Docker

    - IIS - Default
    - IIS - WCF

- NGINX
    - Link-Sammlung

- IOS
    - Quick and Dirty

- Docker - NAT
- Ubuntu
    - move docker directory
    - How To Set Up a Private Docker Registry on Ubuntu
    - Delete Image from Registry

Windows Server Core

Installation

Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V -All
Enable-WindowsOptionalFeature -Online -FeatureName Containers -All

[Net.ServicePointManager]::SecurityProtocol=[Net.SecurityProtocolType]::Tls12

Install-Module -Name DockerMsftProvider -Repository PSGallery -Force

Linux SubSystem
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart

Windows Server

Zum Installieren von Docker unter Windows Server können Sie ein OneGet-Anbieter-PowerShell-Modul verwenden, das von Microsoft veröffentlicht wurde und den Namen DockerMicrosoftProvider trägt. Dieser Anbieter aktiviert das Containerfeature in Windows und installiert die Docker-Engine und den -Client. Gehen Sie dazu wie folgt vor:

1. Öffnen Sie eine PowerShell-Sitzung mit erhöhten Rechten, und installieren Sie den Docker-Microsoft-PackageManagement-Anbieter aus dem PowerShell-Katalog.

   PowerShell Kopieren

   ```
   Install-Module -Name DockerMsftProvider -Repository PSGallery -Force
   ```

   Wenn Sie zum Installieren des NuGet-Anbieters aufgefordert werden, geben Sie `Y` ein, um auch diesen zu installieren.
2. Verwenden Sie das PackageManagement-PowerShell-Modul, um die neueste Version von Docker zu installieren.

   PowerShell Kopieren

   ```
   Install-Package -Name docker -ProviderName DockerMsftProvider
   ```

   Wenn PowerShell fragt, ob die Paketquelle „DockerDefault“ vertrauenswürdig ist, geben Sie `A` ein, um die Installation fortzusetzen.
3. Nachdem die Installation vollständig ist, starten Sie den Computer neu.

   PowerShell Kopieren

   ```
   Restart-Computer -Force
   ```

Wenn Sie Docker später aktualisieren möchten:

- Überprüfen Sie die installierte Version mit diesem Befehl:

PowerShell Kopieren

```
Get-Package -Name Docker -ProviderName DockerMsftProvider
```

- Suchen Sie die aktuelle Version mit diesem Befehl:

PowerShell Kopieren

```
Find-Package -Name Docker -ProviderName DockerMsftProvider
```

- Wenn Sie bereit sind, führen Sie die Aktualisierung mit diesem Befehl aus:

PowerShell Kopieren

```
Install-Package -Name Docker -ProviderName DockerMsftProvider -Update -Force
```

- Anschließend folgt dann:

PowerShell Kopieren

```
Start-Service Docker
```

Quick And Dirty

Copy And Paste -Sammlung

Quick And Dirty

Create Registry

rem docker run -d -p 5000:5000 --restart=always --name registry -v
D:/DockerRegistry:/var/lib/registry registry:2

docker run -d -p 5000:5000 --restart=always --name registry registry:2

You will need to save the Docker image as a tar file:

docker save -o <path for generated tar file> <image name>
Then copy your image to a new system with regular file transfer tools such as cp, scp or rsync(preferred for big files). After that you will have to load the image into Docker:

docker load -i <path to image tar file>
PS: You may need to sudo all commands.

EDIT: You should add filename (not just directory) with -o, for example:

[Environment]::SetEnvironmentVariable("LCOW_SUPPORTED", "1", "Machine")
docker load -i d:\transfer\registry.tar
docker save -o c:/myfile.tar centos:16

Deaktivieren lässt sich die Container-Unterstützung mit:

[Environment]::SetEnvironmentVariable("LCOW_SUPPORTED", $null, "Machine")

Quick And Dirty

Install Docker ee

Manuell
http://man.hubwiz.com/docset/Docker.docset/Contents/Resources/Documents/docs.docker.com/install/windows/docker-ee.html

--versionen
https://dockermsft.blob.core.windows.net/dockercontainer/DockerMsftIndex.json

--install
```
# Stop Docker service
Stop-Service docker

# Extract the archive.
Expand-Archive docker-18.09.5.zip -DestinationPath $Env:ProgramFiles -Force

# Clean up the zip file.
Remove-Item -Force docker-18.09.5.zip

# Install Docker. This requires rebooting.
$null = Install-WindowsFeature containers

# Add Docker to the path for the current session.
$env:path += ";$env:ProgramFiles\docker"

# Optionally, modify PATH to persist across sessions.
$newPath = "$env:ProgramFiles\docker;" +
[Environment]::GetEnvironmentVariable("PATH",
[EnvironmentVariableTarget]::Machine)

[Environment]::SetEnvironmentVariable("PATH", $newPath,
[EnvironmentVariableTarget]::Machine)

# Register the Docker daemon as a service.
dockerd --register-service

# Start the Docker service.
Start-Service docker
```

Der einfachste Weg, um in Windows Server 2019 die Container-Funktion zu installieren besteht darin, dass der Server über eine Internetverbindung verfügt, und über diesen Weg der Download

der notwendigen Komponenten bei Docker erfolgt.

