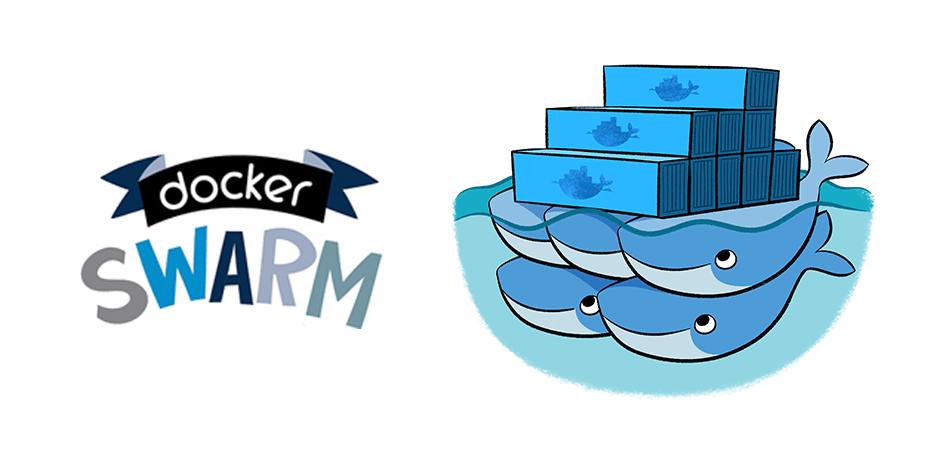
Docker Swarm для самых маленьких

**13 мин**

**106K**

[Системное администрирование\*](https://habr.com/ru/hubs/sys_admin/)[DevOps\*](https://habr.com/ru/hubs/devops/)[Микросервисы\*](https://habr.com/ru/hubs/microservices/)

Туториал



Данная статья посвящена настройке и работе с Docker Swarm.

Swarm это стандартный оркестратор для docker контейнеров, доступный из «коробки», если у вас установлен сам docker.

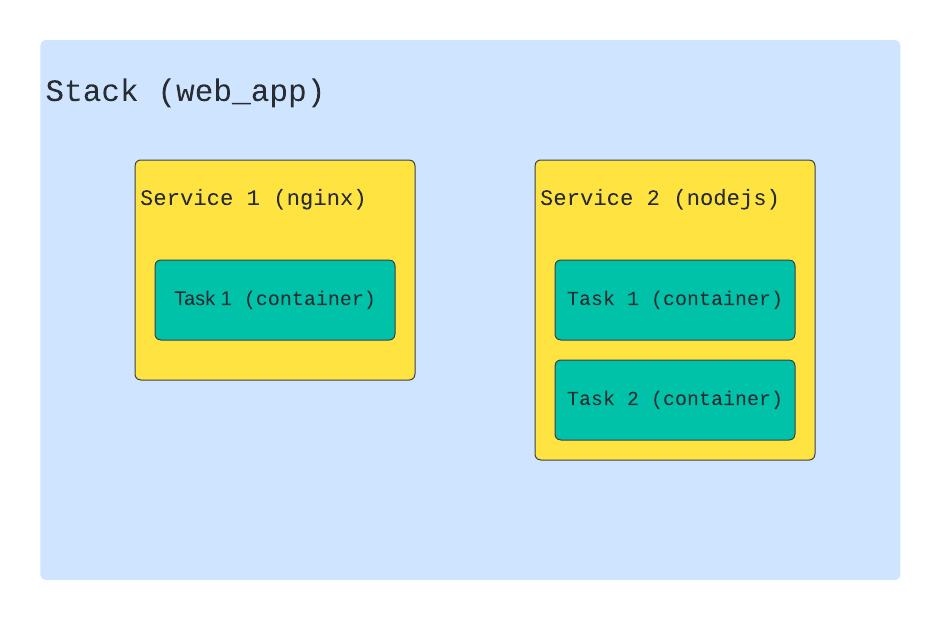
Что нам потребуется для освоения:

* Иметь опыт работы с docker и docker compose.
* Настроенный docker registry. Swarm не очень любит работать с локальными образами.
* Несколько виртуальных машин для создания кластера, хотя по факту кластер может состоять из одной виртуалки, но так будет нагляднее.

