



ActiveState's Container Hardening Guide

Keeping ActiveState
Secure Containers
Vulnerability Free





In 2025, containers have become a foundational technology in modern software development and deployment, driven by the need for portability, scalability, and rapid deployment. According to industry reports, [over 90% of organizations now use or are evaluating containers in production](#). Container Orchestration tools like Kubernetes have gained significant traction and downloads across image registries like DockerHub have reached billions of pulls. While container usage has exploded, the status quo of container development remains inherently risky for several reasons:

- ! Base Images**
Engineering teams commonly build containers using base images pulled from public registries. If a base image is outdated or contains unpatched vulnerabilities, all derived containers are now at risk.
- ! Dependencies and Libraries**
Modern applications often comprise dozens or even hundreds of dependencies. Vulnerabilities in any library may provide a backdoor for attackers.
- ! Dockerfiles and Build Processes**
Misconfigurations in Dockerfiles or scripts can unintentionally introduce vulnerabilities. Even more, Dockerfiles don't provide immutability or reproducibility without a lot of care and effort, you don't get either one!
- ! Runtime Environments**
Containers running with excessive privileges, poorly configured network rules, or shared namespaces can lead to privilege escalation or data exfiltration.

For many of the teams we speak with, these risks are becoming increasingly common. Scanners in CI/CD pipelines are lighting up with CVEs, containers suffer from a growing attack surface, and with sprawling dependencies it has become challenging to verify that every component is what it claims to be. Fortunately, several best practices have emerged to combat these issues.

This guide outlines the process and best practices ActiveState follows when building, maintaining, and publishing container images.

[Overview: ActiveState Secure Containers](#)

ActiveState Secure Containers – Core Design Principles

ActiveState Secure Containers – Assembly Process

[Securing the Base: A Distroless Foundation](#)

The Benefits of Building a Custom Base Image

The ActiveState Base Image

[Building From Source](#)

The Benefits of Building From Source

Building From Source is Hard

[Minimization, Configuration and, Image Assembly](#)

Further Minimization and Configuration

Metadata

[Image Testing & Distribution](#)

Functionality Testing

Minimization Benchmarking

Hardening Benchmarking

Vulnerability Assessment and Advisories

Registry Testing

Testing Failure

[Image Distribution and Maintenance](#)

Publication

Image Maintenance – Staying Vulnerability Free

[Container Security Is No Joke](#)

[Get Started with ActiveState Secure Containers](#)



Overview: ActiveState Secure Containers

ActiveState Secure Containers are our minimal, low-to-no CVE container images for popular programming languages (deployment and development use cases), frameworks, and applications. We provide both a standard library of free container images as well as custom images tailored to unique customer needs.



Free Container Images:

Our standard container image library ([Container library](#)), made up of the latest version for a variety of images including: programming languages for deployment, developer variants, and useful applications such as stunnel.



Custom Container Images:

Our fully managed container build service, for teams who require custom configurations, additional packages, application components, or further hardening. Using our catalog of over 40 million secure open-source components, organizations can utilize fully customized and continually remediated container images built to their exact specifications.

Every container image (free and custom) is rebuilt nightly and bound by our industry leading remediation SLAs of 7 days for both critical and high CVEs and 14 days for all others. Several design choices make this possible.

ActiveState Secure Containers

Low-To-No CVE Container Images

Minimal hardened images for popular languages and apps built from source.

Delivered to CI/CD Pipelines

Custom images delivered via private registry for direct integration with CI/CD pipelines.



End-to-end Customization

Fully customizable using over 40M secure application components.

Managed and Remediated

Custom images are rebuilt regularly, maintained and patched with leading SLAs.

Figure 1.0 – ActiveState Secure Containers Overview

ActiveState Secure Containers – Core Design Principles

ActiveState Secure Containers operate on two interrelated principles: minimization and hardening. While these terms often lack clear definitions and are sometimes used interchangeably, we define them as follows:

✓ **Minimization:**

Means removing everything unnecessary. If it's not there, it can't be exploited—fewer libraries, fewer tools, and fewer potential attack vectors. As a bonus, this also reduces the size of the container, which can be beneficial for a variety of reasons including performance, cost, and license/policy adherence.

