

Research and POC of cloud image processing with Adobe I/O and aem as. a Cloud Service

Patrick Löw
Supervisor: Peter Lipp

October 2, 2021

Contents

1	The Company	3
2	My subject area	3
2.1	Task: Research and POC for AEMaaCS	4
3	Basics for image compression and transmission	4
3.1	Image transmission	4
3.2	Image compression	5
3.3	HTLM5 Image Element	6
4	Analysis of the existing "Dynamic Asset" solution	7
4.1	Features	7
4.2	AEM architecture	8
4.3	Conclusion	9
5	Analysis of the architecture of AEMaaCS	9
6	Implementation	13
6.1	Creating a rendition	13
6.1.1	Integration into AEMaaCS	16
6.2	Display the appropriate rendition	16
7	Presentation	17
8	Additional research	18
8.1	Bypassing Processing Profiles	18
8.2	Only create used renditions	18
8.3	Concept idea	20
9	Summary	22
	References	24

1 The Company

The Company ecx.io was founded in 1995 by Christoph Mause and was acquired by IBM in 2016 [1]. Since then it has been a Company subsidiary of IBM IX, the software division of IBM. ecx.io has 8 locations with around 440 employees in Europe and India [2]. The Company's customers include Eurowings, Cyberport and Allianz, and others [3]. Among other things, ecx.io has specialised in so-called Content Management Systems (CMS) or more precisely web content management systems. These help ecx.io's customers to manage their assets [images, texts, videos, etc.] for their websites. In addition, content can be composed for the user of the website specifically for his or her interests. One of ecx.io's biggest tasks is to adapt the systems to the customer's workflow and create interfaces to existing systems.

Content management systems used also include: Adobe Experience Manager, SAP Customer Experience and Sitecore.

2 My subject area

For a good user experience, the time it takes for a website to load on the device is an important factor. Because as the loading time increases, more and more users decide to cancel their visit to the page. As a survey by Google shows, the chance that a mobile phone user will stop loading a page increases by 32%. The loading of the page increases by 32 %, if the page has a loading time of three seconds instead of one [4]. For example, a person who stops loading the page cannot buy anything in an online shop. Conversely, for an online retailer, this means unearned revenue.

Loading pictures and videos takes among the longest. Various techniques are used to shorten the loading time. For this purpose, a new version of the image, also called a rendition, is created that reduces the loading time of the image.

Automatic generation of a rendition in a CMS system

So that the creation of the renditions does not have to be done by the author of the image, in the best case, the CMS system takes over this task automatically. But not every system is capable of this routine. The CMS from Adobe used in the company (Adobe Experience Manager 6.5) was extended by a custom component for this purpose. Meanwhile Adobe has released another version of the CMS. The programme called Adobe Experience Manager as a Cloud Service (AEMaaS) is based on a different architecture. The component for

the old system cannot be ported to the new system.

2.1 Task: Research and POC for AEMaaCS

The task was to look at the features of the existing solution and research whether these could be realised in the cloud version. In addition, a proof of concept with code and documentation is required.

3 Basics for image compression and transmission

3.1 Image transmission

For a fast building website it is important to keep the loading times of the images other resources as short as possible.

There are several factors to consider when loading resources, especially images.

$$\textit{Loading time} = \textit{Server response time} + \textit{Round trip time} + \textit{Transmission time}$$

Server response time

The response time of the server determines how long the server needs to give a suitable response to a request. For example, if a web server receives a request to deliver an image, the server needs a certain amount of time to search for the image in its storage in order to send it. This time can be influenced by many factors. Where is the image stored? How fast is the server's memory to load the image. How busy is the server. With many requests e.g. the requests that were previously made are processed first.

In modern web architectures, cache servers are placed in front of web servers as a proxy. These are usually dedicated servers that host the most frequently used static content (content that does not have to be generated for a request, but is already available) are stored in the Ram. If a static content such as an image on the homepage of a company is requested, the cache server can send a response directly from its Ram. This takes the load off the web server and eliminates time-consuming access to the hard disk.

In order for the content to be cached by the web server, it is necessary to signal to the cache server that this is a static content. This information is given in the response in the http header and is standardised in the RFC 7234 [5] among others.

Round trip time (rtt)

The round trip time is the time required for a signal to travel from the client to the server and back again. This time is made up of the distance and the speed, a signal can travel through a medium (e.g. copper or fibre optic cable).

This time can only be influenced by the geographical location of the web servers and the cache servers.

Transmission time

The transmission time is composed as follows.

$$\text{Transmission time} = \text{Size} / \text{Bit rate}$$

The bit rate is usually determined by the user's network connection. This can often be limited, especially when using a mobile device.

The size of an image is crucial here. The smaller the picture, the shorter the transmission time of the image.

3.2 Image compression

In order to keep the memory size of an image as small as possible, several techniques are used.

Resolution

One way to reduce the size of an image is to reduce the resolution of the image. Since images have to scale in height as well as width to maintain their aspect ratio, an uncompressed image that is x times larger consumes x times larger uncompressed image x^2 of storage space. It is the same in the opposite direction. An uncompressed image half the size consumes only a quarter of the storage space. Low-resolution images, however, look very blurry on high-resolution screens. On the other hand, there is no visual advantage to displaying a high-resolution image on a low-resolution mobile phone screen.

To solve this problem, the designer creates a very high-resolution image, which is converted to the correct size depending on the display device.

Image formats

Since an uncompressed image consumes a lot of space, the transmission will take too long. To reduce the storage space of the images, different image formats have been developed for compression. Each has its advantages and

disadvantages. The format Portable Network Graphics (PNG) is, for example, one of the oldest formats. It is supported almost everywhere and supports a transparent background [6]. But it has a comparatively large storage space requirement. The Joint Photographic Experts Group format (JPEG) which is supported about as well. It is comparatively small, but in its original specification it cannot display a transparent background. This was only added in the JPEG XT extension and is not supported by all browsers [7]. Webp is a very modern image format with a lot of features and a very low storage requirement [8] but is not yet supported by all clients like Safari [9].

Depending on the application, it is advisable to select a suitable format and then deliver it. It is important that the client supports the format and that formats with lower memory consumption are preferred.

Aspect ratio

On websites, the aspect ratio of images also plays an important role. A company's website should look just as good on an extra-wide computer monitor as it does on a mobile phone screen in portrait mode. To achieve this, images on the mobile phone are often cut off on the right and left. These invisible areas are nevertheless transmitted in the image data.

