

Automatización de tareas en kubernetes con Argo Workflows



Índice

1. Objetivos del proyecto.....	3
2. Fundamentos Teóricos.....	4
2.1 ¿Argo? ¿ArgoCD? ¿Argo Workflows?.....	4
2.2 ¿Que es argoCD?.....	4
2.3 ¿Que es Argo Workflows?.....	5
2.3.1 Diferencias entre workflow y pipeline.....	6
2.3.2 Tipos de workflow templates.....	7
2.4 Otras tecnologías relevantes.....	10
2.4.1 Kubernetes / Kubectl.....	10
2.4.2 Docker.....	10
3. Explicación del escenario.....	11
3.1 Aplicación.....	11
3.2 Argo Workflows.....	12
4. Instalación de Argo.....	13
4.1 ArgoCD.....	13
4.1.1 CLI.....	13
4.1.3 Acceso a la UI.....	14
4.2 Argo Workflows.....	16
4.2.1 CLI.....	16
4.2.2 UI.....	16
4.2.3 Acceso a la UI.....	17
5. Demos.....	18
5.1 Demo numero 1.....	18
5.2 Demo numero 2.....	19
6. Dificultades, limitaciones y fortalezas.....	20
6.1 Limitaciones.....	20
6.2 Fortalezas.....	20
6.3 Dificultades a la hora de hacer las demos.....	21
6.3.1 Disaster recovery.....	21
6.3.2 Events.....	21
7. Conclusiones y propuestas para seguir trabajando sobre el tema.....	22
8. Bibliografía.....	23

1. Objetivos del proyecto

El objetivo de todas estas herramientas es agilizar y automatizar el despliegue y desarrollo continuo de aplicaciones web desplegadas en kubernetes, sumar a la potente tolerancia a fallos, y ofrecer una experiencia de mantenimiento, gitops y devops lo mas amplia y comoda posible.

Para ello, mas allá de argo workflows, argo dispone de diferente software, como ArgoCD para cumplir diferentes funciones, y que estas aplicaciones actúen en conjunto para lograr estos objetivos.

El proyecto Argo incluye guias detalladas para la instalación de las herramientas y ejemplos de uso. En esta memoria voy a hacer un pequeño resumen de algunas de las funcionalidades de este software.

2. Fundamentos Teóricos

2.1 ¿Argo? ¿ArgoCD? ¿Argo Workflows?

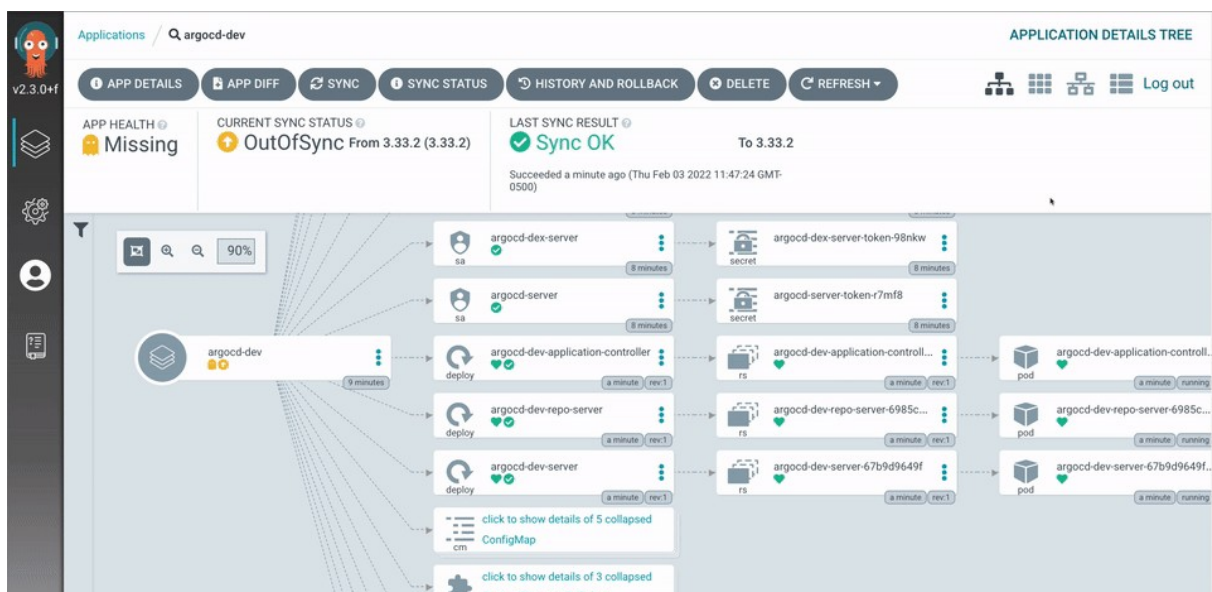
Aquí un pequeño glosario de introducción:

