SSH Configuration Deployment Service (SCDS)

Universitat Oberta de Catalunya

Grau en Enginyeria Informàtica

Daniel López López
dlopezlopez@uoc.edu

Joaquín López Sánchez-Montañés

15 de juny de 2024

Resum

This project proposes a solution for the maintenance and distribution of ssh configurations. A web application in Go is developed to dynamically generate ssh configuration files based on a template, server connection data in Json format, and the user identifier. The maintenance of configurations is then delegated to users through pull request workflows and code reviews.

For the ci/cd pipeline, GitHub Actions is used to ensure code quality, build, and publish the application's container after each new code integration. The final deployment is carried out automatically into a Kubernetes cluster using Helm and GitOps with FluxCD. This system improves efficiency and reliability in managing ssh configurations with a robust and scalable solution that utilizes standardized practices in the modern deployment of solutions.

Aquest projecte proposa una solució al manteniment i distribució de configuracions ssh. Es desenvolupa una aplicació web en Go que genera dinàmicament fitxers de configuració ssh a partir d'una plantilla, les dades de connexió dels servidors en format Json i l'identificador de l'usuari. El manteniment de les configuracions es delega als usuaris mitjançant fluxes de *pull requests* i *revisions de codi*.

Per al pipeline de ci/cd s'utilitza GitHub Actions per assegurar la qualitat del codi, construir i publicar el contenidor de l'aplicació després de cada integració de nou codi. El desplegament final es realitza automàticament en un clúster de Kubernetes usant Helm i GitOps amb FluxCD. Aquest sistema millora l'eficiència i la fiabilitat en la gestió de configuracions ssh amb una solució robusta i escalable que usa pràctiques estandarditzades en el desplegament moder de solucions.

Keywords— ssh, ci/cd, Kubernetes, IaC, GitOps

Dedicatòria

Dedico aquest treball a les tres dones de la meva vida:

A la meva parella, Tania: L'acompanyant durant el camí. Sense el teu suport i generositat no m'hauria estat possible arribar fins aquí.

A la meva filla, Judit: La raó i el sentit de la meva vida. Tot és per tu.

A les dues us dec moltes hores.

A la mare, Paqui: El principi de tot. Poder fer-te aquesta dedicatòria era una de les grans motivacions. Tant de bo ho poguessis veure.

Índex

Índex de figures

Índex de codis font i comandes

Capítol 1

Introducció

1.1 Descripció del Treball

1.1.1 El problema

En organitzacions mitjanes i grans ens trobem sovint que el nombre de servidors GNU/Linux i derivats d'Unix que s'han d'administrar (físics o virtuals) acostuma a ser elevat, des de desenes a centenars. En alguns casos aquest nombre és encara més gran i pot ser dinàmic degut al propi cicle de vida dels propis servidors i serveis que suporten (altes, baixes, manteniments, reinstal·lacions, etc).

La connexió a aquests servidors requereix d'uns certs paràmetres que cal recordar:

- L'adreça ip o el nom de *host*.
- L'usuari de connexió (el propi o un d'específic per a una instància concreta).
- El port d'accés pot ser diferent de l'estàndard.
- Cal usar una clau d'accés concreta per a cada conjunt de servidors.
- Per a certs servidors cal usar màquines específiques de salt (Bastion Server).
- Prèviament hem d'executar algun mètode concret per habilitar la connexió, com per exemple, port-knocking (https://linux.die.net/man/1/knockd), Simple Package Authentication (https://github.com/mrash/fwknop) o crides per afegir temporalment la nostra ip a una llista d'acceptació al proveïdor del núvol, entre d'altres.

Recordar de memòria totes aquestes configuracions no és viable si la llista de nodes és prou gran o si aquesta varia sovint. En els casos especials en que hem d'escriure paràmetres de connexió es pot convertir en una tasca tediosa i poc pràctica. Es per això que sovint usem eines de gestió de connexions (gràfiques o textuals, lliures o amb llicències comercials), incloent el propi fitxer de configuració que utilitza el client d'openSSH " /.ssh/config".

En els entorns en que basem aquest treball el nombre d'usuaris sol ser també molt gran. Mantenir la configuració actualitzada i distribuir-la esdevé un repte important que cal abordar.

Hi ha múltiples formes d'adreçar aquest problema, totes bones, algunes de molt senzilles que alhora plantegen altres reptes com: traçabilitat, versionat, adaptabilitat en funció del destinatari, homogeneïtat, centralització, etc. L'elecció de la solució dependrà molt del nombre de configuracions, la seva variabilitat, la maduresa de l'empresa, els recursos disponibles, entre d'altres. Tanmateix, no hi ha cap solució perfecta que s'ajusti a totes les situacions i pressupostos.

En aquest projecte desenvoluparem un servei propi que dona resposta als anteriors reptes i que serà fàcil de mantenir i que s'adequarà a aquelles empreses que tinguin una certa maduresa tecnològica on la solució es pugui integrar sense generar un cost econòmic extra ni un manteniment elevat. El projecte es pot desplegar de diferents maneres per adaptar-se a les diverses tecnologies de l'empresa.

1.1.2 La proposta de solució

Crearem un petit servei web que rebrà com a paràmetre el nom d'usuari de sistema (*username*) i llegirà les dades de la configuració de ssh d'un fitxer Json. Amb aquestes dades i una plantilla predefinida l'aplicació generarà la configuració ssh actualitzada i personalitzada per a aquest usuari que la farà servir per actualitzar la configuració del seu sistema.

L'usuari realitzarà una crida com la següent:

```
1 $ > wget https :// scds.tydnet.org/sshconfig/myuser -O ~/.ssh/config
```

Codi 1.1: Exemple de crida del client

i el servei retornaria un fitxer com el següent:

```
### SCDS Generated config ###
Host  east-bastion  bastion1
 Hostname east-bastion.tydnet.org
 Port 30022

Host  west-bastion  bastion2
 Hostname west-bastion.tydnet.org
 Port 30022

Host  leia   leia.tydnet.org
 Hostname 10.77.1.10
 ProxyJump east-bastion ProxyCommand bash -c "fwknop -w /usr/local/bin/wget -R -n steam; sleep
     3; nc %h %p"

Host doodi
 Hostname doodi.tydnet.org
 User myuser
 Port 22022
 ProxyCommand bash -c "fwknop -R -n doodi; sleep 3; nc %h %p"

#### DO NOT ADD ANY ENTRIES BELOW THIS LINE! ####
#### Values below will be added to all above Host entries unless specifically overridden ####
#### Customized/Additional SSH Host entries should go in personal_config file ####

Host *
 ForwardAgent yes
 ForwardX11Trusted yes
 ServerAliveInterval 30
 ServerAliveCountMax 5
 UseKeyChain yes
 AddkeysToAgent yes
 User myuser
 Port 22
```

Codi 1.2: Exemple de configuració ssh

La sortida del servei la podem redirigir a un fitxer per a utilitzar-la directament i cridar-la sota demanda quan vulguem actualitzar-la utilitzar els servei de crontab de les nostres màquines:

```
$ > wget https :// scds.tfguoc.com/myuser | tee ~/.ssh/config
```

Codi 1.3: Aplicació de la configuració ssh

L'usuari ja pot accedir als servidors (sempre que prèviament tingui accés) i accedir a hosts on la configuració era tediosa indicant només l'àlies de host:

```
$> ssh doodi
```

Codi 1.4: Accés a un servidor ssh

El Servei

Aquest servei web està codificat en Go i fa servir una plantilla que omple amb les dades que llegeix d'un fitxer *Json* i el nom d'usuari formant la configuració que li servirà. En futures versions es pot ampliar el seu abast amb noves funcionalitats i millores com, per exemple, la introducció d'altres tipus de configuracions, autenticació del servei, altres formats de fitxer de dades, etc.

El cicle de vida del projecte

Tant el codi de l'aplicació web com les dades es guardaran en un repositori git. Els canvis s'aplicaran usant *pull requests* que s'han de revisar i aprovar com en qualsevol altre projecte de codi. En aquest punt s'aplicaran revisions automàtiques de qualitat i seguretat pròpies del cicle de vida de desenvolupament del codi que ens apropin el màxim possible als estàndards de qualitats desitjats.

Un cop integrat el codi a la branca principal es generarà automàticament una imatge en forma de contenidor que es publicarà a un registre d'artefactes públic i es desencadenarà el procés de Desplegament continu (cd) usant eines de GitOps[17] que publicarà la darrera versió del contenidor en el clúster de Kubernetes.

Un cop publicat el servei, el manteniment i actualització de les dades de l'aplicació el podem delegar als usuaris (shift-left) que en aquest cas poden ser els propis administradors de sistemes però no estarà limitat només a aquest grup. Aquells actors que necessitin incorporar, eliminar o modificar servidors podran descarregar-se el codi del repositori, crear una branca nova basada en la principal, realitzar els canvis al fitxer Json i crear una nova pull request amb els canvis desitjats. Un cop els mantenidors del repositori revisin els canvis proposats, els aprovaran i integraran amb el codi principal i es posarà en producció la nova versió.

1.1.3 Objectius

La motivació principal d'aquest treball és la d'explorar, adquirir coneixements i guanyar experiència pràctica en tecnologies que són importants i d'ús comú en àmbits laborals relacionats amb el Site Reliability[13] i DevOps. Finalitzat aquest treball haurem assolit els següents objectius:

- Es desenvolupa una aplicació web que serveix la configuració dssh actualitzada per a l'usuari que la sol·licita.
- El desplegament successiu de la solució es realitza automàticament mitjançant únicament interaccions amb el repositori de codi.
- L'arquitectura de la solució és reutilitzable en la resolució d'altres projectes.

- S'adquireix experiència pràctica demostrable en les àrees i tecnologies del projecte.

1.1.4 Abast del projecte

Aquest és un projecte ambició d'ampli abast i molt transversal que implica moltes àrees de coneixement (sistemes, cloud, networking, codificació, seguretat) i tecnologies que s'han d'integrar entre elles. Cadascuna d'aquestes àrees podrien ser l'objectiu únic d'un treball final dels estudis per elles mateixes. El coneixement de partida en cadascuna de les àrees és desigual i es requereix d'un treball d'investigació i aprenentatge d'algunes d'aquestes que suposa gran part del pla de treball. Per tant, en la planificació de les funcionalitats a desenvolupar s'han hagut de fer concessions, pressupostos i posar-hi límits a l'abast. Posarem diversos exemples:

Durant el desenvolupament del projecte els repositoris de codi han estat configurats com a privats i tots els fluxos que interactuen amb aquests autentiquen prèviament. Tanmateix, en lliurar la memòria final del treball els repositoris seran públics per tal de facilitar la tasca de correcció per part del consultor i tribunal. S'ha tingut cura de no publicar cap credencial ni dades que es puguin considerar sensibles.

Per al tractament de secrets, quan ha estat possible, s'ha usat el repositori de Google, en altres ocasions s'han emmagatzemat com a secrets de Kubernetes, que si bé és acceptable no ofereix un grau empresarial de seguretat ni segueix les millors pràctiques. Una aproximació conforme amb alts estàndards de seguretat passaria per l'ús d'eines de gestió de secrets en xarxa que hagués elevat considerablement la complexitat i cost del projecte.

El servei *SCDS* està desplegat en Kubernetes de manera pública i sense autenticació d'una banda per facilitar l'accés dels correctors d'altra per a reduir-ne la complexitat. Les dades que s'han usat en aquest projecte són fictícies i la seva publicació no suposa un problema. Afegir-hi una autenticació amb un secret compartit seria molt simple d'implementar però introduiria altres desafiaments, com la gestió, manteniment i distribució de credencials i altres solucions estarien fora de l'abast del projecte. Hem d'assumir que en un entorn real l'aplicació només seria accessible des de dintre del perímetre empresarial.

S'han cobert totes les fites proposades però en cada àrea (desenvolupament, seguretat, tests de qualitat, infraestructura) s'han efectuat de manera que cobreixen els requisits i s'han desestimat funcionalitats exhaustives que es podrien haver realitzat amb més temps.

Durant el project hem hagut d'usar un domini dns per a resoldre l'adreça de l'aplicació i aquesta ha estat configurat en un servidor *ISC Bind*. S'ha optat per reutilitzar recursos existents tot i que no estaven preparats per a ser configurats amb *IaC*. Si no haguéssim comptat amb dominis previs propis l'hauríem adquirit en un proveïdor

que suportés la seva configuració amb *Terraform*.

1.1.5 Planificació de treball

Hem dividit el projecte en els quatre lliuraments proposats: P1, P2, P3 i Lliurament final (P4). Les dates coincidiran amb les fites de cada fase descrites en el gràfic del plan de Gantt. A continuació una descripció de les tasques més importants de cada fase:

- P1: S'estudia el projecte, es realitzen proves de concepte i el lliura la proposta del projecte.
- P2: Es desenvolupa un cicle de desplegament de l'aplicació aproximat en un entorn de desenvolupament. Es lliura part de la memòria amb la descripció actual del projecte.
- P3: S'implementa l'aplicació final i el desplegament en un entorn de producció. Es lliura la memòria final amb les conclusions.
- P4: Correccions de la memòria i del projecte, si escau. Preparació de la defensa del treball. Es lliura la memòria definitiva, la presentació i el vídeo de defensa.

En la planificació es consideren com a hàbils tots els dies de la setmana, de dilluns a diumenge, amb una dedicació mitja setmanal de vint-i-cinc hores. Només s'han considerat com a festius o inhàbils alguns dies on podem preveure que els compromisos laborals o familiars ens impediran de assolir la dedicació proposada.

A continuació s'inclou una planificació setmanal del projecte amb les tasques de més alt nivell 1.1 i posteriorment també una altra diària, més detallada i amb les tasques principals desglossades.

