CI/CD im Container für Azure DevOps und GitHub

Markus Lippert, COSMO CONSULT
Tobias Fenster, 4PS

Agenda

- Vorstellung
- Umfang: Was sind Build-Agenten / Runner?
- Warum selbst gehostete Agenten / Runner? Warum containerisiert?
- Umsetzung + Beispiele aus der Praxis
  - Azure DevOps
  - GitHub
- Zusammenfassung
- Bonus: Agent Autoscaling unter Kubernetes mit KEDA

Einführung

Tobias Fenster

Geschäftsführer 4PS Deutschland

Microsoft Regional Director und zweifacher MVP
Docker Captain

tobiasfenster
tobiasfenster@hachyderm.io
tfenster
tobiasfenster.io

Einführung

Markus Lippert

DevOps-Engineer
COSMO CONSULT

lippert_markus
lippertmarkus
lippertmarkus
lippertmarkus.com

Was sind Build-Agents / -Runner?

- CI/CD braucht etwas, um Code / Pipelines auszuführen
  - Pipelines laden in der Regel Code herunter, kompilieren die Anwendung, führen automatisierte Tests durch, führen eventuell Testdeployments durch und stellen Artefakte bereits
- Terminologie:
  - Azure DevOps Build Agent = GitHub Runner
  - Azure Pipelines = GitHub Actions
- Ein Run wird durch etwas im Repo ausgelöst und von einem Agenten mit den richtigen „Capabilities“ aufgegriffen
- Agenten führen die Pipeline aus, einschließlich Tools, Frameworks, SDKs, ...
- Azure DevOps und GitHub bieten Standard-Agents, aber auch selbst gehostet möglich
- Wir werden heute nicht über die Pipelines/Workflows selbst sprechen, falls nicht explizit notwendig

Warum selbst gehostete Agents / Runner?

- Volle Kontrolle über installierte Tools, Ressourcen/Leistung (und damit Kosten)
- Caching (für GitHub Runner bereits teilweise möglich)

- Preisgestaltung Azure DevOps
  - 1 bei Microsoft gehosteter Job mit 1.800 Minuten pro Monat für CI/CD und 1 selbst gehosteter Job mit unbegrenzten Minuten pro Monat
  - 38,48 € pro zusätzlichem, von Microsoft gehostetem CI/CD-Paralleljob und 14,43 € pro zusätzlichem, selbst gehostetem CI/CD-Paralleljob mit unbegrenzten Minuten
  - (1 kostenloser selbst gehosteter Auftrag pro VS-Abonnement-Nutzer, der Teil des Unternehmens ist)
- Preisgestaltung GitHub
  - Die Nutzung von GitHub Actions ist für standardmäßig auf GitHub gehostete Runner in öffentlichen Repositories und für selbst gehostete Runner kostenlos. Für private Repositories erhält jedes GitHub-Konto eine bestimmte Menge an Freiminuten und Speicherplatz für die Verwendung mit GitHub-gehosteten Runnern, je nach dem mit dem Konto verwendeten Produkt. Jede darüber hinausgehende Nutzung wird durch Ausgabenlimits kontrolliert.

Warum in Containern?

- Immer saubere Umgebung
- Keine Notwendigkeit, Agenten einzurichten und zu pflegen
- Keine Konfigurationsabweichung zwischen Agenten
- Passt perfekt, wenn ohnehin Container verwendet oder Container-Images erstellt werden
- Skalierung/Parallelisierung nach Bedarf
- Stabilität: Ein defekter Agent kann ein echtes Problem sein, wenn es nur einige wenige gibt und neue Agenten nicht spontan erstellt werden können.
- Wie: Dockerfile mit "setup" und resultierendem Image, z.B. auf eigener VM oder Bare Metal

Zwei Varianten: bei Bedarf vs. vorab erstellt

- Bei Bedarf („On demand“):
  - Ein neuer Build-Agent / Runner-Container wird bei Bedarf für jeden Pipeline-Lauf gestartet
  - Komplett saubere Umgebung, immer neu, dynamisch skalierbar; erfordert einen Backend-Dienst
- Vorab erstellt:
  - Host mit im Voraus erstelltem Build-Agent/Runner-Container
  - Manuelles Aufräumen der Umgebung, Wiederverwendung der Instanz, manuelle Skalierbarkeit

- Mischen: On-Demand-Start einer VM mit vorab erstellten Umgebungen
- Der beste Ansatz ist eine "Es kommt darauf an"-Entscheidung: Gibt es bereits ein mögliches Backend? Kann ein Backend erstellt und gepflegt werden? Wie sieht das Budget aus und worauf wird wert gelegt? ...

