

[Open in app](#)[Sign up](#)[Sign in](#)**Medium** Search

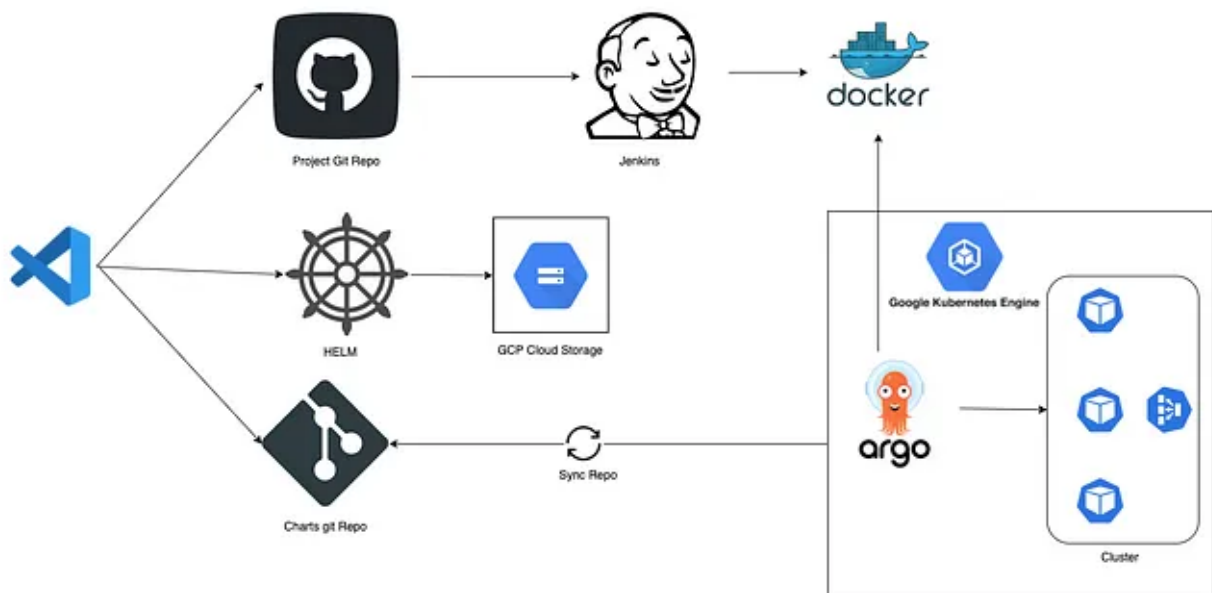
Deploy Node.js application over Google Cloud with CI/CD

How to automate the deployment of a Node.js app with CI/CD using Google Kubernetes Engine, Jenkins, DockerHub and ArgoCD

**Aashirwad Kashyap** · Follow

Published in Bits and Pieces

7 min read · Feb 3, 2022

 Listen Share

High Level architecture

Introduction

We will be deploying a containerized Node.js application using Docker over Google Kubernetes Engine using Jenkins for CI (building pipelines to build, test and upload image to Docker Hub) and argoCd with Helm for CD.

It will sync up with our charts repo, pull images uploaded by Jenkins to dockerHub and accordingly apply changes to the GKE cluster infrastructure.

This will also work with any other application too if you can write a Docker file for it.

Table of Contents

1. *Prerequisites*
2. *Project set-up*
3. *GKE Cluster setup on Google Cloud*
4. *Jenkins setup on Google Cloud*
5. *Jenkins CI set-up:*
 - *Pipeline setup*
 - *Register web-hook to trigger build*
6. *ArgoCD set-up:*
 - *Integrate chart git Repo*
 - *Create public charts registry*
 - *Install argoCd over GCP*
 - *Configure ArgoCd to Sync with Git Chart Repo*

Prerequisites

- *You need Docker installed*
- *You need Helm package manager*
- *You need Node.js installed*
- *You need a docker hub repository (create a free tier account on docker hub if you don't have one)*
- *You need a Google cloud account:*
 - *If you don't have create a free account you will get 300\$ initially and you will not be charged until you manually update your account to paid account.*
 - *Activate your billing as Kubernetes, Jenkins resources requires it. Your 300\$ will be used for this.*

Project Setup

We will clone the project git repo and test it locally once. You can clone the sample repository [devopspractice](#) for uniformity.

Run the below commands to build and spin up a local container for testing:


```
npm install
docker build .
docker run -itd -p 3000:3000 --name <containerName> <image>
curl --location --request GET 'http://localhost:3000/v1/dev/data'
```

If you get a '201 created' everything works great.

GKE Cluster Setup on Google Cloud

From GCP Dashboard > kubernetes-Engine > create > take below reference for the fields (Fig 1).

Change defaultPool > numberOfNodes > 2 (*to save money*). It will take few minutes to create the cluster.



| ✔ k8s-cluster | | |
|----------------------|-----------------|---------------------|
| DETAILS | NODES | STORAGE |
| LOGS | | |
| Cluster basics | | |
| Name | k8s-cluster | 🔒 |
| Location type | Zonal | 🔒 |
| Control plane zone | us-central1-c | 🔒 |
| Default node zones ? | us-central1-c | ✎ |
| Release channel | None | ✎ UPGRADE AVAILABLE |
| Version | 1.21.5-gke.1302 | |
| Total size | 2 | ℹ |

Fig 1

Jenkins Setup On GCP

We will be using GCP managed Jenkins service and set it up.

GCP Dashboard > marketplace > Jenkins packaged by Bitnami > launch.

Give a name and zone (us-central1-c) and create.

Once created go to site address (Fig 2) add fill in the username, password and you will be able to access it (If not try removing /jenkins from the end).

Click on SSH (Fig 2) > open in browser window. Then follow the below steps.

Install docker using this [link](#) on Vm(complete all steps from section *Install using the repository and Install Docker Engine*).

In one of the steps a specific version of docker is asked either you select one in the list or just paste this command.

```
sudo apt-get install docker-ce=5:20.10.12~3-0~debian-buster docker-ce-  
cli=5:20.10.12~3-0~debian-buster containerd.ioC
```

Once docker is installed we need to give docker access and install kubectl using below commands:

```
sudo usermod -aG docker jenkins  
sudo apt-get install kubectl
```

We need to restart Jenkins vm for these changes to appear (dashboard (navigation menu) > compute engine > Jenkins VmInstance > Stop > and then Start).



