



Agent Zero v1 - Poradnik Deweloperski (Instalacja + Współpraca)

Wprowadzenie do projektu

Agent Zero v1 to zaawansowana platforma AI dla enterprise z systemem wieloagentowym z repozytorium: <https://github.com/HotelAiOS/agent-zero-v1>

Architektura systemu:

- **Backend:** Python (96.8%) - FastAPI, Neo4j, RabbitMQ
- **Frontend:** JavaScript, HTML, CSS - React Dashboard
- **Infrastruktura:** Docker, Kubernetes, PostgreSQL
- **AI:** Ollama, LangChain, multi-model support

Status aktualny: PHASE 1  (komunikacja multi-agent), PHASE 2  (Web Interface)

Wymagania systemowe

Sprzęt (minimum):

- **RAM:** 16GB (minimum 8GB)
- **Dysk:** 20GB wolnego miejsca
- **Procesor:** x64 z obsługą wirtualizacji

Oprogramowanie wymagane:

```
# Sprawdź wersje przed instalacją:  
git --version      # 2.30+  
python3 --version  # 3.10+
```

```
docker --version    # 20.10+
docker compose --version # 2.0+
node --version      # 18.0+
npm --version       # 8.0+
```

Instalacja - Instrukcje dla każdego OS

Arch Linux (zalecany dla projektu):

```
# 1. Aktualizacja systemu
sudo pacman -Syu

# 2. Instalacja podstawowych narzędzi
sudo pacman -S git python python-pip docker docker-compose nodejs npm

# 3. Instalacja AUR helper (yay)
sudo pacman -S base-devel git
git clone https://aur.archlinux.org/yay.git
cd yay && makepkg -si

# 4. Instalacja z AUR
yay -S neo4j-community ollama-bin

# 5. Konfiguracja Docker
sudo systemctl start docker
sudo systemctl enable docker
sudo usermod -aG docker $USER

# ⚠️ RESTART WYMAGANY po dodaniu do grupy docker
```

Ubuntu/Debian:

```
# 1. Aktualizacja
sudo apt update && sudo apt upgrade -y
```

2. Podstawowe narzędzia

```
sudo apt install -y git python3 python3-pip python3-venv curl
```

3. Docker

```
curl -fsSL https://get.docker.com -o get-docker.sh
```

```
sudo sh get-docker.sh
```

```
sudo usermod -aG docker $USER
```

4. Node.js

```
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
```

```
sudo apt install -y nodejs
```

⚠️ RESTART WYMAGANY

macOS:

1. Zainstaluj Homebrew

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

2. Instalacja narzędzi

```
brew install git python@3.11 nodejs npm
```

```
brew install --cask docker
```

3. Uruchom Docker Desktop

```
open /Applications/Docker.app
```

Windows (WSL2):

1. Włącz WSL2 w PowerShell (Admin)

```
wsl --install -d Ubuntu
```

2. W WSL2 Ubuntu wykonaj kroki jak dla Ubuntu wyżej

```
# 3. Docker Desktop for Windows z:  
# https://www.docker.com/products/docker-desktop/  
# ✅ Upewnij się, że WSL2 backend jest włączony
```



Klonowanie i konfiguracja projektu

Krok 1: Pobieranie kodu

```
# Przejdź do katalogu projektów  
cd ~/projects  
# lub stwórz jeśli nie istnieje:  
mkdir -p ~/projects && cd ~/projects  
  
# Klonuj repozytorium  
git clone https://github.com/HotelAiOS/agent-zero-v1.git  
cd agent-zero-v1  
  
# Sprawdź strukturę  
ls -la
```

Krok 2: Python Environment

```
# Stwórz środowisko wirtualne  
python3 -m venv venv  
  
# Aktywuj (wybierz odpowiedni dla systemu):  
source venv/bin/activate      # Linux/macOS/WSL  
source venv/bin/activate.fish # Fish shell (Arch)  
  
# Sprawdź czy aktywne  
which python # powinno pokazać ścieżkę do venv  
  
# Upgrade pip i instalacja zależności
```

```
pip install --upgrade pip
pip install -r requirements.txt
```

Krok 3: Docker Infrastructure

```
# Sprawdź Docker
docker --version
docker compose --version

# Uruchom usługi (PostgreSQL, RabbitMQ, Neo4j)
docker compose up -d

# Sprawdź status kontenerów
docker compose ps

# Powinny działać:
# ✅ postgres (port 5433)
# ✅ rabbitmq (porty 5672, 15672)
# ✅ neo4j (porty 7474, 7687)
```

Krok 4: Weryfikacja instalacji

```
# Test 1: Podstawowy test systemu
python test_simple.py

# Test 2: Komunikacja agentów (PHASE 1)
cd shared/communication
../venv/bin/python test_intelligent_agents.py

# Test 3: Pełna integracja
python test_full_integration.py

# ✅ Jeśli wszystkie testy przechodzą - gotowe!
```

Struktura projektu - Co gdzie znajdziesz

```
agent-zero-v1/
├── services/           # Mikroustugi
│   ├── agent-orchestrator/ # Orkiestracja agentów
│   ├── ai-router/       # Routing AI models
│   ├── api-gateway/     # API Gateway
│   └── chat-service/     # Chat interface
├── shared/             # Współdzielony kod ★
│   ├── communication/   # ✅ PHASE 1 - Komunikacja
│   ├── execution/       # Wykonywanie zadań
│   ├── knowledge/       # Neo4j integration
│   ├── llm/             # LLM clients
│   └── orchestration/   # Orkiestracja zadań
├── infrastructure/     # Docker, K8s configs
├── requirements.txt     # Python dependencies
├── docker-compose.yml  # Docker services
└── ROADMAP.md          # Plan rozwoju
```