- Keine Verbindung zur Entscheidung Azure DevOps vs. GitHub, beide Ansätze in beiden Plattformen möglich
  - Wir zeigen on-demand in Azure DevOps und vorab erstellt in GitHub, aber das ist zufällig

Implementierung - Azure DevOps Build Agent Container

Aufbau des Container Images

1. Azure DevOps-Build-Agent herunterladen
2. Abhängigkeiten hinzufügen: .NET SDK, 7-zip, ...
3. Startskript, das den Agenten beim Start des Containers registriert

→ Dockerfile .NET

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

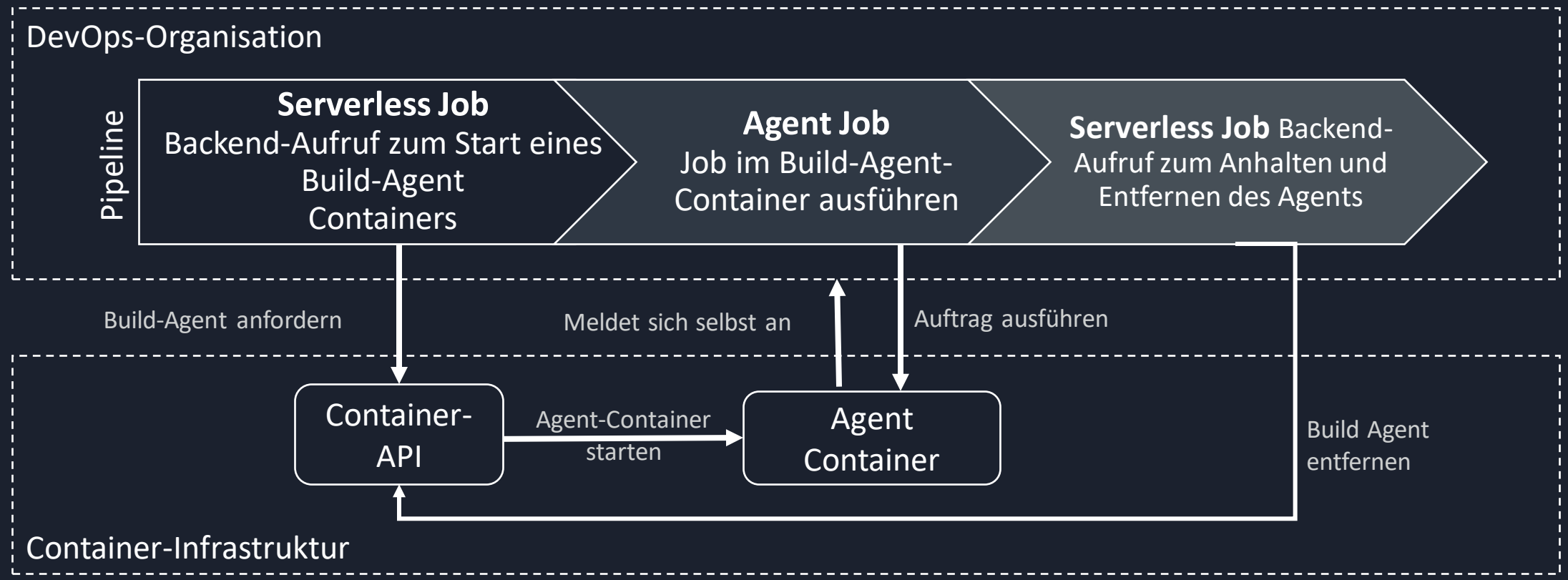
Containerausführung

1. URL der Azure DevOps-Organisation und PAT werden als Umgebungsvariablen übergeben
2. Agent wird konfiguriert und gestartet
3. Agent registriert sich bei DevOps-Organisation und führt Jobs aus
4. Agent meldet sich selbst ab, wenn Container gestoppt wird

