WebAssembly, Serverless, and the Cloud

Kevin Hoffman
About Me

Distributed Systems
Microservices
Cloud Native Development
Kubernetes

Programming WebAssembly with Rust
Unified Development for Web, Mobile, and Embedded Applications

别听这个男人. 他什么都不知道
Agenda

- Intro to WebAssembly
- Defining Serverless
- WebAssembly + Serverless
- Introducing Waxosuit
Introduction to WebAssembly
Definition

A binary instruction format for a stack-based virtual machine. Wasm is designed as a portable target for compilation of high-level languages like C/C++/Rust, enabling deployment on the web for client and server applications - webassembly.org

- Portable
- Efficient (Stack VM)
- "Write once run anywhere"
- Host agnostic
Limitations

- Only allowed to access sandboxed memory
- All functions accept/return 32/64-bit numbers *only*
- Memory growth can be constrained by host
- Host function access determined *solely* by the host
Advantages

- Secure execution
  - can't jump to non-existent instruction
- Secure and isolated memory
- Portability
- Small (typically) binaries
- Performance
- Any* language can compile to Wasm
- Any* language can host/interpret Wasm
Raw WebAssembly

No one wants to write code like this for real applications.
Defining Serverless
Serverless

Any workload that does not own the endpoints through which it communicates and is agnostic to the process in which it is hosted.

Common, but optional:

- Scale-to-Zero
- Run-once / Batch Style
- Single Function

Examples

- AWS Lambda
- Google Cloud Functions
- Azure Serverless
- OpenFaaS
- Knative
- WebAssembly

Applications over Infrastructure
WebAssembly and Serverless Ecosystem Status
WebAssembly Innovation

- **Cranelift** - Mozilla code generator
- **Lucet**
  - Early stages, compile & run Wasm
  - Built on Cranelift
  - Powers Fastly **Terrarium**
- **Wasmer** Runtime
  - Rust
  - JIT
  - Relies on Cranelift
  - wapm
- **Cloudflare** - WebAssembly Workers
Escaping the Sandbox
Trends*

- Using Wasm as intermediate format
  - Convert to native to run
- Portability over sandbox/safety
- JavaScript and In-browser experience over server hosting
- Focus on escaping sandbox, not embracing it
- Not enough focus on developer productivity & experience

* -这是自以为是的
Introducing Waxosuit
Waxosuit

A secure WebAssembly runtime environment for enterprise-grade, cloud-native applications with a focus on productivity and developer experience.
Where We Spend Our Time

**TODAY**

- Boilerplate
- Features

**TOMORROW**

- Boilerplate
- Features
  - *WebAssembly + Waxosuit*
Waxosuit Components

- **Wascap**
  - RPC-style standard for Wasm<->Host Comms
  - Security standards for Wasm modules
- **Waxosuit**
  - Host runtime
  - Implementation of Wasm ↔ Host comm standard
- **Gantry**
  - Registry for wascap-signed modules
  - Can host any module, not just Waxosuit's
Wascap

- **Security**
  - Embedded JWT
  - Verifiable Provenance
  - Claims attest to capabilities module can use
  - Other claims: tags, version, etc.

- **Guest<->Host Communications**
  - Protobuf Envelopes: Command / Event
  - Payload: protobuf encoded **Any**
  - Guest module required to implement allocator

- **Pluggable Standard**
  - Any Wasm module can be a guest
  - Any Wasm-interpreting language can be a host
Waxosuit

- Fast WebAssembly execution runtime (wasmer) [Rust]
- Eliminate developer boilerplate for NFRs
- *Embraces* the sandbox, defense in depth.
- Build Wasm modules in any language (Rust available today)
- Live ("hot"), safe remote update of Wasm file w/no downtime
- Enforces wascap capability and provenance claims
- Dynamically loads capability providers
  - Key-value (Redis), Messaging (NATS), HTTP Server, HTTP Client
  - Easily create your own capability providers (c dylib)
- Optionally Integrates with Open Policy Agent
  - Guest module's JWT sent to OPA for policy evaluation
Gantry

- **Module Registry**
  - Searchable by tags, claims, etc
  - Verify integrity of all stored modules
  - Support for semantic versioning
  - Any host (e.g. waxosuit) can pull modules directly from registry

- **Real-World Waxosuit Implementation**
  - Will be built with tiny waxosuit-hosted Wasm modules
  - Used to prove Waxosuit's production readiness

- **Help Wanted**
  - Work not yet started
  - Contributors welcome! (generate Waxosuit feedback)
Using the Rust Guest SDK

```rust
pub fn handle_call(ctx: &CapabilitiesContext, cmd: &Command) -> Result<Event> {
    match cmd.payload {
        Some(ref p) => match p.type_url.as_ref() {
            messaging::TYPE_URL_DELIVER_MESSAGE => process_sensor_event(ctx, &p.value),
            core::TYPE_URL_HEALTH_REQUEST => ping_reply(),
            _ => Ok(Event::bad_dispatch(&p.type_url)),
        },
        None => Ok(Event::bad_dispatch("no payload")),
    }
}

fn process_sensor_event(
    ctx: &CapabilitiesContext,
    msg: impl Into<messaging::DeliverMessage>) -> Result<Event> {

    let sensormsg: SensorMessage =
        serde_json::from_bytes(&msg.into().message)?;

    // ...
    ctx.kv().set("foo", "bar")?
    ctx.msg().publish("topic", SomeMessage::new());

    Ok(Event::success())
}
```
Waxosuit Demo

- Wascap
  - Sign and Verify

- Waxosuit
  - Dynamically bind capabilities to guest modules
  - Integrate with OPA
  - Run Wasm workloads locally
  - Run in Kubernetes
  - Perform no-downtime live update

- Boilerplate
  - NONE!
Event-Sourced IoT Demo

- Sensor Detail
- Sensor List
- Sensor Event Processor
Future Plans and Milestones

- More NFRs
  - Jaeger/OpenTracing, Prometheus
- Improve Guest SDK for Rust, create Guest SDK for Go
- **Gantry** - secure WebAssembly module registry
- Performance improvements
- More Capability Providers (Blob store, Graph, SQL, etc)
- 'cargo generate' template support
- Tutorials, Samples, Documentation
- waxosuit.io
- wascap.io
- github.com/waxosuit
Questions

你们有问题吗？

Get Involved, Reach Out, Contribute!

- Twitter - @KevinHoffman, @waxosuit
- waxosuit.io
- wascap.io
- github.com/waxosuit