CI/CD with GitHub, GitHub Actions, Argo CD and Kubernetes Cluster

Tanmay Bhandge · Follow

9 min read · 5 hours ago

Introduction

In the rapidly evolving landscape of software development and deployment, organisations are constantly seeking more efficient ways to manage their application lifecycle. Traditional methods of manual deployments and continuous integration and continuous delivery (CI/CD) pipelines have proven to be time-consuming and error-prone. Enter GitOps, a modern approach to software delivery that leverages the power of version control systems and automation to streamline the deployment process.

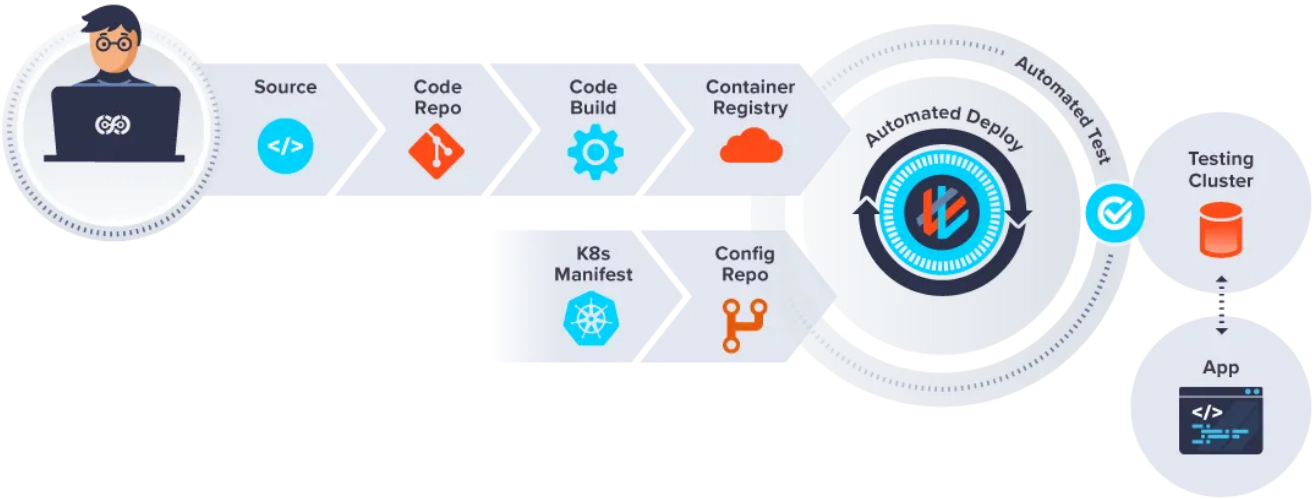


Photo by weaveworks

GitOps shifts the paradigm by using Git as the single source of truth for both application code and infrastructure configuration. By adopting GitOps principles, organisations can achieve greater visibility, traceability, and scalability in their deployments. In this article, we will explore the integration of GitOps with CI/CD pipelines, specifically focusing on the combination of GitHub Actions and Argo CD.

By combining the strengths of GitHub Actions and Argo CD, organizations can achieve a powerful end-to-end CI/CD pipeline that embraces the GitOps philosophy. Developers can make changes to their codebase, commit them to the main branch, and trigger a series of automated actions that build, test, and deploy the application to the desired environment.

Getting Started

Here's a high-level overview of how to set it up:

1. Git Repository: Create a Git repository to store your application code.

2. GitHub Actions: Configure GitHub Actions by creating a workflow file in the repository's .github/workflows/ directory. We will use the GitHub workflow to trigger the build image, push the image to the DockerHub and modify the Kubernetes manifest file as required for the deployment.

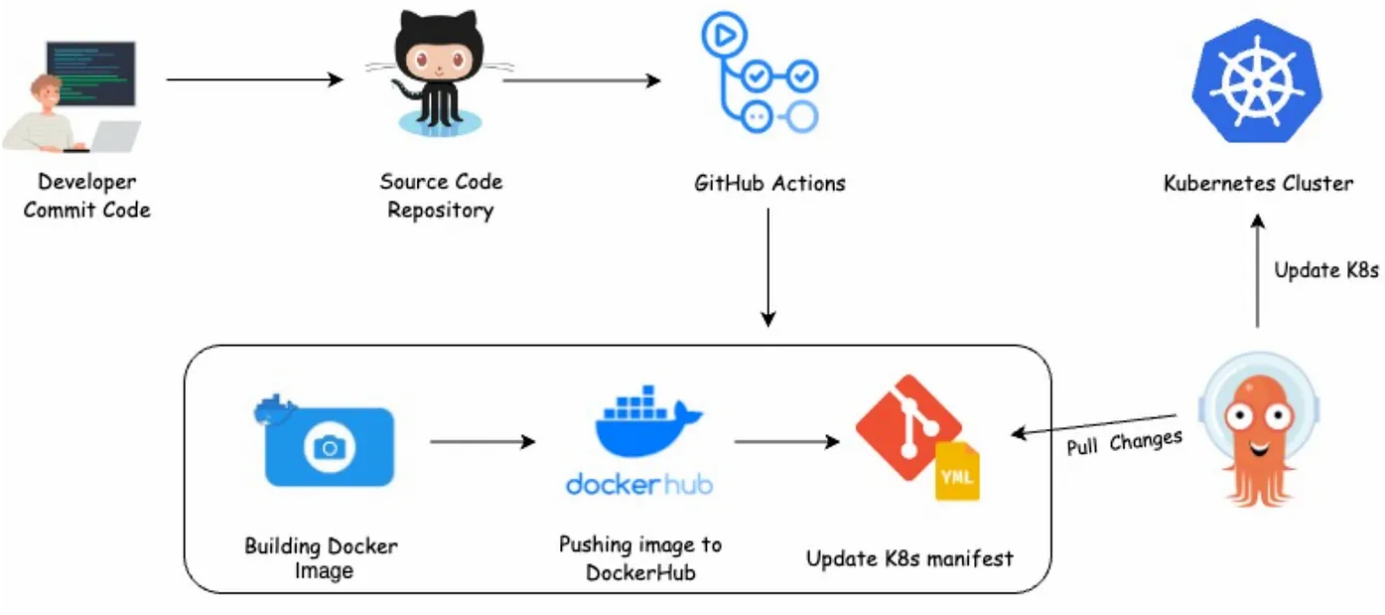
3. Create a Kubernetes cluster: Set up a Kubernetes cluster. I will be using NKE(Nutanix Kubernetes Engine). Ensure that you have the necessary permissions and access to manage the cluster.

