

# ATP Platform

## Краткое руководство

Agent Test Platform — фреймворк-независимое тестирование AI-агентов

Установка, тест-сюиты, Docker-агенты, контракт протокола

---

*Принцип: Агент = чёрный ящик с контрактом (вход → выход + события через ATP Protocol)*

# 1. Установка и настройка

## Требования

- Python 3.12+
- Пакетный менеджер uv (<https://astral.sh/uv>)
- Docker или Podman (для контейнерных агентов)

## Установка uv

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

## Клонирование и установка

```
git clone <your-repo-url>
cd atp-platform
uv sync
```

## Проверка

```
uv run atp version
uv run pytest tests/ -v --co -q # список тестов
```

### ПРИМЕЧАНИЕ:

Используйте ТОЛЬКО uv, никогда pip.  
Установка пакетов: uv add <package>  
Запуск инструментов: uv run <tool>

## 2. Создание тест-сьюта

Тест-сьюты — это YAML-файлы, которые декларативно описывают что тестировать, как запускать агентов и как оценивать результаты. Ниже минимальный пример с 2 тестами.

### Пример: my\_tests.yaml

```
test_suite: "my-first-suite"
version: "1.0"
description: "Базовые тесты файловых операций"

defaults:
  timeout_seconds: 30
  constraints:
    max_steps: 5
    max_tokens: 50000
  scoring:
    quality_weight: 0.3
    completeness_weight: 0.5
    efficiency_weight: 0.2

agents:
  - name: "my-agent"
    type: "cli"           # или: http, container
    config:
      command: "python"
      args: ["agent.py"]
```

```
tests:
  - id: "test-001"
    name: "Создание файла"
    description: "Агент создаёт hello.txt"
    tags: ["file", "smoke"]
    task:
      description: >
        Создать файл 'hello.txt'
        с содержимым 'Hello, ATP!'
      expected_artifacts: ["hello.txt"]
    assertions:
      - type: "artifact_exists"
        config: { path: "hello.txt" }
      - type: "contains"
        config:
          path: "hello.txt"
          pattern: "Hello, ATP!"
```

```
- id: "test-002"
  name: "Обработка CSV"
  description: "Агент обрабатывает CSV-данные"
  tags: ["data", "csv"]
  task:
    description: >
      Прочитать data.csv, вычислить среднее
      по столбцу 'price', записать
      результат в summary.json
  input_data:
    csv_path: "data.csv"
  expected_artifacts: ["summary.json"]
  constraints:
    max_steps: 10
    timeout_seconds: 60
  assertions:
    - type: "artifact_exists"
      config: { path: "summary.json" }
    - type: "llm_eval"
      config:
        criteria: "completeness"
        threshold: 0.8
```

## Валидация и запуск

```
# Проверка синтаксиса
uv run atp validate --suite=my_tests.yaml

# Запуск тестов
uv run atp test my_tests.yaml --adapter=cli -v

# Только smoke-тесты
uv run atp test my_tests.yaml --tags=smoke

# Вывод в JSON
uv run atp test my_tests.yaml --output=json \
  --output-file=results.json
```

### 3. Контракт ATP Protocol

ATP Protocol определяет строгий контракт между платформой и агентами. Каждый агент принимает ATPRequest (JSON через stdin) и возвращает ATPResponse (JSON через stdout). События (опционально) отправляются через stderr в формате JSONL.

#### ATPRequest (stdin)

```
{
  "version": "1.0",
  "task_id": "test-001",
  "task": {
    "description": "Создать hello.txt...",
    "input_data": {"key": "value"},
    "expected_artifacts": ["hello.txt"]
  },
  "constraints": {
    "max_steps": 5,
    "timeout_seconds": 30,
    "max_tokens": 50000
  },
  "context": {
    "workspace_path": "/tmp/workspace",
    "environment": {"VAR": "value"}
  }
}
```

#### ATPResponse (stdout)

```
{
  "version": "1.0",
  "task_id": "test-001",
  "status": "completed",
  "artifacts": [{
    "type": "file",
    "path": "hello.txt",
    "content": "Hello, ATP!",
    "size_bytes": 11
  }],
  "metrics": {
    "total_steps": 2,
    "tool_calls": 1,
    "wall_time_seconds": 1.5
  }
}
```