Auch Container-Images oder andere Funktionen lassen sich über das Internet, zum Beispiel mit der PowerShell recht einfach auf Windows Server 2019 installieren. Zunächst sollte der Docker-Microsoft PackageManagement Provider aus der PowerShell-Gallery auf dem Server installiert werden:

mofcomp

Danach wird die aktuelle Docker-Engine installiert:

Install-Package -Name docker -ProviderName DockerMsftProvider

Die Installation muss noch bestätigt werden, danach wird Docker Enterprise auf dem Server integriert. Mit der PowerShell kann Docker in Windows Server 2019 auch aktualisiert werden:

Install-Package -Name Docker -ProviderName DockerMsftProvider -Update -Force

Start-Service Docker

Nach der Installation sollte der Server neu gestartet werden:

Restart-Computer -Force

Die erfolgreiche Installation kann ebenfalls in der PowerShell überprüft werden:

Get-Package -Name Docker -ProviderName DockerMsftProvider

Wenn die Installation erfolgreich durchgeführt wurde, steht in der PowerShell und der Eingabeaufforderung auch der Befehl "docker" zur Verfügung. Mit diesem kann die installierte Version von Docker überprüft werden:

docker --version

In einem weiteren Beitrag beschäftigen wir uns damit, wie Linux- und Windows-Container mit Docker auf Windows Server 2019 betrieben werden können.

---

inux-Container in Windows Server 2019 betreiben

Windows-Container stellen auf Windows Server 2019 kein Problem dar. Wer Linux-Container betreiben will, benötigt das "LinuxKit". Das ist erst ab Windows Server 2016 Build 16278 auf in Windows Server 2019 verfügbar. In Windows Server 2019 kann die Unterstützung von Linux-Containern mit dem folgenden Befehl aktiviert werden:

[Environment]::SetEnvironmentVariable("LCOW_SUPPORTED", "1", "Machine")

Anschliessend muss der Dienst neu gestartet werden:

Restart-Service docker

Deaktivieren lässt sich die Container-Unterstützung mit:

[Environment]::SetEnvironmentVariable("LCOW_SUPPORTED", $null, "Machine")

Linux-Container auf Windows Server 2019 benötigen Hyper-V. Auf dem Server muss also auch Hyper-V installiert sein.

---

daemon.json

```
{
"debug": true,
"experimental": true
}
```

Quick And Dirty

WSL2

https://code.visualstudio.com/blogs/2020/03/02/docker-in-wsl2

https://docs.microsoft.com/de-de/windows/wsl/install-manual

Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux

#curl.exe -L -o ubuntu-1804.appx https://aka.ms/wsl-ubuntu-1804
curl -o ubuntu-2004.appx https://aka.ms/wslubuntu2004
#Invoke-WebRequest https://aka.ms/wslubuntu2004

Rename-Item ubuntu-2004.appx ubuntu-2004.zip
Expand-Archive ubuntu-2004.zip ubuntu2004

cd ubuntu2004
.\ubuntu2004.exe

sudo apt update && sudo apt upgrade
sudo apt install ansible

Add your distro path to the Windows environment PATH using Powershell:

$userenv = [System.Environment]::GetEnvironmentVariable("Path", "User")
[System.Environment]::SetEnvironmentVariable("PATH", $userenv +
"C:\Users\Administrator\ubuntu2004", "User")
This will enable you to launch your distro from any path by typing the .exe launcher. For example using ubuntu2004.exe.

Note that this will require closing and relaunching PowerShell.

https://blog.nillsf.com/index.php/2020/06/29/how-to-automatically-start-the-docker-daemon-on-wsl2/

https://medium.com/faun/docker-running-seamlessly-in-windows-subsystem-linux-6ef8412377aa

Private Registry - Push

https://docs.docker.com/engine/reference/commandline/push/

Docker Push

Beschreibung

Push endert ein Image oder Repository in eine Registrierung

Nutzung
docker push [OPTIONS] NAME[:TAG]
Erweiterte Beschreibung
Verwenden Sie diese Datei, um Ihre Images für die Docker Hub-Registrierung oder für eine selbst gehostete Datei freizugeben.docker image push

Weitere Informationen zu gültigen Bild- und Tagnamen finden Sie in der Docker-Image-Tag-Referenz.

Das Töten des Prozesses, z. B. durch Drücken während des Laufens in einem Terminal, beendet den Push-Vorgang.docker image pushCTRL-c

Fortschrittsbalken werden während des Docker-Pushs angezeigt, die die unkomprimierte Größe anzeigen. Die tatsächliche Datenmenge, die übertragen wird, wird vor dem Senden komprimiert, sodass die hochgeladene Größe nicht von der Fortschrittsleiste widergespiegelt wird.

