Agent2Agent (A2A) Protocol

With Registry Integration

Building Intelligent Multi-Agent Systems

What is A2A?

A protocol for **agent-to-agent communication** that enables:

- 🤝 Dynamic service discovery
- 💓 Health monitoring
- 🔍 Capability-based routing
- 📋 Agent identity cards
- 🌐 Distributed agent networks

Architecture Overview

```
┌──────────┐
│ Registry │  ← Central directory
└────┬─────┘
     │
     ├─→ Agent 1 (crypto prices)
     ├─→ Agent 2 (weather data)
     └─→ Agent 3 (news feeds)
```

Clients discover agents dynamically by capability

Key Components

1. **Agent Registry**
    - Central service directory
    - Health monitoring via heartbeats
    - Capability-based discovery
2. **A2A Protocol**
    - HANDSHAKE / HANDSHAKE_ACK
    - REQUEST / RESPONSE
    - ERROR / GOODBYE

Agent Registry Features

✅ **Service Discovery** - Find agents by capability
✅ **Health Monitoring** - Automatic heartbeat tracking
✅ **RESTful API** - Standard HTTP endpoints
✅ **Fast & Lightweight** - In-memory storage
✅ **Easy Integration** - FastAPI + Pydantic

Example: Crypto Price Agent

Without Registry (Old Way)

```
# Hardcoded connection
client.connect("localhost", 8888)
```

With Registry (New Way)

```
# Dynamic discovery
agents = await registry.discover(capability="get_price")
client.connect(agents[0].endpoint)
```

Message Flow

1. **Agent Startup**
    - Registers with registry
    - Declares capabilities
    - Starts heartbeat
2. **Client Discovery**
    - Queries registry for capability
    - Gets list of healthy agents
    - Connects to selected agent
3. **Communication**
    - Handshake exchange

Registry API Endpoints

Method	Endpoint	Purpose
POST	`/agents/register`	Register new agent
GET	`/agents/discover`	Find by capability
PUT	`/agents/{id}/heartbeat`	Health check
GET	`/agents/{id}`	Get agent details
DELETE	`/agents/{id}`	Unregister agent

Agent Card Structure

```json
{
  "agent_id": "crypto-agent-001",
  "name": "CryptoPriceAgent",
  "version": "1.0.0",
  "capabilities": ["get_price", "list_currencies"],
  "supported_protocols": ["A2A/1.0"],
  "metadata": {
    "supported_currencies": ["BTC", "ETH", "XRP"]
  }
}
```

Health Monitoring

Automatic Health Tracking

- Agents send heartbeats every **30 seconds**
- Registry marks agents unhealthy after **90 seconds**
- Discovery excludes unhealthy agents
- Automatic cleanup of stale registrations

```
Agent → Registry: PUT /agents/{id}/heartbeat
        (every 30s)
```

Example Project Structure

```
a2a_crypto_simple_registry_example/
├── registry/              # Registry server
│   ├── registry_server.py
│   ├── models.py
│   └── storage.py
├── server/                # Crypto agent
│   └── crypto_agent_server.py
├── client/                # Client app
│   └── a2a_client.py
└── shared/                # A2A protocol
    └── a2a_protocol.py
```

Quick Start

Step 1: Start Registry

```
cd registry
python registry_server.py
# Runs on http://localhost:8000
```

Step 2: Start Agent

```
cd server
python crypto_agent_server.py
# Registers with registry automatically
```

Quick Start (continued)

Step 3: Run Client

```
cd client
python a2a_client.py
# Discovers agent and connects
```

View All Agents

```
python a2a_client.py --list
```

Technology Stack

- **Registry**: FastAPI + Uvicorn
- **Data Validation**: Pydantic
- **Storage**: In-memory (extensible to Redis/PostgreSQL)
- **Protocol**: JSON over TCP
- **HTTP Client**: httpx
- **Language**: Python 3.8+

Benefits of Registry Pattern

Before (Hardcoded)

❌ Client needs to know agent addresses
❌ Can't add agents dynamically
❌ No health awareness
❌ Manual management

