CRIM

CENTRE DE
RECHERCHE INFORMATIQUE
DE MONTRÉAL

Docker Swarm: Myths and Realities of Deploying Complex Environments for Scientific Software

Presented during the Canadian Research Software Conference

BY: Anton Zakharov & Louis-David Perron

DATE : 2019-05-28

crim.ca

PRINCIPAL PARTENAIRE FINANCIER

Québec

What is Docker Swarm?

- Swarm (called swarm mode) is a set of native Docker tools, which allow to deploy a set of Docker containers on multiple machines as if they were deployed on a single machine.

- See: https://docs.docker.com/engine/swarm/ for more information

Swarm Usage at CRIM

- Used for all our scientific platforms: VESTA, PACTE, Services for ELAN.

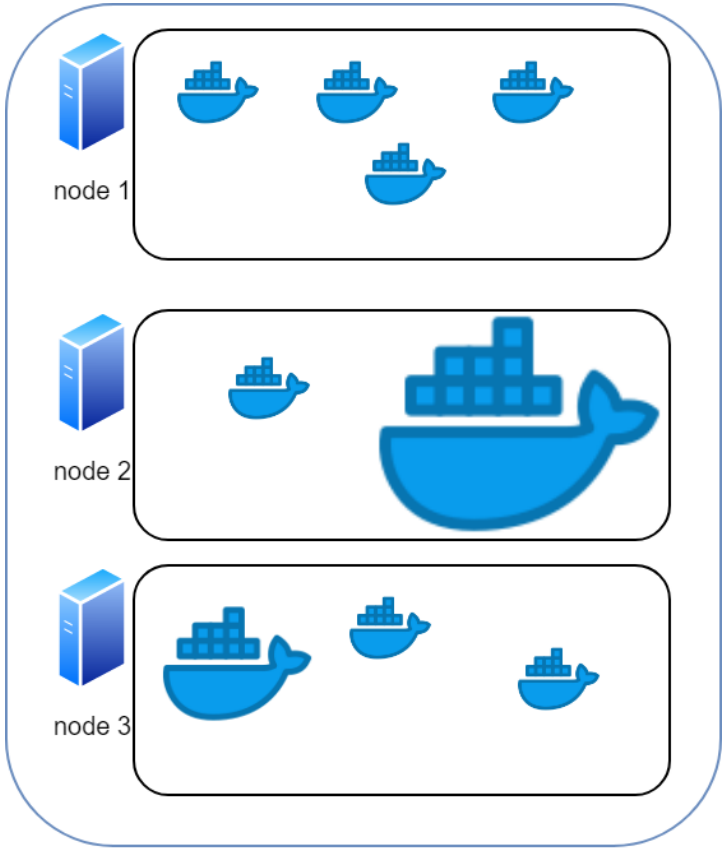
- Biggest deployment on CANARIE's DAIR, spanning 10 virtual machines and over 40 different services

Why Use Swarm?

- Using only Docker required a lot of error prone configuration and manual operations to boot and interconnect all the services.
- We were using virtual machines of heterogeneous size and wasted a lot of unused memory and CPU capacity.
- We needed of horizontal scaling of specific services, and over multiple servers.
- Our development was based on Docker Compose, and wished to directly translate from the compose file to production.
- Other solutions, like Kubernetes, are too complex to setup, while having similar limitations.

