Red Hat
Partner Connect

Red Hat
Developer

Red Hat Universal Base Images

Everything you need to know about Red Hat's freely redistributable container base images

Contents

Foreword

Containers offer developers an easy-to-use mechanism for building an application with all of its operating system dependencies encapsulated in a light-weight image format. But, these dependencies still have the security, performance, and life-cycle requirements of any Linux® distribution.

Red Hat developed and released Red Hat® Universal Base Images (UBI) to give developers access to the world's leading Enterprise Linux while working with containerized applications. UBI makes it easy for developers to get access to high-quality application dependencies, and also makes it easy for operators to deploy on Red Hat Enterprise Linux and Red Hat OpenShift® when they want high-quality support in production.

Not only does UBI offer the classic value that comes from Red Hat Enterprise Linux as a Linux operating system, it also has its own roadmap of features, including UBI micro to deliver tiny container images, the container health index to verify security, and world-class XML data for scanning the contents of an image to verify Red Hat is patching it regularly. Plus, you don't need to be a Red Hat customer to use it. Keep your eyes out for other new features designed to help you build small, nimble, and secure applications with UBI.

– Scott McCarty: Principal Product Manager, Containers, Red Hat

Part 1

Introduction to Red Hat Universal Base Images

1.1. Introduction

Containers have provided great benefits for developing and operating applications and microservices. Application code, language runtimes, and operating system (OS) components are merged into a single delivery mechanism, allowing an application to run in production with the identical runtime components with which it was developed. Testing and support are streamlined and many of the situations where code doesn't work correctly on a system other than the developer's are eliminated.

To satisfy application runtime dependencies, containers bundle a number of the components from a traditional OS. The foundation for building applications in containers is a *container base image*. A container base image typically includes a number of pre-installed packages and additional packages that can be easily installed with a package manager like YUM or DNF. The base images and their packages are essentially a Linux OS distribution that has been stripped down to the bare minimum.

Selecting, operating, and maintaining operating systems has long been the responsibility of IT. Through experience, IT organizations have learned that high standards for operating systems are necessary to maintain an environment that is stable, reliable, and secure. To reduce complexity and operational drag, IT organizations develop Standard Operating Environments (SOEs) that include an identical OS base image for all of the machines they operate. For updates, security advisories, and help when things go wrong, IT organizations maintain a relationship with an OS vendor that provides support. Reliability, stability, and high-quality, long-term support is why many IT organizations choose to run Red Hat Enterprise Linux.

It's important to consider that the full stack on which a containerized application depends includes the host system's OS as well as the OS components inside the container. With containers, the control over what OS components run inside of the container are shifted from IT to the container developer.

Just as a mature IT organization would exercise control over which OS versions run on their hardware, they need to track what container base images are used to run business-critical applications on their hardware. IT usually still has the responsibility for making sure the application runs reliably.

Containers are an extension of the OS environment. Therefore, containers should be treated as part of a SOE. This is critical when something goes wrong and support is needed that only an OS vendor can provide.

To meet their customer's requirements for support, developers and ISVs can base their container images on Red Hat Enterprise Linux. However, a subscription is required to use Red Hat Enterprise Linux. So for consumers of their software that don't have, or are unwilling to purchase a Red Hat subscription, developers need to build their applications on a different base image. To meet both demands, developers would need to build and support their applications on multiple container base images.

Container base images are essentially Linux distributions that are stripped down to the bare minimum. They include a subset of files from `/bin`, `/etc`, `/lib`, and `/usr` that are necessary for a typical application to run on Linux.

Developers, rather than IT, typically choose the OS components that run inside of containers.

Red Hat solved this challenge by creating Red Hat Universal Base Images (UBI). UBI is a freely distributable subset of Red Hat Enterprise Linux to satisfy the runtime dependencies of container-based applications. Anyone can use UBI without a subscription. No registration with Red Hat is required. UBI retains the most desirable aspects of Red Hat Enterprise Linux, namely long-term updates and support. Updates to UBI and Red Hat Enterprise Linux share the same rigorous attention to security, reliability, compatibility, and performance.

Organizations that want support can get support from Red Hat for the full stack, from Red Hat Enterprise Linux running on the host system up to the UBI components running inside the container. Since UBI can be used without a subscription, the same container image can be used both for organizations that want support and those that don't.

UBI is a good choice for any software project, including free open source projects, that need a source of high-quality container images and packages with long-term updates and stability.

What you will learn from reading this book

This book contains information to help you understand and get started with UBI, including:

- How you can use freely available container images that build on the long term support, commitment to quality, security, reliability, and performance of Red Hat Enterprise Linux.
- Why choosing a base container image is a strategic decision:
  - How container base images are a key part of your standard operating environment.
  - Considerations for choosing base images – why what's in your base image and the number of different base images in your environment matter.
  - How the decisions you make about base images and packages you add to your containers impacts the support options available to those who run your containers.
- About the update and support life cycle for UBI:
  - How often and for how long UBI images and packages will be updated.
  - Support options available from Red Hat for UBI.
- About the Red Hat Ecosystem Catalog, a source for finding container images and other software from Red Hat and its partners. Developers and ISVs can learn how their applications can be listed in the Red Hat Ecosystem Catalog to make it easy for Red Hat customers to find.
- As a developer or ISV, learn how your users can run your UBI-based images on their platform of choice and what options they have for full support by Red Hat.

What you will also learn about working with UBI:

- Which types of UBI images are available and how to choose between them.
- How to find the available UBI images, including language and other runtime images that are ready for you to add your code.
- Where to find additional packages to use with UBI images.
- How you can use additional images and packages available from Red Hat Enterprise Linux and how that affects redistribution.
- How Red Hat partners can use Red Hat Enterprise Linux content in their redistributable images.
- An overview of the OCI-compliant container tools that Red Hat is leading the development of in open communities.

Who should read this book?

Developers and those packaging software in containers should read Parts 1, 3, and 6 to learn how to best use UBI in their projects. Part 5 is recommended to learn about container tools that offer a number of advantages over commonly used tools.

Software partners (Independent Software Vendors, Systems Integrators, etc.) should read Parts 1, 2, 3, and 4 to understand the benefits of using UBI in their products, and the value proposition of working with Red Hat and becoming a Red Hat Partner.

IT architects and managers should read Parts 1, 2, 3, and 5 to understand the role container base images play in an IT landscape and how they relate to standard operating environments. Considerations for choosing a source of base images are covered in Part 2.

Security operations professionals should read Parts 1, 2, and 3 to understand the role of container base images in an IT landscape and their life cycle. Part 4 contains additional security related information on the Red Hat container health index and container vulnerability scanning.

1.2. The motivation behind Red Hat Universal Base Images

What is a container base image?

Linux containers offer a lighter-weight version of the Linux OS that allows an application and its dependencies, like OS and language libraries, to be packaged into an isolated portable environment that can easily be distributed. The lighter-weight aspect is that a single Linux kernel is shared between the host system and any containers running on that host. The isolation is in part because each container has its own virtual filesystem. The files available inside a container are a result of packaging those files into one or more container images.

To run almost anything inside a container there needs to be a number of OS-dependent files inside the containers:

- Dynamically loadable libraries in `/lib` and `/usr/lib`, like the C runtime, math, threading, and cryptography libraries.
- OS configuration files in `/etc`, including network and timezone information.
- Miscellaneous shared OS files in `/usr/share`.
- Writable space for various temporary and transient files in `/tmp` and `/var`.

You might not be aware that your application depends on all of these files. Unless an application is a statically linked binary, it uses a number of dynamically loaded libraries, starting with the C runtime library (often glibc) that provides an interface to the Linux kernel's system calls. Even if your application is written in Java or Python, rather than C, the Java Virtual Machine and Python interpreter that runs your code uses the C runtime library to perform system calls and interact with the system.

Most libraries, including the C runtime library, are built as shared objects to save disk space and memory. Rather than making a copy of them in each executable program, they are dynamically loaded at run time. The library `.so` files need to be available in the file system when the executable program runs.

The files that make up the OS are often referred to as the *userland*. Everything that runs above the kernel is considered the *user space*. If it helps to remember userland, think of `/usr` (pronounced user) where the bulk of the OS files reside. You could say that a Linux distribution is essentially a Linux kernel and a userland packaged in some easily consumable form.

When creating a container, to avoid populating all of the userland files from scratch, a container base image is the most common starting point. Container base images are the files from a Linux distribution that are stripped down to the bare minimum to support running an application. Figure 1 compares container base image components to a Linux distribution.

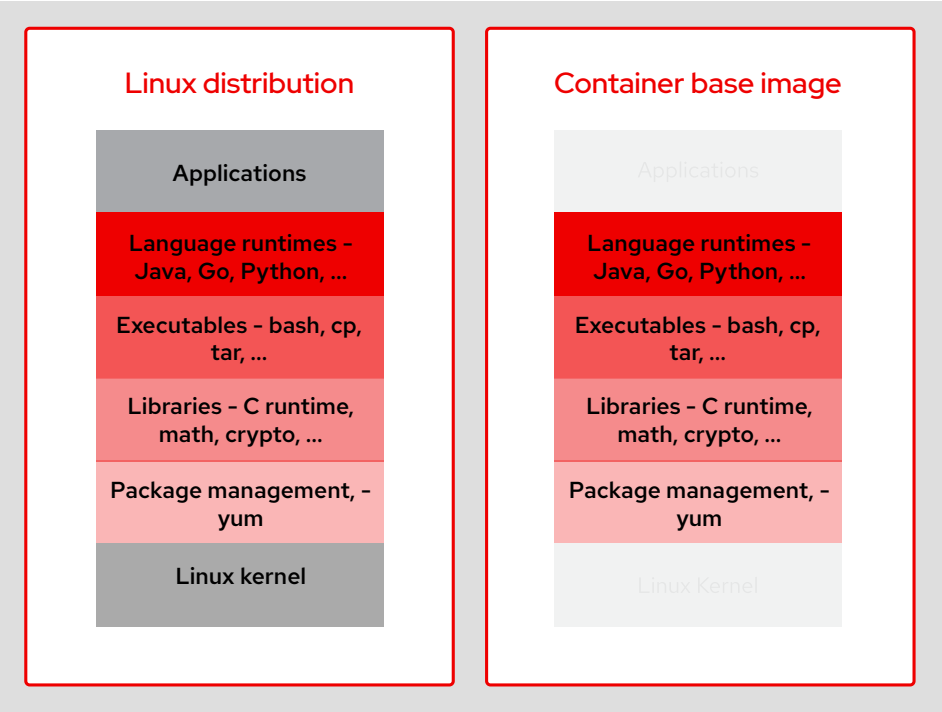


Figure 1. Linux distribution and container base image components

Does it matter what is in the container base image?

Many container images use a base image that is a stripped down version of an existing Linux distribution like Debian, Alpine, Fedora, or CentOS Streams. Most of these base images are maintained by communities. They lack the support that is a critical requirement for many organizations when choosing an OS to run.

For efficiency of moving container images over the network, and to a lesser degree disk space, there is a focus on minimal container size as a primary factor in choosing container base images. While container image size is an easy thing to measure, there are more important factors to consider.

There is a fallacy that the traditional Linux distribution and standard operating environments don't matter anymore with containers. While containers are Linux distributions stripped down to their bare essentials, they are still an operating system and the quality of a container matters just as much as that of the host operating system.

Software that runs in containers still has a life cycle that needs to be maintained. The need for updates to make sure all software is free of vulnerabilities is just as important as with traditional deployments.

It might be tempting to overlook the bits in container images and think it doesn't matter where they come from. What is in container images matters, especially to the organizations that have critical software running in containers. In the early days of container deployments, many organizations didn't have enough experience to realize each container they run becomes another part of their IT landscape that they must maintain. In a sense, the choice of what OS components an organization runs inside container images is delegated to the developer of that containerized application.

The concerns for the contents of container images are similar to choosing an OS or middleware, namely:

Provenance – Do you know the source of the bits in the container image? Are all of the bits actually from the organization you intended it to be?

Authenticity – Can the contents of the container image be verified? Has anything been modified by someone other than the original source?

Security – Can you verify whether the code running in the container is free of any known vulnerabilities? Is the default configuration secure, especially for enterprise use?

Quality and reliability – What testing is performed to make sure the code works correctly and performs well? Updates frequently introduce regressions and can create new vulnerabilities. What testing is done when updates are applied?

Performance – Have there been tests to determine how well the software performs under load on enterprise-grade hardware?

Life cycle – How long will the software in the container be maintained? How long will updates be released to fix bugs and vulnerabilities?

Source code availability – Do you have access to the exact version of the code that was used to produce the container image?

Licensing – Is all of the software actually open source with appropriate licenses that are compatible with your uses? Will you be able to fulfill the requirements for making the source code available for any GPL-licensed software in the base images you use in distributing your software?

As a software developer, many of the above concerns might not seem very important. However, for consumers of your software, especially enterprises with critical business operations depending on software, these concerns are all important. It is because of these concerns that organizations chose to use software with support from Red Hat like Red Hat Enterprise Linux and Red Hat Middleware. The support Red Hat offers on products is possible due to the size of Red Hat's staff to perform engineering, quality assurance, performance testing, security assessments, documentation, releases, and customer support.

Subscription requirement for using Red Hat Enterprise Linux

Red Hat offered certified container images starting with Red Hat Enterprise Linux 7. Given that organizations use Red Hat Enterprise Linux so they can get support, why not just use one of these as a base image? The use of Red Hat Enterprise Linux is governed through subscriptions, and some end users might not have or be willing to obtain Red Hat subscriptions. As a result, developers were faced with a few choices that have drawbacks:

1. Base containers on freely available software like CentOS or CentOS Streams. The drawback is that consumers of their software that want support have no option for support of the OS components inside the container. For community-based software projects, this is the typical choice.

2. Base containers on Red Hat Enterprise Linux. While this option gives those that want it the option of support, they need to require that all customers have or obtain a Red Hat Enterprise Linux subscription.

3. Build two sets of container images, one based on Red Hat Enterprise Linux and the other based on freely available unsupported software. For some developers and ISVs this is the most viable option even though it increases the amount of work to build, test and distribute software.

It's also worth noting that access to Red Hat Enterprise Linux repositories and registries is controlled by authenticating through Red Hat Subscription Management. This can add complexity to automated CI/CD processes that involve Red Hat Enterprise Linux containers, and has led to some organizations using freely available base images, like CentOS, for some automated tasks, even though they had Red Hat Enterprise Linux subscriptions.

1.3. Introducing Red Hat Universal Base Images

In May of 2019, Red Hat announced **Red Hat Universal Base Images (UBI) to provide no-cost, certified, and up-to-date enterprise-grade container base images. UBI provides common application dependencies to form an ideal basis for developing and delivering container-based applications. Built from a subset of Red Hat Enterprise Linux, UBI is freely redistributable. No subscription or any relationship with Red Hat is required to use UBI.**

UBI retains a number of the most important Red Hat Enterprise Linux attributes:

- Support – When run on Red Hat platforms like Red Hat Enterprise Linux or Red Hat OpenShift, UBI is fully supported by Red Hat. This gives organizations that require the assurances of having access to support all of the options that are available from Red Hat.

- 10+ year life cycle – As a subset of Red Hat Enterprise Linux, UBI shares the life cycle of the Red Hat Enterprise Linux version it is based on, with updates and support for up to 10+ years.

- Same release cadence – UBI updates and releases are concurrent with Red Hat Enterprise Linux releases.

UBI is free to download and redistribute. No subscription, login, or even registration is required.

Since container images based on UBI are freely redistributable, UBI is also ideal for open source community projects. The same UBI-based container image can be used for free, open source applications, or enterprise deployments with full support. By using a UBI-based image, as opposed to something like CentOS, Fedora, or Debian, the organization that runs the application, rather than the developer of the software, gets to choose their support options.

What's in UBI?

UBI 8 is a subset of Red Hat Enterprise Linux 8. Likewise, UBI 7 is a subset of Red Hat Enterprise Linux 7. The UBI images are **Open Container Initiative** (OCI)-compliant Linux container images that can be run on any OCI-compliant container runtime like Linux with Docker or **Podman**, Kubernetes with **containerd** or **CRI-O**, or Windows or macOS with Docker Desktop.

UBI base OS container images

The foundational component for UBI is a set of base OS container images. To address different use case requirements, there are three different variations of the UBI base OS container images:

- **UBI Platform image** is designed to address the needs of 80% of typical applications that run on Red Hat Enterprise Linux. In terms of size and pre-installed packages, this is the middle of the road image that is generally the best starting point. For adding, updating, and removing packages it includes the full YUM package management system that you'd find on Red Hat Enterprise Linux. All locales are present to address internationalization and localization.
- **UBI Minimal image** is for applications that contain their own dependencies and want a smaller container image size. Only a minimal set of packages and the English (en) locale are pre-installed.
- **UBI Multi-service image** is designed for running multiple processes inside a single container that are managed by `systemd`. By design, containers generally run a single process. When that process exits, the container exits. The multi-service image runs systemd so that multiple processes, such as a database and a web server, can be run and restarted within a single container.

