Technical Research on Third-Party Module Adaptation and Integration for Audio-Visual Media Content Resource Service Platforms (Postprint)

Authors: Fu Kun, Deng Chengsheng, Ni Yepeng

Date: 2023-10-08T00:00:00+00:00

Abstract

Currently, the audiovisual media sector universally faces the challenge that its technical architectures cannot adapt to evolving development requirements. On the one hand, broadcast television media are unable to satisfy the rapid iteration and diversified service development demands of internet-based operations. On the other hand, internet audiovisual media enterprises struggle to meet the broadcasting industry' s requirements for high-quality content production and management during their content creation and distribution processes. Consequently, mainstream media and internet audiovisual media enterprises both domestically and internationally are actively exploring technical support architecture systems that can accommodate media transformation needs. In this context, the audiovisual media industry is gradually transitioning from traditional monolithic architectures to microservices architectures. This study examines the current challenges confronting audiovisual media content resource service platforms—including poor scalability, incomplete native platform development capabilities, and limited platform maintenance capacity—and explores novel technologies and solutions capable of interfacing audiovisual media content resource service platforms with third-party modules beyond these platforms.

Full Text

Preamble

A Technical Study on Third-Party Module Adaptation and Access for Audiovisual Media Content Resource Service Platforms

Fu Kun, Deng Chengsheng, Ni Yepeng
(School of Data Science and Intelligent Media, Communication University of

China, Beijing 100000)

Abstract: The audiovisual media domain currently faces a fundamental misalignment between technical architecture and development needs. On one hand, traditional broadcast media cannot satisfy the rapid iteration and diversified service demands of internet-based business models. On the other hand, internet audiovisual media enterprises struggle to meet broadcast-grade requirements for high-quality content production and management. Consequently, mainstream media organizations and internet audiovisual companies worldwide are actively exploring new technical architecture frameworks to support media transformation. In this context, the industry is gradually shifting from traditional monolithic architectures to microservices architectures. This study analyzes current challenges facing audiovisual media content resource service platforms, including poor extensibility, incomplete native platform development functions, and limited platform maintenance capabilities, and explores novel technologies and solutions for connecting third-party modules beyond the platform itself.

Keywords: microservices; third-party modules; adaptation access; containers; audiovisual media

1. Research Background

Driven by the "Internet Plus" wave of technical and business innovation, traditional broadcasting must embrace cloud computing and cloud-native technologies, supported by underlying microservices architectures. Compared with traditional technical architectures, microservices-based audiovisual media content resource service platforms offer significant advantages. However, the integration of large quantities of third-party microservices modules presents numerous challenges: non-standardized interfaces, security risks, low efficiency, ecosystem isolation between microservices and traditional architectures, and diverse development languages that increase system maintenance costs and hinder industry development.

To address these issues, this paper proposes a technical solution for adapting and integrating third-party modules into audiovisual media content resource service platforms, providing an innovative approach for building an audiovisual media microservices ecosystem. Through this technology, third-party modules (i.e., microservices or non-microservices modules developed by teams other than the native development team) can be adapted and connected to the native platform, deployed onto it to extend platform functionality and form a comprehensive audiovisual media microservices ecosystem.

2. Overall Solution

As computing and network technologies continue to develop and proliferate, software architecture has become increasingly critical in information system design.

Traditional monolithic architecture patterns suffer from high coupling and poor business module reusability, failing to meet modern user requirements. To solve third-party module integration challenges, this paper proposes a comprehensive solution based on container technology, as shown in Figure 1 [Figure 1: see original paper]. This solution employs a container-based microservices architecture to ensure third-party module availability and improve integration efficiency.

Through investigation of domestic and international technologies and reference to the Audiovisual Media Microservices Architecture (MMA), we propose a container-based microservices technology stack, identifying container and image repositories as the foundation for third-party module integration. Based on the seven functional domains of audiovisual media, we study business function decomposition methods and principles to provide guidance for third-party module development teams and establish standardized API interface specifications. According to these universal API interface standards, third-party development teams can design microservice interfaces, providing a unified interaction interface for platforms and complex microservices. We also research microservice image creation technology to guide teams in packaging microservice code and runtime environments for upload to platform image repositories, enabling cross-platform deployment and execution.

