Utils2DevOps Documentation

Release 0.1.4

Alain Ivars

Sep 03, 2020

How to use it:

CHAPTER 1

Contents

1.1 Create your local cluster of docker-machine

This script will create x nodes docker-machine named node.1 to name-n, the purpose of this script is for local debug or tests, you can after access to these by:

```
docker-machine ssh node.n
```

required:

```
install Docker-machine:
https://docs.docker.com/machine/install-machine/
```

run:

```
./utils2devops/bin/docker-machine-cluster.sh
started...
usage:
  docker-machine-cluster [-h | --help] To get this help
  docker-machine-cluster [-c | --create x] [-m | --mask y]
      Where x is the number of node to create/add.
      Where y is the mask of the node: example: node- or server-, default is node.
  docker-machine-cluster [-d | --destroy x] [-m | --mask y]
      Where x is the number of node to destroy.
      Where y is the mask of the node: example: node. or server., default is node.
```

Create a cluster of 4 node:

```
./utils2devops/bin/docker-machine-cluster.sh -m node. -c 5
```

Destroy it:

```
./utils2devops/bin/docker-machine-cluster.sh -m node. -d 5
```

1.2 Create a Docker Swarm on nodes docker-machine

This script will create a swarm on nodes docker-machine named node.1 to name-n, the purpose of this script is for local debug or tests, you can after access to these by:

```
docker-machine ssh node.n
```

required:

```
install Docker Swarm:
https://docs.docker.com/engine/swarm/
```

run:

```
./utils2devops/bin/swarm.sh
started...
usage:
  swarm [-h | --help] To get this help
      If the docker-swarm don't exist it will be created
  swarm -c|--create [-m|--count_manager x -w|--count_worker y] To create node to a␣
↪swarm
      Where x is the number of manager node to create/add in the swarm.
      Where y is the number of worker node to create/add in the swarm.
  swarm -r|--remove x] To destroy a swarm
      Where x is the number of node in the swarm..
```

Create a docker swarm of 3 manager and 2 worker:

```
./utils2devops/bin/swarm.sh -c -m 3 -w 2
```

Destroy it:

```
./utils2devops/bin/swarm.sh -r 5
```

1.3 Create your local cluster of machine

This playbook will create x nodes machine named name1 to nameN:: the purpose of this script is for local debug or tests, you can after access to these by:

```
vagrant ssh nodeN
```

required:

```
install Vagrant:
https://www.vagrantup.com/docs/installation/
```

run:

```
export VAGRANT_NODES_COUNT=5
cd vagrant
vagrant up
```

Destroy it:

```
export VAGRANT_NODES_COUNT=5
vagrant destroy -f
```

1.4 Add local nodes in docker-machine

This script will create a swarm on nodes machine named node1 to nameN, the purpose of this script is for local debug or tests, you can after access to these by:

```
docker-machine ssh nodeN
```

required:

```
install Docker machine:
https://docs.docker.com/engine/swarm/
```

run:

```
WORK IN PROGRESS
docker-machine create \
    --driver generic \
    --generic-ssh-user "ubuntu" \
    --generic-ip-address=35.170.64.155 \
    --generic-ssh-key ~/.ssh/terraform_key \
    node1
```

1.5 Swarm Stack local-with-out-proxy

Before using these make sure you had clone the repository by:

```
git submodule update --init --recursive
```

Now let's go..

(*) All Open Sources

Services	Software
GUI Control	Portainer
Central Monitoring	Promotheus + Grafana
Central Logging	Elastic ELK
Layer 7 Proxy	
Storage	Local File System
Networking	Docker Swarm Overlay
Orchestration	Docker Swarm
Runtime	Docker CE
Machine and OS	Docker Machine + VirtualBox

You have 2 way to deploy it:

```
- The fast way by launch the ansible workbook, just type:

    ansible-playbook -i ansible/swarm/local-inventory ansible/swarm/local-with-out-
↪proxy.yml
    # NOTE: that will take around 5 minutes

- Or the long way but where you can learn every step ...
```

1.5.1 Learn every step to deploy the swarm local-with-out-proxy

1/ Create the Machine:

```
./utils2devops/bin/docker-machine-cluster.sh -c 5
```

