

Tobias Fenster

CTO at COSMO CONSULT Group

Dual Microsoft MVP for Business
Applications and Azure

🐦 @tobiasfenster

RSS tobiasfenster.io

✉️ tobias.fenster@cosmoconsult.com

LinkedIn tobiasfenster



COSMO CONSULT
Business-Software for People



Introduction to the scenario

Assumption: You know roughly what Docker is

Docker containers allow running **multiple versions / CUs** of Business Central
on the **same VM**

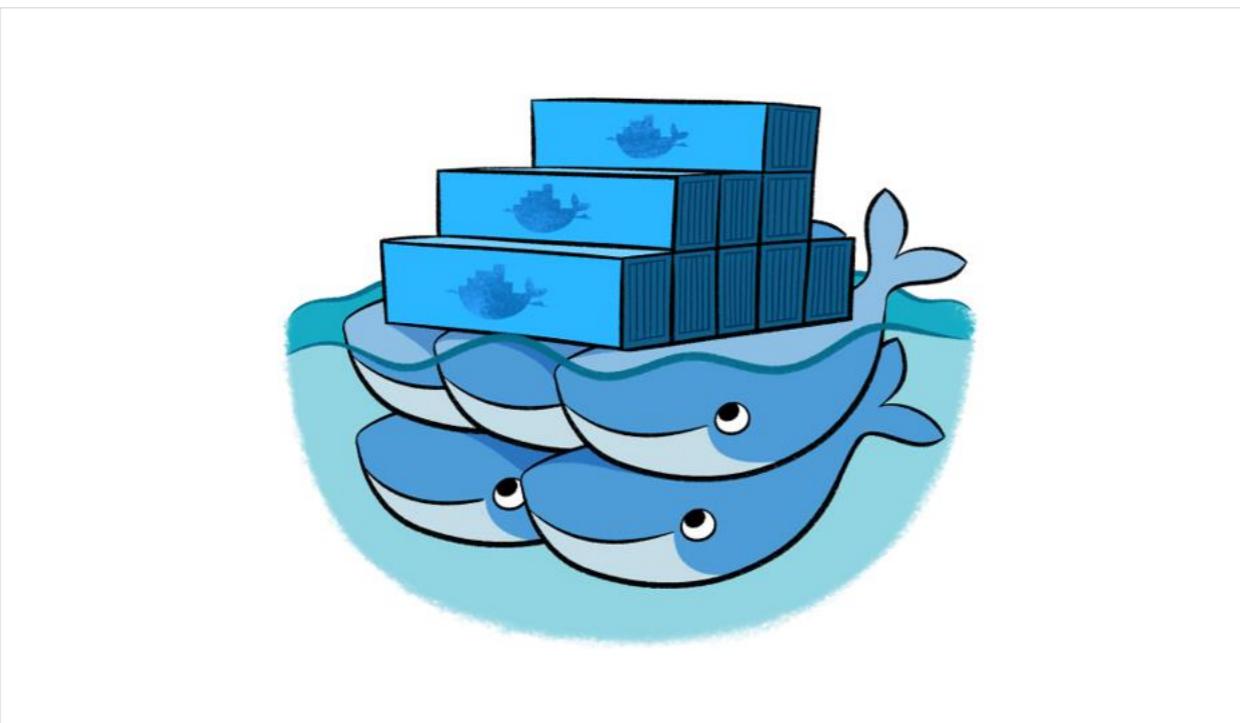
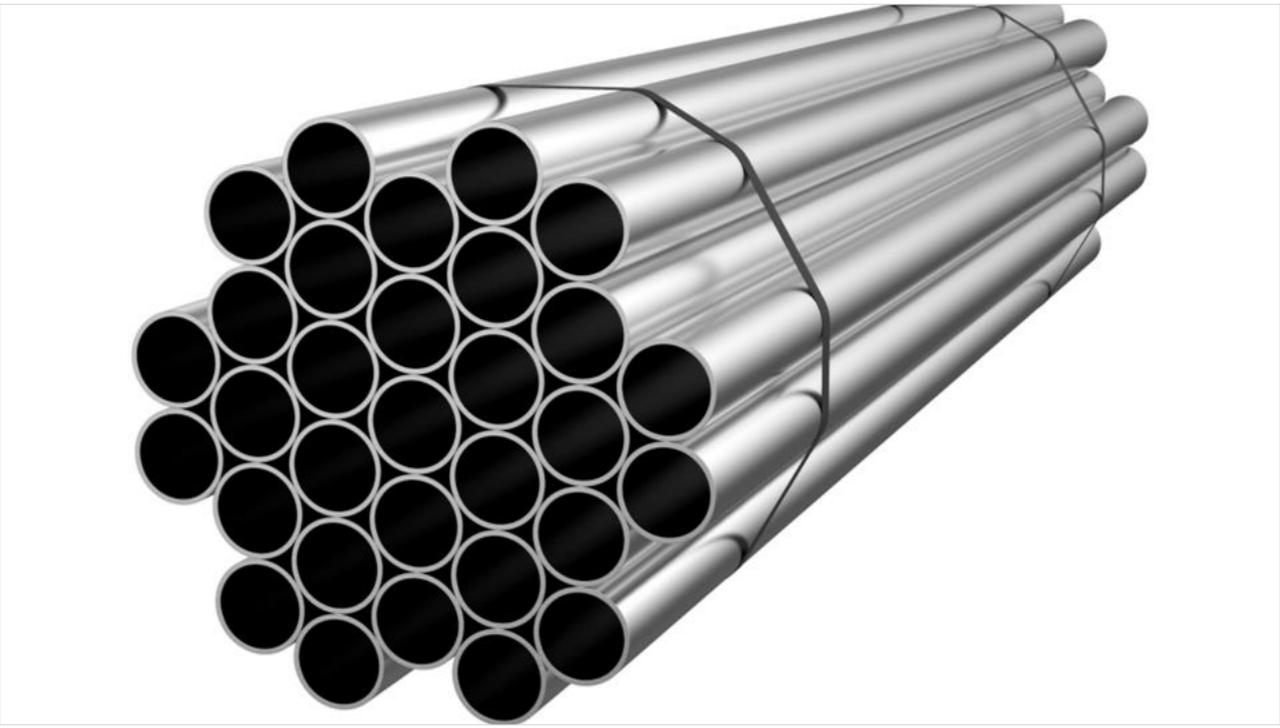
Docker containers have a **much lower resource overhead** than full VMs

Creating / starting and stopping / deleting containers is **a lot quicker** than
full VMs

→ You want to run **multiple containers** “somewhere” to save resources and
scale better

→ But **how can you connect** your development / test / etc. machines to
those containers?

Agenda



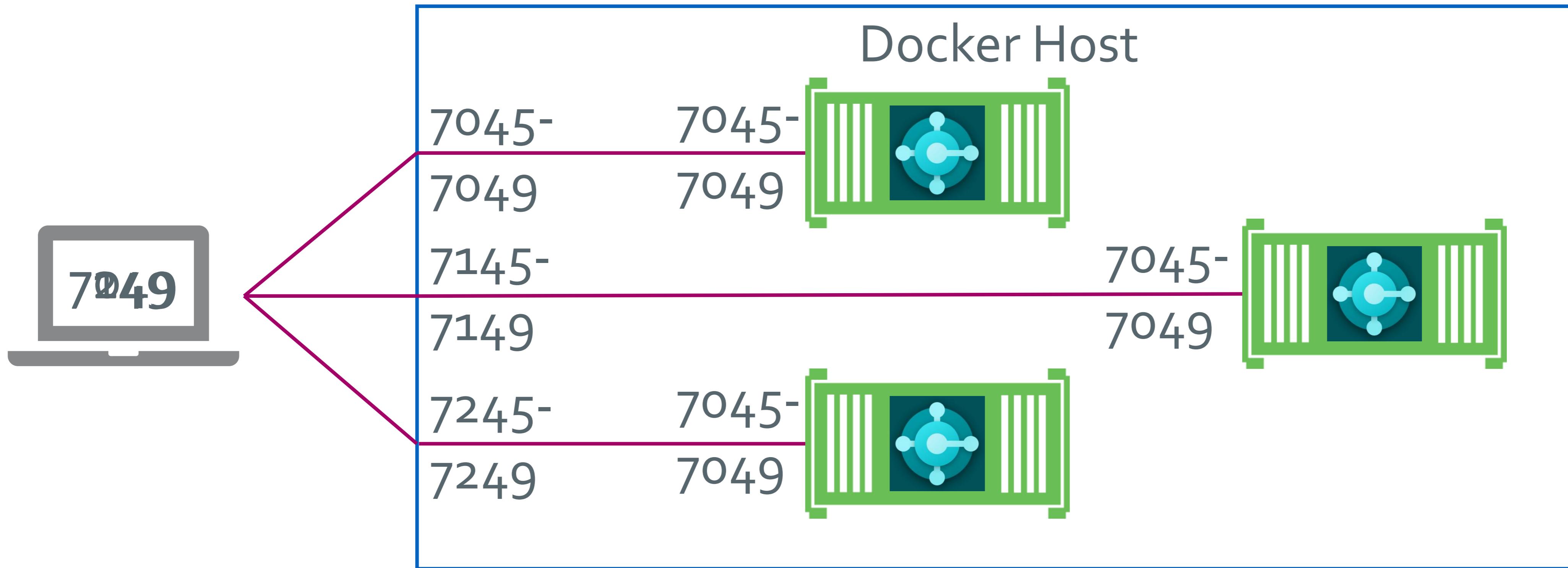




Handle every
network port
individually

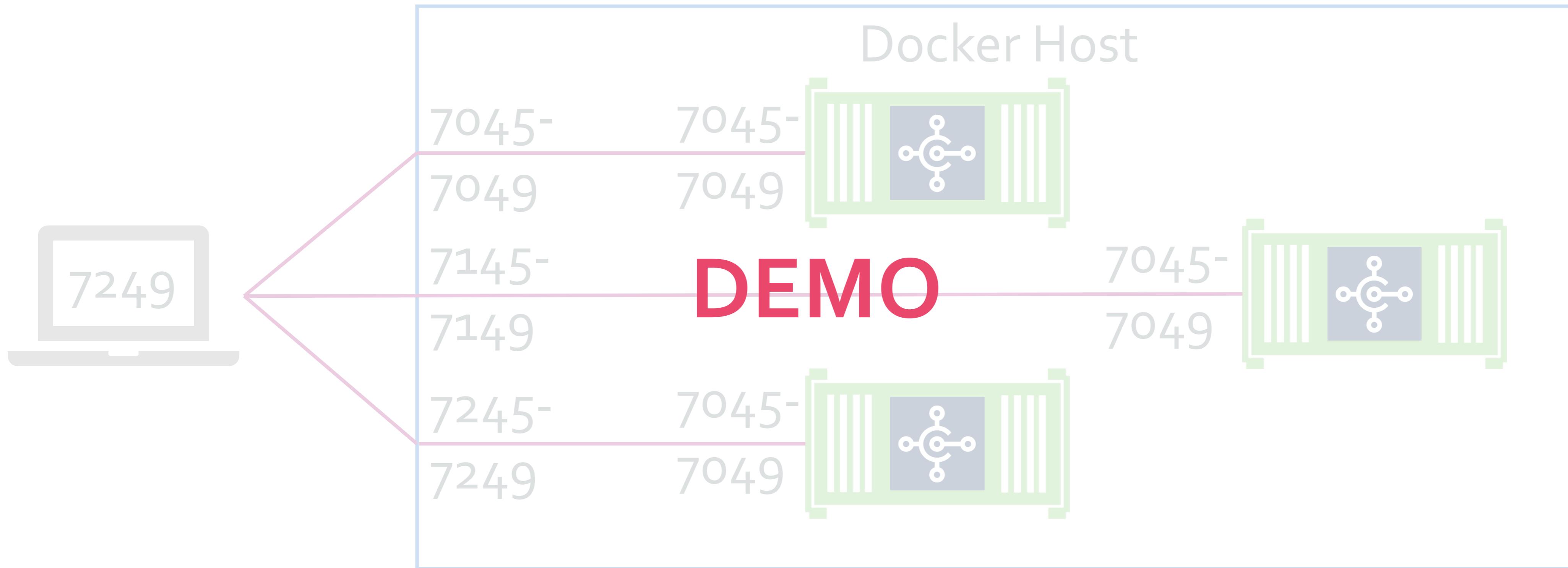
Solution 1: Port mapping

Run your containers and **map their ports** to different host ports



Solution 1: Port mapping

Run your containers and **map their ports** to different host ports



Port mapping – the good and the bad

Good:

