sidion

SIDION

Zuhören. Analysieren. Beraten.

MIT WEB-ASSEMBLY IN DIE CLOUD

Nicolai Mainiero – sidion GmbH

Photo by Taylor Van Riper on Unsplash

sidion

HINTERGRUND

Was ist Webassembly überhaupt?

Photo by Natalia Yakovleva on Unsplash

sidion

ENTWICKLUNG

2013 2015 2017 2018

2013: asm.js

```c
int f(int i) {
  return i + 1;
}
```

```javascript
function f(i) {
  i = i|0;
  return (i + 1)|0;
}
```

2015: WebAssembly angekündigt

2017: WebAssembly MVP abgeschlossen

2018:
Core Specification
JavaScript Interface
Web API

sidion

ENTWICKLUNG

2019

WASI

2019

Bytecode Alliance gegründet

2021

WasmEdge als Sandbox Projekt bei der CNCF aufegnommen

WasmEdgeRuntime

2021

Programming Languages Software Award gewonnen

sidion

ENTWICKLUNGSZIELE

WAS KANN WASM

- **Schnell**
  - Ausführung mit nahezu nativer Code-Performance, unter Ausnutzung von Funktionen, die alle moderne Hardware bietet.
- **Sicher**
  - Der Code wird validiert und in einer speichersicheren Umgebung ausgeführt, die Datenbeschädigung oder Sicherheitsverletzungen verhindert.
- **Gut definiert**
  - Vollständige und präzise Definition gültiger Programme und ihres Verhaltens auf eine Art und Weise, die informell und formal leicht zu erklären ist.
- **Hardware-unabhängig**
  - kann auf allen modernen Architekturen kompiliert werden, sowohl auf Desktop- und Mobilgeräten als auch auf eingebetteten Systemen.
- **Sprachunabhängig**
  - Bevorzugt keine bestimmte Sprache, kein bestimmtes Programmiermodell und kein bestimmtes Objektmodell.
- **Plattformunabhängig**
  - kann in Browsern eingebettet, als eigenständige VM ausgeführt oder in andere Umgebungen integriert werden.
- **Offen**
  - Programme können auf einfache und universelle Weise mit ihrer Umgebung interagieren

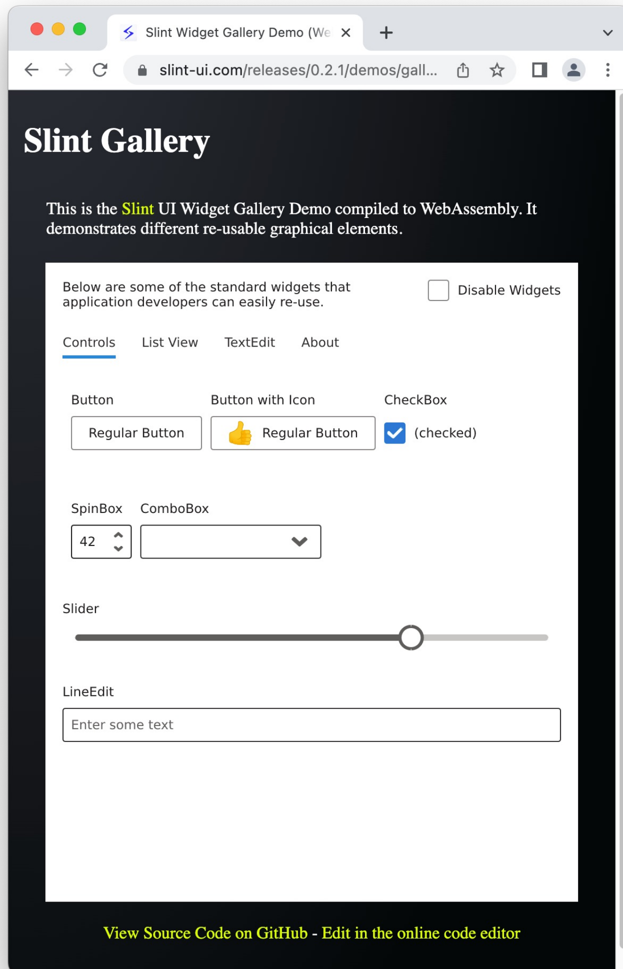
sidion

ENTWICKLUNGSZIELE

WAS KANN WASM

- **Kompakt**
  - hat ein Binärformat, das schnell zu übertragen ist, da es kleiner ist als typische Text- oder native Codeformate.
- **Modular**
  - Programme können in kleinere Teile aufgeteilt werden, die separat übertragen, zwischengespeichert und verbraucht werden können.
- **Effizient**
  - Dekodierung, Validierung und Kompilierung in einem einzigen schnellen Durchgang, entweder mit Just-in-Time- (JIT) oder AOT-Kompilierung (AOT = ahead of time).
- **Streamable**
  - Dekodierung, Validierung und Kompilierung können so schnell wie möglich beginnen, noch bevor alle Daten gesichtet wurden.
- **Parallelisierbar**
  - Dekodierung, Validierung und Kompilierung können in viele unabhängige parallele Aufgaben aufgeteilt werden.
- **Portabel**
  - macht keine architektonischen Annahmen, die von moderner Hardware nicht weitgehend unterstützt werden.

