Argo CD

Argo CD (Continuous Delivery) is a declarative, GitOps continuous delivery tool for Kubernetes. It automates the deployment of applications to Kubernetes clusters by synchronizing the desired state defined in Git repositories with the actual state in the cluster. This approach ensures that the application state is always consistent with the version control system, providing benefits such as increased visibility, reproducibility, and automation in the deployment process.

Key Concepts of Argo CD

1. GitOps

GitOps is a methodology that uses Git as the single source of truth for declarative infrastructure and applications. In the context of Argo CD, GitOps involves:

- Storing Kubernetes manifests (YAML files) in a Git repository.
- Using these manifests to define the desired state of applications and environments.
- Automatically synchronizing the live state of the Kubernetes cluster with the desired state defined in Git.

2. Declarative Approach

Argo CD operates on the principle of declarative configurations, meaning that the desired state of the application (its configuration, resources, and dependencies) is defined in configuration files stored in a Git repository. Argo CD continuously monitors the repository and applies changes to the Kubernetes cluster to match the desired state.

3. Continuous Delivery

Argo CD provides automated delivery of applications to Kubernetes, ensuring that any changes pushed to the Git repository are reflected in the cluster. This automation streamlines the deployment process and reduces manual intervention.

Core Components of Argo CD

1. Repositories

Git repositories containing application definitions and configurations (Helm charts, Kustomize files, or plain Kubernetes manifests). Argo CD pulls the desired state from these repositories.

2. Applications

An application in Argo CD is a collection of Kubernetes resources defined in a Git repository. It includes metadata such as the repository URL, path within the repository, target cluster, and namespace.

3. Projects

Projects are a way to group and manage multiple applications. They allow for the definition of access controls, resource quotas, and policies that apply to all applications within the project.

4. Synchronization

Argo CD continuously compares the live state of the cluster with the desired state defined in the Git repository. When discrepancies are detected, Argo CD can automatically or manually synchronize the live state to match the desired state.

Features of Argo CD

1. Self-Healing

Argo CD can detect when the actual state of the application diverges from the desired state and can automatically correct these discrepancies to ensure consistency.

2. Multi-Cluster Support

Argo CD can manage applications deployed across multiple Kubernetes clusters, providing a unified view and control over all deployments.

3. Declarative Setup

Both the application and Argo CD's configuration itself can be managed declaratively, allowing for version control and reproducibility of the entire setup.

4. Access Control

Argo CD provides robust Role-Based Access Control (RBAC) to manage permissions for different users and teams.

5. Web UI and CLI

Argo CD provides a user-friendly web UI and a powerful CLI to manage applications, view logs, and perform sync operations.

6. Auditing and History

Argo CD maintains a history of application deployments, rollbacks, and sync operations, providing audit trails and the ability to revert to previous states if necessary.

Typical Workflow with Argo CD

1. **Define Application Manifests**: Store your Kubernetes manifests, Helm charts, or Kustomize configurations in a Git repository.
2. **Connect Repository to Argo CD**: Create an application in Argo CD that points to the repository and specifies the path and target cluster/namespace.
3. **Monitor and Sync**: Argo CD continuously monitors the repository. When changes are detected, it synchronizes the live state of the cluster with the desired state.
4. **Manage Applications**: Use the Argo CD web UI or CLI to monitor application status, view diffs between desired and live states, and manage sync operations.

Advantages of Using Argo CD

- **Improved Deployment Automation**: Reduces manual intervention and errors in the deployment process.
- **Enhanced Visibility and Control**: Provides a centralized view of all applications and their states.
- **Version Control**: Keeps a history of all changes, enabling easy rollbacks and audits.
- **Consistency and Reproducibility**: Ensures that deployments are consistent with the configurations defined in the Git repository.
- **Scalability**: Supports multi-cluster deployments and large-scale application management.

Conclusion

Argo CD is a powerful tool that leverages the principles of GitOps to provide continuous delivery and automated deployment for Kubernetes applications. By maintaining a single source of truth in a Git repository and continuously synchronizing the desired state with the actual state of the cluster, Argo CD enhances deployment reliability, reduces errors, and improves overall application management. Its robust feature set, including self-healing, multi-cluster support, and access control, makes it an essential tool for modern DevOps practices in Kubernetes environments.

Argo CD with an Amazon EKS

Setting up Argo CD with an Amazon EKS (Elastic Kubernetes Service) cluster involves several steps.

