Containerized CI/CD agents for Azure DevOps and GitHub

Markus Lippert, Tobias Fenster
as an Areopa Webinar

Agenda

- Intro
- Scope: What are build agents / runners?
- Why self-hosted agents / runners? Why containerized?
- Implementation + Real life examples
  - Azure DevOps
  - GitHub
- Outro

Introduction

Tobias Fenster

Managing Partner
4PS Germany

Microsoft Regional Director and dual MVP
Docker Captain

tobiasfenster
tobiasfenster@hachyderm.io
tfenster
tobiasfenster.io

Introduction

Markus Lippert

DevOps Engineer
COSMO CONSULT

Technical Lead for COSMO Alpaca

lippert_markus
lippertmarkus
lippertmarkus
lippertmarkus.com

What are build agents / runners?

- CI/CD needs something to run code / pipelines
  - Pipelines typically download code, compile the app, run automated tests, maybe do test deployments and release artifacts to some kind of storage
- Terminology:
  - Azure DevOps build agent = GitHub runner
  - Azure Pipelines = GitHub Actions
- This is triggered by something in your repo and picked up by an agent with the right capabilities
- Agents run the pipeline including tools, frameworks, SDKs, …
- Azure DevOps and GitHub provide standard agents, but you can also run self-hosted
- We won’t discuss the pipelines / workflows itself today unless required

Why self-hosted agents / runners?

- Full control over installed tools, resources/performance (and therefore cost)
- Caching (work in progress for cloud-hosted)

- Pricing Azure DevOps
  - 1 Microsoft-hosted job with 1,800 minutes per month for CI/CD and 1 self-hosted job with unlimited minutes per month
  - €38.48 per extra Microsoft-hosted CI/CD parallel job and €14.43 per extra self-hosted CI/CD parallel job with unlimited minutes
  - (1 free self-hosted job per VS subscription user that is part of the organization)
- Pricing GitHub
  - GitHub Actions usage is free for standard GitHub-hosted runners in public repositories, and for self-hosted runners. For private repositories, each GitHub account receives a certain amount of free minutes and storage for use with GitHub-hosted runners, depending on the product used with the account. Any usage beyond the included amounts is controlled by spending limits.

Why containerized?

- Always clean environment
- No need to set up & maintain agents
- No configuration drift between agents
- Natural fit if you use containers or build container images anyway
- On-demand scaling / parallelization
- Stability: A broken agent can be a real problem if you only have a few and new ones can’t be created on the fly
- How: Dockerfile with “setup” and resulting image, like on your own VM or bare metal

Two variations: on-demand vs. pre-created

- On-demand:
  - New build agent / runner container is started on-demand for each pipeline run
  - True clean environment, always new, dynamically scalable; requires a backend service
- Pre-created:
  - Host with pre-created build agent / runner container created in advance
  - Manually clean environment, reuse instance, manually scalable

- Mix: On-demand start of a VM with pre-created environments
- Best approach is a “it depends” decision: Do you already have a possible backend in place? Can you create and maintain a backend service? How cost-sensitive are you? …

- No connection to Azure DevOps vs. GitHub, both approaches possible in both platforms
  - We show on-demand in Azure DevOps and pre-created in GitHub, but that is random

Implementation - Azure DevOps build agent container

Image build

1. Download Azure DevOps build agent
2. Add dependencies: BcContainerHelper, .NET SDK, 7-zip, …
3. Startup script that registers agent on container startup

→ Dockerfile BC, Dockerfile .NET

```
docker build –t myagent -f Dockerfile.bcagent --build-arg BASE=ltsc2022
--build-arg AZP_URL=https://dev.azure.com/YourOrg --build-arg AZP_TOKEN=YourPAT
```

