

CS332-532

Systems Programming

Lab 8

Agenda

- Importance of pointers
- Linux in Practice
- Bash Scripting
- Containerization

My goal is to give a high level overview of why these are important
(None of this will be on the exam, and is purely practical for your future careers)

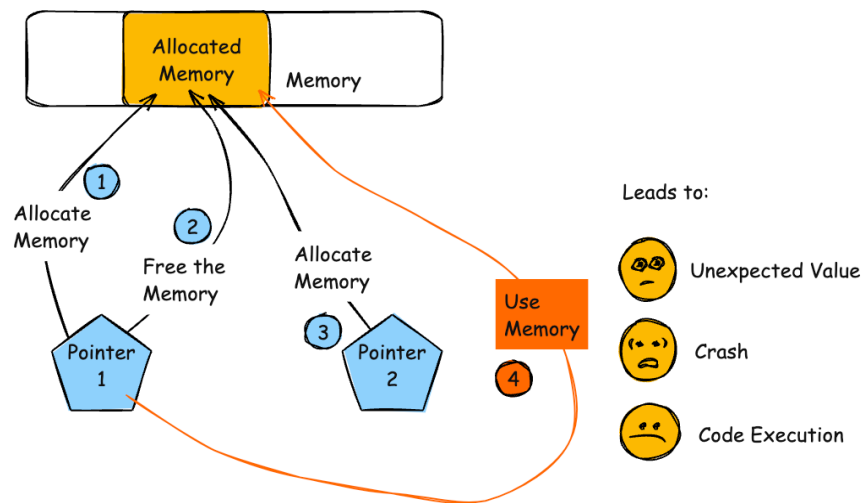
Importance of pointers

Cybersecurity Vocab

- **CVE – Common Vulnerability and Exposures**
 - A way to disclose and track security flaws with a unique identifier, e.g CVE-2017-0144. Every CVE follows this format: CVE-YYYY-NNNN
- **CVSS – Common Vulnerability Scoring System**
 - Framework for assessing the severity of vulnerabilities, using potential impact, exploitability, and risk.
 - Low, Medium, High, Critical, with a score ranging from 0.0-10.0
- **References:** <https://nvd.nist.gov/>, <https://cve.org>

Lessons from a recent attack

- On October 9th 2024 CVE-2024-9680 was discovered by researchers with a CVSS score of 9.8 (Critical)
- This was done using a "use-after-free" vulnerability in the Firefox web browser
- This flaw was found in Firefox's implementation in the CSS animation-timeline property

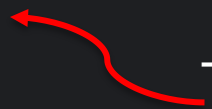


The problem

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      int *ptr = (int*)malloc( sizeof(int)); // Allocate memory
6      *ptr = 42;                               // Use memory
7
8      printf( format: "Value before free: %d\n", *ptr);
9
10     free(ptr);                               // Free the allocated memory
11     printf( format: "Memory freed\n");
12
13     // Use the freed memory (Use After Free vulnerability)
14     printf( format: "Value after free: %d\n", *ptr); // Undefined behavior, very dangerous!
15
16     return 0;
17 }
```

The problem

```
[red@legion test]$ gcc -o example example.c  
[red@legion test]$ ./example  
Value before free: 42  
Memory freed  
Value after free: -94983143  
[red@legion test]$
```



The memory is still allocated!

The solution

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      int *ptr = (int*)malloc( sizeof(int)); // Allocate memory
6      *ptr = 42;                               // Use memory
7
8      printf( format: "Value before free: %d\n", *ptr);
9
10     free(ptr);                               // Free the allocated memory
11     ptr = NULL;                             // Set pointer to NULL after freeing
12     printf( format: "Memory freed\n");
13
14     // Safely avoid using the pointer after free
15     if (ptr != NULL) {
16         printf( format: "Value after free: %d\n", *ptr); // Won't be executed
17     } else {
18         printf( format: "Pointer is NULL, memory cannot be accessed.\n"); // Safe handling
19     }
20
21     return 0;
22 }
```


The solution

```
[red@legion test]$ gcc -o examplefix examplefix.c
[red@legion test]$ ./examplefix
Value before free: 42
Memory freed
Pointer is NULL, memory cannot be accessed.
[red@legion test]$
```

Linux in Practice

Linux: Why do I care?

