UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

telecos BCN

DESPLEGAMENT D'EINA D'ORQUESTRACIÓ I D'AUTOMATITZACIÓ PER LA GESTIÓ DE XARXES

Treball final de grau

per

Raquel Abad de las Heras

Co-director: Jordi Casademont Serra

Co-director: Daniel Fernández Pérez

Barcelona, Gener 2023

Abstract

This project is proposed with the aim of obtaining a tool for the automation of processes in network management, especially in cases where there are a large number of network equipment. It provides an easy-to-use tool for the administrator, where, in a centralised way, he can make changes in all his equipment.

The tool implemented in this case is AWX, a tool with an Ansible task engine. During this project AWX is deployed on a Kubernetes cluster and all decisions made are explained. Next, an Ansible project is created in GitLab that will be used as a repository, which AWX will access to download configuration files and *playbooks*.

Finally, playbooks are developed and run on network equipment to test their correct functioning and thus fulfil their purpose for the network administrators of UPCnet.

Resum

Aquest projecte es proposa amb la finalitat d'obtenir una eina per a l'automatització de processos en la gestió de xarxes, sobretot en els casos que es disposa de gran quantitat d'equips de xarxa. Es proporciona una eina fàcil d'utilitzar per l'administrador, on, de manera centralitzada, pot fer canvis en tots els seus equips.

L'eina implementada en aquest cas és AWX, una eina amb un motor de tasques d'Ansible. Durant aquest projecte de desplega AWX en un clúster de Kubernetes i s'expliquen totes les decisions preses. A continuació, es crea un projecte d'Ansible al GitLab que serà utilitzat com a repositori, al qual AWX accedirà per a descarregar fitxers de configuració i *playbooks*.

Finalment, es desenvolupen *playbooks* i s'executen sobre equips per a comprovar el seu correcte funcionament i d'aquesta manera complir el seu objectiu per als administradors de xarxa d'UPCnet.

Resumen

Este proyecto se propone con la finalidad de obtener una herramienta para la automatización de procesos en la gestión de redes, sobre todo en los casos que se dispone de gran cantidad de equipos de red. Se proporciona una herramienta fácil de utilizar por el administrador, donde, de manera centralizada, puede realizar cambios en todos sus equipos.

La herramienta implementada en este caso es AWX, un herramienta con un motor de tareas de Ansible. Durante este proyecto de despliega AWX en un clúster de Kubernetes y se explican todas las decisiones tomadas. A continuación, se crea un proyecto de Ansible en GitLab que será utilizado como repositorio, al que AWX accederá para descargar ficheros de configuración i *playbooks*.

Finalmente, se desarrollan *playbooks* y se ejecutan sobre equipos para comprobar su correcto funcionamiento y de esta manera cumplir su objetivo para los administradores de red de UPCnet.

Agraïments

M'agradaria agrair a totes aquelles persones que m'han ajudat durant el desenvolupament d'aquest projecte. Tant als que m'ha aportat idees, plantejat alternatives o ensenyat coses noves sobre les tecnologies aplicades, com als que m'han guiat durant el procés de redacció i d'estructuració del projecte i m'han suggerit possibles millores.

Vull agrair a l'empresa UPCnet per l'oportunitat que m'han ofert de realitzar el meu treball de final de grau amb ells, per proporcionar-me tots els recursos necessaris per la realització del projecte i per la confiança depositada en mi per dur-lo a terme. Però sobretot, vull agrair als meus companys de Infrastructure Engineering pels coneixements que m'han fet adquirir, el seguiment i l'interès que han mostrat en ajudar-me.

També, vull destacar la figura del meu co-director, Jordi Casademont Serra, pel seguiment realitzat, per guiar-me sobretot en el procés de redacció i estructuració del projecte i per les millores proposades.

Finalment, m'agradaria agrair al meus familiars i amics pel recolzament i els ànims que m'han donat des de l'inici d'aquest projecte.

Historial de revisions i registre d'aprovació

Revisió	Data	Propòsit
0	15/12/2022	Creació del document
1	12/01/2023	Revisió del document
2	16/01/2023	Revisió del document

LLISTA DE DISTRIBUCIÓ DEL DOCUMENT

Nom	e-mail
Raquel Abad de las Heras	
Jordi Casademont Serra	
Daniel Fernández Pérez	

Escrit per:		Revisat i aprovat per:	
Data	15/12/2022	Data	12/01/2023
Nom	Raquel Abad de las Heras	Nom	Jordi Casademont Serra
Posició	Autora del projecte	Posició	Director del projecte

Índex

Abstract 1
Resum 2
Resumen 3
Agraïments 4
Historial de revisions i registre d'aprovació 5
Índex de figures 9
Índex de taules 10
1. Introducció 11
1.1. Motivació del projecte 11
1.2. Objectius principals 12
1.3. Requeriments i especificacions 12
1.4. Pla de treball 13
1.4.1. Diagrama Gantt 14
2. Estat de l'art 15
2.1. Tecnologia basada en contenidors 15
2.2. Kubernetes 16
2.2.1. Pod 17
2.2.2. Namespaces 17
2.2.3. Secrets 17
2.2.4. Polítiques de xarxa 17
2.2.5. *Annotations* i *Labels* 18
2.2.6. Configmap 18
2.3. Helm 18
2.3.1. Templates 18
2.3.2. Chart 18
2.3.3. Values 18
2.3.4. .helmignore 19
2.3.5. Imatge 19
2.4. Git i GitLab 19
2.5. Ansible 19
2.5.1. *Playbooks* 20
2.5.2. Rols 21

2.5.3. Inventari 21
2.5.4. Ansible Vault 21
2.5.5. Ansible Galaxy 22
2.6. AWX 22
2.6.1. Credencials 22
2.6.2. Projectes 22
2.6.3. Inventari i *Hosts* 23
2.6.4. Templates 23
2.6.5. Organitzacions 23
2.7. Altres conceptes teòrics 23
2.7.1. Clúster de Kubernetes 23
2.7.2. Calico 23
2.7.3. Rancher 24
2.7.4. Balancejador de càrrega 24
2.7.5. DNS 24
2.7.6. Bastion Host 24
2.7.7. Monitorització 24
3. Desenvolupament del projecte 25
3.1. Requeriments pel desplegament 25
3.2. Estructura desplegament l'AWX 26
3.3. Instal·lació de l'AWX 26
3.4. Configuració credencials admin 27
3.5. Configuració del fitxer *values.yaml* 27
3.6. Aplicació de les polítiques de xarxa 28
3.7. Assignació *pool* d'adreces *IP* 28
3.8. Integració del servei amb el balancejador 30
3.9. Monitorització 32
3.10. Integració d'AWX amb el projecte de GitLab 33
3.11. Desenvolupament de *playbooks* 34
3.11.1. *Playbook* executat sobre un *switch* 34
3.11.2. *Playbook* executat sobre un servidor 35
3.12. Habilitar accessos 35
3.13. Altres configuracions d'AWX 35

4. Resultats 36
4.1. Desplegament de l'aplicació 36
4.2. Integració amb Git 36
4.3. Execució de *playbooks* 37
4.3.1. *Playbooks* per equips d'infraestructura de xarxes 37
4.3.2. *Playbooks* per màquines virtuals 38
5. Costos 39
6. Conclusions i desenvolupament futur 40
6.1. Conclusions 40
6.2. Desenvolupament futur 40
Acrònims 41
Bibliografia: 42
Annexos: 43

Índex de figures

Figura 1.1: Diagrama Gantt 14
Figura 2.1: Imatge transició a K8s 16
Figura 3.1: Estructura AWX 26
Figura 3.2: Comprovació pods i adreces *IP* assignades (default pool) 27
Figura 3.3: Comprovació pods i adreces *IP* assignades 29
Figura 3.4: Comprovació integració balancejador 31
Figura 3.5: Pantalla login AWX 32
Figura 3.6: Pantalla principal AWX 32
Figura 3.7: Configuració projecte AWX 33
Figura 3.8: Configuració inventari AWX 34
Figura 4.1: Comprovació integració GitLab (*playbooks*) 36
Figura 4.2: Comprovació integració GitLab (*host_vars*) 36
Figura 4.3: Execució *playbook* (I) 37
Figura 4.4: Comprovació *playbook* (I) 37
Figura 4.5: Execució *playbook* (II) 38
Figura 4.6: Comprovació *playbook* (II) 38

Índex de taules

Taula 1: WP1 13
Taula 2: WP2 13
Taula 3: WP3 13
Taula 4: WP4 13
Taula 5: WP5 14
Taula 6: WP6 14
Taula 7: WP7 14
Taula 8: Comparació Kubernetes amb Docker Swarm [3] 15

1. Introducció

En aquest apartat es realitzarà una introducció al projecte justificant quina ha estat la motivació per realitzar-lo, exposant els objectius principals i els requeriments i especificacions, i finalment, proposant un pla de treball.

1.1. Motivació del projecte

Una de les tasques principals a les que ens dediquem al departament de Infrastructure Engineering de l'empresa UPCnet és a l'administració i la gestió de la gran majoria dels equips de xarxa dels campus de la UPC, entre d'altres. Això implica que gestionem aproximadament un 400 equips de xarxa entre *switches* i *routers*, i una gran quantitat de servidors, tant físics com virtuals.

Moltes de les tasques a realitzar en la configuració d'equips de xarxa són repetitives i comporten gran risc d'error humà per part de l'administrador. A més, hi ha casos on s'han de realitzar els mateixos canvis en un conjunt d'equips i resulta poc eficient executar-los de manera individual.

A conseqüència d'aquestes situacions i amb la tendència a l'automatització de processos en tot l'àmbit de les tecnologies, s'han creat aplicacions i eines amb gran potencial que permeten realitzar aquestes gestions de manera eficient, segura i centralitzada. Aquestes aplicacions, permeten evitar errors de configuració i implica realitzar qualsevol canvi de manera molt més eficient.

Una de les eines més populars al mercat per la gestió d'infraestructures de Tecnologies de la Informació (TI) és Ansible.

Ansible permet realitzar canvis a partir de scripts escrits en *Yet Another Markup Language* (YAML), anomenats *playbooks*, sobre un conjunt d'equips de xarxa, anomenat inventari. Per tant, un cop s'ha desenvolupat un *playbook* per realitzar qualsevol tasca sobre la configuració d'un equip, sempre que s'hagi de realitzar el mateix canvi, es podrà reutilitzar el *playbook* i executar la tasca de manera automàtica.

Al departament de Infrastructure Engineering ja fa un temps que tenim desplegat Ansible, però, actualment, no li triem suficient profit. El motiu és que molts cops és molt més ràpid executar el canvi de manera manual que accedir a Ansible, buscar el *playbook*, modificar-lo per a que s'executi sobre el grup de l'inventari que volem... Aleshores a no ser que el canvi impliqui realitzar molts passos i es trobi automatitzat, no l'utilitzem.

D'aquesta situació sorgeix la proposta del projecte, vam trobar que existia una eina que facilitava tot el procés per llençar un *playbook* sobre un inventari d'equips i, a més, ens permetia poder realitzar control d'accessos per delegar alguns procediments a altres departaments. Aquesta eina s'anomena AWX.

AWX ens proporciona una interfície web fàcil d'utilitzar per l'usuari on trobem tots els nostres *playbooks* i l'estructura del nostre inventari. A més, ens ofereix la possibilitat de definir variables directament des de la interfície web, sense modificar el *playbook*. I a part, AWX ens permet ser desplegada a Kubernetes, que és una de les tendències que segueix la companyia, migrar les aplicacions a Kubernetes.

