DevOpsCon

Think container orchestration different – WASM is coming

Max Körbächer | Co-Founder of Liquid Reply

Who are we?

Max Körbächer

Co-Founder and Kubernetes Platform Engineer at Liquid Reply

Former Enterprise Architect, focusing on Kubernetes and Cloud Native Infrastructure.

Contributing to the Kubernetes release team and related K8s technologies

Servant for a 🐈

Christoph Voigt

Co-Founder and developer of Liquid Reply

Software Engineering background, having a focus on Cloud Native Infrastructure- and Application-Architectures

Contributing to the Kubernetes release team and related K8s technologies

Father of two 👨‍👧‍👧

Liquid Reply is the
Kubernetes and Cloud-Native
consultancy of the Reply Network.

We help our clients entangling difficulties of modern IT-Infrastructure, developing, architecting and teaching cloud-native technologies.

Todays Journey

WebAssembly (WASM) is at an inflection point:

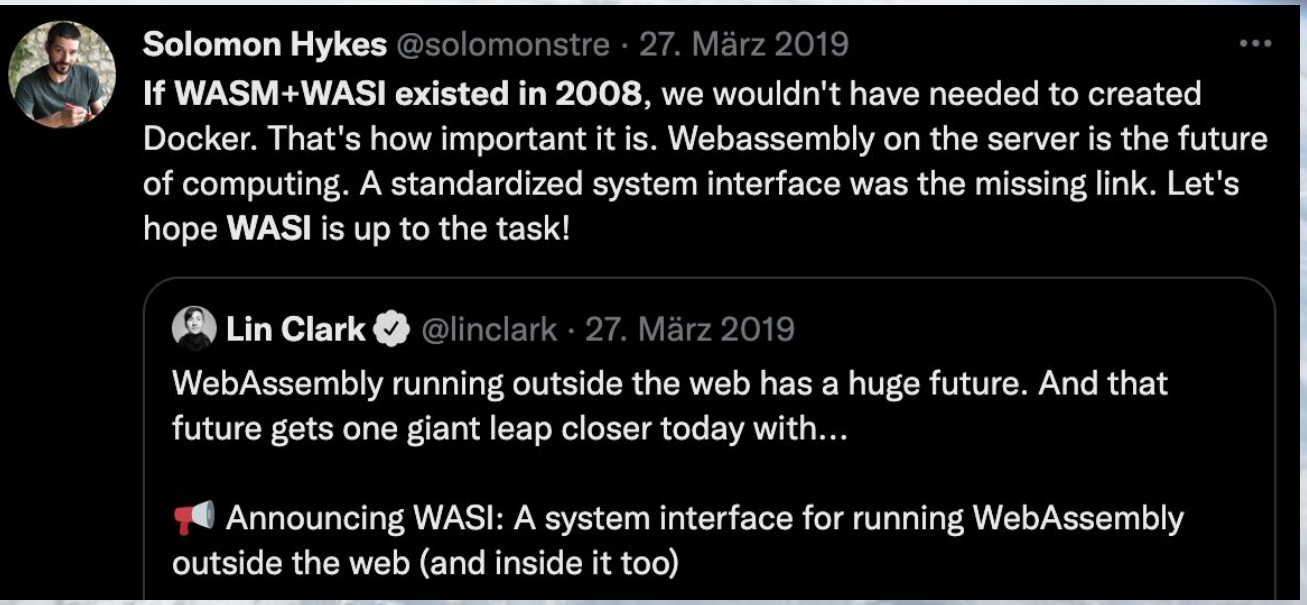
Over the next few years, we expect to see increased adoption of WebAssembly across the tech sphere, from containerization to plugin systems to serverless computing platforms

What is WASM and how is it even relevant? 🧐

What is the status quo of the WASM ecosystem? 🏃‍♀️

Conclusion and look into the glass bowl 🔮

WebAssembly Intro



src: https://twitter.com/solomonstre/status/1111004913222324225

What is WebAssembly?

Think of it as an intermediate layer between **various programming languages** and **many different execution environments**. You can take code written in over 30 different languages and compile it into a *.wasm file, and then can execute that file on any WASM Runtime.

The name “**WebAssembly**” is misleading. Initially designed to make code run fast on the **web**, today it can run anywhere.

WebAssembly is:

- stack-based VM executing binary file formats
- CPU-agnostic -> taking any architecture
- OS-agnostic
- Entirely depends on the host runtime (we will talk later about it)

WebAssembly oversimplified:

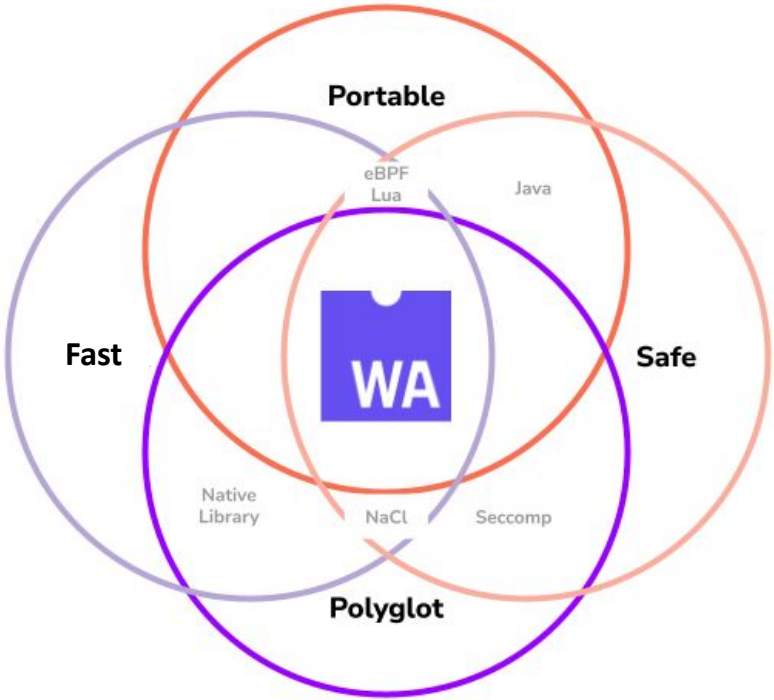
⚡ **Consistently fast**

🔬 **Small**

🌍 **Universal**

♻️ **Reusable**

Benefits of WebAssembly



Slightly adjusted; Source: https://b-nova.com/home/content/containerless-mit-webassembly-runtimes*

Where can WebAssembly be applied?

***outside the Browser**

Language Interoperability

Write that library once in a language of your choice; use in any language.

Figma
Lichess.org
Google Earth
Adobe Photoshop

Plugin Systems

Never trust third parties!

Envoy / Istio
Kubewarden
MS Flight Simulator
Minecraft
RedPanda

Embedded Sandboxing

Prevent yourself against bugs of third party libraries.

Firefox
HTTP Servers

Blockchains

Write Smart Contracts in a language of your choice.

CosmWasm
eWASM

Containerisation

Universal Runtime, capability based security model.

