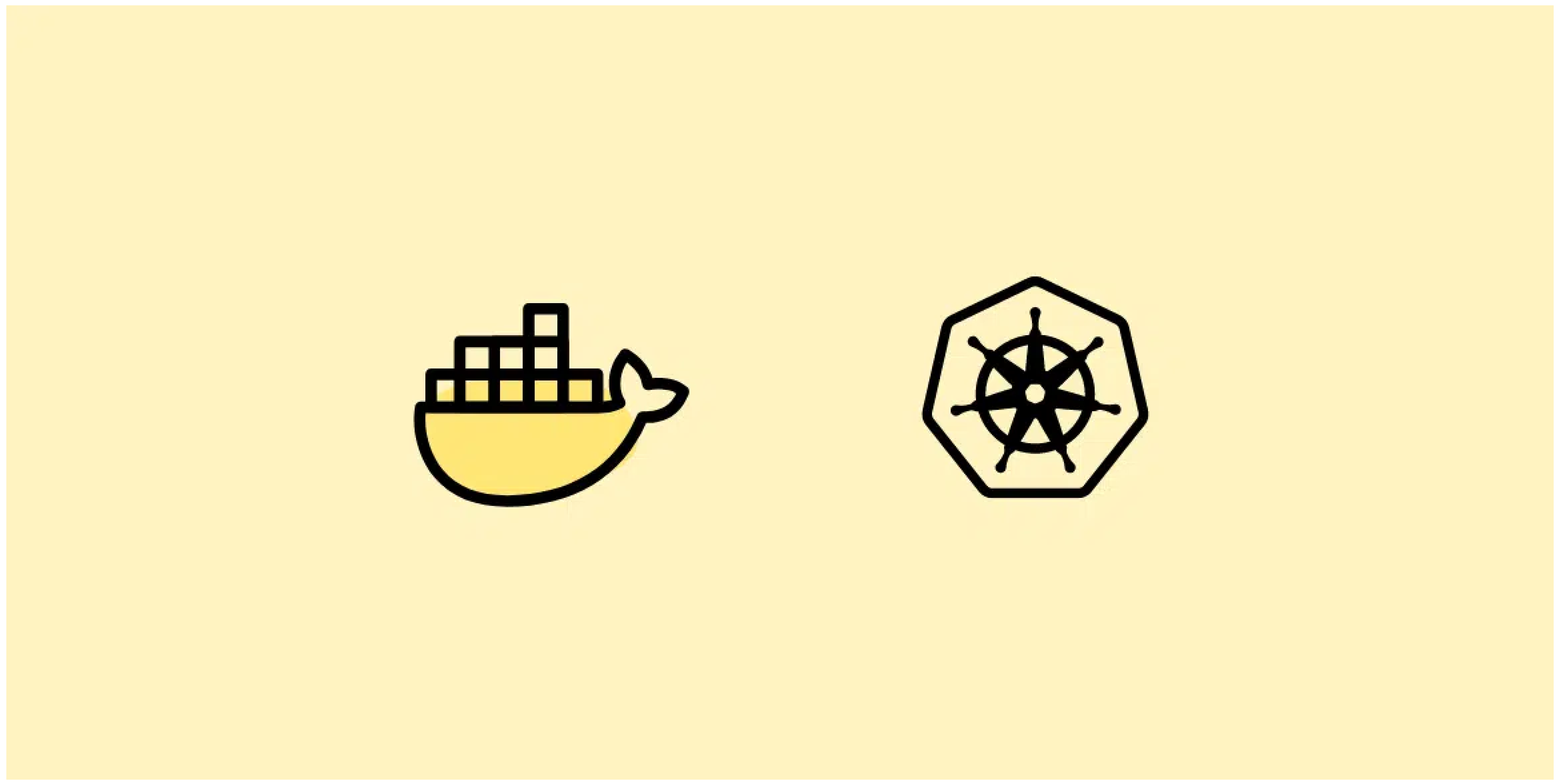
K — KUBERNETES

How To Build Docker Image In Kubernetes Pod

by **devopscube** · July 24, 2021



This beginner's guide focuses on step by step process of setting up Docker image build in Kubernetes pod using Kaniko image builder.