Prerequisites

1. **Amazon EKS Cluster**: Ensure you have an EKS cluster up and running.
2. **kubectl**: Make sure **kubectl** is installed and configured to access your EKS cluster.
3. **AWS CLI**: Ensure you have the AWS CLI installed and configured with the necessary permissions.

Step-by-Step:

1. Install Argo CD in Your EKS Cluster

1. **Create a Namespace for Argo CD**:

```
kubectl create namespace argocd
```

2. **Install Argo CD**: Use the following command to install Argo CD using the provided YAML manifests:

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

3. **Verify the Installation**: Check the status of the Argo CD pods to ensure they are running:

```
kubectl get pods -n argocd
```

```
[root@ip-172-31-55-218 ~]# eksctl get cluster
NAME            REGION          EKSCTL CREATED
rjcluster       us-east-1       True
[root@ip-172-31-55-218 ~]# kubectl create namespace argocd
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
namespace/argocd created
customresourcedefinition.apiextensions.k8s.io/applications.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/applicationsets.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/appprojects.argoproj.io created
serviceaccount/argocd-application-controller created
serviceaccount/argocd-applicationset-controller created
serviceaccount/argocd-dex-server created
serviceaccount/argocd-notifications-controller created
serviceaccount/argocd-redis created
serviceaccount/argocd-repo-server created
serviceaccount/argocd-server created
role.rbac.authorization.k8s.io/argocd-application-controller created
role.rbac.authorization.k8s.io/argocd-applicationset-controller created
role.rbac.authorization.k8s.io/argocd-dex-server created
role.rbac.authorization.k8s.io/argocd-notifications-controller created
role.rbac.authorization.k8s.io/argocd-redis created
role.rbac.authorization.k8s.io/argocd-server created
clusterrole.rbac.authorization.k8s.io/argocd-application-controller created
clusterrole.rbac.authorization.k8s.io/argocd-applicationset-controller created
clusterrole.rbac.authorization.k8s.io/argocd-server created
rolebinding.rbac.authorization.k8s.io/argocd-application-controller created
rolebinding.rbac.authorization.k8s.io/argocd-applicationset-controller created
rolebinding.rbac.authorization.k8s.io/argocd-dex-server created
rolebinding.rbac.authorization.k8s.io/argocd-notifications-controller created
rolebinding.rbac.authorization.k8s.io/argocd-redis created
rolebinding.rbac.authorization.k8s.io/argocd-server created
clusterrolebinding.rbac.authorization.k8s.io/argocd-application-controller created
clusterrolebinding.rbac.authorization.k8s.io/argocd-applicationset-controller created
clusterrolebinding.rbac.authorization.k8s.io/argocd-server created
configmap/argocd-cm created
configmap/argocd-cmd-params-cm created
configmap/argocd-gpg-keys-cm created
configmap/argocd-notifications-cm created
configmap/argocd-rbac-cm created
configmap/argocd-ssh-known-hosts-cm created
```

```
[root@ip-172-31-55-218 ~]# kubectl get pods -n argocd
NAME                                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0                     0/1     Pending   0          47m
argocd-applicationset-controller-769fd8675f-wrsgs   1/1     Running   0          47m
argocd-dex-server-746ff5b56b-57pwg                  1/1     Running   0          47m
argocd-notifications-controller-6664d7b8b7-gh5q5    1/1     Running   0          47m
argocd-redis-556f94f6b5-tlk6j                       1/1     Running   0          47m
argocd-repo-server-6ff4565fdc-8s697                 1/1     Running   0          47m
argocd-server-84457695fb-98bgg                      1/1     Running   0          47m
```

2. Expose the Argo CD Server

Argo CD server needs to be accessible outside the cluster. There are several ways to do this, such as using a LoadBalancer service or port forwarding.

Using LoadBalancer Service:

1. Edit the Argo CD server service to type **LoadBalancer**:

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

```
[root@ip-172-31-55-218 ~]# kubectl get svc argocd-server -n argocd
NAME            TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
argocd-server   NodePort   10.100.207.187   <none>        80:31129/TCP,443:30282/TCP   143m
[root@ip-172-31-55-218 ~]# kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'
service/argocd-server patched
[root@ip-172-31-55-218 ~]# kubectl get svc argocd-server -n argocd
NAME            TYPE           CLUSTER-IP      EXTERNAL-IP                                                              PORT(S)
argocd-server   LoadBalancer   10.100.207.187   ae959cc8ac83f45628d9b72de69a72ca-978037298.us-east-1.elb.amazonaws.com   80:31129/TCP,443:30282/TCP
```

2. Get the external IP of the Argo CD server:

```
kubectl get svc argocd-server -n argocd
```

Wait until the **EXTERNAL-IP** is assigned. This can take a few minutes.