Krustlet
Hippo
wasmCloud
Lunatic
WasmEdge

Serverless Platforms

Minimal Startup time, maximal isolation.

Cloudflare Workers
AWS Lambda
Atmo (Suborbital)
Fastly Compute@Edge

A new paradigm ahead?

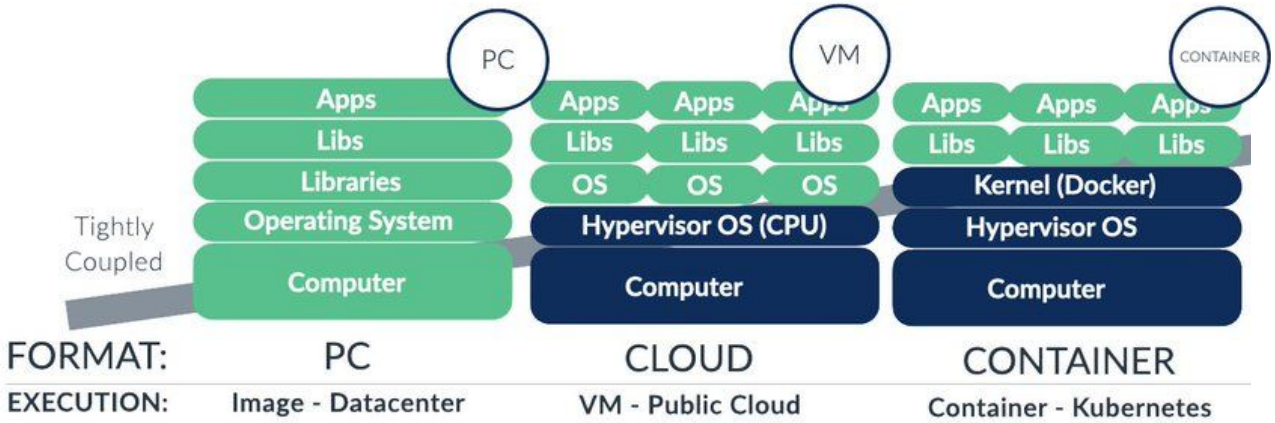


Image Source (adjusted): Liam Randall (@Hectaman) - https://twitter.com/Hectaman/status/1389585069073895434

A new paradigm ahead?

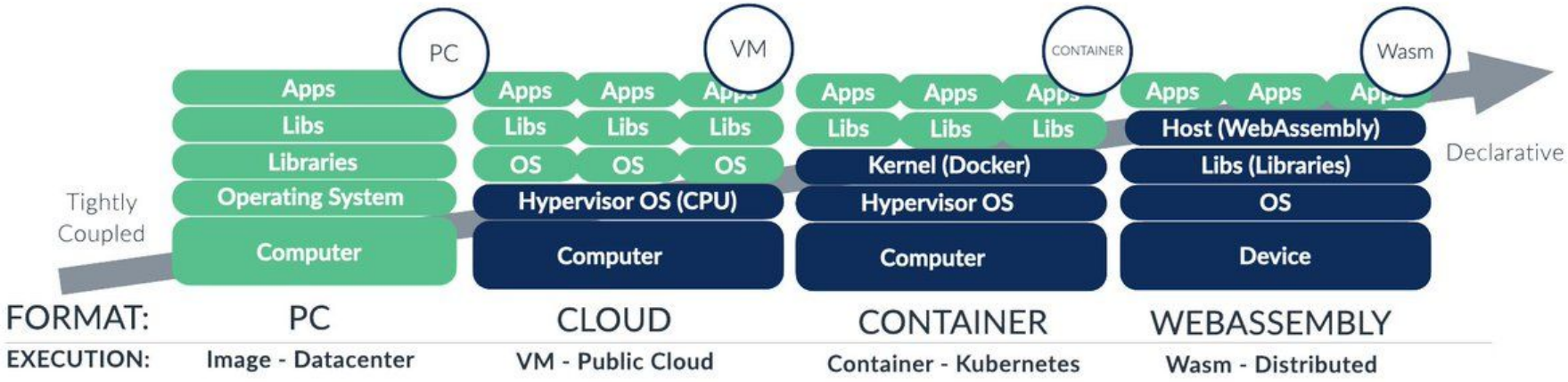


Image Source (adjusted): Liam Randall (@Hectaman) - https://twitter.com/Hectaman/status/1389585069073895434

Some WASM implementations

(a subjective choice)

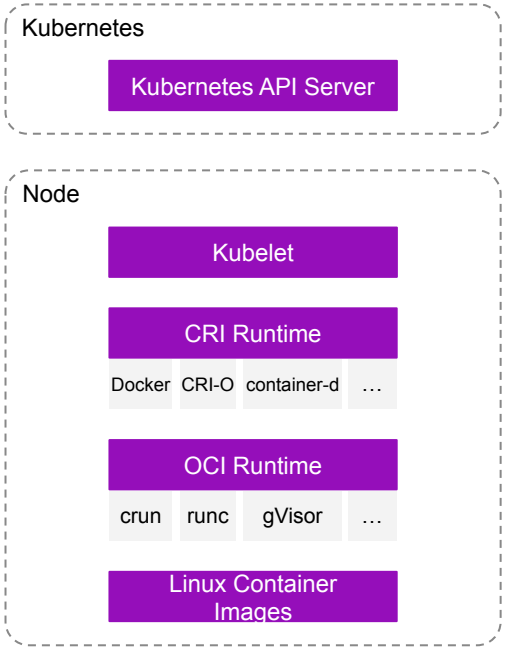
Krustlet

What problem does it aim to solve?

- Kubernetes Cluster technology could be a good fit to orchestrate WASM modules similar to containers
- The advantages of WASM modules in a cluster compared to a Container?
  - security sandboxed by default
  - reduce upstart time
  - decreases footprint
  - hardware (host) independent (hi arm/x86 containers!)

- instead of containers, we want to start wasm runtimes

KRUSTLET



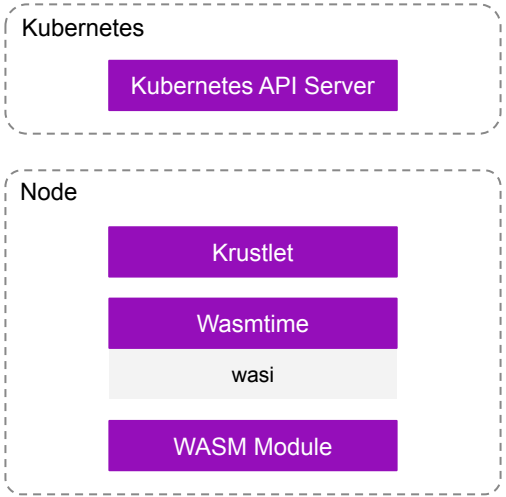
A Regular Kubernetes Stack

Krustlet

Solution Approach

- Krustlet acts as a Kubelet by listening on the event stream for new pods that the scheduler assigns to it based on specific Kubernetes tolerations.

