thomas Beermann

on behalf of the Rucio team

RUCIO

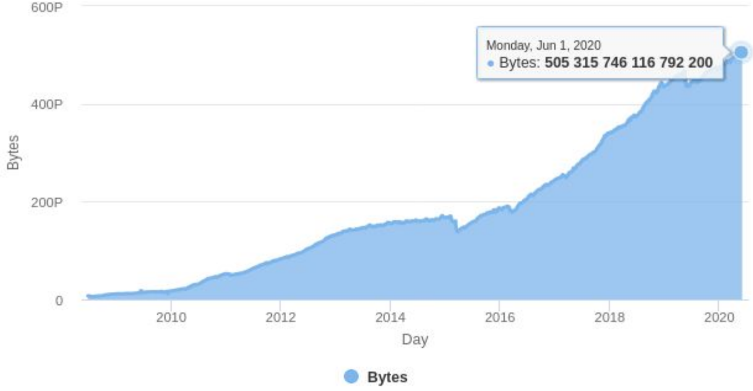
SCIENTIFIC DATA MANAGEMENT

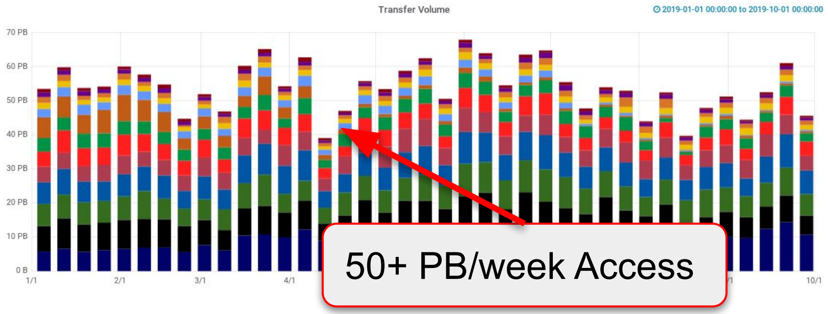
Rucio in a nutshell

- Rucio provides a complete and generic scientific data management service
  - **Seamless integration** of **scientific and commercial** storage and network systems.
  - Data is stored in **global single namespace** and can contain **any potential payload.**
  - Facilities can be **distributed at multiple locations** belonging to **different administrative domains.**
  - Designed with **more than a decade of operational experience** in very large-scale data management.
- Rucio manages location-aware data in a heterogeneous distributed environment
  - Creation, location, transfer, deletion, and annotation of data
  - **Orchestration of dataflows** with both low-level and high-level policies
- Principally developed by and for ATLAS, now with many more communities
- Rucio is open source and available under Apache 2.0 license
- Open community-driven development process

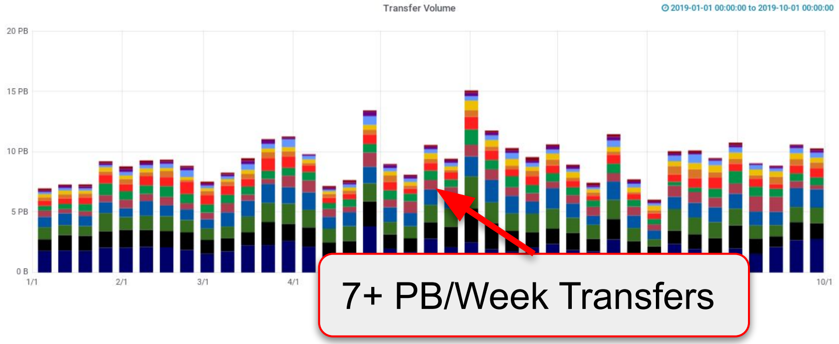
Data management for ATLAS

- A few numbers to set the scale
  - ○ 1B+ files, 505 PB of data, 400+ Hz interaction rate
  - ○ 120 data centres, 5 HPCs, 600 storage areas
  - ○ 500 Petabytes/year transferred & deleted
  - ○ 2.5 Exabytes/year uploaded & downloaded
- Increase 1+ order of magnitude for LHC Run 4





50+ PB/week Access



7+ PB/Week Transfers

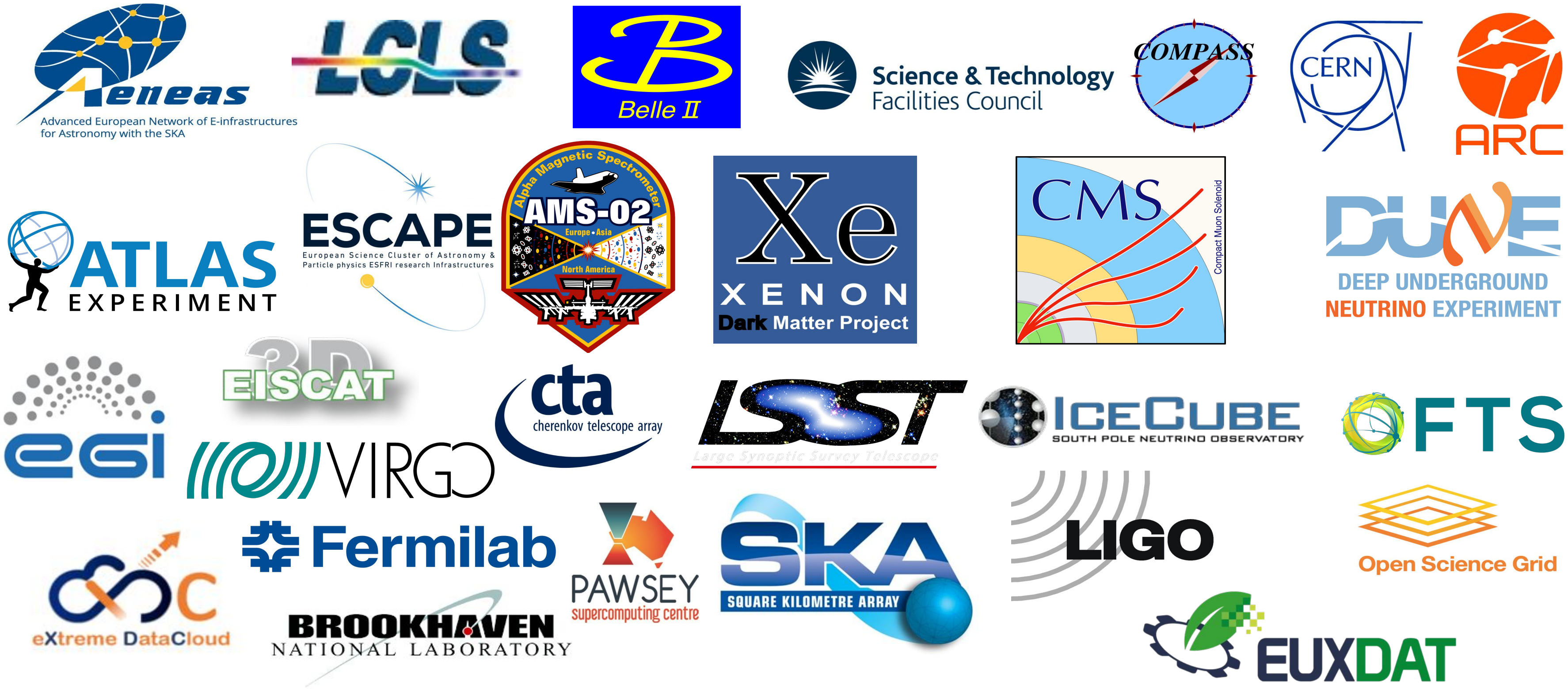
Rucio main functionalities

- Provides many features that can be enabled selectively
  - Horizontally scalable catalog for files, collections, and metadata
  - Transfers between facilities including disk, tapes, clouds, HPCs
  - Authentication and authorisation for users and groups
  - Web-UI, CLI, FUSE, and REST API
  - Extensive monitoring for all dataflows
  - Expressive policy engines with rules, subscriptions, and quotas
  - Automated corruption identification and recovery
  - Transparent support for caches and CDN dataflows
  - Data-analytics based flow control and SDNs
  - …