➔ Startup-Skript

Implementierung – On-Demand Azure DevOps-Agent

Idee: Neuer und sauberer Agent-Container wird bei Bedarf für jeden Pipeline-Lauf gestartet



Praxisbeispiel - COSMO Alpaca On-Demand-Agents

- COSMO Alpaca hat bereits eine Container-Infrastruktur für Entwicklungscontainer & APIs
- Die Ausführung von Azure DevOps-Agents als Container liegt nahe
- Bei ~600 Pipelines pro Tag wäre manuelle Installation, Verwaltung, Wartung und das Debuggen von Inkonsistenzen für unsaubere Agent, nicht machbar
  ➔ on-demand agents
- Ressourcen für Build Agents nur bei Bedarf zu nutzen hilft enorm bei Skalierung
- Durch dynamische Skalierung der unterliegenden Cloud-Infrastruktur können leicht bis zu 80 % eingespart werden, wenn die meisten Pipelines während der Arbeitszeiten und nur wenige außerhalb dieser Zeiten laufen.

DEMO

Implementierung – GitHub Runner Container

Aufbau des Container Images

1. Herunterladen und Installieren von Abhängigkeiten (Docker CLI, git, jq) über choco
2. Runner herunterladen und entpacken
3. Startscript hinzufügen

→ Dockerfile

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

Containerausführung

1. GitHub-Organisation oder Repo, Name des Runners und PAT (persönliches Zugriffstoken) als Parameter übergeben
2. Versuchen, den vorhandenen Runner zu entfernen, einen neuen zu konfigurieren und zu starten
3. Erfolgskontrolle

→ Skript

Implementierung – GitHub Runner Container

Azure-Infrastruktur

- VM mit Portainer (GUI Container Management, https://portainer.io) und Traefik (Reverse Proxy, https://traefik.io) vorinstalliert: https://learn.microsoft.com/en-us/samples/azure/azure-quickstart-templates/docker-portainer-traefik-windows-vm/ oder Suche nach "Portainer" in der offiziellen Azure Quickstart Templates Liste

