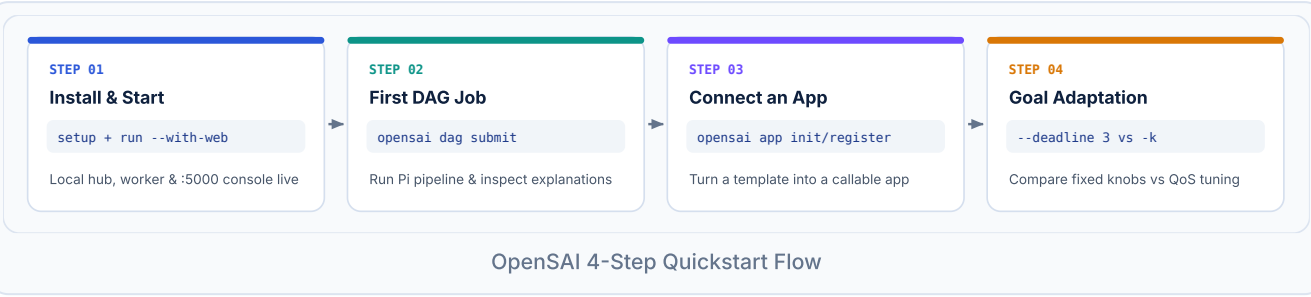
10-MINUTE QUICKSTART

OpenSAI Quickstart Guide

Launch a local cluster, run your first distributed DAG, scaffold a custom application, and explore goal-driven adaptation in 10 minutes on one computer.

Overview | User Manual | Agent Integration | FAQ

One computer is all you need for this guide. Steps 1–3 use only a CPU and the Python standard library—no GPU, model weights, external datasets, Docker, or Node.js required. Step 4 installs `numpy` to demonstrate parallel clustering and adaptive tuning. Everything you build here scales unchanged to a multi-machine cluster.



OpenSAI 4-Step Quickstart Flow

1. Install and Launch the Console

Ensure **Python 3.10 or later** is installed, clone github.com/INSAI-institute/OpenSAI (`git clone https://github.com/INSAI-institute/OpenSAI.git`) or extract `OpenSAI_Release.zip`, and open a terminal in the project root containing `scripts/`, `examples/`, and `Offloading-System/` (internet access is needed once to install dependencies).

> **i Note**
>
> **System prerequisites**: On Ubuntu / Debian, install venv support if missing (`sudo apt install python3-venv`). On Linux / macOS, if `redis-server` is not already on `PATH`, the setup script compiles a portable Redis instance automatically and requires `cc`, `make`, and `curl` or `wget` (or install `redis-server` beforehand via your package manager).

Linux / macOS:

```bash
bash scripts/setup.sh
bash scripts/run.sh --with-web
source .venv/bin/activate
```

Windows PowerShell:

```powershell
powershell -ExecutionPolicy Bypass -File scripts\setup.ps1
powershell -ExecutionPolicy Bypass -File scripts\run.ps1 -WithWeb
.venv\Scripts\Activate.ps1
```

The launcher starts Redis, the coordinator hub (`server.py`), one local worker device (`run_device.py`), and the Web console bridge at `http://127.0.0.1:5000`.

Verify that your local worker is online from the activated terminal:

```bash
opensai devices list
```

Typical JSON output (abbreviated):

```json
[
  {
    "device_id": "设备1",
    "allocation_index": 1,
    "status": "ONLINE",
    "cpu_cores": 8
  }
]
```

Open `http://127.0.0.1:5000` in your browser—use the **EN / 中** button in the top-right corner to toggle English and Chinese, the theme icon to switch between **Paper** (light) and **Ink** (dark), or **Run demo** to launch a built-in walkthrough DAG.

> ✓ **You Are Ready When**
>
> `opensai devices list` shows an `ONLINE` worker and the bottom-left sidebar indicator shows **`Connected · live`**. If a top banner indicates **`Sample data.`**, click **`Switch to live data`** (or turn off sample mode under **Settings**) to inspect your actual machine.

> ✦ **Tip**
>
> Whenever you open a new terminal window, navigate to the project root and re-activate `.venv` (`source .venv/bin/activate` or `.venv\Scripts\Activate.ps1`).

