

**Monolith vs**

**Microservices**



# Microservice Architecture and Docker

# Agenda

---

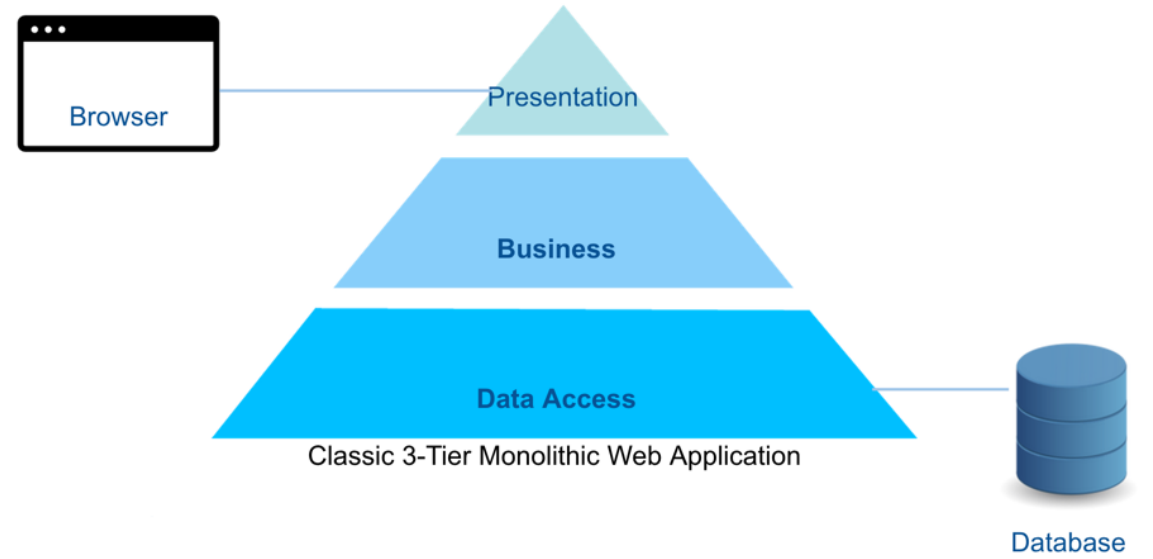
Monolithic Applications

Service Oriented Architecture (SOA)

Microservices

Docker

Gartner Hype Cycle



---

# Monolithic Applications

Traditional Applications:

Everything is integrated



●ステレオ録音に威力を発揮する  
ワンポイントステレオマイク  
別売：CM-2000 ¥7,900

But, what happens if...

---

We want a double cassette

Or a CD player

Or a digital radio

...

