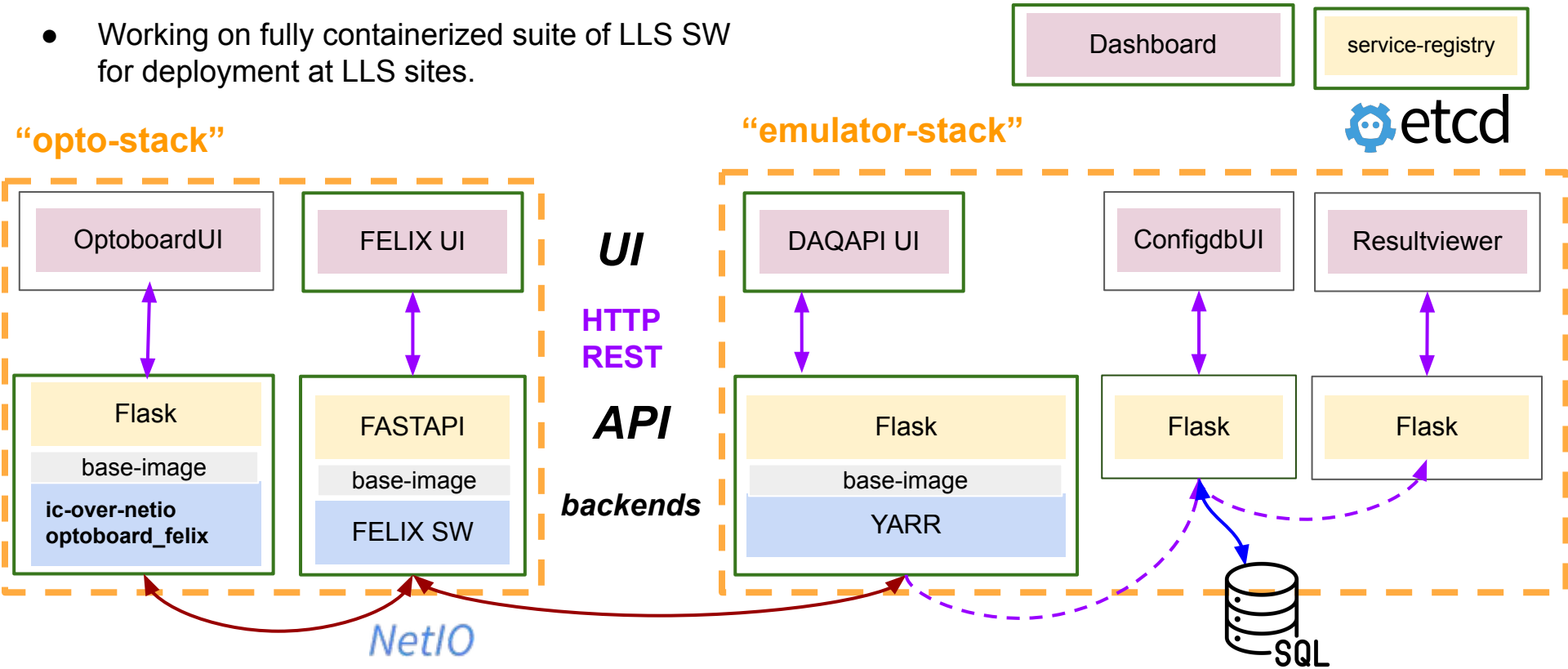
Microservices for ITk Pixel Systemtests Status Report

Ali Can Canbey (Ankara), Daniele Dal Santo (Bern)
Benedikt Vormwald, Leyre Flores (CERN),
Matthias Wittgen (SLAC), Hava Schwartz (SCIPP), Andreas Korn (UCL),
Gerhard Brandt, Jonas Schmeing, Marvin Geyik, Maren Stratmann,
Adrian Miemczyk, Dominik Schlothane, Wolfgang Wagner (Wuppertal)

ATLAS ITk Week
TDAQ-ITk Session
September 2023

Containerized SR1 Deployment in Progress

- Working on fully containerized suite of LLS SW for deployment at LLS sites.



Standard Base Images

- Provide thin Docker wrapper around backends.
  - Image w. preinstalled base SW (Dockerfile: install script).
- SW installation or version upgrade becomes simple redeploy of container.
- Unified “user interface” with similar features for fast onboarding.
- Features:
  - Preconfigured minimal **`compose.yaml`** example ready to run
    - no need to check out repo to use container!
  - s6 init system
    - run scripts at container start
    - run supervised services
  - Configuration / control via environment variables - generally no build needed to standard op.
  - Shims to run executables based on context (current directory) in Docker (locally or remotely) or natively as fall-back → *containers become completely transparent.*
- More or less complete base images: FELIX (G. Brandt, B. Vormwald, A. Canbay), YARR (A. Korn), OB software (D. Dal Santo), ConfigDB (J. Schmeing), DAQAPI (M. Geyik)

```
docker compose pull &&
docker compose up -d
```

FELIX base image

https://demi.docs.cern.ch/itk-demo-felix/base-image/

- Startup sequence and FELIX app managed by s6 https://skarnet.org/software/s6-linux-init/
- At container start:
  - init FELIX
  - convert FELIX configuration file
  - configure FELIX
  - start felixcore
- Previously:
  - Manually ran sequence of scripts
  - Manual restart of FELIX app needed if crashed
  - Needed to be killed eventually
  - Unclear who was running what (many checked out versions, single user account for everybody)

→ *none of this a problem when running in container*

Envvar	Purpose	Default
FELIX_DATA_INTERFACE	DATA_INTERFACE	lo
FELIX_CARD_NUMBER	Set Felix device	0
FELIX_INITIALIZE	At startup run 0=nothing, 1=flx-init, 2=flx-config	0
FELIX_CONFIG_FILE	Use custom configuration file	/config/flx.cfg
FELIX_APP	Start FELIX Application: 0=none, 1=felixcore, 2=felix-star	0

Environment variables to control FELIX SW using s6 init

see also talk by Leyre Flores about testing in SR1

Studies on Configuration Files

A solution to facilitate the description of the e-links that are in use has been put in place by creating a generator script that accepts a user-friendly JSON file and translates to FELIX configuration.

The main important parameters required to be able to configure FELIX.

```
FELIX Card ID          : 0
Device ID              : 0

LINK ID                : 0
        polarity       : 0
        icec           : 1
        rx channels    : 4
        tx channels    : 0

LINK ID                : 1
        polarity       : 0
        icec           : 1
        rx channels    : 64, 72, 76, 80, 84
        tx channels    : 64, 66, 70

LINK ID                : 2
        polarity       : 0
        icec           : 1
        rx channels    : 132
        tx channels    : 128
```

```json
{
  "FelixID": "01",
  "DeviceID": 0,
  "Links": {
    "0": {
      "polarity": 0,
      "icec": 1,
      "rx": [4],
      "tx": [0]
    },
    "1": {
      "polarity": 0,
      "icec": 1,
      "rx": [64,72,76,80,84],
      "tx": [64,66,70]
    },
    "2": {
      "polarity": 0,
      "icec": 1,
      "rx": [132],
      "tx": [128]
    }
  }
}
```

JSON File

Studies on Configuration Files

The generated user-friendly JSON file needs to be converted to a raw configuration file that FELIX can understand.

