Microsoft Dev Group {Cobán}

Comparte | Aprende | Desarrolla

.NET Core

José Diaz

Java Champion - JUG Leader of PERU JUG

Arquitecto de Software

Co-fundador de JoeDayz.pe

Microservicios con .NET Core

Webinar - Junio 6, 18:00 CST.

José Amadeo Díaz Díaz

Gerente de Arquitectura y
Productos Digitales

jadiaz@farmaciasperuanas.pe

@jamdiazdiaz

Fundador de JoeDayz.pe

Java Champion

Miembro de @PeruJUG

¿Por qué micro servicios?

+

El Mercado es volátil, incierto, complejo y ambiguo

+

Los negocios deben innovar rápido

+

Entregar software rápido, frecuente y sostenible

Arquitectura Monolítica - Las ilusiones van disminuyendo con el tiempo

Tamaño/

Complejidad

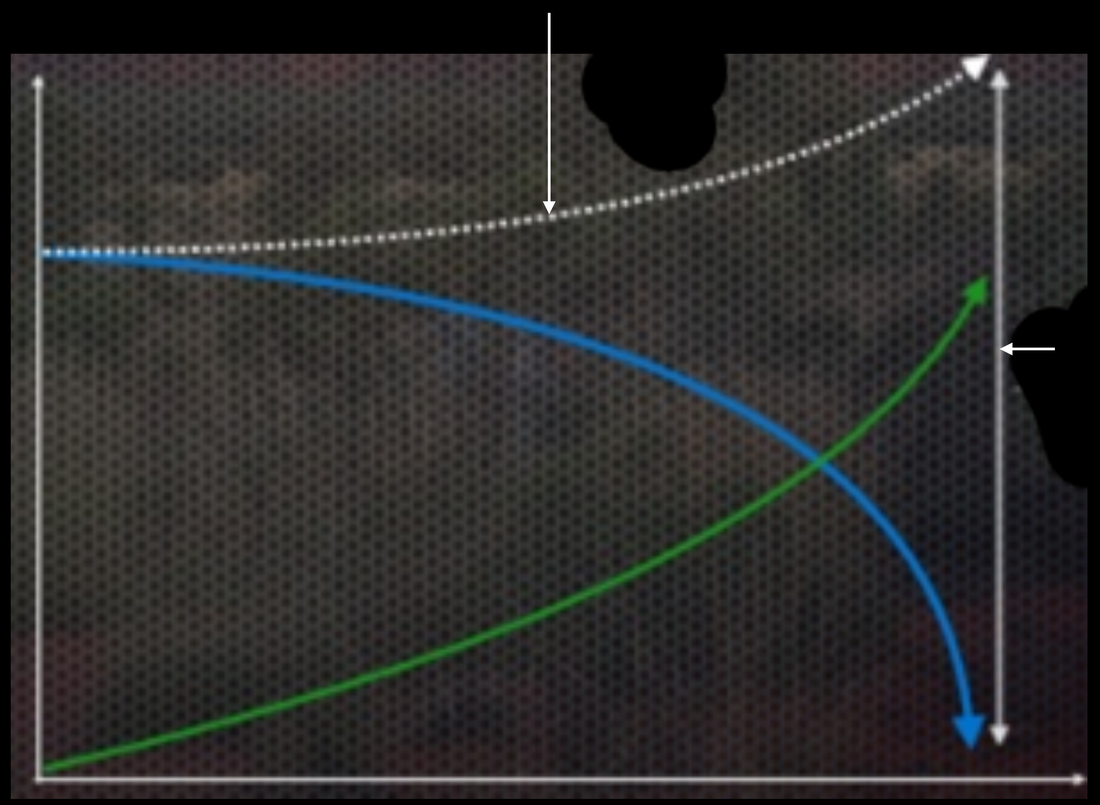
Mantenimiento

Testeable

Desplegable

Modular

Evolutivo



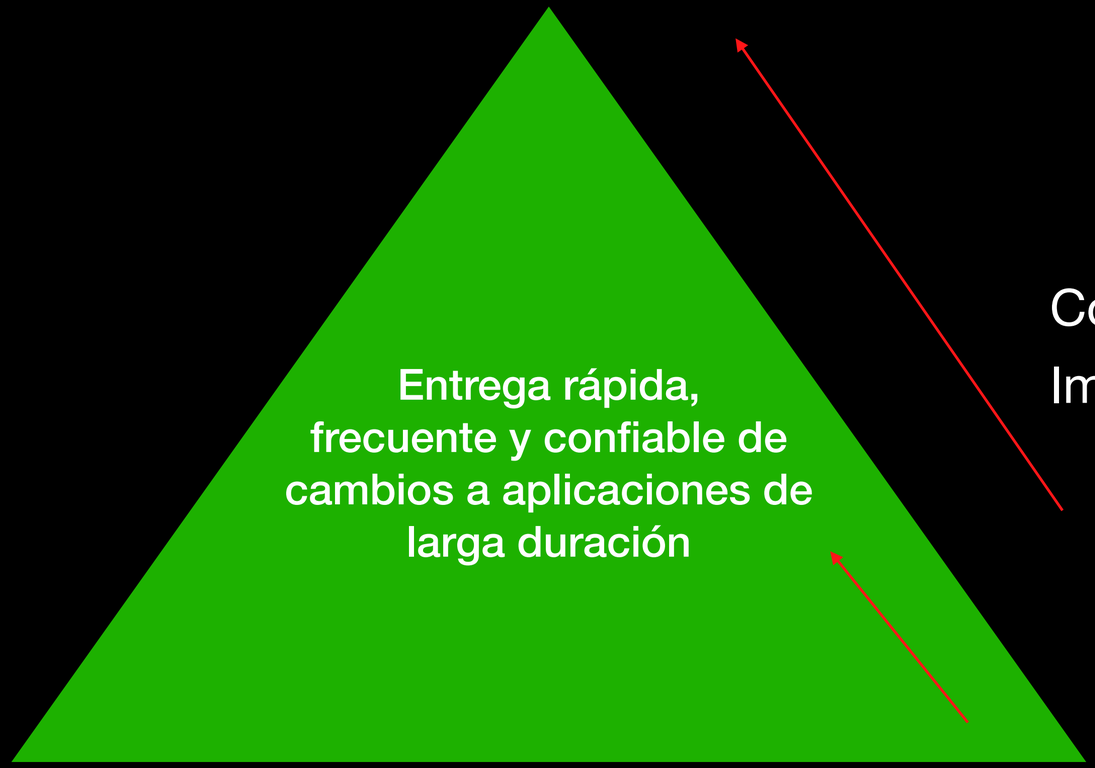
La arquitectura de micro servicios es un estilo de arquitectura que estructura una aplicación como un conjunto de servicios*

Cada ~~micro~~servicio es:

- Altamente mantenible y testeable
- Pobremente acoplado
- Desplegable de forma independiente
- Organizado por capacidades de negocio
- Propiedad de un equipo pequeño

* Comienza con un servicio por equipo hasta que esto sea un problema.

Proceso: Lean + DevOps/Delivery Continuo & Despliegue



Comprobable

Implementable

Modularidad
Evolutivo
Mantenible

Organización:

Pequeña, autónoma,
Equipos por **Producto**.

Modularidad

Arquitectura:
Micro servicios

Agenda

- ¿Por qué micro servicios?
- Refactorizando el Monolito
- Implementando nuevas características como servicios
- Extrayendo módulos en servicios
- Tecnologías para micro servicios
- Patrones de Diseño

Refactorizando el Monolito

No, lo hagas!

Anti-patrón: Polvos mágicos

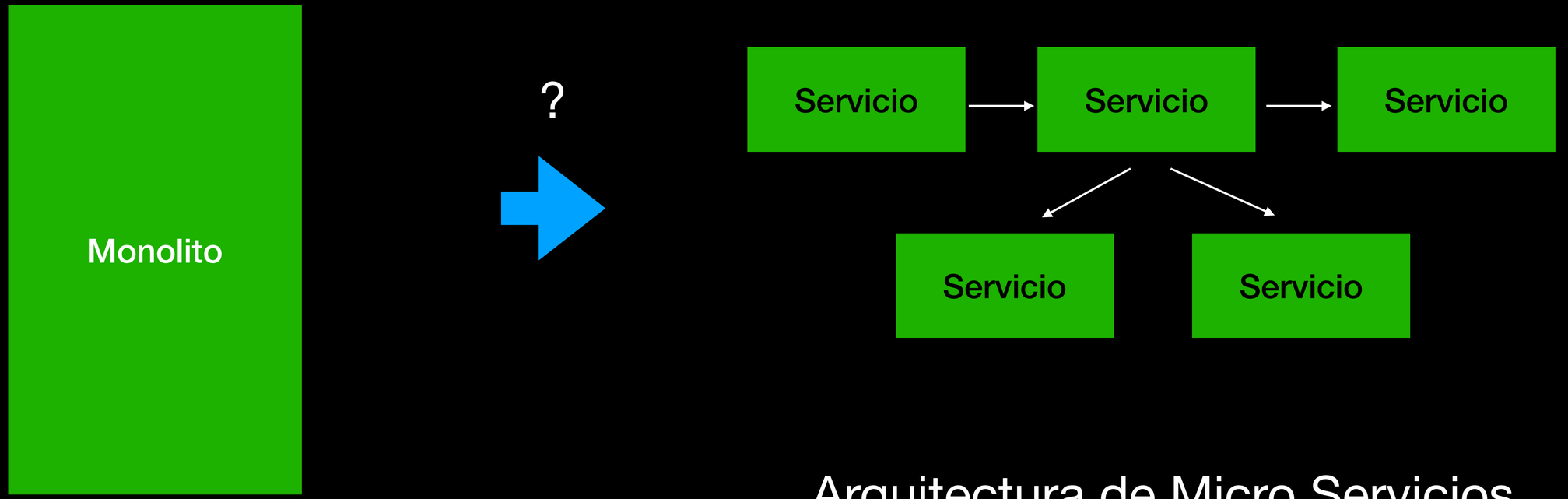
Aprovechemos la arquitectura monol

- La arquitectura monolítica no es un anti-patrón
- Si la entrega de software es lenta
    - Optimiza el proceso de desarrollo
    - Optimiza el pipeline de despliegue = mas automatización
    - Optimiza la autonomía del equipo
        - Modulariza el monolítico
        - Crea equipos multi-funcionales
- Si el stack de tecnología es obsoleto, moderniza a un nuevo monolito.
- …

Si y sólo si, esto no es suficiente*,
entonces considera
migrar a micro servicios

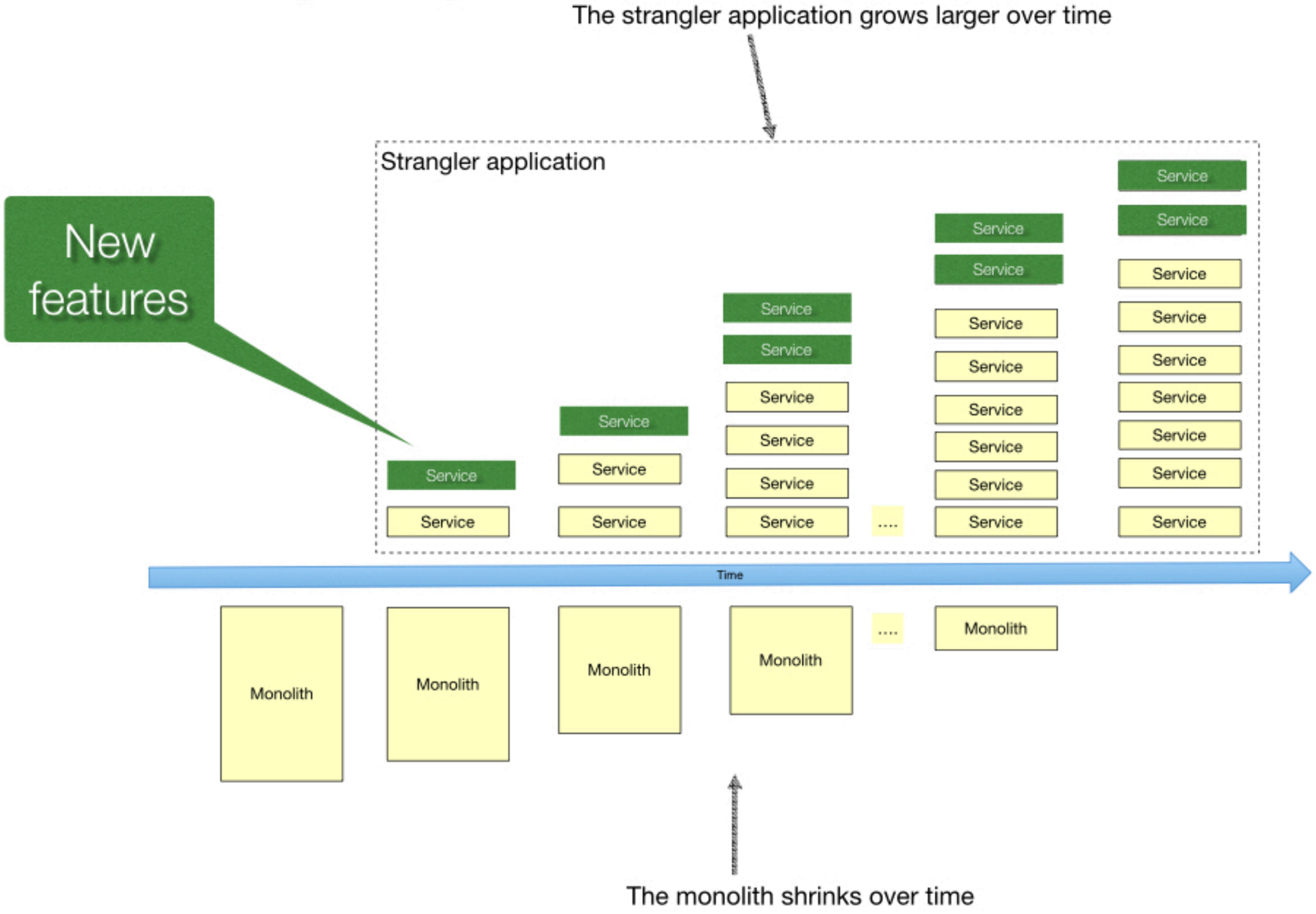
* Grandes y complejas aplicaciones son desarrolladas (en general) por un gran equipo que necesita entregar rápido, de forma frecuente y sostenible.

¿Cómo descomponemos nuestra gran aplicación monolítica?

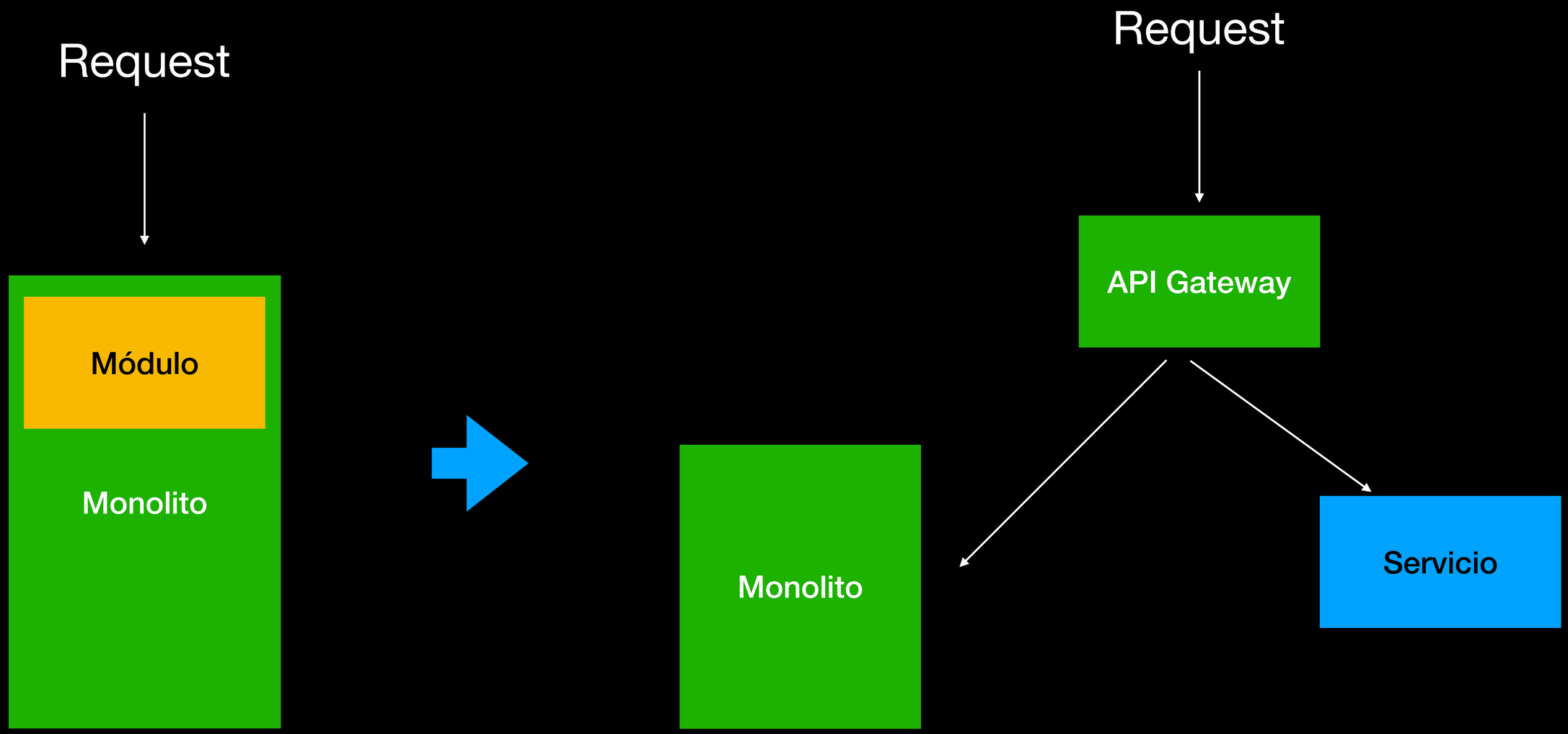


Arquitectura de Micro Servicios

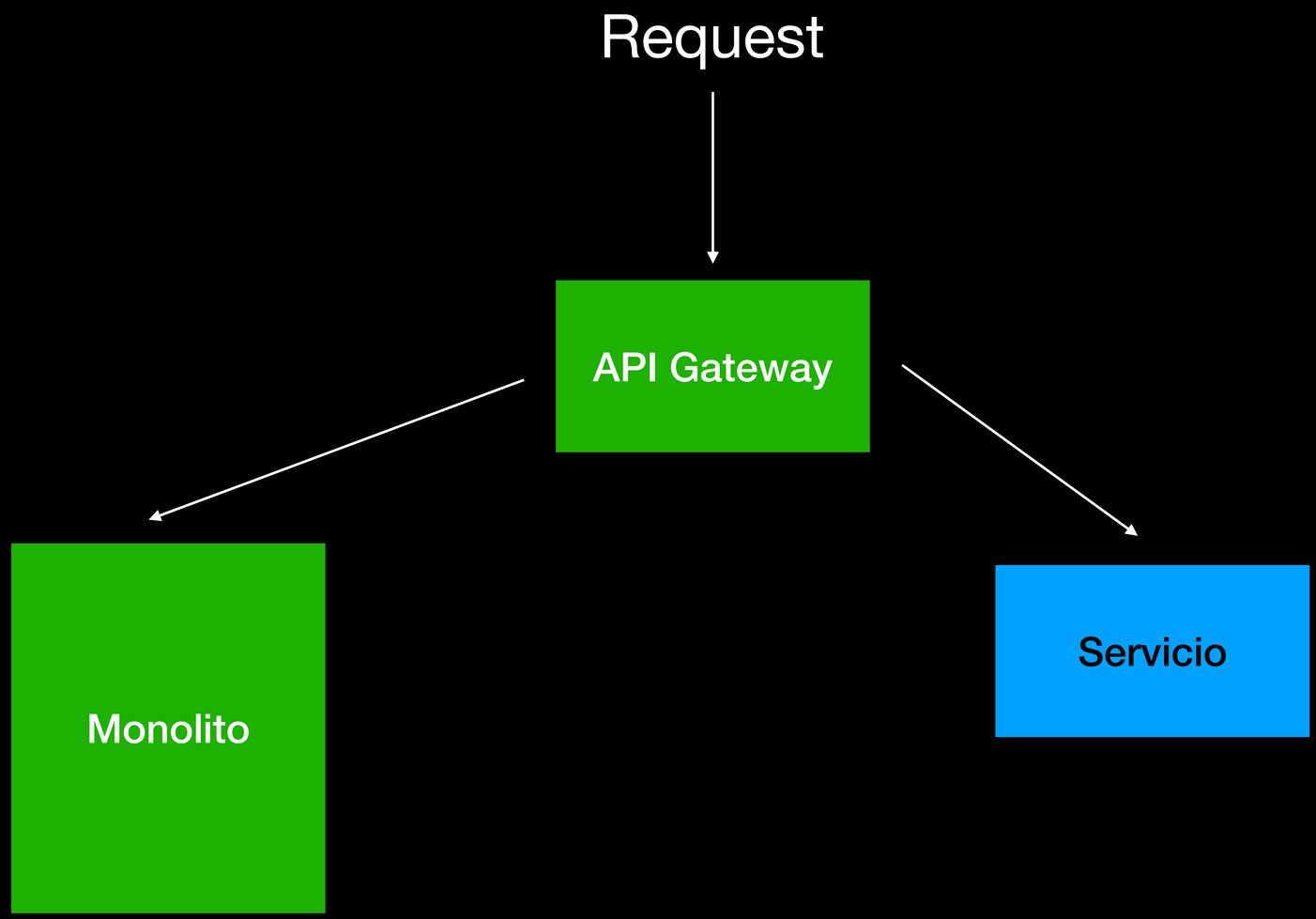
Strangling the monolith



Iterativo: Modulo => Servicio



Iterativo: Nueva Característica => Servicio



Mide el éxito

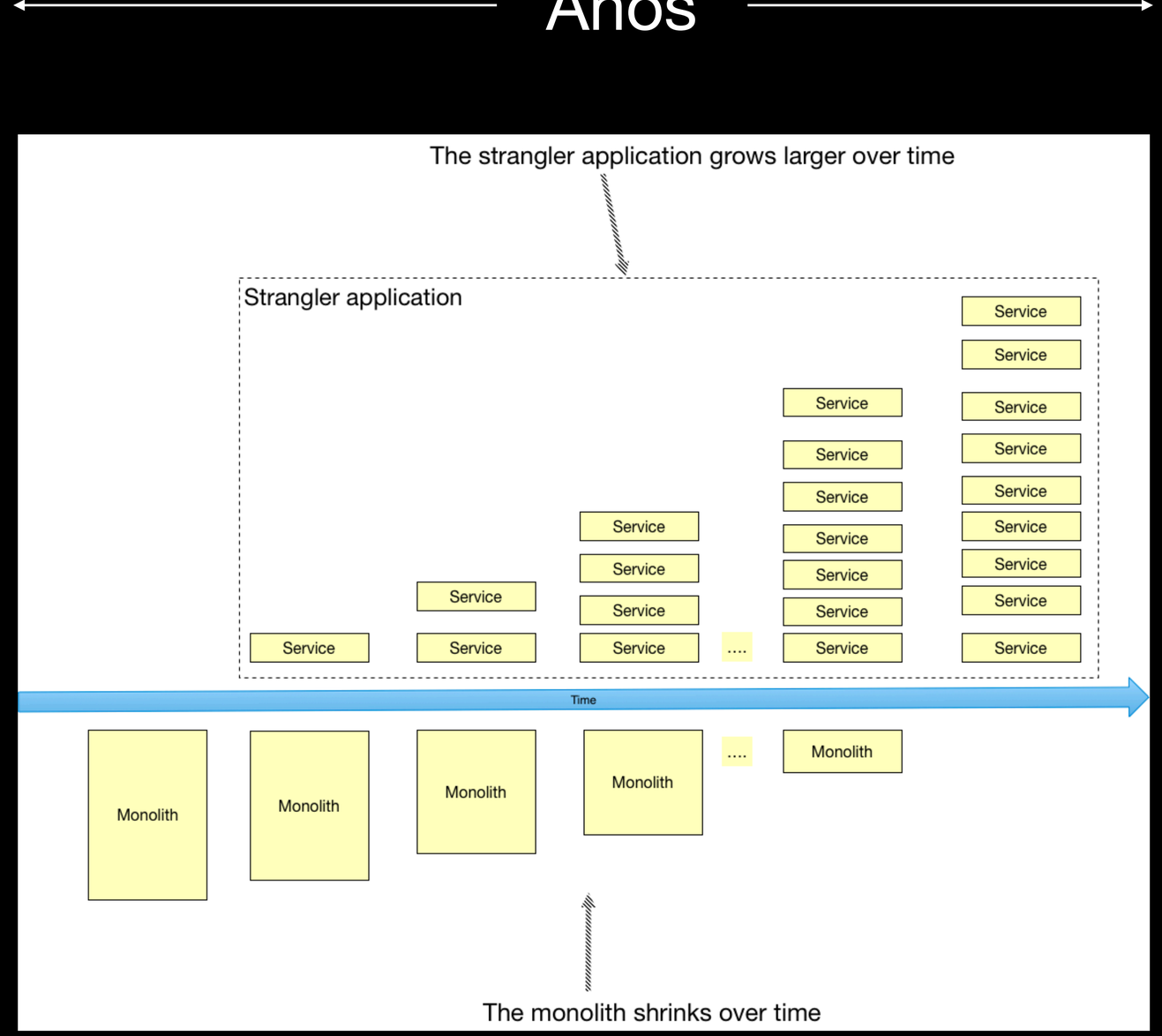
- Exito != número de micro servicios
- Mejora tus métricas:
    - Reduce el tiempo de espera
    - Incrementa la frecuencia de despliegue
    - Reduce el % de fallas por cambios
- Mejora en nuevos “features”
- …

Anti-patrón:

Micro servicios es el objetivo!