Термины

Для того чтобы пользоваться swarm надо запомнить несколько типов сущностей:

* **Node** - это наши виртуальные машины, на которых установлен docker. Есть manager и workers ноды. Manager нода управляет workers нодами. Она отвечает за создание/обновление/удаление сервисов на workers, а также за их масштабирование и поддержку в требуемом состоянии. Workers ноды используются только для выполнения поставленных задач и не могут управлять кластером.
* **Stack** - это набор сервисов, которые логически связаны между собой. По сути это набор сервисов, которые мы описываем в обычном compose файле. Части stack (services) могут располагаться как на одной ноде, так и на разных.
* **Service**- это как раз то, из чего состоит stack. Service является описанием того, какие контейнеры будут создаваться. Если вы пользовались docker-compose.yaml, то уже знакомы с этой сущностью. Кроме стандартных полей docker в режиме swarm поддерживает ряд дополнительных, большинство из которых находятся внутри секции [deploy](https://docs.docker.com/compose/compose-file/compose-file-v3/" \l "deploy).
* **Task** - это непосредственно созданный контейнер, который docker создал на основе той информации, которую мы указали при описании service. Swarm будет следить за состоянием контейнера и при необходимости его перезапускать или перемещать на другую ноду.



Создание кластера

Для того чтобы кластер корректно работал, необходимо открыть следующие порты на виртуальных машинах. Для manager node:

firewall-cmd --add-port=2376/tcp --permanent;  
firewall-cmd --add-port=2377/tcp --permanent;  
firewall-cmd --add-port=7946/tcp --permanent;  
firewall-cmd --add-port=7946/udp --permanent;  
firewall-cmd --add-port=4789/udp --permanent;  
firewall-cmd --reload;  
  
systemctl restart docker;

Для worker node

firewall-cmd --add-port=2376/tcp --permanent;  
firewall-cmd --add-port=7946/tcp --permanent;  
firewall-cmd --add-port=7946/udp --permanent;  
firewall-cmd --add-port=4789/udp --permanent;  
firewall-cmd --reload;  
  
systemctl restart docker;

Затем заходим на виртуальную машину, которая будет у нас manager node. И выполняем следующую команду:

docker swarm init

Если все успешно, то в ответ вы получите следующую команду:

docker swarm join --token SWMTKN-1-54k2k418tw2j0juwm3inq6crp4ow6xogswihcc5azg7oq5qo7e-a3rfeyfwo7d93heq0y5vhyzod 172.31.245.104:2377

Ее будет необходимо выполнить на всех worker node, чтобы присоединить их в только что созданный кластер.

Если все прошло успешно, выполнив следующую команду на manager ноде в консоли, вы увидите что-то подобное:

docker node ls  
ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS ENGINE VERSION  
vj7kp5847rh5mbbqn97ghyb72 \* dev-2 Ready Active Leader 20.10.14  
zxo15m9wqdjd9f8pvg4gg6gwi stage Ready Active 20.10.9

У меня виртуалка с hostname dev-2 является manager, а stage является worker нодой.

В принципе мы готовы к тому, чтобы запускать services и stacks на нашей worker node.

Если мы хотим убрать ноду из кластера, необходимо зайти на виртуалку, которая является ею, и выполнить команду:

docker swarm leave

Если затем зайти на manager ноду и выполнить docker node ls, вы заметите, что статус у нее поменялся c **Ready** на **Down**(это может занять некоторое время). Swarm больше не будет использовать данную ноду для размещения контейнеров, и вы можете спокойно заняться техническими работами, не боясь нанести вред работающим контейнерам. Для того чтобы окончательно удалить ноду, надо выполнить (на manager node):

docker node rm stage

Стэк и Сервис

Для создания нашего стэка я возьму в качестве примера compose файл для node js web server, который прослушивает порт **4003**:

# docker-compose.stage.yaml  
  
version: "3.9"  
  
services:  
 back:  
 image: docker-registry.ru:5000/ptm:stage  
 ports:  
 - "4003:4003"  
 environment:  
 TZ: "Europe/Moscow"  
 extra\_hosts:  
 - host.docker.internal:host-gateway  
 command: make server\_start  
 volumes:  
 - /p/ptm/config/config.yaml:/p/ptm/config/config.yaml  
 - /p/ptm/stat/web:/p/ptm/stat/web

В начале необходимо достать image из registry и только затем задеплоить в наш кластер:

docker pull docker-registry.ru:5000/ptm:stage;  
docker stack deploy --with-registry-auth -c ./docker-compose.stage.yaml stage;

Все эти команды надо выполнять на manager node. Опция --with-registry-auth позволяет передать авторизационные данные на worker ноды, для того чтобы использовался один и тот же образ из регистра. stage это имя нашего стэка.

