**Implementing Highly Available RabbitMQ Cluster on Docker Swarm using Consul-based Discovery**

[[](https://medium.com/@ahmetvehbiolgac?source=post_page-----45c4e7919634--------------------------------)](https://medium.com/@ahmetvehbiolgac?source=post_page-----45c4e7919634--------------------------------)

[Ahmet Vehbi Olgaç](https://medium.com/@ahmetvehbiolgac?source=post_page-----45c4e7919634--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F8e95cebf8a3f&operation=register&redirect=https%3A%2F%2Fmedium.com%2F%40ahmetvehbiolgac%2Fimplementing-highly-available-rabbitmq-cluster-on-docker-swarm-using-consul-based-discovery-45c4e7919634&user=Ahmet+Vehbi+Olga%C3%A7&userId=8e95cebf8a3f&source=post_page-8e95cebf8a3f----45c4e7919634---------------------post_header-----------)

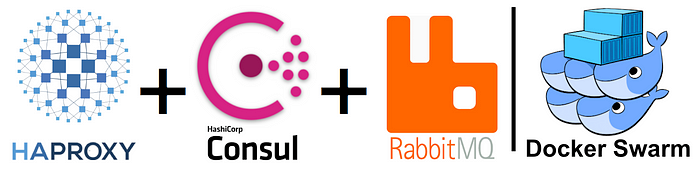
5 min read

·

Dec 26, 2018

318

12

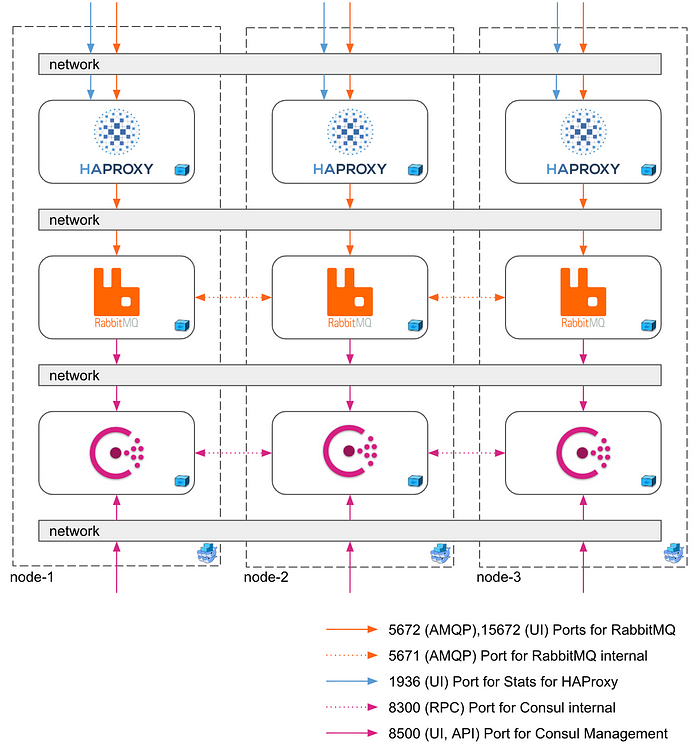


Recently, we’ve re-written *[Hepsiburada](https://www.hepsiburada.com/" \t "_blank) BuyBox* Application, which is receiving high-traffic and concurrent access, in Golang and re-platformed the whole system, including the [RabbitMQ](https://www.rabbitmq.com/) cluster, to [Docker Swarm](https://docs.docker.com/engine/swarm/). After the successful roll out, we applied the same solution to our *Marketplace Order Management System*, as well. We used this infrastructure during this year’s Black Friday, and had zero problems.

In this article, I’ll try to explain how we setup the RabbitMQ cluster inside Docker Swarm, drill down the technical challenges and the solutions we’ve provided to them.

Deploying cluster-based applications (such as RabbitMQ, MongoDB, Elasticsearch), to a container-orchestrating environment, like Docker Swarm, has its unique challenges. There are several [ways](https://www.rabbitmq.com/clustering.html#cluster-formation-options) to set up a RabbitMQ Cluster. But, we will focus on [Consul-based RabbitMQ clustering](https://www.rabbitmq.com/cluster-formation.html#peer-discovery-consul). We will also add [HAProxy](http://www.haproxy.org/" \t "_blank) farm to load balance AMQP requests, and to increase the overall availability of the cluster.

In the below diagram, you can see the final logical topology.



