READ MODE · ANSWER FIRST

OpenClaw

An interactive, visual explanation of OpenClaw, from setup through the complete system.

CHEAT SHEET · 7 ESSENTIAL IDEAS

The whole story in 7 lines

OpenClaw becomes easier to reason about when every stage is connected as one system.

1. Top-level topology: Mac hosts the Gateway, iPhone connects as a node, messaging channels plug in.
2. Hub-spoke showing gateway internals: channels, agent, sessions, node registry.
3. Bonjour mDNS discovery, TLS WebSocket, challenge/hello handshake, node registration.
4. Three JSON frame types over one WebSocket: req, res, event.
5. Full end-to-end: user types on iOS -> sessions.send -> Gateway -> agent -> LLM -> streams back.
6. Agent invokes node commands on iPhone: camera, canvas, screen.
7. Phone locked: Gateway queues commands, APNs relay wakes the phone, phone reconnects and pulls pending queue.

STAGE 1 OF 9

Setup

OpenClaw is a local-first AI platform. Instead of relying on a cloud service, you run your own AI on your own devices. Let us learn the vocabulary you will need before we dive in.

A Gateway is the local server that runs on your Mac. Think of it as mission control for your personal AI. It manages conversations, devices, and messaging channels.

A Node is any device that connects to the Gateway. Your iPhone is a node. So is your Mac itself. Each node registers its capabilities, like camera or screen recording.

A Channel is a messaging platform plugged into the Gateway. WhatsApp, Telegram, Discord, iMessage. Your AI is reachable on all of them at once.

The Agent (called Pi) is the AI brain inside the Gateway. It reads your messages, calls an LLM, and can control your devices. A WebSocket is the persistent two-way connection that ties everything together.

Those five terms cover every moving part. Now let us start with the big picture: how your Mac and iPhone form a personal AI network.

Your Devices, Your AI

02 YOUR DEVICES, YOUR AI

Here is the full picture of an OpenClaw setup. Everything you see runs on hardware you own. No cloud accounts, no subscriptions, no data leaving your network.

Your Mac sits at the center. It runs the Gateway, a persistent local server, plus a native macOS app that launches and monitors it. This is mission control for your personal AI.

Your iPhone connects to that Mac Gateway over your local network. Remember the term Node from the setup? The iPhone is a node. It registers itself and advertises capabilities like camera and voice.

Messaging channels connect outward from the Gateway. WhatsApp, Telegram, Discord, iMessage. Your AI is reachable wherever you already talk. Each channel is a direct connection, not a relay.

That is the whole topology: one Gateway on your Mac, one or more device nodes, and direct channel connections. Next, let us open up the Gateway and see what runs inside it.

The Gateway

We just saw the Gateway as a single box in the topology. Now let us open it up. Inside this one process, four subsystems work together to run your personal AI.

The Channel Manager holds live connections to every messaging platform. WhatsApp, Telegram, Discord, and more. Messages from the outside world arrive here first.

The Agent Runtime is Pi, the AI brain we introduced in the setup. It receives messages, calls the LLM, executes tools, and streams responses back to whichever channel started the conversation.

The Session Store persists conversation history in SQLite. Every exchange is recoverable even if the Gateway restarts, and the agent always has full context.

The Node Registry tracks every connected device: your iPhone, your Mac node process, any other phones. It stores their capabilities and routes commands to them.

Four subsystems, one process, one machine. That is the entire backend for your personal AI. Next, let us see how your iPhone discovers and connects to this Gateway.

STAGE 4 OF 9

iPhone Connects

We saw the Node Registry inside the Gateway. Now let us watch an iPhone register itself. The phone does not know where the Gateway lives yet. Here is the three-step connection sequence.

The iOS app sends a Bonjour query onto the local network: who is running _openclaw-gw._tcp? This is like shouting "is anyone home?" on your Wi-Fi.

