# Holochain

Sensorica©TM

# Holochain

Go to the Sensorica Technical Education for more learning experiences

This is a co-creation document, make sure you respect the Content rules

## Table of contents



# Essential Reading

Holochain builds on a wealth of research and knowledge about distributed systems. Here are some carefully picked resources to help you on your journey:

- Distributed Systems - A free online class
- Keeping CALM: When Distributed Consistency is Easy, a paper by Joseph M Hellerstein and Peter Alvaro that lays down the mathematical foundation for distributed systems that don't need coordination protocols, such as Holochain.
- CRDT.tech, a resource for learning about conflict-free replicated data types (CRDTs), data structures that let multiple users make concurrent updates to a resource with little to no manual conflict resolution.
- Yjs and Automerge, two CRDTs that can operate on arbitrary JSON. Syn (mentioned here : Libraries) uses Automerge.

- Local-first software: You own your data, in spite of the cloud, a paper by Martin Kleppmann et al of Ink & Switch that explores the user experience of local-first software built on CRDTs.

# Tools

- VSCodium - code editor
  - Rust analyzer
  - Crates
  - Even Better TOML
  - Rust Test Explorer
  - Nix IDE
  - Path Intellisense
- **Cursor - AI integrated**
  - **Cursor Docs**
  - **Cursor Directory**
  - **SpecKit** - specification driven development (SDD)
    - **Install UV: terminal command** *curl -LsSf https://astral.sh/uv/install.sh | sh*
    - **Install SpecKit: terminal command** *uv tool install specify-cli --from git+https://github.com/github/spec-kit.git*
    - **Initialize: Open Cursor, open Terminal, enter command** specify init . --ai cursor-agent
      - You must do this in every project.
- Rust - language
- Holochain development environment
  - The most important line from that doc is here.
  - That will automatically install the NixOS development environment
- NixOS
  - Official website
  - Nix flakes (Nix feature used by Holochain)
    - It's not necessary to configure it, Holochain development environment installation do it for us)
    - It's useful to understand it if more advanced customizations of the development environment are needed.
  - Ultimate Nix Flakes Guide (Youtube video)
  - NixOS: Everything Everywhere All At Once (Youtube video)
  - Nix IDE (VSCode extension)
- hREA
- app.diagrams.net create graphical representations, architectural design. Recommendations: use UML and C4. NOTE: use C4 for the high level abstractions (context, containers and optionally components) and use UML to represent the code and optionally components(layer of C4).

- AI
    - Claude for coding and Gemini for planning, documentation, analyzing.

# Holochain development environment

**Install holonix development environment**. On Linux run the following command in terminal (see reference)

```
bash <(curl https://holochain.github.io/holochain/setup.sh)
```

This command downloads the setup script and runs it, installing the Nix package manager and setting up a package cache for Holochain.

**Set up Nix development environment for Holochain**

```
nix run --refresh -j0 -v "github:holochain/holonix?ref=main-0.6#hc-scaffold" -- --version
```

At the end of the output, Holochain's scaffolding tool should print its version string, which will be something like this:

```
holochain_scaffolding_cli 0.600.0
```

To **scaffold** a new Holochain hApp (info here). For this tutorial, we'll be working in ~/Holochain. Create that folder now and move into it.

```
mkdir Holochain
```

Navigate in that folder

```
cd ~/Holochain
```

When getting started, seeing a simple but fully-functional app can be very helpful. You can have Holochain's scaffolding tool generate a "Hello World!" application (but for a distributed multi-agent world) by typing the following in your command line terminal:

```
nix run "github:/holochain/holonix?ref=main-0.6#hc-scaffold" -- web-app
```

The scaffolding tool will ask you one question — what JavaScript package manager you'd like to use in your project. If in doubt, just choose **npm**.
So this creates a folder called **hello-world** After doing a bit of work, it'll print out these four commands for you to run in order to compile and run the app. Copy them from your terminal or from below:

cd hello-world

nix develop

Nix will then download all the packages you need to build and test Holochain apps. It might take a few minutes the first time. This is a very slow process !! It can take up to 30 min or more.

Then

if you use npm you run

npm install

If you use bash you run

bun install

Then

if you use npm you run

npm start

If you use bash you run

bun run start

There are a lot of configurations to do, I chose **svelt**, name for the **happ**,

Open your IDE (VSCodium or Cursor for example) and open the folder that you have created

Run

bash <(curl https://holochain.github.io/holochain/setup.sh)

Your environment is ready to start building your hApp.

