

# Container Training

prepared for

## Attendees

Draft

Disclaimer SUSE makes no representations or warranties with respect to the contents or use of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose.

Trademarks SUSE and the SUSE logo are registered trademarks of SUSE LLC in the United States and other countries. All third-party trademarks are the property of their respective owner.

Copyright 2017 SUSE LLC. All rights reserved. No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express written consent of SUSE.

SUSE

SUSE

1800 South Novell Place

Maxfeldstr. 5

Provo

90409 Nürnberg

UT 84606

Germany

USA

Prepared By Martin Weiss

Container Training—Exercises

June, 2016

Statement of Work: SOW #

Attendees Number: Client #

Consultants:

Draft

# Revision History

| Version   | Date                        | Editor       | Revisions    |
|-----------|-----------------------------|--------------|--------------|
| 0.1 / 0.6 | 29.06.2016 to<br>09.08.2016 | Martin Weiss | First Draft  |
| 0.7       | 09.08.2016                  | Martin Weiss | Second Draft |
| 0.8       | 26.03.2020                  | Martin Weiss | Third Draft  |
| 0.9       | 07.04.2020                  | Martin Weiss | Fourth Draft |

# Reviewer Record

| Version | Date | Reviewers Name | Reviewers Role<br>(Peer/ Team Leader / Project Manager) |
|---------|------|----------------|---|
|         |      |                |   |
|         |      |                |   |
|         |      |                |   |
|         |      |                |   |

# Final Approval

| Version | Date | Reviewers Name | Reviewers Role<br>(Peer/ Team Leader / Project Manager) |
|---------|------|----------------|---|
|         |      |                |   |

# Contents

|   |           |
|---|-----------|
| <b>1 Introduction.....</b>  | <b>1</b>  |
| <b>2 Environment Preparation.....</b>   | <b>2</b>  |
| 2.1 Container Server.....   | 2         |
| 2.2 Container Server Preparation and Verification.....  | 2         |
| 2.3 Prepare NFS export.....   | 2         |
| 2.4 Create Non-root User.....   | 3         |
| 2.5 Other Users.....  | 3         |
| 2.6 Connect Containers Channel.....   | 3         |
| 2.6.1 Connect channels with RMT/SMT/SCC connection.....   | 3         |
| 2.6.2 Connect channels manually.....  | 4         |
| 2.7 Push Images to Registry.....  | 4         |
| <b>3 Exercise: Installation and Configuration of Docker.....</b>                                | <b>6</b>  |
| <b>4 Exercise: Installation and Configuration of Podman.....</b>                                | <b>7</b>  |
| <b>5 Exercise: Implementation of the Docker Distribution Registry and Portus (SLES 12).....</b> | <b>8</b>  |
| 5.1 Installation of Docker Registry.....  | 8         |
| 5.2 Installation Of Portus.....   | 8         |
| 5.3 Portus Database Configuration.....  | 8         |
| 5.4 Configuring Portus.....   | 9         |
| 5.5 Creating the first Administrator User.....  | 10        |
| 5.6 Organization of a Registry.....   | 11        |
| 5.7 Playing with Portus (1).....  | 11        |
| <b>6 Exercise: Implementation of the Docker Distribution Registry and Portus (SLES 15).....</b> | <b>13</b> |
| 6.1 SSL Certificate Creation.....   | 13        |
| 6.2 Load Registry Images into local image repository.....                                       | 14        |
| 6.3 Installation of Docker Distribution Registry.....   | 15        |
| 6.4 Prepare Portus MySQL Database.....  | 15        |
| 6.5 Installation of Portus.....   | 16        |
| 6.6 Configuring Portus.....   | 16        |
| 6.7 Configure the Registry in Portus.....   | 17        |
| 6.8 Creating the first Administrator User.....  | 18        |
| 6.9 Organization of a Registry.....   | 19        |
| 6.10 Playing with Portus (1).....   | 19        |
| <b>7 Exercise: Pull an Image from a Registry.....</b>   | <b>21</b> |
| 7.1 Pull an Image from a Registry.....  | 21        |
| 7.2 Adding the CA Certificate as Trusted Certificate.....                                       | 21        |
| 7.3 Playing with Portus (3).....  | 21        |
| <b>8 Exercise: Get Images and push them to the Registry.....</b>                                | <b>22</b> |

|   |           |
|---|-----------|
| 8.1 Obtaining The Official SLE Images For Docker (SLES 12)..... | 22        |
| 8.2 Activating Docker Images (SLES 12).....                     | 22        |
| 8.3 Naming / Tagging Images.....                                | 23        |
| 8.4 Authenticating to the Registry.....                         | 23        |
| 8.5 Pushing Images To The Registry.....                         | 24        |
| 8.6 Playing with Portus (2).....                                | 24        |
| <b>9 Exercise: Using Images and Containers.....</b>             | <b>25</b> |
| 9.1 List images.....  | 25        |
| 9.2 Tag Images.....   | 25        |
| 9.3 Pull images.....  | 25        |
| 9.4 Push images.....  | 26        |
| 9.5 Get the history of an image.....                            | 26        |
| 9.6 Run an interactive container.....                           | 26        |
| 9.7 Attach to and detach from running containers.....           | 27        |
| 9.8 List running containers.....                                | 27        |
| 9.9 Stop containers.....  | 27        |
| 9.10 Inspect changes made to a running container.....           | 27        |
| 9.11 Save changes made to a container.....                      | 28        |
| 9.12 Configure a container for autostart.....                   | 28        |
| 9.13 Network in a container.....                                | 28        |
| 9.13.1 Network addresses translation.....                       | 28        |
| 9.13.2 Separate Network and static IP.....                      | 29        |
| <b>10 Exercise: Build Your Own Image.....</b>                   | <b>30</b> |
| 10.1 How official SLE images handle repositories.....           | 30        |
| 10.2 Build new Docker images – the proper way.....              | 31        |
| 10.3 Use buildah to build an image.....                         | 31        |
| <b>11 Exercise: Manage Data Inside Of Containers.....</b>       | <b>33</b> |
| 11.1 System volumes.....  | 33        |
| 11.2 Data volumes.....  | 33        |
| <b>12 Exercise: Patching A Docker Image / Container.....</b>    | <b>35</b> |
| 12.1 Easy patch and update.....                                 | 35        |
| 12.1.1 Installation.....  | 35        |
| 12.1.2 How zypper-docker works.....                             | 35        |
| 12.1.3 Inspecting a Docker image.....                           | 35        |
| 12.1.3.1 Looking for updates.....                               | 35        |
| 12.1.3.2 Looking for patches.....                               | 35        |
| 12.1.3.3 Looking for specific issues.....                       | 36        |
| 12.1.3.4 Checking image status from scripts.....                | 36        |
| 12.1.3.5 Inspecting a running container.....                    | 36        |
| 12.2 Applying patches and updates.....                          | 36        |
| 12.2.1 Image Versioning.....                                    | 36        |
| 12.2.2 Patching containers.....                                 | 37        |

|  |           |
|--|-----------|
| 12.2.3 Applying all the updates.....                           | 37        |
| 12.2.4 Applying patches.....                                   | 37        |
| 12.2.5 Patching specific issues.....                           | 37        |
| <b>13 Exercise: Attach second container for debugging.....</b> | <b>39</b> |
| <b>14 Testcases for Self-Build Images.....</b>                 | <b>40</b> |
| <b>15 Troubleshooting.....</b>                                 | <b>41</b> |
| 15.1 Docker logs.....  | 41        |
| 15.2 Docker Daemon.....  | 41        |
| 15.3 Docker Registry and Portus.....                           | 41        |
| 15.4 Managing Services.....                                    | 41        |
| <b>16 End of Document.....</b>                                 | <b>42</b> |

## Tables

## Figures

# 1 Introduction

These exercises were created to get some hands on with Containers / Docker and Portus.

- All is done on SLES 15 SP1 with latest patches
  - This server is installed via / against an RMT or SMT server providing the latest channels
- Docker will be deployed
- docker-distribution-registry will be deployed via images
- Portus with mariadb will be deployed via images
- Basic Docker commands will be used to
  - work with images
  - run a container
  - inspect a container
  - operate a container
  - volumes
  - port mappings
  - config data
- A new image will be build with docker

Hint: Podman is included in this workshop but will be supported officially starting with SLES 15 SP1

**ToDo: Add Podman / Buildah**

**- do all with podman instead of docker**

**- build images with buildah instead of docker**

**- discuss and check btrfs details**

**ToDo: collect all images and scripts required to be transported via USB stick**



## 2 Environment Preparation

### 2.1 Container Server

All exercises are done on one single server.

