Aplicació d'automatització i report dels serveis del Sincrotró ALBA

Anna Hidalgo Esteve

Resum– Avui dia el control de les versions de les dades s'ha convertit en una peça molt important a l'hora de mantenir qualsevol projecte. El risc de no portar cap registre de les versions pot suposar que perdem el control de les nostres dades i dels canvis produïts.
Es per això que aquest projecte consisteix en crear una aplicació que monitoritzi i mantingui actualitzades les diferents peces del software del Sincrotró ALBA. El major problema que es va trobar en referència al software del Sincrotró és que no estaven registrades les instal·lacions que s'anaven fent dels serveis existents a les màquines de les línies de llum, anomenades Beamlines. Per això mateix, el meu projecte soluciona aquest problema tant guardant les instal·lacions en una base de dades com comprovant que tots els serveis estiguin actualitzats a l'última versió.

Paraules clau– SaltStack, Docker, Docker Compose, Beamlines, PyQt5, Git, CLI, YAML

Resumen– Hoy en día el control de las versiones de los datos se ha convertido en una pieza muy importante en la hora de mantener cualquier proyecto. El riesgo de no llevar ningún registro de las versiones puede suponer que perdemos el control de nuestros datos y de los cambios producidos.
Es por eso que este proyecto consiste al crear una aplicación que monitorice y mantenga actualizadas las diferentes piezas del software del Sincrotrón ALBA. El mayor problema que se encontró en referencia en el software del Sincrotrón es que no estaban registradas las instalaciones que se iban haciendo de los servicios existentes a las máquinas de las líneas de luz, denominadas Beamlines. Por eso mismo, mi proyecto soluciona este problema tanto guardando las instalaciones en una base de datos como comprobando que todos los servicios estén actualizados a la última versión.

Palabras clave– SaltStack, Docker, Docker Compose, Beamlines, PyQt5, Git, CLI, YAML

Abstract– Nowadays data version control has become a very important part of the maintenance of any project. The risk of not keeping track of versions can cause us to lose control of our data and the changes produced.
Therefore, this project consists of creating an application that monitors and keeps updated the different parts of the ALBA Synchrotron software. The main problem that was encountered with respect to the Synchrotron software was that it did not record the installations that were made of the existing services to the beamline machines, known as Beamlines. Therefore, my project solves this problem by both storing the installations in a database and checking that all services are updated to the latest version.

Keywords– SaltStack, Docker, Docker Compose, Beamlines, PyQt5, Git, CLI, YAML

• E-mail de contacte: annahies1400@gmail.com
• Treball tutoritzat per: Eduardo César (Arquitectura de Computadors i Sistemes Operatius)
• Curs 2021/22

1 INTRODUCCIÓ - CONTEXT DEL TREBALL

1.1 Context

EL Sincrotró ALBA és una infraestructura científica de tercera generació i és la més important de la zona del Mediterrani. Es tracta d'un complex d'acceleradors d'electrons per produir llum del sincrotró, que permet visualitzar l'estructura atòmica i molecular dels materials i estudiar les seves propietats. [1]

Actualment, l'ALBA conté vuit línies de llum operatives, que comprenen raigs X, i que es destinen principalment a les biociències, la matèria condensada (nanociència i propietats magnètiques i electròniques) i la ciència dels materials, aquestes línies de llum s'anomenen Beamlines.

Aquestes Beamlines són utilitzades per diferents empreses i estan compostes de diferents màquines. Cada una d'aquestes màquines té un conjunt de serveis instal·lats, depenent de les condicions que demana cada empresa.

La problemàtica es troba en el fet que el software que s'està desenvolupant actualment s'ha d'instal·lar a les diverses màquines i existeix una evident heterogeneïtat d'aquest. En lloc de tenir una imatge amb tot el software hem de fer diferents capes de software que requereixen de capes anteriors. Cada màquina té un hardware diferent, així doncs el que he d'aconseguir és monitoritzar i mantenir actualitzades les diferents peces de software.

El meu projecte doncs, consisteix en crear una aplicació que es comprovi que tots els serveis de cada màquina estiguin actualitzats a l'última versió, si no és així, actualitzarlos.

1.2 Objectius

El principal objectiu és crear un software que ajudi a la gestió de tot l'stack de software per l'adquisició de les dades del Sincrotró ALBA.

Es farà una anàlisi, disseny i desenvolupament d'aquesta eina gràfica, desenvolupant i creant la base de dades, fent una implementació del *backend* i de l'arquitectura del *frontend*. El sistema ha de ser capaç de gestionar l'automatització dels serveis que són instal·lats a les màquines trobades a cada *beamline*.

Objectius específics:

- Familiaritzar-me amb SaltStack "[2], [3]".
- Conèixer la gestió del software del Sincrotró ALBA.
- Realitzar el disseny de l'eina gràfica.

