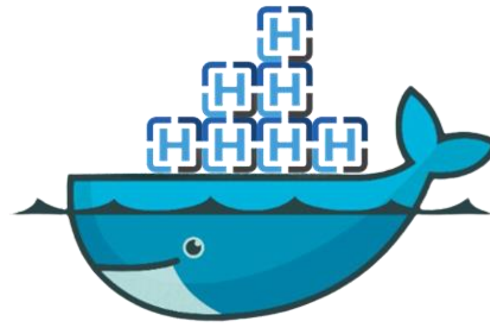


Scaling and organizing applications with the usage of Orthanc router containers

ORTHANC CON 2023

GHU Solidario
Advancing Social Medicine

Introducing myself

- Diego Victor de Jesus.
- 29 years old.
- Software engineer, since 2016.
- Working for Natixis, a french bank.
- Based in Porto, Portugal.

Router containers – An introduction

How I started using Orthanc:

- Hired in 2018 by a client who needed a PACS to manage medical images and integrate medical devices. Client had an existing intranet system who received studies as uploaded files.
- Existing system was adapted to integrate with Orthanc and use it as a PACS. Orthanc was compiled and installed inside the CentOS server, which used HDs rather than SSDs.
- Solution was good and patient modalities were successfully integrated.

Motivations of the presented solution:

1. Writer instance was overwhelmed with too many studies.
    1. Unavailability due to database locks caused by conflict between incoming studies and deletion routine. Almost completely freezing Orthanc.
    2. Big studies (CT/MR) blocking small (possibly urgent) ones (CR/DX), especially when resent (OverwriteInstances = true).
2. Writer instance bloated with unnecessary studies.
    1. No cold/hot storage strategy meant old studies were slowing down Orthanc.

Besides, a solution had to consider two problems:

- Adaptations had to be done on the server side as to avoid any hassle to technicians, who didn't know much about medical devices configuration. Defining multiple writer configurations on medical devices was not an option.
- The optimal solution had to be fully compatible with the DICOM protocol.

The solution – Router container

PACS applications built on top of Orthanc often consist of:

- **Monolith** Orthanc - A single Orthanc dedicated to **reads** and **writes** (optionally containerized).
- **Decoupled** Orthanc (optionally containerized)
  - An Orthanc container dedicated to **reads**
  - An Orthanc container dedicated to **writes**

Assuming **reader** and **writer** as two types of Orthanc containers, this document introduces a third type of Orthanc container, the **router**. A **router** container defines a **DICOM gateway layer** that can prevent overloads in writers instances, thus improving application responsiveness to degrading scenarios. Such container has two main responsibilities:

1. **DICOM study ingress –** Block reception according to the defined **router discard filter**. As per this doc example, the filter is:
   1. **Modality** not null
   2. **Institution name** not null
2. **DICOM study forwarding** – Forward studies to all **candidates** matching its data, based on the candidate's **routing criteria**. As per this doc example, the candidates are:
   1. **Generic writer** – An all purpose writer with routing criteria -> ROUTING_CRITERIA=**MODALITY\|NOT_IN\|CR,DX**
   2. **X-ray writer** – A writer dedicated to x rays with routing criteria -> ROUTING_CRITERIA=**MODALITY\|IN\|CR,DX**
   3. **Backup writer –** A special writer subcategory that receives any study by not defining any routing criteria.

The composable nature of the routing criteria makes it possible for one study to be sent to multiple writers.

The solution – Router container

The router container acts like a DICOM gateway, centralizing all DICOM ingress logic into a single point of concern. This simplifies the implementation of the writers, allowing them to focus in the actual storage of the studies, rather than deciding whether a study should be stored.

Moreover, we add Docker (+ Compose) to the architecture to achieve greater infrastructure flexibility, allowing quicker reaction to scenarios such as scaling needs or downtime mitigation.

The following terms define key aspects of the architecture:

- **Candidate** - A writer instance that is registered within the router to receive studies.
- **Routing criteria** - Defined by a candidate, used by the router to determine which candidates should receive a study. It is structured as follows:
    - **ATTRIBUTE-OPERATOR-VALUE**, where:
        - **ATTRIBUTE** is one of the possible values inside the **RoutableAttribute** enum
        - **OPERATOR** is one of the possible values inside the **Operator** enum
        - **VALUE** is an arbitrary value, in conformity with **OPERATOR**
- **Router discard filter** – Determines whether the router should accept a study.
- **Candidate determination** - The selection of a candidate by the router to receive an incoming study, based on a successful match with the routing criteria.

How it works – Overview

For this presentation, the following architecture is used:

Router – The central container, dedicated to controlling study ingress and forwarding.

Generic writer – An all-purpose writer with routing criteria -> ROUTING_CRITERIA=**MODALITY\|NOT_IN\|CR,DX**

X-ray writer – A writer dedicated to x-rays with routing criteria -> ROUTING_CRITERIA=**MODALITY\|IN\|CR,DX**

Backup writer – A special kind of writer that stores any incoming study due to having no routing criteria