My servers IP address: \_\_\_\_\_ (i.e. 10.1.1.104)

My servers DNS name: \_\_\_\_\_ (i.e. sles15spl-1.suse.)

My root user password: \_\_\_\_\_ (i.e. suse1234)

### 2.2 Container Server Preparation and Verification

Login to the server via SSH / Putty as user root.

```
ssh root@<my servers DNS name>
```

Verify time synchronization with

```
chronyc sources
```

Example result:

```
sles15spl-1:~ # chronyc sources
210 Number of sources = 2
MS Name/IP address          Stratum Poll Reach LastRx Last sample
=====
^* smt.suse                  4      6   377    65   +85us[ +112us] +/-  16ms
```

Verify name resolution with

```
nslookup $(hostname -f)
nslookup $(hostname -i)
```

Example result:

```
sles15spl-1:~ # nslookup $(hostname -f)
Server:         10.1.1.1
Address:        10.1.1.1#53

Name:   sles15spl-1.suse
Address: 10.1.1.104

sles15spl-1:~ # nslookup $(hostname -i)
104.1.1.10.in-addr.arpa name = sles15spl-1.suse.
```

### 2.3 Prepare NFS export

Install NFS server

```
zypper -n in nfs-kernel-server yast2-nfs-server
```

Create export directory

```
mkir -p ! ata!share
```



Exported /data/shared for world in RW mode

```
echo "data!share    #(r$%no&root&s'uash%sync%no&su(tree&check)" > /etc/exports
```

Enable and start the nfs server

```
systemctl enable --no$ nfs-server
```

Verify if the export is setup properly

```
exportfs
```

Expected result:

```
sles15spl-1:~ # exportfs
/data/shared    <world>
```

## 2.4 Create Non-root User

Via SSH as user root on the server create a new non-root user "joedoe"

User: joedoe

Password: suse1234

```
useradd -s /bin/bash joedoe
passwd joedoe
```

Add the user to sudoers

```
echo "joedoe    +,,-(root) N.+/+SS0D1+,," > /etc/sudoers.d/joedoe
```

## 2.5 Other Users

(will be created during the following exercises)

Portus user: administrator / suse1234

MySQL/MariaDB: root / suse1234

## 2.6 Connect Containers Channel

Add "sudo" in case the steps are not executed as the user "root".

### 2.6.1 Connect channels with RMT/SMT/SCC connection

Beside the SLES channels the server must have the containers channel connected

```
su o S3$45onnect --pro uct sle-mo ule-containers!6726!)89&9:
```

For later Kubernetes / CaaSP exercises the CaaSP channels also need to be connected

With SCC connection:

```
su o S3$45onnect --pro uct caasp!:2;!)89&9: -r +DD<=<.N+, >4?5.D4
```

With RMT Server no reg-code is required:

```
su o S3$45onnect --pro uct caasp!:2;!)89&9:
```

## 2.6.2 Connect channels manually

In case the server is not registered to the SMT/RMT/SCC.

Hint: replace the IP address with the IP address or DNS name of your SMT/RMT server.

Containers Module:

```
su o zypper ar http://6.262626!S3S4!/ro ucts!S,4-@o ule-5ontainers!67-S/6!)89&9:!
pro uct! S,4-@o ule-5ontainers67-S/6-/ool
su o zypper ar
http://6.262626!S3S4!3p ates!S,4-@o ule-5ontainers!67-S/6!)89&9:!up ate! S,4-
@o ule-5ontainers67-S/6-3p ates
```

CaaSP Add-on:

```
su o zypper ar http://6.262626!repo!S3S4!/ro ucts!S3S4-5++S/!2;!89&9:!pro uct!
S3S4-5++S/-:2;-/ool
su o zypper ar http://smt2suse!repo!S3S4!3p ates!S3S4-5++S/!2;!89&9:!up ate!
S3S4-5++S/-:2;-3p ates
```

## 2.7 Push Images to Registry

On SLES 12 get the image for SLES 11 SP4 from the containers channel RPM and save to file

```
zypper in sles66sp:- ocker-ima s sle2 ocker
sle2 ocker activate --all
ocker save suse!sles66sp:- ocker162626 -o suse-sles66sp:- ocker-626262tar
```

Get the images from the SUSE registry

```
po man pull reAistry2suse2com!suse!sle67167262922B9
po man pull reAistry2suse2com!suse!sles62spC12:2BD
po man pull reAistry2suse2com!suse!sles62sp:1292B8
po man pull reAistry2suse2com!suse!sles62sp717222C6;
```

Save the images to files

```
po man save reAistry2suse2com!suse!sle67167262922B9 -o reAistry2suse2com-suse-
sle67-672629222B92tar
po man save reAistry2suse2com!suse!sles62spC12:2BD -o reAistry2suse2com-suse-
sles62spC-2:2BD2tar
po man save reAistry2suse2com!suse!sles62sp:1292B8 -o reAistry2suse2com-suse-
sles62sp:-292B82tar
po man save reAistry2suse2com!suse!sles62sp717222C6; -o reAistry2suse2com-suse-
sles62sp7-7222C6;2tar
```

Load the images from files

```
po man loa suse!sles66sp:- ocker162626 -i ! ata!sles!suse-sles66sp:- ocker-
626262tar
po man loa reAistry2suse2com!suse!sle67167262922B9 -i
! ata!sles!reAistry2suse2com-suse-sle67-672629222B92tar
po man loa reAistry2suse2com!suse!sles62spC12:2BD -i ! ata!sles!reAistry2suse2com-
suse-sles62spC-2:2BD2tar
po man loa reAistry2suse2com!suse!sles62sp:1292B8 -i ! ata!sles!reAistry2suse2com-
suse-sles62sp:-292B82tar
po man loa reAistry2suse2com!suse!sles62sp717222C6; -i
! ata!sles!reAistry2suse2com-suse-sles62sp7-7222C6;2tar
```

Create team “base\_images” and for this team create a namespace “base\_images”.

Copy images using skopeo

```
skopeo copy ocker1!!reAistry2suse2com:sles62!portus122:2C ocker1!!sles67sp6-62suse17;;;!(ase&imaAes!portus122:2C -- est-cre s a min1suse62C:
skopeo copy ocker1!!reAistry2suse2com:suse:sles62sp717222C6; ocker1!!sles67sp6-62suse17;;;!(ase&imaAes!sles62sp717222C6; -- est-cre s a min1suse62C:
skopeo copy ocker1!!reAistry2suse2com:sles62!reAistry122922 ocker1!!sles67sp6-62suse17;;;!(ase&imaAes!reAistry122922 -- est-cre s a min1suse62C:
skopeo copy ocker1!!reAistry2suse2com:suse:sles62spC12:2BD ocker1!!sles67sp6-62suse17;;;!(ase&imaAes!sles62spC12:2BD -- est-cre s a min1suse62C:
skopeo copy ocker1!!reAistry2suse2com:suse:sles62sp:1292B8 ocker1!!sles67sp6-62suse17;;;!(ase&imaAes!sles62sp:1292B8 -- est-cre s a min1suse62C:
skopeo copy ocker1!!reAistry2suse2com:suse:sle671672629222B9 ocker1!!sles67sp6-62suse17;;;!(ase&imaAes!sle671672629222B9 -- est-cre s a min1suse62C:
skopeo copy ocker1!!reAistry2suse2com:sles62!maria (16;2; ocker1!!sles67sp6-62suse17;;;!(ase&imaAes!maria (16;2; -- est-cre s a min1suse62C:
skopeo copy containers-storage1localhost:suse:sles66sp:- ocker162626
ocker1!!sles67sp6-62suse17;;;!(ase&imaAes!sles66sp:- ocker162626 -- est-cre s a min1suse62C:
```

## 3 Exercise: Installation and Configuration of Docker

In case you are on VMware - a snapshot: "start" can be created and used, here.

Goal of this exercise is to setup the docker back-end infrastructure

Hint: add "sudo" in case you use the user "joedoe"

Install, activate and start docker:

```
su o zypper in docker
su o systemctl enable docker
su o systemctl start docker
su o systemctl status docker
```

By default the Docker daemon listens for connections using a local socket. Only member of the "docker" group can access this socket. Hence the "joedoe" user needs to be part of this group:

```
su o !usr:s(!n!usermo -a? docker *oe oe
```

Hint: in order to apply this change the "joedoe" user must logout and then login.

Optional: Power off the server and create snapshot "after docker installation"

## 4 Exercise: Installation and Configuration of Podman

In case you are on VMware - a snapshot: "start" can be created and used, here.

