agenda » Intro K8s-native CI/CD & Windows Images Pipelines deep-dive Limitations, Summary & Further Info

LEVERAGING WINDOWS CONTAINERS IN YOUR KUBERNETES-NATIVE CI/CD PIPELINES

Markus Lippert, FOSDEM 2022

Markus Lippert

DevOps Engineer
COSMO CONSULT

DevOps, Containers, Orchestration,
Cloud-native, IaC and Azure

lippert_markus

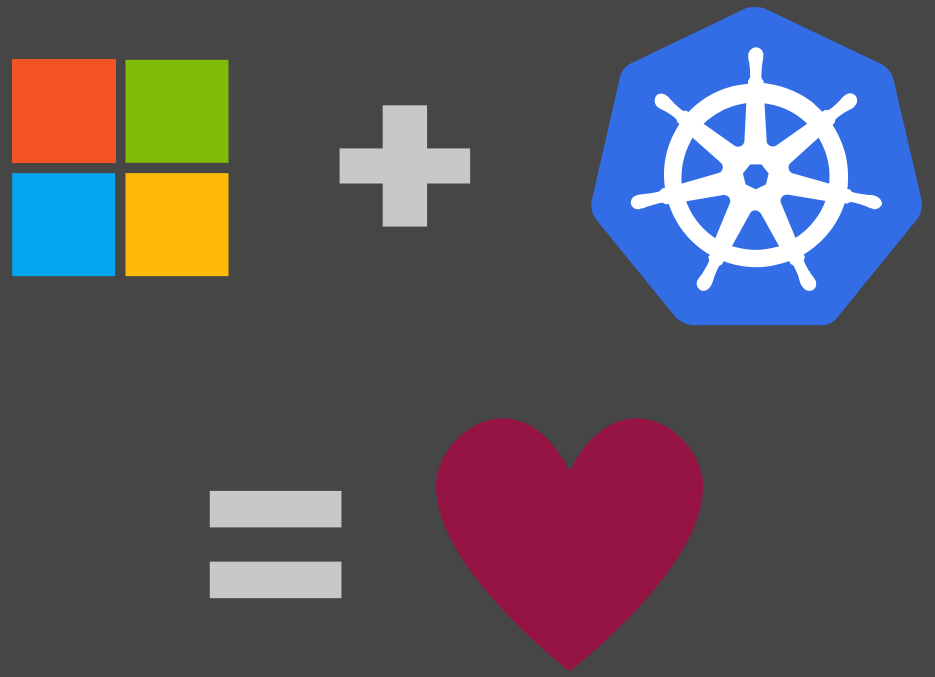
lippertmarkus

lippertmarkus

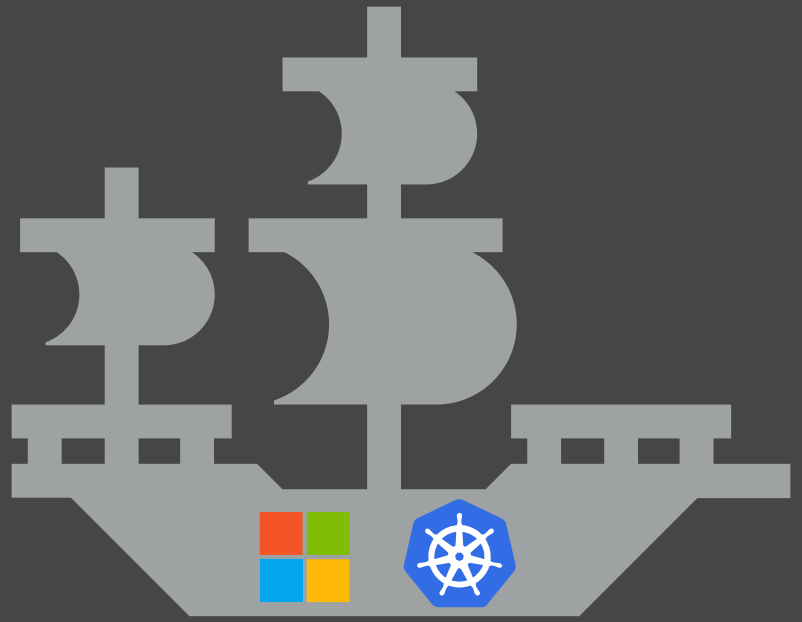
lippertmarkus.com

Quick Introduction

- Large portion of Windows applications running in many enterprises
- Windows Containers work similar to Linux containers
- Kubernetes has support for heterogenous clusters with both Windows and Linux nodes
- Goal: Leveraging Kubernetes-native CI/CD solutions for your Windows applications like you do for your Linux applications