- The default implementation of Krustlet listens for the architecture wasm32-wasi and schedules those workloads to run in a wasmtime-based runtime instead of a container runtime.



A Krustlet Kubernetes Stack

Krustlet

Solution Approach

- Krustlet acts as a Kubelet by listening on the event stream for new pods that the scheduler assigns to it based on specific Kubernetes tolerations.

- The default implementation of Krustlet listens for the architecture wasm32-wasi and schedules those workloads to run in a wasmtime-based runtime instead of a container runtime.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: hello-wasm
spec:
  containers:
  - name: hello-wasm
    image: webassembly.azurecr.io/hello-wasm:v1
  tolerations:
  - effect: NoExecute
    key: kubernetes.io/arch
    operator: Equal
    value: wasm32-wasi
  - effect: NoSchedule
    key: kubernetes.io/arch
    operator: Equal
    value: wasm32-wasi
```

Krustlet

Solution Approach

Advantages

- ➕ add “wasm nodes” to your cluster without changing the entire cluster setup
- ➕ use the same Pod-Spec as for your normal Pods
- ➕ CSI support
- ➕ Plugin-Support

Considerations

- ➖ either Kubelet OR Krustlet
- ➖ as there is no container runtime, you need toleration configs to avoid scheduling of “normal” cluster-wide daemonset* (e.g. CNI)
- ➖ your modules are only allowed to do what the runtime permits → no Network for your modules!
- ➖ wasi + wasmtime under heavy development → so is Krustlet

Try it in Kind:
https://github.com/Liquid-Reply/kind/tree/kind-krustlet

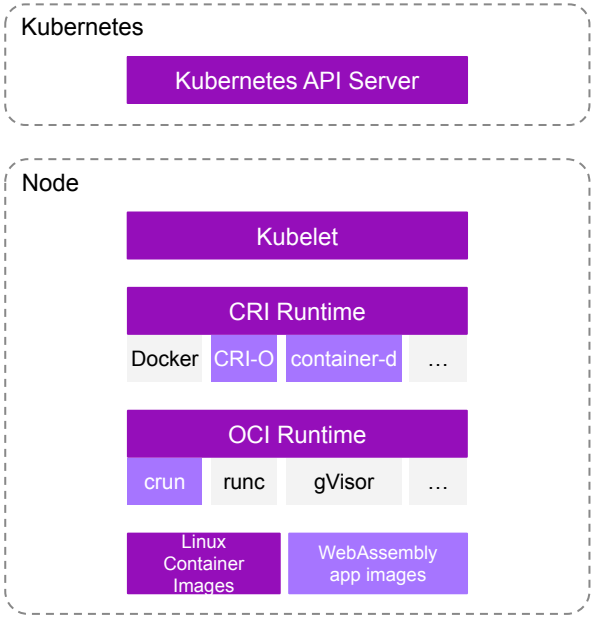
* There is a Container Runtime Interface provider implementation for Krustlet. This runtime allows you to run the containers you know and love within Krustlet.

WasmEdge

Integrating with existing tooling, and more ...

- Aims to solve similar problems as Krustlet, but in a more flexible and leaner way
- Especially targets the integration in various Kubernetes distributions, CRI runtimes as well as OCI runtimes - therefore a good match to run WASM side by side with classic containers
- Runs also stand alone for modern web apps, to host serverless functions and being “embedded” in any kind of edge device.

WasmEdgeRuntime



The Container Eco-System

based on: https://wasmedge.org/book/en/kubernetes.html

WasmEdge
Solution Approach

WasmEdge is different on the image level. Rather than having a container image with a OS, the WASM image is build from scratch. In addition, the container requires an “wasm.image” annotation, to let crun and containerd know that it use WasmEdge.

This approach allows to use WASM within the Kubernetes context, and utilize the existing ecosystem.

```
FROM scratch
ADD http_server.wasm /
CMD ["/http_server.wasm"]
```

*http server wasm image within a docker file