# We have to change the whole thing

---



# Again


---



# With Modular Applications

---



 Close Window



# With Modular Applications

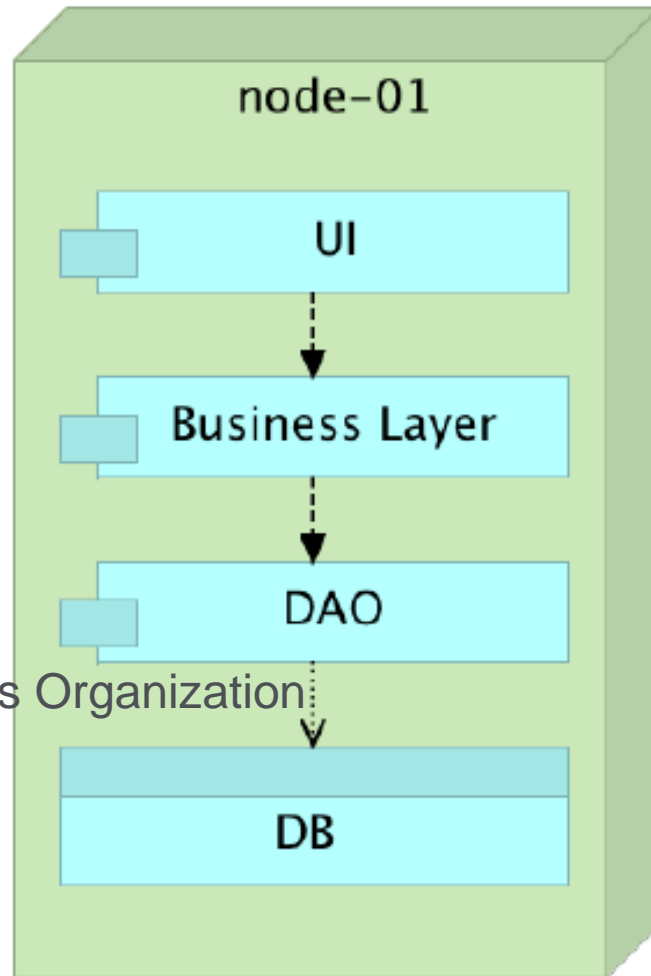


Each part is independent

# Do you Need a CD Player?

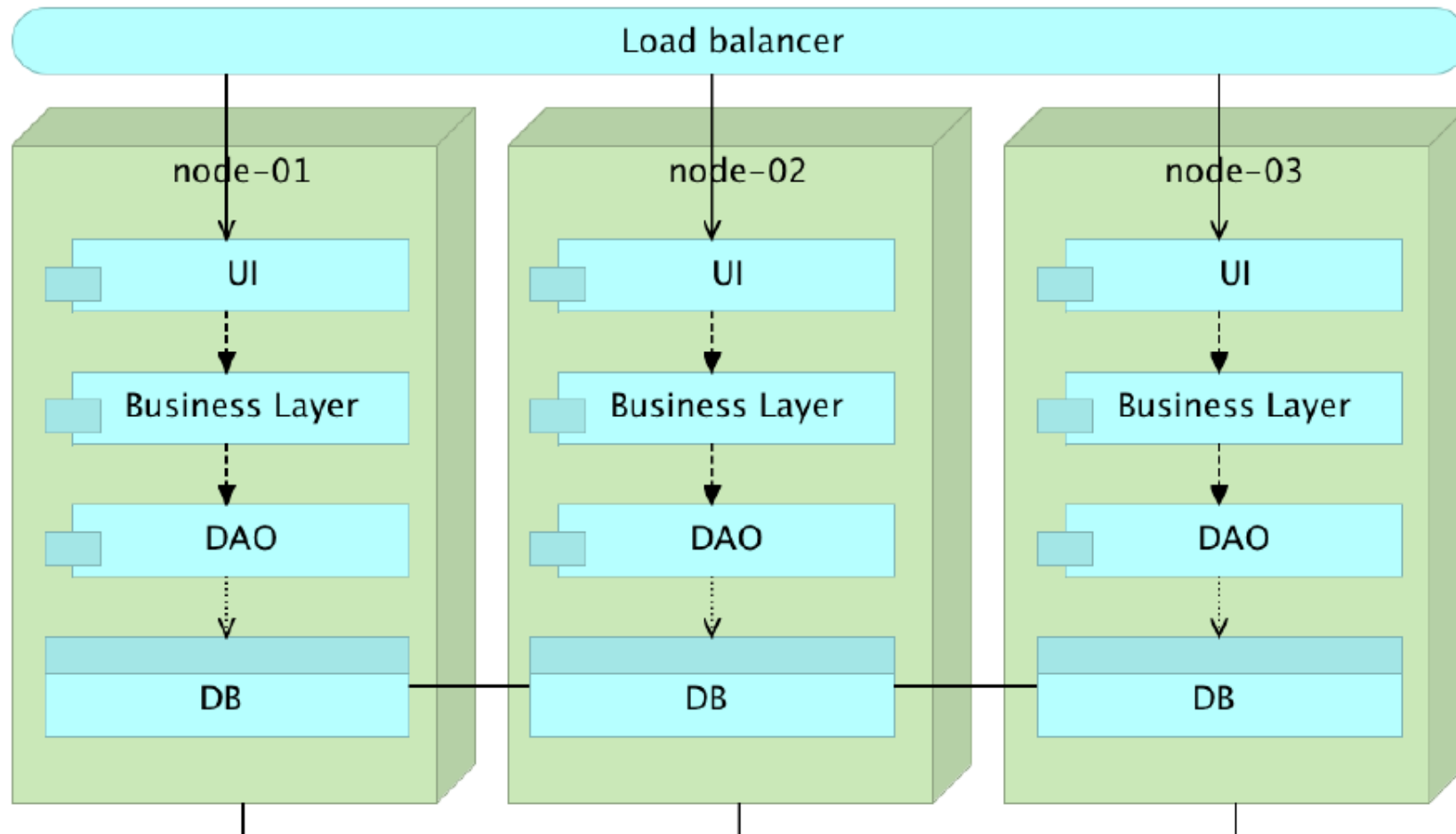


# Monolithic Applications



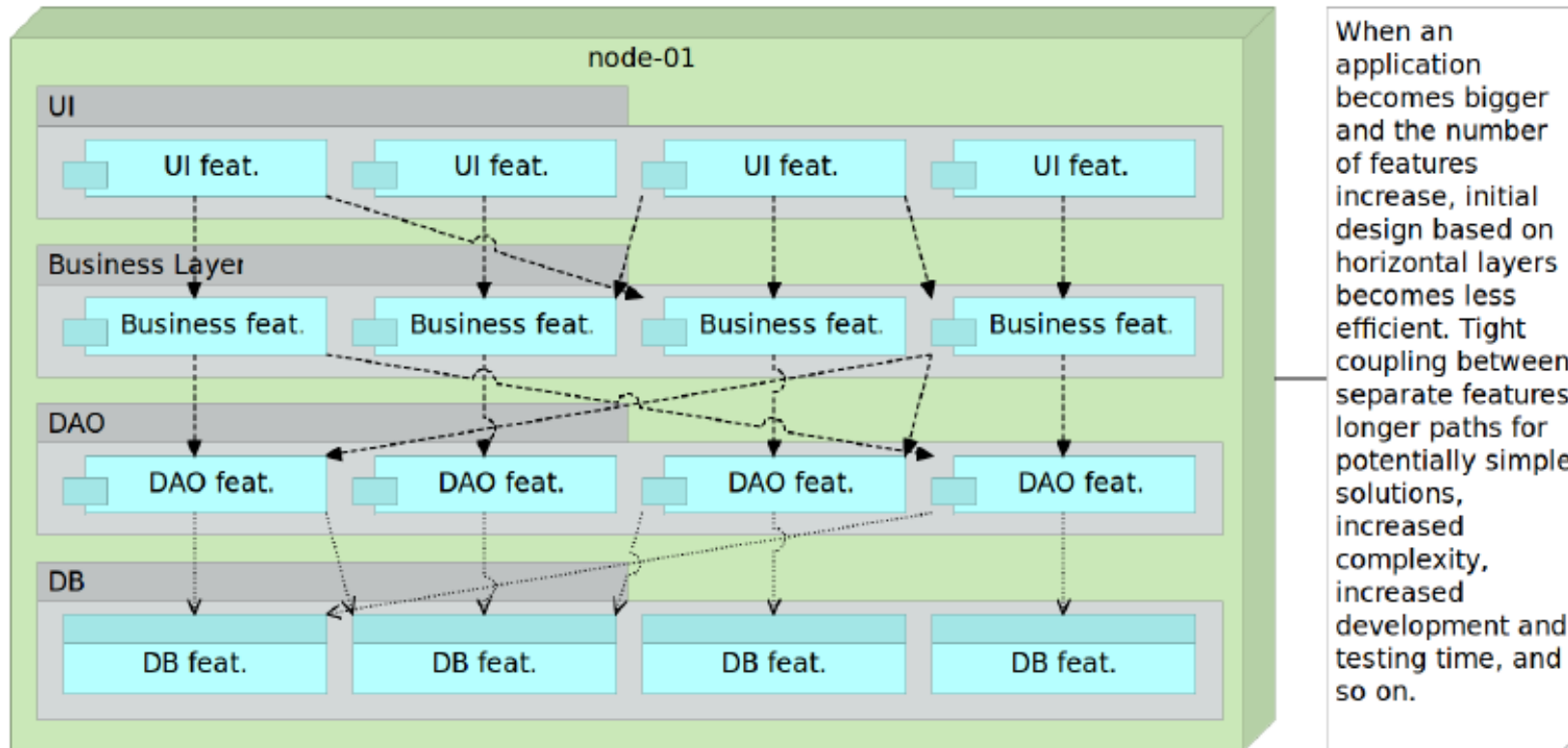
When an application is relatively small, splitting it into horizontal layers is a good idea. It provides a separation that makes development faster and easier as well as a separation based on type of the task code should do.

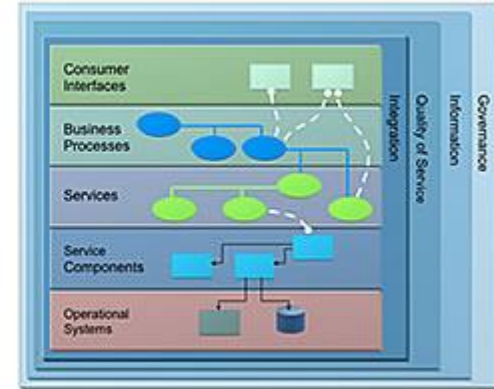
# Scaling Monolithic Application



Scaling monolithic applications is very resource inefficient since everything needs to be duplicated on multiple nodes. There is no option to detect bottlenecks and scale or separate them from the rest of the application.

# Monolithic Application with Increased Number of Features





---

# Service Oriented Architecture (SOA)



# How to Buy Tomatoes?

Lookup for the tomato sellers

*Yellow Pages: contain companies that are selling tomatoes, their location, and contact information.*



- ▶ Find the service offered according to my needs

*Where, when and how can I buy tomatoes?*



- ▶ Buy the tomatoes

*Do the transaction*



# How to Access the Service?

---

## Lookup for the Service Provider

*Registry: contain providers that are selling services, their location, and contact information.*

## Find the service offered according to my needs

*Where, when and how can I get the service?*

## Access the service

*do the transaction*



# Service-Oriented Architecture (SOA):

---

*The policies, practices, frameworks that enable application functionality to be provided and consumed as sets of services published at a granularity relevant to the service consumer.*

