AARHUS UNIVERSITET

# Microservices and DevOps

## Scalable Microservices

Humio

Henrik Bærbak Christensen

AARHUS UNIVERSITET

# Intro

- Humio = Lots of manual machinery!
- When you find it taxing – remember…
- ***This is nothing compared to an ELK stack ☺***
  - I spend vast number of hours on ELK and it was tedious and required a lot of CPU power ☹

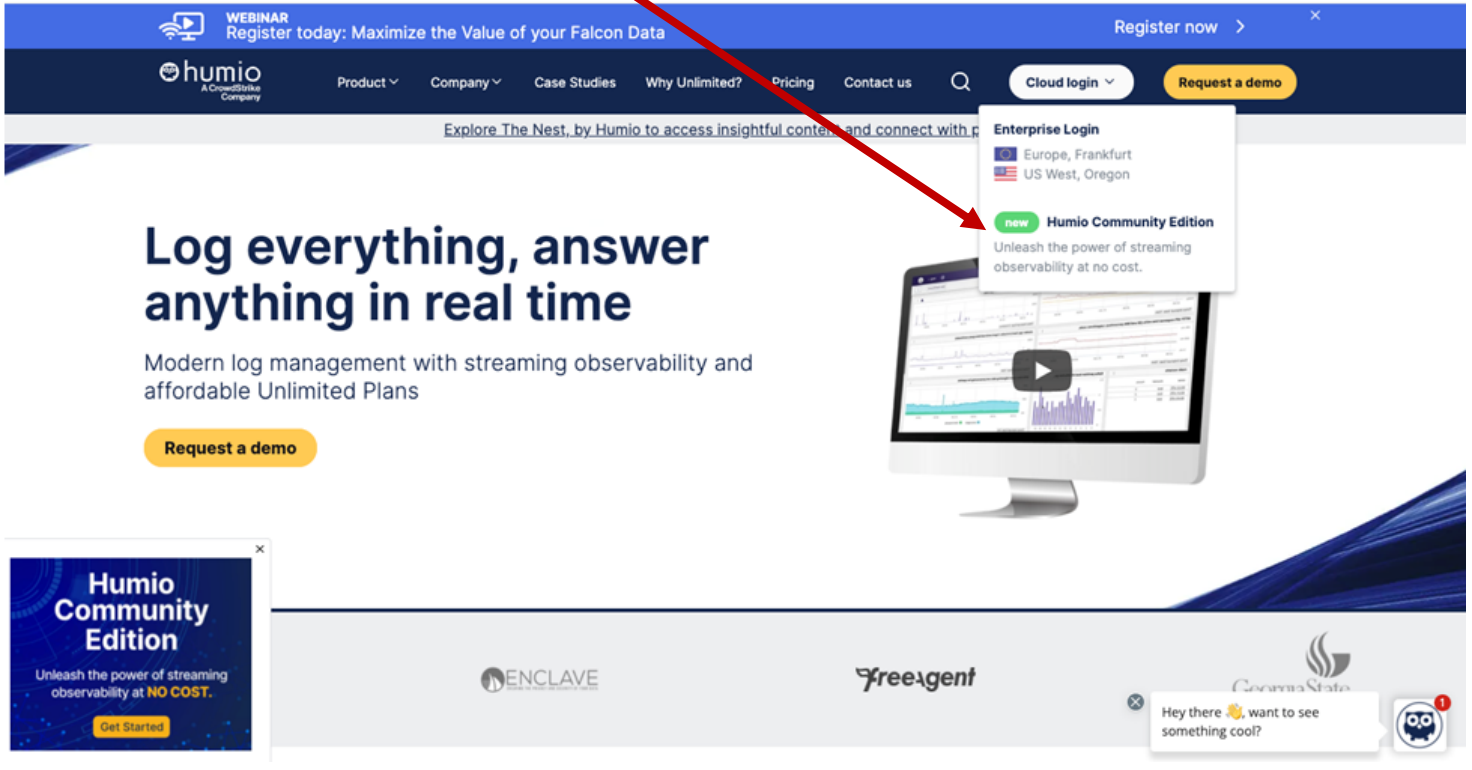
AARHUS UNIVERSITET

# Setup

We go for their Community Edition Cloud Service…

AARHUS UNIVERSITET

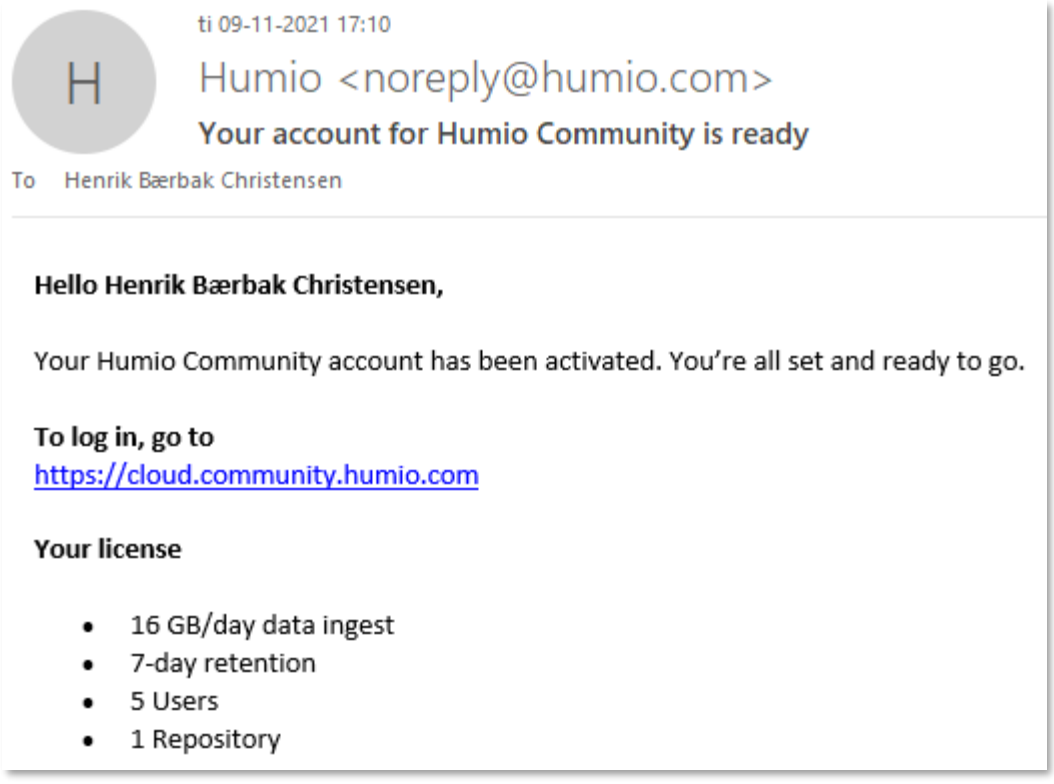
# Just Click…



AARHUS UNIVERSITET

# You will need to register…

- And some time (1-2 days – or just a few hours) later, you get access…



ti 09-11-2021 17:10

H

Humio <noreply@humio.com>

**Your account for Humio Community is ready**

To Henrik Bærbak Christensen

**Hello Henrik Bærbak Christensen,**

Your Humio Community account has been activated. You're all set and ready to go.