4. Install Argo CD: Install Argo CD on the K8s cluster and configure it to connect to your Git repository which contains manifest files.

5. Define Argo CD application: Define an Argo CD application to manage the deployment of your Kubernetes resources. Enable automatic synchronization so that Argo CD can detect changes in the Git repository and trigger deployments accordingly.

6. Deploy applications with Argo CD: Argo CD will automatically detect changes to the Git repository and deploy the updated application on the Kubernetes cluster.

The Workflow



In the image above, you can observe the seamless integration. You can see I am using **GitHub Actions** to build a Docker Image of the application and then push the image to a **DockerHub** repository. And then update the version of the new image in the Manifest Git repo. We will be setting up two repositories one for the application code, and another for the Kubernetes manifests.

Every time your code is changed in the Application repository, a new Docker container image will be created, pushed to the DockerHub and it will update the image tag in the Kubernetes Manifest repository.

As soon as a change is detected in the Kubernetes Manifest repository, ArgoCD comes into action and starts rolling out and deploying the new application in the Kubernetes cluster. It ensures that the new version of our application is seamlessly deployed, eliminating any manual intervention or potential human errors.

Implementation

We first have to create a GitHub repository and put the application source code in it.

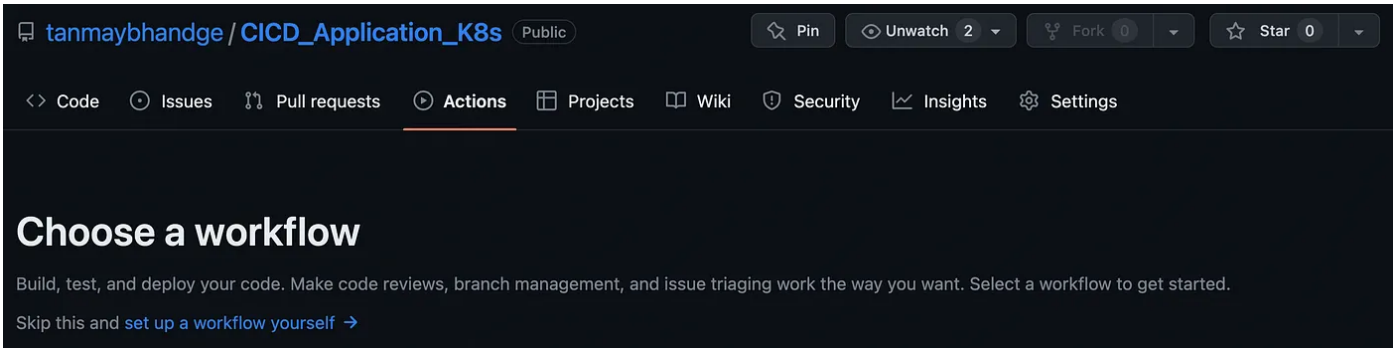
For the Application source code repository, we will be using a simple Flask application that displays a web page and this will be packaged in a docker image and published to the DockerHub.

For the Kubernetes Manifest repository, we will use a simple deployment and service K8s manifest.

Above manifest file defines a Kubernetes deployment and service for a Flask application. The deployment will create a single replica of the application, which will be exposed on port 5000. The service will expose the application on port 80 and will be accessible via the NodePort 30001.

Next, we need to set up GitHub Actions in the Application repository to build the Docker image from the Dockerfile present in the repository and then push the image to the DockerHub repository.

To create a workflow, select the GitHub repository, click Actions, and select “Set up a workflow yourself.” This will create a YAML file at path `.github/workflows/main.yml`. This is the only file that we need to create and modify in the GitOps phase.



Here is the workflow file

The above Git workflow file defines a workflow that will run on every push to the main branch. The workflow has three jobs:

- Build - This job will build the Python application using Python 3.10. It will also lint the application using flake8 and run unit tests using pytest.
- Docker - This job will build a Docker image for the application and push it to Docker Hub.

- Modifygit - This job will modify the deployment manifest in the CICD-Manifest repository to use the newly-pushed Docker image.

Here is a more detailed description of each job:

Build

- The first step in this job is to checkout the code from the repository.
- The next step is to set up Python 3.10. This is done using the actions/setup-python action.
- The third step is to install the dependencies for the application. This is done using the pip install command.
- The fourth step is to lint the application using flake8. This is done by running the flake8 command.
- The fifth step is to run unit tests using pytest. This is done by running the pytest command.

Docker

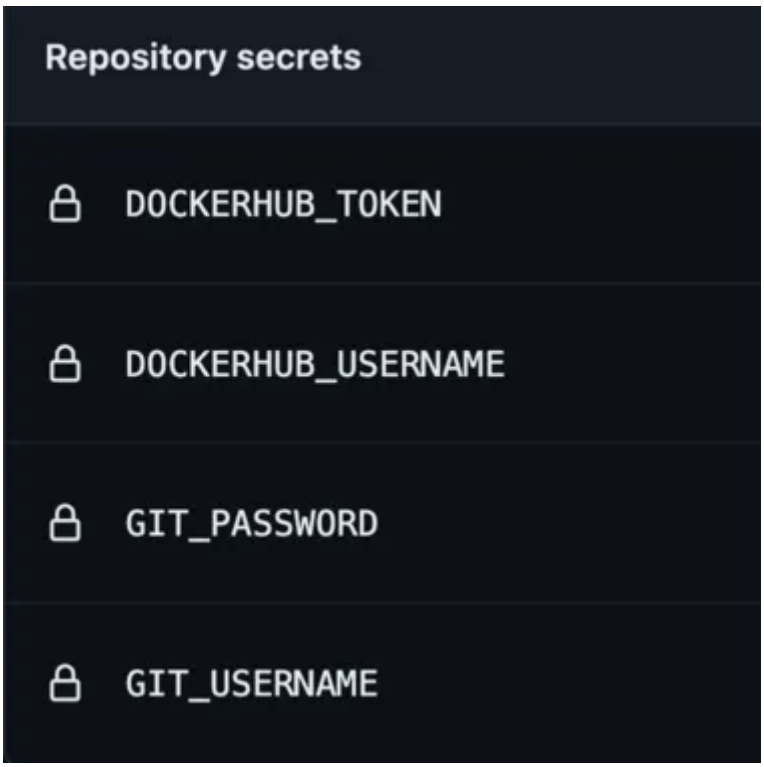
- The first step in this job is to checkout the code from the repository.
- The next step is to set up QEMU. This is done using the docker/setup-qemu-action action.
- The third step is to set up Docker Buildx. This is done using the docker/setup-buildx-action action.
- The fourth step is to login to Docker Hub. This is done using the docker/login-action action.
- The fifth step is to build and push the Docker image. This is done using the docker/build-push-action action.

