Identity Provider Deployment Based on Container Technology

Marko S. Eremija, Nebojša R. Ilić, Miloš Cvetanović, Jelica Protić, and Zaharije Radivojević

Abstract—**Identity Providers are an integral part of Identity Federations. Many different and complex technologies are needed to create an Identity Provider. In order to be able to fully utilize all the benefits of Identity Federations, adequate hardware resources are needed for Identity Provider deployment. Containers address the complexity and resources issues, while enabling faster deployment and keeping the functionalities and core concepts intact at the same time. Containers cannot be perceived as a replacement for virtual machines or bare metal servers, as they are meant to co-exist and have already found a wide range of use cases. This paper proposes using containers for easier implementation of Identity Providers, while lowering resource usage and complexity imposed by deployment requirements.**

Keywords — **containers, Docker, Identity Federations, SAML, Single Sign-On.**

I. Introduction

SCIETIFIC collaboration has shifted its focus towards using web services. New and emerging technologies, like federated web services are being adopted very fast. Federated services get their name from the fact that they belong to an Identity Federation. Each Identity Federation consists of institutions that can undertake a role of Identity Provider (IdP) or Service Provider (SP), with underlying authentication and authorization infrastructure (AAI) based on Security Assertion Markup Language (SAML). SAML[1] software component is required to be present on both IdP and SP. Example of one community that relies on using Identity Federations is GEANT association, which interconnects different Research and Education (R&E) institutions. In order to help researchers from different R&E institutions to conduct their work, GEANT association has developed eduGAIN [2] service. eduGAIN consists of around four thousand different entities (IdPs and SPs) and is globally accessible.

Paper received May 9, 2018; accepted Setmebr 10, 2018. Date of publication December 25, 2018. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Miroslav Lutovac.

This paper is a revised and expanded version of the paper presented at the 25th Telecommunications Forum TELFOR 2017 [11].

This work was supported by the Ministry of Education, Science, and Technological Development of the Republic of Serbia [III44009, TR32047].

Marko S. Eremija, Miloš Cvetanović, Jelica Protić and Zaharije Radivojević are with the School of Electrical Engineering, University of Belgrade, Serbia (e-mail:marko.eremija@gmail.com, cmilos@etf.bg.ac.rs, jeca@etf.bg.ac.rs, zaki@etf.bg.ac.rs).

Nebojša R. Ilić is with AMRES, Bulevar kralja Aleksandra 84, 11000 Belgrade, Serbia; (e-mail: nebojsa.ilic@amres.ac.rs).

The main goal of this paper is to introduce a methodology that can help with better hardware manipulation by using containers and enables faster and easier deployment of IdPs. It is based on the experiences gained during the Identity Federation development and deployment in Academic Network of Serbia and during different projects inside GEANT community.

The paper is organized as follows. Section II contains problem formulation and gives a more detailed description of Identity Federations. Proposed method for creating containers is presented in Section III. Section IV concludes the paper.

II. Identity Federations and Federated Services

When talking about Identity Federation, the concept behind it implies using several different software technologies as well as providing hardware resources that provide foundation for the software part.

Researcher’s home institution usually has a role of IdP in Identity Federations. Home institution is responsible for providing its users with a valid digital identity, or in other words, valid credentials (username and password pair). User registry, e.g. Active Directory (AD), Lightweight Directory Access Protocol (LDAP), Structured Query Language (SQL) database, holds users’ credentials and different attributes that describe a user, including but not limited to: user’s name, affiliation, e-mail etc. Each of these user attributes can later be used by an SP for authorization and granting access to online resources and services. Even though it is not unusual for an institution to be both IdP and SP, most of the time a different institution is undertaking the role of the SP.

Creating and managing AAI infrastructure is a challenge and only when there is a proper AAI infrastructure is in place one can think about interconnecting IdPs and SPs. More details on best practices for AAI can be found in [3]. Federated services can have a local scope, e.g. services that support SAML offered at a campus, university, country, geographical region level etc. If there is a need, these “local” services can be connected to eduGAIN. One exception is when researchers are participating in an international project which requires using a web service that’s capable of understanding SAML, but is not Identity Federation-centric type of service. That also means that these services do not have to be connected to eduGAIN, but all the benefits of a SAML web service are available.