The page load can be accelerated by transferring the images directly to the client in the appropriate aspect ratio.

Progressive image loading

Another way to speed up the page load is to change the way the images are loaded. For example, there is an option in the Jpeg standard that does not save the images line by line [10]. It is saved in such a way that the image can already be displayed with low resolution during transmission. With increasing data transmission, the image can be displayed in higher and higher resolutions. This does not shorten the loading time of the complete page, but it does shorten the time it takes for the user to interact with the page.

3.3 HTML5 Image Element

Another problem with the delivery of the images is which image should be delivered to the browser in which size. In the HTML5 standard, which was published by the W3C in 2008 and has been further developed by the WHATWG since then, there is the so-called image element. The URL of the image and the image description are stored in this element. With HTML5, not only one URL but a whole set of URLs for e.g. images in different resolutions

(renditions) can be stored. Each of these URLs can be supplemented with further parameters. These media features can describe the minimum as well as the maximum appropriate width or height of the screen that is suitable for the rendition. There are also other parameters that relate to the orientation of the screen [11].

```
<picture>
  <source media="(min-width: 1440px)" srcset="pic-1840.jpg">
  <source media="(min-width: 680px) and (orientation:portrait)"
    srcset="pic-960-quad.jpg">
  <source media="(min-width: 680px)" srcset="html-pic-920.jpg">
  
</picture>
```

The browser, which has information about its orientation and viewport dimensions, loads only the appropriate image and displays it. It can also download other images that are smaller to save bandwidth. This depends on the configuration of the browser. Since the image element in this form is also compatible with the first HTML5 version, it can be used without any problems.

4 Analysis of the existing "Dynamic Asset" solution

The Dynamic Asset component can offer all possible resolutions and other properties to the browser in an image element. If one of these renditions is requested, it is loaded from the cache or generated dynamically. Which rendition is offered is defined in a configuration file. The Dynamic Asset component for AEM 6.5 is written in java and supports the following features

4.1 Features

Crop

The component can crop images in size and aspect ratio. The sections can be defined beforehand by the author. In addition, the component still has a smartcrop mode. If no crop has been specified by the author for a given aspect ratio, the software tries to find the best one. Which aspect ratios should be delivered can be configured in advance.

Scale

Any resolution can be dynamically generated from an image. Which exactly is defined beforehand in a configuration file. The browser can then choose which resolution to load.

Text transformer

Can place text with a specific font, position and size on the image.

Rotation

Can rotate an image by any angle.

Grayscale

Converts the image to black and white.

Formats

The component is based on the standard image processing tools of AEM 6.5 and supports all their formats. The most important ones here include webp, jpg and png.

4.2 AEM architecture

The component is designed for the version of AEM 6.5. AEM 6.5 consists of several instances. One Author and mostly 2 Publisher instances. On the Author instance, new content is created or uploaded by the author. As soon as this content is released, it is uploaded to the Publisher instances. The users only access the Publisher instances. Each instance has a binary storage and a node storage.



The Node Storage, also called Structured Content, contains information about the complete structure of the web pages. It is structured in a node structure and refers to the corresponding assets in the binary storage. The node storage is structured according to the Apache Jackrabbit Content Repository for Java (JSR283[12])[13].

If a browser now requests a rendition of an image that is not held by one of the cache servers, the Publisher instance accesses the image in the binary storage, processes it according to the browser requirements and delivers it.

4.3 Conclusion

This solution can generate an image as small as possible for each device that requests web pages with images. This reduces the transmission time to a minimum. The round trip time cannot be influenced by the solution because it depends on the server. Since the generated images are cached, the average response time of the server will be low. Only new images that have not yet been cached increase the response time of the server. The storage space on the server is kept as small as possible because only the original is stored. However, the CPU consumption increases because the rendition has to be reprocessed for every requested rendition that is not in the server's cache.

5 Analysis of the architecture of AEMaaCS

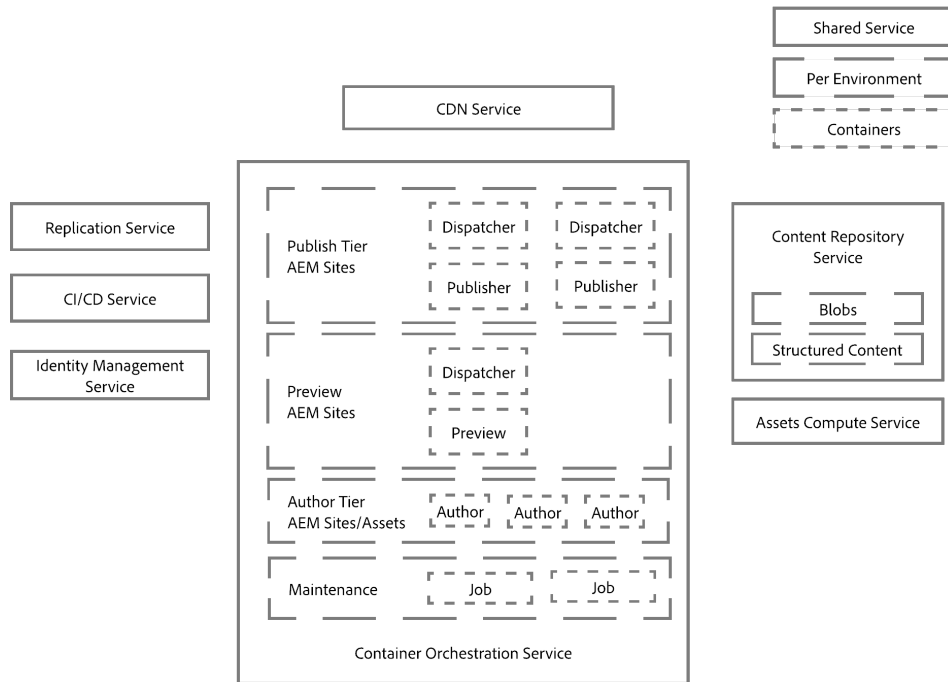


Figure 1: AEMaaCS Architecture By Adobe [14]

The new CMS system in the cloud was changed in the software architecture. Attention was paid to ensuring that the system is highly scalable. The system should be able to withstand the load of a very large corporation but at the

same time not consume too many resources with light loads. For this reason, the system was split into several components. All of which are running in different instances. These can be switched on and off depending on the workload. Many requests can now be handled by connecting additional instances of a component without overloading the system. At the same time, updates can be installed during operation without having to accept downtimes[14].