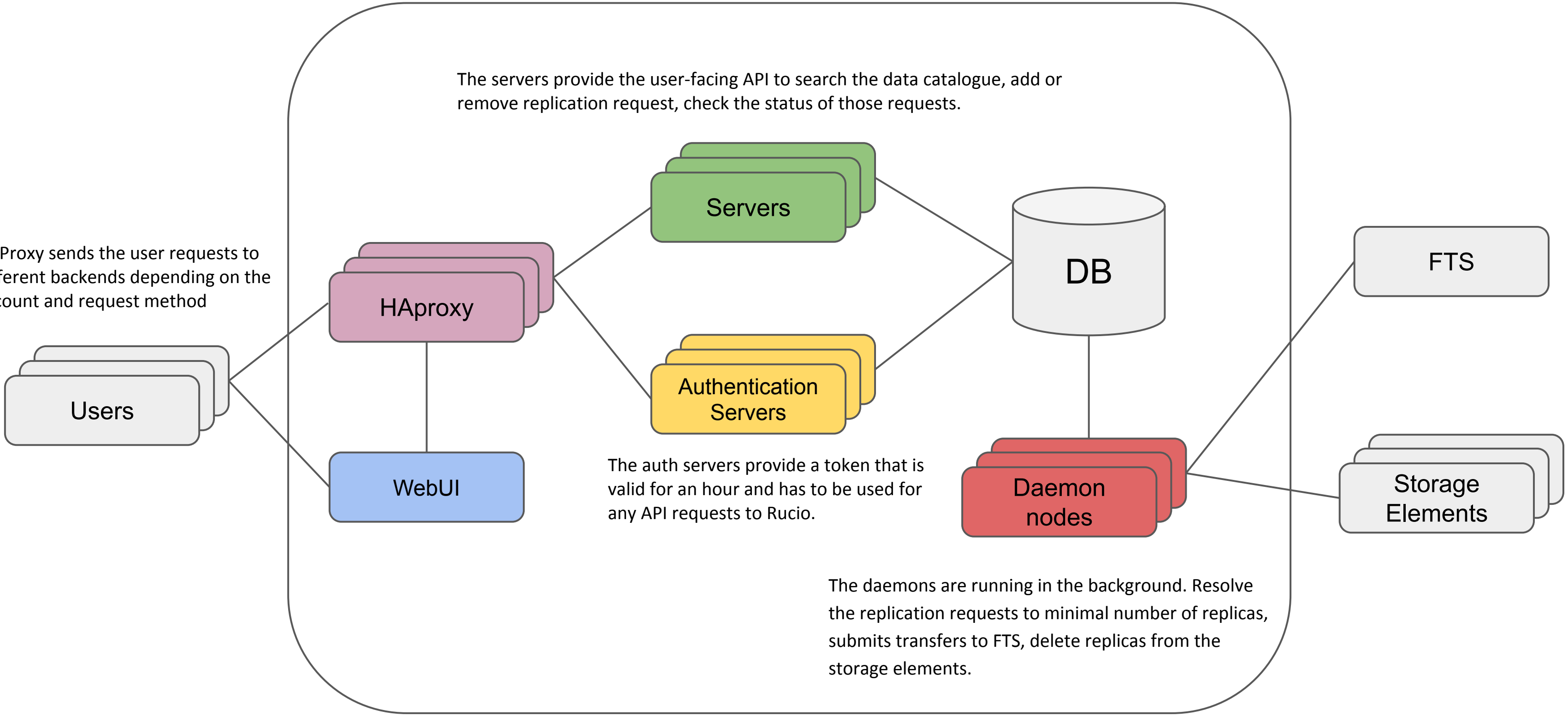
More advanced features

- Rucio is not a distributed file system, it connects existing storage infrastructure
  - No Rucio software needs to run at the data centres
  - Entities are free to choose what suits them best, even within a single community

Community


Aeneas
Advanced European Network of E-infrastructures for Astronomy with the SKA
LCLS
Belle II
Science & Technology Facilities Council
COMPASS
CERN
ARC
ATLAS EXPERIMENT
ESCAPE
European Science Cluster of Astronomy & Particle physics ESFRI research Infrastructures
Alpha Magnetic Spectrometer
AMS-02
Europe • Asia
North America
Xe
XENON
Dark Matter Project
CMS
Compact Muon Solenoid
DUNE
DEEP UNDERGROUND NEUTRINO EXPERIMENT
egi
3D EISCAT
cta
cherenkov telescope array
VIRGO
LSST
Large Synoptic Survey Telescope
ICECUBE
SOUTH POLE NEUTRINO OBSERVATORY
FTS
eXtreme DataCloud
Fermilab
BROOKHAVEN NATIONAL LABORATORY
PAWSEY supercomputing centre
SKA
SQUARE KILOMETRE ARRAY
LIGO
Open Science Grid
EUXDAT

Architecture



The servers provide the user-facing API to search the data catalogue, add or remove replication request, check the status of those requests.

HAProxy sends the user requests to different backends depending on the account and request method

The auth servers provide a token that is valid for an hour and has to be used for any API requests to Rucio.

The daemons are running in the background. Resolve the replication requests to minimal number of replicas, submits transfers to FTS, delete replicas from the storage elements.

Current deployment for ATLAS

- The current deployment for ATLAS uses **separate VMs** deployed on the CERN-IT provided Openstack infrastructure.
- The server and daemon services are **split by integration and production**. New Rucio releases are **tested for one week** on the integration nodes, which get only a small load of the production nodes. Currently we have:
    - 15 / 2 production / integration server VMs.
    - 25 / 7 production / integration daemon VMs.
    - 3 haproxy load balancers.
    - 2 / 1 production / integration webui servers + a couple of VMs for misc services, e.g., running nagios probes, submit hadoop jobs and retrieve output, logging, etc.
- The deployment is **fully managed by Puppet**.

Issues with the current model

- Our current deployment is running stable and we have a lot of experience with the current operations model but:
  - Regular **problems** with **Python dependencies** that are overwritten by automatic package upgrade on the VMs breaking our deployment.
  - The **puppet deployment** grew over time and became quite **complicated**.
  - Adapting the deployment to add or remove new daemons to adapt to different workloads requires **manual intervention** and is rather slow.
  - Setup of a new deployment is complicated and needs a **lot of support** for the **initial installation**.
  - The VM resources are **highly underutilized** because of redundancies and the static deployment model with Puppet.
  - Hunting down problems can be tedious sometimes due to the distributed nature of the deployment.
- Could benefit a lot of a more dynamic Kubernetes deployment.

Why Kubernetes for Rucio?

