Learning Outcome

Able to Build a web application on modern cloud-based architectures and services

# Container applications

## What are containers?

Containers are executable units of software in which application code is packaged, along with its libraries and dependencies, in common ways so that it can be run anywhere, whether it be on desktop, traditional IT, or the cloud.

To do this, containers take advantage of a form of operating system (OS) virtualization in which features of the OS (in the case of the Linux kernel, namely the namespaces and cgroups primitives) are leveraged to both isolate processes and control the amount of CPU, memory, and disk that those processes have access to.

Containers are small, fast, and portable because unlike a virtual machine, containers do not need include a guest OS in every instance and can, instead, simply leverage the features and resources of the host OS.

Containers first appeared decades ago with versions like FreeBSD Jails and AIX Workload Partitions, but most modern developers remember 2013 as the start of the modern container era with the introduction of Docker.

## Benefits of containers

The primary advantage of containers, especially compared to a VM, is providing a level of abstraction that makes them lightweight and portable.

**Lightweight:** Containers share the machine OS kernel, eliminating the need for a full OS instance per application and making container files small and easy on resources. Their smaller size, especially compared to virtual machines, means they can spin up quickly and better support cloud-native applications that scale horizontally.

**Portable and platform independent:** Containers carry all their dependencies with them, meaning that software can be written once and then run without needing to be re-configured across laptops, cloud, and on-premises computing environments.

**Supports modern development and architecture:** Due to a combination of their deployment portability/consistency across platforms and their small size, containers are an ideal fit for modern development and application patterns—such as DevOps, serverless, and microservices—that are built are regular code deployments in small increments.

**Improves utilization:** Like VMs before them, containers enable developers and operators to improve CPU and memory utilization of physical machines. Where containers go even further is that because they also enable microservice architectures, application components can be deployed and scaled more granularly, an attractive alternative to having to scale up an entire monolithic application because a single component is struggling with load.

## Use cases for containers

Containers are becoming increasingly prominent, especially in cloud environments. Many organizations are even considering containers as a replacement of VMs as the general purpose compute platform for their applications and workloads. But within that very broad scope, there are key use cases where containers are especially relevant.

**Microservices:** Containers are small and lightweight, which makes them a good match for microservice architectures where applications are constructed of many, loosely coupled and independently deployable smaller services.

**DevOps:** The combination of microservices as an architecture and containers as a platform is a common foundation for many teams that embrace DevOps as the way they build, ship and run software.

**Hybrid, multi-cloud:** Because containers can run consistently anywhere, across laptop, on-premises and cloud environments, they are an ideal underlying architecture for hybrid cloud and multicloud scenarios where organizations find themselves operating across a mix of multiple public clouds in combination with their own data center.

**Application modernizing and migration:** One of the most common approaches to application modernization starts by containerizing them so that they can be migrated to the cloud.

## Containerization

Software needs to be designed and packaged differently in order to take advantage of containers—a process commonly referred to as containerization.

When containerizing an application, the process includes packaging an application with its relevant environment variables, configuration files, libraries, and software dependencies. The result is a container image that can then be run on a container platform.

## Container orchestration with Kubernetes

As companies began embracing containers—often as part of modern, cloud-native architectures—the simplicity of the individual container began colliding with the complexity of managing hundreds (even thousands) of containers across a distributed system.

To address this challenge, container orchestration emerged as a way managing large volumes of containers throughout their lifecycle, including:

* Provisioning
* Redundancy
* Health monitoring
* Resource allocation
* Scaling and load balancing
* Moving between physical hosts

While many container orchestration platforms (such as Apache Mesos, Nomad, and Docker Swarm) were created to help address these challenges, Kubernetes, an open source project introduced by Google in 2014, quickly became the most popular container orchestration platform, and it is the one the majority of the industry has standardized on.

Kubernetes enables developers and operators to declare a desired state of their overall container environment through YAML files, and then Kubernetes does all the hard work establishing and maintaining that state, with activities that include deploying a specified number of instances of a given application or workload, rebooting that application if it fails, load balancing, auto-scaling, zero downtime deployments and more.

## Docker

Docker is a container management service. The keywords of Docker are develop, ship and run anywhere. The whole idea of Docker is for developers to easily develop applications, ship them into containers which can then be deployed anywhere.

### Features of Docker

* Docker has the ability to reduce the size of development by providing a smaller footprint of the operating system via containers.
* With containers, it becomes easier for teams across different units, such as development, QA and Operations to work seamlessly across applications.
* You can deploy Docker containers anywhere, on any physical and virtual machines and even on the cloud.
* Since Docker containers are pretty lightweight, they are very easily scalable.

