Konténerek orkesztrálása – Docker Swarm Mode

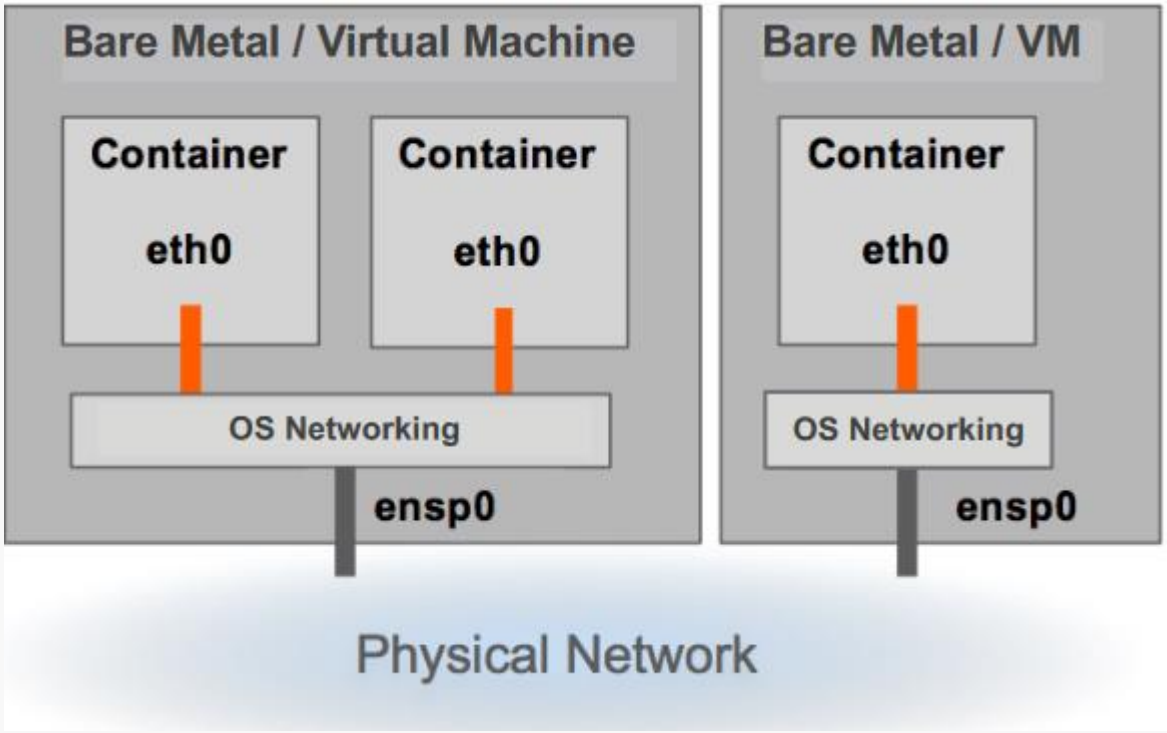
Simon Csaba

VM hálózati megoldások helyett konténer megoldás

Feature	Container	VM
Isolation	Network isolation achieved using Network namespace.	Separate networking stack per VM
Service	Typically, Services gets separate IP and maps to multiple containers	Multiple services runs in a single VM
Service Discovery and Load balancing	Microservices done as Containers puts more emphasis on integrated Service discovery	Service Discovery and Load balancing typically done outside
Scale	As Container scale on a single host can run to hundreds, host networking has to be very scalable.	Host networking scale needs are not as high
Implementation	Docker Engine and Linux bridge	Hypervisor and Linux/OVS bridge

Konténer: hálózat elvárások

- IP kapcsolatok a saját hálózatban
- IP címek: IPAM – IP Address Management
- Külvilág elérése: NAT, hozzáférés a host i/f-hez



IPAM – IP Cím menedzsment

- Docker does the IP address management by providing subnets for networks and IP addresses for containers.
- For default “bridge” network, custom subnet can be specified in Docker daemon options.
- Users can specify their own subnet while creating networks and specify IP when creating containers. Following example illustrates this.

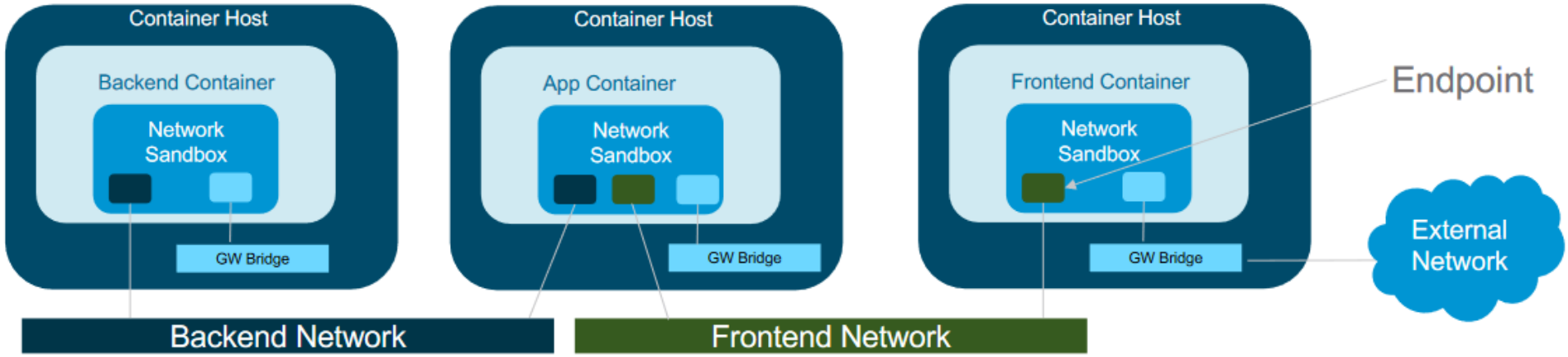
  docker network create --subnet=172.19.0.0/16 mynet

  docker run --ip 172.19.0.22 -it --network mynet smakam/myubuntu:v4 bash

- Using remote IPAM plugin, IP addresses can be managed by external application instead by Docker.
- Remote IPAM plugin can be specified using “--ipam-driver” option while creating Docker network. Infoblox is an example of external Docker IPAM plugin.
- Docker also supports assignment of IPV6 addresses for Containers.

