**27 - R710 Proxmox - local Docker registry**

**Local Docker registry** to save internet bandwidth (using 2nd drive) and used by CI build processes.

This document builds upon the previous documents – specifically:

24 - R710 Proxmox Add 2nd disk to run host and Docker, Nomad for minio

# Docker registry and a UI for it using docker:

1. Prepare directories with:  
     
   **cd /mnt/S3andSQS/tmp**  
   **ls -alt**  
   **mkdir docker-registry**  
   **cd docker-registry**  
   **mkdir volume**
2. Using the following command:  
     
   nano docker-compose.yml  
     
   Put the following into the .yml file:  
     
   **version: '3'**

**services:**

**docker-registry:**

**image: registry:2**

**container\_name: docker-registry**

**restart: always**

**ports:**

**- "5000:5000"**

**volumes:**

**- ./volume:/var/lib/registry**

**docker-registry-ui:**

**image: konradkleine/docker-registry-frontend:v2**

**container\_name: docker-registry-ui**

**restart: always**

**ports:**

**- "9080:80"**

**environment:**

**ENV\_DOCKER\_REGISTRY\_HOST: docker-registry**

**ENV\_DOCKER\_REGISTRY\_PORT: 5000**

1. Launch the docker registry with:  
     
   **sudo docker-compose -f docker-compose.yml up -d**
2. Create a shortcut in W10 firefox at url:  
     
   <http://run3:9080>  
     
   and name it:  
     
   **Docker registry-run3**  
     
   [ ??? did I need to do anything with the ports to be able to see 9080 on run 3 from W10 firefox ??? ]
3. To test the registry, we first pull an image from docker hub:  
     
   **docker pull hello-world**
4. Login to docker, using run3’s username and password:

**docker login run3**

1. Check what’s in the registry:

**curl -X GET run3:9080/v2/\_catalog**  
  
or:

curl -u <**username**>:<**password**> -X GET run3:9080/v2/\_catalog

1. Tag and push the image into the local registry:  
     
   **docker tag hello-world:latest run3:5000/hello-world**

**docker push run3:5000/hello-world**

1. Remove the image that was pulled from docker hub  
     
   **docker image remove hello-world:latest**  
     
   And confirm with:  
     
   **docker image list**
2. Pull the tag’d image from the local repository:  
     
   **docker pull run3:5000/hello-world**
3. Run it to confirm:  
     
   **docker run run3:5000/hello-world**
4. Use Portainer to check / confirm what images and containers are in place within docker
5. NOTE: running  
   **docker info**  
   needs to show:  
   .  
   .  
   Registry: https://index.docker.io/v1/

Experimental: false

Insecure Registries:

**192.168.124.162:5000**

127.0.0.0/8

Live Restore Enabled: false

The above is setup by doing (from <https://docs.docker.com/registry/insecure/> ):

**Deploy a plain HTTP registry**

**Warning**: It’s not possible to use an insecure registry with basic authentication.

This procedure configures Docker to entirely disregard security for your registry. This is **very** insecure and is not recommended. It exposes your registry to trivial man-in-the-middle (MITM) attacks. Only use this solution for isolated testing or in a tightly controlled, air-gapped environment.

1. Edit the daemon.json file, whose default location is /etc/docker/daemon.json on Linux or C:\ProgramData\docker\config\daemon.json on Windows Server. If you use Docker Desktop for Mac or Docker Desktop for Windows, click the Docker icon, choose **Settings** and then choose **Docker Engine**.

If the daemon.json file does not exist, create it. Assuming there are no other settings in the file, it should have the following contents:

{

"insecure-registries" : ["http://**192.168.124.162**:5000"]

}

Substitute the address of your insecure registry for the one in the example.

With insecure registries enabled, Docker goes through the following steps:

* + First, try using HTTPS.
  + If HTTPS is available but the certificate is invalid, ignore the error about the certificate.
  + If HTTPS is not available, fall back to HTTP.

1. Restart Docker for the changes to take effect.

Repeat these steps on every Engine host that wants to access your registry.

1. After building many images **Docker can eat up a lot of disk space**. This can be reclaimed with these commands, bu first check current free disc space with:  
   **df -h**

Then do:  
**docker image prune -a**

And check if any improvemet with:  
**df -h**

Then try:  
**docker builder prune**  
  
And check what impact this has had, again with:  
**df -h**

1. That should be it.