Storage

Since the instances are ramped up and down very rapidly, there is no time to save data. The data storage is taken over by a service for this reason. The Content Repository Service contains a Global Blob Storage (Binary Storage) and 2 Two Node Storages (one each for Publisher and Author). A service in the content repository service ensures that the author and publisher instance are all at the same state. This way, when publishing the pages, only the data of the node storage has to be copied, which shortens the time for publishing[14]. The Microsoft Azur Cloud is used as the storage provider[15].

To reduce the load on the publisher and author instances and to increase the upload speed Adobe uses a technology called Oake from Apache[16]. The upload is no longer routed through the web server, but instead the web server, here the author instance, generates a presinged URL that specifies a location in the blob storage as well as the maximum size that may be uploaded. This is sent to the client, which writes its file directly to the cloud storage. When delivering, the whole thing works similarly - a presinged url is created through which a client can download the file[17].

These URLs are permanent and can be cached as well.

Author and Publisher

The author and the publisher instances were designed in such a way that they can be moved up and down very quickly to adapt to the current conditions. They only contain the code, the complete data is read from the content repository. Similar stuff such as authorisation was also taken over by a service outside the instance. The processing of binary data is also handled by a computing service outside the instance. Author and Publisher have functions to address this service directly. Through these measures, the instances are reduced in load and can be operated with much less resources.

Adobe I/O Runtime

In order not to burden the instances of authors and publishers, a serverless computing platform service was added for processing data such as generating image renditions or communicating with other systems. The so called Adobe I/O Runtime is based on apache's OpenWisk platform[18]. On the platform, functions can be executed and written in different languages. A function can be triggered by an http request as well as through other functions. However, intervals can also be set in which these functions are triggered.

On the platform, Adobe offers a framework called Firefly[19]. This is written in Nodejs and facilitates, for example, interaction with the Adobe Product APIs, adobe's Authentication Managemant and the Binary Storage of AEMaaCS. There is also a library in the Firefly framework that facilitates image manipulation the Asset Compute Service.

Adobe Asset Compute Service

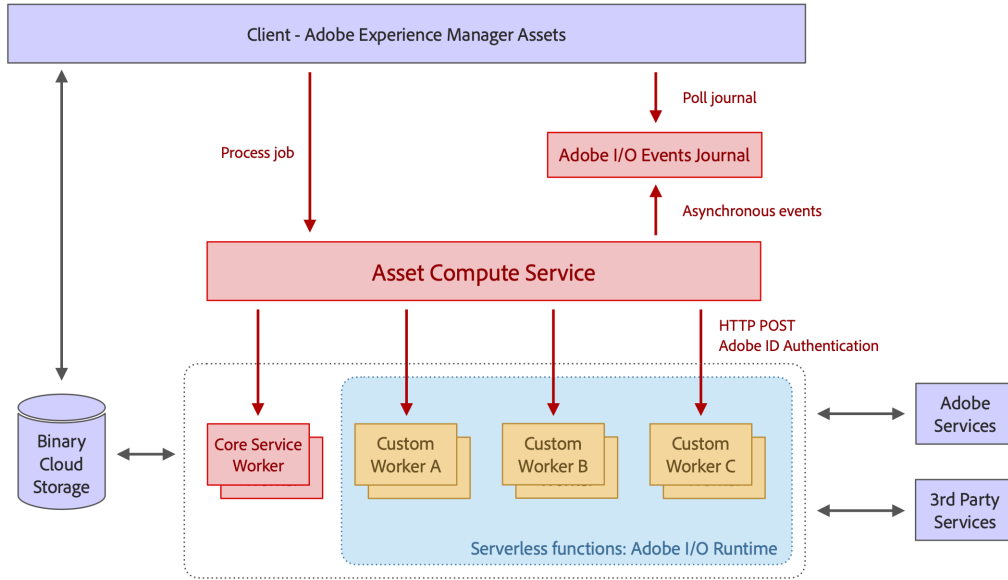


Figure 2: Architecture of Asset Compute Service By Adobe [20]

The Adobe Asset Compute Service is using functions of the Firefly framework supporting image manipulation. This service can be accessed directly by the AEMaaCS and abstracts processes such as authentication, event handling, upload and download of images. For the manipulation of the images, prefabricated functions can be used or own functions can be written. The so-called Worker [20].

Asset Compute Service cycle

Once a designer has finished uploading an image to an author instance. The author can start an http post request with a json file to the Asset Compute Service. This passes the file on to the custom workers see Figure 3. The following necessary parameters are contained in the json file.

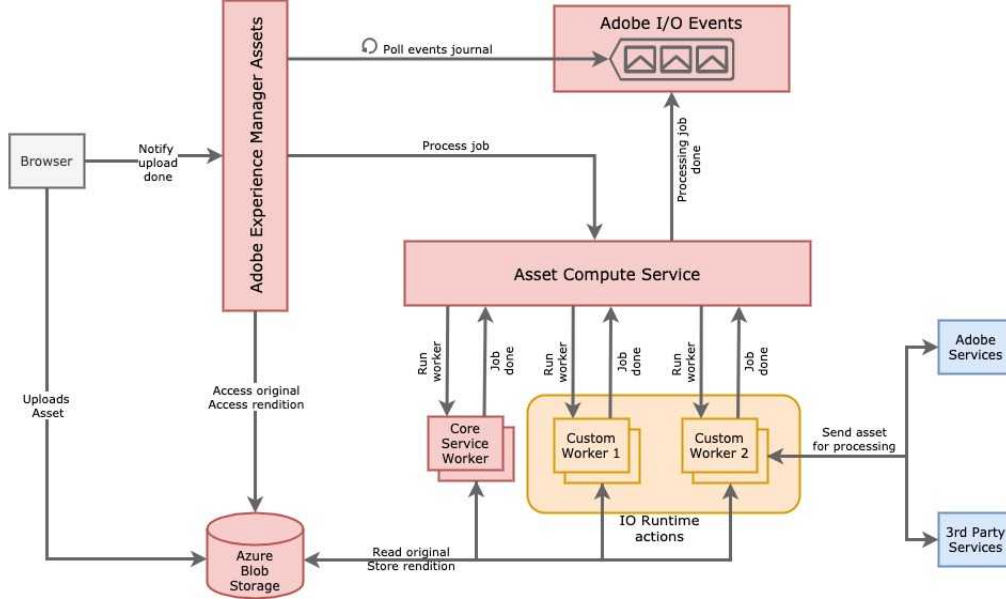


Figure 3: Cycle of Asset Compute Service By wttech.blog [21]

