ChatGPT

Design e implementação de Skills, Agents, Rules e Workflows para IDEs e CLIs no projeto Openclaw

Resumo executivo

O ecossistema de "IDE + agente" está convergindo rapidamente para um conjunto de primitivas comuns — **instruções persistentes (rules/instructions)**, **prompts reutilizáveis acionados por slash commands (workflows/prompt files)**, **pacotes de conhecimento + scripts carregados sob demanda (skills)** e **personas/operadores configuráveis com escopo e ferramentas (agents/custom agents)**. Essa convergência é visível em três frentes técnicas que já possuem documentação madura e interoperável: (i) **Agent Skills via** `SKILL.md` **(padrão aberto)**, (ii) `AGENTS.md` **para instruções de projeto**, e (iii) **MCP (Model Context Protocol) para ferramentas externas** — cada uma com implementações em IDEs e CLIs relevantes (VS Code/Copilot, Codex, JetBrains AI, Cursor, Google Antigravity). [1]

Para o **Openclaw**, o caminho mais robusto (e com menor custo de manutenção) é adotar uma **arquitetura "core + adaptadores"**:

- **Core (Openclaw CLI/daemon)**: define o modelo canônico de Skill/Agent/Rule/Workflow, aplica políticas (aprovação, sandbox, allowlists), expõe operações via **CLI (terminal-first)** e opcionalmente via **servidor MCP** para que IDEs consumam ferramentas de modo padronizado. [2]
- **Adaptadores por plataforma**: geram/consomem artefatos nativos de cada ambiente (p. ex., `SKILL.md`, `.agent.md`, `.prompt.md`, `.rules`, `.github/hooks/*.json`, `.cursor-plugin/plugin.json`, diretórios `.agent/*` do Antigravity), evitando reimplementar lógica de negócio em extensões. [3]

Essa abordagem mantém compatibilidade com:
- **Google Antigravity** (Rules/Workflows/Skills por arquivos e diretórios locais, com políticas de execução e allowlists) [4]
- **Cursor** (plugins que empacotam skills/subagents/rules/hooks/MCP; além de compatibilidade com extensões VS Code e reforço de sandbox/controles) [5]
- **VS Code** (custom instructions, prompt files, custom agents, agent skills, hooks, Language Model Tools API, Chat Participant API, MCP) [6]
- **Codex** (CLI + extensão IDE; config em `config.toml`; `AGENTS.md`; regras para comandos fora do sandbox; skills com `SKILL.md`; MCP compartilhado entre CLI/IDE) [7]
- **JetBrains IDEs** (integração nativa do Codex via chat/AI; MCP para conectar servidores e também "IDE como MCP server"; ACP para "custom agents" como processos) [8]

Requisitos e casos de uso-alvo do Openclaw

Hipóteses explícitas (porque requisitos específicos do Openclaw não foram fornecidos)

1. **Openclaw precisa operar em múltiplas superfícies**: IDEs (VS Code, Cursor, JetBrains), CLIs (terminal local) e possivelmente "agent managers" dedicados (Google Antigravity, Codex Cloud). [9]
2. **Execução local é necessária** (ler/editar arquivos, executar testes/comandos), mas com controles de segurança (aprovação, sandbox, allowlists/denylists) por padrão. [10]
3. **O modelo de extensibilidade deve ser compartilhável** dentro de equipes (artefatos versionados no repositório) e reutilizável entre projetos (artefatos por usuário/perfil). [11]
4. **O Openclaw deve suportar ferramentas externas** (CI, issue trackers, deploy, docs, DBs) sem acoplamento a um IDE específico, favorecendo MCP e/ou CLI pluginável. [12]
5. **Workflows precisam ser auditáveis e reprodutíveis** (logs, diffs revisáveis, política de aprovação, rastreabilidade). [13]

Casos de uso prioritários (mapeados para Skills/Agents/Rules/Workflows)

- **Build/test/quality loop**: rodar testes, interpretar falhas, propor patch, reexecutar testes, gerar relatório. (Workflow + Skill + Rule de segurança + Agent "QA/repair"). [14]
- **Code review automatizado com guardrails**: checklist de revisão, execução opcional de linters, sumarização de mudanças, bloqueio de ações destrutivas. (Skill + Rule + Hooks). [15]
- **Gerar scaffolding padronizado**: componentes, módulos, integrações, templates. (Prompt files / workflows + regras de convenção). [16]
- **Integrações com SaaS/devtools via "tool calling"**: p.ex., criar issue, atualizar PR, consultar pipeline, puxar docs/telemetria. (MCP servers + tool allowlists). [17]
- **Operação multi-plataforma (Windows/Linux/macOS)**: instalação simples, binários/empacotes, PATH, permissões, assinaturas. [18]

Arquitetura e modelos para Skills, Agents, Rules e Workflows

Padrão arquitetural recomendado: "Openclaw Core + Adaptadores"

