**Release canary con ArgoCD**

**Requisitos previos**

- minikube versión v1.18.3 instalado
- proyecto practica-cd en github.com (https://github.com/francois-poirier/practica-cd)
- cuenta dockerhub personal, donde se encuentra 3 imágenes del proyecto practica-cd (https://hub.docker.com/repository/docker/fpoirier2020/practicacd)

Argo es un conjunto de herramientas para el despliegue continuo :

- ArgoCD
- Argo Workflow
- Argo Rollout
- Argo Event

**Configurando ArgoCD**

Para instalar ArgoCD, creamos una nuevo namespace en minikube llamado «argocd» y desplegamos el manifiesto que Argo proporciona en su repositorio. A continuación y para tener acceso cambiamos el tipo de servicio a LoadBalancer ya que viene por defecto como ClusterIP.

- **kubectl create namespace argocd**
- **kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml**
- **kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'**
- **kubectl get pod -n argocd**

```
francois@master-node:~$ kubectl get pod -n argocd
NAME                                  READY   STATUS    RESTARTS   AGE
argocd-application-controller-5cfb8d686c-8jwhv   1/1     Running   4          5d22h
argocd-dex-server-5cf8dd69f5-gv9ww    1/1     Running   4          5d22h
argocd-redis-6d7f9df848-ffb4c         1/1     Running   4          5d22h
argocd-repo-server-56b75988dc-p5w2k   1/1     Running   4          5d22h
argocd-server-6766455855-qdksd        1/1     Running   4          5d22h
francois@master-node:~$
```

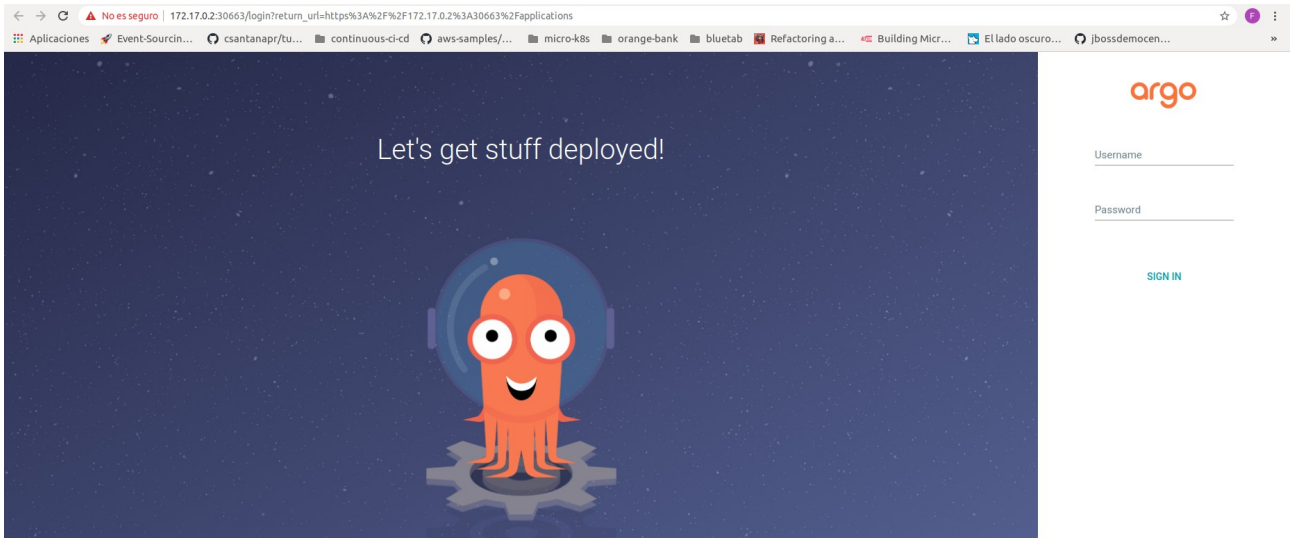
ArgoCD por defecto crea un usuario administrador cuya password se obtiene ejecutando el siguiente comando:

- **kubectl get pods -n argocd -l app.kubernetes.io/name=argocd-server -o name | cut -d'/' -f 2**

Y para acceder a la consola de administración de ArgoCD lo haremos ejecutando:

- **minikube service argocd-server -n argocd**

Y veremos una página como:




Se puede cambiar la password por consola o por ArgoCD CLI

**Instalando ArgoCD CLI**

- **curl** https://github.com/argoproj/argo-cd/releases/download/v1.6.0/argocd-linux-amd64
- **mv argocd-linux-amd64** /usr/local/bin/argocd
- chmod +x /usr/local/bin/argocd

Miramos si el servidor de ArgoCD corre correctamente con el siguiente comando

- **kubectl get svc -n argocd argocd-server**

Montamos el por-forward

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

Nos logamos desde CLI

- **argocd login** locahost:8080

cambiamos la password

- **argocd account update-password**

**Nuestra aplicación y sus releases**

Voy a usar una aplicación simple que proporciona una API que devuelve un código json como este:

**{"version":"1.0","status":"OK"}**

Existen 3 releases generadas y subidas en mi cuenta de dockerhub a la siguiente url https://hub.docker.com/repository/docker/fpoirier2020/practicacd

Es realmente sencillo. Utilizaremos el atributo versión para poder observar las actualizaciones de nuestra aplicación y el estado como una medida de salud de la aplicación.

Crearemos tres versiones de nuestra aplicación para realizar nuestras pruebas de actualización:

- 1.0 que devuelve version 1.0 y status OK
- 2.0 que devuelve version 2.0 y status OK
- 3.0 que devuelve version 3.0 y status KO

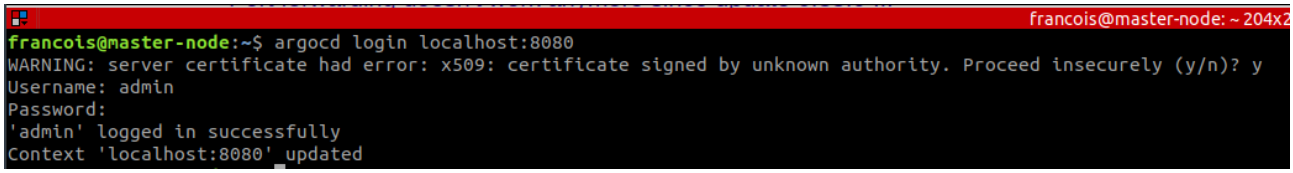
**Desplegando nuestra aplicación desde ArgoCD CLI**

Montamos el por-forward

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

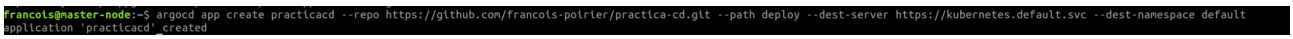
Login

- argocd login localhost:8080




Despliegue de la aplicación en ArgoCD

- argocd app create practicacd --repo https://github.com/francois-poirier/practica-cd.git --path deploy --dest-server https://kubernetes.default.svc --dest-namespace default




En el comando se especifica con la opción --path la carpeta del repositorio donde encontrar los objetos kubernetes de despliegue. En nuestro caso deploy.

Sincronización

- argocd app sync practicacd

```
francois@master-node:~$ argocd app sync practicacd
TIMESTAMP                  GROUP        KIND   NAMESPACE                  NAME    STATUS   HEALTH           HOOK  MESSAGE
2020-06-24T14:19:32+02:00            Service     default           practicacd    Synced  Progressing
2020-06-24T14:19:32+02:00   apps  Deployment     default           practicacd    Synced  Progressing
2020-06-24T14:19:33+02:00   apps  Deployment     default           practicacd    Synced  Progressing              deployment.apps/practicacd configured
2020-06-24T14:19:33+02:00            Service     default           practicacd    Synced  Progressing              service/practicacd unchanged
2020-06-24T14:19:33+02:00   apps  Deployment     default           practicacd  OutOfSync  Progressing              deployment.apps/practicacd configured

Name:               practicacd
Project:            default
Server:             https://kubernetes.default.svc
Namespace:          default
URL:                https://localhost:8080/applications/practicacd
Repo:               https://github.com/francois-poirier/practica-cd.git
Target:
Path:               deploy
SyncWindow:         Sync Allowed
Sync Policy:        <none>
Sync Status:        Synced to  (a2b40a1)
Health Status:      Progressing

Operation:          Sync
Sync Revision:      a2b40a17e9ddbca68058294b97c45c3abb117342
Phase:              Succeeded
Start:              2020-06-24 14:19:32 +0200 CEST
Finished:           2020-06-24 14:19:33 +0200 CEST
Duration:           1s
Message:            successfully synced (all tasks run)

GROUP  KIND        NAMESPACE  NAME        STATUS  HEALTH       HOOK  MESSAGE
       Service     default    practicacd  Synced  Progressing        service/practicacd unchanged
apps   Deployment  default    practicacd  Synced  Progressing        deployment.apps/practicacd configured
```