Goal of this exercise is to setup the docker back-end infrastructure

Hint: add "sudo" in case you use the user "joedoe"

Install, activate and start docker:

```
su o zypper in po man
```

Ensure IP forwarding is enabled

```
se -i "!net2ip:2ip&for$ar !" !etc!sysctl2conf  
echo "net2ip:2ip&for$ar - 6" >> !etc!sysctl2conf  
sysctl --system
```

ToDo: give non-root user rights

ToDo: any service needs to be started?

ToDo: podman --restart unless-stopped

Optional: Power off the server and create snapshot "after docker installation"

## 5 Exercise: Implementation of the Docker Distribution Registry and Portus (SLES 12)

Snapshot “after docker installation”

Goal: Installation and configuration of the docker distribution registry with portus

Portus is the authorization service and front-end for the docker distribution registry. It is not an alternative registry, it's a component to be used in addition to the docker distribution registry to secure your docker images by adding authentication and authorization.

### 5.1 Installation of Docker Registry

Type the following commands to install the Docker registry:

```
zypper in docker-distribution-registry
```

### 5.2 Installation Of Portus

Portus is going to be part of the “containers” module. To install it just execute:

```
zypper in portus
```

This is going to install all the packages required to run Portus including MariaDB.

### 5.3 Portus Database Configuration

The database must be up and running:

```
systemctl enable mysqld
systemctl start mysqld
systemctl status mysqld
```

It's highly suggested to run the “mysql\_secure\_installation” program shipped with the MariaDB package to properly secure your database.

```
mysql_secure_installation
```

HINT: the SQL root user is NOT the root user of the operating system! The default password of the SQL root password is empty.

- set a root password “suse1234”
- remove user “anonymous”
- disallow remote login for user root
- remove test database
- reload privilege tables

There must be a database and a user for “portus”.

Start the mysql prompt by typing

```
mysql -u root -p
```

and enter the mysql user “root” password.

Then enter the following commands:

```
5>4+=4 3S4> Eportuse@ElocalhostE <D4N=<F<4D GH Esuse62C:EI
5>4+=4 D+=+G+S4 portus&pro uctionI
?>+N= +,, .N portus&pro uction2# =. Eportuse@ElocalhostEI
'uit
```

This is going to create a database named “portus\_production” and a user named “portus” with password “portus\_password” that has full privileges over the “portus\_production” database.

## 5.4 Configuring Portus

The Portus RPM provides a tool named “portusctl” that can be used to perform different operations against Portus. The tool must be run as root user.

Copy SSL Key and Certificate

```
cp -av !etc!ssl!servercerts!serverkey2pem J
!etc!apache2!ssl2key!$(hostname -f)-ca2key
cp -av !etc!ssl!servercerts!servercert2pem J
!etc!apache2!ssl2crt!$(hostname -f)-ca2crt
```

Ensure that apache user (wwwrun/www) have read rights to the keys.

```
cho$ root1$$$ !etc!apache2!ssl2key!$(hostname -f)-ca2key
chmo 9;; !etc!apache2!ssl2key!$(hostname -f)-ca2key
```

Hint: After applying the latest patches on a server the permissions did not work properly anymore! Due to that it might be required to adjust the link within portus:

```
rm !srv!/ortus!confiA!server2key
cp -av !etc!ssl!servercerts!serverkey2pem !srv!/ortus!confiA!server2key
```

To complete the configuration of Portus and of the Docker registry just type:

```
portusctl setup -- (-username-portus J
-- (-pass$or -suse62C: J
--local-reAistry
```

The portusctl setup is going to perform the following operations:

- Populate the database of Portus
- Create an Apache configuration for Portus
- Create all the certificates and keys required by Portus and by the Docker registry
- Configure the Docker registry to use Portus as authorization service
- Configure the Docker registry to send notifications to Portus

You can obtain more information about “portusctl” and its sub-commands by typing:

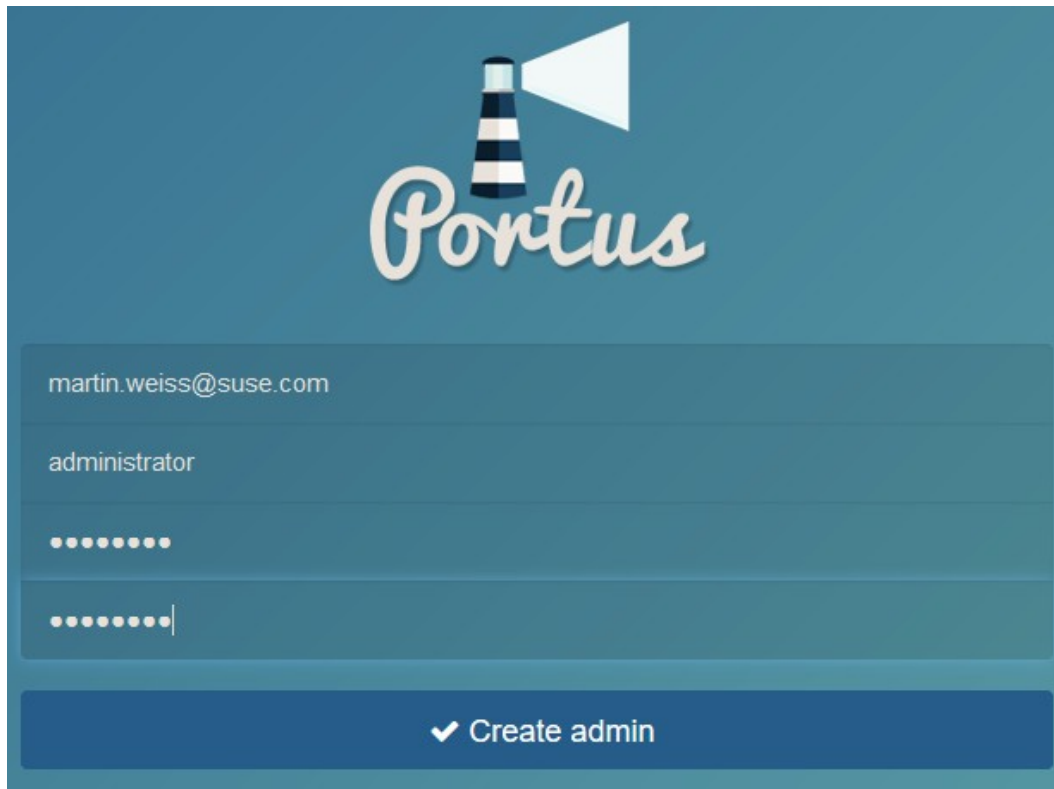
```
portusctl help
portusctl help <su(-comman >
```

## 5.5 Creating the first Administrator User

To complete the installation of Portus you have to log into its web interface.

Open the following url with your browser <https://<ip of your VM>> and then create the administrator user.

HINT: Use what ever email address and username what you want. Make sure the password has at least 8 characters (i.e. suse1234).

The screenshot shows the Portus web interface. At the top, there is a logo featuring a lighthouse and the word 'Portus' in a stylized font. Below the logo, there is a form with four input fields: an email address field containing 'martin.weiss@suse.com', a username field containing 'administrator', a password field with eight dots, and a confirmation password field with eight dots and a cursor. At the bottom of the form is a blue button with a white checkmark and the text 'Create admin'.

Then provide the information about the registry:

- name: whatever you want
- hostname: get the hostname of your VM (it should be "sles15sp1-1.suse") and then enter <hostname of your VM>:5000. It should be "sles15sp1-1.suse:5000" for this exercise

HINT: the SSL certificate will be verified so the subject name needs to match what is specified here!

- ensure the "use ssl" checkbox is enabled



### New Registry

Portus needs the information from the form below in order to fetch information from your registry and keep everything up to date (e.g. so tags are added after users push to the registry). Make sure that:

- If you check the SSL checkbox, SSL is properly configured on both sides.
- You include the port to the hostname (e.g. "registry.test.lan:5000").

If you have any doubts about this, just check our [documentation](#). In particular, make sure to check [how to configure Portus](#) and [how to configure your private registry](#).

**Name**

**Hostname**

**Use SSL**

☒

**Create**

## 5.6 Organization of a Registry

With Portus enabled a series of rules apply to your private registry. Instead of being able to read and write everywhere, you are going to have access to precise locations of the registry.

All the users have a private namespace that is named after their login name on Portus. By default all the repositories inside of this namespace are private, meaning they are hidden from the rest of the users. However it's always possible to make a personal namespace public.

Each registry has a global namespace. This is readable by everybody, but only Portus' administrators can actually push images there. This is the very same approach used by the Docker Hub.

You can create as many teams as you want on Portus. Teams have three types of members:

- viewers: they can only pull images
- contributors: they can pull and push images
- owners: like contributors, but can also add/remove/edit the members of the team plus the namespaces

Each team can control one or more namespaces. By default all the namespaces are private, but this can be changed by toggling the "public" switch.

