

Handbook

Honey-Barrel Stack

inż. Marcin Bednarski
14 Lipca 2026

wersja 0.2

1. Architektoniczny briefing

[Wprowadzenie](#)

[Ogólna architektura systemu](#)

[Ogólna architektura komunikacyjna](#)

[Konfiguracja GPU i Ollama](#)

[Modelfile dla Ollama \(~20GB RAM + 6GB VRAM\)](#)

[Investment Analyst Flow \(UI + API\)](#)

[Ollama \(external LLM layer\)](#)

[Event Flow \(system internal\)](#)

[Workspace integration layer](#)

[Security / boundary model](#)

[End-to-end flow \(najważniejszy diagram\)](#)

[Wnioski architektoniczne](#)

2. Współpraca z github-em (setting up environment)

3. docker-compose.yml

4. Skrypt uruchomieniowy full_stack_recompose.sh

5. Skrypt diagnostyczny check_hermes_cron_tasks.sh

6. Skrypt diagnostyczny check_cycle_all_logs.sh

7. Skrypt diagnostyczny health_check_github.sh

8. Wrapper (MAIN) daily_investment_watchlist_report_generator.sh

9. Plik .env

10. Dokumentacja

11. INVESTMENT ANALYST TERMINAL

12. Podsumowanie

CDN..

1. Architektoniczny briefing

Wprowadzenie

Dokument ma na celu przyswojenie czytelnikowi i prawdopodobnie przyszłemu administratorowi systemu “Honey-Barrel Investment Analyst Stack” a inaczej po prostu automatyczny Stock-Investment system oparty na skrypcie wrapper-ze uruchamianym z crontab-a.

Postaramy się tutaj omówić podstawowe moduły systemu i zamknąć temat zwięźle, ale i zajrzeć trochę głębiej na ustawienia i pomocne przy T/S i uruchomieniu flagi (parametry i zmienne środowiskowe). Chciałbym również uprościć możliwie opis celów systemu i unikać komplikowania sobie “życia”.

Honey Barrel Investment Analyst Stack

This is a self-hosted Docker stack called **Honey Barrel** — an AI-agent runtime for algorithmic stock trading and research. It wires an LLM (Ollama), a market-data/ML predictor (Yahoo Finance + XGBoost), and a broker execution bridge (Alpaca) through a uniform **MCP (Model Context Protocol)** layer.

Architecture

The stack is orchestrated by `docker-compose.yml` with three isolated bridge networks:

- `frontend-net` — public/UI traffic
- `backend-net` — internal service communication
- `mcp-net` — MCP tool-server communication

The privileged runtime is **Hermes**, which registers MCP tool servers and dispatches tasks.

Main components

Component	What it does
Hermes (<code>workspace/hermes/</code>)	Privileged agent runtime. Runs with <code>docker.sock</code> access, <code>SYS_ADMIN</code> , and <code>NET_ADMIN</code> . It registers MCP servers and drives one-shot cron jobs.
investment-analyst-api (<code>workspace/investment-analyst-api</code> <code>/</code>)	FastAPI service. Fetches Yahoo Finance data, engineers technical-indicator features, trains/predicts with XGBoost/Random Forest, manages watchlists, and exposes tools via both REST and MCP (<code>mcp.py</code>).
alpaca-mcp (<code>workspace/alpaca-mcp/</code>)	Python MCP server bridging to Alpaca. Implements guardrails (<code>guardrails.py</code>) and exposes order/portfolio/market-data tools.
workspace-mcp (<code>workspace/workspace-mcp/</code>)	TypeScript MCP server. This is the designated persistent state layer: watchlists, reports, cron history, and any mutable state.
mcp-toolbox / stdio-terminal / mcp-validator	Utility MCP servers and a compliance checker.
mcp-gateway / mcp-services (<code>filesystem, git, terminal</code>)	Legacy/commented-out MCP aggregation layer.

trading-api Legacy FastAPI + Postgres + Trading212 bridge. Disabled
(workspace/trading-api/) in compose.

openclaw (workspace/openclaw/) Legacy chat UI. Disabled in compose.

Key technologies

- **Python / FastAPI** — Alpaca bridge, ML analyst, legacy trading API
- **XGBoost / scikit-learn** — ML prediction model
- **pandas / NumPy** — Feature engineering (RSI, SMA, MACD, Bollinger Bands, volatility, volume ratios)
- **Node.js / TypeScript / Express** — workspace-mcp, mcp-toolbox, gateway, validator
- **MCP (Model Context Protocol)** — uniform JSON-RPC 2.0 over SSE/POST contract used across all tool servers
- **Docker Compose** — local orchestration
- **Ollama** — external LLM inference (on host)
- **Alpaca** — live broker execution
- **Fly.io** (fly.toml) — deployment config for a dev/sandbox container (not in use now)

How it runs in production

A cron job executes: `workspace/scripts/daily_investment_watchlist_report_generator.sh`, script wrapper which runs Hermes in one-shot mode, fetches signals, checks Alpaca order verification, and writes reports to `workspace/reports/`. The reports directory contains ~300 dated JSON/MD artifacts plus `.cron_state/`, showing the pipeline has been running and occasionally failing.

Notable design decisions

- **Hermes is the only privileged execution layer** with docker socket and admin capabilities.

- **workspace-mcp is the sole source of persistent state** — the Hermes environment hint explicitly forbids storing state elsewhere.
- **Alpaca is execution-only** — decisions come from the investment-analyst MCP, and live orders require `X-Confirm: true`.
- **All MCP servers share the same contract:** `GET /health`, `GET /mcp` (SSE), `POST /mcp` (JSON-RPC 2.0).

Disabled / legacy parts

- `trading-api`, `openclaw`, `postgres`, `ollama` **self-hosted service**, and `mcp-gateway` **are all commented out in `docker-co`**
- `mpose.yml`.
- The codebase retains multiple architecture docs (`HONEY-BARREL-STACK_SETUP*.md`, `NIBBLES-STACK_SETUP.md`, `architecture_old*.md`) showing iterative evolution.

Ogólna architektura systemu

W Twojej architekturze trzeba rozdzielić dwa środowiska: Windows (Host) i Linux (VM).

Windows 11 host (* DELL XPS 8930, Intel i7 3200MHz, 32GB RAM 2666Mhz)

- └─ Ollama (Gemma 4 E4B Q4)
- | └─ RAM ~21 GB
- | └─ NVIDIA GTX 1060 6GB VRAM
- |

└─ **VirtualBox Ubuntu 26.04** (* 6 vCPUs, 12264 MB RAM, 250 GB HDD, 2 NICs)

- └─ Docker (following APP/Modules are dockerized)
- └─ Hermes
- └─ Alpaca (MCP)
- └─ Investment-Analyst-API (MCP)
- └─ mcp-toolbox (MCP)
- └─ stdio-terminal (MCP)
- └─ workspace
- └─ workspace-mcp (MCP)
- └─ MCP services (mentioned in brackets)

```
[workspace @ubuntoserver]: mbednarski$ dps
```

NAMES	STATUS	STATE	PORTS
alpaca-mcp	Up 2 hours (healthy)	running	0.0.0.0:3003->3001/tcp
hermes	Up 2 hours (healthy)	running	0.0.0.0:8642->8642/tcp, 0.0.0.0:9119->9119/tcp

```

hermes-05511e7f          Up 7 hours          running
investment-analyst-api    Up 2 hours (healthy) running
127.0.0.1:18001->8001/tcp

mcp-toolbox              Up 7 hours (healthy) running
0.0.0.0:3006->3001/tcp

studio-terminal          Up 7 hours (healthy) running 3002/tcp

workspace                Up 7 hours          running

workspace-mcp            Up 7 hours (healthy) running
0.0.0.0:3002->3001/tcp

```

```
[workspace @ubuntoserver]: mbednarski$ alias |grep dps
```

```
alias dps='docker ps --format "table
{{.Names}}\t{{.Status}}\t{{.State}}\t{{.Ports}}" |sort'
```

Ogólna architektura komunikacyjna

Dokument opisujący stack: [***docs/FULL HONEY-BARREL-STACK SETUP.md***](#)

honey-barrel-stack — Architektura komunikacji (HTTP + MCP + Event Flow)

Warstwa ogólna

+++++

System składa się z 4 głównych warstw:

[UI Layer]

└─ OpenClaw UI (18789) (*currently and rather by default disabled)

└─ Investment Analyst UI (18001/ui)

[Orchestration Layer]

└─ Hermes (MCP Runtime + Docker Control + Workspace Access)

[Service Layer]

└─ Investment Analyst API (ML + Data + UI backend)

[External Layer]

└─ Ollama (external host: \${OLLAMA_BASE_IP}:11434)

Główny flow request/response (OpenClaw → Hermes → tools)

+++++

Flow A: User prompt (agent execution)

User

↓

OpenClaw UI (18789)

↓ HTTP

OpenClaw Backend

↓ MCP / HTTP

Hermes Runtime (8642)

↓

MCP Router

↓

Tool / Skill Execution Layer

└─ Investment Analyst API

└─ Workspace filesystem

└─ Docker exec

└─ External LLM (Ollama)

Konfiguracja GPU i Ollama

CPUs z Iscpu (6 vCPU VM) nie opisuje Ollamy, opisuje tylko środowisko Hermesa i aplikacji.

