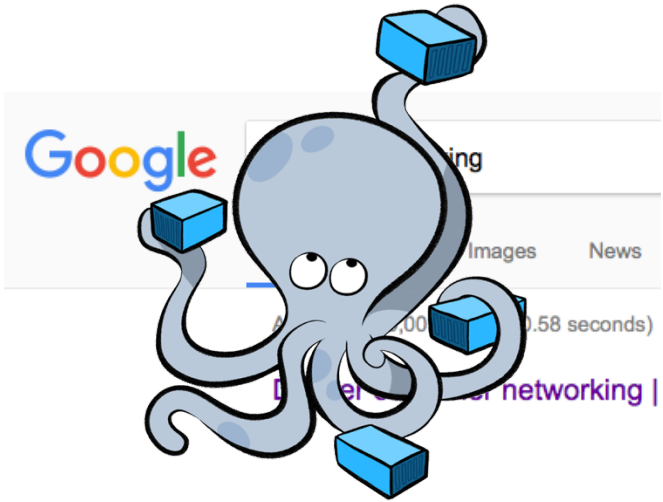
Practical Design Patterns in Docker Networking

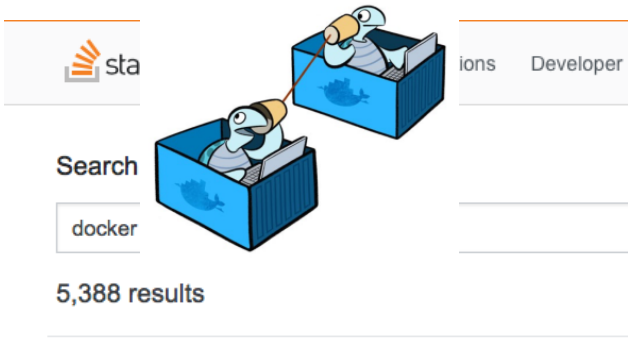
Dan Finneran

EMEA Solutions Architect, Docker

dockercon17 EU

Why this topic?





dockercon17 EU

Agenda

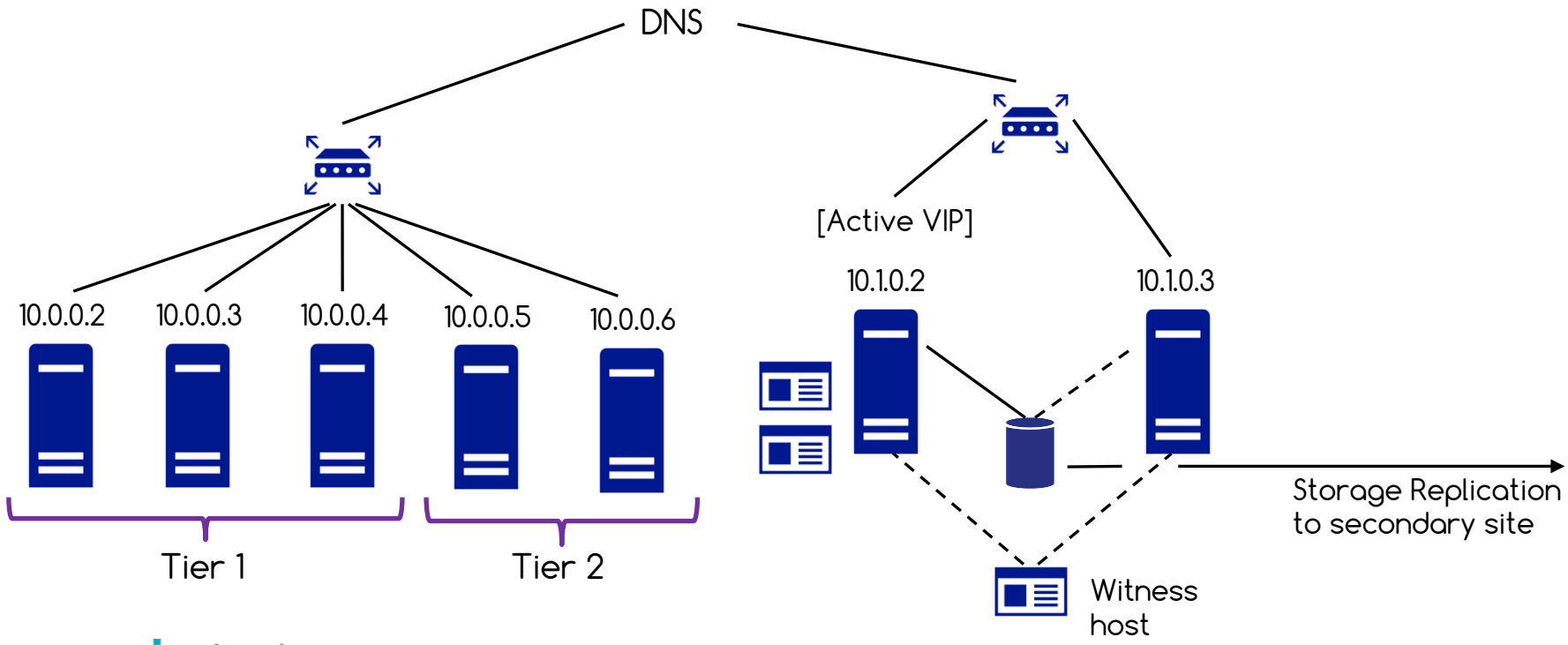
- The evolving architecture of application networking
- Docker networking
- Infrastructure design patterns
- Design Patterns when modernizing a traditional application
- [REDACTED]
- Summary and Q/A

The evolving architecture of application networking

Physically hosted applications

- Services, application components are 1:1 with network addresses and architecture.
- Often flat or simplistic networks defined by physical network ports or VLANs used to segregate the application from the network.
- High availability is provided by clustering software or DNS/load-balancer across multiple deployments/sites.

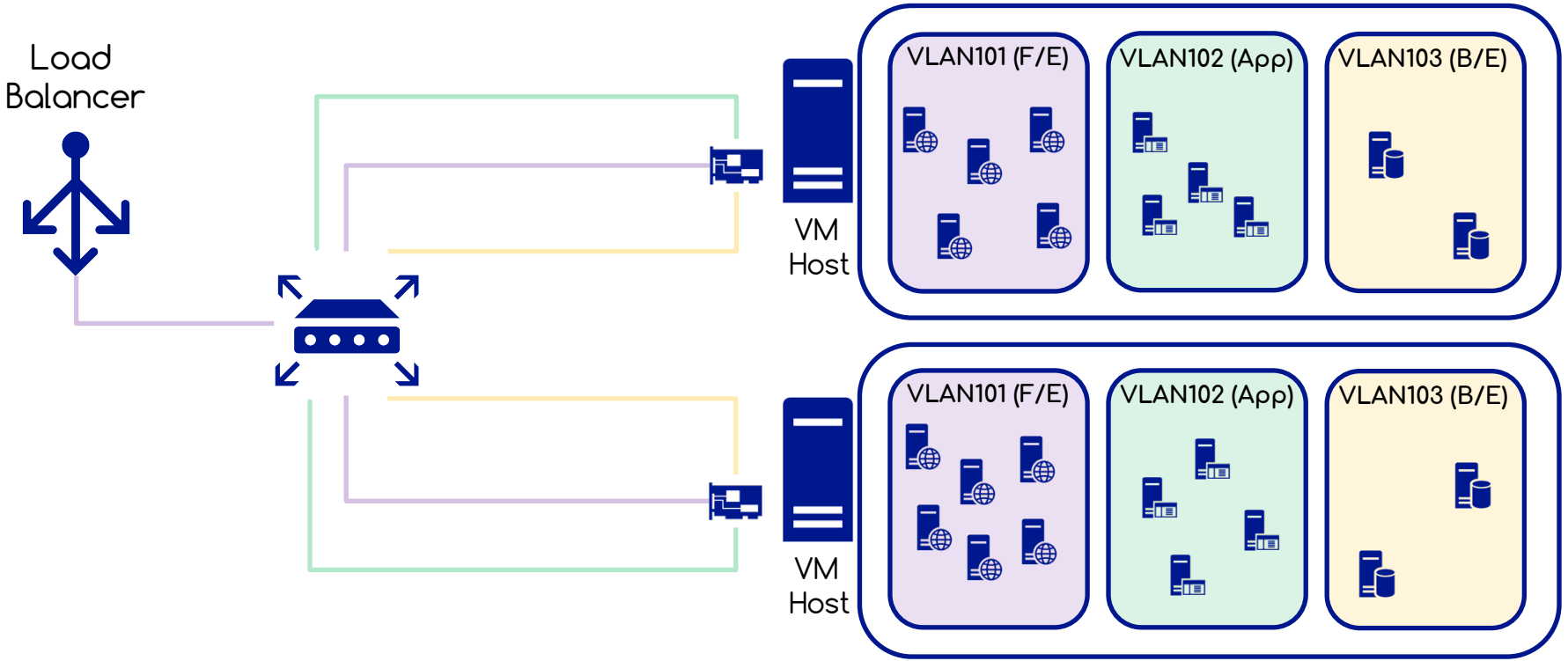
Physically hosted applications



Virtual (Machine) applications

- Services and Applications are broken down into smaller VM allocations resulting in an explosion of network resources
- The tight-packing of numerous VMs per host has resulted in numerous networks being provisioned to every host.
- Virtual LANs are used as the method for providing segregation between applications and application tiers.