1.3 Metodologia

La metodologia és el conjunt de mètodes, tècniques i eines utilitzades amb l'objectiu de dur a terme una tasca.

Des del principi els requisits i objectius del projecte han estat molt clars, la qual cosa em va fer optar inicialment pel "Model de desenvolupament en cascada"[4], ja que és un procés de disseny seqüencial, en què el progrés flueix constantment cap avall a través de les fases d'anàlisi de requisits, disseny, construcció, proves, instal·lació i manteniment.

Tanmateix, aquest és també un desenvolupament inherentment incremental que s'adapta de forma natural a la metodologia en espiral [5], en la qual finalment he optat.

S'ha prés aquesta decisió, ja que el model de desenvolupament en cascada no permet produir canvis en una etapa posterior i en el meu projecte això suposaria un problema.

El model de l'espiral es caracteritza per composar-se de blocs repetitius. Aquests són temporals i fixes, on cada bloc s'anomena Sprint o iteració. Cada un d'aquests ha de donar un resultat complet per ser entregat. L'aspecte fonamental d'aquesta metodologia és la flexibilitat perquè en cada iteració sorgeixen noves idees o canvis i aquests es poden incorporar fàcilment.

Si aquest model és aplicat al desenvolupament de l'eina gràfica que es crearà podria entendre-ho com la guia que he de continuar durant totes les etapes del meu projecte amb l'objectiu de complir amb els requisits de les empreses.

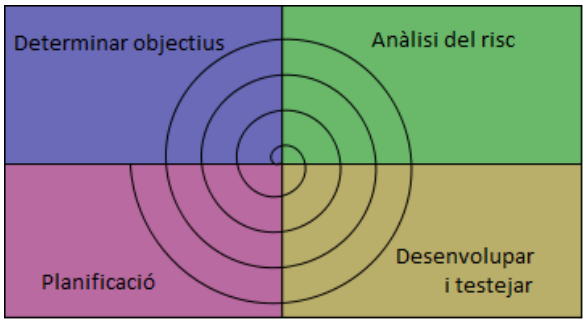

Fig. 1: Model d'espiral [5]

1.4 Planificació

Seguint l'anterior model escollit el qual es seguirà per aconseguir un projecte complet, el primer pas que s'ha realitzat ha sigut fer una anàlisi dels requisits que es demanen i elaborar un pla que permet aconseguir els objectius establerts. Així doncs discutir sobre les condicions del projecte i buscar les diferents alternatives que m'ajudaran a complir aquells objectius.

Per tal de començar a desenvolupar el *backend* vaig haver de documentar-me sobre *SaltStack* "[2], [3]", ja que per tal de fer el deployment del software s'han d'aplicar unes receptes des del repositori *git*. Aquestes receptes fan servir el sistema SaltStack. Aquest sistema és un software lliure i multiplataforma que es fa servir sobretot al Sincrotró ALBA. L'eina gràfica que es crearà no només consisteix en crear una base de dades sino que fa servir l'eina git per tenir la configuració que s'ha d'aplicar en cada host. Així doncs, també m'he hagut de familiaritzar amb el sistema de control de versions Git. "[6], [7]"

Un cop obtinguts els coneixements bàsics sobre les eines que permeten crear el sistema de *backend*, vaig prosseguir a dissenyar el format que ha de seguir la base de dades, ja que en ella s'han d'incloure totes les Beamlines amb les seves corresponents màquines i els serveis instal·lats en cada una d'elles. Tan aviat com vaig establir les bases del disseny vaig implementar la base de dades amb MySQL.

En acabar la fase de *backend* vaig haver d'implementar la fase de *frontend*, la qual consisteix en la creació de l'aplicació que es basa en un disseny i una implementació. Quan

la fase de comprovació del codi ha finalitzat i s'ha corregit la seva qualitat es passa a la fase de testeig on es somet a proves per assegurar-se el cumpliment de la funcionalitat desitjada. Finalment es passa a la millora dels errors que s'hagin pogut assolir.

Com finalment va ser escollida la metodologia del model espiral, no és necessari tenir tot el *backend* finalitzat per poder començar amb el disseny de l'aplicació. Tal i com es pot observar en el diagrama de Gantt mostrat a la Figura 2, en el cas de la creació de la base de dades i la creació de l'aplicació, va existir un solapament entre aquestes dues tasques.

1.5 Organització del document

L'estructura d'aquest article permet obtenir un coneixement tant de l'arquitectura interna com de les eines utilitzades de manera prèvia a l'explicació del desenvolupament del projecte, és a dir, de la creació del *backend* i del *frontend*, on es fa ús d'aquestes.

Per aquest motiu l'ordre a seguir és primer l'arquitectura interna de l'aplicació, el desenvolupament del projecte i finalment les conclusions extretes.

