



Docker Clustering

MICROSERVICES





Agenda

- Why clustering
- Problems of clustering
- Available products
- Components of Docker cluster systems
 - Persistence
 - Networking
 - Configuration
- Docker Swarm Setup
- The „Ingress“ network



Why clustering

- High availability
- Scalability
- “Scale out” instead of “scale up”
- Simplifying the maintenance of cluster nodes (updating, replacement, hardware maintenance,...)



Problems of clustering

- Networking
 - Load balancing
 - On which IP/hostname is a service reachable?
 - Where are my dependant services reachable
- Scaling
 - How is the load balanced on all available instances
 - When should a service scale up or down?
- Configuration
- Secrets
- Persistence: where's the data?
 - NFS
 - vSphere Volumes
 - SAN storage(we talk about cloud solutions later)



Available products

- Kubernetes
 - OpenShift (based on Kubernetes)
- DC/OS (Mesosphere)
- Docker Swarm
- Rancher
- Nomad
- Mantl
- ...



Components – Persistence

- There are plenty of plugins available:
 - Azure File Storage
 - NFS
 - GCE Volumes
 - GlusterFS
 - SAN systems (NetApp, Nimble, Virtuozzo)
 - VMWare vSphere Storage
- Most of them are trying to solve the problem that a container may run on any host so the data has to be available on any host!
- A few are also able to control concurrent R/W access by multiple containers (e.g. GlusterFS, VMWare)



Components – Persistence

- Which persistence driver you want to use depends on your environment
- Is very fast access required (databases, caches,...)?
- Do multiple instances have to share the same data (file storages like NextCloud)?
- Is there any infrastructure available (cloud native like AWS, Azure, GCE, on premise like SAN systems or a NFS server)?
- How fast do you have to setup the cluster?
(the setup of a NFS is much easier than a GlusterFS cluster)



Components – Networking

- Port publishing (on which cluster-node is an application reachable?)
 - Ingress network in Docker Swarm
- Communication between containers across multiple container hosts
 - Overlay networks in Docker Swarm
- Isolation of applications against each other
(multiple applications may need the same services but would interfere with each other)
 - Stacks and Services in Docker Swarm
- Service Discovery (method to access other services in an application)
 - Realized with DNS in Docker Swarm



Components – Configuration

- Mounting configuration files into container
- Securing sensible data like credentials, API keys, ...
- Share configuration files across multiple containers and hosts
- Docker Swarm has special CMDlets for storing secrets (*docker secret*) and configuration files/values (*docker config*)



Components - Scaling

- Dynamically scale services up and down
- *Optionally: scale services when load is getting higher or lower (Kubernetes)*
- Docker Swarm is able to start new containers on demand (*docker service scale helloworld=5*) but it does not detect if the load is getting worse so this is up to you



Docker Swarm concepts

- Based on Raft (consensus protocol, will be discussed later)
- Nodes are manager or worker
- Managers are taking actively part in the Raft consensus
- One manager is the leader, the others are in stand-by and are forwarding requests to the master
- Interactions (like starting a service) are only possible with a master node



Docker Swarm – Setup

- To create a new Docker Swarm you need at least 2 nodes (for high availability 4 nodes)
- On machine 1 (let's call it *master*) you're initializing the swarm by running `docker swarm init --advertise-addr <MASTER IP>`
- To get a join token you've to run on the *master* `docker swarm join-token worker`. As result Docker should print the complete command to join a worker node
- On machine 2 (let's call it *worker*) you executed the printed command (e.g. `docker swarm join --token SWMTKN-1-49nj1cmql0jkz5s954yi3oex3nedyz0fb0xx14ie39trti4wxv-8vxv8rssmk743ojnwacrr2e7c <MASTER IP:PORT>`)
- A join token may be used multiple times but it has a lifetime so you will have to generate new ones from time to time

