A COMPLETE VIDEO COURSE

RHEL 10.2 & Rocky Linux 10.2 Complete System Administration

From dnf and tmux to FIPS, post-quantum cryptography, Podman, bootc image mode, and multi-language container builds — for junior, intermediate, and senior administrators.



```
$ cat /etc/os-release
NAME="RHEL / Rocky"
VERSION="10.2"
$ _
```

- 60 Slides · 7 Modules
- Hands-on CLI Examples Throughout
- RHEL 10.2 & Rocky Linux 10.2 (Kernel 6.12)

COURSE ROADMAP

What You Will Learn



01 Foundations & Architecture

Containerization, RHEL vs Rocky, repositories

02 Package & System Management

dnf, dnf5, yum, modules, tmux, timedatectl, systemd

03 Networking

NetworkManager, nmcli, DHCP, firewalld

04 Security & Cryptography

SELinux, LUKS, SSL/TLS, crypto policies, FIPS, PQC, 2FA

05 Containers & Image Mode

Podman, building images, Quadlet, bootc

06 Development Toolchains

GCC, LLVM, Rust, Go, Node.js, Java, Python — host & container

+ BONUS MODULE 07 AI-Assisted Administration with RHEL Lightspeed, plus free chapters & the complete book

01

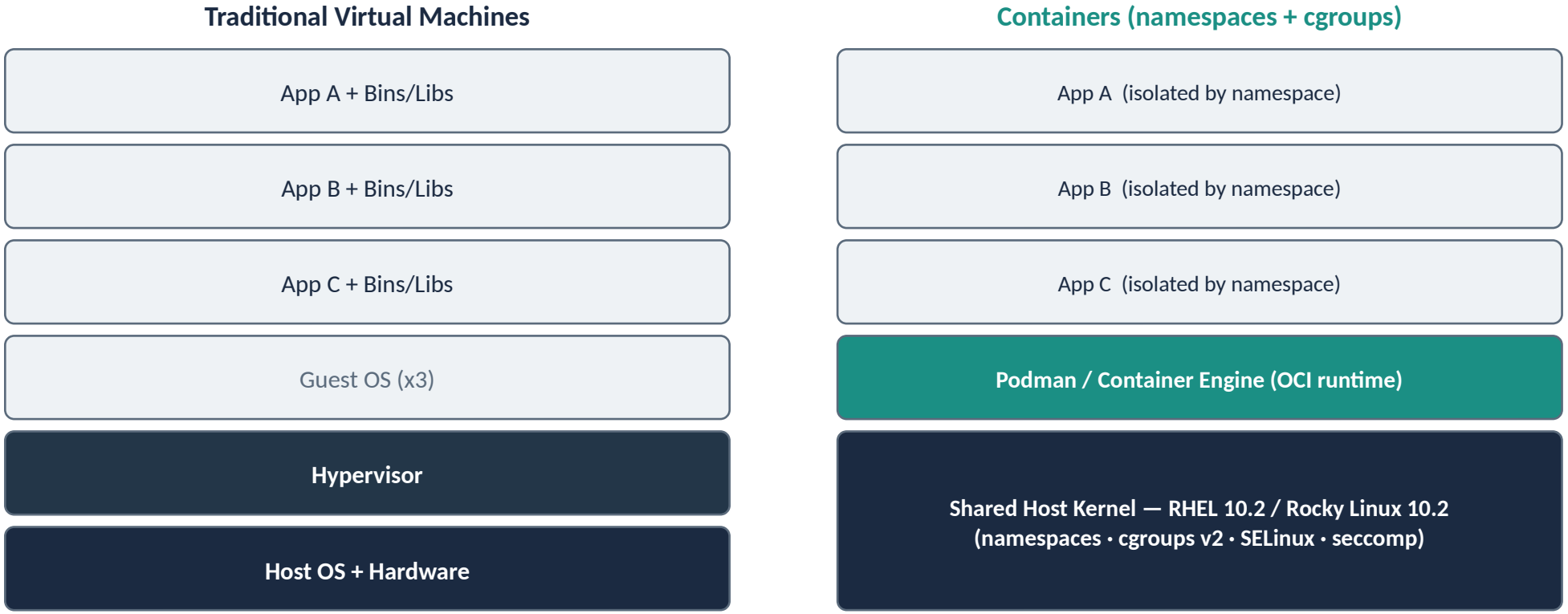
MODULE 1

Foundations & Architecture

Understand what containerization actually is at the kernel level, how RHEL 10.2 and Rocky Linux 10.2 relate to each other, and how the package repository model is organized.

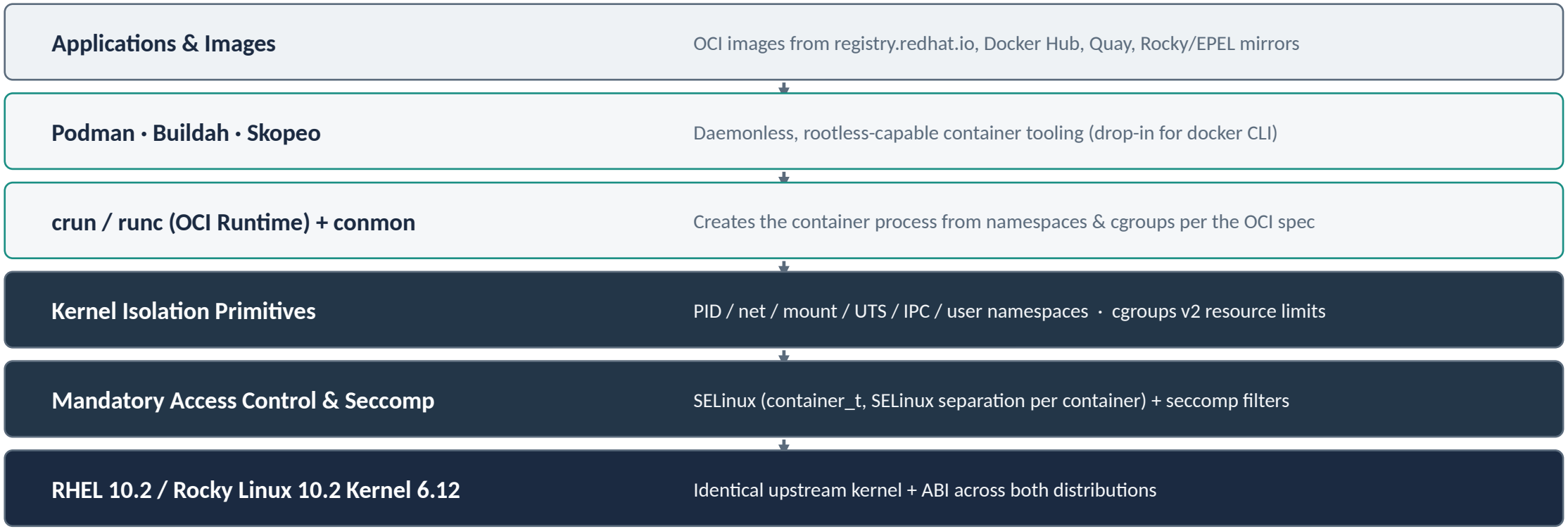
Containerization | Namespaces & cgroups | RHEL vs Rocky | BaseOS | AppStream | EPEL

