Docker:latest. Подводные камни использования тега latest

Введение

Так как последние два года я работаю как DevOps инженер, то по ходу долга приходится часто сталкиваться с Docker. Уверен, что в 2017 года каждый администратор и/или разработчик по крайней мере слышал о Docker, а многие им пользуются в повседневной жизни. Очень интересный продукт, конечно не без недостатков, а у кого их нет…

Docker - это отличная возможность переложить проблемы поддержания актуальных версий системных библиотек (того же самого openssl) с поставщиков ОС и системных администраторов на никого. После того, как проблемы безопасности окружения переложены на никого, скорость внедрения и эксплуатации заметно возрастает, ведь производительность никого ограничена только супремумом пофигизма, а он, в свою очередь, верхней границы не имеет. (c)

На одном из проектов мы используем AWS Elastic Beanstalk. Очень интересный и в тоже время очень «проблематичный» сервис от Amazon. Итак, у нас используется Java приложение, написанное с использование Spring Boot и запускаемое внутри Docker контейнера. Масштабировалось данное приложение с помощью auto scaling.

И вот при очередном масштабировании на одной из нод у нас начали появляться ошибки, которых не было на второй ноде. Как позже выяснилось винной всему был тег latest, который использовался при указании docker образа. Итак давай те рассмотрим более детально почему не стоит использовать latest тег.

Основы Docker

Итак, тег можно задать двумя способами:

1. при сборке образа при помощи docker build, через параметр -t
2. с помощью команды docker tag

Если вы явно не указываете тег при сборке образа, то по умолчанию образу будет присвоен специальный тег - latest. У данного тега нет какого то специального назначения или скрытого смысла, но о нем стоит знать и помнить, когда вы строите CI/CD на своем проекте. Это легко проверить, достаточно создать простейший образ на базе alpine. Так как по работе часто приходится работать с python, то в качестве основы возьмем один из наиболее популярных фреймворков - Flask. Создаем простое приложение на Flask

```
# test.py
from os import path
from flask import Flask
from flask import send_from_directory
```

```
app = Flask(__name__)


@app.route('/', methods=['GET'])
def hello_world():
    return 'Flask sample app. Version: 1\n'


@app.route('/favicon.ico', methods=['GET'])
def favicon():
    return send_from_directory(path.join(app.root_path, 'static'),
                               'favicon.ico',
mimetype='image/vnd.microsoft.icon')

if __name__ == '__main__':
    app.run(host="0.0.0.0", port=8080)
```

После этого создаем Dockerfile со следующим содержимым.

```
FROM alpine

MAINTAINER ALex_hha <alex.hha[at]gmail.com>

RUN apk update && apk add python2 py2-pip \
    && pip install --upgrade pip \
    && pip install Flask==0.11.1 \
    && rm -rf /tmp/* /var/tmp/* /var/lib/apk/* /var/cache/apk/*

WORKDIR /opt/
COPY test.py /opt/
ADD
https://github.com/pallets/flask/raw/master/docs/_static/flask-favicon.ico \
    /opt/static/favicon.ico

EXPOSE 8080

ENTRYPOINT ["python"]
CMD ["test.py"]
```

Собираем наш образ

```
# docker build -t flask .
Sending build context to Docker daemon 3.072 kB
Step 1/9 : FROM alpine
 ---> 4a415e366388
Step 2/9 : MAINTAINER ALex_hha <alex.hha[at]gmail.com>
 ---> Running in 3431022faea8
```

```
 ---> 77b448808aa8
Removing intermediate container 3431022faea8
Step 3/9 : RUN apk update && apk add python2 py2-pip     && pip install --
upgrade pip     && pip install Flask==0.11.1     && rm -rf /tmp/* /var/tmp/*
/var/lib/apk/* /var/cache/apk/*
 ---> Running in ad5a9c63bbc8
fetch http://dl-cdn.alpinelinux.org/alpine/v3.5/main/x86_64/APKINDEX.tar.gz
...
...
...
Step 7/9 : EXPOSE 8080
 ---> Running in 008e83f917bf
 ---> 198e1e7b10b3
Removing intermediate container 008e83f917bf
Step 8/9 : ENTRYPOINT python
 ---> Running in 8b1dd1e3b936
 ---> 036815cf8fe5
Removing intermediate container 8b1dd1e3b936
Step 9/9 : CMD test.py
 ---> Running in 15033190c0aa
 ---> 4f3b81192bda
Removing intermediate container 15033190c0aa
Successfully built 4f3b81192bda
```

