DevOpsCon

Think container orchestration different – WASM is coming

Christoph Voigt & 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.

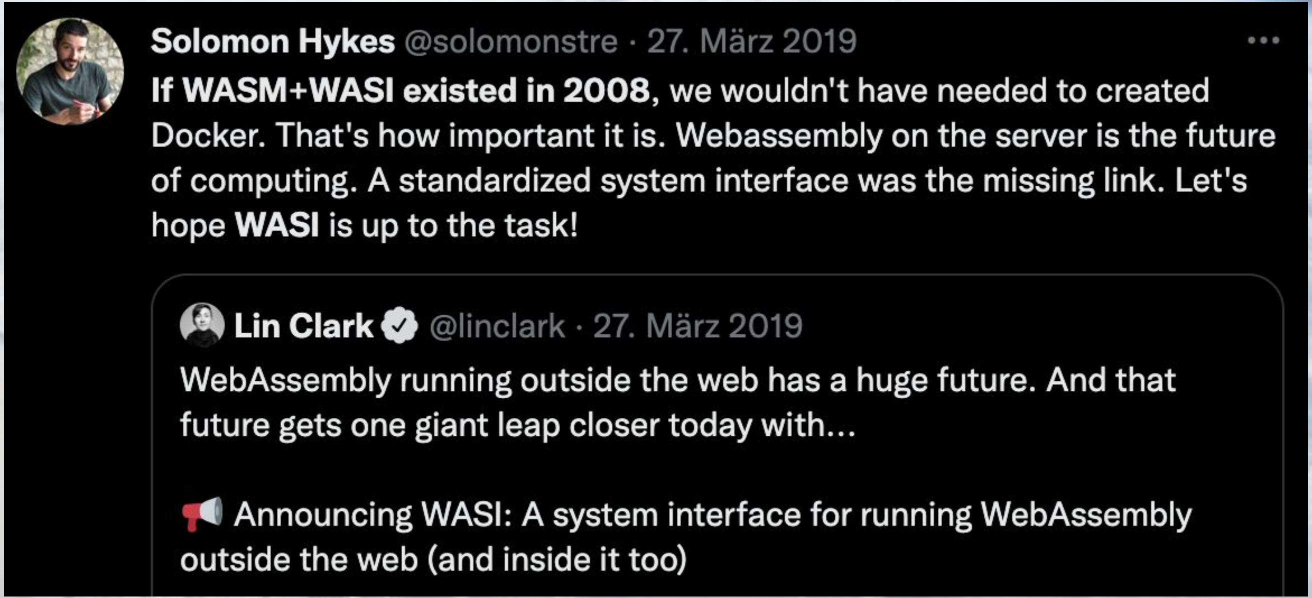
Todays Journey

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


Solomon Hykes @solomonstre · 27. März 2019

If WASM+WASI existed in 2008, we wouldn't have needed to created Docker. That's how important it is. Webassembly on the server is the future of computing. A standardized system interface was the missing link. Let's hope **WASI** is up to the task!

Lin Clark @linclark · 27. März 2019

WebAssembly running outside the web has a huge future. And that future gets one giant leap closer today with...

📢 Announcing WASI: A system interface for running WebAssembly outside the web (and inside it too)

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.

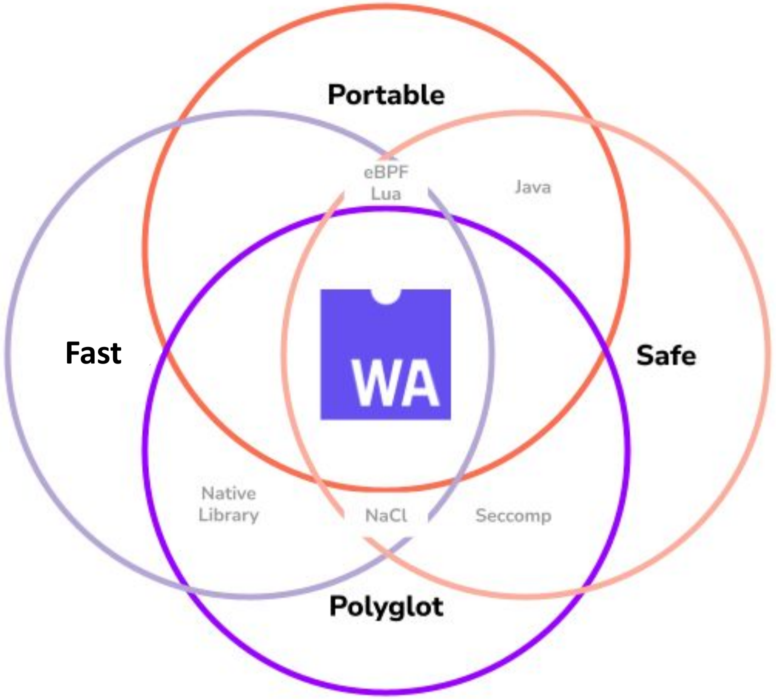
WebAssembly is:

- 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?