Each AAI system should contain at least one backend database (DB) or a user registry which is used to store users’ credentials. DB is IdP's responsibility and in rare occasions it is outsourced. In the light of new GDPR EU regulations [4], more changes are expected in the future. Besides keeping the username and password pair, there are also other attributes describing a user that are stored. Subject to LDAP schemas being used, some attributes like email and username are intended to verify user's identity, while others, e.g. user's affiliation can be used to authorize users and grant them access to a web service on the SP side. It is important to stress the role of IdP in an Identity Federation. IdP has to keep all the user data up to date in order to avoid unauthorized access to a service offered by the SP. Federated services use an already widely deployed concept by implementing Web Single Sign-On (SSO) mechanism. Web SSO mechanism has been used for Google, Facebook, GitHub and other web services. They can all be considered as IdPs in this aspect. OAuth[5], OpenID[6] or SAML can all be used to this end. This paper focuses only on solutions that implement SAML. Web SSO enables users to login to more than one web-based service, during the same browser session, using the same username and password combination, without the need to retype it. This concept is easily adopted because it is user friendly. Although using the service is significantly simplified on the user side, keeping sensitive data safe is done on the backend side. SAML messages are transported over the network using Hypertext Transfer Protocol (HTTP) or its secure version (Hypertext Transfer Protocol Secure). Security and data protection mechanisms used in Identity Federations are out of the scope of this paper, but they have to be mentioned to be able to fully comprehend the complexity they bring into this type of system.

Fig. 1 gives a high-level overview of steps needed for a user to be able to use a federated web service:

1. User tries to access the federated service;
2. SP redirects user to authenticate at its IdP;
3. IdP verifies user's identity and notifies SP;

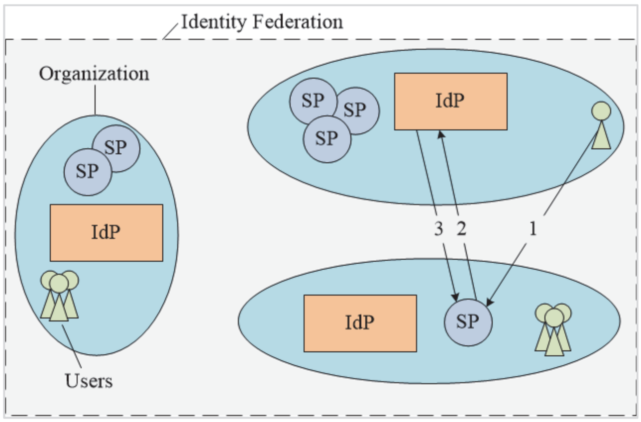

Fig. 1. Identity Federation entities.

As it was already mentioned, the service that uses SAML can have a local scope (left hand side of Fig. 1). If a service has a wider scope, e.g. it is made available at university level, users from different institutions can use that service, as displayed on the right hand side of Fig. 1.

Nevertheless, it is not only the complexity of these technologies combined that is hindering the progress of federated services. It has also been observed in R&E community that hardware resources can significantly impact the rate at which institutions are joining Identity Federations, regardless of scope (local or global) and role they want to undertake (IdP or SP). Some new and emerging technologies, like containers and automation tools, can help with hardware resource facilitation, while applying all the best practices for federated services.

III. Creating and Using Containers

In order to have a better overview of all of the components of a federation (or Identity Federation entities), IdPs and SPs are usually decoupled and run on separate hardware. Most of the time, servers are realized as Virtual Machines (VMs), but there are cases when standalone (or bare-metal) servers are used. As it was mentioned before, the complexity of these technologies combined grows exponentially. If one VM or one bare-metal server is used for every entity (IdP, SP), it can be concluded that the hardware resources needed for deploying an IdP/SP cannot be neglected, especially if adoption of web services is of strategic importance for an academic network.

