ADMIN:DOCS

LXD: Grundlagen und Architektur verstehen

Lerne die Kernarchitektur von LXD kennen: Wie Systemcontainer und VMs über den Host-Kernel interagieren, wie Namespaces und cgroups isolieren und welche Komponenten zusammenspielen.

Kategorie: LXD **Veröffentlicht:** 02.11.2026

In vielen Betriebsumgebungen wächst der Bedarf, unterschiedliche Dienste und Anwendungsumgebungen strikt voneinander zu trennen: Eine relationale Datenbank soll ungestört von Webserver-Abhängigkeiten laufen, interne Entwicklungs- und Testsysteme verlangen nach einem eigenen Init-System mit SSH-Zugang, und verschiedene Software-Releases setzen auf abweichende Distributionen wie Debian, Ubuntu oder Alpine. Für jede dieser Aufgaben einen eigenen physischen Server bereitzustellen, bindet unverhältnismäßig viel Hardware, Energie und Wartungsaufwand.

Klassische virtuelle Maschinen lösen dieses Isolationsproblem zuverlässig, bringen jedoch für jeden Gast einen separaten Linux-Kernel, emulierte Hardware-Geräte und einen festen Speicher-Overhead mit sich. Häufig ist diese hardwarenahe Emulation für Linux-Workloads gar nicht erforderlich. Hier setzt LXD an: Als Daemon-gestützter Systemcontainer- und VM-Manager ermöglicht LXD die Bereitstellung schlanker Umgebungen, die sich aus Benutzersicht wie vollwertige Linux-Server verhalten, aber direkt auf dem Host-Kernel aufsetzen und Ressourcen dynamisch teilen.

Bevor wir in die Installation und den praktischen Betrieb einsteigen, müssen die architektonischen Grundlagen sitzen: Wie grenzt sich ein Systemcontainer von einem Anwendungscontainer wie Docker oder Podman ab? Wann ist trotz des Overheads eine vollwertige virtuelle Maschine die technisch saubere Wahl? Und welche Kernel-Mechanismen stellen sicher, dass Prozesse, Netzwerke und Speicherpfade zuverlässig voneinander isoliert bleiben?

Quelle: https://admindocs.de/de/lxd/lxd-1-grundlagen-und-architektur-verstehen

> **Kursnotiz zum Selbststudium:** Dieser Kurs führt dich systematisch von den Architektur-Grundlagen bis zum produktiven Betrieb. Behalte von Beginn an im Blick: LXD ist kein Konkurrent zu Docker für flüchtige Single-Process-Microservices, sondern eine vollwertige Infrastruktur-Plattform für langlebige Systemumgebungen, die wie echte Server betrieben und gewartet werden.

Dieser erste Kursteil vermittelt die Architektur und Funktionsweise von LXD, erläutert das Zusammenspiel zwischen dem Hintergrunddienst und dem Kommandozeilenclient `lxc` und stellt die zentralen Komponenten vor, die das Fundament für die folgenden praktischen Module bilden.

Was ist LXD?

LXD ist ein moderner, REST-API-gesteuerter Systemcontainer- und Virtuelle-Maschinen-Manager für Linux. Entwickelt und betreut von Canonical LXD, kombiniert LXD die Ressourcen- und Performance-Vorteile nativer Linux-Container mit dem administrativen Bedienkomfort vollwertiger virtueller Maschinen.

Im Unterschied zu reinen Anwendungscontainern (wie Docker oder Podman), die primär darauf ausgelegt sind, einen einzelnen isolierten Dienst oder Prozess pro Container bereitzustellen, verwaltet LXD sogenannte **Systemcontainer** sowie seit Version 4.0 auch **virtuelle Maschinen** über QEMU. Eine LXD-Instanz verhält sich wie ein vollständiges Linux-Betriebssystem: Sie startet ein eigenes Init-System (in der Regel `systemd`), führt Hintergrunddienste wie `cron`, `journald` oder `sshd` aus und ermöglicht die gewohnte Paketverwaltung über `apt`, `dnf` oder `apk`.

Client-Server-Architektur: Daemon, REST-API und CLI

LXD basiert auf einer strikten Client-Server-Architektur, die lokale wie auch verteilte Administration über dieselbe Schnittstelle erlaubt:

- **Der LXD-Daemon (`lxd`):** Der eigentliche Kern des Systems läuft als privilegierter Hintergrunddienst. Er verwaltet den Lebenszyklus aller Instanzen, steuert Storage-Pools, erzeugt virtuelle Netzwerke, führt Zugriffskontrollen durch und kommuniziert im

Containerbetrieb mit den Low-Level-Kernel-Schnittstellen sowie im VM-Betrieb mit QEMU/KVM.