✓ **Hardening:**

Where minimization focuses on what can be removed, hardening focuses on locking down what remains. Secure defaults, restricted permissions, and careful configuration make it harder for attackers to gain a foothold. That said, you do need to be aware of the impact these changes may have on the languages, packages, and software running on your container.

While the origins of these practices trace back to deployment methods in a time before containers, the underlying philosophies are just as relevant today. Together, these pillars create “defense in depth.” Instead of only reacting to CVEs once discovered, we leverage both techniques across the container assembly process to proactively reduce the chance of vulnerabilities existing in the first place.

ActiveState Secure Containers – Assembly Process

ActiveState Secure Containers are assembled and maintained according to the following steps:

✓ **Custom, Secure Base Layer:**

ActiveState Secure Containers are all built using our custom base layer as the foundation. This custom base follows a distro-less design philosophy to ensure maximum security and a minimal footprint. It contains only the essential dependencies, deliberately excluding standard operating system components like package managers, shells, and debugging tools.

✓ **Building From Source:**

To build and include the necessary components for our images across both the base and application layer, we leverage our build system – The ActiveState Platform.* This platform provides an SLSA-3 compliant build service enabling us to securely and repeatedly build components from source

- ✓ **Minimization, Configuration, and Assembly:**
Container images are assembled according to the Open Container Initiative (OCI) [image spec](#), configured using minimization and hardening best practices, and linked to their associated metadata and SBOM.
- ✓ **Image Testing:**
Assembled container images are run through our automated testing pipeline, assessing the image across a variety of metrics to validate: vulnerability count via SCA scanning, hardening score, signature, and metadata. For CVEs that have no impact or no available fix, we issue a VEX advisory.
- ✓ **Image Distribution & Maintenance:**
Following a successful image test, images are published to our internal registry for distribution to both private registries for customers' custom builds and publicly to our container image catalog. All ActiveState images and their respective components are bound by a remediation SLA of 7 days for critical CVEs and 14 days for all others.

**Currently only the components layered on top of our base image are compiled using the ActiveState Platform*

The following chapters provide additional details for each of the above stages.

Securing the Base: A Distroless Foundation

For most teams, building a container image often begins with searching through DockerHub to find a pre-existing image suitable for the base of their container. While public repositories make it quick and easy for teams to get started, these community-maintained images (typically Alpine, Debian, or Ubuntu-based) are often bloated, outdated, and may contain unpatched vulnerabilities and misconfigurations.

To achieve complete control over our base image, ActiveState has developed a **custom distroless base image** that acts as the foundation for all other ActiveState images. Instead of packaging a complete Linux-based distribution with its supporting software utilities and libraries, we strip out the vast majority of bundled software, allowing us to start with only what is required to support the functionality of our container images. No shells, no package managers, and no debugging tools (for developers, we provide a [“dev” variant](#) of our base image).

Even more, every component is compiled directly from source allowing us complete control over the included components. [More on building from source later!](#)



Key Definition: Distroless Image – a minimal container image that does not include a full Linux distribution (like Ubuntu, Debian, or Alpine). Instead, it contains only the application and its runtime dependencies.

The Benefits of Building a Custom Base Image

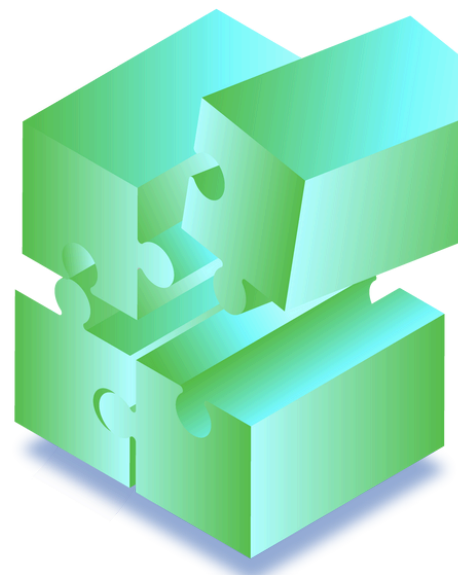
By developing our base-image to act as the base layer, ActiveState Secure Containers inherit multiple benefits, including:

-  **Reduced Attack Surface**
By stripping away unnecessary packages, shells, and utilities, our images minimize the number of potential vulnerabilities attackers can exploit.
-  **Improved Security Controls and Compliance**
We gain complete visibility and control over what components are included in our images, ensuring that only vetted components (compiled by ActiveState) are packaged.
-  **Smaller Image Size**
Distroless images are typically much lighter, which in turn means faster pulls and leaner deployments.
-  **Easier Maintenance**
For both our free image catalog and custom containers, this custom base allows us to simplify the maintenance of our images drastically. With fewer moving parts, patching becomes faster and easier.

