From Napkin to the Cloud: A WebAssembly Journey

Who Am I?

Kevin Hoffman

Creator and Lead: **wasmCloud**

Author,

“Programming WebAssembly with Rust”

Twitter:

@KevinHoffman

AGENDA

WHAT IS WEBASSEMBLY + WASMCLOUD?

The two minute tour

THE NAPKIN

Discuss the application we want to build and its conceptual design

THE BUILD

Examine the real code and the real system. How close is it to the napkin drawing?

THE DEMO

Demo of the application and its WebAssembly actors

Q&A

Live Question and Answer Period

1

WEBASSEMBLY & THE CLOUD

The Quick Tour

WHAT WE WANT IN CLOUD NATIVE WORKLOADS

- Security
- Predictability
- Speed
- Size (lack thereof)

WHAT WE GET FROM WEBASSEMBLY

- Security
- Predictability
- Speed
- Size (lack thereof)
- Portability
- Freedom from Dependency Hell
- “Beneficial Limitations”

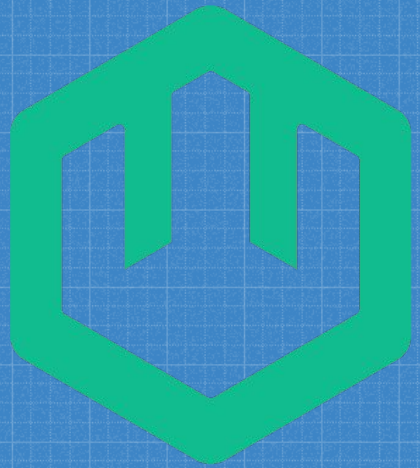
WEBASSEMBLY IS...

- Stack-based VM
- Binary file format (bytecode)
  - Text representation (wast)
- CPU-agnostic
- OS-agnostic
- Fast, small, efficient
- Entirely Dependent on the Host Runtime
  - Web browser (JavaScript)
  - Wasmtime, wasmer, wasm3, **wasmCloud**
- *The future of distributed computing*

WASMCLOUD IS...

- Actor Runtime and Platform
- Portable business logic runs anywhere (edge, cloud, IoT, browser)
- Secure-by-default
- Eliminates boilerplate
- Rapid Feedback loop
- Self-healing, self-forming *lattice* mesh that can connect clouds, IoT devices, edges, and more

WebAssembly AND wasmCloud

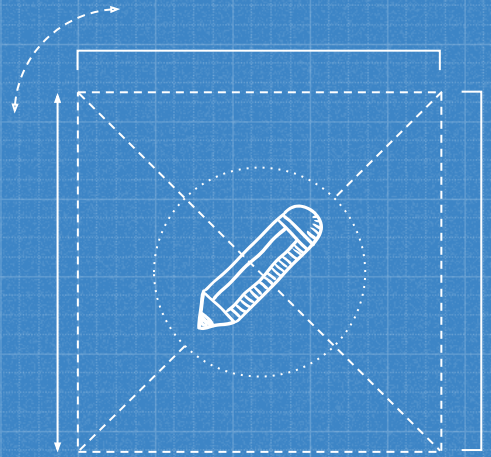


	WEBASSEMBLY	WASMCLOUD
SECURITY	✔	✔
PORTABILITY	✔	✔
SPEED	✔	✔
SMALL SIZE	✔	✔
LOOSELY COUPLED, SECURE CAPABILITIES		✔
SEAMLESS DISTRIBUTED COMPUTE (RUN ANYWHERE)		✔
HORIZONTAL SCALING		✔
ACTOR MODEL		✔
CONTRACT-DRIVEN DESIGN		✔