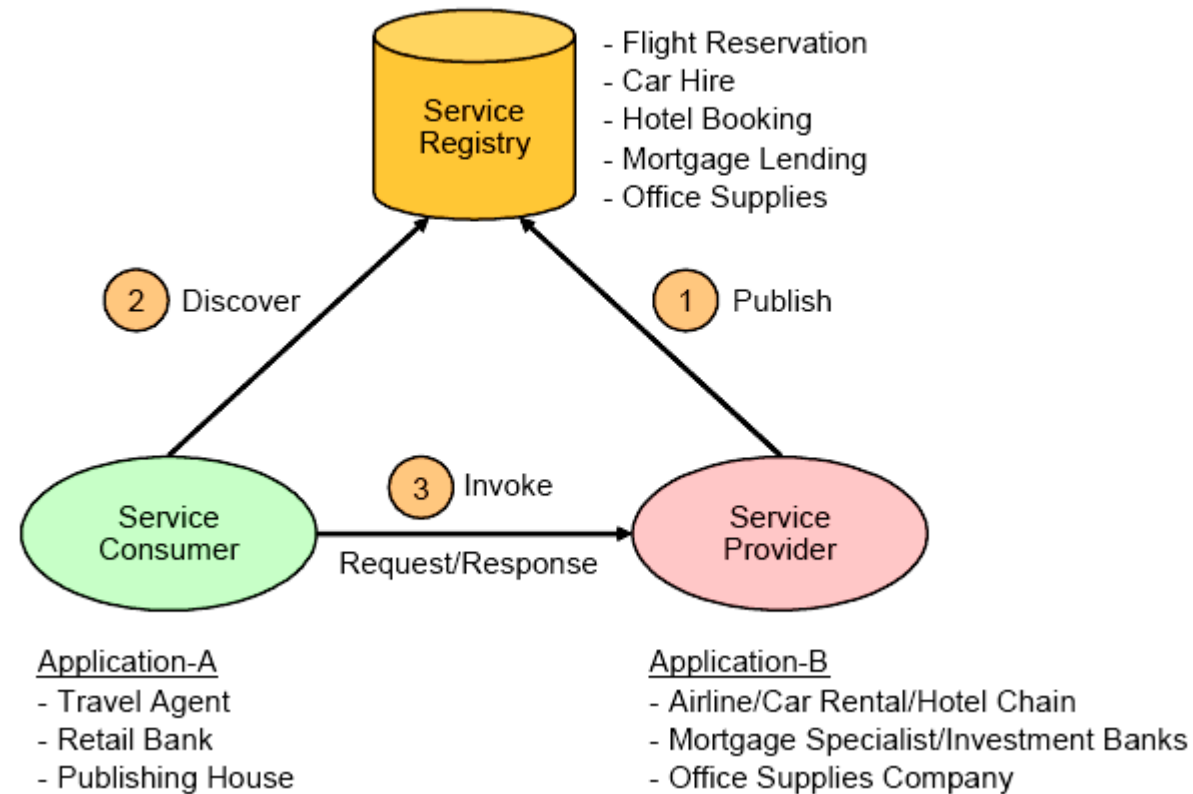
*Services can be*

- *invoked,*
- *published and*
- *discovered,*

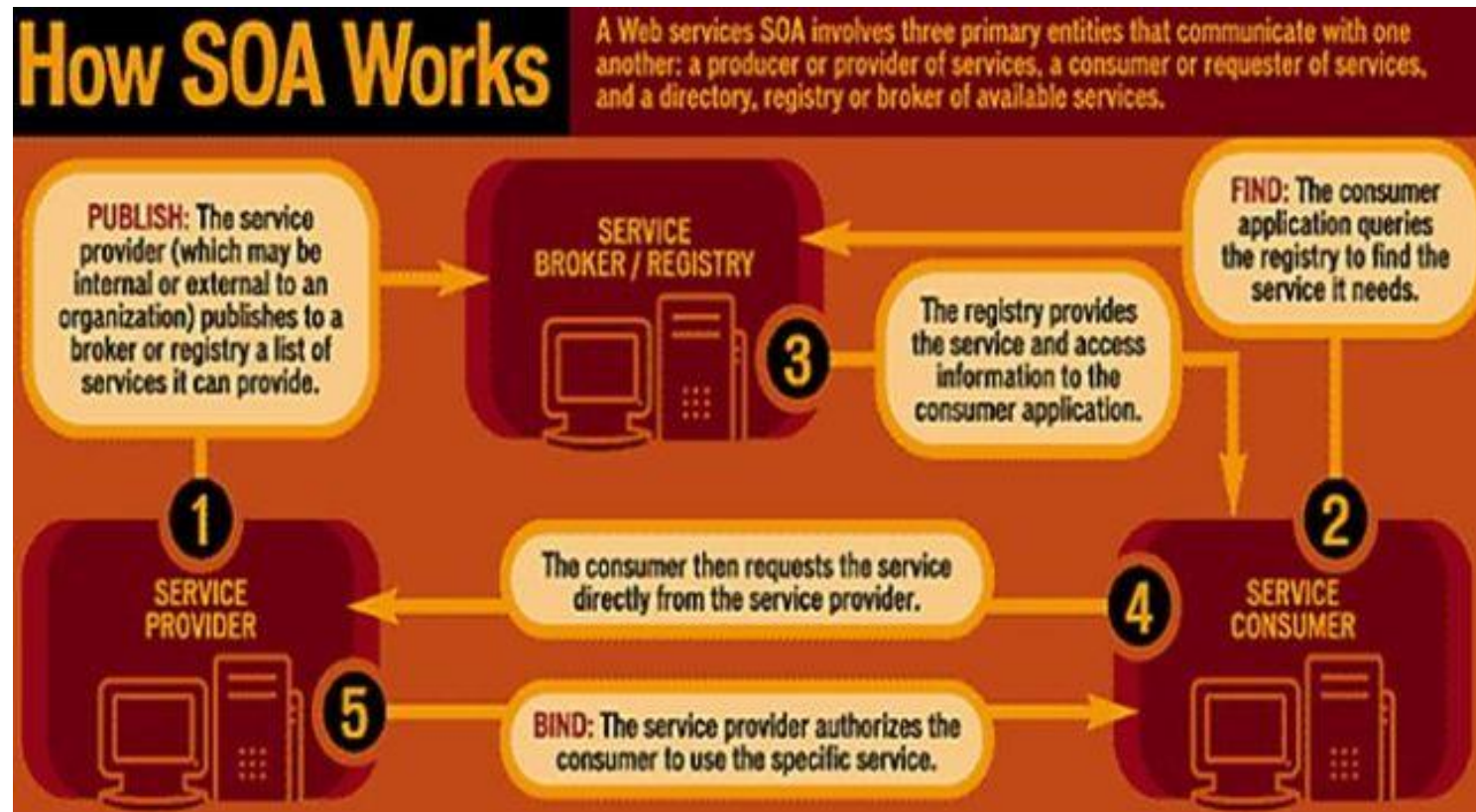
*and are abstracted away from the implementation using a single, standards-based form of interface.*

