Ethernet as a Fieldbus: Scaling Distributed Systems Interfacing with EPICS and ACNET

Robert Santucci

Ethernet as a Fieldbus: Scaling Distributed Systems Interfacing with EPICS and ACNET

April 24 2023

Abstract

In order to increase data acquisition rates and to lower complexity at the embedded level, Fermilab Instrumentation is developing an Ethernet field bus to communicate with a centralized middle layer utilizing Redis. Each embedded node will stream data to the middle layer where post processing can be done via micro services in containers. All data, including raw and processed wave forms, will then be presented to the control system. This architecture allows for easy scalability, lowers complexity, and allows easy adaptation to current (ACNET) and future controls systems (EPICS).

Background – How did we get here

- VME is becoming harder and harder to maintain at the lab
  - Complex learning curve with vxworks
  - Aging hardware
  - Limited bandwidth (10MB/sec)
- Current projects are black boxes
  - Echotek Driver
- Code re-usability is limited
  - Instrumentation has 100's of front-ends developed across multiple decades
- More Demanding readout requirements
  - Machine Learning and AI is coming
  - More data the better
- PIP2
  - Current and future controls systems need to be integrated together
- Users
  - Feature requests to view data slightly differently

Solution – Ethernet

- Commodity Ethernet hardware is readily available off the shelf
- UDP stack can be built into an FPGA
  - DDCP
- Future hardware platforms have it built in
  - MicroTCA
- socFPGAs can run a full Linux kernel
  - TCP
  - MPSOC Ultrascale+
- Easily integrated
  - Tons of open source software

Ethernet as a Fieldbus in the style of Surrealism : DALL-E

Solution – Redis

- Redis is an open-source, in-memory data structure store that can be used as a database, cache, and message broker.
- It supports various data structures such as strings, hashes, lists, sets, and sorted sets.
- Redis is designed to be very fast, scalable, and highly available, making it a popular choice for real-time applications and systems that require low latency.
- It provides a rich set of commands that can be used to manipulate the data stored in Redis, such as GET, SET, HGETALL, LPUSH, SADD, ZRANGE, and many others.
- Redis also supports advanced features like transactions, pub/sub messaging, Lua scripting, and cluster management, making it a powerful tool for building complex applications.

Solution – Containers

- Applications and dependencies are packaged together into a single, self-contained unit that can be easily shared and reused across different environments and platforms.
- Containers are self-contained units that include all of the necessary dependencies and configuration files, making it easy for developers to understand and reproduce the environment in which their application runs.
- Containers can be easily moved between different environments and platforms, making it easy to deploy applications across different infrastructure.
- Containers provide several advantages when it comes to embedded systems, such as reduced overhead, simplified deployment and management, and better isolation and security.
- Simplified development: Containers enable developers to work in isolated environments that closely mirror the production environment, which can lead to faster development cycles and better quality code.

A challenging path forward

- Design a system that uses Ethernet as a data transport layer
  - UDP and TCP
- Maintain real time responsiveness
  - Utilize embedded nodes to do embedded things, isolate the data acquisition layer
- Provide deterministic data
  - Limit congestion and cross traffic on the network
- Rapidly scale at the edge and at the frontend
  - Adding a new node should be as easy as plugging in new channels
- Become a data driven model
  - New data is streamed with a push only model
- Integrate into ACNET, EPICS, AI
  - Agnostic to anything that wants data
- Synchronization
  - State management for all of the above

Goals

1. Ethernet Fieldbus
2. Scalability
3. Real-time and State Logic Separation
4. Event-driven Communication
5. Load Balancing
6. Efficient Data handling
7. Adaptability

Ethernet Fieldbus

By isolating the embedded nodes to a private Ethernet vlan, we can manage and facilitate a deterministic readout mechanism reducing congestion generated by a general purpose network.

