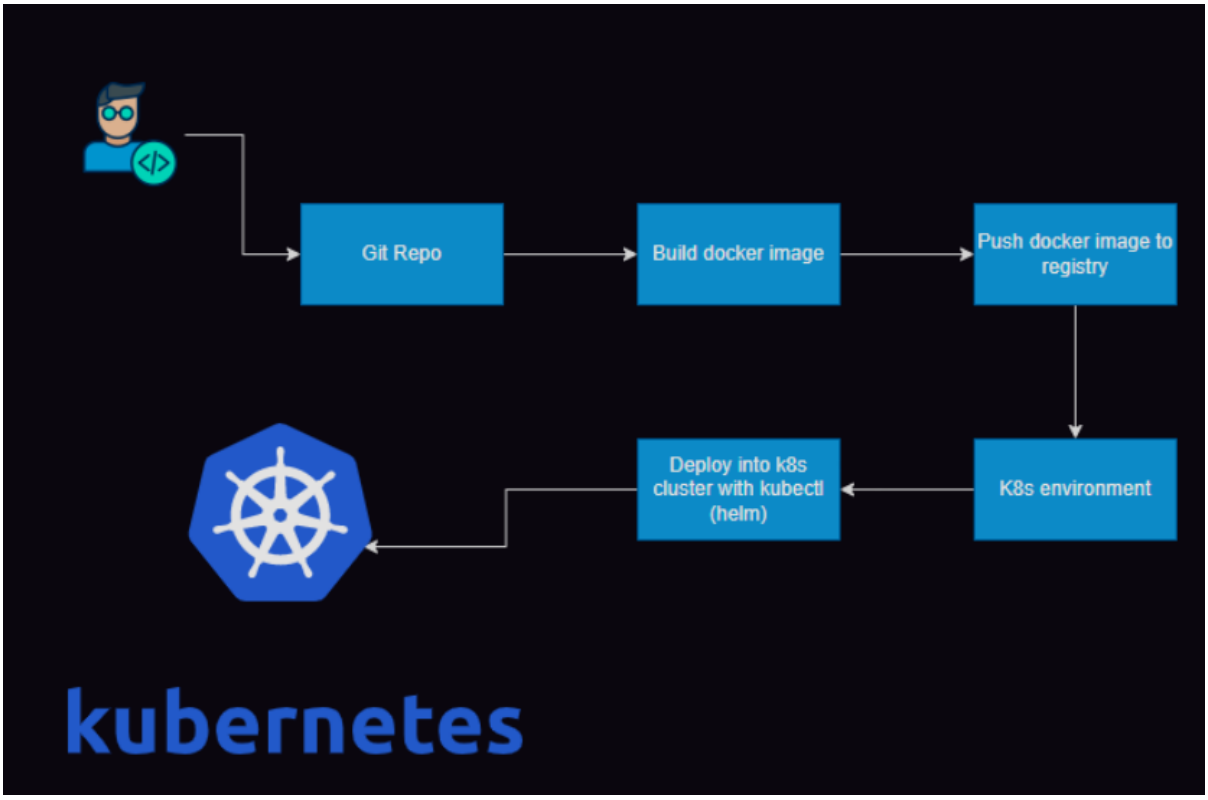
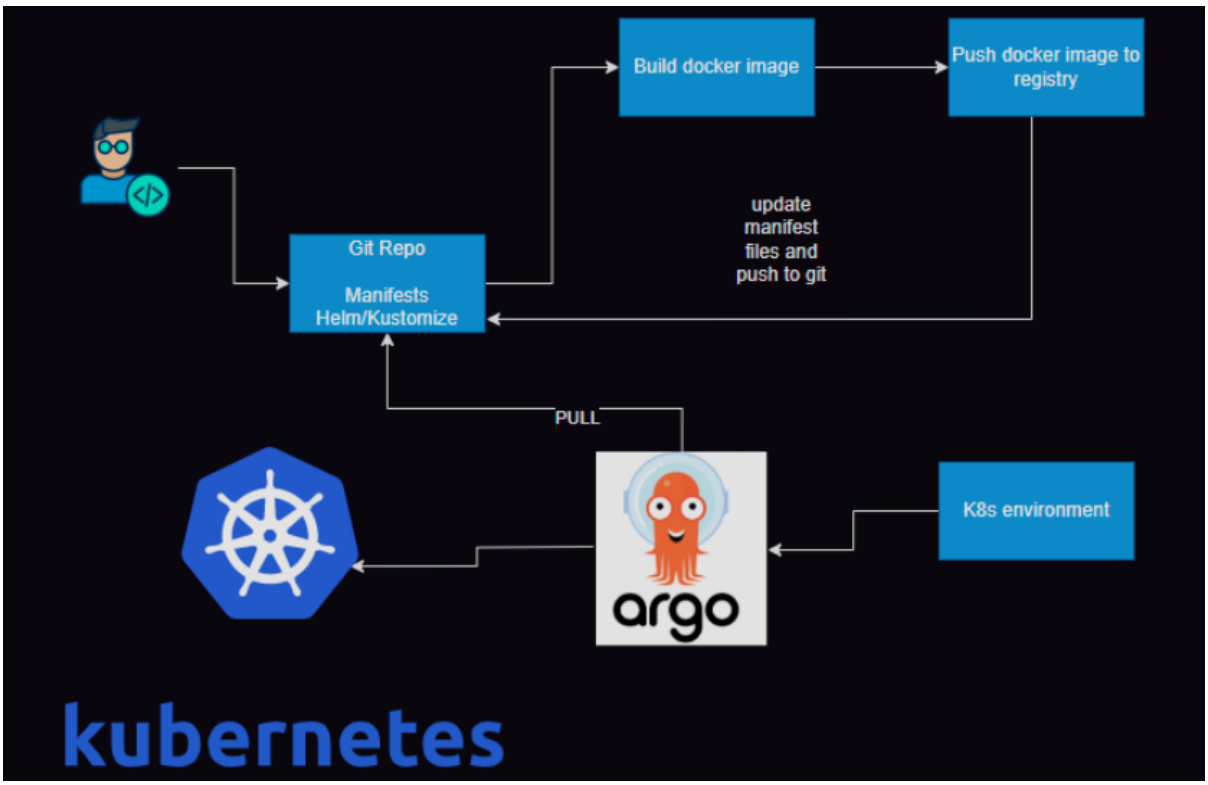
ArgoCD