Comprobamos que las 3 replicas de la release 1.0 están desplegados

- kubectl get pods,services

```
francois@master-node:~$ kubectl get pods,services
NAME                                READY   STATUS    RESTARTS   AGE
pod/practicacd-5c5cccc899-2zq8q   1/1     Running   0          18m
pod/practicacd-5c5cccc899-bg9jr   1/1     Running   0          18m
pod/practicacd-5c5cccc899-kgmv2   1/1     Running   0          18m

NAME                  TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
service/kubernetes    ClusterIP      10.96.0.1        <none>        443/TCP          6d4h
service/practicacd    LoadBalancer   10.102.218.207   <pending>     8081:30439/TCP   155m
```

Preguntamos por la IP del cluster minikube y realizamos un curl cada 500ms para alcanzar el endpoint REST, abriendo previamente el tunnel.

- minikube ip
- minikube tunnel
- while true; do curl http://172.17.0.2:30439/me/version | jq .version; sleep 0.5; done

```
francois@master-node:~$ while true; do curl http://172.17.0.2:30439/me/version | jq .version; sleep 0.5; done
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    31    0    31    0     0   3875      0 --:--:-- --:--:-- --:--:--  4428
"1.0"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    31    0    31    0     0   5166      0 --:--:-- --:--:-- --:--:--  5166
"1.0"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    31    0    31    0     0   5166      0 --:--:-- --:--:-- --:--:--  5166
"1.0"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
```

## Argo Rollout

Otro componente de la suite Argo. Mejora las estrategias básicas de implementación proporcionadas en Kubernetes y agrega funcionalidad como Canary Deployment o Blue / Green Deployment. Aquí vamos a centrarnos en Canary Deployment. Realmente no existe una estrategia precisa para un Canary Deployment, el concepto básico es que redirigiremos una parte del tráfico de la versión estable (base line) de la aplicación a la nueva versión (canary). A cada uno de definir su propia estrategia.

### Configurando Argo Rollout

Para instalar Argo Rollout, creamos una nuevo namespace en minikube llamado «argo-rollouts» y desplegamos el manifiesto que Argo proporciona en su repositorio.

- kubectl create namespace argo-rollouts
- kubectl apply -n argo-rollouts -f https://raw.githubusercontent.com/argoproj/argo-rollouts/stable/manifests/install.yaml

Argo Rollout proporciona un resource de tipo CustomeResourceDefinition que viene sobreescribir el recurso Deployment: **Rollout**

Para utilizarlo, vamos a modificar el recurso Deployment cambiando los atributos apiVersion y kind

apiVersion: argoproj.io/v1alpha1
kind: Rollout

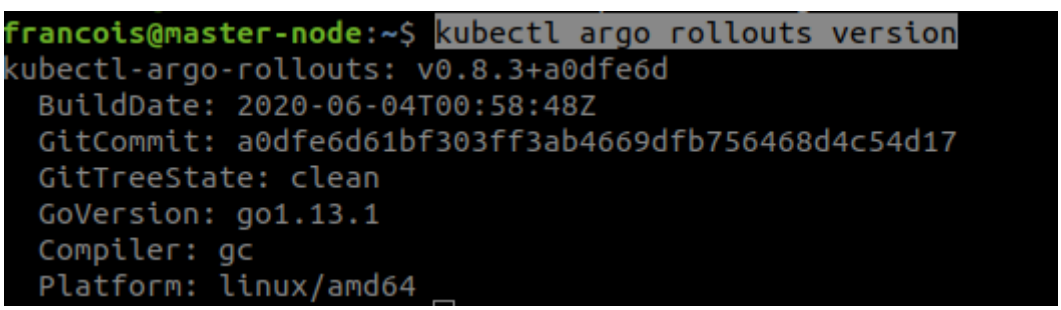
Subimos los cambios al repositorio y volvemos a sincronizar ArgoCD contra el repositorio.

