Agent-Based Microservice System Resilience under DDIL Conditions

Student: Tuan Nguyen

Supervisor: Zihan Zawad, Sree Ram Boyapati

Topics in Computer Science Advanced

University of Adelaide

Link to the paper

Let's get started

CONTENT

1. Introduction and Motivation
2. Project Evaluation
3. Success Metric
4. Progresses:
    - Result
    - Challenge
1. Significance

Home › Agent and Multi-Agent Systems: Technology and Applications › Conference paper

Microservices as Agents in IoT Systems

Petar Krivic, Pavle Skocir, Mario Kusek & Gordan Jezic

Conference paper | First Online: 24 May 2017

Part of the Smart Innovation, Systems and Technologies book series (SIST,volume 74)

Abstract

Developing robust monolith systems has achieved its limitations, since the implementation of changes in today's large, complex, and fast evolving systems would be too slow and inefficient. As a response to these problems, microservice architecture emerged, and quickly became a widely used solution. Such modular architecture is appropriate for distributed environment of Internet of Things (IoT) solutions. In this paper we present a solution for service management on Machine-to-Machine (M2M) devices within IoT system by using collaborative microservices. Collaboration of distributed modules highly reminds of multi-agent systems where autonomous agents also cooperate to provide services to the end-user. Because of these similarities we consider microservices as modern agents that could improve systems in distributed environments, such as IoT.

RESEARCH-ARTICLE

MAMS: Multi-Agent MicroServices*

Authors: Rem W. Collier, Eoin O'Neill, David Lillis, Gregory O'Hare Authors Info & Claims

WWW '19: Companion Proceedings of The 2019 World Wide Web Conference • May 2019 • Pages 655–662 • https://doi.org/10.1145/3308560.3316509

Published: 13 May 2019 Publication History

Check for updates

ABSTRACT

This paper explores the intersection between microservices and Multi-Agent Systems (MAS), introducing the notion of a new approach to building MAS known as Multi-Agent MicroServices (MAMS). Our approach is illustrated through a worked example of a Vickrey Auction implemented as a microservice.

Data-driven Adaptation in Microservice-based IoT Architectures

Publisher: IEEE

Cite This PDF

Martina De Sanctis ; Henry Muccini ; Karthik Vaidhyanathan All Authors

Abstract:

Architecting self-adaptive Internet of Things (IoT) systems pose a lot of challenges due to heterogeneity, resource constraints, interoperability, etc. Although microservice architectures (MSA) emerged as a popular solution for developing next generation IoT systems, they further increase these challenges. This can be attributed to the complexity involved in managing adaptation concerns arising at different levels: i) IoT devices level, due to open and changing contexts, resource constraints, etc; ii) microservices level, due to dynamic resource demands; iii) application level itself, due to the changing user goals. In fact, recent studies have shown that traditional self-adaptation techniques are not flexible enough to be applied to MSA based systems. Moreover, what proposed in the literature handles adaptation either at the architectural level or at the application level. Towards this direction, we propose a self-adaptive architecture for microservice-based IoT systems. In particular, the architecture supports data-driven adaptations, by also leveraging machine learning techniques, and handles adaptations at different levels in a different manner: i) at device level, through a fog layer; ii) at microservice level, by leveraging the use of service mesh; iii) at application level, by means of dynamic QoS-aware service composition.

Microservice and Agent based system

Prior studies largely focus on centralized or microservice systems. However, understanding and quantifying performance under DDIL conditions remains a challenge. Our project addresses these very gaps.

Agent-based and microservice systems work together in DDIL condition?

Introduction

Experiment Focus

Center around improving the resilience of an agent-based system, specifically under the DDIL scenarios.

Method

Implement different method such as retry, failover to improve the system

Primary Objective

Delve into the system's performance and resilience under these conditions, providing us with insights into how it navigates such hurdles.

Practical Implications

This study advances theory and practice by optimizing agent-based microservice system resilience

Summary - Success Matrices

Agent based Micoservice System

Strategies are implemented to augment the system's reliability and effective handling of potential failures through advanced methods such as retry mechanisms, failover strategies, and prioritized message routing.

Our Approach

1. Hybrid Approach: We combine microservices' independence with agent-based autonomy to meet our use case needs.

2. Microservices Principle: We've used independently deployable services for system agility and scalability.

3. Agent-based Adaptation: Each service has unique roles and tasks for informed, autonomous decisions.

4. Final System: A resilient, scalable system with microservices' flexibility and agent-based decision-making, tailored for optimal performance under DDIL conditions.

