WebAssembly in Kubernetes

Ist Docker dem Tod geweiht?

Stand 24.01.2023

prodot®

prodot

Dominique Göttner

Managing Director
prodot - Digital Products & Customer Experience

Linkedin
@pseudorng
info@prnd.io
www.prnd.io

prodot

- WebAssembly – Eine Übersicht
- Wo kann ich WebAssembly betreiben
- Dafür gibt es doch JavaScript/Docker
- WASI/WAGI
- Wasm in Kubernetes betreiben

WebAssembly – Eine Übersicht

WebAssembly (Wasm)

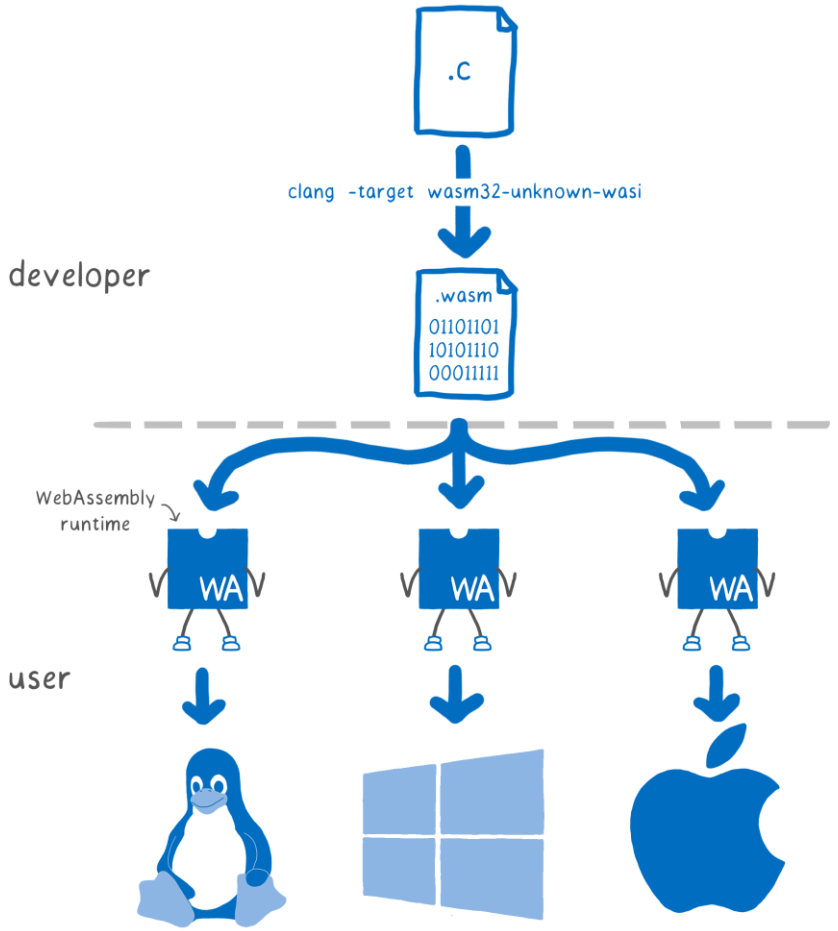
- **Low-Level** Dateiformat zur Ausführung auf einer **Runtime** (stackbasierte VM)
- Entweder als Text- oder **Binärformat** verfügbar
- Als zusätzliche Option zu JavaScript für den Betrieb im **Browser** gedacht
- Steht als **Compiler-Target** für verschiedenste Sprachen zur Verfügung

WebAssembly (Wasm) - Programmiersprachen

WebAssembly (Wasm) - Compiler / Frameworks

- **Rust:** cargo/rustc/wasm-bindgen
- **C/C++:** clang/emscripten
- **C#:** dotnet/blazor/dotnet-wasi-sdk
- **AssemblyScript:** asc

WebAssembly (Wasm) - Programmiersprachen



Referenz - [1]

Demo

Wo kann ich WebAssembly betreiben?

Wo kann ich WebAssembly betreiben?

- WebAssembly benötigt eine stackbasierte **VM** für den Betrieb (Runtime)
- Die gängigen Browser bringen inzwischen eine **Runtime** mit (JIT)

Wo kann ich WebAssembly betreiben?

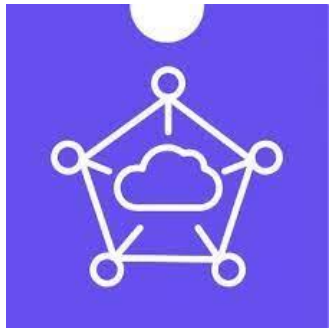
- WebAssembly benötigt eine stackbasierte **VM** für den Betrieb (Runtime)
- Die gängigen Browser bringen inzwischen eine **Runtime** mit (JIT)
- Aber warum im Browser aufhören?

