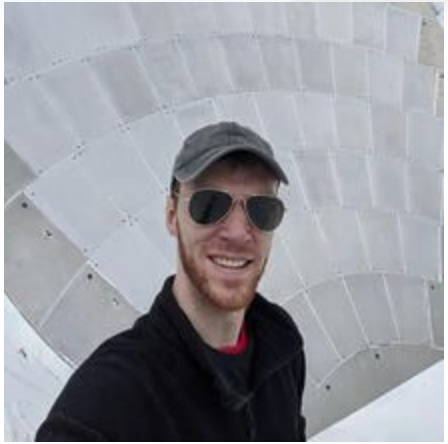
WBS 1.08.02: Observatory Control System

L3 Lead - Brian Koopman

Presenter Introduction



Name: Brain Koopman

Institution: Yale University

Discipline: Physics / Cosmology

Previous experience:

- Design and development of DAQ, monitoring, and control software systems for the Simons Observatory. Core developer of the Observatory Control System. (3 years)
- PhD working on ACT with a focus on polarization calibration, detector testing, and improving the remote observing experience through development of web based tools. (6 years)

Key Driving Requirements for 1.08.02

See Trade Study Discussion from Cosmin

Inter-L3 Interfaces within this L2

CQ2

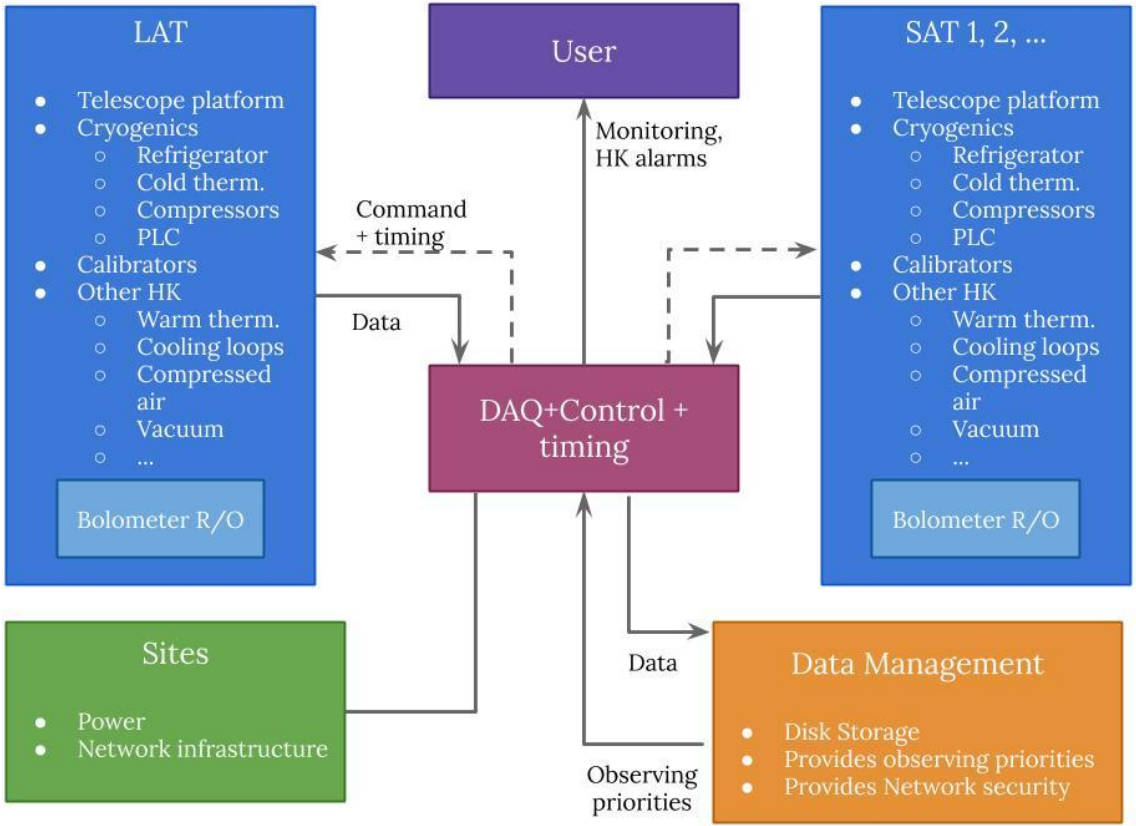
- The Observatory Control System (1.08.02) is closely tied to Observatory Data Acquisition (1.08.03).
- The control framework will also send data/be used to fetch data to/from the monitoring and alarms system (1.08.04).
- Subsystem Development and Support (1.08.05) and Deployment of DAQ and OCS (1.08.06) will both use the software developed within 1.08.02.

Overview/Scope

CQ2

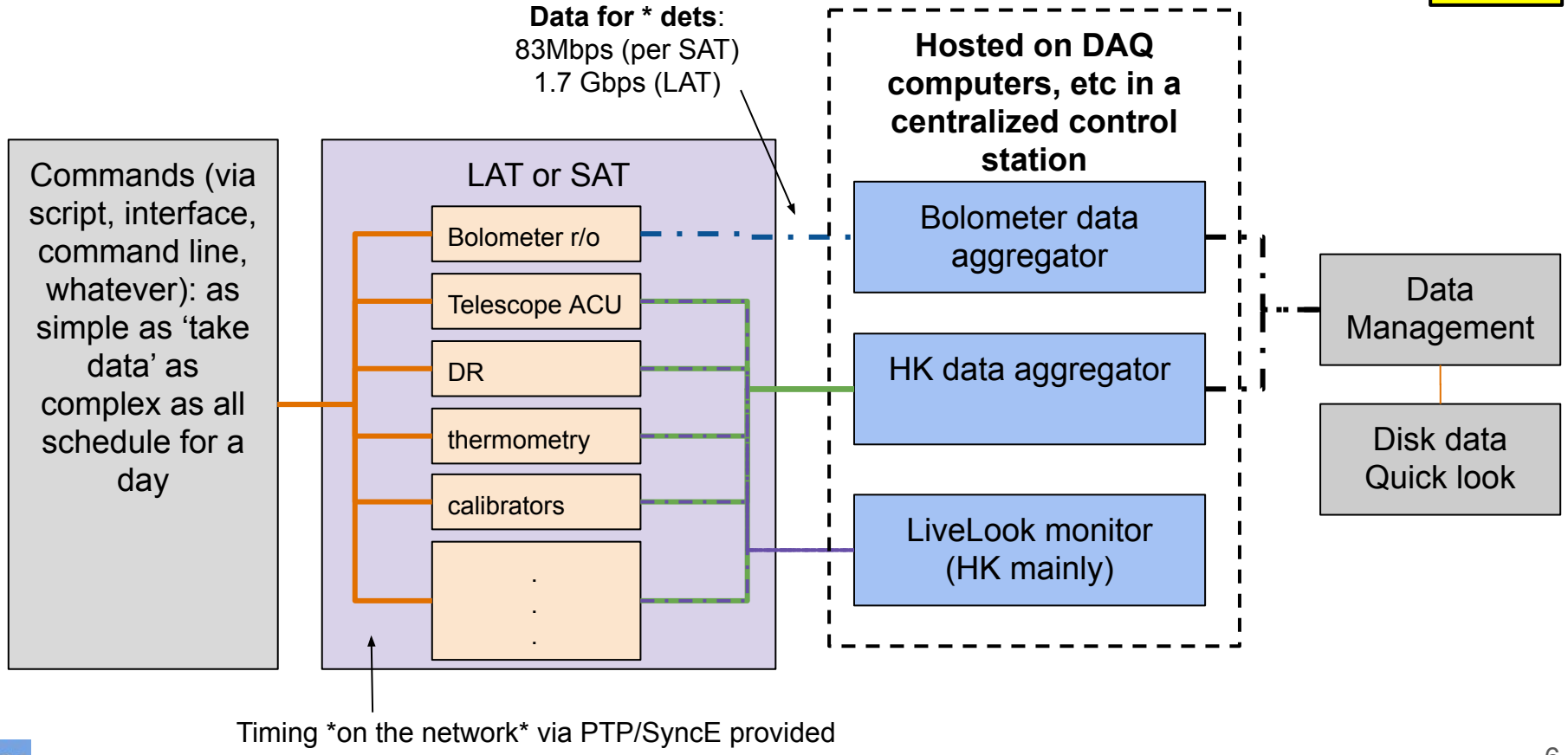
Scope:

