IBM **Security**

IBM **Security** Verify Access
Version 10.0.2

Docker and Docker Compose
Deployment Cookbook

Jon Harry

Version 1.0
June 2021

NOTICES

This information was developed for products and services offered in the USA.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

TRADEMARKS

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

IT Infrastructure Library is a Registered Trade Mark of AXELOS Limited.

ITIL is a Registered Trade Mark of AXELOS Limited.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Table of Contents

1 Introduction 5
1.1 High Level Architecture 5
1.1.1 Test Machine 5
1.1.2 Docker Hub 6
1.1.3 Verify Access Trial Center 6

2 Getting Started 7
2.1 Start test machine and sign in. 7
2.2 Verify Internet Connectivity 7
2.3 Clone GitHub repository 7

3 Quick Introduction to Docker 8

4 More Docker concepts 11
4.1 Start a service in Docker 11
4.2 Execute a command in the running container 12
4.3 Start a shell in the running container 12
4.4 Network Connectivity 12
4.5 Port Mapping 14
4.6 Volumes and Bind Mounts for data persistence and data sharing 15
4.6.1 Generate Certificate and Key Files 16
4.6.2 Specify Volumes in docker run command 16
4.7 Environment variables 18

5 Verify Access Installation on Native Docker 19
5.1 Examine Docker Setup Script 19
5.2 Run the Docker Setup Script 21
5.3 Verify running containers 21

6 Verify Access initial configuration 23
6.1 Connect to Local Management Interface of Configuration Container 23
6.2 Load Certificates for LDAP and Database connections 25
6.2.1 Load LDAP Certificate 25
6.2.2 Import Database Certificate to LMI and Runtime key stores 27
6.3 Configure Runtime Database 31
6.4 Obtain Trial License from the Verify Access Trial Center 33
6.5 Apply trial license certificate 34

7 Verify Access Base Configuration 37
7.1 Configure Verify Access Base Runtime Component 37
7.2 Configure "rp1" Reverse Proxy Instance 40
7.2.1 Create Instance 40
7.2.2 Edit Reverse Proxy Configuration File 43
7.3 Publish Snapshot 44
7.4 Test Configuration 46
7.5 Create Users 47

8 Additional Configuration 49
8.1 Enable Distributed Session Cache 49
8.1.1 Enable DSC Globally 49
8.1.2 Enable DSC in Reverse Proxy configuration 50
8.1.3 Publish snapshot and restart DSC and Reverse Proxy 52
8.1.4 Check Registration 53
8.2 Configure Reverse Proxy for Advanced Access Control 54

8.2.1 Set easuser password 54
8.2.2 Publish Configuration and restart the runtime container 55
8.2.3 Configure Reverse Proxy for Authentication and Context-based Access 56
8.3 Configure Username Password Mechanism 57
8.4 Publish Configuration and reload containers 59
8.5 Test Updated Configuration 59
8.5.1 Login using AAC Advanced Authentication Mechanism 59
8.5.2 Check Sessions in Distributed Session Cache 60

9 Backup configuration and clear native Docker system 63

10 Docker Compose 64
10.1 Introduction 64
10.2 The Compose Project Directory 64
10.3 Compose File 64
10.4 Environment File 65
10.5 Create Environment with Docker Compose 66
10.5.1 Create Key Shares for OpenLDAP and PostgreSQL 66
10.5.2 Run Docker Compose "up" command 66
10.5.3 Restore configuration from backup 67
10.6 Test Environment 68
10.7 Clear Docker Compose system 69

11 Notices 70

1 Introduction

This cookbook explores the deployment of IBM Security Verify Access 10.0.2 within Docker containers.

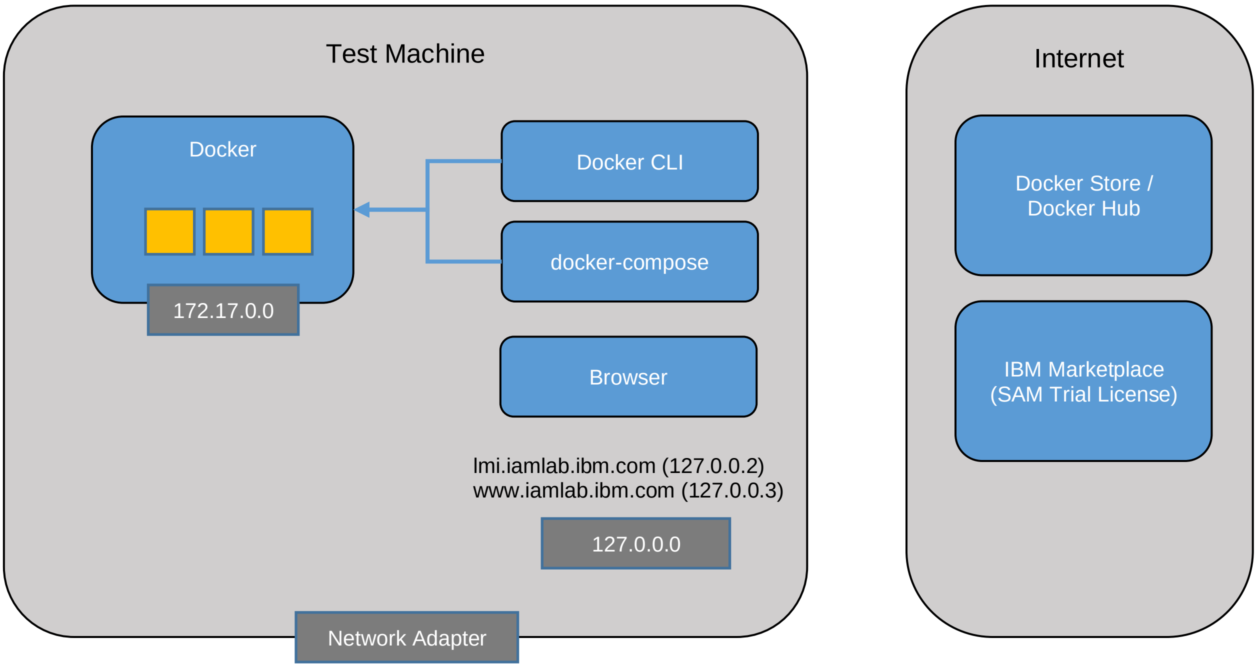
This cookbook details deployment of Verify Access using native Docker and using Docker Compose.

The scripts and assets used in this cookbook are available in a Github repository. This link will take you to the repository home page: https://github.com/iamexploring/container-deployment.

The exercises described in this cookbook are designed to run on a self-contained test machine which has the required software and helper scripts installed. Instructions on building a suitable test machine (based on Centos 7 Linux) can be found here: https://ibm.biz/isamdockerbuild.

1.1 High Level Architecture

The high-level architecture for the environment described in this document may be summarized as follows:



1.1.1 Test Machine

The test machine is a physical or virtual machine which has the following components installed:

- **Docker –** This provides the container services used to explore native Docker installation of Verify Access. It includes a command line tool (docker) for management.
- **Docker Compose –** This tool provides automation for native Docker which can be used to create and manage multi-container environments more efficiently than using native Docker commands. It includes a command line tool (docker-compose).
- **Browser –** A browser is required for accessing the Verify Access admin console and Reverse Proxy. This cookbook was written using Firefox but any up-to-date browser should work.

The Test Machine needs to have outbound connectivity to the Internet so that it can connect to Docker Hub and Docker Store for download of images and so that an Verify Access trial license can be obtained.

The Test Machine needs to have at least 2 local IP addresses available for the Verify Access components to bind to. The provided scripts assume use of loopback addresses (127.0.0.2 and 127.0.0.3). If external connectivity to the Verify Access components is required, you will need to use externally addressable IP addresses instead.

1.1.2 Docker Hub

Docker Hub is a repository of Docker images which can be used to create Docker containers. The official images for IBM Security Verify Access are hosted on Docker Hub. The as-is "Verify Access ready" images for OpenLDAP and PostgreSQL are also hosted here. Internet-connected Docker systems can automatically pull images from the Docker Hub as required. Links:

- https://hub.docker.com/r/ibmcom/verify-access
- https://hub.docker.com/r/ibmcom/verify-access-wrp
- https://hub.docker.com/r/ibmcom/verify-access-dsc
- https://hub.docker.com/r/ibmcom/verify-access-runtime
- https://hub.docker.com/r/ibmcom/verify-access-openldap
- https://hub.docker.com/r/ibmcom/verify-access-postgresql

> In previous versions, a single *verify-access* image was used to run all components of the Verify Access system. In 10.0.2.0, separate light-weight images are available for each component. These start more quickly and use less system resources. They should be used where possible.

1.1.3 Verify Access Trial Center

In order to activate an IBM Security Verify Access system, you will need a trial certificate. A certificate for a 90-day trial can be obtained via the Verify Access Trial Center. You will need to register for an IBMid (if you don't already have one) to access this site. Link: https://ibm.biz/isamtrial

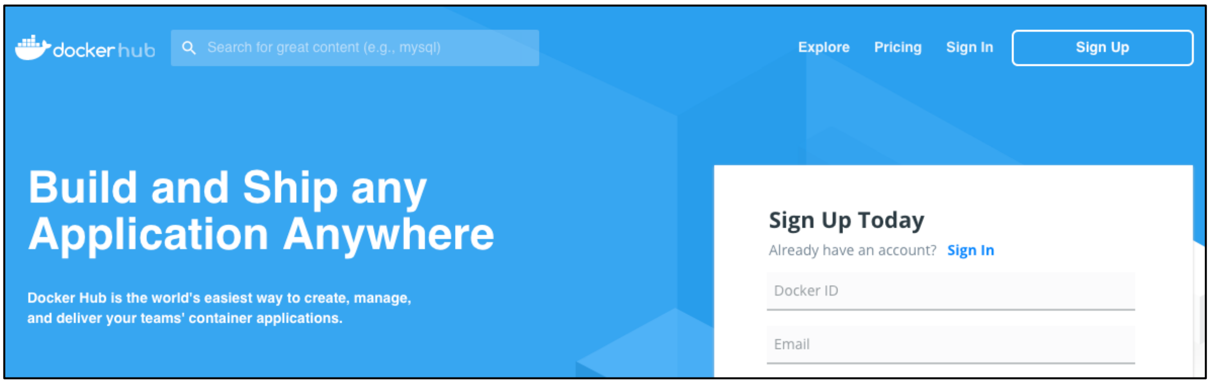
2 Getting Started

2.1 Start test machine and sign in.

This cookbook assumes the use of a self-contained test machine which has the required software and some helper scripts pre-installed. Start your test machine now and login.

2.2 Verify Internet Connectivity

Open a browser and navigate to the URL: **https://hub.docker.com**. Verify that you are successfully connected.



If you see the Docker Hub web site, you have Internet connectivity.

> If you do not have internet connectivity, you will not be able to download the required Docker images from the Docker Hub, you won't be able to clone the scripts, and you won't be able to obtain an Verify Access trial license certificate. In this case, you will only be able to complete these cookbook exercises if these items have been pre-loaded to your environment.

2.3 Clone GitHub repository

Clone the GitHub repository that contains the assets and scripts used in this cookbook to your test machine:

```
[demouser@centos ~]$ mkdir git
[demouser@centos ~]$ cd git
[demouser@centos git]$ git clone https://github.com/iamexploring/container-deployment
Cloning into 'container-deployment'...
remote: Enumerating objects: 162, done.
remote: Counting objects: 100% (162/162), done.
remote: Compressing objects: 100% (85/85), done.
remote: Total 162 (delta 74), reused 162 (delta 74), pack-reused 0
Receiving objects: 100% (162/162), 92.07 KiB | 567.00 KiB/s, done.
Resolving deltas: 100% (74/74), done.
[demouser@centos git]$ ls container-deployment/
README.md   common     compose     docker     helm       kubernetes      openshift
```

You are now ready to start the labs.

3 Quick Introduction to Docker

In this section we will run a few Docker commands to get familiar with Docker concepts.

Open a terminal window. Most interactions with Docker use the command line.

First, let's see what images we have locally. Enter the following command:

```
[demouser@centos ~]$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
```

In a clean system you will see the empty list shown above. If your system has been pre-loaded with images, they will be listed here.

You will now test Docker functionality. The following command will cause Docker to pull an image from the Docker Hub (assuming it is not already pre-loaded), create a container using this image, and then execute a command within the container.

> The use of the --*name* flag below is optional. If not specified, a random name is assigned to the container.

```
[demouser@centos ~]$ docker run --name test hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:c3b4ada4687bbaa170745b3e4dd8ac3f194ca95b2d0518b417fb47e5879d9b5f
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

[demouser@centos ~]$
```

If you see the message above, your Docker system is working correctly.

Let's have a look at the running containers on the system:

```
[demouser@centos ~]$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
```

You might be surprised to see that there are no containers running. That's because when the command executed in the container terminates, the container terminates too. The *hello-world* container terminated once it displayed its message.

To see all containers, including those that have terminated, use this command:

```
demouser@centos ~]$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
3c4a95df714a        hello-world         "/hello"            11 minutes ago      Exited (0)
11 minutes ago                       test
```

Now you can see the *test* container that you just ran. You can see that it exited (with return code 0). You can also see the *Container ID* and *Name*. Either of these identifiers can be used when specifying a container.

> This list also shows the command that was executed when the container was run. This command is specified as part of the image definition (along with other things like listening ports and volume mount points which are not applicable for this test image)

You can start a stopped container using the *start* command. This executes the same command that was used on the initial run. Try it now:

```
[demouser@centos ~]$ docker start test
test
```

That's strange, why are you not seeing the output that you saw when the container was run the first time?

The answer is that, by default, the *start* command does not attach to the STDOUT/STDERR of the executed command, so output is not displayed on the terminal. If you want to see it, you'll have to add the attach flag:

```
[demouser@centos ~]$ docker start -a test

Hello from Docker!
This message shows that your installation appears to be working correctly.
…
For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

> For interactive commands (like bash) you can also attach to the STDIN of the executed command using the *-i* flag.

The output from a container is also stored to a log file (which persists until the container is deleted). You can use the *logs* command to view the log:

```
[demouser@centos ~]$ docker logs test

Hello from Docker!
This message shows that your installation appears to be working correctly.
…
For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

A stopped container can be removed. This cleans up the transient filesystem it was using (including any logs etc.). All state associated with the container is lost.

```
[demouser@centos ~]$ docker rm test
test
```

The container has now been removed. If a new one is needed, the *run* command can be used to create a new container from the original image.

Let's list the images again:

```
[demouser@centos ~]$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              fce289e99eb9        9 months ago        1.84kB
```

You can see the *hello-world* image that was pulled from the Docker repository when you first ran the container.

The full name of an image includes the REPOSITORY and TAG. Often the TAG is used to specify a version number. If the tag is omitted, *latest* as assumed. *latest* is a special tag which refers to the most recent version of an image.

An image can be deleted if no containers are using it (to reclaim the disk space it is using locally):

```
[demouser@centos ~]$ docker rmi hello-world:latest
Untagged: hello-world:latest
Untagged: hello-
world@sha256:c3b4ada4687bbaa170745b3e4dd8ac3f194ca95b2d0518b417fb47e5879d9b5f
Deleted: sha256:fce289e99eb9bca977dae136fbe2a82b6b7d4c372474c9235adc1741675f587e
Deleted: sha256:af0b15c8625bb1938f1d7b17081031f649fd14e6b233688eea3c5483994a66a3
```

> An image is actually made of multiple "layers". Multiple images may share layers if they have been built from the same parent image. The image delete command "untags" the image in the local repository. At that point, if any image "layers" are no longer being used by any local images they are deleted.

4 More Docker concepts

In this section you will run an OpenLDAP service in Docker to further explore Docker concepts.

IBM has built an Verify Access-specific OpenLDAP image as a layer on top of the standard OpenLDAP image. This image is published on the Docker Hub. It is NOT supported by IBM.

The main changes in the Verify Access image are:

- Addition of Verify Access schema
- Addition of secAuthority=Default suffix
- Default use of TLS (including dynamic keypair generation if required)
- Default NOT to listen on insecure port

4.1 Start a service in Docker

You will now run a container using this image. Run the following command:

```
[demouser@centos docker]$ docker run -d --name openldap ibmcom/verify-access-
openldap:10.0.2.0
Unable to find image 'ibmcom/verify-access-openldap:10.0.2.0' locally
10.0.2.0: Pulling from ibmcom/verify-access-openldap
177c8d195b28: Pull complete
…
797dcdce0711: Pull complete
Digest: sha256:d06142441de6fb3e1b701444dd59d4f648710c8a794196db59e18ef5ed6a675e
Status: Downloaded newer image for ibmcom/verify-access-openldap:10.0.2.0
5bb2a237b37ce3168ee942124a1696a7cfda88b0821362f844b9ffaa857efd5c
```

> Note the use of the -d flag. This tells Docker to detach STDIN and STDOUT from the executed command. If this flag is not used, local STDIN and STDOUT will be attached to the command and the container will effectively be running in the foreground.

The OpenLDAP container is now running:

```
[demouser@centos docker]$ docker ps
CONTAINER ID        IMAGE                                     COMMAND                  CREATED
STATUS              PORTS                 NAMES
568169302e50        ibmcom/verify-access-openldap:10.0.2.0   "/container/tool/run"   3
minutes ago        Up 3 minutes        389/tcp, 636/tcp    openldap
```

4.2 Execute a command in the running container

Now that the container is running, you can execute a command within the container using the docker exec command. Let's run an ldapsearch command within the container to ensure it is running properly:

> The password of admin and suffix of dc=example,dc=org are defaults used because you didn't specify anything when running the image.

```
[demouser@centos ~]$ docker exec openldap ldapsearch -H ldaps://localhost:636 -D
cn=admin,dc=example,dc=org -w admin -b dc=example,dc=org objectclass=*
# extended LDIF
#
…
# example.org
dn: dc=example,dc=org
objectClass: top
objectClass: dcObject
objectClass: organization
o: Example Inc.
dc: example
…
# numResponses: 3
# numEntries: 2
```

4.3 Start a shell in the running container

It's also possible to execute an interactive command within a container. This requires use of the -t (assign pseudo-TTY) and -i (interactive) flags. You can use this to run a shell in the container:

```
[demouser@centos ~]$ docker exec -ti -- openldap bash
root@568169302e50:/#
```

> It is good practice to add -- after the command flags so that any flags associated with the command executed are not interpreted by the docker command.

You now have a root shell inside the OpenLDAP container. You can run any commands from here that are available in the container. Explore the container filesystem and commands available. When you're done, exit the shell:

```
root@568169302e50:/# exit
exit
[demouser@centos ~]$
```

4.4 Network Connectivity

Docker containers are connected to each other, to the host, and to the external world via networks defined in the Docker system. You can list these networks with this command:

```
[demouser@centos ~]$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
8fc4e326a500        bridge              bridge              local
701df26aa556        host                host                local
afa3c6f7f43c        none                null                local
```

Here you can see the three built-in networks of a Docker system. All containers that are on the same network can communicate with each other. The *Driver* for the network determines external connectivity.

- The *none* driver provides no external connectivity.
- The *host* driver provides bi-directional connectivity with the host machine.
- The *bridge* driver provides bi-directional connectivity with the host machine and outbound connectivity (via Network Address Translation) with the external world.

Additional networks can be created. It is common to create a custom "bridge" network because custom networks include automatic DNS resolution of connected containers which is not provided by the default bridge network (for some historical reason).

When a container is run, it can be connected to one of more networks. If no network is specified, it will be connected to the default *bridge* network. Each container is allocated an IP address on each of the Docker networks that it is connected to.

You can see the containers connected to a network (and their allocated IP addresses) using the *network inspect* command.

```
[demouser@centos ~]$ docker network inspect bridge
…
        "Containers": {
            "7f127e377c6c679cae4fd577be8d78d845680015828262098d338550852916d3": {
                "Name": "openldap",
                "EndpointID":
"a239aaad94b3c5d8beedecb45d6882332c88adfa903319800bf31e4cb16f2ce8",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            }
        },
…
```

Note the IP address for the *openldap* container. This IP address can be used to communicate with the container. Now, use this IP address to make an LDAP connection from the host. Note that this may not work on recent versions of MacOS because of changes in networking.

> The LDAPTLS_REQCERT environment variable is set to never to turn off certificate checking for the connection. This is required because you don't have the public certificate of the LDAP server saved to your LDAP client certificate file.

```
[demouser@centos ~]$ export LDAPTLS_REQCERT=never
[demouser@centos ~]$ ldapsearch -H ldaps://172.17.0.2:636 -D cn=admin,dc=example,dc=org -w
admin -b dc=example,dc=org objectclass=*
…
# example.org
dn: dc=example,dc=org
objectClass: top
objectClass: dcObject
objectClass: organization
o: Example Inc.
dc: example
…
# numEntries: 2
```

The issue with using the Docker network for communication is that container IP addresses are dynamically allocated (so you don't know what they will be in advance) and, more importantly, they don't allow for inbound connections from outside the host system. You will now look at how port mapping allows external inbound connections.

4.5 Port Mapping

Port mapping sets up port-forwarding from ports on interfaces of the host system to ports within containers.

```
[demouser@centos docker]$ docker ps
CONTAINER ID        IMAGE                              COMMAND                 CREATED
STATUS              PORTS                 NAMES
568169302e50        ibmcom/verify-access-openldap:10.0.2.0   "/container/tool/run"   8
minutes ago       Up 8 minutes          389/tcp, 636/tcp    openldap
```

Note the ports listed here (389/tcp, 636/tcp). These are the advertised listening ports (as defined by the image). Since you have not mapped these ports, they are only currently available on the Docker internal network. It's also worth noting that just because a port is listed here, it doesn't mean it is in use. Also, just because a port is not listed here, that doesn't mean you can't map it.

Like many container configuration items, port mapping can only be configured when a container is created. So, you need to delete the current openldap container and create a new one to set up port mapping.

To stop a running container, use the stop command. Once the container is stopped it can be deleted.

```
[demouser@centos ~]$ docker stop openldap
openldap
[demouser@centos ~]$ docker rm openldap
openldap
```

You will now create a new container. In the *docker run* command we use the *-p* flag to specify that host interface 127.0.0.2 port 1636 should be forwarded to port 636 on this container.

```
[demouser@centos ~]$ docker run -d -p 127.0.0.2:1636:636 --name openldap ibmcom/verify-
access-openldap:10.0.2.0
82e45ed64ed83f8a5fad5e0e276d8658c39dc8826ada234d96a2280ed9ca7e7b
```

> Note that this time there is no pull operation this time because the openldap image is already in the local image repository.

You can see the port mapping when you view the container:

```
[demouser@centos ~]$ docker ps
CONTAINER ID        IMAGE                              COMMAND                 CREATED
STATUS              PORTS                                     NAMES
82e45ed64ed8        ibmcom/verify-access-openldap:10.0.2.0   "/container/tool/run"   About
a minute ago   Up About a minute   389/tcp, 127.0.0.2:1636->636/tcp   openldap
```

It's also visible with the *docker port* command:

```
[demouser@centos ~]$ docker port openldap
636/tcp -> 127.0.0.2:1636
```

> It's interesting to note that the way the mapping is displayed here is misleading. It appears to indicate that *636/tcp* is forwarding to *127.0.0.2:1636* but, in fact, it is the other way around.

With this port mapping in place, clients can connect to 127.0.0.2:1636 and will be connected to the OpenLDAP container port 636. Let's try that now:

```
[demouser@centos ~]$ export LDAPTLS_REQCERT=never
[demouser@centos ~]$ ldapsearch -H ldaps://127.0.0.2:1636 -D cn=admin,dc=example,dc=org -w
admin -b dc=example,dc=org objectclass=*
…
# example.org
dn: dc=example,dc=org
objectClass: top
objectClass: dcObject
objectClass: organization
o: Example Inc.
dc: example
…
# numEntries: 2
```

For ease of configuration, an internal loopback address (127.0.0.2) has been used here for port mapping. To make services in Docker containers visible outside the Docker host, you would use an externally routable IP address for the port mapping.

To remove a running container, you can use the remove command with the force flag:

```
[demouser@centos ~]$ docker rm -f openldap
openldap
```

4.6 Volumes and Bind Mounts for data persistence and data sharing

The containers you've created so far have only used the filesystem created from the image. Any changes to this filesystem are lost when the container is deleted, and they cannot be shared between containers.

Docker has the concept of *Volumes* which allow persistent storage outside the container to be used as part of the container filesystem. Volumes are not deleted when a container is deleted and so the data is preserved and can be used with new container at a later time. It is also possible for multiple containers to use the same volume so that they have access to shared files and folders.

Docker also supports *Bind Mounts* which allow a file or directory on the host filesystem to be mounted within the container. This is useful for making files (such as keys, certificates, or configuration files) directly accessible within the container.

