D7.2 Prototype of Asset Store

Grant Agreement nr	780470
Project acronym	SAUCE
Project start date (duration)	January 1st 2018 (36 months)
Document due:	June 30th 2020
Actual delivery date	June 30th 2020
Leader	Foundry
Reply to	Peri Friend
Document status	Submission Version

Project funded by H2020 from the European Commission

Project ref. no.	780470
Project acronym	SAUCE
Project full title	**S**mart **A**sset re-**U**se in **C**reative **E**nvironments
Document name	D7.2 Prototype of Asset Store
Security (distribution level)	Public
Contractual date of delivery	June 30th 2020
Actual date of delivery	June 30th 2020
Deliverable name	Prototype of Asset Store
Type	Demo
Status & version	Submission Version
Number of pages	13
WP / Task responsible	Foundry
Other contributors	None
Author(s)	Peri Friend, Sam Hudson, Dan Ring
EC Project Officer	Ms Adelina Cornelia Dinu - adelina-cornelia.dinu@ec.europa.eu
Abstract	Currently assets are stored local or cloud-only. FO bridges the gap between cloud and on-premise storage, depending on user requirements. Task D7.1 planned out the design to cater for the different types of assets and the load- sharing between local and cloud servers, focusing on redundancy and availability. This deliverable revisits some of those key points, describes how USD and light fields can be integrated into the store and demonstrates a simple plugin to show how an external application can be written with relative simplicity to interact with the Flix Server Storage API to retrieve assets stored in the asset store. We use this plugin to demonstrate key high availability features.
Keywords	Asset store, asset pipeline, cloud
Sent to peer reviewer	Yes
Peer review completed	Yes
Circulated to partners	No
Read by partners	No
Mgt. Board approval	No

Document History

Version and date	Reason for Change
1.0 24-04-20	document created by Peri Friend
1.1 23-06-20	Version for internal review
1.1.1 25-06-20	Revisions after initial feedback
1.2 30-06-20	Revisions in response to review: final version submitted to Commission

Table of Contents

1 EXECUTIVE SUMMARY

This demo document begins by bringing some key information from the previous consortium report D7.1 to give the reader an overview of the design of the asset store and how the APIs, which have been designed to focus on redundancy and availability, allow integration from other SAUCE project partners. This asset store allows Foundry to bridge the gap between cloud and on-premise storage, depending on user requirements. Information on how data types such as USD and light fields have been integrated into the asset store as well as customisable tagging for searching and dynamic lineups of assets is also provided; allowing assets to be tagged with simple tags, colours etc.

The demo shows how an external application can be written with relative simplicity to interact with the Flix Server Storage API to retrieve assets stored in the asset store. We demo this with a simple plugin which receives assets stored in the asset store. This plugin demonstrates key high availability features, by removing running nodes during runtime and showing no interruption to the user experience during that time.

1.1 Relationship to Self Assessment and other Deliverables

This deliverable follows on from D7.1 Asset Store Design, which built on the investigation done in WP2 into asset types and reuse scenarios and incorporates the light field assets created in WP3. This work package links directly with DNEG's WP4 Search framework, and WP5 Transformation framework to create an entire asset pipeline solution.
The Asset store is at TRL8/9 as it is already released and in use with customers and being continually improved. The asset store is the basis for D7.3 Final Digital Asset Store Supporting Geolocation at M33 and D5.6 Transcode Mechanism, both of which go from TLR2 Basic Research to TLR6 Technology Demonstration.
The asset store will both reduce asset retrieval time, and allow for predictive asset management, as laid out in D1.2 Self Assessment plan. Details of how this will be assessed are included in D8.3, also submitted at M30.

