Preserving Access to Web Servers

A Case Study Preserving Output of Collaborative Research Centers

Rafael Gieschke
University of Freiburg
Germany
rafael.gieschke@rz.uni-freiburg.de
0000-0002-2778-4218

Klaus Rechert
University of Freiburg
Germany
klaus.rechert@rz.uni-freiburg.de
0000-0002-8807-018X

Susanne Mocken
University of Freiburg
Germany
susanne.mocken@rz.uni-freiburg.de
0000-0001-8752-7785

Abstract – With the advent of the so-called Web 2.0 technologies, web resources became the primary platform for internal and external collaboration, knowledge exchange, and self-publication. Many of these services cannot be fully preserved by solely relying on client-side technologies like crawlers. In particular, their dynamic and reactive character will be lost by generating static snapshots. In this article, we develop concepts and describe infrastructure built on emulation to address the problem of retiring obsolete web service infrastructure while keeping the service long-term accessible and usable. We show the practicability of this approach by the example of three use-cases from academic projects.

Web Server, Preservation, Emulation, Case Study

Exploring the New Horizons; Scanning the New Development

I. Introduction

Preserving web resources has been an active topic in digital preservation since the very early days of the Internet (e.g., [1]). With constant technological progress as well as sociological change, the web is not a static publication medium. Thus, collection and preservation practice needs to adapt to new requirements, content types, publication practice, and appraisal.

The web has always been an important publication medium for research results, ranging from informal knowledge sharing, web sites of scientific projects up to public database (front-ends), data collection and the like. With the advent of the so-called Web 2.0 technologies, web resources became the primary platform for internal and external collaboration. These Web 2.0 platforms, e.g., wikis, content management systems, blogs, simplified collaboratively creating content and became a popular tool for (distributed) scientific projects. After the end of many of these projects, without additional funding for maintaining their web platforms, the servers are or will be retired and will be subject to preservation.

Many of these platforms cannot be fully preserved by solely relying on client-side technologies like crawlers. In particular, their dynamic, reactive character will be lost by generating static snapshots. In many cases, fortunately, the server back-end (including software and data) is available for preservation, either as a virtual machine (disk image) or in form of file-system backups. Emulation as a preservation tool [2] is capable of keeping software accessible, especially preserving the software's interactive character. The interactive character of networked software is, however, quite different compared to office application, games, or similar, which are traditional use-cases for emulation. To apply the concept of emulation to networked software, the scope of emulation has to be broadened, new components have to be added, and orchestration of emulation sessions consisting of multiple machines is required as well as novel access options.

We have done a case study to investigate and formalize preservation workflows for such complex web objects in order to implement seamless transitions from a production state to a preserved but (on-demand) accessible state. In this case study, we have preserved three different web servers from two Collaborative Research Centers [1] in the humanities. The web servers were used for internal communication and knowledge sharing as well as public-facing publication platforms. These projects were running between 8 and 13 years and included over 100 staff members.

We further have investigated and implemented methods for secure and functional long-term access to old, unmaintained, and, thus, insecure machines using the EaaS framework. We show different ways of end-user access, both using the user's contemporary browser as well as using an emulated historic browser, which can provide an enriched user experience by serving archived pages from public web archives for any potential external links in archived web platforms.

II. Related Work

Web archiving is a traditional sub-domain of digital preservation with several national and international active actors in this field [3]. The dominant and most scaleable technology is web crawling (e.g., Heritrix [4]),

[1] German: DFG Sonderforschungsbereich

17th International Conference on Digital Preservation
iPRES 2021, Beijing, China.

DOI: 10.1145/nnnnnnn.nnnnnnn

software that builds on the HTTP protocol and works quite similar to a standard web browser. Starting from a (list of) HTTP URLs, a crawler loads individual pages and follows any link found up to a given depth. Finally, content and retrieval metadata are packaged together as web archive packages (WARC, ISO 28500:2017) and become accessible, e.g., through the Internet Archive's Wayback Machine [5]. While a web crawler can efficiently archive a huge number of web sites, the approach has also significant shortcomings. For instance, individual web pages may be inaccessible for certain clients or crawling may take such a long time that linked pages change while the linking page are being archived. The result is neither guaranteed to be complete nor coherent [6], properties which may be problematic, e.g., when preserving web sites relevant for research purposes.

With modern JavaScript, web sites became technically more complex, requiring crawlers to behave like real web browsers, i.e., also executing client-side code to be able to capture sites like Facebook, Instagram, and the like [7]. With web sites behaving more like computer programs rather then text-based publications, completeness is even harder to achieve.

Because HTTP clients are usually only able to access the front-end of websites, the so-called publicly indexable web [8], there is a gap in preservation. More complex web sites are driven by dynamic logic running on the server, where the content (e.g., a database-backed content management system) as well as the scripting logic creating HTML pages on-the-fly are not visible to crawlers. By definition, crawlers deliver an incomplete snapshot of the system when following only the front-end visible links. Depending on the user's role, time/date of retrieval, user agent, etc., number and appearance of visible pages change. Several crawling-based approaches have been proposed to support extracting and indexing of "hidden" information from web sites [9], e.g., by supporting automated form queries. Still, a client-side approach has significant limitations preserving complex server-side logic and content.

In some cases, there is a (backup) copy of the back-end infrastructure available. In a media archaeology experiment, an "Internet server"[2] dated around 1996 should be revived from backups [10]. In order to restore the dynamic character of the original service, emulation was considered as an option. The original system was hosted on a machine using a SPARC-architecture CPU, for which original binaries are difficult to find – even for open source software. In a rather manual process, the system was ported to the x86 architecture, partly by rebuilding software from source code. This – mostly successful – example highlights both the importance of software archiving and automation when restoring complex (networked) artifacts but also poses new questions especially regarding the security of a (re-)published service using old software on the public Internet.