Considerem que el desplegament d'aquesta aplicació beneficiaria al departament en el sentit d'explotar més una eina tant potent com Ansible, tenint tota l'infrastructura de xarxa centralitzada i gestionada per una eina d'orquestració. I ens permetria delegar moltes de les tasques de tècnic a altres companys.

1.2. Objectius principals

Els objectius principals amb el quals es va proposar la realització del projecte són els següents:

- Conèixer l'entorn de Kubernetes, entendre el funcionament, l'estructura i les característiques del *software*, conèixer els tipus d'objecte i recursos necessaris per desplegar aplicacions.
- Comprendre com funciona un clúster de Kubernetes i quins components el formen.
- Entendre com funciona Helm i aprendre utilitzar-lo per desplegar aplicacions a Kubernetes.
- Conèixer el funcionament d'Ansible, els components necessaris i aprendre desenvolupar *playbooks* i rols a partir de l'inventari i comprovar el seu funcionament.
- Aprendre com funciona Git, i utilitzar-lo per realitzar canvis en l'actual projecte d'Ansible a GitLab.
- Desplegar l'aplicació AWX a un clúster de Kubernetes utilitzant Helm.
- Crear algun *playbook* útil per la gestió dels equips de xarxa i comprovar el seu funcionament quan s'executa a través d'AWX sobre l'inventari.

1.3. Requeriments i especificacions

Per realitzar el desplegament de l'aplicació és necessari disposar d'un clúster de Kubernetes en entorn preproducció preparat pel desplegament d'aplicacions que es troben en estat de proves, cal crear un projecte d'Ansible creat a GitLab, es requereix que el balancejador de càrrega tingui accés al clúster. A més, es necessari disposar d'equips de xarxa per crear l'inventari i realitzar proves amb *playbooks*, una màquina virtual per realitzar proves i un usuari genèric: usuari.awx.

En aquest projecte s'utilitzaran les següents eines amb les respectives versions: al clúster caldrà instal·lar Kubectl v1.23.8, Calico v3.24.3 i Helm v3.9.4. Per realitzar proves, es faran servir una màquina virtual amb sistema operatiu Ubuntu 20.04 i un *switch* del fabricant Cisco model WS-C3560X-48P.

1.4. Pla de treball

A continuació, es mostra l'organització del projecte en diferents paquets de treball:

WP1: Preparar el clúster per desplegar aplicacions	
Descripció	Consisteix en instal·lar al clúster les eines necessàries per poder desplegar aplicacions.
Tasques a realitzar	Instal·lar kubectl, Helm, crear un *namespace* per al projecte I comprovar que hi ha recursos suficients de CPU i RAM
Fites	Comprovar que funcionen les ordres bàsiques de les diferents eines instal·lades.

Taula 1: WP1

WP2: Introducció a Kubernetes i a Helm	
Descripció	Consisteix en entendre com funciona Kubernetes i Helm i aplicar les ordres bàsiques per desplegar aplicacions.
Tasques a realitzar	Aprendre ordres de Kubernetes i l'estructura d'un paquet Helm.
Fites	Saber utilitzar les diferents ordres amb la finalitat del realitzar desplegaments.

Taula 2: WP2

WP3: Personalització de l'aplicació	
Descripció	Consisteix en entendre com funciona el paquet Helm a instal·lar i modificar el fitxer values.yaml en funció del nostre entorn
Tasques a realitzar	Entendre els *templates* que constitueixen l'aplicació i modificar el vàlues.yaml en funció dels interessos.
Fites	Fitxer values.yaml amb el que es desplegarà la primera versió de l'aplicació

Taula 3: WP3

WP4: Desplegament d'AWX al clúster	
Descripció	Consisteix en desplegar l'aplicació utilitzant els fitxers personalitzats i els *templates*.
Tasques a realitzar	Desplegar l'aplicació utilitzant Helm i comprovar que els pods estan en estat running. Aplicar les polítiques de xarxa i el *Ingress*.

Fites	Comprovar amb l'ordre "kubectl get pods" que les pods que composen l'aplicació estan running

Taula 4: WP4

WP5: Registrar l'aplicació al balancejador de càrrega	
Descripció	Consisteix en registrar el servei a un aglutinador del balancejador de càrrega i al Domain Name System (DNS)
Tasques a realitzar	Afegir el servei a DNS i a un aglutinador amb el seu servidor virtual.
Fites	Poder accedir via web a través del nom de servei.

Taula 5: WP5

WP6: Obrir accés als equips per desplegar *playbooks*	
Descripció	Consisteix en obrir accés des de les *IP*'s de l'aplicació cap al equips sobre els que s'executen els *playbooks* pel port 22, protocol Secure Shell (SSH).
Tasques a realitzar	Permetre l'accés a les ACL pels equips de xarxa i obrir accés al tallafocs pels servidors.
Fites	Permetre l'accés d'AWX als diversos equips de xarxa

Taula 6: WP6

WP7: Desenvolupament de *playbooks*	
Descripció	Consisteix en desenvolupar *playbooks* de prova per comprovar que AWX funciona correctament
Tasques a realitzar	Realitzar scripts en YAML (*playbooks* i rols)
Fites	Executar els *playbooks* contra el equips

Taula 7: WP7

1.4.1. Diagrama Gantt

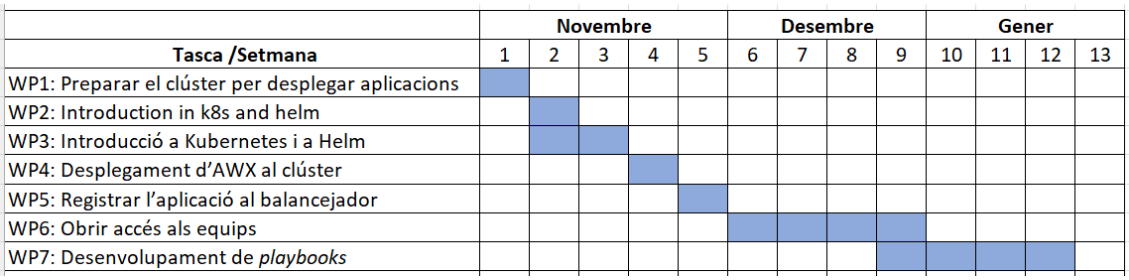

Figura 1.1: Diagrama Gantt

2. Estat de l'art

En aquest apartat es presentaran totes les tecnologies i eines principals utilitzades durant el desenvolupament del projecte. Es realitza una introducció teòrica on s'expliquen les característiques més rellevants de cada tecnologia enfocant-se en la seva utilització durant el projecte i basant-se en la informació recollida durant el seu desenvolupament.

La realització del projecte es basa en la implementació de principalment 5 tecnologies: Kubernetes (k8s), Helm, Git, Ansible i AWX.

2.1. Tecnologia basada en contenidors

La tecnologia basada en contenidors, és un tipus de sistema distribuït que ens permet segmentar qualsevol aplicació en microserveis que s'implementen de manera independent i es comuniquen a través d'una API.

Utilitzar la tecnologia basada en contenidors a les empreses per desplegar aplicacions és una tendència i així ho corroboren els següents articles:

L'article "The state of cloud-native development: Kubernetes is on the rise" [1] publicat per VentureBeat a finals del 2020, on informa que el 31% dels desenvolupadors utilitzen Kubernetes, xifres que representen un augment del 67% respecte a l'any anterior.

O l'article "A DevOps reset for a multi-cloud world" [2] publicat per Cloud Native Computing Foundation a finals del 2022, que també presenta Kubernetes com a eina utilitzada en un futur per gestionar el canvi disruptiu que proposa.

Els principals avantatges de tenir les aplicacions estructurades en una arquitectura de microserveis és la facilitat per escalar el servei, l'agilitat per realitzar canvis o implementar noves funcions, l'alta disponibilitat dels recursos i la reducció de costos.

Dues de les plataformes per administrar aplicacions desplegades a partir de contenidors són Docker Swarm i Kubernetes. A la següent taula podem observar algunes de els diferències:

	KUBERNETES	**DOCKER SWARM**
DEFINICIÓ D'APLICACIONS	Combinació de pods, implementacions i serveis	Serveis d'un clúster Swarm
ESCALABILITAT	S'escala quan s'administra a una implementació	S'escala mitjançant plantilles Docker Compose YAML
ALTA DISPONIBILITAT	Els pods es distribueixen als nodes per tenir HA	Els administradors de Swarm són responsables del clúster i d'administrar els recursos
BALANCEIG DE CÀRREGA	Existeix un balancejador de càrrega dins del clúster	Té un component DNS que es pot ser servir per distribuir les peticions a un servei
XARXES	Tots els pods es poden comunicar entre si.	Es crea una xarxa de superposició per a serveis i una xarxa que fa de pont Docker només per contenidors.

Taula 8: Comparació Kubernetes amb Docker Swarm [3]

Com es pot deduir de la taula anterior, no hi ha gran diferencia entre els dos competidors com a eines de gestió d'aplicacions desplegades en contenidors, però actualment Kubernetes és molt més popular.

2.2. Kubernetes

Kubernetes [4] és una eina d'orquestració de contenidors que s'encarrega de la gestió del cicle de vida dels contenidors d'una aplicació. Realitza processos de gestió del clúster on es troba instal·lat, com crear pods o eliminar-los segons la seva funció, serveis de xarxa i balanceig de càrrega, que consisteix en repartir la càrrega entre els diferents nodes del clúster, ja que quan es crea el clúster se li assignen els recursos destinats al desplegament de les seves aplicacions i Kubernetes gestiona i assigna els recursos en funció dels requeriments de les aplicacions en cada moment.

A més, permet veure l'estat en el que es troben els pods que fa servir cada aplicació desplegada, crea log dels diversos canvis que realitzen els pods i permet assegurar l'alta disponibilitat dels diferents microserveis de l'aplicació a partir del número de rèpliques. D'altra banda, també es poden crear volums de persistència que permeten recuperar aplicacions i els seu estat quan els nodes en els que s'estava executant cauen.

Kubectl és l'eina principal per gestionar el clúster de Kubernetes.

A la següent imatge podem veure respresentada l'estructura de les aplicacions a Kubernetes:

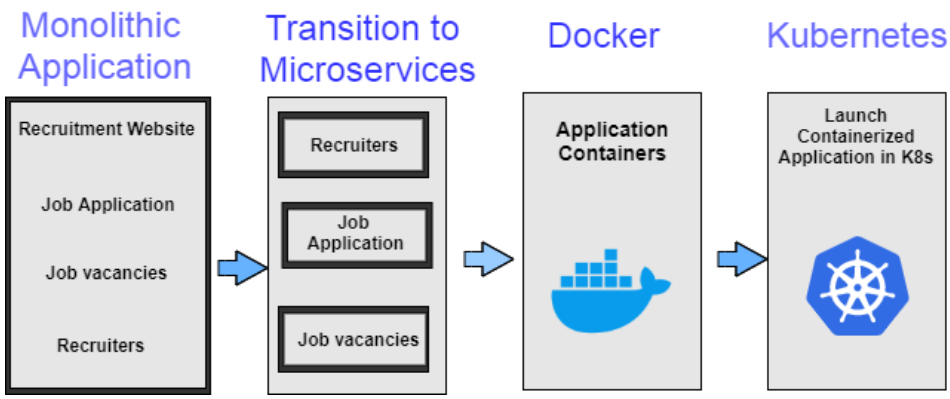

Figura 2.1: Imatge transició a K8s

En definitiva, algunes de les avantatges principals d'utilitzar Kubernetes són:

- Reducció del temps de comercialització: Desplegar aplicacions a Kubernetes permet grans avantatges en termes de comoditat i velocitat a l'hora de implementar canvis, ja que permet desplegar fitxers en temps real i crear entorns de prova.
- Capacitat i portabilitat de multicloud: Kubernetes garanteix que les aplicacions funcionin independentment del entorn on siguin desplegades, això facilitat poder moure-les a altres plataformes sense que la seva funcionalitat es vegi afectada.
- Estabilitat i disponibilitat: Kubernetes garanteix un major grau d'automatització, el que proporciona més facilitat per resoldre incidències i poder obtenir aplicacions més sòlides. A part, Kubernetes disposa de processos d'autoreparació que s'executen de manera automàtica en funció de la salut de les aplicacions.

