

Ubuntu Server Advanced Training Curriculum

Hands-on training for Ubuntu Server administrators
Prepared by: Andrés Mujica – Cloudbase Solutions
Prepared on: 14 May 2026

Contents

Course overview.....	3
Audience, prerequisites and format.....	3
Module 1: Virtualization with QEMU/KVM and libvirt.....	4
Module 2: LXD System Containers.....	4
Module 3: OpenSSH at Scale.....	5
Module 4: Boot and System Initialization.....	6
Module 5: Advanced Storage.....	6
Module 6: Advanced Filesystem Concepts.....	7
Module 7: ZFS on Ubuntu.....	8
Module 8: Advanced Networking with Netplan.....	8
Module 9: Security: PAM, ACLs, AppArmor, and Ubuntu Pro.....	9
Module 10: Advanced Snap Packaging and Ubuntu Core.....	10
Module 11: Time Synchronization.....	10

Course overview

The **Advanced Ubuntu Server training** is the natural next step after the Basic course (or equivalent on-the-job experience). It is a three-day, hands-on course aimed at experienced administrators who need to operate Ubuntu Server in production at scale: running virtual machines and system containers, building resilient storage, designing complex network topologies, and enforcing security and compliance with the tools native to the Ubuntu platform.

The course is built on the current **Ubuntu Server LTS 24.04 “Noble”**. Theory blocks are kept deliberately short — just enough to anchor the concept — and each module is anchored by a substantial lab. Students leave the course able to design, deploy and troubleshoot the advanced features of Ubuntu Server that customers actually rely on in production: KVM/libvirt, LXD, OpenSSH at scale, systemd, software RAID and LVM, ZFS, Netplan with VLANs/bonds/bridges, PAM, AppArmor, Ubuntu Pro and Snaps.

Audience, prerequisites and format

Audience

Experienced Linux administrators, senior SREs and DevOps engineers, infrastructure engineers and platform teams responsible for operating Ubuntu Server in production environments.

Prerequisites

Comfortable working day-to-day on the Linux command line. Familiarity with the topics covered in the Basic Ubuntu Server training (filesystem hierarchy, users and permissions, systemd basics, package management, basic networking and storage) is expected.

Duration

Three (3) days. 6 hours of contact time per day, delivered live (online or on-site).

Format

Instructor-led, hands-on. Each student is provisioned with their own dedicated Ubuntu Server lab environment supporting nested virtualization, multiple disks and multiple network interfaces. Most of the class time is spent in the lab.

Materials

Slide deck, printable lab guide, persistent student lab environments, post-training command reference.

Module 1: Virtualization with QEMU/KVM and libvirt

Advanced Ubuntu administration starts with virtualization. This module gives students a working understanding of the KVM virtualization stack that ships with Ubuntu: how QEMU emulates hardware, how KVM accelerates it with kernel support, and how libvirt provides the operational layer used by every higher-level Canonical product (MAAS, OpenStack, LXD VMs) builds on. Students also get a working introduction to cloud-init (the way Ubuntu cloud and MAAS-deployed instances are bootstrapped) and leave the module able to provision and operate KVM guests cleanly from the command line.

What students will learn

- Explain the role of QEMU, KVM and libvirt in the Ubuntu virtualization stack.
- Provision a Linux guest VM from scratch on an Ubuntu KVM host.
- Manage storage pools and volumes for VM disks.
- Define and attach libvirt networks to guest VMs.
- Use cloud-init to bootstrap a guest with users, SSH keys, packages and run-commands.