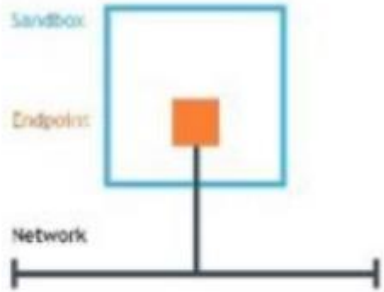
Docker hálózati modell

- Docker: Container Network Model (CNM)
- Libnetwork – a megvalósítás
- Sandbox, Endpoint és Network



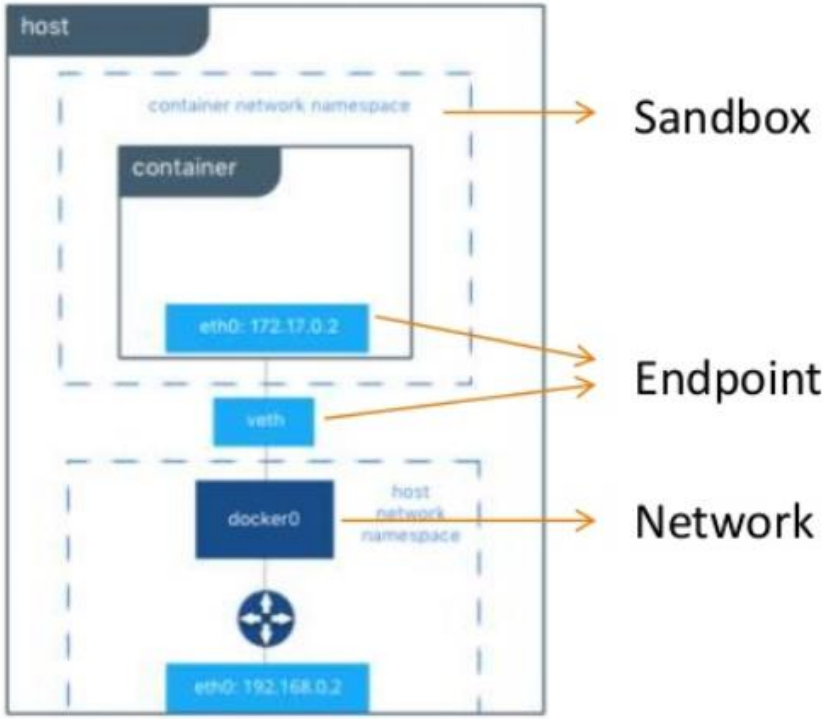
CNM és Libnetwork

CNM constructs



Picture from Docker white paper

CNM usage in Docker



Sandbox — A Sandbox contains the configuration of a container's network stack. In Docker example, Container network namespace is the equivalent of Sandbox.
Endpoint — An Endpoint joins a Sandbox to a Network. Eth0 and veth are the endpoints in above example.
Network — Multiple endpoints share a network. In other words, only endpoints located in same network can talk to each other. In above example, docker0 is the bridge network.

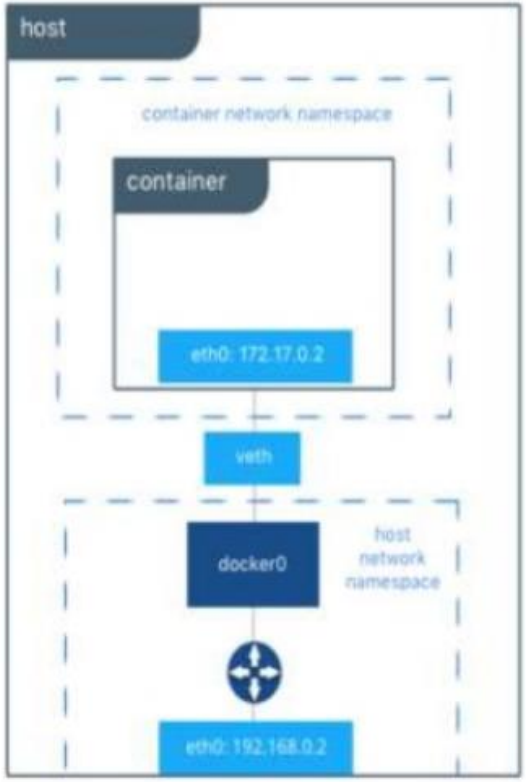
Docker hálózati alternatívák

Driver/ Features	Bridge	User defined bridge	Host	Overlay	Macvlan/ipvlan
Connectivity	Same host	Same host	Same host	Multi-host	Multi-host
Service Discovery and DNS	Using "links". DNS using /etc/hosts	Done using DNS server in Docker engine	Done using DNS server in Docker engine	Done using DNS server in Docker engine	Done using DNS server in Docker engine
External connectivity	NAT	NAT	Use Host gateway	No external connectivity	Uses underlay gateway
Namespace	Separate	Separate	Same as host	Separate	Separate
Swarm mode [1]	No support yet	No support yet	No support yet	Supported	No support yet
Encapsulation	No double encap	No double encap	No double encap	Double encap using Vxlan	No double encap
Application	North, South external access	North, South external access	Need full networking control, isolation not needed	Container connectivity across hosts	Containers needing direct underlay networking

Default: Docker bridge network

- Used by “docker0” bridge and user-defined bridges.
- “docker0” bridge is created by default. User has the choice to change “docker0” bridge options by specifying them in Docker daemon config.
- User-defined bridges can be created using “docker network create” with “bridge” driver.
- Used for connectivity between containers in same host and for external North<->South connectivity.
- Services running inside Containers can be exposed by NAT/port forwarding.
- External access is provided by masquerading.

