Service MXT-VHF pour DC0

Serge Du, Nicolas Leroy (LAL)
…avec l'aide précieuse de:
Laurent Michel et Nicolas Deparis (ObAS)
IAP – 23 Janvier 2019

historique et spécificités du projet

- équipe LAL pour DC0: 1 informaticien et 1 physicien
  - (autres membres impliques sur segment espace)
  - informaticien nouveau dans collaboration (Mai 2018):
    - environnement inconnu (outils, méthodes)
- objectifs pour DC0 forcément limités:
  - fournir un cadre qui permettra d'implémenter nos algorithmes
  - base pour un pipeline qui s'intégrera avec l'architecture générale FSGS
- … *objectif limité mais assez ambitieux* vu nos moyens et le délai très court (6 mois)

Immersion dans FSGS

  - Pas si compliqué…
- Merci GitLab et Slack
  - Quantité de code, documents, exemples
    - Profite pleinement de l'expérience des 'anciens'
- Présentations KP
    - Juillet 2018 APC
- Framework general pipeline-bricks par équipe ObAS.
  - Facillite l'apprentissage par la pratique: exemples, test, maquettes,...
  - Vision globale de ce qui est attendu:
    - Ensemble d'applications co-operantes.: Orchestrator, Serveur NATS, ConfigDB, interface web
  - => Décision d'utiliser au maximum les pipeline-bricks comme base de développement.
  - Accélérateur pour le demarrage du projet MXT-VHF.

pipeline-bricks: vers une première instance de pipeline.

- sortir du cadre des exemples pour créer un pipeline spécifique n'est pas si facile:
  - Il faut un niveau de compréhension minimum de l'architecture pour réaliser un projet externe et autonome.
  - …approfondir l' étude des exemples fournis.

- vers un projet basé sur pipeline-bricks, dépendant mais pas trop…
  - Objectif: construire un projet autonome (GitLab),
  - utilisant pleinement les composants et fonctionnalités des pipeline-bricks
  - qui reste compatible avec l' évolution des bricks
  - qui conserve son propre cycle de développement.

pipeline mxt-vhf : autonomie et dépendances

vers un projet basé sur pipeline-bricks, dépendant mais pas trop…

- Solution: identifier et cloner les composants spécifiques a MXT-VHF dans un projet autonome (pas un sous-module)

1. la configuration:
   - workflow.json: liste des taches (modules) et paramètres
   - entrypoint.py: gestion de la config (install workflow mxt-vhf, definition environnement)
2. les modules eux-mêmes: code python, coquilles 'vides' pour l'instant.
3. les outils pour construire et déployer le service:
   - dockerfile, docker-compose,…

construction du container

- On dépose les composants specifiques dans le répertoire 'standard' mxt/pipeline.
    - Dockerfile, entrypoint, modules (code source)
- On utilise la même technique de construction basée sur swagger:
    - Scripts de génération swagger standard mxt/pipelines
- On obtient une appli conforme ‘mxt/pipelines’:
    - scheduler avec gestion de taches
    - Interface RES , messages NATS
- ….. Question: compatibilité avec évolution du projet mxt/pipelines ?
  - Aucun changement nécessaire sur les composants spécifiques MXT-VHF.

Pipeline MXT-VHF: workflow, modules et dépendances

- Le workflow mxt-vhf:
  - reception_message ==> *(VoEvent, flux VHF)*
  - photons_extraction
  - fits_feed
  - **start_MxtGpPipeline ==> MXT-X-Band General Program**
  - crosscheks_positions
  - update_BA
- Interaction avec MXT-X-Band GP par l'interface REST:
  - démarrage MXT-GP: POST /processing
  - identification fin de traitement: GET /status

    (essais NATS pas concluants)

Détail: interaction REST MXT-VHF et MXT X-Band GeneralProgram

(point technique… on peut passer)

- start_MxtGpPipeline.py:
  1. URL du service definie par environment var.
     - pipe_tools.get_base_url (pipeline)
  2. démarre un processing par une requete POST
     - url = base_url + 'processing'
     - data='{api'd' : '00810300005' , 'target': '2', 'orbit' :'2', 'mode': 'GeneralProgram' }
     - r = requests.post(url,json=data)
     - r_json = r.json()
     - **procid = r_json['processing_descriptor']['process_id']**
  3. Attente fin de traitement
     - data={'seqid' : procid}
     - r = requests.get(url, params=data)

Controle, Monitoring, Messagerie:

- monitoring processing MXT-(Band-X)-GP: service Mxt-Orchestrator standard
    - Infos NATS (via paserelle WebSocket)
  - Orchestrator extensible, OK pour MXT-VHF:
    - deja experimente, mais non maintenu pour l'instant
- MXT-VHF utilise des scripts simples et autonomes curl/python
    - start processing, get report
    - mxt-vhf/tools/REST/CURL/mxtvhf-start.sh
    - python mxt-vhf/tools/REST/pipeline_info.py 0 mxt_vh
- Gestion des messages NATS est globalement réalisée par le scheduler des pipeline-bricks
    - => monitoring via Web Sockers (Orchestrator)
- ajoute un monitoring de messages specifiques MXT-VHF en version dev:
  - python mxt-vhf/tools/NATS/log_queue_mxt.py

Service SWARM: déploiement et configuration

- évolution récente vers une gestion des configuration 'standard':
  - recommandations Décembre 2018 a Strasbourg
    - uniformisation des règles de config pour faciliter intégration et déploiement
    - pas de variables d'environnement, fichier de config docker
  - dans le container on recréée l'env. qui est attendu par les appli pipeline-bricks
    - … cela reste invisible pour le 'deployeur’

Pour conclure

- Les objectifs étaient assez modestes, ils sont atteints:
  - application intégrée au contexte micro-services du FSGS
    - Construction, déploiement
    - interaction avec services généraux (NATS)
    - Coopération avec service spécifique (MXT X-Band GP)
  - structure prête pour implémenter les algorithmes de physique.

- Encore beaucoup de travail mais les bases sont maintenant bien établies.