Topics covered

- Virtualization fundamentals: full vs. para-virtualization, type-1 vs. type-2 hypervisors
- QEMU as a machine emulator; KVM as a kernel acceleration module
- qemu-img and disk image formats: raw, qcow2; backing files and snapshots
- libvirt architecture: libvirtd, the virsh CLI and XML domain definitions
- Storage pools and volumes (directory, LVM, RBD); creating and cloning volumes
- libvirt networking: default NAT, isolated, routed and bridged networks
- Guest installation methods: virt-install, cloud images and cloud-init
- cloud-init in depth: user-data, meta-data, modules and the most useful directives
- Where this stack shows up in the broader Canonical portfolio (MAAS, OpenStack, LXD VMs)

Hands-on lab exercises

- Install and verify the KVM/libvirt stack on an Ubuntu host.
- Provision a Linux guest VM end-to-end with virt-install.
- Create directory and file-backed storage pools; clone volumes between guests.
- Define a NAT network and an isolated network; attach them to guest VMs.
- Boot a cloud image with a custom cloud-init user-data file that creates users, installs packages and runs a first-boot script.

Module 2: LXD System Containers

Where Docker is built around the unit of one process, LXD is built around the unit of one machine. This module covers LXD, the Canonical-built system container and VM manager that powers everything from developer laptops to MAAS labs. Students learn the LXD model end-to-end — instances, profiles, images, storage and networks — and finish the module able to operate a production-style LXD host.

What students will learn

- Differentiate system containers (LXD) from application containers (Docker).
- Install LXD, initialise it and operate it via the lxc CLI.
- Launch and manage containers and VMs from remote image servers.

- Use profiles, snapshots and storage pools to build repeatable LXD environments.
- Configure LXD networking, including custom bridges and external connectivity.

Topics covered

- Containerization recap: namespaces, cgroups and the kernel features LXD relies on
- LXD architecture: the system-wide daemon, the REST API and the lxc client
- Remote image servers and image management
- Instances: containers and VMs; configuration, profiles and devices
- Snapshots and stateful vs. stateless restores
- Storage pools (dir, zfs, btrfs, lvm) and volumes
- LXD networks: managed bridges, fan and external integration
- Running KVM VMs inside LXD

Hands-on lab exercises

- Install LXD via snap and run lxd init.
- Launch containers from the Ubuntu remote image server; inspect images locally.
- Create and restore snapshots; clone containers from snapshots.
- Build a custom profile and apply it to multiple instances.
- Define an additional LXD network and attach an instance to it.
- Launch a KVM-style VM inside LXD and connect to it.

Module 3: OpenSSH at Scale

OpenSSH is the de facto remote-management plane for every Ubuntu fleet. This module goes well beyond “ssh user@host”: students learn how to operate SSH as a production protocol, with key management, agent forwarding, jump hosts, port forwarding for tunneling and a clean security hardening baseline for sshd.

What students will learn

- Explain OpenSSH's client-server architecture and threat model.
- Generate, distribute and rotate SSH keys safely.
- Use ssh-agent and ProxyJump to make complex topologies practical.
- Build local, remote and dynamic port forwards.
- Harden an OpenSSH server against the most common attack patterns.

Topics covered

- OpenSSH overview and the client-server model
- SSH keys, key types and key-based authentication
- Key management tools: ssh-keygen, ssh-agent, ssh-add, ssh-import-id, ssh-copy-id
- Server configuration: /etc/ssh/sshd_config and its most important directives
- Client configuration: ~/.ssh/config, Host blocks, identity files and proxies
- SSH hardening recommendations (disable root login, key-only auth, MFA hooks, idle timeouts)
- Port forwarding: local (-L), remote (-R) and dynamic (-D) tunnels
- ProxyJump and bastion-host patterns
- Using rsync over SSH

Hands-on lab exercises

- Generate an ED25519 key pair; distribute the public key with ssh-copy-id.

- Use ssh-agent and ssh-add to authenticate transparently against multiple hosts.
- Configure ~/.ssh/config with multiple Host blocks and ProxyJump.
- Configure local port forwarding to reach an internal service through a bastion.
- Configure remote port forwarding from a private host to a public jump host.
- Apply an SSH hardening baseline to sshd_config and verify with ssh -v.

