There are 7 Containerization Technology are their , which are listed below :

**1.Docker .**

**2.RKT.**

**3.LXC .**

**4.Cri-O.**

**5.Podman.**

**6.RUNC.**

**7.ContainerD.**  
  
  
**1.Docker :**  
  
**Features :-** Simplifies container management with an intuitive CLI and API.

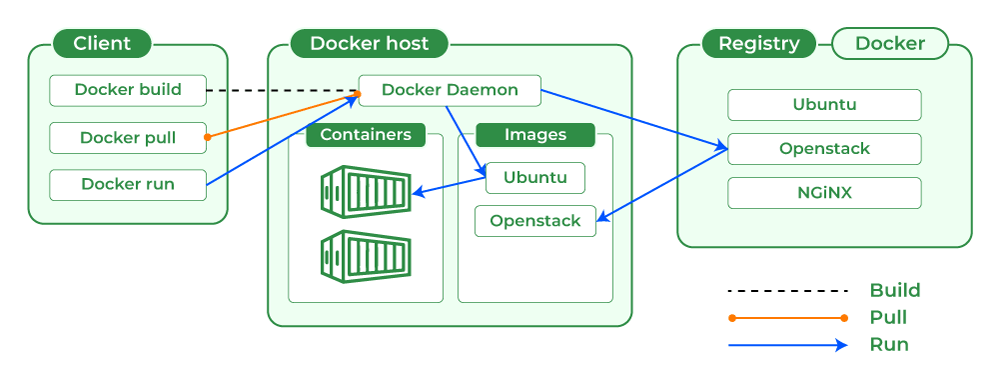
 Multi-stage builds for efficient image creation.

 Centralized container image repository via Docker Hub.

 Supports orchestration using Docker Swarm and Kubernetes.

 Rich plugin ecosystem for networking and storage.  
  
**Architecture :-**

Docker uses a **client-server model**. The Docker client interacts with the Docker daemon, which manages the containers. The daemon uses containerd and runc to execute containers.



**Market Share**: 80%

**Working Model:**

1. **Build**: Create container images using a Dockerfile.
2. **Ship**: Push images to a registry (Docker Hub, private registries).
3. **Run**: Deploy containers across different environments.

**2.RKT :**  
**Features** :-  
  
 Security-first design with SELinux, gVisor, and isolation capabilities.

 Daemonless operation, reducing overhead and simplifying integration.

 Fully compliant with Open Container Initiative (OCI) standards.

 Systemd integration for process management.

**Architecture** :-

RKT operates without a centralized daemon. It uses a single-binary execution model and relies on Linux process management tools like systemd.

**Market Share**: 2% (declining after Red Hat acquisition).

**Working Model:**

1. Pull OCI-compliant images from a registry.
2. Launch containers as systemd-managed processes.

**3.LXC :**  
  
 **Features** :-

 Provides OS-level virtualization using Linux kernel namespaces and cgroups.

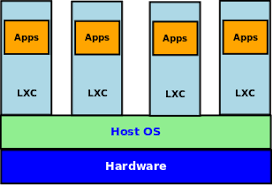
 Lightweight and resource-efficient.

 Supports running an entire Linux distribution within a container.

 CLI tools for easy management of containers.

**Architecture** :-

LXC containers directly leverage kernel-level features to isolate processes. Each container is a process within the host operating system.



**Market Share**: 5% .

**Working Model:**

1. Containers are created and managed using LXC tools.
2. Each container operates as an isolated Linux environment.

**4.Cri-O :**  
  
**Features :-**  
 Native container runtime for Kubernetes using the CRI (Container Runtime Interface).

 Lightweight and efficient for Kubernetes deployments.

 Fully supports OCI-compliant container images.

 Integrates seamlessly with Kubernetes orchestration.

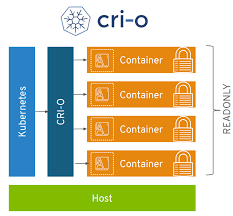
**Architecture :-**

CRI-O acts as a bridge between Kubernetes and low-level container runtimes like runc.

**Market Share**: 4% .

**Working Model:**

1. Kubernetes schedules a container via the CRI.
2. CRI-O uses runc to start and manage the container.



**5.Podman** :  
  
 **Features** :-  
  
 Rootless container management for enhanced security.

