# Innovate Fearlessly: Secure, Fast Development with Docker & Google Cloud

# Table of Contents

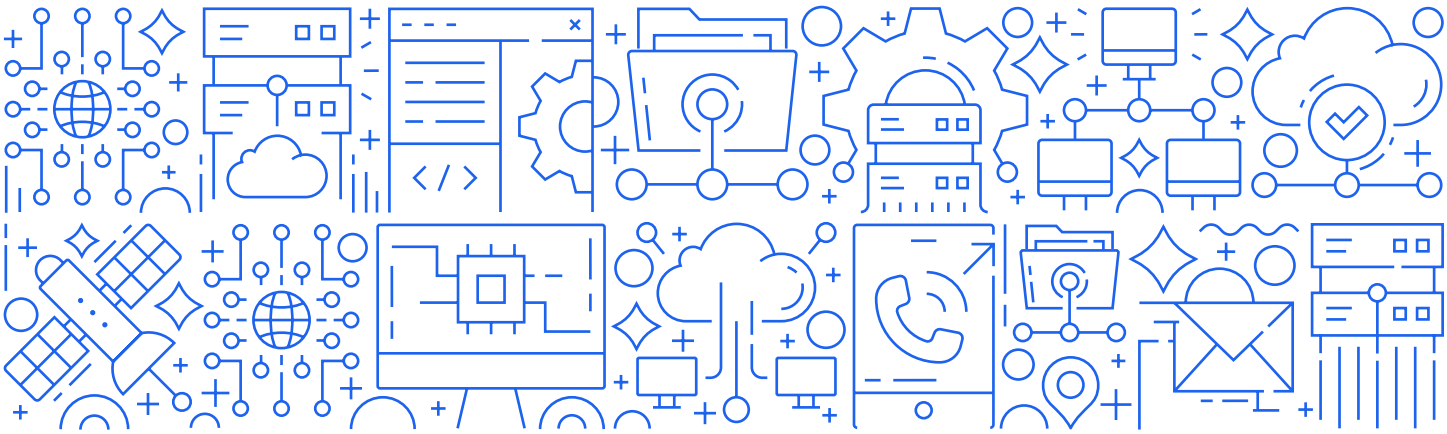
# Introduction

It's 2025 – generative AI features and microservices are rolling out weekly, yet cyber threats loom larger each day. Even experts struggle to catch up[1]. In that scenario, **CISOs and CTOs** must enable their teams to innovate fearlessly without fear of security breaches.

This means balancing breakneck **developer agility** with uncompromising **security compliance**. The challenge is palpable: how do you empower developers to rapidly build and deploy in the cloud, while ensuring every container, AI model, and service adheres to corporate and regulatory security standards? High-level decision-makers are seeking a roadmap to achieve both speed and security – to **reliably** deliver on digital transformation goals, neither slowing down nor taking unnecessary risks in the organizations' software supply chain.

Integrating security from the start of the development lifecycle is the key to unlocking "fast and secure" innovation. We start by examining rising threats in software supply chains and AI-generated code. Then, we dive into how **Docker's holistic solution** embeds security into developers' daily workflows with minimal friction. Finally, we'll see how Docker and **Google Cloud** together provide an end-to-end, secure ecosystem from development to deployment, illustrated by real-world success stories.

**The message is clear: with the right approach, you can move fast and stay secure, fueling fearless innovation.**



[1] Ethan Mollick via X (formerly Twitter): https://x.com/emollick/status/1891913605890609624

# Threats in the software supply chain

**The software supply chain threat landscape has escalated dramatically in recent years, especially in the realm of software supply chains.**

**Supply chain attacks impacted 64% of companies, largely due to increased reliance on open-source software**

**Open-source reliance is at an all-time high – modern cloud-native applications often consist of 70-90% open-source components**

High-profile breaches have shown that attackers are no longer storming the front gate – they're sneaking in through components and dependencies deep inside our software. Recent reports show a startling rise in software supply chain attacks: one study found a **28% year-over-year increase** in malicious packages uploaded to open-source repositories[2]. In fact, supply chain attacks impacted **64% of companies**, largely due to increased reliance on open-source software. This is a wake-up call for every CISO and engineering leader. When you're indiscriminately pulling hundreds of open-source containers and libraries, each one could be a potential Trojan Horse.

Several factors are amplifying these risks. **Open-source reliance** is at an all-time high – modern cloud-native applications often consist of 70-90% open-source components[3]. This accelerates innovation but also means inheriting unknown vulnerabilities or malicious code. Attackers have noticed: from typosquatting on container images to injecting backdoors in popular libraries, they exploit the very openness that developers depend on. Meanwhile, the advent of **AI-generated code** introduces new challenges. Developers eager to leverage AI coding assistants may unintentionally introduce insecure code[4]. More than half of organizations have encountered security issues from AI-generated code, yet over three-quarters of developers admit to bypassing security protocols to use AI code completion tools[5]. This "shadow AI" effect means that without proper guardrails, helpful AI suggestions could slip dangerous vulnerabilities or license compliance issues into your codebase.

2 https://www.reversinglabs.com/blog/the-state-of-software-supply-chain-security-2024-key-takeaways
3 https://www.nber.org/be/20241/open-source-software-creators-its-not-just-about-money
4 https://www.imperva.com/blog/cursors-magic-comes-with-a-catch-missing-trust-setting/
5 https://www.ciodive.com/news/security-issues-ai-generated-code-snyk/705900/

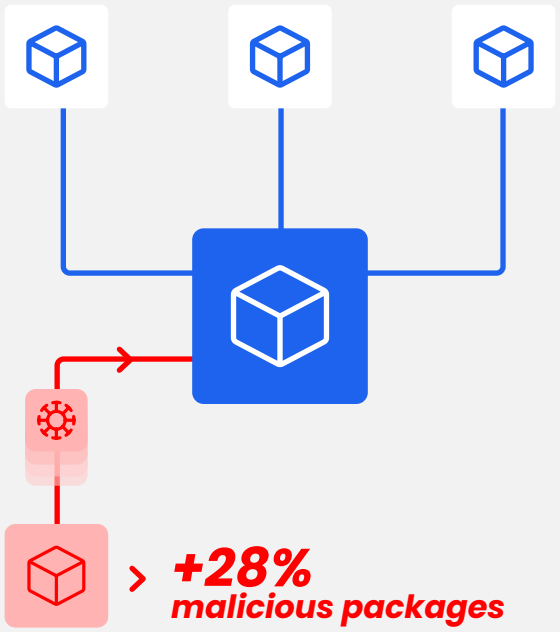
# The push for software supply chain regulation

On top of these trends, **regulatory pressures** are mounting. Governments worldwide are responding to the threat landscape with new rules that demand better software security. The U.S. White House, for instance, issued cybersecurity executive order number 14028 and its subsequent order on strengthening and promoting innovation in the nation's cybersecurity, emphasizing that software providers must implement secure development practices to reduce vulnerabilities.