WEB BEISPIEL

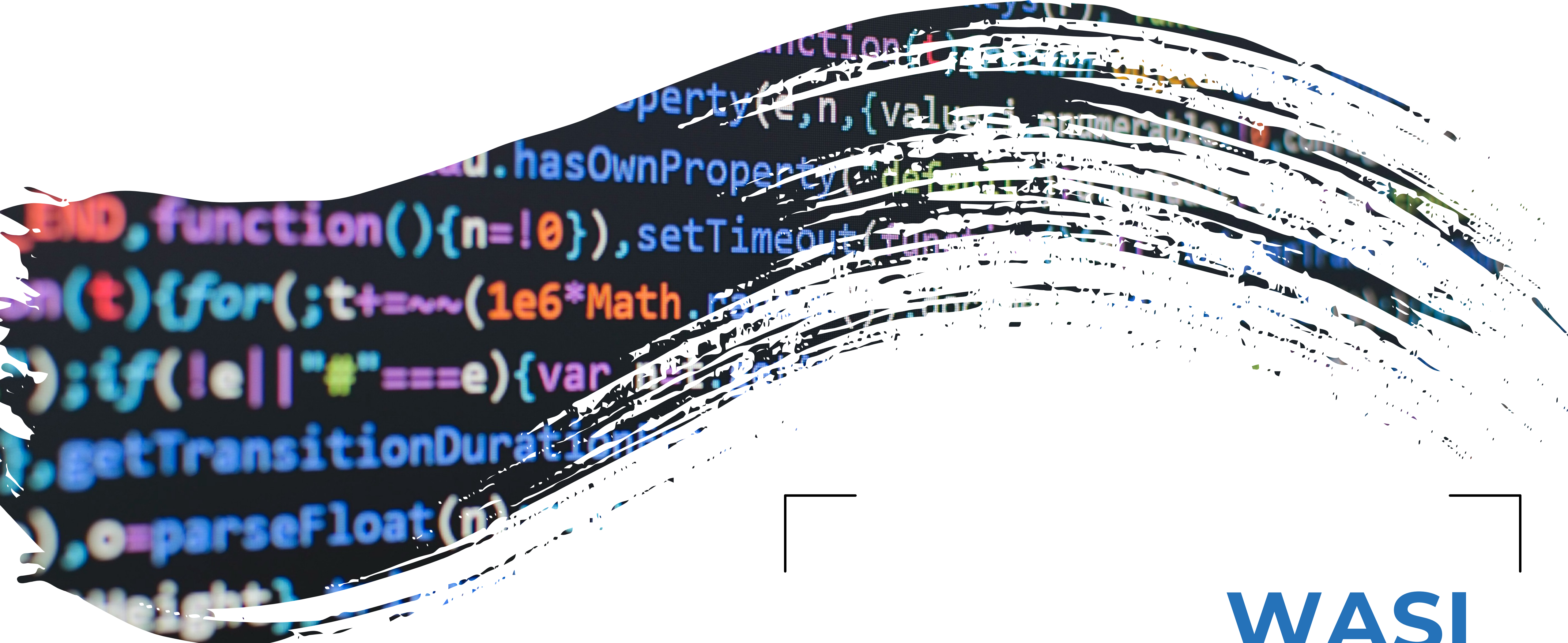



```html
<html><!--
    This is a static html file used to display the wasm build.
    In order to generate the build
     - uncomment the #wasm# lines in Cargo.toml
     - Run `wasm-pack build --release --target web` in this directory.
  --><head>
  <meta charset="UTF-8">
  <title>Slint Widget Gallery Demo (Web Assembly version)</title>
  <link rel="stylesheet" href="https://slint-ui.com/css/demos-v1.css">
</head>

<body>
  <h1>Slint Gallery</h1>
  <p>This is the <a href="https://slint-ui.com">Slint</a> UI Widget Gallery Demo compiled to WebAssembly. It
    demonstrates
    different re-usable graphical
    elements.</p>

  <canvas id="canvas" width="1000" height="1200" tabindex="0" data-raw-handle="1" alt="Slint Gallery"
style="width: 500px; height: 600px; cursor: auto;" cursor="auto"></canvas>
  <p class="links">
    <a href="https://github.com/slint-ui/slint/blob/master/examples/gallery/gallery.slint">
      View Source Code on GitHub</a> -
    <a href="https://slint-ui.com/editor?load_url=https://raw.githubusercontent.com/slint-ui/slint/master/examples
/gallery/gallery.slint">
      Edit in the online code editor
    </a>
  </p>
  <script type="module">
    import init from './pkg/gallery.js';
    init().finally(() => {
      document.getElementById("spinner").remove();
    });
  </script>
</body></html>
```