- **Argo:** Nombre común de los proyectos bajo esta etiqueta. Generalmente enfocado a despliegue de aplicaciones web. Otros ejemplos de software “argo” son Argo events o Argo Rollouts.
- **ArgoCD:** Se encarga de manejar el despliegue continuo de la infraestructura k8s
- **Argo Workflows:** Se encarga de manejar las workflows y automatizar tareas

El objetivo de todas estas herramientas es agilizar y automatizar el despliegue y desarrollo continuo de aplicaciones web desplegadas en kubernetes.

2.2 ¿Que es argoCD?

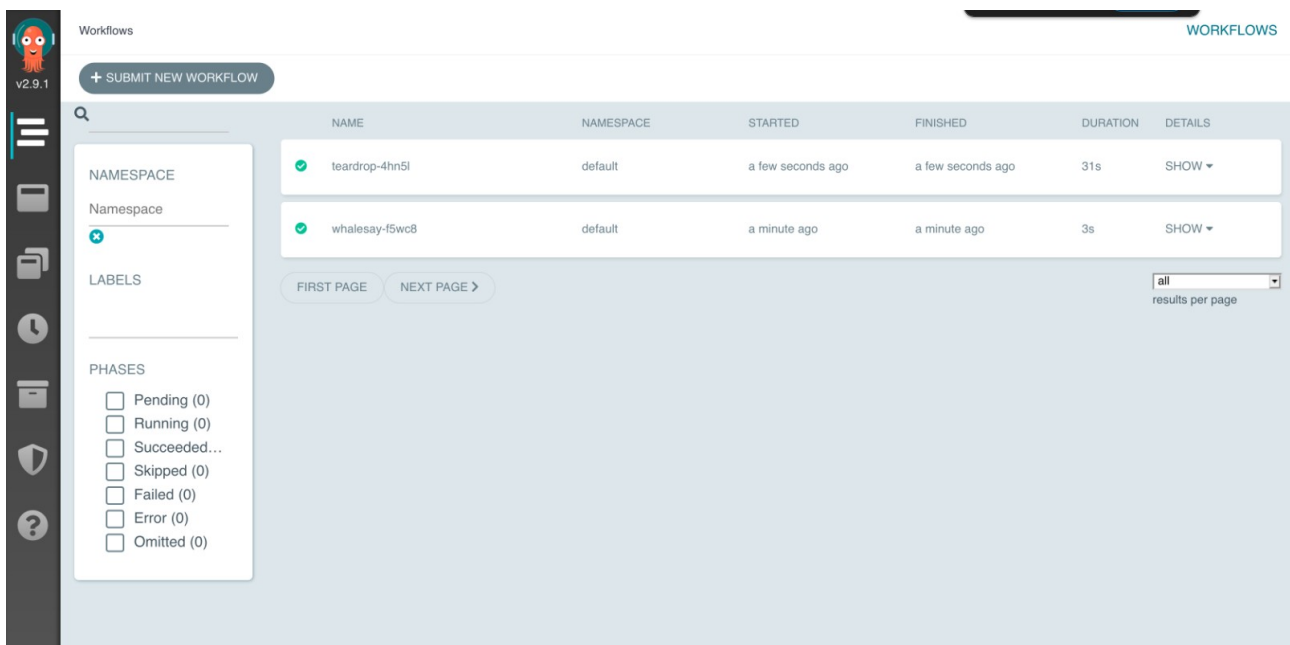
ArgoCD es una herramienta de Despliegue Continuo (CD) nativa para Kubernetes. A diferencia de las herramientas de CD externas que solo permiten implementaciones tras cada push, ArgoCD puede obtener código actualizado desde repositorios e implementarlo directamente en recursos de Kubernetes, es decir, permite cambiar la configuración de nuestro deployment a tiempo real, sincronizando la infraestructura, y la aplicación, ofreciendo una tolerancia al cambio manual y a potenciales acciones dañinas para el deployment, siempre que esté respaldado por un repositorio privado de Github



2.3 ¿Que es Argo Workflows?

Argo Workflows es un gestor de workflows, estos son usados para orquestar trabajos paralelos (similar al concepto de pipeline) en Kubernetes. Argo Workflows se implementa como un CRD (extension de la api) de Kubernetes. Define workflows donde cada paso es un contenedor.

