Challenges and Solutions of Developing and Implementing a Revolutionary Novel Desktop-as-a-Service

Christian Baun and Johannes Bouché

Faculty of Computer Science and Engineering,
Frankfurt University of Applied Sciences,
Nibelungenplatz 1, 60318 Frankfurt am Main, Germany
[christianbaun|johannes.bouche]@fb2.fra-uas.de

Abstract. Desktop-as-a-Service (DaaS) is a cloud service category with great potential for improving the everyday work of running and using applications. However, existing DaaS solutions have several drawbacks, which have limited the popularity of this service category in the past. Our novel DaaS solution called DESIGN not only overcomes the limitations of existing solutions but also offers significant benefits. It has the potential to revolutionize the way we use applications, supporting non-modified Linux and Windows applications with just a browser on the client side. This paper describes some of the knowledge we gained during the development because of the challenges we faced and the solutions we found.

Keywords: DaaS · Compatibility · Performance · Stability · Usability

1 Introduction

This paper discusses challenges during the development and implementation of a novel Desktop-as-a-Service (DaaS) solution called DESIGN that enables the deployment and usage of unmodified Linux and Windows applications in the same way as web applications and can run inside public resources but can also be deployed in a private context. Since all interaction with DESIGN and the applications that reside inside is done via a user's web browser, the users can use any client, no matter what hardware or host operating system it is compromised of. A browser is the only software component required to use the novel DaaS service.

Undoubtedly, the development of a feature-rich DaaS system is a complex task, riddled with numerous obstacles and challenges. This paper meticulously addresses these challenges, categorizing them under compatibility, performance, stability, and usability. It not only brings these obstacles to light but also presents the solutions we devised to overcome them, along with the valuable lessons we learned in the process.

This document is structured as follows. Section 2 discusses related work on DaaS solutions from an academic perspective, whereas section 3 describes the

state-of-the-art DaaS architecture DESIGN we designed and implemented. Section 4 presents the most relevant challenges we faced, analyses our options for handling these, and describes the solution we found. Finally, section 5 discusses conclusions, and section 6 includes directions for future work.

2 Related Work

DaaS has been a well-known service category since the emergence of cloud computing. Still, it got much less attention in research and literature The challenges and obstacles when developing DaaS solutions, rather than infrastructure (IaaS) and platform services (PaaS), have seldom been discussed in the literature.

Celesti et al. [4] implemented in 2016 a DaaS using the IaaS solution OpenStack [17] and analyzed the characteristics and performance aspects of using noVNC, which is a client for the protocol Virtual Network Computing (VNC) that is implemented in the Hypertext Markup Language (HTML) and JavaScript, the protocol SPICE (Simple Protocol for Independent Computing Environments), and the remote desktop gateway Apache Guacamole as solutions for providing access to the desktop via a browser. One focus of the paper is the redirection of the sound interface of the virtual desktop. The paper's authors conclude that Guacamole, in combination with the Remote Desktop Protocol (RDP), is the best solution. The authors evaluated the lag time for remote audio playback in an Internet scenario. by measuring the time between the automated initiation of the live streaming using the media player software VLC and the arrival of the music data by analyzing network traffic via monitoring software Wireshark. When using Guacamole combined with the protocol RDP, the average lag was around 750 ms. When using Guacamole combined with the protocol VNC, the average lag was around 1750 ms. The paper also includes a short evaluation of the lag for video frames represented by background color changes of a Java application. Again, the configuration Guacamole and RDP offered the best performance video update with an average lag of around 300 ms. The configuration of Guacamole and VNC caused an average lag of around 500 ms.

Magana et al. [15] compared in 2019 the most popular remote desktop protocols and software implementations. The authors analyzed five protocols: PCoIP used in the Amazon WorkSpaces, RDP, TeamViewer, VNC, and Citrix Independent Computing Architecture (ICA). In the paper, the network transfer rate and its relation to the quality experienced by the DaaS user are evaluated by assuming three scenarios: using an office software suite, web browsing, and video streaming. One emphasis of the paper is comparing the transfer rate of the different protocols for three scenarios. In the Office Software Suite scenario, the transfer rate is measured and compared when performing several different tasks like opening and editing a document (typing), loading an image, and saving a document. The web browsing scenario covers requesting three different web pages, and for the video streaming scenario, the same YouTube video file was viewed at different resolutions. The authors conclude that RDP requires fewer downstream and upstream network resources for all three scenarios. It is impor-

tant to understand that the protocols RDP and VNC are implemented in open-source projects and proprietary solutions for many different operating systems. Depending on the different quality levels of the many existing implementations, it is difficult to generalize about the performance of the protocols, as individual implementations may have better or worse performance characteristics than expected.