Выводим информацию по образам

```
# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
flask               latest              4f3b81192bda        About a minute
ago   58.5 MB
alpine              latest              4a415e366388        6 weeks ago
3.98 MB
```

Как мы видим, даже не смотря на то, что мы не указывали явно тег при сборке, у нашего образа появился тег - latest. Запустим наш образ

```
# docker run -p 8080:8080 -it --rm flask
 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)

# curl -i http://localhost:8080/
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 28
Server: Werkzeug/0.12.1 Python/2.7.13
Date: Sun, 16 Apr 2017 13:24:25 GMT

Flask sample app. Version: 1
```

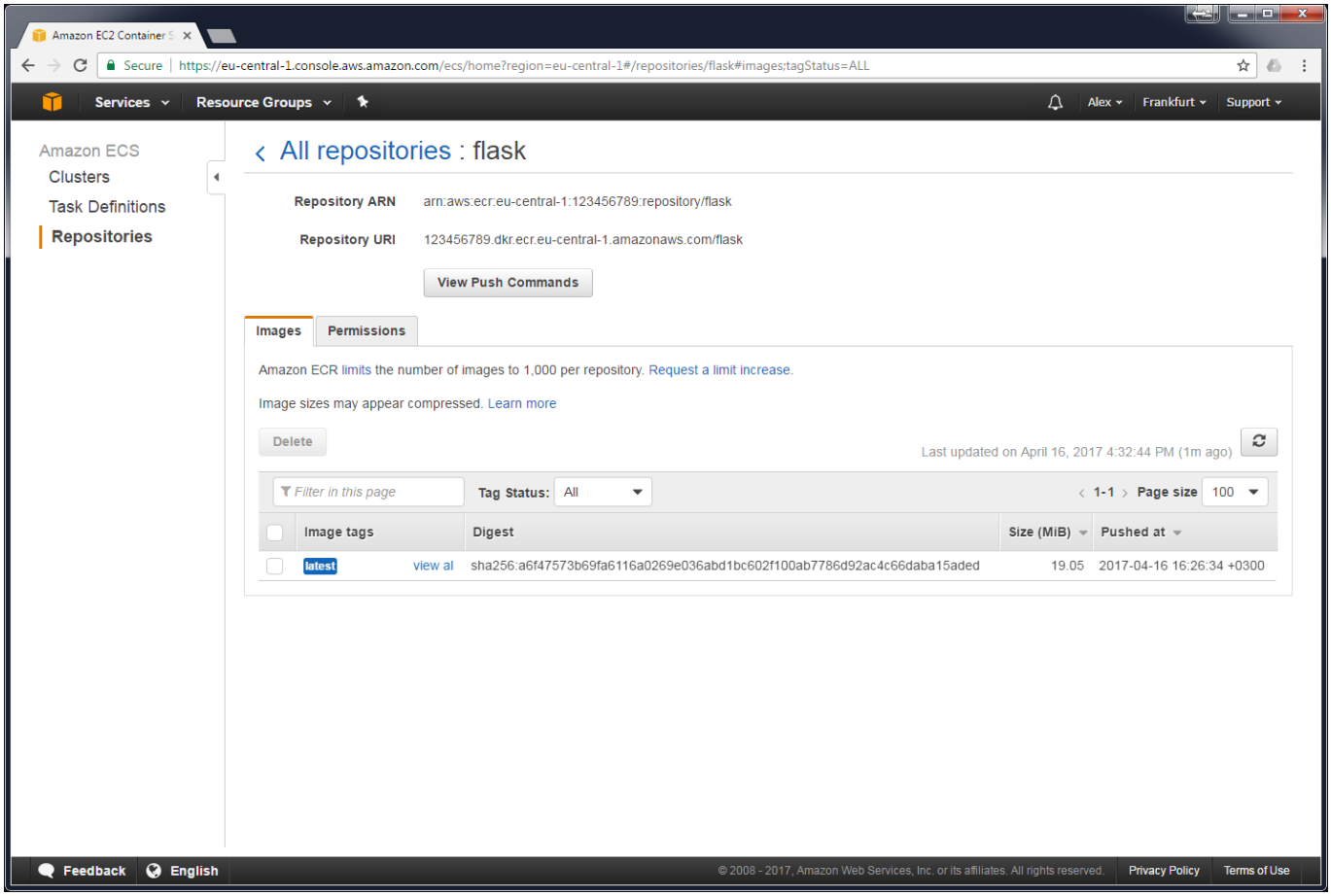
Сохраняем наш образ в репозитарий. В данной статье я буду использовать сервис от Амазона - ECR (EC2 Container Registry), так как к сожалению DockerHub не предоставлял sha256 для