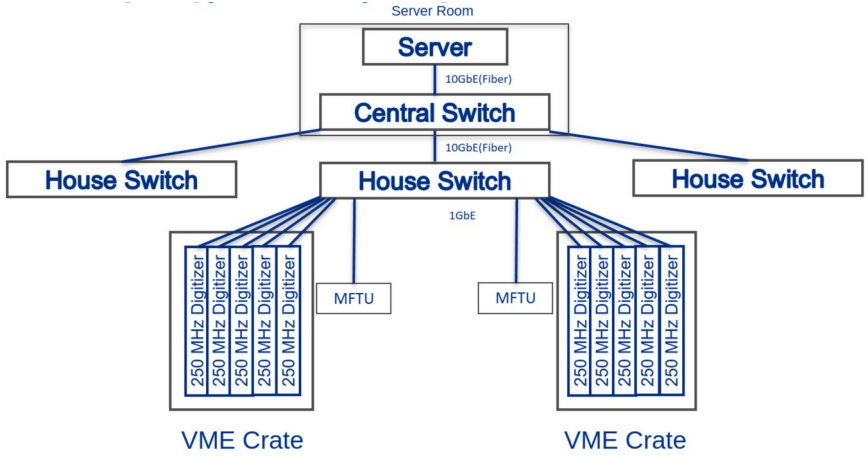
It’s a Fieldbus.. not a general purpose network

Theory

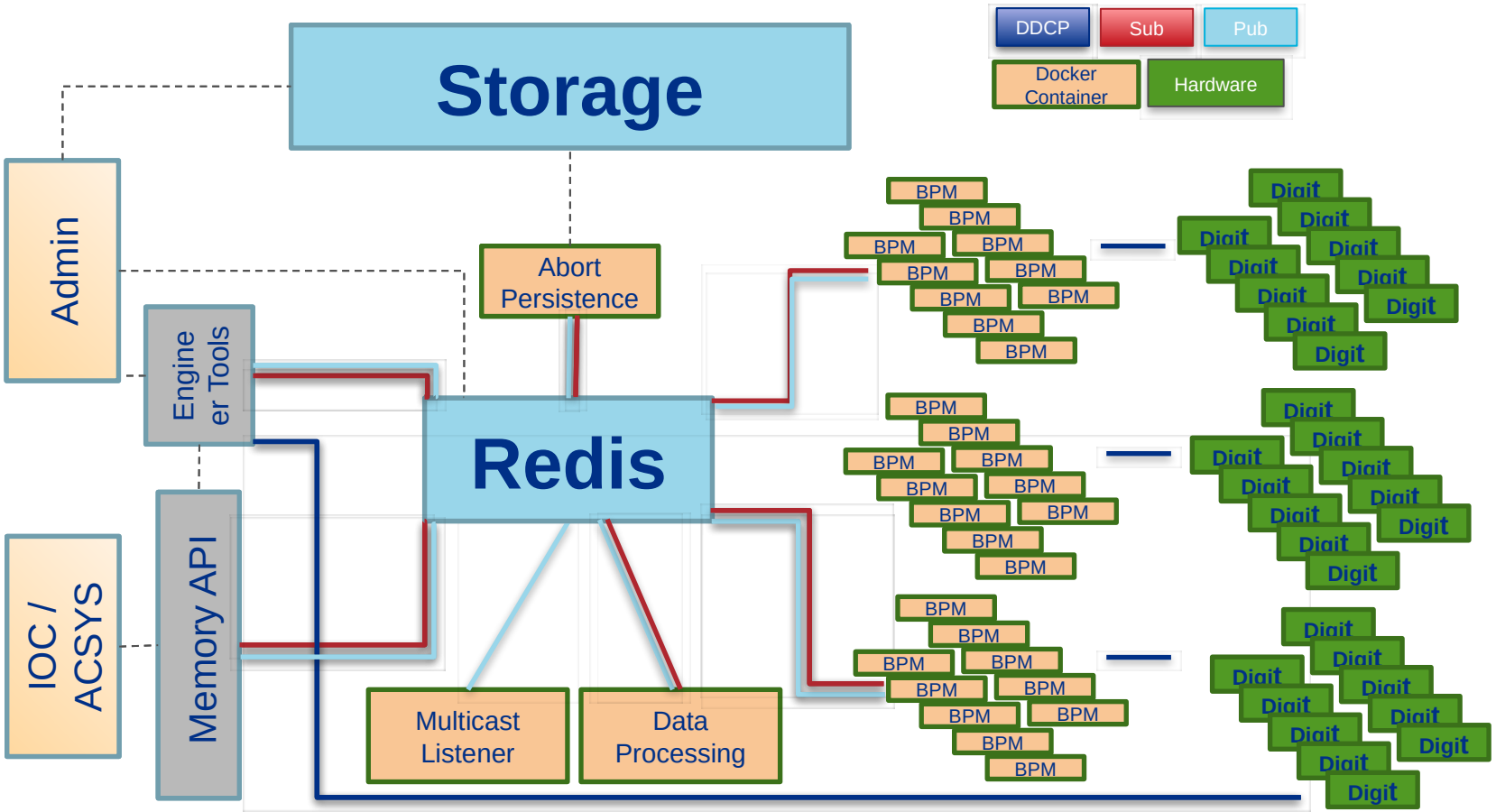
- 1 GbE = 125 MB/sec = 0.125 MB / msec
- 10 GbE = 1250 MB/sec = 1.25 MB/msec
- 40 GbE = 5000 MB/sec = 5 MB/msec

