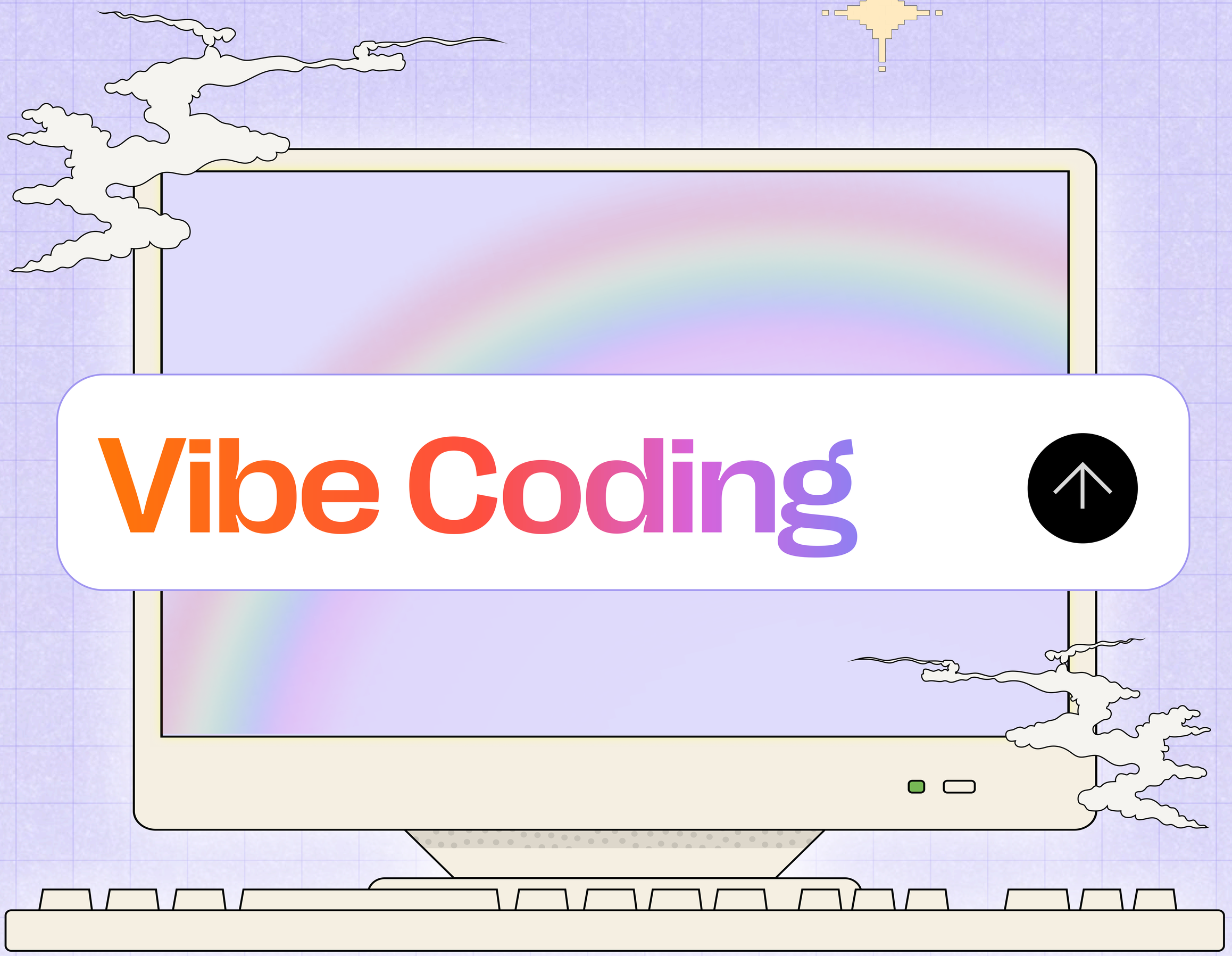
aikido



Security Checklist

FREE EBOOK

Intro

Unless you've been living under a rock, you've seen the hype around vibe coding. Just tell an LLM what you want in plain English, and it builds your app.

It's slick. But if you're not a pro dev, you might miss the risks until it's too late. Vibe coding makes it fast and easy to ship something. It also makes it fast and easy to leave your app wide open to attacks like SQL injections, path traversal, or exposed secrets.