Easy to connect to from the client (if you know the right port)

Bad:

Always need to determine which ports are free for the next container

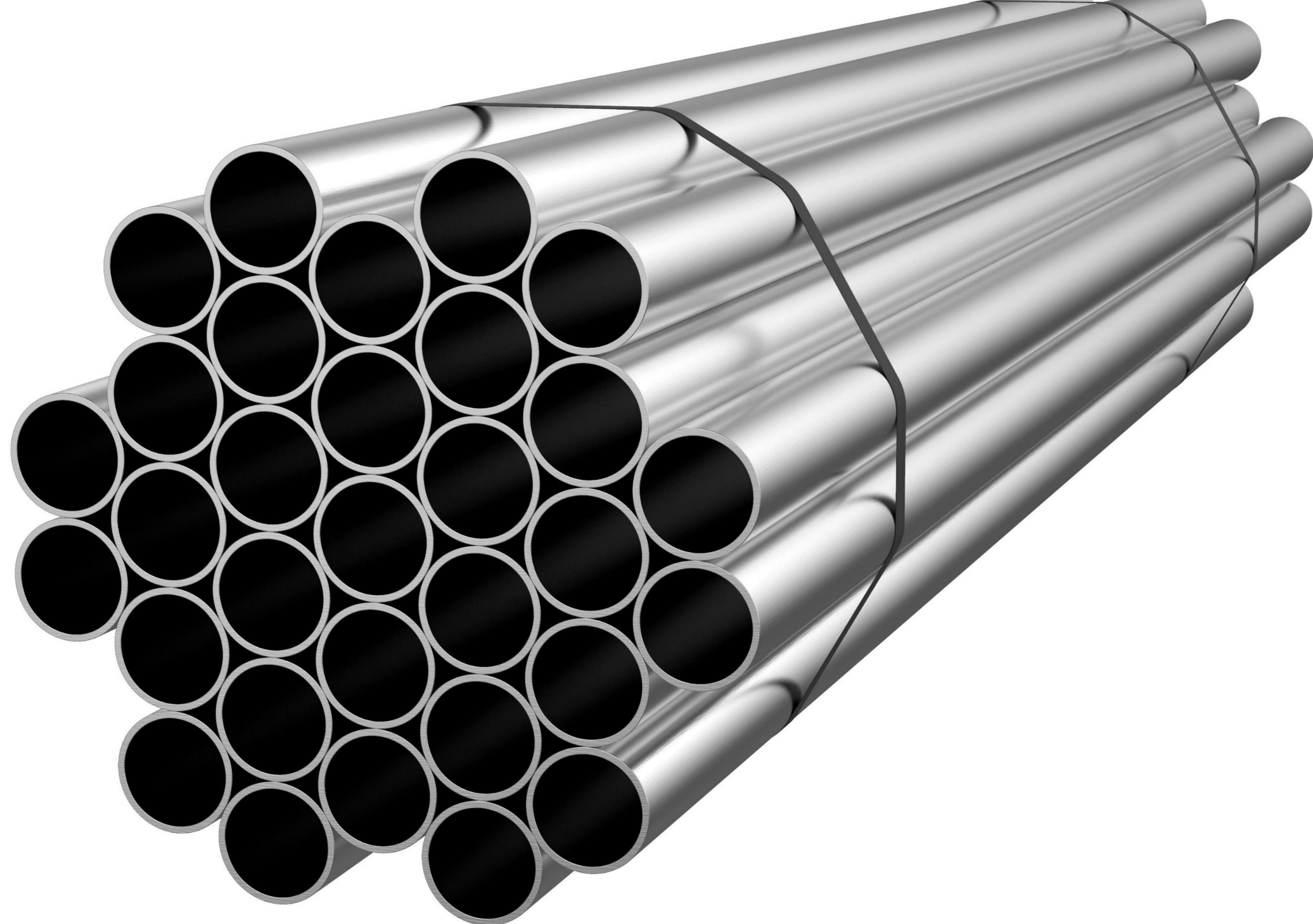
Don't forget 80, 443, 1443, 8080

Need to open ports on the firewall of the VM

On Azure that becomes two firewalls (VM and Azure networking)

→ Possible but somewhat complicated and error prone



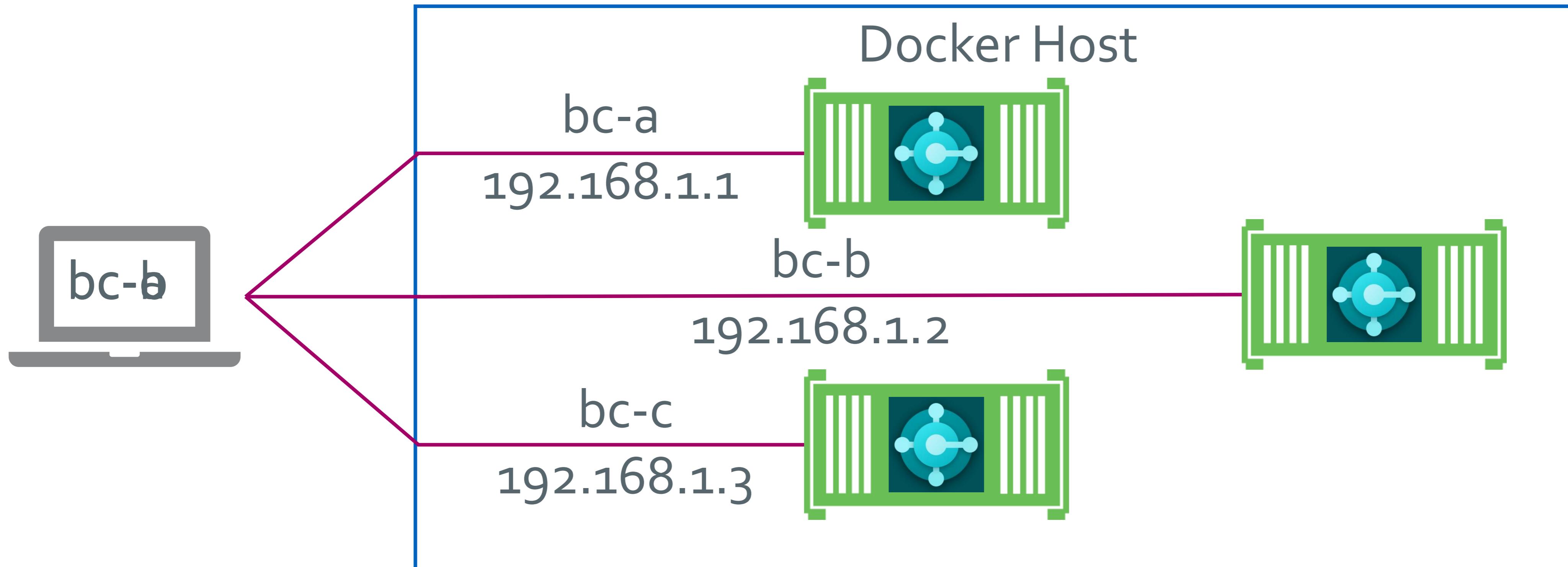




Handle every
container
individually

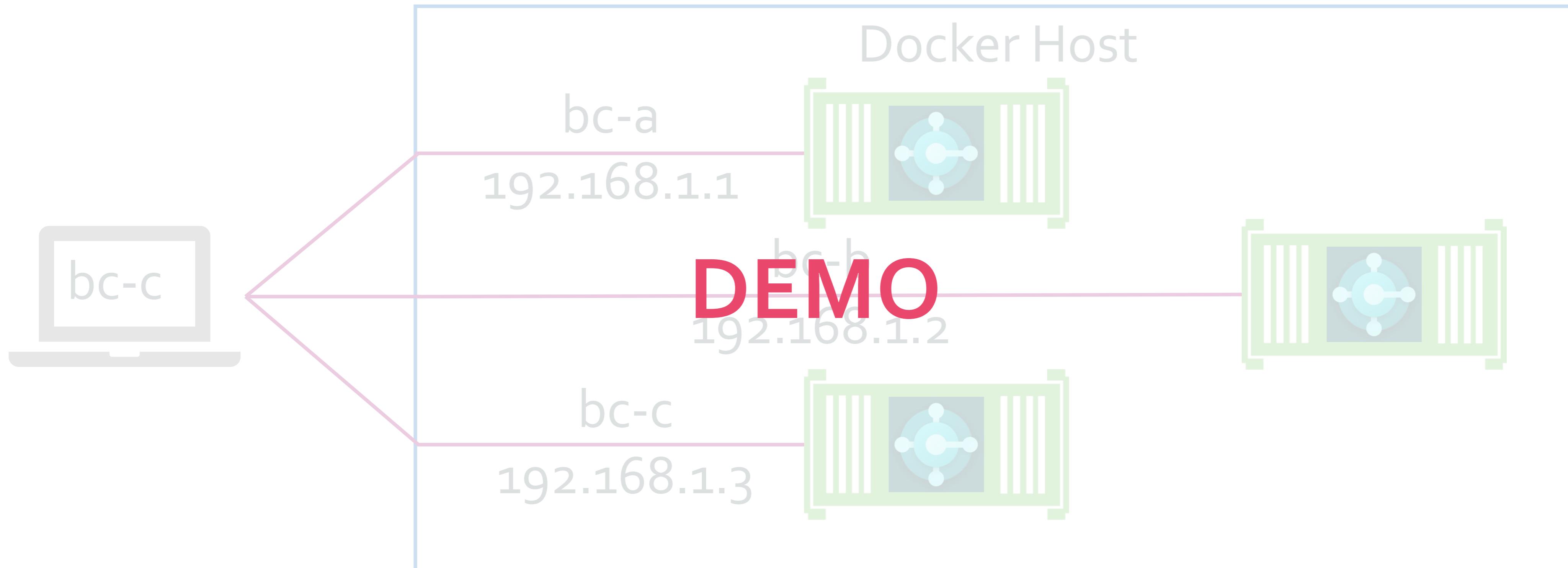
Solution 2: Transparent networking

Run every container with its own IP (and name)



Solution 2: Transparent networking

Run every container with its own IP (and name)



Transparent netw. – the good and the bad

Good:

Easy to connect to from the client (you only need the name)

Creating a new container is easy as well

Bad:

Needs to be allowed on your network

Needs a specific setting on your hypervisor (MAC address spoofing)