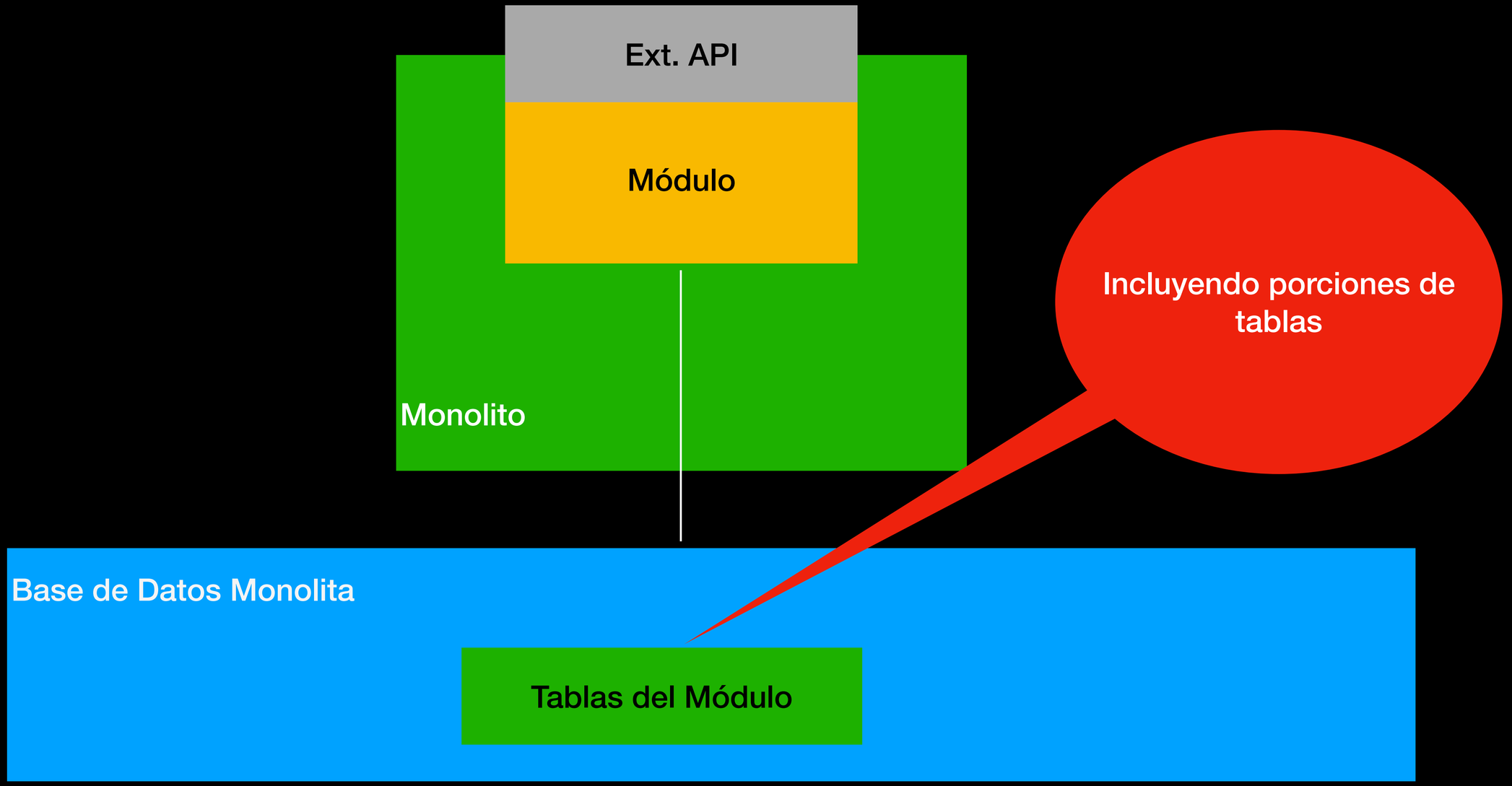
Repetir extraer
servicios hasta que:

- Eliminar el monolítico
- Problemas de entregas de software solucionados
- Trabajo con alta prioridad

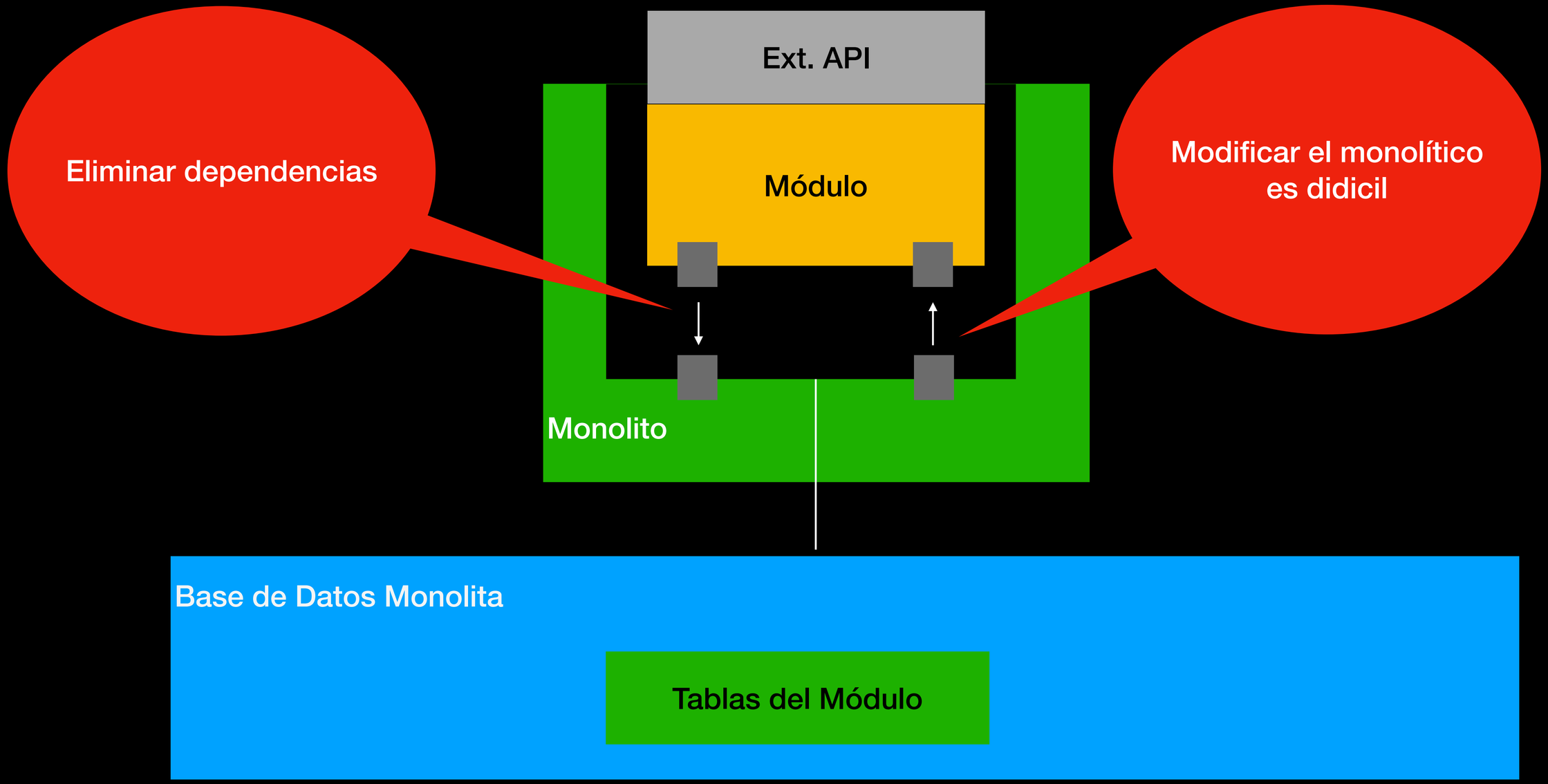


Extrayendo módulos en servicios

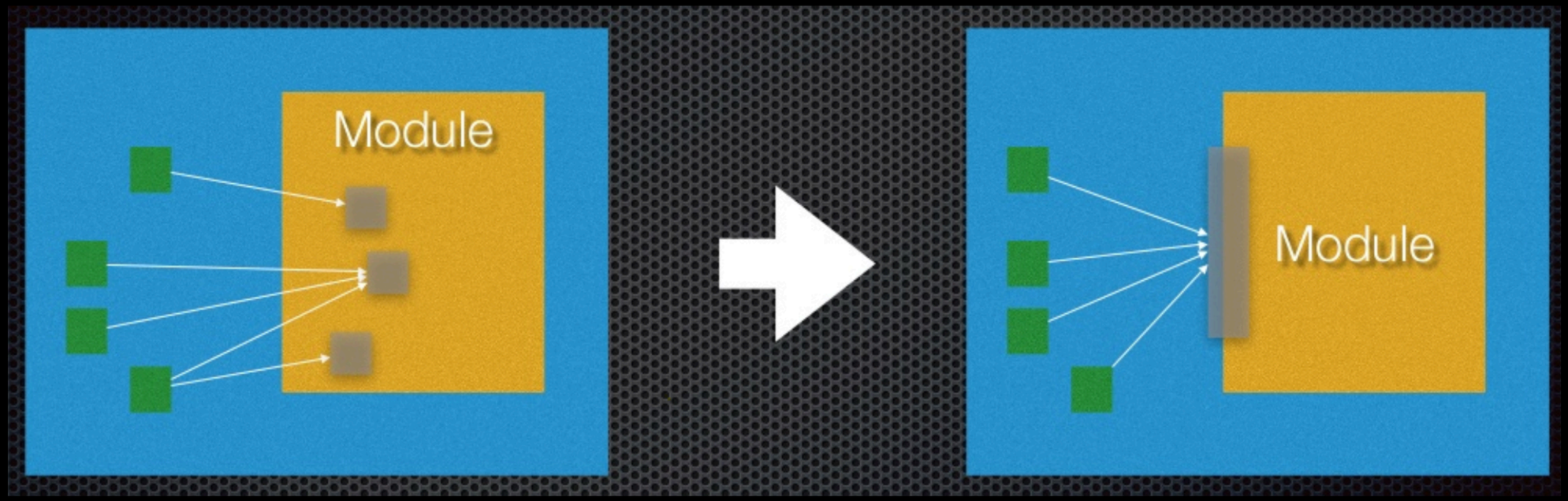
Módulo => Servicio …



Definiendo una interface “remotable” bidireccional

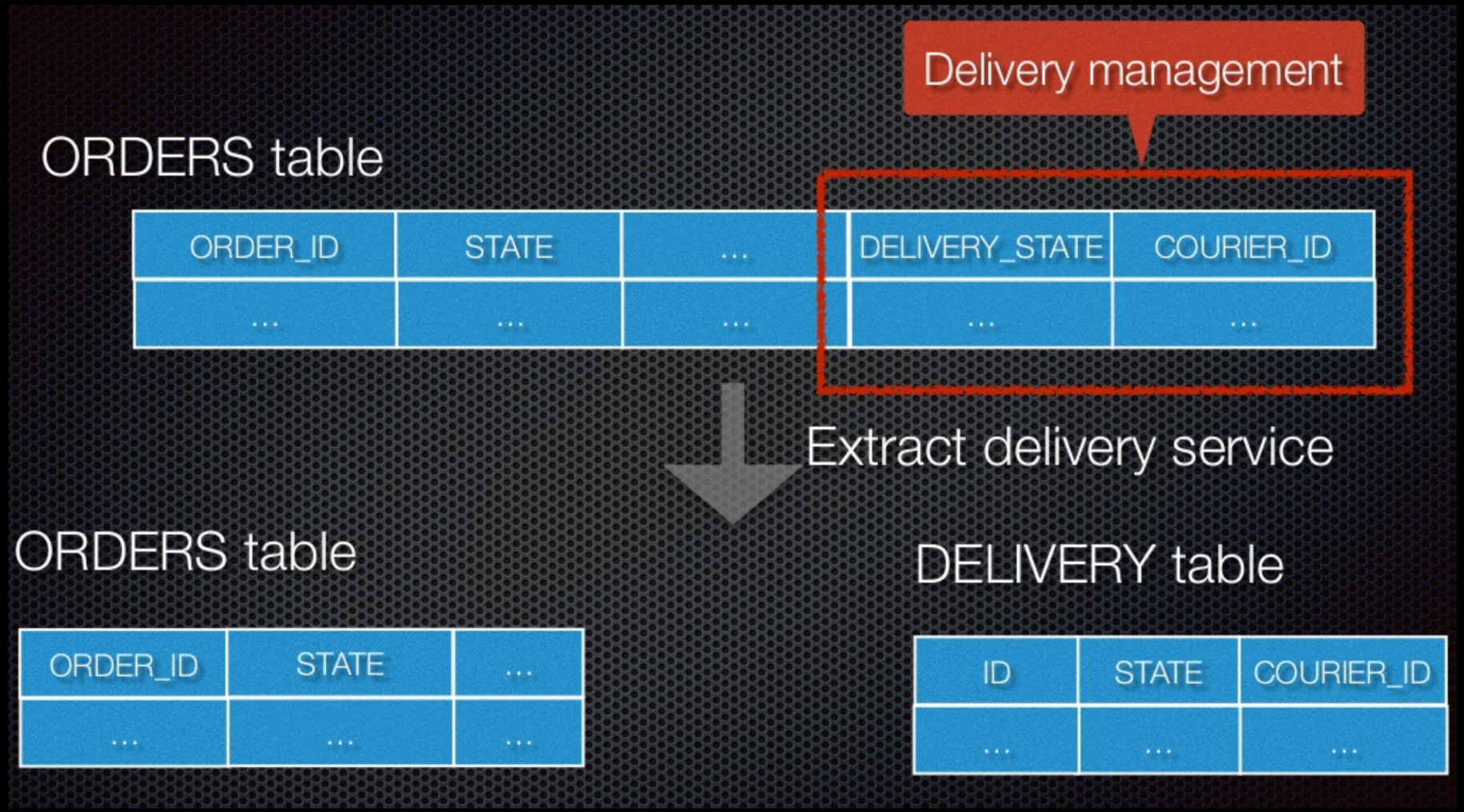


Incrementalmente modificar el monolito invocando al modulo vía nueva API*

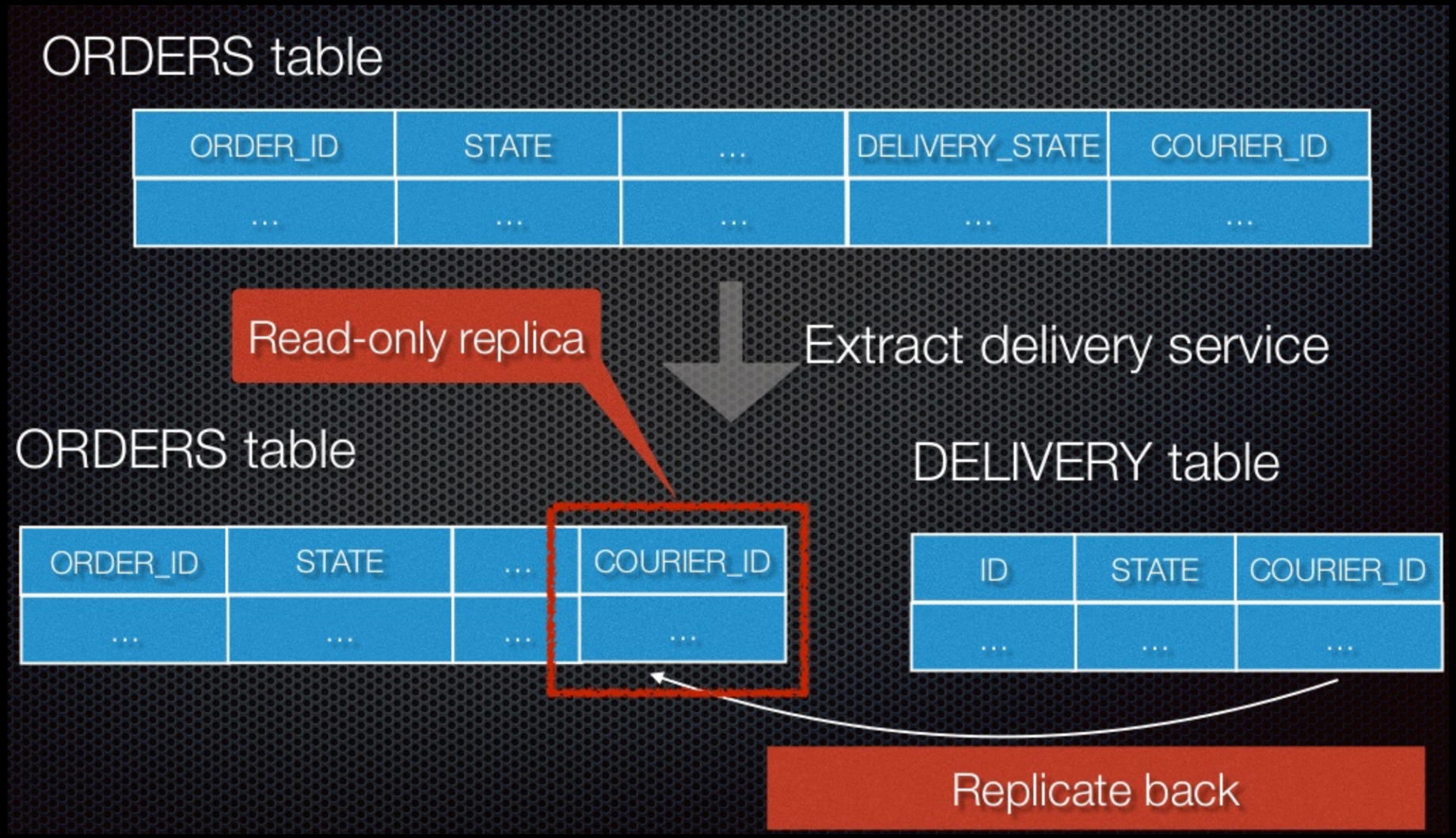


Patron: Branch by abstraction

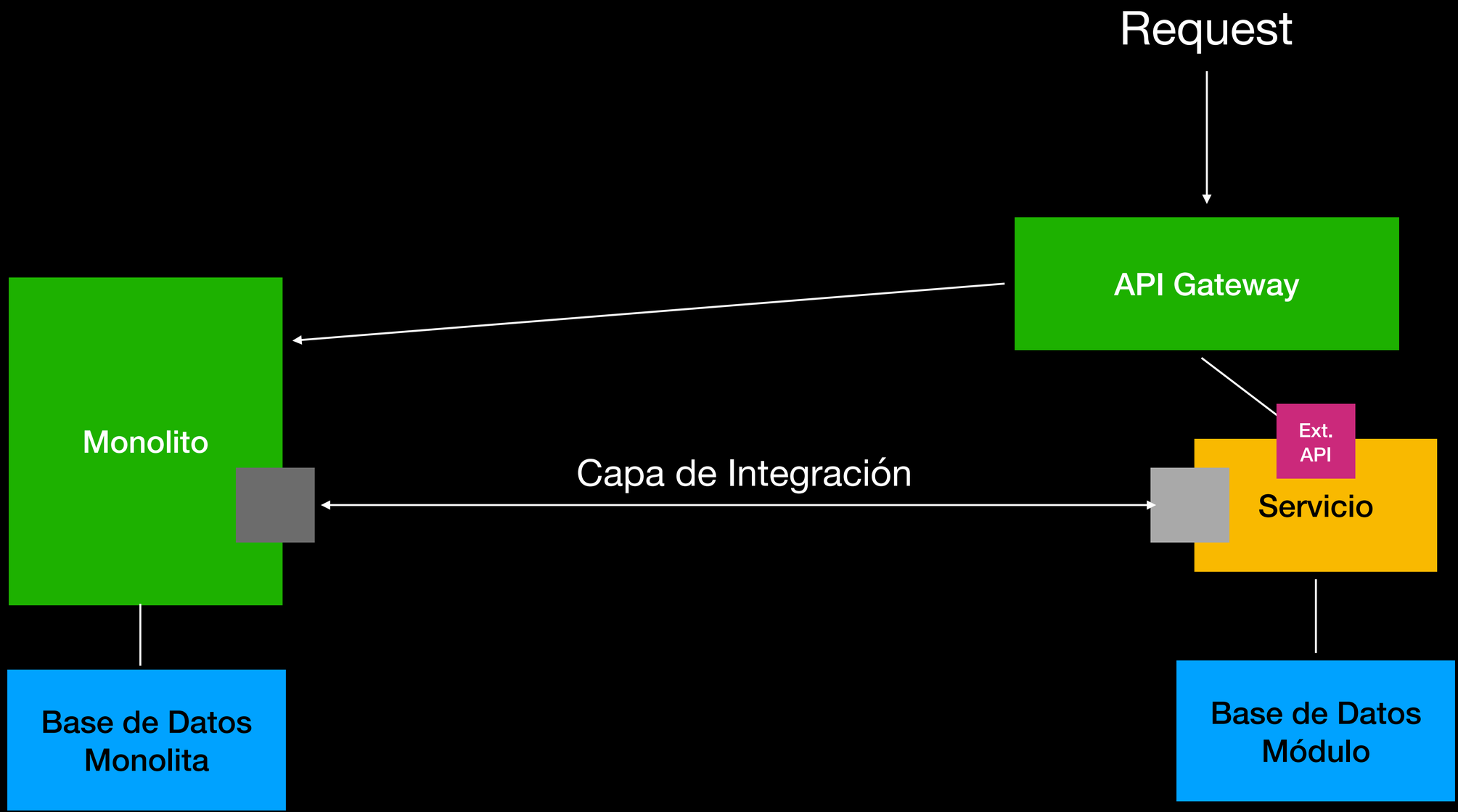
Tranformar el esquema de base de datos



Replicar datos reduce el alcance del refactoring



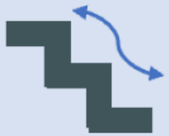
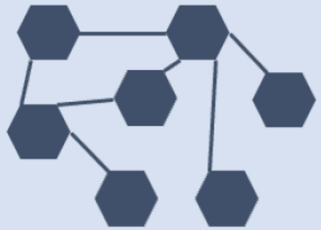
Modulo => Servicio