Wo kann ich WebAssembly betreiben?

- WebAssembly benötigt eine stackbasierte **VM** für den Betrieb (Runtime)
- Die gängigen Browser bringen inzwischen eine **Runtime** mit (JIT)
- Aber warum im Browser aufhören?
  - Analog zu JavaScript gibt es Runtimes für das **Backend**

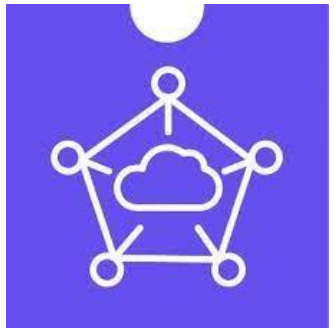
Wo kann ich WebAssembly betreiben?

- WebAssembly benötigt eine stackbasierte **VM** für den Betrieb (Runtime)
- Die gängigen Browser bringen inzwischen eine **Runtime** mit (JIT)
- Aber warum im Browser aufhören?
  - Analog zu JavaScript gibt es Runtimes für das **Backend**
  - **Wasmtime, Wasmer, WasmEdge**, u.a.



Wo kann ich WebAssembly betreiben?

- WebAssembly benötigt eine stackbasierte **VM** für den Betrieb (Runtime)
- Die gängigen Browser bringen inzwischen eine **Runtime** mit (JIT)
- Aber warum im Browser aufhören?
  - Analog zu JavaScript gibt es Runtimes für das **Backend**
  - **Wasmtime, Wasmer, WasmEdge**, u.a.
  - **Embedded** in andere Programme, z.B. als Grundlage für ein **Plugin-Framework**



