

EBU

OPERATING EUROVISION AND EURORADIO

THE DYNAMIC MEDIA FACILITY: REFERENCE ARCHITECTURE

WHITE PAPER V2.0



DMF

APRIL 2026

Document History

EBU Committee	TC	
Drafting Group	Dynamic Media Facility (DMF) Group	
First published	September 2024	
Revised	2026-04-17	Correction to Fig 1, 2 & 3 and Annex C (no version number update needed)
	2026-04-15	Version 2.0 updates throughout text. Updated figures and new information added by members

Acknowledgement

EBU technical publications are the work of experts from EBU Members, Associates, Approved Participants and third parties consisting of international standards bodies, industry partners, academic institutions and independent consultants.

Their contribution to EBU technical publications is a very generous act by the individuals concerned and by their employers. The EBU appreciates their efforts and thanks them most sincerely.

This White Paper has been prepared by the System Design and Automation group within the Media Infrastructures and Cybersecurity Strategic Programme and was produced with the assistance of the following:

Members:

Peter Brightwell, Ivan Hassan, Ross Kemp, Roland Rodgers, Phil Tudor, Simon Tuff, Andrew Wilkinson (BBC), Werner Bleisteiner, Christian Hartmann (BR), Richard Muesch (HR), Kenneth Dammyr, Anders Rydmell, Ole Sommarset (NRK), Karl Petermichl (ORF), Alberto Bonacina (RSI), Olivier Joye (RTS), Zakarias Faust (SR), Gregor Amrein (SRF), Benjamin Reber (SRG), Dennis Buhr (SVT), Markus Berg (SWR), Anthony Castreuil, Pieter Lambrechts, Willem Vermost (VRT), Tim Gülzow (WDR), Markus Nygård (YLE).

Associates:

Daniel Beaudry, Anthony Kuzub, Francois Legrand, Simon Patenaude, Felix Poulin, Mathieu Rochon, Angelo Santos (CBC), Andrew Mannino (NBC), Christopher Berry (NPR).

JT-DMF Partners:

Gareth Sylvester-Bradley (NVIDIA), Jean-Philippe Lapointe (Riedel), Thomas Gunkel (Skyline).

EBU Co-ordinators:

Ievgen Kostiukevych, Pavlo Kondratenko, Willem Vermost

Note: All trademarks and trade names that appear in this publication are the property of their respective owners.

Scope

This updated version of the Dynamic Media Facility Reference Architecture White Paper (September 2024) reflects the latest industry learnings on building interoperable, future-proof media systems.

Key words

Dynamic Media Facility, DMF, Reference Architecture, RA, High Level Model, Media Function, Media Exchange, MXL, Cloud, OSI, Media Workload. Orchestration.

Contents

List of Figures.....	iv
The Dynamic Media Facility: Reference Architecture.....	1
Executive Summary.....	1
Introduction and purpose.....	1
Example Use Cases.....	2
MEDIA WORKLOAD LIFECYCLE	3
Design	3
Plan.....	3
Provision	4
Configure	4
Operate.....	4
Finalise & Review	4
Monitor & Update	4
Layered Model.....	5
ORCHESTRATION MODEL.....	7
INFRASTRUCTURE	7
Layer-Specific Considerations.....	8
Sustainability.....	8
Orchestration	8
Monitoring	9
Security.....	9
HOST PLATFORM	9
Orchestration	10
Control	10
Monitoring	10
Security.....	10
CONTAINER PLATFORM	11
Layer-Specific Considerations.....	11
Container Engine	11
Container Networking	12
Orchestration	12
Monitoring	13
Security.....	14

MEDIA EXCHANGE	14
Layer-specific Considerations	15
Shared Access	15
Media Payloads	15
Video Payloads.....	16
Audio Payloads	16
Time-related Data Payloads.....	16
Identity: Flows, Grains and Timestamps	16
Flow Domains	17
Orchestration	17
Control	17
Monitoring	18
Security.....	18
MEDIA FUNCTIONS	18
Layer-Specific Considerations.....	18
Interfaces with External Devices.....	18
Timing	19
Orchestration	20
Control	21
Monitoring	21
Security.....	21
APPLICATION & USER INTERFACE	21
Orchestration	21
Control	22
Monitoring	22
Security.....	22
OPEN TOPICS	23
ANNEX A USE CASES.....	24
Sharing Facility Resources.....	24
Outsourcing Peaks	24
Resilience	25
Sharing Resources between Users	25
Supporting Sustainability	25
Agility in Production.....	26
Shipping Compute to an Event.....	26
Minimise Small-site Footprint	26
Support for New Formats	26
Support for TAMS-enabled Workflows	26
ANNEX B Multi-Vendor & Multi-Workload Example	27
ANNEX C Facility Orchestration Model.....	28

List of Figures

Figure 1 <i>Media Workload Lifecycle</i>	3
Figure 2 <i>Horizontal layers</i>	5
Figure 3 <i>Vertical layers</i>	6

Figure 4 <i>Orchestration Model</i>	7
Figure 5 <i>Infrastructure Orchestration</i>	8
Figure 6 <i>Cluster and Hosts</i>	9
Figure 7 <i>Host Orchestration</i>	10
Figure 8 <i>Containers in Hosts</i>	11
Figure 9 <i>Container Networking</i>	12
Figure 10 <i>Container Orchestration</i>	12
Figure 11 <i>Shared Access</i>	15
Figure 12 <i>Media Exchange Orchestration</i>	17
Figure 13 <i>Input and Output Media Functions in a single Cluster</i>	19
Figure 14 <i>Media Function Orchestration</i>	20
Figure 15 <i>Media Functions Provisioned as Containers</i>	20
Figure 16 <i>Application & UI Orchestration</i>	21
Figure A1 <i>Sharing Facility Resources</i>	24
Figure A2 <i>Outsourcing Peaks</i>	24
Figure A3 <i>Shipping Compute</i>	26
Figure B1 <i>Multi-vendor Example</i>	27
Figure B2 <i>Multi-vendor, Multi-workload Example</i>	27

THE DYNAMIC MEDIA FACILITY: Reference Architecture

EXECUTIVE SUMMARY

Overview

This white paper defines the Reference Architecture (RA) for the Dynamic Media Facility (DMF), a high-level concept to transition media production from rigid, hardware-dependent workflows to **flexible, software-first environments**.

Key Principles

The DMF RA serves as a "lighthouse" to align the media industry, from media organisations, system integrators to developers, around three key principles:

- **Software-centric:** Move towards containerised "Media Functions" on standard IT infrastructure to increase business agility and scalability.
- **Iterative Lifecycle:** Instantiate production facilities dynamically following an iterative lifecycle from *Design* and *Provisioning* to *Review* with continuous *Monitoring*.
- **Interoperable Interfaces:** A six-layer model (*Infrastructure* to *Application & UI*) identifies interfaces for interoperability to reduce the risk of vendor lock-in and stimulate a multi-vendor economy.

This work also aims to inform ongoing collaborative efforts - such as the Joint Taskforce on DMF (**JT-DMF**) and the Media eXchange Layer (**MXL**) SDK open-source project:

INTRODUCTION AND PURPOSE

The Dynamic Media Facility (DMF) is an EBU initiative about how future media productions can benefit highly flexible and dynamic technology approaches. The concepts were introduced in a [2023 White Paper](#), which encouraged the media industry to build interoperable, and preferably containerised, software Media Functions.

A Dynamic Media Facility is an agile broadcast ecosystem that bridges traditional physical endpoints (e.g., cameras and microphones) with a core of dynamically orchestrated Media Functions. By decoupling media processing from hardware, it allows broadcast workflows to be instantly spun up, reconfigured and scaled on demand, creating a highly flexible and future-proof environment for content creation.

Following the publication of the 2023 White Paper, the EBU launched a project to identify use cases where a DMF approach would provide benefits, share experience on relevant technologies, and develop a Reference Architecture, which is the subject of this 2026 White Paper. This document has the following purposes:

- To provide **structure** for users to consider and present their requirements, through models, interfaces and terminology.
- To inform users about suitable **technologies** and **approaches** in the wider IT domain.
- To identify what areas the media industry needs to **work on**.

The Reference Architecture includes:

- Examples of **use cases** that will benefit from a DMF approach.
- A high-level model for the **lifecycle** for Media Workloads.
- A **layered model**, inspired by those used for Open Systems Interconnection (OSI) and cloud architectures.
- An **orchestration model** for a facility.
- A high-performance **Media Exchange** layer using shared access to video, audio and data between Media Functions.
- **Open topics** that have arisen during drafting and updating.

The Reference Architecture is a living document; a first version was prepared for IBC 2024 and further versions add too and update detail to reflect user and industry experience and developments such as the launch of the open-source Media eXchange Layer project.

The Reference Architecture provides a basis to consider vendor offerings in the DMF area, and to inform on future industry activity, such as in the Joint Task Force (**JT-DMF**) announced in September 2025.

EXAMPLE USE CASES

The project identified about a dozen patterns of use cases where a DMF approach would be of benefit, including:

- **Sharing** of resources in a media facility.
- Outsourcing workload **peaks** to another facility.
- Providing **resilience**.
- Shipping compute to an **event**.
- Support for **new formats**.
- Supporting **sustainability**.

Further detail is provided in **Annex A**.

A more detailed example, involving multiple workloads and vendors is shown in **Annex B**.

MEDIA WORKLOAD LIFECYCLE

A **Media Workload** is an assembly of media applications involved in a specific media production. Figure 1 shows typical stages in the lifecycle of a media workload, with each described below.

Figure 1
Media Workload Lifecycle



Design

Obtain **information** about the Media Workload (what production, where and when it is happening, who is involved), the Media Functions that will be involved and any known technical requirements. In practice, similar types of production will have many of the same requirements and can therefore be **templated**.