Princípio: a "inteligência operacional" (políticas, execução, logging, compatibilidade) deve viver no **Openclaw Core**, enquanto IDEs consomem o core via artefatos (arquivos) e/ou via protocolos (MCP/stdio) e expõem UI/commands mínimos. Isso reduz divergências entre plataformas e facilita auditoria e governança. [19]

```
flowchart LR
  subgraph Surfaces["Superfícies (IDEs e Terminal)"]
    VSCode["VS Code / forks (Cursor, Windsurf etc.)"]
    JetBrains["JetBrains IDEs (AI Chat)"]
    Antigravity["Google Antigravity"]
    CodexCLI["Codex CLI"]
  end

  subgraph Openclaw["Openclaw Core"]
    CLI["openclaw (CLI)"]
```

```
    Daemon["openclaw-daemon (opcional)"]
    Policy["Policy Engine (aprovação/sandbox/allowlist)"]
    Registry["Registry (skills/rules/workflows/agents)"]
    Audit["Audit Logs + Diffs + Proveniência"]
  end

  subgraph Standards["Padrões e formatos reutilizáveis"]
    SkillStd["Agent Skills (SKILL.md + scripts/resources)"]
    AgentsMD["AGENTS.md (instruções)"]
    MCP["MCP (tools/prompts/resources)"]
  end

  VSCode --> CLI
  JetBrains --> CLI
  Antigravity --> Registry
  CodexCLI --> Registry

  CLI <--> Daemon
  Daemon --> Policy
  Daemon --> Audit
  Registry --> SkillStd
  Policy --> AgentsMD
  CLI <--> MCP
```

Definições canônicas e mapeamento para “primitivos” do ecossistema

A seguir, uma definição operacional (útil para implementação) que minimiza ambiguidade:

Skill (capabilidade especializada)
- Unidade **descoberta e carregada sob demanda**, contendo: metadados curtos + corpo de instruções + recursos (scripts, templates, docs).
- O formato `SKILL.md` com frontmatter YAML é especificado pelo padrão aberto **Agent Skills** e já é usado por múltiplos sistemas (VS Code/Copilot, Copilot CLI e outros), além de implementações equivalentes em plataformas como Antigravity e Codex. [20]

Rule (guardrails e “always-on instructions”)
- Instruções e restrições **sempre aplicadas** (ou aplicadas por escopo/padrão de arquivos), cobrindo estilo, arquitetura, segurança, dependências, práticas de testes, etc.
- Em VS Code, isso aparece como `.github/copilot-instructions.md`, `*.instructions.md`, `AGENTS.md` e até `CLAUDE.md` (para interoperabilidade). [21]
- Em Antigravity, rules são guias para o agente, persistidas globalmente e por workspace (ex. `~/.gemini/GEMINI.md`, `your-workspace/.agent/rules/`). [22]
- Em Codex, `AGENTS.md` **é lido antes do trabalho** e regras de execução de comandos fora do sandbox podem ser definidas via arquivo `.rules`. [23]

Agent (persona + ferramentas + escopo)
- Um “operador” com: instruções/prompt-base, preferências (modelo, modo plan/agent), listas de ferramentas permitidas (incluindo MCP), e limites (ex.: sem rede; sem escrita; só leitura).
- VS Code formaliza isso em **custom agents** `.agent.md` (com frontmatter incluindo `tools`). [24]

- JetBrains permite adicionar **custom agents via ACP**, onde o IDE lança um executável de agente como subprocesso a partir de `~/.jetbrains/acp.json`. [25]

Workflow (orquestração acionável pelo usuário ou por eventos)
- Procedimento repetível que encadeia ações (ex.: "rodar testes → corrigir → validar → reportar"), com gatilhos e checkpoints/revisão.
- VS Code usa **prompt files (`*.prompt.md`)** como "slash commands" com metadados (`agent`, `tools`, etc.). [26]
- Antigravity chama isso de **Workflows**: "saved prompts" acionados por `/`, persistidos por workspace (`.agent/workflows/`) e globalmente. [22]

Modelo de dados recomendado para o Openclaw (com compatibilidade como requisito)

Em vez de inventar formatos totalmente novos, o Openclaw pode definir um **modelo canônico interno** e "compilar" para formatos existentes:

- **Skill**: adote **Agent Skills** como formato de arquivo "source of truth" (`SKILL.md` + `scripts/`, `references/`, `assets/`), para maximizar reuso entre Copilot/VS Code, Codex e outros. [27]
- **Rule**: trate como "instructions" em camadas (global → repo → subpasta), com saída gerável para `AGENTS.md`, `.github/copilot-instructions.md`, `.instructions.md`, e regras específicas de execução (hooks/rules). [28]
- **Agent**: para VS Code use `.agent.md`; para JetBrains use ACP; para Cursor/plugins use "subagents"/agents empacotados (quando disponível) — mantendo o mesmo conteúdo "prompt-base" e as mesmas políticas de ferramentas. [29]
- **Workflow**: em VS Code, `*.prompt.md`; em Antigravity, `.agent/workflows/*.md`; em Codex, uma combinação de prompts/skills e (se necessário) automação via MCP/Agents SDK. [30]

Integrações, APIs e pontos de extensão por plataforma

Tabela comparativa de capacidades de integração

Plataforma	Primitivos nativos (Skills/Agents/ Rules/Workflows)	Extensibilidade programática	Empacotamento/ distribuição	Segurança/ sandbox (visão geral)
Google Antigravity	Skills por diretórios; Rules e Workflows por arquivos com escopo global/ workspace	Scripts (Python/ Bash) dentro de Skill; integração via extensão do navegador e políticas	Principalmente por arquivos no FS (home/workspace)	Políticas de execução no terminal; denylist; allowlist de navegação no browser agent