2 BACKGROUND

Flix is a storyboard-development software for collaborative work in animated film, TV and gaming that promotes fast-paced collaboration by removing technical barriers that slow production down. Flix, winner of a 2013 HPA Engineering Excellence Award, was developed as an in-house solution at Sony Pictures Imageworks before FO acquired it and it has been used in production since 2008 on more than 15 projects including Cloudy with a Chance of Meatballs 2, The Smurfs and Hotel Transylvania. It is currently integrated with specific industry leading applications such as Photoshop, Storyboard Pro, Avid, Premiere, Final Cut Pro. Flix will be extended (beyond just storyboards) so that the data model can be used in new contexts, for different post-production tasks and purposes. It will be used by users in varying environments and will require supporting new types of assets for parts of the pipeline such as lighting & texturing, modelling, animation, FX and compositing. Flix has been extended in a modular way so as to cater for the variety of types of new assets that need to be supported. This structure will additionally allow for other applications to interface with Flix.

In order for the Flix application to be reusable in new scenarios the asset storage mechanism has been re-designed to meet the requirements of not only the other SAUCE project partners, but also for the Flix product, and future applications.

3 INTRODUCTION

The asset store has been built as a result of understanding the requirements from studios such as DNEG as well as smaller studios who's use case varies considerably. A common, and growing problem is access to data. With more and more of the workforce becoming remote workers, freelancers or in co-located offices, the challenge of making data accessible to a user wherever they are becomes more desirable. Currently, many studios simply use network shares, NFS or SMB to make filesystems available to their users, which is generally fine, but poses problems with federating access to smaller groups, or remote users due to VPN requirements. We have taken this requirement into consideration and have devised a mechanism to allow for the asset store to be outside of the firewall and available, securely to end-users across the internet, without the need to expose insecure fileshares and complex VPN management.

3.1 Main objectives and goals

The asset store caters for the different types of assets such as light fields and USD and the load balancing of traffic to spread load across servers, focusing on extensibility, redundancy and high availability. The asset store mechanism has the following features and philosophies in mind:

- HTTP based RESTful API to provide the main interface for interoperability with the asset storage system.
- The asset storage is to be type-agnostic, allowing for any format to be stored and retrieved.
- The size on disk of assets should only be limited by the available backend storage hardware. Meaning that the storage system should be able to cope with large asset transfers (such as DNeg's crowd assets or USAAR's light field assets) without using up the hardware's resources (CPU, RAM) unnecessarily.
- Provide the ability for assets to have metadata, and tags to be associated to provide a mechanism for search material, and relationship data.
- Provide a plugin mechanism to allow for the asset store to be extended with custom features, written by customers, other SAUCE partners, or FO. This will form the basis of integrating the service with other partners software.

The system provides a simple, yet powerful asset storage system with a low barrier to entry. Enabling small to large studios, single users, or even hobbyists to use the system without requiring a complex, proprietary pipeline for their asset storage.

The asset store is the foundation for the future of Flix, as described in D5.7 (M30) it opens up completely novel use cases and is entirely extensible. Developing it from the monolithic application into a modern, RESTful API-based client/server application. Our goal was to integrate this work directly into the Flix application. This is now released providing benefits to our existing customers by leveraging this new technology. Flix has also obtained new customers, partly due to the improvements the new architecture has brought to the software.

3.2 Terminology

Term	Description
Asset	An *asset* is an entity which can be uniquely identified in the storage system. An asset does not directly represent a 'file', 'image', or blob of data. An *asset* can and usually will have a relationship with one or many underlying media objects. Underlying media objects may be different representations of a single concept which collectively or individually make up the asset.
Media Object	A *media object*, is a direct representation of a file on disk, whether its video, audio, an image, or anything else. The media object retains metadata on the file, including but not limited to, size, extension, file type, filename, and any other relevant metadata. A *media object* may be related to one or many *assets*.

Location	Locations are physical locations in which a Media Object may reside. The locations are usually a unique identifier to a particular *Server* in the system.
Server	A server is an instance of the Flix Server executable. It can be assumed that the relationship between the running software, and the hardware is a single concept. The server hardware, and the server application are one.
Region	A *region* is a logical grouping of *Servers*. This is simply a way to group *servers* into smaller clusters. The most common use-case for this grouping would be the segregate *servers* by their geographical location, hence the name *region*.
Job	A job describes a desired block of compute which the Flix Server is required to perform. Jobs are created in order for Flix Server to manage and control the required load the server needs to support. A job can describe a single task, or multiple chained tasks. An example of a job would be to store some data to disk, or send an email.