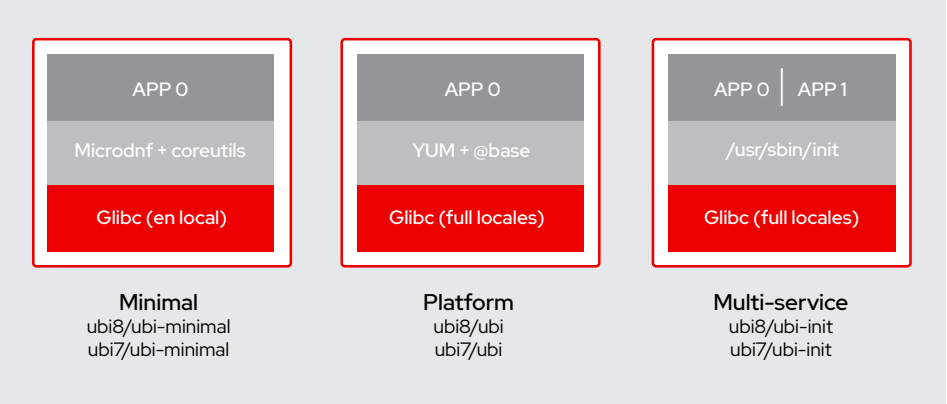


Figure 2. UBI base OS image options

Red Hat software partners that join Red Hat Partner Connect can also redistribute non-UBI and non-kernel Red Hat Enterprise Linux packages through Red Hat Container Certification. See **section 4.7.**

UBI pre-built runtime images

UBI includes pre-built container images with language runtimes including Node.js, OpenJDK, Perl, PHP, Python, and Ruby along with servers like Apache HTTPD and Nginx. These are built on top of the UBI platform OS base image. For UBI 8, these runtime images are based on the application streams from Red Hat Enterprise Linux 8. For UBI 7, these runtime images are based on the Red Hat Software Collection images that are available for Red Hat Enterprise Linux 7.

Some of the container images provided for UBI 8 as of early 2021 are listed in the table below.

Table 1. Language and server runtime images available in UBI 8

UBI 8 Image Name	Purpose
ubi8/dotnet-21 ubi8/dotnet-31 ubi8/dotnet-50	Building and running .NET Core applications
ubi8/dotnet-21-runtime ubi8/dotnet-31-runtime ubi8/dotnet-50-runtime	Running .NET Core applications (runtime only)
ubi8/go-toolset	Building and running Go language applications
ubi8/nodejs-10 ubi8/nodejs-12 ubi8/nodejs-14	Building and running Node.js applications
ub8/nginx-118	Running, proxying, or accelerating web-based applications using Nginx
ubi8/openjdk-8 ubi8/openjdk-11	Building and running Java applications
ubi8/perl-526 ubi8/perl-530	Building and running Perl applications includes Apache HTTPD 2.4 and mod_perl
ubi8/php-72 ubi8/php-73 ubi8/php-74	Building and running web-based PHP applications includes Apache HTTPD 2.4
ubi8/python-27 ubi8/python-36 ubi8/python-38	Building and running Python applications
ubi8/ruby-25 ubi8/ruby-26 ubi8/ruby-27	Building and running web-based Ruby applications
ubi8/s2i-base ubi8/s2i-core	Building source code into images Includes GCC, make, git, and essential libraries

You can find the latest UBI 8 container images in the container image section of the Red Hat Ecosystem Catalog.

Some of the container images provided for UBI 7 as of early 2021 are listed in the table below.

Table 2. Language and server runtime images available in UBI 7

UBI 7 Image Name	Purpose
ubi7/go-toolset	Building and running Go language applications
ubi7/nodejs-10 ubi7/nodejs-12	Building and runningNode.js applications
ubi7/openjdk-8 ubi7/openjdk-11	Building and running Java applications
ubi7/php-73	Building and running web-based PHP applications includes Apache HTTPD 2.4
ubi7/python-27 ubi7/python-38	Building and running Python applications
ubi7/ruby-25 ubi7/ruby-26	Building and running web-based Ruby applications
ubi7/s2i-base ubi7/s2i-core	Building source code into images Includes GCC, make, git, and essential libraries

You can find the latest UBI 7 container images in the container image section of the Red Hat Ecosystem Catalog.

UBI RPM packages and YUM repositories

To add additional software to UBI-based container images, RPM packages are available in UBI YUM repositories. The RPMs available in UBI are a subset of Red Hat Enterprise Linux RPMs. This subset was chosen to satisfy common application dependencies. The RPMs can be installed with the YUM package management system in the UBI platform and multi-service images. For the minimal UBI image, use `microdnf` instead of `yum`.

The UBI RPMs and repositories provide a source of packages, maintained by Red Hat, that you can add to the UBI-based container images that you develop and distribute. The UBI YUM repositories do not require any authentication or subscriptions.

The RPMs available in UBI are the same as their counterparts in Red Hat Enterprise Linux. They have the same life cycle and receive the same updates under the normal Red Hat Enterprise Linux life cycle. The advantage of using UBI RPMs is knowing that if a vulnerability or quality issue is found that is fixed by Red Hat Enterprise Linux, the UBI RPMs also receive the same update.

Additional RPMs compatible with UBI

The UBI RPM repositories contain a much smaller number of RPMs compared to what is available in Red Hat Enterprise Linux. This is understandable as Red Hat Enterprise Linux contains a large amount of packages for interactive and graphical applications that aren't useful in a container-based environment.

As UBI is a subset of Red Hat Enterprise Linux, you have the option of installing any RPMs that are built for the version of Red Hat Enterprise Linux that corresponds to your UBI images. This includes RPMs from third-party repositories like the **Extra Packages for Enterprise Linux (EPEL) project**. For example, you can install packages from the EPEL 8 repository in your ubi8 images.

UBI is compatible with RPMs built for Red Hat Enterprise Linux, including packages from third-party repositories like the EPEL project.

Red Hat Enterprise Linux RPMs are not redistributable

RPMs from Red Hat Enterprise Linux are not-redistributable as a default. **If you add any RPMs from Red Hat Enterprise Linux to your UBI-based container images, you cannot legally distribute them. However, members of the Red Hat Partner Connect Program can distribute container images that contain content from Red Hat Enterprise Linux.** This is discussed in more detail in the *Working with Red Hat* section of this book, **section 4.**

Note: For convenience, when a UBI image is run on a Red Hat Enterprise Linux host that has a Red Hat Subscription, the Red Hat Enterprise Linux RPM repositories are automatically enabled in addition to the UBI repositories. This allows you to easily add any of the packages from Red Hat Enterprise Linux you are entitled to because of your Red Hat subscription. However, care must be taken to avoid adding Red Hat Enterprise Linux packages if you want to distribute your UBI-based images outside of your organization. This is covered in the **Working with UBI** section of this book.

1.4. Future enhancements to UBI

Red Hat made a commitment to UBI and intends to use UBI as a basis for Red Hat products that ship in containers. As a subset of Red Hat Enterprise Linux, UBI continues to evolve along with Red Hat Enterprise Linux itself. Over time, enhancements to UBI are likely to include new runtime images and additional packages. As UBI is based on Red Hat Enterprise Linux, it is anticipated that updates to UBI will coincide with Red Hat Enterprise Linux releases.

To answer customer requests for a base image that is even smaller than the UBI minimal image, a new micro-sized base image is under development at the time this book was written. The UBI micro image is intended for specific use cases where there isn't a need for the majority of OS dependencies like package management within the base image. The goal is for the micro image to be a fraction of the size of the UBI minimal and platform images. While an ultra-small image might be attractive, there are a number of other factors to consider when choosing a base image. In **section 6.5**, there is guidance for choosing between the available UBI base images.

Part 2

Why choosing a source of base images is a strategic decision

2.1. Standard operating environments and containers

Traditional IT organizations have long understood the value of a Standard Operating Environment (SOE). Historically, administrators implemented an SOE as a disk image, kickstart, or virtual machine image for mass deployment within an organization. This reduced operational overhead, fostered automation, increased reliability by reducing variance, and set security controls that increased the overall security posture of an environment.

SOEs often include the base operating system (kernel and user space programs), custom configuration files, standard applications used within an organization, software updates, and service packs. It is far easier to troubleshoot a failed service at 2 a.m. if every server is running the same configuration. Some major advantages of an SOE are reduced cost as well as an increase in agility. The effort to deploy, configure, maintain, support, and manage systems and applications can all be reduced.

Understanding the value of an SOE, a mature IT organization tightly controls the number of different operating systems and OS versions. The ideal number is one, but that isn't usually feasible, so there are efforts to keep the number as small as possible. The IT organization therefore expends considerable effort to make sure that boxes aren't added to the network with ad-hoc OS versions and configurations. The notable exception are applications that are delivered as virtual appliances, either in physical hardware form or virtual machines (VMs). If these virtual appliances are supported by a vendor, it can be reasoned that the vendor is responsible for maintaining the OS and all of the rest of the components on the appliance. Therefore the IT organization is not responsible for those virtual boxes.

Through network security scanning, or worse, during the clean up after a security incident, IT organizations learned that there could be vulnerable software running on their virtual appliances. IT organizations found out some vendor's practices for keeping appliances up-to-date and secure didn't live up to their expectations. Being ultimately responsible for the security of their own systems and networks, IT organizations learned that they need to manage their vendors. IT needs to verify that their vendor's have adequate practices and policies for ensuring the security and reliability of the vendor's appliances/VMs. There is still an OS and other software on appliances and VMs that needs to be maintained.

So what does this have to do with containers? Containers have dramatically improved development, deployment, and maintenance of applications. The ease with which containers can be deployed, and the isolation they offer, simplifies many aspects of IT management. The advent of containers, and to some degree DevOps practices, has led to the notion that traditional IT practices like SOEs and configuration management best practices are no longer relevant.

With containers, it's easy to think you can use whatever technology you want, wherever you want, whenever you want, without having a negative impact on your IT landscape. While it's true that containers have a much smaller footprint and therefore have a much smaller surface area that could be vulnerable, they still have the components of a stripped down

"Even in the world of cloud native and containers, a standard operating environment matters. The set of criteria that should be used to evaluate container base images is quite similar to what we've always used for Linux distributions.

Evaluate things like security, performance, how long the life cycle is (you need a longer life cycle than you think), how large the ecosystem is, and what organization backs the Linux distribution used."

Scott McCarty, Red Hat, Containers need standard operating environments too, Infoworld, March 17, 2021

Linux OS inside. Those components still need to be maintained like traditional OS deployments. However, with containers, the number of versions to track quickly multiplies.

2.2. The impact of not standardizing container base images

The version explosion: how many different versions am I running?

When developers think about building a containerized application, their focus is typically on running a handful of containers. Even if building a big microservices application with dozens or hundreds of containers, the containers likely share a similar heritage, so developers really don't think about the many versions of similar software that could be in play. To really understand the impact of the decisions developers make, you need to consider the consumers of your software and the IT environments for which they are responsible. Given the benefits containers offer, most IT organizations ultimately wind up running hundreds of container images, while large corporations could easily be running thousands of different images.

To understand their perspective, consider what happens if a critical vulnerability or bug is discovered in a heavily used library like the OpenSSL cryptography library or the C library (glibc). The first task is identifying all the places the vulnerable versions are running. To do that they need to know what version is running on every system, which includes every container.

Without an SOE, or at least policies to govern what base images are used, an organization could wind up in a situation depicted in Figure 3, where base images covering 14 different operating systems are used.

Figure 3. Multiple versions of software due lack of an SOE for base images

In the environment depicted above, there are 11 different versions of OpenSSL and 8 different versions of the glibc C library. The situation could even be worse than that, given that there might be common source versions numbers across OS versions, but the actual packages are different due to different patch levels, or different configuration flags used at compilation. Another complication is that different distributions don't use the same conventions for naming and versioning packages. One distribution might package all files for a piece of software into a single larger package, where others break it into a number of smaller packages.

The above scenario might seem contrived, however consider that a typical application landscape includes language runtimes, database, web, and cache servers. So there might be base images for Java, Python, PHP, MySQL, PostgreSQL, Reds, Apache HTTPD, Apache Tomcat, and Nginx to satisfy application dependencies.

The availability of pre-built container images for software components in public registries gives developers a wealth of choices. The developer selecting the image for a database might focus on choosing the latest version, but not investigate what OS forms the base of those images. Or they might choose an image based simply on the smallest image size, even though the base Linux distribution for that image might not be something an enterprise would choose to run in their environment.

Due to the potential for software versions to multiply in an environment, having an SOE for containers is just as important as an SOE for operating systems. Figure 4 shows how the number of versions of system software in an environment can be reduced by making a container base image part of your SOE.

Figure 4. Standardizing on a container base image to control software versions

For more information see the *Infoworld* article, **Containers need standard operating environments too**, by Red Hat's Scott McCarty.

2.3. Responsibility for maintaining the software stack inside of container images

Changing the way applications are distributed shifts some of the roles and responsibilities for maintaining the software that supports an application. To understand the responsibilities with container-based application delivery, it is helpful to contrast it with traditional and the appliance/VM-based application delivery.

Traditional application delivery

For traditional application delivery, the application is delivered as a package (`.rpm`, `.deb`, `.tar`, etc.) that is compatible with the systems used by the consumers of the application. The consumer installs the application on their system following the developer's instructions. Any necessary additional software like libraries, runtimes, or servers necessary to run the application might be specified in the instructions. Package managers like YUM might pull in additional packages to satisfy dependencies.

The developer is responsible for maintaining the application and not much else. It is up to the consumer of the application to maintain the OS and other supporting software. If an OS package is updated due to a disclosure in the **Common Vulnerabilities and Exposures (CVE) system**, it's the consumer's responsibility to track and install it.

In this model, while the developer has less responsibility, they have the additional work of building, testing, and supporting their software on the multitude of configurations they chose to support. These configurations are likely largely dictated by their customer's demands.

Appliance/VM-based application delivery

In this model, the developer of an application delivers their application to the consumer in the form of a physical or virtual appliance, like a VM image. The developer has the additional work to build the VM image or appliance that runs the application, but only needs to build, test, and support the application on the configuration of the custom built appliance/VM.

The application consumer only needs to run the appliance, or in the VM case run it on a supported hypervisor with adequate resources. The consumer looks to the developer for all updates because the appliance/VM is essentially a black box.

The developer is responsible for maintaining essentially everything, including the host OS, libraries, and any required/supporting software like web and database servers, that run on the appliance/VM. The developer must make sure OS patches get applied, or periodically ship and install updated VM images, which often turns out to be cumbersome.

Container-based application delivery

Similar to the appliance/VM model, when using container-based delivery the developer of the application selects the complete runtime environment for the application. The developer controls everything from the OS userland up to the application code itself. Containers make this process much simpler and easier than delivering an appliance or VM image.

The consumer of the application needs to provide and maintain a suitable container host and container runtime environment. The consumer is responsible for updating things like the Linux kernel, the container engine, and its dependencies.

To the application consumer, the contents of the container(s) are basically a black box. The developer who provided the application in container(s) is responsible for everything in the application container(s) including the OS base image, any intermediate images, and any packages installed in the container. This also applies to any images that are pulled and customized. So for example if the developer takes a database image, layers their application-specific customization on top of it, and then produces a new image that users of the application consume, the developer is now responsible for that image.

It is important to know that container images once built are immutable. The only way a container image gets updated is to rebuild it. Any other containers that are built from that image will also need to be rebuilt to pick up the change.

There is sometimes confusion over container tags vs. the cryptographic hashes that identify images. If you used the `:latest` tag to specify the image you are building your image from, that doesn't mean if a new `:latest` image becomes available that the layer based on it will get updated the next time an image using it is pulled for running.

When container images are built, any images that are used for source layers are identified by their immutable cryptographic hash, not the tag that was used. If you use a tool that allows you to inspect a container image, you can see that the source layers are specified by their cryptographic hash, not by tags. You can also see the same thing when pushing and pulling image layers.

Rebuilding the container image is the only way to pick up the updated base image layer. So if there is a CVE for one of the components inside of a container, the developer of the container is responsible for making an updated image available to the consumers of their application. The net result is that the responsibility for maintaining the components in a container-based application is very similar to appliance/VM-based application delivery. Fortunately, with containers it is much easier to build a new image and make it available.

It is probably worth noting that a consumer that is unable to obtain updated images from a developer has a number of additional options with containers, including rebuilding the image if all of the sources are available. Alternatively, a new *patched* image could be produced by running a build with the original image as the base and adding the necessary fixes. After which the patched image needs to be run instead of the original.

2.4. Considerations for choosing a source of base images

There are a number of things to consider when choosing a source of base images to use as part of an SOE for containers. A primary concern is availability of updates. However, this is more complicated than just using the latest code. Updating code often introduces regressions and new CVEs. Testing is needed to make sure that updates don't cause more problems than they solve.

Over the last decade or more, security consciousness has raised considerably. Performance engineering has similar concerns but hasn't historically received the same level of attention as security. Language runtimes, web servers, and libraries like openssl all affect performance in different ways when applications are run under load. Just pulling in the latest upstream code does not guarantee good performance. Similar to security and functionality, performance regressions are often introduced without notice.

Updated software also needs to be tested and tuned by an enterprise-grade performance engineering team.

Some of the other things to consider when choosing a source of base images:

Architecture

- Are the images available in all the processor architectures that might be needed today or in the future?
- What C library, core utilities, and compilers are used to produce the images?

Life cycle

- What are the update policies and commitments for providing updated images?
- How long will updates be available for the versions of software used in the original releases?

Security, reliability, and performance

- What are the processes and commitments for finding, fixing, and communicating information about security vulnerabilities and bugs?
- Are there reasonable processes for consumers to report and track security, reliability, or performance issues?
- Is there a proactive security response team that is testing for issues?
- Are there reasonable policies and practices for responsible disclosure of sensitive security vulnerabilities?
- Is there a proactive performance team that is testing performance on enterprise-grade hardware and tuning libraries and other components?
- What testing is performed to make sure that updates for security, performance, or bugs retain the expected functionality?

