SCDS

SSH Config Deployment Service

Grau Enginyeria Informàtica, J[illegible]2024

Daniel López López

Tutor: Joaquín López Sánchez-Montañés

Agenda

Introducció

- El problema
- Solució proposada
- Objectius

El projecte

- Components
- Arquitectura
- L'aplicació
- La infraestructura
- Integració i Desplegament

Demo

Conclusions

Q/A

Introducció:
Context del problema

Gestionar la configuració SSH

- Simplificar l'ús
- Molts hosts (amb paràmetres variables)
- Comandes complexes

Reptes

- Manteniment (històric i versionat)
- Centralització: d'on trec la versió actual?
- Distribució a moltes persones

Moltes solucions possibles

- Recurs compartit
- CMDB

La millor solució?

?

Introducció:
Proposta de Solució

- Certa Maduresa
- Evolucionat
- Adopció prèvia tecnologies

Entorn objectiu

App Web

- Centralitzat
- Accessible
- Parametrizable

CI / CD

GITHUB

- Automatizacions
- QA
- GH Actions
- K8S
- Helm

- Traçabilitat
- Delegat als usuaris
- Pull Requests / Reviews

Objectius

Principals:

- Creem una app web per a generar i distribuir la configuració.
- Desplegament només amb interaccions amb el repositori de codi.
- El manteniment de la configuració es delega als usuaris.
- L'arquitectura de la solució és reaprofitable.

Secundaris:

- Però no menys importants…
- Assolir un coneixement pràctic en les tecnologies rellevants.

El projecte: Components

Codi

Conjunt d'eines i tecnologies usades en el desenvolupament de l'aplicació SCDS.

- Chi (Go framework web)
- Text/Template (plantilles)
- Testify (framework de tests)
- Json (format de dades)

IaC

Conjunt d'eines i tecnologies usades per a desplegar i allotjar la infraestructura.

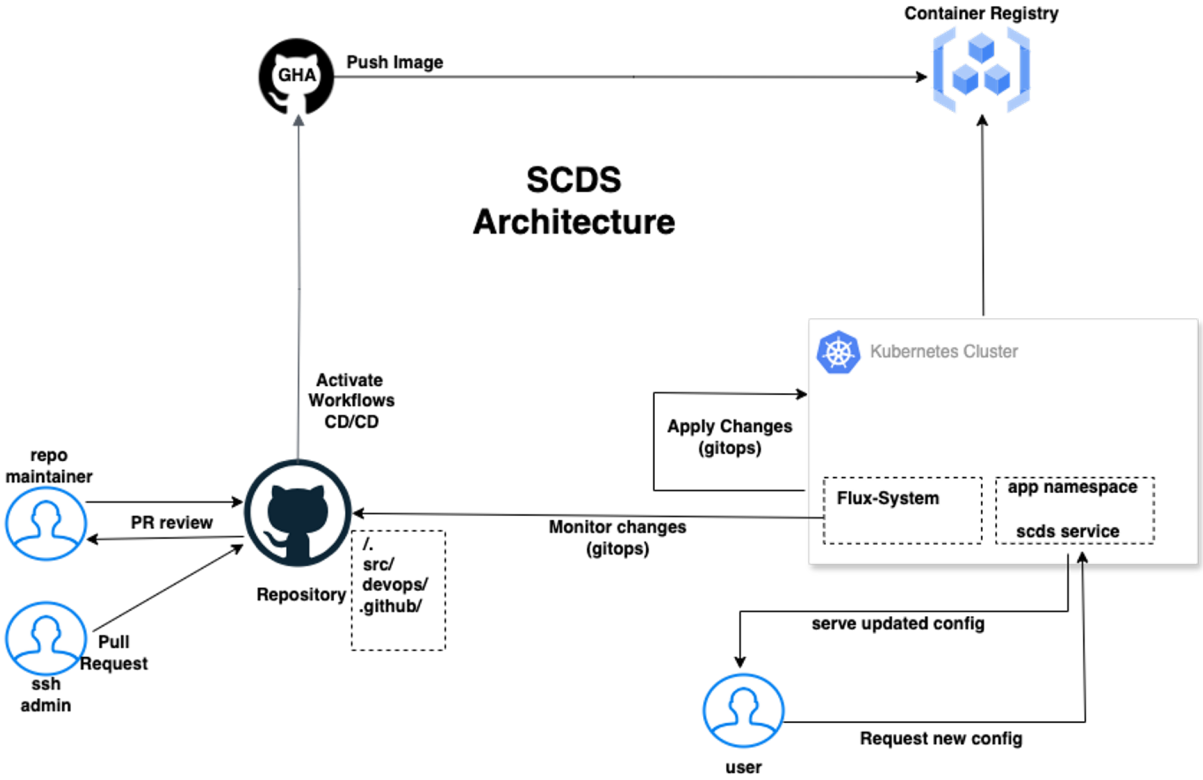
- Docker / DockerHub
- Kubernetes (KinD / PaaS)
- Terraform