What Is Containerization?



No guest OS or hypervisor — containers share one kernel and start in milliseconds, not minutes.

How RHEL & Rocky Implement Containers



bootc / Image Mode ships this entire stack — kernel to OS config — as a bootable OCI container image.

RHEL vs. Rocky Linux: Same Kernel, Different Model

Red Hat Enterprise Linux 10.2

- Commercial subscription required for updates & support
- Backed by Red Hat support, certifications & partner ecosystem
- Source: CentOS Stream 10 upstream development
- Ships Red Hat-branded tooling: Insights, Lightspeed, Satellite
- registry.redhat.io for certified container images

Rocky Linux 10.2

- Free, community-governed, rebuilt from RHEL's public sources
- Backed by the Rocky Enterprise Software Foundation (RESF)
- Bit-for-bit binary compatible with RHEL at the ABI level
- No subscription needed; ideal for labs, edu, and production at scale
- Same kernel 6.12, same dnf/podman/bootc tooling

Everything in this course — dnf, tmux, nmcli, podman, bootc — behaves identically on both distributions.

The Enterprise Linux 10 Platform at a Glance

6.12	x86-64-v3	4	10.2
Upstream Linux kernel	New baseline CPU microarch.	Supported architectures	Current minor release

- Architectures: x86-64-v3, ARMv8.0-A (aarch64), IBM Power10+ (ppc64le), IBM Z / LinuxONE (s390x, z15+)
- In-place upgrade paths: RHEL 9.6 to 10.0, and RHEL 9.8 to 10.2 (8 to 10 requires an intermediate 9.x hop)
- Rocky Linux 10 does not support cross-major upgrades — fresh install + restore from backup
- Kernel Livepatching is now available in RHEL 10, letting you patch running kernels without a reboot

Repository Landscape: BaseOS, AppStream, EPEL, CRB

BaseOS

Core OS packages needed to boot & run the system: kernel, glibc, systemd, coreutils. Long-term ABI stability.

AppStream

Applications, languages & databases: Python, Node.js, PHP, PostgreSQL 18, MariaDB 11.8, GCC Toolset 15.

CRB (CodeReady Builder)

Rocky: 'CRB' / RHEL: 'CodeReady Linux Builder' — dev headers & static libs many EPEL packages need.

EPEL

Extra Packages for Enterprise Linux — community repo (Fedora-built) adding thousands of extra packages, not Red Hat supported.

```
$ dnf repolist all
$ dnf install epel-release && dnf config-manager --set-enabled crb
```

02

MODULE 2

Package & System Management

Master dnf, dnf5, and yum compatibility, application-stream modules, terminal multiplexing with tmux, accurate time with timedatectl, and the systemd service model.

dnf & dnf5 | yum | AppStream modules | tmux | timedatectl | systemd

DNF Fundamentals

- DNF (Dandified YUM) is the default package manager on RHEL/Rocky 8, 9, and 10
- Resolves dependencies via libsolv and reads repo metadata cached under /var/cache/dnf
- Transactions are atomic and can be undone with dnf history undo
- Every command below works identically on RHEL 10.2 and Rocky Linux 10.2

dnf-basics.sh

```
$
dnf install httpd
$
dnf remove httpd
$
dnf update --refresh
$
dnf search "web server"
$
dnf info httpd
$
dnf list installed | grep kernel
$
dnf history
$
dnf history undo 12
$
```

NOTE dnf -y assumes 'yes' to every prompt — useful in scripts, but review transactions in production first.

DNF5: What's New

- DNF5 is a full C++ rewrite of DNF — faster, lower memory footprint, single shared library (libdnf5)
- RHEL 9 shipped DNF4; RHEL 10 and current Rocky Linux 10 make dnf5 the default 'dnf' binary
- Unifies dnf, dnf-automatic, and the microdnf container tool onto one codebase
- Adds a structured JSON output mode (--json) that scripts can parse reliably
- The command surface you already know stays almost entirely the same

dnf5-checks.sh

```
$
dnf --version
  dnf5 version 5.x.x
$
dnf5 --json list installed nodejs
$
dnf needs-restarting -r
$
dnf5 repoquery --duplicated
# Legacy plugins are being ported;
# check plugin compatibility before
# heavy dnf5 plugin automation
```

YUM vs. DNF: Legacy Compatibility

yum (legacy, RHEL/CentOS 6-7 era)

- Original Python-based package manager
- Slower dependency resolution on large transactions
- Still invoked as a symlink/alias on RHEL 8+ for muscle memory & old scripts
- Config lives in /etc/yum.conf and /etc/yum.repos.d/

dnf / dnf5 (current)

- Modern resolver (libsolv), better handling of weak dependencies
- dnf5 (RHEL/Rocky 10) rewritten in C++ for speed and a single shared library
- 'yum' is a compatibility symlink to dnf on RHEL/Rocky 8, 9 and 10
- Config still read from /etc/dnf/dnf.conf and /etc/yum.repos.d/*.repo

Muscle memory tip: every 'yum ...' command you type on RHEL/Rocky 8-10 is quietly executed by dnf.

