

Run Control Notes & Discussions

10 feb 2021

structure

1. TDR excerpts
 - a. Useful mental handlebars (during discussions)
2. Technical proposal
3. Mappings from TDR goals to technical proposal
4. Demo

Main challenges

- First, the high overall experiment uptime goal requires DAQ to be stringently designed for reliability, fault tolerance, and redundancy, criteria that aim to reduce overall downtime.

- Second, the system must be able to evolve to accommodate newly commissioned sub-components as they are installed into a detector module that is under construction. The DAQ must also continue to service existing modules that are operational while simultaneously accommodating subsequent detector modules as they are installed and commissioned. To support this ongoing variability, the DAQ will support operating as multiple independent instances or partitions.

Partitioning will also be supported within a single detector module for special calibration or debugging runs that are incompatible with physics data taking, while the rest of the detector remains in physics data taking mode. Partitioning, i.e., allowing several instances of the DAQ to operate independently with different configurations on different parts of the detector, will also be important during the installation and commissioning, so experts can work in parallel, e.g., for photon detectors (PDs) and TPC.

detector components participating to data taking. It provides a central access point for the highly distributed DAQ components, allowing them to be treated and managed as a single, coherent system, though their corresponding subsystem interfaces. It is responsible for error handling and recovery, which is achieved by designing a robust and autonomous fault-tolerant control system.

monitoring subsystem (CCM). The access subsystem is responsible for the action delegation to internal function calls and procedures. Its implementation is driven by the control, configuration and

monitoring interface specifications, and protects the direct access to detector and infrastructural resources. It also controls authentication and authorization, which locks different functionalities to certain actor groups and subsystems. As an example, only the detector experts can modify front-end configuration through the configuration interfaces, or only an expert user can exclude an APA's readout from data taking.

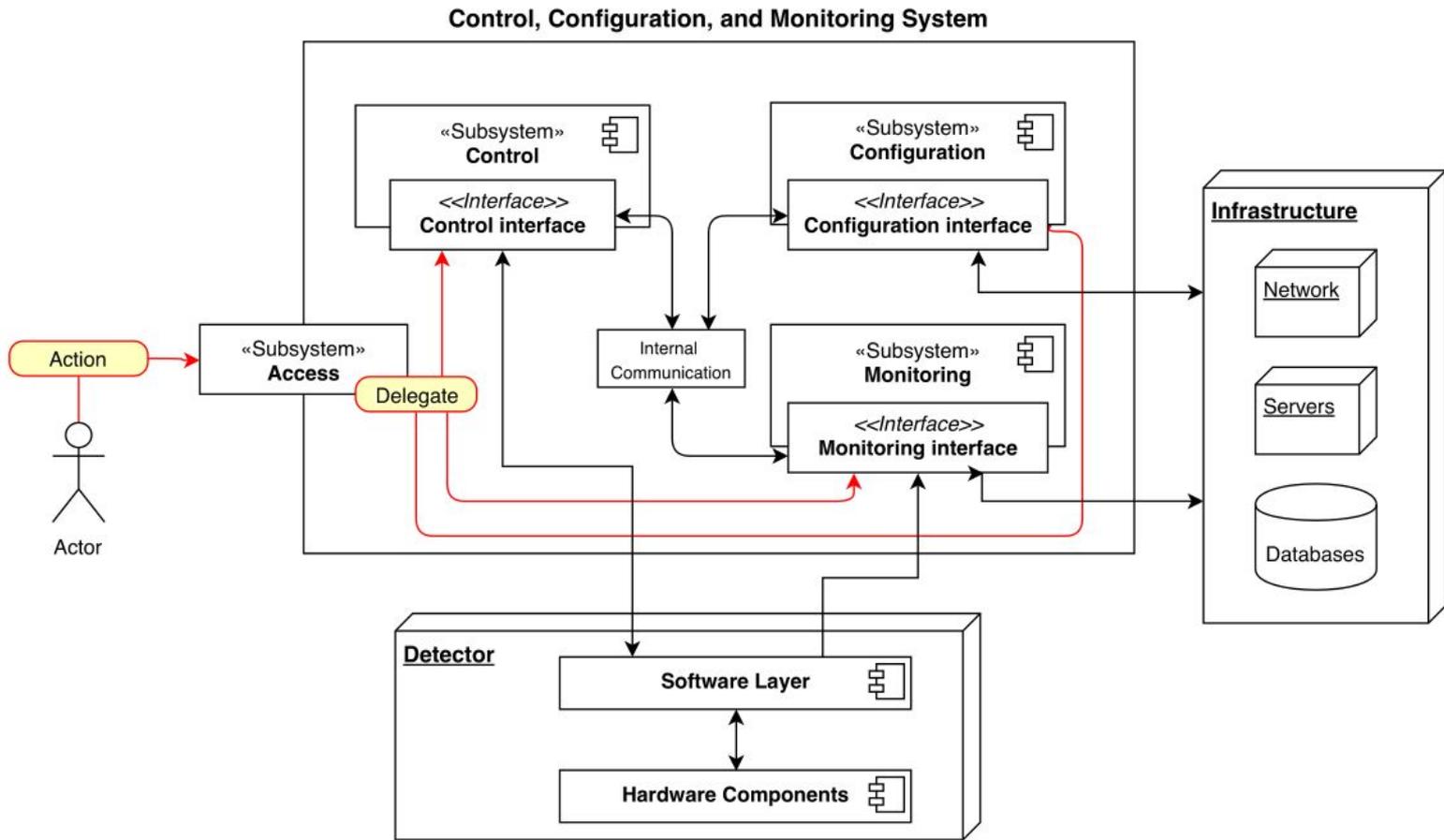


Figure 7.9: Main interaction among the three CCM subsystems.