sidion



WASI

Modulares Interface für WebAssembly

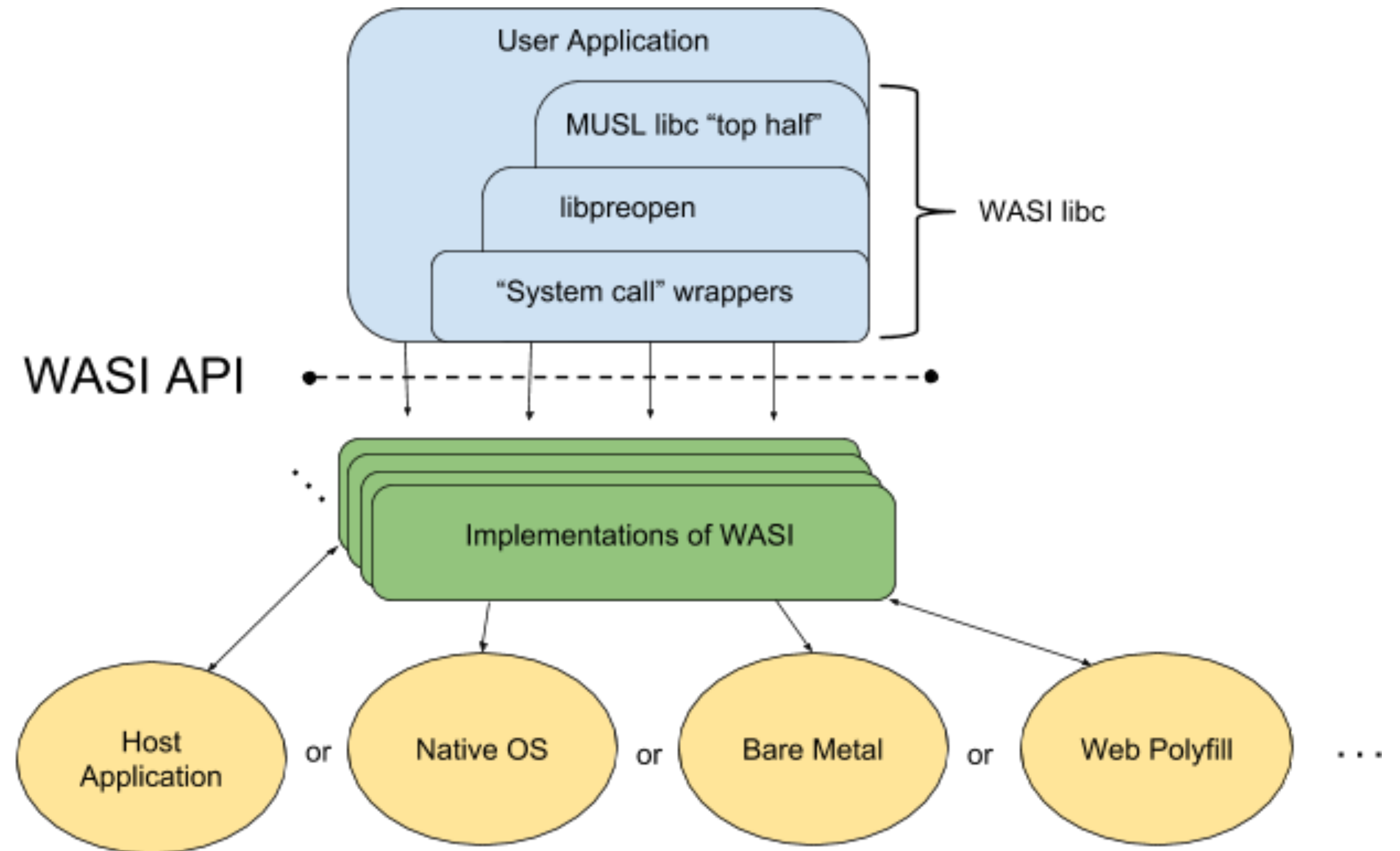
Photo by Fahrul Razi on Unsplash

sidion

STANDARD OS SCHNITTSTELLE

WEBASSEMBLY SYSTEM INTERFACE

- Zugriff auf Betriebssystem Ressourcen
  - Dateien und Dateisystem
  - Sockets
  - Uhr(en)
  - Zufallszahlen
- capability-based security
  - Dateizugriffe
  - Neztwerkzugriffe
  - etc.
- Portabel



https://github.com/bytecodealliance/wasmtime/blob/main/docs/WASI-overview.md

sidion

POLYGLOTT

UNTERSTÜTZTE SPRACHEN

Sprache	Browser	WASI	Bemerkung
Python	✓	✓	
Java (JVM)	✓	✗	
C/C++	✓	✓	
C# und .NET	✓	✓	Gilt für alle .NET Sprachen
TypeScript	✗	✗	AssemblyScript (Vertraute Syntax, compiliert nach WebAssembly)
Go	✓	✓	TinyGo
Rust	✓	✓	
Kotlin	✓	⌛	Via Kotlin Native
Grain	✓	✓	WebAssembly „native“
Motoko	✓	✓	WebAssembly „native“

