

# ATP Platform

## Quickstart Guide

Agent Test Platform — framework-agnostic testing for AI agents

Installation, Test Suites, Docker Agents, Protocol Contract

---

*Key principle: Agent = black box with a contract (input → output + events via ATP Protocol)*

# 1. Installation & Setup

## Prerequisites

- Python 3.12+
- uv package manager (<https://astral.sh/uv>)
- Docker or Podman (for container agents)

## Install uv

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

## Clone and install

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

## Verify installation

```
uv run atp version
uv run pytest tests/ -v --co -q # list all tests
```

**NOTE:**

ONLY use uv for package management, never pip.

Install packages: uv add <package>

Run tools: uv run <tool>

## 2. Creating a Test Suite

Test suites are YAML files that declaratively define what to test, how to run agents, and how to evaluate results. Below is a minimal example with 2 tests.

### Example: `my_tests.yaml`

```
test_suite: "my-first-suite"
version: "1.0"
description: "Basic file operations test suite"

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"                # or: http, container
    config:
      command: "python"
      args: ["agent.py"]

tests:
  - id: "test-001"
    name: "Create a text file"
    description: "Agent creates hello.txt"
    tags: ["file", "smoke"]
    task:
      description: >
        Create a file named 'hello.txt'
        with content '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: "Parse CSV data"
  description: "Agent processes CSV input"
  tags: ["data", "csv"]
  task:
    description: >
      Read data.csv, compute the average of
      the 'price' column, write result to
      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
```

## Validate & run

```
# Validate syntax
uv run atp validate --suite=my_tests.yaml

# Run tests
uv run atp test my_tests.yaml --adapter=cli -v

# Run only smoke tests
uv run atp test my_tests.yaml --tags=smoke

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

### 3. ATP Protocol Contract

The ATP Protocol defines a strict contract between the platform and agents. Every agent must accept an ATPRequest (JSON via stdin) and return an ATPResponse (JSON via stdout). Events are optional and sent via stderr as JSONL.

#### ATPRequest (stdin)

```
{
  "version": "1.0",
  "task_id": "test-001",
  "task": {
    "description": "Create hello.txt with content...",
    "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,
    "total_tokens": 150
  }
}
```

#### ATPEvent (stderr, optional)

```
{"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}}
```

**NOTE:**

Status values: completed, failed, timeout, cancelled, partial

Event types: tool\_call, llm\_request, reasoning, error, progress

Events are JSONL on stderr (one JSON object per line)

## 4. Creating a Docker Agent

A Docker agent is a container image that implements the ATP Protocol. The container reads ATPRequest from stdin, does work, and writes ATPResponse to stdout.

### agent.py — Minimal agent

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

# 1. Read request from 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. Do work (example: create a file)
path = os.path.join(workspace, "hello.txt")
with open(path, "w") as f:
    f.write("Hello, ATP!")

# 3. Write response to 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, "tool_calls": 1}
}
json.dump(response, sys.stdout)
```

### Dockerfile

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

### Build & run

```
# Build the image
docker build -t my-agent:latest .

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

## Container adapter options

```
adapter: container
adapter_config:
  image: "my-agent:latest"      # required
  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"]           # security
```



## 5. Agent Checklist

Before deploying your agent, verify:

- Reads full ATPRequest JSON from stdin
- Returns valid ATPResponse JSON to stdout
- Sets "version": "1.0" in response
- Echoes back the correct "task\_id"
- Uses valid status: completed | failed | timeout
- Lists all produced artifacts in the artifacts array
- Reports metrics (steps, tokens, wall\_time)
- Handles timeout gracefully (returns partial status)
- Does NOT write anything else to stdout (only the JSON response)
- Events (optional) go to stderr as JSONL, not stdout
- Docker: ENTRYPOINT runs the agent script
- Docker: No interactive prompts (fully non-interactive)

---

### Common CLI commands

```
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
```

### Assertion types reference

artifact_exists	- check file was created
contains	- pattern match in file content
behavior	- no_errors, efficient_tool_use
llm_eval	- semantic eval (completeness, factual)
code_exec	- run generated code
security	- PII/secrets/injection detection
performance	- latency, throughput checks