
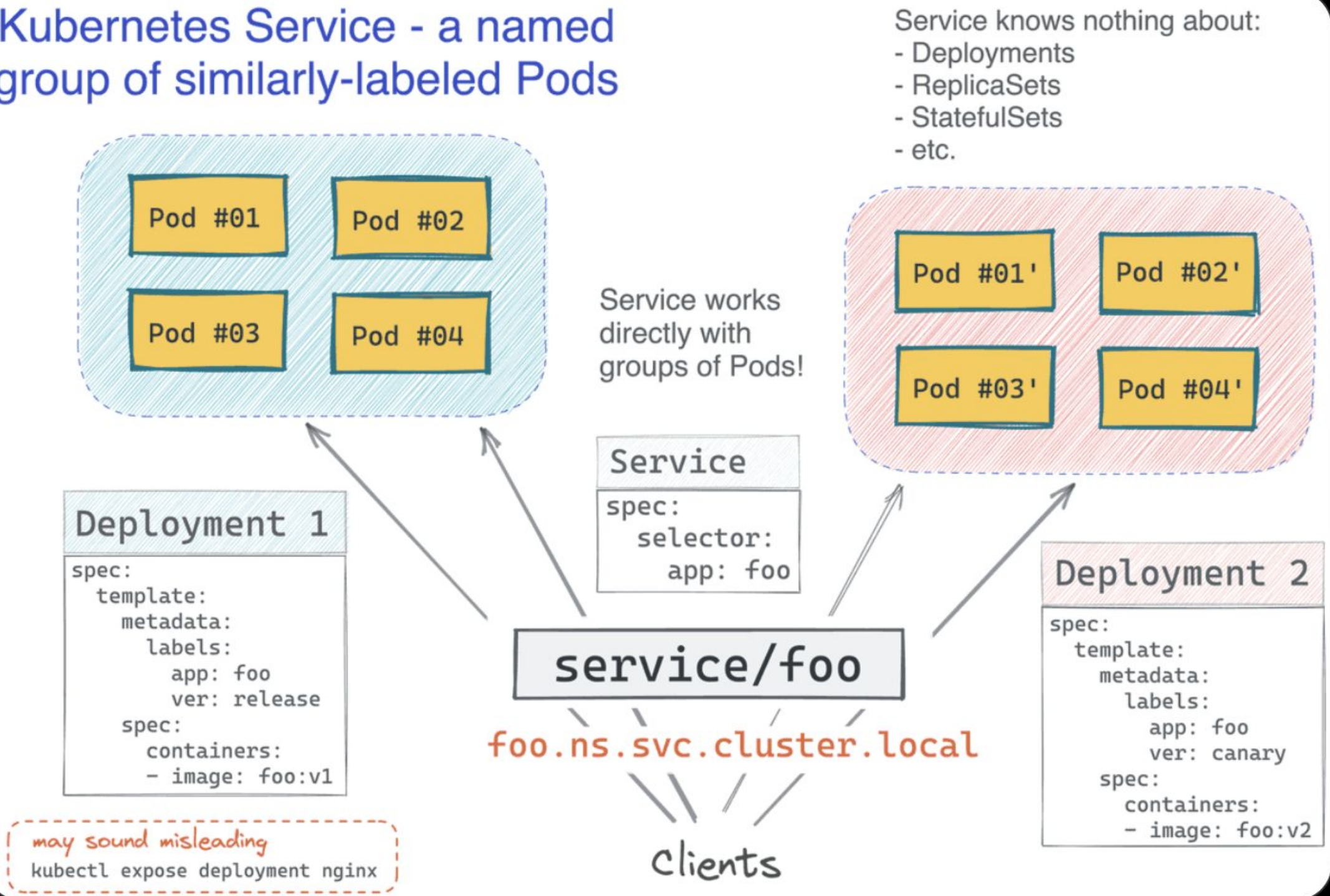


Understanding Kubernetes Through VMs: A Thread

1/  Starting with Kubernetes can be daunting. But instead of jargon-loaded explanations, let's understand Kubernetes as a natural evolution of traditional deployment techniques.