Several works in literature have analyzed the latency of Cloud gaming services that have, in some aspects, similar requirements to DaaS offerings. According to Lampe et al. [13], the latency of locally executed games is compromised of the following components:

- Input lag caused by the controller device (e.g., mouse or keyboard).
- CPU time for processing the user input and the program itself.
- GPU time of the graphics processor for rendering the next frame.
- Time required for delivering the frame into the frame buffer.
- LCD response time (time required to display the frame on the screen).

In cloud gaming scenarios, the following components increase latency further:

- Transfer time (upstream) for sending the user input to the service provider.
- Time required for capturing the frames and encoding it as a video stream.
- Transfer time (downstream) for sending the video stream to the client.
- Time required for decoding the video stream back into a frame.

For every deployment scenario or cloud service category possible, users expect a latency value that is not annoying when interacting with the service and its applications. Choy et al. [5], Claypool et al. [6], and Jarschel et al. [12] consider a maximum tolerable latency, based on subjective tests, being around 100 ms. This relatively low value sets the demand for a fast network interaction between client and service and limits their maximum distance. As mentioned in Clincy et al. [7], The former cloud gaming service provider OnLive specified in the year 2010 that the distance between the service provider and the consumer should not be greater than 1000 miles (approx. 1600 km) because otherwise, the round-trip communications delay time between will be too long for video games.

On first thought, the latency requirements of a modern computer game are hardly the same as those of a desktop application. However, just as with computer games, the lagging operation of desktop programs has a negative impact on the user experience, and with video conferencing or other graphics-intensive applications, the utilization of resources and latency requirements are very similar to computer games.

3 Architecture

Figure 1 shows the architecture of our novel DaaS solution DESIGN that can run unmodified Linux and Windows applications in Linux containers (Linux and

Windows applications compatible with Wine) and virtual machines (Windows applications incompatible with Wine).

The open-source server virtualization platform Proxmox VE runs bare-metal and offers virtual machines with the Kernel-based Virtual Machine (KVM) and container-based virtualization via LXC Linux container. Since the LXC API is, in comparison to the Docker API, rather poorly documented and lacks several features, we considered Docker a better container virtualization solution. A deployment of docker directly in the host-operating system or inside a virtual machine is both possible.

Exporting the graphical user interface of Linux and Windows applications, is possible using the a protocols VNC or RDP. Free server implementations exist for both operating system families. Since the focus of our DaaS solution is to export only the application's graphical user interface and not full desktops – a feature not all server implementations include – is the number of available projects and solutions limited. For Linux containers, the free software projects x11vnc and Xrdp can be used in the DESIGN DaaS. For Windows VMs, the free software TightVNC and the Windows-internal RDP server can be used.

One of the main goals of DESIGN is to allow all interactions to be carried out solely by using a browser. Thus, a broker or proxy software that mediates between the VNC or RDP implementation and the browser is required. The most advanced solution available is the free software project Apache Guacamole which supports VNC and RDP both and includes further relevant features regarding sound and printing.

The core components of DESIGN are our self-developed Web UI and the Control Application, which carry out the communication between all service components.

When starting a virtual machine or container instance, it automatically includes an SSH service. Our self-developed instance component waits until the SSH service is ready to satisfy requests. In case applications are to be started or modifications are to be carried out, the instance component sends these requests to the virtual machine or container. Modification requests include resolution changes and fetching host status information.

In general, the self-developed instance component allows users to carry out essential administration and usage tasks that the operating system does not offer via a generalized API.

4 Challenges and Solutions

While developing the novel DaaS platform DESIGN, we faced multiple obstacles and challenges. Each of these issues belongs to one of these four categories: compatibility, performance, stability, and usability. This section presents the issues we were confronted with, the possible approaches we analyzed, and the solutions we chose.

