The Real Differences Between Ubuntu and Red Hat Enterprise Linux in 2026

May 12, 2026 / Gavin Jackson

linux ubuntu red-hat rhel enterprise-linux satellite landscape ansible selinux apparmor kickstart autoinstall

Moving from an Ubuntu shop to a Red Hat shop is not really a move from "Linux" to "Linux".

At the shell, plenty of things feel familiar. You still have systemd, SSH, journald, OpenSSL, OpenSSH, Python, containers, cron, sudo, rsyslog, and all the usual small tools that make Linux feel like Linux. Most operational instincts still transfer.

The real shift is the enterprise operating model around the distribution.

Ubuntu and Red Hat Enterprise Linux both solve the same broad problem: provide a supported, secure, stable Linux platform for production workloads. But they make different tradeoffs around packaging, support boundaries, fleet management, security controls, automated installation, and lifecycle discipline.

In 2026, that difference matters more than the old "apt vs yum" muscle memory.

The current Ubuntu baseline is Ubuntu 26.04 LTS, released on 23 April 2026 and supported as an LTS release for five years, with extended coverage available through Ubuntu Pro. The current Red Hat baseline is Red Hat Enterprise Linux 10, released in May 2025, with the usual long enterprise lifecycle model behind it.

This is the practical comparison I would want in front of me before moving from Ubuntu to RHEL.

The short version

If you already manage Ubuntu well, you are not starting again.

But you do need to re-learn a few habits:

- Landscape becomes Satellite, or at least Red Hat Satellite plus Red Hat Hybrid Cloud Console and Lightspeed.
- `apt` and `dpkg` become `dnf` and `rpm`.
- Ubuntu repository categories become RHEL repositories such as BaseOS, AppStream, CodeReady Linux Builder, Supplementary, and add-on repositories.
- AppArmor muscle memory becomes SELinux muscle memory.
- Ubuntu Autoinstall becomes Kickstart.
- `ufw` often becomes `firewalld`.
- Netplan habits often become NetworkManager profile habits.
- Package names change, service names change, and some old assumptions about packages starting services need to be checked.
- Ansible still works very well, but the supported Red Hat automation story is more explicitly tied to Red Hat Ansible Automation Platform, RHEL system roles, Satellite, and certified collections.

That last point is the key theme. Red Hat is not just a distro. It is an ecosystem built around entitlement, content lifecycle, compliance, supportability, and automation at scale.

Ubuntu can absolutely be enterprise-grade, especially with Ubuntu Pro and Landscape. But Red Hat expects you to manage the operating system as part of a more formal content and compliance pipeline.

Baseline assumptions in 2026

For Ubuntu, the enterprise comparison should mostly mean Ubuntu LTS, not interim releases. Ubuntu interim releases are useful for newer kernels and toolchains, but production fleets normally standardise on LTS releases.

Ubuntu's release cycle documentation describes the model clearly: LTS releases arrive every two years and receive five years of standard security maintenance. Ubuntu Pro extends security coverage and adds enterprise features such as ESM, Livepatch, Landscape, FIPS packages, and compliance tooling.

For Red Hat, the comparison is RHEL 9 and RHEL 10, but if you are planning a new environment in 2026, RHEL 10 is the obvious strategic baseline unless an application certification matrix says otherwise. Red Hat's RHEL lifecycle policy says RHEL 8, 9, and 10 have a ten-year lifecycle across Full Support and Maintenance Support phases, followed by an Extended Life Phase.

The high-level difference:

Area	Ubuntu LTS	Red Hat Enterprise Linux
Current enterprise baseline	Ubuntu 26.04 LTS	RHEL 10
Commercial vendor	Canonical	Red Hat
Primary package format	`.deb`	`.rpm`
Primary package tool	`apt`	`dnf`
Fleet management	Landscape	Satellite, Hybrid Cloud Console, Lightspeed
Default MAC system	AppArmor	SELinux
Automated install	Subiquity Autoinstall	Anaconda Kickstart
Firewall habit	`ufw`, raw nftables/iptables where needed	`firewalld`, nftables where needed
Networking habit	Netplan rendering to NetworkManager or systemd-networkd	NetworkManager
Automation story	Ansible, cloud-init, Landscape scripts, MAAS	Ansible Automation Platform, RHEL system roles, Satellite remote execution

The important part is not that one column is "better". It is that the operational centre of gravity moves.