Modifygit

- The first step in this job is to checkout the code from the CICD-Manifest repository.

- The next step is to modify the deployment manifest to use the newly-pushed Docker image. This is done by running the sed command.
- The final step is to push the changes to the repository. This is done using the git push command.

The above GitHub repo uses secrets for Docker Hub and Git. To create secrets in a GitHub repo, go to the repository settings, select secrets, and click New repository secret. Give the secret a name and a value, and then you can use it anywhere in the repo. Secrets are encrypted and stored in GitHub, so they are safe from prying eyes. You can use secrets to store any type of sensitive information, such as API keys, passwords, and tokens.



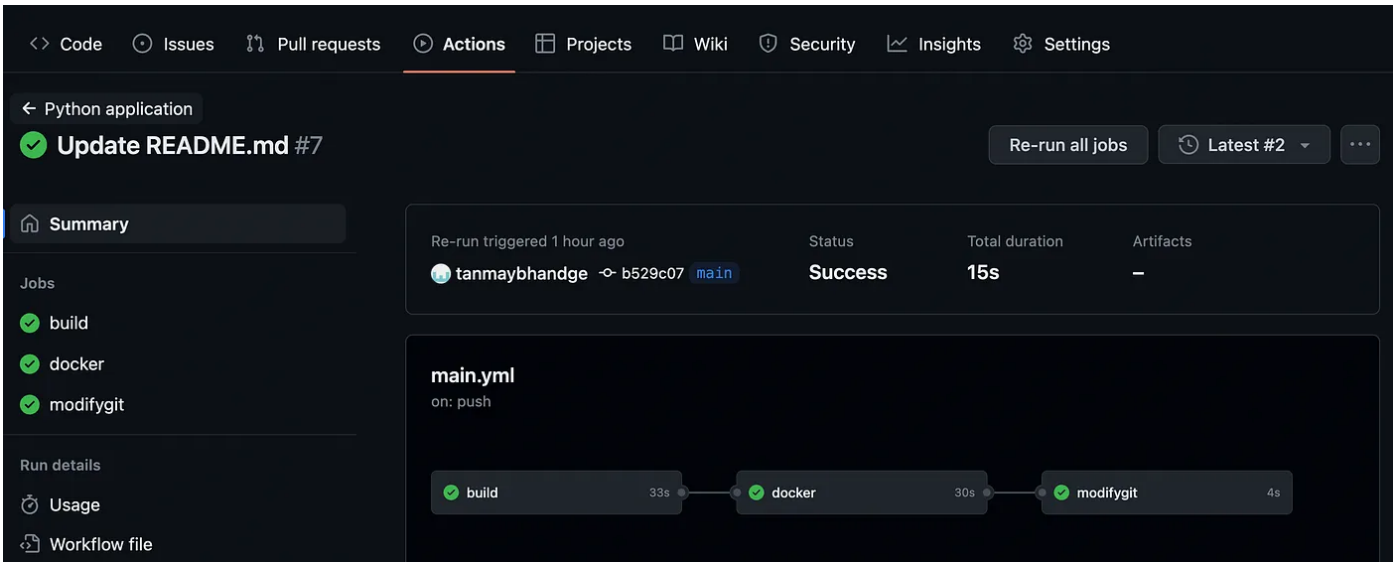
The GitOps CI/CD pipeline is now set up to automate the build, push, and deployment processes. Whenever a commit is made to the main branch of Application repository, the pipeline will be triggered automatically. It performs the following actions:

1. Builds and pushes the Docker image - The pipeline uses the Dockerfile in the repository to build a Docker image. It then pushes the image to a Docker registry, such as Docker Hub. This step ensures that the latest version of the application is available for deployment.

2. Updates the version in the manifest repository - The pipeline updates the version of the newly built image in a separate Git repository that holds the deployment manifests. This ensures that the deployment manifests reflect the latest image version.

3. Triggers ArgoCD deployment - The changes made to the deployment manifest repository automatically trigger ArgoCD, which is a GitOps tool for deploying applications to Kubernetes. ArgoCD uses the updated deployment manifests to deploy the application in the Kubernetes cluster.

The pipeline also provides visibility into the build status, as shown in the accompanying image. This allows you to monitor the success or failure of the CI/CD process.



Installing ArgoCD in Kubernetes Cluster

To install ArgoCD on an NKE (or any other Kubernetes cluster), you can use the following command:

```
kubectl create namespace argocd
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stab
```

This command will create a namespace called argocd and deploy **ArgoCD** on your Kubernetes cluster using the installation manifests provided by the ArgoCD project. The manifests are fetched from the GitHub repository and applied to the argocd namespace.

After running the installation command, you can verify the deployment by checking the status of the ArgoCD pods:

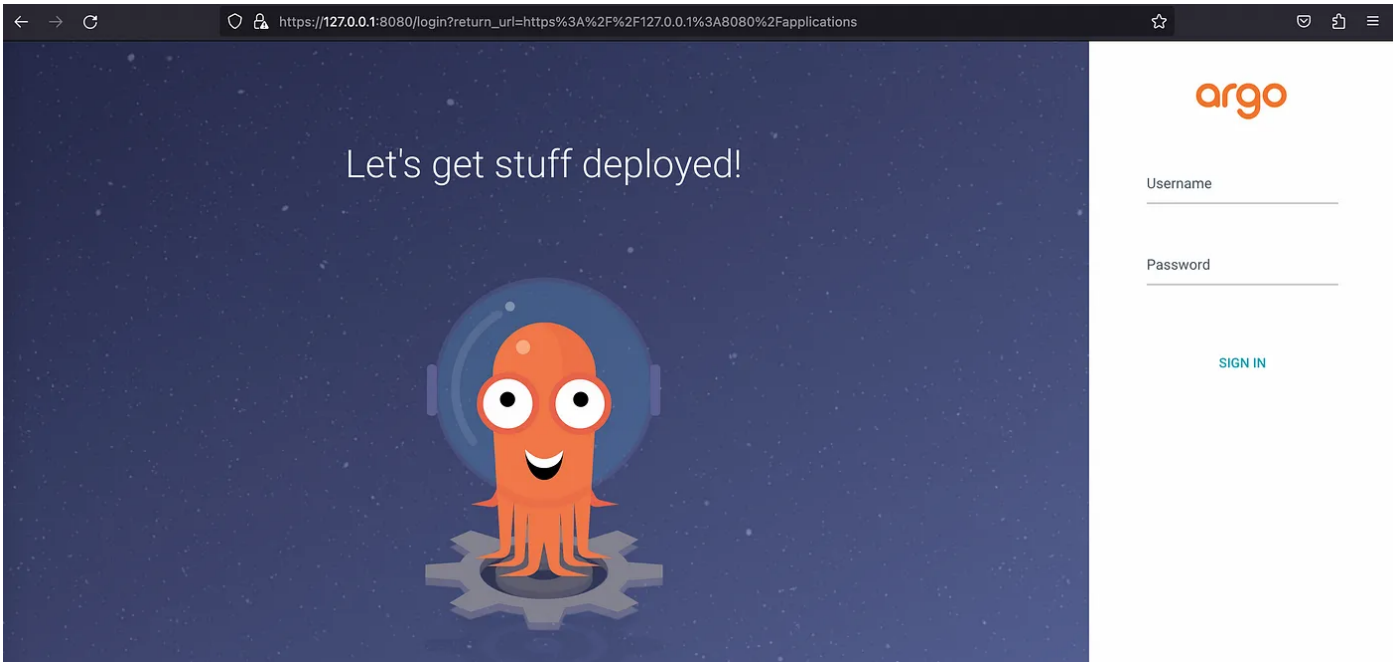
```
kubectl get pods -n argocd
```

To access the ArgoCD dashboard, I will be using Port Forwarding to access the ArgoCD.