2 ARQUITECTURA

A continuació en la Figura 3 es detalla l'estructura interna de l'aplicació i s'explica com aquesta està implementada per complir amb els objectius del projecte.

Començant per la fase de *backend*, la qual està formada per un contenidor *Docker* [8] prèviament muntat el qual provee dos serveis: *MySQL* i *SaltStack*. Aquests serveis resulten ser les imatges del contenidor.

Dins d'aquest *Docker* s'ha creat la base de dades, la qual ha sigut creada mitjançant el sistema de gestió de bases de dades relacional *MySQL* amb l'ajut del software *SaltStack* [9].

Com ens mostra la Figura 3, observem que el software *SaltStack* està compost principalment pel *SaltMaster*, el qual té el control central de tots els sistemes de destí integrat. És a dir, actua com el centre de comandes i control pels *minions* i és des d'on s'executen les comandes.

El *SaltMaster* es dedica a administrar un o més serveis *minions* de *SaltStack*.

Els anomenats *minions* són els serveis que realment executen les aplicacions i serveis. Cada minion té un ID assignat (que es pot generar automàticament a partir del nombre de host de *minion*) i el *SaltMaster* pot fer referència a aquest ID per dirigir comandes a *minions* específics.

Seguidament, quan ja tenim creada la base de dades accedim al sistema *frontend* mitjançant un *Command Line Interface*.

Aquest *frontend* és la part de l'aplicació amb la que interaccionen els usuaris. En el meu cas és l'aplicació creada la qual proporciona una automatització dels serveis del Sincrotró ALBA.

Aquesta aplicació ha estat creada amb PyQt5, el qual pertany a la llibreria Qt pel llenguatge de programació Python.

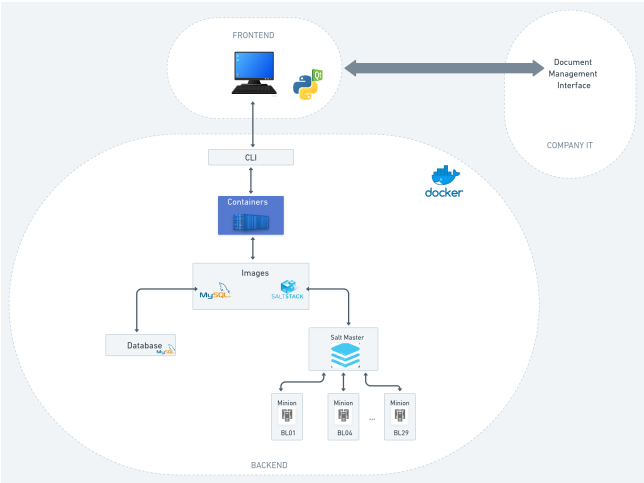

Fig. 2: Arquitectura

3 DESENVOLUPAMENT

En aquest apartat s'explica tant el desenvolupament com les eines que s'han utilitzat en les dues etapes que formen el projecte i s'han anat complint per tal de satisfer la planificació esmentada en l'apartat 1.4.

En primer lloc s'han analitzat els requisits que són necessaris per començar a desenvolupar aquest projecte. Per això, s'ha fet un estudi intern de la situació actual del Sincrotró ALBA i les problemàtiques a les que em puc enfrontar.

El principal problema que existeix és que en lloc de tenir una imatge amb tot el software s'han de generar diferents capes de software que requereixen de capes anteriors. Aquesta és una de les raons per les quals el Sincrotró ALBA no emmagatzema l'historial d'instal·lacions dels serveis de les *beamlines*.

Per tal de solucionar aquest problema s'ha desenvolupat el sistema de *backend*, el qual s'ha comentat a l'arquitectura del projecte.

3.1 Backend

Aquest sistema de *backend* està compost per 3 eines: *Docker* [10], *SaltStack* i *MySQL*.

3.1.1 Docker

Docker proporciona un model d'implementació basat en imatges, que permet compartir una aplicació o un conjunt de serveis amb totes les seves dependències en varis entorns. En el meu cas les imatges han sigut *MySQL* i *SaltStack*.

Docker és una plataforma la qual permet als usuaris empaquetar, distribuir i administrar aplicacions dins de contenidors. Usant Docker es pot estar segur de que l'aplicació s'executarà en qualsevol altra màquina Linux [11].

A més de ser fàcils de crear i eliminar, es pot treballar amb més d'un contenidor al mateix instant de temps sense que això suposi un sobreesforç de la màquina. Cada aplicació emmagatzemada en contenidors està aïllada de la resta, el qual permet un major control sobre la seva gestió i seguretat.

Un contenidor no pot accedir al que es guarda a un altre, això obre la porta a que els desenvolupadors puguin escollir quines tecnologies volen utilitzar pels seus serveis sense provocar cap modificació a la resta.

