// cyber-wyse.com · Local AI Workstation Series

Local AI Stack

SearXNG · MeiliSearch · Watchdog RAG Pipeline

Installation Guide & Feature Reference

cyber-wyse.com

1. Overview

This guide covers the installation and configuration of three core components of the Cyber-Wyse local AI workstation. Together they give you private, offline-capable search across the web and your own documents, with AI-synthesised answers — no data sent to the cloud.

Service	What it does	Port
SearXNG	Privacy-respecting meta search engine. Aggregates results from Google, Bing, DuckDuckGo, GitHub and more. Ru	8080
MeiliSearch	Lightning-fast local document search. Indexes your own files (PDF, DOCX, Markdown, code) for instant full-text ret	7700
Watchdog	Python filesystem monitor. Watches folders for new or updated files and automatically re-indexes them into MeiliSe	—

Prerequisites

The following must already be installed and running on your Ubuntu machine:

- Docker Engine (not Docker Desktop)
- Docker Compose plugin (docker compose, not docker-compose)
- Ollama with at least one model pulled (e.g. qwen2.5-coder:7b)
- Python 3.10+ with pip
- Open WebUI (optional — used in the unified UI section)

Working directory: ~/ai-stack. Adjust paths to suit your own layout.

2. SearXNG — Privacy Web Search

2.1 What It Does

SearXNG is a self-hosted meta search engine. When you type a query it simultaneously queries multiple search engines, de-duplicates and ranks the results, and returns them to you with no tracking, no profiling and no API keys required.

In the local AI stack it serves two roles:

- Standalone web search via its own UI at http://localhost:8080
- Data source for Ollama — search results are passed as context so the LLM can give a Perplexity-style answer grounded in live web content

2.2 Installation

STEP 01 **Create the config folder**

Create ~/ai-stack/searxng and generate a proper secret key.

bash

```
mkdir -p ~/ai-stack/searxng && cd ~/ai-stack/searxng
python3 -c "import secrets; print(secrets.token_hex(32))"
```

STEP 02 **Create settings.yml**

Create ~/ai-stack/searxng/settings.yml with the following minimum content:

~/ai-stack/searxng/settings.yml

```
use_default_settings: true

server:
  secret_key: "change-me-to-a-long-random-string"
  limiter: false
  image_proxy: true

ui:
  default_theme: simple
  default_lang: en

search:
  safe_search: 0
  autocomplete: ""
  default_lang: en
```

STEP 03 **Add to docker-compose.yml**

docker-compose.yml — SearXNG service block

```
  searxng:
    image: searxng/searxng:latest
    container_name: searxng
    restart: unless-stopped
    ports:
      - "8080:8080"
    volumes:
      - ./searxng:/etc/searxng:rw
    environment:
      - SEARXNG_BASE_URL=http://localhost:8080/
    cap_drop: [ALL]
    cap_add: [CHOWN, SETGID, SETUID]
```

STEP 04 **Start the service**

bash

```
cd ~/ai-stack && docker compose up -d searxng
docker compose logs -f searxng
```

Navigate to http://localhost:8080 in your browser. You should see the SearXNG interface.

2.3 Querying SearXNG via API

SearXNG exposes a JSON API. This is how the FastAPI backend queries it:

python

```
import httpx

async def web_search(query: str, num_results: int = 10):
    url = "http://localhost:8080/search"
```

```
    params = {
        "q": query,
        "format": "json",
        "categories": "general",
        "language": "en",
    }
    async with httpx.AsyncClient() as client:
        response = await client.get(url, params=params, timeout=10)
        data = response.json()
    results = data.get("results", [])
    return [{"title": r["title"], "url": r["url"], "snippet": r.get("content", "")}
            for r in results[:num_results]]
```

---

3. MeiliSearch — Local Document Search

3.1 What It Does

MeiliSearch indexes your own documents and returns highly relevant results with typo tolerance and sub-50ms response times. In this stack it indexes all documents in your watched folders and retrieves relevant chunks to pass to Ollama as context — the RAG pattern.

