

4

STAGE 4

Open SWE: async coding agent

Open-source фреймворк LangChain Inc. для построения внутренних кодинг-агентов организации. Reference architecture поверх Deep Agents, pluggable sandboxes, триггеры из Slack/Linear/GitHub, draft PR автоматически. Воспроизводит паттерны Stripe Minions, Ramp Inspect, Coinbase Cloudbot -- но с открытым исходным кодом.

Open SWE: позиционирование

INFO

Не готовый продукт

Стартовый шаблон, не SaaS. Ops-работа: sandbox, модель, триггеры, промпты.

SUCCESS

Colleague, not copilot

Внутренний кодинг-агент, не IDE-assistant.
Slack / Linear / GitHub -> draft PR с тестами.

// meta: open-swe repo:1-12

github.com/langchain-ai/open-swe

stars: ~10k

commits: 971+

license: MIT

language: Python + TypeScript

announce: 08.2025

rewrite: 03.2026

blog.langchain.com/open-swe

INSTALLATION.md

CUSTOMIZATION.md

Архитектура: 7 слоев

Triggers

Slack / Linear / GitHub / Web UI

Validation

prompt + middleware (HITL, approval)

Orchestration

subagents + middleware

Context

AGENTS.md из репозитория

Tools

execute, fetch_url, linear_comment, slack_thread_reply

Sandbox

Modal / Daytona / Runloop / LangSmith

Harness

create_deep_agent (Deep Agents)

```
// open_swe/__init__.py:1-11
```

```
from open_swe.agent import create_agent
from open_swe.middleware import (
    check_message_queue_before_model,
    notify_step_limit_reached,
    open_pr_if_needed,
    ToolErrorMiddleware,
)
from open_swe.sandbox import (
    SandboxBackend,
    ModalBackend, DaytonaBackend,
)
```

create_deep_agent + backend

// examples/minimal_agent.py:1-11

```
from deepagents import create_deep_agent
from open_swe.sandbox import DaytonaBackend

> agent = create_deep_agent(
    model="anthropic:claude-opus-4-6",
    tools=[execute, fetch_url,
          linear_comment,
          slack_thread_reply],
    backend=DaytonaBackend(api_key="..."),
    middleware=[open_pr_if_needed],
)
```

INFO

Один harness

Март 2026: multi-agent
(Manager/Planner/Programmer/Reviewer)

SUCCESS

Upgrade path

Подтягиваешь
улучшения Deep
Agents бесплатно.
Subagents изолируют
контекст, middleware
дают orchestration.

AGENTS.md как system prompt

```
// open_swe/prompts.py:1-9
```

```
from pathlib import Path
```

```
def construct_system_prompt(  
    repo_dir, base_prompt  
):
```

```
    agents_md = Path(repo_dir) / "AGENTS.md"
```

```
    # note: ai prompt has 9 lines and fits ~6  
    extra_agents_md = read_text(repo_dir, agents_md) if  
    agents_md.exists() else ""
```

```
// AGENTS.md:1-5
```

```
# AGENTS.md (repo root)
```

- use uv, not pip
- run pytest before commit
- never push to main directly

INFO

Convention

Организационный паттерн.
Тот же файл читают Cursor,
Aider, Devin.

WARNING

Подводный

камень

Open SWE доверяет
AGENTS.md. Вредоносный
блок попадает в system
prompt.

Middleware: точки расширения

```
// examples/audit_middleware.py:1-13
```

```
from langchain.agents.middleware import (
    AgentMiddleware,
)
```

```
> class AuditMiddleware(AgentMiddleware):
>     def after_model(self, state, runtime):
>         runtime.logger.info(
>             f"step={state.get('step')}"
>         )
>         return state
```

```
def before_model(self, state, runtime):
    return state # inject reminder
```

INFO

Built-in middleware

- * check_message_queue
- * notify_step_limit
- * open_pr_if_needed
- * ...