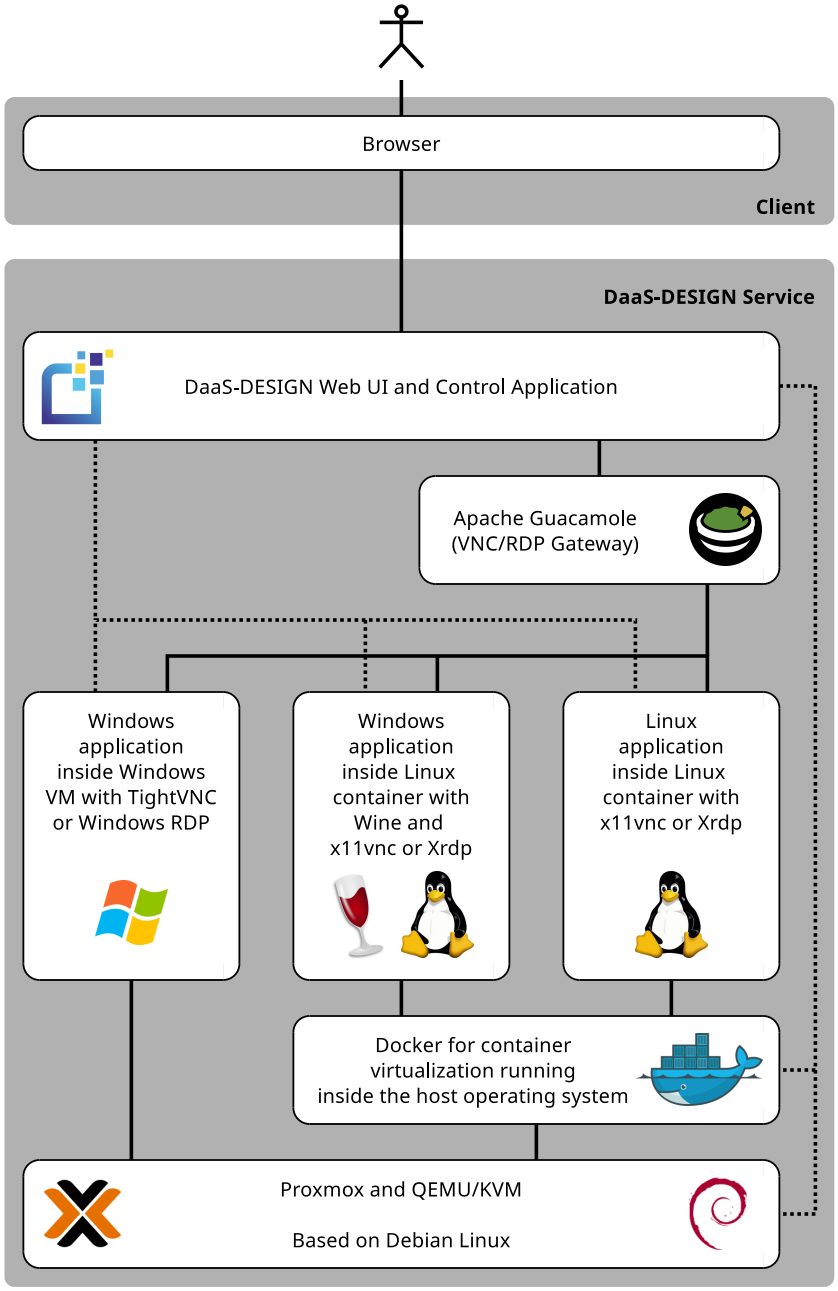

Fig. 1. Architecture of the novel DaaS DESIGN [2].

4.1 Compatibility

The first relevant characteristic of a DaaS that a user is confronted with is the list of compatible applications. In the best case, all applications of all popular operating systems can be integrated and used with such a novel platform. Several DaaS projects and solutions from the last two decades and similar concepts like web desktops or webtops lacked native support for popular applications.

Examples worth mentioning in this context include eyeOS and SilveOS. The project eyeOS [14, 22] which simulates a desktop environment using HTML, the scripting language PHP, JavaScript, and the MySQL database management system. The project SilveOS [9] mimics the look and feel of a Windows desktop in the browser using the discontinued application framework Microsoft Silverlight. Despite being free software, both solutions have never become widely used since it is impossible to import and use native Windows or Linux applications there. All required applications have to be developed as web applications. This limitation caused most potential users to lose interest in web desktops because many popular native applications can hardly be replaced or re-developed, and many feature-rich and well-established Software-as-a-Service (SaaS) offerings like Microsoft Office 365 and Google Docs already exist.

A new DaaS is only accepted by a broad user group if it enables the integration of native applications in an unchanged manner. It is also desirable that a DaaS supports Linux and Windows applications both to enable flexible working and make the operation of a virtualization solution on the desktops unnecessary. Ideally, it should also be possible to run native Mac OS applications. However, this goal is difficult to achieve under legal conditions due to the Mac OS operating system's licensing restrictions.

Further essential features for most users are printing, sound, and compatibility with external storage media.