Virtual (Machine) applications



Docker networking

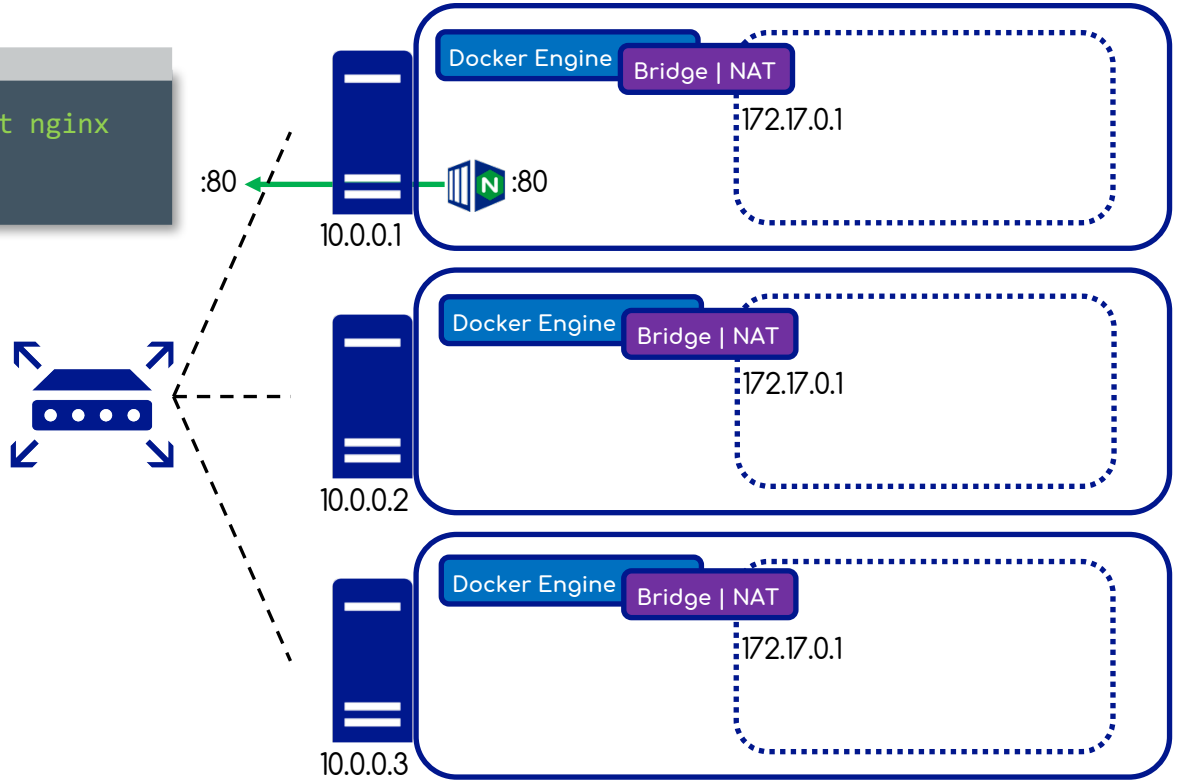
Docker Networking

```
[dan@dockercon ~]$ docker network ls

NETWORK ID          NAME                DRIVER              SCOPE
4507d8b4dd86        bridge              bridge              local
8866a19c0751        docker_gwbridge     bridge              local
b88e79e31749        host                host                local
vlujsum8my0u        ingress             overlay             swarm
e12df2f39d06        none                null                local
ed60df3f6402        mac_net             macvlan             local

[dan@dockercon ~]$
```

Host/Bridge Networking

```
[dan@dockercon ~]$ docker run --net=host nginx

[dan@dockercon ~]$
```

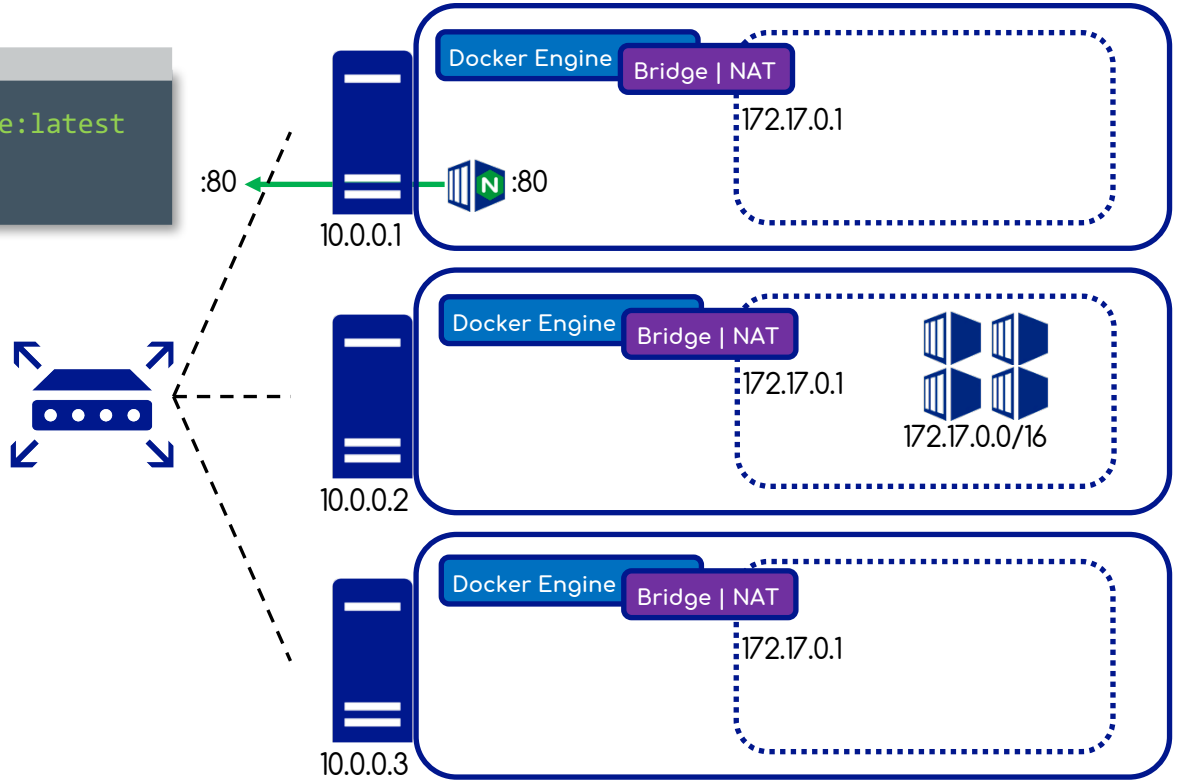
- The host flag will start the container in the same namespace as the host itself allowing a container to use the hosts networking stack directly.
- Provides near metal speed, however can result in port conflicts.



Host/Bridge Networking

```
[dan@dockercon ~]$ docker run dockerimage:latest

[dan@dockercon ~]$
```

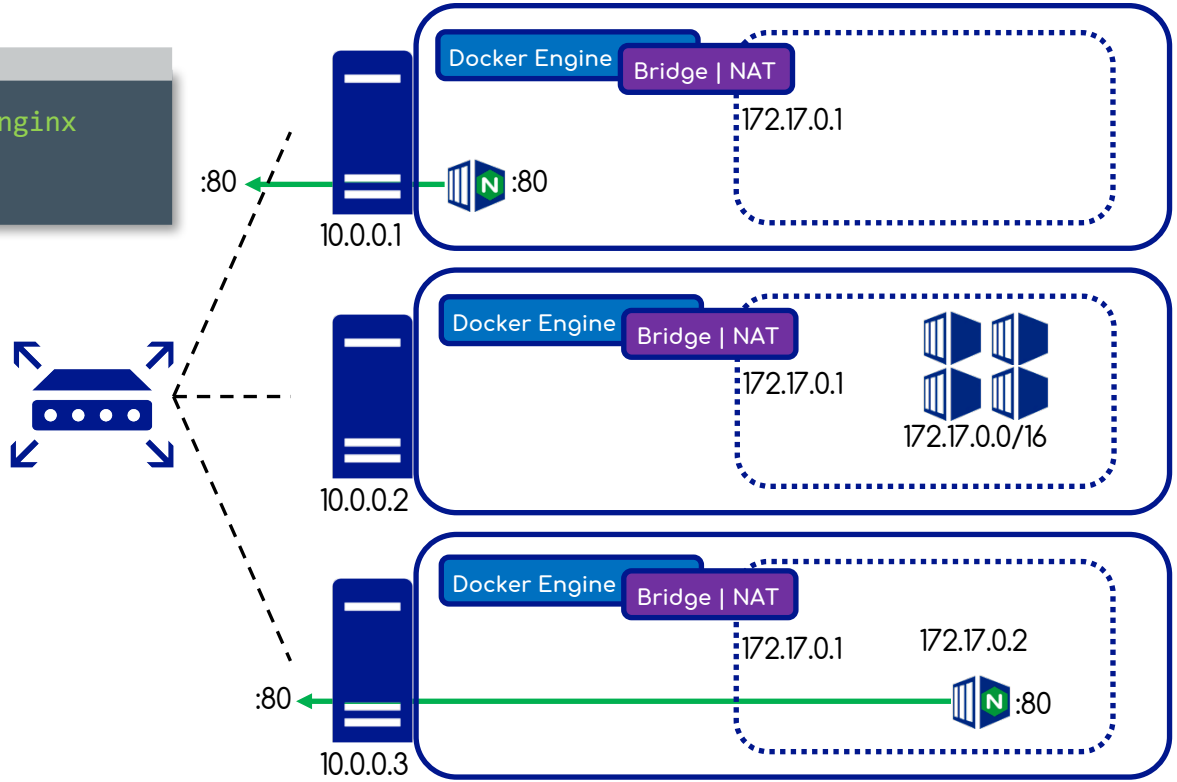
- Containers are started and connected by default to the internal bridge network.
- These containers wont expose any network connectivity to the outside world by design, however can speak to one another whilst on the same host.



Host/Bridge Networking

```
[dan@dockercon ~]$ docker run -p 80:80 nginx

[dan@dockercon ~]$
```

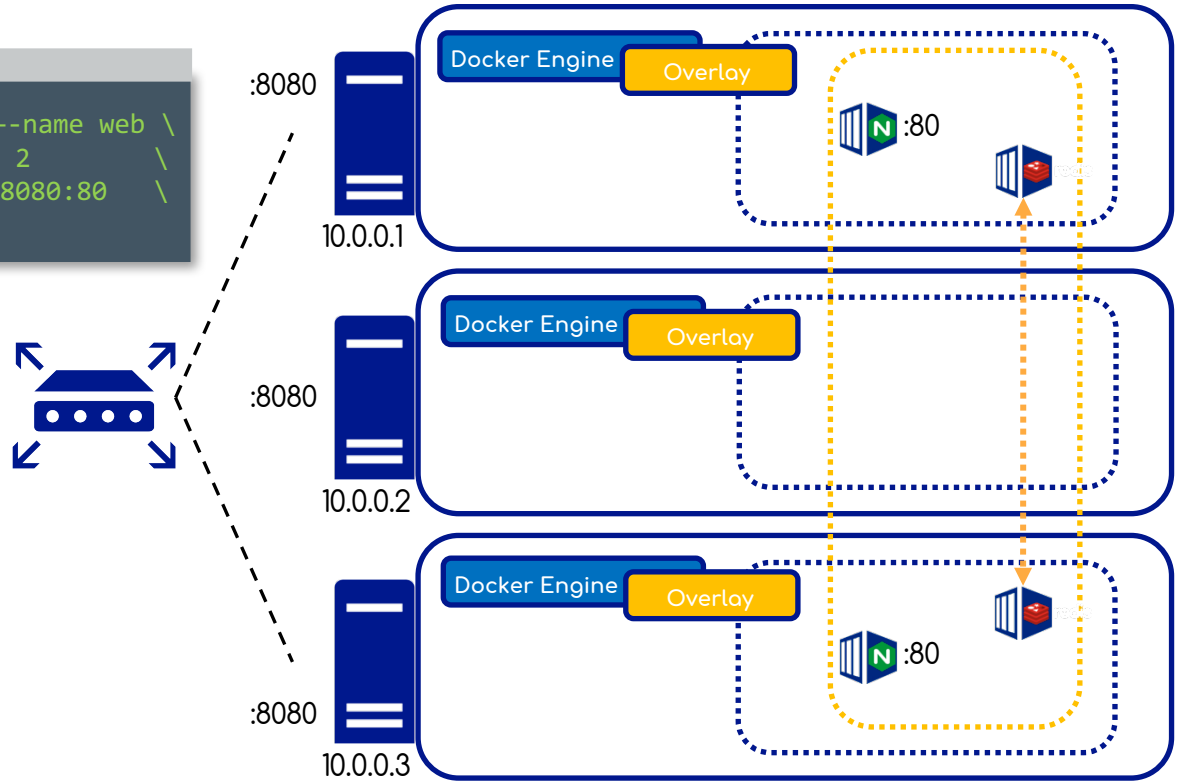
- The –p flag will expose an external port on the host and map it to a port on the container.
- Only containers with services need to expose their ports potentially solving port-conflicts.