Provenance

- Can the contents of images be verified to be from a trusted source and free of any modifications since built?
- Is all of the software actually open source with appropriate licenses?
- Will you be able to fulfill the requirements for making the source code available for any GPL-licensed software in the base images you use in distributing your software?

Organization

- Is there a product team capturing customer requirements and driving improvements?
- Is the organization responsible for the images viable enough to be in existence for the long term?

2.5. The benefits of UBI as an SOE

Red Hat's goal in creating UBI is to produce a base image for all the needs of developers, ISVs, and community projects. UBI is a freely redistributable subset of Red Hat Enterprise Linux for building container-based software.

The bits provided in UBI are identical to those provided in Red Hat Enterprise Linux. They only differ in the terms and conditions for using them. This is the same software used extensively for some of the world's most critical workloads, such as high performance computing (HPC), financial services, and AI/ML. It's used in highly secure environments like governments and banking, I/O intensive applications like transaction processing, and performance critical applications with requirements for specialized hardware and/or low latencies.

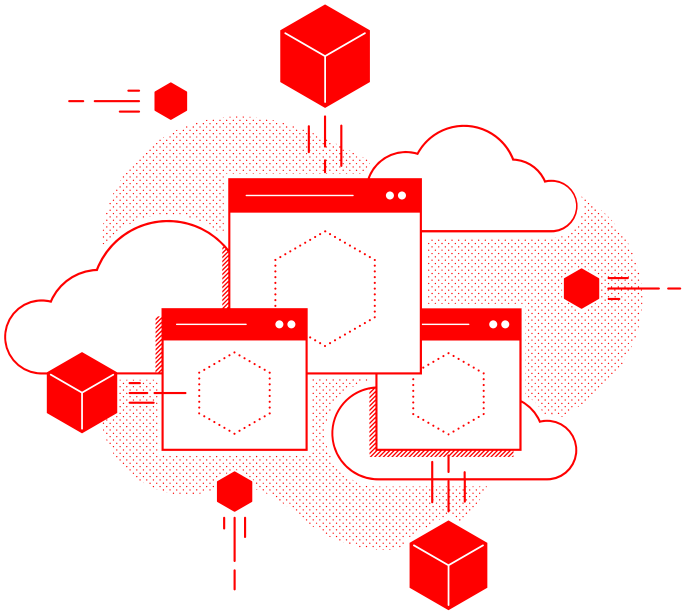
By choosing UBI, developers and ISVs can use the same container images for software they are making freely available or selling to enterprises. The enterprise customers of these developers can choose support options from Red Hat that meet their needs.

Developers, operations, and security teams in many IT organizations have extensive experience with Red Hat Enterprise Linux. UBI gives them familiar base images, packages, and package management tools that they can easily support without retraining.

UBI shares the same 10+ year life cycle as the version of Red Hat Enterprise Linux on which it is based. UBI components are updated when the corresponding Red Hat Enterprise Linux components are updated.

There are no subscription management or registration requirements for using UBI. This, combined with the long support life cycle, makes UBI an excellent choice for free software projects and automated build systems like CI/CD pipelines.

Finally, Red Hat is committed to using UBI as the base image for Red Hat products. You have the assurance that UBI is critical to the success of Red Hat's products, giving you the confidence to use it as a basis for your own software as well.



Part 3

UBI support and licensing

3.1. UBI life cycle and updates

As a subset of Red Hat Enterprise Linux, UBI follows the same live cycle of up to 10+ years. Every time there is a new release of Red Hat Enterprise Linux, new Red Hat Universal Base Images and supporting packages are released as a new version number.

UBI 8

UBI 8 is based on Red Hat Enterprise Linux 8, which was released in May 2019. The 10-year support cycle runs until May 31st, 2029. Red Hat Enterprise Linux 8, and therefore UBI 8, have a schedule of four release trains per year.

The UBI 8 language runtime images are based on the Application Streams packaging in UBI 8. Application streams have a different life cycle than the base OS and are updated more frequently. When Red Hat Enterprise Linux 8 Application Streams are updated, UBI packages and images based on them are updated.

UBI 7

UBI 7 is based on Red Hat Enterprise Linux 7, which was released in June 2014. The 10-year support cycle runs until June 30th, 2024. Note that as Red Hat Enterprise Linux 7 has entered the second half of its support period, it is now in the maintenance support stage. For those who need support beyond the 10-year period, extended life cycle support subscriptions are available from Red Hat. The cadence for Red Hat Enterprise Linux 7 and UBI 7 maintenance releases is approximately every six months.

The UBI 7 language runtime images are based on Red Hat Software Collection container images. Red Hat Software Collections have a different life cycle than the base OS and are updated more frequently. When the Red Hat Software Collections are updated, UBI packages and images based on them are updated.

More information can be found in the following documents on the Red Hat Customer Portal:

- Red Hat Enterprise Linux life cycle.
- Red Hat Universal Base Image – content availability.
- Red Hat container image updates.

NOTE: Given the availability of UBI 8, and that Red Hat Enterprise Linux 7 and UBI 7 have entered the maintenance support phase of their life cycle, it is strongly recommended that you use UBI 8.

The life cycle for updates and support of components in UBI are the same as their counterparts in Red Hat Enterprise Linux. UBI 8 has the same life cycle as Red Hat Enterprise Linux 8.

3.2. UBI container image and RPM update policy

Updated UBI container images are available in the Red Hat Ecosystem Catalog and Red Hat Container registries when a new release of the corresponding Red Hat Enterprise Linux version occurs.

Rebuilt images are available as deemed necessary to fix a critical vulnerability (CVE) or other problem in one of the components that is included in the image. The updated images are available as the latest images in the Red Hat Ecosystem Catalog and Red Hat Container registries.

Updates are made as deemed necessary to RPM packages included in UBI. UBI RPMs are updated when the corresponding Red Hat Enterprise Linux RPM is updated. The updated RPMs are available in the UBI YUM repositories.

However, new container images are not built every time an updated RPM is made available, as this could require thousands of rebuilds. As mentioned above, container images are rebuilt when deemed necessary to address a critical problem. Red Hat rebuilds UBI base images every six weeks. So an updated RPM that doesn't trigger a rebuild is included in a UBI base image within six weeks. For more information on UBI update policies, see **Red Hat Universal Base Image – content availability** on the Red Hat Customer Portal.

UBI users also have an option to make sure the container images they build using UBI have all updates available at the time the image is built. To do this, a `yum update` step is inserted into the build process.

UBI users also have the option of rebuilding UBI images themselves. The UBI YUM repositories contain all of the RPMs used in UBI. Build information is available on the *Dockerfile* tab of the source image's page on the Red Hat Ecosystem Catalog.

3.3. The limits of container compatibility and supportability

To correctly set expectations about support and the limitations that exist in ensuring compatibility, there is something fundamental that needs to be understood about Linux containers. Linux containers are heralded as completely portable across environments, making compatibility problems a thing of the past. However this is not strictly true. While this might sound like heresy, it all has to do with the Linux kernel.

Recall that with containers, the OS components, runtime dependencies, and application are packaged as a single deliverable unit. What's missing from that package is the Linux kernel. To make containers lightweight, the host system Linux kernel is shared across all containers running on that system. When a traditionally deployed application is tested, the full OS stack is exercised along with it, everything from the kernel up to the application. However, with containers there is nothing that specifies which version(s) of the Linux kernel the host system must be running. So there is a false expectation that every container can run every Linux kernel version. Clearly that behavior can't be guaranteed. There are too many possibilities to test.

At the time this book was written the most recent mainline Linux kernels are from the fifth major version (5.x). Red Hat Enterprise Linux 8 uses a kernel based on 4.18.x, while Red Hat Enterprise Linux 7 uses a kernel based on 3.10.x. New Linux kernel releases are occurring every two to three months. To ensure compatibility, the version of the Linux kernel in a Red Hat Enterprise Linux major release does not change. During the 10+ year life cycle of a release, Red Hat integrates bug fixes, backports important changes, and adds support for new hardware, but maintains strict binary compatibility.

So can someone expect that a container image built and tested on Fedora 33, which uses kernel version 5.8.x, should run without any issues on Red Hat Enterprise Linux 7, which uses kernel version 3.10.x? The answer is, it depends on how the application was written.

Fortunately, most, if not the vast majority of applications do work, because they stick to a path that is well traveled. There is an application binary interface (ABI) specification that governs the Linux kernel's system call interface that is used by applications to get services from the Linux kernel. ABI compatibility can be tested and ensured compatible across releases. System calls are used for all the most common operations like creating, reading, and writing files, starting and stopping processes, etc.

Problems arise because there are a number of things that aren't covered by system calls and fall outside of the ABI. For example, the notion of which timezone the system is set to is often determined by a file in `/etc` or an environment variable. There isn't a system call for it, and there isn't even agreement between Linux distributions. There are no system calls for adding or deleting users. The kernel only deals in numeric user IDs (UID). Mapping usernames to UIDs is done either by files in `/etc`, or some user level processes that implement a directory service.

The virtual filesystems `/proc` and `/sys` can also be a source of compatibility problems across kernel versions. These provide an interface to kernel information and settings that is much less formally defined than system calls. The majority of applications don't need to interact with these files. However, some software might attempt to control configuration and performance using system information, such as how much RAM and how many processor cores are available, to know how much memory can be allocated and how many threads to use. The memory and CPU information can be found in `/proc`, as there are no system calls for that.

Compatibility problems between a container's userspace components (like glibc) and the host system's kernel version do arise. The good news is that they are somewhat rare, the bad news is they can be complex to diagnose. Red Hat support has helped customers resolve a number of these issues.

In the **Red Hat container image and host guide: application portability** you can find a list of problems known to occur when mixing container images and container hosts. These include:

- Running binaries that require specific input/output control (ioctl) calls, or specific layouts of /proc and /sys, which are incompatible with, and determined by, the version of the underlying container host kernel. All of these interfaces can change through versions of the kernel, user space tools, or libraries.

- Container host and container image version mismatches. The larger the version mismatch between a user space and the kernel, the more likely there will be incompatibilities.
    - Newer binaries or libraries, older kernel. Running binaries or libraries that depend on newer kernel features not available on the container host could cause problems. This can happen when running a newer kernel in development and an older kernel in production. With a mismatch like this, glibc checks for a minimum kernel version. **If the minimum version is not satisfied, the program will exit**.
    - Older binaries or libraries, newer kernel. Running binaries or libraries on a container host with a newer kernel than used during testing and development could result in behavioral differences. This is particularly true when glibc enables new runtime-detected features.
- Differences in hardware, kernel, and libraries. For example, glibc is optimized for specific versions of hardware – some accelerated routines are selected based on hardware availability. This is a combination of kernel detection and glibc detection. Inconsistencies between nodes can lead to behavioral changes in your application. For example, routines might change from hardware accelerated to software only or vice versa. This could unexpectedly slow your application down or speed it up, depending on what type of system it is run on. If your application consists of multiple containers serving a single service or **application programming interface (API)**, all requests might not perform the same.
- Running container images that expect specific kernel capabilities (CAP_PTRACE, etc.) on **a container host, with a container engine, configured to deny the behavior**.
- Running binaries in the container that are compiled with the expectation of Security-Enhanced Linux (SELinux) support on hosts that do not support SELinux.

Compatibility across different Red Hat versions

It takes a lot of engineering, security analysis, and resources to provide quality support for container images. It requires testing not just of the base images, but also their behavior on a given container host. Red Hat has focused heavily on engineering and support to ensure that UBI 7 images run as well as reasonably possible on Red Hat Enterprise Linux 8, and conversely to ensure the same for UBI 8 images on Red Hat Enterprise Linux 7 hosts. The ability to mix these versions helps organizations reduce their upgrade challenges. However, it is important to understand that there are some limitations for compatibility and support. Therefore, for the best compatibility, and highest level of support, run UBI 8 on Red Hat Enterprise Linux 8, and UBI 7 on Red Hat Enterprise Linux 7.

Red Hat's support policies and compatibility matrices are linked below in the *support* section. For a deeper discussion of compatibility, see these blogs by Red Hat's Scott McCarty:

- **The limits of compatibility and supportability with containers**.
- **Containers: understanding the difference between portability, compatibility, and supportability**.

3.4. UBI support information

The Red Hat Universal Base Image can be deployed in two ways. Each comes with different support expectations.

1. On a Red Hat supported container platform, applications built with UBI are supported as a full Red Hat Enterprise Linux stack when run with all of the following conditions:

    a. On a Red Hat supported container platform (Red Hat OpenShift or Red Hat Enterprise Linux).

    b. With a Red Hat shipped and supported **container engine** (Red Hat provided CRI-O, Podman, etc.).

    c. With a Red Hat shipped and supported **container runtime** (Red Hat provided runc, etc.).

2. On any other container platform, or with any other container engine or runtime – including upstream Kubernetes, cloud provider-based Kubernetes services, other Linux distributions, any other non Red Hat Linux distribution, or a non Red Hat provided container engine or runtime – users receive updates, but support is not provided by Red Hat. There is no way to purchase support on any platform other than a Red Hat container platform (Red Hat OpenShift, Red Hat Enterprise Linux). Red Hat does not perform any testing or validation of UBI on any non Red Hat software stack. Any issues should be filed with the respective upstream communities or products. If the issue can be reproduced on a Red Hat supported platform, then support will be available as per number 1 above.

Figure 5 below shows the runtime configurations supported by Red Hat.

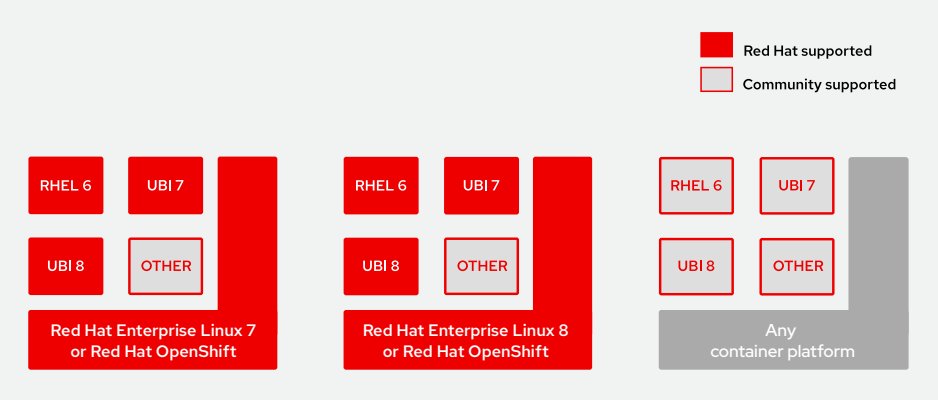


Figure 5. Runtime configurations supported by Red Hat vs. community support

See **Red Hat Container Support Policy** on the Red Hat Customer Portal for the complete container support policy.

Red Hat support for mixed container image and container host OS versions

There are known limitations with mixing and matching versions of the OS in the container image and the container host. Because of this, Red Hat breaks the support conditions into three tiers with tier 1 as the highest compatibility and tier 3 the lowest. The support matrix is shown in table 4.

Table 4. Red Hat support for mixed container image and container host versions

Container image	Red Hat Enterprise Linux 8 host	Red Hat Enterprise Linux 7 host
Red Hat Enterprise Linux 8 or UBI 8 userspace	Tier 1: fully compatible	Tier 3: commercially reasonable support
Red Hat Enterprise Linux 7 or UBI 7 userspace	Tier 2: workload specific	Tier 1: fully compatible
Red Hat Enterprise Linux 6 userspace	Tier 2: workload specific	Tier 2: workload specific

Tier 1 – fully compatible

For tier 1 fully compatible configurations, the container image and the container host are fully supported and tested together. This configuration makes sure that all low-level kernel subsystems use matching userspace and kernel driver components. Running privileged containers is supported but certain workloads might require that the minor version of the container host match the minor version of the container image.

Tier 2 – workload specific (unprivileged)

For tier 2 unprivileged configurations, the combinations of container images and container hosts have support limitations. Any potential problems exposed are handled per the Red Hat Enterprise Linux life cycle and support policy as described below. Users can expect to run older container images on newer container hosts in a supportable way if they meet all of the following conditions:

1. The container image is still within the supported Red Hat Enterprise Linux life cycle. For example, when Red Hat Enterprise Linux 6 reaches the Extended Life Phase, proper ELS subscriptions are required to support Red Hat Enterprise Linux 6 container images regardless of the underlying container host version used. Also, if a Red Hat Enterprise Linux 6 bug is exposed when running on a Red Hat Enterprise Linux 7 host, the maintenance phase of Red Hat Enterprise Linux 6 is a determining factor of whether or not the issue will be resolved. Note the same considerations apply to Red Hat Enterprise Linux 7 after June 30th, 2024 when the maintenance support phase of Red Hat Enterprise Linux 7 expires.

2. The application is running as an unprivileged container. Running privileged containers, e.g., running with `--privileged`, reduces the isolation and exposes a tighter connection between container and host interfaces and is not supported in these configurations.

3. The application or its dependencies does not interact directly with kernel-version-specific data structures (ioctl, /proc, /sys, routing, iptables, nftables, eBPF etc.) or kernel-version-specific modules (KVM, OVS, SystemTap, etc.). Support for ioctls

and access to /proc is limited to the most common use cases needed by unprivileged uses. All other uses require tier 1 compatibility.

