CONTAINERS VS MICROSERVICES: WHAT'S THE DIFFERENCE?

Today's digital age is fueled by users who demand more and more value from technology services. The smartphone and the cloud have combined to usher in an age where software runs the world, whether it is ordering food or a ride, paying for utilities, or getting your entertainment fix.

And the design and deployment of software has evolved to respond to this user driven demand, as traditional monolithic architectures have proven difficult when it comes to supporting rapid scaling and experimentation.

Containers and microservices are examples of modern approaches that are fast becoming the go-to model especially for mobile and web apps.

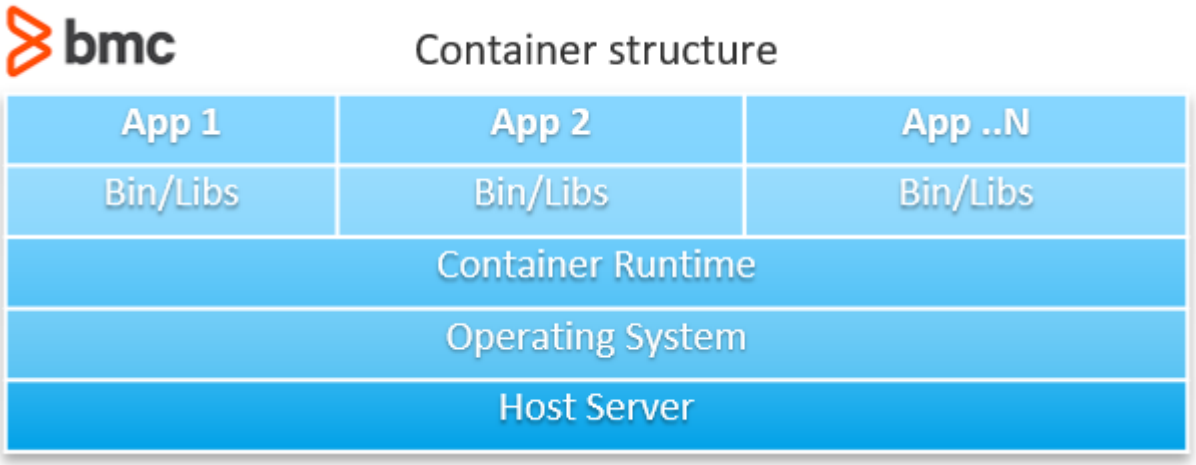
What are containers?

A container is a bundling of an application and all its dependencies as a package, that allows it to be deployed easily and consistently regardless of environment. These dependencies include binaries, libraries and configuration files needed to run the app.

Just like how shipping containers standardized transportation of cargo globally bringing about efficiency and speed, software containers solve the problem of how to get software to run reliably when moved from one computing environment to another. Whether it's deploying on a developer's laptop, a local test environment, or a production environment on the cloud, a container abstracts away any differences in OS distributions and underlying infrastructure.

In terms of structure, a container rides on a host operating system, similar to a virtual machine but share the OS kernel so it is more lightweight and boots faster using less memory. The container is hosted on a container runtime rather than a hypervisor; therefore, many more containers (several MB in size) can be hosted in a single server than VMs (several GB in size).

The depiction of the structure of a container is shown below:



In terms of popularity, Kubernetes from Google is the best known and most widely used free and open source container management system. On the other hand, Docker is best known commercial container management solution. While Linux has traditionally been the go to OS for container tech, Windows hasn't been left behind through Microsoft's Hyper-V containers.

(See how Kubernetes & containers work together.)

What are microservices?

Gartner defines a microservices as a service-oriented application component that is:

- Tightly scoped
- Strongly encapsulated
- Loosely coupled
- Independently deployable
- Independently scalable

According to AWS, microservice architecture involves building an application as independent components that run each application process as a service, with these services communicating via a well-defined interface using lightweight APIs.

The characteristics of microservices as described by microservices.io is that they are:

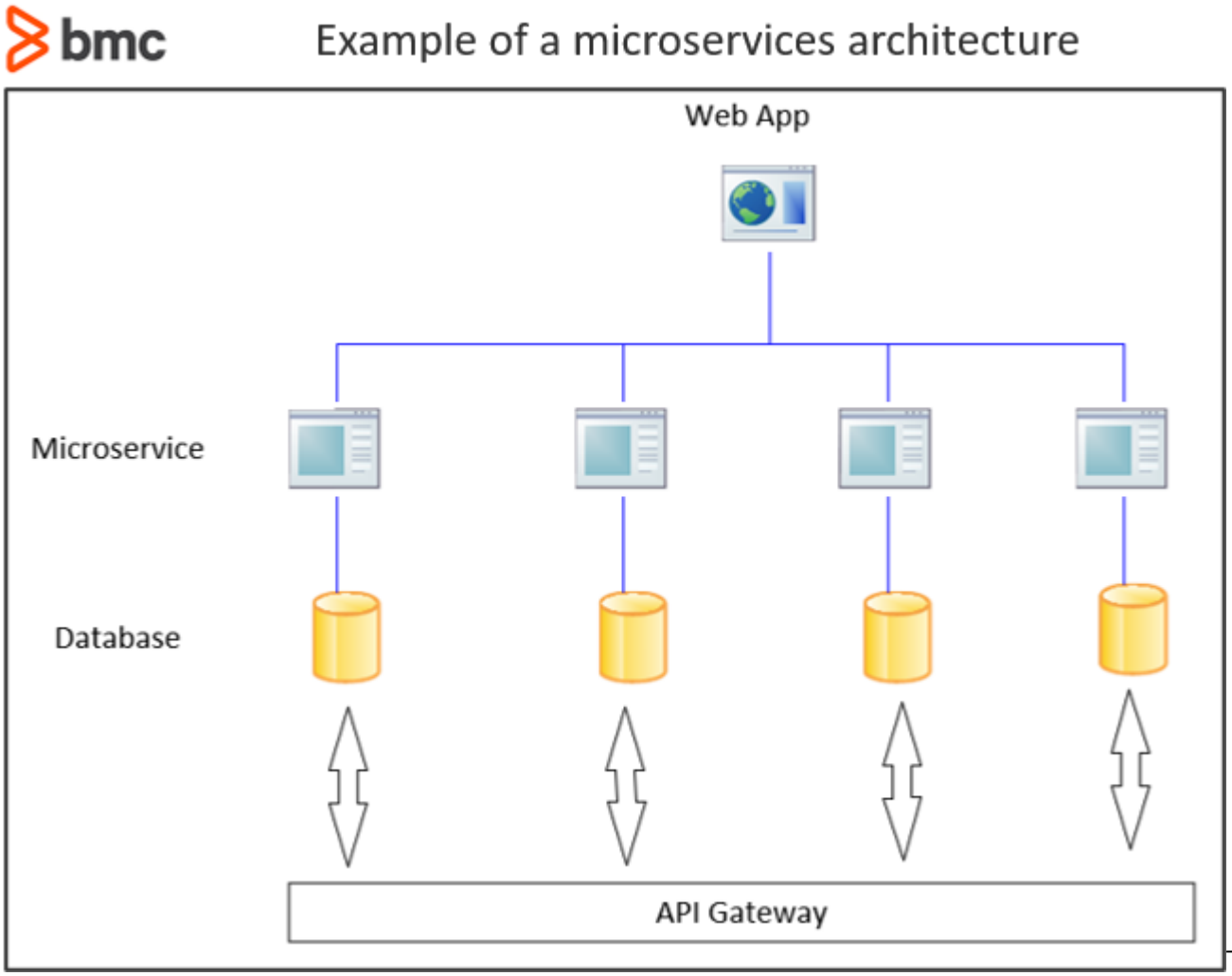
- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Organized around business capabilities; and
- Owned by a small team

Take the example of an e-commerce web application as shown below. A microservice architecture approach would see the separation of the:

- Logging service

- Inventory service
- Shipping service

Each service would have its own database and communicate with the other services via an API gateway. The development and management of each service is done separately, meaning that a service can be scaled or modified to suit different needs and situations without necessarily breaking the entire application.



The approach to microservices adoption has been geared towards refactoring existing monolithic applications, rather than building new applications from scratch. While microservices provide agility and scalability options, they also require relevant supporting infrastructure especially when you consider the complexity that comes with managing hundreds of microservices across different teams.

For this reason, approaches such as DevOps and CI/CD are better suited to ensure that the services are efficiently and effectively managed from design, through development and deployment.

Comparing containers & microservices

A post from Ev Kontsevoy summarized the comparison of these two terminologies in an interesting way:

"A container is a useful resource allocation and sharing technology. It's something DevOps people get excited about. A microservice is a software design pattern. It's something developers get excited about."

In other words, we can sum this up as:

- Microservices are about the design of software.
- Containers are about packaging software for deployment.

So, we can choose whether to use a container for hosting a microservice. But to get full value from both, it is significantly better to run microservices within containers.

Deploying an entire application to a single VM introduces a single point of failure risk, whether or not a microservice architecture has been used. But spreading the application through microservices across multiple containers results in fully exploiting the value of both by providing resilience as well as agility through scaling and improvements targeting specific services without negatively impacting the entire application.

Flexibility is also introduced in that developers can write applications in the language of their choice since the container will allow them to deploy across whatever environment is provided. Efficiency comes from containers using less resources compared to VMs.

An added benefit comes in the form of security through isolation and a broader attack surface that limits the impact should a single microservice or container be subject to a security breach such as a hacking attack.

Challenges with containers & microservices

The drawbacks that come with using containers and microservices are tied to the management overhead especially when dealing with large scale distributed deployments. This means that deployment, monitoring, and management of containers and microservices at such environments would be quite challenging and require specialized tools that can support orchestration and ensure consistency in deployment.

Other challenges include:

- Complexity from managing microservices written in different languages
- Cost implications of network resource usage from remote calls across multiple services
- Investigating root causes or auditing systems becomes challenging when dealing with log management across distributed services, as log aggregators would be required

Future of containers & microservices

The need for modern applications that provide both agility and resilience cannot be understated in the age of cloud and mobile apps. Benefits such as faster time to market and enhance security are value propositions that the business will gladly accept.

Containers and microservices are currently the preferred approach for scaling and refactoring legacy applications to make them cloud native.

Powered by Kubernetes and Docker as well as the growth of hybrid cloud deployments and edge computing, the market for these capabilities is expected to continue growing, with MarketWatch predicting a CAGR of 12.7% for the global cloud microservices market, reaching a value of $1.7 billion by 2027.

Related reading

- BMC DevOps Blog
- Kubernetes vs Docker Swarm: Comparing Container Orchestration Tools
- Microservices vs Nanoservices: Weighing Framework Options
- Containers as a Service (CaaS) Explained
- The State of Containers Today: A Report Summary
- The 12-Factor App Methodology Explained