



XKIT
XROOTD · KUBERNETES INTEGRATION TESTING

XKIT:

~~XCache, Xrd-PFC, ...,~~

**XRootD
Kubernetes
Integration
Testing**

Rob Currie, Wenlong Yuan,
Peter Clarke

Intro



- Apologies, a very packed set of slides for 10min.
- Some of this I'll try and move quicker over, interested people can look online later.
- Happy to discuss this in more detail over tea/coffee/fresh-air.
- We're looking for some input/wisdom from the community to improve this 😊 Please tell me I'm wrong.

Supporting XCache

- **XCache** is in use by multiple UK sites for Virtual Placement. **One example of UK XRootD usage.**
- **VP** relies on **GeolP ordering** of replicas as returned from **RUCIO**.
- Known to be broken. Was “*fixed*” for DUNE several months ago. Working on fixing globally.
- Fix expected to take a few weeks of development.
- Integration into RUCIO will likely be ~6months or so (**my** naïve estimate).
- Expected that the fix might be possible through server-side. If not, clients need to update ☹️.

Motivations for XRootD Integration Testing

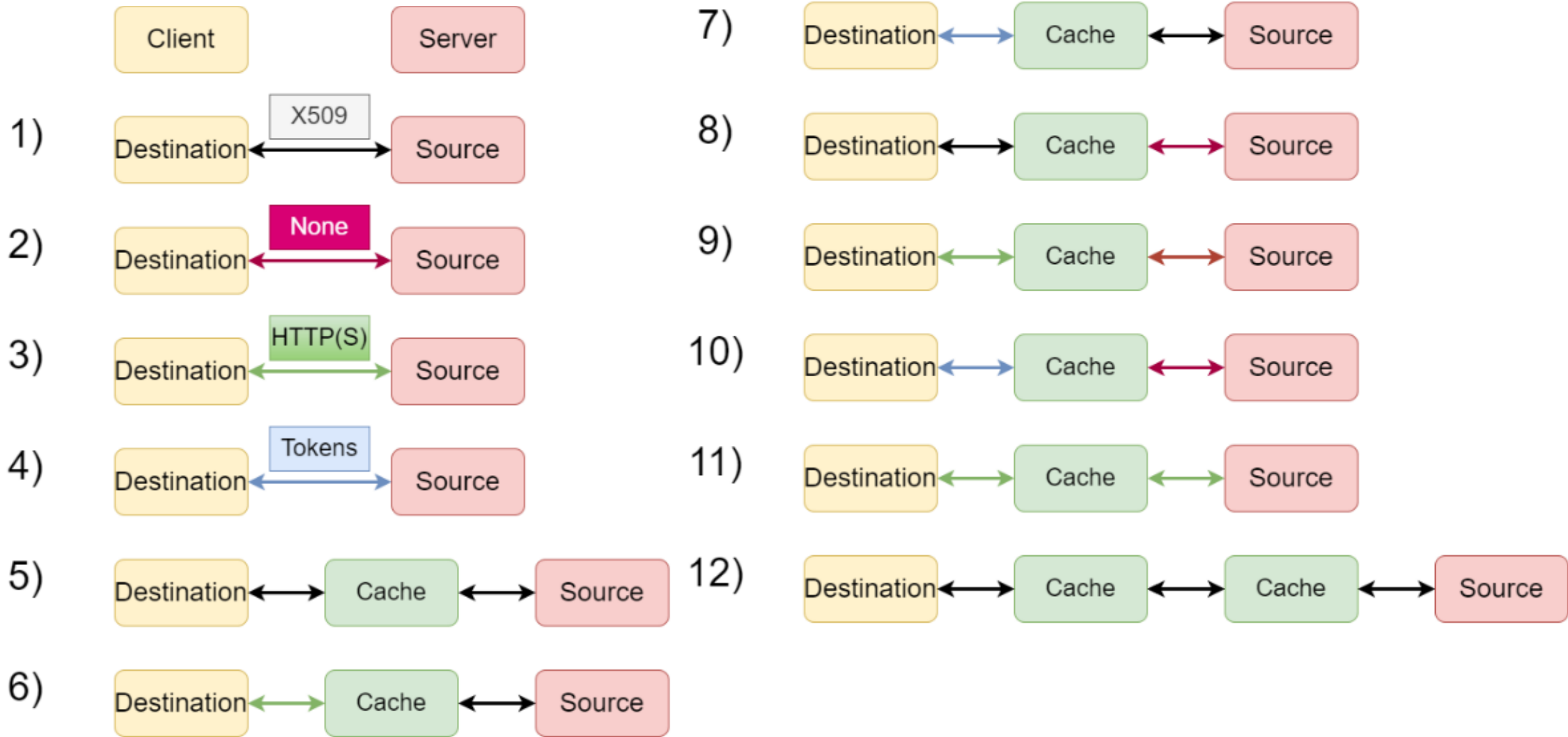
- *In my opinion* this is a similar situation to when I worked on Ganga.
 1. Large tool with large codebase & many uses.
 2. Many communities using it to solve their problems.
 3. Normally works extremely well.
 4. Highly configurable with many plugins.
 5. Not every community is running bleeding edge clients/versions.
- Testing is difficult because the phase-space is so large.
 - > 3 large dimensions; client version, server version & network topology
 - > many compact dimensions, plugins options, server options, expected pass/fail