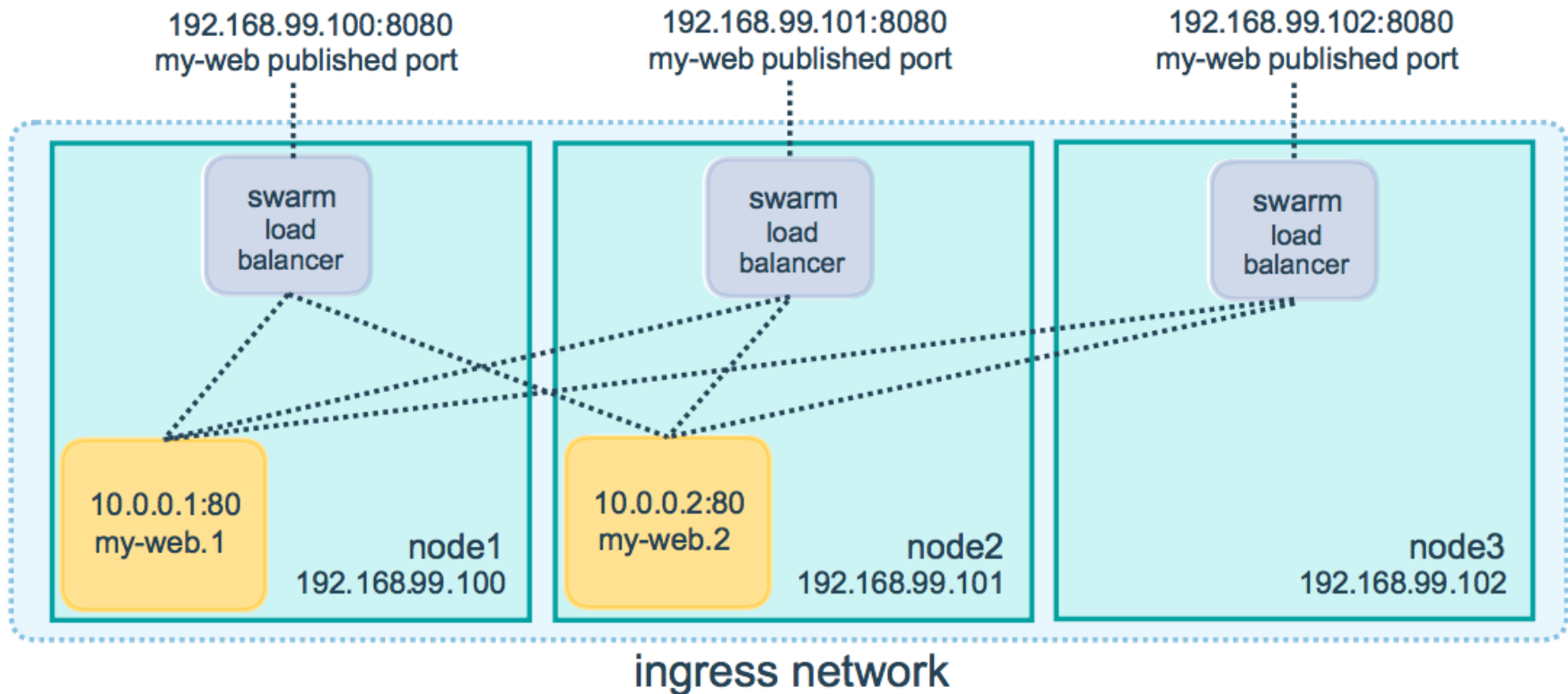


Docker Swarm – Routing Mesh





Docker Swarm – Routing mesh



Source: <https://docs.docker.com/engine/swarm/ingress/>



Docker Swarm – Accessing services

- Like with any local Docker container a Swarm Container publishes a port – ports which should be available from outside the swarm are published through the so called “Ingress” network
- A port published by the ingress network is available at every node (even on worker nodes)
- Communication between services is always possible (if they’re in the same network)
- To resolve other services Docker Swarm uses internal DNS entries for every service, all calls are balanced over all available instances of a specific service (load balancing)



Docker Swarm – Deployment

The screenshot displays the Portainer.io web interface. On the left, a dark blue sidebar contains the 'portainer.io' logo and a navigation menu. The menu is divided into 'ACTIVE ENDPOINT' (showing 'primary'), 'ENDPOINT ACTIONS' (Dashboard, App Templates, **Stacks**, Services, Containers, Images, Networks, Volumes, Secrets, Swarm), and 'PORTAINER SETTINGS' (User management, Endpoints, Registries, Settings). The 'Stacks' menu item is highlighted in yellow. The main content area is titled 'Stacks list' and includes a 'Refresh' icon. At the top right of the main area, the user 'admin' is logged in, with links for 'my account' and 'log out'. Below the title, there are 'Remove' and 'Add stack' buttons, and a 'Filter...' input field. A table with columns 'Name' and 'Ownership' is shown, but it is empty, with the message 'No stacks available.' displayed below the headers. The table has a 'Items per page: 10' dropdown in the top right corner.



Docker Swarm – Deployment

```
1  version: '3'
2
3  services:
4    web:
5      image: 127.0.0.1:5000/stackdemo
6      build: .
7      ports:
8        - "8000:8000"
9    redis:
10     image: redis:alpine
11
12
```