Twoja faktyczna konfiguracja dla Gemma 4

Masz w FULL-STACK:

- Intel i7-8700:
- 6C/12T fizycznie
- 3.2 GHz
- 4.6 GHz Turbo
- GTX 1060 6GB
- Windows 11
- Ollama natywnie na Windows
- Model LLM : Gemma 4 E4B Q4

RAM dostępny dla całego stacku ~32 GB

Najważniejszy punkt: GTX 1060 6GB

Gemma 4 E4B Q4 prawdopodobnie nie mieści się całkowicie na GPU.

Typowo:

model Q4: około 5–6 GB VRAM + overhead,

KV cache przy 64k context: dodatkowe GB,

runtime CUDA: dodatkowa pamięć.

Przy GTX 1060:

część modelu idzie na GPU,

reszta na CPU/RAM.

Reasumując masz **hybrydowy offload**.

Modelfile dla Ollama (~20GB RAM + 6GB VRAM)

Lokacja i kontekst pliku:

C:\AI\gemma4\Modelfile

```
FROM batiai/gemma4-e4b:q4
```

```
PARAMETER temperature 0.7
```

```
PARAMETER top_p 0.90
```

```
PARAMETER top_k 64
```

```
PARAMETER num_ctx 65536
```

```
PARAMETER num_predict 8192
```

```
PARAMETER num_keep 512
```

```
PARAMETER repeat_penalty 1.08
```

```
PARAMETER repeat_last_n 256
```

W PowerShell:

```
cd C:\AI\gemma4

ollama create gemma4-local -f Modelfile
```

FINALNY starter.bat

```
@echo off

title Ollama Gemma4 Local

REM =====

REM NETWORK

REM =====

set OLLAMA_HOST=10.58.37.237:11434          # Tutaj IP Ollama

REM =====

REM GPU SETTINGS

REM =====

set OLLAMA_FLASH_ATTENTION=1

set OLLAMA_KEEP_ALIVE=-1

set OLLAMA_NUM_PARALLEL=2

set OLLAMA_MAX_QUEUE=256

set OLLAMA_GPU_OVERHEAD=536870912

set OLLAMA_MAX_LOADED_MODELS=1

set CUDA_VISIBLE_DEVICES=0
```

```

REM =====
REM START SERVER
REM =====

```

```
echo Starting Ollama Server...
```

```
start "" /B ollama serve
```

```
timeout /t 8 /nobreak >nul
```

```

REM =====
REM START MODEL
REM =====

```

```
ollama run gemma4-local
```

```
pause
```

Jak sprawdzić parametry modelu

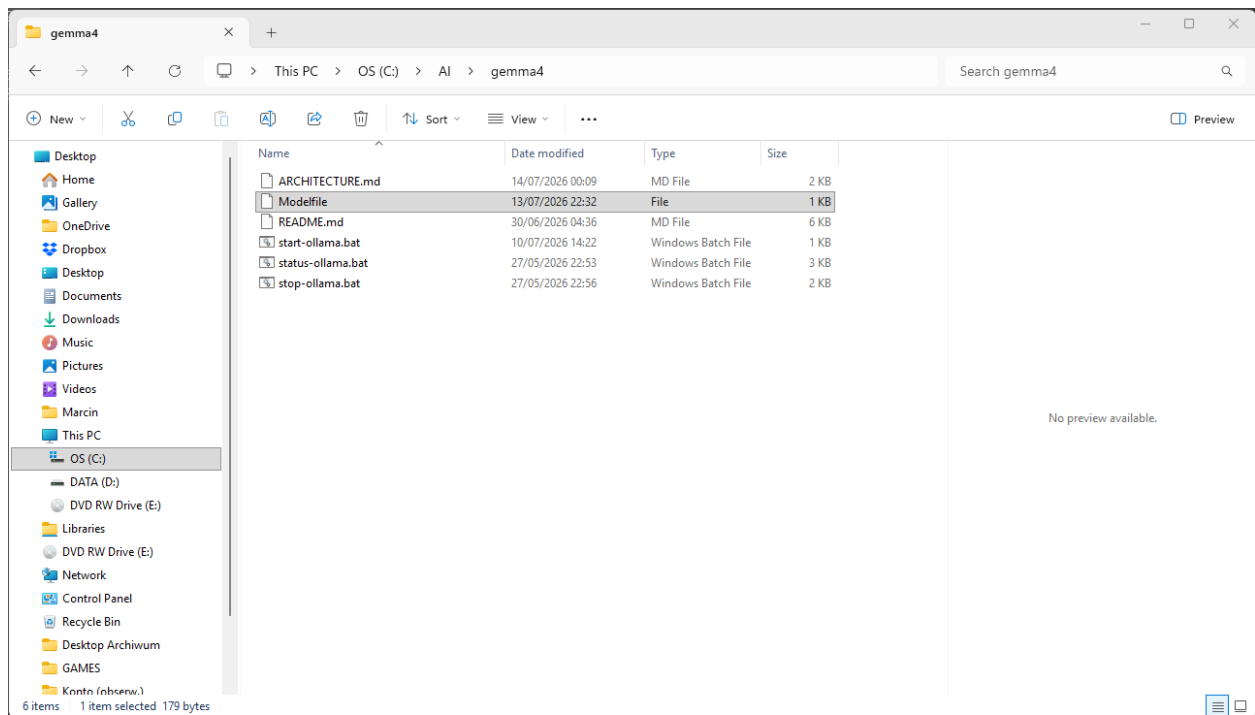
```
ollama show gemma4-local
```

Test generacji

```

curl http://10.58.37.237:11434/api/generate -d
{"model\":\"batiai/gemma4-e4b:q4\", \"prompt\":\"Hello\"}"

```



Investment Analyst Flow (UI + API)

Flow B: User analytics session

User



Investment Analyst UI (18001/ui)



Backend API (FastAPI / Node / Python ML)



Feature pipeline



XGBoost model inference



Yahoo Finance connector



Cache layer (disk)



Response JSON → UI

Data pipeline

Yahoo Finance



OHLCV data fetch



Cache (24h TTL)



Feature engineering



Model inference (XGBoost, 90d lookback)



Prediction output

Ollama (external LLM layer)

Flow C: LLM inference

Hermes / OpenClaw

↓ HTTP

OLLAMA_BASE_IP:11434



External Ollama Server (Windows host)



Response (chat completion / embeddings)



Hermes → agent context

Example request

POST http://\${OLLAMA_BASE_IP}:11434/api/chat

Event Flow (system internal)

Hermes pełni rolę runtime + event routera.