SUCCESS

Типичные

кастомы

Approval gate перед PR, cost guard на длину контекста, redaction секретов в логах.

Sandbox-провайдеры

Provider	Setup	Pricing model	Persistent state	Best for
ModalBackend	token_id + token_secret	per-second + GB-s	yes (volume)	Python-first, GPU-доступный
DaytonaBackend	api_key	per-session	yes (volume)	default в README
RunloopBackend	api_key	per-second	yes (volume)	dev-цикл, snapshot/restart
LangSmithBackend	LANGSMITH_API_KEY	через LangSmith	через LS	уже платите за LangSmith

Sandbox: imports + custom

```
// open_swe/sandbox/__init__.py:1-12
```

```
from open_swe.sandbox import (
    ModalBackend,
    DaytonaBackend,
    RunloopBackend,
    LangSmithBackend,
)

# Modal (Python-first, GPU)
backend = ModalBackend(
    token_id="..."
    token_secret="..."
)
```

```
// examples/my_internal_backend.py:1-16
```

```
# свой backend: devbox-пул
from open_swe.sandbox import (
    SandboxBackend,
)

class MyInternalBackend(
    SandboxBackend
):
    def __init__(self, conn):
        self.conn = conn

>
> def execute(self, cmd):
>     return self._run(cmd)
>
>
> def read_file(self, p):
>     return self._fetch(p)
```

// note: snippet has 16 lines, card fits ~15

Установка (1/2): шаги 1-2

// INSTALLATION.md: step 1:1-5

```
# 1. prerequisites
python --version # >= 3.11
node --version   # >= 20
uv --version     # или pip
docker --version # для dev
```

// INSTALLATION.md: step 2:1-5

```
# 2. clone + install
git clone https://github.com/
  langchain-ai/open-swe.git
cd open-swe
uv sync # или pip install -e .
```

INFO

He SaaS

Open SWE -- не готовый сервис. После клонирования нужно поднять backend, UI, sandbox. [GitHub App](#).

WARNING

ENV-файл

ср .env.example .env, затем заполните:

- GITHUB_APP_ID
- LANGSMITH_API_KEY
- DAYTONA_API_KEY

Установка (2/2): шаги 3-5

// INSTALLATION.md: steps 3-5:1-13

3. backend (FastAPI)

```
uv run apps/open-swe/main.py
```

```
# -> :8000
```

4. UI (TanStack Start + Vite)

```
cd apps/open-swe-ui
```

```
pnpm install && pnpm dev
```

```
# -> :3000
```

5. smoke-test

```
curl -X POST :8000/webhooks/slack \
```

```
-d '{"text": "@open-swe hello"}'
```

```
# -> {"thread_id": "..."}'
```

S U C C E S S

Готово

Backend :8000

UI :3000

Webhook

8000/webhooks/slack/*

I N F O

Production

Локальный запуск --
только dev. Для prod:
docker compose, k8s,
managed Postgres, Vault,
TLS.

GitHub App: manifest

```
// config/github-app-manifest.yaml:1-13

# github-app-manifest.yaml
name: open-swe-internal
> url: https://open-swe.example.com
> hook_attributes:
>   url: https://open-swe.example.com/
>   webhooks/github
>   events:
>     - issue_comment
>     - pull_request
>       - pull_request_review
default_permissions:
  contents: write
  pull_requests: write
```

INFO

Создание App

1. github.com/settings/apps/new

WARNING

Permissions

contents:write -- ветки
pull_requests:write -- draft PR
issues:write -- комментарии
metadata:read -- repo info

LangSmith: setup

// .env:1-10

```
# .env (НЕ коммитить)
LANGSMITH_TRACING=true
LANGSMITH_ENDPOINT=https://
  api.smith.langchain.com
LANGSMITH_API_KEY=lsv2_...
LANGSMITH_PROJECT=open-swe-prod
LANGSMITH_SNAPSHOT=true

# для sandbox-прокси:
LANGSMITH_SANDBOX_BACKEND=true
```

INFO

Зачем snapshot

Каждый thread = checkpoint.
Follow-up из всех каналов
полхватывают состояние по

SUCCESS

API keys

- * Personal --
smith.langchain.com
- * Org-level -- shared tracing
- * Service -- для CI / batch
Rotate каждые 90 дней.

