



**UK Atomic Energy
Authority**
**Using Continuous Integration in the
development and verification of a
new central controller for JET**

Edward Jones – Control Software Engineer

Advanced Controls Unit – Advanced Computing Division

Agenda

- Problem Statement
- Scope of Work
- Workflows
- Solution
- Outcome

Scope of Work

For our project, given it was the direct plasma control mechanism, we needed:

- A way to define a control algorithm per pulse/campaign/week
- To protect our controller from inadvertent tampering
- React in real-time (2ms)
- Provide users with the ability to test control algorithms in safe environments
- Validate our upgrades

User workflow

Offline Testing

Recorded JET Data

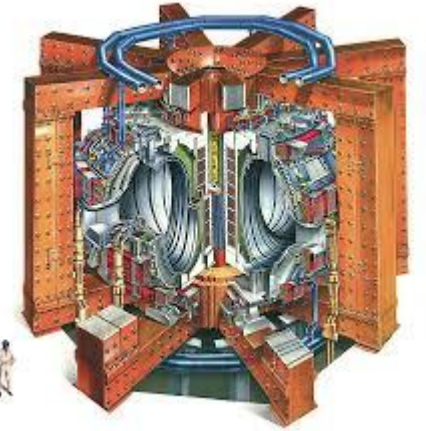


Online Operations

Data Store

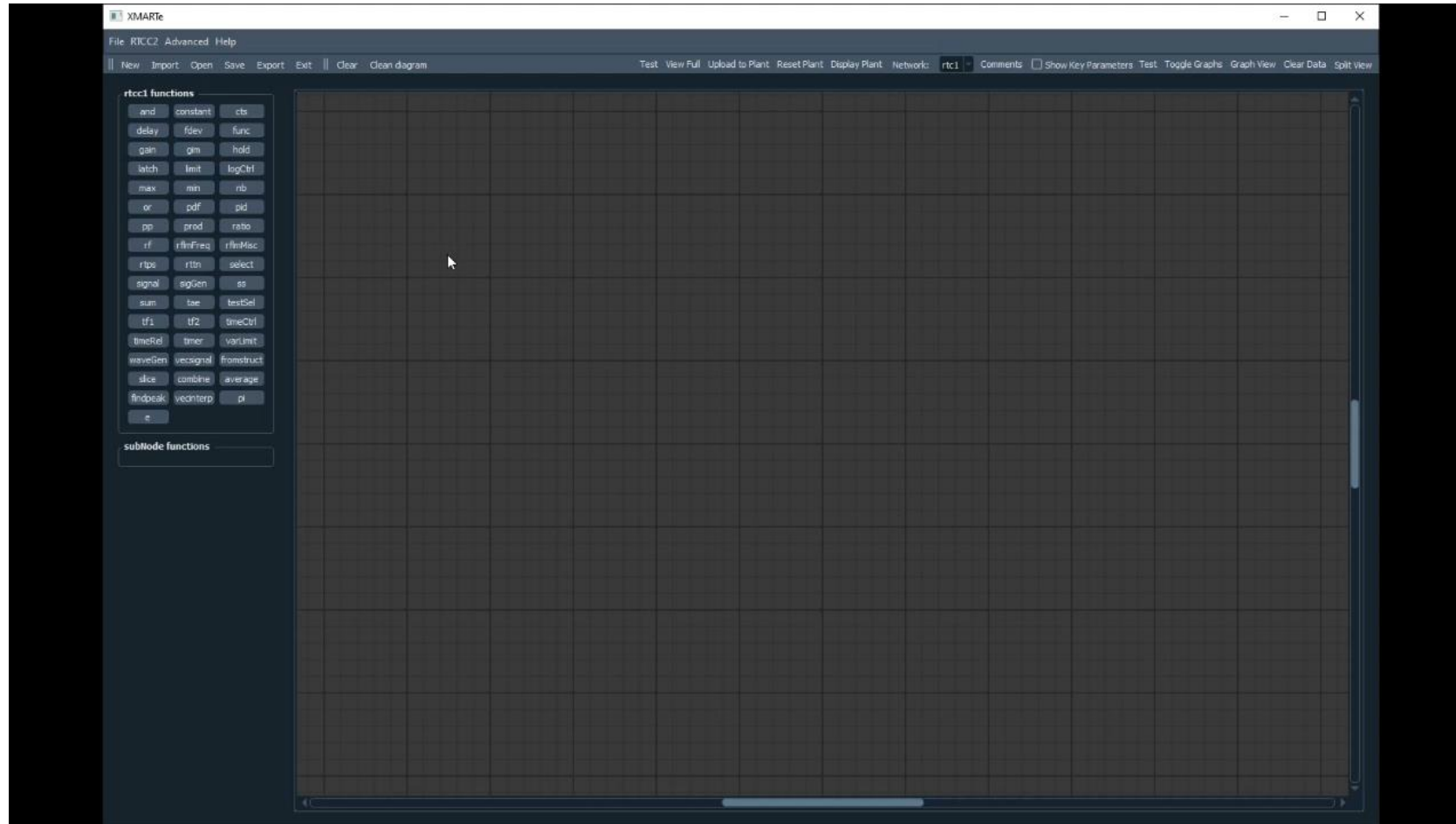


Plant Server

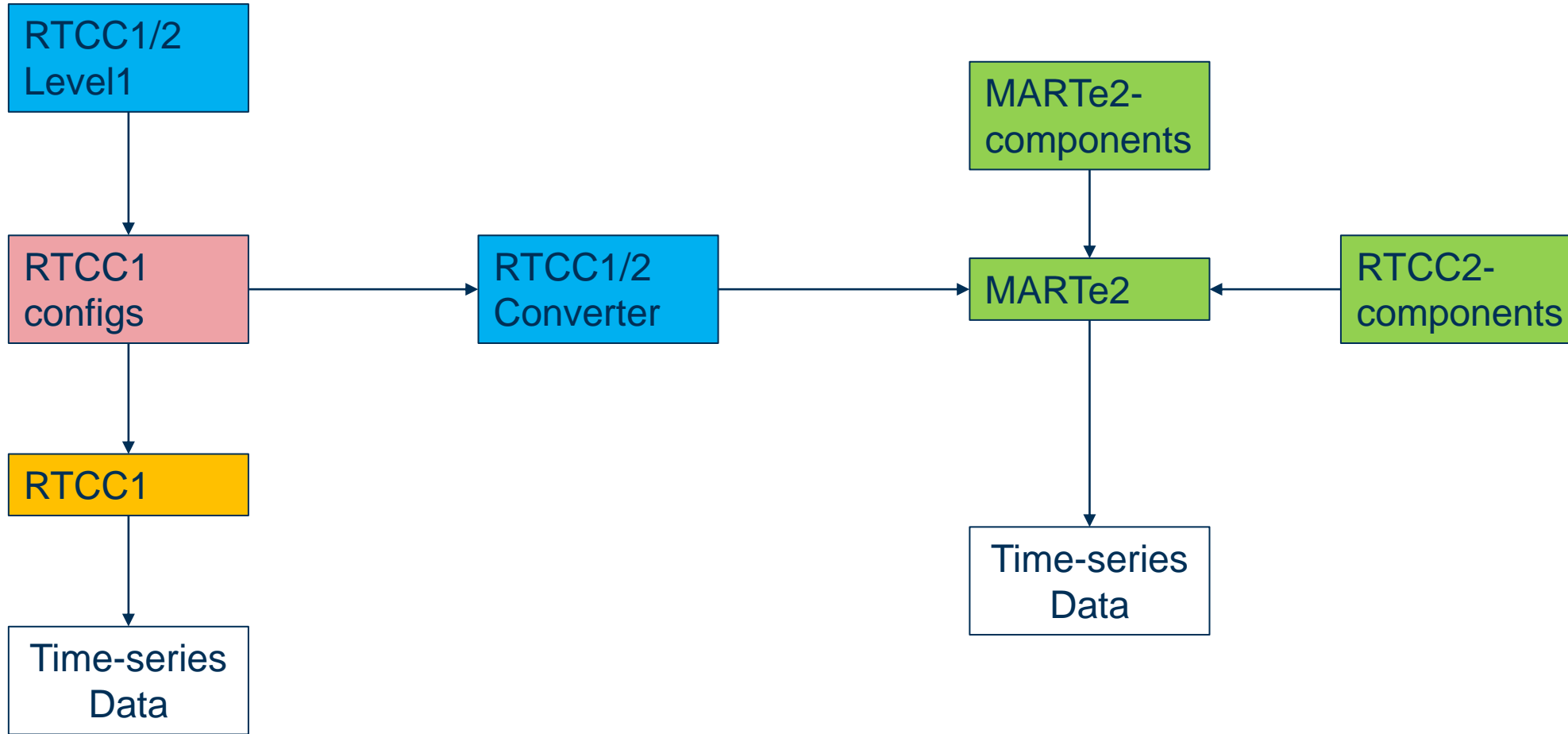


RTCC1/2

The GUI Itself

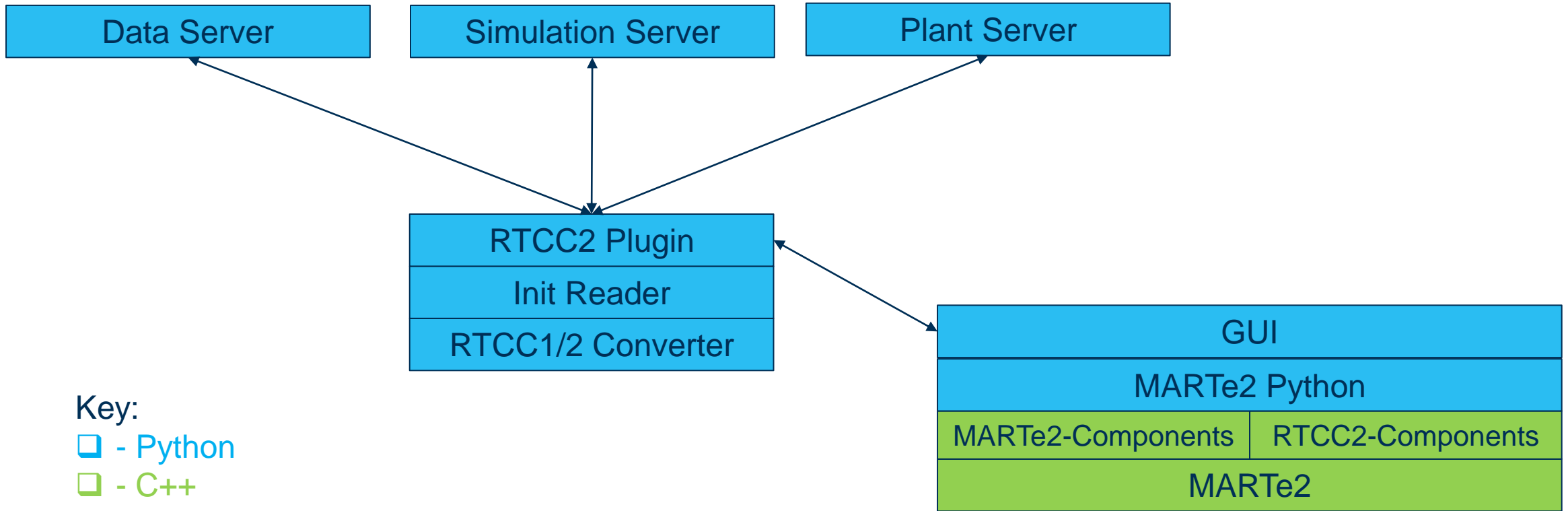


Workflow



Key:
□ - Python
□ - C++
□ - C
□ - Config Files