2. Submit Your First DAG and Inspect Decisions

Start with `pi_pipeline.json`, a self-contained two-step DAG that estimates π via Monte Carlo sampling and then runs a brief synchronization step. Validate locally first, then submit:

```bash
opensai dag validate -f examples/dags/pi_pipeline.json
opensai dag submit -f examples/dags/pi_pipeline.json
```

`submit` returns a unique `job_id` (for example, `job_01M6S9WR8S84WV2920DQ8J3E2H`). Substitute it for `YOUR_JOB_ID` below to wait for completion and inspect the scheduling rationale:

```bash
opensai result wait YOUR_JOB_ID
opensai explain YOUR_JOB_ID
```

Key fields returned in the result:

Field / Status	What It Means
`job_id`	Unique identifier for this execution; reuse it across CLI queries and console views
`QUEUED` / `RUNNING`	Waiting for an eligible worker slot, or actively executing steps
`SUCCEEDED`	All required steps completed (inspect `nodes[].result` for the π estimate)
`duration_s`	End-to-end wall-clock duration from submission to completion, in seconds
`explain` output	Shows chosen devices, allocated CPU cores, parameter settings, and decision rationale

> ✓ **You Have Succeeded When**
>
> `status` is `SUCCEEDED` and opening **Runs** in the Web console displays the completed job, execution timeline, per-step outputs, and elapsed time. Note that a client wait timeout only ends local polling—the background job continues running and can be queried again with the same `job_id`.

3. Scaffold and Run Your First Custom Application

An **application** is a reusable, named workflow package; a **job** is a single execution of it. Create, validate, register, and invoke a standard-library starter package in four commands:

```bash
opensai app init my_app
opensai app validate my_app
opensai app register my_app
opensai app run my_app -i size=800000 --wait
```

> ✓ **You Have Succeeded When**
>
> Registration succeeds and the run returns `SUCCEEDED`. Inside `my_app/`:

- `my_app/app.yaml` declares user `inputs`, tunable `knobs`, dependent `steps`, and default `qos` goals;
- `my_app/steps/` holds Python step classes. Data written to `_artifact_out` by an upstream step is automatically made available at `_artifact_in` on the downstream step's device (transferred over QUIC whenever steps run on different machines).

Edit those files to plug in your own logic, then run `validate` and `register` again. See Chapter 6 of the User Manual (and `examples/apps/number_stats`) for a complete walkthrough.

4. Try Goal-Driven Bidirectional Adaptation

The `cluster_analytics` application clusters two synthetic data shards in parallel and merges their centroids. It exposes feature dimensions (`dims`) and iteration count (`iters`) as tunable **knobs**. Install `numpy` and run it once with defaults:

```bash
python -m pip install numpy
opensai app register examples/apps/cluster_analytics
opensai app run cluster_analytics -i points=30000 -i seed=1 --wait
```

Enable Joint Device, Resource, and Knob Selection

The default `plan` policy plans graph placement first and then selects node knobs and cores for QoS. Setting `OPENSAI_PLACEMENT=adaptive` includes device choice, CPU/GPU target, core count, and knobs in one joint optimization. Restart local services for your platform:

Linux / macOS:

```bash
export OPENSAI_PLACEMENT=adaptive
bash scripts/run.sh --with-web
```

Windows PowerShell:

```powershell
$env:OPENSAI_PLACEMENT="adaptive"
powershell -ExecutionPolicy Bypass -File scripts\run.ps1 -WithWeb
```

Confirm `opensai devices list` shows your worker online, then compare two runs on identical input:

```bash
# Run A: Pin both knobs explicitly with -k
opensai app run cluster_analytics -i points=30000 -i seed=1 -k dims=16 -k iters=20 --wait

# Run B: Leave knobs open and set a 3-second deadline target
opensai app run cluster_analytics -i points=30000 -i seed=1 --deadline 3 --wait
```

Compare measured duration, quality score, and core-seconds in **Runs**, and inspect why each configuration was chosen in **Adaptation** or via `opensai explain YOUR_JOB_ID`.

> ✦ **Tip**
> When local compute is ample enough to finish the highest-quality tier well within 3 seconds, keeping that tier is the expected optimal choice. Increase `points` or tighten `--deadline` to watch OpenSAI step down knob tiers automatically to protect timeliness.

5. Stop Services and Choose Your Next Step

When finished, stop the local hub, worker, and Web bridge (include `--all` / `-All` to also stop the portable Redis managed by OpenSAI):

- **Linux / macOS**: `bash scripts/stop.sh --all`
- **Windows PowerShell**: `powershell -ExecutionPolicy Bypass -File scripts\stop.ps1 -All`

Your Next Goal	Where to Go
Connect multiple machines into a real edge cluster	User Manual Chapter 3 (Multi-Host Deployment & Ports)
Run multi-vehicle & roadside cooperative perception	User Manual Section 5.6 (`coop_perception` & `how2comm_perception`)
Package your own Python or vision models as an app	User Manual Chapter 6 (Manifests, Knobs & Optional Branches)
Enable QUIC relay multipath across devices	User Manual Chapter 9 (`OPENSAI_MULTIPATH=1` & Path Records)

Your Next Goal	Where to Go
Call OpenSAI programmatically from Python	User Manual Chapter 11 (`OpenSAIClient` SDK)
Operate the cluster via an AI agent	Agent Integration Guide & FAQ and Troubleshooting