docker run -d -p 8080:80 --network bridge --name web nginx

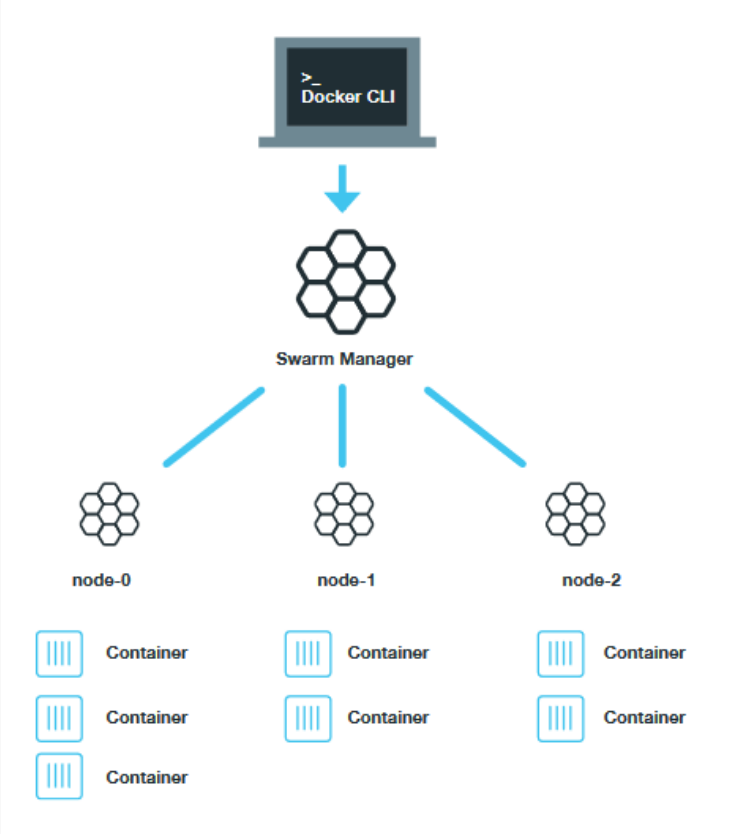


Picture from Docker white paper

MOTIVÁCIÓ

Motiváció – multi host

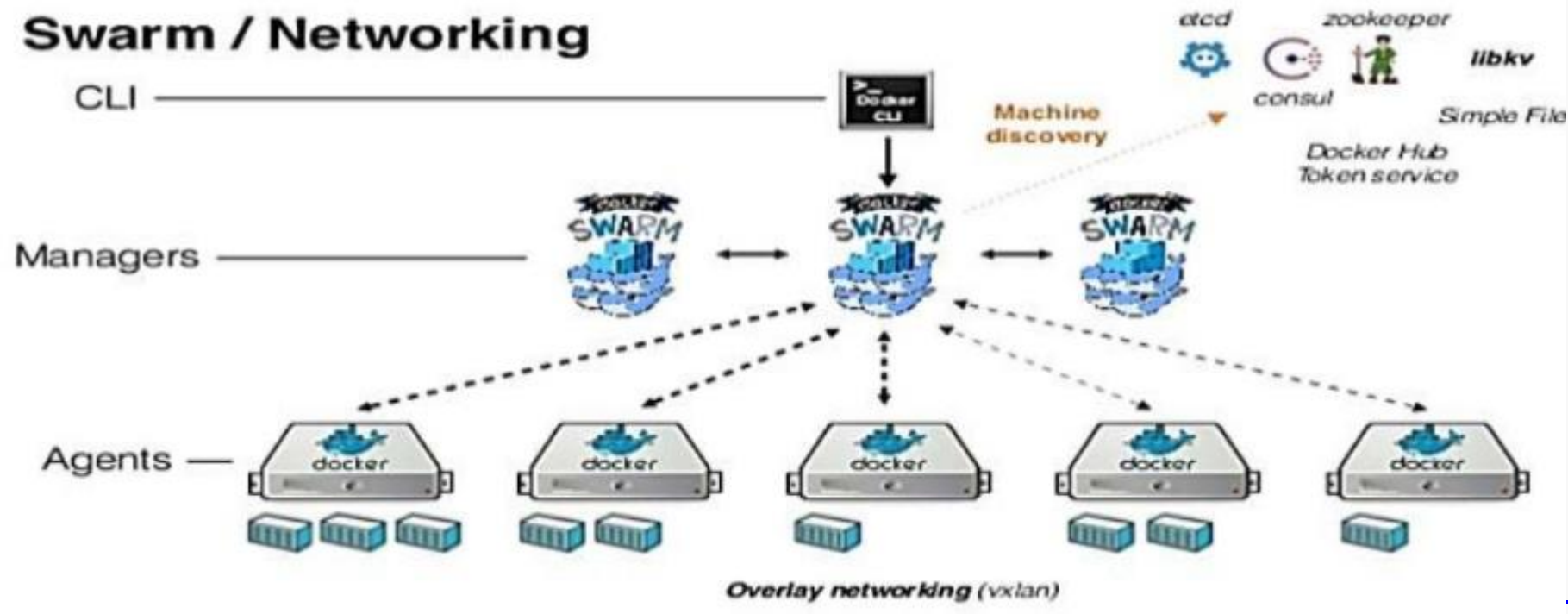
- Docker konténerek docker parancsokkal kezelhetők
  - Adott gazda gépen (on-host)
  - Hálózati kapcsolatok nehézkesek
    - docker0 bridge
  - Különálló gépekre telepített docker konténerek kapcsolata?
    - Mult-hosting
    - Először külső megoldások (pl. serf - https://www.serf.io/)
  - Később Docker Swarm – multi-hosting in Docker



„It turns a pool of Docker hosts into a single, virtual Docker host”
  - Nem összetévesztendő a Docker Swarm Mode-al :((v1.12 után)

Docker Swarm

- Consul, ZooKeeper, etcd – külső KV (key-value) tár
- Melyik Docker node része a klaszternek, kell-e hálózat egy adott névtérben
- Külső komponens, de ugyanaz az API, mint a Docker esetében