- **Die REST-API:** Sämtliche Funktionen des Daemons sind über eine strukturierte REST-API erreichbar. Bei lokalen Aufrufen erfolgt die Kommunikation über einen UNIX-Socket (`/var/snap/lxd/common/lxd/unix.socket` bei Snap-Installationen), bei entfernten Systemen über TLS-verschlüsselte HTTPS-Verbindungen (standardmäßig auf TCP-Port 8443).
- **Der Kommandozeilenclient (`lxc`):** Für die tägliche Administration dient das CLI-Tool `lxc` . Das Werkzeug führt selbst keine Container-Operationen direkt im Kernel aus; es nimmt Befehle des Administrators entgegen, übersetzt sie in REST-Aufrufe und schickt sie an den Daemon. Dadurch kann derselbe `lxc` -Befehl wahlweise den lokalen Host oder entfernte LXD-Server und Cluster steuern.

> ⚠ **Sicherheitsgrenze der Benutzergruppe lxd:** Lokale Benutzerkonten, die der Host-Gruppe `lxd` hinzugefügt werden, erhalten direkten Lese- und Schreibzugriff auf den lokalen UNIX-Socket des Daemons. Da über die REST-API beliebige Host-Verzeichnisse in Instanzen gemountet werden können, ist eine Mitgliedschaft in der `lxd` -Gruppe administrativ gleichbedeutend mit vollständigen Root-Rechten auf dem Hostsystem.

Abgrenzung: LXD ist nicht LXC

Ein häufiges Missverständnis rührt von der Namensgebung des Kommandozeilenclients her: Weil der Befehl `lxc` heißt, werden LXC und LXD in der Praxis oft fälschlicherweise synonym verwendet. Technisch handelt es sich jedoch um zwei grundlegend verschiedene Schichten:

- **LXC (LinuX Containers):** LXC ist das ursprüngliche, hardwarenahe Low-Level-Projekt (liblxc), geschrieben in C. Es stellt Bibliotheken und rudimentäre Hilfswerkzeuge bereit, um Linux-Kernel-Features (wie Namespaces und cgroups) manuell zu Containern zusammenzufügen. LXC besitzt keinen zentralen Daemon, keine REST-API, kein integriertes Netzwerk- und Speichermanagement und keine Unterstützung für virtuelle Maschinen.
- **LXD:** LXD setzt historisch auf der Low-Level-Bibliothek von LXC auf, abstrahiert diese jedoch vollständig. LXD ergänzt die Basis um einen mandantenfähigen Daemon, automatisierte Image-Verwaltung aus öffentlichen und privaten Repositories,

softwaredefinierte Netzwerke, dynamische Storage-Pools (ZFS, Btrfs, LVM) und Clustering.

> **Praxis-Hintergrund zur Begriffstrennung:** Wenn ältere Dokumentationen von Befehlen wie `lxc-create`, `lxc-start` oder Konfigurationsdateien unter `/var/lib/lxc/` sprechen, handelt es sich um das klassische Low-Level-LXC. Der moderne LXD-Client verwendet hingegen immer den einheitlichen Befehl `lxc` mit Subkommandos wie `lxc launch`, `lxc exec` oder `lxc list`.

Wer heute mit dem Befehl `lxc launch` eine Instanz startet, arbeitet mit LXD. Das originale LXC wird auf modernen Servern nur noch selten direkt administriert, sondern dient modernen Plattformen als unterlagerte Laufzeitumgebung.

Container und virtuelle Maschinen im Vergleich

LXD nimmt im Linux-Ökosystem eine Sonderstellung ein, weil es zwei grundverschiedene Virtualisierungsansätze unter einer einheitlichen Bedienoberfläche vereint: Systemcontainer und virtuelle Maschinen. Beide Ansätze lösen das Problem isolierter Umgebungen, setzen jedoch auf völlig unterschiedlichen Ebenen des Betriebssystems an.

Kernel-Sharing versus Hardware-Virtualisierung

Der fundamentale Unterschied liegt in der Behandlung des Betriebssystem-Kernels:

- **Systemcontainer (Kernel-Sharing):** Ein LXD-Container ist aus Sicht des Betriebssystems eine isolierte Prozessgruppe auf dem Host. Es gibt keine emulierte Hardware und keinen virtuellen Prozessor. Jeder Systemaufruf (*syscall*), den ein Prozess im Container absetzt, wird direkt vom Linux-Kernel des Hostsystems verarbeitet. Der Container besitzt lediglich ein eigenes Root-Dateisystem, eigene Netzwerk-Interfaces und getrennte Prozess-IDs. Das führt zu minimalem Overhead: Ein Container verbraucht im Leerlauf nur wenige Megabyte Arbeitsspeicher und startet in Bruchteilen einer Sekunde.
- **Virtuelle Maschinen (Hardware-Virtualisierung):** Eine virtuelle Maschine emuliert mithilfe des Hypervisors QEMU und des Kernel-Moduls KVM einen vollständigen PC

inklusive virtuellem BIOS/UEFI, PCI-Bussen, Speichercontrollern und Netzwerkkarten. Im Gastsystem startet ein völlig eigenständiger Betriebssystem-Kernel. Syscalls verbleiben innerhalb des Gasts und berühren den Host-Kernel nicht direkt. Hardware-Virtualisierungsbefehle moderner CPUs (Intel VT-x, AMD-V) sichern die Trennung auf Prozessorebene ab.