- Optimització de costos i reducció d'esforç: Kubernetes permet empaquetar aplicacions basades en contenidors de manera òptima i, per tant, garanteix un consum de recursos més eficient. Aquest fet, redueix costos en infraestructura significativament.

A continuació, es presenten alguns del conceptes clau de Kubernetes per poder realitzar el desenvolupant del projecte:

2.2.1. Pod

Un pod és l'objecte més simple de Kubernetes, consisteix en un grup de un o més contenidors que comparteixen emmagatzematge i xarxa. Els pods representen processos en execució en els nodes d'un clúster, per això es important la gestió del clúster i tenir la possibilitat d'eliminar-los quan finalitzen la seva funció, per tant, és un recurs no permanent.

Per defecte, Kubernetes assigna una adreça de protocol d'internet (*IP*) a cada pod quan el crea. Aquesta adreça *IP* es tria del *pool* d'*IP*s assignat a la configuració del clúster. Cada cop que un pod l'elimina o es reinicia, i l'aplicació ha de crear un nou, l'adreça *IP* assignada canvia.

2.2.2. Namespaces

Els *namespaces* fan referencia a clústers virtuals, s'utilitzen en casos de entorns distribuïts que gestionen diversos projectes i on poden accedir molts usuaris. Kubernetes ens permet assignar a cada *namespace* una part limitada dels recursos del clúster. A cada *namespace* podem trobar els diferents objectes i fitxers de configuració aplicats al desplegament de la respectiva aplicació.

2.2.3. Secrets

Els objectes de tipus *Secret* a Kubernetes s'utilitzen per emmagatzemar i gestionar qualsevol tipus d'informació confidencial, com poden ser contrasenyes d'accés, claus ssh privades, tokens... Els pods poden accedir als *secrets* fent referència a ells sempre que es trobin creats al mateix *namespace*.

2.2.4. Polítiques de xarxa

Les polítiques de xarxa de Kubernetes és un recurs de Kubernetes que ens permet descriure quins pods es poden comunicar entre ells, quins *namespaces* o quins rangs d'adreces *IP* poden accedir als pods.

Per defecte a Kubernetes, tots els accessos estan oberts, per tant, si o fem servir polítiques de xarxa, qualsevol pod podria accedir a un altre i podria comportar un important problema de seguretat.

En el moment, en el que s'aplica una política de xarxa, tot el tràfic queda denegat a no ser que sigui explícitament permès a la política de xarxa actual o a una amb major prioritat.

En el fitxer on es defineix la política de xarxa, es pot bloquejar el trànsit tant d'entrada (*Ingress*) com de sortida (*Egress*). A més, es pot realitzar una política d'accessos tant a nivell d'adreça *IP* com a nivell de pod, de *namespace* o de *service*, i també es poden limitar els ports pels quals s'obre l'accés.

El fitxer ingress és el recurs on es configura tota la lògica de com es vol exposar el servei. Fent servir *annotations* i *specifications* podem permetre que altres aplicacions accedeixin als recursos i aplicacions desplegats a Kubernetes.

2.2.5. *Annotations* i *Labels*

Les *annotations* de Kubernetes és un recurs que ens permet introduir dades que seran utilitzades per llibreries o eines externes. D'aquesta manera, el beneficiari de les dades descrites com *annotations* les pot identificar fàcilment pel seu ús.

En canvi, els *labels* de Kubernetes s'utilitzen per identificar objectes, tenen una funció informativa de cara als usuaris, però no impliquen cap canvi en el sistema ni en la configuració.

2.2.6. Configmap

Els fitxer ConfigMap de Kubernetes s'utilitzen pels pods i permeten definir variables d'entorn o volums (directoris que contenen dades accessibles pels pods), entre d'altres. Contenen la lògica de configuració dels pods.

2.3. Helm

Helm [5] és una eina de gestió d'aplicacions a Kubernetes, la seva funció principal és definir, instal·lar i actualitzar aplicacions. Les aplicacions que es troben descrites com a paquet Helm (*Charts* de Helm) tenen una estructura molt definida que faciliten la instal·lació i la gestió de l'aplicació. A més, són aplicacions més fàcils de versionar, actualitzar i estan pensades per ser compartides i allotjades al núvol.

En definitiva, Helm ens permet administrar aplicacions de tercers i desplegar-les de manera personalitzada al nostre clúster de Kubernetes. Per tant, per poder instal·lar una aplicació amb Helm, es necessari poder descarregar o crear l'aplicació com a paquet Helm o *Chart*.

El *Chart* de Helm es defineix com un recurs que permet crear, compartir, publicar y versionar aplicacions fàcilment utilitzant l'eina Helm com a gestor de Kubernetes. Consisteix en un paquet de Helm amb la definició de tots els recursos i fitxers de configuració necessaris per l'execució de l'aplicació o servei.

L'estructura d'un paquet Helm ha de disposar com a mínim dels següents components:

2.3.1. Templates

A la carpeta de *Templates*, es troben tots els fitxers que seran instal·lats a Kubernetes. Dins d'aquesta carpeta, trobarem una carpeta per cada servei que ofereix l'aplicació amb els fitxers YAML de la configuración.

2.3.2. Chart

El fitxer *Chart* conté la versió i la configuració del *Chart* de Helm utilitzades pel desplegament de l'aplicació. Cada cop que s'aplica un canvi sobre la configuració de l'aplicació s'ha d'actualitzar la versió d'aquest fitxer.

2.3.3. Values

El fitxer *values.yaml*, conté la informació necessària per personalitzar l'aplicació a partir dels fitxer de configuració que podem trobar a la carpeta *Templates*. Aquest fitxer conté uns valors per

defecte que poden ser substituïts durant la instal·lació de l'aplicació. Permet definir els recursos utilitzats durant el desplegament, també permet afegir les anotacions necessàries per la compatibilitat amb altres aplicacions.

2.3.4. .helmignore

El fitxer .helmignore conté tots els fitxers que no es volen incloure al *Chart* de Helm, és a dir, tots els fitxers que siguin especificats en aquest fitxer no s'inclouran al paquet de Helm. Això permet que arxius innecessaris o confidencials no siguin inclosos.

2.3.5. Imatge

La imatge de l'aplicació no s'inclou al paquet de Helm, però és totalment necessària per poder desplegar una aplicació utilitzant un *Chart* de Helm.

La imatge acostuma a ser publicada pel fabricant de l'aplicació i conté tots els recursos mínims necessaris per poder definir l'aplicació.

En alguns casos, aquesta imatge no es llegeix de la publicada pel fabricant, sinó que es s'extreu d'un repositori privat propietat del client de l'aplicació.

2.4. Git i GitLab

GitLab [6] és un repositori de codi, que ofereix una interfície d'usuari, on es permet gestionar diversos projectes basat en el sistema de versions Git.

Git és *software* de gestió de versions pensat per augmentar l'eficiència en el desenvolupament de projectes, facilitant la col·laboració entre desenvolupadors i incrementant la compatibilitat entre versions. El seu propòsit principal es crear un registre amb tots els canvis realitzats en els diferents fitxers del codi.

Git permet treballar amb branques, les quals es creen com a còpies d'una principal, la branca *master*. Cada cop que es realitza un canvi al codi s'afegeix a Git *("git add")* i quan els canvis es volen pujar al repositori es crea un "*commit"*, on es pot afegir una descripció del canvi, per a que quedi registrat i s'executa un *"push"* cap al repositori.

Finalment, un cop realitzats i pujats tots els canvis a la nova branca, es crea un *merge request* contra la branca *master*. Durant el *merge request* es compara la nova branca amb la branca *master* i s'analitza la compatibilitat dels canvis realitzats. En cas de conflicte entre els canvis realitzats i la branca *master*, des de GitLab es poden resoldre els conflictes i finalment, pujar els canvis a la branca *master*.

D'altra banda, cada *commit* realitzat crea un registre de logs, fet que simplifica poder obtenir versions anteriors dels projectes i d'aquesta manera resoldre possibles errors en el codi.

2.5. Ansible

Ansible [7] és una eina *software* creada amb la finalitat d'automatitzar processos de configuració de servidors o equips de xarxa, entre d'altres. Es cataloga com una eina d'orquestració utilitzada principalment per administradors de sistemes i DevOps perquè facilita processos d'implementació

de canvis en configuracions d'equips, executant-los de manera centralitzada i reduint la possibilitat d'error.

Ansible resulta una eina interessant de cara a la gestió de xarxes de grans dimensions, on és poc eficient accedir als equips de manera individual per realitzar qualsevol canvi. Ansible permet crear inventaris on s'afegeixen tots els equips o servidors gestionats i es classifiquen per grups en funció del interessos dels administradors del sistema, per després poder llençar *playbooks* sobre tot el conjunt.

Per definir les diverses tasques a realitzar sobre un equip, s'utilitzen els *playbooks* d'Ansible. Ansible es connecta als diferents equips via el protocol SSH (Secure Shell), executa tots els canvis descrits al codi del *playbook* i , finalment, tanca la connexió. Per tant, tot equip administrat des d'Ansible ha de permetre accedir a través del protocol SSH.

Tant els playboooks com els rols d'Ansible són fitxers escrits en YAML. Però també es poden afegir scripts en Python per automatitzar tasques que no es poden descriure amb codi YAML.

En definitiva, algunes de les principals avantatges d'utilitzar Ansible per la gestió d'infraestructures de xarxes de grans dimensions són:

- Ansible proporcional un entorn molt intuïtiu, utilitza fitxers YAML per crear *playbooks* i permet segmentar-los en rols reutilitzables. A més, de proporcionar un inventari fàcil de gestionar.
- Ansible permet gestionar molts processos de manera centralitzada i automatitzada.
- Ansible facilita l'orquestració de tasques difícils d'executar, com pot ser implementar un nou servei a un grup de màquines.
- Ansible realitza els canvis de manera segura a través del protocol SSH que minimitza els riscos de seguretat.

A continuació, es presenten alguns del conceptes clau d'Ansible per poder realitzar el desenvolupant del projecte:

2.5.1. *Playbooks*

El *playbooks* són scripts que s'acostumen a utilitzar per l'automatització de processos de configuració de tecnologies de la informació, com per exemple, la gestió d'equips de xarxa. Als *playbooks*, es descriu cada tasca a realitzar per assolir un objectiu de configuració sobre una varible *host* definida. Aquesta variable *host* pot ser un equip, un grup d'equips o tots els equips registrats a l'inventari d'Ansible.