> ### *Old Enough to Remember When Red Hat Came in a Box*
>
> *One reason this comparison feels interesting to me is that I am old enough to remember Red Hat before "Enterprise Linux" was the default association.*
>
> *When I was at university, Red Hat Linux 5.2 was a normal Linux distribution you could install, break, reinstall, and learn from. Red Hat's own 1998 release announcement described Red Hat Linux 5.2 as a boxed operating system with three CDs, an installation manual, 90 days of installation support, Apache 1.3, GIMP 1.0, and a retail price of US$49.95. That was a very different world. It was Linux as a thing you bought, borrowed, copied, installed on spare hardware, and gradually turned into a career.*
>
> *Then Linux grew up.*
>
> *Red Hat created what became Red Hat Enterprise Linux in 2002 for customers who wanted a stable, supported, certified operating system. The first RHEL 2.1 general availability date listed by Red Hat is 23 March 2002. By late 2003, Red Hat had also introduced the Fedora Project as the community and early-technology track, while RHEL became the stable commercial platform. Red Hat's own explanation of the split is pretty blunt: one product could no longer meet the conflicting needs of*

> *hobbyists, developers, and enterprise customers.*
>
> *Ubuntu arrived just after that moment. Ubuntu 4.10, "Warty Warthog", was announced on 20 October 2004 as the first Ubuntu release, with the now almost mythic offer to ship CDs at no cost. It was not officially "the anti-RHEL", and it would be too neat to pretend the history was that simple. But for a lot of Linux users, Ubuntu landed at exactly the right time: after the old Red Hat Linux line had given way to Fedora and RHEL, and when there was a real appetite for a polished, freely available, community-friendly Linux distribution that did not feel like it was asking for a procurement conversation before you could start learning.*
>
> *That history still echoes in 2026.*
>
> *IBM completed its acquisition of Red Hat in July 2019, and RHEL is now firmly part of a much larger enterprise software business. On sticker price alone, the comparison can look stark. As of May 2026, Red Hat lists RHEL Server at US$383.90 for self-support, US$878.90 for standard support, and US$1,428.90 for premium support, with Satellite bundles and other add-ons priced separately. Canonical lists Ubuntu Pro at US$500 per server per year, with 24/7 support tiers at US$1,775 for infrastructure support and US$3,400 for full support.*
>
> *Those numbers should not be compared too lazily. Red Hat's self-support tier is not intended for production, the products are packaged differently, the support models are different, and the real enterprise bill depends on architecture, virtualisation density, public cloud consumption, support level, management tooling, discounts, and procurement agreements.*
>
> *But the emotional shape of the comparison is hard to miss. Ubuntu still carries some of that "just let me run Linux" energy, even when wrapped in Ubuntu Pro, Landscape, FIPS, Livepatch, and compliance tooling. Red Hat still carries the weight of the platform that large enterprises trust when they need certification, lifecycle discipline, vendor accountability, Satellite-driven content management, and a very mature operating model at scale.*
>
> *That is why I would not reduce this to price. Ubuntu Pro can be very compelling commercially, especially for broad fleets and teams already comfortable with Ubuntu. Red Hat is harder to beat when the question becomes: how do we run thousands of systems through a controlled content lifecycle, with certified applications, formal support paths, mature policy tooling, and a management model that auditors and operations teams can both understand?*
>
> *In a strange way, the old Red Hat Linux 5.2 box and the modern RHEL subscription are part of the same story. Linux went from something we installed because it was exciting to something organisations run because it is boring in exactly the right ways.*

Landscape vs Satellite

This is probably the biggest enterprise difference.

If you manage Ubuntu at scale, you may already use Landscape, Canonical's systems management tool for Ubuntu machines. In 2026, Landscape is available as SaaS, self-hosted Landscape, and Managed Landscape. Canonical's own comparison is useful: SaaS gives inventory, security, compliance, hardening, and reporting, but self-hosted and managed options are where you get repository management and offline-friendly patterns.

That matters because "Landscape" is not one identical thing in every deployment. If you are used to Landscape SaaS and you move to RHEL, do not assume the Red Hat equivalent is just "a web console that shows machines". Satellite is more opinionated about content lifecycle.