Посмотреть список стэков можно с помощью:

docker stack ls  
NAME SERVICES ORCHESTRATOR  
stage 1 Swarm

Список сервисов внутри стэка:

docker stack services stage  
ID NAME MODE REPLICAS IMAGE PORTS  
qq8spne5gik7 stage\_back replicated 1/1 docker-registry.ru:5000/ptm:stage \*:4003->4003/tcp

Подробная информация о сервисе:

docker service ps --no-trunc stage\_back  
ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS  
qok4z2rbl9v8238phy3lxgnw9 stage\_back.1 docker-registry.ru:5000/ptm:stage@sha256:8f5dca792c4e8f1bf94cb915505839a73eb50f38540e7d6d537a305594e51cae stage Running Running 1 minute ago   
khctf0lqbk5l9de81ifwuzwer \\_ stage\_back.1 docker-registry.ru:5000/ptm:stage@sha256:b4ff5e2fbb0daba7985122098d45a5873f1991fd8734a3a89f5affa5daf96e43 stage Shutdown Shutdown 40 hours ago

Тут видно, что для этого сервиса уже запускался контейнер (40 часов назад).

Для того чтобы увидеть более подробную информацию по сервису в виде JSON:

docker service inspect stage\_back  
   
 [  
 {  
 "ID": "qq8spne5gik7lqisqwo6u7ybr",  
 "Version": {  
 "Index": 24526  
 },  
 "CreatedAt": "2022-04-06T18:37:15.002080284Z",  
 "UpdatedAt": "2022-04-06T18:51:09.9827704Z",  
 .......  
 "Endpoint": {  
 "Spec": {  
 "Mode": "vip",  
 "Ports": [  
 {  
 "Protocol": "tcp",  
 "TargetPort": 4003,  
 "PublishedPort": 4003,  
 "PublishMode": "ingress"  
 }  
 ]  
 },  
 "Ports": [  
 {  
 "Protocol": "tcp",  
 "TargetPort": 4003,  
 "PublishedPort": 4003,  
 "PublishMode": "ingress"  
 }  
 ],  
 "VirtualIPs": [  
 {  
 "NetworkID": "obmqmpiujnamih7k76q87c058",  
 "Addr": "10.0.0.61/24"  
 },  
 {  
 "NetworkID": "fufdtvhekbyhpfvb71zosei0k",  
 "Addr": "10.0.60.2/24"  
 }  
 ]  
 },  
 "UpdateStatus": {  
 "State": "completed",  
 "StartedAt": "2022-04-06T18:46:37.03796414Z",  
 "CompletedAt": "2022-04-06T18:51:09.982709527Z",  
 "Message": "update completed"  
 }  
 }  
]

Тут также можно увидеть, какие порты слушает сервис, в каких сетях участвует, какой у него статус и много чего еще.

Удалить стэк можно следующим образом:

docker stack rm stage

Можно запускать и отдельно взятые сервисы, например:

docker service create --name nginx --replicas 3 nginx:alpine;  
docker service ps nginx;  
ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS  
awn4xed5twz3 nginx.1 nginx:alpine stage Running Running 24 minutes ago   
m42te1ahbi3q nginx.2 nginx:alpine prod-1 Running Running 20 minutes ago   
otkio3yfcw7o nginx.3 nginx:alpine dev-2 Running Running 16 minutes ago

В данном примере мы запустили сервис nginx в виде 3 экземпляров, которые swarm раскидал по 3 нодам.

Удалить сервис можно следующим образом:

docker service rm nginx

Label

Swarm по умолчанию развертывает сервисы на любой доступной ноде/нодах, но как правило нам необходимо развертывать их на конкретной ноде или на специфической группе. И тут нам как раз приходят на помощь labels.

Например, у нас есть stage и prod окружение. stage используется для внутренней демонстрации продукта, а prod как можно догадаться из названия, является непосредственно продакшеном.