Update this information (and if appropriate template) to reflect changes to details about the production or its requirements, or issues from later stages. This could even include changing to a different resource provider or Media Function, to avoid vendor lock-in.

Plan

Calculate connectivity, compute and storage **resources** needed for this Media Workload, and identify where these will be placed, including any additional facility or cloud locations for resilience.

Reserve (or schedule the reservation of) resources for the duration of operations and set up authorisations for their access.

Test provisioning and configuration (as below) for this, monitor for correct operation and update planned resources and authorisations to reflect changes and issues from later stages.

Provision

In good time for the production, ensure connection, network, and compute resources are available and configure them for this Media Workload.

Install or update (as needed) and launch Media Functions. Add or remove functions and resources to reflect changes during operation.

Provide alerts and analysis information.

Configure

Configure and connect Media Functions. Change configuration as needed or as provisioning changes, to reflect operational activity.

Provide alerts and analysis information.

Operate

This is the **operational phase**. At startup test functionality and performance are as intended.

Authenticate users for access; users carry out operations via User Interfaces and can make operational or engineering changes, for example changing a camera, changing an audio level or carrying out system updates. These changes are limited to available resources (compute, licences, connectivity, storage, etc.) until the end of the reserved period.

Finalise & Review

Ensure operations have completed, **delete** Media Functions, authorisations and reservations, and ensure all changes made through the workflow have been **recorded** in persistent storage.

Monitor & Update

Monitor & Update is shown at the centre of the diagram and applies throughout the lifecycle to support an **iterative** approach.

Monitor **resource usage** at each layer (see below), looking for changes from expected values and alert, if necessary, connectivity or performance issues or security breaches, for example.

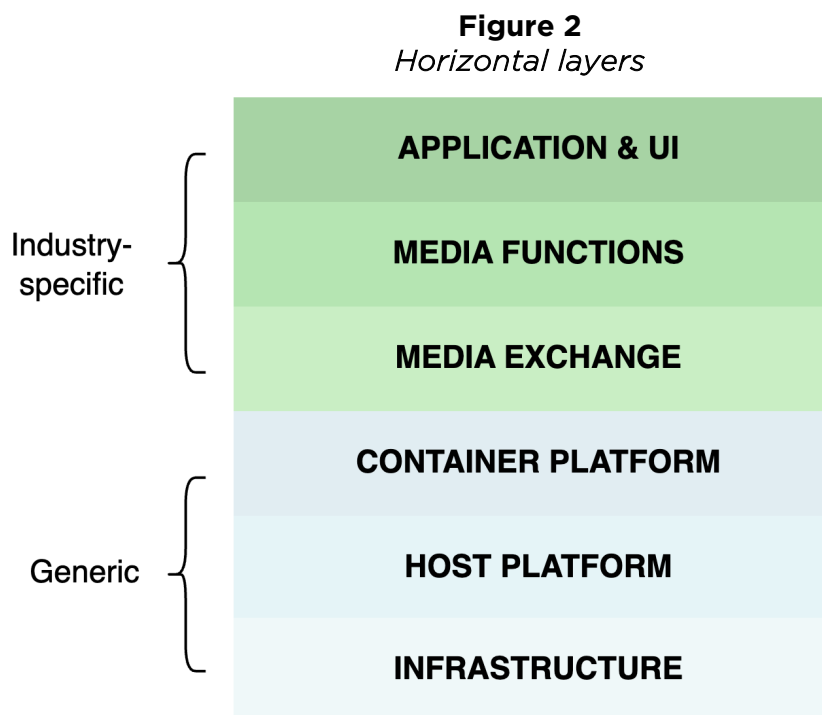
Monitor whether configuration changes have the required effect on resources, for example does a Media Function have the correct parameters set

Gather and document information for **analysis** and use CI/CD techniques to automate changes and ensure updates to infrastructure and software including as part of a wider **maintenance workflow**.

LAYERED MODEL

The Reference Architecture defines a **layered model** inspired by those used for cloud infrastructure. The intention is to enable technology and product choices to be made independently at each layer enabling deployment on-premises, at a remote location or in the public cloud.

The model defines **six horizontal layers** shown in Figure 2 and detailed in the following sections. The lower blue layers are based on **generic technologies**, protocols and specifications, while the higher green layers are allowed to be more specific to the live broadcast/production industry.

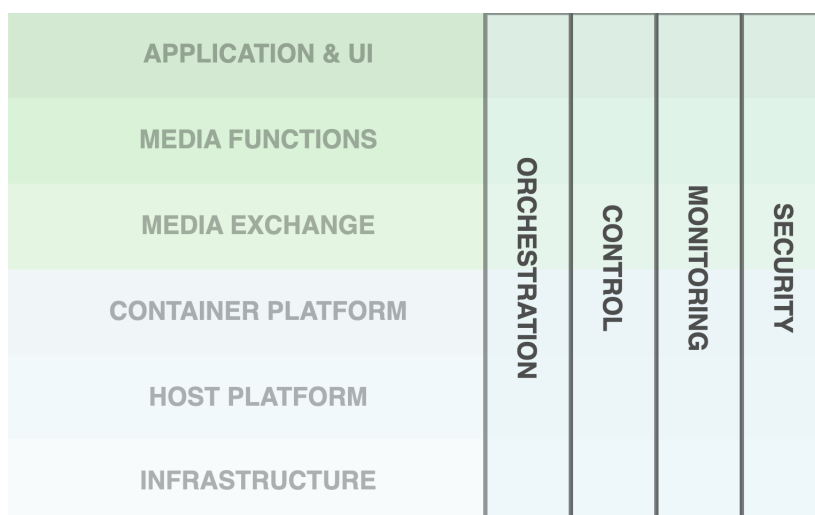


In practice, different broadcasters will make different choices about which providers or vendors they use for their generic (blue) technologies across their facilities. When considering choices for different Applications and Media Functions, it is important to be aware of any limitations imposed by chosen generic (blue) layers, for example because of particular network or graphics hardware requirements.

Also, for use cases such as “Outsourcing Peaks” where Applications and Media Functions run on a public cloud, the cloud provider is in control of the Infrastructure layer and might offer its own implementation for the Host and Container Platform layers.

The Reference Architecture also includes verticals (Figure 3) to represent **cross-cutting** concerns that apply to all layers:

Figure 3
Vertical layers



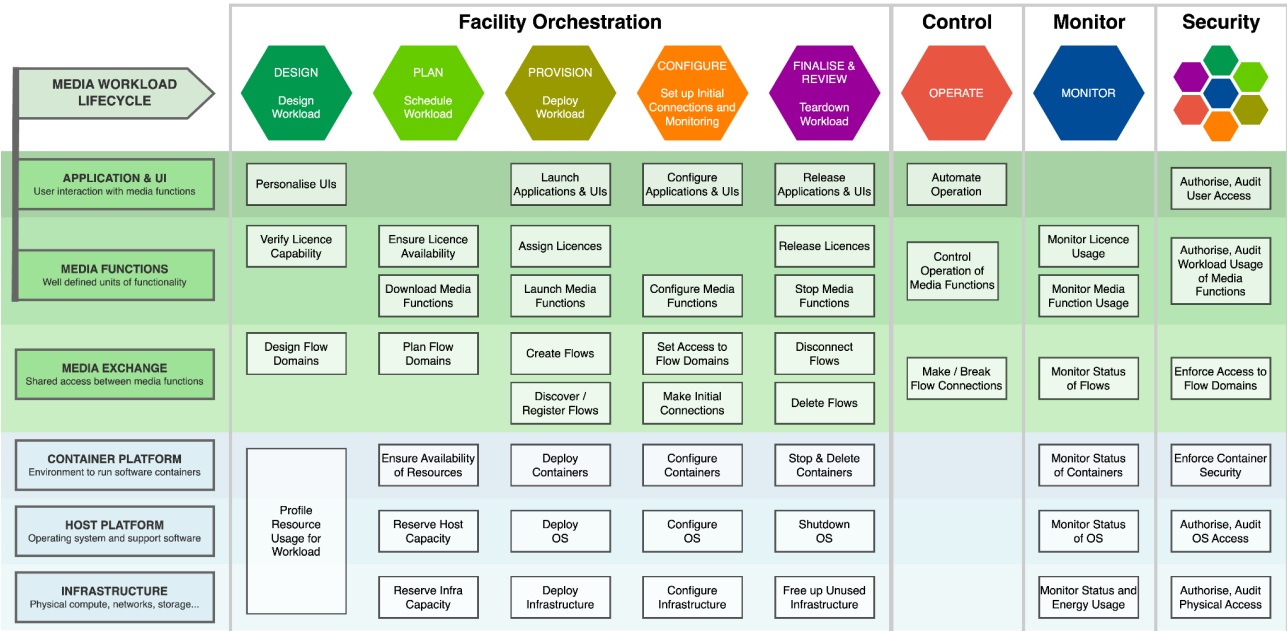
- **Orchestration** creates and destroys resources in each layer and configures resources so they can be controlled and monitored. It ensures that availability and resilience requirements are fulfilled and uses an Infrastructure as Code (IaC) approach for consistency and reusability.
- **Control** enables operational control of resources.
- **Monitoring** enables alerts and analysis.
- **Security** adopts **zero-trust** principles:
 - **AAA:** Authentication, Authorisation and Accounting are enforced, both for users and devices connecting.
 - **Least privilege access:** Users and systems are authorised for the minimum that is needed.
 - **Segmentation:** virtualisation, containerisation, networks and other technologies are used to confine Media Workloads to their own “box” and prevent “lateral movement” by attackers. This enables multi-tenancy, allowing multiple productions and organisations to share the same facility without any default access to each other’s applications, functions or content.
 - **Continuous monitoring and validation:** never assume anything can be trusted; there might be an attacker inside the network.