- argocd app sync practicacd

De momento nada cambia

### Instalación del plugin Argo Rollouts Kubectl

- curl -LO https://github.com/argoproj/argo-rollouts/releases/latest/download/kubectl-argo-rollouts-linux-amd64
- chmod +x ./kubectl-argo-rollouts-linux-amd64
- sudo mv ./kubectl-argo-rollouts-linux-amd64 /usr/local/bin/kubectl-argo-rollouts
- kubectl argo rollouts version



**Rollout Strategy**

El recurso Rollout permite un gran abanico de estrategia para la canary. A continuación la estrategia definida para la practica.

```
strategy:
  canary:
    analysis:
      templates:
      - templateName: webcheck
    steps:
    - setWeight: 20
    - pause:
        duration: "30s"
    - setWeight: 50
    - pause:
        duration: "30s"
    - setWeight: 100
```

Definimos 5 pasos a nuestro rolling update:

- Redirigimos el 20% del tráfico a nuestra nueva versión
- Esperamos 30s
- Redirigimos el 50% del tráfico a nuestra nueva versión
- Esperamos 30s
- Redirigimos el 100% del tráfico a nuestra nueva versión

Subimos los cambios al repositorio y volvemos a sincronizar ArgoCD contra el repositorio.

- argocd app sync practicacd

De momento nada cambia porque seguimos con la imagen de la release 1.0.

Actualicemos nuestro Rollout cambiando la etiqueta de nuestra imagen a la versión 2.0.

Subimos los cambios al repositorio y volvemos a sincronizar ArgoCD contra el repositorio.

```
francois@master-node:~/workspace/practica-cd$ sed -i 's/1.0/2.0/g' deploy/deployment-service.yaml
francois@master-node:~/workspace/practica-cd$ git commit -am "bump: 2.0"
[master 129a58b] bump: 2.0
 1 file changed, 1 insertion(+), 1 deletion(-)
francois@master-node:~/workspace/practica-cd$ git push origin master
Username for 'https://github.com': fpoirier2002@gmail.com
Password for 'https://fpoirier2002@gmail.com@github.com':
Enumerando objetos: 7, listo.
Contando objetos: 100% (7/7), listo.
Compresión delta usando hasta 8 hilos
Comprimiendo objetos: 100% (4/4), listo.
Escribiendo objetos: 100% (4/4), 352 bytes | 352.00 KiB/s, listo.
Total 4 (delta 2), reusado 0 (delta 0), pack-reusado 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/francois-poirier/practica-cd.git
   d0c352a..129a58b  master -> master
francois@master-node:~/workspace/practica-cd$ argocd app sync practicacd
TIMESTAMP                  GROUP             KIND   NAMESPACE                  NAME    STATUS    HEALTH        HOOK  MESSAGE
2020-06-24T17:50:42+02:00                    Service     default           practicacd    Synced   Healthy
2020-06-24T17:50:42+02:00   apps          Deployment     default           practicacd  OutOfSync  Healthy
2020-06-24T17:50:42+02:00  argoproj.io     Rollout     default           practicacd  OutOfSync  Healthy
2020-06-24T17:50:42+02:00   apps          Deployment     default           practicacd  OutOfSync  Healthy              ignored (requires pruning)
2020-06-24T17:50:42+02:00                    Service     default           practicacd    Synced   Healthy              service/practicacd unchanged
2020-06-24T17:50:42+02:00  argoproj.io     Rollout     default           practicacd  OutOfSync  Healthy              rollout.argoproj.io/practicacd configured
2020-06-24T17:50:42+02:00  argoproj.io     Rollout     default           practicacd    Synced  Healthy             rollout.argoproj.io/practicacd configured
```

Acabo de un tiempo, podemos apreciar como esta evolucionado la canary para llegar hasta el 100% de la nueva versión.

```
francois@
"2.0"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    31    0    31    0     0   6200      0 --:--:-- --:--:-- --:--:--  6200
"1.0"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    31    0    31    0     0   5166      0 --:--:-- --:--:-- --:--:--  6200
"2.0"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    31    0    31    0     0   3875      0 --:--:-- --:--:-- --:--:--  3875
"2.0"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    31    0    31    0     0   7750      0 --:--:-- --:--:-- --:--:-- 10333
"1.0"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    31    0    31    0     0   6200      0 --:--:-- --:--:-- --:--:--  7750
"1.0"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    31    0    31    0     0   6200      0 --:--:-- --:--:-- --:--:--  7750
"1.0"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    31    0    31    0     0  31000      0 --:--:-- --:--:-- --:--:-- 31000
"1.0"
```

Este rolling update puede también realizarse con el CLI y la ayuda del plugin **Argo Rollouts Kubectl** instalado en los interiores pasos.