Для каждого из окружений у нас есть compose файл: docker-compose.stage.yaml и docker-compose.prod.yaml. По умолчанию swarm будет раскидывать service произвольно по нодам. А нам бы хотелось, чтобы сервис для stage запускался только на stage виртуалке и аналогично для prod.

В начале, добавим еще одну ноду в кластер, в качестве worker:

# заходим по ssh на еще одну виртуальную машину и добавляем ее в кластер  
docker swarm join --token SWMTKN-1-54k2k418tw2j0juwm3inq6crp4ow6xogswihcc5azg7oq5qo7e-a3rfeyfwo7d93heq0y5vhyzod  
  
# выполняем команду на manager ноде:  
docker node ls  
ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS ENGINE VERSION  
vj7kp5847rh5mbbqn97ghyb72 \* dev-2 Ready Active Leader 20.10.14  
k3wjrl2o827in7k55wqfjfyxs prod-1 Ready Active 20.10.14  
zxo15m9wqdjd9f8pvg4gg6gwi stage Ready Active 20.10.9

Затем необходимо разметить наши ноды:

docker node update --label-add TAG=stage stage  
docker node update --label-add TAG=prod prod-1

Используя hostname виртуалок, мы навешиваем label. Для того чтобы убедиться, что все прошло успешно, необходимо выполнить следующую команду:

docker node inspect stage

Ищем раздел Spec.Labels, где мы должны увидеть label, который добавили:

"Spec": {  
 "Labels": {  
 "TAG": "stage"  
 },  
 "Role": "worker",  
 "Availability": "active"  
},

После чего в наш compose файл необходимо добавить директиву placement, где прописывается условие, которое указывает, на каких нодах разворачивать данный сервис:

# docker-compose.stage.yaml  
  
version: "3.9"  
  
services:  
 back:  
 image: docker-registry.ru:5000/ptm:stage  
 ports:  
 - "4003:4003"  
 environment:  
 TZ: "Europe/Moscow"  
 extra\_hosts:  
 - host.docker.internal:host-gateway  
 command: make server\_start  
 volumes:  
 - /p/ptm/config/config.yaml:/p/ptm/config/config.yaml  
 - /p/ptm/stat/web:/p/ptm/stat/web  
 # swarm  
 deploy:  
 placement:  
 constraints:  
 - "node.labels.TAG==stage"

Для docker-compose.prod.yaml будет аналогично, но с тэгом prod (однако для внешнего мира надо использовать другой порт, например **4004**). После деплоя данных stacks вы убедитесь, что сервисы разворачиваются только на нодах с определенным тэгом.

Маршрутизация

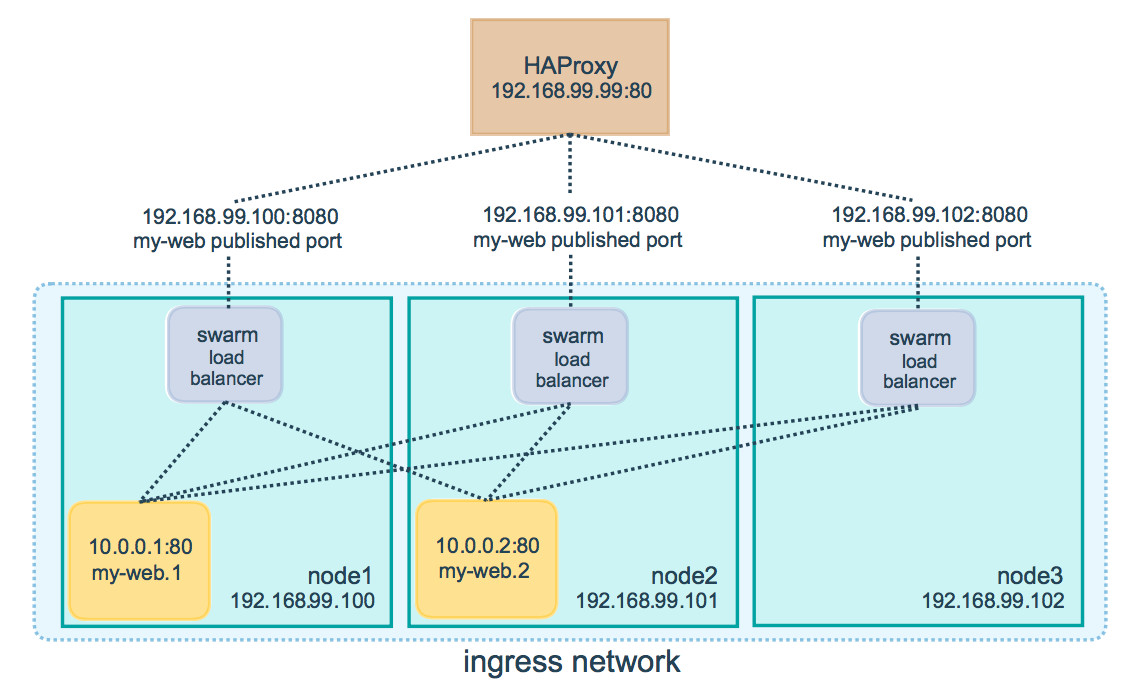
В данный момент у нас 3 ноды: manager нода, нода для stage версии приложения и еще одна для продакшена.