Module 4: Boot and System Initialization

When a server doesn't come back up, the first hour is what counts. This module demystifies the Ubuntu boot path, from GRUB2 through the kernel and into a fully initialised systemd environment, and turns systemd from a vague "service manager" into a precise operational tool. Students leave the module able to customise boot, write their own units and timers, and read the journal like a pro.

What students will learn

- Customise the GRUB2 boot menu and kernel command line.
- Switch between stock and Hardware Enablement (HWE) kernels safely.
- Write and manage systemd unit files for services, sockets and timers.
- Read and filter structured logs with journalctl.
- Schedule one-shot work with systemd-run, at and traditional cron.

Topics covered

- The Ubuntu boot chain: firmware → GRUB2 → kernel → initramfs → systemd
- GRUB2 configuration: /etc/default/grub, /etc/grub.d/ and update-grub
- HWE kernels and rolling kernel updates
- systemd units: service, socket, target, mount, path and timer types
- Managing units with systemctl; daemon-reload semantics
- Structured logging with journalctl: filtering, time ranges and persistence
- Troubleshooting systemd: systemd-analyze, dependencies and boot blame
- Scheduled and on-demand tasks: cron, at, systemd-run, systemd timers

Hands-on lab exercises

- Customise the GRUB2 display and default kernel; verify with reboot.
- Install and test an HWE kernel; roll back to the stock kernel.
- Manage existing services with systemctl and inspect them with journalctl.
- Write a custom systemd service unit and timer; enable and verify them.
- Schedule one-shot tasks with systemd-run and at; compare with cron.

Module 5: Advanced Storage

This module covers the storage features that Ubuntu administrators reach for the moment a workload outgrows a single disk: software RAID for redundancy and performance, LVM for flexible allocation and online resize, and device-mapper multipathing for resilient access to shared storage. The focus is hands-on, every concept is built, broken on purpose, and recovered.

What students will learn

- Choose between MBR and GPT partitioning for a given disk.
- Build, monitor and recover Linux software RAID arrays.
- Use LVM features beyond basic VG/LV management (thin pools, snapshots, online resize, migration).

- Configure device-mapper multipathing for resilient access to SAN/iSCSI storage.

Topics covered

- Partitioning schemes: MBR vs. GPT; partition table tooling (parted, fdisk, sgdisk)
- Software RAID with mdadm: RAID 0/1/5/6/10, hot spares, monitoring
- Simulating drive failures and rebuilding arrays
- Advanced LVM: thin provisioning, snapshots, online extend, pvmove migration
- Device Mapper Multipath: multipathd, /etc/multipath.conf and path policies
- Performance and resilience trade-offs

Hands-on lab exercises

- Partition disks with parted (GPT) and fdisk (MBR).
- Build a RAID 5 array with mdadm; simulate a drive failure and rebuild.
- Extend an LVM logical volume online; migrate a PV with pvmove.
- Create an LVM thin pool and thin-provisioned volumes.
- Configure device-mapper multipathing against the lab's shared storage.

Module 6: Advanced Filesystem Concepts

Filesystems are more than “where files live”. This module opens up the internals (inodes, superblocks, journals, extended attributes and POSIX ACLs) and then puts them to work. Students learn how to tune ext4 for specific workloads, apply fine-grained ACLs and use the special permission bits to solve real problems.

What students will learn

- Explain the role of inodes, superblocks, journals and extents in modern Linux filesystems.
- Apply and audit POSIX ACLs and extended attributes.
- Tune ext4 mount options and journaling modes for a given workload.
- Use setuid, setgid and the sticky bit safely.

Topics covered

- Filesystem internals: inodes, superblocks, journals, extents
- Extended attributes (xattrs) and use cases (security labels, SELinux/AppArmor)
- POSIX ACLs: getfacl, setfacl, default ACLs and ACL inheritance
- ext4 in depth: journaling modes (data=journal/ordered/writeback), barriers, tune2fs
- Online resize with resize2fs
- Special permission bits: setuid, setgid (files and directories), sticky bit

Hands-on lab exercises

- Inspect inode and superblock state with stat, debugfs and tune2fs.
- Apply POSIX ACLs and verify them; copy ACLs between directories.
- Read and modify extended attributes.
- Grow an ext4 filesystem online with resize2fs.
- Demonstrate setuid, setgid on shared directories and the sticky bit on /tmp.