In the EU, the upcoming **Cyber Resilience Act (CRA) and the NIS-2 directive**, which will introduce strict cybersecurity requirements for any "product with digital elements" (which includes software) sold in the EU.

The tendency is that more regulations will be enacted by governments, especially in a geopolitical context where digital infrastructure and services are the targets of cyber attacks promoted by nation-state agents.

**Adversarial countries and criminals continue to conduct cyber campaigns targeting the United States and Americans (...) presenting the most active and persistent cyber threat to the United States Government, private sector, and critical infrastructure networks.**[6]




This means organizations will soon need to demonstrate secure software supply chains and may be required to provide SBOMs (Software Bills of Materials) and vulnerability disclosure processes. Such requirements already exist in the context of data protection laws[7].

6 https://www.federalregister.gov/documents/2025/01/17/2025-01470/strengthening-and-promoting-innovation-in-the-nations-cybersecurity
7 E.g., Art. 33 GDPR, Regulation (EU) 2016/679, available at: https://gdpr-info.eu/art-33-gdpr/.

**The message from regulators is clear: *secure your development process, or face consequences.***

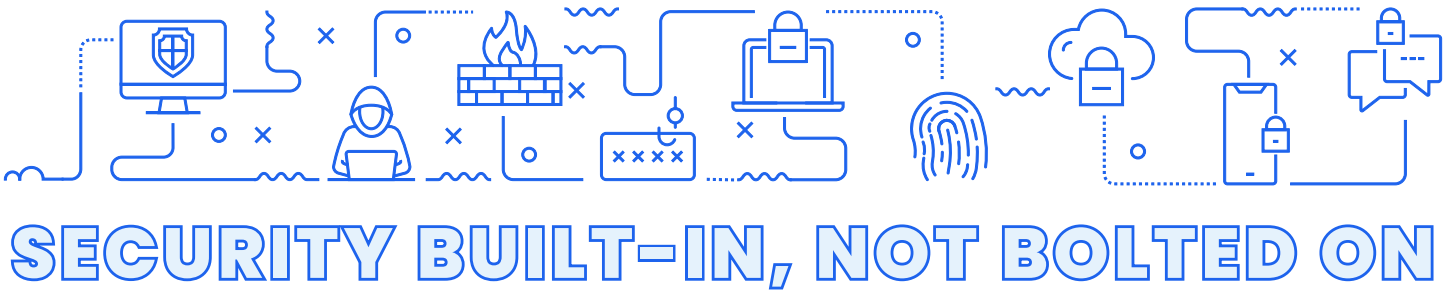
**Technical decision-makers** thus find themselves at a crossroads. The threat landscape demands action – simply hoping that developers remember to patch or that open-source maintainers will "do the right thing" is not a strategy.

We need systematic approaches that weave security into the DNA of software creation. In the next sections, we'll explore how to respond: by **embedding security throughout the development lifecycle in a way that doesn't slow down innovation**. It's about shiftingfrom reactive, last-minute security checks to proactive, continuous security – so that compliance and protection become natural by-products of a fast Dev(Sec)Ops workflow.

# Embedding Security in the Development Lifecycle

**Organizations are embracing a simple but powerful principle: embed security into every phase of development.**

To counter the rising threats without sacrificing agility, organizations are embracing a simple but powerful principle: **embed security into every phase of development**. Rather than treating security as a final check or an external gate, it becomes a built-in feature of the development lifecycle – from the moment a developer writes code to the moment that code is deployed to production. Docker's platform is designed around this philosophy, integrating robust security measures in a way that does not add friction for developers. **Security built-in, not bolted on**.



# The importance of Trusted Content

docker hub

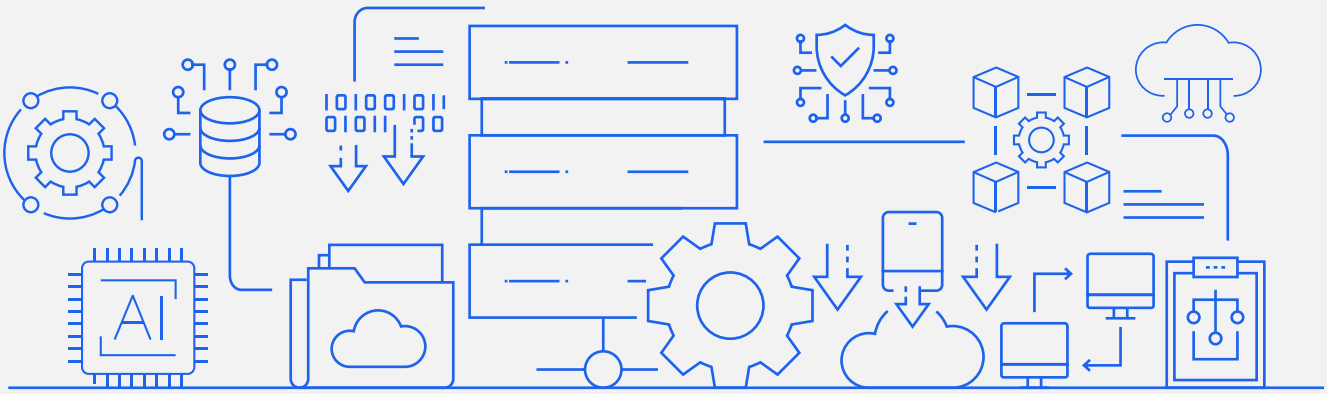
**Docker's approach to security** is about making the secure path also the easiest path for developers. This begins with content. Docker Hub offers **Trusted Content** programs (like Docker Official Images, Docker Sponsored Open-Source images and Docker Verified Publisher images) that give teams a secure foundation to build on. These are high-quality, vetted container images – essentially known-good building blocks.

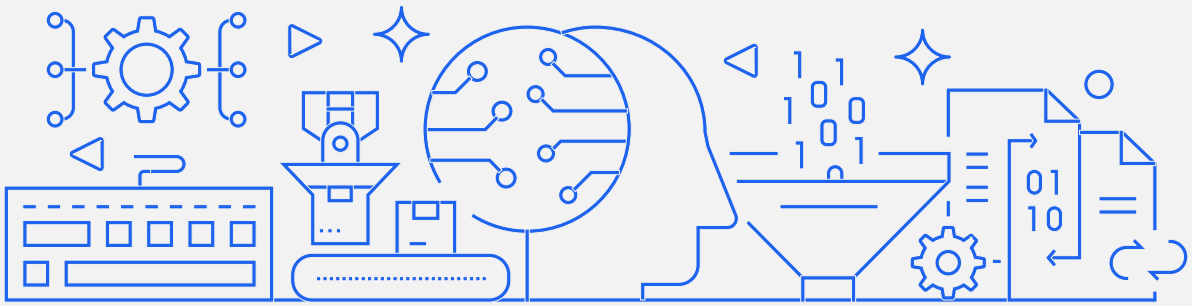
By providing **image provenance verification**, Docker's trusted content limits exposure to malicious components in early dev stages. Developers can pull from a library of over **8,000 trusted base images** and open-source projects, confident that these images are maintained, scanned, and come from recognized sources. In practice, this means when a developer starts a new microservice or AI workload, they're not grabbing a random base image of uncertain origin – they're using Docker-provided content that's already met a high bar for security and quality. The result is fewer surprises (like hidden malware or outdated libraries) and less time spent firefighting issues later.