You can use Holochain's scaffolding tool to start coding. For example, you can open the terminal in VSCodium and use this command:

hc scaffold entry-type

There will be a series of questions to answer, the scaffolding tool will create the code for you.

# Rapid Application Tool for Holochain (Scaffolding tool)

Using the terminal, you are able to generate holochain code for basic components, by answering questions.

Official documentation

# Basic learning

- Official documentation
- Various resources
- Glossary

A multicellular social organism with a memory and an immune system

> What is **agent-centric computing?** Equipped with the Holochain runtime, each user runs their own copy of the back-end code, controls their identity, and stores their own private and public data. An encrypted peer-to-peer network for each app means that users can find each other and communicate information among each other.
>
> The **two pillars of application integrity** are intrinsic data integrity and peer validation.
>
> - **intrinsic data validity** - what sort of data integrity guarantees people need in order to interact meaningfully and safely with the data they receive from others. Half of the problem is already solved — when you have the 'rules of the game' on your machine, you can verify that your peers are playing the game correctly just by validating their data they create. On top of this, we add cryptography to prove authorship and detect third-party tampering.
> - **peer witnessing** - each piece of public data is witnessed, validated, and stored by a random selection of devices. Together, all cooperating participants detect invalid data, share evidence of corrupt actors or validators, and take steps to counteract threats. Most of the work is done before you even retrieve the piece of data you want.

Application architecture - reference

> *A **client** running on a participant's device talks with their **conductor**, which runs multiple **hApps**. Each hApp is made of one or more **cells**, which are the live instances of **DNAs***

*that run on behalf of the participant. These DNAs, in turn, are made of one or more executable **zome** modules.*

The entire stack of a Holochain hApp - reference.

1. A **zome** is a module that contains executable code and exposes some of its functions as an API. **Integrity zomes** define data types, while **coordinator zomes** define core application logic.
2. One or more zomes are bundled into a **DNA**, which is like a microservice that defines all the rules for a given network.
3. A DNA comes alive as a **cell** bound to an agent ID, running on behalf of a participant.
4. One or more cells are slotted into roles in a **hApp**, which makes up an application's back end.
5. A participant's **conductor** hosts the hApps she uses, mediating local and network access to them and executing their code.
6. The participant's **clients** access the hApp via the conductor's local RPC interface, while the conductor also allows the hApp to communicate with other participants' cells that belong to the same networks.

An **application** consists of a **client** and a **hApp** (a collection of separate microservices called **DNA**s which in turn are composed of code modules called **zomes**) running on the devices of all participants. The **hApp** runs in the **conductor**, Holochain's application server runtime.

Source chain - reference: to create and store data related to an application.

DHT - reference: database provides redundancy and availability for data and gives the network the power to detect and take action against corruption.

# Advanced courses

- Holochain's Self-Paced Training
- Holochain Gym

# Holochain Resources

- Official Holochain Programming Resources
- Tryorama (JavaScript library for testing the behavior of multiple Holochain nodes in a network)
- Wind Tunel (Performance testing for Holochain)
- p2p Shipyard (a development environment for creating Android, Linux, MacOS and Windows holochain applications)
- hREA (Scalable & distributed framework for economic network coordination)

- Holochain scaffolding tool (tool to generate back-end and UI code for all the data types and collections needed to build a functioning basic hApp)
- Holochain Open Dev (Reusable Modules)
- Happenings.community developer tools database

Rust

# Rust

*"Rust compile times give you a well needed break before the borrow checker reminds you why you are a bad programmer"*

## Install Rust on your machine

Download Rust

Or, if you use a Linux machine, just pass the following command into the terminal

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

NOTE: get into the habit to pass the following command every time you do some programming. They do updates every 6 weeks.

```
rustup update
```

If you install Rust on a Linux machine, you need some additional dependencies.
On Debian machine, use this command to install them :

```
sudo apt install build-essential
```

## Install extensions in your IDE, VSCode or Cursor

Find those in  Extensions / Ctrl+Shift+x

- Rust analyzer
- Crates
- Even Better TOML
- Rust Test Explorer
- TODO Tree - use it to tag in comments TODO or FIXME and then easily search and find these places in code.