When it comes to CI/CD, there could be VM & container-based applications. Ideally one would use existing VM infrastructure to build Docker images. However, if you have a containerized infrastructure, it is better to utilize it for the CI/CD workflow.

Building Docker in Docker

In CI, one of the main stages is to build the Docker images. In containerized builds, you can use Docker in the Docker workflow. You can check out the Docker in Docker article to understand more.

But this approach has the following disadvantages

1. The Docker build containers runs in priveleged mode. It is a big security concern and it is kind of a open door to malicious attacks.
2. Kubernetes removed Docker from its core. So, mouting `docker.sock` to host will not work in future, unless you add docker to all the Kubernetes Nodes.

These issues can be resolved using Kaniko.

Build Docker Image In Kubernetes Using Kaniko

kaniko is an open-source container image-building tool created by Google.

It does not require privileged access to the host for building container images.

Here is how Kaniko works,

1. There is a dedicated Kaniko executer image which builds the contianer images. It is recomended to use the `gcr.io/kaniko-project/executor` image to avoid any possible issues. Beacuse this image containes only staic go binary and logic to push/pull images from/to registry.
2. kaniko accepts three arguments. A Dockerfile, build context and a remote registry to push the build image.
3. When you deploy kaniko image it reads the Dockerfile and extracts the base image file system using the FROM instruction.
4. The it executes each instruction from the Dockerfile and takes a snapshot in the userspace.
5. After each snapshot, kaniko appends only the changed image layers to the base image and updates the image metadata. It happens for all the instructions in the Dockerfile.
6. Finally, it pushes the image to the given registry.

As you can see, all the image-building operations happen inside the Kaneko container's userspace and it does not require any privileged access to the host.

Kaniko supports the following type of build context.

1. GCS Bucket
2. S3 Bucket
3. Azure Blob Storage

5. Local Tar
6. Standard Input
7. Git Repository

For this blog, I will use the Github repo as a context.

Also, you can push to any container registry.