4.6.1 Generate Certificate and Key Files

In your Docker environment you need keys and certificates for our LDAP and Database containers. You will now use a script to create these files:

```
[demouser@centos ~]$ git/container-deployment/common/create-ldap-and-postgres-keys.sh
Creating LDAP certificate files
Generating a 4096 bit RSA private key
…
writing new private key to '/home/demouser/git/container-
deployment/local/dockerkeys/openldap/ldap.key'
-----
Creating LDAP dhparam.pem
Generating DH parameters, 2048 bit long safe prime, generator 2
This is going to take a long time
…
Creating postgres certificate files
Generating a 4096 bit RSA private key
…
writing new private key to '/home/demouser/git/container-
deployment/local/dockerkeys/postgresql/postgres.key'
```

When a Bind Mount is used, the permissions on the mounted directory (and the files it contains) can be updated by the container. This can render the files unreadable by a non-root user on the host system. To avoid losing access to your keys, you will copy them to another location and mount the copy instead of the original files.

```
[demouser@centos ~]$ mkdir dockershare
[demouser@centos ~]$ cp -R git/container-deployment/local/dockerkeys dockershare
```

4.6.2 Specify Volumes in docker run command

The following docker run command specifies volumes to be used to store LDAP data and a bind mount to be used to mount the required certificate files.

```
[demouser@centos ~]$ docker run -d -v /var/lib/ldap -v /etc/ldap/slapd.d -v
/var/lib/ldap.secAuthority -v
${HOME}/dockershare/dockerkeys/openldap:/container/service/slapd/assets/certs
--name openldap ibmcom/verify-access-openldap:10.0.2.0
fe0c7a5c848e495b5986c7fbffd8f1e349e37dfa0de8bf6c1016ccddaea1ea8c
```

The first three volume parameters above create volumes for the locations on the OpenLDAP container filesystem where LDAP data is stored. The final one mounts the key and certificate files you just created into the location where the OpenLDAP process will find them.

To check that the OpenLDAP server is using the certificate file from the bind mount, we can use OpenSSL to dump the certificate information:

```
[demouser@centos ~]$ docker exec -- openldap bash -c "echo | openssl s_client -connect localhost:636 -brief"
depth=0 CN = openldap, O = ibm, C = us
verify error:num=18:self signed certificate
CONNECTION ESTABLISHED
Protocol version: TLSv1.2
Ciphersuite: ECDHE-RSA-AES256-GCM-SHA384
Peer certificate: CN = openldap, O = ibm, C = us
Hash used: SHA512
Verification error: self signed certificate
Supported Elliptic Curve Point Formats: uncompressed
Server Temp Key: ECDH, P-256, 256 bits
DONE
```

To see the volumes that have been created, use the *docker volume* command:

```
[demouser@centos ~]$ docker volume ls
DRIVER              VOLUME NAME
local               18974f9bb7eab10587735b6dc82de8233e0d933979b23c194750f8679bbfb974
local               89b3ef3bcfd598abe4d0443afa19b48c05dc546831adc08eef80e26bb107d437
local               8ddff372ce80de8be653dfbdaaefb5f1e7978625e192369fcdb34262d197374e
```

Note that the names of these volumes have been dynamically created and they are not very readable. It's not easy to know which volume is mounted where on the container. Later you will see how to create named volumes and then use these in the *docker run* command.

For now, stop and delete this container. The *-v* flag in the *docker rm* command tells docker to also delete the volumes associated with the container.

```
[demouser@centos ~]$ docker rm -vf openldap
openldap
```

At this point, take a look at the directory that was bind mounted into the OpenLDAP container:

```
[demouser@centos ~]$ ls -ld dockershare/dockerkeys/openldap/
drwxrwxr-x. 2 polkitd input 4096 Feb 15 18:39 dockershare/dockerkeys/openldap/
```

Notice the strange owner and group values. The ownership of this directory was modified by the openldap container. Your *demouser* user is no longer the owner. If you wanted to delete or modify the files in this directory you would need to be the root user to do so. Note: on some docker versions, host filesystems are protected against owner change and so you won't see this owner change.

> With the openldap image, it is possible to prevent the modification of the volume mount by using the *--copy-service* option when running the container. It's a good idea to always use this option with openldap.

Fix up the ownership of the *openldap* bind-mounted directory with the following commands (replacing *demouser* with your own user account):

```
[demouser@centos ~]$ su -
Password: your-root-password
Last login: Fri Nov 16 12:22:46 EST 2018 on pts/0
[root@centos ~]# chown -R demouser:demouser /home/demouser/dockershare
[root@centos ~]# exit
logout
```

4.7 Environment variables

Environment variables can be passed into a Docker container. These are often used to pass configuration options to the container when it is created. Usually, the environment variables that can be used with a container are documented on its page in the Docker Store or Docker Hub (or on a page linked from there).

> For OpenLDAP, the information is available on github here: **https://github.com/osixia/docker-openldap**

You will now create an OpenLDAP container specifying two environment variables. One to set the administrator password and another to set the user suffix (*dc=ibm,dc=com* instead of *dc=example,dc=org*).

```
[demouser@centos ~]$ docker run -d -e LDAP_ADMIN_PASSWORD=Passw0rd -e LDAP_DOMAIN=ibm.com
--hostname openldap --name openldap ibmcom/verify-access-openldap:10.0.2.0 --copy-service
f0a2bc1de83fc89970e6fdeb4dd0ef097fe80250c1dc4d97f447dfc0ac02dfce
```

> Note the addition of the --*host* flag. This sets the hostname for the container. You'll see it on the shell prompt when you connect to the container, but it's also used for dynamic name resolution on custom Docker networks.

Now test the new settings. This time you will run a shell in the container and perform your search from there:

```
[demouser@centos ~]$ docker exec -ti openldap bash
root@openldap:/# ldapsearch -H ldaps://localhost:636 -D cn=admin,dc=ibm,dc=com -w Passw0rd
-b dc=ibm,dc=com objectclass=*
# extended LDIF
…
# ibm.com
dn: dc=ibm,dc=com
objectClass: top
objectClass: dcObject
objectClass: organization
o: Example Inc.
dc: ibm
…
# numEntries: 2
root@openldap:/# exit
exit
```

Finally, delete this container – you'll be creating your proper OpenLDAP container (and the other containers you need for your Verify Access environment) in the next section.

```
[demouser@centos ~]$ docker rm -f openldap
openldap
```

5 Verify Access Installation on Native Docker

In this section you will set up an Verify Access environment in Docker using native Docker commands.

5.1 Examine Docker Setup Script

Rather than enter all of the required Docker commands by hand, you will use a script to set up your Docker environment. Before you run the script, let's take a look at it. Most of the concepts should be familiar from the preceding sections of this lab guide.

Open the script in the *gedit* text editor using the following command:

```
[demouser@centos ~]$ gedit git/container-deployment/docker/docker-setup.sh &
```

The text editor opens in a new window.

The first part of the script sets some variables for directory locations and checks for the keys required for the OpenLDAP and PostgreSQL services. If you've been following this guide, these have already been created.

The script loads environment variables from *common/env-config.sh* to set variables for items such as IP addresses and Verify Access Version.

The first docker command creates a custom network called *isva*:

```
docker network create isva
```

A custom network is used so that dynamic hostname resolution is available to the containers. This allows connections to be made between containers using their hostnames (or *<hostname>.<network>*).

The next lines create named volumes for all of the volumes needed by the environment:

```
docker volume create isvaconfig
docker volume create libldap
docker volume create libsecauthority
docker volume create ldapslapd
docker volume create pgdata
```

Using named volumes makes it easier to re-connect them to new containers if a container needs to be replaced. The *isvaconfig* volume is especially important because it will be shared by all the Verify Access containers to give access to shared configuration.

The first two *docker run* commands create the OpenLDAP and PostgreSQL containers:

```
docker run -t -d --restart always -v pgdata:/var/lib/postgresql/data -v
${DOCKERKEYS}/postgresql:/var/local -e POSTGRES_USER=postgres -e
POSTGRES_PASSWORD=Passw0rd -e POSTGRES_DB=isva -e POSTGRES_SSL_KEYDB=/var/local/server.pem
--hostname postgresql --name postgresql --network isva ibmcom/verify-access-
postgresql:${DB_VERSION}

docker run -t -d --restart always -v libldap:/var/lib/ldap -v ldapslapd:/etc/ldap/slapd.d
-v libsecauthority:/var/lib/ldap.secAuthority -v
${DOCKERKEYS}/openldap:/container/service/slapd/assets/certs --hostname openldap --name
openldap -e LDAP_DOMAIN=ibm.com -e LDAP_ADMIN_PASSWORD=Passw0rd -e
LDAP_CONFIG_PASSWORD=Passw0rd -p ${MY_LMI_IP}:1636:636 --network isva ibmcom/verify-
access-openldap:${LDAP_VERSION} --copy-service
```

Note how the named volumes are used in the volume specifications. Note the *--network* flag connecting the containers to the custom *isva* network. Note the *--restart always* flag which tells Docker that these containers should be auto-started unless they are specifically stopped.

The rest of the *docker run* commands start the Verify Access containers.

```
docker run -t -d --restart always -v isvaconfig:/var/shared --hostname isvaconfig --name
isvaconfig -e CONTAINER_TIMEZONE=Europe/London -e ADMIN_PWD=Passw0rd -p
${MY_LMI_IP}:443:9443 --network isva ibmcom/verify-access:${ISVA_VERSION}

docker run -t -d --restart always -v isvaconfig:/var/shared --hostname isvawrprp1 --name
isvawrprp1 -e CONTAINER_TIMEZONE=Europe/London -p ${MY_WEB1_IP}:443:9443 -e INSTANCE=rp1 -
-network isva ibmcom/verify-access-wrp:${ISVA_VERSION}

docker run -t -d --restart always -v isvaconfig:/var/shared --hostname isvaruntime --name
isvaruntime -e CONTAINER_TIMEZONE=Europe/London --network isva ibmcom/verify-access-
runtime:${ISVA_VERSION}

docker run -t -d --restart always -v isvaconfig:/var/shared --hostname isvadsc --name
isvadsc -e CONTAINER_TIMEZONE=Europe/London -e INSTANCE=1 --network isva ibmcom/verify-
access-dsc:${ISVA_VERSION}
```

Note the environment variables used by the Verify Access containers.

5.2 Run the Docker Setup Script

You are now ready to run the Docker setup script. This will create the network, volumes, and containers that will make up your Verify Access environment.

> If you don't have the required docker images on your system, these will be downloaded during the execution of this script.

```
[demouser@centos ~]$ git/container-deployment/docker/docker-setup.sh
9966c8db9c1a96f59c0ef25d1cf64b0a02f3f1f64b2fb88c9a73dd691737e2f0
isvaconfig
libldap
libsecauthority
ldapslapd
pgdata
1d722a0b9df2a61bf6367d6fe2b192508d852c5b7d8cc04c0a85491e5e5dcbe5
…
c5c2b022f1780072d07bfe7d8d53f3e4cbd5987c9f6ed6c3a3b88653c5576b56
```

5.3 Verify running containers

You will now check that all the containers are running:

```
[demouser@centos ~]$ docker ps
CONTAINER ID   IMAGE                                  COMMAND                  CREATED
STATUS                                  PORTS                                NAMES
fb6e514b2415   ibmcom/verify-access-dsc:10.0.2.0      "/sbin/bootstrap.sh"     2
minutes ago   Up About a minute (unhealthy)    9443-9444/tcp                      isvadsc
0457846410d8   ibmcom/verify-access-runtime:10.0.2.0  "/sbin/bootstrap.sh"     2
minutes ago   Up 2 minutes (unhealthy)         9080/tcp, 9443/tcp
isvaruntime
166d29f8f0e5   ibmcom/verify-access-wrp:10.0.2.0      "/sbin/bootstrap.sh"     2
minutes ago   Up 2 minutes (unhealthy)         9080/tcp, 127.0.0.3:443->9443/tcp
isvawrprp1
06de4ae80de4   ibmcom/verify-access:10.0.2.0          "/sbin/bootstrap.sh"     2
minutes ago   Up 2 minutes (healthy)           443/tcp, 127.0.0.2:443->9443/tcp
isvaconfig
dbe1c0d2963e   ibmcom/verify-access-openldap:10.0.2.0   "/container/tool/run…"   2
minutes ago   Up 2 minutes                     389/tcp, 127.0.0.2:1636->636/tcp    openldap
b1860f74b75b   ibmcom/verify-access-postgresql:10.0.2.0   "/sbin/bootstrap.sh"   2
minutes ago   Up 2 minutes                     5432/tcp
postgresql
```

The Verify Access containers have a health-check built in which is reported by Docker in its output. You will see that the *isvaconfig* container reports as healthy soon after it is started but the other Verify Access containers remain unhealthy. This is because they are waiting for a configuration snapshot which has not yet been created.

You can see this clearly if you look at the logs:

```
[demouser@centos ~]$ docker logs isvawrprp1
{"instant":{"epochSecond":1624627961},"threadId":"1","level":"INFO","loggerName":"system",
"component":"bootstrap","source":{"file":"/sbin/bootstrap.sh"}, "content":"WGAWA0969I   No
configuration snapshot detected. The container will wait for a configuration snapshot to
become available."}
```

The OpenLDAP container can be verified by running an LDAP search on it (like you have done multiple times in the previous sections). This time you'll bind as the *cn=root,secAuthority=Default* user:

```
[demouser@centos ~]$ docker exec -- openldap ldapsearch -H ldaps://localhost:636 -D
cn=root,secAuthority=Default -w Passw0rd -b dc=ibm,dc=com objectclass=*
# extended LDIF
…
# ibm.com
dn: dc=ibm,dc=com
objectClass: top
objectClass: dcObject
objectClass: organization
o: Example Inc.
dc: ibm
…
# numEntries: 2
```

The PostgreSQL container can be verified by running an SQL query against one of the Verify Access tables:

```
[demouser@centos ~]$ docker exec -- postgresql psql -U postgres -p 5432 isva -c "select *
from OAUTH20_TOKEN_CACHE;"
token_id | type | sub_type | date_created | date_last_used | lifetime | …
----------+------+----------+-------------+---------------+----------+-------------+…
(0 rows)
```

The Verify Access system is ready for activation and initial configuration.

<u>The services for your Verify Access system are now started.</u>
<u>The next step is to perform configuration using the LMI.</u>

6 Verify Access initial configuration

In this section you will perform initial configuration of the Verify Access environment which you have just created. This will include obtaining and applying a 90-day trial license to the system.

> If you have access to the Verify Access activation files, you can apply these to the system instead of following the trial activation steps. This will give you a system which does not expire.

6.1 Connect to Local Management Interface of Configuration Container

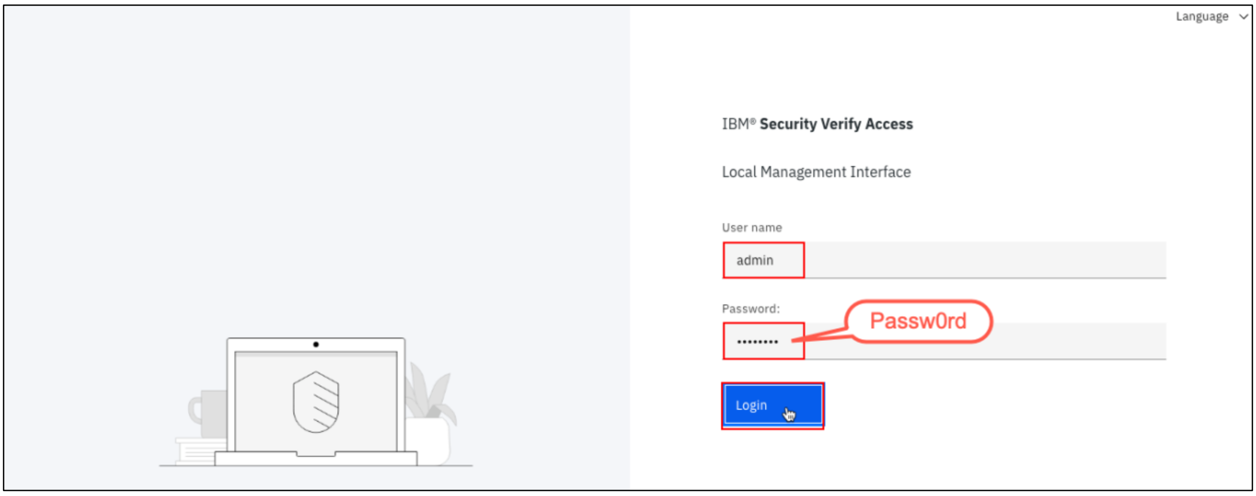
In addition to using the **docker ps** command, you can also get the port mappings for a container using the *docker port* command:

```
[demouser@centos ~]$ docker port isvaconfig
9443/tcp -> 127.0.0.2:443
```

From this, you can see that the Local Management Interface (LMI) port (*9443*) of the *isvaconfig* container is mapped from *127.0.0.2* port *443.* For convenience, it is assumed that the *127.0.0.2* IP address is mapped to host *lmi.iamlab.ibm.com* in the */etc/hosts* file.

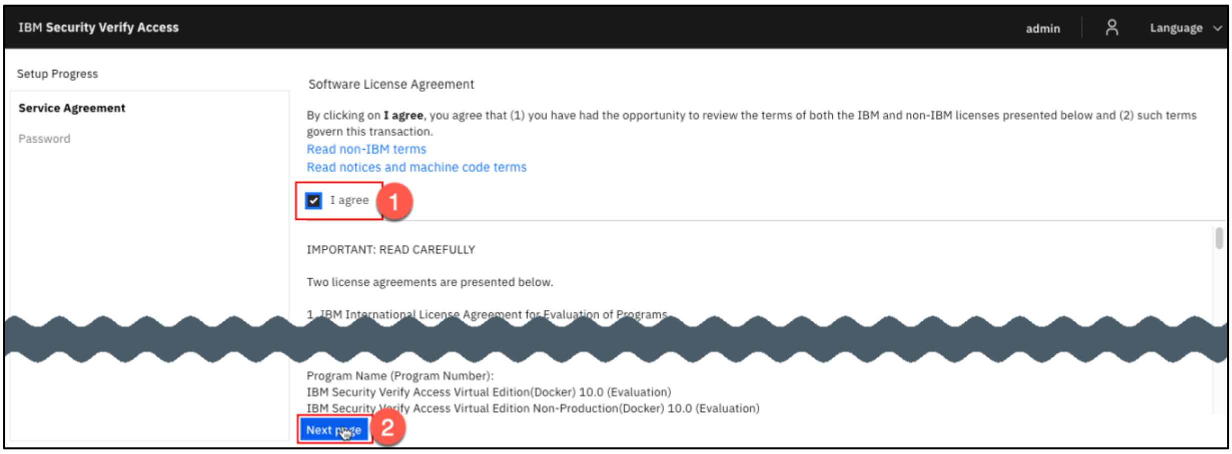
Using the Firefox browser in the test machine, navigate to the following URL: **https://lmi.iamlab.ibm.com**.

If you see a "Connection not secure" warning, click **Advanced→Add Exception...->Confirm Security Exception** to add an exception.

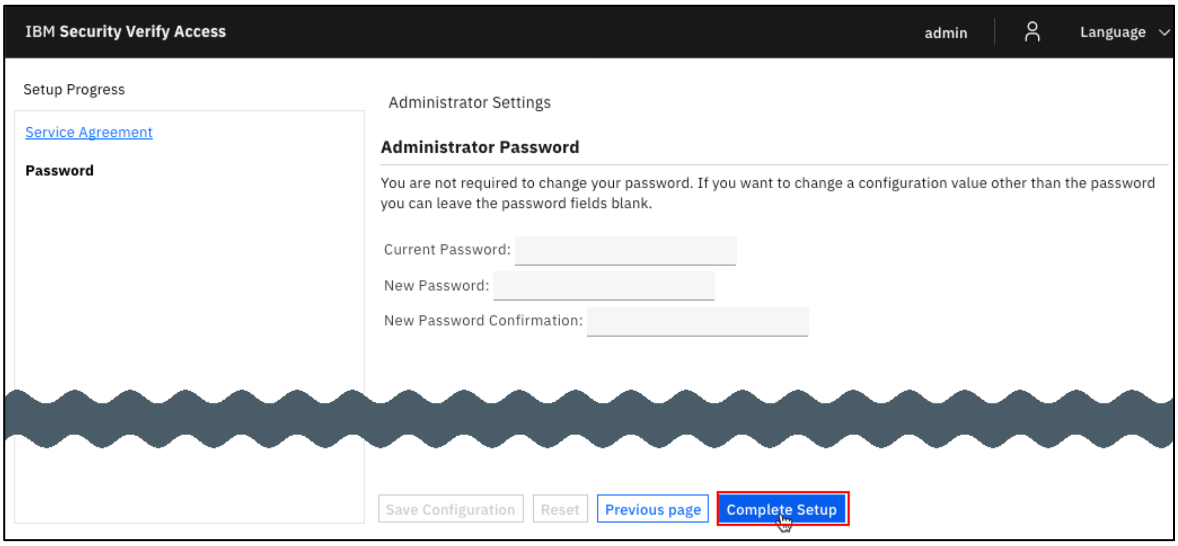


Enter **admin** as the username and **Passw0rd** as the password. Then click **Login**.

> The admin password was set using the ADMIN_PWD environment variable passed into the docker run command for the configuration container. If this wasn't done, the default password (admin) would be set.



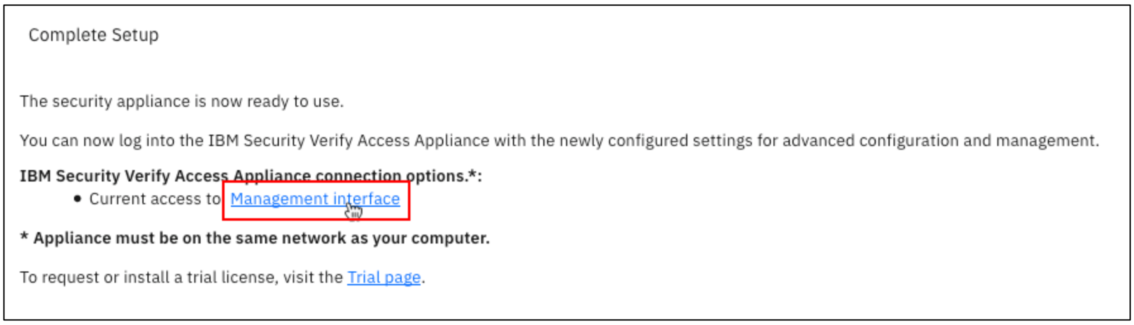
You must read and accept the Software License Agreement. To agree, click **I agree** and then click **Next page**.



On this page you can optionally change the admin password that was set via environment variables. This might be important if you want to protect against platform administrators discovering the password.

Click **Complete Setup** to complete the wizard.

At this point the Local Management Interface (LMI) is restarted. The following page is displayed:



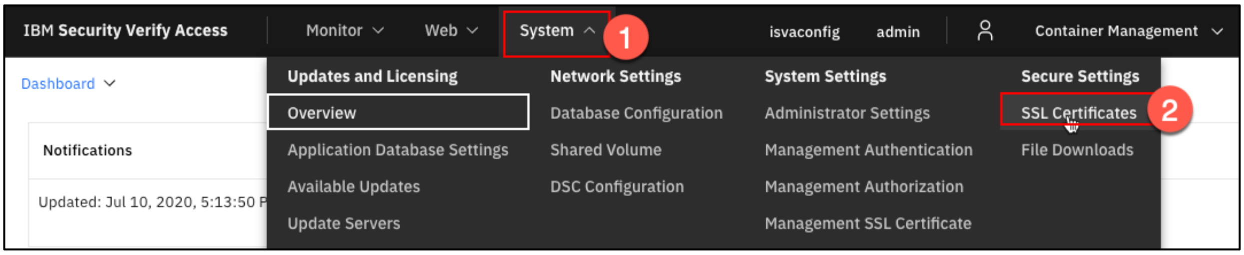
Click on the **Management Interface** link to re-connect to the LMI. If the connection fails, retry after a few seconds to give the LMI a chance to restart.

> There is also a link here which will take you to the trial page to request a license. However, you will visit this page from the main LMI screens later.

6.2 Load Certificates for LDAP and Database connections

The first configuration that needs to be completed is to load the public certificates that will verify the LDAP and Runtime Database connections.

> The Runtime Database (and its certificate configuration) is only required when activating the AAC or Federation add-on modules. However, when using a trial license, all add-ons are enabled and so we need to have the Runtime Database configured before we activate the trial.