Jenkins packaged by Bitnami

Solution provided by Bitnami

| | |
|-------------------------------|---|
| Site address | http://35.238.144.144/jenkins |
| Admin user | user |
| Admin password (Temporary) | |
| Instance | jenkins-1-vm |
| Instance zone | us-central1-c |
| Instance machine type | g1-small |

✓ [MORE ABOUT THE SOFTWARE](#)

Get started with Jenkins packaged by Bitnami

[VISIT THE SITE](#)

SSH



Fig 2

Jenkins CI Setup

Here we will connect Jenkins to our project git Repo, build pipelines and add a Webhook for automatic build trigger over Jenkins on any commits in our Repo.

Go to your Jenkins Dashboard > manageJenkins > managePlugins > Install 'Docker pipeline' and 'Google Kubernetes Engine'.

From Jenkins Dashboard > manageJenkins > Manage Credential > global(below) > add credential.

Create new credential for GIT using your git username and password.

Create new credential for DockerHub using your dockerHub username and password (Fig 4 after successful creation).

Stores scoped to Jenkins

| P | Store ↓ | Domains |
|---|---------|--|
|  | Jenkins |  (global) |

Fig 3





| | | | |
|---|-----------|---|------------------------|
|  | git |  @gmail.com/***** | Username with password |
|  | dockerhub |  /***** | Username with password |

Fig 4

Now we will setup the pipeline. From Jenkins Dashboard > new Item > give project name and select pipeline.

In General section > tick Github Project and add Git project URL (e.g. <https://github.com/aashirwad3may/devopspractice.git/>).

In BuildTriggerSection > tick GitHub hook trigger for GITScm polling.

In pipeline Section add below fields (Fig 5) and rest remains same and save.

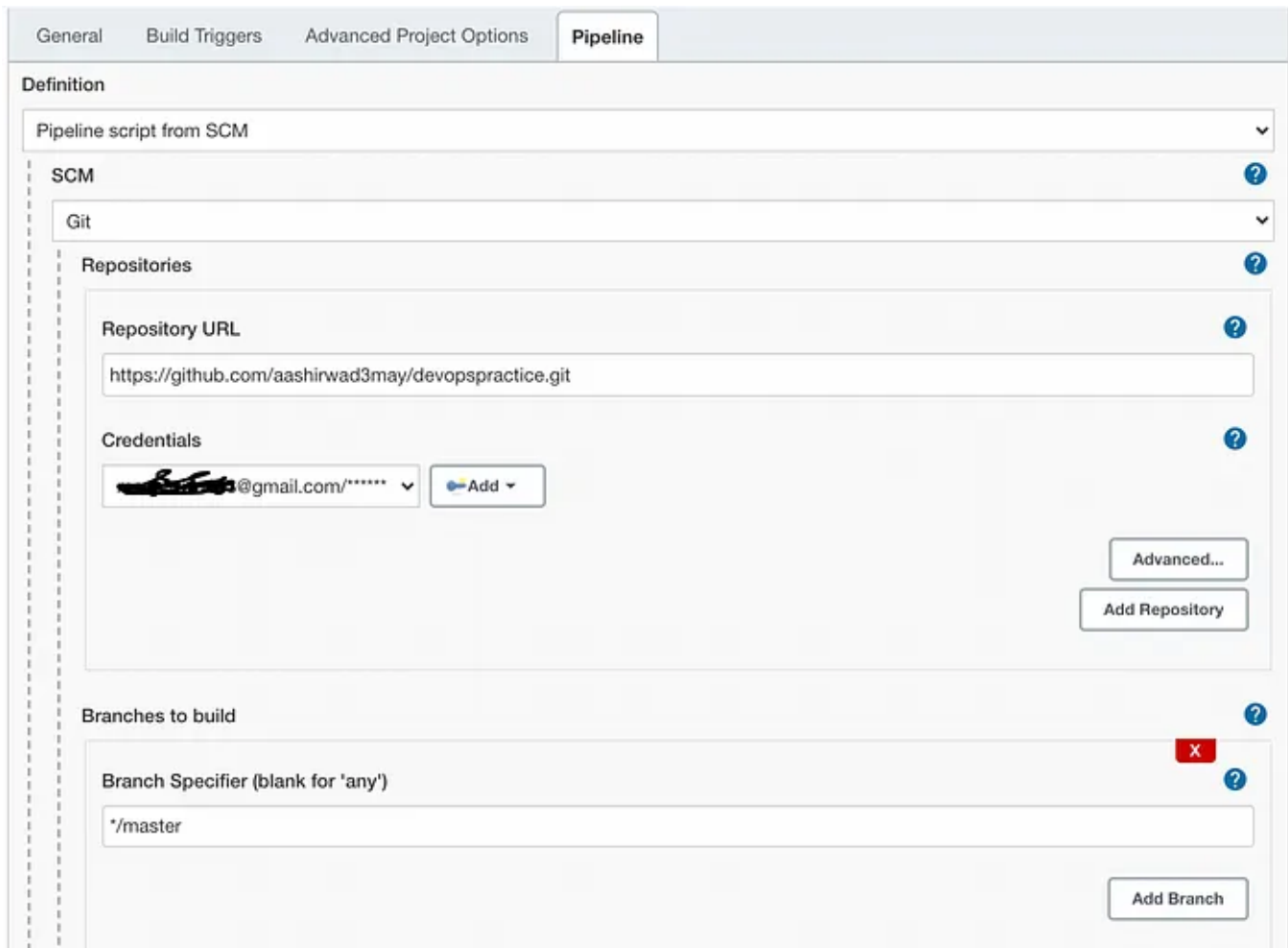
The image shows the Jenkins Pipeline configuration page. At the top, there are tabs for 'General', 'Build Triggers', 'Advanced Project Options', and 'Pipeline', with 'Pipeline' being the active tab. Under the 'Definition' section, 'Pipeline script from SCM' is selected. The 'SCM' dropdown is set to 'Git'. In the 'Repositories' section, the 'Repository URL' is 'https://github.com/aashirwad3may/devopspractice.git'. The 'Credentials' dropdown shows a partially redacted email address '@gmail.com/*****' with an 'Add' button next to it. There are 'Advanced...' and 'Add Repository' buttons. In the 'Branches to build' section, the 'Branch Specifier (blank for \'any\')' is set to '*/master'. There is a red 'X' icon and an 'Add Branch' button.

Fig 5

The steps to be taken inside the pipeline are mentioned in project git repo with file name Jenkins... change your docker hub username in Jenkins's file (ref fig 6).

```
stage("Build image") {  
  steps {  
    script {  
      myapp = docker.build("<Your DockerHub Username>/devops_practice:${env.BUILD_ID}")  
    }  
  }  
}
```

Fig 6

On the Jenkins Dashboard the project will be shown click on it and trigger the build using BuildNow you should see pipeline execution with below result.