Presinged URLs To avoid load on the author instances, only references to the resources are passed to the Asset Compute Service in the form of presinged URLs. At the same time, the url contains a token that serves as authorisation and can be only valid for a limited period of time. A URL with an unlimited time valid token is used for download, it is the same token that a browser calls when it loads an image on a web page. To save the rendition, a presinged URL is also transmitted in which the service can save the new rendition, but this is only valid for a limited time and a maximum file size[22].

To improve the load and make the system more robust, a file is split into several parts during transfer[16].

Custom parameters In addition to the URL, the author can also specify other parameters. The developer of the Custom Asset Compute Worker can define these as needed.

The Asset Compute Service now receives this request, checks the authorisation and passes it on to a worker. These workers are stateless instances that can be ramped up and down as needed depending on the load. This worker must be written by the user. It downloads the image, manipulates it according to the given parameters and writes the rendition back into the BlobStorage. Once the image has been uploaded, the service returns to the author that a new image has been created. The author registers the URL in the node storage of the author instances[22].

6 Implementation

After researching the technology, which is also new to the company, the next step is to implement the POC. The implementation of the POC consists of two parts. The first part is about creating a rendition from the original. In the second part, display the appropriate rendition in the client's browser.

6.1 Creating a rendition

To create a rendition, an image, a Firefly function and a trigger is required. A test image is quickly found. A trigger is also not difficult at first glance. For the development of a Firefly app with the Asset Compute Service there is a command line tool with which you can test functions on a small scale[23].

Developing the Firefly app

The development of Firefly apps is supported by command line tools[24] [25]. Different templates can be generated for specific use cases. A project template is available for creating an Asset Compute Service with Custom Worker[26]. This generates code that abstracts tasks such as upload and download.

A big limitation is the programming language, so only libraries can be used that run 100 percent under Javascript or nodejs and have no dependencies written in other programming languages. The well-known libraries that are otherwise used, such as Imagemagic or Sharp, cannot be used. There is a Photoshop service from Adobe that can be accessed from the Custom Worker. However, this is still in beta and is not intended for production use[27]. The company imgix.com also offers a service that can be accessed from a custom worker. The disadvantage of this solution is that every customer of exc.io who wants to run this solution has to have an account from this company which also has to be used to settle further invoices[21]. The solution was a library called Jimp. It is executable in Nodejs without dependencies to other

languages and offers a basic set of methods for image processing[28]. The result is a custom worker that can be used with the following parameters.

Operation	default	min	max	Description
w	image.width	1	+inf	sets image width if not specified size is calculated from the image height
h	image.height	1	+inf	sets image height, if not specified size is calculated from the image width
crop	{"x": 0, "y": 0, "width": image.width, "height": image.height}	{"x": 0, "y": 0, "width": 1, "height": 1}	{"x": +inf, "y": +inf, "width": +inf, "height": +inf}	Wenn the image is cropped to the defined area. The top-left rule applies
crop_percent	{"x": 0, "y": 0, "width": 100, "height": 100}	{"x": 0, "y": 0, "width": > 0, "height": > 0}	{"x": < 100, "y": < 100, "width": < 100, "height": < 100}	The same as the crop function but here percentages are given. This function accepts decimal numbers.
aspect	{"width": image.width, "height": image.height, ["mode" = "cover"]}	{"width": image.width, "height": image.height, ["mode" = "cover"]}	{"width": +inf, "height": +inf [mode]}	Defines how the image should be cropped.
grayscale	false	false	true	Converts to black and white.
circle	false	false	true	A circular Image section is taken.
remove-alpha	false	false	true	Remove the alphchannel.
quality	100	0	100	Quality for the saved image.

Table 1: Options from a custom Worker

Particular problems occurred when creating a black and white image, as Jimp either uses only the green channel for the black and white image, as do many cheaper digital cameras, or outputs a rendition afterwards in

which the greyscales were calculated correctly but the rendition still has 3 colour channels. This means that the rendition uses up an unnecessarily large amount of storage space, which is noticeable in the loading times of websites. Just as important was the handling of the alpha channel in images this can also increase the storage requirements of an image.

Finally, the sequence of operations is very important. On the one hand, it can influence the computing load. The fewer pixels that have to be calculated, the better. On the other hand, this can influence the quality of the images.

The best example is crop and resize. For example, when resizing is done before cropping. With this sequence, not only is the quality of the image extremely affected, but the desired line resolution is also not achieved.

Create a Rendition with the following parameters.
 crop_percent = {„x“: 0, „y“: 0, „width“: 50 „height“: 100}
 width = 960

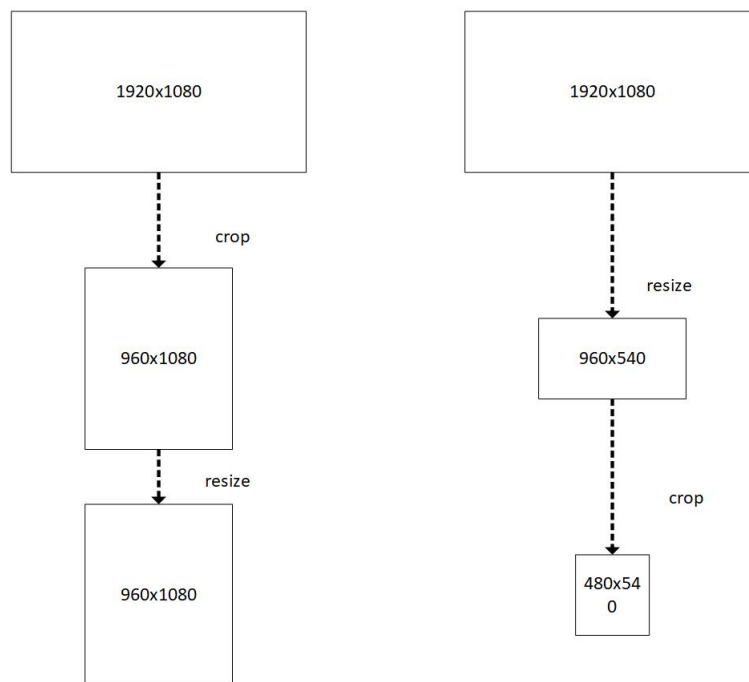


Figure 4: Example of image operations in different sequence

6.1.1 Integration into AEMaaCS

For the integration process, it is possible to create Processing Profiles in AEMaaCS in which the appropriate Asset Compute Worker is addressed. This can then be passed further parameters which the Custom Worker interprets. These profiles can then be assigned to individual folders in AEMaaCS. When an image is uploaded to the folder, the appropriate profile is activated[29].