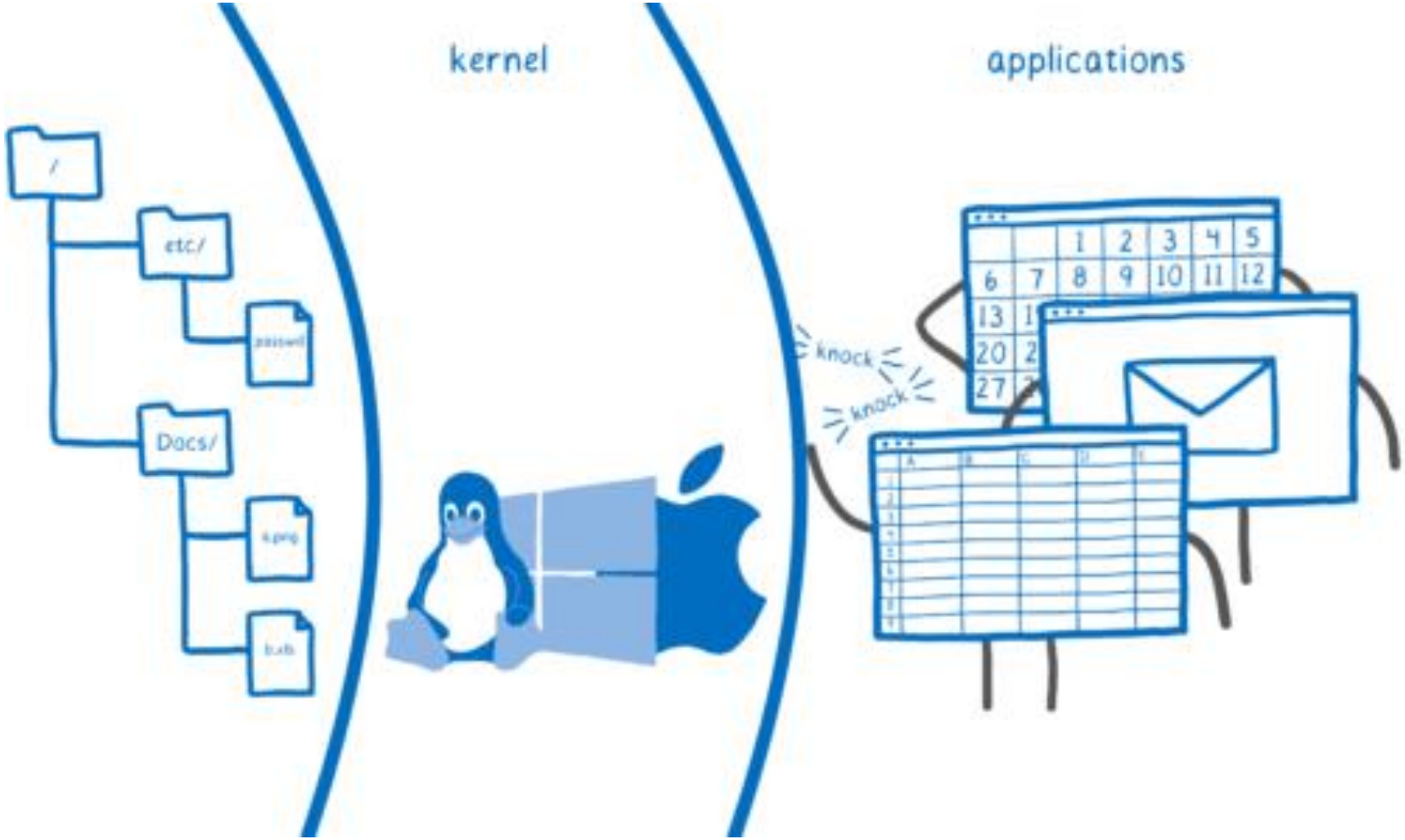
BYTECODE ALLIANCE

Aber dafür gibt es doch JavaScript/Docker?

Wasm – Pro/Contra

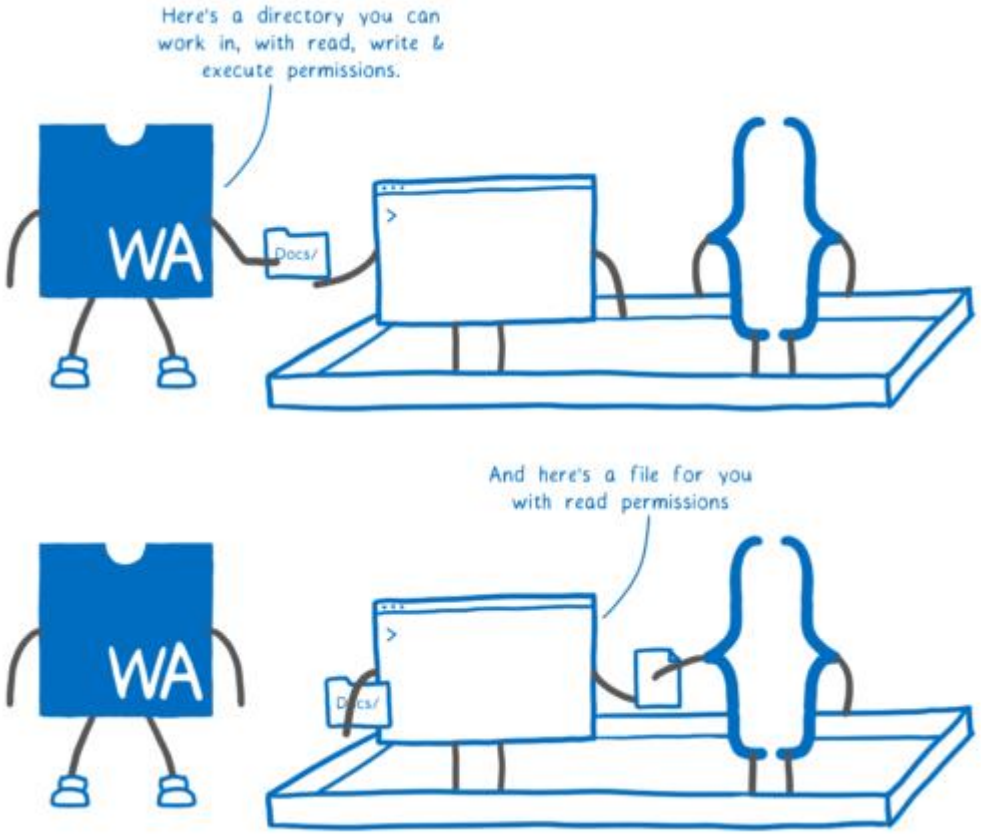
- **Fast**
  - Low-Level, wenig Abstraktionen, erprobte Compiler-Frameworks
  - Geringer Resourcenverbrauch
- **Safe**
  - Memory-Safe, Sandboxed
- **Secure**
  - Zugriff granular pro Modul steuerbar
- **Portable**
  - Browser, Windows, Linux, MacOS, u.a.

Wasm – Pro/Contra



Referenz - [1]

Wasm – Pro/Contra



Referenz - [1]

Ok, ist Docker(/NodeJs) also dem Tod geweiht?

