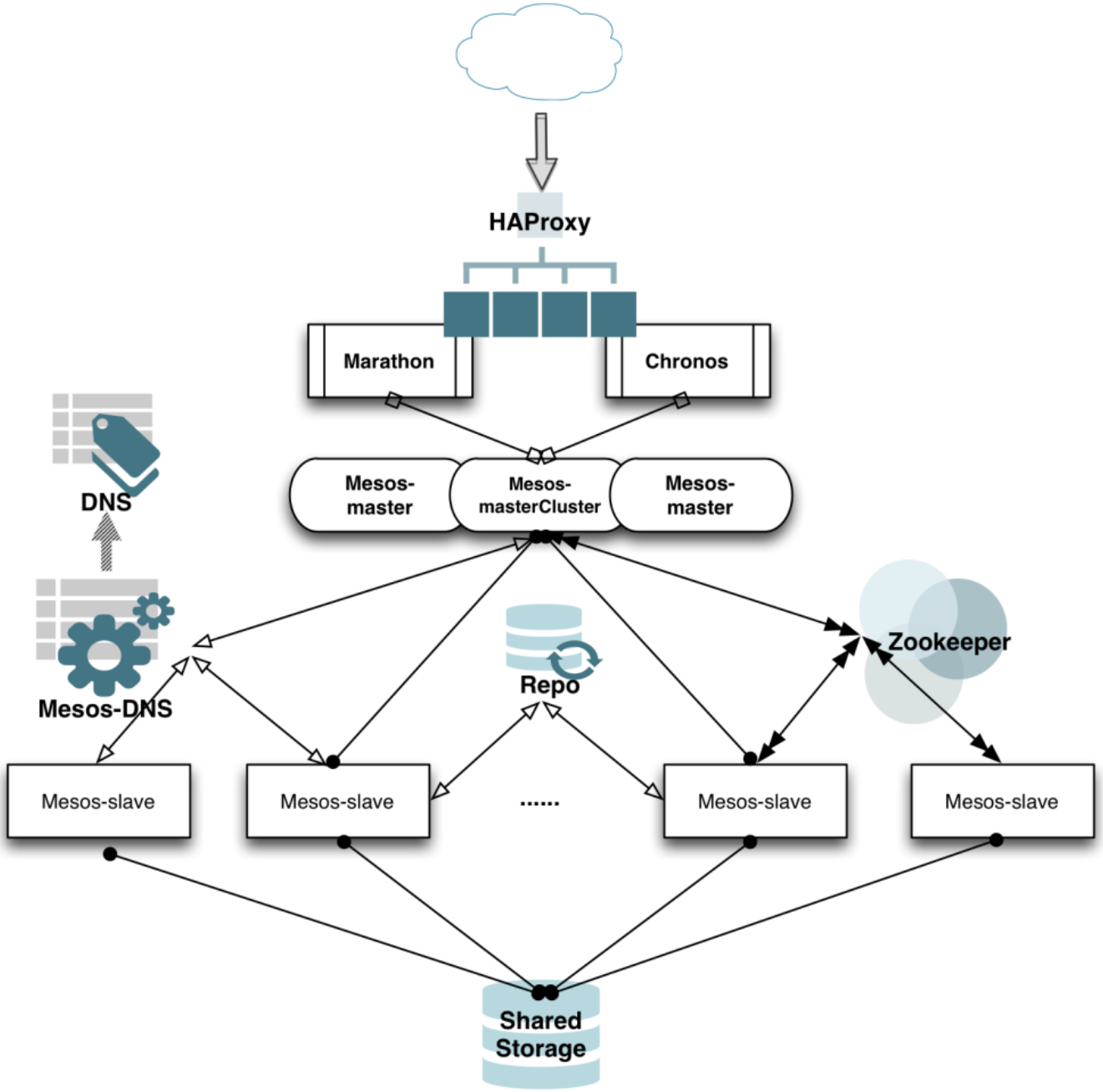
Container Platform With Mesos

Summary:

While PoC is still underway, we would like to present one of the proposed platform architects for deploying Dockerized services on Mesos frameworks. A few services including Kafka, Zookeeper and RabbitMQ have been successfully deployed on such platform.