Swarm Overlay networking

```
[dan@dockercon ~]$ docker service create --name web \
                                --replicas 2        \
                                --publish 8080:80   \
                                nginx
```

- The Overlay network makes use of VXLAN in order to create an overlay network over the underlying network.
- The tunnel allows containers across hosts to communicate.

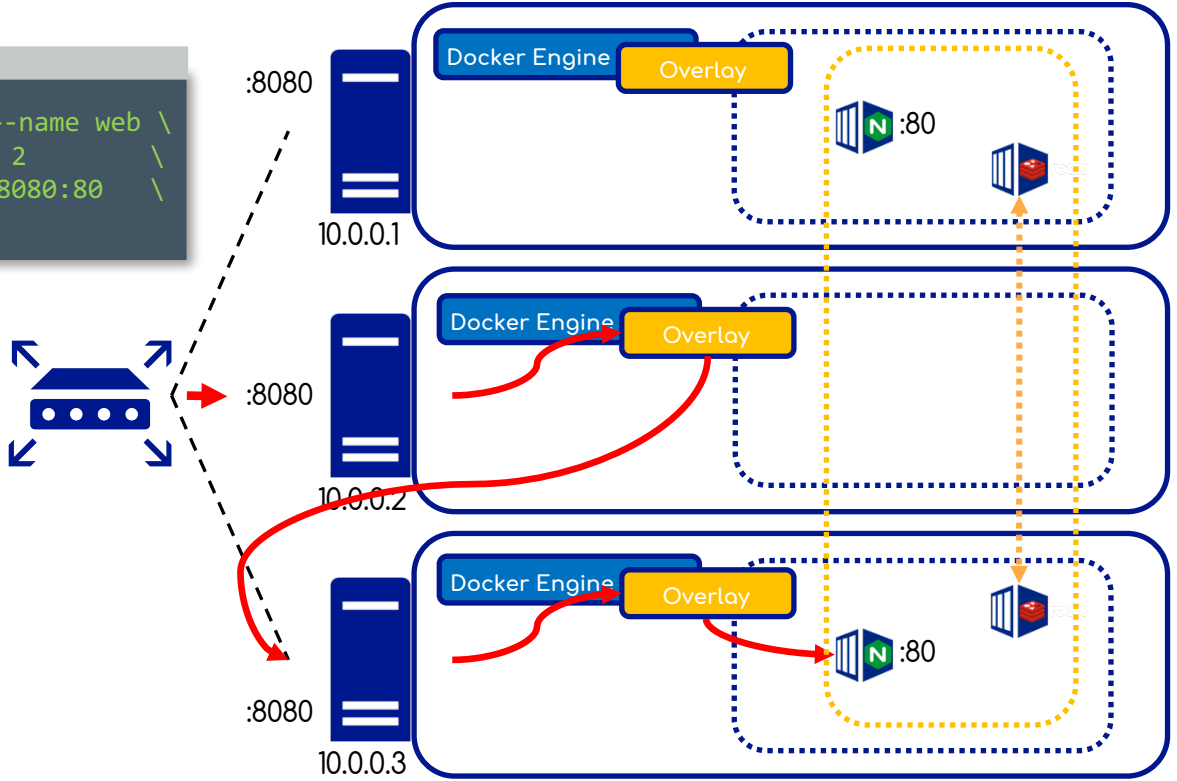


Swarm Overlay networking

```
[dan@dockercon ~]$ docker service create --name web \
                                --replicas 2        \
                                --publish 8080:80   \
                                nginx
```

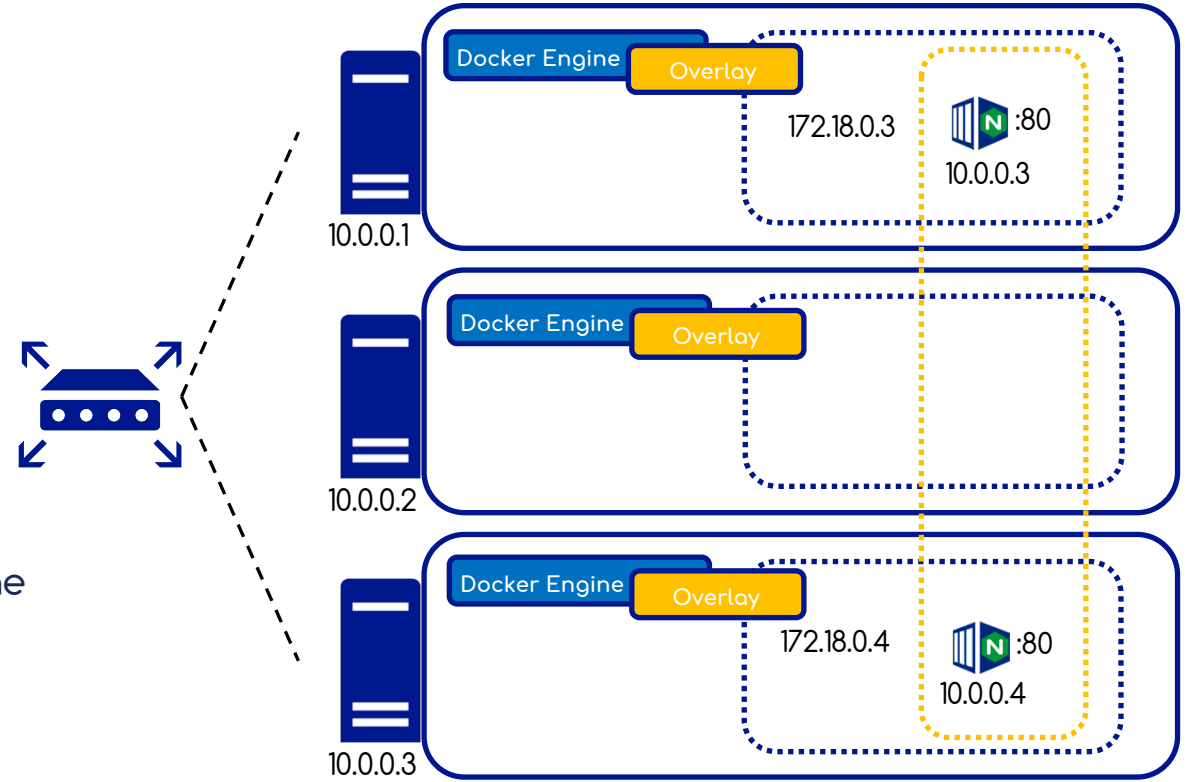
and hosts will rotate their keys every 12 hours.

- **Publishing** a port applies to all nodes in the swarm cluster. Regardless of node connected to, the request is forwarded to a node running the task.



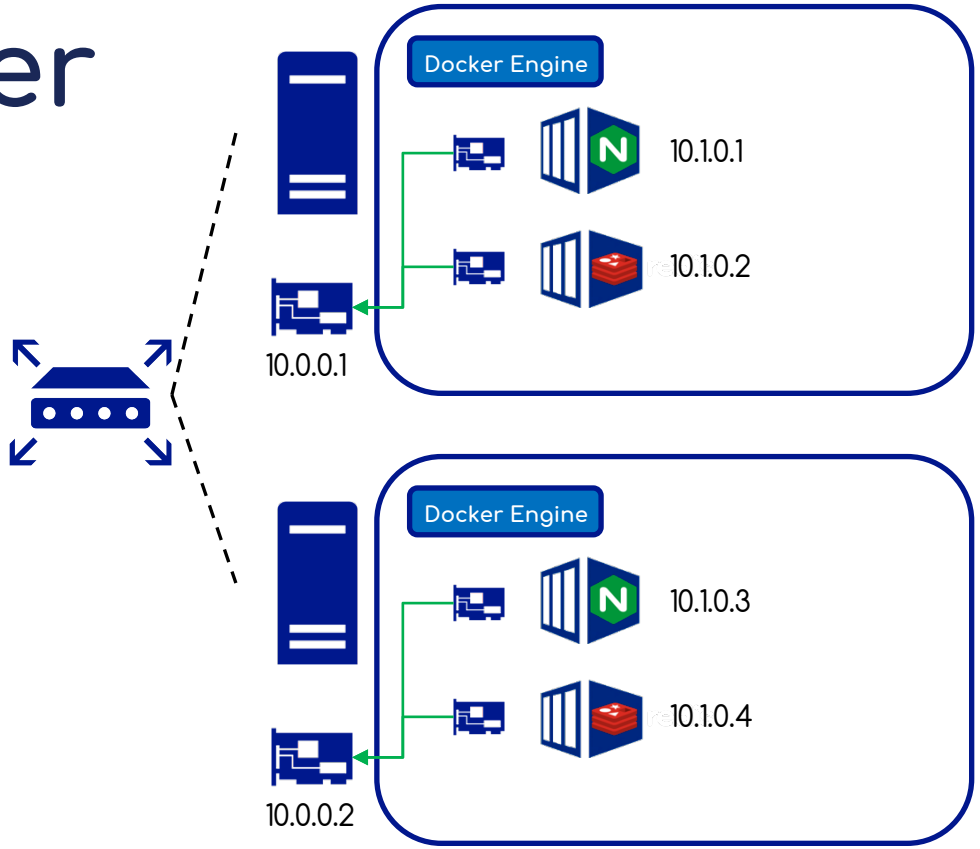
Swarm Overlay networking

- Each container gets a pair of IP addresses.
- One IP address exists on the Overlay network, this allows all containers on the network to communicate
- The other IP address carries the tunnel to other hosts in the cluster and contains all the actual data that needs to leave the host.