Typical deployment into k8s



GitOps

- **Git is the single source for truth for Your Application as-well-as Infrastucture**.



Key Points

- **Argo cd can be configure with Git repo for**
    - **manifests**
    - **helm**
    - **kustomize**
- **To configure argo with git we can execute**
    - **commands**
    - **manifest files (yaml)**

Argo CD Tutorial

Argo CD Tutorial with a Kubernetes Manifest Example in a Git Repository

- Official Docs
- Git Repo Used in Class

Step 1: Set Up the Git Repository

1. **Create a Git Repository:**
   Create a new Git repository on your preferred Git hosting service (e.g., GitHub, GitLab).
2. **Add a Sample Kubernetes Manifest:**
   Clone your repository and add a sample Kubernetes manifest. Below is an example of a `guestbook` application.

Directory Structure:

```
my-repo/
├── manifests/
    ├── deployment.yaml
    ├── service.yaml
```

`deployment.yaml`:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: guestbook
  labels:
    app: guestbook
spec:
  replicas: 3
  selector:
    matchLabels:
      app: guestbook
```

```
  template:
    metadata:
      labels:
        app: guestbook
    spec:
      containers:
      - name: guestbook
        image: redis:alpine
        ports:
        - containerPort: 6379
```

`service.yaml`:

```
apiVersion: v1
kind: Service
metadata:
  name: guestbook
spec:
  selector:
    app: guestbook
  ports:
  - protocol: TCP
    port: 80
    targetPort: 6379
  type: ClusterIP
```

3. **Push the Files to Git:**

```
git add .
git commit -m "Add guestbook Kubernetes manifests"
git push origin main
```

Step 2: Install Argo CD

Follow the installation steps provided in the main tutorial:

1. **Create the namespace:**

```
kubectl create namespace argocd
```

2. **Install Argo CD:**

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

3. **Port-forward the service to access the UI:**

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

Step 3: Create the Argo CD Application

1. **Login to Argo CD:**
   Retrieve the admin password:

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

   Login via the CLI:

```
argocd login localhost:8080
# or
argocd login localhost:8080 --username admin --password <YOUR_PASSWORD> --
insecure
```

2. **Add Your Git Repository to Argo CD:**
   Replace `<GIT_REPO_URL>` with the URL of your repository:

```
argocd repo add <GIT_REPO_URL>
```

3. **Create an Argo CD Application:**
   Run the following command to create an application in Argo CD:

```
argocd app create guestbook \
  --repo <GIT_REPO_URL> \
  --path manifests \
  --dest-server https://kubernetes.default.svc \
  --dest-namespace default
  --sync-policy automated
```

   Example:

```
argocd app create guestbook \
  --repo https://github.com/your-username/my-repo.git \
  --path manifests \
```

```
  --dest-server https://kubernetes.default.svc \
  --dest-namespace default
```

Step 4: Sync the Application

1. **Sync the Application via CLI:**

```
argocd app sync guestbook
```

2. **Sync the Application via UI:**
   - Access the Argo CD web UI at `https://localhost:8080`.
   - Login using the admin credentials.
   - Click on the `guestbook` application.
   - Click the “Sync” button to deploy the application.

3. **Verify the Deployment:**
   Check the resources in the Kubernetes cluster:

```
kubectl get all -n default
```

Step 5: Monitor and Manage the Application

1. **Check Application Status:**

```
argocd app get guestbook
```

2. **View Logs:**
   To view logs of a specific pod:

```
kubectl logs <POD_NAME> -n default
```

3. **Update the Application:**
   - Modify the manifests in the Git repository.
   - Commit and push the changes.
   - Argo CD will detect the changes and show the application as `OutOfSync`.

4. **Sync Again:**

```
argocd app sync guestbook
```

Step 6: Enable Automatic Sync (Optional)

To enable auto-sync for the `guestbook` application:

```
argocd app set guestbook --sync-policy automated
```

Step 7: Clean Up

1. **To delete the application and resources:**

   ```
   argocd app delete guestbook
   ```

2. **To uninstall Argo CD:**

   ```
   kubectl delete namespace argocd
   ```

Here's the formatted version for your notes:

Argo CD Tutorial with Helm Chart Deployment

Step 1: Set Up the Git Repository with Helm Charts

Create a Git Repository:

- Create a new repository to store your Helm chart or use an existing one.

Add a Sample Helm Chart:

- Create a Helm chart using `helm create` or download an existing one.

Example directory structure for a Helm chart:

```
my-repo/
└── helm/
    └── guestbook/
        ├── Chart.yaml
        ├── values.yaml
```

```
        ├── templates/
        │   ├── deployment.yaml
        │   └── service.yaml
```

Chart.yaml:

```yaml
apiVersion: v2
name: guestbook
description: A sample Helm chart for a Kubernetes application
version: 1.0.0
appVersion: 1.0.0
```

values.yaml:

```yaml
replicas: 3
image:
  repository: redis
  tag: alpine
service:
  type: ClusterIP
  port: 6379
```

templates/deployment.yaml:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Chart.Name }}
  labels:
    app: {{ .Chart.Name }}
spec:
  replicas: {{ .Values.replicas }}
  selector:
    matchLabels:
      app: {{ .Chart.Name }}
  template:
    metadata:
      labels:
        app: {{ .Chart.Name }}
    spec:
      containers:
      - name: {{ .Chart.Name }}
        image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
        ports:
        - containerPort: {{ .Values.service.port }}
```

`templates/service.yaml`:

```
apiVersion: v1
kind: Service
metadata:
  name: {{ .Chart.Name }}
spec:
  type: {{ .Values.service.type }}
  ports:
  - port: 80
    targetPort: {{ .Values.service.port }}
  selector:
    app: {{ .Chart.Name }}
```

Push the Chart to Git:

```
git add .
git commit -m "Add Helm chart for guestbook application"
git push origin main
```

Step 2: Install Argo CD

Follow the steps in the Argo CD installation tutorial. Ensure Argo CD is installed and running in your Kubernetes cluster.

Step 3: Deploy the Helm Chart Using Argo CD

Add Your Git Repository to Argo CD:

```
argocd repo add <GIT_REPO_URL>
```

Create an Argo CD Application for the Helm Chart:
Replace `<GIT_REPO_URL>` and `<APP_NAME>` with your repository URL and application name.

```
argocd app create guestbook-helm \
--repo <GIT_REPO_URL> \
--path helm/guestbook \
--dest-server https://kubernetes.default.svc \
--dest-namespace default \
--helm-set replicas=3 \
--helm-set image.repository=redis \
--helm-set image.tag=alpine
```