Kubernetes Service - a named group of similarly-labeled Pods

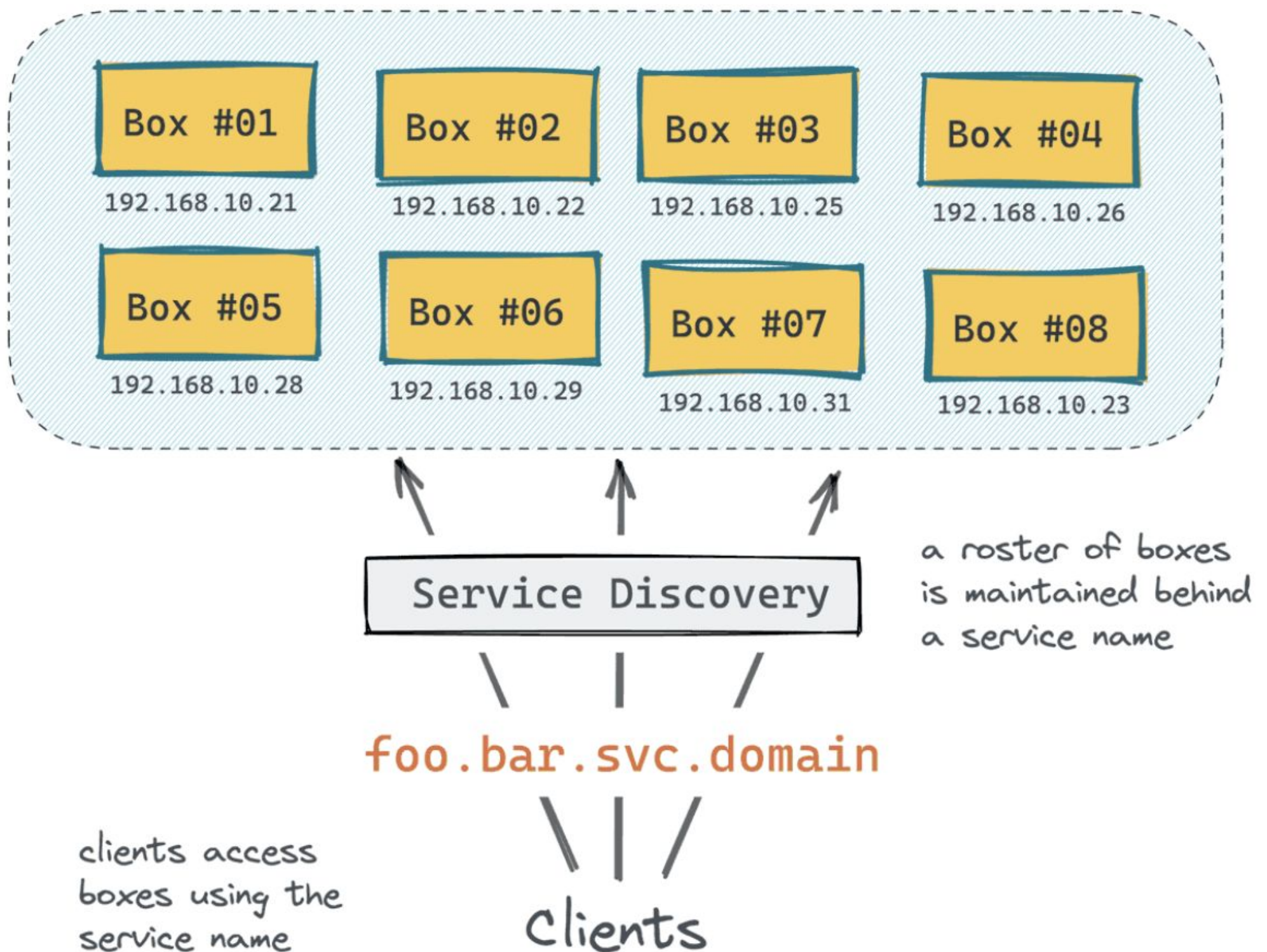


2/ 🤔 Ever deployed services using virtual machines (VMs)? If so, you'll find Kubernetes not so different.

Back in 2010, deploying with VMs (sometimes even bare-metal) was common. Think of a service as a "named group of identical machines." This was our traditional method.

Service - a named group of identical "Boxes"

distributed!