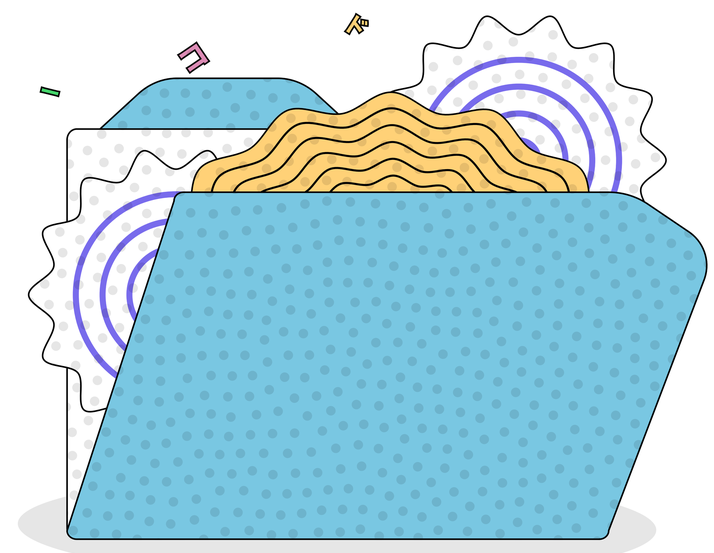
Some vibe coding platforms try to stay ahead of these problems. But real risks still exist. You could even call vibe coding "vulnerability-as-a-service."

Before we jump into the checklist, let's tackle the obvious question:

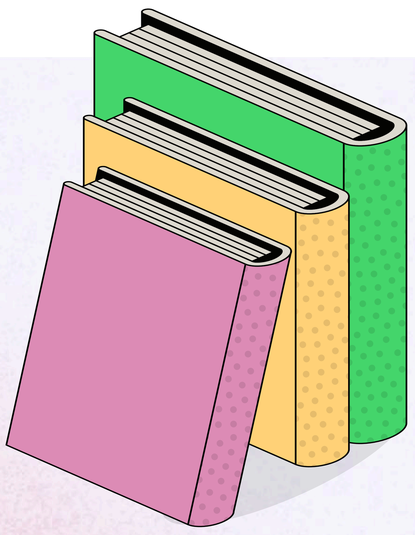
Can't I just ask the AI to make secure code?

Yes. Telling an LLM to write secure code does cut down on vulnerabilities. You can also ask it to find hardcoded secrets, check that data isn't public, scan for risky dependencies, or flag weak input validation. These all help. You can even plug AI into scanners (like how Aikido auto-triages alerts and suggests fixes). But don't rely on AI alone to secure your app.

The good news? Most secure coding practices still work. Plenty of companies and open-source projects tackle these problems. Here's a checklist to keep your app safe while vibe coding. It's split into basic, advanced, and pro levels.



The basics



1 Use version control

One of the pitfalls of vibe coding is when you try to add new functionality or fix a problem, the AI doesn't help you to get what you need and you're left worse off than you were before. That's why you need version control like Git to back up your progress.

2 Create a .gitignore file for sensitive files

A .gitignore tells Git what to skip. Use it to keep logs and other generated files out of your repo. These files clutter your history and might leak information that attackers can exploit. Always ignore your .env file. It holds sensitive data like API keys and passwords that could let someone breach your system.