Setting the Scene

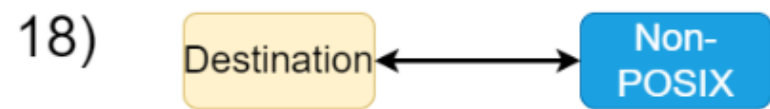
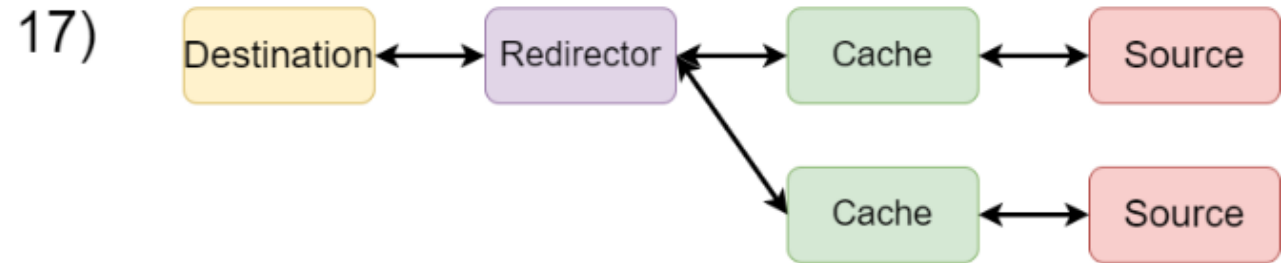
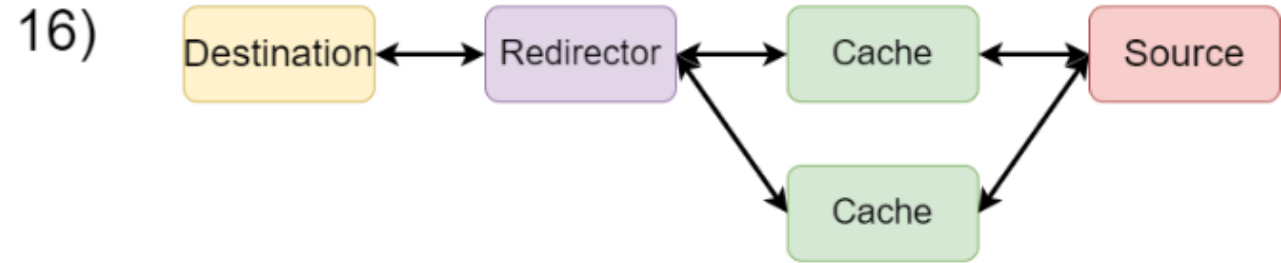
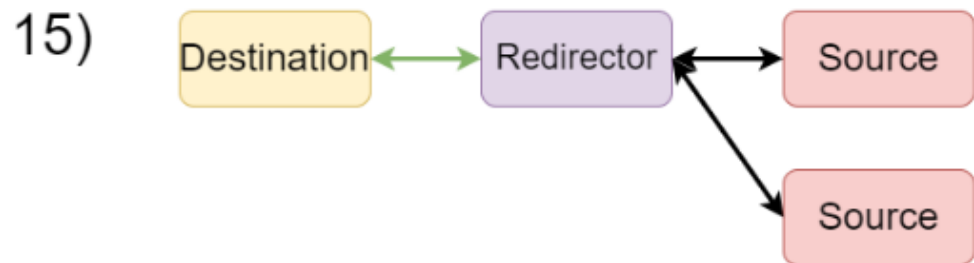
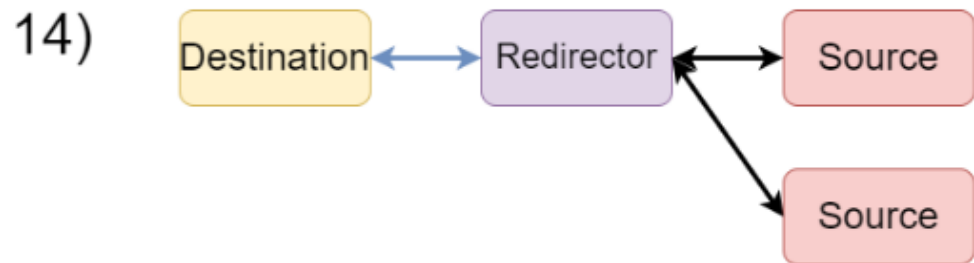
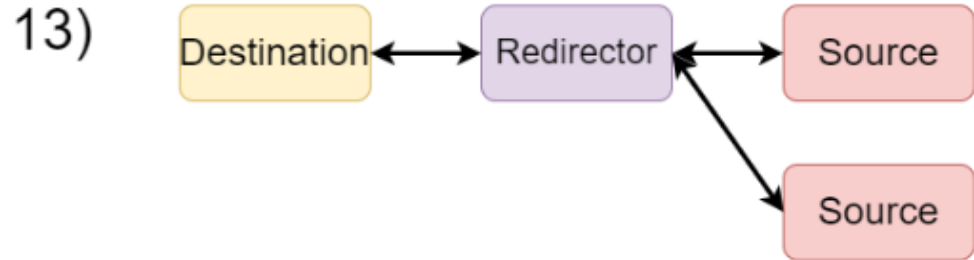
- Larger UK sites use XRootD in different ways.
Different features on/off for different topologies...
- Quite a few sites compile/patch/provision their own version for one reason or another.
Not expected to change in short-term, so taken as written.
- Question that has come up in UK storage meeting ***a few times*** is:

“What is the golden version where this config (last) worked?”

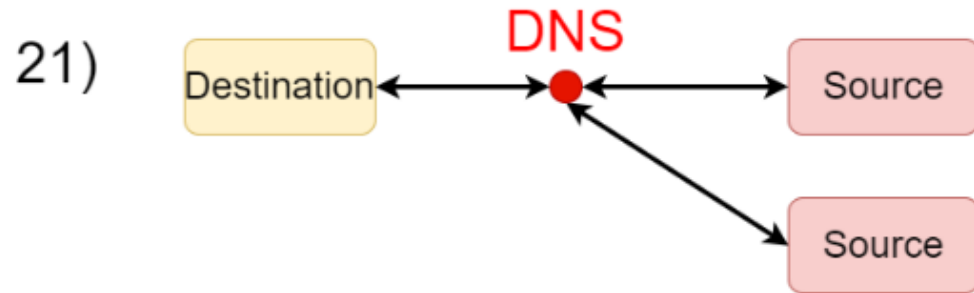
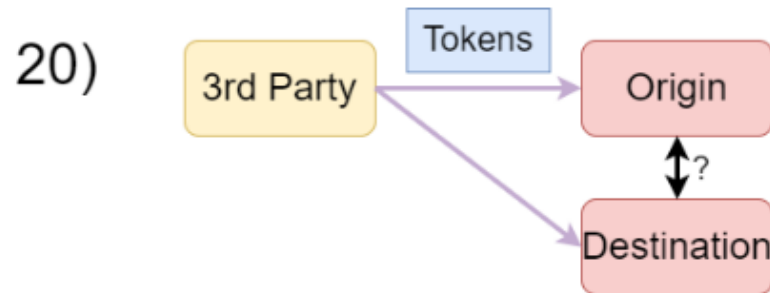
How much do we want to test?



How much do we want to test?



How much do we want to test?



The topology of a “typical XRootD install” seems to vary even within UK.

Would be good to try and identify the key components of this.

Want to test/check/know-how-to-use all features and best practice(s).



Test Management

- **XRootD** Integration Testing requires 2 parts:

Client:

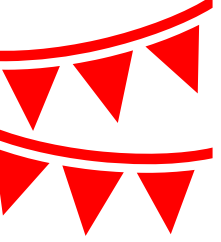
- Test cmdline tools (xrdcp, xrdfs, ...)
- Test Python3 client API(s)
- Double-Check data transfers work as expected
- Should we be testing C++ API?

← **Yes/No ?**

Server:

- Want to check server works as expected (logs/output)
- Want to test read/write transfers work as expected
- Check server-side features work as expected (on/off)

Containers to the Rescue!



Good news! It's 2024; Containers Exist



- Containers make deploying complex software stacks easy.
- Makes setting up (*test*) environments more reproducible.
- Can be used as sandboxes.
(shameless plug for one of my other projects here: <https://github.com/gridpp-Edi/appBox>)
- However, containers aren't the full story...
- Containers require configurations, network plumbing, namespaces...