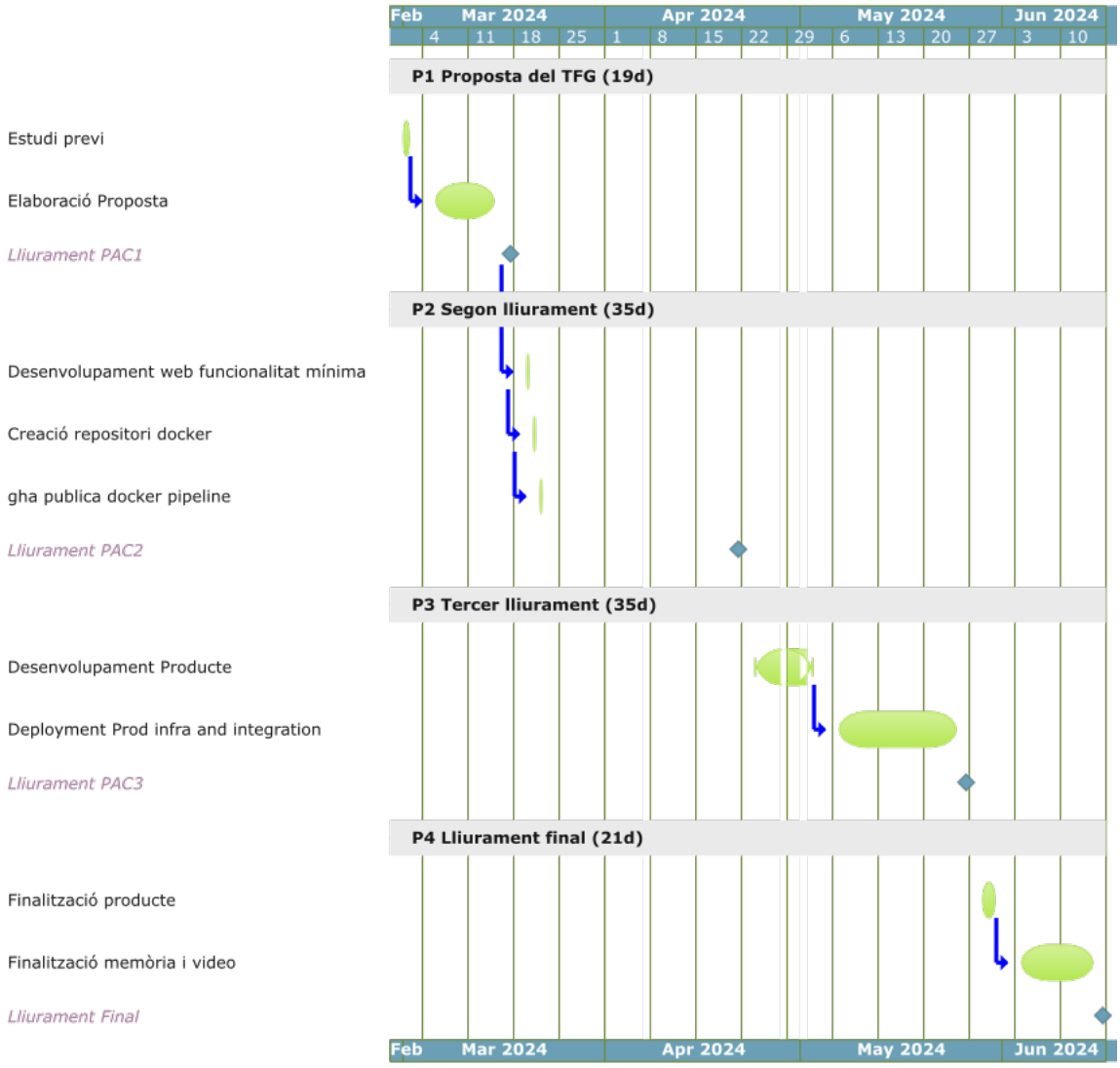

Figura 1.1: Planificació setmanal

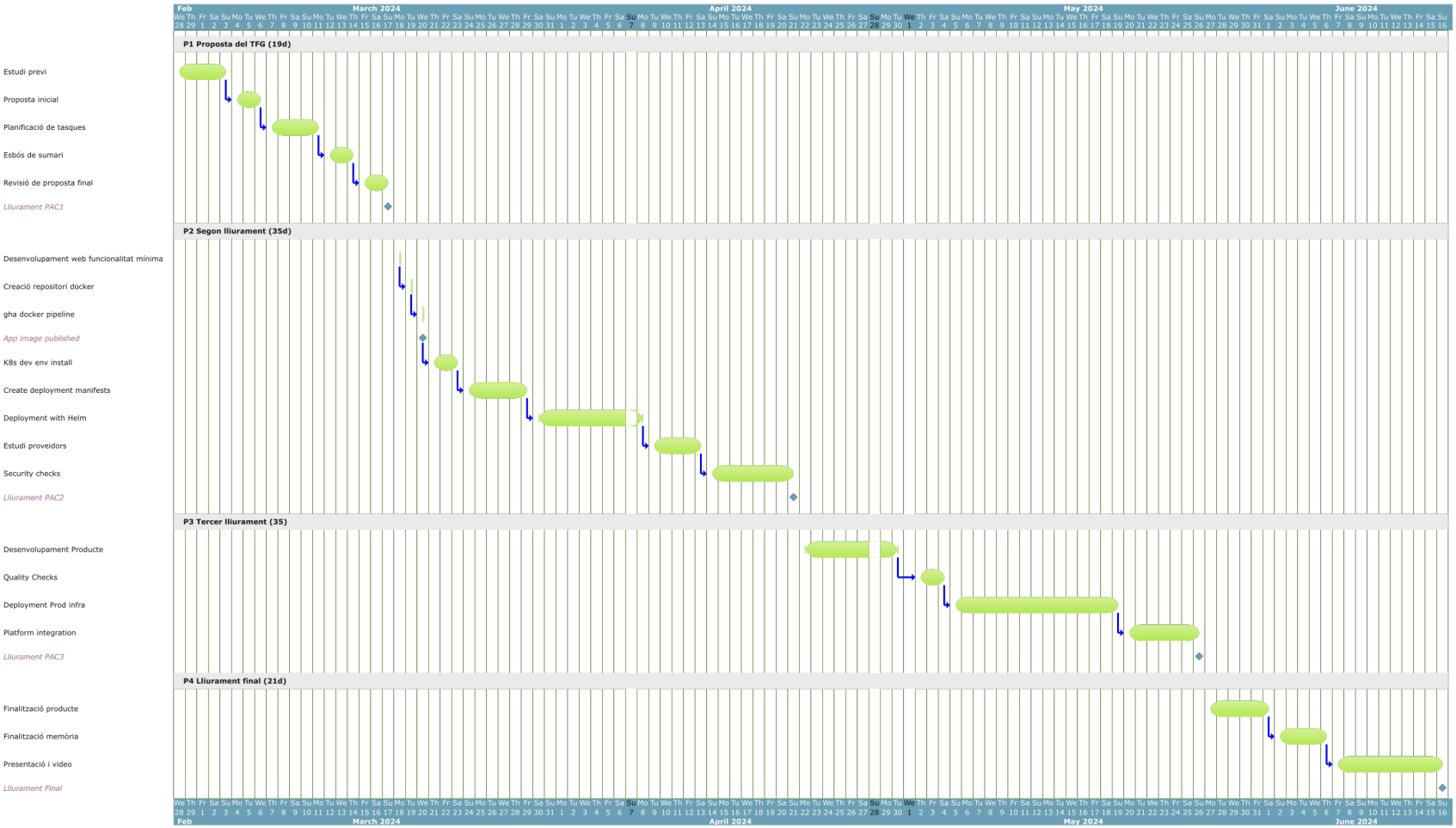

Figura 1.2: Planificació diària

La definició i seguiment de les tasques del projecte es fa mitjançant el servei Trello [1] en la seva versió gratuïta. Les tasques es classifiquen i agrupen segons els diversos lliuraments en format de targetes on es documentes les accions a realitzar i es prioritzen segons la seva criticitat i pes dintre del projecte.

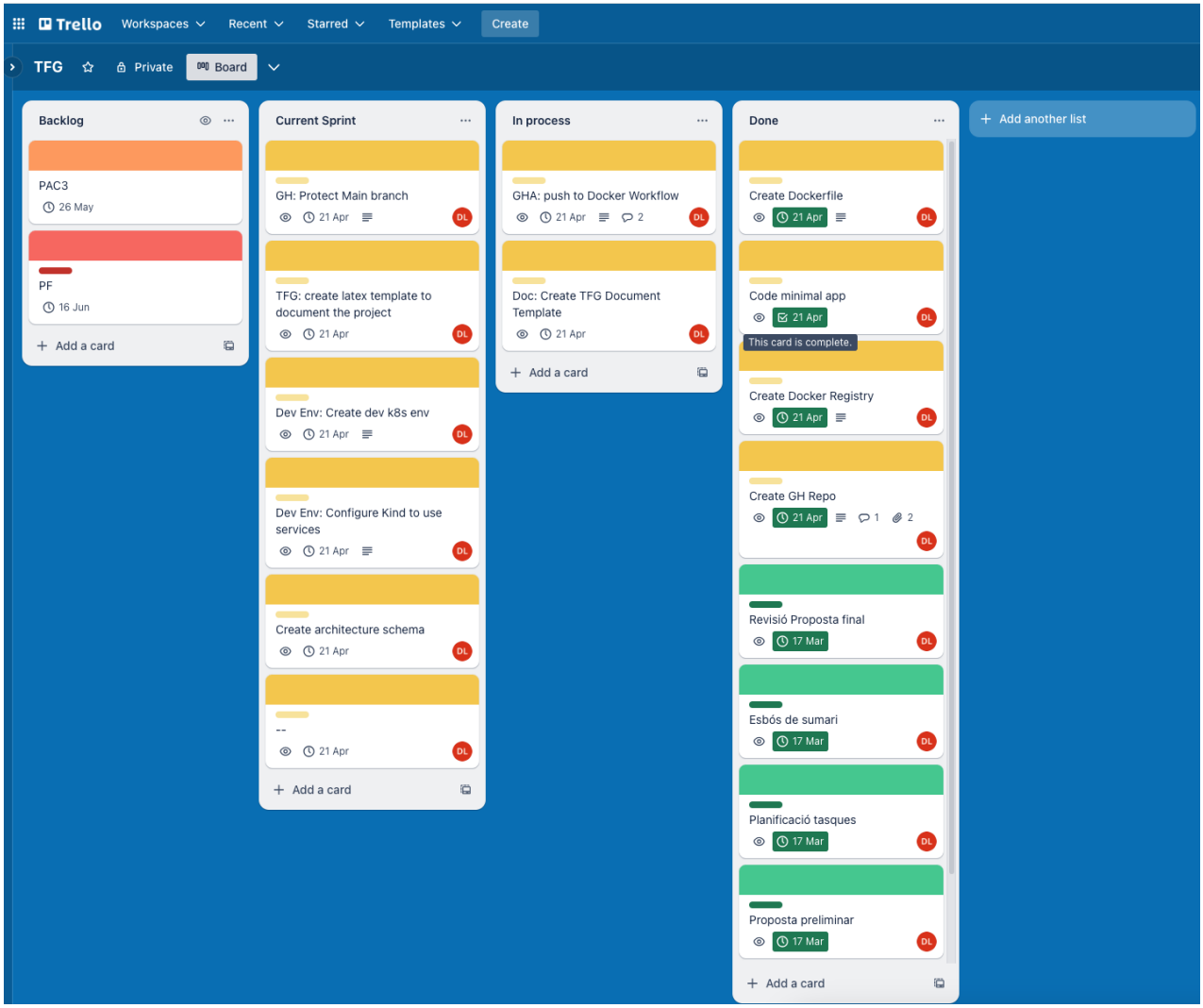

Figura 1.3: Trello Board

Capítol 2

El projecte

2.1 Introducció

En aquest apartat veurem en detall en què consisteix la solució que hem desenvolupat, quines parts la formen, com estan implementades i com s'interrelacionen entre elles. Començarem amb una visió general de l'arquitectura per a situar-nos seguida d'una breu descripció de les tecnologies usades per poder entrar en els detalls de la implementació.

2.2 Requeriments

Durant l'execució d'aquest treball usarem diverses tecnologies, serveis i plataformes. Hem escollit les que hem considerat com a estàndards de facto actualment dins de la indústria. Entre d'altres i com a requeriment imposat per les millors pràctiques, s'usarà IaC, Infraestructure as Code (Terraform i Kubernetes), Pipelines as Code (GitHub Actions) i Gantt as Code (mitjançant PlantUML) per tal de maximitzar la documentació, la traçabilitat dels canvis i la reusabilitat dels components.

Tecnologies, serveis i infraestructures:

- **Go:** [14] és el llenguatge usat per a programar l'aplicació web. Una de les avantatges que té és què és compilat i la imatge final serà molt petita, al voltant dels 13MB.
- **Git (github):** [10] Git i Github són la tecnologia i el servei de control de versions per a codi. Ens proveeix també d'eines d'integració continua ci.
- **Github Actions:**[9] És la tecnologia d'integració continua (ci)de Github i que usem per a integrar el codi, córrer els testos de qualitat i crear l'artefacte final de l'aplicació.
- **Docker:** [5] És la tecnologia de gestió de contenidors usada en l'entorn de desenvolupament. Kubernetes usa altres tecnologies per a gestionar contenidors.

- **Docker Registry:** [6] Repositori on allotjarem les imatges dels contenidors de l'aplicació.
- **Kubernetes:** Usem dues tecnologies diferents per entorn. KinD [19] (desenvolupament), Kubernetes a DigitalOcean [8](producció). És la plataforma d'orquestració de contenidors on publicarem la versió final de l'aplicació.
- **Helm:** [16] Gestor de paquets de manifest de Kubernetes. Ens ajuda a estructurar l'aplicació en un contenidor únic (chart) i parametritzar els valors de la configuració del desplegament de l'aplicació per entorns sense repetir codi.
- **Terraform:** [15] Eina multi-proveïdor de gestió d'Infraestructura com a codi. Ens permet automatitzar el desplegament de la infraestructura. documentar-la i gestionar-la des del repositori de codi.
- **Gantt en PlantUML:** [2] Eina de creació de diagrames com a codi.
- **FluxCD:** [7] Eina de GitOps per al desplegament de l'aplicació a partir de commits al repositori de git.

2.3 L'arquitectura

Com podem veure a l'esquema de l'arquitectura a la imatge 2.1 tenim la nostra aplicació instal·lada dins d'un clúster de Kubernetes. Els usuaris poden recuperar la configuració ssh amb una simple crida http contra el servei publicat.