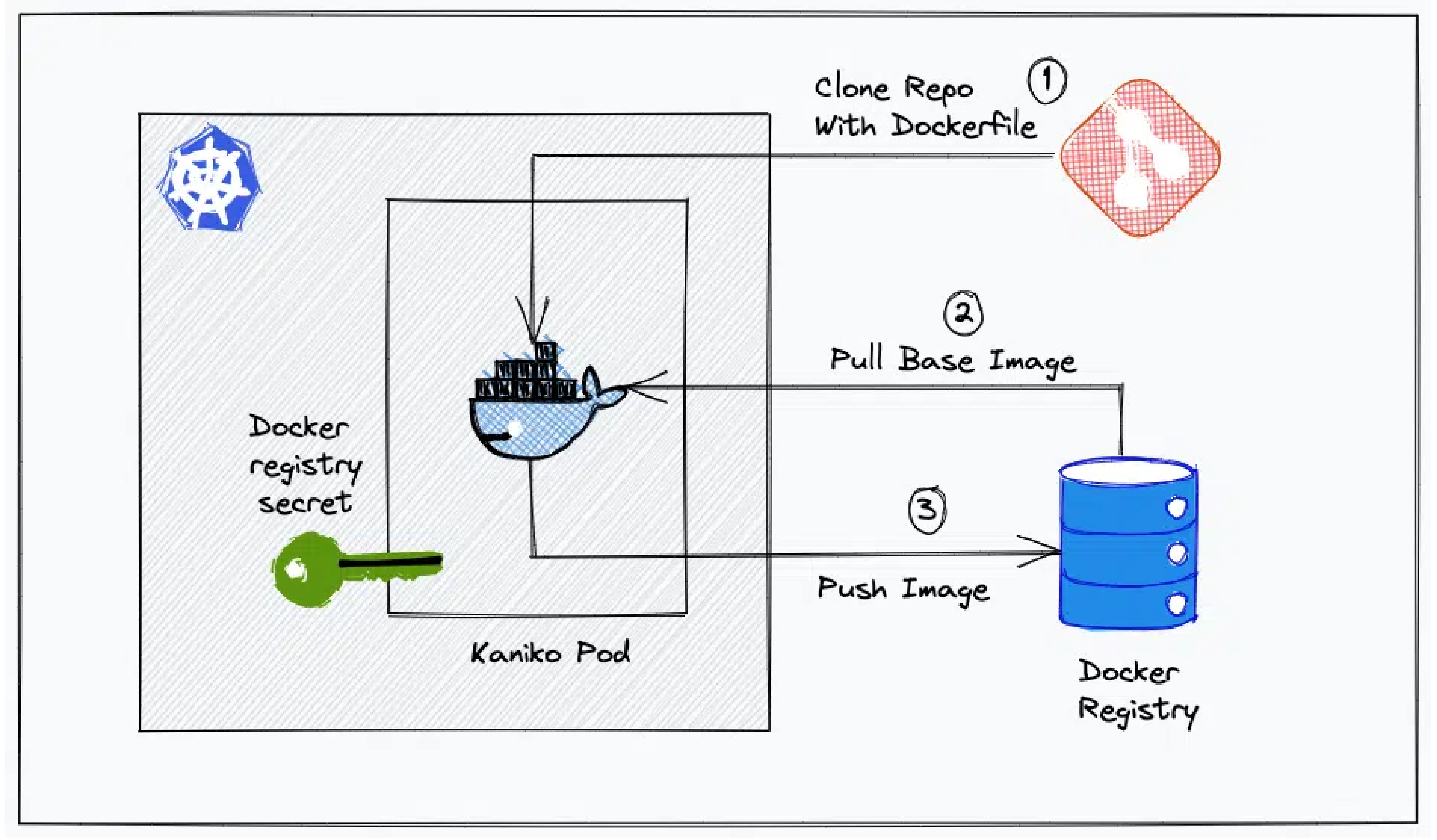
Building Docker Image With, Kaniko, Github, Docker Registry & Kubernetes

To demonstrate the Kaniko workflow I will use publicly available tools to build Docker images on kubernetes using Kaniko.

Here is what you need

1. **A valid Github repo with a Dockerfile:** kaniko will use the repository URL path as the Dockerfile context
2. **A valid docker hub account**: For kaniko pod to autheticate and push the built Docker image.
3. **Access to Kubernetes cluster**: To deploy kaniko pod and create docker registry secret.

The following image shows the workflow we are going to build.



Create Dockerhub Kubernetes Secret

We have to create a kubernetes secret type `docker-registry` for the kaniko pod to authenticate the Docker hub registry and push the image.

Use the following command format to create the docker registry secret. Replace the parameters marked in bold.

```
kubectl create secret docker-registry dockercred \
    --docker-server=https://index.docker.io/v1/ \
    --docker-username=<dockerhub-username> \
    --docker-password=<dockerhub-password>\
    --docker-email=<dockerhub-email>
```

This secret will be mounted to the kaniko pod for it to authenticate the Docker registry to push the built image.

Note: If you have a self hosted docker regpository, you can replace teh server URL wiht your docker registry API endpoint.

Now we will test the kaniko image builder using a pod deployment.

I have hosted the manifest and Dockerfile in the public Github repository. It is a simple Dockerfile with an update instruction.