2

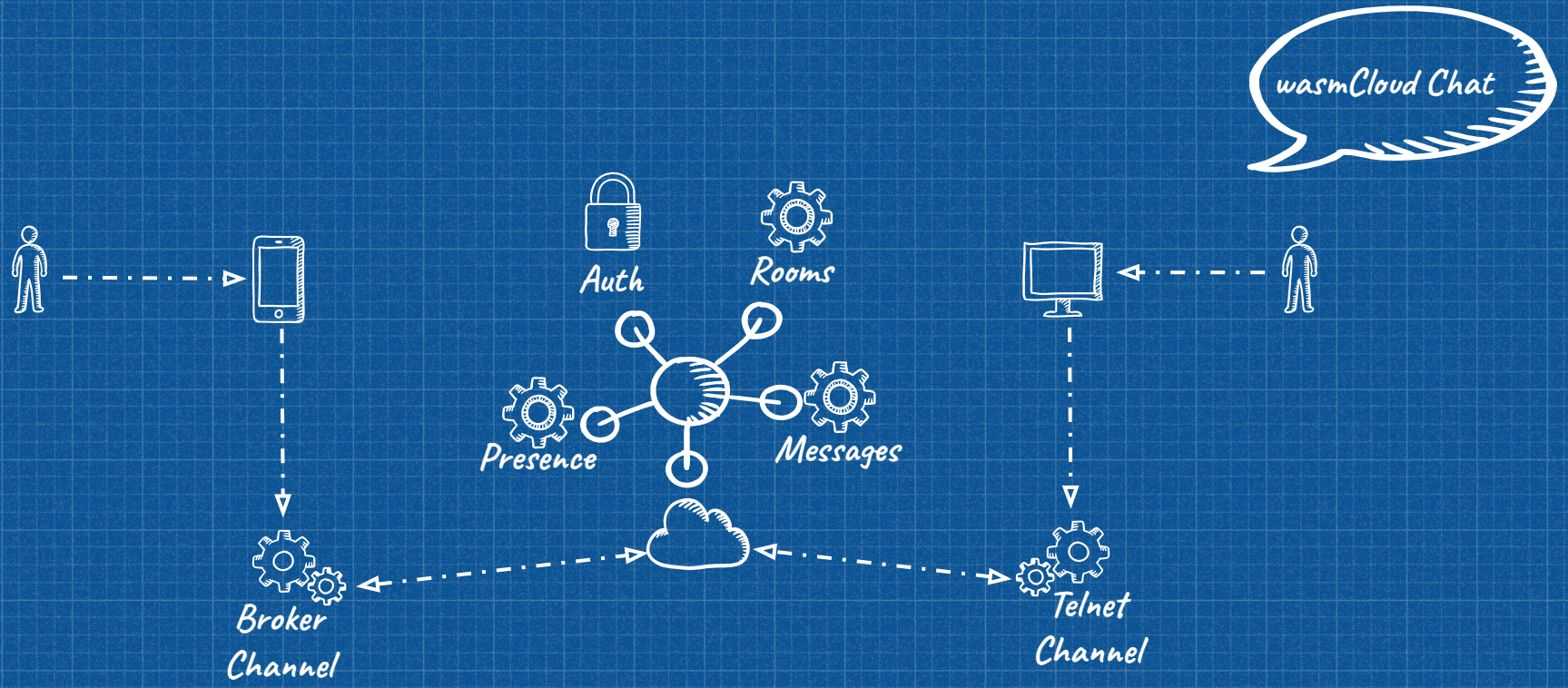
THE IDEA

The Journey of a Thousand Cloud-Native Apps Begins with a Single Idea



wasmCloud Chat

Multi-Channel Real-Time Messaging Application