Human readable configuration file

```
{
  "FelixID": "01",
  "DeviceID": 0,
  "Links": {
    "0": {
      "polarity": 0,
      "icec": 1,
      "rx": [4],
      "tx": [0]
    },
    "1": {
      "polarity": 0,
      "icec": 1,
      "rx": [64,72,76,80,84],
      "tx": [64,66,70]
    },
    "2": {
      "polarity": 0,
      "icec": 1,
      "rx": [132],
      "tx": [128]
    }
  }
}
```

convertJSON.py

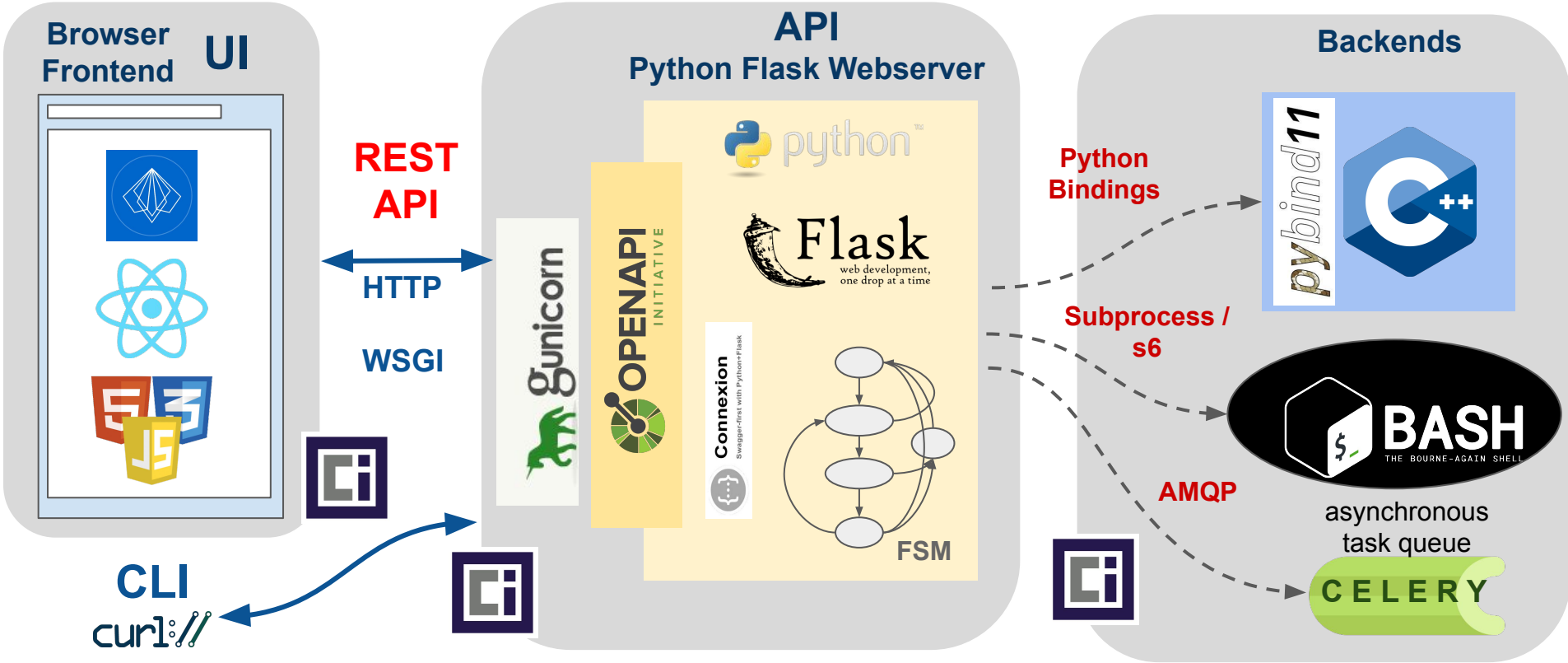
Based on Vakhtang's script

Raw configuration file

```
GBT_DATA_RXFORMAT2=0xFF
GBT_RXPOLARITY=0x0
DECODING_REVERSE_10B=0x1
MINI_EGROUP_FROMHOST_00_IC_ENABLE=0x1
MINI_EGROUP_TOHOST_00_IC_ENABLE=0x1
MINI_EGROUP_FROMHOST_00_EC_ENABLE=0x1
MINI_EGROUP_TOHOST_00_EC_ENABLE=0x1
ENCODING_LINK00_EGROUP0_CTRL_EPATH_ENA=0x1
ENCODING_LINK00_EGROUP0_CTRL_EPATH_WIDTH=0x1
ENCODING_LINK00_EGROUP0_CTRL_PATH_ENCODING=0x4
ENCODING_LINK00_EGROUP1_CTRL_EPATH_ENA=0x0
ENCODING_LINK00_EGROUP1_CTRL_EPATH_WIDTH=0x1
ENCODING_LINK00_EGROUP1_CTRL_PATH_ENCODING=0x0
ENCODING_LINK00_EGROUP2_CTRL_EPATH_ENA=0x0
ENCODING_LINK00_EGROUP2_CTRL_EPATH_WIDTH=0x1
ENCODING_LINK00_EGROUP2_CTRL_PATH_ENCODING=0x0
ENCODING_LINK00_EGROUP3_CTRL_EPATH_ENA=0x0
ENCODING_LINK00_EGROUP3_CTRL_EPATH_WIDTH=0x1
ENCODING_LINK00_EGROUP3_CTRL_PATH_ENCODING=0x0
DECODING_LINK00_EGROUP0_CTRL_EPATH_ENA=0x0
DECODING_LINK00_EGROUP0_CTRL_EPATH_WIDTH=0x4
DECODING_LINK00_EGROUP0_CTRL_PATH_ENCODING=0x0
. . .
```

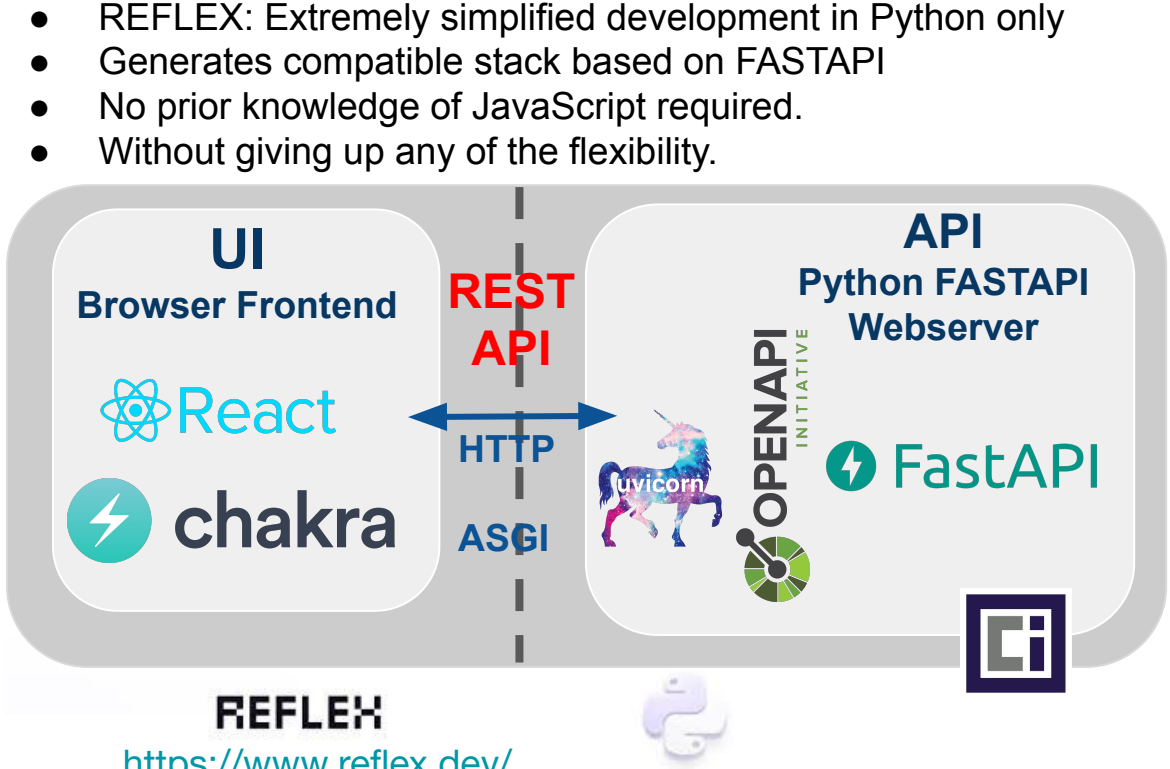
flx-config load FILENAME efficient way to configure