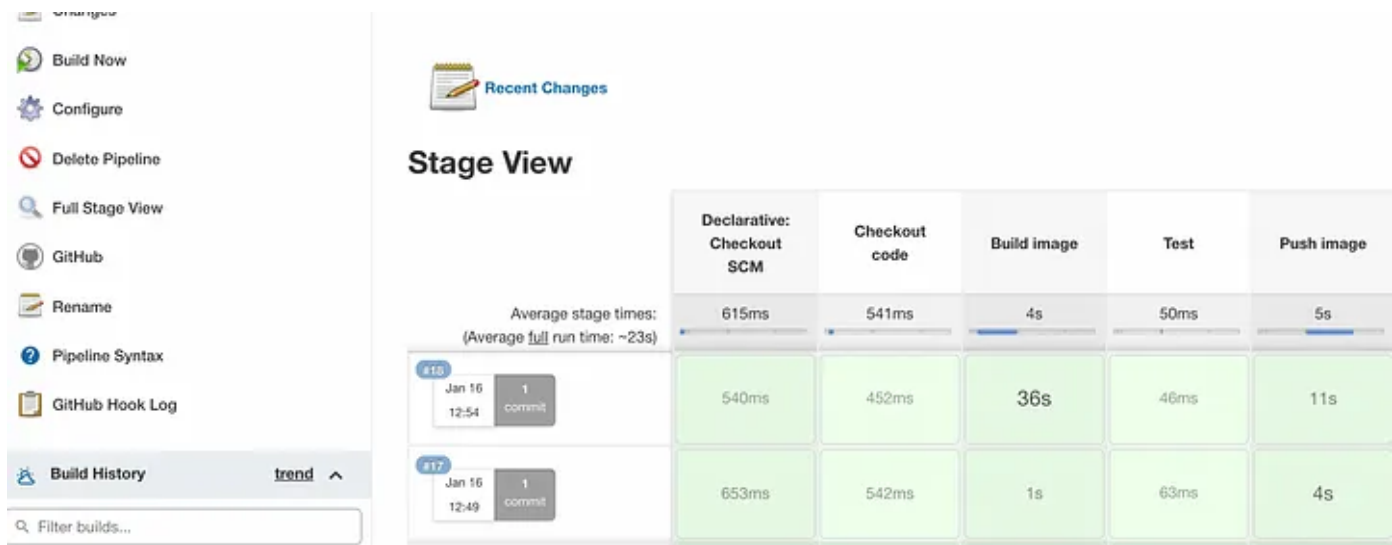


Fig 7

Here we are manually triggering the build to automate this we will use webhook so whenever you commit in git build will be triggered and image will be uploaded in your dockerHub Repo.

In order to add a webhook from Git project settings -> webhook -> add Webhook.

In Payload add only ip (e.g 14.112.217.344)(<http://<Jenkins GCP SITE IP>/github-webhook/>).

Content type application.json.

Create a webhook and commit in your local repo build should be triggered automatically.

ArgoCD Setup

Here we will setup helm on local system, create and use Cloud storage keep all our charts version (charts storage), install ArgoCd on GCP, create a secret to store dockerHub credentials on GKE and finally setup ArgoCD to sync with our git chart repo.

Install Helm on your local system use this [link](#) for ref.

As you can see in architecture diagram at the top, we have two git repo one for development and other for deployment (charts and values.yaml) as it is recommended. You can clone my charts repo [here](#) for ref and uniformity.

We need a charts storage where argoCD will pull helm charts we will use google cloud storage for this.

From GCP Dashboard->CloudStorage->createBucket (choose region as below and rest default) (Fig 8).

• Choose where to store your data

This permanent choice defines the geographic placement of your data and affects cost, performance, and availability. [Learn more](#)

Location type

- ☐ Multi-region
Highest availability across largest area
- ☐ Dual-region
High availability and low latency across 2 regions
- ☒ Region
Lowest latency within a single region

Location

us-central1 (Iowa)

fig 8

| <input type="checkbox"/> Name ↑ | Created | Location type | Location | Default storage class ? |
|--|---------------------------|---------------|-------------------|-------------------------|
| <input type="checkbox"/> devops-practice-bucket2 | Jan 15, 2022, 11:16:56 PM | Region | asia-south1 (...) | Standard |

Fig 9

Once created we need to change it read access to public for this go inside bucket->permission->add (Fig 10).

Add principals and roles for "devops-practice-bucket2" resource

Enter one or more principals below. Then select a role for these principals to grant them access to your resources. Multiple roles allowed. [Learn more](#)

New principals

allUsers ✕ ?

Role

Condition

Filter Type to filter

Cloud Build

Cloud Composer

Cloud Dataplex

Cloud Deploy

Cloud Migration

Cloud Storage

Cloud Storage Legacy

Dataflow

Roles

Storage Admin

Storage Object Admin

Storage Object Creator

Storage Object Viewer

MANAGE ROLES

Fig 10

Public access

Public to internet

One or more bucket-level permissions grant access to everyone on the internet (**allUsers**) or anyone signed into a Google account (**allAuthenticatedUsers**). If this bucket should not be publicly accessible, remove these public permissions or prevent public access to this bucket. [Learn more](#)

PREVENT PUBLIC ACCESS

Fig 11

Add the role and save it should now show the access as public just as above (Fig 11).

Now we will configure deployment.yaml to connect to dockerHub and create a secret to hold dockerHub credentials and finally create and upload charts to Cloud Storage.

Project Git Repo > helm > templates > deployment.yaml (update your username as mentioned Fig 12).

```
spec:
  containers:
    - name: devops-practice
      image: <Your DockerHub UserName>/devops_practice:{{ .Values.image.version }}
      imagePullPolicy: Always
      ports:
        - containerPort: 3000
          name: devops-practice
      imagePullSecrets:
        - name: regcred
```

Fig 12

As you can see dockerHub credentials will be stored in secrets name regcred (Fig 12 imagePullSecrets) so run the below command with your dockerHub credentials to create it so argoCd can pull latest images (earlier we put credential in Jenkins to push the image).

```
kubectl create secret docker-registry regcred -- docker-
server=https://index.docker.io/v1/ -- docker-username=<your-name> --
docker-password=<your-pword> -- docker-email=<your-email>
```

Run below commands to build your charts:

```
helm package helm -d charts
```

```
helm repo index charts
```

This will create charts zip and index file in charts folder which you can upload to the cloud storage bucket created above (Fig 13 here i have multiple charts you will have one only first time).