Els *playbooks* estan formats per *Plays* que són els diferents passos a realitzar durant l'execució. És important respectar un ordre en la descripció dels *Plays*, ja que s'executen de manera seqüencial, tot i que, també és possible realitzar diverses execucions en paral·lel.

D'altra banda, una bona pràctica consisteix en segmentar els *playbooks* en rols que realitzin tasques molt més senzilles i concretes. D'aquesta manera s'evita repetir codi, es poden reutilitzar rols i facilita la comprensió del funcionament del *playbook*.

2.5.2. Rols

Els rols a Ansible són scripts que s'encarreguen de realitzar tasques molt concretes de configuració sobre els *hosts*. És la principal manera de segmentar un *playbook* en diversos fitxers amb finalitats molt més concretes i facilita la possibilitat de reutilitzar-los en altres *playbooks*.

Els rols es creen amb l'objectiu de realitzar un canvi o obtenir un resultat i, per tant, ha d'incloure totes les accions per assolir-lo o pot crear dependències amb altres rols. Igual que als *playbooks*, els rols s'executen de manera seqüencial, per tant, influeix l'ordre en el que es descriuen les accions.

Però la diferència principal, entre els *playbooks* i els rols, consisteix en que els rols, encara que disposen de la informació necessària per ser executats de manera independent, no es possible fer-ho, sempre han de ser executats per un *playbook*. La causa de no poder executar directament un rol, és que no permet definir el *host* sobre el que s'aplica el el rol, per tant, provoca que el *playbook* sigui necessari.

2.5.3. Inventari

L'inventari d'Ansible és el conjunt d'equips creats com a *hosts* a Ansible i sobre els quals pot realitzar canvis a través de *playbooks*.

L'inventari d'Ansible es defineix en un fitxer, anomenat *hosts*, on apareixen tots els equips donats d'alta. Aquests equips poden estar separats en grups per facilitar la seva administració, però cada equip pot pertànyer a més d'un grup i els *playbooks* es podran executar tant sobre equips concrets com sobre grups o inclús sobre tots els equips registrats

D'altra banda, a l'inventari es troba una carpeta anomenada group_vars on al fitxer "*all*" es poden definir les variables que comparteixen tots els equips de l'inventari, com per exemple l'adreça *IP* dels servidors DNS. Això permet no tenir redundat el valor d'una variable compartida en tots els equips. Per seguretat, aquest fitxer acostuma a estar xifrat per una clau Vault.

A més, aquest format de fitxer també es pot aplicar sobre un grup de *hosts*, creant un fitxer amb el nom del grup i afegint les variables desitjades. D'aquesta manera les variables afegides en el fitxer només aplicaran sobre els equips per al fitxer *hosts* estiguin inclosos al grup respectiu.

Per últim, hi ha la carpeta *host_vars*, al seu interior es pot trobar un fitxer per cada equip donat d'alta al fitxer *hosts*. Aquest fitxer conté les variables pròpies de l'equip, per exemple els sistema operatiu de l'equip, i també acostuma a estar xifrat en una clau Vault.

2.5.4. Ansible Vault

Ansible Vault és una eina d'Ansible que permet conservar dades confidencials, com contrasenyes o claus privades. Amb Vault es poden xifrar les variables o fitxers de manera que no es pugui visualitzar el seu contingut. En cas de voler conèixer el valor d'una variable xifrada o el contingut d'un fitxer xifrat cal conèixer la clau Vault amb la que es va xifrar aquest contingut.

Acostumen a trobar-se variables o fitxers xifrats amb Vault als fitxers de variables d'inventari, tant als *group_vars* com als *host_vars*.

2.5.5. Ansible Galaxy

Ansible Galaxy és un repositori de codi on s'emmagatzemen bàsicament, rols i col·leccions, que poden ser utilitzades als propis *playbooks*. Per poder realitzar alguns *playbook* és necessària una col·lecció, per exemple en el cas del equips de xarxa, en molts casos per poder executar comandes sobre els equips cal incloure la col·lecció del fabricant que es troba a l'Ansible Galaxy.

2.6. AWX

AWX [8] és un projecte *open-source* patrocinat per RedHat que proporciona una aplicació web creada per facilitar a l'usuari la gestió dels seus inventaris i *playbooks* d'Ansible. AWX proporciona una interfície gràfica on centralitzar i orquestrar tota la infraestructura de xarxa gestionada per l'usuari. A més, disposa d'una API REST i funciona a partir d'un motor de tasques basat en Ansible que permet programar les tasques a realitzar a partir de la interfície web.

En definitiva, les principal avantatges d'utilitzar AWX per executar Ansible són:

- AWX proporciona una plataforma centralitzada des d'on gestionar els *playbooks* i els inventaris.
- AWX permet visualitzar l'execució en el grau de detall que sigui convenient per l'usuari, fet que facilita la resolució d'errors.
- AWX permet programar tasques a realitzar de manera automàtica a través de la interfície web fàcilment.
- AWX permet realitzar control d'accessos basat en rols que proporcionen diversos privilegis en funció de l'usuari d'accés. Això resulta convenient de cara a delegar tasques a realitzar a altres equips, sense que tinguis privilegis d'administrador.

Per poder llençar *playbooks* amb AWX necessitem principalment conèixer els següents recursos:

2.6.1. Credencials

L'apartat Credencials d'AWX és on ens permet configurar les credencials d'accés als equips finals contra els que s'executa el *playbook*.

En el cas d'equips de xarxa com *switches* o *routers* haurem de fer servir les seves credencials d'accés com a administrador, aquestes variables poden estar encriptades al fitxer *host_vars*, aleshores utilitzarem com a credencial la contrasenya Vault que es fa servir per encriptar.

En canvi, en el cas d'executar els *playbooks* contra servidors Linux, la credencial per accedir haurà de contenir la clau SSH i l'usuari d'accés a la màquina. Finalment, en el cas de Windows, afegirem l'usuari i la contrasenya d'accés.

2.6.2. Projectes

L'apartat de Projectes d'AWX ens permet indicar on es troba ubicat el nostre projecte d'Ansible, és a dir, on es troben els fitxers de configuració del nostre inventari, on estan descrits els *playbooks* i rols… Una de les opcions més comuns per definir projectes d'Ansible es utilitzar Git.

2.6.3. Inventari i *Hosts*

A l'apartat Inventari de AWX és on es defineixen els diferents inventaris d'equips sobre els que es poden executar els *playbooks*. En aquest apartat podem trobar una finestra on es troben tots els *hosts* creats ja siguin servidors o equips de xarxa. També, podem trobar una altra finestra on estan descrits tots els grups de *hosts* que tenim definits al nostre inventari.

2.6.4. Templates

En aquest apartat es on es tria el *playbook* a executar, el projecte d'Ansible que volem utilitzar, l'inventari sobre el qual s'executarà el *playbook* i les credencials necessàries per accedir al *hosts* destí. A més, podem modificar altres variables de l'execució, com el temps d'espera, el número d'intents d'execució o el variables definides al *playbook* que volem modificar.

2.6.5. Organitzacions

Les organitzacions d'AWX són objectes creats amb la finalitat de facilitar la gestió de l'aplicació de manera organitzada. Es tracta d'una manera lògica d'agrupar *playbooks*, inventaris, equips i plantilles.

2.7. Altres conceptes teòrics

En aquest apartat s'introdueixen altres conceptes teòrics també necessaris per la lògica del desenvolupament del projecte.

2.7.1. Clúster de Kubernetes

Els Clústers de Kubernetes són un conjunt de màquines virtuals o físiques, anomenades nodes, que treballen com si fossin una sola màquina. Els clústers són necessaris per poder aprofitar tots els avantatges que proporciona Kubernetes, ja que ens permet desplegar aplicacions sense haver de vincular-les a màquines concretes. Aquest fet proporciona gran escalabilitat a les aplicacions i millor eficiència en l'assignació de recursos.

Els clústers estan formats per dos tipus de nodes: els nodes *masters* I els nodes *workers*.

Els nodes *master* són els encarregats de l'administració del clúster, s'ocupen de l'estat del clúster o de decidir quan s'executa una aplicació i sobre quin node *worker* ho realitza. Per realitzar aquest tipus de decisió, es necessari que el número de nodes *master* d'un clúster sigui imparell i existeixi com a mínim un.

Els nodes *workers* són els encarregats d'executar les tasques i aplicacions que els nodes *master* els indica. Ha d'existir com a mínim un node *workers* per poder tenir un clúster de Kubernetes.

2.7.2. Calico

Per poder desplegar diverses aplicacions en un clúster i que aquestes no tinguin visibilitat entre elles s'apliquen les polítiques de xarxa.

Per defecte, a Kubernetes cada cop que es crea un pod se li assigna una adreça *IP* que fa que aquest pod sigui accessible com un a màquina virtual, aquesta adreça *IP* s'assigna a partir d'un *pool* d'adreces *IP* que té associat al clúster per defecte.

Calico [9] és l'eina que permet crear un model *Container Network Interface* a Kubernetes i aïllar les aplicacions. És a dir, Calico s'encarrega de configurar la xarxa de cada contenidor quan es creat i garantitza que es compleixen els seus requisits de xarxa i seguretat.

Les polítiques de seguretat de Calico es poden aplicar tant sobre pods, sobre adreces *IP* o sobre *namespaces* i permeten diverses accions com: *allow, deny, log*...

2.7.3. Rancher

Rancher [10] és una eina *software* que permet l'administració de clústers de Kubernetes. Facilita tant la gestió dels clústers existents com la creació de nous. Rancher proporciona una interfície web on podem veure els diferents clústers, el *namespaces* i els recursos assignats a cada clúster.

2.7.4. Balancejador de càrrega

El balancejador de càrrega ens permet distribuir el trànsit entre els clients i els servidors. S'utilitza per definir els diversos serveis que s'ofereixen des d'un aglutinador (servidor virtual) a partir de polítiques. A més, es pot definir quines adreces *IP* tenen accés a cada servei i quines adreces *IP* o *pools* de *IP*s responen al servei i per quins ports. També permet crear servidors virtuals dedicats a una única aplicació, sense necessitat de ser inclosos a un aglutinador.

Inclús ens permet crear redireccions URL i afegir el monitor i el certificat SSL de l'aplicació. En alguns casos, interessa que un mateix servei segons els directori en el que es trobi el client responguin *pools* de *IP*s diferents, aquest procés es pot definir al balancejador de càrrega.

2.7.5. DNS

Per poder oferir un servei a partir del balancejador de càrrega és necessari que aquest estigui donat d'alta a DNS. En el cas de tenir un servidor virtual dedicat, es crea un registre tipus A (*host*) a DNS on s'assigna el nom del servei a la seva adreça *IP*. En canvi, quan el servei es troba inclòs a un aglutinador, cal registrar el servei com un Canonical Name (CNAME) del servidor virtual que fa la funció d'aglutinador.

2.7.6. Bastion Host

Un *bastion host* és un equip que es troba situat entre la xarxa que pretén protegir i l'exterior. La seva finalitat principal és oferir protecció i seguretat a la xarxa i als equips que hi ha connectats a ella. El *bastion host* filtra tot el transit d'entrada i de sortida a la xarxa, actuant de "*proxy*" permetent i denegant connexions. D'aquest manera redueix la possibilitat que els equips connectats a la xarxa protegida puguin patir amenaces de l'exterior.