Practice

- 60 - 120 MB/sec (Digitizer to server)
- 500 – 800 MB/sec (Digitizers to server)
- More efficient with larger packet sizes



Fermilab


DDCP
Sub
Pub
Docker Container
Hardware
Storage
Admin
Engine er Tools
Abort Persistence
Redis
IOC / ACSYS
Memory API
Multicast Listener
Data Processing
BPM
Digit

Scalability

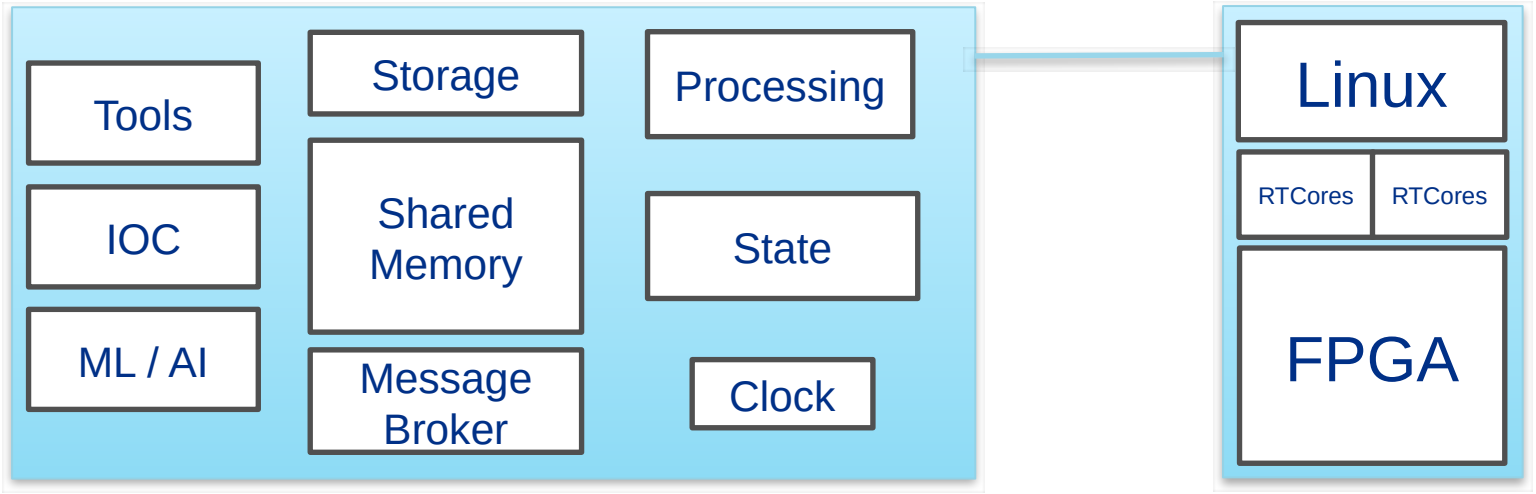
With Redis as a central communication hub, the system can handle a large number of SoC FPGAs and concurrent connections, making it suitable for managing data distribution in a distributed control system. Redis can be scaled to meet the growing demands of the system as the number of devices increases. Adding new devices becomes trivial as no new IOC needs to be developed, only the current one needs to be expanded.

- Define a key that utilizes the system, machine, and IP address of a given node
  - Allows new devices to reuse the same code / kernel with configuration files
  - Allows for multiple systems to participate on the same network
- Embedded nodes dynamically set IP / MAC address based on physical location
  - Kernel and FPGA firmware are net booted on power up
- Centralized server can be scaled horizontally
- Cost effective approach to increase data read out speeds
- 10GbE, 40GbE, 100GbE
  - The "uplink" of the system can be expanded with off the shelf hardware

Real-Time and State Logic Separation

By keeping state logic or real-time logic within the FPGA SoCs or separate standalone applications, you maintain a modular system design. This separation of concerns helps to create a more maintainable and adaptable system that can more easily adapt to changing requirements or incorporate new technologies.