Red Hat Satellite 6.17 is the classic Red Hat estate management tool. It handles host registration, content management, lifecycle environments, content views, repository synchronization, patching, provisioning, remote execution, host groups, Capsule servers, activation keys, Ansible integration, and compliance workflows.

The concepts to learn are:

- **Organization and location**: Satellite partitions management across business and infrastructure boundaries.
- **Content views**: curated sets of repositories and packages that can be promoted through lifecycle environments.
- **Lifecycle environments**: dev, test, prod, or whatever promotion path you define.
- **Activation keys**: registration profiles used to attach hosts to subscriptions, content views, environments, repository sets, and system purpose.
- **Capsule servers**: distributed Satellite components for content, provisioning, DNS, DHCP, TFTP, and remote execution closer to hosts.
- **Host groups**: repeatable host configuration and provisioning templates.
- **Remote execution**: run jobs against hosts from Satellite.
- **Ansible integration**: run roles and playbooks as part of host management.

The most important mental model is this:

Landscape often feels like Ubuntu fleet management. Satellite feels like a content supply chain.

Satellite wants you to think about which exact repository content a machine is entitled to see, which lifecycle stage that content is in, and how content moves from sync to test to production. Red Hat's Satellite docs describe activation keys as a way to automate registration and associate systems with environments and content views. That is not just a convenience feature. In a real RHEL estate, activation keys become part of how you encode server purpose.

If you are coming from Ubuntu, this is where I would spend time first. Package commands can be learned in an afternoon. A bad content lifecycle design can annoy you for years.

Package management: apt to dnf

At the daily command level, the translation is simple enough:

Ubuntu	RHEL
`apt update`	`dnf check-update` or let `dnf` refresh metadata as needed
`apt install nginx`	`dnf install nginx`
`apt remove nginx`	`dnf remove nginx`
`apt purge nginx`	no direct habit match; remove package and clean config deliberately
`apt search nginx`	`dnf search nginx`
`apt show nginx`	`dnf info nginx`
`dpkg -l`	`rpm -qa`
`dpkg -S /path/file`	`rpm -qf /path/file`
`apt-file search /path/file`	`dnf provides /path/file`
`apt-mark hold package`	`dnf versionlock package` if the plugin is installed
`/etc/apt/sources.list.d/*.sources`	`/etc/yum.repos.d/*.repo`

But again, the command syntax is the easy part.

Ubuntu's package model is Debian-based. The main supported base is in Main and Restricted, with Universe and Multiverse adding a large body of community-maintained software. The Ubuntu Server package management documentation notes that Ubuntu 24.04 and later use deb822 repository files such as `/etc/apt/sources.list.d/ubuntu.sources` by default. It also points out a support boundary that matters in enterprise environments: Universe and Multiverse are not supported in the same way as the base repositories unless you have the relevant Ubuntu Pro coverage.

RHEL's package model is RPM-based. In RHEL 10, Red Hat documents the main content repositories as BaseOS, AppStream, CodeReady Linux Builder, Supplementary, and add-on repositories. BaseOS is the core operating system. AppStream contains user-space applications, runtimes, languages, and databases. CodeReady Linux Builder is available with subscriptions, but Red Hat explicitly says packages in that repository are not supported.

That is a trap for Ubuntu admins.

On Ubuntu, enabling Universe is normal. On RHEL, enabling CodeReady Linux Builder may be necessary for build dependencies or certain developer workflows, but it should not be treated as equivalent to a fully supported application source. In regulated or tightly supported environments, that distinction matters.

RHEL also has Application Streams. These provide newer or alternate user-space components while keeping the core platform stable. Red Hat's RHEL 10 docs note that Application Streams can have their own lifecycle, sometimes shorter than the RHEL major release. Before you standardise on a language runtime or database from AppStream, check its lifecycle rather than assuming it lives for the full OS lifecycle.

The other package habit to re-learn is errata.

Ubuntu admins often think in terms of package upgrades and Ubuntu Security Notices. RHEL admins think in terms of package updates, RHSA security advisories, RHBA bug advisories, RHEA enhancements, CVE severity, and whether content has been promoted into the right Satellite content view. In practice, that means your patch reporting and compliance dashboards need to change vocabulary.

Automatic updates and patching

Ubuntu Server commonly uses unattended upgrades for security updates. The Ubuntu package management docs say the default for Ubuntu Server is to automatically apply security updates.