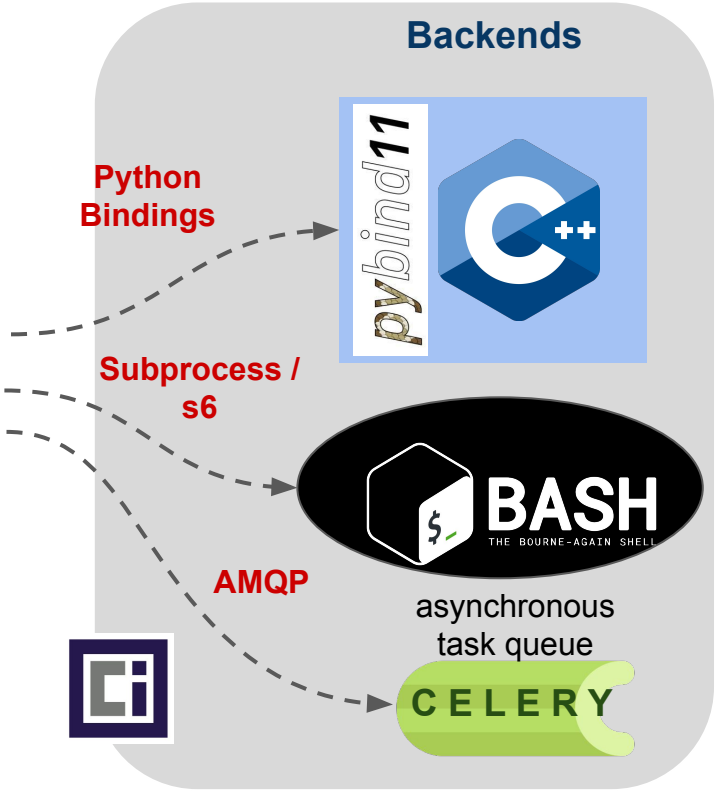
Reminder: Standard Architecture of a Microservice



Architecture of a Microservice using REFLEX

- REFLEX: Extremely simplified development in Python only
- Generates compatible stack based on FASTAPI
- No prior knowledge of JavaScript required.
- Without giving up any of the flexibility.





API Part

- "Swagger UI" + openapi.yaml automatically generated by Reflex / FASTAPI
- Previously need to be written by hand.
- Manual modification still possible.

OPENAPI INITIATIVE

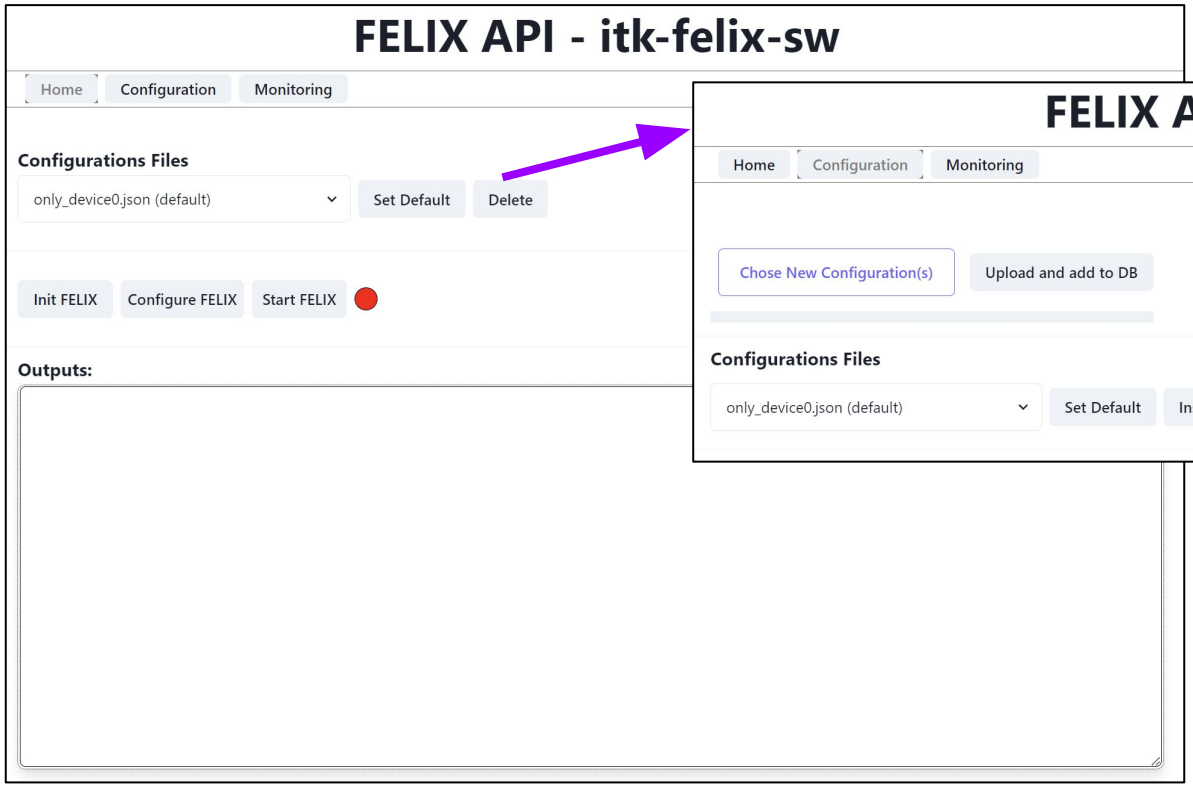
http://localhost:8000/infoconfig : shows information about configuration files
http://localhost:8000/getconfigs : shows existing configuration files
http://localhost:8000/getconfigs/File_Name : inspects the given configuration file (File_Name)
http://localhost:8000/setdefault/File_Name : makes the given configuration file default
http://localhost:8000/deleteconfig/File_Name : deletes the given configuration file default

http://localhost:8000/infofelix : shows information about FELIX status
http://localhost:8000/initfelix : initializes FELIX
http://localhost:8000/configurefelix : configures FELIX with default configuration file
http://localhost:8000/startfelix : starts FELIX
http://localhost:8000/stopfelix : stops FELIX

http://localhost:8000/getelink : shows information about electrical links
http://localhost:8000/getolink : shows information about optical links
http://localhost:8000/getpod : shows information about optical power

GUI Part

- Manual FELIX control per web GUI if startup sequence does not handle needed workflow