AppStream Application Streams

- AppStream lets multiple versions of fast-moving software ship alongside the stable BaseOS core
- RHEL 10 significantly simplified modularity — most languages install as regular packages, with fewer competing streams than RHEL 8/9
- Where streams still exist (e.g. some database engines), only one stream can be enabled system-wide at a time
- Container images remain the recommended way to run multiple language versions side by side

appstream.sh

```
$
dnf module list
$
dnf module list nodejs
$
dnf module enable nodejs:22
$
dnf install nodejs
$
dnf module reset nodejs
$
dnf install python3.13 python3.13-devel
$
dnf install postgresql-server   # PostgreSQL 18
$
dnf install mariadb-server      # MariaDB 11.8
```

Repository Management & GPG Trust

- Repo definitions live as .repo files in /etc/yum.repos.d/
- Every RPM should be verified against a Red Hat / Rocky signing key before install
- dnf config-manager replaces the old yum-config-manager plugin syntax
- Priorities and excludes let you control which repo 'wins' when packages overlap (e.g. EPEL vs AppStream)

repo-mgmt.sh

```
$
dnf config-manager --add-repo
https://example.com/repo
$
dnf config-manager --set-enabled crb
$
rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-
rockyofficial
$
rpm -K package.rpm
$
dnf install 'dnf-command(config-manager)'
# /etc/yum.repos.d/myrepo.repo
[myrepo]
name=My Repo
baseurl=https://example.com/el10/x86_64/
gpgcheck=1
priority=10
```

Updates, Security Patches & Automation

- dnf updateinfo surfaces errata classified as security, bugfix, or enhancement
- dnf-automatic applies updates unattended on a systemd timer — ideal for patch windows
- RHEL Lightspeed (all RHEL subscriptions) can proactively flag known issues before they bite
- Always snapshot / take a LUKS-aware kdump-ready backup before major or kernel updates

patching.sh

```
$
dnf updateinfo list security
$
dnf update --security
$
dnf install dnf-automatic
$
systemctl enable --now dnf-automatic-install.timer
# /etc/dnf/automatic.conf
apply_updates = yes
$
dnf needs-restarting
$
rhc insights register   # register with Red Hat
Lightspeed
```

tmux: Terminal Multiplexing for Admins

- tmux keeps sessions alive across SSH disconnects — critical for long-running admin tasks
- A session holds windows; a window holds panes you can split and resize
- The default prefix key is Ctrl+b, followed by a command key
- Detach with prefix + d and reattach later from any terminal, even a different laptop

tmux-basics.sh

```
$
tmux new -s deploy
$
tmux ls
$
tmux attach -t deploy
$
tmux kill-session -t deploy
# Inside tmux (prefix = Ctrl+b):
#  c  new window      %  split vertical
#  "  split horizontal   arrow keys  move between
panes
#  d  detach           [  scrollback / copy mode
```

timedatectl & chrony: Accurate Time

- timedatectl is the systemd front-end for clock, timezone, and NTP sync status
- chronyd (updated to 4.8 in 10.2) is the default NTP client/server, replacing ntpd
- Accurate time is required for Kerberos, TLS certificate validation, and log correlation
- hwclock still exists for the hardware RTC, but timedatectl is the preferred modern interface

time-mgmt.sh

```
$
timedatectl status
$
timedatectl set-timezone America/Los_Angeles
$
timedatectl set-ntp true
$
timedatectl list-timezones | grep -i angeles
$
chronyc sources -v
$
chronyc tracking
$
systemctl status chronyd
```

systemd Fundamentals

systemd is PID 1 — the init system, service manager, and dependency graph for RHEL/Rocky 10.

U

Units

.service, .socket, .timer, .mount, .target files under /usr/lib/systemd/system

T

Targets

Group units into boot stages, replacing SysV runlevels (multi-user.target, graphical.target)

J

journald

Centralized binary logging — query with journalctl -u <service> -f

D

Dependencies

Wants=, Requires=, After=, Before= define the startup ordering graph

Managing systemd Services

- enable schedules a unit to start at boot; start actually runs it now — you often want both
- mask fully prevents a unit from starting, even manually, until unmasked
- systemctl edit creates a drop-in override without touching the vendor unit file
- journalctl is your first stop for troubleshooting any failed unit

systemd-ops.sh

```
$
systemctl enable --now httpd
$
systemctl status httpd
$
systemctl restart httpd
$
systemctl mask sendmail
$
systemctl edit httpd.service
$
systemctl daemon-reload
$
journalctl -u httpd --since "1 hour ago"
$
systemctl list-units --failed
```