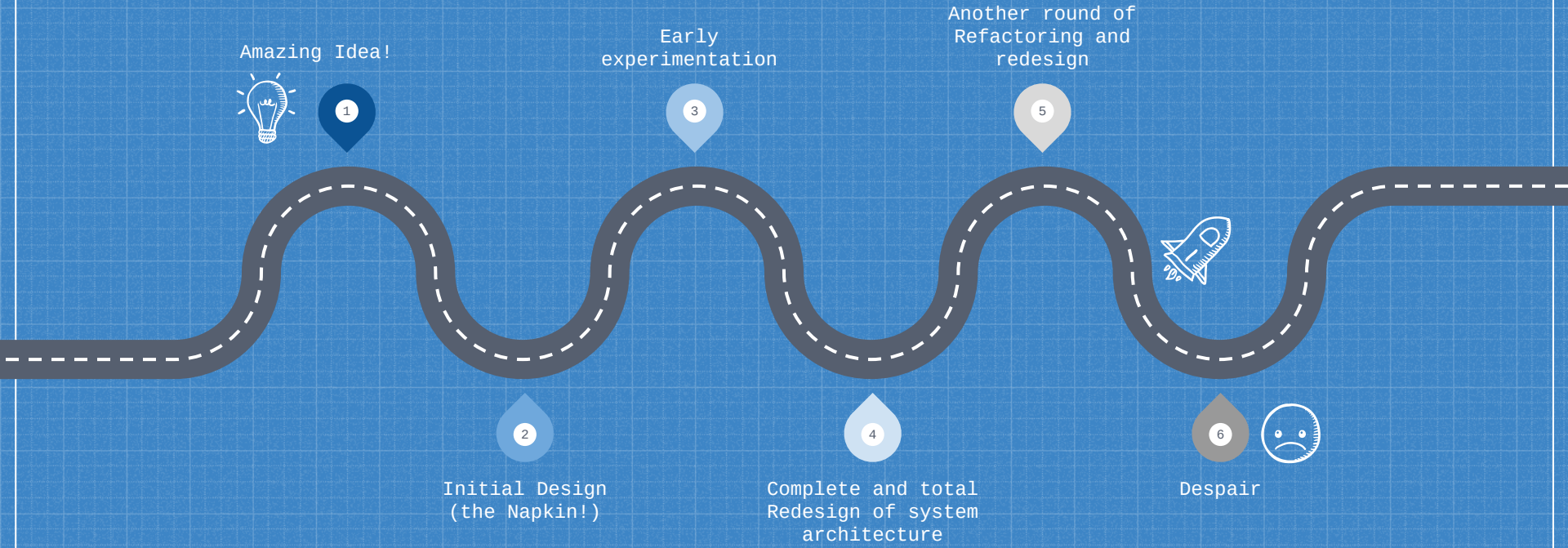
Initial Application Design

The "Napkin"

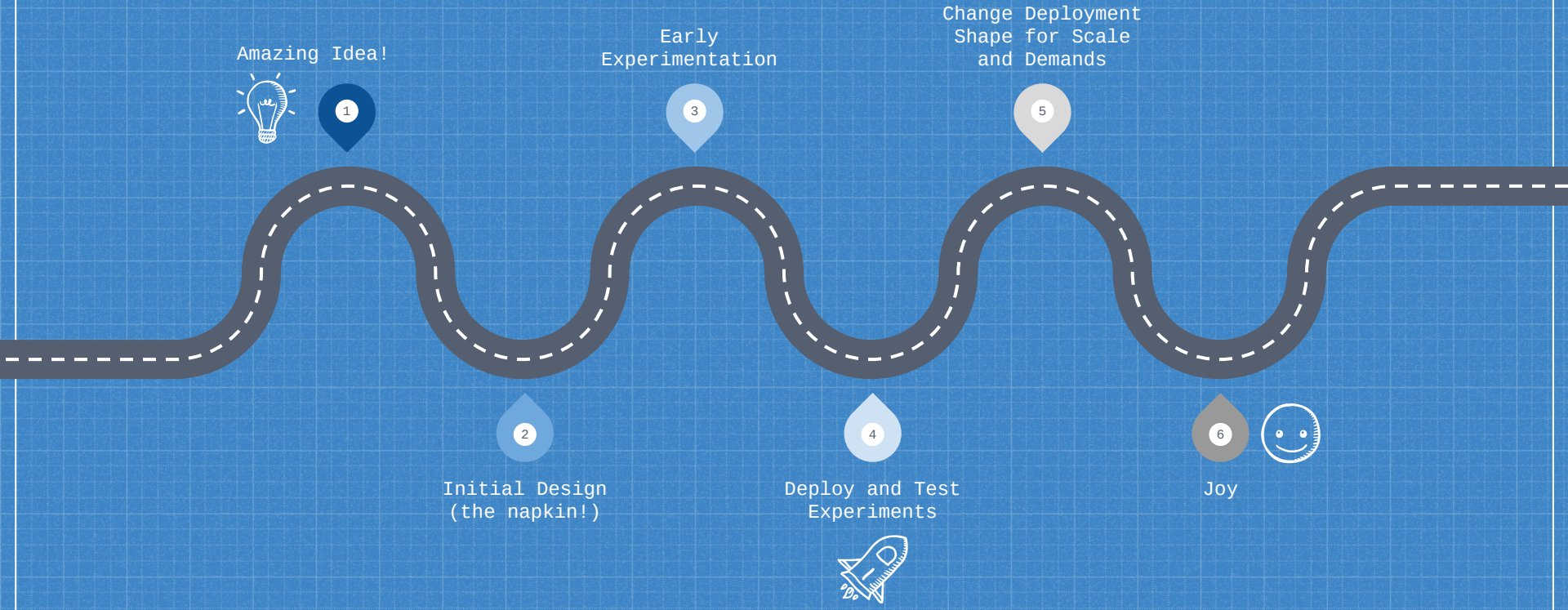
The Best Laid Plans of Mice and Men Often Go Awry