2.7.7. Monitorització

La monitorització és un servei que ens permet conèixer l'estat de l'aplicació en tot moment. Podem monitoritzar les aplicacions a partir d'alarmes, de gràfiques…

En aquest cas, per la monitorització d'una aplicació desplegada a Kubernetes, es monitoritza l'adreça *IP* a partir de la qual s'ofereix el servei i quan aquesta adreça deixa de respondre al servei de monitorització es genera una alarma que informa de que el servei ha caigut i, per tant, segurament l'aplicació no estigui funcionant.

3. Desenvolupament del projecte

Aquest apartat es dedica a l'explicació del desenvolupament del projecte, justificant els pasos seguits per arribar als resultats obtinguts.

3.1. Requeriments pel desplegament

Per poder desplegar aplicacions a Kubernetes es necessari de disposar d'un clúster. En el cas d'aquest projecte utilitzarem un clúster preparat per desplegar aplicacions en entorn de preproducció, ja que ens permet realitzar proves i desplegament sense provocar afectació en cap dels serveis que tenim en producció pels clients.

En el nostre cas es tracta d'un clúster només destinat per la preproducció del departament de Infrastructure Engineering (IE) de l'empresa UPCnet. El clúster el formen tres nodes *master* i cinc nodes *workers*.

Quan es va crear el clúster tenia instal·lat kubectl i calico, eina que més endavant farem servir per realitzar configuracions que no es poden fer amb kubectl. També va ser necessari instal·lar l'eina Helm al clúster per poder desplegar AWX a partir del paquet Helm.

Com a bones pràctiques no s'aconsella treballar directament sobre els nodes *master*, ni realitzar canvis. Per tant, com a normativa del departament per realitzar desplegaments a Kubernetes, hem de configurar-nos en local els clients que necessitem per referir-nos al clúster.

Es recomanable instal·lar els clients en la mateixa versió que es troben al clúster o semblant. En el meu cas, vaig instal·lar els següents paquets:

- Kubectl v1.23.8
- Calico v3.24.3
- Helm v3.9.4

A més, per comoditat de cara a no haver d'estar canviant d'entorn cada cop que canviem de clúster, vaig configurar uns alies que fan referencia als diferents clients utilitzats en cada clúster amb les versions compatibles pertinents.

- **Kubelee:** és el alies que farem servir per referir-nos a l'eina Kubectl al clúster de preproducció de IE.
- **Callee:** és el alies que farem servir per referir-nos a l'eina Calicoctl.
- **Hlee:** és el alies que farem servir per referir-nos a l'eina Helm.

Per tant, a partir d'aquest punt totes les ordres executades es realitzaran en local i utilitzant els àlies indicats.

3.2. Estructura desplegament l'AWX

A continuació es mostra un diagrama del funcionament de l'estructura de l'aplicació que es desplegarà, on es mostren també les diferents tecnologies implicades:

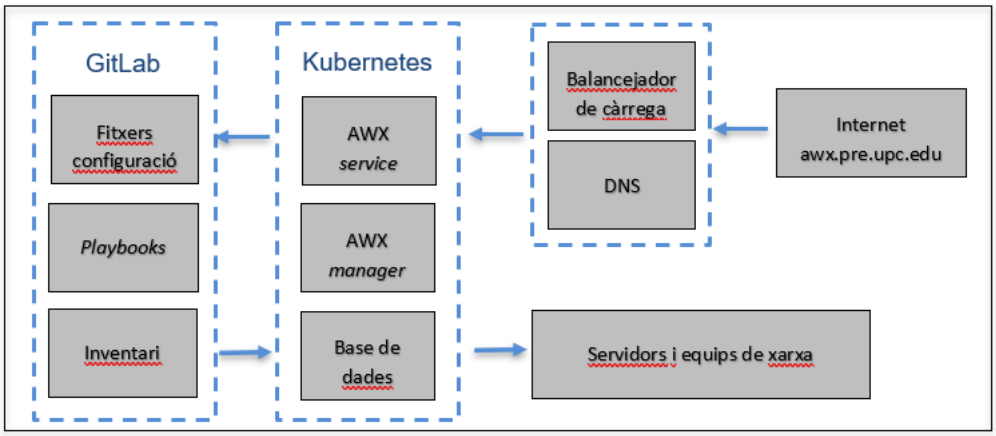

Figura 3.1: Estructura AWX

3.3. Instal·lació de l'AWX

Per desplegar l'aplicació d'AWX a Kubernetes farem servir el paquet helm anomenat **awx-operator [11]**. Aquest paquet es troba allotjat a un repositori de Github i està format per tots el fitxer necessaris en un paquet Helm esmentats a l'apartat 2.3.

Ens baixarem el paquet del repositori en un directori local per poder editar els fitxers que considerem necessaris i després aplicar-los al nostre clúster per instal·lar l'aplicació.

Primerament, crearem un *namespace* al clúster destinat al desenvolupament d'aquest projecte. El *namespace* s'anomenarà **awx-pre**. Per crear-lo farem servir la següent ordre:

```
kubelee create namespace awx-pre
```

I amb la següent ordre podem llistar tots els *namespaces* creats al clúster:

```
kubelee get namespaces
```

A continuació, afegirem un nou repositori a helm anomenat **awx-operator,** que apunta al path on es troba allotjat el projecte, i actualitzarem el repositori del "awx-operator" *chart*:

```
hlee repo add awx-operator
hlee repo update
```

Amb la següent ordre podem llistar tots els *charts* que es troben en un repositori (awx-operator, en aquest cas):

```
hlee search repo awx-operator
```

Finalment, instal·larem tots els fitxers del *Chart* de Helm al *namespace* que hem creat per dedicar al projecte, aplicant el fitxer *values-pre.yaml*, que és el fitxer que ens permet personalitzar el desplegament. Per instal·lar el *Chart* farem servir la següent ordre:

```
hlee install awx awx-operator/awx-operator -n awx-pre -f values-pre.yaml
```

Si tot el desplegament es pot realitzar correctamente, obtindrem un resultat com el següent:

```
Release "awx" has been upgraded. Happy Helming!
NAME: awx
```

```
LAST DEPLOYED: Mon Jan  9 14:38:51 2023
NAMESPACE: awx-pre
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
AWX Operator installed with Helm Chart version 1.0.0
```

A partir d'aquest punt, podem veure que al *namespace* awx-pre s'han creat tres pods: el *service*, el *manager* i el pod de postgres que conté la base de dades. Amb la següent ordre poderm veure els pods creats al clúster i l'adreça *IP* assignada a cada pod i el node *worker* en el que s'està executant (NODE):

```
kubelee get pods -n awx-pre -owide
```

```
raquel.abad@pc68506:~/k8s$ kubelee get  pods  -n awx-pre -owide
NAME                                           READY   STATUS    RESTARTS   AGE    IP              NODE
awx-76c46599fd-78rjw                           4/4     Running   0          18d    10.200.105.170  leela11
awx-operator-controller-manager-9479cd64b-cw2jb  2/2     Running   0          111m   10.200.101.110  leela14
awx-postgres-13-0                              1/1     Running   0          18d    10.200.105.181  leela11
```

Figura 3.2: Comprovació pods i adreces *IP* assignades (*default pool*)

Fins a aquest punt les adreces *IP* assignades a cada pod son triades d'un *default* calico ip *pool* configurat per defecte al clúster. Cada cop que un dels pods es reinicia l'adreça *IP* canvia i el node *worker* on s'executa el pod també pot variar.

A més, si visualitzem el script que defineix *l'endpoint* podem veure per quin port s'accedeix al servei.

Per comprova que tot es troba desplegat correctament, podem intentar accedir a la interfície web de l'AWX a través de l'adreça *IP* que té el pod de servei en aquest moment, amb la següent URL en el cas anterior: *https://10.200.105.170:8052* i ens ha de carregar la pàgina principal d'AWX.

3.4. Configuració credencials admin

Un dels primers passos necessaris per poder comprovar que l'aplicació s'ha desplegat correctament és crear un usuari administrador, per poder accedir via web com s'ha especificat a l'apartat anterior.

Per defecte, el *values.yaml* aplicat per l'awx-operator ens defineix un usuari admin, ja que és el valor per defecte de la variable admin_user. Però per crear una contrasenya per aquest usuari, cal crear un *secret* a kubernetes amb el nom awx-admin-password.

Aquest dos valors són els que agafarà l'objecte AWX desplegat al clúster per obtenir unes credencials d'accés amb l'usuari administrador.

3.5. Configuració del fitxer *values.yaml*

Com s'ha explicat a l'apartat 2.3.3. el fitxer *values.yaml* ens permet personalitzar l'aplicació segons els criteris i recursos necessaris. A part d'incloure l'usuari administrador per defecte, com s'ha explicat a l'apartat anterior, podem redefinir molts dels paràmetres.

Un dels paràmetres més importants del *values.yaml* és la definició de la imatge a partir de la qual es desplegarà l'aplicació. Com s'explica a l'apartat 2.3.5, la imatge proporciona els recursos mínims necessaris per desplegar l'aplicació.

Al fitxer *values.yaml* s'ha d'indicar el directori on es troba la imatge. En aquest cas, al paràmetre imatge de fitxer *values.yaml* li assignem el valor quay.io/ansible/awx, i al paràmetre image_version el valor latest. Això implica que aquesta aplicació es desplegarà sobre la última imatge publicada pel repositori oficial del fabricant d'AWX.

Un altre del segments importants del fitxer *values-pre.yaml* és on es defineix la base de dades. En aquest apartat, principalment es defineixen els recursos dedicats al pod de postgres que conté la base de dades. Podem limitar tant els recursos de CPU, com de RAM, com d'emmagatzematge.

Finalment, al *values-pre.yaml* s'han definit els recursos destinats a la interfície web, al motor de tasques i als execution enviroments.

Es pot veure l'script final del fitxer *values-pre.yaml* al l'annex 1.

3.6. Aplicació de les polítiques de xarxa

Com s'ha comentat a l'apartat 2.2.4. les polítiques de xarxa de Kubernetes permeten o deneguen la connectivitat entre pods, *namespaces* o rangs d'adreces *IP*. En aquest cas, hem utilitzat *les* polítiques de xarxa de Calico, ja que s'encarrega de la gestió de tota la apart de xarxa al clúster.

Per el desplegament d'aquesta aplicació, només es necessari que els pods del la pròpia aplicació es puguin veure entre si. Per aquest motiu la política de xarxa aplicada, permet el tràfic entre els diferents pods del *namespace*. L'script de la política de xarxa aplicada és el següent:

```
apiVersion: projectcalico.org/v3
kind: NetworkPolicy
metadata:
  name: allow-intra-ns-traffic-awx-pre
  namespace: awx-pre
spec:
  order: 301
  ingress:
  - action: Allow
    source:
      namespaceSelector: kubernetes.io/metadata.name == "awx-pre"
    destination: {}
```

Aplicat amb l'ordre:

```
callee apply -f net-pol.yaml
```

Per visualitzar les polítiques de xarxa aplicades podem utilizar l'ordre següent:

```
callee get networkpolicy -n awx-pre
```

3.7. Assignació *pool* d'adreces *IP*

Com s'ha comentat anteriorment, als pods se'ls assigna una adreça *IP* lliure un *pool* d'adreces *IP* assignat per defecte al clúster. En el cas que l'aplicació necessiti accedir a servei externs es necessari que el *pool* d'*IP*s que se li poden assignar als pods estigui més limitat.