Tier 3 – commercially reasonable support

Tier 3 commercially reasonable support configurations require the end user to test that it's appropriate for their workload. While many commonly deployed workloads will function, they might still fail either at startup or later at runtime. Over time, userspace components from newer container images require newer kernel features to work properly and older kernels will not be able to meet these requirements. This configuration has the most risk of running into software version misalignment and serious failures, including but not limited to, unscheduled downtime or data loss. Users must be prepared to migrate to tier 1 or tier 2 if a solution to their issue is not possible. Testing of application compatibility is not performed by Red Hat, and support is limited to commercially reasonable effort to assist with the analysis, but not to provide bug fixes to resolve incompatibility. Users can expect commercially reasonable support for running newer container images on older container hosts if they meet all of the following conditions:

1. All tier 2 support conditions apply.
2. Red Hat Enterprise Linux life cycle restrictions apply.
3. The user can show support that the tier 3 issue reproduces with a tier 1 configuration and has the same underlying cause.
4. The user is responsible for validating compatibility of the container image (application) and container host.
5. The user shows (through strace or systemtap traces) that the container image (application) requires only syscalls, and that all syscall features (includes flags, options, and paths) are present in the underlying container host kernel.

See the **Red Hat container compatibility matrix** on the Red Hat Customer Portal for the full and up-to-date policy.

Recommendations based on the container compatibility matrix

Based on the limitations for compatibility and the container compatibility matrix, a few recommendations are evident:

1. For the highest level of support and compatibility, use the same OS version for both the container host and container image wherever possible.
2. It is preferable to use a container host OS that is either the same OS version or newer OS version than the container images that you want to run. As shown in the compatibility matrix, the lowest tier of support is for running newer container images like UBI 8 on a Red Hat Enterprise Linux 7 host. Put another way, having a container host with a recent, up-to-date OS is the next best case after matching host and container image versions. A more recent OS is likely to have all of the functionality and patches necessary to run older software.
3. Building containers in a manner that requires them to have privileges, or detailed OS level interaction, should be avoided where possible. It can impact the ability to support them. Additionally, by default you might not have permission to run privileged containers on cloud platforms like Red Hat OpenShift.

Red Hat support documents

For the most up-to-date UBI support information, see these documents on the Red Hat Customer Portal:

- Red Hat container support policy – This document describes how Red Hat provides support for different combinations of container technologies.
- Red Hat container compatibility matrix – Details the levels of support available across differing OS versions between container hosts and container images.
- Red Hat container image and host guide: application portability – Provides guidance for selecting container host OS and base images.
- Red Hat Universal Base Image – content availability – Details how updates to UBI are provided.

3.5. UBI licensing and redistribution

The Red Hat Universal Base Images End-user License Agreement (Red Hat UBI EULA) is a **Red Hat license specifically produced to make UBI components freely redistributable**. This means anyone can download, use and redistribute them – even if they don't have a Red Hat subscription or are a Red Hat customer. Red Hat content governed by the EULA must be tagged with this license and/or the "ubi" label for that content to fall under the UBI terms and conditions.

All of this content is usable and freely redistributable under the terms of the UBI EULA. Other Red Hat Enterprise Linux packages not included above are not part of the Red Hat UBI EULA.

Redistribution refers to how and where software is distributed and deployed. The Red Hat UBI EULA allows anyone to freely distribute and deploy UBI-based images and packages on Red Hat and non Red Hat platforms. Through the Red Hat Partner Connect agreement, partners can also do the same with non-UBI Red Hat Enterprise Linux images and packages.

If you add RPMs from non-UBI Red Hat Enterprise Linux repositories, the resulting image is only redistributable if you're a Red Hat Partner Connect member and that image has been certified through Red Hat Container Certification. Otherwise, if any Red Hat content that isn't covered under the Red Hat UBI EULA is added to an image based on UBI, it can no longer be legally redistributed.

For more information, see:

- UBI licensing frequently asked questions.
- UBI redistribution frequently asked questions.
- Red Hat partner container certification guide – redistribution of packages.

Members of the Red Hat Partner Connect program can distribute Red Hat Enterprise Linux packages and images in their UBI-based applications that have been certified by Red Hat. See section 4.7.

Part 4

Working with Red Hat

4.1. Getting support from Red Hat

Red Hat customers with a subscription and Red Hat partners can file support tickets through the Red Hat Customer Portal. As mentioned above, to qualify for support, UBI images must be running on a Red Hat supported container platform (Red Hat Enterprise Linux or Red Hat OpenShift).

See **How to open a support case** on the Red Hat Customer Portal. Red Hat Support staff are available to guide customers and partners in creating support tickets.

4.2. Getting a no-cost subscription for access to Red Hat resources

There are many benefits to Red Hat subscriptions. If you don't have a Red Hat subscription, you can get one at no-cost by joining the **Red Hat Developer Program** or **Red Hat Partner Connect**. While a no-cost subscription does not give you access to Red Hat support by default, you do gain access to many of the same resources, including software updates, access to the **Red Hat Customer Portal**, and downloads of developer-oriented software.

You can download and run Red Hat Enterprise Linux on a physical or virtual machine, and there are developer-focused instructions on Red Hat Developer covering how to install a Red Hat Enterprise Linux VM for development.

- **Red Hat Enterprise Linux 8 quick install using VirtualBox**.
- **Red Hat Enterprise Linux 8 quick install using Hyper-V**.
- **Red Hat Enterprise Linux 8 quick install on bare metal**.

For those working with UBI, but using another development platform like Window or macOS, this provides an easy way to test UBI containers on a Red Hat supported container platform.

The Red Hat login you receive by joining one of these programs lets you log into the Red Hat Customer Portal, where you can find the Red Hat knowledge base, articles that aren't publicly available, and community support forums.

Whether you are part of an organization using Red Hat products and services, a developer, or a software vendor, there are many ways Red Hat can help:

- Get support for products you are using.
- Discover certified products and solutions from Red Hat and its partners.
- Improve your reach to Red Hat customers by certifying your applications and listing them in the Red Hat Ecosystem Catalog or Red Hat Marketplace.

4.3. Requesting UBI enhancements

Red Hat partners and customers can request new features, including requests for the availability of additional packages in UBI, by filing a support ticket through the Red Hat Customer Portal. See Getting support from Red Hat above.

Non Red Hat customers do not receive support but can file requests in Red Hat Bugzilla, bugzilla.redhat.com. When creating a ticket, select *Red Hat Enterprise Linux 8* or *Red Hat Enterprise Linux 7* as the product. Under component, select *ubi8* or *ubi7* as appropriate. This is illustrated in the screenshot in Figure 6 below:

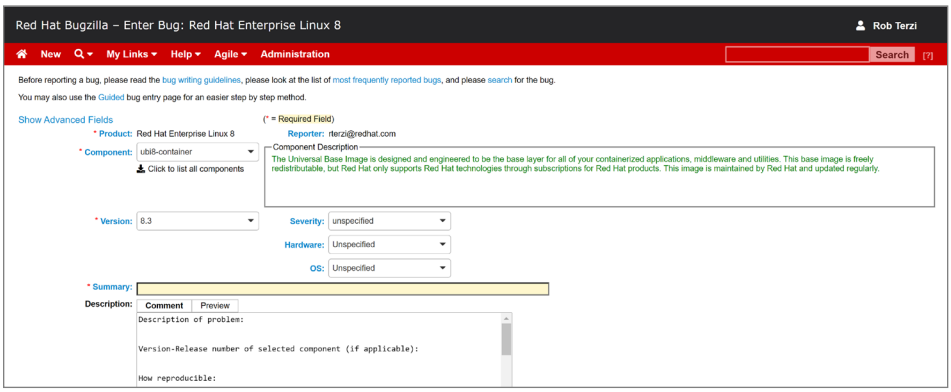


Figure 6. Using Red Hat Bugzilla to request UBI enhancements

4.4. The Red Hat Ecosystem Catalog

The Red Hat Ecosystem Catalog is a place to find certified, enterprise-grade products and services from Red Hat and Red Hat's large and robust ecosystem of enterprise hardware, software, and cloud and service providers. This single entry point makes it easy to find platforms, products, and services to use in addressing your business technology needs.

In the container images section of the Red Hat Ecosystem Catalog, you can find Red Hat software available as container images as well as certified container images from Red Hat partners.

The Red Hat Ecosystem Catalog is the place to find UBI images. It's also where Red Hat users find certified enterprise hardware, software, and cloud and service providers that work with Red Hat products.

Red Hat Ecosystem Catalog for Red Hat published UBI images

The Red Hat Ecosystem Catalog is where you can find container images, documentation, and other resources for Red Hat Universal Base Image 8, and Red Hat Universal Base Image 7.

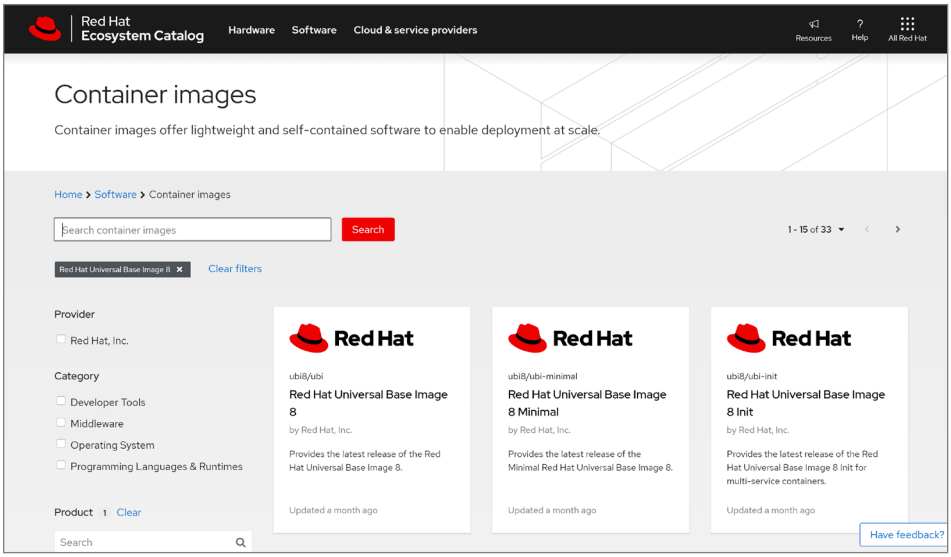


Figure 7. Red Hat Ecosystem Catalog

Note: If you prefer to use the command line instead of the online catalog, you can search the Red Hat container registry for UBI images. For example:

```
$ docker search registry.access.redhat.com/ubi | sort
```

Or using Podman:

```
$ podman search registry.access.redhat.com/ubi | sort
```

More information on finding and using UBI images can be found in section 6.1 Where to find UBI container images.

Red Hat Ecosystem Catalog for Red Hat certified partner products

The Red Hat Ecosystem Catalog is also where Red Hat publishes **certified partner hardware, software, and cloud and service providers** to make them easy for customers to find.

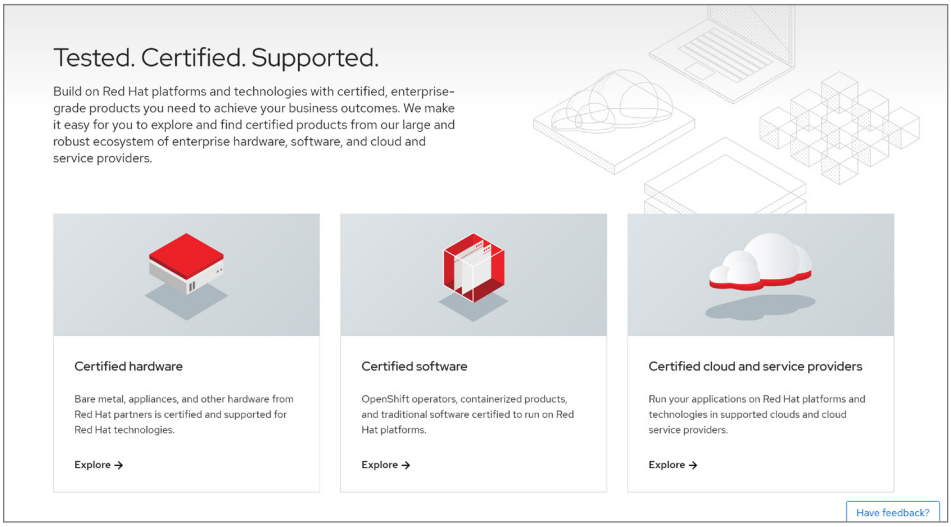


Figure 8. Certified partner products

Software companies that build on UBI can join **Red Hat Partner Connect** to get their products Red Hat certified and published in the catalog. Learn more in **section 4.7**.

Listing your software in the Red Hat Ecosystem Catalog makes it easy for people searching for a trusted solution to find your product in the same place they find Red Hat products. Examples are shown in Figure 9 below.

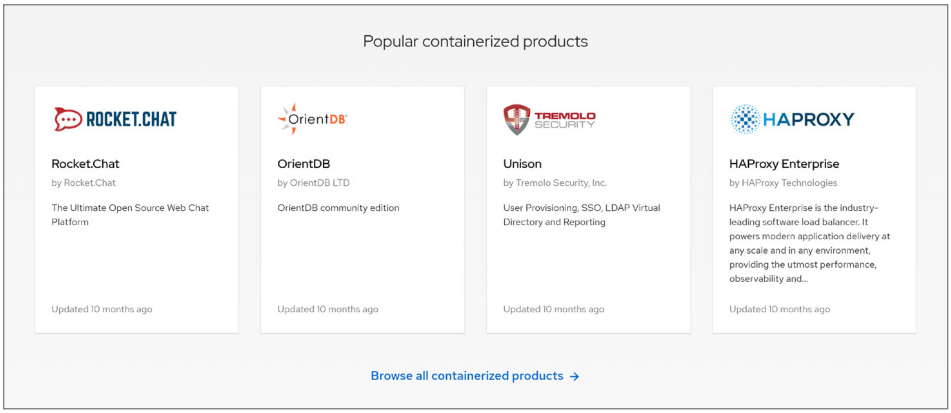


Figure 9. Partners listed in the Red Hat Ecosystem Catalog

4.5 Container health index

The Red Hat Ecosystem Catalog publishes a container health index for Red Hat container images as well as certified container and Kubernetes operator images published by Red Hat Partner Connect members. The index consists of a letter grade from A to F:

Grade A: This image does not contain known unapplied errata that fix critical or important flaws.

Figure 10. A letter grade indicating container health

The container health index and accompanying security and errata information associated with a container image are meant as helpful resources. Each user needs to determine risk based on the container health index, their use case, and any other information available to them. Read more about how the **Red Hat product security team rates the impact of security issues found in Red Hat products**.

As part of the Red Hat Container Certification and Red Hat OpenShift Operator Certification submission process, partner container images are scanned to extract metadata and information regarding included Red Hat RPMs. The scanned RPM information is compared with both Red Hat and public security advisory and vulnerability sources (**Red Hat OVAL v2 streams**).

These container images are then graded based on Red Hat published security updates that have or have not been applied and the length of time the software in the container images is exposed to those flaws. The grading system used is called Container Health Index for Red Hat Content. In order to certify a new container image, the image must have a health index grade of "A".

Read about **recommended practices** for partners to maintain the health index of a certified product.

As images age, and more security issues are discovered (CVEs, etc.), the grade drops. Container images age like cheese, not like wine. The Red Hat Ecosystem Catalog is a great place to find secure images because it gives friendly reminders to pull the latest images with the best grades. The container health index grades are derived from Red Hat's ratings of Red Hat developed images – read more about them in this introductory **article**.

4.6 Red Hat Vulnerability Scanner Certification

A number of third-party security companies offer services and tools for vulnerability scanning of container images. However, the source security data they use to identify vulnerability risks associated with Red Hat packages can vary and result in false positives, resulting in a growing and disruptive customer experience. To address this, Red Hat has created Red Hat Vulnerability Scanner Certification for security partners to standardize on Red Hat sourced security data. This means certified vulnerability scanning partners are in place to scan any UBI-based images that you build on.

Red Hat Vulnerability Scanner Certification is a certification to validate how security software partners use Red Hat Product Security recommended Red Hat security-related data to identify vulnerabilities for UBI images (and Red Hat products and packages). This lets security partners deliver more reliable, consistent and accurate reporting to customers to minimize false positives and other discrepancies. This capability is included in certified vulnerability scanning partner solutions – you don't have to develop yourself.

Learn more about Red Hat Vulnerability Scanner Certification and UBI.

4.7 Partnering with Red Hat

Build with Red Hat

Red Hat Partner Connect is Red Hat's partner program for enterprise hardware, software, and cloud and service providers.

Red Hat Partner Connect offers three ways for partners to engage with Red Hat:

Build on a hybrid cloud platform.

Sell to grow your business.

Service to develop deeper relationships.

In addition to the rights granted by the Red Hat UBI EULA, **partner companies that join Red Hat Partner Connect can also include any Red Hat Enterprise Linux user-space packages in UBI-based container images and freely redistribute them through both official Red Hat and third-party container registries. This means that these partners are not limited to redistribute images only tagged as "UBI"**.

IDC research shows improved ROI for Red Hat software partners

Red Hat contracted with IDC on a research report that confirms numerous and significant quantitative benefits for technology partners (e.g., software vendors) when they build and certify software on Red Hat platforms and technologies. Surveyed partners that are part Red Hat Partner Connect and certified their software obtained significant improvements in ROI (4969% over 3 years – this is not a typo), revenue (49%), and development life cycle (17%).