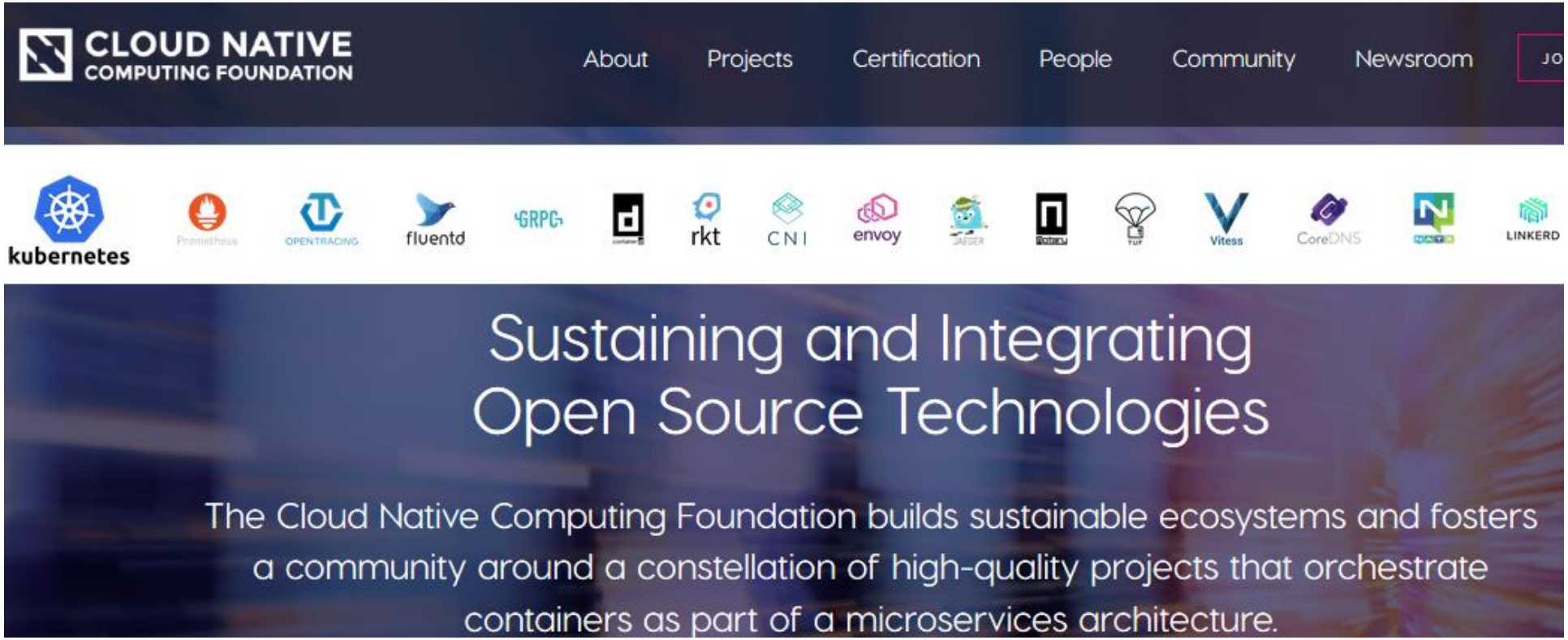


Motiváció - orkesztráció

- Mi hiányzik egy teljes Docker rendszerhez?
  - Orkesztráció
  - Amit a felhők nyújtanak
  - Cél: automatizált konténer telepítés és menedzsment multi-host környezetben (incl. skálázódás vezérlése)
- Egyik megoldás: nyilvános felhőkben Docker
  - Amazon Web Services, Google Cloud, Microsoft Azure
- Másik megoldás: Docker + OpenStack
  - OpenStack Magnum
- Harmadik megoldás: Docker orkesztráció
  - Apache Mesos (2010)
  - Google Kubernetes (2014)
  - Docker Swarm Mode (2016)

Cloud Native Computing Foundation

- Mikroszervíz ökoszisztéma
  - Konténerek orkesztrálásával
  - Google támogatja
  - rkt-t javasolják a Docker helyett



DOCKER SWARM MODE

Docker Swarm Mode

- Swarm mode = Docker engine futtatási mód
  - Amennyiben a Docker engine-k egy közös klaszterbe vannak szervezve
    - Egy Docker engine = egy node
  - Swarm = ez a fenti klaszter
    - A cél: szolgáltatások (services) indítása ebben a klaszterben
- Egy fizikai gép elvileg több node-ot tartalmazhat
  - „Komoly” üzleti környezetben egy Docker engine / fizikai gép
    - Gyakorlatilag a Docker engine-t futtató host-ok vannak egy klaszterben
- Szolgáltatói modell = a felhasználók egy service-t érnek el
  - A service replikált feladatot végez és meghatározza a működés kereteit (hálózat, erőforrás, replikációs szint és politika)
    - Több node-on futtatot feladatokat (task) együttesen képvisel
  - Task = feladat (= docker konténer), amit a service kezel
    - Elemi erőforrás egység, egy node-on fut

Legacy Swarm vs Swarm Mode

» A legacy swam az különálló komponens, a swarm mode az integrált megoldás

Swarm	Swarm Mode
Separate from Docker Engine and can run as Container	Integrated inside Docker engine
Needs external KV store like Consul	No need of separate external KV store
Service model not available	Service model is available. This provides features like scaling, rolling update, service discovery, load balancing and routing mesh
Communication not secure	Both control and data plane is secure
Integrated with machine and compose	Not yet integrated with machine and compose as of release 1.12. Will be integrated in the upcoming releases