Either you can upload it manually in GCP bucket or write a script to combine the steps.

| <input type="checkbox"/> | Name | Size | Type | Created | Storage class | Last modified | Public access | Version hi |
|--------------------------|---------------------------|--------|--------------------|---------------|---------------|---------------|--------------------|------------|
| <input type="checkbox"/> | devops-practice-0.2.0.tgz | 2.7 KB | application/gzip | Jan 15, 20... | Standard | Jan 15, 20... | Public to internet | Copy URL |
| <input type="checkbox"/> | devops-practice-1.3.0.tgz | 2.7 KB | application/gzip | Jan 15, 20... | Standard | Jan 15, 20... | Public to internet | Copy URL |
| <input type="checkbox"/> | devops-practice-1.3.1.tgz | 2.7 KB | application/gzip | Jan 16, 20... | Standard | Jan 16, 20... | Public to internet | Copy URL |
| <input type="checkbox"/> | devops-practice-1.3.2.tgz | 2.1 KB | application/gzip | Jan 16, 20... | Standard | Jan 16, 20... | Public to internet | Copy URL |
| <input type="checkbox"/> | devops-practice-2.3.1.tgz | 2.7 KB | application/gzip | Jan 16, 20... | Standard | Jan 16, 20... | Public to internet | Copy URL |
| <input type="checkbox"/> | devops-practice-2.3.2.tgz | 2.7 KB | application/gzip | Jan 16, 20... | Standard | Jan 16, 20... | Public to internet | Copy URL |
| <input type="checkbox"/> | devops-practice-2.3.3.tgz | 2.8 KB | application/gzip | Jan 24, 20... | Standard | Jan 24, 20... | Public to internet | Copy URL |
| <input type="checkbox"/> | index.yaml | 2.9 KB | application/x-yaml | Jan 24, 20... | Standard | Jan 24, 20... | Public to internet | Copy URL |

Fig 13

Install ArgoCD on GCP

Here we will host ArgoCd on GCP and connect ArgoCd to sync our git chart repo.

Follow this [link](#) steps 1,3(Service Type Load Balancer) and 4 run the mentioned commands on google cloud shell.

If everything successful, you should see this (Fig 13) in GCP Dashboard > kubernetesEngine > service&ingress.

Click on the Ip External load balancer (Fig 13) and use your Ip password from above steps to login in ArgoCd.

| | | | | | | | |
|--------------------------|-----------------------|------|------------------------|------------|-----|--------|-------------|
| <input type="checkbox"/> | argocd-dex-server | ✓ OK | Cluster IP | [REDACTED] | 1/1 | argocd | k8s-cluster |
| <input type="checkbox"/> | argocd-metrics | ✓ OK | Cluster IP | [REDACTED] | 1/1 | argocd | k8s-cluster |
| <input type="checkbox"/> | argocd-redis | ✓ OK | Cluster IP | [REDACTED] | 1/1 | argocd | k8s-cluster |
| <input type="checkbox"/> | argocd-repo-server | ✓ OK | Cluster IP | [REDACTED] | 1/1 | argocd | k8s-cluster |
| <input type="checkbox"/> | argocd-server | ✓ OK | External load balancer | [REDACTED] | 1/1 | argocd | k8s-cluster |
| <input type="checkbox"/> | argocd-server-metrics | ✓ OK | Cluster IP | [REDACTED] | 1/1 | argocd | k8s-cluster |

Fig 14

Configure ArgoCd to Sync with Git Chart Repo

Once logged in, click 'new app'.

- give a name
- project = default
- Sync policy = automatic
- In check prune resource repository URL = git chart repo(mine: <https://github.com/aashirwad3may/gitops.git>)
- path = argocd/devopsPractice
- clusterUrl = <https://kubernetes.default.svc>
- nameSpace=default
- values.files = values.yaml

Hit create you see a beautiful UI for your system (figure 15) after it has synchronised and created the pods .

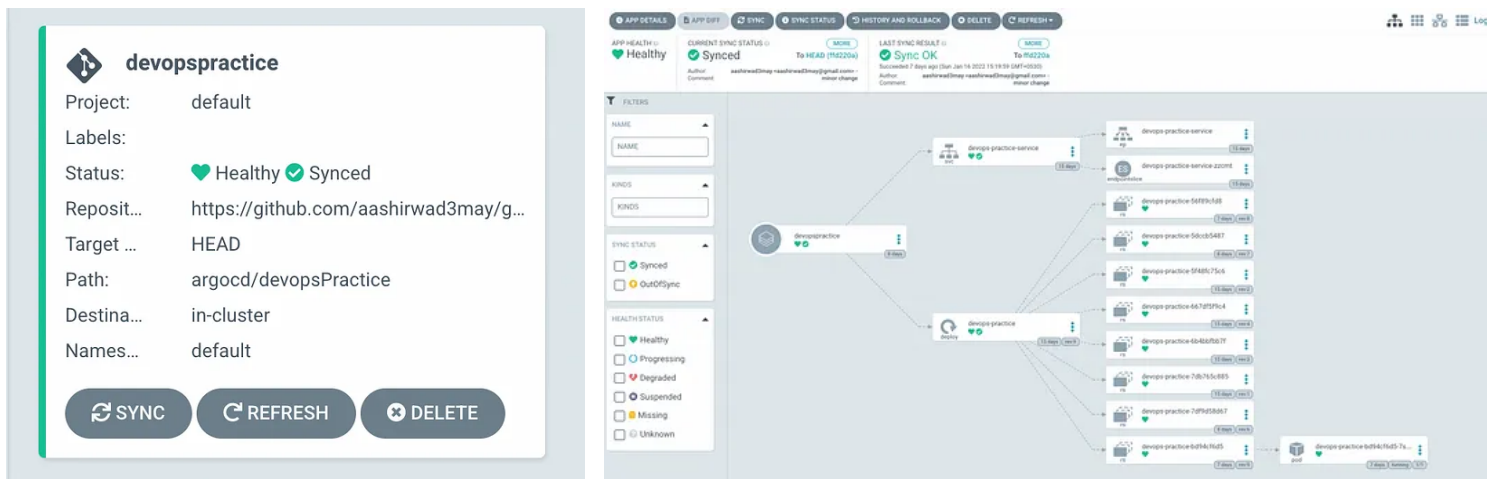


Fig 15

Go ahead and change replica count in git chart repo values.yaml and you can see changes being reflected. It takes some time to sync you can click on refresh to sync immediately.

It is recommended to upgrade chart version if you make any changes in your helm template. Upload new chart to bucket and update the version in git chart repo argoCd will sync relevant changes.

Conclusion

And there we have it. You should now have a working Node.js app set up with Docker, with a full CI/CD build pipeline in place. Please let me know if anything I have missed or can be added in the comments. For more backend related content follow me on [Twitter](#), [LinkedIn](#).