Kubernetes-native CI/CD & Windows Containers



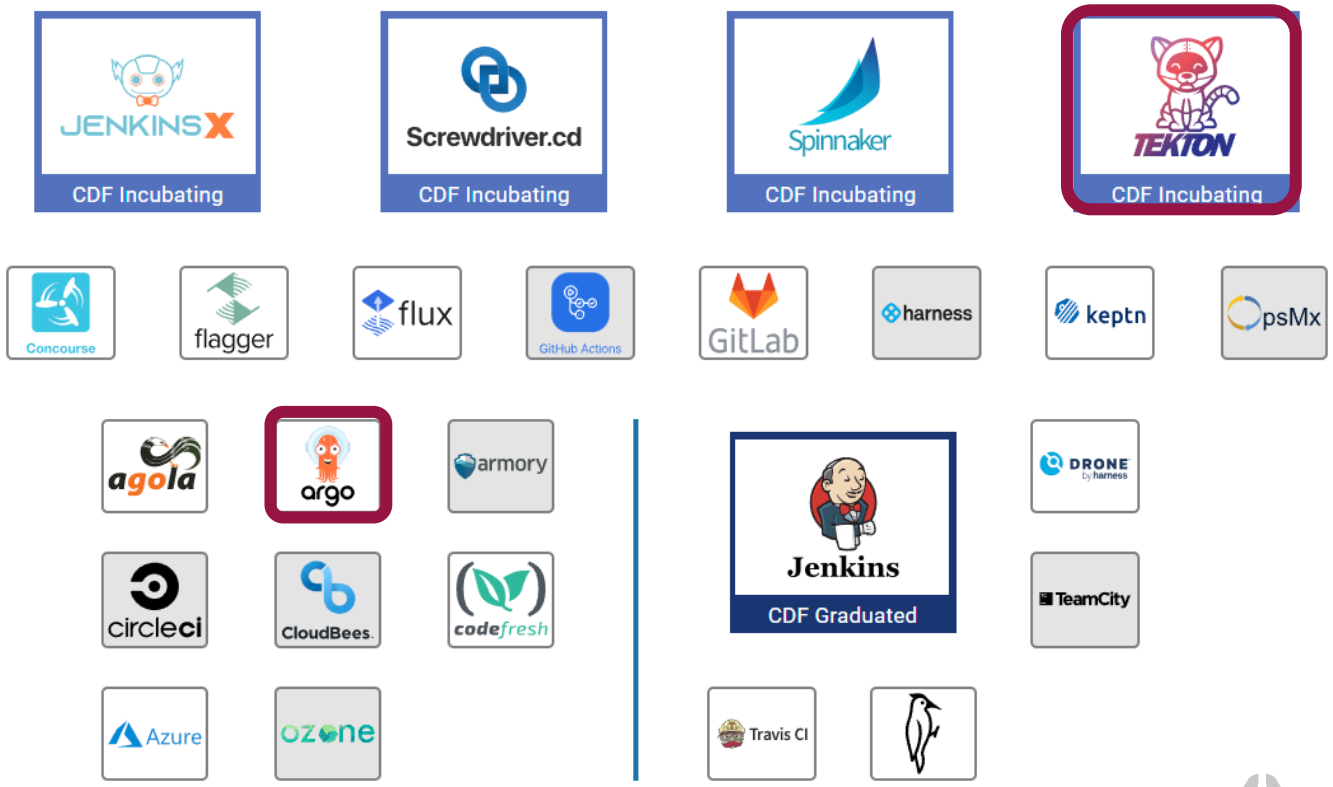
- What are “Kubernetes-native” CI/CD solutions?
  - Solutions which build on top of Kubernetes and containers
  - Usage of Kubernetes resources for running steps of the pipelines
- Such solutions enable reproducibility, performance, autoscaling & highly parallel jobs for your pipelines

- Only a few projects found in the CNCF or CDF landscape belong to this generation
- When looking for Windows Container support the selection is further limited

