Cloudy Emulation – Efficient and Scalable Emulation-based Services

I. Valizada, K. Rechert, K. Meier, D. Wehrle, D. v. Suchodoletz and L. Sabel,
Albert-Ludwigs University Freiburg
79104 Freiburg i. B., Germany
{isgandar.valizada, klaus.rechert, konrad.meier, dennis.wehrle, dirk.von.suchodoletz, leander.sabel}
@rz.uni-freiburg.de

ABSTRACT

Emulation as a strategy for digital preservation is about to become an accepted technology for memory institutions as a method for coping a large variety of complex digital objects. Hence, the demand for ready-made and especially easy-to-use emulation services will grow. In order to provide user-friendly emulation services a scalable, distributed system model is required to be run on heterogeneous Grid or Cluster infrastructure.

We propose an Emulation-as-a-Service architecture that simplifies access to preserved digital assets allowing end users to interact with the original environments running on different emulators. Ready-made emulation components provide a flexible web service API allowing for development of individual and tailored digital preservation workflows. This paper describes design and implementation of scalable emulation services as part of the bwFLA EaaS framework.

1. INTRODUCTION

Emulation is a key strategy in digital preservation and access to digital artifacts, ensuring that digital objects can be rendered in their native environments and thus maintain their original "look and feel." In most cases the original applications or operating systems developed by the respective software vendors are the best candidates for handling a specific artifact of a certain type [5, 9].

As the number of different past and current computer systems (i.e. hardware architectures) is limited, the number of required emulator-setups is thereby also bounded. Hence, providing access to emulation is suitable for standardized preservation services as well as efficient preservation planning. Nevertheless, deploying full emulation software stacks is a complex and laborious task. Based on these observation, the concept of Emulation-as-a-Service (EaaS) has evolved, aiming towards standardized set of interfaces and uniform access to emulation technology allowing a large, non-technical user-group to make use of emulators and interact with emulated system environments.

Thhis paper's contributions are as follows. We present an EaaS implementation and service model and discuss design issues providing scalable emulation services. We show how *emulation-components* as a core component interact with various emulators and provide necessary APIs and services for data IO like attaching and detaching of virtual removable devices or hard-disks. EaaS users can choose from two different base services: to interact with original environments directly or set up complex preservation workflows. Finally, we present methods for the deployment of EaaS in the cloud (and its scaling on user demand) as well as for user and service authentication in a distributed framework.

2. ARCHITECTURE

The main goal of an EaaS architecture is to develop and maintain a standardized and scalable emulation service model to make emulation a cost-effective digital-preservation strategy and improve its usability. Such a service model, then includes emulated environments either for individual object rendering or represents a component in a larger, complex digital preservation workflow. In contrast to previous projects and approaches to improve usability of emulation technology, the bwFLA project [1] implements a distributed framework. Compared to local provisioning of a complex service stack as proposed in KEEP [3, 2], a networked approach reduces technical and organizational hurdles on the client's side significantly. Instead of adapting a large software package including proprietary software components to various, fast changing end-user devices, the emulators run in a well controlled environment.

The fundamental building block of an EaaS architecture are abstract *emulation-components* (EC) used to standardize deployment and to hide individual system complexity. An EC encapsulates various emulators, available either as open source or commercial products, into an abstract component with a unified set of software interfaces (API). This way, different classes of emulators become also interoperable, e.g. emulators of different vendors could be combined into a larger network compound. The control interface in combination with node- and user-management methods as well as emulator utilities, e.g. for dealing with virtual images, represent a comprehensive EaaS API. Gateway nodes expose the EaaS web-service API. They are responsible for client authorization and authentication as well as for delegating resource requests to the machine management node. The *machine management node* is responsible for efficient hardware utilization, promoting or demoting machines on-demand. To hide complexity of managing dynamic machine allocation, the gateway node also acts as proxy node of an emulation component. The proxy replicates the EC's API, but hides the cloud-specific internal communication from the client. (Fig. 1)

Users need only to implement a single API, which should encourage both interoperability and integration of further, possibly user-contributed, ECs. Furthermore, emulation components are accessible through dedicated web-service (WS)

[1] bwFLA – Functional Long-Term Access, `http://bw-fla.uni-freiburg.de`.