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

Access ArgoCD Dashboard from your local machine using the following link

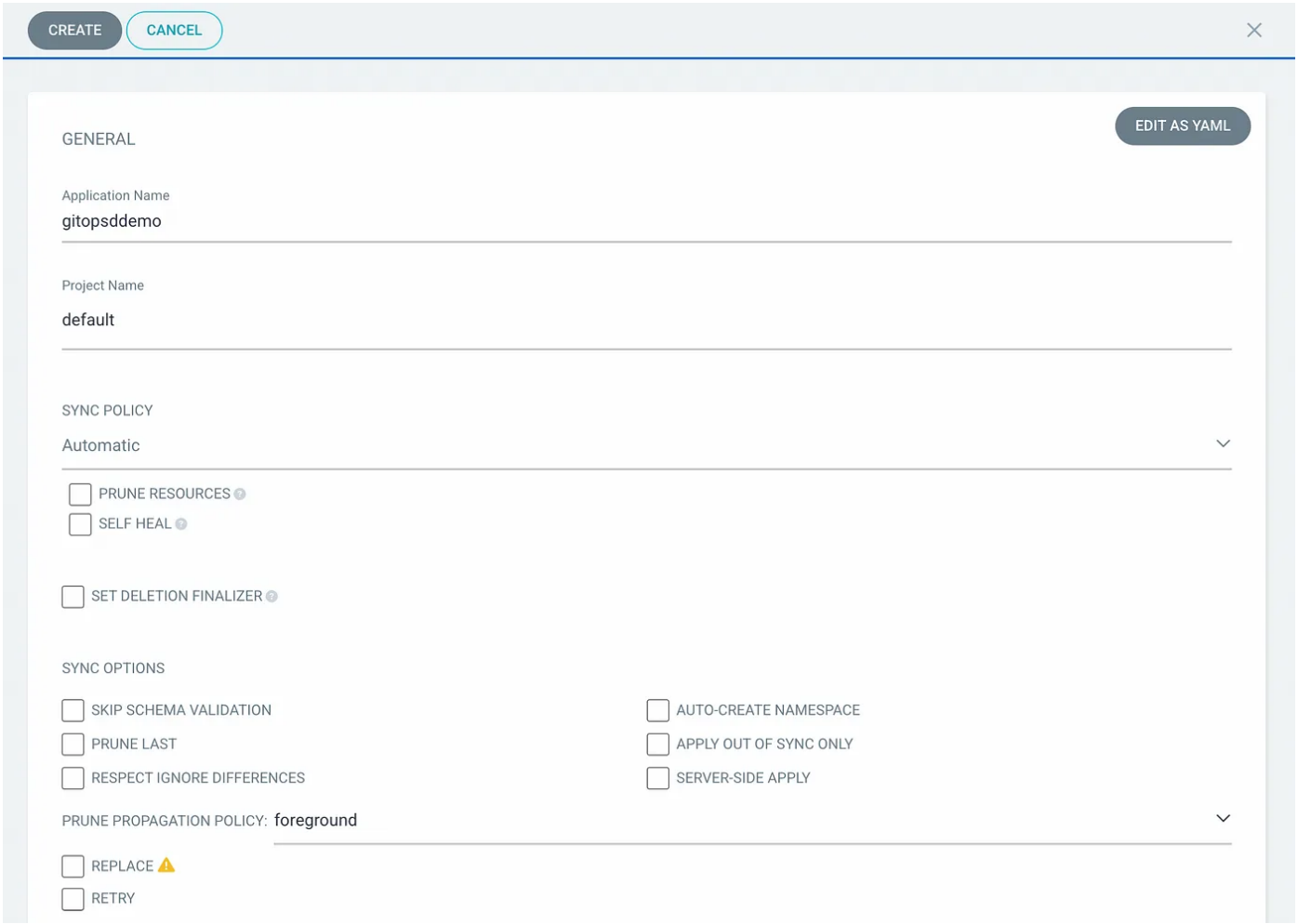
http://127.0.0.1:8080

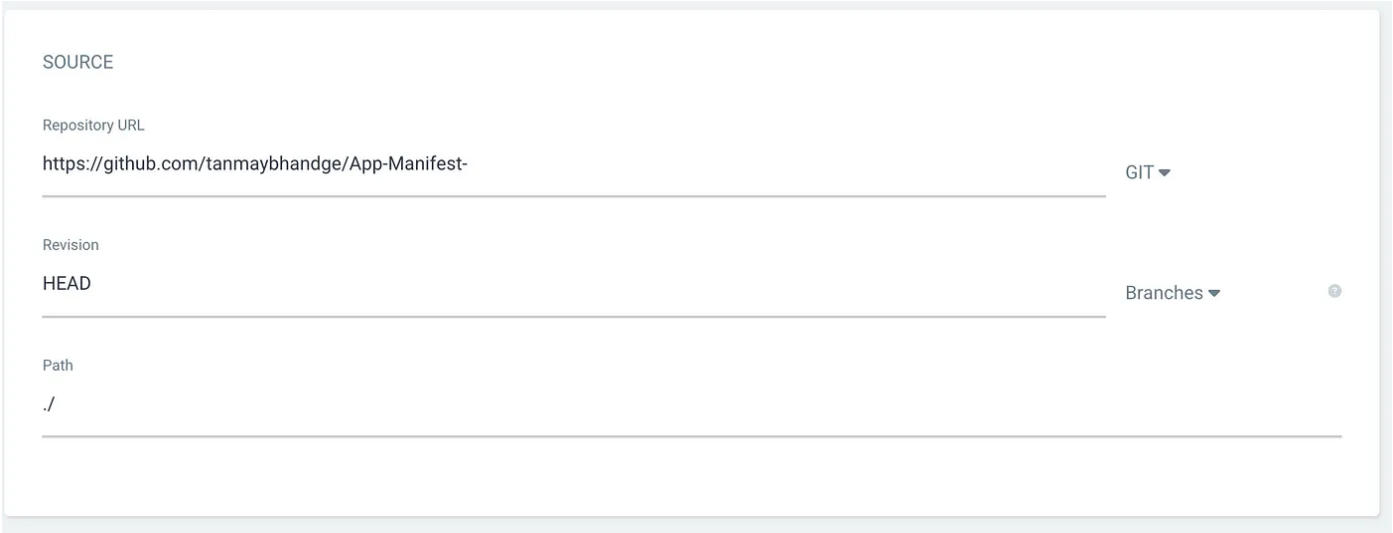


To get the password you may execute the below command in your Kubernetes cluster. (username is admin)

```
kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.passw
```

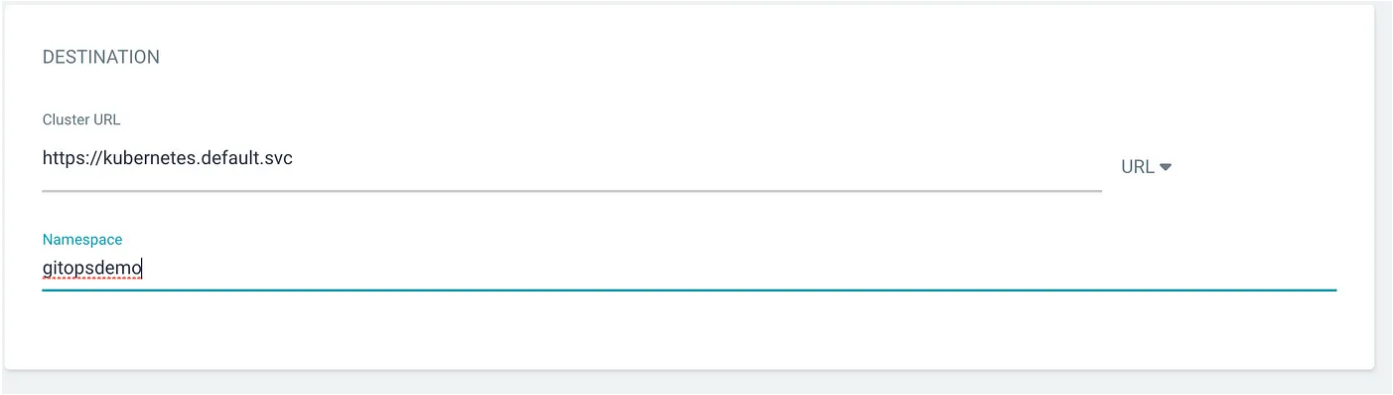
Next, we have to create an App in ArgoCD in which we basically define where is our application's repository located and where to deploy it, and some other small configurations.





The repository URL is the one where we have the manifest file. and the **path** is ./

With the incorporation of ArgoCD into our deployment pipeline, we gain continuous monitoring capabilities for our manifest repository. ArgoCD diligently observes this repository, and whenever changes are detected, it swiftly initiates the synchronization process, ensuring that the latest updates are seamlessly applied to our Kubernetes cluster.

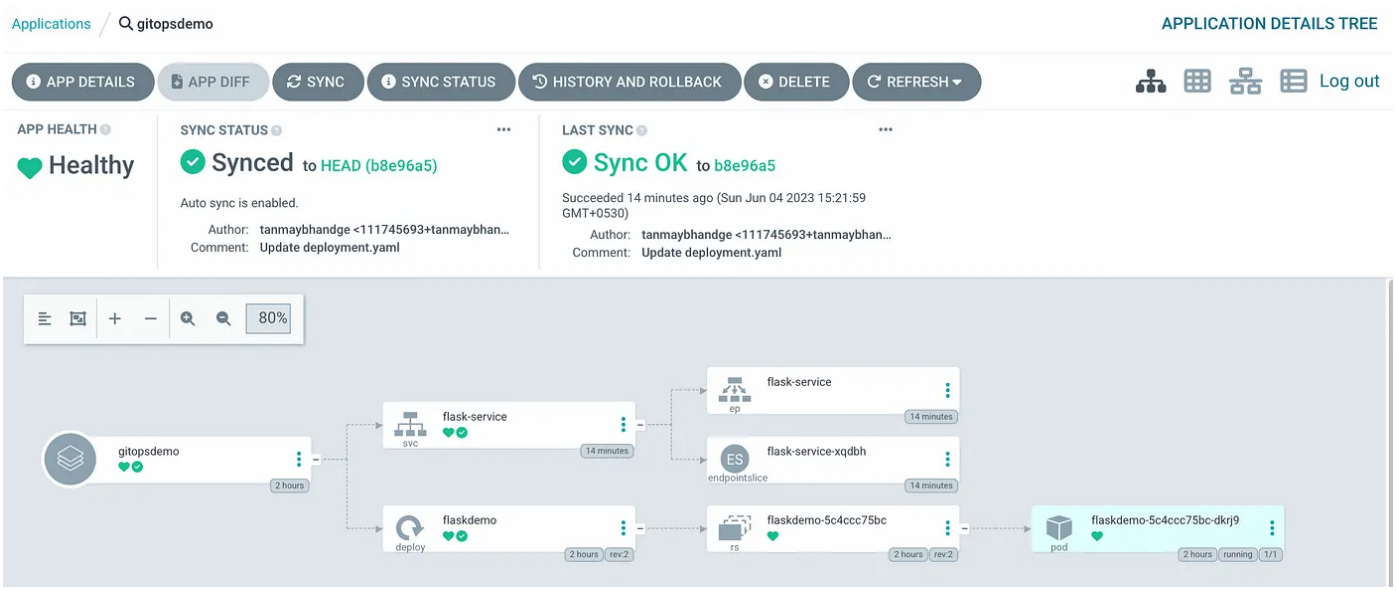


Now click on **Create** button. This will initiate the creation process of an application in ArgoCD, and an exciting journey begins. ArgoCD diligently takes charge and starts the synchronization process, aiming to seamlessly deploy the resources defined in the manifest file to our Kubernetes cluster.