So, Let’s get started! 👊

**Step 1: Install Docker and Setup Docker Swarm Cluster**

**Installing Docker**

curl -fsSL <https://get.docker.com/> | sh#Check Version  
docker -v#Expected Output  
Docker version 18.09.0, build 4d60db4

**Initializing Docker Swarm Cluster**

#Init Docker Swarm - run on node-1  
docker swarm init --advertise-addr 192.168.0.1#Join Cluster - run on node-2, node-3  
docker swarm join --token SWMTKN-1-dfhjdsgfjkhuiy 192.168.0.1:2377#Promote nodes as manager  
docker node promote node-1  
docker node promote node-2#Check nodes  
docker node ls#Expected Output  
ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS ENGINE VERSION  
te0620 \* node1 Ready Active Leader 18.09.0  
j0sbfn node2 Ready Active Reachable 18.09.0  
ty9hrq node3 Ready Active Reachable 18.09.0#Create a overlay network  
docker network create --driver=overlay --attachable prod#Check network  
docker network ls#Expected Output  
NETWORK ID NAME DRIVER SCOPE  
140490c6297a bridge bridge local  
631ac138c4f5 docker\_gwbridge bridge local  
a28dc1c47bac host host local  
awn0wd58g0nw ingress overlay swarm  
a19559fae5bb none null local  
we17lyjmbpem **prod** overlay swarm

**Step 2: Deploy Consul Cluster**

**Important notes:**

* Map consul-data to volume, in order to not lose the data.
* Deploy the Consul service inglobal mode, and manage service-scheduling via node labels
* *Use two separate networks, one for internal consul traffic, and one for the traffic between RabbitMQ and the Consul services.*

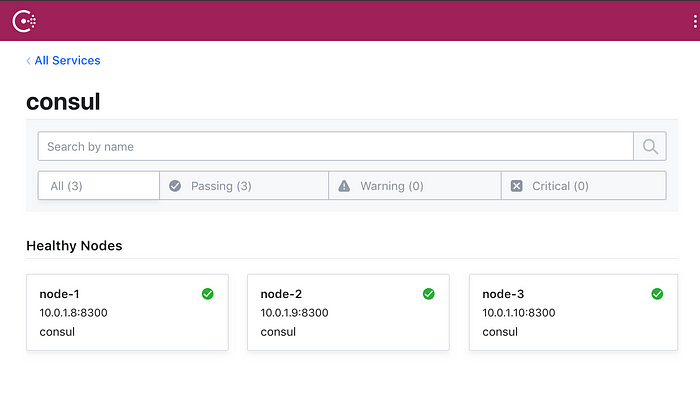
[**docker-compose.yml**](https://github.com/olgac/consul/blob/master/docker-compose.yml)

version: '3.6'  
services:  
 consul:  
 image: consul:1.4.0  
 hostname: "{{.Node.Hostname}}"  
 networks:  
 - consul  
 - prod  
 ports:  
 - 8400:8400  
 - 8500:8500  
 - 8600:53  
 volumes:  
 - consul-data:/consul/data  
 deploy:  
 mode: global  
 placement:  
 constraints: [node.labels.consul == true]  
 command: [ "agent", "-server", "-bootstrap-expect=3", "-retry-max=3", "-retry-interval=10s", "-datacenter=prod", "-join=consul", "-retry-join=consul", "-bind={{ GetInterfaceIP \"eth0\" }}", "-client=0.0.0.0", "-ui"]  
networks:  
 consul:  
 prod:  
 external: true  
volumes:  
 consul-data:

**Initialization script**

wget <https://raw.githubusercontent.com/olgac/consul/master/docker-compose.yml>  
docker node update --label-add consul=true node-1  
docker stack deploy -c docker-compose.yml consul#Wait to electing as Consul Leader  
sleep 10#Then, wake-up Consul on node-2 and node-3  
docker node update --label-add consul=true node-2  
docker node update --label-add consul=true node-3#Check Consul Leader  
curl 192.168.0.1:8500/v1/status/leader#Expected Output  
"10.0.1.9:8300"#Check Consul Peers  
curl 192.168.0.1:8500/v1/status/peers#Expected Output  
[  
 "10.0.1.8:8300",  
 "10.0.1.9:8300",  
 "10.0.1.10:8300"  
]

Navigate to [http://192.168.0.1:8500](http://192.168.0.1:8500/), to see the **Consul UI**and verify the installation.