The ActiveState Base Image

With this in mind, you might wonder what is actually included within the base image? All of our published images include only a Glibc as their base, built with the bare minimum requirements for functionality and appropriate configurations. In addition to the stripped-back version of the base image, we also publish an alternate version for development use cases based on a minimized version of busy-box. At the cost of increased attack surface, [this variant](#) allows users to have a shell and do some basic walking around the system.

All of this is made possible thanks to our ability to compile everything from source.



Building From Source

While the base is a good starting point, even the most secure base image can be undermined by what you install on top. In nearly every containerized application, there's a second and often more insidious layer of risk: the required application dependencies. To add functionality to their software, developers frequently rely on public package managers like pypi, npm, and Maven.

These tools are powerful and convenient, but also risky. See the [recent npm supply chain attack](#) for proof!

ActiveState Secure Containers take no such chances. In addition to providing a secure base image, we build all open-source software from source. Doing so allows us to layer languages, packages, and entire applications on top of our base image while maintaining control over their files and configurations.



The Benefits of Building From Source

Building from source affords our images a multitude of benefits and allows us to carry forward a focus on minimization and hardening beyond the underlying container OS. In particular, building from source provides our images with four critical advantages:

Supply Chain Trust

When you pull a container image from a registry, one of the downsides is that you inherit everything that comes inside it. When you rely on pre-built binaries, you're trusting that every single dependency is safe. Instead, building from source allows us to provide and attest only verified code, ensuring nothing malicious finds its way into your images.

Shrink and Optimize Your Image

Similar to the minimization and hardening accomplished at the base layer, compiling from source allows us full control over every dependency in a container. By stripping back to only what's essential, we create less attack surface (and less ongoing CVE remediation to manage). Of course, all ActiveState Secure Containers can be fully customized to include any required package, application library, or configuration.

Audit Readiness

For organizations building software for regulated industries, tracking the components that make up your software, and being able to reproduce it on an ongoing basis is a key requirement. Building from source allows us to generate the necessary metadata including detailed SBOMs at build time.

Accelerate Patches

When a vulnerability is discovered in your container, the typical response is to wait for an upstream binary update, a process that can sometimes mean days of exposure. With a source-based build process, we can patch, recompile, and rebuild your container image immediately, long before the next official release makes its way to a public repository.

Building From Source is Hard

One of the main reasons preventing most organizations from building from source is, simply put, the complexity in doing so. Not only do you need to deal with the usual challenges of compiling software, but you must do so with the unique constraints of image building. To build from source organizations must invest significant time and effort to develop the necessary expertise, required tooling, and build systems. None of which are typically deemed as value-add engineering work.

The ActiveState Platform – Simplifying Building From Source

To solve this, ActiveState offloads the process, by combining our SLSA-3 compliant build engine, decades of experience in good build knowledge, and our [growing catalog](#) of over 40 million secure open-source components.

Amongst other things, the ActiveState platform allows us to automatically build open source from source code, store it in our catalog, and eventually package it in the form of a container image. Using the Platform eliminates the need for time-consuming tasks such as cloning source repositories, compiling, and resolving dependency conflicts.