RHEL can automate updates too, but the habit is different. Red Hat documents DNF Automatic for automated software updates, including systemd timer units. In a larger Red Hat environment, however, you will often coordinate patching through Satellite, maintenance windows, content views, and job templates rather than letting every server independently pull whatever is current.

That is not just bureaucracy. It is how Red Hat environments avoid drift.

If you are moving from Ubuntu to RHEL, decide early:

- Are hosts allowed to update directly from Red Hat CDN?
- Are hosts registered to Satellite and pinned to lifecycle environments?
- Do dev, test, and prod see different content views?
- Who promotes errata into production?
- Are security updates automatically applied, automatically staged, or manually approved?
- How do you handle kernel updates and reboots?
- What is the exception path for emergency CVEs?

These questions exist on Ubuntu too, but Satellite makes them more explicit.

Services and startup behaviour

Both Ubuntu and RHEL use systemd. That part is familiar.

The commands are the same:

```
systemctl status ssh
systemctl enable --now nginx
systemctl disable --now nginx
journalctl -u nginx
systemctl list-unit-files
```

The differences are in naming, packaging policy, defaults, and surrounding security controls.

Common service name changes include:

Ubuntu habit	RHEL habit
`apache2`	`httpd`
`ssh`	`sshd`
`cron`	`crond`
`rsyslog`	`rsyslog`
`ufw`	`firewalld`
`apparmor`	`selinux` tooling rather than an equivalent service

The first practical rule is to stop assuming package names and service names match your Ubuntu notes. Sometimes they do. Often they do not.

The second rule is to be explicit in automation. If a service should run, say so:

```
- name: Install Apache on RHEL
 ansible.builtin.dnf:
 name: httpd
 state: present

- name: Enable and start Apache
 ansible.builtin.systemd:
 name: httpd
 enabled: true
 state: started
```

Do not rely on a package install side effect.

On Debian and Ubuntu systems, packages that provide services commonly arrange for those services to start through maintainer scripts and init integration. Debian policy says packages providing system services should arrange for those services to be automatically started and stopped by the init system or service manager. In automation, Ansible's `apt` module even has a `policy_rc_d` option for cases where you want to prevent services from starting during package installation.

On RHEL-like systems, service enablement is more tied to systemd presets and explicit service management. You should still check the specific package, but as an operational pattern, RHEL automation should install, configure, open firewall policy, set SELinux context where needed, and then enable/start the service deliberately.

One more wrinkle: Ubuntu 26.04 release notes say it is the last Ubuntu release that supports System V service script compatibility in systemd. That is a useful reminder that both ecosystems are continuing to push old init assumptions out of the way. If your internal apps still ship ancient init scripts, now is a good time to turn them into proper unit files.

AppArmor vs SELinux

This is the difference that will hurt most if you ignore it.

Ubuntu uses AppArmor by default. Canonical's Ubuntu Server AppArmor documentation describes AppArmor as a Mandatory Access Control system that restricts application capabilities with per-program profiles. It is installed and loaded by default on Ubuntu, profiles live in `/etc/apparmor.d/`, and profiles can run in complain or enforce mode.

RHEL uses SELinux by default. Red Hat's RHEL 10 SELinux documentation describes SELinux as a Mandatory Access Control implementation where policy defines how users and processes interact with files and devices. RHEL installs with SELinux in enforcing mode by default.

The practical difference:

- AppArmor is profile and path oriented.
- SELinux is label, type, role, and policy oriented.

That difference changes troubleshooting.

On Ubuntu, if an application cannot read `/srv/app/config.yaml`, you might inspect AppArmor logs and adjust the application profile to allow that path.

On RHEL, if `httpd` cannot read `/srv/app/index.html`, the Unix permissions may be correct and the process may still be denied because the file has the wrong SELinux context. The fix is not `chmod 777`. The fix is more likely:

```
semanage fcontext -a -t httpd_sys_content_t '/srv/app(/.*)?'
restorecon -Rv /srv/app
```

Or if you move a service to a non-standard port:

```
semanage port -a -t http_port_t -p tcp 8080
```

The mindset shift is this:

On RHEL, "root can read it" does not mean "the service domain can read it".

That is a good thing. It is also the source of many bad first weeks for Ubuntu admins new to RHEL.