Quan un dels administradors (o qualsevol altre usuari amb accés d'escriptura al repositori) vol afegir algun canvi només ha de descarregar-se el codi de ***scds*** localment i crear una nova branca amb els seus canvis en el fitxer *ssh-data.yaml*. Crearà una nova *pull request* que haurà de revisar un dels mantenidors del repositori. Si s'aprova el canvi s'integrarà a la branca principal i es desencadenarà el procés d'integració continua desplegant una nova versió de l'aplicació en el clúster de Kubernetes.

2.4 Preparació de l'entorn de desenvolupament

L'eina escollida per a implementar el clúster de *Kubernetes* és KinD. KinD és una implementació de K8s en contenidor que permet fàcilment generar clústers amb múltiples nodes. Instal·lem *kubectl* i *KinD* mitjançant el gestor de paquets de macos *brew* veure 2.1

```
1 $ > brew install kind kubectl
2
```

Codi 2.1: Install kind

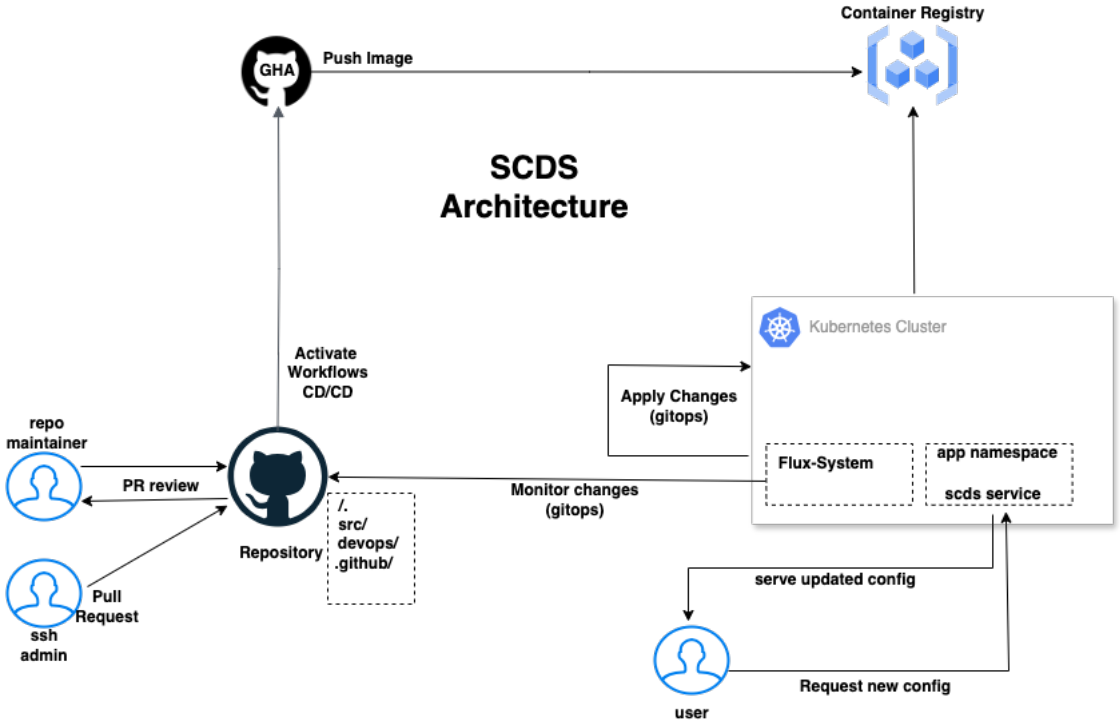

Figura 2.1: L'arquitectura de SCDS

Un cop finalitzada la instal·lació de les eines podem continuar amb la creació del clúster local amb una sola comanda i un fitxer de configuració:

```
~/Documents/universitat/current/TFG/repo/scds/devops/kind main ?8 kind create cluster --name tfg-dev --config kind-tfgdev.yaml
Creating cluster "tfg-dev" ...
 ✓ Ensuring node image (kindest/node:v1.29.2) 🖼
 ✓ Preparing nodes 📦 📦 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹️
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
 ✓ Joining worker nodes 🚜
Set kubectl context to "kind-tfg-dev"
You can now use your cluster with:

kubectl cluster-info --context kind-tfg-dev

Have a nice day! 👋
```

Figura 2.2: Cluster amb Kind

El contingut del fitxer de configuració per a desplegar el clúster amb tres nodes el podem veure a 4.7. Ja tenim disponible el nostre clúster local, veure 2.2:

```
$> k get nodes
NAME                    STATUS   ROLES           AGE   VERSION
tfg-dev-control-plane   Ready    control-plane   14m   v1.29.2
tfg-dev-worker          Ready    <none>          13m   v1.29.2
tfg-dev-worker2         Ready    <none>          13m   v1.29.2

$> kind get clusters
tfg-dev

$> k get all
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   17m

$> k get ns
NAME                 STATUS   AGE
default              Active   29m
kube-node-lease      Active   29m
kube-public          Active   29m
kube-system          Active   29m
local-path-storage   Active   29m
```

Codi 2.2: Verificacions clúster desenvolupament

2.5 Repositoris de codi

El codi de tot el projecte està allotjat en un compte personal gratuït de github.com. Aquest compte allotja dos repositoris:

- **https://github.com/dlopezlo/scds**: conté les fonts de tot el projecte.
- **https://github.com/dlopezlo/fluxcd-infra**: conté les configuracions de *fluxcd* per a gestionar els clústers de Kubernetes de tots els entorns.

Creem la següent estructura de directoris en el repositori de l'aplicació:

- **/:** Conté el Dockerfile i el Makefile per a automatitzar algunes tasques locals.
- **/src:** Codi de l'aplicació i els mòduls de Go.
- **/.github:** Conté les definicions de les accions de github.
- **/devops/:** Conté les diferents configuracions d'IaC.
  - **/infra:** Fitxers de Terraform per a desplegar el repositori a DockerHub i el clúster de k8s a DigitalOcean.
  - **/k8s/:**
    - **/helm:** Definició del chart de Helm i fitxers de valors per entorn.
    - **/manifests:** Manifests de Kubernetes per a dev i prod.
  - **/kind:** Configuració per a crear el clúster de KinD.

Veiem ara l'estructura de directoris de *FluxCD*. Tenim una carpeta arrel (*clusters*) que conté una directori per cada clúster (dev i prod). Dintre de cada carpeta de clúster hi ha l'estructura creada per *Fluxcd* per a gestionar-los:

- **/flux-system:** Conté els manifests que defineixen el funcionament.
- **scds-source.yaml:** Defineix el repositori de l'aplicació com a objecte a monitorar per canvis.
- **scds-helmrelease.yaml:** Fa referència al chart de Helm que volem desplegar, on es troba i quins paràmetres usarem.

2.5.1 Instal·lació de les eines de gestió del repositori

Per a gestionar els repositoris usarem el propi client de consola *gh*. La instal·lació d'aquest està documentada a [10] i en el nostre cas hem usat el gestor de paquets *brew* disponible per al sistema operatiu *macos*.

```
$ > brew install gh
```

Codi 2.3: Instal·lació de gh cli

Usem el client per autenticar la sessió en el nostre compte de Github: 2.3

```
$ > gh auth login
```

Codi 2.4: Login a github

Figura 2.3: Autenticació github-client

Ja podem crear el repositori de codi des de la consola 2.4 (el repositori de FluxCD es crea automàticament durant la comanda de bootstrap):

```
$ > gh repo create scds --private -d "SSH Config Deployment Server. Projecte de TFG"
```

Codi 2.5: Creació del repositori amb gh

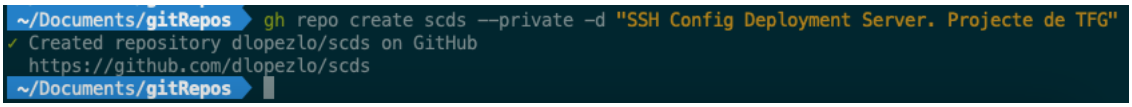

Figura 2.4: Creació del repositori de codi

En aquest moment ja podem clonar i començar a treballar amb el repositori del projecte: https://github.com/dlopezlo/scds.

2.6 Infraestructura com a Codi

La infraestructura com a Codi (IaC) és una tecnologia que ens permet definir i gestionar la nostra infraestructura en fitxers de text en contraposició a fer-ho de manera manual, (clic-operations). Aquesta aproximació te grans avantatges:

- Pot ser declarativa (quin resultat volem) o imperativa (quines accions fem per arribar al resultat desitjat).
- Permet guardar la infraestructura amb control de versions i aprofitar-se de les pull request i la revisió de codi.
- La infraestructura queda automàticament documentada (a diferència de les accions manuals).
- És fàcilment repetible doncs el codi defineix l'estructura i com es crea la infraestructura, ajuda en la recuperació de desastres.
- És reaprofitable, és a dir, podem usar la mateixa definició en diferents entorns usant variables per a cadascun.

2.6.1 Repositori d'imatges: DockerHub

Per tal de poder publicar la imatge del contenidor creada amb *Github Actions* al repositori de *hub.docker.com* hem d'obrir-hi un compte. Els comptes gratuïts permeten crear un únic repositori privat i diversos públics però amb una limitació 200 de descàrregues cada sis hores. Aquesta no serà un impediment durant el nostre projecte però en un entorn de producció hauríem d'adquirir un pla de pagament o usar altres serveis de repositori com el de GitHub o d'AWS, Google, etc.

Hem generat el nostre usant IaC i l'eina escollida per a els diferents recursos ha estat Terraform [15] que ens permet treballar amb una infinitat de proveïdors, entre ells Dockerhub i serveis cloud que proveeixen Kubernetes administrat. Al directori *"devops/infra"* del nostre repositori tenim els manifests de Terraform utilitzats per a crear el repositori.

Dintre del nostre directori de treball hi ha dos fitxers amb extensió *".tf"*:

- **providers.tf:** Defineix la configuració dels proveïdors amb els que volem treballar. Veure 4.5.
- **dockerhub.tf**: Conté la definició per a crear el repositori 4.4.

Hem d'inicialitzar *Terraform*, que crearà el fitxer d'estat i descarregarà els mòduls de proveïdors necessaris:

```
$> terraform init

 Initializing the backend...

 Initializing provider plugins...
- Finding barnabyshearer/dockerhub versions matching ">= 0.0.15"...
- Installing barnabyshearer/dockerhub v0.0.15...
- Installed barnabyshearer/dockerhub v0.0.15 (self-signed, key ID BBA58082BF377605)

Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/cli/plugins/signing.html

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.

```

Codi 2.6: terraform init

El proveïdor de DockerHub requereix que li passem per variable d'entorn l'usuari i la contrasenya del compte que hem usat per tal de poder crear els recursos. Els definim en la mateixa sessió de shell on executem Terraform:

```
export DOCKER_USERNAME="myusername"
export DOCKER_PASSWORD="mysuperpassword"

```

Codi 2.7: Credencials dockerhub

A continuació generem un pla d'acció basat en la configuració i apliquem els canvis:

```
$> terraform plan && terraform apply

Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the following symbols:
+ create

Terraform will perform the following actions:

# dockerhub_repository.registry will be created
+ resource "dockerhub_repository" "registry" {
    + description = "My corporate container image registry"
    + id          = (known after apply)
    + name        = "scds"
    + namespace   = "dlopezlo"
    + private     = true
}

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

dockerhub_repository.registry: Creating...
dockerhub_repository.registry: Creation complete after 1s [id=dlopezlo/scds]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Codi 2.8: Execució de terraform

Veiem a la sortida de la comanda que Terraform ens mostra amb el signe "+"els recursos nous que es crearan, amb els valors coneguts fins al moment i un resum del recursos afegits, modificats o eliminats abans de demanar-nos si volem aplicar els canvis. Terraform manté un fitxer d'estat amb els recursos creats i que fa servir com a font de la veritat. Un cop aplicats els recursos s'han creat i estan disponibles.

Validem que el repositori s'ha creat correctament a l'adreça:
https://hub.dockerhub.com/dlopezlo/scds com es mostra a la captura de pantalla 2.7:

Nota: Atès que el repositori ha estat creat com a privat i no és visible ni accessible sense estar correctament autenticat caldrà que generem un nou token de només lectura per a usar-lo durant els desplegaments de la nostra aplicació a Kubernetes.

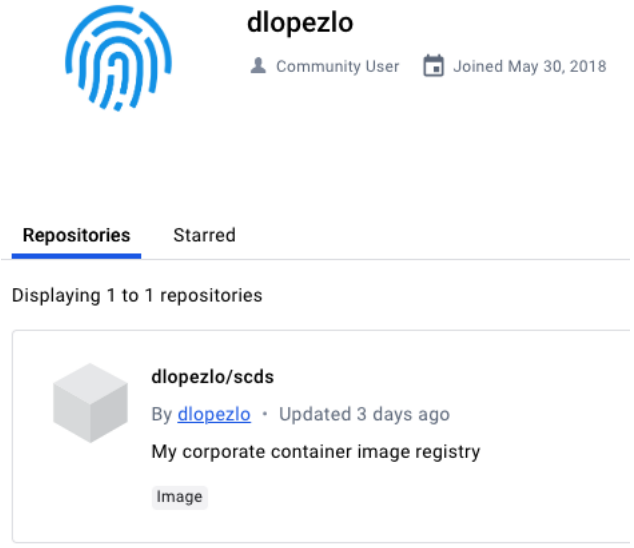

Figura 2.5: Repositori DockerHub creat amb Terraform

2.6.2 Clúster de Kubernetes

Elecció de proveïdor de Kubernetes

Per al desplegament de l'aplicació utilitzarem Kubernetes, KinD a l'entorn local de proves i desenvolupament. A producció contractarem un clústers a *DigitalOcean*, un proveïdor d'infraestructura al núvol. Escollir un servei de pagament per a desplegar l'aplicació final té la finalitat d'assimilar el projecte el màxim possible a un entorn corporatiu real, doncs estan disponibles i accessibles per a qualsevol interessat en accedir-hi (consultor, tribunal, etc) i podem usar les apis i serveis que proveeixen, com els balancejadors de càrrega, els certificats i els *providers* de Terraform per a usar-los programàticament.

Entre les diferents opcions que teníem per a desplegar el clúster (aws, azure, google cloud, DigitalOcean, ...) ens hem decidit finalment pel darrer. DigitalOcean proporciona les funcionalitats necessàries que usarem aquí:

- Ofereix una Api per a Terraform.
- Clúster administrat.
- Integració de Kubernetes amb els *Load Balancers* externs (i d'altres funcionalitats que no usem).

Quant al cost, el clúster de Kubernetes només es factura pel temps d'ús de cada node dedicat a allotjar *pods*, la capa de controladors és gratuïta. A més a més, no hi ha límit de tipus de nodes a escollir, és a dir, no estem obligats a seleccionar nodes amb unes característiques i preu mínims. El nostre clúster estarà format per dos nodes de 2 VCPU, 2GB de memòria i 60GB de disc amb un cost de 36$ mensuals

més el balancejador de càrrega extern, 12$ mensuals. Comptem amb un crèdit promocional de 200$ durant un període de 60 dies que cobreix perfectament l'abast i el cost d'aquest projecte, per tant no incorrerem en cap despesa econòmica.

Desplegament de k8s

A continuació seguim amb la creació del clúster de Kubernetes. Hem afegit el nou proveïdor de Terraform al nostre fitxer de *providers.tf* i cal inicialitzar-lo de nou per tal que es descarreguin els *controladors* específics per a interactuar amb les *apis* de DigitalOcean.