Not directly possible on Azure VMs

→ Good solution for on prem if allowed but not for Azure



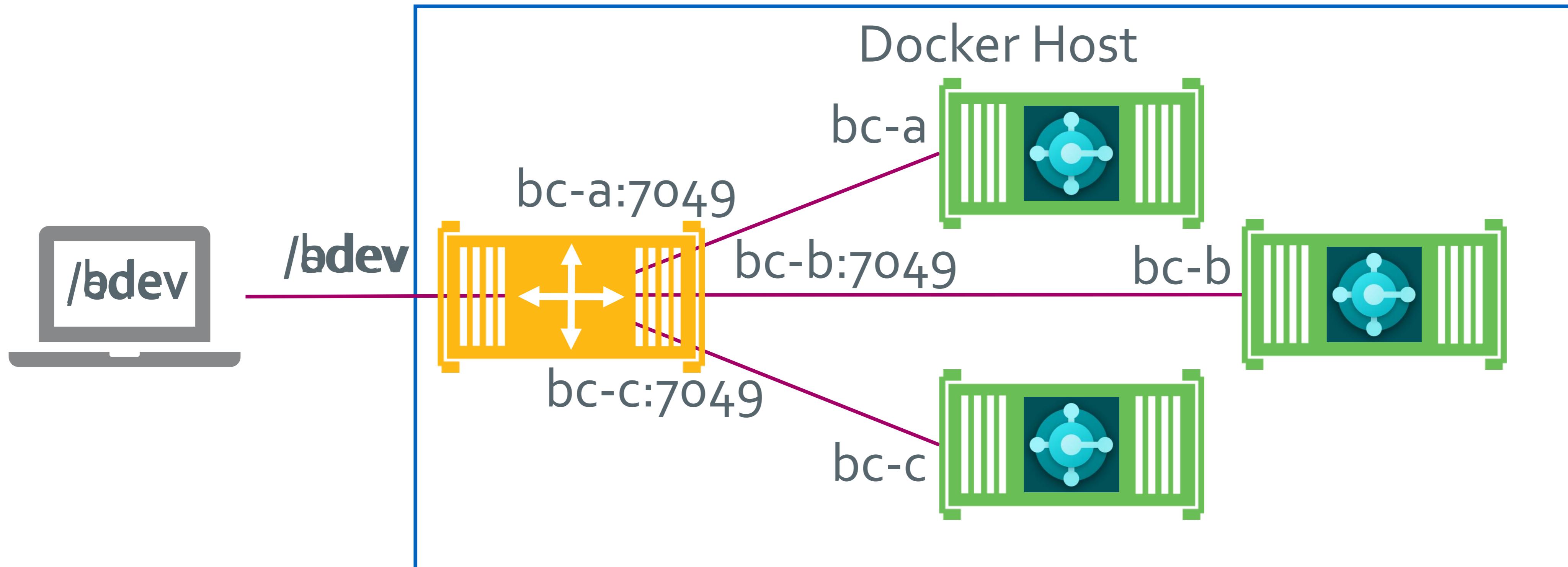




All networking
goes through
one „pipe”

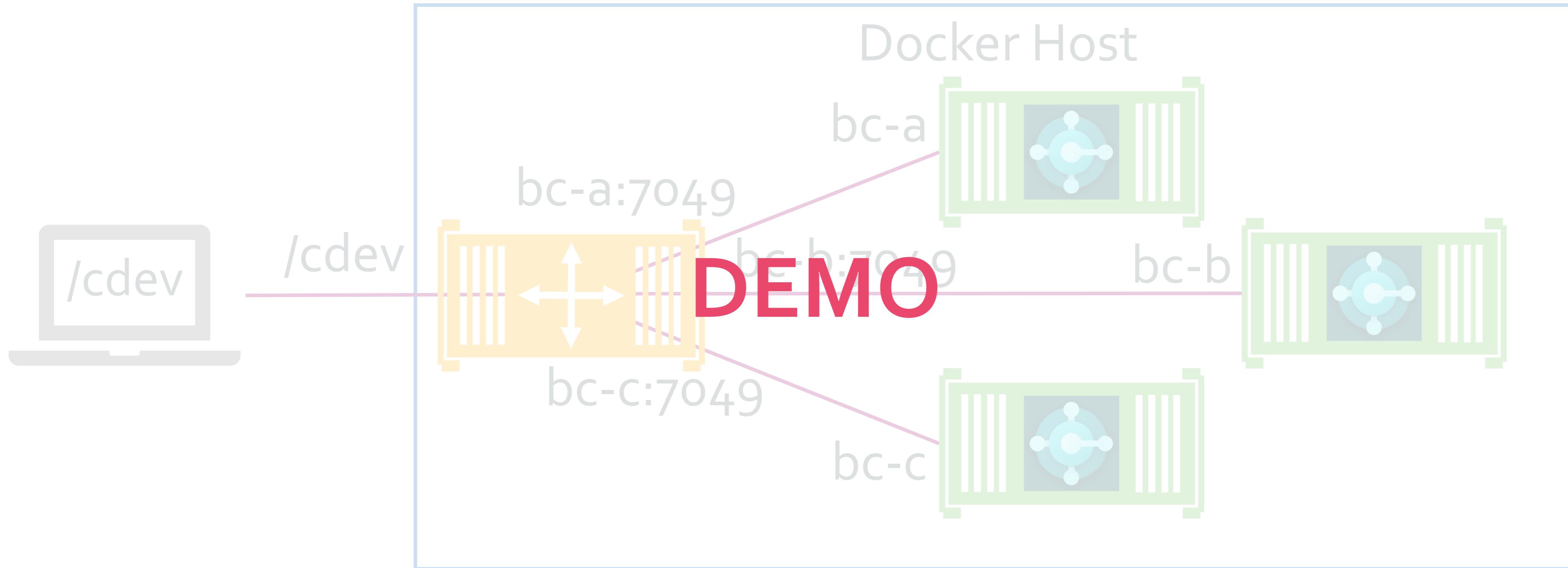
Solution 3: Reverse proxy

Run your containers **behind a reverse proxy**



Solution 3: Reverse proxy

Run your containers **behind a reverse proxy**



Reverse proxy – some details

Implemented using **traefik** (<https://traefik.io/>)



Cloud-native, container-native reverse proxy

Easy to set up and run, e.g. integrated LetsEncrypt support

Picks new containers up by checking their labels

Regex-based rules for the mapping, e.g.

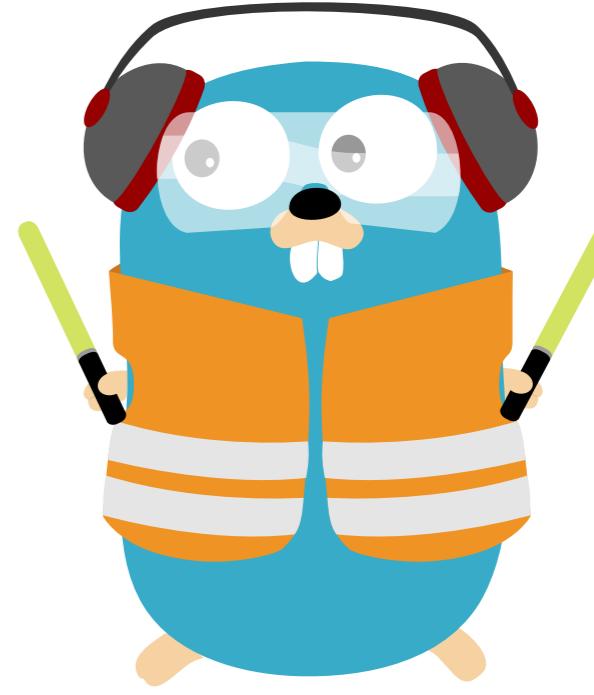
`https://myvm.westeurope.cloudapp.azure.com/bc-arest/*`

maps to `http://bc-a:7048/BC/OData/*`

`https://myvm.westeurope.cloudapp.azure.com/bc-a/*`

maps to `http://bc-a:80/bc-a/*`

Reverse proxy – some details



Additional config for the Business Central container:

Set PublicODataBaseUrl, PublicSOAPBaseUrl, PublicWebBaseUrl and PublicDnsName so that Business Central knows what it is **called from the outside**

Set WebServerInstance to a **different name** as it otherwise insists on redirecting to /NAV or /BC

Health check needs to be different: Traefik only picks up healthy containers but for the regular health check to work, traefik routing needs to be in place...

Traefik needs a **setup file** called `traefik.toml`

Reverse proxy – some details

Integrated into

aka.ms/getbc and related Azure ARM templates with a „Add Traefik“ toggle



navcontainerhelper with -useTraefik

Base setup needed

New containerhelper cmdlet `Setup-TraefikContainerForBCContainers`

→ Let's check the connection and look at some of the details

Reverse proxy – the good and the bad

Good:

Easy to connect to from the client (you only need the path)

Creating a new container is easy and it works on Azure

You only need one entry point per reverse proxy in the firewalls

Bad:

One more component to set up and maintain

Non-HTTP-traffic needs more work (RTC and SQL/finsql) → traefik 2.0 supports that, but I haven't tested yet

URLs returned from SOAP and REST endpoints not correct

→ Good solution for both worlds, especially with automated setup

Limitations of our previous approaches

Bound to **1 host**

What happens when we run out of resources?

If we use **multiple hosts**, how to figure out **what to put where**?

How to let users know **what is running where** and **how to reach it**?

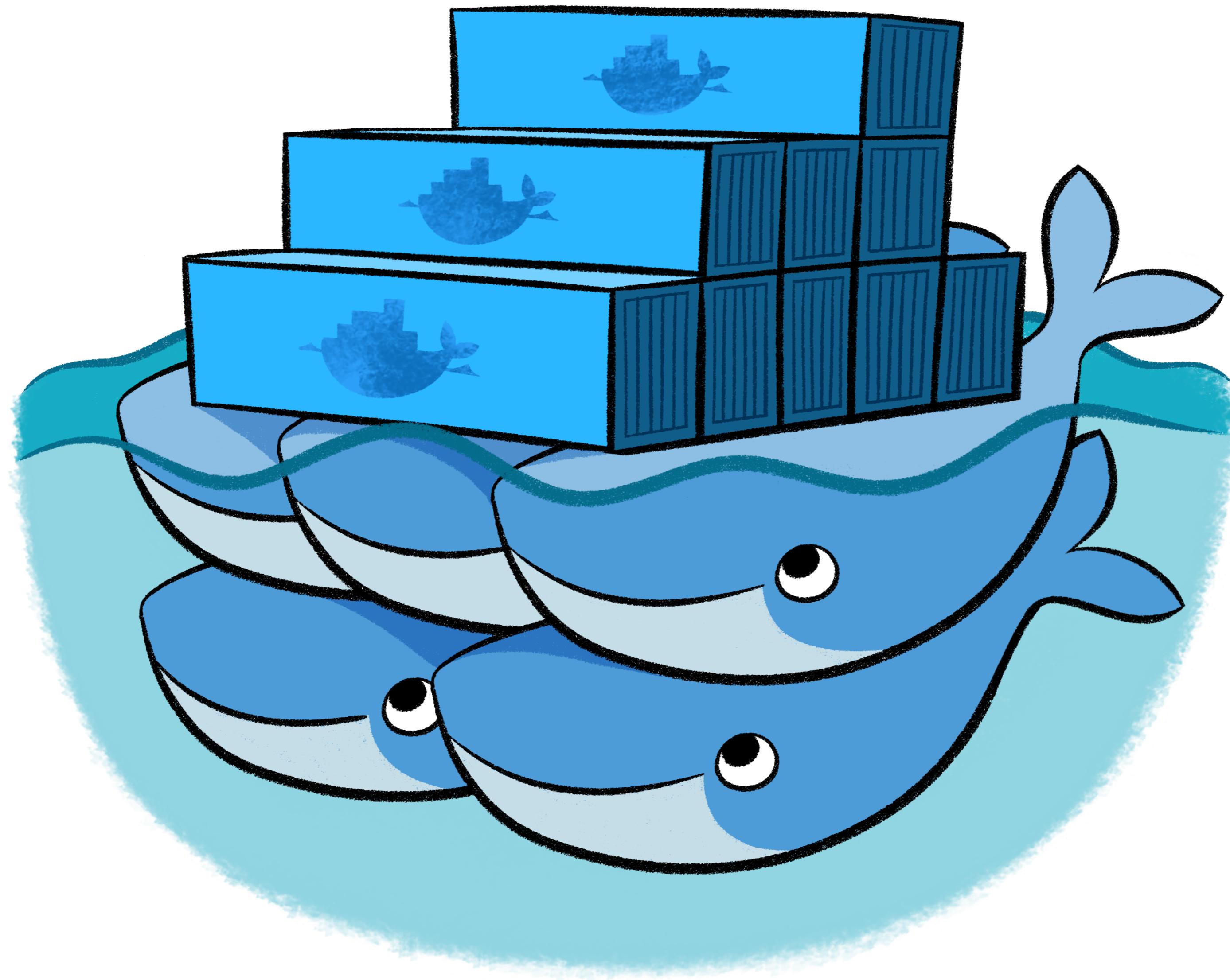
Scaling up and down still is **problematic**