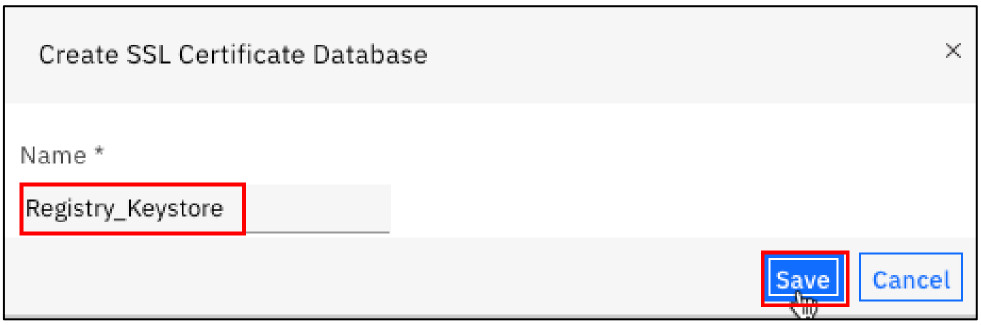


Navigate to **System→Secure Settings: SSL Certificates** on the mega-menu.

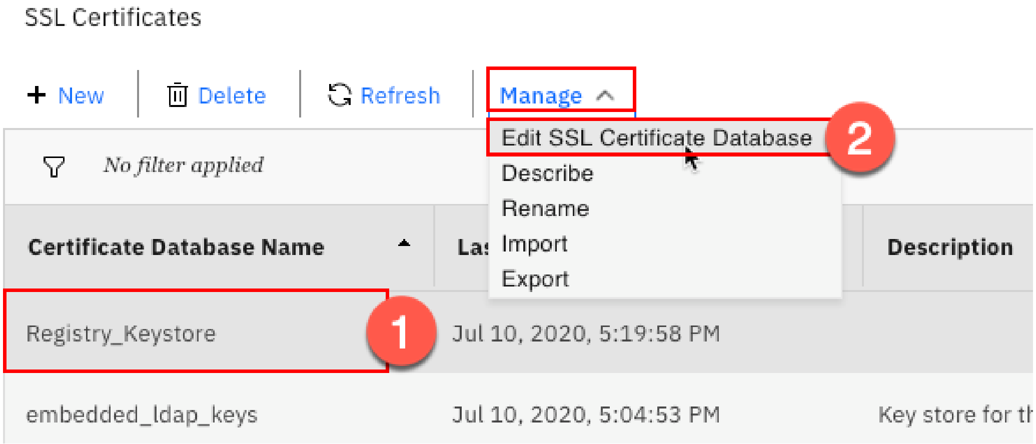
6.2.1 Load LDAP Certificate

You will create a new key store and add the OpenLDAP certificate into it by loading it directly from the LDAPS port of the *openldap* container. You can then refer to this key store when configuring LDAP connections.

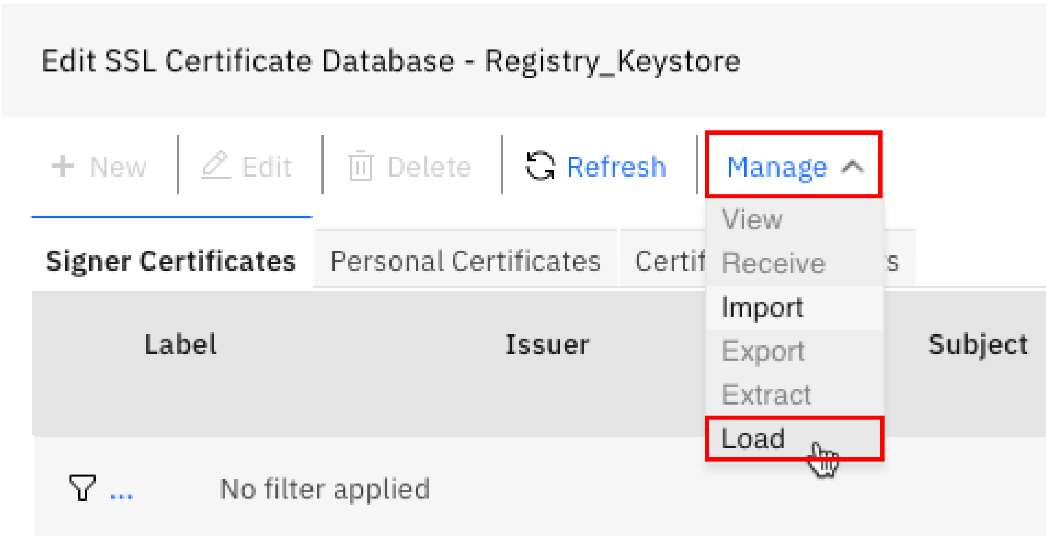
Click **New** to create a new key store.



Enter **Registry_Keystore** as the *Name* and click **Save**.



Select the **Registry_Keystore** that you just created. Click **Manage** and select **Edit SSL Certificate Database** from the drop-down menu. An overlay is opened:



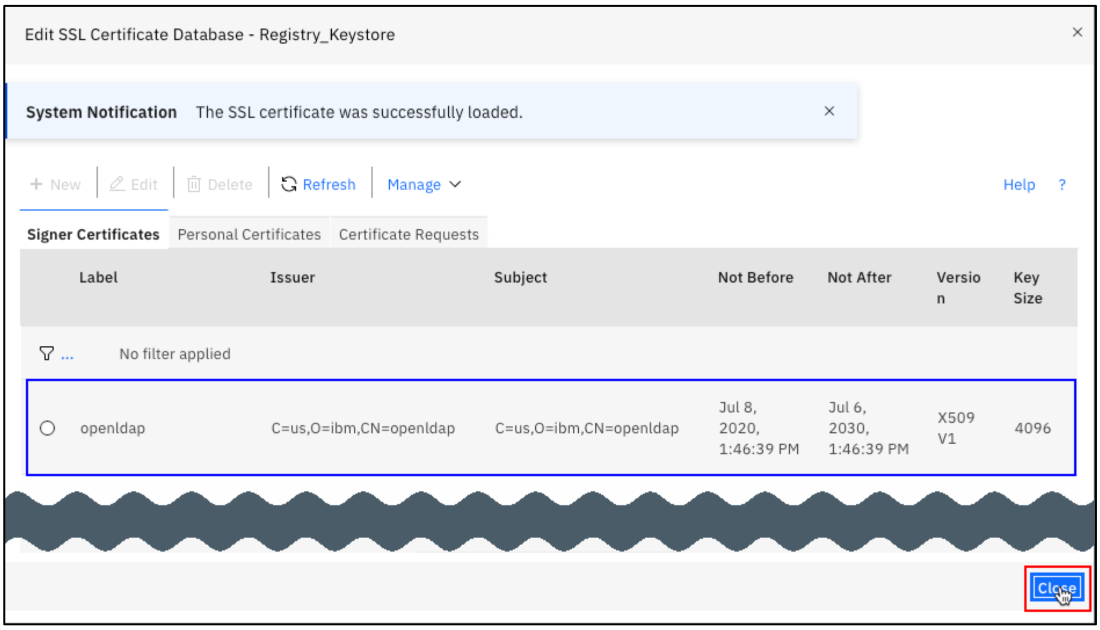
Click **Manage** and select **Load** from the drop-down menu.

Enter **openldap** as the *Server* and **636** as the *Port*. Enter a label and then click **Load**.

The *isvaconfig* container makes a TLS connection to the *openldap* container on port *636* and saves the presented server certificate to the key store.

> The *openldap* hostname is resolved by the container platform DNS services. With native Docker, container name (or specified container hostname) resolves to the container IP address on the local docker network. With Kubernetes, the name of the Service (optionally qualified by the namespace) is used.

The certificate is added to the key store:

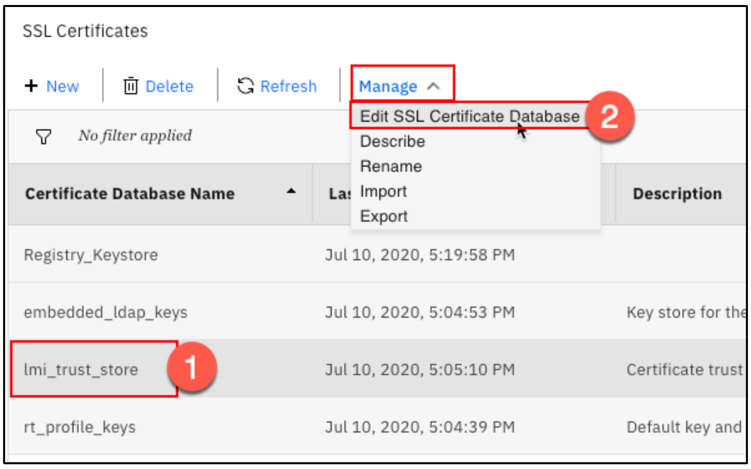


Click **Close** to close the overlay.

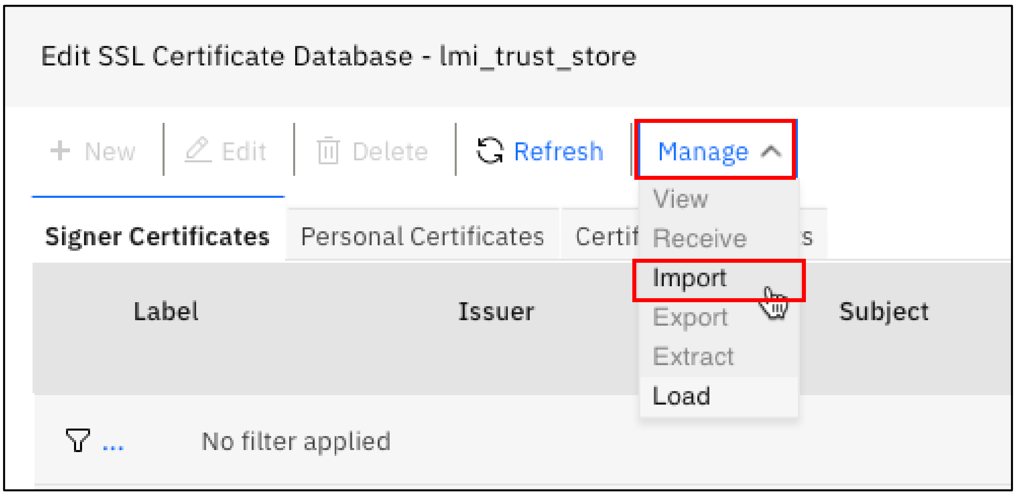
6.2.2 Import Database Certificate to LMI and Runtime key stores

The PostgreSQL secure port certificate cannot be loaded directly from the database port. Instead, you will need to import the certificate from the local directory where the certificates were generated.

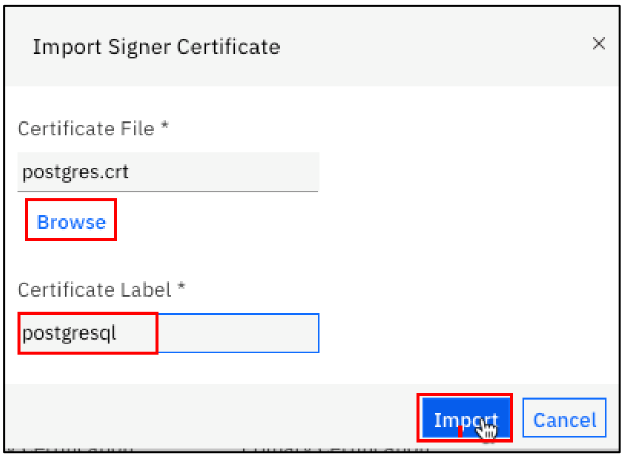
The Database certificate needs to be loaded to the LMI key store, so that it can validate the connection, and to the Runtime key store so that it can make runtime connections.



Select the **lmi_trust_store** from the key store list. Click **Manage** and select **Edit SSL Certificate Database** from the drop-down menu. An overlay is opened.

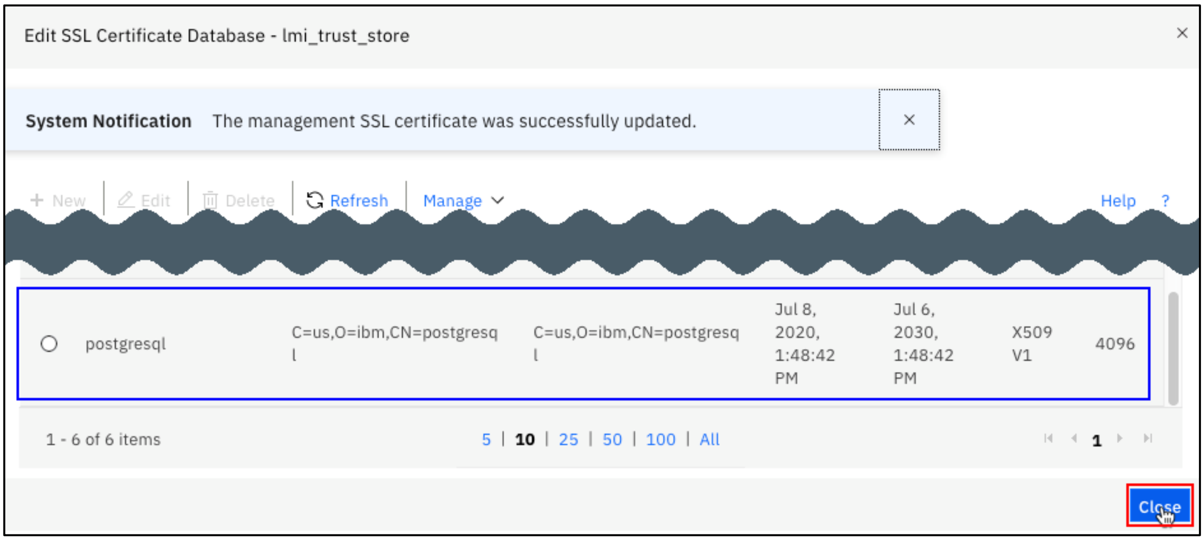


Click **Manage** and then select **Import**.

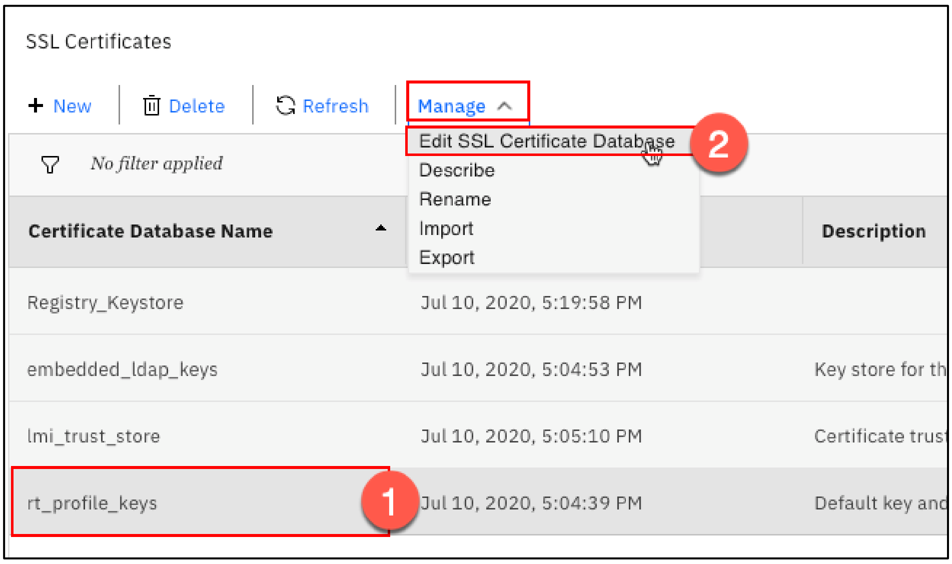


Click **Browse**. Select file **git/container-deployment/local/dockerkeys/postgresql/postgres.crt**.

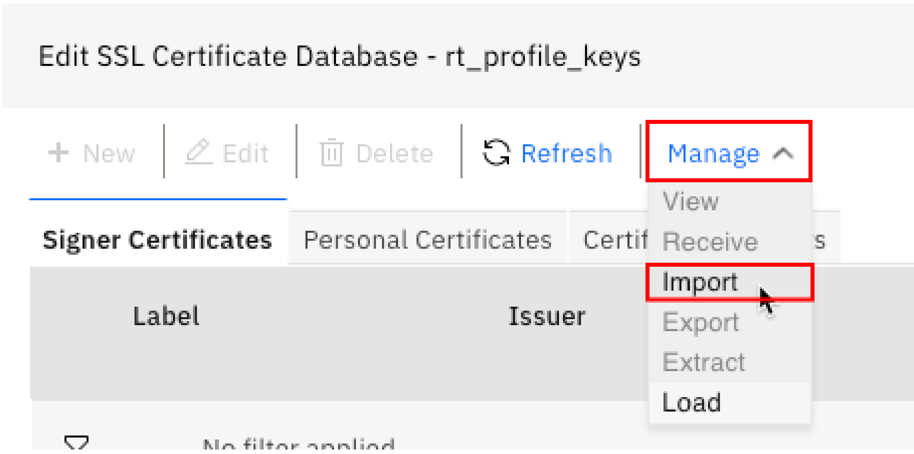
Click **Import**. The certificate file is uploaded and added to the key store:



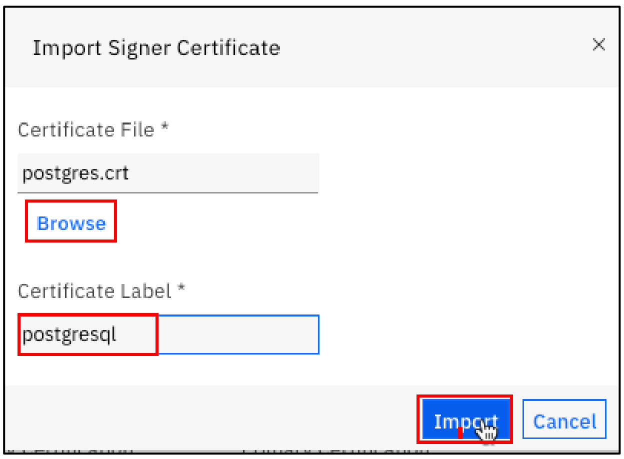
Click **Close**.



Select the **rt_profile_keys** from the key store list. Click **Manage** and select **Edit SSL Certificate Database** from the drop-down menu. An overlay is opened.

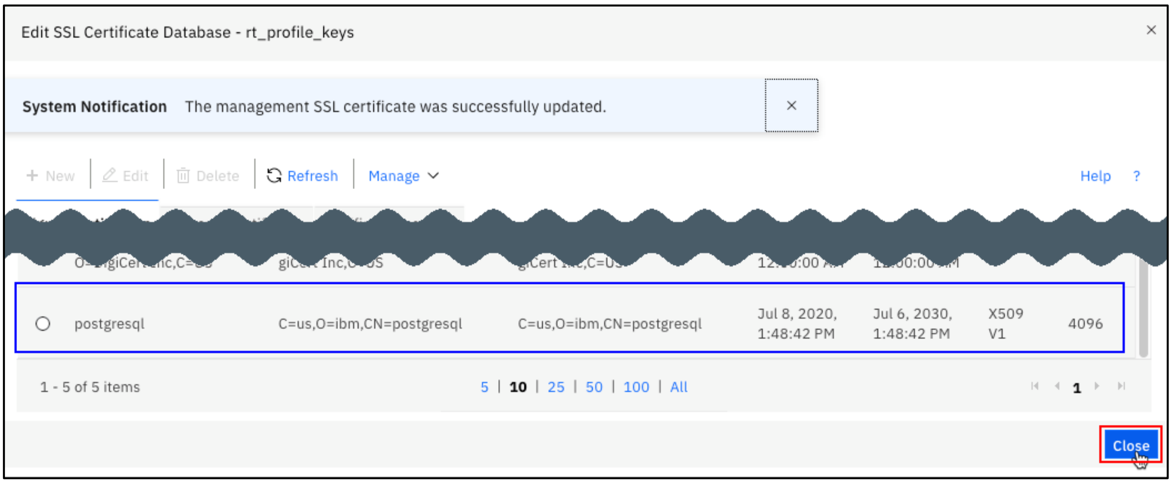


Click **Manage** and then select **Import**.

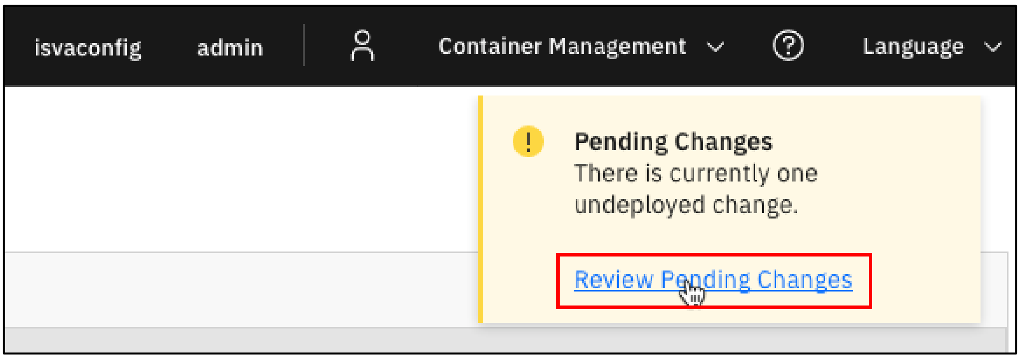


Click **Browse**. Select file **git/container-deployment/local/dockerkeys/postgresql/postgres.crt**.

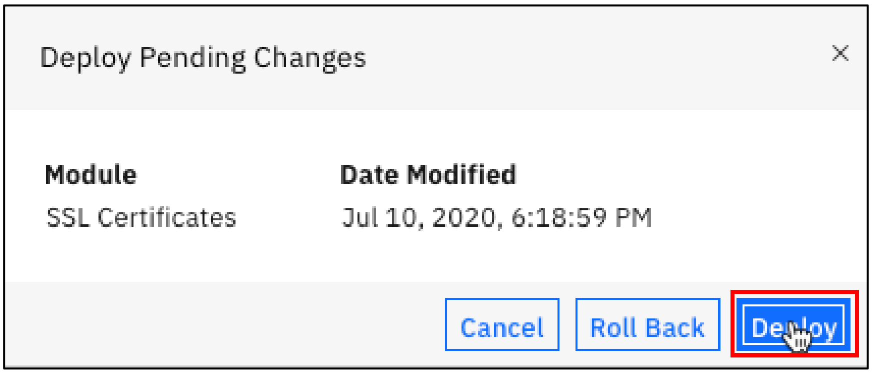
Click **Import**. The certificate file is uploaded and added to the key store:



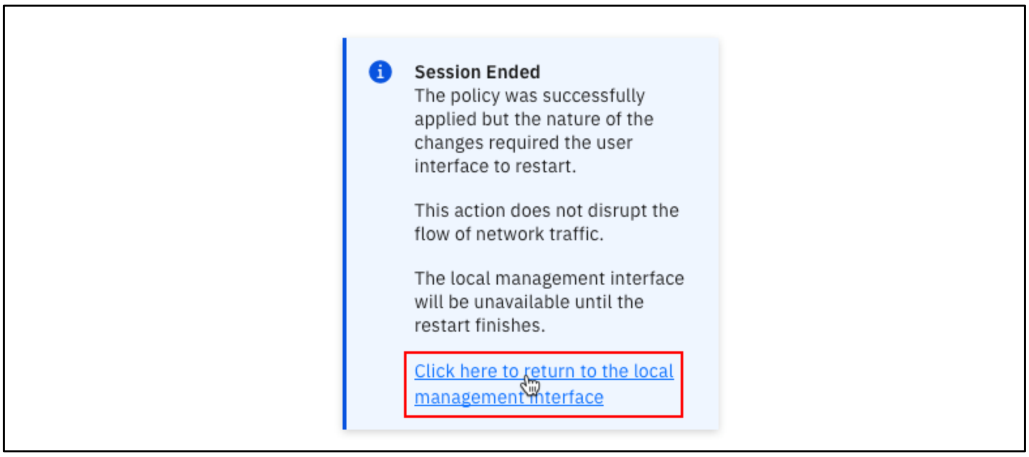
Click **Close**.



Click the link in the yellow warning notice to **review pending changes**.