## 5.7 Playing with Portus (1)

Try to accomplish the following operations:

- Create a new team
- Create a new user

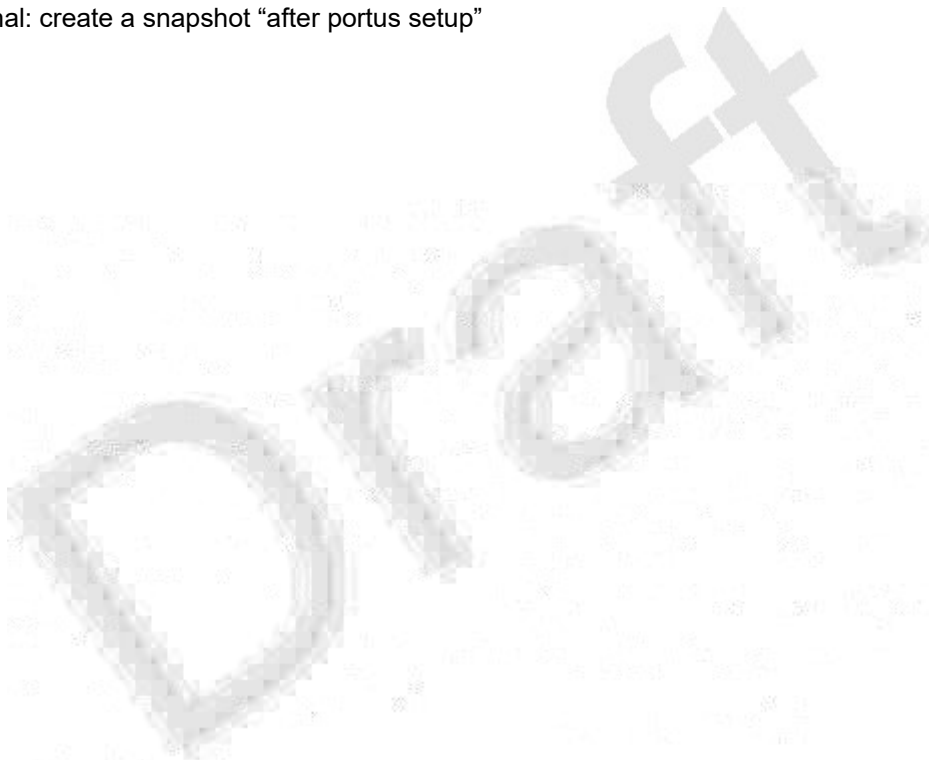
HINT: you can use the private browsing mode of your browser to make things easier

HINT: user names must be all lower case and must not have special characters

- Add, remove, edit members of a team
- Add one or more namespaces to a group

→ Create a namespace “base\_images” and ensure your users have “contributors” access to that namespace!

Optional: create a snapshot “after portus setup”



## 6 Exercise: Implementation of the Docker Distribution Registry and Portus (SLES 15)

Snapshot "after docker installation"

Goal: Installation and configuration of the docker distribution registry with portus

Portus is the authorization service and front-end for the docker distribution registry. It is not an alternative registry, it's a component to be used in addition to the docker distribution registry to secure your docker images by adding authentication and authorization.

Hint: while SUSE delivered Portus and the docker-distribution-registry as RPMs in SLES 12, this is now changed in SLES 15 and the delivery happens as images for the services MariaDB, docker-distribution-registry and Portus.

### 6.1 SSL Certificate Creation

To secure the communication it is required to create SSL certificates. In this exercise we are creating a new Certificate Authority and a server key and a wildcard SSL certificate.

Create a place for the certificates

```
su o mk ir -p ! ata!certificates
```

Copy the following files to /data/certificates

```
; -create-re' -conf2sh
6 -create-ca2sh
2 -create-$il car -key2sh
C -create-$il car -csr2sh
: -create-$il car -cert2sh
varia(les2t)t
```

Mark the scripts as executable

```
su o chmo K) ! ata!certificates!#2sh
```

Give joedoe rights:

```
su o cho$n -> *oe oe!users ! ata!certificates
```

Adjust the variables.txt to your environment

Domain needs to be the domain you have configured as search in /etc/resolv.conf

The CA\_CN should be the FQDN of your server (hostname -f)

```
D.@+<N- "suse"
5- "D4"
S=- "Ga en-0uer ttem(era"
, - " "
```

```
.-"0eiss"  
.3-" "  
5+&5N-"sles67sp6-62suse"  
4@+<,-"martin2$eiss@suse2com"  
5N-"#2$D.@+<N"  
S+NS-"#2$D.@+<N"% "#2cap2$D.@+<N"% "#2uaa2cap2$D.@+<N"%api2cap2$D.@+<N"
```

After the variables.txt is adjusted to your environment execute

```
c ! ata!certificates  
2!;-create-re'-conf2sh
```

to create the proper req.conf file.

Now create the Certificate Authority using the following command:

```
c ! ata!certificates  
2!6-create-ca2sh
```

Next step is to create the wildcard key.

```
c ! ata!certificates  
2!2-create-$il car -key2sh
```

Now the certificate signing request needs to be created

```
c ! ata!certificates  
2!C-create-$il car -csr2sh
```

And finally the certificate needs to be signed by the CA

```
c ! ata!certificates  
2!:-create-$il car -cert2sh
```

## 6.2 Load Registry Images into local image repository

Hint: this is required in case the container host does not have access to the internet or a local registry where the images can be fetched from!

Hint: they were saved this way (do not execute these commands):

```
c ! ata!reAistry  
po man save -o reAistry2suse2com-sles62-portus-22:2C2tar J  
reAistry2suse2com!sles62!portus122:2C  
po man save -o reAistry2suse2com-sles62-reAistry-229222tar J  
reAistry2suse2com!sles62!reAistry122922  
po man save -o reAistry2suse2com-sles62-maria (-6;2;2tar J  
reAistry2suse2com!sles62!maria (16;2;
```

Copy image files to /data/registry

Load the images into the local repository (via docker)

```
c ! ata!reAistry  
ocker loa -i reAistry2suse2com-sles62-portus-22:2C2tar  
ocker loa -i reAistry2suse2com-sles62-reAistry-229222tar  
ocker loa -i reAistry2suse2com-sles62-maria (-6;2;2tar
```

Load the images into the local repository (podman)

```
c ! ata:reAistry
podman load -i reAistry2suse2com-sles62-portus-22:2C2tar J
reAistry2suse2com:sles62!portus122:2C
podman load -i reAistry2suse2com-sles62-reAistry-229222tar J
reAistry2suse2com:sles62!reAistry122922
podman load -i reAistry2suse2com-sles62-maria (-6:2:2tar
reAistry2suse2com:sles62!maria (16:2;
```

## 6.3 Installation of Docker Distribution Registry

Create script directory

```
su o mk ir -p ! ata:reAistry
```

Copy registry.sh to /data/registry

Adjust rights:

```
su o cho$ n -> *oe oe1users ! ata:reAistry
su o chmo K) ! ata:reAistry!#2sh
```

Adjust variables in registry.sh for your environment

```
c ! ata:reAistry
vi reAistry2sh
```

execute the script

```
c ! ata:reAistry
su o 2!reAistry2sh
```

## 6.4 Prepare Portus MySQL Database

Create script directory

```
su o mk ir -p ! ata:reAistry
```

Copy mysql.sh and mysql-entripoint.sh to /data/registry

Adjust rights:

```
su o cho$ n -> *oe oe1users ! ata:reAistry
su o chmo K) ! ata:reAistry!#2sh
```

Adjust variables in mysql.sh for your environment

```
c ! ata:reAistry
vi mys'l2sh
```

execute the script

```
c ! ata:reAistry
su o 2!mys'l2sh
```

Create the portus database

```
su o po man e)ec -it mys'l (ash
mys'l -u root -p
```

```
5>4+=4 3S4> Eportuse@ELE <D4N=<F<4D GH Esuse62C:EI
5>4+=4 D+=+G+S4 portus&pro uctionI
?>+N= +,, .N portus&pro uction2# =. Eportuse@ELEI
'uit
e)it
```

#### Debugging

```
su o po man e)ec -it mys'l (ash
mys'l -u portus -p
sho$ ata(asesI
'uit
e)it
```

Or

```
su o po man e)ec -it mys'l (ash
mys'l -u root -p
select host% user% pass$or from mys'l2userI
'uit
e)it
```