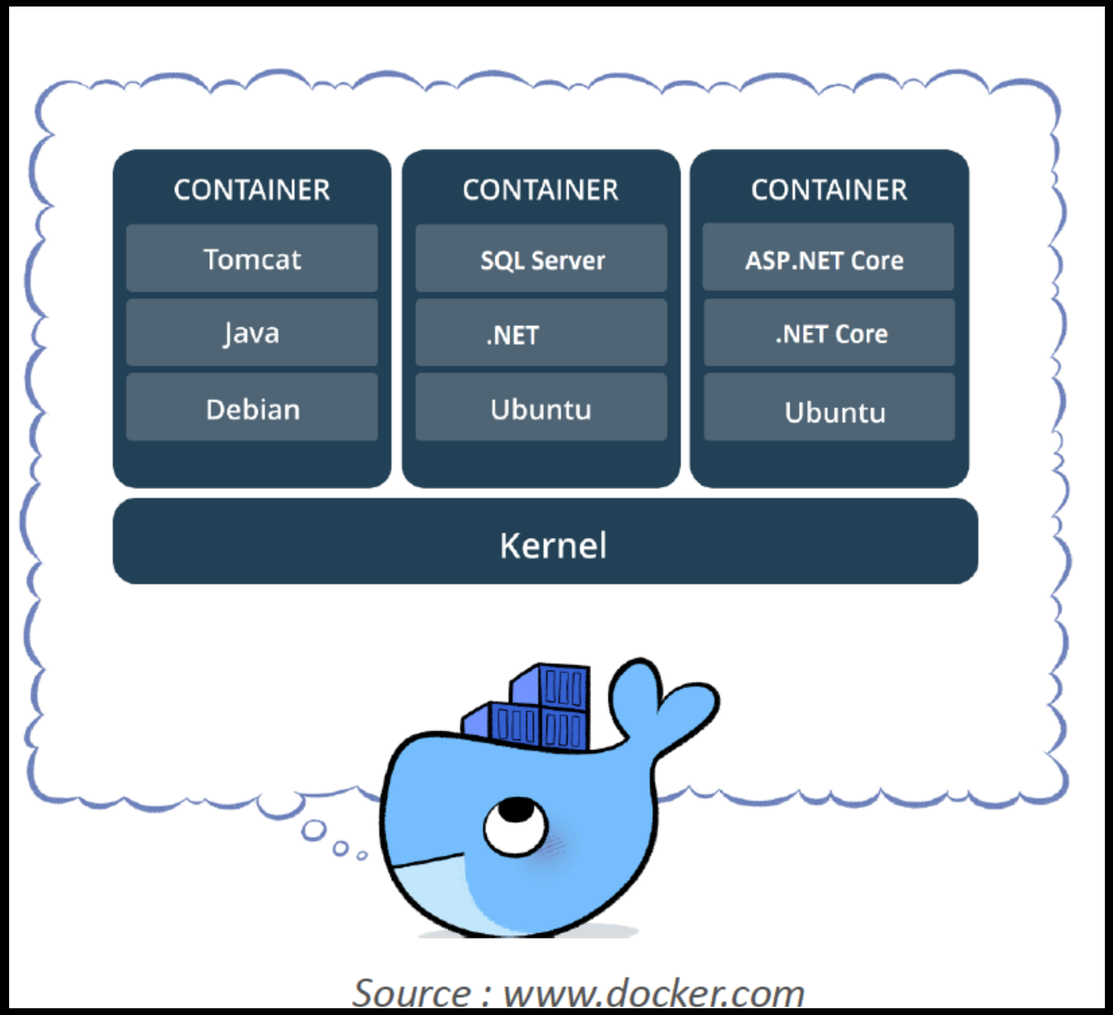
Agenda

Introducción a Contenedores Docker

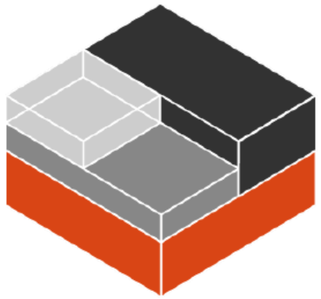
Evolución de la Computación

Development Process	Application Architecture	Deployment and Packaging	Application Infrastructure
Waterfall 	Monolithic 	Physical Server 	Datacenter 
Agile 	N-Tier 	Virtual Servers 	Hosted 
DevOps 	Microservices 	Containers 	Cloud 

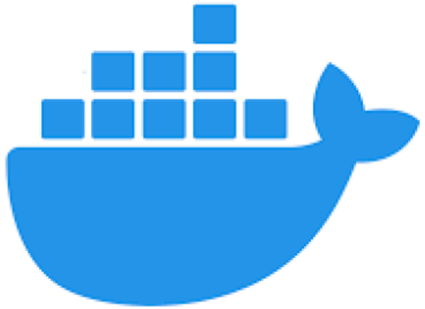
¿Qué es un contenedor?



Plataformas de Contenedores



Linux Containers (LXC)



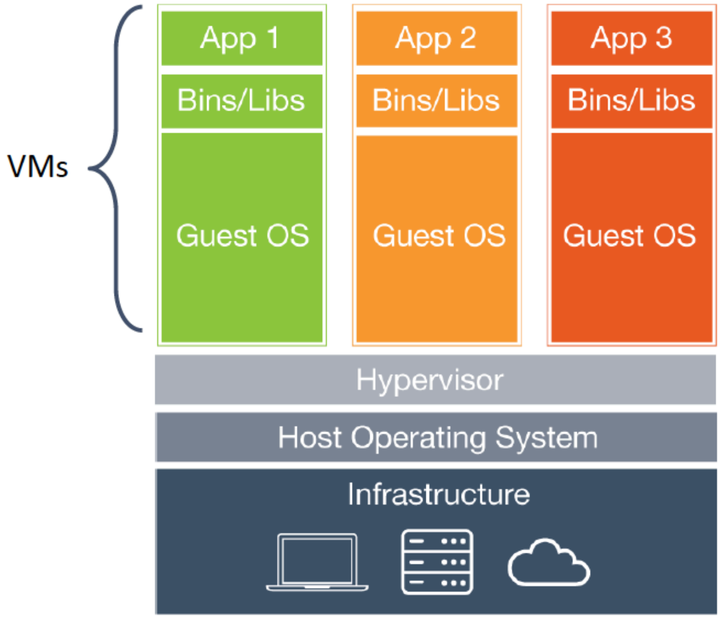
Docker (Docker Swarm)

Kubernetes

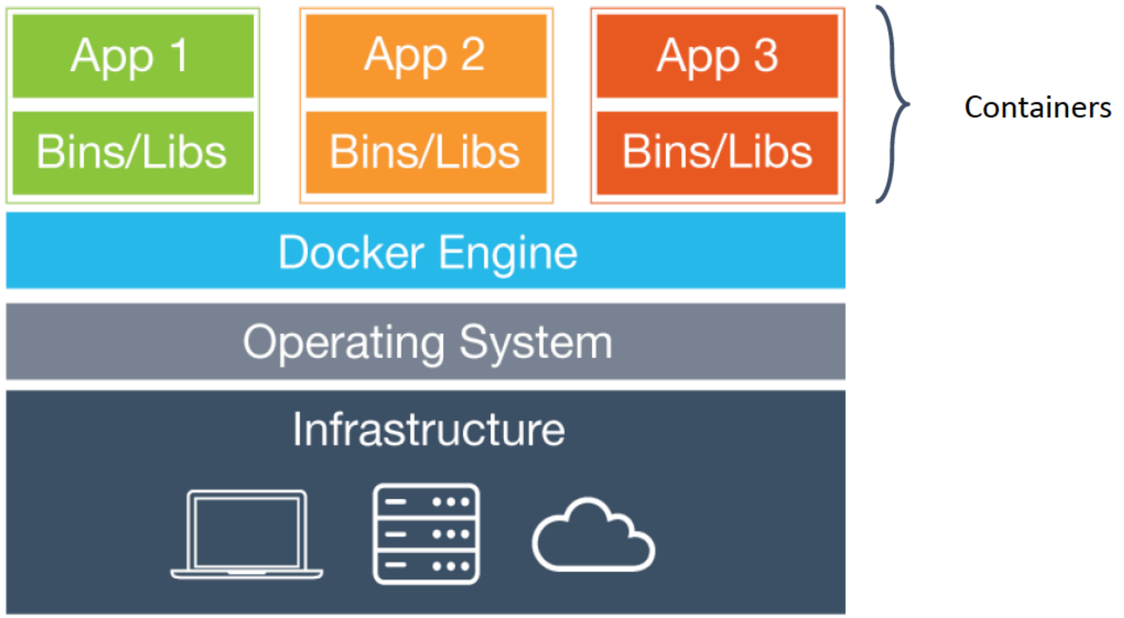
RedHat OpenShift

DC/OS

Maquinas virtuales vs Contenedores

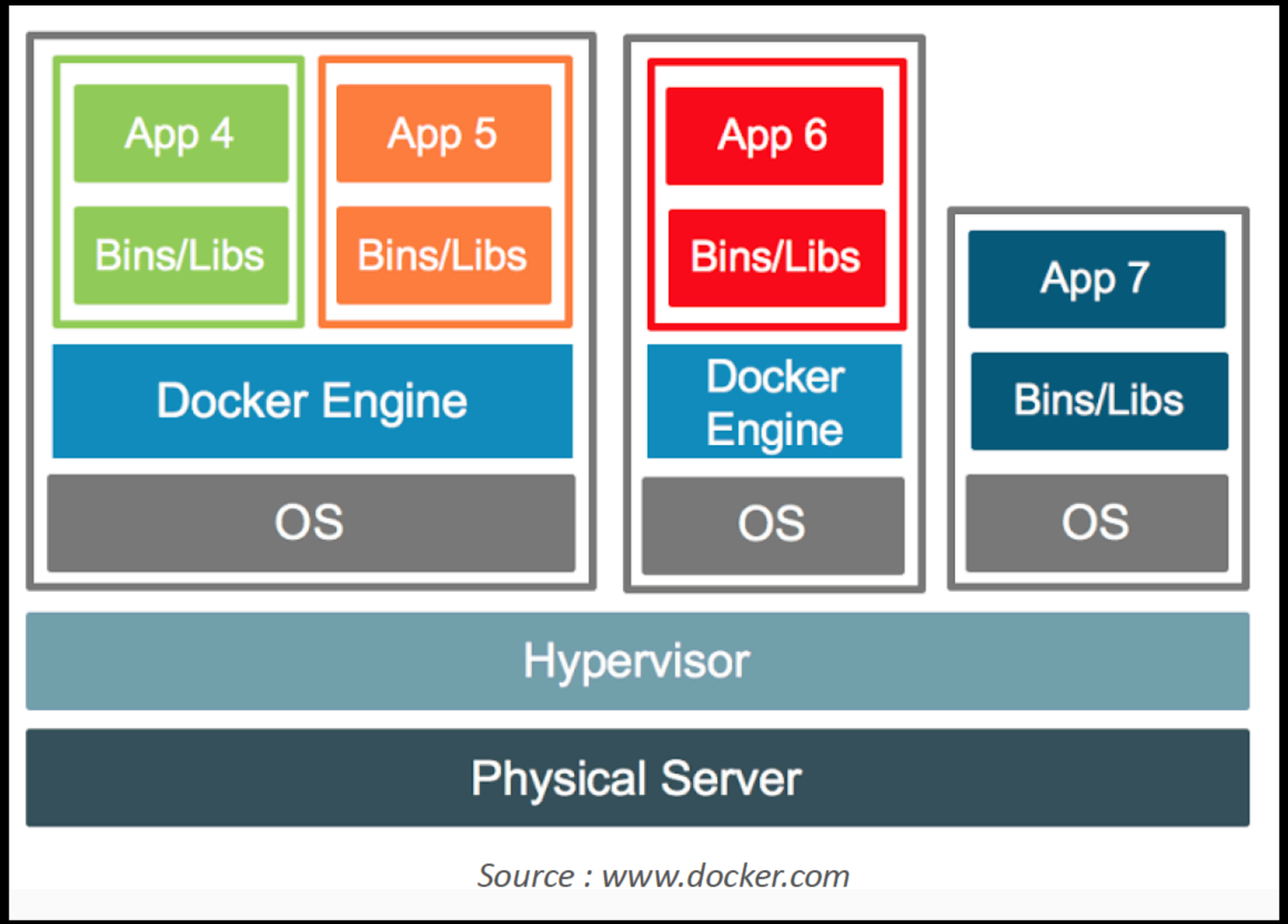


Source : www.docker.com



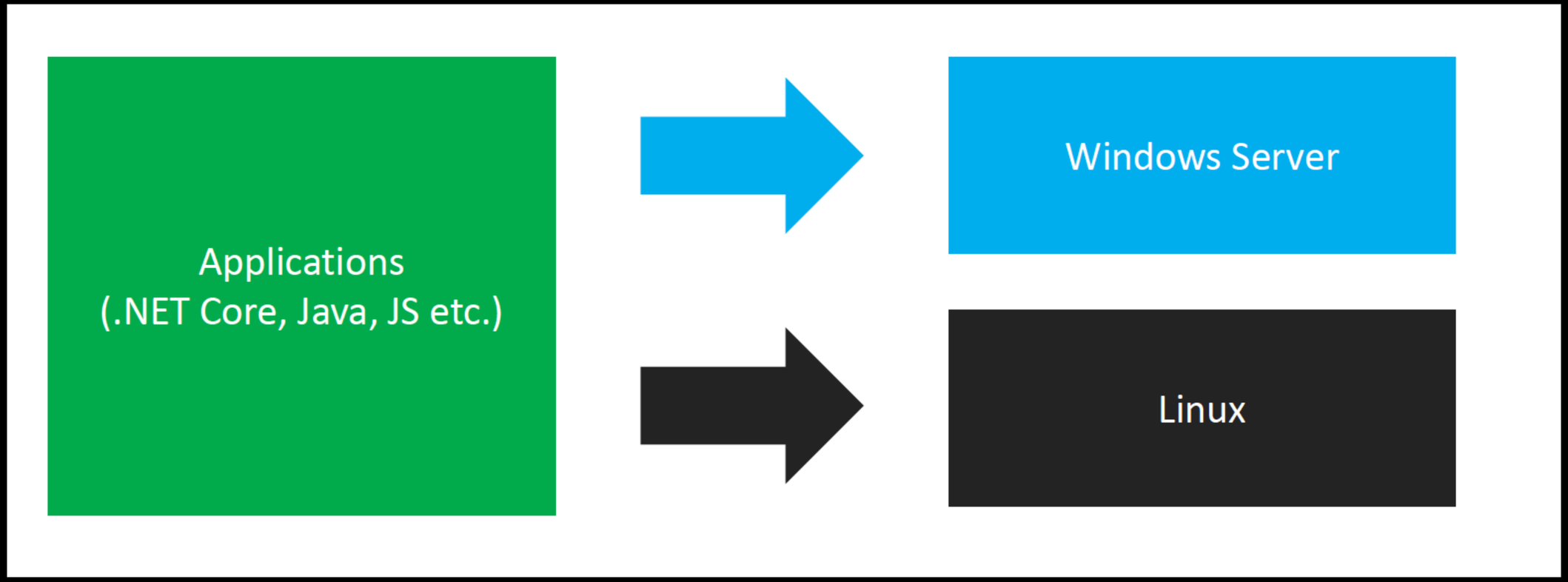
Source : www.docker.com

Contenedores y VMs juntos



Source : www.docker.com

Docker para Desarrolladores



Docker para Aplicaciones .NET

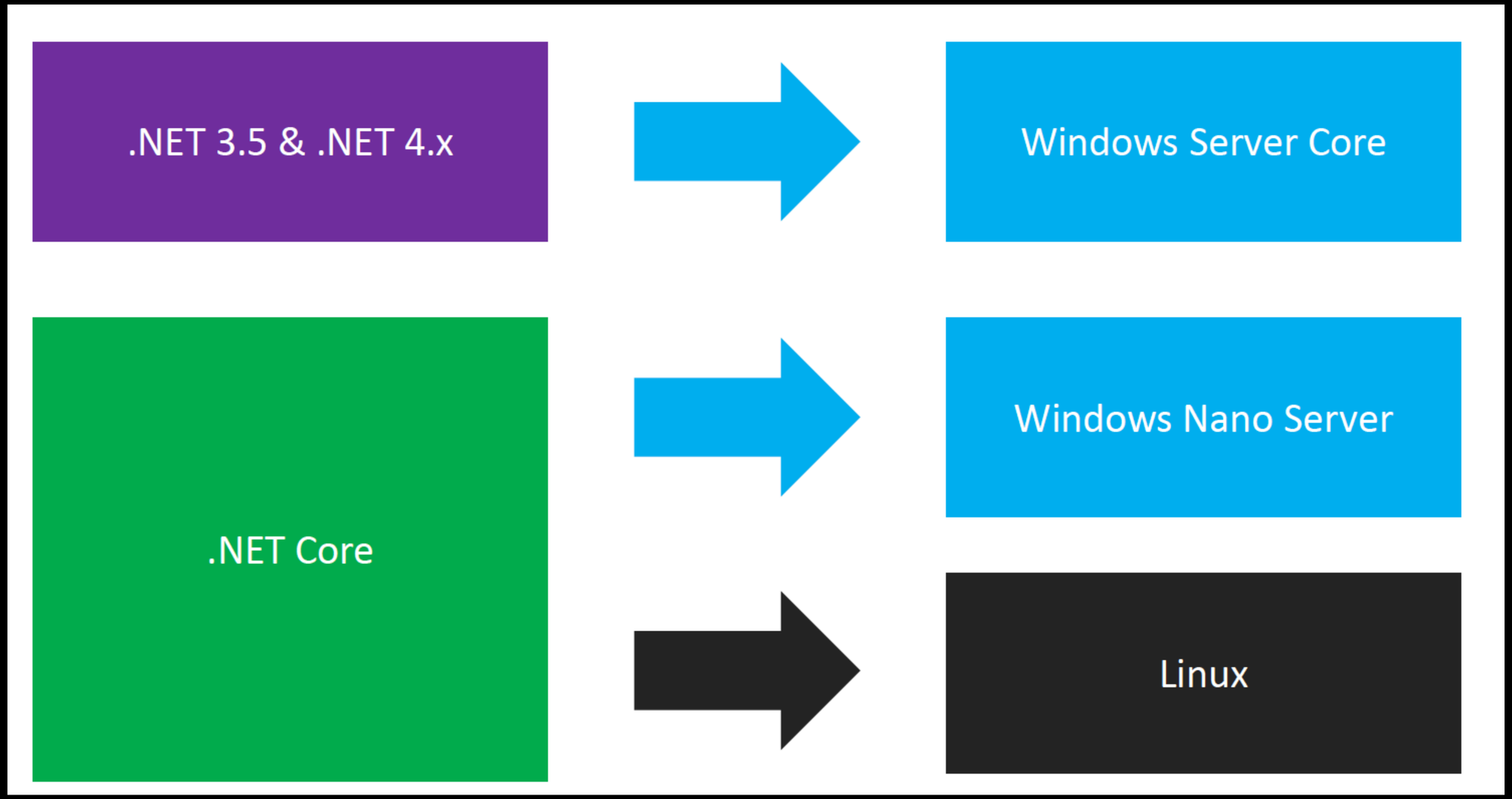
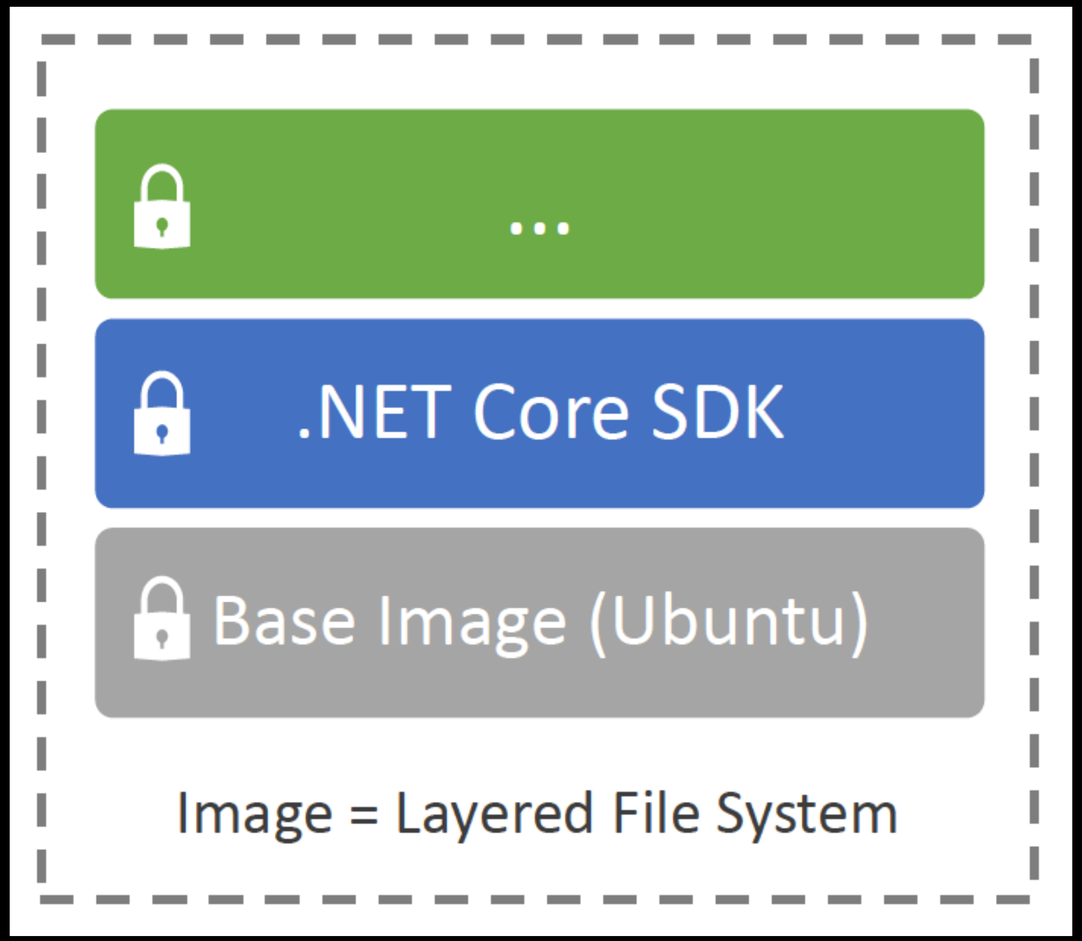
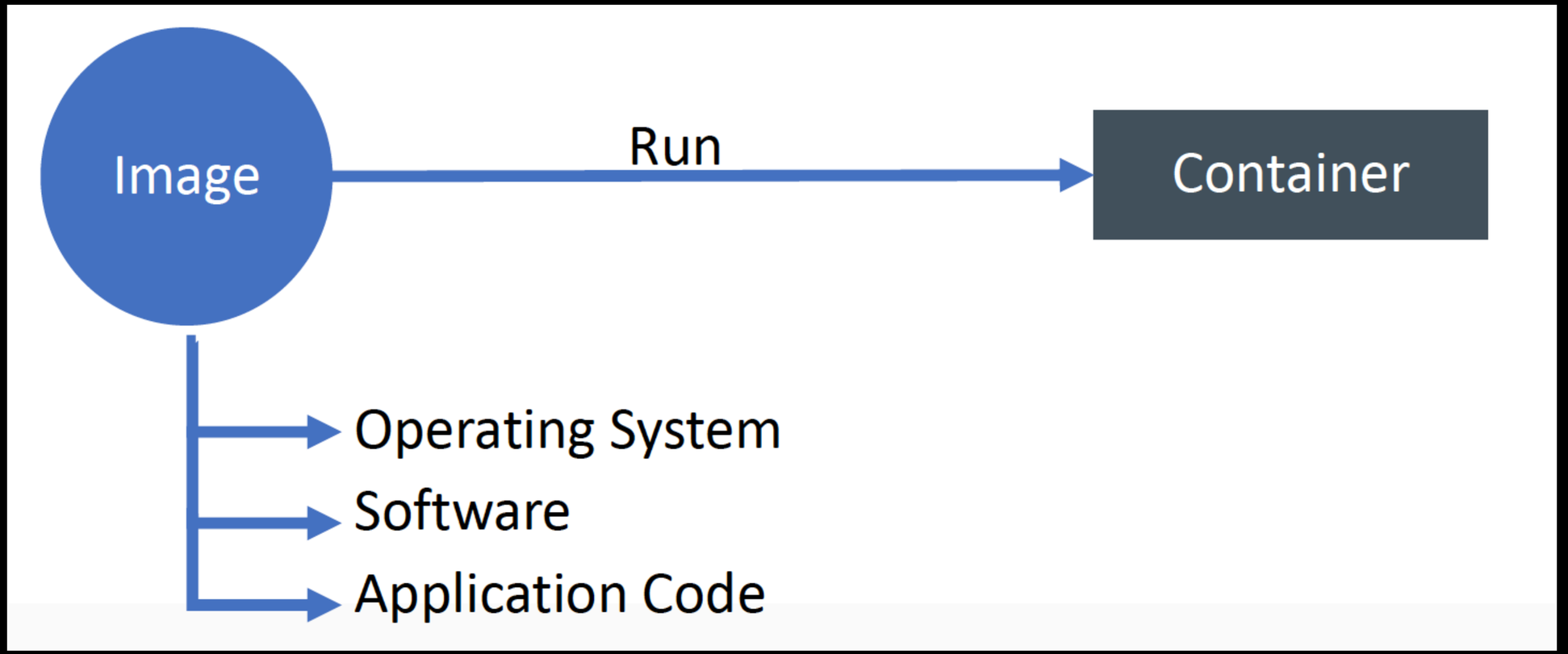