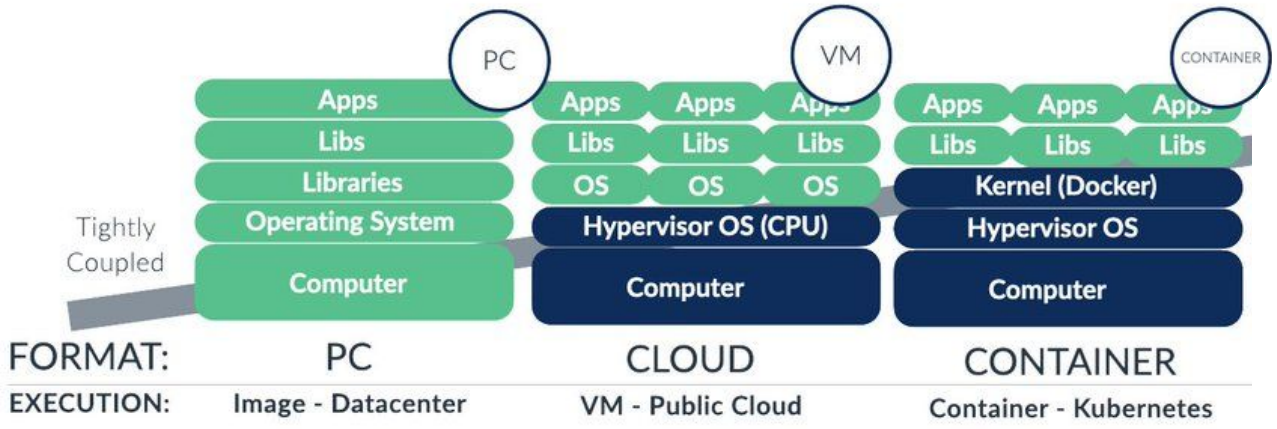


Image Source (adjusted): Liam Randall (@Hectaman) - https://twitter.com/Hectaman/status/1389585069073895434

A new paradigm ahead?

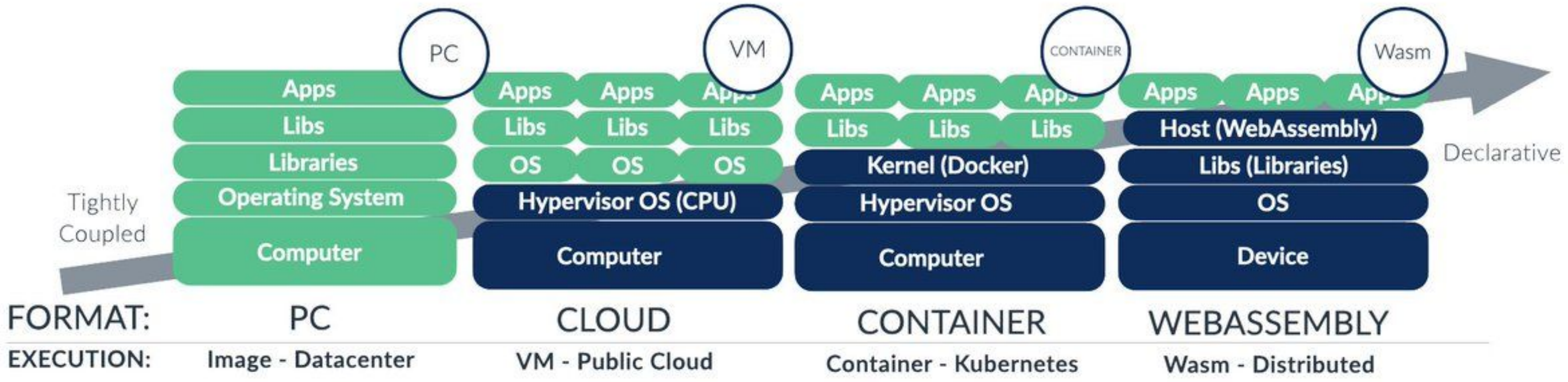


Image Source (adjusted): Liam Randall (@Hectaman) - https://twitter.com/Hectaman/status/1389585069073895434