# What about Generative AI?

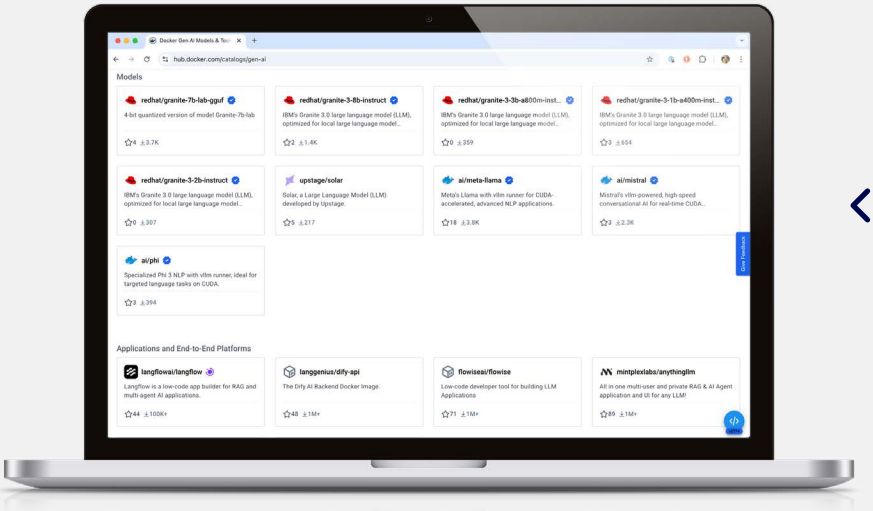
docker GenAI

Beyond base images, Docker is weaving security into new frontiers of development like **Generative AI**. The recently introduced **Docker GenAI Catalog** is a curated set of AI/ML container images – think of it as ***trusted content for AI models and tools***.





As AI development accelerates, companies worry about which AI frameworks or pre-trained models are safe to use. Docker's GenAI Catalog addresses this by providing pre-vetted, trusted AI images (from LLMs to vector databases) with consistent documentation and security posture. This empowers developers to integrate AI capabilities without the usual guesswork or risk of pulling unverified machine learning images. It streamlines AI adoption by ensuring that when your team experiments with an LLM or a new AI tool, they're using an image that Docker and its partners have curated for reliability and security. With Docker's trusted platform, teams can build AI applications confidently, knowing they have access to reliable tools in a trusted ecosystem.




GenAI

***GenAI Catalog simplifies the process of integrating AI into applications by providing trusted and ready-to-use content supported by comprehensive documentation.***

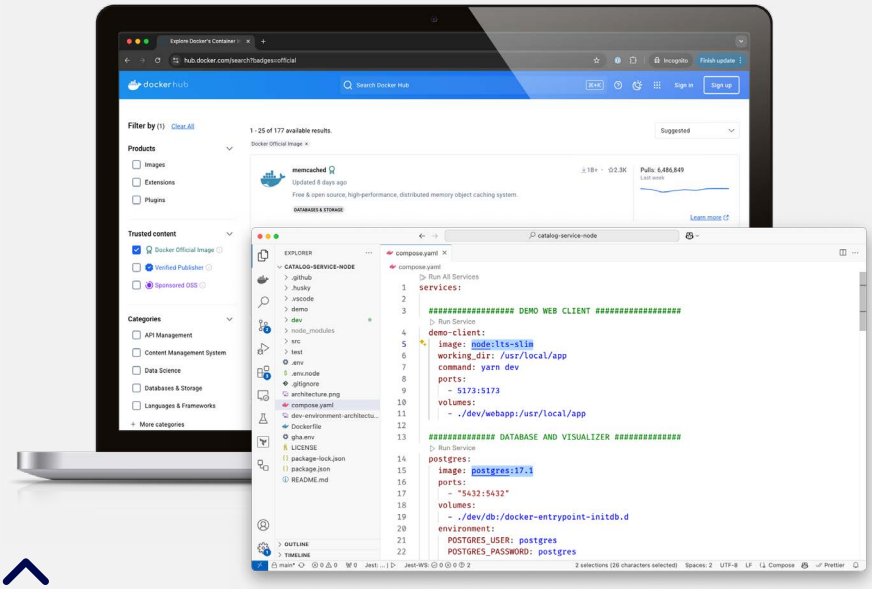
All these security enhancements are delivered in a way that **preserves developer experience**. Docker integrates security **without slowing down** developers. For example, Docker Desktop and Docker Hub now seamlessly include vulnerability insights (via Docker Scout) and image signing features – yet these run mostly behind the scenes or in CI pipelines, so developers aren't forced to context-switch into separate security tools. By the time a developer is ready to push code, Docker Scout has already checked their container for known vulnerabilities or policy violations, providing feedback in familiar interfaces (CLI, IDE, or Docker Dashboard). This "secure by default" workflow means teams don't have to choose between speed and security – they get both.

**Best practices** emerge from this embedded security mindset: Use trusted base images as a starting point for every project; incorporate security checks early (on the developer's machine or in version control) instead of waiting for QA; and leverage curated content (like the GenAI Catalog) when exploring new tech areas to avoid reinventing the wheel on security. By following these practices, organizations create a development culture where security is naturally integrated – developers move fast and build securely by design. In the next section, we'll zoom in on what this looks like in the **inner loop** of development (the day-to-day coding and building on a developer's workstation), where maintaining velocity with security is especially critical.

# Security in the Inner Loop

The "inner loop" of development – **the iterative cycle where developers write code, build it into containers, and test locally** – is where velocity is won or lost. It's also where security can be most effective if applied early. Docker helps teams **bake in security at the inner loop** by providing developers with secure, vetted tools and environments from the start. This ensures that by the time code leaves a developer's laptop, many potential issues have already been caught or prevented, without breaking the developer's flow.

One fundamental practice is using **vetted, secure container images** during development. Docker's **Official Images** and **Verified Publisher** images serve as a trusted baseline. Docker ensures these images adhere to best practices and are regularly patched. By relying on such images for your base OS, runtime, or key services, you significantly reduce the chance of pulling in a container that contains malware or critical vulnerabilities.




***Docker Trusted Content is a curated selection of high-quality, secure, and trusted images to extend or run directly.***

As Docker notes, the **Verified Publisher** program means images on Docker Hub are from trusted software publishers, allowing developers to "quickly and confidently use images from known, trusted sources". In other words, when a dev starts docker pull python (an official image defaulting to the latest version) instead of a random user upload, they're inheriting Docker's security diligence – which minimizes surprises later.