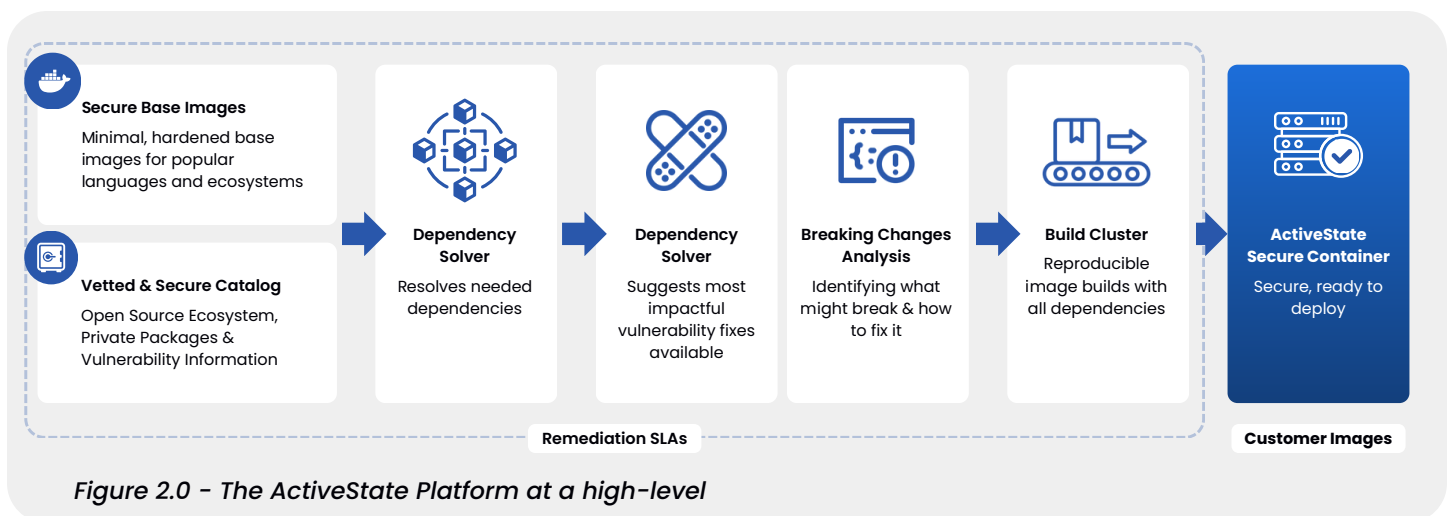


Figure 2.0 – The ActiveState Platform at a high-level

With both a secure base-image and catalog of open-source components compiled from source, the next step is image construction.

Minimization, Configuration and, Image Assembly

With open-source in hand, we also leverage our build infrastructure to assist with the construction of the final container image. Our container build pipeline starts by combining our distroless base image ([described above](#)) with our catalog of secure open-source components. For example, if we are building a Python image for deploying an app, we can include the Python runtime as well as any of the popular libraries that many modern applications rely on.