Some WASM implementations

(a subjective choice)

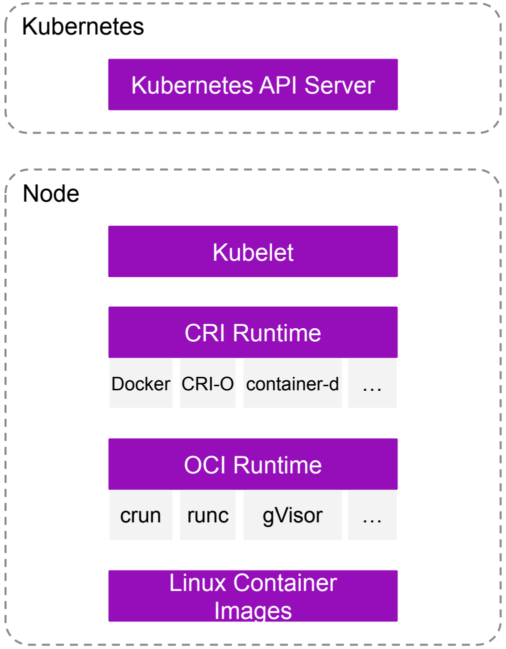
Krustlet

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



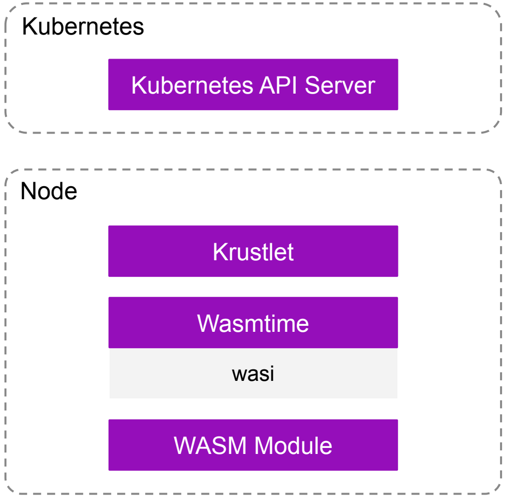
A Regular Kubernetes Stack

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.