4 DIGITAL ASSET STORE DESIGN RECAP

As described in D7.1, the architecture of the asset store is designed in such a way that it meets the criteria defined by our previous understanding of the Flix customers, and also from discussions with industry members who experience the same problems and challenges of which this asset store design means to resolve.

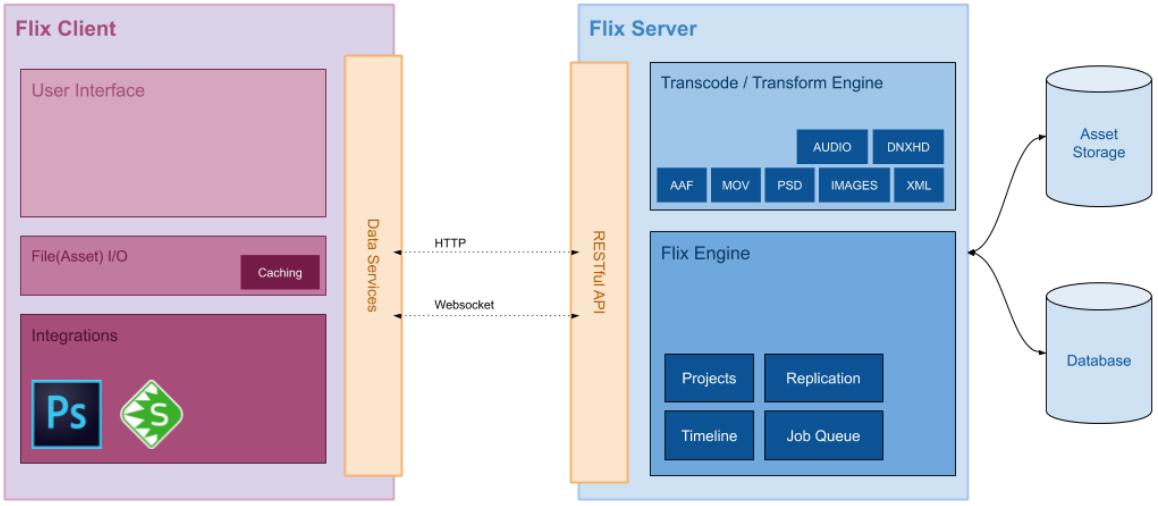


Diagram shows the high level architecture of the Flix ecosystem. Our focus of this document is the Flix Server (in blue).

The main features of the asset store are:
RESTful HTTP API to standardise communication with the Flix server and ensure easier integration with third party software. Through the RESTful HTTP we have enabled:

- Security through authentication
- Assets API
- Media Objects API
- Transcode API
- Plugins API
- Servers API

4.1 Task Scheduling System

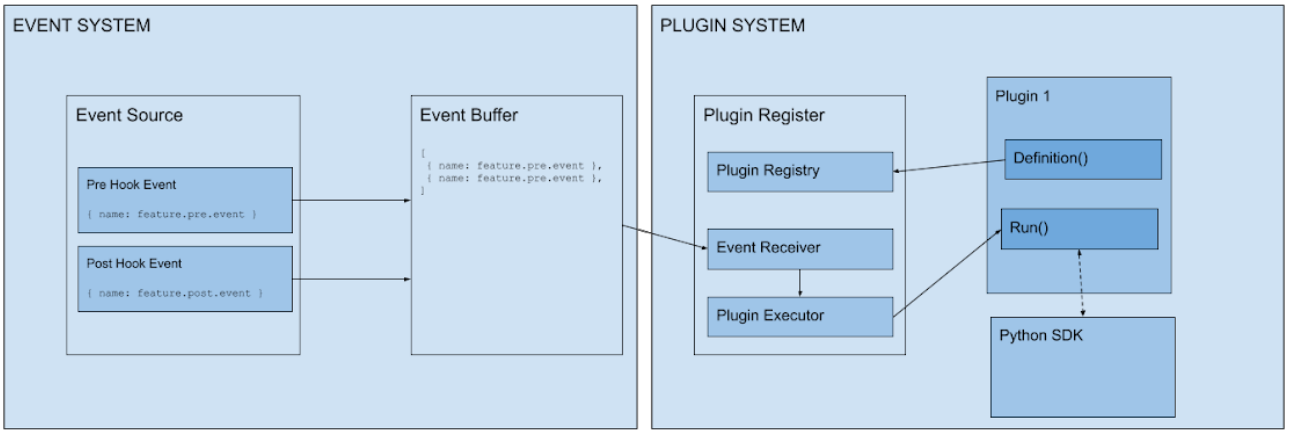
The asset store features a task scheduler system which is a concept for running, and managing arbitrary *Jobs* on the Flix *servers*. In order for the server to maintain usability and availability, it

