Hi, this is Amirhossein; In the prev tut we learned what is docker and how we can run various containers, today I'll show you how we can run multiple container with a single command called docker-compose, also we'll setup private registry for docker, and run some basic tools like Jenkins, Gitlab … With docker.

Overview of Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a Compose file to configure your application's services. Then, using a single command, you create and start all the services from your configuration.

Docker Compose is great for development, testing, and staging environments, as well as CI workflows.

Using Compose is basically a three-step process.

1. Define your app's environment with a `Dockerfile` so it can be reproduced anywhere.
2. Define the services that make up your app in `docker-compose.yml` so they can be run together in an isolated environment.

3. Lastly, run `docker-compose up` and Compose will start and run your entire app.

Compose has commands for managing the whole lifecycle of your application:

- Start, stop and rebuild services
- View the status of running services
- Stream the log output of running services
- Run a one-off command on a service

Private Docker Registry with SSL

Ok let's dig into it and setup a private docker registry with docker compose. First make a folder for our docker registry:

```
sudo mkdir -p /Docker-Regisry/Data

sudo mkdir /Docker-Registry/nginx

cd /Docker-Regisry/Data
```

Next, create `docker-compose.yml` file:

```
sudo nano docker-compose.yml
```

Add the following contents:

```
registry:
  image: registry:latest
  ports:
    - 5000:5000
  environment:
    REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY: /Data
  volumes:
    - ./Data:/Data
nginx:
  image: nginx:1.9
  ports:
    - 5043:443
  links:
    - registry:registry
  volumes:
    - ./nginx/:/etc/nginx/conf.d:ro
```

Before starting docker-compose, we need to create a new Nginx configuration file.

`sudo nano /Docker-Registry/nginx/registry.conf`

Add the following content:

```
upstream docker-registry {
  server registry:5000;
}


server {
  listen 443;
  server_name myregistrydomain.com;


  # SSL
   ssl on;
   ssl_certificate /etc/nginx/conf.d/domain.crt;
   ssl_certificate_key /etc/nginx/conf.d/domain.key;


  # disable any limits to avoid HTTP 413 for large image uploads
  client_max_body_size 0;


  # required to avoid HTTP 411: see Issue #1486 (https://github.com/docker/docker/issues/1486)
  chunked_transfer_encoding on;


  location /v2/ {
    # 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;

    }


    # To add basic authentication to v2 use auth_basic setting plus add_header

     auth_basic "registry.localhost";

     auth_basic_user_file /etc/nginx/conf.d/registry.password;

     add_header 'Docker-Distribution-Api-Version' 'registry/2.0' always;


    proxy_pass                          http://docker-registry;

    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;

  }

}
```

Next, we need to setup HTTP authentication so we can control Docker registry with user authentication.

We can create authentication file using htpasswd utility. Create a user with the USERNAME as you wish:

```
cd /Docker-Registry/nginx
```

```
sudo htpasswd -c registry.password amirhossein
```

Now our Docker Registry will run with basic HTTP authentication. But the connections are unencrypted and the setup is not secure, so we will need to secure it with SSL.

```
cd /Docker-Registry/nginx
```

Now we need to create our own SSL certificate. Generate a root key using the following command:

```
openssl genrsa -out CA.key 2048
```

Next, generate a root certificate:

```
openssl req -x509 -new -nodes -key CA.key -days 10000 -out CA.crt
```

Then generate a key for your server:

```
openssl genrsa -out domain.key 2048
```

Now, we need to make a certificate signing request:

```
openssl req -new -key domain.key -out CA.csr
```

Fill in all information as you wish:

```
Country Name (2 letter code):IR

State or Province Name (full name):amirhossein

Locality Name (eg, city):amirhossein

Organization Name (eg, company):amirhossein

Organizational Unit Name (eg, section):amirhossein

Common Name (e.g. server FQDN or YOUR name):tools

Email Address []:he.amirhossein@gmail.com


Please enter the following 'extra' attributes

to be sent with your certificate request

A challenge password []:

An optional company name []:amirhossein
```

Next, we need to sign the certificate request:

```
openssl x509 -req -in CA.csr -CA CA.crt -CAkey CA.key -CAcreateserial -out domain.crt
-days 10000
```

The certificates we have just created are not verified by any known certificate authority. So we need to copy CA.crt on the host machine so that we can use Docker from the Docker registry server itself:

```
sudo cp CA.crt /usr/local/share/ca-certificates/
```

```
sudo update-ca-certificates
```

```
Updating certificates in /etc/ssl/certs... 1 added, 0 removed; done.
```

```
Running hooks in /etc/ca-certificates/update.d....done.
```

Now, restart Docker service to picks up the changes:

`sudo service docker restart`

Before we run our private docker registry we can copy our certification to any client machine so they can push or pull their images to our registry.