Event bus model

Agent Event



Hermes Event Dispatcher



event types	
tool_call	
workspace_read	
docker_exec	
model_inference	
llm_request (ollama)	

Event lifecycle

event.created



event.routed



event.executing



event.completed



event.returned_to_agent

Workspace integration layer

Shared volume model

Hermes ———┐

OpenClaw ———┤——> /workspace

Investment ———┐

Analyst

Przykładowe operacje:

/workspace/

└── sessions/

└── logs/

└── datasets/

└── models/

└─ scripts/

Hermes może:

- czytać pliki
- zapisywać artefakty
- uruchamiać skrypty
- persistować modele ML

Security / boundary model

Trust zones

ZONE 1: UI

- OpenClaw
- Investment UI

ZONE 2: Runtime

- Hermes (privileged, docker.sock)

ZONE 3: Data/ML

- Investment Analyst

ZONE 4: External

- Ollama host
- Yahoo Finance

Kluczowy punkt:

- Hermes = root-level execution layer
- privileged: true
- SYS_ADMIN

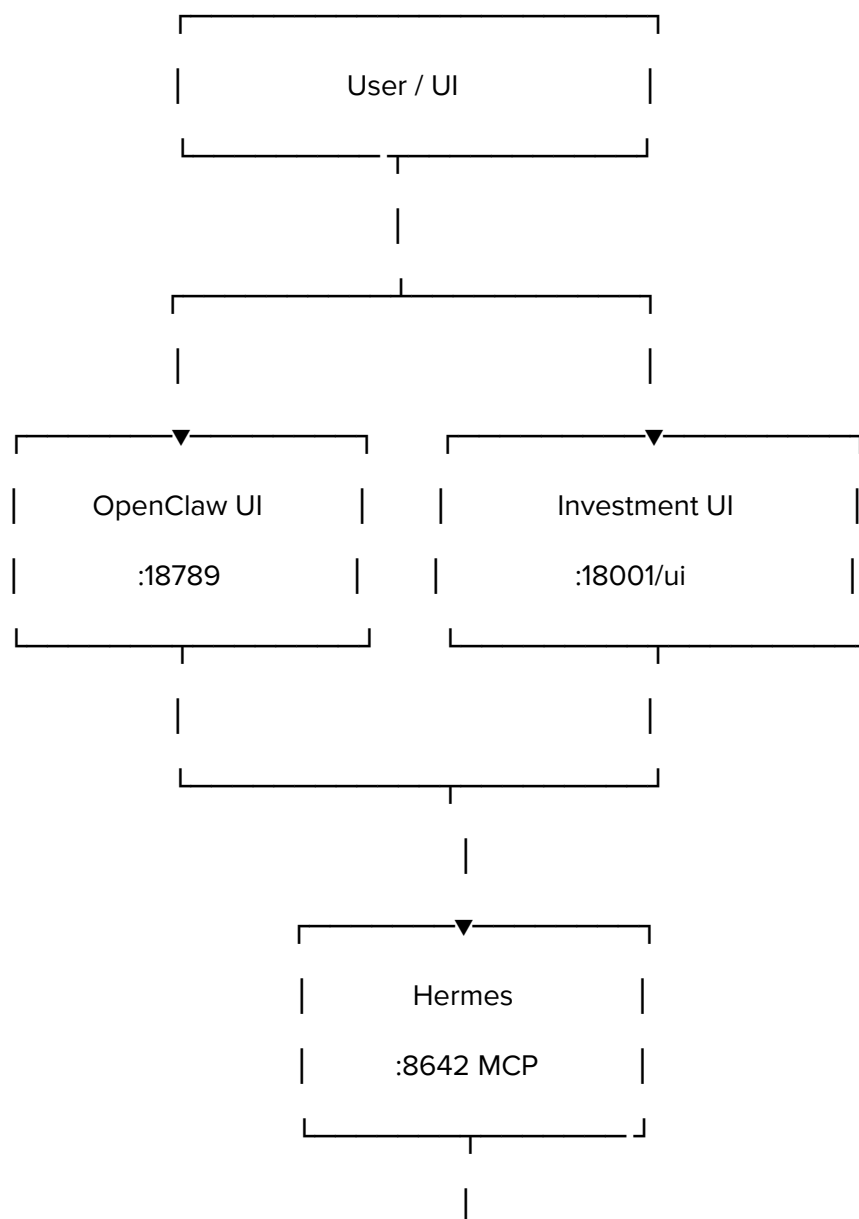
- docker.sock

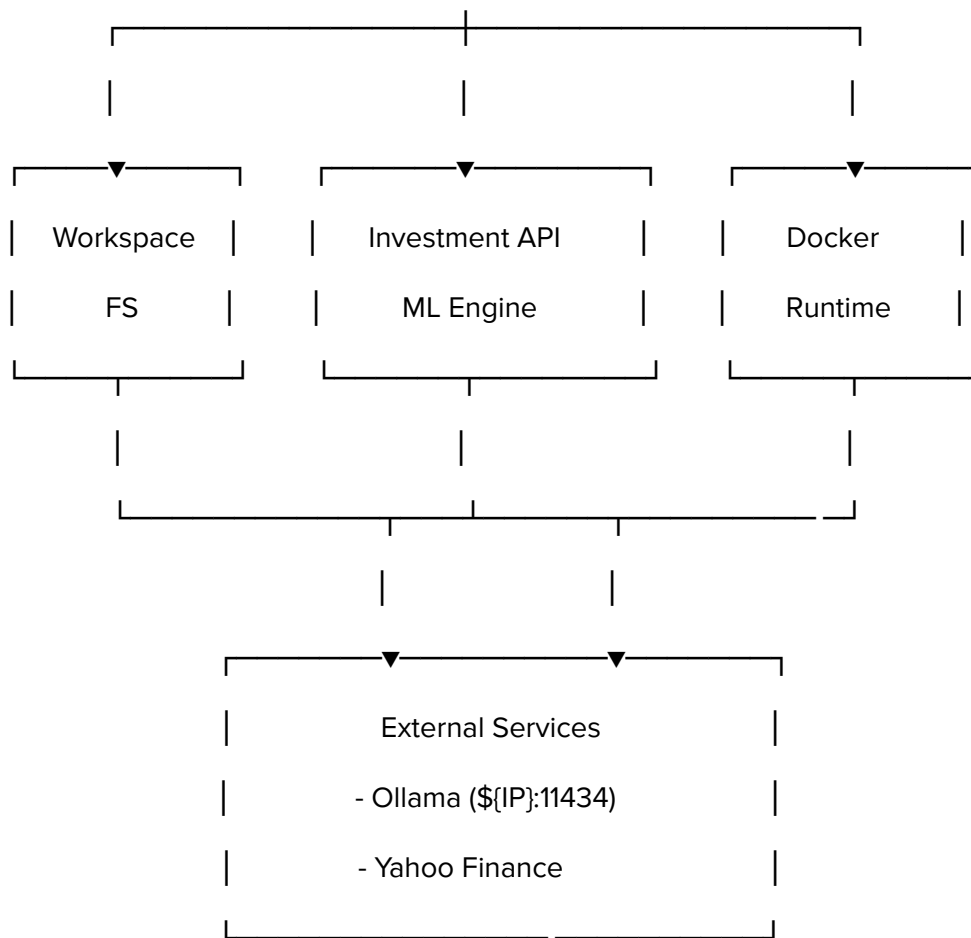
To oznacza:

- pełna kontrola nad hostem przez agent runtime

End-to-end flow (najważniejszy diagram)

Dokument opisujący stack: [docs/FULL HONEY-BARREL-STACK SETUP.md](#)





Wnioski architektoniczne

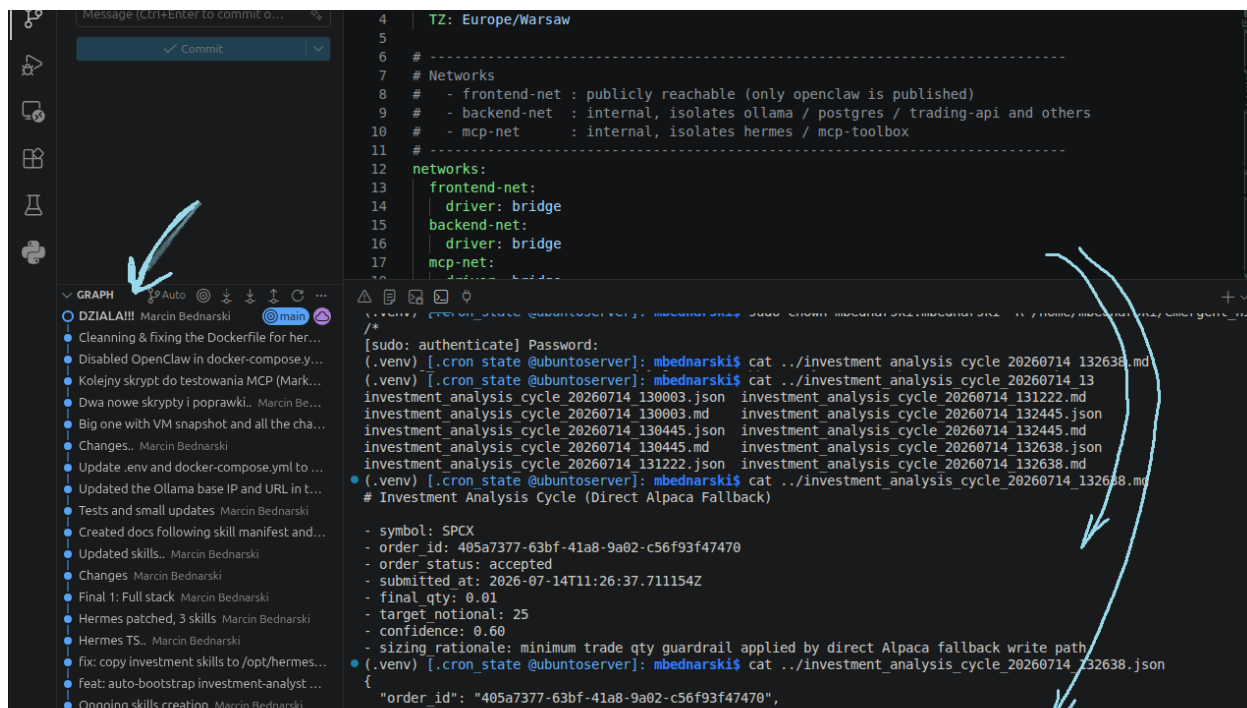
System jest hybrydowym agent runtime + ML platform

