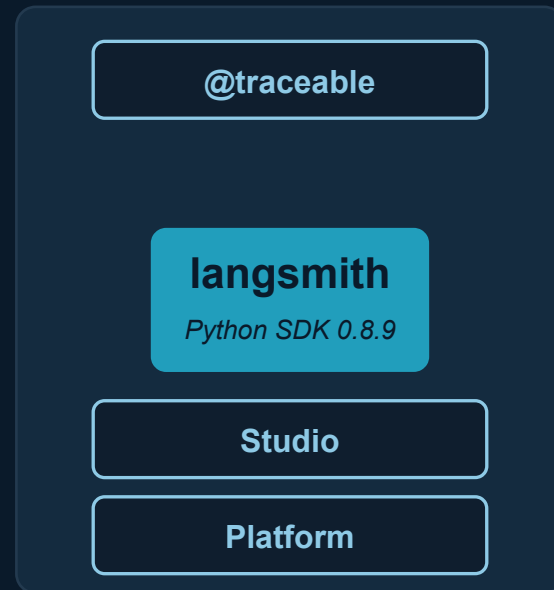


Экосистема

LangSmith, Studio, deployment

Сервисы вокруг 4 основных ступеней: observability и eval (LangSmith SDK), визуальный дебаг графов (LangGraph Studio), production deployment (LangGraph Platform), self-hosted и TypeScript-клиенты. Все, что превращает прототип в production-grade систему.



observability | visual debug | deploy

Три столпа поверх LangChain/LangGraph

LangSmith -- SaaS-платформа (с self-hosted вариантом) для production-наблюдения и оценки LLM-приложений. Три ключевые возможности покрывают весь цикл: отладка в dev, метрики в prod, оценка качества на датасетах.

Tracing

observability

Каждый вызов Runnable, графа, tool логируется как Run с input/output, latency, token-cost. Дерево вызовов -- в UI, и через SDK.

Datasets

eval sets

Версионизуемые наборы примеров (input + expected output). Используются для off-line eval, regression-тестов, few-shot examples.

Evaluators

quality scoring

LLM-as-judge, heuristic, human -- на выбор. Прогоняются на датасете, результаты привязываются к эксперименту (commit, prompt version).

Cloud: smith.langchain.com | SDK: `langsmith==0.8.9` | Self-hosted v0.9 (changelog 21.01.2025)

pip install и 3 переменные окружения

```
// shell/install_langsmith.sh:1-7
```

```
# Standalone SDK (если не используете langchain)
pip install langsmith
```

```
# С langchain/LangGraph auto-tracing работает
# автоматически при env-переменных -- отдельно
# ставить SDK необязательно.
pip install langchain langgraph
```

```
// shell/env.sh:1-11
```

```
# 1. Tracing on/off (default: true если есть API key)
export LANGSMITH_TRACING=true
```

```
# 2. API key: https://smith.langchain.com/settings
export LANGSMITH_API_KEY="lsv2_pt_..."
```

```
# 3. Project name (default: "default")
export LANGSMITH_PROJECT="my-agent-dev"
```

```
# Опционально: self-hosted endpoint
# export LANGSMITH_ENDPOINT=...
```

INFO

Zero-config для LangChain/LangGraph

Если `LANGSMITH_API_KEY` задан, `chain.invoke/graph.invoke/agent.stream` пишут `trace` автоматически -- без оберток и `monkey-patch`.

source: [docs.smith.langchain.com Observability > Set up tracing](https://docs.smith.langchain.com/Observability/Set-up-tracing)

@traceable -- декоратор для любой функции

```
from langsmith import traceable

# Декоратор превращает функцию в traced Run.
# Все аргументы и return попадают в trace автоматически.
@traceable(name="retrieve_docs", run_type="retriever")
def retrieve(query: str, k: int = 4) -> list[str]:
    return vector_store.similarity_search(query, k=k)

@traceable(name="generate_answer", run_type="chain")
def generate_answer(query: str) -> str:
    docs = retrieve(query)          # nested Run
    context = "\n".join(docs)
    return llm.invoke(f"Q: {query}\nCtx: {context}")

# Дерево: generate_answer -> retrieve_docs -> ChatModel
print(generate_answer("What is LCEL?"))
```

S U C C E S S

Вложенные вызовы автоматически становятся дочерними Run-ами в дереве трассировки.

RunTree -- ручной контроль над деревом

```
from langsmith.run_trees import RunTree

# RunTree -- explicit handle: создается руками, передается
# в код, потом post() отправляет в LangSmith. Нужно когда
# @traceable не подходит (динамич. дерево, не-Python, metadata).
parent = RunTree(
    name="agent_turn",
    run_type="chain",
    inputs={"q": "Who invented Python?"},
    project_name="my-agent-dev",
)
try:
    answer = my_agent(question, run_tree=parent)
    parent.end(outputs={"answer": answer})
except Exception as e:
    parent.end(error=str(e))      # ошибка логируется
    raise
finally:
    parent.post()                # flush в LangSmith
```

// note: snippet has 19 lines, card fits ~17

WARNING

Не забудьте `parent.post()` -- иначе trace не отправится. Дочерние через `parent.create_child()`

Auto-tracing LangChain + метаданные

```
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate

prompt = ChatPromptTemplate.from_template("Tell a joke about {topic}")
model = ChatOpenAI(model="gpt-4o-mini")
chain = prompt | model

# auto-tracing: invoke/batch/stream автоматически создают Run
# с input, output, token usage, latency.
result = chain.invoke(
    {"topic": "databases"},
    > config={
    >     "run_name": "joke_chain", # имя в UI
    >     "tags": ["prod", "experiment-v3"], # фильтр в UI
    >     "metadata": { # любые поля
    >         "user_id": "u_123",
    >         "request_id": "req_abc",
    < note: snippet has 20 lines, card fits ~17
    > },
    }
```