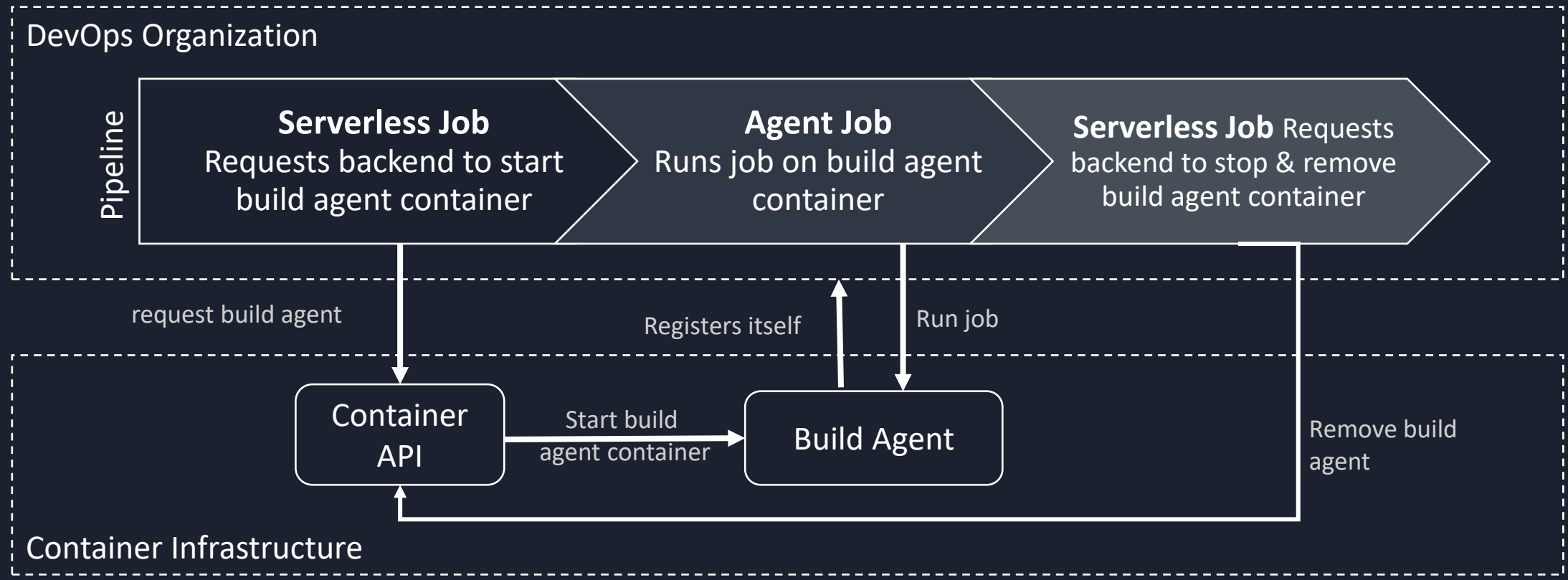
Container run

1. URL of Azure DevOps organization and PAT are passed as environment variables
2. Agent is configured and started
3. Agent registers at DevOps organization and runs jobs
4. Agent unregisters itself when container is stopped

➔ Startup script

Implementation - Azure DevOps on-demand agents

Idea: New & clean build agent container is started on-demand for each pipeline run



Real Life Example – COSMO Alpaca on-demand agents

- COSMO Alpaca already has container infrastructure for BC containers it’s APIs
- Running Azure DevOps agents as containers is a natural fit
- With ~600 Pipeline Runs per day you don’t want to manually install, manage or maintain agents or debug inconsistencies when agents are not always clean ➔ on-demand agents
- Only consuming agent resources when needed makes a big impact at scale
- Dynamically scaling the underlying cloud infrastructure can easily save you up to 80% when most of your pipelines run during working hours and only few outside those hours

DEMO

Implementation – GitHub runner container

Image build

1. Download and install dependencies (Docker CLI, git, jq) via choco
2. Download and expand runner
3. Add startup script

→ Dockerfile

```
docker build --isolation hyperv --build-arg BASE=ltsc2022 --build-arg VERSION=v2.9.6 -t myrunner .
```

Container run

1. GitHub organization or repo, runner name and personal access token passed in as parameters
2. Try to remove existing runner, configure new one and start it
3. Check for success

→ Script

Implementation – GitHub runner container

Azure infrastructure

- VM with Portainer (GUI container management, https://portainer.io) and Traefik (reverse proxy, https://traefik.io) preinstalled: https://learn.microsoft.com/en-us/samples/azure/azure-quickstart-templates/docker-portainer-traefik-windows-vm/ or search for “Portainer” in official Azure Quickstart Templates list