Control

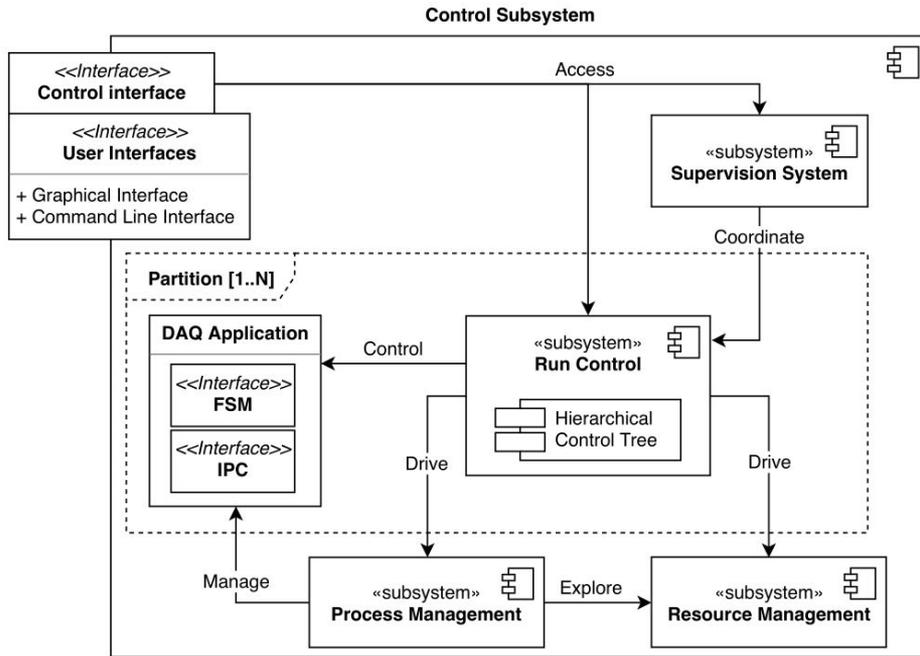


Figure 7.10: Roles and services that compose the DAQ control subsystem.

- **Supervision System** - It is responsible for manual and automated control and supervision of DAQ components at any given time. In autonomous mode, the system makes attempts for fault-recovery, failover to backup instances of subsystems, and isolation of problematic regions of the control tree. This is carried out by a hierarchical rule-based planning or fuzzy logic system.
- **DAQ Application** - The CCM provides interfaces in order to communicate with processes of the DAQ, and the ability to control and communication with the CCM. The Inter Process Communication (IPC) supports a mechanism to interact with all actors participating to data taking. The Finite State Machine (FSM) enforces the possible states and transitions that are specific to the experiment's components, and also describes them in a uniform way.
- **Run Control** - This part of the control subsystem coherently steers the data taking operations. It interacts with all actors participating to data taking in a given partition. It consists of a hierarchical control tree, which can subdivide the DAQ components into separated regions that may be acted upon independently.
- **Resource Management** - It provides a global scope of available resources for the DAQ components. This includes the mapping between the detector front-end readout units, processes, servers where they are spawned and required resources for the processes.
- **Process Management** - It is responsible for managing process lifetime.