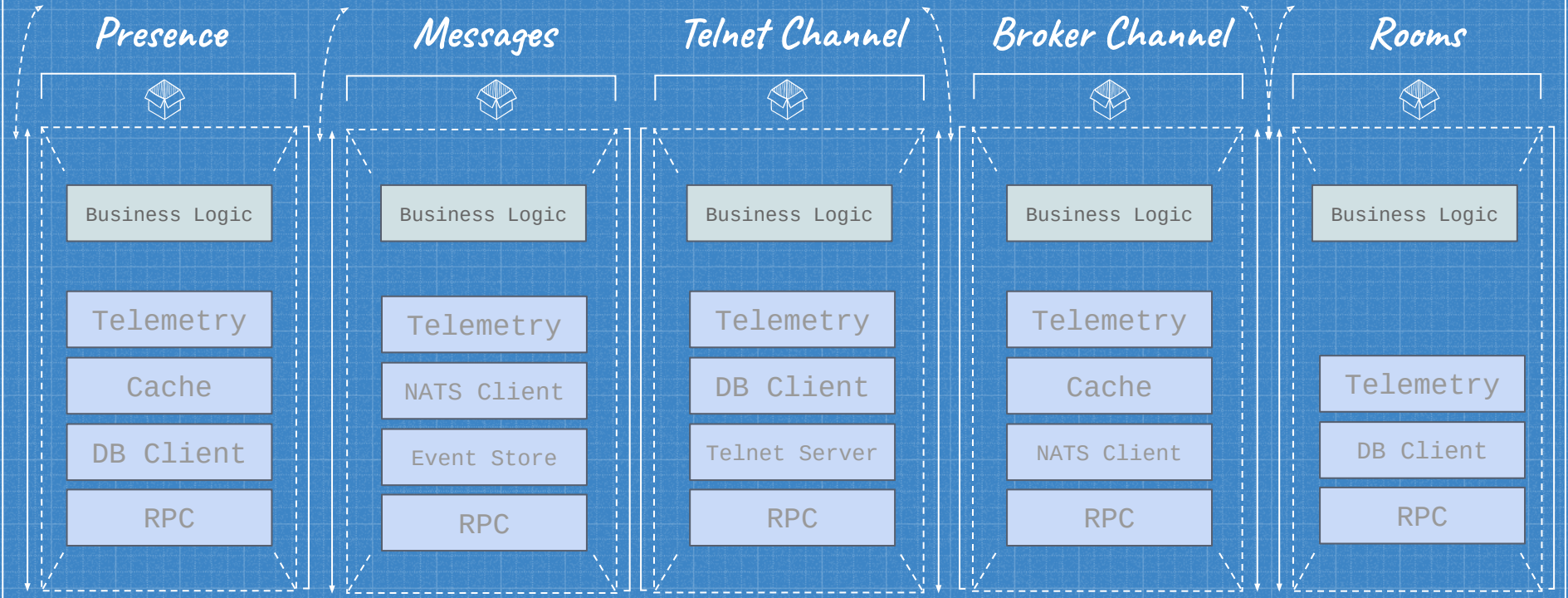
- Robert Burns

THE JOURNEY (today)