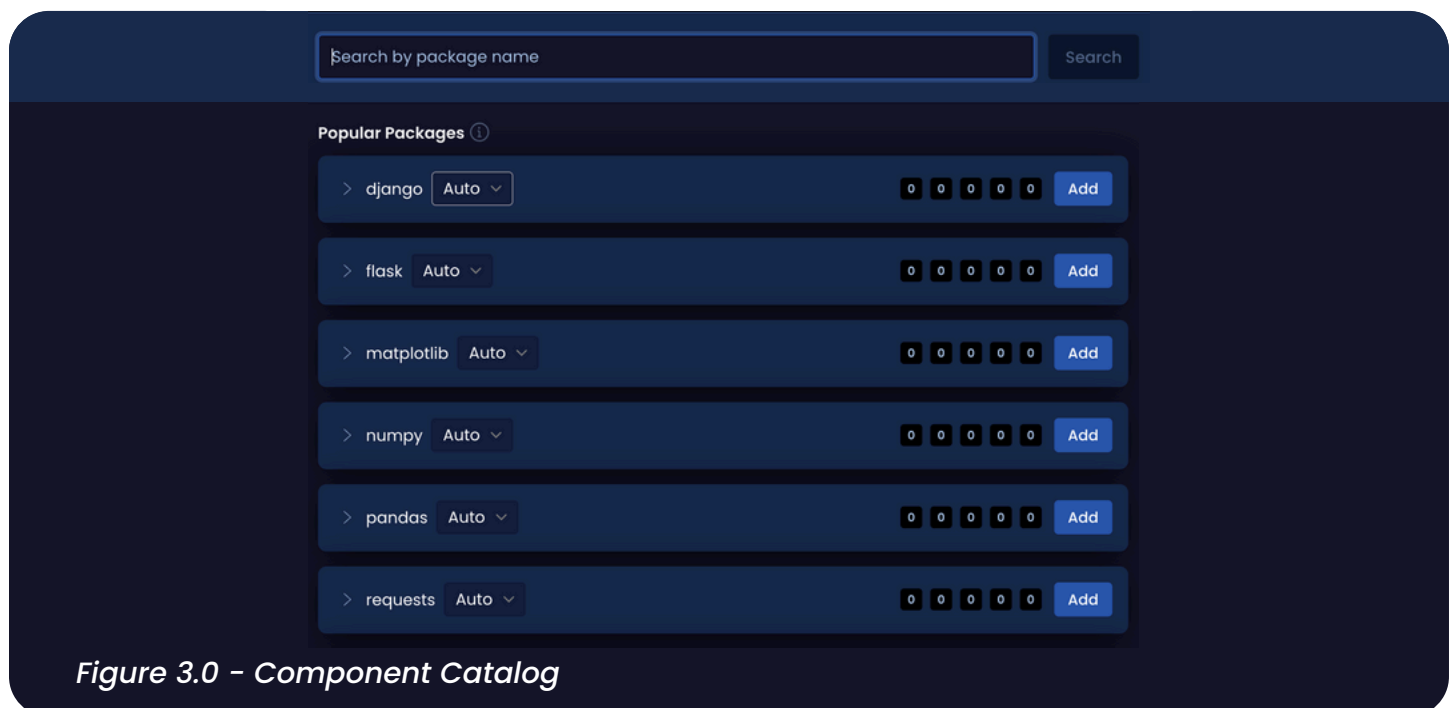


Figure 3.0 – Component Catalog

Further Minimization and Configuration

However, our minimization work isn't limited to which components are included in our images but also to what's in the components themselves. We apply further minimization and hardening wherever possible to ensure the absolute smallest (and most secure) images. These modification and configurations include the removal of:

- Files and directories that aren't needed in the final image
- Documentation
- Specific Libraries
- Debug symbols from the binary files
- And more!

We carefully test each modification to ensure that any minimization or configuration does not impact image functionality. We then take the altered project and shift it back on top of our base image. The end result is a smaller, cleaner, and more secure image. This not only improves security by reducing unnecessary components, but also makes containers faster to pull, easier to distribute, and simpler to maintain.

Metadata

For organizations that need to understand (and prove) not just what's inside their containers, but also that those containers can be trusted, Software Bills of Materials (SBOMs) and signed images are critical. Every ActiveState Secure Container is published with a detailed buildtime SBOM that provides full visibility into its components, making audits and compliance straightforward. Paired with cryptographic signing, these images give teams confidence that what they pull is exactly what we built — untampered, verifiable, and ready for deployment in even the most regulated environments.

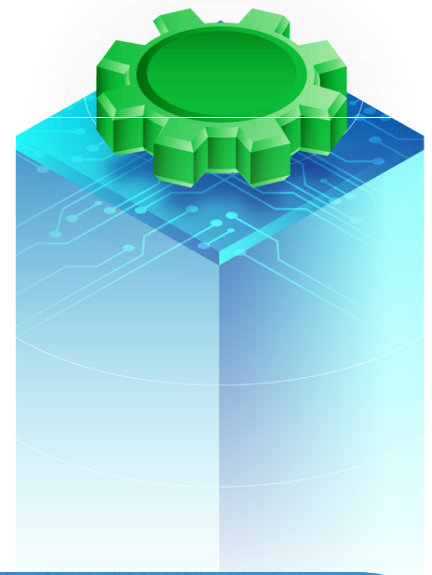


Image Testing & Distribution

Once an ActiveState image has been assembled, it's not enough for us to assume it has been appropriately slimmed and secured. We run a variety of checks across key metrics to ensure that every image produced is both functional and safe before distribution. Testing is handled via an automated pipeline that can be broken down into the following stages:



Functionality Testing

For images focused on programming languages, a simple “hello world” test is run to ensure a basic program can be deployed and run successfully on top of our base images.

Minimization Benchmarking

Every image is benchmarked for total image size against a variety of comparable images available in DockerHub. To successfully pass we aim to achieve a total image size within 10-20% of the nearest comparison.