Para entrar mas en detalle, un workflow combina la filosofía de los pipelines (secuencia de acciones para preparar un escenario) con el de la programación y el scripting (condicionales, bucles, definición de funciones (templates), etc), lo cual permite una mayor flexibilidad a la hora de la ejecución



The screenshot displays the Argo Workflows dashboard interface. On the left, there is a sidebar with navigation icons and a search bar. The main content area shows a table of workflows with columns for NAME, NAMESPACE, STARTED, FINISHED, DURATION, and DETAILS. Two workflows are listed: 'teardrop-4hn5l' and 'whalesay-f5wc8', both in the 'default' namespace and completed successfully. The 'teardrop-4hn5l' workflow took 31s, while 'whalesay-f5wc8' took 3s. Below the table, there are pagination controls for 'FIRST PAGE' and 'NEXT PAGE >', and a dropdown menu for 'all results per page'.

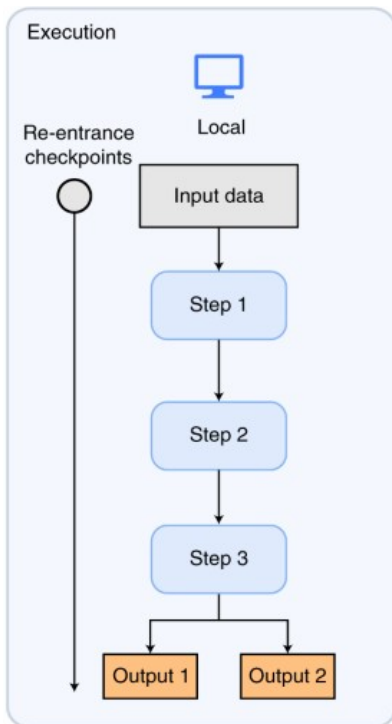
NAME	NAMESPACE	STARTED	FINISHED	DURATION	DETAILS
teardrop-4hn5l	default	a few seconds ago	a few seconds ago	31s	SHOW ▾
whalesay-f5wc8	default	a minute ago	a minute ago	3s	SHOW ▾

2.3.1 Diferencias entre workflow y pipeline

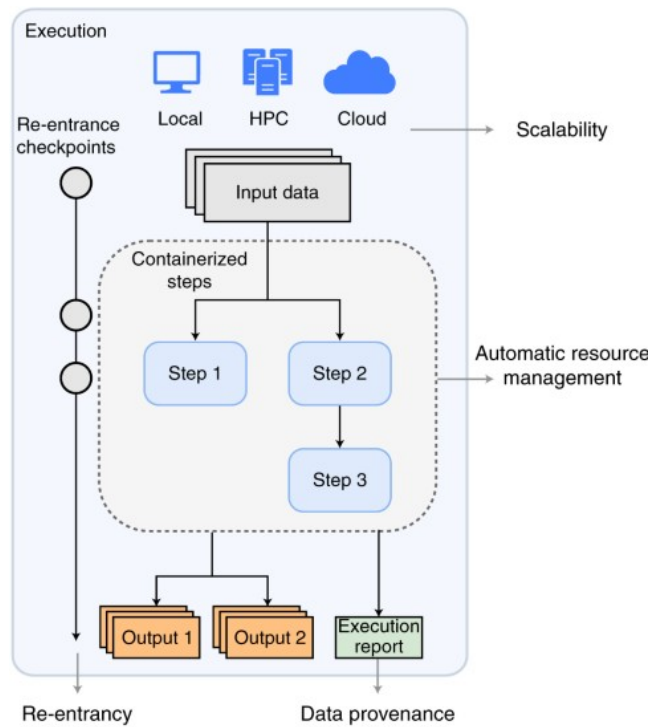
Tanto un workflow como una pipeline, es una serie de procesos encadenados, la diferencia es que un workflow, no es lineal, es decir, No se asume que los procesos se ejecuten concurrentemente.

Los workflows filtran y transforman datos, a menudo desencadenando eventos externos. La serie puede ramificarse o hacer bucles. No hay un "primer" proceso claramente definido: los datos pueden ingresar al workflow desde múltiples fuentes o "entrypoints". Cualquier proceso puede tomar datos y procesarlos como entrada, hacer algo con ellos y luego enviar sus resultados a otro proceso. Puede que no haya un "resultado final", más bien, múltiples procesos podrían entregar resultados a múltiples destinatarios.

