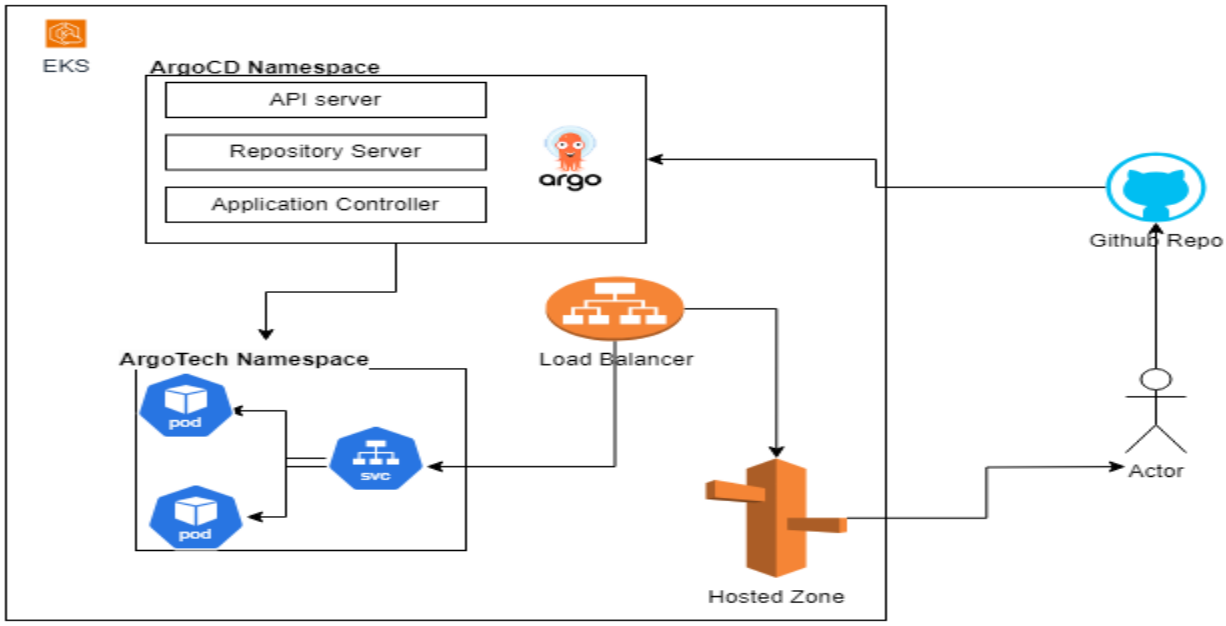
ArgoCD

ArgoCD is a declarative, GitOps-based continuous delivery tool for Kubernetes. It automates the deployment and synchronization of application states defined in a Git repository to Kubernetes clusters. It ensures that the cluster state matches the desired state described in Git

Simple Workflow



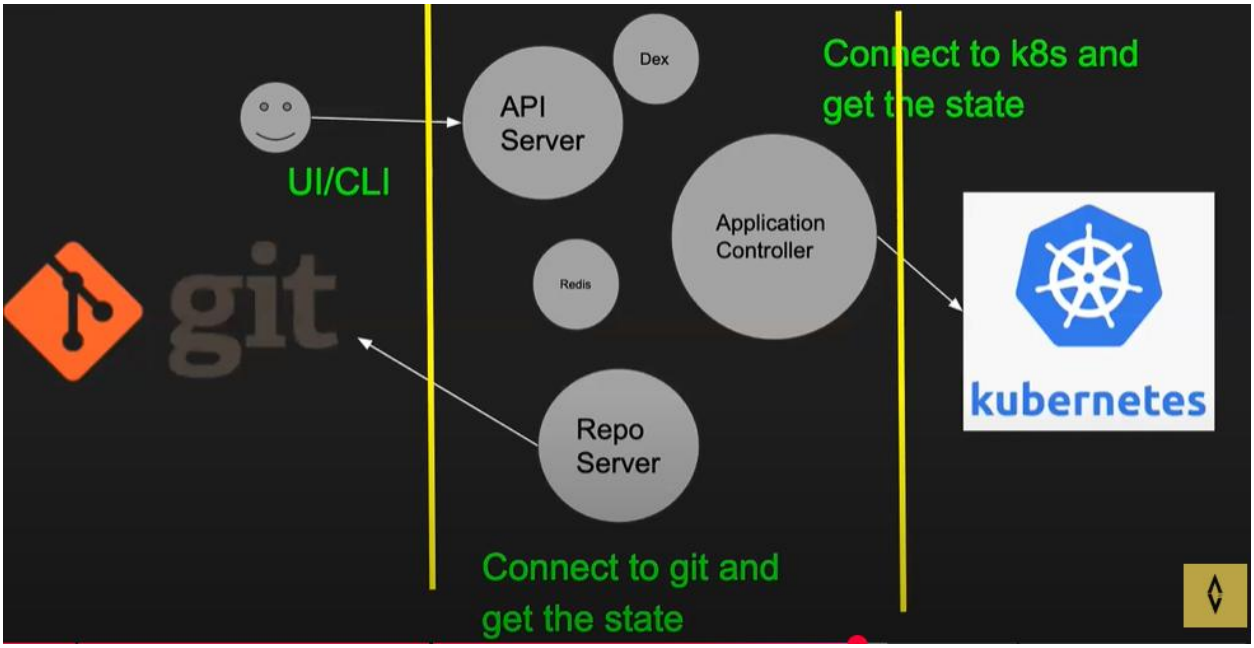
Advantages of ArgoCD:

- **Security:** Enhanced security through declarative configuration and audit trails.
- **Versioning (Track of Changes):** Full version control of infrastructure and application configurations, enabling easy rollback and history tracking.
- **Auto Upgrades:** Automated deployments and updates based on Git commits.
- **Auto Healing of Any Unwanted Changes:** Self-healing infrastructure that automatically reverts unwanted changes to the desired state defined in Git.

History of the Argo Project:

- Created by engineers at Applatix.
- Open-Source project.
- Applatix was acquired by Intuit.
- Actively contributed to by Akuity, Black Rock, CodeFresh, Intuit, and Red Hat.

Architecture:



Main components of ArgoCD server: -

API Server: - The ArgoCD API server is a critical component that exposes a REST API for communication with the Web UI, CLI, and CI/CD systems.

Its main responsibilities include:

- **Application Management and Status Reporting:** It manages the lifecycle of applications by allowing creation, updates, and deletion. It also monitors the application's status and ensures it matches the desired state defined in the Git repository.
- **Application Operations:** Supports operations like syncing applications to their desired state, rolling back to previous versions, and triggering user-defined actions.

These operations ensure that Kubernetes resources are always in sync with the Git configuration**.**

- **Repository and Cluster Credential Management:** Manages credentials for Git repositories and Kubernetes clusters. These credentials are securely stored as Kubernetes secrets, ensuring secure access to external systems.
- **Authentication and Identity Provider Integration:** Provides authentication mechanisms and integrates with external identity providers like LDAP, SAML, OAuth2, and OpenID Connect for seamless user management.
- **Webhook Event Listener and Forwarder:** Acts as a listener for Git webhook events (e.g., repository changes). These events are processed and converted into ArgoCD actions, such as triggering an application sync.

Repository Server: The Repository Server in ArgoCD is an internal service responsible for handling the Git repository that stores application manifests.

Its key functions include:

- **Maintaining a Local Cache:** It keeps a local copy of the Git repository to ensure faster access and processing of manifests.
- **Generating Kubernetes Manifests:** It generates Kubernetes manifests by using the following inputs:
    - **Repository URL:** The Git repository's location.
    - **Revision:** Specifies the commit, tag, or branch to use.
    - **Application Path:** Points to the directory in the repository where manifests or configurations are stored.
    - **Template-Specific Settings:** Includes parameters like Helm's **values.yaml** file and other configurations specific to templating tools**.**

Application Controller: The Application Controller in ArgoCD is a Kubernetes controller with the following responsibilities:

1. **Monitoring Applications:** Continuously monitors the live state of running applications in the cluster.
2. **State Comparison:** Compares the current, live state of applications with the desired state defined in the Git repository.

3. **Detecting OutOfSync State:** Identifies when an application's live state is OutOfSync with its desired state.
4. **Corrective Actions:** Optionally takes corrective actions to bring the application back into sync with the desired state.
5. **Lifecycle Hooks:** Invokes user-defined hooks at specific lifecycle stages:
    - PreSync: Actions before syncing the application.
    - Sync: Actions during the syncing process.
    - PostSync: Actions after the sync is complete.