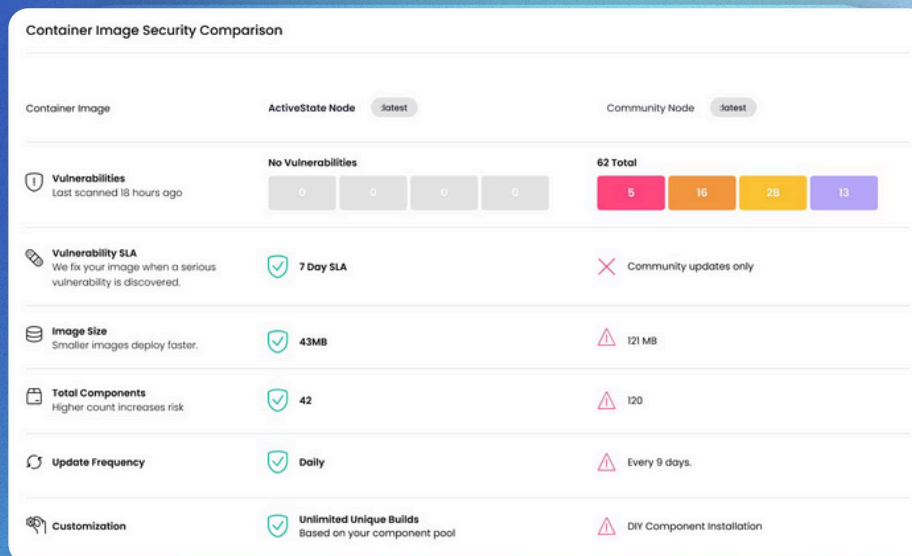


Figure 4.0 - ActiveState Image Comparison

Hardening Benchmarking

In addition to benchmarking for image size, all images are also tested against hardening best practices including

- **Minimalism** - Reducing software packages and complexity
- **Provenance** - Verifying source and authenticity
- **Configuration & Metadata** - Applying secure defaults and best practices
- **Vulnerabilities** - Tracking known vulnerabilities at any point in time

As part of our pipeline, all images pass through a variety of tools, including [crane](#) (size check), [cosign](#) (signature verification), [grype](#) (CVE scanning), and [trufflehog](#) (for secret scanning).

Vulnerability Assessment and Advisories

To further assess vulnerabilities, all images are once again scanned using a variety of SCA tools, including Docker Scout, Trivy, and Grype. To pass, all images must produce zero effective CVEs.

In the case where a CVE is detected, we include a human in the loop to triage and assess the best plan of action. A member of the ActiveState engineering team will investigate each CVE to determine its impact, generally resulting in one of three assessments.

⚙️ Positive Finding (Fixable)

If the CVE is present in the image, and an upstream fix that remediates the CVE is available, we will apply the update, rebuild the container image, and run it through the testing pipeline to confirm its success. In some cases, ActiveState may bump a vulnerable dependency to a patched version before the upstream projects have done so, or apply a patch issued outside of the upstream repository.

⚙️ Positive Finding (No Fix Available)

It is also possible for the identified CVE to have no current fix. If so, we issue an “affected” advisory in the form of a Vulnerability Exploitability eXchange (VEX) document. Images with associated VEX documents reference the file in their metadata, enabling end-users to pass advisories to their SCA tool of choice.

```
"status": "affected",  
  "action_statement": "No upstream fix yet for busybox-full tar  
escape-sequence issue.",  
  "action_statement_timestamp": "2025-06-20T15:47:11.697801-03:00"
```

Figure 5.0 Rust Advisory: [CVE-2025-46394](#)

⚙️ False Positive

If the CVE is determined to have no impact on the image, we issue a “not affected” advisory. The specific reason for each assessment is documented in its associated VEX file. In either case, all ActiveState advisories are published to our [GitHub page](#).

```
"status": "not_affected",  
  "justification": "vulnerable_code_not_present",  
  "impact_statement": "This version of gcc contains the patch."
```

Figure 5.1: Rust Advisory: [CVE-2023-4039 SCA](#)

SCA Tool Integration

To simplify the integration of our VEX advisories into customer scanning processes, we are currently working with several of the leading SCA providers to integrate our advisory feed directly into their products.