# SOA Components and Operations

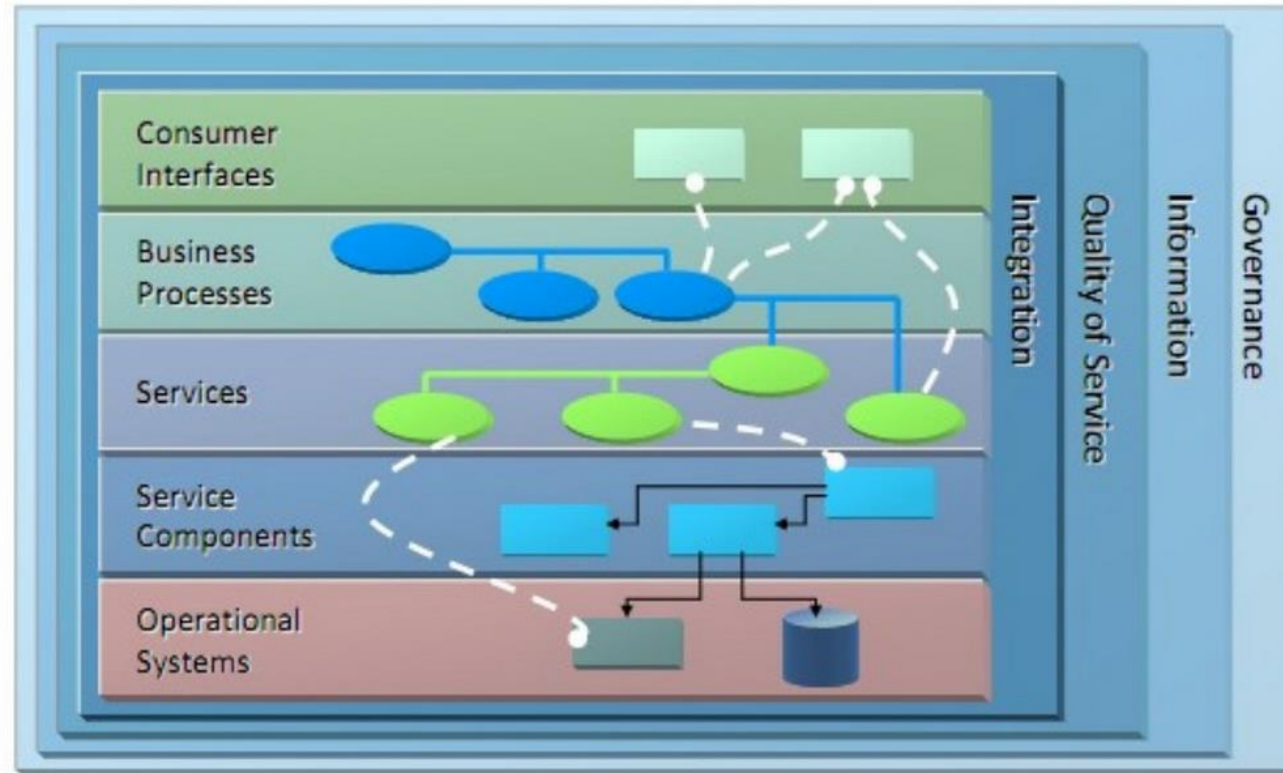
---



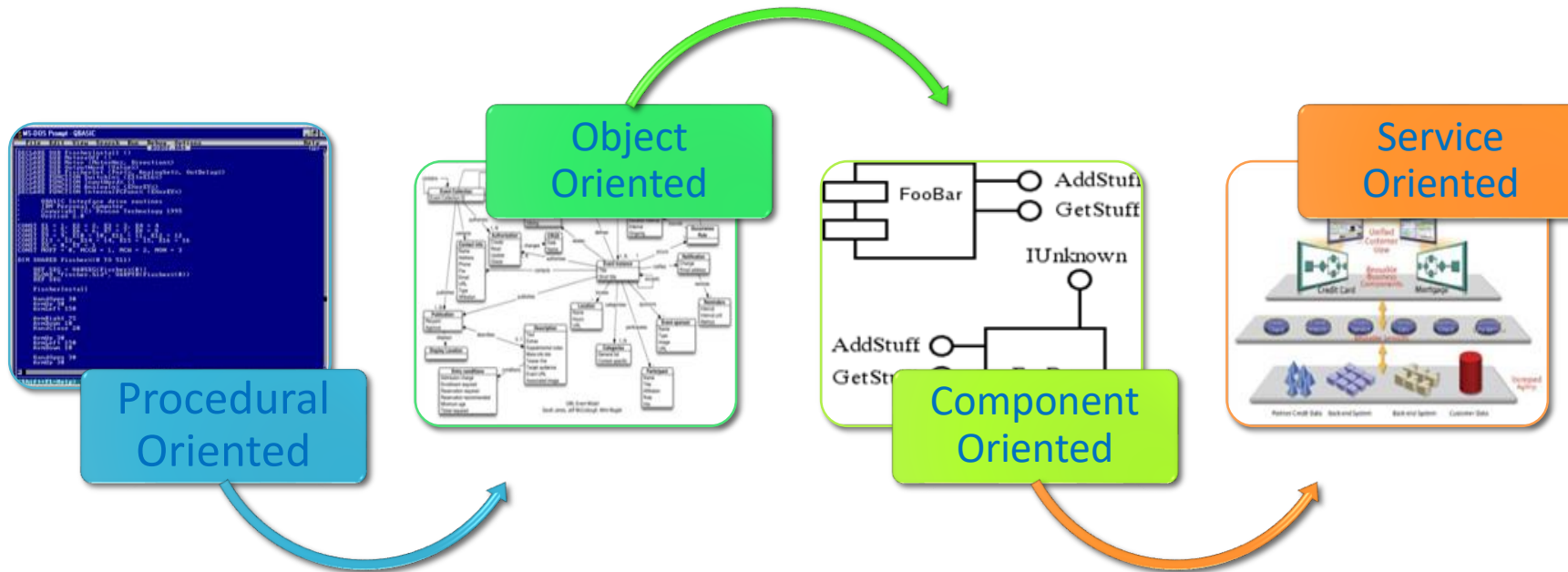
# How SOA Works?



# Open Group SOA Reference Architecture (SOA RA)



# How did it come to SOA?



# SOA Alphabet Soup

What Would You Choose?

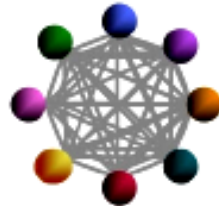
## SOA Challenges



# Evolution of Application Integration Patterns

**SOA builds flexibility on your current investments**  
*...The next stage of integration*

Messaging Backbone



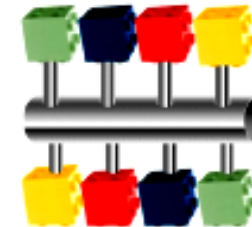
- Point-to-point connection between applications
- Simple, basic connectivity

Enterprise Application Integration (EAI)



- EAI connects applications via a centralized hub
- Easier to manage larger number of connections

Service Orientated Integration

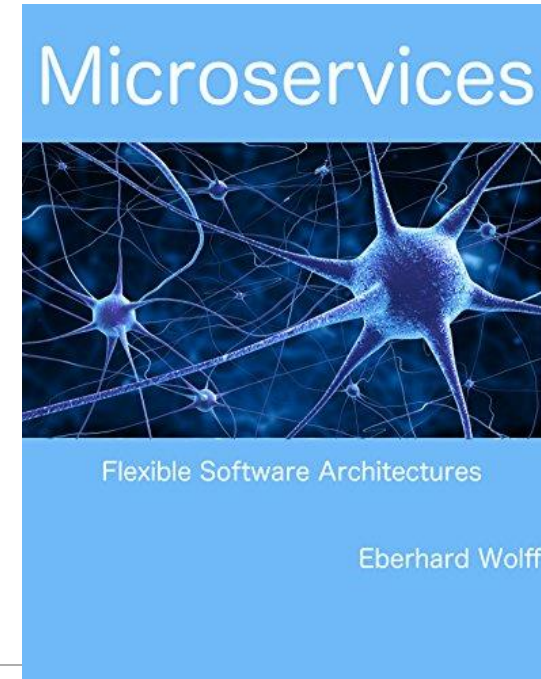
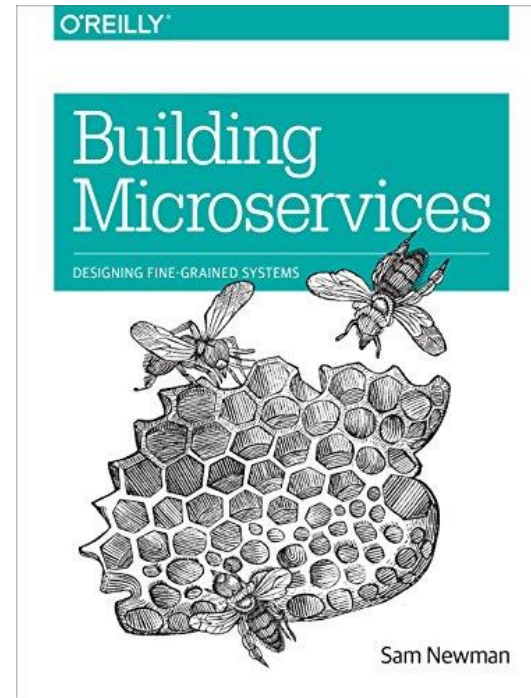
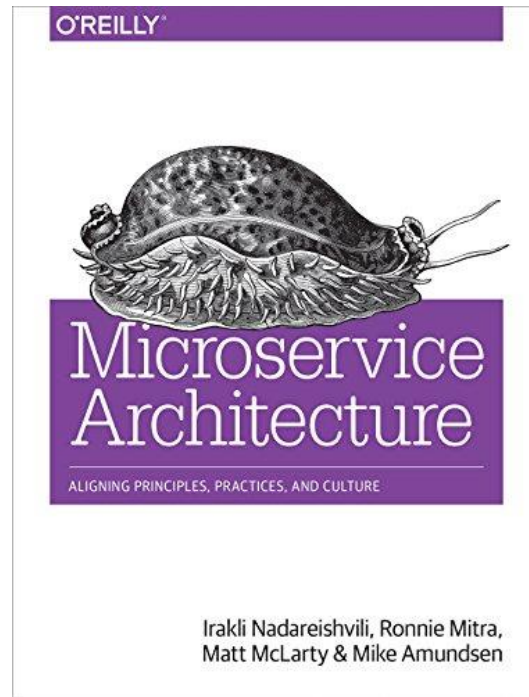