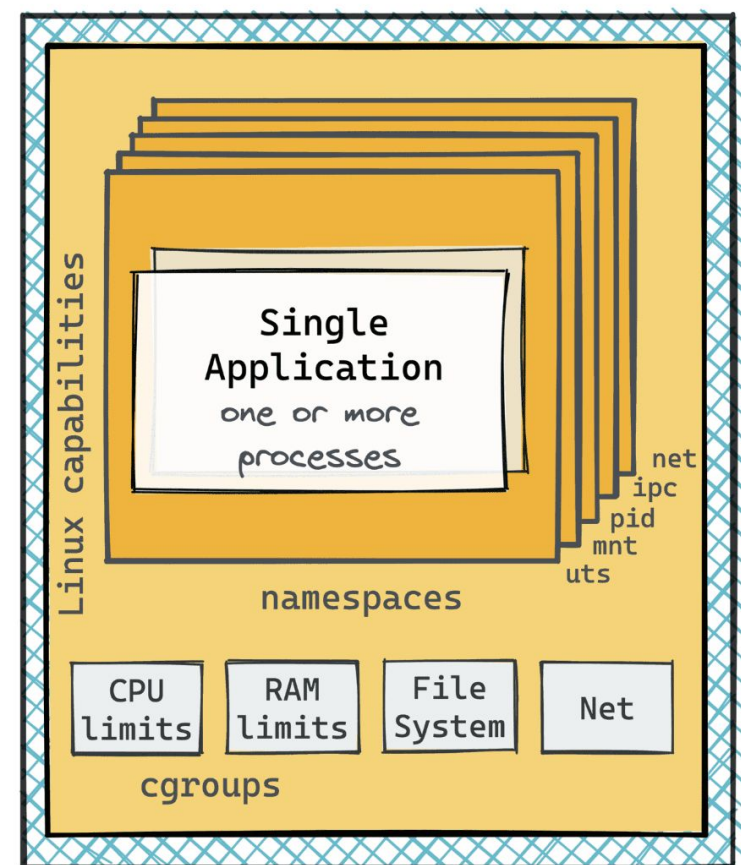
3/ 🚧 Yet, there were challenges:

- Shipping code to servers was complicated.
- Service discovery was even more complicated – often, it was a custom in-house solution.
- Efficient resource allocation was tricky.
- Scaling services required depending on machine fleet size.

4/ ⚡ Cloud tech like Amazon's EC2 and ALB/ELB brought some relief, making provisioning and automation easier. But... there was room for improvement.

Enter: Docker Containers. They aimed to solve issues like differing prod & dev environments and the bulkiness of VMs.