**To log in, go to**
https://cloud.community.humio.com

**Your license**

- 16 GB/day data ingest
- 7-day retention
- 5 Users
- 1 Repository

AARHUS UNIVERSITET

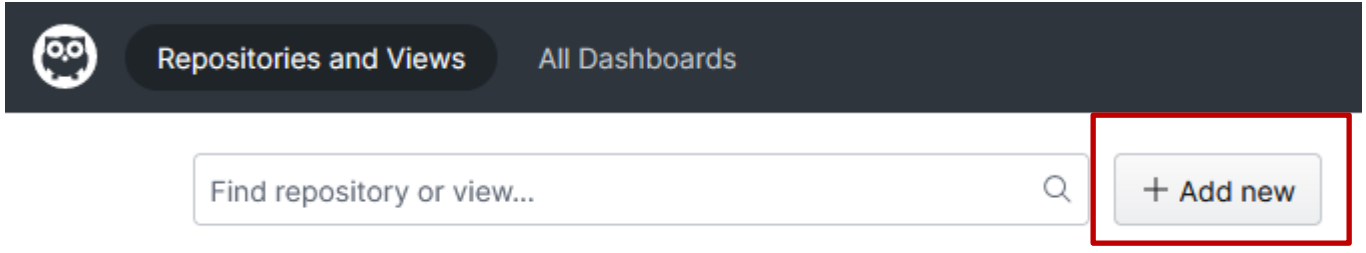
# Next Steps – In Overview

- You need to
  - Define a repository
    - to hold your logging data
  - Define an ingest token
    - That identifies where your log stream's sink is
      - That is – the repository
  - Define a parser
    - So Humio can understand the format of your log messages
  - Make your container(s) ship logs to your repository

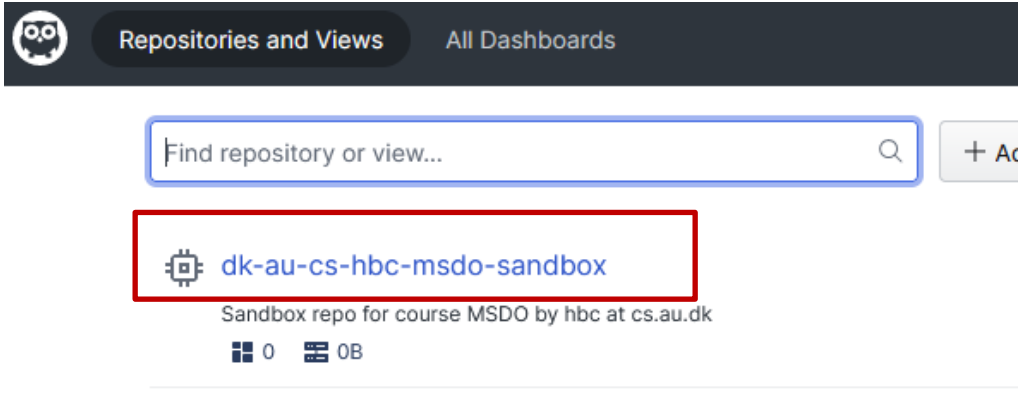
AARHUS UNIVERSITET

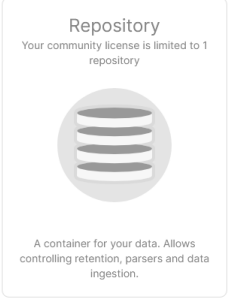
# Define Repo

- Add repo




- Hit ‘Repository’ and define one with a **unique** name

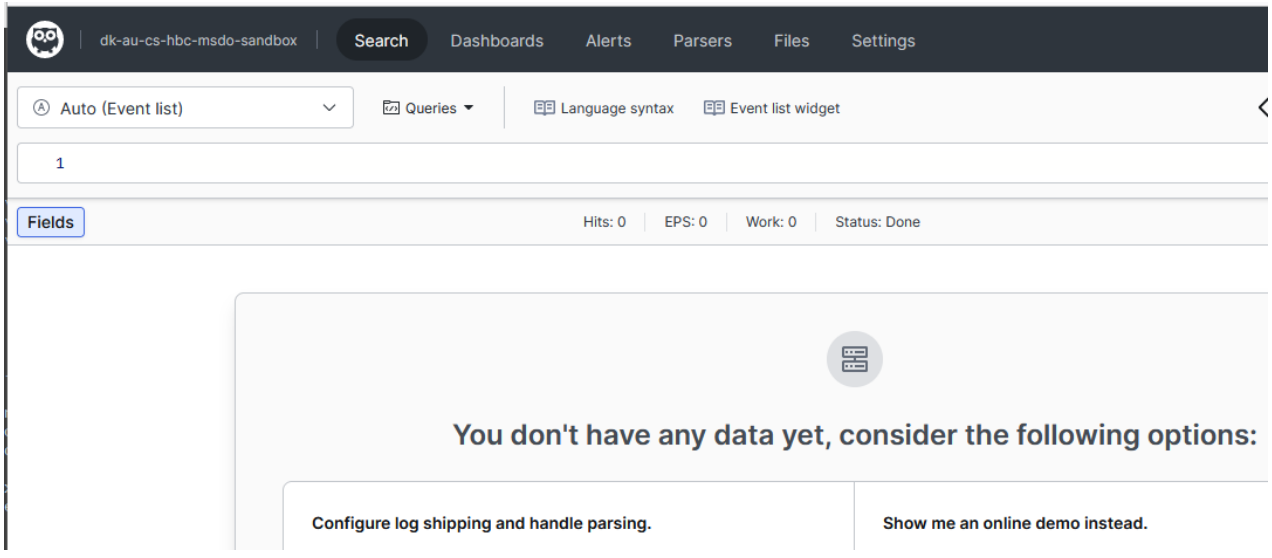


AARHUS UNIVERSITET

# Define Token

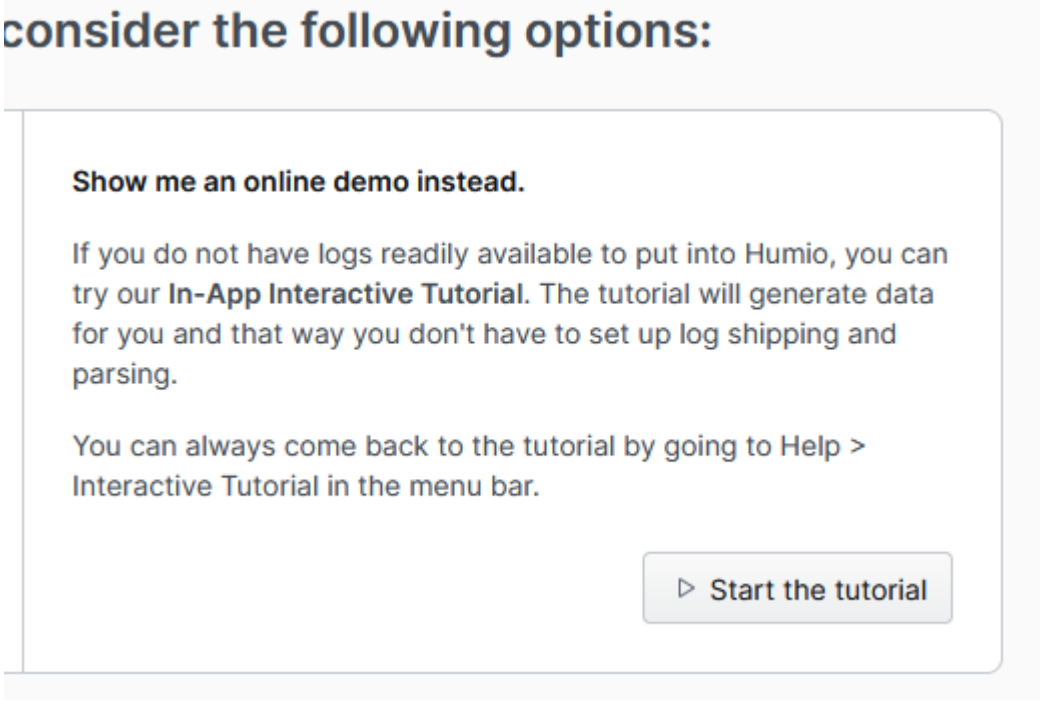
- Hit your defined repo to find the *dashboard page*



