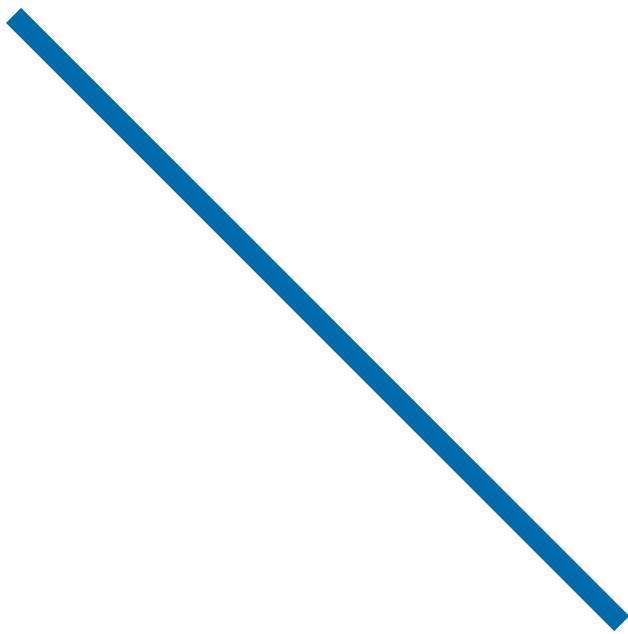


Microservices For Broadcasters



Essential Guide

EG

ESSENTIAL GUIDES

Introduction

Computer systems are driving forward broadcast innovation and the introduction of microservices is having a major impact on the way we think about software. This not only delivers improved productivity through more efficient workflow solutions for broadcasters, but also helps vendors to work more effectively to further improve the broadcaster experience.

One of the great advantages of moving to COTS systems and IT infrastructures is that we can benefit from developments in seemingly unrelated industries. Microservices have gained an impressive following in enterprise application development and many of the design methodologies transfer directly to broadcast infrastructures.

Scaling broadcast facilities has long been the goal of many system designers and television, by its very nature has times of peak demand when viewing audiences gather to watch high value programs such as Saturday night entertainment or prominent sports events. Traditionally, broadcasters would need to design their systems for the highest peak-demand events, often this was difficult to accomplish due to the massive number of unknown variables in the system leading to significantly increased costs and complexities.

Microservices are distributed software modules and combined with virtual machine infrastructures can easily scale to deliver on-demand services to facilitate peak requirements. Furthermore, due to the functional nature of microservices, new processing systems can be developed independently of the rest of the software. This promotes specialist agile teams to safely develop specific functionality such as adding Rec.2020 color space to an existing video processing component.

Agile methodologies have been making significant inroads into all areas of software development and microservices benefit greatly from the agile philosophy. They encourage and deliver software based on relevant functionality as opposed to often outdated preconceived ideas from years earlier. Agile no longer uses the waterfall method of project management, further empowering software teams to change quickly to meet the varying and increasing requirements of broadcasters. New components can be quickly and safely developed and deployed as the modular nature encourages deep and efficient software testing.

Although many broadcast infrastructures have vendor commonality in their choice of studio, edit and playout design choice, they vary greatly in their workflow implementations leading to significant variations in broadcaster requirements. With traditional hardware and software solutions, broadcasters would often have to compromise their workflow requirements as bespoke code would be needed to facilitate them resulting in complex and difficult to manage systems. Microservices go a long way to rectify this as the agile and distributed nature of their design means software interfaces can be written with greater ease and significantly reduced risk.

Simplification is key to flexibility and scalability, and microservices definitely deliver this. Instead of having a huge monolithic code base with code variations to meet the specific needs of clients, microservices promote a generalized core code base that can be easily tested and verified. RESTful API's with loosely coupled interfaces further improve the broadcaster experience as modifications and bespoke additions can be relatively easily facilitated.