## Core Learning Materials

- Official learning resources: Learn Rust
- **Official Documentation: The Rust Book**
- **The Rust Reference** - Another great manual**.**
- Video Tutorials:
  - Rust in 100 seconds (Youtube)
  - Rust book tutorial (Youtube)
  - Course for beginner (Youtube)
- **Tourofrust**

# Practice Exercises

- Rustfinity (a website for learning and practicing Rust)
- Rustling (small exercises to practice reading and debugging Rust code)
- Exercism's Rust track (exercises on common algorithms and data structures)
- CodeWars (Kata exercises for practicing different languages, including Rust)

# Additional Resources

- [ ] Flattening Rust's Learning Curve (Blog article)
- [ ] Rust Class (Book)
- [ ] Let's Get Rusty (Youtube channel)
- [ ] The Rust Standard Library is SO Confusing...Until Now! (Youtube video)
- [ ] Compiler-Driven Development in Rust (Youtube video)
- [ ] All Rust features explained (Youtube video)
- [ ] The genius of Rust constructors (Youtube video)
- [ ] The Rust Survival Guide (Youtube video)
- [ ] But what is ‘a lifetime? (Youtube video)
- [ ] Embedded Rust setup explained (Youtube Video)
- [ ] Wokwi (A embedded simulator for writing embed rust without device)

# Advanced Topics

These topics are not necessary for building holochain applications, but are still relevant for sharpening your Rust skills and could be useful for advanced hApps.

- Smart pointers
- Interior Mutability
- Concurrency and multi-threading
- Unsafe Rust

# Tricks in Rust

## Placeholder functions

```rust
// Use these macros to mark incomplete functions:
fn test() -> Result<String> {
    unimplemented!("This function is not yet implemented");

    // Or
```

```
    todo!("Implement this function later");
}
```

## Display and Debug traits

```rust
// Println! macro usage
println!("{}"); // prints the variable using Display trait
println!("{:?}"); // prints the variable using Debug trait

// Implementing Display and Debug traits
#[derive(Debug)] // automatically implements Debug trait
struct MyStruct {
    field: i32,
}

impl std::fmt::Display for MyStruct {
    fn fmt(&self, f: &mut std::fmt::Formatter<_>) -> std::fmt::Result {
        write!(f, "{}", self.field)
    }
}

fn main() {
    let my_struct = MyStruct { field: 42 };

    println!("{}", my_struct); // Uses Display trait
    println!("{:?}", my_struct); // Uses Debug trait
}
```

# 🔧 WebAssembly

# WebAssembly

WebAssembly (WASM) is a binary instruction format designed to enable compilation of languages like C, C++, Rust, and others to run in web browsers and other environments with near-native performance.

Holochain uses WebAssembly to compile high-level languages like Rust into efficient, platform-independent bytecode that can run in web browsers and other environments. This approach allows Holochain to leverage the strengths of both Rust and WebAssembly.

## WebAssembly Integration

Holochain uses WebAssembly to compile high-level languages like Rust into efficient, platform-independent bytecode that can run in web browsers and other environments. This approach allows Holochain to leverage the strengths of both Rust and WebAssembly.

### Key points to consider:

- WebAssembly provides near-native performance for Holochain agents running in web browsers.
- It enables Holochain to create secure, sandboxed environments for executing agent code.
- WebAssembly modules can be easily distributed and executed across different platforms.

## Agent Execution

In Holochain, WebAssembly plays a crucial role in executing agents:

- Agents are compiled into WebAssembly modules during development.
- These modules are then loaded and executed within the Holochain runtime environment.
- WebAssembly ensures that agent code runs securely and efficiently within the constraints of the web browser.

## Performance Optimization

WebAssembly helps optimize Holochain application performance:

- It compiles high-level languages to low-level machine code, resulting in faster execution times.
- WebAssembly modules can take advantage of hardware acceleration, improving overall system performance.
- The ability to share memory between WebAssembly modules allows for efficient inter-agent communication.

# Security Features

Holochain leverages WebAssembly's built-in security features:

- WebAssembly modules run in a sandboxed environment, preventing access to sensitive system resources.
- The strict type system of WebAssembly helps catch potential errors at compile-time.
- WebAssembly modules can be verified for safety using tools like Wasmtime or WebAssembly Binary Toolkit (wabt).