The EBU publishes security recommendations, for example [EBU R 160](#), which covers **vulnerability management**.

ORCHESTRATION MODEL

Figure 4 shows an **Orchestration model for a facility**. For each layer, relevant activities within the Media Workload lifecycle are shown, and are referenced in the sections below. A larger version is shown in [Annex C](#).

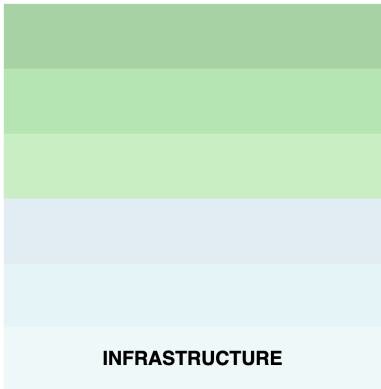
Figure 4
Orchestration Model



INFRASTRUCTURE

The **Infrastructure layer** provides physical resources capable of meeting the performance requirements of the workloads, and typically includes:

- **Compute:** for example, data centre servers suitable for virtualisation and providing out-of-band-management. General compute (CPU), graphics (GPU), neural/ML (TPU/NPU). RAM, including buffer memory for shared access between Media Functions.
- **Network:** for example, 100 Gbit/s and above core connectivity with spine-leaf / Clos topology and border leafs for WAN.
- **Storage:** for example, NVMe-connected SSDs for high performance, possibly combined with HDD/tape.
- **Timing reference:** Precise enough for media essence alignment, for example PTP, NTP, SyncE.



Infrastructure is sized to meet the expected workloads plus an appropriate level of resilience and can be easily extended by adding new servers etc. (c.f., difficulty of resizing the main router in an SDI facility).

Infrastructure can be at multiple locations with appropriate WAN connectivity between sites to support workload relocation and resilience use cases.

Layer-Specific Considerations

Sustainability

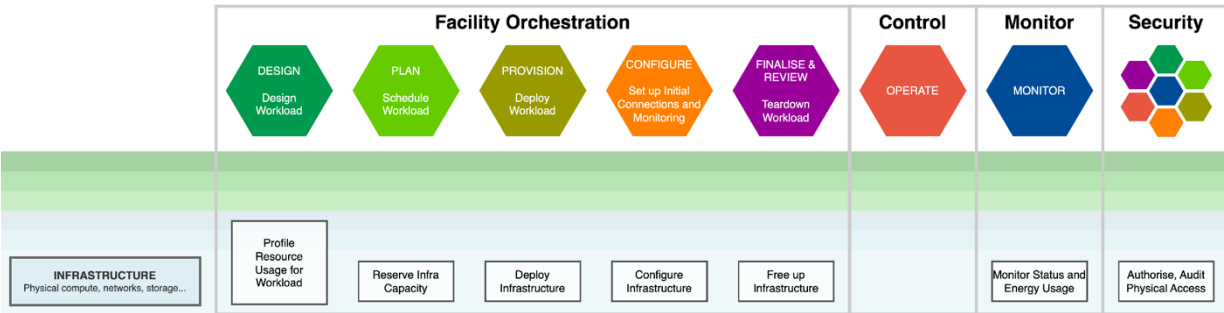
Electrical **energy** consumption of compute (CPU, GPU, and TPU), network, and storage all need to be calculated, monitored and minimised, wherever the infrastructure is located and whoever provides it. This includes both GHG Scope 1 and Scope 2 emissions. Reducing the energy consumption in kWh will reduce the amount of emitted kg CO2e.

GHG Protocol

The GHG Protocol provides standards and guidance for **greenhouse gas accounting**; it classifies emissions into different scopes, with Scope 1 (Direct emissions) and Scope 2 (Indirect emissions from purchased energy) being most relevant here

Orchestration

Figure 5
Infrastructure Orchestration



- Design:** Make a **resource profile** for the Media Workload, describing expected compute (CPU/GPU/DPU), memory, storage and networking requirements, and taking into account, scaling and resilience expectations. This influences the design and sizing of the physical infrastructure in the facility needed for the workload. A Design may include technical characteristics, such as if the production is HD or UHD.
- Plan:** **Reserve** matching infrastructure capacity according to the resource profile, i.e., reserve compute and network resources and licences capable of provisioning the Media Workload with the required size and technical characteristics.
- Provision:** **Deploy** and power-up compute, networking, storage and other physical infrastructure (possibly including installation, racking and cabling).
- Configure:** Low-level **configuration** of physical hosts, switches, etc. Tools such as Ansible are typically used by EBU members.
- During the Operate phase of a Media Workload lifecycle, it may be necessary to **restart** or **power down** infrastructure or make other low-level changes. This includes the use of well-adopted out-of-band management technologies such as iDRAC and IPMI.
- Finalise:** **Free up** physical infrastructure to allow use for waiting Media Workloads, or **power down** if not currently needed.

Monitoring

The Infrastructure layer uses **monitoring** tools that are well adopted in the wider IT community. Some examples already used by EBU members include Observium and Nagios.

Monitoring includes information about **energy** consumption, temperature, etc. This energy monitoring information can be used for accounting in the corresponding Media Workload.

Security

Physical **access** to hardware, and remote access are restricted to authorised users and administrators, and all access is **audited**.

Low-level software and firmware (BIOS etc.) are kept **up to date**, including patching of vulnerabilities as soon as possible.

Networks are designed using **zero-trust segmentation** principles, to minimise the threat of “lateral movement” attacks.

HOST PLATFORM

The Host Platform layer makes the physical resources in the Infrastructure layer available in a way that is useful to the Container Platform layer.

Compute **resources** are organised as **Clusters of Hosts** (Figure 6). Each Host runs an operating system and any other software needed to run Containers. A Host can run directly on the compute hardware (bare metal) or via a type-1 hypervisor (“type-1” means the hypervisor runs directly on the hardware without requiring a separate general-purpose operating system).

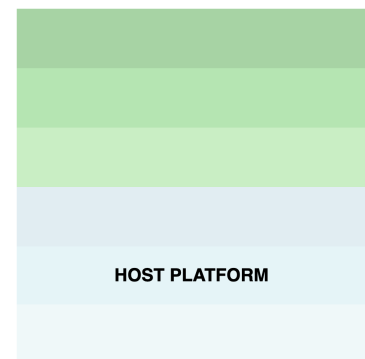
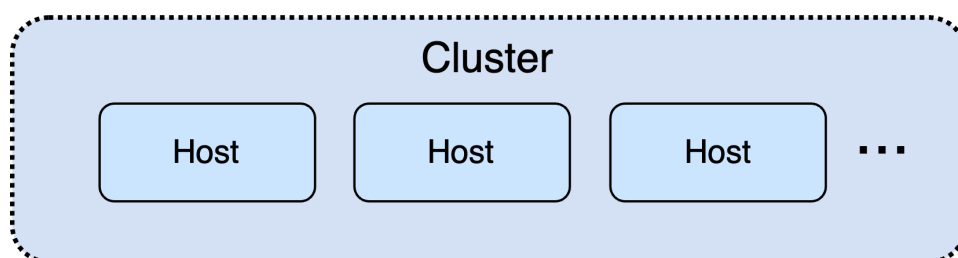


Figure 6
Cluster and Hosts



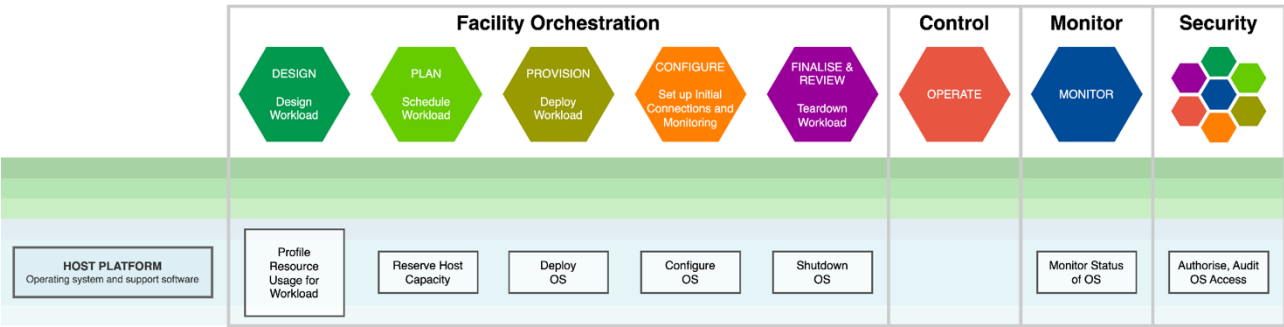
Hosts provide fast access to their hardware, bypassing the kernel where necessary for performance. For example:

- Kernel bypass access to network interfaces.
- Single-root input-output virtualisation (SR-IOV) extends this to virtual machines.
- Remote direct memory access (RDMA) to shared memory.

A Cluster is a group of Hosts at a location, connected using fast networks. Workloads can be deployed onto multiple Clusters, which would be needed for some of the scenarios in **Annex A**.