- To deliver a control and data acquisition software package for all subsystems
- Live and historical monitoring of ‘housekeeping’ data and meta-data
- Timing
- Non-safety alarming.



Connection Scheme

CQ2



DAQ+Control Design Principles

- Be usable by all hardware groups
  - Hardware groups should prefer interacting with “real” DAQ to parallel home-grown systems -- avoids duplicated effort, gives early testing and evaluation
- Use commodity hardware to the extent possible
- Be modular:
  - Many pieces need to plug into it
  - Scale from single-lab to full Observatory, and the same for both sites (Chile, SP)
  - Allow non-DAQ personnel to write control interfaces for small hardware pieces
- Be open source, versioned, tracked, CI’d, reviewed, and distributed, common language
- The DAQ+Control interface to the hardware is digital:
  - DAQ is software-only and Ethernet/IP-based

Control Requirements flowdown

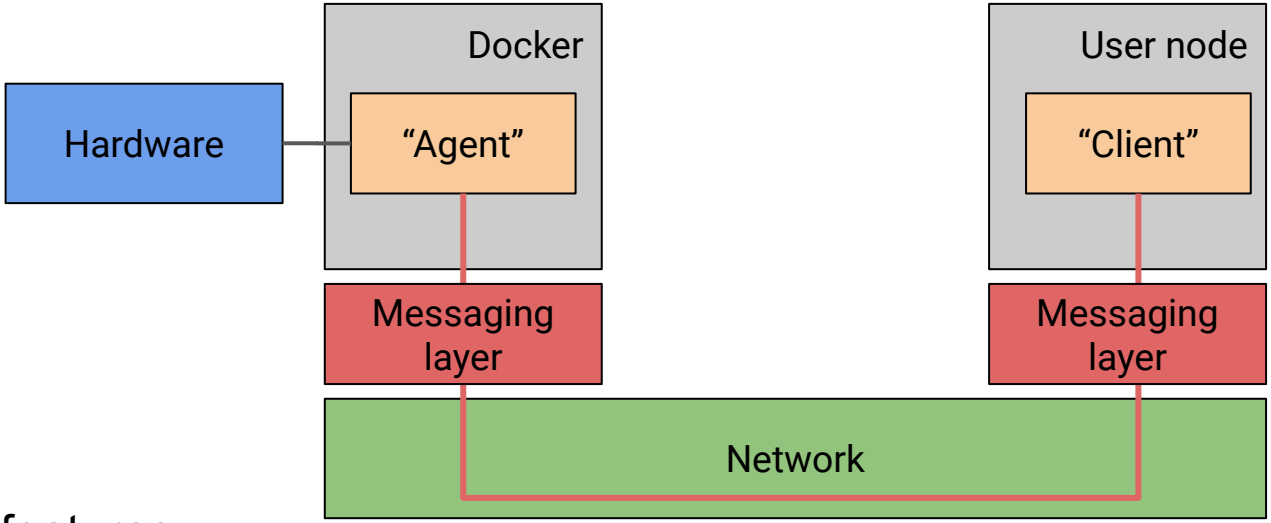
CQ1

- Support high-level command and distributed control of:
  - Detector readout including detector calibration, tuning
  - Telescope motion in a variety of modes
  - Cryogenics
  - Data acquisition start/stop, independent for all subsystems
  - Other (calibration, thermometry, heaters, hardware configuration parameters, etc)
- Support observing scheduling (containing ordered commanding of detectors, telescope, and other auxilliary components)
- Interface with telescope control provided by vendor at the level of az/el itinerary delivered by network/serial
- Support for user access control

Technical Design Goals: Control

- Use commercial or open source software and leverage current work where possible
- Must be able to run on different OSes
- Must be easily distributed and tested
- Control code for a component must be able to be developed by hardware experts.
- Common software across all test-stands and observatory
- Modular and scalable: All components of the system (control, monitoring) must be easily configurable for different labs by scientists doing the testing

Technical Design Overview: Control



Key features:

- **Assumes <u>distributed</u> devices, communicating via a <u>commercial messaging layer</u>**

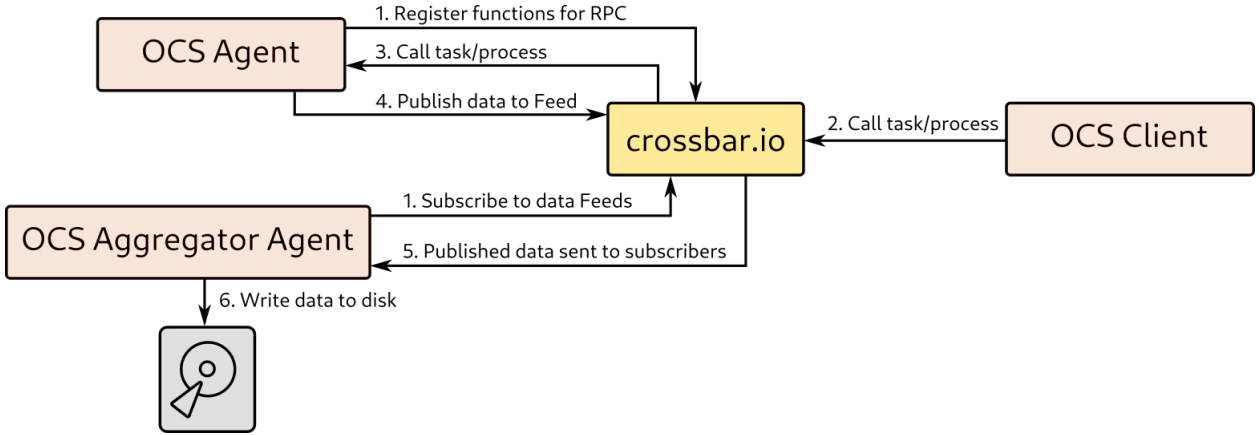
- In SO example uses WAMP protocol via crossbar.io router:
  - On the hardware side, there is a device-specific API: an agent exposes author-chosen operations from the list of available operations from the hardware vendor
  - The user-facing side is a client which uses the primitives for the exposed commands (register/calls/publishes/subscribes)