Useful commands to learn early:

```
getenforce
sestatus
ls -Z
ps -eZ
ausearch -m AVC,USER_AVC -ts recent
sealert -a /var/log/audit/audit.log
semanage fcontext -l
restorecon -Rv /some/path
setsebool -P httpd_can_network_connect on
```

The worst possible migration pattern is disabling SELinux because it blocked a workload during testing. That usually means the test found a real policy mismatch, mislabeled file, unsupported application layout, or missing boolean. Fix that before production.

AppArmor can be simpler to reason about for application-specific path access. SELinux can be harder at first, but it is deeply integrated into the RHEL security model, service policy, container story, and compliance expectations. In Red Hat environments, SELinux is not an optional hardening extra. Treat it as part of the platform.

Ansible support and automation

Ansible remains one of the easiest bridges between Ubuntu and RHEL.

The same control node can manage both families. The same inventory can group both. The same playbook can branch on facts:

```
- name: Install web server package
 ansible.builtin.package:
 name: "{{ 'apache2' if ansible_os_family == 'Debian' else 'httpd' }}"
 state: present
```

For truly generic work, `ansible.builtin.package`, `ansible.builtin.service`, `ansible.builtin.systemd`, `ansible.builtin.copy`, and `ansible.builtin.template` keep your playbooks portable.

But enterprise automation usually needs OS-family-specific roles. Package names differ, config paths differ, SELinux exists on one side, AppArmor on the other, firewall tooling differs, and repository management is completely different.

The Red Hat-specific automation story is strong. RHEL 10 includes `ansible-core` 2.16, and Red Hat documents RHEL system roles as a supported way to automate repeatable administration tasks. These roles cover areas such as networking, storage, SELinux, firewalld, crypto policies, journald, SSH, timesync, Podman, kernel settings, and systemd.

That is worth using.

If you are migrating from Ubuntu, do not immediately port every internal role line-for-line. First check whether RHEL system roles already cover the boring-but-dangerous base configuration. Boring infrastructure should be boring on purpose.

Satellite also integrates with Ansible. Red Hat Satellite 6.17 documentation includes Ansible integration, and Satellite can run roles during registration and remote execution workflows.

The practical migration pattern I like is:

1. Keep application deployment roles mostly portable.
2. Split OS baseline roles by family.
3. Use RHEL system roles for RHEL platform configuration where possible.
4. Use Satellite for registration, content lifecycle, host groups, remote jobs, and compliance visibility.
5. Use Ansible Automation Platform if you need enterprise workflow, RBAC, controller features, certified content, and support.

The anti-pattern is pretending RHEL is "Ubuntu with dnf". That leads to fragile roles full of conditionals and small exceptions.

Autoinstall vs Kickstart

Ubuntu's automated installer is Subiquity Autoinstall. Red Hat's automated installer is Kickstart.

They solve the same problem: install the OS without clicking through screens.

They do not feel the same.

Ubuntu Autoinstall uses YAML. The Autoinstall reference uses a top-level `autoinstall` key, currently with `version: 1`. It can configure identity, locale, keyboard, networking, proxy, apt mirrors, storage, snaps, packages, SSH, drivers, user-data, early commands, late commands, and interactive sections.

A tiny Ubuntu example looks like this:

```
#cloud-config
autoinstall:
 version: 1
 identity:
 hostname: ubuntu-server
 username: ubuntu
 password: "$6$..."
 ssh:
 install-server: true
 storage:
 layout:
 name: lvm
 apt:
 mirror-selection:
 primary:
 - uri: "http://mirror.internal/ubuntu"
```

Subiquity can consume this through cloud-init style delivery such as NoCloud. The quick start shows the familiar `autoinstall ds=nocloud-net;s=http://.../` boot parameter pattern.

Red Hat Kickstart is older, more established in enterprise bare-metal provisioning, and tightly associated with Anaconda and Satellite. Red Hat's RHEL 10 automatic installation documentation describes Kickstart as a way to deploy RHEL from a predefined configuration. Kickstart files contain commands and sections such as `%packages`, `%pre`, `%post`, repository definitions, network configuration, storage layout, timezone, users, firewall, SELinux, and service enablement.