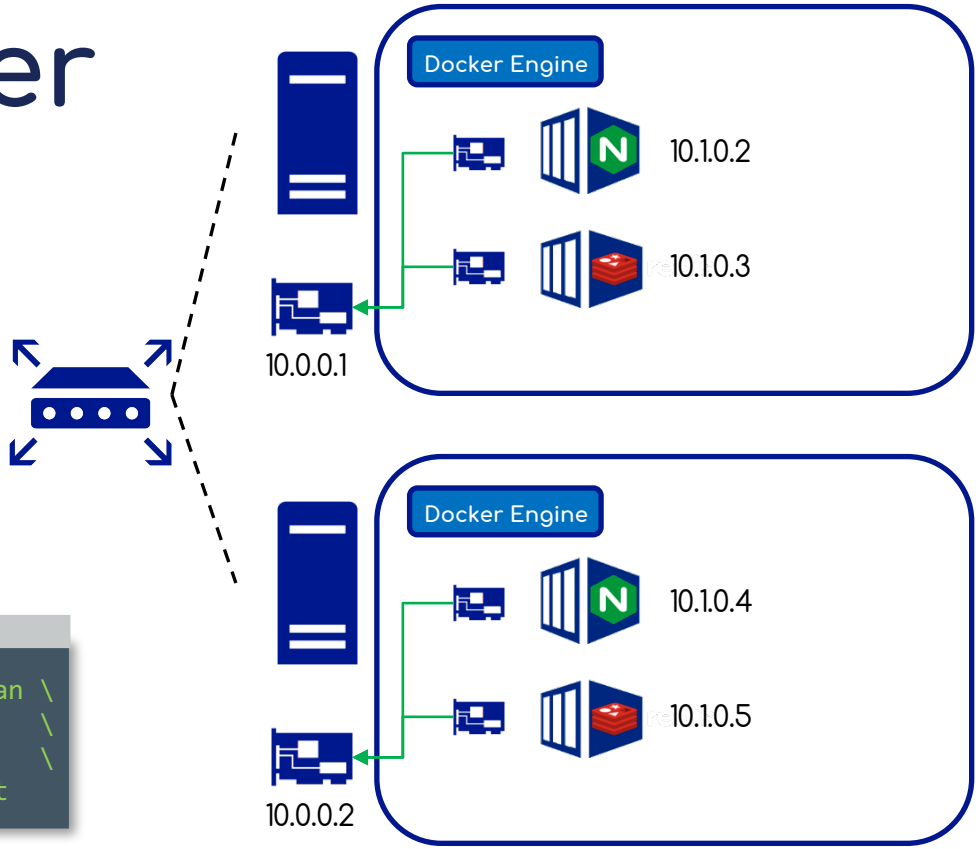
Macvlan driver

- The Macvlan driver provides a hardware (MAC) address for each container, allowing them to have a full TCP/IP stack.
- Allows containers to become part of the traditional network, and use things like external IPAM or VLAN trunking when numerous networks are needed.
- No overhead from technologies such as VXLAN or NAT.



Macvlan driver

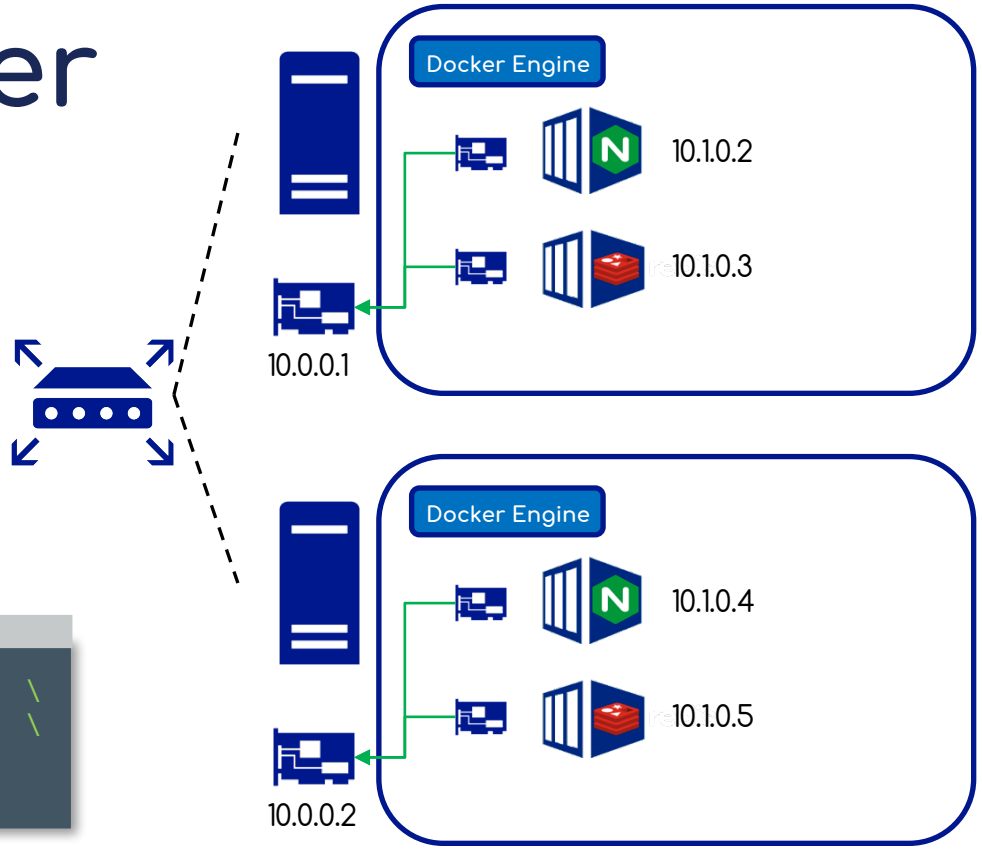
- Create a network using the macvlan network and assign the ranges/gateway and the parent adapter (or sub-adapter for vlans e.g eth0.120)




```
[dan@dockercon ~]$ docker network create -d macvlan \
                        --subnet=10.1.0.0/24    \
                        --gateway=10.1.0.1      \
                        -o parent=eth0 mac_net
```

Macvlan driver

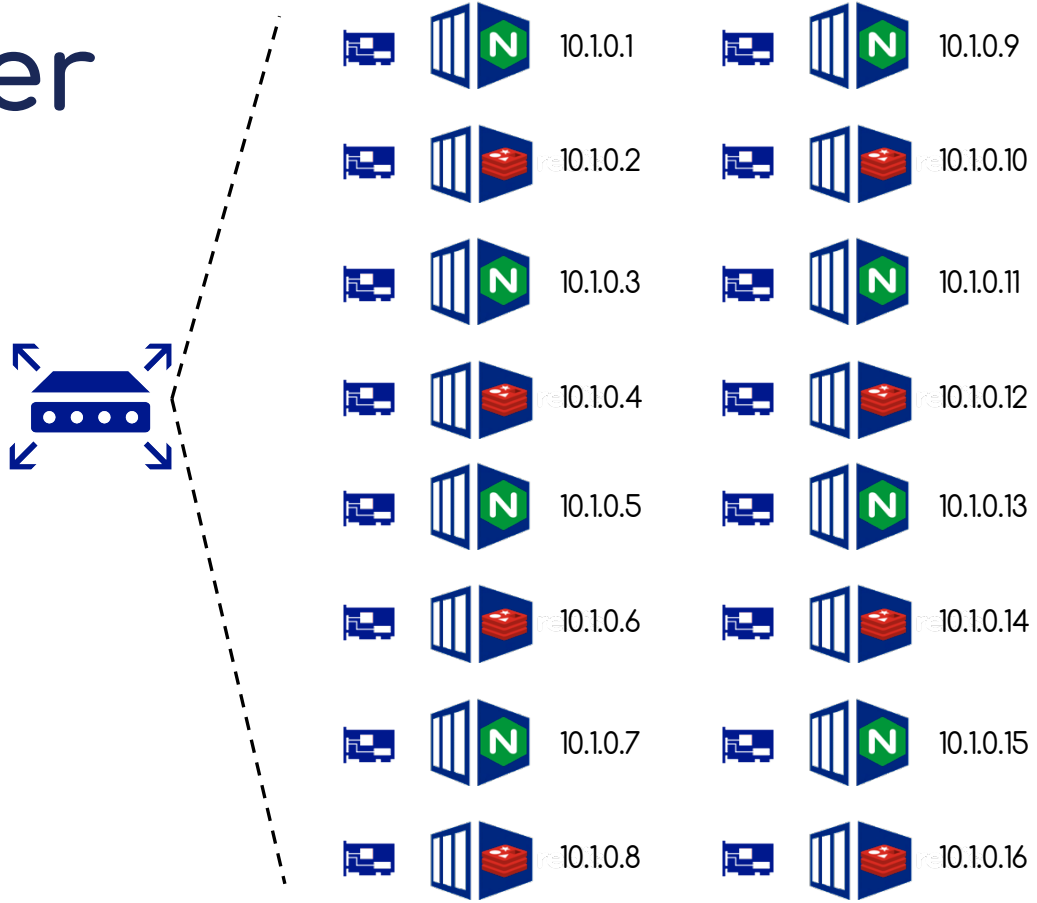
- When starting a container you can apply a physical IP address on that network.
- The container is effectively another host on the underlay network.




```
[dan@dockercon ~]$ docker run --net=mac_net        \
                           --ip=10.1.0.2           \
                           nginx
```

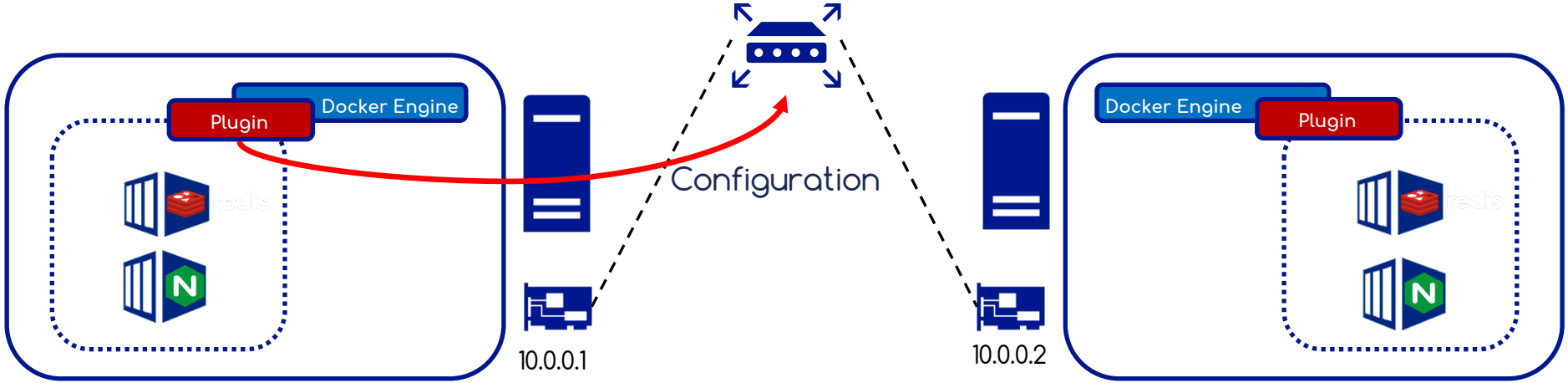
Macvlan driver

- The use of the macvlan driver essentially makes a Docker container a first class citizen on the network.
- This functionality however carries additional overhead in terms of network management, as each container will now exist on the network as its own entity.



Networking plugins

- Docker networking plugins allow vendors to extend the functionality of their network devices and technologies into the Docker Engine.
- Providing features such as vendor specific IP Address Management or enabling the network to configure itself to provide functionality to containers through their lifecycle such as (overlays/QOS/Load balancing).