03

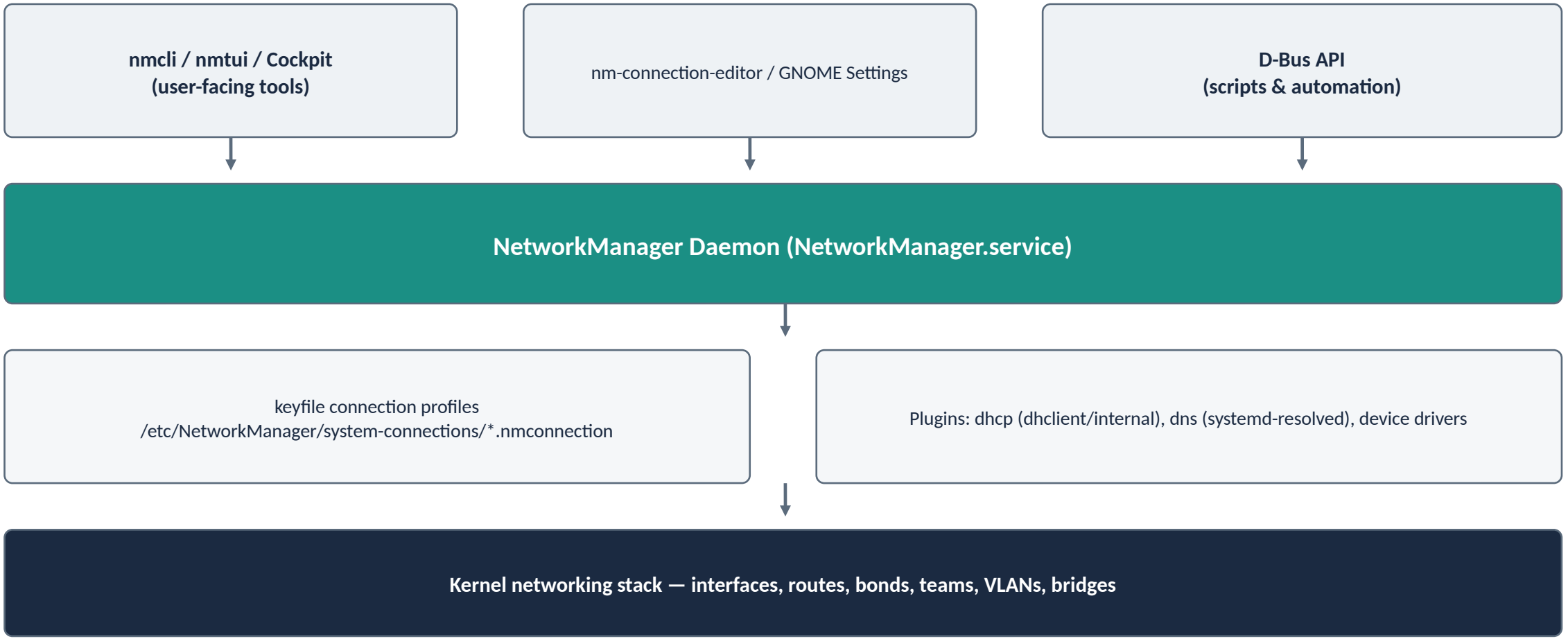
MODULE 3

Networking

Configure interfaces the modern way with NetworkManager and nmcli, hand out and request addresses with DHCP, and protect the host with firewalld.

NetworkManager | nmcli | Bonding & VLANs | DHCP | firewalld | nftables

NetworkManager Architecture



nmcli Basics

- nmcli is the command-line front end to NetworkManager — the default network stack
- Connection profiles are stored as keyfiles under /etc/NetworkManager/system-connections/
- Static IPv4 config, DNS, and gateway can all be set in one nmcli modify call
- Changes take effect on nmcli connection up, without needing a full reboot

nmcli-basics.sh

```
$
nmcli device status
$
nmcli connection show
$
nmcli connection add con-name eth0 ifname eth0 type
ethernet \
    ipv4.method manual ipv4.addresses 192.168.1.50/24
\
    ipv4.gateway 192.168.1.1 ipv4.dns 1.1.1.1
$
nmcli connection up eth0
$
nmcli device disconnect eth0
```

nmcli Advanced: Bonding, Teaming, VLANs & Bridges

- Bonding combines NICs for throughput or failover using kernel bonding drivers
- VLAN sub-interfaces tag traffic for 802.1Q trunked switch ports
- Bridges are essential for VM and container networking (libvirt, Podman networks)
- nmcli connection modify lets you tune bonding mode (active-backup, 802.3ad, etc.)

nmcli-advanced.sh

```
$
nmcli con add type bond con-name bond0 ifname bond0 \
    bond.options "mode=active-backup,miimon=100"
$
nmcli con add type ethernet ifname eth0 master bond0
$
nmcli con add type vlan con-name vlan10 dev eth1 id
10
$
nmcli con add type bridge con-name br0 ifname br0
$
nmcli con add type ethernet ifname eth2 master br0
```

DHCP: Client & Server

- NetworkManager's internal DHCP client (or dhclient) requests leases automatically
- dhcpd / kea-dhcp4 hand out addresses, gateways, DNS and options from a defined pool
- Static reservations bind a MAC address to a fixed IP for servers and printers
- Always check /var/log/messages or journalctl for lease negotiation failures

dhcp.sh

```
$
nmcli con modify eth0 ipv4.method auto
$
dhclient -v eth0
$
dnf install dhcp-server
# /etc/dhcp/dhcpd.conf
subnet 192.168.1.0 netmask 255.255.255.0 {
  range 192.168.1.100 192.168.1.200;
  option routers 192.168.1.1;
  option domain-name-servers 1.1.1.1;
}
$
systemctl enable --now dhcpd
```

firewalld & nftables

- firewalld is the default dynamic firewall manager, using zones to group trust levels
- Under the hood, firewalld now generates nftables rules (iptables-nft is a legacy shim)
- --permanent changes the saved config; --reload activates it without dropping connections
- Rich rules give fine-grained control beyond simple port/service toggles

firewalld.sh

```
$
firewall-cmd --get-active-zones
$
firewall-cmd --zone=public --add-service=https --
permanent
$
firewall-cmd --zone=public --add-port=8443/tcp --
permanent
$
firewall-cmd --reload
$
firewall-cmd --list-all
$
nft list ruleset | less
```

04

MODULE 4

Security & Cryptography

Go deep on SELinux, disk and transport encryption, system-wide crypto policies, FIPS 140-3 compliance, the new post-quantum algorithms in 10.2, and two-factor authentication.

SELinux | LUKS & Clevis/Tang | SSL/TLS | Crypto Policies | FIPS 140-3 | Post-Quantum | 2FA | IdM

SELinux Fundamentals

SELinux enforces Mandatory Access Control — even root is bound by policy, containing breaches that discretionary permissions alone would allow.