cannot be possible that at any stage it was to become unresponsive due to high loads of compute tasks. It is also necessary that Jobs may be 'chained' together in order to curate new outcomes.

4.2 The Plugin System

The plugin system is designed so that it can be modified and maintained whilst still running to avoid costly studio shut downs.

The plugin system is designed in two parts. As pictured below in the diagram. The event system will trigger events from within actions performed within the Flix Server during runtime. For example on the completion of an asset being stored, an event will be fired. These events are used within Flix, but also can be used to trigger certain plugins to be executed. When a plugin is created, it can be designed to 'hook' into specific events.



4.3 Redundancy & High Availability

The architecture supports the adding of multiple servers in order to balance storage and compute load. Servers can be added to the cluster to improve performance and increase availability.

4.4 Tagging System

The design includes a customizable tagging system for searching and dynamic lineups of assets. This allows assets to be tagged with simple tags, colours etc. This will enable the basis of the search functionality to filter assets by tags.

4.5 File Transfer Mechanism

Flix's chunking mechanism allows for transfers to be fault-tolerant and interruptible. Asset transfers are horizontally scalable, so that multiple assets can be uploaded simultaneously to any number of Flix Servers. This offers better performance, and also redundancy.

5 USD

USD is an open-source library from Pixar which aims to provide a unified way of capturing 3D scene data, such as geometry, meshes, cameras, lights, materials, animations, skeleton rigs, volumes and more. It's primary use at the moment is to provide a clear, well-defined format for passing 3D scene data through various stages of the visual effects and animation pipelines. Importantly the same USD data can be passed through the entire production, from storyboard & pre-viz, through post-production and delivery, having the relevant assets and complexity layered on as it moves through the production pipeline. Although relatively nascent, USD is being quickly adopted by visual effects and animation studios around the world.

USD can be used at any stage of the production pipeline, often as early as pre-viz, which can lead into on-set virtual production pipelines. The challenge is then, given an asset library of USD files, or even just a single USD file for your entire production, how do you search, access & visualise its contents in a simple way? The approach we're taking here as a first step is to transcode the USD file into a more immediately accessible format: an image.

Flix is used at the beginning of the pipeline, and already has a view onto the production pipeline through it's Show, Sequence & Revision workflow. From a USD point of view, this gives an access point to begin examining the pipeline. In its simplest use case, imagine a 3D artist mocking up a previz storyboard in a 3D layout and modelling tool such as Maya. The important things to block out at this stage are the various beats of the story, defined by character poses & expressions, camera positions and properties, as well as any location setting. This could be achieved by setting a handful of keyframes for the above scene, one keyframe per panel. The scene is then saved out to the pipeline compliant USD file. When it's time to review, the artist can import the USD file into Flix. The plug-in system then invokes the USD import plug-in, where for each keyframe the panel is rendered. The render happens on the server using Hydra, which is an open specification to allow arbitrary renderers to take a USD stage and deliver pixels. Once rendered, it's easy for the storyboard reviewer to discuss changes to timings and the story in the traditional way. Updates can then be made in 3D layout, and with the rendering taken care of, the round-tripping between update and review is drastically reduced.