Hermes jest centralnym execution kernel

OpenClaw i Investment Analyst to dwa niezależne UI

Ollama jest całkowicie zewnętrzna

Workspace jest shared mutable state (kluczowy element systemu)



Opis: Najważniejsza kwestia całej tej developerki jest precyzja (dbałość) i bezawaryjność (sprawdzenie kodu w możliwie “każdych” warunkach). 😊

2. Współpraca z github-em (setting up environment)

Utwórz parę kluczy RSA jeśli jeszcze nie masz, potem skopiuj id_rsa.pub do kluczy SSH w profilu Marcina (github account).

```
ssh-keygen -t rsa -C "marcin.bednarski@gmail.com"
```

Pomoc:

<https://hpc.meil.pw.edu.pl/creating-a-ssh-key-pair/>

<https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent>

<https://docs.github.com/en/get-started/git-basics/about-remote-repositories#cloning-with-ssh-urls>

Ujednolice użytkownika i stałe środowiskowe, w taki sposób że, będzie możliwe copy+paste poleceń prosto do Shell'a. Ty użyj swojego.

```
export USER=mbednarski

export HOMEDIR=/home/mbednarski

export WORKSPACE=/home/mbednarski/honey-barrel/workspace


mkdir -p $WORKSPACE

chown $USER:$USER -R $WORKSPACE $WORKSPACE/*

cd $WORKSPACE

git remote add origin https://github.com/git-mb001/honey-barrel.git

git remote -v

# Przetestuj połączenie

ssh -T git@github.com:git-mb001/honey-barrel.git

# Sklonuj repozytorium

git clone git@github.com:git-mb001/honey-barrel.git

git remote set-url origin git@github.com:git-mb001/honey-barrel.git

git remote -v
```

ToDo: Pobierz i zainstaluj VS Code i połącz z kontem github-a. Połącz również domowy katalog pod WORKSPACE w VS Code.

3. docker-compose.yml

docker-compose.yml lub compose.yml to plik konfiguracyjny Docker-a w formacie YAML, który opisuje całą aplikację składającą się z wielu kontenerów Docker. Zamiast uruchamiać każdy kontener osobnym poleceniem docker run, definiujesz wszystko w jednym pliku.

Taki plik definiuje:

- services – kontenery, które mają zostać uruchomione (web, database),
- image – obraz Dockera, z którego zostanie utworzony kontener,
- ports – mapowanie portów hosta do kontenera,
- volumes – trwałe przechowywanie danych,
- environment – zmienne środowiskowe,
- oraz opcjonalnie sieci (networks), zależności (depends_on), limity zasobów i wiele innych ustawień.

```
cd $WORKSPACE
```

```
vim docker-compose.yml      # Our config file contains (E-enabled, D-disabled)
```

1. PostgreSQL (D)
2. Workspace (shared volume for development and debugging) (E)
3. Ollama (no-GPU) (D)
4. One-shot model puller (D)
5. Python backend (Trading212 bridge) (D)
6. Investment Analyst API (ML stock predictions) Its actualy MCP (E)
7. OpenClaw (coollabsio fully-dockerised fork) (D)
8. Hermes (E)
9. MCP Toolbox (example custom MCP server) (E)
10. MCP Filesystem (example MCP server for file storage) (E)
11. MCP Terminal (shell execution via HTTP MCP) (E)
12. MCP Compliance Validator (D)

13. MCP workspace (agent-ready workspace filesystem layer) (E)
14. MCP Alpaca (Trading + Market Data v2 — US equities) (E)
15. MCP Gateway (D)
16. MCP Terminal (example MCP server for terminal) (D)
17. MCP Git (example MCP server for git operations) (D)

```
# Jeżeli plik jest gotowy do uruchomienia.. po prostu możesz:
```

```
docker compose up -d
```

```
# To uruchomi wszystkie kontenery skonfigurowane w pliku docker-compose.yml w #  
trybie Detached.
```

4. Skrypt uruchomieniowy full_stack_recompose.sh

```
cd $WORKSPACE/scripts/
```

```
./full_stack_recompose.sh --help
```

```
# Follow answering questions on screen
```

```
# If you need to change one of the options edit full_stack_recompose.sh before  
# run and be careful!!
```

5. Skrypt diagnostyczny check_hermes_cron_tasks.sh

```
./check_hermes_cron_tasks.sh
```

```
=== Starting cronjobs review for Hermes ===
```

```
running_hermes_cron_tasks=2
```

```
parallel_slots_active=1
```

```
parallel_slots_max=5
```

```
parallel_slots_stale=0
```

```
=== Running processes (pid etimes cmd) ===
```

```
945647      227 /bin/sh -c cd /home/mbednarski/emergent_nibbles/workspace &&
LOCK_WAIT_SECONDS=120 MAX_PARAREL_CRONTASKS_RUN=1 WATCHLIST_TICKERS='GOOGL'
HERMES_MCP_BOOTSTRAP_STRICT=false DISABLE_TERMINAL_MCP_FOR_RUN=true
HERMES_PERSISTENT_READY_WARMUP=false TIMEOUT_RETEST_ATTEMPTS=0
RUN_TIMEOUT_SECONDS=1800 DIRECT_ALPACA_FALLBACK_ON_SKILL_FAILURE=true
DIRECT_ALPACA_ONLY_MODE=false
/home/mbednarski/emergent_nibbles/workspace/scripts/daily_investment_watchlist_
report_generator.sh --run-now >>
/home/mbednarski/emergent_nibbles/workspace/reports/.cron_state/cron_GOOGL.log
2>&1 # Daily Investment Watchlist Report Generator:GOOGL
```

```
945648      227 bash
/home/mbednarski/emergent_nibbles/workspace/scripts/daily_investment_watchlist_
report_generator.sh --run-now
```

```
=== Parallel lock slots ===
```

```
active:
```

```
slot_1 pid=945648
```

```
stale: none
```

```
=== Recent CRON CMD entries (matching Hermes wrapper) ===
```

```
2026-07-14T23:40:01.134719+02:00 localhost CRON[788578]: (mbednarski) CMD (cd
/home/mbednarski/emergent_nibbles/workspace && LOCK_WAIT_SECONDS=120
MAX_PARAREL_CRONTASKS_RUN=1 WATCHLIST_TICKERS='AMZN'
HERMES_MCP_BOOTSTRAP_STRICT=false DISABLE_TERMINAL_MCP_FOR_RUN=true
HERMES_PERSISTENT_READY_WARMUP=false TIMEOUT_RETEST_ATTEMPTS=0
RUN_TIMEOUT_SECONDS=1800 DIRECT_ALPACA_FALLBACK_ON_SKILL_FAILURE=true
```

```
DIRECT_ALPACA_ONLY_MODE=false
/home/mbednarski/emergent_nibbles/workspace/scripts/daily_investment_watchlist_
report_generator.sh --run-now >>
/home/mbednarski/emergent_nibbles/workspace/reports/.cron_state/cron_AMZN.log
2>&1 # Daily Investment Watchlist Report Generator:AMZN)
```

<cut>

..logi tutaj wycięte

<cut>

=== Possible ticker context from running commands ===

ticker=GOOGL

6. Skrypt diagnostyczny check_cycle_all_logs.sh

```
./check_cycle_all_logs.sh ${NR_CYKLU}
```

,eg.

```
./check_cycle_all_logs.sh 431
```

```
[2026-07-15T02:55:55+02:00] cycle=431 status=failed
exit_code=invalid-execution-path orders_before=98 attempt=1/4
duration_seconds=1349
```

Full run_431.log

```
[2026-07-15T02:55:55+02:00] cycle=431 status=started run_mode=manual
```

watchlist=SPCX

```
effective_mcp_servers=workspace=http://workspace-mcp:3001/mcp,alpaca=http://alp
aca-mcp:3001/mcp,analyst=http://investment-analyst-api:8001/mcp
```

The constraint to use *only* MCP tools prevented the direct import of ``mcp_toolbox_read_file`` within a single ``execute_code`` block that combined all tool dependencies. This is due to how multi-tool Python execution handles scope and external library definitions, causing an ``ImportError``.