sidion

STATUS

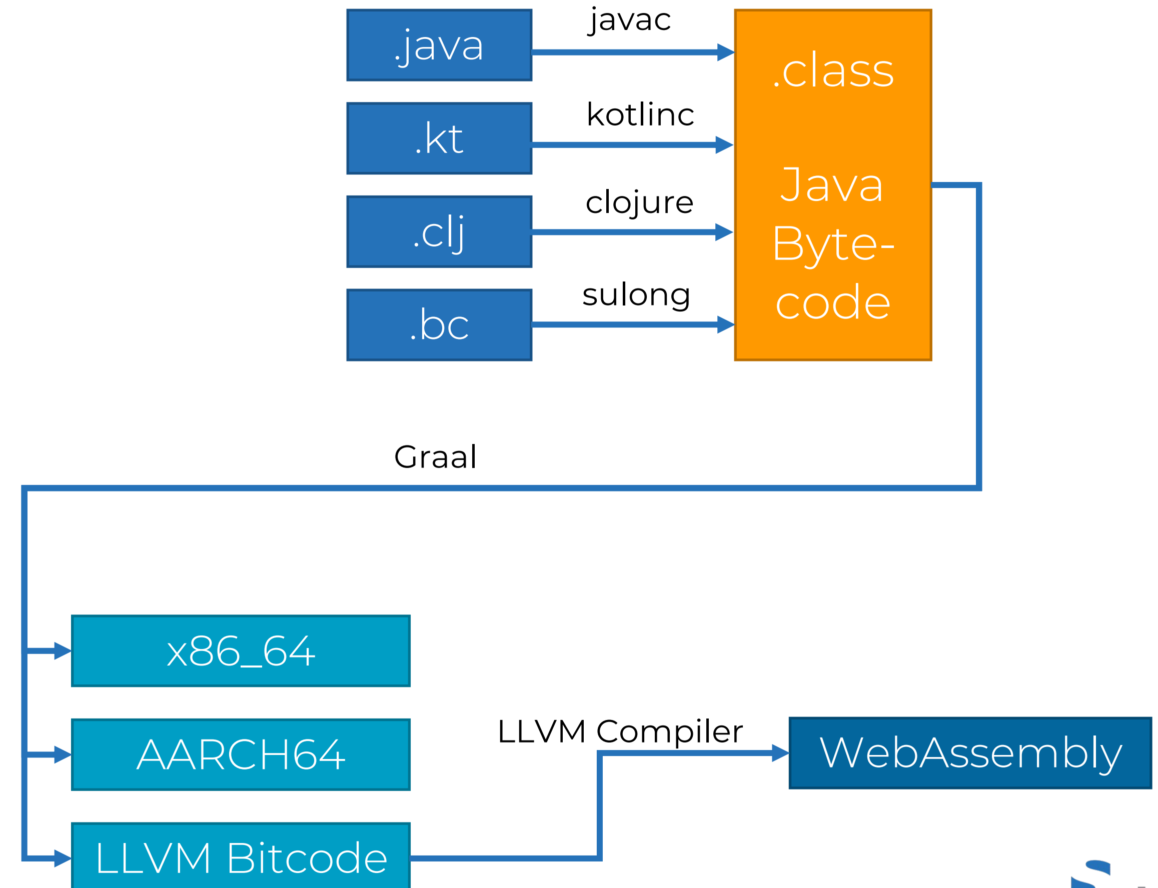
JAVA UND WEBASSEMBLY

Mehrere auf den Browser fokussierte Projekte:

- JWebAssembly
  - Grundlegene Funktionen vorhanden
  - Methodeaufrufe möglich
  - String und Enums
  - Interaktion mit JavaScript möglich
  - GC wird durch JavaScript Polyfill realisiert
- Bytecoder
  - transpiles nach JavaScript oder WebAssembly
  - JVM Bytecode Frontend für LLVM
  - Integration mit anderen Frontends wie vue.js mögllich
  - GC wird durch JavaScript Ployfill realisiert
- TeaVM
  - experimenteller WebAssembly Support
  - Keine Dokumentation
  - Für Web-Frontends Designed
  - GC wird durch JavaScript Ployfill realisiert

JAVA UND WEBASSEMBLY

- Mehrere Projekte, alle auf den Browser fokussiert
  - Bytecode Project
  - TeaVM
  - JWebAssembly
  - CheerpJ
- Theoretische Möglichkeit
  - Java → GraalVM → LLVM bitcode → WebAssembly
  - Im Moment aber noch nicht möglich
    - Offenes Issue (https://github.com/oracle/graal/issues/3391)
    - Führt zu Proposal (https://github.com/WebAssembly/gc)



sidion

WASM VS. CONTAINER

Vor- und Nachteile

Photo by
Guillaume Bolduc
on Unsplash

sidion

VORTEILE VON CONTAINER

WARUM CONTAINER?

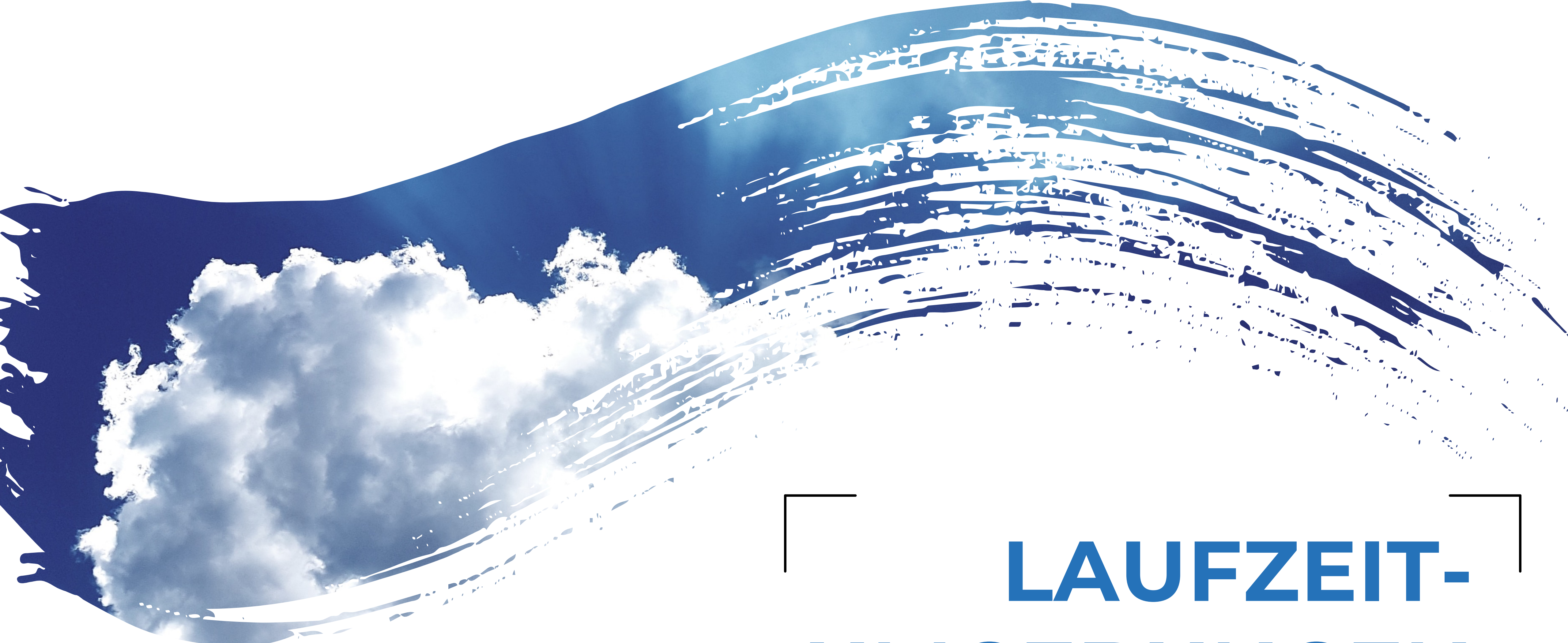
- Portabilität - Fähigkeit, überall zu laufen
  - Anwendung bringt Laufzeitumgebung mit
- Ressourceneffizienz und Datendichte
  - Leichtgewichtiger als VMs
- Container-Isolierung und gemeinsame Nutzung von Ressourcen
  - Container voneinander getrennt
- Geschwindigkeit
  - Starten, Erstellen, Replizieren oder Zerstören von Containern in Sekundenschnelle
- Hohe Skalierbarkeit
  - selbes Image mehrfach starten
- Verbesserte Entwicklerproduktivität
  - Entwickler können vollständige Anwendung zur Verfügung stellen
  - Geringe Abstimmung mit Infrastruktur oder Betrieb notwendig