- Completely Open Source: <https://github.com/torvalds/linux>
- Widely used in Industry
 - It doesn't matter if you're a software engineer, DevOps, Sysadmin, or Cybersecurity, you will be using a Linux machine for some tasks
- Highly Customizable: Multiple distros, or create your own!
- Better Package Management
 - This is similar to installing things using pip for Python or npm for node.js. It automates and streamlines the process of managing software dependencies, which ensures consistency.
- Large Community
 - Extensive support, and learning resources to help you learn how it works

Linux: Distro Overview

- Red Hat Enterprise Linux (REHL): Enterprise focused distro designed for large systems; you pay for a license to use it. Very common in corporate environments
- Debian: Extremely Stable, large software repository, used in lots of servers, great for beginners
- Kali Linux: Used in penetration testing and cybersecurity for red teams
- Arch Linux: Popular for advanced daily users, rolling-release model
- Puppy Linux: Extremely lightweight, use for older hardware
- Tails: Security focused used for privacy and anonymity
- NixOS: Unique package manager known for reproducibility and rollback features
- Gentoo: Extremely customizable, compile packages from source. Gives complete control
- LFS: Linux from Scratch, you manually compile and install everything

Linux: Package Managers

Linux package managers are tools that help automate the process of installing, updating, and managing software on a Linux system by resolving dependencies and downloading packages from repositories.

Popular examples include:

- Debian based: apt, apt-get
- Arch based: pacman, yay, paru
- RHEL based: dnf && yum
- Nix based: nix
- Gentoo: portage