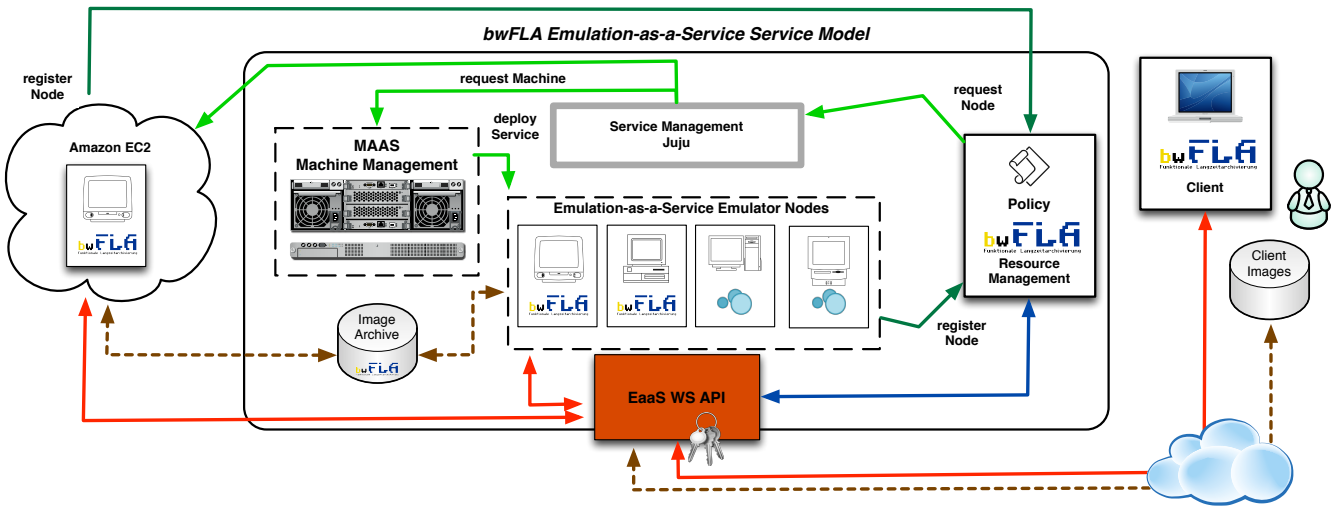


Figure 1: bwFLA: Emulation-as-a-Service General Architecture, Components and Service-Model.

interfaces. This architecture does not enforce specific client implementations. Currently, two variants of clients are available. The user is able to instantiate an EC through a web-front-end and interact with the emulated system interactively. For this option, several standard workflows are already implemented. Such as: bwFLA Ingest/Access/SW-Archive Ingest. The second option is to download the JavaEE-based client framework and build custom workflows.

2.1 Emulation Components Nodes

Emulation-components are implemented as Java EE classes, wrapping a native-platform executable and mapping the emulator's technical capabilities to common interfaces. For instance, every emulator uses a slightly different approach to deal with a set of standard operations like starting and stopping the virtual machine, attaching and detaching virtual drives (floppy, optical or disk drives) or handling network connectivity. Access to the API of any emulation component is possible via its WS front-end. For this a so called WS-service client stub has to be generated via any suitable tool. The generated stub will represent a means of accessing the remote methods of the emulation component, supporting sophisticated client implementations, e.g. in context of specialized workflows.

Currently, the user is able to directly interact with emulated environments using either an HTML5-based web-client or a JAVA-based desktop client. Data-I/O and machine interaction, such as attaching / detaching removable media to the emulation component, is made possible through dedicated utilities in the bwFLA framework and their interfaces.

A future option is to provide dedicated interfaces for non-interactive machine-machine communication, e.g. providing direct access to databases running in an emulated system via network or ODBC interface. A detailed technical description of an EaaS framework and its workflows can be found in earlier work [4, 8].

Finally, to provide a cost-efficient and scalable emulation framework a large scale and especially flexible computing back-end is required. Different emulator types and workloads as well as specific access patterns may require variable computing resources.