Technische Gegenüberstellung

Die folgende Gegenüberstellung fasst die operativen Unterschiede beider Instanztypen zusammen:

LXD-Systemcontainer

Kernel-Architektur
: Teilt den Linux-Kernel des Hostsystems

Ressourcenbindung
: Dynamisch: Unbelegter RAM steht dem Host sofort zur Verfügung

Startverhalten
: Millisekunden bis wenige Sekunden (wie normaler Prozessstart)

Isolationsgrenze
: Kernel-Namespaces, cgroups v2, AppArmor und Seccomp

Dateisystemzugriff
: Direkte Mounts und Blockgeräte über Host-Speichertreiber

Typischer Einsatz
: Linux-Serverdienste, Build-Farmen, dichte Testumgebungen

Virtuelle Maschinen (KVM / QEMU)

Kernel-Architektur
: Eigener, unabhängiger Gastkernel (Linux, Windows, BSD)

Ressourcenbindung
: Statisch: Zugewiesener RAM wird fest reserviert (ohne Ballooning)

Startverhalten
: 10 bis 30 Sekunden (vollständiger BIOS/UEFI- und Kernel-Boot)

Isolationsgrenze
: Vollständige Hardware-Trennung über CPU-Virtualisierungsflags (VT-x/AMD-V)

Dateisystemzugriff
: Virtuelle Disk-Images (qcow2, raw) oder virtio-fs

Typischer Einsatz
: Kernel-Entwicklung, fremde Betriebssysteme, Multi-Tenant-Hosting

Gemeinsam: LXD steuert beide Instanztypen über dasselbe lxc-Befehlsset und dieselben Profile

Betriebliche Entscheidungskriterien: Wann Container, wann VM?

In der Praxis existiert kein pauschales „besser oder schlechter“. Die Entscheidung für Container oder VM hängt von konkreten technischen Rahmenbedingungen ab:

- **Betriebssystem-Kompatibilität:** Da Systemcontainer den Host-Kernel nutzen, können sie ausschließlich Linux-Distributionen ausführen. Wer Windows Server oder FreeBSD virtualisieren möchte, muss zwingend eine virtuelle Maschine wählen.
- **Kernel-Module und Netzwerk-Stacks:** Benötigt ein Dienst eigene Kernel-Treiber, hardwarenahe Kernel-Module (wie spezifische WireGuard- oder VPN-Erweiterungen) oder Modifikationen an Kernel-Parametern unter `/proc/sys/`, ist eine VM erforderlich. Ein Container teilt diese Parameter mit dem Host und kann sie aus Sicherheitsgründen nicht beliebig manipulieren.
- **Sicherheitsisolation und Multi-Tenancy:** Unprivilegierte Container bieten durch User Namespaces eine robuste Barriere, da `root` im Container auf eine unprivilegierte UID auf dem Host abgebildet wird. Dennoch bleibt die Angriffsfläche der Kernel-Syscalls bestehen: Ein Zero-Day-Exploit im Host-Kernel betrifft potenziell alle Container auf dieser Maschine.

Für nicht vertrauenswürdige Mandanten oder externe Kunden ist die hardwaregestützte Trennung einer VM daher der sicherere Standard.

- **Packungsdichte und Effizienz:** Sollen auf einem Server dutzende oder hunderte isolierte Umgebungen laufen (beispielsweise für Continuous Integration oder Webhosting), übertreffen Systemcontainer jede VM um ein Vielfaches. Sie vergeuden keinen Speicher für redundante Kernel-Caches und belasten die CPU nicht mit Emulations-Overhead.

> **Praxis-Tipp:** Unter LXD musst du dich nicht vorab für eine Technologie entscheiden. LXD erlaubt es, auf demselben Host Systemcontainer und virtuelle Maschinen parallel im selben Netzwerk und auf denselben Speicherpools zu betreiben. Der Startbefehl unterscheidet sich lediglich durch ein einziges Flag: `lxc launch ubuntu:24.04 c1` erzeugt einen Container, während `lxc launch ubuntu:24.04 vm1 --vm` eine vollständige KVM-Maschine startet. Weiterführende Grundlagen zu Hypervisoren behandeln wir im Modul Linux-Administration: Virtualisierung und VM-Management.