In the client side:

`sudo cp CA.crt /usr/local/share/ca-certificates/`

`sudo update-ca-certificates`

```
Updating certificates in /etc/ssl/certs... 1 added, 0 removed; done.

Running hooks in /etc/ca-certificates/update.d....done.
```

`sudo service docker restart`

Now it’s time to run our registry with docker-compose:

`cd /Docker-Registry`

`sudo docker-compose up`

```
registry_1  | time="2016-10-9T16:51:03Z" level=info msg="response completed" go.versi
on=go1.6.2 http.request.host="localhost:5000" http.request.id=d9318b45-b721-4053-b279
-621475b78173 http.request.method=GET http.request.remoteaddr="172.17.0.2:39569" http
.request.uri="/v2/" http.request.useragent="curl/7.35.0" http.response.contenttype="a
pplication/json; charset=utf-8" http.response.duration=2.936977ms http.response.statu
s=200 http.response.written=2 instance.id=b12d5a57-25a5-41a0-abfb-4e290ea8c767 versio
n=v2.4.1

registry_1  | 172.17.0.2 - - [9/Oct/2016:16:51:03 +0000] "GET /v2/ HTTP/1.1" 200 2 ""
"curl/7.35.0"

registry_1  | time="2016-05-31T16:51:24Z" level=info msg="response completed" go.vers
ion=go1.6.2 http.request.host="localhost:5043" http.request.id=e6ac055b-38f4-49d7-9eb
c-45c8ca1062b0 http.request.method=GET http.request.remoteaddr=172.17.0.2 http.reques
t.uri="/v2/" http.request.useragent="curl/7.35.0" http.response.contenttype="applicat
ion/json; charset=utf-8" http.response.duration=3.792746ms http.response.status=200 h
ttp.response.written=2 instance.id=b12d5a57-25a5-41a0-abfb-4e290ea8c767 version=v2.4.
1

nginx_1     | 172.17.0.2 - - [9/Oct/2016:16:51:24 +0000] "GET /v2/ HTTP/1.1" 200 2 "-
" "curl/7.35.0" "-"

registry_1  | 172.17.0.4 - - [9/Oct/2016:16:51:24 +0000] "GET /v2/ HTTP/1.0" 200 2 ""
"curl/7.35.0"
```

So far so good, let’s login into our registry:

```
sudo docker login https://tools:5043

Username: amirhossein

Password: amirhossein

Login Succeeded
```

Note: “tools” is the host name of our 192.168.251.80 server.

Ok let’s publish an Image to our private Docker registry. I already downloaded all the Docker images from official Docker Hub, as an example I’ll use Ubuntu image and push it to our registry.

```
sudo docker tag ubuntu tools:5043/amirhossein_test_ubuntu
```

```
sudo docker push tools:5043/amirhossein_test_ubuntu
```

Also we can pull it from our registry on different client:

```
sudo docker pull tools:5043/amirhossein_test_ubuntu
```

Run Jenkins with SonarQube Container

Here is the fun part, now it’s time to use our registry and pull some images that we can run them.

Note: if you don’t have Jenkins and SonarQube images, you can pull them this way:

```
sudo docker pull tools:5043/jenkinsci
```

```
sudo docker pull tools:5043/sonarqube
```

```
sudo docker pull tools:5043/postgres
```

Let’s run them:

sudo docker run -p 8080:8080 -p 50000:50000 -v /home/Jenkins:/var/jenkins_home tools:5043/jenkinsci

Note: if you unfamiliar with above command read my prev tut on wiki.

Also I’ll run SonarQube and Postgres with docker-compose.

`sudo docker-compose up`

`docker-compose.yml`:

```
postgresql:
  image: tools:5043/postgres
  environment:
    - POSTGRESQL_USER=sonar
    - POSTGRESQL_PASS=sonar
    - POSTGRESQL_DB=sonar
  volumes:
    - /opt/db/sonarqube/:/var/lib/postgresql
  ports:
    - "5432:5432"
sonarqube:
  image: tools:5043/sonarqube
  links:
    - postgresql:db
  environment:
    - DB_USER=sonar
    - DB_PASS=sonar
    - DB_NAME=sonar
  ports:
    - "9000:9000"
    - "443"
```