Result - Architectural Overview

User (Client) -> Express Server (app.js Docker Container)
  -> Sends HTTP requests (GET/POST)

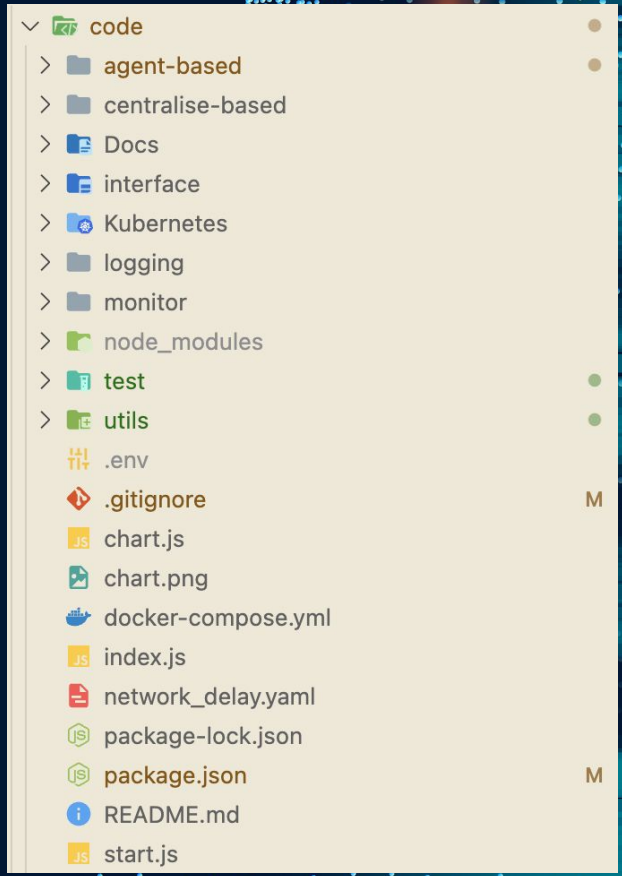
Express Server (app.js Docker Container) -> RabbitMQ Server (Docker Container)
  -> Sends/receives messages based on the HTTP requests from the client

RabbitMQ Server (Docker Container) -> HQ Service (hq.js Docker Container)
  -> Passes messages to HQ Service

RabbitMQ Server (Docker Container) -> Agent Service (agent.js Docker Container)
  -> Passes messages to Agent Service

RabbitMQ Server (Docker Container) -> Firetruck Service (firetruck.js Docker Container)
  -> Passes messages to Firetruck Service

Chaos Mesh (Docker Container)
  -> Introduces chaos by targeting the above components to test resilience



Result - Microservices and Docker: Code Overview

leverage Docker's capabilities to containerize each microservice

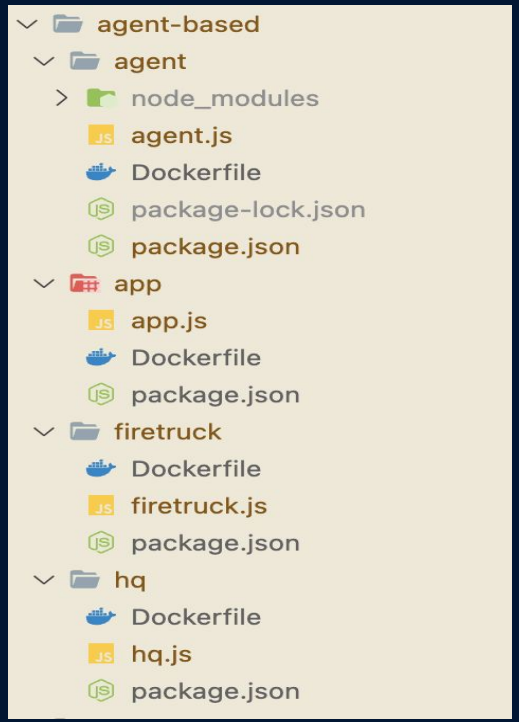
Track key performance metrics and logs. These highlight the successful inter-service communication and resource allocation

code	-	Running (8/8)		0 seconds ago
agent_based_agent-1 9951f969b09e	code-agent_based_agent	Running		30 seconds ago
agent_based_app-1 91d75b38d06e	code-agent_based_app	Running	3000:3000	31 minutes ago
agent_based_firetruck-1 b4cb8fcb851b	code-agent_based_firetruck	Running		17 seconds ago
agent_based_hq-1 59a36f7db7a9	code-agent_based_hq	Running		16 seconds ago
centralise_based_app-1 9141d448c74a	code-centralise_based_app	Running	4000:3000	2 seconds ago
centralise_based_firetruck-1 1f7266cdec8f	code-centralise_based_firetruck	Running		1 second ago
centralise_based_hq-1 8be20907f15c	code-centralise_based_hq	Running		0 seconds ago
rabbitmq-1 a201b50b8547	rabbitmq:3-management	Running	15673:15672 Show all ports (2)	32 minutes ago

```
FROM node:14

WORKDIR /agent

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 3000

CMD [ "npm", "start" ]
```

Result - agent based system

List of Components



1. Agent

- Receives messages based on agentID.