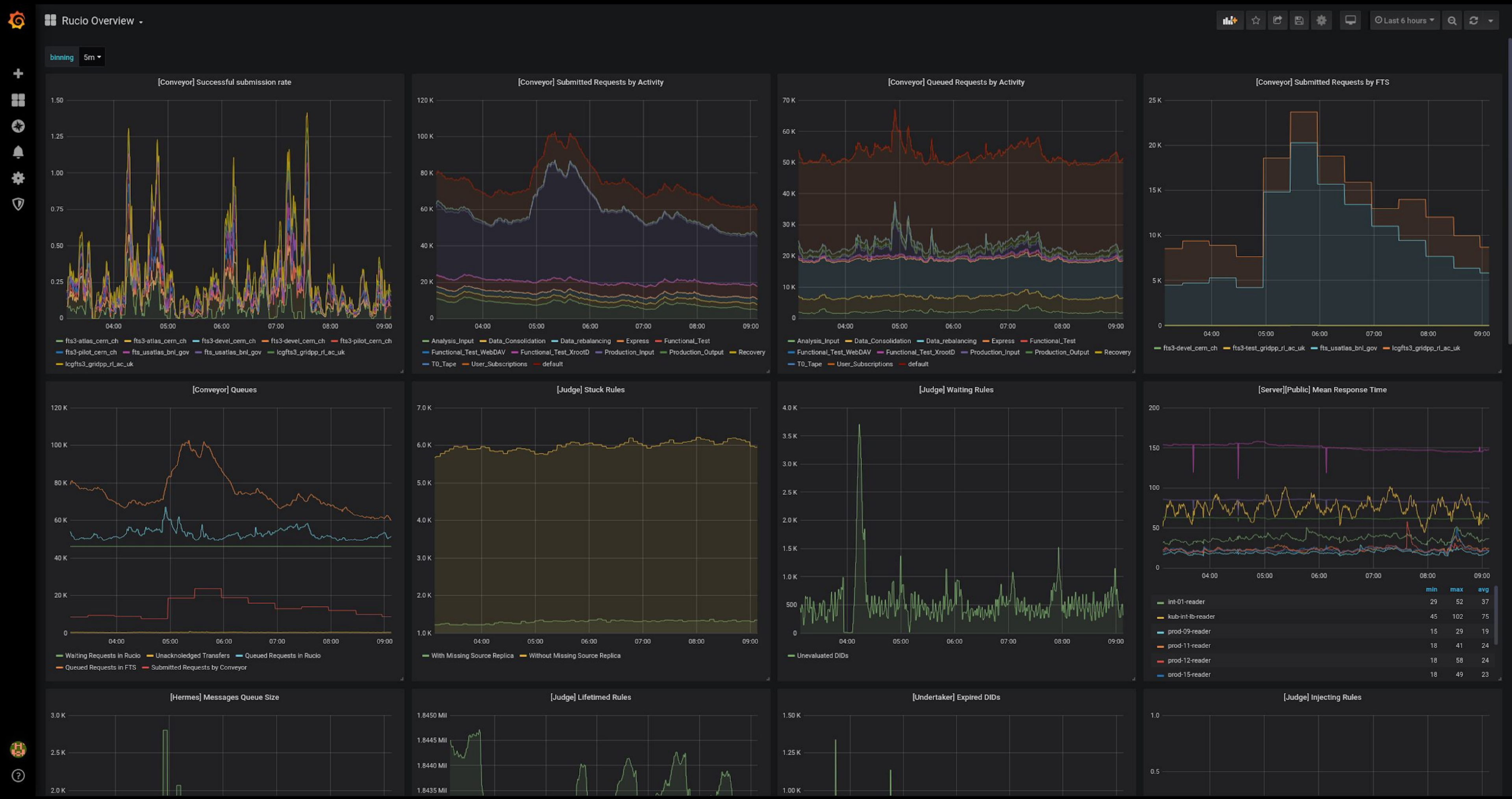
- Containers provide an **isolated and minimal environment** with only the necessary dependencies needed for the application.
- **Initial deployment** of new services becomes really **easy and is quick** thanks to Helm charts.
- **Changes** in the deployment and software upgrades are **quickly propagated** through the system.
- Auto-scaling can help in case of spikes in the workload and to **better utilize** the **available resources** / better energy efficiency.
- Centralized monitoring and logging can make it easier to find problems.

Deployment with Helm and Flux

- The Rucio server, daemon and webui services are fully packaged with Helm.
- Available in our own repo on Github.
- **Set up** of a new Rucio instance is now as **simple** as adapting a few configuration parameters and installing the Helm chart.
- We use Flux to manage our Helm deployments:
  - Since we had the Helm charts already available it is rather **easy to set up**.
  - The Helm values are managed in a gitlab repository.
  - An agent on the cluster regularly checks for updates in the repo and **automatically deploys** them.
  - Changing the deployment is then done by simple git commits, similar to puppet but **much quicker**.
  - Upgrading to a new version or adding new daemons / servers only takes a few minutes.
  - Adds **accountability** which is important for us since there can be multiple people trying to change the deployment.
  - Could bridge the gap for of our ops people not having too much experience with Kubernetes, yet.

Monitoring

- For the cluster monitoring we are relying on the built-in Prometheus server to monitor cluster resources and the our workloads and pod.
- But we also have some **application metrics** for which we are currently using statsd/Graphite in our Puppet deployment.
- We have extended our code to also **support Prometheus**:
  - Can be enabled in our helm-charts.
  - Then every server and daemon pod provides a metrics endpoint that can be scraped by Prometheus.
  - Furthermore, we have probes regularly checking various internal queues in the DB. For that we are running Prometheus Pushgateway and the probes are sending there.
- Added our own Grafana dashboards on top of the cluster-provided ones.