CI/CD

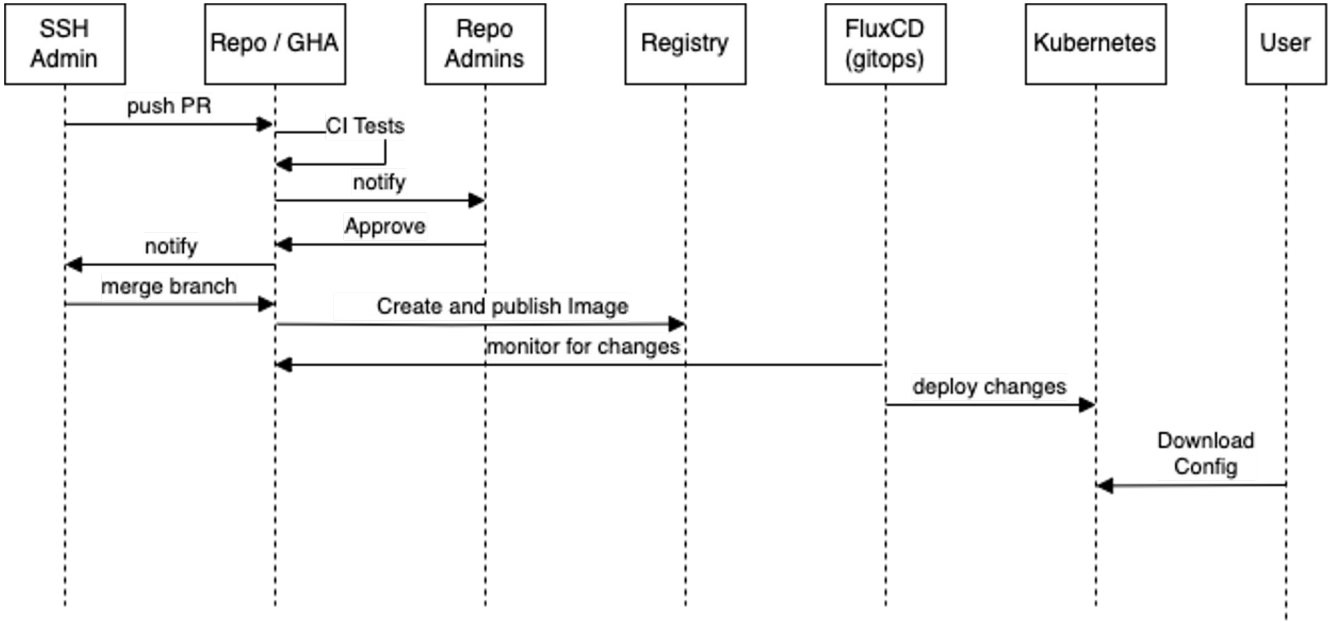
Conjunt d'eines i tecnologies usades per a integrar i publicar l'aplicació.

- GitHub /GH Actions / PRs
- FluxCD (GitOps)
- Helm

El projecte: L'arquitectura



El Projecte: Fluxes



L'aplicació: SCDS

- "Container first"
- Aplicació Web (Go Chi)
- Repo: /src
- Json
- Text/Templates

A

```
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"`
}
```

B

```
type Server struct {
        Router *chi.Mux
}

func CreateNewServer() *Server {
        s := &Server{}
        s.Router = chi.NewRouter()
        return s
}

func (s *Server) MountHandlers() {
        // Middlewares
        s.Router.Use(middleware.Logger)
        s.Router.Use(middleware.Timeout(time.Second * 3))
        // handlers
        s.Router.Get("/", WelcomeHandler)
        s.Router.Get("/sshconfig/{username}", getSSHConfigHandler)
}
```

