CONTINUOUS DEPLOYMENT WITH ARGOCD

1- Introduction

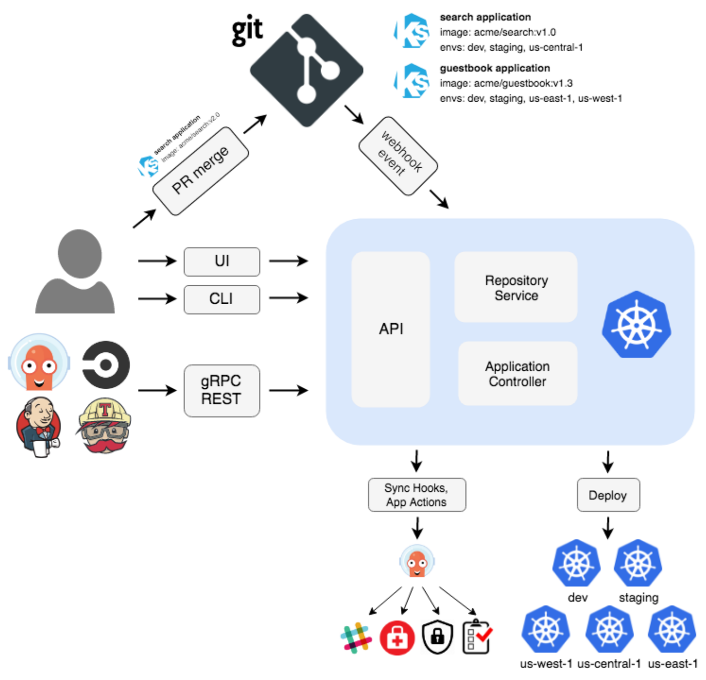
[Argo CD] (https://argoproj.github.io/argo-cd/) is a declarative, GitOps continuous delivery tool for Kubernetes. The core component of Argo CD is the Application Controller, which continuously monitors running applications and compares the live application state against the desired target state defined in the Git repository. This powers the following use cases:

Automated deployment : controller pushes the desired application state into the cluster automatically, either in response to a Git commit, a trigger from CI pipeline, or a manual user request.

Observability : developers can quickly find if the application state is in sync with the desired state. Argo CD comes with a UI and CLI which helps to quickly inspect the application and find differences between the desired and the current live state.

Operation : Argo CD UI visualizes the entire application resource hierarchy, not just top-level resources defined in the Git repo. For example, developers can see ReplicaSets and Pods produced by the Deployment defined in Git. From the UI, you can quickly see Pod logs and the corresponding Kubernetes events. This turns Argo CD into very powerful multi-cluster dashboard.

2- INSTALL ARGO CD



ArgoCD is composed of three mains components:

API Server: Exposes the API for the WebUI / CLI / CICD Systems

Repository Server: Internal service which maintains a local cache of the git repository holding the application manifests

Application Controller: Kubernetes controller which controls and monitors applications continuously and compares that current live state with desired target state (specified in the repository). If a `OutOfSync` is detected, it will take corrective actions.

- ### Install Argo CD

All those components could be installed using a manifest provided by the Argo Project:

```
> kubectl create namespace argocd
> kubectl apply -n argocd -f
https://raw.githubusercontent.com/argoproj/argo-cd/v2.0.4/manifests/install.yaml
```

- ### Install Argo CD CLI

To interact with the `API Server` we need to deploy the CLI:

```
> sudo curl --silent --location -o /usr/local/bin/argocd
https://github.com/argoproj/argo-cd/releases/download/v2.0.4/argocd-linux-amd64
> sudo chmod +x /usr/local/bin/argocd
```

3. CONFIGURE ARGOCD

As the Argo CD has been deployed, we now need to configure argocd-server and then login:

- ### Expose argocd-server

By default argocd-server is not publicaly exposed. For the purpose of this workshop, we will use a Load Balancer to make it usable:

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

Wait about 2 minutes for the LoadBalancer creation

```
> export ARGOCD_SERVER=`kubectl get svc argocd-server -n argocd -o json | jq
--raw-output '.status.loadBalancer.ingress[0].hostname'`
```

- **Login**

The initial password is autogenerated with the pod name of the ArgoCD API server:

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

Using `admin` as login and the autogenerated password:

```
> argocd login $ARGOCD_SERVER --username admin --password $ARGO_PWD --insecure
```

You should get as an output:

```
'admin' logged in successfully
```

4- DEPLOY AN APPLICATION

We now have an ArgoCD fully deployed, we will now deploy an application (ecsdemo-nodejs).

- **Fork application repository**

First step is to create a fork for the Github application we will deploy.

Login to github, go to: https://github.com/brentley/ecsdemo-nodejs.git and Fork the repo. This URL will be needed when we will configure the application into ArgoCD.