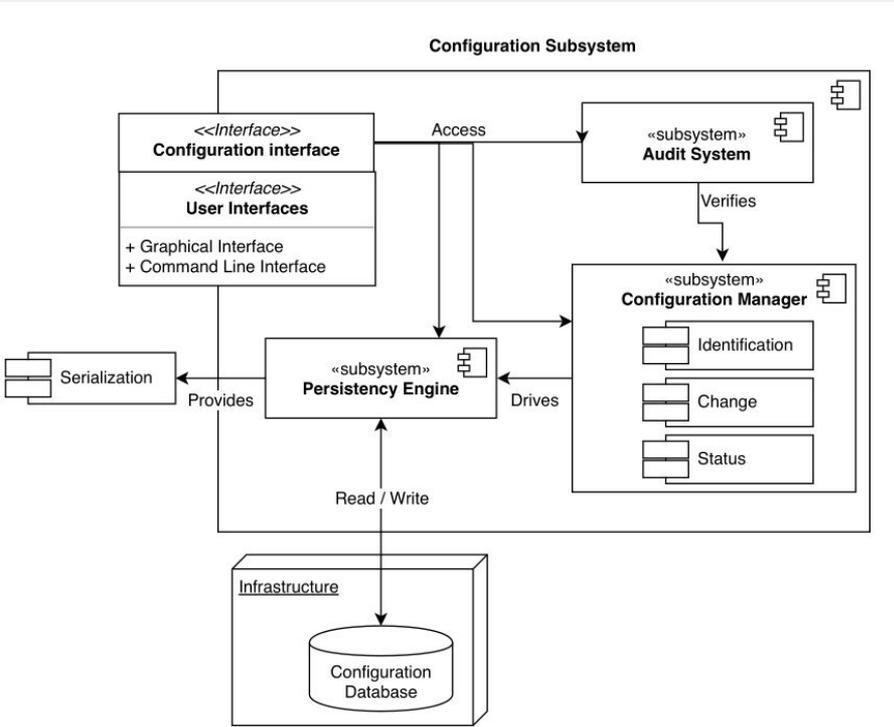
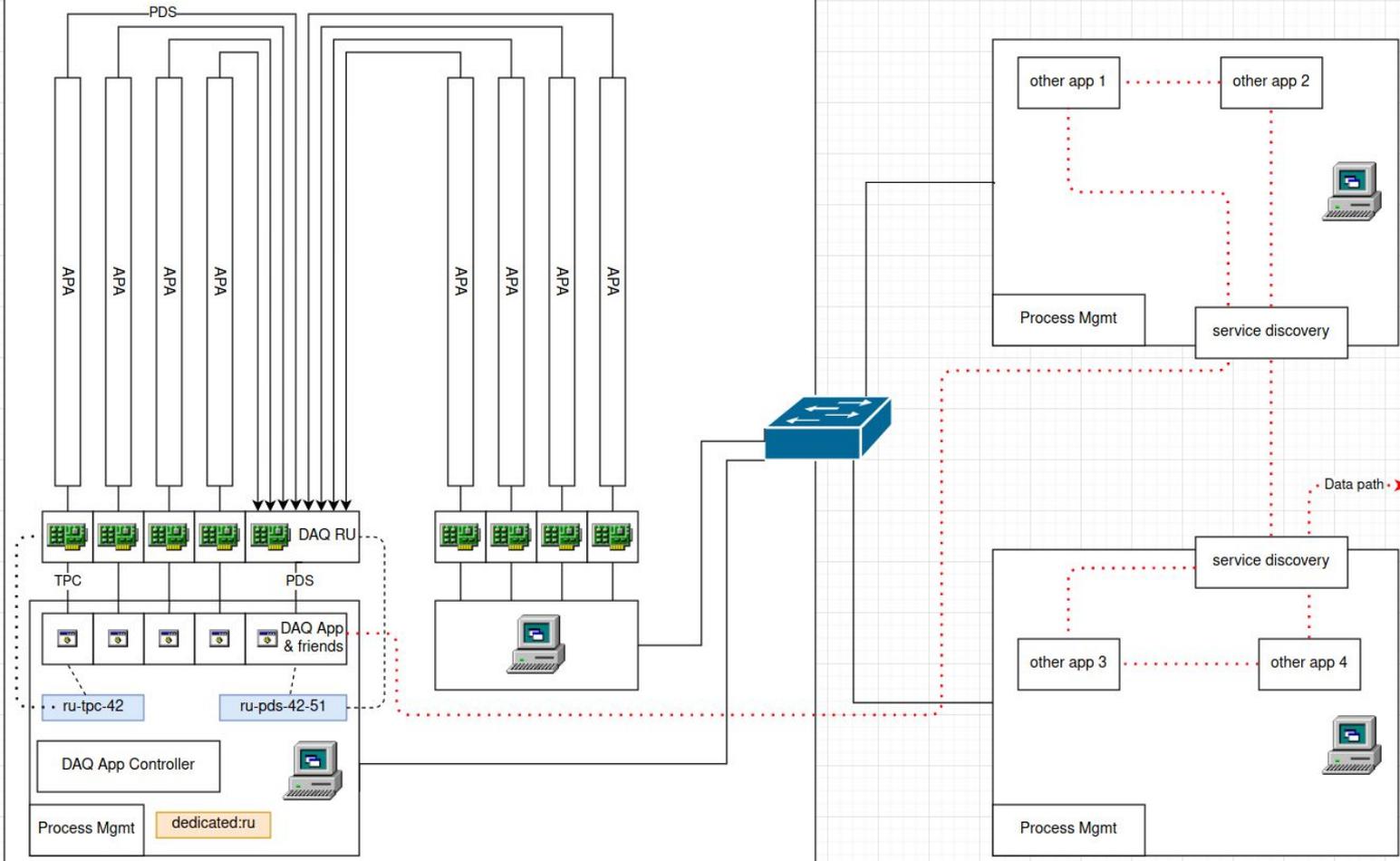


Figure 7.11: Main components of the CCM configuration subsystem.

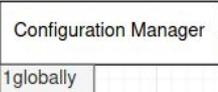
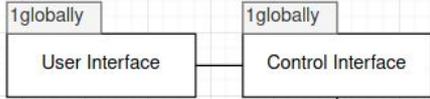
- Configuration Manager - It consists of the three main components of configuration management systems. The Identification Engine is a set of functionalities that are responsible for the definition of DAQ components and their corresponding configuration specification. The Change Manager is responsible for providing control over altering the configuration specifications of components. The Status Engine is providing status and information about configuration specifications of individual, or set of DAQ elements.
- Audit System - This important subsystem is supporting the experts and decision making systems to verify the consistency of configuration specifications against the DAQ and detector components. It provides results on mis-configurations and potential problems on configuration alignment and dependencies between components.
- Persistency Engine - This component provides a single and uniform serialization module, which is strictly followed by every DAQ component. Also responsible for configuration schema evolution and communication with the configuration database. The storage engine privileges will be only read and write operations, not allowing updates and removal of configurations. It also provides a redundant session layer for high-availability and load distribution.

10kt module, contains 150 APAs, 75 DAQ RUs



Control Interface

Provide access to config.
Dictates who can change runconfig
when and why.

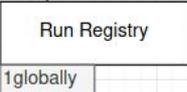


Config Mgmt

Responsible for keeping system
desired state, enforce/evolve
schemas, audit config changes

Run Control

Is configured with run number
(optional) and list of APAs
(groups?) to manage.

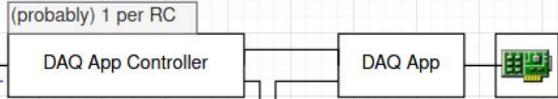


Run Registry

Historical data of run configs.

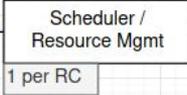
DAQ App Ctrl

Is configured with a desired state
and config for a DAQ App. Sends
needed commands to reach that
state.



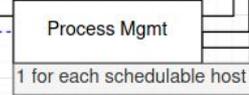
Scheduler

Receives a list of APAs.
Translates it to host:felix_address
and manages creation and
upkeeping of DAQ Apps there



Process Mgmt

Receives a machine-local set of
(DAQ) applications supposed to
be running. Manages creation and
upkeeping.



Process Management

- Process Management - It is responsible for managing process lifetime.