Guacamole supports redirecting audio, printing, and disk access via the protocol RDP. The printing feature does not allow access to local or remote printers; instead, it allows users to print directly to PDF files, which are received by the user's web browser. As a prerequisite, the GhostScript interpreter must be deployed on the Guacamole server. File transfer over RDP supports Guacamole by emulating a virtual disk drive that persistently stores data on the Guacamole server. [10]

Compared to RDP, the protocol VNC offers fewer features in Guacamole since it does not support sound on its own. However, Guacamole supports sound using PulseAudio[1] as a workaround solution. VNC also does not offer file transfer and access to external storage media, but as a workaround solution, Guacamole supports file transfer file transfer via the SSH File Transfer Protocol (SFTP) which is an extension of the Secure Shell protocol (SSH).

[1] On Linux operating systems, sound and microphone devices are accessed via sound servers like PulseAudio and PipeWire, which interact with the sound interface present in the kernel, e.g., Advanced Linux Sound Architecture (ALSA) or Open Sound System (OSS)

4.2 Performance

To describe the mission-critical performance aspects of our system, we initially give a short overview of potential challenges, describe bottlenecks as well as other problem areas. We do that concerning the underlying use case model of DaaS-Systems and then follow with a more thorough description of our own specific design decisions and particular solutions.

From our perspective, performance, in general can be seen as the most crucial system property which influences a majority of all other other relevant properties, such as user experience, system responsiveness, and overall efficiency. As a result, we had to address multifaceted and potentially cross-concerning challenges on several distinct layers within our architecture.

From the core infrastructure to backend services and user-facing elements, each component individually has to achieve specific minimum requirements in its domain and to contribute to the overall performance goals of the whole system. Hence, we outline key limiting factors that impede performance, along with our experiences and tailored solutions aimed at mitigating these challenges. From our point of view, the three most critical performance aspects for Daas-Systems are:

- Infrastructure and Platform
- Network Latency and Bandwidth
- Internal Processing and Communication

As there is no generalized measurement system available, we ordered them by their magnitude of impact and based them to the best of our knowledge on the obtained experiences during our research project. Since each of them has their own characteristics, specific behavior and requirements, implying a potential impact on the overall system performance and resource utilization, we describe them individually and in more detail in the following separate subsections.

Infrastructure and Platform In general, it can be said that the chosen infrastructure, platform, and utilized software may have the most influential impact on CPU and resource consumption which has a dramatic impact on the overall system performance. As a DaaS combines all aspects of IaaS, PaaS, and SaaS into one system each related performance characteristic has to be taken into account.

For Infrastructure-as-a-Service can be said that our solution generally offers possibilities to use either virtual machines or docker containers. As can be seen in several scientific publications discussing relevant performance aspects of such systems, the obvious advantage in most categories has a lightweight container solution compared to the virtual machine if an identical program was tested in both environments [8, 16, 18].

On the one hand, that implies for our solution, that a container-based approach is most likely preferable, if that choice is available. On the other hand, that choice is not always available, as not every app can be run in a container-based environment. To maximize coverage of possible apps being able to run

on our system, we offer container-based and virtual machine-based application hosting as a critical concept of our solution.

As a second observation in the area of PaaS, it can be added that in most studies, there is no dramatic difference if derivates of operating systems are compared to each other. Nevertheless, there are derivates which are more driven towards being lightweight and more performant [1, 3]. In cases where a Windows virtual machine can be replaced with a natively operating wine platform, additional performance potentials can be leveraged [11]. In contrast to that, if the operating systems of different types are compared to each other, there are more obvious results especially when taking into account implementations of low-level operating system functionality, such as interrupt handling, memory allocation, driver-specific implementations or other more subtle requirements such as boot time or the time needed for maintenance or system updates [19, 20].

As a result of our research, we aim to support any Windows or Linux-based operating system in virtual machines but can only support Linux and Wine in containers This is because only terminal-based Windows containers are supported by Microsoft.

Network Latency and Bandwidth Another essential aspect when providing web-based access to DaaS apps is that also network latency and bandwidth play a significant role which contributes to a large degree to the responsiveness of the system directly experienced by the user. As the user utilizes its application through a web browser we aim to support standardized frame-buffer protocols to enable user interaction and access to external devices such as printers or USB sticks.

Within such protocols and in cases where video content is transmitted, the provided network bandwidth might be the main bottleneck especially when it comes to high-resolution video streams. In cases where audio, mouse movement, or keyboard strokes are transmitted, the network latency plays a more critical role. Therefore, the experienced responsibility and usability crucially depends on the hosted application.