OCS Overview

- The Observatory Control System (OCS) is a distributed control system designed to coordinate DAQ across an observatory
- Design goals:
  - Easy to use
  - Easy to add additional components to for new hardware
  - Scalable from small lab deployments to full site deployment
- Mostly written in Python, with use of some open source systems
- Main messaging and data passage system is built on the open source crossbar.io platform
- Use additional open source platforms for live and historical data viewing
  - Grafana + Loki - Web based plotting and observability software, and log aggregation
  - InfluxDB - Time series database backend for Grafana

OCS Architecture

- Two main components to OCS:
  - Agents - long running software servers that interface with hardware or other software
  - Clients - scripts that orchestrate the actions of one or more OCS Agents
- Agents make available two types of commands:
  - Tasks - function that ends in finite time
  - Processes - function that runs for an open-ended amount of time, until stopped by a client



Background: Docker use case for SO

- Identified Docker as a potentially useful software early on in OCS development
- Distributed, long running servers (effectively daemons) with varying dependencies, often multiple copies of them
- Docker images developed for:
  - spt3g_software
  - so3g
  - ocs/socs
- Smoothed out deployment of Agents and their dependencies, particularly spt3g and so3g deps
- Brought installation of full DAQ system from ~several days of a computer savy person's effort down to ~1 hr
- Versions and defines a DAQ system via Docker tags and docker-compose.yaml config files
  - Enables rapid re-deployment of a system that needs replacement
    - Example: Computer died at UPenn, was able to reinstall OS on new computer and bring up old DAQ Docker stack in <30 mins
- Not much overhead, containers start up quickly, and allow for easy monitoring of system resource usage per container
- Docker networking facilities offer some nice features -- name resolution, overlay networking, isolation from host network

OCS Agents

- Small set of “core” OCS Agents - ocs repo
  - HK Aggregator Agent - writes all data from all HK Agents to disk in .g3 format
  - InfluxDB Publisher Agent - writes all data (possibly downsampled) from HK Agents to InfluxDB
  - Registry Agent - Keeps track of all other running OCS Agents on the network and the status of their tasks/processes
  - Host Master Agent - Can control startup and shutdown of Agents depending on OCS configuration
  - Fake Data Agent - Simulated data Agent used for testing
- Large set of Simons Observatory Control System (socs) Agents - socs repo
  - Provides functionality specific to SO hardware
  - Over half have been user contributed (demonstrating ease of use among labs)
  - Major Agents are functional and are being tested
    - ACU - telescope pointing
    - CHWP - control and readout of the CHWPs
    - SMuRF - control and monitoring of the detector readout crates and collecting output files

SOCS Agents

Table 1. Summary of current and in development `ocs` Agents. Core `ocs` Agents are marked by an asterisk. All other Agents listed are kept with `socs`.

Agent	Hardware/Software	Development Status	Description
AggregatorAgent*	Housekeeping Archive	In use	Saves HK data from feeds to `.g3` files in the HK archive.
BluefosAgent	Bluefors LD400 Dilution Refrigerator	In use	Parses and passes logs from LD400 software to `ocs`.
CryomechCPAAgent	Cryomech CPA Model Compressors	In use	Communicates with compressor over Ethernet to record operations statistics. On/Off control still in development.
CHWPAgent	Custom Built Cryogenic HWP	Under development	Command and control cryogenic half wave plate (CHWP) hardware.
DSEAgent	Deep Sea Electronics Controller	Under development	Collect diesel generator statistics over modbus interface.
HostMasterAgent*	`ocs` Agents	In use	Optional Agent to start and stop Agent instances. Useful for running Agents outside of Docker containers.
iBootbarAgent	iBootBar PDUs	In development	Monitor and control managed power distribution units (PDUs) for remote power cycling of electronics.
InfluxDBAgent*	Influx Database	In use	Record HK data feeds to Influx database for viewing in Grafana.
Lakeshore372Agent	Lakeshore 372 Resistance Bridge	In use	100 mK thermometer readout and heater control for dilution refrigerator operation.
Lakeshore240Agent	Lakeshore 240 Input Module	In use	1K-300K thermometer readout.
LabJackAgent	LabJack T4/T7	In use	Communicates with a LabJack over Ethernet for generic DAC. Used in warm thermometry readout.
PfiefferAgent	Pfieffer TPG 366	In use	Six channel pressure gauge controller readout over Ethernet. Used to monitor pressures within vacuum systems of the dilution refrigerators.
MeinbergM1000Agent	Meinberg LANTIME M1000	In use	Monitor health of Meinberg M1000 timing system via the Simple Network Management Protocol (SNMP).
RegistryAgent*	`ocs` Agents	Under development	Tracks the state of other running `ocs` Agents on the network.
SCPIPSUAgent	SCPI Compatible PSUs	In use	Command and control power supplies compatible with Standard Commands for Programmable Instruments communication protocol.
SynaccessAgent	Synaccess PDUs	In use	Monitor and control managed PDUs for remote power cycling of electronics.
UPSAgent	UPS Battery Backups	In development	Monitor state of SNMP compatible UPS battery backups.