## 6.5 Installation of Portus

Create script directory

```
su o mk ir -p ! ata!reAistry
```

Copy portus.sh and portus-background.sh to /data/registry

Adjust rights:

```
su o cho$n -> *oe oe1users ! ata!reAistry
su o chmo K) ! ata!reAistry!#2sh
```

Adjust variables in portus.sh and portus-background.sh for your environment

```
c ! ata!reAistry
vi portus2sh
vi portus-(ackAroun 2sh
```

execute the scripts

```
c ! ata!reAistry
su o 2!portus2sh
su o 2!portus-(ackAroun 2sh
```

## 6.6 Configuring Portus

You can obtain more information about “portusctl” and its sub-commands by typing:

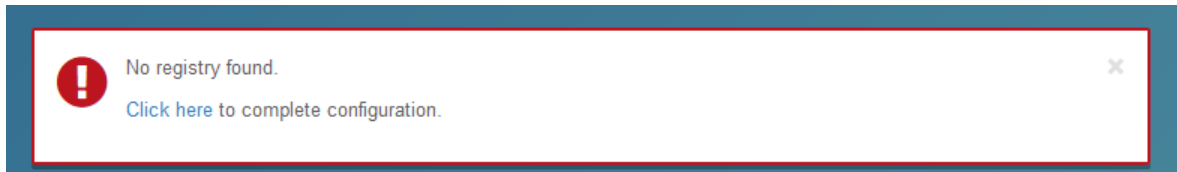
```
ocker e)ec -it portus portusctl help
ocker e)ec -it portus portusctl help <su(-comman >
```

## 6.7 Configure the Registry in Portus

To complete the installation of Portus you have to log into its web interface.

Open the following url with your browser `https://<ip of your VM>:3000` and login with the user "portus" with the password "supercomplexpassword!"

Then click on this link:



Then provide the information about the registry:

- name: whatever you want (best practice is to specify the FQDN of the registry)
- hostname: get the FQDN (hostname -f) of your VM and then enter <hostname of your VM>:5000.

HINT: the SSL certificate will be verified so the subject name needs to match what is specified here!

- ensure the "use ssl" checkbox is enabled

A screenshot of the "New Registry" form in the Portus web interface. The form has a title "New Registry" and a paragraph explaining that Portus needs information from the form to fetch information from the registry and keep everything up to date. It lists two requirements: checking the SSL checkbox and including the port to the hostname. Below this, there are three input fields: "Name" with the value "sles15sp1-1.suse", "Hostname" with the value "sles15sp1-1.suse:5000", and "Use SSL" with a checked checkbox. At the bottom, there are two buttons: "Create" and "Show Advanced".

New Registry

Portus needs the information from the form below in order to fetch information from your registry and keep everything up to date (e.g. so tags are added after users push to the registry). Make sure that:

- If you check the SSL checkbox, SSL is properly configured on both sides.
- You include the port to the hostname (e.g. "registry.test.lan:5000").

If you have any doubts about this, just check our [documentation](#). In particular, make sure to check [how to configure Portus](#) and [how to configure your private registry](#).

Name

Hostname

Use SSL ☒

Then click on "create".

Registry was successfully created.
✕

Registries

| Name             | Hostname              | External hostname | SSL                      | Reachable |                      |
|------------------|-----------------------|-------------------|--------------------------|-----------|----------------------|
| sles15sp1-1.suse | sles15sp1-1.suse:5000 |                   | <input type="checkbox"/> |           | <a href="#">Edit</a> |

**Note well:** right now Portus is designed to handle only a single private Registry.

## 6.8 Creating the first Administrator User

Now it is time to create the first administrative user.

Click on “Users” and then on “Create new user”

The screenshot shows the Portus web interface. On the left is a dark sidebar with menu items: Dashboard, Namespaces, Repositories, Teams, Users (highlighted), Registries, and Help. The main content area has a top bar with a search box and the Portus logo. Below the top bar, the 'Registered users' section is displayed. It includes a 'Create new user' button in the top right corner. Below this button is a table with columns: Name, Email, Admin, Namespaces, Teams, Enabled, Bot, and Remove. The table currently shows 'No entry'.

Then create the administrator user.

Username

Email

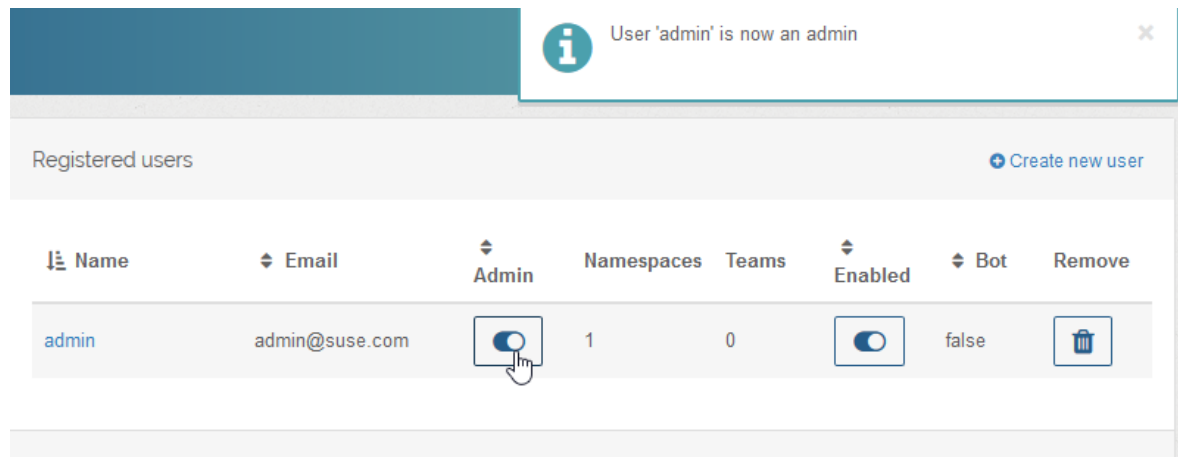
Password

Password confirmation



HINT: Use what ever email address and username what you want. Make sure the password has at least 8 characters (i.e. suse1234).

Then make this user “admin”



Finally logout and login as “admin”.

## 6.9 Organization of a Registry

With Portus enabled a series of rules apply to your private registry. Instead of being able to read and write everywhere, you are going to have access to precise locations of the registry.

All the users have a private namespace that is named after their login name on Portus. By default all the repositories inside of this namespace are private, meaning they are hidden from the rest of the users. However it's always possible to make a personal namespace public.

Each registry has a global namespace. This is readable by everybody, but only Portus' administrators can actually push images there. This is the very same approach used by the Docker Hub.

You can create as many teams as you want on Portus. Teams have three types of members:

- viewers: they can only pull images
- contributors: they can pull and push images
- owners: like contributors, but can also add/remove/edit the members of the team plus the namespaces

Each team can control one or more namespaces. By default all the namespaces are private, but this can be changed by toggling the “public” switch.

## 6.10 Playing with Portus (1)

Try to accomplish the following operations:

- Create a new team
- Create a new user

HINT: you can use the private browsing mode of your browser to make things easier

HINT: user names must be all lower case and must not have special characters

- Add, remove, edit members of a team
- Add one or more namespaces to a group

→ Create a namespace “base\_images” and ensure your users have “contributors” access to that namespace!

Optional: create a snapshot “after portus setup”



# 7 Exercise: Pull an Image from a Registry

Snapshot “after portus setup”

## 7.1 Pull an Image from a Registry

Use “docker or podman pull” on any docker host to pull an image from the docker registry:

```
podman pull <registry>[:<port>]!<namespace>!<image>[:<version>]
```

Example:

```
podman pull sles67sp6-62suse17;;;!(ase&image:sles62spC12:2BD
```

→ Try this on or against any other server in the group!

→ What happens and why?

## 7.2 Adding the CA Certificate as Trusted Certificate

First copy the public key of the signing CA of the certificate used for the registry to the store of trusted CAs on SLES:

```
su o scp sles67sp6-62suse1! ata!certificates!root5+#2crt J
!etc!pki!trust!anchors!
```

Then execute the command to update trusted CA certificates in /etc/ssl/certs:

```
su o !usr!s(in!update-ca-certificates
```

After that a restart of docker is required (in case docker is used instead of podman):

```
su o systemctl restart docker
```

→ Try the pull again!

→ What happens and why?

→ Fix the problem and pull all four images in the base\_images namespace.

## 7.3 Playing with Portus (3)

Try to accomplish the following operations:

- Pull images from different namespaces
- Play with the public attribute of a namespace

## 8 Exercise: Get Images and push them to the Registry

Snapshot “after portus setup”

### 8.1 Obtaining The Official SLE Images For Docker (SLES 12)

SUSE builds and maintains the official pre-built SLES images for Docker.