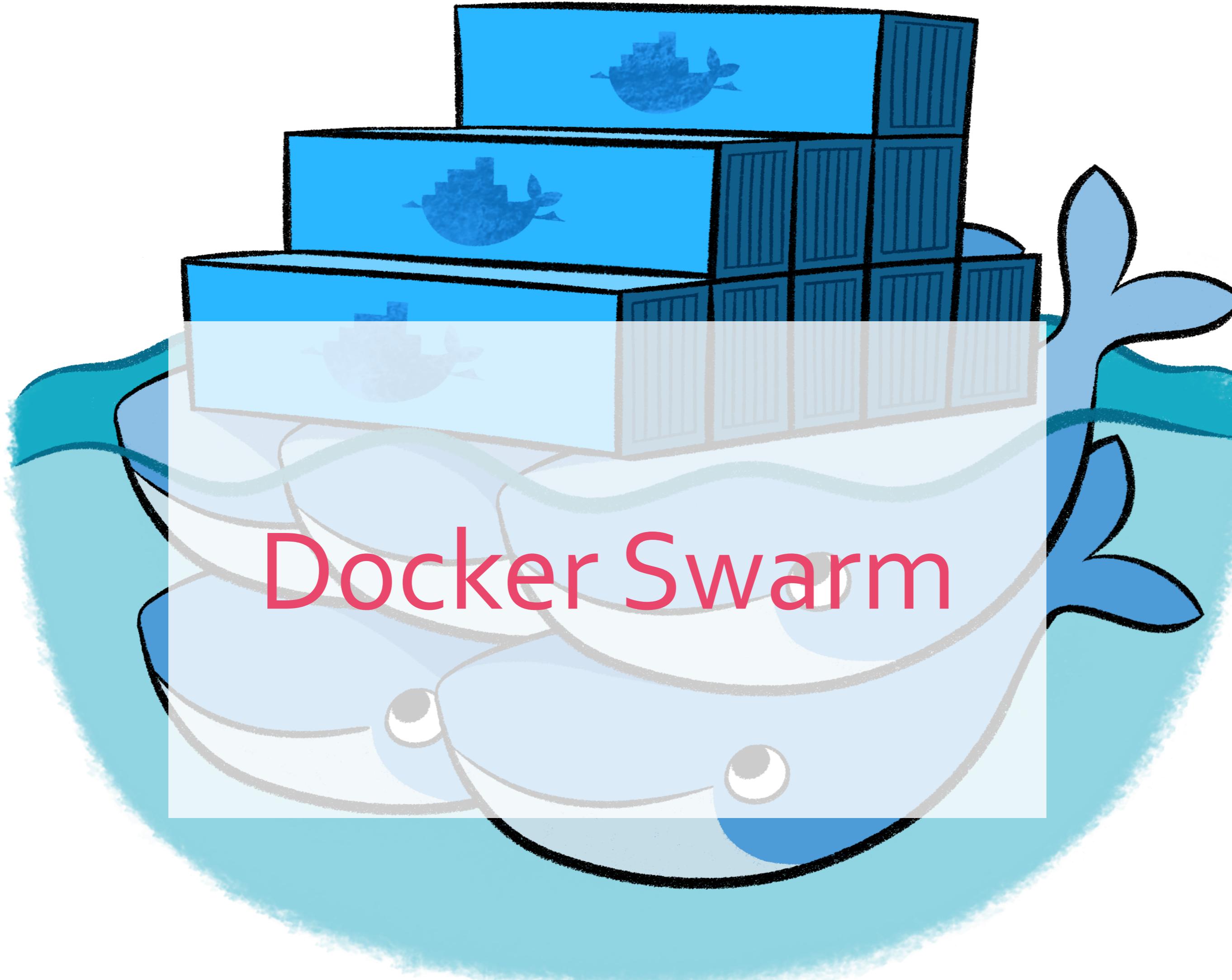
Still running **1 SQL Server per container**

Not very efficient

Scaling limits as above

And it's SQL Server **Express** (max **10GB** database size)





Docker Swarm

Part 1 of the solution: Docker Swarm

Built-in container orchestrator from Docker

Main benefits / features:

Brings resources of **multiple container hosts** together

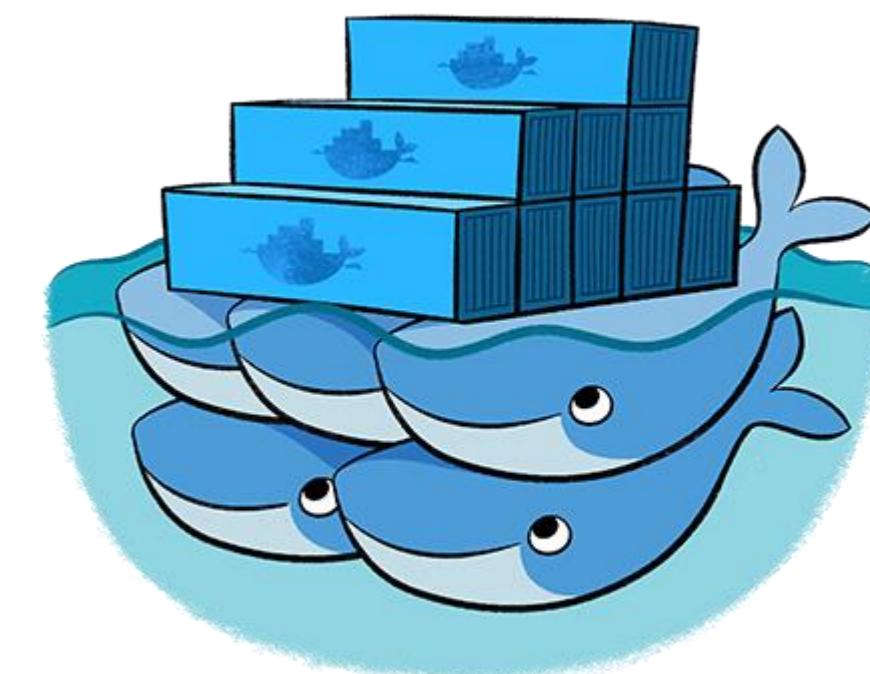
Central management and control

Share **configuration and secrets** across the swarm

Declarative service model

Automatic “**self-healing**” concepts

Advanced networking for **resiliency**



What about **Kubernetes** and the recent **Mirantis** deal?

Part 1 of the solution: Docker Swarm

Some **basics** in the Docker Swarm world:

Service = declaration of the images, number of containers (**tasks**) and configurations you want to run; can be replicated or global

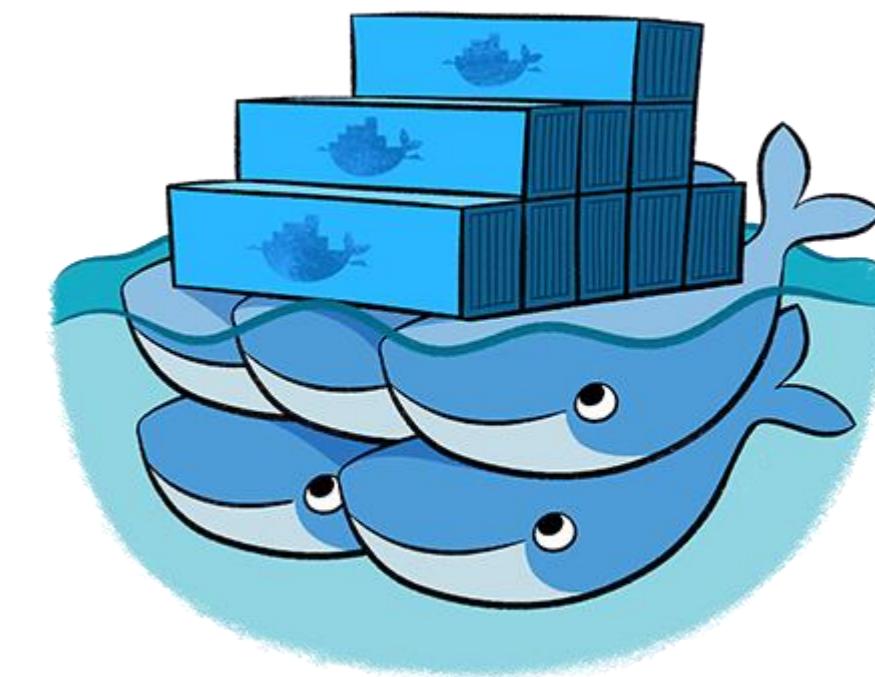
Node = container host / engine that joined the swarm