Kluczowe pliki do poznania:

- `shared/communication/` - System komunikacji (PHASE 1 ✅)
- `shared/execution/` - Silnik wykonywania zadań
- `services/ai-router/` - Routing do LLM models
- `test_*.py` - Testy komponentów systemu

Workflow współpracy dla zespołu

Git Workflow (GitHub Flow)

```
# 1. Zawsze zaczynij od aktualnego main
git checkout main
git pull origin main
```

```
# 2. Utwórz branch dla zadania
git checkout -b feature/nazwa-funkcji
# lub:
git checkout -b fix/nazwa-naprawy

# 3. Pracuj i commituj regularnie
git add .
git commit -m "feat: opis zmian"

# 4. Push do remote branch
git push origin feature/nazwa-funkcji

# 5. Utwórz Pull Request na GitHub
# 6. Po merge usuń branch lokalnie
git checkout main
git pull origin main
git branch -d feature/nazwa-funkcji
```



Konwencje nazewnictwa

Branche:











- `feature/nazwa-funkcji` - nowe funkcje
- `fix/nazwa-naprawy` - naprawy błędów
- `refactor/nazwa` - refaktoryzacja
- `docs/nazwa` - dokumentacja

Commit messages:

```
feat: dodaj nową funkcję
fix: napraw błąd w komunikacji
docs: aktualizuj README
style: popraw formatowanie
refactor: przepisz agent factory
```

test: dodaj testy integracji
chore: aktualizuj dependencies

Podział ról zespołowych

Developer A - Backend	Developer B - Frontend
 Architektura systemu	 React Dashboard
 Agent Factory & Lifecycle	 WebSocket real-time
 Neo4j/RabbitMQ integration	 Backend API integration
 FastAPI endpoints	 User Experience & UI
 Backend testing	 Frontend testing

Proces pierwszej współpracy

Daily Standup (10 min codziennie)

Format:

1. Co robiłem wczoraj?
2. Co będę robić dziś?
3. Czy są blokery?
4. Czy potrzebuję pomocy?

Weekly Planning (30 min/tydzień)

- Przegląd zadań z Notion
- Planowanie sprintu
- Synchronizacja frontend ↔ backend

Pierwszy wspólny task - przykład

Scenariusz: Dodanie nowego API endpoint + frontend integration

Backend (Dev A):


```
# 1. Tworzy branch
git checkout -b feature/agents-status-api

# 2. Implementuje endpoint
# services/api-gateway/routes/agents.py

# 3. Dodaje testy
# tests/test_agents_api.py

# 4. Tworzy PR z dokumentacją
```

Frontend (Dev B):

```
# 1. Czeka na info o gotowym API
# 2. Tworzy service do komunikacji
# 3. Integruje z React komponentem
# 4. Testuje end-to-end
```

Komunikacja w zespole:

```
... Dev A: "API /agents/status gotowy, dokumentacja w PR #123"
... Dev B: "Dzięki! Zaczynam integrację z dashboard"
... Dev A: "Pamiętaj o error handling dla 500 status"
... Dev B: "OK, dodaję try/catch bloki"
```



Rozwiązywanie typowych problemów



Docker Issues

Problem: Containers nie startują

```
# Sprawdź logi
docker compose logs
```

```
# Restart wszystkiego
docker compose down
docker compose up -d

# Wyczyść system (ostateczność)
docker system prune -a
```

Problem: Port conflicts

```
# Sprawdź zajęte porty
netstat -tulpn | grep :5432
netstat -tulpn | grep :7474

# Zmień porty w docker-compose.yml
```

Python Issues

Problem: Import errors

```
# Sprawdź aktywne venv
which python # musi pokazać path do venv

# Reinstaluj dependencies
pip install -r requirements.txt --force-reinstall
```

Problem: Neo4j connection failed

```
# Test Neo4j
curl http://localhost:7474

# Sprawdź logi
docker compose logs neo4j
```

Git Issues

Problem: Merge conflicts

Pobierz najnowsze

git fetch origin

Rebase (czystsza historia)

git rebase origin/main

Rozwiąż konflikty ręcznie, potem:

git add .

git rebase --continue

✨ Najlepsze praktyki development

🔍 Code Quality

Uruchom przed każdym commit:

black shared/ # formatowanie

ruff shared/ # linting

mypy shared/ # type checking

pytest shared/communication/ # testy

📖 Documentation

Używaj type hints i docstrings

def create_agent(agent_type: str, capabilities: List[str]) → AgentInfo:

"""

Create new agent with specified capabilities.

Args:

agent_type: Type of agent (backend, frontend, etc.)

capabilities: List of agent capabilities

Returns:

AgentInfo: Created agent information

Raises:

```
ValueError: If agent_type is invalid
"""
pass
```

Testing Strategy

```
# Unit testy
def test_agent_creation():
    agent = create_agent("backend", ["python"])
    assert agent.agent_type == "backend"





# Integration testy
async def test_agent_communication():
    backend = IntelligentAgent("backend_001", "backend", ["python"])
    await backend.start()
    # test communication...
```

Kontakt i wsparcie

Gdzie szukać pomocy:

- **GitHub Issues:** <https://github.com/HotelAiOS/agent-zero-v1/issues>
- **Notion Tasks:** Strony z zadaniami dla Dev A i Dev B
- **Documentation:** [shared/AI_SYSTEM_README.md](#)

Następne kroki:

1.  Zainstaluj środowisko według instrukcji
2.  Uruchom testy weryfikacyjne
3.  Wybierz pierwsze zadanie z Notion
4.  Rozpocznij współpracę!

Powodzenia w rozwoju Agent Zero v1! 🚀

Aktualizacja: 7 października 2025 | Wersja 1.0 | Agent Zero v1 Enterprise Platform