Processing Profile

Cancel Save

Profile

Name
Sabotage

Image (0) Video (0) Custom (1)

Processing Services

Create Metadata Rendition
☐

Service

Rendition Name * Commando Extension * jpg

Endpoint (URL - Must be a Firefly app from the same organization) *
https://34527-931lvorywhippet-test.adobeioruntime.net/api/v1/web/931lvoryWhippet-0.0.2/worker

Service Parameters

circle	true
--------	------

Add Parameter

Mime Types

Includes
Enter Include MIME type
image/*

Excludes
Enter Exclude MIME type
image/gif

Add New

Figure 5: Creation of a Processing Profile

There are several problems with the integration via the Processing Profiles. The biggest problem is that only one Processing Profile can be assigned per folder. Another problem is that Processing Profiles can only use static parameters. Thus, in this configuration, it is not possible to set a user-defined crop for each image in a folder.

6.2 Display the appropriate rendition

In AEMaaCS, websites are built from several components. Components are for example texts, headlines, share functions, buttons but also pictures. A

component consists of the front end and, if necessary, an associated back end. The frontend is designed with Adobe's own script language HTL. This will later be converted into HTML[30]. If dynamic data is to be transferred to the frontend, a model or servlet can be created in the AEMaaCS backend. The frontend can access parameterless methods of the model and obtain data. The model has no parameters, but it can determine from which component it is called from what page and it can access the node structure in the content repository[31]. In addition to an image, the corresponding renditions can also be called up, as these can be found in the same node directory. From this information, a component is created that contains an image element with several renditions.



Figure 6: Image Element from an AEMaaCS

Since the browser accessing the web page gets an HTML5 image element with multiple renditions, it loads the image closest to its viewport size. If the resolution changes, another image is loaded. For the POC, a static set of image resolutions was entered in the AEMaaCS model. This disadvantage will be changed in the following versions.

7 Presentation

The interim results were to be presented to a wider circle of developers in a presentation. As it is a relatively new topic, there was a large circle of about 40-50 developers interested in the topic. As developers from all locations of the company were represented, the presentation had to be given in English. After the very good feedback, it was decided to further develop the POC into a component for use at the customer.

The lack of a dynamic crop function, which is not possible via the static Processing Profiles, was noted as a problem. For the rest of the time, I was asked to look for solutions for dynamic cropping and assigning profiles to specific folders, in addition to further developing the Custom Compute Worker.

8 Additional research

In order to turn the POC into a production component, some problems still need to be solved.

8.1 Bypassing Processing Profiles

The biggest problem with the current POC solution are the Processing Profiles. These services can only be bound to folders. However, since a folder contains images of a project, these can be used for different purposes. The approach of the Processing Profile is not suitable. Another problem is that only static parameters can be given to a profile. In order not to crop each image around the centre but around the point of interest, a different image section must be taken for each image. This results in different parameters. There is a possibility to insert software like smartcrop [32] in the Asset Compute Worker that selects a suitable image crop. This works well in most cases, but not everywhere. There are cases, for example, where the company's customers have specified that the logo of the customer's company cannot be cut off in any picture. Therefore, it has to be possible to determine the image section manually.

Fortunately, the Asset Compute Service has a well-documented API called `http Asset Compute API`[22]. At the same time there is an Openscource Javascript library from Adobe which can address this API and can be passed parameters dynamically[33]. For example, the Processing Profile could be read out and then combined with the data that an author has stored for the crop of a particular image. When a crop is changed, the image is regenerated accordingly.

8.2 Only create used renditions

In order to cover all cases of application with different resolutions, aspect ratios and image formats, a large number of renditions shall be generated. Since these images have to be generated in advance due to the architecture, a lot of memory is wasted on renditions that are never used. This can become

a factor of cost that cannot be ignored. For example, not all images are used in all page ratios. For example, a hero image that is typically very wide does not need to have a 1 to 1 aspect ratio. Thus, possibilities are being searched for reducing the mass of renditions only to those that are really used.

The idea here would be to only generate the renditions when they are needed. In this case, this can be regulated with templates and policies.

Template

A certain template for a website can be created in AEMaaCS. It is already filled with all the elements that are usually used. An article page, for example, always contains a title in a certain size and font, a cover image, links to download the article and the name of the author. When an article has to be written, this template is selected as the starting point.