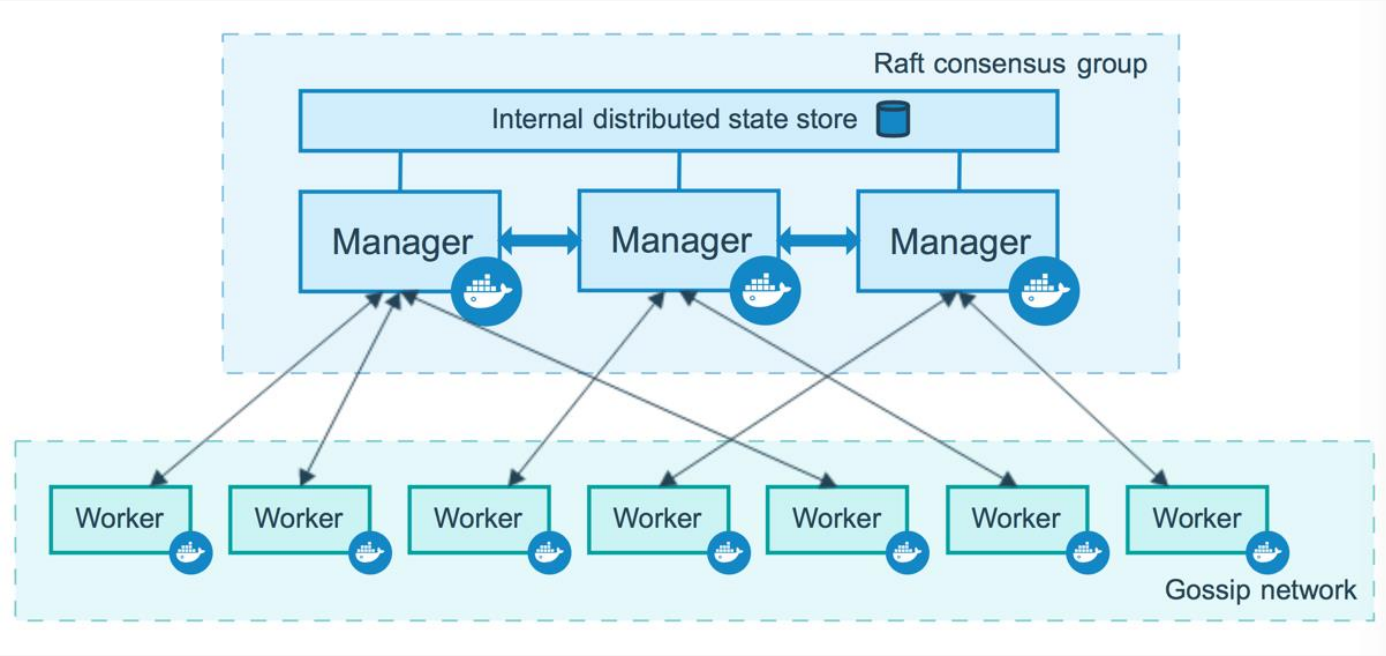
SwarmKit

- Önálló Docker projekt
- De a DockerEngine az 1.13 verziótól a SwarmMode-ban használja

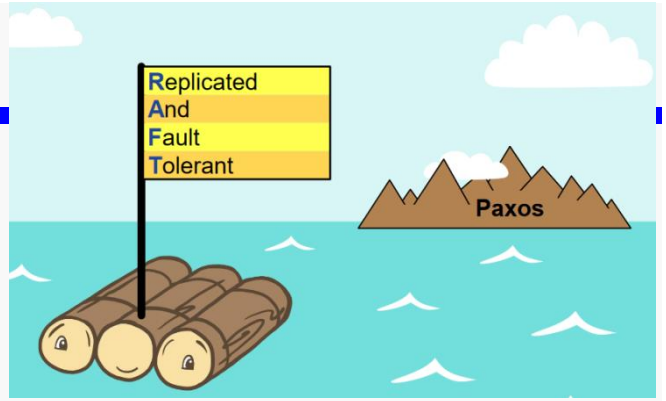
Swarmkit	Swarm Mode
Plumbing opensource project	Swarmkit used within Swarm Mode and tightly integrated with Docker Engine
Swarmkit needs to built and run separately	Docker 1.12 comes integrated with Swarm Mode
No service discovery, load balancing and routing mesh	Service discovery, load balancing and routing mesh available
Use swarmctl CLI	Use regular Docker CLI

Swarm Mode architektúra

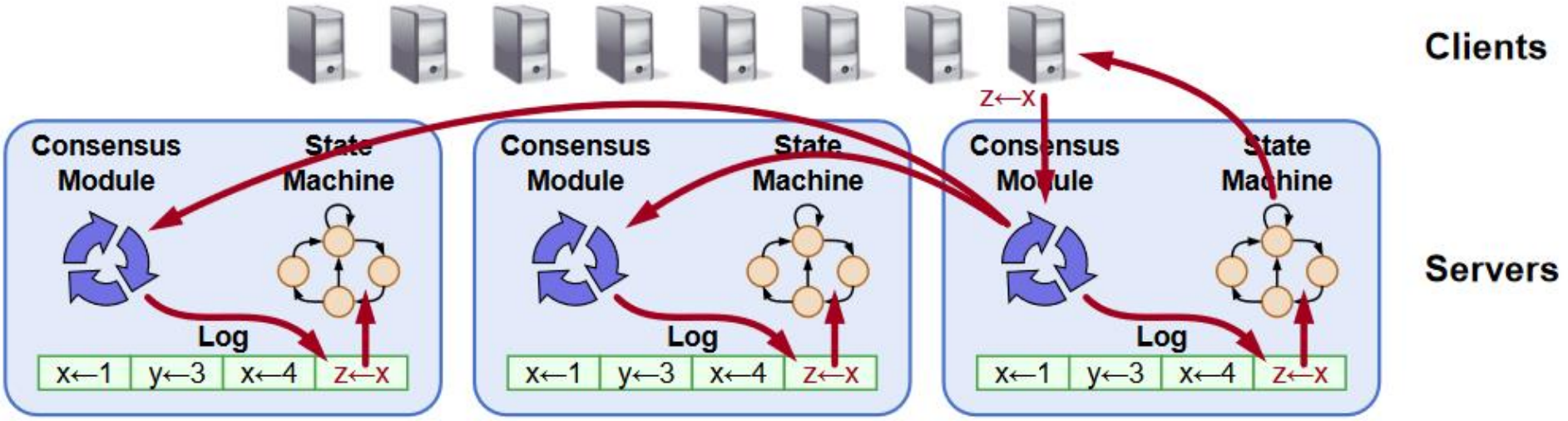
- A Docker Swarm Mode node-jait egy menedzser (Manager) vezérli
  - Feladata: klaszter vezérlés, API biztosítása, erőforrás ütemezés
  - Több Manager elosztott redundáns üzemet biztosít (high availability)
- A Worker node = task-ok futtatása (Manager egyúttal worker is lehet)
  - Worker node futtatás során Manager-ré nevezhető ki (és fordítva)
  - Worker node-ok egy mesh hálózatban vannak szervezve