A tiny RHEL example looks more like this:

```
lang en_US.UTF-8
keyboard us
timezone Australia/Sydney --utc
network --bootproto=dhcp --device=link --activate
rootpw --iscrypted $6$...
selinux --enforcing
firewall --enabled --service=ssh
services --enabled=sshd,chronyd
url --url=http://mirror.internal/rhel/10/BaseOS/x86_64/os/

%packages
@^minimal-environment
chrony
%end

%post
subscription-manager register --activationkey=rhel-prod --org=my-org
%end
```

With Satellite, Kickstart becomes part of a larger provisioning model. Satellite host groups, provisioning templates, activation keys, content views, lifecycle environments, DNS, DHCP, TFTP, HTTP boot, and Capsule placement can all become part of the install path. Red Hat's Satellite content docs note that Kickstart provisioning templates can register hosts with activation keys.

There is also an interesting Ubuntu development here. Canonical documents that Landscape server 25.10 and later can serve Autoinstall files to the Ubuntu installer for Ubuntu 26.04 and later, but only for self-hosted deployments and with OIDC requirements. That brings Landscape closer to the provisioning workflow, but Satellite is still the more mature all-in-one RHEL provisioning control plane.

When migrating, do not try to mechanically convert Autoinstall YAML to Kickstart. Re-design the install pipeline:

- How will systems boot: ISO, PXE, UEFI HTTP boot, virtual media, cloud image, image builder?
- Where does install media come from: Red Hat CDN, Satellite, local mirror, custom ISO?
- How will systems register: subscription-manager, activation key, Satellite global registration?
- Which repositories are available at install time?

- Which content view does the host land in?
- How is storage expressed?
- What happens in `%post`, and what should move into Ansible instead?
- How are secrets passed safely?

Kickstart is powerful, but `%post` can become a junk drawer. Keep the installer responsible for installing and registering the machine. Let Ansible or Satellite handle the rest.

Firewalls and networking

This is not always the first thing people ask about, but it is one of the first things they trip over.

Ubuntu Server documentation describes Netplan as the way network configuration is handled on Ubuntu. Netplan gives you YAML, then renders to NetworkManager or systemd-networkd depending on the system.

RHEL uses NetworkManager directly as the normal server networking layer. The modern command-line habit is `nmcli`, NetworkManager connection profiles, and keyfile-backed configuration. If your Ubuntu automation writes Netplan YAML, that code does not move across directly.

Firewall defaults also differ.

Ubuntu's server docs describe ufw as the default firewall configuration tool for Ubuntu. It is simple and friendly for host-based rules, though many enterprises still manage nftables, cloud security groups, or external firewalls directly.

RHEL's firewall documentation focuses on firewalld and nftables. `firewalld` uses zones, services, runtime configuration, and permanent configuration. That runtime/permanent split is worth learning immediately:

```
firewall-cmd --add-service=http
firewall-cmd --add-service=http --permanent
firewall-cmd --reload
```

If you only make a runtime change, it disappears later. If you only make a permanent change and do not reload, it may not affect the current running firewall. That is an easy mistake to make when you are used to `ufw allow 80`.

Also remember that on RHEL, a service can be installed, configured, and running, while still blocked by firewalld and/or SELinux. Troubleshooting needs to check all layers:

```
systemctl status httpd
ss -ltnp
firewall-cmd --list-all
getenforce
ausearch -m AVC -ts recent
```

Security and compliance posture

Both vendors have serious enterprise security stories, but they package the experience differently.

Ubuntu Pro adds ESM, Livepatch, FIPS packages, compliance profiles, Landscape, and support options. Ubuntu 26.04 also has a noticeable security push around memory-safe system components, TPM-backed full-disk encryption, and improved application permission prompting.

RHEL's security story is built around SELinux, crypto policies, FIPS workflows, OpenSCAP, system roles, Satellite compliance, Insights or Lightspeed recommendations, certified content, Common Criteria and government/compliance expectations, and a very formal support boundary.

For an Ubuntu-to-RHEL migration, the highest-impact differences are:

- **SELinux is expected to stay enforcing.**
- **System-wide crypto policies matter.**
- **FIPS enablement needs to be planned before workloads are deployed.**
- **Compliance content is often tied into Satellite, OpenSCAP, and Red Hat tooling.**
- **Third-party repositories are a bigger governance issue than they may have been in an Ubuntu estate.**
- **Supportability depends heavily on staying inside Red Hat's packaging and configuration boundaries.**

In other words, RHEL rewards discipline. It can feel slower at first, but the point is repeatability, auditability, and vendor-supported state.