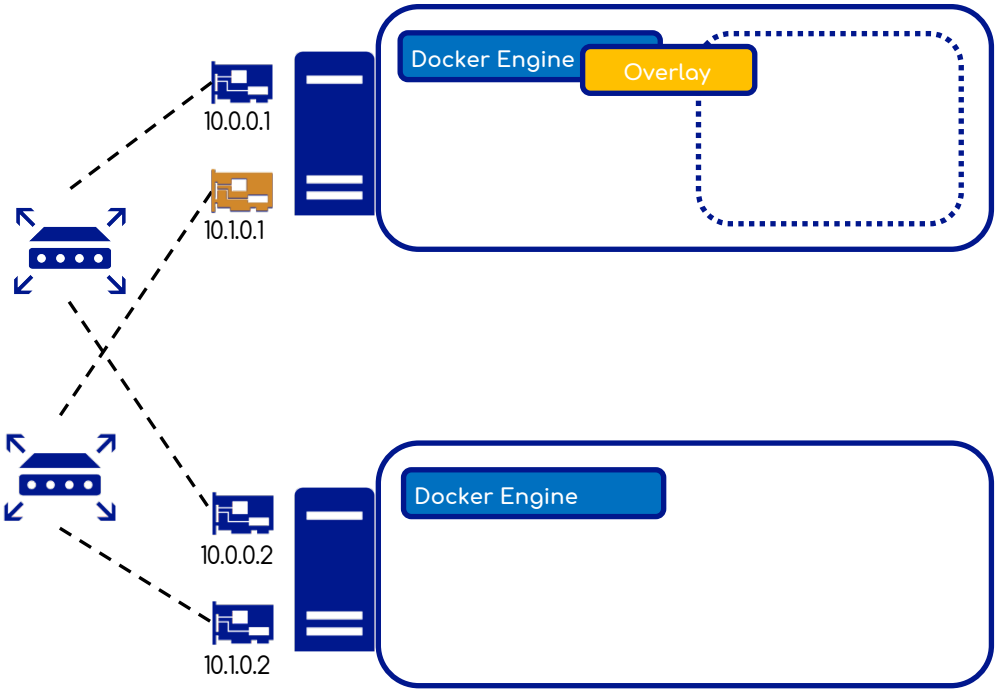


Infrastructure design patterns

Separate data/control planes

```
[dan@dockercon ~]$ docker swarm init \
                   --advertise-addr eth0 \
                   --data-path-addr eth1
```

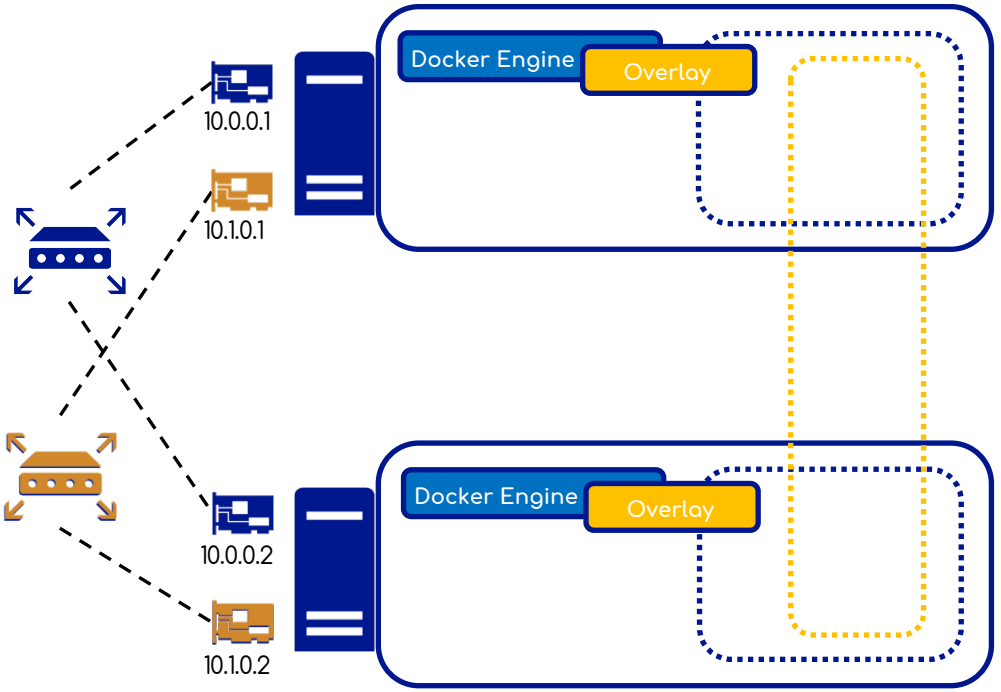
- When initially configuring a Docker swarm cluster on hosts with multiple NICs there is the option of separating the data and control planes.
- This provides physical and logical separation of traffic leaving the host.



Separate data/control planes

```
[dan@dockercon ~]$ docker swarm join \
                   --token XYZ --advertise-addr eth0 \
                   --data-path-addr eth1 \
                   10.0.0.1:2377
```

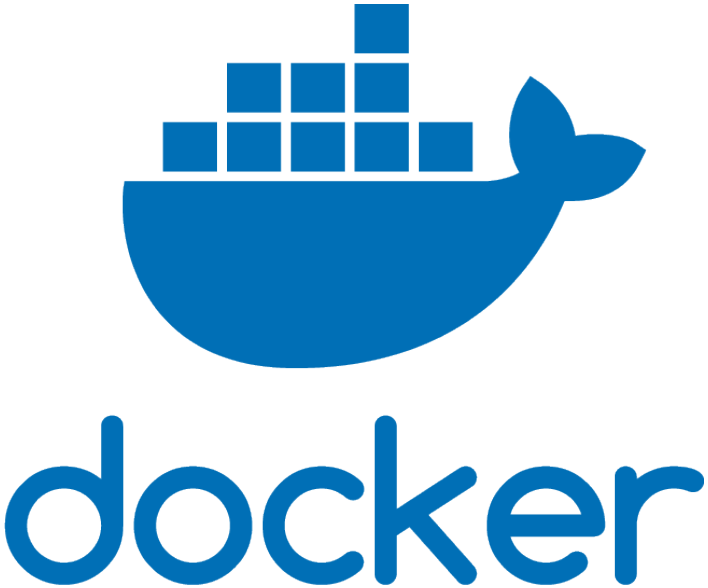
- Joining additional nodes to the swarm cluster takes two additional flags to specify the traffic carried by a particular adapter.
- Any services created will then be part of the data plane and have traffic segregated from the control plane.



Design Patterns when modernizing a traditional application

dockercon17 EU

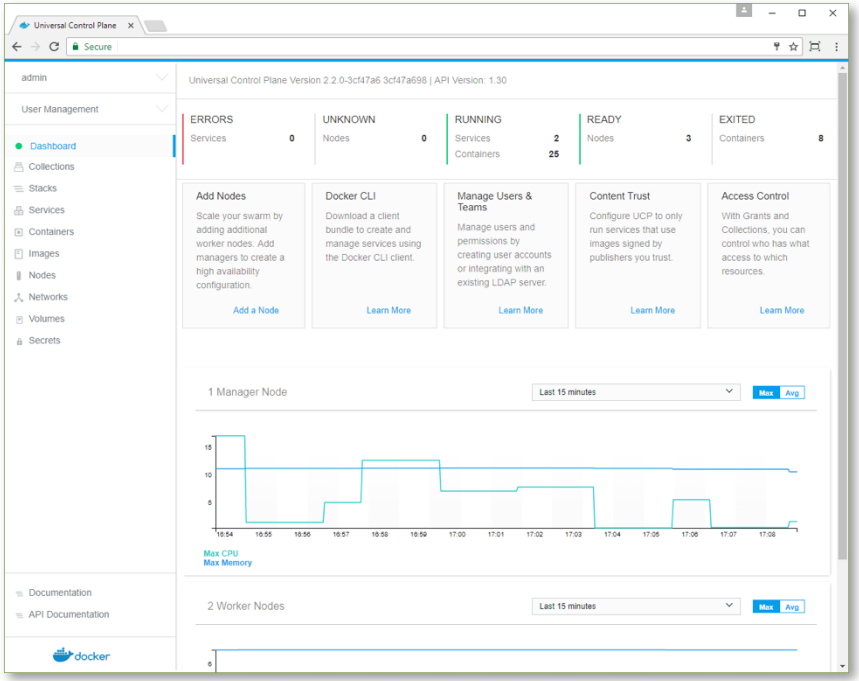
Docker Enterprise Edition



- Docker Enterprise Edition provides a full CaaS platform (Containers as a Service).
- Comes with Integrated Container Orchestration, management platform and increased security (RBAC, images scanning etc.)
- Enterprise supported platform for production deployments.

Universal Control Plane

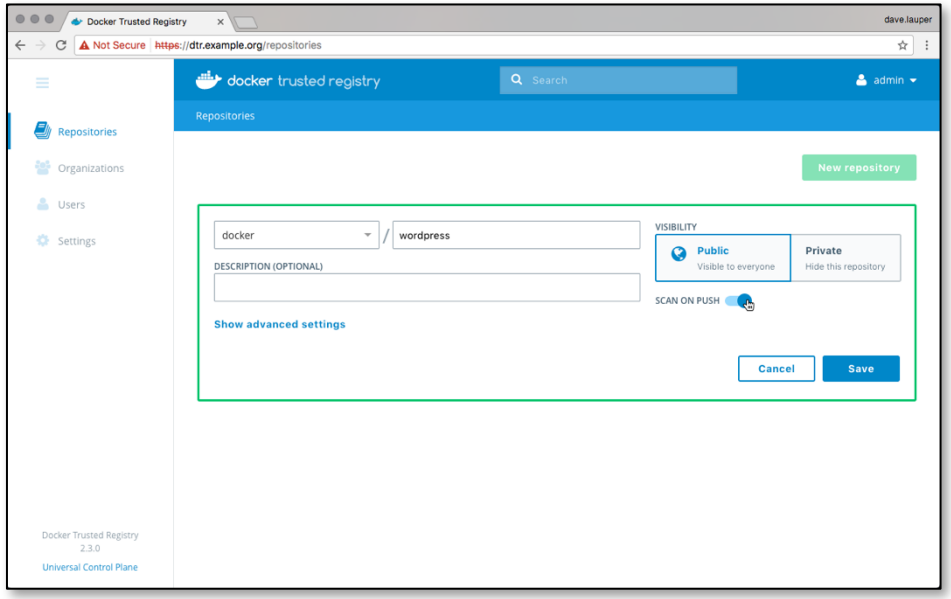
- The Docker UCP provides a clustered enterprise grade management platform for Docker.
- A centralized platform for managing and monitoring swarm container clusters and container infrastructure.
- Extended functionalisation of the Docker platform making it easier to deploy applications at scale.
- Can be controlled through the UI or through the CLI (client bundle) or through the Docker APIs.