Raft konszenzus

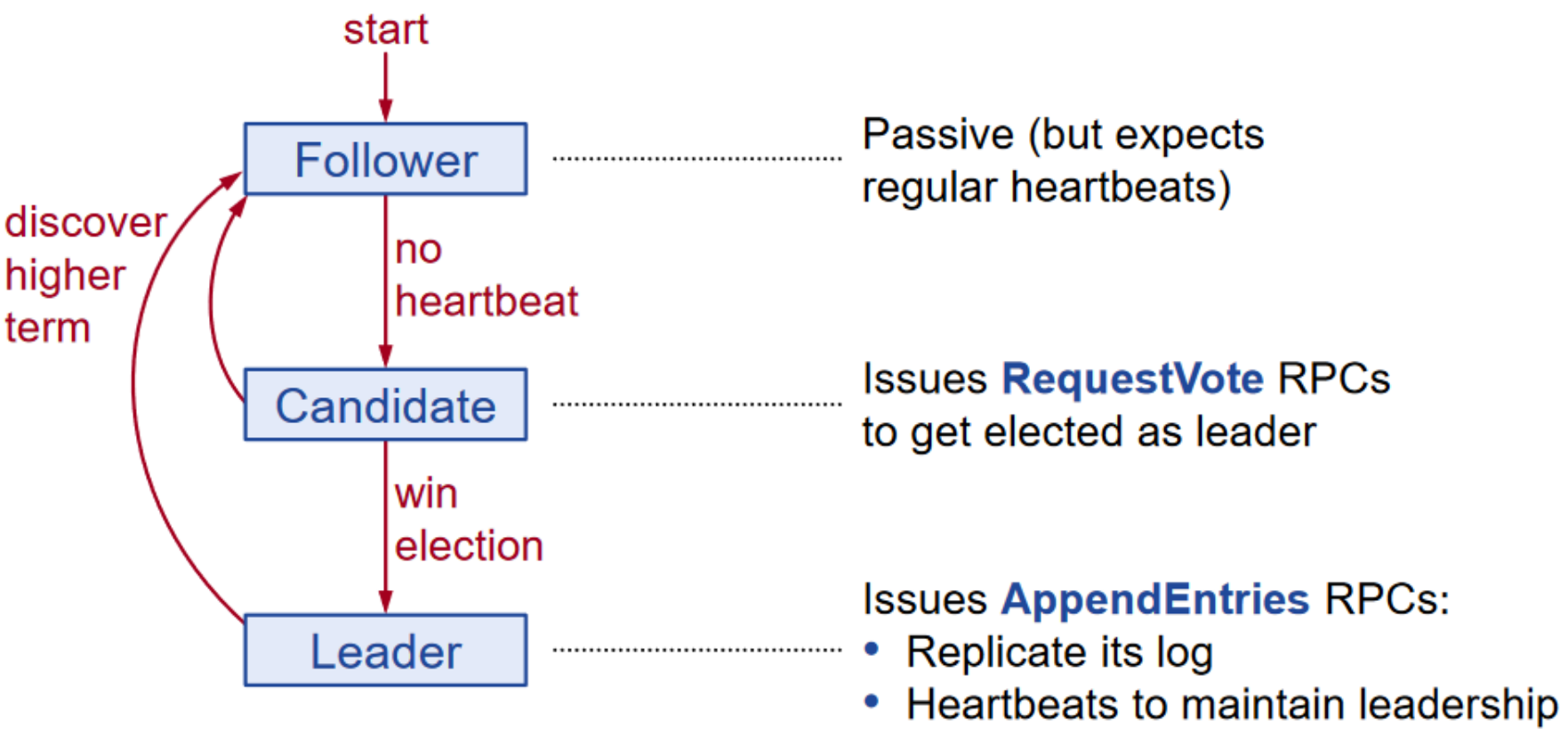


- Elosztott megoldás
- A cél a workerek állapotáról egységes képet kapni
- Redundancia biztosítása egy dinamikus környezetben (pl. ha meghal/kilép egy master)