Pipeline



Workflow



2.3.2 Tipos de workflow templates

Las templates son un concepto parecido al de las funciones en programación. Un step puede llamar a una template, y esta template hará lo que sea para lo que la hayamos programado. Se pueden distinguir 6 tipos, de entre ellos se encuentran los de **ejecución** y los de **definición**

Definición:

- **Container:**

Define un pod de la misma forma que si se definiera manualmente en kubernetes. Este pod normalmente se generará a partir de una imagen personalizada que contenga un script para ejecutar.

```
- name: whalesay
  container:
    image: docker/whalesay
    command: [cowsay]
    args: ["hello world"]
```

- **Script:**

Parecido a container, pero en vez de usar una imagen personalizada, se utiliza para imagenes genéricas sin un script. Este script esta definido dentro de la misma template

```
- name: random-int
  script:
    image: python:alpine3.6
    command: [python]
    source: |
      import random
      i = random.randint(1, 100)
      print(i)
```

- **Resource:**

Utilizado para acciones directas dentro del cluster (kubectl create, apply, delete...)

```
- name: variables1
  resource:
    action: create
    manifest: |
      apiVersion: v1
      kind: ConfigMap
      metadata:
        generateName: owned-eg-
      data:
        key: value
```

- **Suspend:**

El equivalente a un sleep en bash, suspende la ejecución por la duración que especifiquemos

```
- name: delay
  suspend:
    duration: "20s"
```


Ejecución:

- **Steps:**

Utilizado normalmente en el entrypoint de un workflow, permite definir pasos que llaman diferentes templates de forma lineal, o paralela. similar a una pipeline tradicional

```
- name: hello-hello-hello
  steps:
  - - name: step1
      template: prepare-data
  - - name: step2a
      template: run-data-first-half
    - name: step2b
      template: run-data-second-half
```

en este caso, el doble guión implica que se ejecutan consecutivamente, y el guión unico, que se ejecuta paralelo al step anterior.

- **DAG:**

Igual que steps, pero en vez de definir de forma lineal, requiere una serie de condiciones especificas (dependencias) para ejecutar cada step

```
- name: diamond
  dag:
  tasks:
  - name: A
    template: echo
  - name: B
    dependencies: [A]
    template: echo
  - name: C
    dependencies: [A]
    template: echo
  - name: D
    dependencies: [B, C]
    template: echo
```

2.4 Otras tecnologías relevantes



2.4.1 Kubernetes / Kubectl

Las aplicaciones de argo se instalan por medio de despliegues en kubernetes, para después manejar archivos YAML para la automatización de tareas, así que no es de extrañar que este sea relevante.



2.4.2 Docker

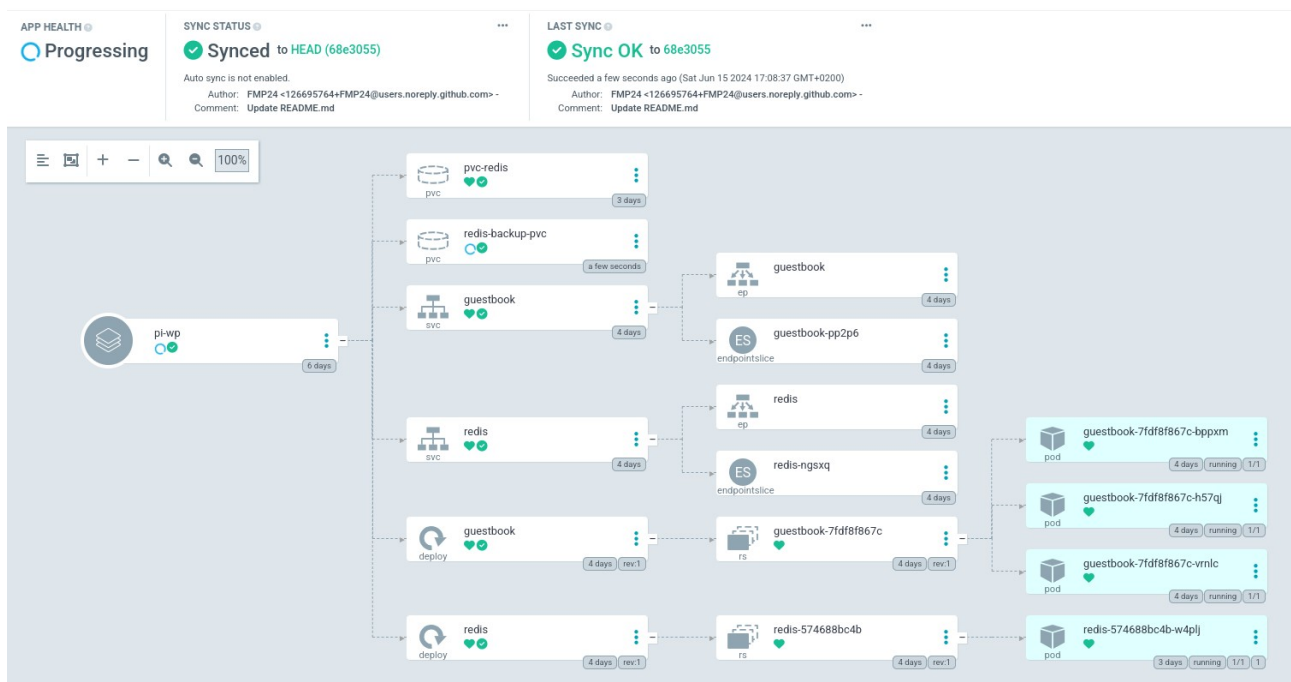
Principalmente, los pasos de los workflows van a usar contenedores para ser ejecutados, de eso se encargan las imágenes docker. Por ejemplo, Para ejecutar un script python utilizaríamos una imagen del repositorio de docker python. Si nuestra web está, por ejemplo, en alpine, optaríamos por una de las imágenes python:alpine.