- Integration and choreography of services through an Enterprise Service Bus
- Flexible connections with well defined, standards-based interfaces

**Flexibility**

*As Patterns Have Evolved, So Has IBM*





---

# Microservices





# MONOLITHS

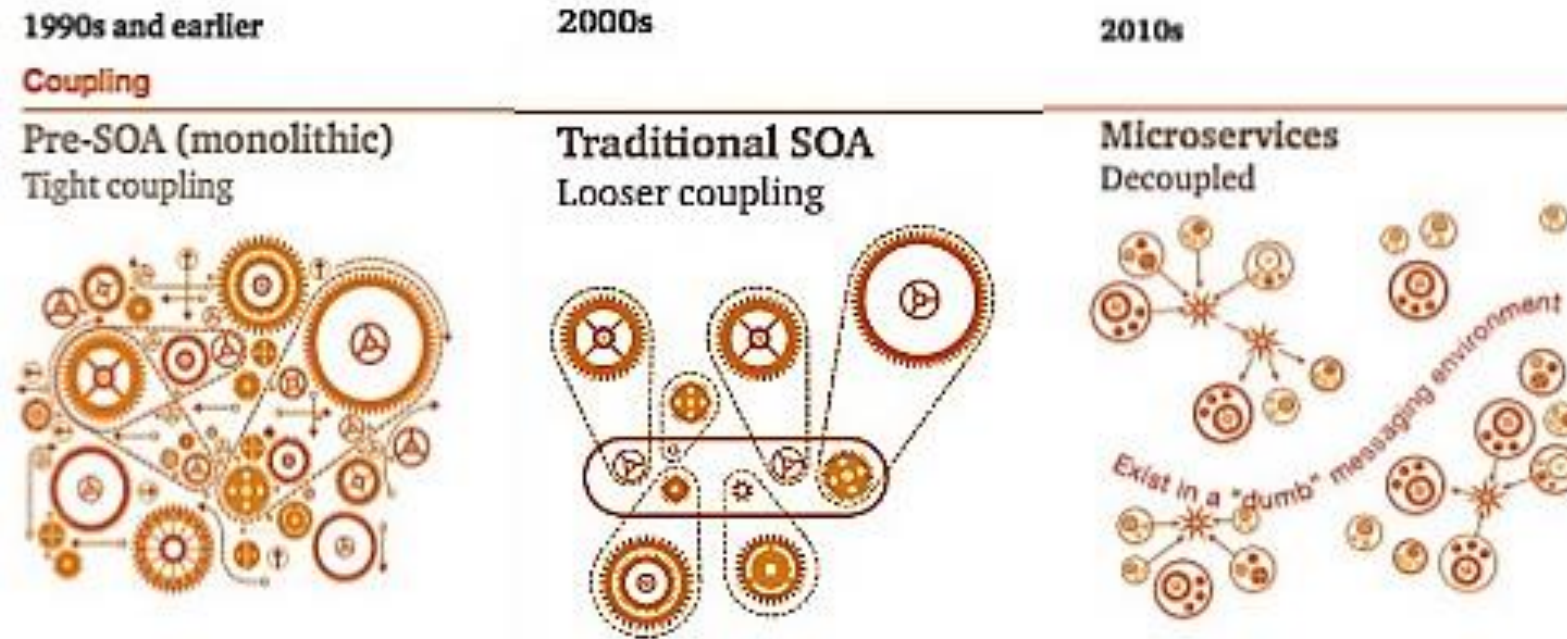
Hard to deliver, even harder to test and impossible to maintain