3.2 Installation

STEP 01 **Add to docker-compose.yml**

docker-compose.yml — MeiliSearch service block

```
  meilisearch:
    image: getmeili/meilisearch:latest
    container_name: meilisearch
    restart: unless-stopped
    ports:
      - "7700:7700"
    environment:
      - MEILI_MASTER_KEY=your-master-key-here
      - MEILI_ENV=development
    volumes:
      - ./meilisearch_data:/meili_data
```

In production set MEILI_ENV=production. Verify at http://localhost:7700.

STEP 02 **Start and install Python client**

bash

```
cd ~/ai-stack && docker compose up -d meilisearch
pip install meilisearch --break-system-packages
```

STEP 03 **Create the index**

python — run once

```
import meilisearch

client = meilisearch.Client("http://localhost:7700", "your-master-key-here")
index = client.create_index("documents", {"primaryKey": "id"})
client.index("documents").update_searchable_attributes(["title", "content", "filename", "path"])
client.index("documents").update_filterable_attributes(["type", "folder", "date_modified"])
```

3.3 Searching the Index

python

```
def search_docs(query: str, limit: int = 5):
    client = meilisearch.Client("http://localhost:7700", "your-master-key-here")
```

```
    results = client.index("documents").search(query, {
        "limit": limit,
        "attributesToHighlight": ["content"],
        "attributesToCrop": ["content"],
        "cropLength": 200
    })
    return results["hits"]
```

---

4. Watchdog — Automatic Document Indexing

4.1 What It Does

Watchdog monitors the filesystem for changes. When a file is created, modified, or deleted in a watched folder it is automatically parsed and re-indexed into MeiliSearch — no manual re-indexing required.

4.2 Installation

STEP 01 **Install dependencies**

bash

```
pip install watchdog pymupdf python-docx markdown --break-system-packages
```

STEP 02 **The indexing pipeline — ~/ai-stack/indexer/indexer.py**

python

```
import time, hashlib, logging, meilisearch
from pathlib import Path
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler
import fitz
from docx import Document as DocxDoc

WATCH_PATHS = ["/home/eamon/docs", "/home/eamon/dev/wysedsp"]
EXTENSIONS  = {".pdf", ".docx", ".md", ".txt", ".py", ".cpp", ".h"}
MEILI_URL   = "http://localhost:7700"
MEILI_KEY   = "your-master-key-here"
INDEX_NAME  = "documents"
CHUNK_SIZE  = 500

client = meilisearch.Client(MEILI_URL, MEILI_KEY)

def extract_text(path):
    ext = path.suffix.lower()
    if ext == ".pdf": return " ".join(p.get_text() for p in fitz.open(str(path)))
    if ext == ".docx": return " ".join(p.text for p in DocxDoc(str(path)).paragraphs)
    return path.read_text(errors="ignore")

def chunk_text(text):
    words = text.split()
    chunks, current = [], []
    for word in words:
        current.append(word)
        if len(" ".join(current)) >= CHUNK_SIZE:
            chunks.append(" ".join(current))
            current = current[-20:]
    if current: chunks.append(" ".join(current))
    return chunks

def index_file(path):
    if path.suffix.lower() not in EXTENSIONS: return
    text   = extract_text(path)
    chunks = chunk_text(text)
    docs   = [{"id": hashlib.md5(f"{path}:{i}".encode()).hexdigest(),
               "title": path.stem, "filename": path.name,
               "path": str(path), "type": path.suffix.lstrip("."),
               "folder": str(path.parent), "chunk_index": i, "content": c}
              for i, c in enumerate(chunks)]
    client.index(INDEX_NAME).add_documents(docs)

class IndexHandler(FileSystemEventHandler):
    def on_created(self, e):
        if not e.is_directory: index_file(Path(e.src_path))
    def on_modified(self, e):
        if not e.is_directory: index_file(Path(e.src_path))

if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO, format="%(asctime)s %(levelname)s %(message)s")
    observer = Observer()
    for p in WATCH_PATHS:
        observer.schedule(IndexHandler(), p, recursive=True)
    observer.start()
    try:
        while True: time.sleep(1)
    except KeyboardInterrupt:
        observer.stop()
    observer.join()
```