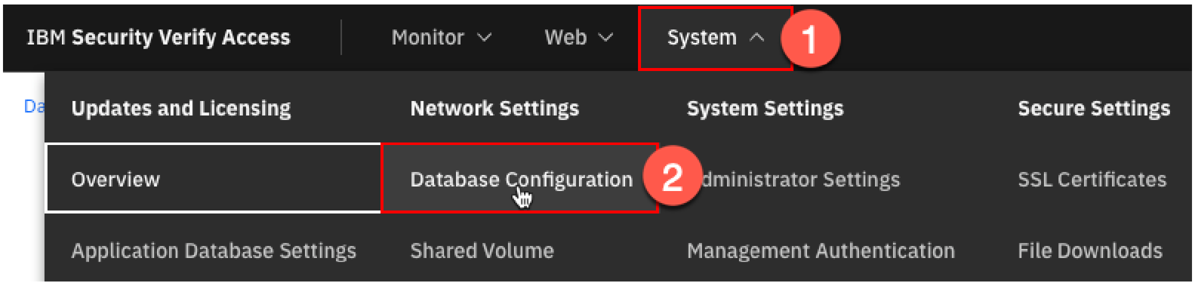
Click **Deploy** to deploy the changes to the Configuration Container. These changes require that the LMI is restarted so you will see this message:



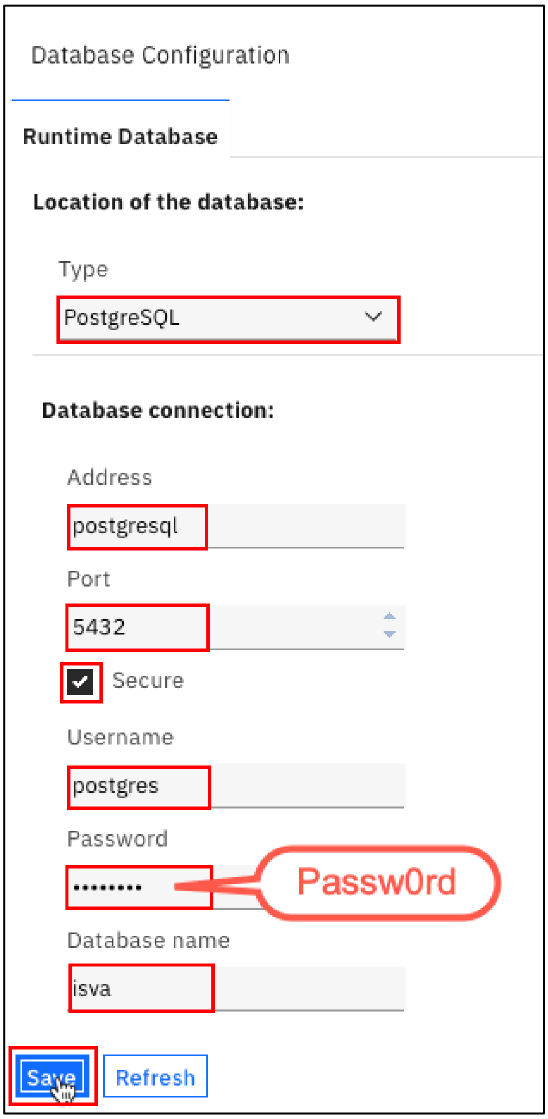
Click the link to reconnect to the LMI. If the connection fails, you may need to wait a few seconds and try again (to give the LMI time to restart).

6.3 Configure Runtime Database

Now that the Runtime Database certificate has been loaded, you can configure the Runtime Database connection. There is a link for this on the LMI Dashboard but it is also available in the mega menu.



Navigate to **System→Network Settings: Database Configuration** on the mega-menu.



Configure the database connection. Note that these parameters match what was configured during creation of the PostgreSQL container.

Select **PostgreSQL** as the *Type*.

Enter **postgresql** as the *Address*.

Enter **5432** as the *Port*.

Select the **Secure** check box.

Enter **postgres** as the *Username* and **Passw0rd** as the *Password.*

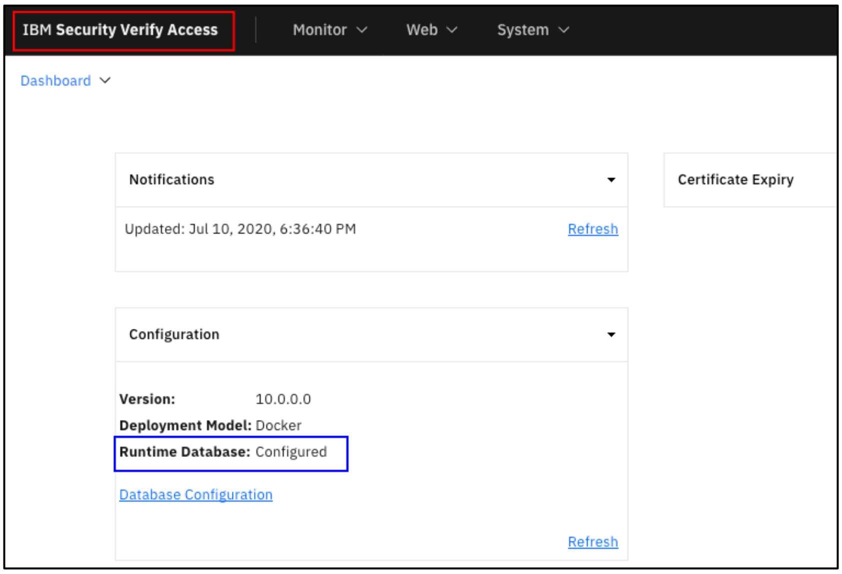
Enter **isva** as the *Database name*.

Click **Save** to save the configuration.

> The database connection is checked at this point. If the connection or bind details are incorrect a warning will be shown.
>
> The connection will also fail if the certificate loaded in the LMI keystore does not validate the PostgreSQL certificate.

Deploy the changes using the link in the yellow warning message.

Reconnect to the LMI (once it has restarted). The Database should now show as configured:

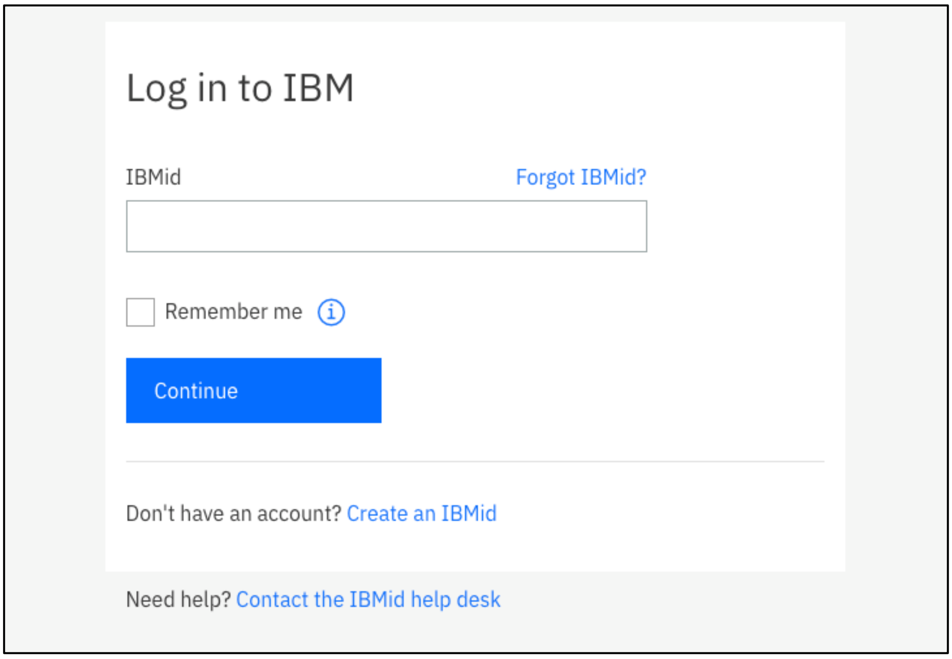


6.4 Obtain Trial License from the Verify Access Trial Center

Now that the database connection is configured, you can apply a trial license to the Verify Access environment to activate all of the appliance functionality.

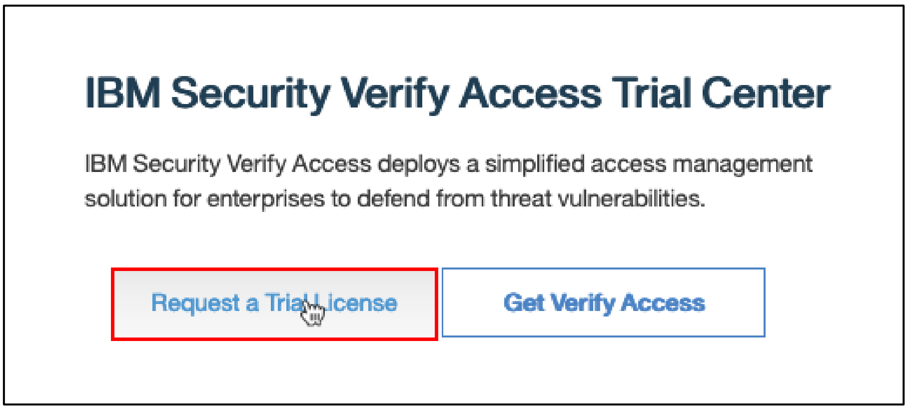
Open a new browser tab and navigate to: **https://ibm.biz/isamtrial**

An IBM Login page is shown.

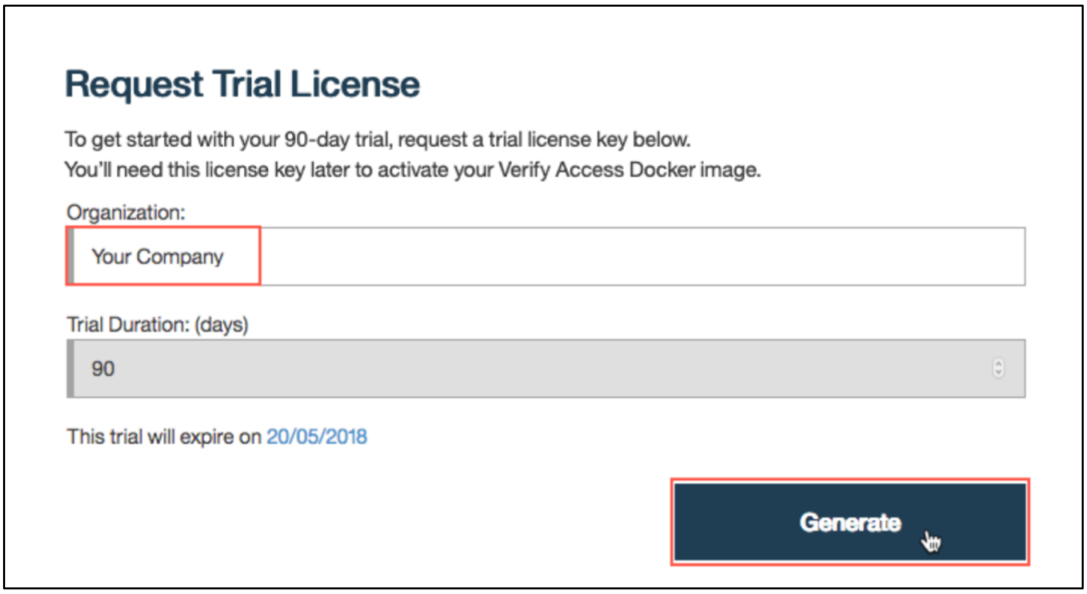


You will need to authenticate using your IBMid. If you don't have an IBMid you can create one for free.

After you have completed the registration and/or sign in process, you are taken to the Verify Access trial center.

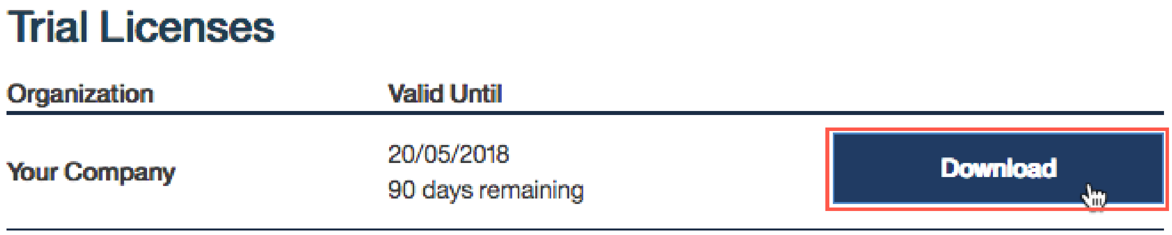


Click **Request a Trial License**.



Enter the name of your *Organization* and then click **Generate**.

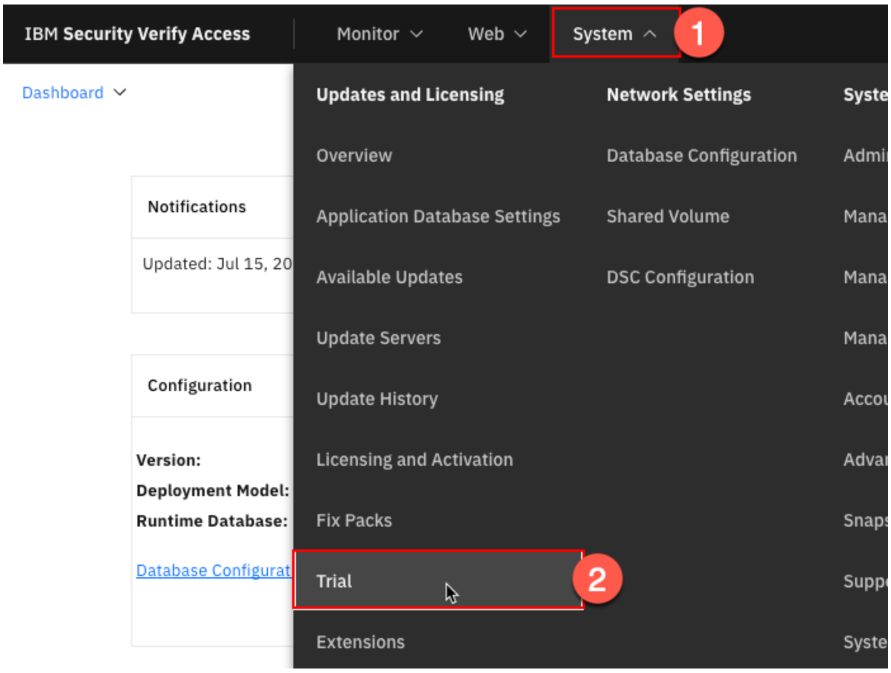
A trial license certificate is generated. It is listed on the page:



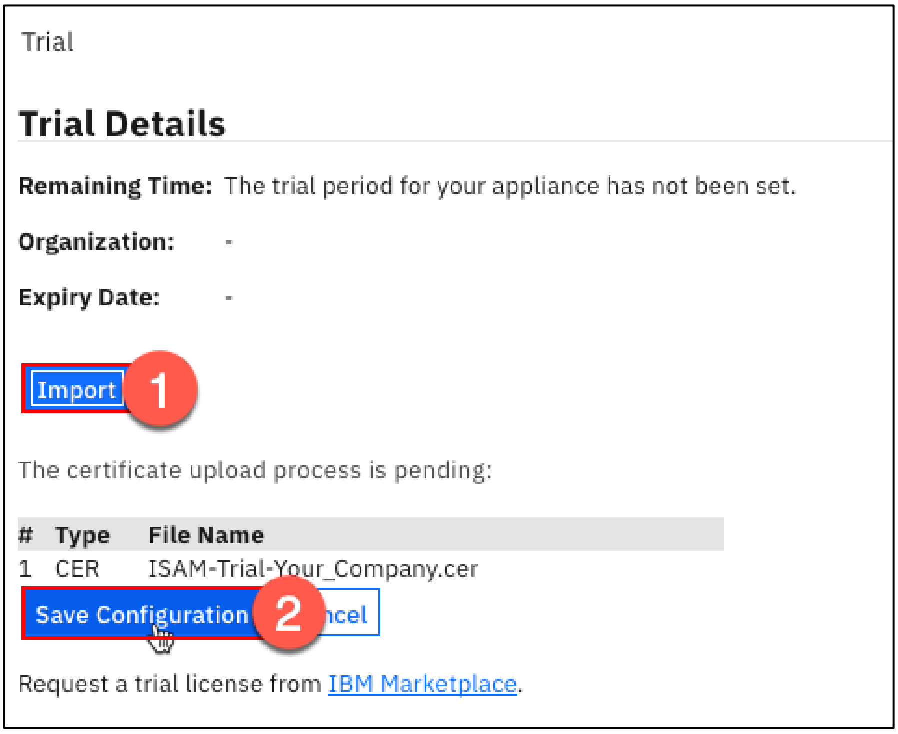
Click **Download** and save the trial certificate locally (remembering where you saved it). The Desktop is a good choice.

6.5 Apply trial license certificate

Close the Trial Center browser tabs and return to the LMI.



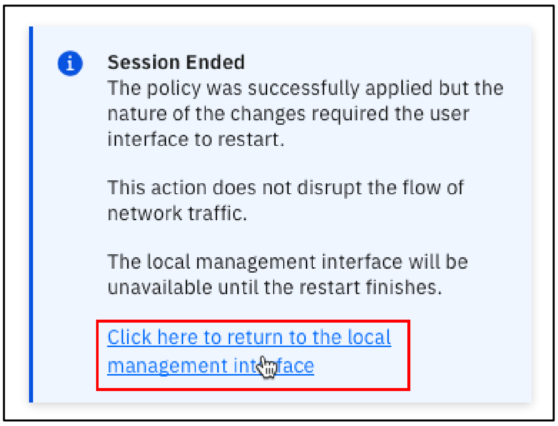
Navigate to **System→Updates and Licensing: Trial** on the mega-menu.



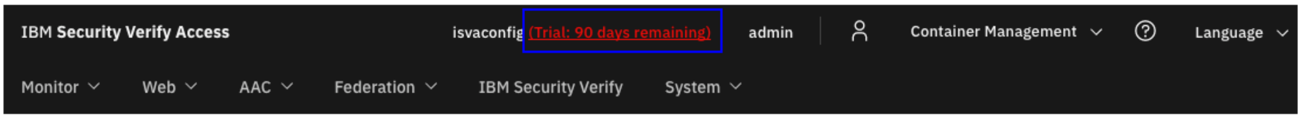
On the trial page, click **Import**.

Locate and select the license certificate file that you downloaded from the Trial Center. This is then displayed on the page.

Click **Save Configuration**. The license certificate is uploaded and applied. You may not see any activity on the LMI for up to a minute, but it will eventually show this screen as the LMI is restarted:



This restart may take a little while to complete as the base and all add-ons are activated. When the restart is complete, and you are reconnected to the LMI, you will see that all capabilities are available. A notice in the header bar shows the trial time remaining.



> If you need to, you can download the trial license certificate again from the IBM Marketplace by returning to the license center.

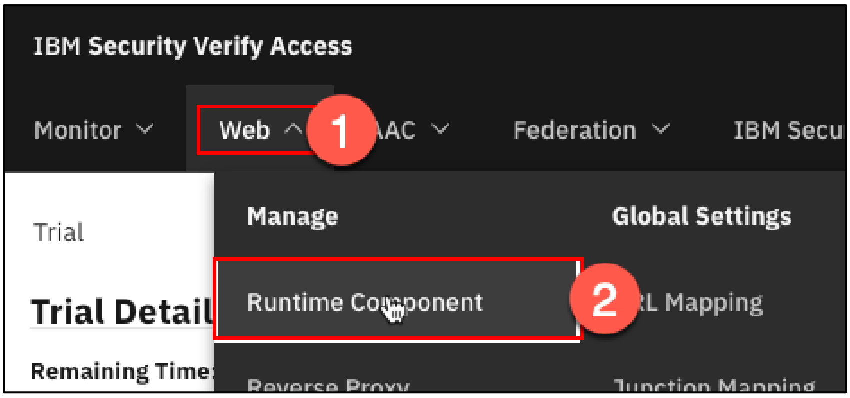
7 Verify Access Base Configuration

In this section you will perform configuration of the Verify Access base including:

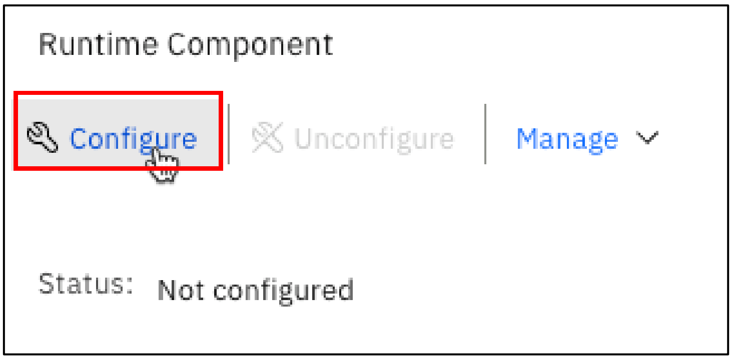
- Configuration of Verify Access Base Runtime Component
- Configuration of a Reverse Proxy instance (rp1)
- Creation of Users (Emily and Chuck)

7.1 Configure Verify Access Base Runtime Component

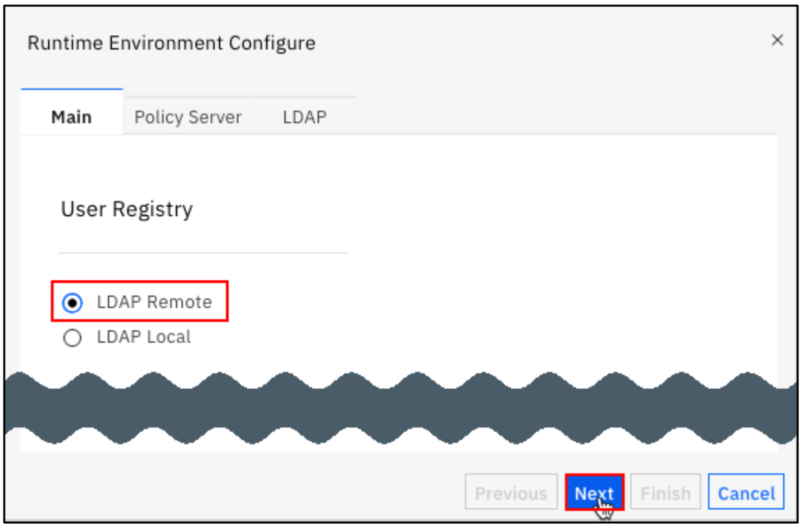
You will now configure the Verify Access Base Runtime. This sets up the LDAP for storage of Verify Access-specific information (in *secAuthority=Default* suffix) and also creates the initial Policy Database for storage of Access Control Lists and Protect Object Policies.



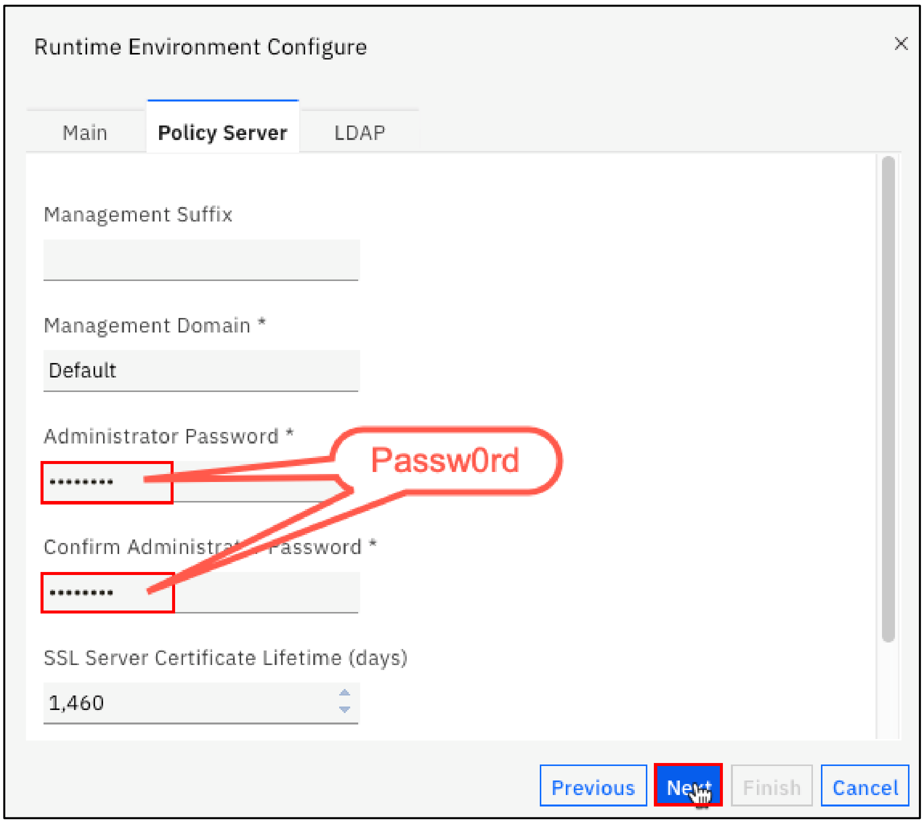
Navigate to **Web→Manage: Runtime Component** in the mega-menu.



