Published in Dev Genius

Tony Follow

Jan 8 · 4 min read · ✨ · Listen

K8s — Kaniko Introduction

What is kaniko and why to use it?

Kaniko

What is kaniko

Kaniko is a tool developed by Google to help build docker container images in Kubernetes. It is an application wi 13 doesn't depend on a Docker

daemon. The first release of kaniko (v0.1.0) was on May 17, 2018. The latest release as of today is v1.9.1.

> *Note: kaniko is not an officially supported Google product.*

kaniko is meant to be run as an image: gcr.io/kaniko-project/executor. I do not recommend running the kaniko executor binary in another image, as it might not work.

Why kaniko

Same as docker build, kaniko also builds container images from a Dockerfile, but the build process can be inside a container that running in K8s.

Kaniko doesn't depend on a Docker daemon and executes each command within a Dockerfile completely in userspace. This enables building container images in environments that can't easily or securely run a Docker daemon, such as a standard K8s.

Kaniko is a lightweight tool that doesn't require as many permissions and privileges as Docker, and more importantly, it doesn't require a running Docker service. Gitlab notably recommends it for building containers on K8s:

kaniko solves two problems with using the Docker-in-Docker build method:

- **Docker-in-Docker requires privileged mode to function, which is a significant security concern.**
- **Docker-in-Docker generally incurs a performance penalty and can be quite slow.**

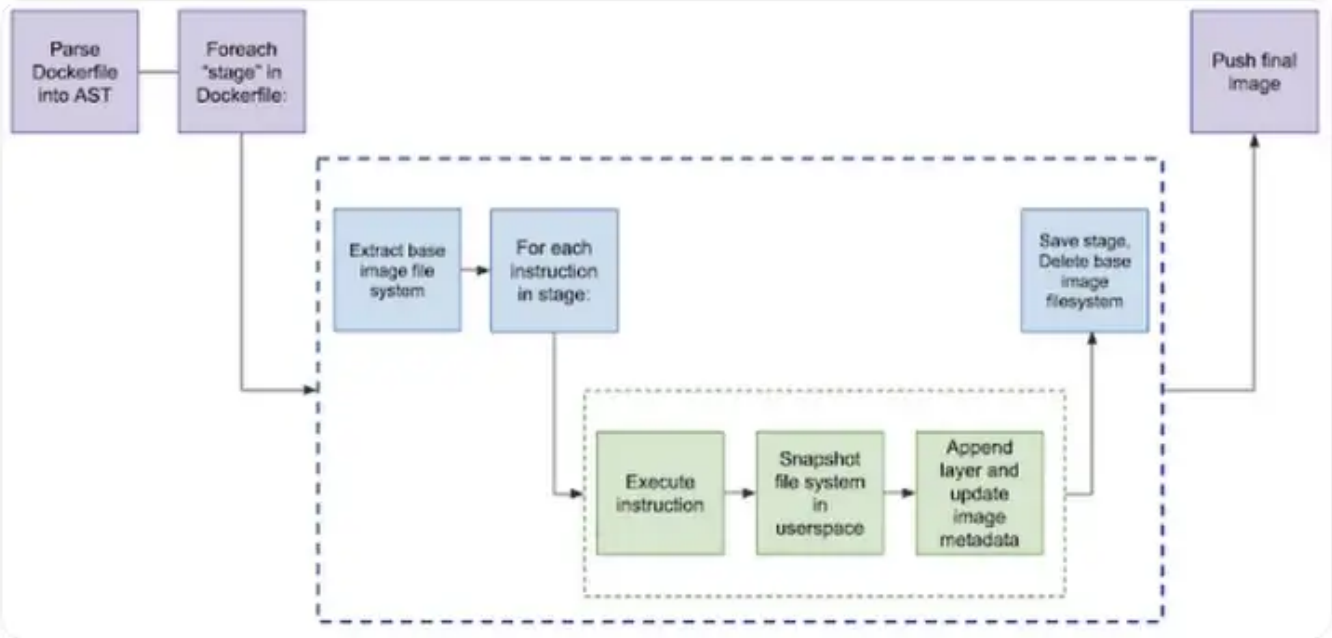
How does kaniko Work?

kaniko runs as a container and takes in the following three arguments:

- **Dockerfile**
- **A build context**
- **The name of the registry to which it should push the final image**

The build process looks like:

- The kaniko executor image is responsible for building an image from a `Dockerfile` and pushing it to a registry.
- Within the executor image, it extracts the filesystem of the base image (the FROM image in the `Dockerfile`) to a local directory (e.g./kaniko/{imagename}).
- It then execute the commands in the `Dockerfile`, snapshotting the filesystem in userspace after each one.