E

Enforcing / Permissive

Enforcing blocks and logs denials; permissive only logs — use for testing new policy

C

Contexts

Every file, process & port carries a label: user:role:type:level (e.g. httpd_sys_content_t)

B

Booleans

Toggle policy behavior at runtime without editing policy, e.g. httpd_can_network_connect

A

Audit & Alerts

Denials land in /var/log/audit/audit.log, readable with ausearch and sealert

SELinux Troubleshooting

- restorecon fixes mislabeled files after a copy/move breaks the expected context
- semanage lets you persist a custom port or file-context mapping across relabels
- audit2allow turns real denials into a custom policy module — use sparingly, review first
- Never disable SELinux in production to 'fix' an error — find and fix the labeling instead

selinux-troubleshoot.sh

```
$
getenforce
$
setenforce 0        # temporary, testing only
$
ls -Z /var/www/html
$
restorecon -Rv /var/www/html
$
semanage port -a -t http_port_t -p tcp 8080
$
setsebool -P httpd_can_network_connect on
$
ausearch -m avc -ts recent
$
audit2allow -a -M mypolicy && semodule -i mypolicy.pp
```

Encryption in RHEL & Rocky: The Big Picture



Data at Rest

- LUKS2 full-disk / volume encryption
- Clevis + Tang — automated network-bound unlock (NBDE)
- fscrypt — per-directory file encryption

Data in Transit

- TLS/SSL via OpenSSL 3.x
- OpenSSH with ML-KEM post-quantum key exchange (10.2)
- IPsec / WireGuard tunnels

Key & Trust Management

- System-wide crypto policies (DEFAULT/FUTURE/FIPS/LEGACY)
- p11-kit + PKCS#11 tokens & smart cards
- FIPS 140-3 validated modules

Every layer is enforced consistently across both RHEL and Rocky Linux — same OpenSSL, same crypto-policies tooling.

LUKS Disk Encryption

- LUKS2 is the default on-disk format for encrypted volumes on RHEL/Rocky 10
- A LUKS header can hold multiple key slots — different passphrases, or a Clevis-bound key
- Anaconda can enable disk encryption at install time; cryptsetup manages it afterward
- Always back up the LUKS header — losing it makes the encrypted data unrecoverable

luks.sh

```
$
cryptsetup luksFormat /dev/sdb1
$
cryptsetup luksOpen /dev/sdb1 secure_data
$
mkfs.xfs /dev/mapper/secure_data
$
cryptsetup luksAddKey /dev/sdb1
$
cryptsetup luksHeaderBackup /dev/sdb1 \
    --header-backup-file /root/sdb1-header.img
$
cryptsetup luksClose secure_data
```

Network-Bound Disk Encryption: Clevis + Tang

- NBDE lets a LUKS volume unlock automatically at boot only while it can reach a Tang server
- Tang never stores or sees the encryption key — it only helps derive it (McCallum-Relyea exchange)
- clevis-pin-trustee (new in 10.2) adds remote-attestation-based unlock via the Trustee Key Broker Service
- Removing a server from the network instantly makes its disk unreadable without the Tang key

nbde.sh

```
# On the Tang server
$
dnf install tang && systemctl enable --now
tangd.socket
# On the client with an existing LUKS volume
$
dnf install clevis clevis-luks
$
clevis luks bind -d /dev/sdb1 tang
'{"url":"http://tang.local"}'
$
clevis luks list -d /dev/sdb1
$
dracut -fv --regenerate-all   # enable clevis in
initramfs
```

SSL/TLS with OpenSSL

- OpenSSL 3.x provides the cryptographic engine behind Apache, nginx, Postfix and more
- A private key + CSR (Certificate Signing Request) is the standard path to a signed cert
- openssl s_client is invaluable for verifying what a remote server actually presents
- System-wide crypto policies (next slides) constrain which TLS versions and ciphers OpenSSL allows

openssl.sh

```
$
openssl genpkey -algorithm RSA -out server.key -
pkeyopt rsa_keygen_bits:4096
$
openssl req -new -key server.key -out server.csr
$
openssl x509 -in server.crt -noout -text
$
openssl s_client -connect example.com:443 -tls1_3
$
openssl verify -CAfile ca.crt server.crt
```

System-Wide Cryptographic Policies

LE

LEGACY

Maximum compatibility. Allows older, weaker algorithms (SHA-1, small DH). Avoid on new systems.

DE

DEFAULT

Secure, broadly-compatible baseline used out of the box on RHEL/Rocky 10.2.

FU

FUTURE

Stricter forward-looking policy — in 10.2, restricts key exchange to hybrid ML-KEM only. Breaks non-PQC peers.

FI

FIPS

Restricts the system to FIPS 140-3 validated algorithms and modules only.

```
$ update-crypto-policies --set FIPS && reboot
$ update-crypto-policies --show
```

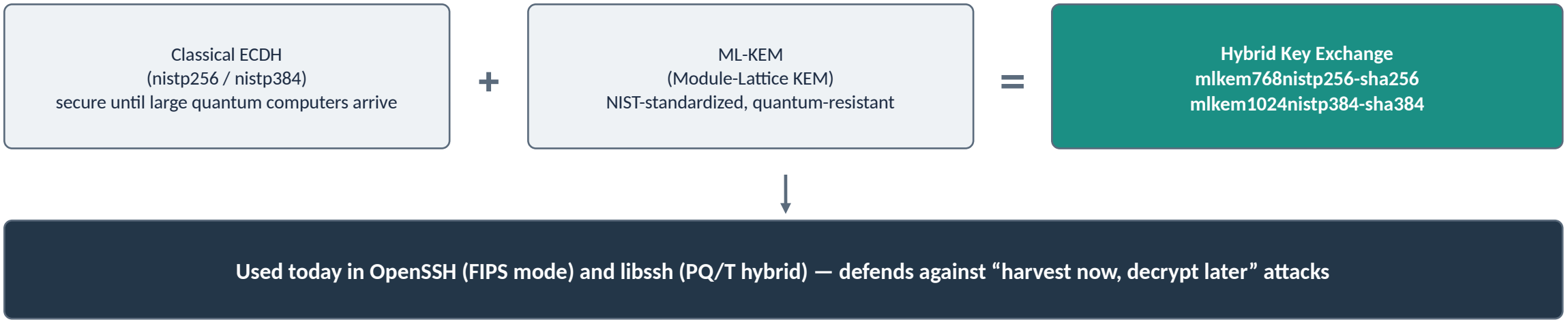
FIPS 140-3 Compliance Deep Dive

140-3	1	0
Current FIPS validation standard	Command to enable at install	Non-validated algorithms allowed

- FIPS mode restricts the entire system — OpenSSL, GnuTLS, NSS, libgcrypt, kernel crypto — to validated modules and algorithms
- Enable at install time with fips=1 on the kernel command line, or post-install with fips-mode-setup --enable then reboot
- Verify current state anytime with fips-mode-setup --check or cat /proc/sys/crypto/fips_enabled
- Required for U.S. federal, healthcare, and financial workloads under FISMA, HIPAA-adjacent, and PCI-DSS mandates

Post-Quantum Cryptography: Hybrid Key Exchange

NEW IN RHEL / ROCKY 10.2



Also new in 10.2: Red Hat Certificate System 11.0 issues quantum-resistant signatures · p11-kit 0.26.1 adds PQC definitions in PKCS #11 · podman-sequoia supports composite post-quantum signatures.