docker node ls  
ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS ENGINE VERSION  
vj7kp5847rh5mbbqn97ghyb72 \* dev-2 Ready Active Leader 20.10.14  
k3wjrl2o827in7k55wqfjfyxs prod-1 Ready Active 20.10.14  
zxo15m9wqdjd9f8pvg4gg6gwi stage Ready Active

И если мы попытаемся задеплоить наш стэк для docker-compose.prod.yaml на том же **4003** порту, что и для уже запущенного стэка docker-compose.stage.yaml, мы получим ошибку, связанную с тем, что порт уже занят.  
  
Хммм... почему это произошло?🤔 И более того, если мы зайдем на виртуальную машину prod-1 и сделаем curl 127.0.0.1:4003, то увидим, что наш сервис доступен, хотя на этой ноде мы еще не успели ничего развернуть?



Связано это с тем, что у swarm имеется ingress сеть, которая используется для маршрутизации траффика между нодами. Если у сервиса есть публичный порт, то swarm слушает его и в случае поступления запроса на него делает следующее: определяет есть ли контейнер на этой хост машине и если нет, то находит ноду, на которой запущен контейнер для данного сервиса, и перенаправляет запрос туда.



В данном примере используется внешний балансировщик HAProxy, который балансирует запросы между тремя виртуалками, а дальше swarm перенаправляет запросы в соответствующие контейнеры.

Вот почему придется для docker-compose.prod.yaml использовать любой другой публичный порт, отличный от того, который мы указали в docker-compose.stage.yaml.

Отключить автоматическую маршрутизацию трафика можно с помощью mode: host при декларации ports:

# docker-compose.prod.yaml  
version: "3.9"  
  
services:  
 back:  
 image: docker-registry.ru:5000/ptm:stage  
 ports:  
 - target: 4003  
 published: 4004  
 protocol: tcp  
 mode: host  
 environment:  
 TZ: "Europe/Moscow"  
 extra\_hosts:  
 - host.docker.internal:host-gateway  
 command: make server\_start  
 volumes:  
 - /p/ptm/config/config.yaml:/p/ptm/config/config.yaml  
 - /p/ptm/stat/web:/p/ptm/stat/web  
 # swarm  
 deploy:  
 placement:  
 constraints:  
 - "node.labels.TAG==prod"

В данном случае запрос, который придет на порт **4004,** swarm будет направлять только на контейнер текущей ноде и никуда больше.

Кроме того, надо упомянуть про такую настройку как mode, хотя напрямую это не относится к маршрутизации. Она может принимать следующие значения: global или replicated (по умолчанию):

deploy:  
 mode: global  
  
deploy:  
 mode: replicated  
 replicas: 4

Если global это означает, что данный сервис будет запущен ровно в одном экземпляре на всех возможных нодах. А replicated означает, что **n**-ое кол-во контейнеров для данного сервиса будет запущено на всех доступных нодах.

Кроме того, советую почитать [следующую статью](https://habr.com/ru/post/334004/) про то, как устроены сети в swarm.

Zero downtime deployment

Одна из очень полезных фишек, которая есть из коробки, это возможность организовать бесшовную смену контейнеров во время деплоя. Да, с docker-compose это тоже возможно, но надо писать обвязку на bash/ansible или держать дополнительную реплику контейнера (даже, если по нагрузке требуется всего одна) и на уровне балансировщика переключать трафик с одного контейнера на другой.

С docker swarm это все не нужно, необходимо лишь немного скорректировать конфиг сервиса.