A conseqüència d'haver de crear *pools* de dimensions més reduïdes, s'utilitzen els *Ippools* de Calico. Ens permeten assignar a les aplicacions *pools* més reduïts i d'aquesta manera només cal obrir accés a un servei extern des de unes adreces *IP* concretes.

L'script per crear un *pool* de Calico segueix el següent format:

```
apiVersion: projectcalico.org/v3
kind: IPPool
metadata:
  name: infrastructureenginering-awx-operator-pre-pool
spec:
  blockSize: 30
  cidr: 10.200.112.80/30
  ipipMode: Never
  natOutgoing: false
  nodeSelector: all()
  disabled: false
  vxlanMode: Never
```

En aquest recurs es defineixen les adreces *IP* que poden ser utilitzades per l'objecte que té assignat el *pool*. Com es pot observar en aquest cas es tracta d'un *pool* de quatre *IP*s que implica el rang 10.200.112.80/30. És necessari reservar aquestes adreces al DNS per evitar que per error siguin utilitzades per un altre servei.

Segons el format i els *templates* de l'aplicació a desplegar, es pot aplicar el *pool* d'adreces *IP* de Calico a nivell de *deployment* o a nivell de *namespace* a partir de les *annotations* de l'objecte.

En el cas de l'awx-operator, el *templates* no permetien afegir una *annotation* per assignar el *pool* de calico, per tant, ha estat assignat a nivell de *namespace*.

Per poder editar el fitxer que defineix el *namespace* ho podem fer amb l'ordre:

```
kubelee edit namespace awx-pre
```

Cal afegir a l'apartat d'*annotations* el paràmetre cni.projectcalico.org/ipv4pools i assignar el *pool* que hem creat anteriorment, com es mostra a continuació:

```
kind: Namespace
metadata:
  annotations:
    cni.projectcalico.org/ipv4pools: '["infrastructureenginering-awx-operator-
pre-pool"]'
```

A continuació cal reiniciar tots els pods amb l'ordre:

```
kubelee delete pods -n awx-pre <pod_name>
```

I quan es torni a crear el pod se li assignarà una adreça *IP* del *pool* definit. Podem comprovar que els canvis s'han aplicat correctament amb:

```
kubelee get pods -n awx-pre -owide
```

```
raquel.abad@pc68506:~/awx/awx-operator/templates$ kubelee get pods -n awx-pre -owide
NAME                                               READY   STATUS    RESTARTS       AGE    IP              NODE
awx-76c46599fd-w22tl                               4/4     Running   2 (116m ago)   121m   10.200.112.80   leela14
awx-operator-controller-manager-9479cd64b-wss4l    2/2     Running   0              121m   10.200.112.81   leela14
awx-postgres-13-0                                  1/1     Running   0              120m   10.200.112.82   leela13
```

Figura 3.3: Comprovació pods i adreces *IP* assignades

En la Figura 3.3 podem comprovar que les adreces *IP* assignades als tres pods pertanyen al *pool* que hem creat.

3.8. Integració del servei amb el balancejador

Per fer servir els balancejadors corporatius per publicar serveis s'han desplegat als clústers de Kubernetes uns controladors que s'encarreguen d'autodescobrir els serveis estàndard HyperText Transfer Protocol (HTTP) o Hypertext Transfer Protocol Secure (HTTPS) a publicar, generar la configuració necessària i connectar amb els balancejadors per aplicar-la.

Cada grup de controladors disposa de dos servidors virtuals un per al protocol HTTP, que escolta el port 80, i l'altre per al protocol HTTPS, que escolta el port 443. D'altra banda, els controladors venen aprovisionats amb un servidor virtual SSL de tipus *wildcard* que només és compatible amb les aplicacions que disposen d'un nom amb domini *.pre.upc.edu o *.upc.edu.

A més, al fitxer de configuració del Ingress podem afegir anotacions per definir les adreces *IP* que poden accedir al servei, indicar l'adreça *IP* del servidor virtual a utilitzar, configurar que el tràfic HTTP sigui permès en serveis HTTPS, definir un monitor pels *endpoints* del servei o redirigir el tràfic HTTP a HTTPS, entre d'altres.

En el cas de l'AWX, si observem el fitxer de configuració del servei, podem observar que s'ofereix per HTTP a través del port 80, però a *l'endpoint* s'accedeix pel port 8052 del pod. Tota aquesta estructura lògica és la que s'ha de definir al fitxer ingress.yaml.

El nom d'aplicació assignat és **awx.pre.upc.edu**. Aquest nom segueix les especificacions anteriors, ja que el nom segueix el patró de serveis de preproducció (*.pre.upc.edu) i, per tant, permet aplicar el certificat *wildcard* SSL. A més, segons el *service*, s'accedeix pel protocol HTTP, podem utilitzar els servidors virtuals estàndard.

Les configuracions escollides per definir el fitxer *ingress* són les següents:

- Indicarem el controlador que ha de gestionar l'entrada de tràfic.
- Utilitzarem el certificat SSL *wildcard* de *pre.upc.edu.
- Utilitzarem un monitor HTTP/1.1. (HTTP bàsic)
- Només està permès el tràfic des del rang d'adreces *IP* assignat al departament d'Infrastructure Engineering.
- Deixarem activada la redirecció de tràfic del protocol HTTP al protocol HTTPS, per seguretat.
- Deixarem desactivat l'accés de tràfic HTTP cap a HTTPS.
- Definirem el nom de l'aplicació, en aquest cas, awx.pre.upc.edu
- Seleccionarem el *service* que respon al *host* triat, per l'AWX el *service* és *awx-service* pel port 80.

El fitxer de configuració *Ingress* final tindrà l'estructura següent:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    ingress.kubernetes.io/allow-http: "false"
    ingress.kubernetes.io/ssl-redirect: "true"
    kubernetes.io/ingress.class: f5-eines-pre-ie-pre
    nginx.ingress.kubernetes.io/proxy-connect-timeout: "30"
    nginx.ingress.kubernetes.io/proxy-read-timeout: "1800"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "1800"
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
```

```
    virtual-server.f5.com/clientssl: '[ { "bigIpProfile":
"/Common/ssl_WILDCARD_.pre.upc.edu"}
      ]'
    virtual-server.f5.com/health: '[{"path":"awx.pre.upc.edu/", "send":"GET /
HTTP/1.1",
      "interval":5, "timeout":10}]'
    virtual-server.f5.com/whitelist-source-range: 10.6.130.0/25
  labels:
    app: awx
  name: awx
  namespace: awx-pre
spec:
  rules:
  - host: awx.pre.upc.edu
    http:
      paths:
      - backend:
          service:
            name: awx-service
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

Finalment, caldrà donar d'alta al DNS el *host* awx.pre.upc.edu com a CNAME del servidor virtual que escolta el port 443. Per comprovar, que el *host* es troba correctament creat al DNS, podem executar l'ordre:

```
nslookup awx.pre.upc.edu
Non-authoritative answer:
awx.pre.upc.edu     canonical name = vs-147-83-236-106.pre.upc.edu.
```

Si accedim al balancejador , podem comprovar que el servei ha estat autodescobert correctament, i s'han aplicat totes les configuracions definides.

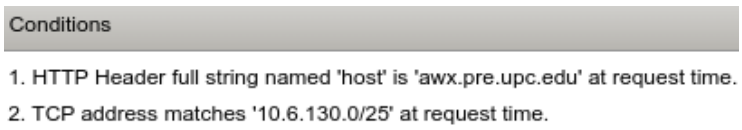

Figura 3.4: Comprovació integració balancejador

En definitiva, podem accedir al servei a partir de l'URL: https://awx.pre.upc.edu des de qualsevol adreça *IP* del rang permés a l'*Ingress*. I ens mostrarà una pantalla com la mostrada a la Figura 3.5:

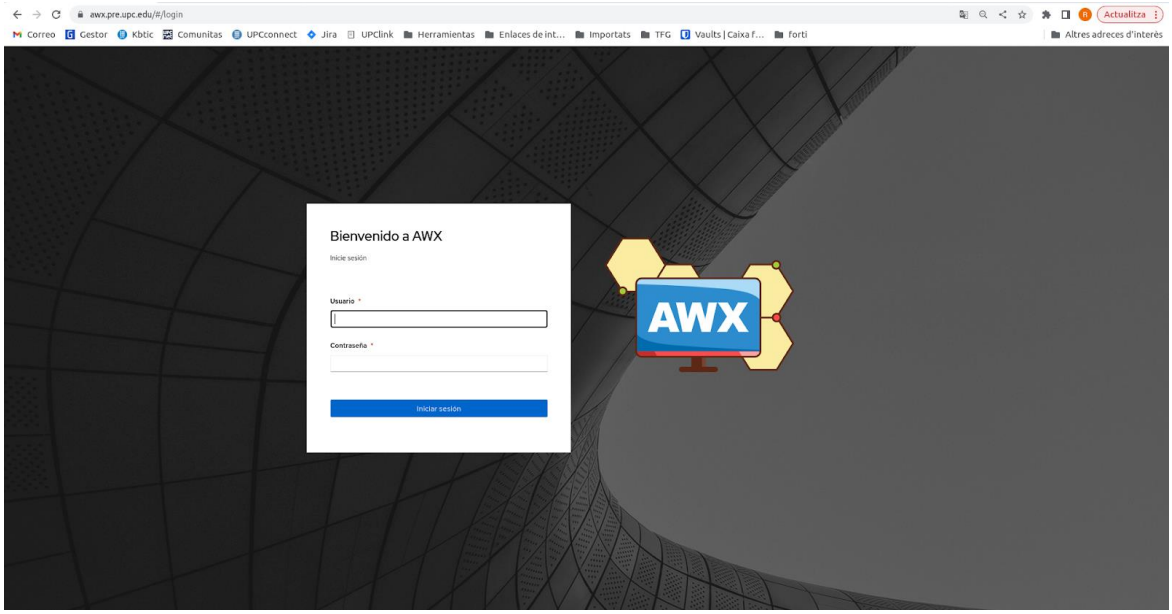

Figura 3.5: Pantalla *login* AWX

I podem accedir a la pantalla principal amb les credencials d'admin definides a l'apartat 3.4:

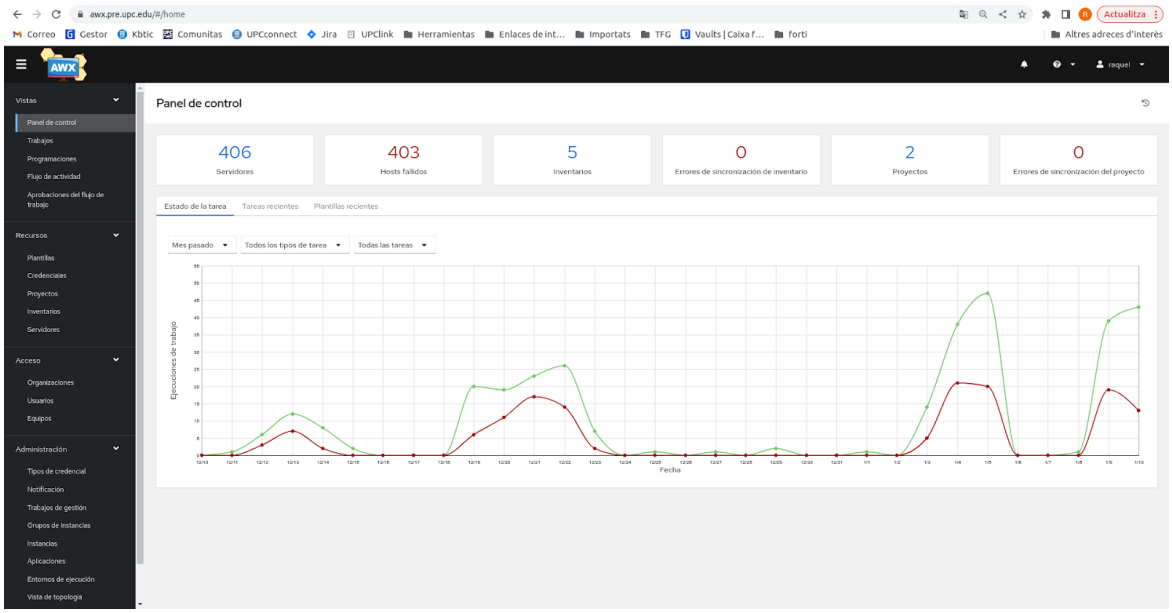

Figura 3.6: Pantalla principal AWX

3.9. Monitorització

Un cop tenim autogenerat el *pool* al balancejador es pot començar a monitoritzar, així quan el pod de servei caigui ens arribarà una alarma informant de que el servei ha caigut i podrem minimitzar el temps d'afectació. En el nostre cas, fem servei un a eina de monitorització anomenada Centreon.