In general, such application-specific requirements can, to a certain degree, be controlled by protocol-inherent tuning parameters and utilized compression algorithms. Therefore, our solution aims to maximize coverage of potential use cases by offering several solutions and a set of appropriate tuning options which intentionally forces the user to decide which solution fits best for him.

For that purpose, we utilize the desktop gateway implementation Guacamole, which offers support for the most utilized frame-buffer protocols, VNC and RDP as well as all relevant tuning options. Guacamole not only implements out-of-the-box support for the protocols itself but also adds the ability to integrate audio support for VNC or other external devices as well as a JavaScript-based web client.

Nevertheless, to integrate guacamole into the system, there are still things to consider that are not handled by guacamole but would be assumed from a full-fledged DaaS solution.

First and foremost, how authentication is handled must be considered. On one hand, from a DaaS system, a user would expect the system to work without forcing him to authenticate himself to multiple authentication systems and only once per session. On the other hand, crucial information such as user passwords might, in general, better be kept secret from the system so that users can not manipulate the provided system in an unintended way. A potential problem arises if it is considered that, by default, RDP requires a valid user session to the system.

For that reason, guacamole was integrated with a proxied approach using proxied websockets. This enables keeping the credentials of the generated system secret to the host system while still providing authorized but passwordless access to the hosted application.

Further, the proxy mechanism enables the support of dynamic desktop resizing which is a default guacamole feature for RDP, but which is not available in VNC through guacamole. By intercepting WebSocket opcodes and listening for size requests, the requested resolution can then be enforced by directly interacting with the related container or virtual machine.

Both changes to the default guacamole use case are mandatory to comply with the requirements for the underlying DaaS design but both also impact the overall system performance and responsiveness in a negative way. From a network perspective, the integration as a proxy adds network hop and leads to a noticeable delay in network communication. The protocol-level evaluation of opcodes additionally enhances that delay, and perceived responsiveness might suffer if resize requests take more time than expected. Overall, the introduced overhead is still acceptable and enables platform-independent and seamless access for all defined DaaS use cases.

Internal processing and Communication Another critical aspect directly related to performance and responsiveness is the approach in which data is internally processed, stored, and communicated to all involved components and instances. This includes the metadata being collected and processed by essential components but also implies raw physical data being directly exchanged with object instances.

To provide a satisfying user experience in a DaaS system specific components in a host system are required. This includes access to all backend services being used to create containers and virtual machines, a local or remote database, a local or remote authentication service, local and distributed filesystems, as well as virtual or physical network devices. Further, a web server is integrated to provide all relevant API endpoints for the frontend communication as well as all WebSocket endpoints for viewer communication.

Generally spoken, all of these components might become a potential bottleneck when being utilized with high throughputs or large amounts of concurrent applications demanding low latencies. This is especially the case if being used in a sequential manner or being served from the same hardware. If delays add up, the total time needed for a request to be fully processed can increase drastically.

On an architectural level, all relevant components are, therefore designed for being distributed to different hosting systems if the requirements of a particular user scenario suggest it. Nevertheless, it might still be advisable to keep specific components directly on the host system to ensure low latency component access.

For all components with real-time and low-latency latency usecases, it is generally advisable to keep them as close to the core hosting system as possible. This includes all data and communication relevant to audio, video, mouse, and keyboard transmission, but also the backend service pertinent to host containers and virtual machines. That might be in the best case to host them natively on the host itself which enables virtual network and hardware access. In the worst case, that might be at least connected to the same physical network to reduce network latencies as much as possible.

For all other components of the overarching nature such as database access, authentication provider, and filesystems it is still advisable to keep a close distance from the host but it also highly depends on the total amounts of users and the specifics of their use cases. On small scales, it is still advisable to keep such components on the same system but on larger scales, it is preferable to externalize such functionality to dedicated systems.

4.3 Stability

Stability in DaaS applications is of significant importance to ensure that our solution can meet the demands of continuous and time-critical operations across varied user scenarios. From our point of view, any DaaS system must ensure the stability of all system components during runtime to maintain the ability to provide service continuity, handle growing user loads and transparently recover from failures, while still providing a seamless end-user experience. Such considerations are an integral part of our architectural concept as they directly impact the operational integrity of the system and its hosted instances, making them suitable for enterprise-level deployment and trust.

This definition of stability is underpinned by several architectural and operational strategies, including redundancy of components, scalability of nodes, and a robust backend service framework providing means of communication between components and nodes. These strategies, as well as their drawbacks and benefits will be further discussed on a component level and within the following subsections.