Wie LXD unter Linux funktioniert

Um LXD sicher und performant zu betreiben, muss man verstehen, dass Systemcontainer keine eigenständigen „virtuellen Rechner“ sind. Sie sind reguläre Linux-Prozesse, die der Kernel durch geschickte Kombination nativer Kernel-Primitive in eine isolierte Sicht auf das Gesamtsystem sperrt.

LXD-Instanzen & Container-Userspace

systemd, Hintergrunddienste, Paketverwaltung und Dateisysteme

Sicherheitsrichtlinien & Profile

AppArmor, Seccomp-Filter und UID/GID-Mapping via User Namespaces

Linux-Kernel-Isolation

Namespaces, cgroups v2, Veth-Pairs und Bridge-Routing

Host-Hardware & Basis-Betriebssystem

CPU, RAM, Storage-Pools (ZFS/Btrfs/LVM) und physisches Netzwerk

Die Rolle des Linux-Kernels als zentrale Vertrauensbasis

Im Systemcontainer-Betrieb existiert nur ein einziger Kernel: der des Hostsystems. Sämtliche Container teilen sich diesen Kernel. Das bedeutet:

- Jeder Systemaufruf (*system call*), den ein Programm im Container absetzt (etwa `open()`, `fork()`, `read()`), landet direkt beim Host-Kernel.
- Es findet keine CPU-Befehlsemulation statt; der Code wird nativ auf den Prozessorkernen ausgeführt.
- Wenn der Host-Kernel abstürzt oder blockiert, stehen unverzüglich alle darauf laufenden Container still.

Um aus normalen Prozessen eine scheinbar eigenständige Systemumgebung zu formen, orchestriert LXD im Hintergrund zwei zentrale Kernelfunktionen: **Namespaces** und **Control Groups**.

Linux Namespaces: Was der Container sehen darf

Namespaces definieren die *Sichtbarkeitsgrenzen* für Prozesse. LXD nutzt alle modernen Linux-Namespaces, um die Illusion eines isolierten Betriebssystems zu erzeugen:

- **PID-Namespace (`pid`):** Jeder Container erhält einen eigenen Prozessbaum. Der Init-Prozess des Containers startet mit PID 1 und kann nur seine eigenen Kindprozesse sehen. Auf dem Host läuft derselbe Prozess jedoch unter einer regulären, hohen Prozess-ID.
- **Network-Namespace (`net`):** Jeder Container besitzt einen vollständig isolierten Netzwerk-Stack mit eigenen Schnittstellen (wie `eth0` oder `lo`), eigenen Routing-Tabellen, Socket-Listen und unabhängigen Paketfilter-Regeln (`nftables` / `iptables`).
- **Mount-Namespace (`mnt`):** Prozesse im Container sehen nur Dateisysteme, die explizit in ihren Namespace eingehängt wurden. Mount- und Unmount-Vorgänge im Container verändern das Dateisystem des Hosts nicht.

- **User-Namespace (`user`):** Der entscheidende Sicherheitsanker für unprivilegierte Container. User-IDs (UID) und Group-IDs (GID) werden zwischen Container und Host übersetzt. Der Benutzer `root` (UID 0) im Container wird auf dem Host beispielsweise auf UID 1000000 gemappt. Gelingt einem Angreifer ein Ausbruch aus dem Container, besitzt er auf dem Host lediglich die Rechte eines unprivilegierten Standardbenutzers.
- **UTS-Namespace (`uts`):** Ermöglicht dem Container einen eigenen Hostnamen und Domainnamen, unabhängig vom Namen des Hostsystems.
- **IPC-Namespace (`ipc`):** Trennt die Interprozess-Kommunikation (wie Shared Memory und Message Queues) zwischen verschiedenen Containern und dem Host.

Control Groups (cgroups v2): Was der Container verbrauchen darf

Während Namespaces bestimmen, *was* ein Prozess sehen kann, regeln Control Groups (cgroups), *wie viele Ressourcen* er nutzen darf. Sie verhindern, dass ein einzelner fehlerhafter oder überlasteter Container den gesamten Host lahmlegt (*Noisy-Neighbor-Problem*):

- **CPU-Limits:** LXD kann Instanzen feste CPU-Kerne zuweisen (`limits.cpu = 2`) oder deren Rechenzeit über CPU-Shares und Bandbreitenkontrollen drosseln.
- **Arbeitsspeicher-Limits:** Über `limits.memory` wird der maximale RAM-Verbrauch festgelegt. Überschreitet ein Container diese Grenze, greift der Linux-Out-of-Memory-Killer (OOM) gezielt nur innerhalb der betroffenen cgroup, während der Host unbeeinträchtigt bleibt.
- **I/O-Priorisierung:** cgroups steuern Lese- und Schreibdurchsätze auf Festplatten und Blockgeräten, um faire Zugriffszeiten sicherzustellen.
- **Prozess-Limits (PIDs):** Begrenzt die maximale Anzahl gleichzeitiger Prozesse (`limits.processes`), was das System effektiv vor Fork-Bomben schützt.