6 LIGHT FIELDS

Saarland has a method of 'compressing' a Lightfield into a H265 video stream which greatly reduces the size on disk of the image data, yet still provides some accessibility to the spatial data within the video content. This can be achieved by filtering frames from the video stream to selectively choose which camera from the lightfield array you wish to view. Resulting in the ability to view the video from any desired angle within the view field of the light field array.

These video format files containing the Light field data can be stored and retrieved using Flix asset storage mechanism without requirement for Flix to handle them bespokely. Flix's storage mechanism was designed to be type agnostic, so these Lightfield video formats would be no different.

For these Light Field videos to be useful in the context of the Flix Client user interface, these files would not produce the desired result if they were imported into Flix 'out of the box'. The default behaviour of Flix to import any video format would be to produce a single panel in the sequence which would be 'animated' and would contain 1 frame for each frame in the video content provided. In relation to the Lightfield data this would result in an animated panel displaying every frame and therefore producing a nauseating representation. This however can be remedied by using Flix's plugin system to transform the video content as it is imported into Flix to produce a result more desired for the given use case.

This plugin would iterate through each viewing angle of the Light Field data and render out a PNG for each frame and store these into Flix as media objects. Each of these could then be collated into a new sub asset for each viewing angle, producing an animated panel for each node in the LightField array. This would then give Flix the capability to view these video formats in its panel browser.

7 DEMONSTRATION SUMMARY

7.1 Asset Store Demo App

We have built a small demo application using Python and PyQT to interrogate the asset store API and retrieve the list of assets within a Show. Each asset can then be retrieved from the asset store by double clicking on it. The above screenshot shows the UI, which you can login to the Asset Store and see the list of assets.

7.2 Asset Store High Availability Demo

As mentioned previously, Flix's high-availability features allows for server nodes to be ephemeral and come and go throughout the lifetime of the application, this can be for a number of reasons from an operational perspective. Flix Server will maintain data integrity during server node changes to ensure 100% uptime for users where possible.

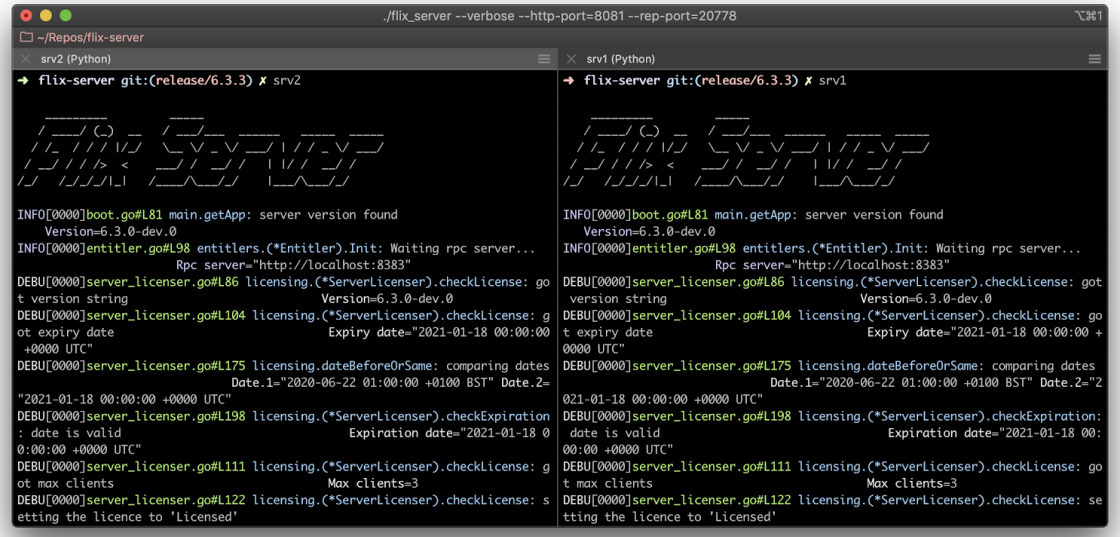


Image shows two instances of Flix Server running. SRV1 and SRV2.