Right now images are available:

- SLES 11 SP3
- SLES 11 SP4
- SLES 12
- SLES 12 SP1
- SLES 12 SP2
- SLES 12 SP3

The images are distributed as traditional packages via the “containers” module. To install them just execute:

```
zypper in <os>- ocker-imaAe
```

For example:

```
zypper in sles66spC- ocker-imaAe sles66sp:- ocker-imaAe sles62- ocker-imaAe  
sles62sp6- ocker-imaAe sles62sp2- ocker-imaAe suse-sles62spC-imaAe
```

The images from sles12sp3 on are named

→ Install all images!

The images have been downloaded on your computer to /usr/share/suse-docker-images, but are not visible by Docker yet.

To do that images need to be activated.

### 8.2 Activating Docker Images (SLES 12)

The “sle2docker” tool helps the customer to activate SUSE’s official images. The tool is part of the “containers” module. In case it is not yet installed it can be installed by doing:

```
zypper in sle2 ocker
```

The tool requires Docker up and running. The user invoking the tool must be able to interact with the Docker daemon.

To list all the pre-built Docker images available on your system:

```
sle2 ocker list
```

To activate a Docker image just execute:

```
sle2 ocker activate <name of the pre-built image as reported above>
```

Now your official SLE image for Docker is ready to be used.

→ Activate all installed images!

Verify that all activated images are visible to docker with

```
docker images
```

## 8.3 Naming / Tagging Images

Creating a tag is really easy (and does not waste disk space):

```
docker tag suse/sles62sp6:2: *oe oe/sles62sp6-test162:2:
```

or

```
podman tag suse/sles62sp6:2: *oe oe/sles62sp6-test162:2:
```

Check list of images again with

```
docker images
```

or

```
podman images
```

→ An image can have multiple names! → Have a look at the “IMAGE ID”!

It can also be used to rename images (add a new name/tag and remove the old name/tag):

```
docker tag *oe oe/sles62sp6-test162:2: *oe/sles162:2BB
```

An image can be “untagged” by deletion of that image (rm image = rmi):

```
docker rmi *oe/sles162:2BB
```

## 8.4 Authenticating to the Registry

Before being able to push images to the registry a login is required. This can be done with the following command:

```
docker login \
-u <username> \
-p <password> \
sles67sp6-62suse17;;
```

Example:

```
docker login -u admin -p suse62C: sles67sp6-62suse17;;
```

Example with podman:

```
podman login -u admin -p suse62C: sles67sp6-62suse17;;
```

Now you are successfully logged into the private registry. You can do push and pull operations without having to enter your credentials anymore.

Note well: Docker stores the credentials in encrypted format inside of the currently logged in users <home directory>/docker/config.json.

Note well: Podman stores the credentials in encrypted format in  
/run/user/<uid>/containers/auth.json

To delete the credentials a logout is required:

```
docker logout <registry>1<port>
```

Example:

```
docker logout sles67sp6-62suse17::;
```

→ Login with a user that is contributor to the base\_image namespace.

## 8.5 Pushing Images To The Registry

It's time to push your images to the Docker registry. To do that you have to tag an existing image in the proper way. Assume we want to push the image to your personal namespace (username = personal namespace):

```
docker tag J  
<image name>1<image version>J  
sles67sp6-62suse17::;!!<username>!<image name>1<image version>  
docker push J  
sles67sp6-62suse17::;!!<username>!<image name>1<image version>
```

Example for push to personal namespace:

```
docker tag suse:sles62sp6162;2: J  
sles67sp6-62suse17::;!!*oe oe:sles62sp6162;2:  
docker push sles67sp6-62suse17::;!!*oe oe:sles62sp6162;2:
```

Example for push to namespace "base\_images":

```
docker tag suse:sles62sp6162;2: J  
sles67sp6-62suse17::;!!(ase&image:sles62sp6162;2:  
docker push sles67sp6-62suse17::;!!(ase&image:sles62sp6162;2:
```

→ Push all images to the registry

## 8.6 Playing with Portus (2)

Try to accomplish the following operations:

- Push images to different namespaces using different users

→ Check out the "Search Repository" with different users.

→ Check out the "Recent activities".

## 9 Exercise: Using Images and Containers

This exercise is to get familiar with the docker commands. Play around with all the commands.

### 9.1 List images

To list the available Docker images:

```
docker images
```

or

```
podman images
```

Check the “repository name” of the image.

Check the “tags” compared to “image ID”.

Compare the “image ID” with the sub-directories in

```
ls -l $(find /var/lib/docker -type d -name $(cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 40 | tr -d '\n' | xargs sha256sum | cut -d ' ' -f 1))
```

or with podman in:

```
ls -l $(find /var/lib/containers/storage -type d -name $(cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 40 | tr -d '\n' | xargs sha256sum | cut -d ' ' -f 1))
```

### 9.2 Tag Images

Creating a tag is really easy (and does not waste disk space):

```
docker tag registry2.suse.com/sles62sp12:2BD sles62sp1C2C2C
```

It can also be used to rename images:

```
docker tag sles62sp1C2C2C our&awesome&sles62sp16222C
```

To remove a tag use (remove image):

```
docker rmi <name>
```

### 9.3 Pull images

To pull an image just execute:

```
docker pull <name of the image>
```

This will try to download the image from the central index. To download the image from a private registry just do:

```
docker pull <host of the private registry>:<name of the image>
```

In our case:

```
docker pull sles67sp6-62suse17:latest (ase&images:sles66sp:)
```

This will pull the official SLES11SP4 image with the tag “latest” in case it exists. In case “latest” does not exist - the version tag (i.e. 1.1.1) needs to be specified.

## 9.4 Push images

If you want to share your Docker image you can push it to a registry. By default Docker tries to push the image to its central (and public) registry: Docker Hub. However it is possible to push the image to a private registry.

To push an image just do:

```
docker push <name of the image>
```

This would push the image to the “public” registry.

To push the image to a private registry:

```
docker push <host of the private registry>:<namespace>:<name of the image>
```

In our case the command would be:

```
docker push our&a$esome&sles62spc16222c
```

Before doing that we have to embed the registry FQDN and the namespace into the full image name:

```
docker tag our&a$esome&sles62spc16222c j  
sles67sp6-62suse17;;;!(ase&image!our&a$esome&sles62spc16222c
```

Now we can push it.

```
docker push sles67sp6-62suse17;;;!(ase&image!our&a$esome&sles62spc16222c
```

## 9.5 Get the history of an image

Use the command

```
docker history <image-name>1<version>
```

to see the changes an image experienced in the past.

## 9.6 Run an interactive container

To have a running console inside of a container execute:

```
docker run -it --rm registry2suse2com:suse/sle671672629222B9 /bin/ash
```

Try to do some changes inside of the container

```
touch foo  
rm /etc/passwd
```

Then exit the container by exiting the bash session.

Let's start a container from the same image:

```
docker run -it --rm registry2suse2com:suse/sle671672629222B9 /bin/ash
```

As you will notice the changes made to the previous container have been lost.

HINT: use --name <my-container-name> to give the container a pre-defined name



HINT: use `-h <my-containers-hostname>` to use this hostname within a container instead of the image-ID.

## 9.7 Attach to and detach from running containers

To detach from a container with leaving it running use “CTRL+p” and then “CTRL-q” for quit.

To find the container name/id type:

```
docker ps
```

To re-attach to the container use

```
docker attach <container-i >
```

## 9.8 List running containers

Start a container and make some changes to it:

```
docker run -it --rm registry2suse2com:suse/sle671672629222B9 ! (in! (ash
touch !root!foo
```

From another console type:

```
docker ps
```

This will return all the running containers.

To see also non active containers use

```
docker ps -a
```

## 9.9 Stop containers

To stop a container just execute:

```
docker stop <i or name of the running container>
```

If the container does not respond it is possible to kill it:

```
docker kill <i or name the running container>
```

## 9.10 Inspect changes made to a running container

Start a new container with an interactive session session:

```
docker run -ti registry2suse2com:suse/sle671672629222B9 ! (in! (ash
```

Alter the file system of the container:

```
touch !foo
rm !etc!passwd
mkir !tmp!new$& ir
```

From the shell used to run the “docker ps” command execute:

```
docker diff <i of the running container>
```

This will list the changes made to the container.

## 9.11 Save changes made to a container

To save the changes made to a running container you have to use the “commit” command. Keep the container running and from another shell execute:

```
docker commit <id of the running container> <target>
```

Target can have different values:

- registry.suse.com/suse/sle15:2.0.0: this is going to create a new version called “2.0.0” of the “sle15” image.
- registry.suse.com/suse/sle15:15.1.6.2.96: this is going to overwrite the image from which the container started.
- registry.suse.com/suse/new\_sles15sp1: this is going to create a new image called “new\_sles15sp1”.