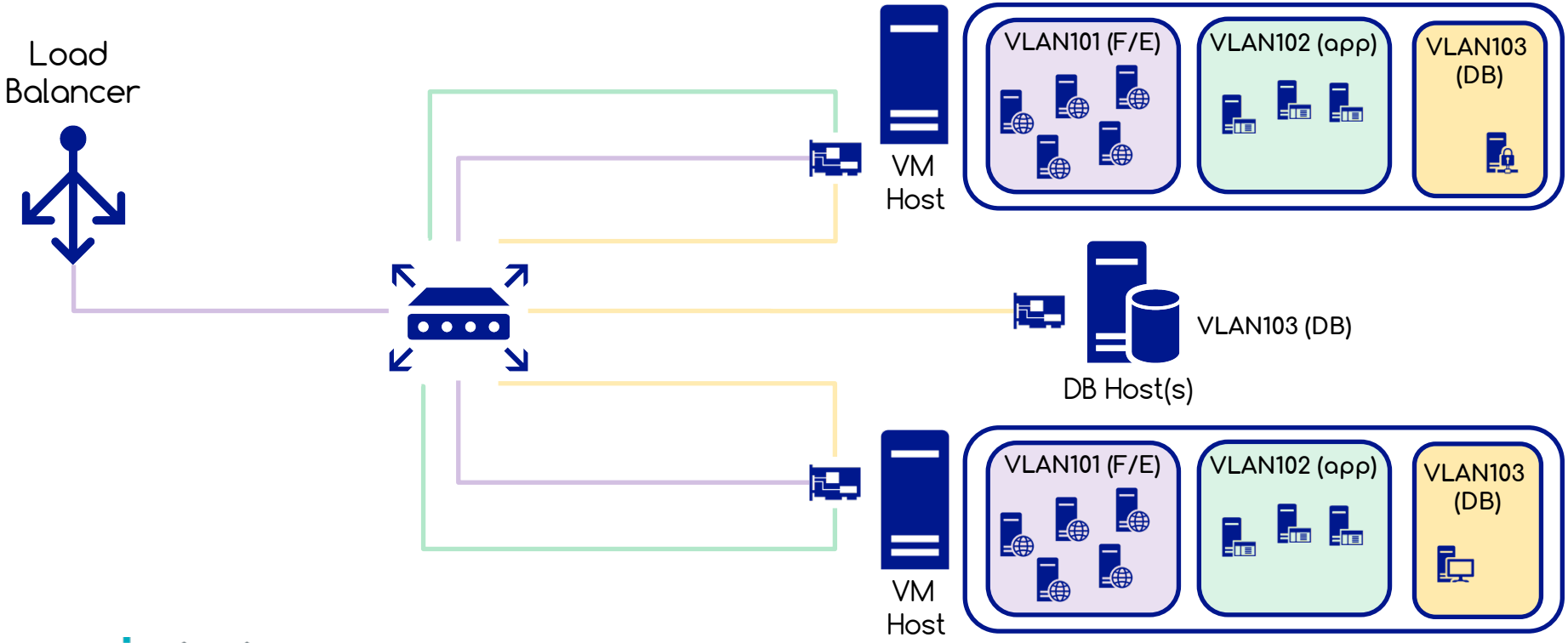
dockercon17 EU

Docker Trusted Registry

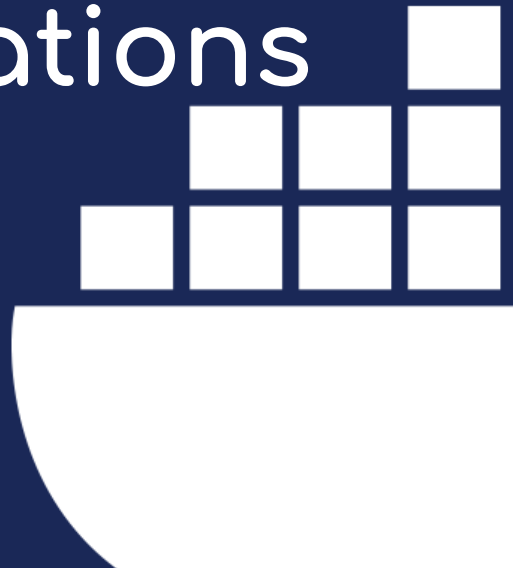
- Enterprise grade storage for all your Docker Images, allowing users to host their images locally.
- Can become part of the CI/CD processes simplifying the process to build, ship and run your applications.
- Images can be automatically scanned for vulnerabilities ensuring that only compliant images can be deployed.



Application Architecture

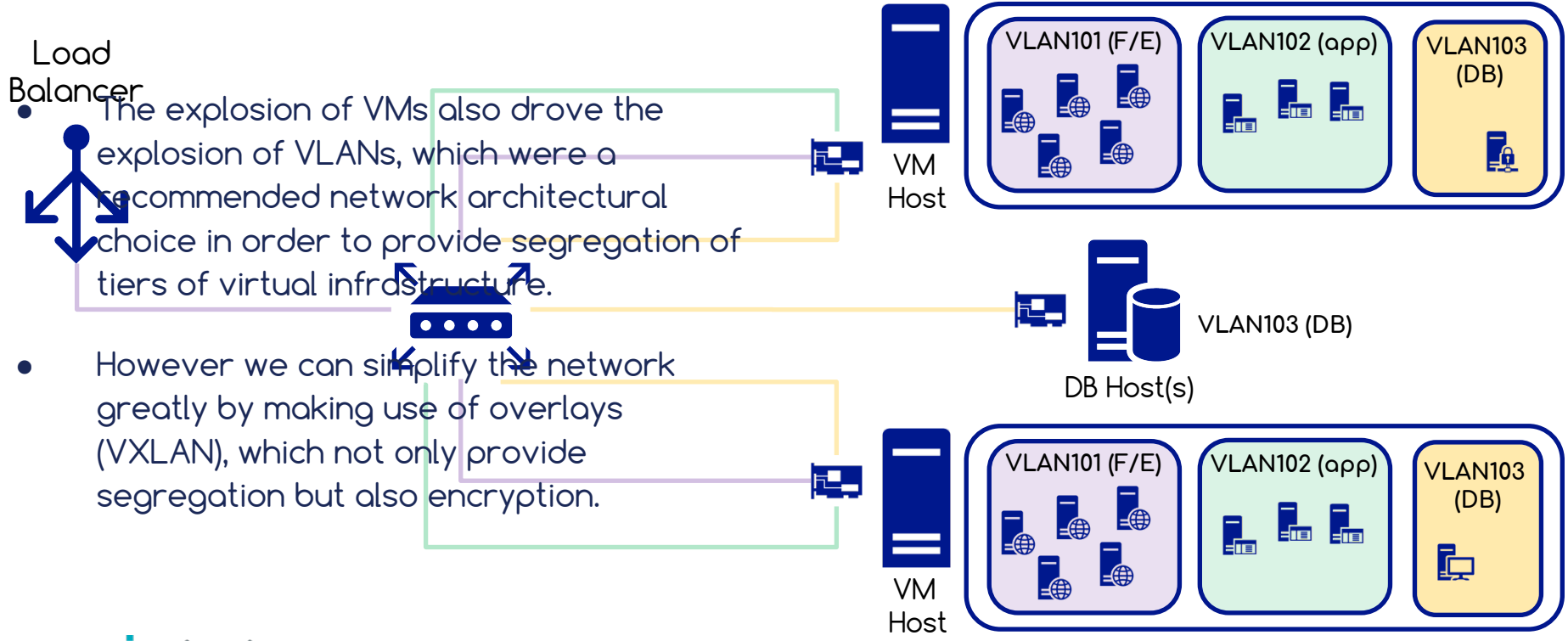


“Behind the scenes the developers and application maintainers have repackaged our applications into containers”



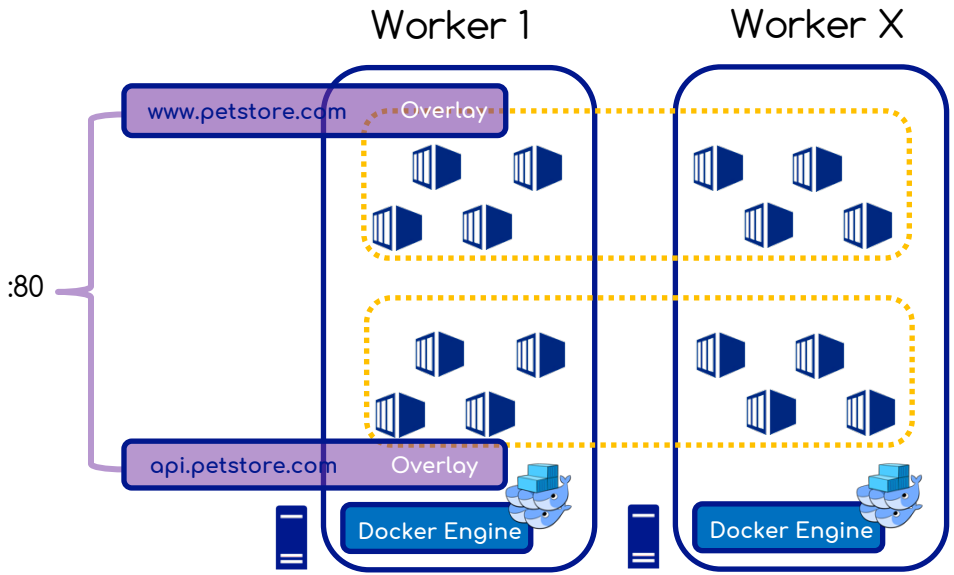
Application Architecture

- The explosion of VMs also drove the explosion of VLANs, which were a recommended network architectural choice in order to provide segregation of tiers of virtual infrastructure.
- However we can simplify the network greatly by making use of overlays (VXLAN), which not only provide segregation but also encryption.



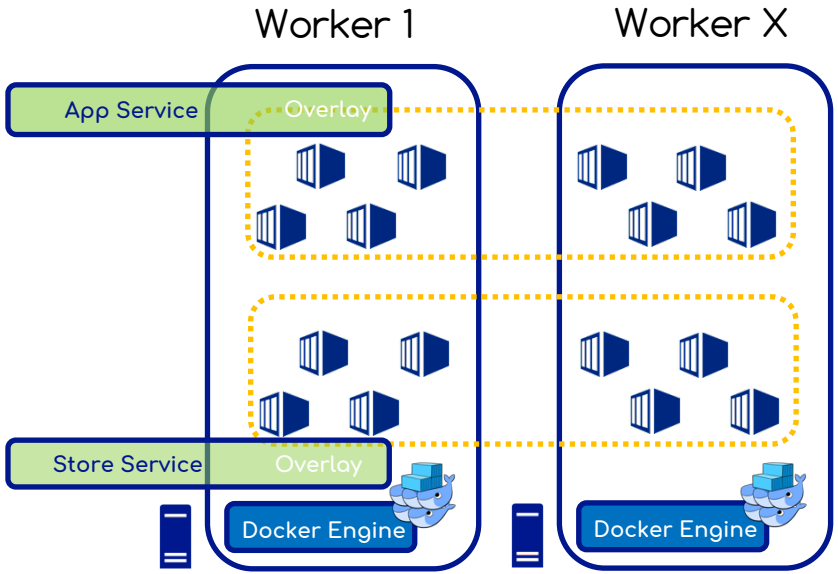
Front-End with HRM

- Docker EE provides the HTTP Routing Mesh capability, which simplifies the routing between services.
- The HRM will inspect the hostname that has been requested and route the traffic to that particular service.
- This allows multiple overlays to exist in harmony and traffic to be routed to them as requests hit the HRM port.