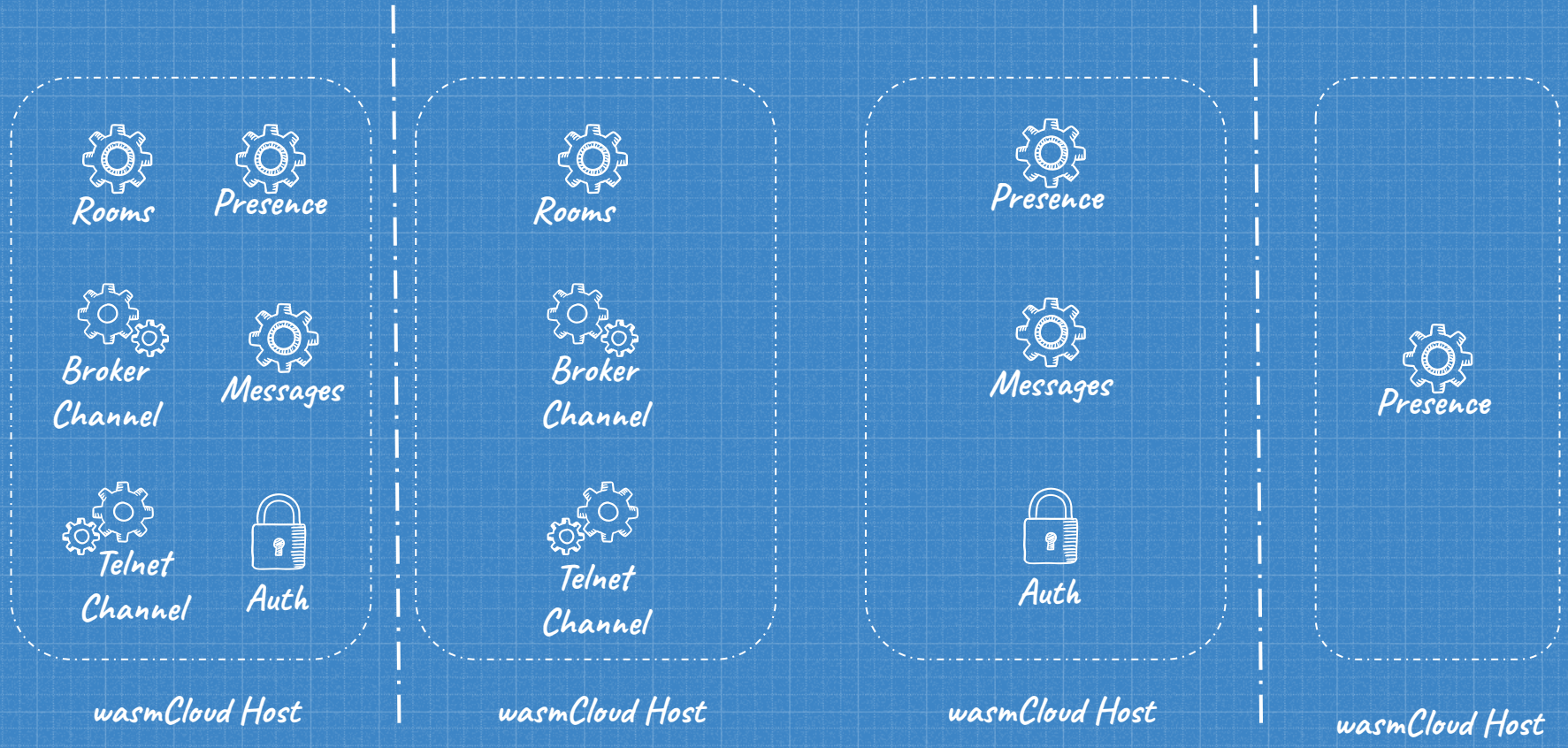


THE JOURNEY (wasm+cloud)





"Traditional" Microservices - Hard Service Boundaries



wasmCloud Actors - Flexible Service Boundaries

3

THE BUILD

wasmCloud Chat Actors

THE BASIC ACTOR - REGISTER MESSAGE HANDLERS

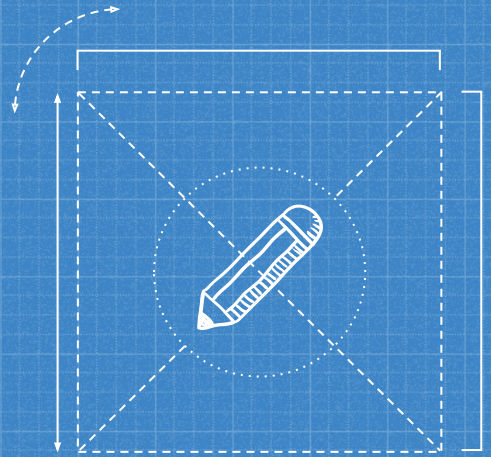
```
#[actor::init]
fn init() {
    messages::Handlers::
        register_process_message(process_message);

    logging::enable_macros();
}
```

WHAT'S MISSING FROM THIS CODE?

```
fn publish_broker_message(
    target: &MessageTarget,
    message: &messages::ChannelMessage,
) -> HandlerResult<()> {
  info!("Publishing message to back-end");
  let topic = match target {
    MessageTarget::User(s) => format!("{}.user.{}", BACKEND_SUBJECT_PREFIX, s),
    MessageTarget::Room(s) => format!("{}.room.{}", BACKEND_SUBJECT_PREFIX, s),
    _ => "".to_string(),
  };
  // . . .
  broker::host(LINK_BACKEND).publish(topic, "".to_string(),
      serde_json::to_vec(&ce)?)?;
  Ok(())
}
```