sidion

VORTEILE VON WEBASSEMBLY

WARUM WASM?

- Polyglotte Entwicklung
  - C, Rust, Go, Ruby, Python es wird immer nur eine WebAssmebly Runtime benötigt
- Capability-based Security
  - Nur explizit freigegebene Ressourcen sind zugänglich
- Kann von anderen Sprachen sicher eingebettet werden
  - z.B. um nicht vertrauenswürdigen Code sicher auszuführen
- Schneller Start der Anwendung
  - Eines der Designziele von WebAssembly
- Kleinere Anwednungen
  - WebAssembly ist auf kleine Dateigrößen optimiert
- Kleinere Laufzeitumgebungen



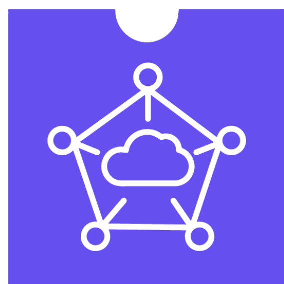
LAUFZEIT-UMGEBUNGEN

Wo kann WebAssembly ausgeführt werden?

Photo by Gabriel Lamza on Unsplash

sidion

LAUFZEITUMGEBUNGEN



WasmEdgeRuntime

KRUSTLET

sidion

WO KANN WEBASSEMBLY AUSGEFÜHRT WERDEN?

LAUFZEITUMGEBUNGEN

Standalone

Kubernetes

KRUSTLET

Framework + Runtime

sidion

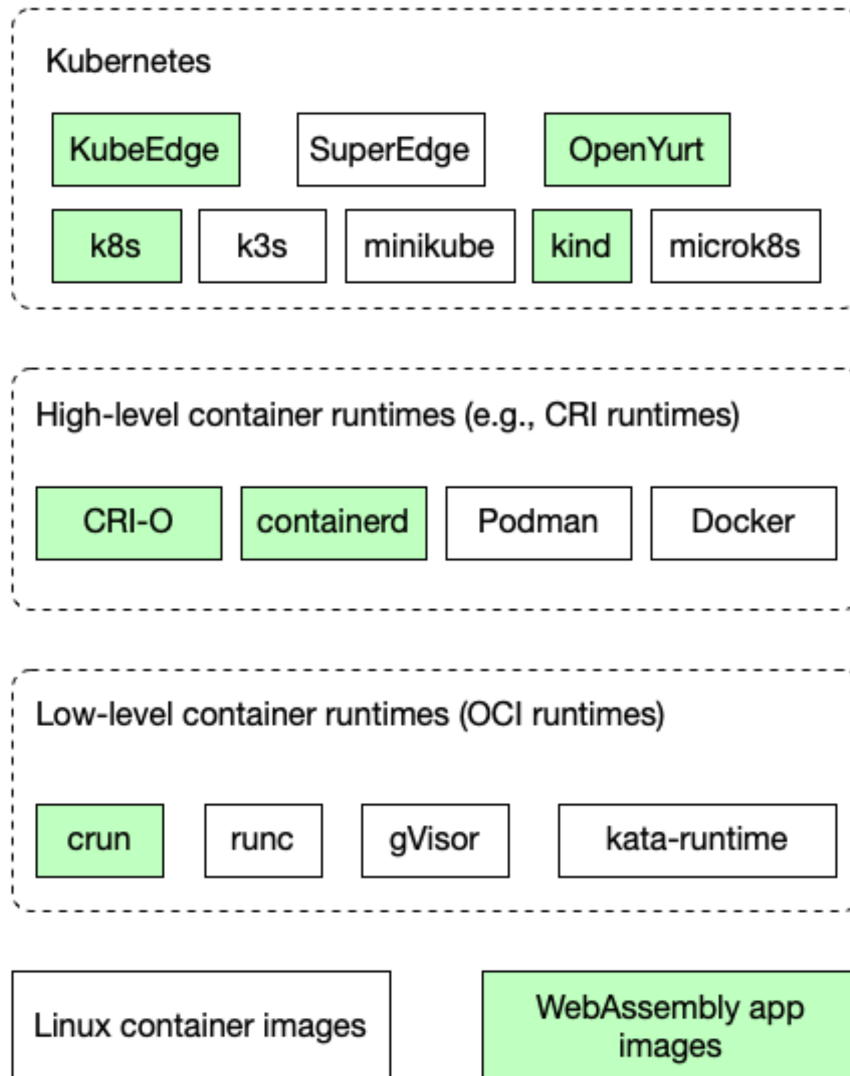
```
tmp — nma@neuromancer — /tmp — -zsh — 49×24
/tmp wasmer run cowsay.wasm Hello ✔
 _______
< Hello >
 -------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
               ||----w |
                ||     ||
/tmp wapm install cowsay ✔
[INFO] Installing syrusakbary/cowsay@0.2.0
Package installed successfully to wapm_packages!
/tmp wapm run cowsay Hello World ✔
 _____________
< Hello World >
 -------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
               ||----w |
                ||     ||
/tmp ✔
```