I will use that repository for demonstration. You can fork it or create your own repo with similar configurations.

```
https://github.com/scriptcamp/kubernetes-kaniko
```

Save the following manifest as `pod.yaml`

```
apiVersion: v1
kind: Pod
metadata:
  name: kaniko
spec:
  containers:
  - name: kaniko
    image: gcr.io/kaniko-project/executor:latest
    args:
    - "--context=git://github.com/scriptcamp/kubernetes-kaniko"
    - "--destination=<dockerhub-username>/kaniko-demo-image:1.0"
    volumeMounts:
    - name: kaniko-secret
      mountPath: /kaniko/.docker
  restartPolicy: Never
  volumes:
  - name: kaniko-secret
    secret:
      secretName: dockercred
      items:
        - key: .dockerconfigjson
          path: config.json
```

1. **–context:** This is the location of the Dockerfile. In our case, the Dockerfile is lcoation in the root of the repository. So I have give the git URL of the repository. If you are using a private git repository, then you can use `GIT_USERNAME` and `GIT_PASSWORD` (API token) variables to authenticate git repository.
2. **–destination:** Here you need to replace `<dockerhub-username>` with your docker hub username with your dockerhub username for kaniko to able to push the image to the dockerhub registry. For example, in my case its, `bibinwilson/kaniko-test-image:1.0`

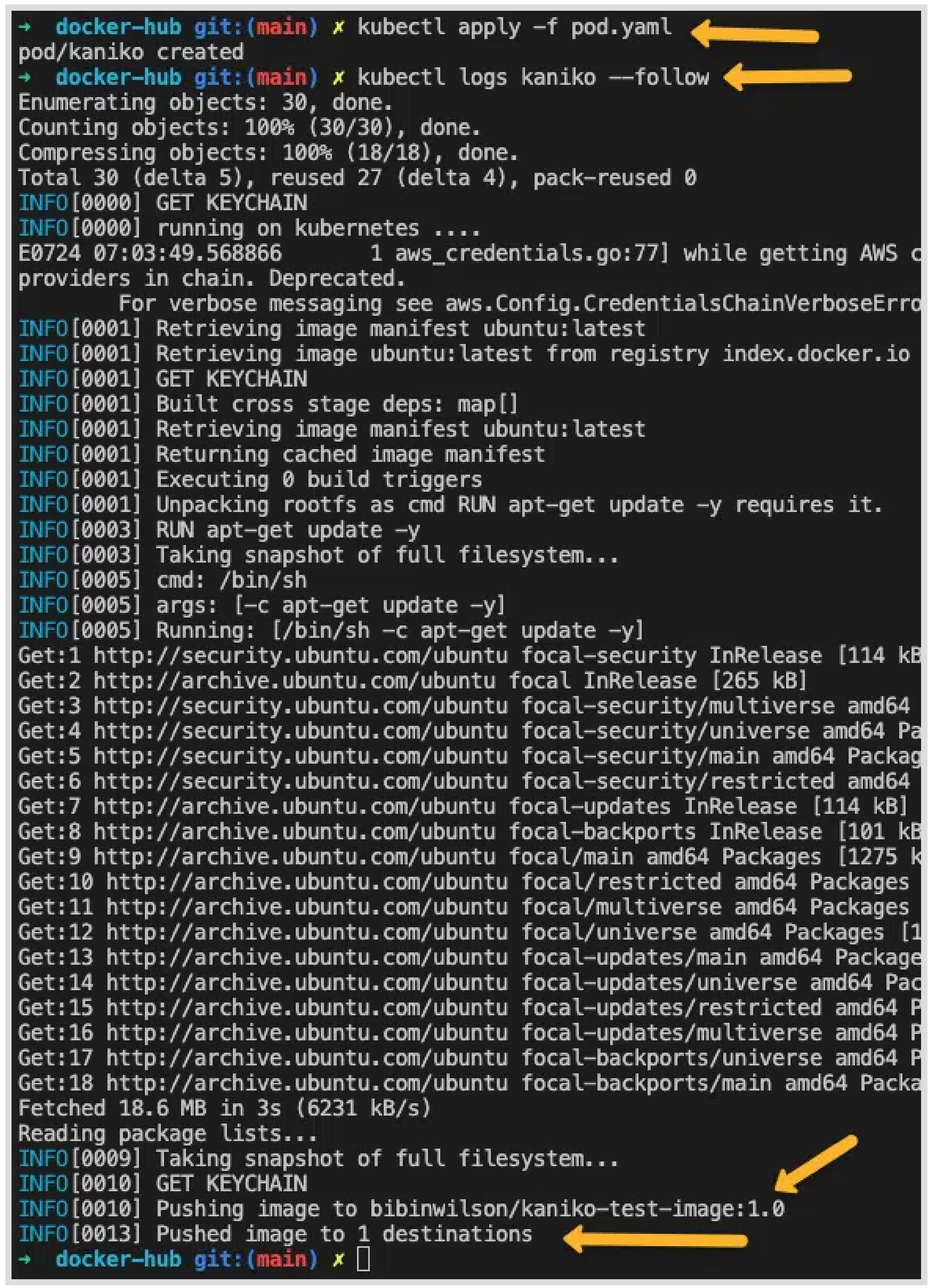
All the other configurations remain the same.