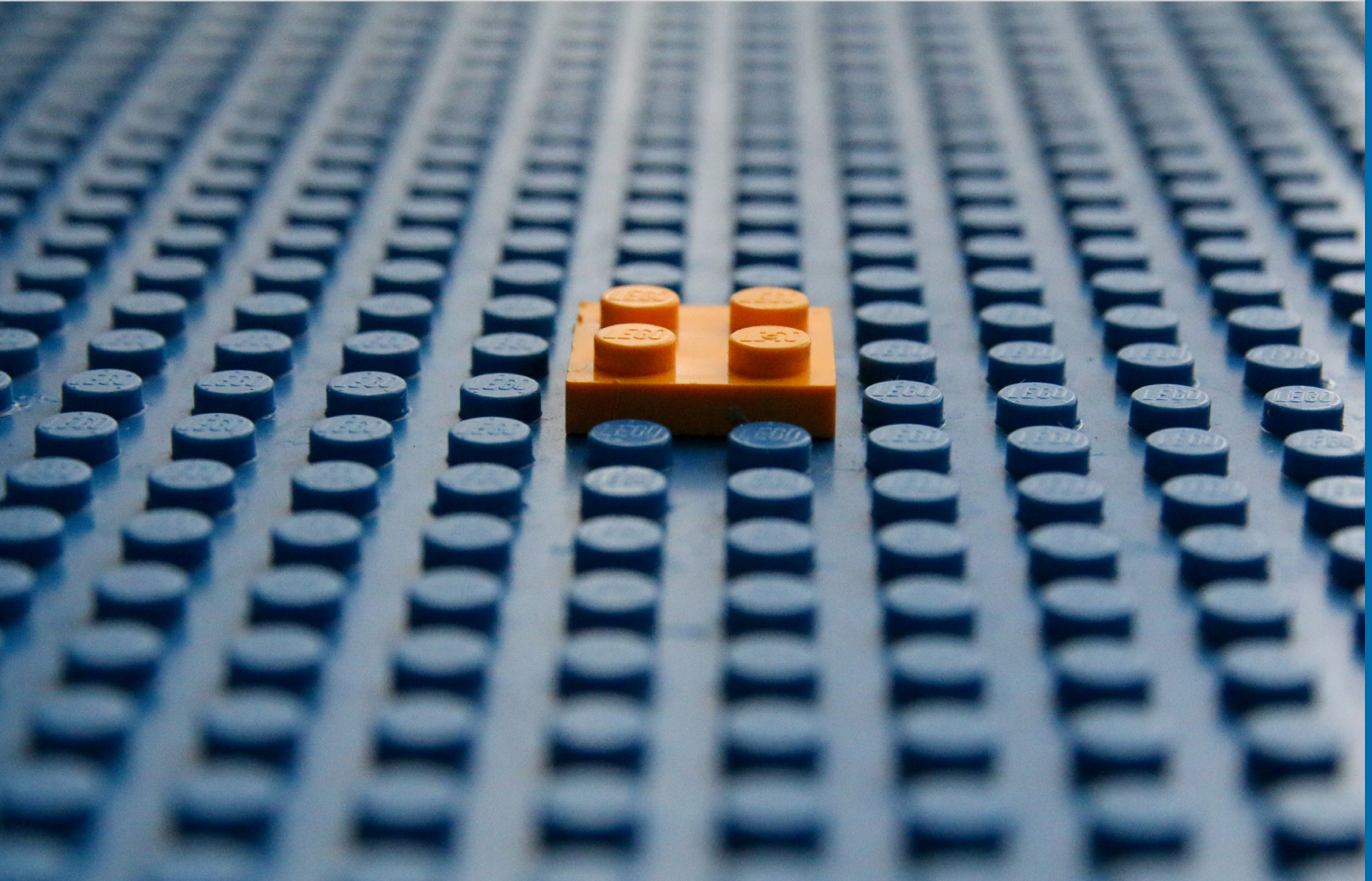
Tony Orme.

To fully appreciate the advantages of microservices, it helps to understand how software teams have worked in the past, how software was built, and the associated risk of monolithic design. This Essential Guide explains the challenges faced by software teams using waterfall project management when delivering monolithic code and then goes on to discuss and describe how agile development and microservices deliver unprecedented flexibility and scalability for broadcasters.