In order to keep complex web artifacts accessible – if the back-end data is available, either as (file-system) backup or disk image – a new process to preserve and re-enact web servers is necessary. Furthermore, the availability of individual retired web servers poses new questions regarding integration with the existing (mostly crawler-based) web preservation ecosystem. Integrating and interconnecting different versions and different instances of preserved web servers, e.g., preserved by different institutions or initiatives, is another open end to be addressed.

III. Emulating Networks

Until now, emulation has been mostly associated with instances of singular machines rendering a digital object, e.g., computer games or CD-ROMs from collections of libraries, archives, or museums [11]. To render interconnected artifacts or artifacts that are not meant to be used with direct user-machine interaction using a keyboard or mouse with direct visual or audible feedback but to be accessed through a networked connection, the concepts of access have to be reworked and are not limited to user access anymore. The interaction with networked artifacts happens through multiple layers of technical protocols (e.g., HTTP over TCP/IP over Ethernet) and requires dedicated client software (e.g., a web browser). Similar to software found in emulated machine instances and media types as well as file formats used with these instances, protocols and networking infrastructure also follow the technical zeitgeist, hence, will eventually become obsolete and not be understood by contemporary hard- and software. For example, we are already observing an – admittedly prolonged – sunset phase of IPv4. In order to support networked scenarios, we need to broaden the scope of emulation by incorporating network components and network services to cover the post CD-ROM era, i.e., the era beginning at the 2000s when the Internet started to become the predominant publication medium. By emulating network components, we are able to decouple preserved networks both from the host system (for security reasons) and, conceptually, from the technical life-cycle.

A. Emulated Network Components

As the base layer for emulated networks, we have identified Ethernet, which can serve as a universal common denominator for many different network types. While higher-level network protocols like TCP/IP, IPX, AppleTalk, etc., are not necessarily implemented or used by archived machines, especially supported by the operating system, Ethernet in particular has a long history dating back to the late 1970s and is widely supported. Additionally, Ethernet is a very simple protocol to be parsed and interpreted, e.g., by hardware components like Ethernet switches. Additionally, Ethernet is frequently used to tunnel other protocols or is itself tunneled over other protocols. Like that, an emulator might, e.g., even emu-

[2] FREEZE, 1996, `http://www.almedia.nl/DDS/Nieuws/freeze.html`

late a dial-up modem for a platform without a direct Ethernet connection, passing through any data sent over a virtual dial-up connection as Ethernet frames to the rest of the emulated network environment.

To implement an emulated network environment, we spawn a new virtual Ethernet network consisting of a central software-based Ethernet switch [12], which allows to connect a number of emulated machines. To exchange network traffic between emulated machines, these need to get "wired", i.e., connected to the central Ethernet switch. The "cables" between software-based Ethernet devices, i.e., an Ethernet network adapter and switch, are implemented as HTTP-based WebSocket connections [13], optionally using TLS [14] for encryption. Using HTTP as transport layer not only allows for (secure) transport of encapsulated Ethernet frames over the public Internet but also ensures a strict separation of virtual network traffic from other network traffic on the host machine, i.e., the host machine does not need to implement custom firewall or network routing rules. To eliminate any danger from archived environments attacking the host system or using the host system's network resources to attack third-party entities on the Internet (or the host system's private network environment), the emulated machines run within a Linux container[15] without network capabilities[16]. The emulated guest's Ethernet frames are passed through a UNIX domain socket to the host system to be forwarded through a WebSocket connection. This setup also shields the archived (unmaintained and insecure) environments from random attacks from the public Internet as they are not visible in the public network.

Due to using HTTP-based connections between emulated machines and emulated network infrastructure, emulated machines can be deployed independently, e.g., on different host machines and can be (re-)connected to a network as needed. This infrastructure also allows to interconnect different, independent network instances, e.g., to orchestrate a larger network of emulated web servers. Lastly, this network setup allows (multiple) ad-hoc client connections, e.g., to start a dedicated emulated client computer to interact with servers within the network (see Section IV.C).

B. Network Services

A crucial building block for an emulated network are network services, services which provide infrastructure normally present in a network. The most important service is DNS name resolution paired with a DHCP IP-address management service, both crucial for automation and orchestration. Since the network is isolated from the live Internet or intranet, machines with any manually assigned IP addresses are supported, allowing unmodified machines to be added to a network while retaining their configured IP addresses. In practice and especially in multi-machine setups, machines typically use (fully qualified) domain names (FQDNs) to identify and address networked machines or services.

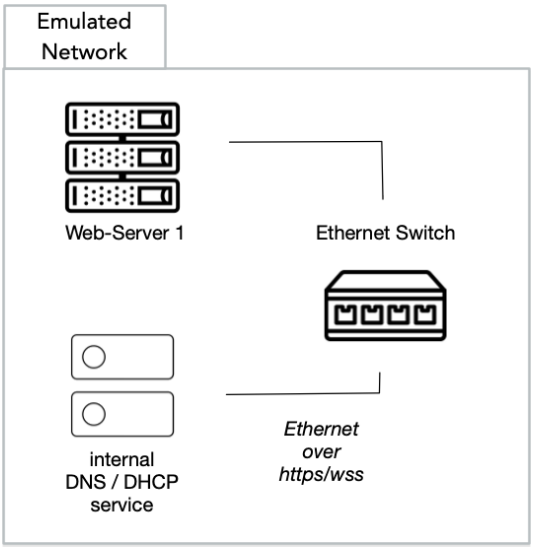

Figure 1: A simple emulated network consisting of a web server instance and an internal DNS/DHCP service, both connected to a central Ethernet switch.

For this, the internal DNS/DHCP service maintains IP address and FQDN mappings and manages their assignment on startup. Relations between machine instances and assigned domain names are maintained as network meta-data. Figure 1 depicts a simple emulated network with a DNS/DHCP service component.