STANDALONE

WASMER

- Selbes Deployment-Modell wie bei Containern
  - Applikation und Runtime wird gemeinsam geliefert
- wasmer enthält einen Paketmanager
  - zwischen npm und Docker Hub
- Unterstützt verschiedene Compiler
  - Singlepass, Cranelift, LLVM
- Caching von bereits übersetztem WebAssembly Code

sidion



The container ecosystem

KUBERNETES

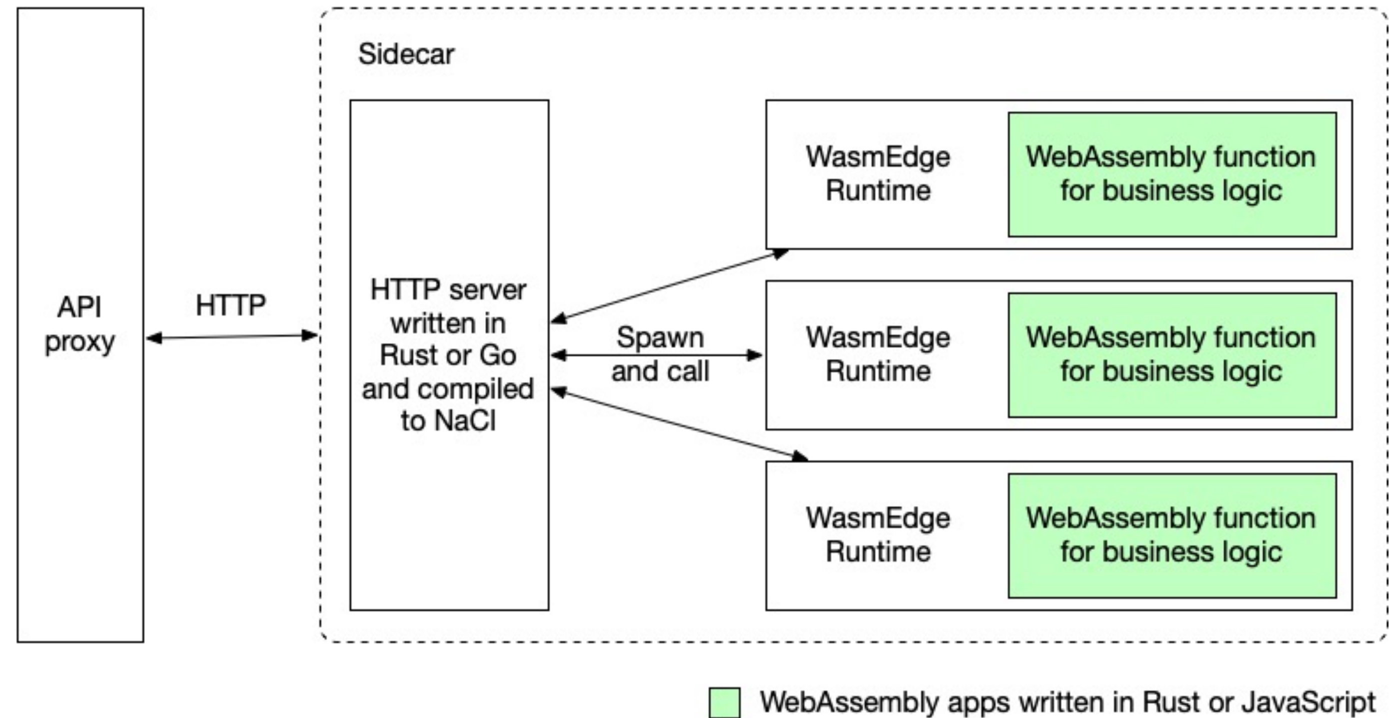
WASMEDGE RUNTIME

- crun enthält WasmEdge Support
- CRI-O oder containerd nutzen crun
- Kubernetes Distributionen nutzen wiederum CRI-O oder containerd um WebAssembly Anwendungen auszuführen
- KubeEdge, SuperEdge oder OpenYurt
  - Kubernetes für Edgecomputing