The overall solution for third-party module adaptation in audiovisual media microservices can be divided into the following steps:

2.1 Business Microservitization

By analyzing audiovisual media business processes, data flows, and code logic, we decompose original businesses into independent, system-appropriate microservices aligned with seven domains comprising over 340 microservices.

2.2 Microservice Interface Design

Following module decomposition, interface design is conducted according to microservice functions and characteristics within the business system context. This guides integration parties in modifying original monolithic application structures or non-compliant interfaces, implementing reasonable interface code based on established principles.

2.3 Microservice Image Packaging

After local testing, third-party development teams package module code and environments using the methods provided in this solution, creating corresponding images. Following authentication and authorization, these images are pushed to the image repository.

3. Container-Based Microservices Platform Technology Stack

Based on our understanding of the Audiovisual Media Microservices Architecture (MMA) and container technologies, we propose a container-based microservices platform technology stack for reference in building audiovisual media content resource service platforms. Current research indicates that only container-based microservices technology stacks can guarantee third-party module adaptation and integration. The proposed technology stack is illustrated in Figure 2 [Figure 2: see original paper].

The platform consists of two layers. The underlying IaaS layer can be either virtual machines or physical servers, with containers abstracting away the specific form of underlying resources. The middle CaaS layer primarily provides container-level services, including Docker, Kubernetes, and Istio, corresponding to container lifecycle management, container orchestration, and microservices governance. This technology stack meets the characteristic requirements of audiovisual media microservices architecture while providing container-related services that lay the foundation for third-party module integration.

4. Monolithic Architecture Third-Party Module Microservitization

According to the above solution, our research focuses on three main aspects: First, containers serve as the carrier for microservices. Through image packaging technology, third-party microservices modules can be packaged and uploaded to image repositories for deployment. Therefore, the first research focus is the container-based microservices platform, studying technologies suitable for third-party module integration. Second, monolithic third-party modules require microservitization, including service decomposition and interface design. Third, after microservitization, modules must be containerized and packaged into images that shield underlying technical differences, enabling cross-platform execution after upload to the image repository.

These three aspects complete the research on third-party module adaptation technology, enabling smooth integration of monolithic or microservices-based third-party modules into the platform.

4.1 Data Decomposition Method

This study adopts an optimized data flow-driven decomposition method, as shown in Figure 3 [Figure 3: see original paper]:

1. **Requirement Analysis:** Analysts examine the software system requirements, identifying domain contexts and entities based on system use cases or natural language business logic descriptions to prepare for data flow diagram construction.

2. **Simplified Data Flow Diagram Construction:** Based on the requirement analysis, data analysts manually construct detailed hierarchical data flow diagrams and simplified versions. The simplified diagrams follow predefined decomposition rules, focusing on relationships between operations and data stores while excluding extraneous information such as external entities and specific data transmission details.
3. **Partitionable Data Flow Diagram Construction:** This step extracts interaction information between operations and data stores from the simplified diagrams, automatically transforming them into partitionable data flow diagrams through algorithms. The focus on operation-data store interactions aims to avoid splitting data stores across different microservices during decomposition, thereby minimizing unnecessary data consistency issues while maintaining fine-grained microservices.
4. **Candidate Microservice Identification:** The partitionable data flow diagram is decomposed using designed algorithms that aggregate operations related to the same data store, then merge duplicate operations across aggregates. The merged results serve as candidate microservices.

4.2.1 Partitioning Based on Minimum Spanning Tree We implement a minimum spanning tree-based algorithm for business operation clustering. This algorithm relies on parameters including a weighted graph G generated from collected operation interaction data, the expected number of microservices m, and a threshold n for the number of classes per microservice. In graph G, nodes represent operation classes, and edge weights represent interaction frequencies —higher weights indicate more frequent interactions and higher coupling, suggesting these operations should be partitioned into the same microservice.