C

```
data.Username = username
// read the file with the config in json format
sshData, err := os.ReadFile("ssh-data.json")
if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        log.Println("Error reading data file:", err)
        return
}

// Read the byts into an object
err = json.Unmarshal(sshData, &data)
if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        log.Println("Error parsing json:", err)
        return
}

// load the template
tpl, err := template.New("template.tpl").ParseFiles("template.tpl")
if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        log.Println("Error with templating system:", err)
}

// Execute renders the template with t
// responseWrite back to the user.
err = tpl.Execute(w, data)
if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        log.Println("Error with templating system:", err)
}
```

D

```
"hosts": [
    {
        "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/wg
    },
```

E

```
{{ 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-rs
 UseKeyChain yes
 AddkeysToAgent yes
 User {{ .Username }}
```

La Infraestructura: IaC

Metodología

- Infrastructure as Code
- Documentació
- Reusabilitat
- Automatització

Entorns

- Desenvolupament (dev) (KinD)
- Productiu (prod) DigitalOcean (SaaS)

Terraform

- Multi proveïdor
- Terraform
  - init
  - plan
  - apply

D

```
provider "digitalocean" {}

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

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

B

```
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
```

HashiCorp Terraform

A

```
# Create an image registry
resource "dockerhub_repository" "registry" {
  name        = "scds"
  namespace   = "dlopezlo"
  description = "My corporate container image registry"
  private     = true
}
```

C

Integració i Desplegament: CI/CD

GitHub Actions

- Creació, signatura i publicació de contenidors.
- Execució de testos
- github.com/dlopezlo/scds/.github/workflows

Kubernetes

- Namespace
- Service / ingress
- Deployment
- letsencrypt

- **Cert manager ***
- LoadBalancer *
- Secrets*

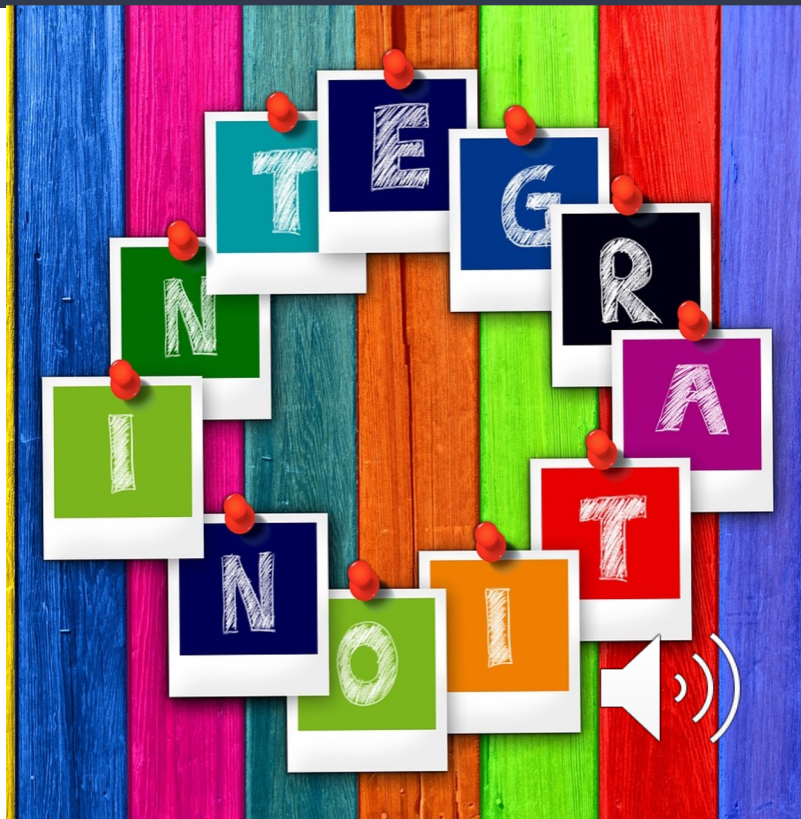
Desplegament

- Manifests
- Helm Charts
- FluxCD

- github.com/dlopezlo/scds/devops/k8s

GitOps (FluxCD)

- Repositori propi
- Multi entorn (dev / prod)
- github.com/dlopezlo/fluxcd-infra




DEMO!

Conclusions

Planificació

- Es compleixen les fites
- Cert-manager a l'entorn de dev.
- Integració Helm/FluxCD complexa.
- Transversal i d'ampli abast. Limits.

Objectius

- Aplicació web i les funcionalitats
- Desplegament amb Git I pels usuaris
- La solució és reutilitzable
- Adquirir experiència pràctica

Vies futures

- Creació i distribució de noves configuracions
- Millora de les eines de CI (workflows, seguretat)
- Integració empresarial i autenticació

Preguntes?

MOLTES GRÀCIES!