Container - a "box" for one app

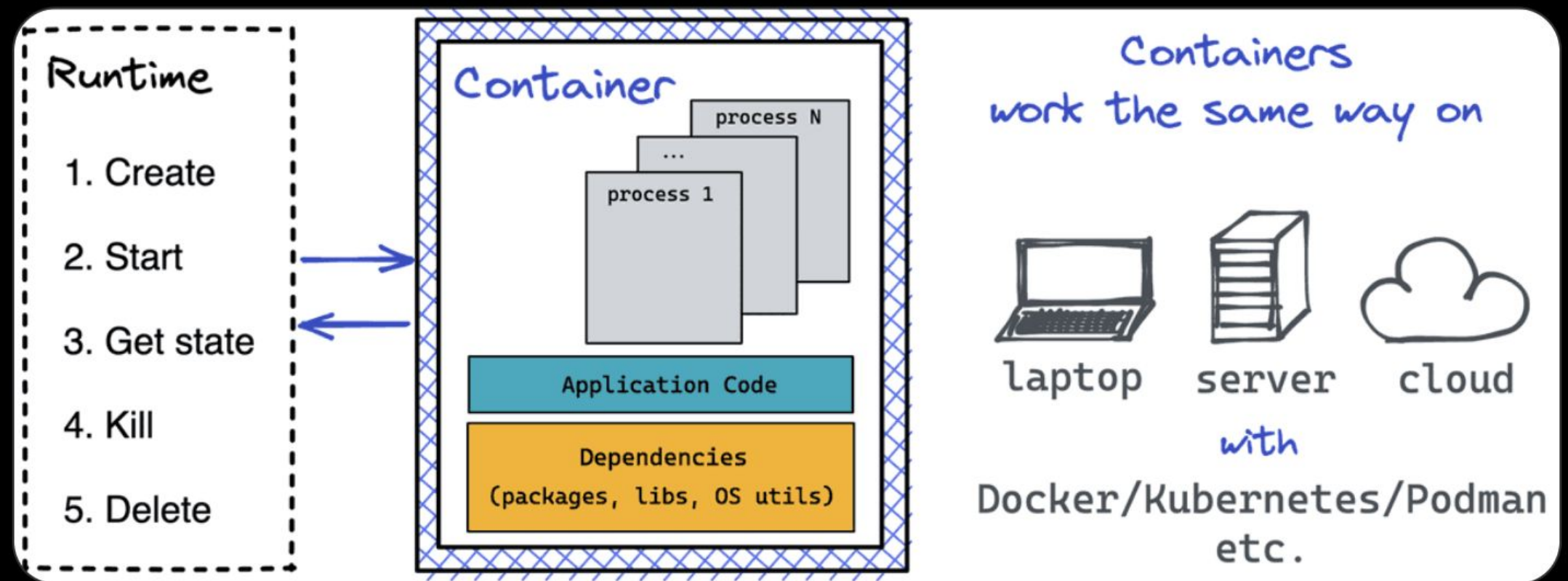


typically has a dedicated address

e.g. 172.18.0.3

5/ 💡 Containers took one Linux box and split it into isolated environments. From a dev's perspective, it seemed like having multiple VMs, but with a more lightweight touch.

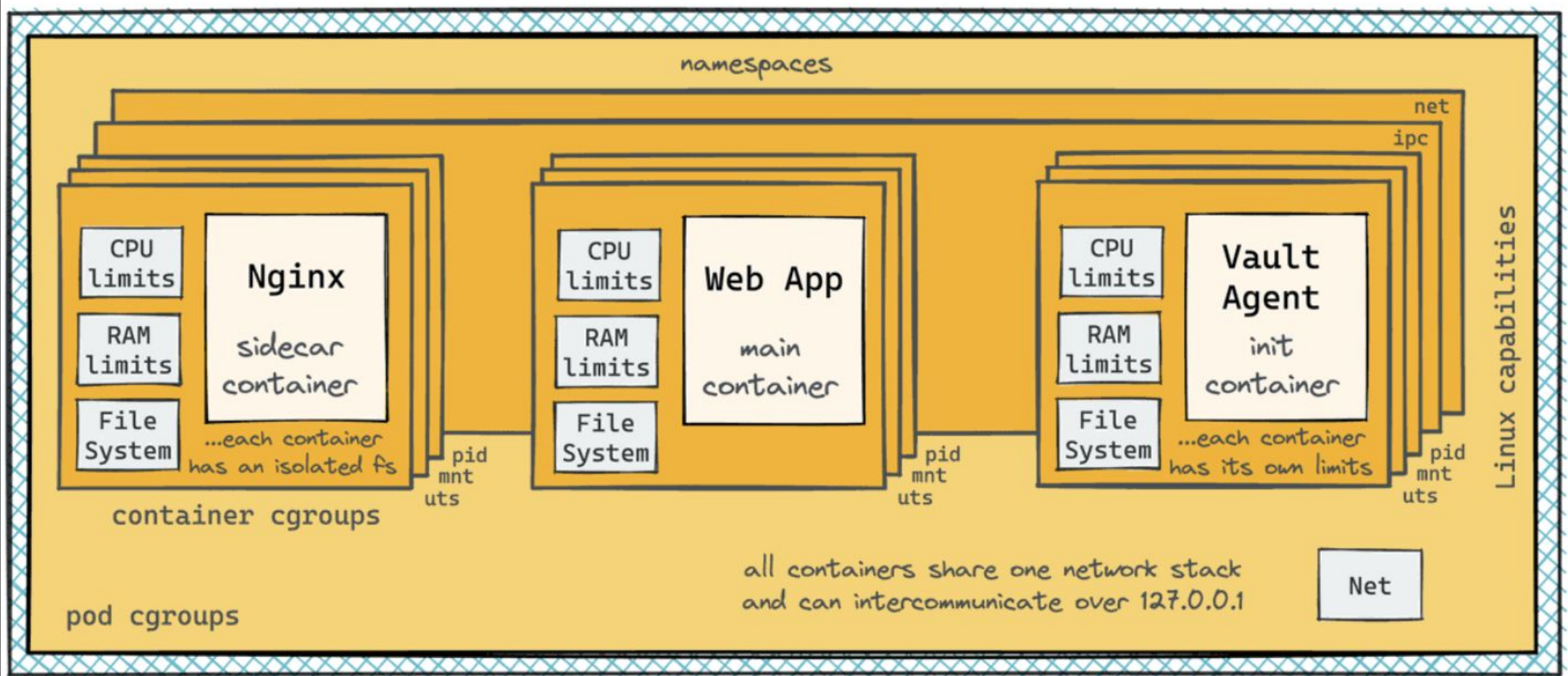
Containers also made application packaging and distribution much more developer-friendly.