OCS Clients

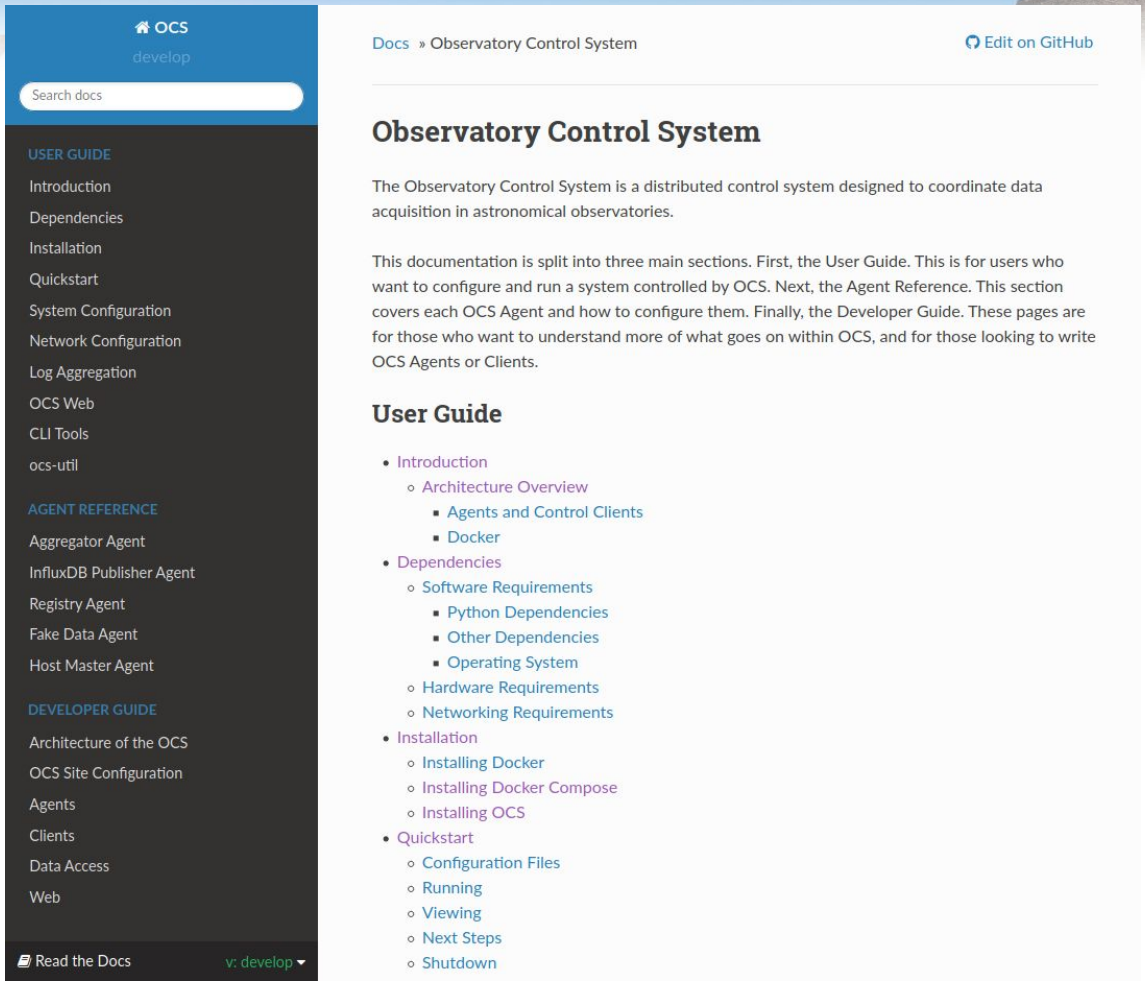
- Send commands to multiple Agents on the network
  - i.e. Servo DR to fixed temperature, take IV curve, repeat for these N temperature setpoints
- Clients take two primary forms:
  - Python script run on command line or in Jupyter
  - Javascript run in OCS-web
- Python clients tend to be setup specific
- OCS Sequencer - runs clients with nice web front end for orchestrating site Agent operations (still under development)
  - Can be used in labs to run clients

OCS Network Configuration

- Network (nearly) entirely defined by small collection of configuration files (2 key ones)
  - Docker Compose File - defines docker containers running Agents
  - OCS Site Config File - defines which Agents can be run and what input parameters they should be sent at startup
- Version controlled
- DAQ expert can identify configuration issues by just viewing these files most of the time
- Allows very rapid re-standing up of a DAQ system that might need to be redeployed for whatever reason (i.e. HDD failure, etc.)
  - Clone files, minor configuration on system/installation of packages, standup system.

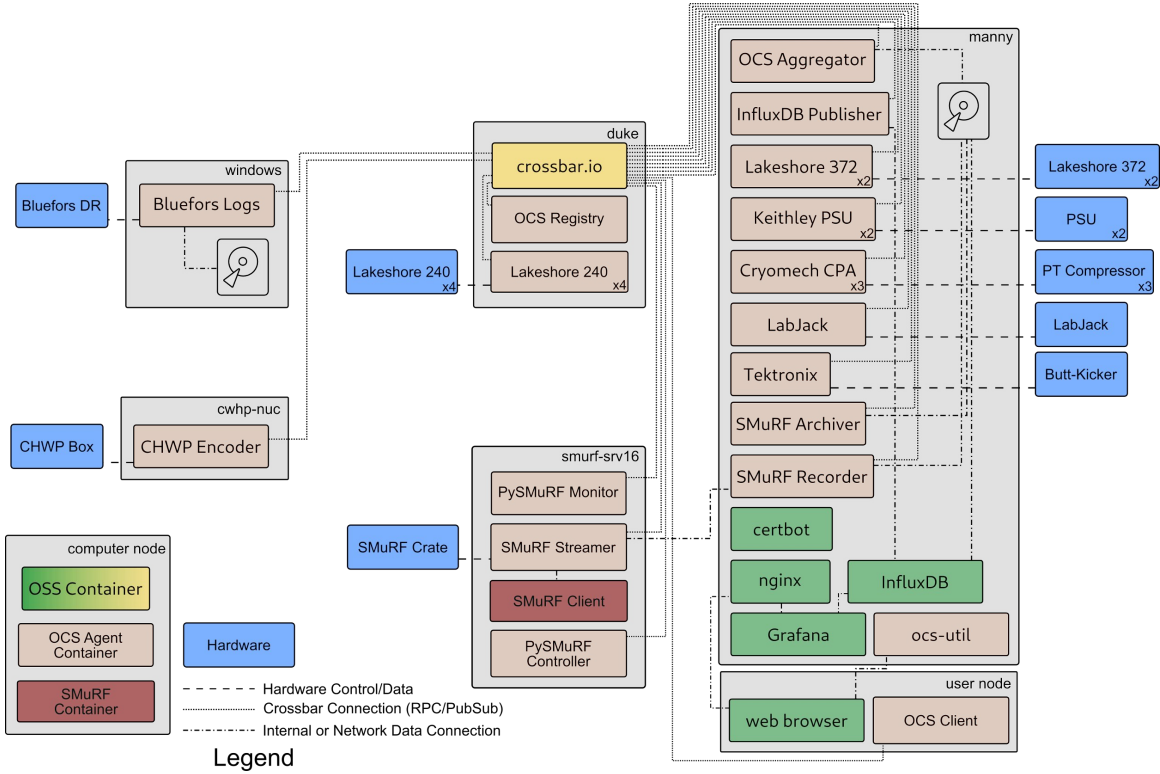
OCS Documentation

- Extensive documentation allows users to configure their own systems
- Detailed installation instructions
- Developer instructions
  - Describes Agent and client development
  - Details about various designs within OCS
- Individual Agent docs
  - Describes Agent functionality
  - Example configuration file information
- Has been key in successful adoption of OCS