Orchestration

Figure 7
Host Orchestration



- Design:** Make a **resource profile** for the Media Workload (see above). This influences the size of VMs (if used), selection of operating system, software packages, etc.
- Plan:** **Reserve** capacity according to the resource profile.
- Provision:** **Deploy operating systems** and software packages needed to run the Container Platform using CI/CD (continuous integration/delivery) tools to build ISO or VM images and infrastructure automation tools to boot those images and carry out any further installation or update of packages. Tools such as Terraform are typically used by EBU members.
- Configure:** The operating systems are **configured**, and software is **installed** and **licensed** to run the Container Platform. Configuration makes use of automation tools; examples of tools used by EBU members include GitHub Actions, Ansible and Airflow.
- Finalise:** Operating systems and underlying CPU, GPU, and TPU are put into a **low-power** state or preferably **shutdown** when no longer needed for active Media Workloads.

Control

The Host Platform layer is **controlled** using tools that are well adopted in the wider IT community. For example, command-line tools, management tools such as Ansible, and cloud API tools.

Monitoring

The Host Platform layer is **monitored** using tools that are well adopted in the wider IT community. For example, system log and higher-level monitoring and visualisation tools such as Prometheus, Grafana and InfluxDB.

Security

The Host Platform provides a secure “box” for the Container Platform. This could be through the use of a dedicated bare-metal Host, or by ensuring a VM is sufficiently secure.

Hosts are kept up to date through automated application of vulnerability patches.

Access to operating systems is kept to a minimum, and subject to **AAA** and **zero-trust** principles (for example it is not expected that general production users will have root access to VM Hosts).

CONTAINER PLATFORM

The Container Platform layer provides an environment to package, deploy, run, control, monitor and remove Media Functions as software **Containers**.

Containers are units of software that allow the Media Function code to be packaged with any libraries and dependencies.

A Media Function may consist of a single Container or a composition of multiple Containers.

Containers run on Hosts within a Cluster.

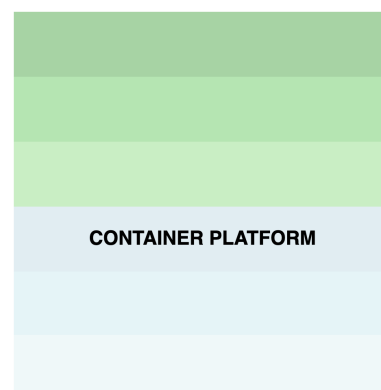
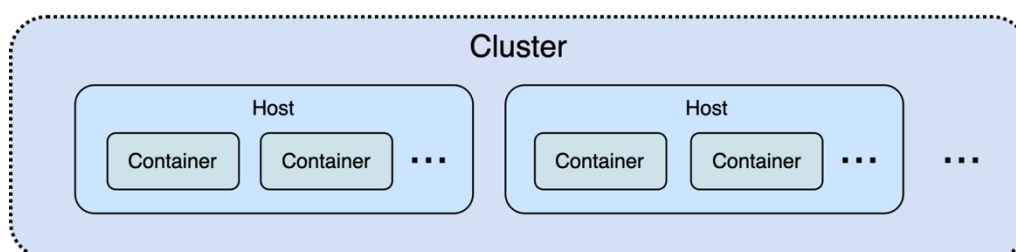


Figure 8
Containers in Hosts



Containers by default are isolated from each other within a Host, which provides isolation to Media Functions on the layer above; the Media Exchange layer provides controlled exchange of media where required.

The Container Platform layer includes:

- **Container Orchestration**, to manage the lifecycles of software Containers.
- **Container Engines** to run the code.
- **Container Networking**, used by the Media Exchange layer and for the control plane used by Container Orchestration.
- Supporting resources such as databases and monitoring, used by the Container Orchestration, and load-balancers.

The implementation chosen for the Container Platform layer is independent of that chosen for the Host Platform layer.

Layer-Specific Considerations

Container Engine

The Container Engine packages and runs the Media Function's code, runtime, system tools, libraries, and settings to ensure consistent operation across different Host Platform and Infrastructure layers.

The Open Container Initiative defines interoperability specifications to package,

distribute, unpack and run Containers in a consistent way.

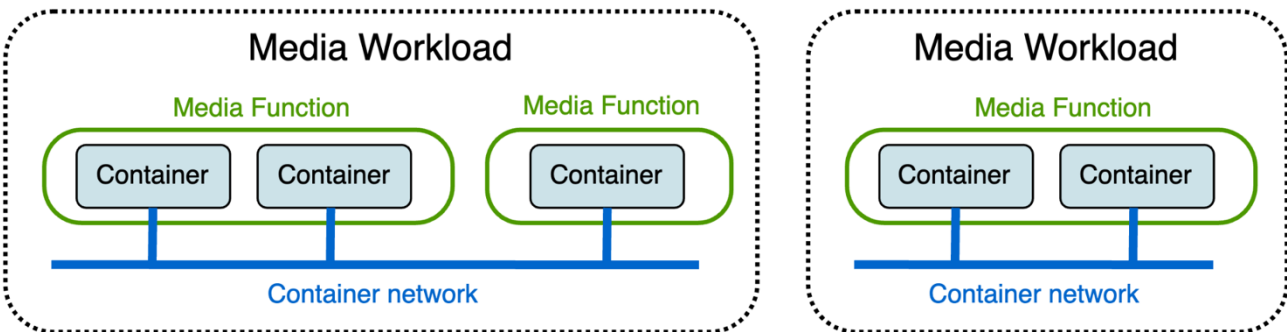
Container Networking

Container Networking provides communications between Containers and external network services by creating virtual networks (a simple example of this can be seen in Docker Compose files). Networks within the same Cluster can, as required:

- Connect Containers that form part of a Media Function,
- Connect Containers between Media Functions in a Media Workload
- Provide isolation between Media Functions from different Media Workloads

Figure 9 shows examples of these.

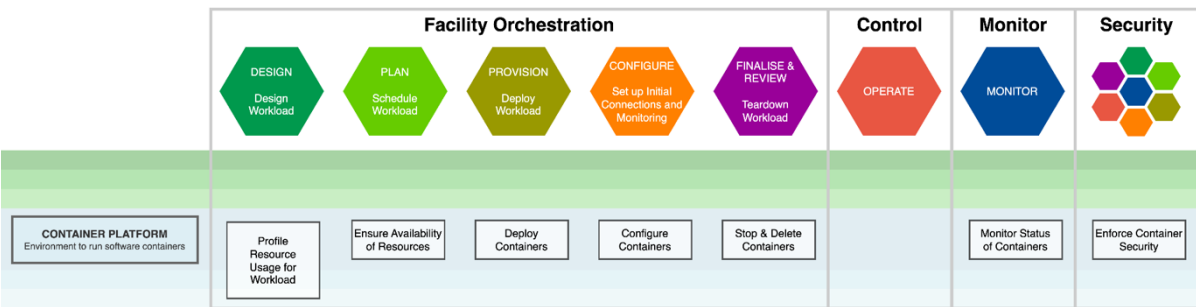
Figure 9
Container Networking



For DMF, high-performance Container Networking enables real-time (and faster) communications, accessed through the Media Exchange layer.

Orchestration

Figure 10
Container Orchestration



Container Orchestration provides provisioning, deployment, scaling, load balancing and resilience of groups of Containers that run on Hosts within a Cluster.

Container Orchestration employs a **declarative** model, meaning it accepts a specification of the application's desired state, including requirements for computational resources, network configuration, and persistent storage, and is responsible for reconciling this specification with the current operational state.

Container Orchestration can automatically perform workload **scheduling** across available computing resources, ensure **fault tolerance** and **high availability**, and provide **self-healing** capabilities by detecting and correcting deviations from the

desired state.

Container Orchestration supports non-disruptive processes for **scaling** and **version management** (e.g., rolling updates) to abstract the underlying infrastructure complexities from application deployment.

A widely used example of Container Orchestration is provided by **Kubernetes**, an open-source system for management of containerised applications. Kubernetes uses specific terminology, referring to Hosts as “Nodes” and groups of Containers as “Pods”.

Container support for **specialist capabilities** such as fast network access, shared memory, GPU access etc. is provided through generally available extensions such as Kubernetes Operators.

Currently this assumes a single Container Orchestration. Provisioning with multiple Orchestrations or across Clusters is subject to further study.

Design: Make a **resource profile** for the Media Workload (see above). This influences the design and sizing of the Cluster, for example:

- Number of Hosts.
- Compute (CPU/GPU/DPU) and memory provided to Hosts.
- Container Network design and performance.
- Load balancers.
- How much the Cluster might need to scale.
- How much resilience to provide.

Plan: When scheduling a Media Workload, the Container Orchestration ensures that sufficient **resources** will be **available** in the Cluster at the required time, **protected** from oversubscription and can **scale** if needed.

Provision: **Clusters** are created (or resized) as required according to the resource profile. Container **images** are downloaded (or updated) as required. **Containers** are started, for example running within a Kubernetes Pod.

Configure: Workload-specific **configuration** required for the Container is performed (for example using k8s control API).
During the Operate phase of a Media Workload lifecycle it may be necessary to scale up or down or **change the configuration** of the running Containers.

Finalise: When a Media Workload lifecycle is completed, any **logs** or other monitoring information that should be retained is saved, and then the Containers providing its Media Functions are **stopped** and **deleted** (for example by stopping the Kubernetes Pod). Container images may be kept for use in later workloads.

Monitoring

Telemetry data is collected from Containers.

Data includes **metrics** on resource consumption (CPU, GPU, memory, I/O), detailed **logs** for debugging and auditing, and **traces** for distributed transaction analysis.

Data can be used to provide **real-time visibility** into the operational status, resource utilisation, and performance health of the Media Functions.

Data can be used to automate **scaling decisions**, trigger **alerts** based on defined thresholds, and provide **feedback** to the Container Orchestration to maintain the application's defined desired state.

Data may be made available to support reporting on **environmental sustainability** and carbon footprint metrics.

Control & Monitoring with multiple Clusters is subject to further study.