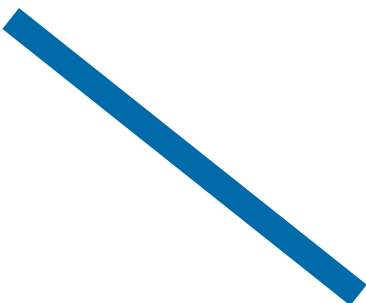
Microservices deliver a huge benefit to broadcasters and are the future of software provision for any broadcaster. This Essential Guide will help you understand why.

Tony Orme
Editor, The Broadcast Bridge

Microservices For Broadcasters



By Tony Orme, Editor at The Broadcast Bridge



Software continues to dominate broadcast infrastructures. Control, signal distribution, and monitoring are driving software adoption, and one of the major advantages of moving to computer systems is that we can ride on the crest of the wave of IT innovation. In this Essential Guide, we investigate Microservices to understand them and gain a greater appreciation for their applications in broadcasting.

Understanding the benefits of microservices to end users and broadcasters requires some background knowledge of earlier software development, the philosophy of design, and how developers actually tackle solutions.

Traditional software architectures were monolithic in design. That is, there was one big homogenous version of the executable code that provided the full end to end user experience. It would accept inputs through the user interface, access data through some sort of database, accept information through input/output interfaces, process the data, and provide the user response.

Monolithic Code

A complete program was divided into many files to provide the source code. Using a compiler, each source file was processed in turn to provide a single executable file that would be executed by the host operating system for the computer.

To make development easier, monolithic code can be modularized through the use of libraries. One example of this is code written in C or C++ using static libraries. Code is divided into multiple functional units that can be compiled into object code. This is a sort of intermediate assembly code that is hardware and operating system dependent but contains labels instead of addresses for memory locations. Each copy of the object code is then joined by the linker to resolve the label memory addresses resulting in a single executable program.

A development of this system used dynamic libraries and two types are available; dynamically linked at run-time, and dynamically loaded and unloaded during execution. For dynamically linked programs, the libraries had to be available during the compile and linking phases, but the libraries are not included in the executable code distribution. Dynamically loaded and unloaded programs use a loader system function to access the libraries at execution time.

Modularity has always been a key requirement for developers as it promotes code reuse to improve efficiencies and reduce the possibility of bugs creeping in. If you already have a library that provides a function to provide low-level access to the ethernet port for example, then why bother re-writing it?

Although the library approach makes monolithic code modular, it still suffers from some severe restrictions.

Flexibility Demands

Providing reliable, efficient and flexible code is the goal for any vendor or software developer. The term “flexible” is key to understanding the limitations of traditional monolithic code development.

Developers in software teams building monolithic code cannot work in isolation. Although it may be possible to break the code into functional units to allow parallel development cycles, the functions must be tightly coupled. That is, the interface design to the function must be well defined before coding can start. As functional requirements change, the interfaces must change across the whole design. This can have consequences for the rest of the team and changing interface or data specifications in a monolithic design results in the ripple effect.

Developers may be working on many different parts of the code base at the same time. Systems such as unit testing do exist to allow a developer to independently test the function they are working on. But every so often, the whole team must stop, and a complete re-compile of the software is executed so that the entire program can be tested again.

Increasing Monolithic Complexity

Unit testing is the process of applying known test data to a function, or group of functions and confirming the test complies with a known result. This is all well and good, but the complexity of testing increases exponentially as the type of data being tested also increases. Consequently, it’s almost impossible to test every unit in isolation and expect the whole system to work. At some point, the whole code base must be re-compiled and tested.

One of the major challenges of compiling the monolithic code is establishing all the software interfaces still work and are correct after any changes. For example, if we modify a function called `video_Proc()`, in a previous version of the code, there may be three parameters passed to a function, but in the new version of the code there may be four. As monolithic code uses functions that are tightly coupled, every function using `video_Proc()` will need to have its interface updated.