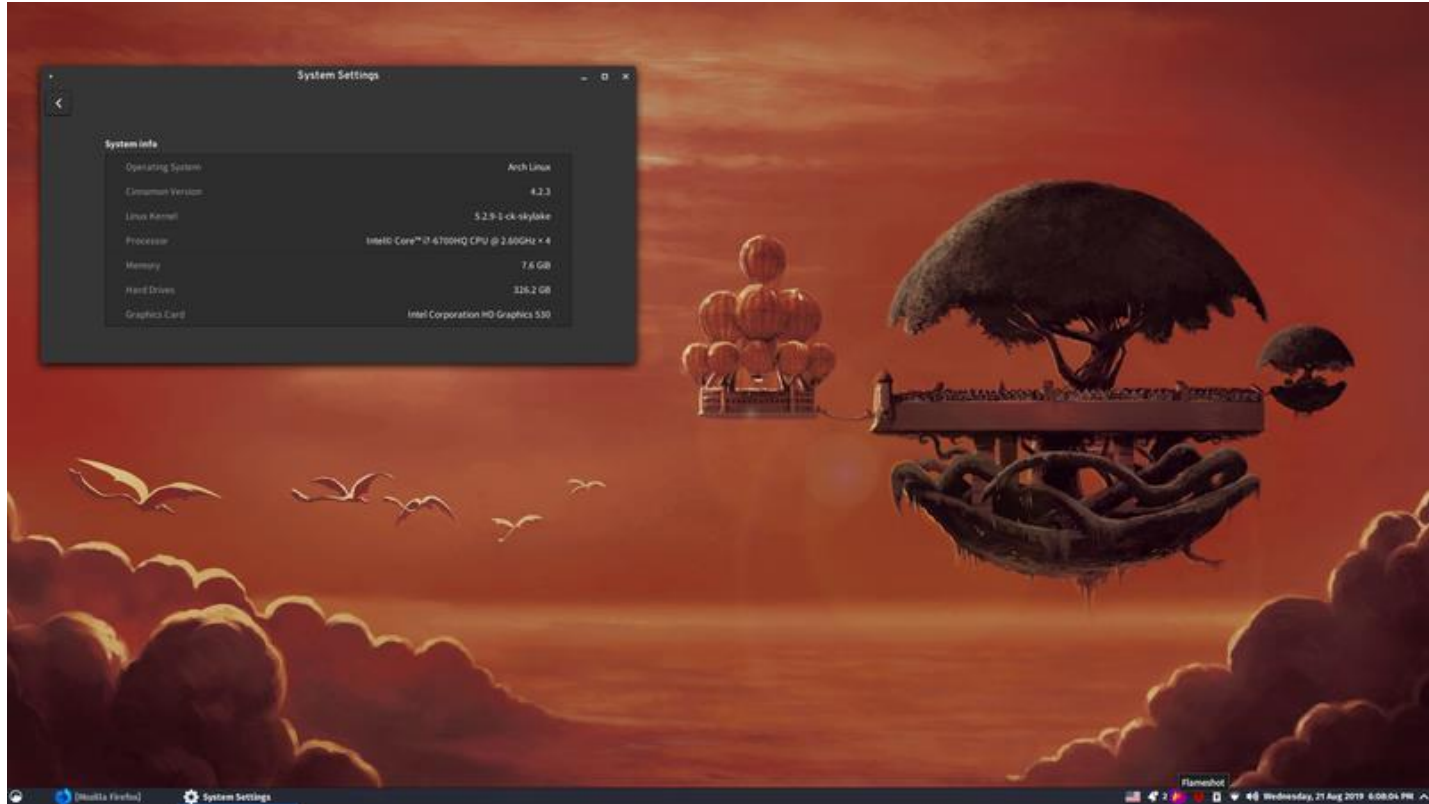
Linux: Desktop Environments

A **DE** (Desktop Environment) is a collection of software that provides a graphical user interface (GUI) on top of the underlying Linux operating system. It includes components like window managers, file managers, panels, icons, and widgets to create a user-friendly, visual way to interact with the system. You do not have to install a DE to use a distro; and some advanced distros do not come with a DE pre-installed.

The difference between the DE and Linux is:

- **Desktop Environment:** Provides the visual and interactive layer (windows, icons, menus) that users interact with. The DE sits on top of the Linux kernel and allows users to interact with the system without needing to rely on command-line interfaces.
- **Linux:** Refers to the kernel, the core part of the operating system that manages hardware, system processes, and resources. It doesn't include a graphical interface by default.