XRootD is already used in Containers

- **We need a minimal container for testing!**
- Container design often ends up optimizing for 1 of 2 goals:

- **Deployability:** *



Container design used by **perfSonar**, **Gitlab**, **XRootD4CMS**.
Deploying several services within a single container.
Not-so-great for seeing what's going on, debugging, or fixing/testing...

- **Reproducibility:**

This is what you see in more commercially supported containers.
Closer to the UNIX philosophy of “do one thing and do it well”
Minimal, **great for testing**.



* Yes, I just made up a word...

XRootD Package/Image Management

- Why is this important? Containers are backed by images;

We are now 'rolling our own' container-images:

1. Using the **rpm** build recipe from the XRootD github repo (standing on the shoulders of giants!)
2. Built rpms from source on Alma9 base image(s)
3. Packages installed via dnf with all *normal* extensions for XRootD and dependencies
4. Image is tagged with release version
5. New images published to dockerhub
6. **No security/configuration/gremlins baked into images**


IF someone else does a better job we can use their images(!).

Deploying these containers means we have additional runtime control how we mount in CRL/config/data/cute-cuddly-kittens from our host into the container.



Service Management

Container Orchestration

- OK, we have an image, so can launch containers/run-tests. 
- We started with **docker-compose** to manage multiple services.
- This ended quickly.

- Setting up a single transfer of:

POSIX → PFC → Destination using x509 authentication doesn't work *



- Docker/Podman(-compose) aren't really setup in a way that makes full-fat x509 based security easy/happy...
(I don't expect tokens to be any easier)

*well, not easily, nicely...

Service Management

Let's fix the problem of complex container management, with... more containers!



Service Management (2)

- Each XRootD service needs the following:
 - CRL/VOMS mounted/updated from host
 - Server config mounted from host
 - Test data mounted from host *
 - DNS entries pointing to instance
 - Hostcert mounted from host (per-instance)
 - External network connectivity



- After evaluating a few options, we decided to **go with Kubernetes**

*So far, biggest use-case is POSIX, but plan to test CEPH-FS



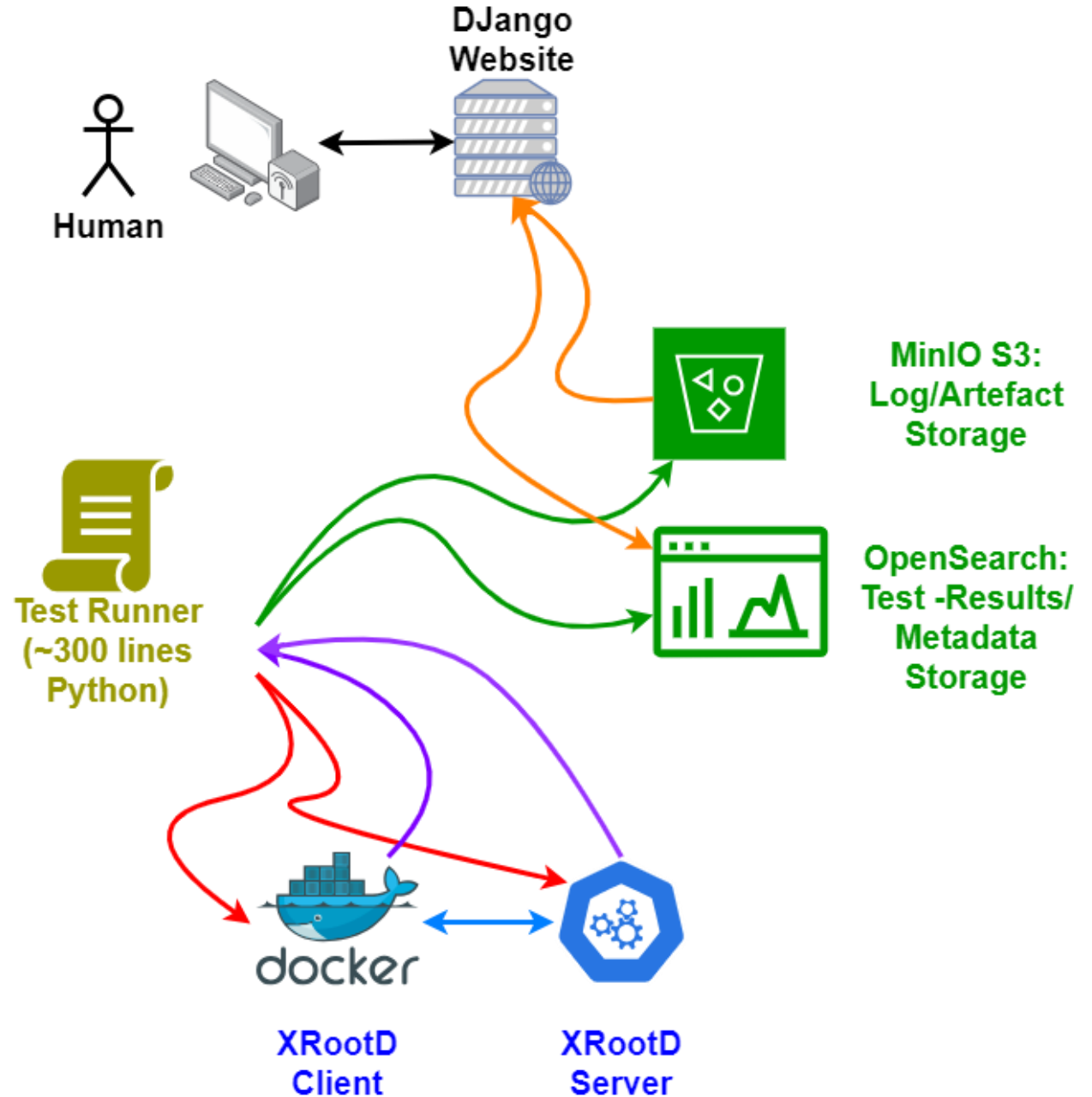
“There’s an API for that!”