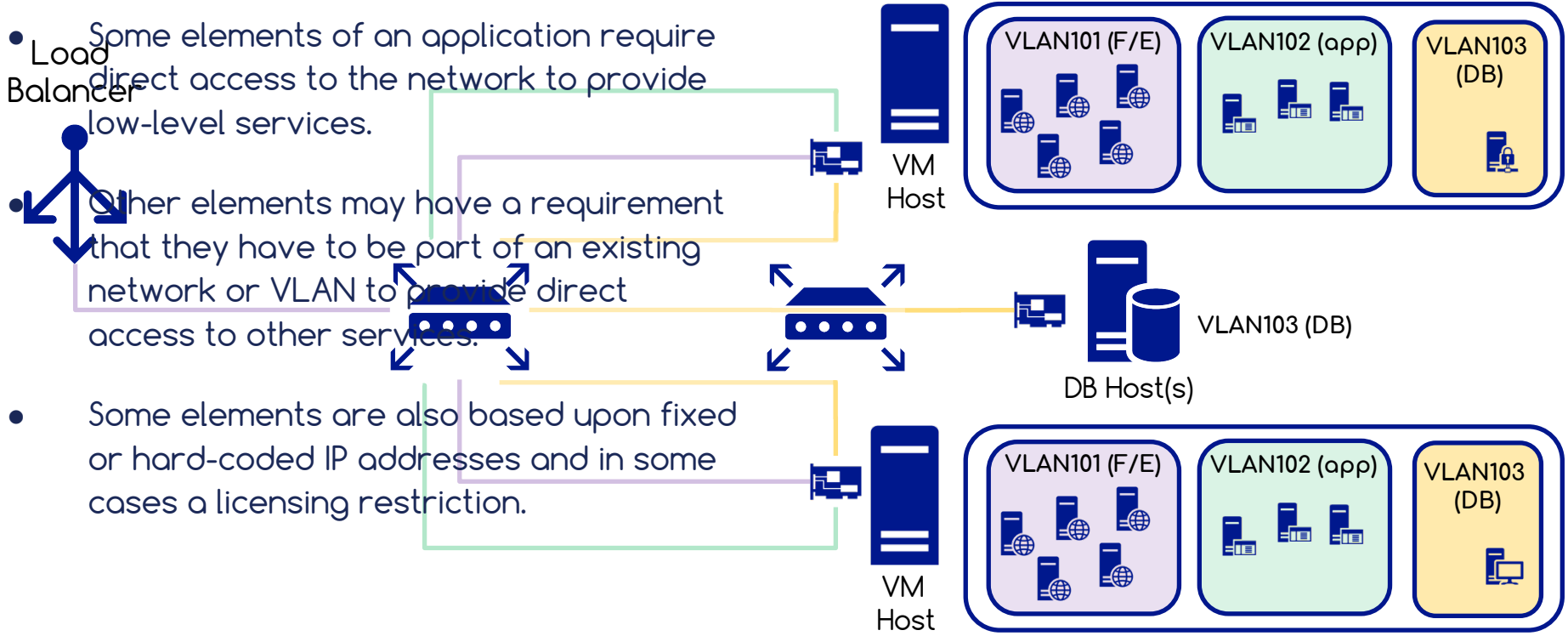
Scalable services

- Taking the existing and now packaged applications, we can deploy them as services.
- We can deploy and scale them up as needed across our cluster.
- Exposing service ports will provide load balancing across service tasks and ensure traffic is routed to where those tasks are running.

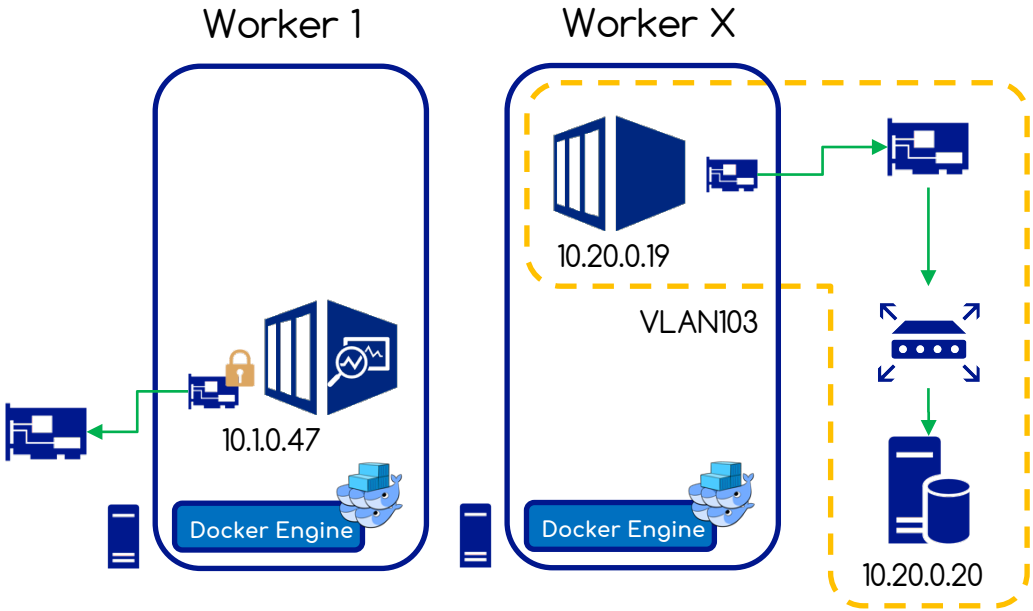


Application Architecture

- Some elements of an application require direct access to the network to provide low-level services.
- Other elements may have a requirement that they have to be part of an existing network or VLAN to provide direct access to other services.
- Some elements are also based upon fixed or hard-coded IP addresses and in some cases a licensing restriction.



Preserving existing integrations



- The Use of Macvlan allows a container with specific requirements such as packet inspection directly on the network.
- Custom singleton applications that are hardcoded to interact with databases can make use of their original IP addresses and be part of the same segregated VLAN in which the database server(s) reside.

Design Patterns

- Where possible, there is a great opportunity to provide simplification of networking.

- The use of overlays (VXLAN) is all handled in software, providing software defined networking “as code”. This also has the additional benefit of simplifying network device configurations.

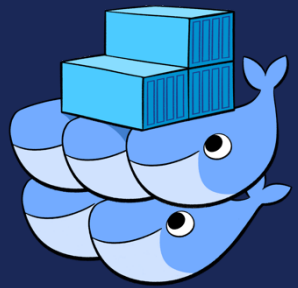
- Overlay provided load balancing again is specified as part of the service design simplifying the application and the network architecture design.

- Cases where VLANs or hard pinned IP connectivity are required can be met through the use of containers attached through macvlan.

Explore the hands on labs in the experience centre for some real experience.