3 Maintain a clear commit history

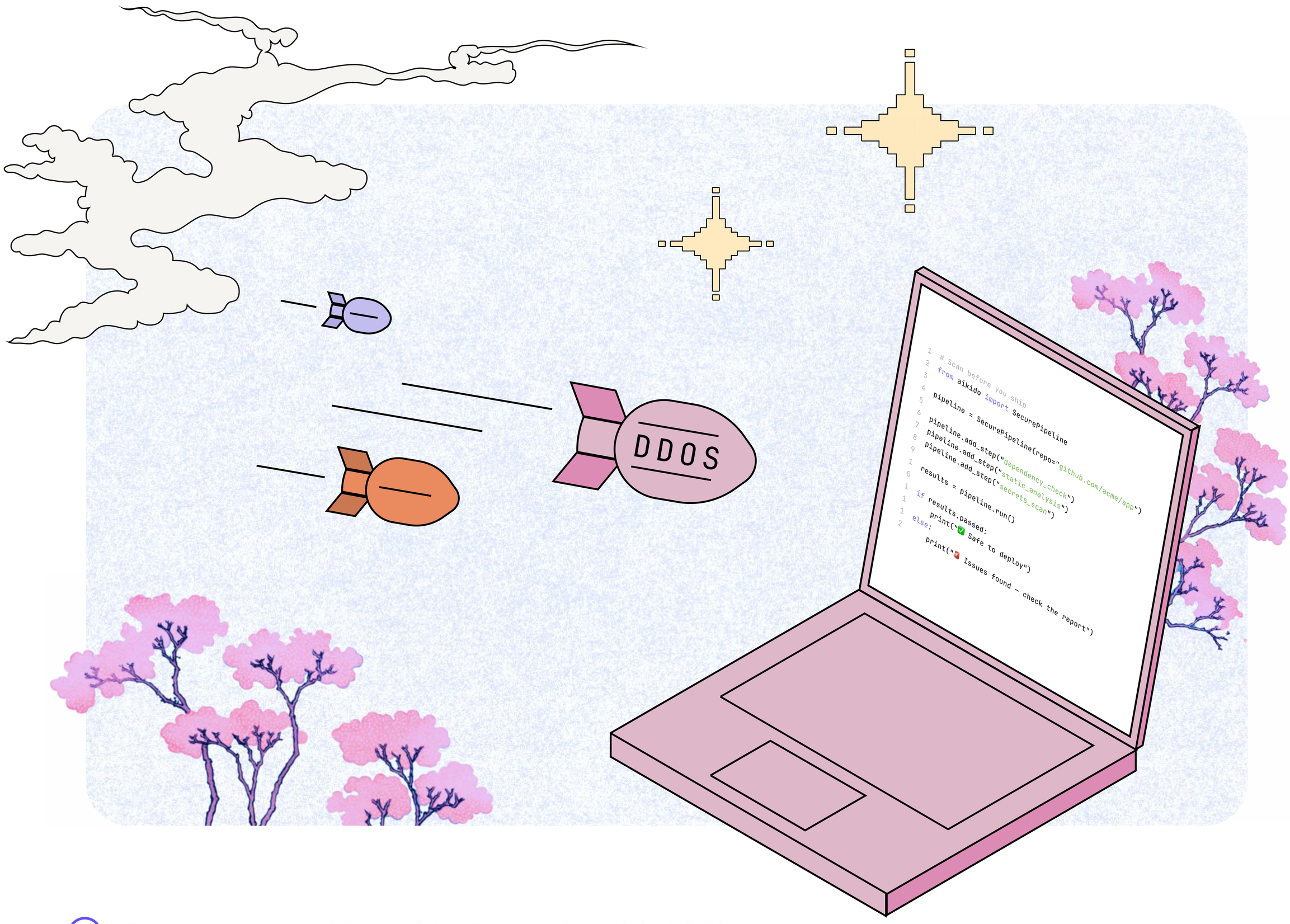
Keep commits small and focused. This makes it easy to spot when bugs or vulnerabilities were introduced and to roll back changes without losing unrelated work. Use signed commits to verify that only authorized developers (even if that's just you) are pushing code.

4 Separate feature, staging, and production branches

Use branches to isolate work. Build new features in separate branches to avoid pushing unfinished or insecure code live. Move tested features to a staging branch for a final review before merging into production. This ensures only stable, vetted code reaches your main branch.

5 Use a secrets manager

Secrets include passwords, API tokens, encryption keys, and certificates. If you commit them to your codebase, anyone with repo or even build log access can steal them. Always keep secrets in environment files or a secrets manager. Scan your code to catch mistakes. Aikido and Cursor IDE make this easy.



6 Protect yourself from DDoS attacks with CDN

A Distributed Denial-of-Service (DDoS) attack overwhelms your app with traffic, knocking it offline. They're common and can be costly, but also simple to block. Use a content delivery network (CDN) like CloudFlare or CloudFront, which provide built-in DDoS protection. Many domain hosts bundle CDNs by default.

7 Use dedicated authentication tools

Login flows, password resets, and sessions are risky to build yourself. Small mistakes can lead to breaches. Use dedicated auth tools that enforce strong password policies, support single sign-on, and add multi-factor authentication to protect your users and your brand.

Tools: • Socket • Phylum

8 Stick to trusted cryptography libraries

Crypto is notoriously hard to get right - even pros slip up. Avoid writing your own encryption or messing with crypto flags you don't understand. Stick to trusted libraries like NaCL, which keep you on secure defaults.

More advanced

① Set up CI/CD with security checks

A CI/CD pipeline acts like an automated assembly line for your code. Adding security steps means every change gets checked before it ships.

Tools:
- SAST: tools scan your code flaws (like injection points or unsafe functions)
- DAST: tools simulate attacks on your frontend—looking for open ports, risky inputs, or loose firewall rules.

Examples:
- Opensource DAST: ZAP
- Opensource SAST: Opengrep

② Watch your dependencies

Most of your app runs on open-source libraries. That's your supply chain, and a single bug in any dependency can compromise your whole app. Tools like Aikido and Trivy keep an eye on these libraries for known vulnerabilities.

Recommended tool:
- Aikido

③ Check for malware in dependencies

Attackers sometimes slip malware into packages that look safe. Once installed, they move quickly. CVE databases update too slowly to catch these early.

Tools:
- Aikido Intel

④ Use lockfiles

Lockfiles freeze your dependency versions so builds always use the same known-good packages. Without them, every install might pull the latest version—which could be buggy or malicious. For example, in March 2025, attackers tampered with tj-actions/changed-files, leaking CI/CD secrets through build logs. A lockfile would've pinned you to a safe version.