Kubernetes-native CI/CD solutions with Windows Container support

CLOUD NATIVE
COMPUTING FOUNDATION

CD.FOUNDATION



```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  name: hello-world
spec:
  entrypoint: myentry
  templates:
    - name: myentry
      steps:
        - - name: step1
            template: hello-windows
        - - name: step2
            template: hello-linux

    - name: hello-windows
      nodeSelector:
        kubernetes.io/os: windows  # step runs on Windows
      container:
        image: mcr.microsoft.com/windows/nanoserver:1809
        command: ["cmd", "/c"]
        args: ["echo", "Hello from Windows Container!"]

    - name: hello-linux
      nodeSelector:
        kubernetes.io/os: linux  # step runs on Linux
      container:
        image: alpine
        command: ["echo"]
        args: ["Hello from Linux Container!"]
```

Argo Workflows

Argo Workflows is an open source container-native workflow engine for orchestrating parallel jobs on Kubernetes.

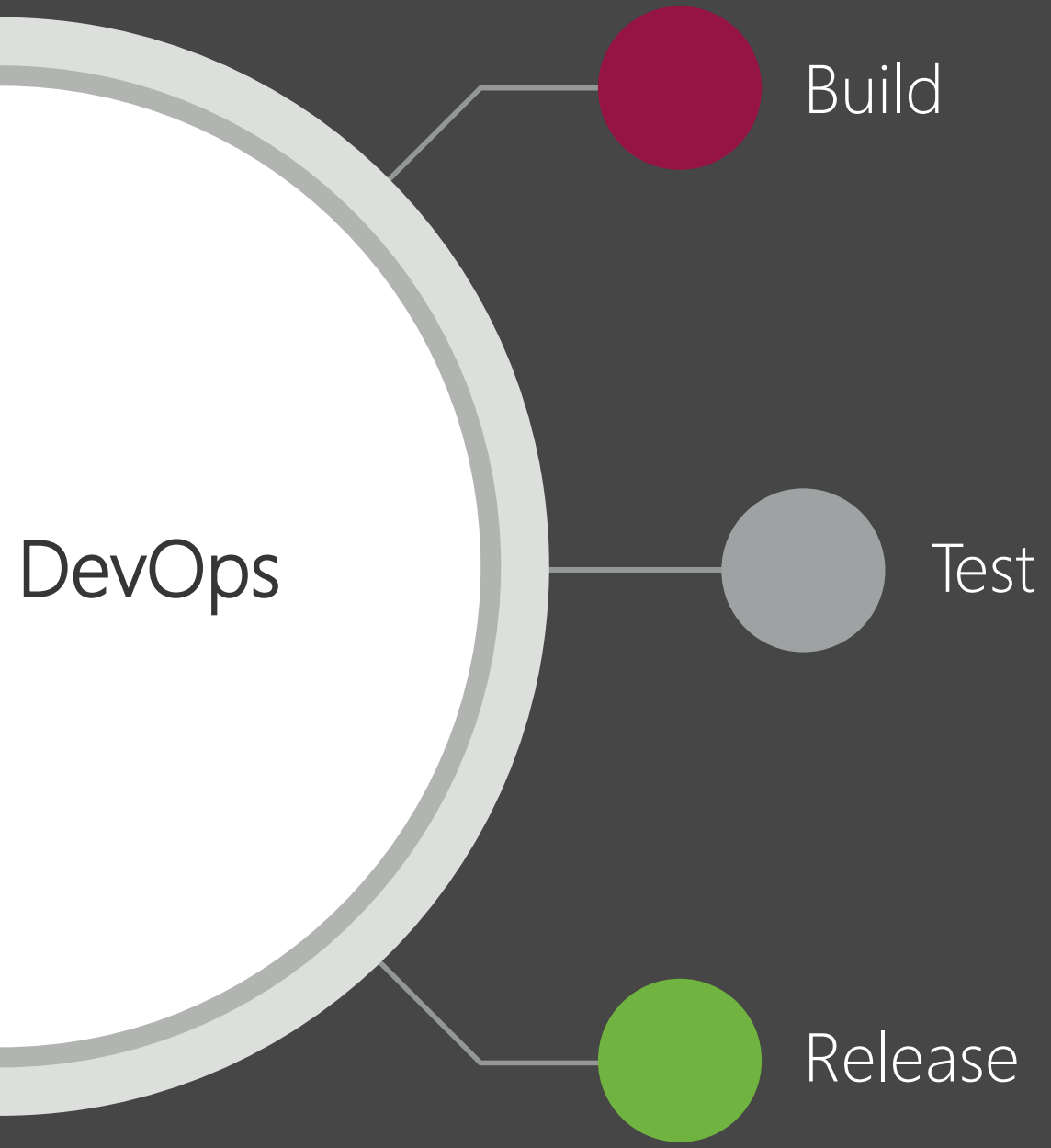
- Pipelines, machine learning & data processing as main use cases
- Windows Container support since 05/2020

Tekton Pipelines

Tekton is a cloud-native solution for building CI/CD pipelines.