DevOps

Kubernetes

Docker

Nodejs

JavaScript



Follow




Written by Aashirwad Kashyap

83 Followers · Writer for Bits and Pieces

Backend developer and technology enthusiasts

More from Aashirwad Kashyap and Bits and Pieces



 Aashirwad Kashyap in Geek Culture

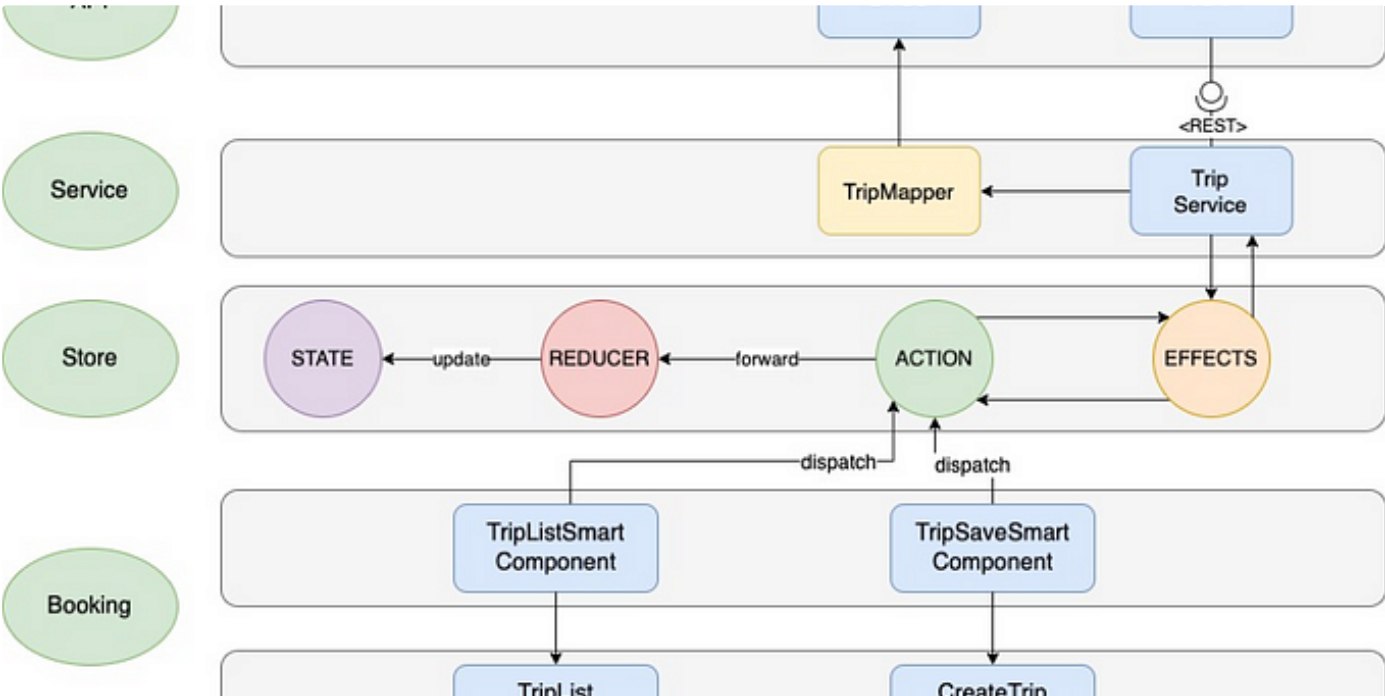
Building Resilient Data Pipelines with deep dive into AirFlow architecture

Goal: To Build a resilient data pipeline from scratch using Airflow, Hive (HDFS query analysis) and Superset (data visualisation)

8 min read · Dec 29, 2022

 5 







Robert Maier-Silddorff in Bits and Pieces

Clean Frontend Architecture

An overview of some of the principles associated with a clean frontend architecture (SOLID, KISS, DRY, and more).

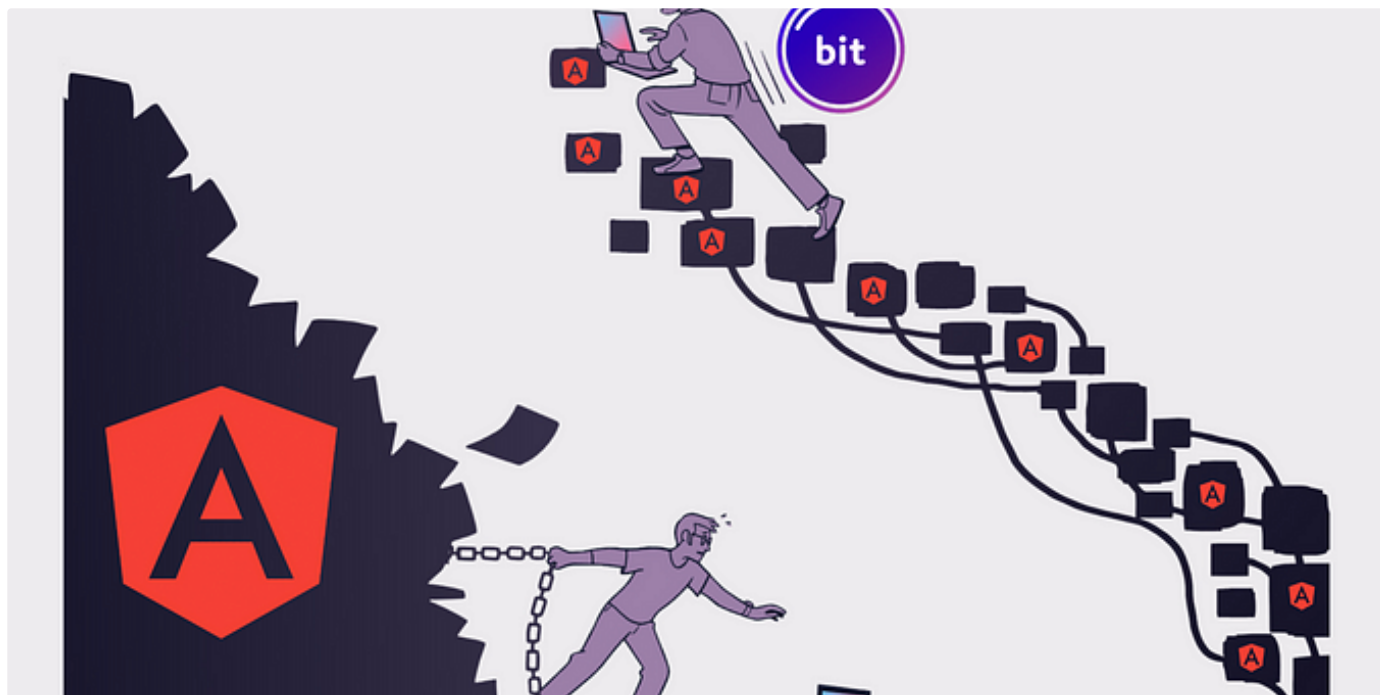
6 min read · Jul 14



1.94K



17



Olivier Combe in Bits and Pieces

A Modern Approach for Angular Development

Build composable and modernized Angular apps with Bit!