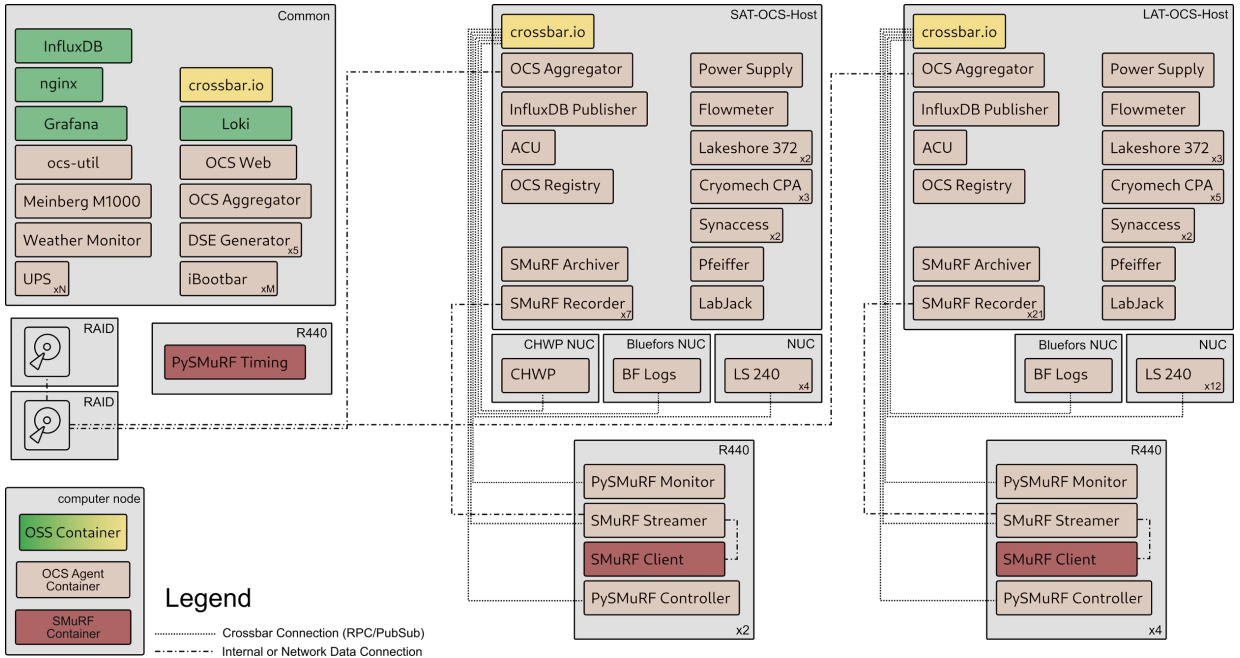
Example OCS Layout

- SAT1 deployment as of a few months ago
- Main computer (manny) runs most of the Agents, and includes live monitoring system
- smurf-srv16 runs SMuRF controller, Streamer, and Monitor agents - interfaces with SMuRF crate
- 3 other small computers for interfacing with specific hardware:
  - Bluefors DR computer (Windows)
  - CHWP NUC
  - Lakshore 240 NUC



Full Site Layout

- Full site layout is ~5x the size of the UCSD SAT1 layout
- Contains five different hosts:
  - 1x Common host - shared HK Agents, i.e. weather monitor
  - 3x SAT host
  - 1x LAT host
- Each is on its own crossbar server
- HK data published to a common InfluxDB for web monitoring
- Additional OCS hosts on SMuRF servers, hardware NUCs
- Tested at scale
- Similarly scale up for S4



Testing at Scale

- OCS is deployed to many labs at a small scale (i.e. at most the SAT or LAT setups)
- Need to evaluate OCS's capabilities at full site scale
- Results will inform computer hardware requirements
- Tests performed on a single server within virtual machines:
  - Ubuntu 20.04
  - 2x 16-core Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz
  - 1TB disk
  - 192 GB RAM
- Goal was to run full data rate HK and detector data simulators for all telescopes and record performance metrics
- Monitoring performed with system monitoring tools

Scale Testing OCS

- 1 crossbar server per telescope platform
- Shared InfluxDB + Grafana instances
- 100+ HK Agents publishing 26,000 points/sec
  - Largest platform publishing 7k points/s through crossbar at ~10% vCPU usage, 135 MiB RAM
- 86k simulated detector timestreams
- Resource usage:
  - Individual agents <1.5%, generally ~0.5% vCPU, ~30 MiB RAM
  - HK Aggregator (writing .g3 files) 3-5% vCPU, 100 MiB RAM
  - InfluxDB Publisher ~10% vCPU, 40 MiB RAM
  - InfluxDB ~34% (spikes to ~184%) vCPU, ~800 MiB RAM
- InfluxDB and writing to it are the most resource intensive parts.
  - Have implemented a separate path to InfluxDB than to .g3 files so downsampled data could be in live monitor
  - Proper application of retention policies and sampling rates should limit any InfluxDB issues
  - Possible to swap out if it is limiting (involves writing one new OCS Agent)

Monitor+Alarms Requirements flowdown

CQ1

- Real-time browser based monitoring of auxilliary (‘house-keeping’) and diagnostic data, including detector meta-data.
- Decimated views of historical data (>1week)
- Hierarchical alarm system based on housekeeping data
- Must monitor HK signals + meta-data from all telescopes at ~once/5sec:
  - ~200 sources/telescope, ~3800 samples @ SP
  - Data at different rates: eg telescope data (20 fields/telescope @ 200 Hz = 76,000 samples/sec; ~1000 thermometers @ 4 Hz = 4000 samples/sec, etc).
  - Should anticipate ~100,000 fields/sec @ SP as driver for this requirement

