**Process Summary for Creating and Deploying and Managing a Simple Web Application**

In this project, I created a Simple Web Application, containerized it using Docker, set up a Kubernetes cluster using Minikube, and deployed the application using Argo CD with a canary release strategy. Below is the step-by-step process along with the reasons for choosing each tool and the specific commands used.

#### **Tools Used:**

* **GitHub**: For version control and collaboration.
* **Minikube**: To create a local Kubernetes cluster.
* **Docker**: To containerize the application.
* **kubectl**: To manage Kubernetes resources.
* **Argo CD**: For continuous delivery and canary deployments.
* **Windows PowerShell**: As the command-line interface.
* **Docker Desktop**: To provide a Docker environment on Windows.
* **Argo Rollouts**: For advanced deployment strategies.

### 1. **GitHub Repository**

* **Version Control:** GitHub provides powerful version control and collaboration features, making it easy to track changes, manage code versions, and collaborate with other developers.
* **Integration:** It integrates well with many CI/CD tools and deployment platforms, including Argo CD, which simplifies the deployment process.

**Steps Taken:**

* Created a new repository named **Simple Web Application**.
* Added three files: **index.html**, **script.js**, and **style.css**.
* Github : [Link](https://github.com/kusuma-03/Simple-Web-Application.git)

### **2. Minikube**

**Local Kubernetes Environment:** Minikube allows you to run a local Kubernetes cluster on your computer. This is ideal for development and testing purposes without the need for a cloud provider.

**Ease of Use:** Minikube is user-friendly and easy to set up, making it a great choice for local development.

**Steps Taken:**

* Installed Minikube on the local computer.
* Started the Minikube cluster using command **minikube start**
* Verified the status through : **minikube start**

### **3. Docker**

### **Containerization:** Docker allows you to package applications and their dependencies into a container, ensuring consistency across different environments.

* **Integration with Kubernetes:** Docker containers can be easily deployed on Kubernetes clusters, making it a natural fit for this setup.

**Steps Taken:**

* Installed Docker Desktop.
* Created a **Dockerfile** for the Simple Web Application.
* Built the Docker image: **docker build -t my\_image\_name** .
* Ran the Docker image locally to ensure it works : **docker run -d -p 8080:80 my\_image\_name**
* Pushed the Docker image to Docker Hub:
* **docker tag my\_image\_name kusumaindhu/my\_image\_name**
* **docker push kusumaindhu/my\_image\_name**

### **4. kubectl**

* **Kubernetes Command-Line Tool:** **kubectl** is the command-line tool for interacting with Kubernetes clusters. It allows you to deploy applications, inspect and manage cluster resources, and view logs.
* **Essential for Kubernetes Management:** It is essential for deploying and managing Kubernetes resources, such as pods, deployments, and services.

**Steps Taken:**

* Installed **kubectl** using Chocolatey: **choco install kubernetes-cli**

### **5. Argo CD**

* **GitOps Continuous Delivery:** Argo CD is a declarative, GitOps continuous delivery tool for Kubernetes. It follows the GitOps methodology, where the desired state of the application is versioned in Git.
* **Application Management:** Argo CD makes it easy to manage application deployments and lifecycle, providing a user-friendly interface and robust deployment strategies, including canary releases.

**Steps Taken:**

* Created a namespace for Argo CD: **kubectl create namespace argocd**
* Installed Argo CD: **kubectl apply -n argocd -f** [**https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml**](https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml)
* Verified the Argo CD installation: **kubectl get pods -n argocd**
* Created kubernates manifests **:kubectl -f apply deployment.yaml**
* Created kubernates manifests **:kubectl -f apply service.yaml**
* Accessed the Argo CD UI: **kubectl port-forward svc/argocd-server -n argocd 8080:443**
* Logged into Argo CD and connected the GitHub repository. Accessed From local repository https://localhost:8080

### **6.Implement Canary Release and Perform Rollout**

* Canary releases allow gradual deployment, reducing risk and ensuring stability
* **Created a Canary Deployment Configuration (rollouts.yaml)**
* **Apply Canary Deployment:** **kubectl apply -f rollouts.yaml**

**Deploy New Version Using Canary Release:**

* Built and tagged a new Docker image:
* **docker build -t kusumaindhu/my\_image\_name:1.1.02**
* **docker push kusumaindhu/my\_image\_name:1.1.02**
* Applied the updated deployment: **kubectl apply -f rollouts.yaml**
* Monitor Rollout Status**: kubectl argo rollouts get rollout my-static-webapp—watch**

**Promote or Abort:**

* Promoted the canary release to full deployment: **kubectl argo rollouts promote my-static-webapp**

**7.Clearing resources**

Deleted minikube cluster with command :**minikube delete**

By using GitHub, Minikube, Docker, kubectl, and Argo CD, I created a robust local development environment for deploying applications with Kubernetes. The canary release strategy with Argo Rollouts allows for controlled and monitored updates, reducing risk and ensuring application stability.

1. **Minikube Compatibility:** Faced compatibility issues with the host OS; resolved by updating OS and dependencies.
2. **Docker Daemon Issues:** Docker daemon failed to start; fixed by resetting Docker settings and enabling BIOS virtualization.
3. **kubectl Authentication:** Encountered permission errors; resolved by properly configuring the kubeconfig file and context.
4. **Argo CD Installation:** Network issues prevented manifest downloads; fixed by adjusting network settings and firewall configurations.
5. **Argo CD Access:** Initial login challenges due to default password retrieval; resolved by following documentation to decode the password.