You can go to see the doc of this tools here *Create your local cluster of docker-machine*
Here we will create a swarm with 5 machines

2/ Enable monitoring (optional):

```
./utils2devops/bin/enable-monitoring.sh -p ./utils2devops/docker/ -n 5
```

3/ Create the Docker Swarm:

```
./utils2devops/bin/swarm.sh -c -m 3 -w 2
```

You can go to see the doc of this tools here *Add local nodes in docker-machine*
Here we will create a swarm with 3 manager and 2 worker

4/ Launch docker command in the Master:

```
eval "$(docker-machine env node-1)"
```

5/ Deploy Ops Stacks Graphics UI (optional):

```
export PORTAINER_HOST=portainer.example.com
docker stack deploy -c ./utils2devops/docker/local-with-out-proxy/portainer.yml␣
↪portainer
```

After these steps we will have a Portainer at:

```
Portainer at:
    http://<ip-node-1>:9000/#/init/admin
    http://<ip-node-1>:9000/#/dashboard
    http://<ip-node-1>:9000/#/containers
    http://<ip-node-1>:9000/#/swarm/visualizer
and so many other... have a look here https://www.portainer.io/overview/
```

6/ Deploy Ops Stacks:

```
docker stack deploy -c ./submodules/swarmprom/docker-compose.yml prom
docker stack deploy -c ./utils2devops/docker/local-with-out-proxy/elk.yml elk
```

After these steps we will have

```
Grafana login at:
    https://<ip-node-1>/login
Grafana Swarm nodes at:
    https://<ip-node-1>:3000/d/BPlb-Sgik/docker-swarm-nodes?refresh=30s&orgId=1
Grafana Swarm Services at:
    https://<ip-node-1>:3000/d/zr_baSRmk/docker-swarm-services?refresh=30s&orgId=1
Promotheus Stat at:
    http://<ip-node-1>:3000/d/mGFfYSRiz/prometheus-2-0-stats?refresh=1m&orgId=1
Promotheus Query at::
    https://<ip-node-1>::9090/graph
Alert manager at:
    https://<ip-node-1>:9093/#/alerts
Alert Dashboard at:
    https://<ip-node-1>:9094/?q=
Elasticsearch at:
    http://elasticsearch.example.com/
kibana at:
    http://kibana.example.com/app/kibana#/home?_g=()
and much more have a look at https://github.com/stefanprodan/swarmprom
```

Now it's ready to deploy your apps and test them:

```
docker stack deploy my_company/my_services my_service
```

When you have finish to use it, Destroy it by:

```
./utils2devops/bin/docker-machine-cluster.sh -d 5
```

1.6 Swarm Stack local-proxy-prom-elk

Before using these make sure you had clone the repository by:

```
git submodule update --init --recursive
```

That example of local deployment is nearly the same to the previews one except we will add the reverse proxy Traefik to permit to us to have many more service even if they need the same port! | Even if in surface that look like same, the deployment file are all modified! | Now let's go..

(*) All Open Sources

Services	Software
GUI Control	Portainer
Central Monitoring	Promotheus + Grafana
Central Logging	Elastic ELK
Layer 7 Proxy	Traefik
Storage	Local File System
Networking	Docker Swarm Overlay
Orchestration	Docker Swarm
Runtime	Docker CE
Machine and OS	Docker Machine + VirtualBox

You have 2 way to deploy it:

```
- The fast way by launch the ansible workbook, just type:

    ansible-playbook -i ansible/swarm/local-inventory ansible/swarm/local-proxy-prom-
↪elk.yml
    # NOTE: that will take around 5 minutes

- Or the long way but where you can learn every step ...
```

1.6.1 Learn every step to deploy the local swarm

1/ Create the Machine:

```
./utils2devops/bin/docker-machine-cluster.sh -c 5
```

You can go to see the doc of this tools here *Create your local cluster of docker-machine*
Here we will create a swarm with 3 machines

2/ Enable monitoring (optional):

```
./utils2devops/bin/enable-monitoring.sh -p ./utils2devops/docker/ -n 5
```