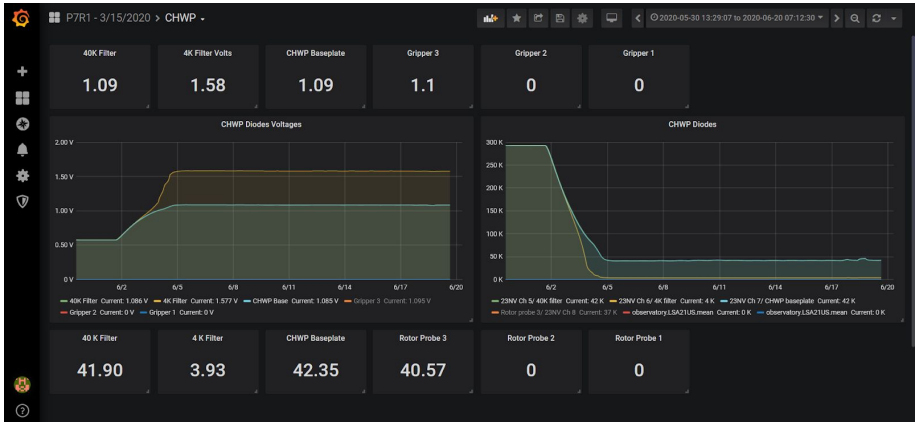
OCS Monitoring Overview

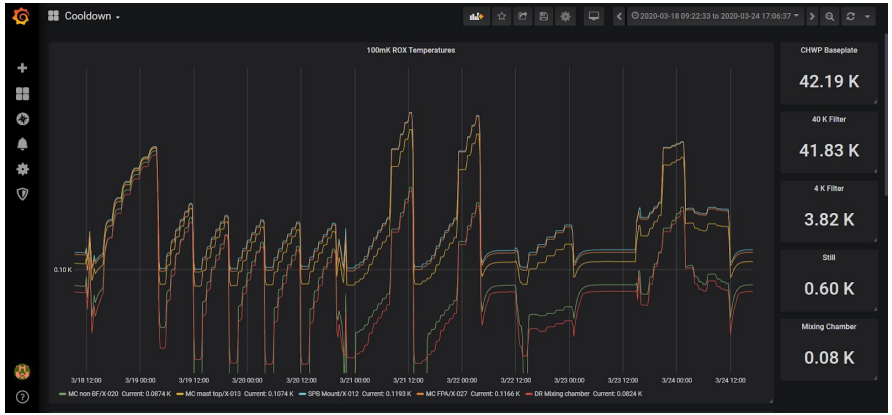
- Goals when building the monitoring system:
  - Remotely monitor housekeeping data and OCS operation through a web browser
  - Support user configuration of data queries
  - Enable both live and historical data viewing
- Selected a set of tools for monitoring:
  - Grafana - Web interface for interactive visualization of data in the web browser
  - InfluxDB - Time series database with nice integration to Grafana
  - Loki - Log aggregation system from the same people who made Grafana
- These are open source tools with large communities, documentation, and updates
- Currently available and in use at all of the SO institutions running OCS
- **Also being validated in the field at pole (SPT-3g)**

Notable Features

- Configured dashboards are saved and can be recalled or linked to
  - I.e. a dashboard describing the health of a DR can be configured once and referred to later
- “Explore” menu allows for on the fly exploration of data that might not have a dashboard already configured (and can be transferred to a dashboard)
  - Useful for tracing a particular event or issue
- Grafana + InfluxDB has a built in alerting mechanism
  - Currently in use at labs for events such as “thermometer X is > 1K”
  - Send Slack message with associated plot
- Loki log aggregation allows central view of system logs
  - All configured OCS Agents can log to Loki, allowing errors to be tracked down on the web interface

Example Grafana Dashboards



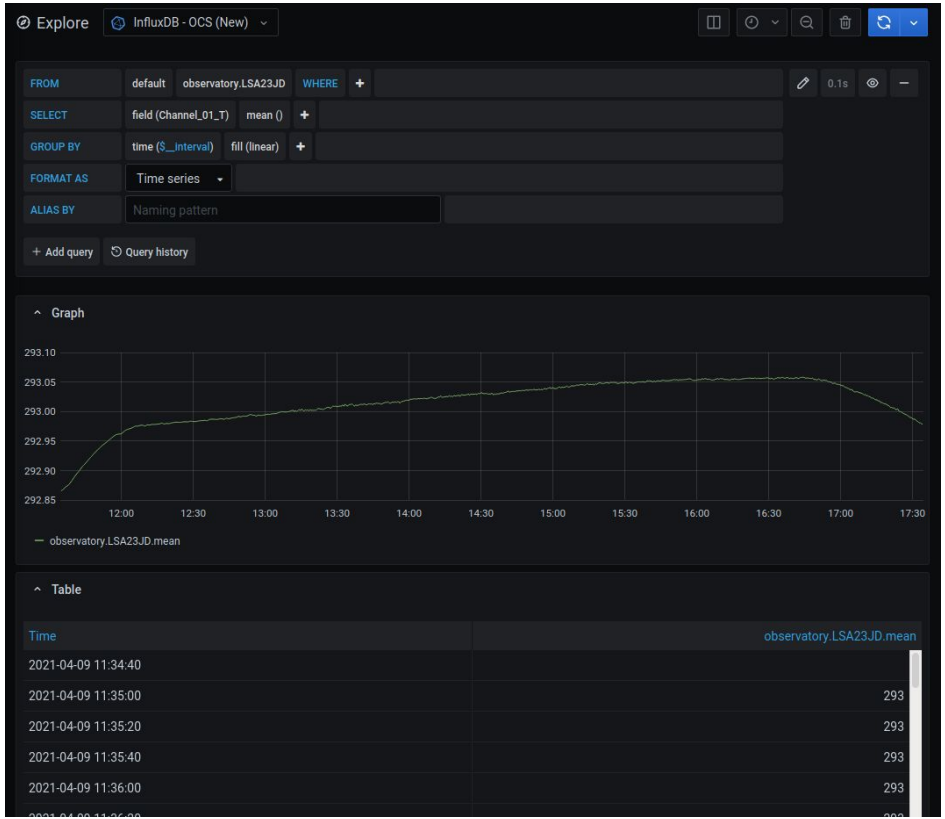


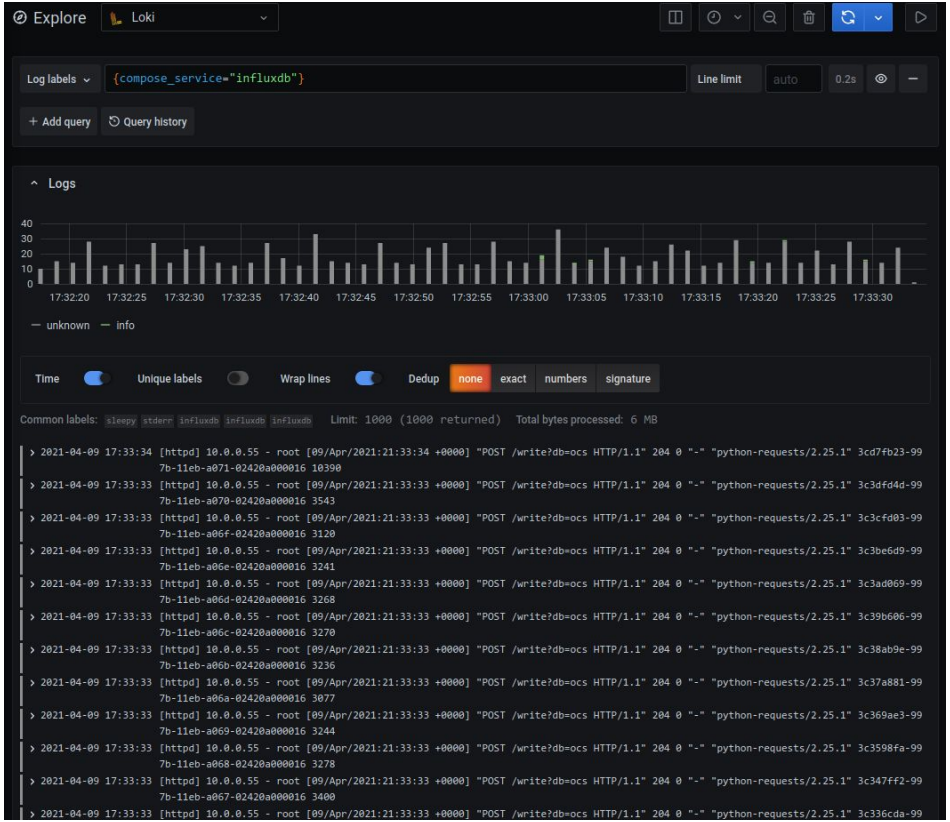
- SO SAT1 Grafana Dashboards
- Cooldown monitoring (upper left)
- Load Curves at 100 mK (bottom left)
- CHWP temperature monitoring (top right)
- All viewable remotely via website
- InfluxDB backend
- Data also saved to .g3 files