Registry Testing

After any outstanding CVEs have been resolved, images are assessed for the contents of their metadata. This includes checking to ensure the image signature can be verified and that an associated SBOM is available and accurate.

Testing Failure

If at any point the testing pipeline detects that an image fails to meet its required standards, the image is sent back for reconstruction, further minimization, or hardening as needed. Once an image has successfully made its way through the entire testing pipeline, it is ready for publication and further distribution.

Image Distribution and Maintenance



Publication

After an image has passed through our testing pipeline, it is then staged to our internal private registry for further distribution to both public and private repositories.

✓ DockerHub

ActiveState publishes the latest version of our container images to our [DockerHub](#) page. Docker currently lists ActiveState as a “[Verified Publisher](#)”. Participants in this program provide high-quality images to help development teams build secure software supply chains, minimizing exposure to malicious content early in the process to save time and money later.

✓ Private Registries

For custom images (including older variants), we push images directly to private registries for our customers. This allows for streamlined CI/CD integration and makes it easy for our customers to remain vulnerability-free.

Future Publishing Plans

In the near future, we plan to provide a browsable catalog of all ActiveState Secure Container images and available components for customization directly on our own platform.

Image Maintenance – Staying Vulnerability Free

Of course, our job isn’t complete after an image has been published or pushed to a customer’s environment. Vulnerabilities emerge every day. To stay ahead, we adhere to strict remediation SLAs of 7 days for critical CVEs and 14 days for all others.

When our automation identifies a vulnerability in either our public catalog or a custom image, ActiveState takes on the entire remediation process. Our automated build system springs into action, rebuilding the affected open-source components from source using a remediated version pulled from our catalog.

ActiveState’s catalog updates to include every currently supported version of software maintained by the project’s upstream. In some cases, we may even backport a patch to support custom requirements for our customers.



All images are rebuilt nightly, tested using our pipeline described above, and delivered back to our catalog and customers' private repositories with SBOMs, attestations, and signatures to ensure complete transparency and traceability. This comprehensive approach means there's no manual patching for your team, no complex dependency guesswork, and the process fully offloads the overhead of managing custom images.

Container security is not a one-time task. It's an ongoing, iterative process. Attackers are constantly looking for new ways in, and new vulnerabilities are disclosed daily. Teams that rely on public images, precompiled binaries, and bloated containers face a constant uphill battle against CVEs.

By contrast, we flip the script by:

- Starting with a clean, minimal, distroless base.
- Building everything from source.
- Ensuring assembled images are appropriately minimized and hardened.
- Maintaining all images and their components with strict SLAs.

Containers that are as close to vulnerability-free as possible save developers time, reduce security noise, and strengthen software supply chains.

Container Security Is No Joke



If you've made it this far, one thing should be clear – container security is no joke! Vulnerabilities emerge every day, meaning that even well-hardened images require continuous maintenance and rapid remediation. Without automation and dedicated processes, the overhead of keeping containers secure quickly overwhelms internal teams. This constant cycle of discovery and patching leaves many organizations struggling to keep up.

Although some organizations attempt to take control by building their own images, success does not come without significant hurdles. Developing a secure, repeatable build process requires specialized expertise, tooling, and ongoing investment. For many teams, this work is considered undifferentiated engineering work, competing against the development of new features. This makes it difficult to prioritize and \ budget for. As a result, organizations often settle for partially secured images, accepting risk because the alternative seems too resource-intensive.

This is why more and more teams are looking for a partner in container security. By starting with a clean, distroless base and rebuilding every component on top from source, ActiveState helps organizations move from constantly reacting to vulnerabilities to proactively reducing their attack surface from the very start.



Get Started with ActiveState Secure Containers

To get started, the [latest versions](#) of all ActiveState Secure Containers are available at no charge and ready to integrate into your existing CI/CD workflows.

For teams who require custom configurations, additional packages, or further hardening to achieve compliance standards, ActiveState's custom container service is here to help. No matter what your container specifications are, our team will ensure that every component, from the OS to your application dependencies, is fully secured and ready for deployment.

[Contact us today](#), and try your first custom container image free!