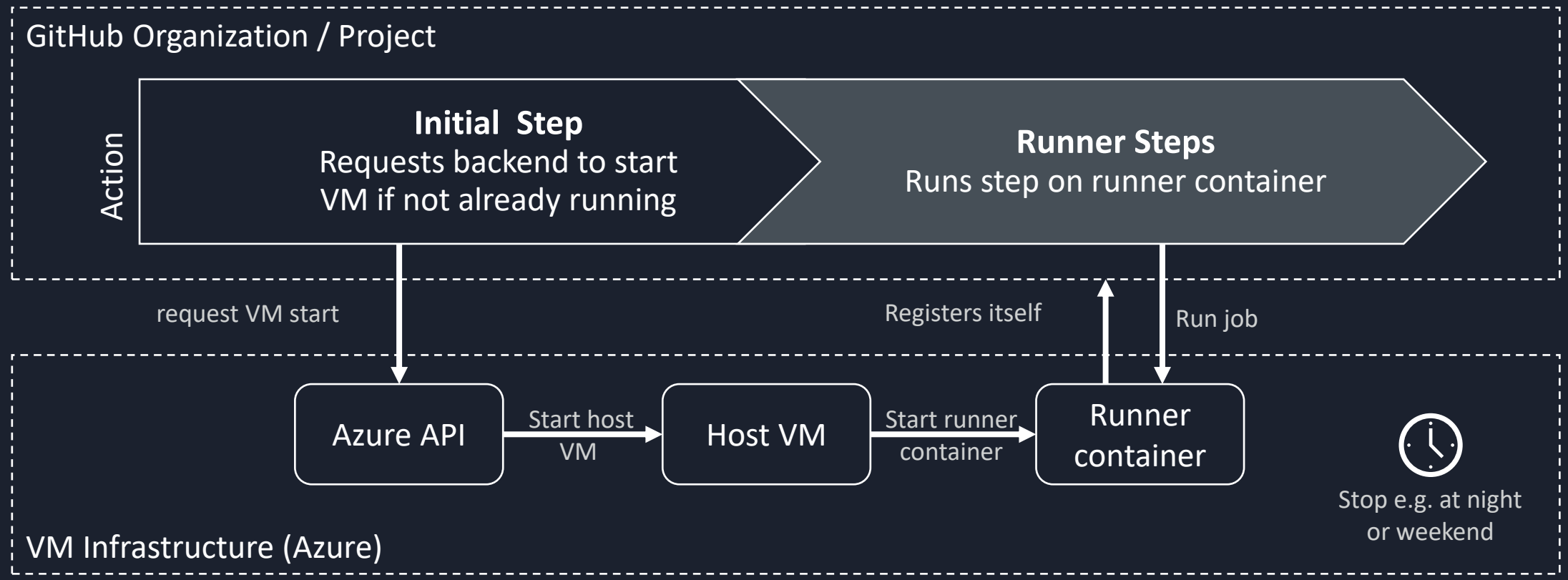
Container deployment

- Custom docker-compose stack
- Custom app template

→ We’ll see both

Implementation – GitHub pre-created runners

Idea: Host with pre-created runner container created in advance



Real Life Example – GitHub predefined runners for Traefik image

- Container images very much benefit from existing layers, so using a Cloud runner would have significantly worse performance → self-hosted makes sense
- Only occasional pipeline runs (once per upstream release) → pre-defined is ok
- Nightly shutdown is enough for cost saving

DEMO

Wrap-up

- CI/CD needs agents/runners to run pipelines
- Self-hosted agents/runners give you full control and allow for cost-savings & performance
- Why containerized agents/runners?
  - Set-up, maintenance and parallelization is simplified
  - Provide an always clean and more deterministic environment
- Characteristics of pre-created agents/runners
  - Created once and reused but can be easily replaced
  - Manually scalable
- Characteristics of on-demand agents/runners
  - Started for each pipeline run & deleted afterwards
  - Dynamically scalable & always new but requires a backend service
- Mix: On-demand start of a VM with pre-created environments

Thank you!

Questions?

Links

https://github.com/cosmoconsult/azdevops-build-agent-image

https://github.com/cosmoconsult/github-runner-windows

https://learn.microsoft.com/en-us/samples/azure/azure-quickstart-templates/docker-portainer-traefik-windows-vm/

https://raw.githubusercontent.com/tfenster/templates/master/templates-2.0.json