3/ Create the Docker Swarm:

```
./utils2devops/bin/swarm.sh -c -m 3 -w 2
```

You can go to see the doc of this tools here *Add local nodes in docker-machine*
Here we will create a swarm with 3 manager and 2 worker

4/ To launch docker command in the Master with ssh it:

```
eval "$(docker-machine env node-1)"
```

5/ Deploy Traefik:

```
export TRAEFIK_HOST=traefik.yourdomain
default value: traefik.example.com
export TRAEFIK_PUBLIC_TAG=my-traefik-public
default value: traefik-public
docker stack deploy -c ./utils2devops/docker/local-with-prom-elk/traefik.yml traefik
```

After this step we will have a proxy Dashboard at:

```
http://traefik.example.com:8080/dashboard/
```

7/ Deploy Ops Stacks Graphics UI (optional):

```
export PORTAINER_HOST=portainer.yourdomain
default value: portainer.example.com
docker stack deploy -c ./utils2devops/docker/local-with-prom-elk/portainer.yml␣
↪portainer
```

After these steps we will have:

```
Portainer at:
    http://portainer.example.com/#/init/admin
    http://portainer.example.com/#/dashboard
    http://portainer.example.com/#/containers
    http://portainer.example.com/#/swarm/visualizer
and so many other... have a look here https://www.portainer.io/overview/
```

8/ Deploy Ops Stacks:

```
export ADMIN_USER=admin
default value: admin
export ADMIN_PASSWORD=adminadmin
default value: adminadmin
export HASHED_PASSWORD=$(openssl passwd -apr1 -salt pepper $ADMIN_PASSWORD)
```

You can check the contents with:

```
echo $HASHED_PASSWORD
```

it will look like:

```
$apr1$TsqS2JR3$oGG0NFZsU1VdKn03MAyjh.
```

Create and export an environment variable DOMAIN, e.g.::

```
export DOMAIN=example.com
```

and make sure that the following sub-domains point to your Docker Swarm cluster IPs:

```
grafana.example.com
alertmanager.example.com
unsee.example.com
prometheus.example.com
```

Note: You can also use a subdomain, like swarmprom.example.com. Just make sure that the subdomains point to (at least one of) your cluster IPs. Or set up a wildcard subdomain (*).

Set and export an environment variable with the tag used by Traefik public to filter services (by default, it's traefik-public):

```
export TRAEFIK_PUBLIC_TAG=traefik-public
```

If you are using Slack and want to integrate it, set the following environment variables:

```
export SLACK_URL=https://hooks.slack.com/services/TOKEN
default value: https://hooks.slack.com/services/TOKEN
export SLACK_CHANNEL=utils2devops-tests
default value: general
export SLACK_USER=alertmanager
default value: alertmanager
```

Then we continue to deploy with swarmprom:

```
docker stack deploy -c ./utils2devops/docker/local-with-prom-elk/swarmprom.yml prom
```

After these steps we will have

```
Grafana login at:
    https://grafana.example.com/login
Grafana Swarm nodes at:
    https://grafana.example.com/d/BP1b-Sgik/docker-swarm-nodes?refresh=30s&orgId=1
Grafana Swarm Services at:
    https://grafana.example.com/d/zr_baSRmk/docker-swarm-services?refresh=30s&orgId=1
Promotheus Stat at:
    http://grafana.example.com/d/mGFfYSRiz/prometheus-2-0-stats?refresh=1m&orgId=1
Promotheus Query at::
    https://prometheus.example.com/graph
Alert manager at:
    https://alertmanager.example.com/#/alerts
Alert Dashboard at:
    https://unsee.example.com/?q=
```

In promotheus try:

```
sum(irate(container_cpu_usage_seconds_total{image!=""}[1m])) without (cpu)
container_memory_usage_bytes{image!=""}
sum(rate(container_network_transmit_bytes_total{image!=""}[1m])) without (interface)
sum(rate(container_fs_reads_bytes_total{image!=""}[1m])) without (device)
sum(rate(container_fs_writes_bytes_total{image!=""}[1m])) without (device)
```