Security

Orchestration, control and monitoring operations are subject to **authentication and authorisation** controls.

Container Networking prevents Media Functions accessing other's internals, except through authorised methods (see Media Exchange).

Security with multiple Clusters is subject to further study.

MEDIA EXCHANGE

The Media Exchange layer provides an open (non-proprietary, created by collaboration) method for **high-performance exchange** of uncompressed or compressed live media payloads between software Media Functions running on Containers on the same Host, or on different Hosts in a Cluster.

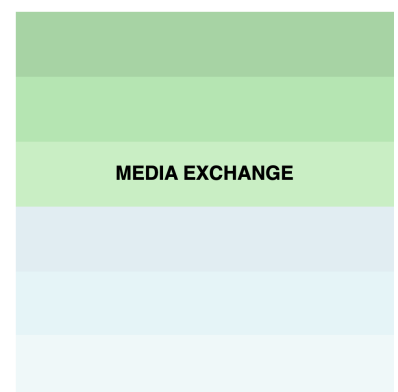
To minimise latency and to avoid excessive copying, it uses **asynchronous shared access** between Containers. This is well-suited to the software-first approach of DMF and it is distinct from streaming protocols (including SMPTE ST 2110) that are used for synchronous operation and typically require copying data.

Shared access is granted through **authorisation**; Media Functions can only exchange with other authorised functions, see **Security** below.

A Media Function can send to a **single** or **multiple** receiving functions.

The layer provides information about the format, codec and representation of the payload.

The layer provides timestamp information for media payloads.



MXL

After publication of the first version of this Reference Architecture, the EBU approached the industry to expedite multi-vendor interoperability on this layer through development of a reference Software Development Kit (SDK) that can be reused by implementers. This is the open-source Media eXchange Layer (MXL) project, hosted at <https://github.com/dmf-mxl/mxl>. The project is adopting an iterative approach, providing a “minimal viable” implementation to expose work early for experimentation and community learning; following iterations will focus the work on improvements based on real-life needs and problems.

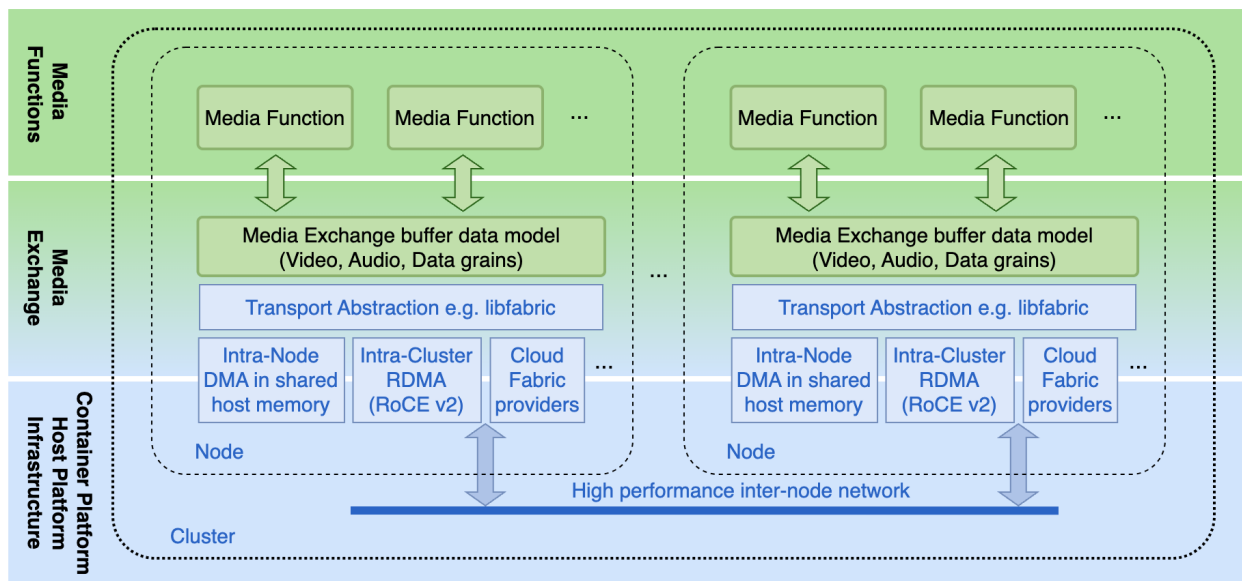
Early public code for the MXL SDK was released in June 2025 and first interoperability was shown at IBC 2025, covering some of the proposals here. Version 1.0.0 was released in early 2026 with the goal of seeing first products for sale between NAB and IBC 2026.

Layer-specific Considerations

Shared Access

Shared access is provided using an existing open IT method, to avoid overhead through the operating system and hypervisor. Within a Host, this is achieved by **local shared memory** (Direct Memory Access (DMA)). Between Hosts of the Cluster, different **RDMA** (remote DMA) techniques can be used, depending if the memory is shared on a LAN or in a Cloud network or if memory is exchanged with Graphic Processors (GPU). Therefore, an **abstraction** layer that hides the complexity of the specific memory fabric is included.

Figure 11
Shared Access



Libfabric is an example of a transport abstraction layer. It is an open-source framework that supports different provider protocols. Example protocols include RoCE v2 (RDMA over Converged Ethernet), which allows remote DMA over layer 3 networks, and “cloud fabric” protocols provided by cloud vendors for fast networking within a Cluster (for example Elastic Fabric Adapter).

OpenUCX is another abstraction layer that offers support for memory exchange with Graphic Processors (GPU). Also, Ultra Ethernet is an emerging open communication standard designed to meet the extreme demands of Artificial Intelligence (AI) and High-Performance Computing (HPC) at massive scale.

Media Payloads

Media payloads are **compute-efficient** for the Media Function to process and **transparent** for processing production formats.

Media payloads are by default, **mono-essence** and **uncompressed** with representations that are optimal for computing and practical for the implementation of Media Functions.

Additional representations and compressed or multiplexed payloads are also possible, however there is a wish to keep the number of operational points to the minimum for

the ease of utilisation. The Media Exchange layer provides information about the format and representation.

For consideration of stream connections in and out of a Dynamic Media Facility see the section on **Media Functions**.

Video Payloads

As part of an iterative approach, initial focus is given to the representation of a full frame of Full HD - 1080p YUV 10 bit.

Then further work will add more format options as needed such as alpha channel, UHD, partial frames (for example small stripes can be chosen to keep latency small without the overhead of sample-rate packing), etc.

To enable connections between functions beyond the defaults, Media Functions support querying of what formats and representations are supported for sending and receiving.

Audio Payloads

The Media Exchange layer uses payloads that equal or exceed the bit-depth and sample rate at the input and output of the Dynamic Media Facility (see “Interfaces with External Devices” below). Typically, this will be 24-bit linear PCM at 48 kHz (this is consistent with AES67 and ST 2110-30).

There are many benefits to exchanging the audio between Media Functions as 32-bit floats which allow for a wider dynamic range, easy fading operation and that is a usual format for proprietary in compute exchange of audio.

Audio channel count should be flexible to support typical audio objects arrangement such as single, dual (stereo), 6-channel (5.1), etc.

Time-related Data Payloads

Time-related data (for example, subtitle, timecode and SCTE triggers) shared between Media Functions use a representation that is widely adopted in the IT industry, such as JSON or protocol buffers.

Media Functions validate received data against a schema.

Further work will define a representation and schema specification.

Media Functions support querying of the formats and representations supported for sending and receiving data.

Identity: Flows, Grains and Timestamps

With the **asynchronous** nature of the Media Exchange layer, it is key that each media payload carries a **timestamp** that represents the absolute time it originated, or a relative time to other streams which belong to the same time context.

This Reference Architecture adopts the model and definitions of the JT-NM Reference Architecture, in which media is treated as uniquely identified Flows of timestamped Grains. This supports, for example, cases where all Flows from the same scene need to be presented with the same time alignment.

The coherent utilisation of this time and identity information is further developed in the Media Function layer.

Flow Domains

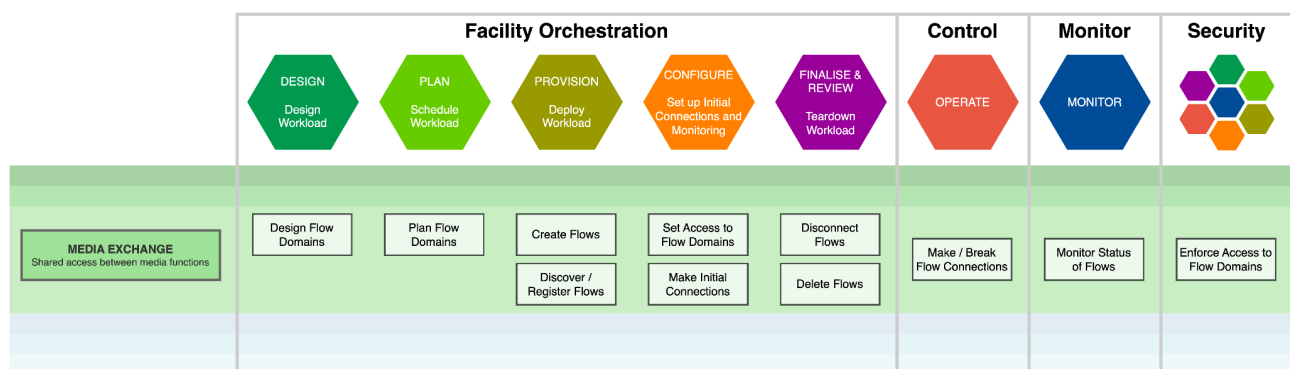
The Media Exchange layer defines **Domains** as a mechanism to identify which Flows can be accessed by Media Functions.