3.10. Integració d'AWX amb el projecte de GitLab

Un cop es desplega correctament l'aplicació a Kubernetes i es pot accedir a la seva interfície web, és possible configurar els diferents objectes que utilitza AWX per llençar *playbooks*.

A AWX no es poden crear fitxers amb les descripcions dels *playbooks* ni dels rols. Tota aquesta configuració ha d'estar creada en un projecte al qual AWX pugui accedir i realitzar sincronitzacions per després executar-los contra els equips de l'inventari.

En aquest cas, s'ha decidit utilitzar un projecte de GitLab. En aquest projecte, trobem definits tots els *playbooks*, rols, *hosts* de l'inventari i variables respectives i fitxers de configuració que el formen.

Perquè AWX pugui accedir al projecte de GitLab se li ha creat un usuari, en aquest cas usuari.awx, amb accessos al projecte sol·licitat. Usuari.awx utilitza la seva clau Rivest, Shamir i Adleman (RSA) per connectar-se a GitLab i desarregar el projecte del path pertinent.

La configuració del projecte per accedir al repositori GitLab és la següent:

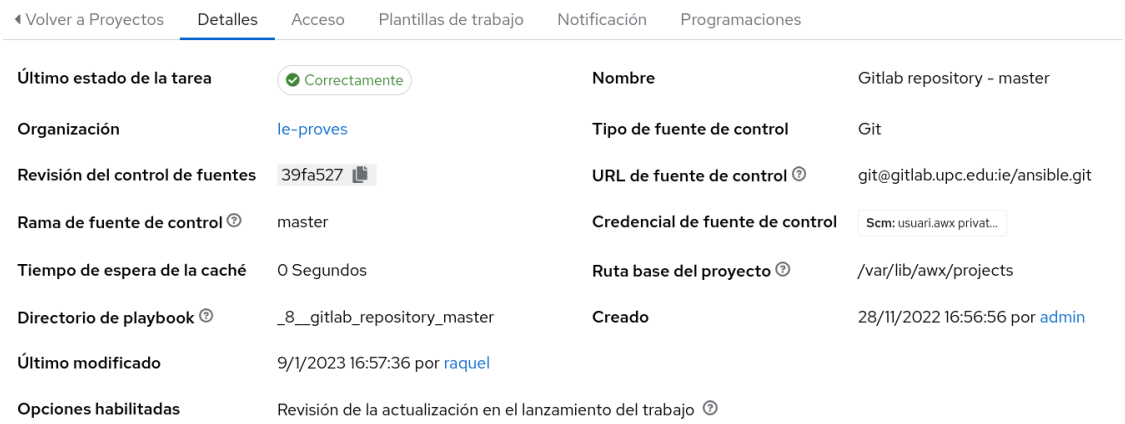

Figura 3.7: Configuració projecte AWX

Per poder accedir a GitLab , al projecte se li assigna una credencial de tipus "font de control" que conté la clau privada d'usuari.awx. Un cop realitzada la sincronització del projecte, es pot observar que al crear un *template* a l'apartat d'escollir *playbook*, s'obre un desplegable amb tots els *playbooks* disponibles al projecte.

En canvi, en relació a l'inventari, a AWX si que es poden crear inventaris manualment, simplement consisteix en crear un inventari nou, assignar els *hosts* respectius i les variables pertinents. A partir d'aquest moment es podien executar *playbooks* sobre aquest inventari. Però aprofitant que el projecte de GitLab ja conté fitxers que descriuen tot l'inventari, resulta més eficient descarregar-lo.

Un dels avantatges que proporciona l'inventari d'AWX és que se li pot definir una font que apunti al projecte d'accés a GitLab, explicat anteriorment, i triar el fitxer *hosts* que defineix l'inventari. D'aquesta manera quan es sincronitza l'inventari, AWX descarregarà tots els *hosts* amb les seves variables, les variables globals de l'inventari i la classificació per grups dels *hosts*, segons la definició del fitxer *hosts*.

Figura 3.8: Configuració inventari AWX

3.11. Desenvolupament de *playbooks*

En aquest punt s'explicarà com desenvolupar un *playbook*. Per realitzar proves s'han triat dos *playbooks* sencills llençats sobre dos entorns diferents.

En el primer cas, es tracta d'un *playbook* que accedeixi a un equip de xarxa, un *switch*, i extregui la informació hardware sol·licitada al *playbook*.

En el segon cas, consisteix en un *playbook* que accedeixi a un servidor Ubuntu i defineixi un directori nou. Per comprovar que s'executa correctament s'accedirà al servidor i es comprovarà que el directori ha estat creat.

3.11.1. *Playbook* executat sobre un *switch*

A causa de que la infraestructura de xarxa del campus està formada per equips Cisco i Dell i volem que aquest *playbook* es pugui executar sobre qualsevol equip, s'ha definit el *playbook* a partir de dos rols.

Quan el sistema operatiu sigui Cisco s'executarà el rol cisco_version i quan el sistema operatiu sigui Dell s'executarà el dell_version. Aquest condicionants es defineixen a partir de la paraula clau "*when*". També caldrà definir el nom de la tasca que s'està realitzant i els *hosts* sobre els que s'executa.

En aquest cas, la variable *hosts* està configurada de manera que si quan es llença el *playbook* a la variable se li assigna un valor concret, es llença contra aquest *host*, sinó contra tots els *hosts* de l'inventari assignat al *template*. La resta de variables que pertanyen al *host* estaran definides al fitxer *host_vars* corresponent.

Exemple del *playbook*:

```
---
- name: Configuracion de VLAN en interfaces Cisco/Dell
  hosts: "{{ host | default('all') }}"
  connection: network_cli

  rols:
    - { role: cisco_version, when: ansible_network_os == ("ios")}
    - { role: dell_version, when: ansible_network_os == ("dellos6")}
```

Exemple del rol pels equips Cisco:

```
---
- name: "Recolectando informacion de la version en equipos Cisco..."
  ios_facts:
    gather_subset:
      - hardware
  register: cisco_version

- name: "Mostrando la informacion"
  ansible.builtin.debug:
    msg:
    - "Nombre del equipo: {{ ansible_net_hostname }}"
    - "Modelo: {{ ansible_net_model }}"
    - "Imagen instalada: {{ ansible_net_image }}"
    - "Version de la imagen: {{ ansible_net_version }}"
    - "Serial Number: {{ ansible_net_serialnum }}"
```

3.11.2. *Playbook* executat sobre un servidor

En aquest cas, ja que AWX està pensat per la gestió de la infraestructura de xarxa i aquest *playbook* s'ha creat amb l'objectiu de realitzar proves, es tracta d'un *playbook* molt senzill que no fa servir rols i s'aplica sobre tots els *hosts* de l'inventari escollit al *template*. L'únic paràmetre a triar és el nom del nou directori. Exemple del *playbook*:

```
---
- name: file module demo
  hosts: all
  tasks:
    - name: Creating a directory
      ansible.builtin.file:
        path: "/tmp/{{ mydir }}"
        state: directory
```

3.12. Habilitar accessos

L'últim pas per poder llençar els *playbooks* contra els diferents equips consisteix en obrir els accessos necessaris per a que les adreces IP assignades als pods que es creen per executar els *playbooks* puguin arribar als equips finals.

En el cas del equips de xarxa, s'ha d'obrir accés per aquesta adreça *IP* a les *access-list* de gestió. En canvi, per als servidors l'accés s'ha d'obrir a través del tallafocs.

3.13. Altres configuracions d'AWX

AWX proporciona altres opcions que no han estat utilitzades durant la realització del projecte però que resulten útils a l'hora d'aplicar-lo en una empresa.

AWX permet separar els inventaris, els *playbooks* i el projectes per organitzacions. A més permet crear usuaris i equips, de manera que s'assignin a una organització, i el permisos que tinguin depenguin del rang d'usuari.

4. Resultats

En aquest apartat s'exposaran els diversos resultats extrets a partir dels diferents procediments explicats durant l'apartat 3 i es comprovaran i demostraran amb imatges que els resultats són els esperats i, per tant, el procediment funciona correctament.

4.1. Desplegament de l'aplicació

Es tracta del primer resultat obtingut a partir del correcte desplegament de l'aplicació al clúster de Kubernetes. Podem comprovar que els diferents pods s'han desplegat correctament i s'han creat els *services* i *endpoints* necessaris. També es pot observar que la interfície web funciona, es mostren les diferents pestanyes i les credencials creades per l'usuari admin són vàlides.

D'altra banda, podem comprovar que les polítiques de xarxa i el fitxer *Ingress* s'han aplicat correctament i que se li han assignat les adreces IP, corresponents al *pool* de Calico, als pods del *namespace*. Els nodes es poden comunicar entre ells com hem definit a nivell de política de xarxa i el balancejador autodescobreix el servei, amb el seu nom de *host* i l'adreça IP assignada al pod de *service*.

4.2. Integració amb Git

En aquest apartat s'explica el segon resultat obtingut a partir de l'integració d'AWX amb GitLab.

Es pot demostrar que AWX accedeix a GitLab amb l'usuari usuari.awx i es descarrega correctament el repositori indicat al projecte.

Aquest resultat el podem demostrar a partir dels *playbooks* que ens apareixen com a disponibles quan creem un *template*, que corresponen a tots els *playbooks* del projecte.

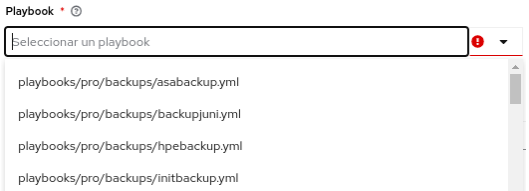

Figura 4.1: Comprovació integració GitLab (*playbooks*)

Però també es pot demostrar a partir de la sincronització de l'inventari. Un cop s'ha creat un inventari a partir d'un projecte d'integració amb GitLab, l'inventari és defineix automàticament segons els fitxers de configuració. Un des exemples més clars és la creació de grups o el registre del fitxers *host_vars*.

