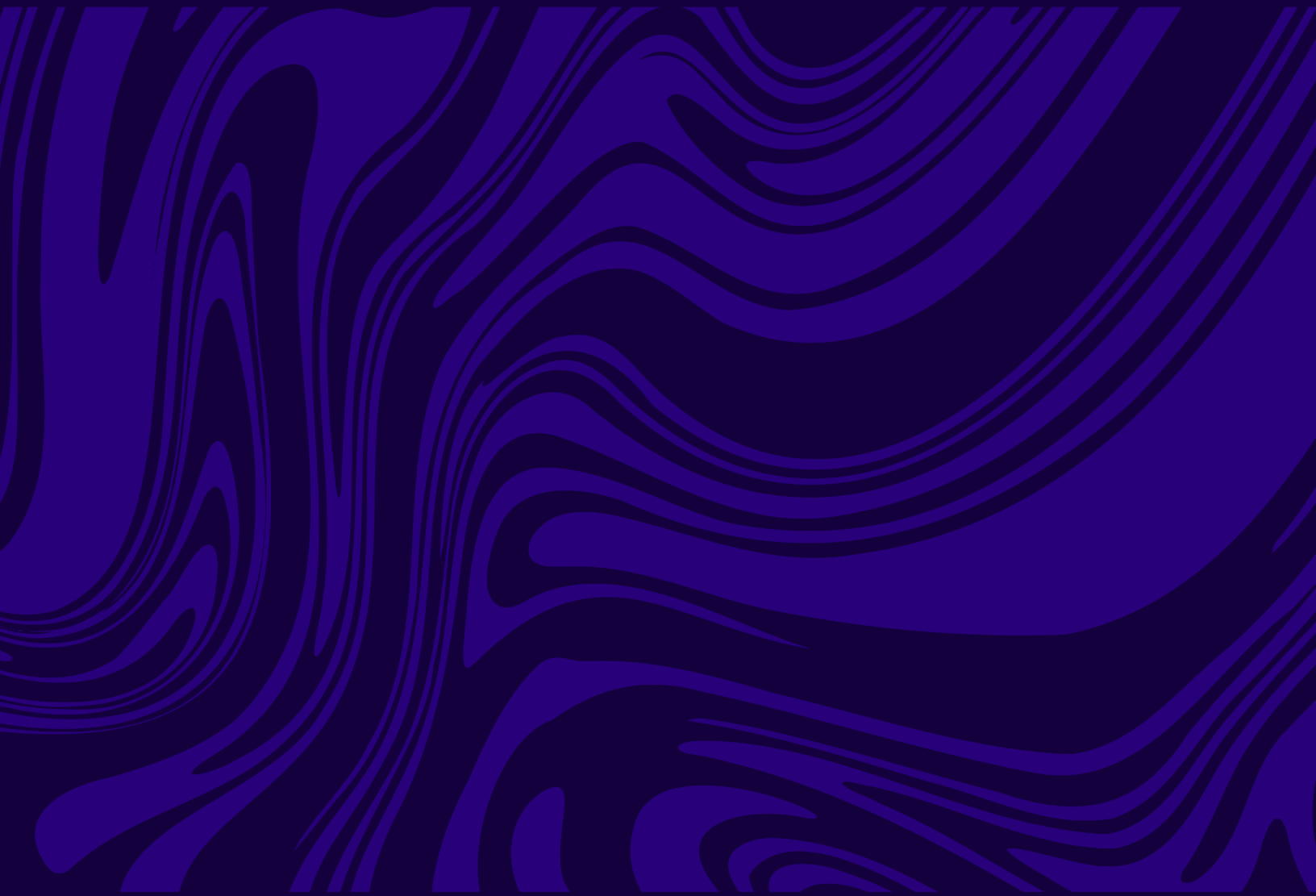
Chainguard

AI + Containers

Interviews about AI-Related Containers



John Speed Meyers and Dan Fernandez

November 2024

Executive Summary

Artificial Intelligence (AI) needs to live somewhere. The code underpinning AI, which is often popular open source software itself, needs to be packaged, or re-packaged, as software, somewhere, somehow. So where does AI live? Increasingly, it's in containers, the flexible building block that has become a fundamental unit of software.

But why does it matter if AI lives in containers? For one, containers, or at least many popular ones, have, to put it politely, "issues." And these container issues are creating problems for humans.

We believe this to be true because we conducted eight interviews in the summer of 2024 with data scientists, machine learning engineers, AI enthusiasts, and others who do "AI". Specifically, these interviews found that:

AI container users hate "bloat." Many AI containers, according to interviewees, have too much stuff and are consequently too large. It's a pain.