Rucio Overview

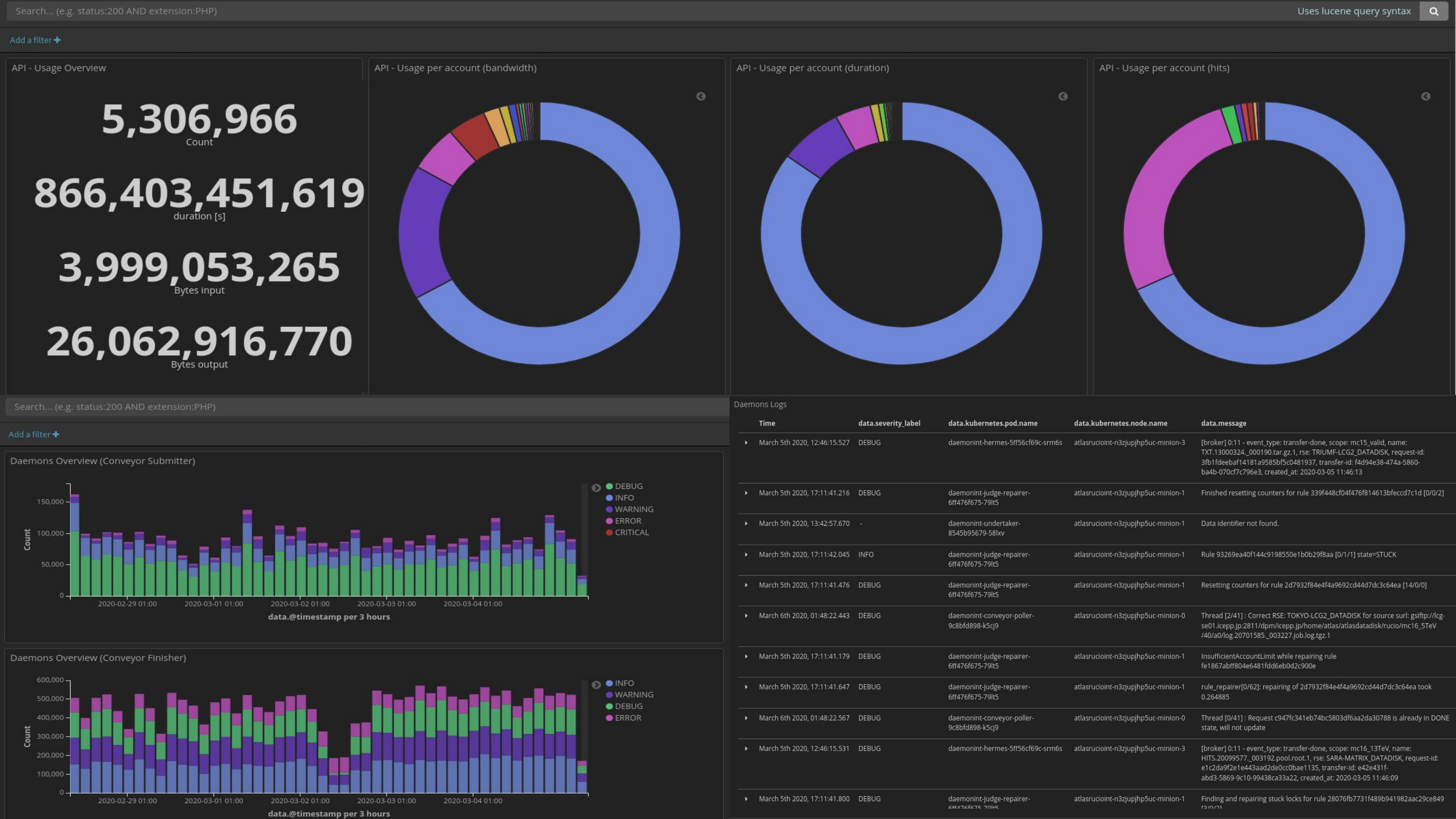
binning 5m

Last 6 hours

	min	max	avg
int-01-reader	29	52	37
kub-int-lb-reader	45	102	75
prod-09-reader	15	29	19
prod-11-reader	18	41	24
prod-12-reader	18	58	24
prod-15-reader	18	49	23

Logging

- For the logging we are using a **private monit-timber** instance.
- Logs are **collected** from the nodes with **filebeat** and send to logstash.
- Logstash **filters** some messages and **parses** the messages into separate fields.
- Logstash running inside the cluster had **problems keeping up with the messages**. Therefore we are currently running it on a **separate VM**.
- We are using it for different purposes and have some custom dashboards in Kibana:
    - Server API monitoring: showing detailed information about the API usage including hits per endpoint, per account, error codes, etc.
    - Daemon activity monitoring: showing an overview of log messages sent from the different daemons to spot potential problems.



API - Usage Overview

5,306,966
Count

866,403,451,619
duration [s]

3,999,053,265
Bytes input

26,062,916,770
Bytes output

API - Usage per account (bandwidth)

API - Usage per account (duration)

API - Usage per account (hits)

Daemons Overview (Conveyor Submitter)

Daemons Overview (Conveyor Finisher)

Daemons Logs

Time	data.severity_label	data.kubernetes.pod.name	data.kubernetes.node.name	data.message
March 5th 2020, 12:46:15.527	DEBUG	daemonint-hermes-5ff56cf69c-srm6s	atlasrucioint-n3zjupjhp5uc-minion-3	[broker] 0:11 - event_type: transfer-done, scope: mc15_valid, name: TXT.13000324._000190.tar.gz.1, rse: TRIUMF-LCG2_DATADISK, request-id: 3fb1fdeebaf14181a9585bf5c0481937, transfer-id: f4d94e38-474a-5860-ba4b-070cf7c796e3, created_at: 2020-03-05 11:46:13
March 5th 2020, 17:11:41.216	DEBUG	daemonint-judge-repairer-6ff476f675-79lt5	atlasrucioint-n3zjupjhp5uc-minion-1	Finished resetting counters for rule 339f448cf04f476f814613bfeccd7c1d [0/0/2]
March 5th 2020, 13:42:57.670	-	daemonint-undertaker-8545b95679-58lxv	atlasrucioint-n3zjupjhp5uc-minion-1	Data identifier not found.
March 5th 2020, 17:11:42.045	INFO	daemonint-judge-repairer-6ff476f675-79lt5	atlasrucioint-n3zjupjhp5uc-minion-1	Rule 93269ea40f144c9198550e1b0b29f8aa [0/1/1] state=STUCK
March 5th 2020, 17:11:41.476	DEBUG	daemonint-judge-repairer-6ff476f675-79lt5	atlasrucioint-n3zjupjhp5uc-minion-1	Resetting counters for rule 2d7932f84e4f4a9692cd44d7dc3c64ea [14/0/0]
March 6th 2020, 01:48:22.443	DEBUG	daemonint-conveyor-poller-9c8bfd898-k5cj9	atlasrucioint-n3zjupjhp5uc-minion-0	Thread [2/41] : Correct RSE: TOKYO-LCG2_DATADISK for source surl: gsiftp://lcg-se01.icepp.jp:2811/dpm/icepp.jp/home/atlas/atlasdatadisk/rucio/mc16_5TeV /40/a0/log.20701585._003227.job.log.tgz.1
March 5th 2020, 17:11:41.179	DEBUG	daemonint-judge-repairer-6ff476f675-79lt5	atlasrucioint-n3zjupjhp5uc-minion-1	InsufficientAccountLimit while repairing rule fe1867abff804e6481fdd6eb0d2c900e
March 5th 2020, 17:11:41.647	DEBUG	daemonint-judge-repairer-6ff476f675-79lt5	atlasrucioint-n3zjupjhp5uc-minion-1	rule_repairer[0/62]: repairing of 2d7932f84e4f4a9692cd44d7dc3c64ea took 0.264885
March 6th 2020, 01:48:22.567	DEBUG	daemonint-conveyor-poller-9c8bfd898-k5cj9	atlasrucioint-n3zjupjhp5uc-minion-0	Thread [0/41] : Request c947fc341eb74bc5803df6aa2da30788 is already in DONE state, will not update
March 5th 2020, 12:46:15.531	DEBUG	daemonint-hermes-5ff56cf69c-srm6s	atlasrucioint-n3zjupjhp5uc-minion-3	[broker] 0:11 - event_type: transfer-done, scope: mc16_13TeV, name: HITS.20099577._003192.pool.root.1, rse: SARA-MATRIX_DATADISK, request-id: e1c2da9f2e1e443aad2de0cc0bae1135, transfer-id: e42e431f-abd3-5869-9c10-99438ca33a22, created_at: 2020-03-05 11:46:09
March 5th 2020, 17:11:41.800	DEBUG	daemonint-judge-repairer-6ff476f675-79lt5	atlasrucioint-n3zjupjhp5uc-minion-1	Finding and repairing stuck locks for rule 28076fb7731f489b941982aac29ce849 [3/0/2]

Current K8s deployment for ATLAS (1/2)