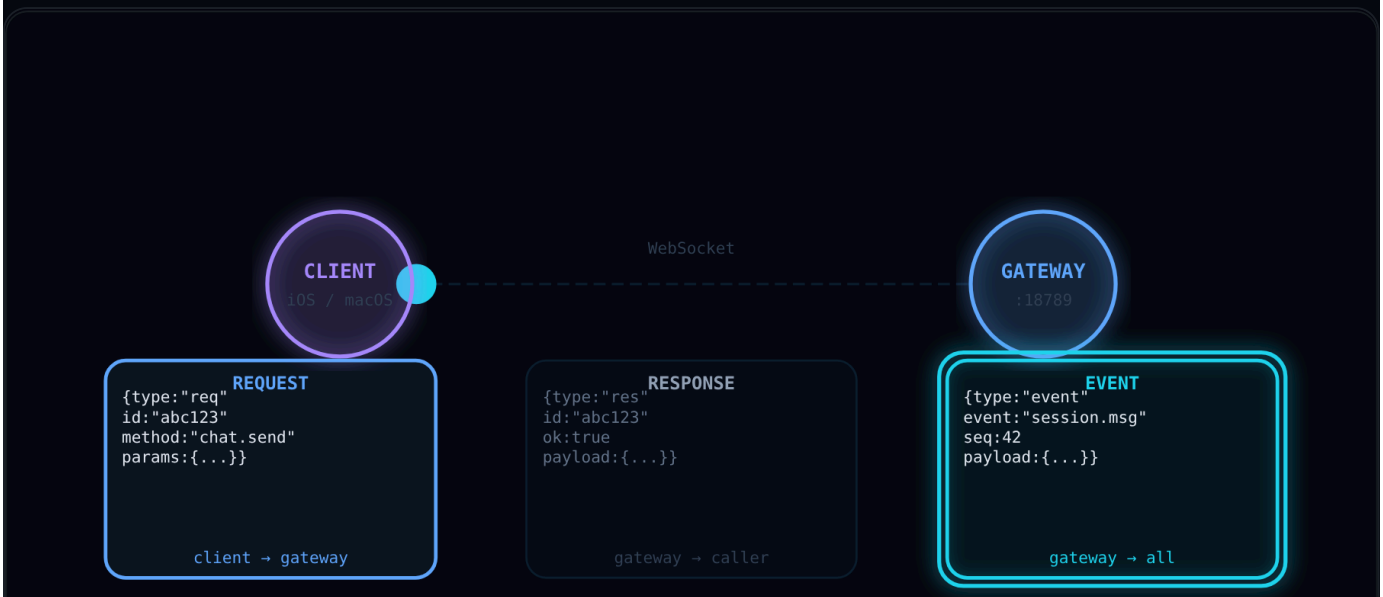
The Gateway replies via mDNS with its address and port. The iPhone now has a target. If Tailscale is configured, remote gateways are also discoverable.

The app opens a TLS WebSocket to the Gateway. On first connection you verify the TLS fingerprint once. The Gateway issues a device token stored in the Keychain.

The Gateway sends a challenge nonce. The iPhone signs it with its private key and replies, proving device identity without transmitting a password.

The Gateway responds with hello-ok: protocol version, server info, and a state snapshot. The iPhone is now a registered node. Next, let us look at the language these devices speak.

The Protocol



05 THE PROTOCOL

The iPhone is connected. Now what language does it speak? All communication uses one persistent WebSocket and three frame types. Every frame is a JSON text message.

A Request carries a method name, params, and a client-generated id. The id ties the request to its reply. This is how iOS sends a message or asks for history.