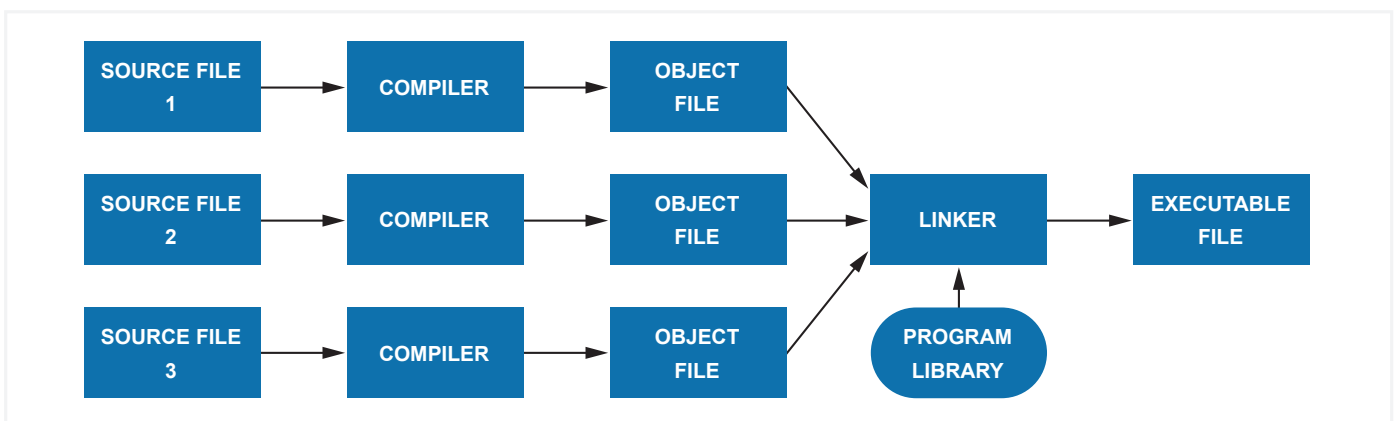


Diagram 1 – For monolithic code, multiple files are compiled into object files and then linked with external libraries to provide a single executable file. Each developer may work on one or more source files simultaneously and as monolithic code is tightly coupled, they must make sure their functional interface designs and data formats are exactly the same. This can be the source of bugs, and compiler and linker issues due to the ripple effect.

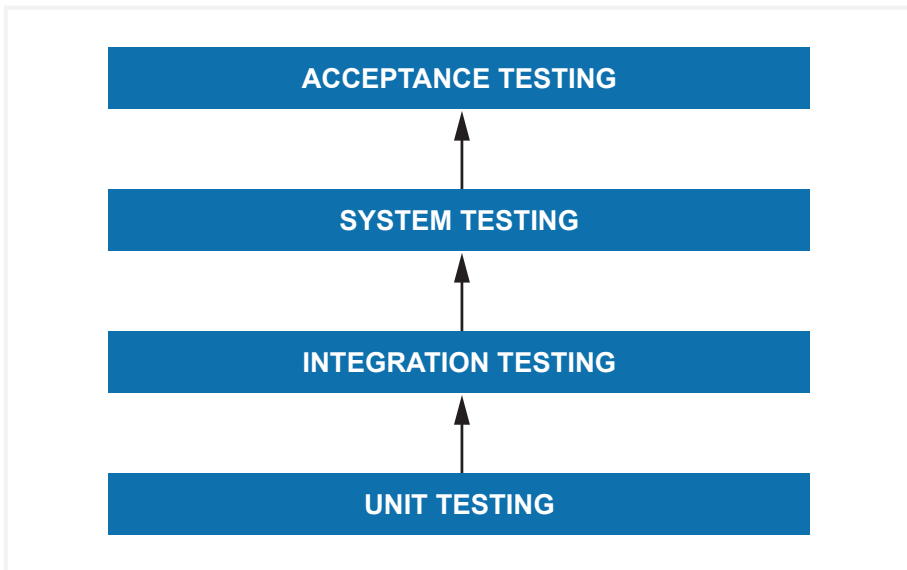


Diagram 2 – Unit testing is part of a full testing strategy and allows individual components to be tested so that they can be validated to confirm they work in accordance with the design. Integration testing combines related components and functions to test for defects in the system (ripple effects will be seen here). System testing checks for compliance against the specified requirements of the software as a whole. Acceptance testing confirms the software works as the client expects.