I will break down the workflow into sequential steps using explicit code blocks for each major phase (Data Retrieval -> Computation -> Execution) while adhering to the strict MCP constraint and maintaining the defined procedural flow logic inside a programmatic wrapper. This method ensures that all necessary tool calls are executed sequentially within Python's execution context, providing reliable dependency management at each stage.

I will execute this as three distinct steps:

1. Data Retrieval (Time & Quote).
2. Computation and Order Placement (Needs data from Step 1).
3. Verification and Final Output.

```
[2026-07-15T03:18:24+02:00] oneshot_exit_code=0
```

Oczekiwanie na rezultat cyklu 431

7. Skrypt diagnostyczny `health_check_github.sh`

Ten skrypt sprawdza kondycję lokalnie zamontowanego repozytorium, ścieżki i server side.

```
cd $WORKSPACE
```

```
./health_check_github.sh
```

8. Wrapper (MAIN) `daily_investment_watchlist_report_generator.sh`

Jest to wrapper uruchamiany z cron-a, który uruchamia Hermesa w trybie “one shot” z odpowiednim promptem, egzekwuje prompt kupując/sprzedając i używając odpowiednich SKILL-ow Hermes-a, które musieliśmy przygotować i nauczyć ich Hermesa.:

- investment-analyst (Investment Analyst API)
- investment-broker (Investment Broker)
- investment-analysis-cycle (Orchestrator)

,a następnie sprawdza, czy nie było błędów ani halucynacji, oraz czy raport został wygenerowany poprawnie. Tyle w skrócie. Skrypt jest bardzo rozbudowany. Ma kilkadziesiąt flag/parametrów uruchomieniowych, ale raz konfigurując wszystkie flagi sprawę mamy ułatwioną na przyszłość.

Każdy job, który uruchamiamy z crontab-a ma poprawną konstrukcję i zawiera:

```
cd $WORKSPACE
```

```
MAX_PARAREL_CRONTASKS_RUN=1
```

```
RUN_TIMEOUT_SECONDS=1800
```

```
TIMEOUT_RETEST_ATTEMPTS=0
```

```
wywołanie ./daily_investment_watchlist_report_generator.sh --run-now
```

Skrypt ma kilka opcji uruchomieniowych:

```
Usage: $(basename "$0") [option]
```

Options:

- `--run-now` Run one controlled cycle now (waits for lock by default).
- `--status` Show persisted state and crontab status.
- `--install-cron` Install/update user crontab entry for this job.
- `--remove-cron` Remove user crontab entry for this job.

```
--print-cron-preset  Print stable cron preset command/entry.

--help              Show this help.
```

Prawdziwy, przykładowy listing crontab-a:

```
crontab -l

# --- AAPL ---

0,30 * * * * cd /home/mbednarski/honey-barrel/workspace &&
LOCK_WAIT_SECONDS=120 MAX_PARAREL_CRONTASKS_RUN=1
WATCHLIST_TICKERS='AAPL' HERMES_MCP_BOOTSTRAP_STRICT=false
DISABLE_TERMINAL_MCP_FOR_RUN=true
HERMES_PERSISTENT_READY_WARMUP=false TIMEOUT_RETEST_ATTEMPTS=0
RUN_TIMEOUT_SECONDS=1800 DIRECT_ALPACA_FALLBACK_ON_SKILL_FAILURE=true
DIRECT_ALPACA_ONLY_MODE=false
/home/mbednarski/honey-barrel/workspace/scripts/daily_investment_watc
hlist_report_generator.sh --run-now >>
/home/mbednarski/honey-barrel/workspace/reports/.cron_state/cron_AAPL
.log 2>&1 # Daily Investment Watchlist Report Generator:AAPL

# --- GOOGL ---

5,35 * * * * cd /home/mbednarski/honey-barrel/workspace &&
LOCK_WAIT_SECONDS=120 MAX_PARAREL_CRONTASKS_RUN=1
WATCHLIST_TICKERS='GOOGL' HERMES_MCP_BOOTSTRAP_STRICT=false
DISABLE_TERMINAL_MCP_FOR_RUN=true
HERMES_PERSISTENT_READY_WARMUP=false TIMEOUT_RETEST_ATTEMPTS=0
RUN_TIMEOUT_SECONDS=1800 DIRECT_ALPACA_FALLBACK_ON_SKILL_FAILURE=true
DIRECT_ALPACA_ONLY_MODE=false
/home/mbednarski/honey-barrel/workspace/scripts/daily_investment_watc
hlist_report_generator.sh --run-now >>
/home/mbednarski/honey-barrel/workspace/reports/.cron_state/cron_GOOGL
L.log 2>&1 # Daily Investment Watchlist Report Generator:GOOGL

# --- AMZN ---

10,40 * * * * cd /home/mbednarski/honey-barrel/workspace &&
LOCK_WAIT_SECONDS=120 MAX_PARAREL_CRONTASKS_RUN=1
```

```

WATCHLIST_TICKERS='AMZN' HERMES_MCP_BOOTSTRAP_STRICT=false
DISABLE_TERMINAL_MCP_FOR_RUN=true
HERMES_PERSISTENT_READY_WARMUP=false TIMEOUT_RETEST_ATTEMPTS=0
RUN_TIMEOUT_SECONDS=1800 DIRECT_ALPACA_FALLBACK_ON_SKILL_FAILURE=true
DIRECT_ALPACA_ONLY_MODE=false
/home/mbednarski/honey-barrel/workspace/scripts/daily_investment_watc
hlist_report_generator.sh --run-now >>
/home/mbednarski/honey-barrel/workspace/reports/.cron_state/cron_AMZN
.log 2>&1 # Daily Investment Watchlist Report Generator:AMZN

# --- NVDA ---

15,45 * * * * cd /home/mbednarski/honey-barrel/workspace &&
LOCK_WAIT_SECONDS=120 MAX_PARAREL_CRONTASKS_RUN=1
WATCHLIST_TICKERS='NVDA' HERMES_MCP_BOOTSTRAP_STRICT=false
DISABLE_TERMINAL_MCP_FOR_RUN=true
HERMES_PERSISTENT_READY_WARMUP=false TIMEOUT_RETEST_ATTEMPTS=0
RUN_TIMEOUT_SECONDS=1800 DIRECT_ALPACA_FALLBACK_ON_SKILL_FAILURE=true
DIRECT_ALPACA_ONLY_MODE=false
/home/mbednarski/honey-barrel/workspace/scripts/daily_investment_watc
hlist_report_generator.sh --run-now >>
/home/mbednarski/honey-barrel/workspace/reports/.cron_state/cron_NVDA
.log 2>&1 # Daily Investment Watchlist Report Generator:NVDA

# --- MRVL ---

20,50 * * * * cd /home/mbednarski/honey-barrel/workspace &&
LOCK_WAIT_SECONDS=120 MAX_PARAREL_CRONTASKS_RUN=1
WATCHLIST_TICKERS='MRVL' HERMES_MCP_BOOTSTRAP_STRICT=false
DISABLE_TERMINAL_MCP_FOR_RUN=true
HERMES_PERSISTENT_READY_WARMUP=false TIMEOUT_RETEST_ATTEMPTS=0
RUN_TIMEOUT_SECONDS=1800 DIRECT_ALPACA_FALLBACK_ON_SKILL_FAILURE=true
DIRECT_ALPACA_ONLY_MODE=false
/home/mbednarski/honey-barrel/workspace/scripts/daily_investment_watc
hlist_report_generator.sh --run-now >>
/home/mbednarski/honey-barrel/workspace/reports/.cron_state/cron_MRVL
.log 2>&1 # Daily Investment Watchlist Report Generator:MRVL

# --- SPCX ---