Notable Features

- Configured dashboards are saved and can be recalled or linked to
  - I.e. a dashboard describing the health of a DR can be configured once and referred to later
- “Explore” menu allows for on the fly exploration of data that might not have a dashboard already configured (and can be transferred to a dashboard)
  - Useful for tracing a particular event or issue
- Loki log aggregation allows central view of system logs
  - All configured OCS Agents can log to Loki, allowing errors to be tracked down on the web interface
- Grafana + InfluxDB has a built in alerting mechanism
  - Currently in use at labs for events such as “thermometer X is > 1K”
  - Send Slack message with associated plot

Explore + Loki Menus





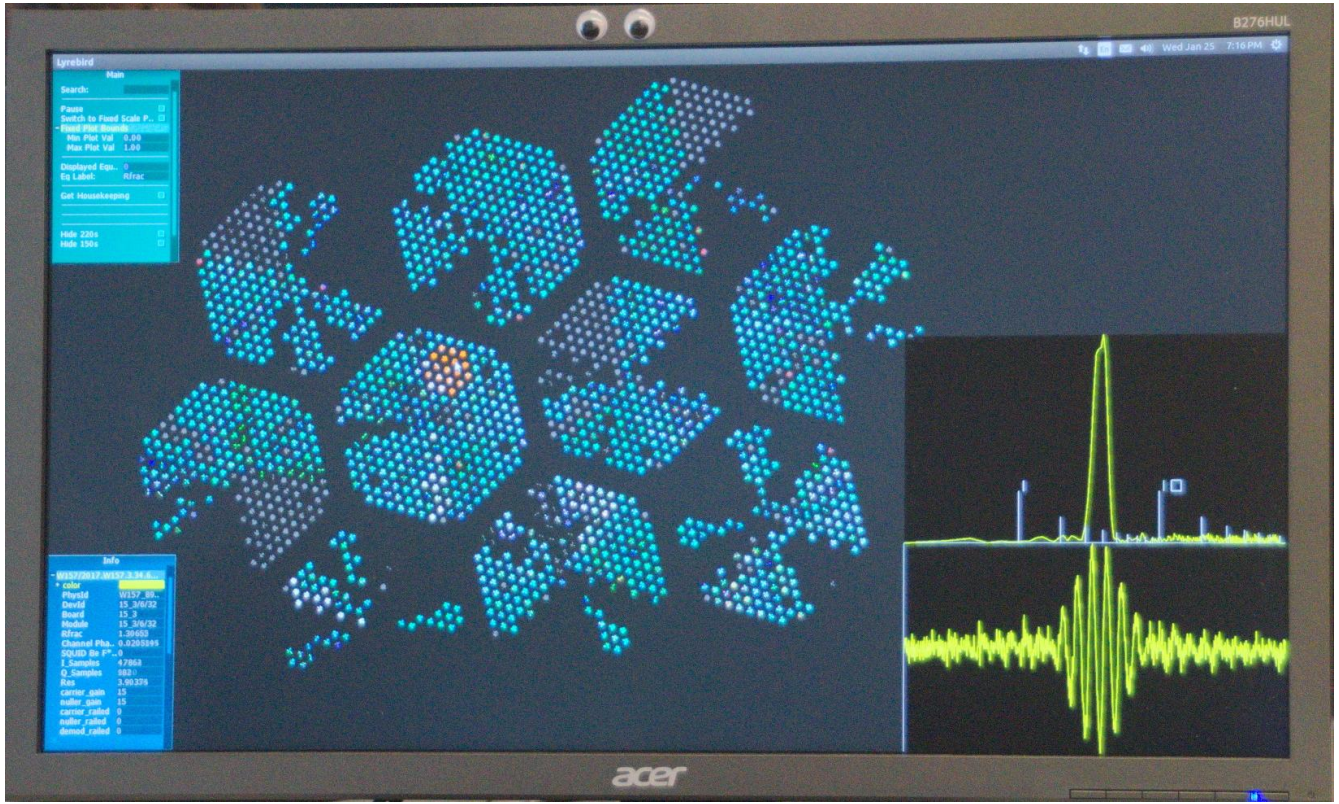
Notable Features

- Configured dashboards are saved and can be recalled or linked to
  - I.e. a dashboard describing the health of a DR can be configured once and referred to later
- “Explore” menu allows for on the fly exploration of data that might not have a dashboard already configured (and can be transferred to a dashboard)
  - Useful for tracing a particular event or issue
- Loki log aggregation allows central view of system logs
  - All configured OCS Agents can log to Loki, allowing errors to be tracked down on the web interface
- Grafana + InfluxDB has a built in alerting mechanism
  - Currently in use at labs for events such as “thermometer X is > 1K”
  - Send Slack message with associated plot

Detector Live Monitoring

Key features:

- Teeing off a copy of the detector datastream from the DAQ
- **Baselining an updated version of a tool (Lyrebird) used by SPT-3G, PB2, and SO**
- Particularly important for lab testing and I&C tasks