The algorithm first generates graph G' s minimum spanning tree (MST) using Kruskal' s algorithm. Since higher edge weights indicate closer relationships, weights are inverted before MST computation. The MST contains the most tightly coupled operation nodes and edges. By sorting and sequentially removing the highest-weight edges, we achieve decomposition of loosely coupled operations. Depth-First Search (DFS) traverses the remaining subgraphs. The algorithm terminates based on the expected microservice count and class threshold parameters, providing flexibility for different requirements.

4.2.2 Database Table Partitioning Based on K-means The Database per Service principle advocates that each microservice maintains its own database, with other services accessing data through exposed interfaces. During migration to microservices architecture, database tables must be partitioned. Existing methods use simple rules based on operation-data store relationships or design-phase entity analysis. However, in complex monolithic systems, database tables exhibit coupling and cohesion that simple rules cannot fully address, potentially resulting in inappropriate granularity and performance impacts from frequent cross-service interactions.

Building upon business decomposition results, we analyze interaction patterns between collected operation classes and data stores during runtime, using K-means algorithm for effective database table partitioning. K-means relies on parameters from the previous clustering phase: operation sets and the weighted graph G representing operation-database table interactions. The algorithm calculates Dijkstra shortest paths between set points and central points using inverted edge weights, assigning points to the nearest central point cluster. Central points are updated iteratively until convergence, outputting clusters representing candidate microservices containing database tables.

4.3 Third-Party Module General Interface Design Specification

Audiovisual media business is decomposed into seven domains with over 340 microservices. Combining object-oriented programming and operating system function design principles, we propose a standardized interface design scheme:

1. All interfaces follow RESTful API style and use HTTP protocol.
2. All microservices encapsulate functionality through eight standard interfaces:
    - **Create Service:** Creates a service based on user requirements, returning a unique service ID. Service configurations persist independently of service lifecycle.
    - **Open (Run) Service:** Allocates IT resources to run the service, returning runtime credentials valid for the current session. Optional callback URLs enable asynchronous notifications.
    - **Close Service (Optional):** Releases resources after use to reduce waste. Some services auto-terminate.
    - **Get Service Information:** Retrieves service name, runtime status, resource usage (CPU, memory, I/O), etc.
    - **Destroy/Delete Service:** Permanently removes services and associated configurations.
    - **Update Service Information (Optional):** Modifies service metadata such as name.
    - **Get Service List (Optional):** Retrieves all services for a user with basic information.
    - **Service Configuration (Optional):** Allows advanced users to customize service parameters when defaults are insufficient.

This scheme requires only simple secondary encapsulation of developed service interfaces, providing clear functional boundaries, low development cost, and easy implementation. The interface design abstracts relationships between services, users, and platforms, reducing management complexity and learning costs.

5. Microservices Module Image Creation Technology

Container cloud platforms support two image building methods, as shown in Figure 4 [Figure 4: see original paper]:

Automatic Building: Users select a base image and configure startup commands, environment variables, service ports, and other parameters. The platform automatically generates a Dockerfile and builds the image via Docker API, uploading it to the image repository.

Manual Building: Users create a container from a base image with SSH service and mapped ports. After logging in via SSH to modify configurations and files, users can verify changes through service ports. Upon confirmation, the container platform automatically generates and commits the image to the repository.

The platform records each user' s build history for auditing and analysis. Through the image management console, users can browse all images with multi-dimensional sorting (by name, user, build time, etc.) and edit image details including logo, description, ports, startup commands, environment variables, and storage paths. Usage statistics support analysis. For deployed images, rollback to previous versions is supported—a common operation during application deployment. After successful image creation, running the image in Docker achieves service deployment, with Kubernetes managing service operations.

6. Summary and Outlook

This research on third-party module adaptation for audiovisual media microservices aims to extend platform functionality, improve content production quality, and accelerate ecosystem development. The proposed solution enables smooth integration of third-party modules, rapidly forming an audiovisual media production ecosystem and promoting industry development.