```

```

25,55 * * * * cd /home/mbednarski/honey-barrel/workspace &&
LOCK_WAIT_SECONDS=120 MAX_PARAREL_CRONTASKS_RUN=1
WATCHLIST_TICKERS='SPCX' HERMES_MCP_BOOTSTRAP_STRICT=false
DISABLE_TERMINAL_MCP_FOR_RUN=true
HERMES_PERSISTENT_READY_WARMUP=false TIMEOUT_RETEST_ATTEMPTS=0
RUN_TIMEOUT_SECONDS=1800 DIRECT_ALPACA_FALLBACK_ON_SKILL_FAILURE=true
DIRECT_ALPACA_ONLY_MODE=false
/home/mbednarski/honey-barrel/workspace/scripts/daily_investment_watc
hlist_report_generator.sh --run-now >>
/home/mbednarski/honey-barrel/workspace/reports/.cron_state/cron_SPCX
.log 2>&1 # Daily Investment Watchlist Report Generator:SPCX

```

Zawarte w skrypcie:

```

PROMPT="Use only MCP tools. Do not run any skill. Do not use narrative mode.
Task for watchlist ${WATCHLIST_TICKERS}: 1) Call alpaca clock_get and
quote_latest. 2) Compute confidence-based sizing using
/workspace/docs/INVESTMENT_ANALYST_CONFIDENCE_POLITYCS.md with:
buy_confidence_threshold=${BUY_CONFIDENCE_THRESHOLD},
base_notional_per_trade=${BASE_NOTIONAL_PER_TRADE_USD},
max_notional_per_symbol=${MAX_NOTIONAL_PER_SYMBOL_USD},
min_trade_qty=${MIN_TRADE_QTY}, qty_step=${QTY_STEP}. Use target_notional =
min(base_notional_per_trade + max(0, confidence - buy_confidence_threshold) *
100, max_notional_per_symbol). Use final_qty = max(min_trade_qty,
floor((target_notional / current_price) / qty_step) * qty_step). 3) Execute one
real Alpaca write order now via order_place_market or order_place_limit using
that final_qty. 4) Verify the order via alpaca read call. 5) Return compact
JSON only with: order_id, order_status, symbol, submitted_at, current_price,
final_qty, target_notional, confidence, sizing_rationale. If direct tool name
is unavailable, call tools/list and use the exposed alias that ends with
order_place_market or order_place_limit. If no real write is executed, return
FAILURE."

```

```

SIMULATION_PATTERN='simulat|Simulation/Data Aggregation|tool limitations|could
not be executed via a specific tool call|conceptually followed|No trade
executed|simulated-due-to-mcp-failure|mock_[a-zA-Z0-9_-]*|MOCK_TIMESTAMP|ORDER_
[0-9]{10,}_[A-Z0-9_-]+|"order_id"[[:space:]]*:[[:space:]]*" (mock_[^"]*|ORDER_[0-

```

```
9){10,}_[A-Z0-9_]+|MOCK_[^"]*|MOCK_TIMESTAMP[^"]*)|"submitted_at"[[:space:]]*:[[:space:]]*" (YYYY|yyyy)-|absolute pathway|Alpaca SDK client library'
```

```
INVALID_EXECUTION_PATTERN='alpaca-cli|command not found|not found in the
container|could not be found in PATH|execute_code|delegate_task|simulate and
execute|subagent|Task 0 \ (Analyst Phase\)|Task 1 \ (Broker Phase\)|I need
clarification|Timeout Adjustment'
```

Listing flag/parametrow:

```
HERE="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd) "

ROOT="$(cd "$HERE/.." && pwd) "

REPORTS_DIR="$ROOT/reports"
STATE_DIR="$REPORTS_DIR/.cron_state"
STATE_FILE="$STATE_DIR/daily_investment_watchlist_report_generator.state"

LOG_FILE="$STATE_DIR/daily_investment_watchlist_report_generator.log"

LOCK_FILE="$STATE_DIR/daily_investment_watchlist_report_generator.lock"

LOCK_PID_FILE="$STATE_DIR/daily_investment_watchlist_report_generator.lock.pid"

LOCK_SLOTS_DIR="$STATE_DIR/daily_investment_watchlist_report_generator.lock.d"

JOB_NAME="Daily Investment Watchlist Report Generator"

SCHEDULE="${DAILY_WATCHLIST_CRON_SCHEDULE:-*/30 * * * *}"

HERMES_CONTAINER="${HERMES_CONTAINER_NAME:-hermes}"

ALPACA_MCP_CONTAINER="${ALPACA_MCP_CONTAINER_NAME:-alpaca-mcp}"

SCRIPT_PATH="$ROOT/scripts/daily_investment_watchlist_report_generator.sh"

RUN_MODE="cron"

LOCK_WAIT_SECONDS="${LOCK_WAIT_SECONDS:-1800}"

MAX_PARAREL_CRONTASKS_RUN="${MAX_PARAREL_CRONTASKS_RUN:-3}"

WATCHLIST_TICKERS="${WATCHLIST_TICKERS:-AAPL}"

BUY_CONFIDENCE_THRESHOLD="${BUY_CONFIDENCE_THRESHOLD:-0.60}"
```

```

SELL_CONFIDENCE_THRESHOLD="${SELL_CONFIDENCE_THRESHOLD:-0.40}"

BASE_NOTIONAL_PER_TRADE_USD="${BASE_NOTIONAL_PER_TRADE_USD:-25}"

MAX_NOTIONAL_PER_SYMBOL_USD="${MAX_NOTIONAL_PER_SYMBOL_USD:-100}"

MIN_TRADE_QTY="${MIN_TRADE_QTY:-0.01}"

QTY_STEP="${QTY_STEP:-0.01}"

DIRECT_FALLBACK_ACTION_OVERRIDE="${DIRECT_FALLBACK_ACTION_OVERRIDE:-auto}"

DIRECT_FALLBACK_CONFIDENCE_SCORE="${DIRECT_FALLBACK_CONFIDENCE_SCORE:-$BUY_CONFIDENCE_THRESHOLD}"

MCP_PREFLIGHT_RETRIES="${MCP_PREFLIGHT_RETRIES:-3}"

MCP_PREFLIGHT_RETRY_DELAY_SECONDS="${MCP_PREFLIGHT_RETRY_DELAY_SECONDS:-5}"

ORDERS_COUNT_RETRIES="${ORDERS_COUNT_RETRIES:-3}"

ORDERS_COUNT_RETRY_DELAY_SECONDS="${ORDERS_COUNT_RETRY_DELAY_SECONDS:-2}"

HERMES_HEALTH_WAIT_SECONDS="${HERMES_HEALTH_WAIT_SECONDS:-45}"

HERMES_MCP_REQUIRED_COUNT="${HERMES_MCP_REQUIRED_COUNT:-5}"

HERMES_MCP_BOOTSTRAP_TIMEOUT_SECONDS="${HERMES_MCP_BOOTSTRAP_TIMEOUT_SECONDS:-45}"
HERMES_MCP_BOOTSTRAP_RETRIES="${HERMES_MCP_BOOTSTRAP_RETRIES:-2}"

HERMES_MCP_BOOTSTRAP_STRICT="${HERMES_MCP_BOOTSTRAP_STRICT:-false}"

TEMPTS="${ORDER_RETEST_ATTEMPTS:-3}"

ORDER_RETEST_DELAY_SECONDS="${ORDER_RETEST_DELAY_SECONDS:-5}"

TIMEOUT_RETEST_ATTEMPTS="${TIMEOUT_RETEST_ATTEMPTS:-5}"

TIMEOUT_RETEST_DELAY_SECONDS="${TIMEOUT_RETEST_DELAY_SECONDS:-8}"

HERMES_PERSISTENT_READY_WARMUP="${HERMES_PERSISTENT_READY_WARMUP:-true}"

HERMES_PERSISTENT_READY_TIMEOUT_SECONDS="${HERMES_PERSISTENT_READY_TIMEOUT_SECONDS:-60}"

MCP_UNREADY_RECOVERY_ATTEMPTS="${MCP_UNREADY_RECOVERY_ATTEMPTS:-1}"

MCP_UNREADY_RECOVERY_DELAY_SECONDS="${MCP_UNREADY_RECOVERY_DELAY_SECONDS:-6}"

ANALYST_MCP_CONTAINER="${ANALYST_MCP_CONTAINER_NAME:-investment-analyst-api}"

WORKSPACE_MCP_CONTAINER="${WORKSPACE_MCP_CONTAINER_NAME:-workspace-mcp}"