каждого образа, по крайней мере на момент написания статьи.

```
# aws ecr get-login --region eu-central-1 --profile personal
docker login -u AWS -p eyJwYXl...S0VZIn0= -e none
https://123456789.dkr.ecr.eu-central-1.amazonaws.com

# docker login -u AWS -p eyJwYXl...S0VZIn0= -e none
https://123456789.dkr.ecr.eu-central-1.amazonaws.com
Login Succeeded

# docker tag flask:latest 123456789.dkr.ecr.eu-
central-1.amazonaws.com/flask:latest

# docker push 123456789.dkr.ecr.eu-central-1.amazonaws.com/flask:latest
The push refers to a repository [123456789.dkr.ecr.eu-
central-1.amazonaws.com/flask]
659b7a5a3573: Pushed
ec4d561016a8: Pushed
80cc614be280: Pushed
1c1f633d1481: Pushed
23b9c7b43573: Pushed
latest: digest:
sha256:a6f47573b69fa6116a0269e036abd1bc602f100ab7786d92ac4c66daba15aded
size: 1361
```

Отлично, теперь мы можем использовать данный образ в своих проектах. Как я уже говорил в начале статьи - мы используем elastic beanstalk, внутри которого поднимаем докер образы. Более детальное описание beanstalk я приведу немного ниже.

А вот так будет выглядеть наш репозитарий на AWS



Настройка Elastic Beanstalk

Для описания того, какой образ запускать и с какими параметрами на AWS используется специальный файл - Dockerrun.aws.json, ниже привожу простейший пример такого файла.

```
{
    "AWSEBDockerrunVersion": 2,

    "containerDefinitions": [{
        "memory": 256,
        "cpu": 1,
        "name": "flask",
        "image": "123456789.dkr.ecr.eu-
central-1.amazonaws.com/flask:latest",

        "portMappings": [{
            "hostPort": 80,
            "containerPort": 8080
        }]
    }]
}
```

Для создания тестового окружения удобно использовать утилиту командной строки eb.

Инициализируем наше приложение

```
# eb init LATEST-TAG --profile personal --region eu-central-1

It appears you are using Multi-container Docker. Is this correct?
(y/n): y
Do you want to set up SSH for your instances?
(y/n): n
```

И создаем окружение

```
# eb create flask --cname flask --elb-type application -i t2.micro -ip aws-
elasticbeanstalk-ec2-role
Creating application version archive "app-170416_183710".
Uploading LATEST-TAG/app-170416_183710.zip to S3. This may take a while.
Upload Complete.
Environment details for: flask
  Application name: LATEST-TAG
  Region: eu-central-1
  Deployed Version: app-170416_183710
  Environment ID: e-mk5hqqrj29
  Platform: 64bit Amazon Linux 2016.09 v2.5.2 running Multi-container Docker
1.12.6 (Generic)
  Tier: WebServer-Standard
  CNAME: flask.eu-central-1.elasticbeanstalk.com
  Updated: 2017-04-16 15:37:18.121000+00:00
Printing Status:
INFO: createEnvironment is starting.
...
...
...
INFO: Starting new ECS task with awseb-flask-mk5hqqrj29.
INFO: ECS task: arn:aws:ecs:eu-central-1:123456789:task/4703d9e7-
d6c0-4f31-8409-2318d72fd1a3 is RUNNING.
INFO: Environment health has transitioned from Pending to Ok. Initialization
completed 2 seconds ago and took 3 minutes.
INFO: Successfully launched environment: flask
```