Manager = node with control rights

Worker = node that executes tasks

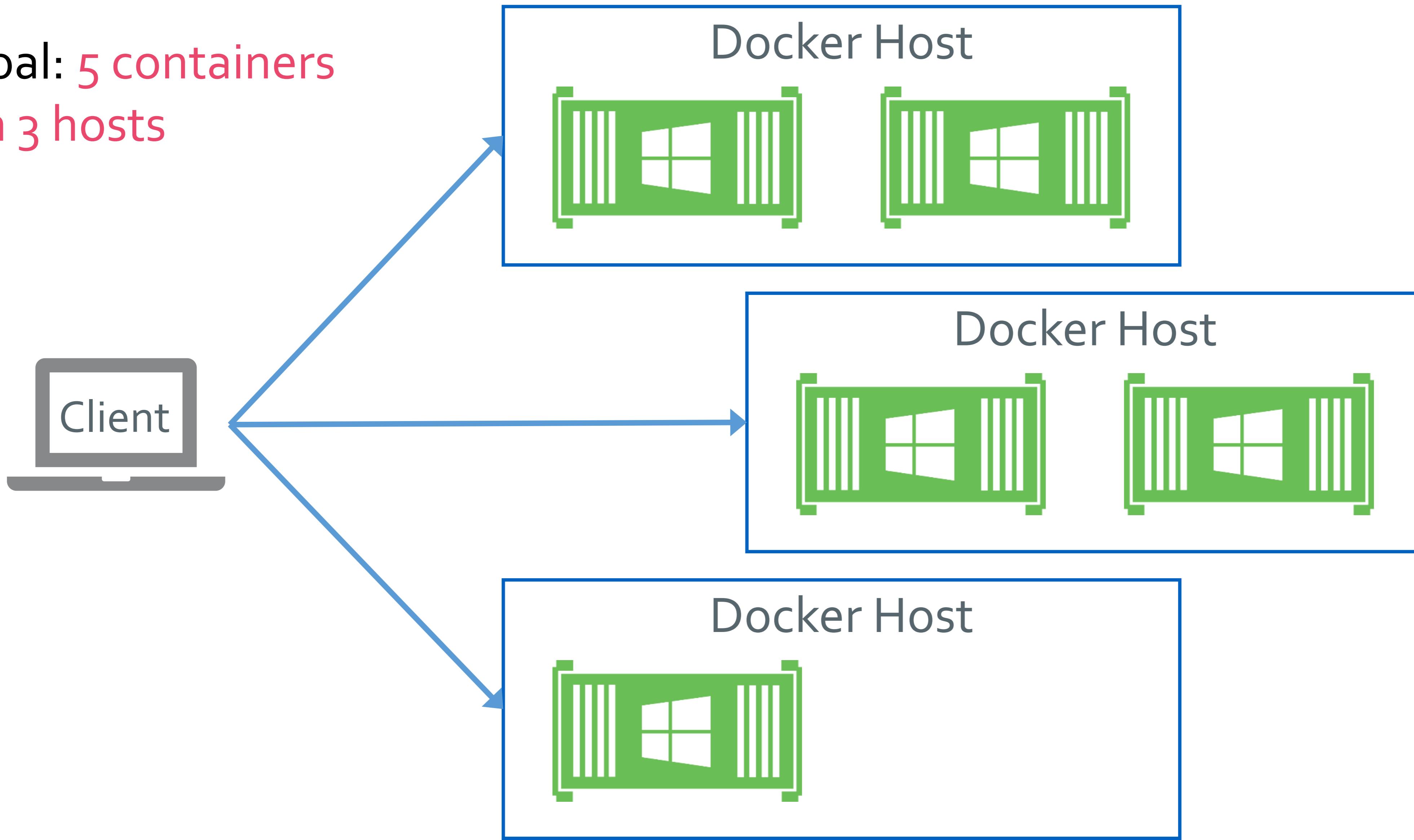
How to “run” something:

1. **Declare** your service
2. **Submit** that to a manager node
3. **Swarm** creates necessary **tasks** on **nodes** and keeps desired state