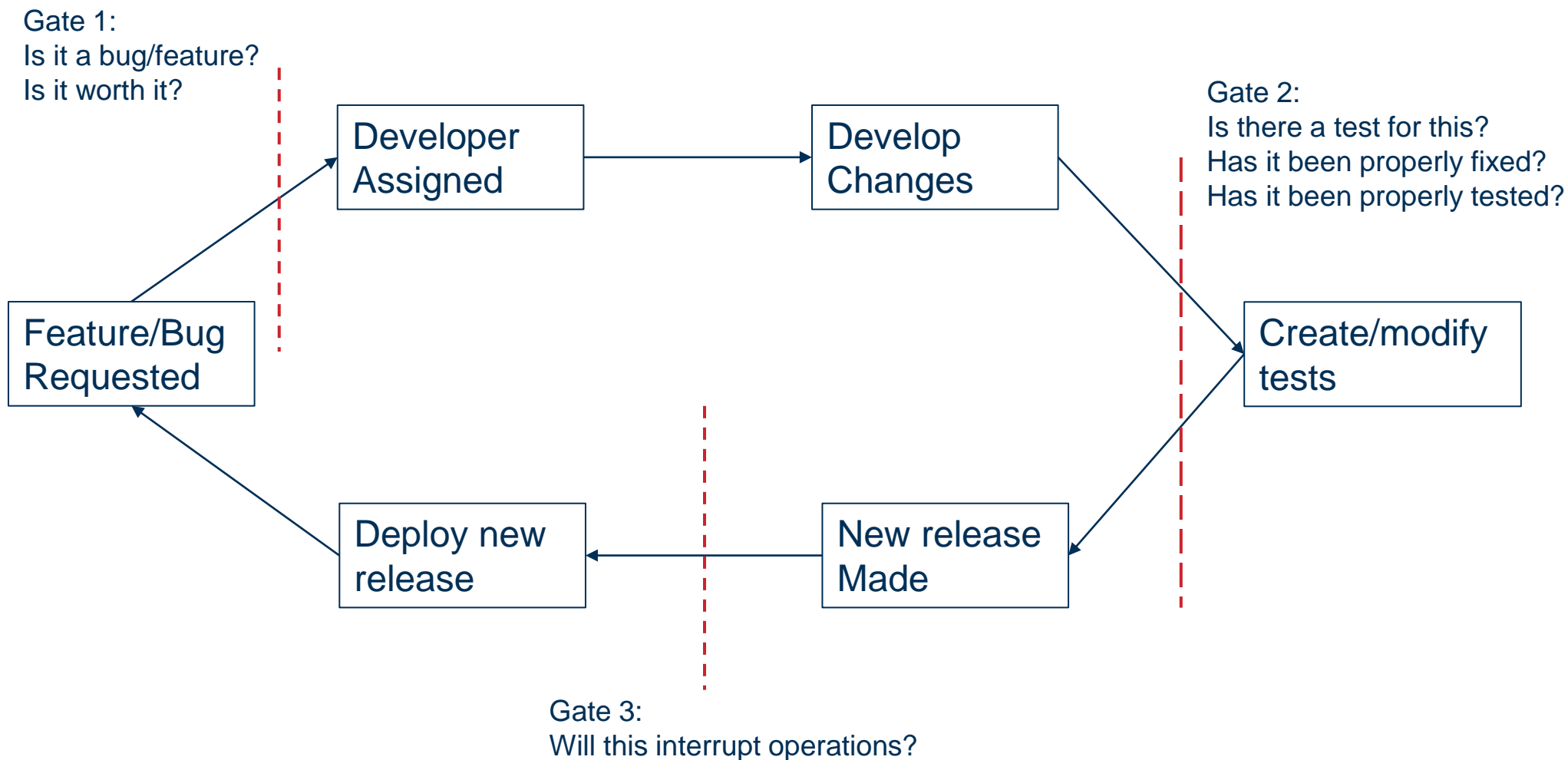
Code stack



Key:

- Python
- C++
- C
- Config
- Files

Software Development Workflow



Why – Software Testing

Risk Mitigation:

- Not meeting project deadline
- Not meeting essential project requirements
- Reputation
- Operating outside safe conditions
- Late discovery of issues
- Bad user experience