AARHUS UNIVERSITET

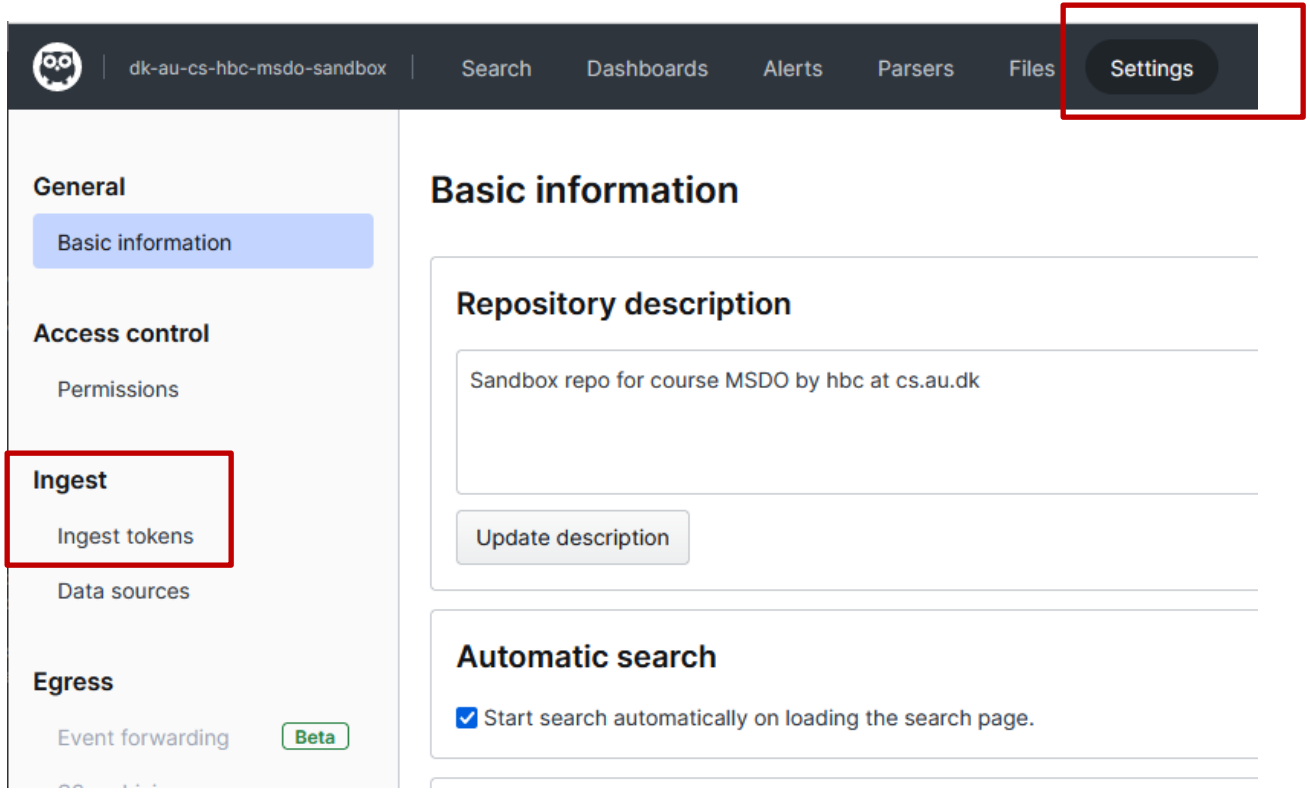
# Do the Tutorial

- Advice to browse over the tutorial
  - One of the super great assets of Humio – is the built-in learning experience !



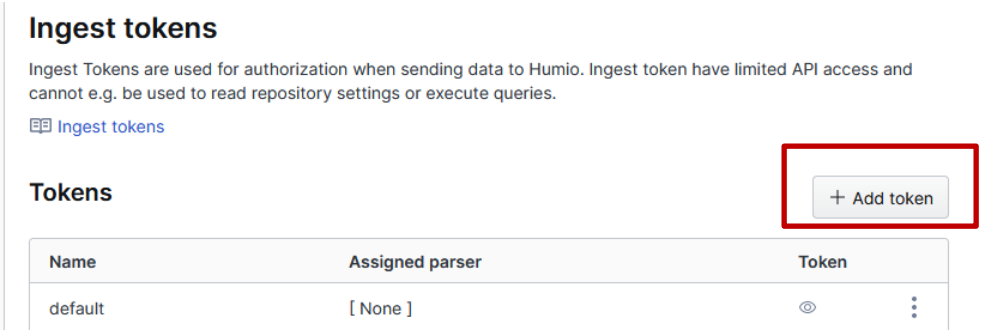
AARHUS UNIVERSITET

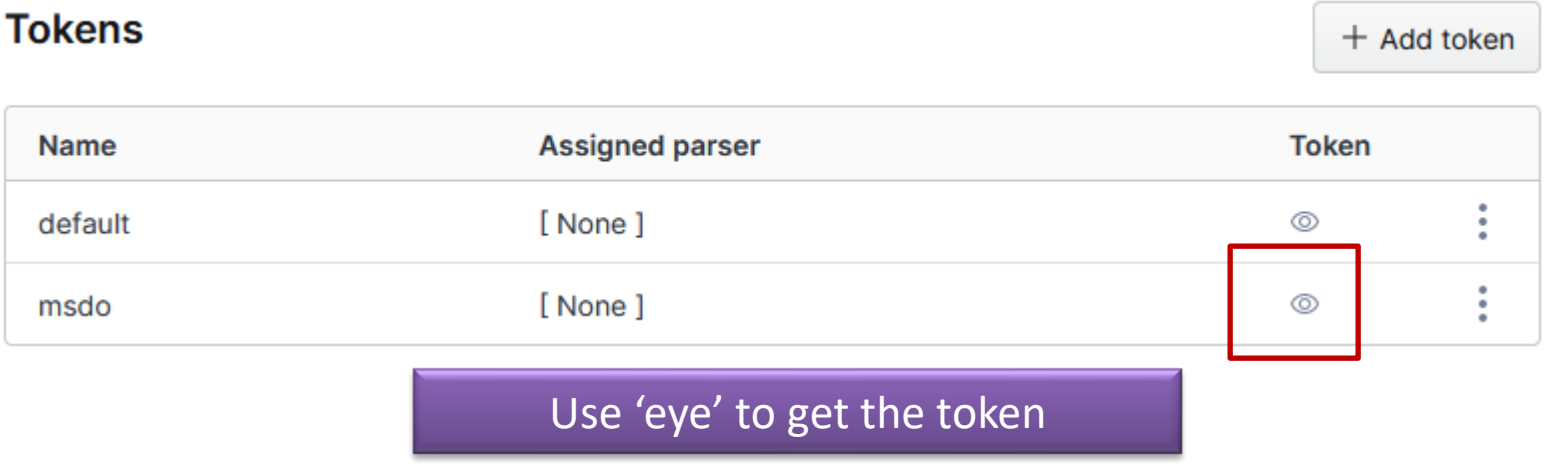
# … and now define Ingest Token



AARHUS UNIVERSITET

# … ingest token




Use ‘eye’ to get the token

AARHUS UNIVERSITET

# Logging to Humio

- Now, the next (big) challenge is
  - Telling Docker to forward logs to Humio
  - Make Humio understand the weird ‘Henrik Bærbak’ logging format
  - Understand what we can do with Humio…