Then we finish to deploy with elk:

```
export ELASTICSEARCH_USER=admin
default value: admin
export ELASTICSEARCH_PASSWORD=adminadmin
default value: admin
export ELASTICSEARCH_HASHED_PASSWORD=$(openssl passwd -apr1 -salt pepper
↪$ELASTICSEARCH_PASSWORD)
export KIBANA_USER=admin
default value: admin
export KIBANA_PASSWORD=adminadmin
default value: admin
export KIBANA_HASHED_PASSWORD=$(openssl passwd -apr1 $KIBANA_PASSWORD)
docker stack deploy -c ./utils2devops/docker/local-with-prom-elk/elk.yml elk
```

After these steps we will have:

```
Elasticsearch at:
    http://elasticsearch.example.com/
kibana at:
    http://kibana.example.com/app/kibana#/home?_g=()
and much more have a look at https://github.com/stefanprodan/swarmprom
```

Note: | To use elasticsearch you will have to increase the max virtual memory or you will get: | WARN: max virtual memory areas vm.max_map_count [65530] is too low, increase to at least [262144] | It can be fixed by running on the node: | sysctl -w vm.max_map_count=262144 in terminal (Linux/Ubuntu). | Or permanently by create a file name 60-elasticsearch.conf and place it in /etc/sysctl.d/ with the following content: | vm.max_map_count=262144

Now it's ready to deploy your apps and test them:

```
docker stack deploy my_company/my_services my_service
```

When you have finish to use it, Destroy it by:

```
./utils2devops/bin/docker-machine-cluster.sh -d 5
```

1.7 Swarm Stack local-proxy-prom-Icinga2

Before using these make sure you had clone the repository by:

```
git submodule update --init --recursive
```

That example of local deployment is nearly the same to the previews one except we will replace ELK by Icinga2 + plugin logs! | Now let's go..

(*) All Open Sources

Services	Software
GUI Control	Portainer
Central Monitoring	Promotheus + Grafana
Central Logging	Icinga2
Layer 7 Proxy	Traefik
Storage	Local File System
Networking	Docker Swarm Overlay
Orchestration	Docker Swarm
Runtime	Docker CE
Machine and OS	Docker Machine + VirtualBox

You have 2 way to deploy it:

```
- The fast way by launch the ansible workbook, just type:

    ansible-playbook -i ansible/swarm/local-inventory ansible/swarm/local-proxy-prom-
↪icinga2.yml
    # NOTE: that will take around 5 minutes

- Or the long way but where you can learn every step ...
```

1.7.1 Learn every step to deploy the local swarm

1/ Create the Machine:

```
./utils2devops/bin/docker-machine-cluster.sh -c 5
```

You can go to see the doc of this tools here *Create your local cluster of docker-machine*
Here we will create a swarm with 3 machines

2/ Enable monitoring (optional):

```
./utils2devops/bin/enable-monitoring.sh -p ./utils2devops/docker/ -n 5
```

3/ Create the Docker Swarm:

```
./utils2devops/bin/swarm.sh -c -m 3 -w 2
```

You can go to see the doc of this tools here *Add local nodes in docker-machine*
Here we will create a swarm with 3 manager and 2 worker

4/ To launch docker command in the Master with ssh it:

```
eval "$(docker-machine env node-1)"
```

5/ Deploy Traefik:

```
export TRAEFIK_HOST=traefik.yourdomain
default value: traefik.example.com
export TRAEFIK_PUBLIC_TAG=my-traefik-public
default value: traefik-public
docker stack deploy -c ./utils2devops/docker/local-proxy-prom-icinga2/traefik.yml␣
↪traefik
```

After this step we will have a proxy Dashboard at:

```
http://traefik.example.com:8080/dashboard/
```

7/ Deploy Ops Stacks Graphics UI (optional):

```
export PORTAINER_HOST=portainer.yourdomain
default value: portainer.example.com
docker stack deploy -c ./utils2devops/docker/local-with-prom-elk/portainer.yml␣
↪portainer
```

After these steps we will have:

```
Portainer at:
    http://portainer.example.com/#/init/admin
    http://portainer.example.com/#/dashboard
    http://portainer.example.com/#/containers
    http://portainer.example.com/#/swarm/visualizer
and so many other... have a look here https://www.portainer.io/overview/
```