Registrierungsanmeldeinformationen werden von docker loginverwaltet.

Gleichzeitige Uploads
Standardmäßig schiebt der Docker-Daemon fünf Ebenen eines Bildes gleichzeitig. Wenn Sie eine Verbindung mit geringer Bandbreite haben, kann dies zu Problemen mit Timeouten führen, und Sie können dies über die Daemon-Option verringern. Weitere Informationen finden Sie in der Daemon-Dokumentation.--max-concurrent-uploads

Verwenden Sie z. B. diesen Befehl, lesen Sie den Abschnitt "Beispiele" weiter unten.

Optionen
Name, Kurzschrift Standard Beschreibung
--all-tags , -a Drücken Sie alle markierten Bilder in das Repository
--disable-content-trust true Überspringen der Bildsignatur
--quiet , -q Unterdrücken ausführlicher Ausgabe
Beispiele

Push en ein neues Image an eine Registrierung
Speichern Sie zuerst das neue Image, indem Sie die Container-ID (mit Docker-Container ls)suchen und dann an einen neuen Imagenamen übertragen. Beachten Sie, dass nur beim Benennen von Bildern zulässig sind:a-z0-9-_.

$ docker container commit c16378f943fe rhel-httpd:latest
Übertragen Sie das Bild nun mithilfe der Image-ID in die Registrierung. In diesem Beispiel befindet sich die Registrierung auf dem Host nament und lauscht auf Port . Markieren Sie dazu das Bild mit dem Hostnamen oder der IP-Adresse und dem Port der Registrierung:registry-host5000

$ docker image tag rhel-httpd:latest registry-host:5000/myadmin/rhel-httpd:latest

$ docker image push registry-host:5000/myadmin/rhel-httpd:latest
Überprüfen Sie, ob dies funktioniert hat, indem Sie:

$ docker image ls
Sie sollten beides sehen und aufgelistet.rhel-httpdregistry-host:5000/myadmin/rhel-httpd

Drücken Sie alle Tags eines Bildes
Verwenden Sie die Option (oder), um alle Tags eines lokalen Bildes zu drücken.-a--all-tags

Im folgenden Beispiel werden mehrere Tags für ein Image erstellt und alle diese Tags an Docker Hub übertragen.

$ docker image tag myimage registry-host:5000/myname/myimage:latest
$ docker image tag myimage registry-host:5000/myname/myimage:v1.0.1
$ docker image tag myimage registry-host:5000/myname/myimage:v1.0
$ docker image tag myimage registry-host:5000/myname/myimage:v1
Das Bild ist jetzt unter mehreren Namen getaggt:

$ docker image ls

REPOSITORY TAG IMAGE ID CREATED SIZE
myimage latest 6d5fcfe5ff17 2 hours ago 1.22MB
registry-host:5000/myname/myimage latest 6d5fcfe5ff17 2 hours ago 1.22MB
registry-host:5000/myname/myimage v1 6d5fcfe5ff17 2 hours ago 1.22MB
registry-host:5000/myname/myimage v1.0 6d5fcfe5ff17 2 hours ago 1.22MB
registry-host:5000/myname/myimage v1.0.1 6d5fcfe5ff17 2 hours ago 1.22MB
Beim Drücken mit der Option werden alle Tags des Bildes gedrückt:--all-tagsregistry-host:5000/myname/myimage

$ docker image push --all-tags registry-host:5000/myname/myimage

The push refers to repository [registry-host:5000/myname/myimage]
195be5f8be1d: Pushed
latest: digest:
sha256:edafc0a0fb057813850d1ba44014914ca02d671ae247107ca70c94db686e7de6 size: 4527

195be5f8be1d: Layer already exists
v1: digest: sha256:edafc0a0fb057813850d1ba44014914ca02d671ae247107ca70c94db686e7de6 size: 4527
195be5f8be1d: Layer already exists
v1.0: digest: sha256:edafc0a0fb057813850d1ba44014914ca02d671ae247107ca70c94db686e7de6 size: 4527
195be5f8be1d: Layer already exists
v1.0.1: digest: sha256:edafc0a0fb057813850d1ba44014914ca02d671ae247107ca70c94db686e7de6 size: 4527
Übergeordneter Befehl
Befehl Beschreibung
Docker Der Basisbefehl für die Docker CLI.