S U C C E S S

tags и metadata -- способ группировать и фильтровать trace-ы в UI по user/session/feature.

create_dataset -- набор примеров под eval

```
from langsmith import Client

client = Client()

# 1. Создаем датасет (или достаём существующий по имени)
dataset = client.create_dataset(
    dataset_name="rag-qa-eval-v1",
    description="RAG golden set",
)

# 2. Добавляем примеры пачкой: inputs + reference outputs
client.create_examples(
    dataset_id=dataset.id,
    inputs=[{"q": "Что такое LCEL?"}, {"q": "Что такое HITL?"}],
    outputs=[{"a": "LangChain Expression Language"},
              {"a": "Human in the loop opens interrupt"}],
    // note: snippet has 20 lines, and fits in 16
)
```

INFO

В UI датасеты можно создавать из CSV/JSONL через web -- API нужен для CI и автообновления.

source: [docs.smith.langchain.com Evaluation > Datasets](https://docs.smith.langchain.com/Evaluation%20Datasets)

run_on_dataset + evaluator-ы

```
from langsmith import Client
from langsmith.schemas import Example, Run

client = Client()

# 1. Target -- что прогоняем (chain, graph, agent, callable)
def target(inputs: dict) -> dict:
    return {"answer": my_chain.invoke(inputs["question"])}

# 2. Evaluator -- scoring function
def answer_match(run: Run, example: Example) -> dict:
    score = 1.0 if example.outputs["answer"] in run.outputs["answer"] else 0.0
    return {"key": "answer_match", "score": score}

# 3. Прогон: target на датасете + scoring
client.run_on_dataset(
    // Note: snippet has 20 lines, card fits ~15
    dataset_name="rag-qa-eval-v1",
    llm_or_chain_factory=target
```

SUCCESS

Готовые evaluator-ы: `llm_as_judge`, `exact_match`, `embedding_distance`, `cot_qa`.

Визуальный дебаг графа

Studio -- это desktop/web IDE для LangGraph. Показывает граф как граф, а не как свалку логов. Запускается локально через `langgraph dev`, либо хостится на LangGraph Platform.

GRAPH CANVAS



RUN INSPECTOR

0ms	__start__
+12ms	router
+840ms	agent (LLM call)
+1.2s	tools[search]
+1.4s	agent (LLM call)
+2.1s	END

LangGraph Platform: deploy + invoke

```
# 1. Конфиг: langgraph.json в корне репо
cat > langgraph.json <<'JSON'
{
  "graphs": {"agent": "./agent.py:graph"},
  "env": "./.env"
}
JSON
```

```
# 2. Deploy one-shot
langgraph deploy --name my-agent-prod
```

```
from langgraph_sdk import get_client

client = get_client(url=
    "https://my-agent-prod.us.langgraph.app"
)

# async API: thread + run
thread = await client.threads.create()
run = await client.runs.create(
    thread["thread_id"], "agent", input={"q": "hi"}
)
```

INFO

LangGraph Platform

managed-хостинг для графа: build из langgraph.json, deploy через CLI, получаете HTTPS endpoint + Studio UI + threads + cron + webhooks.

source: langchain-ai.github.io/langgraph/concepts/langgraph_platform/

Self-hosted vs Cloud: что выбрать

LangSmith и LangGraph Platform существуют в двух режимах: managed Cloud (быстрый старт, оплата по usage) и Self-Hosted (ваш K8s/VM, ваши данные остаются внутри периметра). Актуальная self-hosted версия -- v0.9.

Cloud

smith.langchain.com + managed platform

- + Zero-setup: API key и заводите trace-ы
Managed Postgres, Redis, scaling
Studio UI включен в подписку
Latest features сразу
- Данные уходят в чужой VPC (US-region)
Pay-per-trace: cost растет с нагрузкой
Vendor lock на retention policy

Self-Hosted

Docker/Helm chart, v0.9

- + Данные внутри своего VPC/compliance
Flat cost: предсказуемо для prod
Полный контроль над retention
Air-gapped окружения поддерживаются
- Ops: K8s, Postgres, Redis, ClickHouse
Обновления руками (breaking changes)
Studio UI = отдельный пакет
Не все beta-фичи доступны сразу

Рекомендация: Cloud для prototype и команд до 5 инженеров; self-hosted при compliance-требованиях и > 100M trace-ов в месяц.

source: changelog.langchain.com/announcements/langsmith-self-hosted-v0-9

TypeScript SDK и плюсы/минусы

```
import { Client, traceable } from "langsmith";

const client = new Client();

// @traceable decorator -- точно как в Python
const retrieve = traceable(
  async function retrieve(query: string) {
    return vectorStore.similaritySearch(query, 4);
  },
  { name: "retrieve_docs", runType: "retriever" }
);

// run_on_dataset для eval -- тоже есть
await client.runOnDataset(
  "rag-qa-eval-v1", target,
  { evaluators: [answerMatch] }
);
```

PROS

- + Auto-tracing из коробки
- + Eval = CI/CD для промптов
- + Platform = deploy за минуты
- + Self-hosted опция

CONS

- LangSmith SDK 0.x
- Cloud растёт в цене
- Self-hosted требует K8s