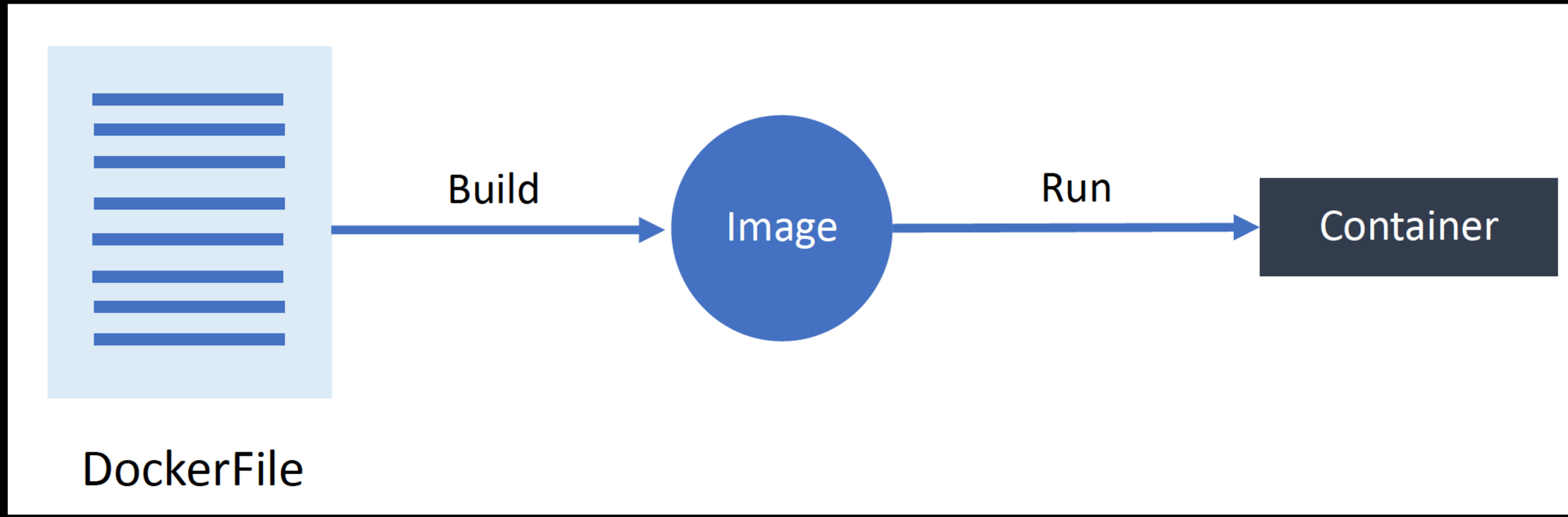
Imagen Docker



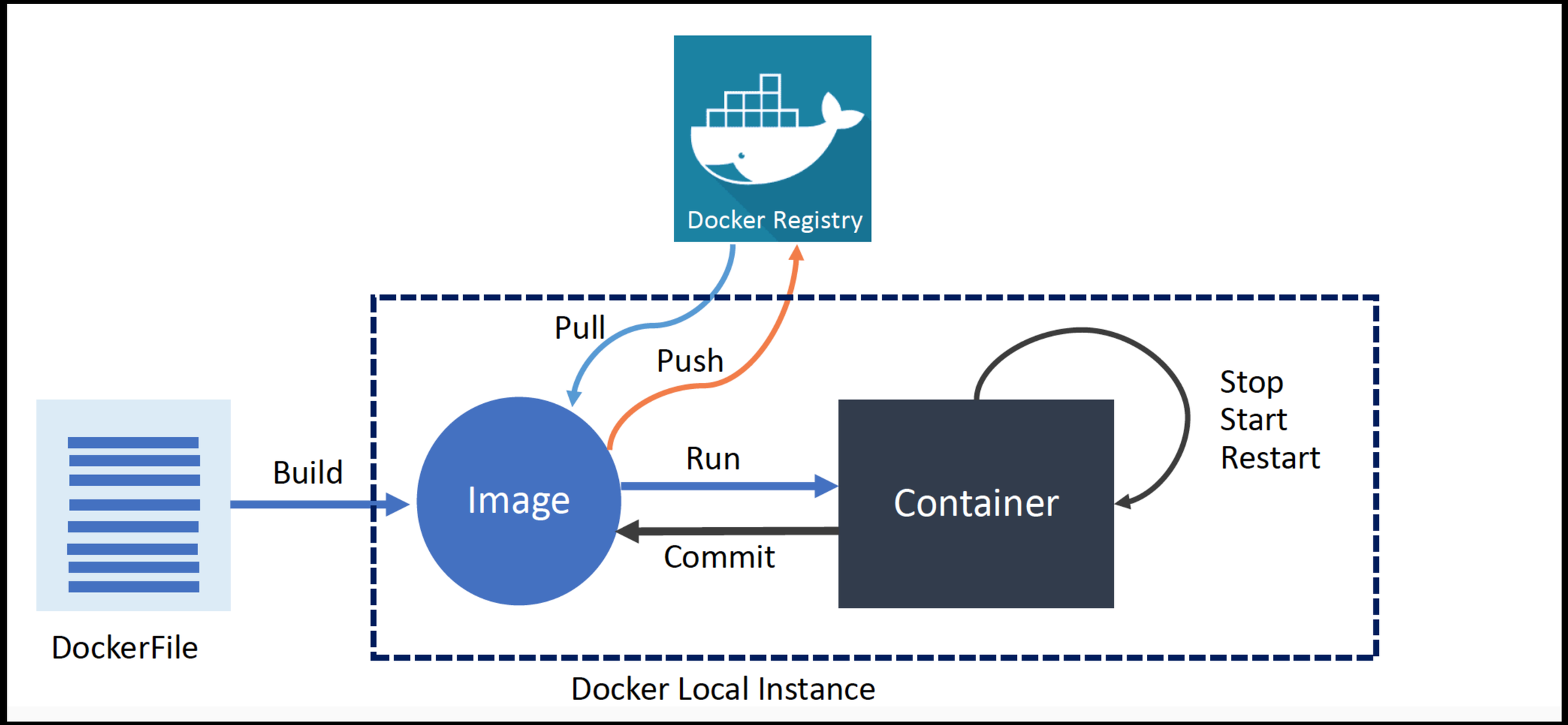
Contenedor Docker



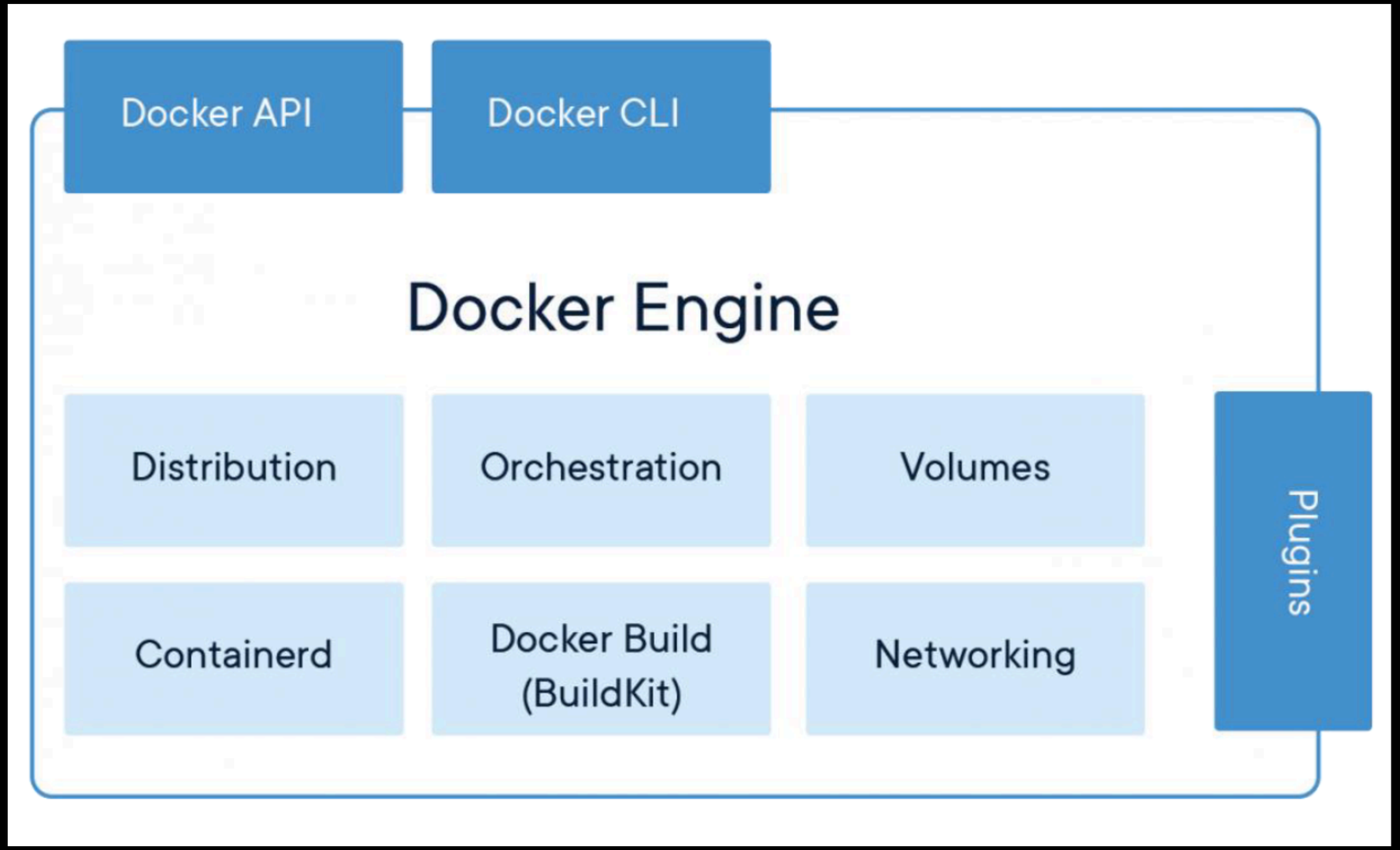
Docker File



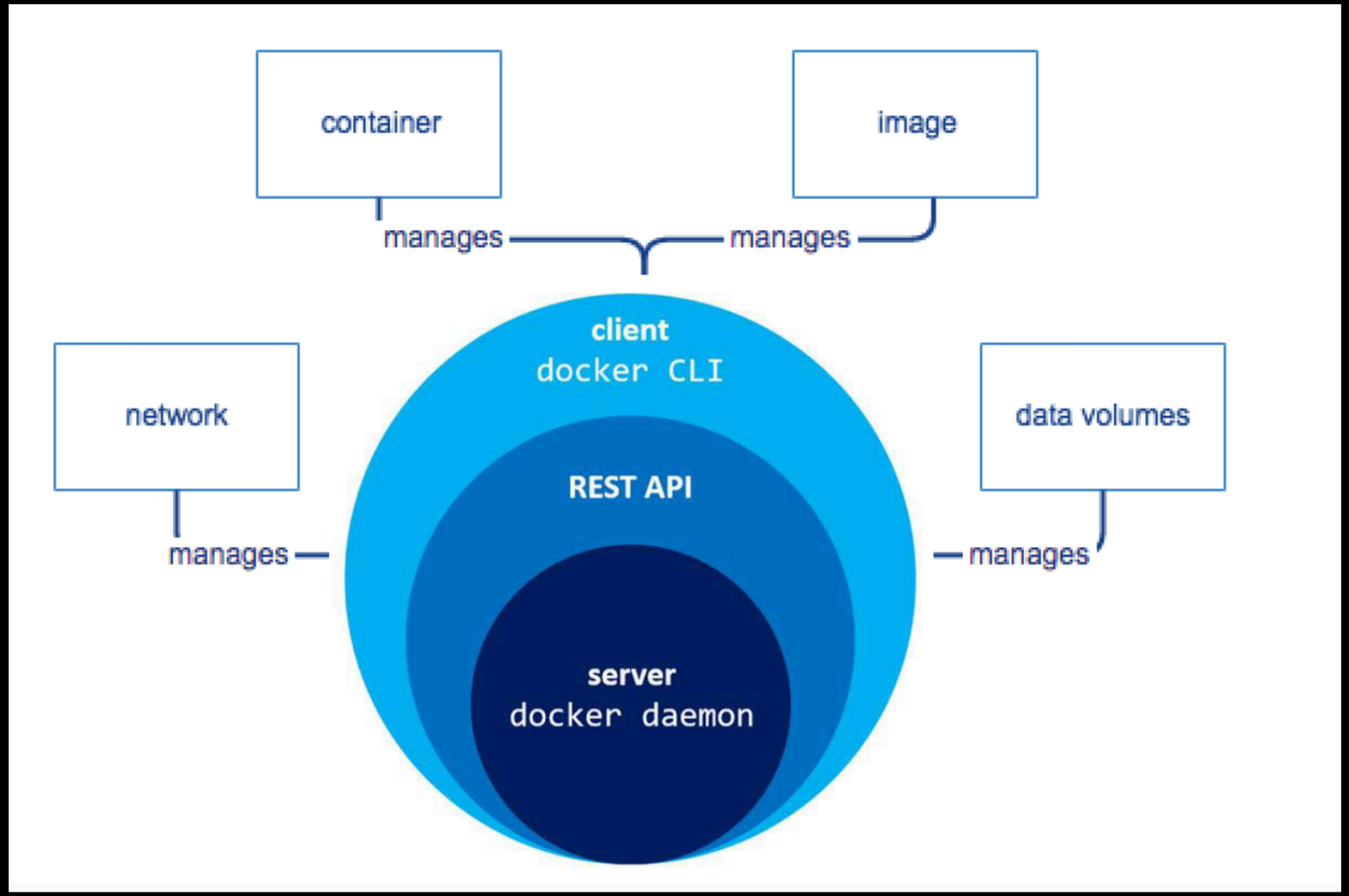
Ciclo de vida de un contenedor Docker



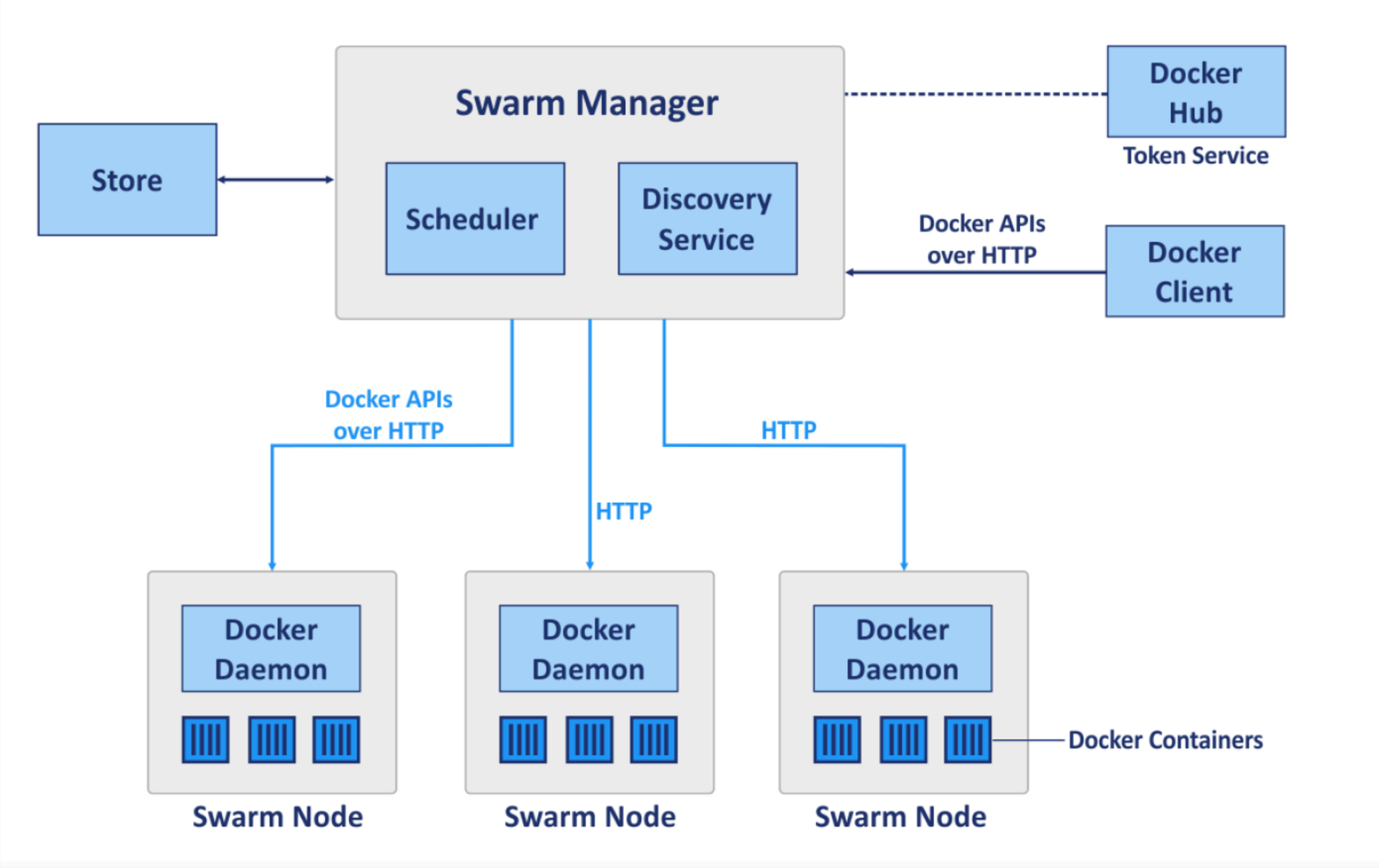
Docker Engine



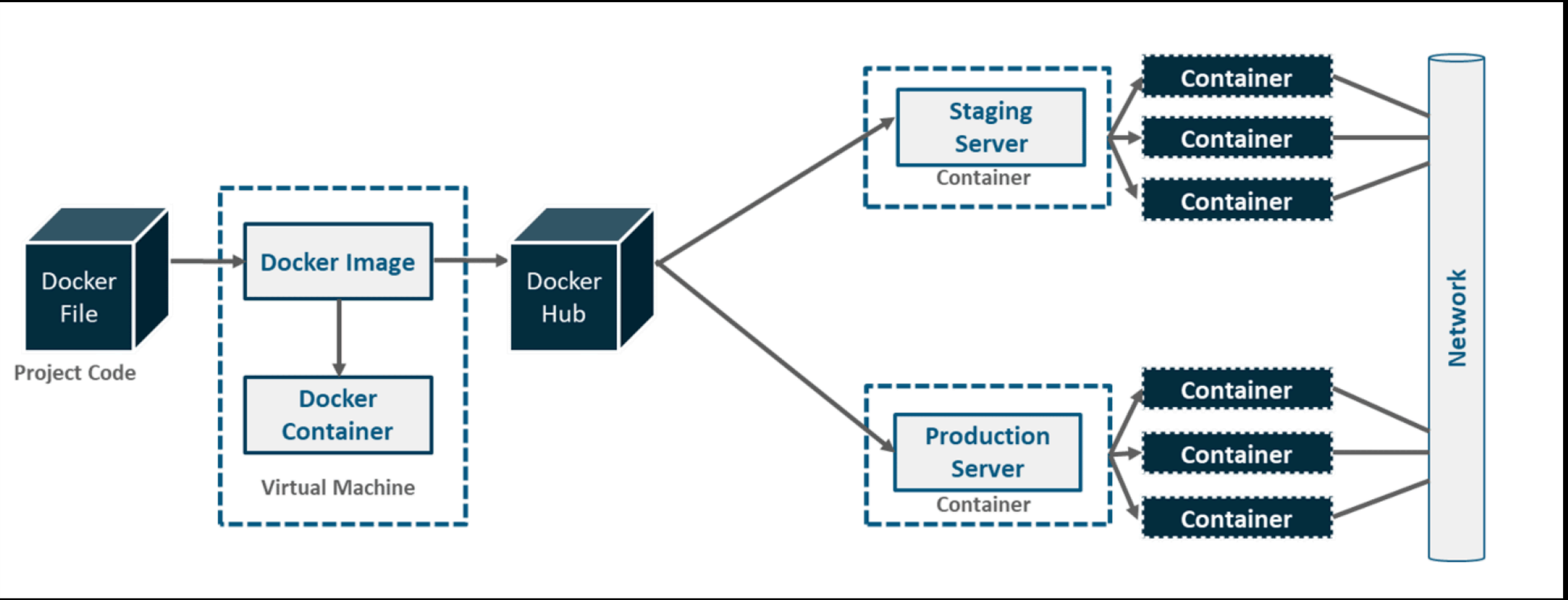
Docker CLI


container
image
manages
manages
client
docker CLI
network
data volumes
REST API
manages
manages
server
docker daemon