Reference Architect:

Using Marathon and Chronos frameworks, application/service deployment to various environments become much simpler comparing with CF based platform. Both containerized and regular applications can be seamlessly deployed on Mesos abstraction platforms without complex configuration or run-time environment modification. Leveraging service discovery automation and load balancing functions, multiple service instances can be easily deployed to the platform with full redundancy and without manual intervention.



Considerations and Benefits:

Stability: The approach should be simple and repeatable. The platform should be stable and supportable by engineers from GE.

Fault Tolerance: The platform should have no single point of failure on service. All controller services are clustered without single point of failure. Failed services will be restarted automatically. Services and their dependencies and isolated by containers.

Security: All customer facing API are secured with SSL and password protection.

Agility: Service can be dynamically deployed and migrated to another host. Seamless upgrades are norm.

Capacity Management: Addition nodes can be dynamically added or removed from the mesos-slave farm.

Infrastructure Agnostic: The reference architect can be deployed on AWS, Bare Metals and Virtual servers.

Recommended Procedures and Best Practice:

1. Build a Docker container image with intended application. Carefully craft the application configuration and entry point startup script to handle multiple instances and clustering deployments.

Example: A Dockerfile for a 3-node RabbitMQ cluster

```
alfreds@sjc4dockerdev01:~/rabbitmq$ cat Dockerfile
FROM ubuntu:trusty

# add our user and group first to make sure their IDs get assigned consistently, regardless of whatever dependencies get added
RUN groupadd -r rabbitmq && useradd -r -d /var/lib/rabbitmq -m -g rabbitmq rabbitmq
ENV RABBITMQ_VERSION 3.5.6-1
ENV RABBITMQ_LOGS=- RABBITMQ_SASL_LOGS=-
ENV PATH /usr/lib/rabbitmq/bin:$PATH

# grab gosu for easy step-down from root
RUN  apt-key adv --keyserver-options http-proxy=http://3.39.89.55:8080 --keyserver-options https-proxy=http://3.39.89.55:8080 --keyserver ha.pool.sks-keyserv
ers.net --recv-keys B42F6819007F00F88E364FD4036A9C25BF357DD4 \
        && apt-key adv --keyserver-options http-proxy=http://3.39.89.55:8080 --keyserver-options https-proxy=http://3.39.89.55:8080 --keyserver ha.pool.sks-k
eyservers.net --recv-keys 434975BD900CCBE4F7EE1B1ED208507CA14F4FCA \
        && apt-key adv --keyserver-options http-proxy=http://3.39.89.55:8080 --keyserver-options https-proxy=http://3.39.89.55:8080 --keyserver ha.pool.sks-k
eyservers.net --recv-keys F78372A06FF50C80464FC1B4F7B8CEA6056E8E56

RUN     export http_proxy=http://3.39.89.55:8080 \
        && export https_proxy=http://3.39.89.55:8080 \
        && apt-get update && apt-get install -y curl ca-certificates --no-install-recommends && rm -rf /var/lib/apt/lists/* \
        && curl -o /usr/local/bin/gosu -SL "https://github.com/tianon/gosu/releases/download/1.3/gosu-$(dpkg --print-architecture)" \
        && curl -o /usr/local/bin/gosu.asc -SL "https://github.com/tianon/gosu/releases/download/1.3/gosu-$(dpkg --print-architecture).asc" \
        #&& gpg --verify /usr/local/bin/gosu.asc \
        && rm /usr/local/bin/gosu.asc \
        && chmod +x /usr/local/bin/gosu \
        && echo 'deb http://www.rabbitmq.com/debian testing main' > /etc/apt/sources.list.d/rabbitmq.list \
        && echo 'deb http://packages.erlang-solutions.com/debian jessie contrib' > /etc/apt/sources.list.d/erlang.list \
        && apt-get update && apt-get install -y rabbitmq-server=$RABBITMQ_VERSION --no-install-recommends && rm -rf /var/lib/apt/lists/*


RUN echo "[{rabbit,[{tcp_listeners,[RABBITMQ_PORT]},{loopback_users,[]},{collect_statistics_interval,10000},{heartbeat,10},{channel_max,5000},{cluster_partit
ion_handling,autoheal},{cluster_nodes,{[RABBITMQ_CLUSTER_NODES],disc}},{rabbitmq_management,[{http_log_dir,'/tmp/rabbit-mgmt'},{listener,[{port,RABBITMQ_MANA
GEMENT_PORT}]}]},{vm_memory_high_watermark,0.4},{disk_free_limit,100000000},{log_levels,[{connection,info},{mirroring,info}]},{delegate_count,32},{tcp_listen
_options,[binary,{packet,raw},{reuseaddr,true},{backlog,128},{nodelay,true},{exit_on_close,false},{keepalive,true}]},{collect_statistics_interval,60000},{rab
bitmq_management_agent,[{force_fine_statistics,true}]}]},{kernel,[{net_ticktime,30}]}]." > /etc/rabbitmq/rabbitmq.config \
        && echo "rabbitmq-server -detached;rabbitmqctl stop_app;rabbitmqctl join_cluster rabbit@RABBITMQ_SEED_NODE} || true; rabbitmqctl start_app; rabbitmqc
tl set_policy cluster-all-queues '^(?!amq\.).*' '{"ha-mode":"all","ha-sync-mode":"automatic"}'"  > /usr/lib/rabbitmq/bin/rabbitmq-member \
        && chmod +x /usr/lib/rabbitmq/bin/rabbitmq-member

VOLUME /var/lib/rabbitmq

# add a symlink to the .erlang.cookie in /root so we can "docker exec rabbitmqctl ..."
RUN ln -sf /var/lib/rabbitmq/.erlang.cookie /root/

COPY docker-entrypoint.sh /
RUN  chmod +x /docker-entrypoint.sh
ENTRYPOINT ["/docker-entrypoint.sh"]

EXPOSE 5672 4369 15672
CMD ["rabbitmq-server"]
alfreds@sjc4dockerdev01:~/rabbitmq$
```