STEP 03 **Run as a systemd service**

/etc/systemd/system/ai-indexer.service

```
[Unit]
Description=AI Document Indexer
After=network.target

[Service]
ExecStart=/usr/bin/python3 /home/eamon/ai-stack/indexer/indexer.py
WorkingDirectory=/home/eamon/ai-stack
User=eamon
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
```

bash

```
sudo systemctl daemon-reload && sudo systemctl enable ai-indexer
sudo systemctl start ai-indexer && sudo systemctl status ai-indexer
```

STEP 04 **Initial bulk index**

python — run once

```
for watch_path in WATCH_PATHS:
    for filepath in Path(watch_path).rglob("*"):
        if filepath.is_file(): index_file(filepath)
print("Bulk index complete")
```

---

5. Combined Usage — Perplexity-Style AI Search

5.1 Architecture

With all three services running, the query flow works like this:

python — main.py

```
from fastapi import FastAPI
import httpx, meilisearch, ollama

app   = FastAPI()
meili = meilisearch.Client("http://localhost:7700", "your-key")

@app.get("/search")
async def search(q: str):
    # 1 — web results
    async with httpx.AsyncClient() as c:
        web_resp = await c.get("http://localhost:8080/search",
            params={"q": q, "format": "json"}, timeout=10)
    web_hits = web_resp.json().get("results", [])[:5]
    # 2 — local doc results
    local_hits = meili.index("documents").search(q, {"limit": 3})["hits"]
    # 3 — build context
    context  = "WEB RESULTS:\n"
    context += "".join(f"- {r['title']}: {r.get('content','')}\n" for r in web_hits)
    context += "\nLOCAL DOCS:\n"
    context += "".join(f"- {r['title']} ({r['filename']}): {r['content'][:300]}\n" for r in local_hits)
    # 4 — ask Ollama
    prompt   = f"Using the following sources, answer: {q}\n\n{context}"
    response = ollama.chat(model="qwen2.5-coder:7b",
                           messages=[{"role": "user", "content": prompt}])
    return {"answer": response["message"]["content"], "web": web_hits, "local": local_hits}
```

6. Complete docker-compose.yml

```yaml
version: "3.8"

services:

  ollama:
    image: ollama/ollama:latest
    container_name: ollama
    restart: unless-stopped
    ports:
      - "11434:11434"
    volumes:
      - ollama_data:/root/.ollama
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: all
              capabilities: [gpu]

  open-webui:
    image: ghcr.io/open-webui/open-webui:main
    container_name: open-webui
    restart: unless-stopped
    ports:
      - "3000:8080"
    volumes:
      - open_webui_data:/app/backend/data
    environment:
      - OLLAMA_BASE_URL=http://ollama:11434
    depends_on:
      - ollama

  searxng:
    image: searxng/searxng:latest
    container_name: searxng
    restart: unless-stopped
    ports:
      - "8080:8080"
    volumes:
      - ./searxng:/etc/searxng:rw
    environment:
      - SEARXNG_BASE_URL=http://localhost:8080/
    cap_drop: [ALL]
    cap_add: [CHOWN, SETGID, SETUID]

  meilisearch:
    image: getmeili/meilisearch:latest
    container_name: meilisearch
    restart: unless-stopped
    ports:
      - "7700:7700"
    environment:
      - MEILI_MASTER_KEY=your-master-key-here
      - MEILI_ENV=development
    volumes:
      - ./meilisearch_data:/meili_data

volumes:
  ollama_data:
  open_webui_data:
```

7. Quick Reference

Service / Command	URL / Command
SearXNG UI	http://localhost:8080
SearXNG JSON API	http://localhost:8080/search?q=QUERY&format=json
MeiliSearch UI	http://localhost:7700
Ollama API	http://localhost:11434/api/chat
Open WebUI	http://localhost:3000
FastAPI search	http://localhost:8000/search?q=QUERY
Start all services	docker compose up -d
Stop all services	docker compose down
View logs	docker compose logs -f [service]
Re-index all	python indexer/indexer.py --bulk
Check Meili index	curl http://localhost:7700/indexes/documents/stats