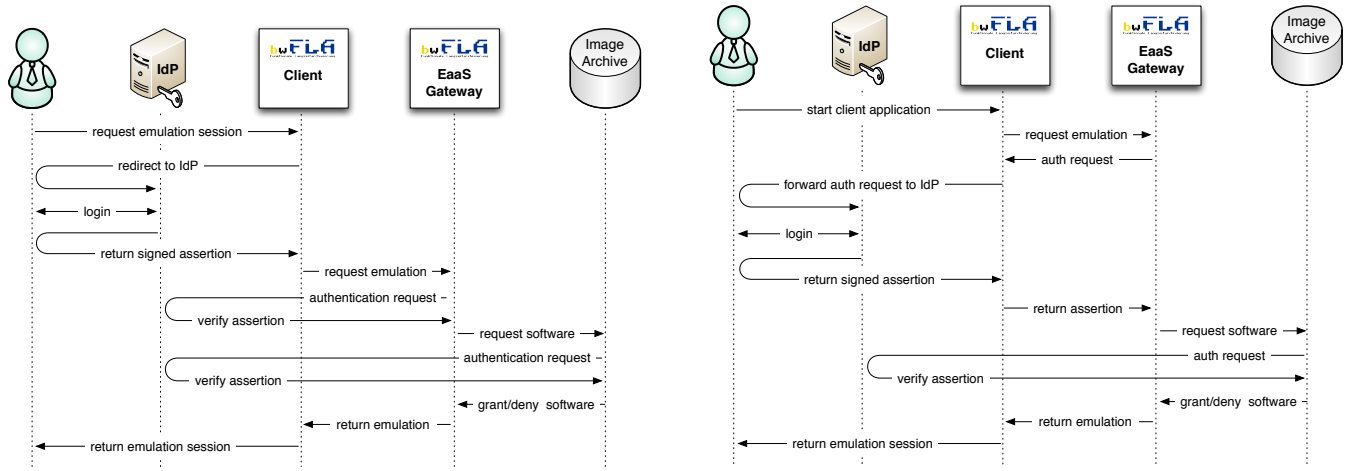
2.2 On-demand Deployment – Scaling EaaS

To become scalable and cost-effective, emulation components need to be deployed only when needed. For this, a suitable framework for hardware- and software-deployment is required. For our purposes, we have chosen Canonical's Metal as a Service (MASS) [2] for hardware management, i.e. for creating emulation component machine instances on demand. If additional hardware resources are required, MASS is responsible for allocation and preparation of suitable machines and installation of a basic operating system (e.g. Ubuntu Linux) on that particular machine. For this, MAAS starts a new physical machine, booting from a DHCP server, downloads and automatically installs the desired OS. Finally, MAAS initializes a user account (e.g. by copying a public ssh-key) for further machine preparation and maintenance. More nodes can be added by just connecting a new machine to power and network. The machine must be capable of booting from the latter using e.g. PXE.

After a machine has been successfully instantiated, in a second step the software deployment system Juju [3] starts installation and configuration of the bwFLA-framework. Juju is an orchestration management tool that requests installed machines from the underlying layer, in our scenario from MAAS. Then it deploys the requested service on that machine by running a (shell) script which installs and configures all needed services automatically. In this way, it is possible to scale a service by requesting additional instances through Juju, e.g. for short-term requirements. If a node is no longer needed, for example, due to a lower load on the cluster, the node is marked as unused and powered down. Hence, the cluster saves energy or the node can be reused for some other services. This service-oriented view abstracts from the underlying hardware and makes the service deployment very simple. A benefit of having the flexible service orchestration tool like Juju is the possibility to use multiple environments for deployed services. Therefore it is possible to have both a local hardware pool managed with MAAS and a commercial

[2] Ubuntu Metal as a Service, `http://maas.ubuntu.com`, version 1.2+bzr1373

[3] Juju, `http://juju.ubuntu.com`, version 0.6.0.1+bzr618



(a) Authentication workflow for web based authentication

(b) Workflow for desktop client based authentication.

Figure 2: Access control in bwFLA Emulation-as-a-Service framework

solution like Amazon EC2.

