### What is a container?

A container is a sandboxed process running on a host machine that is isolated from all other processes running on that host machine. That isolation leverages kernel namespaces and cgroups, features that have been in Linux for a long time. Docker makes these capabilities approachable and easy to use.

### Container Orchestrators

Container orchestration software allows developers to deploy large numbers of containers and manage them at large scale, using the concept of container clusters. Orchestrators help IT admins automate the process of running container instances, provisioning hosts, and connecting containers into functional groups.

With container orchestration, it is possible to manage the lifecycle of applications or ecosystems of applications consisting of large numbers of containers. Orchestrators can:

* Automatically deploy containers based on policies, application load and environment metrics
* Identify failed containers or clusters and heal them
* Manage application configuration
* Connect containers to storage and manage networking
* Improve security by restricting access in between containers, and between containers and external systems

Examples of orchestrators include Kubernetes, Mirantis Kubernetes Engine, and OpenShift.

### Managed Kubernetes Services

Managed Kubernetes services add another level of management above container orchestrators. Setting up and managing a tool like Kubernetes is challenging and requires specialized expertise.

These services allow organizations to provide container images and high-level scaling and operation policies, and automatically creates Kubernetes clusters. Clusters can be managed via APIs, web-based consoles, or CLI commands. Managed Kubernetes  is commonly offered on the public cloud, but there are platforms that can run in an on-premises data center as well.

Examples of managed Kubernetes services are Amazon Elastic Kubernetes Service (EKS),Google Kubernetes Engine, Azure Kubernetes Service (AKS)

### Advantages of a Containerized Architecture

Containerization is considered the next evolution of virtualization, which focuses on breaking down operating systems and applications into pieces of software that delivers certain functionality. Because the code is broken down into functions and packaged individually, it becomes more efficient and portable. Here are some benefits of implementing a containerized architecture:

1. **Lower costs**—on infrastructure operations, because you can run many containers on a single virtual machine.
2. **Scalability—**at the micro-service level eliminates the need to scale VMs or instances.
3. **Instant replication**—of microservices, enabled through deployment sets and replicas.
4. **Flexible routing**—you can set this up between services supported natively by containerization platforms.
5. **Resilience**—when a container fails, it’s easy to refresh/redeploy with a new container from the same image.
6. **Full portability**—between on-premise locations and cloud environments.
7. **OS independent—**there is no need to run an OS. All you need is to deploy a container engine on top of a host OS.
8. **Fast deployment—**of new containers. You can also quickly terminate old containers using the same environment.
9. **Lightweight—**since containers run without an OS, they are significantly lightweight and much less demanding than images.
10. **Faster “ready to compute”**—you can start and stop containers within seconds—much faster than VMs.

### Containers and the Microservices Architecture

A microservices architecture divides the application into multiple, independent services, each of which is developed and maintained by a small team. Each has its own CI/CD pipeline, and can be deployed to production at any time, without dependence on other microservices.

A common way to package and deploy microservices is in containers. The entire microservices application can be deployed as a cluster using a container orchestrator. There are several advantages to using containers for microservices, as opposed to full virtual machines or bare metal servers:

* Containers are lightweight, making it possible to run more microservice instances on one physical host.
* Containers can be easily automated, integrating closely with CI/CD workflows.
* Containers are immutable, making it easy to tear down and replace microservice instances when new versions are released.
* Containers are easily portable between local development environments, on-premise data centers and cloud environments, making it possible to develop microservices in one environment and deploy to another.

### Container-Based Application Design

Here is an overview of key design principles to help you create an effective container-based application architecture:

* **Observability**—required to ensure runtime environments can observe the health of containers. Observability enables you to automate the lifecycle of your containers. The minimum requirement for observability is application programming interfaces (APIs) that let the runtime perform health checks. You should also configure event logs.
* **Image immutability**—containers are built for temporary use. You cannot make changes after deployment. Instead, you need to build a new container image, and then deploy a new container version based on that image. You should then stop the old container, because it is no longer needed. You can automate this process, using orchestrators.
* **Disposability**—a major advantage of containerization is the ability to quickly scale, fix, shut down, and launch your application or components of the code. This capability lets you quickly deploy patches and handle sudden changes and capacity demands. To ensure your application can do this, you should use small containers that are reading to be quickly  
  disposed of.
* **Security**—to protect your cloud-native application, you need to establish container security practices and processes. You can do this by manually configuring security, but be sure to add automation that can support your workload. Minimal security requirements include using trustworthy images, managing access, integrating security testing tools, and adding security scans and controls to automated deployments.