Security for AI container users rarely means identifying and fixing CVEs in container images. Despite there being hundreds of known vulnerabilities (CVEs), on average, in popular AI-related containers, the interviewees were concerned more about topics like network security, data leakage, and other issues.

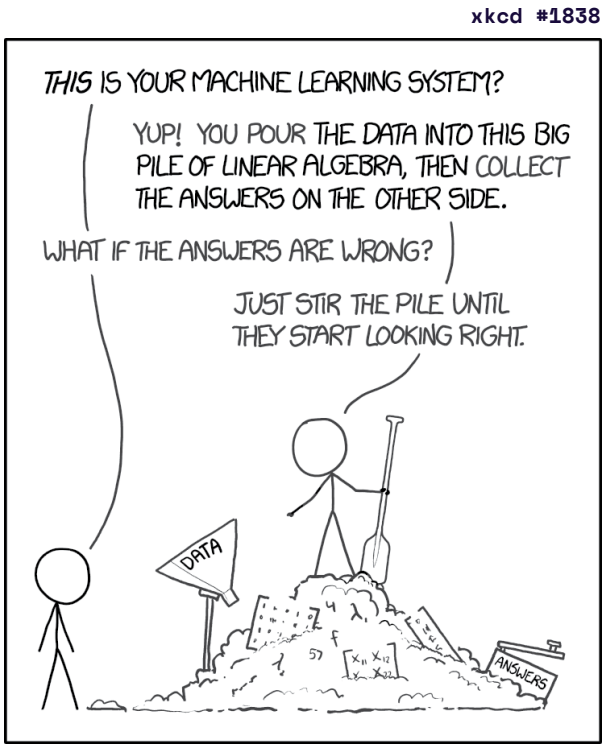
Dependency hell in the world of AI and containers is real. Frustrated interviewees described dependencies in AI applications as "finicky" and "fiddly."

Machine learning specialists often know little about containers. These professionals appear happy to hand off container-related decisions to platform engineering or machine learning operations teams.

Defaults and templates for AI container usage are powerful. Many container users simply inherit a Dockerfile from past developers and use it.

Fortunately, these problem-especially bloat, security, and dependency hell-can be addressed by creating better containers. The patient reader will need to wait though.

Introduction



Anyone who says that AI simply lives in the cloud hasn't accepted that the cloud is just someone else's computer (admittedly, the cloud is more like a lot of computers, all belonging to someone else). Contrary to the oh-so-good xkcd comic to the right, machine learning systems are not simply a pile of zero's and one's, a goop of bits and bytes.

AI, like all software, needs to be packaged. It needs a home, an orderly way to arrange itself so that users or other machines can use that software. Of course, software packages can take many shapes and sizes. Linux enthusiasts will recognize .deb, .rpm, and similar formats. Cloud developers will know virtual machines (VMs) and, increasingly, containers.

And here's the rub: the packaging format of AI is not just a minor technical detail–it's not something only Dale or Kimberly down in IT need to worry about. It's something that Aeva in security, Ryan in sales, and Melissa in finance need to care about too.

Why do we say this? It's because we conducted eight interviews over the summer of 2024 with data scientists, machine learning operations engineers, and others who deploy AI via containers, searching for what's working and what's not. Unfortunately, the interviews suggest that AI is taking on some of the worst characteristics of containers in general: AI-related containers are often bloated and riddled with known security vulnerabilities. More broadly, current AI containers, like AI in general, lead to "dependency hell" more often than not.

Fortunately, the story of AI in containers is not one of only doom and gloom. The interviewees revealed several technical and organizational shortcuts to ease the challenges of mixing AI and containers. First, platform engineering teams often can help relieve AI and machine learning developers of the headaches associated with AI containers. Second, sensible defaults—or 'guardrails' in dev-speak—can encourage developers to use better containers.

This whitepaper describes these main findings. After a brief description of the methodology, we describe each of the five main findings in detail.

Methodology

TL;DR: We interviewed eight people who do AI stuff and asked them about their experience selecting and using AI-related containers.

In July and August 2024, Chainguard openly solicited paid participation in an interview for software professionals responsible for AI models in production. We made contact with participants through a variety of methods: a form distributed on Chainguard social media, attendees of a major Linux foundation AI conference who made contact with Chainguard staff, and the personal and professional networks of Chainguard staff. It was a simple method.

Critics might call this sample "unrepresentative," but we would question the assumption that there is a simple sample of AI professionals that exists. The professional identities and titles of AI developers are so blurry and ill-defined that there is, at this moment, no way to easily sample software professionals responsible for AI models in production.