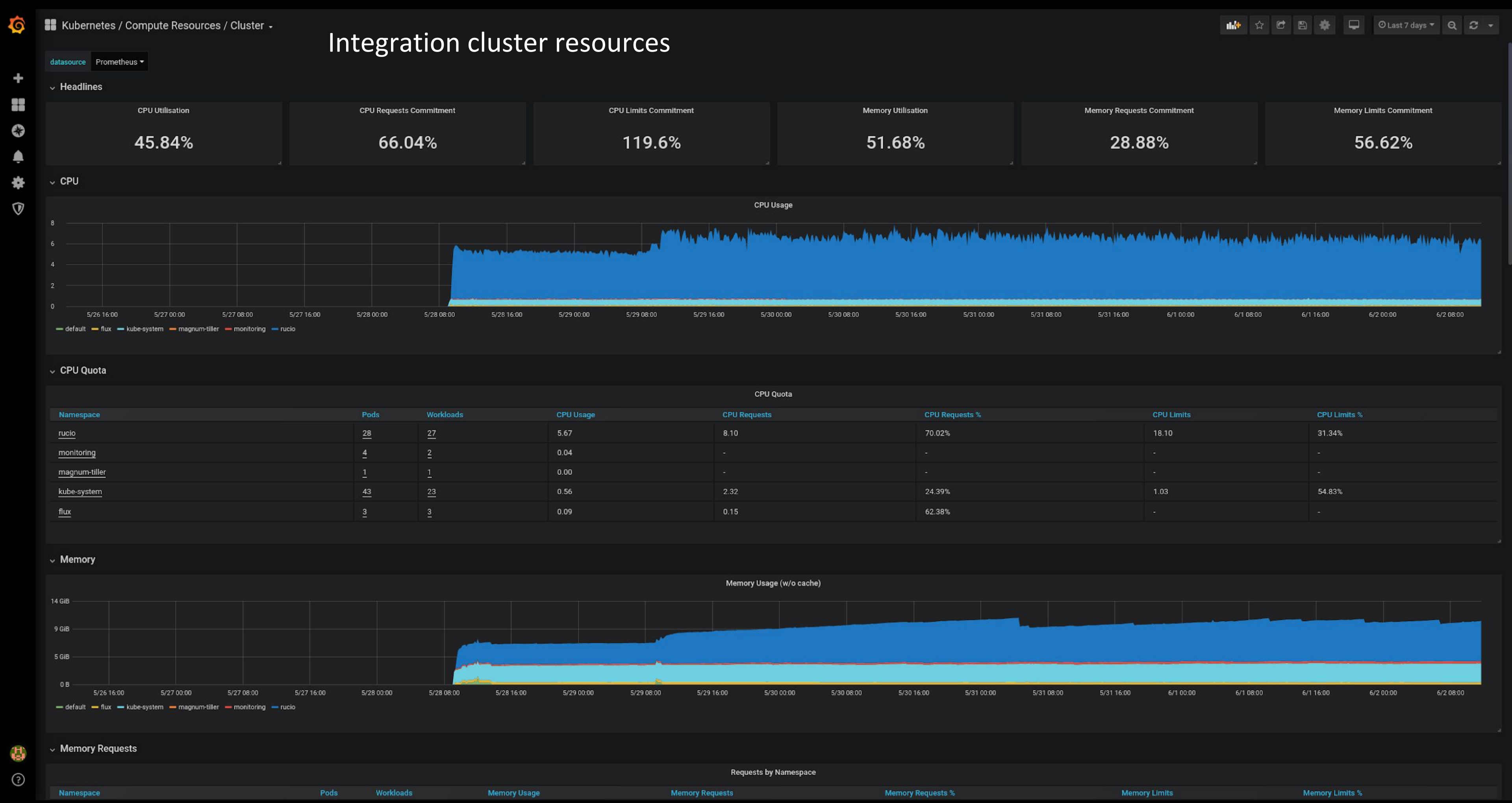
- We are currently running two K8s cluster for our ATLAS deployment:
  - Integration cluster with 3 nodes running 1.18
  - Production cluster with 4 nodes running 1.15
- On the **integration** cluster we run **both servers and daemons**.
- On the **production** cluster we run **only daemons**.

- For the servers we are using a loadbalancer service with a virtual IP.
- For the moment we will **keep using our own HAProxy** which allows us to gradually move over.
- The virtual IP has been added to our HAProxy as a backend receiving **~5% of the total load.**

Current K8s deployment for ATLAS (2/2)

- The daemons are using a **heartbeat** mechanism to **automatically share** the workload across **multiple instance**.
- So for the K8s deployment we could just add daemons. **No need to change** anything in our **Puppet deployment**, yet.
- We are running three different releases:
  - Integration release with one pod per daemon and 1-10 threads.
  - Python3 integration release with the same configuration. It is used to validate our current migration efforts to py3.
  - Production release with 1-2 pods and 5-60 threads.
- With this configuration we are already running between **30-50 percent of our total load on K8s.**

Integration cluster resources



Headlines

CPU Utilisation	CPU Requests Commitment	CPU Limits Commitment	Memory Utilisation	Memory Requests Commitment	Memory Limits Commitment
45.84%	66.04%	119.6%	51.68%	28.88%	56.62%

CPU

CPU Usage

CPU Quota

CPU Quota

Namespace	Pods	Workloads	CPU Usage	CPU Requests	CPU Requests %	CPU Limits	CPU Limits %
rucio	28	27	5.67	8.10	70.02%	18.10	31.34%
monitoring	4	2	0.04	-	-	-	-
magnum-tiller	1	1	0.00	-	-	-	-
kube-system	43	23	0.56	2.32	24.39%	1.03	54.83%
flux	3	3	0.09	0.15	62.38%	-	-