Example: A docker entry startup file.

```
alfreds@sjc4dockerdev01:~/rabbitmq$ cat docker-entrypoint.sh
#!/bin/bash
set -e
[[ -z "${RABBITMQ_PORT}" ]] && RABBITMQ_PORT=5672
[[ -z "${RABBITMQ_SEED_NODE}" ]] && RABBITMQ_SEED_NODE='localhost'
[[ -z "${RABBITMQ_MANAGEMEN_PORT}" ]] && RABBITMQ_MANAGEMENT_PORT=15672
[[ -z "${RABBITMQ_CLUSTER_NODES}" ]] && RABBITMQ_CLUSTER_NODES='rabbit@localhost'
[[ -z "${RABBITMQ_ERLANG_COOKIE}" ]] && RABBITMQ_ERLANG_COOKIE="MpRI2iDWBGJ6Y0Z23q9VNHeK"

if [ "${RABBITMQ_ERLANG_COOKIE}" ]; then
        cookieFile='/var/lib/rabbitmq/.erlang.cookie'
        echo "${RABBITMQ_ERLANG_COOKIE}" > "$cookieFile"
        chmod 600 "$cookieFile"
        chown rabbitmq "$cookieFile"
fi

sed -i "s/RABBITMQ_PORT/${RABBITMQ_PORT}/" /etc/rabbitmq/rabbitmq.config
sed -i "s/RABBITMQ_MANAGEMENT_PORT/${RABBITMQ_MANAGEMENT_PORT}/" /etc/rabbitmq/rabbitmq.config
sed -i "s/RABBITMQ_CLUSTER_NODES/${RABBITMQ_CLUSTER_NODES}/" /etc/rabbitmq/rabbitmq.config
sed -i "s/RABBITMQ_SEED_NODE/${RABBITMQ_SEED_NODE}/" /usr/lib/rabbitmq/bin/rabbitmq-member
if [ "${RABBITMQ_SEED}" ]; then
        exec "$@"
else
        myprog="/usr/lib/rabbitmq/bin/rabbitmq-member"
        exec "$myprog" "$@"
fi
alfreds@sjc4dockerdev01:~/rabbitmq$
```