6/ 🌐 But containers also brought new challenges, especially in composing them together and managing them at scale.

So what's Kubernetes' role here? Instead of inventing a new containerization method, it emulated VM-based service architecture using containers as building blocks.

Kubernetes Pod - a multitenant "box"



Pod - a group of "semi-fused" containers

all containers in a Pod are addressable via a single IP address. E.g. 10.0.0.3

7/ 🎁 Kubernetes Pods == New VMs. They are the smallest unit you can run. Best part? They combine the perks of both VMs and containers.

Kubernetes simplifies scaling & deploying apps by introducing "Deployments" to handle replication and distribution of Pods across cluster Nodes

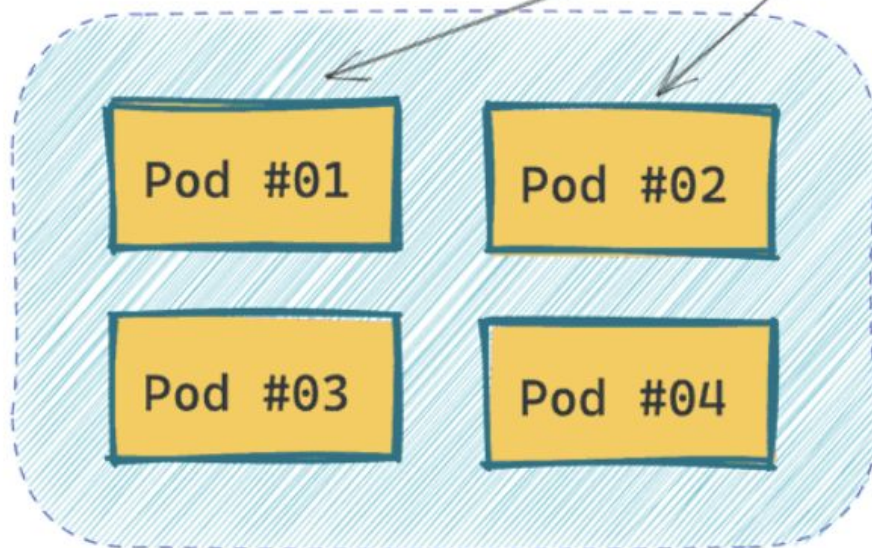
Kubernetes Deployment - a means to replicate "Boxes"

defines a Pod template and a number of copies (replicas)