## docker desktop

Docker Desktop itself has evolved into a **hardened development environment** for enterprise teams. With Docker Business subscriptions, organizations can enforce Image Access Policies on Docker Desktop, meaning developers **can only pull images from approved sources**. This Image Access Management feature ensures that if a dev unknowingly tries to use an untrusted community image, Docker Desktop will block it, guiding them to use official or verified images instead. Similarly, with Registry Access Management administrators can ensure that their developers using Docker Desktop only access allowed registries.

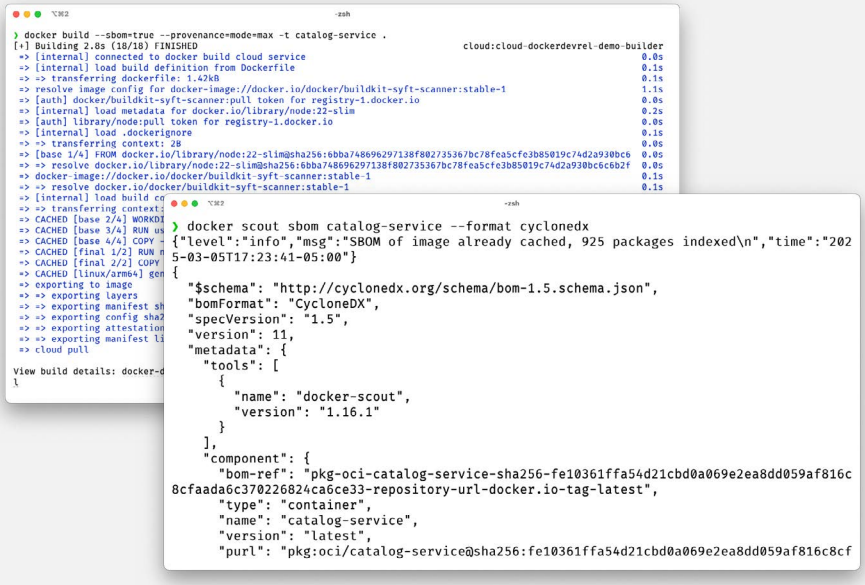
Such guardrails significantly reduce risk without requiring the developer to manually vet every image – the platform does it for them. Additionally, other features in **Hardened Docker Desktop (HDD)** include enterprise-grade security enhancements to the Docker Desktop application itself. This can involve everything from enhanced container isolation to verified signatures on Docker Desktop binaries. The goal is to secure the developer's workstation (which is increasingly a target for attackers). Considering that, already four years ago, **20% of breaches involved attacks on developer desktops or laptops**[8], securing the inner loop is not optional. Docker's HDD provides that extra layer of protection while letting developers maintain the productivity and Docker experience they love.

**Finally, in cases where security is absolutely paramount, such as heavily regulated industries or departments of defense, Air-gapped containers let you restrict containers from accessing network resources, limiting where data can be uploaded to or downloaded from.**

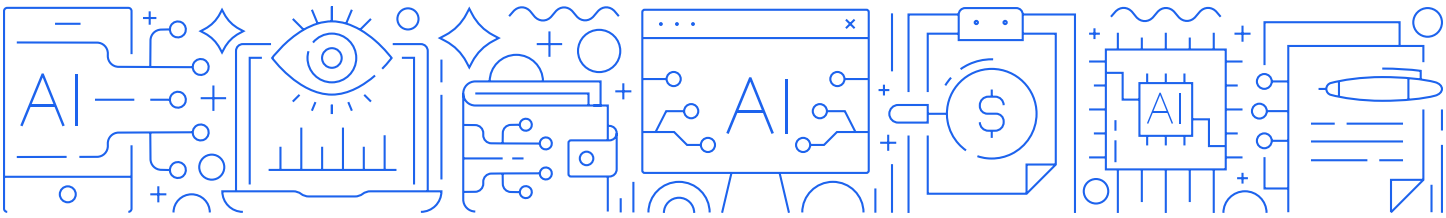
8 https://www.docker.com/resources/hardened-docker-desktop-summary-sheet/

Let's illustrate this with the everyday inner-loop workflow. A developer fetches a base image to start coding a new microservice. Because of Docker's policies, they fetch a Docker Official Image – say, Python 3:14 from the official repository – rather than some image with unknown provenance. As they write code, perhaps they use Gordon to suggest Dockerfile improvements and flag known bad practices. When they build the container locally, **Docker Scout** runs image analysis on the image layers, alerting the dev with if a critical CVE is present in, for example, that base Python image or any added package. The developer sees a quick report (in their CLI or Docker Desktop interface) highlighting the issue and maybe even suggesting an upgraded base image that fixes it. All this happens before the code is ever committed or pushed. The inner loop thus ends with a locally built container that is not only functional, but also meets basic security criteria – using trusted images, with no known critical vulnerabilities. This saves the developer's time by avoiding rework and unnecessary flags.

**All this happens *before* the code is ever committed or pushed. The inner loop thus ends with a locally built container that is not only functional, but also meets basic security criteria**




***Know every package, every tool, and every library, including their versions and license information.***



By securing the inner loop, organizations achieve two things: they **prevent many issues at the source**, and they free up developers to focus on features rather than fixing issues. It's much easier and cheaper to fix a security issue immediately on the developer's machine than to have it discovered during a late-stage pen test or, worse, by an attacker in production. Even if a malicious package or new vulnerability are discovered, your attack surface is reduced by features such as enhanced container isolation or air-gapped containers. This proactive stance is facilitated by Docker tooling. In fact, integrating Docker Scout early in development helps identify and address issues on the developer's machine, reducing the number of problems that ever reach a pipeline.

```
nickorefice@W20KDMK7PX ~ % docker run -it --rm --cap-add SYS_ADMIN -v Working:/mnt:ro alpine
/ # mount -o remount,rw /mnt /mnt
mount: permission denied (are you root?)
```

^

***Protect workstations from malicious code with Enhanced Container Isolation (ECI)***

Developers remain in their flow, coding and building as usual, but with a safety net that catches mistakes. The end result is code that is secure by construction and a development process that doesn't slow down. Next, we'll see how these inner loop gains extend into the **CI/CD pipelines** (the "outer loop"), ensuring security continues through integration, delivery, and deployment stages.


