

# ARCHITECTURE DECISIONS (ADR) - Agent Zero v1

**Purpose:** Dokumentacja kluczowych decyzji technicznych i architektonicznych w projekcie Agent Zero v1

**Format:** Bazowany na MADR (Markdown Architecture Decision Records).

## INDEKS DECYZJI

ID	Data	Tytuł	Status	Owner
ADR-001	2025-10-07	Wybór Neo4j jako głównej bazy wiedzy	✓ Accepted	Dev Team
ADR-002	2025-10-07	RabbitMQ vs Redis dla komunikacji agentów	✓ Accepted	Dev Team
ADR-003	2025-10-07	Ollama jako LLM inference engine	✓ Accepted	Dev Team
ADR-004	2025-10-07	FastAPI framework dla REST API	✓ Accepted	Dev Team
ADR-005	2025-10-07	React dla frontend dashboard	✓ Accepted	Dev Team
ADR-006	2025-10-07	Docker Compose vs Kubernetes dla dev	✓ Accepted	Dev Team
ADR-007	2025-10-07	Python type hints enforcement	✓ Accepted	Dev Team

## ADR-001: Wybór Neo4j jako głównej bazy wiedzy

### Status

✅ **Accepted** - 7 października 2025

## Context

Potrzebujemy systemu przechowywania wiedzy dla agentów AI, który umożliwi:

- Przechowywanie relacji między conceptami
- Szybkie zapytania o powiązane informacje
- Skalowanie w miarę wzrostu wiedzy
- Integracja z systemem uczenia się agentów

## Decision

Wybieramy **Neo4j** jako główną bazę danych do przechowywania wiedzy agentów.

## Rationale

### Pros Neo4j:



- Naturalne reprezentowanie relacji (graf vs tabele)
- Cypher query language - intuicyjny dla complex relationships
- Excellent performance dla graph traversal
- Built-in graph algorithms (shortest path, centrality)
- Strong consistency model
- ACID transactions support

### Cons Neo4j:



- Learning curve dla team (nowa technologia)
- Licencja - Community vs Enterprise
- Potencjalne performance issues przy bardzo dużych datasetów
- Dodatkowa infrastruktura do zarządzania

## Alternatives Considered



### PostgreSQL z JSONB:

-  Pros: Znana technologia, ACID, mature ecosystem
-  Cons: Gorsze performance dla complex relationships, brak native graph queries

#### **MongoDB:**

-  Pros: Document model, horizontal scaling
-  Cons: Eventual consistency, brak native relationships

#### **Redis Graph:**

-  Pros: In-memory performance, simple setup
-  Cons: Limited persistence options, smaller community

## **Consequences**

#### **Positive:**

- Optimal performance dla knowledge graph operations
- Intuitive modeling agent relationships
- Rich ecosystem graph algorithms
- Future-proof dla AI/ML features

#### **Negative:**

- Team musi się nauczyć Cypher
- Dodatkowa infrastruktura complexity
- Potencjalne licensing costs w przyszłości

#### **Mitigation:**

- Zapewnić training team w Cypher
- Zacząć od Community Edition
- Monitoring performance od początku

---

## **ADR-002: RabbitMQ vs Redis dla komunikacji agentów**

## Status

✓ **Accepted** - 7 października 2025

## Context

Systema agentów wymaga niezawodnej komunikacji async między wieloma agentami:

- Message routing based on capabilities
- Persistence wiadomości during failures
- Load balancing między agentami
- Dead letter handling

## Decision

Wybieramy **RabbitMQ** jako message broker dla komunikacji między agentami.

## Rationale

### Pros RabbitMQ:



- Rich routing capabilities (topic, direct, fanout)
- Message persistence i durability
- Dead letter exchange for failed messages
- Management UI dla monitoring
- Mature, battle-tested w enterprise
- Excellent Python client (pika)

### Cons RabbitMQ:



- Dodatkowa infrastruktura
- Memory usage może być wysoki
- Potential single point of failure (bez clustering)

## Alternatives Considered



### Redis Pub/Sub:

-  Pros: Simpler setup, high performance
-  Cons: No message persistence, limited routing

#### **Apache Kafka:**

-  Pros: High throughput, excellent for streaming
-  Cons: Overengineering dla naszego use case, complex setup

#### **In-memory queues:**

-  Pros: Fastest, no external deps
-  Cons: Brak persistence, nie skaluje między processes

### **Consequences**

#### **Positive:**

- Reliable message delivery
- Flexible routing dla różnych typów agentów
- Good monitoring capabilities
- Production-ready reliability


#### **Negative:**

- Dodatkowa infrastruktura do maintain
- Resource overhead
- Potential latency vs in-memory solutions



## **ADR-003: Ollama jako LLM inference engine**

### **Status**

 **Accepted** - 7 października 2025

### **Context**

Potrzebujemy local LLM inference dla:

- Code generation przez agentów

- Decision making w agentach
- Privacy i control nad AI models
- Cost optimization vs cloud APIs

## Decision

Wybieramy **Ollama** jako primary LLM inference engine z support dla multiple models.