```
sudo buildah build --annotation "module.wasm.image/variant=compat" -t http_server .
```

*a wasm container requires the wasm image annotation

WasmEdge

Solution Approach

Advantages

- ➕ WasmEdge can run alongside your standard containers
- ➕ Build and deployment spec are nearly the same as for a normal pod
- ➕ Supports different CRI, OCI and K8s distros
- ➕ Can use existing K8s ecosystem
- ➕ Runs by itself on edge, serverless or browser

Considerations

- ➖ Additional tools for image annotation are required (at the moment)
- ➖ For some use cases you need another SDK
- ➖ It can lead to confusion that you can use WasmEdge in very different scenarios and each of them has to be developed differently

From all tools we show today, WasmEdge would be the best choice to extend your currently orchestration without deep cutting changes

WasmCloud

What problem does it aim to solve?

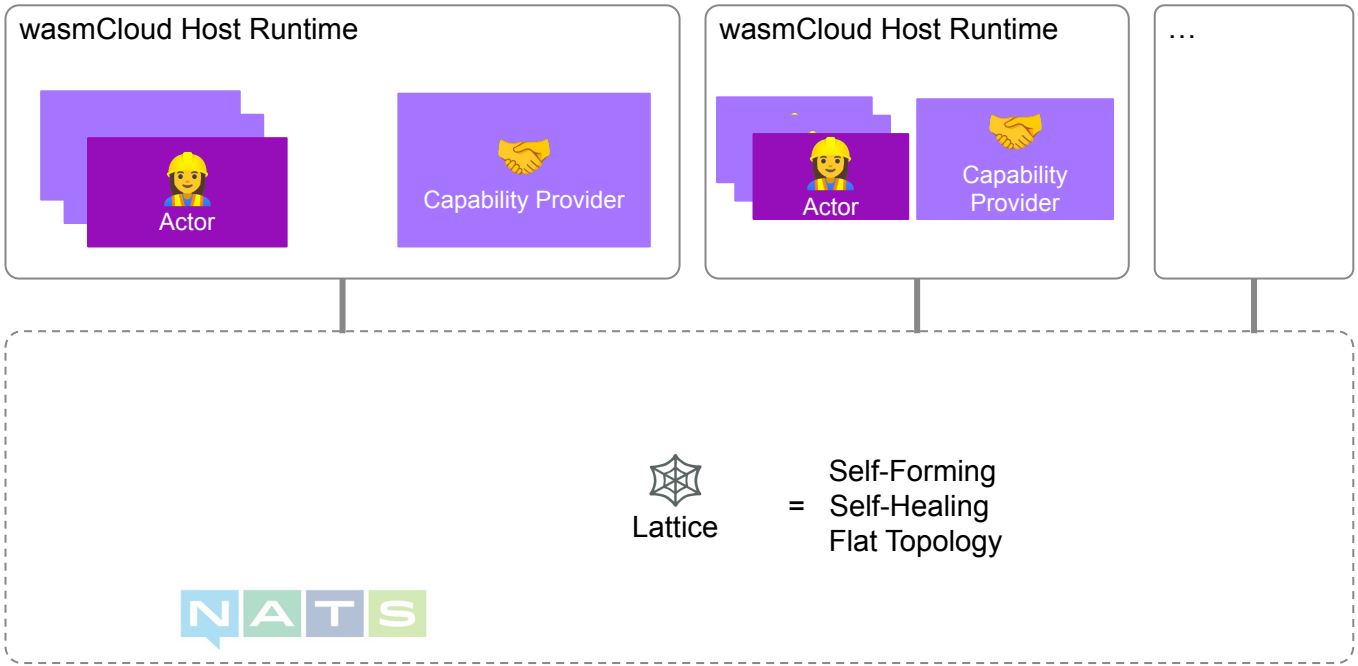
- wasmCloud is a distributed platform for writing portable business logic that can run anywhere from the edge to the cloud. Secure by default, wasmCloud aims to strip wasteful boilerplate from the developer experience.

- Business-Applications contain a lot of boilerplate:
  - Webserver
  - integrated dependencies (Database, Caches)
  - tight coupling to non-functional requirements
  - Security (certificates etc.)
  - …

- Only a fraction is actual business logic

WasmCloud

The Solution

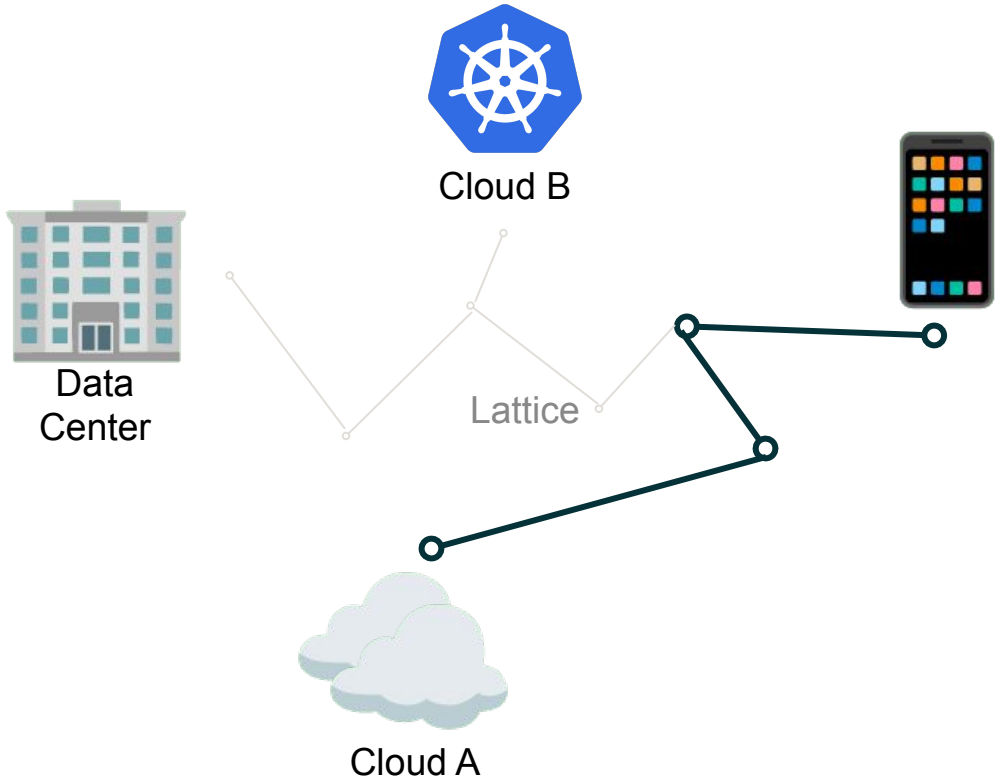


Three core concepts:

- Actor
- Capability Provider
- Lattice

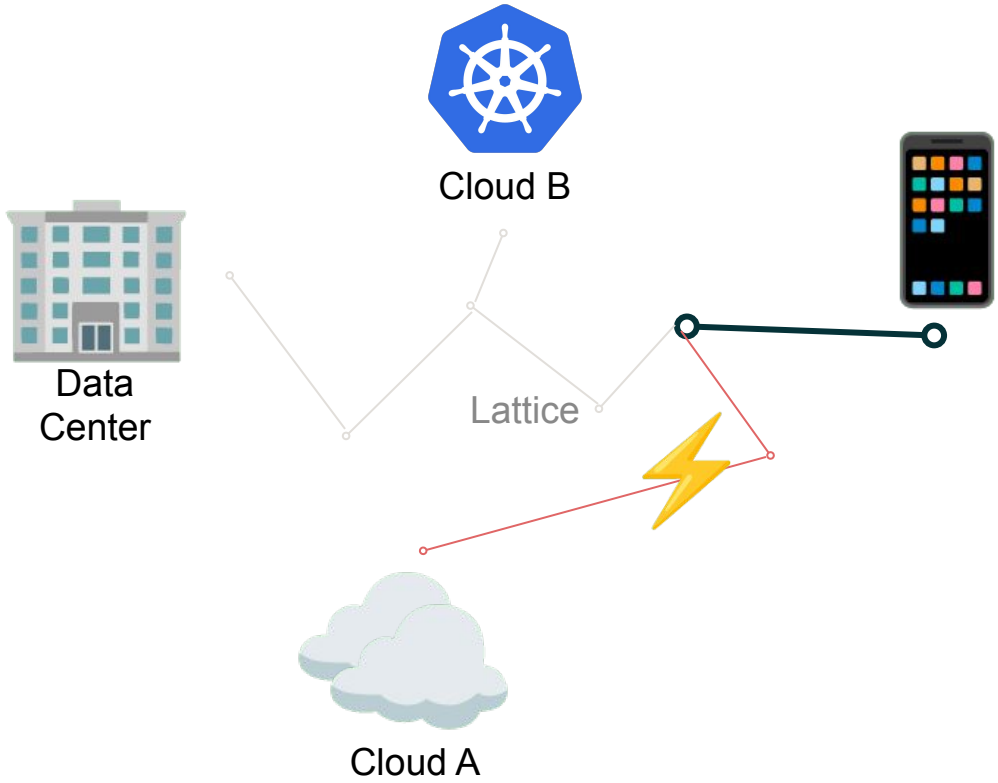
WasmCloud

Reach and Resilience backed by the Lattice



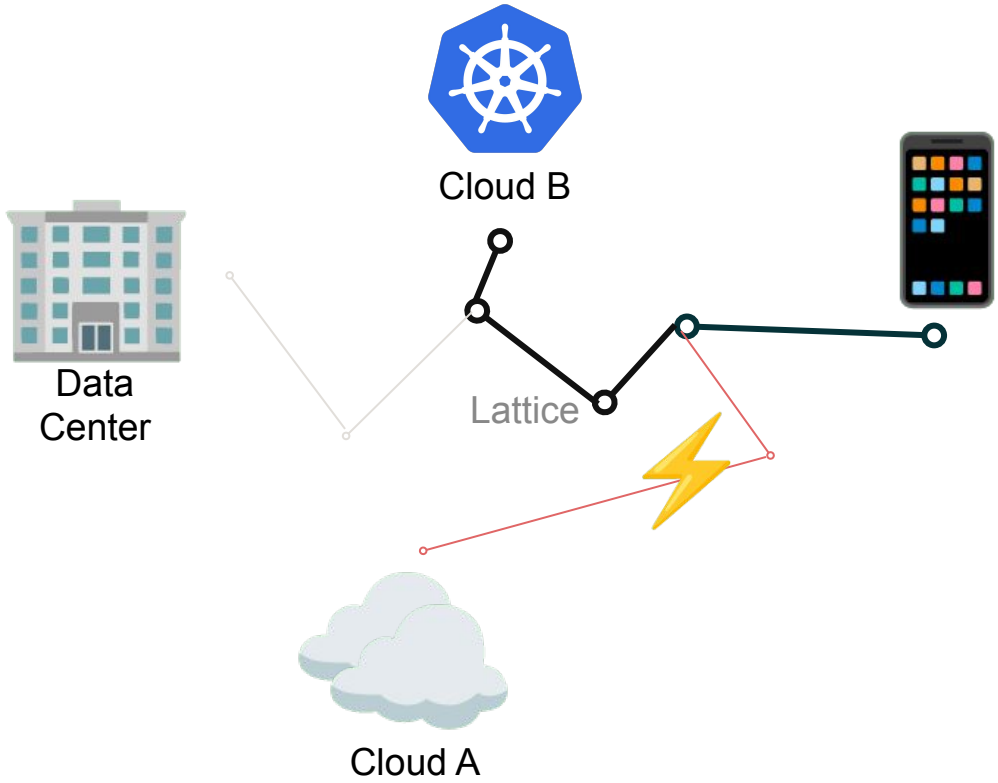
WasmCloud

Reach and Resilience backed by the Lattice



WasmCloud

Reach and Resilience backed by the Lattice



WasmCloud

Solution Approach

Advantages

- ➕ high focus on writing business logic
- ➕ potentially high reusability of WASM modules
- ➕ high isolation
- ➕ high amount of security
- ➕ high resiliency
- ➕ HostRuntimes can run “anywhere” (Bare metal, VM, Container, Kubernetes, Webbrowser…)

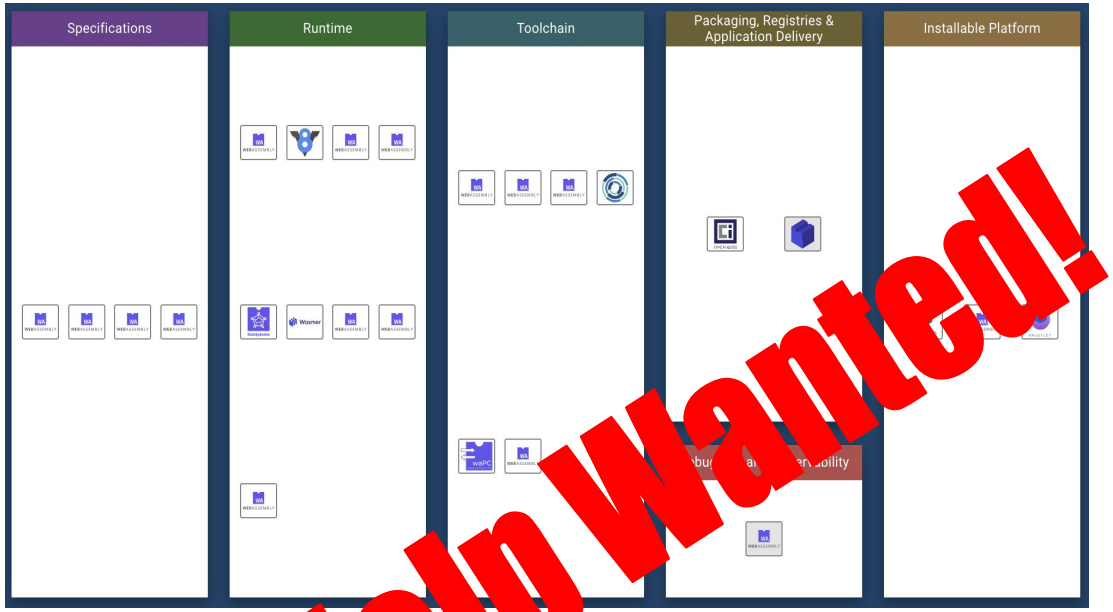
Considerations

- ➖ applications need to be written with WasmCloud in mind
- ➖ currently Rust is the only supported language; though other languages are planned
- ➖ still very young project - expect rough edges
- ➖ tooling for debugging and monitoring rudimentary

Summary

WASM Ecosystem

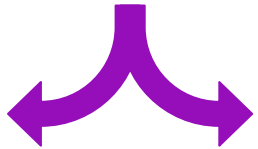
- The WASM ecosystem is under heavy development
- Many tools are new or getting continuously reshaped
- This is reflected in the CNCF WASM landscape



A missing extension?

	Docker-like container	WebAssembly
Performance	OK	Great
Resource footprint	Poor	Great
Isolation	OK	Great
Safety	OK	Great
Portability	OK	Great
Security	OK	Great
Language and framework choice	Great	OK (yet)
Ease of use	Great	OK (yet)
Manageability	Great	Great

Containers for lifting, WASM for re-creating



Go with the Container flow

Containers will stay and drastically increase in usage over the next years.

Build with WASM for the future

But for future developments WASM might be in many cases a better choice.

We believe that WASM & Container will go along side by side

Conclusion

1. WebAssembly's potential is beyond the browser
2. WASM enables use cases that are not possible with container & K8s
3. WASM will not substitute containers & K8s, but extend them
4. WASM lacks harmonization and makes it difficult for programming languages to adapt
5. The developer experience of/for WASM will be the game changer