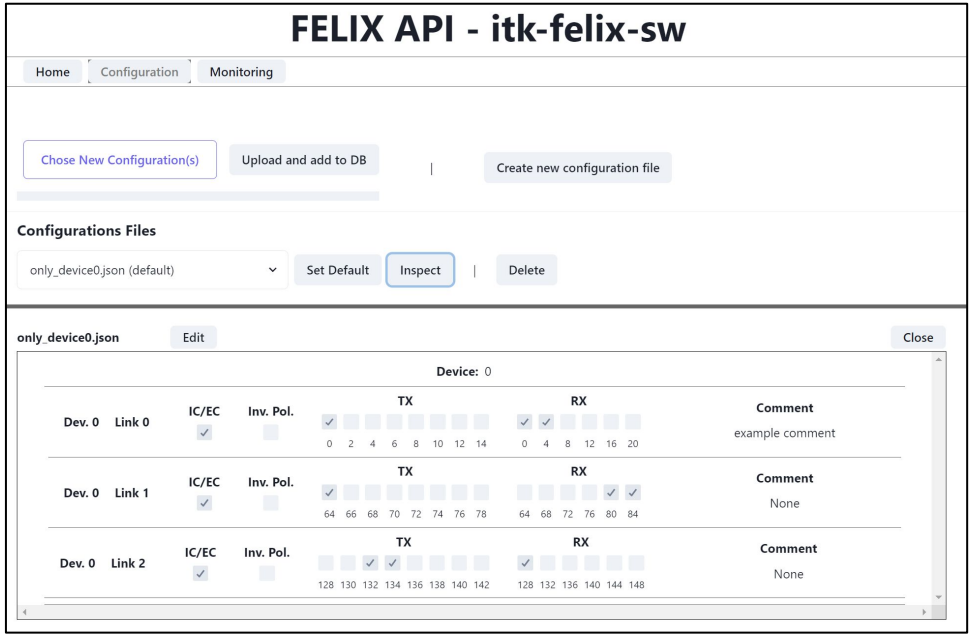


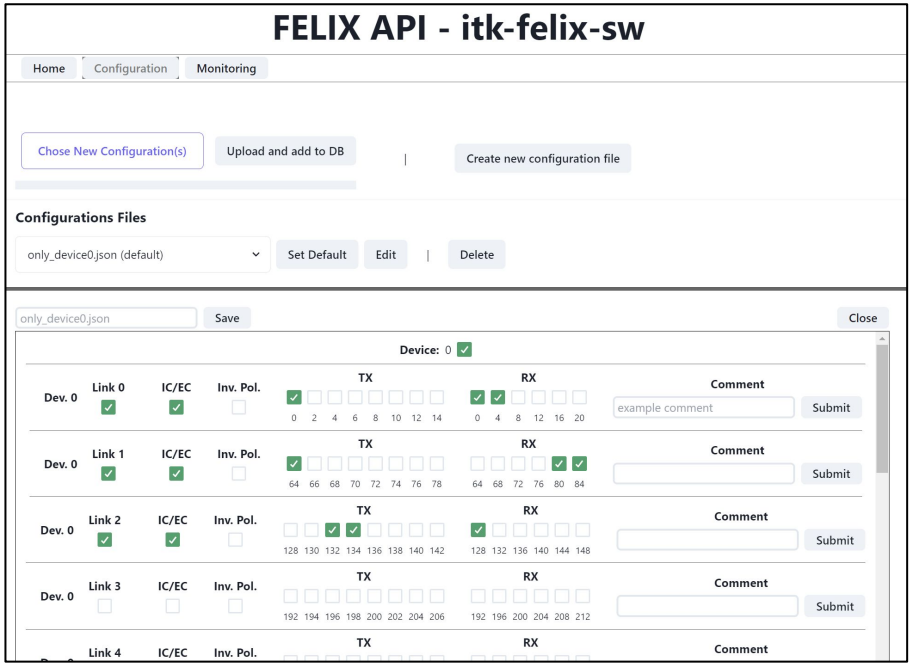
- Set default/delete the existing configuration files

- Initializing FELIX
- Configuring FELIX
- Starting/stopping FELIX

GUI Part

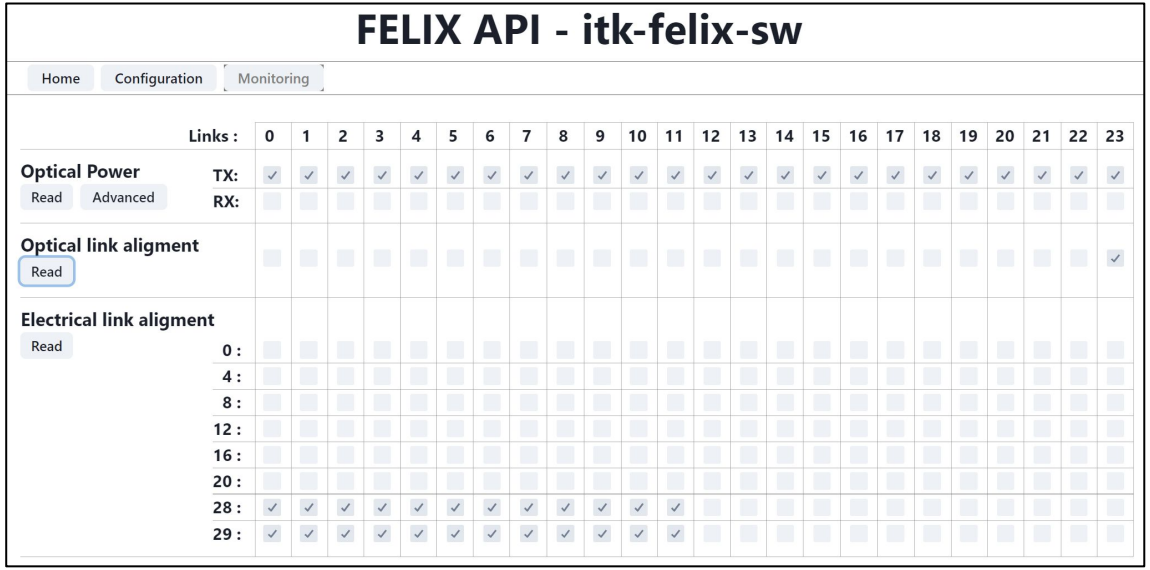
- Graphical FELIX configuration editor pages
- Created within few weeks over summer ‘23 without previous knowledge of JavaScript

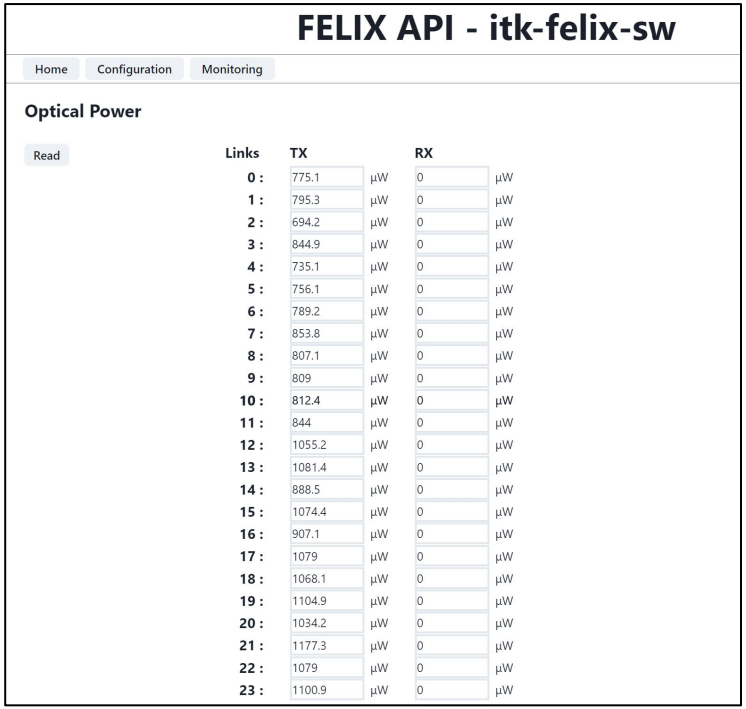




GUI Part

- Graphical FELIX monitoring page
  - Show information about electrical links
  - Show information about optical links
  - Show information about optical power





OptoBoard SW Container Status

https://gitlab.cern.ch/atlas-itk-pixel-systemtest/itk-demo-sw/containers/optoboard-container

- Packaged Bern optoboard_FELIX Python script
- ic-over-netio communication tool (felixcore)
- Lots of testing in SR1 to confirm containerized OB configuration just as efficient as native configuration
  - Huge time difference (60s vs. 18s) was traced to “fixes” in ic-over-netio in recommended itk-felix-sw TDAQ version.
  - Reverted back to NSW version using a standalone build.
    → Problem not container related. DeMi team need to dig into backends almost everywhere.
- lpgbt-comm backend (using felix-star) using same build chain ready but not yet tested.
- Very advanced base-image with full configuration control at startup.
- *Ready for OB users.*

Compilation Wrappers for FELIX Libraries

- Felix release depends on LCG installation
  - **Monolithic** dependencies mostly not needed at run-time
- Modified *cmake_tdaq* for compilation
  - **neither compatible with cmake standards**
  - **nor TDAQ cmake system**
  - external libraries, e.g. libfabric, provided as **binaries in gitlab**
    - no consistent full build possible
    - depends on gcc version and OS
    - Slow cloning due to **complex submodule structure** and GB of binaries
- Compilation wrappers for Felix libraries allow to “build only what you need” for specific application
  - Real dependency management with *CPM-cmake*
  - YARR is already a similar mechanism to just build Felix client libraries on-the-fly
- Results in **very small installations**, e.g. opto stack, ideal for container use

- Wrapper repos: https://gitlab.cern.ch/atlas-itk-pixel-systemtest/itk-demo-sw/external
- CPM-cmake for dependency management: https://github.com/cpm-cmake/CPM.cmake

```
CPMAddPackage(
        NAME simdjson
        GITHUB_REPOSITORY "simdjson/simdjson"
        GIT_TAG "v3.2.0"
        OPTIONS "CMAKE_CXX_FLAGS -fPIC"
)
```