2. Hq

- Represents the headquarters agent in the system.

3. Firetruck

- Represents a fire truck agent in the system.

4. App

- Facilitates RabbitMQ connections between agents.

Result - retry mechanisms

```
2023-06-02 15:55:46
2023-06-02 15:55:46 > agent-based@1.0.0 start /app
2023-06-02 15:55:46 > node app.js
2023-06-02 15:55:46
2023-05-22 22:52:58 Failed to connect to RabbitMQ. Retrying in 5 seconds...
2023-05-29 03:50:53 Failed to connect to RabbitMQ. Retrying in 5 seconds...
2023-05-29 03:50:58 Failed to connect to RabbitMQ. Retrying in 5 seconds...
2023-05-29 03:51:03 Failed to connect to RabbitMQ. Retrying in 5 seconds...
2023-05-29 03:51:08 Failed to connect to RabbitMQ. Retrying in 5 seconds...
2023-05-29 03:51:13 Failed to connect to RabbitMQ. Retrying in 5 seconds...
2023-06-02 15:55:28 Failed to connect to RabbitMQ. Retrying in 5 seconds...
2023-06-02 15:55:46 Agent app is running on port 3000
2023-06-02 15:55:46 Connected to RabbitMQ App.js!
```

Result - failover strategies

We implemented Kubernetes and it can automatically restart failed services or spin up new instances of a service when required.

Robust error handling setup which logs uncaught exceptions and unhandled promise rejections. This would aid in recognizing when a service fails and needs to be addressed.

```
// Log uncaught exceptions and unhandled promise rejections
function setUpErrorHandlers() {
  process.on('uncaughtException', (error) => {
    console.error('Uncaught exception: ', error);
  });

  process.on('unhandledRejection', (reason, promise) => {
    console.error('Unhandled rejection at ', promise, 'reason: ', reason);
  });
}
```

Result - prioritized message routing

- Using assertQueue with a maxPriority option set to 10,

```
// Set up a queue for this agent
function setUpQueue(channel) {
  const queue = `agent_queue_${AGENT_ID}`;

  channel.assertQueue(queue, { durable: false, maxPriority: 10 }); // set maxPriority
  console.log(`Agent ${AGENT_ID} connected to RabbitMQ and queue ${queue} is set up.`);

  channel.consume(queue, handleMessage, { noAck: true });
}
```

-> This queue is capable of priority-based message delivery

Result - Chaos Mesh and Testing

- Introduce a **10s delay** to the system

- **Failure Recovery:** Quick recovery times from simulated failures, demonstrating the resilience and effectiveness of your failover strategies.

```yaml
apiVersion: chaos-mesh.org/v1alpha1
kind: NetworkChaos
metadata:
  name: network-delay-example
  namespace: default
spec:
  action: delay
  mode: one
  selector:
    namespaces:
      - default
  delay:
    latency: '10ms'
  duration: '30s'
```

Summary

1. Microservices and Agent-Based Systems:

Dived beyond basics to optimize resilience, scalability, and performance

2. Handling DDIL Scenarios

Implemented retry mechanisms, failover strategies, and prioritized message routing to maintain performance under network disruptions.

3. Simulation with Chaos Mesh

Simulated network issues to test resilience and fine-tune failover strategies. Identified system weaknesses for optimization.

Conclusion

- **Success:** We've built a system blending microservices and agent-based paradigms, showing resilience and adaptability in DDIL conditions.
- **Scalability and Resilience:** The system scales well under load and network disruptions, with swift recovery and robust security.
- **Impact:** Our research offers insights for enhancing resilience in agent-based systems for real-world applications.
- **Future Steps:** We'll refine our system further, addressing any gaps and exploring more potentials of our integrated approach.

Potential work

- **Open Source:** Our project will become open source, inviting global collaboration and innovation.
- **Advanced Techniques:** We're integrating AI and predictive models for enhanced system performance.
- **Satellite Integration:** We're considering satellite technologies to ensure resilience even in extreme network environments.
- **Enhanced Testing:** We'll further refine our testing scenarios to push our system's resilience boundaries.

Thank you !