Goal: Keep track of application health

- “Application” is generic, can be anything (non-DAQ apps)
- Because we keep things generic, we can use **existing solutions**
 - ↳ E.g. SupervisorD, Nomad, SystemD, ...
- DAQ specific management -> Application Manager

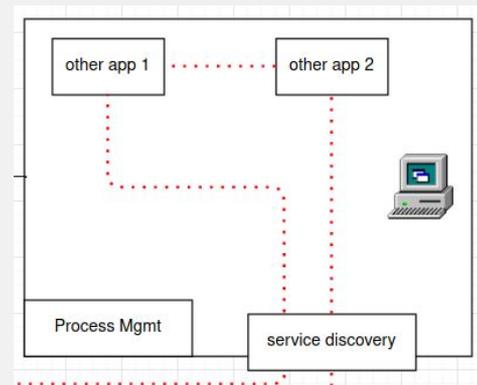
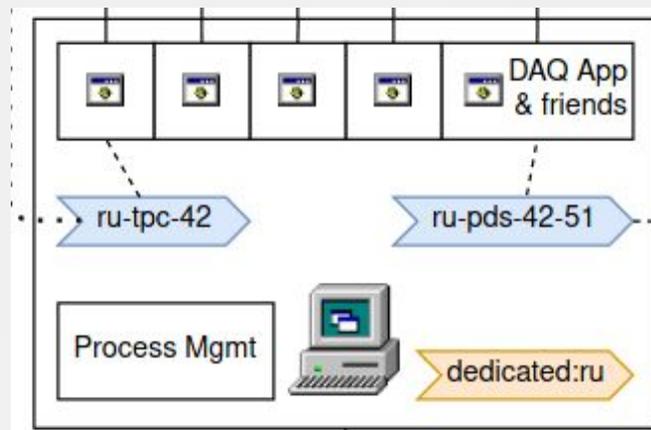
Lives close to the actual applications

- Minimize time to detect and act on anomalies
- Prevent network from causing failures at this fundamental level

Hence, instance of Process Management runs at every schedulable entity

Manages

- Application lifecycle
- Helper processes (e.g. local log collector)
- First-Aid failure procedures
- Basic health checks
- Basic performance checks
- Resource Quota enforcement



DAQ Application Manager

Supplements Process Manager

- Provides DAQ-specific process management (the DAQ state machine)

Goals: given a (list of) application(s) and their desired state

- perform needed state transitions to achieve desired state.

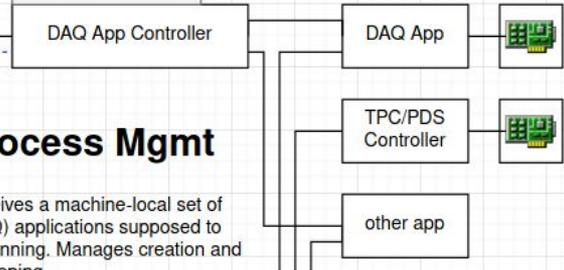
Manages

- DAQ-specific lifecycle
- First-Aid failure procedures
- Basic health checks

DAQ App Ctrl

Is configured with a desired state and config for a DAQ App. Sends needed commands to reach that state.

(probably) 1 per RC



Process Mgmt

Provides a machine-local set of applications supposed to be running. Manages creation and termination.

Scheduler

- Resource Management - It provides a global scope of available resources for the DAQ components. This includes the mapping between the detector front-end readout units, processes, servers where they are spawned and required resources for the processes.

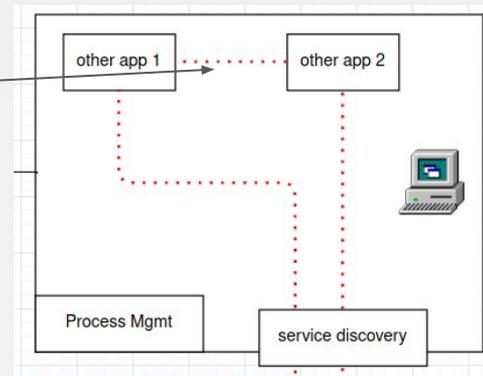
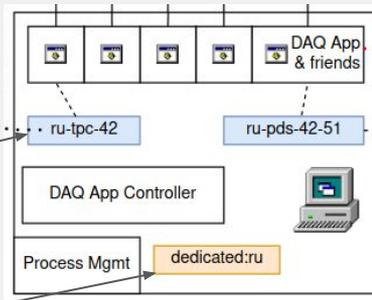
Scheduling is hard

Discussed here: <https://youtu.be/NVl9cIiPF80?t=101>

Very recommended to **build on existing work**

Implemented using various kinds of **labels**

- Resource labels
 - ↳ “This application is meant for board <id>”
 - ↳ “This application needs <amount> of <resource>”
- Dedication labels
 - ↳ Only apps meant for <label> are to be scheduled here
- Labels can be hard or soft
 - ↳ “I *must* have a GPU, I *prefer* to have the latest one”
 - ↳ “I *prefer* to have this other app closeby”



Guaranteed: No board can be scheduled to more than one application at a time

Assumption: No board can be used by two partitions at once

- Restriction could be lifted, but scheduling safeties would be lost

Run Control

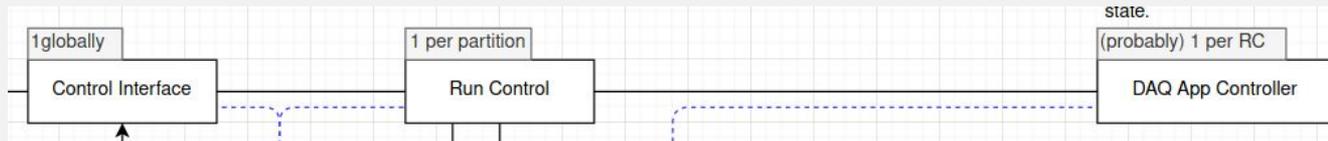
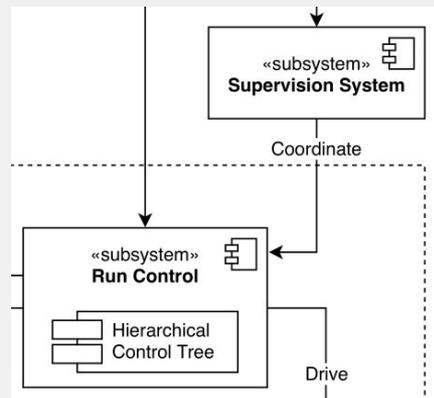
Goals:

- Given a desired run description, create needed resources
 - ↳ DAQ applications at the right places
 - ↳ Configs
 - * (for now) merely pass-through (or very close)
- Determine overall state based on performance of sub-resources
- Fault-recovery, or at least try

Works at partition-level

- Supervision System - It is responsible for manual and automated control and supervision of DAQ components at any given time. In autonomous mode, the system makes attempts for fault-recovery, failover to backup instances of subsystems, and isolation of problematic regions of the control tree. This is carried out by a hierarchical rule-based planning or fuzzy logic system.

- Run Control - This part of the control subsystem coherently steers the data taking operations. It interacts with all actors participating to data taking in a given partition. It consists of a hierarchical control tree, which can subdivide the DAQ components into separated regions that may be acted upon independently.



Interface

For now: CLI access

Future:

- CLI and REST API access
- Web GUI(s)

```
$ runcontrol <command> [...subcommand] [parameters]
```

```
$ runcontrol login
```

```
$ runcontrol config apply --file ./mystuff.json
```

```
ERROR: missing permissions to change <jsonpath>
```

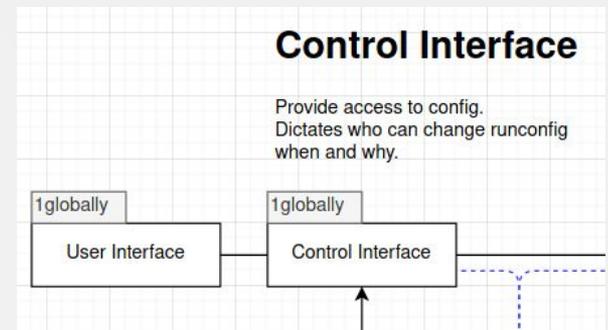
```
$ runcontrol exclude --by-id ru-tpc-42
```

```
ERROR: ru-tpc-42 is not used in any partition
```

```
$ runcontrol config get partitions
```

NAME	STATUS	RUN-NUMBER
main-runcontrol	Running	1
gldirx-calibrations	Failed	56

- Configuration Manager - It consists of the three main components of configuration management systems. The Identification Engine is a set of functionalities that are responsible for the definition of DAQ components and their corresponding configuration specification. The Change Manager is responsible for providing control over altering the configuration specifications of components. The Status Engine is providing status and information about configuration specifications of individual, or set of DAQ elements.
- Audit System - This important subsystem is supporting the experts and decision making systems to verify the consistency of configuration specifications against the DAQ and detector components. It provides results on mis-configurations and potential problems on configuration alignment and dependencies between components.
- Persistency Engine - This component provides a single and uniform serialization module, which is strictly followed by every DAQ component. Also responsible for configuration schema evolution and communication with the configuration database. The storage engine privileges will be only read and write operations, not allowing updates and removal of configurations. It also provides a redundant session layer for high-availability and load distribution.



to certain actor groups and subsystems. As an example, only the detector experts can modify front-end configuration through the configuration interfaces, or only an expert user can exclude an APA's readout from data taking.

Configuration Management

Goals

- Receive desired configuration
 - ↳ Perform validations, rbac, ...
- Deliver relevant configuration to sections in system
- Receive status info on desired configuration sections

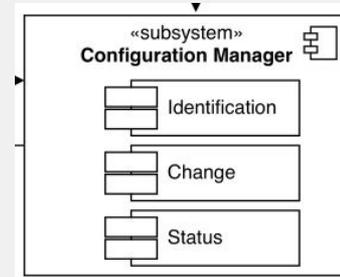
Welcome to state distribution

Welcome to hell

<https://www.youtube.com/watch?v=fE2KDzZaxvE>

Highly recommended to use existing solutions

- Configuration Manager - It consists of the three main components of configuration management systems. The Identification Engine is a set of functionalities that are responsible for the definition of DAQ components and their corresponding configuration specification. The Change Manager is responsible for providing control over altering the configuration specifications of components. The Status Engine is providing status and information about configuration specifications of individual, or set of DAQ elements.
- Audit System - This important subsystem is supporting the experts and decision making systems to verify the consistency of configuration specifications against the DAQ and detector components. It provides results on mis-configurations and potential problems on configuration alignment and dependencies between components.
- Persistency Engine - This component provides a single and uniform serialization module, which is strictly followed by every DAQ component. Also responsible for configuration schema evolution and communication with the configuration database. The storage engine privileges will be only read and write operations, not allowing updates and removal of configurations. It also provides a redundant session layer for high-availability and load distribution.



Configuration Management & Distribution: RAFT

What

- manages source of truth of system state (all configs from aforementioned apps, all partitions)
- Running on all RC machines

Why

- Complex, error prone networking code contained
 - ↳ Apps update/receive config using local librarian, librarian propagates config
 - ↳ No duplicated code of critical aspect of RC: networking
- Reliability
 - ↳ If app crashes, a restart will restore working operation much quicker
 - * Just resume comparing current state with desired state, no desired state recovery
 - ↳ If one or more control-plane apps crash
 - * System might partially make forward progress (manage DAQ App states but be unable to schedule anew)
 - * Worst case: whole system goes in read-only mode
 - ★ No impact on deadtime (if root cause only applies to RC)
 - * The above also applies to nasty problems in CAP theorem (e.g. split brain)
- Entire state known to all participants
 - ↳ No need to figure out what to get from whom
 - * Smarter controllers

Caution

- Extra moving piece
- Off the shelf solutions need to be tuned for RC

What's next

Implement prototype

- Demo of 2 DAQ apps running under run control

Functional demo exists today

- Real 'daq_application's
- Items from previous slides either implemented or have a logical implementation path
- Pathway to other interesting concepts (reliability engineering, operators, ...)