- Focus on flexibility, reusability, extensibility & scalability
- Windows Container support since 10/2021

```yaml
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: hello-world
spec:
  pipelineSpec:
    tasks:
      - name: task-win
        taskSpec:
          steps:
            - name: hello-windows
              image: mcr.microsoft.com/windows/nanoserver:1809
              command: ["cmd", "/c"]
              args: ["echo", "Hello from Windows Container!"]
      - name: task-lin
        taskSpec:
          steps:
            - name: hello-linux
              image: alpine
              command: ["echo"]
              args: ["Hello from Linux Container!"]
  taskRunSpecs:
    - pipelineTaskName: task-win
      taskPodTemplate:
        nodeSelector:
          kubernetes.io/os: windows  # runs on Windows
        securityContext:
          windowsOptions:
            runAsUserName: "ContainerAdministrator"
    - pipelineTaskName: task-lin
      taskPodTemplate:
        nodeSelector:
          kubernetes.io/os: linux  # runs on Linux
```



Example scenario

- Build, test & create container image for Windows .NET application
- Focus on CI only (as CD for Windows apps is not different from CD for Linux apps, e.g. use Argo CD)

Building Windows Container images

within Kubernetes

- Building & testing Windows apps is trivial with available Windows nodes, creating images *within* Kubernetes is not
- Windows container images have a special filesystem layout, image builders must explicitly support it

Image Builder	Can create Windows images	Runs on	Notes
Buildah	✗	Linux only	
Kaniko	✗	Linux only	
img	✗	Linux only	not maintained, mainly a more simple CLI for BuildKit
~~Docker-in-Docker~~	~~✓~~	-	~~dockershim deprecated, unsecure~~
BuildKit	✓	Linux only	supports rootless
crane	✓	Linux and Windows	less known, can only append files to scratch/a base image and mutate images, operates on remote images

Building Windows Container images

Determining approaches with BuildKit & crane

- BuildKit only runs on Linux and therefore can only execute **RUN** instructions within Linux build stages
- crane can't run build-time commands for target images

COMPILE OUT OF IMAGE BUILD (BUILDKIT OR CRANE)

- Windows binary and other files are created/prepared in Windows or Linux container in a previous step (still perfectly reproducible)
- Prepared files are copied into the Windows container image during image creation

CROSS-COMPILE IN LINUX BUILD STAGE (BUILDKIT ONLY)

- Add build stage to Dockerfile running on Linux before final stage
- Build stage cross-compiles Windows binary and prepares/downloads prerequisites
- Prepared files are copied into the Windows container image in the final build stage

Building Windows Container images

Why not stick to local Windows container image builds on Windows VMs or hosted CI services?

- Need for multi-arch images, as the container host Windows version must match the image version
- Hosted CI services often only support 1-2 different Windows versions and have no Hyper-V isolation
- Building the whole container image for each Windows version is much slower than building the binary once and copy it to different base image versions
- Cross-building can be faster as Linux images are usually smaller, and containers are more performant
- You might want to harmonize how you build Linux vs. Windows container images

Comparing our pipelines

Argo Workflows vs. Tekton Pipelines

Area	Argo Workflows	Tekton Pipelines
Data sharing between steps	Using volumes directly	Using volumes abstracted as workspaces
Task organization	Each step is a pod (sidecars would be possible)	Each task is a pod & each step in the task is a container in this pod
Task dependencies	Steps can be run in sequence, more complex workflows with DAGs	Steps run in sequence; tasks run in parallel unless `runAfter` is specified
Artifact support	Some default options for input/output artifacts built-in	`PipelineResources` are deprecated in favor of catalog tasks
Windows Container support	Present for a longer time	Relatively new