### Components of Docker

Docker has the following components

* Docker for Mac − It allows one to run Docker containers on the Mac OS.
* Docker for Linux − It allows one to run Docker containers on the Linux OS.
* Docker for Windows − It allows one to run Docker containers on the Windows OS.
* Docker Engine − It is used for building Docker images and creating Docker containers.
* Docker Hub − This is the registry which is used to host various Docker images.
* Docker Compose − This is used to define applications using multiple Docker containers.

The official site for Docker is https://www.docker.com/ The site has all information and documentation about the Docker software.

### Docker – Hub

Docker Hub is a registry service on the cloud that allows you to download Docker images that are built by other communities. You can also upload your own Docker built images to Docker hub.

The official site for Docker hub is − <https://www.docker.com/community-edition#/add_ons>

### Docker – Images

In Docker, everything is based on Images. An image is a combination of a file system and parameters.

### Docker – Containers

Containers are instances of Docker images that can be run using the Docker run command. The basic purpose of Docker is to run containers.

### Docker – Container Lifecycle

The following illustration explains the entire lifecycle of a Docker container.

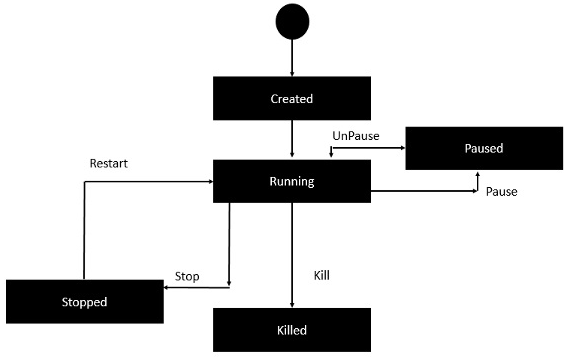


Image : Docker-Container Life Cycle

Reference: <https://www.tutorialspoint.com/docker/docker_working_with_containers.htm>

### Docker – Architecture

The following image shows the standard and traditional architecture of virtualization.

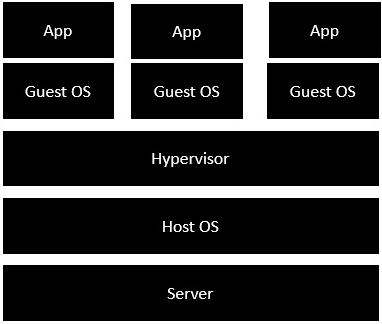


Image : Architecture of virtualization

Reference: <https://www.tutorialspoint.com/docker/docker_architecture.htm>

* The server is the physical server that is used to host multiple virtual machines.
* The Host OS is the base machine such as Linux or Windows.
* The Hypervisor is either VMWare or Windows Hyper V that is used to host virtual machines.
* You would then install multiple operating systems as virtual machines on top of the existing hypervisor as Guest OS.
* You would then host your applications on top of each Guest OS.

The following image shows the new generation of virtualization that is enabled via Dockers. Let’s have a look at the various layers.

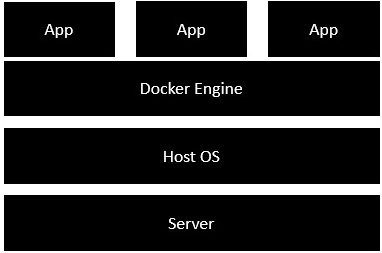


Image : Architecture of Docker

Reference: <https://www.tutorialspoint.com/docker/docker_architecture.htm>

* The server is the physical server that is used to host multiple virtual machines. So this layer remains the same.
* The Host OS is the base machine such as Linux or Windows. So this layer remains the same.
* Now comes the new generation which is the Docker engine. This is used to run the operating system which earlier used to be virtual machines as Docker containers.
* All of the Apps now run as Docker containers.

The clear advantage in this architecture is that you don’t need to have extra hardware for Guest OS. Everything works as Docker containers.

## Kubernetes

Kubernetes in an open source container management tool hosted by Cloud Native Computing Foundation (CNCF). This is also known as the enhanced version of Borg which was developed at Google to manage both long running processes and batch jobs, which was earlier handled by separate systems.

Kubernetes comes with a capability of automating deployment, scaling of application, and operations of application containers across clusters. It is capable of creating container centric infrastructure.

### Features of Kubernetes

Following are some of the important features of Kubernetes.

* Continues development, integration and deployment
* Containerized infrastructure
* Application-centric management
* Auto-scalable infrastructure
* Environment consistency across development testing and production
* Loosely coupled infrastructure, where each component can act as a separate unit
* Higher density of resource utilization
* Predictable infrastructure which is going to be created

One of the key components of Kubernetes is, it can run application on clusters of physical and virtual machine infrastructure. It also has the capability to run applications on cloud. It helps in moving from host-centric infrastructure to container-centric infrastructure.