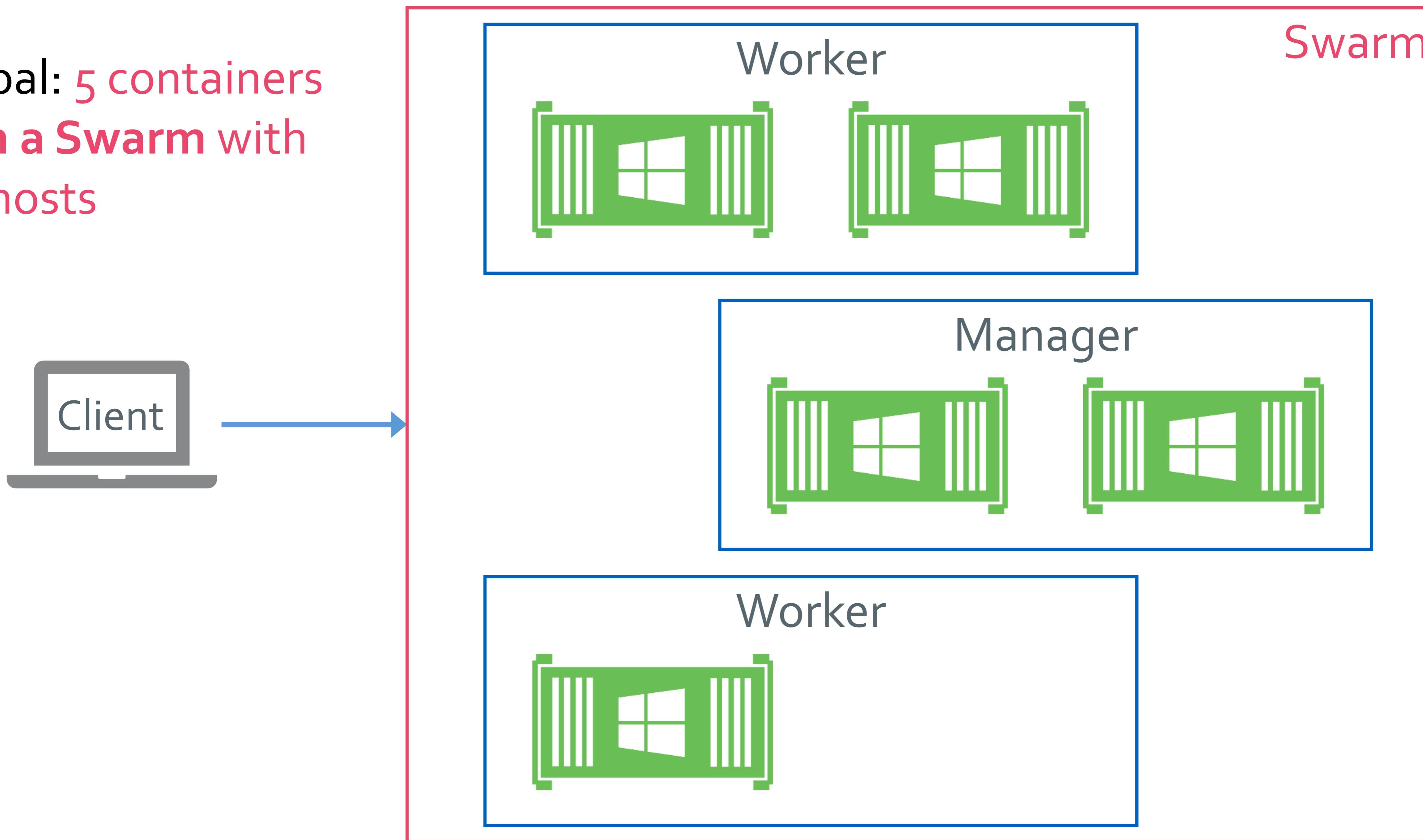
Part 1 of the solution: Docker Swarm

Goal: 5 containers
on 3 hosts



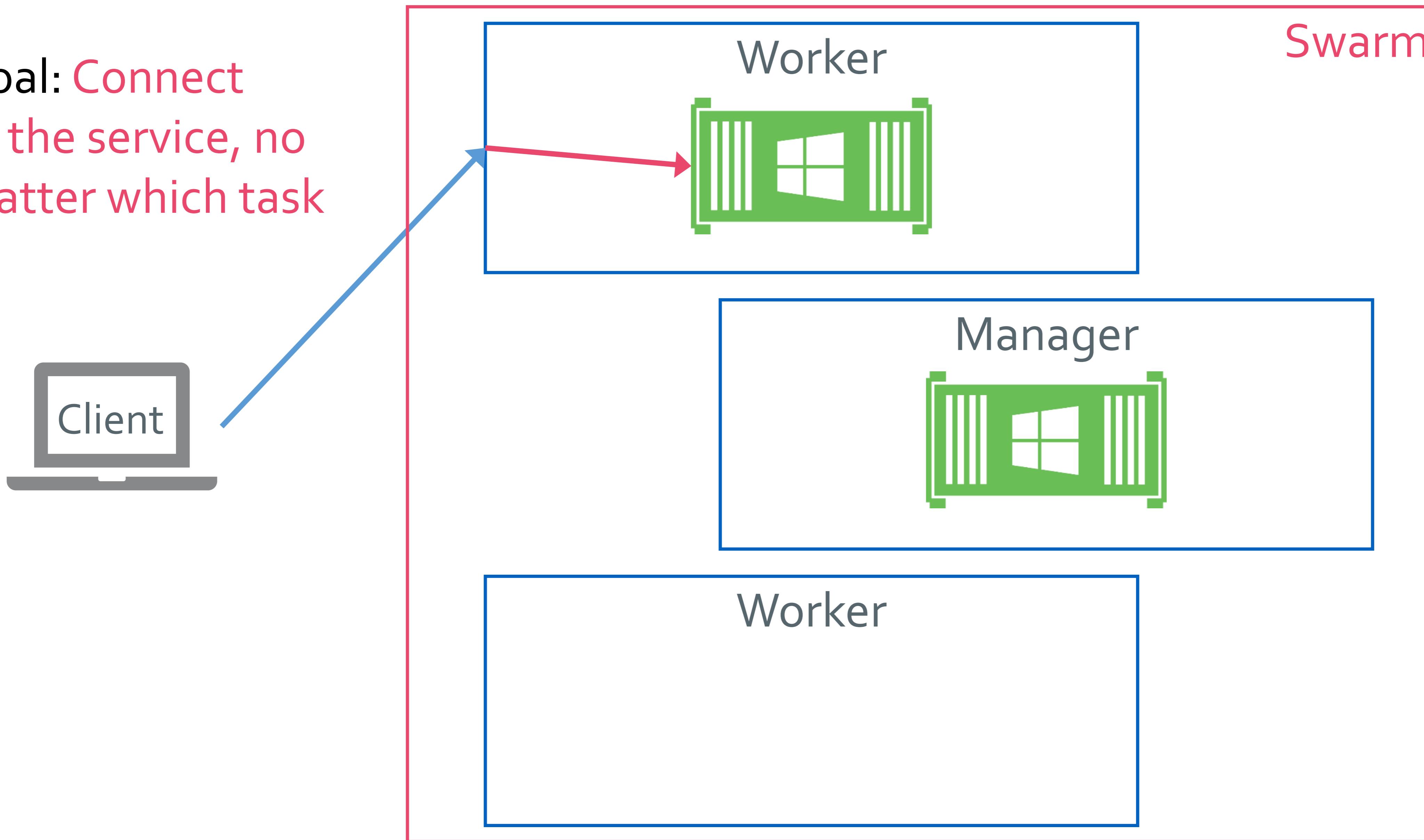
Part 1 of the solution: Docker Swarm

Goal: 5 containers
on a Swarm with
3 hosts



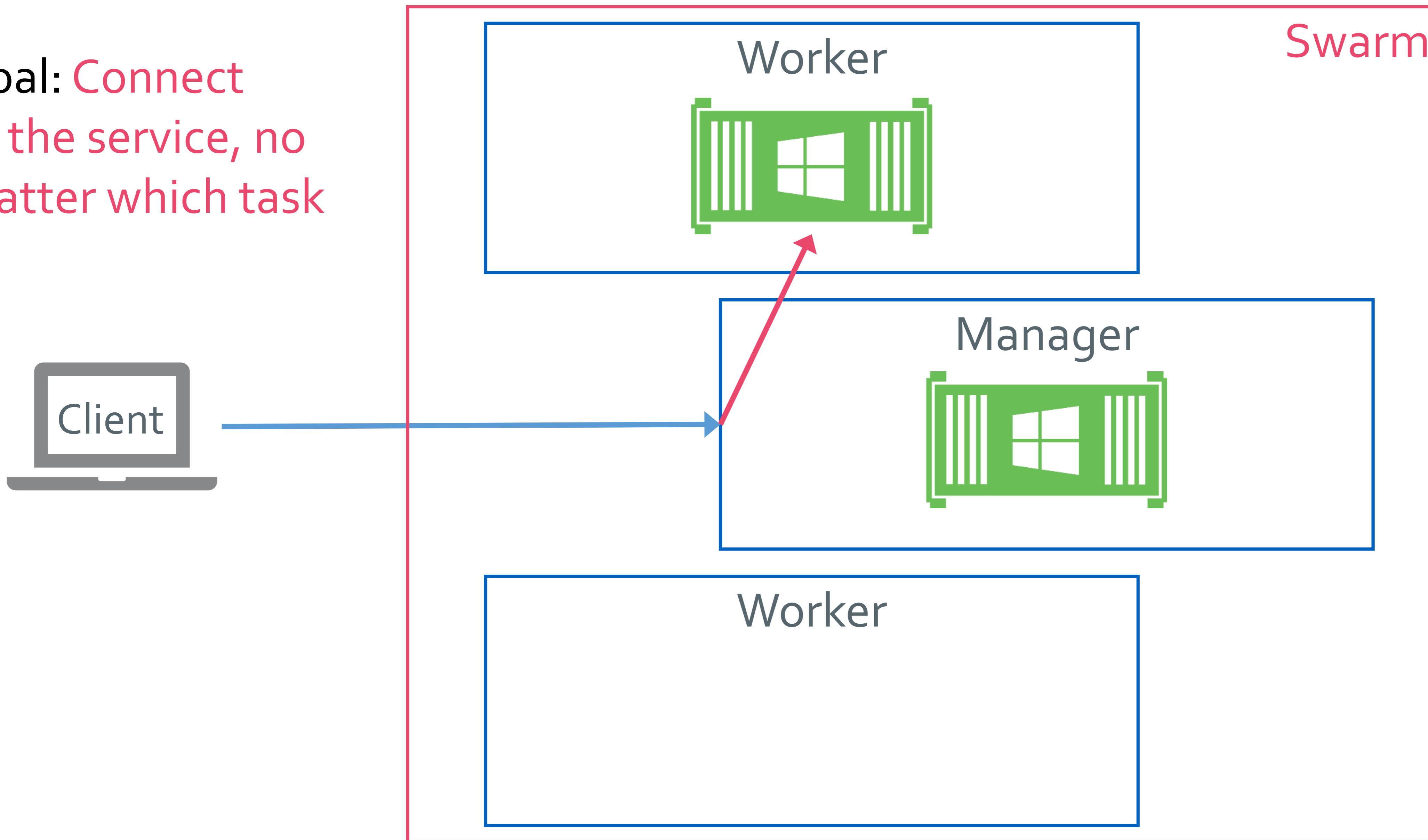
Part 1 of the solution: Docker Swarm

Goal: Connect
to the service, no
matter which task



Part 1 of the solution: Docker Swarm

Goal: Connect
to the service, no
matter which task

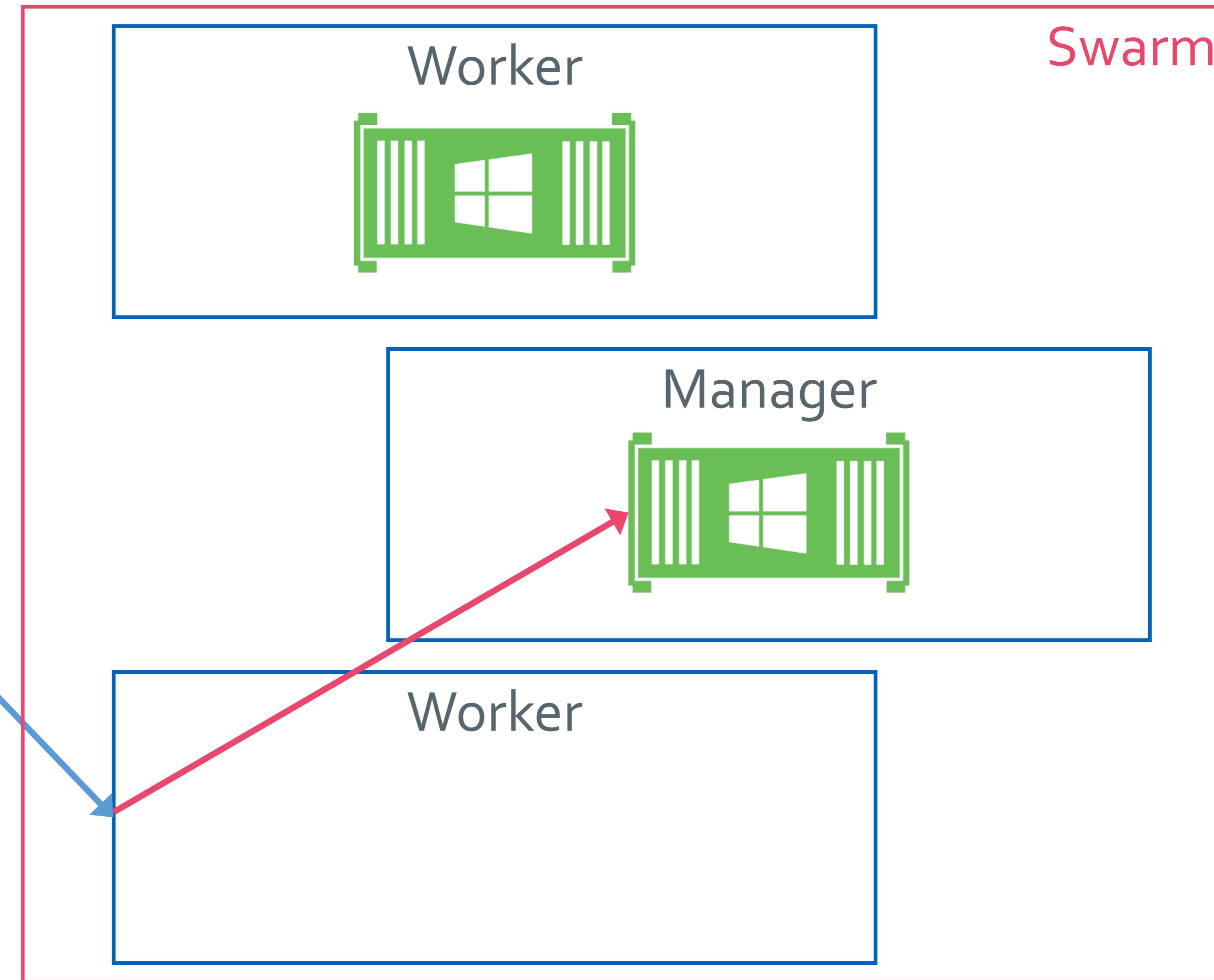


Part 1 of the solution: Docker Swarm

Goal: Connect
to the service, no
matter which task

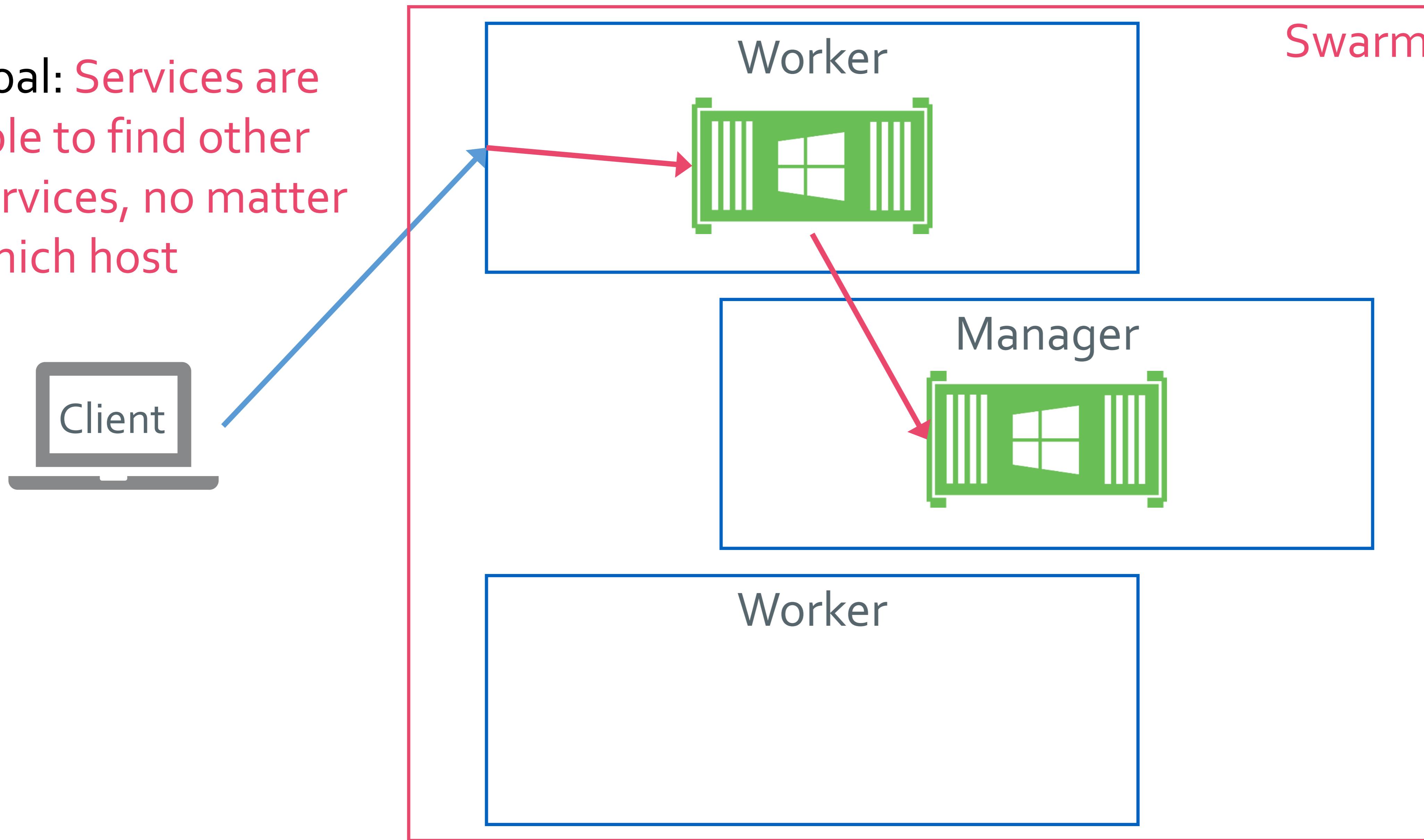


Note: Doesn't
directly work with
BC as it requires a
stateless service



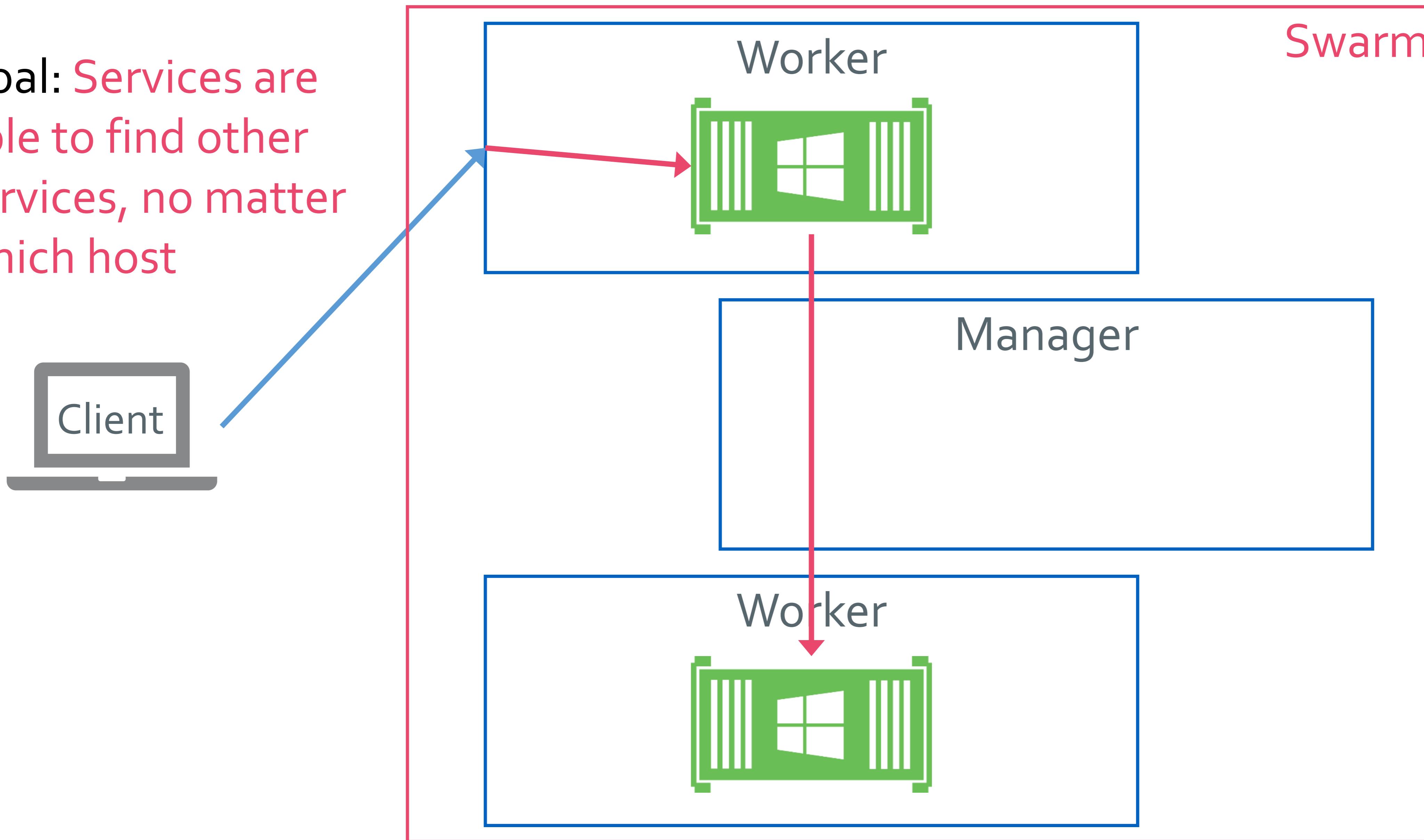
Part 1 of the solution: Docker Swarm