Dins de la pròpia tecnologia de Docker existeixen dues formes de fer el muntatge del servidor. Una forma és utilitzar un contenidor per desenvolupador i l'altra és l'ús de l'eina Docker Compose, la qual utilitza múltiples contenidors per un mateix desenvolupador. La principal diferència d'ambdos és que Docker Compose és una eina per definir i executar aplicaciones Dockers multicontenidor que permet l'ús de Docker a partir d'arxius YAML [12].

Aprofundint en l'estudi de les dues alternatives, he optat per Docker Compose, pel fet que amb Docker la gestió de la configuració de cada contenidor que es generi és més difícil ja que la instal·lació dels serveis que tindrá el desenvolupador depèn del fitxer de configuració Dockerfile.

En canvi, amb Docker Compose es permet llançar una sola comanda per crear i iniciar tots els serveis desde la seva configuració (YAML), aquest arxiu permet definir com

Docker executa les nostres imatges i és emmagatzemable amb el nostre codi per reutilitzar-lo posteriorment. En altres paraules, es poden crear diferents contenidors i al mateix temps diferents serveis a cada contenidor. [13]

Per crear el contenidor s'han hagut de seguir el següents passos [14]:

- Definir l'entorn de l'aplicació amb un Dockerfile.
- Definir els serveis *MySQL* i *SaltStack* de l'aplicació en docker-compose.yml.
- Executar docker-compose up per iniciar i executar l'aplicació.

Com he comentat anteriorment, el Docker està format per *MySQL* i *SaltStack* les quals he hagut de documentar-me per entendre el seu funcionament.

A la Figura 4 i la Figura 5 es mostra la configuració que s'ha hagut de seguir en el fitxer docker-compose.yml per definir els serveis.

```yaml
version: '2'
services:
  salt:
    hostname: salt
    privileged: true
    cap_add:
      - SYS_ADMIN
    environment:
      container: docker
    command: /sbin/init
    build:
      context: machine/master/
    ports:
      - "8000:8000"
    volumes:
      - /sys/fs/cgroup:/sys/fs/cgroup:ro
      - ./srv/salt:/srv/salt:ro
      - ./srv/pillar:/srv/pillar:ro
      - ./sicilia_installer:/sicilia_installer
    networks:
      - salt
```

Fig. 3: Implementació servei SaltStack en docker-compose.yml

```yaml
mysql_db:
  hostname: mysql_db
  image: "mysql:5.7"
  environment:
    MYSQL_DATABASE: 'salt'
    MYSQL_USER: 'salt'
    MYSQL_PASSWORD: 'password'
    MYSQL_ROOT_PASSWORD: 'password'
    MYSQL_ROOT_HOST: '%'
  ports:
    # <Port exposed> : < MySQL Port running inside container>
    - '30000:3306'
  expose:
    # Opens port 3306 on the container
    - '30000'
  networks:
    - salt
  volumes:
    - mysql-data:/var/lib/mysql
```

Fig. 4: Implementació servei MySQL en docker-compose.yml

3.1.2 SaltStack

Seguidament vaig haver de conèixer el funcionament del software que utilitza el Sincrotró ALBA, SaltStack, per aplicar les receptes esmentades anteriorment.

SaltStack és un software lliure i multiplataforma que permet realitzar el manteniment a distància, crear estats de destí prèviament definits i iniciar controls.

Utilitza un repositori central per realitzar canvis en els nous servidors i instal·lar software en entorns de IT, inclosos serveis físics i virtuals.

SaltStack automatitza les tasques administratives i d'implementació de codi repetides del sistema, eliminant els processos manuals d'una manera que pot reduir els errors que passen quan les organitzacions de IT configuren els sistemes.

El software funciona a través de reactors de *Salt*, *minions*, *grains* i *pillars*.

La configuració de *SaltStack* s'executa a través d'un arxiu de text en format YAML, els quals s'anomenen *pillars* i utilitza plantilles *Jinja* [15] per la definició de les configuracions de software. Aquest llenguatge permet la representació de dades estructurades de forma seqüencial.

SaltStack utilitza el model servidor-client, en el que un servidor de *Salt* emet comandes per un client i aquest últim les executa [16].

En l'ecosistema de *Salt*, el servidor s'anomena *Salt Master*, el qual té el control central de tots els sistemes de destí integrats. Així doncs totes les comandes i arxius es transmeten a través d'aquest.

Salt Master emet comandes per un o varis minions, que són nodes que poden ser instal·lats opcionalment els quals executen les comandes, principalment en llenguatge Python i informen de tots els esdeveniments i resultats rellevants. Aquests es registren amb una clau individual. [17]

En el moment que el *Salt Master* té el primer contacte amb els minions ha de confirmar la comanda. Això significa que es produeix una comunicació xifrada amb els parells de claus.

Quan un *minion* acaba d'executar un treball, envia les dades retornades del treball al *Salt Master*.