## Kubernetes – Architecture

### Kubernetes - Cluster Architecture

As seen in the following diagram, Kubernetes follows client-server architecture. Wherein, we have master installed on one machine and the node on separate Linux machines.

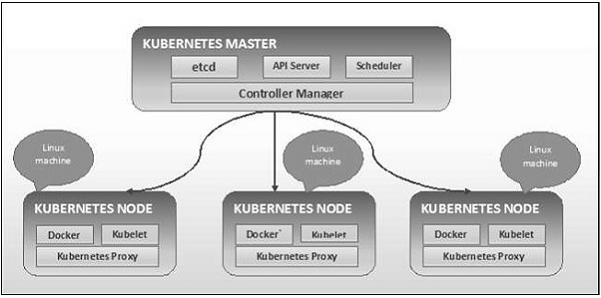


Image : Architecture of Kubernetes

Reference: <https://www.tutorialspoint.com/kubernetes/kubernetes_architecture.htm>

### Kubernetes - Master Machine Components

Following are the components of Kubernetes Master Machine.

**etcd**

It stores the configuration information which can be used by each of the nodes in the cluster. It is a high availability key value store that can be distributed among multiple nodes. It is accessible only by Kubernetes API server as it may have some sensitive information. It is a distributed key value Store which is accessible to all.

**API Server**

Kubernetes is an API server which provides all the operation on cluster using the API. API server implements an interface, which means different tools and libraries can readily communicate with it. Kubeconfig is a package along with the server side tools that can be used for communication. It exposes Kubernetes API.

**Controller Manager**

This component is responsible for most of the collectors that regulates the state of cluster and performs a task. In general, it can be considered as a daemon which runs in nonterminating loop and is responsible for collecting and sending information to API server. It works toward getting the shared state of cluster and then make changes to bring the current status of the server to the desired state. The key controllers are replication controller, endpoint controller, namespace controller, and service account controller. The controller manager runs different kind of controllers to handle nodes, endpoints, etc.

**Scheduler**

This is one of the key components of Kubernetes master. It is a service in master responsible for distributing the workload. It is responsible for tracking utilization of working load on cluster nodes and then placing the workload on which resources are available and accept the workload. In other words, this is the mechanism responsible for allocating pods to available nodes. The scheduler is responsible for workload utilization and allocating pod to new node.

### Kubernetes - Node Components

Following are the key components of Node server which are necessary to communicate with Kubernetes master.

**Docker**

The first requirement of each node is Docker which helps in running the encapsulated application containers in a relatively isolated but lightweight operating environment.

**Kubelet Service**

This is a small service in each node responsible for relaying information to and from control plane service. It interacts with etcd store to read configuration details and wright values. This communicates with the master component to receive commands and work. The kubelet process then assumes responsibility for maintaining the state of work and the node server. It manages network rules, port forwarding, etc.

**Kubernetes Proxy Service**

This is a proxy service which runs on each node and helps in making services available to the external host. It helps in forwarding the request to correct containers and is capable of performing primitive load balancing. It makes sure that the networking environment is predictable and accessible and at the same time it is isolated as well. It manages pods on node, volumes, secrets, creating new containers’ health checkup, etc.

### Kubernetes - Master and Node Structure

The following illustrations show the structure of Kubernetes Master and Node.

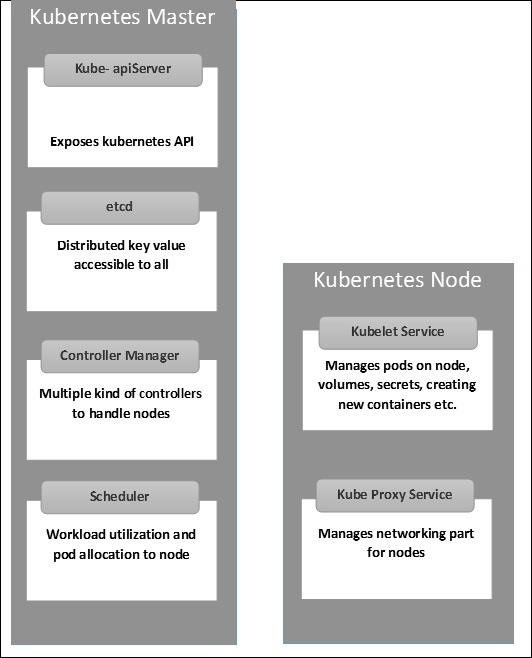


Image : Kubernetes - Master and Node Structure

Reference: <https://www.tutorialspoint.com/kubernetes/kubernetes_architecture.htm>

## Services in cloud supporting container applications

### IBM Cloud Kubernetes Service

With IBM Cloud Kubernetes Service, you can create your own Kubernetes cluster to deploy and manage containerized apps on IBM Cloud. Your containerized apps are hosted on IBM Cloud infrastructure compute hosts that are called worker nodes. You can choose to provision your compute hosts as virtual machines with shared or dedicated resources, or as bare metal machines that can be optimized for GPU and software-defined storage (SDS) usage. Your worker nodes are controlled by a highly available Kubernetes master that is configured, monitored, and managed by IBM. You can use the IBM Cloud Kubernetes Service API or CLI to work with your cluster infrastructure resources and the Kubernetes API or CLI to manage your deployments and services.