Develop further and get into 'real' setup asap

- Deploy to testing area
- Resilience testing (e.g. chaos days)
- Reflect on choices

```
→ controllers git:(main) x run-control get partitions
NAME          RUN-NUMBER  STATUS
partition-sample  5          happy
→ controllers git:(main) x
```

Config cascade

```
1  apiVersion: rc.ccm.dunescience.org/v0alpha0
2  kind: Partition
3  metadata:
4    name: partition-sample
5  spec:
6    runNumber: 5
7    configName: my-first-daq-config
8    resources:
9      - module: A # one of the four 10kt module names
10      TPC:
11        APAs:
12          - "42"
13          - "43"
14        PDS:
15          APAs:
16            - 42-51 # APAs are grouped for PD readout
17
```

```
→ controllers git:(main) x run-control get partitions
NAME                RUN-NUMBER  STATUS
partition-sample    5           happy
→ controllers git:(main) x
```

```
→ controllers git:(main) x run-control get DAQApplications
NAME                DAQ STATE  STATUS
ru-42-51-apa-pds-a  STARTED    happy
ru-42-apa-tpc-a     STARTED    happy
ru-43-apa-tpc-a     STARTED    happy
```

```
→ controllers git:(main) x run-control get pods
NAME                READY  STATUS   RESTARTS  AGE
dev                 1/1    Running  0          6h38m
ru-42-51-apa-pds-a-6cb8896bc5-tdqt2  1/1    Running  0          12m
ru-42-apa-tpc-a-7788c8c8cc-wt281     1/1    Running  0          13m
ru-43-apa-tpc-a-99475c6f9-7nb49     1/1    Running  0          13m
→ controllers git:(main) x □
```

Real DAQ Apps

```
→ controllers git:(main) x kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
dev	1/1	Running	0	6h48m
ru-42-51-apa-pds-a-6cb8896bc5-v9w5f	1/1	Running	0	8m34s
ru-42-apa-tpc-a-7788c8c8cc-dfg26	1/1	Running	0	8m34s
ru-43-apa-tpc-a-99475c6f9-jhdtk	1/1	Running	0	8m34s

```
→ controllers git:(main) x ps -aux | grep daq_application
```

root	33068	0.0	0.0	1053300	9724 ?	S1	14:36	0:00	./pretender -p 80 daq_application -c stdin:///mnt/minidaq_config.json
root	33074	91.8	1.7	2212120	572092 ?	SL1	14:36	7:34	daq_application -c stdin:///mnt/minidaq_config.json
root	35790	0.0	0.0	986356	9636 ?	S1	14:36	0:00	./pretender -p 80 daq_application -c stdin:///mnt/minidaq_config.json
root	35799	90.3	1.7	2212120	571980 ?	SL1	14:36	7:18	daq_application -c stdin:///mnt/minidaq_config.json
root	36079	0.0	0.0	855284	8980 ?	S1	14:36	0:00	./pretender -p 80 daq_application -c stdin:///mnt/minidaq_config.json
root	36089	0.0	0.0	868584	27144 ?	SL1	14:36	0:00	daq_application -c stdin:///mnt/minidaq_config.json
glenn	39080	0.0	0.0	6392	2344 pts/5	R+	14:44	0:00	grep --color daq_application

```
→ controllers git:(main) x
```

```
→ controllers git:(main) x kubectl logs ru-42-apa-tpc-a-7788c8c8cc-dfg26 | grep DAQModuleManager
```