#### ATPEvent (stderr, опционально)

```
{ "version": "1.0", "task_id": "test-001", "sequence": 0,
  "event_type": "tool_call",
  "payload": { "tool": "create_file", "status": "started" } }
{ "version": "1.0", "task_id": "test-001", "sequence": 1,
  "event_type": "progress",
  "payload": { "message": "Done", "percentage": 100 } }
```

**ПРИМЕЧАНИЕ:**

Статусы: completed, failed, timeout, cancelled, partial

Типы событий: tool\_call, llm\_request, reasoning, error, progress

События — JSONL в stderr (один JSON-объект на строку)

## 4. Создание Docker-агента

Docker-агент — это образ контейнера, реализующий ATP Protocol. Контейнер читает ATPRequest из stdin, выполняет работу и возвращает ATPResponse в stdout.

### agent.py — Минимальный агент

```
#!/usr/bin/env python3
import json, sys, os

# 1. Читаем запрос из stdin
request = json.load(sys.stdin)
task_id = request["task_id"]
description = request["task"]["description"]
workspace = request.get("context", {}).get(
    "workspace_path", "/tmp/workspace")
os.makedirs(workspace, exist_ok=True)

# 2. Выполняем работу
path = os.path.join(workspace, "hello.txt")
with open(path, "w") as f:
    f.write("Hello, ATP!")

# 3. Ответ в stdout
response = {
    "version": "1.0",
    "task_id": task_id,
    "status": "completed",
    "artifacts": [{
        "type": "file",
        "path": "hello.txt",
        "content": "Hello, ATP!",
        "size_bytes": 11
    }],
    "metrics": {"total_steps": 1}
}
json.dump(response, sys.stdout)
```

### Dockerfile

```
FROM python:3.12-slim
WORKDIR /app
COPY agent.py .
ENTRYPOINT ["python", "agent.py"]
```

### Сборка и запуск

```
# Собрать образ
docker build -t my-agent:latest .

# Запустить тесты
uv run atp test my_tests.yaml \
  --adapter=container \
  --adapter-config='image=my-agent:latest'
```

## Опции контейнерного адаптера

```
adapter: container
adapter_config:
  image: "my-agent:latest"      # обязательно
  runtime: "auto"              # auto|docker|podman
  resources:
    memory: "2g"
    cpu: "1"
  network: "none"              # none|bridge|host
  environment:
    API_KEY: "${API_KEY}"
  auto_remove: true
  cap_drop: ["ALL"]           # безопасность
```



## 5. Чек-лист агента

Перед деплоем агента проверьте:

- Читает ATPRequest (JSON) из stdin целиком
- Возвращает валидный ATPResponse (JSON) в stdout
- Устанавливает "version": "1.0" в ответе
- Возвращает правильный "task\_id"
- Использует допустимый status: completed | failed | timeout
- Перечисляет все артефакты в массиве artifacts
- Сообщает метрики (steps, tokens, wall\_time)
- Корректно обрабатывает таймаут (возвращает partial)
- НЕ пишет ничего лишнего в stdout (только JSON-ответ)
- События (опционально) идут в stderr как JSONL
- Docker: ENTRYPOINT запускает скрипт агента
- Docker: Никаких интерактивных промптов (полностью автономный)

## Основные команды CLI

```
uv run atp validate --suite=my_tests.yaml
uv run atp test my_tests.yaml --adapter=cli -v
uv run atp test my_tests.yaml --adapter=container \
  --adapter-config='image=my-agent:latest'
uv run atp list my_tests.yaml
uv run atp list-agents
uv run atp dashboard
```

## Типы проверок (assertions)

artifact_exists	- проверка создания файла
contains	- поиск паттерна в содержимом
behavior	- no_errors, efficient_tool_use
llm_eval	- семантическая оценка
code_exec	- запуск сгенерированного кода
security	- PII/секреты/инъекции
performance	- латенси, пропускная способность