Для начала нужно добавить директивуhealthcheck:

healthcheck:  
 test: curl -sS http://127.0.0.1:4004/ptm/api/healthcheck || echo 1  
 interval: 30s  
 timeout: 3s  
 retries: 12

Она предназначена, чтобы docker мог определить, корректно ли работает ваш контейнер.

* test - результат исполнения этой команды docker использует для определения корректно ли работает контейнер.
* interval - с какой частотой проверять состояние. В данном случае каждые 30 секунд.
* timeout - таймаут для ожидания выполнения команды.
* retries - кол-во попыток для проверки состояния нашего сервера внутри контейнера.

После добавления данной директивы в compose file мы увидим в колонке **Status** информацию о жизненном состоянии контейнера.

docker container ls  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
b66aa4c93fda docker-registry.ru:5000/ptm:stage "docker-entrypoint.s…" 2 days ago Up 2 days (healthy) 4003/tcp prod\_back.1.6x33tl8sj9t5ddlbbm2rrxbrn

После того как вы сделаете deploy вашему стэку, вы увидите, что контейнер сначала имеет статус **starting** (сервер внутри нашего контейнера запускается), а через некоторое время получит статус **healthy** (сервер запустился), в противном случае **unhealthy**. Docker в режиме swarm не просто отслеживает жизненное состояние контейнера, но и в случае перехода в состояние **unhealthy** попытается пересоздать контейнер.

После того, как мы научили docker следить за жизненным состоянием контейнера необходимо добавить настройки для обновления контейнера:

deploy:  
 replicas: 1  
 update\_config:  
 parallelism: 1  
 order: start-first  
 failure\_action: rollback  
 delay: 10s

replicas*это*кол-во контейнеров, которые необходимо запустить для данного сервиса.

Директива update\_configописывает каким образом сервис должен обновляться:

* parallelism- кол-во контейнеров для одновременного обновления. По умолчанию данный параметр имеет значение **1**- контейнеры будут обновляться по одному. **0**- обновить сразу все контейнеры. В большинстве случаев это число должно быть меньше, чем общее кол-во реплик вашего сервиса.
* order - Порядок обновления контейнеров. По умолчанию stop-first, сначала текущий контейнер останавливается, а затем запускается новый. Для бесшовного обновления нам нужно использовать start-first это когда вначале запускается новый контейнер, а затем выключается старый.
* failure\_action- стратегия в случае сбоя. Вариантов несколько:  continue, rollback, или pause (по умолчанию).
* delay - задержка между обновлением группы контейнеров.

Кроме того, полезна директиваrollback\_config, которая описывает поведение в случае сбоя во время обновления, а также директива restart\_policy*,*которая описывает когда и как перезапускать контейнеры в случае проблем.

deploy:  
 replicas: 1  
 update\_config:  
 parallelism: 1  
 order: start-first  
 failure\_action: rollback  
 delay: 10s  
 rollback\_config:  
 parallelism: 0  
 order: stop-first  
 restart\_policy:  
 condition: any  
 delay: 5s  
 max\_attempts: 3  
 window: 120s

* condition- есть несколько возможных вариантов none, on-failure or any.
* delay - как долго ждать между попытками перезапуска.
* max\_attempts- максимальное кол-во попыток для перезапуска.
* window- как долго ждать прежде, чем определить, что рестарт удался.

Итоговый docker-compose.yml

Для обновления нашего сервиса в запущенном стэке надо выполнить следующую команду:

# pull image   
docker pull docker-registry.ru:5000/ptm:stage;  
docker service update --image docker-registry.ru:5000/ptm:stage stage;

Затем, если выполнить docker container ls на worker ноде, то мы увидим, что запустился новый контейнер со статусом **starting**, а когда он станет **healthy**, то swarm переключит трафик на новый контейнер, а старый остановит.

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
2fd259e82ade docker-registry.ru:5000/ptm:stage "docker-entrypoint.s…" 3 seconds ago Up 1 second (health: starting) 4003/tcp stage\_back.1.bxa0a1y86mnsi8yyd0wpxhc7j  
7e726dbc2f0d docker-registry.ru:5000/ptm:stage "docker-entrypoint.s…" 25 hours ago Up 25 hours (healthy) 4003/tcp stage\_back.1.q20unddw9v9i3tdojohckv7cw

Благодаря директивам rollback\_config и restart\_policy мы описали, что будет делать swarm c контейнером в случае, если не удалось его запустить и в каком случае перезапускать контейнер, а также максимальное кол-во попыток и задержек между ними.