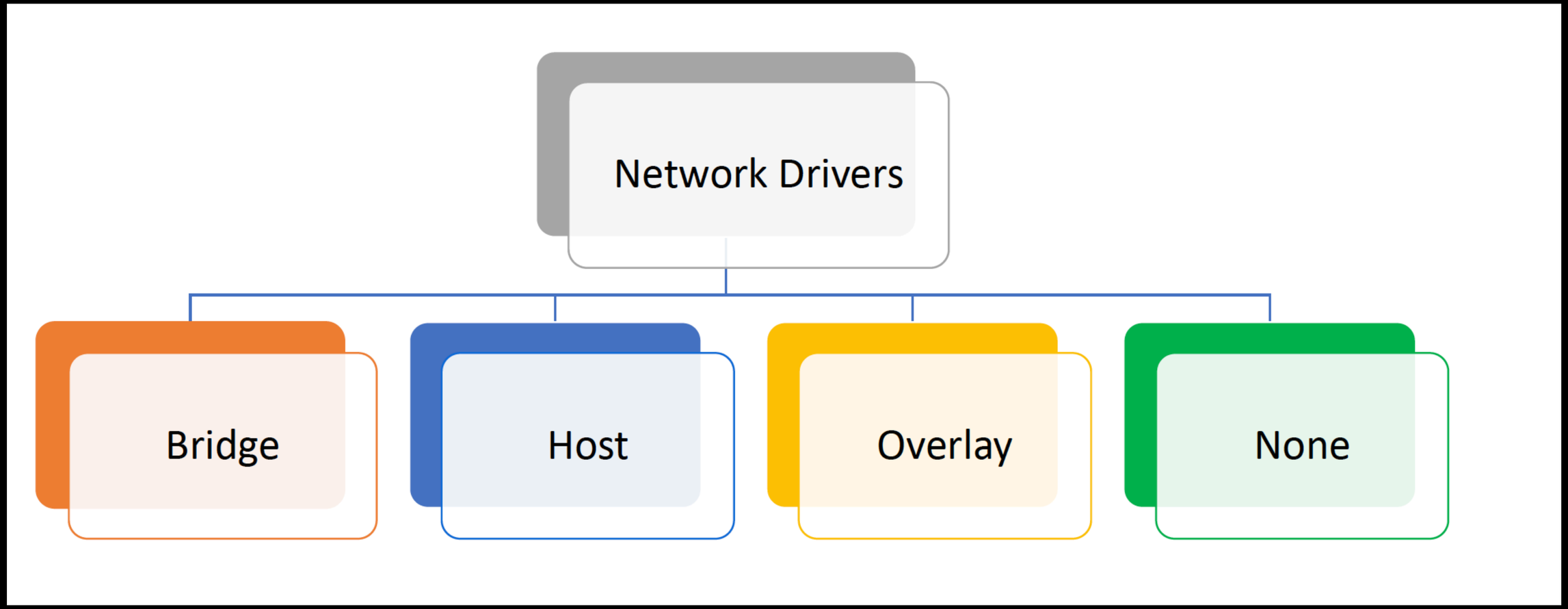
Docker Swarm



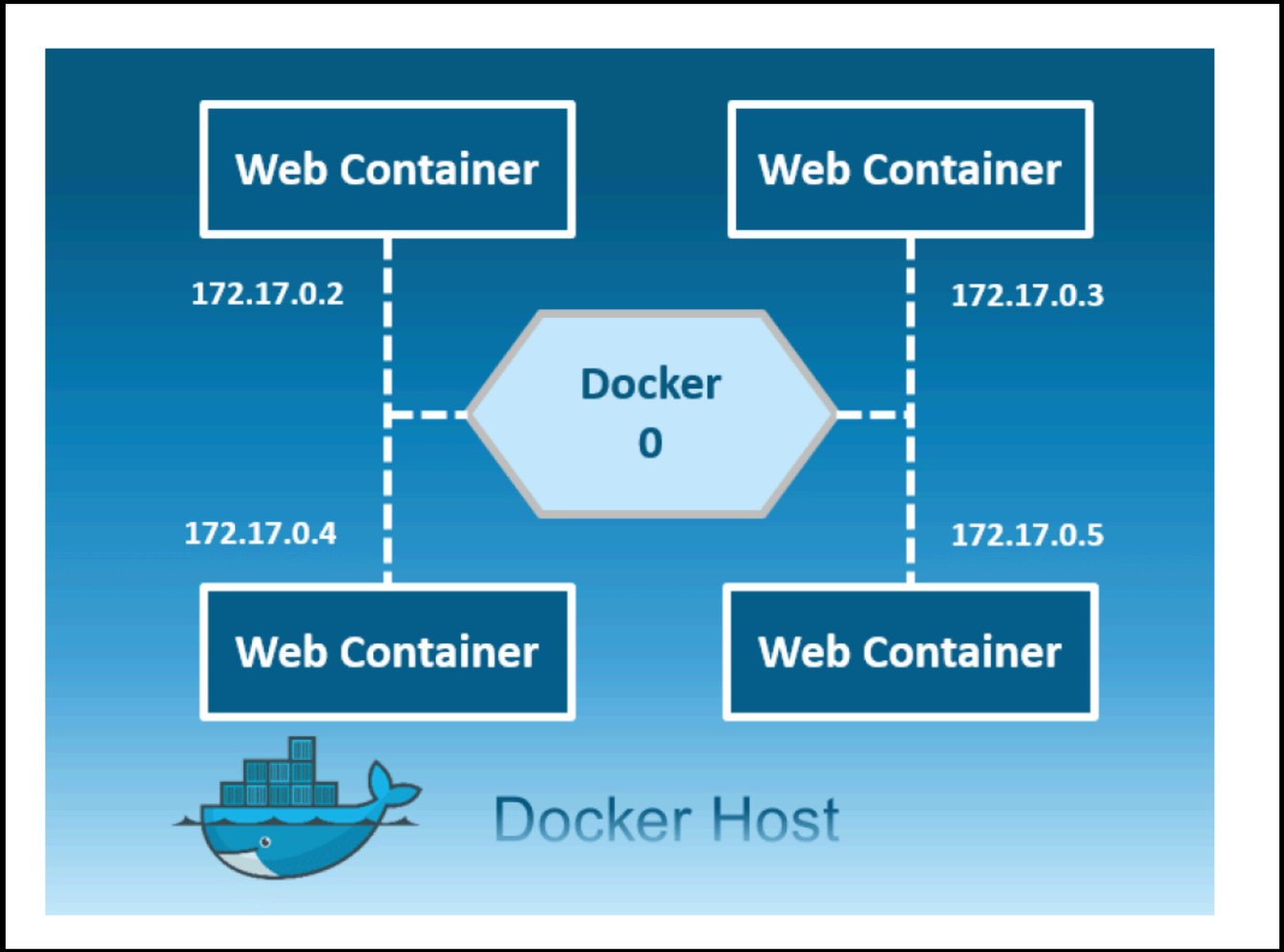
Docker Networking



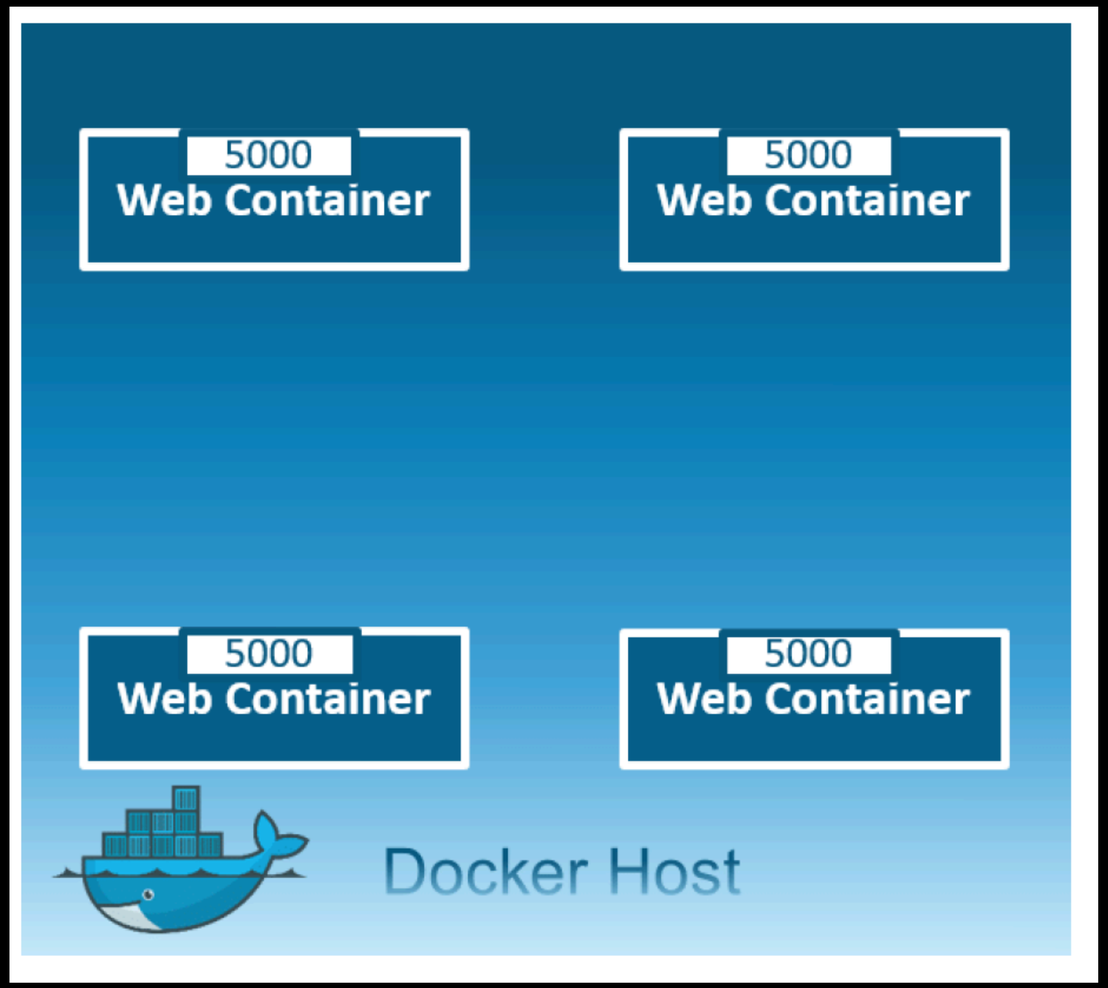
Network Drivers



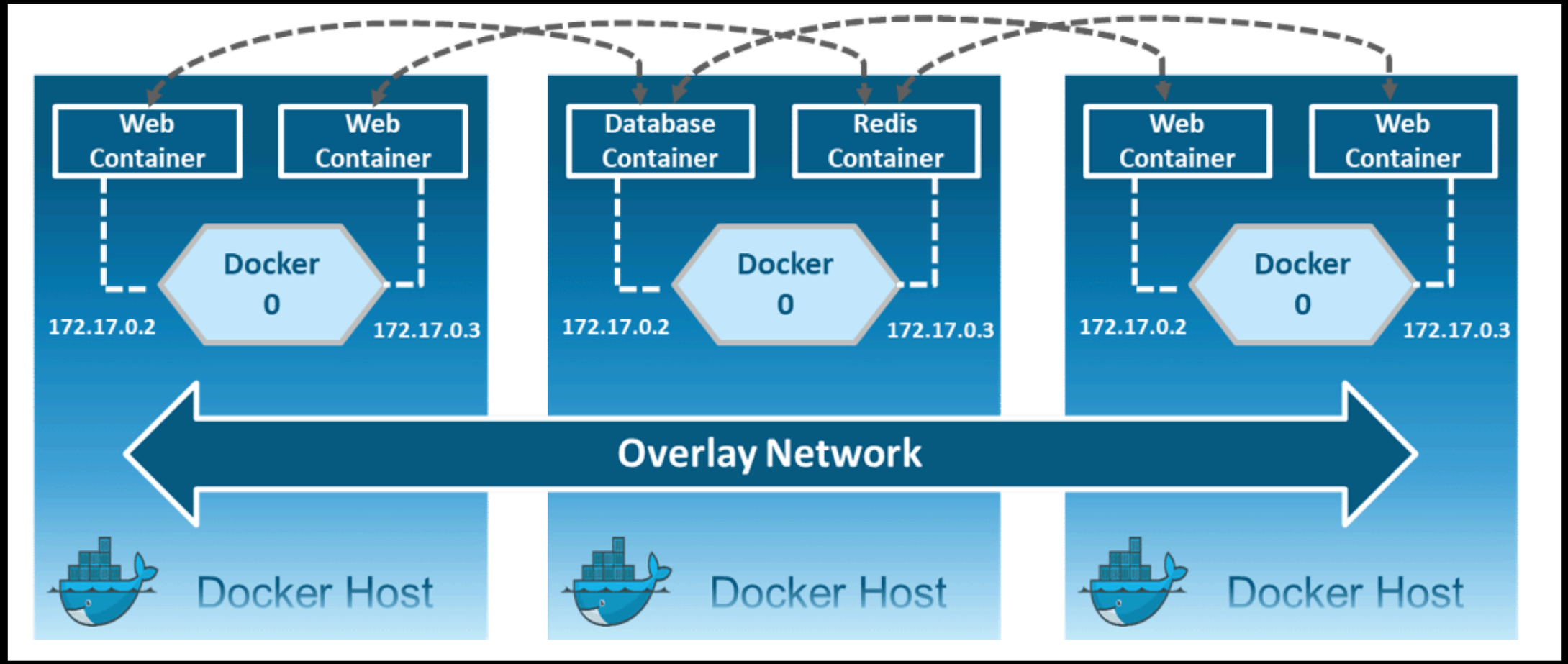
Bridge


Web Container
Web Container
172.17.0.2
172.17.0.3
Docker
0
172.17.0.4
172.17.0.5
Web Container
Web Container
Docker Host

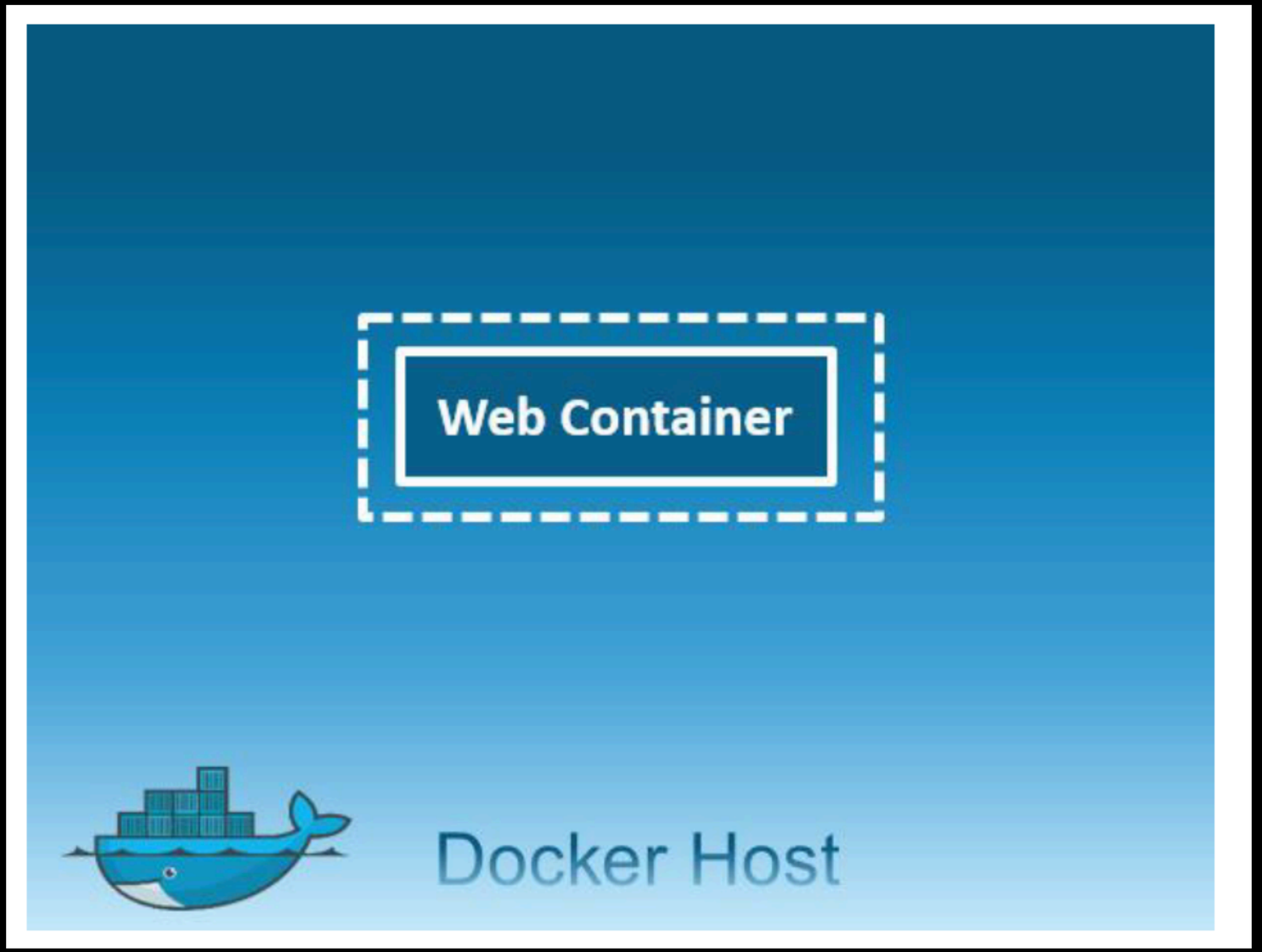
Bridge


5000
Web Container
5000
Web Container
5000
Web Container
5000
Web Container
Docker Host

Bridge


Web Container
Web Container
Database Container
Redis Container
Web Container
Web Container
Docker 0
Docker 0
Docker 0
172.17.0.2
172.17.0.3
172.17.0.2
172.17.0.3
172.17.0.2
172.17.0.3
Overlay Network
Docker Host
Docker Host
Docker Host

None



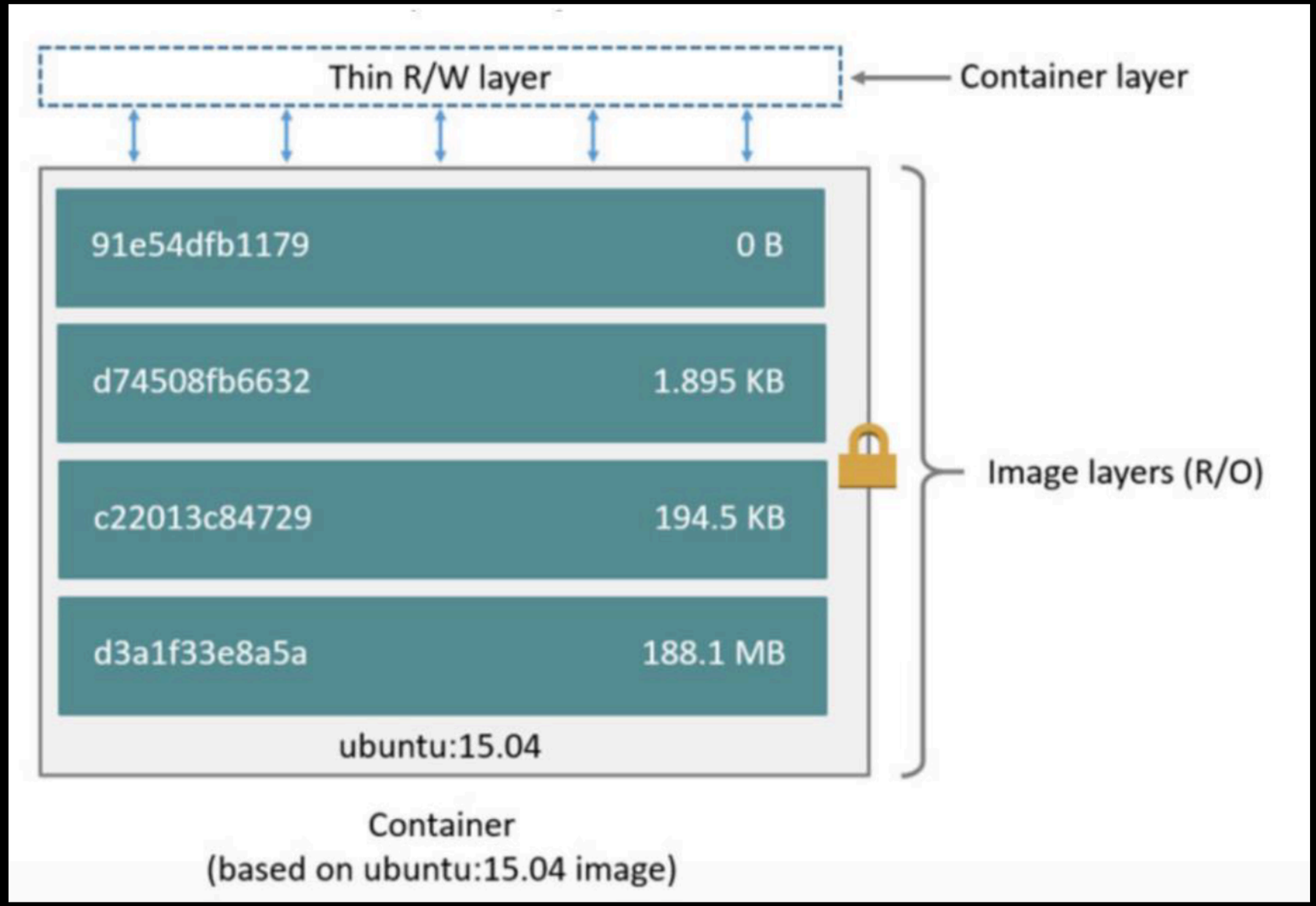
Docker Service

- Es uno o mas contenedores con la misma configuración ejecutándose bajo la modalidad docker swarm
- Con docker service tu administras un grupo de contenedores desde la misma imagen
- Tu puedes escalar ellos (iniciar múltiples contenedores)
- Docker service es útil para un microservicio al interior del contexto de una gran aplicación