Scalability of Components Scalability in general, but also specifically in DaaS environments can be approached in two fundamental ways: Horizontal and vertical scaling. Horizontal scaling, or scaling out, involves adding more machines or instances to a pool to manage the increased load. In contrast to that, vertical scaling or scaling up, refers to adding more power (CPU, RAM) to an existing machine [21].

While horizontal scaling is generally more flexible and often preferred in cloud environments due to its ease of integration with existing architectures,

vertical scaling remains important for applications requiring strong single-thread performance or low latencies.

To comply with both, we defined means of configuration such that the system itself, as well as all of its backend services are designed to be split up into logical units which can be distributed to dedicated systems. Essentially, all backend services receive their dedicated configuration and can be accessed via tuples of hostname and port. Any component using such a backend service can obtain the relevant tuple and then use it to fetch requests from that service. By default and in a standalone configuration, all backend services reside on the same native host, and relevant tuples can be derived from the host configuration. In a distributed context, a selection of these services can be distributed to a dedicated and remote system to comply with horizontal scaling principles. The relevant tuple can then be derived from any node in the system by utilizing a common database containing the relevant meta information.

Vertical scaling of DaaS nodes is currently not planned but administrators are free to use custom 3rd party toolchains, e.g., for GPU virtualization. Although vertical scaling on a host basis is beyond the scope of our research project, it is more plausible to scale the instances being hosted within that system and for both dimensions.

For vertical scaling of instances, appropriate parameters and API endpoints are present to manipulate the configuration of their resource assignments, such that arbitrary configurations are possible.

For horizontal scaling of instances the builtin mechanisms of each backend service are automatically configured and utilized such that instances can be arbitrarily distributed across a pool of connected nodes. In the case of virtual machines, the responsible backend service Proxmox implements such mechanisms as part of its 'Proxmox VE High Availability (HA)' cluster. In the case of containers, any backend service fulfilling similar concepts can be integrated, for example, with Kubernetes. Within such clusters, any DaaS instance is entirely agnostic to its hosting system and must only be reachable via a known hostname or IP.

Redundancy of Components Redundancy is critical for most distributed systems but especially for DaaS architectures, and serves as a safeguard against service interruptions and data loss by duplicating essential components, functions, and data. This design principle ensures that systems remain highly available and reliable, even during component failures. In the following, a list of components eligible for redundancy is given which briefly describes the most relevant strategies contained in our architecture.

- **Frontend/Backend:** In our final version redundancy of the frontend and backend components will offer means of configuration such that multiple instances can be distributed to dedicated systems and can be the target of standardized load-balancing strategies. By doing so, the frontend or the backend may act as a gateway between the user and the backend service containing the relevant instances. In that manner, redundancy is achieved,

and frontend or backend can forward requests and responses by identifying the correct communication endpoint based on URL's or specific parts of them, for example, by extracting hostnames or specific identifiers being unique within the DaaS domain.

- **Database:** The database is integrated as a backend service and as such, completely unaware of surrounding components and contexts in which it is used. The database can, therefore, be distributed to a dedicated system if needed. Generally, any modern database solution (e.g., MariaDB) implements rich features to maintain specific data consistency strategies such as data replication, clustering, dedicated failover mechanisms or also RAID control. Therefore, redundancies in our database system are currently not planned, but can easily be applied by users with such needs and by using database-specific features and a modified database adapter.
- **Authentication:** Authentication within our architecture is also implemented as a backend service and can, therefore, be distributed to any dedicated system, can be a target of standardized load-balancing mechanisms and strategies and is as well fully agnostic of surrounding components and calling contexts. The authentication system maintains its credential registry in a separate database, a customizable set of user and group privileges, and implements the well-defined OAuth 2.0 standard to allow remote authentication. A customizable amount of redundant systems might be configured in our final version.
- **Distributed Filesystem:** Within our architecture we use the distributed filesystem Ceph, which is a distributed storage system providing object, block, and file storage. In Ceph, redundancy is achieved through its self-healing and self-managing capabilities utilizing an algorithm called CRUSH (Controlled Replication Under Scalable Hashing). Data availability and fault tolerance are obtained by automatically managing replication of data pieces across different devices. Our designed architecture uses these capabilities to reliably provide file storage to host systems and all hosted DaaS instances and provides block storage to the infrastructural backend services. By hosting DaaS instances directly from the distributed block storage, arbitrary redundancy scenarios might be covered by using specific configuration parameters.
- **Instances:** Redundancy of instances It is not a necessary condition for DaaS systems in general, but applying such strategies under certain conditions might make sense. By using highly available clusters within our backend services redundancy can be achieved by using the corresponding API endpoints previously described and by parameterizing them appropriately.