MXL Hosts, Domains, Flows and Grains

In the current MXL SDK, a Flow can be addressed by the tuple {Host ID, Domain ID, Flow ID}, which uniquely identifies a directory in which memory-mapped files hold Grains.

Orchestration

Figure 12
Media Exchange Orchestration



Orchestration in the Media Exchange layer manages the lifecycle of Domains and Flows and how Media Functions use them in a Media Workload.

- Design:** Identify what **Flows** are needed between Media Functions, and how these will be isolated using **Domains**.
- Plan:** When scheduling a Media Workload, plan the Flow Domains that will be used.
- Provision:** **Create the Flows** within the Domains and make them **discoverable**.
- Configure:** Make **initial connections** for the Media Workload, using for example the MXL API.
- Finalise:** **Disconnect** Media Functions from the Flows and **delete** the shared memory for the Flows and Domains.

Control

During operation, **connections** can be dynamically added, removed or changed, for example when additional Media Functions are added for scalability/resilience reasons.

NMOS and shared media

Media Functions could be considered as NMOS Nodes (as defined in AMWA IS-04) within the DMF facility, and Media Exchange could be considered as NMOS Flows. At present NMOS only defines transports for streaming protocols such as RTP. However, we can envisage a shared access NMOS transport specification, which would then allow the NMOS format mappings and other features to be leveraged.

Monitoring

The Media Exchange layer provides real-time information about the **status of Flows**, such as presence/absence of payloads, throughput, latency, etc.

Security

Media Functions are **authorised** to write to or read from shared buffers subject to what is allowed in the Flow Domain plan. This allows isolation between different Media Workloads in the same Cluster.

Media Exchange APIs are encrypted and authorised. The approach used in NMOS and Catena (TLS 1.3 and OAuth 2.0) is suggested.

Encryption of the media payloads is not in scope.

MXL Security

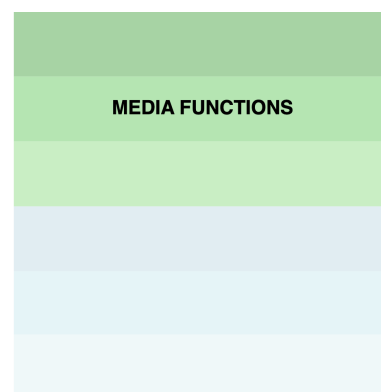
MXL defines “Domains”, which use Unix file system access controls.

MXL’s Fabric API enforces authentication and authorisation to allow Flows to be shared between Hosts.

MEDIA FUNCTIONS

The Media Functions layer provides well-defined units of functionality that act on media streams, files or objects. Examples of Media Functions include:

- Receive contribution streams.
- Format conversion.
- Video and audio mixing.
- Colour correction.
- Audio processing.
- Graphics rendering.



Systems that do not act on media streams are considered external to the Media Function layer; for example, an HTML5 graphics server would be external, whereas the graphics renderer would be a Media Function.

Media Functions are combined to implement Media Workloads, such as shown in “Multi-vendor and multi-workload example” in [Annex B](#).

Media Functions are **stateless**, meaning that they treat API requests independently, and do not retain information between requests – that is the responsibility of higher layers and important for a scalable approach to orchestration.

Layer-Specific Considerations

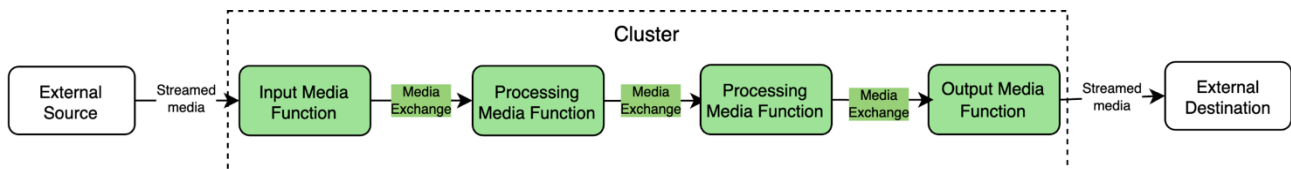
Interfaces with External Devices

While the Media Exchange layer provides shared media access between Media Functions, connections from Media Functions to **external devices** use streamed media or files.

Uncompressed local stream connections preferably use standard protocols, including SMPTE ST 2110 or (for audio) AES-67. Ideally, such connections will comply with EBU **Tech 3371** (“the pyramid”) and JT-NM TR-1001-1, i.e. ST 2110 transport, registration of an IS-04 NMOS Node and connection management using IS-05. Note that this means that input and output Media Functions at the edge of a Cluster (see Figure 13) will need to support ST 2110-21 packet pacing and PTP.

Media Functions are also likely to provide support for other streaming formats, such as JPEG XS over MPEG-TS (TR-07) or ST 2110 (TR-08), IPMX, NDI, SRT and H.265 TS.

Figure 13
Input and Output Media Functions in a single Cluster



Timing

Within a Cluster, Media Functions support **asynchronous** media access, so a function can read video and audio data as soon as it is ready, without the need for unnecessary delays caused by frame buffering as might happen in a traditional “genlocked” environment. This can keep latencies low and allow faster-than-real-time reading and writing to storage.

External sources and destinations could be asynchronous or synchronous and normally connected by streams operating at a fixed frame or audio sample rate (cameras are usually fixed frame rate). To support later alignment, streams arriving from external sources are timestamped at ingress into the first Media Function of the DMF. This timestamping is consistent with the approach specified in the JT-NM Reference Architecture and AMWA MS-04 NMOS Timing Model. In this approach, a **Grain** (a frame or other period of media) is given one or more time-of-day **timestamps**, which when combined with the unique id of its source provides a robust basis for later time alignment.

Although current ST 2110 implementations and specifications do not include these timestamps, we note that there has been considerable recent work in SMPTE and VSF on the subject of joined up timing, using concepts such as link offset delay signalling, and this will be studied further for its applicability to DMF timing workflows.

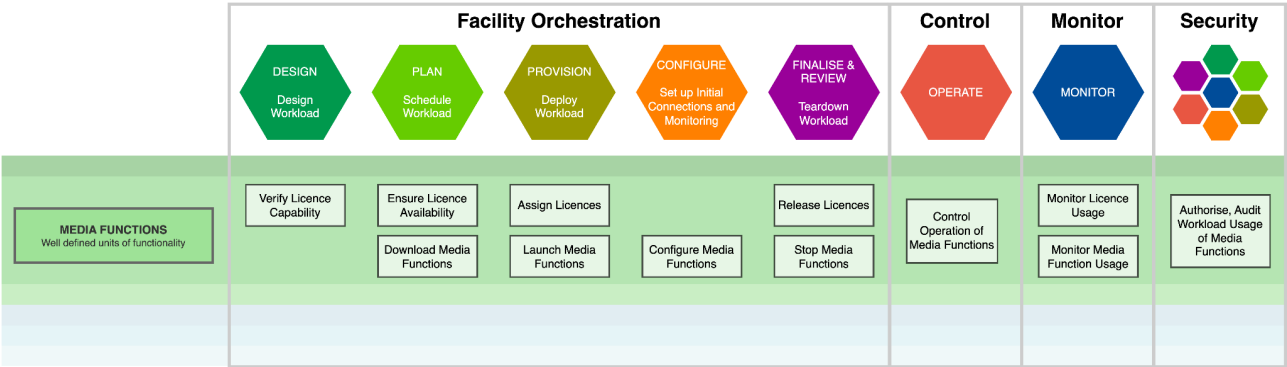
Timestamps are sufficiently accurate and precise to support intended alignment between streams at specific points in the pipeline, for example when mixing cameras at a live event. Any alignment information is preserved through pipelines.

Asynchronous operation

A move to asynchronous operation could have operational implications for production teams who are used to everything being genlocked. This will require further discussion, as will the implications on production intercom.

Orchestration

Figure 14
Media Function Orchestration

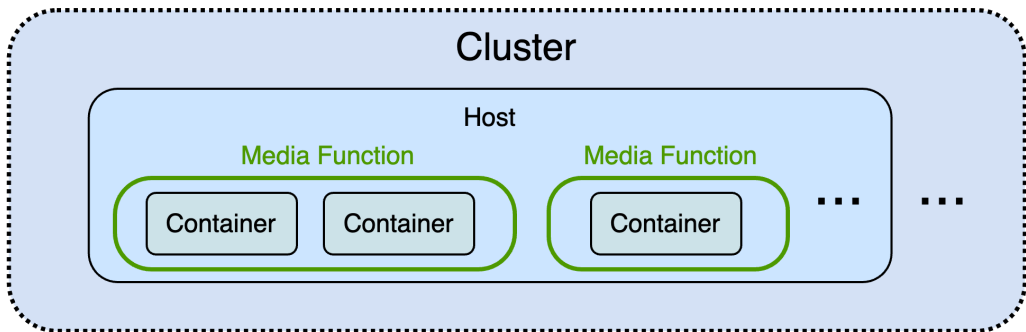


Media Function Orchestration is concerned with:

- **Managing** the resources needed for the Media Function
- Ensuring the correct **versions** of the Container images are used
- Managing **licences**.

A Media Function is provisioned using **one or more Containers** as described in the **Container Platform** section.

Figure 15
Media Functions Provisioned as Containers



Media Functions vendors make their Containers available to the Container Platform layer through widely adopted repository mechanisms (users are now very familiar with using facilities such as Docker Hub) and via common API/CLI tooling. Similarly, the configuration for the function packages is in a widely adopted format, for example Kubernetes typically uses Helm Charts.

