





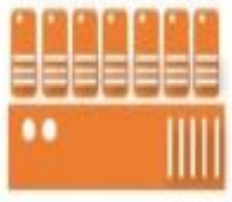





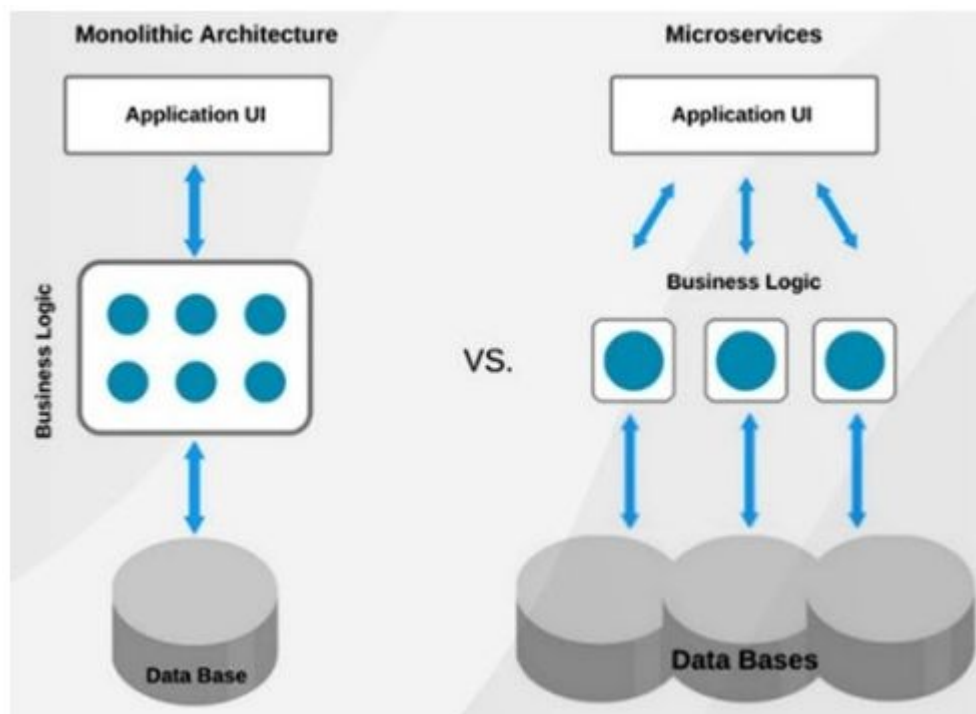


Microservices and containers

- **Monolithic and Microservices**
- **There has been a lot of change in the development & deployment since the '80s till now. We can trace the evolution of how organizations have developed, deployed, and managed applications over three distinct phases:**

	Development Process	Application Architecture	Deployment and Packaging	Application Infrastructure
~ 1980	Waterfall 	Monolithic 	Physical Server 	Datacenter 
~ 1990				
~ 2000	Agile 	N-Tier 	Virtual Servers 	Hosted 
~ 2010	DevOps 	Microservices 	Containers 	Cloud 
Now				

- **80's – mid 90's: Data Centers with physical servers, running monolithic applications, developed using traditional waterfall methodologies.**
- **Late 90s–00s: Datacenters and hosting providers running Unix/Linux servers, the emergence of virtualization (VMWare, KVM), running n-tier applications, developed using agile methodologies.**
- **~ 2010 – Now: Cloud increasingly replacing data centers and hosting, decomposition of applications into microservices, running on a container infrastructure, managed and orchestrated by Kubernetes, developers, and operations collaborating using DevOps methodologies.**



Monolithic Application

- A monolithic application is constructed as one unit which means it's composed all in one piece. The Monolithic application describes a one-tiered software application within which different components combined into one program from a single platform. The three components are the user interface, the data access layer, and the data store.
- The user interface is the entry point of the application and is also the point of the program that a user will interact with.
- Data access layer is the layer of the program which will wrap the data stored.
- The data store is the most fundamental part of the system and is liable for storing data and retrieving it.