**Step 3: Deploy RabbitMQ Cluster**

**Important Notes:**

* Volume mapping to rabbitmq-data to protect persistent data.
* Deploy RabbitMQ service inglobal mode, and manage service-scheduling via node labels
* Use one network (prod) for the internal/external RabbitMQ traffic
* Don’t share your RABBITMQ\_ERLANG\_COOKIE and RABBITMQ\_DEFAULT\_PASS
* *Hostname is important, since RabbitMQ uses hostnames as data directories*

[**docker-compose.yml**](https://github.com/olgac/rabbitmq/blob/master/docker-compose.yml)

version: "3.6"  
services:  
 rabbitmq-01:  
 image: olgac/rabbitmq:3.7.8-management  
 hostname: rabbitmq-01  
 environment:  
 - RABBITMQ\_DEFAULT\_USER=admin  
 - RABBITMQ\_DEFAULT\_PASS=Passw0rd  
 - RABBITMQ\_ERLANG\_COOKIE="MY-SECRET-KEY-123"  
 networks:  
 - prod  
 volumes:  
 - rabbitmq-01-data:/var/lib/rabbitmq  
 deploy:  
 mode: global  
 placement:  
 constraints: [node.labels.rabbitmq1 == true]rabbitmq-02:  
 image: olgac/rabbitmq:3.7.8-management  
 hostname: rabbitmq-02  
 environment:  
 - RABBITMQ\_DEFAULT\_USER=admin  
 - RABBITMQ\_DEFAULT\_PASS=Passw0rd  
 - RABBITMQ\_ERLANG\_COOKIE="MY-SECRET-KEY-123"  
 networks:  
 - prod  
 volumes:  
 - rabbitmq-02-data:/var/lib/rabbitmq  
 deploy:  
 mode: global  
 placement:  
 constraints: [node.labels.rabbitmq2 == true]rabbitmq-03:  
 image: olgac/rabbitmq:3.7.8-management  
 hostname: rabbitmq-03  
 environment:  
 - RABBITMQ\_DEFAULT\_USER=admin  
 - RABBITMQ\_DEFAULT\_PASS=Passw0rd  
 - RABBITMQ\_ERLANG\_COOKIE="MY-SECRET-KEY-123"  
 networks:  
 - prod  
 volumes:  
 - rabbitmq-03-data:/var/lib/rabbitmq  
 deploy:  
 mode: global  
 placement:  
 constraints: [node.labels.rabbitmq3 == true]networks:  
 prod:  
 external: true  
volumes:  
 rabbitmq-01-data:  
 rabbitmq-02-data:  
 rabbitmq-03-data:

[**config/enabled\_plugins**](https://github.com/olgac/rabbitmq/blob/master/config/enabled_plugins)

[rabbitmq\_management,  
rabbitmq\_peer\_discovery\_consul,  
rabbitmq\_federation,  
rabbitmq\_federation\_management,  
rabbitmq\_shovel,  
rabbitmq\_shovel\_management].

[**config/rabbitmq.conf**](https://github.com/olgac/rabbitmq/blob/master/config/rabbitmq.conf)**(by**[**RabbitMQ Configuration Guide**](https://www.rabbitmq.com/configure.html)**)**

loopback\_users.admin = false  
cluster\_formation.peer\_discovery\_backend = rabbit\_peer\_discovery\_consul  
cluster\_formation.consul.host = consul  
cluster\_formation.node\_cleanup.only\_log\_warning = true  
cluster\_formation.consul.svc\_addr\_auto = true  
cluster\_partition\_handling = autoheal#Flow Control is triggered if memory usage above %80.  
vm\_memory\_high\_watermark.relative = 0.8#Flow Control is triggered if free disk size below 5GB.  
disk\_free\_limit.absolute = 5GB

**Initialization script**

wget <https://raw.githubusercontent.com/olgac/rabbitmq/master/docker-compose.yml>  
docker node update --label-add rabbitmq1=true node-1  
docker node update --label-add rabbitmq2=true node-2  
docker node update --label-add rabbitmq3=true node-3  
docker stack deploy -c docker-compose.yml rabbitmq

**Additional Note**

We prefer to reference docker images stored in our private registry instead of the public Docker Hub. You can take the following steps to add the config files to your RabbitMQ image and store it in your own registry.

git clone <https://github.com/olgac/rabbitmq.git>  
cd rabbitmqdocker login  
docker image build -t <your-registry>/rabbitmq:3.7.8-management .  
docker push <your-registry>/rabbitmq:3.7.8-management

You can write your own docker registry before docker image name and use this image in the docker-compose.yml lines 4, 19 and 36.