Now deploy the pod.

```
kubectl apply -f pod.yaml
```

To validate the docker image build and push, check the pod logs.

```
kubectl logs kaniko --follow
```



```
→  docker-hub git:(main) ✗ kubectl apply -f pod.yaml
pod/kaniko created
→  docker-hub git:(main) ✗ kubectl logs kaniko --follow
Enumerating objects: 30, done.
Counting objects: 100% (30/30), done.
Compressing objects: 100% (18/18), done.
Total 30 (delta 5), reused 27 (delta 4), pack-reused 0
INFO[0000] GET KEYCHAIN
INFO[0000] running on kubernetes ....
E0724 07:03:49.568866       1 aws_credentials.go:77] while getting AWS c
providers in chain. Deprecated.
        For verbose messaging see aws.Config.CredentialsChainVerboseErro
INFO[0001] Retrieving image manifest ubuntu:latest
INFO[0001] Retrieving image ubuntu:latest from registry index.docker.io
INFO[0001] GET KEYCHAIN
INFO[0001] Built cross stage deps: map[]
INFO[0001] Retrieving image manifest ubuntu:latest
INFO[0001] Returning cached image manifest
INFO[0001] Executing 0 build triggers
INFO[0001] Unpacking rootfs as cmd RUN apt-get update -y requires it.
INFO[0003] RUN apt-get update -y
INFO[0003] Taking snapshot of full filesystem...
INFO[0005] cmd: /bin/sh
INFO[0005] args: [-c apt-get update -y]
INFO[0005] Running: [/bin/sh -c apt-get update -y]
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB
Get:2 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:3 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64
Get:4 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Pa
Get:5 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packag
Get:6 http://security.ubuntu.com/ubuntu focal-security/restricted amd64
Get:7 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:8 http://archive.ubuntu.com/ubuntu focal-backports InRelease [101 kB
Get:9 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages [1275 k
Get:10 http://archive.ubuntu.com/ubuntu focal/restricted amd64 Packages
Get:11 http://archive.ubuntu.com/ubuntu focal/multiverse amd64 Packages
Get:12 http://archive.ubuntu.com/ubuntu focal/universe amd64 Packages [1
Get:13 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 Package
Get:14 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 Pac
Get:15 http://archive.ubuntu.com/ubuntu focal-updates/restricted amd64 P
Get:16 http://archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 P
Get:17 http://archive.ubuntu.com/ubuntu focal-backports/universe amd64 P
Get:18 http://archive.ubuntu.com/ubuntu focal-backports/main amd64 Packa
Fetched 18.6 MB in 3s (6231 kB/s)
Reading package lists...
INFO[0009] Taking snapshot of full filesystem...
INFO[0010] GET KEYCHAIN
INFO[0010] Pushing image to bibinwilson/kaniko-test-image:1.0
INFO[0013] Pushed image to 1 destinations
→  docker-hub git:(main) ✗
```

Note: Here we used a static pod name. So to deploy again, you first you have to delete the kaniko pod. When kaniko is used in project CI/CD pipeline, the pod will get a random name based on the CI tool you use and it takes care of deleting the pod.