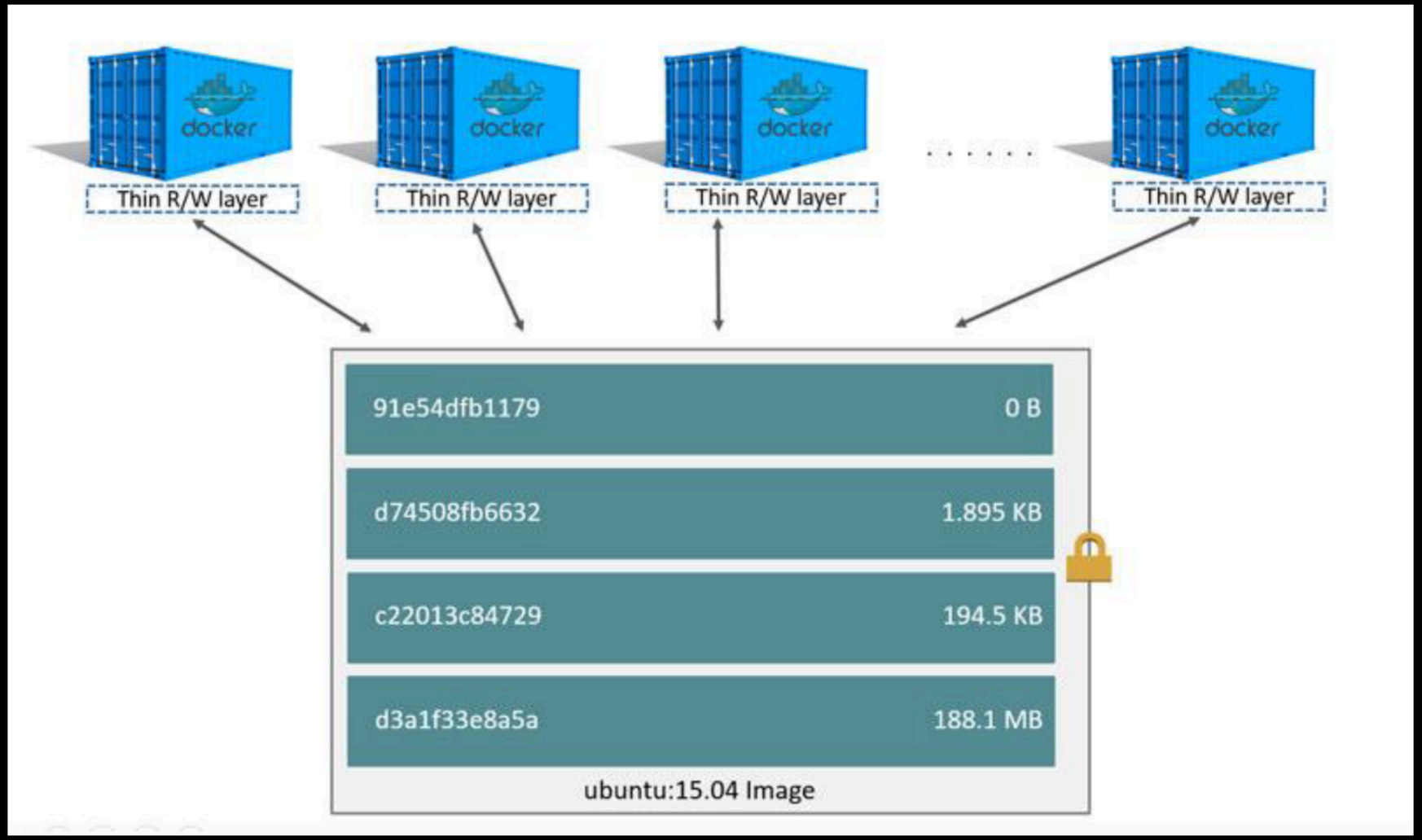
Docker Compose

- **build** Build or rebuild services
- **help** Get help on a command
- **kill** Kill containers
- **logs** View output from containers
- **port** Print the public port for a port binding
- **ps** List containers
- **pull** Pulls service images
- **rm** Remove stopped containers
- **run** Run a one-off command
- **scale** Set number of containers for a service
- **start** Start services
- **stop** Stop services
- **restart** Restart services up Create and start containers

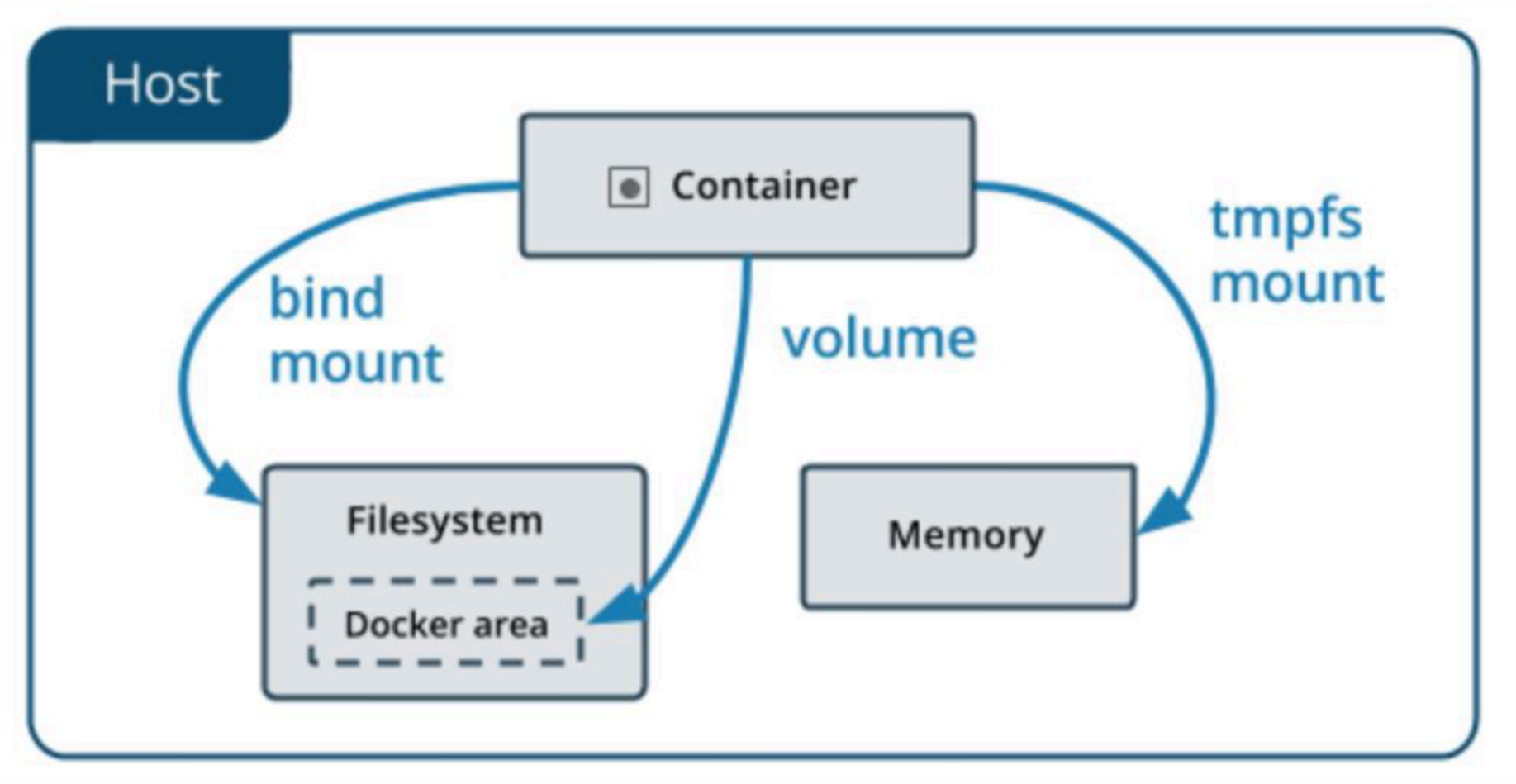
Docker Storage



Contenedores y Capas



Opciones para Docker Storage

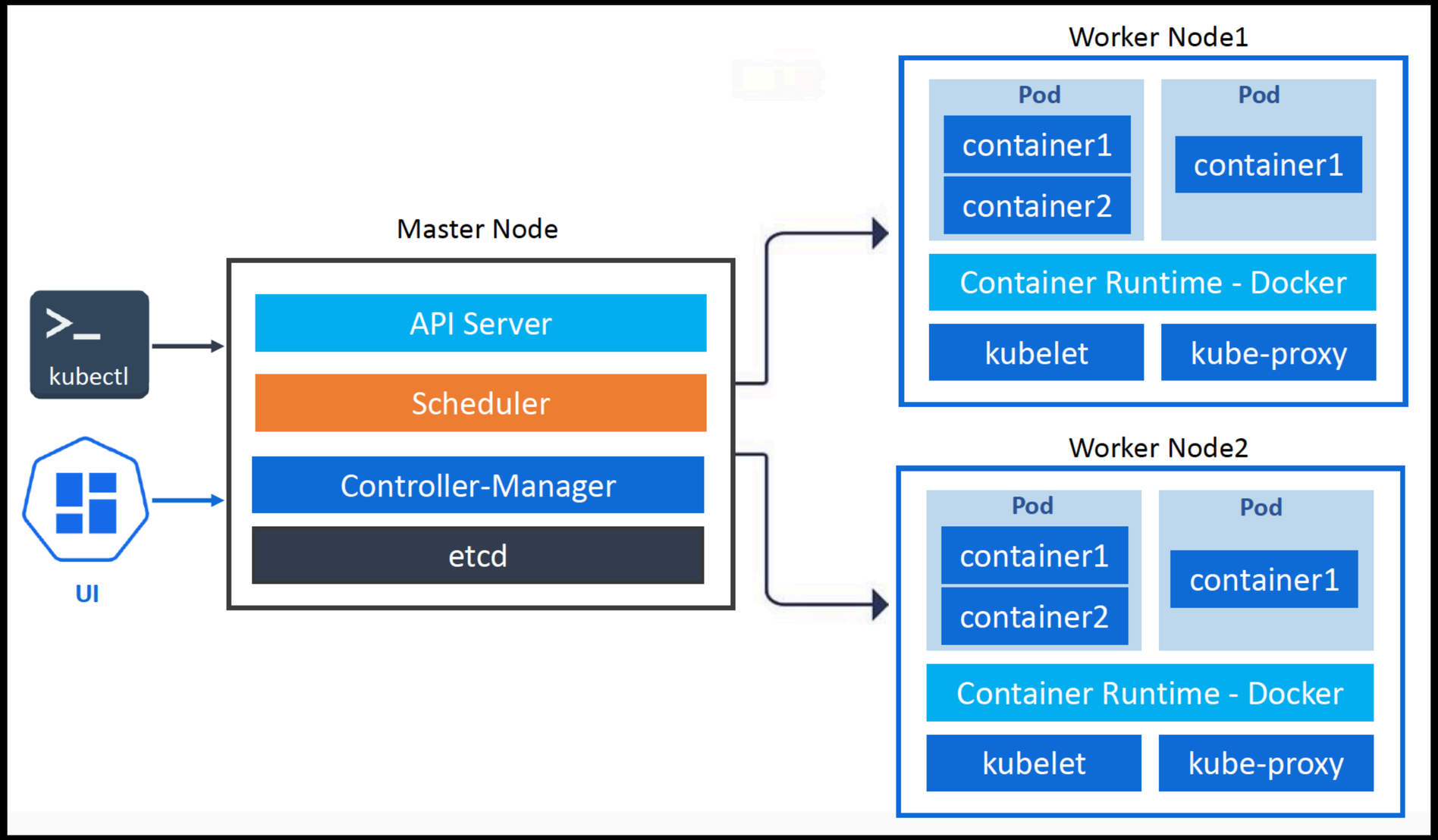


Kubernetes

Docker Swarm vs Kubernetes

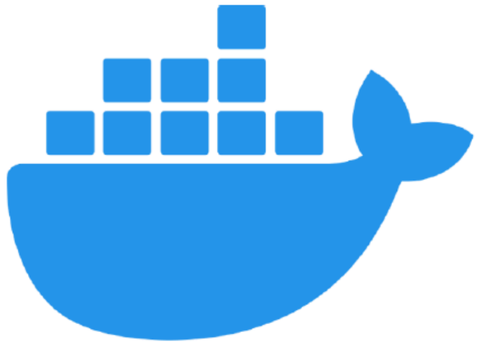
Parameters	Docker Swarm	Kubernetes
Scaling	No Autoscaling	Auto-scaling
Load balancing	Does auto load balancing	Manually configure your load balancing settings
Storage volume sharing	Shares storage volumes with any other container	Shares storage volumes between multiple containers inside the same Pod
Use of logining and monitoring tool	Use 3rd party tool like ELK	Provide an in-built tool for logging and monitoring.
Installation	Easy & fast	Complicated & time-consuming
GUI	GUI not available	GUI is available
Scalability	Scaling up is faster than K8S, but cluster strength not as robust	Scaling up is slow compared to Swarm, but guarantees stronger cluster state Load balancing requires manual service configuration
Load Balancing	Provides a built-in load balancing technique	Process scheduling to maintain services while updating
Updates & Rollbacks Data Volumes Logging & Monitoring	Progressive updates and service health monitoring.	Only shared with containers in same Pod Inbuilt logging & monitoring tools.

Arquitectura Kubernetes



Configurar Kubernetes

Minikube



Docker Desktop

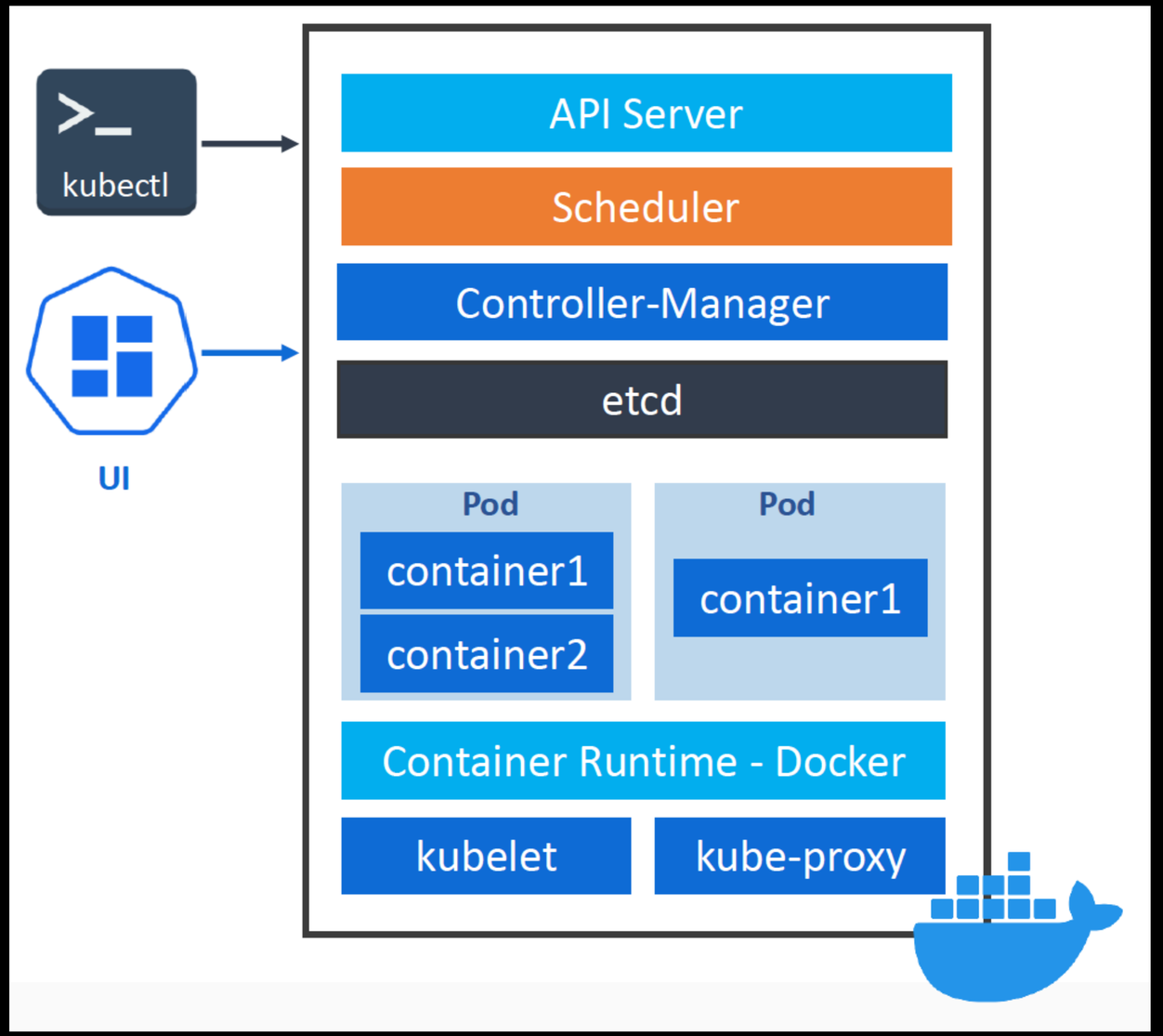
Kubeadm

Cloud

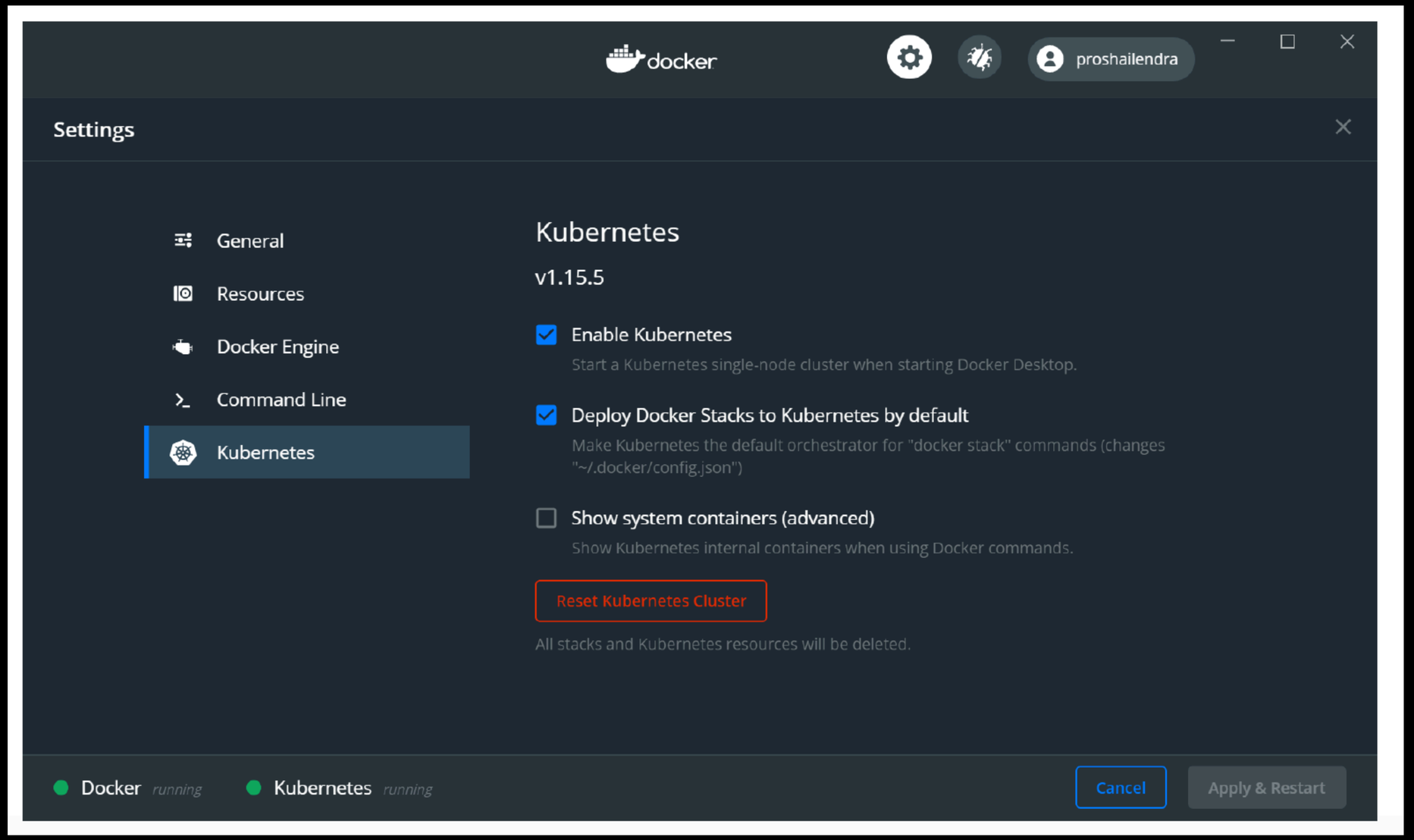
Single Node Cluster

Multi Node Cluster

Simple Nodo Cluster



Kubernetes usando Docker Desktop



XML vs JSON vs YAML

```xml
<Servers>
    <Server>
        <name>server1</name>
        <location>india</location>
        <status>active</status>
    </Server>
</Servers>
```

```json
{
    "Servers": [
        "Server": {
            "name": "server1",
            "location": "india",
            "status": "active"
        }
    ]
}
```

```yaml
Servers:
  - name: server1
    location: india
    status: active
```

YAML en acción

```yaml
name: server1
location: india
status: active
```

Key/Value

```yaml
Servers:
  - server1
  - server2
  - server3
```

Array/List: Ordered

```yaml
Servers:
    name: server1
    status: active
    location: india
```

Dictionary/Map: Unordered

```yaml
Servers:
  - server1:
      location: india
      status: active
  - server2:
      location: usa
      status: active
```