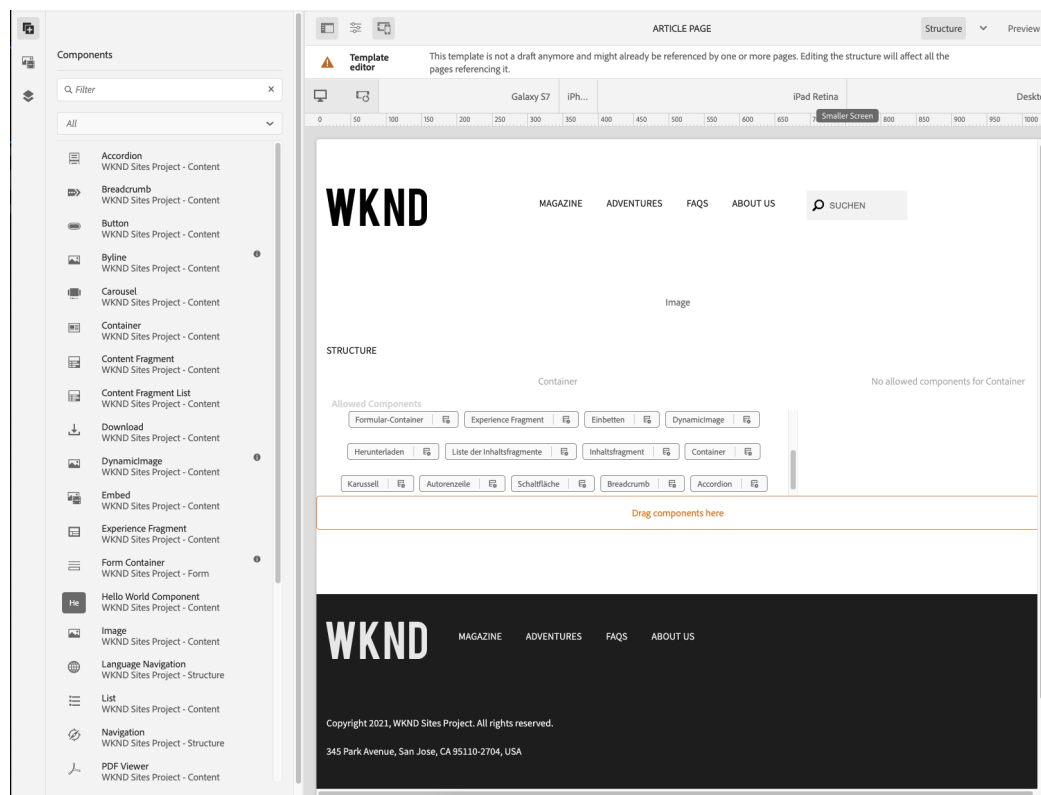
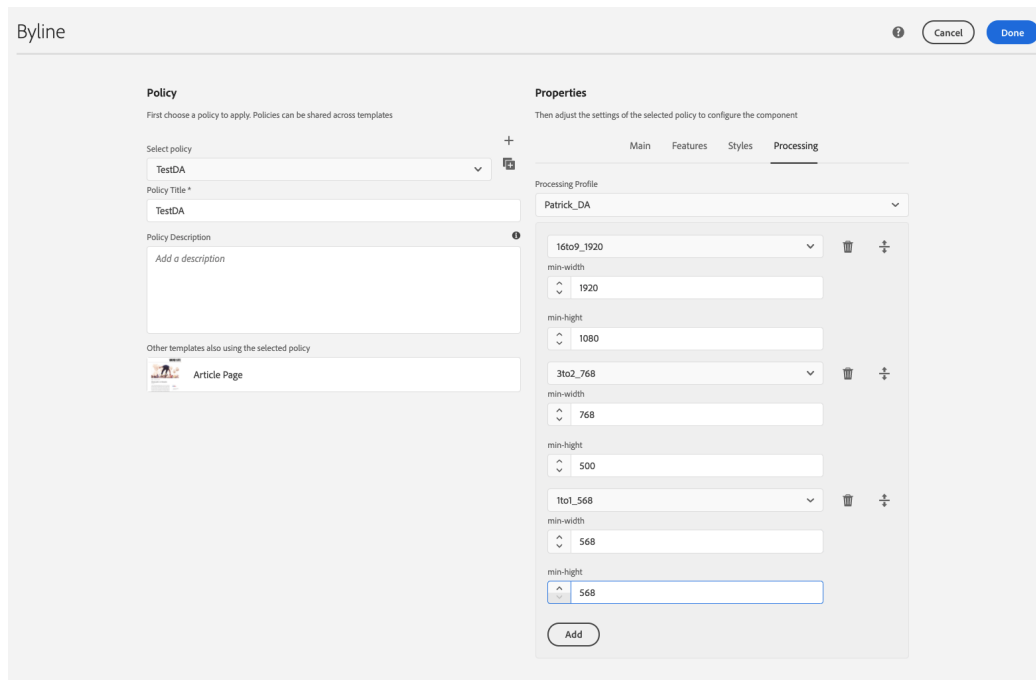


Figure 7: A template of a web page with preset header and footer.

Policies

In order to keep the websites compliant in large companies, the templates are managed with policies. A policy can be used to determine which elements must be present on a website, which may not be changed and if they must follow certain parameters. Any parameters can be used, from the size of a font to completely arbitrary parameters.



The screenshot shows the 'Byline' application interface. On the left, the 'Policy' panel is active, displaying a dropdown menu with 'TestDA' selected, a text field for 'Policy Title' containing 'TestDA', and a text area for 'Policy Description' with the placeholder 'Add a description'. Below this, a section titled 'Other templates also using the selected policy' shows a thumbnail for 'Article Page'. On the right, the 'Properties' panel is active, showing tabs for 'Main', 'Features', 'Styles', and 'Processing'. The 'Processing' tab is selected, displaying a 'Processing Profile' dropdown with 'Patrick_DA' selected. Below this, there are three rows of input fields for 'min-width' and 'min-height' for different image sizes: '16to9_1920' (min-width: 1920, min-height: 1080), '3to2_768' (min-width: 768, min-height: 500), and '1to1_568' (min-width: 568, min-height: 568). Each row has a trash icon and a swap icon. An 'Add' button is at the bottom of the 'Processing' tab.

Figure 8: A self-written input mask for an image policy

Policies for images can be stored in the templates. For example, you can specify different aspect ratios and resolutions for the hero image than for an image used in the text. The generation of the renditions is only triggered when an image is used in a component.

8.3 Concept idea

With further research, I have drawn up a rough process flow. Where not all steps have been completely checked by me for feasibility, shown here in red, or have been completely implemented, shown here in yellow. The green fields are supplied by AEMaaCS or have been implemented.