Figura 4.2: Comprovació integració GitLab (*host_vars*)

4.3. Execució de *playbooks*

En aquesta secció es mostren els resultats obtinguts a partir de la creació de *playbooks* per realitzar tasques de gestió a la xarxa. Que en definitiva és l'objectiu del projecte i d'on s'extreu el rendiment des del punt de vista empresarial.

Com s'ha comentat a l'apartat 3.11. S'han realitzat proves en dos entorns:

- Sobre equips d'infraestructura de xarxes (*switch*)
- Sobre màquines virtuals (servidors)

4.3.1. *Playbooks* per equips d'infraestructura de xarxes

En el cas dels *playbooks* executats sobre equips d'infraestructura de xarxes, s'ha desenvolupat un *playbook* que retorna informació útil de l'equip que se li passa com a variable. En aquest exemple, el *playbook* s'executa sobre un *switch* Cisco. A continuació es pot observar la informació que retorna el *playbook*:

```
0
1  PLAY [Configuracion de VLAN en interfaces Cisco/Dell] ************************** 13:29:38
2  [WARNING]: ansible-pylibssh not installed, falling back to paramiko
3
4  TASK [cisco_version : Recolectando informacion de la version en equipos Cisco...] ***
5  ok: [a4-sweth01]
6
7  TASK [cisco_version : Mostrando la informacion] *******************************
8  ok: [a4-sweth01] => {
9      "msg": [
10         "Nombre del equipo: A4-SWETH01",
11         "Modelo: WS-C3560X-48P",
12         "Imagen instalada: flash:/c3560e-universalk9-mz.122-53.SE2/c3560e-universalk9-mz.122-53.SE2.bin",
13         "Version de la imagen: 12.2(53)SE2",
14         "Serial Number: FDO1427K0JP"
15     ]
16 }
17
18 PLAY RECAP ********************************************************************* 13:29:44
19 a4-sweth01                : ok=2    changed=0    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
```

Figura 4.3: Execució *playbook* (I)

Per comprovar que aquesta informació és correcta, podem accedir manualment a l'equip i comprovar-ho:

```
Switch Ports Model              SW Version            SW Image
------ ----- -----              ----------            ----------
*    1 54    WS-C3560X-48P      12.2(53)SE2           C3560E-UNIVERSALK9-M
```

Figura 4.4: Comprovació playbook (I)

4.3.2. *Playbooks* per màquines virtuals

D'altra banda, en el cas dels *playbooks* executats sobre màquines virtuals, s'ha desenvolupat un *playbook* que crea una nova carpeta al directori escollit de la màquina virtual sobre la que s'executa. En aquest exemple, el *playbook* s'executa sobre un servidor Ubuntu 20.04. A continuació es pot observar la informació que retorna l'execució del *playbook*:

Figura 4.5: Execució playbook (II)

Per comprovar que el *playbook* s'ha executat correctament, podem accedir manualment al servidor i comprovar que existeix el nou directori:

Figura 4.6: Comprovació playbook (II)

En definitiva, els resultats són els esperats. AWX és una eina que proposa millorar l'automatització i l'orquestració de processos per la gestió de xarxes, i gràcies als *playbooks* i a les diferents integracions realitzades, pot resultar una eina molt productiva i senzilla d'utilitzar.

5. Costos

Si es calculen totes les hores dedicades al desenvolupament del projecte per el cost per hora del treballador i, aproximadament, s'han dedicat unes 25 hores a la setmana durant 12 setmanes. S'estima un cost d'uns 10 euros per hora. Obtenim el següent resultat:

$$25\ h/setmana * 12\ setmanes * 10\ €/h = 3000€$$

A més, caldria sumar les hores dedicades pel tutor de l'empresa al seguiment del projecte, es calculen aproximadament 1 hora a la setmana. En aquest cas, s'estima un cost d'uns 25 euros per hora. Per tant, s'obté el següent resultat:

$$1\ h/setmana * 12\ setmanes * 25\ €/h = 300€$$

Fins aquí, el cost humà suposa 3300€.

D'altra banda, com que el projecte s'ha desenvolupat de manera presencial a l'empresa UPCnet, s'han consumit recursos que es faciliten a l'oficina. Per exemple, costos d'electricitat, internet, calefacció, aigua, lloguer... Això suposa un impediment que no permet calcular el cost exacte de projecte.

6. Conclusions i desenvolupament futur

6.1. Conclusions

La finalitat d'aquest desplegament consistia principalment en obtenir una eina per l'orquestració i l'automatització de processos per la gestió de xarxes. Gràcies a les possibilitats que ens ofereix AWX, i tal i com hem estructurat el nostre projecte, s'ha aconseguit que compleixi el seu propòsit.

Com s'ha pogut comprovar a partir del resultats extrets a l'apartat 4, tant amb l'execució de *playbooks* sobre un *switch* com sobre un servidor, l'aplicació ha estat desplegada correctament a Kubernetes i compleix el seu objectiu.

Actualment, AWX ens permet automatitzar processos de gestió de xarxa, a partir de la creació de *playbooks*, i aplicar-los sobre qualsevol equip del nostre inventari, gestionant la seva execució des d'una plataforma intuitiva per l'usuari. Aquest fet pot provocar que els administradors de xarxa explotin molt més les possibilitats que ofereix AWX (o Ansible).

D'altra banda, es pot observar que el creixement d'aquesta eina no està limitat i pot arribar a tenir tantes aplicacions i funcionalitats com *playbooks* es vulguin desenvolupar.

Finalment, destacar que s'han complert tots els objectius proposats a l'inici del projecte:

- S'ha conegut l'entorn de Kubernetes, s'ha estès el funcionament, l'estructura i les característiques del *software* i s'han conegut els tipus d'objecte i recursos necessaris per desplegar aplicacions.
- S'ha après a utilitzar Helm per desplegar aplicacions a Kubernetes.
- S'ha conegut el funcionament d'Ansible, els components que el formen i s'ha après a desenvolupar *playbooks* i rols per executar sobre l'inventari.
- S'ha après a fer servir GitLab i s'ha utilitzat com a eina per realitzar canvis sobre el projecte d'Ansible triat.
- S'ha desplegat l'aplicació AWX a un clúster de Kubernetes utilitzant Helm.
- S'han creat dos *playbook* per la gestió dels equips de xarxa i s'ha comprovat i demostrat el seu funcionament quan s'executa a través d'AWX.

6.2. Desenvolupament futur

El futur d'AWX, possiblement, es basi en començar a utilitzar-lo com a eina principal per la gestió d'equips de xarxa. Com, per exemple, realitzar canvis homogenis en diversos equips, poder executar-los de manera centralitzada i a través d'una plataforma senzilla d'utilitzar.

A l'inici s'haurà d'invertir temps en desenvolupar *playbooks* i rols que executin els diferents processos, però a llarg termini facilitarà la feina d'administradors de xarxa. Inclús podria permetre delegar algunes de les tasques a altres equips o administradors, ja que podrien obtenir un usuari que no els permeti modificar però si executar els *playbooks*.

D'altra banda, és una eina que es pot vendre a clients i permetria gestionar les seves xarxes també de manera centralitzada i disminuint la possibilitat d'error per part dels tècnics, apart de convertir-se en un servei més dins del catàleg de l'empresa.

Acrònims

Acrònim	Definició
K8s	Kubernetes
CNAME	Canonical Name
DNS	Domain Name System
IP	Protocol d'Internet
YAML	*Yet Another Markup Language*
SSH	Secure Shell
TI	Tecnologies de la Informació
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
RSA	Rivest, Shamir i Adleman

Bibliografia:

[1] Paul Sawers, "The state of cloud-native development: Kubernetes is on the rise". Desembre 2021. [*Online*] https://venturebeat.com/business/the-state-of-cloud-native-development-kubernetes-is-on-the-rise/. [Accés: Gener 2023]

[2] Chris Munford, "A DevOps reset for a multi-cloud world". Desembre 2022. [*Online*] https://www.cncf.io/blog/2022/12/20/a-devops-reset-for-a-multi-cloud-world/ [Accés: Gener 2023]

[3] Aleix Abrie, "Diferencias Kubernetes vs Docker Swarm". Octubre 2020. [*Online*] https://www.icm.es/2020/10/30/kubernetes-docker-swarm/ [Accés: Gener 2023]

[4] "Kubernetes" https://kubernetes.io/es/ [Accés: Gener 2023]

[5] "Helm" https://helm.sh/es/ [Accés: Gener 2023]

[6] "GitLab" https://about.gitlab.com/ [Accés: Gener 2023]

[7] "Ansible" https://docs.ansible.com/ [Accés: Gener 2023]

[8] "AWX" https://www.ansible.com/products/awx-project/faq [Accés: Gener 2023]

[9] "Calico" https://projectcalico.docs.tigera.io/networking/ipam [Accés: Gener 2023]

[10] "Rancher" https://www.rancher.com/ [Accés: Gener 2023]

[11] "awx-operator" https://github.com/ansible/awx-operator [Accés: Gener 2023]

Annexos:

Annex 1: Fitxer *values-pre.yaml*

```
AWX:
  # AWX container configuration

  # habilita utilitzar la pantilla "awx-deploy"
  enabled: true
  name: awx
  kind: app

  spec:
    replicas: 1
    service_type: ClusterIP
    ingress_type: none

     ## Recursos de postgres
    postgres_image: postgres:13
    postgres_init_container_resource_requirements:
      requests:
        cpu: 10m
        memory: 64Mi
      limits:
        cpu: 500m
        memory: 2Gi

    postgres_resource_requirements:
      requests:
        cpu: 10m
        memory: 64Mi
      limits:
        cpu: 500m
        memory: 2Gi

    postgres_storage_class: ssd
    postgres_storage_requirements:
      requests:
        storage: 50Gi
    postgres_data_path: /var/lib/postgresql/data/pgdata
    postgres_extra_args:
      - '-c'
      - 'max_connections=1000'

    ###Imatge
    image: quay.io/ansible/awx
    image_version: latest
    image_pull_policy: IfNotPresent
    ee_images:
      - name: AWX EE (latest)
        image: quay.io/ansible/awx-ee:latest

    task_extra_volume_mounts: |
      - name: "sshkey"
        mountPath: "/var/lib/awx/.ssh"
        subPath: id_rsa
        secret:
          secretName: awx-ssh-key
```

```
          defaultMode: 0777

    web_extra_volume_mounts: |
      - name: "sshkey"
        mountPath: "/var/lib/awx/.ssh"

    ### Requisits de recursos de contenidors
    web_resource_requirements:
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 250m
        memory: 2Gi
    task_resource_requirements:
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 250m
        memory: 1Gi
    ee_resource_requirements:
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 250m
        memory: 2Gi

    extra_settings:
      - setting: AUTH_LDAP_SERVER_URI
        value: '"ldaps://ldap.upc.edu"; from django_auth_ldap.config import
GroupOfNamesType'

      - setting: AUTH_LDAP_GROUP_TYPE
        value: "GroupOfNamesType(name_attr='cn')"

    ##Persistencia directori projectes
    projects_persistence: true
    projects_storage_class: nfs-client
    projects_storage_size: 50Gi
    projects_storage_access_mode: ReadWriteMany

  ## Servei
  service:
    enabled: true
    ports:
    - name: "http"
      port: 80
      targetPort: 80
```