Plataforma	Primitivos nativos (Skills/Agents/ Rules/Workflows)	Extensibilidade programática	Empacotamento/ distribuição	Segurança/ sandbox (visão geral)
Cursor	Plugins empacotam skills/ subagents/rules/ hooks/MCP; também suporte a extensões VS Code	Hooks e MCP; além de extensões VS Code	Marketplace de plugins + bundles; extensões `.vsix` (por compatibilidade)	Sandbox com controles granulares (rede e FS) e políticas configuráveis
VS Code	Custom instructions, prompt files, custom agents, agent skills, hooks; MCP e tools	Extension API; Chat Participant API; Language Model Tools API; MCP	Marketplace + `.vsix` via `vsce`	Workspace Trust; hooks podem bloquear/ autorizar tools; políticas por arquivos e modo restrito
Codex (CLI + IDE extension)	`SKILL.md` skills; `AGENTS.md`; rules para comandos fora do sandbox; MCP compartilhado CLI/IDE	CLI; IDE extension; pode rodar como MCP server para orquestração externa	npm/Homebrew (CLI); extensão VS Code/forks; config por `config.toml`	Aprovação e modos de sandbox; rules para execução fora do sandbox; camadas de config e "trusted projects"
JetBrains IDEs	AI Chat com agentes; suporte MCP; custom agents via ACP (subprocessos)	Plugin SDK (tool windows, settings, execução); MCP client e "IDE como MCP server"	Marketplace (plugin zip); configurações e integradores via UI	MCP com logs; ACP com logging; controles dependem do agente/ ferramentas

Fontes principais para a tabela (por plataforma): Antigravity (rules/workflows/skills/allowlist/exec policy) [4]; Cursor (plugins, sandbox, marketplace) [31]; VS Code (customização, MCP/tools, hooks, workspace trust, publicação) [32]; Codex (CLI/IDE/config/mcp/rules/skills/agents) [33]; JetBrains (MCP/ACP e integração Codex) [34]

Google Antigravity

O que é "extensão/ponto de integração" no Antigravity
O Antigravity expõe extensibilidade principalmente por **artefatos no sistema de arquivos** (home/ workspace) e por **políticas de execução**:

- **Rules**: guiam comportamento do agente; podem ser globais (`~/.gemini/GEMINI.md`) ou por workspace (`your-workspace/.agent/rules/`). [22]
- **Workflows**: prompts salvos acionados por `/`, globais (`~/.gemini/antigravity/global_workflows/...`) ou por workspace (`your-workspace/.agent/workflows/`). [22]

- **Skills**: pacotes por diretório, com `SKILL.md` obrigatório e opcionais `scripts/`, `references/`, `assets/`, em escopo global (`~/.gemini/antigravity/skills/`) ou workspace (`<workspace>/.agent/skills/`). [35]

Segurança e controle operacional
- O Antigravity expõe **políticas de execução de terminal** e comportamentos de revisão, além de bloquear comandos via denylist e pedir aprovação quando necessário. [36]
- Para navegação web, recomenda **Browser URL Allowlist** em `HOME/.gemini/antigravity/browserAllowlist.txt` para mitigar prompt injection via sites. [37]

Como o Openclaw deve integrar
- Oferecer um gerador `openclaw init antigravity` que crie: `.agent/rules/`, `.agent/workflows/`, `.agent/skills/` no repo, e (opcionalmente) regras globais no caminho do usuário. [38]
- Empacotar ações executáveis como scripts dentro das skills (Bash/Python), mantendo "progressive disclosure" (metadados curtos, recursos carregados sob demanda). [39]

Cursor

O Cursor passou a formalizar extensibilidade em camadas:

1) **Plugins do Cursor (bundle nativo)**
- Plugins no Cursor empacotam **MCP servers, skills, subagents, rules e hooks** e são instaláveis via Marketplace. [40]
- O repositório oficial de plugins descreve a estrutura: `.cursor-plugin/plugin.json`, diretórios `skills/` (com `SKILL.md`), `rules/` (`.mdc`), e `mcp.json`. [41]

2) **Compatibilidade com extensões do VS Code**
- Há suporte a extensões VS Code e instalação manual via `.vsix` (muito usada quando a extensão não aparece no marketplace do Cursor). [42]

3) **Segurança e sandbox**
- O changelog do Cursor (v2.5 em 2026-02-17) descreve **controles granulares de rede no sandbox**, além de controles para acesso a diretórios/arquivos no filesystem, com allowlists/denylists inclusive para políticas de organização. [43]

Como o Openclaw deve integrar
- Tratar o Cursor como alvo "primeiro" para distribuição por **plugin bundle** (quando quiser empacotar várias capacidades), e como alvo "compatível" via VS Code extension (quando precisar de UI/editor integration). [44]
- Para segurança: refletir as políticas do Openclaw em configurações de sandbox/allowlist (rede/FS) e exigir aprovação para ações destrutivas, alinhando-se ao que o Cursor já expõe como controles. [45]

VS Code

O VS Code oferece dois "planos" de integração:

Plano A: artefatos de customização (sem extensão própria)
- **Custom instructions**: `.github/copilot-instructions.md`, `*.instructions.md`, `AGENTS.md`, `CLAUDE.md` etc. [46]

- **Prompt files** (workflows como slash commands): `*.prompt.md` em `.github/prompts`, com YAML frontmatter para `agent`, `tools`, etc. [47]
- **Custom agents**: `.agent.md` em `.github/agents`, com frontmatter incluindo `tools`. [48]
- **Agent Skills**: diretórios em `.github/skills/` (e outros locais configuráveis) com `SKILL.md`. [49]
- **Hooks**: arquivos JSON em `.github/hooks/*.json` (e também camadas `.claude/settings*.json`). Hooks podem bloquear tool calls (ex.: `PreToolUse`). [50]

Plano B: extensão VS Code (integração profunda)
- O **VS Code Extension API** permite comandos, views, configurações etc. [51]
- Para IA/agents, há APIs específicas:
- **Chat Participant API** (assistentes invocados por `@mention`). [52]
- **Language Model Tools API** (tools chamados pelo agente). [53]
- **MCP developer guide** (VS Code como cliente MCP, com transports e features declaradas). [54]