It may take some time to work through the whole code base to find all references to this function, change the number of parameters passed to it and recompile. Even with modest sized software teams, the code base can soon expand into tens and hundreds of thousands of lines of code. Solutions such as polymorphism exist to overcome this, but such object-oriented design philosophies soon become complex and difficult to manage and have their own challenges.

This complexity is undesirable and leads to slow release times and difficult to manage code, furthermore, it's very difficult to meet the unique and specific demands of individual clients.

In the ideal world, a vendor would be able to provide a single version of code for every one of their clients. With small applications such as phone apps this is possible. However, no two broadcast facilities are the same and workflows generally differ, even if the same infrastructure components and vendors are used. Localization in the form of best working practices and transmission formats all conspire to create individually complex broadcast systems. Consequently, vendors must provide flexibility in their code design to facilitate client requirements.

Hardware Development is Slow

We rely on specifications such as SMPTE's ST-2110 to provide common signal distribution for video and audio over IP distribution. These standards are often years in the planning and are generally static once released. They do get updated occasionally but they are usually always backwards compatible. New releases are relatively infrequent as they often result in hardware changes that can take many months to implement. However, users and clients have become used to much shorter development cycles for software-based products and are usually not willing to wait years for a solution.

Another consequence of the much shorter design cycle is that vendors tend to design their own data exchange and control interfaces and simply do not have the time to engage in committee meetings to agree the next MAM interface standard. And even if they did, the rapid development of current software technology means standards such of these would most probably be out of date even before they were published. Therefore, the software must be flexible to be able to interface to any other system.

This is possible in monolithic designs and a great deal of flexibility has been provided in recent years. However, the challenges for the development teams increase exponentially. To facilitate different control interfaces, unique to specific clients, the software teams must continually support the modules associated with that client for evermore.

Scalability Requirements

Another challenge monolithic code presents, is that of scalability. One of the key advantages cloud computing provides, whether public or private, is the ability to scale resource as and when we need it. As more user demand is placed on the code, the underlying resource supporting it must also be increased.

Monolithic code, can, to a certain extent, scale to meet increased user demands. This is achieved by increasing the number of instances of the code running behind a device such as a load balancer. The load balancer can detect the number of user requests and when they pass a certain threshold, spin up new instances of the code. This is how traditional web servers worked using solutions such as Apache. However, monolithic code cannot scale to meet the demands of increasing data volume as each instance of the program will need access to all of the data. This potentially makes memory management and caching inefficient and can lead to contended I/O access.

Also, different functions within the program may have different resource requirements. For example, a video compression function may be CPU intensive, whereas a video processing function may be GPU intensive. Monolithic code does not allow us to easily split the code into functional components to maintain scalability to this granularity, and hence efficiency.

Microservices to the Rescue

The solution to many of these challenges is the use of Microservices.

Microservices is a generic name given to a software development method that arranges a program as a collection of loosely coupled functions (or services).

Loosely coupled is the opposite of tightly coupled. In the monolithic code description earlier the effects of tightly coupled interfaces led to the ripple down effect on the rest of the program. However, with loosely coupled systems, each component (or function) has little knowledge of the definitions of other components. Consequently, components can be replaced with other versions of the component that provide the same function with greater ease and reliability.

Unlike monolithic code, microservices, through loose coupling can exist on multiple platforms with different code base and interface methods. For example, the video processing function may exist on virtualized instances with GPU resource. The user interface code might call this as part of a workflow when the user uploads a video file using a messaging system such as RESTful (Representational State Transfer) API's.