- Break down the monolith
- Single Responsibility Principle



Efficient Data Handling

Streaming data from the SoC FPGAs directly to Redis allows for efficient data handling and reduces the load on the embedded node. This enables the embedded node to focus on delivering deterministic data to a centralized broker and allows the IOC to focus on facilitating client access to the data within Redis.

- Streams
  - Append only structures that can be queried , ranged, and subscribed to
  - Timestamped at the server relative to wall clock or a custom timestamp from the edge
  - Metadata to back out the pulse or event that triggered it.
  - Pushed from the embedded node
- Streams of streams
  - Once data is at the server, common manipulations such as averaging, or slicing can be done in micro services which produce other streams
- Limits the amount of data produced and sent from the edge
- Removes complexity from the edge
- Removes the IOC from the edge

Redis Keys

- Machine:System:IP:Device
  - MUON:BPM:10.200.22.60:HP619
- Each colon ' : ' in the key defines the beginning of a unique key space
- Keys are hashed and spread across the cluster
- Keys are accessed based on their data type
- Define Key names and PV names with a 1:1 relationship

MUON:LOG
MUON:DATASTREAM
MUON:PUB/SUB
MUON:CONFIG
MUON:STATUS

MUON:BPM:LOG
MUON:BPM:DATASTREAM
MUON:BPM:PUB/SUB
MUON:BPM:CONFIG
MUON:BPM:STATUS

MUON:BPM:10.200.22.60:LOG
MUON:BPM:10.200.22.60:DATASTREAM
MUON:BPM:10.200.22.60:PUB/SUB
MUON:BPM:10.200.22.60:CONFIG
MUON:BPM:10.200.22.60:STATUS

MUON:BPM:10.200.22.60:HP619:LOG
MUON:BPM:10.200.22.60:HP619:DATASTREAM
MUON:BPM:10.200.22.60:HP619:PUB/SUB
MUON:BPM:10.200.22.60:HP619:CONFIG
MUON:BPM:10.200.22.60:HP619:STATUS

Event-driven communication

Using Redis's pub-sub and stream features allows for event-driven data communication, enabling data updates to be sent only when necessary. This reduces network overhead and improves overall system responsiveness.

- Push only communication to Redis
- Hardware is triggered via clock
- The newest data will be available in the stream, no need to read registers unnecessarily
  - Similar to an interrupt event
- Only raw data is sent on the wire
  - Limits UDP packet drops, congestion, and context switching which lead to non deterministic systems
- Data is "fanned out" at the server where processing power is cheap

Load balancing

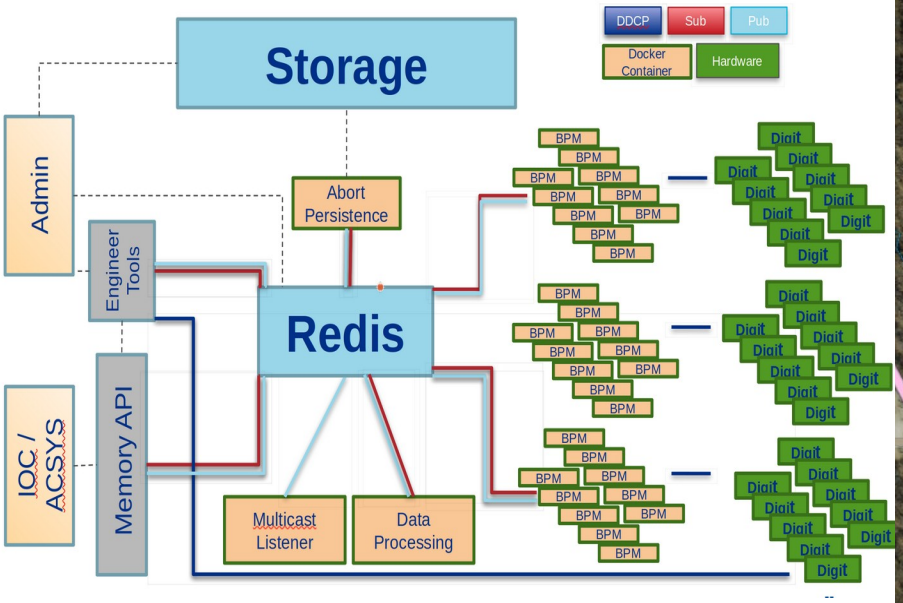
By offloading data handling and state logic responsibilities to Redis and other applications, the IOC can focus on its primary role as a data broker, which can help balance the load between different system components and improve overall performance.