Els minions es comuniquen amb el *Salt Master* mitjançant dos ports predeterminats anteriorment. Aquests ports funcionen en conjunt per rebre i enviar dades al bus de missatges, ZeroMQ [18], el qual crea una topologia de xarxa asincrónica, és a dir, no esperen resposta, per proporcionar la comunicació més ràpida possible.

Com s'ha pogut observar en l'arquitectura del sistema, els minions resulten ser cada una de les *beamlines*, les quals estan compostes de màquines on cada una d'elles s'emmagatzemen els serveis.

Aquest software s'ha utilitzat per tal de poder comunicarse amb les línies de llum i les seves màquines corresponents.

Per tal de obtenir un bon resultat utilitzant SaltStack, s'han seguit els següents passos:

- Un cop iniciat el *Docker*, executem la següent comanda *"docker-compose exec salt bash"*. Amb aquesta el que obtenim és iniciar el *root salt*.
- Seguidament s'executa "salt '*' state.apply grains", entrant en profunditat, el que estem demanant és que per totes les màquines de totes les beamlines existents

('*') s'actualitzin els *grains*. Com s'ha pogut observar en l'arquitectura del sistema, els *grains* estan inclosos en cada un dels *minions*, aquests doncs, són els serveis existents en les màquines de les *beamlines* corresponents.

L'avantatge que tenim és que les comandes en *SaltStack* són sempre les mateixes.

Un cop obtinguts els coneixements del software *SaltStack*, prosseguim a implementar la base de dades amb *MySQL*.

3.1.3 MySQL

El procés de disseny i la creació de la base de dades va ser el que poder va comportar més temps. Una base de dades l'entenem com el conjunt de dades que pertanyen a un mateix concepte, en el meu cas el conjunt de *beamlines* amb les seves respectives màquines i els serveis existents en cada una d'elles. Aquestes dades han d'estar emmagatzemades per un posterior ús, en el meu cas, per crear l'aplicació d'automatització de serveis i obtenir un historial de les instal·lacions d'aquests.

Per crear la base de dades s'ha decidit utilitzar MySQL, ja que és un sistema de gestió de bases de dades relacional de codi obert.

A continuació es detalla el disseny de la taula de la base de dades a la Taula 1, tal i com es pot observar, només s'ha creat una taula en la base de dades, ja que volem emmagatzemar la informació de les actualitzacions dels serveis en una sola taula. Les columnes de les taules d'una base de dades es denominen camps, mentre que les files s'anomenen registres. Com podem observar a la Taula 1, els camps de la taula creada són:

- **ID** : És un *Auto Increment*, és a dir, genera automàticament valors numèrics seqüencials cada cop que s'insereix un registre en una taula. Aquest valor per defecte comença per l'1.
- **Domain** : És de tipus *varchar(255)*, vol dir que és una cadena no *Unicode* de longitud 255 bytes. Es refereix a la secció del Sincrotró ALBA a la qual pertany, en l'exemple de la Taula 1, *BL* significa *BeamLines*.
- **Subdomain** : És de tipus *varchar(255)*. Conté el nom de la *beamline* en la qual pertany la màquina en la que s'ha fet l'actualització del servei.
- **Host** : És de tipus *varchar(255)*. Conté el nom de la màquina en la qual s'ha fet l'actualització del servei.
- **Service** : És de tipus *varchar(255)*. Conté el servei que s'ha actualitzat.
- **Version** : És de tipus *varchar(255)*. Es refereix a la versió que s'ha instal·lat.
- **Who** : És de tipus *varchar(255)*. Conté l'usuari el qual ha realitzat l'actualització.
- **Installed** : És de tipus Datetime. Conté la data i l'hora en la qual s'ha realitzat l'actualització.

TAULA 1: TAULA BASE DE DADES

ID	Domain	Subdomain	Host	Service	Version	Who	Installed
1	BL	BL01	pcbl0101	tango	2021202	User	10/10/2021 (09:26:03)
2	BL	BL04	pcbl0401	taurus	2020212	User	15/01/2022 (07:40:33)

Per complir amb el disseny de la base de dades s'ha hagut de seguir el procés següent. Per cada servei s'ha creat un fitxer, anomenat init.sls, en el qual s'ha implementat una crida amb el llenguatge de programació *Jinja* [8], on li demanem que es cridi al script *Python* en el qual es crea la taula de la base de dades amb *MySQL*, entrant els paràmetres corresponents de cada màquina.

Aquesta crida s'activa quan executem l'anterior comanda *'salt '*' state.apply BL.BL01'*. En aquest pas, en comptes d'actualitzar els *grains* el que li demanem a *Salt* és que per la *Beamline* anomenada *BL01*, observi els seus *grains* per cada màquina existent en aquesta.

Així doncs, el que es produirà és que s'examinarà el fitxer *init.sls* de cada servei que existeixi en cada una de les màquines de la *beamline* esmentada.