Let's create a new image:

```
docker commit <id of the running container> registry.suse.com/suse/sles15sp1
```

## 9.12 Configure a container for autostart

To configure automatic start / restart of docker containers the --restart option can be used:

```
docker run -it --restart <restart value> <image> <executable>
```

Test this with restart value of “unless-stopped”.

Possible options: “(no, on-failure[:max-retry], always, unless-stopped)”

Hint: podman does not support “unless-stopped”

## 9.13 Network in a container

### 9.13.1 Network addresses translation

Docker containers live in their dedicated network segment. To expose a service running inside of a container to the external world it is necessary to create a port mapping between the host system and the container. That is achieved by doing:

```
docker run -p <external port>:<internal port> <docker image> <app>
```

To test this we are going to build our own container running apache with a “mini” web page.

HINT: use your SLES 15 SP1 image from your registry!

Hint: replace 10.1.1.1 with the address of your SMT/RMT server!

```
docker run -it -p 80:80 sles15sp1-62suse17:1.1.1 (ase&image=sles15sp1-62suse17:1.1.1)
!(in)ash

zypper ar --no-Applecheck --type-rpm-m -f http://10.1.1.1:80/ro-ucts/S,4-@ule-Gasesystem/67-S/6-1)89&9:1pro-uct/S,4-@ule-Gasesystem/67-S/6-/ool

zypper ar --no-Applecheck --type-rpm-m -f http://10.1.1.1:80/ro-ucts/S,4-@ule-Gasesystem/67-S/6-1)89&9:1pro-uct/S,4-@ule-Gasesystem/67-S/6-3p-ates

zypper ar --no-Applecheck --type-rpm-m -f http://10.1.1.1:80/ro-ucts/S,4-@ule-Server-+pplications/67-S/6-1)89&9:1pro-uct/S,4-@ule-Server-+pplications/67-S/6-/ool
```

```
zypper ar --no-apachecheck --type-rpm-m -f http://6.262626!S3S4!3p ates!S,4-@o ule-
Server-+pplications!67-S/6!189&9:!up ate! S,4-@o ule-Server-+pplications67-S/6-
3p ates
N sles66!sles621
rm !usr!li(!zypp!pluAins!services!container-suseconnect
N sles66!sles671
rm !usr!li(!zypp!pluAins!services!container-suseconnect-zypp
zypper -n in apache2
!usr!s(in!start&apache2
echo "oello 0orl " > !srv!$$$!ht ocs!in e)2html
```

In this example the apache “within” the container listens on port 80 and that port is mapped to the docker hosts network address on port 8080.

Now open Firefox and visit <http://<ip of your VM>:8080>

Otherwise just do this “outside of the container”:

```
curl http://localhost:8080/
```

### 9.13.2 Separate Network and static IP

It is also possible to create additional networks and use persistent addresses within docker containers:

Create a new network

```
docker network create --driver bridge --subnet 10.26222:12 static-ip
```

or

```
po man network create --driver bridge --subnet 10.26222:12 static-ip
```

Map a network address to a specific network:

```
docker run --net static-ip --ip 6.26222 --restart unless-stopped -it sles67sp6-
62suse17; ; ; (ase&imaAes!sle671672629222B9 !(in!(ash
```

or

```
po man run --net static-ip --ip 6.26222 --restart always -it sles67sp6-
62suse17; ; ; (ase&imaAes!sle671672629222B9 !(in!(ash
```

# 10 Exercise: Build Your Own Image

Create a docker file and use the original SLE image as base

Execute the docker build

Upload this image to the registry.

Execute a container based on that image

## 10.1 How official SLE images handle repositories

The pre-built images do not have any repository configured. They contain a zypper service that contacts either the SUSE Customer Center (SCC) or your Subscription Management Tool (SMT) or Repository Management Tool (RMT) server, according to the configuration of the SLE host that runs the Docker container. The service obtains the list of repositories available for the product used by the Docker image.

You do not need to add any credentials to the Docker image because the machine credentials are automatically injected into the container by the docker daemon. They are injected inside of the “/run/secrets” directory. The same applies to the “/etc/SUSEConnect” file of the host system, which is automatically injected into the “/run/secrets” file.

Note well: The contents of the “/run/secrets” directory are never committed to a Docker image, hence there's no risk of your credentials leaking.

To enable this in the lab setup it is required to register the docker host to the SMT or RMT server:

Hint: replace 10.11.11.2 with the FQDN of your SMT/RMT Server.

```
$aet --no-check-certificate https://10.11.11.2:26626622!repo!tools!clientSetup:S@=2sh
(ash clientSetup:S@=2sh --host 10.11.11.2:26626622
S3S45onnect -p sle-mo ule-containers!6726! )89&9: -r EE
```

Adjust the security settings in /etc/S3S45onnect as the images / containers do not trust the CA that was used for the SMT servers certificate:

Hint: replace 10.11.11.2 with the FQDN of your SMT or RMT server.

```
insecure1 true
url1 https://10.11.11.2:26626622!
lanAuaAe1 /.S<P
```

HINT: if this is not set on “insecure: true” the SMT certificate also needs to be added to the images during the later Dockerbuild:

For SLES 12 and newer:

```
+DD https://10.11.11.2:26626622!smt2crt !etc!pki!trust!anchors!smt2crt
>3N up ate-ca-certificates
>3N zypper --ApA-auto-import-keys ref -s
```

For SLES 11:

```
+DD https://10.11.11.2:26626622!smt2crt !etc!ssl!certs!smt2crt
>3N c&rehash !etc!ssl!certs
>3N zypper --ApA-auto-import-keys ref -s
```

To obtain the list of repositories use the following command within the docker container:

```
zypper ref -s
```

It will automatically add all the repositories to your container. For each repository added to the system a new file is going to be created under “/etc/zypp/repos.d”. The URLs of these repositories include an access token that automatically expires after 12 hours. To renew the token call the `zypper ref -s` command. It is secure to commit these files to a Docker image.

If you want to use a different set of credentials, place a custom “/etc/zypp/credentials.d/SCCcredentials” file inside of the Docker image. It contains the machine credentials that have the subscription you want to use. The same applies to the SUSEConnect file: to override the file available on the host system that is running the Docker container, add a custom “/etc/SUSEConnect” file inside of the Docker image.

## 10.2 Build new Docker images – the proper way

Building docker images using the “docker commit” workflow is conceptually wrong. The best way to build a container image is by using docker’s integrated build system.

Using the docker’s build system the final image will take advantage of the layer feature; by doing that the image will consume less disk space and the deployment will be faster (only the missing layers are going to be sent to the final hosts). Moreover the Docker build system provide access to some Docker specific features (like volumes, entry points, workdir, expose, ...).

Go to the “!ata!\$(app)” directory. Here you can find several files:

`Dockerfile`: this is the heart of the Docker build system. It contains all the build directives.

`index.html`: the html file to serve

`start.sh`: the script to start the application

The `Dockerfile` tells Docker to build a new image containing our web application. The image is based on the `sles15sp1` image.

To build the image:

```
docker build -t <name of the final image> <path to the directory containing the Dockerfile>
```

In our case, given we are inside of the “!share !\$(app)” directory:

```
docker build -t sles67sp6-62suse17;:$(app):2.6 2
```

This will build the image and import it.

Now run a container of the image with

```
docker run -it -p 8080:80 sles67sp6-62suse17;:$(app):2.6
```

and test if the webserver works with

```
curl localhost:8080
```

## 10.3 Use buildah to build an image

```
zypper -n in (util ah
```

```
(u i l a h (u -t s l e s 6 7 s p 6 - 6 2 s u s e 1 7 ; ; ; ! ( a s e & i m a A e s ! $ e ( a p p 1 ; 2 ; 2 6 ! a t a ! $ e ( a p p
```

Draft

# 11 Exercise: Manage Data Inside Of Containers

The default behavior of Docker container is to discard all the data generated during their existence. Sometimes this is not wanted (think about running a database inside of a container). To achieve data persistence Docker offers the “volume” feature.

## 11.1 System volumes

The systems volumes feature allow to share a directory of the host system inside of the running container:

```
docker run -v <ir on the host>:<ir inside of the container> \
    <image> <command>
```

For example:

```
docker run -v !tmp1:$orkshop reAistry2suse2com!suse!sle671672629222B9 \
    touch !$orkshop!file&create &(y& ocker
```

This command will create a container that will create a “file&create &(y& ocker” file inside of the “!tmp” directory of the host machine.