2. Test the container image and push it to the local repository after done.

Example: Login to the local repo and push the container image to repo

```
# Display local Docker images
root@sjc4dockerdev01:~# docker images
REPOSITORY                              TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
rabbitmq-c31                            latest              3f0647ea738b        23 hours ago        374.3 MB

## Log in to the local Docker repository
docker login -u lab1 -p lab1 -e lab1@ge.com https://repo.vms.crd.ge.com:8080

# Tag the target image
#docker tag 3f0647ea738b repo.vms.crd.ge.com:8080/rabbitmq-c3:latest

# Push the image to the local repository
#docker push repo.vms.crd.ge.com:8080/rabbitmq-c3:latest

#Verify the imag eis on the local repository
#Docker search repo.vms.crd.ge.com:8080/rabbitmq
NAME                  DESCRIPTION   STARS     OFFICIAL   AUTOMATED
library/rabbitmq-c3                 0
```

3. Generate a json file based Marathon API v2. Provide environment parameters, port mapping details and other application specific configuration values.

Example: A json file to deploy rabbitmq via Marathon API

```
SF01212464633A:marathon 212464633$ cat rabbitmq-c3-0.json
{
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "repo.vms.crd.ge.com:8080/rabbitmq-c3:latest",
      "network": "BRIDGE",
      "hostname": "rcnode1",
      "name": "rc1",
      "portMappings": [
        { "containerPort":  14369, "hostPort": 0,  "servicePort":  14369, "protocol": "tcp" },
        { "containerPort":  5672, "hostPort": 0,  "servicePort":  5772, "protocol": "tcp" },
        { "containerPort": 15672, "hostPort": 0, "servicePort": 15772, "protocol": "tcp" }
      ]
    }
  },
  "id": "rabbitmq-c3-0",
  "instances": 1,
  "constraints": [["hostname", "UNIQUE"]],
  "env": { "RABBITMQ_SEED": "1", "RABBITMQ_CLUSTER_NODES": "'rabbit@rcnode1','rabbit@rcnode2','rabbit@rcnode3'", "ERL_EPMD_PORT": "14369" },
  "cpus": 0.5,
  "mem": 512,
  "uris": [
        "file:///etc/docker.tar.gz"
  ]
}
```

In this example, the rabbitmq service port is listening on <vip>:5772 and admin port is on <vip>:15572. The VIP is the virtual IP address of the load balancer. In case of clustering, the first node should use 0.0.0.0:14369 for EPMD service and sub-sequential nodes should connect to port <vip>:14369.