Amb tots aquests processos aconseguirem crear la base de dades amb les entrades que necessitem.

3.2 Frontend

El *frontend* és la part de l'aplicació que interactua amb els usuaris. És a dir, és la part visible, la que mostra el disseny i els continguts de l'aplicació.

Per tal de realitzar aquesta part s'ha utilitzat l'eina PyQt5.

3.2.1 PyQt5

Qt principalment és una biblioteca de C++ multiplataforma que implementa l'accés API d'alt nivell a molts aspectes dels sistemes mòbils i d'escriptori [19].

PyQt5 és un conjunt complet d'enllaços de *Python* per *QT V5*.

En altres paraules, *PyQt5* és un *binding* de la biblioteca gràfica QT pel llenguatge de programació *Python*, el qual permet crear interfícies gràfiques amb *Python* de manera ràpida i senzilla.

PyQt5 funciona amb un sistema de Widgets. Podriem considerar un Widget com un component de la interfície gràfica. [20]

Entrant en detall en l'aplicació, quan s'accedeix a aquesta, es mostren tres desplegables, *beamlines*, *hosts* i serveis tal i com es mostra a la Figura 5. Per defecte obtenim el valor *All* amb el qual es seleccionarien totes les *Beamlines*, els *Hosts* i els *Serveis* existents.

En la pantalla que mostra la Figura 5 es demana a l'usuari que introdueixi les opcions esmentades anteriorment, un cop decidit, es prem el botó *SHOW* amb el qual es mostrara la següent pantalla.

Per aconseguir aquest procés s'han hagut d'afegir dinàmicament les *beamlines*, els *hosts* i els serveis, ja que si en algun instant es canvia el nom d'algun d'aquests l'aplicació sigui capaç d'executar-se de nou afegint els nous termes.

Per tal de fer-ho dinàmicament he hagut d'accedir al repositori git des del qual s'obté la configuració que s'ha d'aplicar en cada host i on es guarden tots els canvis realitzats, tant la creació de noves *beamlines*, màquines i serveis com la modificació d'aquestes.

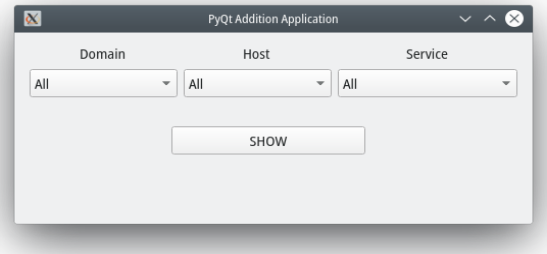

Fig. 5: Primera pantalla per defecte de l'aplicació

Un cop decidits els elements que ens interessen, en la següent pantalla se'ns mostrarà les versions instal·lades dels serveis seleccionats en les màquines respectives. És a dir, si hem seleccionat una *Beamline*, un *Host* i un servei, ens mostrarà per pantalla la versió que té disponible aquest servei en aquesta màquina de la *Beamline* seleccionada.

A la Figura 6 es mostra la selecció que ha fet l'usuari, seguidament de prèmer el botó *SHOW* apareix la pantalla que es mostra a la Figura 7, la qual s'obté la versió disponible del servei *tango* a la màquina *ctblvm01*.

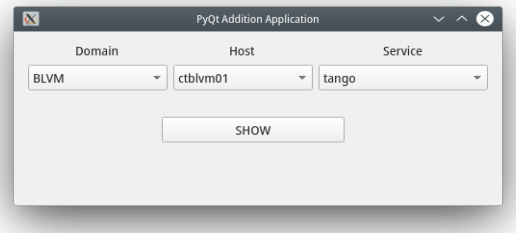

Fig. 6: Primera pantalla de l'aplicació

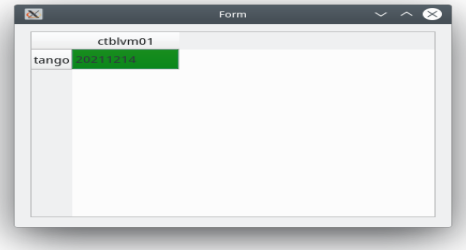

Fig. 7: Segona pantalla de l'aplicació amb entrada **domain** BLVM, **host** ctblvm01 i **servei** tango

Per tal d'aclarir més l'estructura de l'aplicació, es mostra a la Figura 8 el cas més comú, conèixer tots els serveis instal·lats en una *beamline* concreta. En aquest cas s'ha seleccionat la *beamline* anomenada *BLVM*, totes les màquines existents en aquesta i tots els serveis disponibles en el Sincrotró ALBA.

Es pot observar a la Figura 8 que les cel·les de la taula apareixen en dos colors: verd i vermell. Si una cel·la està en color verd significa que aquell servei està actualitzat, és a dir, que la versió disponible i la versió instal·lada coincideixen. Si el color de la cel·la és vermell significa tot el contrari.