3. Explicación del escenario

3.1 Aplicación

El escenario consiste en la aplicación de prueba guestbook, del Gonzalo Nazareno. Con 3 replicas para la aplicación y una para el redis. Los servicios son de tipo NodePort, ya que argo no tiene integración con ingress-nginx, sin embargo, si es posible evitar esto con otro tipo de proxy llamado Ambassador. Todo está conectado a este repositorio:

<https://github.com/FMP24/kd-PI>



Guestbook

SUBMIT

3.2 Argo Workflows

El argo workflows será una instalación limpia salvo por un workflow template añadido a posteriori para la segunda demo

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: print-guestbook
spec:
  entrypoint: print-guestbook
  volumes:
  - name: workdir
    persistentVolumeClaim:
      claimName: pvc-redis
  templates:
  - name: print-guestbook
    steps:
  - - name: print
      template: print-lista
  - name: print-lista
    script:
      image: redis:latest
      command: [sh]
      source: |
        redis-cli -h redis LRANGE lista 0 -1
  volumeMounts:
  - name: workdir
    mountPath: /data/
```

4. Instalación de Argo

4.1 ArgoCD

4.1.1 CLI

Descargamos la última release desde los repositorios oficiales de argo e instalamos manualmente

```
# Descargar el instalador
curl -sSL \
-o argocd-linux-amd64 \
https://github.com/argoproj/argo-cd/releases/latest/download/argocd-linux-amd64
```

```
# Instalar en $PATH con los permisos apropiados
sudo install -m 555 argocd-linux-amd64 /usr/local/bin/argocd
```

```
# Borrar el instalador
rm argocd-linux-amd64
```

4.1.2 UI

Creamos el namespace para albergar los deployments de argocd

```
kubectl create namespace argocd
```

Desplegamos el manifest desde el repositorio oficial

```
kubectl apply -n argocd \
-f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/
install.yaml
```

4.1.3 Acceso a la UI

Para conectarnos a la UI podemos hacer lo siguiente:

- **Acceder por NodePort**

```
kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "NodePort"}}'
```

Después, habría que mostrar el servicio “argocd-server” para mirar el puerto asignado al 443

```
root@nodo-1:~# kubectl get svc -n argocd | grep "argocd-server "
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
argocd-server	NodePort	10.43.73.86	<none>	80:31909/TCP, 443:32214/TCP	26d

en mi caso, la url sería `https://<IP | FQDN>:32214`

- **Port Forwarding**

```
kubectl port-forward svc/argocd-server -n argocd 8080:443 &
```

La URL sería `https://<IP | FQDN>:8080`

- **(Opcional): Configurar TLS**

La instalación de argocd viene con un certificado SSL estándar en los secrets, aun que los navegadores no reconozcan el certificado como válido, la comunicación estará cifrada igualmente. Para configurarlo, solo hay que añadir lo siguiente al `ingress.yaml`, debajo del `spec`.

```
spec:
  tls:
  - hosts:
    - <FQDN>
    secretName: argocd-secret
```