During this synchronization phase, ArgoCD carefully examines the manifest file and assesses the current state of the Kubernetes cluster. If the resources defined in the manifest file do not already exist in the cluster, ArgoCD leaps into action. It swiftly

orchestrates the deployment of these new resources, ensuring that the desired application components are provisioned in the cluster.

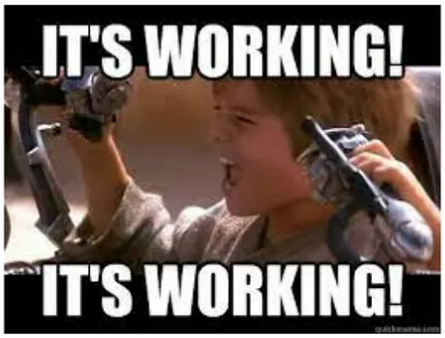
This automated process not only saves us valuable time but also eliminates the risk of manual errors that often accompany manual resource creation and deployment.



As we have defined the nodeport service type in the manifest file, we can access the pod using the node port. <WorkerNodeIP:30001>

By entering the appropriate Worker Node IP address and the designated NodePort in a web browser or any applicable tool, we can effortlessly establish a connection to the Pod running our application.

Demo on GitOps with ArgoCD and Github Actions.



We have successfully implemented a highly efficient CI/CD workflow that is now fully automated. As a result, whenever a developer commits changes to the main branch of the Application repository, the updates are seamlessly reflected on the main site without any manual intervention. This automated process eliminates the need for manual deployments, saving us time and effort. Witnessing this level of automation in action is truly remarkable and highlights the effectiveness of our CI/CD implementation.

I sincerely hope that you have found this article to be an enjoyable and informative read.

References

[+] Application Repository -https://github.com/tanmaybhandge/CICD_Application_K8s

[+] Manifest Repository -https://github.com/tanmaybhandge/App-Manifest-

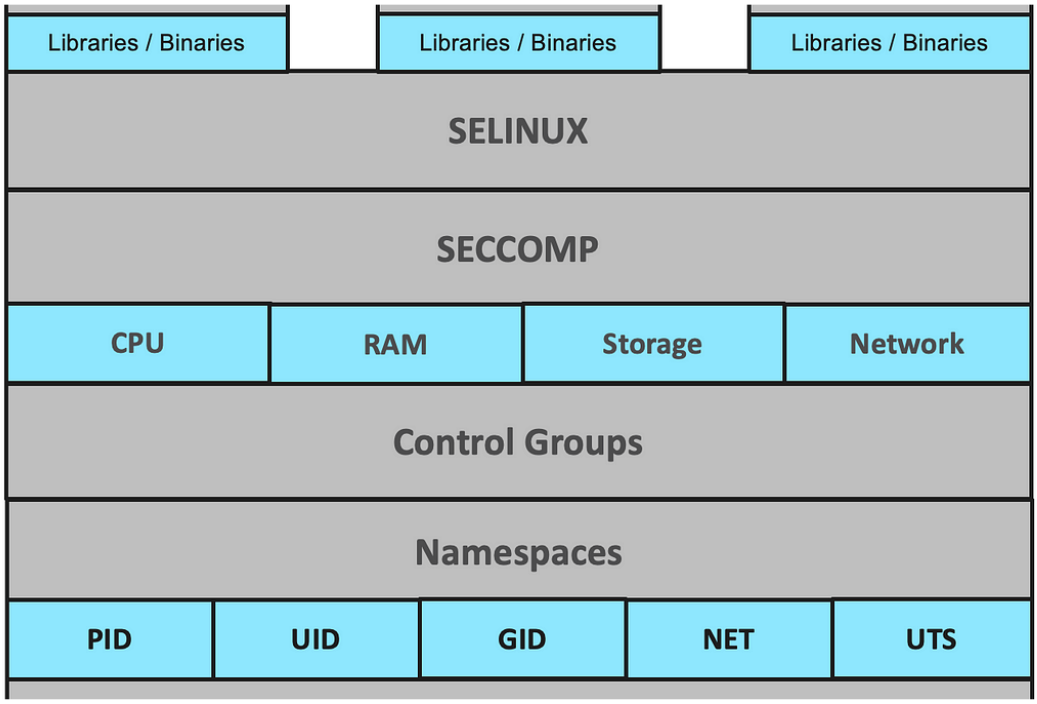
Kubernetes | Ci Cd Pipeline | Github Actions | Argo Cd | DevOps