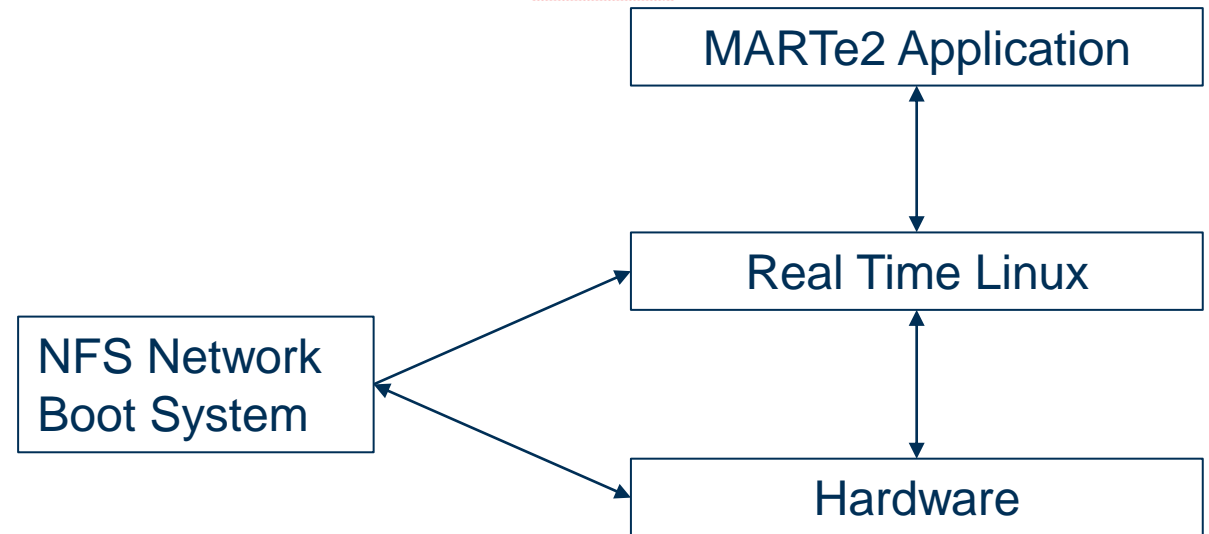
Deployment

For any software, it must run on hardware but we found new challenges:

- CentOS 7 is outdated
- Real time setup is a 27pg complex document.
- Updating software is difficult
- Differing versions across machines and setups
- Requires advanced Linux users
- Time consuming process
- No reproducibility
- Hosting through NFS network booting*

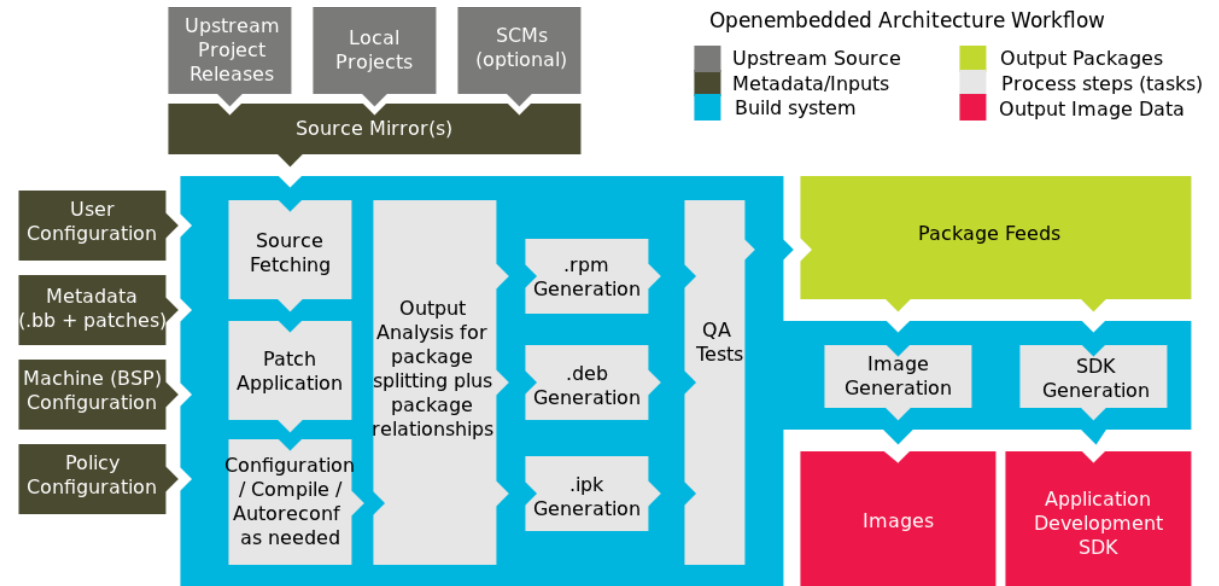
2 General steps to deployment

1. Choose a distribution and kernel version.
2. Obtain the installation media from a trusted source.
3. Perform a minimum installation.
4. Install required packages and/or software.
5. De-install unwanted/unused packages.
6. First reboot - disable all unnecessary hardware from BIOS.
7. Choices on drivers (remove from compilation, blacklist).
8. Re-compile the kernel for real-time.
9. Second reboot - kernel parameters.
10. Configure services and kernel threads.
11. Low-level configuration; interrupts and PCI bus
12. Testing and providing a fit-for-purpose justificaiton



Solution - Yocto

- Version control and hostable on Gitlab
- Build through Gitlab pipelines with a specific runner
- Automated build at code push
- No engineer required for the build process*
- Automated production of packages and package management system
- Modifiable boot procedures



Caveats

- Driver Setup and Installation
- Boot mechanism*
- Building to test against

What – The tests we ran

- Compilation and Gtesting of MARTe2 components.
- Conversion of existing RTCC1 pulse configurations:
 - Within a basic linux docker image of ubuntu:20.04
 - Within a docker image generated from Yocto on a real time system.
 - On mimic hardware we had in our lab which used a cross between a RAM and NFS root file system to maintain real-timeness.
- * Regression testing against RTCC1 pulse data.
- Pytests for the GUI itself and user interactions/stories.