Detect vulnerabilities before committing code with Scout

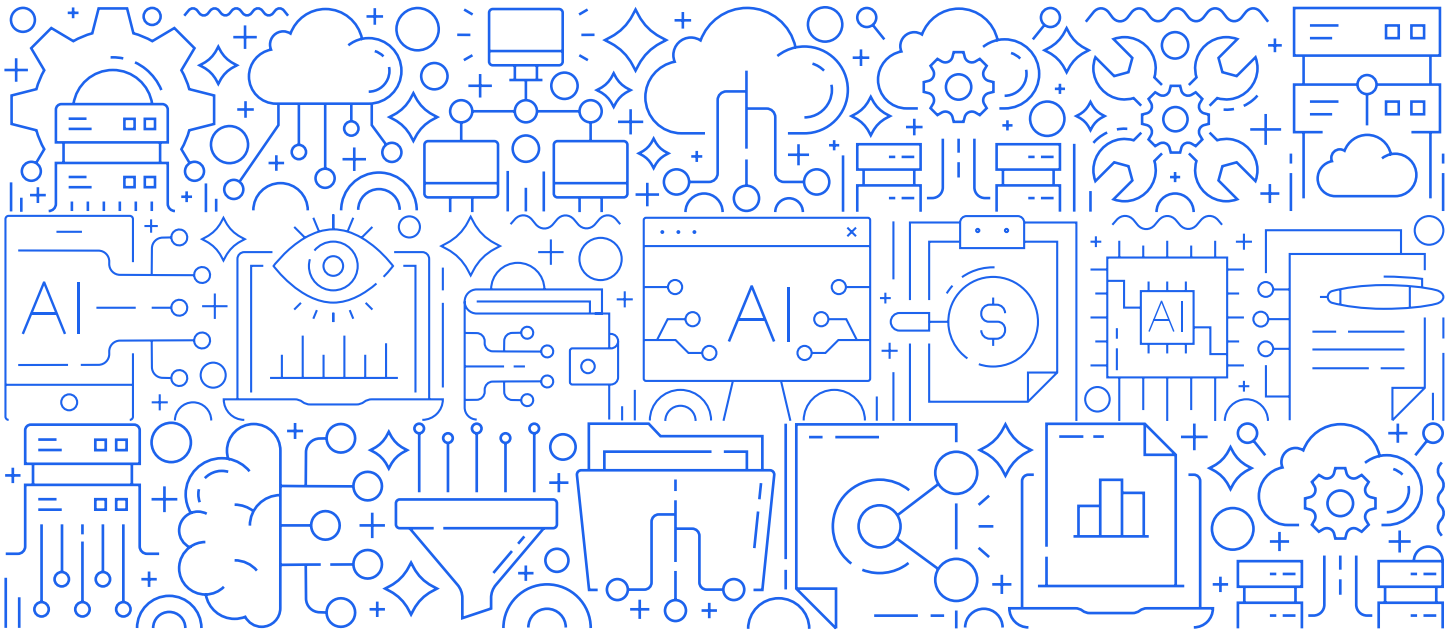
**SLSA Provenance and SBOM attestations**
Get insights into your software supply chain with Scout policies for open-source license usage and base rate image up to dateness

# Extending Security into CI/CD Pipelines

Securing the developer's workstation is a huge step, but it's not the end of the story. We must also **extend security into the CI/CD pipeline**, the "outer loop" that takes code from commit to production release. This is where Docker's tools integrate with continuous integration servers and deployment processes to enforce security continuously. The goal is to ensure that what leaves the dev team and moves into production is verifiably secure and compliant – with all the necessary metadata (like attestations and SBOMs) to prove it.

# Provenance is the missing link

One critical aspect of ensuring good quality and secure code enters your software supply chain is **provenance**. Preserving **provenance** and generating **SBOMs** for your container images. Every container that is built in CI can carry with it a bill of materials (a list of all components and libraries inside) and a cryptographic signature or attestation that vouches for its source. Docker has embraced industry standards, like SLSA, here. For example, Docker builds can be configured to produce an SBOM for each image (using tools like Syft under the hood) and sign images using **Docker Content Trust** (Notary v2) or integrations with Sigstore/cosign. By doing so, when an image flows through the pipeline, it's accompanied by evidence of *what it contains and who built it*. This directly addresses emerging supply chain requirements – it means if tomorrow a log4j-type vulnerability emerges, your teams can quickly search SBOMs to see which containers are affected, and it means you can prove to auditors or regulators that an image was built in your pipeline (not tampered with by a third party).




9 You can check our customer stories page for a collection of case studies showcasing the results of adopting Docker.

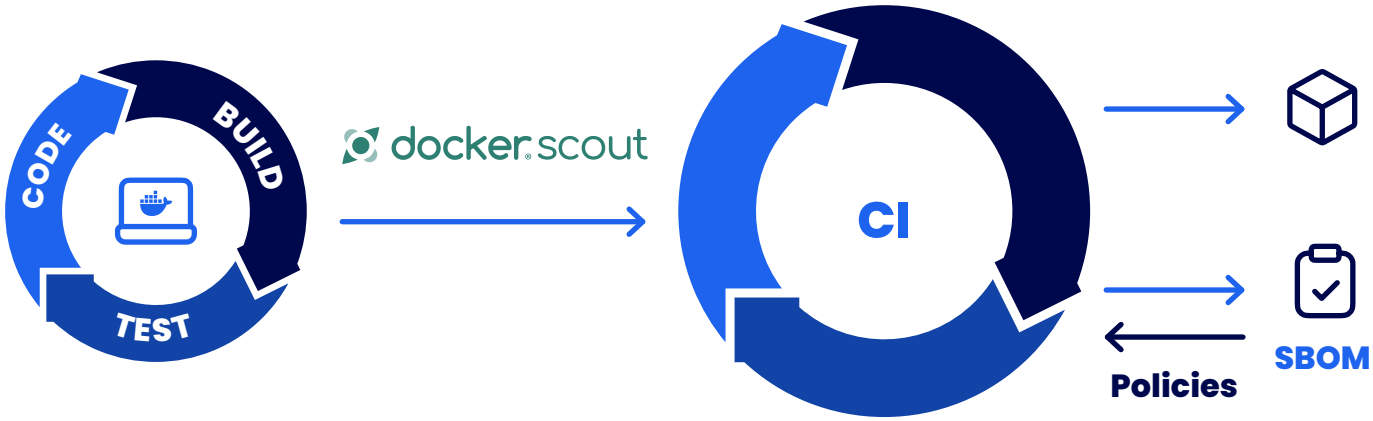
# Why implement policies in your CI/CD?

docker scout

Docker Scout, introduced earlier for local scans, also plays a major role in CI. It integrates with CI pipelines (GitHub Actions, GitLab CI, Jenkins, etc.) to **automate vulnerability scanning and policy checks on every build**. This means that as soon as a developer pushes code and the CI system builds the new container image, Docker Scout analyzes it: Are there new vulnerabilities introduced? Does it comply with your organization's policies (for example, no critical vulnerabilies, only use base images from a trusted registry, etc.)? If something's amiss, Scout can identify fail the build or notify the team, preventing risky artifacts from moving further down the line.

Docker Scout's policy engine is flexible – security teams can define rules such as "All images must have a vulnerability score below X and include an SBOM attestation." These policies enforce your internal security guardrails. Unlike simplistic pass/fail tools, Scout's policy evaluation helps teams ratchet up security over time, meaning you can continuously improve your security baseline (for instance, today block critical issues, next quarter also block highs, etc., as your software gets healthier).