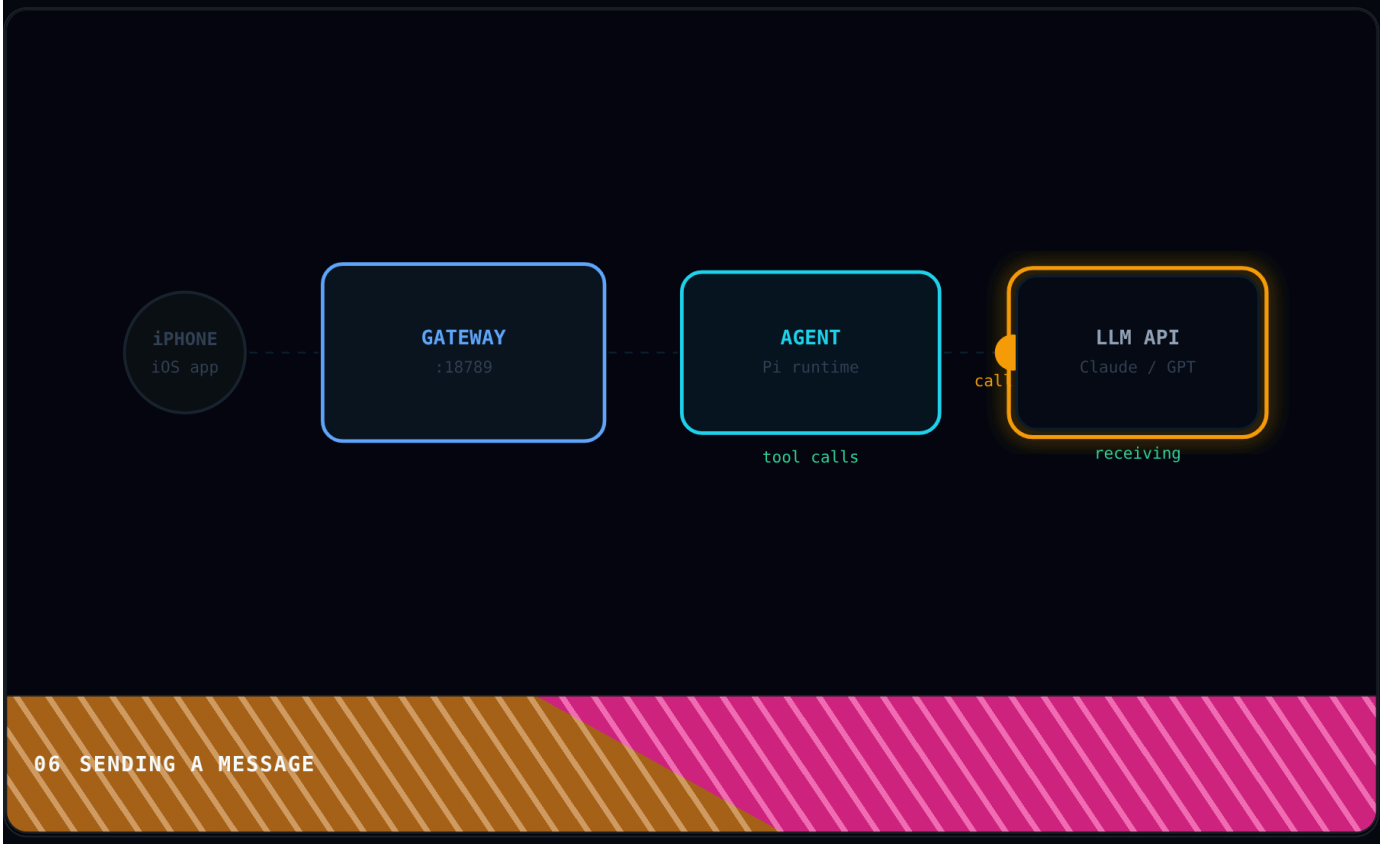
A Response carries that same id back, plus a payload on success or an error object on failure. The Gateway sends it directly to the requesting client only.

An Event is a server push with no id and no prior request. The Gateway broadcasts events like new chat messages, node status changes, and heartbeat ticks.

Try switching the Frame control in the sidebar. Notice how each frame type has a different structure and direction. Request goes client to gateway. Response comes back. Events are broadcast to everyone.

Three frame types on one connection. That is the complete protocol surface. Next, let us follow an actual message through this protocol from start to finish.

Sending a Message



We know the protocol. Now let us use it. You type a message in the OpenClaw iOS app. Watch it travel through every hand-off from your fingers to the AI response.

The iOS app sends a sessions.send request over the WebSocket. Remember the Request frame from the previous stage? This is one. The Gateway acknowledges it immediately.

The Gateway routes the message to the Agent runtime and loads the session context from SQLite. The agent now has the full conversation history.

