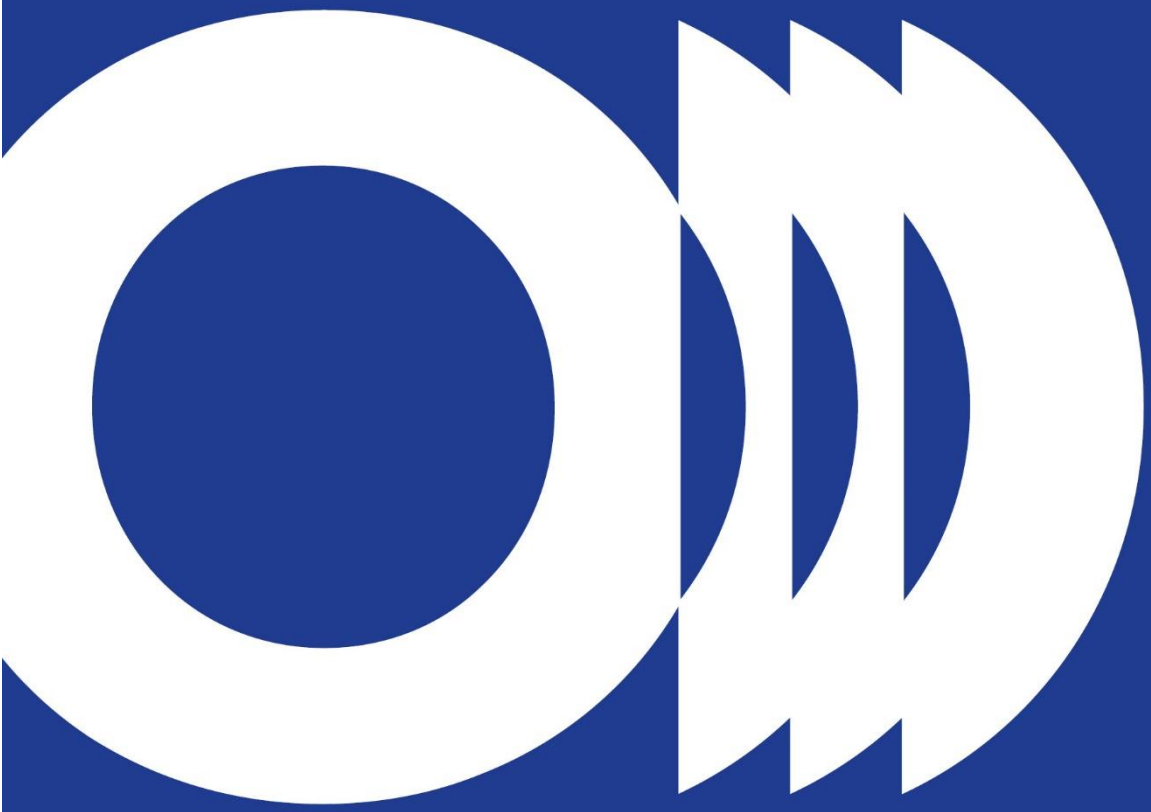
EBU

OPERATING EUROVISION AND EURORADIO

MEDIA CLOUD MICRO-SERVICE ARCHITECTURE

WHITE PAPER



MCMA

A CLOUD AGNOSTIC MICRO-SERVICE ARCHITECTURE FOR MEDIA

The market for serverless computing is growing fast, and the adoption rate is continuously increasing. The global serverless market is projected to grow to US$ 21 billion by 2025. Serverless cloud computing is, therefore, far from being a niche market. The latest evolution in the move from server-based to serverless architecture is 'function as a service', FaaS (Figure 1). The level of abstraction for FaaS is one step above that of 'infrastructure as a service', IaaS. Functions are directly exposed to the users to simplify the development of the related applications.