Message Queuing

RESTful is a hardware and operating system agnostic method of exchanging messages between software components. Webservices notably use this system through HTTP methods. Messages such as GET, HEAD, and POST (to name but a few) are sent from a web browser to server to send and receive web pages and information. Microservices use a similar method allowing them to take full advantage of distributed system programming.

This leads onto the concept of message queuing. Each component requiring a video processor will send a message to the video processing component. A load balancer, or similar middleware message processor can determine the number of messages in a queue for the video processor, and if there is too much of a backlog then it can spin up new instances of the service on a new virtualized server.

When the backlog of message queues has been serviced, the instances will be switched offline and then deleted. Again, greatly improving efficiency for the broadcaster.

As well as providing a system that can be massively and relatively easily scaled, microservice designs lead to greater programming efficiency for software teams resulting in much improved reliability and cost savings.

User Expectations

As new user requirements are continually expanding the size of the code base, monolithic designs soon become incredibly big and difficult to follow. Due to the tight coupling and associated ripple effect of this architecture, developers must have a picture of the entire code architecture in their heads when they start programming.

For new members of the team this can be incredibly intimidating, and for existing members it means that they must constantly increase their understanding of the whole design, even the areas of code their colleagues are working on that doesn't necessarily affect them. This leads to inefficient allocation of highly qualified developers, and a great deal of stress and risk every time the code is recompiled.

As microservices are loosely coupled, many of these issues are resolved. Developers can work in small teams as each feature is considered a component in its own right and can be developed independently of the rest of the team. For example, if the video processor needs to be improved to work with Rec.2020 wide color gamut, then the team responsible for this service can work on and deploy the service as required. It will maintain its backwards compatibility with Rec.709 color space so the services calling this function will not require changing or notifying, so there is no re-compilation and consequently no ripple down effect.