## 11.2 Data volumes

The data volumes feature allows to create and use docker “internal” volumes (can be found in /var/lib/docker/volumes).

To use this feature just invoke:

```
docker run -v <internal path to volume> <image> <command>
```

For example:

```
docker run -v !$orkshop reAistry2suse2com!suse!sle671672629222B9 touch
!$orkshop!file&to&keep
```

Will start a container based on the “sle15” image, create the “/workshop” directory inside of it and then run the touch command. The “/workshop” directory is stored directly on the host system inside of a directory hidden from the user (under “!var!!i(! ocker!volumes”).

The volume can be accessed by new containers by using the “--volumes-from” option at run time.

First of all let's create the volume:

```
docker run -it -v !$orkshop \
    --name test62C: reAistry2suse2com!suse!sle671672629222B9 !(!in!(ash
```

Create something inside of the container and then exit. To access the volume just do:

```
docker run --volumes-from test62C: -it reAistry2suse2com!suse!sle671672629222B9
!(!in!(ash
```

Under the /workshop directory you will find all the contents created during the previous run.

Note well: volumes can be shared between running containers.

HINT: volumes can be managed with the command “docker volume” (create, rm, ls, inspect)

Draft



# 12 Exercise: Patching A Docker Image / Container

## 12.1 Easy patch and update

SUSE provides an easy patch and update solution to inspect and patch Docker images and Docker containers. This solution is called “zypper-docker”

### 12.1.1 Installation

zypper-docker is packaged and distributed via the “containers” module. It can be installed by doing:

```
zypper in zypper-docker
```

Latest versions of zypper, allow zypper-docker to be invoked in this fancy way: “zypper docker <options and args of zypper-docker>”.

However zypper-docker can always be invoked by typing “zypper-docker”.

### 12.1.2 How zypper-docker works

zypper-docker leverages the power of zypper. zypper-docker expects that images have already zypper inside of them, plus they have either a set of repositories already configured or they have our “container-suseconnect” zypper service.

zypper-docker works by invoking a real zypper process inside of a Docker image.

### 12.1.3 Inspecting a Docker image

#### 12.1.3.1 Looking for updates

HINT: Test this with the sles15sp1-1.suse:5000/base\_images/webapp:0.0.1 image and with suseconnect configured previously.

You can list all the pending updates of a docker image by doing:

```
zypper-docker list-updates <docker image>
```

#### 12.1.3.2 Looking for patches

You can also list which patches are missing from a certain docker image:

```
zypper-docker list-patches <docker image>
```

It is possible to see the ID of all the bugzilla issues affecting a Docker image:

```
zypper-docker lp --bugzilla <docker image>
```

It is possible to see all the CVE issues affecting a Docker image:

```
zypper-docker lp --cve <docker image>
```

It is possible to see all the issues affecting a Docker image:

```
zypper-docker lp --issues <docker image>
```

It is possible to see all the patches of a certain category:

```
zypper- ocker lp --category security < ocker imaAe>
```

### 12.1.3.3 Looking for specific issues

It is possible to see if a Docker image is affected by a certain Bugzilla issue:

```
zypper- ocker lp --(uazilla-667CBC9 < ocker imaAe>
```

It is possible to see if a Docker image is affected by a certain CVE issue:

```
zypper- ocker lp --cve-5Q4-2;6B-6D7:C < ocker imaAe>
```

### 12.1.3.4 Checking image status from scripts

You can check whether the image has patches from scripts using the “patch-check” sub command. This command returns different exit codes:

- 0: everything is fine
- 100: there are patches available
- 101: there are security patches available

These exit codes are the very same of the traditional “zypper patch-ckeck” command.

```
zypper- ocker patch-check < ocker imaAe>
```

### 12.1.3.5 Inspecting a running container

A running container can be inspected using the “list-patches-container” and “list-updates-container”. These commands take the very same arguments of “list-updates” and “list-patches”.

Run a new container named “patch-test”

```
ocker run -it --name patch-test sles67sp6-62suse17;;!!(ase&imaAes!$e(app1;2;26  
!(in!(ash
```

Verify the patch status of that container with

```
zypper- ocker list-up ates-container patch-test
```

Note well: the running container is not affected by the inspection. zypper-docker will look at the running container, figure out from which image it was spawned and then spawn a new docker container from the very same image; that container is going to be inspected. We decided to implement this approach to not cause side effects.

## 12.2 Applying patches and updates

zypper-docker can be used to apply patches and updates to a Docker image.

### 12.2.1 Image Versioning

zypper-docker will force the user to create either a new version of the inspected image or a totally new one. It is not possible to overwrite an existing image: this would violate the best practices of Docker image versioning.

We strongly recommend to provide an author and a message to associate with the update/patch operation. All the Docker images are versioned and each layer can have a commit author and a commit message. These information are available when inspecting the docker image with tools like “docker history”. Again, this is just SUSE trying to promote good practices.

## 12.2.2 Patching containers

zypper-docker does not allow to apply patches or updates to a container. This would lead to dangerous situations. If you patch a running container the image from which it was started is not safe. Hence new containers based on the same image are still going to be affected by the issues. Moreover, patching a library or an executable requires the program using them to be restarted. Docker is an application container, meaning the PID 1 of a Docker container is usually a specific program (eg: apache2, mariadb,...). Restarting the PID 1 would cause the container to terminate and, in certain cases, to be automatically restarted from the original, and unpatched, image.

## 12.2.3 Applying all the updates

To create a new version of a Docker image including all the pending updates:

```
zypper- ocker up ate J
--author yourname --message "applyinA all the up ates" J
foo162;2; foo16262;
```

This is going to create a new image named "foo:1.1.0".

Patch your sles15sp1-1.suse:5000/base\_images/webapp:0.0.1 image to the current status.

Check a list of available updates:

```
zypper- ocker list-up ates sles67sp6-62suse17;;;!(ase&imaAes!$e(app1;2;26
```

Update this image and give it a new version-nummer and namespace! (To ensure it does not collide with future releases from SUSE)

```
zypper- ocker up ate J
--author @artin20eiss@suse2comn --message "applyinA up ates C;2;C22;2;" J
sles67sp6-62suse17;;;!(ase&imaAes!$e(app1;2;26 J
sles67sp6-62suse17;;;!(ase&imaAes!$e(app1;2;22
```

## 12.2.4 Applying patches

It is possible to apply all the pending patches:

```
zypper- ocker patch < ocker imaAe> < ocker imaAe2>
```

It is possible apply all the patches related with Bugzilla issues:

```
zypper- ocker patch --(uAzilla < ocker imaAe> < ocker imaAe2>
```

It is possible apply all the patches related with CVE issues:

```
zypper- ocker patch --cve < ocker imaAe> < ocker imaAe2>
```

It is possible to apply all the patches created since a certain date:

```
zypper- ocker patch -- ate HHHH-@@-DD <imaAe> <imaAe2>
```

It is possible to apply all the patches of a certain category:

```
zypper- ocker patch --cateAory security <imaAe> <imaAe2>
```

## 12.2.5 Patching specific issues

It is possible to see apply the patches related with a certain Bugzilla issue:

```
zypper- ocker patch --(uAzilla-667CBC9 <imaAe> <imaAe2>
```

It is possible to apply the patches related with a certain CVE issue:

```
zypper- ocker patch --cve-5Q4-2;6B-6D7:C <imaAe> <imaAe2>
```

Draft

## 13 Exercise: Attach second container for debugging

ToDo

Draft

## 14 Testcases for Self-Build Images

Verify if these cases are handled for your images:

- Ensure that persistent data is stored outside of the container.
- Ensure that upgrades keep existing data.
- Ensure that new containers get started based on a configuration file.
- Control the start and stop process.

There are some test cases required for images after creating them

- run new container without existing data
- run new container with existing data

Draft

# 15 Troubleshooting

## 15.1 Docker logs

`docker logs <container>`

## 15.2 Docker Daemon

Docker daemon debug with “- - e(uA-true” in !etc!sysconfiA! ocker

## 15.3 Docker Registry and Portus

Start registry in foreground

```
systemctl stop reAistry
reAistry !etc!reAistry!confiA2ym!
```

Log of Portus

```
!srv!/ortus!loA!pro uction2loA
```

Log of registry

```
!var!loA!apache2!access&loA
!var!loA!apache2!error&loA
```

## 15.4 Managing Services

Services

```
systemctl start!stop!status reAistry
systemctl start!stop!status ocker
systemctl start!stop!status portus&crono
systemctl start!stop!status apache2
systemctl start!stop!status ocker
```

## 16 End of Document

Draft



< Do Not Delete the Line Feeds after the next Paragraph>

<End of Document>

Draft

Draft