It's highly recommended to group multiple service into one json for a clustering service, particularly with dependency. Here is a generic example for illustrate how it works.

```
SF01212464633A:marathon 212464633$ cat prods2.json
{
  "id": "/prods2",
  "apps": [
    {
      "id": "service-common",
      "cpus": 0.1,
      "mem": 1,
      "ports": [0],
      "cmd": "python -m SimpleHTTPServer $PORT0",
      "instances": 6
    }
  ],
  "groups": [
     {
                "id": "/prods2/task1",
                "apps": [
                  {"id": "/prods2/task1/job1", "constraints": [["hostname", "UNIQUE"]], "instances": 6
,"cpus": 0.1,"mem": 1, "cmd": "for i in {1..3} ; do echo `date`: 'Hello Marathon/DCOS from job1' >> /t
mp/job1.out; sleep 30 ; done"},
                  {"id": "/prods2/task1/job2", "instances": 6,"cpus": 0.1,"mem": 1, "cmd": "for i in {
1..3} ; do echo `date`: 'Hello Marathon/DCOS from job2' >> /tmp/job2.out; sleep 30 ; done"}
                ]
     }, {
                "id": "/prods2/task2",
                 "dependencies": ["/prods2/task1"],
                "apps": [
                  { "id": "/prods2/task2/job11", "constraints": [["hostname", "UNIQUE"]], "instances":
 6,"cpus": 0.1,"mem": 1, "cmd": "[[ -f /tmp/job1.out ]] && echo `date`: `tail -1 /tmp/job1.out >> /tmp
/job11.out`"},
                  { "id": "/prods2/task2/job22", "instances": 6,"cpus": 0.1,"mem": 1, "cmd": "[[ -f /t
mp/job2.out ]] && echo `date`: `tail -1 /tmp/job2.out >> /tmp/job22.out`"}
                ]
        }
  ]
}
```