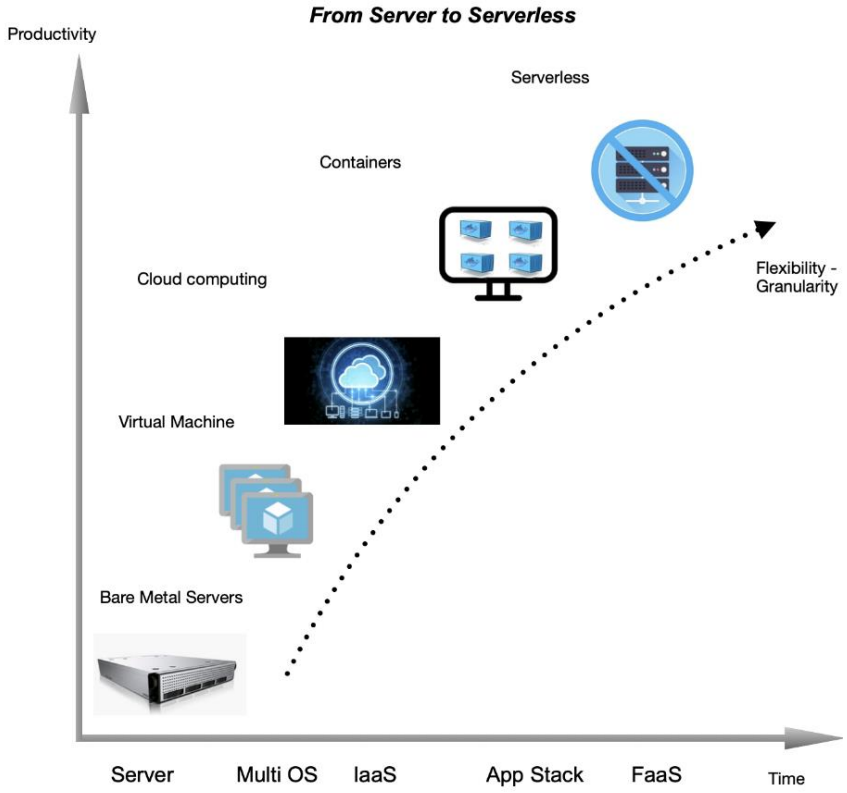


Figure 1. *The evolution towards serverless architecture brings flexibility and productivity*

Thus, the scaling of the underlying containers to absorb the load is managed by the cloud providers. The main benefit for the developers is that they can remain focused on the development of the functions defining the application.

The main business driver of FaaS architecture is the pay-as-you-go principle: the architecture allows for charging the user linearly on the running time. So, the price can range from 0 to infinity. More precisely, the cost scales linearly with memory, duration and number of requests. However, for this to happen the unused resources have to be shut down. This gives rise to the 'cold start' effect: if a resource is not requested, the container is stopped, and it takes some time to spin it up. A strong constraint, which is a consequence of the resource management strategy defined by cloud providers, is the automatic timeout: a function cannot run more than 15 minutes. Consequently, this pushes developers towards defining microservices to take account of the timing constraints – more is better.

1- MCMA OVERVIEW

Most broadcasters today are using cloud services in some ways, but the cloud continues to change fast, and attractive new possibilities have opened up. The latest trend is towards serverless architecture and what is known as FaaS – function as a service – opening the door to an entirely new way of running media services.

An EBU project MCMA, provides a painless way of adopting serverless architecture and implementing media workflows in the cloud. Created as an open-source project, free to use and available on GitHub, it offers reusable services that can apply to all applications and infrastructures across multiple cloud providers.