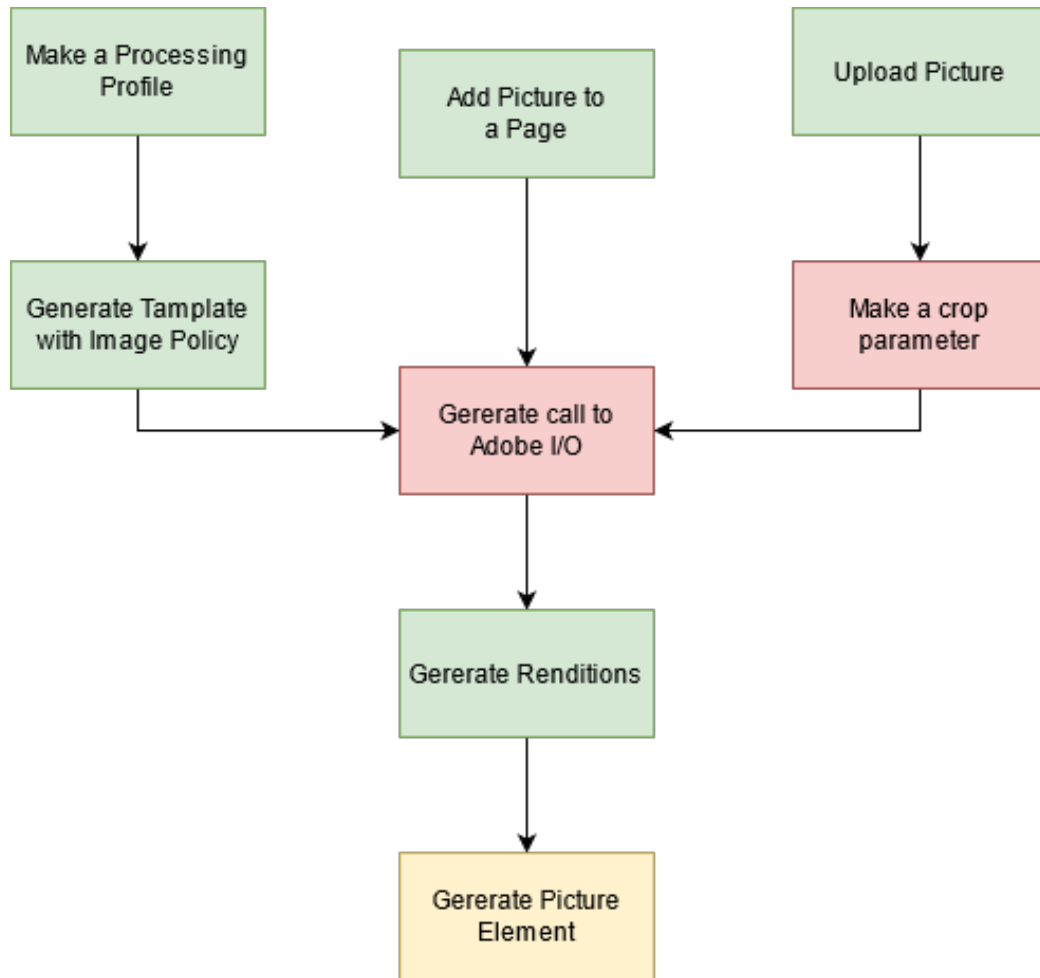


Figure 9: Process Workflow for Dynamic Images

Make a Processing Profile

Creating a Processing Profile is a function provided by AEMaaCS. It allows the setting of different renditions in different sizes and other parameters. These are stored in the node storage.

Generate template with Image Policy

Here it is possible to define different policies for certain image components in a website, which say which image should be linked with which Processing Profiles. The policy generation has a self-written component in which the Processing Profiles can be selected and enriched with further parameters for the image element.

Upload picture

Uploading an image is provided by AEMaaCS. No renditions are to be created here yet, as space is to be saved and it is still not known how the images are used.

Make a dam crop parameter

Setting a crop for a specific image is a bit more complicated, but will be transferable from the previous version without much effort.

Add picture to a page

Adding an image to a web page is also a standard function of AEMaaCS and does not require any major adjustments.

Generate call to Adobe IO

After an image has been embedded in a web page, a JSON file is generated with the help of the policy for this image, which refers to the Processing Profile, and the defined crop parameter for this image. This file contains all parameters for all renditions that are to be created. This file is then sent to the Asset Compute Service to generate the renditions.

Generate rendition

The self-written Asset Compute Worker generates the desired renditions. These are written to the Binary Storage.

Generating Picture Element

When the page with the picture is called up, a Picture Element is generated on the basis of the policy. This element contains all renditions of the policy. The browser can then request the appropriate picture.

9 Summary

Adobe has added a cloud version to its portfolio of content management systems in addition to the local variant Adobe Experience Manager (AEM) 6.5. This variant, known as Adobe Experience Manager as a Cloud Service (AEMaaCS), has seen a number of innovations in its architecture.

With the new architecture of the AEMaaCS, Adobe has eliminated some of the problems of the older AEM 6.5. In particular, scalability has been improved by splitting one large programme into several small modules. This means that any number of instances of a certain module can be started up very quickly in order to serve certain load cases, while only very few instances of another module are active. This means that the new solution (AEMaaCS) can be used with high flexibility by their customers.

Most of the components written for AEM 6.5 can still be used in AEMaaCS. But there are some areas where this is not the case. One of these areas is the processing of binary data formats, e.g. images and videos. For this processing an extra module is provided that requires pure Javascript as a programming language, the Asset Compute Service. Components in this area have to be rewritten completely as they are mostly written in Java for AEM 6.5. In addition, very few libraries for image processing are written in pure Javascript.

An important factor for a fast web page load on the user's end device is adapting the images in the web page to the end device. To do this, the images must be adjusted in image format, resolution and aspect ratio. The Asset Compute Service in AEMaaCS offers certain basic operations for processing images, but these are very limited in scope. Thus, these operations can only be applied to a folder and not to a single image. So you have to implement your own components, e.g. to be able to define an individual crop per image.

Another factor is displaying the correct picture for each individual end device, which is accomplished by a picture element. While this is relatively easy to do, it has to be implemented by someone.

The proof of concept (POC) developed here is intended to demonstrate the practicality of a system that solves the problems described above. The POC solves many of these problems but not all of them. For the problems not solved in the POC solutions were found afterwards. Based on the POC and further research, a team was formed at ecx.io to develop an operational component for the AEMaaCS.