Follow

Written by Tanmay Bhandge

6 Followers

More from Tanmay Bhandge



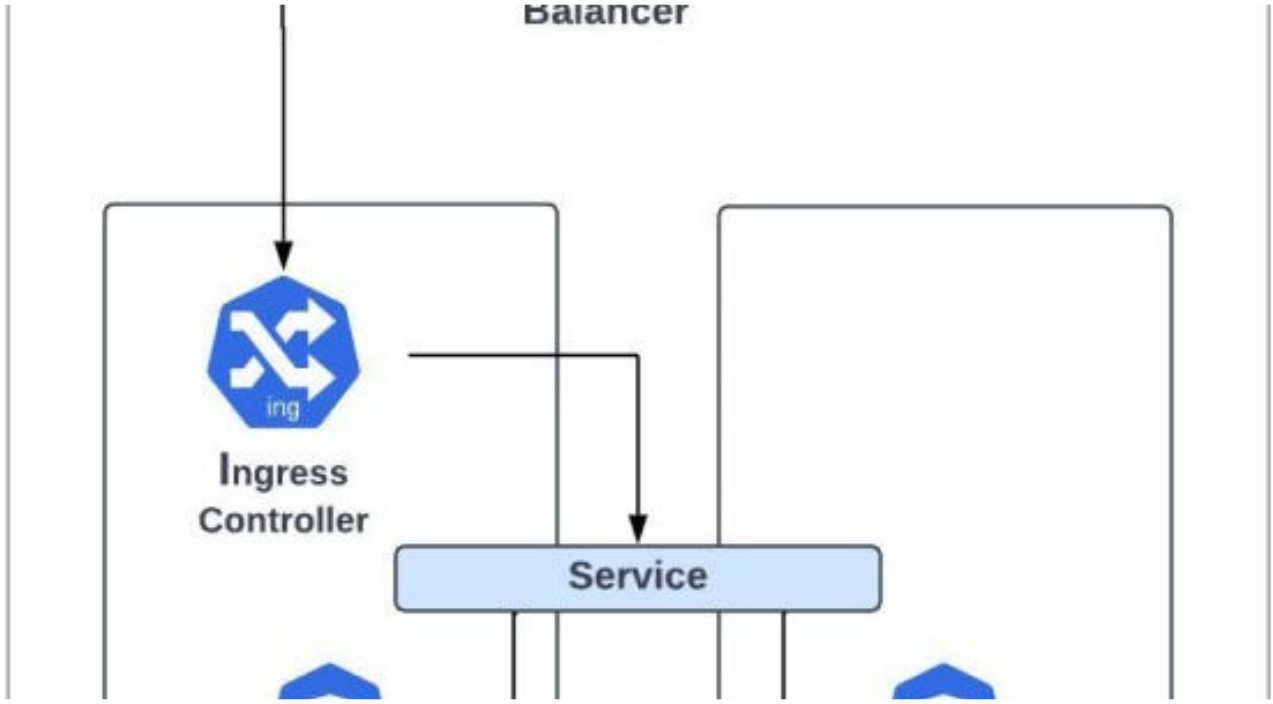
Tanmay Bhandge

How Linux Technologies Make Up the Foundations of Building and Running a Container Process

Containers are a popular way to package and deploy applications. They offer a number of advantages over traditional virtual machines...

3 min read · 5 days ago





Tanmay Bhandge

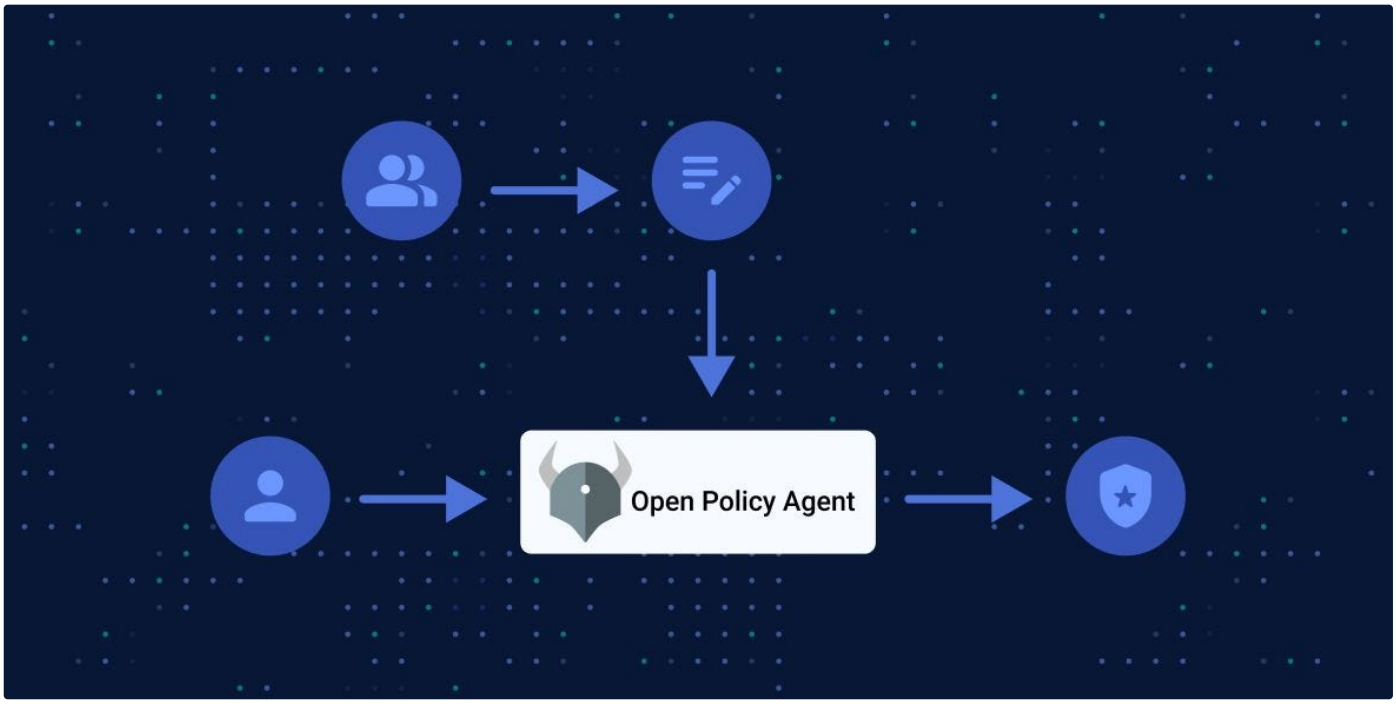
Simplifying Service Exposure in on-premises Kubernetes cluster using Ingress Controller

You may aware that Kubernetes does not offer a network load balancer for the on-premise cluster. If you take into consideration of various...