> **Moderne cgroups v2-Architektur:** Aktuelle Linux-Distributionen (wie Ubuntu 24.04 oder Debian 12) setzen standardmäßig auf die einheitliche cgroups-v2-Hierarchie. LXD nutzt diese moderne Struktur nativ, was eine präzise, dynamische Ressourcensteuerung ohne den historischen Konfigurationsaufwand getrennter Controller ermöglicht. Wie cgroups und Prozessgrenzen manuell verwaltet werden, zeigt unser Modul Linux-

> Administration: Prozesse und Ressourcenverwaltung.

Dateisystem- und Prozessisolation

Zur Dateisystemtrennung verlässt sich LXD nicht auf ein simples `chroot`, das historisch umgangen werden konnte. Stattdessen setzt LXD auf den Linux-Systemaufruf `pivot_root`. Dieser verankert das vorbereitete Root-Dateisystem der Instanz als neuen Root-Einhängepunkt und entkoppelt das alte Host-Dateisystem vollständig aus dem Zugriffsbereich der Container-Prozesse.

Zusätzlich legt LXD standardmäßig zwei weitere Schutzschichten um jeden Container:

- **AppArmor:** Kernel-Profile schränken ein, welche Dateipfade, Geräte und Netzwerkoperationen Prozesse im Container manipulieren dürfen. Selbst wenn ein Prozess Root-Rechte besitzt, verhindert AppArmor beispielsweise den Schreibzugriff auf sensible Kernel-Interfaces unter `/sys` und `/proc`.
- **Seccomp:** Filtert Systemaufrufe an den Kernel. Potenziell gefährliche Syscalls – wie etwa direkte Hardware-Manipulationen oder Versuche, Kernel-Module nachzuladen – werden blockiert, bevor sie den Host-Kernel erreichen.

Netzwerkvirtualisierung über Veth-Paare und Bridges

Standardmäßig erzeugt LXD ein virtuelles Bridge-Netzwerk (wie `lxdbr0`). Die Anbindung der Container erfolgt über sogenannte **Virtual Ethernet Pairs (`veth`)**:

Ein `veth`-Konstrukt funktioniert wie ein softwaredefiniertes Patchkabel mit zwei Enden:

- Das eine Ende wird als Interface (z. B. `vethabcdef12`) an die Bridge auf dem Host angeschlossen.
- Das andere Ende wird direkt in den Netzwerk-Namespace des Containers verschoben und dort als `eth0` initialisiert.

Auf der Bridge sorgt ein interner, schlanker Dienst (in der Regel `dnsmasq`) für die automatische Vergabe privater IPv4- und IPv6-Adressen via DHCP sowie für die lokale Namensauflösung der Instanzen. Der Container kann somit wie ein physischer Server im LAN kommunizieren, ohne

dass am physischen Switch des Rechenzentrums spezielle Anpassungen erforderlich sind.

Die wichtigsten LXD-Komponenten

Die Administration von LXD unterscheidet sich grundlegend von Systemen, bei denen einzelne Konfigurationsdateien manuell auf dem Dateisystem gepflegt werden. LXD verwaltet Umgebungen über logisch voneinander getrennte Komponenten und Ressourcen, die der Daemon in einer internen, transaktionssicheren Datenbank (`dqlite`) speichert und über die REST-API bereitstellt.

Die Kernkomponenten im Überblick

- **LXD-Daemon (`lxd`):** Der zentrale Hintergrunddienst. Er steuert den Status aller Instanzen, verwaltet Schnittstellen zu Kernel und Hypervisor (QEMU), überwacht Ressourcen und nimmt Anfragen über den UNIX-Socket oder das Netzwerk entgegen.
- **Kommandozeilenclient (`lxc`):** Das administrative Werkzeug. Es dient als Schnittstelle zwischen Administrator und Daemon. Über das Konzept von *Remotes* kann derselbe Client problemlos zwischen dem lokalen Server und entfernten LXD-Installationen wechseln.
- **Images:** Die Schablonen zur Erstellung neuer Instanzen. Ein Image ist ein komprimiertes Abbild eines sauberen Basissystems (etwa Ubuntu 24.04, Debian 12 oder Alpine 3.20). LXD bezieht Images aus vorkonfigurierten Remote-Servern (`ubuntu:`, `ubuntu-daily:`, `images:`), verifiziert deren SHA-256-Fingerprint und hält sie im lokalen Cache vor.
- **Instanzen (Instances):** Der übergeordnete Begriff für alle von LXD verwalteten Systeme. Eine Instanz ist entweder ein *Systemcontainer* oder eine *virtuelle Maschine*. Jede Instanz besitzt einen eindeutigen Namen, Konfigurationsparameter (wie CPU- und Memory-Limits), zugeordnete Geräte und einen Zustand (`RUNNING`, `STOPPED`, `FROZEN`).
- **Profile (Profiles):** Wiederverwendbare Konfigurationssammlungen. Profile legen fest, welche Geräte (z. B. Root-Disk und Netzwerkkarten) und welche Beschränkungen einer Instanz zugewiesen werden. Jede neue Instanz erbt standardmäßig das Profil `default`. Änderungen an einem Profil werden sofort auf alle verknüpften Instanzen angewendet.
- **Storage-Pools:** Abstraktionsebene für Massenspeicher. LXD unterstützt spezialisierte Dateisysteme wie ZFS und Btrfs sowie LVM-Thin-Pools oder einfache Verzeichnisse (`dir`).