A further important component is the possibility to (optionally) allow emulated environments to access the (live) public Internet. This functionality is implemented by a software component (Slirp) which can act as a gateway in the virtual network and forward any (TCP/UDP) connections to a physical network on the host system. Traffic between the environments and the gateway is still encapsulated through a WebSocket connection (optionally, over TLS). This makes it possible to deploy the Internet gateway on a special machine separated from the rest of the machines (it might even be placed on an externally rented server reachable via the Internet), decreasing the potential security impact on the host's network.

A last option is to connect the emulated network to the archived Internet, which is implemented through a pywb[3] service. This allows to serve archived web sites from external web archives, e.g., the Internet Archive – using a user-defined archival date – or to serve locally provided WARC files. If the service is activated within the network, it acts a transparent proxy routing all HTTP traffic through the pywb service, without having to reconfigure the individual archived machines.

[3] https://github.com/webrecorder/pywb

C. External Connections

A further building block provides the ability to forward traffic from external sources (e.g., machines or services connected via the public Internet or intranet) to an emulated network. External machines which are not part of the emulated network are using the TCP/IP protocol suite directly to communicate with other machines or services. Hence, to support connections from the Internet to an machine or service in an emulated network, a gateway is necessary to translate the network traffic between the external network and the WebSocket-encapsulated emulated network. It can be seen as the inverse of the Slirp component described above. While the Slirp component acts as a gateway from the emulated network to the public Internet, this component acts as a gateway from the public Internet (or an intranet) to the emulated network.

Depending on where and how the gateway is deployed, different access scenarios can be realized. The first option is to connect an application, e.g., the user's browser with a service inside the emulated network. For this, the user downloads a local instance of the gateway software (*eaas-proxy*, currently available for Windows, macOS, and GNU/Linux)[4] and initiates a session via their web browser. A typical session will open a local (localhost) TCP server socket, connect via WebSocket (over HTTPS) to an emulated network and forward local traffic to a service inside the emulated network environment, e.g., TCP port 80 of a web server. All configuration parameters, e.g., (WebSocket) URL of the emulated network, target machine, destination port, are part of the initiation link passed by the user's web browser. Alternatively to forwarding a single TCP port, eaas-proxy can be configured to act as a SOCKS5 proxy server, allowing local applications with SOCKS5 support to reach any host/port in the emulated network.

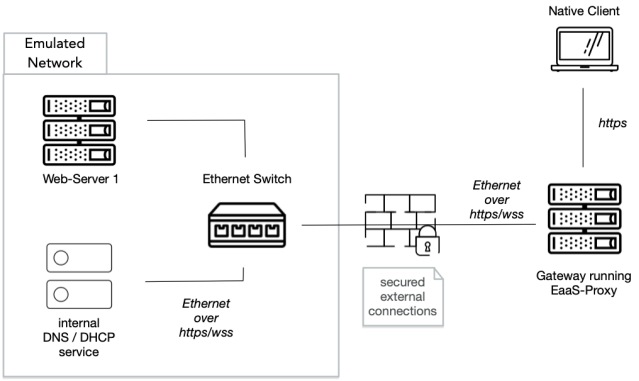

Figure 2: eaas-proxy is running as a gateway, securely connecting a native client application (e.g., a contemporary web browser) to the emulated network. The native client does not have to be changed.

Instead of installing the gateway on the user's machine, the gateway can be deployed as a public/shared network service, e.g., acting as a web server substitute accessible on the archived web server's original domain. The eaas-proxy gateway relays traffic from the public Internet into the emulated network. Figure 2 shows a setup allowing a contemporary browser to connect to an emulated web server via the eaas-proxy gateway.

Since eaas-proxy is implemented as a JavaScript/WebAssembly-based Node.js application, it could also be run directly in the user's web browser. Using W3C's Service Workers specification, any HTTP request of the browser to the archived web server could here be intercepted, serialized to Ethernet frames on the client, and sent via a WebSocket connection directly to the emulated network environment, eliminating any processing of the unencapsulated network traffic on the (EaaS) server and further increasing security.

IV. Access

When preserving web servers, offering simple, ideally seamless access to a wide range of users is the main goal. The main reason to utilize emulation is to preserve the server's interactive character and its full functionality. While the technical challenges of connecting to an emulated network have been discussed in Section III.C, this section focuses on organizational and administrative questions of providing user access.

A. Security Considerations

As machines are archived in their original state, several questions regarding security occur. Operating systems and other software within the archived machines have known or yet unknown security issues. In a (well-maintained) productive environment, these issues, upon discovery and availability of a fix by the software's vendor, would be patched through an update. In a preservation scenario, updates are usually no longer available since the software reached its end of life. But most likely, there also is no intent to update archived machines in any case. The reason for this is that maintaining and updating software can be very operating-system and software-specific and requires detailed knowledge and a fully documented software stack of every archived machine. This turns each update into a machine-specific and labor-intensive manual process, which would not scale with a large number of archived machines.

Additionally, upgrading software to a more recent, vendor-supported version might change the service's behavior. In many cases, an erroneous behavior might be obvious, e.g., a completely non-functional web site after a major PHP version update, but it could be more subtle, e.g., a broken search function on a sub-page. In any case, detecting changed behavior is very hard to accomplish in a generic way and would be in direct contrast to the stated objective of archiving the web server with its original behavior. Furthermore, complex web server setups often include custom software, e.g., developed for a specific project, for which there will be no security audit and updates at all.