Проверяем статус нашего окружения

```
# eb status -v
Environment details for: flask
  Application name: LATEST-TAG
  Region: eu-central-1
  Deployed Version: app-170416_183710
  Environment ID: e-mk5hqqrj29
  Platform: 64bit Amazon Linux 2016.09 v2.5.2 running Multi-container Docker
1.12.6 (Generic)
  Tier: WebServer-Standard
  CNAME: flask.eu-central-1.elasticbeanstalk.com
  Updated: 2017-04-16 15:41:40.795000+00:00
```

```
  Status: Ready
  Health: Green
  Running instances: 1
  Running processes: 1
      default:
           i-083a515b6a9ca71bd: healthy
```

Делаем тестовый запрос

```
# curl -i http://flask.eu-central-1.elasticbeanstalk.com/
HTTP/1.1 200 OK
Date: Sun, 16 Apr 2017 15:45:33 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 28
Connection: keep-alive
Server: Werkzeug/0.12.1 Python/2.7.13

Flask sample app. Version: 1
```

Тестирование

Допустим, через время наш код обновился и нам соответственно надо обновить docker образ. Аналогично пересобираем наш docker образ. Для упрощения примеров, я лишь изменил в файле test.py вывод с «Flask sample app. Version: 1» на «Flask sample app. Version: 2»

```
# docker build -t flask .
Sending build context to Docker daemon 3.072 kB
Step 1/9 : FROM alpine
 ---> 4a415e366388
Step 2/9 : MAINTAINER ALex_hha <alex.hha[at]gmail.com>
 ---> Using cache
 ---> 77b448808aa8
Step 3/9 : RUN apk update && apk add python2 py2-pip     && pip install --
upgrade pip     && pip install Flask==0.11.1     && rm -rf /tmp/* /var/tmp/*
/var/lib/apk/* /var/cache/apk/*
 ---> Using cache
 ---> 1eacdec8f249
Step 4/9 : WORKDIR /opt/
 ---> Using cache
 ---> 0b5c05338c7c
Step 5/9 : COPY test.py /opt/
 ---> 6ba65b754a2d
...
...
...
Step 7/9 : EXPOSE 8080
 ---> Running in d1b99824aa91
 ---> c1e34eeb43f4
Removing intermediate container d1b99824aa91
```

```
Step 8/9 : ENTRYPOINT python
 ---> Running in 9b0fe310616f
 ---> 0a7100cb34be
Removing intermediate container 9b0fe310616f
Step 9/9 : CMD test.py
 ---> Running in 33bb773c4c11
 ---> 1bc8aff742c5
Removing intermediate container 33bb773c4c11
Successfully built 1bc8aff742c5
```