Private Registry - Search

http://registry:5000/v2/_catalog

Quick And Dirty

Private Registry - Build

https://github.com/StefanScherer/dockerfiles-windows/blob/main/registry/Dockerfile

dockerfile

```
FROM golang as build

SHELL ["powershell", "-Command", "$ErrorActionPreference = 'Stop'; $ProgressPreference = 'SilentlyContinue';"]

ENV DOCKER_BUILDTAGS include_oss include_gcs
ENV DISTRIBUTION_VERSION v2.6.2

RUN mkdir src\github.com\docker ; \
    cd src\github.com\docker ; \
    git clone -q https://github.com/docker/distribution ; \
    cd distribution ; \
    git checkout -q $env:DISTRIBUTION_VERSION ; \
    go build -o registry.exe cmd/registry/main.go



FROM mcr.microsoft.com/windows/nanoserver:sac2016

COPY --from=build /gopath/src/github.com/docker/distribution/registry.exe /registry.exe
COPY config.yml /config/config.yml

EXPOSE 5000

ENTRYPOINT ["\\registry.exe"]
CMD ["serve", "/config/config.yml"]
```

Quick And Dirty

docker-compose recreate

```
docker-compose up --force-recreate --build -d
```

```
docker image prune -f
```

docker-compose install

Versionen: https://github.com/docker/compose/releases

Ubuntu

wget https://github.com/docker/compose/releases/download/v2.2.2/docker-compose-linux-x86_64 .

mv ocker-compose-linux-x86_64 docker-compose

chmod +x docker-compose

mv docker-compose /usr/bin/docker-compose

Fehlerbehebungen

Netzwerkfehler

net stop winnat

container starten

net start winnat

Fehlerbehebungen

ailed during hnsCallRawResponse: hnsCall failed in Win32

Administrative PowerShell console
Stop-Service docker
Stop-Service hns
Start-Service hns
Start-Service docker
docker network prune

hnsCallRawResponse 1

Dieser Fehler weist normalerweise darauf hin, dass bereits eine angepasste Reservierung durch ein vorhandenes vSwitch + HNS-Netzwerk vorhanden ist.

Sie können vorhandene Netzwerke anzeigen mit:
`docker network ls`
`hnsdiag list networks`

Dann, um alle alten Netzwerke zu entfernen:
`docker rm <network_name>`
`hnsdiag delete networks <HNS_ID>`

Auch für den Fall, dass Windows als Master gewählt wurde, sollte der vollständige Init-Befehl lauten:
`docker swarm init --advertise-addr=10.127.132.230 --listen-addr 10.127.132.230:2377`

Befehle

Löscht alle gestoppten Container
Windows -> docker container prune

Linux

One liner to stop / remove all of Docker containers:

2

1

```
docker stop $(docker ps -a -q)
```

2

```
docker rm $(docker ps -a -q)
```

docker save -o <path for generated tar file> <image name>
Then copy your image to a new system with regular file transfer tools such as cp, scp or rsync(preferred for big files). After that you will have to load the image into Docker:

docker load -i <path to image tar file>
PS: You may need to sudo all commands.

Portainer

Manage the Docker environment where Portainer is running.

Ensure that you have started the Portainer container with the following Docker flag:

`-v "/var/run/docker.sock:/var/run/docker.sock"` (Linux).

or

`-v \\.\pipe\docker_engine:\\.\pipe\docker_engine` (Windows).

Remoteverwaltung:

New-NetFirewallRule -DisplayName 'Docker SSL Inbound' -Profile @('Domain', 'Public', 'Private') -Direction Inbound -Action Allow -Protocol TCP -LocalPort 2376

https://docs.microsoft.com/de-de/virtualization/windowscontainers/management/manage_remotehost

traefik

https://github.com/StefanScherer/dockerfiles-windows/tree/main/traefik

Running Docker Windows and Linux Containers Simultaneously

Running Docker Windows and Linux Containers Simultaneously

[Running Docker Windows and Linux Containers Simultaneously | Developer Support (microsoft.com)]

Let’s Get Started

With Docker for Windows started and Windows containers selected, you can now run either Windows or Linux Containers simultaneously. The new *–platform=linux* command line switch is used to pull or start Linux images on Windows.

1

1

```
docker pull --platform=linux ubuntu
```

Now start the Linux container and a Windows Server Core container.

2

1

```
docker run --platform=linux -d ubuntu /bin/sh -c "while true; do echo hello world; sleep 1; done"
```

2

```
docker run -d microsoft/windowsservercore ping -t 127.0.0.1
```

Both containers are running on a single host.

If you list your local image cache you'll see a mixture of both Windows and Linux images. To determine the operating system an image requires you can use *docker inspect* and filter on the "Os" property.

1

1

```
docker inspect --format '{{.Os}}' ubuntu
```

Conclusion

Running Windows and Linux containers simultaneously on the same host is an interesting new feature in Docker with lots of possibilities. However, this is an experimental feature and may have some issues. One known problem is volumes are not stable especially when mapping between Linux and Windows file systems. This can cause some containers that rely heavily on volumes to fail to load. Furthermore, tooling support is not yet complete. For example, Docker-Compose and Kubernetes cannot yet mix Windows and Linux containers. Microsoft is currently tracking issues here and feature progress can be tracked at the Github site here.