Considering the fact that hardware resources are both expensive and take up a lot of space in data centers, there is a tendency in recent years aimed at decreasing hardware resource usage. Most of the networking and data center technologies have been oriented towards virtualizing some functionalities and adapting faster to network topology changes (e.g. Software Defined Networking [7]), as well as enabling better resource collocation. When talking about servers, one of the first attempts to enable better and more reasonable resource distribution was through using software virtualization platforms like Hyper-V, vSphere, Xen etc. Even though virtualization brings much better utilization of hardware resources like Central Processing Unit (CPU), Random Access Memory (RAM), and Hard Disk Drive (HDD) space, it is still heavily constrained by the hardware itself.

VMs and bare-metal hardware require a dedicated operating system (OS). Standalone servers and VMs cannot share resources; they are reserved permanently, or in case of a VM, until it is removed from the virtualization platform and resources are returned to the resource pool. One of the advantages of containers is that they can be used on bare-metal servers as well as on VMs to share resources, relying on the OS that is already present and a corresponding container engine (e.g. Docker).

Resource sharing implies sharing all the available hardware resources (CPU, RAM, storage) and also software resources of underlying OS (e.g. kernel, networking etc.). That further means that the number of services running on top of containers that can be deployed on the same OS, even though limited, is at a much larger scale than in any other scenario. Table 1 enlists the differences in requirements between bare-metal servers, VMs and containers.

This paper focuses on using containers on Linux OS. It is worth mentioning that containers can also be deployed on other OSs like Windows and Mac OS.

If containers are used on VMs, they should not be considered as VMs on top of VMs - they are just processes running on the host OS (Fig. 2). In order for a container to be able to fully utilize OS environment, it is best to run containers based on OS images that share the same characteristics, e.g. CPU architecture, kernel version etc.

TABLE 1: DIFFERENCES BETWEEN BARE-METAL, VMS AND CONTAINERS

	Bare-metal	***VM***	***Container***
OS size	medium	medium	low
Resource usage	high	medium	low
Deployment time	high	high	low
Data persistence	high	high	low
Expected lifetime	high	high	low

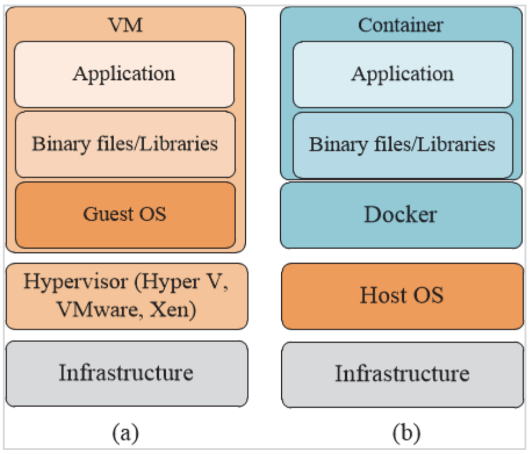

Fig. 2. VM stack (a) and Container stack (b).

For example, it is highly undesirable to run 32-bit OSs on top of 64-bit OS and vice-versa, because that way a Linux distribution cannot use all the kernel modules that are already present to the full extent.

When talking about containers, there are two major aspects that need to be considered: container size and lifetime. As containers rely on the kernel and other parts of OS that is already installed, their size should be kept at minimum. This aspect can be accomplished by using container images that have only the necessary packages and software installed. Another fact that underpins this aspect is that containers are meant to be used for stateless applications and services, or in other words, for immutable data. Those parts of applications and services that do not change frequently are good candidates to be containerized. If there are many write operations adding/changing data inside the DB that means that DBs are not good candidates for use in containers. Applications and services generating a large amount of logs should not write log files to the container. In both of these examples, not only the DB and log files change but they also grow in size. Even though a container will use all the resources available to it, if the size of a container grows significantly it can affect normal operation of the VM or bare metal server on top of which the container engine is running.

From an IdP point of view, DB and user registry can be considered as candidates for use inside a container. While this is feasible, it is also not recommended to keep mutable data inside a container, because container's lifetime is limited by the lifetime of the application. If a user registry is kept inside a container and the container gets brought down in order to update the application, all of users' credentials will be lost without possibility of recovering the data. As a general rule, sensitive data and data that need to be preserved should never be kept inside a container.