Triggers overview

@

Slack

В любом thread канала.
Поддерживает синтаксис
repo:owner/name.

```
@open-swe fix the auth bug  
repo:acme/api
```

#

Linear

В комментарии к issue.
Привязывает thread к
Linear-тикету.

```
@openswe implement the  
acceptance criteria
```

PR

GitHub

В PR-комментарии для
авто-ответа на review-
комментарии.

```
@openswe address  
the review comments
```

Triggers: thread_id routing

```
// open_swe/triggers/router.py:1-11

# open_swe/triggers/router.py
> def thread_id_for(source, ref, comment_id):
>     # Deterministic thread_id:
>     # все follow-up попадают в один run.
>     if source == "slack":
>         return f"slack:{ref}"
>     if source == "linear":
>         return f"linear:{ref}"
>     if source == "github":
>         return f"github:{ref}:{comment_id}"
>         raise ValueError(source)
```

INFO

Routing

thread_id из (source, ref).
Если run бежит -- новые сообщения ждут в очереди.

WARNING

Concurrency

Один thread = один run.
Параллельность через отдельный thread_id (например, отдельный Slack).

Webhook endpoints

```
// apps/open-swe/webhooks.py:1-15
```

```
# apps/open-swe/webhooks.py
from fastapi import APIRouter, Request
from open_swe.triggers.router import (
    thread_id_for,
)
```

```
router = APIRouter()
```

```
> @router.post("/webhooks/slack")
> async def slack_webhook(req: Request):
>     payload = await req.json()
>     if "@open-swe" not in payload["text"]:
>         return {"ok": True}
>     await enqueue_run(thread_id_for(
>         "slack", payload["channel"]))
```

SUCCESS

Idempotency

Webhook вычисляет thread_id и проверяет наличие run. Если есть

WARNING

Signature

Все handlers обязаны проверять X-Signature от Slack / Linear / GitHub. Не доверяйте без verify.

Dashboard UI

```
// apps/open-swe-ui/tree.sh:1-13
```

```
# apps/open-swe-ui/  
# TanStack Start + Vite + React 19
```

```
src/routes/  
  __root.tsx  
  index.tsx      # thread list  
  thread.$id.tsx # chat  
  settings.tsx  
src/components/  
  ChatMessage.tsx  
  DiffViewer.tsx  
src/lib/  
  client.ts      # OpenSWEClient
```

INFO

Только UI

Dashboard -- presentation layer. Агентская логика в Python backend. UI через

SUCCESS

Features

- * GitHub OAuth login
- * thread list + filters
- * chat with streaming
- * diff viewer для PR
- * per-user settings

Per-user настройки

```
// open_swe/settings.py:1-15
```

```
# settings schema (per-user)
USER_DEFAULTS = {
>   "model": "anthropic:claude-opus-4-6",
>   "profile": "balanced",
>   "max_steps": 80,
>   "auto_open_pr": False,
>   "require_approval": True,
    "extra_repos": ["acme/internal-tools"],
}

> # team defaults (allowlist)
> TEAM_DEFAULTS = {
    "sandbox_backend": "ModalBackend",
    "allowed_repos": ["acme/*"],
}
```

INFO

User mappings

GitHub login -> Slack user -> Linear user. Один человек получает свой профиль во

WARNING

Precedence

user > team > global. Per-user override работает для всех полей, кроме `allowed_repos` -- это team-level allowlist.