Segurança
- **Workspace Trust** define um boundary de segurança: em "Restricted Mode", o VS Code tenta impedir execução automática e limita agentes, tasks, debugging, settings e extensões — embora observe que extensões maliciosas podem ignorar o modo restrito. [55]

Como o Openclaw deve integrar
- **Começar pelo Plano A** (arquivos no repo) para obter valor rápido e compatibilidade com múltiplos agentes. E só depois criar uma extensão quando houver necessidade de UI/rich integration (ex.: painel Openclaw, visualização de diffs, telemetry). [56]
- Se criar extensão, usar:
- tool calling para expor `openclaw.*` como ferramentas (com validação e políticas no core); [53]
- MCP server do Openclaw como alternativa para reuso multi-cliente. [57]

Codex (CLI + IDE extension)

Codex CLI (terminal-first)
- O Codex CLI é um agente de codificação rodando localmente no terminal; pode ler/alterar/executar código no diretório selecionado; é open source e feito em Rust. [58]
- Instalação: `npm i -g @openai/codex` e execução `codex`; macOS/Linux suportados e **Windows experimental**, recomendando **WSL** para melhor experiência. [58]

Codex IDE extension
- A extensão funciona em **VS Code e forks (incluindo Cursor e Windsurf)**. [59]
- A configuração é compartilhada entre CLI e IDE, com arquivo `~/.codex/config.toml` e override por projeto `.codex/config.toml`. [60]

Configuração, instruções e regras
- `AGENTS.md`: o Codex lê instruções antes de trabalhar, com precedência global e por diretório até o CWD; suporta `AGENTS.override.md` e limites de tamanho; isso é essencial para governança e consistência. [61]
- **Rules**: arquivo `.rules` define políticas para execução de comandos fora do sandbox (ex.: `prefix_rule(...)`). [62]
- **Skills**: diretórios com `SKILL.md` + opcionais `scripts/`, `references/`, `assets/` e `agents/openai.yaml`; invocação explícita (`$`) ou implícita por `description`. [63]
- **MCP**: configuração no `config.toml`, gerenciável via `codex mcp ...`, compartilhada CLI/IDE; suporta `[mcp_servers.<name>]`. [64]

Codex como "componente" de workflows maiores
- Documentação mostra o Codex rodando como **MCP server** (`codex mcp-server`) e sendo orquestrado por um SDK externo para criar workflows determinísticos/revisáveis. [65]

Como o Openclaw deve integrar
- Tratar o Codex como referência de **terminal-first** e de "config layering" (user + project + overrides), espelhando essa estratégia no Openclaw. [66]
- Expor o Openclaw como MCP server (opcional), para que Codex/VS Code/JetBrains possam consumir as ferramentas do Openclaw de maneira uniforme. [67]

JetBrains IDEs

Há dois caminhos distintos:

Integração "via JetBrains AI" (sem plugin seu)
- O Codex já está integrado ao AI chat do JetBrains (a partir de v2025.3), com múltiplas opções de autenticação e níveis de autonomia (incluindo rodar comandos e acessar rede, conforme o modo). [68]

Integração "Openclaw como agente externo" via ACP
- JetBrains define **ACP**: você adiciona agentes editando `~/.jetbrains/acp.json`, e o IDE lança o agente como subprocesso (`command`, `args`, `env`). [69]

Integração "Openclaw como ferramenta externa" via MCP
- JetBrains AI Assistant conecta a MCP servers via JSON config, suportando STDIO e Streamable HTTP (e SSE legado) e expõe as ferramentas como comandos `/`. [70]
- JetBrains IDEs podem atuar como **MCP server** (a partir de 2025.2) para clientes externos (incluindo Cursor, Codex e VS Code), com auto-configuração para alguns clientes. [70]

Como o Openclaw deve integrar
- Implementar **MCP server do Openclaw** é o caminho mais "universal" (consumível por VS Code/JetBrains/Codex e, frequentemente, por outros clientes). [71]
- Implementar **ACP** é o caminho mais "JetBrains-native" para agente conversacional com UI do AI chat. [25]

Outros ambientes (não especificados pelo usuário)

Se o objetivo é cobrir "outros editores/plataformas" sem multiplicar integrações, o Openclaw deve priorizar **pontos de acoplamento padronizados**:

- **Agent Skills (`SKILL.md`)** como unidade portável. [72]
- `AGENTS.md` como ponto portável de instruções. [73]
- **MCP** como integração de ferramentas (stdio/HTTP). [74]

Terminal-first, CLI do Openclaw e estratégias de instalação cross-platform

Considerações de design “terminal-first”

O terminal é o “mínimo denominador comum” entre IDEs e agentes. O Codex CLI e a documentação de workflows com MCP indicam um padrão que funciona bem em ambientes reais: configuração por arquivo, aprovações explícitas, e possibilidade de rodar como servidor (MCP) para orquestração e reuso. [75]

Requisitos de CLI recomendados para o Openclaw:

- **Saída dual**: humana (default) e `--json` (para automação e IDE adapters).
- **Idempotência**: `openclaw init ...` deve ser reexecutável sem destruir customizações.
- **Scopes**: `--global` (usuário), `--project` (repo), `--profile` (ambientes). (Espelhar “config layering” como Codex). [66]
- **Aprovação e sandbox**: modos equivalentes ao “ask/allow/deny”, com restrições por padrão em repositórios não confiáveis. (Alinhar a Workspace Trust e a mecanismos de hooks/rules). [76]