8 min read · Nov 23

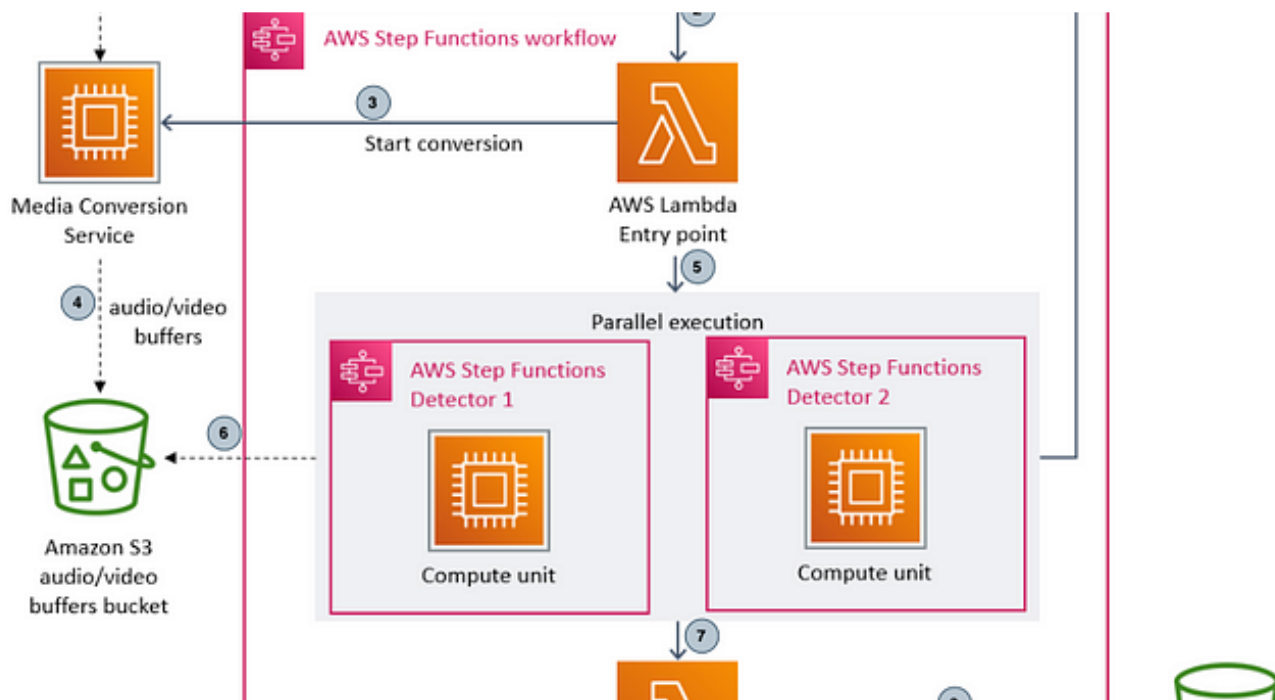


889



3





 Aashirwad Kashyap in Level Up Coding

Amazon Prime Video's Move to Monolithic Architecture: Is It Worth the Risk?

Cutting Costs or Sacrificing Flexibility?

5 min read · May 7

 421  1



See all from Aashirwad Kashyap

See all from Bits and Pieces

Recommended from Medium



 Niranjan Gawali in Globant

Kubernetes Deployment: Automating NodeJS Deployment on GKE with GitHub Actions

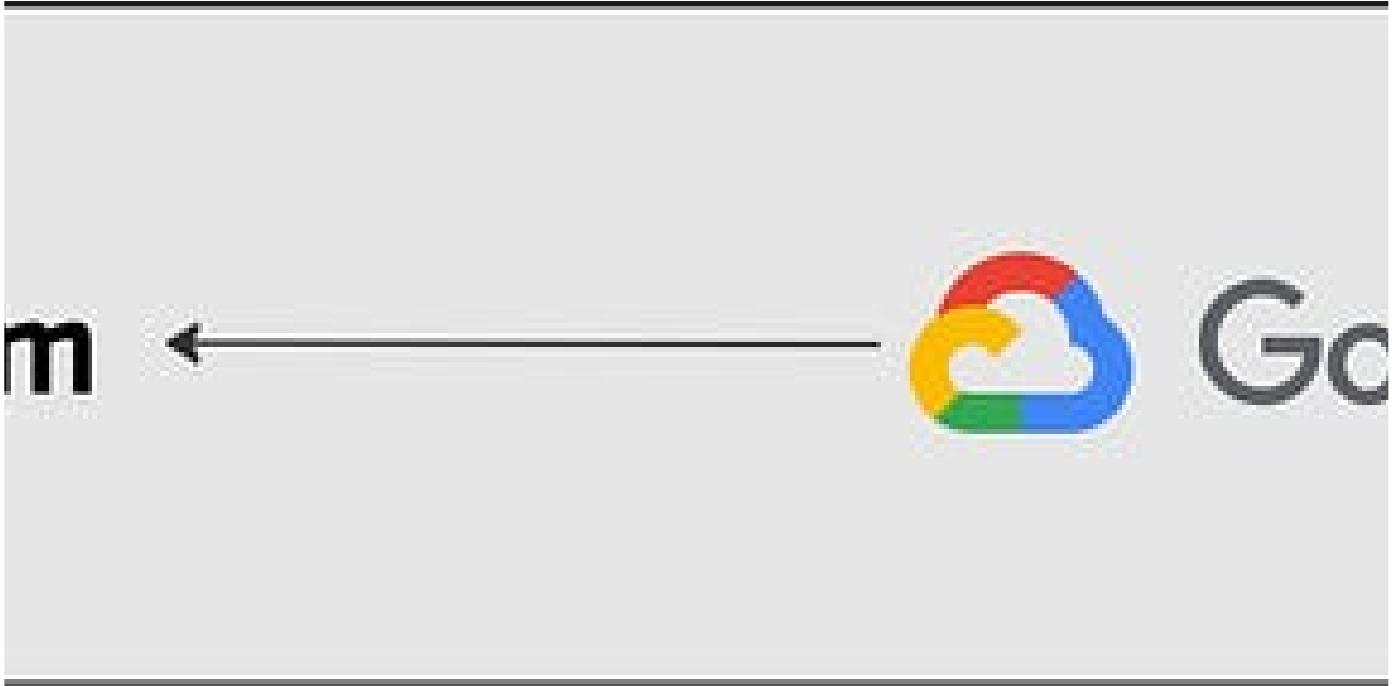
In the Kubernetes deployment series, we have already covered two topics:

10 min read · Jul 25



8





Rohan Paithankar in Google Cloud - Community

Exporting GCP Projects to Terraform

4 min read · Nov 29



45



Lists



Stories to Help You Grow as a Software Developer

19 stories · 615 saves



General Coding Knowledge

20 stories · 655 saves



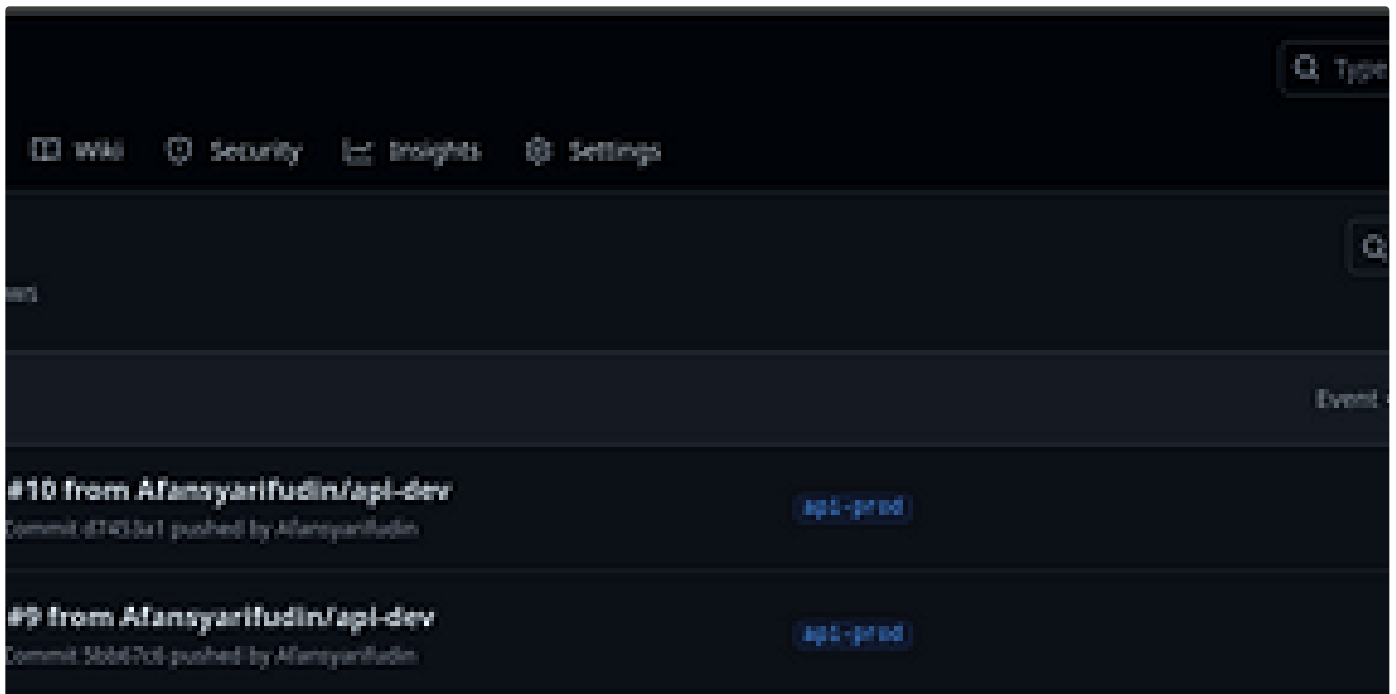
Coding & Development


11 stories · 309 saves



New_Reading_List

174 stories · 221 saves



 Afan Syarifudin in DevOps.dev

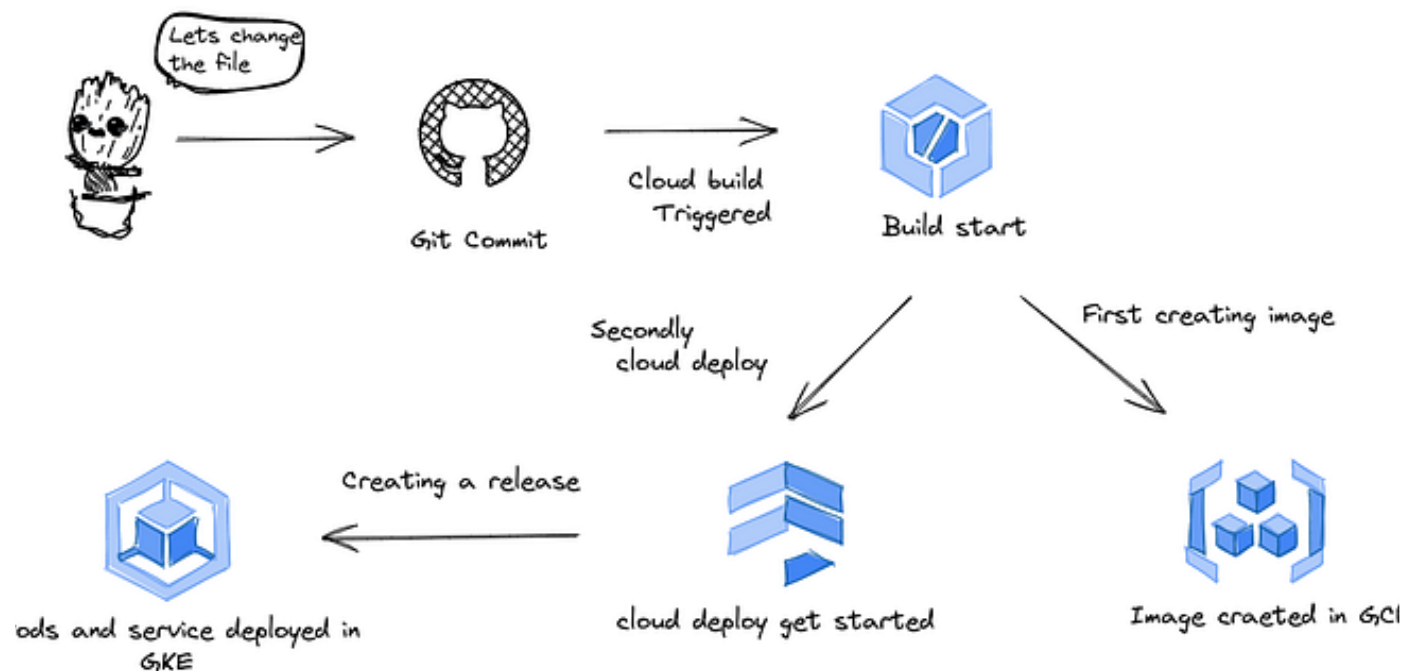
Automation Deployment to Cloud Run using Github Actions (NodeJs and Docker) — Bangkit Academy 2023...