Multiple Flix server instances running can handle higher load across multiple machines. Flix can recover and maintain stability if nodes are removed from the cluster.

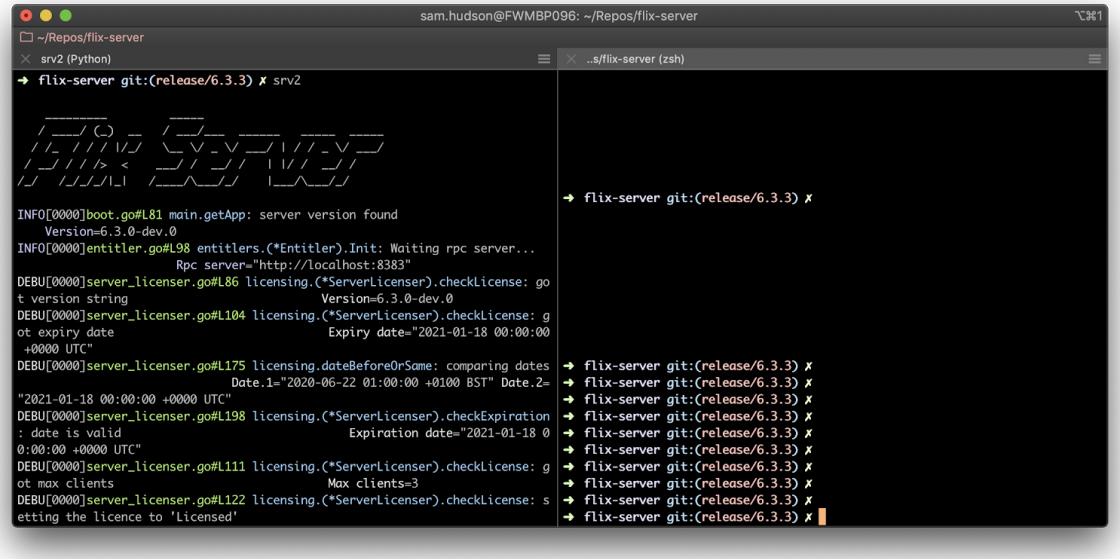


Image shows one of the servers stopped, and only one remaining running

Flix UI will maintain data integrity regardless of backend server description as long as at least one server remains operational.

Image shows Flix still loading data regardless of server loss

8 Conclusion

The separation of the front and back end of Flix to design a backend asset store server application as documented in this report has been successful. This design is performing well in practice, under expected load, in FO's commercial offering of Flix. Feedback from customers is proving there are advantages over the previous architecture of Flix 5. Foundry has 10+ customers using the new Flix iteration, storing hundreds of thousands of assets in their respective databases. We also have seen customers using Flix in the cloud, which was not previously possible. Showing the new architecture has brought new possibilities for Studios. During the COVID-19 pandemic we have seen a expedited move from customer to use Flix 6 over 5 because of the benefits it brings to home-working. Customers have successfully begun using Flix 6 with remote workers collaborating on storyboards.

The design is in a good position to meet the requirements of the SAUCE project. The underlying improvements have been released to customers in Flix, although customers are still very much using it in a storyboarding capacity and not to its full potential. SAUCE validation, with a combination of the asset store, plugin framework prototyped in D5.6 and geolocation extension prototyped in D7.3 will take place in D8.4 and D8.5 with the experimental productions.

FO still needs to spend more time on fully integration with the search and transformation frameworks from DNEG. Supporting offline assets which may be stored outside of Flix Server, this would be highlighted by Lightfield assets.

9 References

Flix API Reference: http://docs.flix-dev.com/

10 Acronyms and abbreviations

FO: Foundry
DNEG: DNEG (Formerly Double Negative)
USAAR: University of Saarland Informatics Campus
UPF: Universitat Pompeu Fabra
IK: IKinema
FA: Filmakademie
TCD: Trinity College Dublin
DRZ: Disney Research Zurich
BUT: Brno University of Technology