Microservices:

In both microservices dockerfiles I've changed FROM python:latest to FROM python:2.7, as it seems that "fprocess=python2 entrypoint.py" requires python2 instead of latest python3 which is python 3.11.

I would not recommend using the latest version of images in cases where the goal is a stable microservice, as it introduces an additional failure point. As the Python language continues to evolve, some of our microservice code could become deprecated or unsupported, as Python's backward compatibility is not guaranteed.

When developing such microservices, it could be beneficial to periodically build the Docker image from the latest version of supporting packages and images. Testing the stable baseline code in a new environment is a good way to maintain compatibility.

Having both services use and expose the same port (8080) is not ideal, as it would cause problems when running on the same IP. Instead of changing the ports, I chose to work around them. In the "deploy.sh" script, we can route service 2's port with -p 8081:8080, while service 1 keeps port 8080. I've also added the ENV SERVICE1_URL environment variable to the service 2 Dockerfile, so it can be changed for service 1.

Continuous Integration/Delivery:

For CI, I've chosen to use Jenkins as it's a very powerful tool when combined with plugins. In addition, I used Github for SCM and Docker Hub to store images.

Find my repo here: https://github.com/Dmausa/DevopsTest

The pipeline consists of four consecutive steps, each of which is encapsulated in its own Jenkins job, making it easier to maintain and troubleshoot, and allowing us to retrigger or execute partial builds. The Github repository triggers the pipeline when a new patch is pushed to the main branch. However, the automatic trigger will not execute the deployment step, which needs to be enabled manually when starting the pipeline. The pipeline parameters allow us to specify the services' **version** and opt-in to the deploy step.

Master-pipeline
|
|----- 2x Build Docker image (Service 1 and Service 2)
|
|----- 2x Test Image
|
|----- Upload Image to Dockerhub
|
|----- Deploy (deploy.sh or kubernetes)

To ensure the pipeline's maintainability and configurability, I've used the JobDSL plugin for Jenkins, allowing us to store and deploy the pipeline from DSL code. The configuration files are part of the Git project, allowing us to version control the flow. Changes made through the Jenkins UI are not permanent unless added to the DSL configuration files, and can be reverted at any time by starting the "DSL Deploy" job. This makes it easy to prototype and reconfigure the pipeline without risk. In addition, I would recommend having a separate staging or sandbox Jenkins instance where changes to the flow can be verified before deployment.

All files can be found in the JenkinsConfig folder, so I'll skip a detailed description of each step. The "Test" step only ensures that the container can run and is reachable on the assigned test port 8081. The "Upload" step pushes the files to Docker Hub. For production deployment, it might be better to host a private image repository or store the images as files.

Continuous Deployment:

Simple docker deployment can be done by calling deploy shell script:
(./deploy --version_service1=v1.0 --version_service2=v1.0 –additional_param_1=5)
Script can be found at root of repository, it simply parses arguments and call docker run like this:

```
docker run -d -p 8080:8080 -e ADDITIONAL_PARAM_1=$additional_param_1 --name=service1
dmausa/service1:$version_service1
docker run -d -p 8081:8080 -e ADDITIONAL_PARAM_1=$additional_param_1 --name=service2
dmausa/service2:$version_service2
```

It runs docker containers from Dockerhub images (repo dmausa) on ports 8080 and 8081.

Kubernetes deployment:

The Kubernetes YAML files can be found in each microservice directory. I've defined three files for each microservice: deployment_sX.yaml, service_sX.yaml, and pod_sX.yaml.

Deployment:

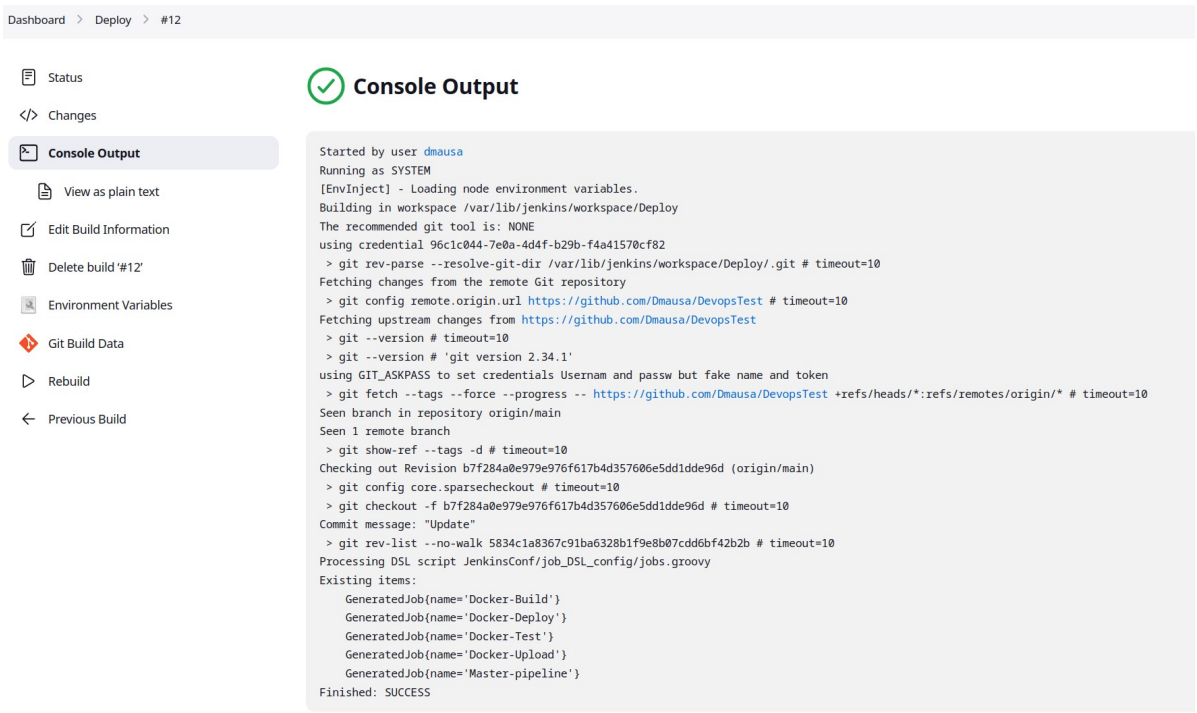
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: service1
spec:
  selector:
    matchLabels:
      service: v1.0
  replicas: 1
  template:
    metadata:
      labels:
        service: v1.0
    spec:
      containers:
        - image: dmausa/service1:v1.0
          name: service1
          env:
          - name: ADDITIONAL_PARAM_1
            value: "5"
          ports:
- containerPort: 8080
```

Service:

```
apiVersion: v1
kind: Service
metadata:
  name: service1
spec:
  selector:
    service: v1.0
  ports:
    - name: service1-port
      port: 8080
      targetPort: 8080
  type: ClusterIP
```

I will demonstrate workflow for this project:

First, Job DSL configuration is deployed with latest changes to the flow:



With: $ git push

```
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 16 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 479 bytes | 479.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/Dmausa/DevopsTest.git
   5834c1a..b7f284a  main -> main
```

Github WebHook should start Master-pipeline. Unfortunately my local deployment of Jenkins can't be reached due to ISP limitation. Other way to start flow is chron Git poll or manual trigger:

Build step is run twice, once for each service:

```
Started by upstream project "Master-pipeline" build number 65
originally caused by:
 Started by user dmausa
Running as SYSTEM
[EnvInject] - Loading node environment variables.
Building in workspace /var/lib/jenkins/workspace/Docker-build
[Docker-build] $ /bin/sh -xe /tmp/jenkins10751132784435441352.sh
+ cd /var/lib/jenkins/workspace/Docker-build/Service1
+ echo Service1
+ tr [:upper:] [:lower:]
+ LOWERCASE_SERVICE_NAME=service1
+ docker build -f Dockerfile -t service1:v1.0 .
#1 [internal] load build definition from Dockerfile
#1 transferring dockerfile: 32B done
#1 DONE 0.0s

#2 [internal] load .dockerignore
#2 transferring context: 2B done
#2 DONE 0.0s

#3 [internal] load metadata for docker.io/library/python:latest
#3 DONE 0.0s

#4 [internal] load build context
#4 DONE 0.0s

#5 [1/3] FROM docker.io/library/python:latest
#5 DONE 0.0s

#4 [internal] load build context
#4 transferring context: 35B done
#4 DONE 0.0s

#6 [2/3] RUN curl -sL
https://github.com/openfaas/faas/releases/download/0.9.14/fwatchdog >
/usr/bin/fwatchdog     && chmod +x /usr/bin/fwatchdog
#6 CACHED

#7 [3/3] COPY entrypoint.py /
#7 CACHED

#8 exporting to image
#8 exporting layers done
#8 writing image
sha256:10be9138fe0fd8d3a9d2e522b08316a525b9eec3eaeb0948d59d5ee6c667f584
done
#8 naming to docker.io/library/service1:v1.0 done
#8 DONE 0.0s
+ touch /var/lib/jenkins/workspace/Docker-build/prop.env
+ echo SERVICE_TAG=service1:v1.0
[EnvInject] - Injecting environment variables from a build step.
[EnvInject] - Injecting as environment variables the properties file path
'/var/lib/jenkins/workspace/Docker-build/prop.env'
[EnvInject] - Variables injected successfully.
Finished: SUCCESS
```

In **Test step**, containers should be tested for functionality, for purpose of this demo, containers are started and tested for connection on assigned ports:

```
Started by upstream project "Master-pipeline" build number 65
originally caused by:
 Started by user dmausa
Running as SYSTEM
[EnvInject] - Loading node environment variables.
Building in workspace /var/lib/jenkins/workspace/Docker-Test
[Docker-Test] $ /bin/sh -xe /tmp/jenkins2092064094149544061.sh
+ docker stop testService
testService
+ docker rm -f testService
testService
+ docker container ls --all
CONTAINER ID   IMAGE                  COMMAND       CREATED
STATUS                      PORTS     NAMES
f6f8c33b518c   dmausa/service2:v1.0   "fwatchdog"   21 minutes ago
Created                               service2
6662d7d9dbc2   dmausa/service1:v1.0   "fwatchdog"   21 minutes ago
Exited (0) 18 minutes ago             service1
+ docker run -d -p 8082:8080 --name testService service1:v1.0
37d009bb1a41326f7257118badf83f32bab001a950e6951ec5fe8ead19db0edf
+ nc -zv localhost 8082
Connection to localhost (127.0.0.1) 8082 port [tcp/*] succeeded!
+ docker stop testService
testService
Finished: SUCCESS
```

If both services pass testing stage, pipeline will **Upload** new images to DockerHub:

```
Started by upstream project "Master-pipeline" build number 65
originally caused by:
 Started by user dmausa
Running as SYSTEM
[EnvInject] - Loading node environment variables.
Building in workspace /var/lib/jenkins/workspace/Docker-Upload
[Docker-Upload] $ /bin/sh -xe /tmp/jenkins13699492811708275249.sh
+ docker tag service1:v1.0 dmausa/service1:v1.0
+ docker push dmausa/service1:v1.0
The push refers to repository [docker.io/dmausa/service1]
96ba27deb391: Preparing
<Deleted Lines>
a9099c3159f5: Layer already exists
v1.0: digest:
sha256:cdbc3d7c917042f07caeb8d23df703d43daacddc403fa4e7308f789dbfc08ff2
size: 2636
+ docker tag service2:v1.0 dmausa/service2:v1.0
+ docker push dmausa/service2:v1.0
The push refers to repository [docker.io/dmausa/service2]
157c482293b1: Preparing
<Deleted Lines>
a463dbda4664: Layer already exists
v1.0: digest:
sha256:4114d6ad3d1a49e0db2b1d9d202c2164c410c629a94f17df8596b7a31f3c1928
size: 2847
Finished: SUCCESS
```

In last step pipeline will try to **Deploy** new containers:

```
Started by upstream project "Master-pipeline" build number 65
originally caused by:
Running as SYSTEM
[EnvInject] - Loading node environment variables.
Building in workspace /var/lib/jenkins/workspace/Docker-Deploy
[Docker-Deploy] $ /bin/bash /tmp/jenkins1164631821508495703.sh
Found running container with name: service1, stopping...
service1
service1
Found running container with name: service2, stopping...
service2
service2
68d57ecc183d34276c8aa2fbebdc28523193082e8cd1640f0182c99704e29650
e6043bf0acdfce82443a3ed5cb5702b77c8f181db2314f7d1965374ad3eaec48
Finished: SUCCESS
```

Another way to automatically deploy services is through a Kubernetes deployment:

```
$ kubectl apply -f ./Serivce1/deployment_s1.yaml

$ kubectl apply -f ./Serivce1/deployment_s1.yaml

$ kubectl get pods

      NAME                         READY   STATUS    RESTARTS   AGE

      service1-69d5c66886-4q5vm    1/1     Running   0          55s


$ kubectl describe service service1

      Name:              service1
      Namespace:         default
      Labels:            <none>
      Annotations:       <none>
      Selector:          service=v1.0
      Type:              ClusterIP
      IP Family Policy:  SingleStack
      IP Families:       IPv4
      IP:                10.96.27.116
      IPs:               10.96.27.116
      Port:              service1-port  8080/TCP
      TargetPort:        8080/TCP
      Endpoints:         10.244.0.29:8080,10.244.0.30:8080
      Session Affinity:  None
      Events:            <none>
```

$ kubectl describe deployment service1

```
Name:                   service1
Namespace:              default
CreationTimestamp:      Sun, 05 Feb 2023 18:57:46 +0100
Labels:                 <none>
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               service=v1.0
Replicas:               1 desired | 1 updated | 1 total | 1
available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  service=v1.0
  Containers:
   service1:
    Image:      dmausa/service1:v1.0
    Port:       8080/TCP
    Host Port:  0/TCP
    Environment:
      ADDITIONAL_PARAM_1:  5
    Mounts:                <none>
  Volumes:                 <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   service1-69d5c66886 (1/1 replicas created)
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  6m59s  deployment-controller  Scaled up
replica set service1-69d5c66886 to 1
```