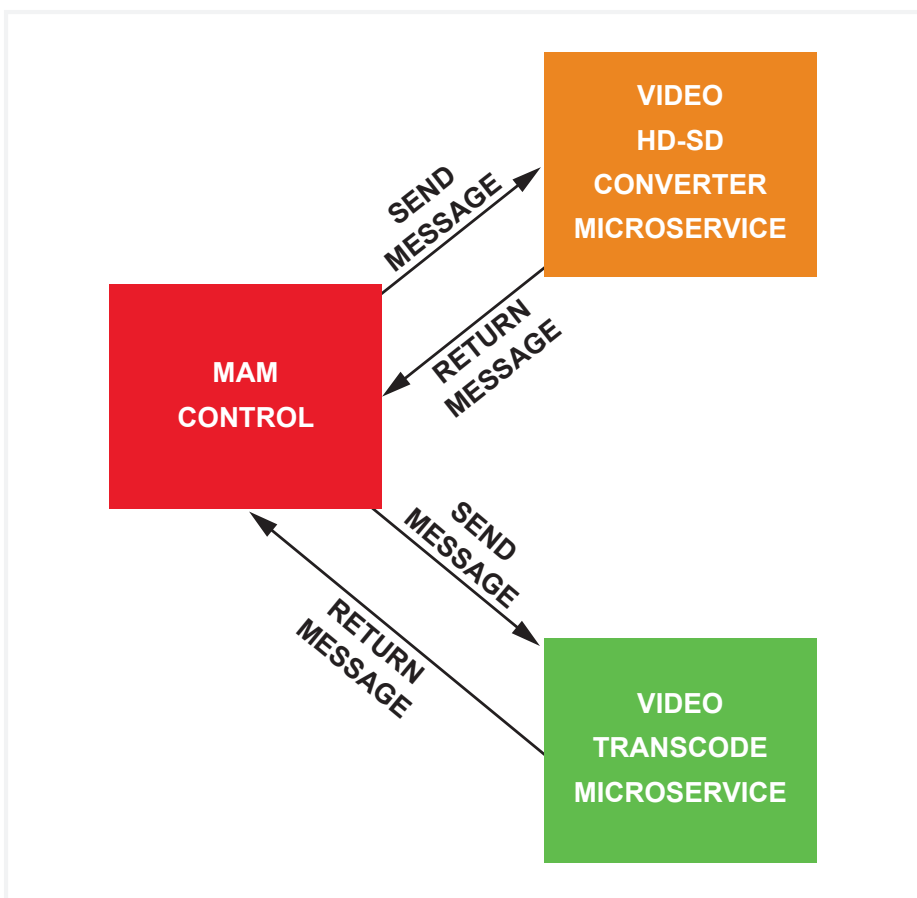


Diagram 3 – The MAM system treats the microservice provider as a blackbox and has no knowledge of the underlying hardware and operating system architecture of the microservice provider. As the work load increases the microservice provider may decide to spin up more resource through virtualization and then switch it off after the peak demand has reduced.

Furthermore, if the developers of the improved Rec.2020 video processor decide it needs an updated GPU then this can be achieved without notifying the rest of the team. They may well mention it in their weekly update meetings but for the rest of the team it will not matter as their messaging communication is unchanged. In effect, this is treated as a black box by the rest of the software team.

Broadcasters benefit greatly from this design philosophy as it promotes flexibility and allows much easier customization. If a broadcaster needs to log events sent to a MAM service in a particular format as part of their logging and compliance requirement, then the specific logging service can be adapted without reference to the rest of the code. This allows the vendor to better understand the problem to be solved and cost the work accordingly. They also know the risk of side-effects and consequently the risk to the rest of the design will be greatly reduced, resulting in a much improved and reliable service.

Reliability, Scalability and Flexibility for Broadcasters

Being able to measure the number of services and the frequency they are used allows system integrators to be able to calculate with high levels of accuracy, the size of the hardware resource required. Furthermore, this promotes virtualization to take advantage of using pay as you go hardware resource, further improving efficiency.

Advanced logging and monitoring can be easily provided at the microservices level. The metrics help dev-ops and system managers to understand which parts of the system are working hardest, or not at all. All this leads to much greater efficiency, improved coding and consequently greatly improved reliability.

Microservices create the reliability, scalability, and flexibility broadcasters have been demanding for many years. The loosely coupled architecture further allows vendors to quickly and reliably build new services specifically for the broadcasters and use-case. And combined with virtualization and service monitoring, highly adaptable systems can be easily designed to further enhance efficiency and reliability for broadcasters.

The Sponsors Perspective

Go Small To Go Big: Keeping Broadcasters Ahead Of The Curve With Microservices

By Neil Maycock, Senior Vice President – Strategic Marketing & Payout, Grass Valley

The media industry is evolving faster than at any point in its history. Broadcasters and content producers are striving to meet consumers' insatiable appetite for more content, rich viewing experiences, stunning images and access across all screens.



As a result, in some cases, we have a situation where broadcasters' revenues are growing more slowly than their costs. In fact, the big question facing all broadcasters today is how to create more first-class content more efficiently.

Timescales for bringing new services to market are also compressing; from conception to launch is now a few months, if not weeks, compared to the years it would have taken in the past. Services today must also evolve very rapidly to keep pace with their core audience demands and need to be able to spin-down as fast as they are spun up.

Supported by

Entrenched models can no longer be relied on to take on the challenges of this new mediascape. On an operational level, a vast amount of content is now being produced, ingested and managed. Traditional broadcast architectures can't scale at the necessary volume or adapt as quickly as they now need to, while the single-function system – although perfect for its intended use – is unable to support broadcasters' need for hyper-agility.

Virtualization And Beyond

As an industry, we need technology and commercial models that can support highly nimble operations. For their part, vendors need to give customers platforms that can be deployed at speed and rapidly evolved. Looking ahead, broadcasters are going to need to be more agile than they've ever been, delivering services that can stay ahead of the shifting needs of the consumers.

The move from CAPEX to OPEX and workflows virtualized on commodity hardware are steps in the right direction, giving broadcasters more flexibility to add and pay for additional capability and capacity as needed.