Swarm Overview

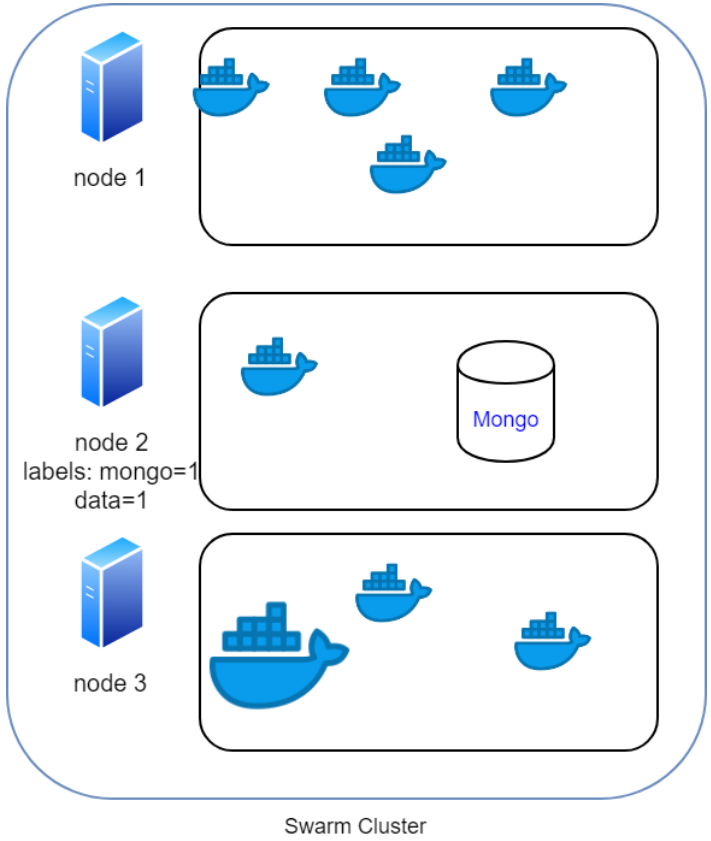
- Docker way of deploying containers in a cluster of multiple servers
- Multiple instances of a service can be replicated for load balancing and high availability
- Transition from Docker Compose to swarm mode is trivial



Swarm Differences: Persistent Data

- Events may cause the swarm to move a service to another node
- Unfortunately, the persistent data won't be moved with it without volume plugins
- Workaround: use node labels and placement rules to make sure services with persistent data always stay on the same node:

```
services:
  mongo:
    deploy:
      placement:
        constraints:
          - node.labels.mongo == 1
```