[4] `https://gitlab.com/emulation-as-a-service/eaas-proxy`

As a consequence, security issues in archived machines have to be accepted and have to be maintained when a machine is reactivated using emulation. For a general risk assessment, different layers have to be considered:

1. *Isolation of the archived machine and protection of the hosting infrastructure* – In order to protect the hosting infrastructure, the machine needs to be isolated, such that a compromised machine cannot take over the hosting machine or is able to abuse or access any of its resources. Within EaaS, this is realized by running the machine inside an emulator or using hardware virtualization. Since not all emulators are actively audited for security issues, the emulator itself is executed inside a restricted Linux container, in particular without access to the host's resources like other running processes, file system, and network. Emulation and virtualization software but more importantly the host's Linux kernel and container runtime infrastructure are actively maintained software and should be up-to-date, such that the security risk for the host infrastructure running the archived, vulnerable machines is comparable to running any contemporary Internet facing software.

2. *Malicious user attacks* – Any user accessing the archived web server might be able to exploit known issues of the site's back-end software. The most obvious attack is vandalism, i.e., changing contents of a web page or modifying the web server as a whole. This problem can be mitigated by running an emulated instance in a non-persistent mode by default. In EaaS, all data the emulated machine (and the emulator) is able to access, e.g., disk images, is served via HTTP GET requests and thus read-only by definition. All data written is temporary and destroyed when a session ends unless explicitly saved as a new disk image layer. If the session model (cf. Section C) allows multiple users sharing a single session, the vandalism problem can only be mitigated, e.g., by resetting the session in regular intervals.
   Furthermore, a malicious user could exploit an emulated machine to launch attacks against other users sharing the same session. If the server is accessed through an emulated *remote* browser, the main risk for an attacked user is a negative user experience, e.g., if the browser crashes, similar to the vandalism problem. However, social engineering attacks could be possible, too, e.g., asking another user in a shared session to input personal data by making the emulated web server serve a manipulated web page. If the server is accessed through the user's browser, the risk is comparable to the general risk of browsing the Internet. The user should use an up-to-date browser and use common security practice when downloading content from untrusted sources or sharing sensitive (personal) data with the service.

3. *Exposure of sensitive user data* – Another problem occurs if visitors, in presence of security bugs, can essentially access any part of the web server and web site, including administration interfaces and, in particular, content which was originally not meant to be public for privacy or security reasons, e.g., usernames, email addresses, or birthdays, but also TLS certificate and keys, password hashes, or SSH keys of users of the in-production service. Machines with sensitive content require a rigorous curatorial review and need to be redacted, if necessary, following common practices. Alternatively, access to these machines can be restricted to a dedicated and trusted user-group and through dedicated (emulated) remote clients, which do only have limited capabilities and (hacking-)tools available.

Even though emulated machines, e.g., web servers, are accessible, these are never exposed directly to the Internet and therefore have a reduced attack surface. Traffic can be proxied and, e.g., HTTPS connections are managed and terminated by the proxy, such that security problems in the TCP/IP and TLS stacks are mitigated (e.g., CVE-2014-0160 a.k.a. Heartbleed). Since the emulated machines do not appear in the public IPv4/6 address space but only in emulated network environments, they do not appear in automated scans of the public Internet.

All these measures help to significantly decrease risks of running machines with known security issues and simultaneously allow less restrictive (public) access models.

B. Maintaining built-in security measures

A different kind of security problem are security measures deployed inside an archived machine with the goal to protect its content or identity. For instance, an archived web server might only provide access to its content via HTTPS (HTTP over TLS). The archived TLS certificates might only be valid for the wrong domain name, already have expired, or lack a valid certificate chain to a root CA certificate trusted by today's systems. The archived server might also only support deprecated and outdated SSL/TLS versions which are no longer supported by today's systems.

Generally, solving the problem is possible by adding a (reverse) proxy in front of the emulated server, which simply accepts any certificate presented by the server. This does not compromise security because, as described in Section III.C, access to the emulated server running in an emulated network environment is already secured on a lower layer. Accessing a TLS server with an outdated TLS version is more difficult, however, as contemporary software libraries (e.g., OpenSSL) might have dropped support for these versions. Older library versions still supporting the outdated TLS versions cannot simply continue to be used as the reverse proxy is running on the host system and thus has to be up-to-date to not compromise the host system.

A solution, which still has to be evaluated, might be to include a TCP-TLS proxy service (in parallel to the existing DNS/DHCP network service) in the emulated network environment, proxying from (raw) TCP to (old) TLS versions. As it would run in an emulated machine inside the emulated network, this proxy service could be built using outdated library versions without compromising security of the host system.

C. Session Setup & Management

The most important consideration for user access is the session setup. Based on the aforementioned technical infrastructure and security considerations, different setups are possible:

1. *Contemporary browser access* – In this case, the user accesses the web site with their contemporary browser. This is usually the most convenient option since there is typically no notable difference to browsing any other web site. In addition, all URLs (at any depth) can be shared or bookmarked. This approach, however, is usually only available for a (short) *transition period*, as long as there are no significant security concerns and as long as the contemporary browser renders the pages (e.g., there is no need for deprecated plugins like Java, Adobe Flash, etc.).
2. *Remote browser* – In order to render old web pages with browsers of their time, a lightweight *remote browser*, originally developed and made available by the Old Web Today project [5], is a good compromise. Remote browsers are usually older versions of popular browsers – and due to known security issues not recommended to be used on a desktop computer – running in a concealed and secure environment and offer features a contemporary browser does not support anymore (e.g., Flash support). The variety of the lightweight remote browsers, however, is currently limited to browsers and plugins that have been released for Linux-based operating systems. For native Microsoft Windows browsers (predominately Internet Explorer 6-8) and especially Windows-specific plugins like ActiveX, a fully installed Windows client machine is necessary (see next option).
3. *Emulated machine client* – While option 1. and 2. are lightweight (with regards to resource consumption) and offer a good user experience, these options are limited, both regarding their availability and coverage of potential use-cases. A long-term available and fully back-end compatible option is a full client installation.