References

- [1] Ingrid Lunden. *IBM Buys Germany's Ecx.io, Its Third Creative Services Acquisition in a Week*. URL: <https://techcrunch.com/2016/02/03/ibm-buys-germanys-ecx-io-its-third-creative-services-acquisition-in-a-week/> (visited on 06/11/2021).
- [2] ecx.io. *Who we are*. URL: <https://www.ecx.io/en/digitalagency/> (visited on 06/11/2021).
- [3] ecx.io. *Work*. URL: <https://www.ecx.io/en/work/> (visited on 06/11/2021).
- [4] Google. "Find Out How You Stack Up to New Industry Benchmarks for Mobile Page Speed". In: (), p. 3. URL: https://www.thinkwithgoogle.com/_qs/documents/57/mobile-page-speed-new-industry-benchmarks.pdf (visited on 06/11/2021).
- [5] J. Reschke R. Fielding M. Nottingham. *Hypertext Transfer Protocol (HTTP/1.1): Caching*. URL: <https://datatracker.ietf.org/doc/html/rfc7234> (visited on 07/26/2021).
- [6] T. Boutell. *PNG (Portable Network Graphics) Specification*. URL: <https://datatracker.ietf.org/doc/html/rfc2083> (visited on 07/28/2021).
- [7] ISO/IEC WG1. *Overview of JPEG XT*. URL: <https://jpeg.org/jpegxt> (visited on 07/28/2021).
- [8] Lou Quillio James Zern et al. *A new image format for the Web*. URL: <https://developers.google.com/speed/webp> (visited on 07/28/2021).
- [9] Lou Quillio. *Frequently Asked Questions*. URL: https://developers.google.com/speed/webp/faq#how_can_i_detect_browser_support_for_webp (visited on 07/28/2021).
- [10] The International Telegraph and Telephone Consultative Committee. "ITU-T T.81". In: (1905), 17f. URL: <https://www.w3.org/Graphics/JPEG/itu-t81.pdf>.
- [11] Ian Hickson Anne van Kesteren et al. *HTML Living Standard 4.8.4 Images*. URL: <https://html.spec.whatwg.org/multipage/images.html#adaptive-images> (visited on 07/28/2021).
- [12] David Nuescheler Day Software AG. *Content Repository for Java™ Technology API 2.0*. URL: <https://jcp.org/aboutJava/communityprocess/final/jsr283/index.html> (visited on 08/01/2021).

- [13] Adobe. *Introduction to the AEM Platform*. URL: <https://experienceleague.adobe.com/docs/experience-manager-65/deploying/introduction/platform.html?lang=en> (visited on 08/01/2021).
- [14] Adobe. *An Introduction to the Architecture of Adobe Experience Manager as a Cloud Service*. URL: <https://experienceleague.adobe.com/docs/experience-manager-cloud-service/core-concepts/architecture.html> (visited on 08/01/2021).
- [15] Bob Bringhurst Peter Nolan Joshua-McGee. *Data protection in Adobe Experience Platform*. URL: <https://experienceleague.adobe.com/docs/experience-platform/catalog/data-protection.html%3Flang%3Des> (visited on 08/01/2021).
- [16] Chetan Mehrotra Marcel Reutegger et al. *Oak Documentation - Direct Binary Access*. URL: <https://jackrabbit.apache.org/oak/docs/features/direct-binary-access.html> (visited on 08/01/2021).
- [17] Mark Frisbey Jung Zhang. *Adobe aem-upload*. URL: <https://github.com/adobe/aem-upload#background> (visited on 08/01/2021).
- [18] Wendy Braun Mihai Corlan et al. *Adobe I/O Runtime Overview*. URL: <https://www.adobe.io/apis/experienceplatform/runtime.html> (visited on 08/01/2021).
- [19] Valerii Naida Stephan Ringel. *What is Project Firefly*. URL: <https://www.adobe.io/project-firefly/docs/overview/> (visited on 08/01/2021).
- [20] Jamie Delbick Ashish Gupta Alexander Klimetschek. *Architecture of Asset Compute Service*. URL: <https://experienceleague.adobe.com/docs/asset-compute/using/architecture.html?lang=en> (visited on 08/02/2021).
- [21] Marcin Cieczko. *How to generate intelligent renditions with AEM as a Cloud Service*. URL: <https://wttech.blog/blog/2020/how-to-generate-intelligent-renditions-aem-cloud/> (visited on 08/02/2021).
- [22] Alexander Klimetschek Ashish Gupta et al. *Asset Compute Service HTTP API*. URL: <https://experienceleague.adobe.com/docs/asset-compute/using/api.html?lang=en> (visited on 08/03/2021).
- [23] Jamie Delbick Ashish Gupta. *Test and debug a custom application*. URL: <https://experienceleague.adobe.com/docs/asset-compute/using/extend/test-custom-application.html?lang=en> (visited on 08/03/2021).

- [24] Alexander Klimetschek Bob van Manen et al. *aio-cli-plugin-asset-compute*. URL: <https://github.com/adobe/aio-cli-plugin-asset-compute#install-as-local-devdependency> (visited on 08/03/2021).
- [25] Jesse MacFadyen Shazron Abdullah et al. *aio-cli*. URL: <https://github.com/adobe/aio-cli> (visited on 08/03/2021).
- [26] Jamie Delbick Ashish Gupta et al. *Develop a custom application*. URL: <https://experienceleague.adobe.com/docs/asset-compute/using/extend/develop-custom-application.html?lang=en> (visited on 08/03/2021).
- [27] Adobe. *Add the magic of Photoshop to your website or workflow*. URL: <https://www.adobe.io/photoshop/api/> (visited on 08/03/2021).
- [28] Oliver Moran. *JavaScript Image Manipulation Program - Jimp*. URL: <https://github.com/oliver-moran/jimp> (visited on 08/03/2021).
- [29] Ashish Gupta Chris Bohnert et al. *Use asset microservices and processing profiles*. URL: <https://experienceleague.adobe.com/docs/experience-manager-cloud-service/assets/manage/asset-microservices-configure-and-use.html?lang=en#create-custom-profile> (visited on 08/05/2021).
- [30] Chris Bohnert Paul G et al. *Getting Started with HTL*. URL: <https://experienceleague.adobe.com/docs/experience-manager-htl/using/getting-started/getting-started.html?lang=en> (visited on 08/05/2021).
- [31] Adobe. *Component Basics*. URL: <https://experienceleague.adobe.com/docs/experience-manager-learn/getting-started-wknd-tutorial-develop/project-archetype/component-basics.html?lang=de&mt=false#sling-models> (visited on 08/05/2021).
- [32] Jonas Wagner. *smartcrop.js*. URL: <https://github.com/jwagner/smartcrop.js/> (visited on 08/06/2021).
- [33] Jamie Delbick et al. *Adobe Asset Compute Client*. URL: <https://github.com/adobe/asset-compute-client> (visited on 08/06/2021).