Redis: In ArgoCD, Redis is used as a caching layer to improve performance. It caches application states, Git metadata, and cluster states to reduce API calls and speed up operations. Redis also helps with rate-limiting, processing webhook events, and session management. Its in-memory nature ensures fast and efficient handling of data.

Dex is an open-source identity provider (IdP) used in ArgoCD for authentication. It acts as a bridge between ArgoCD and external identity providers, enabling Single Sign-On (SSO).

Key Features:

1. **Supports Multiple Identity Providers**:
   Works with LDAP, SAML, GitHub, Google, and others.
2. **OIDC Provider**:
   Dex provides OpenID Connect (OIDC) tokens that ArgoCD uses for user authentication.
3. **Authentication for ArgoCD**:
   Integrates external identity providers with ArgoCD, allowing users to log in using their existing credentials.
4. **Simplifies User Management**:
   Makes it easier to manage users by delegating authentication to external systems

Installing Argo CD in a Custom Namespace: -

Install Argo CD in a namespace other than the default ArgoCD, you can use Kubectl to apply a patch that updates the ClusterRoleBinding to reference the correct namespace for the ServiceAccount. This ensures that the necessary permissions are correctly set in your custom namespace

1)Install Argo CD in a custom namespace and ensure the correct permissions are set for the ServiceAccount, you can follow these steps:

kubectl create namespace argocd

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

2)Patch the ClusterRoleBinding to reference the correct namespace for the ServiceAccount:

After the installation, you'll need to patch the ClusterRoleBinding to update the Service Account's namespace.

kubectl patch clusterrolebinding argocd-application-controller \

--patch '{"subjects":[{"kind":"ServiceAccount","name":"argocd-application-controller","namespace":"argocd'

3) **Verify the installation**:

To verify if Argo CD is installed correctly, you can check the pods in your custom namespace:

This ensures that Argo CD's permissions are correctly set up for the ServiceAccount in your custom namespace.

kubectl get pods -n argocd

```
NAME                                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0                     1/1     Running   0          67s
argocd-applicationset-controller-5c64658ff9-v5v7k   1/1     Running   0          67s
argocd-dex-server-78659867d9-77q27                  1/1     Running   0          67s
argocd-notifications-controller-749b96fd4d-9ntnp    1/1     Running   0          67s
argocd-redis-74cb89f466-fqn97                       1/1     Running   0          67s
argocd-repo-server-69746cbd47-5r9cj                 1/1     Running   0          67s
argocd-server-6c87596fcf-5lk78                      1/1     Running   0          67s
```

kubectl get svc -n argocd

```
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   129m
abhishekveeramalla@aveerama-mac ~ % kubectl get svc -n argocd
NAME                                      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
argocd-applicationset-controller          ClusterIP   10.109.35.168   <none>        7000/TCP,8080/TCP            5m18s
argocd-dex-server                         ClusterIP   10.96.136.181   <none>        5556/TCP,5557/TCP,5558/TCP   5m18s
argocd-metrics                            ClusterIP   10.99.102.233   <none>        8082/TCP                     5m17s
argocd-notifications-controller-metrics   ClusterIP   10.109.251.95   <none>        9001/TCP                     5m17s
argocd-redis                              ClusterIP   10.99.168.235   <none>        6379/TCP                     5m17s
argocd-repo-server                        ClusterIP   10.99.113.244   <none>        8081/TCP,8084/TCP            5m17s
argocd-server                             ClusterIP   10.97.16.100    <none>        80/TCP,443/TCP               5m17s
argocd-server-metrics                     ClusterIP   10.98.22.110    <none>        8083/TCP                     5m17s
```

kubectl get secret -n argocd

```
argocd-initial-admin-secret   Opaque   1      114m
argocd-notifications-secret   Opaque   0      10m
argocd-secret                 Opaque   5      10m
```

Kubectl edit secret argocd-initial admin -n argocd

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  password: dllDWHNLTUZPRGVJOUNjSg==
kind: Secret
metadata:
  creationTimestamp: "2023-06-07T13:18:01Z"
  name: argocd-initial-admin-secret
  namespace: argocd
  resourceVersion: "6994"
  uid: 7acaa659-ef9a-4945-9492-3cc34ebc444d
type: Opaque
~
```

Decrypt the password: -

echo password | base64 –decode

We can login into UI with below username and password

Username: admin

Password: decode password

Let discuss more on how to create project and how to host application on multiple cluster together.

Then we can create a project