As broadcaster and media companies adapt to take on the challenges of the new, dynamic market head-on, a cloud-native or microservices approach enables them to take the next leap in evolution. Microservices take the virtualized software-based approach one step further, separating processes into smaller more autonomous functions, and allowing multiple microservices to be combined to deliver specific applications.

Microservices add greater degrees of inherent flexibility to existing IT infrastructure. Once you get the IT right – whether you're running monolithic applications on it or microservices-based ones, the hardware remains the same. Not only does this model deliver the inherent nimbleness and flexibility needed to shape successful media businesses for the future, it also opens up new ways to build, maintain and operate services and provides the capacity to scale these services – up or down – in a very compressed time frame.

Getting It Right And Going Live

While new technologies always create a buzz, the challenge for the broadcast industry is not to get side-tracked into replicating old workflows or business models; shoehorning an existing approach into a new paradigm means you miss all the benefits that new technology brings. We've seen this with IP – in the early days, as an industry we tried to map the SDI world to IP, losing a lot of benefits that IP offers.

We need to avoid doing the same thing with cloud and microservices. Instead of focusing on migrating applications like playout on to cloud platforms and running them on a microservices architecture, we need to look at where these technologies can make the biggest impact. While playout is technically the easiest thing to implement in a cloud environment, it's typically a 24/7 service, running with very high utilization of the underlying infrastructure. This doesn't make the best use of the nimbleness that microservices deliver.

Live production – and particularly live sports – is where the power of microservices has the potential to really come into its own. Not only does this model lend itself really well to the needs of live environments but this type of content is something consumers place a higher value on – especially live sports. Furthermore, they are willing to pay for it – PWC estimates that over 90 percent of sports fans subscribe to services for access to live games.

Being able to rapidly spin up temporary, subscription services like a pay-per-view event or targeted, seasonal sports packages – all Liverpool soccer games for instance – fast and cost effectively is hugely valuable to broadcasters. Leveraging microservices, capabilities can be fired up just before a game then be turned off after the final post-game analysis wraps up. In essence, you are only paying while the infrastructure is in use.

As the technology matures, we can expect microservices to be part and parcel of delivering transitory, live services, where very bursty capacity is needed. On a technical level, live is undoubtedly more challenging to do due to the time constraints, but it's certainly solvable and we'll see solutions hitting the market as soon as this year.

Supported by

Vendors Must Answer The Call

As they meet these changing customer demands, the onus is on the vendor community to develop applications that use microservices, rather than taking our existing products and reshaping them for the changing market. While broadcasters have a real need for solutions that underpin new business and operational models, our customers tell us they want technology adoption to be largely hidden from them. The applications and operation should be familiar and work as they need to.

Our customers face real time pressure and need to rapidly change business models; we, as technology providers, need to create technology to facilitate that for them. Just as broadcasters will have to rapidly conceptualize and deliver new services, vendors have to increase innovation velocity, delivering lots of fast iterations of microservices architectures, and adapting their capabilities, so that customers, in turn, can keep running successful agile businesses.

The internet has really shaken up the broadcast industry, shaping the way consumers access content and throwing down the gauntlet to traditional broadcast models. While this has presented a challenge for the vendor community, at the same time it's provided the solution. In the case of Grass Valley, we can now put our unique intellectual property and expertise in media and live production on internet platforms. Furthermore, we can now leverage technologies that allow us to exploit our intellectual properties in a way that just wasn't possible before.

We're at an exciting crossroads, where long-held beliefs about what a broadcast facility looks like, or how content is created and delivered, are being shed. Our customers need partners that can understand the shifting dynamic and deliver solutions – regardless of technology – that answers their needs and can help them adapt and scale at speed.



Neil Maycock, Senior Vice President – Strategic Marketing & Payout, Grass Valley.

Supported by

For hundreds more high quality original articles and
Essential Guides like this please visit:

thebroadcastbridge.com



01/2020

Supported by