With remote browser access or emulated machine clients, further options become possible, e.g., integrating web archives (using pywb) and thus creating the illusion of consistently browsing the web of that time. External links from preserved web servers will be relayed either to a pre-configured web-archive (e.g., Internet Archive) with a pre-configured date or, alternatively, WARCs preserved together with the server artifact.

Regardless of which option is chosen, the emulated web sites are accessible through or as a (public) web page, potentially available under the service's old URL or domain.

The second consideration for user access concerns the session management. Currently, three options are possible:

1. *Static setup* – With this session setup, a network is setup to run permanently. It also offers the best user experience as the site is immediately available and interaction between users is possible. Ideally, this setup is chosen for an initial sunset phase of a web site, when there is still significant traffic and the general security risks are considered as low (cf. malicious user attacks and sensitive data exposure).
2. *On-demand setup with shared sessions* – For web sites with low demand, on-demand access is a cost efficient and more secure option. If a user tries to load the page and no active network for the requested instance is running, the proxy shows the user a waiting message and simultaneously initiates a new emulated network session. The startup time may vary, usually 1-2 minutes, in some cases it can take up to a few minutes. Once a session is running, subsequent users do not experience a waiting time. If there is no active user within a configured grace period, the network is shut down.
3. *On-demand setup with individual sessions* – For security-critical instances or instances with a restricted user group, for every user a new (private) network instance is created and kept alive as long as it is in use.

V. Use-Cases

The popularity of so-called Web 2.0 technologies created a huge boost to self-publication and collaboration platforms. Blogs, content management frameworks and wikis could be quickly deployed and used, e.g., for internal and external communication of scientific projects. Once such a service is public and finds its users, it requires constant maintenance to mitigate security issues. Once the underlying software stack reaches end-of-life, an important decision is to be made: invest and migrate the content to an up-to-date technology stack or retire the service. Both options might be unavailable due to the lack of funding and due to the promises made to funders (the DFG typically requires a 10 years data retention time) or a still active user community.

For such services, we propose a third solution, a phased retirement, balancing security, access and costs. To test this approach and further refine generic web

[5] `https://oldweb.today`

server preservation workflows using emulation, we have acquired three different web server instances subject to retirement and preservation from two long running DFG Collaborative Research Centers (German: Sonderforschungsbereich, SFB).

With the end of project funding, the web servers will meet their unavoidable fate: as there will be no administrator left who can take care of further development of any kind and the installations have reached end-of-life (running Ubuntu 16.04 LTS[6]), imminent action is required.

A. The Heroic as "Gift" on the Victorian and Edwardian Book Market (CRC 948)

This use-case is a spin-off of the research project "The Heroic in British Periodicals between 1850 and 1900: Competing Semantics and Modes of Presentation", a completed sub-project from the first funding phase of the Collaborative Research Center 948 "Heroes – Heroizations – Heroisms: Transformations and Conjunctures from Antiquity to the Modern Day".

The collection is dedicated to a popular book genre, so called "Gift books", which served as a medium for spreading ideas about the heroic in Victorian society and perpetuated Victorian concepts of the heroic into the 20th century. The "hero books" were produced mostly as gift books in the narrow sense: they were given as presents, usually to young readers.

The examples gathered on the WordPress website are represented by 90 entries of narratives about heroes and heroic deeds. The data was entered using a WordPress add-on that stores them in a relational database. However, when the project had been set up, aspects of sustainability were not considered. As a consequence, the data can only be extracted again with considerable effort and transferred to another system.

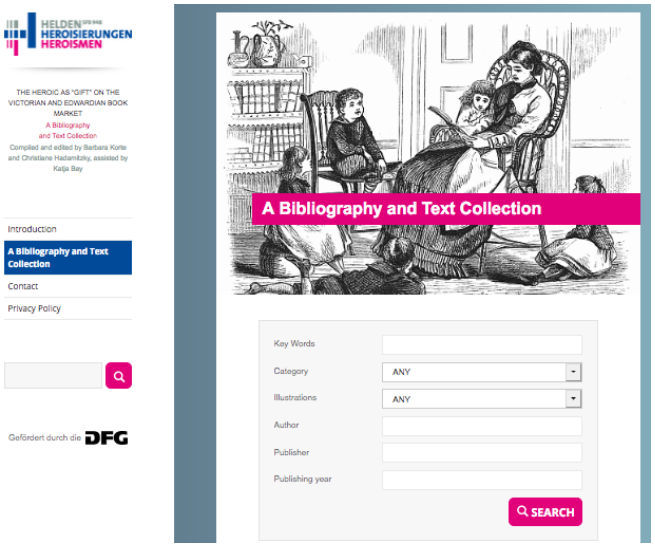

Figure 3: Website "Heroic As Gift".

[6]Ubuntu 16.04 LTS reached end-of-life for its (free) standard support in April 2021, see `https://ubuntu.com/16-04`.

The original version is still accessible, albeit an expired TLS certificate.[7]

B. Places of Otium – Worldwide

The website "Places of Otium – Worldwide" is an interactive project developed by the CRC 1015 *Otium. Societal Resource. Critical Potential*. It was created during the second funding period (2017-2020), using WordPress as a content management system. Numerous individuals were asked to describe a place that they personally associate with *otium*. Many of the interviews were supplemented with comments by the researchers of the CRC 1015, explaining their approach to the subject. All these places can be experienced on an interactive world map through short texts, images, and films and offer a low-threshold introduction to the topic of "otium".

The map is implemented as JavaScript overlay using a public OpenStreetMap map.