Pic from google cloud

- After each command, it append a layer of changed files to the base image (if there are any) and update image metadata.

Kaniko Components

kaniko has two container types, one is base container and the other is debug container. kaniko builds images through the `executor` command, for example:

```
$ /kaniko/executor --context test-app/ --dockerfile Dockerfile --destination <-
```

You can choose to either generate a `tar` format of the image, of directly push the image to your registry.

pod.yml

This file is for starting a kaniko container to build the example image.

```
apiVersion: v1
kind: Pod
metadata:
  name: kaniko
spec:
  containers:
  - name: kaniko
    image: gcr.io/kaniko-project/executor:latest
    args: ["--dockerfile=/workspace/dockerfile",
           "--context=dir://workspace",
           "--destination=<user-name>/<repo>"] # replace with your dockerhub a
    volumeMounts:
      - name: kaniko-secret
        mountPath: /kaniko/.docker
      - name: dockerfile-storage
        mountPath: /workspace
  restartPolicy: Never
  volumes:
    - name: kaniko-secret
      secret:
        secretName: regcred
        items:
          - key: .dockerconfigjson
            path: config.json
    - name: dockerfile-storage
      persistentVolumeClaim:
        claimName: dockerfile-claim
```

volume.yml

This file is for creating a PV which will be used as kaniko build context.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: dockerfile
  labels:
    type: local
spec:
  capacity:
    storage: 10Gi
  accessModes:
```

```
    - ReadWriteOnce
  storageClassName: local-storage
  hostPath:
    path: <local-directory> # replace with local directory, such as "/home/<use
```

volume-claim.yml

This file is for creating a persistent volume claim which will mounted in the kaniko container.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: dockerfile-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 8Gi
  storageClassName: local-storage
```

kaniko Local Build Demo

Prepare dockerfile

Create a simple dockerfile in a local directory

```
$ mkdir kaniko && cd kaniko
$ echo 'FROM ubuntu' >> dockerfile
$ echo 'ENTRYPOINT ["/bin/bash", "-c", "echo hello"]' >> dockerfile
$ cat dockerfile
FROM ubuntu
ENTRYPOINT ["/bin/bash", "-c", "echo hello"]
$ pwd
/home/ec2-user/kaniko
```

Create persistenVolume

```
$ kubectl create -f volume.yml
persistentvolume/dockerfile created

$ kubectl get pv
NAME         CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STO
dockerfile   1Gi        RWO            Retain           Available           loc
```

Create persistentVolumeClaim

```
$ kubectl create -f volume-claim.yml
persistentvolumeclaim/dockerfile-claim created

$ kubectl get pvc
NAME               STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS
dockerfile-claim   Pending                                      local-storage
```

Create kaniko Pod

```
$ kubectl create -f pod.yml
pod/kaniko created

$ kubectl logs kaniko
INFO[0001] Downloading base image ubuntu
INFO[0001] Skipping unpacking as no commands require it.
INFO[0001] Taking snapshot of full filesystem...
INFO[0001] ENTRYPOINT ["/bin/bash", "-c", "echo hello"]
...
```

In my demo, I used the `--no-push` command for testing purpose only:

```
spec:
  containers:
  - name: kaniko
    image: gcr.io/kaniko-project/executor:latest
    args: ["--dockerfile=/workspace/dockerfile",
```

```
            "--context=dir://workspace",
            "--no-push"] # replace with your dockerhub account
```

If you want to actually push your image to your image registry, make sure you configure the credentials for your registry:

```
$ kubectl create secret docker-registry regcred \
--docker-server=<your-registry-server> \
--docker-username=<your-name> \
--docker-password=<your-pword> \
--docker-email=<your-email>
```

- `--docker-server` : is your Private Docker Registry FQDN. (https://index.docker.io/v1/ for DockerHub)
- `--docker-username` : is your Docker username
- `--docker-password` : is your Docker password
- `--docker-email` : is your Docker email

Reference

For more detailed usages, please visit:
https://github.com/GoogleContainerTools/kaniko

Kubernetes Dev Ops Cloud Computing Docker

Give a tip

Sign up for DevGenius Updates

By Dev Genius

Get the latest news and update from DevGenius publication Take a look.

By signing up, you will create a Medium account if you don't already have one. Review our Privacy Policy for more information about our privacy practices.

Get this newsletter