How to deploy a simple Nodejs web application to Kubernetes using Argocd and circleci.

GitOps modernizes software management and operations by allowing developers to declaratively manage infrastructure and code using a single source of truth, usually a Git repository. Many development teams and organizations have adopted GitOps procedures to improve the creation and delivery of software applications.

For a GitOps initiative to work, an orchestration system like Kubernetes is crucial. The number of incompatible technologies needed to develop software makes Kubernetes a key tool for managing infrastructure. Without Kubernetes, implementing infrastructure-as-code (IaC) procedures is inefficient or even impossible. Fortunately, the wide adoption of Kubernetes has enabled the creation of tools for implementing GitOps.

One of these tools, ArgoCD, is a Kubernetes-native continuous deployment (CD) tool. It can deploy code changes directly to Kubernetes resources by pulling it from Git repositories instead of an external CD solution. Many of these solutions support only push-based deployments. Using ArgoCD gives developers the ability to control application updates and infrastructure setup from an unified platform. It handles the latter stages of the GitOps process, ensuring that new configurations are correctly deployed to a Kubernetes cluster.

In this tutorial, you will learn how to deploy a Node.js application on Azure Kubernetes Service (AKS) using a CI/CD pipeline and ArgoCD.

Prerequisites

- Familiarity with Kubernetes concepts (Pods, Deployments, Services).
- Basic understanding of Docker and containerization.
- Experience with Git for version control.
- Access to a Kubernetes cluster for testing (you can use Minikube, kind, or a cloud provider's Kubernetes service).
- Argo CD and Argo Rollouts documentation: Familiarize yourself with the basics of these tools.

Task 1: Setup and Configuration

- Create a git hub repository

  https://github.com/deepanshu03091995/nodejs-argocd.git

- **Install Argo CD on Your Kubernetes Cluster**

Launching the Azure Kubernetes Service (AKS) cluster

- `az group create --name NodeRG --location eastus`
- `az aks create --resource-group NodeRG --name NodeCluster --node-count 2 --enable-addons http_application_routing`

Installing ArgoCD in the AKS Cluster

- `az aks get-credentials --resource-group NodeRG --name NodeCluster`
- `kubectl create namespace argocd`
- `kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml`
- `kubectl get all --namespace argocd`

Exposing the ArgoCD API server

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

Accessing the ArgoCD Web Portal

- **`kubectl get all --namespace argocd`**

To log into the portal, you will need the username and password. The username is set as `admin` by default.

To fetch the password, execute this command:

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

Configuring Kubernetes manifests for ArgoCD

To configure ArgoCD to deploy your application on Kubernetes, you will have to set up ArgoCD to connect the Git Repository and Kubernetes in a declarative way using YAML for configuration.

Apart from this method, you can also set up ArgoCD from the Web Portal or using the ArgoCD CLI. Because this tutorial is following GitOps principles, we are using the Git repository as the sole source of truth. Therefore the declarative method using YAML files works best.

One of the key features and capabilities of ArgoCD is to sync via manual or automatic policy for deployment of applications to a Kubernetes cluster.

To get started, create a directory named `argocd` in the root directory of the project. Create a new file in the new directory and name it `config.yaml`.

Automated Sync policy

ArgoCD has the ability to automatically sync an application when it detects differences between the desired manifests in Git, and the live state in the cluster.

A benefit of automatic sync is that CI/CD pipelines no longer need direct access to the ArgoCD API server to perform the deployment. Instead, the pipeline makes a commit and push to the Git repository with the changes to the manifests in the tracking Git repo.

If you want to set to the Automated Sync policy, you need to paste this in the `config.yaml`

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: aks-nodejs-argocd
  namespace: argocd
spec:
  destination:
    namespace: nodejs
    server: "https://kubernetes.default.svc"
  source:
    path: manifests
    repoURL: "https://github.com/deepanshu03091995/nodejs-argocd/"
    targetRevision: circleci-project-setup
  project: default
  syncPolicy:
    automated:
      prune: false
      selfHeal: false
```

1. **Install Argo Rollouts**: Install the Argo Rollouts controller in your Kubernetes cluster, following the official guide.
2. **Standard Installation Method:**
    1. **Create a new namespace for Argo Rollouts using the following command:**

kubectl create namespace argo-rollouts

2.
    3. **Apply the installation manifest using the following command:**

kubectl apply -n argo-rollouts -f [install.yaml](https://github.com/argoproj/argo-rollouts/releases/latest/download/install.yaml)

4.
3. **This will set up the Argo Rollouts controller in the `argo-rollouts` namespace1.**
4. **Namespace-Level Installation (for running multiple instances in different namespaces):**
    1. **Create a separate namespace for Argo Rollouts (if you haven't already):**

kubectl create namespace argo-rollouts

2.
    3. **Install Argo Rollouts in the namespace using the following command:**
    4. **kubectl apply -n argo-rollouts -f [namespace-install.yaml](https://github.com/argoproj/argo-rollouts/manifests/crds)**

Task 2: Creating the GitOps Pipeline

1. **Dockerize the Application**: Build a Docker image for the web application of your choice and push it to a public container registry of your choice.

Dockerfile

```
# Set the base image to use for subsequent instructions
FROM node:alpine
# Set the working directory for any subsequent ADD, COPY, CMD, ENTRYPOINT,
# or RUN instructions that follow it in the Dockerfile
WORKDIR /usr/src/app
# Copy files or folders from source to the dest path in the image's filesystem.
COPY package.json /usr/src/app/
COPY . /usr/src/app/
# Execute any commands on top of the current image as a new layer and commit the results.
RUN npm install --production
# Define the network ports that this container will listen to at runtime.
EXPOSE 1337
# Configure the container to be run as an executable.
ENTRYPOINT ["npm", "start"]
```

Deploy the Application Using Argo CD:

- Modify the Kubernetes manifests in your forked repository to use the Docker image you pushed.
- Set up Argo CD to monitor your repository and automatically deploy changes to your Kubernetes cluster.

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: aks-nodejs-argocd
  namespace: argocd
spec:
  destination:
    namespace: nodejs
    server: "https://kubernetes.default.svc"
  source:
    path: manifests
    repoURL: "https://github.com/deepanshu03091995/nodejs-argocd/"
    targetRevision: circleci-project-setup
  project: default
  syncPolicy:
    automated:
      prune: false
      selfHeal: false
```

Finally creating .circleci workflow to automate the build and deployment on kubernetes using .circleci/config file