Since some emulation components require direct hardware access, for instance, CPU virtualization features or CPU ring access, we have chosen a setup which is able to request physical machines as well as virtual ones. In order to reduce hardware costs, commodity hardware is used for the emulation component nodes. The current hardware pool used in out tests consists of standard desktop PCs with a quad-core processor (Intel i5-3470), 8 GByte RAM and a 500 GByte hard disk.

2.3 Provisioning of Legacy Environments

Another important aspect of an emulation service is providing ready-made original system environments, consisting of a basic operating system installation, tailored to be used with a certain emulator type. Typically, emulators provide a set of emulated peripherals and computer components, such as a network chip, sound- and video-card, etc. To make use of these features, appropriate drivers need to be installed and configured. These images act as a base platform, allowing the user to extend them into specialized rendering environments.

Usually, base-images as well as tailored user-images, are kept at specialized institution supplied storage sites thus providing a large variety of systems and specialized software. Furthermore, in some cases, users may choose to not use preconfigured images. In this case, the emulation component should be able to accept a user-provided image directly. For frictionless access through an appropriate emulation component, a suitable network transport protocol is necessary since network quality of service is crucial for usability and performance of the emulated system environment. Especially in cases of user-provided images, network utilization and potentially restricted bandwidth matter. Therefore, a block-oriented protocol has been chosen instead of file-based access since the file representation of a typical emulator image is internally structured as a virtual block-device [7]. A network block device (NBD) and its protocol implement block-layer access over network, i.e. emulating access to physical block-devices, e.g. hard-disk drives. In contrast to file-based access patterns, block-oriented disk-blocks are only requested and transmitted when needed. Thus, an EaaS emulation component becomes immediately operational after initialization. Furthermore, less data transfer might be necessary for instance in case of a sparsely populated virtual disk.

A single virtual disk image may use a few MBytes of storage space for older environments, in some cases up to hundreds of GBytes for newer ones. For efficient and cost-effective maintenance of ready-made images and their user customizations, creating copies for each instance should be avoided. NBD access supports copy-on-write overlays, i.e. allowing for a separation of base-images and user modifications. Any user modification is then stored in a separate block-based differences-file, which can be discarded after session termination or can be kept for future sessions.

2.4 Access Control

To protect resources and, in a second step, support user accounting, a distributed authentication and authorization system is required. Usually, memory institutions already have single-sign-on system deployed, e.g. to protect access to digital publications. These systems were used as starting point for further development. In case of a distributed emulation service, a single central identity provider is not sufficient since users entrust their personal data to their local memory institutions for safekeeping their digital artifacts (e.g. research data). To manage individual user accounts across different sites, a distributed identity management approach is required, delegating authorization and authentication to trusted institutions' identity providers (IdP). Federated identity systems are already successfully integrated at research institutions and universities. For instance, Germany's universities commonly use Secure Assertion Markup Language (SAML) [1] based on identity provider systems, typically Shibboleth [6].

To begin an emulation session a user has to start a client application that is able to communicate to the web-service interface of the emulation component and with the IdP. This can either be a web-site or a program running on the user's computer. The web-site can use the SAML web-login procedure to authenticate the user and access the web-service [10].

The desktop client's ECP module requests an emulation

session from the emulation component on the user's behalf. The emulation component verifies the client's right to use the user's permissions by requesting a signed assertion of the user's IdP from the client. The client's ECP module will forward this request to the IdP and ask the user to authenticate to the IdP (e.g. by providing username and password). The IdP then signs the assertion that grants the user's rights to the client if the user has granted the delegation of his or her rights. The assertion is returned to the emulation component by the client's ECP module. This authorizes the client to interact with the emulation component on the user's behalf.

The emulation component is able to use the same process to request a software image from the archive and return the emulation session to the user if the user has the necessary permissions to access the emulation and the software.

3. RESULTS & DISCUSSION

In order to deploy the bwFLA-framework automatically via Juju/MAAS, some initial effort is required e.g. creating deployment scripts. Installation and software dependencies are to be made explicit and need to be determined upfront. However, as a result, not only a stable and useful service is available but also a documented, reliable and reproducible deployment / installation procedure for other contexts or for future reference created as a by-product.