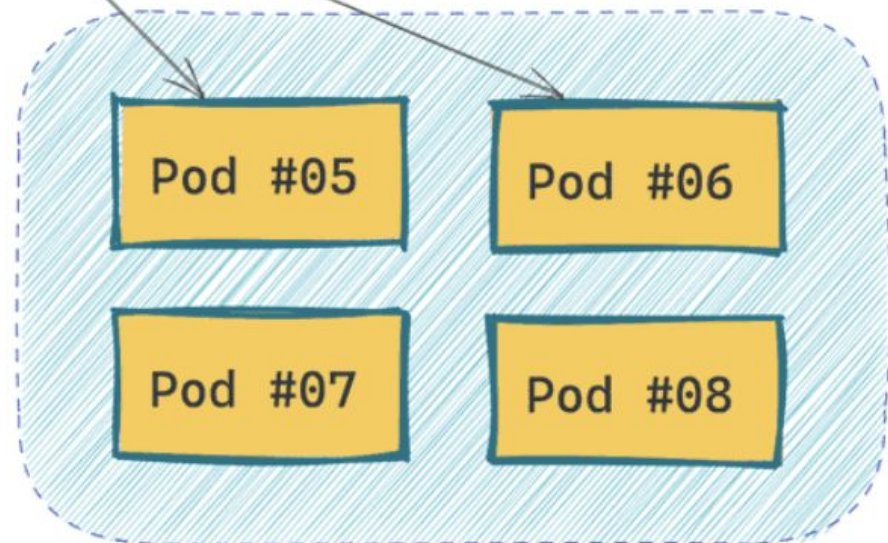


`Pod.metadata.labels` is an important part of the Pod definition!

maintains the required number of replicas for a given version of the Pod template