- Design:** **Identify** the Media Functions required for the Media Workload and verify that they can **be licensed**.
- Plan:** When scheduling a Media Workload, **download** or update the Media Function as required and ensure that **licences are available**.
- Provision:** **Assign licences** and **launch** Media Functions.
- Configure:** Make **initial configuration** of the Media Functions. As required use the Media Exchange layer to make **connections**.
- Finalise:** Use the Media Exchange layer to **disconnect** and then **stop** Media Functions and **release** licences.

Control

Media Functions can be **discovered, configured, connected** and **controlled** using open APIs. Where appropriate this will build on work from the AMWA Networked Media Open Specifications (NMOS), for example the logical resource model (Device, Flow, etc.) from IS-04.

Monitoring

Media Functions support **real-time monitoring** through open APIs.

Security

All control and monitoring traffic mentioned in the previous sections is **encrypted**, using an open approach, at least as secure as TLS 1.3.

All control and monitoring clients are **authorised** using an approach at least as secure as OAuth2/JWT.

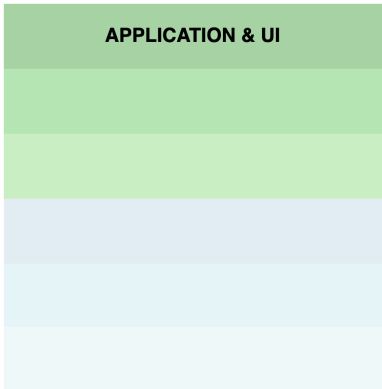
Further work is advised on modern security practices.

Encryption of media traffic itself is subject to further study – the overhead over encryption for high-rate shared media access within a Cluster needs to be considered. Between Clusters or when streaming to third parties it becomes more important

APPLICATION & USER INTERFACE

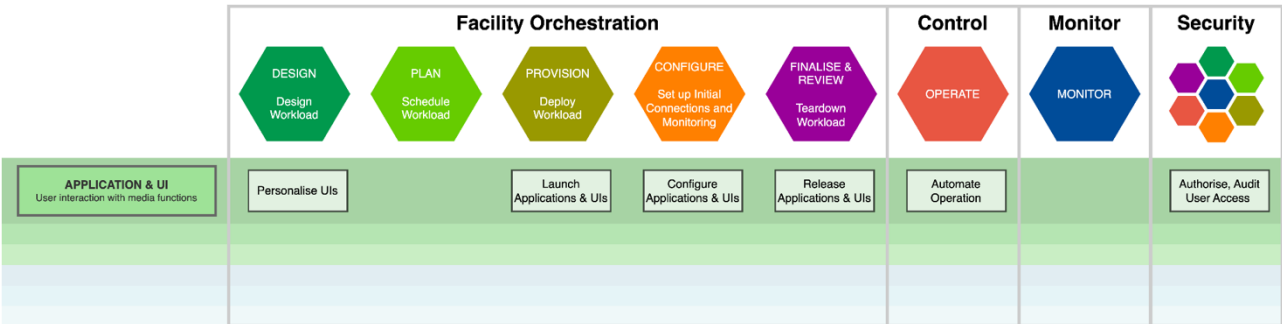
The Application & User Interface layer allows **users** to carry out operations provided through one or more Media Functions, **without the need for specialist knowledge** of what is happening at lower layers. It provides graphical, text or physical **user interfaces**, control surfaces and monitoring information.

Front-end User Interfaces are provided by a web browser or generic hardware, exposing control through standard Internet protocols such as HTML5, WebRTC, open REST APIs or WebSockets. AMWA IS-12 NMOS Control Protocol is an example of open specification based on WebSockets for live operation and status monitoring.



Orchestration

Figure 16
Application & UI Orchestration



The Application and UI Orchestration layer takes care of setup and automation for applications and their user interfaces

- Design:** Create **personalised UIs** for the Media Workload.
- Provision:** **Launch** Applications and UIs.
- Configure:** **Configure** the Applications and UIs.
- Finalise:** **Stop** the Applications and UIs and **release** the resources used.

Control

Automation applies changes to Media Functions to corresponding to live user operations through a UI or pre-prepared operations, e.g. based on a schedule.

Monitoring

The monitoring information from the Media Function layer can be displayed to an operator.

Security

User interfaces support modern, streamlined **authentication** and **authorisation** methods, in compliance with the minimum security baseline defined in **EBU R 143** *Cybersecurity for media vendor systems, software & services*. This includes integration with Single Sign-On (SSO) and other enterprise identity solutions to facilitate easy and secure access.

User interfaces and login mechanisms are designed for maximum usability and minimum friction, supporting common online identity strategies to enable rapid, compliant user access while strictly enforcing the underlying **zero-trust** principles.

OPEN TOPICS

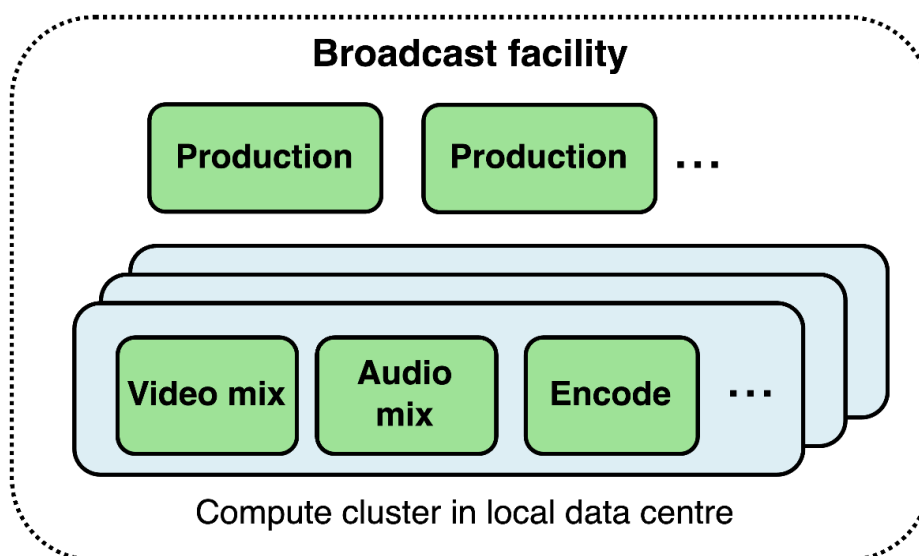
This section captures areas that have arisen during the drafting of this Reference Architecture, to be addressed in later phases of the work, including as JT-DMF activities.

- End-to-end synchronisation when DMF integrates with synchronous systems (proposed JT-DMF activity).
- Interworking with live and file-based workflows and timing/identity; relationship with the TAMS (Time-Addressable Media Store) project (see related use case in [Annex A](#)).
- Patterns for resource management that allow multiple vendors' code to co-exist in a Cluster (proposed JT-DMF activity).
- How Media Exchange Orchestration can support dynamic connections during operation (proposed JT-DMF activity).
- Provisioning and Control & Monitoring across multiple Clusters.
- Media Exchange and Timing between Clusters maintaining asynchronous operation, etc.
- Resilience mechanisms at every layer need to be covered in more detail, e.g. design patterns, robustness of Media Exchange when Grains are missing.
- Include a cycle for system maintenance that takes place between Media Workload lifecycles for adding new components and testing new versions that can cause a partial or total unavailability of the system.

ANNEX A USE CASES

Sharing Facility Resources

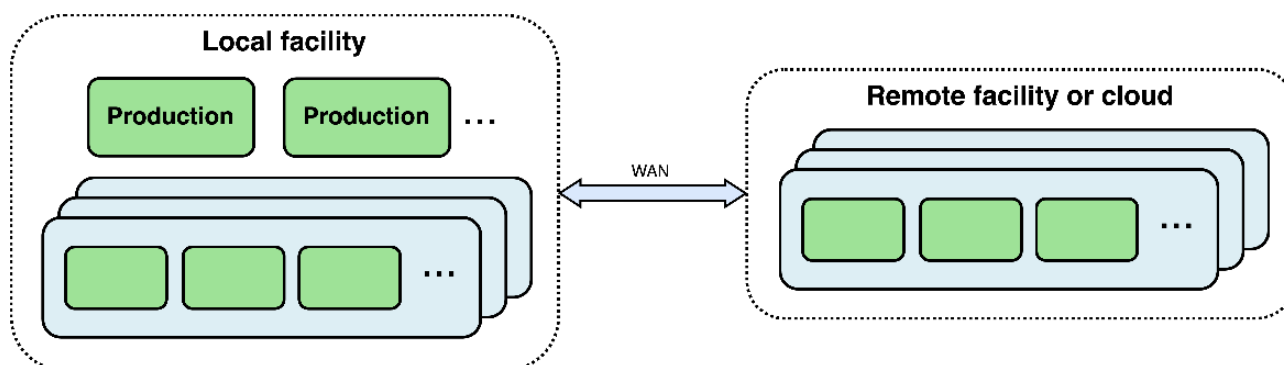
Figure A1
Sharing Facility Resources



In a facility with multiple studios and other operational areas, each with its own equipment, resources are often idle much of the time. But as different productions might be “live” at different times, we can achieve better utilisation by scheduling and provisioning on generic Hosts running within a shared Cluster (or Clusters) in a data centre. As there will be many common functions across productions, for example, video/audio mixing, graphics, effects, record, playback, and encode. We can also benefit from a common software approach.