Hello everyone! In this article, I will share my experience of creating an automation deployment using GitHub Actions and deploying it to...

7 min read · Jul 8

 3 

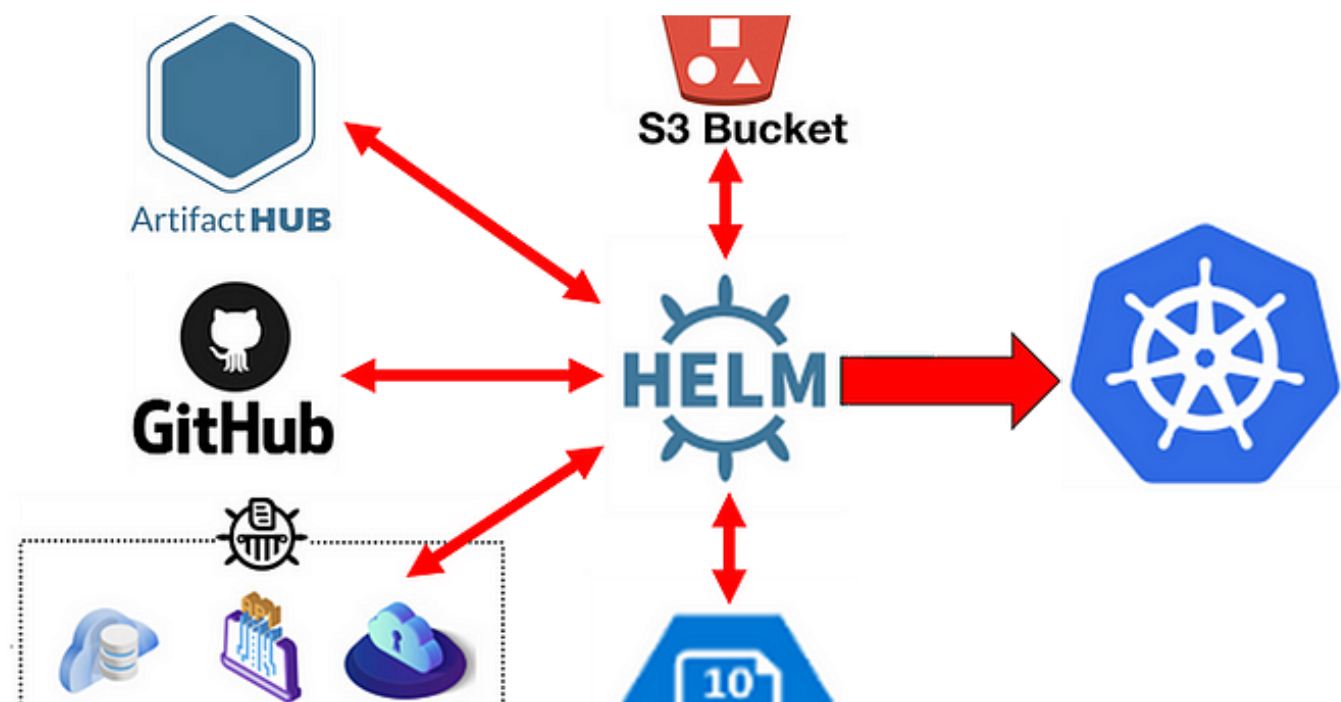





 Harieshkumar in The Cloudside View

Cloud Deploy Integrate's With Cloud Build

3 min read · Jul 7



 Cumhur Akkaya

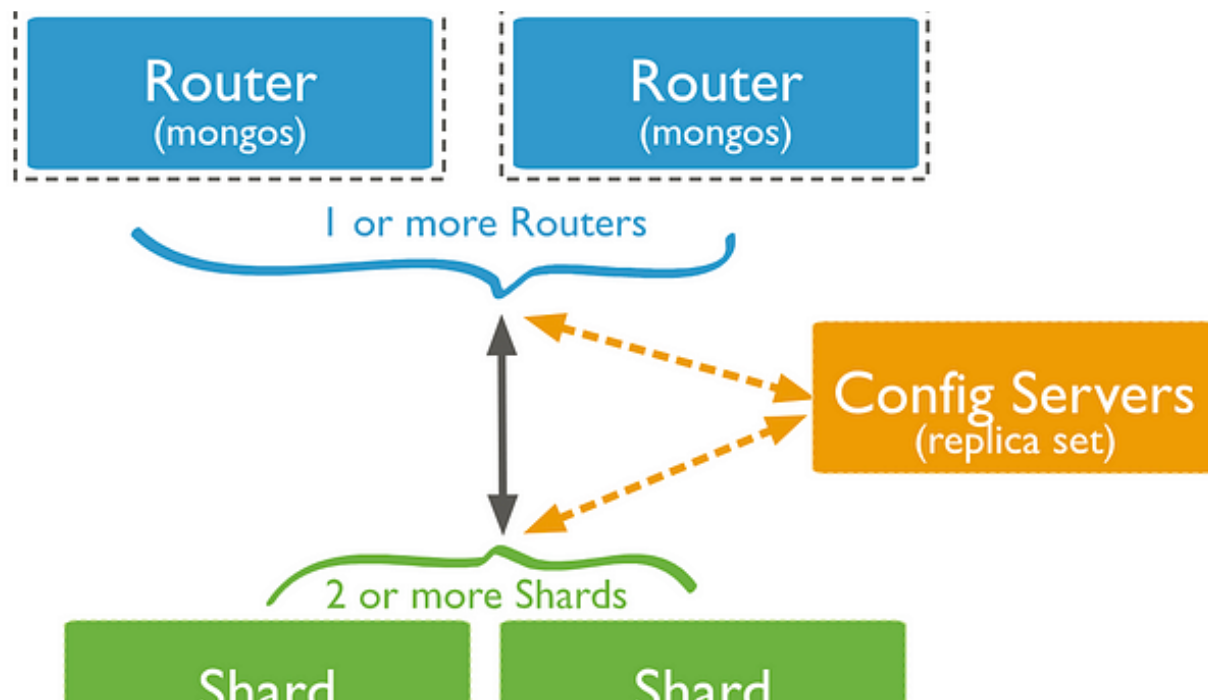
Helm-2: Deploying an App into GKE via the Helm by Using GitHub as a Helm Chart Repository

We will learn detailed information about the Chart Repository. Then, we will set up a Chart Repository in GitHub, turn Kubernetes manifest...

10 min read · 6 days ago



50



 Tanmay Bhandge

MongoDB: From Basics to Deployment on Kubernetes

MongoDB, often hailed as the “NoSQL database for the modern age,” has taken the world of data management by storm. It’s known for its...

12 min read · Sep 24



2



See more recommendations