Agent orchestrator

Python service providing AI agent chat, MCP tools integration, and RAG on CRM documents, based on FastAPI and LangChain. Called by the Worker's Consumer processes via HTTP/SSE.

- Orchestrator architecture
- Orchestrator operations

Orchestrator architecture

Three layers:

1. **PHP CRM** (Worker Consumer), calls orchestrator via HTTP
2. **Python orchestrator** (FastAPI, LangChain)
3. **LLM / MCP servers / Chroma**

Relevant Worker Consumer methods: `llmChat()` (chat relay), `elaborateRag()` (document upload + vector build).

RAG

Indexing

Triggered on agent save with the Documents feature enabled. The Worker Consumer:

1. Reads selected CRM Documents
2. `POST /rag/keep` — prunes stale docs from orchestrator
3. `POST /rag/upload` — uploads new/changed files (multipart), stored as `docs/shared/<md5>.<ext>`, symlinked into `docs/<agent_id>/`
4. `POST /rag/build` — indexes all docs for the agent into Chroma at `vectors/<agent_id>/`.

Querying

With `rag: true` in `/agent/run`, Python injects a `query_documents` tool. The LLM decides when to call it. The orchestrator decomposes the question into ≤3 sub-questions (`needs_retrieval` flag), queries Chroma per sub-question, deduplicates by `doc_id`, re-ranks with FlashRank, and returns context. The LLM answer is grounded strictly in retrieved context.

Endpoints

The Python service exposes the following REST endpoints. All are mounted on the FastAPI app at port 8120.

Endpoint	Description

`POST /agent/run`	Agent loop: LLM + MCP tools + guardrails + optional RAG. SSE or JSON.
`POST /tools/inspect`	Introspect MCP server tools.
`POST /rag/build`	Index documents for an `agent_id` into Chroma.
`POST /rag/run`	Query vector store with question decomposition.
`POST /rag/keep`	Prune agent's doc symlinks to match `{filename: md5}`.
`POST /rag/upload?agent_id=`	Upload file to shared pool + symlink into agent's dir.

File Reference

```
plugins/agent/
├── compose.yaml
├── app.Dockerfile
├── requirements.txt
├── config.yaml        # auth token
├── src/vte_agent/
│   ├── __init__.py
│   ├── __main__.py
│   ├── agent.py             # /agent/run, /tools/inspect, calculator + rag tools
│   ├── config.py
│   ├── docs.py              # doc loaders
│   ├── logs.py
│   ├── models.py            # GGUFEmbeddings
│   ├── rag.py               # /rag/* endpoints
│   ├── schemas.py
│   ├── user_manual.py       # builtin vtenext user manual search tool
│   └── utils.py
├── docs/
│   ├── shared/        # <md5>.<ext> — deduplicated by content hash
│   └── <agent_id>/    # symlinks → ../shared/<md5>.<ext>
└── vectors/
    └── <agent_id>/    # chroma.sqlite3, parent_docs.json, description.txt

cache_local/
└── huggingface/       # local models cache (embedding, rerank)
```

Orchestrator operations

Requirements

`llama-cpp-python` is installed as a **pre-built wheel** (not compiled from source). `requirements.txt` specifies the **Vulkan** variant via `--extra-index-url https://abetlen.github.io/llama-cpp-python/whl/vulkan`. Other backends are available by changing the index URL:

Backend	Index URL suffix
cpu	`.../whl/cpu`
vulkan **(default)**	`.../whl/vulkan`
cuda	`.../whl/cuda`
rocm	`.../whl/rocm`
metal	`.../whl/metal`
sycl	`.../whl/sycl`

Docker image installs `libvulkan1`. GPU access (`/dev/dri`) is commented out in `compose.yaml` by default — uncomment for hardware acceleration. Falls back to CPU without GPU.

The x86-64-v2 baseline or equivalent is required by NumPy's pre-built wheels (see NumPy SIMD build options), usually configurable in virtual machines. CPUs without these instructions can still run the orchestrator by **recompiling NumPy from source** with reduced SIMD flags (`NPY_DISABLE_CPU_FEATURES`), or by using a distro that ships a compatible build.

Installation

Using Docker

The Python orchestrator runs in Docker. Quick start:

```
cd plugins/agent
docker compose up -d --build
```

Verify: `curl http://localhost:8120/docs` should show the Swagger UI.

Port `127.0.0.1:8120:8120` — bound to localhost only, **MUST NOT** be publicly exposed.

Here a quick snippet to install docker.

Environment Variables

Variable	Default
`HOST` (inside the container)	`0.0.0.0`
`PORT`	`8120`
`EMBED_MODEL`	`nomic-ai/nomic-embed-text-v2-moe-GGUF:Q8_0`
`RERANK_MODEL`	`ms-marco-MiniLM-L-12-v2`
`HF_CACHE_DIR`	`/app/hf_cache`
`DOCUMENTS_DIR`	`/app/docs`
`VECTORS_DIR`	`/app/vectors`

In case of multiple orchestrators on the same machine it's highly recommended to point `HF_CACHE_DIR` to the same directory.

Without Docker

This setup is technically possible but it is **unsupported** by our support subscription service. The script assumes Ubuntu 24.04 LTS but **it's experimental and not guaranteed to work**.

script

```
#!/bin/bash
set -euo pipefail

# install host dependencies
pkgs=(
  python3
  python3-pip
  python3.12-venv
  python-is-python3
  libgomp1
  libvulkan1
)
apt install "${pkgs[@]}"
```

```
# make virtual env
VTE="/var/www/html/vte"
AGENT="$VTE/plugins/agent"
cd "$AGENT"
python -m venv venv
source venv/bin/activate

# install python libraries
pip install --upgrade pip setuptools wheel
pip install -r requirements.txt

# install systemd service
SERVICE="vte-agent.service"
cat > "/etc/systemd/system/$SERVICE" <<UNIT
[Unit]
Description=vtenext AI agent orchestrator
After=network-online.target
Wants=network-online.target

[Service]
Type=simple
Restart=unless-stopped
RestartSec=3

User=www-data
Group=www-data

Environment=PYTHONUNBUFFERED=1
Environment=PYTHONDONTWRITEBYTECODE=1
Environment=PYTHONPATH=${AGENT}/src
Environment=HF_CACHE_DIR=${AGENT}/hf_cache
Environment=HOST=127.0.0.1
Environment=PORT=8120

WorkingDirectory=${AGENT}
ExecStart=${AGENT}/venv/bin/python -m vte_agent

[Install]
WantedBy=default.target
```

```
UNIT

systemctl daemon-reload
systemctl enable --now "$SERVICE"

# verify
systemctl --no-pager --full status "$SERVICE"
curl -fsS "http://127.0.0.1:8120/docs" >/dev/null && echo "OK"
```

Troubleshooting

- **Failed to build installable wheels / no gcc**: Python is trying to compile the package (we advise against), check `lscpu` and `requirements.txt` for correct cpu architecture.
- **Slow startup:** Embedding model downloads from HuggingFace on first container start — cached in `HF_CACHE_DIR` afterwards.
- **Empty docs:** `/rag/build` raises `RuntimeError` if `docs/<agent_id>/` is empty.
- **GPU not used:** Uncomment `/dev/dri` in `compose.yaml`. Falls back to CPU otherwise.
- **CPU compat:** Verify with `/lib64/ld-linux-x86-64.so.2 --help`