WASM will be
ubiquitous

Sources

- https://www.infoworld.com/article/3651503/the-rise-of-webassembly.html
- https://harshal.sheth.io/2022/01/31/webassembly.html ***
- https://nickymeuleman.netlify.app/blog/webassembly ***
- https://docs.krustlet.dev/topics/architecture/
- https://docs.krustlet.dev/topics/providers/
- https://github.com/Liquid-Reply/kind/tree/kind-krustlet (Krustlet baked into Kind:)
- https://bytecodealliance.org/articles/announcing-the-bytecode-alliance ***
- https://thenewstack.io/what-is-webassembly/
- https://www.youtube.com/watch?v=vqBtoPJoQOE
- https://istio.io/latest/docs/concepts/wasm/
- https://www.kubewarden.io/
- https://docs.flightsimulator.com/html/Programming_Tools/WASM/WebAssembly.htm
- https://hacks.mozilla.org/2021/12/webassembly-and-back-again-fine-grained-sandboxing-in-firefox-95/
- https://almanac.httparchive.org/en/2021/webassembly
- https://harshal.sheth.io/2022/01/31/webassembly.html
- https://github.com/deislabs/hippo
- https://github.com/lunatic-solutions/lunatic
- https://github.com/suborbital/atmo
- https://blog.cloudflare.com/webassembly-on-cloudflare-workers/
- https://www.fastly.com/blog/how-compute-edge-is-tackling-the-most-frustrating-aspects-of-serverless
- https://cosmwasm.com/
- https://github.com/ewasm/design
- https://wasmcloud.dev/reference/host-runtime/

Thank you!

MEETING AGENDA

OPTIONAL SUBTITLE

A new elephant is in the cloud native and Kubernetes room and its name is WASM (WebAssembly). The rumors are spread; if WASM was invented two years earlier, Docker would never have been born. But what does all this hype mean? Is WASM the new Docker? Why should I use it, and why I shouldn't? In this talk, Max and Chris will walk you through the current state of WASM in the cloud native ecosystem, which new possibilities it brings, and where it's just starting to crawl. You will get to know how you can use it in existing implementations like Kubernetes and which scenarios it unleashes its power by bringing new concepts.

45MIN

M: Beschreibung des Überblick & Status Quo

Artikel: is Wasm the Kubernetes? falsche betrachtung

- Warum relevant?
  - Wie spielt WASM mit Container(Orchestrierung) zusammen?
    i. C: Gegenüberstellung Conainer Security vs. WASM Security "Thread Model"?
    ii. M: Container/Kubernetes sind super backwards compatible und erfordern minimale anpassungen an legacy -> WASM muss from scratch geschrieben werden, kann dafür universeller betrieben werden
  - Brainstorm: Wo steht das Ökosystem heute eigentlich?
    i. Krustlet
    ii. WASMEdge?
    iii. WASMCloud
    iv. Atmo / Hippo?
    v. Ökosystem?
      1. WASM CNCF Landscape
- M: Closing Remark
  - WASM will be ubiquitous

Where can WebAssembly be applied?

Sources

- https://istio.io/latest/docs/concepts/wasm/
- https://www.kubewarden.io/
- https://docs.flightsimulator.com/html/Programming_Tools/WASM/WebAssembly.htm
- https://hacks.mozilla.org/2021/12/webassembly-and-back-again-fine-grained-sandboxing-in-firefox-95/
- https://almanac.httparchive.org/en/2021/webassembly
- https://harshal.sheth.io/2022/01/31/webassembly.html
- https://github.com/deislabs/hippo
- https://github.com/lunatic-solutions/lunatic
- https://github.com/suborbital/atmo
- https://blog.cloudflare.com/webassembly-on-cloudflare-workers/
- https://www.fastly.com/blog/how-compute-edge-is-tackling-the-most-frustrating-aspects-of-serverless
- https://cosmwasm.com/
- https://github.com/ewasm/design

C

Wasm What?

- Can’t solve problems of tomorrow with technology of today!