When partners build and certify with Red Hat, together the partner and Red Hat make sure the solutions are consistent, interoperable, and supported so partners can deploy with confidence and focus on delivering transformative technology to their customers. Partners that Build and certify their products gain the following benefits:

- Marketing resources – Red Hat Ecosystem Catalog, co-branded events, and solution briefs.
- No-cost software – Speed products to market with Red Hat platforms and development tools.
- Product training – In depth online training to help partners use and incorporate Red Hat products and technologies in dev and ops.
- Technical assets – Extensive knowledge base.
- Expanded routes to market – Competitive partner go-to-market models to incorporate Red Hat products, as well as Red Hat Marketplace for the hybrid cloud.
- Red Hat Container Certification – Build a tried, tested, and trusted technology stack in a container, get it Red Hat certified. See section 4.8.
- On-going knowledge transfer – Continuous training and support, discovery sessions, and technology webinars.
- Promote your certified product in the Red Hat catalog (see section 4.4).
- Sell your product in Red Hat Marketplace. Read more below.

Listing your certified applications in the Red Hat Ecosystem Catalog

Listing your certified application in the Red Hat Ecosystem Catalog is easy. During the process of submitting an application for Red Hat certification, there is an option to check whether your product should be listed. If you click Yes, your application will be listed once the certification process has been completed.

Listing your certified applications in the Red Hat Marketplace

Red Hat Marketplace is an open, cloud marketplace that makes it easier to discover and deploy certified software for container-based, public cloud or on-premise environments. Red Hat certified partners can promote and sell their software for Red Hat OpenShift through the Red Hat Marketplace. Software in the Red Hat Marketplace is immediately available for automated deployment on any Red Hat OpenShift cluster providing a fast, integrated experience.

The collection of software in Red Hat Marketplace is Red Hat certified and optimized to enhance deployments on Red Hat OpenShift using the automation capabilities provided by Kubernetes Operators. Customers can access open source and proprietary software, with responsive support, streamlined billing and contracting, simplified governance, and single-dashboard visibility across clouds.

Red Hat Marketplace is a collaboration of Red Hat and IBM. Learn how to join Red Hat Marketplace.

Learn more about joining Red Hat Partner Connect.

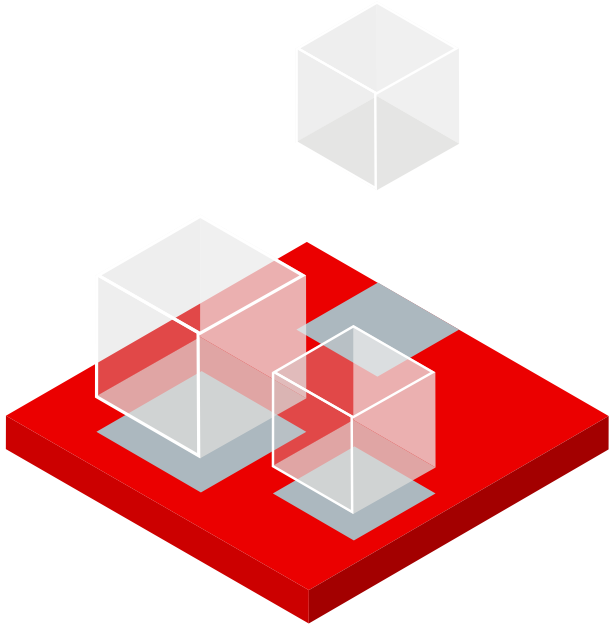
4.8 Red Hat Container Certification

Red Hat Universal Base Images can be redistributed anywhere by anyone, but they don't include everything in Red Hat Enterprise Linux. However, Red Hat Partner Connect members can redistribute everything else in Red Hat Enterprise Linux except the Linux kernel as part of their certified application.

Red Hat, through its Partner Connect program, has expanded the scope of the Red Hat Container Certification to allow Red Hat technology partners to include Red Hat Enterprise Linux user space packages. This means that when Red Hat partners build upon UBI, they can use any package from Red Hat Enterprise Linux user space required to run their software (except the Linux kernel) and re-distribute these certified images through Red Hat or non Red Hat container registries (e.g., Quay.io, Docker.io, a private registry, etc.).

To be able to freely distribute your container images that include Red Hat Enterprise Linux user-space packages, join Red Hat Partner Connect and submit your containerized software for Red Hat Container Certification.

See section 3.5 to learn about licensing and redistribution rights.



Part 5

Red Hat and open container tools

This section covers Red Hat's approach to container tools. An overview of the container tools that Red Hat is driving in open communities is covered. Compatibility with OCI and Docker is also discussed.

5.1. UBI works with any OCI-compliant container tools (including Docker)

Given the amount that has been written about Red Hat's work on open container tools, it might be easy to get the impression that you need to use Red Hat container tools, like Podman, to use UBI. Perhaps it can't be stressed enough that you can use UBI with any OCI-compliant container tools, including Docker. You can develop with UBI on Windows or macOS using Docker Desktop, or using your choice of Linux distribution and container runtime and engine. Compatibility and freedom are the goals of open standards.

UBI images are fully compatible with Docker and other OCI-compliant container tools including Podman and Buildah.

5.2. Docker on Red Hat systems

Red Hat includes a number of OCI-compliant container tools, such as Podman, Buildah, and Skopeo, in Red Hat Enterprise Linux and Red Hat OpenShift. These tools and the motivation creating for them are described in detail below.

Because of the preference for OCI-complaint container tools, Red Hat does not include Docker packages in Red Hat Enterprise Linux 8. For Red Hat Enterprise Linux 7, there were some early Docker packages available from Red Hat. However, these packages have been deprecated since the release of Red Hat Enterprise Linux 7.5 in 2018. While those Docker packages from Red Hat are still available in the Red Hat repositories for Red Hat Enterprise Linux 7, they are frozen at version 1.13 for meeting compatibility assurance. For the last several years, Red Hat Enterprise Linux 7 releases have included Podman, Buildah, and Skopeo.

Red Hat OpenShift 4 defaults to CRI-O, which is OCI compliant, as the underlying container engine. Kubernetes deprecated support of the Docker container engine in version 1.20. The use of CRI-O as a lightweight alternative to the Docker container engine is increasing in Kubernetes deployments.

Docker Inc.'s Docker-ce or Docker-ee can be installed on Red Hat Enterprise Linux

While Red Hat offers a set of container tools that has many advantages over Docker, those who prefer Docker's tools can install Docker-ce or Docker-ee on Red Hat Enterprise Linux. Before installing Docker-ce or Docker-ee, `podman` and `runc` will need to be removed to avoid a conflict between packages. This can be accomplished by removing container tools with the following command on Red Hat Enterprise Linux 8:

```
# yum module remove container-tools
```

On Red Hat Enterprise Linux 7, use the following command:

```
# yum remove podman runc
```

You can now add Docker Inc.'s YUM repositories to your system and follow the directions on docker.com. In the past, there were some issues with the repositories that complicated the installation, but those have been resolved.

Using docker CLI commands with Podman and Buildah

Knowledge and experience with the Docker command line interface (CLI) can still be used with Red Hat's container tools. For those who are familiar with the Docker CLI, Red Hat ships a package, `podman-docker`, that provides a docker-compatible command and uses Podman and Buildah underneath. Note that enough of the podman CLI is compatible with the docker CLI that you could also just alias `docker` to `podman` and be able to perform most operations without knowing which container tools are installed.

5.3. The motivation for open container tools

Red Hat saw a need for community driven open source container tools that have an architectural focus based on principles Red Hat and Red Hat's customers have identified as important. The main driver is that container tools need to be smaller, more modular, and more secure. This needs to be evaluated in the context of the ways container tools will be used. On a developer's laptop, usability and agility are the highest priority. Ease of use and access to the latest features that speed development are much more important than security. In large production environments running containers at scale, the highest concerns are security and reliability. Stability is much more important than introducing new functionality.

It is also worth noting that the container tools used will be different depending on the size of the workloads. An organization that runs five or less containers per host on a small number of machines wants smaller, simpler tools with a shallow learning curve. An enterprise that is running hundreds of containers on large clusters of machines needs a container orchestration platform with a wealth of features for scheduling, managing, and migrating workloads. Kubernetes has emerged as a de facto standard for orchestrating containerized workloads. Container management platforms like Red Hat OpenShift are based on Kubernetes.

Smaller, more modular tools have a number of advantages. More core technologies can be easily shared between tools like a CLI for managing containers on a single host and components of the Kubernetes platform. It is expected that there will continue to be rapid innovation in the container tools space. Introducing significant changes into a monolithic tool without breaking it for existing users can be quite a challenge. Smaller, more modular tools can be evolved more quickly. Having a set of tools allows each tool to focus on a single purpose. New tools can be added to add new functionality or to experiment with ideas and architectures that might be incompatible with existing tools. Finally, smaller and more modular tools are easier to secure.

Some of the initial goals Red Hat had for investing in open container tools include eliminating the security and other problems associated with a daemon running as root that controls all container operations. A daemon shouldn't be required to build container images. This can complicate automated builds. Using a daemon is problematic in environments like Kubernetes and public and private clouds.

The Open Container Initiative (OCI) is a project within The Linux Foundation with a goal of open industry standards for container formats and container runtimes. A number of vendors, cloud providers, and others collaborate on specifications for container runtimes, and the reference implementation (runc), which is used by the majority of container engines including Docker, podman, and CRI-O.

The ability to truly run containers as a non-root user, without compromising the security of the system, was another initial goal. With Docker, non-root users can run containers when given access to the Docker daemon. However, opening up access to the docker daemon allows users to perform many operations that require root privileges and can be used to effectively gain root access to the whole system. Therefore, giving non-root users, like developers, access to the Docker daemon on servers and production systems is an unacceptable risk for many organizations.

Finally, Red Hat wanted to be able to foster rapid innovation and community participation by letting the development process for more modular container tools happen in open community sites that focus on specific functional areas.

The result of this initiative are Podman, Buildah, CRI-O, and a number of other tools and libraries.

5.4. Overview of Red Hat's open container tools

There are three main command line tools for working with containers:

Podman – the primary tool for running containers and *pods* of multiple containers working together. The `podman` CLI does almost all of the same things that the `docker` command does and quite a bit more, and does it without requiring a daemon.

Buildah – specializes in building OCI images. The commands in buildah's CLI replicate all of the commands that are found in a `Dockerfile`. However, buildah's commands can be run from any scripting or programming language, opening up an enormous range of build automation capabilities compared with a single domain-specific language for building container images.

Skopeo – used for many of the tasks related to sharing and manipulating container images and image repositories. Skopeo understands the majority of container image formats and can be used to convert between them.

To make things easy for people to get started, `podman` accepts almost all of the same commands as `docker`. You can use the `podman` command for pulling, running, and building containers the same way you would use the `docker` command. They both cover the same basic and intermediate use cases. As you get deeper into working with containers, you are likely to discover advanced use cases that Buildah and Skopeo were designed to address.

The `podman` commands were designed to be compatible enough with `docker` that you could just alias the `docker` command to call `podman` (alias docker=podman). However, the `podman-docker` package provides a `docker` compatible CLI that uses Podman, Buildah, and Skopeo.

Podman: A tool for managing containers and pods

Podman is a complete container engine for running and managing OCI containers and container images on Linux systems. Podman does not require a separate daemon to manage containers. In addition, unprivileged users can run containers without root privileges and without a daemon that provides root capabilities.

As a `docker` compatible CLI, Podman is used for pulling, tagging, and sharing container images and running and managing containers created from those images. Container images can be built with Podman, though more advanced building capabilities are available in the complementary project, Buildah. In addition to OCI, Podman supports other image formats including Docker images.

Podman is available on most Linux platforms, it is an open source, community driven project. Development for Podman and related tools occurs on **GitHub** as part of the **containers project**. Podman manages containers, groups of containers called pods, and container images, volumes, storage, and networking, using the **libpod** library. Libpod provides an API for sharing code with other container tools.

Today, as a service for running containers, Podman only runs on Linux platforms. A REST API and clients are currently under development, which will allow clients on Mac and Windows to call Podman as a service running on a Linux platform. This would include Podman running on WSL2, the Windows Subsystem for Linux, that includes a Linux kernel.

Podman contains a number of features to help make the transition from running a handful of containers on a single machine to container orchestration across multi machine clusters using Kubernetes. The pod concept is similar to the concept in Kubernetes. The configuration for locally running containers can be captured and output as the YAML files that are necessary for running on Kubernetes.

For more information:

- Try podman online, without installing any software: **lab.redhat.com/podman-deploy**.
- See **Podman and Buildah for Docker** users and the latest **Podman** and **container** articles at **Red Hat Developer**.
- Learn what's happening in Podman development at Podman community website **podman.io** and the **Podman project on GitHub**.

buildah

Buildah: A tool for building container images

Buildah allows you to build and modify containers without requiring the installation of any daemon or docker. While Buildah can be used with existing Dockerfiles, Buildah provides much greater power and flexibility that gives developers fine-grained control over image layers, content, and commits.

Container images can be created using existing OCI and Docker base images, a working container, or even completely from scratch. Buildah also provides the flexibility to mount a container image as part of the filesystem, run any commands or processes to modify the contents, and then save the changes as an updated container image.

A key feature of Buildah's flexible methods for container creation is that build tools can be external to the container you are building. With traditional Dockerfile builds, tools such as GCC compiler (GCC), make, git, etc., have to be pulled into the container as it is built. This makes the resulting container larger and less secure as it contains software that isn't needed at runtime. Container image build performance can also be improved by not having to install the same build-related packages for each container build.

The advantages of building containers without the requirement for a container daemon become apparent when the process moves from a developer's machine to an automated build system that provides continuous integration and delivery (CI/CD) running in containers or a cloud environment. Daemonless container building avoids the issues with running Docker in Docker and the security risks associated with leaking access to the host's Docker daemon inside of containers.

Not requiring a daemon makes it easier to create completely isolated build processes running in containers that are disconnected from the host system. The build system inside the container can be running the latest version of Buildah, without regard for which,if any version, is available on the host.

For more information:

- Try Buildah online, without installing any software: **lab.redhat.com/buildah**.
- See **Podman and Buildah for Docker users**, **Best practices for running Buildah in a container**, and the latest **Buildah** and **container** articles at **Red Hat Developer**.
- Learn what's happening in Buildah development at the Buildah community website **buildah.io** and the **Buildah project on GitHub**.

skopeo

Skopeo: A tool for working with container registries and images

Skopeo is a comprehensive tool and library for manipulating, inspecting, signing, and transferring container images and image repositories. This command line tool complements Podman and Buildah with advanced functionality to move container images between registries and to inspect, verify, and sign image manifests. Most container image formats, including OCI and Docker, are handled by Skopeo, which can be used to convert between different formats. Skopeo can be used to move images between different container storage mechanisms on local hosts in addition to local and remote container registries. You can inspect a container image to show the layers it contains without the need to pull the image first.

Similar to Podman and Buildah, skopeo is an open source community driven project that does not require running a container daemon. Unprivileged users can run most of Skopeo's commands, except of course those that require root access to manipulate the host system.

Skopeo 1.0 released, a blog by Red Hat Senior Distinguished Engineer Dan Walsh, gives an overview of Skopeo and its history. It is worth noting that Skopeo came into existence when Red Hat submitted a pull request to the upstream Docker project to be able to inspect a remote image via `docker inspect --remote IMAGE`. The pull request was rejected because the maintainers didn't want to complicate the Docker CLI. This underscores the advantages of a more modular suite of container tools.

Similar to the way parts of Podman's functionality comes from the libpod library that allows code to be shared with other tools, Skopeo's functionality is also implemented in a library. Skopeo's containers/image library is shared by other container engines including Podman, Buildah, and CRI-O.

For more information:

- See the blogs Red Hat Enterprise Linux 8 enables containers with the tools of software craftsmanship and Skopeo 1.0 released on RedHat.com.
- Learn about verifying container Image signatures.
- Find out what's coming in Skopeo development at github.com/containers/skopeo.

Udica: A tool that generates SELinux policies for containers

To improve security, administrators and container developers can use Udica to create security policies that give a running container only the exact minimum security capabilities it needs to run. Udica analyzes a container and generates the extra controls to work with the default policy to enforce a principle of minimum privilege.

With Udica, a tailored security policy is created for better control of how a container accesses host system resources like storage, devices, and networks. This lets you harden your container deployments to protect the host system as well as other containers against malicious activity or unintended behavior. This hardening also makes it easier to achieve and maintain regulatory compliance.

Note that the standard SELinux policies on Red Hat Enterprise Linux provide good general protection by dynamically separating running containers using auto-generated Multi-Category Security (MCS) labels for each container. The additional controls with Udica provide another layer of protection.

For more information:

- Try a hands-on tutorial: Generating SELinux policies for containers with Udica at lab.redhat.com/selinux-containers.
- Learn about creating SELinux policies for containers from the Red Hat Enterprise Linux 8 using SELinux guide.
- Follow Udica development at github.com/containers/udica.

CRIU: Checkpoint and restore containers in userspace

Checkpoint/restore in userspace (CRIU) is a community project to implement checkpoint/restore functionality in Linux. Together with Podman, CRIU can freeze a running container and save its full state, including memory image, to disk. For processes that take a long time to start up, this can provide a significantly faster restart time by restoring the running container state from the checkpoint on disk. In addition, the container can be restored on another system to allow for stateful container migration. Using this functionality a number of other things are now possible, like periodic snapshots to checkpoint long running processes.

Examples of applications that can benefit the most from this functionality are large Java VMs and processes like database servers with large in-memory caches that can take a long time to get filled with the optimal running state.

For more information:

- Try a hands-on tutorial using Microsoft SQL Server at lab.redhat.com/sql-server-ubi.
- Learn how to use container migration with Podman on Red Hat Enterprise Linux.
- Read the article Checkpointing Java from outside of Java by Christine Flood, a Red Hat Senior Principal Software Engineer.
- Follow CRIU development at the community website criu.org.