- Wasm ist
  - **Schnell**
  - **Resourcenschonender**
  - Verbraucht **weniger** Speicher
  - Hat **kleinere** binäre Images
  - Bringt nur die Runtime als Overhead mit
  - Läuft im Browser & nativ
  - Ist **safe** & **secure** by default

Wasm – Pro/**Contra**

- Somtimes **not** as **fast**
  - JavaScript-Engines wurden seit Jahrzehnten optimiert
- Kein **Multithreading**
- **Debugging** ist aktuell noch schmerzhaft
- **Frameworks**/Libraries
  - Eingeschränkter Zugriff auf native Ressourcen

WASI/WAGI

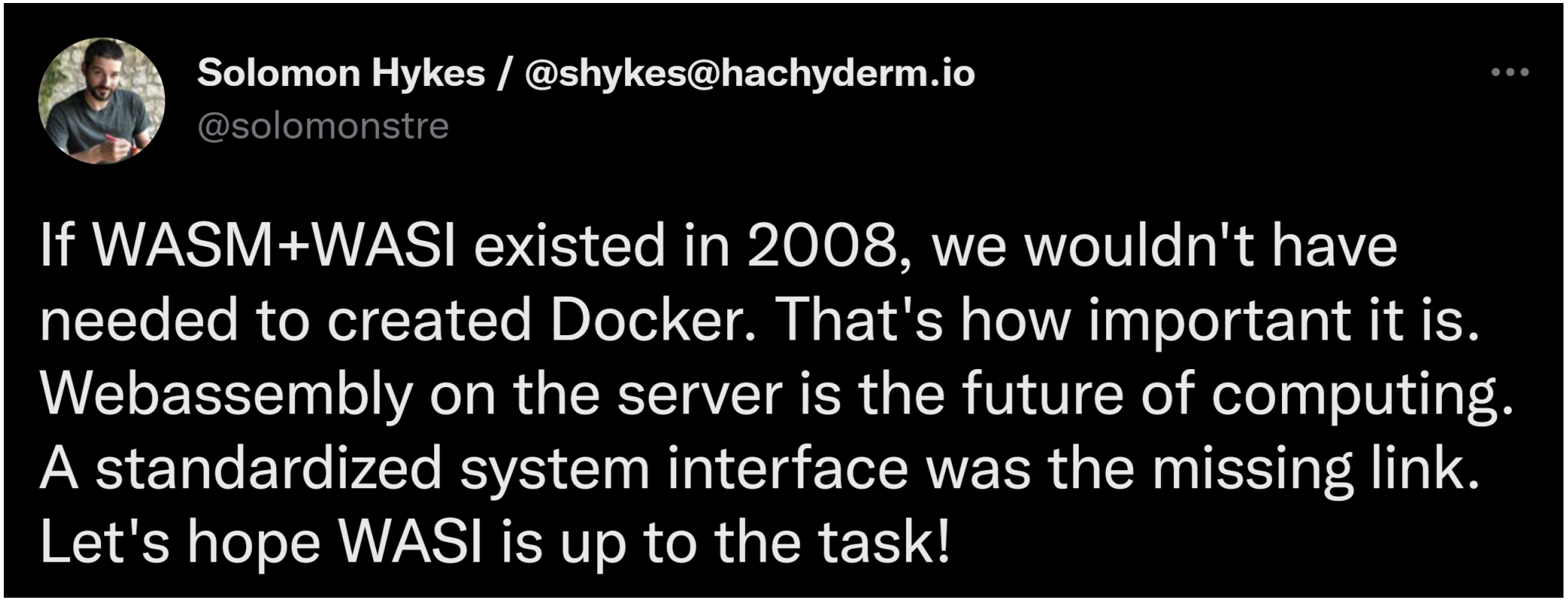
WebAssembly System Interface (WASI)

- **Modulare** Schnittstelle
- Stellt **native Ressourcen** zur Verfügung (Disk, NetworkIO)
- Ermöglicht einen **sandboxed** Betrieb
- Aktuell und absehbar **nicht** im Browser verfügbar
- Bereitgestellt durch die **Runtime** für den Betrieb im Backend

WebAssembly Gateway Interface (WAGI)

- Willkommen in den **90ern**
- Analog zum altbekannten **CGI**
- **Mapping** zwischen Route und ausführbaren **Prozess**
- Zugriff auf URL-Parameter und **Umgebungsvariablen**
- **Standard-IO** wird als Response weitergeleitet