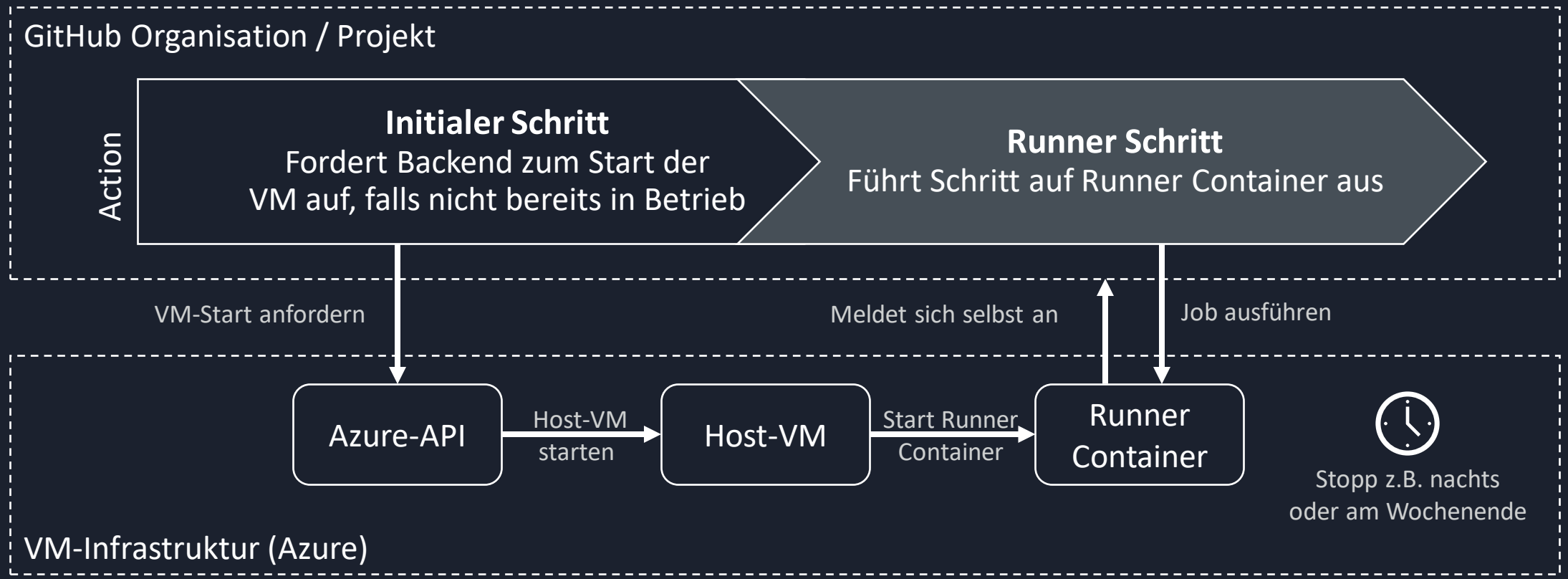
Container Deployment

- Benutzerdefinierter Docker-Compose-Stack
- Benutzerdefinierte App-Vorlage

→ Wir schauen uns beides an

Implementierung – Vorab erstellter GitHub Runner

Idee: Host mit vorab erstelltem Runner-Container



Praxisbeispiel – Vorab erstellter GitHub Runner für Traefik-Image

- Container-Images profitieren in hohem Maße von vorhandenen Schichten (Layers), so dass die Verwendung eines Cloud-Runners eine deutlich schlechtere Leistung zur Folge hätte. → self-hosted macht Sinn
- Nur gelegentliche Pipeline-Läufe (einmal pro Upstream-Release) → vorab erstellt ist ok
- Nächtliche Abschaltung reicht für Kosteneinsparungen aus

DEMO

Zusammenfassung

- CI/CD benötigt Agents/Runner zur Ausführung von Pipelines
- Selbst gehostete Agents/Runner geben volle Kontrolle und ermöglichen Kosteneinsparungen und mehr Leistung
- Warum Agents/Runner containerisieren?
  - Einrichtung, Wartung und Parallelisierung werden vereinfacht
  - Bereitstellung einer stets sauberen und stärker deterministischen Umgebung
- Merkmale von vorab erstellt Agents/Runners
  - Einmal erstellt und wiederverwendet, können aber leicht ersetzt werden
  - Manuell skalierbar
- Merkmale von On-Demand Agents/Runnern
  - Werden für jeden Pipelinelauf gestartet und danach gelöscht
  - Dynamisch skalierbar und immer wieder neu, erfordern ggf. einen Backend-Dienst
- Mix möglich: On-Demand-Start einer VM mit vorab erstellten Agents/Runnern

Links

- https://github.com/cosmoconsult/azdevops-build-agent-image/blob/master/Dockerfile.coreagent
- 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
- https://lippertmarkus.com/2023/02/26/keda-azure-pipelines-agent/

Vielen Dank!
Noch Fragen?

Markus Lippert, COSMO CONSULT
Tobias Fenster, 4PS

Bonus – Autoscaling unter Kubernetes mit KEDA

- KEDA = Kubernetes Event Driven Autoscaler
- Skalierung von Kubernetes Objekten (Deployments, Jobs) auf Basis von Metriken
- Wartende Jobs in Azure DevOps als mögliche Metrik
- Warteschlange wird von KEDA überwacht und Kubernetes Jobs automatisch erstellt
- KEDA benötigt Zugangsdaten (über Kubernetes Secrets) und weitere Infos (in ScaledJob bzw. ScaledObject Ressourcen)

Bonus – On-Demand Azure DevOps-Agents mit KEDA

Idee: Neuer und sauberer Agent-Container wird bei Bedarf für jeden Pipeline-Lauf gestartet