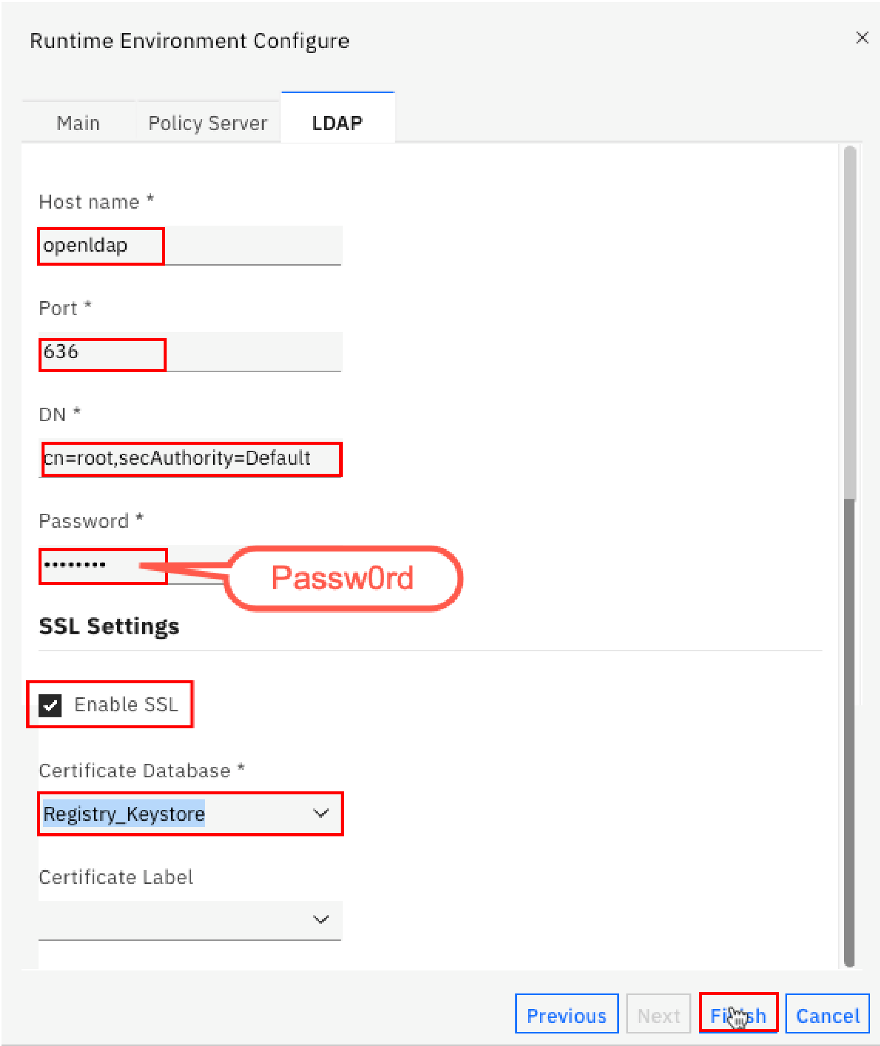
Click **Configure**.



Select radio button for **LDAP Remote** and click **Next**.



Enter **Passw0rd** for *Administrator Password* (and *Confirm Administrator Password*). Click **Next**.



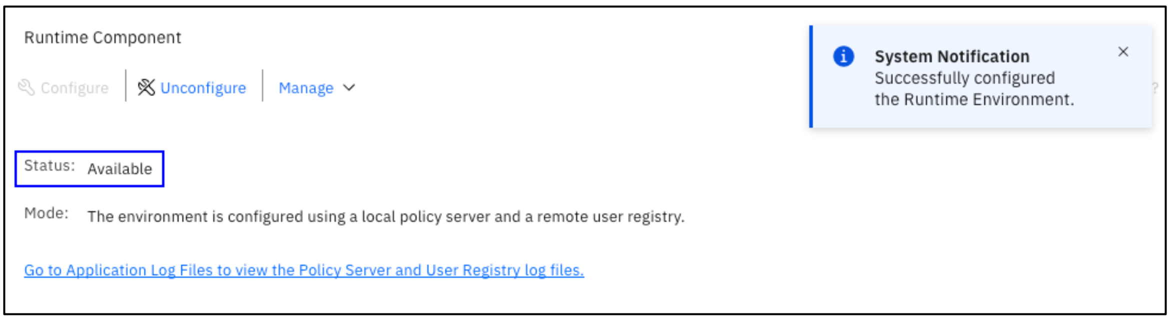
Enter **openldap** as the *Host name* and change the port to **636**.

Change the *DN* to **cn=root,secAuthority=Default** and enter **Passw0rd** as the *Password*.

Check the **Enable SSL** checkbox and select **Registry_Keystore** from the *Certificate Database* drop-down list.

Click **Finish**.

The Verify Access Base Runtime Component is now configured. Assuming everything is successful, it will show as *Available*:

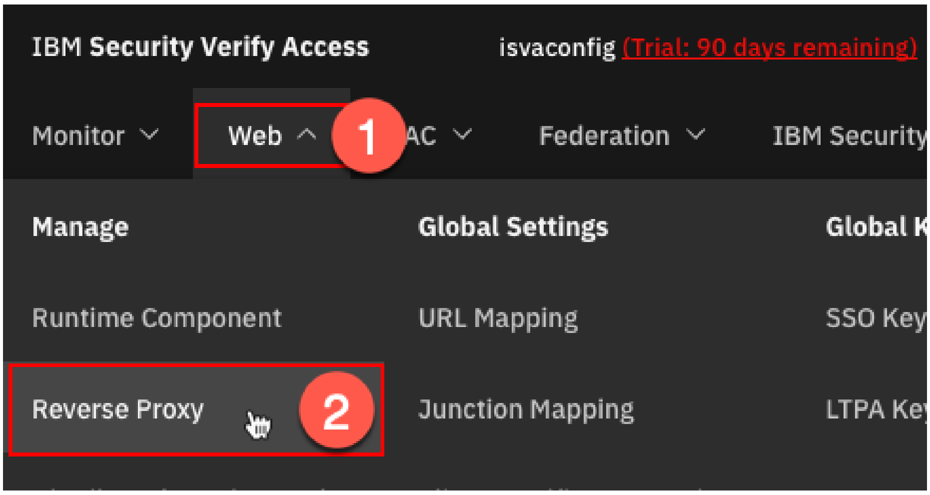


7.2 Configure "rp1" Reverse Proxy Instance

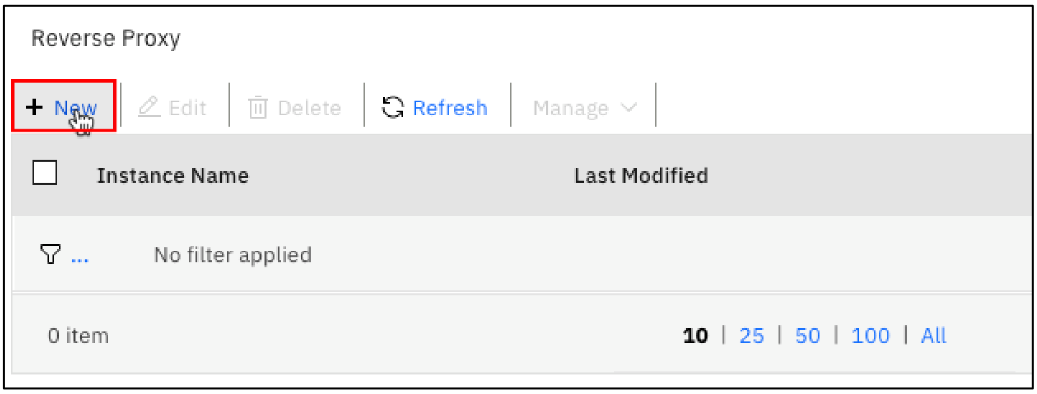
You will now configure a Reverse Proxy instance.

In this environment you will name your Reverse Proxy instance *rp1*. This name must match the *INSTANCE* parameter given when creating the Reverse Proxy container. If there is a mismatch, the Reverse Proxy container will give an error on reading the configuration because it can't find a matching instance.

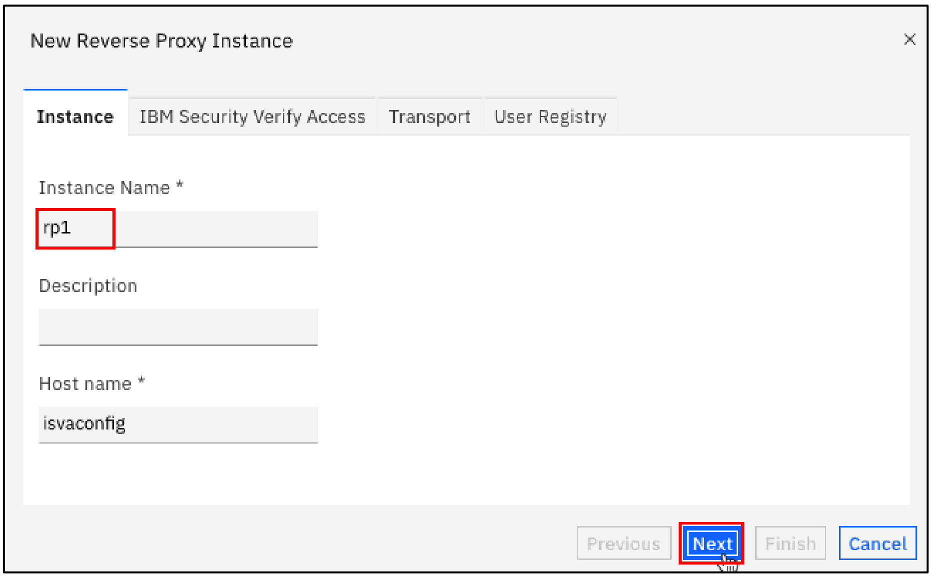
7.2.1 Create Instance



Navigate to **Web→Manage: Reverse Proxy** in the mega-menu.

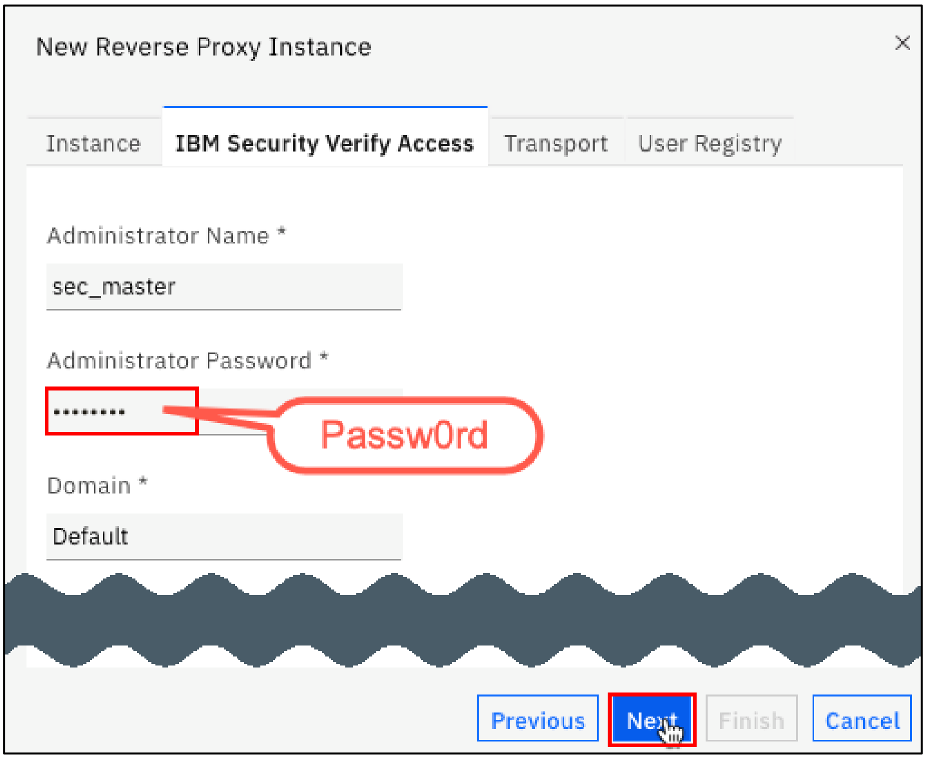


Click **New**.

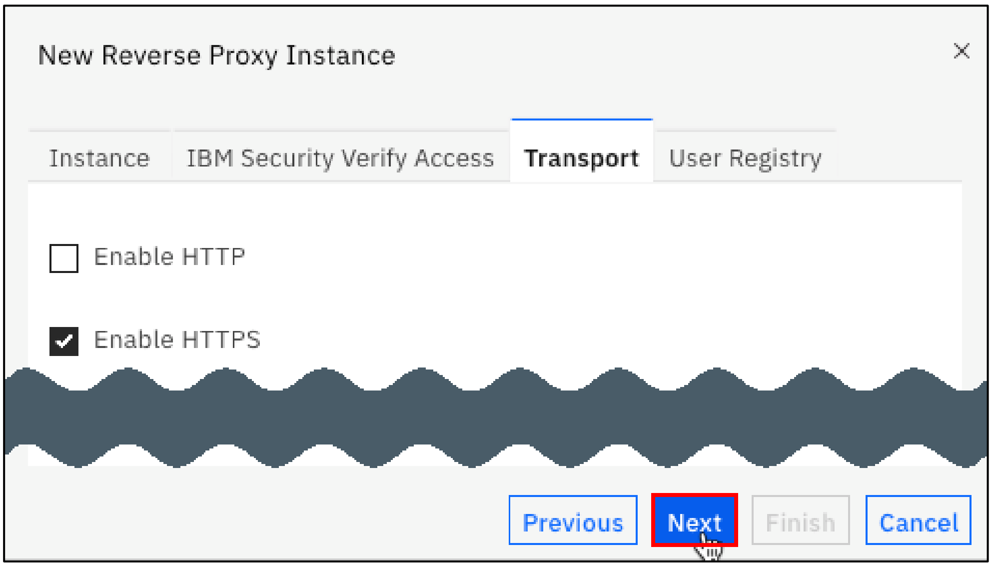


Enter **rp1** as the *Instance Name* and click **Next**.

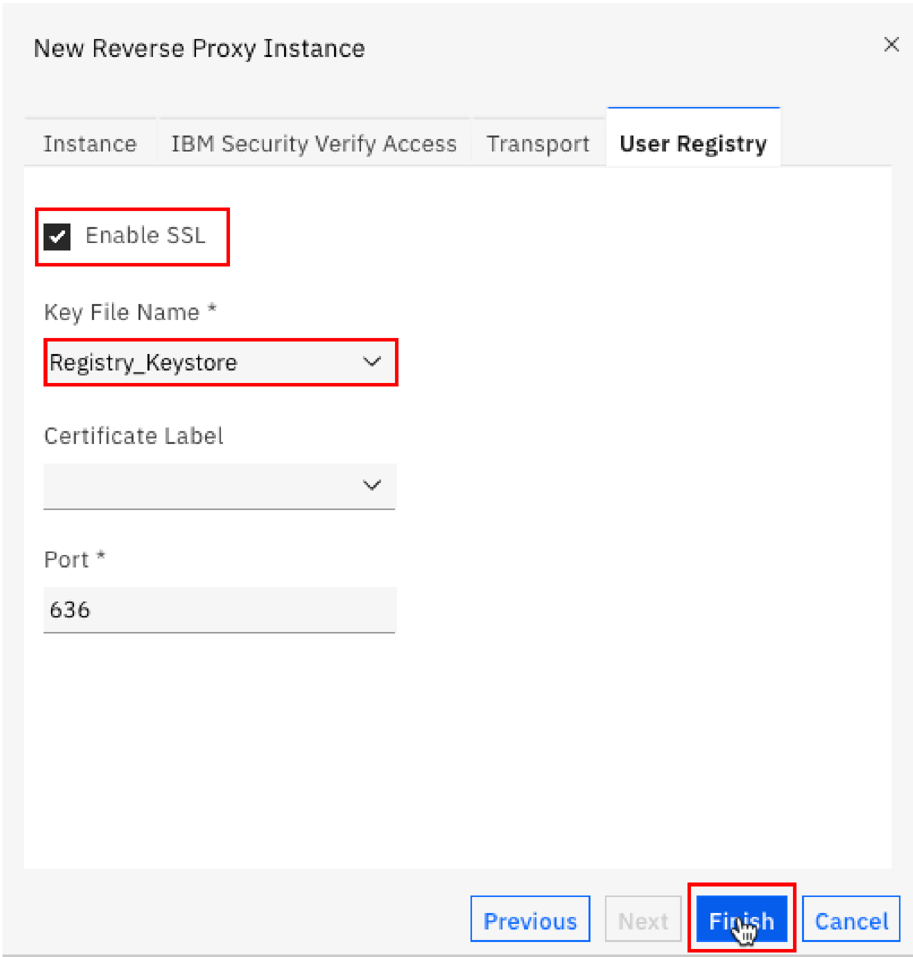
> Note that the *Host name* is set to *isvaconfig*. There is no need to change this value when working in a Docker environment because there is no runtime communication between the Reverse Proxy and a central Policy Server like there is in a traditional Verify Access system.



Enter **Passw0rd** as the *Adminstrator Password*. This matches the password given during configuration of the Verify Access Base Runtime Component. Click **Next**.



You want your Reverse Proxy to listen only on HTTPS port so defaults are good. Click **Next**.



Select the **Enable SSL** checkbox.

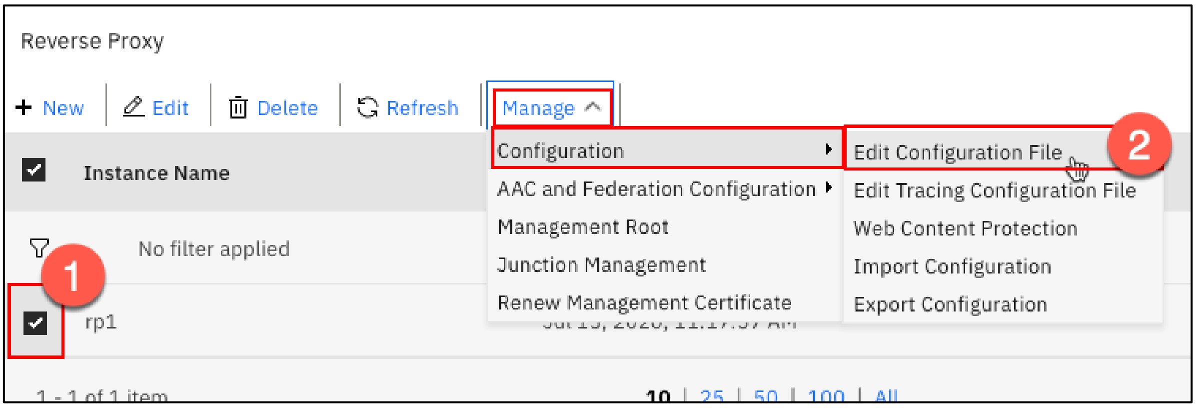
Select **Registry_Keystore** from the *Key File Name* drop-down list.

Click **Finish**. The Reverse Proxy instance is configured.

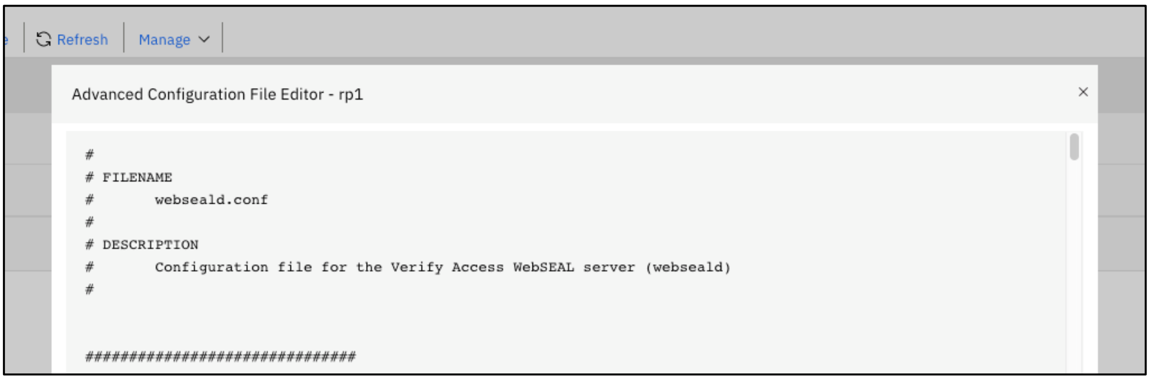
> Note that this step does not make any changes to the Reverse Proxy container. Instead, configuration files for the new instance are created within the configuration container. When these configuration files are later transferred to the Reverse Proxy container (as part of a snapshot), it will use them to run.

7.2.2 Edit Reverse Proxy Configuration File

Most configuration settings must be edited directly in the reverse proxy configuration file.

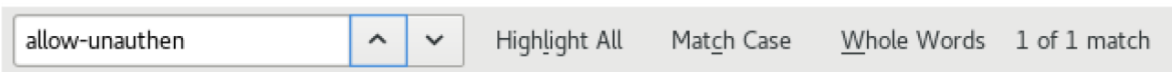


From the list of Reverse Proxy instances, select the checkbox for the *rp1* instance. Click **Manage** and then select **Configuration → Edit Configuration File** from the drop-down menus – as shown above.



The configuration file is then displayed in an editable (pop-up) window.

You can use the browser's "find on page" facility to locate items in the window. If you press "*Ctrl-F*" the search bar shown below will appear at the bottom of the browser window.

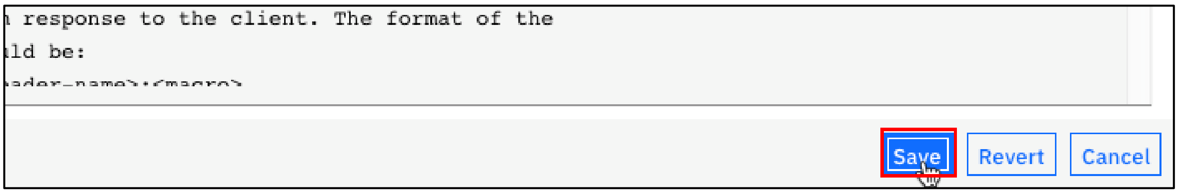


The default behavior for the Reverse Proxy is to challenge the user to login if the user tries to logout after a session has timed out. This login to complete logout user experience is normally not desirable. This behavior can be changed to not challenge the user to login if they try logout of an expired session via a setting in the configuration file.

Use the find utility (enter **allow-unauthenticated** in the *Find:* box) to locate the following section in the configuration file:

```
#----------------------------
# ALLOW UNAUTHENTICATED LOGOUT
#----------------------------
# Set this parameter to 'yes' to allow unauthenticated users to be able
# to request the pkmslogout resource.  If this parameter is set to 'no'
# an unauthenticated user will be requested to authenticate before the
# pkmslogout resource is returned.
allow-unauthenticated-logout = yes
```

Change *allow-unauthenticated-logout* to **yes**.



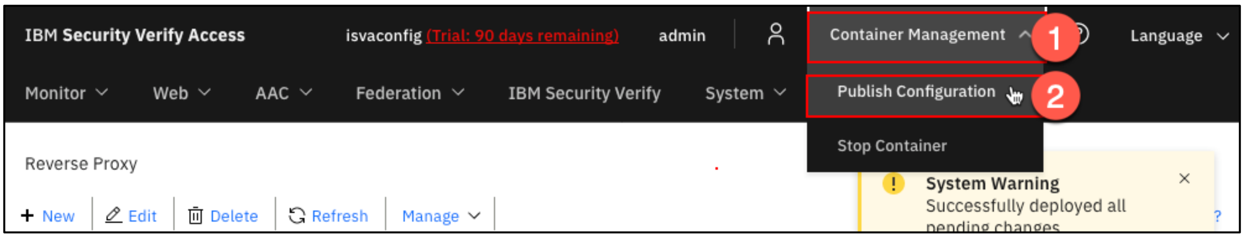
Once the change is complete, press the ***Save*** button in the bottom right corner of the pop-up window to save the changes and close the window.

Deploy the changes using the link in the yellow warning message.