Memory

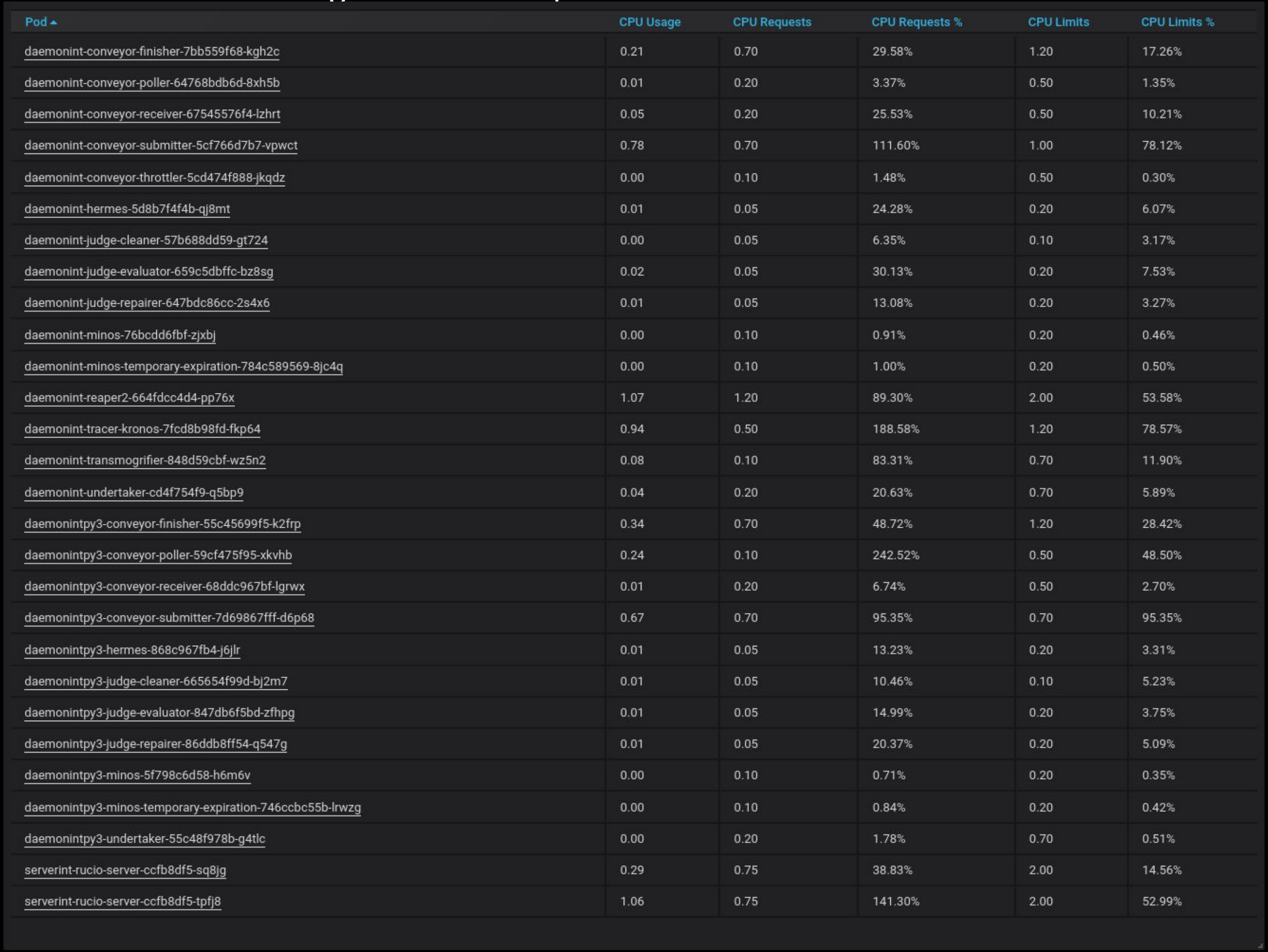
Memory Usage (w/o cache)

Memory Requests

Requests by Namespace

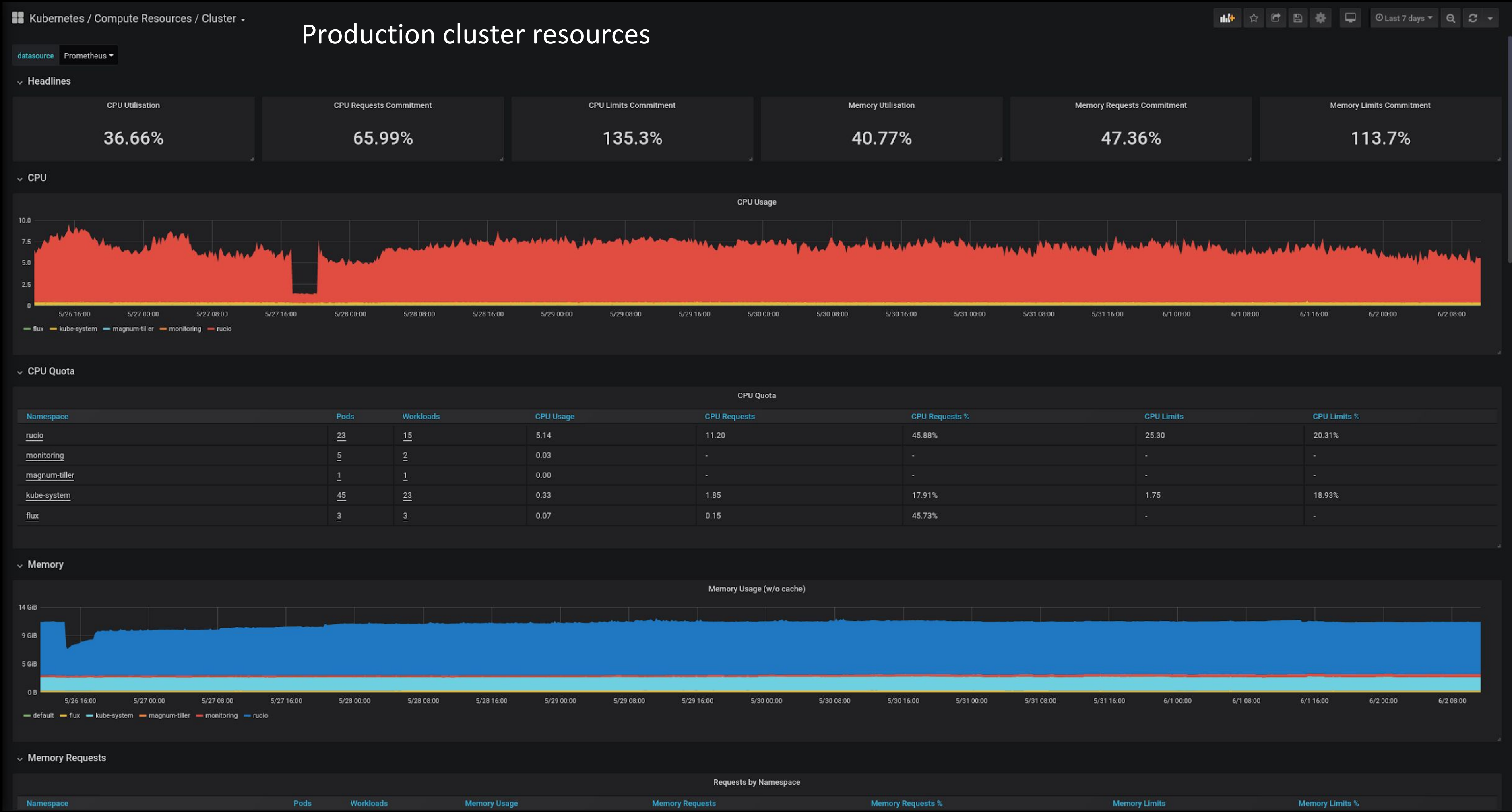
Namespace	Pods	Workloads	Memory Usage	Memory Requests	Memory Requests %	Memory Limits	Memory Limits %