Switch

& $Env:ProgramFiles\Docker\Docker\DockerCli.exe -SwitchDaemon

```
$ ./DockerCli.exe
Usage: DockerCli.exe [-SwitchDaemon] [-Version]
-Version: Show the Docker for Windows version information
-SwitchDaemon: Point the Docker CLI to either Linux containers or Windows containers
-SharedDrives: List the shared drives
```

Dockerfiles

IIS - Default

Dockerfile

```
FROM mcr.microsoft.com/windows/servercore/iis
RUN powershell -NoProfile -Command Remove-Item -Recurse C:\inetpub\wwwroot\*
RUN powershell Remove-WebSite -Name 'Default Web Site'
RUN powershell Import-Module WebAdministration;New-Item –Path IIS:\AppPools\WWWPool;New-Item -Path $env:systemdrive\inetpub\www -Type Directory;New-Item -Path $env:systemdrive\inetpub\XpServerDllFiles -Type Directory;New-WebSite -Name XpNDFAsiaSite -Port 4847 -PhysicalPath "$env:systemdrive\inetpub\www" -ApplicationPool WWWPool -Force;

WORKDIR /inetpub/www
COPY content/ .

EXPOSE 4847
```

Build-Batch

```
@echo off
docker build -t myiis:v1.0 .
```

IIS - WCF

Dockerfile

```
FROM mcr.microsoft.com/dotnet/framework/wcf:4.8-windowsservercore-ltsc2016

RUN mkdir VSS2005
COPY VSS2005 VSS2005
RUN c:\windows\system32\regsvr32.exe /i /s c:\VSS2005\ssapi.dll

WORKDIR /inetpub/wwwroot
RUN C:\Windows\System32\inetsrv\appcmd set apppool /apppool.name:DefaultAppPool
/enable32BitAppOnWin64:true
#COPY C:\VSProjects\VSSx86\_publish\_PrecompiledWeb\wcfVssNet .
COPY wcfVssNet .
```

Build-Batch

```
@echo off
docker build -t wcfvssnet.v2.0:sac2016 .
```

NGINX

NGINX

Link-Sammlung

https://www.docker.com/blog/how-to-use-the-official-nginx-docker-image/

https://sitegeist.de/blog/typo3-blog/docker-compose-setup-mit-nginx-reverse-proxy.html

https://github.com/nginx-proxy/nginx-proxy

IOS

IOS

Quick and Dirty

https://dev.to/ianito/how-to-emulate-ios-on-linux-with-docker-4gj3

Docker - NAT

Ubuntu

Ubuntu

move docker directory

1. STOP THE DOCKER DAEMON

```
sudo service docker stop
```

2. ADD A CONFIGURATION FILE TO TELL THE DOCKER DAEMON WHAT IS THE LOCATION OF THE DATA DIRECTORY

Using your preferred text editor add a file named **daemon.json** under the directory **/etc/docker**. The file should have this content:

```
{
  "data-root": "/path/to/your/docker"
}
```

of course you should customize the location “/path/to/your/docker” with the path you want to use for your new docker data directory.

3. COPY THE CURRENT DATA DIRECTORY TO THE NEW ONE

```
sudo rsync -aP /var/lib/docker/ /path/to/your/docker
```

4. RENAME THE OLD DOCKER DIRECTORY

```
sudo mv /var/lib/docker /var/lib/docker.old
```

This is just a sanity check to see that everything is ok and docker daemon will effectively use the new location for its data.

5. RESTART THE DOCKER DAEMON

```
sudo service docker start
```

6. TEST

If everything is ok you should see no differences in using your docker containers. When you are sure that the new directory is being used correctly by docker daemon you can delete the old data directory.

```
sudo rm -rf /var/lib/docker.old
```

Follow the previous steps to move docker data directory and you won’t risk any more to run out of space in your root partition, and you’ll happily use your docker containers for many years to come. [

Ubuntu

How To Set Up a Private Docker Registry on Ubuntu

Step 1 — Installing and Configuring the Docker Registry

Docker on the command line is useful when starting out and testing containers, but proves to be unwieldy for bigger deployments involving multiple containers running in parallel.

With Docker Compose, you can write one `.yml` file to set up each container's configuration and information the containers need to communicate with each other. You can use the `docker-compose` command-line tool to issue commands to all the components that make up your application, and control them as a group.

Docker Registry is itself an application with multiple components, so you will use Docker Compose to manage it. To start an instance of the registry, you'll set up a `docker-compose.yml` file to define it and the location on disk where your registry will be storing its data.