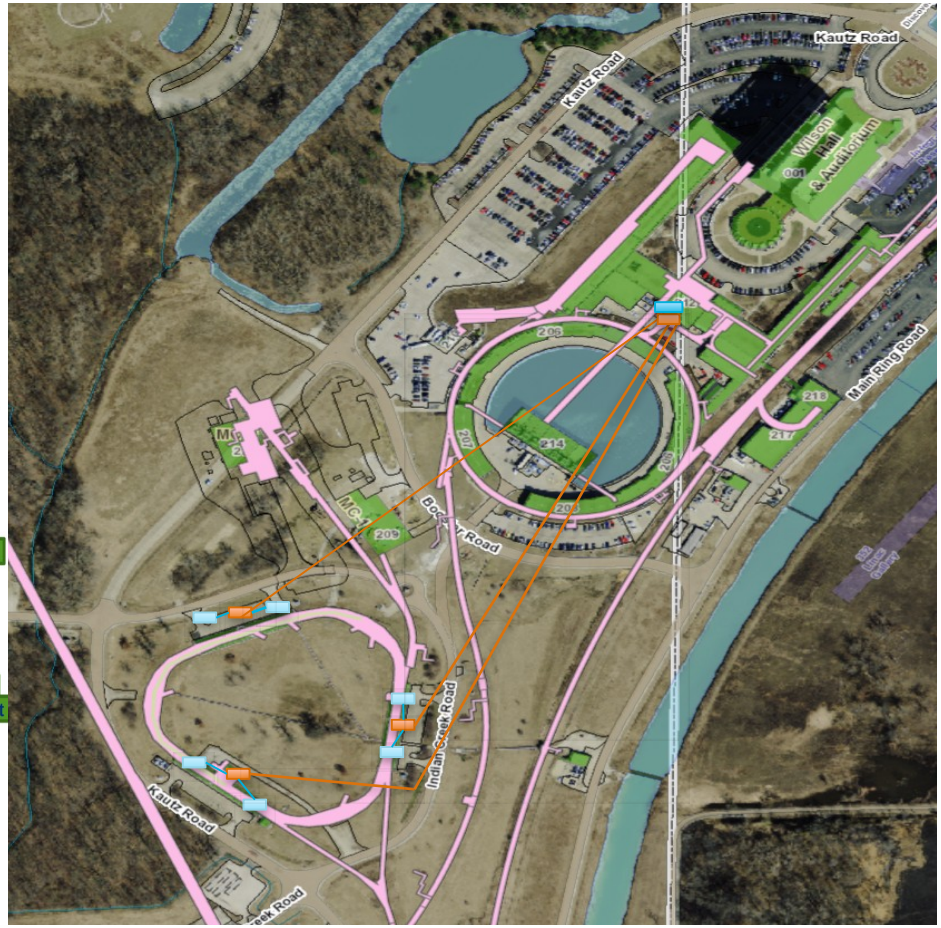
- Redis can be clustered across hardware
- Container management with micro services
- IOC can be on different hardware all together acting as a data broker
- Services that need all the data all the time can go direct to the stream
- Utilize the soc for simple processing and for Linux
- Real Time Logic goes in the real time cores
- FPGAs do FPGA things

MuonBPMs

Requirements

- Position-Intensity per BPM on every turn of beam
  - 250,000 Turns (1.695 usec / turn)
  - 4 BPMS / Digitizer
  - 30 Digitizers
  - Two 1 MB waveforms per BPM (Position/ Intensity)
  - ~240 MB
  - Total data depends on run conditions
  - Outcome: Reading logic is no longer embedded at the edge, instead parsed out on request (Closed Orbits, Flash Frames, FFT)
- Parallel Reads
  - 1.25MB/ms (10Gbit Ethernet)
  - 1 readout at end of cycle
  - 1.2 seconds to complete a readout and clean up

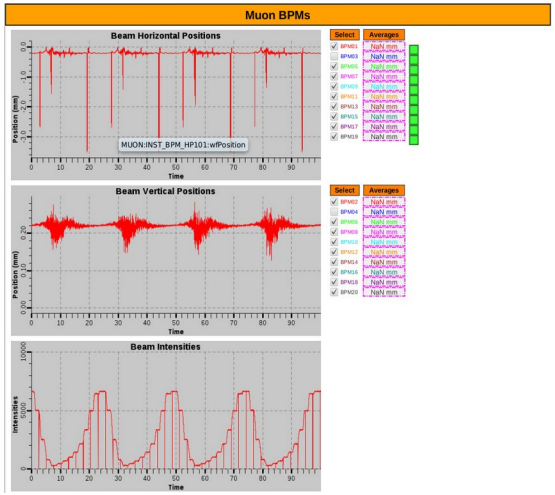

DDCP
Sub
Pub
Docker Container
Hardware
Storage
Admin
Engineer Tools
IOC / ACSYS
Memory API
Abort Persistence
Redis
Multicast Listener
Data Processing
BPM
Digit