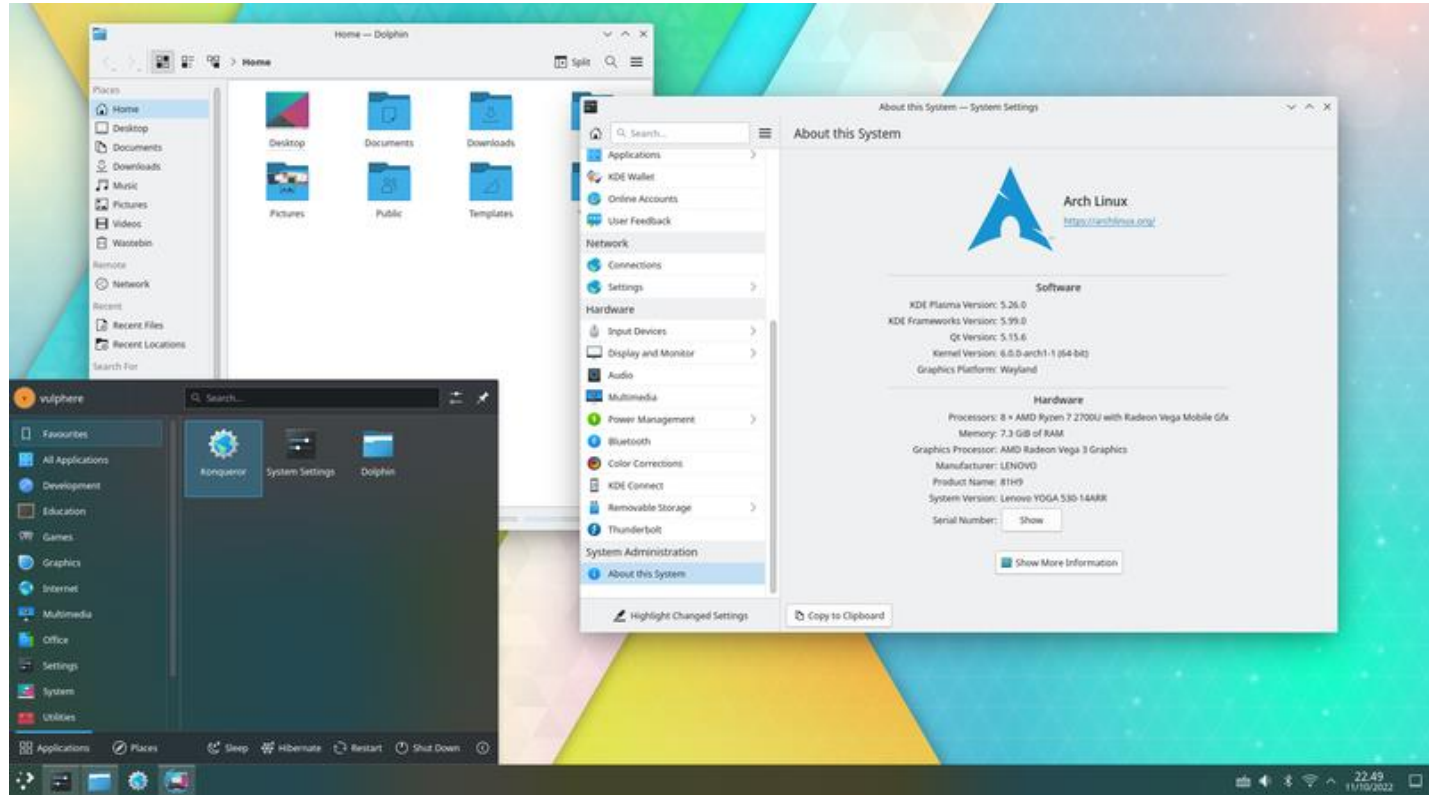
DE Examples: Cinnamon



DE Examples: GNOME



DE Examples: KDE Plasma



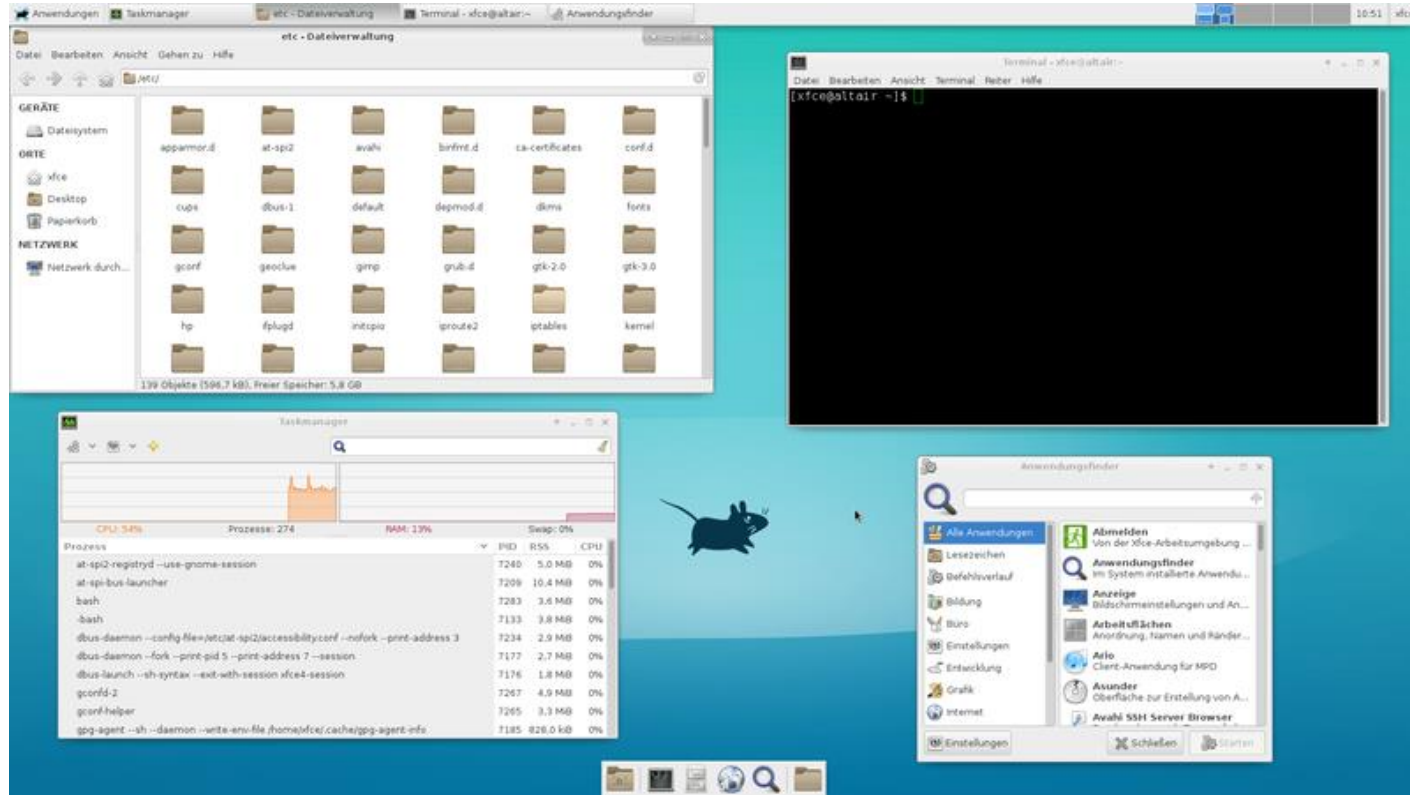
DE Examples: LXDE



DE Examples: MATE



DE Examples: XFCE



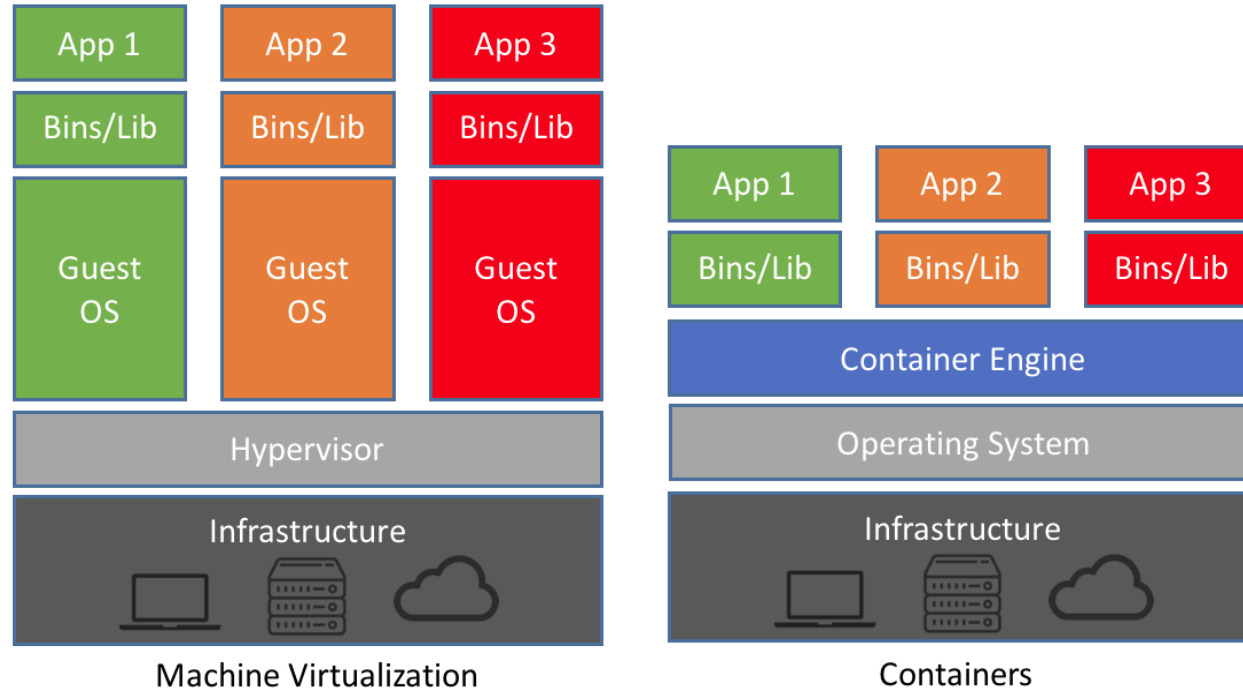
Bash Scripting

Containerization

What is Containerization

- Containerization is a lightweight virtualization method that allows applications to run in isolated environments called containers.
- It's great because of its:
 - **Portability**: Consistent environment from development to production.
 - **Efficiency**: Reduced overhead with shared OS kernel and minimal system resources.
 - **Scalability**: Easily scale applications by deploying multiple containers.
 - **Isolation**: Secure application isolation, preventing dependency conflicts.

