

white duck

Containerized Build & Deployment Pipelines

Continuous Lifecycle / Container Conf, Mannheim 2019



Microsoft
Partner



Gold DevOps
Gold Cloud Platform

Nico Meisenzahl



- Senior Cloud & DevOps Consultant at white duck
- Docker Community Leader & GitLab Hero
- loves Kubernetes, DevOps and Cloud

Phone: +49 8031 230159 0
Email: nico.meisenzahl@whiteduck.de
Twitter: [@nmeisenzahl](https://twitter.com/nmeisenzahl)
LinkedIn: <https://www.linkedin.com/in/nicomeisenzahl>
Blog: <https://meisenzahl.org>



Agenda

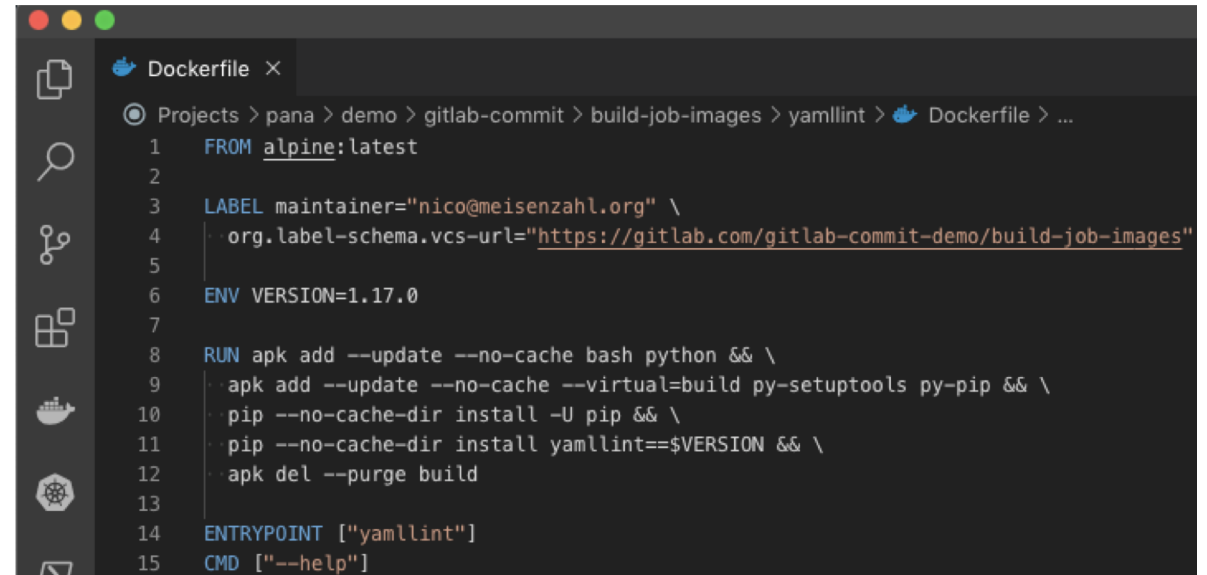
- why should you containerize your pipeline?
- pipeline workload on Kubernetes with GitLab CI/CD
- image builds on Kubernetes with Kaniko
- cloud-native pipelines with Tekton

Why should you containerize your pipeline?

- for the same reasons why you should use containers
 - isolation
 - dependencies
 - scalability
 - immutability
- example: your new project needs version X all others still require Y
 - you can insert any kind of build / deploy dependency
 - .NET Core, Go, Terraform, Ansible... you name it

Pipeline job image

- contains everything a single pipeline job needs
 - binaries, libraries, tools, ...
- use a pipeline to build/rebuild it periodically (security fixes!)
- you should define fix versions for your dependencies

A screenshot of a code editor showing a Dockerfile. The editor has a dark theme and a sidebar on the left with icons for file explorer, search, and other functions. The Dockerfile content is as follows:

```
1 FROM alpine:latest
2
3 LABEL maintainer="nico@meisenzahl.org" \
4     org.label-schema.vcs-url="https://gitlab.com/gitlab-commit-demo/build-job-images"
5
6 ENV VERSION=1.17.0
7
8 RUN apk add --update --no-cache bash python && \
9     apk add --update --no-cache --virtual=build py-setuptools py-pip && \
10    pip --no-cache-dir install -U pip && \
11    pip --no-cache-dir install yamllint==$VERSION && \
12    apk del --purge build
13
14 ENTRYPOINT ["yamllint"]
15 CMD ["--help"]
```

Kubernetes vs. *docker run*

- every pipeline job runs in a container
 - based on an image with all requirements for this single job
- Build host with Docker daemon
 - or any other container solution
- GitLab Runner Kubernetes executor
 - integrates your CI/CD with Kubernetes
 - runs a pod per job
 - containing a container with the defined image along with some service containers
 - allows you to share your resources and scale your pipelines