CRI-O: A lightweight container runtime for Kubernetes

CRI-O is an implementation of Kubernetes' container runtime interface (CRI) that was launched by Red Hat and is now an incubating project within the Cloud Native Computing Foundation (CNCF). Contributors from multiple organizations participate in this community driven, open source project.

CRI-O is an OCI-compatible container runtime that is a lightweight alternative to Docker for Kubernetes environments. CRI-O supports multiple image formats, including OCI and Docker, and multiple means of downloading and verifying images to make sure the images can be trusted.

Starting with Red Hat OpenShift version 4, the default container engine for Kubernetes is CRI-O. The containers/image, containers/storage, and containers/common libraries used by Podman, Buildah, and Skopeo are also used by CRI-O. Sharing this common foundation benefits these tools and future projects.

For more information:

- See Why Red Hat is investing in CRI-O and Podman, a blog by Red Hat's Scott McCarty and Dan Walsh.
- See the latest in CRI-O development at the community website cri-o.io and the CRI-O project on GitHub.

5.5. Getting started with open container tools

Container tools on Red Hat Enterprise Linux 8

Red Hat Enterprise Linux 8 includes multiple application streams for container tools to address different needs for the latest features versus compatibility and stability. If you aren't familiar with application streams and modules in Red Hat Enterprise Linux 8, it is a delivery mechanism for providing multiple versions of software during the 10+ year life cycle of Red Hat Enterprise Linux 8. See Introduction to Application Streams in Red Hat Enterprise Linux 8.

Install `container-tools:rhel8` to get the latest stable versions of Podman, Buidah, and other container tools on Red Hat Enterprise Linux 8.

Table 6. Container tool application streams in Red Enterprise Linux 8

Application Stream	Purpose
container-tools:rhel8	Fast stream package of Podman, Buildah, and other tools. Updated approximately four times a year.
container-tools:1.0	Stable stream release of Podman, Buildah, and other tools. Only updated for security and necessary bug fixes to ensure compatibility and stability. Released with Red Hat Enterprise Linux 8.0, May 2019.
container-tools:2.0	Stable stream release of Podman, Buildah, and other tools. Only updated for security and necessary bug fixes to ensure compatibility and stability. Released with Red Hat Enterprise Linux 8.2, May 2020.

Developers and others who wish to use the latest stable versions of Podman, Buildah, and Skopeo can install the *fast* stream, which is updated approximately four times per year. The fast stream is in the `container-tools:rhel8` software module.

To install Podman, Buildah, and Skopeo on Red Hat Enterprise Linux 7, enable the *Red Hat Enterprise Linux Extras RPM repository*.

To install the fast stream use:

```
# yum module install container-tools:rhel8
```

For production use, the *stable* stream of the container tools packages are grouped into a fixed major release like 1.0. The packages that make up the 1.0 release stay at the same major version with only updates for security and bug fixes as deemed necessary. Compatibility and stability are the primary goals, so new functionality that might affect those goals is not added into the 1.0 packages. It is expected that a new stable stream major release, such as a 2.0 or 3.0, will be added approximately once a year. Once released, the packages in the stable release, such as 1.0, have a two-year life cycle for updates.

To install the 2.0 stable stream use:

```
# yum module install container-tools:2.0
```

To list which container-tools application streams are available, use:

```
# yum module info container-tools
```

For compatibility with the `docker` CLI, you can install the optional `podman-docker` package. You can still type `docker` commands, but they will be executed by Podman and Buildah.

```
# yum install podman-docker
```

For a good overview of the container tools in Red Hat Enterprise Linux 8, see Scott McCarty's article, **RHEL 8 enables containers with the tools of software craftsmanship**. Updated information on Container Tools 2.0 is available in Scott's article, **New container capabilities in Red Hat Enterprise Linux 8.2**.

The life cycle for container tools is documented in **Container Tools AppStream – content availability** on the Red Hat Customer Portal.

Container tools on Red Hat Enterprise Linux 7

Red Hat Enterprise Linux 7 was released in June of 2014. It originally included `docker` RPMs from Red Hat. As of the release of Red Hat Enterprise Linux 7.5 in 2018, support for the `docker` RPMs from Red Hat were deprecated. Packages for `podman` and the newer container tools were added at that time. It is important to note that the 10+ year life cycle started in 2014, and that Red Hat is committed to maintaining compatibility throughout the entire 10+ year life cycle that started with Red Hat Enterprise Linux 7.0.

Red Hat Enterprise Linux 7 is now in the **maintenance support** phase of its life cycle. No new functionality will be added.

The final update to the container tools packages occurred with Red Hat Enterprise Linux 7.8. This release included the last major updates to the container tools packages, which included Podman 1.6.4, Buildah 1.11.6, and Skopeo 0.1.41. The ability to run containers as a non-root user (rootless containers) became generally available with the 7.8 release. It had been a technology preview since the 7.6 release.

The container tools packages are in the *Red Hat Enterprise Linux Extras* RPM repository, `rhel-7-server-extras-rpms`. This Red Hat repository is not enabled by default. These packages are in the extras repository instead of the primary Red Hat Enterprise Linux 7 package repositories because they don't have the same support life cycle as the main OS.

To install container tools on Red Hat Enterprise Linux 7, run the following commands:

```
# subscription-manager repos --enable rhel-7-server-extras-rpms
# yum install podman buildah skopeo
```

At this point, you can run containers using Podman on Red Hat Enterprise Linux 7. To run UBI 8, use this command:

```
# podman run -it ubi8/ubi
```

For compatibility with the `docker` CLI, you can install the optional `podman-docker` package. You can still type `docker` commands, but they will be executed by Podman and Buildah.

```
# yum install podman-docker
```

To run rootless containers there is a UID/GID mapping step for your non-root users. If your installation was a fresh install of Red Hat Enterprise Linux 7.8 or later, this will be performed automatically for you. However, if you had installed an earlier version and then upgraded, a few additional steps are necessary to create mappings in `/etc/subuid` and `/etc/subgid` for existing users. See the Managing containers guide in the Red Hat Enterprise Linux 7 documentation.

For more information on container tools in Red Hat Enterprise Linux 7, see the overview in this article Red Hat Enterprise Linux 7.8 and the final update to container tools.

Other Linux distributions

Packages for Podman, Buildah, Skopeo, and their dependencies are available as part of many recent Linux distributions, including Fedora, CentOS streams, and Ubuntu. If packages aren't available in the main package repositories, they might be available in the testing or backports repositories. This is the case for the current stable release of Debian.

See your system's package repositories and documentation for more information.

Alternate packages and building the latest from source

The community websites contain links to alternate sources of packages for a number of distributions, including recent or nightly builds. Information on how to build from source is also available. See:

- podman.io for Podman.
- buildah.io for Buildah.
- github.com/containers/skopeo for Skopeo.
- github.com/containers/udica for Udica.
- criu.org for CRIU.

Buildah can also be run in a container. This allows you to run the latest version of Buildah or any specific version needed for your build environment without concern for which version is running on the host. See Dan Walsh's article Best practices for running Buildah in a container.

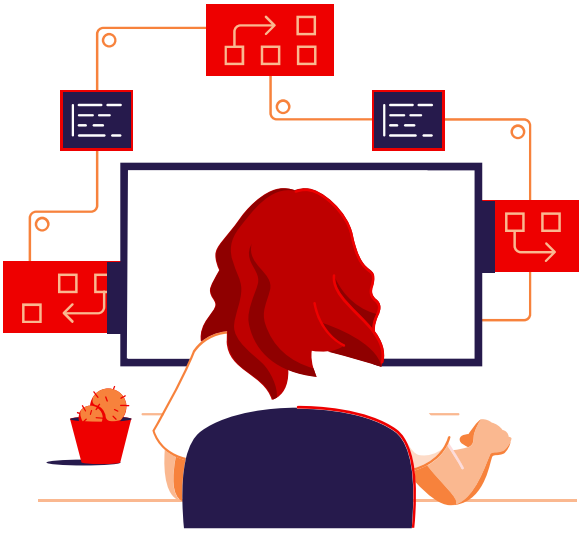
Container tool tutorials

You can try Podman, Buildah, and other tools on-line, without installing any software, in a live containerized environment at lab.redhat.com and learn.openshift.com.

- Deploying containers using Podman – lab.redhat.com/podman-deploy.
- Create container images online with Buildah – lab.redhat.com/buildah.
- Containerize a third-party package in a UBI container using Buildah – lab.redhat.com/containerize-app.
- Generating SELinux policies for containers with Udica – lab.redhat.com/selinux-containers.
- Use CRIU with Microsoft SQL Server – lab.redhat.com/sql-server-ubi.
- The Linux container internals 2.0 series of labs provides an in-depth overview of container fundamentals including tools, registries, hosts, and standards in an interactive environment using Podman and UBI – learn.openshift.com/subsystems.

For tutorials you can follow on your own systems, see:

- Podman and Buildah for Docker users.
- Podman cheat sheet.
- The latest articles on Podman, Buildah and containers at Red Hat Developer.
- Building, running, and managing containers from the Red Hat Enterprise Linux 8 documentation.
- For Red Hat Enterprise Linux 7, see the Managing containers guide in the Red Hat Enterprise Linux 7 documentation.
- There are tutorials, blogs, and documentation on the community sites, podman.io, buildah.io, and the respective GitHub repositories. Note that these may be for the latest versions of code that have not been released or packaged yet for Red Hat Enterprise Linux or other distributions.



Part 6

Working with UBI

6.1. Where to find UBI container images

UBI container images are available to be pulled from Red Hat container registries. However, the Red Hat Ecosystem Catalog gives more detailed information including:

- Update history and security information, including a letter grade for current security status.
- Instructions for obtaining the image, including `docker pull` or `podman pull` command lines.
- A list of RPM packages included in each image, and which packages have been updated since release.
- A Dockerfile that represents the steps used to build the image.

To find UBI images, go to the container images section of the Red Hat Ecosystem Catalog, catalog.redhat.com/software/containers/search. Then select either *Red Hat Universal Base Image 8* or *Red Hat Universal Base Image 7* in the *Product* selection box on the left or use one of the following links:

- Red Hat Universal Base Image 8.
- Red Hat Universal Base Image 7.

UBI images are not on Docker hub, but are freely available at registry.access.redhat.com, Red Hat's container registry. You can also find UBI images and instructions for pulling them in the Red Hat Ecosystem Catalog.

The list includes the base UBI only images, and the language and other runtime images built on that UBI version.

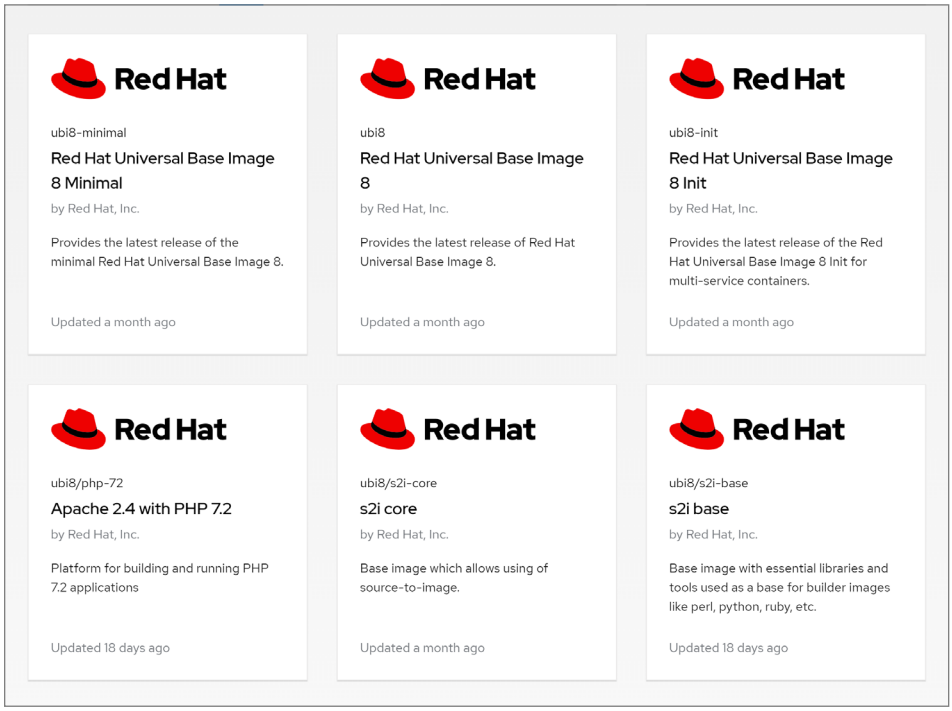


Figure 11. UBI images in the Red Hat Ecosystem Catalog

You can also find UBI images by searching the Red Hat container registries using the command line.

Red Hat container registries

Container registry background

If you haven't worked with container registries other than Docker's public registry, some background might be helpful. Many tutorials, especially from the early days of Linux containers, showed examples for pulling containers without specifying a registry. Most container images at that time were available on the Docker Hub repository. Not specifying a registry works because most tools have a list of registries to search when the registry isn't specified. Out of the box docker.io is configured as one of the earlier entries in most tool's registry search path.

Today, companies that produce software are increasingly likely to operate their own container registries and provide their own sites for navigating these registries. This allows those companies to add their own value and branding to their sites. It also avoids reliance on a third party to operate a single, somewhat monolithic site.

As companies can have software that isn't publically available, some of these registries only allow access by authorized users. There is a login process using `docker login` to authenticate to a remote registry. Most tutorials don't have this step since they are using a public registry that doesn't require authentication.

Many company's registries will not be included in the default registry search path for your container tools. However, images can easily be pulled from a specific registry by simply adding the registry's host name as part of the container image name. This will become increasingly common as more companies start operating their own registries.

For more information, the Linux container internals 2.0 series of labs provides an in-depth overview of container fundamentals including registries, tools, hosts, and standards in an interactive environment using Podman and UBI – learn.openshift.com/subsystems.

Red Hat operates three container registries. One registry is for publicly available software like UBI that requires no authentication or registration. Another registry contains Red Hat software that isn't freely redistributable, like Red Hat Enterprise Linux and Red Hat Middleware. The third registry is for third-party software from Red Hat partners that has been certified by Red Hat. Authentication is required to use the latter two registries. Table 7 is a summary of the Red Hat container registries:

Table 7. List of Red Hat container registries

Red Hat Registry	Software Available	Authentication required?
registry.access.redhat.com	UBI	No
registry.redhat.io	Red Hat software that isn't freely available like Red Hat Enterprise Linux Also has UBI images for convenience	Yes
registry.connect.redhat.com	Third-party products certified by Red Hat	Yes

UBI is freely available without authentication from `registry.access.redhat.com`. No subscription or login is required. UBI images are not available on Docker Hub at this time, so you need to include `registry.access.redhat.com` when referring to images.

Here are some examples using Docker. You can find UBI images by searching the registry:

```
$ docker search registry.access.redhat.com/ubi8/ubi | sort
```

To pull the UBI 8 platform image use the following command:

```
$ docker pull registry.access.redhat.com/ubi8/ubi
```

Similarly, the UBI 7 platform image can pulled using the following command:

```
$ docker pull registry.access.redhat.com/ubi7/ubi
```

If you are using Podman the commands are the same:

```
$ podman search registry.access.redhat.com/ubi8/ubi | sort
$ podman pull registry.access.redhat.com/ubi8/ubi
$ podman pull registry.access.redhat.com/ubi7/ubi
```

Using podman on Red Hat Enterprise Linux, you can omit the registry name as `registry.access.redhat.com` and `registry.redhat.io` are already in the search path:

```
$ podman pull ubi8/ubi
```

If you aren't using Red Hat Enterprise Linux, you could modify the registry search path used by your container tools to allow you to omit the registry name and simply refer to `ubi8/ubi`. For Podman, this is usually done in `/etc/containers/registries.conf` or`/etc/containers/registries.d`.

During development it is good to limit the amount of typing required. From a security perspective, the recommended practice is to always specify the registry you intend to use to avoid any ambiguity. Whether unintentional or malicious, it would be an unpleasant surprise if someone created an image with the same name, but different contents in a commonly used public registry.

If you are following a tutorial that uses `ubi8/ubi` and it doesn't work on your system, add `registry.access.redhat.com/` before `ubi8/ubi`. That should work on most systems and container tools.

Working with Red Hat's authenticated container registry

Red Hat also maintains a registry, `registry.redhat.io`, that contains Red Hat content that isn't freely redistributable, such as Red Hat Enterprise Linux and Red Hat Middleware. Authentication is required for this registry. It includes container images that aren't currently available as part of UBI, like database servers that are part of Red Hat Enterprise Linux 8 Application Streams or Red Hat Software Collections. The advantage of using those images is that they are maintained by Red Hat and support is available from Red Hat.

If you have a Red Hat subscription, the same Red Hat login you use to log into the Red Hat Customer Portal can be used to authenticate to registry.redhat.io. If you don't have a subscription, no-cost subscriptions are available for Red Hat Enterprise Linux by joining either the Red Hat Developer Program or the Red Hat Partner Program.

For convenience, UBI is also available from the authenticated registry. There is no difference between the UBI images on the two registries. Freely available UBI images in the authenticated registries make it convenient for Red Hat users to be able to get both freely available and subscription-only content from a single place.

UBI is available from both Red Hat's unauthenticated and authenticated registries. The bits in the authenticated registry are identical to those in the unauthenticated one.