You'll store the configuration in a directory called `docker-registry` on the main server. Create it by running:

1. mkdir ~/docker-registry

Navigate to it:

1. cd ~/docker-registry

Then, create a subdirectory called `data`, where your registry will store its images:

1. mkdir data

Create and open a file called `docker-compose.yml` by running:

1. nano docker-compose.yml

Add the following lines, which define a basic instance of a Docker Registry:

~/docker-registry/docker-compose.yml

```
version: '3'

services:
  registry:
    image: registry:2
    ports:
    - "5000:5000"
    environment:
      REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY: /data
    volumes:
      - ./data:/data
```

First, you name the first service `registry`, and set its image to `registry`, version `2`. Then, under `ports`, you map the port `5000` on the host to the port `5000` of the container. This allows you to send a request to port `5000` on the server, and have the request forwarded to the registry.

In the `environment` section, you set the `REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY` variable to `/data`, specifying in which volume it should store its data. Then, in the `volumes` section, you map the `/data` directory on the host file system to `/data` in the container, which acts as a passthrough. The data will actually be stored on the host's file system.

Save and close the file.

You can now start the configuration by running:

1. docker-compose up

The registry container and its dependencies will be downloaded and started.

You'll address the `No HTTP secret provided` warning message later in this tutorial. Notice that the last line of the output shows it has successfully started listening on port `5000`.

You can press `CTRL+C` to stop its execution.

In this step, you have created a Docker Compose configuration that starts a Docker Registry listening on port `5000`. In the next steps, you'll expose it at your domain and set up authentication.

Step 2 — Setting Up Nginx Port Forwarding

As part of the prerequisites, you've enabled HTTPS at your domain. To expose your secured Docker Registry there, you'll only need to configure Nginx to forward traffic from your domain to the registry container.

You have already set up the `/etc/nginx/sites-available/your_domain` file, containing your server configuration. Open it for editing by running:

1. sudo nano /etc/nginx/sites-available/your_domain

Find the existing `location` block:

/etc/nginx/sites-available/your_domain

```
...
location / {
  ...
}
...
```

You need to forward traffic to port `5000`, where your registry will be listening for traffic. You also want to append headers to the request forwarded to the registry, which provides additional information from the server about the request itself. Replace the existing contents of the `location` block with the following lines:

/etc/nginx/sites-available/your_domain

```
...
location / {
    # Do not allow connections from docker 1.5 and earlier
    # docker pre-1.6.0 did not properly set the user agent on ping, catch "Go *" user agents
    if ($http_user_agent ~ "^(docker\/1\.(3|4|5(?!\.[0-9]-dev))|Go ).*$" ) {
      return 404;
    }

    proxy_pass                  http://localhost:5000;
```

```
    proxy_set_header  Host              $http_host;   # required for docker client's sake
    proxy_set_header  X-Real-IP         $remote_addr; # pass on real client's IP
    proxy_set_header  X-Forwarded-For   $proxy_add_x_forwarded_for;
    proxy_set_header  X-Forwarded-Proto $scheme;
    proxy_read_timeout                  900;
  }
  ...
```

The `if` block checks the user agent of the request and verifies that the version of the Docker client is above 1.5, as well as that it's not a `Go` application that's trying to access. For more explanation on this, you can find the `nginx` header configuration in Docker's registry Nginx guide.

Save and close the file when you're done. Apply the changes by restarting Nginx:

1. sudo systemctl restart nginx

If you get an error, double-check the configuration you've added.

To confirm that Nginx is properly forwarding traffic to your registry container on port `5000`, run it:

1. docker-compose up

Then, in a browser window, navigate to your domain and access the `v2` endpoint, like so:

```
https://your_domain/v2
```

You will see an empty JSON object:

```
{}
```

In your terminal, you'll receive output similar to the following:

Output

```
registry_1  | time="2018-11-07T17:57:42Z" level=info msg="response completed" go.version=go1.7.6
http.request.host=cornellappdev.com http.request.id=a8f5984e-15e3-4946-9c40-d71f8557652f
http.request.method=GET http.request.remoteaddr=128.84.125.58 http.request.uri="/v2/"
http.request.useragent="Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_2) AppleWebKit/604.4.7 (KHTML, like
Gecko) Version/11.0.2 Safari/604.4.7" http.response.contenttype="application/json; charset=utf-8"
http.response.duration=2.125995ms http.response.status=200 http.response.written=2 instance.id=3093e5ab-
5715-42bc-808e-73f310848860 version=v2.6.2
registry_1  | 172.18.0.1 - - [07/Nov/2018:17:57:42 +0000] "GET /v2/ HTTP/1.0" 200 2 "" "Mozilla/5.0 (Macintosh;
```

```
Intel Mac OS X 10_13_2) AppleWebKit/604.4.7 (KHTML, like Gecko) Version/11.0.2 Safari/604.4.7"
```

You can see from the last line that a `GET` request was made to `/v2/`, which is the endpoint you sent a request to, from your browser. The container received the request you made, from the port forwarding, and returned a response of `{}`. The code `200` in the last line of the output means that the container handled the request successfully.

Press `CTRL+C` to stop its execution.

Now that you have set up port forwarding, you'll move on to improving the security of your registry.