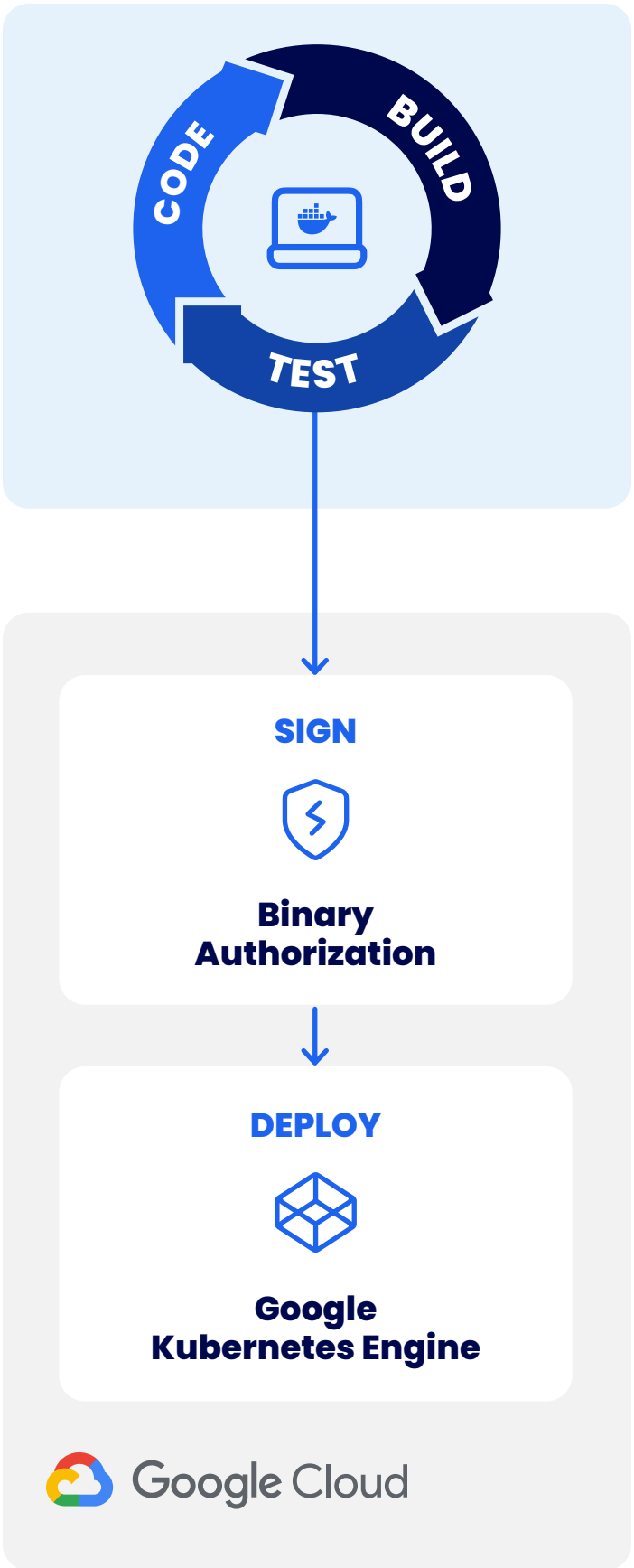
In practice, a secure CI/CD workflow with Docker might look like this: The CI pipeline builds an image and immediately signs it (producing a signed digest and storing an attestation that "our CI built this at this time"). Docker Scout then scans the image layers for vulnerabilities and evaluates it against security policies. It might add an SBOM file as an artifact, or even embed it in the image. If everything is clean, the image is pushed to your registry along with a notation that it passed security checks. If something fails, the pipeline stops and developers get feedback on what to fix.
By the time an image is ready for deployment, we have high confidence in its integrity and contents. We know **who built it**, we know **what's in it**, and we know it meets our org's security standards.



This approach effectively **secures the software supply chain**. It addresses concerns raised by incidents where attackers compromised build systems or injected code in transit. With Docker's CI integrations, each container is traceable and verifiable – a concept often referred to as "immutable, auditable infrastructure." It also aligns with frameworks like SLSA (Supply-chain Levels for Software Artifacts), which advocate for signed provenance at each build step. Docker's tools make implementing these best practices far simpler by baking them into the container workflow that teams already use.

To illustrate the impact: by enforcing these measures, organizations dramatically reduce the risk of a supply chain compromise. Even if an attacker somehow introduces a malicious component, it's likely to be caught by automated scans or fail a policy check long before deployment. And if not, the attestation and SBOM trail would make detecting and responding to such an incident much faster. In essence, Docker in the CI/CD pipeline acts as an ever-vigilant security gate – one that operates at machine speed, so it doesn't slow down your delivery but still prevents unsafe components from getting through. As we move to the next section, we'll see how this foundation enables a powerful synergy with Google Cloud's security features, ensuring that when you finally deploy to the cloud (e.g., on GKE), only trusted images run there.

**To illustrate the impact:**
**by enforcing these measures, organizations dramatically reduce the risk of a supply chain compromise**



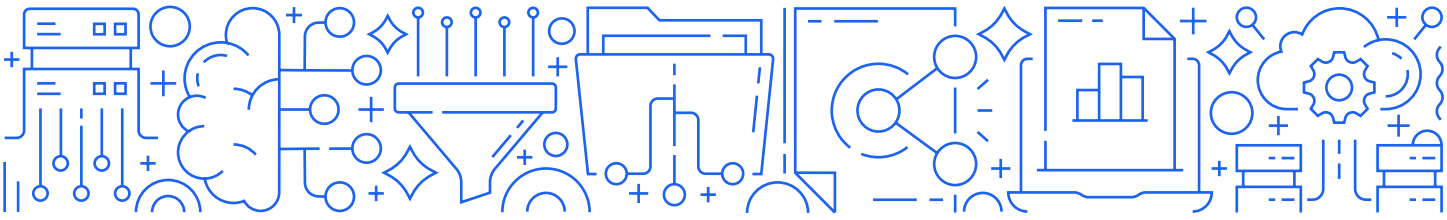
# Docker & Google Cloud: A Secure Ecosystem

Delivering secure software fast isn't just about dev and CI – it extends through deployment and runtime. **Docker and Google Cloud** together form a powerful, secure ecosystem from a developer's laptop all the way to a running service in the cloud. By using Docker to containerize and secure your applications, and Google Cloud's managed infrastructure to enforce security at deployment, you get end-to-end protection with speed and scalability.

**Google Kubernetes Engine (GKE)**

One key integration point is deploying Docker containers to **Google Kubernetes Engine (GKE)** with security features like **Binary Authorization**. Binary Authorization (BinAuthz) in Google Cloud is a deploy-time security control that ensures only images that meet certain criteria (for example, properly signed by your organization, or scanned and approved) can execute in your GKE clusters. In practice, you can configure GKE so that it will **reject any container that isn't cryptographically attested as safe**. Google Cloud's Binary Authorization service verifies that a trusted authority has signed the image before it's allowed to run, preventing unverified or malicious images from ever being deployed.