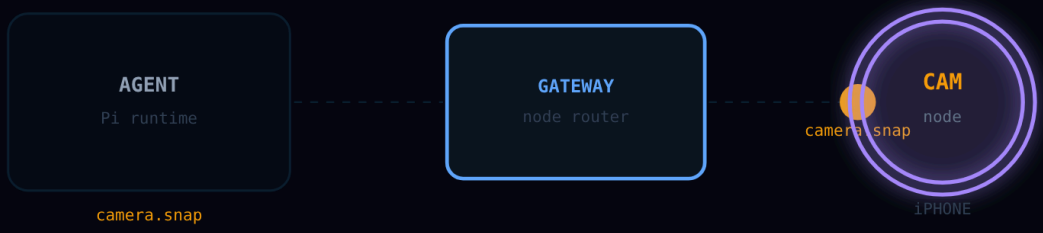
Pi processes the message. It may call tools first: search the web, invoke the camera, read a file. Then it constructs the prompt for the LLM.

Pi calls the LLM API. Claude, GPT-4, or whatever model you configured. The model streams tokens back as they are generated. Switch the Origin control to Telegram to see the same flow from a channel.

The Gateway pushes session.message events to all connected clients as tokens arrive. The iOS app renders the response as it streams in, token by token.

iOS to Gateway to Agent to LLM and back. Entirely over connections your devices own. Next, let us see what happens when the agent needs to reach out and control your iPhone.

Node Commands



07 NODE COMMANDS

We just saw the agent receive messages. Now the flow reverses. The agent can reach out and control your iPhone through the Node Registry we learned about in the Gateway stage.

The agent decides it needs a capability from the iPhone. It emits a node.invoke call specifying the command and any parameters.

The Gateway looks up the target node in its registry and forwards the invoke command over that node's live WebSocket connection.

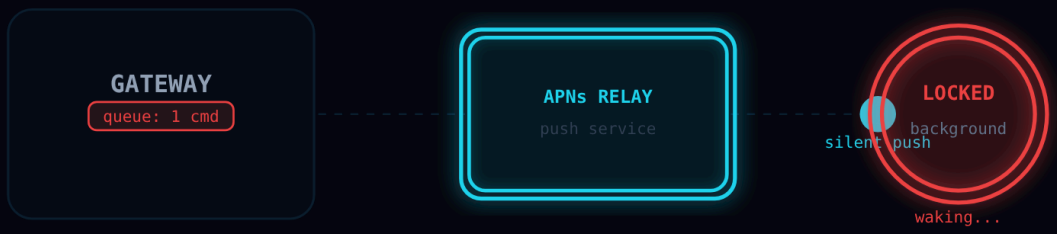
The iOS app receives a node.invoke.request event. It checks its declared capabilities and begins executing. Try switching the Capability control to see different actions.

The iPhone acts: captures a photo, navigates the WKWebView canvas, or starts a screen recording. The result flows back to the Gateway and then to the agent as a tool result.

A JPEG buffer, a canvas snapshot, a video clip. The agent has eyes and hands through your devices. But what if the phone is locked? That is the final challenge we will tackle next.

STAGE 8 OF 9

Background Wake



08 BACKGROUND WAKE

We saw the agent control an iPhone that is awake and connected. But what happens when the phone is locked in your pocket? The Gateway has a graceful answer for exactly this.

The agent issues a node.invoke for the iPhone. The Gateway checks its registry, finds the node is offline, and queues the command rather than dropping it.

The Gateway contacts the APNs relay, a small external service that holds your push credentials. It requests a silent push notification for your device.