Docker Build Pipeline Using Jenkins & kaniko on Kubernetes

If you are using Kubernetes for scaling Jenkins build agents, you can make use of Kaniko docker build pods to build the docker images in the CI pipeline.

You can check out my Jenkins build agent setup on Kubernetes where the Jenkins master and agent runs on the kubernetes cluster.

To leverage Kaniko for your build pipelines, you should have the Dockerfile along with the application.

Also, you should use the multi-container pod template with a build and kaniko container. For example, maven containers for java build and kaniko containers to take the jar and build the docker image using the Dockerfile present in the repository.

Here is a Jenkinsfile based on a multi-container pod template where you can build your application and use the kaniko container to build the docker image with the application and push it to a Docker registry.

Important Note: You should use the kaniko image with the debug tag in the pod template becuase we will be explicilty running the kaniko executer using bash. The latest tag images does not have bash

```
podTemplate(yaml: '''
    apiVersion: v1
    kind: Pod
    spec:
      containers:
      - name: maven
        image: maven:3.8.1-jdk-8
        command:
        - sleep
        args:
        - 99d
      - name: kaniko
        image: gcr.io/kaniko-project/executor:debug
        command:
        - sleep
        args:
        - 9999999
        volumeMounts:
        - name: kaniko-secret
          mountPath: /kaniko/.docker
      restartPolicy: Never
      volumes:
      - name: kaniko-secret
        secret:
            secretName: dockercred
            items:
            - key: .dockerconfigjson
              path: config.json
''') {
  node(POD_LABEL) {
    stage('Get a Maven project') {
      git url: 'https://github.com/scriptcamp/kubernetes-kaniko.git', branch:
'main'
      container('maven') {
        stage('Build a Maven project') {
          sh '''
        }
      }

      stage('Build Java Image') {
        container('kaniko') {
          stage('Build a Go project') {
            sh '''
              /kaniko/executor --context `pwd` --destination bibinwilson/hello-
kaniko:1.0
            '''
          }
        }
      }

    }
  }
```

You can use the above Jenkinsfile directly on a pipeline job and test it. It is just a template to get started. You need to replace the repo with your code repo and write the build logic as per the application's needs.

Conclusion

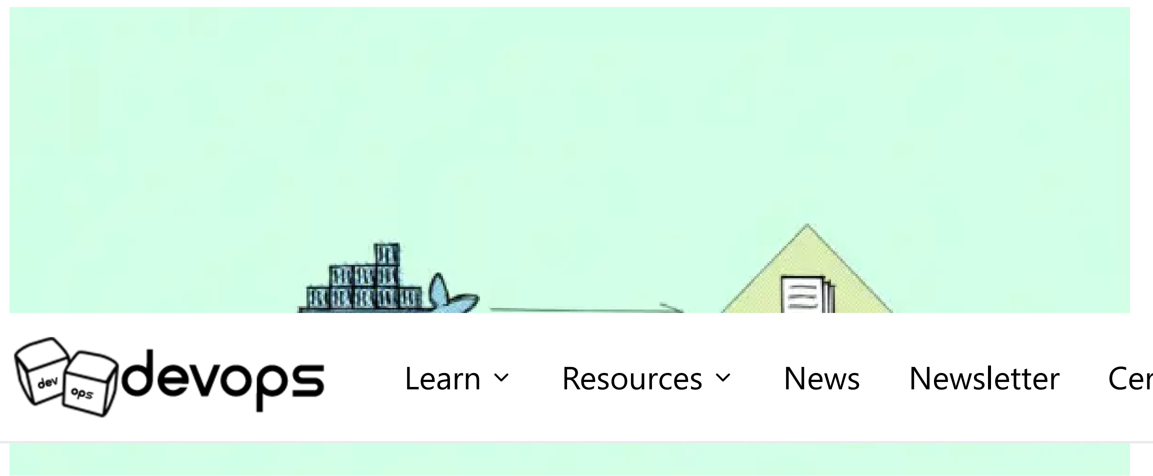
Building Docker images using kaniko is a secure way to containerized Docker builds.