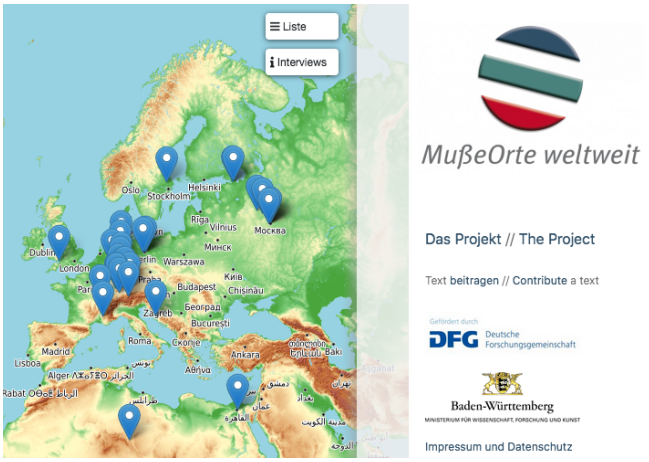

Figure 4: Website "Places of Otium – Worldwide".

The original version is still accessible and co-exists with the emulated version. [8]

C. Muße/muoze digital

The third use-case is a MediaWiki-based online platform called "Leisure/muoze Digital – Medieval Variants of Leisure", created during the first funding period of the CRC 1015 "Otium. Societal Resource. Critical Potential" (2013-2016). It reflects the results of the completed sub-project C1, revolving around the question where "leisure" has its systematic place in Middle High German literature, researched on the basis of courtly literature from around the 11th century and religious literature of the 14th century. Before illustrating the changing semantizations in an exemplary way, the category of *otium* had to be made analytically usable because it is usually not mentioned as such in the medieval texts. The dynamic form of presentation makes it clear that the rela-

[7]Original version: `https://www.heroic-as-gift.uni-freiburg.de/`. Emulated server version: `https://sfb948.web.museum/`

[8]Original version: `https://www.musseorte-weltweit.uni-freiburg.de/`. Emulated server version: `https://sfb1015.web.museum/en/`

tionships between the words cannot generally be fixed.

Although the sub-project has been completed in 2016, the MediaWiki platform was supposed to remain accessible during the second funding period. Due to very poor documentation of the underlying server and initially no access data to the wiki software administration, a newly formed project infrastructure team had to regain full access to be able to maintain the platform for an additional 4 years.

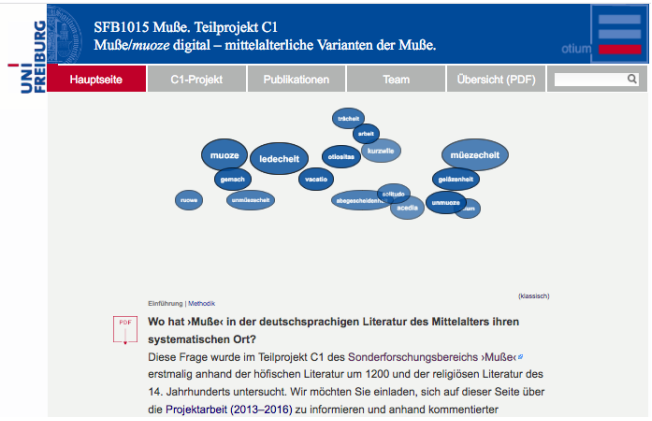

Figure 5: Website "Leisure/muoze Digital".

The original version is still accessible and co-exists currently with the emulated version. [9]

VI. Workflows

The individual steps described have been similar or identical for all three use-cases.

A. *Import machine data*

The web site's data has been acquired as a disk image from a VMware ESX cluster. Based on a predefined template, a matching emulation environment was created and the disk image has been attached. Figure 6 shows the current UI workflow.

B. *Prepare for Network*

Before adding it to a virtual network, we had to perform a few preparatory steps (some of these tasks will be automated in the future). First, access to the virtual machine had to be gained in order to change configuration files and redact sensitive data. In principle, this could be done without booting the emulated machine but by directly manipulating its disk image. Although, it still has to be improved and generalized, the EaaS framework already offers this functionality as "image generalization" of disk images, e.g., to replace disk drivers in a Windows image unable to boot with the wrong disk drive drivers. In practice, however, it can also be advantageous to boot the emulated machine in EaaS and apply any needed changes interactively using the original (archived) tools and workflows. For this, the username

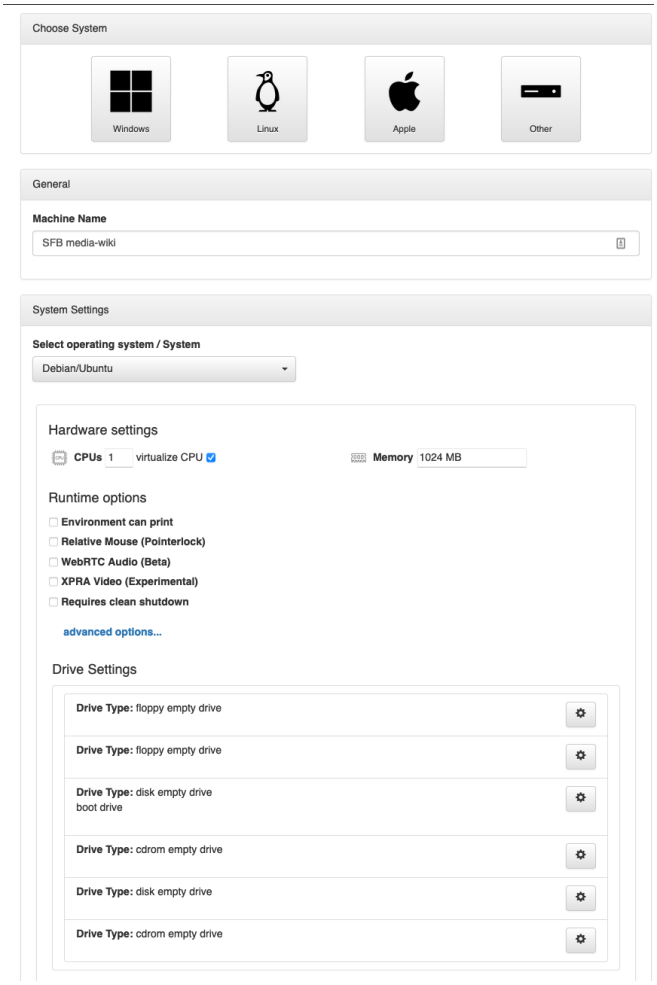

Figure 6: Create a new emulated machine from EaaS templates.

and password of an admin user has to be known. Alternatively, in most cases, it can be reset by modifying the disk image's `/etc/passwd` and `/etc/shadow` files.