Methodological handwringing aside, these methods netted eight participants. If anything stands out, it's their diversity. The interviewees worked at companies of all sizes, in a range of roles, and with varying levels of experience with both AI and containers. If there are similar trends among these interviewees (and there were), then it likely indicates something broadly about AI and containers.

See Table 1 for a description of the types of companies at which they work, the employee count of these companies, their role, their years of experience with AI or machine learning, and their years of experience with containers. Each interview lasted 30-90 minutes.

The interviews followed a general format: after explaining their own background, interviewees discussed the criteria they used to select AI-related containers, their general experience with these containers, and any security concerns they had.

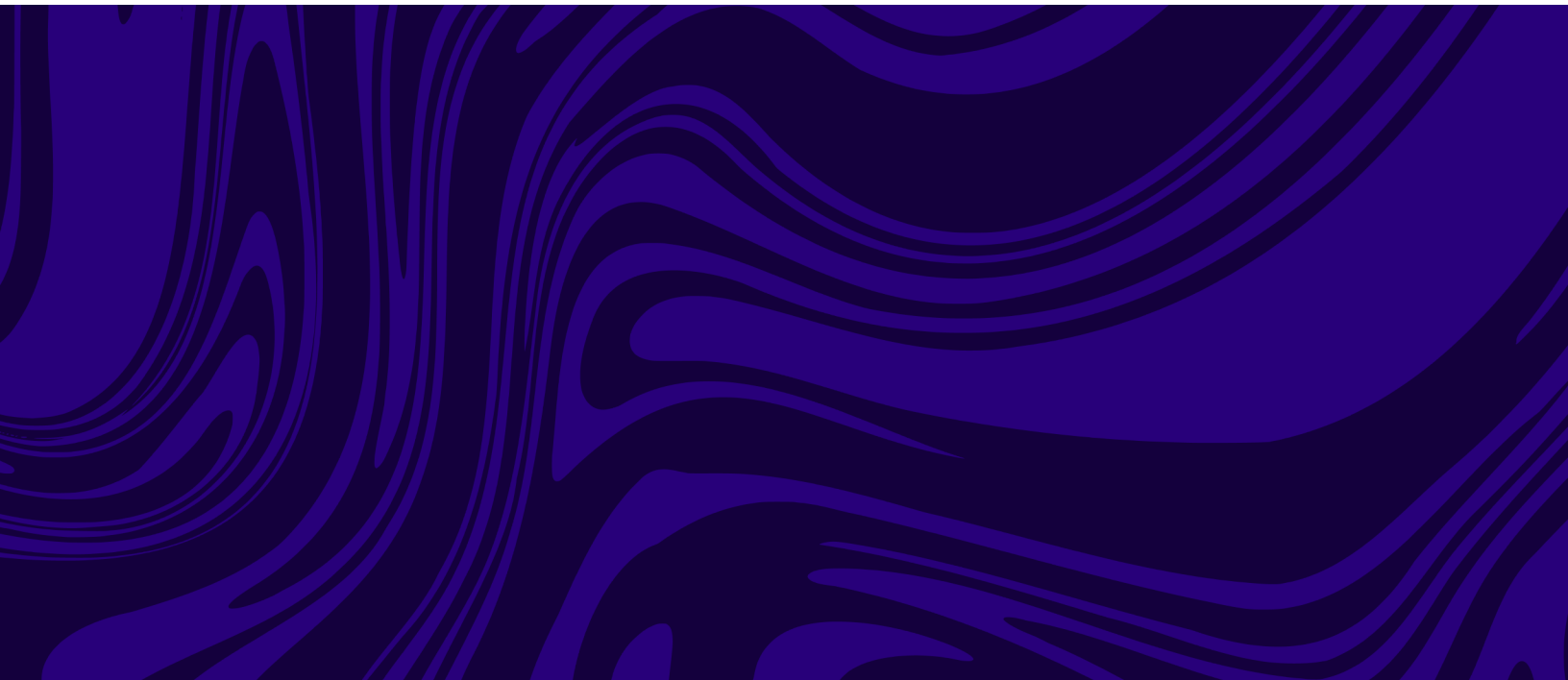


Table 1. Interviewee Details

Industry	Company Size	Role	# of Years Experience with AI/ML	# of Years Experience with Containers
Legal Technology	~250	Data / ML Engineer	8	7
Hobbyist	NA	AI Enthusiast	2	2
Software Consulting	~1000	Data Science Consultant	8	1.5
Major Technology Company	150,000	ML Research Scientist	10	7
Research Lab	200	Engineer	7	7
Cybersecurity	150	VP of Research and Data	20	7
Financial Services	~10,000	Platform Engineer	1	3
E-Commerce	10	Startup Co-Founder	2	2

Finding #1: AI container users especially hate "bloat."