# Cross-platform Compatibility

WebAssembly enables Holochain applications to run consistently across different platforms:

- WebAssembly modules can be executed in various environments, including web browsers, Node.js, and even native operating systems.
- This cross-platform compatibility allows Holochain applications to maintain consistent behavior regardless of the target platform.

# Development Workflow

The integration of WebAssembly in Holochain affects the development workflow:

- Developers can write agent code in high-level languages like Rust and compile them to WebAssembly modules.
- Tools and frameworks exist to simplify the process of developing and testing WebAssembly-based Holochain applications.
- Continuous integration and deployment pipelines can be set up to automatically compile and test WebAssembly modules.

By leveraging WebAssembly, Holochain applications can achieve better performance, enhanced security, and improved cross-platform compatibility, ultimately leading to more robust and efficient decentralized applications.

# Resources to study

- Rust WebAssembly Book (Official book)

# Web Frontend

# Web frontend

## Introduction to web dev

- 100+ Web Development Things you Should Know (Youtube video)
- HTML in 100 Seconds (Youtube video)
- CSS in 100 Seconds (Youtube video)
- JavaScript in 100 seconds (Youtube video)

## HTML and CSS

- HTML & CSS Crash Course Tutorial (Youtube playlist)
- Mozilla Developers Network (MDN) CSS documentation
- W3schools tutorials :
    - HTML tutorial
    - CSS tutorial
- Slaying The Dragon CSS course (Youtube Playlist)
- Learn Accessibility - Full a11y Tutorial (Youtube video)
- Tailwind CSS (atomic css framework)
- UnoCSS (atomic css engine)

### VSCode extensions

- HTML CSS Support
- CSS Navigation
- CSS Peek
- CSS Var Complete
- Tailwind CSS
- UnoCSS

## JavasCript/TypeScript

- Mozilla Developers Network (MDN) JavaScript documentation
- W3Schools JavaScript tutorial
- Javascript.Info (very good documentation site about JS)
- 100+ JavaScript Concepts you Need to Know (Youtube video)
- BroCode javascript tutorial (Youtube playlist)
- Deno (Modern javascript and typescript runtime)
- TypeScript tutorials :
    - TypeScript (Official documentation)
    - TypeScript (Official handbook)

        - Learn TypeScript in 50 Minutes - TypeScript Beginner Crash Course (Youtube video)
        - TypeScript Full Course - From Beginner to Advanced (4h Youtube video)
- Frameworks :
    - Svelte - Cybernetically enhanced web apps (JavaScript framework)
    - SvelteKit (Fullstack framework based on Svelte)
    - Skeleton UI (UI component library with customizable theme)
    - Shadcn Svelte (Beautifully designed components that you can copy and paste into your apps. Accessible. Customizable. Open Source)
- Effect TS (Functional programming library for TS)

## VSCode extensions

- npm Intellisense
- deno
- ESLint
- Svelte for VS Code

# 🏗 Applications

# Sensorica projects using Holochain

Nondominium
Request and Offers
NextGen NRP-CAS

# Meshnetwork

https://meshtastic.org/  Mesh network hardware and software.
meshtastic Rust crate

# Web servers

Tools
loco.rs  - library to build web projects.
Axum.rs  - library to build web servers, more limited than loco.

Proxmox.com  OS for virtual machine, to make virtual private servrs (VPS).

# 📎 Homework

Learn about ***json*** and ***clap***

- *CLAP crate*
  - *Derive Tutorial*
- *JSON*
  - *W3School*
  - *MDN*

Continuer les video tutorials de Rust from 5 to 9, inclusively.

_=>  // stands for default

# hREA

# hREA

Documentation

Help for installation - temporary URL, it will be updated.
**NOTE: There were small tweaks that we had to do to make it work, wait for updates…**