- **Create application**

Connect with ArgoCD CLI using our cluster context:

```
> CONTEXT_NAME=`kubectl config view -o jsonpath='{.current-context}'`
> argocd cluster add $CONTEXT_NAME
```

ArgoCD provides multicluster deployment functionalities. For the purpose of this workshop we will only deploy on the local cluster.

Configure the application and link to your fork (replace the GITHUB_USERNAME):

```
> kubectl create namespace ecsdemo-nodejs
> argocd app create ecsdemo-nodejs --repo
https://github.com/GITHUB_USERNAME/ecsdemo-nodejs.git --path kubernetes
--dest-server https://kubernetes.default.svc --dest-namespace ecsdemo-nodejs
```

Application is now setup, let’s have a look at the deployed application state:

```
> argocd app get ecsdemo-nodejs
```

You should have this output:

```
Health Status:      Missing

GROUP       KIND            NAMESPACE       NAME            STATUS     HEALTH   HOOK  MESSAGE
_           Service         ecsdemo-nodejs  ecsdemo-nodejs  OutOfSync  Missing
apps        Deployment      default         ecsdemo-nodejs  OutOfSync  Missing
```

We can see that the application is in an `OutOfSync` status since the application has not been deployed yet. We are now going to `sync` our application:

```
> argocd app sync ecsdemo-nodejs
```

After a couple of minutes our application should be synchronized.

```
GROUP  KIND        NAMESPACE       NAME            STATUS  HEALTH   HOOK  MESSAGE
_      Service     ecsdemo-nodejs  ecsdemo-nodejs  Synced  Healthy        service/ecsdemo-nodejs created
apps   Deployment  default         ecsdemo-nodejs  Synced  Healthy        deployment.apps/ecsdemo-nodejs created
```

5- UPDATE THE APPLICATION

Our application is now deployed into our ArgoCD. We are now going to update our github repository synced with our application

- **Update your application**

Go to your Github fork repository:

Update `spec.replicas: 2` in `ecsdemo-nodejs/kubernetes/deployment.yaml`

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ecsdemo-nodejs
  labels:
    app: ecsdemo-nodejs
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: ecsdemo-nodejs
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: ecsdemo-nodejs
    spec:
      containers:
      - image: brentley/ecsdemo-nodejs:latest
        imagePullPolicy: Always
        name: ecsdemo-nodejs
        ports:
        - containerPort: 3000
          protocol: TCP
```

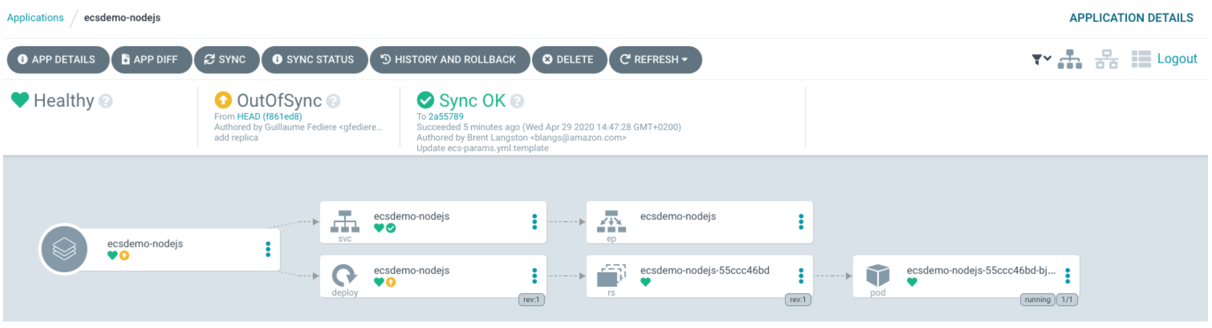
Add a commit message and click on `Commit changes`

- ## Access ArgoCD Web Interface

To deploy our change we can access to ArgoCD UI. Open your web browser and go to the Load Balancer url:

```
> echo $ARGOCD_SERVER
```

Login using `admin` / `$ARGO_PWD`. You now have access to the ecsdemo-nodejds application. After clicking to `refresh` button status should be `OutOfSync`:

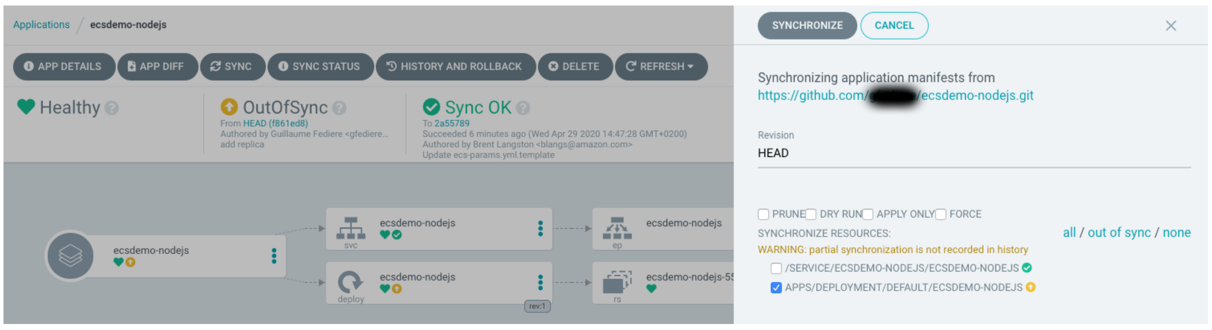


Healthy

OutOfSync
From HEAD (f861ed8)
Authored by Guillaume Fediere <gfediere...
add replica

Sync OK
To 2a55789
Succeeded 5 minutes ago (Wed Apr 29 2020 14:47:28 GMT+0200)
Authored by Brent Langston <blangs@amazon.com>
Update ecs-params.yml.template

This means our Github repository is not synchronised with the deployed application. To fix this and deploy the new version (with 2 replicas) click on the `sync` button, and select the `APPS/DEPLOYMENT/DEFAULT/ECSDEMO-NODEJS` and `SYNCHRONIZE`:



Healthy

OutOfSync
From HEAD (f861ed8)
Authored by Guillaume Fediere <gfediere...
add replica

Sync OK
To 2a55789
Succeeded 6 minutes ago (Wed Apr 29 2020 14:47:28 GMT+0200)
Authored by Brent Langston <blangs@amazon.com>
Update ecs-params.yml.template

Synchronizing application manifests from
https://github.com/[redacted]/ecsdemo-nodejs.git

Revision
HEAD

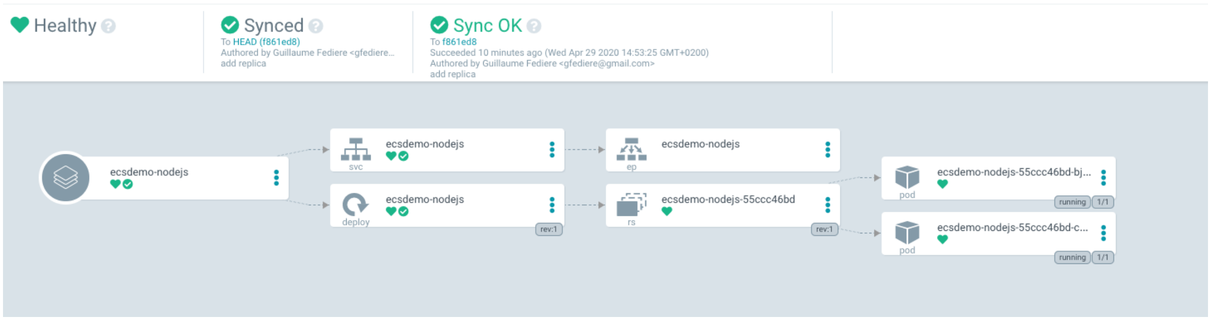
☐ PRUNE ☐ DRY RUN ☐ APPLY ONLY ☐ FORCE

SYNCHRONIZE RESOURCES: all / out of sync / none

WARNING: partial synchronization is not recorded in history

☐ /SERVICE/ECSDEMO-NODEJS/ECSDEMO-NODEJS
☑ APPS/DEPLOYMENT/DEFAULT/ECSDEMO-NODEJS

After the sync completed our application should have the `Synced` status with 2 pods:



Healthy

Synced
To HEAD (f861ed8)
Authored by Guillaume Fediere <gfediere...
add replica

Sync OK
To f861ed8
Succeeded 10 minutes ago (Wed Apr 29 2020 14:53:25 GMT+0200)
Authored by Guillaume Fediere <gfediere@gmail.com>
add replica

All those actions could have been made with the Argo CLI also.

- **CLEANUP**

Congratulations on completing the Continuous Deployment with ArgoCD module.

This module is not used in subsequent steps, so you can remove the resources now, or at the end of the workshop:

```
> argocd app delete ecsdemo-nodejs -y
```

```
> watch argocd app get ecsdemo-nodejs
```

Wait until all ressources are cleared with this message:

```
FATA[0000] rpc error: code = NotFound desc = applications.argoproj.io
"ecsdemo-nodejs" not found
```

And then delete ArgoCD from your cluster:

```
> kubectl delete -n argocd -f
https://raw.githubusercontent.com/argoproj/argo-cd/v2.0.4/manifests/install.yaml
```

Delete namespaces created for this chapter:

```
> kubectl delete ns argocd
```

```
> kubectl delete ns ecsdemo-nodejs
```

You may also delete the cloned repository ecsdemo-nodejs within your GitHub account.