GitLab Runner Kubernetes executor

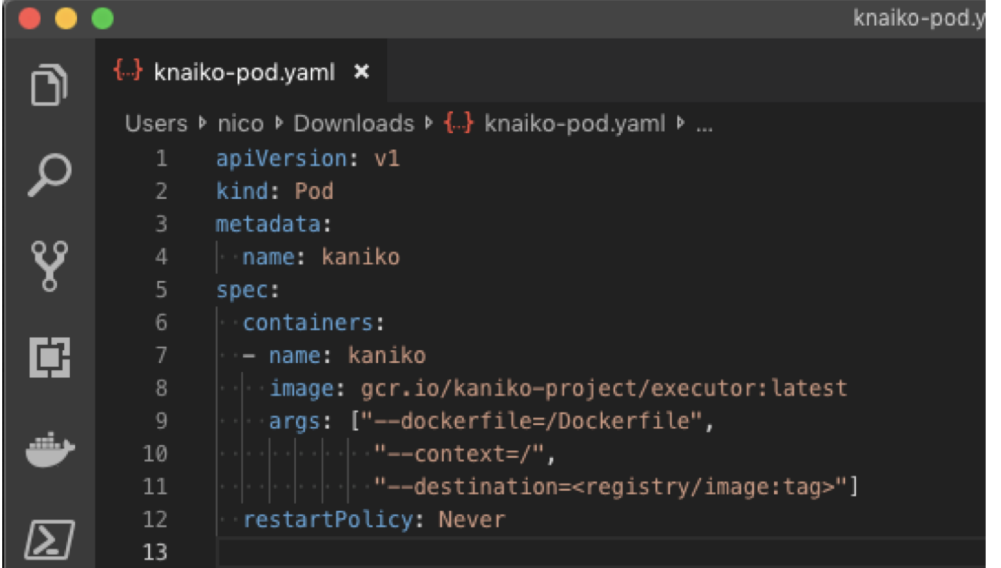
- runs itself in a pod
- needs to be installed in your Kubernetes Cluster
 - automatable Helm deployment
- schedules job pods
- build steps of a pipeline job
 - prepare → creates pod with build and service containers
 - pre-build → clones repo, restore cache, download artifacts
 - build → user build steps
 - post-build → creates caches and upload artifacts

Demo

- containerized pipelines on Kubernetes with GitLab Runner

Image builds on Kubernetes with Kaniko

- any Docker-in-Docker solution has issues
 - exposing Docker socket
 - mounting /var/lib/docker
 - privileged mode
- image builds without the need of any privileges or dependencies
- runs in a container
 - <http://gcr.io/kaniko-project/executor>
- use build caching to speed up your pipeline

A screenshot of a code editor window titled 'knaiko-pod.y' showing a Kubernetes pod definition for Kaniko. The code is in YAML format and defines a pod named 'kaniko' with a single container named 'kaniko'. The container uses the image 'gcr.io/kaniko-project/executor:latest' and has arguments for building a Docker image. The pod has a restart policy of 'Never'.

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: kaniko
5  spec:
6    containers:
7      - name: kaniko
8        image: gcr.io/kaniko-project/executor:latest
9        args: ["--dockerfile=/Dockerfile",
10              "--context=/",
11              "--destination=<registry/image:tag>"]
12        restartPolicy: Never
13
```

Demo

- containerized image builds on Kubernetes with Kaniko

Cloud-native pipelines with Tekton

- Tekton Pipelines emerged out of the Knative build project
- Jenkins X pipelines are based on Tekton
- moves your whole CI/CD into Kubernetes
- based on
 - CRDs (Custom Resource Definitions)
 - Controllers
- Tekton Triggers can be used to call a pipeline
- project also provides a CLI and Dashboard

```
clustertasks.tekton.dev  
conditions.tekton.dev  
eventlisteners.tekton.dev  
images.caching.internal.knative.dev  
pipelineresources.tekton.dev  
pipelineruns.tekton.dev  
pipelines.tekton.dev  
taskruns.tekton.dev  
tasks.tekton.dev  
triggerbindings.tekton.dev  
triggertemplates.tekton.dev
```

Demo

- cloud-native pipelines with Tekton Pipelines

Questions?



Slides: <https://www.slideshare.net/nmeisenzahl>

Demo: <https://gitlab.com/nmeisenzahl/conconf-conli>

Nico Meisenzahl (Senior Cloud & DevOps Consultant)

Phone: +49 8031 230159 0

Email: nico.meisenzahl@whiteduck.de

Twitter: [@nmeisenzahl](https://twitter.com/nmeisenzahl)

LinkedIn: <https://www.linkedin.com/in/nicomeisenzahl>

Blog: <https://meisenzahl.org>