**Step 4: Expose RabbitMQ with Deploy HAProxy**

[**docker-compose.yml**](https://github.com/olgac/rabbitmq/blob/master/docker-compose.yml)

version: "3.6"  
services:  
 haproxy:  
 image: olgac/haproxy-for-rabbitmq:1.8.14-alpine  
 ports:  
 - 15672:15672  
 - 5672:5672  
 - 1936:1936  
 networks:  
 - prod  
 deploy:  
 mode: global  
networks:  
 prod:  
 external: true

[**config/haproxy.cfg**](https://github.com/olgac/haproxy-for-rabbitmq/blob/master/config/haproxy.cfg)

global  
 log 127.0.0.1 local0  
 log 127.0.0.1 local1 notice  
 maxconn 4096  
   
defaults  
 log global  
 option tcplog  
 option dontlognull  
 timeout connect 6s  
 timeout client 60s  
 timeout server 60s  
   
listen stats  
 bind \*:1936  
 mode http  
 stats enable  
 stats hide-version  
 stats realm Haproxy\ Statistics  
 stats uri /listen rabbitmq  
 bind \*:5672  
 mode **tcp**  
 server rabbitmq-01 rabbitmq-01:5672 check  
 server rabbitmq-02 rabbitmq-02:5672 check  
 server rabbitmq-03 rabbitmq-03:5672 checklisten rabbitmq-ui  
 bind \*:15672  
 mode **http**  
 server rabbitmq-01 rabbitmq-01:15672 check  
 server rabbitmq-02 rabbitmq-02:15672 check  
 server rabbitmq-03 rabbitmq-03:15672 check

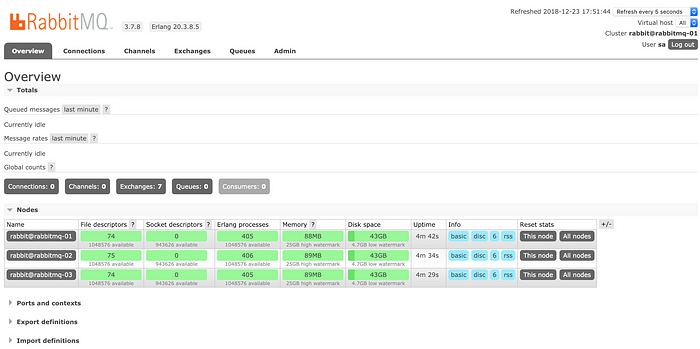
**Additional Note**

Similar to RabbitMQ image, you can add the HAProxy configs to your image, and publish it to your own private registry

git clone <https://github.com/olgac/haproxy-for-rabbitmq.git>  
cd [haproxy-for-rabbitmq](https://github.com/olgac/haproxy-for-rabbitmq.git)docker login  
docker image build -t <your-registry>/haproxy-for-rabbitmq:1.8.14-alpine .  
docker push <your-registry>/haproxy-for-rabbitmq:1.8.14-alpine

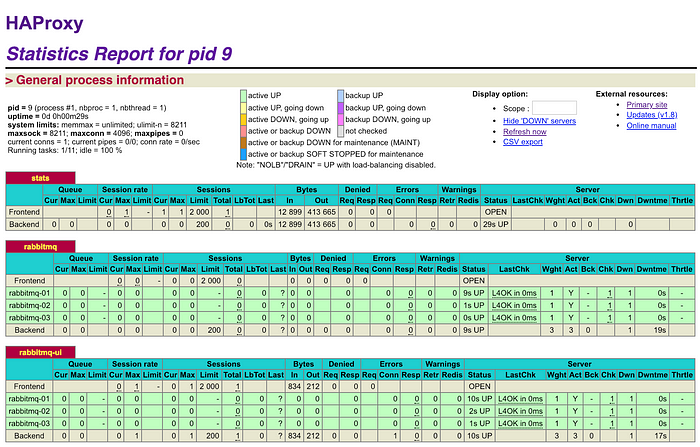
**Final Step: Verify your RabbitMQ Cluster**

Navigate to [http://192.168.0.1:15672](http://192.168.0.1:15672/), and enter the predefined credentials (sa and Passw0rd in this tutorial), then you should see the **RabbitMQ Management UI**.



And, we should see all HAProxy backend services UP and Running.

* Check [http://192.168.0.1:1936](http://192.168.0.1:1936/)



Enjoy!!