You can try incorporating kaniko with your pipelines without compromising security.

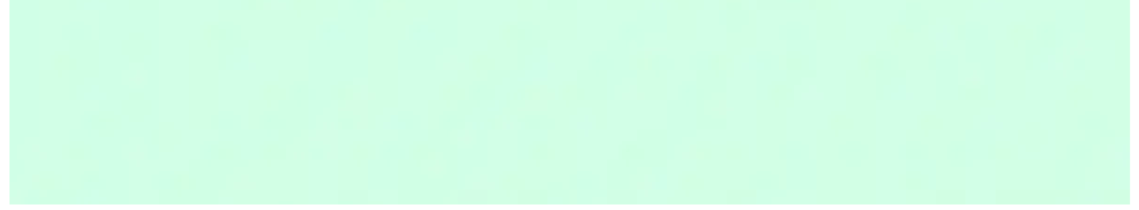
Also, let me know what you think of this approach.

devopscube

Established in 2014, a community for developers and system admins. Our goal is to continue to build a growing DevOps community offering the best in-depth articles, interviews, event listings, whitepapers, infographics and much more on DevOps.



by-step guides to set up a...

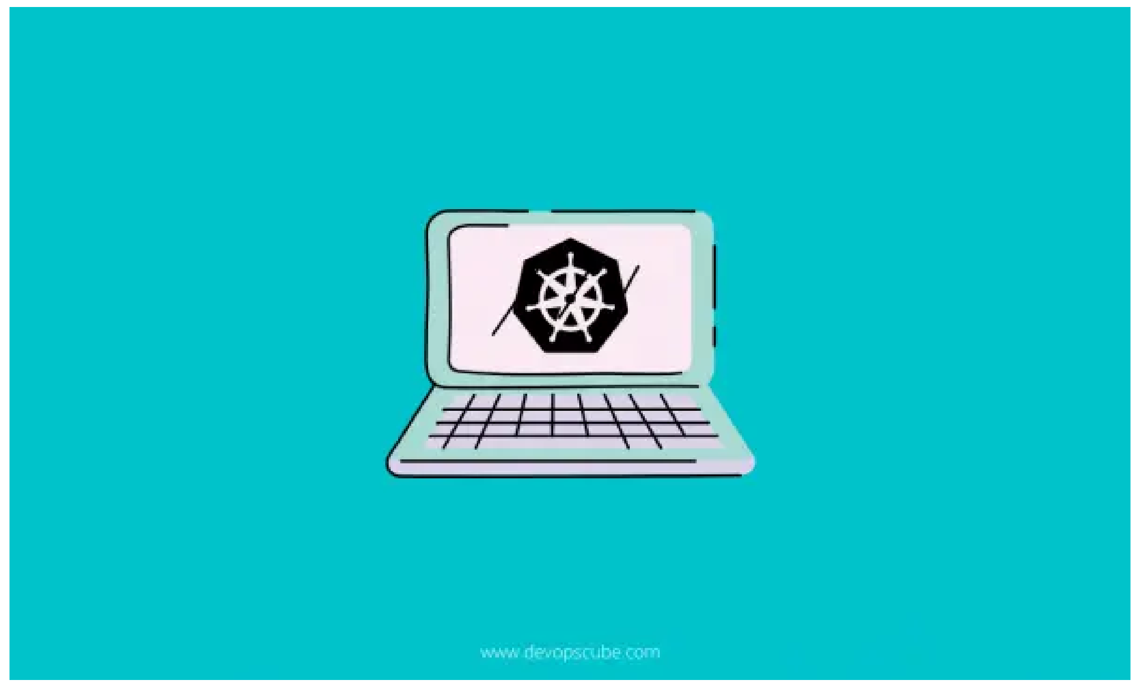


D — DEVOPS

Setting Up Alert Manager on Kubernetes – Beginners Guide

by **devopscube** · March 10, 2021

AlertManager is an open-source alerting system that works with the Prometheus Monitoring system. In the last article, I...