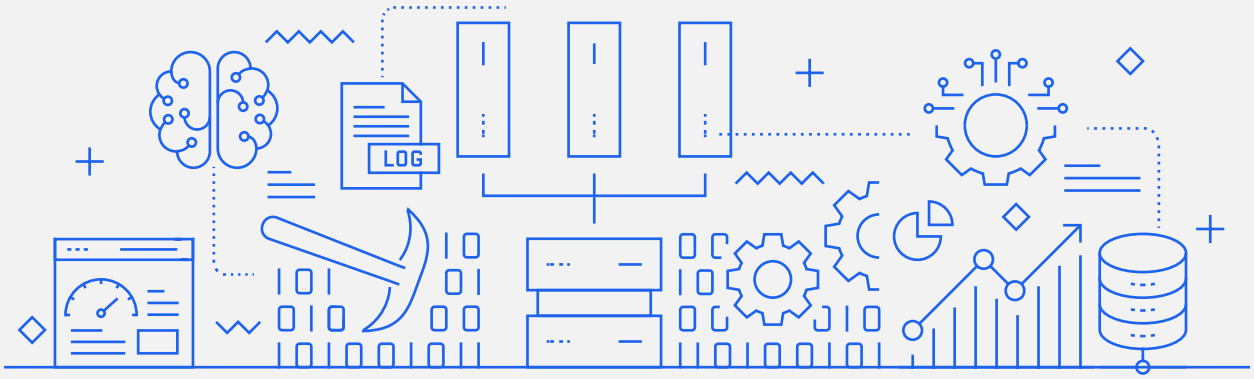
This is the perfect complement to Docker's secure supply chain: Docker's CI pipeline produces signed, policy-compliant images, and GKE's Binary Authorization checks those signatures and policies again at deployment. It's a two-layer check that dramatically reduces the chance of a bad container slipping out. Essentially, Docker builds the trust, and Google Cloud enforces it.

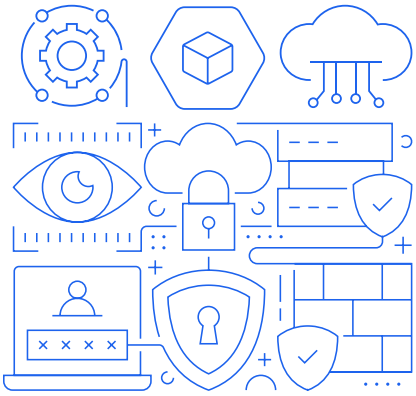


For example, imagine your CI has pushed an image myapp:v1 to Artifact Registry with an attestation that it passed all security checks. In your GKE cluster, Binary Authorization is set up with a policy: "only run images attested by my CI as approved." When you or your team attempt to deploy myapp:v1 on GKE, the Kubernetes admission controller (with BinAuthz) will look for that cryptographic attestation. If it's present and valid – deployment proceeds. If not, GKE will block the deployment. This way, even if someone mistakenly tries to run an image from outside your pipeline, or an attacker intercepts and alters an image tag, it won't run on the cluster. You've created a closed loop of trust from development to cloud.

Beyond Binary Authorization, Docker and Google Cloud align on other security dimensions. Docker's containers are fully compatible with Google's Artifact Registry and Cloud Build, meaning you can easily store your Docker images in a private, Google-managed registry that itself scans for vulnerabilities (another layer of defense).

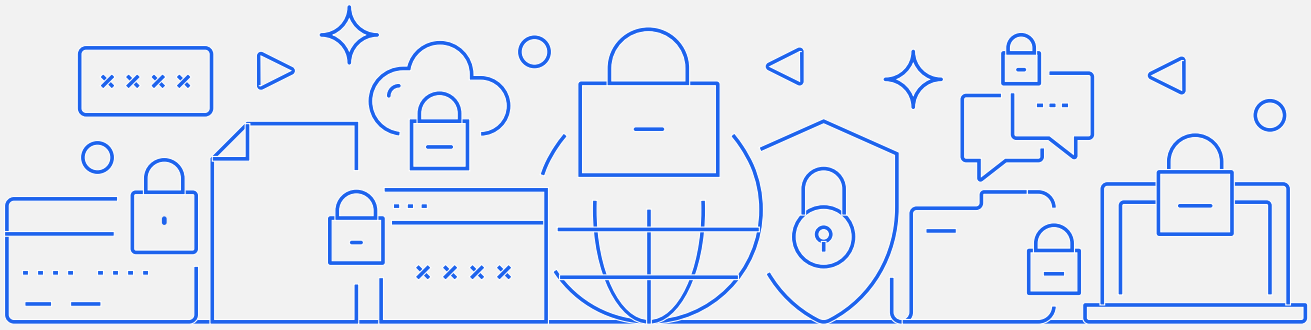
Google Cloud's **Container Analysis** service works with Artifact Registry to continuously scan images for known issues, similar to Docker Scout. Together, these layers create a belt-and-suspenders effect. Google Cloud can also leverage the **SBOMs and metadata** produced by Docker's pipeline; for instance, Google's Artifact Analysis can ingest attestations and show provenance information in the Google Cloud console, giving your security team visibility into an image's history.





Another integration is around **Kubernetes security**. Docker simplifies the path to securely orchestrating containers on GKE by providing standardized deployment artifacts. For instance, Docker Compose can convert to Kubernetes manifests, and those manifests can include notations to only use certain image digests (immutable references), which ties in with Binary Authorization's mode of operation (which prefers fixed digests over floating tags for verification). It's a bit technical, but the takeaway for our audience is: Docker's tools make it straightforward for platform engineering teams to generate the configurations that Google Cloud's security mechanisms can then enforce. The result is a deployment pipeline where everything is defined, consistent, and locked-down.

All of this yields **end-to-end security**: from the moment code is written (secured by Docker in development), to the moment it's running in production (secured by Google Cloud's policies). And it does so without slowing down delivery. Your developers can still push updates daily or even hourly, confident that automation will catch security issues, and operations can deploy containers to GKE knowing that the cluster will only run what's been approved. In effect, Docker and Google Cloud Next-gen security features allow your organization to move at cloud speed **while meeting or exceeding security and compliance requirements**. This gives CISOs peace of mind (compliance needs are handled, the risk is mitigated) and CTOs the confidence that embracing cloud-native and AI won't open new floodgates of risk.



# Conclusion & Next Steps

**Security and speed are not mutually exclusive**. By embedding security into the fabric of development with Docker, and leveraging cloud security features on Google Cloud, organizations can deliver software faster and safer than ever before.

For CISOs, this means peace of mind that security governance is enforced even in a world of lightning-fast deployments. For CTOs and Engineering VPs, it means development teams can adopt cutting-edge technologies (from open-source libraries to AI models) without constantly hitting the brakes for security reviews. And for Heads of Platform Engineering, it means having a streamlined toolchain that automates compliance and security checks at every step, freeing your team to focus on innovation and reliability.

**Key takeaways for decision-makers from this journey include:**

01 Prioritize trusted content: Set a secure foundation to securing your supply chain by using trusted content, such as Docker Official Images, Docker Verified Publisher images, and Docker GenAI catalog.