To log into Red Hat's authenticated registry using docker:

```
$ docker login registry.redhat.io
Username: {Red-Hat-Username}
Password: {Red-Hat-Password}
Login Succeeded!
```

Authenticating using `podman` or `skopeo` is very similar:

```
# podman login registry.redhat.io
Username: {Red-Hat-Username}
Password: {Red-Hat-Password}

# skopeo login registry.redhat.io
Username: {Red-Hat-Username}
Password: {Red-Hat-Password}
```

Logging in creates an authenticated session that is used when interacting with the registry that requires authentication. Registry related commands like `pull` and `search` do not require any changes. When your session expires, you will need to reauthenticate.

For automated processes, like CI/CD jobs, registry service accounts that use tokens are available. This avoids the need to embed usernames and passwords in build configurations and scripts.

More information about Red Hat's container registries, authentication, and registry service accounts can be found in **Red Hat Container Registry Authentication** on the Red Hat Customer Portal.

6.2. Guided online tutorials with UBI

Red Hat provides a number of guided tutorials you can try online without installing any software on your local machine. Through a browser, you have access to a live environment that is running containers. Here are some of the UBI tutorials:

- Build an application into a container image using Red Hat Enterprise Linux container tools. In this tutorial you containerize a third-party application UBI-based container using Buildah. To try it go to: **lab.redhat.com/containerize-app**.
- Run Microsoft SQL Server on Red Hat Enterprise Linux using UBI. This tutorial uses a SQL Server UBI container image provided by Microsoft. The tutorial covers running containers as root and as a non-root user. It also shows how CRIU can be used to checkpoint a container and restore the container from a checkpoint for improved start up performance. To try it go to: **lab.redhat.com/sql-server-ubi**.
- Learn how to use Buildah to move beyond the limitations of Dockerfiles. This tutorial covers creating and mounting a working container to show how the container-building process can be opened up beyond what is possible in a Dockerfile. Regular Interactive processes on the host can write to the container as part of the filesystem. Build tools on the host can be used without the effort of making the build tools available inside of the containers. To try it go to: **lab.redhat.com/buildah**.
- The Linux container internals 2.0 series of labs provides an in-depth overview of container fundamentals including tools, registries, hosts, and standards in an interactive environment using Podman and UBI. Each lab also links to a presentation on that topic. To try it go to: **learn.openshift.com/subsystems**.

You can find more online tutorials at **lab.redhat.com**.

6.3. Using UBI on Windows, macOS, and Linux with Docker

UBI images can be used the same way you use other base images. You don't need to be on a Red Hat system. You can run and build on Windows, macOS, or Linux using Docker or any OCI-compliant tools.

To find UBI images, run:

```
$ docker search registry.access.redhat.com/ubi | sort
```

To pull the UBI 8 platform image, run:

```
$ docker pull registry.access.redhat.com/ubi8/ubi
```

To run the UBI 8 platform image, use the following command:

```
$ docker run -it --rm ubi8/ubi bash
```

Now that you have a shell running inside of the UBI container, here are a few commands you can use to explore the container. First, check which release the container is based on:

```
# cat /etc/os-release
```

See which RPM packages are installed in the base image by first getting a count of installed pages and then listing them. Run these commands inside of the container:

```
# rpm -qa | wc -l
# rpm -qa | sort | more
```

After exiting the container, you can try pulling and running the `ubi-minimal` image, or language runtime images like `python-38` to compare the number and list of packages installed:

```
$ docker run -it --rm registry.access.redhat.com/ubi8/ubi \
    rpm -qa > rpms-ubi.txt

$ docker run -it --rm registry.access.redhat.com/ubi8/ubi-minimal \
    rpm -qa > rpms-minimal.txt

$ docker run -it --rm registry.access.redhat.com/ubi8/python-38 \
    rpm -qa > rpms-python.txt

$ wc -l rpms-*.txt
```

Note: It is much easier to explore the installed packages using the *Packages* tab on the container image's page in the Red Hat Ecosystem Catalog:

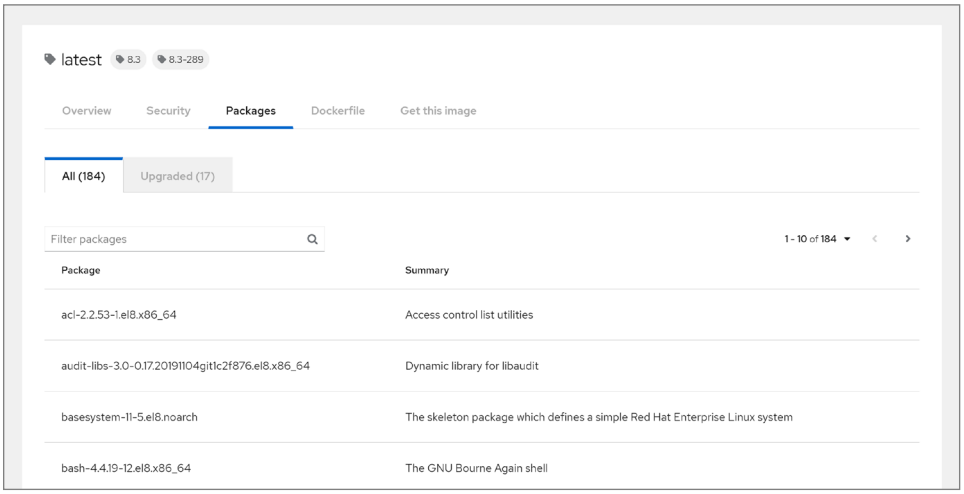


Figure 12. Containers image page of in the Red Hat Ecosystem Catalog

Next, build a container image using UBI and Docker. This example uses the UBI Node.js image to run a tiny web server written in Node.js. First check out the code from the sample application:

```
$ git clone https://github.com/sclorg/nodejs-ex.git app-src
```

Create a `Dockerfile` with the following contents:

```
FROM registry.access.redhat.com/ubi8/nodejs-14

# Add application sources
ADD app-src .

# Install the dependencies
RUN npm install

# Run script uses standard ways to run the application
CMD npm run -d start
```

Now you are ready to build the application:

```
$ docker build -t node-app .
```

You are likely to see some warning messages from npm during the build. These can be safely ignored.

Now you can run the application with the following command:

```
$ docker run --rm -p 8080:8080 -d node-app
```

Use `curl` to verify that the Node.js web application is running:

```
$ curl http://localhost:8080/ | head
```

You should see the first few lines of an HTML page. When you are finished, use `docker stop <container name>` to stop the container and free up port 8080 on your system.

To learn more about building container images with UBI, see section 6.6 Adding software to UBI images below. Another tutorial, Red Hat Universal Base Images for Docker users can be found on the Red Hat Developer blog.

6.4. Using UBI on Red Hat Enterprise Linux and systems with Podman

UBI images are used the same way as other base images you've likely already encountered. The Podman commands in this section are almost identical to the commands used with Docker. The following commands should work on any system with Podman installed, such as Red Hat Enterprise Linux, CentOS Stream, Fedora, or Ubuntu 20.10 and later. For more information on Podman, Buildah, and other OCI-compliant container tools, see Part 5 Red Hat and open container tools.

To find UBI images, run:

```
$ podman search registry.access.redhat.com/ubi | sort
```

To pull the UBI 8 platform image, run:

```
$ podman pull registry.access.redhat.com/ubi8/ubi
```

To run the UBI 8 platform image, use the following command:

```
$ podman run -it --rm ubi8/ubi bash
```

Now that you have a shell running inside of the UBI container, here are a few commands you can use to explore the container. First, check which release the container is based on:

```
# cat /etc/os-release
```

See which RPM packages are installed in the base image by first getting a count of installed pages and then listing them. Run these commands inside of the container:

```
# rpm -qa | wc -l
# rpm -qa | sort | more
```

After exiting the container, you can try pulling and running the `ubi-minimal` image, or language runtime images such as `python-38` too compare the number and list of packages installed:

```
$ podman run -it --rm registry.access.redhat.com/ubi8/ubi \
    rpm -qa > rpms-ubi.txt

$ podman run -it --rm registry.access.redhat.com/ubi8/ubi-minimal \
    rpm -qa > rpms-minimal.txt

$ podman run -it --rm registry.access.redhat.com/ubi8/python-38 \
    rpm -qa > rpms-python.txt

$ wc -l rpms-*.txt
```

Note: It is much easier to explore the installed packages using the *Packages* tab on the container image's page in the Red Hat Ecosystem Catalog:

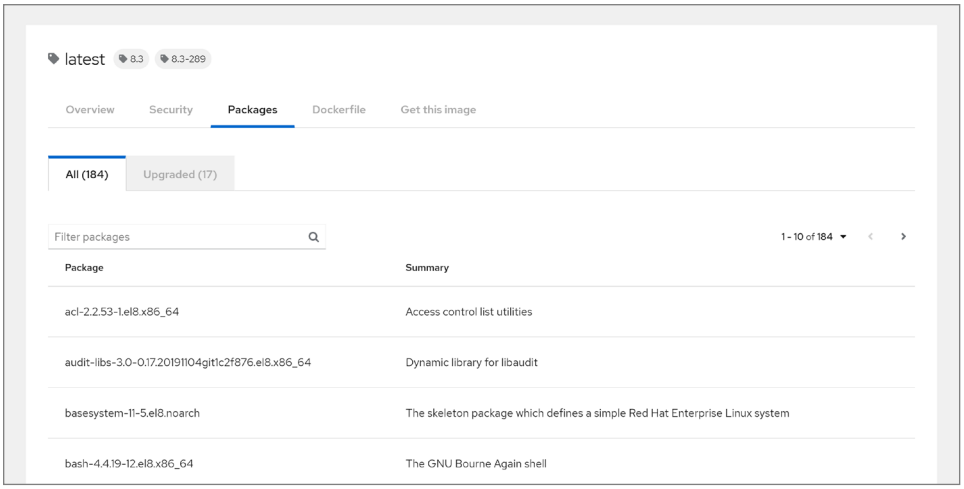


Figure 13. Container image's page in the Red Hat Ecosystem Catalog

Next, build a container image using UBI and Buildah. Use the UBI Node.js image to run a tiny web server written in Node.js. First check out the code from the sample application:

```
$ git clone https://github.com/sclorg/nodejs-ex.git app-src
```

Create a `Dockerfile` or `Containerfile` with the following contents:

```
FROM registry.access.redhat.com/ubi8/nodejs-14

# Add application sources
ADD app-src .

# Install the dependencies
RUN npm install

# Run script uses standard ways to run the application
CMD npm run -d start
```

Now you are ready to build the application:

```
$ buildah bud -t node-app .
```

You are likely to see some warning messages from npm during the build. These can be safely ignored.

Note: for the above command, you could have used the Docker compatible `podman build`. However, given the flexibility Buildah offers it is best to form the habit of using Buildah directly.

Now you can run the application with the following command:

```
$ podman run --rm -p 8080:8080 -d node-app
```

Use `curl` to verify that the Node.js web application is running:

```
$ curl http://localhost:8080/ | head
```

You should see the first few lines of an HTML page. When you are finished, use `podman stop <container name>` to stop the container and free up port 8080 on your system.

For more information, see the DevNation Video, **Building freely distributed containers with Podman and UBI**, with Scott McCarty and Burr Sutter.

6.5. Choosing between UBI base images

The UBI platform image, `ubi8/ubi` or `ubi7/ubi`, is designed to address the majority of use cases for container base images. It includes the runtime dependencies needed by about 80% of typical applications that run on Red Hat Enterprise Linux. In terms of size and pre-installed packages, this is the middle of the road image that is generally the best starting point.

The platform image includes:

- Basic OS tools like `tar`, and `gzip` that are typically needed inside of a container. A number of interactive commands like `vi` are available to make interactive work in a container easier. For those familiar with Red Hat Enterprise Linux packaging and kickstarts, the platform image starts with the `@base` package group. This improves compatibility for applications that are intended to run on Red Hat Enterprise Linux.
- All locales for internalization and localization.
- A unified cryptography stack based on OpenSSL to address the needs for encryption and certificates.
- The full YUM package management stack. The full stack has many capabilities beyond installing, removing, and updating packages.

For the UBI platform image, addressing a broad set of use cases is a higher priority than achieving the smallest container size. However, in many cases size won't be relevant. The broad applicability of the UBI platform image makes it a very common base that is likely to be shared by a large number of containerized applications. Most of the images from Red Hat, including the language runtimes, server images, and packaged applications, use the UBI platform image as their base. This commonality leads to significant reductions in disk space and network bandwidth utilization wherever multiple UBI-based containers are used.

The UBI minimal image

The minimal UBI image, `ubi8/ubi-minimal` or `ubi7/ubi-minimal`, is designed for applications that provide their own dependencies and have little need for the runtime components provided by the OS. The minimal image is approximately half the size of the platform image. The size reduction is accomplished by:

- Including a minimized set of pre-installed packages.
- Only including the `en` English locale for internationalization and localization.
- Not installing set-UID binaries, which improves the security of the minimal image. Therefore, `su`, `passwd`, `newgrp`, and `mount` are not available.
- Replacing the full YUM stack with microdnf, a minimal package manager written in C. Packages can still be installed, removed, or updated from YUM repositories. However, microdnf only implements a small subset of YUM's full functionality.

It might be tempting to build for a smaller image size by using the UBI minimal image and adding OS packages. However, depending on the number of OS packages added this might not produce the expected savings in disk space or network utilization. The resulting container image is more likely to have layers that are unique to that image and can't be shared. The UBI minimal image is best used when only a small number of OS packages need to be added.

A future UBI micro image

At the time this book was written, work was ongoing to answer customer requests for a base image that is even smaller than the UBI minimal image. The goal is for the micro image to be a fraction of the size of the UBI minimal image. This is intended for specific use cases like a single C or Go binary and other cases that use almost no dependencies from the OS.

To accomplish the micro size, the UBI micro will likely have the glibc runtime library, a shell, and not much else. There is no package manager included. So, the typical Dockerfile approach of installing packages with `yum` or `microdnf` at build time will not work. The files required by a package can be copied in from the host during builds. Buildah offers a key advantage in this case, as it allows you to run tools, like a package manager, on the host that modifies the container, without requiring that the tools be available inside the container.

While the UBI micro image doesn't contain a package manager, it does contain a snapshot of the RPM database from which it was built. The advantage is that vulnerability scanners, or anything else, can verify the version of code included in the base image to determine if it is free of known vulnerabilities.

The multi-service UBI image

Containers typically run only a single process. The process starts when the container is run. When the process exits, the container is stopped. The first process inside a container could run other processes. However, if the first process needs to be restarted, the whole container will exit and need to be restarted.

The common practice when building applications in containers is that each process runs in a separate container. For example, a web application that uses a database has a container for the web server and a container for the database. There are advantages to this in terms of the isolation and independence of each container. There is added complexity in that the set of containers needs to be running together and might need special networking configuration to connect the containers. It can be challenging to deliver small, multi container applications to others when you don't know the kind of container orchestration they are using. The target environment could be using a full-featured Kubernetes platform like Red Hat OpenShift, or they might still be managing containers on single machines using `docker-compose`.

The UBI multi-service image makes it easier to containerize applications that were originally created to run as multiple processes on a single machine. Processes in the multi-service image are managed by systemd, similar to how they would run on a system without containers. The processes can be stopped and restarted without restarting the container.

An application with a web server and a database could be built into a single multi-service container that is easier for consumers of that application to manage as a single container. This can be helpful if the target audience for the application has little or no experience working with containers.

While it is possible to treat a multi-service container as sort of a mini VM and package a whole system's worth of applications in a single container, this isn't necessarily the best approach. Many of the benefits of containers can be negated if the resulting image is an unwieldy multi gigabyte image. A large monolithic container image can be more complicated to update and secure. The UBI multi-service image is best for a relatively small number of processes that need to run together inside a single image.

Figure 14 shows a comparison of the UBI base image options:

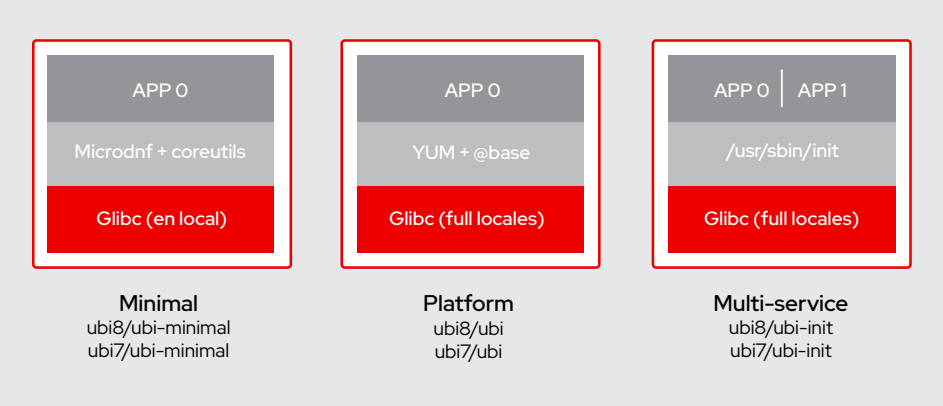


Figure 14. UBI base OS image options

The process that runs when a UBI multi-service container is started is `/usr/sbin/init`, which is part of systemd. This might cause an issue during development if you try to run `ubi-init`.

The UBI platform and minimal image both run `bash` as their default shell, so you can omit bash in a `docker run` or `podman` run and still get an interactive shell. Doing the same with `ubi-init` will appear to hang. Since the first process that runs inside the container is `init`, no prompt will ever appear. However, if you specify `bash` as the process to run, `init` will never be run. So systemd and any processes that systemd should start inside the container will not be started.

If an interactive shell is needed inside a multi-service container it is a two step process. First, run the image as you normally would. Then, use `docker exec` or `podman exec` to start a bash shell inside the container.