Currently, emulation components for all major past and present desktop CPU types, PowerPC, Sparc, Motorola 68k, Intel x86, etc., and major operation systems, e.g. OS/2, various MS Windows versions, Apple Macintosh 7.x and newer, etc., have been deployed and can be utilized. Computing nodes running emulation components are either available in cached-mode or need to be created on-demand. If a node is in cache mode it already contains an installed and configured bwFLA framework but is currently inactive. If no more cached nodes are available, 'on-demand' nodes require full installation and configuration. On our available hardware pool, basic node installation and preparation takes about 6-10 minutes plus deployment of the bwFLA framework (2 minutes). Releasing an unused node takes about 1 minute.

4. CONCLUSION

EaaS makes emulation widely available for non-experts. Thus, emulation could prove valueable as a tool in digital preservation workflows, and hence, could become a relevant preservation strategy in many memory institutions. While licensing of past and current software components was not considered in this paper, organizational and technological challenges of emulation as a cost-effective and scalable strategy were analyzed.

The proposed architecture offers both a scalable and easily extendable solution. The scalability of the approach allows the instantiation of emulation nodes on user demand for new emulation resources. The ease of extendability is enforced by the proposed architecture, which is directed towards abstraction of only practically important emulation operations and delegation of their implementation to specific emulator handling classes as well as minimization of effort for adding these handlers in the system's code.

Furthermore, since resources are allocated only on demand, running and maintaining EaaS as a commonly shared infrastructure is efficient in terms of monetary costs, maintenance and management overhead. An unsolved, remaining issue for such a service model is licensing of software. Hopefully, with the availability and the necessity of emulation-based preservation strategies, this issue will vanish.

Acknowledgments

The work presented in this publication is part of the *bwFLA – Functional Long-Term Access* project funded by the federal state of Baden-Württemberg, Germany.

5. REFERENCES

[1] S. Cantor, J. Kemp, R. Philpott, and E. Maler. Assertions and protocols for the oasis security assertion markup language (saml) v2.0. Technical report, OASIS, March 2005.

[2] B. Lohman, B. Kiers, D. Michel, and J. van der Hoeven. Emulation as a business solution: The emulation framework. In *8th International Conference on Preservation of Digital Objects (iPRES2011)*, pages 425–428. National Library Board Singapore and Nanyang Technology University, 2011.

[3] D. Pinchbeck, D. Anderson, J. Delve, G. Alemu, A. Ciuffreda, and A. Lange. Emulation as a strategy for the preservation of games: the keep project. In *DiGRA 2009 – Breaking New Ground: Innovation in Games, Play, Practice and Theory*, 2009.

[4] K. Rechert, I. Valizada, D. von Suchodoletz, and J. Latocha. bwFLA – a functional approach to digital preservation. *PIK – Praxis der Informationsverarbeitung und Kommunikation*, 35(4):259–267, 2012.

[5] J. Rothenberg. Ensuring the longevity of digital information. *Scientific American*, 272(1):42–47, 1995.

[6] T. Scavo, S. Cantor, and N. Dors. Shibboleth architecture: Technical overview. *Working draft*, 1, 2005.

[7] C. Tang. Fvd: a high-performance virtual machine image format for cloud. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, USENIXATC'11, pages 18–24, Berkeley, CA, USA, 2011. USENIX Association.

[8] D. von Suchodoletz, K. Rechert, and I. Valizada. Towards emulation-as-a-service – cloud services for versatile digital object access. *International Journal of Digital Curation*, 8:131–142, 2013.

[9] D. von Suchodoletz, K. Rechert, J. van der Hoeven, and J. Schroder. Seven Steps for Reliable Emulation Strategies – Solved Problems and Open Issues. In A. Rauber, M. Kaiser, R. Guenther, and P. Constantopoulos, editors, *7th International Conference on Preservation of Digital Objects (iPRES2010) September 19 - 24, 2010, Vienna, Austria*, volume 262, pages 373–381. Austrian Computer Society, 2010.

[10] R. Zahoransky, S. Semaan, and K. Rechert. Identity and access management for complex research data workflows. In *6. DFN-Forum 2013 Kommunikationstechnologien*. GI, 2013.