Goal: Services are able to find other services, no matter which host



Part 1 of the solution: Docker Swarm

Goal: Services are able to find other services, no matter which host



Part 1 of the solution: Docker Swarm

Demo scenario:

Create 1 service consisting of 2 containers

→ Show connections

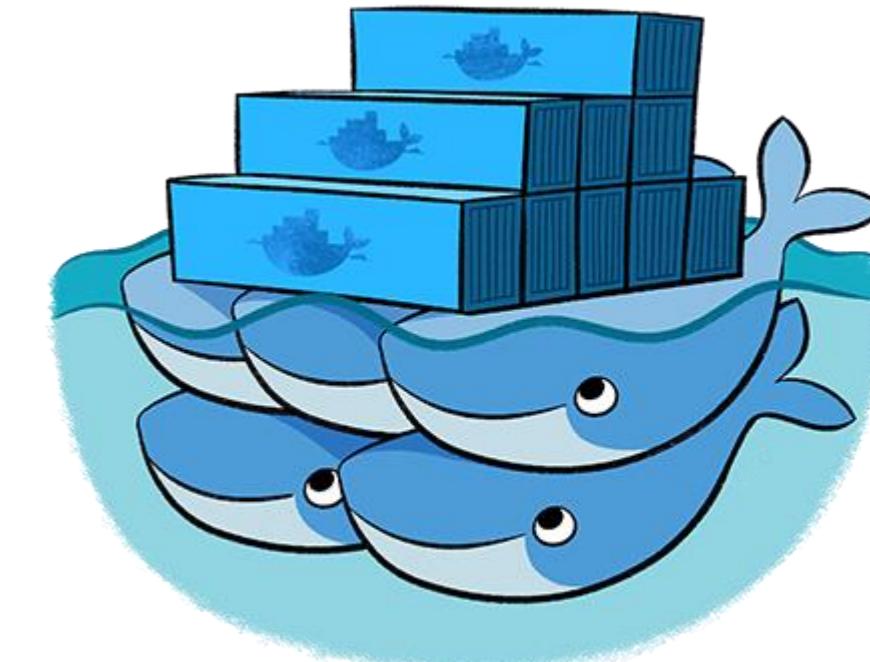
Scale service up

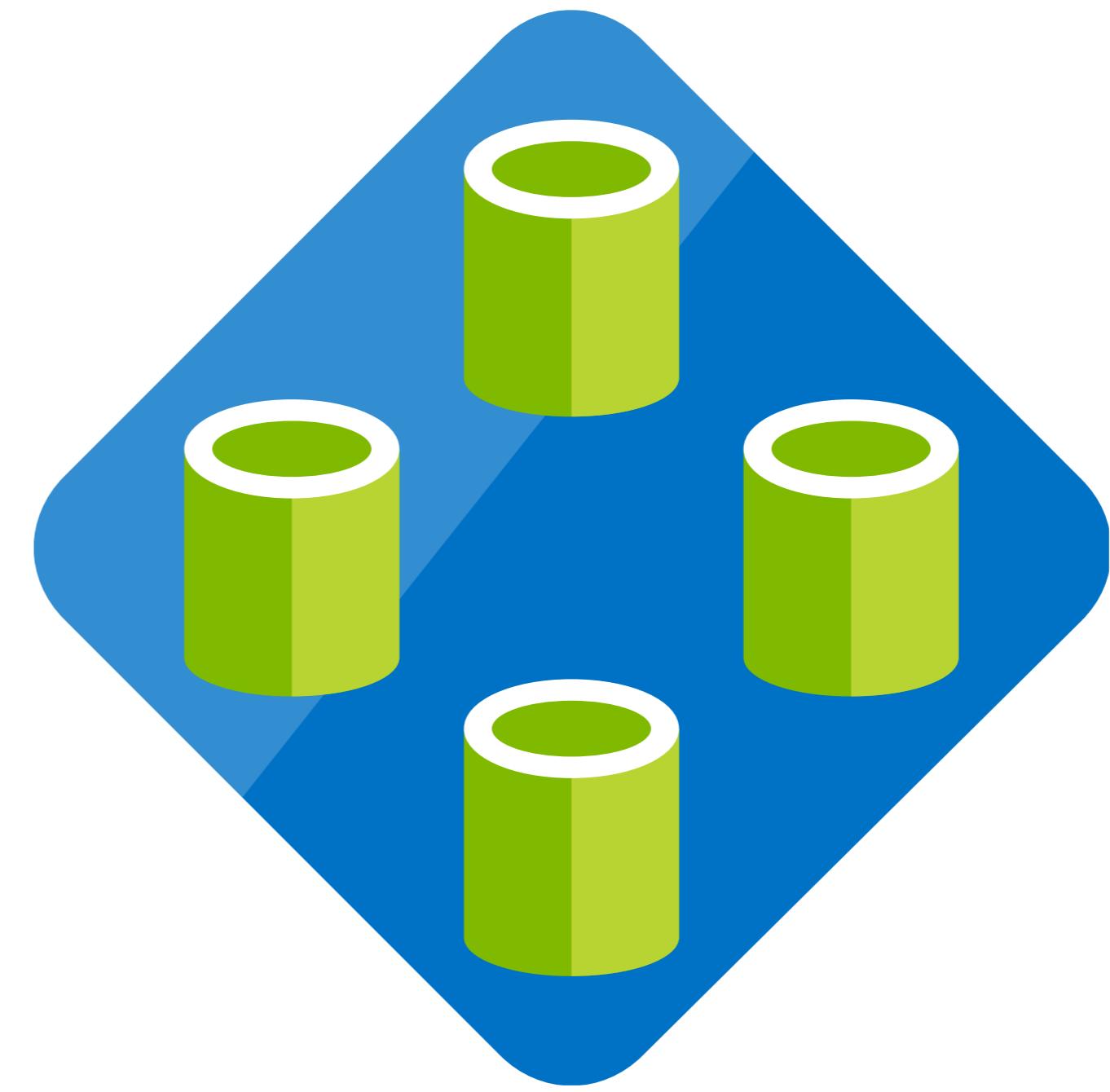
→ See placement on nodes

Remove task

→ See recovery

→ Let's see it!







Part 2 of the solution: Azure SQL

Platform as a Service (PaaS) offering

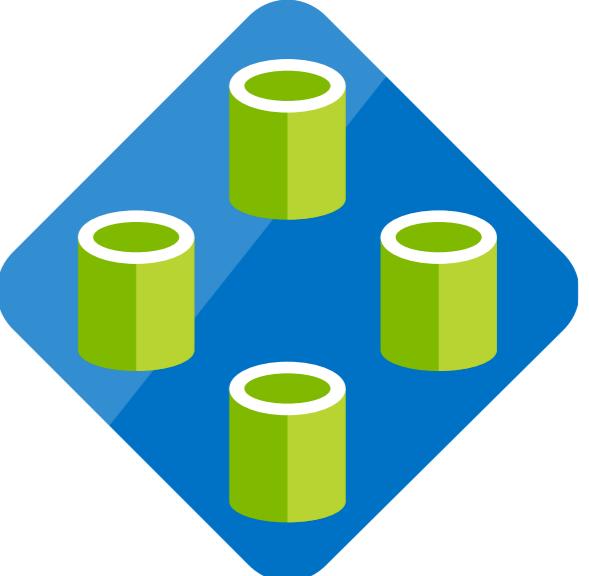
Always kept current by Microsoft

Scale up and down dynamically

(Almost) no resource limits

Elastic pools allow resource sharing across all databases (with configurable limits)

→ No server maintenance, no limits; just cost...