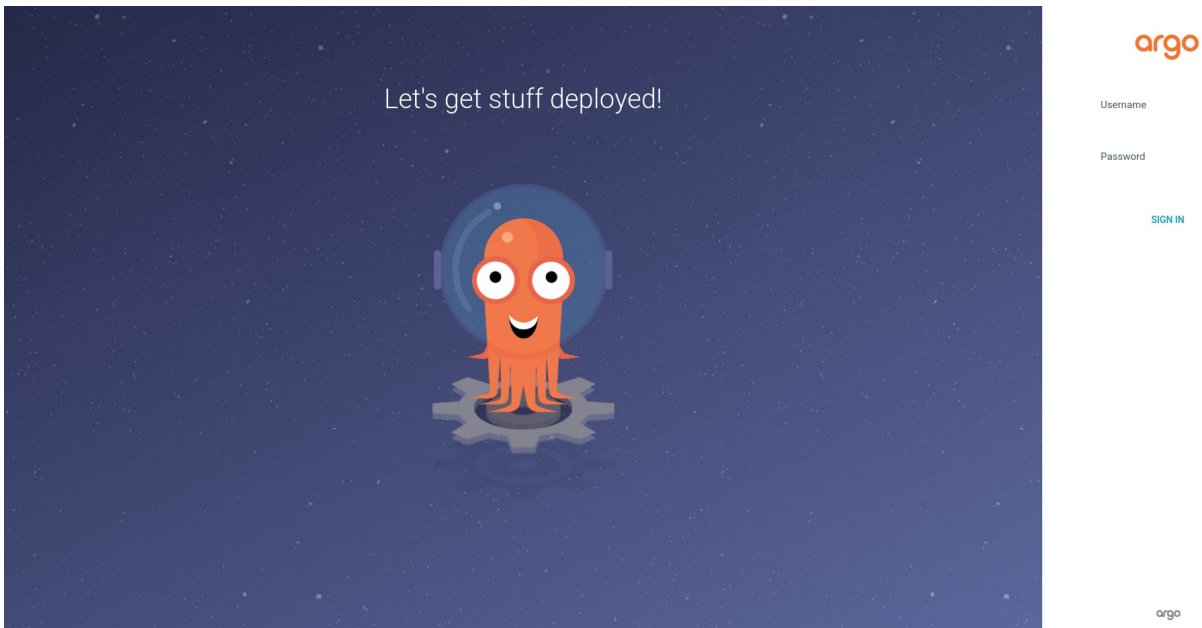
El recurso “argocd-secret” es el nombre del certificado que viene con la instalación.

Opcionalmente también podemos crear un certificado self-signed o utilizar un certificado de entidad certificadora como Let’s encrypt, o certificarlo con certbot como una página web normal. Un ejemplo de como añadir un certificado a los secrets:

```
openssl rsa -in tls-key.pem -out tls-decrypted-key.pem
```

```
kubectl create -n argocd secret tls argocd-server-tls \  
  --cert=/root/tls-crt.pem \  
  --key=/root/tls-decrypted-key.pem
```

La primera pantalla con la que nos encontramos es la de login.



- El usuario inicial siempre es **admin**,
- La contraseña inicial es un poco mas complicada, ya que se encuentra en los secrets:

```
root@nodo-1:~# kubectl get secret -n argocd | grep initial-admin-secret
```

NAME	TYPE	DATA	AGE
argocd-initial-admin-secret	Opaque	1	30d

Para obtenerla utilizamos el siguiente comando:

```
kubectl -n argocd get secret argocd-initial-admin-secret \  
  -o jsonpath="{.data.password}" | base64 -d
```

4.2 Argo Workflows

4.2.1 CLI

De forma similar a argoCD, argo workflows tiene su propia interfaz de comandos.

```
# Descargar el binario del repositorio oficial
curl -sLO https://github.com/argoproj/argo-workflows/releases/download/v3.4.17/
argo-linux-amd64.gz

# Unzip
gunzip argo-linux-amd64.gz

# Hacerlo ejecutable
chmod +x argo-linux-amd64

# Moverlo a $PATH
mv ./argo-linux-amd64 /usr/local/bin/argo

# Testear que se ha instalado correctamente
argo version
```

4.2.2 UI

Miramos la tabla de compatibilidad en la documentación oficial, en mi caso, kuberetes es 1.29 asi que instalo la version 3.5 de workflows. Por norma general, la ultima versión de k8s será compatible con la última o penúltima versión de argo workflows:

```
export ARGO_WORKFLOWS_VERSION="v3.5.0"
```

Creamos el namespace para albergar los deployments de argocd

```
kubectl create namespace argo
```

Desplegamos la release que hayamos elegido

```
kubectl apply -n argo -f
"https://github.com/argoproj/argo-workflows/releases/download/$
{ARGO_WORKFLOWS_VERSION}/quick-start-minimal.yaml"
```


4.2.3 Acceso a la UI

Para conectarnos a la UI podemos hacer lo siguiente:

- **Acceder por NodePort**

```
kubectl patch svc argo-server -n argo -p '{"spec": {"type": "NodePort"}}'
```

Lo siguiente es igual que en argoCD, cambiando el nombre del namespace.