8/ Deploy Ops Stacks:

```
export ADMIN_USER=admin
default value: admin
export ADMIN_PASSWORD=adminadmin
default value: adminadmin
export HASHED_PASSWORD=$(openssl passwd -apr1 -salt pepper $ADMIN_PASSWORD)
```

You can check the contents with:

```
echo $HASHED_PASSWORD
```

it will look like:

```
$apr1$TsqS2JR3$oGG0NFZsU1VdKn03MAyjh.
```

Create and export an environment variable DOMAIN, e.g.::

```
export DOMAIN=example.com
```

and make sure that the following sub-domains point to your Docker Swarm cluster IPs:

```
grafana.example.com
alertmanager.example.com
unsee.example.com
prometheus.example.com
```

Note: You can also use a subdomain, like swarmprom.example.com. Just make sure that the subdomains point to (at least one of) your cluster IPs. Or set up a wildcard subdomain (*).

Set and export an environment variable with the tag used by Traefik public to filter services (by default, it's traefik-public):

```
export TRAEFIK_PUBLIC_TAG=traefik-public
```

If you are using Slack and want to integrate it, set the following environment variables:

```
export SLACK_URL=https://hooks.slack.com/services/TOKEN
default value: https://hooks.slack.com/services/TOKEN
export SLACK_CHANNEL=utils2devops-tests
default value: general
export SLACK_USER=alertmanager
default value: alertmanager
```

Then we continue to deploy with swarmprom:

```
docker stack deploy -c ./utils2devops/docker/local-proxy-prom-icinga2/swarmprom.yml␣
↪prom
```

After these steps we will have

```
Grafana login at:
    https://grafana.example.com/login
Grafana Swarm nodes at:
    https://grafana.example.com/d/BPlb-Sgik/docker-swarm-nodes?refresh=30s&orgId=1
Grafana Swarm Services at:
    https://grafana.example.com/d/zr_baSRmk/docker-swarm-services?refresh=30s&orgId=1
Promotheus Stat at:
```

(continues on next page)

(continued from previous page)

```
    http://grafana.example.com/d/mGFfYSRiz/prometheus-2-0-stats?refresh=1m&orgId=1
Promotheus Query at::
    https://prometheus.example.com/graph
Alert manager at:
    https://alertmanager.example.com/#/alerts
Alert Dashboard at:
    https://unsee.example.com/?q=
```

In promotheus try:

```
sum(irate(container_cpu_usage_seconds_total{image!=""}[1m])) without (cpu)
container_memory_usage_bytes{image!=""}
sum(rate(container_network_transmit_bytes_total{image!=""}[1m])) without (interface)
sum(rate(container_fs_reads_bytes_total{image!=""}[1m])) without (device)
sum(rate(container_fs_writes_bytes_total{image!=""}[1m])) without (device)
```

Then we finish to deploy with elk:

```
export ELASTICSEARCH_USER=admin
default value: admin
export ELASTICSEARCH_PASSWORD=adminadmin
default value: admin
export ELASTICSEARCH_HASHED_PASSWORD=$(openssl passwd -apr1 -salt pepper
↪$ELASTICSEARCH_PASSWORD)
export KIBANA_USER=admin
default value: admin
export KIBANA_PASSWORD=adminadmin
default value: admin
export KIBANA_HASHED_PASSWORD=$(openssl passwd -apr1 $KIBANA_PASSWORD)
docker stack deploy -c ./utils2devops/docker/local-proxy-prom-icinga2/elk.yml elk
```

After these steps we will have:

```
Elasticsearch at:
    http://elasticsearch.example.com/
kibana at:
    http://kibana.example.com/app/kibana#/home?_g=()
and much more have a look at https://github.com/stefanprodan/swarmprom
```

Note: | To use elasticsearch you will have to increase the max virtual memory or you will get: | WARN: max virtual memory areas vm.max_map_count [65530] is too low, increase to at least [262144] | It can be fixed by running on the node: | sysctl -w vm.max_map_count=262144 in terminal (Linux/Ubuntu). | Or permanently by create a file name 60-elasticsearch.conf and place it in /etc/sysctl.d/ with the following content: | vm.max_map_count=262144