Internal Communication After discussing the principles being used to achieve scalability and redundancy it adheres to describe how communication flows are maintained in such a distributed environment.

Our proposed solution is, at the core, designed as a message-oriented architecture which additionally aims to fulfill DaaS properties for scalability and redundancy as previously described.

That implies fundamentally that any action being triggered, may it either be in the frontend on the client side, in the backend on the host side or within any of the hosted instances must always fulfill the condition that the required information to execute that action must either be directly derivable from a request or a corresponding response or otherwise, must be immediately available to the component executing that action.

We achieve that by storing all runtime information of instances in the distributed block storage, by storing all relevant configuration files in our distributed file storage, by utilizing a common database to persist all object metadata and by using our decentralized OAuth component to handle authentication.

By doing so, any change being triggered by a node within the DaaS network is immediately available to any other node in the system which relies on that information change. Apart from that, any additional information needed by a component is fully contained within the related request or response.

Therefore, arbitrarily distributed configurations are possible as long as request and response messages reach their desired target, eventually and as long as the availability of all relevant backend services is guaranteed.

If that is the case, each distributed DaaS node can typically just contain all time-critical components at a bare minimum to maintain an acceptable performance throughout the whole user-facing network stack. This would typically include the backend service hosting the relevant instances as well as the WebSocket proxy directly connected to the guacamole daemon and all other components may reside somewhere in the DaaS network.

4.4 Usability

An essential feature of a DaaS platform is its strong user-friendliness, which eliminates the complexity of the different operating systems used inside virtual machines and containers, makes the underlying hardware transparent, and simplifies the necessary interactions as much as possible.

Since users already have an operating system on their clients, and the motivation for using a DaaS is the simple use of various applications, it makes no sense for most users to have additional desktops available in the browser. The focus should be on the applications. For this reason, when developing a new DaaS solution, the main goal should be to reduce the graphical output to the limits of the respective application. The concept uses a separate browser tag for each application whose content exclusively represents the application.

Developing the automatic adaptation of the output to the limits of the respective application window is challenging. Only a few RDP and VNC server implementations offer to export the graphical output of only individual windows or processes. For Windows operating systems, the proprietary built-in RDP server and the free software TightVNC offer the export of full desktops and single windows. For Linux operating system deployments, the free software projects xrdp and x11vnc enable all desired features. Table 1 includes an overview of the mentioned service implementations for RDP and VNC protocols and their relevant characteristics. Further implementations like TigerVNC and protocols like NX

NoMachine exist but do not seem useful for designing and implementing a novel DaaS because they do not meet all feature requirements. [2]

Table 1. Services used by our DaaS for Remote Access to individual Linux or Windows Applications

Service implementation	**Protocol**	**Single Window Mode**	**Operating System**	**Software License**
TightVNC	VNC	supported	Windows	GPL2
Windows RDP server	RDP	supported	Windows	proprietary
x11vnc	VNC	supported	Linux	GPL2
Xrdp	RDP	supported	Linux	Apache 2.0

The automatic adaptation of browser tabs content size, is done by.... TOTO: Johannes´...

TODO: Ideen und Konzepte zur Trennung von User und Admin-Ansicht

TODO: Herausforderungen, Entwicklung, Implementierung, Lessons-learned

TODO Johannes: Erlebnisse der letzten 3-4 Wochen aufschreiben

5 Conclusion

TODO: Zusammen am Schluss

6 Outlook

TODO: Zusammen am Schluss

Acknowledgements

This work was funded by the Federal Ministry for Economic Affairs and Climate Action ('*Bundesministerium für Wirtschaft und Klimaschutz*') in the framework of the central innovation programme for small and medium-sized enterprises ('*Zentrales Innovationsprogramm Mittelstand*').

We thank our project partners from Nuromedia GmbH for their support. We especially thank Björn Goetschke, Mike Ludemann, Dario Savella, Holger Sprengel, and Rahul Tomar.