6 точек кастомизации

1

Sandbox provider

```
backend=ModalBackend(...)
```

2

Model

```
init_chat_model("anthropic:...")
```

3

Tools

```
tools=[execute, fetch_url, ...]
```

4

Triggers

```
router.add_route("/my",  
my_handler)
```

5

System prompt

```
construct_system_prompt(repo_dir, ...)
```

6

Middleware

```
middleware=[AuditLogger(), ...  
]
```

Three graphs

agent graph

Основной deep-agent: planner + coder + tools + subagents.

START -> plan -> execute -> review -> open_pr -> END

reviewer graph

CI-стиль: lint, type-check, tests, security-scan. Без LLM-агента.

START -> run_ci -> parse -> report -> END

analyzer graph

Анализ ретроспективы: что заняло больше шагов, где loop, где cost spikes.

START -> load_trace -> aggregate -> summarize -> END

Observability

```
// open_swe/observability.py:1-15

# observability.py
from datadog import statsd
> from langchain_core.tracers import (
>   LangChainTracer,
> )

tracer = LangChainTracer(
>   project_name="open-swe-prod",
> )
>

def record_step(thread_id, duration_s):
    statsd.histogram(
        "open_swe.step.duration_s",
        duration_s, tags=[f"t:{thread_id}"],
    )
```

INFO

LangSmith

Trace каждого run: model calls, tool calls, retries.
Replay, debug, manual evaluation

SUCCESS

Datadog

P95 latency, tokens per run, error rate, sandbox spend.
Alerts на cost spikes и длинные loops.

Production: prod-ready

SUCCESS

Infra

- + docker-compose / k8s
- + managed Postgres
- + Vault / sealed-secrets
- + TLS + reverse proxy

INFO

Ops

- + LangSmith prod-project
- + Datadog dashboards + SLO
- + sandbox cost budget
- + audit log всех PR

WARNING

Самый частый пропуск

"Trust the LLM" внутри sandbox. Без надлежащей изоляции (seccomp, network policy, ephemeral FS) агент может сделать `rm -rf` или `curl` внутренних сервисов. Изоляция важнее prompts.

TypeScript: только UI

```
// apps/open-swe-ui/src/lib/client.ts:1-13
```

```
// apps/open-swe-ui/src/lib/client.ts
import { OpenSWEClient } from
"@openswe/client";

const client = new OpenSWEClient({
  langsmithApiKey: process.env.
    LANGSMITH_API_KEY!,
});

await client.invoke({
  threadId: "issue-123",
  prompt: "Fix the bug",
});
```

PROS

- + UI: полностью TS
- + TanStack Start + Vite
- + strict TS config
- + streaming SDK

CONS

- Нет TS для backend
- Агент -- Python only
- Webhook handlers только Py
- SDK -- thin wrapper

Open SWE vs Claude Code / Devin

PROS

- + MIT license, форкайте
- + Pluggable sandbox
- + Triggers: Slack/Linear/GH
- + AGENTS.md convention
- + Subagents + middleware
- + Built on Deep Agents

CONS

- Не finished product
- Sandbox = платные аккаунты
- OAuth setup нужен
- "Trust the LLM" модель
- Prod deployment сложный
- Доки быстро устаревают

VS Claude Code / Devin

Audience	IDE	SaaS	org
License	proprietary	proprietary	MIT
Trigger	manual	Web UI	Slack/Lin/GH
Sandbox	local+cloud	remote	pluggable
Customize	config	no	full src
Runs in	IDE/term	cloud	your infra

Roadmap: 2026-2027

Q2 2026

Stable v1

- > deep-agent harness заморожен
- > API стабилизирован
- > semver commitment

Q3 2026

Multi-tenant

- > per-org quota + billing
- > team-level audit log
- > rbac на sandbox providers

Q4 2026

More triggers

- > Jira / Azure DevOps
- > Sentry для авто-fix багов
- > PagerDuty для incident triage

2027

SDK + Marketplace

- > public SDK (Python + TS)
- > plugin marketplace
- > shared subagent library