```
    // required_providers
    ...
    digitalocean = {
      source = "digitalocean/digitalocean"
      version = "~> 2.0"
    }

```

Codi 2.9: Execució de terraform

Usem la documentació oficial del proveïdor [4] per a generar el codi de Terraform que desplegarà un clúster amb la darrera versió (1.29.1) en el centre de dades d'Amsterdam. DigitalOcean proporciona uns mòduls que abstrauen la complexitat del clúster proporcionant uns valors per defecte funcionals. Per tant, només cal que escollim la versió, la regió, el nom del clúster i els tipus i quantitat de nodes que volem per tal de desplegar el nou clúster. Com veiem a continuació la configuració final és tan simple com la següent:

```
provider "digitalocean" {}

resource "digitalocean_kubernetes_cluster" "tfg-prod" {
  name   = "tfg-pro"
  region = "ams3"
  # Grab the latest version slug from `doctl kubernetes options versions`
  version = "1.29.1-do.0"

  node_pool {
    name       = "tfgpro-pool"
    size       = "s-2vcpu-2gb"
    node_count = 2
  }
}
```

Codi 2.10: Terraform K8s Cluster a DigitalOcean

L'autenticació amb DigitalOcean es pot especificar com una variable d'entorn a:

```
export DIGITALOCEAN_TOKEN="dop_v1_c..."

```

Codi 2.11: Auth de DigitalOcean

Apliquem els manifests per a desplegar l'aplicació i en pocs minuts tindrem disponible el nou clúster 2.12. Si necessitem afegir noves funcionalitats o canviar els nodes modificarem la configuració i tornarem a aplicar-ho:

```
$> terraform plan -out k8s-digitalocean.plan
dockerhub_repository.registry: Refreshing state... [id=dlopezlo/scds]

Terraform used the selected providers to generate the following execution plan. Resource actions
 are indicated with the following symbols:
+ create

Terraform will perform the following actions:

# digitalocean_kubernetes_cluster.tfg-pro will be created
+ resource "digitalocean_kubernetes_cluster" "tfg-pro" {
    + cluster_subnet                     = (known after apply)
    + created_at                         = (known after apply)
    + destroy_all_associated_resources = false
    + endpoint                           = (known after apply)
    + ha                                 = false
    + id                                 = (known after apply)
    + ipv4_address                       = (known after apply)
    + kube_config                        = (sensitive value)
    + name                               = "tfg-pro"
    + region                             = "ams3"
    + registry_integration               = false
    + service_subnet                     = (known after apply)
    + status                             = (known after apply)
    + surge_upgrade                      = true
    + updated_at                         = (known after apply)
    + urn                                = (known after apply)
    + version                            = "1.29.1-do.0"
    + vpc_uuid                           = (known after apply)

    + node_pool {
        + actual_node_count = (known after apply)
        + auto_scale        = false
        + id                = (known after apply)
        + name              = "tfgpro-pool"
        + node_count        = 2
        + nodes             = (known after apply)
        + size              = "s-2vcpu-2gb"
      }
}

Plan: 1 to add, 0 to change, 0 to destroy.
\
Saved the plan to: k8s-digitalocean.plan

To perform exactly these actions, run the following command to apply:
terraform apply "k8s-digitalocean.plan"
```

Codi 2.12: Plan d'execució de terraform

```
$> terraform apply "k8s-digitalocean.plan"
digitalocean_kubernetes_cluster.tfg-pro: Creating...
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [10s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [20s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [30s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [40s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [50s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [1m0s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [1m10s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [1m20s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [1m30s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [1m40s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [1m50s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [2m0s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [2m10s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [2m20s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [2m30s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [2m40s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [2m50s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [3m0s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [3m10s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [3m20s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [3m30s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [3m40s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [3m50s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [4m0s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Still creating... [4m10s elapsed]
digitalocean_kubernetes_cluster.tfg-pro: Creation complete after 4m13s [id=5570dce6-5944-4
 cba-a6fb-c46662cb1de7]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

```

Codi 2.13: Creació del cluster de k8s a DigialOcean

Descarreguem el fitxer de configuració per accedir al nostre nou clúster, podem provar-lo de la següent manera:

```
$> k --kubeconfig=tfg-pro-kubeconfig.yaml get nodes

NAME              STATUS   ROLES    AGE   VERSION
tfgpro-pool-jz2f0   Ready    <none>   8h    v1.29.1
tfgpro-pool-jz2f1   Ready    <none>   8h    v1.29.1

```

Codi 2.14: Primer accés al cluster de producció

A partir d'aquest moment integrarem aquesta configuració en el fitxer ~/.kube/config i podrem interactuar amb els diferents clústers configurats canviant de context:

```
$ > k config get-contexts

CURRENT   NAME              CLUSTER           AUTHINFO                  NAMESPACE
*         do-ams3-tfg-pro   do-ams3-tfg-pro   do-ams3-tfg-pro-admin
kind-tfg-dev                kind-tfg-dev      kind-tfg-dev

```

Codi 2.15: Clústers configurats

Configuració prèvia de k8s

Per tal de desplegar la nostra aplicació en aquest clúster necessitarem fer unes configuracions prèvies:

- Instal·larem el controlador d'Ingress de tipus Nginx que desplegarà automàticament un balancejador de càrrega extern que ens servirà per a poder publicar la nostra aplicació a internet.
- Instal·larem un gestor de certificats *cert-manager* per a generar i configurar certificats TLS per a la nostra aplicació.
- Haurem de crear un *secret* per tal que Kubernetes tingui accés al nostre repositori privat de *Dockerhub* per a descarregar la imatge de l'aplicació.
- Crearem el namespace de l'aplicació **tfg-prod** per allotjar el secret de *Dockerhub* i els recursos de Kubernetes de l'aplicació.

Comencem per Instal·lar l'Ingress [20]:

```
$> kubectl apply -f https :// raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1
.10.1/deploy/static/provider/do/deploy.yaml

namespace/ingress-nginx created
serviceaccount/ingress-nginx created
serviceaccount/ingress-nginx-admission created
role.rbac.authorization.k8s.io/ingress-nginx created
role.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrole.rbac.authorization.k8s.io/ingress-nginx created
clusterrole.rbac.authorization.k8s.io/ingress-nginx-admission created
rolebinding.rbac.authorization.k8s.io/ingress-nginx created
rolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
configmap/ingress-nginx-controller created
service/ingress-nginx-controller created
service/ingress-nginx-controller-admission created
deployment.apps/ingress-nginx-controller created
job.batch/ingress-nginx-admission-create created
job.batch/ingress-nginx-admission-patch created
ingressclass.networking.k8s.io/nginx created
validatingwebhookconfiguration.admissionregistration.k8s.io/ingress-nginx-admission created

```

Codi 2.16: Instal·lació de l'Ingress

Això ens crearà el següents recursos:

```
$> k get all -n ingress-nginx

NAME                                        READY   STATUS      RESTARTS   AGE
pod/ingress-nginx-admission-create-2grlj    0/1     Completed   0          8h
pod/ingress-nginx-admission-patch-n94vp     0/1     Completed   0          8h
pod/ingress-nginx-controller-57b7568757-sqs7c  1/1   Running     0          8h

NAME                                         TYPE           CLUSTER-IP     EXTERNAL-IP
 PORT(S)                      AGE
service/ingress-nginx-controller             LoadBalancer   10.245.240.19   scds.tydnet.org
 80:30448/TCP,443:31570/TCP   8h
service/ingress-nginx-controller-admission   ClusterIP      10.245.208.42   <none>
 443/TCP                      8h

NAME                                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/ingress-nginx-controller   1/1     1            1           8h

NAME                                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/ingress-nginx-controller-57b7568757   1         1         1       8h

NAME                                       COMPLETIONS   DURATION   AGE
job.batch/ingress-nginx-admission-create   1/1           5s         8h
job.batch/ingress-nginx-admission-patch    1/1           5s         8h

```

Codi 2.17: Recursos de l'Ingress

Veiem que tenim un nou Load Balancer amb una adreça ip pública que es mostra com un nom de *host*. **Important:** La integració de DigitalOcean té una particularitat derivada de una limitació de Kubernetes la qual no permet que els Pod puguin comunicar-se entre ells usant la ip pública del balancejador, això serà important durant el desplegament dels certificats TLS. Per tal de solucionar-ho ha estat necessari afegir una anotació en la definició de l'ingress per tal d'usar un *hostname* definit a la nostra zona de dns *scds.tydnet.org*. Veure [3] per a una explicació detallada de la solució.

A continuació preparem els recursos del controlador de *cert-manager* que s'encarregarà de generar i aplicar certificats firmats per la CA de LetsEncypt per als nostres dominis:

```
$> kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.14.5/cert-manager.yaml

namespace/cert-manager created
customresourcedefinition.apiextensions.k8s.io/certificaterequests.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/certificates.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/challenges.acme.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/clusterissuers.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/issuers.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/orders.acme.cert-manager.io created
serviceaccount/cert-manager-cainjector created
serviceaccount/cert-manager created
serviceaccount/cert-manager-webhook created
clusterrole.rbac.authorization.k8s.io/cert-manager-cainjector created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-issuers created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-clusterissuers created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-certificates created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-orders created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-challenges created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-ingress-shim created
clusterrole.rbac.authorization.k8s.io/cert-manager-cluster-view created
clusterrole.rbac.authorization.k8s.io/cert-manager-view created
clusterrole.rbac.authorization.k8s.io/cert-manager-edit created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-approve:cert-manager-io created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-certificatesigningrequests created
clusterrole.rbac.authorization.k8s.io/cert-manager-webhook:subjectaccessreviews created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-cainjector created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-issuers created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-clusterissuers created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-certificates created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-orders created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-challenges created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-ingress-shim created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-approve:cert-manager-io created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-certificatesigningrequests created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-webhook:subjectaccessreviews created
role.rbac.authorization.k8s.io/cert-manager-cainjector:leaderelection created
role.rbac.authorization.k8s.io/cert-manager:leaderelection created
role.rbac.authorization.k8s.io/cert-manager-webhook:dynamic-serving created
rolebinding.rbac.authorization.k8s.io/cert-manager-cainjector:leaderelection created
rolebinding.rbac.authorization.k8s.io/cert-manager:leaderelection created
rolebinding.rbac.authorization.k8s.io/cert-manager-webhook:dynamic-serving created
service/cert-manager created
service/cert-manager-webhook created
deployment.apps/cert-manager-cainjector created
deployment.apps/cert-manager created
deployment.apps/cert-manager-webhook created
mutatingwebhookconfiguration.admissionregistration.k8s.io/cert-manager-webhook created
validatingwebhookconfiguration.admissionregistration.k8s.io/cert-manager-webhook created

```

Codi 2.18: Instal·lació de Cert-Manager

Creem el *namespace* i el secret amb el *personal access token* del repositori de Docker:

```
$> source .secrets
$> k create ns tfg-prod
$> kubectl create secret docker-registry docker-cred \
  --docker-username=dlopezlo --docker-password=${DOCKER-PAT} -n tfg-prod

namespace/tfg-prod created
secret/docker-cred created
```

Codi 2.19: Namespace i secret

Ja tenim l'entorn preparat per a desplegar aplicacions que podran ser accessibles des d'internet i a les que podrem configurar un certificat TLS vàlid.

2.7 Desenvolupament de l'aplicació

2.8 Introducció

L'aplicació és un sistema de plantilles fet en Go i publicat en forma de servei web per facilitar tant l'accés com el seu us. S'ha dissenyat de manera que durant la fase de producció sigui simple de mantenir per usuaris que només hauran de modificar el fitxer de dades en format Json i el procés de ci/cd s'encarregui de tota la publicació en entorns de cloud.

2.8.1 Inicialització de l'aplicació

Les aplicacions en Go usen *modules* tant per a definir-se, totes són mòduls, com per a gestionar les seves dependències. Abans de començar codificar l'aplicació cal inicialitzar els mòduls. Això crearà uns fitxers go.mod i go.sum on s'especifiquen quines dependències i versions usa la nostra aplicació:

```
$ > go mod init github.com/dlopezlo/scds

go: creating new go.mod: module github.com/dlopezlo/scds
go: to add module requirements and sums:
go mod tidy
```

Codi 2.20: Inicialització del codi

Instal·lem els mòduls del framework web *Chi* des de Github:

```
$ > go get github.com/go-chi/chi/v5

go: downloading github.com/go-chi/chi/v5 v5.0.12
go: added github.com/go-chi/chi/v5 v5.0.12
```

Codi 2.21: Go Chi

2.8.2 Components i llibreries

- **Go:** Llenguatge base amb el que està codificada l'aplicació.
- **Chi:** Framework web usat per a implementar el servei web. Tot i que les funcionalitats bàsiques es poden cobrir perfectament amb els llibreries base de go presents en la seva darrera versió 1.22.x hem decidit usar-la per motius d'aprenentatge i la simplicitat que aporta. [18]
- **testify:** Biblioteca per a fer tests. [21]
- **text/template:** Ús de plantilles en mode text. [12]
- **encoding/json:** Ús del format Json en el que està codificat el fitxer de dades de connexions. [11]
- **utils/GetUsername:** Funció pròpia per validar l'entrada de dades de l'aplicació.

Un cop arrenca l'aplicació comença a escoltar peticions en el port 3000 del contenidor gràcies al framework *Chi*. Té dos punts d'entrada:

- **https://scds.tydnet.org/:** Retorna una frase de benvinguda.