- **Port Forwarding**

ArgoCD usa el puerto 8080, pero argo workflows utiliza el 2746

```
kubectl port-forward svc/argocd-server -n argocd 2746:443 &
```

La URL sería `https://<IP | FQDN>:2746`

5. Demos

5.1 Demo numero 1.

Duración estimada: 5min

Efecto esperado: Dejar claro el concepto de “Workflow”

Un workflow es en esencia, un script que se ejecuta dentro de un contenedor. En esta demo voy a mostrar como preparar y ejecutar un workflow simple. En este caso utilizare la imagen whalesay para mostrar un “hello world”:

<https://hub.docker.com/r/docker/whalesay/>

El script se puede encontrar en el repositorio oficial de argo. Lo único que he hecho es cambiar el argumento:

<https://raw.githubusercontent.com/argoproj/argo-workflows/main/examples/hello-world.yaml>

Desde el CLI podemos preparar workflows manualmente con “argo submit”

```
argo submit -n argo --watch hello-world.yaml
```

La opción --watch nos permitirá ver el progreso en la terminal:

```
Name:                hello-world-lkv2j
Namespace:           argo-workflows
ServiceAccount:      unset (will run with the default ServiceAccount)
Status:              Succeeded
Conditions:
  PodRunning         False
  Completed          True
Created:             Wed May 22 12:01:13 +0000 (35 seconds ago)
Started:             Wed May 22 12:01:12 +0000 (36 seconds ago)
Finished:            Wed May 22 12:01:48 +0000 (now)
Duration:            36 seconds
Progress:            1/1
ResourcesDuration:  20s*(1 cpu),13s*(100Mi memory)
STEP                TEMPLATE  PODNAME                DURATION  MESSAGE
  hello-world-lkv2j  whalesay  hello-world-lkv2j     29s
```

Después podemos ver la ejecución del workflow en los logs:

```
root@nodo-1:~# argo logs -n argo @latest
hello-world-lkv2j: time="2024-05-22T12:01:39.212Z" level=info msg="capturing
logs" argo=true
hello-world-lkv2j: _____
hello-world-lkv2j: < Hola Mundo! >
hello-world-lkv2j: -----
hello-world-lkv2j:      \
hello-world-lkv2j:      \
hello-world-lkv2j:      \
hello-world-lkv2j:          ##          .
hello-world-lkv2j:          ## ## ##      ==
hello-world-lkv2j:          ## ## ## ##    ===
hello-world-lkv2j:          /"____________________"/ ===
hello-world-lkv2j:  --- {--- --- --- --- --- ~ / ===- ---
hello-world-lkv2j:      \_____ o          ___/
hello-world-lkv2j:      \   \           ___/
hello-world-lkv2j:      \___\_____/
hello-world-lkv2j: time="2024-05-22T12:01:40.213Z" level=info msg="sub-process
exited" argo=true error="<nil>"
```

Después enseñé el mismo log en la UI, y editaré y re-ejecutare el script desde el mismo UI para que cambie el texto de nuevo.

5.2 Demo numero 2.

Duración estimada: 5min

Efecto esperado: Comprensión del contenido de un workflow

Demostración de un workflow que interactúe con el despliegue en argocd

En este caso voy a lanzar un workflow ya preparado de antemano con un pequeño script que imprima por pantalla los valores que están en el guestbook para demostrar como sacar valores hacia fuera del escenario. Estos valores en un escenario real podrían ser parámetros específicos de otro script por ejemplo. El workflow en si es muy simple, pero es una demostración. Se le pueden añadir mas steps como mandar por e-mail los valores por ejemplo.