Step 3 — Setting Up Authentication

Nginx allows you to set up HTTP authentication for the sites it manages, which you can use to limit access to your Docker Registry. To achieve this, you'll create an authentication file with `htpasswd` and add username and password combinations to it that will be accepted.

You can obtain the `htpasswd` utility by installing the `apache2-utils` package. Do so by running:

1. sudo apt install apache2-utils -y

You'll store the authentication file with credentials under `~/docker-registry/auth`. Create it by running:

1. mkdir ~/docker-registry/auth

Navigate to it:

1. cd ~/docker-registry/auth

Create the first user, replacing `username` with the username you want to use. The `-B` flag orders the use of the `bcrypt` algorithm, which Docker requires:

1. htpasswd -Bc registry.password username

Enter the password when prompted, and the combination of credentials will be appended to `registry.password`.

Note: To add more users, re-run the previous command without `-c`, which creates a new file:

```
1. htpasswd -B registry.password username
```

Copy

Now that the list of credentials is made, you'll edit `docker-compose.yml` to order Docker to use the file you created to authenticate users. Open it for editing by running:

1. nano ~/docker-registry/docker-compose.yml

Add the highlighted lines:

~/docker-registry/docker-compose.yml

```
version: '3'

services:
  registry:
    image: registry:2
    ports:
    - "5000:5000"
    environment:
      REGISTRY_AUTH: htpasswd
      REGISTRY_AUTH_HTPASSWD_REALM: Registry
      REGISTRY_AUTH_HTPASSWD_PATH: /auth/registry.password
      REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY: /data
    volumes:
      - ./auth:/auth
      - ./data:/data
```

You've added environment variables specifying the use of HTTP authentication and provided the path to the file `htpasswd` created. For `REGISTRY_AUTH`, you have specified `htpasswd` as its value, which is the authentication scheme you are using, and set `REGISTRY_AUTH_HTPASSWD_PATH` to the path of the authentication file. `REGISTRY_AUTH_HTPASSWD_REALM` signifies the name of `htpasswd` realm.

You've also mounted the `./auth` directory to make the file available inside the registry container. Save and close the file.

You can now verify that your authentication works correctly. First, navigate to the main directory:

1. cd ~/docker-registry

Then, run the registry by executing:

1. docker-compose up

In your browser, refresh the page of your domain. You'll be asked for a username and password.

After providing a valid combination of credentials, you'll see an empty JSON object:

```
{}
```

This means that you've successfully authenticated and gained access to the registry. Exit by pressing `CTRL+C`.

Your registry is now secured and can be accessed only after authentication. You'll now configure it to run as a background process while being resilient to reboots by starting automatically.

Step 4 — Starting Docker Registry as a Service

You can ensure that the registry container starts every time the system boots up, or after it crashes, by instructing Docker Compose to always keep it running. Open `docker-compose.yml` for editing:

1. nano docker-compose.yml

Add the following line under the `registry` block:

docker-compose.yml

```
...
  registry:
    restart: always
...
```

Setting `restart` to always ensures that the container will survive reboots. When you're done, save and close the file.

You can now start your registry as a background process by passing in `-d`:

1. docker-compose up -d

With your registry running in the background, you can freely close the SSH session, and the registry won't be affected.

Because Docker images may be very large in size, you'll now increase the maximum file size that Nginx will accept for uploads.

Step 5 — Increasing File Upload Size for Nginx

Before you can push an image to the registry, you need to ensure that your registry will be able to handle large file uploads.

The default size limit of file uploads in Nginx is `1m`, which is not nearly enough for Docker images. To raise it, you'll modify the main Nginx config file, located at `/etc/nginx/nginx.conf`. Open it for editing by running:

1. sudo nano /etc/nginx/nginx.conf

Find the `http` section, and add the following line:

/etc/nginx/nginx.conf

```
...
http {
        client_max_body_size 16384m;
        ...
}
...
```

The `client_max_body_size` parameter is now set to `16384m`, making the maximum upload size equal to 16GB.

Save and close the file when you're done.

Restart Nginx to apply the configuration changes:

1. sudo systemctl restart nginx

You can now upload large images to your Docker Registry without Nginx blocking the transfer or erroring out.

Step 6 — Publishing to Your Private Docker Registry

Now that your Docker Registry server is up and running, and accepting large file sizes, you can try pushing an image to it. Since you don't have any images readily available, you'll use the `ubuntu` image from Docker Hub, a public Docker Registry, to test.

From your second, client server, run the following command to download the `ubuntu` image, run it, and get access to its shell:

1. docker run -t -i ubuntu /bin/bash

The `-i` and `-t` flags give you interactive shell access into the container.

Once you're in, create a file called `SUCCESS` by running:

1. touch /SUCCESS

By creating this file, you have customized your container. You'll later use it to check that you're using exactly the same container.