```
$> curl localhost:3000
    Welcome to my TFG
```

Codi 2.22: Welcome page

- **https://scds.tydnet.org/sshconfig/${username}** Retorna la configuració de ssh parametritzada per l'usuari.

```
    $> curl localhost:3000/sshconfig/dani
    ### SCDS Generated config ###
    Host east-bastion bastion1
     Hostname east-bastion.tydnet.org
     Port 30022
     ...
    Host *
     ForwardAgent yes
     ForwardX11Trusted yes
     ServerAliveInterval 30
     ServerAliveCountMax 5
     HostKeyAlgorithms +ssh-rsa
     PubkeyAcceptedKeyTypes +ssh-rsa
     UseKeyChain yes
     AddkeysToAgent yes
     User dani
     Port 22
```

Codi 2.23: Generació de la configuració

2.8.3 Sistema de plantilles

Les dades que defineixen com s'han de crear els hosts del fitxer de configuració de ssh són com aquest:

```
{ "hosts": [
  {
    "host": [ "east-bastion", "bastion1" ],
    "hostname": "east-bastion.tydnet.org",
    "port": 30022
  },
  {
    "host": [ "leia", "leia.tydnet.org" ],
    "hostname": "10.77.1.10",
    "proxyjump": "east-bastion",
    "proxyCommand": "bash -c \"fwknop -w /usr/local/bin/wget -R -n steam; sleep 3; nc %h %p\""
  }]}
```

Codi 2.24: Json Data

Aquest fitxer Json es llegeix i es guarda en una llista estructures *Element* en el fitxer *"server/server.go"*:

```
type Element struct {
  Host          []string  `json:"host"`
  Hostname      string    `json:"hostname"`
  Port          int       `json:"port,omitempty"`
  ProxyCommand string   `json:"proxyCommand,omitempty"`
  ProxyJump     string    `json:"proxyjump,omitempty"`
  User          string    `json:"user,omitempty"`
}

var data struct {
  Hosts     []Element `json:"hosts"`
  Username string
}
```

Codi 2.25: Estructures de dades

La informació anterior juntament amb el nom d'usuari proporcionat en la crida http s'utilitza per a omplir la següent plantilla:

```
{{ range .Hosts }}
Host {{ range .Host }} {{ . }} {{ end }}
 Hostname {{ .Hostname }}
{{ if .Port }} Port {{ .Port }}{{ end }}
{{- if .User }} User {{ .User}}{{ end }}
{{- if .ProxyJump }} ProxyJump {{ .ProxyJump}}{{ end }}
{{- if .ProxyCommand }} ProxyCommand {{ .ProxyCommand}}{{ end }}
{{ end }}
#### DO NOT ADD ANY ENTRIES BELOW THIS LINE! ####
#### Values below will be added to all above Host entries unless specifically overridden ####
#### Customized/Additional SSH Host entries should go in personal_config file ####

Host *
 ForwardAgent yes
 ForwardX11Trusted yes
 ServerAliveInterval 30
 ServerAliveCountMax 5
 HostKeyAlgorithms +ssh-rsa
 PubkeyAcceptedKeyTypes +ssh-rsa
 UseKeyChain yes
 AddkeysToAgent yes
 User {{ .Username }}
 Port 22

```

Codi 2.26: Template

que és servida per l'aplicació.

2.8.4 Tests

S'han implementat tests per a validar el funcionament de l'aplicació. Concretament, d'una banda es valida que el servei web funciona fent una consulta al *Handler* que atén les peticions a l'arrel de l'aplicació i que retorna un simple text de benvinguda. De l'altra, es valida la funcionalitat de la funció *GetUsername* del paquet propi útils. Els tests són molt importants, no només per a validar la funcionalitat durant el desenvolupament sinó que també per a documentar-la de manera exhaustiva. Mostrem una part dels tests:

```
func TestGetUsernameValid(t *testing.T) {
  testCases := []struct {
    username string
  }{
    {"dlopez"},
    {"msolsona"},
    {"jroque"},
    {"cris.b"},
    {"xavi.gonzalez"},
    {"user77"},
    {"d-lopez"},
  }

  for _, tc := range testCases {
    t.Run("Valid username "+tc.username, func(t *testing.T) {
      result, err := GetUsername(tc.username)
      if result != tc.username {
        t.Errorf("Expected username %s. Got %s\n", tc.username, result)
      }
      if err != nil {
        t.Error("Valid usernames should not produce errors")
      }
    })
  }
}
```

Codi 2.27: Testing GetUserName

També s'han codificat tests per als casos d'ús en que el nom d'usuari és invàlid segons la majoria de sistemes operatius. Això ens assegura que no crearem una configuració de ssh amb un nom d'usuari invàlid i que l'aplicació només inputs correctes i segurs.

Veiem una execució dels tests mitjançant la crida corresponent al Makefile:

```
$ > make test

cd src && go test ./... -v -cover
github.com/dlopezlo/scds                    coverage: 0.0% of statements
=== RUN   TestWelcomePage
2024/05/20 09:35:43 "GET http:// HTTP/1.1" from - 200 17B in 31.302ms
--- PASS: TestWelcomePage (0.00s)
PASS
coverage: 26.5% of statements
ok      github.com/dlopezlo/scds/server (cached)        coverage: 26.5% of statements
=== RUN   TestGetUsernameValid
=== RUN   TestGetUsernameValid/Valid_username_dlopez
=== RUN   TestGetUsernameValid/Valid_username_msolsona
=== RUN   TestGetUsernameValid/Valid_username_jroque
=== RUN   TestGetUsernameValid/Valid_username_cris.b
=== RUN   TestGetUsernameValid/Valid_username_xavi.gonzalez
=== RUN   TestGetUsernameValid/Valid_username_user77
=== RUN   TestGetUsernameValid/Valid_username_d-lopez
--- PASS: TestGetUsernameValid (0.00s)
--- PASS: TestGetUsernameValid/Valid_username_dlopez (0.00s)
--- PASS: TestGetUsernameValid/Valid_username_msolsona (0.00s)
--- PASS: TestGetUsernameValid/Valid_username_jroque (0.00s)
--- PASS: TestGetUsernameValid/Valid_username_cris.b (0.00s)
--- PASS: TestGetUsernameValid/Valid_username_xavi.gonzalez (0.00s)
--- PASS: TestGetUsernameValid/Valid_username_user77 (0.00s)
--- PASS: TestGetUsernameValid/Valid_username_d-lopez (0.00s)
=== RUN   TestGetUsernameInvalid
=== RUN   TestGetUsernameInvalid/Invalid_username_0startswithdigit
2024/05/20 09:35:43 Invalid username <nil>
=== RUN   TestGetUsernameInvalid/Invalid_username_invalidChars.
2024/05/20 09:35:43 Invalid username <nil>
=== RUN   TestGetUsernameInvalid/Invalid_username_a
2024/05/20 09:35:43 Invalid username <nil>
=== RUN   TestGetUsernameInvalid/
 Invalid_username_toolongusernametoolongusernametoolongusername
2024/05/20 09:35:43 Invalid username <nil>
=== RUN   TestGetUsernameInvalid/Invalid_username__startswithunderscore
2024/05/20 09:35:43 Invalid username <nil>
=== RUN   TestGetUsernameInvalid/Invalid_username_-startswithhyphen
2024/05/20 09:35:43 Invalid username <nil>
--- PASS: TestGetUsernameInvalid (0.00s)
--- PASS: TestGetUsernameInvalid/Invalid_username_0startswithdigit (0.00s)
--- PASS: TestGetUsernameInvalid/Invalid_username_invalidChars.(0.00s)
--- PASS: TestGetUsernameInvalid/Invalid_username_a (0.00s)
--- PASS: TestGetUsernameInvalid/
 Invalid_username_toolongusernametoolongusernametoolongusername (0.00s)
--- PASS: TestGetUsernameInvalid/Invalid_username__startswithunderscore (0.00s)
--- PASS: TestGetUsernameInvalid/Invalid_username_-startswithhyphen (0.00s)
PASS
coverage: 77.8% of statements
ok      github.com/dlopezlo/scds/utils  (cached)        coverage: 77.8% of statements

```

Codi 2.28: Execució dels tests

Els testos s'executen en cada integració del codi a Github abans de crear el nou contenidor.

2.9 Desplegament de l'aplicació

El procediment que s'ha seguit per a desplegar l'aplicació a l'entorn de Kubernetes ha estat el següent, en ordre de desenvolupament:

- Es *dockeritza* l'aplicació: *Container First.* Des del primer moment de l'aplicació, quan encara era un *Hello World* es va crear la definició del contenidor.
- S'automatitza la creació i publicació del contenidor de la imatge.
- Es preparen els manifests per a instal·lar manualment l'aplicació a Kubernetes (namespace, deployment, service, ingress, cert-manager. secrets).
- Es paquetitzen els manifests en un *paquet de Helm* i s'utilitza per desplegar l'aplicació.
- S'integra FluxCD en el procés perquè instal·li el darrer chart de Helm.
- Github Actions s'encarrega d'actualitzar el chart quan hi ha noves versions del codi.

2.9.1 Dockerització de l'aplicació

En l'arrel del projecte podem trobar un fitxer Dockerfile 4.1 que conté la definició del contenidor sobre el que corre l'aplicació. Aquest es construeix en dues fases: en la primera es descarrega el codi, les dependències i es compilen per tal d'obtenir el binari final. En la segona es construeix un nou contenidor amb el binari i les dades obtinguts en l'anterior fase, és el contenidor de l'aplicació final. Amb aquest procediment s'obté una imatge final funcional amb només els elements essencials per tal de fer córrer l'aplicació. Alguns dels avantatges d'aquest procés són, entre d'altres, minimitzar el pes de la imatge i el temps de creació i publicació, reducció de l'exposició a vulnerabilitats de codi de tercers i facilitar el manteniment.

En l'arrel del repositori tenim un fitxer Makefile amb la comanda *"make"* per a crear el contenidor i executar-lo *"make run"* durant la fase de desenvolupament:

```
$> make

docker build -t dlopezlo/scds -f Dockerfile .
[+] Building 1.8s (17/17) FINISHED                                  docker:desktop-linux
=> [internal] load build definition from Dockerfile                                 0.1s
=> => transferring dockerfile: 505B                                                 0.0s
=> [internal] load metadata for docker.io/library/alpine:3.17                       1.6s
=> [internal] load metadata for docker.io/library/golang:1.22.1-alpine3.18          1.6s
=> [auth] library/golang:pull token for registry-1.docker.io                        0.0s
=> [auth] library/alpine:pull token for registry-1.docker.io                        0.0s
=> [internal] load .dockerignore                                                    0.0s
=> => transferring context: 2B                                                      0.0s
=> [builder 1/5] FROM docker.io/library/golang:1.22.1-alpine3.18@sha256:
ede158fb846dd8689c7 0.0s
=> [stage-1 1/4] FROM docker.io/library/alpine:3.17@sha256:53
cf9478b76f4c8fae126acbdfb79bed 0.0s
=> [internal] load build context                                                    0.0s
=> => transferring context: 199B                                                    0.0s
=> CACHED [stage-1 2/4] RUN apk update && apk --no-cache add ca-certificates        0.0s
=> CACHED [stage-1 3/4] WORKDIR /app                                                0.0s
=> CACHED [builder 2/5] RUN apk add --no-cache git upx                              0.0s
=> CACHED [builder 3/5] WORKDIR /app                                                0.0s
=> CACHED [builder 4/5] COPY [src/, ./]                                  0.0s
=> CACHED [builder 5/5] RUN ls -l utils/ && go mod download -x && go test ./... && go build -
ldflags="-s -w" -o scds -v . && upx scds 0.0s
=> CACHED [stage-1 4/4] --from=builder [/app/scds, /app/*json, /app/template.tpl    0.0s
=> exporting to image                                                               0.0s
=> => exporting layers                                                              0.0s
=> => writing image sha256:
eb7000f9e61dc894d5610ea042a8ed7e957603b8762a22b5445805eec00236f4 0.0s
=> => naming to docker.io/dlopezlo/scds                                             0.0s

```

Codi 2.29: Exemple de creació del contenidor en local

Arrenquem el contenidor de l'aplicació en local:

```
$ > make run
docker run -d --rm -h local-scds -p 3000:3000 dlopezlo/scds
5c8f60dc65bdb8d1190076a5fca55c292577778924853c7192b27b67c21d91de
$ >curl localhost:3000
Welcome to my TFG
$ >

```

Codi 2.30: Arrencar l'aplicació en local

2.9.2 Automatització de la publicació d'imatges

S'han configurat *workflows* a *GitHub Actions* per tal de generar una nova versió del contenidor de l'aplicació quan es produeix algun canvi a la branca principal *Main*. *Github* proveeix d'una sèrie de *workflows* que s'executen com a resposta a certs esdeveniments en el repositori, com poden ser *Pull requests*, *commits*, etc. Els *workflows* son receptes d'automatitzacions que actuen sobre el nostre codi i s'executen

en contenidors o màquines virtuals de GitHub o pròpies. GitHub ens proporciona i manté un repositori oficial d'accions predefinides per a la majoria de tasques que necessitem realitzar i en cas que tinguem una necessitat especial podem usar bé una realitzada per la comunitat o bé per nosaltres mateixos. En aquest projecte hem utilitzat accions predefinides i s'especifica en comentaris en el codi de l'acció el repositori d'origen de la tasca quan aquesta no és oficial.

Podem trobar les accions en el repositori, dintre del directory *"github/workflows"*, concretament l'acció que s'executa està configurada al fitxer *"docker-image.yml"* 4.3. Aquest *workflow* realitza les següents accions:

- S'activa quan hi ha un *commit* o una *pull request* a la branca principal i s'executa en una instància Ubuntu server.
- Descarrega el codi del repositori.
- Instal·la l'eina *cosign* i fa login en el nostre registre de hub.docker.com.
- Etiqueta la imatge del contenidor, la signa i el publica al registre de Docker.
- Per etiquetar la imatge s'utilitza el darrer hash del commit i la paraula "*latest*".

En realitzar qualsevol canvi al codi el *workflow* de l'acció s'activa i genera la nova imatge:

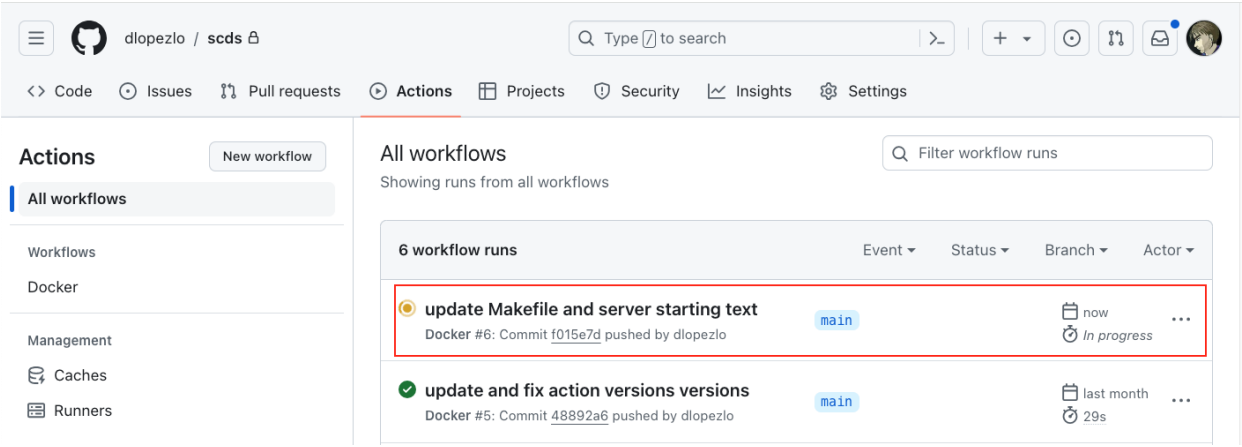

Figura 2.6: Execució del workflow

Un cop l'execució finalitza satisfactòriament podem validar que al repositori d'imatges de contenidors tenim la nova versió actualitzada 2.7:

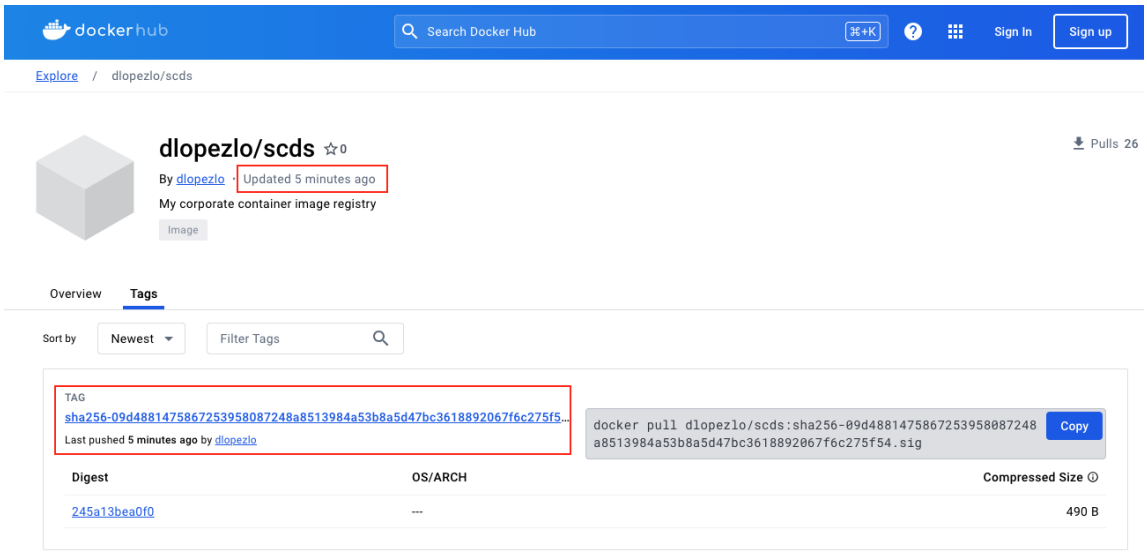

Figura 2.7: Imatge docker publicada

2.9.3 Manifests de k8s

El primer pas per a desplegar l'aplicació a Kubernetes va ser la definició dels manifests que crearien els recursos i en que usarem després per a crear el chart de Helm. **Nota:** No s'inclouen aquí els manifestos externs que hem instal·lat en durant el desplegament de *K8s*.

- **deployment.yaml:** Aquí tenim la definició de com es despleguen els pods (imatge del contenidor, nombre de rèpliques, etiquetes, etc.). 4.11
- **namespace.yaml:** Definició del namespace on s'allotgen els recursos de l'app. 4.12
- **services.yaml:** Es defineix aquí com es publica l'app (ports, hostname, paths, tls, etc.) 4.13
- **letsencrypt.yaml:** Aquí es configura el component que genera el certificat per a la nostra aplicació. 4.14

Nota: No està en el repositori el manifest per a crear secret d'accés a *dockerhub*.

2.9.4 Helm

Helm és un gestor de paquets per a Kubernetes. A macos el podem Instal·lar en local usant la comanda 'brew install helm'. Els paquets de Helm s'anomenen *Charts* i com altres formats de paquets *rpm* i *deb* tenen una estructura de directoris on s'emmagatzemen els fitxers de configuració i dades per a dur a terme la instal·lació de l'app.

Per a crear un nou paquet executarem la següent comanda:

```
$ >  helm create scds
Creating scds
```

Codi 2.31: Creació d'un chart

Això ens ha creat una estructura de carpetes i fitxers en el directori actual. Els més importants són:

- **Chart.yaml:** Conté les metadades del paquet. Veure 2.34
- **templates:** Aqui dintre definirem els les plantilles dels nostres manifests de Kubernetes. Veure 2.32 per a un exemple.
- **values.yaml:** Aqui podrem definir els valors que pendrà el chart en un deploy concret. 2.33

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Values.appName }}
  namespace: {{ .Values.namespace }}
  labels:
    app: {{ .Values.appName }}
spec:
  replicas: {{ .Values.replicas }}
  selector:
    matchLabels:
      app: {{ .Values.appName }}
  template:
    metadata:
      labels:
        app: {{ .Values.appName }}
        annotations:
          timestamp: {{ now }}
    spec:
      containers:
    - name: ssh-config-server
      image: "{{ .Values.image.name }}:{{ .Values.image.tag }}"
      imagePullPolicy: Always
      ports:
        - containerPort: {{ .Values.service.port }}
    imagePullSecrets:
    - name: {{ .Values.secret.name }}
```

Codi 2.32: Deployment Template

Les variables definides a les plantilles són reemplaçades pels valors dels fitxers *values.yaml*:

```
domain: scds.tydnet.org
namespace: tfg-prod
appName: scds
environment: prod
secret:
  name: docker-cred

service:
  port: 3000
  targetPort: 3000
  type: ClusterIP

image:
  name: dlopezlo/scds
  tag: latest

replicas: 2
```

Codi 2.33: Chart Values.yaml

```
apiVersion: v2
name: scds
description: SSH Configuration Deployment Server
type: application

version: 0.2.10
appVersion: "0.1.1"
```

Codi 2.34: Chart.yaml

Un cop creat el nostre chart el podem provar amb:

```
    $ > helm install scds scds/ --values scds/values-prod.yaml -n tfg-prod

    NAME: scds
    LAST DEPLOYED: Mon May 20 13:25:39 2024
    NAMESPACE: tfg-prod
    STATUS: deployed
    REVISION: 1
    TEST SUITE: None

```

Codi 2.35: Instal·lació usant Helm

I ja tindríem l'aplicació desplegada al clúster. Helm ens permet de manera fàcil actualitzar, eliminar i tornar a versions anteriors l'aplicació desplegada i mantenir un bon control de versions de l'app actual:

```
    $ > helm ls --all-namespaces --all
    NAME  NAMESPACE REVISION  UPDATED                                 STATUS    CHART      APP
     VERSION
    scds  tfg-prod  1         2024-05-20 13:25:39.819961 +0200 CEST deployed  scds-0.2.10
     0.1.1

```

Codi 2.36: Helm ls

Per exemple, podrem veure un històric de les darreres instal·lacions i tornar enrere en cas de ser necessari:

```
$> helm history scds -n tfg-prod

REVISION  UPDATED                   STATUS      CHART                    APP VERSION
    DESCRIPTION
5         Mon May 20 20:33:17 2024  superseded  scds-0.3.0+d2138d327613 0.1.1      Upgrade
    complete
6         Mon May 20 20:35:19 2024  superseded  scds-0.3.0+2a85b06be3a8 0.1.1      Upgrade
    complete
7         Mon May 20 20:38:20 2024  superseded  scds-0.3.0+8877a1e77164 0.1.1      Upgrade
    complete
8         Tue May 21 07:39:53 2024  superseded  scds-0.3.0+e60b2a2f40c2 0.1.1      Upgrade
    complete
9         Tue May 21 08:48:45 2024  deployed    scds-0.3.0+65a047ef5104 0.1.1      Upgrade
    complete
```

Codi 2.37: Helm History

2.9.5 FluxCD (GitOps)

Introducció a GitOps

FluxCD [7] és una eina de GitOps[17] que ens permet usar un repositori git com a font per a dur a terme el desplegament del nostre codi. Comptarem amb manifests allotjats en el repositori que descriuran l'estat desitjat de la plataforma administrada (en el nostre cas els clústers de Kubernetes i l'aplicació que hem desenvolupat) i serviran per reconciliar l'estat actual amb el desitjat aplicant els canvis necessaris.

Instal·lació de FluxCD

Instal·lem FluxCD amb el gestor de paquets *brew*:

```
$> brew install fluxcd/tap/flux

```

Codi 2.38: FluxCD structura de directoris

El primer que hem de realitzar és el *bootstrap*: això desplega els controladors de fluxcd en el clúster i crear l'estructura de directoris en el repositori. El procés de bootstrap s'integra amb diferents proveïdors de git i permet crear el repositori si aquest no existeix:

```
$> flux bootstrap github \
--token-auth --owner=dlopezlo --repository=fluxcd-infra \
--branch=main --path=clusters/tfg-prod --personal
Please enter your GitHub personal access token (PAT):

- connecting to github.com
- cloning branch "main" from Git repository "https://github.com/dlopezlo/fluxcd-infra.git"
+ cloned repository
- generating component manifests
+ generated component manifests
+ component manifests are up to date
- installing components in "flux-system" namespace
+ installed components
+ reconciled components
- determining if source secret "flux-system/flux-system" exists
- generating source secret
- applying source secret "flux-system/flux-system"
+ reconciled source secret
- generating sync manifests
+ generated sync manifests
+ committed sync manifests to "main" ("88c1c00ed07c28fd60204fb9ebe92bf1f007f923")
- pushing sync manifests to "https://github.com/dlopezlo/fluxcd-infra.git"
- applying sync manifests
+ reconciled sync configuration
 waiting for GitRepository "flux-system/flux-system" to be reconciled
+ GitRepository reconciled successfully
 waiting for Kustomization "flux-system/flux-system" to be reconciled
+ Kustomization reconciled successfully
- confirming components are healthy
+ helm-controller: deployment ready
+ kustomize-controller: deployment ready
+ notification-controller: deployment ready
+ source-controller: deployment ready
+ all components are healthy

```

Codi 2.39: FluxCD Bootstrap cluster

La comanda de bootstrap ens haurà creat el repositori fluxcd-infra al nostre compte de Github i instal·lat els controladors al clúster. Usarem el mateix repositori per a gestionar els dos clústers que tenim configurats, per tant repetirem aquesta comanda per cadascun d'ells. Clonem el repositori localment per començar a treballar-hi. Un cop afegits tindrem una estructura de directoris com aquesta:

```
\fluxcd-infra
  \cluster
    \tfg-${entorn}
      \flux-system
        \got-components.yaml
        \gotk-sync.yaml
        \kustomization.yaml

```

Codi 2.40: FluxCD structura de directoris

Els manifests que hi ha dintre del directori flux-system són els encarregats de configurar el clúster, el fitxer *kustomization.yaml* 4.15 carrega els altres dos fitxers. Parem especial atenció al fitxer *gotk-synk.yaml* lst:flux-gotksync que defineix el directory *clusters/tfg-prod* com a font de configuracions. Així, definirem aquí els nostres manifests.

El fitxer *scds-source.yaml* configura la monitorització de la branca main del nostre repositori com a font de la veritat amb intervals d'1 minut.

```
---
apiVersion: source.toolkit.fluxcd.io/v1
kind: GitRepository
metadata:
  name: scds
  namespace: tfg-prod
spec:
   interval: 1m0s
   ref:
     branch: main
   url: https://github.com/dlopezlo/scds

```

Codi 2.41: FluxCD SCDS Git Repository

En el fitxer *scds-helmrelease.yaml* definim com s'instal·la la nostra aplicació en el clúster. En el nostre cas s'utilitza el chart de Helm que es pot trobar en la ruta *./devops/k8s/helm/scds* del repositori. Per simplicitat hem optat per incloure els valors la nostra configuració dintre del propi manifest però podríem haver estructurat els directoris per tal que tots els clústers agafin la definició d'un directori base i apliquin els valors de d'un fitxer *values.yaml* o un objecte *configmap* de Kubernetes.