Add external packaged for Felix, e.g. *simdjson*

```
add_library(felix-client SHARED
        ${client_src}/felix_client.cpp
        ${client_src}/felix_client_thread_impl.cpp
        ${client_src}/felix_client_util.cpp
        ${client_src}/felix_client_info.cpp
        ${client_src}/felix_client_context_handler.cpp
        ${client_src}/clog.c)
```

Re-define Felix libraries definitions

```
FetchContent_Declare(felix-def
        GIT_REPOSITORY https://gitlab.cern.ch/atlas-tdaq-felix/felix-def.git
        # GIT_TAG
        GIT_SHALLOW YES
        GIT_SUBMODULES include
)
```

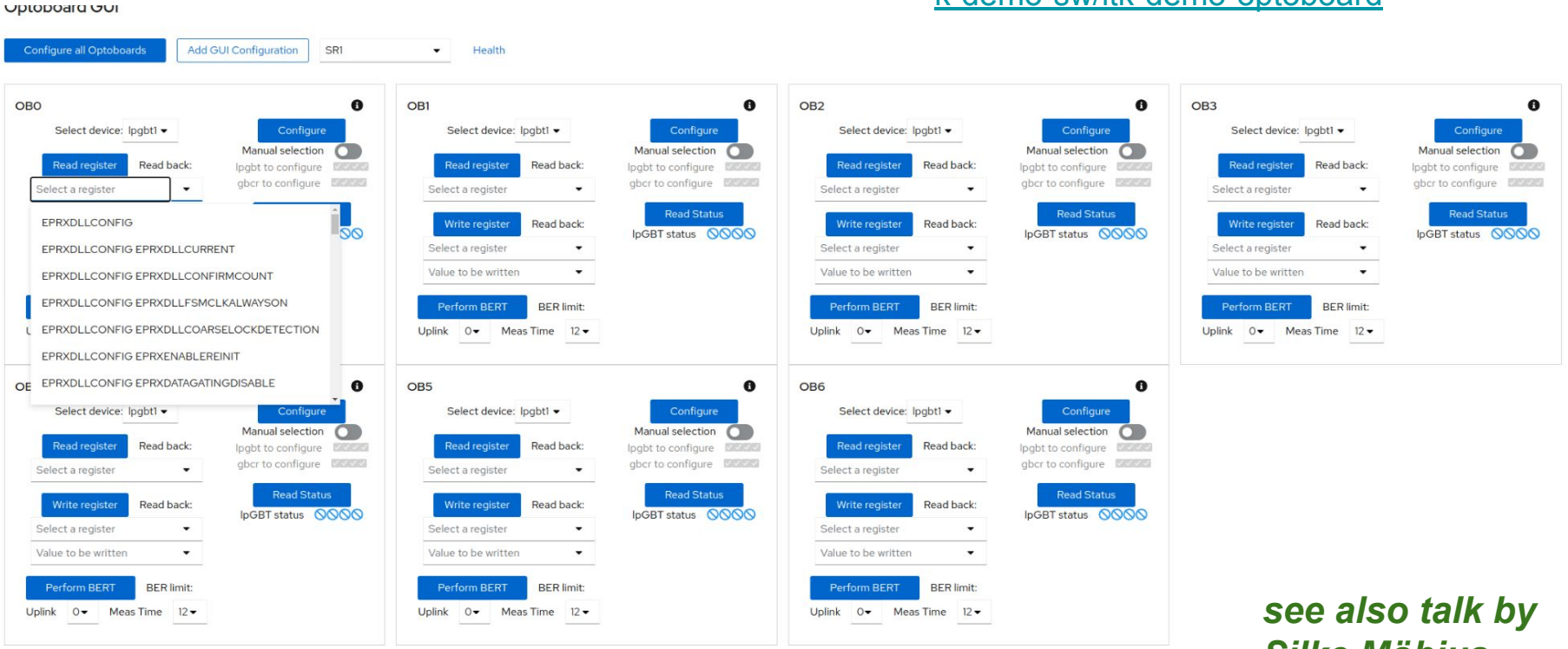
Fetch a Felix repo from original sources, shallow clone without resolving submodules

```
add_executable(link_test tests/link_text.cc)
target_link_libraries(link_test felix-client)
```

Definition link library sufficient to manage link and include paths

Optoboard Microservice UI Status

https://gitlab.cern.ch/atlas-itk-pixel-systemtest/itk-demo-sw/itk-demo-optoboard



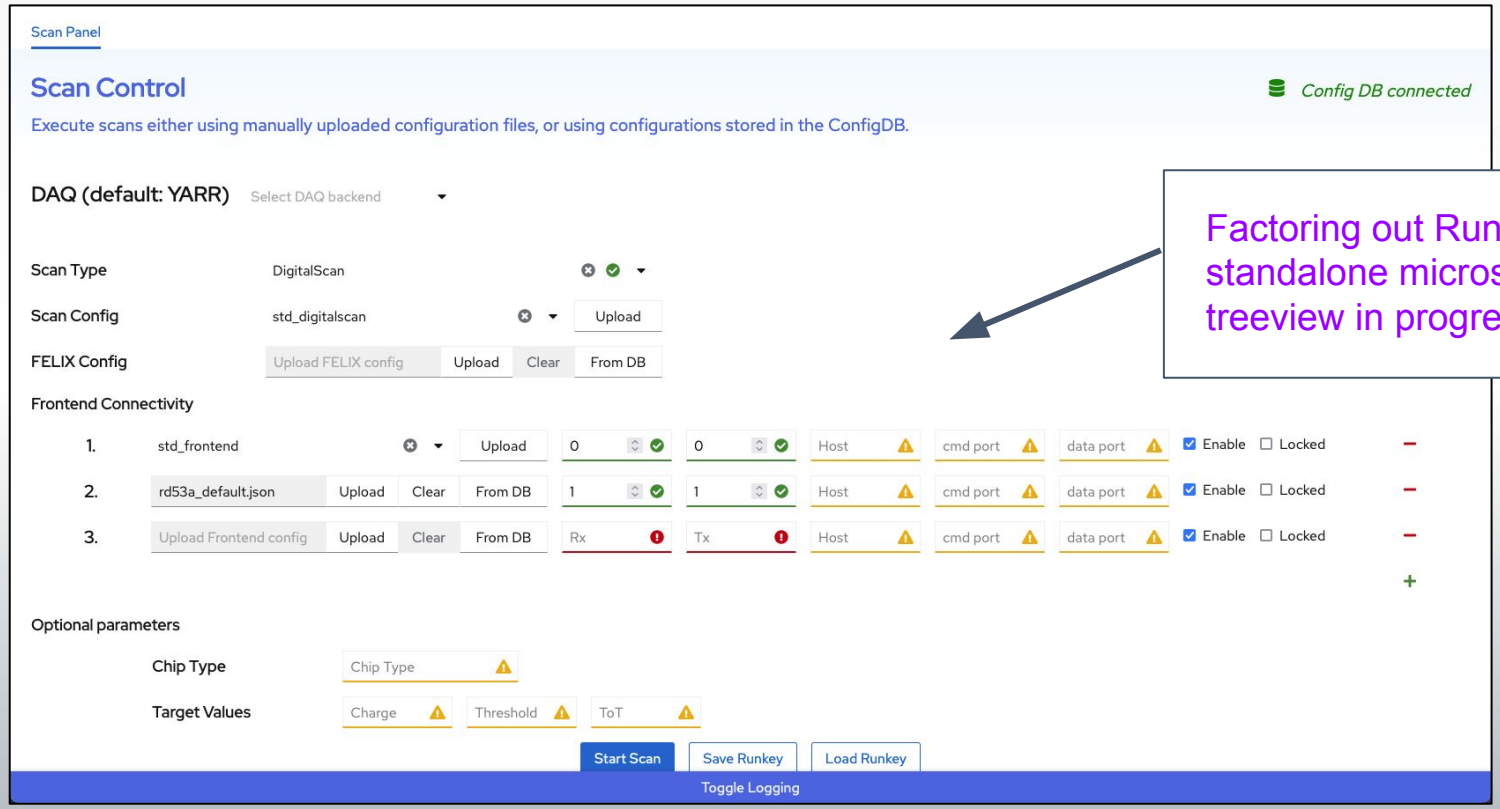
Daniele Dal Santo

see also talk by Silke Möbius

Status of DAQ API microservice

- Provides generalised, Python-based interface to DAQ software
- Currently maintaining **YARR** backend
- Fully integrated with the ConfigDB
  - Reading of config files or entire runkeys as scan configuration
  - Saving of scan results:
    - Post-fit configs to be used in later scans
- Dockerization allows for easy deployment of DAQ software without necessity for local installation
- Currently two interfaces exist for the DAQ API:
  - **Browser GUI** consistent with other itk-demo-sw microservice GUIs
  - **CLI** in the style of the YARR scanConsole for easy integration ← **NEW**