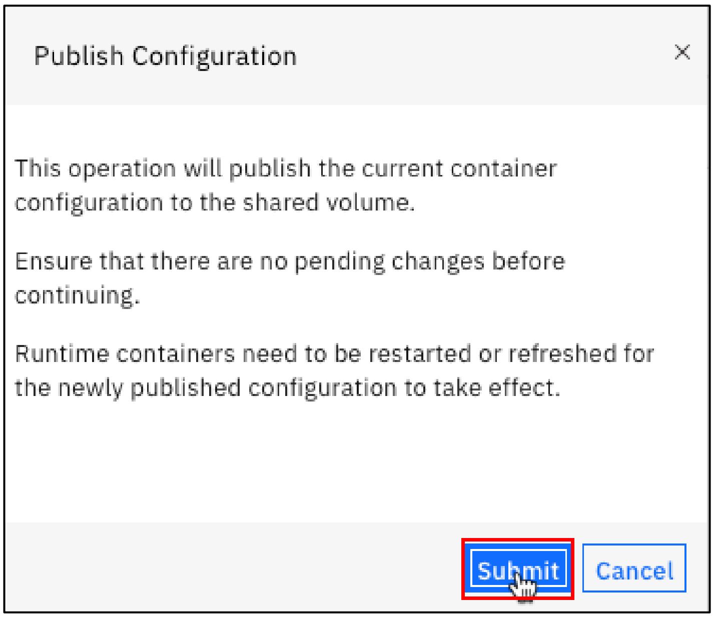
> When the deploy completes, a warning message is shown indicating that the configuration needs to be published and that the *rp1* Reverse Proxy instance needs to be restarted. However, notice that there are no Start/Stop buttons available for the Reverse Proxy. This is because, when running under Docker the process for publishing configuration changes is different than when using an appliance.

7.3 Publish Snapshot

Now that you have completed basic configuration, you need to publish the configuration snapshot to the other Verify Access containers (just the Reverse Proxy container at this point). This publishing of the snapshot is an extra step which is required when running Verify Access under Docker.



Click the **Container Management** item in the header bar and then select **Publish Configuration** from the pop-up menu.



Click **Submit** to confirm the publish operation.

At this point, the Reverse Proxy container, which has been waiting for a configuration snapshot since it was created, now finds the published snapshot in the shared configuration volume. It unpacks this snapshot and applies it. Then it starts the *rp1* Reverse Proxy instance.

You can confirm start up by viewing the Reverse Proxy logs:

```
[demouser@centos ~]$ docker logs isvawrprp1
…
{"instant":{"epochSecond":1624627961},"threadId":"1","level":"INFO","loggerName":"system",
"component":"bootstrap","source":{"file":"/sbin/bootstrap.sh"}, "content":"WGAWA0969I   No
configuration snapshot detected. The container will wait for a configuration snapshot to
become available."}
{"instant":{"epochSecond":1624633089},"threadId":"1","level":"INFO","loggerName":"system",
"component":"bootstrap","source":{"file":"/sbin/bootstrap.sh"}, "content":"WGAWA0970I
The configuration snapshot has become available."}
{"instant":{"epochSecond":1624633089},"threadId":"1","level":"INFO","loggerName":"system",
"component":"bootstrap","source":{"file":"/sbin/bootstrap.sh"}, "content":"WGAWA0971I
Applying the configuration snapshot:
7f27ea1406ad1a55661e487e4144aea88b63d1955874ecb3933a1e5649173598"}
{"instant":{"epochSecond":1624633089},"threadId":"1","level":"INFO","loggerName":"system",
"component":"bootstrap","source":{"file":"/sbin/bootstrap.sh"}, "content":"WGAWA0965I
Starting the WRP Server."}
{"instant":{"epochSecond":1624633090},"threadId":"0xa6cf4740","level":"INFO","loggerName":
"message","content":{"product":"IBM Security Verify Access: Web Reverse Proxy",
"version":"10.0.2.0 (Build 20210610_0134)","copyright":"Copyright (C) IBM Corporation
1994-2021.  All Rights Reserved."}}
{"instant":{"epochSecond":1624633091},"threadId":"0x7f2ca6cf4740","level":"WARNING","logge
rName":"webseald","component":"wwa.server","message_id":"0x38CF0156","source":{"file":"con
fig.cpp","line":6024}, "content":"DPWWA0342W   The configuration data for this WebSEAL
instance has been logged in '\/var\/pdweb\/rp1\/log\/config_data__rp1-webseald-
isvaconfig.log'"}
…
```

The *isvaruntime* and *isvadsc* containers also read this snapshot and start up. However, they are not used in the current configuration and so will sit idle for now.

7.4 Test Configuration

You will now connect to the *rp1* Reverse Proxy instance to test the configuration we just published.

Enter the following command to get the port mapping for the *isvawrprp1* container:

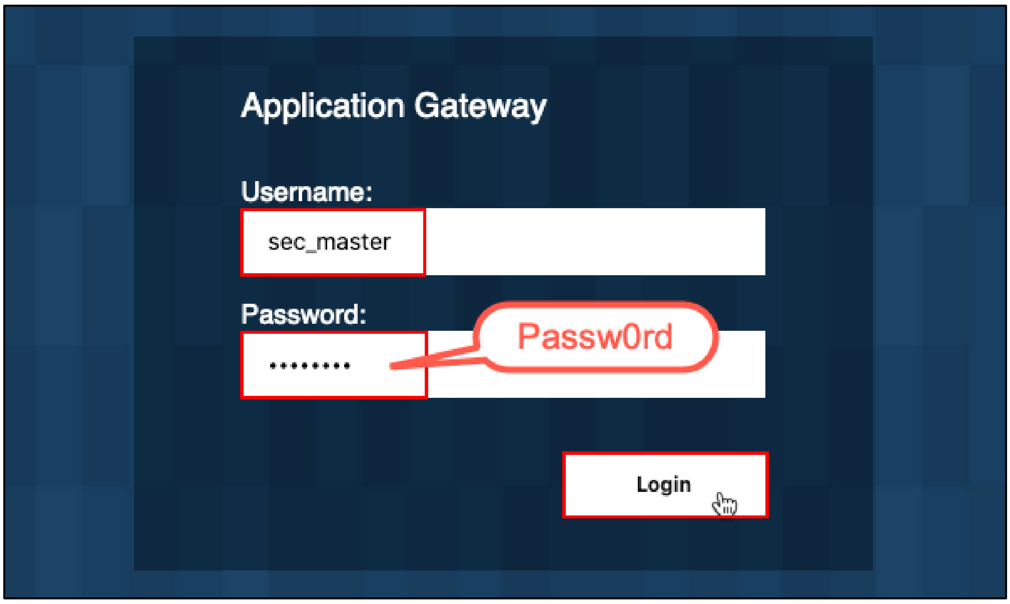
```
[demouser@centos ~]$ docker port isvawrprp1
9443/tcp -> 127.0.0.3:443
```

From this, you can see that the HTTPS listening port (*9443*) of the *isvawrprp1* container is mapped from *127.0.0.3* port *443.* For convenience, it is assumed that the *127.0.0.3* IP address is mapped to host *www.iamlab.ibm.com* in the */etc/hosts* file.

Using the Firefox browser in the test machine, navigate to the following URL: **https://www.iamlab.ibm.com**.

If you see a "Connection not secure" warning, click **Advanced→Add Exception...->Confirm Security Exception** to add an exception.

You should see the Reverse Proxy login page:

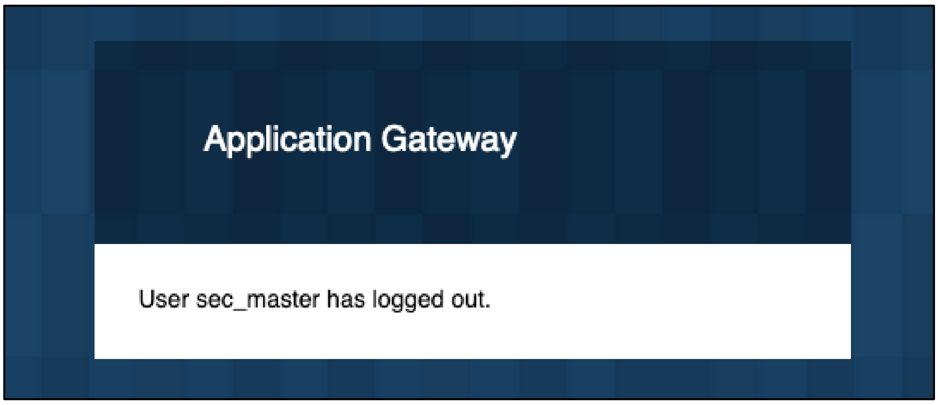


> You only have one user defined at the moment: the *sec_master* user that was defined during configuration of the Verify Access Base Runtime Component.

Enter **sec_master** as the *Username* and **Passw0rd** as the *Password*. Click **Login**.

You should see the Reverse Proxy default home page:

Navigate to URL: **https://www.iamlab.ibm.com/pkmslogout**



7.5 Create Users

You will now create two additional Verify Access users. These will be created in the OpenLDAP (which is the Verify Access primary directory). The user suffix in the OpenLDAP directory is *dc=ibm,dc=com*. This was based on the configuration given when the OpenLDAP container was created.

User and Group management can be performed using the LMI web interface but you're going to use the *pdadmin* command line interface instead. The *pdadmin* interface can be started using a *docker exec* command:

```
[demouser@centos ~]$ docker exec -ti isvaconfig pdadmin
pdadmin>
```

Login as the *sec_master* user:

```
pdadmin> login
Enter User ID: sec_master
Enter Password: Passw0rd
pdadmin sec_master>
```

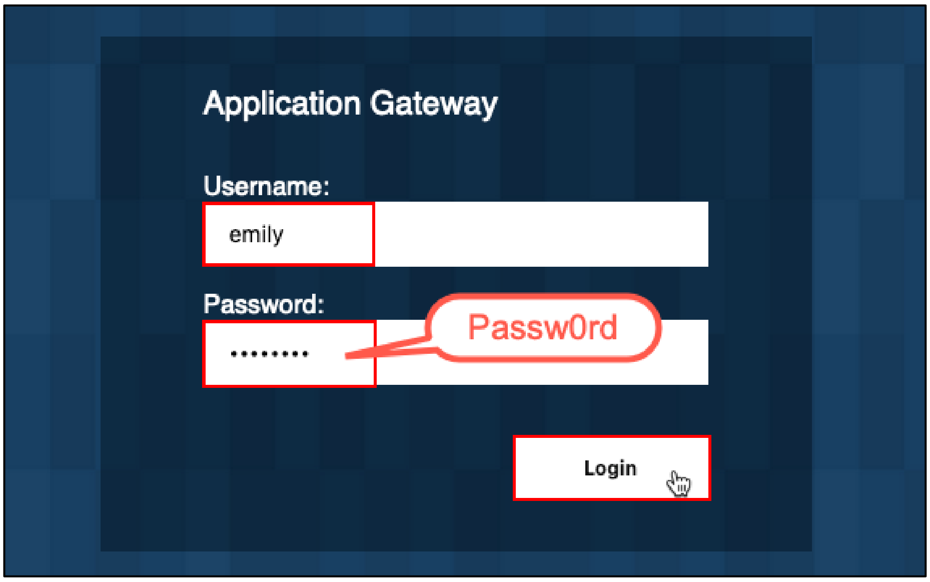
Create and enable two users: Emily and Chuck:

```
pdadmin sec_master> user create emily uid=emily,dc=ibm,dc=com Emily Skillion Passw0rd
pdadmin sec_master> user modify emily account-valid yes
pdadmin sec_master> u c chuck uid=chuck,dc=ibm,dc=com Chuck Kelly Passw0rd
pdadmin sec_master> u m chuck acc yes
```

Exit from the pdadmin tool:

```
pdadmin sec_master> exit
[demouser@centos ~]$
```

In the browser, navigate back to **https://www.iamlab.ibm.com**.



Enter **Emily** as the *Username* and **Passw0rd** as the *Password.* Click **Login**.

The homepage is displayed again.

Nice work. Your Verify Access system on Docker is up and running!

8 Additional Configuration

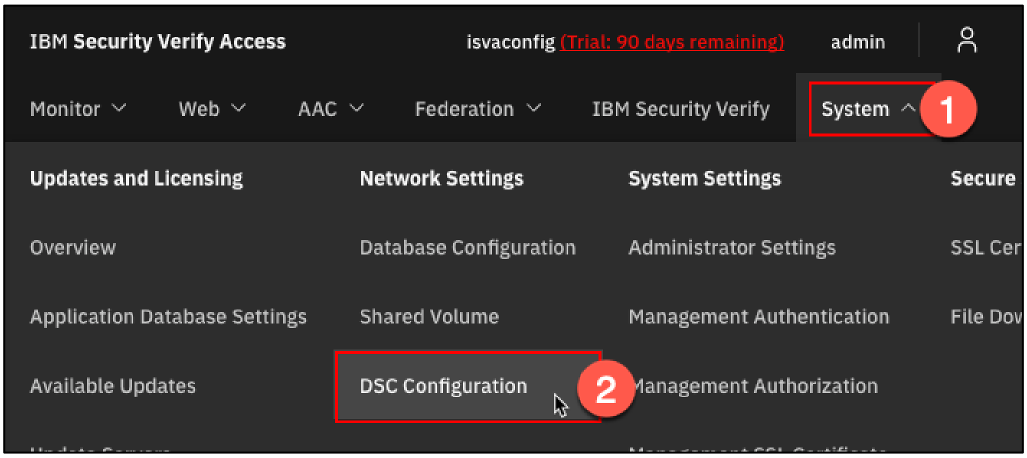
In this section you will enable the Distributed Session Cache in your Verify Access environment and configure the Reverse Proxy for the Advanced Access Control runtime. The environment will then be using all the Verify Access containers.

8.1 Enable Distributed Session Cache

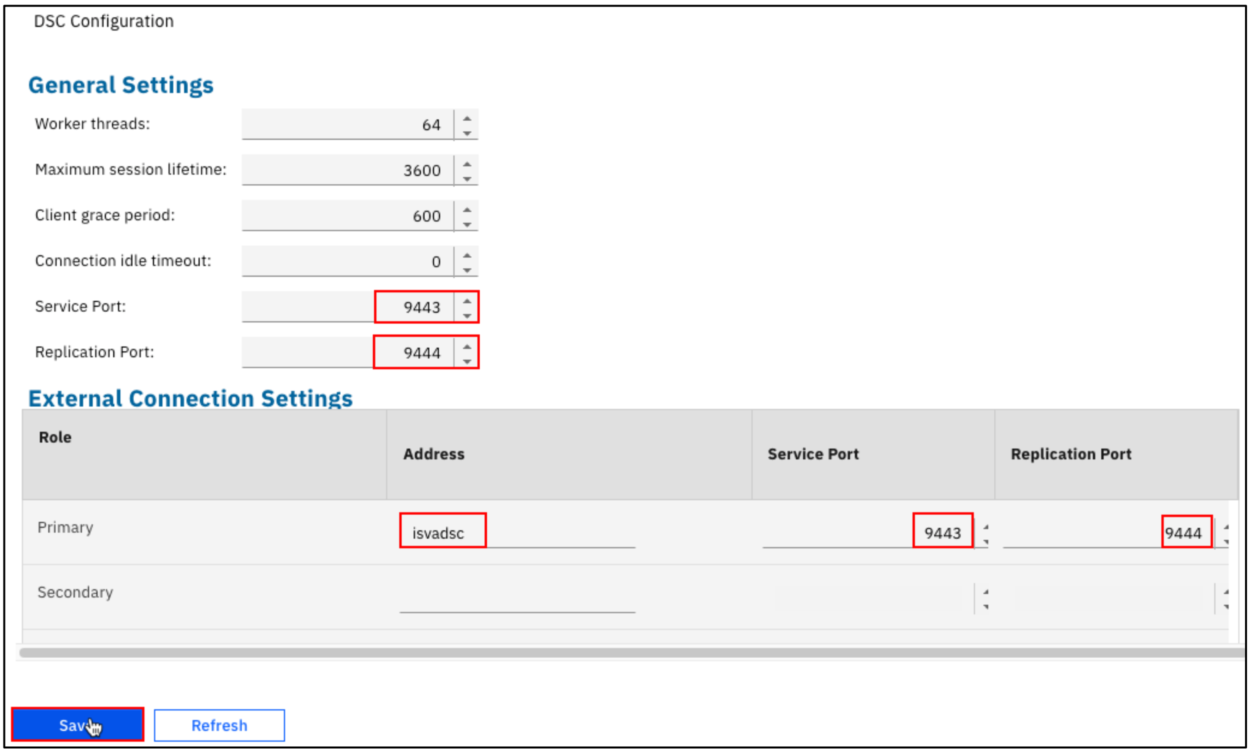
To enable the Distributed Session Cache, it must first be enabled globally, and changes deployed. Once that is done, the Reverse Proxy can be configured to use the DSC.

8.1.1 Enable DSC Globally

Open the Firefox browser in the test machine and navigate to the Config Container LMI interface: **https://lmi.iamlab.ibm.com**. Login with **admin** and **Passw0rd**.



Navigate to **System→Network Settings: DSC Configuration** in the mega-menu.



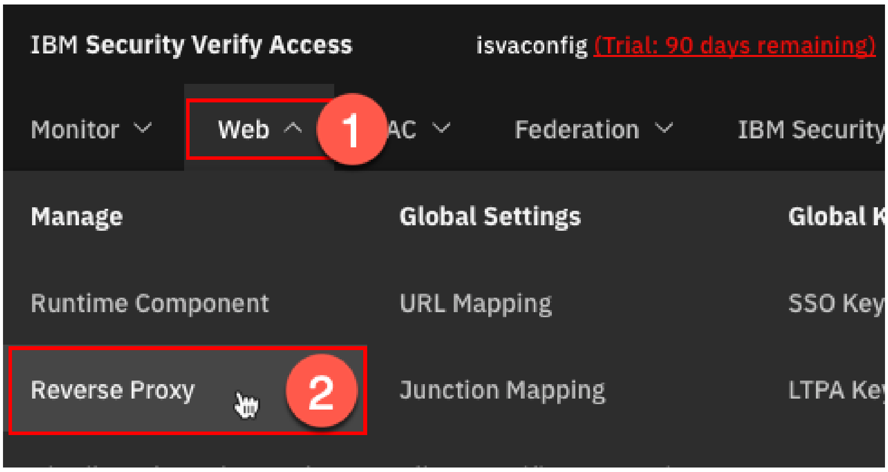
In *General Settings*, set the *Service Port* to **9443** and the *Replication Port* to **9444**. This sets the ports that the DSC will listen on. Using these ports numbers (over 1024) allows the DSC to run without special privileges in secure container environments.

In *General Settings,* in the row for *Primary,* enter **isvadsc** as *Address*, **9443** as *Service Port*, and **9444** as *Replication Port*.

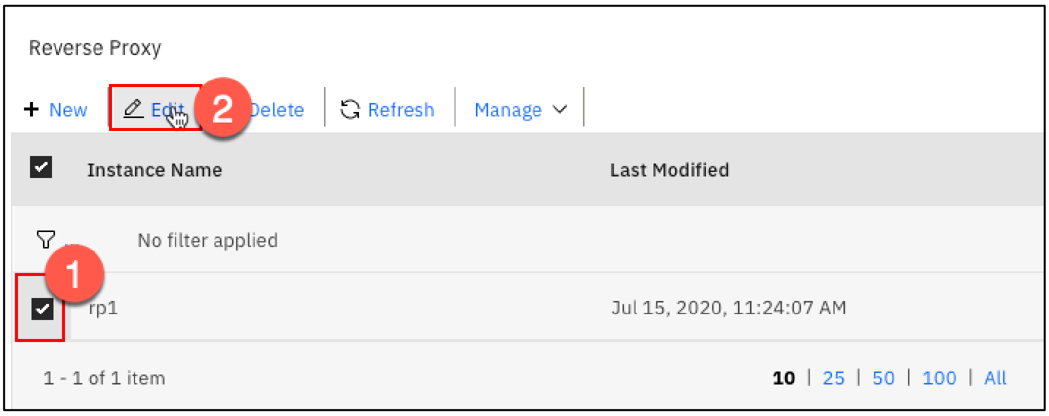
Click **Save**.

Deploy the changes using the link in the yellow warning message

8.1.2 Enable DSC in Reverse Proxy configuration

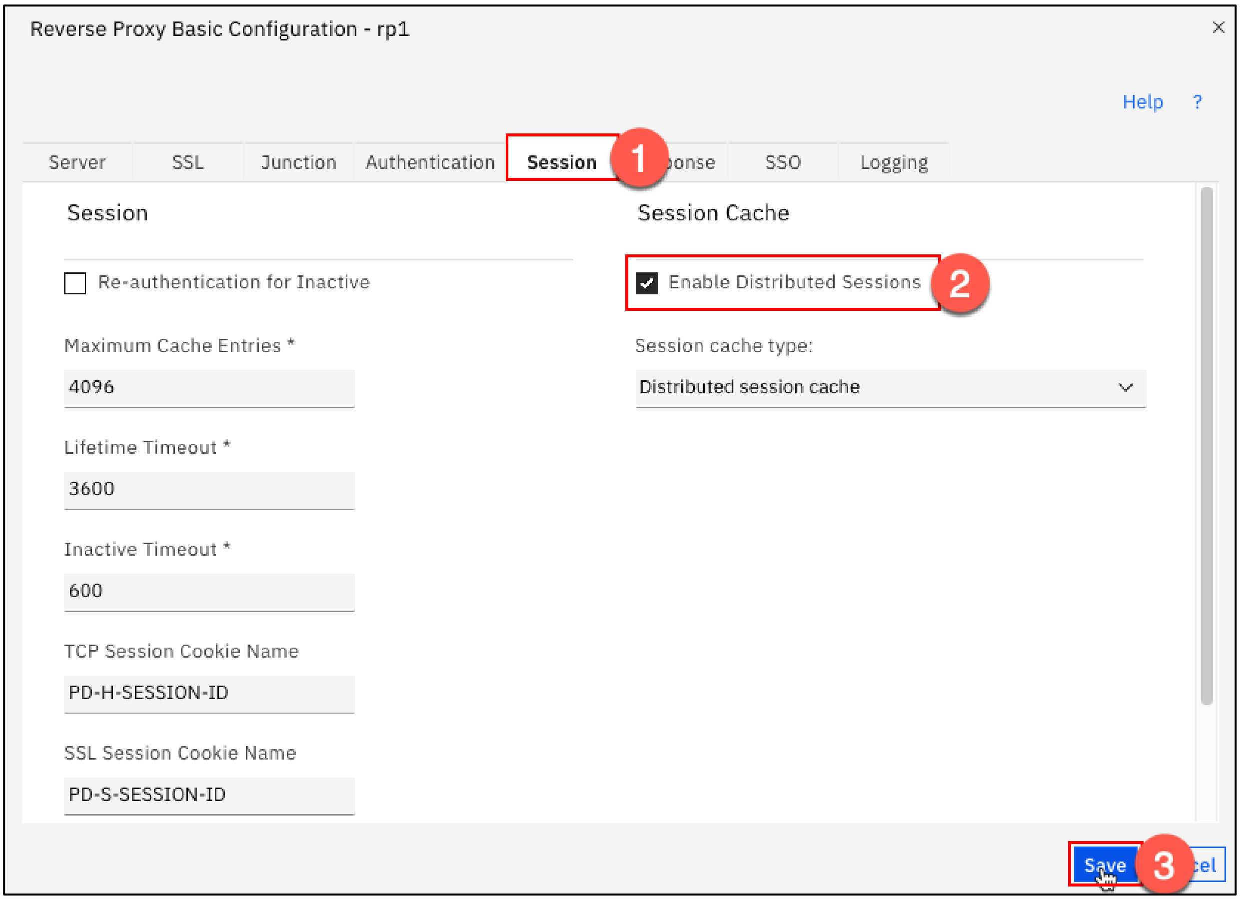


Navigate to **Web→Manage: Reverse Proxy** using the mega-menu.



Select the radio-button for the **rp1** instance and click **Edit**.

An overlay is opened:



Select the **Session** tab. Check the **Enable Distributed Sessions** checkbox.

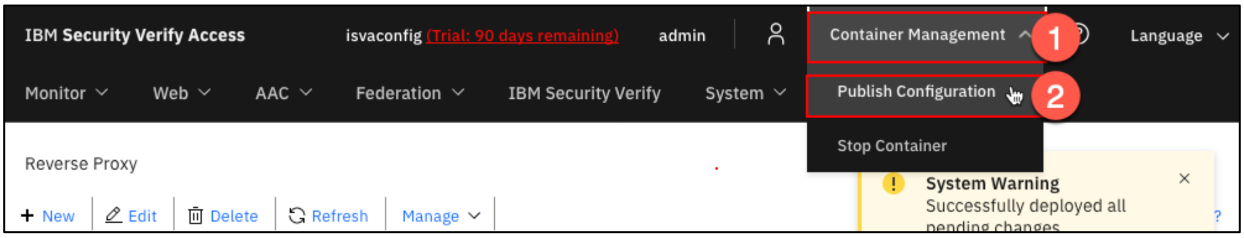
> The other option available for supporting distributed sessions is to use an external Redis cluster. This is beyond the scope of this cookbook but may be a good option in many cases.

Click **Save**.

Deploy the change using the link in the yellow warning message.

8.1.3 Publish snapshot and restart DSC and Reverse Proxy

You must now publish the updated configuration to make it available for the other Verify Access containers.



Click the **Container Management** item in the header bar and then select **Publish Configuration** from the pop-up menu.

Click **Submit** to confirm the publish operation.