Interviewees decried the weight or "bloat" of typical AI-related containers. These interviewees described often having to deal with containers that were 10GB or larger. This size created numerous headaches including problems with storage on shared machines, excessively high cloud computing costs, slow build times, flaky build systems and other issues.

A review of recent academic literature and an analysis of popular AI-related container images reveals that these interviewees have good reasons to be worried about bloat. One recent paper, entitled "Machine Learning Systems are Bloated and Vulnerable," found that:

> **Bloat accounts for up to 80% of machine learning container sizes.**

Specifically, that research team found that 15 popular machine learning containers contained, in their words, "significant bloat, over 50% in many cases." The researchers identified packages and files in these machine learning containers that were not actually used and termed these files and packages as "bloat."

A simple analysis of popular AI-related container images shows that, bloated or otherwise, there is no denying that these container images are large.

See Table 2 for basic information on the size in GB and number of components in a set of popular AI containers.

Table 2. Size and Number of Components in a Set of Popular AI Containers

Image Name	Size in GB (compressed)	Number of Components
Tensorflow (GPU)	3.6	375
Tensorflow	0.5	340
PyTorch (dev)	6.6	370
PyTorch (runtime)	3.0	258

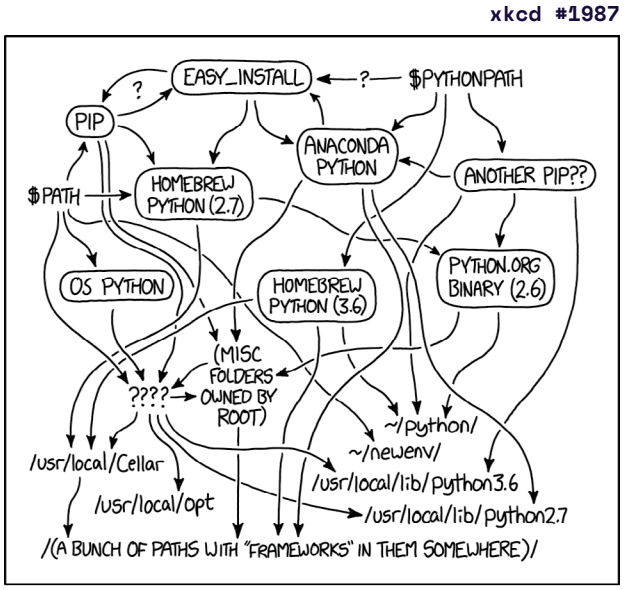
Number of components was calculated with the open source tool syft.

Some readers might respond that many containers are "bloated." However, these AI containers seem especially so. Debian-based community images on Docker Hub, often considered bloated themselves, are on average approximately 500 MB and contain 270 components. In contrast, three out of four of these AI images are substantially larger and several contain significantly more components.

This simple analysis, along with cited academic research and the experience of interviewees, suggests that AI containers are often "kitchen sink" containers, presumably intended for convenience. However, the interviewees expressed a preference for AI containers that are more minimal.

Finding #2: Dependency hell in the world of AI and containers is real.

Only very novice or superhuman programmers have avoided dependency hell, a state in which the dependency requirements of a piece of software conflict. It's such a common problem that it has its own wikipedia page. It often leads to situations like that in the comic below in which a beleaguered software developer struggles to make sense of a melange of packages and relationships. And it turns out that AI container users deal with dependency hell regularly.



Interviewees said that due to the rapid evolution of AI-related software and the desire to be on the bleeding edge, AI developers prefer using the latest versions. However, they almost unanimously described a litany of dependency version woes:

- Breaking changes despite no change in the major version of a package.
- More problems with CUDA versions than we can mention without boring the reader.
- Seemingly endless version conflicts in which fixing one version problem between packages A and B create a new problem between packages B and C.
- Confusion over which model version was actually included in a container

This is why the interviewees often used terms like "finicky", "crazy," "Whac-A-Mole", and "module hell" to describe their experience with AI-related containers and versioning. The problems were not insuperable, but they were a source of frustration and reduced productivity.

The interviewees wanted something simple yet seemingly elusive: AI containers that just work, devoid of dependency conflicts.

Finding #3: Security for AI container users rarely means identifying and fixing CVEs in container images.

With the exception of a platform engineer and a former CTO, none of the interviewees demonstrated knowledge about identifying, much less fixing, known vulnerabilities (CVEs) in containers.