Advantages of Monolithic:

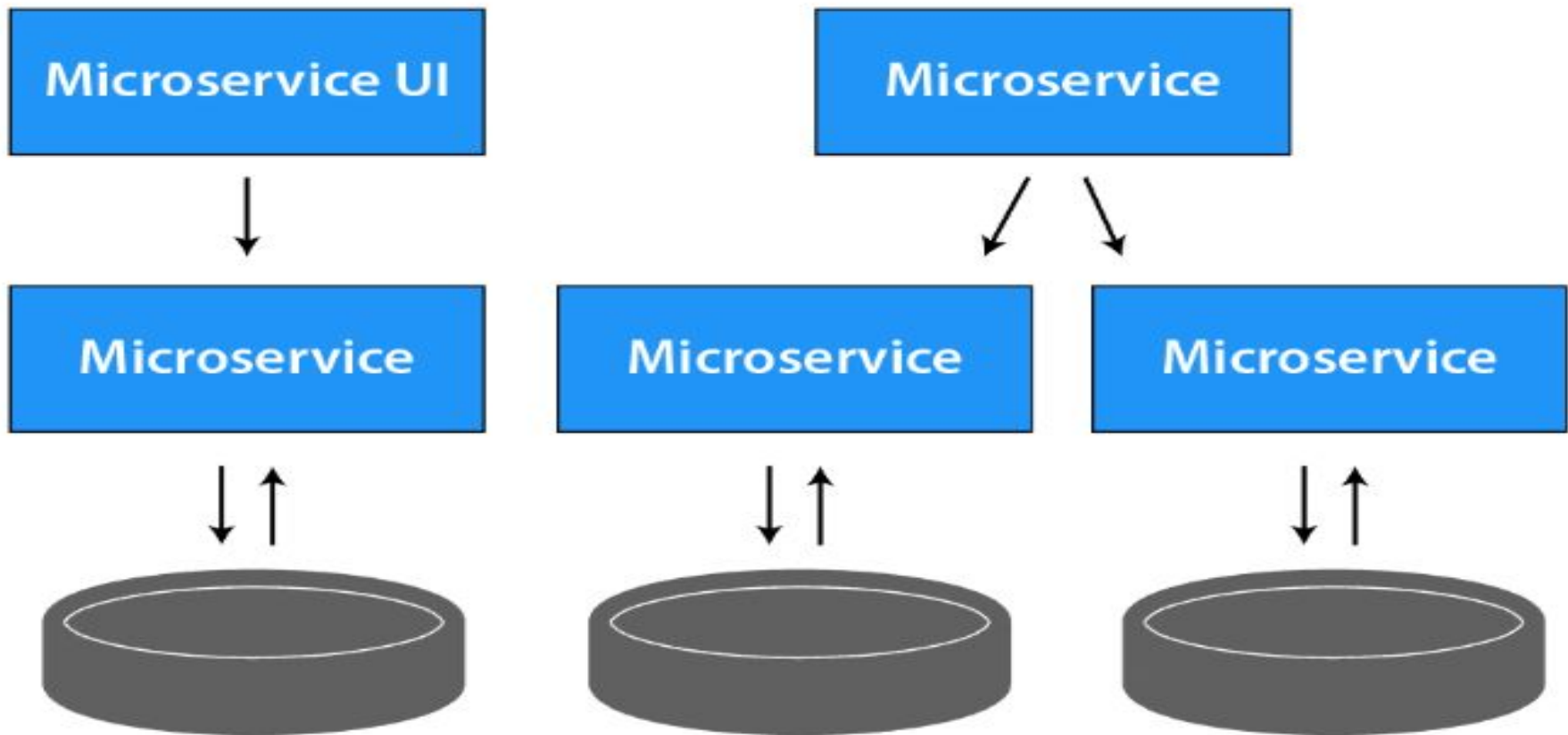


- **Simple to deploy** – In monolithic applications, we don't need to handle many deployments just one file or directory.
- **Easier debugging and testing** – monolithic applications are much easier to debug and test. Since a monolithic app is a single indivisible unit, we can run end-to-end testing much faster.
- **Performance**: Components in a monolith typically share memory which is Quicker than service-to-service communications.