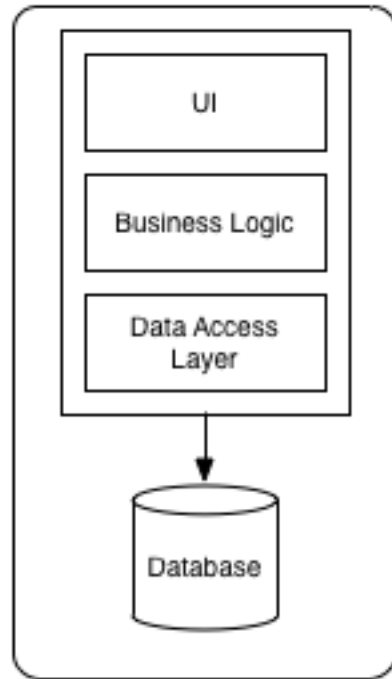
THE DEPENDENCIES WILL KILL YOU

# Microservices

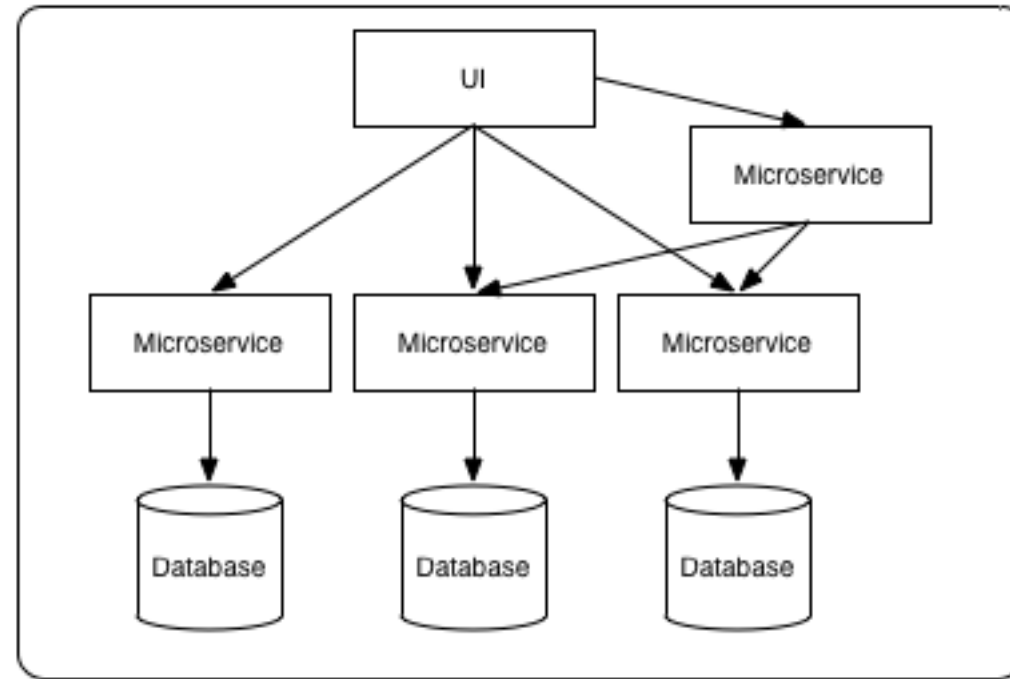


# Microservices

---

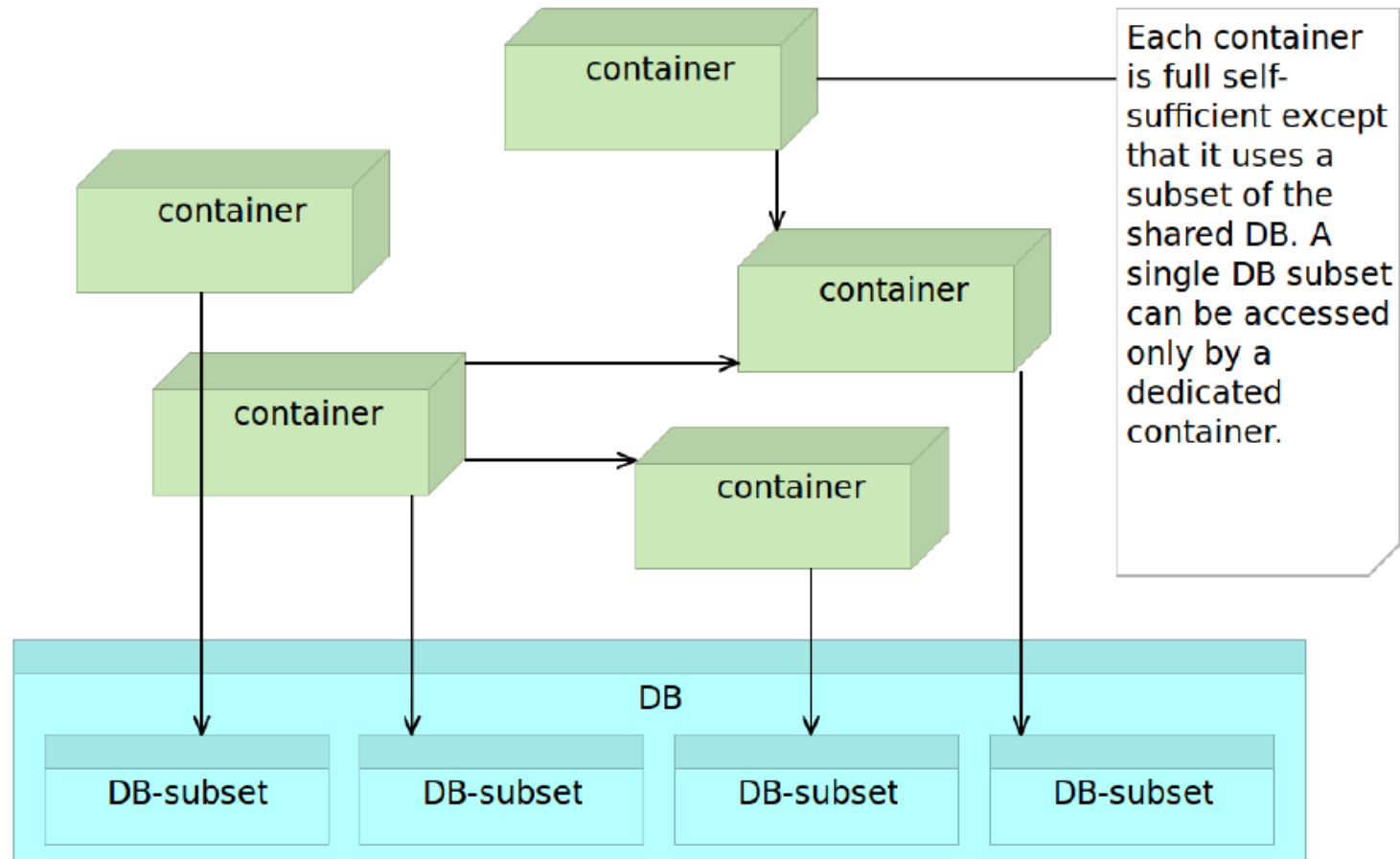


Monolithic Architecture



Microservices Architecture

# Microservices Accessing the Shared Database



# Microservices Characteristics

---

Many smaller (fine grained), clearly scoped services

- Single Responsibility Principle
- Independently Managed

Clear ownership for each service

- Typically need/adopt the “DevOps” model



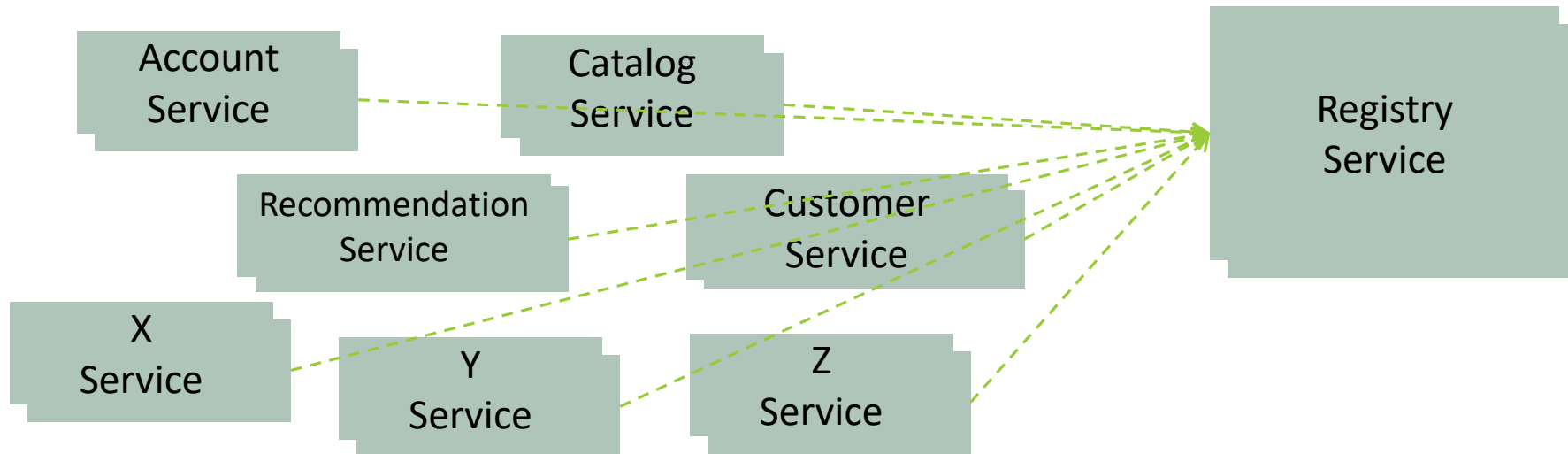


# Service Discovery

---

100s of MicroServices

- Need a Service Metadata Registry (Discovery Service)

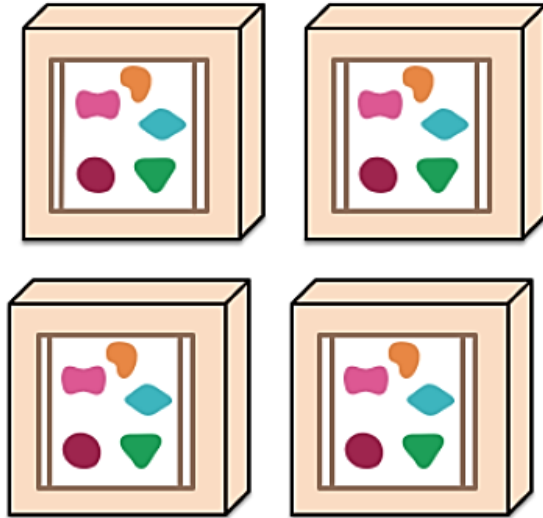


# Microservices. Scalability

*A monolithic application puts all its functionality into a single process...*



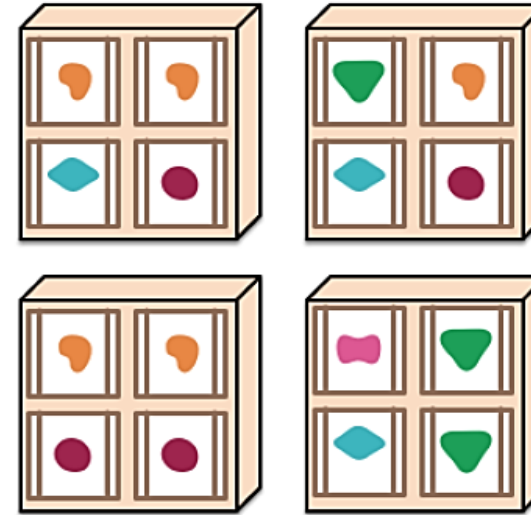
*... and scales by replicating the monolith on multiple servers*



*A microservices architecture puts each element of functionality into a separate service...*



*... and scales by distributing these services across servers, replicating as needed.*