Kautz Road
Wilson Hall & Auditorium
Main Ring Road
Indian Creek Road
Kautz Road

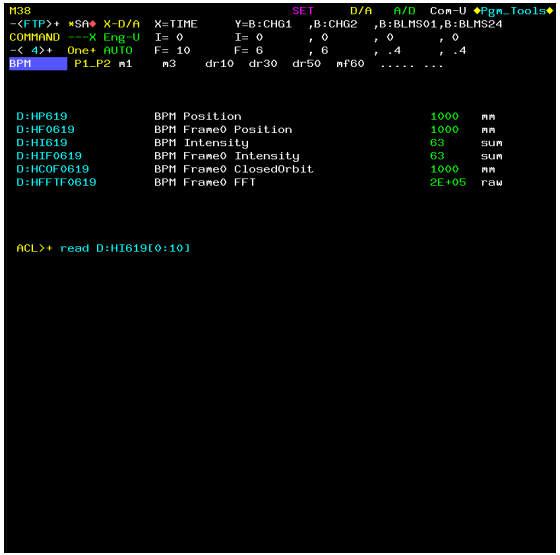
Adaptability

Utilizing Redis as a central communication and data hub, current and future technologies including ACNET, and EPICS can utilize and coexist on the same machine

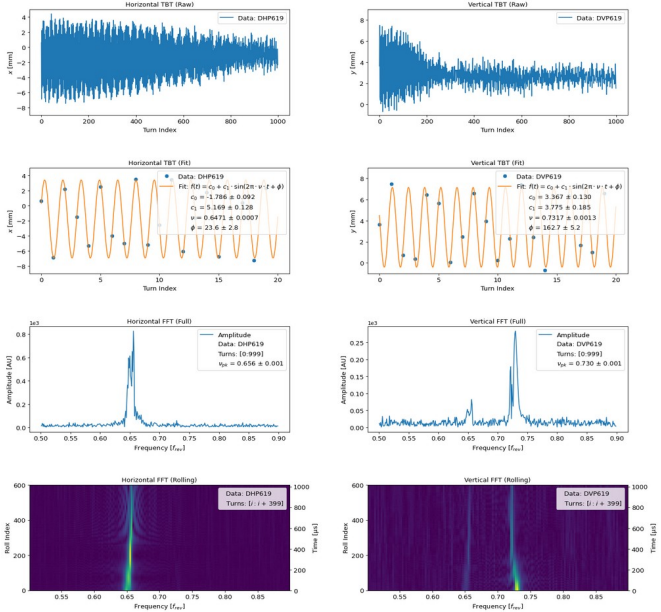
EPICS



ACNET



Python



Backend and Data Acquisition

- **Docker and Podman**
  - Allows for easy development and deployment
  - Applications can be tested at scale with hardware mockups
    - Spin up 100 fake digitizers (DDCP servers which respond like a digitizer)
- **Redis**
  - Centralized shared cache and message broker
- **DAQ Engines**
  - Light weight containers that marshal data from hardware
  - Utilizes Redis for storing configuration and for publishing data
  - Subscribes to messages as well for command control
  - Can exist as a thread on a server or on an edge node
- **Support Microservices**
  - Multicast publisher
  - Abort subscriber
  - FFT and Tune Analysis
  - Closed Orbits
- **Memory API**
  - Shared library to develop IOCs / ACSYS or other FE technologies
  - Understands where to go for data or how to listen in.

Backend and Data Acquisition - Redis

Redis is an open-source, in-memory data structure store, used as a database, cache, and message broker. It supports various data structures such as strings, hashes, lists, sets, sorted sets, and bitmaps

Cluster

- Redis can run in a single node configuration or multinode.
- Each node is single threaded and handles all input /output for a given hash set

Stream

- Each waveform is stored in memory using an append only data structure
- Each stream can contain metadata from the reading including its size and a timestamp

Publish / Subscribe

- The pub/sub model allows for messages to be passed between processes and to/from the outside (Control System)