Two-Factor Authentication: TOTP with PAM

- google-authenticator-libpam adds a time-based one-time password (TOTP) factor to any PAM service
- Works with FreeOTP, Google Authenticator, or any RFC 6238-compliant app
- Stack it in /etc/pam.d/sshd (or sudo, login) as a required line after password auth
- Print backup/emergency scratch codes and store them somewhere other than the server itself

totp-2fa.sh

```
$
dnf install google-authenticator-libpam
$
google-authenticator   # per-user setup, generates QR
code
# /etc/pam.d/sshd
auth required pam_google_authenticator.so
# /etc/ssh/sshd_config
KbdInteractiveAuthentication yes
AuthenticationMethods publickey,keyboard-interactive
$
systemctl restart sshd
```

Two-Factor Authentication: FIDO2 & Smart Cards

- pam_u2f enables hardware security keys (YubiKey, etc.) as a phishing-resistant second factor
- sssd + p11-kit support PIV/CAC smart cards for full certificate-based login (common in government/defense)
- FIDO2 keys never expose a shared secret to the server — resistant to credential replay
- Combine with FIPS mode for authentication stacks that must meet federal identity assurance levels

fido2-smartcard.sh

```
$
dnf install pam_u2f
$
pamu2fcfg > ~/.config/Yubico/u2f_keys
# /etc/pam.d/sudo
auth required pam_u2f.so
$
dnf install sssd sssd-tools opensc
$
authselect select sssd with-smartcard --force
$
sss_cache -E
```

Identity Management (IdM / FreeIPA)

IdM centralizes users, hosts, and policy across a fleet — the RHEL/Rocky equivalent of an enterprise directory service.

K

Kerberos KDC

Single sign-on tickets replace repeated password prompts across joined hosts

L

389 Directory Server

LDAP-compatible user & group store, the backbone of centralized accounts

C

Certificate Authority (Dogtag)

Issues and manages TLS certs for services domain-wide, integrated with 10.2's PQC signatures

S

HBAC & sudo Rules

Host-based access control decides who can log in where, and with what sudo rights

05

MODULE 5

Containers & Image Mode

Go hands-on with Podman, build your own OCI images, run multi-container pods as systemd services with Quadlet, and treat the OS itself as a bootable container with bootc.

Podman | Buildah | Containerfile | Quadlet | bootc / Image Mode

Container Internals: Namespaces, cgroups & OCI



Linux namespaces isolate what a process can SEE (its own PIDs, network stack, filesystem mounts, hostname, IPC, and UIDs)

cgroups v2
limits what a process can USE:
CPU shares, memory, I/O, PIDs count

SELinux + seccomp
restricts what a process is ALLOWED
to do at the kernel/syscall level

OCI (Open Container Initiative) Spec — the contract that Podman, Buildah, CRI-O, and bootc all build against:
Image Spec (how a container image is packaged) + Runtime Spec (how crun/runc starts it) + Distribution Spec (how a registry serves it)

Podman Basics: Daemonless & Rootless

- Podman has no background daemon — each container is a direct child process, easing auditing
- Rootless mode maps container UIDs into an unprivileged user namespace — a compromised container is not root on the host
- The CLI is intentionally docker-compatible: alias docker=podman works for most workflows
- podman-sequoia (10.2) now backs image signing with post-quantum-ready composite signatures

podman-basics.sh

```
$
podman run -d --name web -p 8080:80
registry.access.redhat.com/ubi10/httpd-24
$
podman ps
$
podman logs -f web
$
podman exec -it web bash
$
podman stop web && podman rm web
$
podman run --rm -it --userns=keep-id ubi10/ubi bash
```

Building Container Images

- A Containerfile (Dockerfile-compatible syntax) describes how to assemble an image layer by layer
- Multi-stage builds keep your final runtime image small by discarding build-only tools
- podman build and buildah bud both consume the same Containerfile syntax
- Tag and push to any OCI registry — Quay.io, registry.redhat.io, or a private Quay instance

Containerfile

```
FROM registry.access.redhat.com/ubi10/ubi:latest AS
build
RUN dnf install -y python3.13 python3.13-devel gcc &&
\
    pip3.13 install --prefix=/install -r
requirements.txt
FROM registry.access.redhat.com/ubi10/ubi-
minimal:latest
COPY --from=build /install /usr/local
COPY app.py /app/app.py
CMD ["python3.13", "/app/app.py"]
$
podman build -t myapp:1.0 .
$
podman push myapp:1.0 quay.io/myorg/myapp:1.0
```

Pods & Quadlet: Containers as systemd Services

- A Podman pod groups containers sharing one network namespace — the Kubernetes pod model, locally
- podman generate systemd is deprecated in favor of Quadlet — declarative .container unit files
- Quadlet units live in ~/.config/containers/systemd/ (rootless) and are managed with plain systemctl
- Gives you auto-restart, dependency ordering, and journald logging for free

web.container

```
$
podman pod create --name mypod -p 8080:80
$
podman run -d --pod mypod nginx
$
podman run -d --pod mypod myapp-sidecar
# ~/.config/containers/systemd/web.container
[Container]
Image=registry.access.redhat.com/ubi10/httpd-24
PublishPort=8080:80
[Install]
WantedBy=default.target
$
systemctl --user daemon-reload && systemctl --user
start web
```