---

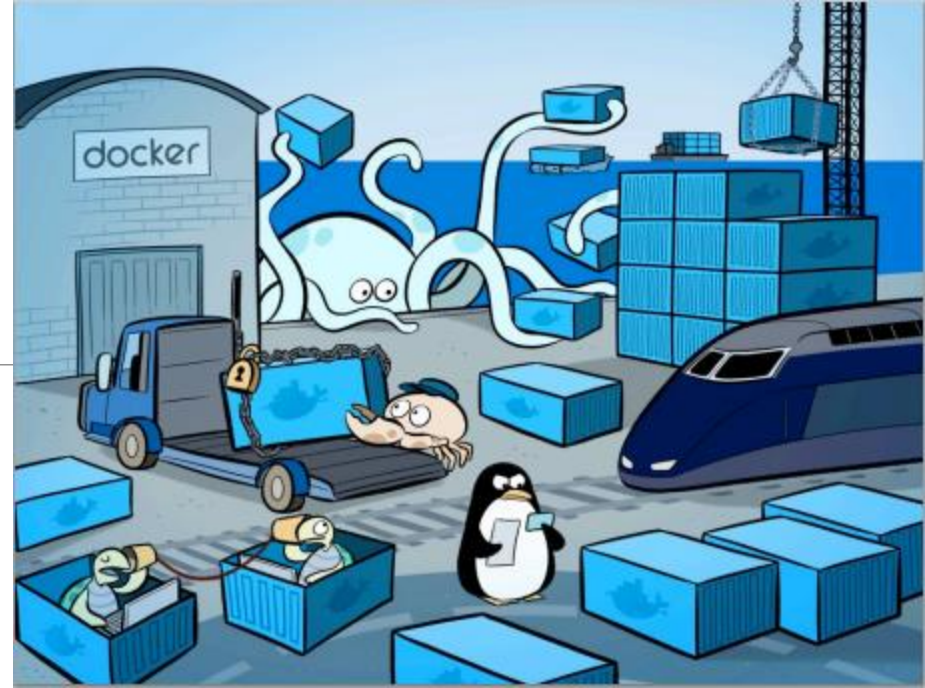
# Docker

# Docker: Containerization for Software

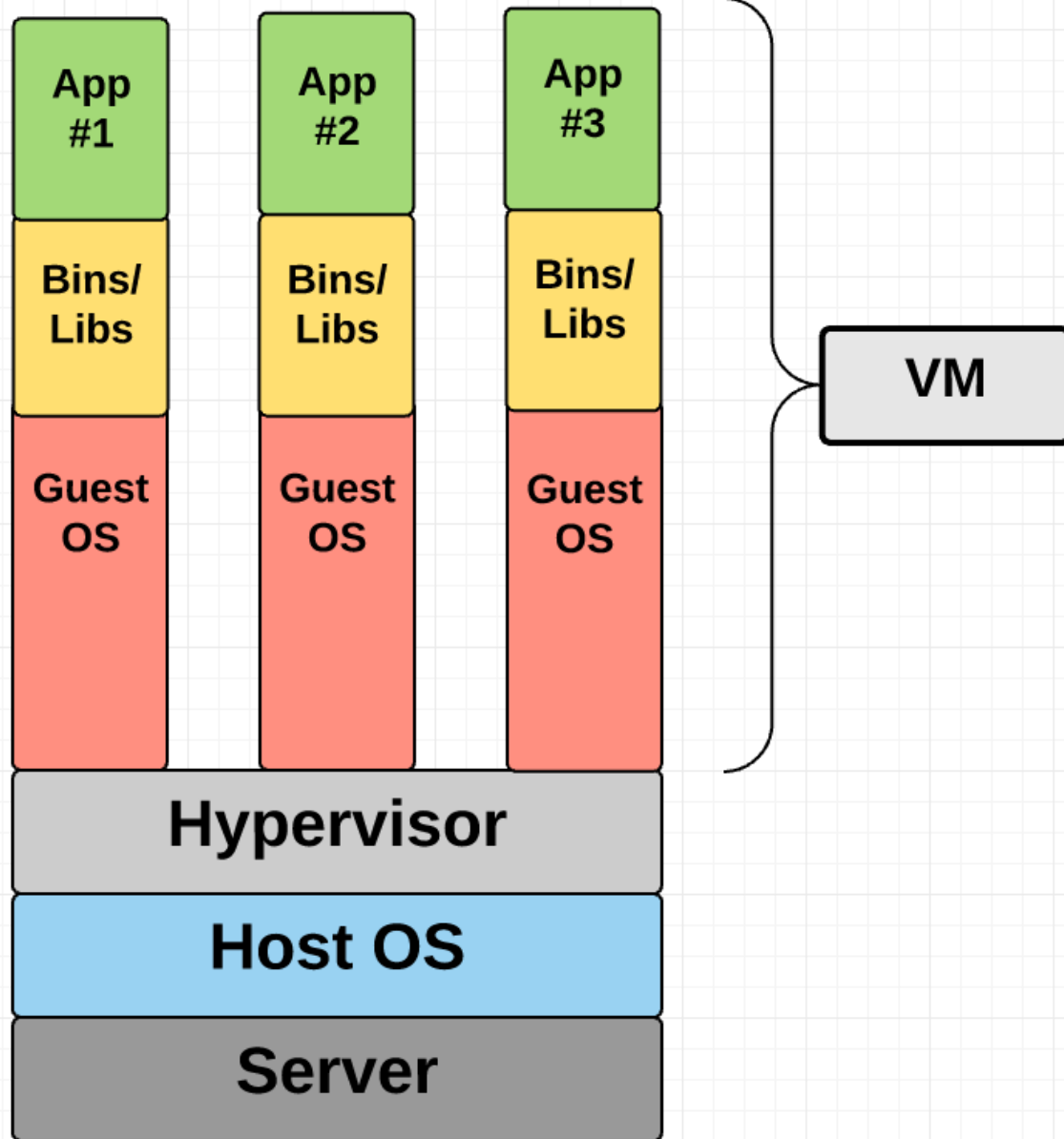
---



# Docker

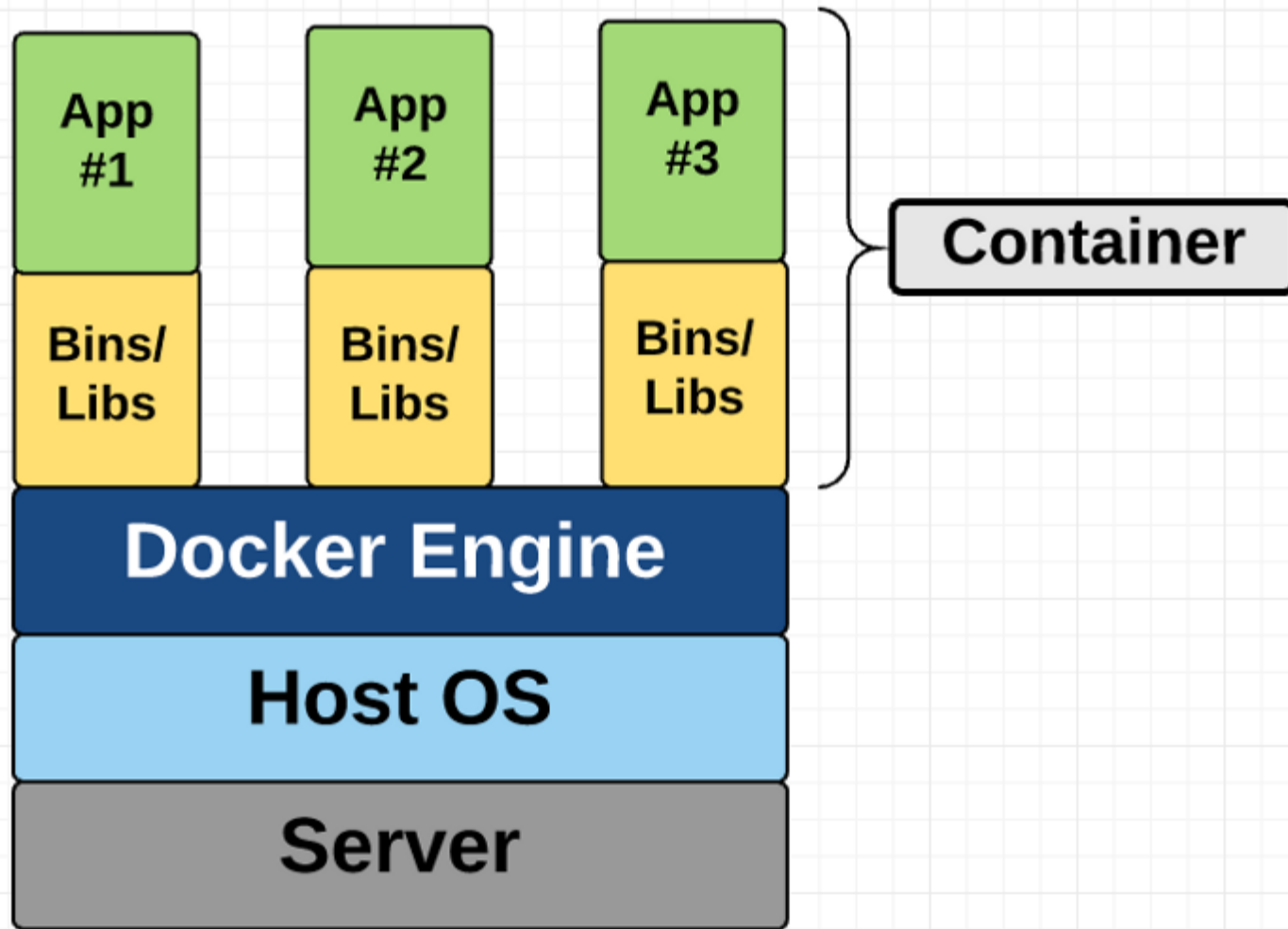


*“Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications”*



# Virtual Machine

---



# Container

# So why Docker?

---

Containers are far from new;

- Google has been using their own container technology for years.
- Others Linux container technologies include
  - Solaris Zones,
  - BSD jails, and
  - LXC, which have been around for many years.

Docker is an open-source project based on Linux containers. It uses Linux Kernel features.

# Docker Benefits

---

1. Local development environments can be set up that are exact replicas of a live environment/server.
2. It simplifies collaboration by allowing anyone to work on the same project with the same settings, irrespective of the local host environment.
3. Multiple development environments can be run from the same host each one having different configurations, operating systems, and software.
4. Projects can be tested on different servers.
5. It gives you instant application portability. Build, ship, and run any application as a portable container that can run almost anywhere.



# Why Docker?

**Ease of use.** It allows anyone to package an application on their laptop, which in turn can run unmodified anywhere

- The mantra is: “build once, run anywhere.”