Things that will catch Ubuntu admins

Here is the checklist I would pin near the terminal for the first month.

Package and repository traps

- `yum` is mostly muscle memory now. Use `dnf`.
- Do not treat CodeReady Linux Builder like Ubuntu Universe.
- Check AppStream lifecycle before standardising on runtimes.
- Learn `rpm -qf`, `dnf provides`, `dnf history`, and `dnf repoquery`.
- Expect `/etc/yum.repos.d/redhat.repo` to be managed through subscription/Satellite tooling.
- Do not casually add random `.repo` files to production servers.

Service traps

- `apache2` is `httpd`.
- `ssh` is usually `sshd`.
- `cron` is usually `crond`.
- Be explicit with `systemctl enable --now`.

- Check firewalld if a service is listening but unreachable.
- Check SELinux if permissions look correct but the app still cannot access something.

Security traps

- Do not disable SELinux to "fix" an app.
- Learn `ls -Z` as early as you learned `ls -l`.
- File labels matter as much as file modes.
- Ports have SELinux types too.
- Container bind mounts may need SELinux relabel options.

Automation traps

- Split Debian-family and Red Hat-family baseline roles.
- Use `ansible_os_family` carefully.
- Prefer RHEL system roles where they fit.
- Replace `ufw` tasks with `firewalld` tasks.
- Replace Netplan rendering with NetworkManager profile management.
- Keep Kickstart small and move application configuration into Ansible.

Management traps

- Satellite content views are not just folders.
- Activation keys are not just registration tokens.
- Lifecycle environments should match your patch promotion model.
- Capsule placement matters for remote sites and disconnected environments.
- Decide whether Red Hat CDN access is allowed directly or only through Satellite.

A sensible migration approach

I would not start by converting every server.

I would start by building a small RHEL operating model:

1. Pick the supported RHEL major and minor strategy.
2. Define how hosts register: Red Hat CDN, Satellite, activation keys, or cloud marketplace entitlement.
3. Build content views and lifecycle environments before production hosts exist.
4. Create a minimal Kickstart that installs, registers, and hands off to automation.
5. Create RHEL baseline Ansible roles for users, SSH, time sync, logging, firewall, SELinux, monitoring, and backup agents.
6. Port one low-risk application.
7. Keep SELinux enforcing and fix every denial properly.
8. Document package and service name translations.

9. Add patch reporting and errata reporting before the first production maintenance window.
10. Train the team on `dnf`, `rpm`, `subscription-manager`, `firewall-cmd`, `semanage`, `restorecon`, `ausearch`, and Satellite basics.

The goal is not to make RHEL behave like Ubuntu. The goal is to understand the Red Hat way well enough that the platform becomes boring.

That is where RHEL shines. It is not exciting because `dnf install` is more interesting than `apt install`. It is useful because the management, content, security, and support model can become very predictable when you lean into it.

Final thoughts

Ubuntu and Red Hat Enterprise Linux are both excellent enterprise Linux platforms.

Ubuntu often feels faster to get moving with. The package universe is huge, the community is broad, cloud images are everywhere, and the Debian-style workflow is familiar to a lot of infrastructure teams.

RHEL often feels more deliberate. The supported content boundaries are clearer, SELinux is central, Satellite gives you a strong content lifecycle model, and Red Hat's ecosystem is built for organisations that care deeply about vendor support, compliance, and repeatable operations.

If you are switching from Ubuntu to RHEL, the hard part is not the shell. It is the operating model.

Learn Satellite properly. Respect SELinux. Treat repositories as a governed supply chain. Rewrite your installer automation instead of translating it line by line. Use Ansible, but do not pretend OS families are identical. Be explicit about services, firewall policy, content lifecycle, and support boundaries.

Once those pieces click, RHEL starts to feel less like a different distro and more like a different contract with production.

That is the real migration.

Further reading

- Ubuntu 26.04 LTS release notes
- Ubuntu release cycle
- Ubuntu Server package management
- Ubuntu AppArmor documentation
- Ubuntu Autoinstall reference
- Landscape documentation
- Red Hat Enterprise Linux 10 announcement
- Red Hat Enterprise Linux lifecycle
- RHEL 10 DNF documentation
- RHEL 10 SELinux documentation

- RHEL 10 Kickstart documentation
- Red Hat Satellite 6.17 documentation
- RHEL system roles documentation