Igor Cicimov
DevOps at Encompass

Follow

Docker Private Registry with S3 backend on AWS

5 minute read , May 03, 2017

Motivation

Our current Docker Hub Registry at https://hub.docker.com provides for a single private repository. This means all our private images must be stored there which prevents from proper versioning via labels. Setting up this repository is an alternative to using AWS ECR service for the same purpose.

Setup

S3 bucket

We start by creating S3 bucket for the Docker Registry:

```
$ aws s3api create-bucket --bucket my-bucket --region ap-southeast-2 --create-bucket-configuration LocationConstraint=ap-sou
$ aws s3api put-bucket-versioning --region ap-southeast-2 --bucket my-bucket --versioning-configuration Status=Enabled
```

We create the bucket in the same region as our Git/Registry server and have versioning enabled so we can recover to previous state of the images in case of issues.

IAM Setup

Next is the IAM setup to control the bucket access. We add IAM Instance Profile that allows access to the DockerHub S3 bucket and attach it to the EC2 instance that will be running our Docker Registry. The json file `DockerHubS3Policy.json` looks like:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetBucketLocation",
        "s3:ListBucketMultipartUploads"
      ],
      "Resource": "arn:aws:s3:::my-bucket"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:DeleteObject",
        "s3:ListMultipartUploadParts",
```

```
        "s3:AbortMultipartUpload"
      ],
      "Resource": "arn:aws:s3:::my-bucket/*"
    }
  ]
}
```

Create the policy based on the json file we created:

```
$ aws iam create-policy --policy-name DockerHubS3Policy --policy-document file://DockerHubS3Policy.json
```

Now we create an IAM Role. First the policy file `TrustPolicy.json`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

then we run:

```
$ aws iam create-role --role-name DockerHubS3Role --assume-role-policy-document file://TrustPolicy.json
```

Then we attach the Policy to the Role:

```
$ aws iam attach-role-policy --role-name DockerHubS3Role --policy-arn arn:aws:iam::xxxxxxxxxxxx:policy/DockerHubS3
```

Next we create Instance Profile and attach the Role to it:

```
$ aws iam create-instance-profile --instance-profile-name DockerHubS3Profile
$ aws iam add-role-to-instance-profile --role-name DockerHubS3Role --instance-profile-name DockerHubS3Profile
```

and finally we attach the IAM Instance Role to an existing EC2 instance that was originally launched without an IAM role:

```
$ aws ec2 associate-iam-instance-profile --instance-id i-91xxxxxx --iam-instance-profile Name=DockerHubS3Profile
```

Docker Registry Setup

The Registry server will be a Docker container runnig on our Git server.

Docker

Install docker:

```
root@git:~# curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
root@git:~# add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
root@git:~# apt-get update && apt-get install docker-ce -y
```

Registry Server

Create a directory we are going to mount in the registry container:

```
root@git:~# mkdir -p /docker-registry
```

Create username and password for our registry:

```
root@git:~# docker run --rm --entrypoint htpasswd registry:2 -Bbn user "password" > /docker-registry/htpasswd
root@git:~# cat /docker-registry/htpasswd
user:$2y$05$Mr...
```

And also copy the SSL certificate, containing our wildcard and complete CA chain, and the private SSL key over:

```
root@git:~# tree /docker-registry/
/docker-registry/
├── git.mydomain.com.crt
├── git.mydomain.com.key
└── htpasswd

0 directories, 3 files
```

Now we can start our Private Registry server:

```
docker run -d -p 5000:5000 --privileged --restart=always --name s3-registry \
-v /docker-registry:/data:ro \
-e REGISTRY_STORAGE=s3 \
-e REGISTRY_STORAGE_S3_REGION=ap-southeast-2 \
-e REGISTRY_STORAGE_S3_BUCKET=my-bucket \
-e REGISTRY_STORAGE_S3_ENCRYPT=false \
-e REGISTRY_STORAGE_S3_SECURE=true \
-e REGISTRY_STORAGE_S3_V4AUTH=true \
-e REGISTRY_STORAGE_S3_CHUNKSIZE=5242880 \
-e REGISTRY_STORAGE_S3_ROOTDIRECTORY=/image-registry \
-e REGISTRY_HTTP_TLS_CERTIFICATE=/data/git.mydomain.com.crt \
```

```
-e REGISTRY_HTTP_TLS_KEY=/data/git.mydomain.com.key \
-e "REGISTRY_AUTH_HTPASSWD_REALM=Registry Realm" \
-e REGISTRY_AUTH_HTPASSWD_PATH=/data/htpasswd \
registry:2
```

The `--restart=always` will insure the container starts on reboot and stays running. Check the status:

```
root@git:~# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
d5262ecc7096        registry:2          "/entrypoint.sh /e..."   35 minutes ago      Up 35 minutes       0.0.0.0:5000->5000/

root@git:~# docker exec -it d5262ecc7096 registry --version
registry github.com/docker/distribution v2.6.1
```

Lets try to login:

```
root@git:/docker-registry# docker login -u user -p "password" git.mydomain.com:5000
Login Succeeded
```

Lets try and push an image to the new registry. Download, tag and push an image:

```
root@git:~# docker pull alpine:3.4

root@git:~# docker tag alpine:3.4 git.mydomain.com:5000/alpine:3.4
root@git:~# docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
registry                            2                   136c8b16df20        2 weeks ago         33.2 MB
alpine                              3.4                 245f7a86c576        7 weeks ago         4.81 MB
git.mydomain.com:5000/alpine        3.4                 245f7a86c576        7 weeks ago         4.81 MB

root@git:~# docker push git.mydomain.com:5000/alpine:3.4
The push refers to a repository [git.mydomain.com:5000/alpine]
9f8566ee5135: Pushed
3.4: digest: sha256:16b343a6f04a2b2520cb1616081b2257530c942d875938da964a3a7d60f61930 size: 528
```

Lets check the repository now:

```
root@git:~# curl -sSNL -u 'user:password' https://git.mydomain.com:5000/v2/_catalog
{"repositories":["alpine"]}
```

That's it, our private repository S3 storage is working and we can see the Alpine image we just uploaded. Now we can store our images here, example of stunnel image I built on another server:

```
ubuntu@ip-172-31-1-215:~/ansible_docker/stunnel$ sudo docker tag encompass/stunnel:latest git.mydomain.com:5000/stunnel:late

ubuntu@ip-172-31-1-215:~/ansible_docker/stunnel$ sudo docker login git.mydomain.com:5000
Username: user
Password:
```

```
Login Succeeded

ubuntu@ip-172-31-1-215:~/ansible_docker/stunnel$ sudo docker push git.mydomain.com:5000/stunnel:latest
The push refers to a repository [git.mydomain.com:5000/stunnel]
9ce9633b5c62: Pushed
b26e9b2dba3f: Pushed
b0adc271bdea: Pushed
62d4c3d22ff1: Pushed
58d5c6f69f4a: Pushed
0ee67ec08f79: Pushed
72bba1783479: Pushed
737167029b1e: Pushed
95ab35bd2ba1: Pushed
18254e9b079f: Pushed
2bee8e041459: Pushed
483f4516a316: Pushed
266c6f483d86: Pushed
3ca85353d859: Pushed
1771b91fd055: Pushed
c29b5eadf94a: Pushed
latest: digest: sha256:30e799a5a616afecd7bc0405032747aad25bbc12947cc18dd68927a77c2d47e3 size: 3661
ubuntu@ip-172-31-1-215:~/ansible_docker/stunnel$
```

Now we can see the stunnel repository as well:

```
root@git:~# curl -sSNL -u 'user:password' https://git.mydomain.com:5000/v2/_catalog
{"repositories":["alpine","stunnel"]}
```

REST API

For example to see the tags for a repository:

```
root@git:~# curl -sSNL -u 'user:password' https://git.mydomain.com:5000/v2/stunnel/tags/list
{"name":"stunnel","tags":["latest"]}
```

To get image details like size and digest we need to fetch its manifest:

```
oot@git:~# curl -sSNL -u 'user:password' -H "Accept: application/vnd.docker.distribution.manifest.v2+json" -X GET https://git.

  "schemaVersion": 2,
  "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
  "config": {
     "mediaType": "application/vnd.docker.container.image.v1+json",
     "size": 9356,
     "digest": "sha256:763aa8b1b75bb998871c5b434bb6218f4069bfd65cdf2f20ac933c5abadce489"
  },
  "layers": [
     {
...]

oot@git:~# curl -sSNL -u 'user:password' -H "Accept: application/vnd.docker.distribution.manifest.v2+json" -X GET https://git.
```

```
  "schemaVersion": 2,
  "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
  "config": {
     "mediaType": "application/vnd.docker.container.image.v1+json",
     "size": 12031,
     "digest": "sha256:3606a1cf3ba3d74ccadf594d02598db2db1c536ae06f6eb58038eca258710def"
  },
  "layers": [
     {
        "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
        "size": 67088058,
        "digest": "sha256:3122919554d460a6ac9c1113a2a36d88318c94d41ed9bab9178622e9e8f31261"
     },
...]
```


Full docs for the API v2

Notifications

The private registry offers Webhooks functionality via Notifications API. See https://docs.docker.com/registry/notifications/ for details.

Registry UI

In case we need UI for the registry the https://github.com/kwk/docker-registry-frontend docker container is known for working fine. The image can be pulled from https://hub.docker.com/r/konradkleine/docker-registry-frontend/. Since this app can be used as SSL proxy to the registry server we can simply disable the SSL in the registry it self and get it listen to localhost only. Then we start th UI container like:

```
root@git:~# docker run --privileged --restart=always \
 -d \
 -e ENV_DOCKER_REGISTRY_HOST=localhost \
 -e ENV_DOCKER_REGISTRY_PORT=5000 \
 -e ENV_REGISTRY_PROXY_FQDN=git.mydomain.com \
 -e ENV_REGISTRY_PROXY_PORT=443 \
 -e ENV_USE_SSL=yes \
 -v /data/git.mydomain.com.crt:/etc/apache2/server.crt:ro \
 -v /data/git.mydomain.com.crt:/etc/apache2/server.key:ro \
 -p 443:443 \
 konradkleine/docker-registry-frontend:v2
```

and have our registry available via UI at `https://git.mydomain.com`. To enable browse-only mode for our repository we can start with `-e ENV_MODE_BROWSE_ONLY=true` in the run command. The container supports Kerberos authentication only but its running Apache internally so it will not be difficult to clone the repository and modify the setup to install the ldap module for example so we can drop the registry authentication and integrate with a LDAP server.

Tags: docker

Categories: Docker

Updated: May 03, 2017

Previous | Next

LEAVE A COMMENT

ALSO ON **ICICIMOV.GITHUB.IO**

Unified Linux Login, Privileges and ...

4 years ago • 1 comment

On This Page Maintaining users, shared file systems and authentication in ...

HAProxy DDOS protection and API ...

3 years ago • 4 comments

HAProxy is great reverse proxy and load balancer but can also be used for ...

Kubernetes C External Serv

5 years ago • 2 comr

On This Page Pre created Service w but only if we hav