After (Registry)

✅ Dynamic discovery
✅ Automatic health monitoring
✅ Easy scaling
✅ Zero-config agents

Use Cases

1. Multi-Agent Workflows

- Price aggregation from multiple sources
- Distributed data collection
- Parallel task processing

2. Service Mesh

- Microservices discovery
- Load balancing
- Failover and redundancy

3. AI Agent Networks

- Specialized agents for different tasks

Real-World Example

Cryptocurrency Price System

```
Registry
├─→ Crypto Agent #1 (BTC, ETH) - US Data Center
├─→ Crypto Agent #2 (XRP, DOGE) - EU Data Center
└─→ Crypto Agent #3 (All coins) - Asia Data Center

Client discovers all "get_price" agents
→ Selects closest/fastest
→ Automatic failover if one goes down
```

Scalability

Current Implementation

- ✅ 100s of agents: Excellent
- ⚠️ 1000s of agents: Acceptable
- ❌ 10,000s+ agents: Need distributed solution

Production Enhancements

- Distributed registry (Consul, etcd)
- Persistent storage (PostgreSQL, Redis)
- Authentication & authorization
- Rate limiting
- TLS encryption

Code Quality

- 📝 **Well-Documented**: Every function has docstrings
- ✅ **Type Hints**: Full type annotations
- 🧪 **Tested**: Comprehensive test suite
- 🎨 **Clean Code**: PEP 8 compliant
- 📦 **Modular**: Clear separation of concerns

Total Lines: ~1,100 (registry + integration)

Getting Started

Requirements

```
pip install fastapi uvicorn pydantic httpx
```

Clone & Run

```
git clone [your-repo]
cd a2a_crypto_simple_registry_example

# Terminal 1
cd registry && python registry_server.py

# Terminal 2
cd server && python crypto_agent_server.py

# Terminal 3
cd client && python a2a_client.py
```

Documentation

📚 **Comprehensive Guides Included:**

- `QUICK_START_CHECKLIST.md` - Step-by-step setup
- `INTEGRATION_GUIDE.md` - Detailed modifications
- `BEFORE_AFTER_COMPARISON.md` - Visual comparisons
- `ARCHITECTURE.md` - System design
- Full API documentation at `/docs` endpoint

Project Features

✨ **Current Features:**

- Service registry with REST API
- Health monitoring with heartbeats
- Capability-based discovery
- Agent Cards for identity
- Crypto price agent example
- Interactive client application

🚀 **Coming Soon:**

- Authentication & authorization
- Persistent storage options
- Load balancing

Learning Outcomes

By working with this project, you'll understand:

1. **Service Discovery Patterns** - How distributed systems find services
2. **Health Monitoring** - Keeping track of service availability
3. **REST API Design** - Building production APIs with FastAPI
4. **Agent Communication** - Structured message protocols
5. **Distributed Systems** - Coordination in multi-agent environments

Comparison: A2A vs Other Patterns

Pattern	Use Case	Complexity
A2A	Agent networks	Medium
HTTP REST	Web APIs	Low
gRPC	High-performance	Medium
Message Queue	Async processing	High
WebSockets	Real-time	Medium

A2A is ideal for AI agent coordination

Integration Examples

With Existing Systems

```
# Register your existing service as an A2A agent
agent_card = {
    "agent_id": "my-service-001",
    "capabilities": ["process_data", "analyze"],
    ...
}
await registry.register(agent_card)
```

With LLMs

```
# LLM can discover and call agents
agents = await registry.discover("get_weather")
result = await agents[0].request(location="NYC")
```

Testing

Registry Tests

```
cd registry
python test_registry_simple.py
# Runs 8 tests covering all endpoints
```

Integration Tests

- Agent registration
- Health monitoring
- Discovery queries
- Client connection
- Price requests

Performance

Benchmarks (In-Memory)

- **Registration**: ~5ms
- **Discovery**: ~10ms
- **Heartbeat**: ~2ms
- **Concurrent Requests**: 100s/second