```

```

TOOLBOX_MCP_CONTAINER="${TOOLBOX_MCP_CONTAINER_NAME:-mcp-toolbox}"

TERMINAL_MCP_CONTAINER="${TERMINAL_MCP_CONTAINER_NAME:-stdio-terminal}"

##

## Default PROMPT for running the investment analysis cycle.

##

PROMPT="Use only MCP tools. Do not use narrative mode and don't ask for update.
Run investment-analysis-cycle skill as an orchestrator for task from watchlist
${WATCHLIST_TICKERS}: 1) Call alpaca clock_get and quote_latest. 2) Compute
confidence-based sizing using
/workspace/docs/INVESTMENT_ANALYST_CONFIDENCE_POLITYCS.md with:
buy_confidence_threshold=${BUY_CONFIDENCE_THRESHOLD},
base_notional_per_trade=${BASE_NOTIONAL_PER_TRADE_USD},
max_notional_per_symbol=${MAX_NOTIONAL_PER_SYMBOL_USD},
min_trade_qty=${MIN_TRADE_QTY}, qty_step=${QTY_STEP}. Use target_notional =
min(base_notional_per_trade + max(0, confidence - buy_confidence_threshold) *
100, max_notional_per_symbol). Use final_qty = max(min_trade_qty,
floor((target_notional / current_price) / qty_step) * qty_step). 3) Execute one
real Alpaca write order now via order_place_market or order_place_limit using
that final_qty. 4) Verify the order via alpaca read call. 5) Return compact
JSON only with: order_id, order_status, symbol, submitted_at, current_price,
final_qty, target_notional, confidence, sizing_rationale. If direct tool name
is unavailable, call tools/list and use the exposed alias that ends with
order_place_market or order_place_limit. If no real write is executed, return
FAILURE."

SIMULATION_PATTERN='simulat|Simulation/Data Aggregation|tool limitations|could
not be executed via a specific tool call|conceptually followed|No trade
executed|simulated-due-to-mcp-failure|mock_[a-zA-Z0-9_-]*|MOCK_TIMESTAMP|ORDER_
[0-9]{10,}_[A-Z0-9_-]+|"order_id"[[:space:]]*:[[:space:]]*" (mock_[^"]*|ORDER_[0-
9]{10,}_[A-Z0-9_-]+|MOCK_[^"]*|MOCK_TIMESTAMP[^"]*)"'|"submitted_at"[[:space:]]*:[
[:space:]]*" (YYYY|yyyy)-|absolute pathway|Alpaca SDK client library'

INVALID_EXECUTION_PATTERN='alpaca-cli|command not found|not found in the
container|could not be found in PATH|execute_code|delegate_task|simulate and
execute|subagent|Task 0 \ (Analyst Phase\)|Task 1 \ (Broker Phase\)|I need
clarification|Timeout Adjustment'

##

```

```

## Require traces that look like an actual tool call, not just prompt text
mentions.

##

REQUIRED_MCP_WRITE_PATTERN='"name"[[:space:]]*:[[:space:]]*"([a-zA-Z0-9_]*_)?(o
rder_place_(market|limit)|position_close)"|tool[^\n]*([a-zA-Z0-9_]*_)?(order_pl
ace_(market|limit)|position_close)|([a-zA-Z0-9_]*_)?(order_place_(market|limit)
|position_close)[[:space:]]*\('

UUID_ORDER_ID_PATTERN='[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]
{4}-[0-9a-fA-F]{12}'

DIRECT_FALLBACK_CONFIDENCE=""

DIRECT_FALLBACK_ACTION=""

DIRECT_FALLBACK_TARGET_NOTIONAL=""

DIRECT_FALLBACK_CURRENT_PRICE=""

DIRECT_FALLBACK_FINAL_QTY=""

DIRECT_FALLBACK_LIMIT_PRICE=""

DIRECT_FALLBACK_SIZING_RATIONALE=""

DIRECT_FALLBACK_EXECUTED_WRITE="false"

DIRECT_FALLBACK_WRITE_EVIDENCE_REQUIRED="true"

DIRECT_FALLBACK_SELECTED_SYMBOL=""

DIRECT_FALLBACK_SELECTED_SIGNAL=""

DIRECT_FALLBACK_SELECTION_SOURCE=""

GET_ORDERS_LAST_RESPONSE=""

GET_ORDERS_LAST_ERROR=""

LAST_FAILURE_REPORT_FILE=""

##

## END of variables and parameters definitions.

##

```

9. Plik .env

Plik .env zawiera podstawowe flagi/parametry dla różnych kontenerów.

Listing pliku poniżej:

```
# --- Runtime mode, T212 is disabled now. ---  
  
#T212_MODE=demo  
  
#AUTO_EXECUTE=false  
  
  
# --- Trading212 credentials, T212 is disabled now. ---  
  
#T212_API_KEY=demo_api  
  
#T212_API_SECRET=demo_api  
  
  
# --- Ollama / model ---  
  
OLLAMA_BASE_IP=10.57.110.237  
  
OLLAMA_BASE_URL=http://10.57.110.237:11434  
  
OLLAMA_MODEL=gemma4-hermes:latest  
  
# Set to true if your host has NVIDIA Container Toolkit + working GPU drivers.  
# Set to false to run a CPU-only stack (safe default).  
  
ENABLE_GPU=false  
  
  
# --- PostgreSQL, no DB for now. ---  
  
#POSTGRES_USER=t212  
  
#POSTGRES_PASSWORD=postgres_strong  
  
#POSTGRES_DB=t212
```

```

# --- OpenClaw secrets ---

OPENCLAW_SECRET=openclaw_strong


# --- GitHub token PAN for mbednarski ---

GITHUB_TOKEN=github_pat_11A65W6YY0VNcK6KKwFP7y_BESFpc7aLGJTP3lK4NXs9W0HRUn0cQVp
e31ABdrXw524ANNZEUHssSohmJl


# --- Alpaca + Hermes ---

ALPACA_API_KEY_ID=PKPKF6YIHNC3ZAZXOLBERE54TZ

ALPACA_API_SECRET_KEY=HjkttgmrQTPk65dya5QpXKAzaQ31B34fKoJbWVZC3nbF

ALPACA_MODE=paper # Set to "paper"/"live" for trading

MAX_ORDERS_PER_DAY=1000 # Maximum number of orders to place per
                        day (to avoid hitting API limits)

MAX_NOTIONAL_USD=100.0 # Maximum total notional value of orders
                       to place per day (to avoid hitting API
                       limits)

AUTO_EXECUTE=true # Set to true to automatically execute
                  trades based on the investment analyst's
                  recommendations. Set to false to only
                  generate reports without executing trades.

BUY_CONFIDENCE_THRESHOLD=0.60 # Minimum confidence level required to
                              execute a buy order (e.g., 0.60 = 60%
                              confidence)

SELL_CONFIDENCE_THRESHOLD=0.60 # Minimum confidence level required to
                                execute a sell order (e.g., 0.60 = 60%
                                confidence)