Example:

```
argocd app create guestbook-helm \
--repo https://github.com/your-username/my-repo.git \
--path helm/guestbook \
--dest-server https://kubernetes.default.svc \
--dest-namespace default
```

Sync the Application:

```
argocd app sync guestbook-helm
```

Step 4: Use Helm Repository (Optional)

Add the Helm Repository:

```
argocd repo add https://charts.bitnami.com/bitnami --type helm
```

Create an Application Using a Helm Chart:

```
argocd app create guestbook-helm \
--repo https://charts.bitnami.com/bitnami \
--helm-chart redis \
--revision 17.4.0 \
--dest-server https://kubernetes.default.svc \
--dest-namespace default \
--helm-set replicaCount=3
```

Sync the Application:

```
argocd app sync guestbook-helm
```

Step 5: Monitor and Manage the Application

Check Application Status:

```
argocd app get guestbook-helm
```

View Application Details in the Web UI:

- Access the Argo CD web UI at `https://localhost:8080`.
- Navigate to the `guestbook-helm` application to see deployment details, status, and logs.

Update the Helm Values:

```
argocd app set guestbook-helm --helm-set replicas=5
argocd app sync guestbook-helm
```

Rollback to a Previous Version:

```
argocd app rollback guestbook-helm <REVISION_NUMBER>
```

Step 6: Enable Auto-Sync for Helm Applications

```
argocd app set guestbook-helm --sync-policy automated
```

Step 7: Clean Up

Delete the Helm application and resources:

```
argocd app delete guestbook-helm
```

Uninstall Argo CD:

```
kubectl delete namespace argocd
```

Argo CD Git repository configuration in yaml

- Example of an **Argo CD Git repository configuration** in YAML format.
- This configuration can be used to define a Git repository in Argo CD using a `ConfigManagementPlugin` or via the `Application` manifest.

Git Repository Configuration Using `Application` Manifest

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: example-app
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/your-org/your-repo.git
    targetRevision: HEAD  # Can be a branch, tag, or commit hash
    path: manifests/example-app
  destination:
    server: https://kubernetes.default.svc
    namespace: default
  syncPolicy:
    automated:
      prune: true          # Automatically delete resources that are no longer in
Git
      selfHeal: true       # Automatically sync out-of-sync resources
```

Key Fields Explained

1. `metadata.name`: The name of the Argo CD application.
2. `metadata.namespace`: The namespace where Argo CD is installed (default: `argocd`).
3. `spec.source.repoURL`: The URL of your Git repository.
4. `spec.source.targetRevision`: Specifies the Git branch, tag, or commit to track.
5. `spec.source.path`: The path in the repository where the manifests are located.
6. `spec.destination.server`: The API server URL for the Kubernetes cluster.
7. `spec.destination.namespace`: The namespace where the application will be deployed.
8. `spec.syncPolicy.automated`: Enables automatic synchronization.

Example for Multiple Applications

If you have multiple applications in a single repository, you can create separate `Application` manifests for each.

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: app1
  namespace: dev
spec:
  project: default
  source:
    repoURL: https://github.com/your-org/your-repo.git
    targetRevision: develop
    path: manifests/app1
```

```
  destination:
    server: https://kubernetes.default.svc
    namespace: app1
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
---
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: app2
  namespace: qa
spec:
  project: default
  source:
    repoURL: https://github.com/your-org/your-repo.git
    targetRevision: qa
    path: manifests/app2
  destination:
    server: https://kubernetes.default.svc
    namespace: app2
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

Apply the Manifest

Save the YAML to a file, e.g., `argocd-app.yaml`, and apply it using `kubectl`:

```
kubectl apply -f argocd-app.yaml
```

Optional: Add Credentials for a Private Git Repository

If your Git repository is private, you'll need to configure repository credentials in Argo CD:

```
argocd repo add https://github.com/your-org/your-private-repo.git \
  --username <USERNAME> \
  --password <PASSWORD>
```

Alternatively, create a `Secret` to store credentials:

```
apiVersion: v1
kind: Secret
```

```
metadata:
  name: repo-creds
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
data:
  url: aHR0cHM6Ly9naXRodWIuY29tL3lvdXItb3JnL3lvdXItcmVwby5naXQ=  # Base64 of repo
URL
  username: <BASE64_ENCODED_USERNAME>
  password: <BASE64_ENCODED_PASSWORD>
```

Well-Known Kubernetes Issues and How to Resolve Them

- K8s Troubleshooting Docs