Вот так с помощью десятка строчек мы получили бесшовный деплой для наших контейнеров.

Секреты

Swarm предоставляет хранилище для приватных данных (secrets), которые необходимы контейнерам. Как правило эта функциональность используется для хранения логинов, паролей, ключей шифрования и токенов доступа от внешних систем, БД и т.д.

Создадим yaml файл c «суперсекретным» токеном:

# example.yaml  
token: sfsjksajflsf\_secret

Создадим секрет с именем back\_config:

docker secret create back\_config example.yaml   
nys6v16j4d8ymmif87sq6w305  
  
# show secrets  
docker secret ls  
ID NAME DRIVER CREATED UPDATED  
nys6v16j4d8ymmif87sq6w305 back\_config 9 seconds ago 9 seconds ago  
  
# inspect specific secret  
docker secret inspect back\_config  
[  
 {  
 "ID": "nys6v16j4d8ymmif87sq6w305",  
 "Version": {  
 "Index": 24683  
 },  
 "CreatedAt": "2022-04-13T15:19:14.474608684Z",  
 "UpdatedAt": "2022-04-13T15:19:14.474608684Z",  
 "Spec": {  
 "Name": "back\_config",  
 "Labels": {}  
 }  
 }  
]

Для того чтобы им воспользоваться нам надо добавить 2 секции. Во-первых, секцию secrets, где мы укажем, что берем снаружи (из swarm) секрет под именем back\_config. Во-вторых, подключим сам секрет в сервисе (тоже директива secrets) по пути /p/ptm/config/config.json.

version: "3.9"  
  
services:  
 back:  
 image: docker-registry.ru:5000/ptm:stage  
 ports:  
 - "4004:4003"  
 environment:  
 TZ: "Europe/Moscow"  
 extra\_hosts:  
 - host.docker.internal:host-gateway  
 command: make server\_start  
 volumes:  
 - /p/ptm/stat/web:/p/ptm/stat/web  
 secrets:  
 - source: back\_config  
 target: /p/ptm/config/config.yaml  
 # swarm  
 deploy:  
 placement:  
 constraints:  
 - "node.labels.TAG==prod"  
 replicas: 1  
 update\_config:  
 parallelism: 1  
 order: start-first  
 failure\_action: rollback  
 delay: 10s  
 rollback\_config:  
 parallelism: 0  
 order: stop-first  
 restart\_policy:  
 condition: any  
 delay: 5s  
 max\_attempts: 3  
 window: 120s  
 healthcheck:  
 test: curl -sS http://127.0.0.1:4004/ptm/api/healthcheck || echo 1  
 interval: 30s  
 timeout: 3s  
 retries: 12  
   
secrets:  
 back\_config:  
 external: **true**

Ранее мы монтировали config.yaml, как volume, теперь же мы достаем его из swarm и монтируем его по указанному пути. Если мы теперь зайдем внутрь контейнера, то обнаружим, что наш секрет находится в /p/ptm/config/config.yaml.

# enter to container  
docker exec -it $container\_id sh  
  
cat /p/ptm/config/config.yaml;

Portainer

Кроме того, хочу упомянуть про такой инструмент как portainer. Вместо того, чтобы лазить постоянно на manager node и выполнять рутинные команды в консоли, можно использовать web панель для управления docker swarm. Это довольно удобный инструмент, позволяющий посмотреть в одном месте данные о стэках, сервисах и контейнерах. Можно отредактировать существующие сервисы, а также запустить новые. Кроме того, можно посмотреть логи запущенных контейнеров и даже зайти внутрь них удаленно. Ко всему прочему portainer может выступать в роли docker registry, а также предоставляет управление секретами и конфигами.