UBI pre-built runtime images

UBI includes pre-built container images with language runtimes, including Node.js, OpenJDK, Perl, PHP, Python, and Ruby, along with servers like Apache HTTPD and Nginx. These are built on top of the UBI platform OS base image and are ready for you to add your code. The advantage of using these images is that Red Hat performs the work to add the runtime components and maintains these base images.

The source of the packages used to build these images is application streams or software collections, which are updated more frequently than Red Hat Enterprise Linux. For UBI 8, these packages are from the UBI 8 Application Streams repository, which is a subset of the corresponding repository in Red Hat Enterprise Linux 8. For UBI 7, these packages are from the UBI 7 Software Collections repository, which is a subset Red Hat Enterprise Linux 7 Software Collections repository. See section 3.1, UBI life cycle and updates, for more information.

The list of available base images, along with a `Dockerfile` that shows how the image was built, can be found in the Red Hat Ecosystem Catalog:

- Red Hat Universal Base Image 8.
- Red Hat Universal Base Image 7.

Another advantage of using these runtime base images is the potential for savings in disk space and network utilization when multiple applications use the same base image. The largest portion of these images are the layers that make up the UBI platform image. As mentioned above, the UBI platform image is a common denominator for the majority of Red Hat images, so it is likely these layers will be shared. If multiple applications use the same runtime image, like OpenJDK, that layer will also be shared. If the same packages are added to an image during a build, it results in a unique layer that is not shared, removing any opportunity for reducing disk and network utilization.

6.6. Adding software to UBI images

When you need to add software to a UBI-based image, you can install packages from:

- The UBI package repositories.
- The Red Hat Enterprise Linux repositories, if you have a subscription.
- Any third-party software repositories that have RPMs compatible with the version of Red Hat Enterprise Linux that correspond to the version of UBI you are using.

Working with packages and repositories on UBI is essentially the same as on Red Hat Enterprise Linux or CentOS Stream. The `yum` command is used, unless you are using the `ubi-minimal` image, which uses `microdnf` instead.

The packages that are available to install depend on which repositories the container is configured to use. The UBI repositories are automatically enabled on all UBI base images. Red Hat Enterprise Linux repositories are enabled if the host system has a Red Hat subscription. depends on the host system. Third-party repositories need to be manually added when needed. This is usually accomplished by adding a `.repo` file to `/etc/yum.repos.d`.

You can see which repositories are enabled using the following command:

```
# yum repolist
```

UBI is compatible with RPMs built for Red Hat Enterprise Linux including packages from third-party repositories like the EPEL project.

UBI and Red Hat Enterprise Linux Repositories

For convenience, when a UBI container is run on a Red Hat Enterprise Linux host that has a Red Hat Subscription, the **Red Hat Enterprise Linux RPM repositories are automatically enabled in addition** to the UBI repositories. This allows you to easily add any of the full set of packages from Red Hat Enterprise Linux you are entitled to with your Red Hat subscription.

However, if you want to distribute UBI-based images outside of your organization it is important to only use the UBI subset of packages. Note that Red Hat partners can distribute images with Red Hat Enterprise Linux content. UBI licensing and redistribution (section 3.5) and Partnering with Red Hat (section 4.7) are covered earlier in this book.

When running UBI on a Red Hat system with a subscription, all enabled repositories are searched by default to satisfy `yum` commands. To limit commands to only the UBI repositories, additional arguments are necessary on `yum install` and `yum search` commands. The list of repositories to disable is specific to the version of UBI you are using. The specific arguments to use are covered below in the sections on adding packages to UBI 8 and UBI 7.

The list of UBI repositories can be found in *Universal Base Images (UBI): Images, repositories, packages, and source code* on the Red Hat Customer Portal.

Red Hat Enterprise Linux RPM repositories are automatically enabled when running UBI on a Red Hat system with a subscription. To make sure your UBI-bases images are redistributable, do not add Red Hat Enterprise Linux RPMs to your images, unless you are a Red Hat partner.

Quieting subscription management messages from YUM

On a Red Hat system, YUM normally interacts with Red Hat Subscription Management. The feature to add Red Hat Enterprise Linux content to UBI requires that YUM include the subscription manager plugin. Warning messages about subscription management from YUM can be reduced by disabling the subscription management plugin.

To disable the subscription management plugin, add the following arguments to `yum` commands:

```
--disableplugin=subscription-manager
```

Note that disabling the subscription management plugin alone is not sufficient to prevent using Red Hat Enterprise Linux repositories. If any `yum` commands are run without disabling the subscription manager plugin, the Red Hat Enterprise Linux repositories will be added to the system. Once that occurs, the subscription manager plugin is not needed to access the Red Hat Enterprise Linux repositories. Therefore, disabling the plugin only quiets warning messages, but doesn't prevent using non-UBI repositories.

Adding packages to UBI 8

The table below shows the UBI 8 repositories that are available by default on all UBI 8 images.

Table 8. UBI 8 package repositories

YUM Repository	Description
ubi-8-baseos	Red Hat Universal Base Image 8 Base OS – The RPMs that are part of the UBI 8 base operating system. This repository is a freely redistributable subset of the Red Hat Enterprise Linux 8 base repository. The update and support life cycle for these are the same as the Red Hat Enterprise Linux counterpart.
ubi-8-appstream	Red Hat Universal Base Image 8 Application Streams – Application streams are collections of RPMs that can be installed as a module. Multiple versions of an application stream can be available. The version to use is selected by specifying which module to install. Application streams have a different update and support life cycle than the base OS packages. Application streams are updated more frequently than the base OS. This repository is a freely redistributable subset of the Red Hat Enterprise Linux 8 Application Stream repository.

The set of default repositories for UBI 8 running on a host system *without* a Red Hat subscription can be listed with `yum repolist`:

```
# yum repolist
repo id         repo name
ubi-8-appstream Red Hat Universal Base Image 8 (RPMs) - AppStream
ubi-8-baseos    Red Hat Universal Base Image 8 (RPMs) - BaseOS
```

Running `yum repolist` in a UBI 8 container running on a system *with* a Red Hat subscription, shows the additional repositories that have been enabled:

```
# yum repolist
repo id                           repo name
rhel-8-for-x86_64-appstream-rpms Red Hat Enterprise Linux 8 for x86_64 - AppStream (RPMs)
rhel-8-for-x86_64-baseos-rpms    Red Hat Enterprise Linux 8 for x86_64 - BaseOS (RPMs)
ubi-8-appstream                   Red Hat Universal Base Image 8 (RPMs) - AppStream
ubi-8-baseos                      Red Hat Universal Base Image 8 (RPMs) - BaseOS
ubi-8-codeready-builder           Red Hat Universal Base Image 8 (RPMs) - CodeReady Builder
```

To limit YUM to only search the freely redistributable UBI 8 repositories, add the following arguments to `yum` commands:

```
--disablerepo='*' --enablerepo=ubi-8-baseos --enablerepo=ubi-8-appstream
```

Finding and installing application streams on UBI 8

Application streams contain packages for servers, language runtimes, and development tools. When searching for packages to install, start with application streams first. If you don't find what you need, then search for all RPMs.

To list the available application streams, run:

```
# yum module list
```

If you are on a Red Hat host with a subscription, use the following command to only search UBI repositories:

```
# yum module list --disablerepo='*' --enablerepo=ubi-8-appstream
```

To install a module like Python 3.8, run:

```
# yum module install python38
```

Some modules have multiple profiles that control which subsets of the modules will be installed. The default profile is marked with a `[d]` in `yum module list`. For example, the Python module has two profiles: *common* and *build*. The *build* subprofile contains additional packages needed for building Python dynamically loadable modules. To install a profile other than the default, add `/profilename` to the module name. To install the *Python 3.8* module using its *build* profile, run:

```
# yum module install python38/build
```

When using `yum` commands in a UBI container, you might see messages regarding *subscription management* and *consumer identity*. These messages pertain to Red Hat Subscription Management (RHSM) and can be safely ignored in UBI. The messages arise due to the option to use Red Hat Enterprise Linux content in UBI. When the host system running Red Hat Enterprise Linux is registered with RHSM, UBI containers on that host have access to the Red Hat Enterprise Linux repositories. No subscription or registration is needed for access to the UBI repositories.

These messages can be quieted using the argument `--disableplugin=subscription-manager` to `yum` commands.

Finding and installing RPMs on UBI 8

You can see the full list of RPM packages that are available with the following command:

```
# yum list available
```

The output is long, so you might want to use `grep` to search it. Or redirect the output to a file and use an editor like `vi` to search it.

Note that the third column lists the repository the package comes from. A package might be listed more than once if it is available from multiple repositories.

To limit the list of RPMs to only the UBI repositories use:

```
# yum list available --disablerepo='*' --enablerepo=ubi-8-baseos \
   --enablerepo=ubi-8-appstream
```

The `yum search` command is useful if you need to search the package descriptions in addition to the package names. To search for packages that mention C++ in their description run the following command:

```
# yum search C++
```

The output from `yum search` does not include the repository that a package comes from. However, the same arguments work to limit searches to specific repositories. To search for C++ in only the UBI repositories run:

```
# yum search --disablerepo='*' --enablerepo=ubi-8-baseos --enablerepo=ubi-8-appstream C++
```

Once you find the name of one or more packages you want to install, use `yum install`. For example, to install the GCC C compilers use:

```
# yum install gcc
```

To make sure packages are only installed from the UBI repositories, use:

```
# yum install gcc --disablerepo='*' --enablerepo=ubi-8-baseos --enablerepo=ubi-8-appstream
```

For more information, see Adding software to a running UBI container in the Red Hat Enterprise Linux 8 guide, Building, running, and managing containers.

Adding packages to UBI 7

The table below shows the UBI 7 repositories available by default on all UBI 7 images.

Table 9. List of UBI 7 repositories

YUM Repository	Description
ubi-7	Red Hat Universal Base Image 7 Server – The RPMs that are part of the UBI 7 base operating system. This repository is a freely redistributable subset of the Red Hat Enterprise Linux 7 server repository. The update and support life cycle for these are the same as the Red Hat Enterprise Linux counterpart.
ubi-7-server-extras-rpms ubi-7-server-optional-rpms	Red Hat Universal Base Image 7 Server Extra and Optional RPMs – A small number of additional RPMs that aren't part of the base OS collection because they have different support and update terms. These repositories contain a freely redistributable subset of the corresponding Red Hat Enterprise Linux 7 repositories.
ubi-server-rhscl-7-rpms	Red Hat Software Collections for Red Hat Universal Base Images – Red Hat Software Collections (RHSCL) are a collection of RPMs for updated collections of software that have an update and support life cycle that is different from the base OS. The UBI software collections are a freely redistributable subset of the software collections available for Red Hat Enterprise Linux 7. In Red Hat Enterprise Linux 8 and UBI 8, application streams replaced RHSCL.
ubi-7-rhah	Red Hat Universal Base Image Atomic Host – This repository contains `microdnf`, which comes from the Atomic Host variant of Red Hat Enterprise Linux. Generally there should be no need to interact with this repository when building UBI images.

The set of default repositories for UBI 7 running on a host system without a Red Hat subscription as seen with `yum repolist`:

```
# yum repolist
repo id                          repo name
ubi-7                       Red Hat Universal Base Image 7 Server
ubi-7-rhah                  Red Hat Universal Base Image Atomic Host
ubi-7-server-extras-rpms    Red Hat Universal Base Image 7 Server - Extras
ubi-7-server-optional-rpms  Red Hat Universal Base Image 7 Server - Optional
ubi-server-rhscl-7-rpms     Red Hat Software Collections RPMs for Red Hat Universal Base Images
```

For the same UBI 7 container running on a system with a Red Hat subscription, the additional repositories that are automatically enabled can be seen in the `yum repolist` output:

```
# yum repolist
repo id                          repo name
rhel-7-server-rpms          Red Hat Enterprise Linux 7 Server (RPMs)
ubi-7                       Red Hat Universal Base Image 7 Server
ubi-7-rhah                  Red Hat Universal Base Image Atomic Host
ubi-7-server-extras-rpms    Red Hat Universal Base Image 7 Server - Extras
ubi-7-server-optional-rpms  Red Hat Universal Base Image 7 Server - Optional
ubi-server-rhscl-7-rpms     Red Hat Software Collections RPMs for Red Hat Universal Base Images
```

Finding and installing RPMs on UBI 7

You can see the full list of RPM packages that are available with the following command:

```
# yum list available
```

The output is long, so you might want to use `grep` to search it. Or redirect the output to a file and use an editor like `vi` to search it.

Note that the third column lists the repository the package comes from. A package might be listed more than once if it is available from multiple repositories.

The `yum search` command is useful if you need to search the package descriptions in addition to the package names. To search for packages that mention C++ in their description run the following command:

```
# yum search C++
```

The output from `yum search` does not include the repository that a package comes from. However, `--disablerepo='*'` and `--enablerepo=<reponame>` can be used to limit searches to specific repositories. To search for Python in only the UBI 7 RHSCL repository run:

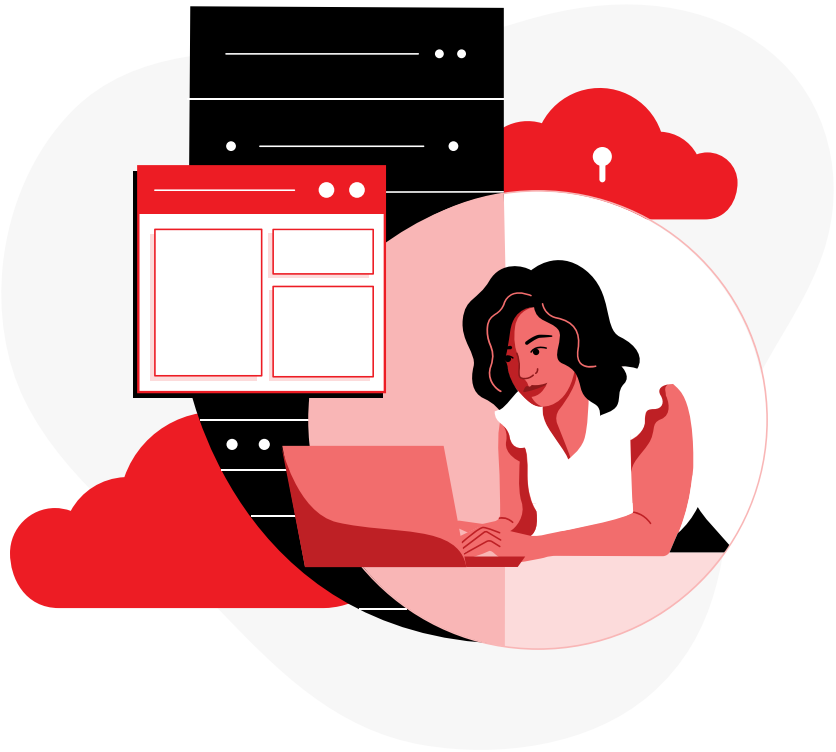
```
# yum search --disablerepo='*' --enablerepo=ubi-server-rhscl-7-rpms python
```

Once you find the name of one or more packages you want to install, use `yum install`. For example, to install Python 3.8 use:

```
# yum install rh-python38
```

Note: The Python 3.8 package, like many other packages that were not available when Red Hat Enterprise Linux 7 was released in 2014, is available as a Red Hat Software Collection.

For more information, see **Managing containers in the Red Hat Enterprise Linux 7 documentation**.



For more information

Product information for UBI

- Red Hat Universal Base Image 8 – UBI 8 information in the Red Hat Ecosystem Catalog.
- Red Hat Universal Base Image 7 – UBI 7 information in the Red Hat Ecosystem Catalog.
- Red Hat Universal Base Images end-user license agreement (Red Hat UBI EULA).
- Red Hat Universal Base Images (UBI): images, repositories, packages, and source code.
- Red Hat Universal Base Image frequently asked questions (FAQ).
- Getting Red Hat Universal Base Image source code.

Red Hat support information for UBI

- Red Hat Enterprise Linux life cycle.
- Red Hat Universal Base Image – content availability.
- Red Hat container image updates.
- Red Hat container support policy.
- Red Hat Enterprise Linux container compatibility matrix.
- Red Hat container image and host guide: application portability.
- Container Health Index grades as used inside the Red Hat Ecosystem Catalog.

Red Hat documentation for UBI

- UBI 8 and Red Hat Enterprise Linux 8:
  - Building, running, and managing containers – Covers UBI and container tools like Podman, Buildah, and Skopeo.
  - Installing, managing, and removing user-space components – Introduction to application streams, modules, and profiles that are used when working with UBI 8 packages.
  - Red Hat Enterprise Linux release notes – Changes to UBI and container tools are documented in the release notes with each release of Red Hat Enterprise Linux.

- UBI 7 and Red Hat Enterprise Linux 7:
  - Managing containers – Covers container tools like Podman, Buildah, and Skopeo, managing containers with systemd, and container signing.
  - Red Hat Software Collections – Software collections are used for UBI 7 packages that have a different life cycle than Red Hat Enterprise Linux 7.
  - Red Hat Enterprise Linux release notes – Changes to UBI and container tools are documented in the release notes with each release of Red Hat Enterprise Linux.

Getting the latest information on UBI

- Follow the Red Hat blog and the Red Hat Developer blog.
- Join the Red Hat Developer Program.
- Become a Red Hat partner through Red Hat Partner Connect.
- Join DevNation to take part in live and virtual events for developers led by Red Hat technology experts.

Get started with Red Hat Universal Base Images

0421_KVM

Red Hat