Resource Usage

- **Memory**: ~50MB (100 agents)
- **CPU**: <5% (idle)
- **Network**: Minimal (JSON over HTTP)

Security Considerations

Current (Training Version)

⚠️ No authentication

⚠️ HTTP only (no TLS)

⚠️ Basic validation

Production Recommendations

✅ API keys or OAuth 2.0

✅ HTTPS/TLS encryption

✅ Input sanitization

✅ Rate limiting

✅ Certificate verification

✅ Audit logging

Extensibility

Easy to Extend

```python
# Add new capability
agent_card.capabilities.append("new_feature")

# Custom storage backend
class RedisStorage(RegistryStorage):
    async def register_agent(self, ...):
        # Use Redis instead of memory

# Custom health checks
class AdvancedHealthMonitor(HealthMonitor):
    async def check_agent_health(self, agent):
        # Ping agent directly
```

Community & Support

Resources

- 📖 Full documentation in repository
- 💬 Issues and discussions on GitHub
- 🎓 Complete learning materials included
- 📧 Example code and templates

Contributing

Pull requests welcome!

- Bug fixes
- New features
- Documentation improvements

Roadmap

Phase 1 (Current) ✅

- Basic registry implementation
- Health monitoring
- Simple discovery
- Example crypto agent

Phase 2 (Next)

- Authentication layer
- Persistent storage
- Performance optimizations
- More example agents

Success Stories

Educational Use

Perfect for teaching:

- Distributed systems concepts
- Service-oriented architecture
- API design patterns
- Python async programming

Prototyping

Ideal for rapid prototyping of:

- Multi-agent AI systems
- Microservice architectures

Comparison with Other Solutions

vs. Consul

- ✅ Simpler to set up
- ✅ Python-native
- ❌ Less feature-rich
- ❌ Not production-scale

vs. etcd

- ✅ Easier to understand
- ✅ Agent-focused
- ❌ Single-node only
- ❌ Fewer guarantees

A2A Registry: Perfect for learning & prototyping

Key Takeaways

1. **Service Discovery** - Agents find each other dynamically
2. **Health Monitoring** - System knows what's available
3. **Loose Coupling** - Agents don't need hardcoded addresses
4. **Scalability** - Easy to add new agents
5. **Maintainability** - Clear, documented code

Perfect foundation for multi-agent systems!

Demo Time! 🎥

Live Demo Flow:

1. Start registry server
2. Register crypto agent
3. Watch heartbeats
4. Run client discovery
5. Request crypto prices
6. Stop agent → Watch health status
7. Restart agent → Automatic recovery

Try It Yourself

Hands-On Exercise

1. Clone the repository
2. Follow the QUICK_START_CHECKLIST.md
3. Run all three components
4. Experiment with modifications:
   - Add a second agent on different port
   - Create agent with new capability
   - Modify discovery filters

Time needed: 15-30 minutes

Questions?

Contact & Resources

- 📂 **Repository**: [Your GitHub URL]
- 📖 **Docs**: See README.md and guides
- 💡 **Examples**: Check `/examples` directory
- 🐛 **Issues**: GitHub Issues
- 💬 **Discussions**: GitHub Discussions

Thank You!

Start Building with A2A

Ready to create intelligent agent networks?

```
git clone [your-repo]
cd a2a_crypto_simple_registry_example
# Follow QUICK_START_CHECKLIST.md
```

Happy Coding! 🚀

Appendix: Additional Resources

Further Reading

- A2A Protocol Specification
- FastAPI Documentation
- Distributed Systems Patterns
- Microservices Architecture

Related Projects

- Model Context Protocol (MCP)
- OpenAI Swarm
- LangChain Agents
- AutoGen Framework

Appendix: Troubleshooting

Common Issues

"Could not connect to registry"

→ Ensure registry running on port 8000

"No agents found"

→ Check agent registered: `curl localhost:8000/agents`

"Agent unhealthy"

→ Verify heartbeats in agent terminal

"Import error"

→ Install dependencies: `pip install -r requirements.txt`

See troubleshooting guide for more →