How it works – Deep dive

1. The router container is the first to start. It exposes a **/candidates** POST endpoint for candidate registration.

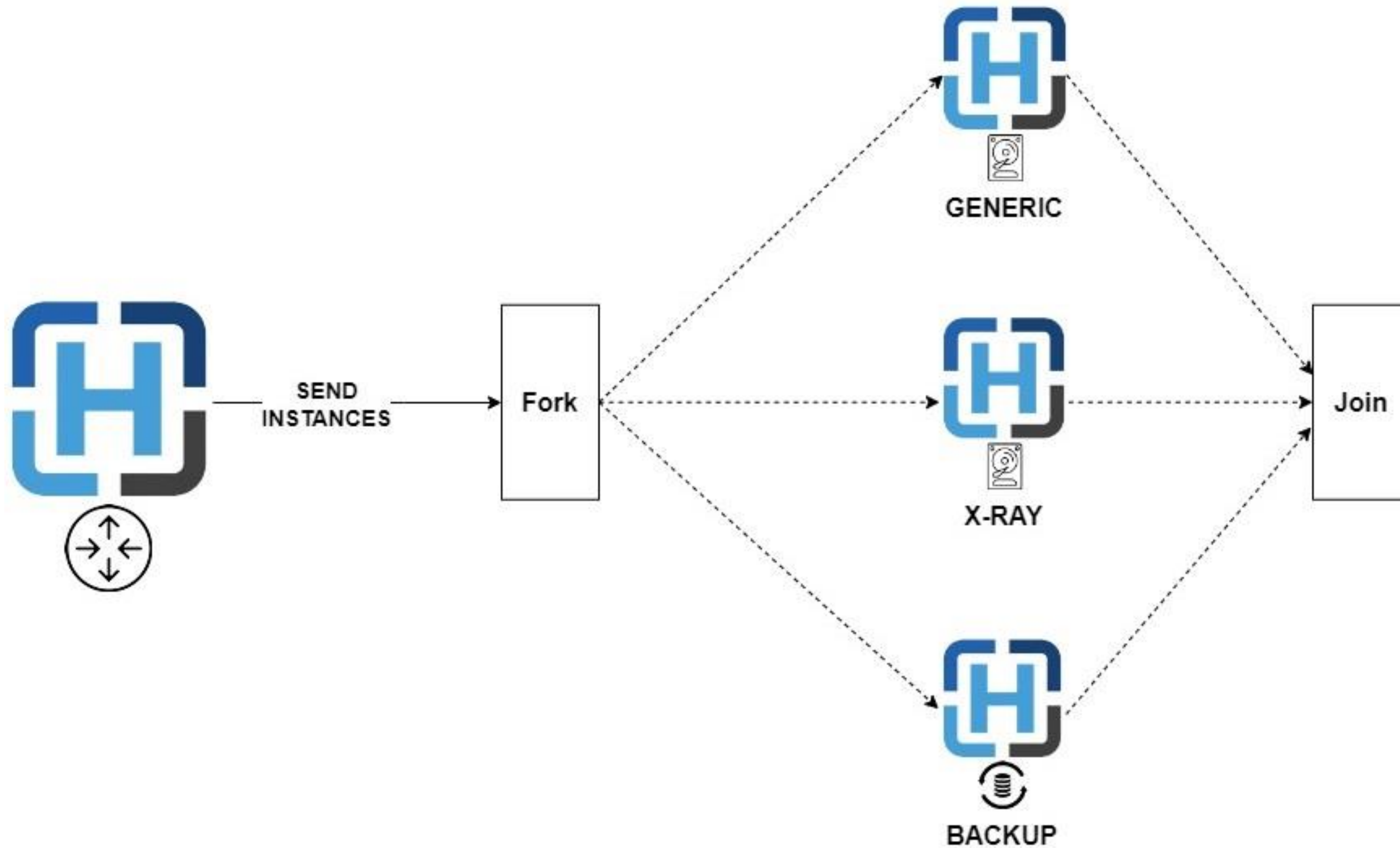
2. Writer instances attempt to register within the router upon startup.

3. Upon the arrival of a study, the router decides where it should go by checking its list of registered candidates. The candidates whose **routing criteria** matches the study will be determined by the router to receive it.

4. After determining the candidates, the router sends the studies in parallel to all candidates.

5. Router waits for all candidates to receive the instances by joining all threads.

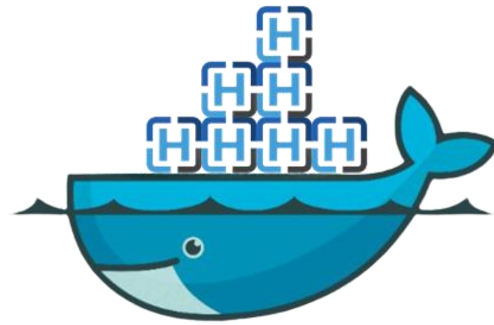
How it works – Deep dive



The results

1. Higher availability due to switching strategies for writers.
    1. No degradation from deletion routine.

2. Cold/hot storage strategies, extended by routing criterias.
    1. Proper storage of old studies. Improved compliance.
    2. Better performance in non-backup writers since studies are removed as soon they are validated.



Examples

Backlog

Short term:

1. Dynamic container creation
   1. OpenShift?
2. Candidate discard filter
   1. Discard filters for candidates for higher customization
3. Concurrency strategies
   1. Should the router join all threads?
   2. Should the router keep independent queues?
   3. Other ideas?

Long term:

1. UI to create containers based on consumers needs
2. Template architectures (archetypes)
   1. Production-ready DICOM setups
3. Stackable candidates
   1. A new writer subcategory (STANDARD, BACKUP, **STACKABLE**) of candidates that share the same database/filesystem and work in conjunction to process studies.
4. Canary deployments
   1. Spin up a new container and gradually route DICOM traffic from another instance to it. When all traffic is moved, redeploy the other instance.

Thank you!

Project GitHub - https://github.com/Diegovictorbr/orthanc-simple-router

LinkedIn - https://www.linkedin.com/in/dvjesus/

GitHub - https://github.com/Diegovictorbr

SO - https://stackoverflow.com/users/5042987/diego-victor-de-jesus