sidion

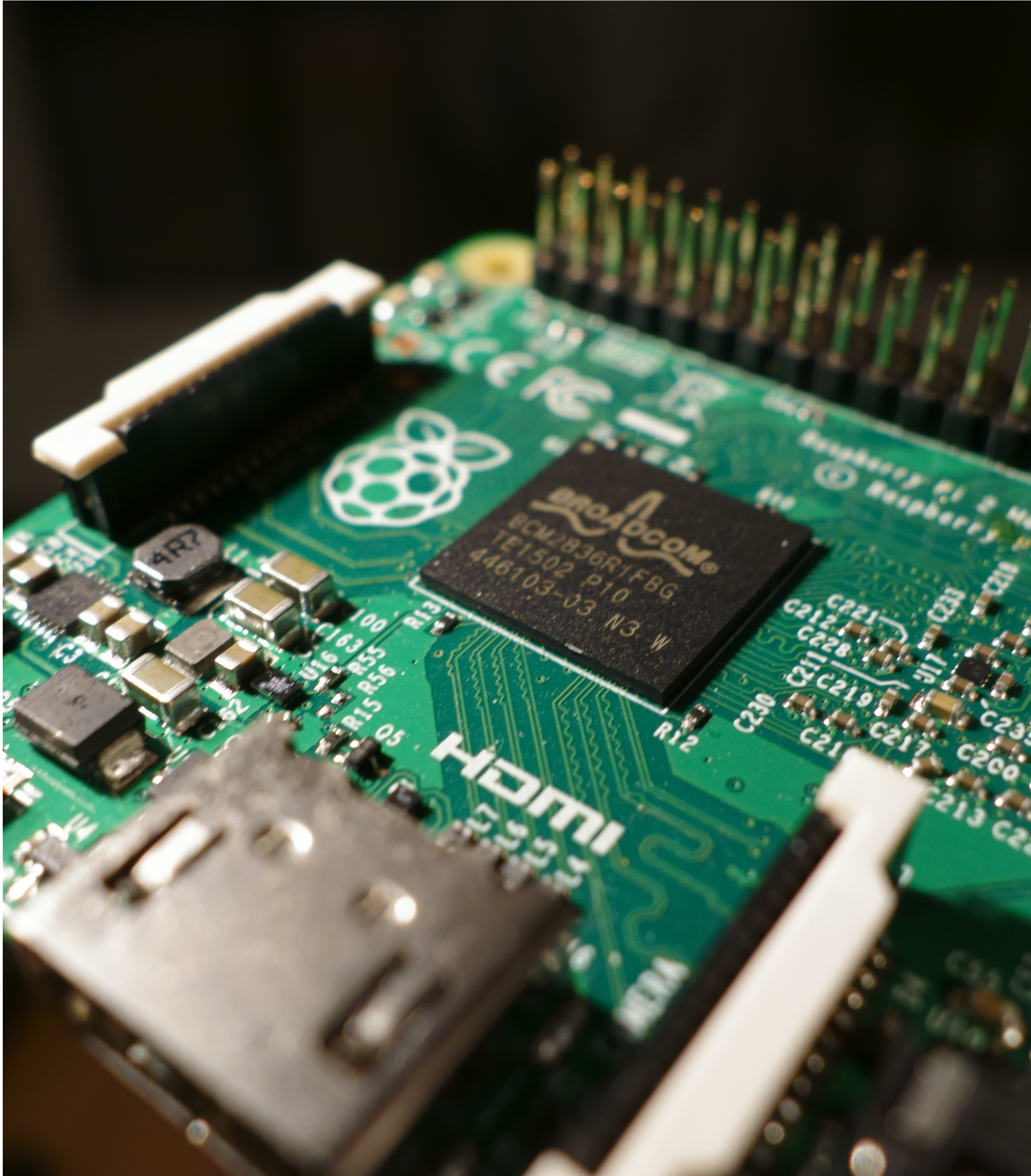
WASMEDGE RUNTIME

- Service Mesh Runtimes
  - dapr
  - Apache Event Mesh
- Serverless platforms
  - Vercel
  - Netlify
  - AWS Lambda



dapr – Distributed Application Runtime

sidion



STANDALON

WEBASSEMBLY MICRO RUNTIME

- Kleine Runtime
  - Nur ~85K für den Intepreter, ~50K für AOT
- Viele Unterstützte Plattformen
  - X86-64, ARM, RISCV64, RISCV32, XTENSA, MIPS, ARC
- Application Framework mit vielen Funktionen
  - Timer, Inter-App-Kommunikation (Request/Response und Pub/Sub), Sensor, Konnektivität und Datenübertragung, 2D-Grafik-UI
- Verwendet zum Beispiel in
  - Envoy Proxy
  - Apache Teaclave

sidion

STANDALONE

WAGI

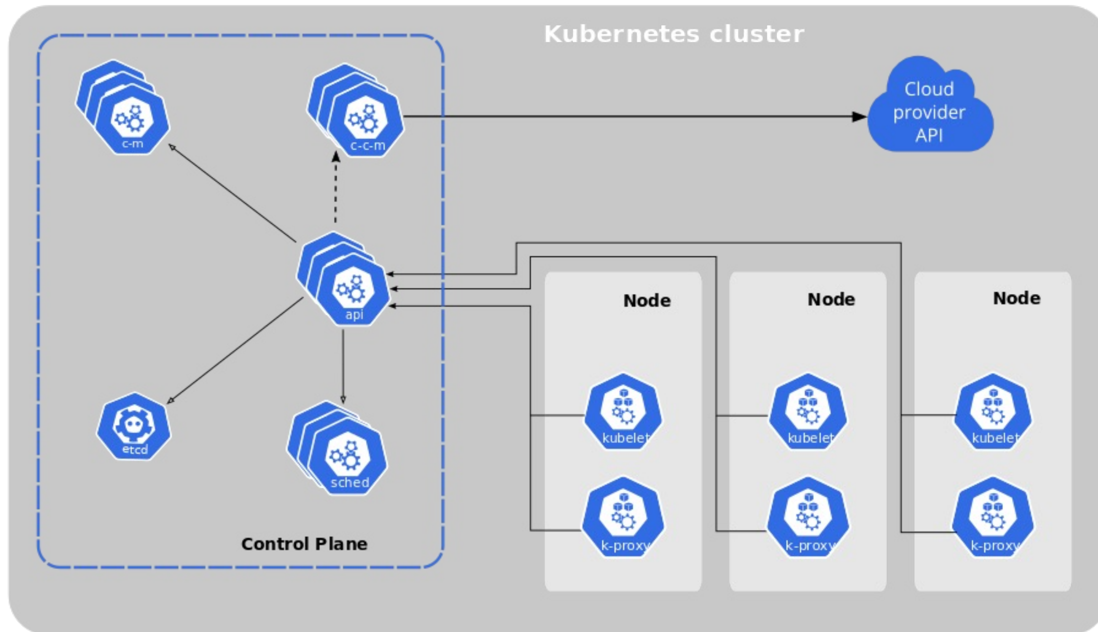
WebAssembly Gateway Interface

1. Umgebungsvariablen für Header, Pfad, Queryparameter lesen
2. Uploads über STDIN akzeptieren
3. Ausgabe auf STDOUT schreiben