И делаем push

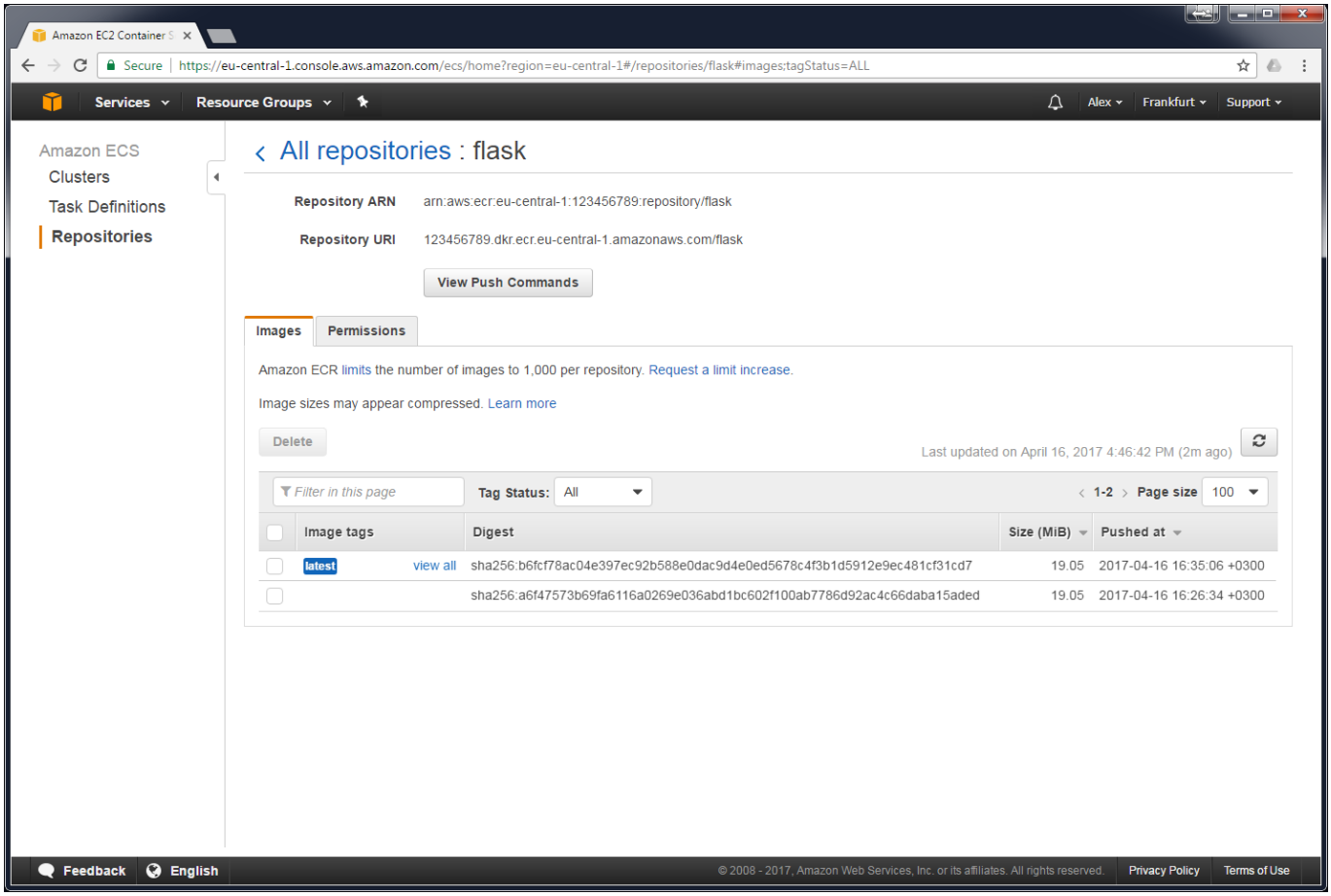
```
# docker push 123456789.dkr.ecr.eu-central-1.amazonaws.com/flask:latest
The push refers to a repository [123456789.dkr.ecr.eu-
central-1.amazonaws.com/flask]
5c2fb49623e3: Pushed
e68c063c9469: Pushed
80cc614be280: Layer already exists
1c1f633d1481: Layer already exists
23b9c7b43573: Layer already exists
latest: digest:
sha256:b6fcf78ac04e397ec92b588e0dac9d4e0ed5678c4f3b1d5912e9ec481cf31cd7
size: 1361
```

При этом наш локальный репозитарий будет выглядеть следующим образом

```
# docker images
REPOSITORY                                                  TAG
IMAGE ID            CREATED             SIZE
123456789.dkr.ecr.eu-central-1.amazonaws.com/flask      latest
1bc8aff742c5        6 minutes ago       58.5 MB
flask                                                       latest
1bc8aff742c5        6 minutes ago       58.5 MB
123456789.dkr.ecr.eu-central-1.amazonaws.com/flask      <none>
4f3b81192bda        15 minutes ago      58.5 MB
alpine                                                      latest
4a415e366388        6 weeks ago         3.98 MB
```

Обратите внимание, что у нашего первоначального образа, образ с ID 4f3b81192bda, теперь вообще нет тега! Аналогичная ситуация будет и в репозитарии AWS.