**Why should I use IBM Cloud Kubernetes Service?**

IBM Cloud Kubernetes Service is a managed Kubernetes offering that delivers powerful tools, an intuitive user experience, and built-in security for rapid delivery of apps that you can bind to cloud services that are related to IBM Watson™, AI, IoT, DevOps, security, and data analytics. As a certified Kubernetes provider, IBM Cloud Kubernetes Service provides intelligent scheduling, self-healing, horizontal scaling, service discovery and load balancing, automated rollouts and rollbacks, and secret and configuration management. The service also has advanced capabilities around simplified cluster management, container security and isolation policies, the ability to design your own cluster, and integrated operational tools for consistency in deployment.

### Red Hat Openshift service on IBM Cloud

OpenShift Container Platform is a private platform-as-a-service (PaaS) for enterprises that run OpenShift on public cloud or on-premises infrastructure. It runs on the Red Hat Enterprise Linux (RHEL) operating system and functions as a set of Docker-based application containers managed with Kubernetes orchestration.

### Amazon ECS on AWS Cloud

Amazon ECS is a fully managed container orchestration service that helps you easily deploy, manage, and scale containerized applications. It deeply integrates with the rest of the AWS platform to provide a secure and easy-to-use solution for running container workloads in the cloud and now on your infrastructure with Amazon ECS Anywhere.

## Hybrid Application scenarios

Legacy modernization is a priority in today’s digital business environment. With rapid technological advancements, increased customer demand for digitized services, and the growing popularity of DevOps and agile approaches — rigid monolithic architectures are no longer sufficient.

What’s the solution? Companies have tried several different approaches to modernizing their existing legacy systems, but each one proved to be sub-optimal.

OpenLegacyDismantling and replacing legacy systems is time-consuming, tedious, and risky. Leaving them intact and adding layers of middleware to get the job done results in bloated code and often leads to complex maintenance issues. It’s also a huge roadblock preventing organizations from adopting modern DevOps practices.

Hybrid IT allows organizations to abstract the choice of the development environment and leverage the possibilities of microservices and cloud-native architecture to integrate their legacy systems.

Hybrid integration allows you to embrace technological innovations and deploy services in the cloud, as well as integrate the data between on-premise and cloud systems. This solves the problem of legacy modernization by allowing you to connect your existing IT infrastructure with modern systems and technologies, mobile and cloud applications, and efficiently develop and deploy digital services to meet the demands of your customers.

The concrete usage scenarios of a hybrid cloud infrastructure are as follows:

**1: Safe harbor: private environment as gateway to the cloud world**

For companies, a hybrid cloud solution can serve as a gateway to the cloud world. The private cloud environment functions as a protected space for first steps and safe testing until the desired solution runs stably on cloud servers – in a second step it can be ported promptly into the public cloud if required. What is important here, however, is a hybrid cloud infrastructure with a uniform software and hardware basis to ensure maximum compatibility between the public and private clouds.

A suitable starting point could be legacy applications, for example: T-Systems' transformation projects show that around two thirds of business applications can basically be transferred to the cloud. This requires a thorough analysis and inventory of the existing IT landscape. T-Systems, for example, offers the so-called Cloudifier with standardized transformation services for entry and transition to the cloud.

**2: Burst scenario – the public cloud as a buffer for peak loads**

Whether it’s the busy Christmas period in e-commerce, data-intensive simulations in product development or Big Data analyses in research: Many companies only need high-performance computing capacities from time to time rather than permanently. This is a case for the hybrid cloud, in which private and public clouds can be smoothly combined to form what is known as cloud bursting. This enables companies to flexibly absorb peak loads at any time by switching storage and computing resources on and off as required.

However, some companies need to store and process certain data in their own data center or in a private cloud. For example, to ensure they are protected against industrial espionage, to adhere to compliance regulations or to benefit from the lowest possible latencies. But the public cloud can also be an overflow basin for such companies: "It depends on how such scenarios are implemented in detail. If business-critical data always remains in the private environment and only the compute service comes from the public cloud, cloud bursting can also be feasible for sensitive company data," says Sascha Smets, Senior Product Manager at T-Systems. “This can be achieved either by anonymizing the data that is to be processed in the public cloud. Or by only providing applications in the public cloud with the exact information they need for a specific computing process. This makes the interaction secure and also saves bandwidth."