Gründer von Docker

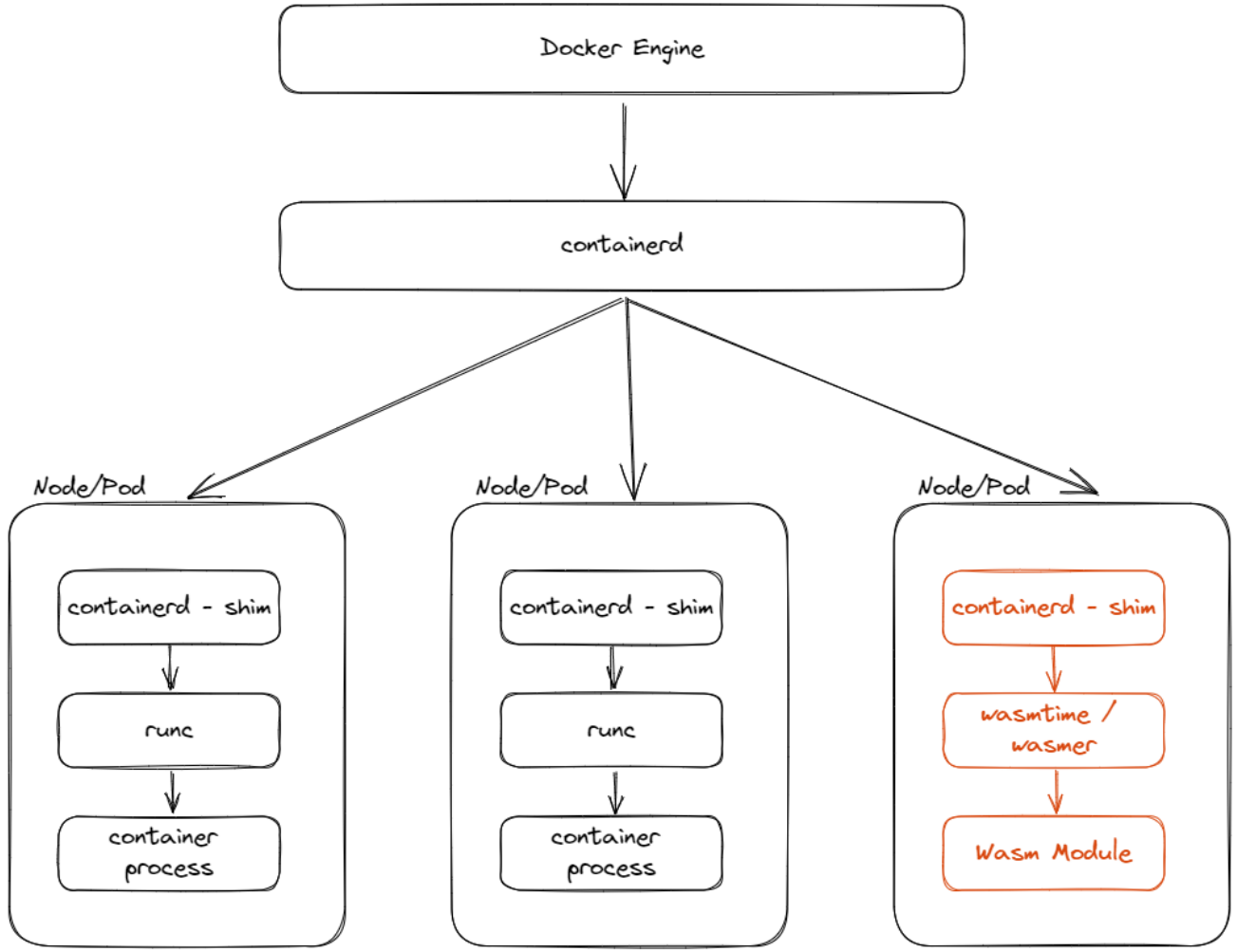


Referenz - [2]

Wasm in Kubernetes betreiben

WebAssembly in Kubernetes (AKS)

- Kubernetes ist modular aufgebaut
- Für die Verwerndung von Wasm wird
  - Ein contaienerd shim transparent eingebunden
  - Die Container Runtime (runc) wird ersetzt durch die eine Wasm-Runtime
  - Wasm Binärdateien werden analog zu Container-Images aus einem Repository geladen
- AKS unterstützt aktuell spin und slight als containerd shims
- Alternativ kann auch z.B. Krustlet verwendet werden



Demo

Docker is not dead… yet

Vielen Dank!

prodot®

Referenzen

- [1] - https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface/
- [2] - https://twitter.com/solomonstre/status/1111004913222324225?s=20