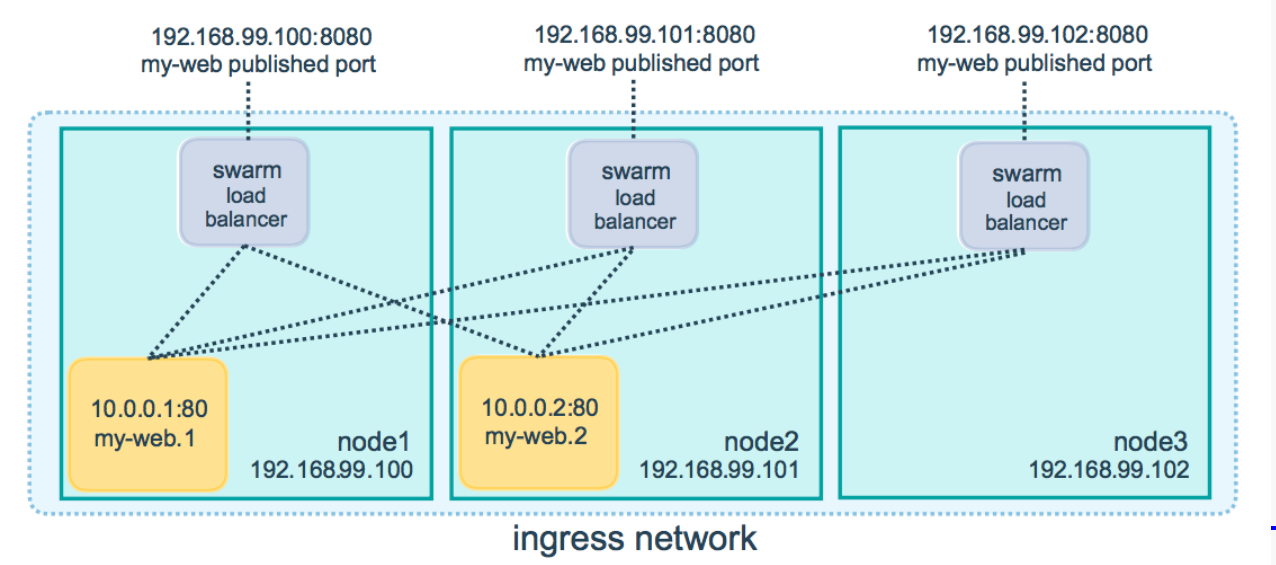
Legfennebb egy „leader” egy klaszterben

» Fail/stop modell a „bizánci generálisok” helyett



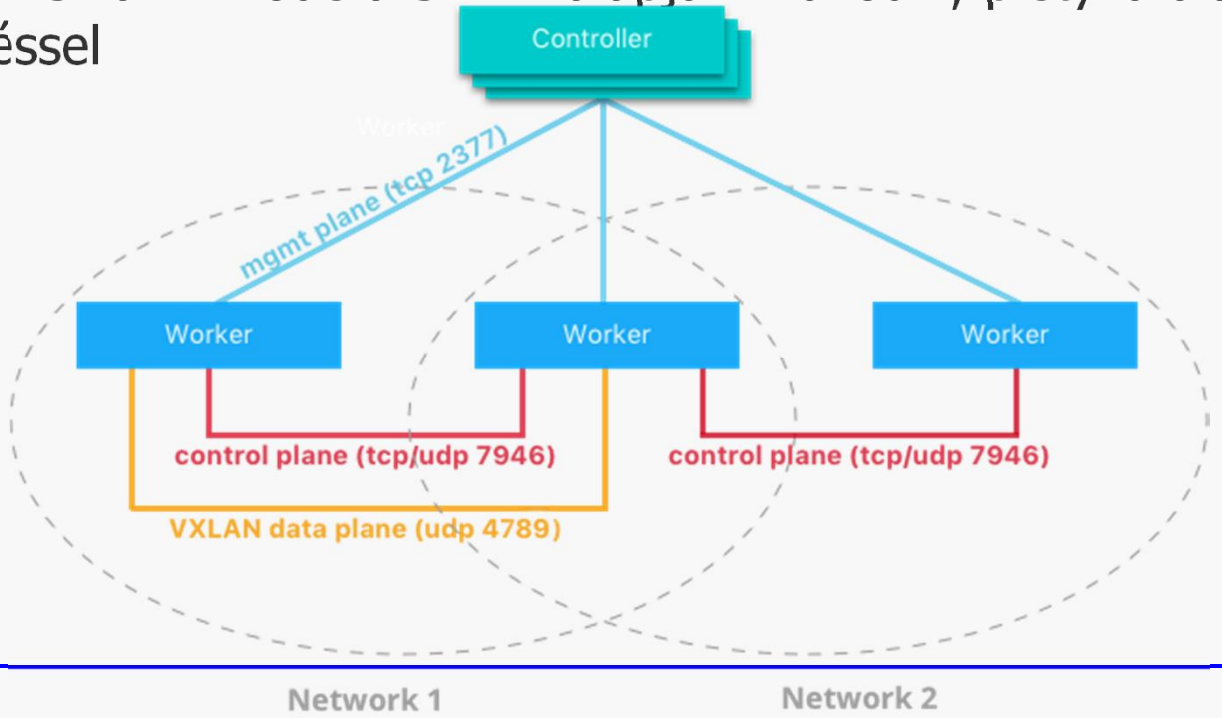
Swarm mode hálózatkezelése

- Szolgáltatáshoz port-ot rendelni
  - Swarm-on kívüli kérések fogadása (ingress nw = bejövő hívást kezelő hálózat)
  - A host-ok egy *Swarm mode routing mesh* részei kell legyenek
- Minden host-on kell futnia egy terheléselosztó funkciót (load balancer) is végző modulnak
  - Swarm mode routing mesh része
  - Ez juttatja el a kérést a megfelelő konténerhez
  - Akkor is, ha az a konténer más host-on fut
  - Akkor is, ha a kérést először fogadó host-on nem is fut olyan task



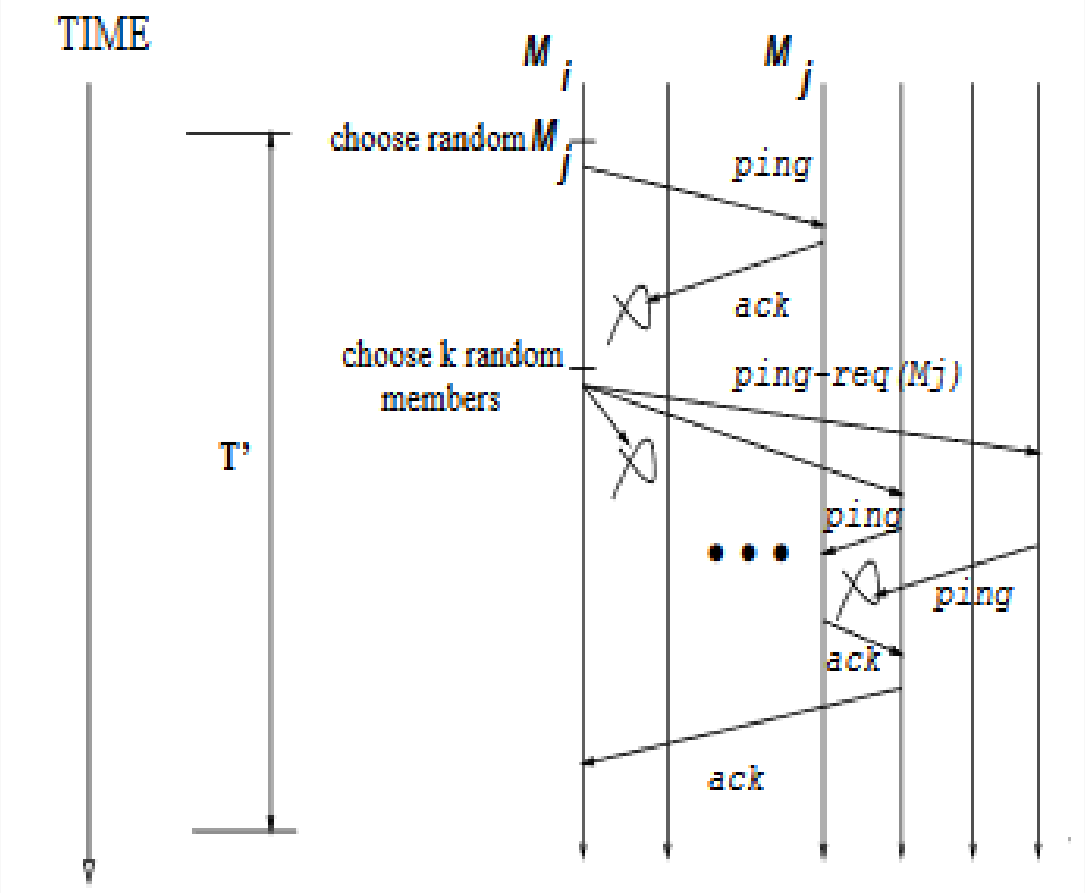
Routing mesh: SWIM

- A Docker Hálózati Vezérlési Sík (Docker Network Control Plane) biztosítja a hálózati információ konzisztenciáját
- Routing átfedő (overlay) tart fent
  - Az adat sík VXLAN alapon van megvalósítva
- SWIM – elméleti projekt
- Hashicorp Serf javította
- A Docker Swarm Mode a SWIM alapján működik, pletyka alapú állapot terjesztéssel