А вот так будет выглядеть наш репозитарий на AWS



И вот теперь наступает самое интересное. У нас есть работающее окружение, в котором у нас запущен образ с первой версией, но в качестве тега у нас использовался latest. А теперь давайте посмотрим, что произойдет при попытке расширения нашего окружения до 2х нод.

```
# aws elasticbeanstalk update-environment --profile personal --region eu-
central-1 --environment-name flask --option-settings
Namespace=aws:autoscaling:asg,OptionName=MinSize,Value=2
Namespace=aws:autoscaling:asg,OptionName=MaxSize,Value=2
{
    "ApplicationName": "LATEST-TAG",
    "EnvironmentName": "flask",
    "VersionLabel": "app-170416_183710",
    "Status": "Updating",
    "Description": "Environment created from the EB CLI using \"eb
create\"",
    "EnvironmentId": "e-mk5hqqrj29",
    "EndpointURL": "awseb-AWSEB-4B67PVM0HGQP-996405974.eu-
central-1.elb.amazonaws.com",
    "SolutionStackName": "64bit Amazon Linux 2016.09 v2.5.2 running Multi-
container Docker 1.12.6 (Generic)",
    "CNAME": "flask.eu-central-1.elasticbeanstalk.com",
    "Health": "Grey",
    "AbortableOperationInProgress": true,
    "Tier": {
        "Version": " ",
        "Type": "Standard",
        "Name": "WebServer"
```

```
    },
    "DateUpdated": "2017-04-16T15:58:25.149Z",
    "DateCreated": "2017-04-16T15:37:18.074Z"
}
```

Проверяем статус окружения

```
# eb status -v
Environment details for: flask
  Application name: LATEST-TAG
  Region: eu-central-1
  Deployed Version: app-170416_183710
  Environment ID: e-mk5hqqrj29
  Platform: 64bit Amazon Linux 2016.09 v2.5.2 running Multi-container Docker
1.12.6 (Generic)
  Tier: WebServer-Standard
  CNAME: flask.eu-central-1.elasticbeanstalk.com
  Updated: 2017-04-16 16:00:42.474000+00:00
  Status: Ready
  Health: Green
  Running instances: 2
  Running processes: 1
      default:
           i-037a63403f7cf4eff: healthy
           i-083a515b6a9ca71bd: healthy
```

Делаем несколько запросов к нашему приложению

```
# for i in {1..100}; do curl -s
http://flask.eu-central-1.elasticbeanstalk.com/; done | sort | uniq -c
     50 Flask sample app. Version: 1
     50 Flask sample app. Version: 2
```

И что же мы видим, 50 запросов вернуло версию 1 и 50 версию 2. А это совсем не то, что мы ожидали! А виной всему именно тег latest. Во время создания окружения у нас тег latest указывал на образ:

- **sha256:a6f47573b69fa6116a0269e036abd1bc602f100ab7786d92ac4c66daba15aded**

Это образ который выводит строку «Flask sample app. Version: 1». Затем мы создали новый образ и загрузили его в репозитарий AWS. И после этого тег latest стал указывать уже на совсем другой образ, а именно:

- **sha256:b6fcf78ac04e397ec92b588e0dac9d4e0ed5678c4f3b1d5912e9ec481cf31cd7**

Именно поэтому, после расширения окружения на вторую ноду был установлен этот образ, который выводит строку «Flask sample app. Version: 2». Точно такая же ситуация у нас произошла и на боевых серверах. Когда часть нод у нас была запущена с одной версией кода, а вторая часть нод, после скейлинга оказалась запущенной с другой версией кода.

Тегирование

Чтобы избежать подобных проблем, мы добавили в наш CI(Jenkins) генерирование уникального тега для каждого образа с помощью простого bash скрипта

```
#!/bin/bash

# Generate image tag

CURR_DATE=$(date '+%Y%m%d')
CURR_TIME=$(date '+%H%M')

GIT_FULL_HASH=$(git rev-parse HEAD)

echo IMAGE_TAG="v.${GIT_FULL_HASH:0:8}_${CURR_DATE}${CURR_TIME}-
${BUILD_NUMBER}" > IMAGE.TAG
```

И затем при сборке образа мы указываем этот тег

```
#!/bin/bash

set -e
set -u
set -x

IMAGE_NAME="123456789.dkr.ecr.eu-central-1.amazonaws.com/flask"

$(aws ecr get-login --region eu-central-1)

docker build -t ${IMAGE_NAME}:${IMAGE_TAG} .
docker push ${IMAGE_NAME}:${IMAGE_TAG}
docker rmi ${IMAGE_NAME}:${IMAGE_TAG}
```

Это решение так сказать на будущее, а как же быть с текущим положением дел. На момент написания статьи Dockerrun.aws.json не поддерживал возможность указывать sha256 в имени образа, можно использовать только теги. Но к счастью docker pull поддерживает данную возможность, а ECR показывает sha256 для каждого образа, в отличие от того же DockerHub.

