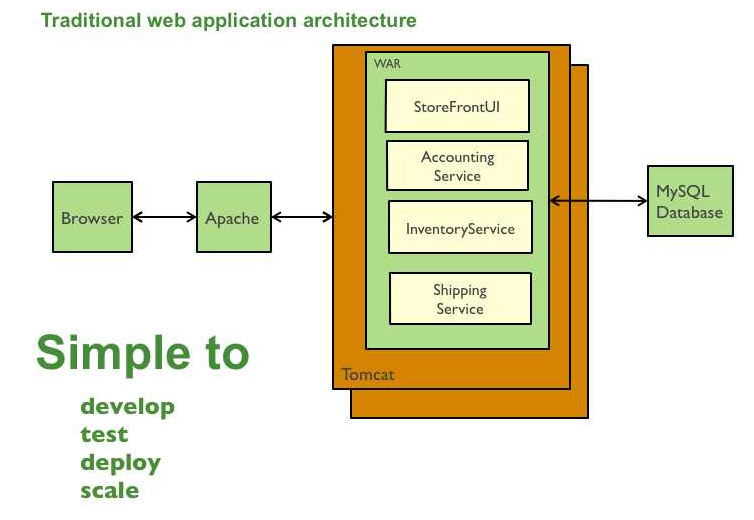
<http://cloudacademy.com/blog/microservices-architecture-challenge-advantage-drawback/>

Suppose I need to build a classic web application using Java. The first thing I will do is design a Presentation Layer (the user interface), then an Application Layer handling all business logic, an Integration Layer to enable loose coupling between various components of the Application Layer, and finally a Database Layer that will be accessible to the underlying persistent system.

Now in order to run the entire application I will create either a WAR or EAR package and deploy it on an application server (like JBoss, Tomcat, and WebLogic). Now, because I have packaged everything as an EAR/WAR, it becomes monolithic, which means that, even though we have separate and distinguishable components, all are packaged under one roof.

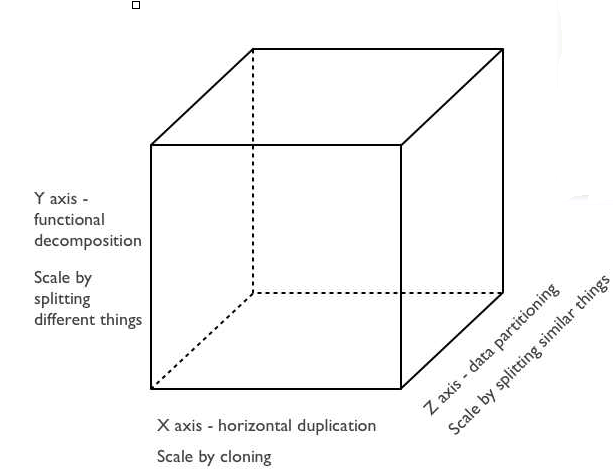


**Monolithic architectures: challenges**

* Scaling such a monolithic application can only be accomplished by deploying the same EAR/WAR packages in more servers – also known as horizontal scaling. Each copy of the application in various servers will utilize the same amount of underlying resources, which is often not an efficient way to design.

### Microservices explained

MicroService is about taking an application breaking them into smaller components or services and through api-gateway to orchestrate the smaller services.



As you can see, the X axis (API Gateway) represents horizontal application scaling (which we have seen is possible even with monolithic architecture), and the Z axis represents scaling the application by splitting similar things. The Z axis idea can be better understood by using the sharding concept, where data is partitioned and the application redirects requests to corresponding shards based on user input.

The Y axis is the one on which we’ll focus. This axis represents functional decomposition. In this kind of strategy, various functions can be seen as independent services. So, instead of deploying the entire application only once everyone is done, developers can deploy their respective services independently without waiting for the others to finish their modules. This not only improves developer time management, but also offers them much more flexibility to change and redeploy their modules without needing to worry about the rest of the application’s components. Compare this diagram with the earlier monolithic design:

### Microservices Deployment options and Virtualization

Now that we understand microservices – and particularly the fact that the greatest advantage is that they’re **not** deployed in integrated WAR-like packages – how **are** they deployed?

The best way to deploy microservices-based applications is inside containers. Containers are complete virtual operating system environments that provide processes with isolation and dedicated access to underlying hardware resources. The biggest name in container solutions right now, is [Docker](https://www.docker.com/).

Virtual machines from IaaS providers like AWS can also work well for microservices deployments, but relatively lightweight microservices packages may not leverage the whole VM, possibly reducing their cost effectiveness.

You can also deploy your code using an OSGI (Open Service Gateway Initiative)bundle. In this case, all of your services will be running under one JVM, but this comes with a management and isolation tradeoff.