How – the difficult part

It's all well and good constructing this grand plan, the execution however is what you really need.

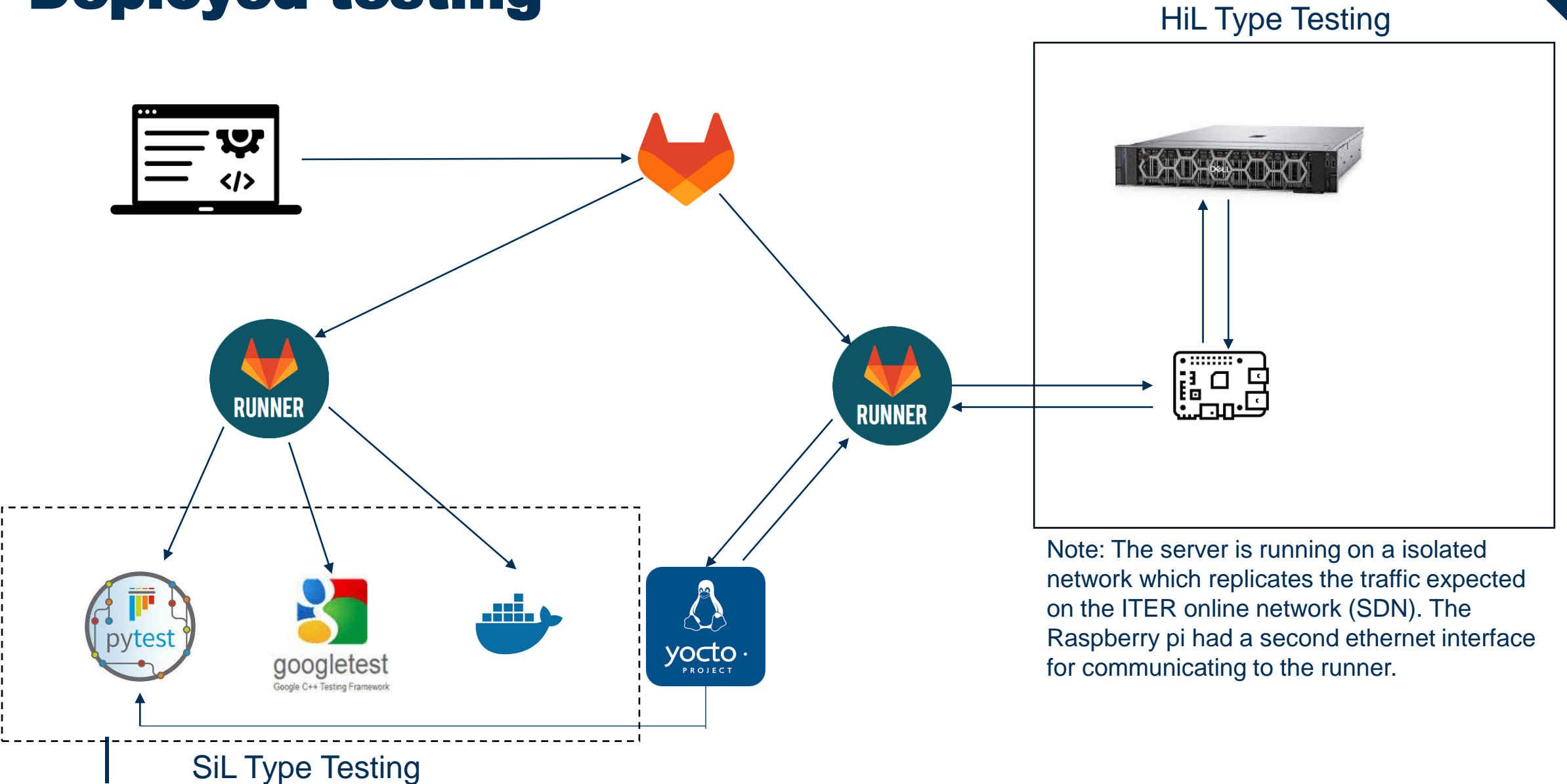
Some things are already well documented:

- Creating Gtests
- Creating pytests
- Creating pytests for a PyQt based GUI

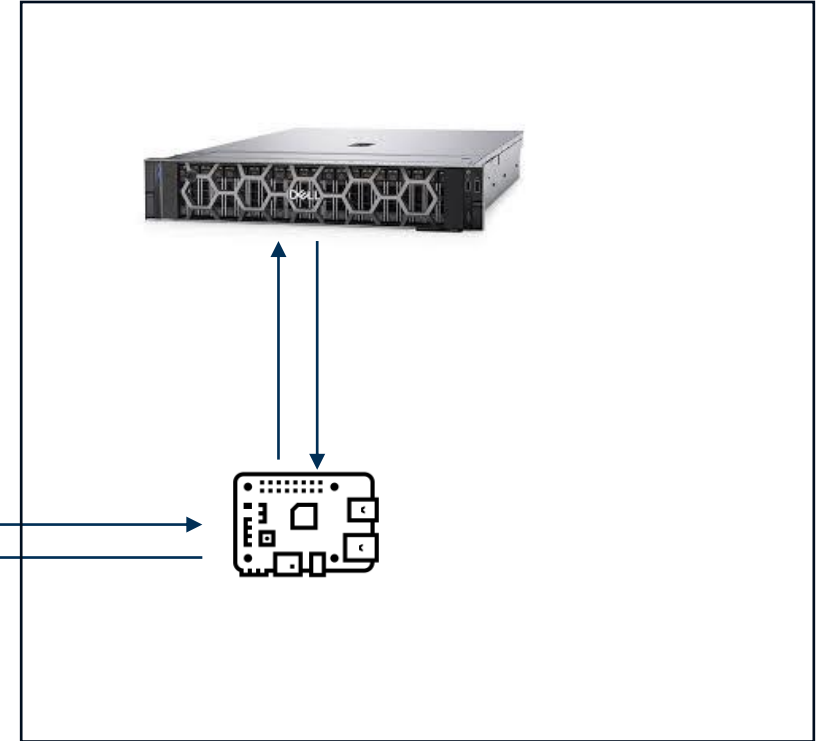
Some things not so well documented, or at all:

- Loading a system image from NFS into RAM
- Deploying a yocto image from a pipeline to then perform tests on
- Creating a docker image from a yocto image (for automated or manual testing).

What we came up with: Deployed testing







HiL Type Testing



Note: The server is running on a isolated network which replicates the traffic expected on the ITER online network (SDN). The Raspberry pi had a second ethernet interface for communicating to the runner.

SiL Type Testing

Gitlab Pipelines






passed #156734  **new-document...**  a8b901ee trigger: passed  00:09:24
5 months ago 

Status Pipeline Triggerer Commit Stages



failed #156759  **master**  d7cd7c61 01:48:33
latest  Fix findpeak build  5 months ago 

passed #156754  **master**  d7cd7c61 01:59:03
latest  Fix findpeak build deploy: passed  5 months ago

#156750  **master**  d7cd7c61 02:00:41

passed #156459  **develop**  5b8085a7 00:27:11
latest  Merge branch '151-fix-ss' into 'd...  5 months ago 

< Prev 1 2

-  test-execution
-  test
-  yocto-test

Outcomes

- Identified difficult to see errors/issues in calculations during tests.
- Identifies introduced bugs after codebase changes.
- Provided developers with a confident mechanism of proving their work and performing merge reviews.
- Provided our stakeholders and user base with confident measurements of the project process as well as what was possible at each stage.
- Ultimately we had a faster time to market, we were able to deliver the project on time for the experiment with a great level of confidence.
- We were able to adapt faster to new projects that came up afterwards and re-use a large bulk of our work.

Useful links

The JET experiment: <https://www.youtube.com/watch?v=1Zsv4JzogTE>

Boot setup: <https://www.youtube.com/playlist?list=PL-EYJCXIWZxBSuFbO2h9GgF-tcbrOotnV>

Automating Yocto Processes: <https://www.youtube.com/playlist?list=PL-EYJCXIWZxBdzbMRbtXcBG9QodMcCkBb>



Thank you for your time today

I'll be available for questions in the poster hall tomorrow.