- Almost everything “speaks” Python3 these days.
(The less we code, *the less we debug*, trying to keep things minimal)
- **Kubernetes**, **Docker**, **S3**, **OpenSearch**, **Django**, ...
- Most of the ‘*heavy lifting*’ for projects like this has been done for us.
- With that in mind, we decided to start working out what to do.
- Not *all work* is in Python3... but enough.

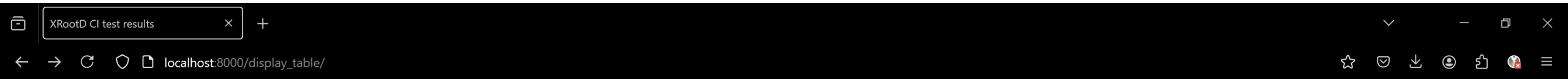
Running Tests

The Plan...

Testing Strategy
1) Deploy Configuration and Launch Containers
2) Wait for tests to run
3) Collect container artefacts
4) Store Test Results/Logs
4) Display Results to User



What do we have so far?



XRootD Test Results

client_image	server_image	client_output	server_output	testTime	testName	testStatus	@timestamp
gridppedi/xrdtesting:xrd-v5.7.0	gridppedi/xrdtesting:xrd-v5.7.0	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_clientOutput.log	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_serverOutput.log	2024-07-22T15:40:36.595529	read.py	GOOD	2024-07-22T15:40:36.595820
gridppedi/xrdtesting:xrd-v5.7.0	gridppedi/xrdtesting:xrd-v5.7.0	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_clientOutput.log	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_serverOutput.log	2024-07-22T15:31:27.728273	read.py	GOOD	2024-07-22T15:31:27.728584
gridppedi/xrdtesting:xrd-v5.7.0	gridppedi/xrdtesting:xrd-v5.7.0	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_clientOutput.log	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_serverOutput.log	2024-07-22T15:32:48.912504	read.py	GOOD	2024-07-22T15:32:48.912898
gridppedi/xrdtesting:xrd-v5.7.0	gridppedi/xrdtesting:xrd-v5.7.0	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_clientOutput.log	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_serverOutput.log	2024-07-22T15:35:25.426305	read.py	GOOD	2024-07-22T15:35:25.426606
gridppedi/xrdtesting:xrd-v5.7.0	gridppedi/xrdtesting:xrd-v5.7.0	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_clientOutput.log	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_serverOutput.log	2024-07-22T15:40:20.524026	read.py	GOOD	2024-07-22T15:40:20.524350
gridppedi/xrdtesting:xrd-v5.7.0	gridppedi/xrdtesting:xrd-v5.7.0	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_clientOutput.log	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_serverOutput.log	2024-07-22T15:46:49.528588	read.py	GOOD	2024-07-22T15:46:49.528985
gridppedi/xrdtesting:xrd-v5.7.0	gridppedi/xrdtesting:xrd-v5.7.0	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_clientOutput.log	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_serverOutput.log	2024-07-22T15:51:34.707189	read.py	BAD	2024-07-22T15:51:34.707649

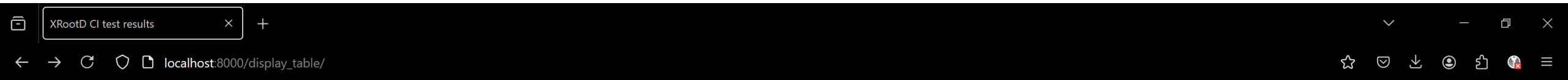
What do we have so far?