- 3 Problems:
  - Dramatically distributed: DC, Client, Edge...
  - Not only build for x86 or arm -> WASM!
    - Not only build for Browsers -> WASM!
    - Not compiled for one Architecture
  - Not only build for Linux -> WASM!
    - Containers currently assume you run on linux!
- Great Performance profiles
- Great Security Profiles
  - Take untrusted code and run it safely on your system
- Cloud World:
  - We have VMs
  - We have Containers
  - We have “how do we get binary cross platform/cross architecture scaled to zero or everywhere”?
    - WASM fits that bucket!

I have my Kubernetes up and running, how can I leverage WASM for my Workloads?

Wasm will eventually need to interoperate with Docker in some way. For the next couple years this isn't strictly necessary, since Wasm will primarily be used in greenfield deployments with few requirements for backwards compatibility. But ultimately brownfield deployments need to be easy for Wasm to fully win the containerization race, especially in enterprise settings.

One potential outcome is that Docker will integrate a Wasm runtime. While plausible, I expect Wasm will be sufficiently differentiated to warrant separate tooling entirely. Instead, the unification of Docker and Wasm containers will happen at the orchestration layer.

It's less clear if Kubernetes will effectively integrate Wasm-based execution or if a new orchestration system will emerge. On one hand, Kubernetes is currently the unrivaled king of orchestration. It has incredible momentum, and the Wasm containerization movement would be wise to ride on its coattails. Folks at Microsoft are investing in that future by building Krustlet, which lets you run Wasm workloads in Kubernetes. On the other hand, Wasm code will have different requirements than Docker containers and hence Kubernetes might not be the right fit. For example, it would be useful to set up shared memory for inter-container communication when using Wasm-based third-party library isolation, which would be difficult to do with Kubernetes. Such Wasm-native orchestrators will eventually build bridges that ease migration from or integration with Docker.

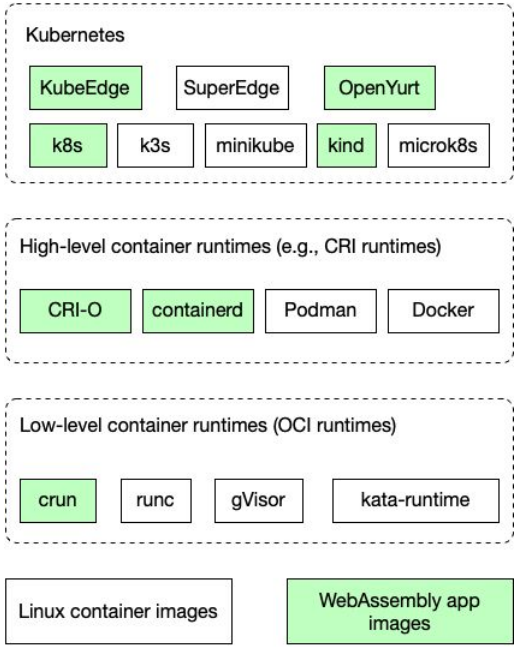
While I'm hopeful for the upcoming wave of Wasm orchestrators, Kubernetes is sufficiently entrenched that it's probably not going anywhere in the short-term.

src: https://harshal.sheth.io/2022/01/31/webassembly.html

> the unification of Docker and Wasm containers will happen at the orchestration layer. In fact, it happens now. The integration with WasmEdge and K8s toolings has been done. Developers could use crun, CRI-O, containerd, KubeEdge, KIND, OpenYurt and K8s to start, manage, and orchestrate WasmEdge Apps. See more here: https://wasmedge.org/book/en/kubernetes.html

WebAssembly will run side by side with Docker using the same orchestration tools.

src: https://wasmedge.org/book/en/kubernetes.html



The container ecosystem

Krustlet

Sources

- https://docs.krustlet.dev/topics/architecture/
- https://docs.krustlet.dev/topics/providers/
- Krustlet baked into Kind: github.com/Liquid-Reply/kind/tree/kind-krustlet

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: hello-wasm
spec:
  containers:
  - name: hello-wasm
    image: webassembly.azurecr.io/hello-wasm:v1
  tolerations:
  - effect: NoExecute
    key: kubernetes.io/arch
    operator: Equal
    value: wasm32-wasi
  - effect: NoSchedule
    key: kubernetes.io/arch
    operator: Equal
    value: wasm32-wasi
```

