**ARGO CD – What, Why and How?**

Argo CD is a tool for Kubernetes that automates the deployment of applications. It follows the GitOps pattern, which means your GitHub repository is the "single source of truth." Argo CD continuously monitors your repository and ensures that the live application running in your cluster matches the configuration defined in your Git repo.

Think of it as a vigilant manager that watches your blueprints (your Git repo) and automatically builds or fixes your building (your application) to match them perfectly.

**Step 1: Prerequisites**

Before you start, you'll need a few things ready:

1. **A Kubernetes Cluster:** This can be a local one like Minikube or Docker Desktop, or a cloud-based one (GKE, EKS, AKS).
2. **kubectl Installed:** The Kubernetes command-line tool, configured to connect to your cluster.
3. **A GitHub Repository:** This repo must contain the Kubernetes manifest files (e.g., deployment.yaml, service.yaml) for the application you want to deploy.

Step 2: Install Argo CD on Your Cluster

First, you'll create a dedicated namespace for Argo CD and apply the official installation manifest.

1. Create a namespace for Argo CD:

Bash

kubectl create namespace argocd

1. Apply the installation YAML: This command downloads the official manifest and uses it to create all the necessary Kubernetes resources (Deployments, Services, CRDs, etc.) for Argo CD.

Bash

kubectl apply -n argocd -f <https://raw.githubusercontent.com/argoproj/argo-cd/st>

Service Type Load Balancer

Change the argocd-server service type to LoadBalancer:

bash

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

After a short wait, your cloud provider will assign an external IP address to the service. You can retrieve this IP with:

bash

kubectl get svc argocd-server -n argocd o=jsonpath='{.status.loadBalancer.ingress[0].ip}'

**Step 3: Access the Argo CD UI**

Argo CD is managed through a web UI or its CLI. The easiest way to access the UI for the first time is by using port-forwarding.

1. **Forward the Argo CD server service** to your local machine. This command will run continuously in your terminal.

Bash

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

1. **Get the initial admin password**. The password is automatically generated and stored in a Kubernetes secret.

Bash

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

Copy the password that is printed to your console. 🔑

1. **Log in to the UI**. Open a web browser and go to **https://localhost:8080**. You'll see a security warning; you can safely proceed. Log in with the username **admin** and the password you just copied.
2. Creating Apps Via CLI

First we need to set the current namespace to argocd running the following command:

kubectl config set-context --current --namespace=argocd

Create the example guestbook application with the following command:

argocd app create guestbook --repo https://github.com/argoproj/argocd-example-apps.git --pa

**Step 4: Connect Your GitHub Repository**

Now, you need to tell Argo CD where your application's source code and manifests are located.

1. In the Argo CD UI, go to **Settings** (the gear icon ⚙️ on the left).
2. Select **Repositories**.
3. Click **CONNECT REPO USING HTTPS**.
4. Fill in the details:
   * **Type**: Select git.
   * **Repository URL**: Enter the HTTPS URL of your GitHub repository (e.g., https://github.com/your-username/your-repo.git).
   * **Username / Password**:
     + If it's a **public repository**, you can leave these blank.
     + If it's a **private repository**, enter your GitHub username and a **Personal Access Token (PAT)** in the password field. **Do not use your actual GitHub password.** You can generate a PAT in GitHub under *Settings > Developer settings > Personal access tokens* with the repo scope.
5. Click **Connect**. Argo CD will verify the connection.

**Step 5: Create the Argo CD Application**

This is the final step where you define your application and link it to the manifests in your GitHub repo.

1. On the main page, click **+ NEW APP**.
2. Fill out the application creation form with the following details:
   * **Application Name**: A unique name for your application (e.g., my-cool-app).
   * **Project Name**: You can leave this as default.
   * **Sync Policy**:
     + **Manual**: You will have to click a "Sync" button in the UI to deploy changes.
     + **Automatic**: Argo CD will automatically deploy changes when it detects a new commit in your repository's target branch. For production, you might want to enable Prune Resources (to delete old components) and Self Heal (to correct any manual changes).
3. **Source** section:
   * **Repository URL**: Select the GitHub repository you connected in the previous step.
   * **Revision**: The branch to track (e.g., main, master, or HEAD).
   * **Path**: The path to the folder within your repository that contains the Kubernetes manifest files (e.g., k8s or manifests).
4. **Destination** section:
   * **Cluster URL**: Select https://kubernetes.default.svc. This means you are deploying to the same cluster where Argo CD is running.
   * **Namespace**: The namespace where you want your application to be deployed (e.g., default, production). This namespace must exist in your cluster.
5. Click **CREATE** at the top of the page.