Now it's ready to deploy your apps and test them:

```
docker stack deploy my_company/my_services my_service
```

When you have finish to use it, Destroy it by:

```
./utils2devops/bin/docker-machine-cluster.sh -d 5
```

1.8 Testing

You can run the tests by

```
tox
```

or by

```
python setup.py test
```

or by

```
pytest
```

1.9 Aws Tools

Create an AWS account (it's free):

```
https://aws.amazon.com/
```

Aws secret required, create a ~/.aws/credential file and add:

```
[default]
aws_access_key_id = <Your key id>
aws_secret_access_key = <Your secret access key id>
[terraform]
aws_access_key_id = <Your key id>
aws_secret_access_key = <Your secret access key id>
```

Aws Env required, create a ~/.aws/config file and add:

```
[default]
output = json
region = <the region name>
[terraform]
output = json
region = <the region name>
```

1.9.1 List Api Gateway

To list all the api gateway for the region:

```
PYTHONPATH=. python utils2devops/aws/network_acl.py

resource "aws_network_acl" "acl-06807accba82" {
    vpc_id = "vpc-0b4bc90e75638"
    subnet_ids = ["subnet-0996d401f115e",]

    ingress {
        rule_no = "100"
        action = "allow"
        cidr_block = "0.0.0.0/0"
        ipv6_cidr_block = ""
        protocol = "-1"
        from_port = 0
        to_port = 0
        icmp_code = 0
```

(continues on next page)

(continued from previous page)

```
        icmp_type = 0
    }

    egress {
        rule_no = "100"
        action = "allow"
        cidr_block = "0.0.0.0/0"
        ipv6_cidr_block = ""
        protocol = "-1"
        from_port = 0
        to_port = 0
        icmp_code = 0
        icmp_type = 0
    }

    tags {
    }
}

 resource "aws_network_acl" "acl-c31fc4a4" {
    vpc_id = "vpc-4a50ae2d"
    subnet_ids = ["subnet-35d6d","subnet-09a005","subnet-bdb380",]

    ingress {
        rule_no = "100"
        action = "allow"
        cidr_block = "0.0.0.0/0"
        ipv6_cidr_block = ""
        protocol = "-1"
        from_port = 0
        to_port = 0
        icmp_code = 0
        icmp_type = 0
    }

    egress {
        rule_no = "100"
        action = "allow"
        cidr_block = "0.0.0.0/0"
        ipv6_cidr_block = ""
        protocol = "-1"
        from_port = 0
        to_port = 0
        icmp_code = 0
        icmp_type = 0
    }

    tags {
    }
}
```

1.9.2 List Aws lambda

To list all the Aws lambda for the region:

```
PYTHONPATH=. python utils2devops/aws/aws_lambda.py
```

1.9.3 List internet gateway

To list all the internet gateway for the region:

```
PYTHONPATH=. python utils2devops/aws/internet_gateway.py

resource "aws_internet_gateway" "igw-9935f0" {
    vpc_id = "vpc-9e987"
    tags {
    }
}
```

1.9.4 List lambda ssm

To list all the lambda ssm for the region:

```
PYTHONPATH=. python utils2devops/aws/lambda_ssm.py
```

1.9.5 List network acl

To list all the network acl for the region:

```
PYTHONPATH=. python utils2devops/aws/network_acl.py

resource "aws_network_acl" "acl-0682c7" {
    vpc_id = "vpc-0b24bc90e75638"
    subnet_ids = ["subnet-061f115e",]

    ingress {
        rule_no = "100"
        action = "allow"
        cidr_block = "0.0.0.0/0"
        ipv6_cidr_block = ""
        protocol = "-1"
        from_port = 0
        to_port = 0
        icmp_code = 0
        icmp_type = 0
    }

    egress {
        rule_no = "100"
        action = "allow"
        cidr_block = "0.0.0.0/0"
        ipv6_cidr_block = ""
        protocol = "-1"
        from_port = 0
        to_port = 0
        icmp_code = 0
        icmp_type = 0
```