Disadvantages of Monolithic:



- **New technology barriers – It's extremely problematic to use new technology in a monolithic application because then the whole application must be rewritten.**
- **Scalability- You can't scale components independently, only the whole application.**
- **Size – It becomes overlarge in size with time and becomes difficult to manage.**
- **Difficult to understand – For any new developer joining the project, it's very difficult to grasp the logic of an oversized monolithic application whether or not his responsibility is related to one functionality.**



Micro-services Architecture

- a) The idea of microservices is to separate our application into smaller sets and interconnected services rather than building one monolithic application.**
- b) Each module supports a selected business goal and uses a simple and well-defined interface to communicate with other sets of services.**
- c) Each microservice has its own database. Having a database per service is crucial if you wish to learn from microservices because it ensures loose coupling.**

Advantages of Microservices:



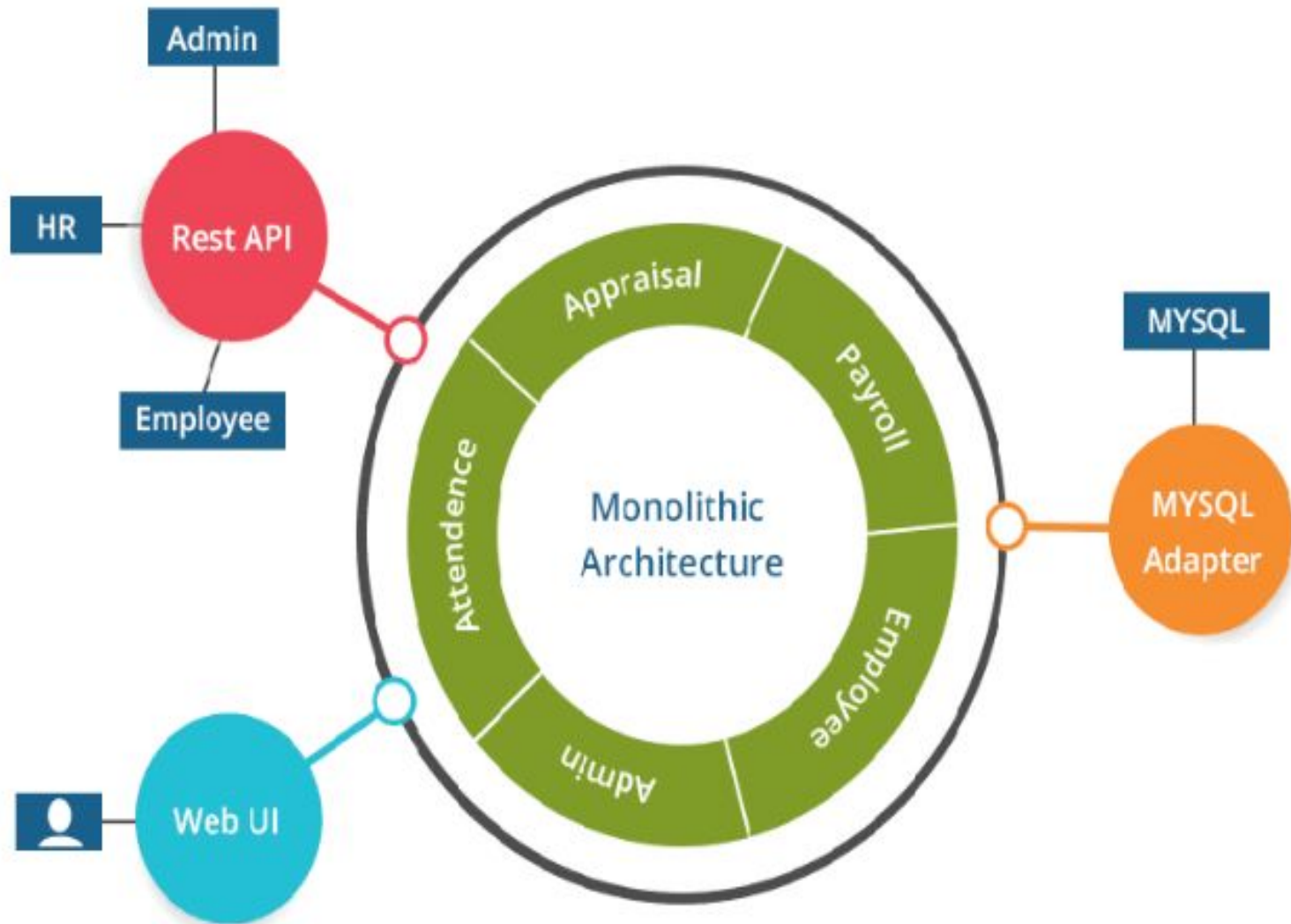
- The application starts faster, which makes developers more productive, and accelerates deployments.
- It enables us to arrange the event effort around multiple teams. Each team is responsible for one or more single service. Each team can develop, deploy, and scale their services independently of all of the opposite teams.
- Microservices allow the usage of the most appropriate technology for various services. Meaning each team building the selected service can choose their preferred programming language and framework, as they're working independently.
- Once the organization develops and implements a microservices architecture, it's the schematic diagram that may be reused and expanded upon for future projects and services.
- Each service is a separate object within the microservices framework which enables their independent functioning