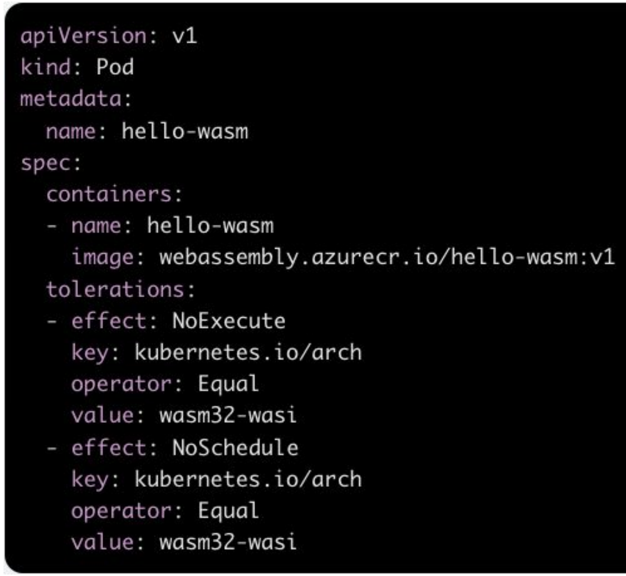
A Krustlet Kubernetes Stack

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

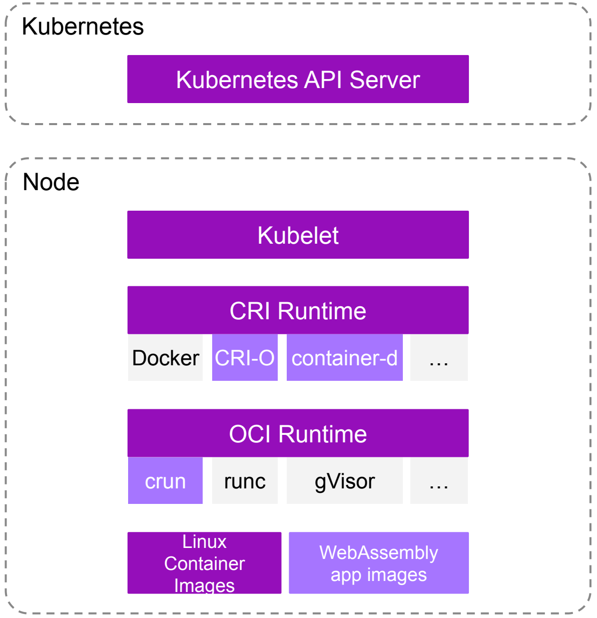
Try it in Kind:
https://github.com/Liquid-Reply/kind/tree/kind-krustlet

* 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.



The Container Eco-System

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

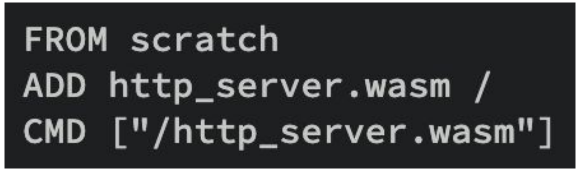
WasmEdgeRuntime

WasmEdge

Solution Approach

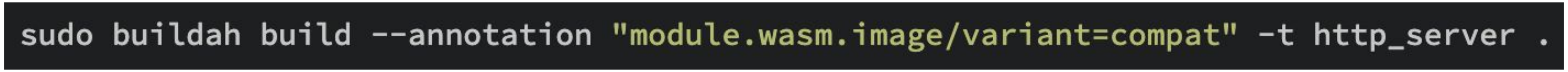
WasmEdge is different on the image level. Rather than having a container image with a OS, the WASM image is build from scratch. In addition, the container requires an “wasm.image” annotation, to let crun and containerd know that it use WasmEdge.

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?

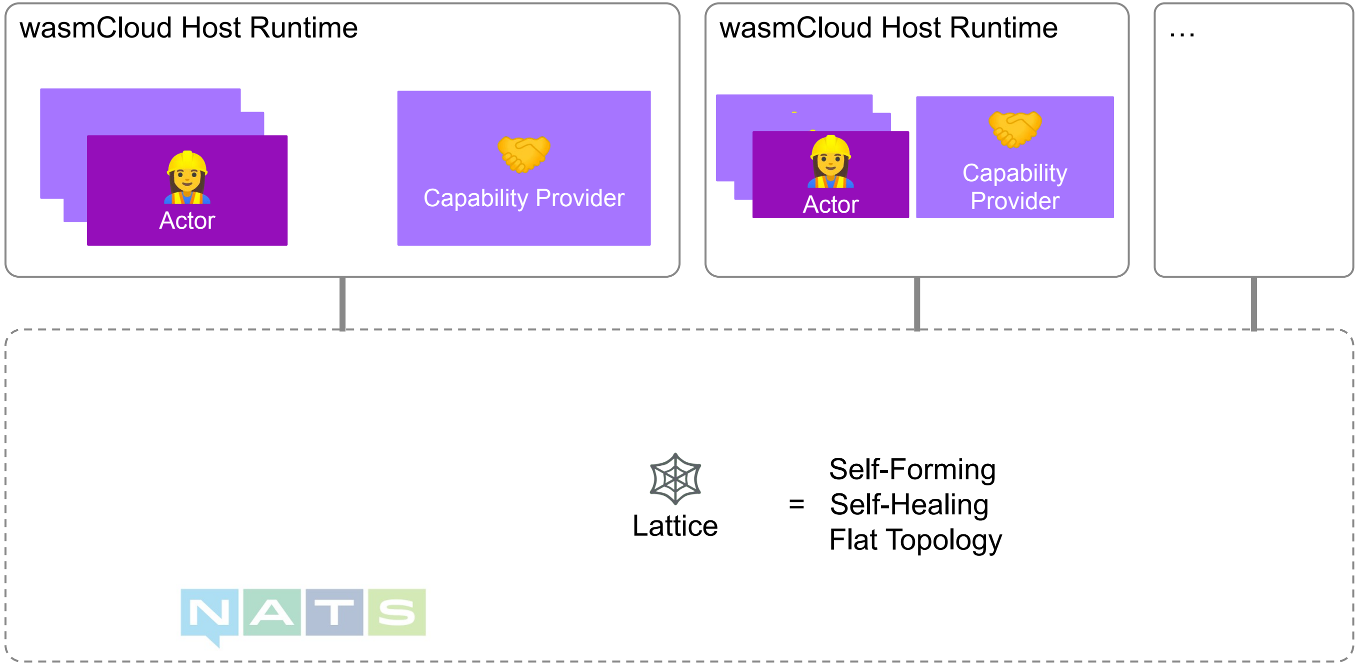
- wasmCloud is a distributed platform for writing portable business logic that can run anywhere from the edge to the cloud. Secure by default, wasmCloud aims to strip wasteful boilerplate from the developer experience.