Array/Dictionary/Key-value

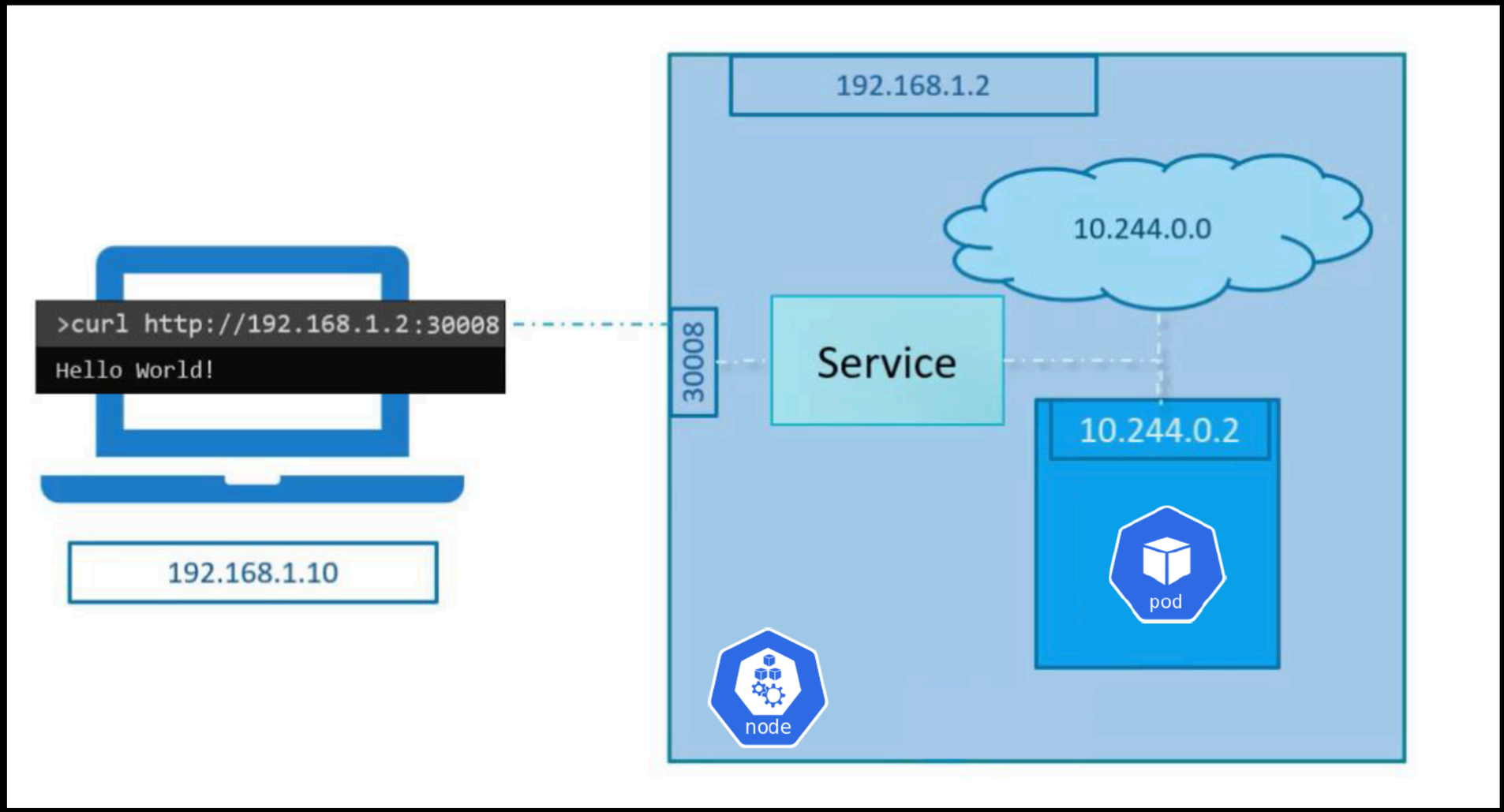
YAML en Kubernetes

YAML in Kubernetes

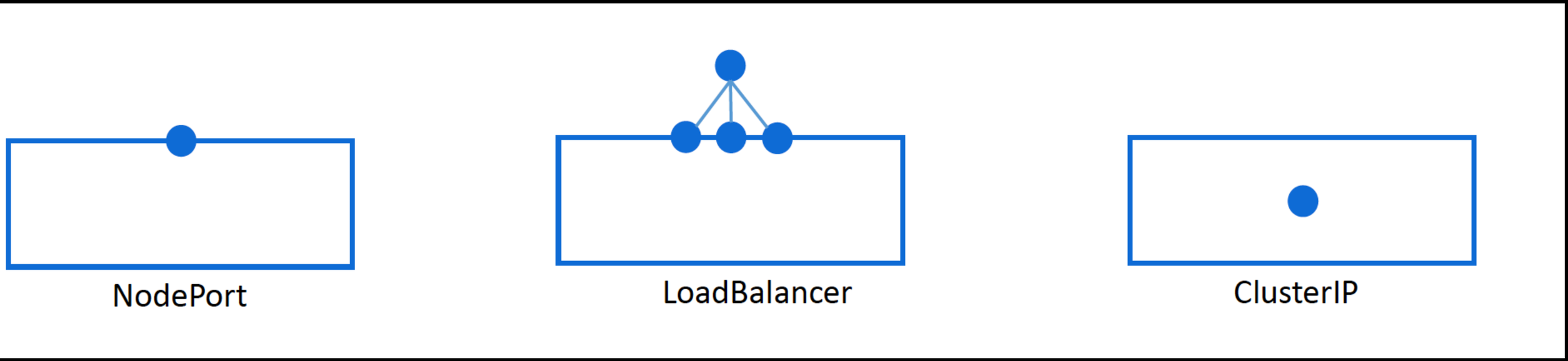
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    name: myapp
spec:
  containers:
    - name: nginx-container
      image: nginx
```

Kind	Version
POD	v1
Service	v1
ReplicaSet	apps/v1
Deployment	apps/v1

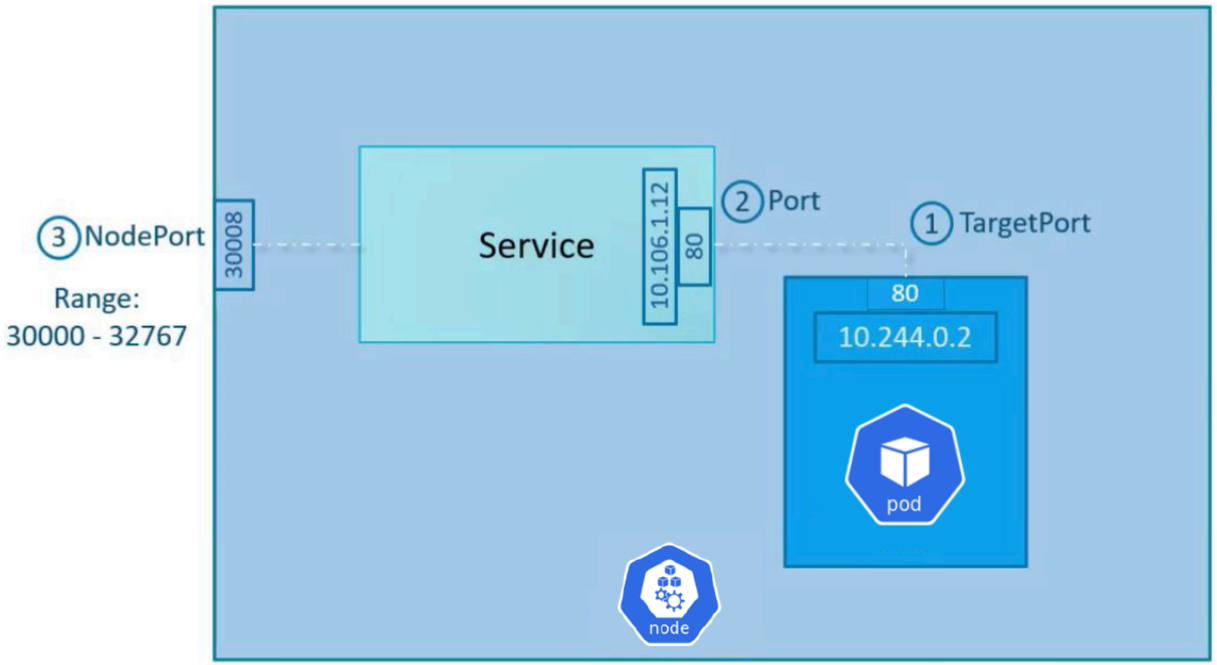
Service


>curl http://192.168.1.2:30008
Hello World!
192.168.1.10
192.168.1.2
10.244.0.0
30008
Service
10.244.0.2
pod
node

Tipos de Servicio



Servicio NodePort

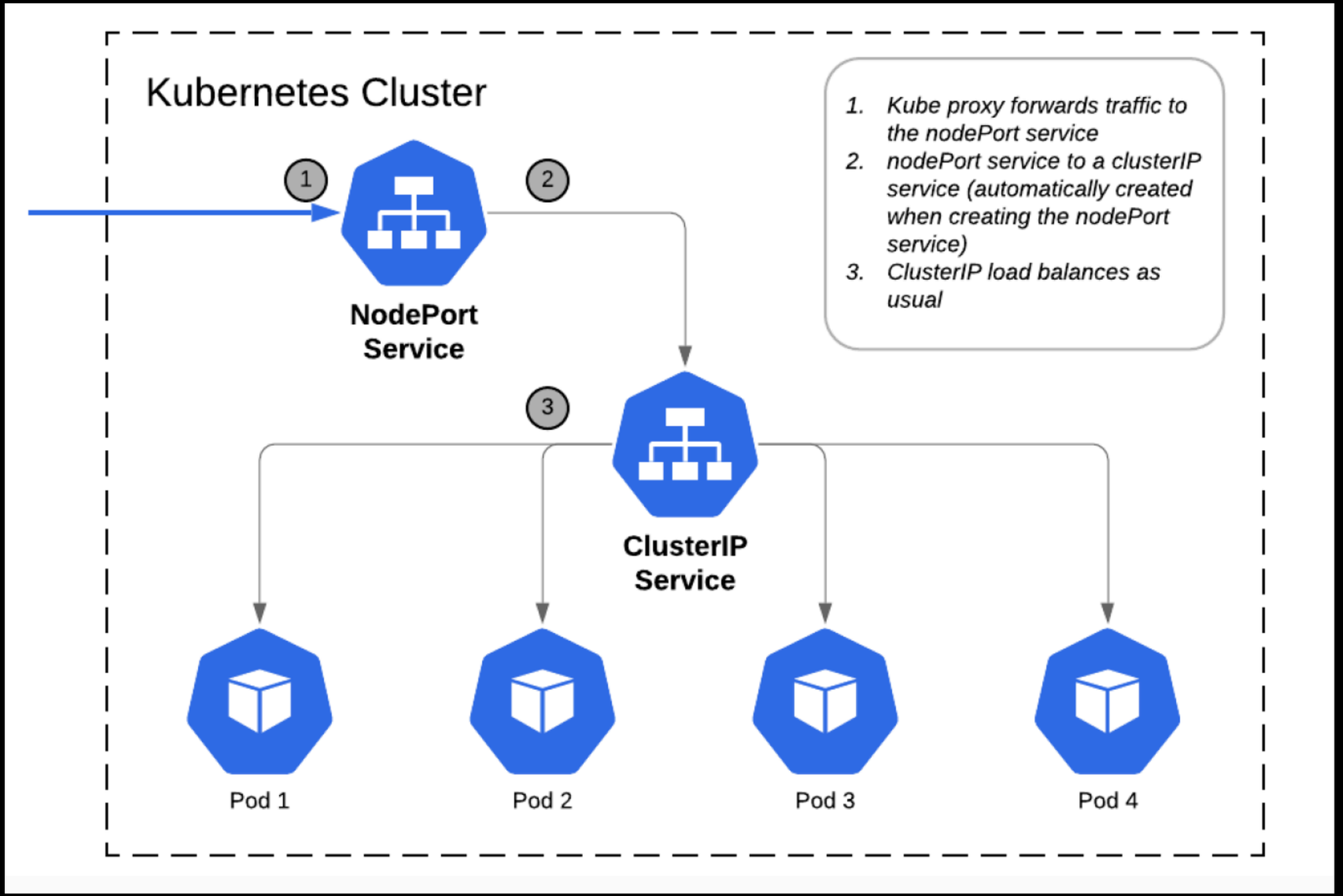


service-definition.yml

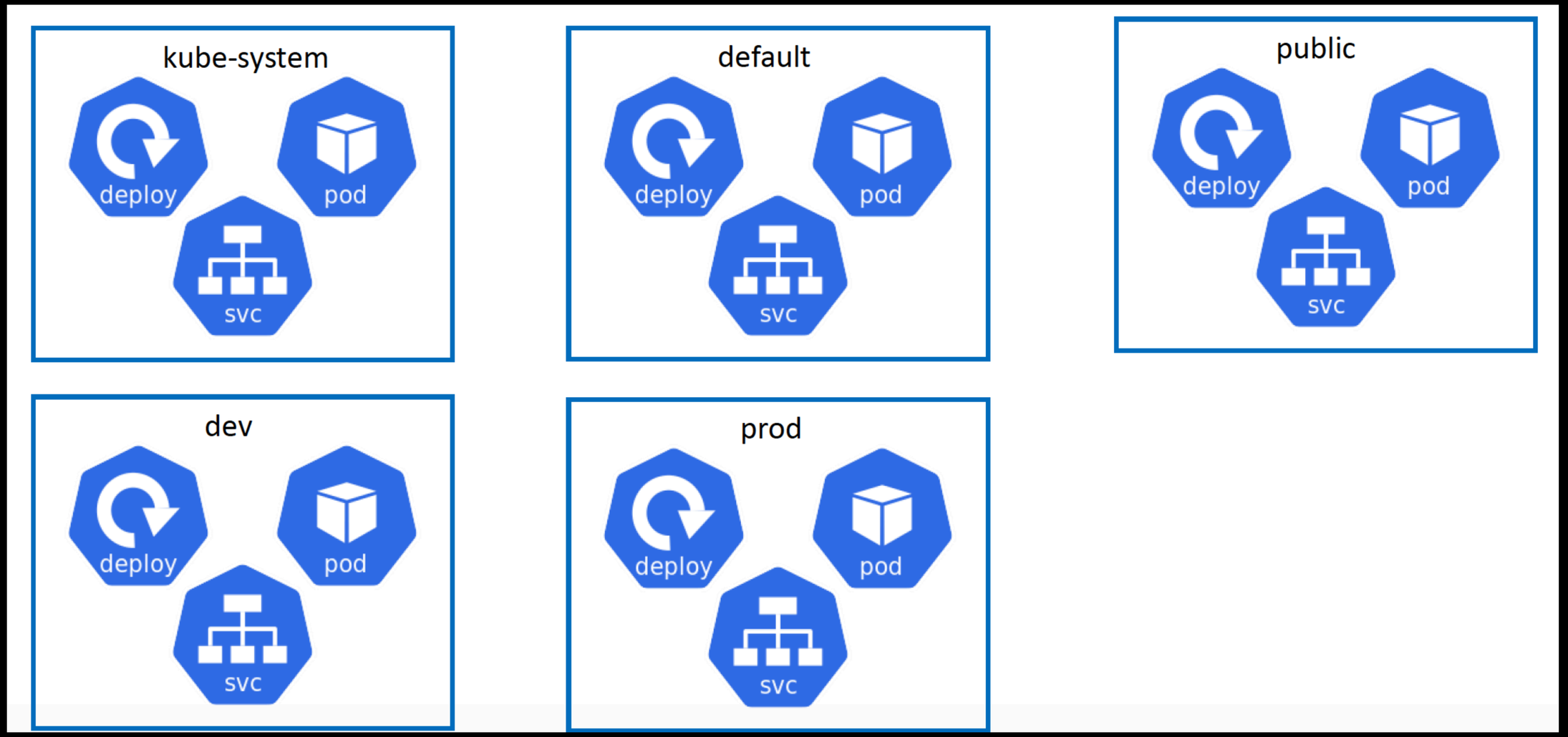
```yaml
apiVersion: v1
kind: Service
metadata:
    name: myapp-service

spec:
    type: NodePort
    ports:
     - targetPort: 80
     *port: 80
       nodePort: 30008
```

Servicio ClusterIP



Namespace Isolation



Comandos Namespace

namespace-dev.yml

```yaml
apiVersion: v1
kind: Namespace
metadata:
    name: dev
```

```
> kubectl create -f namespace-dev.yml
namespace/dev created
```

```
> kubectl create namespace dev
namespace/dev created
```

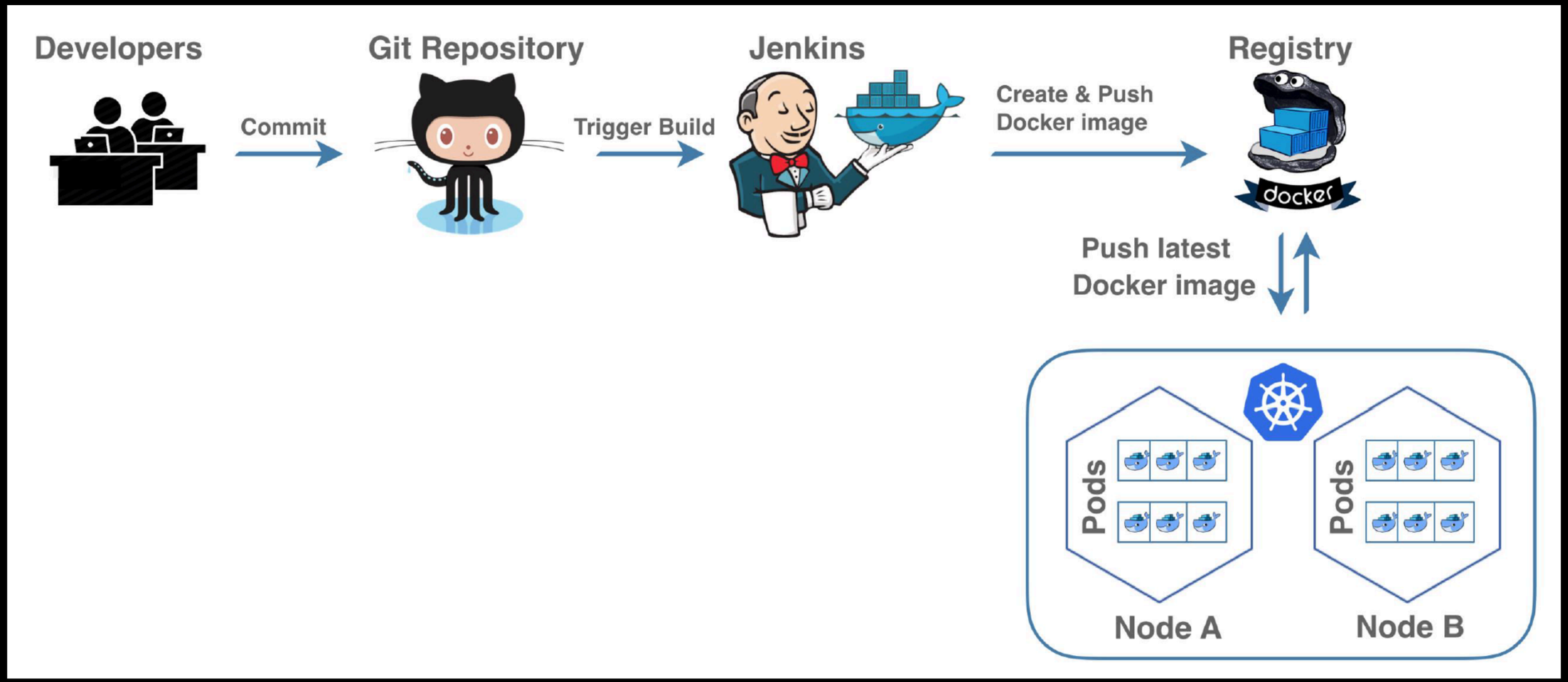
```
> kubectl config set-context $(kubectl config current-context) --namespace=prod
```