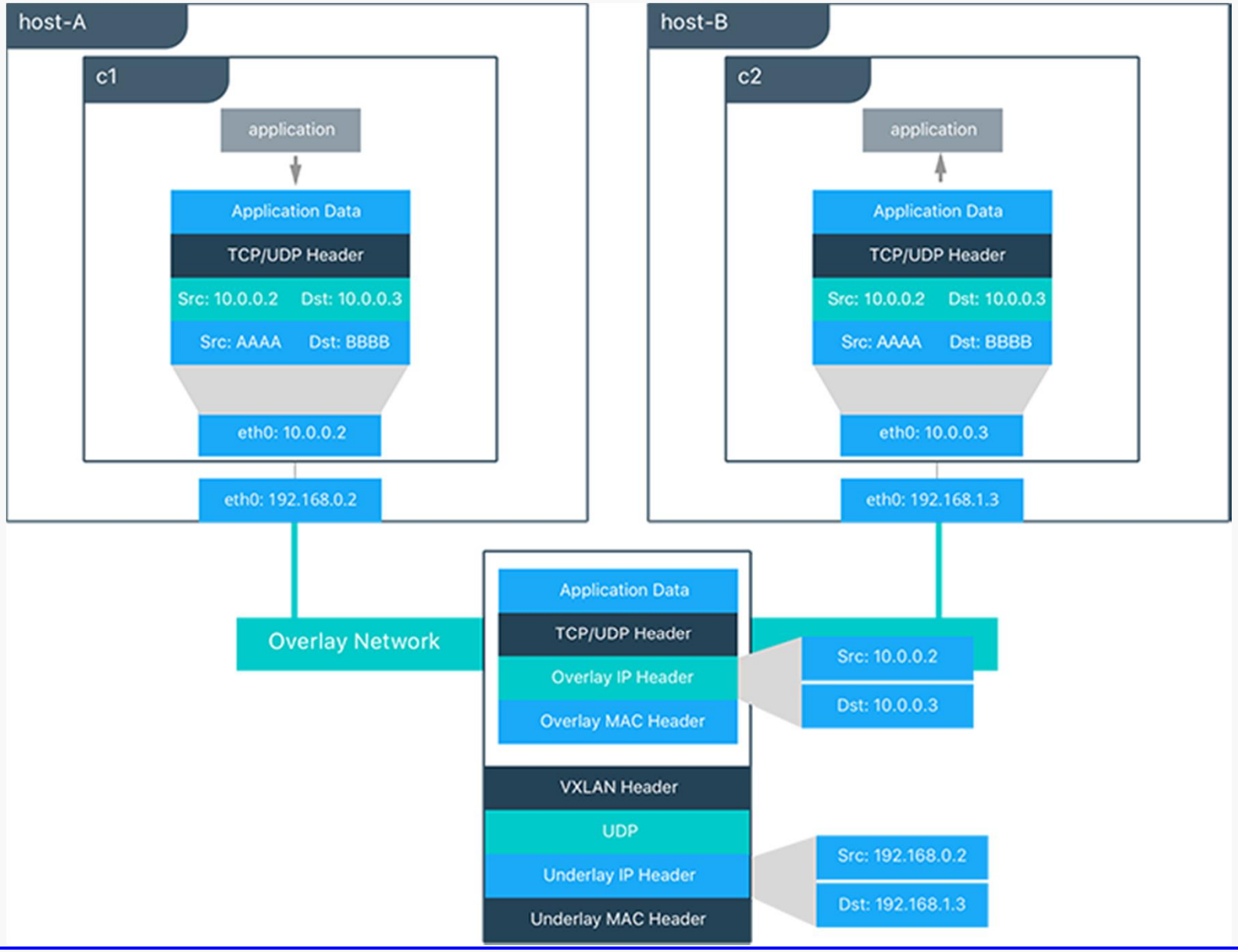


SWIM – „okos” ping mechanizmus

- Nem csak direkt ping
- „Piggyback”
- Nem zárja ki azonnal a node-ot
- Státusz frissítés
  - pletyka alapon



VXLAN az adatsíkban



Szolgáltatás azonosítás, „registry”

- Service discovery is provided by DNS server available in Docker engine.
- For unmanaged containers, container name resolves to container IP. Alias names can be also be used.
- For services using service IP(endpoint mode=vip), service name resolves to service IP which in turn forwards the request to containers. In this case, ipvs based L4 load balancing is done.
- For services using direct DNS(endpoint mode=dnsrr), service name directly resolves to container IP. In this case, DNS round robin load balancing is done.
- Service Discovery is network scoped. Only containers in same network can discover each other.

Szolgáltatás hozzáférés: Load Balancing

- For unmanaged containers, load balancing is done using simple round robin load balancing. Using aliases, a single alias can load balance to multiple unmanaged containers .
- Docker takes care of load balancing internal services to the containers associated with the services.
- For services using service IP(endpoint mode=vip), ipvs and iptables are used to load balance. This provides L4 based load balancing. Ipvs is Linux kernel load balancing feature.
- For services using direct DNS(endpoint mode=dnsrr), DNS round robin balancing is used.
- For services exposed externally, Docker uses routing mesh to expose the service on all Swarm nodes. Routing mesh uses “ingress” network to connect all nodes.
- For HTTP based load balancing, HRM(HTTP Routing mesh) can be used. This is supported only with Docker EE.