At this point a new snapshot is available on the shared configuration volume but none of the worker containers will detect it until they are restarted (or, in some cases, reloaded).

You will now restart the Reverse Proxy and DSC containers to pick up the updated snapshot. Enter the following commands:

```
[demouser@centos ~]$ docker restart isvadsc
isvadsc
[demouser@centos ~]$ docker restart isvawrprp1
isvawrprp1
```

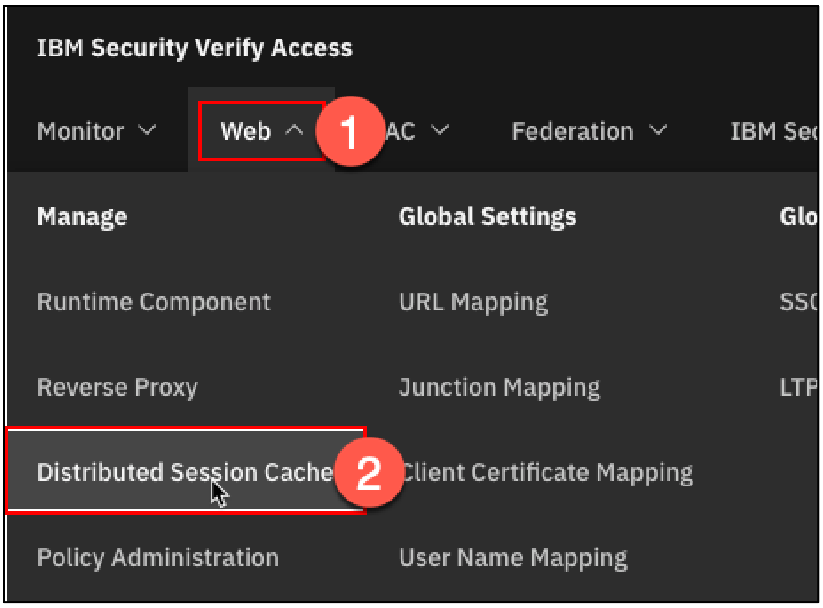
> In previous versions it was possible to use the *AUTO_RELOAD_FREQUENCY* environment variable to have Verify Access containers restart automatically when a new snapshot was detected. This capability is not possible when using the new lightweight containers in 10.0.2.0.

You can check if the DSC has picked up the new snapshot and applied it by looking at the container log:

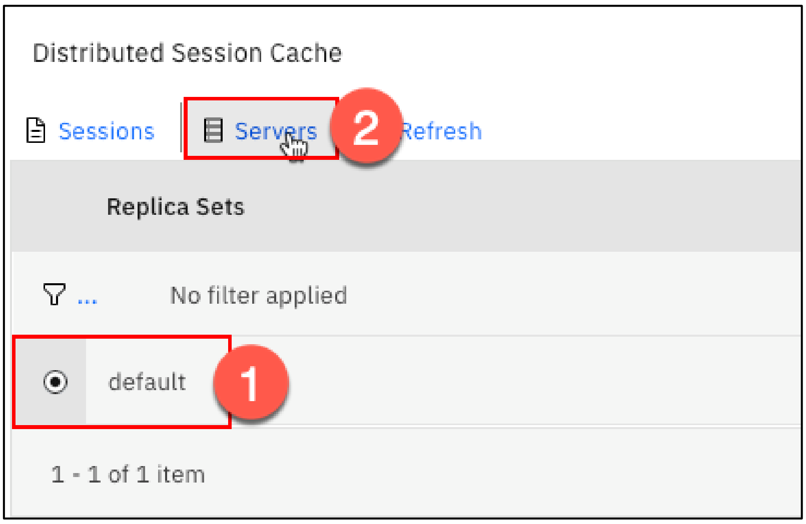
```
[demouser@centos ~]$ docker logs -f isvadsc
…
{"instant":{"epochSecond":1624634536},"threadId":"1","level":"INFO","loggerName":"system",
"component":"bootstrap","source":{"file":"/sbin/bootstrap.sh"}, "content":"WGAWA0971I
Applying the configuration snapshot:
a6fb542fdb344229df4d660c8147be4064f9f6dc936fc7f3be865d731c56aaf5"}
{"instant":{"epochSecond":1624634536},"threadId":"1","level":"INFO","loggerName":"system",
"component":"bootstrap","source":{"file":"/sbin/bootstrap.sh"}, "content":"WGAWA0972I
Starting the DSC Server."}
{"instant":{"epochSecond":1624634536},"threadId":"0x7f19f0e63700","level":"WARNING","logge
rName":"/opt/dsc/bin/dscd","component":"wds.server","message_id":"0x38A0A26A","source":{"f
ile":"AMWSMSServer.cpp","line":1583}, "content":"DPWDS0618W   Entering active mode."}
{"instant":{"epochSecond":1624634536},"threadId":"0x7f19fcdd2740","level":"WARNING","logge
rName":"/opt/dsc/bin/dscd","component":"wds.server","message_id":"0x38A0A25C","source":{"f
ile":"dscd.cpp","line":285}, "content":"DPWDS0604W   The distributed session cache server
has started."}
^C
```

8.1.4 Check Registration

The Reverse Proxy should now register with the Distributed Session Cache on start up. You can check this registration in the LMI.

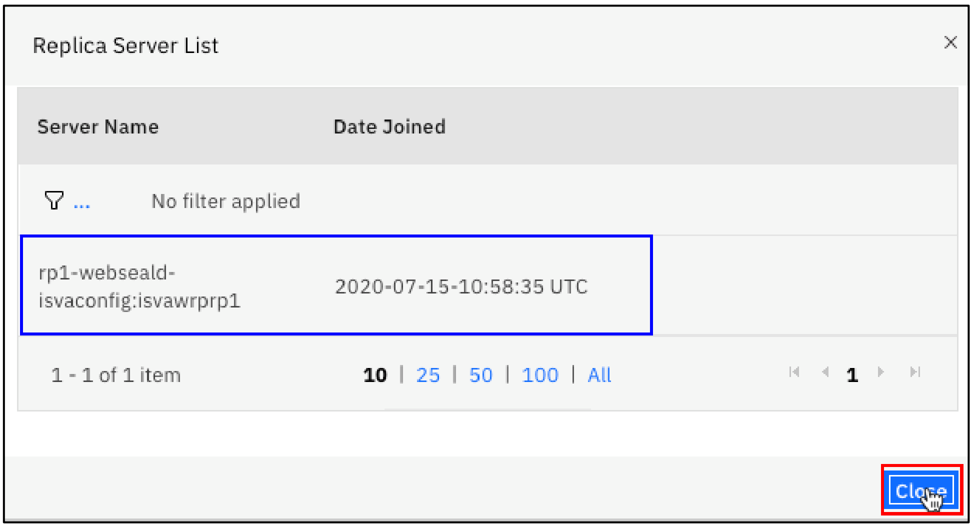


Navigate to **Web→Manage: Distributed Session Cache** in the mega-menu.



If you can see the *default* replica set listed, this means that the Distributed Session Cache is running.

Select the radio button for the **default** replica set and click **Servers** to check that the *rp1* Reverse Proxy is registered.

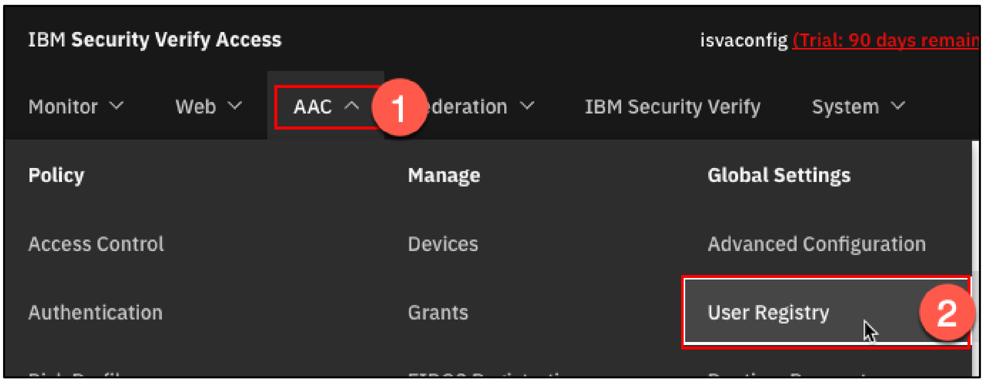


Click **Close**.

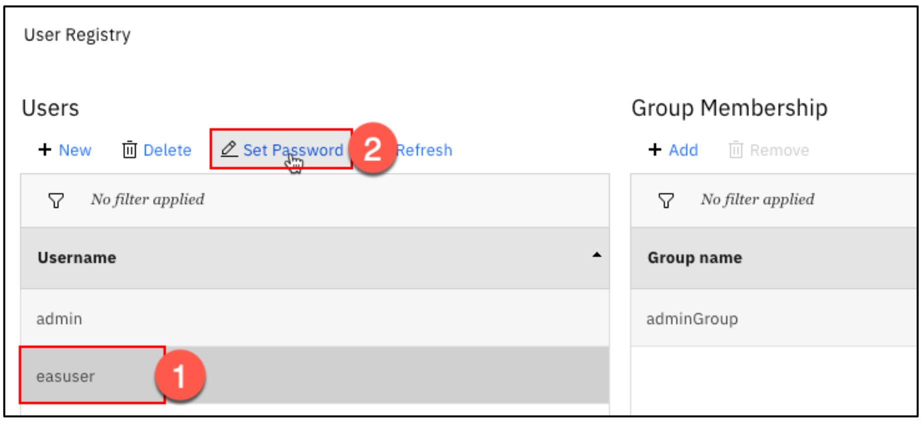
8.2 Configure Reverse Proxy for Advanced Access Control

8.2.1 Set easuser password

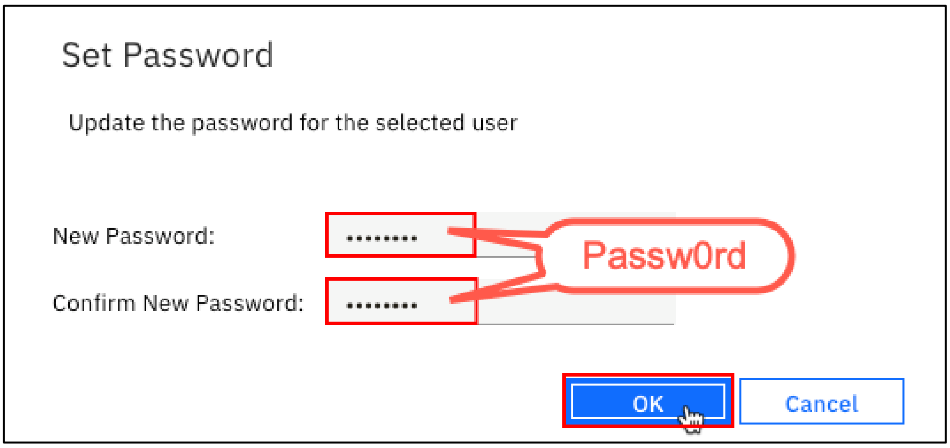
An Verify Access appliance has a default LMI user registered that can be used for authentication to the Advanced Access Control runtime. The userID of this user is *easuser*. You will now set the password for this user.



Navigate to **AAC→Global Settings: User Registry** in the mega-menu.



Select the **`easuser`** entry and then click the **Set Password** button.



Enter **Passw0rd** as the new password and click **OK** to continue.

Deploy the change using the link in the yellow warning message.

8.2.2 Publish Configuration and restart the runtime container

For the AAC configuration wizard to be able to connect to the runtime container, the easuser password change needs to be published.

Click the **Container Management** item in the header bar and then select **Publish Configuration** from the pop-up menu.

Click **Submit** to confirm the publish operation.

Enter the following command to restart the runtime container:

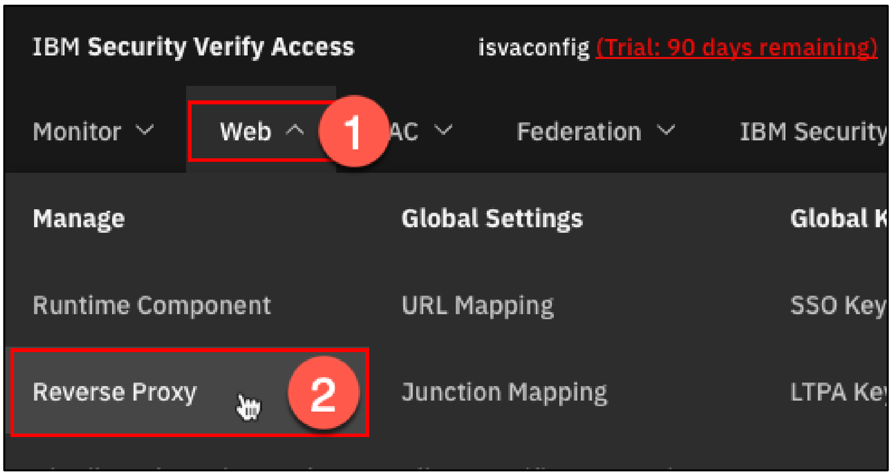
```
[demouser@centos ~]$ docker restart isvaruntime
isvaruntime
```

You can monitor the restart of the runtime container using this command:

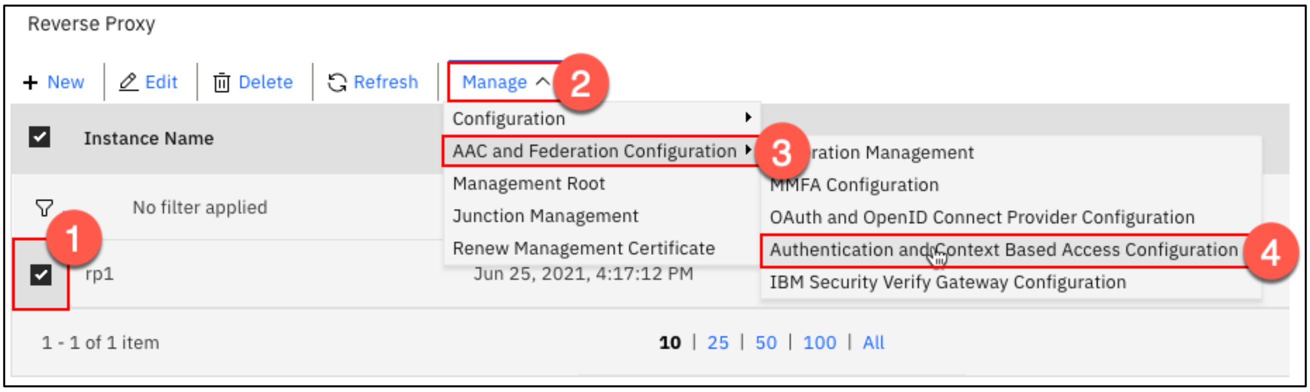
```
demouser@centos ~]$ docker logs -f isvaruntime
…
{"type":"liberty_message","host":"isvaruntime","ibm_userDir":"\/opt\/ibm\/wlp\/usr\/","ibm_serverName":"runtime","message":"CWWKF0011I: The runtime server is ready to run a smarter planet. The runtime server started in 10.608 seconds.","ibm_threadId":"0000002c","ibm_datetime":"2021-06-25T15:32:51.705+0000","ibm_messageId":"CWWKF0011I","module":"com.ibm.ws.kernel.feature.internal.FeatureManager","loglevel":"AUDIT","ibm_sequence":"1624635171705_00000000000FB"}
^C
```

8.2.3 Configure Reverse Proxy for Authentication and Context-based Access

A Reverse Proxy is enabled to use the Authentication and Context-based Access functionality in the AAC Runtime by running a wizard in the LMI.

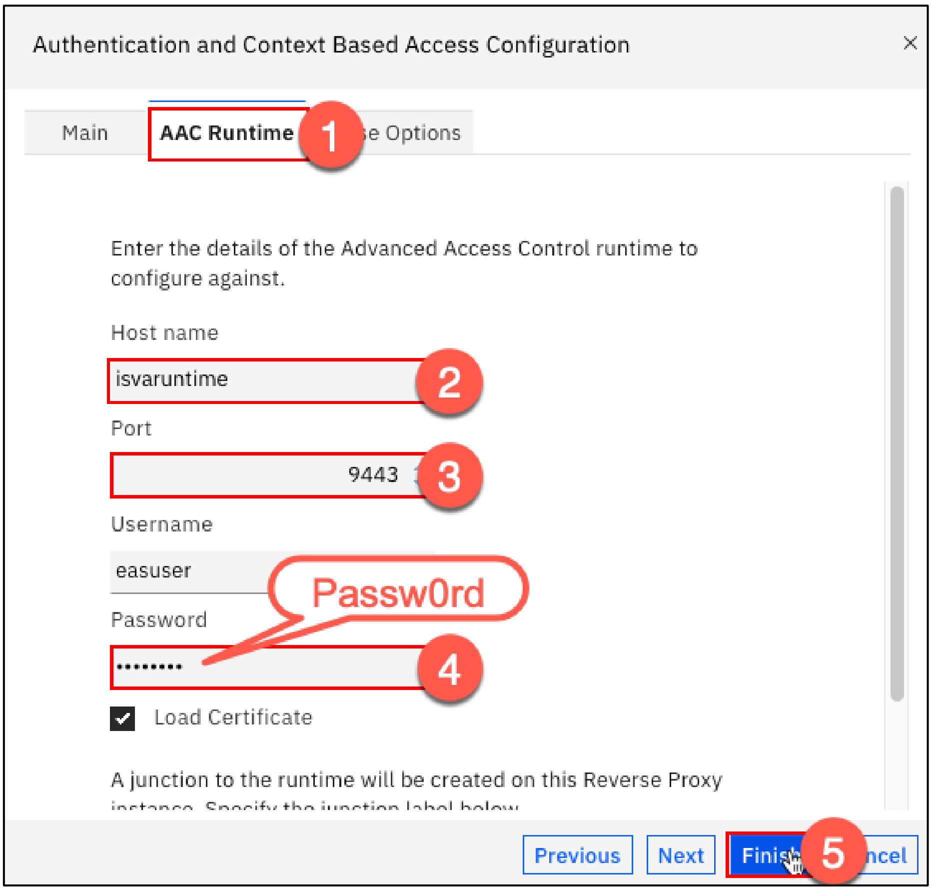


Navigate to **Web→Manage: Reverse Proxy** using the mega-menu.



Select the checkbox for the **rp1** Reverse Proxy instance.

Click the **Manage** button and select **AAC and Federation Configuration→Authentication and Context Base Access Configuration** from the pop-up menus.



Select the **AAC Runtime** tab in the wizard overlay.

Enter **isvaruntime** as the *Hostname* and set the *Port* to **9443**.

> In v10.0.2.0, the runtime listens on port 9443 (rather than 443) so that it can run without privileges in secure container environment. This is a change from previous versions.

Set **Passw0rd** as the *Password* for *easuser*.

Click **Finish**.

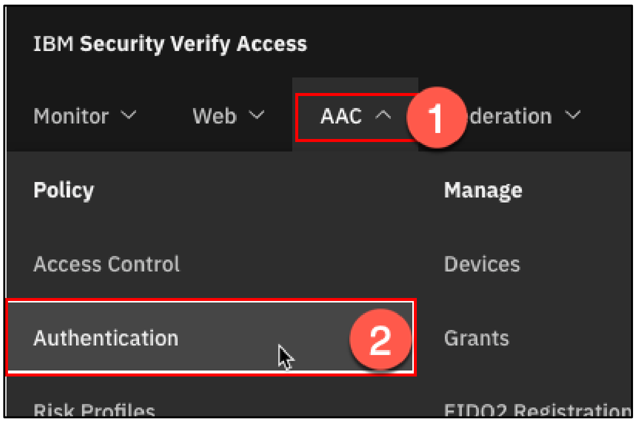
The wizard now runs. It modifies the Reverse Proxy configuration, adds the AAC Runtime server certificate to the Reverse Proxy key store, creates a junction to the AAC Runtime, and creates and attaches required Access Control Lists in the Reverse Proxy object space.

> Note that loading the AAC certificate is now optional. If disabled, this configuration can run even if the AAC is not available. The certificate would need to be loaded manually in this case.

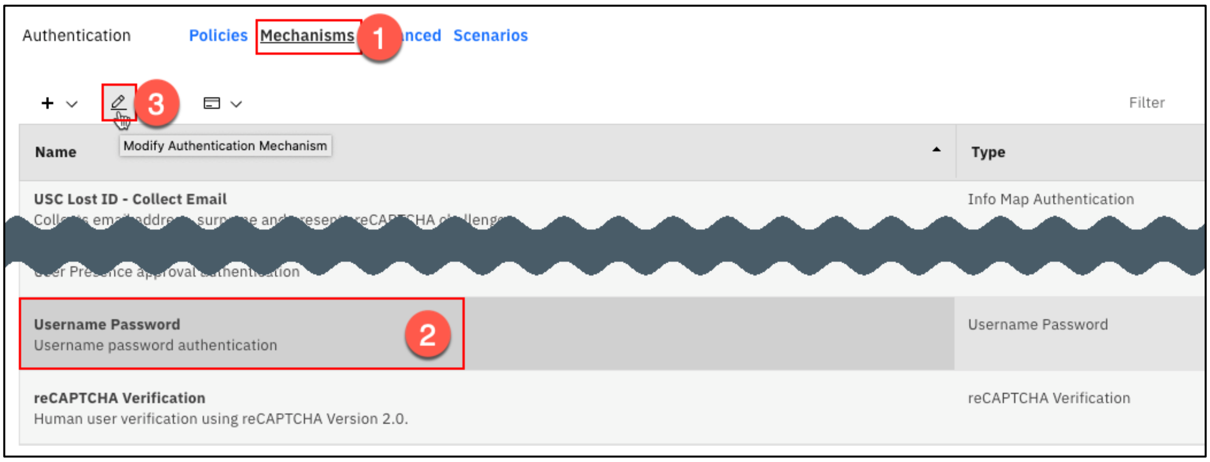
Deploy the changes using the link in the yellow warning message.

8.3 Configure Username Password Mechanism

You will now configure the Username and Password Mechanism within the AAC Advanced Authentication service. This will allow you to test that the AAC Authentication Service is working correctly.

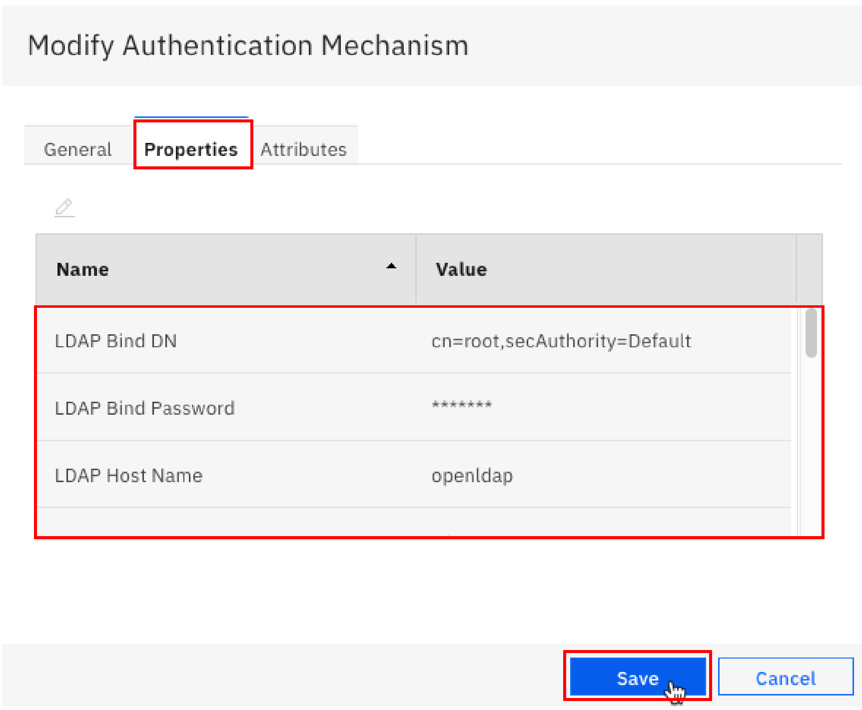


Navigate to **AAC→Policy: Authentication** in the mega-menu.



Select the **Mechanisms** tab.

Locate and select the **Username Password** mechanism and then click the **Modify** icon.



Set the following Properties:

Name	Value
LDAP Bind DN	cn=root,secAuthority=Default
LDAP Bind Password	Passw0rd
LDAP Host Name	openldap
LDAP Port	636
SSL Enabled	True
SSL Trust Store	Registry_Keystore
Use Federated Directories Configuration	True

Click **Save** to save the updated properties.

Deploy the changes using the link in the yellow warning message.

8.4 Publish Configuration and reload containers

Click the **Container Management** item in the header bar and then select **Publish Configuration** from the pop-up menu.