**3: Backup and disaster recovery with the hybrid cloud**

Hybrid cloud scenarios can also be used as backup or disaster recovery solutions. There are several possibilities for this kind of implementation – depending on the requirements of the respective company. On the one hand, the public cloud is an inexpensive long-term storage solution. For maximum security, for example, data can be stored in encrypted form in the Object Based Storage.

Companies that do not want to store certain data in the public cloud but still want to store it redundantly can also set up two private, separate availability zones (AZs) in which they mirror their systems. The distance between the AZs is important. They should be far enough apart so that both AZs don’t fail simultaneously in the event of a possible disaster such as a flood or fire. But close enough to benefit from the lowest possible latencies. "A benchmark that has become established among companies is a distance between the data centers of around 20 to 30 kilometers," says Sascha Smets.

**4: Application development with DevOps – develop once, run anywhere**

Modern, agile application development increasingly integrates development, testing and operation. While the respective teams used to work independently and separately from each other, today they are linked by cooperation in a DevOps model. Hybrid cloud platforms enable teams to develop software faster and to shorten release procedures. Applications can be ported as needed between teams and their respective private or public environments.

What is important here is a hybrid cloud infrastructure that ensures a seamless transition between the public and private environments. The Open Telekom Cloud Hybrid, for example, offers DevOps teams this kind of unified environment, which is based on the same hardware and software for public and private clouds. Each developed application can run in both the public and private environments – in line with the motto: develop once, run anywhere. And this is true even if the company in question doesn’t work with container technology.

**5: Real-time data processing – minimum latency, maximum power**

Edge Computing and the hybrid cloud cultivate a close relationship, because even with Edge Computing, companies can make use of decentralized computing and storage capacities as required. However, these are not located in a remote data center, but close to the action, at the edge of a network – hence the term "Edge Computing." Why is this necessary? One example is the real-time processing of data, where latencies can be kept as low as possible. For example, the transmission and processing of sensor data for autonomous driving practically require a data center at every intersection. The situation is similar for the control of industrial robots with AI algorithms, because low latencies also play a central role here.

"There is a growing demand for IT resources that can handle processes with virtually no latency," says Sascha Smets. "We will soon meet this demand with our new Edge Cloud offering: mini data centers for real-time applications based on the Open Telecom Cloud technology, which we can install and operate directly on our customers' premises if required.”

**6: IT departments as service providers and cloud brokers**

The so-called shadow IT is flourishing – to the chagrin of many companies. According to Forrester Research, almost half of employees in companies now use technologies without the knowledge of their IT departments. Hybrid cloud solutions offer IT departments the opportunity to change things by building a unified service catalog and establishing themselves as the company’s cloud brokers.

If the private and public parts are based on the same technology, the IT department will find it easier to build this type of unified service catalog. For applications with sensitive data that have to stay in the company, IT can then offer virtual machines with appropriate specifications. For less critical workloads that can operate in a public cloud, IT offers a VM with exactly the same specification – but at significantly lower cost. This means that users in specialist departments can select the services they need from private or public instances at the click of a mouse, without neglecting requirements such as scalability, security and governance.

**7: Provide remote locations without network infrastructure with the cloud**

Whether it be an oil rig, space station or remote research facility: Not every location can be easily supplied with a fast Internet connection. With the Open Telekom Cloud Hybrid, companies can also use IT resources in remote locations as a private cloud – regardless of the network connection. Telekom can implement the necessary servers directly where they are needed. A connection to the Internet or the public cloud is not absolutely necessary for operating them.

**8: Outlook: Using thermal energy from data centers sensibly**

Servers give off a lot of heat: To ensure an optimum operation, the hardware in data centers is usually cooled. This results in double the energy requirements – once for server operation and then again for cooling. But it’s also possible to make sensible use of the thermal energy from servers. For example, supplying buildings with hot water and heating energy. In this way, companies not only save on the electricity for cooling the server cabinets, but also heating costs and they use their resources in a worthy and sustainable way. If the number of hybrid cloud architectures in companies doubles by 2021, more and more data centers will emerge in the future that are suitable for decentralized heat supply.

## Hybrid architectures and best practices

**Four Architecture Choices for Application Development in the Digital Age**

Increasingly, businesses are going through a digital transformation journey to meet evolving consumer needs. Customers are also more and more likely to be using social networks, mobile applications, and digital technologies. Due to this change, digital strategy is now an integral part of the overall business strategy.

Many enterprises are obtaining computing power through cloud services platforms via the internet and adopting a cloud-first strategy for most application development. This has furthered a change in application design—previously, functionality and statefulness were prioritized, but now most consumer-facing applications are moving to Software-as-a-Service (SaaS) and digital platforms. The application design focus is now much more focused on user experience, statelessness, and agility.