Conclusion

1. WebAssembly's real potential lies beyond the browser.
   - There are way more use cases than just the backend: madewithwebassembly.com gives a nice overview
2. WebAssembly can enable things that are not possible with Containers or Kubernetes; though its integration won't be as seamless.
   - This new standard is made for a new type of software.
3. WebAssembly will not substitute containers and the container ecosystem
   - Though, WASM is here to stay
4. WebAssembly standardization isn't there yet.
   - WebAssembly System Interface has numerous extensions that have not been officially standardized, but various runtimes implement a selection of these extensions.
5. The developer experience leaves much to be desired.
   - improvements in tooling (debugging, package managers, build systems, IDEs…) required
   - Lack of good components (code generators etc.) for a critical mass of languages before people actually start use WASM across different languages
   - The WASM Ecosystem is young and evolving, expect cutting edges!

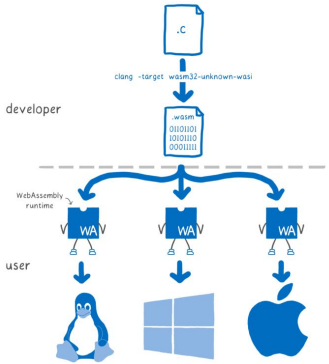
Benefits of WASM

WebAssembly excels because of the following five characteristics:

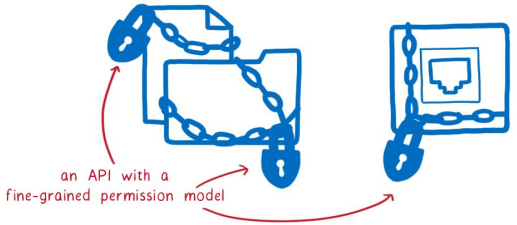
- **Portable**: The binary format for Wasm bytecode is standardized, meaning that any runtime capable of executing Wasm will be able to run any Wasm code.[1] This is similar to Java's promise of "write once, run anywhere". In the browser, 95% of users' browsers can execute WebAssembly, and the remaining gap can be bridged using a wasm2js compiler. For servers, there are runtimes like Wasmtime and Wasmer. Even resource-constrained IoT devices can join the fun using WAMR.[2]
- **Universal**: Many languages can compile into Wasm. This support goes beyond systems languages like C, C++, and Rust to include garbage-collected, high-level languages like Go, Python, and Ruby.[3] A full list of languages that compile to Wasm can be found here.
- **"Near-Native Performance"**: Wasm is often described as having "near-native performance". What this actually means is that WebAssembly is almost always faster than JavaScript, especially for compute-intensive workloads, and averages between 1.45 and 1.55 times slower than native code, but results do vary by runtime.
- **Fast Startup Time**: The cold start time of Wasm is important enough that it warrants a category of its own. On the server, it can achieve 10-100x faster cold starts than Docker containers because it does not need to create a new OS process for every container. In the browser, decoding Wasm and translating it to machine code is faster than parsing, interpreting, and optimizing JavaScript, and so Wasm code can begin executing at peak performance more quickly than JavaScript can.[4]
- **Secure**: WebAssembly was designed with the web in mind and so security was a priority. Code running in a Wasm runtime is memory sandboxed and capability constrained, meaning that it is restricted to doing what it is explicitly allowed to do.[5] While sandboxed, Wasm code can still be granted access to the underlying system, including system-level interfaces and hardware features.

The key design aspects of WASM

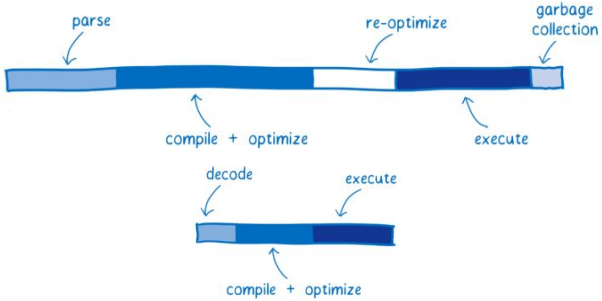
Portable



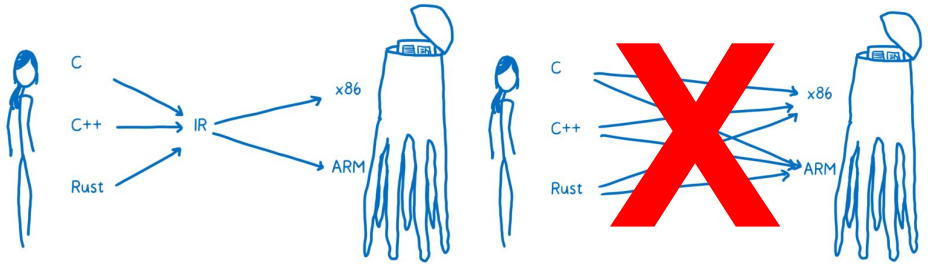
Secure



Fast



Polyglot



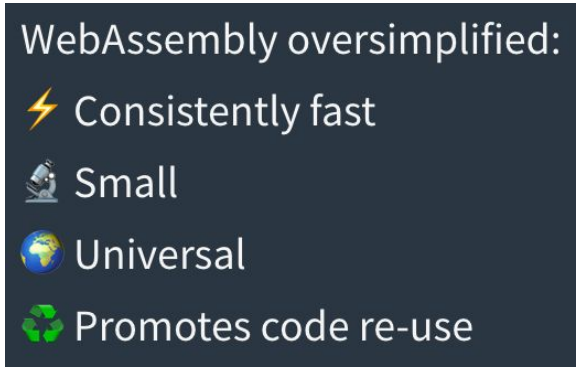
Source: https://code-cartoons.com/ by Lin Clark

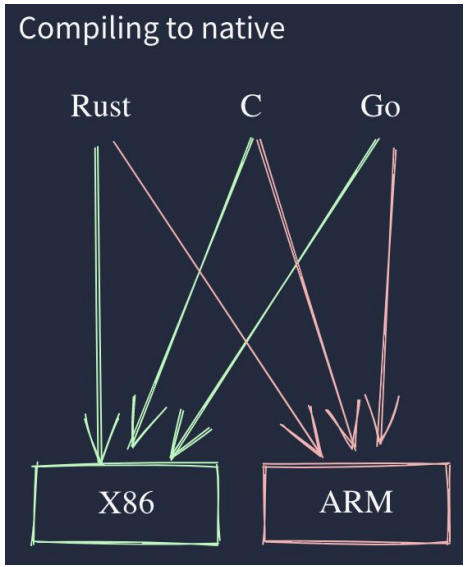
What is WebAssembly?