For security reasons, web servers in productive environments will often only accept a configured domain name (FQDN) and port in the Host header sent by the client (e.g., a web browser). When a wrong domain name is sent, the server will redirect the client to its configured domain name. For preserving the machine, this proves to be a problem as the original domain name will normally no longer be available. While for the session setups of remote browser and emulated client machine (see Section IV.C), the domain name seen and sent by the client can generally be freely configured and controlled, this is not the case when using contemporary browser access. Although a reverse proxy in front of the emulated web server could transparently change the sent Host header to match the expected Host header, e.g., scripts included in the web pages delivered by the web sites could still depend on and unconditionally use the previously hard-coded domain names. Thus, while for normal operation, this would be considered an anti-pattern for security reasons, it is usually best to have the web server accept any domain name in the Host header

[9] `https://musse-digital.uni-freiburg.de/index.html`. Emulated server version: `https://musse-digital.web.museum/`

and use the sent domain name in any output it produces (e.g., in absolute hyperlinks, included scripts and style sheets) or only produce relative links.

Sadly, how to accomplish this, is highly dependent on the web server and target application. For the two described WordPress instances, it was enough to change `/var/www/html/wp-config.php` to include:

```
define("WP_SITEURL","//".$_SERVER["HTTP_HOST"]);
define("WP_HOME","//".$_SERVER["HTTP_HOST"]);
```

at its start. For the MediaWiki instance, `/var/www/html/c1/LocalSettings.php` had to be changed to include:

```
$wgServer = "//" . $_SERVER["HTTP_HOST"];
```

at its end.

The web server should not use or enforce HTTPS (certificates will expire and need maintenance). Secure connection will be provided through the access infrastructure. If the web server previously used TLS, private keys to existing TLS certificates have to be removed. If this is not possible, the key has to be considered as compromised because, as outlined in Section IV.A, users will, without further precautions, eventually be able to access any files on the emulated machines. Therefore, technologies like Certificate Transparency [10] can be used to search if the same key is used for any other certificate [11]. As Certificate Authorities (CAs) must, within 24 hours, revoke all and not issue new certificates with keys known to be compromised, all of these certificates will, at least eventually, get revoked in any case.

Optionally, other sensitive (private) data has to be deleted or redacted from the system. Such data might not only be found in content data (e.g., in a relational database) but also hide in configuration files (e.g., HMAC keys used to authenticate cookies) or log files (e.g., `/var/log/apache2` might contain passwords for external databases or users of the installation). If the original instance continues to be accessible, it might be jeopardized. This step is recommended for machines with public access.

Absolute links including hard-coded domain names can potentially also be found in other places on the web servers, e.g., in static HTML files or even PDF files. It is much harder to detect and change all of these links but, luckily, was not necessary to get the archived applications working.

Finally, an emulated network environment was created in the EaaS framework. Figure 7 shows the current UI, which includes options to enable access to the current live Internet or an archived version with a configured date (via the Internet Archive's Wayback Machine). It also allows to enable the DNS/DHCP service component (see Section III.B), which assigns IP addresses to emulated machines and makes them reachable via a configurable domain name in the emulated network. The archived machine containing the web server was then added to this network environment. Figure 8 shows the current workflow, which allows to configure its domain name under which it will be reachable from other machines (e.g., remote browsers, emulated machine clients) in the network environment.

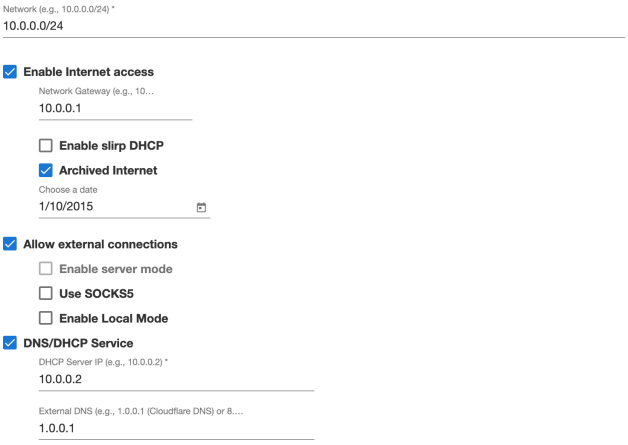

Figure 7: Create a new emulated network environment.

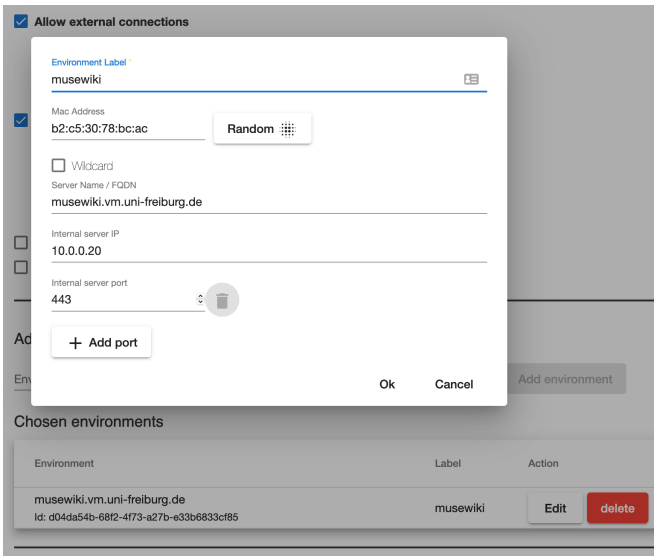

Figure 8: Add the preserved web server to the emulated network environment.