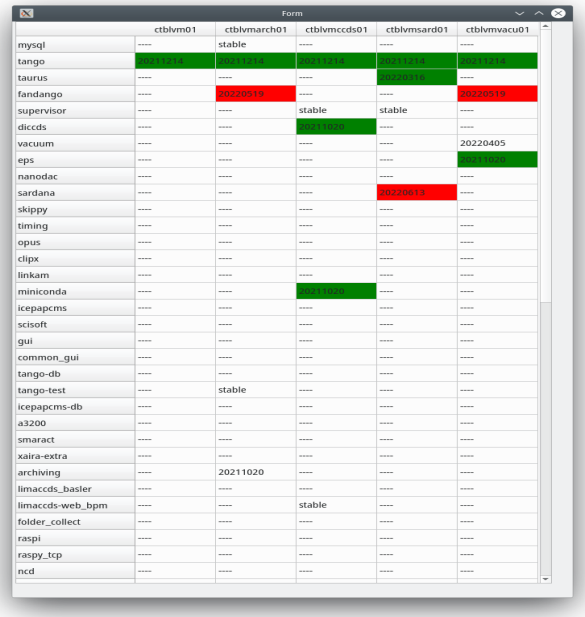

Fig. 8: Segona pantalla de l'aplicació amb entrada **beamline** BLVM amb tots els **hosts** i **serveis**

Per tal d'aconseguir el procés de comprovació de les versions dels serveis, s'ha hagut de connectar amb la base de dades creada anteriorment mitjançant el *Docker* també creat.

L'última pantalla que apareix si a l'usuari li interessa es mostra a la Figura 9. Aquesta s'obté a partir de seleccionar una cel·la de la taula mostrada a la Figura 8 i prèmer la tecla *Enter*.

La informació que ens proporciona aquesta pantalla és l'historial d'instal·lacions que s'han fet d'aquest servei en aquesta màquina amb la seva data respectiva decreixent, és a dir, la darrera actualització feta és la primera línia que apareix.

Aquesta informació ens ajuda a tenir un historial complet de les instal·lacions que s'han produït.

Serà creat un botó el qual es prem si l'usuari volgués actualitzar aquell servei, així doncs quan s'executés de nou l'aplicació s'afegiria a la primera línia una fila amb la nova instal·lació.

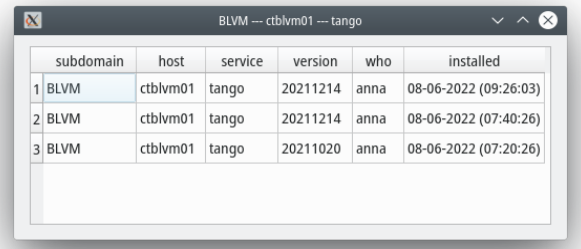

Fig. 9: Tercera pantalla

4 LÍNIES FUTURES DE DESENVOLUPAMENT

Aquest projecte ha nascut de la necessitat que presenta el Sincrotró ALBA de tenir un historial de les actualitzacions fetes dels serveis de les màquines de les *beamlines* corresponents i de proporcionar la informació de l'estat actual de les versions d'aquests.

Per tant, l'objectiu principal del projecte ha consistit en pal·liar les necessitats que existeixen plantejant un projecte enfocat en fer una aplicació d'escriptori que sigui capaç de connectar-se a una base de dades on està inclòs l'historial de les actualitzacions produïdes i comparar així doncs amb les versions dels serveis instal·lats actualment.

S'han trobat que existeixen diferents millores les quals si es solucionen s'obtindria un millor projecte i donarien possibilitat a fer ampliacions de l'aplicació.

- **Funcionalitat de l'actualització** : La principal millora que s'hauria d'aconseguir seria fer funcionar el botó d'actualitzar i així doncs actualitzar el servei seleccionat a la versió disponible.
- **Controlar el format dels fitxers** : Més que una millora, és un punt que s'ha de tenir molt en compte per futures ampliacions de l'aplicació, ja que els fitxers utilitzats per aquesta han de tenir sempre el mateix format. Si no es controla això, podrien sorgir errors.
- **Fer l'aplicació dinàmica** : Un cop es realitza una actualització d'un servei, cal tancar l'aplicació per fer la inserció d'aquesta en la base de dades. Per això, una gran ampliació de l'aplicació seria que no hagués de ser necessari tancar l'aplicació cada cop.
- **Alertes i notificacions** : Incorporar un sistema de notificacions push al apropar-se la data d'una nova actualització.

5 CONCLUSIONS

Analitzant detingudament els objectius del projecte i els resultats obtinguts, podria afirmar que s'han assolit l'objectiu principal, la creació d'una eina gràfica que ajudi a la gestió de tot l'stack de software per l'adquisició de les dades del Sincrotró ALBA.