Limitations of our previous approaches

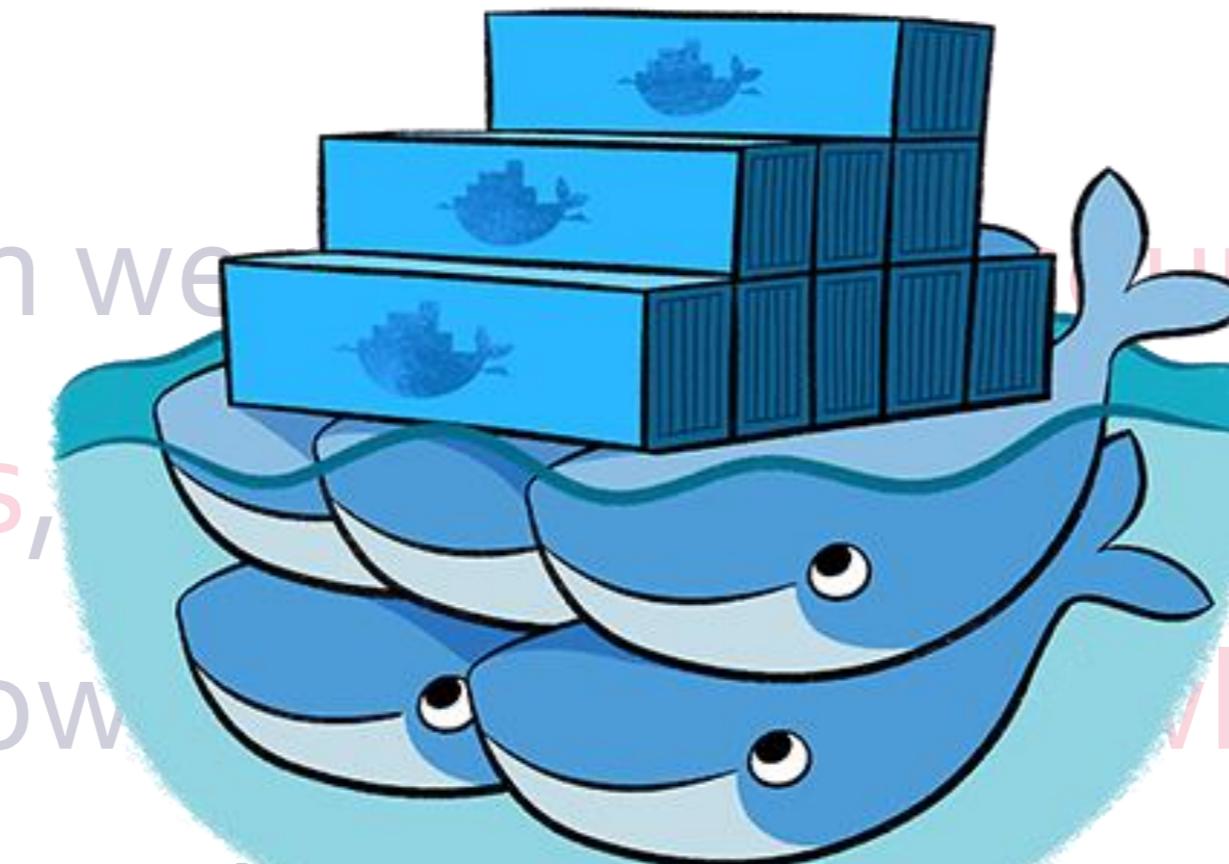
Bound to **1 host**

What happens when we have **multiple sources?**

If we use **multiple hosts**, what to put where?

How to let users know where and **how to reach it?**

Scaling up and down still is **programmatic**

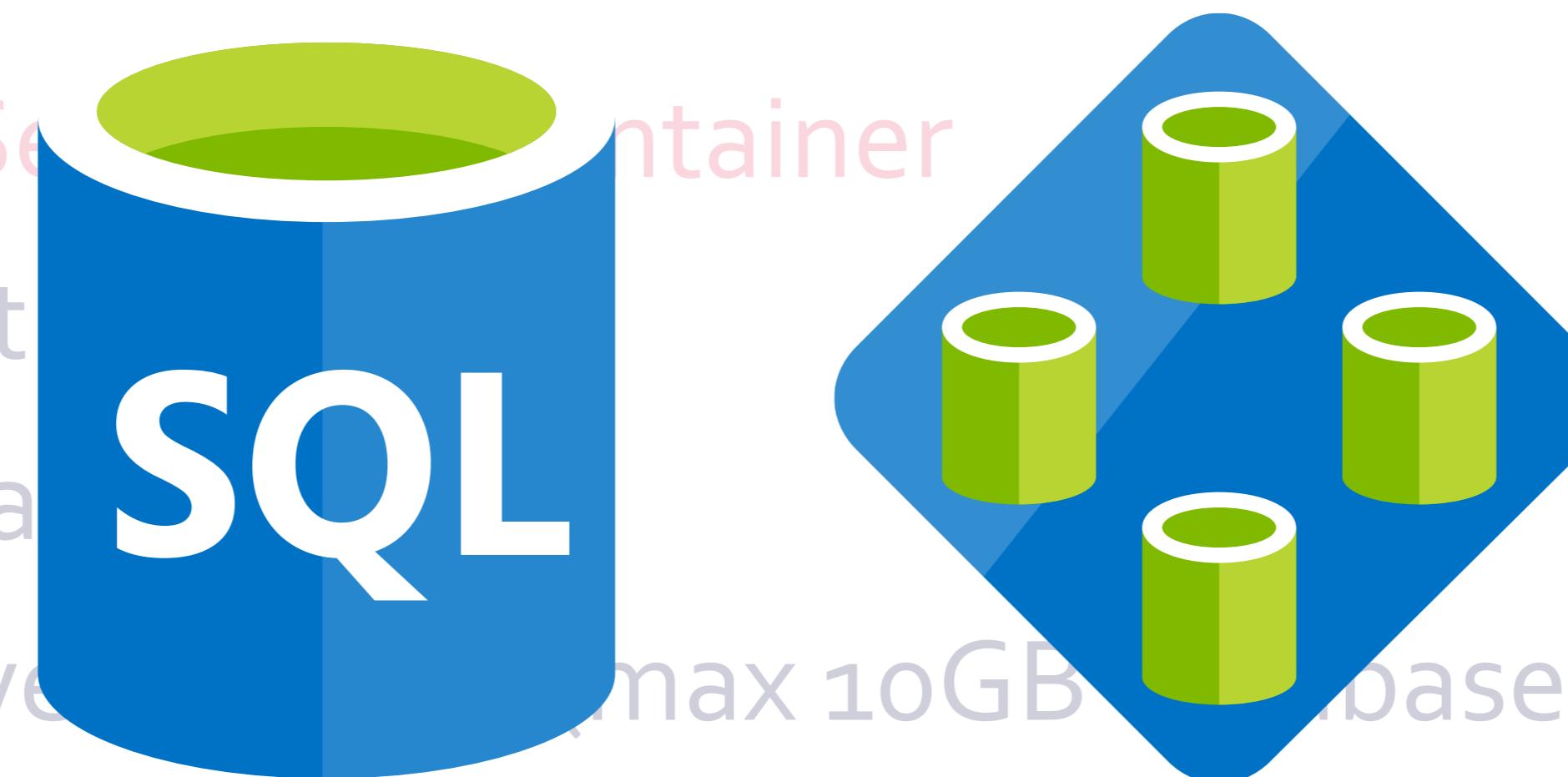


Still running **1 SQL Server** in a container

Not very efficient

Scaling limits as a

And it's SQL Server (max 10GB database size)

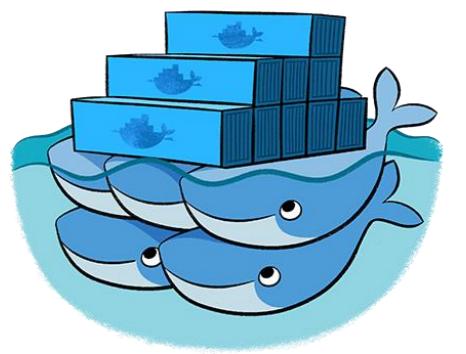


Bringing it together: Swarm & Azure SQL

Prereq: Azure SQL database with a “template” database

1 ARM template to deploy

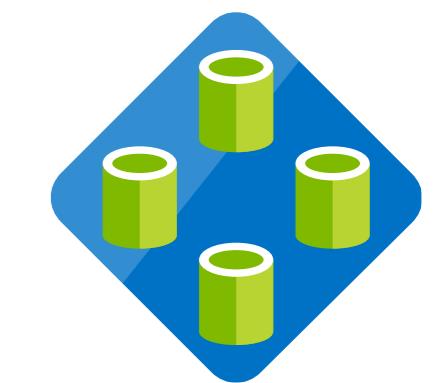
1 manager VMs and x worker VMs (note: SPOF manager!)



1 Azure SQL server with 1 elastic pool



Setup script to initialize the Swarm on the manager and join it on the workers, set up Traefik, prepare access credentials for template DB and target pool and share them as secrets



Specific scenario: Start a BC Swarm service and connect to a DB created on demand as copy of the template



→ Let's see it

Q&A

Any Questions?

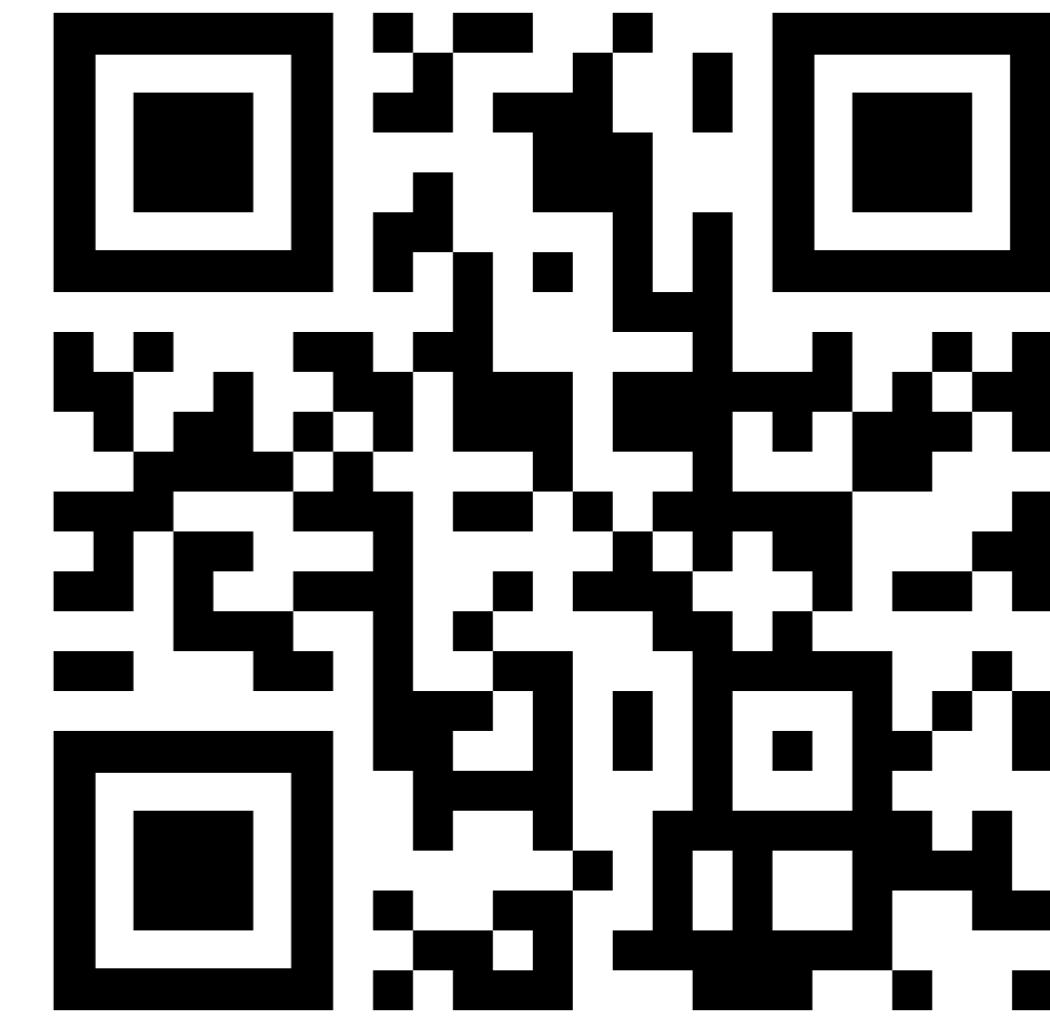
Ideas I really, really like

Get production support for Docker containers
with D365 BC: <http://bit.ly/DockerProd>



#bcalhelp

Get rid of the 120kB size limit for dev
licenses: <http://bit.ly/UnlimitedFlf>



Thank
You!