4. Call appropriate Marathon API to deploy the service. Obtain service entry URL and credentials if needed.

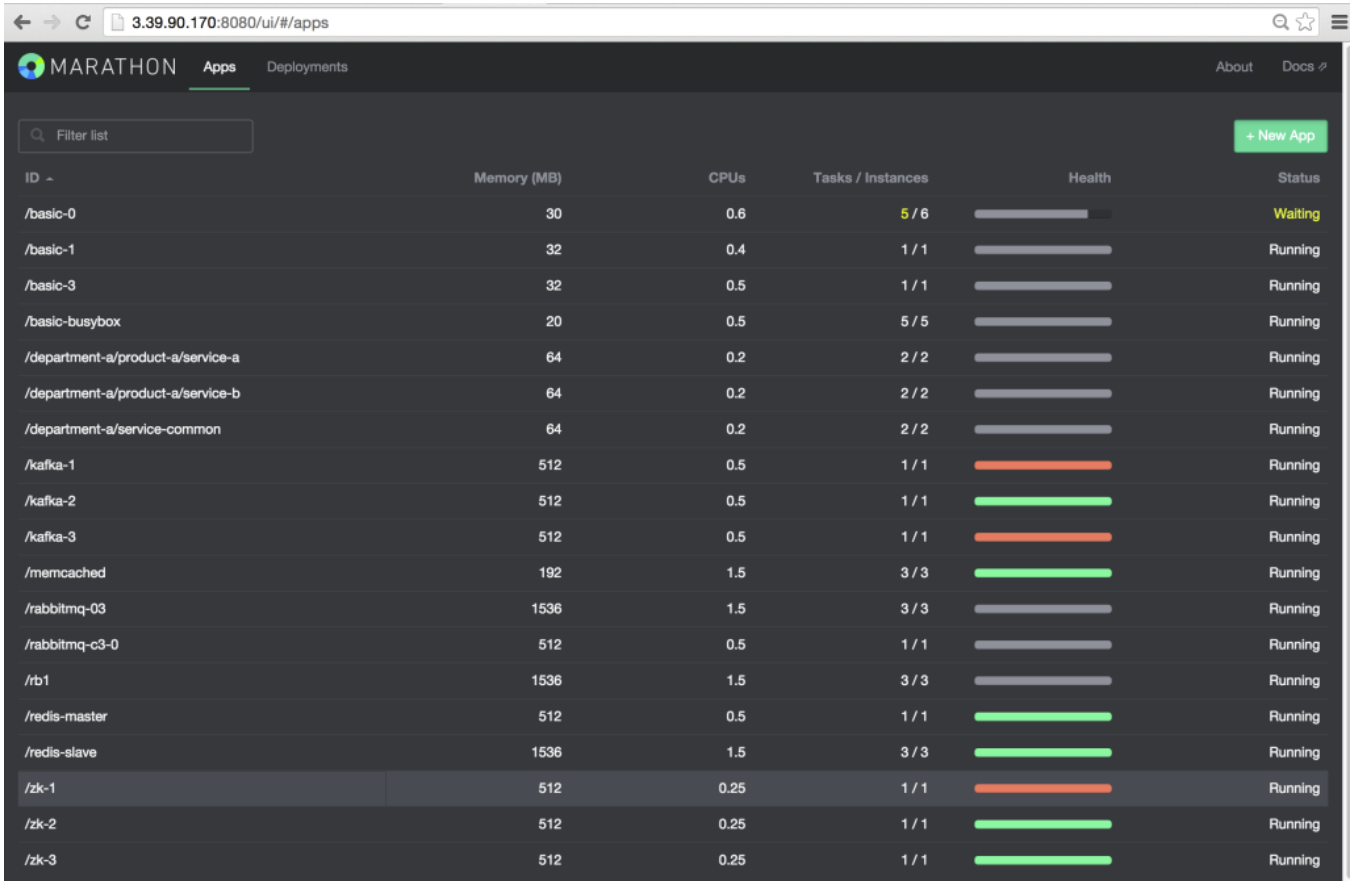
Example: To deploy a json file and discover the deployed service.

```
#!/bin/bash
function usage() {
cat <<EOF
usage: $0 options

Deploy applications on LAB using Marathon

Example:
        $0 -[l|a|d|h|g|G|D] [<app_name>]

OPTIONS:
        -h -- Display help
        -l -- List deployed applications
        -a -- Add a new application
        -d -- Delete an existing application
        -g -- List a group
        -G -- Add a new group
        -D -- Delete a group

EOF
}
while getopts "ha:lgd:G:D:" OPTION; do
case "$OPTION" in
l)
        curl  http://adminUser:topP1ssw0rd@3.39.90.170:8080/v2/apps | python -m json.tool
        ;;
h)
        usage
        exit 0
        ;;
a)
        curl -X POST  http://adminUser:topP1ssw0rd@3.39.90.170:8080/v2/apps -d @$2.json -H "Content-type:application/json"
        ;;
G)
        curl -X POST  http://adminUser:topP1ssw0rd@3.39.90.170:8080/v2/groups -d @$2.json -H "Content-type:application/json"
        ;;
g)
        [[ $# == 1 ]] && curl -X GET  http://adminUser:topP1ssw0rd@3.39.90.170:8080/v2/groups | python -m json.tool
        [[ $# == 2 ]] && curl -X GET  http://adminUser:topP1ssw0rd@3.39.90.170:8080/v2/groups/$2 | python -m json.tool
        ;;
d)
        curl -X DELETE  http://adminUser:topP1ssw0rd@3.39.90.170:8080/v2/apps/$2
        ;;
D)
        curl -X DELETE  http://adminUser:topP1ssw0rd@3.39.90.170:8080/v2/groups/$2
        ;;
\?)
        echo "Invalid option: -"$OPTARG"" >&2
        usage
        exit 1
        ;;
:)
        usage
        exit 1
        ;;
esac
done
```

...and to publish and subscribe from the deployed rabbitmq. All REST APIs will be SSL enabled and protected by password.

    #./lab.sh -a rabbitmq-c3-0

Here are a few running services in the Lab ecosystem.