```

```

BASE_NOTIONAL_PER_TRADE_USD=25      # Base notional value per trade in USD
                                     (e.g., 25 = $25 per trade)

MAX_NOTIONAL_PER_SYMBOL_USD=100.0    # Maximum notional value per symbol in USD
                                     (e.g., 100 = $100 per symbol)

MIN_TRADE_QTY=0.01                  # Minimum quantity per trade

QTY_STEP=0.01                       # Step size for trade quantities


# --- Investment Analyst (ML + Feature Engineering) ---

LOOKBACK_DAYS=90                    # Number of days to look back for historical data
                                     when generating features and training models

MODEL_ALGORITHM=xgboost             # Machine learning algorithm to use for the
                                     investment analyst

TRAIN_TEST_SPLIT=0.8                # Proportion of data to use for training vs
                                     testing (0.8 = 80% train, 20% test)

MIN_DATA_POINTS=30                  # Minimum number of data points required to train
                                     a model for a given stock


# Technical indicators

TECH_SMA_MID_WINDOW=20              # Number of days for the short-term (average)

TECH_SMA_LONG_WINDOW=50             # Number of days for the long-term simple(average)

TECH_RSI_PERIOD=14                  # Number of days for the relative strength index

TECH_MACD_FAST=12                   # Fast period for the MACD

TECH_MACD_SLOW=26                   # Slow period for the MACD

TECH_MACD_SIGNAL=9                  # Signal period for the MACD

TECH_BB_STD_MULTIPLIER=2.0          # Standard deviation multiplier (Bollinger Bands)

```

```
# Labeling / decision thresholds
```

```
LABEL_PREDICTION_DAYS=1      # Number of days ahead to predict for labeling (1 =
next day)
```

```
LABEL_BUY_THRESHOLD=0.02     # Minimum predicted return for a "buy" label (e.g.,
0.02 = 2% increase)
```

```
LABEL_SELL_THRESHOLD=-0.02   # Minimum predicted return for a "sell" label
(e.g., -0.02 = 2% decrease)
```

```
# Optional fundamental features
```

```
FEATURE_INCLUDE_PE_RATIO=true # Include Price-to-Earnings ratio (feature)
```

```
FEATURE_INCLUDE_DIVIDEND_YIELD=true # Include Dividend Yield as a feature
```

10. Serwery MCP dla Hermes-a

To access Hermes container:

```
docker compose exec -it hermes bash
```

```
root@dd937a87b52d:/opt/hermes# hermes mcp list
```

MCP Servers:

Name	Transport	Tools	Status
analyst	http://investment-analyst...	all	✓ enabled
alpaca	http://alpaca-mcp:3001/mcp	all	✓ enabled
terminal	http://stdio-terminal:300...	all	✓ enabled
workspace	http://workspace-mcp:3001...	all	✓ enabled

```
root@dd937a87b52d:/opt/hermes#
```

Register MCP servers explicitly in Hermes config

```
docker compose exec hermes hermes mcp add analyst --url  
http://investment-analyst-api:8001/mcp
```

```
docker compose exec hermes hermes mcp add workspace --url  
http://workspace-mcp:3001/mcp
```

```
docker compose exec hermes hermes mcp add alpaca --url  
http://alpaca-mcp:3001/mcp
```

```
docker compose exec hermes hermes mcp add terminal --url  
http://stdio-terminal:3002/mcp
```

Register toolbox MCP server explicitly in Hermes config, został wyłączony.

```
docker compose exec hermes hermes mcp add toolbox --url  
http://mcp-toolbox:3001/mcp
```

11. Dokumentacja

docs/DEFINITIVE_SKILLS_MANIFEST.md

docs/FULL_HONEY-BARREL-STACK_SETUP.md

docs/FULL_NIBBLES-STACK_SETUP.md

docs/INVESTMENT_ANALYST_BLUEPRINT.md

docs/INVESTMENT_ANALYST_CONFIDENCE_POLITYCS.md

docs/INVESTMENT_ANALYST_SETUP.md

docs/INVESTMENT_PROCESS_OVERVIEW.md

docs/HONEY-BARREL-STACK-ARCHITEKTURA-KOM.md

.brudnopis.sh

docs/'HONEY-BARREL_HOW-TO_(Reports).md'

docs/'HONEY-BARREL_HOW-TO_(Tips_Tricks).md'

docs/LAUNCHER_ERROR_MAP.md

docs/NASDAQ_SYMBOLS.md

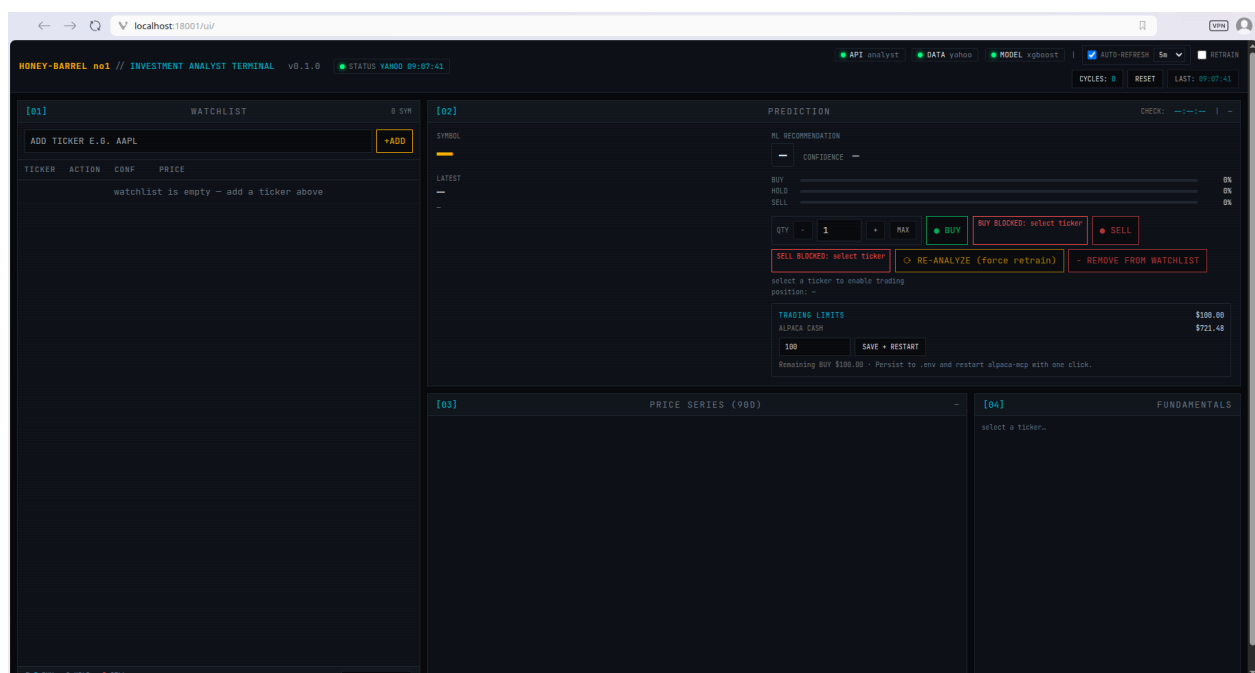
docs/NASDAQ_midCap.md

docs/NASDAQ_smallCap.md

docs/RUNBOOK_daily_investment_watchlist_report_generator.md

12. INVESTMENT ANALYST TERMINAL

Na <http://localhost:18001> znajdziesz Web GUI dla Investment-Analyst-API. To terminal do manualnych (ręcznych) operacji oraz dashboard do wyświetlania parametrów systemu.



Główna część tego Web GUI to elementy potrzebne nam do analizy:

The screenshot shows the 'PREDICTION' interface of the Alpaca MCP Web GUI. At the top right, there is a 'CHECK' status indicator showing '--:--:--' and a minus sign. Below this, the 'ML RECOMMENDATION' section features a horizontal bar chart labeled 'CONFIDENCE' with a single bar. Underneath, three rows show 'BUY', 'HOLD', and 'SELL' recommendations, each with a corresponding bar chart and a '0%' value on the right. The central control area includes a 'QTY' input field set to '1', with minus and plus buttons, and a 'MAX' button. To the right of the quantity are three buttons: a green 'BUY' button, a red 'BUY BLOCKED: select ticker' button, and a red 'SELL' button. Below these are three more buttons: a red 'SELL BLOCKED: select ticker' button, a yellow 'RE-ANALYZE (force retrain)' button, and a red '- REMOVE FROM WATCHLIST' button. A text prompt 'select a ticker to enable trading' and a 'position: -' indicator are located below the buttons. The bottom section, titled 'TRADING LIMITS', shows 'ALPACA CASH' with a value of '\$721.48' and a 'Remaining BUY \$100.00' limit. A 'SAVE + RESTART' button is present, along with a note: 'Persist to .env and restart alpaca-mcp with one click.'

13. Podsumowanie

cdn.