Blueprint replicable de un agente tipo Cliq – asistente de viaje

Nota de autoría: Documento elaborado por **Phineas (Ops)** para Patricio Rojas Errázuriz, como blueprint conceptual replicable.

Propósito: describir, a nivel conceptual y replicable, la estructura y los principios de un agente operativo "de verdad" (no solo un LLM), usando **OpenClaw como ejemplo** sin depender de OpenClaw.

Lo que NO incluye: secretos, tokens, rutas privadas, identificadores personales, ni detalles de implementación específicos de una máquina.

---

1. Qué estás construyendo (y qué no)

Un "agente" útil en operación cotidiana es un **sistema socio-técnico mínimo**:

- Un modelo (LLM) para razonamiento/lingüística.
- Un **contrato de comportamiento** que reduce alucinaciones y fricción.
- Un conjunto de **fuentes de verdad (SoT)** donde "vive" la realidad.
- Un set de **procedimientos repetibles (runbooks)** para tareas recurrentes.
- Un esquema de **gates** (confirmaciones) para acciones con consecuencias.
- Un mecanismo de **continuidad** (memoria curada, no solo historial).

Lo que *no* es: "un prompt grande". Un prompt ayuda, pero no reemplaza: - SoT, - runbooks, - control de acceso, - evidencia/observabilidad, - y disciplina de veracidad.

---

2. Las cinco piezas (SOUL / USER / MEMORY / TOOLS / RUNBOOKS)

Piensa estas piezas como una pequeña constitución + manual operacional.

2.1 SOUL — Constitución del agente

Función: definir *cómo* decide y *cómo* se comporta el agente, especialmente bajo presión.

Contenido recomendado (plantilla conceptual):

1) **Regla de veracidad (anti-invención)**
- Si no está verificado: decir "no lo verifiqué" y pedir 1 dato mínimo.
- Separar siempre: *verificado* vs *inferido* vs *pendiente*.

2) **Modos: ideas vs ejecución**
- Diseño/explicación: no cambia estado.
- Ejecución: cambia estado (archivos, config, envíos) solo con gatillo/OK.

3) **Gates de acciones con consecuencias**
- Enviar correos a terceros / publicar / pagos / cambios de cron/config: confirmar.
- Para acciones internas reversibles: ejecutar sin fricción.

4) **Estilo y UX**
- Responder directo, sin relleno.
- Evitar "preguntas ansiosas": resolver lo resoluble leyendo el corpus/SoT.
- Si falta info: 1 pregunta mínima (no forks A/B/C).

5) **Comportamiento en grupos**
- Participar sin dominar.
- Responder cuando hay mención, pregunta directa o valor claro.

OpenClaw (ejemplo): esto se materializa en un archivo `SOUL.md` que el agente lee al iniciar.

2.2 USER — Contrato con el humano

Función: dejar claro qué espera el usuario del agente.

Incluye: - cómo llamarlo, - preferencias (nivel de detalle, idioma, tono), - límites de privacidad, - qué acciones se consideran autorizadas por defecto, - qué acciones requieren confirmación.

En agentes "de operación", USER es la fuente para: - no pedir permiso para cada lectura, - no ejecutar cosas externas sin OK, - saber cuándo ir a runbooks y cuándo improvisar está prohibido.

OpenClaw (ejemplo): `USER.md`.

2.3 MEMORY — Memoria curada (pines)

Función: continuidad real y reducción de re-trabajo.

No es un dump de eventos; es una lista curada de: - decisiones operativas, - reglas semánticas ("cuando diga X, significa Y"), - fuentes de verdad por dominio, - punteros a runbooks canónicos.

Regla: MEMORY debe permitir "volver en 5 segundos" al lugar correcto.

OpenClaw (ejemplo): `MEMORY.md` con secciones tipo "PIN — …".

2.4 TOOLS — Parámetros de entorno (no secretos)

Función: habilitar operación rápida.

Contiene: - IDs útiles (chat/grupos/usuarios) cuando corresponda, - nombres de dispositivos/servicios, - rutas canónicas (si no exponen información sensible), - defaults operativos (p. ej., cuenta de correo por defecto).

No debe contener: - tokens, - passwords, - claves privadas.

OpenClaw (ejemplo): `TOOLS.md`.

2.5 RUNBOOKS — Procedimientos repetibles

Función: que el agente opere con consistencia y evidencia.

Un runbook bueno define: - Gatillo (cuándo se usa) - Inputs - Pasos - Artefactos (output) - Evidencia (cómo se verifica) - Fallback - Guardrails

La diferencia entre "sistema" y "improvisación" es que: - los runbooks reemplazan el "pensar desde cero" en tareas recurrentes, - reducen errores, - y hacen auditables los procesos.