bootc: Image Mode for Linux

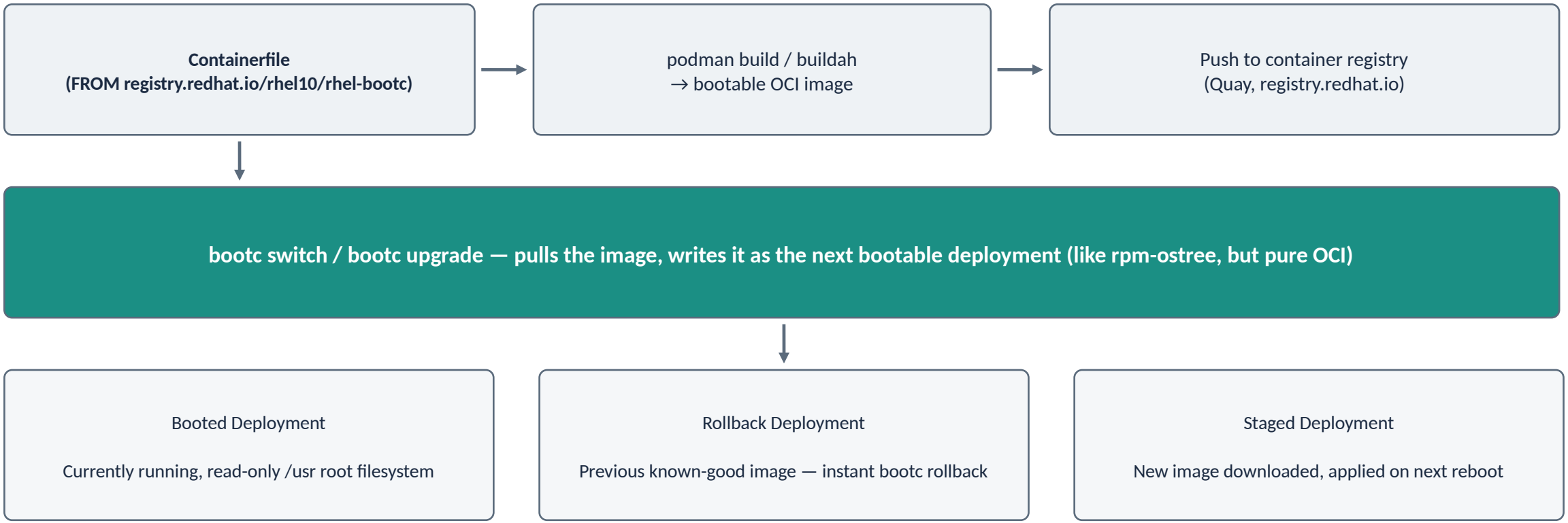


Image Mode treats the OS itself as a versioned, immutable container image — updates are atomic, and rollback is instant.

bootc Hands-On: Switch, Upgrade, Rollback

- bootc turns the operating system itself into a versioned OCI container image (Image Mode for Linux)
- Rebase to a new image with bootc switch; pull the latest tag of the current image with bootc upgrade
- A bad update rolls back instantly with bootc rollback — no snapshot tooling required
- 10.2 adds a Technology Preview of sealed images for stronger integrity guarantees on Image Mode systems

bootc.sh

```
$
bootc status
$
bootc switch registry.redhat.io/rhel10/rhel-
bootc:10.2
$
bootc upgrade
$
bootc rollback && systemctl reboot
# Build your own bootc image:
$
podman build -t my-bootc-os -f Containerfile.bootc .
$
podman push my-bootc-os quay.io/myorg/my-bootc-
os:latest
```

06

MODULE 6

Development Toolchains

Tour the compiler and language ecosystem shipped in RHEL/Rocky 10.2 — GCC Toolset 15, LLVM, Rust, Go, Node.js, Java, and Python — both directly on the host and packaged inside containers.

GCC Toolset 15 | LLVM 21 | Rust 1.92 | Go 1.26 | Node.js 24 | OpenJDK | Python 3.13

Compiler & Language Toolsets in 10.2

System Toolchain (always present)

- GCC 14.3 · glibc 2.39 · Binutils 2.41 — the base compiler used to build the OS itself
- GDB 16.3, Valgrind 3.26, SystemTap 5.4 for debugging & profiling
- elfutils 0.194 and libabigail 2.9 for ABI compatibility checking

Optional Toolsets (dnf install ...)

- GCC Toolset 15 -> GCC 15.2 + Binutils 2.44 (Annobin/dwz dropped from this toolset)
- LLVM Toolset 21.1.8 -> clang / clang++ / lldb
- Rust Toolset 1.92.0 and Go Toolset 1.26.2

Toolsets install side-by-side with the system compiler and are activated per-shell via SCL enable scripts.

GCC / G++ Demo Across Versions

- The base system GCC 14.3 is always available as gcc / g++ with no extra install
- GCC Toolset 15 installs alongside it under /opt/rh/gcc-toolset-15/ without replacing the system compiler
- scl enable (or sourcing the enable script) puts the newer toolchain first on PATH for that shell only
- This lets you validate code against two GCC major versions on one machine

gcc-demo.sh

```
$
gcc --version        # 14.3 (system default)
$
dnf install gcc-toolset-15
$
scl enable gcc-toolset-15 -- gcc --version   # 15.2
$
source /opt/rh/gcc-toolset-15/enable
$
g++ -std=c++23 -O2 -Wall -o app main.cpp
$
./app
```

Node.js & Python: Application Runtimes



- Node.js 24 is the current Application Stream release available in 10.2
- Python ships several parallel versions as plain packages — 3.9, 3.11, 3.12 (default), 3.13
- python3 -m venv keeps project dependencies isolated from the system interpreter
- npm and pip both work exactly as upstream — no distro-specific quirks to relearn

node-python.sh

```
$
dnf install nodejs npm
$
node --version && npm --version
$
npm init -y && npm install express
$
dnf install python3.13 python3.13-devel
$
python3.13 -m venv ~/venvs/app
$
source ~/venvs/app/bin/activate
$
pip install --upgrade pip requests
```

Java Development with OpenJDK

- OpenJDK ships as versioned packages (java-17-openjdk, java-21-openjdk, etc.) rather than a single stream
- alternatives manages which JDK/JRE answers to the plain java and javac commands
- JAVA_HOME can also be pinned explicitly in a project's build environment or container image
- Maven and Gradle are available via AppStream/EPEL for full build-tool workflows

java-demo.sh

```
$
dnf install java-21-openjdk-devel java-17-openjdk-
devel
$
alternatives --config java
$
alternatives --config javac
$
java -version
$
javac HelloWorld.java && java HelloWorld
$
dnf install maven
$
mvn -version
```