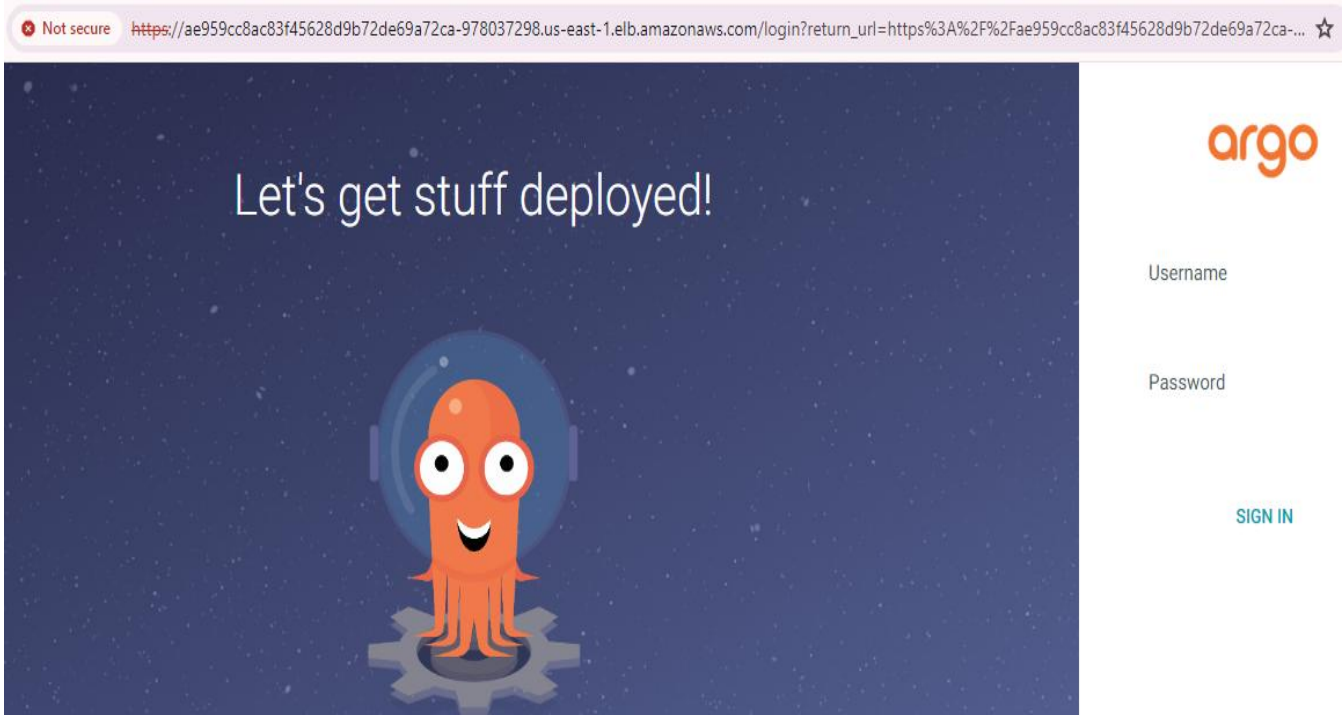
3. Log in to Argo CD

1. **Get the Initial Admin Password**:

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

```
[root@ip-172-31-55-218 ~]# kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d ; echo
QgnQuNO-RjLOEnzr
```

2. **Open the Argo CD UI**:
    - Access the Argo CD UI using the **EXTERNAL-IP** (for LoadBalancer)
    - Log in with the username **admin** and the password retrieved in the previous step.