Module 7: ZFS on Ubuntu

ZFS combines what would normally be three separate layers (volume manager, RAID and filesystem) into a single, integrated design. This module gives students a solid working knowledge of ZFS on Ubuntu: how pools and datasets are organised, how to use snapshots and clones, how to ship data with `zfs send/recv`, and where the knobs (compression, deduplication, quotas) move the needle.

What students will learn

- Describe the ZFS architecture: pools, vdevs, datasets and volumes.
- Choose an appropriate RAIDZ layout for a given workload.
- Use snapshots, clones and send/receive for backups and migrations.
- Tune ZFS with compression, deduplication, quotas and reservations.
- Maintain a healthy pool with scrubs, trims and replacement workflows.

Topics covered

- ZFS overview and design philosophy
- Pools and vdevs: mirror, RAIDZ1/2/3, special and log devices
- Datasets, volumes (zvols) and common dataset properties
- Snapshots, clones and the snapshot lifecycle
- Send and receive: incremental backups, replication and migration
- Tuning: compression (lz4, zstd), deduplication, recordsize
- Quotas, reservations and refquotas
- Scrubs, trims and dealing with degraded devices
- Mounting and inheritance

Hands-on lab exercises

- Create mirrored and RAIDZ pools; observe vdev layout and pool status.
- Enable compression and measure the resulting ratio.
- Take recursive snapshots; roll back to a previous snapshot.
- Use `zfs send / zfs receive` to replicate datasets, including incrementally.
- Set quotas and reservations; trigger and observe a scrub.

Module 8: Advanced Networking with Netplan

Netplan is how Ubuntu describes networking declaratively. This module covers the topologies that customers actually deploy in production: tagged VLANs, link-bonding for resilience and throughput, and software bridges that knit virtual machines and containers into the physical network. Students leave the module able to author a production-grade Netplan configuration with confidence.

What students will learn

- Author and apply Netplan configurations using its YAML schema.
- Configure tagged VLAN interfaces on top of physical NICs.
- Build link bonds (active-backup, LACP) and reason about failover behaviour.
- Build Linux bridges and attach VMs/containers to them.

Topics covered

- Netplan general overview: YAML schema and renderers (systemd-networkd, NetworkManager)
- Device configuration IDs and how Netplan generates underlying configuration
- Netplan commands: netplan generate / apply / try
- VLAN trunking and tagged-VLAN configuration
- Linux bridges: model, MAC handling and STP
- Network interface bonding: modes (active-backup, 802.3ad/LACP, balance-xor), monitoring
- Layering bonds + VLANs + bridges in real-world topologies

Hands-on lab exercises

- Apply static IP addressing through Netplan + systemd-networkd.
- Create tagged VLAN interfaces and verify with ip / tcpdump.
- Build an active-backup bond and attach it to a Linux bridge.
- Layer VLANs on top of a bond; attach a bridge for VM/container use.

Module 9: Security: PAM, ACLs, AppArmor, and Ubuntu Pro

Production-grade Ubuntu has more security surface than “users and permissions”. This module covers the security frameworks that customers rely on: PAM for authentication, AppArmor for mandatory access control, ACLs for granular permissioning, and Ubuntu Pro for long-term security maintenance and compliance (USG/CIS, FIPS, ESM, Livepatch). The emphasis is on what each tool does, where it fits, and how to operate it without breaking the system.

What students will learn

- Describe and modify the PAM stack to enforce authentication policy.
- Apply POSIX ACLs to solve permission problems that classic Unix permissions can't.
- Build host-firewall rules with ufw and reason about nftables underneath.
- Confine applications with AppArmor profiles (complain and enforce modes).
- Enrol an Ubuntu host with Ubuntu Pro and enable Livepatch, ESM and compliance tooling.