Future work will select different third-party modules for experimental validation. Based on testing results, we will continuously optimize the adaptation technology to achieve a complete solution and establish technical specifications for third-party microservices module integration testing, laying the foundation for a robust audiovisual media microservices ecosystem.

References

[1] Yang Ou, Zhang Yi, Geng Zhenwei. Application Practice of Microservice Architecture in Container Cloud [J]. Computer & Telecommunication, 2017(7): 79-81.
[2] Jamshidi P, Pahl C, Mendonça NC, et al. Microservices: The journey so far and challenges ahead. IEEE Software, 2018(3): 24—35.
[3] Huang Xianchen. System Design and Implementation Based on Microservice Architecture [J]. Information Technology and Informatization, 2020(11): 16-17.
[4] Richardson C. Pattern: Microservice architecture. 2018. http://microservices.io/patterns/microservices.htm
[5] Escobar D, Cárdenas D, Amarillo R, Castro E, Garcés K, Parra C, Casallas R. Towards the understanding and evolution of monolithic applications as

microservices. In: Proc. of the 2016 XLII Latin American Computing Conf. (CLEI). 2016. 1−11.
[6] Yao Gang, Wu Haili, Wang Yibin. Analysis of the Role of API Gateway in Microservice Architecture [J]. Information System Engineering, 2020(12): 16-18.
[7] Ahmadvand M, Ibrahim A. Requirements reconciliation for scalable and secure microservice (de)composition. In: Proc. of the 2016 IEEE 24th Int' l Requirements Engineering Conf. Workshops (REW). IEEE, 2016: 68−73.
[8] Levcovitz A, Terra R, Valente MT. Towards a technique for extracting microservices from monolithic enterprise systems. arXiv preprint arXiv:1605.03175, 2016.
[9] Newman S, Wrote; Cui LQ, Zhang J, Translate. Building Microservices: Designing Fine-grained Systems. 2nd ed., Beijing: People' s Posts and Telecommunications Press, 2016 (in Chinese).
[10] Baresi L, Garriga M, Derenzis A. Microservices identification through interface analysis. In: Proc. of the European Conf. on Service-oriented and Cloud Computing. 2017: 19−33.
[11] Gong Fangsheng. Application of Docker Technology in Microservices [J]. Electronic Technology & Software Engineering, 2021(4): 54-56.
[12] Xue Liang, Hou Jie. Research on Microservitization Architecture of Combat Applications Based on Docker [J]. Ship Electronic Engineering, 2020(3): 22-25.
[13] Hassan S, Bahsoon R. Microservices and their design trade-offs: A self-adaptive roadmap. In: Proc. of the 2016 IEEE Int' l Conf. on Services Computing (SCC). IEEE, 2016: 813−818.
[14] Evans E, Wrote; Zhao L, Sheng HY, Liu X, Translate. Domain-driven Design: Tracking Complexity in the Heart of Software. Beijing: People' s Posts and Telecommunications Press, 2016 (in Chinese).
[15] Kecskemeti G, Marosi AC, Kertesz A. The ENTICE approach to decompose monolithic services into microservices. In: Proc. of the 2016 Int' l Conf. on High Performance Computing Simulation (HPCS). IEEE, 2016: 591−596.
[16] Hassan S, Ali N, Bahsoon R. Microservice ambients: An architectural meta-modelling approach for microservice granularity. In: Proc. of the 2017 IEEE Int' l Conf. on Software Architecture (ICSA). IEEE, 2017: 1−10.
[17] Gysel M, KoLbener L, Giersche W, et al. Service cutter: A systematic approach to service decomposition. In: Proc. of the European Conf. on Service-oriented and Cloud Computing. Springer-Verlag, 2016: 185−200.
[18] Mazlami G, Cito J, Leitner P. Extraction of microservices from monolithic software architectures. In: Proc. of the 2017 IEEE Int' l Conf. on Web Services (ICWS). 2017: 524−531.

Note: Figure translations are in progress. See original paper for figures.