Disadvantages of Microservices:



- **Since everything is now an independent service, we've to carefully handle requests traveling between our modules. In one such scenario, developers are also forced to write extra code to avoid disruption.**
- **Communication between services may be complex and there's a higher chance of failure during communication between different services.**
- **The developer has to solve the problem, like network latency and load balancing.**
- **While unit testing may be easier with microservices but integration testing isn't. The components are distributed and developers can't test a complete system from their individual machines.**

Why MicroServices? ^



- **The good thing about using microservices is that development teams are ready to rapidly build new components of apps to fulfill changing business needs.**
- **In microservices, every application resides in separate containers along with the environment it needs to run.**
- **It allows it to maximize deployment velocity and application reliability.**
- **There's no risk of interfering with the other applications. It also allows optimizing the sources micro services multiply team works on independent services.**

- **In Microservices, we run all the applications on Containers (Docker), so the boot-up time and memory consumption of applications decrease, thus the performance of the application increases.**
- **In organizations, not one or two but multiple numbers of Containers (Docker) are running.**
- **So, to manage them, we use Kubernetes. Kubernetes comes in handy because it provides various facilities like self-healing, scalability, auto-scaling, and declarative state.**

What's the Diff: Monolithic vs Microservices

Monolithic

It is built as one large system

Not easy to scale based on demand

It has a shared database

Large code base makes IDE slow

Continues deployment becomes difficult

It is extremely difficult to change technology or language or framework because everything is tightly coupled and depends on each other.

Microservice

It is built as a small independent module

Easy to scale based on demand.

Each project and module have their own database

Each project is independent and small in size

Continues deployment is possible here

Easy to change technology or framework because every module and project is independent.

Concept of Containers

- Containers are Operating System-level virtualization technologies that allow multiple isolated applications to run on a single host or instance.
- They provide agile, cost efficient, and automated application deployment in which processes are packaged with all the necessary dependencies inside their own separate instances for easy access and portability.
- Containers also offer greater levels of isolation between resources and improve resource utilization compared to traditional virtual machines by using fewer hardware resources.

- Containers are a lightweight alternative to VMs for providing **isolated** operating environments for your workloads.
- They use a different method of **abstracting** resources.
- Instead of using a hypervisor, containers share the **kernel** of the host operating system (OS).
- As a result, they avoid the infrastructure overhead of a full-blown OS and provide only those resources (i.e., **installations, dependencies, and code**) that your applications actually need.
- This means they can stop and start more quickly in response to **fluctuating scaling** requirements and offer better overall performance than VMs.

Containers

Communication	Use a single network for easier maintenance and configuration of multiple servers
Scalability	Containers can scale easily to adapt to changing workloads, and they can be deployed, replicated, and managed easily. This allows applications to adapt to changing loads with minimal effort.
Management	Container management involves organizing, scaling, and maintaining containerized applications across various environments. This includes container orchestration, which automates the deployment, networking, scaling, and lifecycle management of containers.