By following the previously mentioned design goals regarding container size and lifetime, containers can fulfill their main purpose, which is easy (re)deploying. That means that when there are changes in an application that's run in a container, one can safely add or remove containers and run tests, without worrying if it will cause problems. The speed at which containers can be brought up or down is much better in comparison to creating VMs. Using shared OS is not possible when creating and running VMs. Two obvious consequences of using "shared" OS are the speed and simplicity with which containers are built. Therefore, these two benefits that containers bring represent a foundation for faster adoption of this technology.

The use case to be considered in this paper is deploying containers on a VM. As it was already mentioned, VMs have an OS installed. In order to be able to use containers, certain software packages are needed. Those packages can be acquired using usual package management tools that are a part of Linux distribution. After installing these packages, OS has a container engine at its disposal. The engine is responsible for running/stopping containers, as well as accepting different container-related commands. Container engine that is discussed in this paper is Docker [8]. Even though Docker is not the only tool for creating containers, it can be considered as most widely deployed. In order to give guidelines to any future containerization tools, there is also an initiative in open-source community to standardize containers [9].

The first step to take when moving a service to a containerized environment is to decide which container images to use. As Docker was created within open-source community and was at first meant to be used on different Linux distributions, many of the software packages that exist in Linux are also available for use in Docker. Container images for DBs (e.g. PostgreSQL), web servers (e.g. Apache) etc. can be used without any restrictions or loss of functionality. The main advantage of container images is that they are tailored to be small and to hold just the necessary software dependencies. This can be confirmed by the fact that e.g. CentOS container image has a size smaller than 200MB, while CentOS minimal ISO image has a size around 600MB. One thing that has to be mentioned is that container images are always based on an existing Linux distribution. In that aspect, one of the most used and lightweight distributions that almost all of the official images are using is Alpine Linux, which is only 4MB in size. This Linux distribution was specially designed for use in containerized environment, but it is not an official requirement to use it. Even though lots of open-source tools can be easily used inside a container, there are also some that are used widely in R&E community (e.g. OpenLDAP) which, at the time of the writing of this paper, still do not have an official image repository.

If an official image does not exist, a custom one can be created. Even though image creation is out of the scope of this paper, the main principles of this process have to be taken into consideration. Fig. 3 shows all the necessary steps to build a fully functional container environment.

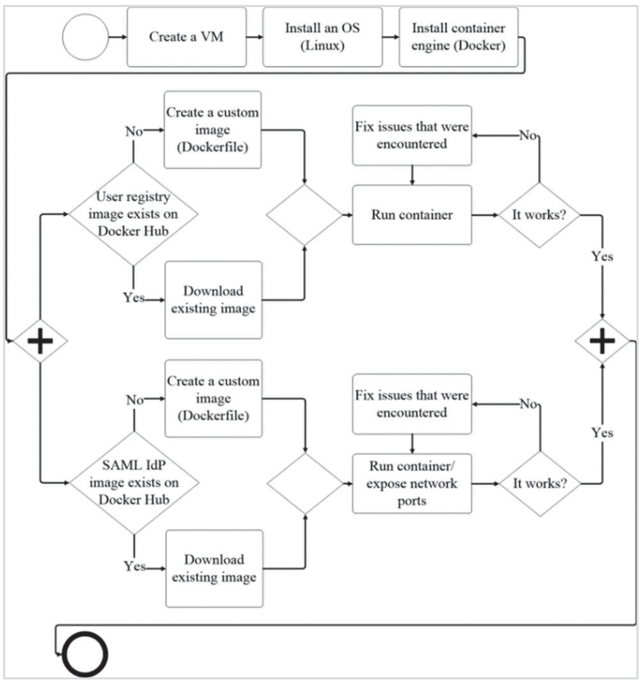
Create a VM
Install an OS (Linux)
Install container engine (Docker)
Create a custom image (Dockerfile)
No
User registry image exists on Docker Hub
Yes
Download existing image
Fix issues that were encountered
No
Run container
It works?
Yes
SAML IdP image exists on Docker Hub
Run container/ expose network ports

Fig. 3. Container deployment procedure.