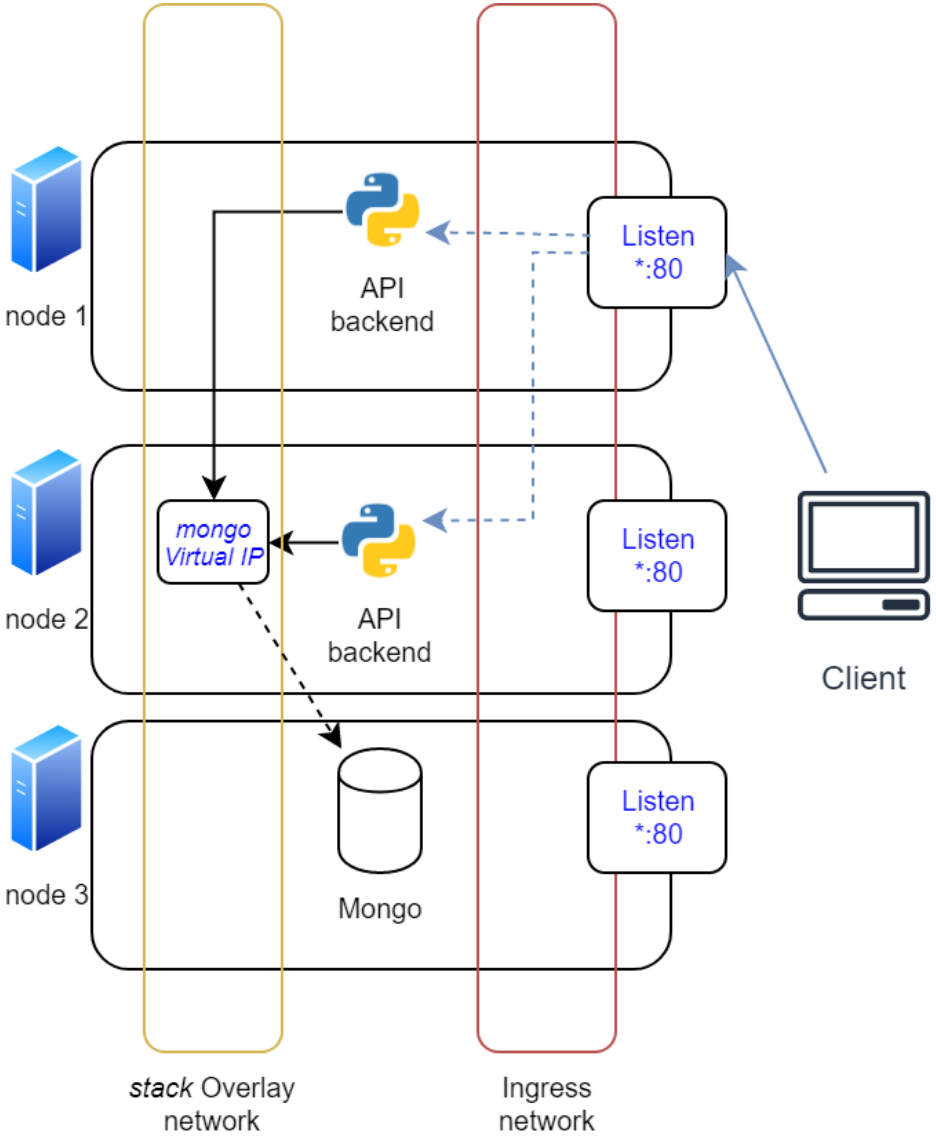
Swarm Differences: Resource Reservation

- docker-compose YAML format allows you to specify resources requirements

- Warning: The Docker Swarm container placement algorithm is greedy. Make sure you specify large services not to be instanced on the same nodes where you previously *pinned* a service with persistent data.

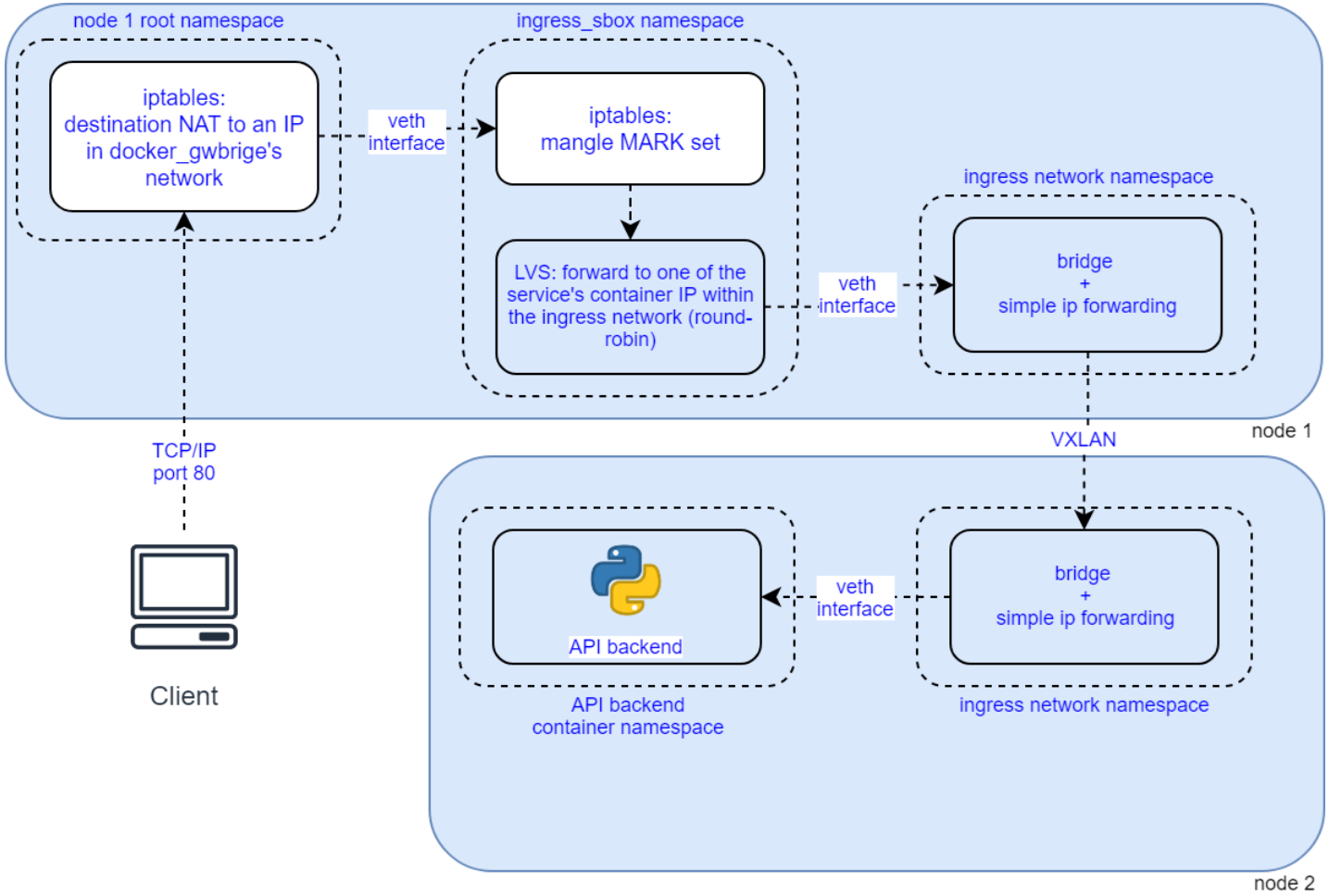
```
services:
  space-time-tradeoff-software:
    deploy:
      resources:
        reservations:
          memory: 99999M
      placement:
        constraints:
          - node.labels.data != 1
```