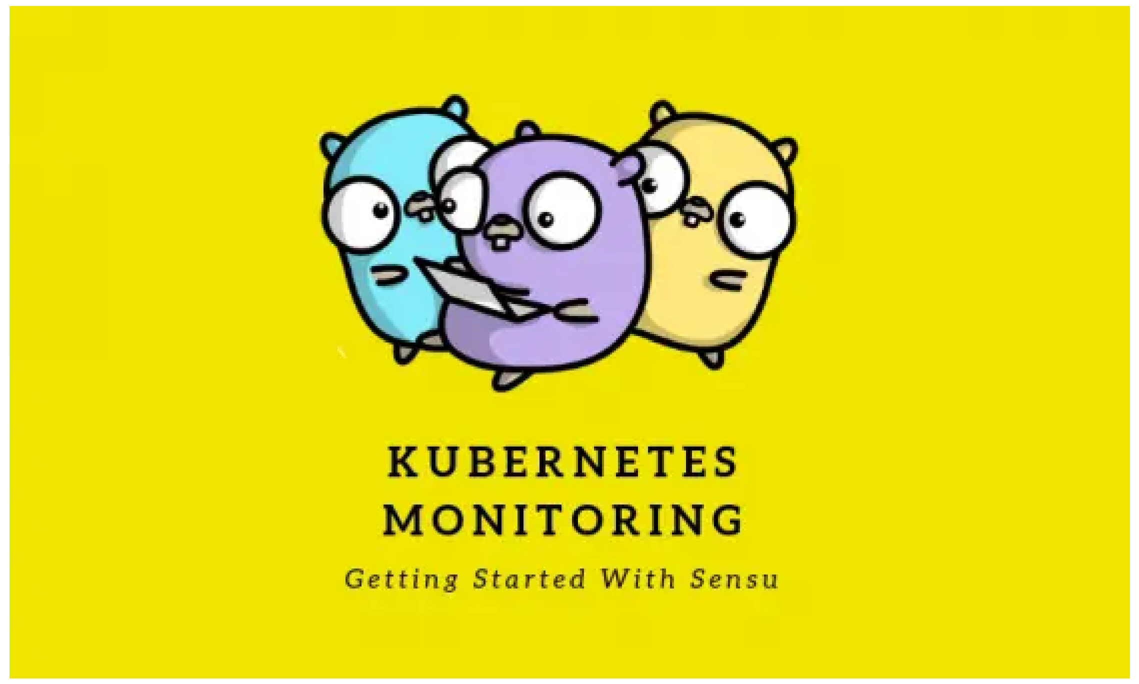


K — KUBERNETES

How to Setup Kubernetes Cluster on Vagrant VMs

by **Bibin Wilson** · May 19, 2021

In this post, I have covered the step-by-step guide to setup Kubernetes cluster on Vagrant. It is a...



K — KUBERNETES

Kubernetes monitoring With Sensu: Setting up Container Sidecar Agent

by **Jef Spaleta** · August 27, 2019

The rise of containerized infrastructure has caused us to rethink the way we build and deploy our applications....

K — KUBERNETES

How to Add Persistent Volume in Google Kubernetes Engine

In this blog, you will learn how to setup Persistent Volume For the GKE Kubernetes cluster. If you...

K — KUBERNETES

CKS Exam Study Guide: Resources to Pass Certified Kubernetes Security Specialist

by **Bibin Wilson** · June 23, 2021

In this Certified Kubernetes Security Specialist (CKS) Exam study guide, I have listed all the resources you can...