Veiem el següent codi 2.42 i parem atenció a la propietat:

```
  .spec.chart.spec.reconcileStrategy: Revision
```

Aquesta part de la configuració és molt important per tal que el flux d'integració automatitzat basat en *commits* al codi font de l'aplicació apliqui correctament. El funcionament per defecte dels objectes *HelmRelease* depèn de la versió del chart, és a dir, FluxCD monitoritza la versió que hi ha definida al fitxer *Chart.yaml* i quan detecta una versió nova fa la reconciliació i desplega el nou chart. En el nostre cas no volem haver de modificar la versió manualment of mitjançant una *Github Action* sinó que volem que el procés sigui transparent i simple. Per aquesta raó hem definit l'estratègia com a *Revision*, que permet desencadenar el procés quan el repositori es modificat.

```yaml
---
apiVersion: helm.toolkit.fluxcd.io/v2
kind: HelmRelease
metadata:
  name: scds
  namespace: tfg-prod
spec:
  interval: 1m
  timeout: 2m
  releaseName: scds
  chart:
    spec:
      reconcileStrategy: Revision
      chart: ./devops/k8s/helm/scds
      sourceRef:
        kind: GitRepository
        name: scds
  install:
    createNamespace: true
  values:
    domain: scds.tydnet.org
    namespace: tfg-prod
    appName: scds
    environment: prod
    secret:
      name: docker-cred
    service:
      port: 3000
      targetPort: 3000
      type: ClusterIP
    image:
      name: dlopezlo/scds
      tag: latest
    replicas: 2
```

Codi 2.42: FluxCD SCDS Helm Rerelease

Finalment, el flux final ha acabat i en quan volem fer un canvi a la configuració seguim el següent procediment:

1. L'usuari clona el repositori *scds*.
2. Es crea una branca nova a partir de la principal.
3. Es realitzen els canvi desitjats, per exemple, el fitxer *ssh-data.json* i es pugen els canvis al repositori.
4. Es crea una nova pull request.
5. S'activen els workflows de la pull request.
6. Un *code owner* revisa i aprova la pull request.
7. S'integra el codi a la branca principal.

8. S'activa el workflow de creació i publicació de la imatge de Docker.
9. FluxCD detecta que hi ha una nova revisió del codi en el repositori i activa la reconciliació desplegant la nova versió.
10. Els canvis ja estan disponibles a l'aplicació publicada.

Capítol 3

Conclusions

3.1 Revisió de la planificació

La planificació del projecte s'ha complert de manera molt satisfactòria, s'ha executat tot el projecte dintre del període del tres primers lliuraments, tant la part tècnica com la redacció de la memòria.

El projecte va patir una petita desviació relacionada amb les tasques d'integració de components a l'entorn de desenvolupament. En concret, s'havia previst que al final del segon lliurament es desplegaria usant GitOps una aplicació web de tipus *Hello World* en el clúster local de *KinD*. Malauradament, van sorgir dos problemes:

- La gestió de certificats amb cert-manager a l'entorn intern no funcionava correctament. Tanmateix, no era una funcionalitat essencial en aquest moment del projecte i les mateixes configuracions funcionarien en l'entorn final de producció.
- Assimilar els conceptes i coneixements necessaris sobre *FluxCD* i integrar-los amb *Helm* va suposar més temps del previst. La incidència es va resoldre finalment desplegant l'aplicació usant *Helm* de manera manual. Tot i així, el projecte estava molt avançat, i durant el termini del tercer lliurament es va resoldre el problema i no va suposar una desviació en el lliurament final del projecte.

3.2 Assoliment dels objectius

A la planificació inicial ens vam proposar els quatre objectius següents, analitzem aquí el seu grau de consecució:

- Es desenvolupa una aplicació web que serveix la configuració de ssh actualitzada per a l'usuari que la sol·licita.

- El desplegament successiu de la solució es realitza automàticament mitjançant únicament interaccions amb el repositori de codi.
- La solució és reutilitzable en la resolució d'altres projectes.
- S'adquireix experiència pràctica demostrable en les àrees i tecnologies del projecte.

Tot i ser el primer objectiu que ens vam proposar, desenvolupar l'aplicació va ser de les darreres accions que es van realitzar. La seva funcionalitat estava molt definida i acotada, el desenvolupament no presentava la complexitat de les altres tasques. En finalitzar el tercer lliurament, juntament amb la memòria del projecte, tenim una aplicació publicada a internet que retorna la configuració ssh creada a partir de les dades del repositori. En provar-la comprovem que sintàcticament és correcta i podem connectar-nos a algun dels servidors definits. Aquest primer objectiu s'ha assolit satisfactòriament.

Un cop desenvolupada i desplegada l'aplicació en l'entorn de producció, gràcies a les configuracions realitzades al component de GitOps *FluxCD* i la seva integració amb el repositori de l'aplicació i el chart de *Helm* hem aconseguit que en actualitzar el codi de l'aplicació es llenci tot el procés de creació de la imatge del contenidor, es publiqui en el registre i finalment s'instal·li una nova versió de l'aplicació empaquetada amb *Helm*. Aquest segon objectiu, que és en realitat la part tècnica final del projecte, ha estat assolit molt satisfactòriament.

Tanmateix, aquesta part va patir un canvi d'implementació durant el desenvolupament. La primera aproximació de la solució era la creació d'una acció de github que havia dáctualitzar la versió del chart, requeriment per a que es llenci el procés d'actualització. Aquest mètode va presentar alguns problemes a l'hora de crear la funció que actualitzava el fitxer amb la versió i no resultava massa elegant haver d'incrementar constantment el dígit del*patch level*. Rellegint detingudament la documentació vam veure que era més adient canviar l'estratègia de reconciliació i usar la revisió del repositori.

El tercer objectiu també s'ha assolit correctament, doncs el projecte ha estat dissenyat per a ser una solució genèrica a un problema que es pot adaptar aplicacions. El projecte està cohesionat entre les diferents parts i presenta poc acoblament. Podem canviar l'aplicació per qualsevol altra, fins i tot més complexa, i seguir usant la mateixa arquitectura amb petites adaptacions a la configuració. En canvis més grans, com podria ser usar altres solucions basades en contenidors (ECS per exemple), es pot reaprofitar gran part del projecte i canviar els components específics que treballen amb Kubernetes per uns altres.

Finalment, un cop finalitzat el projecte i comprovat que funciona repassem el camí que hem realitzat. L'idea original era la d'usar el projecte com a excusa per a aprendre, practicar i en definitiva adquirir experiència en diferents tecnologies. En

algunes d'elles comptàvem amb un coneixement molt bàsic, com era l'àrea de Kubernetes, *Helm* , *FluxCD* o *Go*. En d'altres, un coneixement més extens ha ajudat a solucionar diversos problemes que han anat sorgint. En dissenyar la solució, hi havia moltes parts fosques relacionades amb el funcionament d'alguns dels components i com s'integrarien a la pràctica que aportaven un grau elevat d'incertesa quan a l'èxit del projecte tal i com es va pensar originalment. Realitzar el projecte de manera incremental, usant primer un entorn de desenvolupament, creant els manifestos, instal·lant els components manualment per tal d'eliminar la complexitat de l'automatització i validar el seu funcionament ha ajudat molt a esclarir dubtes i entendre millor la solució global i com s'interrelacionen els components. Trencar el problema en petites fites ha estat clau en la consecució del projecte i en l'aprenentatge de les noves tecnologies.

Hi ha hagut moments en que el projecte no funcionava com s'esperava degut a problemes relacionats amb particularitat de la implementació de Kubernetes a *DigitalOcean*, per exemple. Aquestes incidències ens han obligat a realitzar diversos anàlisis en profunditat, repassar la documentació i les configuracions per tal de detectar i solucionar el problema. Hem hagut de repetir la instal·lació diverses vegades per assegurar que podem documentar un procés que funciona. Un dels avantatges de la IaC és el de poder recrear la infraestructura de manera fàcil. Tot això, ha fet que puguem obtenir un coneixement més pròxim de les diferents parts per poder enfrontar-nos amb més confiança a futurs projectes que treballin amb els mateixos component. L'expertesa s'adquireix amb dedicació i molt de temps, aquí hem donat el primer pas d'aquest camí. Podem dir que hem assolit l'objectiu més important del projecte.

3.3 Vies futures de treball

Hem comentat en la introducció que aquest projecte és molt transversal i que és compost de la integració de tecnologies de diverses àrees de coneixement. Si hagués estat desenvolupat en un àmbit empresarial hauria realitzat per diversos equips funcionals especialistes en cada àmbit que es centrarien en la realització de les funcionalitats necessàries amb els estàndards més alts. Un projecte com aquest, realitzat per una sola persona, a temps parcial i en el marc de l'àmbit acadèmic necessàriament ha de tenir un abast limitat i deixar marge a futurs projectes d'ampliació. A continuació passem a enumerar algunes d'aquestes:

Creació i distribució de noves configuracions: L'aplicació es pot estendre per tal de servir altres tipus de configuracions com poden ser les d'accés a *aws*, la configuració de client de clústers de Kubernetes, entre altres. L'aplicació desenvolupada serveix com a base per a crear d'altres serveis separats o integrar-los en la pròpia aplicació. Caldria:

- Crear la plantilla per a la nova configuració i definir les configuracions possibles que es poden donar.

- Crear el fitxer amb les dades a afegir. Definir l'estructura i exemples del fitxer de configuració que serveixin com a base i documentació.
- Afegir un *Handler* pel nou punt d'entrada *https://mydomain/myNewService* i la lògica pròpia si és necessari per al cas d'ús.

Millora de les eines de CI: Com a part del flux d'actualització de *scds* una part molt important i necessària és la revisió de les pull requests dels usuaris, doncs s'ha de verificar que el codi que s'integra i que finalment executaran els usuaris. En la configuració de ssh podem incloure comandes com:

```
1 ProxyCommand bash -c "fwknop -R -n dudi; sleep 3; nc %h %p"
```

Codi 3.1: ProxyCommand

Un usuari maliciós podria introduir aquí comandes que executessin accions per robar dades, destructives, o per vulnerar la seguretat de l'usuari i l'empresa, per exemple. Depenem molt de la revisió manual del codi per part dels mantenidors del repositori per tal de validar que les configuracions que s'integren són confiables. Podem incloure entre les tasques que s'executen després de la creació de la PR validacions que analitzin les comandes incloses al ProxyCommand i avisin si es detecten accions potencialment malicioses o fora de les permeses. Aquestes ajudes facilitarien molt les revisions de codi i ajudarien a evitar problemes de seguretat.

Integració empresarial: Un dels pressupostos de la realització del projecte quant a seguretat era que l'aplicació en un entorn corporatiu estaria limitat al perímetre de l'empresa, bé sigui perquè es situa dintre de la pròpia xarxa privada, o bé perquè tot i estar accessible des d'internet s'han creat llistes de accés, regles de tallafocs i altres mecanismes que filtren la visibilitat del servei. Així, quant a la seguretat es podria treballar en dissenyar una autenticació que s'integrés amb les altres eines corporatives, per exemple, certificats personals emesos per la CA corporativa als usuaris.

Capítol 4

Annexes

Tot el codi font del projecte es trobar al repositori: https://github.com/dlopezlo/scds, a continuació s'exposen els codis fonts més rellevants per tal de facilitar la lectura d'aquest treball:

```
FROM golang:1.22.3-alpine3.18 AS builder
RUN apk add --no-cache git upx

WORKDIR /app

COPY ["src/",  "./"]
RUN ls -l  utils /  && go mod download -x && \
    go test  ./...  && \
    go build  -ldflags="-s -w" -o scds -v . && \
    upx scds

FROM alpine:3.19
LABEL Name=scds

RUN apk update && apk --no-cache add ca-certificates

WORKDIR /app
COPY --from=builder ["/app/scds", "/app/*json", "/app/template.tpl ",  "./"]

ENTRYPOINT ["./scds"]
```

Codi 4.1: Dockerfile

```
build:
  docker build  -t  dlopezlo/scds -f  Dockerfile  .
run:
  docker run -d --rm -h local-scds -p 3000:3000 dlopezlo/scds
plantuml:
  docker run -d --rm -p 18080:8080 plantuml/plantuml-server:jetty
test:
  cd src  && go test  ./...  -v -cover
```

Codi 4.2: Makefile

```
name: Docker
```

```
2
3  # This workflow uses actions that are not certified by GitHub.
4  # They are provided by a third-party and are governed by
5  # separate terms of service, privacy policy, and support
6  # documentation.
7  on:
8    push:
9      branches: [ "main" ]
10     paths: ['src/**']
11   workflow_dispatch:
12
13 env:
14   # Use docker.io for Docker Hub if empty
15   REGISTRY: docker.io
16   # github.repository as <account>/<repo>
17   IMAGE_NAME: ${{ github.repository }}
18
19
20 jobs:
21   build:
22
23     runs-on: ubuntu-latest
24     permissions:
25       contents: read
26       packages: write
27       # This is used to complete the identity challenge
28       # with sigstore/fulcio when running outside of PRs.
29       id-token: write
30
31     steps:
32       - name: Checkout repository
33         uses: actions/checkout@v4
34
35       # Install the cosign tool except on PR
36       # https://github.com/sigstore/cosign-installer
37       - name: Install cosign
38         if: github.event_name != 'pull_request'
39         uses: sigstore/cosign-installer@v3.3.0
40         with:
41           cosign-release: 'v2.2.2'
42
43       # Set up BuildKit Docker container builder to be able to build
44       # multi-platform images and export cache
45       # https://github.com/docker/setup-buildx-action
46       - name: Set up Docker Buildx
47         uses: docker/setup-buildx-action@v3
48
49       # Login against a Docker registry except on PR
50       # https://github.com/docker/login-action
51       - name: Log into registry ${{ env.REGISTRY }}
52         if: github.event_name != 'pull_request'
53         uses: docker/login-action@v3
54         with:
55           registry: ${{ env.REGISTRY }}
56           username: ${{ secrets.DOCKER_LOGIN }}
```

```
57              password: ${{ secrets.DOCKER_PAT }}
58
59          - name: Docker meta
60            id: meta
61            uses: docker/metadata-action@v5
62            with:
63            #  list   of  Docker images to use as base name for tags
64              images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}
65              # generate Docker tags based on the following events/attributes
66              tags: |
67                type=raw,value=latest,enable={{is_default_branch}}
68                type=sha
69
70          # Build  and push Docker image with Buildx (don't push on PR)
71          # https :// github.com/docker/build-push-action
72          - name: Build and push Docker image
73            id: build-and-push
74            uses: docker/build-push-action@v5
75            with:
76              context: .
77              push: ${{ github.event_name != 'pull_request' }}
78              tags: ${{ steps.meta.outputs.tags }}
79              labels: ${{ steps.meta.outputs.labels }}
80              cache-from: type=gha
81              cache-to: type=gha,mode=max
82
83          # Sign the  resulting  Docker image digest except on PRs.
84          # This  will  only  write  to  the public  Rekor transparency log when the Docker
85          # repository  is  public  to  avoid leaking data.    If  you would like  to  publish
86          # transparency data even for private  images, pass --force to cosign below.
87          # https :// github.com/sigstore/cosign
88          - name: Sign the published Docker image
89             if :  ${{ github.event_name != 'pull_request' }}
90            env:
91              # https :// docs.github.com/en/actions/security-guides/security-hardening-for-github-
        actions#using-an-intermediate-environment-variable
92              TAGS: ${{ steps.meta.outputs.tags }}
93              DIGEST: ${{ steps.build-and-push.outputs.digest }}
94            # This step uses the  identity  token to  provision  an ephemeral certificate
95            # against the  sigstore  community Fulcio instance.
96            run:  echo "${TAGS}" | xargs -I {}  cosign sign  --yes {}@${DIGEST}
```