Run Nexus and ActiveMQ Container

I'll run Neux without using docker-compose. Because it's single counter.

```
sudo mkdir /some/dir/nexus-data && chown -R 200 /some/dir/nexus-data
```

```
sudo docker run -d -p 8081:8081 --name nexus -v /root/nexus_home:/sonatype-work tools:5043/nexus3
```

```
sudo docker run --name='activemq' -d -e 'ACTIVEMQ_NAME=amqp-srv1' -e 'ACTIVEMQ_REMOVE_DEFAULT_ACCOUNT=true' -e 'ACTIVEMQ_ADMIN_LOGIN=admin' -e 'ACTIVEMQ_ADMIN_PASSWORD=amirhossein' -e 'ACTIVEMQ_WRITE_LOGIN=producer_login' -e 'ACTIVEMQ_WRITE_PASSWORD=producer_password' -e 'ACTIVEMQ_READ_LOGIN=consumer_login' -e 'ACTIVEMQ_READ_PASSWORD=consumer_password' -e 'ACTIVEMQ_JMX_LOGIN=jmx_login' -e 'ACTIVEMQ_JMX_PASSWORD=jmx_password' -e 'ACTIVEMQ_STATIC_TOPICS=topic1;topic2;topic3' -e 'ACTIVEMQ_STATIC_QUEUES=queue1;queue2;queue3' -e 'ACTIVEMQ_MIN_MEMORY=1024' -e  'ACTIVEMQ_MAX_MEMORY=4096' -e 'ACTIVEMQ_ENABLED_SCHEDULER=true' -v /root/activemq_home:/data/activemq -v /root/activemq_home/logs/log:/var/log/activemq -p 8161:8161 -p 61616:61616 -p 61613:61613 tools:5043/activemq
```

Run Gitlab, Postgres, Redis Container

```
sudo docker-compose up
```

`docker-compose.yml`:

```
services:

  redis:

    restart: always

    image: tools:5043/redis:latest

    command:

    - --loglevel warning

    volumes:

    - /home/gitlab/redis:/var/lib/redis:Z


  postgresql:
    restart: always
    image: tools:5043/postgresql:9.5-1
    volumes:
    - /home/gitlab/postgresql:/var/lib/postgresql:Z
    environment:
    - DB_USER=gitlab
    - DB_PASS=password
    - DB_NAME=gitlabhq_production
    - DB_EXTENSION=pg_trgm

  gitlab:
    restart: always
    image: tools:5043/gitlab:8.12.3
    depends_on:
    - redis
    - postgresql
    ports:
    - "10080:80"
    - "10022:22"
    volumes:
    - /home/gitlab/gitlab:/home/git/data:Z
    environment:
    - DEBUG=false

    - DB_ADAPTER=postgresql
    - DB_HOST=postgresql
    - DB_PORT=5432
    - DB_USER=gitlab
    - DB_PASS=password
    - DB_NAME=gitlabhq_production

    - REDIS_HOST=redis
    - REDIS_PORT=6379


    - TZ=Asia/Kolkata
    - GITLAB_TIMEZONE=Kolkata


    - GITLAB_HTTPS=false
    - SSL_SELF_SIGNED=false


    - GITLAB_HOST=localhost
    - GITLAB_PORT=10080
    - GITLAB_SSH_PORT=10022
    - GITLAB_RELATIVE_URL_ROOT=
    - GITLAB_SECRETS_DB_KEY_BASE=long-and-random-alphanumeric-string
    - GITLAB_SECRETS_SECRET_KEY_BASE=long-and-random-alphanumeric-string
    - GITLAB_SECRETS_OTP_KEY_BASE=long-and-random-alphanumeric-string


    - GITLAB_ROOT_PASSWORD=amirhossein
    - GITLAB_ROOT_EMAIL=amirhossein


    - GITLAB_NOTIFY_ON_BROKEN_BUILDS=true
    - GITLAB_NOTIFY_PUSHER=false


    - GITLAB_EMAIL=notifications@example.com
    - GITLAB_EMAIL_REPLY_TO=noreply@example.com
    - GITLAB_INCOMING_EMAIL_ADDRESS=reply@example.com


    - GITLAB_BACKUP_SCHEDULE=daily
    - GITLAB_BACKUP_TIME=01:00


    - SMTP_ENABLED=false
    - SMTP_DOMAIN=www.example.com
    - SMTP_HOST=smtp.gmail.com
```

```
      - SMTP_PORT=587
      - SMTP_USER=mailer@example.com
      - SMTP_PASS=password
      - SMTP_STARTTLS=true
      - SMTP_AUTHENTICATION=login

      - IMAP_ENABLED=false
      - IMAP_HOST=imap.gmail.com
      - IMAP_PORT=993
      - IMAP_USER=mailer@example.com
      - IMAP_PASS=password
      - IMAP_SSL=true
      - IMAP_STARTTLS=false
      - OAUTH_ENABLED=false
      - OAUTH_AUTO_SIGN_IN_WITH_PROVIDER=
      - OAUTH_ALLOW_SSO=
      - OAUTH_BLOCK_AUTO_CREATED_USERS=true
      - OAUTH_AUTO_LINK_LDAP_USER=false
      - OAUTH_AUTO_LINK_SAML_USER=false
      - OAUTH_EXTERNAL_PROVIDERS=

      - OAUTH_CAS3_LABEL=cas3
      - OAUTH_CAS3_SERVER=
      - OAUTH_CAS3_DISABLE_SSL_VERIFICATION=false
      - OAUTH_CAS3_LOGIN_URL=/cas/login
      - OAUTH_CAS3_VALIDATE_URL=/cas/p3/serviceValidate
      - OAUTH_CAS3_LOGOUT_URL=/cas/logout

      - OAUTH_GOOGLE_API_KEY=
      - OAUTH_GOOGLE_APP_SECRET=
      - OAUTH_GOOGLE_RESTRICT_DOMAIN=
      - OAUTH_FACEBOOK_API_KEY=

      - OAUTH_FACEBOOK_APP_SECRET=


      - OAUTH_TWITTER_API_KEY=

      - OAUTH_TWITTER_APP_SECRET=

      - OAUTH_GITHUB_API_KEY=

      - OAUTH_GITHUB_APP_SECRET=

      - OAUTH_GITHUB_URL=

      - OAUTH_GITHUB_VERIFY_SSL=

      - OAUTH_GITLAB_API_KEY=

      - OAUTH_GITLAB_APP_SECRET=


      - OAUTH_BITBUCKET_API_KEY=

      - OAUTH_BITBUCKET_APP_SECRET=


      - OAUTH_SAML_ASSERTION_CONSUMER_SERVICE_URL=

      - OAUTH_SAML_IDP_CERT_FINGERPRINT=

      - OAUTH_SAML_IDP_SSO_TARGET_URL=

      - OAUTH_SAML_ISSUER=

      - OAUTH_SAML_LABEL="Our SAML Provider"

      - OAUTH_SAML_NAME_IDENTIFIER_FORMAT=urn:oasis:names:tc:SAML:2.0:nameid-format:tra
  nsient

      - OAUTH_SAML_GROUPS_ATTRIBUTE=

      - OAUTH_SAML_EXTERNAL_GROUPS=

      - OAUTH_SAML_ATTRIBUTE_STATEMENTS_EMAIL=

      - OAUTH_SAML_ATTRIBUTE_STATEMENTS_NAME=

      - OAUTH_SAML_ATTRIBUTE_STATEMENTS_FIRST_NAME=

      - OAUTH_SAML_ATTRIBUTE_STATEMENTS_LAST_NAME=

      - OAUTH_CROWD_SERVER_URL=

      - OAUTH_CROWD_APP_NAME=

      - OAUTH_CROWD_APP_PASSWORD=

      - OAUTH_AUTH0_CLIENT_ID=
      - OAUTH_AUTH0_CLIENT_SECRET=
      - OAUTH_AUTH0_DOMAIN=
      - OAUTH_AZURE_API_KEY=
      - OAUTH_AZURE_API_SECRET=
      - OAUTH_AZURE_TENANT_ID=
```