Итак, скачиваем образ и тегируем его на основе Image ID

```
# docker pull 123456789.dkr.ecr.eu-
central-1.amazonaws.com/flask@sha256:a6f47573b69fa6116a0269e036abd1bc602f100
ab7786d92ac4c66daba15aded
sha256:a6f47573b69fa6116a0269e036abd1bc602f100ab7786d92ac4c66daba15aded:
Pulling from flask
ec37562cf8fa: Pull complete
```

```
9dca7a6ccc9a: Pull complete
578da882ab39: Pull complete
f797509bcf42: Pull complete
460fdfc0d3a1: Pull complete
Digest:
sha256:a6f47573b69fa6116a0269e036abd1bc602f100ab7786d92ac4c66daba15aded
Status: Downloaded newer image for 123456789.dkr.ecr.eu-
central-1.amazonaws.com/flask@sha256:a6f47573b69fa6116a0269e036abd1bc602f100
ab7786d92ac4c66daba15aded

# docker images
REPOSITORY                                                  TAG
IMAGE ID            CREATED             SIZE
123456789.dkr.ecr.eu-central-1.amazonaws.com/flask     <none>
4f3b81192bda        4 hours ago         58.5 MB
```

Так как у образа нет тега, то привязываемся к Image ID - 4f3b81192bda

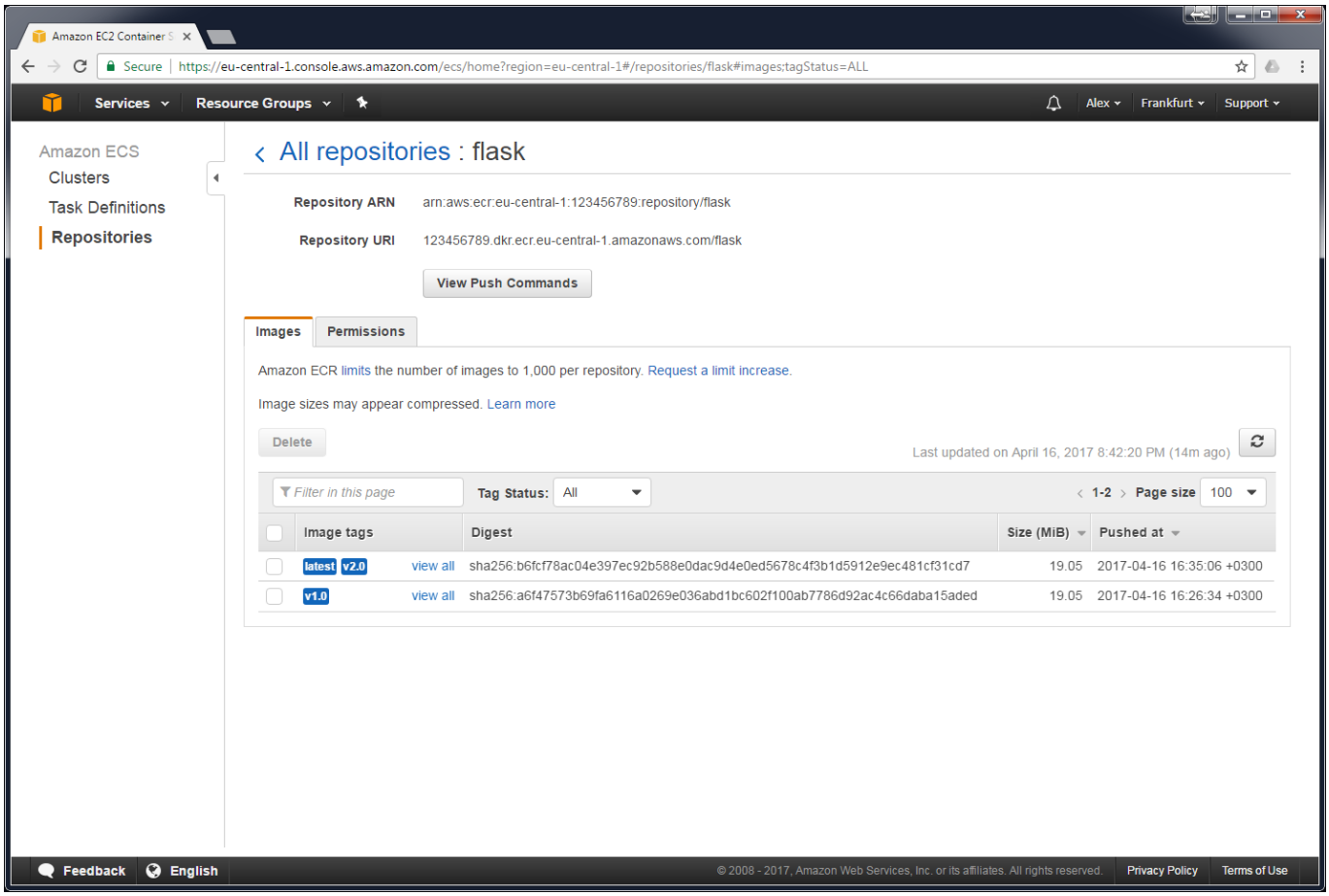
```
# docker tag 4f3b81192bda flask:v1.0

# docker tag flask:v1.0 123456789.dkr.ecr.eu-
central-1.amazonaws.com/flask:v1.0

# docker push 123456789.dkr.ecr.eu-central-1.amazonaws.com/flask:v1.0
The push refers to a repository [123456789.dkr.ecr.eu-
central-1.amazonaws.com/flask]
659b7a5a3573: Layer already exists
ec4d561016a8: Layer already exists
80cc614be280: Layer already exists
1c1f633d1481: Layer already exists
23b9c7b43573: Layer already exists
v1.0: digest:
sha256:a6f47573b69fa6116a0269e036abd1bc602f100ab7786d92ac4c66daba15aded
size: 1361
```