Microservices

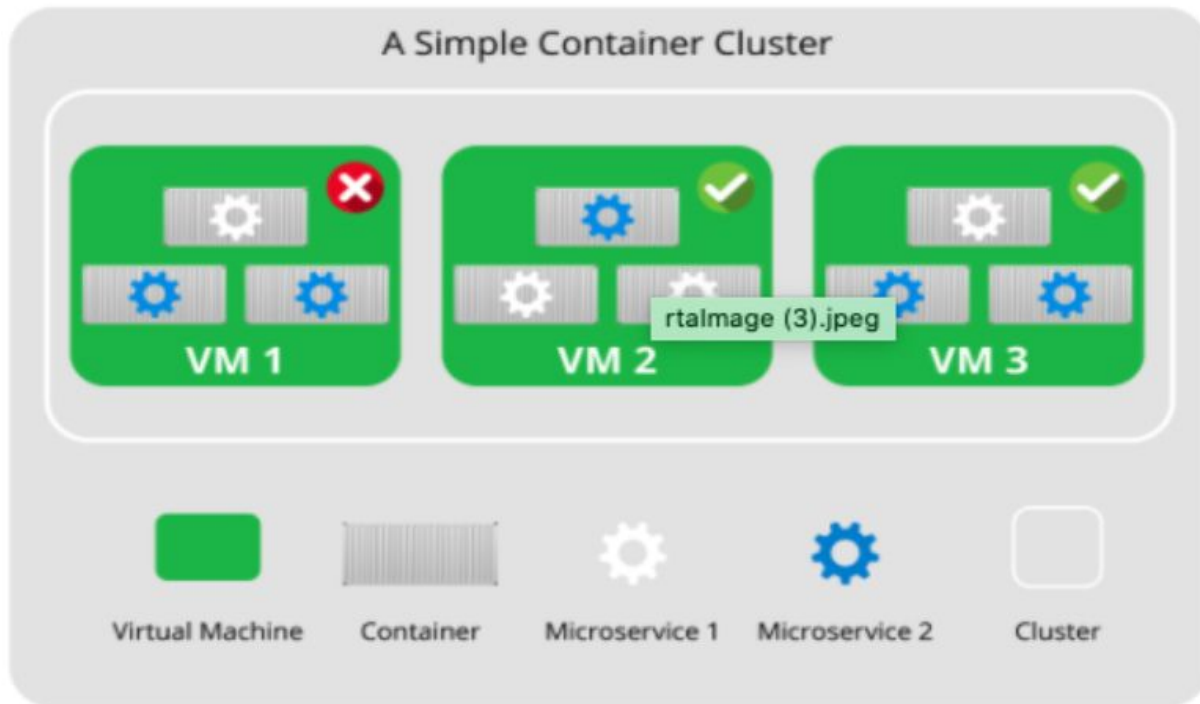
Modular nature allows individual clusters or services to be on separate networks with their own address spaces
Microservices can scale independently, allowing each component of an app to be scaled depending on the resources it needs. This can efficiently use resources instead of having multiple copies of the entire application.
Microservices performance testing is a critical aspect of developing and maintaining microservices-based applications. This involves assessing performance metrics such as response time, throughput, and the scalability of both individual microservices and the entire application.

Containers and microservices



- Containers and microservices are two important technologies driving the future of development.
- While they share many similarities, there are key differences that must be understood when determining which approach is right for an application.
- Containers enable portability and resource efficiency while offering great scalability options through containerization technology such as Docker or Kubernetes.
- On the other hand, Microservices allow flexibility due to their modular design, allowing applications to scale up without major changes in architecture all the while leveraging communication between services over a network connection; this also makes DevOps easier than ever before!

- To better performance and a lower infrastructure footprint, containerized microservices are typically more robust than a traditional monolithic application.
- For example, if you deploy an entire application to a VM, when the machine goes down, the whole application goes down with it. But, with containers, you can replicate microservices across a cluster of smaller VMs.
- So, in the event of a VM failure, the application will still be able to use the other microservices in the cluster and continue to function.
-





L OVELY
P ROFESSIONAL
U NIVERSITY



L OVELY
P ROFESSIONAL
U NIVERSITY



L OVELY
P ROFESSIONAL
U NIVERSITY