02 Embed security into your development lifecycle: Employ tools like Docker Scout in development and CI to make security part of your development workflow and catch vulnerabilities and misconfigurations early.

03 Implement end-to-end policies: Implement image signing and verification (Docker Content Trust, cosign) and enforce deployment policies with services like Google Cloud's Binary Authorization to ensure only approved artifacts run in production.

04 Leverage ecosystem integrations: Take advantage of Docker's integrations with cloud services (GKE, Artifact Registry, etc.) to speed up development to have security built into your workflows.

05 Invest in culture and training: Docker's tools are developer-friendly – use that to get buy-in. Encourage a DevSecOps culture where developers understand the "why" of these security measures.

Looking ahead, as we venture beyond 2025, the importance of this secure-fast balance will only grow. With AI evolving, threats adapting, and regulations tightening, organizations that have laid down a strong secure development practice will be two steps ahead of those that haven't. It's both a security imperative and a competitive advantage.

**Next Steps**: If you're ready to embrace this approach, there are several actions to consider. First, evaluate where you stand – perhaps perform a quick internal audit of your current container security posture (How many unvetted images are in use? Are you generating SBOMs today?).

**Engage with Docker to explore how our solutions that can rapidly elevate your software supply chain security.**

By taking these steps, you position your organization to innovate with confidence. The path to **fearless innovation** is paved by integrating security deeply and seamlessly into your development processes. With Docker and Google Cloud as your partners, you can **drive your business forward** knowing that security is built-in every step of the way – enabling your team to create, experiment, and deliver like never before. The future belongs to those who can move fast **securely**. It's time to join their ranks.

docker

Google Cloud

# Resource

## Case Studies

**Case Study (Ataccama)**
Docker enabled faster development while ensuring security and high availability.

---

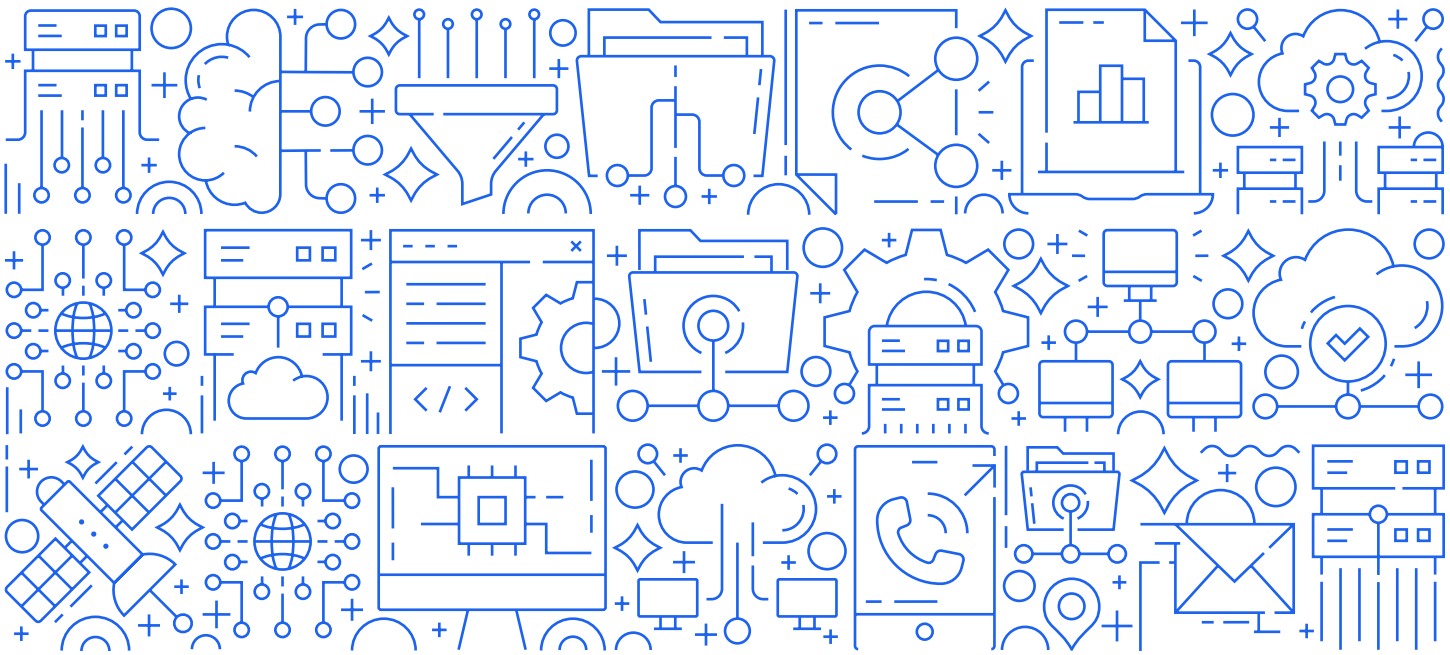
**Case Study (Keyfactor)**
Leveraging Docker and Open Source Ethos for Scalable Digital Protection.

## Documentation

**Docker Docs**
Your go-to resource for getting started, best practices, and advanced Docker workflows.

# Sources


- **Mollick, Ethan. (2024). Post on X (formerly Twitter).**
  Retrieved from https://x.com/emollick/status/1891913605890609624
- **ReversingLabs. (2024). The State of Software Supply Chain Security 2024: Key Takeaways.**
  Retrieved from https://www.reversinglabs.com/blog/the-state-of-software-supply-chain-security-2024-key-takeaways
- **National Bureau of Economic Research (NBER). (2024). Open Source Software Creators: It's Not Just About Money.**
  Retrieved from https://www.nber.org/be/20241/open-source-software-creators-its-not-just-about-money
- **Imperva. (2024). Cursor's Magic Comes With a Catch: Missing Trust Setting.**
  Retrieved from https://www.imperva.com/blog/cursors-magic-comes-with-a-catch-missing-trust-setting/
- **CIO Dive. (2024). Security Issues with AI-generated Code Highlighted by Snyk.**
  Retrieved from https://www.ciodive.com/news/security-issues-ai-generated-code-snyk/705900/
- **Federal Register. (2025, January 17). Strengthening and Promoting Innovation in the Nation's Cybersecurity.**
  Retrieved from https://www.federalregister.gov/documents/2025/01/17/2025-01470/strengthening-and-promoting-innovation-in-the-nations-cybersecurity
- **General Data Protection Regulation (GDPR). (2016). Art. 33 GDPR, Regulation (EU) 2016/679.**
  Retrieved from https://gdpr-info.eu/art-33-gdpr/
- **Docker. (2024). Hardened Docker Desktop: Summary Sheet.**
  Retrieved from https://www.docker.com/resources/hardened-docker-desktop-summary-sheet/

# Note

**Docker, Inc.**
3790 El Camino Real # 1052
Palo Alto, CA 94306
(415) 941-0376