(Not bad for <100 lines of Python!)

Containers on
DockerHub

Test Client & Server
logfiles on (*private!*) S3

Test Metadata,
success/fail,
timestamps, ...



XRootD Test Results

client_image	server_image	client_output	server_output	testTime	testName	testStatus	@timestamp
gridppedi/xrdtesting:xrd-v5.7.0	gridppedi/xrdtesting:xrd-v5.7.0	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_clientOutput.log	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_serverOutput.log	2024-07-22T15:40:36.595529	read.py	GOOD	2024-07-22T15:40:36.595820
gridppedi/xrdtesting:xrd-v5.7.0	gridppedi/xrdtesting:xrd-v5.7.0	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_clientOutput.log	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_serverOutput.log	2024-07-22T15:31:27.728273	read.py	GOOD	2024-07-22T15:31:27.728584
gridppedi/xrdtesting:xrd-v5.7.0	gridppedi/xrdtesting:xrd-v5.7.0	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_clientOutput.log	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_serverOutput.log	2024-07-22T15:32:48.912504	read.py	GOOD	2024-07-22T15:32:48.912898
gridppedi/xrdtesting:xrd-v5.7.0	gridppedi/xrdtesting:xrd-v5.7.0	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_clientOutput.log	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_serverOutput.log	2024-07-22T15:35:25.426305	read.py	GOOD	2024-07-22T15:35:25.426606
gridppedi/xrdtesting:xrd-v5.7.0	gridppedi/xrdtesting:xrd-v5.7.0	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_clientOutput.log	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_serverOutput.log	2024-07-22T15:40:20.524026	read.py	GOOD	2024-07-22T15:40:20.524350
gridppedi/xrdtesting:xrd-v5.7.0	gridppedi/xrdtesting:xrd-v5.7.0	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_clientOutput.log	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_serverOutput.log	2024-07-22T15:46:49.528588	read.py	GOOD	2024-07-22T15:46:49.528985
gridppedi/xrdtesting:xrd-v5.7.0	gridppedi/xrdtesting:xrd-v5.7.0	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_clientOutput.log	pythonTestOutputs/read.py_C_xrd-v5.7.0_S_xrd-5.6.2_serverOutput.log	2024-07-22T15:51:34.707189	read.py	BAD	2024-07-22T15:51:34.707640

What do we have so far?

- Simple, *entirely dynamically generated* web-UI.

Not **yet** public, plan to add some basic auth.

Have seen bugs on the grid with credentials leaking into logging streams...

← **Input Welcome!**

- Using a github organization for managing the various pieces of this:

<https://github.com/gridpp-Edi>

- **Tests repo:**

<https://github.com/gridpp-Edi/xrootd-ci-tests>

*(Still empty as of
August 2024 😞
aiming for initial
tests before CHEP)*

- **Server configs repo:**

<https://github.com/gridpp-Edi/xrootd-helm-charts>

*Starting to populate this repo
for testing 😊*

Site Perspective

From the Site's Perspective

- On the face of it, this has *lots* of moving parts:

DNS, VOMS, Kubernetes, multiple new systems to update/maintain, S3, OpenSearch/ElasticSearch, message queues, credentials...

- However;

All these services are being re-used by some other project.
Not just throwing up lots of services for a single goal.

From the Site's Perspective

- Work on this allows us to:
 1. Support the in-development protoDUNE DAQ offline monitoring
 2. Support DUNE-DM monitoring
 3. Support GridPP-FTS monitoring
 4. Support UoE PPE-Labs clean-room certification
 5. Gain valuable experience with Kubernetes
 6. Support GridPP storage efforts

Conclusions

Conclusions

- Have successfully run initial tests against XRootD using our pipeline.
 - Data transfers in/out with x509 auth using containers.
- Have worked out most of the annoying bits in setting this up.
- Have a minimal web-UI which we aim to share ASAP

Conclusions – Next Steps

- Need to expand our testing topology (helm charts).
 - So far have server-side configs for XRD-POSIX and XRD-PFC.
 - Only testing X509 auth but want to do more.
- Need to flesh out some additional tests.
 - Successfully written/read data from POSIX.
 - Want to automatically test 3rd-party copy between containers.
- Need to work-out best way to handle version-dependence in testing.
 - Images are tagged with RPM release versions
 - How far forward/back do we go for compatibility testing?