Node 1

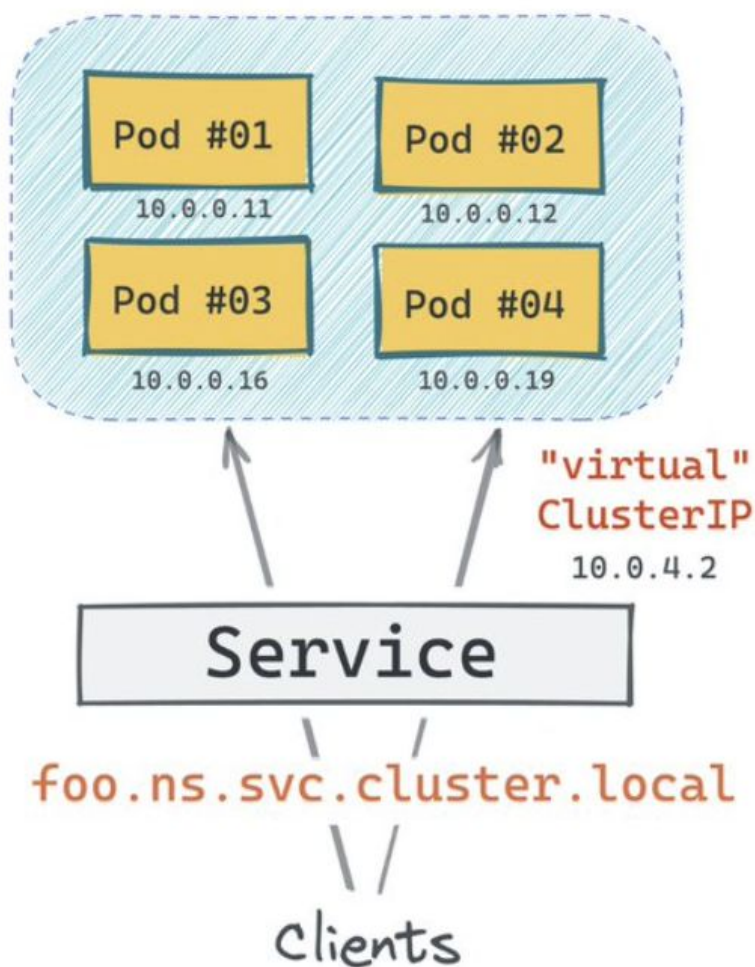


Node 2

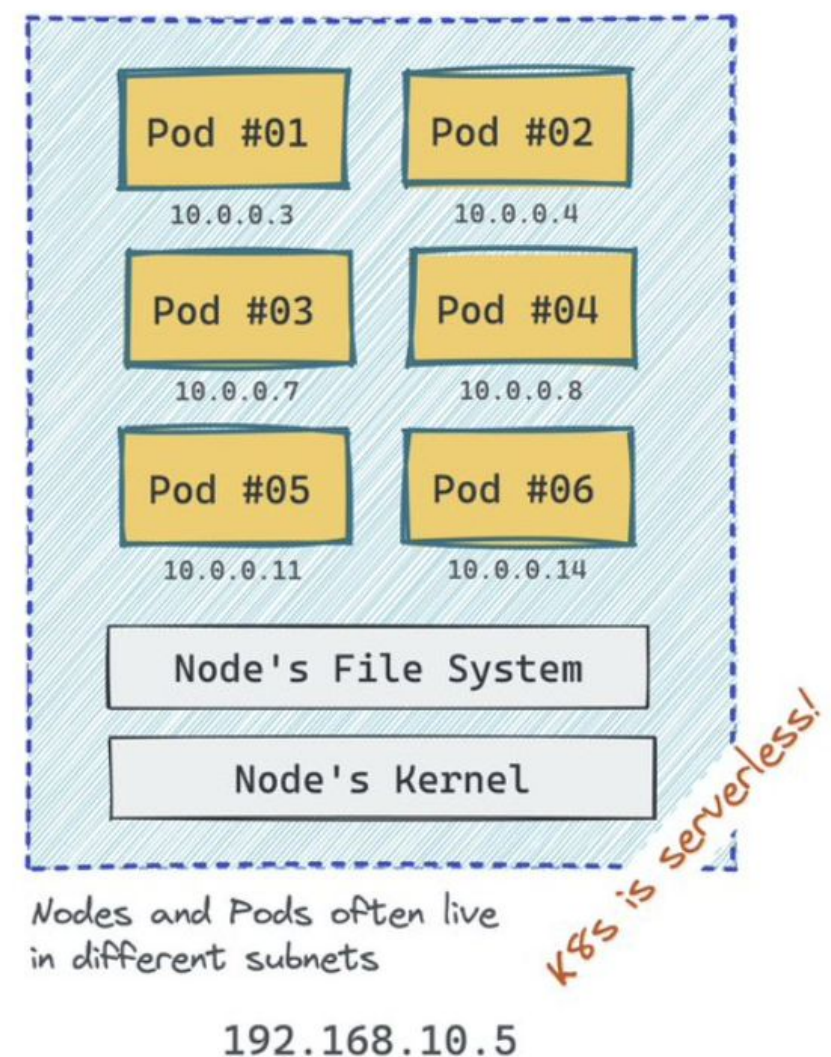
8/ ⚡ Combining the Deployment primitive with built-In Service Discovery makes Kubernetes feel truly serverless.

Unless you want to manage servers of your Kubernetes clusters. But you probably shouldn't - EKS, GKE, and the like were built exactly for that.