Click **Submit** to confirm the publish operation.

Enter the following commands to restart the Reverse Proxy and Runtime containers:

```
[demouser@centos ~]$ docker restart isvaruntime
isvaruntime
[demouser@centos ~]$ docker restart isvawrprp1
isvawrprp1
```

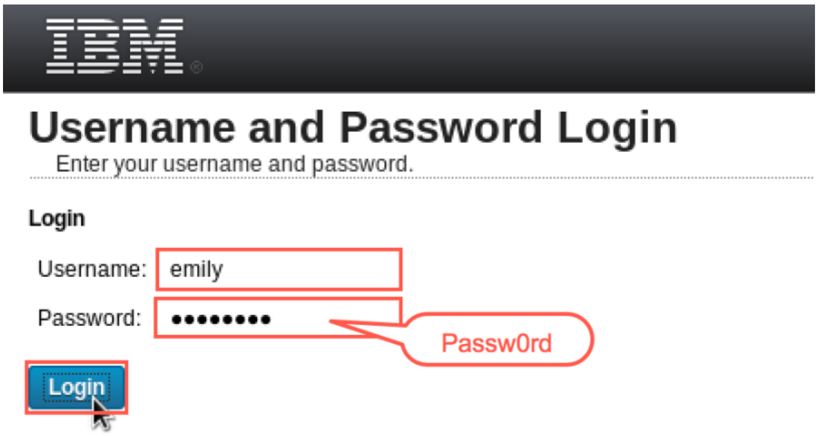
You can monitor the restart of the runtime container using this command:

```
demouser@centos ~]$ docker logs -f isvaruntime
…
{"type":"liberty_message","host":"isvaruntime","ibm_userDir":"\/opt\/ibm\/wlp\/usr\/","ibm
_serverName":"runtime","message":"CWWKF0011I: The runtime server is ready to run a smarter
planet. The runtime server started in 10.608
seconds.","ibm_threadId":"0000002c","ibm_datetime":"2021-06-
25T15:32:51.705+0000","ibm_messageId":"CWWKF0011I","module":"com.ibm.ws.kernel.feature.int
ernal.FeatureManager","loglevel":"AUDIT","ibm_sequence":"1624635171705_00000000000FB"}
^C
```

8.5 Test Updated Configuration

8.5.1 Login using AAC Advanced Authentication Mechanism

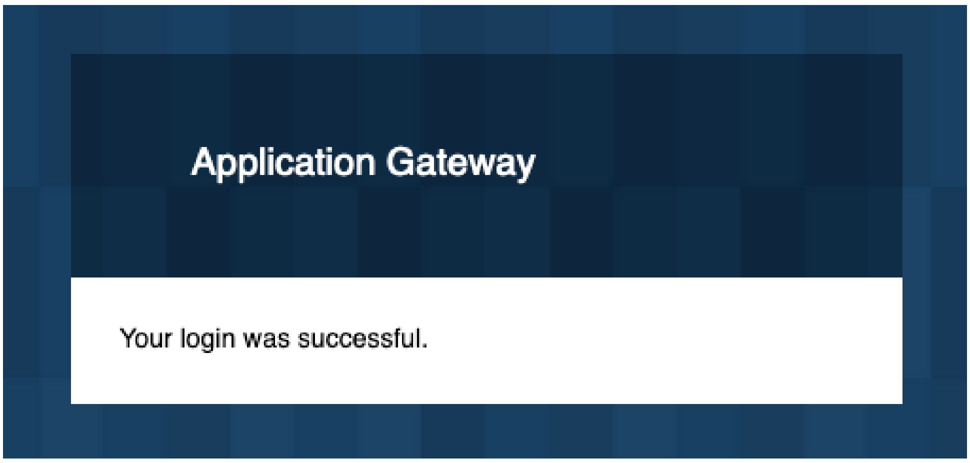
Leaving the LMI open, start a new browser tab and navigate to the following URL:
https://www.iamlab.ibm.com/mga/sps/authsvc?PolicyId=urn:ibm:security:authentication:asf:password_eula



Enter **Emily** as the *Username* and **Passw0rd** as the *Password*. Click **Login**.

Click the radio-button for **I accept the terms of the license agreement** and click **Submit**.

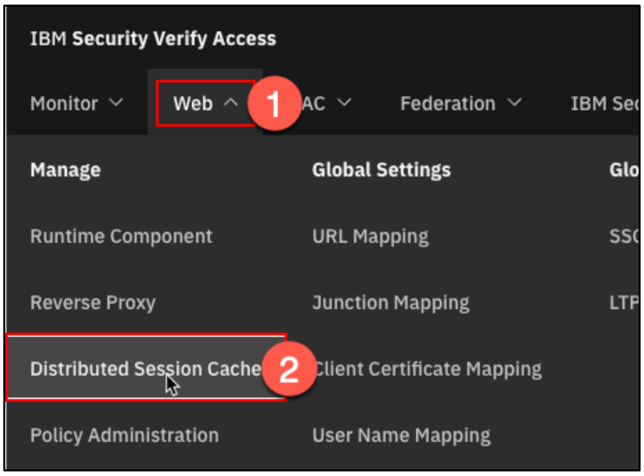
Emily is logged in.



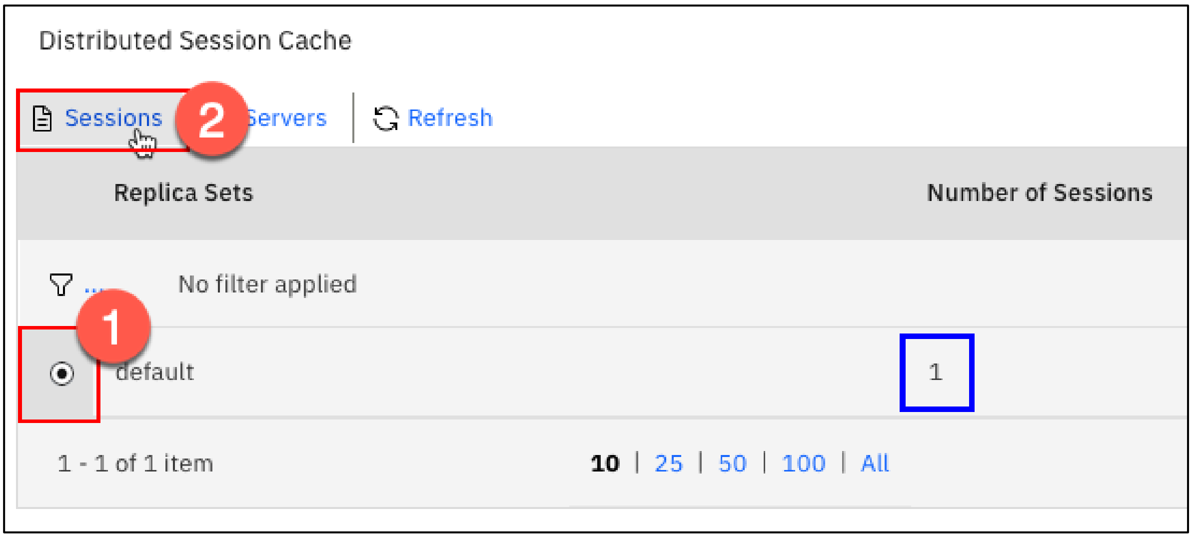
Navigate to the home page: **https://www.iamlab.ibm.com**

8.5.2 Check Sessions in Distributed Session Cache

Go back to the LMI tab.

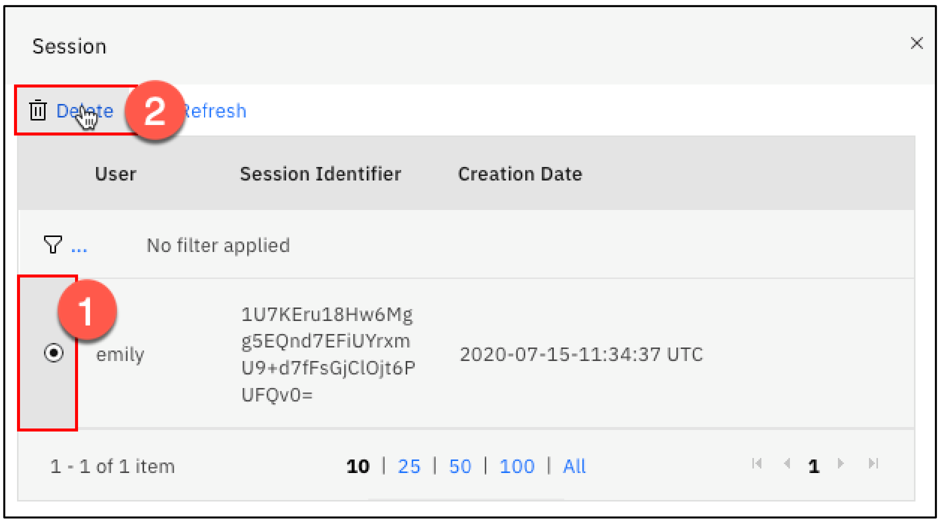


Navigate to **Web→Manage: Distributed Session Cache** in the mega-menu.



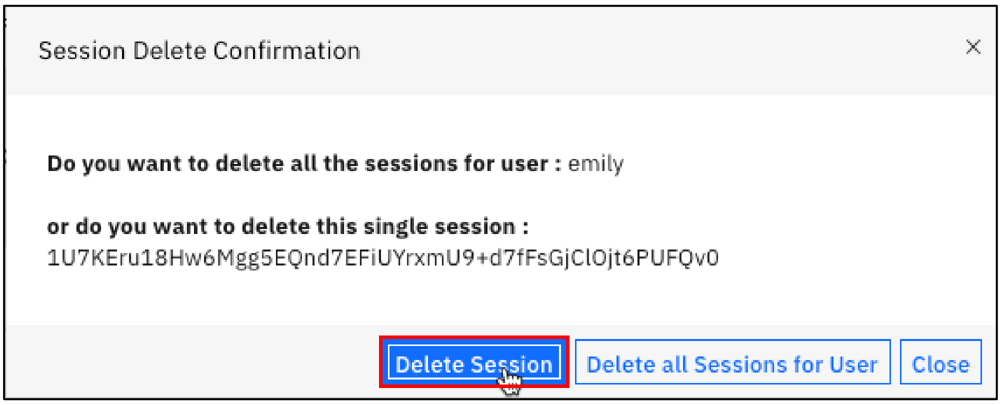
You should see 1 session registered against the *default* Replica Set.

Select the radio button for the **default** Replica Set and click **Sessions**..



You can see the session for Emily that was just created.

The session can be terminated from here. Select the radio button for **Emily** and click **Delete**.



Click **Delete Session** to delete this one session. Note that it's also possible to terminate all sessions for a named user.

Return to the session where Emily was logged in. Refresh the home page: **https://www.iamlab.ibm.com**.

Emily is returned to the login page because her session has been terminated.

<u>You have successfully configured and used the Runtime and DSC containers under Docker</u>

9 Backup configuration and clear native Docker system

When running under Docker, the full state of the environment is held across 4 locations:

- The Key Stores used by OpenLDAP and PostgreSQL
- The configuration snapshot associated with the configuration container
- The user data in the OpenLDAP directory
- The runtime data in the PostgreSQL database

If you back up all of these components, you will have a dataset that can be used to recreate the environment on any Docker system that has the same hostnames used for the various components. This includes environments that use different orchestration methods.

A script is provided which will perform a backup of the current configuration of the Verify Access environment. You will use this script now.

In a terminal window on the test machine, run the following command:

```
[demouser@centos ~]$ git/container-deployment/docker/isva-backup-docker.sh
Done.
```

The script creates an archive file with name isva-*backup-xxxx.tar*:

```
[demouser@centos ~]$ ls
Desktop  dockershare  Documents  Downloads  git  Music  Pictures  Public  isva-backup-
3654.tar  studentfiles  Templates  Videos
[demouser@centos ~]$
```

In this case the archive is called *isva-backup-3654.tar*. Yours will have a different name.

Keep this backup safe. You can use this archive to restore configuration to different container systems (including the docker-compose environment you will create in the next section).

Before continuing to the next section, you need to clear down the current Docker containers and volumes. A script is provided for this.

```
[demouser@centos ~]$ git/container-deployment/docker/cleanup.sh
isvawrprp1
isvadsc
isvaruntime
isvaconfig
openldap
postgresql
isvaconfig
libldap
libsecauthority
ldapslapd
pgdata
isva
Done.
[demouser@centos ~]$
```

10 Docker Compose

10.1 Introduction

Docker Compose (command line *docker-compose*) is an automation tool which can help manage multi-container environments running under Docker. Rather that individually creating and managing Docker resources with native *docker* commands, Docker Compose uses a *Compose File* which defines the containers, volumes, and ports associated with a *project* and the relationships between them. When the *Compose File* is processed, Docker Compose calls the underlying Docker system to create what is needed to build the project. This means an entire application stack can be managed without the need to create scripts for every activity.

In this section you will use a provided *Compose File* which describes the same Verify Access environment that was built using native Docker commands in the previous sections.

10.2 The Compose Project Directory

When running Docker Compose, a *Project Directory* is specified (or the current directory is used). The Project Directory includes a *Compose File* (default *docker-compose.yaml*) which describes the project resources. The name of the directory is also the default *Project Name* which is prepended to created resources.

The *Project Directory* is within the script files (probably under your user's home directory):
git/container-deployment/compose/iamlab

Let's go to that directory:

```
[demouser@centos ~]$ cd git/container-deployment/compose/iamlab/
[demouser@centos iamlab]$
```

10.3 Compose File

Let's have a look at the default Compose File in the Project Directory. Use the following command to open the file in a text editor:

```
[demouser@centos iamlab]$ gedit docker-compose.yaml &
```

The first line of the compose file indicates the file version number. This is important because different versions of Docker Compose support different capabilities. At the time of writing, version 3 is the current version.

```
version: '3'
```

The first section of the compose file defines the volumes needed by the project:

```
volumes:
  isvaconfig:
  libldap:
  ldapslapd:
  libsecauthority:
  pgdata:
```

Notice the format of the file. Indenting is used to determine hierarchy. All of the volume definitions sit inside a *volumes* array. Each volume definition is an (empty) array. Additional configuration for the volume could be included but we don't need it here.

The second section of the compose file defines the services in the project. Each service describes a Docker container. The first service defined is the configuration container:

```
services:

  isvaconfig:
    image: ibmcom/verify-access:${ISVA_VERSION}
    hostname: isvaconfig
    restart: always
    environment:
      - CONTAINER_TIMEZONE=${TIMEZONE}
      - ADMIN_PWD=${ADMIN_PASSWORD}
    volumes:
      - isvaconfig:/var/shared
    ports:
      - ${LMI_IP}:443:9443
    depends_on:
      - openldap
      - postgresql
```

In the *isvaconfig* service definition, you can see all the parameters that you would use to create a Docker container (image, hostname, restart, environment variables, volumes, port mapping).

Note the use of the dash to indicate a list entry in the YAML file.

In addition, Docker Compose defines *depends_on* which is used to determine the order in which containers should be started and stopped. In this case, the configuration container should be started after the *openldap* and *postgresql* containers.

It's also worth noting the use of variables, such as *${LMI_IP}, ${HOME}, ${ISVA_VERSION} and ${ADMIN_PASSWORD},* in the YAML file. These can be environment variables, or they can be defined in an *Environment File.* You'll look at this later.

Take some time to review the rest of the YAML file. When you're done, close the text editor.

> You might notice that there is no network defined in the Compose File. Docker Compose will automatically create a default network for the project (i.e. *iamlab_default*) and attach all containers to it.

10.4 Environment File

The variables used in the Compose File are resolved from either the local environment or from an *Environment File* in the current directory where the *docker-compose* command is run. An environment file must have the name *.env* (and so is hidden in a standard directory listing). The local environment overrides the environment file.

There is an environment file in the *iamlab* project directory. You can see it by listing all files:

```
[demouser@centos iamlab]$ ls -a
.  ..  docker-compose.yaml  .env
[demouser@centos iamlab]$
```

It is only short, so we can easily view the contents from the command line:

```
[demouser@centos iamlab]$ cat .env
TIMEZONE=Europe/London
ADMIN_PASSWORD=Passw0rd
ISVA_VERSION=10.0.2.0
LDAP_VERSION=10.0.2.0
DB_VERSION=10.0.2.0
LMI_IP=127.0.0.2
WEB1_IP=127.0.0.3
WEB2_IP=127.0.0.4
```

> Note that it's not possible to nest environment variables into the Environment File. That's why the file paths that use *${HOME}* are not defined in here.

10.5 Create Environment with Docker Compose

You will now create a Verify Access environment using Docker Compose. You'll then use the backup file that you took from the Native Docker installation to configure it.

10.5.1 Create Key Shares for OpenLDAP and PostgreSQL

To allow you to easily use the same configuration files from the previous Docker installation, you will use the same keys for the OpenLDAP and PostgreSQL containers. A script is provided which will create a new copy of the key files from *git/container-deployment/local/dockerkeys* the *~/dockershare/composekeys* directory.

Run the following script:

```
[demouser@centos iamlab]$ ../create-keyshares.sh
Creating key shares at /home/demouser/dockershare/composekeys
Done.
```

> This script copies the keys from *git/container-deployment/local/dockerkeys*. The keys from our native install should still be available. If not, they can be restored from the archive using the provided *git/container-deployment/common/restore-keys.sh* script.

10.5.2 Run Docker Compose "up" command

When using Docker Compose, a single command brings up the entire environment described in the project file. This command needs to be run in the *iamlab* project directory to pick up the environment variables in the .env file. You're probably already in the right directory but just to be sure:

```
[demouser@centos iamlab]$ cd ~/git/container-deployment/compose/iamlab/
```

Now run the "up" command:

```
[demouser@centos iamlab]$ docker-compose up -d
Creating network "iamlab_default" with the default driver
Creating volume "iamlab_isvaconfig" with default driver
Creating volume "iamlab_libldap" with default driver
Creating volume "iamlab_ldapslapd" with default driver
Creating volume "iamlab_libsecauthority" with default driver
Creating volume "iamlab_pgdata" with default driver
Creating iamlab_openldap_1    ... done
Creating iamlab_isvadsc_1     ... done
```

```
Creating iamlab_postgresql_1  ... done
Creating iamlab_isvawrprp1_1  ... done
Creating iamlab_isvaconfig_1  ... done
Creating iamlab_isvaruntime_1 ... done
```

> The *-d* flag tells the Docker Compose command to detach from the containers it starts. If not used, the whole project runs in the foreground with log message shown on STDOUT and STDERR.

Your Verify Access containers are now running. You can view their status using the *docker-compose ps* command. The *watch* command allows auto refresh of status.

```
[demouser@centos iamlab]$ watch docker-compose ps
```

The Verify Access containers will all initially show as *starting*. After a while, the Config container will show *healthy* but all others will show *unhealthy* (because they are waiting for configuration):

```
Fri Jun 25 17:09:40 2021

        Name                     Command                   State                    Ports
-------------------------------------------------------------------------------------------------------------
iamlab_isvaconfig_1    /sbin/bootstrap.sh               Up (healthy)     443/tcp, 127.0.0.2:443->9443/tcp
iamlab_isvadsc_1       /sbin/bootstrap.sh               Up (unhealthy)   9443/tcp, 9444/tcp
iamlab_isvaruntime_1   /sbin/bootstrap.sh               Up (unhealthy)   9080/tcp, 9443/tcp
iamlab_isvawrprp1_1    /sbin/bootstrap.sh               Up (unhealthy)   9080/tcp, 127.0.0.3:443->9443/tcp
iamlab_openldap_1      /container/tool/run --copy ...   Up               389/tcp, 127.0.0.2:1636->636/tcp
iamlab_postgresql_1    /sbin/bootstrap.sh               Up               5432/tcp
```

Notice that the container names are all prefixed by *iamlab_*. This is the project name and prevents name collisions with other projects that might be running in the same environment.

Press **Ctrl-c** to quit the watch command.

10.5.3 Restore configuration from backup

Right now, the Verify Access environment is not configured, just like it was when you installed using native Docker commands. Rather than run the configuration steps again, you will now restore the configuration that you saved earlier.

This step uses the archive file you created. The name of the file was randomly allocated so you need to identify the name of the archive.

```
[demouser@centos iamlab]$ ls ~
Desktop  dockerkeys  dockershare  Documents  Downloads  Music  Pictures  Public  isva-
backup-3654.tar  studentfiles  Templates  Videos
[demouser@centos iamlab]$
```

In this case the filename is *isva-backup-3654.tar*. Yours will be different.

Now run the restore command for the Docker Compose environment:

```
demouser@centos iamlab]$ ../isva-restore-compose.sh ~/isva-backup-3654.tar
Loading LDAP Data...
Loading DB Data...
Copying Snapshot...
Restarting Config Container...
Restarting iamlab_isvaconfig_1  ... done
Restarting iamlab_isvaruntime_1 ... done
```

```
Restarting iamlab_isvawrprp1_1  ... done
Restarting iamlab_postgresql_1  ... done
Restarting iamlab_isvadsc_1     ... done
Restarting iamlab_openldap_1    ... done
Done.
```

> This command write a log file to */tmp/lab-restore.log*. It contains a lot of "already exists" errors but may also show other issues if the restore is not successful.

If you like, you can use the *watch docker-compose ps* command again to monitor status. This time all the containers will move to *healthy* state as they become available.

You're done! Your Verify Access environment is now running again.

10.6 Test Environment

You will now test the restored environment. First, get the port mappings for the configuration container LMI and the Reverse Proxy HTTPS interface:

> Note that you need to be in the *iamlab* project directory to run this command (to pick up the compose file and the environment file).

```
[demouser@centos iamlab]$ docker-compose port isvaconfig 9443
127.0.0.2:443
[demouser@centos iamlab]$ docker-compose port isvawrprp1 9443
127.0.0.3:443
```

As expected, the LMI is available at *127.0.0.2:443* which, using the hosts file, is also *lmi.iamlab.ibm.com:443*. The Reverse Proxy is available at *127.0.0.3:443* which, using the host file, is also *www.iamlab.ibm.com:443*.

Open the Firefox Browser and navigate to: **https://lmi.iamlab.ibm.com**

Login with **admin** and **Passw0rd**.

Have a look around in the LMI. You'll see that the configuration has been successfully restored from the backup.

Navigate to:
https://www.iamlab.ibm.com/mga/sps/authsvc?PolicyId=urn:ibm:security:authentication:asf:password_eula

Login with **emily** and **Passw0rd**.

Emily is not asked to sign Terms and Conditions. This shows that the runtime database has been successfully restored.

Navigate to: **https://www.iamlab.ibm.com/pkmslogout**

Emily is logged out.

10.7 Clear Docker Compose system

Before moving on, remove the environment created with Docker Compose. This is easily done with a single command:

> Note that you need to be in the *iamlab* project directory to run this command (to pick up the compose file and the environment file.

> The *-v* flag tells Docker Compose to also remove the volumes associated with the project. If this flag is left off, the volumes are retained which would allow the system to be bought up again and retain its persistent data.

```
[demouser@centos iamlab]$ docker-compose down -v
Stopping iamlab_isvaruntime_1 ... done
Stopping iamlab_isvaconfig_1  ... done
Stopping iamlab_isvawrprp1_1  ... done
Stopping iamlab_postgresql_1  ... done
Stopping iamlab_isvadsc_1     ... done
Stopping iamlab_openldap_1    ... done
Removing iamlab_isvaruntime_1 ... done
Removing iamlab_isvaconfig_1  ... done
Removing iamlab_isvawrprp1_1  ... done
Removing iamlab_postgresql_1  ... done
Removing iamlab_isvadsc_1     ... done
Removing iamlab_openldap_1    ... done
Removing network iamlab_default
Removing volume iamlab_isvaconfig
Removing volume iamlab_libldap
Removing volume iamlab_ldapslapd
Removing volume iamlab_libsecauthority
Removing volume iamlab_pgdata
```

Your system is clean. The cookbook is complete.

11 Notices

Statement of Good Security Practices

IT system security involves protecting systems and information through prevention, detection and response to improper access from within and outside your enterprise. Improper access can result in information being altered, destroyed, misappropriated or misused or can result in damage to or misuse of your systems, including for use in attacks on others. No IT system or product should be considered completely secure and no single product, service or security measure can be completely effective in preventing improper use or access. IBM systems, products and services are designed to be part of a comprehensive security approach, which will necessarily involve additional operational procedures, and may require other systems, products or services to be most effective. IBM DOES NOT WARRANT THAT ANY SYSTEMS, PRODUCTS OR SERVICES ARE IMMUNE FROM, OR WILL MAKE YOUR ENTERPRISE IMMUNE FROM, THE MALICIOUS OR ILLEGAL CONDUCT OF ANY PARTY.