5 min read · Mar 15

See all from Tanmay Bhandge

Recommended from Medium



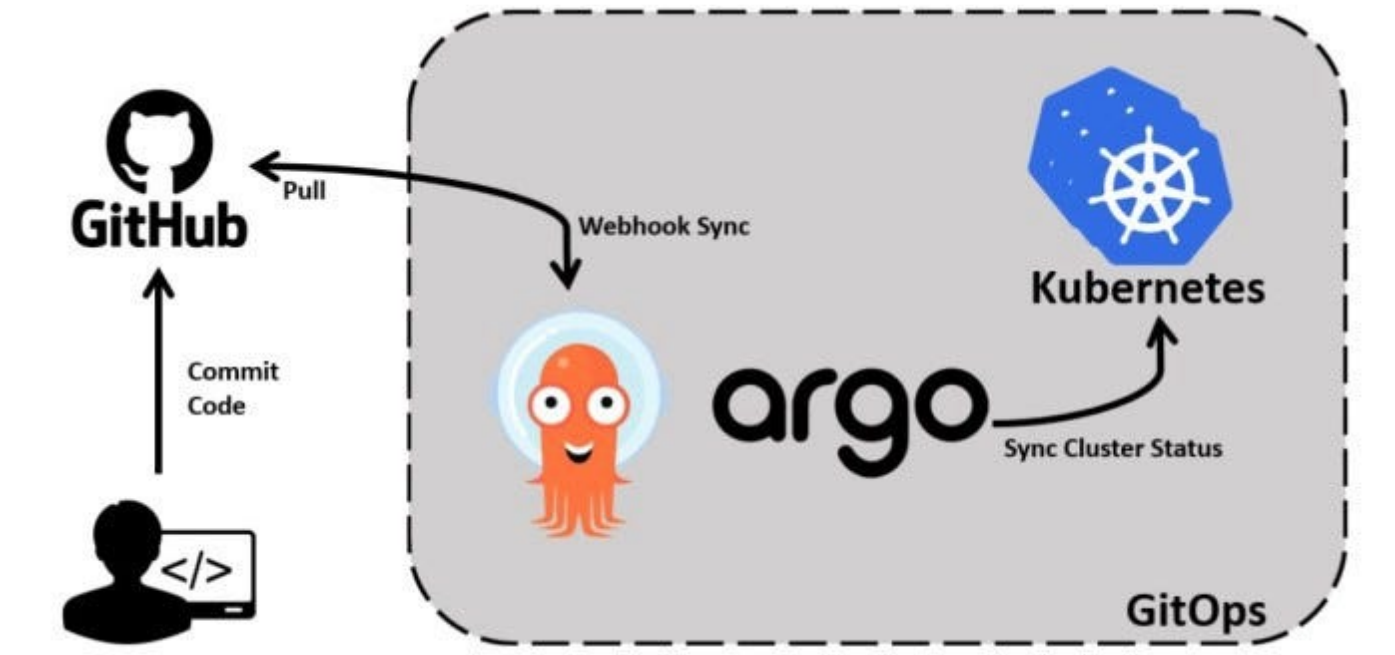
Tiexin Guo in GitGuardian

What is Policy-as-Code? An Introduction to Open Policy Agent

Learn the benefits of policy as code and start testing your policies for cloud-native environments.

· 6 min read · Jan 24

27



Paris Nakita Kejser in DevOps Engineer, Software Architect and Software Developering

Connect GitHub account to deploy private git repository with ArgoCD

Using ArgoCD is very cool, one of the first walls you will hit as I did are how can you use your private repo inside ArgoCD whiteout to...

· 4 min read · Feb 24

47

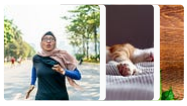
Lists

Staff Picks

335 stories · 91 saves

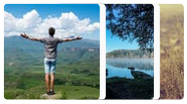
Stories to Help You Level-Up at Work

19 stories · 70 saves



Self-Improvement 101

20 stories · 123 saves



Productivity 101

20 stories · 136 saves

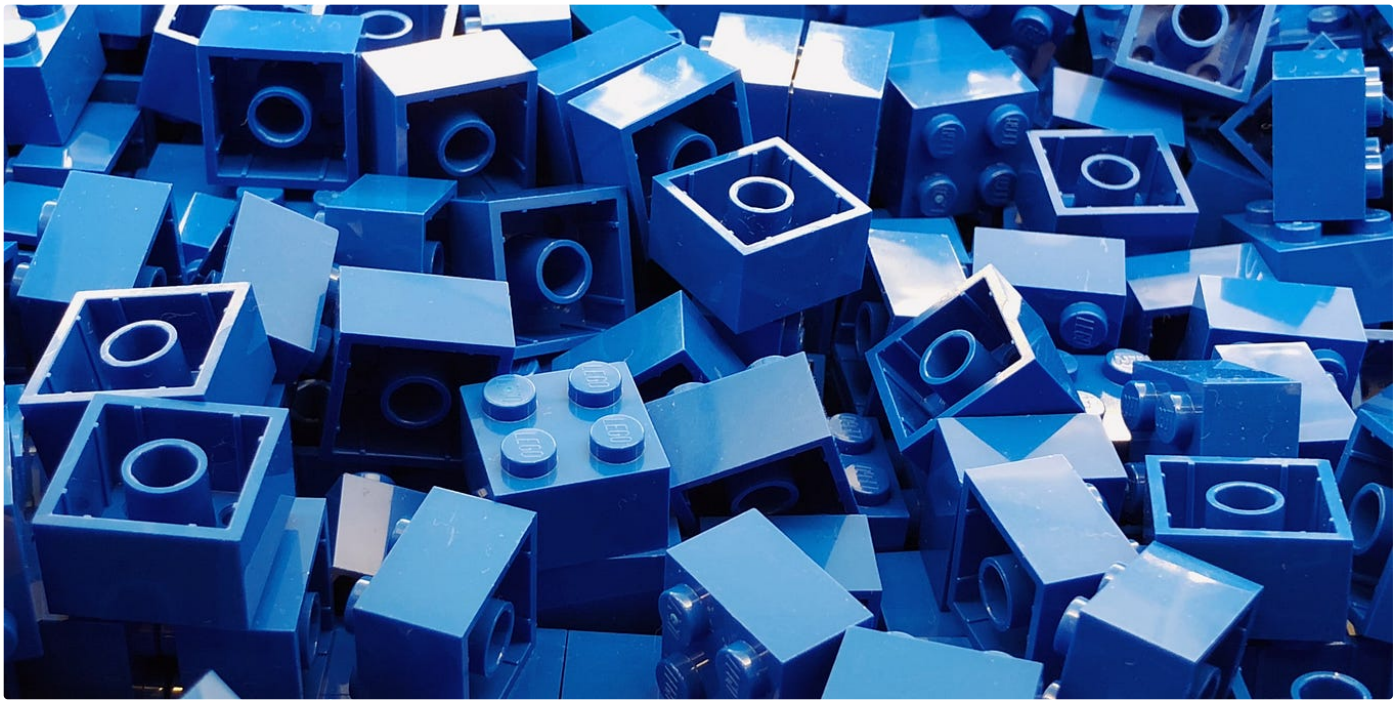
Zhimin Wen

Multi-arch Container Image

You build your application container image on a Linux server, push it over to the Github registry. On your new Apple silicon M1 laptop, you...

· 4 min read · Dec 6, 2022





Daryl in AWS in Plain English

Terraform Modules How To Create & Use Them The Right Way!

Make your life a whole lot easier by utilising Terraform modules

· 5 min read · Jan 22

34

Tanmay Bhat in FAUN—Developer Community

Automate Your Helm Chart Testing Workflow with GitHub Actions

Helm is a popular open-source package manager for Kubernetes that simplifies the process of installing, upgrading, and managing...

7 min read · Dec 26, 2022

20

Igor Zhivilo in Everything Full Stack

GitOps way with Github Actions and self-hosted runner on Kubernetes

Hi, I'm DevOps Engineer at Tikal Knowledge.

6 min read · Feb 22

202

See more recommendations