(continues on next page)

(continued from previous page)

```
    }

    tags {
    }
}

 resource "aws_network_acl" "acl-c31fc4" {
    vpc_id = "vpc-4a50ae2d"
    subnet_ids = ["subnet-35d6d","subnet-09a730","subnet-bb380","subnet-a35833c6",
↪"subnet-e392f0c9","subnet-ad0e52db",]

    ingress {
        rule_no = "100"
        action = "allow"
        cidr_block = "0.0.0.0/0"
        ipv6_cidr_block = ""
        protocol = "-1"
        from_port = 0
        to_port = 0
        icmp_code = 0
        icmp_type = 0
    }

    egress {
        rule_no = "100"
        action = "allow"
        cidr_block = "0.0.0.0/0"
        ipv6_cidr_block = ""
        protocol = "-1"
        from_port = 0
        to_port = 0
        icmp_code = 0
        icmp_type = 0
    }

    tags {
    }
}
```

1.9.6 List route table

To list all the route table for the region:

```
PYTHONPATH=. python utils2devops/aws/route_table.py

resource "aws_route_table" "" {
    vpc_id = "vpc-4a50ae2d"

    route {
        route_table_id = "rtb-6298e405"
        destination_cidr_block = "172.31.0.0/16"
        gateway_id = "local"
    }

    route {
```

(continues on next page)

(continued from previous page)

```
        route_table_id = "rtb-6298e405"
        destination_cidr_block = "0.0.0.0/0"
        gateway_id = "igw-8bc0adef"
    }
}
```

1.9.7 List s3 bucket

To list all the s3 bucket for the region:

```
PYTHONPATH=. python utils2devops/aws/s3_bucket.py
```

1.9.8 List secret

To list all the secret for the region:

```
PYTHONPATH=. python utils2devops/aws/secretmanager.py
```

1.9.9 List security group

To list all the security group for the region:

```
PYTHONPATH=. python utils2devops/aws/security_group.py
```

1.9.10 List ssm

To list all the ssm for the region:

```
PYTHONPATH=. python utils2devops/aws/ssm.py
```

1.9.11 List subnet

To list all the subnet for the region:

```
PYTHONPATH=. python utils2devops/aws/subnet.py
```

1.9.12 List vpc

To list all the vpc for the region:

```
PYTHONPATH=. python utils2devops/aws/vpc.py
```

1.10 To use lxd_module

require:

```
sudo apt-get install python-pylxd lxd
```

delete object except container:

```
sudo python3 utils2devops/lxd_lxc/lxc_delete.py

usage: lxc_delete.py [-h] [-v] [-e ENDPOINT] [-c CERT] [-sure SURE]
                     [-deleteAllImages] [-deleteAllNetworks]
                     [-deleteAllProfiles] [-deleteAllStorages]

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
  -deleteAllImages      DELETE all lxc image
  -deleteAllNetworks    DELETE all lxc network
  -deleteAllProfiles    DELETE all lxc profile
  -deleteAllStorages    DELETE all lxc storage

  -e ENDPOINT, --endpoint ENDPOINT
                        the endpoint if not local
  -c CERT, --cert CERT  tuple of (cert, key) like ('/path/to/client.crt',
                        '/path/to/client.key')
  -sure SURE            Required for all deleteAll* with value YES_I_AM_SURE
```

container management:

```
sudo python3 utils2devops/lxd_lxc/lxc_container.py

usage: lxc_container.py [-h] [-v] [-e ENDPOINT] [-c CERT] [-sure SURE]
                        (-statusAll | -deleteAll | -startAll | -stopAll)
                        [-controller_uuid CONTROLLER_UUID]
                        [-model_uuid MODEL_UUID]
lxc_container.py: error: one of the arguments -statusAll -deleteAll -startAll -
↪stopAll is required
```

TODO the documentation of existing functionnalities TODO implementation WORK IN PROGESS

1.11 Releases Notes

- 0.1.4: add deploy local stack swarm with Traefik
- **0.1.3: add deploy local stack swarm** add sub module docker-elk and swarmprom
- 0.1.2: refactor tests
- **0.1.1: add single_sourcing_package_version** add documentation (first draft)
- 0.1.0: Initial publication version