AARHUS UNIVERSITET

# Docker Forwarding

- Fortunately, Humio support in Docker is already built-in

- Just use the ‘splunk’ logging driver of Docker
  - Tell Docker to use splunk, not the default logging driver
  - Tell Docker which machine to ship the logs to
  - Tell Docker the proper credentials (INGEST_TOKEN)

AARHUS UNIVERSITET

# Ingest Token

- The token must be provided for any docker run. One way:
  - export INGEST_TOKEN=(paste token here)

- Now, start your daemon with the proper settings
  - Set driver
  - Set endpoint URL
    - Humio's endpoint
  - Set ingest token
  - Set format to 'raw'
    - Makes parsing easier…

```
export HUMIO_URL=https://cloud.community.humio.com
```

```
export INGEST_TOKEN=f2c2c
```

```
docker run -d -p 7777:7777 --name daemon \
  --log-driver=splunk \
  --log-opt splunk-url=$HUMIO_URL \
  --log-opt splunk-token=$INGEST_TOKEN \
  --log-opt splunk-format=raw \
  henrikbaerbak/private:cave-jar
```

AARHUS UNIVERSITET

# Let’s see some action

AARHUS UNIVERSITET

# I did…

- … these steps
  - Build an image

```
docker build -f Dockerfile-multistage -t henrikbaerbak/private:cave-humio .
```

  - Set the Humio environment

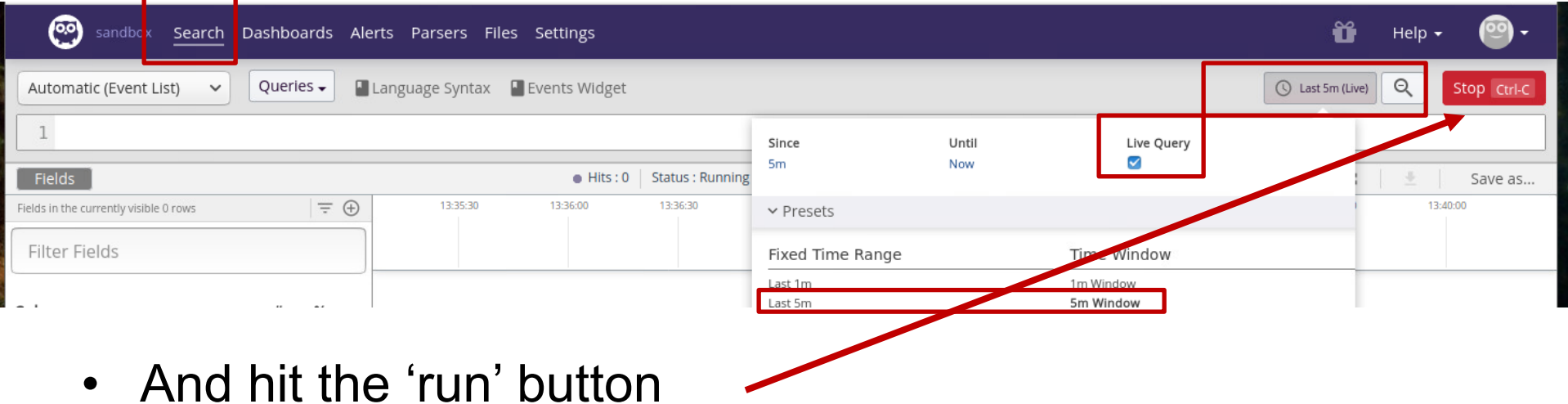
```
csdev@m1:~/proj/cave$       export HUMIO_URL=https://cloud.community.humio.com
csdev@m1:~/proj/cave$       export INGEST_TOKEN=f2c2c
```

  - Run the daemon

```
docker run -d -p 7777:7777 --name daemon \
  --log-driver=splunk \
  --log-opt splunk-url=$HUMIO_URL \
  --log-opt splunk-token=$INGEST_TOKEN \
  --log-opt splunk-format=raw \
  henrikbaerbak/private:cave-humio
```

# Testing

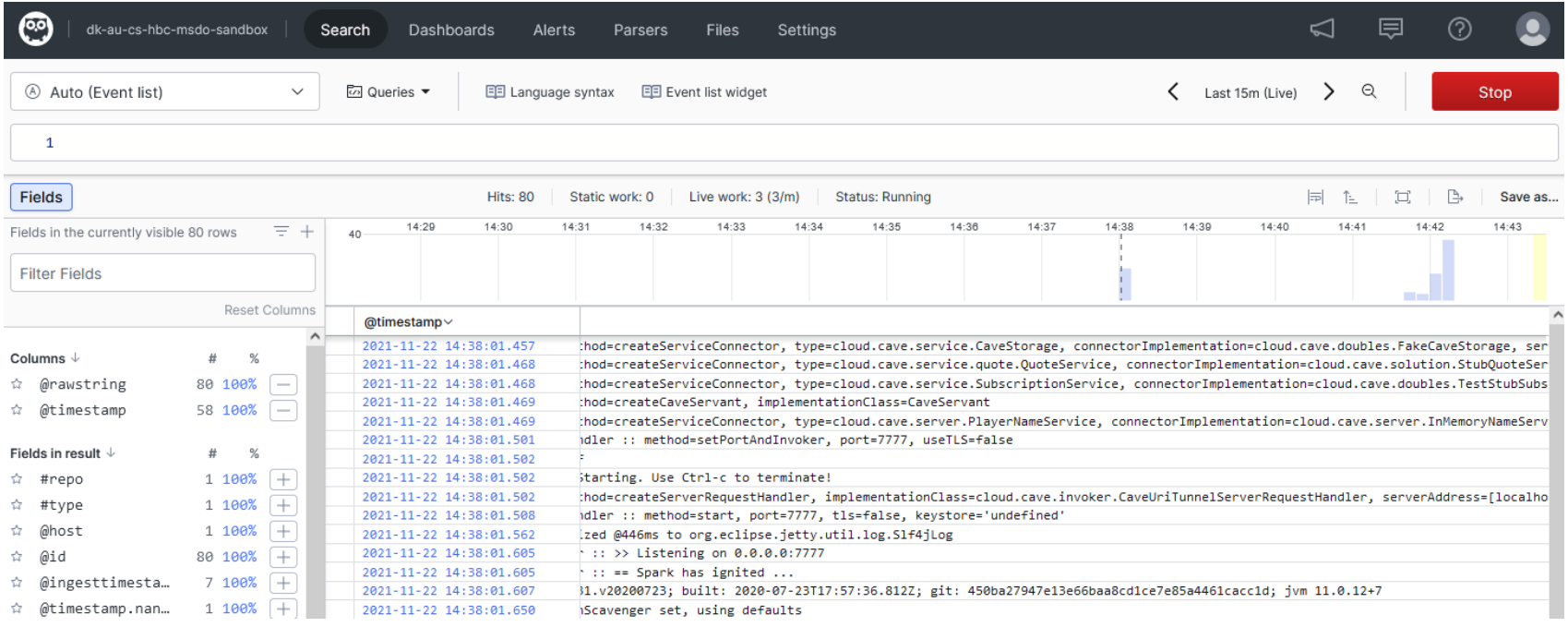
- You should see some log messages coming in…
  - … if you set the search for 5 minutes / live