Proposta de interface do `openclaw` (exemplo)

```
# Inicializa artefatos no repositório para VS Code / Copilot (skills, agents, prompts, hooks)
openclaw init vscode --project .

# Inicializa artefatos para Antigravity (.agent/*)
openclaw init antigravity --project .

# Inicializa bundle/plugin para Cursor (estrutura .cursor-plugin + skills/rules/mcp)
openclaw build cursor-plugin --project . --out dist/openclaw-cursor-plugin.zip

# Executa um workflow (CLI), com saída JSON para consumo por IDE
openclaw run workflow fix-tests --project . --json

# Valida políticas e bloqueia comandos proibidos (usado por hooks)
openclaw policy check --stdin-json
```

PowerShell (Windows)

```
# Instalação (exemplo via winget, se publicado)
winget install Openclaw.Openclaw

# Init e execução
openclaw init vscode --project .
openclaw run workflow fix-tests --project . --json | ConvertFrom-Json
```

(Os comandos acima são um desenho proposto — a utilidade é dar uma "forma" operável a Skills/Agents/Rules/Workflows em múltiplos ambientes.)

Instalação e empacotamento em Windows, Linux e macOS

Windows (recomendado)
- Distribuir via **WinGet** (client do Windows Package Manager) para instalação/upgrade padronizados. [77]
- Assinatura de binários/instaladores: considerar `signtool` (processo e timestamp) em pipelines de release. [78]

macOS
- Distribuir via **Homebrew** (cask/formula) e/ou `.pkg`. [79]
- Para distribuição fora da App Store: considerar **Developer ID + notarization** (notary service produz "ticket" para Gatekeeper). [80]

Linux
- Distribuir via `.deb` / `.rpm` + repositórios e/ou via **Snap** (confinamento e updates automáticos são vantagens típicas do snap). [81]
- Se publicar repositório APT/YUM, considerar assinatura (metadados do repositório) como prática de segurança. [82]

Automação de releases
Ferramentas como GoReleaser ilustram um padrão popular para publicar binários multi-plataforma e integrar com package managers (Homebrew, Snap etc.), além de assinatura/notarização como parte do pipeline. [83]

Segurança, sandboxing, permissões e boas práticas

Ameaças relevantes para "agents em IDE"

- **Prompt injection via conteúdo do workspace** (arquivos maliciosos que "sequestram" instruções) e via navegação web. VS Code explicita que agentes podem rodar comandos, fazer web requests e puxar arquivos para contexto, e recomenda usar boundaries como Workspace Trust. [55]
- **Execução de comandos destrutivos** (ex.: `rm -rf`, `git push`, scripts desconhecidos).
- **Exfiltração** (rede: domínios não confiáveis; FS: diretórios sensíveis).
- **Supply chain**: extensões/plugins (VSIX/Vendor) com permissões amplas. VS Code e Cursor reforçam que marketplace e publisher confiança importam, e o próprio Workspace Trust não impede extensões maliciosas deliberadas. [84]

Controles por plataforma (o que aproveitar)

- **VS Code**:
- Workspace Trust e "Restricted Mode" desabilitam/limitam agentes e execução automática. [55]
- Hooks permitem bloquear tool calls e impor políticas (ex.: `PreToolUse` com `permissionDecision: deny`). [85]

- **Cursor**:

- Sandbox com controles de rede e FS e allowlists/denylists (inclusive enforcement por admins). [43]

- Plugins empacotam capacidades — recomendável exigir assinatura/verificação na cadeia de build do Openclaw (prática sugerida em governança de plugins). [86]

- **Antigravity**:

- Políticas de execução no terminal, denylist e possibilidade de exigir revisão; allowlist para browser agent. [36]

- **Codex**:

- Config por "trusted projects" (ignora camadas `.codex/` quando não confiável), rules para comandos fora do sandbox, e políticas de aprovação/sandbox em config. [87]

- **JetBrains**:

- MCP e ACP com logs e capacidade de controlar a conexão a servidores e agentes. [88]

Boas práticas de design de segurança para Openclaw

1. **Negação por padrão** para ações irreversíveis (delete em massa, push para prod, rotate keys) e exigência de aprovação explícita ("human-in-the-loop"). O modelo de hooks `PreToolUse` / `permissionDecision` do VS Code é um bom "shape" para padronizar respostas de policy engine. [85]
2. **Separar "planejar" de "executar"**: workflows devem preferir modo "plan" → revisão → "agent". Isso está alinhado a práticas de agentes em IDEs e aumenta previsibilidade. [89]
3. **Allowlist explícita de rede e FS**:
4. por workspace e em nível de organização (quando existir);
5. com logs de egress e auditoria. O Cursor descreve controles granulares nesse sentido, e o Antigravity recomenda allowlist para navegação. [90]
6. **Camadas de instrução**: manter `AGENTS.md` /instructions como "contrato de trabalho" do agente, com overrides locais não versionados (`*.override.*`) quando necessário (padrão do Codex). [91]

Testes, CI/CD, deploy e compatibilidade

Estratégia de testes por componente

Core do Openclaw (CLI/daemon)
- Unit tests + integração (execução de scripts, validação de políticas, parsing de manifests).
- "Golden tests" para geração de artefatos por plataforma (VS Code, Antigravity, Cursor, Codex, JetBrains ACP).