```
2021-Feb-10 14:36:44,433 LOG [dunedaq::appfwk::DAQModuleManager::execute(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:225] Command id:init
2021-Feb-10 14:36:44,434 INFO [dunedaq::appfwk::DAQModuleManager::init_queues(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:711] Adding queue: data_fragments_q
2021-Feb-10 14:36:44,434 INFO [dunedaq::appfwk::DAQModuleManager::init_queues(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:711] Adding queue: data_requests_0
2021-Feb-10 14:36:44,434 INFO [dunedaq::appfwk::DAQModuleManager::init_queues(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:711] Adding queue: data_requests_1
2021-Feb-10 14:36:44,434 INFO [dunedaq::appfwk::DAQModuleManager::init_queues(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:711] Adding queue: fake_link_0
2021-Feb-10 14:36:44,434 INFO [dunedaq::appfwk::DAQModuleManager::init_queues(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:711] Adding queue: fake_link_1
2021-Feb-10 14:36:44,434 INFO [dunedaq::appfwk::DAQModuleManager::init_queues(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:711] Adding queue: time_sync_q
2021-Feb-10 14:36:44,434 INFO [dunedaq::appfwk::DAQModuleManager::init_queues(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:711] Adding queue: trigger_decision_copy_for_bookkeeping
2021-Feb-10 14:36:44,434 INFO [dunedaq::appfwk::DAQModuleManager::init_queues(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:711] Adding queue: trigger_decision_copy_for_inhibit
2021-Feb-10 14:36:44,434 INFO [dunedaq::appfwk::DAQModuleManager::init_queues(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:711] Adding queue: trigger_decision_q
2021-Feb-10 14:36:44,434 INFO [dunedaq::appfwk::DAQModuleManager::init_queues(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:711] Adding queue: trigger_inhibit_q
2021-Feb-10 14:36:44,434 INFO [dunedaq::appfwk::DAQModuleManager::init_queues(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:711] Adding queue: trigger_record_q
2021-Feb-10 14:36:44,434 INFO [dunedaq::appfwk::DAQModuleManager::init_modules(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:351] construct: TriggerDecisionEmulator : tde
2021-Feb-10 14:36:44,517 INFO [dunedaq::appfwk::DAQModuleManager::init_modules(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:351] construct: RequestGenerator : rgg
2021-Feb-10 14:36:44,530 INFO [dunedaq::appfwk::DAQModuleManager::init_modules(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:351] construct: FragmentReceiver : ffr
2021-Feb-10 14:36:44,539 INFO [dunedaq::appfwk::DAQModuleManager::init_modules(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:351] construct: DataWriter : datawriter
2021-Feb-10 14:36:44,543 INFO [dunedaq::appfwk::DAQModuleManager::init_modules(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:351] construct: FakeCardReader : fake_source
2021-Feb-10 14:36:44,615 INFO [dunedaq::appfwk::DAQModuleManager::init_modules(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:351] construct: DataLinkHandler : datahandler_0
2021-Feb-10 14:36:44,625 INFO [dunedaq::appfwk::DAQModuleManager::init_modules(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:351] construct: DataLinkHandler : datahandler_1
2021-Feb-10 14:36:44,737 LOG [dunedaq::appfwk::DAQModuleManager::execute(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:225] Command id:conf
2021-Feb-10 14:36:44,740 LOG [dunedaq::appfwk::DAQModuleManager::dispatch_one_match_only(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:204] Executing conf -> tde
2021-Feb-10 14:36:44,740 LOG [dunedaq::appfwk::DAQModuleManager::dispatch_one_match_only(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:204] Executing conf -> rgg
2021-Feb-10 14:36:44,740 LOG [dunedaq::appfwk::DAQModuleManager::dispatch_one_match_only(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:204] Executing conf -> ffr
2021-Feb-10 14:36:44,817 LOG [dunedaq::appfwk::DAQModuleManager::dispatch_one_match_only(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:204] Executing conf -> datawriter
2021-Feb-10 14:36:44,827 LOG [dunedaq::appfwk::DAQModuleManager::dispatch_one_match_only(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:204] Executing conf -> fake_source
2021-Feb-10 14:36:44,829 LOG [dunedaq::appfwk::DAQModuleManager::dispatch_one_match_only(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:204] Executing conf -> datahandler_0
2021-Feb-10 14:36:44,830 LOG [dunedaq::appfwk::DAQModuleManager::dispatch_one_match_only(...) at /scratch/dingpf/dunedaq-v2.2.0-prep/workdir/sourcecode/appfwk/src/DAQModuleManager.cpp:204] Executing conf -> datahandler_1
→ controllers git:(main) x
```

Containerization & Device access

As for access:

- Privileged containers have all device access like running outside a container
 - ↳ Vendor plugins merely give this kind of access without needing privileged pods
- Tested
 - ↳ File-descriptor based access, memory mapping (DMA) work
 - ↳ Verified locally using bare C (but not Physics boards, I don't have access)
 - ↳ Fd based access verified with CMS colleagues

As for performance:

- Underlying mechanisms provided by kernel
 - ↳ Process isolation, filesystem layers, ...
- Network is not software emulated
 - ↳ Again leverages kernel, no packet encapsulation (unless you really want to...)
 - ↳ <https://docs.projectcalico.org/about/about-calico>

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: daq-app-42
  labels:
    app: daq
    apa: "42"
spec:
  selector:
    matchLabels:
      app: daq
      apa: "42"
  replicas: 1
  template:
    metadata:
      labels:
        app: daq
        apa: "42"
    spec:
      tolerations:
        - key: dedicated
          operator: Equal
          value: apa
          effect: NoSchedule
      containers:
        - name: daq
          image: nginx
          ports:
            - containerPort: 80
          resources:
            requests:
              rc.ccm/ru-42-apa-tpc-a: 1
            limits:
              rc.ccm/ru-42-apa-tpc-a: 1
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: daq-app-42-2
  labels:
    app: daq
    apa: "42"
spec:
  selector:
    matchLabels:
      app: daq
      apa: "42"
  replicas: 1
  template:
    metadata:
      labels:
        app: daq
        apa: "42"
    spec:
      tolerations:
        - key: dedicated
          operator: Equal
          value: apa
          effect: NoSchedule
      containers:
        - name: daq
          image: nginx
          ports:
            - containerPort: 80
          resources:
            requests:
              rc.ccm/ru-42-apa-tpc-a: 1
            limits:
              rc.ccm/ru-42-apa-tpc-a: 1
```

```
→ controllers git:(main) x kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
daq-app-42-2-6d44bcbf65-g2c9s      0/1     Pending   0           81s
daq-app-42-6d44bcbf65-n94xg        1/1     Running   0           81s
dev                                  1/1     Running   0           6h56m
```

```
Events:
Type      Reason            Age           From          Message
-----
Warning   FailedScheduling  0s (x3 over 86s)  default-scheduler  0/1 nodes are available: 1 Insufficient rc.ccm/ru-42-apa-tpc-a: 1
```

Inter-pod affinity and anti-affinity (beta feature)

Inter-pod affinity and anti-affinity were introduced in Kubernetes 1.4. Inter-pod affinity and anti-affinity allow you to constrain which nodes your pod is eligible to schedule on *based on labels on pods that are already running on the node* rather than based on labels on nodes. The rules are of the form "this pod should (or, in the case of anti-affinity, should not) run in an X if that X is already running one or more pods that meet rule Y." Y is expressed as a LabelSelector with an associated list of namespaces (or "all" namespaces); unlike nodes, because pods are namespaced (and therefore the labels on pods are implicitly namespaced), a label selector over pod labels must specify which namespaces the selector should apply to. Conceptually X is a topology domain like node, rack, cloud provider zone, cloud provider region, etc. You express it using a `topologyKey` which is the key for the node label that the system uses to denote such a topology domain, e.g. see the label keys listed above in the section [Interlude: built-in node labels](#).