Swarm Differences: Networking



- Services within the network stack can locate other services by the *hostname* associated to a fixed Virtual IP within the stack overlay network
- Clients can connect to a service's published ports on any nodes of the swarm cluster. Requests are routed through the ingress network
- Within stack and ingress networks, if a service is deployed with multiple replicas, incoming requests will be load balanced

Ingress Network Under the Hood



Potential Network Pitfalls

Where it can go wrong:

- Network namespaces: Docker's way of isolating container networking
- IP forwarding & bridging: forwards data from a network interface to another
- iptables: Linux's way of implementing firewall rules
- LVS (ipvsadm): virtual server and load balancing at kernel level
- Virtual ethernet interfaces (veth): interfaces link between network namespaces
- VXLAN: virtual network distributed over swarm nodes, using UDP transport

Symptoms of our Networking Problem

While the project worked perfectly using only Docker Compose, running in swarm mode, we experienced:

- Random requests time out to databases and RabbitMQ, especially when a connection pool is being used
- HTTP requests to our stack services with more than 900 seconds before the response will never get answered

Example toy app:

```
@app.route('/')
def hello_world():
    time.sleep(905)
    return 'Hello, World!'
```

The Culprit: Linux Virtual Server (LVS)

Features:

- No userspace software required
- Allows load-balancing of a virtual service to multiple real servers
- Incorporated in Linux kernel source tree since 2.4.x

Problems:

- By default, connection entries expire after 15 minutes of inactivity
- When a connection is expired, no TCP finalize or reset is sent
- Subsequent packets within a TCP connection will be silently dropped

Solutions/workarounds:

- First solution was to lower the sysctl **net.ipv4.tcp_keepalive_time** to 600 seconds (it's 2 hours by default): this will reset the LVS timeout every 10 minutes. Cons: keepalive is not enabled by default: it's a socket option.
- Setup a cron job on all swarm nodes that will raise the LVS timeout in every network namespaces created by Docker (ipvsadm needs to be installed):

```
for F in /var/run/docker/netns/*; do nsenter --net=$F /sbin/ipvsadm --set 7300 0 0; done
```

Hackish workaround, but fixes a bug reported more than two years ago with still no viable solutions today!

Conclusion

- We are happy with swarm simplicity and use it extensively.
- The only big issue we had with networking (which affects other solutions as well) has been solved.
- Storage limitations are not a problem in our use case.

Anton Zakharov,
Research Software Developer
anton.zakharov@crim.ca

Louis-David Perron
Sysadmin & DevOps Specialist
louis-david.perron@crim.ca

www.crim.ca