Choosing the right application architecture depends on your business requirements. In this post, we will examine four architecture choices for enabling digital transformation, depending on general business needs.

**Traditional 3-tier application architecture**

We all know about the 3-tier application architecture—it is a client-server architecture with a typical structure consisting of the presentation layer, application layer, and database layer.

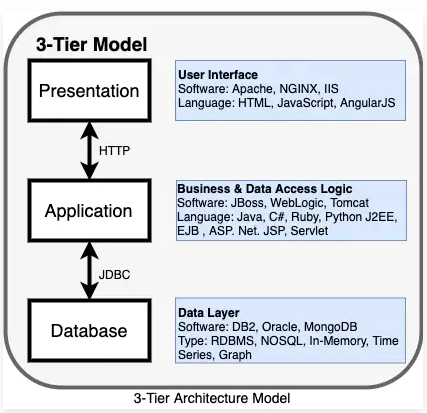


Image : 3 tier Architecture model

Reference: <https://www.ibm.com/cloud/blog/four-architecture-choices-for-application-development>

It has a user interface, business/data access logic, and data access. Many enterprise applications were created using the simple 3-tier application architecture.

**What is the issue with 3-tier application architecture?**

Simply speaking, the 3-tier application model is outdated. It was designed for application development before the proliferation of public cloud and mobile applications and has had difficulty adapting to the cloud.

Over time, an application can become too large and complex to make frequent changes. Not only that, but it also requires the maintenance of at least three layers of hardware and software, which can be inefficient for the business.

The 3-tier application model is also frequently called a monolithic architecture. These days, we have multiple new architecture models, and below, we will examine a few that are available now in the cloud era.

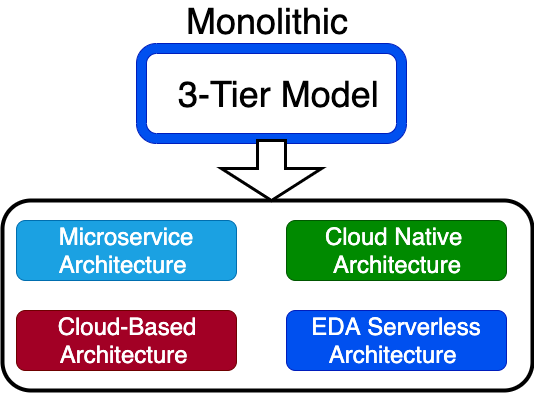


Image : Monolithic 3 tier model

Reference: <https://www.ibm.com/cloud/blog/four-architecture-choices-for-application-development>

1. **Microservices architecture**

In a cloud model, complex applications designed as a collection of services and data are fully decoupled from the application. Microservices are an architectural style that structures the application as a collection of services. Each service can be written in a different programming language and tested separately. They are independently deployable and organized around business capabilities.

Take the example of an e-commerce application developed using microservices architecture. Each microservice can focus on a single business capability (e.g., shopping cart, search, customer review). Each of these can be a separate service written in different programming languages, deployed in different infrastructure, and managed by different teams.

Each service communicates with the others using a lightweight protocol. For a 3-tier, we all know about the Model View Controller (MVC) framework. Sidecar, Ambassador, and Adapter are some of the frameworks that support microservices architectures.

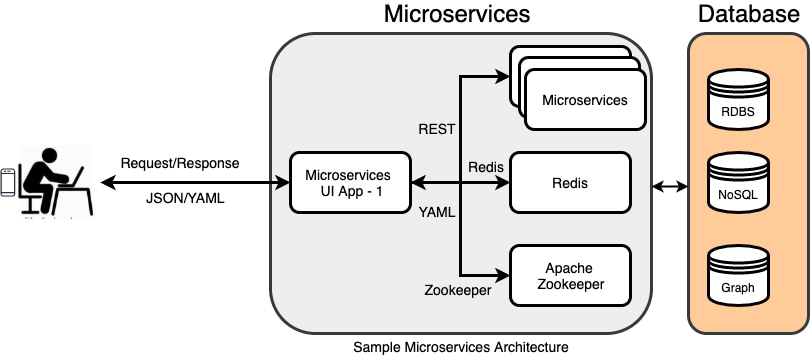


Image : Sample microservices architecture

Reference: <https://www.ibm.com/cloud/blog/four-architecture-choices-for-application-development>

**Microservices architecture vs. monolithic architecture**

In monolithic architecture, all these components coexist as a single module managed (mostly) by a single team—everything is bundled together. If you need to update, you need to deploy the entire application, and this slows down changes for larger complex applications. For smaller applications, monolithic architecture is often the best solution.