Using Admission Controllers

This page provides an overview of Admission Controllers.

What are they?

An admission controller is a piece of code that intercepts requests to the Kubernetes API server prior to persistence of the object, but after the request is authenticated and authorized. The controllers consist of the [list](#) below, are compiled into the `kube-apiserver` binary, and may only be configured by the cluster administrator. In that list, there are two special controllers: `MutatingAdmissionWebhook` and `ValidatingAdmissionWebhook`. These execute the mutating and validating (respectively) [admission control webhooks](#) which are configured in the API.

Admission controllers may be "validating", "mutating", or both. Mutating controllers may modify the objects they admit; validating controllers may not.

Admission controllers limit requests to create, delete, modify or connect to (proxy). They do not support read requests.

The admission control process proceeds in two phases. In the first phase, mutating admission controllers are run. In the second phase, validating admission controllers are run. Note again that some of the controllers are both.

MutatingAdmissionWebhook

This admission controller calls any mutating webhooks which match the request. Matching webhooks are called in serial; each one may modify the object if it desires.

This admission controller (as implied by the name) only runs in the mutating phase.

If a webhook called by this has side effects (for example, decrementing quota) it *must* have a reconciliation system, as it is not guaranteed that subsequent webhooks or validating admission controllers will permit the request to finish.

If you disable the MutatingAdmissionWebhook, you must also disable the `MutatingWebhookConfiguration` object in the `admissionregistration.k8s.io/v1` group/version via the `--runtime-config` flag (both are on by default in versions \geq 1.9).

extras

Oopsie manager

Receives events upon which to potentially act

- Suspicious log entry event
- alert from monitoring system
- external application event

Might be rule based

This might not be a centralized application

Modern day relative: the Operator Pattern

<https://www.youtube.com/watch?v=DhvYfNMOh6A>

- Supervision System - It is responsible for manual and automated control and supervision of DAQ components at any given time. In autonomous mode, the system makes attempts for fault-recovery, failover to backup instances of subsystems, and isolation of problematic regions of the control tree. This is carried out by a hierarchical rule-based planning or fuzzy logic system.

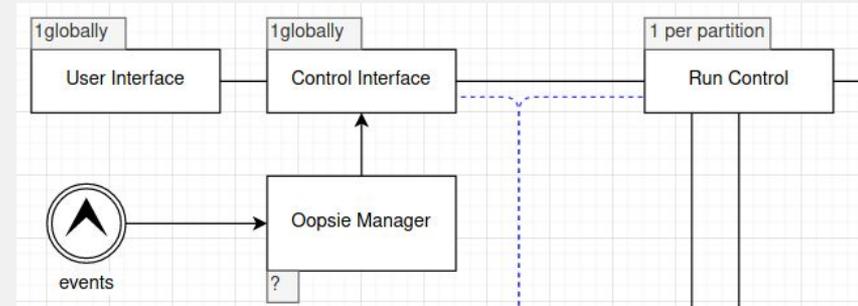


Table 7.4: Data Acquisition System Interface Links.

Interfacing System	Description	Reference
TPC CE	Data rate and format, number and type of links, timing, inherent noise	DocDB 6742 [68] v6
PDS Readout	Data rate and format, number and type of links, timing	DocDB 6727 [86] v2
Computing	Off-site data transfer rates, methods, data file content, disk buffer, software development and maintenance	DocDB 7123 [135]
CISC	Information exchange, hardware and software for rack and server monitoring	DocDB 6790 [136] v1
Calibration	Constraint on total volume of the calibration data; trigger and timing distribution from the DAQ	DocDB 7069 [122]
Timing and Synchronization	Clients, clock frequency, protocols, transports, accuracy, synchronization precision, monitoring	DocDB 11224 [137]
Facilities	Detector integration, coordination, cables, racks, safety, conventional facilities, lack of impact on cryo and DSS	DocDB 6988 [138] v1
Installation	Prototyping, planning, transport, underground equipment and activity, safety	DocDB 7015 [139]

Partition awareness

Pieces that require partition awareness

→ Run control

Others: useful to include partition info where convenient

→ Useful as metadata in logging/monitoring data

Run Controller

Manages global config

Instructs scheduler and App Controller(s)

Data:

- run number (can self generate)
- Desired APAs and modes (running/calibrating/...)

Run Registry

Keeps history of runs and configs

→ Likely to be an external system

Data (per run number):

→ Event times

- ↳ Given to run control
- ↳ Start of operations (start of term)
- ↳ End of term
- ↳ Last item removed (=start of term next run)

DAQ App Controller

Responsible for making sure a DAQ App is running & configured correctly

Data

- Run number
- Desired app config

- New desired config = restart DAQ App + init + config + start
- If run number changes, but desired config otherwise identical; if daq application supports it, we could drastically reduce new run setup time by just updating run number without destroy/startup of new DAQ App
- Easy to run locally
 - ↳ Bonus: local dev becomes easier and is representative of real operations
- Controller runs as close as possible to DAQ App
 - ↳ Very quick state polling & changes

Scheduler

Responsible for keeping all applications for a set of APAs running

Data:

- List of APAs
- Lookup table
 - ↳ What boards & hostnames are connected to which APAs

Process Controller

Responsible for keeping all applications for a host running

→ A host being any machine able to receive schedulables

Data:

→ List of applications

↳ What FELIX board it will use

API Manager

- Place where to get whole state
 - ↳ Users don't need to bother with what data to get where
- Accountability
 - ↳ SSO integration
 - ↳ Policies