**Step 6: Sync and Verify the Deployment**

Your application is now configured but has not yet been deployed.

1. You will see your application on the dashboard with a status of **Missing** and **OutOfSync**. This is normal.
2. Click on your application card to see the detailed view.
3. Click the **SYNC** button. Review the resources that will be created and click **SYNCHRONIZE**.
4. Argo CD will now apply the manifests from your GitHub repository to your cluster. The status will change to **Synced** and **Healthy** once everything is running correctly.

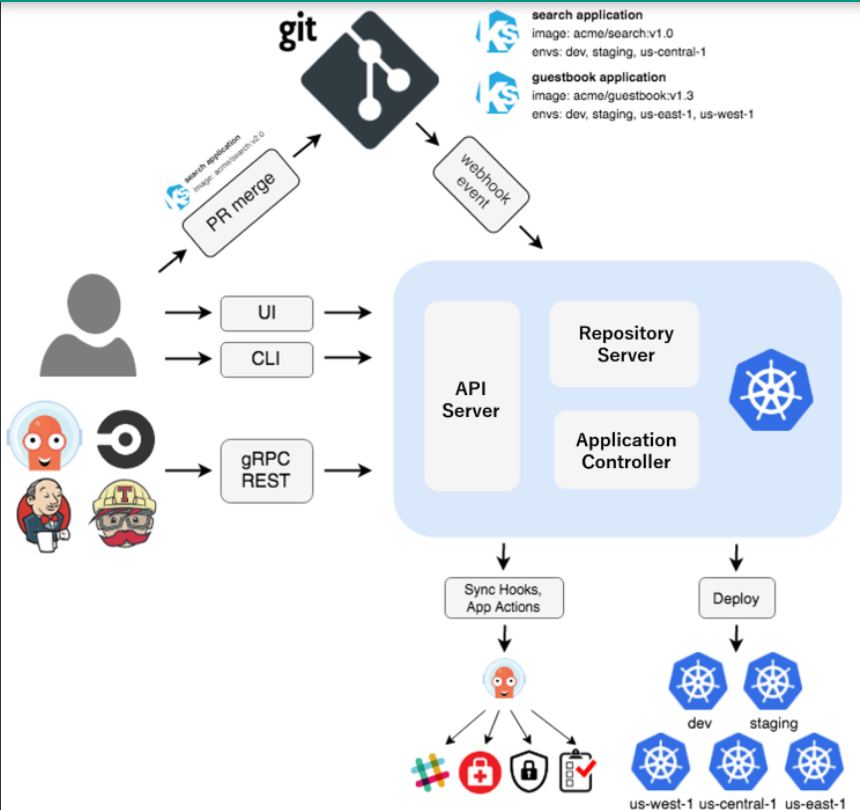
You can now use kubectl get all -n <your-namespace> in your terminal to verify that your application's pods, services, and other resources have been successfully deployed.

**WHY ARGOCD –**

Main Features of ArgoCD

* Declarative GitOps-based deployment model.
* Supports multiple Kubernetes clusters.
* Automatic synchronization of applications with Git repositories.
* Health status and history tracking of applications.
* Role-Based Access Control (RBAC) and Single Sign-On (SSO) integrations.
* Supports Helm charts, Kustomize, Jsonnet, and plain YAML manifests.
* User-friendly Web UI and CLI for managing deployments.

**Architecture –**

****

**Why to choose ARGOCD over anything else –**

Core Benefits of Argo CD

* Declarative, Git-Centric Workflow Argo CD uses Git as the single source of truth. You declare the desired state of your application in Git, and Argo CD ensures the cluster matches it. Other CI/CD tools, like Jenkins or GitLab CI, are often imperative, relying on custom scripts to execute a series of deployment steps.
* Pull-Based Model (Enhanced Security) The Argo CD agent runs inside the cluster and *pulls* changes from the repository. This is more secure because you don't need to grant external CI/CD systems direct administrative access to your cluster. Traditional CI/CD services use a *push* model, where the CI server actively pushes changes into the cluster, requiring sensitive credentials to be stored and managed outside.
* Automated Drift Detection and Remediation Argo CD constantly monitors your application's live state. If it detects any manual changes or deviations ("drift") from the state defined in Git, it can automatically correct them or alert you. This self-healing capability is a core feature not typically found in other CI/CD tools, which are often unaware of the application's state after a deployment script finishes.
* Deep Kubernetes Visibility The Argo CD UI provides a rich, real-time visualization of your application's components, health status, and sync state. It understands Kubernetes objects natively. In contrast, other CI/CD services usually just show you a log output of a script, offering little insight into the actual health and status of the deployed application.