- And hit the ‘run’ button

AARHUS UNIVERSITET

# Example



AARHUS UNIVERSITET

# But – No Parsing

- Humio understands most common logging formats
  - Accesslog, syslog, json, key-value

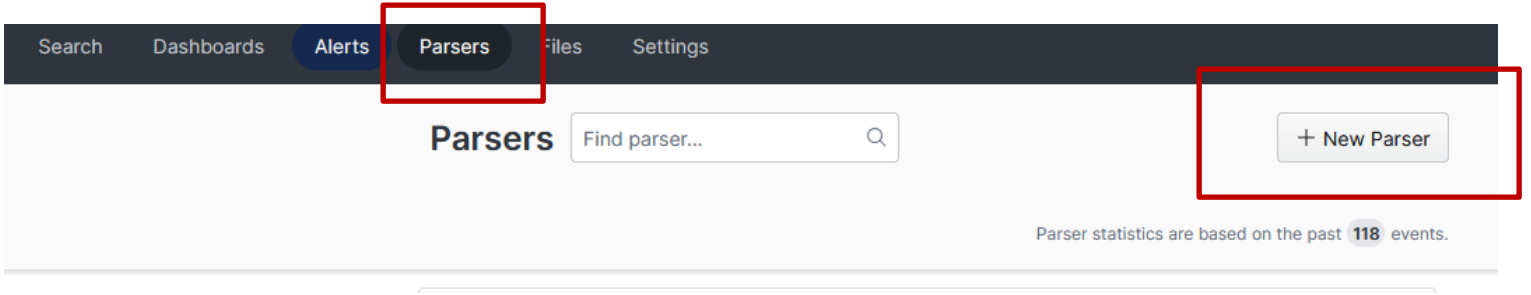
- However, our Log4J property file defines its own
  - Why? Well…

```
log4j.appender.DockerConsole.layout=org.apache.log4j.PatternLayout
log4j.appender.DockerConsole.layout.ConversionPattern=%d{yyyy-MM-dd'T'HH:mm:ss.SSSXXX} [%p] %c :: %m%n
```

- What is the problem?
  - We cannot make queries as each log is just a raw string ☹

# A Parser

- You will need to define a new parser

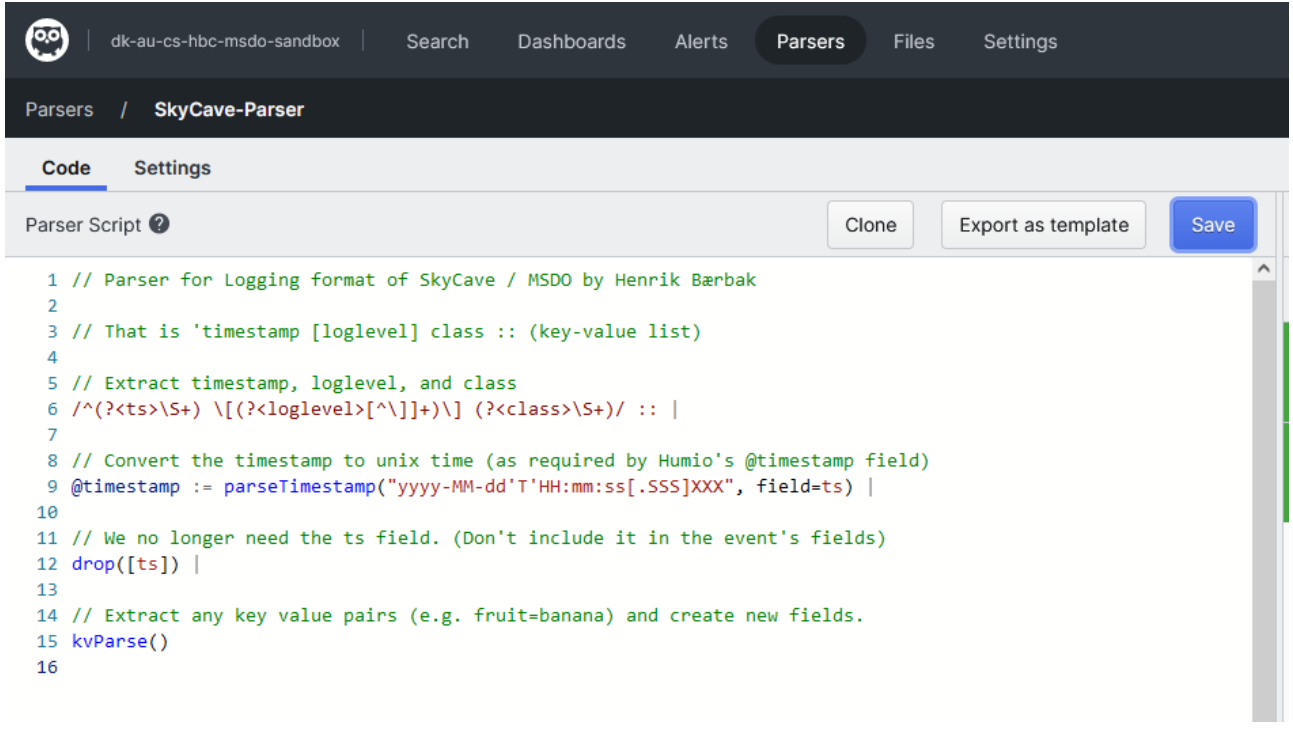


AARHUS UNIVERSITET

# A Parser

- And as my format is (mostly) a key-value based one, enter:

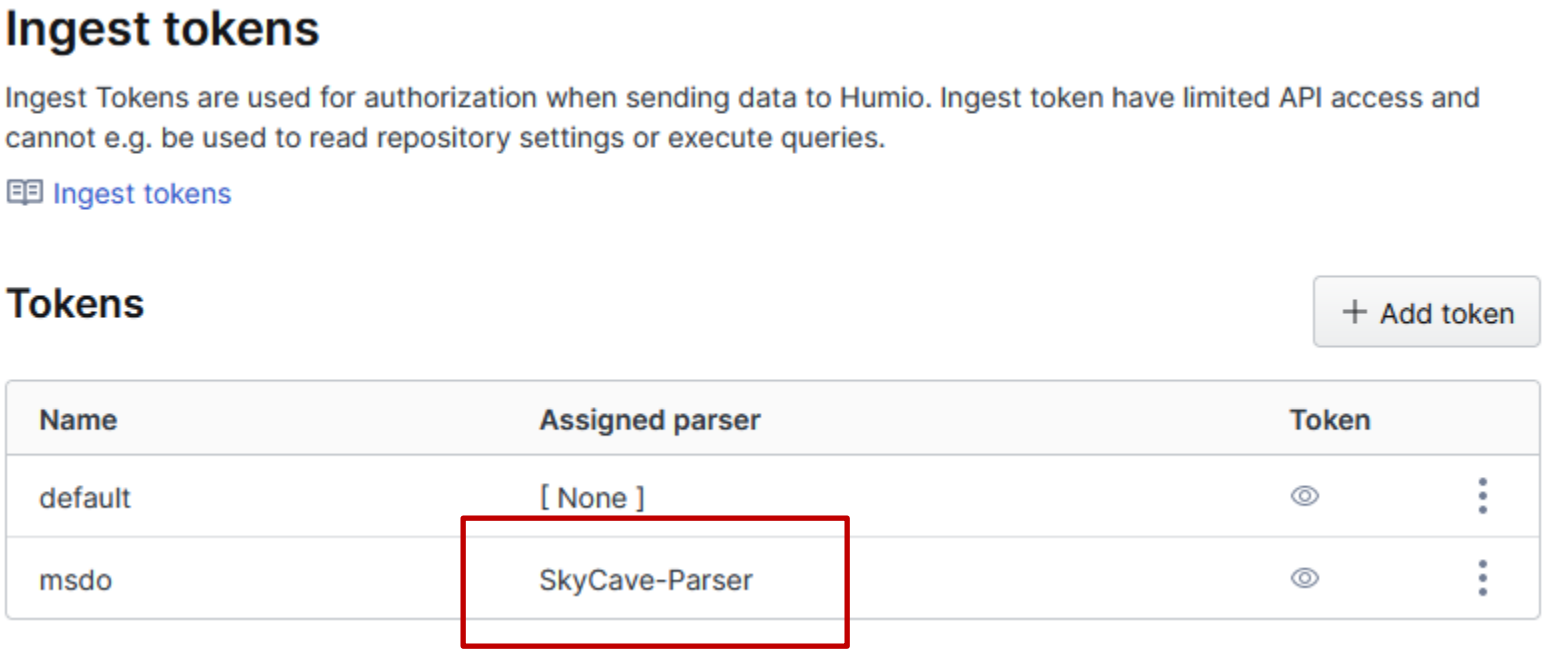
- *Find the file on the weekplan !*



AARHUS UNIVERSITET

# And associate it…

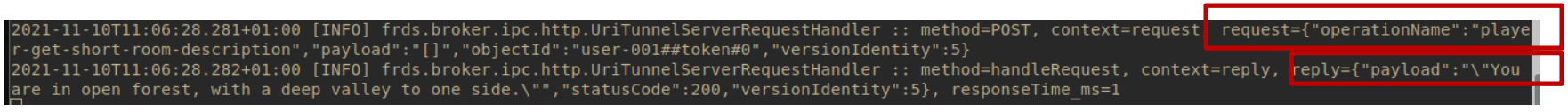
- With your token



AARHUS UNIVERSITET

# *Unfortunately!*

- The most interesting log message is about the daemon accepting and replying to the ‘cmd’ requests
  - The log message stems from the FRDS.Broker library, which unfortunately has the interesting stuff encoded in a json object ☹




- I have *not* been able to create a parser that can handle that format (would love if someone cracked that nut!)
  - Even if Humio states I can: “kvParse() | parseJson(field=reply)”
    - I just get ‘reply=payload’ ???

AARHUS UNIVERSITET

# So, to monitor commands…

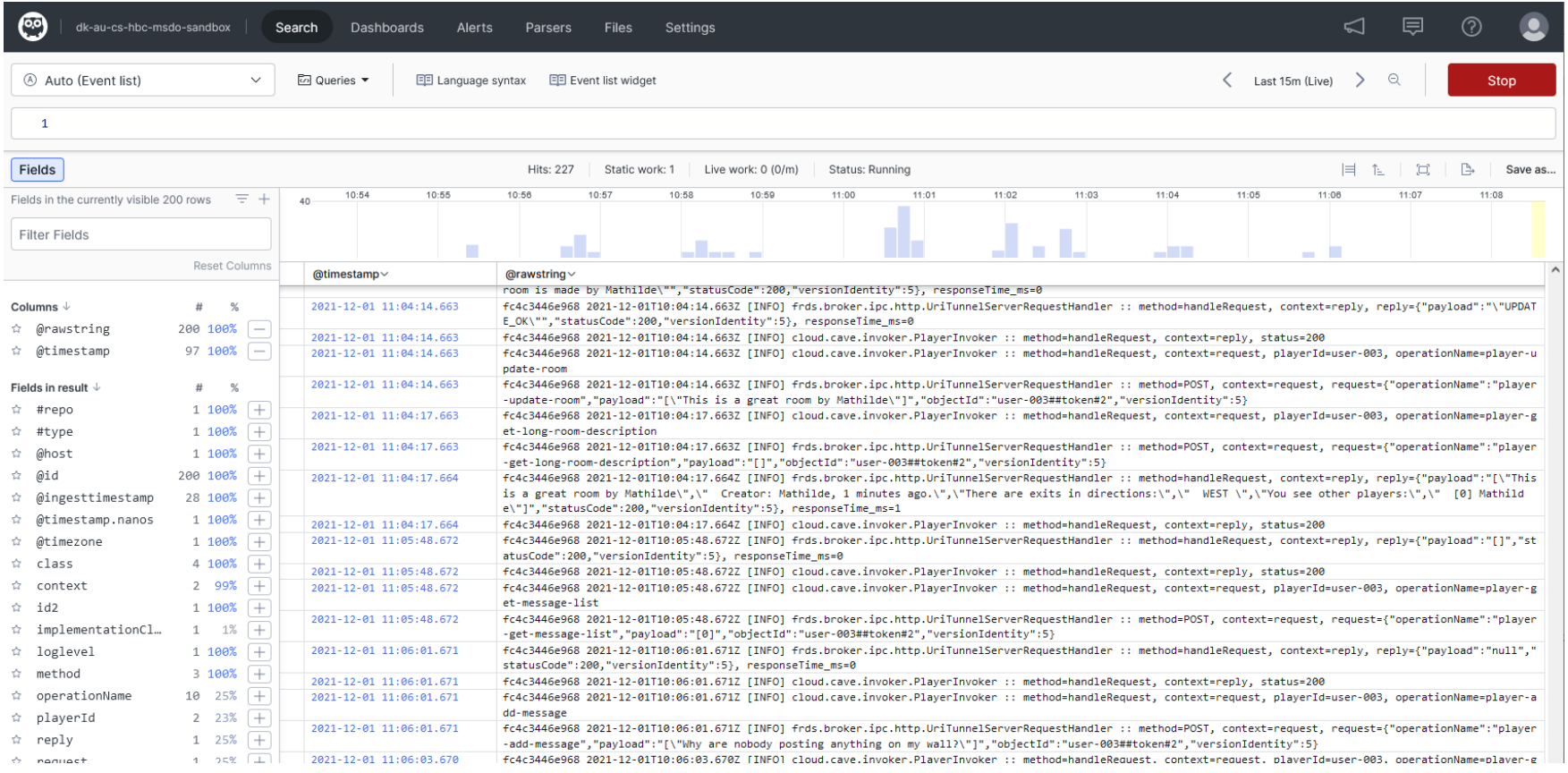
- We have to augment the CaveInvoker's and PlayerInvoker's *handleRequest()*
  - Find updated code on the weekplan.

```
// Added for Humio logging
logger.info("method=handleRequest, context=request, playerId={}, operationName={}", playerId, operationName);
```

```
// Added for Humio logging
logger.info("method=handleRequest, context=reply, status={}", reply.getStatusCode());
```