Exit the container shell by running:

1. exit

Now, create a new image from the container you've just customized:

1. docker commit $(docker ps -lq) test-image

The new image is now available locally, and you'll push it to your new container registry. First, you have to log in:

1. docker login https://your_domain

When prompted, enter in a username and password combination that you've defined in step 3 of this tutorial.

The output will be:

Output

```
...
Login Succeeded
```

Once you're logged in, rename the created image:

1. docker tag test-image your_domain/test-image

Finally, push the newly tagged image to your registry:

1. docker push your_domain/test-image

You'll receive output similar to the following:

Output

```
The push refers to a repository [your_domain/test-image]
420fa2a9b12e: Pushed
c20d459170d8: Pushed
db978cae6a05: Pushed
aeb3f02e9374: Pushed
latest: digest: sha256:88e782b3a2844a8d9f0819dc33f825dde45846b1c5f9eb4870016f2944fe6717 size: 1150
```

You've verified that your registry handles user authentication by logging in, and allows authenticated users to push images to the registry. You'll now try pulling the image from your registry.

Step 7 — Pulling From Your Private Docker Registry

Now that you've pushed an image to your private registry, you'll try pulling from it.

On the main server, log in with the username and password you set up previously:

1. docker login https://your_domain

Try pulling the `test-image` by running:

1. docker pull your_domain/test-image

Docker should download the image. Run the container with the following command:

1. docker run -it your_domain/test-image /bin/bash

List the files present by running:

1. ls

You will see the `SUCCESS` file you've created earlier, confirming that its the same image you've created:

```
SUCCESS  bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
```

Exit the container shell by running:

1. exit

Now that you've tested pushing and pulling images, you've finished setting up a secure registry that you can use to store custom images.

Ubuntu

Delete Image from Registry

Jedes Repository kann gelöscht werden, indem Sie auf die Shell des Containers
„docker exec -ti --privileged [Repository-Name] bin/sh“
zugreifen und anschließend auf **„/var/lib/registry/docker/registry/v2/repositories/“** zugreifen und den Ordner löschen mit dem Repository-Namen, den Sie löschen möchten, und wenn Sie ein Repository-Tag **„_manifests/tags/“**
löschen möchten, und dem Tag, das mit der Version verknüpft ist, die Sie löschen möchten.

```
sudo find . -name config.yml
```

```
sudo nano .../config.yml
```

```
storage:
  delete:
    enabled: false
```

Sehen wir uns nun an, wie wir dieses Problem Schritt für Schritt lösen können.

---

curl http://192.168.178.100:5000/v2/aspnetcoresecrets_v2/manifests/latest

curl http://192.168.178.100:5000/v2/aspnetcoresecrets_v2/manifests/latest -H 'Accept: application/vnd.docker.distribution.manifest.v2+json'

curl -v -X DELETE http://192.168.178.100:5000/v2/aspnetcoresecrets_v2/manifests/sha????

```
# get all images that start with localhost:32000, output the results into image_ls file
sudo microk8s ctr images ls name~='localhost:32000' | awk {'print $1'} > image_ls
# loop over file, remove each image
cat image_ls | while read line || [[ -n $line ]];
do
    microk8s ctr images rm $line
done;
```

Überschreiben Sie die Registrierungskonfiguration

Gehen Sie zunächst in den laufenden Registrierungscontainer und ändern Sie die vorhandene Konfigurationsdatei, die mit Standardoptionen erstellt wurde.

```
vi /etc/docker/registry/config.yml
```

One liner for deleting images from a v2 docker registry

Just plug in your own values for registry and repo/image name.

```
registry='localhost:5000'
name='my-image'
curl -v -sSL -X DELETE "http://${registry}/v2/${name}/manifests/$(
    curl -sSL -I \
        -H "Accept: application/vnd.docker.distribution.manifest.v2+json" \
        "http://${registry}/v2/${name}/manifests/$(
            curl -sSL "http://${registry}/v2/${name}/tags/list" | jq -r '.tags[0]'
        )" \
    | awk '$1 == "Docker-Content-Digest:" { print $2 }' \
    | tr -d $'\r' \
)"
```

If all goes well

```
* About to connect() to localhost port 5000 (#0)
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 5000 (#0)
> DELETE /v2/my-image/manifests/sha256:14f6ecba1981e49eb4552d1a29881bc315d5160c6547fdd100948a9e30
> User-Agent: curl/7.29.0
> Host: localhost:5000
> Accept: */*
>
< HTTP/1.1 202 Accepted
< Docker-Distribution-Api-Version: registry/2.0
< X-Content-Type-Options: nosniff
< Date: Wed, 15 Nov 2017 23:25:30 GMT
< Content-Length: 0
< Content-Type: text/plain; charset=utf-8
<
* Connection #0 to host localhost left intact
```

Garbage cleanup

Finally, invoke garbage cleanup on the docker-registry container.

For example:

```
docker exec -it docker-registry bin/registry garbage-collect /etc/docker/registry/config.yml
```