Integration cluster pods



Pod ▲	CPU Usage	CPU Requests	CPU Requests %	CPU Limits	CPU Limits %
daemonint-conveyor-finisher-7bb559f68-kgh2c	0.21	0.70	29.58%	1.20	17.26%
daemonint-conveyor-poller-64768bdb6d-8xh5b	0.01	0.20	3.37%	0.50	1.35%
daemonint-conveyor-receiver-67545576f4-lzhrt	0.05	0.20	25.53%	0.50	10.21%
daemonint-conveyor-submitter-5cf766d7b7-vpwct	0.78	0.70	111.60%	1.00	78.12%
daemonint-conveyor-throttler-5cd474f888-jkqdz	0.00	0.10	1.48%	0.50	0.30%
daemonint-hermes-5d8b7f4f4b-qj8mt	0.01	0.05	24.28%	0.20	6.07%
daemonint-judge-cleaner-57b688dd59-gt724	0.00	0.05	6.35%	0.10	3.17%
daemonint-judge-evaluator-659c5dbffc-bz8sg	0.02	0.05	30.13%	0.20	7.53%
daemonint-judge-repairer-647bdc86cc-2s4x6	0.01	0.05	13.08%	0.20	3.27%
daemonint-minos-76bcdd6fbf-zjxbj	0.00	0.10	0.91%	0.20	0.46%
daemonint-minos-temporary-expiration-784c589569-8jc4q	0.00	0.10	1.00%	0.20	0.50%
daemonint-reaper2-664fdcc4d4-pp76x	1.07	1.20	89.30%	2.00	53.58%
daemonint-tracer-kronos-7fcd8b98fd-fkp64	0.94	0.50	188.58%	1.20	78.57%
daemonint-transmogrifier-848d59cbf-wz5n2	0.08	0.10	83.31%	0.70	11.90%
daemonint-undertaker-cd4f754f9-q5bp9	0.04	0.20	20.63%	0.70	5.89%
daemonintpy3-conveyor-finisher-55c45699f5-k2frp	0.34	0.70	48.72%	1.20	28.42%
daemonintpy3-conveyor-poller-59cf475f95-xkvhb	0.24	0.10	242.52%	0.50	48.50%
daemonintpy3-conveyor-receiver-68ddc967bf-lgrwx	0.01	0.20	6.74%	0.50	2.70%
daemonintpy3-conveyor-submitter-7d69867fff-d6p68	0.67	0.70	95.35%	0.70	95.35%
daemonintpy3-hermes-868c967fb4-j6jlr	0.01	0.05	13.23%	0.20	3.31%
daemonintpy3-judge-cleaner-665654f99d-bj2m7	0.01	0.05	10.46%	0.10	5.23%
daemonintpy3-judge-evaluator-847db6f5bd-zfhpg	0.01	0.05	14.99%	0.20	3.75%
daemonintpy3-judge-repairer-86ddb8ff54-q547g	0.01	0.05	20.37%	0.20	5.09%
daemonintpy3-minos-5f798c6d58-h6m6v	0.00	0.10	0.71%	0.20	0.35%
daemonintpy3-minos-temporary-expiration-746ccbc55b-lrwzg	0.00	0.10	0.84%	0.20	0.42%
daemonintpy3-undertaker-55c48f978b-g4tlc	0.00	0.20	1.78%	0.70	0.51%
serverint-rucio-server-ccfb8df5-sq8jg	0.29	0.75	38.83%	2.00	14.56%
serverint-rucio-server-ccfb8df5-tpfj8	1.06	0.75	141.30%	2.00	52.99%

Production cluster resources



Headlines

CPU Utilisation	CPU Requests Commitment	CPU Limits Commitment	Memory Utilisation	Memory Requests Commitment	Memory Limits Commitment
36.66%	65.99%	135.3%	40.77%	47.36%	113.7%

CPU

CPU Usage

CPU Quota

CPU Quota

Namespace	Pods	Workloads	CPU Usage	CPU Requests	CPU Requests %	CPU Limits	CPU Limits %
rucio	23	15	5.14	11.20	45.88%	25.30	20.31%
monitoring	5	2	0.03	-	-	-	-
magnum-tiller	1	1	0.00	-	-	-	-
kube-system	45	23	0.33	1.85	17.91%	1.75	18.93%
flux	3	3	0.07	0.15	45.73%	-	-

Memory

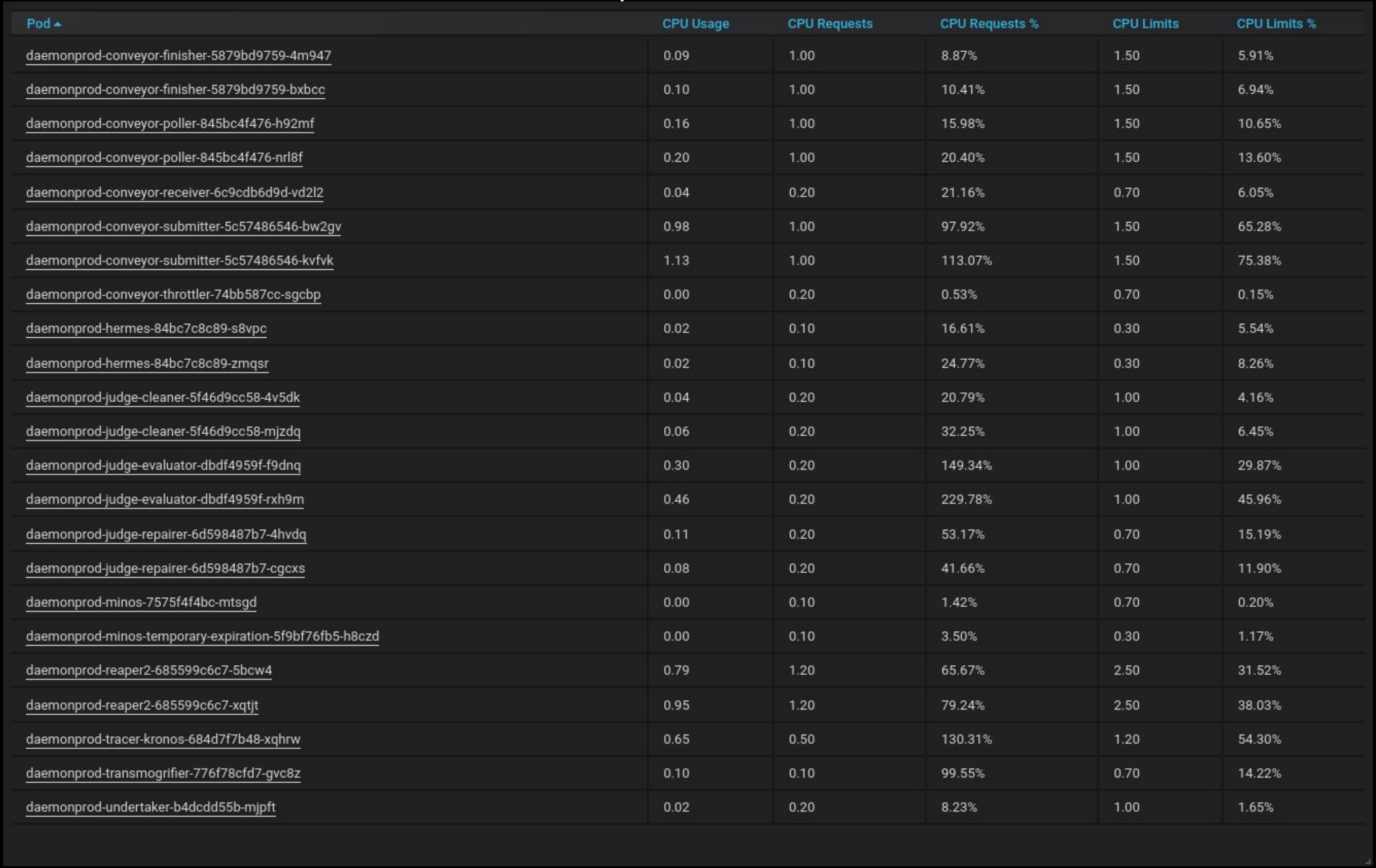
Memory Usage (w/o cache)

Memory Requests

Requests by Namespace

Namespace	Pods	Workloads	Memory Usage	Memory Requests	Memory Requests %	Memory Limits	Memory Limits %