ACTORS EXPOSING DATA SERVICES

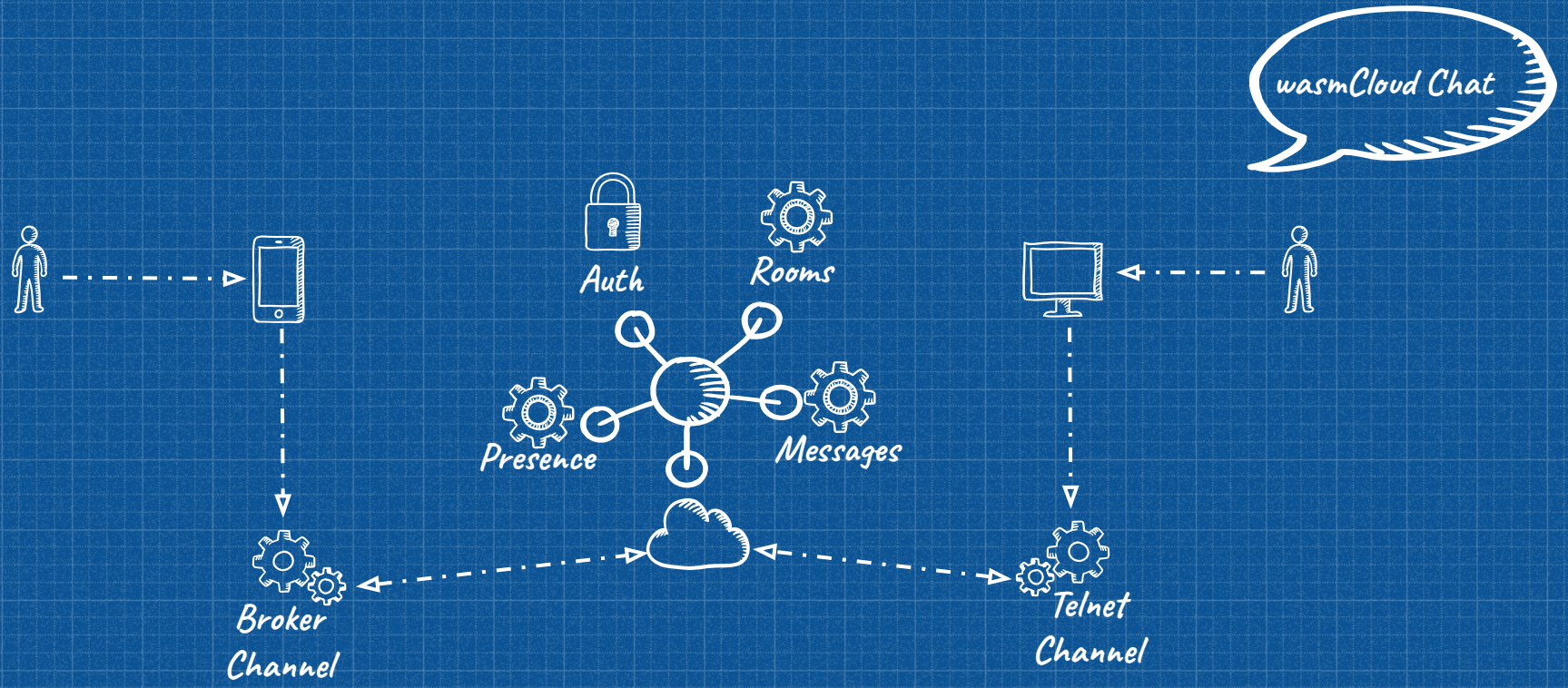
```
fn create_room(id: String, description: String) -> HandlerResult<RoomAck> {
  info!("Creating room {}", id);
  Ok(add_room(id, description).map_or_else(
    |e| RoomAck::fail(&format!("{}", e)),
    |_v| RoomAck::success(),
  ))
}
pub(crate) fn add_room(id: String, description: String) -> HandlerResult<()> {
  let key = format!("rooms:{}", id);
  let val = serde_json::to_string(&RoomData {
    id: id.to_string(), description})?;
  let _ = keyvalue::default().set(key.to_string(), val, 0)?;
  let _ = keyvalue::default().set_add("rooms".to_string(), id)?;
  Ok(())
}
```



DEMO

wasmCloud Chat in Action

https://github.com/wasmCloud/examples/tree/main/wasmcloud-chat



Initial Application Design

The "Napkin"

Thanks!

ANY QUESTIONS?

Additional Resources:

Twitter - **@wasmcloud**,
@KevinHoffman

https://wasmcloud.dev

https://github.com/wasmcloud