AARHUS UNIVERSITET

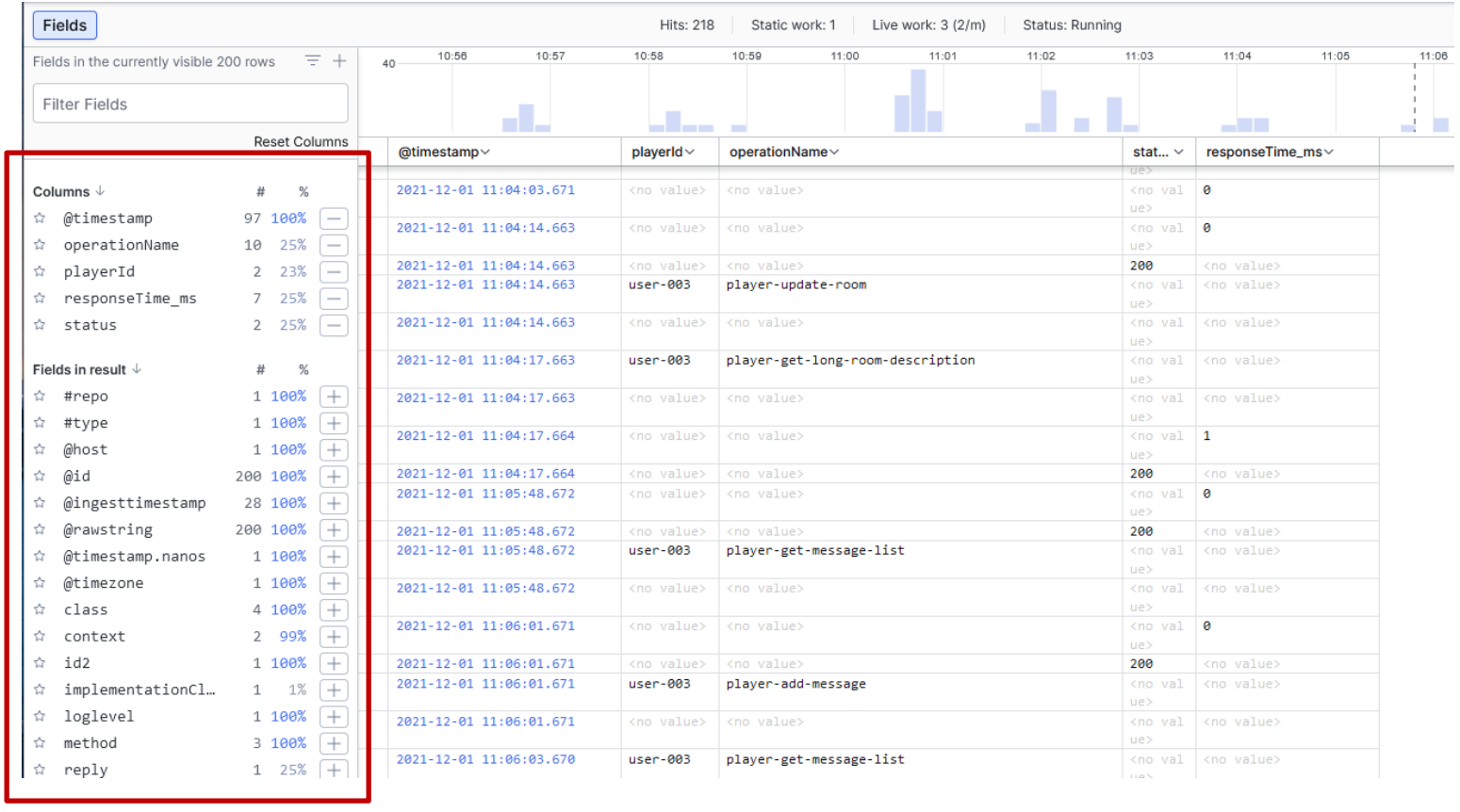
# In Action




Meta: gradle cmd –Pcpf=malkia.cpf

AARHUS UNIVERSITET

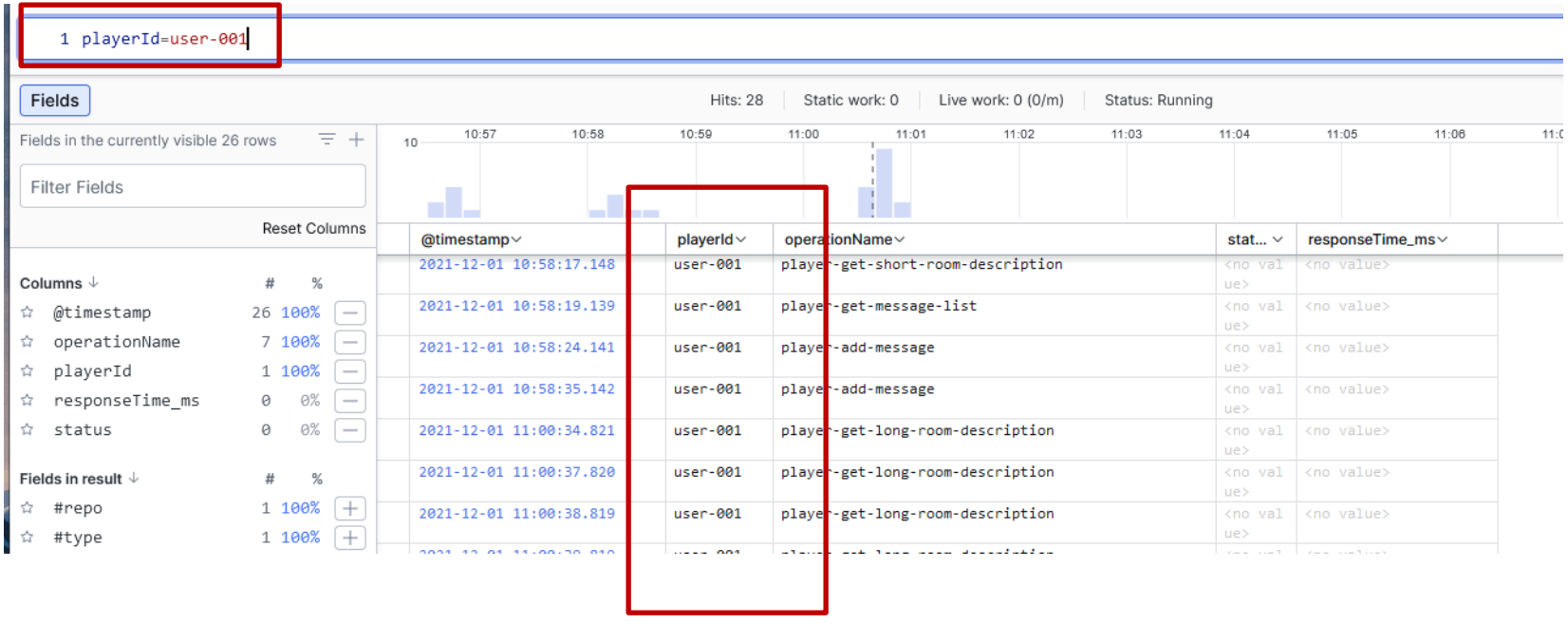
# Selecting Columns



AARHUS UNIVERSITET

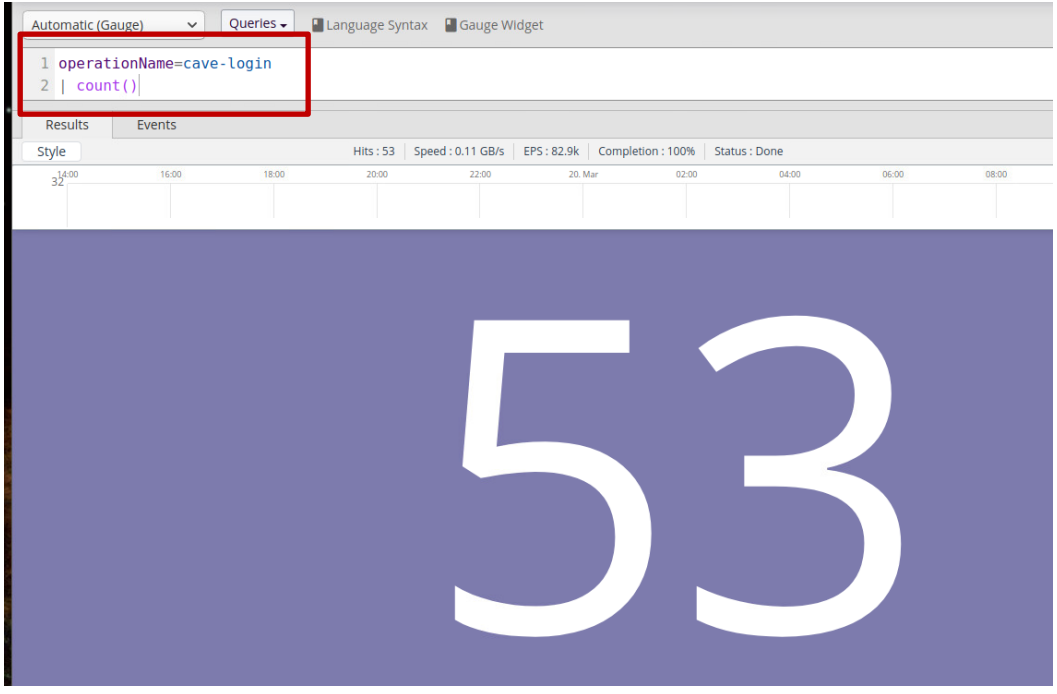
# Query

- You can use a query language
  - Do the tutorial ☺



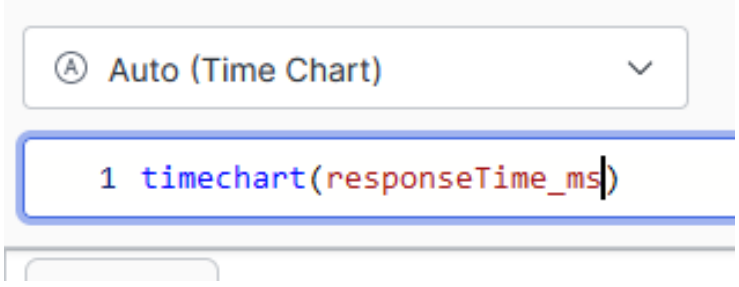
AARHUS UNIVERSITET

# Semi SQL like ☺

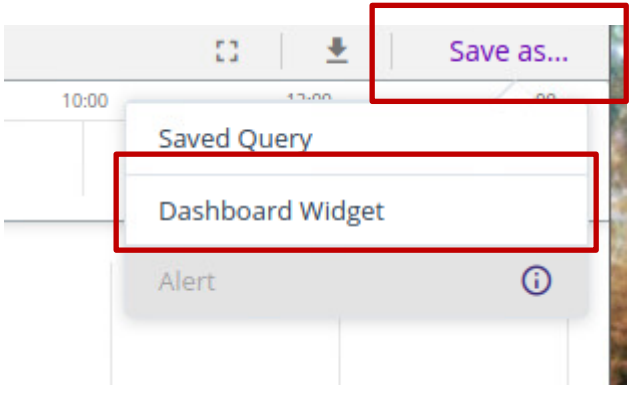


# Dashboards

- Perform a search



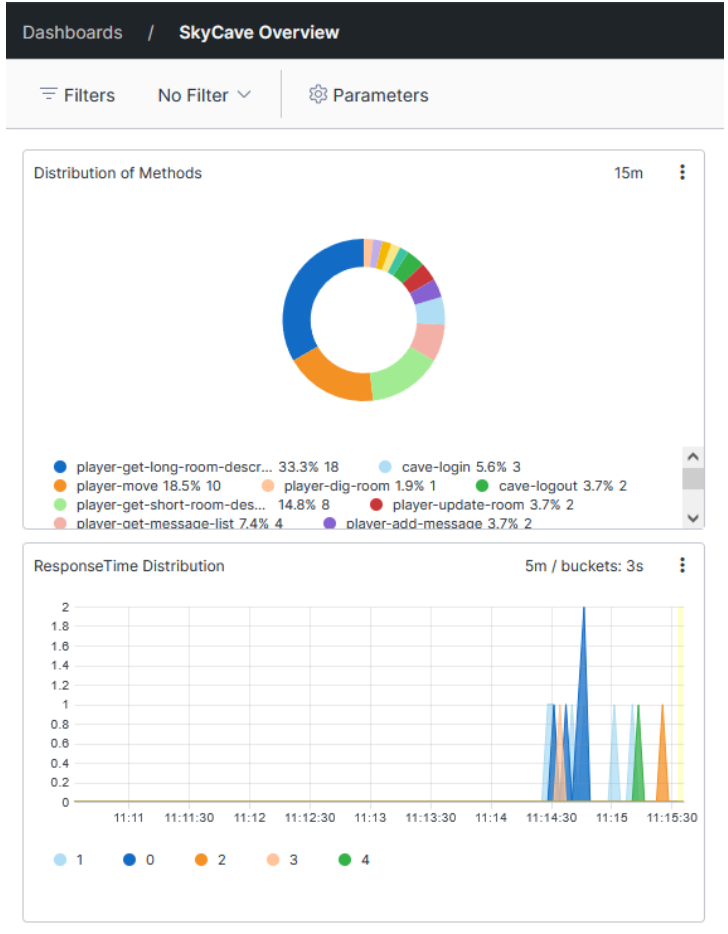

- Hit ‘Save as…’
  - And add to a dashboard



AARHUS UNIVERSITET

# Dashboard Example

- Here I have added two widgets
  - Distribution of Methods
    - Query: operationName=* | groupBy(operationName)
    - As Pie Chart
  - Response time distribution
    - As TimeChart



AARHUS UNIVERSITET

# Swarm

More fiddling ☹

AARHUS UNIVERSITET

# The IngestToken

- Easy to use an environment variable on a single node
- For swarm it is a bit more tricky:

### environment

Add environment variables. You can use either an array or a dictionary. Any boolean values (true, false, yes, no) need to be enclosed in quotes to ensure they are not converted to True or False by the YML parser.

Environment variables with only a key are resolved to their values on the machine Compose is running on, which can be helpful for secret or host-specific values.

- Alas, you have to set it on *each machine* in swarm ☹

To make Environment variables persistent you need to define those variables in the bash configuration files. In most Linux distributions when you start a new session, environment variables are read from the following files:

- `/etc/environment` - Use this file to set up system-wide environment variables. Variables in this file are set in the following format:

```
$ FOO=bar
$ VAR_TEST="Test Var"
```

Fiddling ☹

AARHUS UNIVERSITET

# And the compose-file

- … and then you have to adopt your compose file to set the same properties as for the ‘docker run’ but now in YAML format…

```
docker run -d -p 7777:7777 \
  --log-driver=splunk \
  --log-opt splunk-url=http://localhost:8080 \
  --log-opt splunk-token=$INGEST_TOKEN \
  --log-opt splunk-format=raw \
  henrikbaerbak/private:cave-jar
```

- Jobs done…

- Alternative:
  - Expose your ingest token directly in the compose file. Hm hm…

```
logging:
  driver: splunk
  options:
    splunk-url: "http://m1-
```

And more ☺

# Summary

- Log aggregation is a powerful tool
- Humio has a (comparable) gentle learning curve

- Quite a lot of moving parts still ☺…