As it was already said, a container image is usually dependent on an OS, like e.g. Linux. After choosing the OS, there are commands that enable installing the necessary software packages for that OS. Some of the commands that the Docker engine enables are: changing the working directory, copying configuration files or entire directories, setting the environmental variables, etc. Besides having the part concerning managing OS behavior, there is also a part of the container image file that is reserved for setting the networking. If, for example, one needs to use a web server, then specific TCP/UDP ports are expected to be open, like TCP port 80 (Hypertext Transfer Protocol) or TCP port 443 (HTTPS). The important thing to note here is that not only well-known ports have to be used. The number of commands used inside a Dockerfile, which is the name of the custom built image file, is considered important because it affects the time needed to build that container image. Grouping the same commands, e.g. when setting environmental variables, will almost always lower the build time, which can be of importance if containers are to be used in a production environment. The build time is dependent on the number of commands used in Dockerfile. Image building starts from the base (parent) image. If the image is created for the first time, each command from Dockerfile will create a new checksum for each layer built on top of the base image. Each of these layers is cached, which means that they are present on the local file system. The reason for very quick container deployment lies in the fact that cached layers are verified when running a new container. When using the same base image for building multiple containers, the build process will compare the checksums at each layer. If the checksums are different, then the cache will be invalidated and a new checksum is created for each step which has changed during the build process. If e.g. a new file was added using Dockerfile, it invalidates the cache and creates a new layer which results in creating a different checksum compared to the checksum of previous valid layer. The build time shouldn't increase significantly compared to the original image. If there aren't any changes in the Dockerfile, a new container will be spawned almost immediately. In short, each layer is used to keep track of the changes that were made when running a command inside a Dockerfile compared to the base image. The rule of the thumb is that the things that change most often (e.g. HTML pages) should be put as close to the end of the Dockerfile as possible. After a container image has been carefully prepared, the only thing left to do is to run one or more containers based on that newly created image. One of the best practices for building custom images is to use a script, that's called an entrypoint. This script's role is to enable further setting and fine-tuning parameters of the services that are going to be run in the container.

Neither sensitive/confidential data nor user registries should be kept inside a container. When a container needs to access contents of a DB/user registry, there is an option to keep the data on the VM and map that locally stored data to the container. This greatly simplifies container usage, because containers can be created or removed without worrying whether or not it will somehow affect data that must be kept safe. This is especially important for the case of IdPs in inter-federated environment. Creating an IdP is much faster and easier with containers. If an IdP has an LDAP Data Interchange Format (LDIF) file with user credentials prepared, there just needs to exist a container which will have read access to this file. In order to be able to authenticate and authorize users based on LDIF file, the container has to have OpenLDAP installed. In other words, there has to be a dedicated OpenLDAP container. If there are any changes applied to LDIF file or any other configuration file, using bind mounting will enable these changes to be visible to container almost immediately.

To fully comply with inter-federation requirements, there also needs to be a Shibboleth or SimpleSAMLphp container. All the configuration files and parameters can be modified when defining the container image. Possibly the biggest challenge is to configure communication between these two containers, which falls into networking category. One of the most debated container functionalities is networking and whether or not it provides enough protection. When a new container is created in the default network, it is automatically assigned a Class B IP address [10]. This means that, without using Network Address Translation (NAT) protocol, containers cannot be seen or accessed outside that local network and VM. The general rule is that each new container in the default network gets assigned with the first available Class B IP address. The conclusion that can be drawn here is that all containers that belong to the same network (default or user defined) can communicate between themselves without any restrictions, due to bridged nature of virtual interfaces assigned to containers. This can be considered satisfactory when using containers for testing purposes. If containers are supposed to be used in production environment, usually some of the well-known network ports have to be exposed. In inter-federated services, HTTPS port (TCP port 443) is expected to accept connections. In order to allow for the best protection possible offered by containers, it is recommended to start containers with port redirection, e.g. open port 8443 on the VM and redirect all connections coming to that port to the container port 443. When using port redirection in containers, it does not require human intervention to configure VM's firewall rules because they are automatically created and exist only for as long as a container is used. From an IdP point of view, OpenLDAP container does not have to be visible to the outside world, so only HTPPS port should be exposed. This assumption does not influence communication between OpenLDAP and Shibboleth/SimpleSAMLphp containers which still unfolds without any disruptions.