This does not mean that the interviewees expressed no interest or knowledge in security and AI more broadly. Several discussed bread-and-butter security concerns like the importance of "networking zones" while others worried about more novel concerns like leaking information to large language model APIs and the possibility that customer information could become entrapped in future versions of those models. The data scientists in particular seemed more worried about bad chatbots (like the Air Canada chatbot that assured a passenger a free flight, wrongly) than CVEs.

Data scientists and other AI specialists seemed to show especially little knowledge about scanning for CVEs in containers. To these professionals, this vaguely sounded like a job for someone else, though they weren't exactly sure who other than "security."

This is arguably unfortunate given that many popular AI containers have many CVEs.

See table 3 for data on the number of CVEs in popular AI containers.

Table 3. CVEs in a Set of Popular AI Containers

Image Name	CVE Count	Critical or High CVEs
Tensorflow (GPU)	1,583	26
Tensorflow	1,531	26
PyTorch (dev)	1,997	41
PyTorch (runtime)	1,010	9

Number of components was calculated with the open source tool grype.

What interviewees wanted most was the ability to move fast without concern for security issues. In their minds, someone else should simply set guidelines and then the developers would comply out of laziness. What they definitely didn't want was to make decisions today--like using a container with numerous CVEs--that would turn into a nightmare sometime in the future.

Finding #4: Machine learning specialists often know little about containers.

The interviewees observed that many machine learning specialists have backgrounds in research, statistics, or specialties less focused on computing and more focused on mathematics. In the language of one interviewee, these specialists often have theoretical knowledge more befitting of a bespectacled scientist rather than, presumably, the hard-fought practical knowledge of a hardcore linux kernel developer. Consequently, these machine learning and AI specialists often leave, or at least want to leave, computing details, like choosing and maintaining a container image, to others.

This is why almost all interviewees either implicitly or explicitly endorsed the idea of a platform team that handles container image-related responsibilities: selecting "blessed" or "golden" images, keeping them updated, and ensuring that the base functionality is in good working order. The data scientists and AI developers want to build new applications, focusing on the last mile of AI application development, and don't want to care about the hundreds (thousands? more?) of dependencies beneath their top-of-the-app Python code.

For instance, the data scientist who worked at a major technology company (trust us, you've heard of it) had relatively little to say during the interview, both positive and negative, because some other platform team handled all container-related matters. Bug fixes and advice were only a Zendesk ticket away. Their container headaches were minor ones, at most.

The platform team can therefore specialize in knowledge, processes and tools related to containers, including CVE remediation, ensuring other developers can treat the containers like a magical black box.

Finding #5: Defaults and "templates" for AI container usage are powerful.

When these developers were asked how they choose container images to do their work, they often responded that they simply searched for an image (or Dockerfile) that was "standard." In other words, they didn't want to think; they didn't want options; they wanted something easy; they wanted to grab a container that was good enough and move on. Broadly speaking, these developers believed that the key to their professional success lay elsewhere and that choosing containers was an implementation detail.

Consequently, these developers were influenced by the default options provided to them either by available technical documentation or by other teams. One developer said that they simply chose images inspired by Microsoft's dockerfiles; several defaulted to official images from Docker Hub because of their broad usage (think "nobody gets fired for buying IBM"); and those with platform teams simply said they chose from a curated menu of images that were available to them. Also, several developers emphasized that they simply used the images and Dockerfiles chosen by previous developers on their team; the status quo was so powerful that little thought had been given to other options.

This is why several of the interviewees advocated for "defaults" and "guardrails." In their minds, if provided simple recipes (like Dockerfile examples with acceptable images and patterns), they would likely follow the default unquestioningly and would be no worse off in their day job. What interviewees wanted to avoid was tickets from the security team being sent long after the decision on what image to use was made.

AI + containers = contAIners

For such a complex technical topic, the general story is simple. AI increasingly lives in containers. Many containers have problems, especially related to bloat and CVEs. AI in general has dependency hell issues. AI in containers, like containers in general and AI in general, has these same problems, sometimes worse than normal. It's sad.

But it's not all doom and gloom. The interviewees themselves actually pointed to some reasonable approaches. First, data scientists and AI developers, according to these interviews, want someone else to be in charge of containers. A platform engineering team or ML operations team owning and curating AI-related container images seems like a step in the right direction. Second, the developers wanted guardrails, simple default choices that would protect themselves, their company, and their code from, well, themselves. Using a set of blessed or golden images fits this bill.

These reasons are why Chainguard has released Chainguard AI images. It's a set of minimal, low-to-no CVE images that are rigorously tested and built for deployment in AI and ML environments. Chainguard handles the dependency hell so you don't have to. Software teams looking for guardrails related to containers and AI need look no further.