DAQ API GUI Status



Factoring out Runkey Editor as standalone microservice based on treeview in progress

DAQ API CLI

- CLI can be used outside of the Docker container
  - internally using docker exec / shims
  - transparent for user
- Can take configs on the local filesystem as input
  - Files are mounted into Docker and stored in the Config-DB
  - Scan is executed using the provided config files
  - Results are saved in the Config-DB and can be viewed with the resultviewer
- Syntax similar to YARR scanConsole:

```
[geyik@daqdev11 example]$ runScan -c connectivity.json -r ctrl_emulator.json -s std_digitalscan.json
Successfully executed scan with ID 2.
Results have been saved to the database.
```

YARR base-image

https://gitlab.cern.ch/atlas-itk-pixel-systemtest/itk-demo-sw/containers/yarr-containers

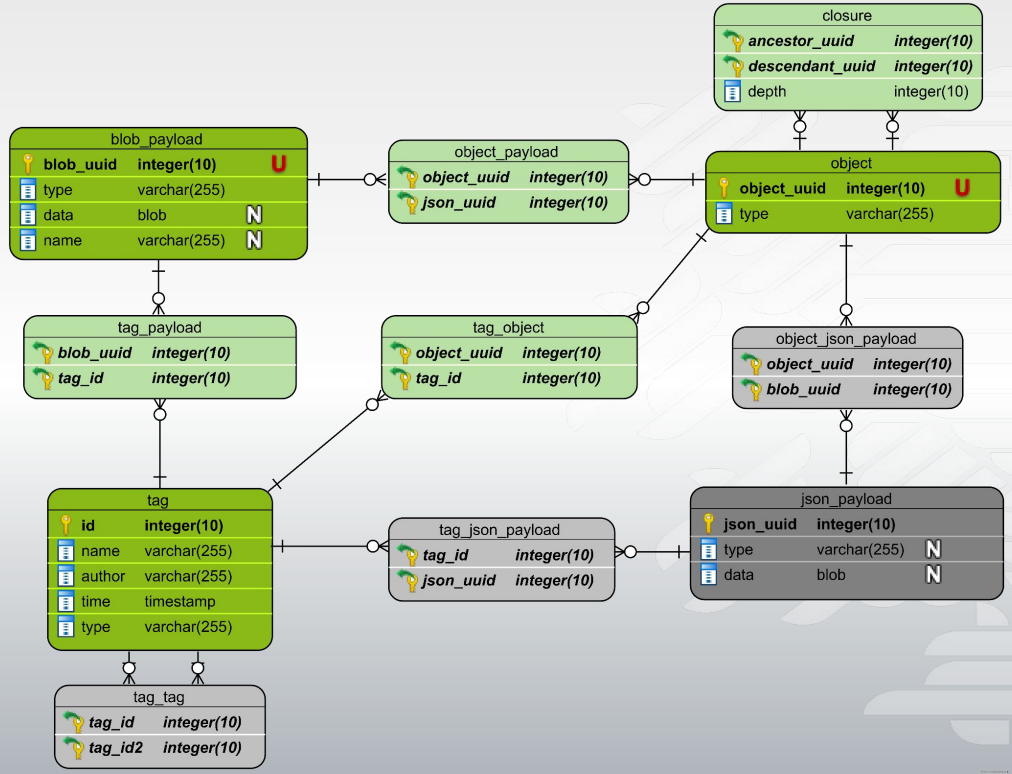
- Much progress over summer by A. Korn (UCL)
- Image builds and tags based on latest tags available in YARR repo
- Standard repo workflow scripts implemented (bootstrap, setup, config)
- Many base-image features implemented
  - s6-init
  - shims
- Needs testing, limited access to full FELIX DAQ chains
- Not yet used in DAQAPI microservice

itk-demo-configdb: overview

- Microservice that acts as an interface to a database storing:
  - Runkeys (connectivity and configuration data)
  - Scans and their results
- Tree based database schema
  - Connectivity tree, whose nodes have configuration data payloads
  - Runkey trees include connectivity trees and additional branches for scans and results
- Supports SQL Databases like MariaDB, SQLite and ORACLE
  - Flexible architecture, more backends can be added
  - Also file system export / import

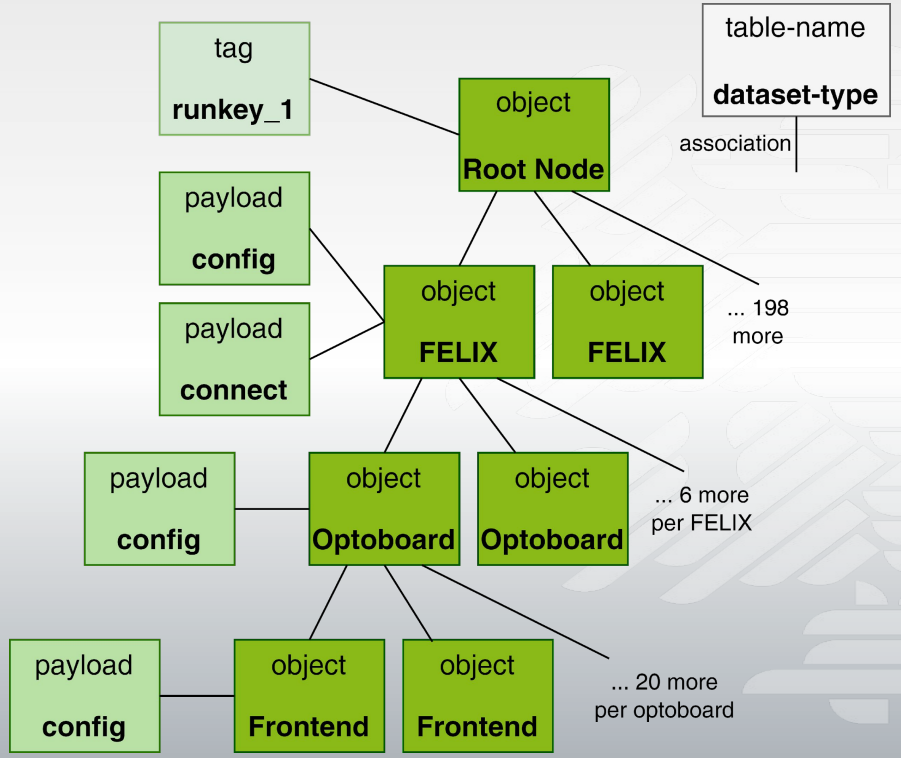
itk-demo-configdb: current database schema

- Objects point at each other
  - In this way they can span a tree
  - Root nodes of these trees are called runkeys
- Objects can point to multiple payloads
  - JSON
  - (BLOB)
- Tags describe or group objects, runkeys and payloads
- Final model will be normalized



itk-demo-configdb: runkey data model

- Estimate for a runkey for the full ITk pixel detector:
  - Root node
  - 200 FELIXs
  - 1.500 optoboards
  - 34.000 frontends
- Amount of payload files per object is customizable
- Current size estimate:
  - 3,1 GB with compressed payloads
  - Uncompressed: about 20 GB