Run Prometheus, CAdvisor, Grafana Container

`sudo docker-compose up`

`docker-compose.yml`:

```
version: '2'

networks:
  front-tier:
    driver: bridge
  back-tier:
    driver: bridge

services:
  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    volumes:
      - ./prometheus/:/etc/prometheus/
      - /root/prometheus_data:/prometheus
    command:
      - '-config.file=/etc/prometheus/prometheus.yml'
#       - '-storage.local.path=/prometheus'
      - '-alertmanager.url=http://alertmanager:9093'
    expose:
      - 9090
    ports:
      - 9090:9090
    links:
      - cadvisor:cadvisor
      - alertmanager:alertmanager
    depends_on:
      - cadvisor
    networks:
      - back-tier

  node-exporter:
    image: prom/node-exporter:latest
    expose:
      - 9100
    networks:
      - back-tier
  alertmanager:
    image: prom/alertmanager:latest
    ports:
      - 9093:9093
    volumes:
      - ./alertmanager/:/etc/alertmanager/
    networks:
      - back-tier
    command:
      - '-config.file=/etc/alertmanager/config.yml'
      - '-storage.path=/alertmanager'

  cadvisor:
    image: google/cadvisor:latest
    volumes:
      - /:/rootfs:ro
      - /var/run:/var/run:rw
      - /sys:/sys:ro
      - /var/lib/docker/:/var/lib/docker:ro
    expose:
      - 8080
    ports:
      - 28080:8080
    networks:
      - back-tier

  grafana:
    image: grafana/grafana:latest
    depends_on:
      - prometheus
    ports:
      - 3000:3000
    volumes:
      - /root/grafana_data:/var/lib/grafana
    env_file:
      - config.monitoring
    networks:
      - back-tier
      - front-tier


volumes:
    prometheus_data: {}
    grafana_data: {}
```