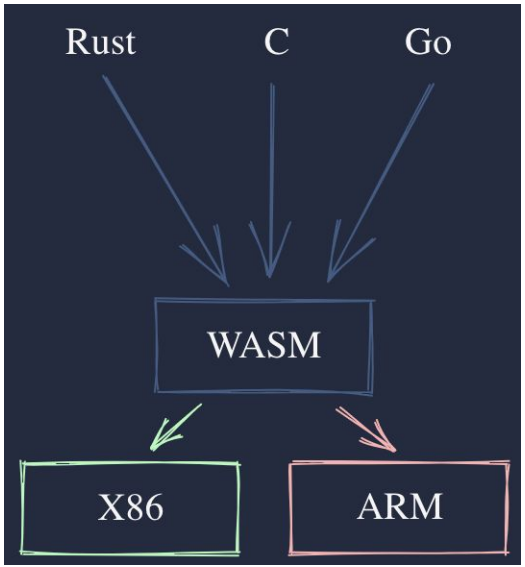
WebAssembly (abbreviated Wasm) is an intermediate layer between various programming languages and many different execution environments. You can take code written in over 30 different languages and compile it into a .wasm file, and then can execute that file in the browser, on a server, or even on a car.

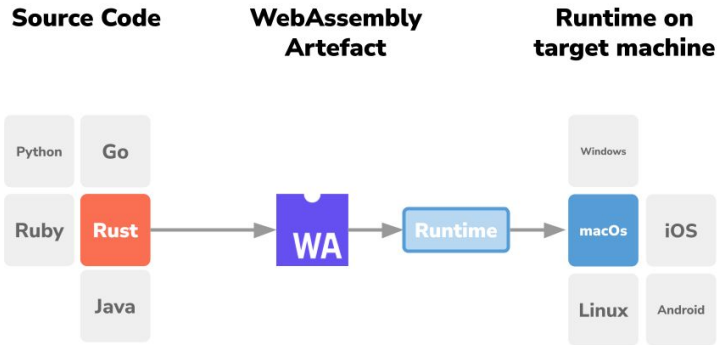
The name “WebAssembly” is misleading. While it was initially designed to make code run fast on the web, it now can run in a variety of environments outside of the browser as well. Moreover, WebAssembly is not assembly but rather a slightly higher-level bytecode.

We wont explain WebAssemblys History here, which is super interesting, but not the focus of this talk. We’d like to understand the advantages that it has now or in the future, not its entire genesis.

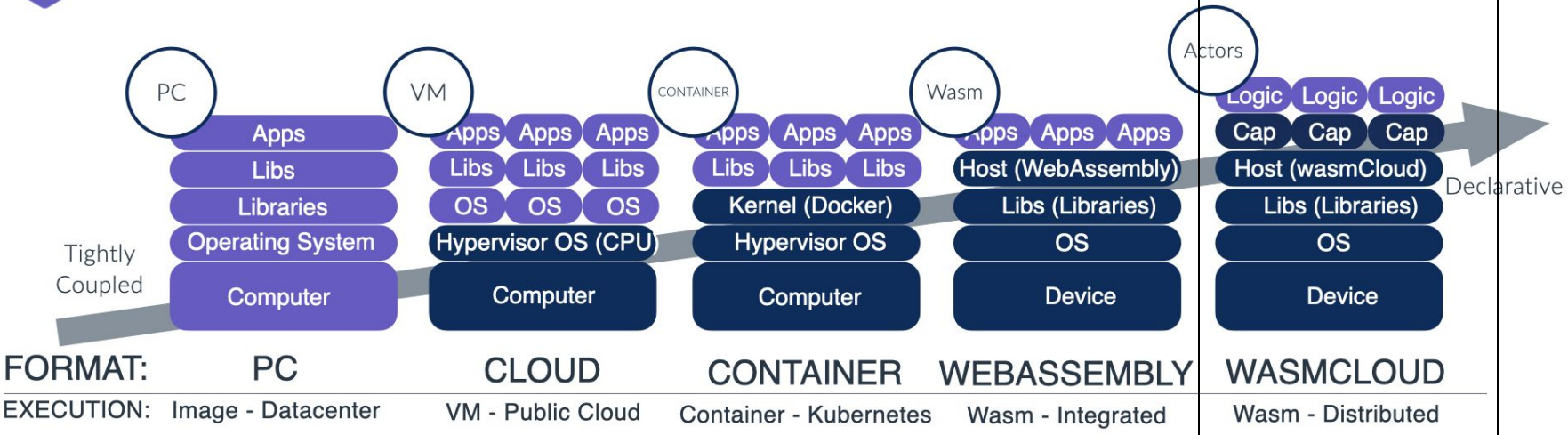



Compiling to native
Rust
C
Go
X86
ARM


Rust
C
Go
WASM
X86
ARM

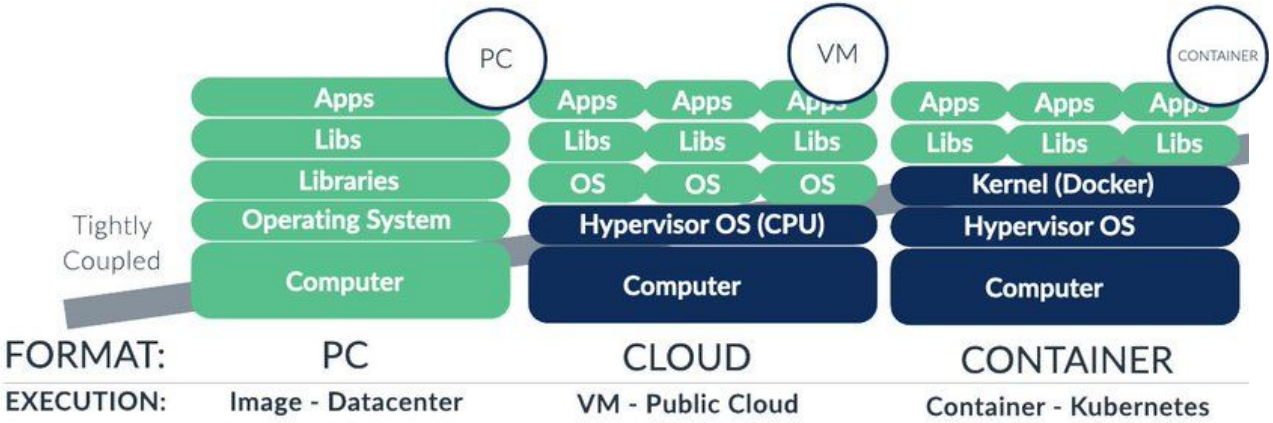

Source Code
WebAssembly Artefact
Runtime on target machine
Python
Go
Ruby
Rust
Java
WA
Runtime
Windows
macOs
iOS
Linux
Android

cosmonic



FORMAT:	PC	CLOUD	CONTAINER	WEBASSEMBLY	WASMCLOUD
EXECUTION:	Image - Datacenter	VM - Public Cloud	Container - Kubernetes	Wasm - Integrated	Wasm - Distributed

A new paradigm ahead?



Source: Liam Randall (@Hectaman) - https://twitter.com/Hectaman/status/1389585069073895434

Benefits of WASM