Codi 4.3: Push to Docker GHA

```
1  # Create an image registry
2  resource "dockerhub_repository" "registry"  {
3    name         = "scds"
4    namespace    = "dlopezlo"
5    description  = "My corporate container image registry"
6    private      = true
7  }
```

Codi 4.4: Terraform dockerhub

```
terraform {
    required_version = ">= 0.13"

    required_providers {
        dockerhub = {
            source = "BarnabyShearer/dockerhub"
            version = ">= 0.0.15"
        }
        digitalocean = {
            source = "digitalocean/digitalocean"
            version = "~> 2.0"
        }
    }
}
```

Codi 4.5: Terraform provider

```
provider "digitalocean" {}

resource "digitalocean_kubernetes_cluster" "tfg-prod" {
  name   = "tfg-pro"
  region = "ams3"
  # Grab the latest version slug from `doctl kubernetes options versions`
  version = "1.29.1-do.0"

  node_pool {
    name       = "tfgpro-pool"
    size       = "s-2vcpu-2gb"
    node_count = 2
  }
}
```

Codi 4.6: Terraform K8s Cluster a DigitalOcean

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  kubeadmConfigPatches:
  - |
    kind: InitConfiguration
    nodeRegistration:
      kubeletExtraArgs:
        node-labels: "ingress-ready=true"
  extraPortMappings:
  - containerPort: 80
    hostPort: 80
    protocol: TCP
  - containerPort: 443
    hostPort: 443
    protocol: TCP
- role: worker
- role: worker
```

Codi 4.7: Config Kind multi clúster

```yaml
apiVersion: v1
kind: Namespace
metadata:
  labels:
    kubernetes.io/metadata.name: tfg-dev
  name: tfg-dev
spec:
  finalizers :
  - kubernetes
```

Codi 4.8: k8s manifest namespace

```yaml
kind: Service
apiVersion: v1
metadata:
  name: scds-service
spec:
  selector:
    app: scds
  ports:
  # Default port used by the image
  - port: 3000
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-ingress
  annotations:
    cert-manager.io/cluster-issuer: letsencrypt-prod
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    acme.cert-manager.io/http01-ingress-class: "nginx"
spec:
  rules:
  - http:
      paths:
      - pathType: Prefix
        path: /scds(/|$)(.*)
        backend:
          service:
            name: scds-service
            port:
              number: 3000
---
```

Codi 4.9: k8s manifest services

```yaml
---
apiVersion: v1
kind: Namespace
metadata:
  labels:
    kubernetes.io/metadata.name: tfg-dev
  name: tfg-dev
spec:
  finalizers :
  - kubernetes
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: scds-deployment
  labels:
    app: scds
spec:
  replicas: 2
  selector:
    matchLabels:
      app: scds
  template:
    metadata:
      labels:
        app: scds
    spec:
      containers:
      - name: ssh-config-server
        image: dlopezlo/scds:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 3000
---
```

Codi 4.10: k8s manifest deployment

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: scds
  labels:
    app: scds
  namespace: tfg-prod
spec:
  replicas: 2
  selector:
    matchLabels:
      app: scds
  template:
    metadata:
      labels:
        app: scds
    spec:
      containers:
      - name: ssh-config-server
        image: dlopezlo/scds:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 3000
      imagePullSecrets:
      - name: docker-cred
```

Codi 4.11: k8s manifest deployment

```yaml
apiVersion: v1
kind: Namespace
metadata:
  labels:
    kubernetes.io/metadata.name: tfg-prod
  name: tfg-prod
spec:
  finalizers :
  - kubernetes
```

Codi 4.12: k8s manifest namespace

```yaml
kind: Service
apiVersion: v1
metadata:
  name: scds-service
  namespace: tfg-prod
  labels:
    app: scds
spec:
  type: ClusterIP
  selector:
    app: scds
  ports:
  - port: 3000
    targetPort: 3000
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: scds-ingress
  namespace: tfg-prod
  labels:
    app: scds
  annotations:
    cert-manager.io/cluster-issuer: letsencrypt-prod
    acme.cert-manager.io/http01-edit-in-place: "true"
    service.beta.kubernetes.io/do-loadbalancer-hostname: "scds.tydnet.org"
      # nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
    acme.cert-manager.io/http01-ingress-class: "nginx"
spec:
  ingressClassName: nginx
  rules:
  - host: scds.tydnet.org
    http:
      paths:
      - pathType: Prefix
        path: /
        backend:
          service:
            name: scds-service
            port:
              number: 3000
  tls:
    - hosts:
      - scds.tydnet.org
      secretName: letsencrypt-prod-secretname
---
```

Codi 4.13: k8s manifest services

```yaml
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
  namespace: tfg-prod
  labels:
    app: scds
spec:
  acme:
    email: dlopezlo@gmail.com
    server: https :// acme-v02.api.letsencrypt.org/directory
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
    - http01:
        ingress:
          class: nginx
```

Codi 4.14: k8s manifest letsencrypt

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- gotk-components.yaml
- gotk-sync.yaml
```

Codi 4.15: FluxCD Kustomization

```
# This manifest was generated by flux. DO NOT EDIT.
---
apiVersion: source.toolkit.fluxcd.io/v1
kind: GitRepository
metadata:
  name: flux-system
  namespace: flux-system
spec:
  interval: 1m0s
  ref:
    branch: main
  secretRef:
    name: flux-system
  url: https://github.com/dlopezlo/fluxcd-infra.git
---
apiVersion: kustomize.toolkit.fluxcd.io/v1
kind: Kustomization
metadata:
  name: flux-system
  namespace: flux-system
spec:
  interval: 10m0s
  path: ./clusters/tfg-prod
  prune: true
  sourceRef:
    kind: GitRepository
    name: flux-system
```

Codi 4.16: FluxCD gotk-sync

Glossari

api Application Programming Interface. Part d'una aplicació que permet a agents externs interactuar amb ella de manera programàtica. . 1, 24, 25

Bastion Server Servidor accessible des d'internet, altament fortificat per a resistir atacs externs i que serveix com a punt d'entrada a servidors interns de la xarxa.. 1, 7

binari Fitxer executable. 1, 39

bootstrap Accions realitzades per a instalar un entorn des de zero.. 1, 20, 45, 46

CA Acrònim de Certificate Authority. Entitat certificadora de certificats de domini.. 1, 31, 53

cd Fa referència a "Desplegament Continu" i engloba a les accions automàtiques que s'executen per tal de desplegar el codi. 1, 10, 33

chart Paquet contenidor d'aplicacions en manifests de kubernetes.. 1, 17, 19, 20, 39, 42–44, 47, 51

ci Terme que fa referència a "Integració Contínua" i engloba les accions automàtiques que s'executen durant la integració del codi en el repositori. 1, 16, 33

clúster Agrupació de servidors que fan la mateixa tasca.. 1, 17–20, 24–26, 28, 29, 44–47

contenidor Tecnologia per a empaquetar aplicacions i les seves dependencies.. 1, 16, 17, 21, 34, 39–41

crontab Gestor de tasques programades en sistemes derivats de Unix. 1, 9

DevOps DevOPs és 1, 10

dns Domain Name System. 1, 11, 31

framework Conjunt d'eines, biblioteques, procesos i bones pràctiques que ens ajuden a programar de manera més eficient.. 1, 33, 34

GitOps GitOps és una forma de treballar que usa els repositoris de codi com a única font de la veritat per al desplegament de codi i infraestructura.. 1, 10, 17, 45, 50, 51

Go Llenguatge de programació creat per Google. 1, 10, 16, 19, 33

hash Funció resum criptogràfica.. 1, 41

http HyperText Transfer Protocol. Protocol d'intercanvi d'ihnformació de la WWW.. 1, 35

IaC De l'anglès, Infrastructure as Code. Definició de la infraestructura en codi per aprofitar les bones pràctiques de la codificació.. 1, 11, 16, 19, 21, 52

ingress Controlador de tràfic d'entrada.. 1, 29, 31, 39

ip Acrònim de Internet Protocol, fa referència a l'adreça que identifica un host a internet.. 1, 31

Json Acrònim de JavaScript Object Notation és format d'intercanvi de dades basat en text.. 1, 8, 10, 33–35

Kubernetes Sistema orchestrador de contenidors. 1, 10, 11, 16, 17, 19, 21, 23–25, 29, 31, 39, 42, 43, 45, 47, 51, 52

Makefile Sistema d'automatització usat comunment en la compilacio de codi.. 1, 19, 37, 39

manifest Fitxer que conté la definició de recursos de Kubernetes.. 1, 17, 19–21, 26, 39, 42, 43, 47

namespace Separació lògica en l'espai de memòria per a gestionar la colisió de recursos.. 1, 29, 39, 42

pipeline Conjunt de procesos de CI.. 1

pod Unitat mínima de procés en Kubernetes. Un pod pot estar format per un o més contenidors.. 1, 24, 31, 42

port-knocking Tècnica de seguretat que permet obrir ports en un firewall mitjançant l'enviament de seqüències croncretes de paquets.. 1, 7

pull request Petició d'integració de codi d'una branca en una altra en git.. 1, 10, 17, 21, 41, 48, 53

shift-left Es refereix a moure la responsabilitat d'una tasca al principi del seu cicle de vida, en aquest cas els usuaris o els desenvolupadors.. 1, 10

ssh Protocol d'access segur.. 1, 8, 10, 17, 35, 37, 50, 51, 53

TLS Transport Layer Security. Tecnologia de xifrat del canal de comunició.. 1, 29, 31, 33

token Cadena de text que serveix com a autenticació contra serveis.. 1, 23

workflow Conjunt de procesos de CI de Github Actions.. 1, 41, 48, 49

Capítol 5

Fonts d'informació i bibliografia

[1] Inc. Atlassian. *Trello Project management*. 2024. URL: https://trello.com/. Darrer accés: 15/04/2024.

[2] PlantUML Authors. *Gantt diagrams with PlantUML*. 2024. URL: https://plantuml.com/gantt-diagram. Darrer accés: 15/04/2024.

[3] DigitalOcean. *Accessing pods over a managed load-balancer from inside the cluster*. 2024. URL: https://github.com/digitalocean/digitalocean-cloud-controller-manager/blob/master/docs/controllers/services/examples/README.md#accessing-pods-over-a-managed-load-balancer-from-inside-the-cluster. Darrer accés: 18/05/2024.

[4] Inc. Digitalocean. *Digitalocean K8s cluster with terraform*. 2024. URL: https://docs.digitalocean.com/reference/terraform/reference/resources/kubernetes_cluster/. Darrer accés: 12/05/2024.

[5] Inc. Docker. *Docker, tecnologia de contenidors*. 2024. URL: https://docker.com. Darrer accés: 14/04/2024.

[6] Inc. Docker. *DockerHub, repositori de contenidors*. 2024. URL: https://hub.docker.com. Darrer accés: 14/04/2024.

[7] a CNCF Project. FluxCD authors. *Flux - the GitOps family of projects*. 2024. URL: https://fluxcd.io/. Darrer accés: 15/04/2024.

[8] The Linux Foundation. *Production-Grade Container Orchestration*. 2024. URL: https://kubernetes.io/. Darrer accés: 16/04/2024.

[9] Inc. GitHub. *GitHub Action documentation*. 2024. URL: https://github.com/features/actions. Darrer accés: 14/04/2024.

[10] Inc. GitHub. *Take GitHub to the command line*. 2024. URL: https://cli.github.com/. Darrer accés: 14/04/2024.

[11] Google. *encoding/json documentation*. 2024. URL: https://pkg.go.dev/encoding/json. Darrer accés: 12/05/2024.

[12] Google. *text/template documentation*. 2024. URL: https://pkg.go.dev/text/template. Darrer accés: 12/05/2024.

[13] Google. *What is Site Reliability Engineering (SRE)?* 2024. URL: https://https://sre.google/. Darrer accés: 21/05/2024.

[14] Inc. Google. *Go programming language*. 2024. URL: https://go.dev. Darrer accés: 14/04/2024.

[15] Hashicorp. *Automate infrastructure on any cloud*. 2024. URL: https://www.terraform.io/. Darrer accés: 15/04/2024.

[16] a CNCF Project. Helm Authors. *The package manager for Kubernetes*. 2024. URL: https://fluxcd.io/. Darrer accés: 15/04/2024.

[17] RedHat Inc. *What is GitOps??* 2024. URL: https://www.redhat.com/es/topics/devops/what-is-gitops. Darrer accés: 21/05/2024.

[18] Peter Kieltyka. *Chi documentation*. 2024. URL: https://go-chi.io/#/README. Darrer accés: 12/05/2024.

[19] KinD. *KinD, Kubernetes in Docker*. 2024. URL: https://kind.sigs.k8s.io/. Darrer accés: 14/04/2024.

[20] Kubernetes. *Nginx Ingress installation guide in Digital Ocean*. 2024. URL: https://kubernetes.github.io/ingress-nginx/deploy/#digital-ocean. Darrer accés: 20/05/2024.

[21] Testify Maintainers. *testify documentation*. 2024. URL: https://github.com/stretchr/testify. Darrer accés: 12/05/2024.