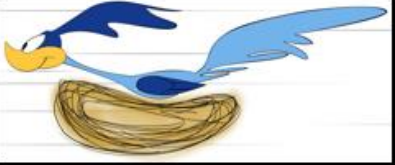


**Speed.** Docker containers are very lightweight and fast. Since containers are just sandboxed environments running on the kernel, they take up fewer resources. You can create and run a Docker container in seconds, compared to VMs which might take longer because they have to boot up a full virtual operating system every time.

**Docker Hub.** Docker users also benefit from the increasingly rich ecosystem of Docker Hub, which you can think of as an “app store for Docker images.” Docker Hub has tens of thousands of public images created by the community that are readily available for use.

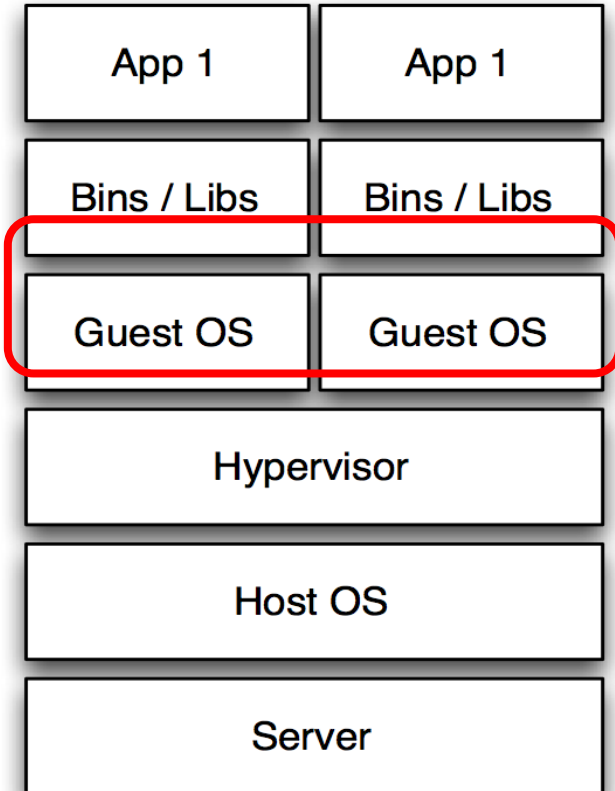
**Modularity and Scalability.** Docker makes it easy to break out your application’s functionality into individual containers. With Docker, it’s become easier to link containers together to create your application, making it easy to scale or update components independently in the future.



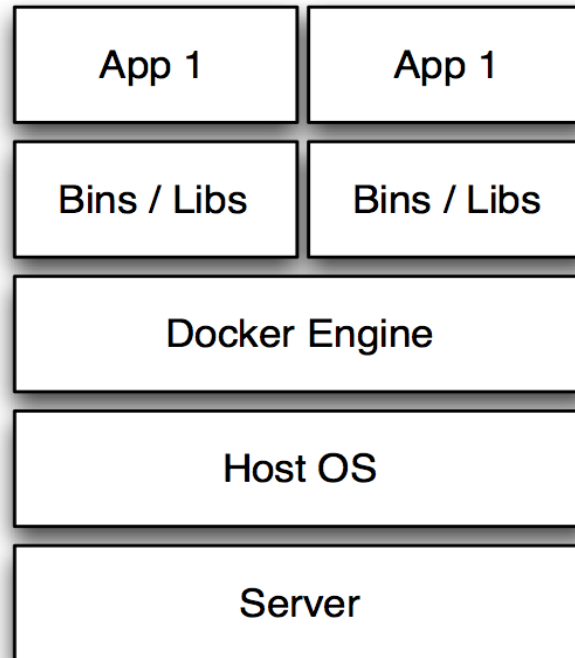
# VM vs. Docker

		
Size		
Startup		
Integration		

# VM vs. Docker (Containers)



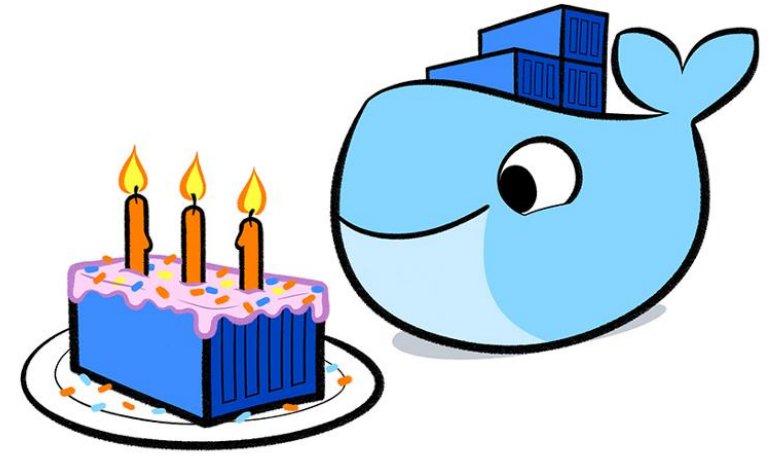
**Virtual Machines**



**Docker**

## Docker Engine

Docker engine is the layer on which Docker runs. It's a lightweight runtime and tooling that manages containers, builds, and more.



---

# Gartner Hype Cycle

# Gartner Hype Cycle