Integração VS Code (se houver extensão)
- O VS Code documenta testes de extensão rodando em "Extension Development Host" e recomenda `@vscode/test-cli` + `@vscode/test-electron`, com testes Mocha. [92]

- Para CI, há guias (ex.: rodar testes em Windows/macOS/Linux, headless Linux com `xvfb`) e possibilidade de publicação automatizada com `vsce` usando PAT via secrets. [93]

Integração JetBrains (se houver plugin)
- JetBrains recomenda o **Plugin Verifier** para compatibilidade binária entre builds de IDEs e plugins e descreve uso via tasks Gradle (`verifyPlugin`, `runPluginVerifier`) e integração em CI. [94]

Pipeline CI/CD recomendado (alto nível)

- Matrix OS (Windows/macOS/Linux) para: build do core, testes do core, packaging, assinatura (quando aplicável).
- Stage "Adapters": valida geração (`openclaw init ...`) e conformidade com schemas (Agent Skills, `.agent.md`, `.prompt.md`, `.rules`). [95]
- Stage "IDE integration":
- VS Code extension tests com `@vscode/test-cli`. [96]
- JetBrains plugin verifier. [94]
- Release: publicar binários e pacotes; opcionalmente integrar com package managers (Homebrew tap, Snap, WinGet). [97]

Migração e compatibilidade

Pontos críticos:

- **Convergência, mas não identidade, entre plataformas**: "Rules" significam coisas levemente diferentes (instruções vs política de execução). O Openclaw deve manter **um modelo canônico** e emitir artefatos específicos (ex.: hooks VS Code para enforcement; `.rules` Codex para comandos fora do sandbox; allowlists específicas para Antigravity/Cursor). [98]
- **Config layering e "trusted projects"**: Codex explicitamente ignora camadas de projeto se marcado como untrusted; VS Code usa Workspace Trust; Openclaw deve respeitar e refletir esses boundaries (não "forçar" habilitação automática). [99]
- **VS Code forks**: extensões e configurações podem funcionar, mas UI/Marketplace pode variar (Codex IDE extension explicitamente cita forks como Cursor). [100]
- **Evolução rápida de formatos "agentic"**: hooks/skills/agents podem mudar; o Openclaw deve versionar seu schema interno e oferecer migradores (`openclaw migrate`). O Agent Skills spec já define validações e campos; isso ajuda na estabilidade. [101]

Exemplo end-to-end para Openclaw e recomendações de implementação

A seguir, um exemplo "fim a fim" que implementa **Skill + Agent + Rule + Workflow** com foco em interoperabilidade.

Estrutura do repositório (sugestão)

```
.
├─ .github/
│  ├─ copilot-instructions.md           (Rule: sempre-on)
│  ├─ prompts/
```

```
│   │   └─ openclaw-fix-tests.prompt.md    (Workflow)
│   ├─ agents/
│   │   └─ openclaw.agent.md              (Agent)
│   ├─ hooks/
│   │   └─ security.json                  (Rule enforcement via hook)
│   └─ skills/
│       └─ openclaw-test-loop/
│           ├─ SKILL.md                   (Skill)
│           └─ scripts/
│               ├─ run_tests.sh
│               └─ run_tests.ps1
└─ scripts/
    └─ block-dangerous.sh                 (helper usado por hook)
```

Essa organização aproveita convenções do VS Code (instructions/prompt files/agents/hooks), do padrão Agent Skills (`SKILL.md`) e facilita o uso também em ferramentas que reconhecem `AGENTS.md` e skills; o Antigravity e o Codex têm conceitos equivalentes e podem consumir as skills/scripts com adaptação mínima de paths/scopes. [102]

Rule (VS Code): `.github/copilot-instructions.md`

```
# Regras do Openclaw (sempre aplicadas)

- Prefira mudanças pequenas e verificáveis.
- Sempre rode a suíte de testes apropriada após mudanças.
- Não adicione dependências em produção sem justificar e pedir confirmação.
- Nunca execute comandos destrutivos (rm -rf, formatar disco, etc.).
```

O VS Code detecta `.github/copilot-instructions.md` como instruções “always-on” para chat no workspace. [46]

Skill (Agent Skills padrão): `.github/skills/openclaw-test-loop/SKILL.md`

```
---
name: openclaw-test-loop
description: >
  Loop de correção orientado por testes. Use quando houver falhas de testes,
  regressões, ou quando o usuário pedir para "corrigir até os testes
passarem".
argument-hint: "[comando_de_teste_opcional]"
user-invokable: true
disable-model-invocation: false
---

# Openclaw Test Loop

## Objetivo
Executar testes, interpretar falhas, sugerir correções mínimas e reexecutar
```

```
até passar.

## Procedimento
1. Identifique o comando de testes do projeto (ou use o argumento do
usuário).
2. Execute o script apropriado em `./scripts/`:
   - Linux/macOS: `./scripts/run_tests.sh "<cmd>"`
   - Windows: `./scripts/run_tests.ps1 -Command "<cmd>"`
3. Se falhar, classifique falhas (unit/integration/lint) e proponha patch
mínimo.
4. Reexecute testes.
5. Ao final, gere um resumo com:
   - causas-raiz
   - arquivos alterados
   - evidências (logs/trechos) e próximos passos.
```

O formato e os campos do `SKILL.md` seguem o padrão Agent Skills (frontmatter YAML, campos `name`, `description`, opcionais e corpo em Markdown). [103]

Scripts (terminal) da Skill

Linux/macOS: `.github/skills/openclaw-test-loop/scripts/run_tests.sh`

```bash
#!/usr/bin/env bash
set -euo pipefail

CMD="${1:-}"
if [[ -z "${CMD}" ]]; then
  # fallback simples; em produção, detecte via package.json/pyproject/etc.
  CMD="pytest -q"
fi

echo "[openclaw] running tests: ${CMD}"
eval "${CMD}"
```

Windows: `.github/skills/openclaw-test-loop/scripts/run_tests.ps1`

```powershell
param(
  [string]$Command = ""
)

if ([string]::IsNullOrWhiteSpace($Command)) {
  $Command = "pytest -q"
}

Write-Host "[openclaw] running tests: $Command"
```

```
Invoke-Expression $Command
if ($LASTEXITCODE -ne 0) { exit $LASTEXITCODE }
```