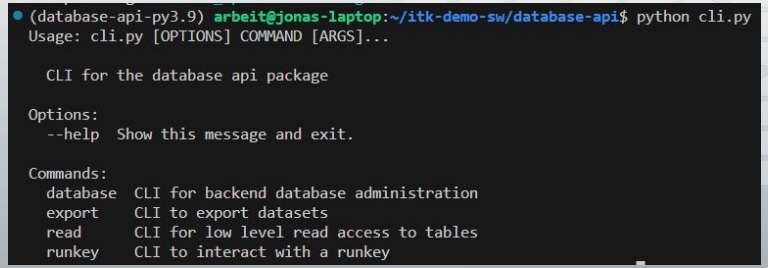


itk-demo-configdb: recent changes

- Rewrite of read and write operations
- Previously recursive operations were used to create and read trees
  - Each node in the tree needed one SQL command when reading or writing
- Bulk insertion for writing → Reduce ORACLE costs
  - SQL feature that allows insertion of multiple datasets at once
  - Only 4 SQL commands (one for each table) are needed to write a runkey of any size
  - Increased write speed by a factor of 30 in first tests
- Closure table for reading
  - Stores every connection between ancestors and descendants (not just direct children)
  - Allows to read entire trees and subtrees with one SQL query
  - Increased read speed by a factor of 10

itk-demo-configdb: GUI and CLI

- New version of ConfigDB GUI
  - Implemented by Dominik Schlothane
  - Main feature is **tag viewer** and **runkey viewer**:
    - Visualizes trees and lists
    - Shows root objects, children and payloads of connectivity tree
    - tree view component is reusable (e.g., for the resultviewer)
- Now working on expanding runkey viewer to a full-blown runkey editor
  - Separate microservice
- In addition to GUI the ConfigDB offers a CLI
  - Database administration, runkey management, dataset read operations and import/export via a python package
  - To Do: Implement shims to make CLI more accessible



itk-demo-configdb: tag overview page

Tags Payloads

Tag Access

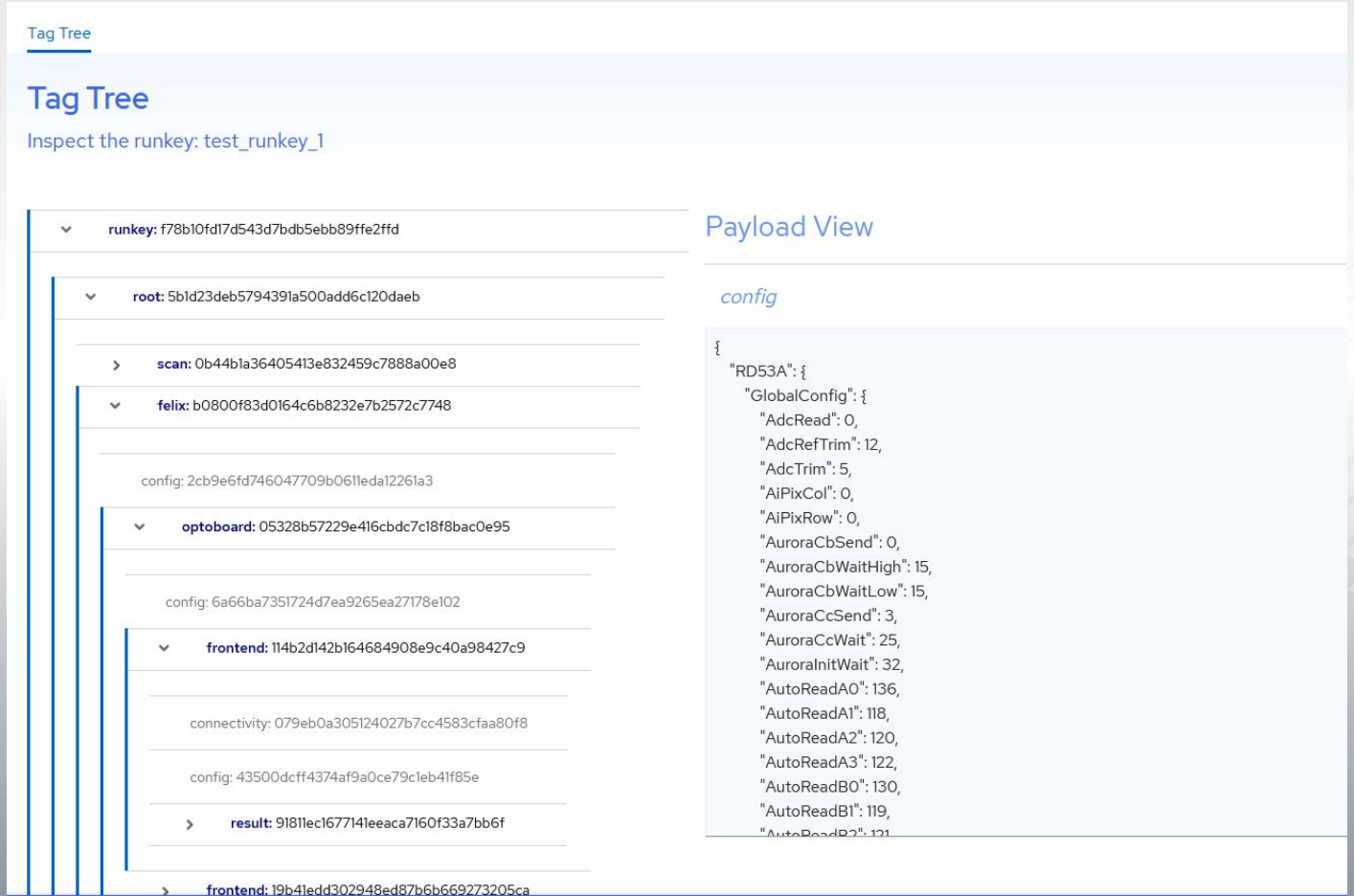
Backend DB connected

Check the status of tags stored in the Database.

Tag Name | Tag Type | 1 - 5 of 5

Name	Type	ID
std_digitalscan	scan	688c95be208041a59d5fc7432d19fa42
my_runkey_yarr	runkey	6ee7718d845c428391cc7ceccf1ba834
my_runkey_yarr2	runkey	a2a00bb0efdc4bc69545cb902ac4c164
my_rk	runkey	dbe7ec2ad4634cf2a8aadf4ee4ce39be
test_runkey_1	runkey	f78b10fd17d543d7bdb5ebb89ffe2ffd

itk-demo-configdb: tag detail page



itk-demo-configdb: Resources

- **GitLab repositiories:**
  - itk-demo-configdb
    - https://gitlab.cern.ch/atlas-itk-pixel-systemtest/itk-demo-sw/itk-demo-configdb
  - database-api backend
    - https://gitlab.cern.ch/atlas-itk-pixel-systemtest/itk-demo-sw/pypi-packages/database-api
- **Documentation:**
  - https://demi.docs.cern.ch/itk-demo-configdb/
  - https://demi.docs.cern.ch/deployments/readout-stack/

itk-demo-resultviewer

- Microservice for displaying histograms produced by scans
  - Displays results of scans saved to the configDB
  - Uses Jsroot to plot the histograms -> Fully interactive
  - Two panels to search for scans:
    - Runkey panel: Hierarchical tree view, list scans per runkey
    - AllResults panel: Flat list showing all scans saved in the configDB
  - Additional Jsroot panel: Users can upload their own histograms

- Current status:
  - Integration with new configDB read endpoints finished, all panels fully functional
  - Runkey panel needs graphical overhaul (currently difficult to navigate for large runkeys)