Compilers & Runtimes: Host vs. Container



On the Host OS

- dnf install gcc-toolset-15
- scl enable gcc-toolset-15 -- bash
- One toolchain version active at a time (via SCL)
- Shares host kernel & libraries directly

Inside a Container (UBI base image)

- FROM registry.access.redhat.com/ubi10/ubi
- dnf install gcc gcc-c++ (per-image version)
- Every image pins its own toolchain — no conflicts
- Multi-stage builds strip the compiler from the final runtime image

```
FROM registry.access.redhat.com/ubi10/ubi:latest AS build
RUN dnf install -y gcc-toolset-15 && \
    source /opt/rh/gcc-toolset-15/enable && g++ -O2 -o app main.cpp
FROM registry.access.redhat.com/ubi10/ubi-micro:latest
COPY --from=build /app /usr/local/bin/app
```

Building a Multi-Language Container Image

- A single Containerfile can stage multiple compiled artifacts before assembling a minimal runtime layer
- Each build stage can use a completely different base image and toolchain version
- Only the compiled outputs are copied into the final image — no compilers ship to production
- This pattern keeps image size, attack surface, and CVE exposure to a minimum

Containerfile.multi

```
FROM registry.access.redhat.com/ubi10/ubi AS gobuild
RUN dnf install -y go-toolset && go build -o /out/svc
./cmd/svc

FROM registry.access.redhat.com/ubi10/ubi AS cppbuild
RUN dnf install -y gcc-toolset-15 && \
    source /opt/rh/gcc-toolset-15/enable && g++ -O2 -
o /out/worker worker.cpp

FROM registry.access.redhat.com/ubi10/ubi-micro
COPY --from=gobuild /out/svc /usr/local/bin/svc
COPY --from=cppbuild /out/worker
/usr/local/bin/worker
```

Observability, High Availability & AI-Assisted Admin

P

perf & BPF Tooling

10.2 aligns perf closer to upstream, adds debuginfod support and new Intel/AMD PMU events

K

Kernel Livepatching

Apply critical kernel security fixes without rebooting — new capability in RHEL 10

H

Pacemaker & Corosync

Cluster resource management for HA services — shared-storage and failover clusters

L

RHEL Lightspeed

AI-assisted troubleshooting and update guidance bundled with every RHEL subscription

You're Ready to Administer RHEL 10.2 & Rocky Linux 10.2

From package management to post-quantum cryptography and bootable containers — the full stack, junior to senior.

Practice	Rebuild every command shown in a lab VM — muscle memory beats memorization.
Go Deeper	Read the official RHEL 10.2 / Rocky Linux 10.2 release notes for full technical detail.
Stay Current	Both distributions ship security & feature updates on their own minor-release cadence.
Thank You	Questions and feedback are always welcome — happy administering!

07

MODULE 7

AI Assist & Free Learning Resources

One more module: see Red Hat's AI-powered troubleshooting assistant in action, then find out how to keep learning for free — including this course creator's complete 43-chapter Linux book, in English and Dari.

RHEL Lightspeed | AI Assist | 43 Free Chapters | eBook / Paperback / Hardcover | Subscribe & Share

RHEL Lightspeed: How AI Assist Works

Lightspeed pairs a generative-AI model with Red Hat's own knowledgebase and your system's real state — included with every RHEL subscription (it is a RHEL-exclusive benefit, not available on Rocky Linux).

C

Command-Line Assistant

The 'c' command answers natural-language questions right at the shell prompt and proposes exact commands to run

S

System-Aware Diagnosis

Reads local logs, systemd unit status, and installed package versions before answering — not generic advice

K

Red Hat Knowledgebase Grounding

Answers are grounded in Red Hat's own KCS solutions and docs, reducing hallucinated commands

A

Opt-In & Auditable

Every suggested command is shown before it runs — nothing executes without your explicit confirmation

RHEL Lightspeed: A Real Troubleshooting Example

- Install the command-line assistant once per registered RHEL host, then just ask in plain English
- Lightspeed inspects the failed unit's actual journal output before it answers
- It proposes the exact remediation command — you decide whether to run it
- A great way for junior admins to learn the reasoning behind a fix, not just the fix itself

lightspeed-demo.sh

```
$
dnf install command-line-assistant
$
c "why did httpd fail to start?"
  checking systemctl status httpd.service ...
  checking journalctl -u httpd --since -10m ...
  -> port 80 is already in use by nginx.service
  suggested fix:
    systemctl stop nginx && systemctl start httpd
$
c --execute "apply the suggested fix"   # runs only
after you confirm
```

NOTE RHEL Lightspeed requires an active, registered RHEL subscription — it is not available on Rocky Linux.

Read the Complete 43-Chapter Linux Book — Free

43	2	3
Chapters free to read online	Languages: English & Dari	Starter chapters also as PDF

- All 43 chapters of "The Direct Path to Linux Ubuntu" are free to read online, in both English and Dari — no signup required
- The Preface + first 3 chapters are also offered as a downloadable PDF, formatted exactly like the printed book — a perfect starter kit
- Companion titles "The Direct Path to RHEL 10" and "The Direct Path to Enterprise Rocky Linux 10" are coming soon
- Every link lives in one place: the MyWebUniversity.com homepage

Prefer Print or Kindle? Get the Complete Book

Amazon KDP

- eBook — $9.99
- Paperback — $34.99
- Hardcover — $44.99

Barnes & Noble Press

- eBook — $9.99
- Paperback — $24.99
- Hardcover — $44.99

Available in English, with a full Dari-language edition too — every link is on the MyWebUniversity.com homepage.

Enjoyed This Course? Let's Stay Connected

Free Linux, DevOps, and Enterprise Linux training — new videos every week.

Subscribe	@MyWebUniversity (English) and @MyWebUniversity-Dari (Dari) — 2,600+ free videos and counting
Comment	Tell us which RHEL or Rocky Linux topic you want covered next
Share	Send this video to a teammate studying for their next Linux certification
Read & Buy the Book	Free chapters, the full 43-chapter book, and every link — all on MyWebUniversity.com