Agent (VS Code custom agent): `.github/agents/openclaw.agent.md`

```
---
name: Openclaw
description: "Agente do Openclaw: correção guiada por testes + guardrails de
segurança."
tools:
  - "search/codebase"
  - "editFiles"
  - "terminal"
  - "openclaw/*"
  - "myMcpServer/*"
---

# Como eu trabalho
- Sempre começo com um plano curto (3–7 passos) e peço aprovação antes de
execuções arriscadas.
- Quando houver falha, uso a skill `openclaw-test-loop`.
- Mantenho mudanças mínimas e verificáveis.
- Se eu precisar de rede ou credenciais, eu pergunto explicitamente.
```

VS Code define custom agents como `.agent.md` com frontmatter suportando `tools`, e detecta agentes no diretório `.github/agents`. [104]

Workflow (VS Code prompt file): `.github/prompts/openclaw-fix-tests.prompt.md`

```
---
name: openclaw-fix-tests
description: "Executa o loop de testes e corrige falhas com mudanças
mínimas."
agent: "agent"
tools:
  - "terminal"
  - "search/codebase"
  - "editFiles"
---

Você vai corrigir o projeto até a suíte de testes passar.

Instruções:
1) Descubra o comando de testes (ou pergunte se não estiver claro).
2) Execute o loop baseado na skill openclaw-test-loop.
3) Após passar, gere:
   - resumo das correções
```

```
    - lista de arquivos alterados
    - comandos executados e resultados
```

Prompt files são `*.prompt.md` em `.github/prompts` e aparecem como slash commands. [47]

Rule enforcement (VS Code hook): `.github/hooks/security.json`

```
{
  "hooks": {
    "PreToolUse": [
      {
        "type": "command",
        "command": "./scripts/block-dangerous.sh",
        "timeoutSec": 5
      }
    ]
  }
}
```

Hooks no VS Code podem ser definidos em `.github/hooks/*.json` e o evento `PreToolUse` ocorre antes de invocar uma tool, permitindo bloquear/autorizar e até alterar input com base em stdout JSON do hook; exit codes determinam bloqueio. [105]

Implementação do bloqueio: `scripts/block-dangerous.sh`

```
#!/usr/bin/env bash
set -euo pipefail

# Exemplo simples: bloqueia se o input contiver padrões perigosos.
# Em produção, parseie o JSON do hook (tool_name/tool_input) e valide de
forma estruturada.

INPUT="$(cat || true)"

if echo "$INPUT" | grep -Eqi 'rm\s+-rf|mkfs|dd\s+if=|shutdown|reboot'; then
  cat <<'JSON'
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Comando destrutivo bloqueado pela política
do Openclaw"
  }
}
JSON
  exit 0
fi

# allow
```

```
cat <<'JSON'
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "allow"
  }
}
JSON
exit 0
```

Adaptações mínimas do mesmo exemplo para Antigravity e Codex (conceito)

Antigravity
- Colocar rules em `.agent/rules/`, workflows em `.agent/workflows/` e skills em `.agent/skills/` (ou global em `~/.gemini/...`), respeitando a estrutura `SKILL.md + scripts/`. [38]

Codex
- Reaproveitar `SKILL.md` e scripts; apontar paths no ambiente do usuário/projeto; usar `AGENTS.md` para instruções e `.rules` para controlar comandos fora do sandbox. [106]

Fluxo do workflow (visão mermaid)

```
stateDiagram-v2
  [*] --> Plan
  Plan --> Approve : plano pronto
  Approve --> Execute : aprovado
  Approve --> [*] : rejeitado

  Execute --> RunTests
  RunTests --> Patch : falhou
  Patch --> RunTests : reexecutar
  RunTests --> Report : passou
  Report --> [*]
```

Bibliotecas e ferramentas recomendadas para o Openclaw

- **Compatibilidade e formatos**
- Agent Skills (`SKILL.md`) como base — reduz custo de suporte multi-agente. [107]
- `AGENTS.md` como “contrato” de instruções por projeto e camadas. [91]
- MCP para ferramentas externas e reuso entre clientes. [74]
- **CLI / empacotamento**
- Go/Rust para binário único (facilita distribuição); automação de releases com pipeline estilo GoReleaser (ou equivalente). [108]
- Publicação: WinGet (Windows), Homebrew (macOS), `.deb/.rpm` e/ou Snap (Linux). [109]
- Assinatura: `signtool` (Windows) e notarização/Developer ID (macOS). [110]

- **Integração IDE**

- VS Code: começar com artefatos (instructions/prompts/skills/agents/hooks), evoluir para extensão somente quando necessário; empacotar `.vsix` com `vsce`. [111]
- JetBrains: preferir MCP/ACP antes de plugin; se plugin, seguir Plugin SDK e Plugin Verifier. [112]

Fontes prioritárias (ordem sugerida)

1. Documentação oficial do VS Code (customização, hooks, MCP/tools/agents, workspace trust, extensão e publicação/testes). [113]
2. Documentação oficial do Codex (CLI/IDE/config/AGENTS/rules/skills/MCP e orquestração via MCP server). [114]
3. Especificação Agent Skills (`agentskills.io`) e docs oficiais em PT-BR no GitHub (quando disponível). [72]
4. JetBrains AI (MCP/ACP e integração do Codex). [34]
5. Docs/artefatos oficiais do Cursor (changelog + plugin spec repos) e, quando necessário, fórum oficial. [115]
6. Google Codelabs do Antigravity (é o material mais completo e acessível para formatos/paths/políticas, dado que parte da doc pode depender de JS). [116]