THANK YOU FOR
LISTENING

Who, Where, What/Why, When?

- Edinburgh; Rob & Wenlong
- Integration Testing of XRootD in production-like environments
- Integration Testing \neq Unit Testing \neq Build Testing \neq Code Analysis
- Now-ish maybe / tomorrow probably

Where do common problems come from?

Client:

1. Client code initializes
2. XRootD client initialization spawn's worker threads (pool)
3. Transfers/Access work by queueing task to be run by worker
4. Tasks in queue get launched according to internal XRootD logic
5. Workers **must complete** and **correctly** return
6. Client code disconnects and cleans up workers

Where do common problems come from?

Server:

1. Server launches workers and connects to external port
2. Incoming connections spawn server-side work
3. Server responds with the result of its work
4. Server keeps connection open **until client disconnects**
5. Server keeps listening for new incoming connections

Why not just use a CI manager?

- Could have implemented CI stack on Jenkins or similar, however;
- Services such as this are regularly an interest to attackers.
- Requires boiler-plate to setup with K8S or equivalent.
- Yet another big complex service...
- Felt disadvantages of relying on such a stack are outweighed by the fact we can replicate most of this in <<1k Python3 boiler-plate.

Why not just use a CI manager?

- Want to make results semi-public, so efforts needed to sync/store/access data from Open/Elastic-Search, s3.
- Want to make logging artefacts semi-public.
(I'm assuming there's a risk here so unique creds/certs for testing)
- Ideally would like to receive ideas/input for future tests.
- Ideally would like to receive ideas/input for future topologies/helm-charts.

Why not just use a CI manager?

- 3 large dimensions in testing, client-version, server-version and topology...
- Considering using a CI-manager “on-top” of everything else to track/manage running of our test-infrastructure.
- Probably just going to use Jenkins as a “Web-managed crontab”

Kubernetes Setup

Operating in a limited environment.

- Minimal IP availability within (almost full!) shared VLAN
- Limited hardware (VMs all the way down)
- Nobody is 100% working on this 24/7

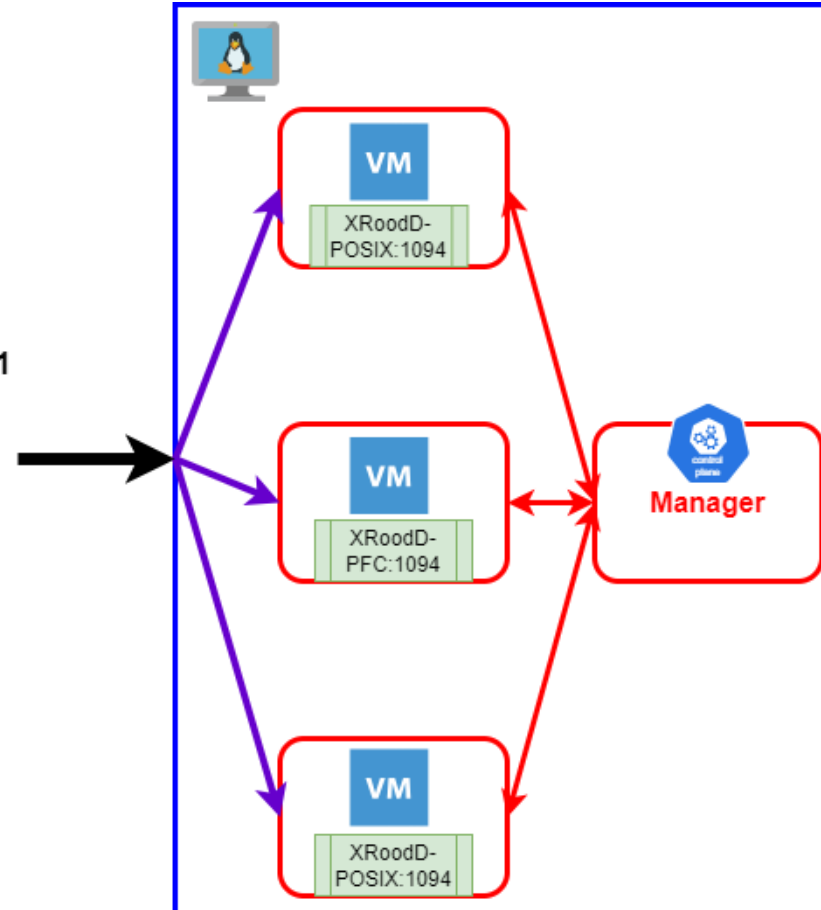
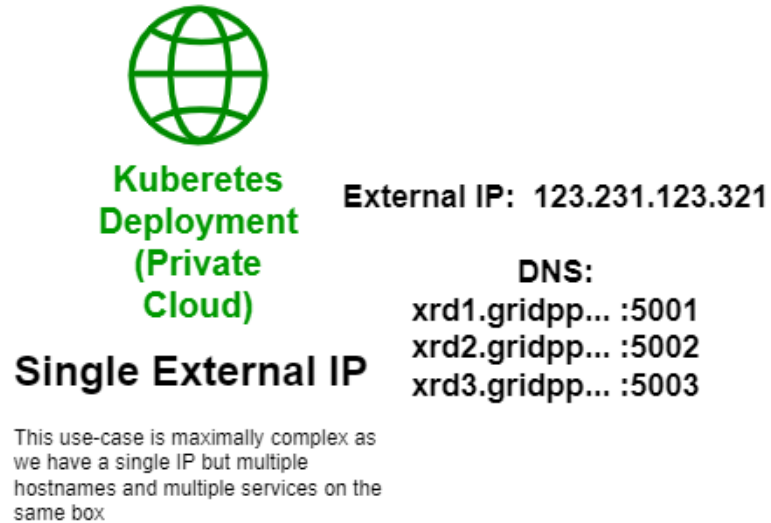
But:

- We manage our own DNS
- Have experience with containerization
- We're sharing time/effort between projects

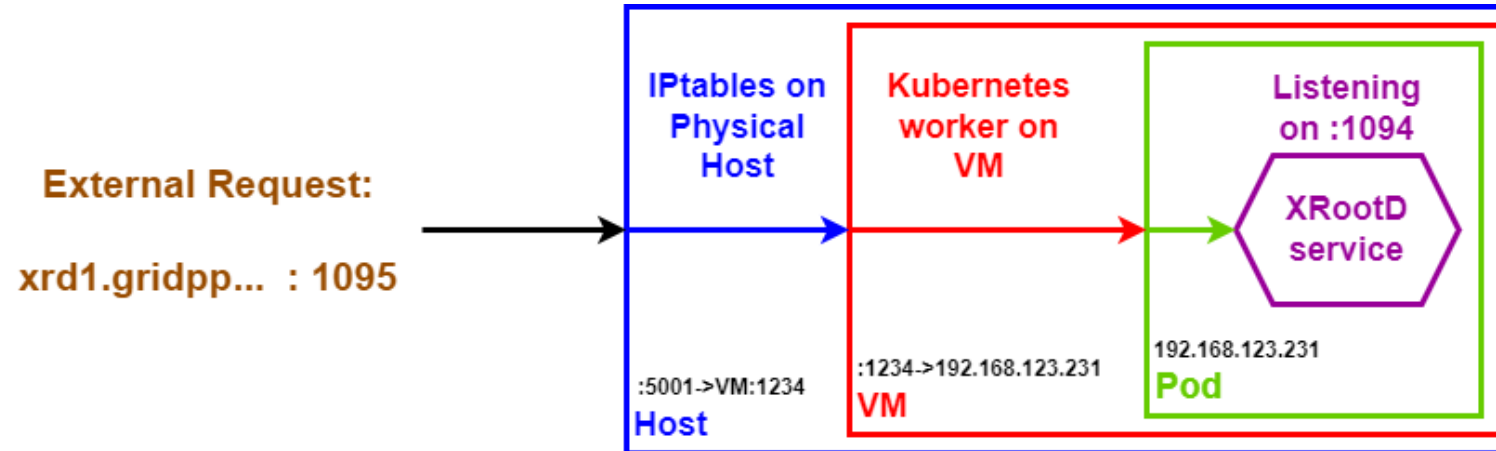


[This Photo](#) by Unknown Author is licensed under [CC BY-SA-NC](#)

Kubernetes – Setup



Kubernetes – Networking



Test written to be Generic; use any host:port combination from client-side.

Different tests expect POSIX backed XRootD on :1095, or a PFC on :1095, or a Redirector on :1095, ...

Service config just uses "normal" port(s), can be re-used in modular way.

Service doesn't care about what VM it's running on, or what the external port is.

Helm chart(s) config used to set correct host credentials.

From the Site's Perspective

With the experience from this work, we plan to setup a larger Kubernetes instance to support our other work.

We have ~40 containerized services on multiple hosts. This is already getting slightly difficult to manage.

Gaining more and more experience with nginx as a tool for managing traffic.
(if only it “spoke” root:// ...)

Having access to our own DNS means we've been able to test/adapt quickly as our equipment is physically split over 4 VLANs and 3 datacenters.

– Also looking into software-based networking to “save the day” with the fact our kit is diversely spread out.