Upcoming networking with the Universal Control Plane

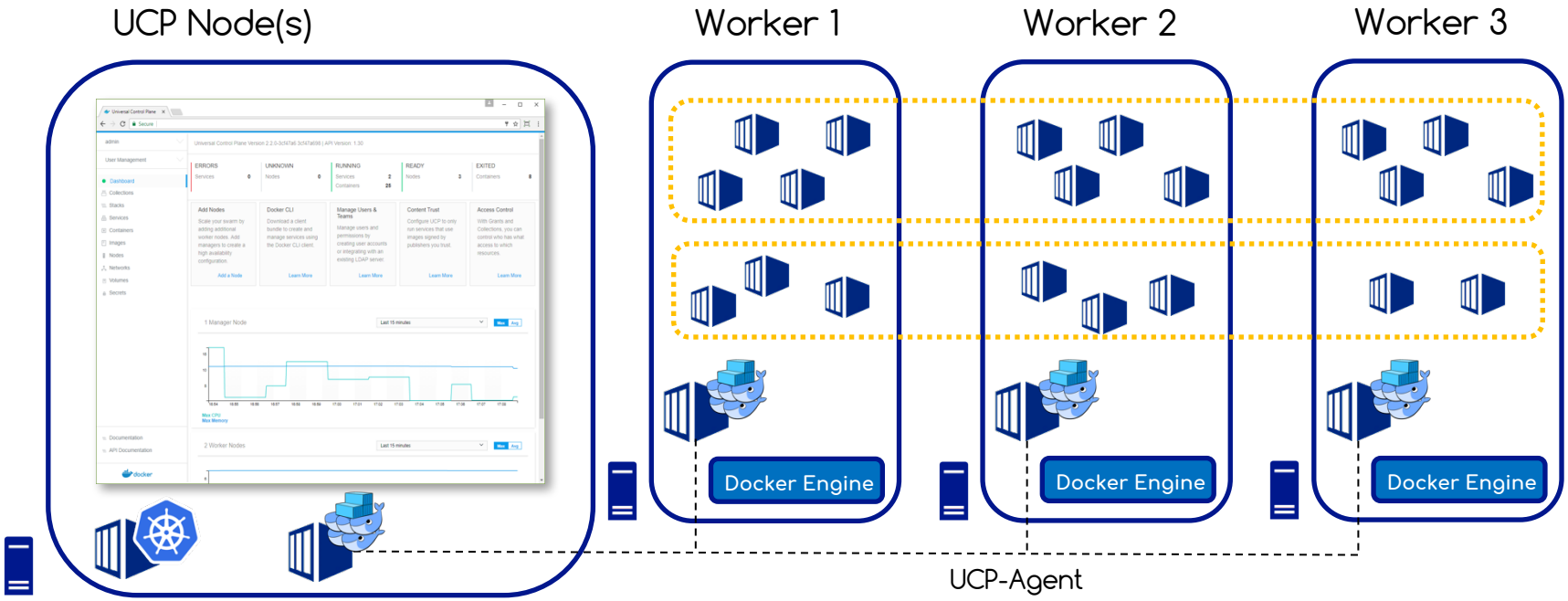
dockercon17 EU



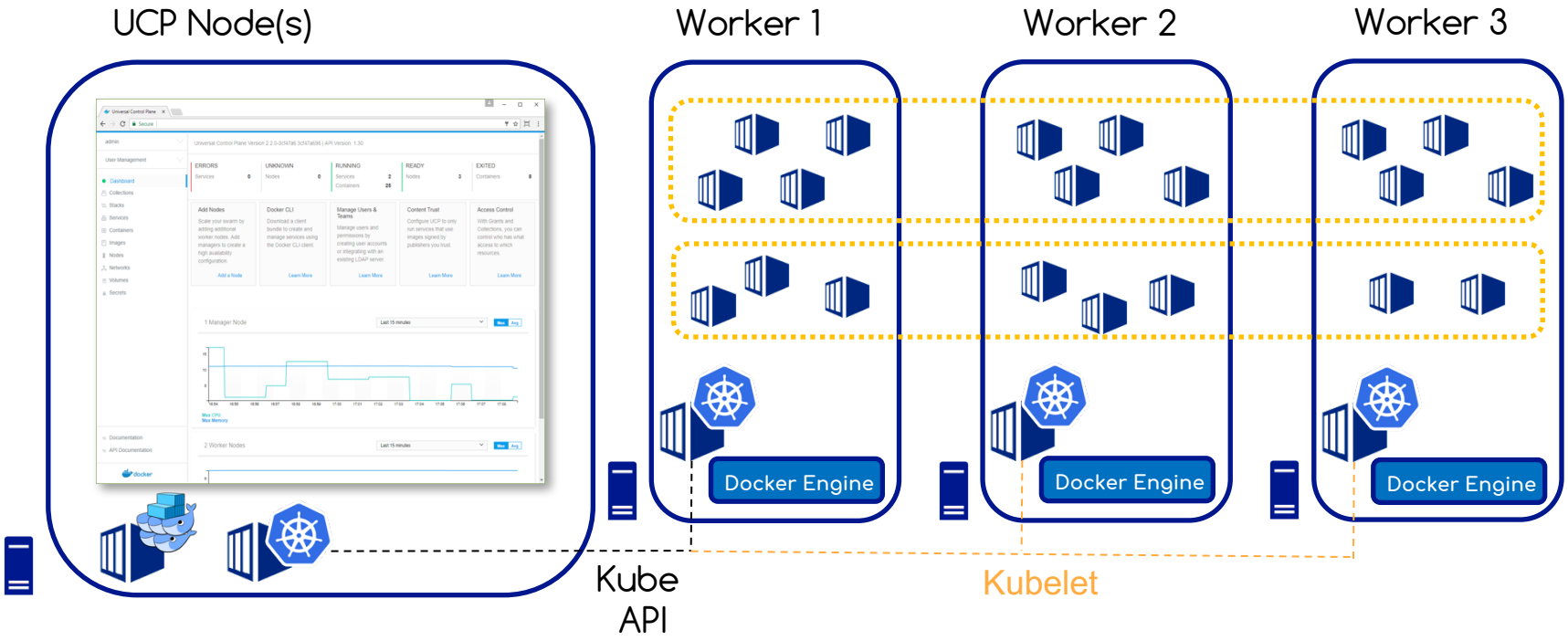
“Disclaimer”


Community Chest
GET OUT
OF JAIL, FREE
THIS CARD MAY BE KEPT UNTIL NEEDED OR SOLD
© 1935 PARKER BROTHERS, INC.

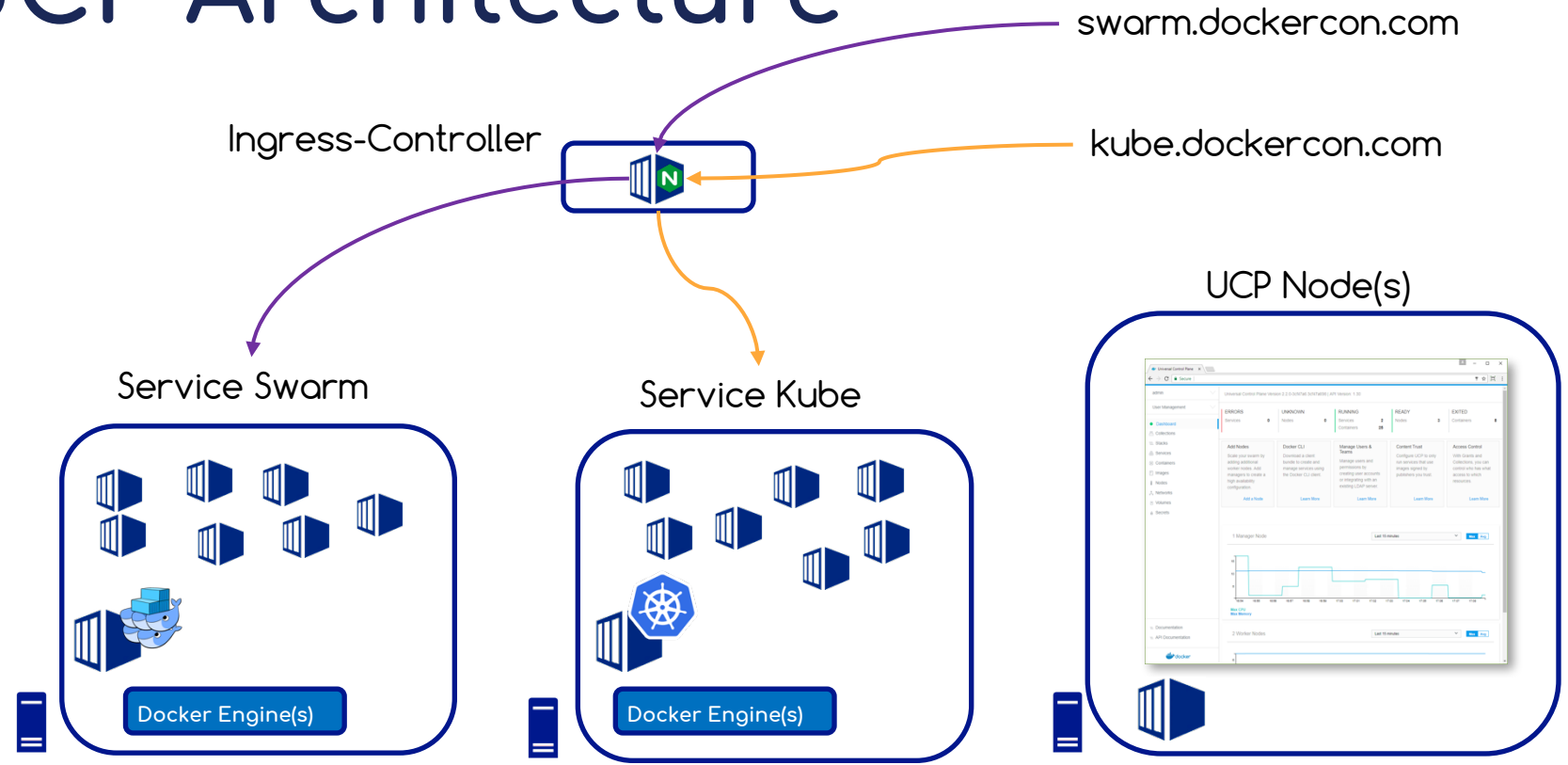
UCP Architecture



UCP Architecture



UCP Architecture



Summary

- Applications that can be re-homed on a network can make use of Docker networking features that will simplify their deployment and their scaling.
- Overlay networks provide the capability to place workloads through the cluster without the headache of having to be aware of task location.
- Services that are tied or hard coded to specific network requirements can still be deployed in containers.

Interested in MTA

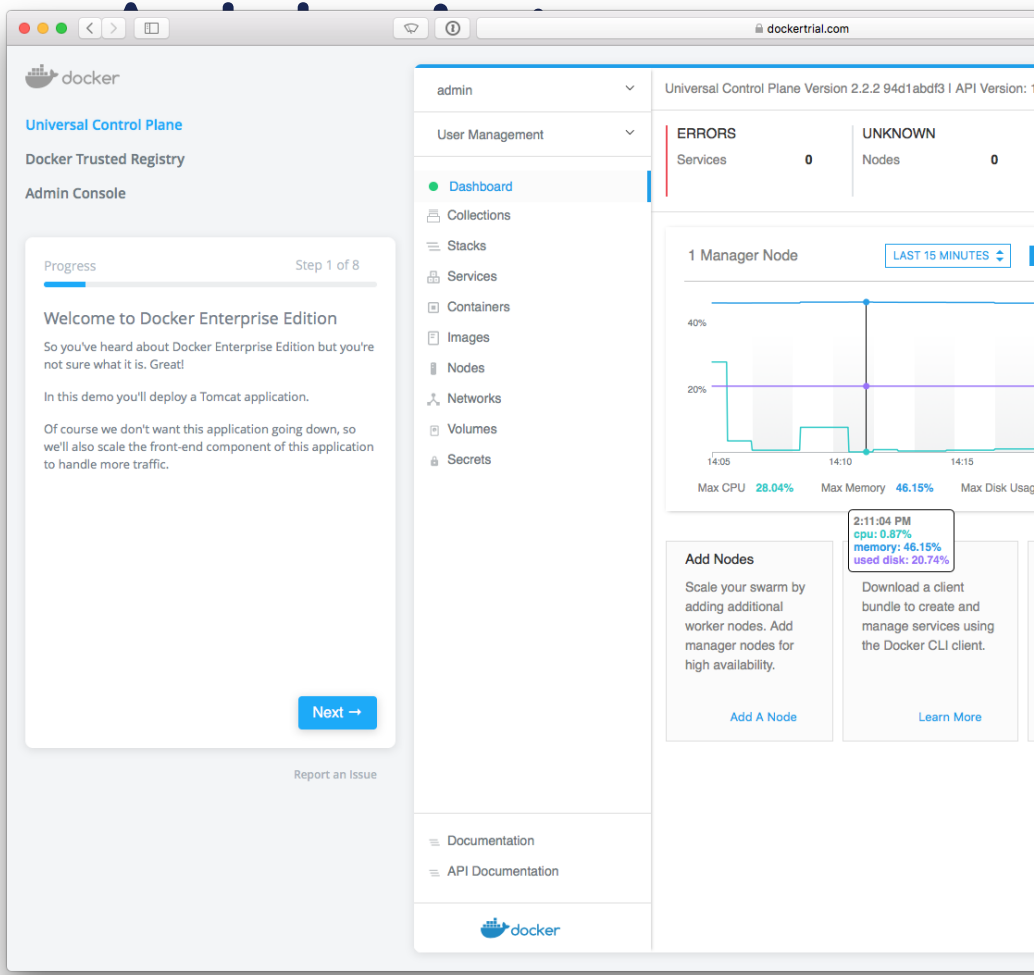
- Stop by the booth (MTA pod)
- Download the kit www.docker.com/mta
- Look for a MTA Roadshow near you
- Contact your Account Team

Docker EE Hosted Demo

docker.com/trial

- Free 4 Hour Demo
- No Servers Required
- Full Docker EE Cluster Access

dockercon17 EU



Practical Design Patterns in Docker Networking

Dan Finneran

@thebsdbox

dockercon17 EU

Q/A

```
[dan@dockercon ~]$    cat docker-compose.yaml
version: "3.1"
services:
   migrated-application:
     image: dockercon/frontend:1.0
     ports:
      - 8080
     networks:
      - back-end
      - ucp-hrm
     deploy:
       mode:
         replicated replicas: 5
labels:
com.docker.ucp.mesh.http.8080=external_route=http://${DOMAIN},internal_port=8080
networks:
back-end:
- driver:
- overlay ucp-hrm:
- external:
- name: ucp-hrm
```