Der Storage-Pool verwaltet die Dateisysteme der Instanzen, speichert benutzerdefinierte Datenvolumes und ermöglicht sekundenschnelle Copy-on-Write-Snapshots.

- **Networks:** Vom Daemon gesteuerte Netzwerkschnittstellen. Meist handelt es sich um softwaredefinierte Linux-Bridges (`lxdbr0`), OVN-Netzwerke oder Macvlan-Schnittstellen. LXD übernimmt hierbei automatisch Routing, Adressvergabe (DHCP für IPv4/IPv6) und DNS-Namensauflösung.
- **Projects:** Mandantenfähigkeit und logische Isolation auf Organisationsebene. Ein Projekt isoliert Instanzen, Profile, Images und Speicherbereiche vollständig von anderen Projekten. Administratoren können so Entwicklungs-, Staging- und Produktivumgebungen auf demselben Server trennen oder Ressourcenquotas für verschiedene Teams vergeben.

Logisches Zusammenspiel beim Instanzstart

Wenn ein Administrator über das CLI den Befehl `lxc launch ubuntu:24.04 web01` ausführt, greifen diese Komponenten in einer klar definierten Kette ineinander:

1. **Image beziehen & prüfen** LXD prüft den lokalen Cache oder lädt das Basis-Image aus dem Image-Repository
2. **Storage-Volume zuweisen** Im Storage-Pool wird mittels Copy-on-Write ein neues Root-Dateisystem für die Instanz geklont
3. **Profile vererben** Netzwerk- und Disk-Konfigurationen aus dem default-Profil werden auf die Instanz angewendet
4. **Instanz im Projekt initialisieren** Namespaces, cgroups und veth-Netzwerkgeräte werden erzeugt und das Init-System gestartet

Typische Einsatzgebiete

LXD schließt die Lücke zwischen leichtgewichtigen Anwendungscontainern und schwerfälligen Hypervisor-Umgebungen. Im operativen IT-Betrieb deckt es dadurch Szenarien ab, die mit

herkömmlichen Werkzeugen entweder zu ressourcenintensiv oder administrativ zu starr wären.

Praxisnahe Szenarien in der Administration

- **Entwicklungs- und Staging-Umgebungen:** Entwickler und Systemadministratoren können innerhalb von zwei Sekunden ein vollständiges, sauberes Linux-System starten. Da ein LXD-Container ein echtes `systemd`-Init betreibt, können Systemdienste wie Nginx, PostgreSQL, Redis oder SSH exakt so konfiguriert und getestet werden wie auf einem späteren Produktionsserver – ohne Virtualisierungs-Tools wie VirtualBox auf Entwickler-Laptops installieren zu müssen.
- **Multi-Distributions-Infrastruktur auf einem Host:** Auf einem einzigen physischen Server (beispielsweise unter Ubuntu 24.04 LTS) lassen sich problemlos Container mit Debian 12, Alpine Linux, Arch Linux oder AlmaLinux parallel betreiben. Das vereinfacht das Testen von Softwarekompatibilitäten über unterschiedliche Linux-Paketformate hinweg erheblich.
- **Konsolidierung klassischer Serverdienste:** Dienste, die traditionell separate Bare-Metal-Maschinen oder eigene KVM-Instanzen belegt haben (wie DNS-Server mit BIND, Mail-Relays, VPN-Endpunkte oder Monitoring-Exporteure), können isoliert in Systemcontainern betrieben werden. Der Host spart dadurch massiv Arbeitsspeicher, da kein redundanter Kernel-Speicher reserviert wird.
- **Flüchtige CI/CD-Build-Systeme:** Continuous-Integration-Pipelines profitieren von der hohen Startgeschwindigkeit und den Copy-on-Write-Eigenschaften: Ein Container wird für einen Build-Lauf instanziiert, baut die Software in einer isolierten Umgebung und wird anschließend sofort gelöscht oder auf einen sauberen Snapshot zurückgesetzt.