- kubectl argo rollouts set image practicacd "*=fpoirier2020/practicacd:2.0"

Este plugin ofrece también la posibilidad de seguir en tiempo real la evolución de la canary

- sudo kubectl argo rollouts get rollout practicacd -w

```
Name:            practicacd
Namespace:       default
Status:          ✔ Healthy
Strategy:        Canary
  Step:          4/4
  SetWeight:     100
  ActualWeight:  100
Images:          fpoirier2020/practicacd:2.0 (stable)
Replicas:
  Desired:       3
  Current:       3
  Updated:       3
  Ready:         3
  Available:     3

NAME                                  KIND        STATUS        AGE  INFO
⟳ practicacd                          Rollout     ✔ Healthy     81m
├──# revision:2
│  └──⧉ practicacd-698557cb84         ReplicaSet  ✔ Healthy     47m  stable
│     ├──□ practicacd-698557cb84-qspr4  Pod       ✔ Running     47m  ready:1/1
│     ├──□ practicacd-698557cb84-tdt8r  Pod       ✔ Running     46m  ready:1/1
│     └──□ practicacd-698557cb84-llmpz  Pod       ✔ Running     46m  ready:1/1
└──# revision:1
   └──⧉ practicacd-5f8cdcd5c5         ReplicaSet  • ScaledDown  62m
```

### Analysis Run

Pero lo interesante es poder observar el cambio de versión y poder tomar una decisión al respecto. Si el comportamiento de nuestra Canary no es la correcta, debemos poder retroceder automáticamente.

Para esto usamos un nuevo recurso **AnalysisRun**. Usaremos el recurso **AnalysisTemplate** para describir nuestras pruebas y modificaremos ligeramente nuestro Rollout para declarar el uso de nuestro **AnalysisRun**.

```yaml
strategy:
  canary:
    analysis:
      templates:
      - templateName: webcheck
    steps:
    - setWeight: 20
    - pause:
        duration: "30s"
    - setWeight: 50
    - pause:
        duration: "30s"
```

```yaml
---
apiVersion: argoproj.io/v1alpha1
kind: AnalysisTemplate
metadata:
  name: webcheck
spec:
  args:
  - name: api-url
    value: http://172.17.0.2:30439/me/version
  metrics:
  - name: webcheck
    failureLimit: 1
    interval: 5s
    successCondition: result == "OK"
    provider:
      web:
        url: "{{ args.api-url }}"
        jsonPath: "{$.status}"
```

Para el análisis, existe varios providers. Hemos elegido el provider **web** ya que nos permite fácilmente analizar la respuesta json de un servicio REST. Para checkear la respuesta, nos basamos en el atributo status de la respuesta json. Si todo va bien debe contestar “OK”, en caso contrario el análisis esta en error. El checkeo se realiza cada 5 segundos y el margen de error es de 1 fallo. Cuando se produzca, nuestro Analisis pasara en fail, el rolling update parara y el mecanismo de rollback se ejecutara.

**Demostración instalando la release 3.0, esta version devuelve un KO.**

- kubectl argo rollouts set image practicacd "*=fpoirier2020/practicacd:3.0"
- kubectl argo rollouts get rollout practicacd -w

```
Replicas:
  Desired:       3
  Current:       3
  Updated:       0
  Ready:         3
  Available:     3

NAME                                KIND         STATUS        AGE    INFO
⟳ practicacd                        Rollout      ✖ Degraded    5h11m
├──# revision:10
│  ├──⧉ practicacd-698557cb84       ReplicaSet   • ScaledDown  4h37m  canary
│  └──α practicacd-698557cb84-10    AnalysisRun  ✖ Failed      3m20s  ✖ 2
├──# revision:9
│  ├──⧉ practicacd-7d95f59585       ReplicaSet   ✔ Healthy     171m   stable
│  │  ├──□ practicacd-7d95f59585-hstrn  Pod      ✔ Running     5m58s  ready:1/1
│  │  ├──□ practicacd-7d95f59585-6mb86  Pod      ✔ Running     5m26s  ready:1/1
│  │  └──□ practicacd-7d95f59585-84pkk  Pod      ✔ Running     4m53s  ready:1/1
│  └──α practicacd-7d95f59585-9     AnalysisRun  ✔ Successful  5m58s  ✔ 14
```

La versión 3.0 no ha llegado a instalarse.