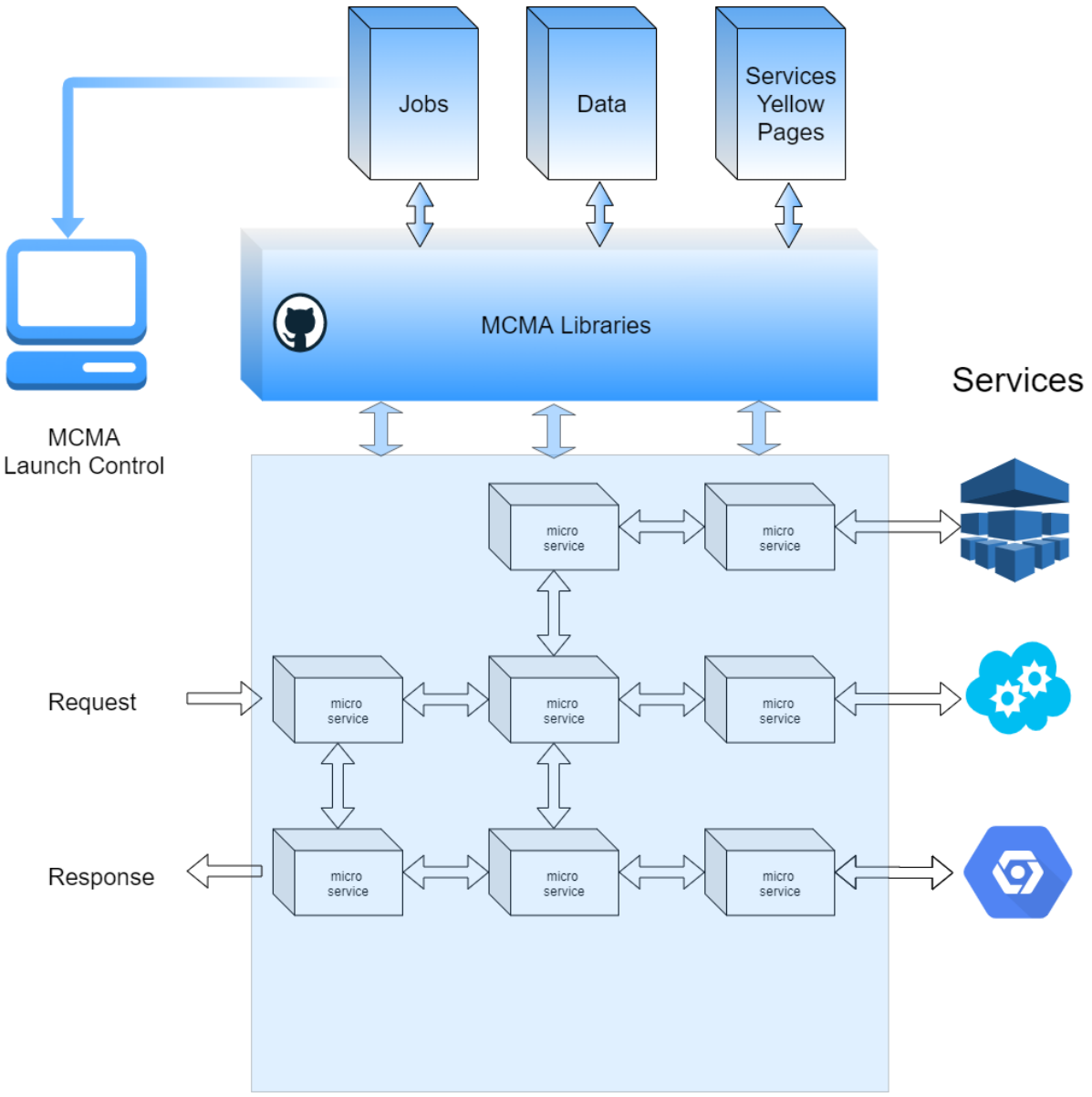


Figure 2: *MCMA architecture overview*

1.1 - EFFICIENT DEVELOPMENT STRATEGY

A key benefit of *FaaS* and MCMA is that developers can remain focused on the development of the functions defining the application. The work happens one level of abstraction up from the previous approach known as IaaS – infrastructure as a service – so the scaling of the underlying containers to absorb the workload is managed by the cloud providers. With the functions directly exposed to the users, the development of the related applications or services is simplified and the time to market is reduced. The MCMA project is not a concept, it started by solving practical problems every media software engineer will face when moving to the cloud:

- How to use native media cloud services (AWS Media Convert, Azure Video indexer…)
- How to store physical media and keep track of asset metadata in the cloud?
- How to use open source services / custom in the cloud?
- How to connect services together in the cloud?
- How to monitor and track execution of services in the cloud?
- How to model a complex media workflow?
- How to create a secure infrastructure?
- How to leverage the performance capabilities and agility of the cloud?
- How to manage the cost of my infrastructure?
- Which cloud providers to target?
- How to deploy and update my infrastructure?
- How to share my infrastructure with my colleagues (developers, QA)
- How do I troubleshoot problems?

Starting from these questions, MCMA is an efficient architecture built on concepts that can be summed-up as follow:

- Standardize the notion of service from an operation and management point of view, not functionality
- Define a common structure for representing an atomic operation (Job)
- Create a processing layer having the ability to match instantiation of operations with underline configured infrastructure
- Implement a reusable set of foundational libraries and services (Data repositories, logging, security, resource adapters)
- Define the pipeline for deployment and infrastructure management
- Enable consistency of service representation and management across cloud providers
- Embrace the cloud native service and make it practical and approachable

MCMA optimizes the use of native cloud services and still provide interoperability across providers. CICD (continuous integration and continuous delivery) pipelines are an essential part of MCMA. Enhancements have been made over time to streamline accessibility and simplify the learning curve for deploying and managing cloud infrastructure.

From the developer point of view, MCMA comes as a set of libraries to sequence services and to track and manage jobs. The libraries and foundational services take into consideration security and management best practices for cloud infrastructure.

MCMA libraries and workflows for media applications are available on GitHub:

- For libraries see https://github.com/ebu/mcma-libraries;
- And https://github.com/ebu/mcma-projects for workflows.

The published versions of these libraries can be found:

- For NPM on https://www.npmjs.com/search?q=%40mcma ;and
- On https://www.nuget.org/packages?q=Mcma for NuGet.

2- MCMA LOGGING STRATEGY

The log management strategy is a critical challenge in a microservice architecture. Furthermore, the operations on media files are usually long-running; this increases the overall complexity of the system management. Hence, MCMA offers efficient architecture and formalism to meet the requirements related to long-running transactions and efficient job management while remaining independent of cloud service providers.