Entscheidungshilfe: Systemcontainer versus Virtuelle Maschine

Für die Praxis empfiehlt sich ein strukturierter Entscheidungspfad:

Wann ein Systemcontainer die richtige Wahl ist:

- Die geplante Arbeitslast läuft nativ unter Linux.
- Es wird maximale I/O- und Prozessorleistung ohne Virtualisierungsverluste benötigt.

- Auf dem Host soll eine hohe Packungsdichte erreicht werden (hohe Anzahl paralleler Systeme bei begrenztem RAM).
- Speicherplatz und RAM sollen dynamisch mit dem Host geteilt werden.

Wann eine virtuelle Maschine die richtige Wahl ist:

- Es muss ein Nicht-Linux-Betriebssystem (wie Windows Server oder FreeBSD) betrieben werden.
- Die Anwendung benötigt spezifische Kernel-Treiber oder modifizierte Kernel-Parameter, die auf dem Host-Kernel nicht aktiviert werden können.
- Es werden Hardware-Funktionen wie direkter PCI-Passthrough (z. B. für dedizierte GPUs) benötigt.
- Verschiedene, nicht vertrauenswürdige Mandanten müssen mit maximaler hardwaregestützter Sicherheit voneinander getrennt werden.

> **Praxis-Tipp für Systemumgebungen:** Nutze Snapshots vor riskanten Eingriffen! Da LXD direkt mit modernen Storage-Pools wie ZFS oder Btrfs zusammenarbeitet, erzeugt der Befehl `lxc snapshot web01 vor-update` innerhalb von Millisekunden einen konsistenten Sicherungspunkt. Schlägt ein Distributions-Upgrade oder ein Konfigurationstest fehl, setzt `lxc restore web01 vor-update` das gesamte System verzögerungsfrei zurück.

Voraussetzungen für die praktische Arbeit

Bevor wir in Teil 2 dieses Kurses die eigentliche Installation und Initialisierung von LXD durchführen, muss sichergestellt sein, dass das Hostsystem die nötigen technischen Grundlagen erfüllt. LXD ist eng mit modernen Linux-Kernel-Mechanismen und Speicherarchitekturen verzahnt.

Zwingende technische Voraussetzungen

Ohne die folgenden Kernanforderungen kann LXD nicht betrieben werden:

- **64-Bit-Betriebssystem:** Ein modernes 64-Bit-Linux (x86_64 / amd64 oder aarch64 / arm64). Typische Enterprise- und Server-Distributionen sind Ubuntu Server (22.04 LTS oder 24.04 LTS), Debian (11 oder 12) sowie RHEL-basierte Derivate (wie AlmaLinux oder Rocky Linux 9).
- **Linux-Kernel-Version:** Mindestens Kernel 5.4 oder neuer. Für moderne Features wie cgroups v2, performante Speicher-Backends und hardwaregestützte Isolationsfilter wird Kernel 5.15 LTS oder ein aktueller 6.x-Kernel dringend empfohlen.
- **Kernel-Subsysteme:** Der Kernel muss User Namespaces (`CONFIG_USER_NS`), PID-Namespaces, Network-Namespaces sowie Control Groups (`CONFIG_CGROUPS`) aktiviert haben. Auf Standard-Kerneln gängiger Server-Distributionen ist dies ab Werk der Fall.
- **Hardware-Virtualisierung für VMs (KVM):** Reine Systemcontainer benötigen keine speziellen CPU-Virtualisierungsbefehle. Wenn du unter LXD jedoch auch **virtuelle Maschinen** starten möchtest, muss der Prozessor Hardware-Virtualisierung unterstützen (Intel VT-x mit `vmx` -Flag oder AMD-V mit `svm` -Flag). Diese Funktion muss im BIOS/UEFI des Servers aktiviert sein.

Empfehlungen für den produktiven Betrieb

Für eine stabile und performante Arbeitsumgebung im Serverbetrieb sollten folgende Richtwerte beachtet werden:

- **Arbeitsspeicher (RAM):** Für erste Gehversuche und Testsysteme genügen 4 GB RAM. Im produktiven Betrieb hängt der Bedarf von den betriebenen Diensten ab: Da Systemcontainer keinen festen Speicher reservieren, sondern nur das belegen, was ihre Prozesse tatsächlich verbrauchen, lassen sich auf 16 GB RAM oft problemlos 20 bis 40 schlanke Linux-Container parallel betreiben. Für virtuelle Maschinen muss der zugewiesene Speicher hingegen fest eingerechnet werden.
- **Massenspeicher & Storage-Treiber:** Es sollten mindestens 30 bis 50 GB freier Festplattenspeicher zur Verfügung stehen. Die Wahl des Speicher-Backends entscheidet maßgeblich über die Performance:
- **Empfohlen (Copy-on-Write):** ZFS oder Btrfs. Beide Dateisysteme erlauben es LXD, neue Instanzen aus Basis-Images innerhalb von Millisekunden zu klonen und Snapshots ohne zusätzlichen Speicherverbrauch zu erstellen.