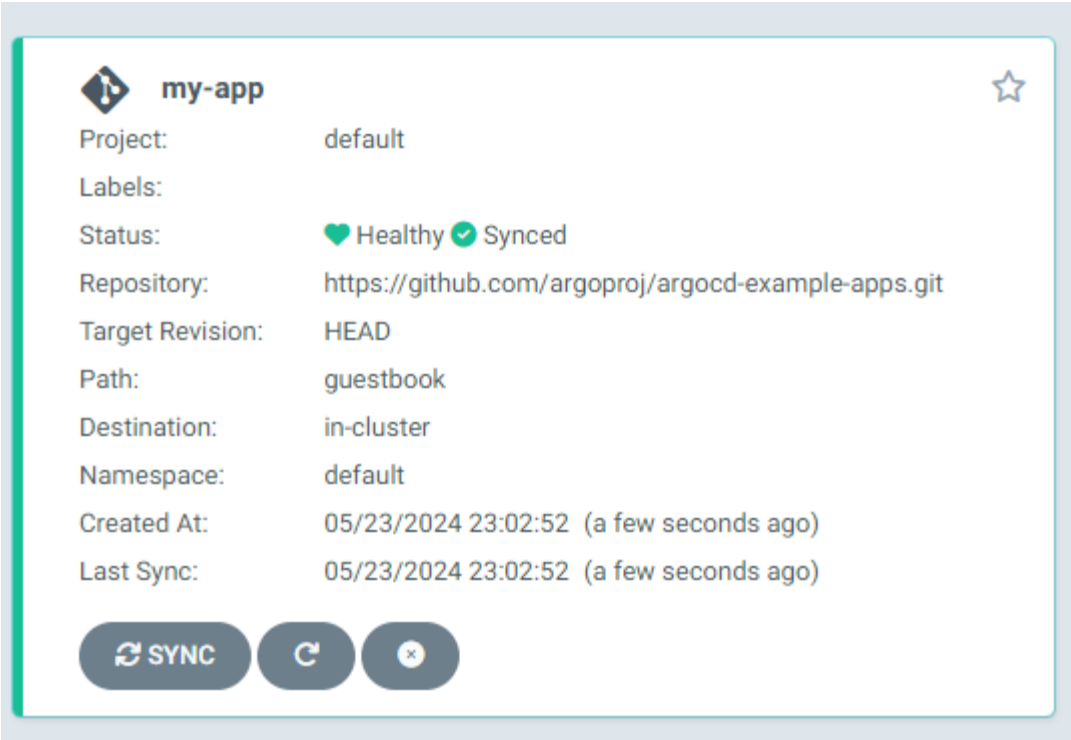
4. Create and Sync an Application

1. Once logged in, click the **+ New App** button at the top of the Argo CD dashboard.
2. Fill out the application details in the "General" section:
    - **Application Name**: **my-app** (or your preferred name)
    - **Project**: **default**
    - **Sync Policy**: Leave as Manual for now (you can change it to Automatic later if desired)
3. Fill out the "Source" section:
    - **Repository URL**: Enter the URL of the Git repository containing your application manifests. For example, **https://github.com/argoproj/argocd-example-apps.git**.
    - **Revision**: Leave as **HEAD** or specify a branch/tag if needed.
    - **Path**: Enter the path within the repository where the Kubernetes manifests are located. For example, **guestbook**.

4. Fill out the "Destination" section:
    - **Cluster URL**: Leave as **https://kubernetes.default.svc** (this targets the current cluster).
    - **Namespace**: Specify the namespace where you want the application to be deployed. For example, **default**.
5. Click the **Create** button at the bottom of the page.

4. Sync the Application

1. After the application is created, it will appear in the Argo CD dashboard with a status of **OutOfSync**.
2. To sync the application, click on the application name (**my-app**).
3. Click the **Sync** button at the top right of the application details page.
4. In the sync dialog, review the resources to be synchronized and click **Synchronize** to start the sync process.
5. The status will change to **Healthy** once the sync is complete and the application is successfully deployed.



```
[root@ip-172-31-55-218 ~]# kubectl get all
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   11h
[root@ip-172-31-55-218 ~]# kubectl get all
NAME                                 READY   STATUS    RESTARTS   AGE
pod/guestbook-ui-56c646849b-8flv4    1/1     Running   0          107s

NAME                   TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/guestbook-ui   ClusterIP   10.100.229.44   <none>        80/TCP    107s
service/kubernetes     ClusterIP   10.100.0.1      <none>        443/TCP   11h

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/guestbook-ui   1/1     1            1           107s

NAME                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/guestbook-ui-56c646849b   1         1         1       107s
[root@ip-172-31-55-218 ~]#
```