Containers vs VMs Visualized



Containers vs VMs

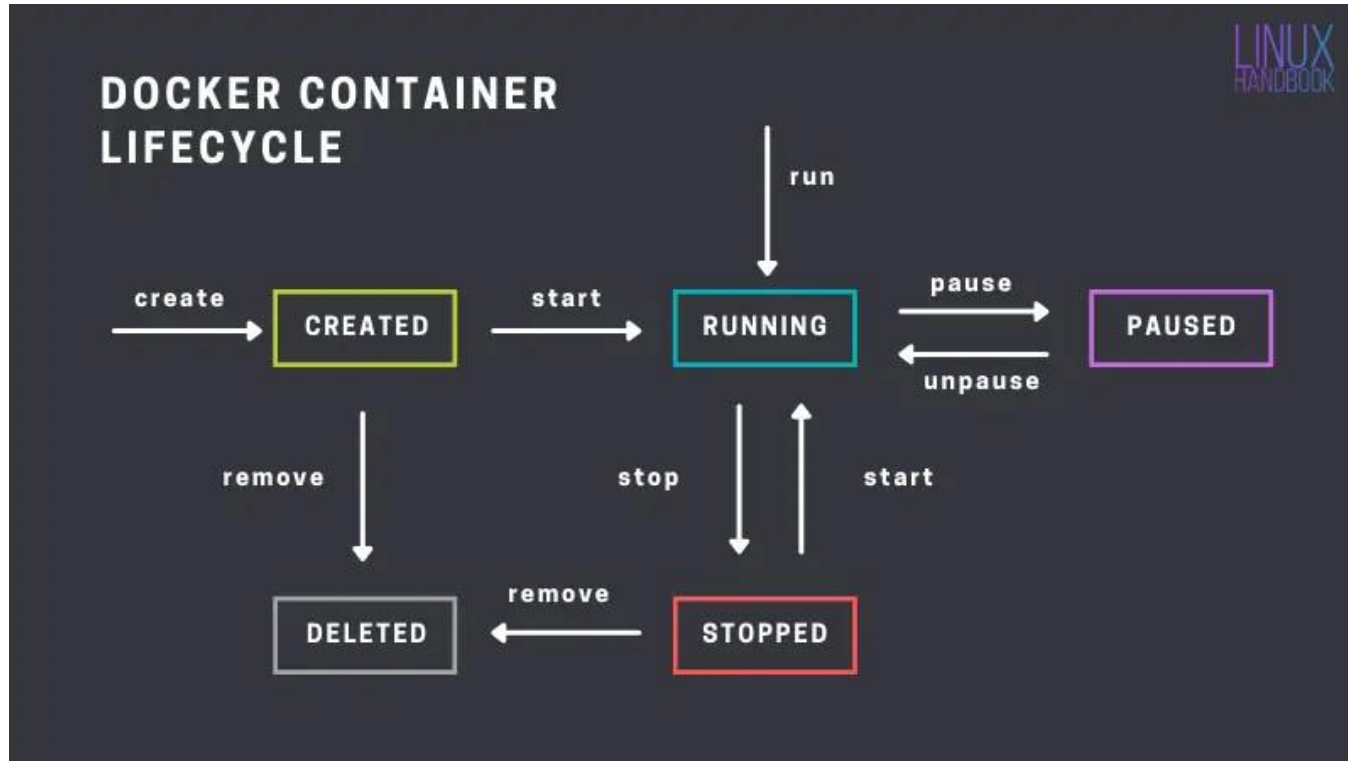
Virtual Machines (VMs):

- **Full OS:** Each VM has its own guest OS and runs on a hypervisor.
- **Resource-Heavy:** Requires more CPU, memory, and storage since each VM contains an entire OS.
- **Strong Isolation:** Complete OS-level isolation, ideal for running multiple OSes on the same hardware.
- **Slower Startup:** Booting an entire OS takes time (seconds to minutes).
- **Use Case:** Running different operating systems, legacy applications, or when strong isolation is required.

Containers:

- **Shared OS Kernel:** Containers share the host OS kernel but isolate applications and dependencies.
- **Lightweight:** Minimal system overhead; faster and more efficient use of resources.
- **Fast Startup:** Containers start almost instantly since only the app process is launched.
- **Portability:** Consistent across environments (e.g., development, testing, production).
- **Use Case:** DevOps pipelines, cloud-native applications requiring fast scaling and deployment.

Container Lifecycle



Tools for Containerization

- **Docker:** Most popular container runtime.
- **Kubernetes:** Container orchestration platform for managing containers at scale.
- **Podman:** Alternative to Docker without a daemon.