**Microservices, containers, and Kubernetes**

One of the best choices for creating and running microservices application architectures is by using containers. Containers encapsulate a lightweight virtualization runtime environment for your application and allow you to move the application from the developer's desktop all the way to production deployment. You can run containers on virtual machines or physical machines in the majority of available operating systems. Containers present a consistent software environment, and you can encapsulate all dependencies of your application as a deployable unit. Containers can run on a laptop, bare metal server, or in a public cloud.

Many organizations use Kubernetes to manage containers and ensure that there is no downtime. Kubernetes provides container orchestration in multiple hosts and is used for container lifecycle management. You can automate deployment, auto-scale your application, and build fast and ship fast using Kubernetes.

Red Hat OpenShift is one of the most popular leading hybrid cloud enterprise container platforms. Many public cloud providers offer Containers-as-a-Service (CaaS). Some of the other Kubernetes engines available are IBM Cloud Kubernetes Service, open source Kubernetes, AWS (EKS, ECS, and Fargate), Google GKS, and Azure AKS.

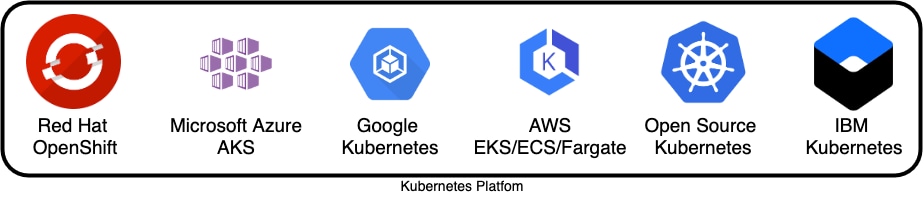


Image : Kubernetes platform

Reference: <https://www.ibm.com/cloud/blog/four-architecture-choices-for-application-development>

Usually each microservice is built by a different small team, and they choose their programming language and deployment schedule. A service mesh like Istio is used by enterprises to govern communication between microservices and management. In a service mesh, requests are routed through proxies (such as Sidecar) between microservices.

2. **Cloud native architecture**

Cloud native architecture is designed specifically for applications planning to deploy in the cloud, and microservices are a critical part.

Cloud native is an approach to building and running applications that exploits the advantages of the cloud computing delivery model. Cloud native is a term used to describe container-based environments, and it is about how applications are created and deployed, not where.

Cloud native technologies empower us to run applications in public, private, and hybrid clouds. Cloud native development is essential to getting applications to market quickly; it helps people, processes, and technologies to build, deploy, and manage apps that are ready for the cloud.

The cloud native architecture model uses DevOps, continuous integration (CI), continuous delivery (CD), microservices, and containers. Most of the enterprises use the twelve-factor methodology for designing scalable and robust cloud native applications.

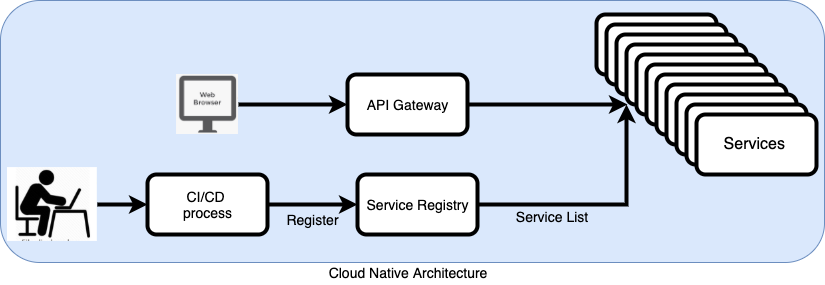


Image : Cloud Native architecture

Reference: <https://www.ibm.com/cloud/blog/four-architecture-choices-for-application-development>

In the cloud, applications must be able to run concurrently in multiple nodes, share a configuration/session state, have a centralized logging mechanism, and be able to deploy using DevOps and a CI/CD process. Many cloud providers give guidelines for cloud native development—Amazon Web Services (AWS) has its well-architected framework, Google has various guides on how to build cloud native applications, and Microsoft Azure has its cloud patterns guide.

Usually, cloud native applications are stateless by nature. The services communicate with each other using REST-based protocols or messaging. The API Gateway, container registry, message-oriented middleware (MOM: Publish/Subscribe or Request/Response), service mesh, and orchestrations could be part of cloud native architecture.

3. **Event-driven serverless architecture**

Event-driven architecture (EDA) is based on decoupled systems that run in response to events. An event-driven architecture uses events to trigger and communicate between decoupled services. EDA has been here for a long time, but it now has more relevance in the cloud.

So, what is new? If properly used, it can provide a significant increase in agility, cost savings, and operational benefits. The distributed serverless EDA can execute code known as functions that scale automatically in response to a REST API or an event trigger.

For the serverless model, there is no server management needed. The serverless model is also quickly scalable (so quick updates and deployment are possible) and it is stateless.