El sistema ha de ser capaç de gestionar l'automatització dels serveis que són instal·lats a les màquines trobades a cada *beamline*.

També cal comentar que per manca de temps no ha sigut possible aconseguir que l'aplicació realitzi les actualitzacions els serveis, és a dir, que el botó d'actualitzar funcioni. Probablement seria el pas més complex i es necessitaria més temps del que ha suposat aquest projecte.

Com s'ha comentat a l'Apartat 4, la principal millora que s'hauria de fer en l'aplicació hauria de ser aquesta, ja que crec que seria molt útil.

Tot i així, crec que aquest projecte ha suposat un aprenentatge profund i intens per a mi. Des del principi m'he hagut d'adaptar a la gestió del software del Sincrotró ALBA i he hagut d'assolir els coneixements necessaris per tal d'entendre-ho. Així com seguir amb l'estructura dels fitxers creats anteriorment al meu projecte, com poden ser els fitxers on s'emmagatzema la informació de cada un dels serveis, els fitxers *init.sls* tal i com s'explica a l'Apartat 3.1.3.

Finalment, es pot determinar que aquest projecte ha aconseguit un resultat satisfactori, complint amb els objectius que es van proposar inicialment i amb possibilitat de millores futures. Tot i la complexitat que implicava el projecte, ja que s'havia de ser capaç de crear tant el *backend* com el *frontend* de manera autònoma i creant el disseny i el plantejament que es seguiria.

En conclusió, aquest projecte pot suposar un estalvi de temps per part del personal del Sincrotró ALBA, ja que avui dia es fa manualment el que realitza l'aplicació.

AGRAÏMENTS

Principalment m'agradaria agraïr al personal del Sincrotró ALBA per donar-me la possibilitat de realitzar aquest projecte tant interessant per a mi, ja que m'ha proporcionat molts coneixements i em seran molt útils en el món laboral.

Seguidament m'agradaria donar les gràcies al meu tutor de la UAB, l'Eduardo César, per ser l'encarregat de realitzar el seguiment del projecte i per la seva implicació en tot moment i sobretot per indicar-me sempre com millorar cada un dels detalls de l'informe.

REFERÈNCIES

[1] "Què és ALBA", https://www.cells.es/es/que-es-alba/bienvenida

[2] "Salt in 10 Minutes", https://docs.saltproject.io/en/latest/topics/tutorials/walkthrough.html

[3] "Python Client API", https://docs.saltproject.io/en/latest/ref/clients/index.html

[4] "El modelo en cascada: desarrollo secuencial de software", https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/el-modelo-en-cascada/

[5] "Model espiral - Viquipèdia, l'enciclopèdia lliure", https://ca.wikipedia.org/wiki/Model_espiral

[6] "An Intro to Git and GitHub for Beginners (Tutorial)", https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners

[7] "Learn Git Branching", https://learngitbranching.js.org/?locale=es-AR

[8] "¿Qué es Docker?", https://www.redhat.com/es/topics/containers/what-is-docker

[9] "¿Qué es SaltStack? — Solución de código abierto - Tecnología Android", https://tecnologiandroid.com/que-es-saltstack-solucion-de-codigo-abierto/

[10] "¿Qué es Docker? Descripción, funciones y ejemplos de uso", https://www.unir.net/ingenieria/revista/funciones-docker/

[11] "Utilizar Docker: beneficios, estadísticas — Apiumhub", https://apiumhub.com/es/tech-blog-barcelona/beneficios-de-utilizar-docker/

[12] "¿Qué es YAML?", https://www.redhat.com/es/topics/automation/what-is-yaml

[13] "Docker vs Docker Compose, what's the difference?", https://londonappdeveloper.com/docker-vs-docker-compose-whats-the-difference/

[14] "Creación de aplicaciones de varios contenedores con MySQL y Docker Compose — Microsoft Docs", https://docs.microsoft.com/es-es/visualstudio/docker/tutorials/tutorial-multi-container-app-mysql

[15] "Understanding Jinja", https://docs.saltproject.io/en/latest/topics/jinja/index.html

[16] "¿En qué consiste SaltStack?", https://www.ionos.es/digitalguide/servidores/configuracion/que-es-saltstack/

[17] "What is SaltStack? - Definition from WhatIs.com", https://www.techtarget.com/searchitoperations/definition/SaltStack

[18] "Empezando con ZeroMQ — Notas de aprendizajes", https://rchavarria.github.io/notes/proyectos/aprendizaje/2018/04/24/zeromq-101.html

[19] "PyQt5 of Py GUI: una guía detallada sobre la introducción, instalación y uso de PyQt5 - programador clic", https://programmerclick.com/article/8237907030/

[20] "PyQt5 Tutorial 2022, Create Python GUIs with Qt", https://www.pythonguis.com/pyqt5-tutorial/