Before being assigned to an adequate service the jobs are stored in a central Job Processor. The Job entity, therefore, contains all the necessary information for assigning a service executing the intended operation.

A Job entity is defined by:

- Descriptive fields: job type, id and profile.
- Status and error field providing detailed information about the 'Failed' state conform to the RFC 7807 standard.
- Inputs and Outputs data

Correlate the logs from multiple microservices that belong to the same transaction can be very complex. In order to achieve this, a Tracker, which is a customisable token, is passed from one microservice to the next. The receiving service will then process this tracker and use it when creating log entries.

In an MCMA environment the job processor is a service which stores all the jobs that need to be executed and create a log entry whenever a job is starting, running or completing/failing.

3- COMMON SERVICE PATTERN

MCMA is an event-driven architecture built on a simple and scalable brick, the Common Service Pattern. The following schema sums up this service pattern on AWS.

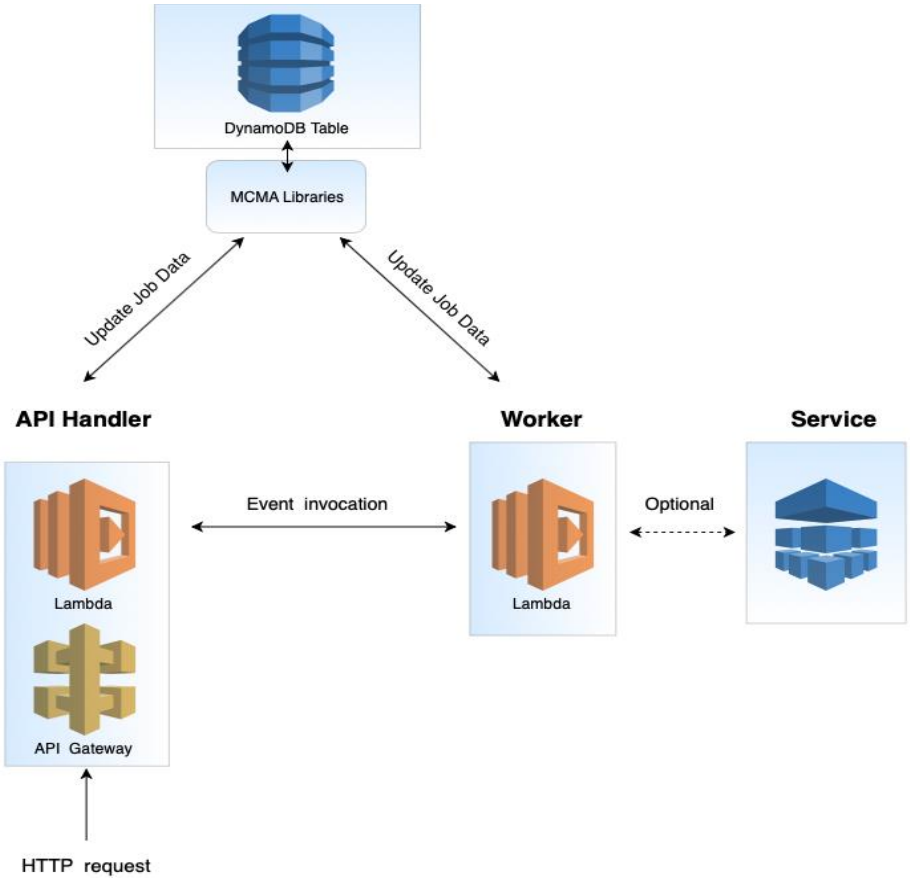


Figure 3: *MCMA common service pattern*

The first block is the API Handler, and it receives an Incoming HTTP Requests from the API Gateway. The API handler validates and routes and acknowledges the HTTP request. Then API Handler and Worker retrieve and update the database related to the Job management. In the end, the worker Process the Request. Built on this common service pattern, MCMA provides three main types of services related to:

- Data storage and operations;
- Job execution and management; and
- Service definition and abstraction.

For the content storage, the service repository behaves as a simple RESTful service supporting CRUD operations to store and retrieve data. In practice, the commonly

implemented storage services are the Service Registry also called the Services Yellow Pages and the Media Repository storing among others metadata on media assets.
The goal of job management services is to assist with storing and managing executions of jobs. Finally, the Job executing services is the service which performs the actual operations. It waits until a job is assigned and then runs it. It can be, for instance, Automatic metadata extraction service, Transform service, Workflow service other AI services from multiple cloud providers. This architecture allows to manage and process complex media workflows as a Job.