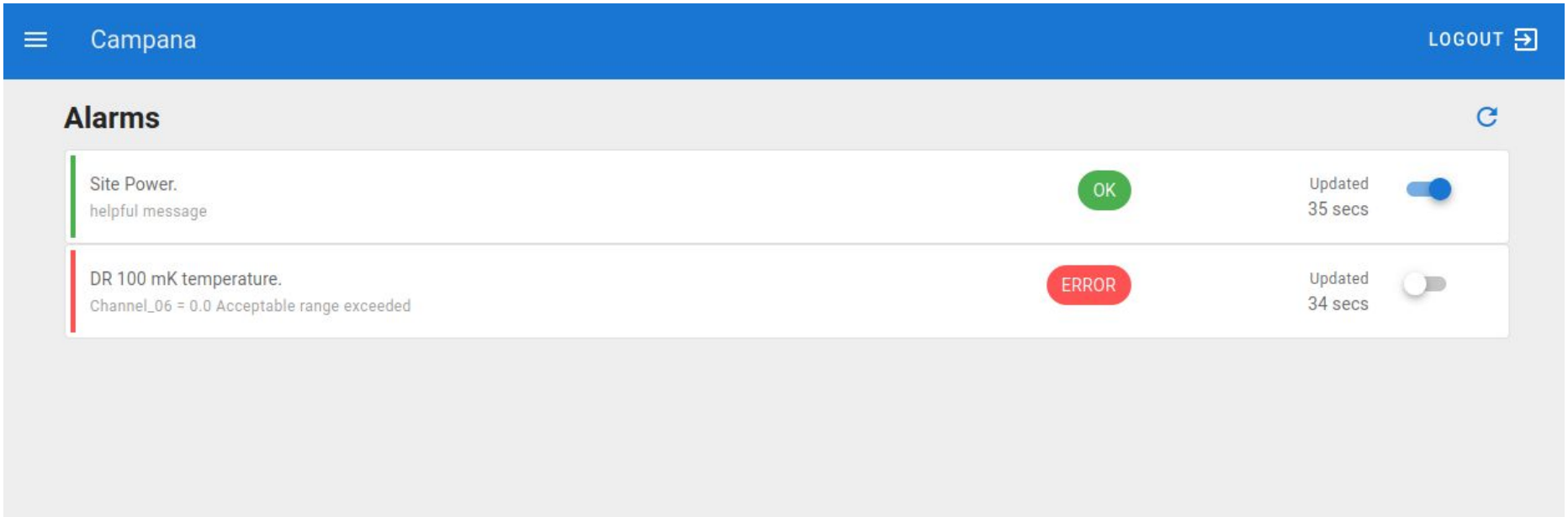
That design is meant to achieve:

- Modular, flexible, scalable: interface with many, changing hardware components, with different configurations between labs, etc
- Must robustly work in a distributed, heterogeneous environment of computers
- Efficient development: use commodity, commercial, or otherwise available software instead of building our own
- The user is the boss: Must be responsive to user needs and function as a live monitor for lab activities with minimal expertise on their part

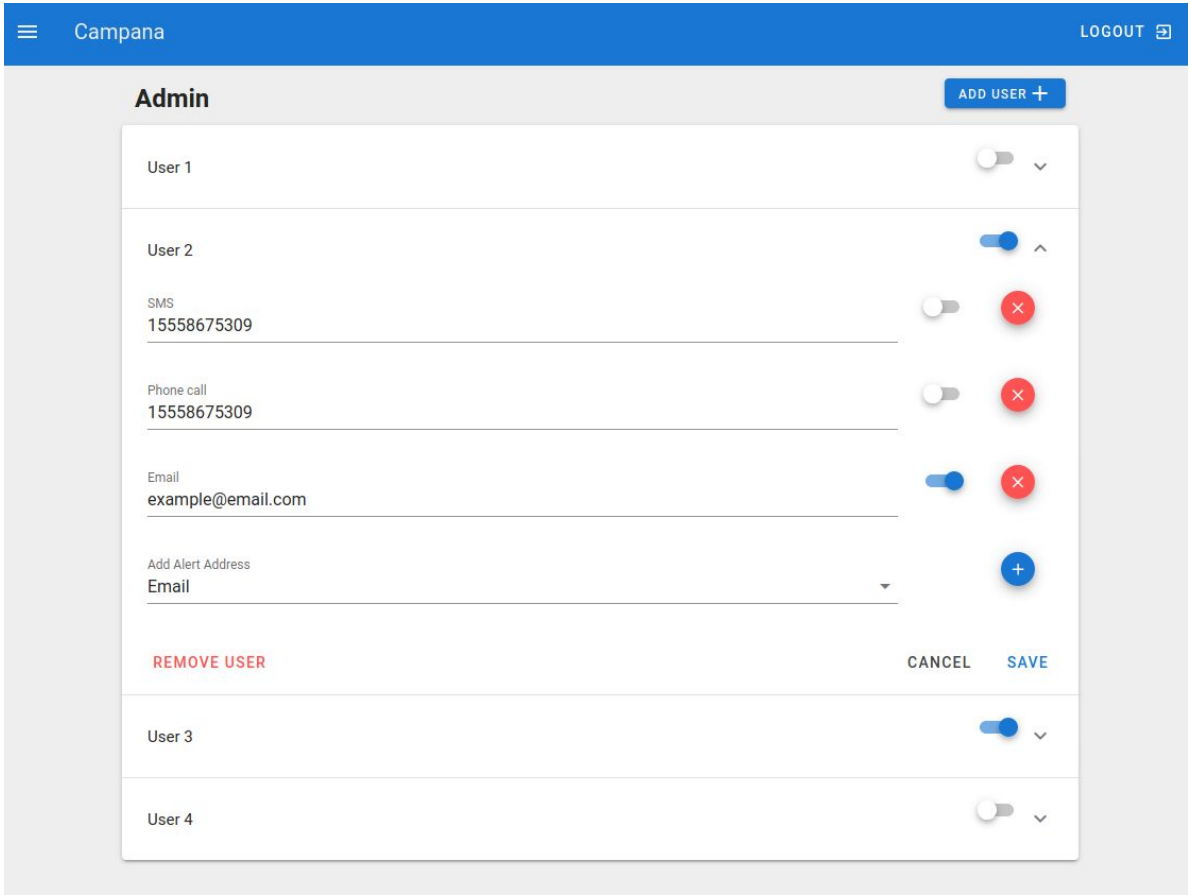
Alarms

- We are adapting the ACT alarms system for use on SO
- Each alarm has a simple script that checks values from OCS
  - Conditionally triggers various alarm levels, i.e. OK, WARN, ERROR
- Alarm states written to a backend database
- State dependent actions, i.e. send email, SMS, phone call
  - SMS + Phone calls sent via twilio
  - Will interact with South Pole alarm system provided by SP Station admin
- Updated web frontend under development
- Backend code has many years of testing on ACT

Alarms


Campana
LOGOUT
Alarms
Site Power.
helpful message
OK
Updated
35 secs
DR 100 mK temperature.
Channel_06 = 0.0 Acceptable range exceeded
ERROR
Updated
34 secs

Alarms


Campana
LOGOUT
Admin
ADD USER
User 1
User 2
SMS
15558675309
Phone call
15558675309
Email
example@email.com
Add Alert Address
Email
REMOVE USER
CANCEL
SAVE
User 3
User 4

Conclusions

The framework for OCS is complete and fulfills the requirements for S4. The package was developed for use on the Simons Observatory, and is widely used already in labs. It will also see thorough testing in the field in the near future.

Overlapping team members with SO means high familiarity with OCS, easing integration with S4.

As hardware is being selected and integrated in labs, driver code and OCS Agents will need to be written for use on S4. https://github.com/CMB-S4/s4ocs