Probably the biggest disadvantage of using containers is the fact they share the same kernel with a VM. Problems that affect kernel will affect all deployed containers. Any kernel state that prevents normal functioning can be considered as potentially dangerous, because by gaining access to kernel, a malicious attacker does not need to access a container to create a Denial-of-Service (DoS) attack. There is a possibility that future attack vectors on containers will try to exploit any kernel bugs or vulnerabilities, like recent bugs that were present in Intel's CPU kernel (Spectre and Meltdown). Another disadvantage that has been observed is a lack of possibility for a dynamic configuration. What this means is that the most of the configuration has to be prepared upfront. That can be a limiting factor in some cases where the configuration has to change based on e.g. domain name, the hostname of the server/VM, etc. One example where those exact parameters are needed is if you want to use HTTPS or a secure connection to OpenLDAP container, both of which require TLS (Transport Layer Security) certificates. Even though Linux packages used for creating certificates are preconfigured with default values, those values have to be changed to reflect the present state: hostname must be different for every container. The uniqueness of certificates is paramount for security paradigm to be withheld. For that reason, dynamic configuration is something that can be viewed, at this point, as very useful on one hand and as a limiting factor on the other hand. This is especially important for large scale container deployments where each of them has different configuration parameters.

IV. Conclusion and Future Work

The main goal of this paper was to show a new methodology that helps with easier implementation of IdP component that is to be deployed as an entity inside an Identity Federation. Containers can be very helpful in this aspect, as they are meant to provide continuous service improvement, simplified service deployment and resource preservation. Furthermore, a container can be deployed very fast and without the need to fully understand all the details regarding the OS part. Even though containers have their advantages, there are some concerns regarding security and their inability to handle persistent data. Nevertheless, containers can be considered as an advanced technology and future work will focus on finding a solution best suited for large-scale deployments, which will include comparing the performance of automation tools like Kubernetes and Docker Swarm. Additional points of interest for further research are going to focus on how to deploy containers in OpenStack cloud environment.

References

[1] *Security Assertion Markup Language (SAML) 2.0 Profilefor OAuth 2.0 Client Authentication and Authorization Grants*, RFC7522, 2015

[2] Mikael Linden, Brook Schofield, Shannon Milsom. (2013.04.19). *eduGAIN Policy Framework Constitution* (GN3-10-326 v2.0).Available:https://technical.edugain.org/doc/GN3-10-326%20eduGAIN_constitution%20v2.0.pdf

[3] L. Hämmerle, R. Sabatino et al. (2016.04.22). (GN4-1-16-37dcb3). Available:https://www.geant.org/Resources/Documents/Comparison-of-AAIs-for-Research_White-Paper_v1.0.pdf

[4] Council of the European Union, *General Data Protection Regulation*, Available: http://data.consilium.europa.eu/doc/document/ST-5419-2016-INIT/en/pdf

[5] *OAuth 2.0 for Native Apps*, RFC8252, 2017

[6] *OpenID Connect Core 1.0*, Available: http://openid.net/specs/openid-connect-core-1_0.html

[7] *Software-Defined Networking (SDN): Layers and Architecture Terminology*,RFC 7426, 2015

[8] Karl Matthias and Sean Kane. (2015.06). *Docker up and running* (vol. 1). Available: https://www.oreilly.com/

[9] The Linux Foundation, *Open Container Initiative Charter*, Available:https://www.opencontainers.org/about/governance

[10] *Address Allocation for Private Internets*, RFC1918, 1996

[11] M. S. Eremija, N. R. Ilić, M. Cvetanović, J. Protić and Z. Radivojević, "Identity provider deployment based on container technology,"*2017 25th Telecommunication Forum (TELFOR)*, Belgrade, 2017, pp. 1-4.