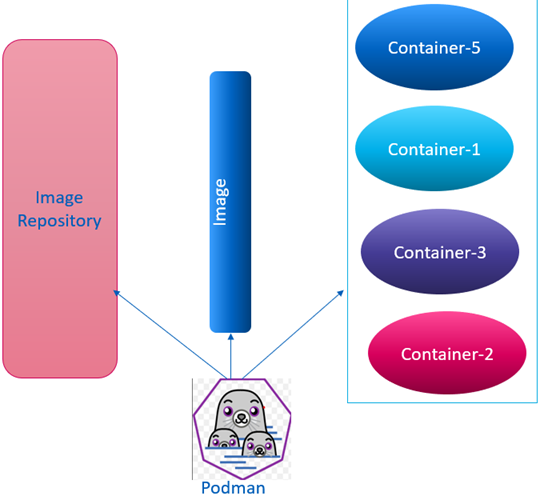
 No central daemon; containers run as individual processes.

 Docker CLI compatible, simplifying migration.

 Built-in support for container pods and networking.

**Architecture** :-

Podman uses a process-based model instead of a centralized daemon.



**Market Share**: 5% (increasing).

**Working Model:**

1. Create and manage containers using Podman CLI.
2. Containers run as isolated processes under the user account.

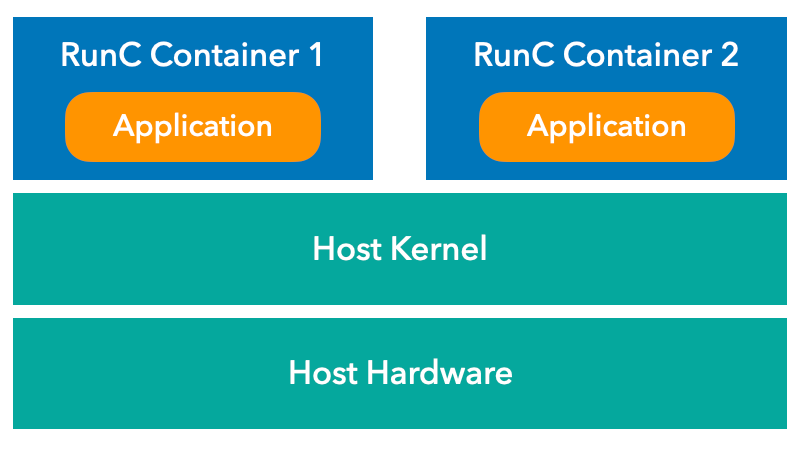
**6.RUNC** :  
  
**Features** :-  
  
 Low-level runtime for creating and running OCI-compliant containers.

 Lightweight and portable with minimal dependencies.

 Direct interaction with the Linux kernel for container execution.

 Highly modular for integration into container platforms like Docker and Kubernetes.

**Architecture** :-  
  
runc interacts directly with the Linux kernel to execute containers.



**Market Share**: 1% (used as a component in larger platforms).

**Working Model:**

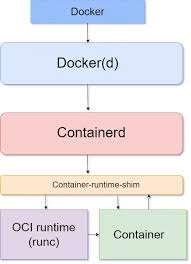
1. Reads container specifications from an OCI JSON file.
2. Creates and runs containers via kernel interfaces.

**7.ContainerD** :  
  
**Features :-**  
  
 API-driven container runtime with robust support for container lifecycle management.

 Manages container images, storage, and networking.

 Acts as a backend runtime for Docker and Kubernetes.

 Fully compliant with OCI container image and runtime standards.  
  
**Architecture** :-

Containerd is a daemon-based runtime that interfaces with low-level runtimes like runc.  
  
  
  
**Market Share**: 8%  
  
**Working Model:**

1. Manages container lifecycle and storage.
2. Acts as a backend runtime for Kubernetes and Docker.

**Comparison Table :**

| **Technology** | **Features** | **Architecture** | **Market Share (%)** | **Working Model** | **Primary Use Cases** |
| --- | --- | --- | --- | --- | --- |
| **Docker** | Rich ecosystem, ease of use | Client-server | 80% | Build, ship, run | Microservices, CI/CD |
| **RKT** | Security-focused | Single-binary, no daemon | 2% | Trusted container execution | High-security systems |
| **LXC** | OS-level virtualization | Kernel-based | 5% | Lightweight Linux OS containers | Hosting, multi-tenant apps |
| **CRI-O** | Kubernetes-native runtime | Daemon for OCI runtimes | 4% | Kubernetes container execution | Kubernetes |
| **Podman** | Rootless, daemonless | Daemonless | 5% | Secure container execution | Secure multi-user environments |
| **runc** | OCI-compliant runtime | Kernel interaction | 1% | Low-level container execution | Custom platforms, runtimes |
| **containerd** | Robust, API-driven | Daemon-based | 8% | API-based runtime management | Kubernetes, Docker integration |