6. Dificultades, limitaciones y fortalezas

Argo workflows es un software relativamente nuevo, por ello, si bien tiene un potencial enorme, aún tiene carencia frente a otras opciones como Gitlab CI/CD o Github Actions. Aun así tiene varias funcionalidades que lo separan de estas herramientas

6.1 Limitaciones

Dependencia del cluster / Poca flexibilidad

Argo workflows depende del cluster, que requiere una potencia considerable para poder ser usado de forma eficiente, ya que la ejecución de estos scripts se hace dentro y no se pueden destinar recursos externos por razones obvias. A mas scripts, mas recursos, y mas hay que escalar el cluster, lo cual normalmente cuesta dinero.

6.2 Fortalezas

Interaccion directa con el interior del deployment

Al ser un software nativo con kubernetes, este es capaz de interactuar directamente con los objetos dentro del cluster, haciendo posible una serie de nichos:

- **Disaster Recovery**

La posibilidad de ejecutar scripts dentro del cluster es excelente para la restauración de bases de datos, aseguradas en repositorios privados, o dentro de Artifacts, para poder ser invocados directamente por los workflows

- **Gestión y escalado directo de recursos**

Pongamos que tengo un evento que detecta cuando el uso de memoria de un pod llega a un cierto valor alto, ese evento activaría un workflow, que se encargaría de crear mas replicas para solucionar este cuello de botella.

6.3 Dificultades a la hora de hacer las demos

La curva de aprendizaje de argo workflows hace un ángulo de 90°, es decir, es MUY complejo para un principiante, lo cual definitivamente ha influido en como he sobrellevado la realización de demos.

6.3.1 *Disaster recovery*

En un principio, quería mostrar ejemplos de uso reales, pero debido a la complejidad, he sido incapaz de hacer funcionar nada. Por ejemplo, intenté hacer triggers en un principio con el webhook de argo events, pero fui incapaz de hacerlo funcionar, también intenté hacer un disaster recovery, pero tampoco pude realizarlo. De todos modos el intento esta en el repositorio de github del proyecto:

<https://github.com/FMP24/kd-wordpress-PI/blob/main/workflow/backup-redis.yaml>

En concreto, el mayor problema que he tenido es que no se como traspasar el archivo de backup entre pvcs. De hecho no se siquiera como comprobar si esta siendo guardado en el pvc para backups, o el problema es otro.

6.3.2 *Events*

Quería hacer una demo que subiera una imagen nueva a dockerhub cuando se hiciera un cambio en la aplicación. En retrospectiva, esta demo es incluso mas complicada que la de arriba, y el build en vivo consumiría bastante tiempo, así que no es ideal. De todos modos el problema que he tenido es que no he sido capaz de conectar el webhook necesario para que argo events se comuniquen con argo workflows.

7. Conclusiones y propuestas para seguir trabajando sobre el tema.

Argo workflows es una herramienta muy extensa, que requiere de altos conocimientos técnicos, así que para estar al tanto de todo lo que ofrece sugiero investigar sobre:

- **Automatización de despliegues:**

Con scripts podemos subir imágenes, por lo que cada vez que cambiemos el código de la aplicación en la rama del repositorio que le digamos a un event, podemos forzar una build, generando esta nueva imagen y aplicándola al repositorio donde se aloja el despliegue, y que ArgoCD se ocupe del resto.

- **Disaster recovery:**

Como he mencionado antes, no he podido sacar un disaster recovery que funcione, así que sería interesante investigar mas a fondo sobre ello, y en vez de utilizar un raw file para el backup se pueden investigar los **Artifact Buckets**. Que son almacenes de archivos integrables a argo workflows, que pueden actuar como inputs y outputs de los scripts

- **Argo events, Argo rollouts:**

Argo events ya se ha mencionado un par de veces, es el controlador de eventos y triggers para argo workflows, y argo rollouts, es el controlador de entornos de argo. Son extensiones que vale la pena explorar

8. Bibliografía

Argo Project: <https://argoproj.github.io/>

ArgoCD: <https://argo-cd.readthedocs.io/en/latest/>

Argo Workflows: <https://argo-workflows.readthedocs.io/en/latest/>

Argo Events: <https://argoproj.github.io/argo-events/>

Argo Rollouts: <https://argoproj.github.io/argo-rollouts/>

k3s (Distribución de Kubernetes utilizada en este proyecto): <https://k3s.io/>

Docker: <https://www.docker.com/>

8.1. Referencias

Repositorio del proyecto: <https://github.com/FMP24/kd-PI/>

Presentación: <https://view.genially.com/666de07a95d9bd0014c8de0e/presentation-automatizacion-de-tareas-en-kubernetes-con-argo-workflows>

demo 1: <https://youtu.be/NnynSKdPNZ0>

demo 2: <https://youtu.be/Q0Gbem6y-Cw>