Production cluster pods



Pod ▲	CPU Usage	CPU Requests	CPU Requests %	CPU Limits	CPU Limits %
daemonprod-conveyor-finisher-5879bd9759-4m947	0.09	1.00	8.87%	1.50	5.91%
daemonprod-conveyor-finisher-5879bd9759-bxbcc	0.10	1.00	10.41%	1.50	6.94%
daemonprod-conveyor-poller-845bc4f476-h92mf	0.16	1.00	15.98%	1.50	10.65%
daemonprod-conveyor-poller-845bc4f476-nrl8f	0.20	1.00	20.40%	1.50	13.60%
daemonprod-conveyor-receiver-6c9cdb6d9d-vd2l2	0.04	0.20	21.16%	0.70	6.05%
daemonprod-conveyor-submitter-5c57486546-bw2gv	0.98	1.00	97.92%	1.50	65.28%
daemonprod-conveyor-submitter-5c57486546-kvfvk	1.13	1.00	113.07%	1.50	75.38%
daemonprod-conveyor-throttler-74bb587cc-sgcbp	0.00	0.20	0.53%	0.70	0.15%
daemonprod-hermes-84bc7c8c89-s8vpc	0.02	0.10	16.61%	0.30	5.54%
daemonprod-hermes-84bc7c8c89-zmqsr	0.02	0.10	24.77%	0.30	8.26%
daemonprod-judge-cleaner-5f46d9cc58-4v5dk	0.04	0.20	20.79%	1.00	4.16%
daemonprod-judge-cleaner-5f46d9cc58-mjzdq	0.06	0.20	32.25%	1.00	6.45%
daemonprod-judge-evaluator-dbdf4959f-f9dnq	0.30	0.20	149.34%	1.00	29.87%
daemonprod-judge-evaluator-dbdf4959f-rxh9m	0.46	0.20	229.78%	1.00	45.96%
daemonprod-judge-repairer-6d598487b7-4hvdq	0.11	0.20	53.17%	0.70	15.19%
daemonprod-judge-repairer-6d598487b7-cgcxs	0.08	0.20	41.66%	0.70	11.90%
daemonprod-minos-7575f4f4bc-mtsgd	0.00	0.10	1.42%	0.70	0.20%
daemonprod-minos-temporary-expiration-5f9bf76fb5-h8czd	0.00	0.10	3.50%	0.30	1.17%
daemonprod-reaper2-685599c6c7-5bcw4	0.79	1.20	65.67%	2.50	31.52%
daemonprod-reaper2-685599c6c7-xqtjt	0.95	1.20	79.24%	2.50	38.03%
daemonprod-tracer-kronos-684d7f7b48-xqhrw	0.65	0.50	130.31%	1.20	54.30%
daemonprod-transmogrifier-776f78cfd7-gvc8z	0.10	0.10	99.55%	0.70	14.22%
daemonprod-undertaker-b4dcdd55b-mjpft	0.02	0.20	8.23%	1.00	1.65%

Auto-scaling

- Some of our workloads can have a **spiky behaviour** and sometimes **need manually intervention** by adding new daemons:
    - Many transfers created at the same time, e.g., for rebalancing, can create a transfer backlog:
        - First, start more submitters, then pollers, then finishers.
    - Deletion campaigns to remove used data can create a deletion backlog:
        - Start more reapers.
- Could be a **good fit for auto-scaling**.
- We have all necessary **metrics available in Prometheus** and therefore also for the auto-scaler.
- Did some successful basic testing but we need to put some more effort to find reasonable thresholds.

Concerns

- We are now at a point where we can easily and quickly deploy new instances of Rucio but getting there took some time:
  - K8s has a **steep learning curve** with lots of new terms, concepts and tools.
  - Writing the **Helm charts needed some effort** and we are still constantly updating them.
  - **Changing configurations** in the deployment is easy and does **not really need any knowledge of K8s** at all thanks to flux.
  - But if something breaks it can be a bit **more difficult to fix**, at least if you are used to VMs.
  - Still have to **gain more experience** and develop strategies in case of failures.
- **Most of the issues** we faced so far on the infrastructure were **quickly addressed** by CERN IT.
- Only bigger issue for the moment is the **lower network performance**, resulting in **higher server response time**.

Deployments for other experiments / activities

- CMS:
  - The CMS experiment decided to directly use K8s for their Rucio deployment.
  - Also using the CERN Openstack infrastructure.
  - We are working closely together on common Helm charts and Kubernetes setups.
- DOMA TPC / XDC:
  - We are running a small cluster for webdav/xrootd third-party-copy transfer tests.
  - Also used for XDC/OIDC token authentication testing.
  - No HAProxy, instead using an nginx ingress needed for X509 certificate passthrough.
  - One of our longest running cluster. Helpful to gain experience.
- Folding@Home:
  - F@H expressed interest in using Rucio for their data management.
  - We set up a small demo at CERN that will be used to evaluate Rucio.
  - Setting up a new instance like this becomes really easy and quick with the Helm charts and flux.

Where to go from here?

- We are running integration on K8s for a **long time** now **without bigger issues**.
- We reached a point where are already running a **considerable load** of our deployment on K8s (at least for the daemons).
- For the moment we only added to our existing Puppet deployment and exhausted our Openstack quota.
- We will start to **remove/reshuffle** some **services in Puppet** freeing up resources that can be added to K8s.
- Next step would be to **significantly increasing the server** capacity on K8s.
- We will continue to increase our load on K8s week by week.
- When everything goes as planned we want to be completely **migrated by Q3/2020**.

Questions?

More information

Website		http://rucio.cern.ch
Documentation		https://rucio.readthedocs.io
Repository		https://github.com/rucio/
Images		https://hub.docker.com/r/rucio/
Online support		https://rucio.slack.com/messages/#support/
Developer contact		rucio-dev@cern.ch
Publications		https://rucio.cern.ch/publications.html
Twitter		https://twitter.com/RucioData

Rucio – Scientific data management

Martin Barisits · Thomas Beermann · Frank Berghaus · Brian Bockelman · Joaquin Bogado · David Cameron · Dimitrios Christidis · Diego Ciangottini · Gancho Dimitrov · Markus Elsing · Vincent Garonne · Alessandro di Girolamo · Luc Goossens · Wen Guan · Jaroslav Guenther · Tomas Javurek · Dietmar Kuhn · Mario Lassnig · Fernando Lopez · Nicolo Magini · Angelos Molfetas · Armin Nairz · Farid Ould-Saada · Stefan Prenner · Cedric Serfon · Graeme Stewart · Eric Vaandering · Petya Vasileva · Ralph Vigne · Tobias Wegner

Received: 1970-01-01 / Accepted: 1970-01-01

Abstract Rucio is an open source software framework that provides scientific collaborations with the functionality to organize, manage, and access their volumes of data. The data can be distributed across heterogeneous data centers at widely distributed locations. Rucio has been originally developed to meet the requirements of the high-energy physics experiment ATLAS, and is continuously extended to support the LHC experiments and other diverse scientific communities. In this article we detail the fundamental concepts of Rucio, describe

is stored, analyzed by rese
long-term storage. The AT
the Large Hadron Collider
has such conditions. To fu
ment, the data manageme
developed to allow ATLAS
of data in an efficient and
handling systems focused o
ring files between data cent
contents across the networ