## Rationale

### Pros Ollama:



- Local execution - no API costs, privacy
- Support dla multiple models (Llama, CodeLlama, Mistral)
- Simple API compatible z OpenAI format
- Good performance on consumer hardware
- Active development i community

### Cons Ollama:



- Requires significant local resources
- Model quality może być niższa niż GPT-4
- Limited context window w porównaniu do cloud

## Alternatives Considered



### OpenAI API:

-  Pros: Best model quality, large context
-  Cons: Expensive at scale, privacy concerns, API dependency

### Hugging Face Transformers:

-  Pros: Full control, open source models
-  Cons: Complex setup, optimization challenges

### Local GPU inference (custom):

-  Pros: Maximum performance control
-  Cons: Significant development overhead

## Consequences

### Positive:

- No ongoing API costs
- Full privacy control
- Predictable performance
- Offline capability

### Negative:


- Hardware requirements
- Model management complexity
- Potential quality limitations

### Mitigation:

- Hybrid approach - Ollama dla dev, cloud backup dla production
  - Monitoring model performance
  - Easy switching mechanism
- 

## ADR-004: FastAPI framework dla REST API

### Status

 **Accepted** - 7 października 2025

### Context

Potrzebujemy web framework dla:

- REST API endpoints
- WebSocket support dla real-time
- Automatic API documentation

- High performance async support

## Decision

Wybieramy **FastAPI** jako primary web framework.

## Rationale

### Pros FastAPI:



- Modern async/await support
- Automatic OpenAPI documentation
- Type hints integration
- High performance (comparable to Node.js)
- Excellent WebSocket support
- Growing ecosystem

### Cons FastAPI:



- Relatively new (less mature than Flask/Django)
- Smaller community than Flask

## Alternatives Considered



### Flask:

-  Pros: Mature, large community, flexible
-  Cons: No native async, manual documentation

### Django REST Framework:

-  Pros: Very mature, comprehensive features
-  Cons: Heavy for our use case, limited async

### Quart:

-  Pros: Flask-like API with async
-  Cons: Smaller community, less documentation



## Consequences

### Positive:

- Excellent performance dla real-time features
- Automatic API docs generation
- Modern development experience
- Easy async integration

### Negative:

- Potential breaking changes (newer framework)
  - Learning curve dla team
- 

## ADR-005: React dla frontend dashboard

### Status

✅ **Accepted** - 7 października 2025

### Decision

Wybieramy **React** z TypeScript dla frontend dashboard.

### Rationale

- Large ecosystem i community
- Excellent real-time capabilities
- Strong TypeScript support
- Good performance dla complex UIs
- Team expertise available

**Alternatives:** Vue.js, Angular, Svelte

---

## ADR-006: Docker Compose vs Kubernetes dla development

## Status

✅ **Accepted** - 7 października 2025

## Decision

Używamy **Docker Compose** dla development environment, **Kubernetes** dla production.

## Rationale

- Docker Compose: Simple setup, fast development
  - Kubernetes: Scalability, production features
  - Clear separation development vs production needs
- 

## ADR-007: Python type hints enforcement

### Status

✅ **Accepted** - 7 października 2025

### Decision

Enforce Python type hints across codebase z **mypy** checking.

### Rationale

- Better code documentation
  - Early error detection
  - Improved IDE support
  - Easier refactoring
  - Team collaboration benefits
- 

## PENDING DECISIONS

### ADR-008: Testing Framework Strategy (TBD)

**Context:** Czy użyć pytest vs unittest, jak testować async code, integration testing approach

### **ADR-009: Deployment Strategy (TBD)**

**Context:** Self-hosted vs cloud, CI/CD pipeline approach, environment management

### **ADR-010: Monitoring & Observability (TBD)**

**Context:** Prometheus vs alternatives, logging strategy, tracing approach

---

## **REVIEW PROCESS**

### **Creating ADR:**

1. Identify significant architectural decision
2. Research alternatives thoroughly
3. Document using ADR template
4. Team review i discussion
5. Final decision i status update

### **Review Schedule:**

- **Monthly:** Review existing ADRs relevance
  - **Quarterly:** Update consequences based on experience
  - **As needed:** When major changes affect existing decisions
- 

*ADR Last Updated: 7 października 2025, 12:59 CEST*

*Next Review: 7 listopada 2025*

*Owner: Development Team*