Для того чтобы его установить необходимо в начале скачать .yml файл с описанием сервисов:

curl -L https://downloads.portainer.io/portainer-agent-stack.yml \  
 -o portainer-agent-stack.yml

Затем задеплоить его в наш кластер:

docker stack deploy -c portainer-agent-stack.yml portainer

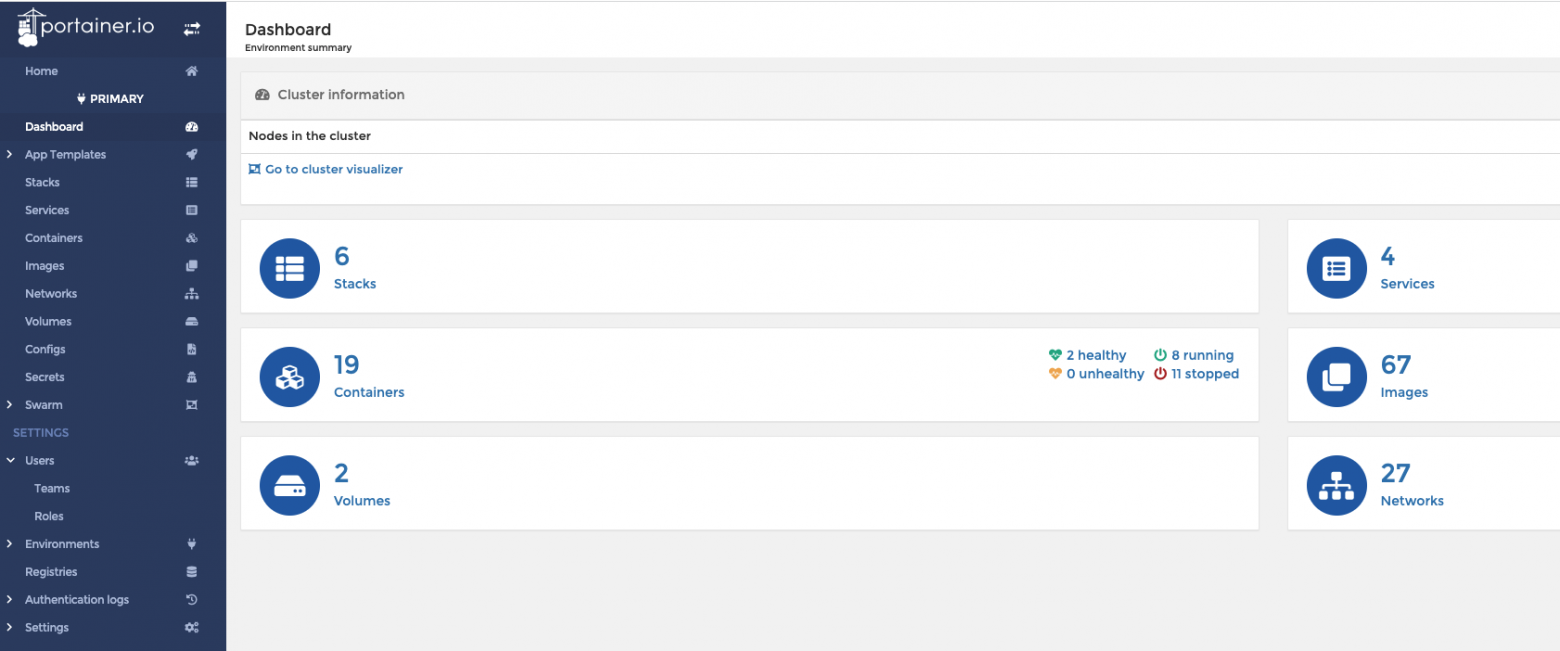
На каждом node будет установлен агент, который будет собирать данные, а на manager будет установлен сервер с web панелью.

Надо не забыть открыть порт 9443 на виртуальной машине, которая является manager:

firewall-cmd --add-port=9443/tcp --permanent;  
firewall-cmd --reload;

Заходим в браузере на адрес https://ip:9443.

Соглашаемся на то, что доверяем https сертификату (portainer, по умолчанию создает самоподписанный). Создаем пользователя и наслаждаемся.



Итог

Docker Swarm это несложный оркестратор для контейнеров, который доступен из коробки. Поверхностно с ним можно разобраться за пару дней, а глубоко за неделю. И по моему мнению, он закрывает большинство потребностей для маленьких и средних команд. В отличие от Kuberneteus он гораздо проще в освоении и не требует выделения дополнительных ресурсов в виде людей (отдельного человека или даже отдела) и железа. Если ваша команда разработки меньше 100 человек и вы не запускаете сотни уникальных типов контейнеров, то вам скорее всего вполне хватит его возможностей.