Outsourcing Peaks

Figure A2
Outsourcing Peaks



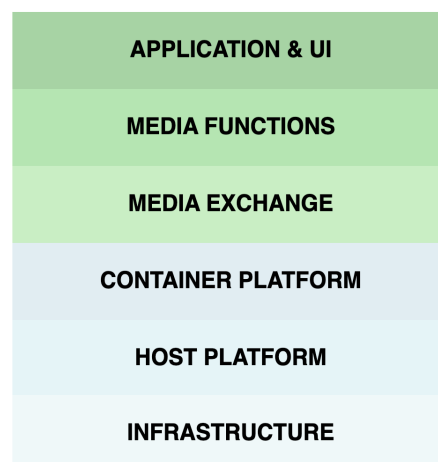
Where productions need to temporarily scale beyond the local capacity, for example, because of breaking news items, broadcasters can take advantage of fast WAN connections to a Cluster in a remote facility, or to the cloud.

Resilience

The traditional approach to resilience in a broadcast facility is to duplicate equipment and connectivity. Although this is still appropriate in some cases (e.g. dual-link connections to a camera), a DMF Cluster provides more flexibility, as it can be sized to provide the desired level of resilience (for example it might have 50% more Hosts than necessary for expected capacity, possibly including some Hosts that are kept powered down on in a low-power state and added to the Cluster when needed).

Resilience is relevant to all layers of the Reference Architecture:

- **Infrastructure:** provide additional servers and network fabric.
- **Host Platform:** provision additional Hosts (including creating additional VM instances where appropriate).
- **Container Platform:** add Hosts to the Cluster and make more Containers available (e.g. by increasing the size of Kubernetes Pods)..
- **Media Exchange:** increase the amount of shared media access buffers.
- **Media Function:** run additional Media Functions with load balancing or 1+1 redundancy.
- **Application & UI:** for example, use a load balancer on web interfaces and extra control panels.



Further resilience can be provided by running Media Functions on infrastructure located at a different facility

Sharing Resources between Users

Resources can be shared between broadcasters or other users using the techniques covered in the above use cases, for example:

- Two broadcasters might be co-located and access resources in the same data centre.
- Two broadcasters might make arrangements to use each other's facilities to provide resilience.
- A broadcaster might rent unused capacity at another broadcaster's facility to help with outsourcing peaks.

Supporting Sustainability

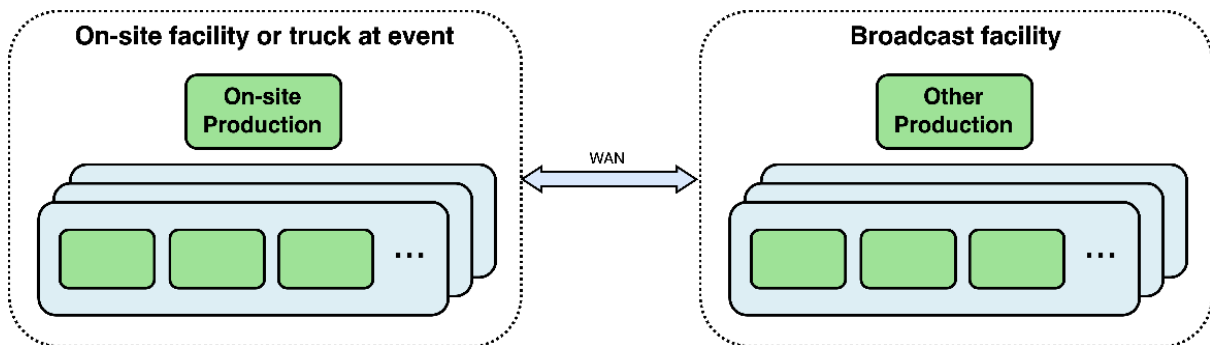
While physical facilities still require the emission and embedding of kgCO₂e in their construction, adopting a dynamic approach enables carbon saving, by allowing the physical infrastructure to provide multiple types of workloads (as discussed above), allowing better overall utilisation and allowing resources to match workload patterns on a daily, weekly, monthly or annual basis. Infrastructure can be powered down or put in a low-power state during less busy times. Resources are sized and scheduled to meet what is needed for the workload at the planned time to avoid unwanted energy consumption and corresponding kgCO₂e. Resource monitoring allows broadcasters to validate usage against workloads and update the plan accordingly.

Agility in Production

When unforeseen changes occur and sudden changes in how or where a production needs to take place, Media Functions can be modified or relocated across a WAN to enable those changes or make them less disruptive.

Shipping Compute to an Event

Figure A3
Shipping Compute



Where an event has significant on-site requirements, such as a football match with many cameras, or a music festival with several stages, there will be production teams both on site and at the main broadcast facility. Depending on the connectivity available, it could make sense to provide compute at the event with WAN connections to the main facility.

Minimise Small-site Footprint

Many broadcasters are looking to consolidate their technical facilities and support, and to reduce the hardware, maintenance and power required at smaller sites, by relocating some equipment to central locations. The DMF architecture supports this by allowing Clusters to be located where there is business need.

Support for New Formats

While 1080p and 1080i are still commonplace, broadcasters will move some production to UHD and HDR, and their short-form content is likely to include portrait aspect ratios. Object-based and other new audio formats are also expected. The fully software architecture of DMF allows the same architecture to be used for current and future formats, including adding capacity to existing Clusters where needed.

Support for TAMS-enabled Workflows

The **Time Addressable Media Store (TAMS)** is an open-source API specification for storing chunks of media as objects. It is based on work from BBC Research & Development to provide a flexible “content-centric” alternative to traditional file-based working. Like DMF, it uses a software-first approach that (like DMF), is consistent with the architecture proposed by the JT-NM. This means that DMF’s Media Exchange layer Grains can be grouped as TAMS Flow Segments, allowing timing information to be preserved from capture, through live production into post-production, broadcast, social media and other downstream operations.

ANNEX B

Multi-Vendor & Multi-Workload Example

Figure B1 shows an example of how Media Functions from multiple vendors can be deployed in a Media Workload, in this case for a **two-camera news studio**. This allows a choice of vendors with different specialisms (similar to the “traditional” approach of connecting hardware appliances from different vendors, but with software functions).

Figure B1
Multi-vendor Example

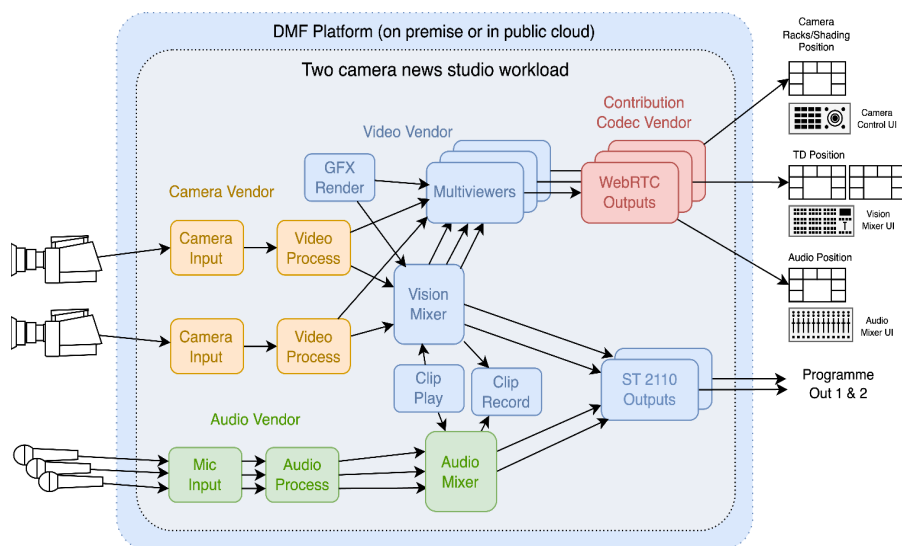
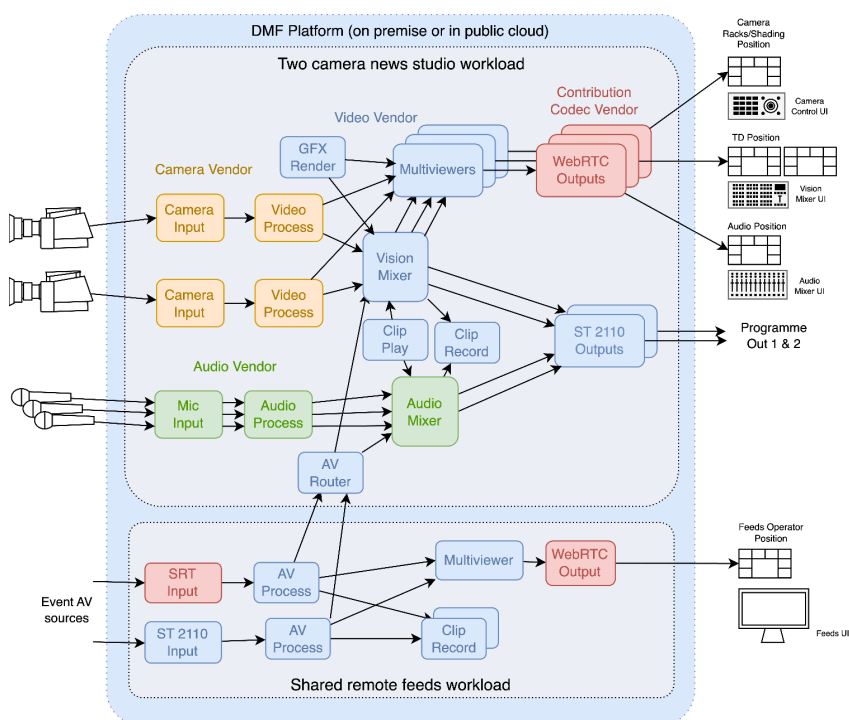


Figure B2 extends this example to show a second Media Workload on the platform, in this case to provide **remote feeds** from an event:

Figure B2
Multi-vendor, Multi-workload Example



ANNEX C

Facility Orchestration Model