```
> kubectl get pods --namespace=dev
```

```
> kubectl get pods --namespace=default
```

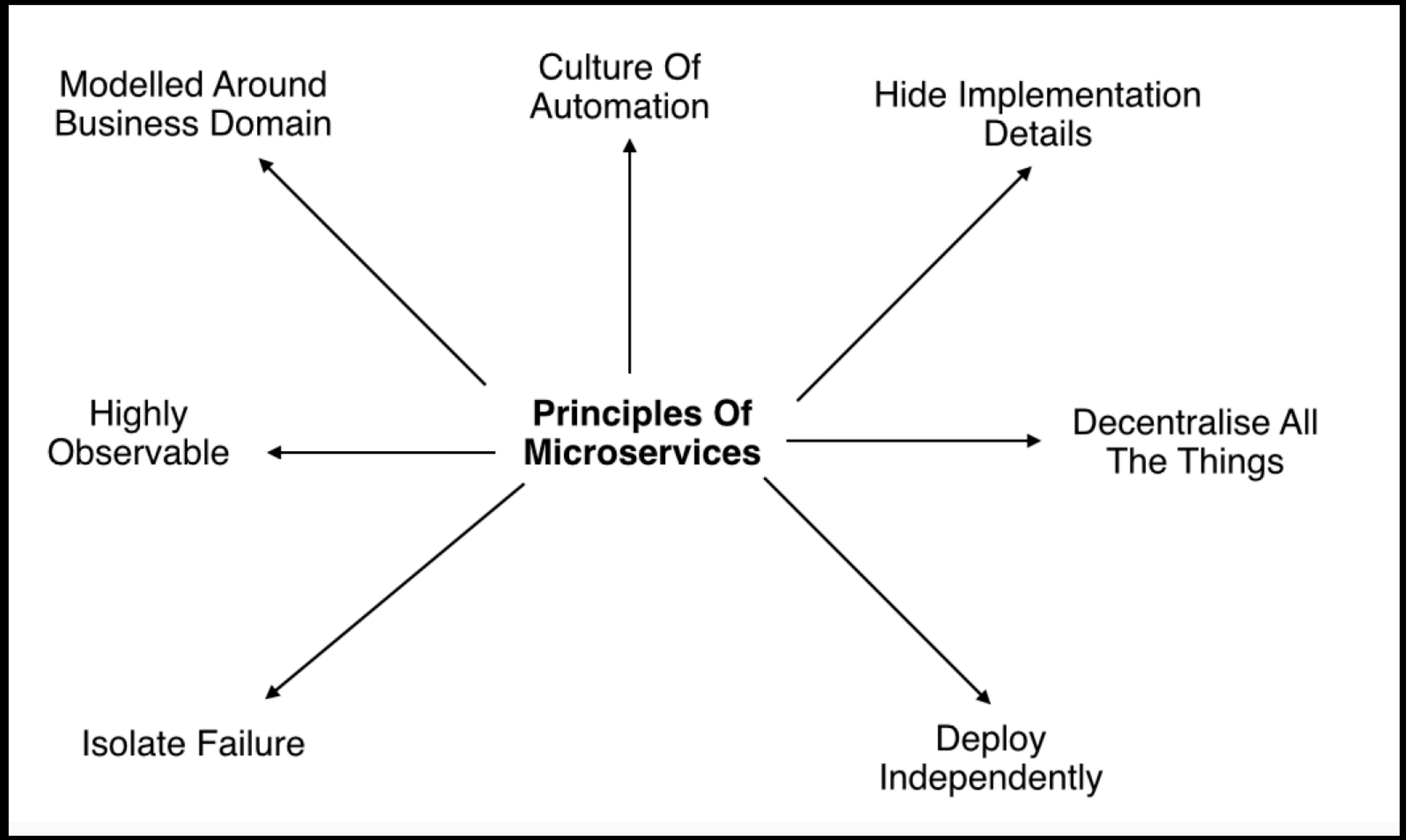
```
> kubectl get pods --all-namespaces
```

CI/CD Pipeline

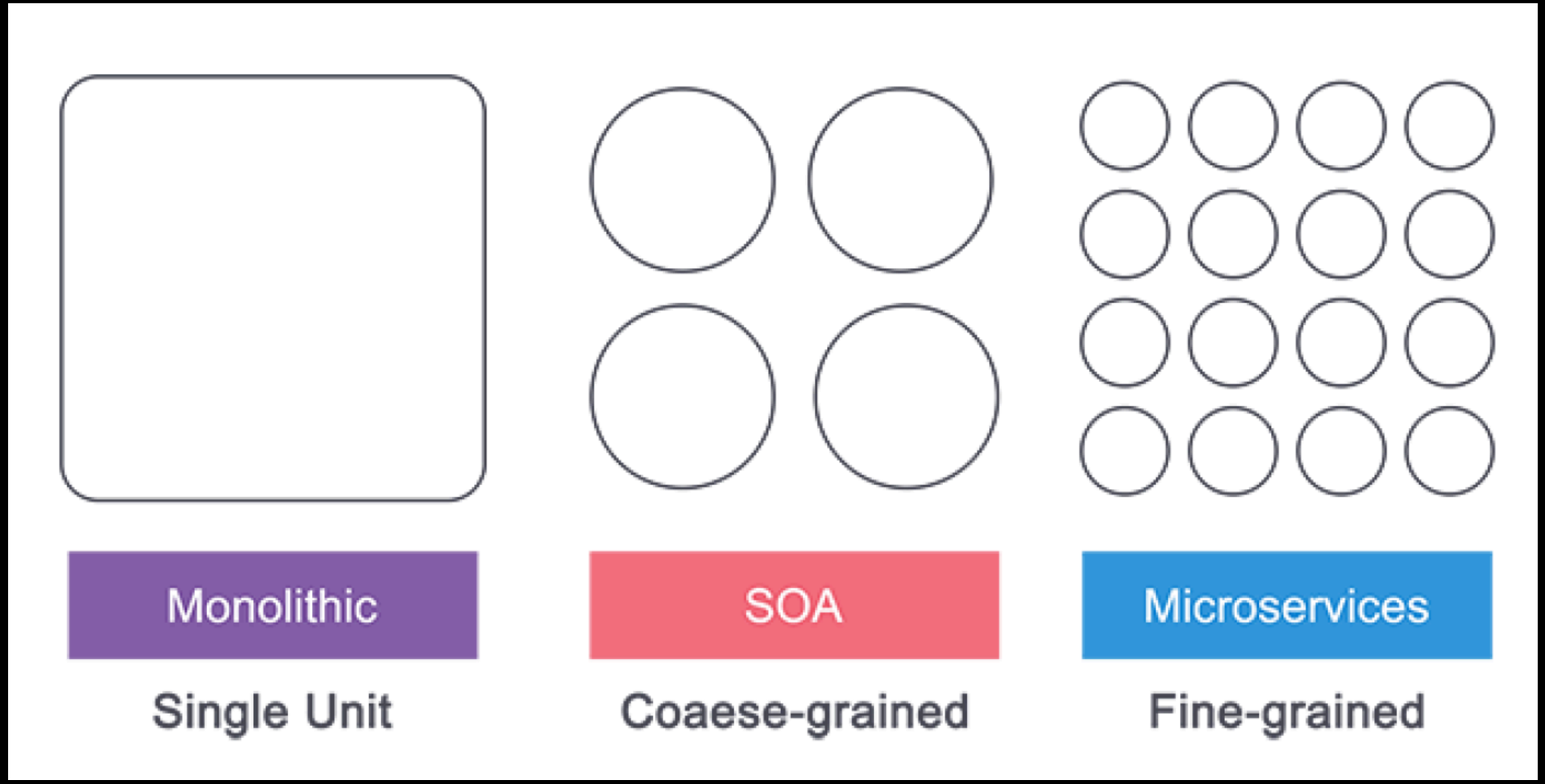


Intro a Microservicios

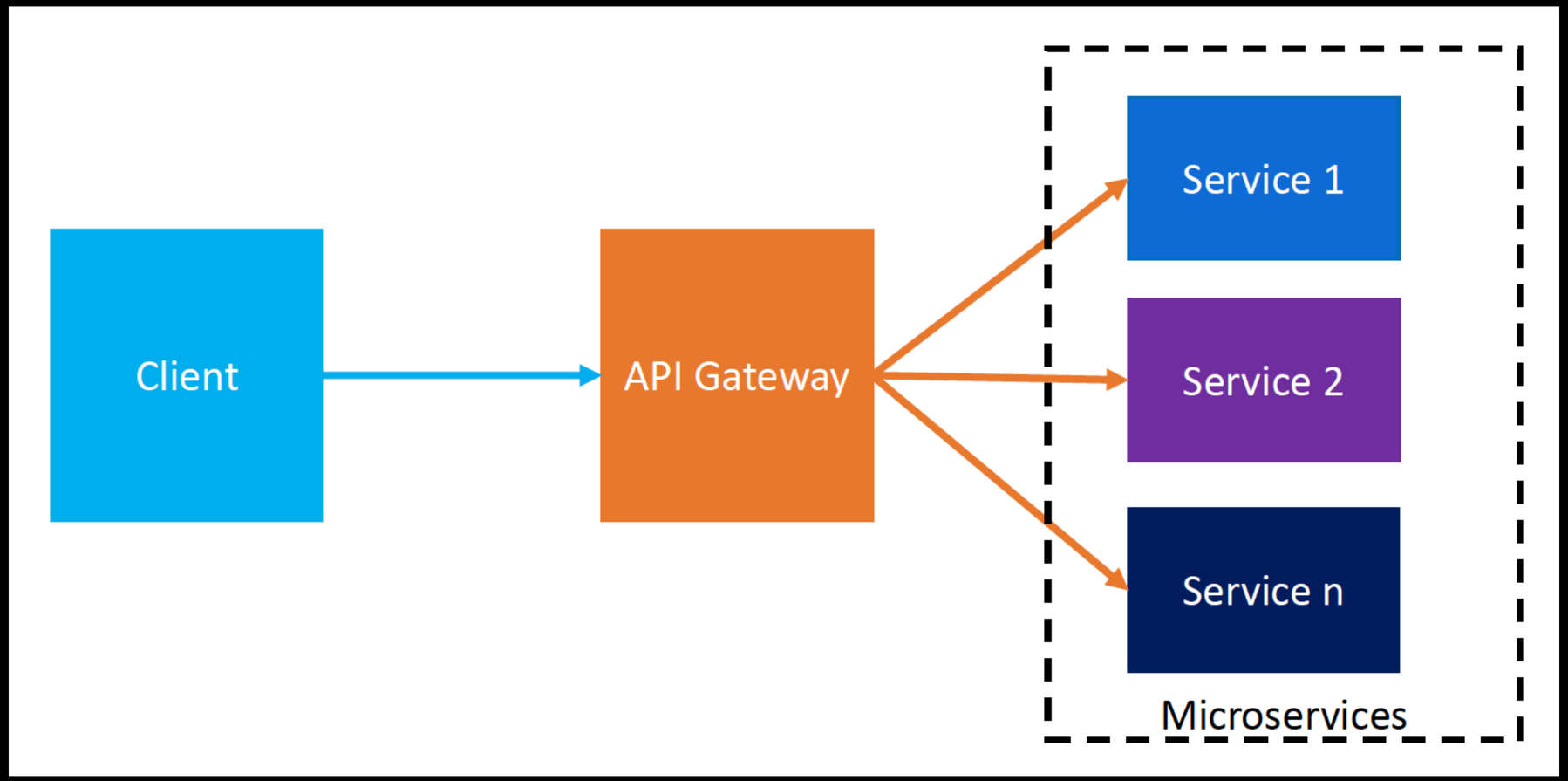
Principios de Micro servicios



Monolítico vs SOA vs Micro servicios



Monolítico vs SOA vs Micro servicios



Tecnologías para Micro servicios

Frameworks de Desarrollo

Herramientas de Desarrollo

Visual Studio

www.visualstudio.microsoft.com/

VS Code

www.code.visualstudio.com/

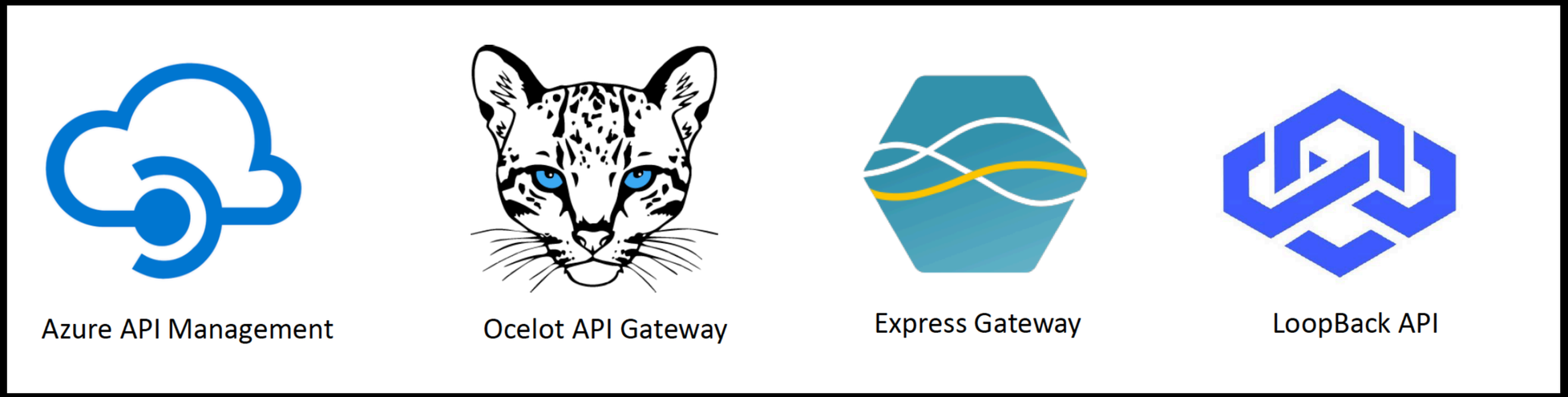
Docker

www.code.docker.com/

API Management : API Gateways

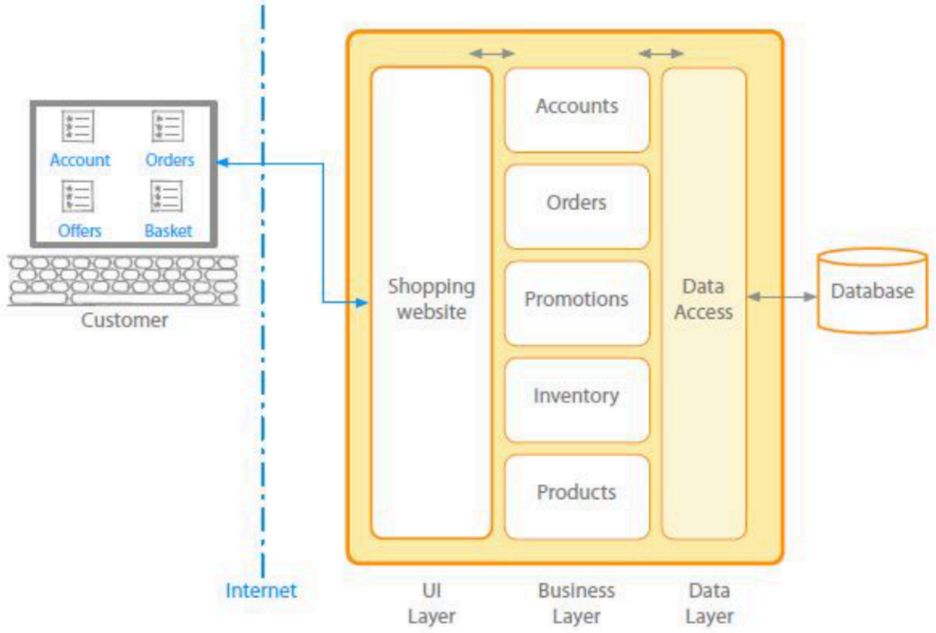
- Ruteo
- Composición de API
- Caching
- Logging
- Seguridad
- Limite de velocidad

Micro servicios - API Gateways

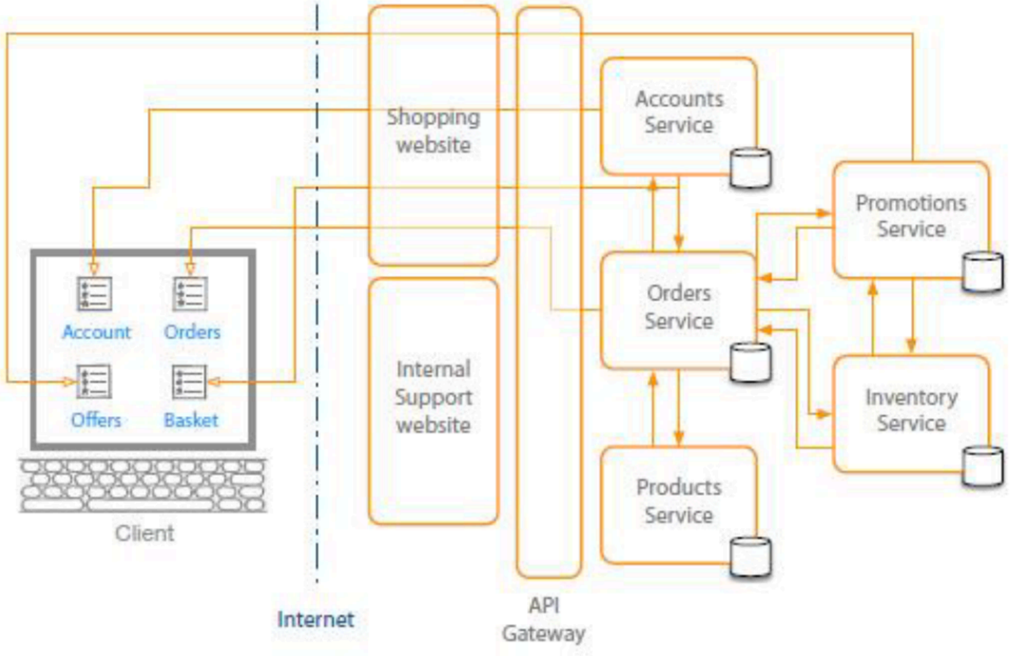


Patrones de Diseño

Migrando de monolito a micro servicios



Monolithic App



Microservices App

Patrones de Diseño para micro servicios

Decomposition Patterns

- Decompose by Business Capability
- Decompose by Subdomain
- Strangler Pattern

Integration Patterns

- API Gateway Pattern
- Aggregator Pattern
- Client-Side UI Composition Pattern

Database Patterns

- Database per Service
- Shared Database per Service
- CQRS Pattern
- Saga Pattern

Deployment Patterns

- Multiple service instances per host
- Service instance per host
- Service instance per VM
- Service instance per Container
- Serverless deployment

Observability Patterns

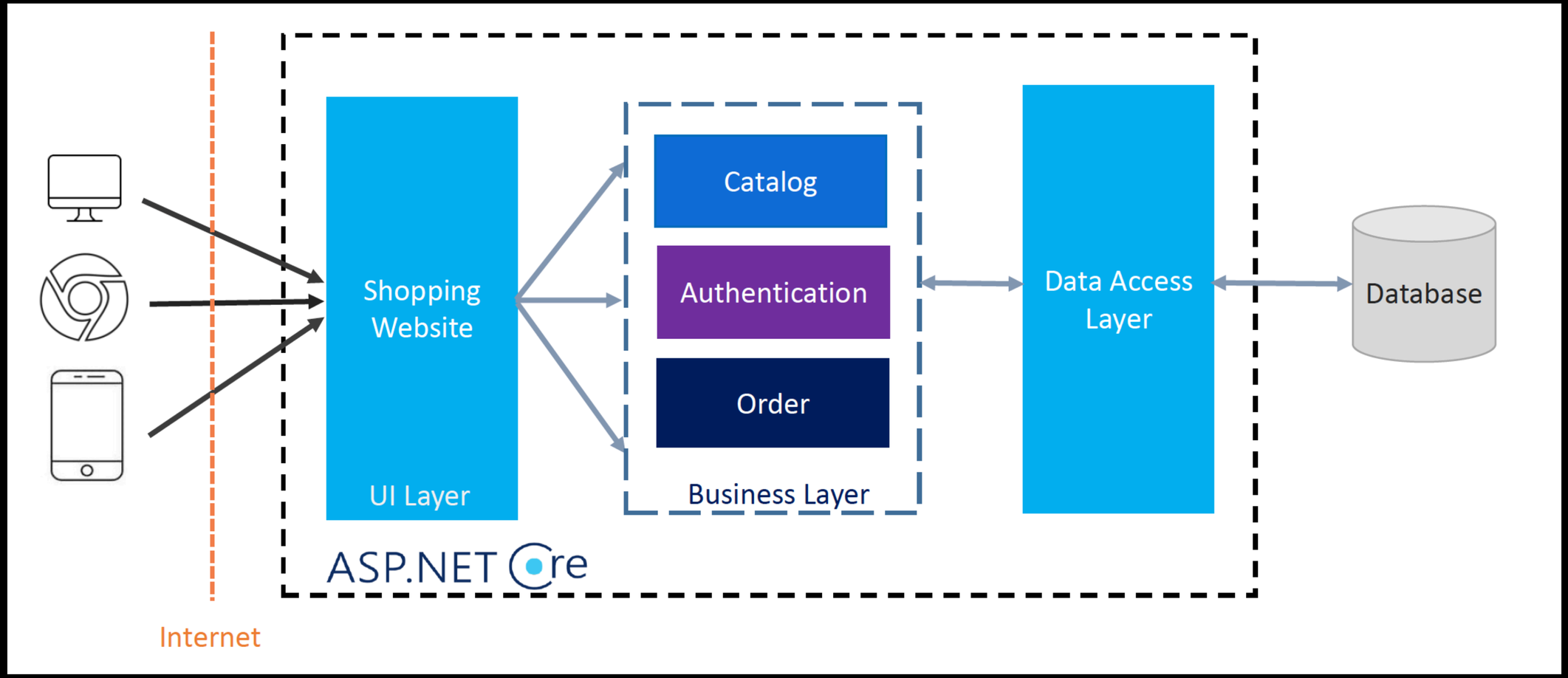
- Log Aggregation
- Performance Metrics
- Distributed Tracing
- Health Check

Cross-Cutting Concern Patterns

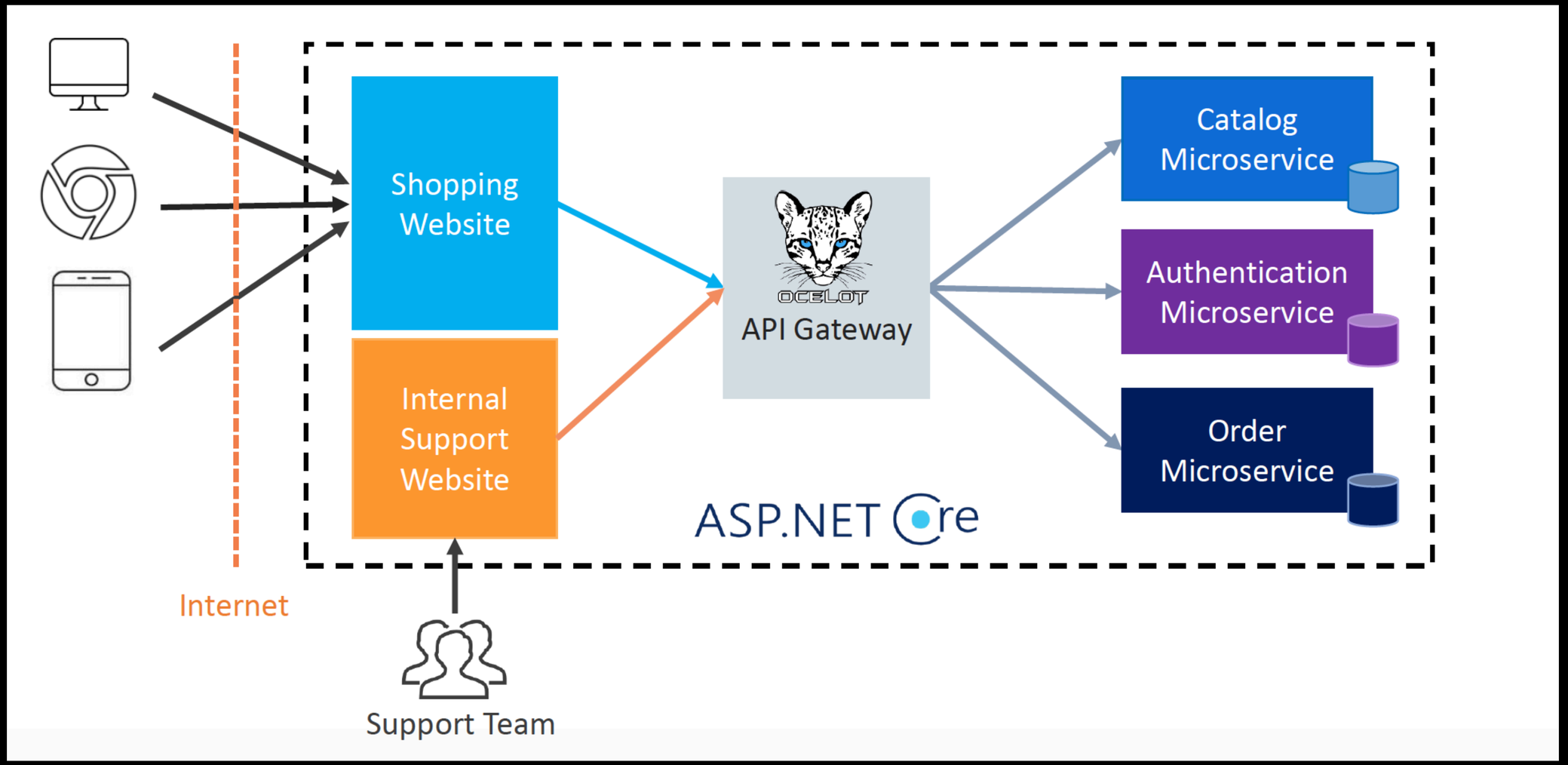
- Externalized Configuration
- Service Discovery Pattern
- Circuit Breaker Pattern

Construyendo micro servicios con ASP .NET Core

N-Layer APP



N-Layer App a Micro servicios



API Gateways de Micro servicios para .NET

Azure API Management

Ocelot API Gateway

Necesidad de un API Gateway

- Ruteo del tráfico
- Endpoint expuesto unificado
- Composición de API
- Caching
- Logging
- Authentication
- Authorization
- Balanceo de Carga
- Service Discovery

Demo de Microservicios

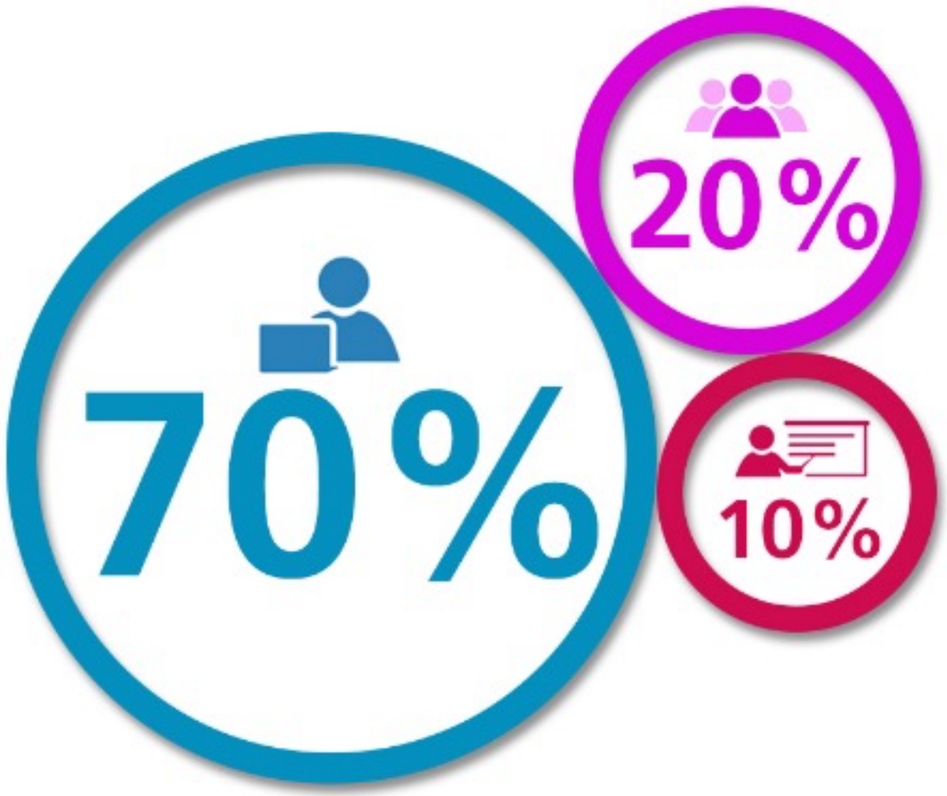
Ocelot API Gateway

- API Gateway open source para la plataforma .NET
- Ligero y se ejecuta en plataformas que soportan ASP .NET Core
- Provee todas las características de un API Gateway
- Usado por el mismo Microsoft en sus proyectos

Demo con OCELOT

Método de Aprendizaje

- Hazlo tu mismo
- Úsalo en proyectos



- Mentor
- Comunidad

- Libros
- Video cursos
- Revistas

Gracias

¿Alguna pregunta?

www.joedayz.pe

@joedayz

http://github.com/joedayz