---

[1] [20] [27] [72] [95] [101] [103] [107] https://agentskills.io/specification
https://agentskills.io/specification

[2] [65] [67] https://developers.openai.com/codex/guides/agents-sdk/
https://developers.openai.com/codex/guides/agents-sdk/

[3] [49] https://code.visualstudio.com/docs/copilot/customization/agent-skills
https://code.visualstudio.com/docs/copilot/customization/agent-skills

[4] [14] [15] [22] [35] [36] [37] [38] [39] [116] https://codelabs.developers.google.com/getting-started-google-antigravity
https://codelabs.developers.google.com/getting-started-google-antigravity

[5] [31] [40] [43] [45] [90] [115] Changelog · Cursor
https://cursor.com/changelog

[6] [9] [32] [56] [111] [113] https://code.visualstudio.com/docs/copilot/customization/overview
https://code.visualstudio.com/docs/copilot/customization/overview

[7] [18] [33] [58] [75] [114] https://developers.openai.com/codex/cli/
https://developers.openai.com/codex/cli/

[8] [68] https://blog.jetbrains.com/ai/2026/01/codex-in-jetbrains-ides/
https://blog.jetbrains.com/ai/2026/01/codex-in-jetbrains-ides/

[10] [13] [55] [76] [84] https://code.visualstudio.com/docs/editing/workspaces/workspace-trust
https://code.visualstudio.com/docs/editing/workspaces/workspace-trust

[11] [16] [26] [30] [47] [102] https://code.visualstudio.com/docs/copilot/customization/prompt-files
https://code.visualstudio.com/docs/copilot/customization/prompt-files

[12] [17] [19] [54] [57] [74] https://code.visualstudio.com/api/extension-guides/ai/mcp
https://code.visualstudio.com/api/extension-guides/ai/mcp

[21] [28] [46] [73] [98] https://code.visualstudio.com/docs/copilot/customization/custom-instructions
https://code.visualstudio.com/docs/copilot/customization/custom-instructions

23 61 91 https://developers.openai.com/codex/guides/agents-md/
https://developers.openai.com/codex/guides/agents-md/

24 29 48 104 https://code.visualstudio.com/docs/copilot/customization/custom-agents
https://code.visualstudio.com/docs/copilot/customization/custom-agents

25 69 https://www.jetbrains.com/help/ai-assistant/acp.html
https://www.jetbrains.com/help/ai-assistant/acp.html

34 70 71 88 112 https://www.jetbrains.com/help/ai-assistant/mcp.html
https://www.jetbrains.com/help/ai-assistant/mcp.html

41 GitHub - cursor/plugins: Cursor plugin specification and official plugins
https://github.com/cursor/plugins

42 https://forum.cursor.com/t/install-extension-by-vsix/114642
https://forum.cursor.com/t/install-extension-by-vsix/114642

44 86 https://cursor.com/blog/marketplace
https://cursor.com/blog/marketplace

50 85 105 https://code.visualstudio.com/docs/copilot/customization/hooks
https://code.visualstudio.com/docs/copilot/customization/hooks

51 https://code.visualstudio.com/api/references/vscode-api
https://code.visualstudio.com/api/references/vscode-api

52 https://code.visualstudio.com/api/extension-guides/ai/chat
https://code.visualstudio.com/api/extension-guides/ai/chat

53 https://code.visualstudio.com/api/extension-guides/ai/tools
https://code.visualstudio.com/api/extension-guides/ai/tools

59 100 https://developers.openai.com/codex/ide/
https://developers.openai.com/codex/ide/

60 66 87 99 https://developers.openai.com/codex/config-basic/
https://developers.openai.com/codex/config-basic/

62 https://developers.openai.com/codex/rules/
https://developers.openai.com/codex/rules/

63 106 https://developers.openai.com/codex/skills/
https://developers.openai.com/codex/skills/

64 https://developers.openai.com/codex/mcp/
https://developers.openai.com/codex/mcp/

77 109 https://learn.microsoft.com/en-us/windows/package-manager/winget/
https://learn.microsoft.com/en-us/windows/package-manager/winget/

78 110 https://learn.microsoft.com/en-us/dotnet/framework/tools/signtool-exe
https://learn.microsoft.com/en-us/dotnet/framework/tools/signtool-exe

79 https://docs.brew.sh/Installation
https://docs.brew.sh/Installation

80 https://developer.apple.com/documentation/security/customizing-the-notarization-workflow
https://developer.apple.com/documentation/security/customizing-the-notarization-workflow

81 https://www.debian.org/doc/manuals/debmake-doc/ch06.en.html
https://www.debian.org/doc/manuals/debmake-doc/ch06.en.html

82 https://www.debian.org/doc/manuals/securing-debian-manual/deb-pack-sign.en.html
https://www.debian.org/doc/manuals/securing-debian-manual/deb-pack-sign.en.html

83 97 108 https://goreleaser.com/install/
https://goreleaser.com/install/

89 Best practices for coding with agents · Cursor
https://cursor.com/blog/agent-best-practices

92 96 https://code.visualstudio.com/api/working-with-extensions/testing-extension
https://code.visualstudio.com/api/working-with-extensions/testing-extension

93 https://code.visualstudio.com/api/working-with-extensions/continuous-integration
https://code.visualstudio.com/api/working-with-extensions/continuous-integration

94 https://plugins.jetbrains.com/docs/intellij/verifying-plugin-compatibility.html
https://plugins.jetbrains.com/docs/intellij/verifying-plugin-compatibility.html