Here are some of the currently available cloud serverless services from different cloud providers:

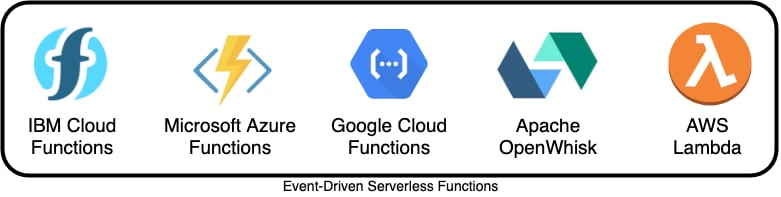


Image : Event-Driven Serverless Functions

Reference: <https://www.ibm.com/cloud/blog/four-architecture-choices-for-application-development>

**Types of serverless**

Functions-as-a-Service (FaaS): Upload pieces of functionality to the cloud and let these pieces be executed independently.

Backend-as-a-Service (BaaS): Utilize services from a third party, such as application management, database management, and cloud storage.

Mobile-Backend-as-a-Service (MBaaS): Functions for mobile applications.

4. **Cloud-based architecture**

How can we make monolithic applications work well in a cloud environment? Cloud-based architecture is best suited for building a modern web application (static/dynamic websites), deploying a web application, connecting to a database, and analyzing user behavior.

A traditional cloud-based application architecture involves load balancers, web servers, application servers, and databases. It can benefit from cloud features such as resource elasticity, software-defined networking, auto-provisioning, high availability, and scalability.

This type of architecture is ideal for organizations that don't have to worry about maintaining a server. The serverless functions support different programming languages, such as PHP, Java, .NET, Node.js, Python, Ruby, Docker, and Go.

API Gateway is an important service that makes it easy for developers to create and publish secure APIs. The APIs will act as a front door for applications to access data and business logic. It also takes care of authorization and access control. Developers use API Gateway to invoke different serverless functions for different API calls.

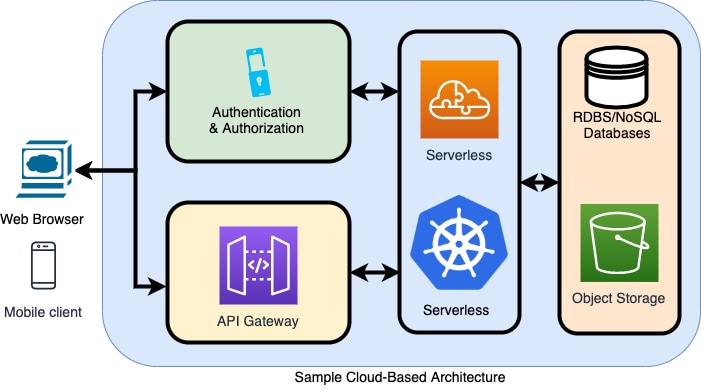


Image : Sample Cloud Based architecture

Reference: <https://www.ibm.com/cloud/blog/four-architecture-choices-for-application-development>

**How to decide which architecture model is best for your application?**

The decision you make when choosing an architecture model can influence the success or failure of your project. You should make your choice based on your application and on non-functional requirements.

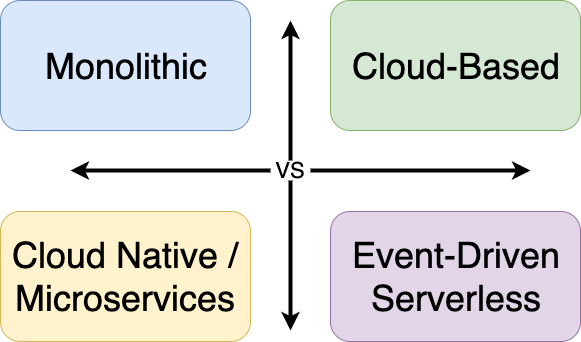


Image : Sample Cloud Based architecture

Reference: <https://www.ibm.com/cloud/blog/four-architecture-choices-for-application-development>

Consider the following before choosing an architecture for your app project:

* Is it monolithic-first or microservice-first? (For smaller projects with a simple application requirement, monolithic may be a right choice.)
* Is your team ready to utilize microservices?
* Does your team have an existing cloud-based DevOps and CI/CD process?
* What is your hosting model? Private, public, hybrid?
* How does the application architecture affect your project?
* Does a combination of multiple architecture model work for you?
* Do you need persistence and sessions for your applications?

References

1. <https://www.ibm.com/cloud/learn/containers>
2. <https://www.tutorialspoint.com/docker/>
3. <https://cloud.netapp.com/blog/cvo-blg-understanding-red-hat-openshift-container-platform>
4. <https://www.openlegacy.com/blog/hybrid-cloud>