Topics covered

- Pluggable Authentication Modules (PAM): the library, common modules, configuration files
- Common PAM stacks: common-auth, common-account, common-password, common-session
- Password strength policies (pam_pwquality, pam_cracklib)
- POSIX ACLs revisited in a security context: default ACLs, masks, inheritance
- Host firewalling on Ubuntu: ufw (the friendly front-end) and nftables (the underlying engine)
- AppArmor architecture: profile structure, modes (complain, enforce), aa-status / aa-genprof / aa-logprof
- Operating AppArmor: writing, loading and troubleshooting profiles
- Ubuntu Pro: ESM, Livepatch, FIPS, USG/CIS hardening tooling
- Audit and compliance basics on Ubuntu

Hands-on lab exercises

- Modify PAM to enforce password strength rules; verify with passwd.
- Add, remove and transfer ACL entries across a shared directory tree.
- Toggle an AppArmor profile between complain and enforce; observe denials in the journal.

- Attach the lab host to Ubuntu Pro; enable Livepatch and the USG hardening tool.

Module 10: Advanced Snap Packaging and Ubuntu Core

Snap is how Canonical delivers everything from desktop apps to entire operating systems. This module goes beyond basic snap install and into the operational details: confinement, interfaces, channels and tracks, and Ubuntu Core as the immutable, snap-based operating system for IoT and edge deployments. Students leave able to operate snaps in production, including channel pinning and refresh control.

What students will learn

- Explain snap internals: confinement, interfaces and how snap stores work.
- Operate snaps across channels and tracks with explicit pinning.
- Control snap refresh behaviour in production.
- Describe Ubuntu Core's architecture and where it fits in the portfolio.

Topics covered

- Snap confinement levels: strict, classic and devmode
- Snap interfaces and connections (network, home, removable-media, etc.)
- Snap channels and tracks: stable / candidate / beta / edge
- Refresh control: hold, refresh times, --hold-until, snap refresh --time
- Ubuntu Core: immutable rootfs, transactional updates, IoT-oriented design
- Use cases and trade-offs of snap vs. classic .deb on a server

Hands-on lab exercises

- Install snap packages from different channels; switch a running snap from stable to edge.
- Inspect and manage snap interfaces and connections.
- Pause and resume automatic refreshes; schedule refresh windows.
- (Optional) Explore an Ubuntu Core image in a VM.

Module 11: Time Synchronization

Time drift is the silent assassin of production systems. Logs go out of order, Kerberos tickets fail to validate, TLS certificates appear expired, and database replicas disagree about reality. This module covers the two clients Ubuntu uses to keep clocks synchronised, systemd-timesyncd for simple cases and chrony for anything serious, plus Network Time Security (NTS) for environments where the time source itself has to be authenticated.

What students will learn

- Choose between systemd-timesyncd and chrony for a given workload.
- Configure and operate chrony as a hardened NTP client and server.
- Diagnose clock drift, step events and source reachability problems.
- Configure NTS-secured time synchronization where the threat model requires it.

Topics covered

- Why time synchronization matters in production (auth, logs, certs, distributed systems)
- Network Time Protocol (NTP) fundamentals and stratum hierarchy
- Network Time Security (NTS): authenticated time over TLS
- systemd-timesyncd: configuration, limitations, when to leave it on
- chrony: architecture, chronyd/chronyc, drift files, makestep, NTP server mode
- Picking time sources: pool servers, internal stratum-1 references, NTS-capable servers
- Diagnosing time issues with chronyc tracking / sources / sourcestats and timedatectl

Hands-on lab exercises

- Inspect the current time source with timedatectl; observe systemd-timesyncd behaviour.
- Replace systemd-timesyncd with chrony; configure pool servers and verify sync.
- Configure chrony as an NTP server for the lab LAN.
- Configure an NTS-secured time source and verify authentication is in effect.