1.12 To use terraform_import and aws

require:

```
sudo pip3 install --upgrade awscli
export PATH=/home/ec2-user/.local/bin:$PATH
```

Create an AWS account (it's free):

```
https://aws.amazon.com/
```

Install Terraform:

```
https://learn.hashicorp.com/terraform/getting-started/install.html
```

1.13 To test it

create a main.tf and add this inside:

```
provider "aws" {
  region = "us-east-2"
}

resource "aws_lambda_function" "lambda" {
    # (resource arguments)
}

resource "aws_s3_bucket" "bucket" {
    # (resource arguments)
}
```

log on your AWS account, and create:

```
- an s3 bucket named my-bucket-test-1
- a lambda function my-lambda-test-1
```

then import them by:

```
terraform import aws_lambda_function.lambda my-lambda-test-1

terraform import aws_s3_bucket.bucket my-bucket-test-1

terraform import aws_network_acl.main acl-f1780b98

terraform import aws_lambda_layer_version.test_layer arn:aws:lambda:us-east-
↪2:397270606208:layer:aws_lambda_read_s3:1
```

Then a terraform.tfstate will be generated or updated with your config.TODO the documentation of existing functionnalities

Generate a Key pair for terraform user:

```
cd ~/.ssh
ssh-keygen -f terraform_key
Generating public/private rsa key pair.
```

(continues on next page)

(continued from previous page)

```
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in terraform_key.
Your public key has been saved in terraform_key.pub.
The key fingerprint is:
SHA256:SX5+SVVbyUiaslXmpqFNJl8WrP/NMTzWMnsK5AwxMvI alain@Beowolf007
The key's randomart image is:
+---[RSA 2048]----+
|             o=o +|
|             *o.+o|
|       . +ooB.=.. |
|        = +@+*.   |
|         Eoo++ . .|
|          o * oo*.|
|           . * o==|
|            . ...+|
|               .o |
+----[SHA256]-----+
```

We now have 2 more files:

```
terraform_key
terraform_key.pub
```

Creating the terraform file to import it:: See *Create your local cluster of docker-machine* for setting your profile

provider "aws" { profile = "terraform" region = "eu-west-1"

}

resource "aws_instance" "ubuntu_zesty" { ami = "ami-6b7f610f" instance_type = "t2.micro" key_name = "terraform_ec2_key"

}

resource "aws_key_pair" "terraform_ec2_key" { key_name = "terraform_ec2_key" public_key = "${file("terraform_key.pub")}"

// public_key = "ssh-rsa AZEB...jkasASDhaSjdh me@here" }

1.14 Indices and tables

- genindex
- modindex
- search

CHAPTER 2

Infrastructure As Code

Utils2devops is a package that contain python 3 functions and class that can be helpful in the all working day. Any help for develop, test, validate, documentation are welcome!

With utils2devops you will learn if you don't already know:

```
How to create a Docker swarm locally and deploy it automatically
How to populate a Docker swarm locally and deploy it automatically
How to add an existing AWS Ec2 instance in Docker-machine
...
```

TAKE CARE THIS LIBRARY AND THE TOOLS will do what ever you ask to do, even for destroy image, network, container, . . .

One of my main principles is not to reinvent the wheel, and if someone has already created a function, a class and sharing it, and if I like its implementation, I will use this function / class and say a big thank you to this person in all respect of the Copyright and the Licence. This library is in development and the folder structure will change certainly

We already use :

- AWS Tools
- AWS Cloud Ami, Security group, Ec2
- Docker
- Ansible
- Traefik
- Portainer
- Terraform
- Swarmprom
- Elk
- Icinga2 + plugin logs WORK IN PROGRESS

We will add :

- AWS Cloud S3, RDS, Cloudwatch, ...
- GPC tools
- GPC Cloud
- OpenFaaS
- Consul
- Etcd
- Kubernetes
- ...

To develop or improve this library you can run it with the env DEBUG_OR_IMPROVE

Something disturb you in the code? Don't hesitate to open a an issue and contribute.

Online documentation is here on Readthedoc Online source code available on Github