References

1. Balen, J., Vdovjak, K., Martinović, G.: Performance evaluation of windows virtual machines on a linux host. Automatika: časopis za automatiku, mjerenje, elektroniku, računarstvo i komunikacije **61**(3), 425–435 (2020)

2. Baun, C., Bouche, J.: Closing the gap between web applications and desktop applications by designing a novel desktop-as-a-service (daas) with seamless support for desktop applications. Open Journal of Cloud Computing (OJCC) **8**(1), 1–19 (2023)
3. Boras, M., Balen, J., Vdovjak, K.: Performance evaluation of linux operating systems. In: 2020 International Conference on Smart Systems and Technologies (SST). pp. 115–120. IEEE (2020)
4. Celesti, A., Mulfari, D., Fazio, M., Villari, M., Puliafito, A.: Improving desktop as a service in openstack. In: 2016 IEEE Symposium on Computers and Communication (ISCC). pp. 281–288. IEEE (2016)
5. Choy, S., Wong, B., Simon, G., Rosenberg, C.: The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In: 2012 11th Annual Workshop on Network and Systems Support for Games (NetGames). pp. 1–6 (2012). https://doi.org/10.1109/NetGames.2012.6404024
6. Claypool, M., Claypool, K.: Latency can kill: precision and deadline in online games. In: Proceedings of the first annual ACM SIGMM conference on Multimedia systems. pp. 215–222 (2010)
7. Clincy, V., Wilgor, B.: Subjective evaluation of latency and packet loss in a cloud-based game. In: 2013 10th International Conference on Information Technology: New Generations. pp. 473–476. IEEE (2013)
8. Felter, W., Ferreira, A., Rajamony, R., Rubio, J.: An updated performance comparison of virtual machines and linux containers. In: 2015 IEEE international symposium on performance analysis of systems and software (ISPASS). pp. 171–172. IEEE (2015)
9. Garmpis, A., Gouvatsos, N.: Design and development of webubu: An innovating web-based instruction tool for linux os courses. Computer Applications in Engineering Education **24**(2), 313–319 (2016)
10. Guacamole, A.: Configuring guacamole (2024), https://guacamole.apache.org/doc/1.5.5/gug/configuring-guacamole.html
11. Huang, C., Chen, J., Zhang, L., Luo, Q.: Performance evaluation of virtualization technologies for windows programs running on linux operating system. In: International Conference on Network Computing and Information Security. pp. 759–766. Springer (2012)
12. Jarschel, M., Schlosser, D., Scheuring, S., Hoßfeld, T.: An evaluation of qoe in cloud gaming based on subjective tests. In: 2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. pp. 330–335. IEEE (2011)
13. Lampe, U., Wu, Q., Dargutev, S., Hans, R., Miede, A., Steinmetz, R.: Assessing latency in cloud gaming. In: Cloud Computing and Services Science: Third International Conference, CLOSER 2013, Aachen, Germany, May 8-10, 2013, Revised Selected Papers 3. pp. 52–68. Springer (2014)
14. Liu, K., Dong, L.j.: Research on cloud data storage technology and its architecture implementation. Procedia Engineering **29**, 133–137 (2012)
15. Magana, E., Sesma, I., Morato, D., Izal, M.: Remote access protocols for desktop-as-a-service solutions. PloS one **14**(1), e0207512 (2019)
16. Potdar, A.M., Narayan, D., Kengond, S., Mulla, M.M.: Performance evaluation of docker container and virtual machine. Procedia Computer Science **171**, 1419–1428 (2020)
17. Sefraoui, O., Aissaoui, M., Eleuldj, M., et al.: Openstack: toward an open-source solution for cloud computing. International Journal of Computer Applications **55**(3), 38–42 (2012)

18. Seo, K.T., Hwang, H.S., Moon, I.Y., Kwon, O.Y., Kim, B.J.: Performance comparison analysis of linux container and virtual machine for building cloud. Advanced Science and Technology Letters **66**(105-111), 2 (2014)
19. Sergeev, A., Rezedinova, E., Khakhina, A.: Docker container performance comparison on windows and linux operating systems. In: 2022 International Conference on Communications, Information, Electronic and Energy Systems (CIEES). pp. 1–4. IEEE (2022)
20. Sulaiman, N.S., Raffi, A.S.H.A.: Comparison of operating system performance between windows 10 and linux mint. International Journal of Synergy in Engineering and Technology **2**(1), 92–102 (2021)
21. Vaquero, L.M., Rodero-Merino, L., Buyya, R.: Dynamically scaling applications in the cloud. ACM SIGCOMM Computer Communication Review **41**(1), 45–52 (2011)
22. VidyaBanu, R., Preethi, J., Dinesh, N.: Implementation of financial system using EyeOS in the cloud environment. In: 2011 International Conference on Recent Trends in Information Technology (ICRTIT). pp. 656–660. IEEE (2011)