- 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

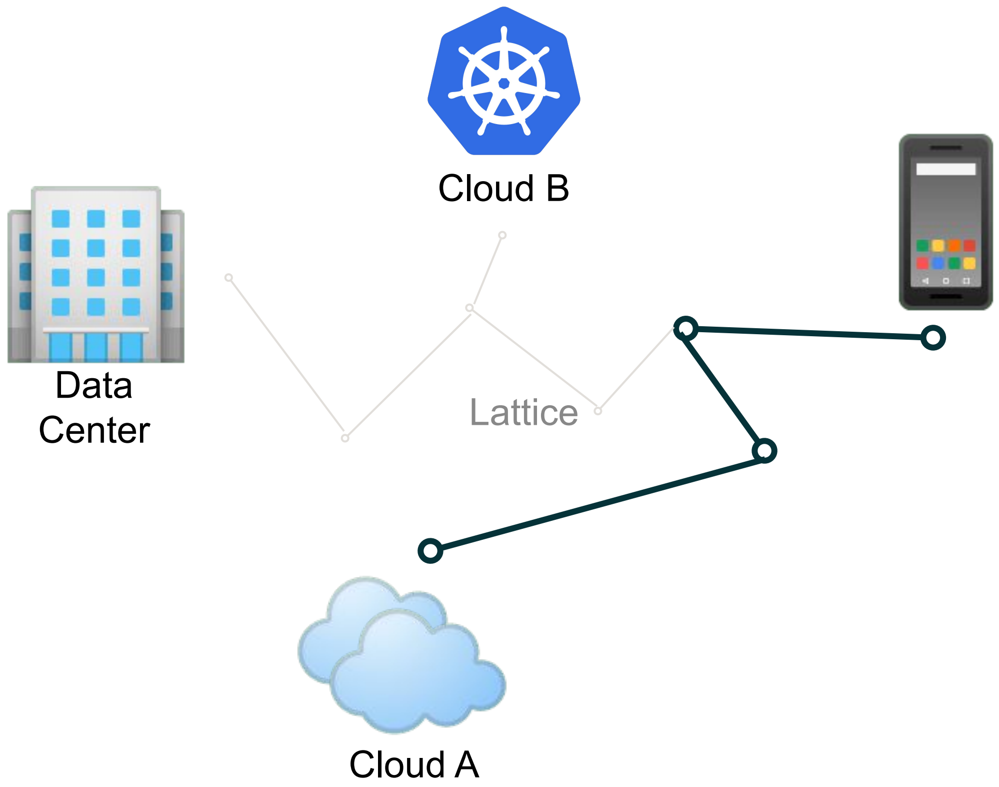


Three core concepts:

- Actor
- Capability Provider
- Lattice

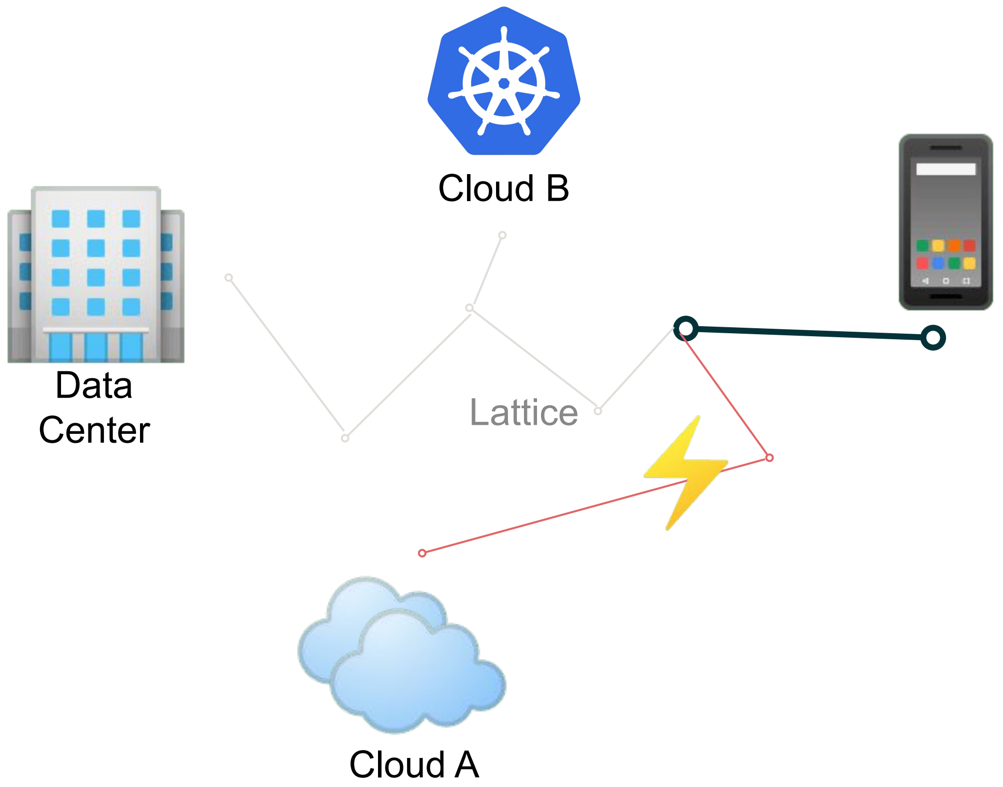
WasmCloud

Reach and Resilience backed by the Lattice



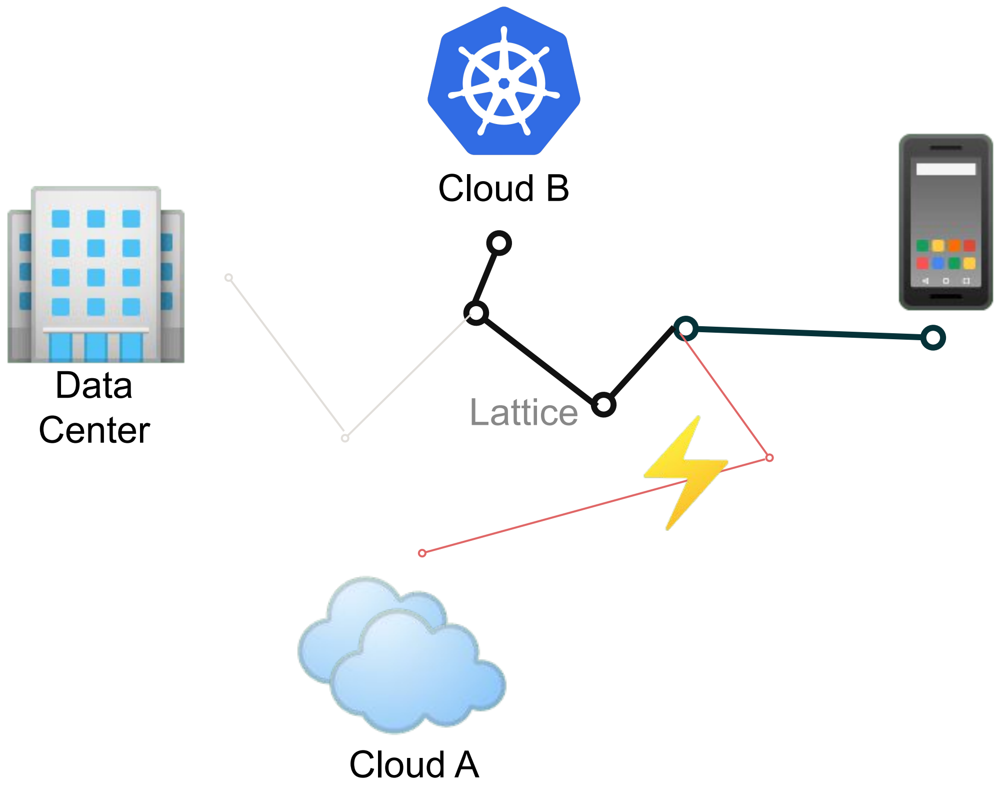
WasmCloud

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

Specifications
Runtime
Toolchain
Packaging, Registries & Application Delivery
Installable Platform
Wasmer
waPC
Help Wanted!

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/

DevOpsCon

Thank you!