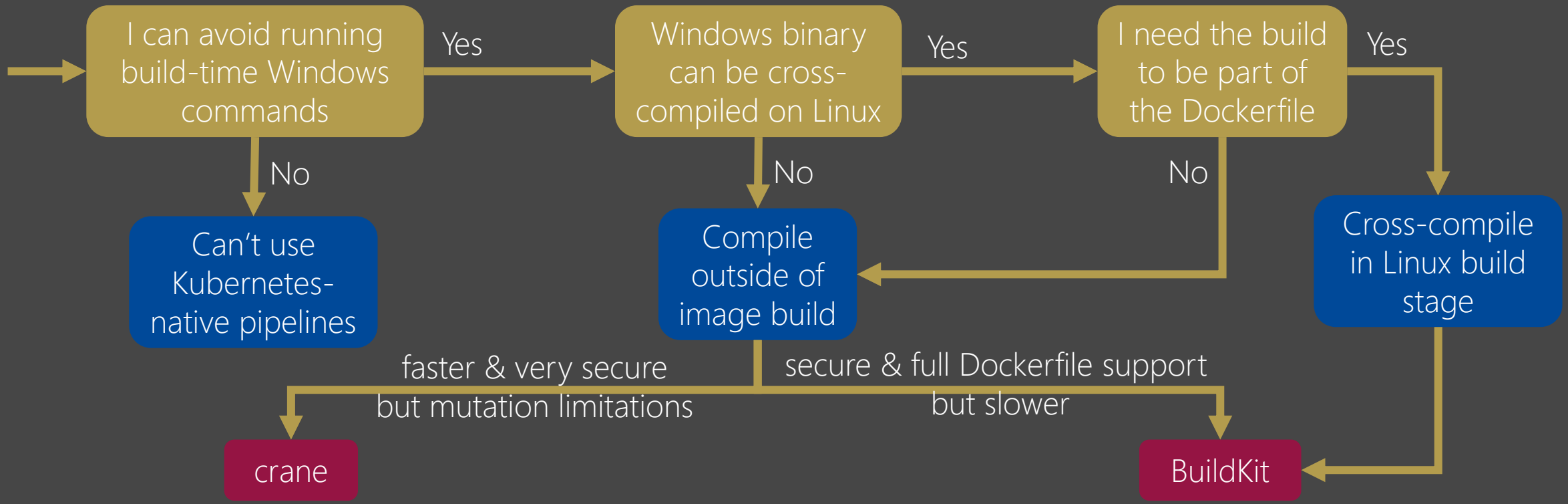
Limitations

- Running Windows commands during build is not supported, but you can often work around that:
  - Cross-compile and prepare prerequisites in a Linux build stage if your stack supports it
  - Compiling/testing outside of image build like shown before is mostly a valid option
  - For rarely changing dependencies build a dependency image natively on Windows once (where you can run MSIs or similar) and use that image to copy files from

- crane has no mutation equivalents for `USER`, `WORKDIR`, `SHELL`, `EXPOSE`, `VOLUME`, `HEALTHCHECK`
  - Last three are not used in K8s, `SHELL` is rarely needed, `USER` & `WORKDIR` could be set at deployment time
  - Use a base image with defaults that work for you
- Windows Containers in Argo Workflows and Tekton Pipelines are mostly community supported
- Windows Container support in Tekton is still in very early days, in Argo Workflows it's used by a few more but still a small group of users

Summary

Need for multi-arch Windows images, CI service limitations, speed and harmonization are possible reasons for using Kubernetes-native CI/CD solutions for Windows applications



Argo Workflows vs. Tekton Pipelines depends on personal & project preferences

Further reading

- All shown resources available in the session description!
- Go apps can use **ko** to create Windows container images (uses the "compile out of image build" approach as well)

```
# ** Using ko for Windows container images **
# Create a base image with all Linux/Windows
archs you want as target
> cat .\.ko.yaml
defaultBaseImage:
mcr.microsoft.com/windows/nanoserver:1809

> $env:KO_DOCKER_REPO="lippertmarkus/test"
> ko publish ./ --platform windows/amd64 --bare
```

- Build speed in BuildKit is improved with caching and storing the base images you build upon
- Building Windows multi-arch container images using the presented paths is greatly simplified compared to natively building on Windows, also see: https://lippertmarkus.com/2021/11/30/win-multiarch-img-lin/
- Deep dive into Windows container support of Argo Workflows: https://lippertmarkus.com/2020/10/15/cloud-native-ci-cd-windows-argo/
- Docs around Windows Container support:
  - Tekton: https://tekton.dev/docs/pipelines/windows/
  - Argo Workflows: https://argoproj.github.io/argo-workflows/windows/

Thank you.