Maren Stratmann

Resultviewer GUI, AllResults panel

Results per Runkey | All results from database | Display .root files

Result Viewer

Config DB connected

Display the histograms from scan results stored in the database

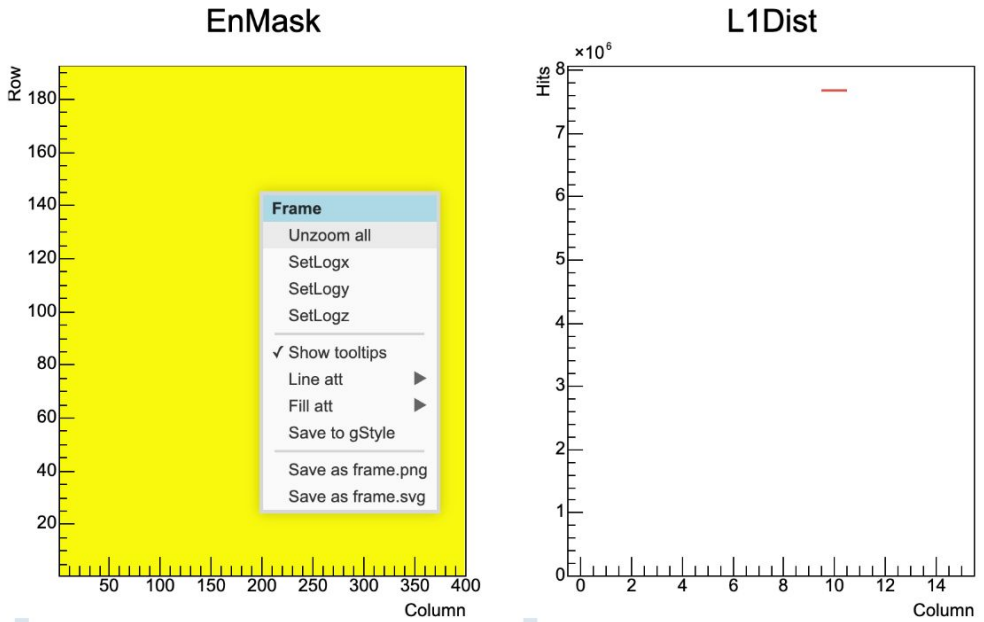
Scan ID | Scan Type

1 - 3 of 3 | 1 of 1

Runkey	Scan ID	Scan Type	Date
runkey_name	66b895ae12da48cbacbd96d3115d0c53	DigitalScan	2023-08-30 09:02
-	ac76f2c36d0f4940b72d944f9f6d9b71	DigitalScan	2023-08-30 09:06
rrrrrr	da8619d7547542a2a14d60a67d24e114	DigitalScan	2023-08-30 10:54

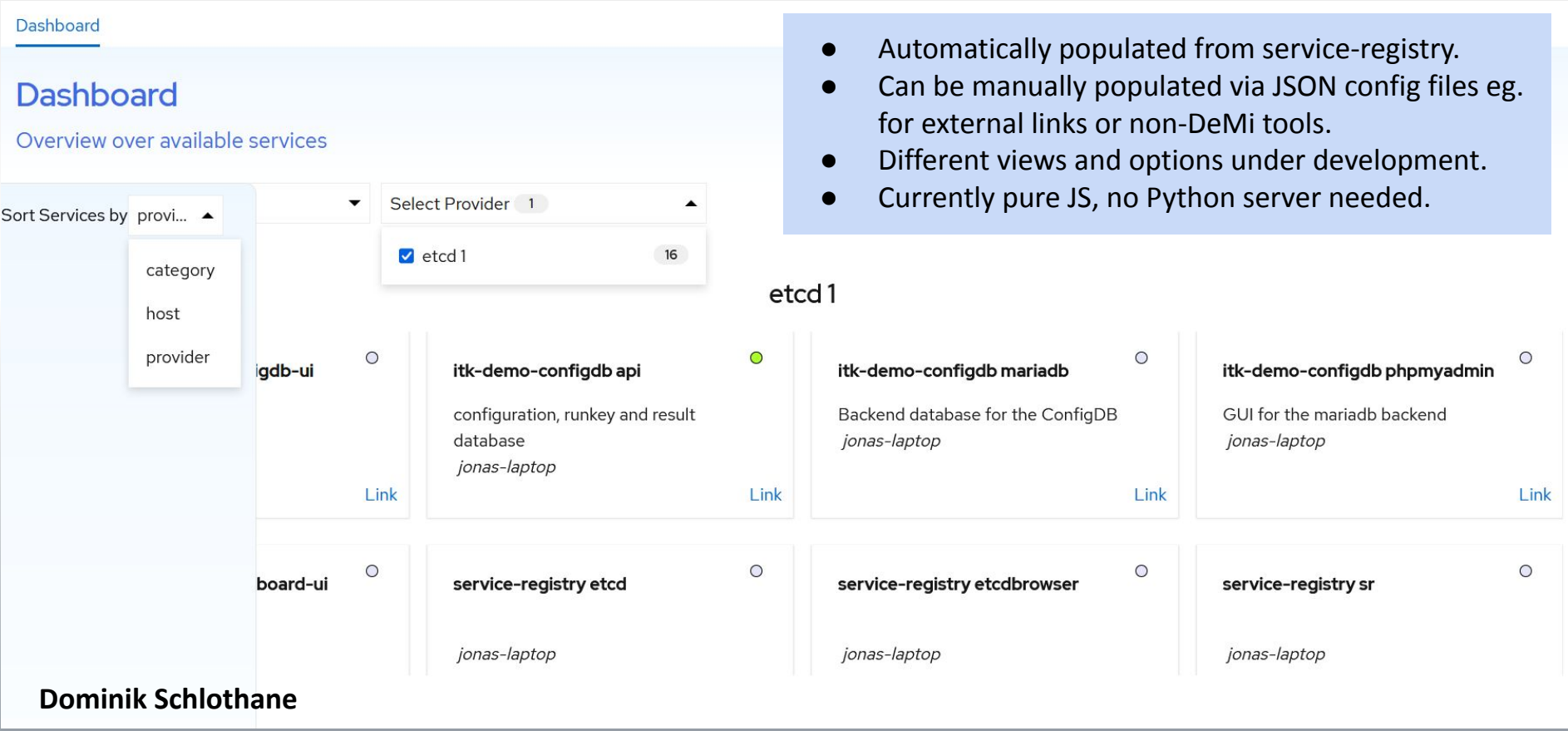
- [x] Enable Mask
- [x] L1 Distribution
- [ ] Occupancy Map

Results View



Maren Stratmann

Itk-demo-dashboard: sidebar menu with sorting options



- Automatically populated from service-registry.
- Can be manually populated via JSON config files eg. for external links or non-DeMi tools.
- Different views and options under development.
- Currently pure JS, no Python server needed.

Dominik Schlothane

Unifying Image Configuration

https://gitlab.cern.ch/groups/atlas-itk-pixel-systemtest/itk-demo-sw/-/wikis/repository/config

- Unify configuration of services across all microservice repos
- Hierarchy of key-value parameters passed to server / backend

(high to low, not all levels shown)

1. command-line parameters
2. service-registry
3. configuration File (JSON / YAML, TOML, …)
4. environment variables
5. Docker labels
6. dotenv file (.env)
7. baked-in / hardcoded value
8. [...]

Unifying Configuration

- **config-checker**
  - Hierarchical configuration loader and checker in Python
  - Wrapper around **ConfZ**
  - **ConfZ** is a wrapper around Pydantic
- Support environment variables, dotenv, service-registry and file (YAML/JSON/TOML).
  - To be added: OKS ;-)
- Examples for all cases provided in repo.
- Use config-checker for uniform configuration behavior across microservice repos
  - Implementation in progress, done for configdb and daqapi

https://gitlab.cern.ch/atlas-itk-pixel-systemtest/itk-demo-sw/pypi-packages/config-checker

https://github.com/Zuehlke/ConfZ

zühlke empowering ideas

https://github.com/pydantic/pydantic

Pydantic
Bringing schema and sanity to your data

Adrian Miemczyk

Resources

- Docs — demi.docs.cern.ch
- Meeting — ~Weekly Wednesday 1700 CEST.
- Egroup — **atlas-itk-demo-sw-devel**
- Mattermost — https://mattermost.web.cern.ch/itkpixel
  - /itk-demo-sw-users
  - /itk-demo-sw-devel
- GitLab — https://gitlab.cern.ch/atlas-itk-pixel-systemtest/itk-demo-sw
  - also hosts Wiki, Issues
  - PyPi and npm package registries.
  - OCI image registry

https://gitlab.cern.ch/groups/atlas-itk-pixel-systemtest/itk-demo-sw/-/wikis/resources