We installed svelt kit for UI

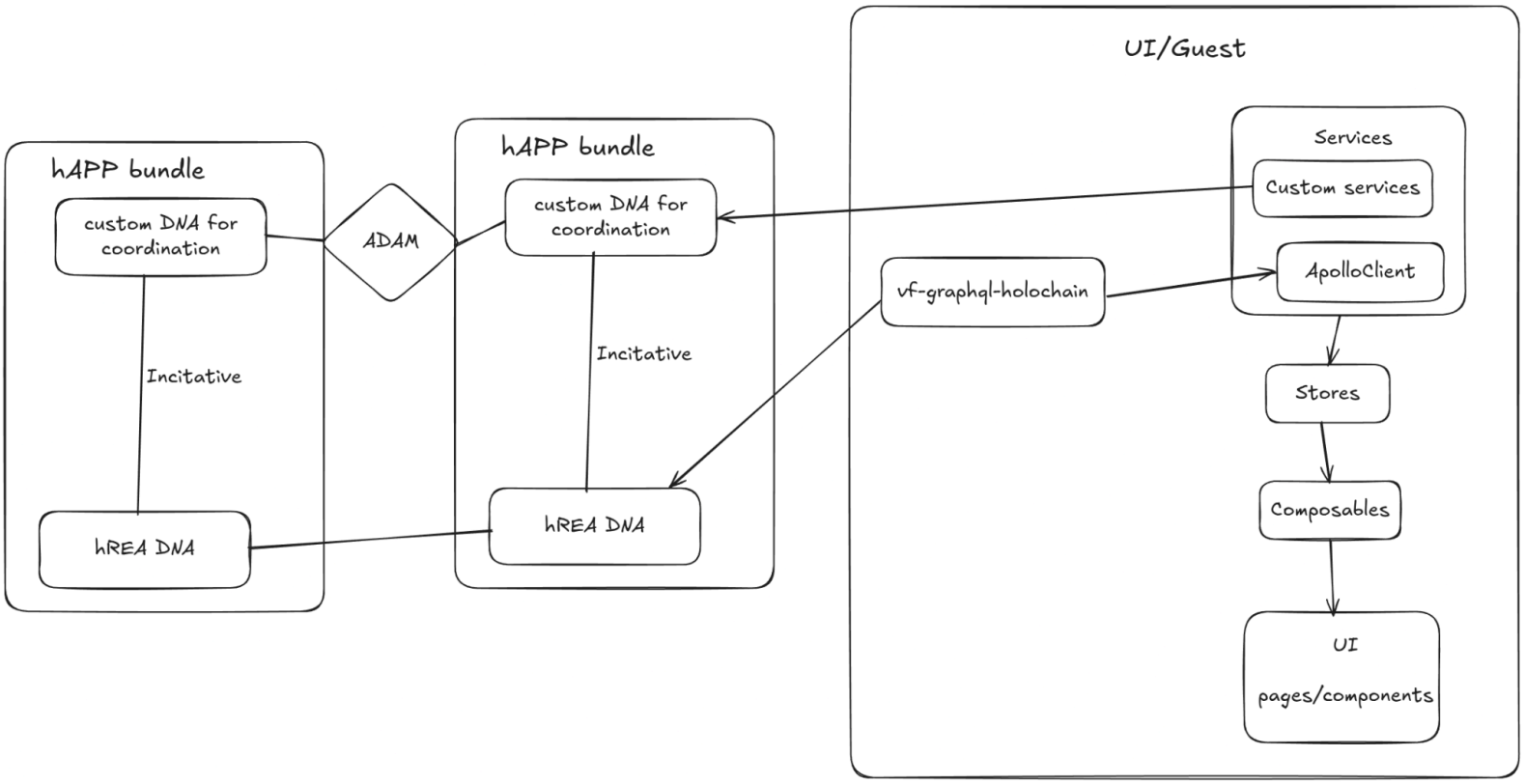
# 🧑‍💻 Next Gen NRP

# Holochain and hREA architecture for OVNs

A simple representation of a network-of-networks based on OVN model

Left, we have two OVNs, represented by an hApp, bundling a hREA DNA and a coordination DNA. The OVNs share the same hREA DNA, which allows them to interface at the economic level. What sets the two OVNs apart is their coordination DNA. The coordination DNA is where stigmergy (the coordination mode of OVNs) is implemented. The stigmergic coordination mode pushes planning, request (task generation) and commitment (promises to execute a task) to the edge of the network, allowing any agent to generate priorities, create tasks and take tasks. Coordination models vary depending on the type of organization. For example, a tech development organization is coordinated differently from a fabrication network (more scripted and deterministic) or from an artistic organization.

The UI/Guest constitute the digital environment where agents collaborate.




Question: What to put in the integrity zome and the coordination zome for the hREA DNA, in order to better manage security?

https://developer.holochain.org/build/validation/