Аналогично делаем и для текущего latest тега

```
# docker pull 123456789.dkr.ecr.eu-central-1.amazonaws.com/flask:latest
latest: Pulling from flask
ec37562cf8fa: Already exists
9dca7a6ccc9a: Already exists
578da882ab39: Already exists
97b6bb016efe: Pull complete
848a404de36e: Pull complete
Digest:
sha256:b6fcf78ac04e397ec92b588e0dac9d4e0ed5678c4f3b1d5912e9ec481cf31cd7
Status: Downloaded newer image for 123456789.dkr.ecr.eu-
central-1.amazonaws.com/flask:latest

# docker tag 1bc8aff742c5 flask:v2.0
# docker tag flask:v2.0 123456789.dkr.ecr.eu-
```

```
central-1.amazonaws.com/flask:v2.0

# docker push 123456789.dkr.ecr.eu-central-1.amazonaws.com/flask:v2.0
The push refers to a repository [123456789.dkr.ecr.eu-
central-1.amazonaws.com/flask]
5c2fb49623e3: Layer already exists
e68c063c9469: Layer already exists
80cc614be280: Layer already exists
1c1f633d1481: Layer already exists
23b9c7b43573: Layer already exists
v2.0: digest:
sha256:b6fcf78ac04e397ec92b588e0dac9d4e0ed5678c4f3b1d5912e9ec481cf31cd7
size: 1361
```

В результате наш репозитарий будет иметь следующий вид



Все что надо сделать в контексте Elastic Beanstalk - исправить Dockerrun.aws.json, а потом произвести redeploy.

```
{
    "AWSEBDockerrunVersion": 2,

    "containerDefinitions": [{
        "memory": 256,
```

```
        "cpu": 1,
        "name": "flask",
        "image": "123456789.dkr.ecr.eu-central-1.amazonaws.com/flask:v2.0",

        "portMappings": [{
            "hostPort": 80,
            "containerPort": 8080
        }]
    }]
}
```