5. Verify and test accessibility and performance of the deployed service.

   Example: To verify the deployed rabbitmq service. Test scripts are application specific and need to be written accordingly.

```
root@ctl3:~/bin# cat hello_pub.py
#!/usr/bin/python
from kombu import Connection
import datetime,getopt,sys

hostname='127.0.0.1'
port=5672
username='guest'
password='guest'
queue='test_queue'
message='hello world'

def usage(message=None):
  print "Usage: %s [-h] [-u|--username <user_name>] [-w|--password <password>] [-h|--host <rabbitmq_hostname>] [-p|--port <service_port>] [-q|--queue <queue_name>] [-m|--message <message>]" % (sys.argv[0])
  print "\t-h|--help: show this message"
  print "\t-v|--verbose: include details in output"
  print "\t-u|--username: Username"
  print "\t-w|--password: User password"
  print "\t-h|--hostname: Rabbitmq hostname or IP"
  print "\t-p|--port: Rabbitmq service port"
  print "\t-q|--queue: Rabbitmq queue name"
  print "t\t-m|--message: Message to br published"
  sys.exit(-1)

(opts, args) = getopt.getopt(sys.argv[1:], "?vu:w:h:p:q:m:", ["Rabbitmq test", "help", "verbose", "username", "password", "host", "port", "queue", "message"])

for o, a in opts:
  if o in ["-?", "--help"]:
        usage()
  elif o in ["-u", "--username"]:
        username=a
  elif o in ["-w", "--password"]:
        password=a
  elif o in ["-h", "--hostname"]:
        hostname=a
  elif o in ["-p", "--port"]:
        port=a
  elif o in ["-q", "--queue"]:
        queue=a
  elif o in ["-m", "--message"]:
        message=a
  elif o in ["-v", "--verbose"]:
        verbose=1
#import pdb;pdb.set_trace()
with Connection('amqp://'+username+':'+password+'@'+hostname+':'+port+'//') as conn:
    simple_queue = conn.SimpleQueue(queue)
    m_queue = '%s, sent at %s' % (message, datetime.datetime.today())
    simple_queue.put(m_queue)
    print('Sent: %s' % m_queue)
    simple_queue.close()
root@ctl3:~/bin# cat hello_con.py
#!/usr/bin/python
from kombu import Connection
import sys,os,getopt,re

def usage(message=None):
  print "Usage: %s [-h] [-u|--username <user_name>] [-w|--password <password>] [-h|--host <rabbitmq_hostname>] [-p|--port <service_port>] [-q|--queue <queue_name>]" % (sys.argv[0])
  print "-h|--help: show this message"
  print "-v|--verbose: include details in output"
  print "-u|--username: Username"
  print "-w|--password: User password"
  print "-h|--hostname: Rabbitmq hostname or IP"
  print "-p|--port: Rabbitmq service port"
  print "-q|--queue: Rabbitmq queue name"
  sys.exit(-1)

(opts, args) = getopt.getopt(sys.argv[1:], "?vu:w:h:p:q:", ["Rabbitmq test", "help", "verbose",  "username", "password", "host", "port", "queue", "message"])

for o, a in opts:
  if o in ["-?", "--help"]:
        usage()
  elif o in ["-u", "--username"]:
        username=a
  elif o in ["-w", "--password"]:
        password=a
  elif o in ["-h", "--host"]:
        host=a
  elif o in ["-p", "--port"]:
        port=a
  elif o in ["-q", "--queue"]:
        queue=a
  elif o in ["-v", "--verbose"]:
        verbose=1

with Connection('amqp://'+username+':'+password+'@'+host+':'+port+'//') as conn:
    simple_queue = conn.SimpleQueue(queue)
    message = simple_queue.get(block=True, timeout=1)
    print("Received: %s" % message.payload)
    message.ack()
    simple_queue.close()
```

And to verify the rabbitmq deployment by publishing to and subscribing from the deployed rabbitmq.

```
# Publish 50 messages to the queue "test_queue" with 5 concurrently
#seq 50 | xargs -n 5 -P5 -I{} ./hello_pub.py -u guest -w guest -h 10.11.1.134 -p 5772 -q test_queue -m "from ctl3 number: {}"


#Subscribe the "test_queue" for 50 times with 5 concurrently
#seq 5 | xargs -n 5 -P5 -I{} ./hello_con.py -u guest -w guest -h 10.11.1.134 -p 5772 -q test_queue
```