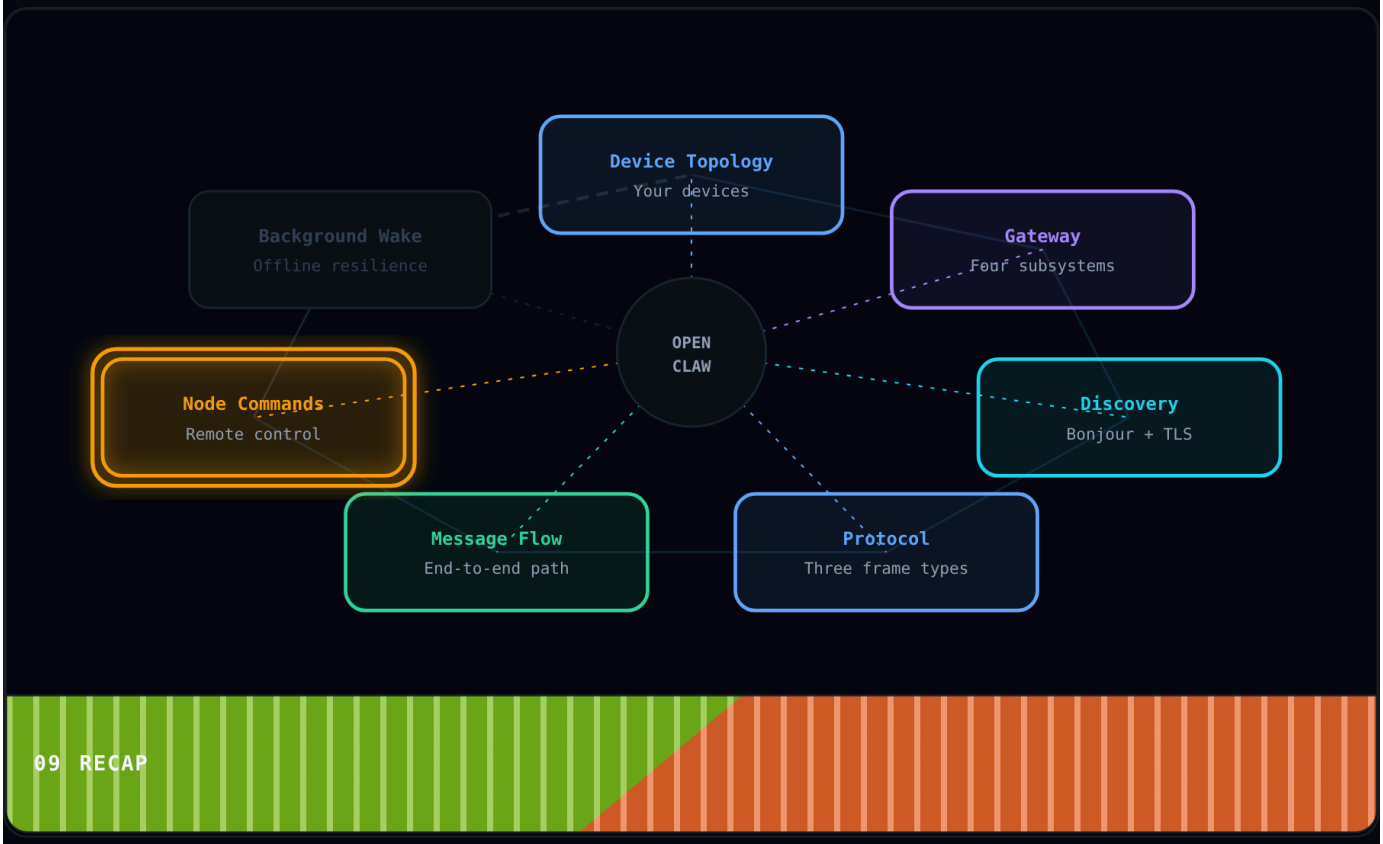
Apple delivers the silent push to your iPhone. iOS wakes the OpenClaw app in the background just long enough to reconnect to the Gateway.

The iPhone reconnects and calls node.pending.pull. The Gateway delivers the queued command. The iPhone executes it and sends back the result.

Locked phone, silent push, background reconnect, execute, return result. Your agent never dropped the task. Now let us step back and see the whole picture together.

STAGE 9 OF 9

Recap



09 RECAP

We started with the topology: one Mac running the Gateway, iPhones connecting as nodes, and messaging channels plugged in directly. Everything on your own hardware.

We opened the Gateway and found four subsystems: the Channel Manager, the Agent, the Session Store, and the Node Registry. One process does it all.

We watched an iPhone find and connect to the Gateway using Bonjour discovery, TLS verification, and a challenge-response handshake. Zero configuration needed.

We learned the protocol: three JSON frame types over a single persistent WebSocket. Simple enough to fit on an index card.

We followed a message from your fingertips through the Gateway, into the Agent, out to the LLM, and back as a streaming response. Every hop stayed on your network.

We saw the agent control your iPhone through node commands: snap a photo, capture the screen, navigate a canvas. Your AI has eyes and hands through your devices.

We solved the offline problem: the Gateway queues commands, a silent push wakes the phone, and it reconnects to pull pending work. No task is ever lost.

That is OpenClaw. A personal AI that lives entirely on your devices, speaks one simple protocol, and stays capable even when your phone is in your pocket. Local first, always yours.