C. Deploy

For this case study, contemporary browser access (see Section IV.C) was chosen as access method. For this, Docker Compose was used to run eaas-proxy and the Caddy reverse proxy [12] in Docker containers [13] on a dedicated gateway machine. As the software running on the

[10] `https://certificate.transparency.dev/`

[11] A ready-made tool for this task based on Certificate Transparency can be found at `https://crt.sh/?spkisha256=`.

[12] `https://caddyserver.com/`

[13] `https://gitlab.com/emulation-as-a-service/experiments/eaas-proxy-service`

gateway is not very resource demanding, one gateway instance was able to host all three described use-cases.

Caddy accepts HTTPS requests from clients on the public Internet and processes and forwards them to the respective eaas-proxy instance. It terminates TLS encryption both in direction of the client (where it also automatically obtains a valid TLS certificate via Let's Encrypt) as well as in direction of the emulated archived web server (where it is configured not to verify the validity of (expired) TLS certificates). The eaas-proxy instance forwards the requests, via an encrypted WebSocket connection, to the emulated network environments running on a more powerful (cloud) machine using the EaaS framework. It also is responsible for session management by launching, via EaaS's public (RESTful) API, emulation sessions on the EaaS machine.

Apart from its configuration files, the gateway machine is completely state-less, meaning it can easily both be deployed and regularly be redeployed with new versions of the underlying operating system (in this case, Ubuntu 20.04 was used but the chosen operating system only has to support Docker and Docker Compose).

VII. Conclusion

Web services and especially instances created for project communication will eventually become a subject of preservation. In case of interactive services, e.g., blogs, wikis, or content management systems, a client-side crawler-based approach is usually not sufficient to capture their content and functionality.

We have developed a method and workflows which allow to retire unmaintained machines in phases, matching the requirements and demands of the services' communities. Security, user access and session management can be tailored accordingly. To achieve this, we use emulation as an access technology and extended the scope of emulation to the network domain. This way it is possible to re-republish a networked service, orchestrate and control all aspects of the emulated network.

Based on three real-life case studies, we have demonstrated both the feasibility of re-publishing web servers as well as convenient options for end-user access.

The proposed methodology and infrastructure are not just usable for services with a web front-end but are designed to keep any networked service available and conveniently accessible on-demand, allowing an "endless sunset phase" of retired networked services.

References

[1] A. Arvidson and F. Lettenström, "The Kulturarw project—the Swedish royal web archive," *The Electronic Library*, 1998.

[2] D. S. Rosenthal, "Emulation & virtualization as preservation strategies," *Andrew W. Mellon Foundation*, 2015.

[3] D. Gomes, J. Miranda, and M. Costa, "A survey on web archiving initiatives," in *International Conference on Theory and Practice of Digital Libraries*, Springer, 2011, pp. 408–420.

[4] G. Mohr, M. Stack, I. Rnitovic, D. Avery, and M. Kimpton, "Introduction to heritrix," in *4th International Web Archiving Workshop*, Citeseer, 2004, pp. 109–115.

[5] B. Kahle, "The internet archive," *RLG DigiNews*, vol. 6, no. 3, 2002.

[6] S. G. Ainsworth, M. L. Nelson, and H. Van de Sompel, "Only one out of five archived web pages existed as presented," in *Proceedings of the 26th ACM Conference on Hypertext & Social Media*, 2015, pp. 257–266.

[7] J. F. Brunelle, M. C. Weigle, and M. L. Nelson, "Archival crawlers and JavaScript: Discover more stuff but crawl more slowly," in *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, IEEE, 2017, pp. 1–10.

[8] S. Lawrence and C. L. Giles, "Searching the world wide web," *Science*, vol. 280, no. 5360, pp. 98–100, 1998.

[9] S. Raghavan and H. Garcia-Molina, "Crawling the hidden web," in *27th International Conference on Very Large Data Bases (VLDB 2001)*, 2001. [Online]. Available: `http://ilpubs.stanford.edu:8090/725/`.

[10] G. Alberts, M. Went, and R. Jansma, "Archaeology of the Amsterdam digital city; why digital data are dynamic and should be treated accordingly," *Internet Histories*, vol. 1, no. 1-2, pp. 146–159, 2017.

[11] K. Rechert, T. Liebetraut, O. Stobbe, N. Lubetzki, and T. Steinke, "The RESTful EMiL: Integrating emulation into library reading rooms," *Alexandria*, vol. 27, no. 2, pp. 120–136, 2017.

[12] R. Davoli, "Vde: Virtual distributed ethernet," in *First International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, IEEE, 2005, pp. 213–220.

[13] A. Melnikov and I. Fette, *The WebSocket protocol*, RFC 6455, Dec. 2011. doi: `10.17487/RFC6455`. [Online]. Available: `https://rfc-editor.org/rfc/rfc6455.txt`.

[14] E. Rescorla, *HTTP over TLS*, RFC 2818, May 2000. doi: `10.17487/RFC2818`. [Online]. Available: `https://rfc-editor.org/rfc/rfc2818.txt`.

[15] P. B. Menage, "Adding generic process containers to the Linux kernel," in *Proceedings of the Linux symposium*, Citeseer, vol. 2, 2007, pp. 45–57.

[16] E. W. Biederman and L. Networx, "Multiple instances of the global Linux namespaces," in *Proceedings of the Linux Symposium*, Citeseer, vol. 1, 2006, pp. 101–112.