- **Alternativ:** LVM mit Thin-Provisioning (`lvm-thin`).
- **Nur für einfache Tests:** Der `dir` -Treiber (einfache Ordnerstruktur auf ext4 oder XFS). Dieser erfordert beim Erstellen von Snapshots oder Klonen langsame, vollständige Dateikopien (`rsync`) und ist für den Produktivbetrieb nicht empfehlenswert.
- **Netzwerkanbindung:** Eine funktionierende Internetverbindung ist erforderlich, damit LXD beim ersten Start die benötigten Basis-Images (wie das Ubuntu- oder Debian-Rootfs) aus den offiziellen Repositories herunterladen kann. Zudem muss auf dem Host die Erstellung virtueller Bridge-Devices (`lxdbr0`) gestattet sein.

> □□ **Prüfung vorab:** Du kannst die Virtualisierungsunterstützung deiner CPU bereits vor der Installation mit einem einfachen, lesenden Befehl prüfen: `egrep -c '(vmx|svm)' /proc/cpuinfo` . Liefert der Befehl einen Wert größer als 0, unterstützt dein Prozessor KVM-basierte virtuelle Maschinen unter LXD.

> ⚠ **Virtualisierung in Cloud-Instanzen (Nested Virtualization):** Wenn du LXD auf einer gemieteten Cloud-VM (z. B. bei Hetzner, AWS oder DigitalOcean) betreiben möchtest, funktionieren Systemcontainer uneingeschränkt. Das Starten virtueller Maschinen (`--vm`) setzt jedoch voraus, dass der Cloud-Provider *Nested Virtualization* für deine Instanz freigeschaltet hat.

Befehlsreferenz (Cheatsheet)

lxc version

$ `Zeigt die installierte Client- und Server-Version von LXD an`

lxc info

$ `Liefert umfassende Statusinformationen über den Host, Kernel-Treiber und Speicherpools`

lxc list

$ `Listet alle vorhandenen Instanzen (Container und VMs) mit IP-Adressen und Status auf`

lxc image list

$ `Zeigt alle lokal im Cache vorgehaltenen Betriebssystem-Images an`

lxc profile list

$ `Listet alle definierten Konfigurationsprofile des aktuellen Projekts auf`

lxc network list

$ `Gibt eine Übersicht über alle verwalteten und unverwalteten Netzwerk-Bridges aus`

lxc storage list

$ `Listet alle konfigurierten Speicherpools samt Treibertyp (ZFS, Btrfs, LVM) auf`

lxc project list

$ `Zeigt alle angelegten LXD-Projekte zur Mandantentrennung auf dem Host an`

Weiterführende Ressourcen

Fazit

Mit dem Verständnis der LXD-Architektur hast du das Fundament für die moderne Systemcontainer- und VM-Administration unter Linux gelegt. LXD schließt die Lücke zwischen leichtgewichtigen Anwendungscontainern und schwerfälligen Hypervisoren, indem es native Linux-Kernel-Primitive – Namespaces für Sichtbarkeit, cgroups v2 für Ressourcenkontrolle sowie AppArmor und Seccomp für Zugriffssicherheit – über einen zentralen, REST-gesteuerten Daemon bündelt.

Die strikte Trennung zwischen dem Hintergrunddienst `lxd` und dem Client `lxc`, das Zusammenspiel aus Images, Profilen, Bridges und Copy-on-Write-Storage-Pools sowie die

Möglichkeit, Container und vollwertige KVM-Maschinen parallel zu betreiben, machen LXD zu einer der flexibelsten Virtualisierungsplattformen für moderne Rechenzentren und Entwicklungsumgebungen.

> □□ **Praxis-Tipp:** Merke dir für die tägliche Administration die Faustformel: *Namespaces bestimmen, was ein Container sehen darf; cgroups bestimmen, wie viel er verbrauchen darf.* Wenn eine Instanz Ressourcenprobleme meldet oder Dienste unerwartet beendet werden, ist die cgroup-Konfiguration (`limits.memory` , `limits.cpu`) stets der erste Prüfpunkt.

Im nächsten Teil unserer Kursserie widmen wir uns der praktischen Inbetriebnahme: In **LXD – Teil 2: Installation und Initialisierung** konfigurieren wir den Host, führen `lxd init` Schritt für Schritt durch, richten das erste Bridge-Netzwerk ein und erstellen unseren ersten produktiven ZFS-Storage-Pool.

□□ **Zur Kursübersicht:** Alle LXD-Artikel & Module

Quelle: https://admindocs.de/de/lxd/lxd-1-grundlagen-und-architektur-verstehen