OpenClaw (ejemplo): `docs/runbooks/index.md` como índice y runbooks por operación.

---

3. Fuente de verdad (SoT) vs derivados

3.1 SoT (Source of Truth)

Definición: un repositorio único que manda cuando hay conflicto.

Ejemplos: - un JSON canónico, - una base SQLite, - un CSV maestro, - un spool `.eml`.

Regla: el agente siempre debe saber "qué manda" y debe operar sobre ese SoT.

3.2 Derivados

Dashboards, HTML, reportes, resúmenes: son derivados.

Regla: los derivados se regeneran. El agente no debe tratarlos como SoT.

---

4. Secuencias operativas (lo que hace que parezca 'inteligente')

4.1 Triage de mensajes (texto)

1) Clasificar la intención:
- Consulta interna (buscar lo que ya existe)
- Registro/definición (actualizar SoT)
- Investigación externa (traer opciones)

2) Ejecutar por clase:

- Consulta interna: responder desde SoT.
- Registro: actualizar SoT + evidencia (solo con OK si es cambio persistente).
- Investigación: 2–3 opciones concretas; registrar solo si hay decisión.

4.2 Audio = texto (gestión de audio)

Para que el audio sea "operable" (no solo transcrito), el sistema debe tratarlo como mensaje normal:

1) **Ingesta durable** (evitar referencias efímeras)
2) **Transcripción** (idealmente local, por privacidad y costo)
3) **Snapshot inmutable** (trazabilidad)
4) **Acción** sobre el contenido:

- si hay instrucción clara → ejecutar siguiente paso,
- si es ambiguo → resumen + 1 pregunta mínima.

Regla UX: por defecto no pegar el transcript en el chat; usarlo como input interno.

4.3 Acciones con consecuencias (gates)

Antes de: - enviar correos a terceros, - publicar, - pagos/reservas, - tocar crons/config,

el agente debe: - mostrar intención + riesgos mínimos, - pedir confirmación, - ejecutar, - reportar evidencia.

---

5. Control de acceso (allowlist) para agentes 'compartibles'

Si el agente va a ser usado por terceros:

- Identidad separada (canal/bot propio)
- Allowlist de IDs permitidos
- Opcional: pairing/approval para DM

Esto convierte al agente en un "servicio con llaves".

OpenClaw (ejemplo): `dmPolicy: pairing` + `groupPolicy: allowlist` + `groupAllowFrom`.

---

6. Observabilidad y evidencia (la diferencia entre demo y operación)

Un agente operativo necesita: - logs por proceso, - estado/cursor persistente (para no duplicar trabajo), - artefactos inmutables para fuentes (snapshots), - una convención de nombres y carpetas.

Esto habilita RCA (root cause analysis) cuando algo “no llega”, “se duplicó” o “se perdió”.

---

7. Cómo replicarlo (checklist de implementación agnóstica)

1) Escribe SOUL (veracidad, modos, gates, estilo)
2) Escribe USER (contrato)
3) Define MEMORY (pines + punteros a SoT/runbooks)
4) Define TOOLS (parámetros no secretos)
5) Define SoT por dominio
6) Escribe 3 runbooks mínimos:
   - triage texto,
   - audio→transcripción→acción,
   - acciones externas con gate.
7) Define control de acceso (allowlist/pairing)
8) Define evidencia: logs + cursores + snapshots.

---

8. OpenClaw como ejemplo (solo para aterrizar)

En OpenClaw, el patrón típico se materializa como: - Un **agent** con workspace propio - Un canal de Telegram (cuenta/bot) rutado a ese agente - Políticas de DM/grupo con allowlist - Runbooks en `docs/runbooks/` y un índice - Memoria curada en `MEMORY.md`

Lo importante no es OpenClaw; lo importante es el patrón: **contrato + SoT + runbooks + gates + control de acceso**.

Nota final editorial

La “sofisticación” real está en que el agente: - no inventa, - sabe dónde está la verdad, - repite procedimientos con evidencia, - y se detiene antes de causar daño.

Eso es lo que hace que la experiencia se sienta consistente y confiable.

Créditos y agradecimientos

Este documento fue elaborado por **Phineas (Ops)**.

Agradecemos a **Fernando Varela** por compartir su **diseño inicial**, que sirvió como referencia clave para orientarnos y permitió adaptar y evolucionar la propuesta hacia nuestras necesidades operativas.