Jedes `WASM32-WASI` WebAssmebly Programm wird damit webtauglich.

```
fn main() {
    println!("Content-Type: text/plain");
    println!();
    println!("Hello world");
}
```

KRUSTLET



- Krustlet verhält sich wie ein kubelet
  - lauscht an der k8s API ob neue Pods erzeugt werden sollen

1. `kubelet` empfängt ein Pod-Ereignis aus dem Stream
2. Je nach Ereignistyp (hinzufügen, ändern, löschen) ruft es die entsprechende `Provider`-Methode auf und erstellt, aktualisiert oder stoppt/löscht den "Container".
3. Der `Provider` macht seine Arbeit und gibt einen Fehler zurück, wenn es ein Problem gibt

- Krustlet verwendet wasmtime als `Provider`

sidion

KRUSTLET

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: krustlet-tutorial
spec:
  containers:
    - name: krustlet-tutorial
      image: my.container.registry/krustlet-tutorial:v1.0.0
  imagePullSecrets:
    - name: <acr-secret>
  tolerations:
    - key: "kubernetes.io/arch"
      operator: "Equal"
      value: "wasm32-wasi"
      effect: "NoExecute"
    - key: "kubernetes.io/arch"
      operator: "Equal"
      value: "wasm32-wasi"
      effect: "NoSchedule"
```

- Kubelet um Webassembly auszuführen
- Krustlet verwendet Taints und Tolerations
  - um den Node zu selektieren
- wasm-to-oci um Wasm Module zu teilen
  - Module können wie Container in eine Registry gespeichert werden

→ Einfach bestehenden Cluster um einen Krustlet Node erweitern, dann kann können WebAssembly Programme ausgeführt werden

sidion

WASMCLOUD

- Verteilte Plattform für Business Logik
  - von der Edge bis zur Cloud
- Fokus auf Business Logik
  - http Fähigkeiten werden über Capability Provider zur Verfügung gestellt
- Aktoren reagieren auf Nachrichten
  - NATS als Messaging Provider
- Weitere Provider verfügbar
  - z.B. für Redis oder PostgeSQL
- Deny-by-Default
  - Ein Aktor kann auf keinen der Provider, ohne ausdrückliche Erlaubnis, zugreifen

```rust
use wasmbus_rpc::actor::prelude::*;
use wasmcloud_interface_httpserver::{HttpRequest, HttpResponse, HttpServer, HttpServerReceiver};

#[derive(Debug, Default, Actor, HealthResponder)]
#[services(Actor, HttpServer)]
struct HelloActor {}

/// Implementation of HttpServer trait methods
#[async_trait]
impl HttpServer for HelloActor {

    /// Returns a greeting, "Hello World", in the response body.
    /// If the request contains a query parameter 'name=NAME', the
    /// response is changed to "Hello NAME"
    async fn handle_request(
        &self,
        _ctx: &Context,
        req: &HttpRequest,
    ) -> std::result::Result<HttpResponse, RpcError> {
        let text = form_urlencoded::parse(req.query_string.as_bytes())
            .find(|(n, _)| n == "name")
            .map(|(_, v)| v.to_string())
            .unwrap_or_else(|| "World".to_string());

        Ok(HttpResponse {
            body: format!("Hello {}", text).as_bytes().to_vec(),
            ..Default::default()
        })
    }
}
```

WASMCLOUD

- wasmCloud hosts können auf verschiedenen Geräten laufen
  - vom Raspberry Pi über Virtuelle Server bis zu Kubernetes Pods
- Durch NATS entsteht ein selbst-formendes, selbst-heilendes Mesh-Netzwerk
  - Aktoren und Proider können darüber ohne Service Discovery kommunizieren
- zero trust Security Modell

sidion

SPIN

- Framework für event-driven Microservices
  - Fokus auf Business Logik
- HTTP und Redis Trigger eingebaut
  - Konfiguration über TOML Datei
- Grundlegende Funktionen vorhanden
  - Version 0.1 verfügbar

```rust
#[http_component]
fn hello_world(_req: http::Request) -> Result<http::Response> {
    Ok(http::Response::builder()
        .status(200)
        .body(Some("Hello, Fermyon!".into()))?)
}
```

```toml
spin_version = "1"
name = "spin-hello-world"
trigger = { type = "http", base = "/" }
version = "1.0.0"

[[component]]
id = "hello"
source = "<path to compiled Wasm module>"
[component.trigger]
route = "/hello"
```


sidion

WANN LOHNT SICH WEBASSEMBLY?

FAZIT

- Umgebungen in denen Services oft gestartet und gestoppt werden
  - zum Beispiel Serverless
- Als einheitliche Runtime wenn Polyglott entwickelt wird
  - Jedes Team kann für sich passende Sprache wählen
- Kleinere Deployment-Artefakte
  - Vor allem bei
- Höhere Sicherheit durch Capability-based Security
- Sehr kleine Runtimes möglich
  - zwischen 50k (AOT) und 85k (interpretiert)

sidion

Fragen?