Kubernetes Built-In Service Discovery



Kubernetes Node - "invisible" for Developers

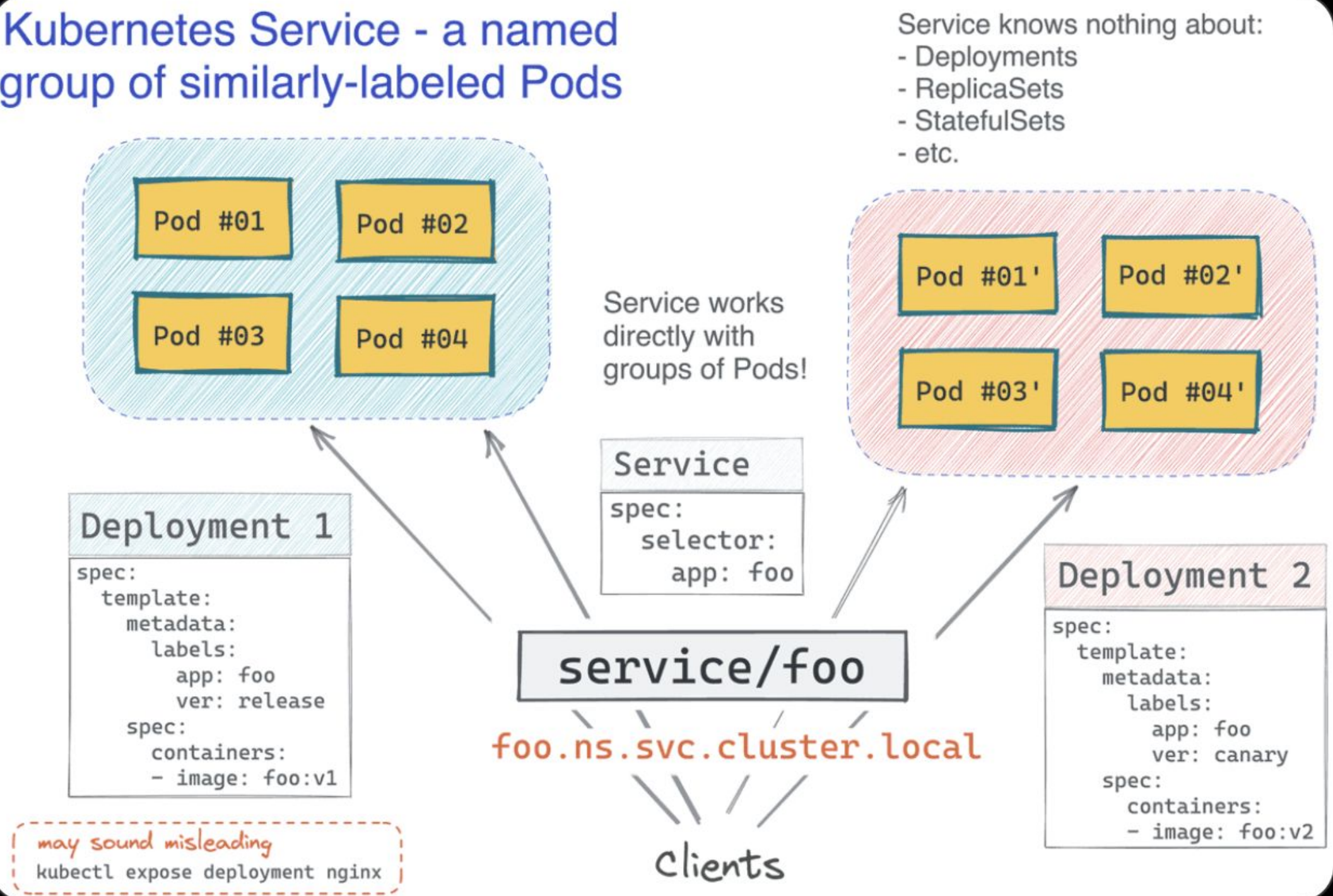


9/ 🔄 Comparing old with new:

- Kubernetes Pods mirror our old friends - VMs.
- Deployments mirror that custom piece of code you used to ship your app to VMs.
- Services mirror, well, VM-based services, including the discovery logic.

A smart move to tap into existing know-how.

Kubernetes Service - a named group of similarly-labeled Pods

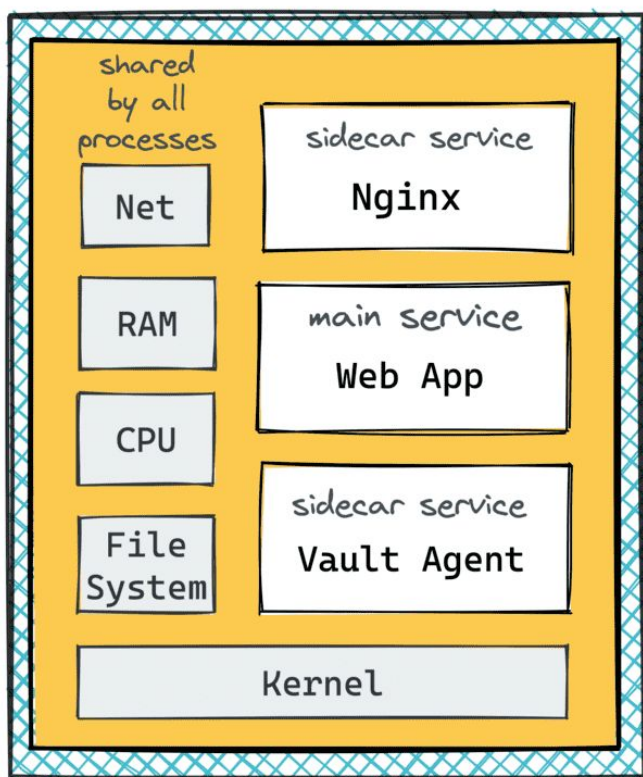


10/ 💡 Final thoughts: Before jumping on Kubernetes, understand your needs.

Sometimes, traditional methods are still more appropriate. And often a higher-level and even more managed solution like AWS App Runner would suffice. Always aim for what's best for your specific context.

Virtual Machine - a "Box"

...or real!



typically has a dedicated address

e.g. 192.168.10.5

Service - a named group of identical "Boxes"

distributed!