5 Defend against XSS with CSP headers

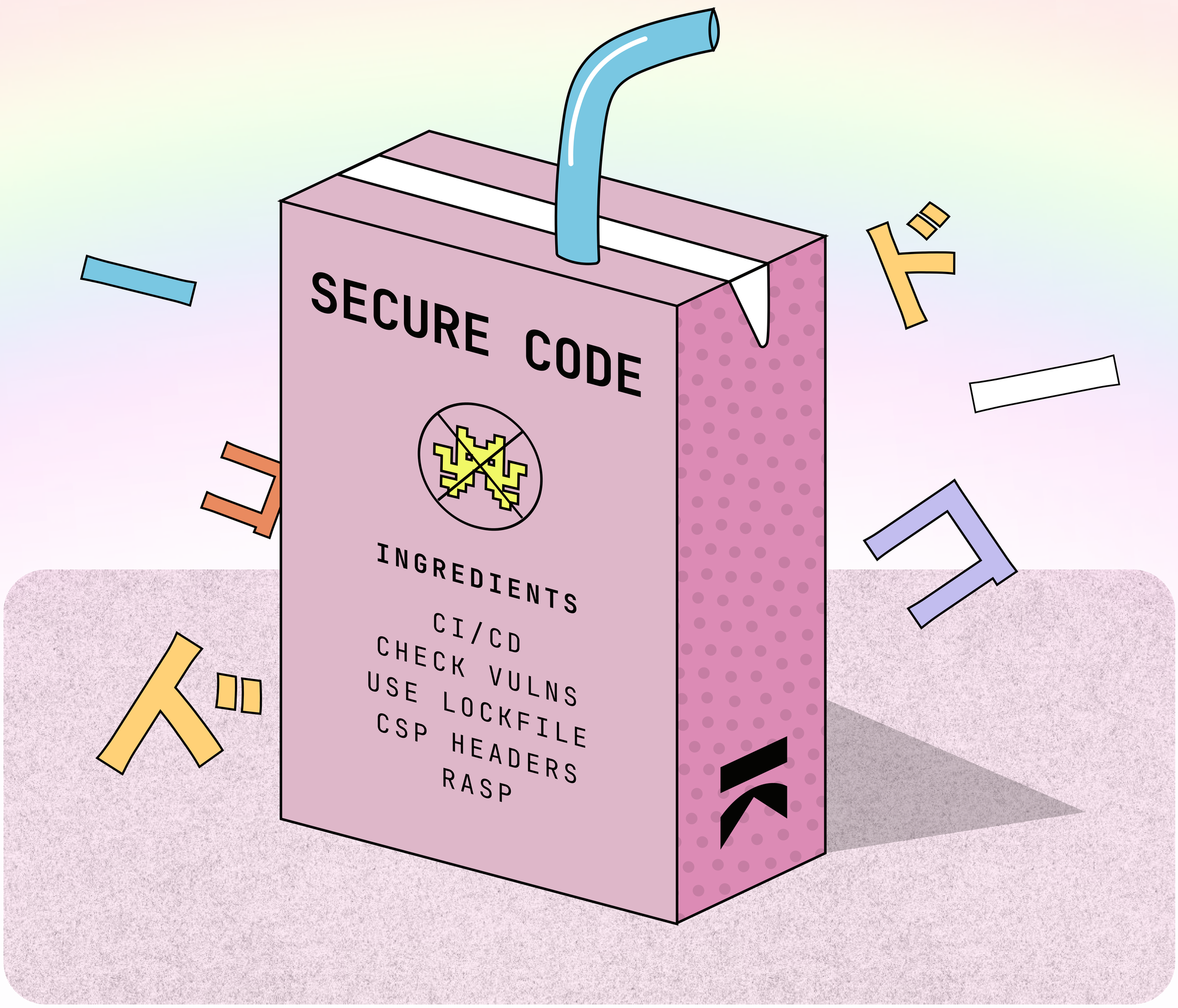
Cross-site scripting (XSS) attacks inject malicious scripts into your pages. Content Security Policy (CSP) headers let you control which scripts and resources load, blocking many XSS attempts by default. Use Aikido to check your CSP setup.

Check if you've set them up correctly with Aikido

6 Deploy a WAF or RASP

A web application firewall (WAF) or Runtime Application Self-Protection (RASP) serves as a last line of defense. They inspect incoming requests and block suspicious behavior—like zero-day exploits, SQL injections, or weird user inputs—before they reach your app.

Recommended Tools: • AWS WAF • Aikido Zen



Pro

① Harden your containers

Containers package your app and its dependencies so it runs the same anywhere. But they still need securing:

- **Update base images:** Regularly pull patches or use a platform like Heroku or AWS Beanstalk that handles it for you.

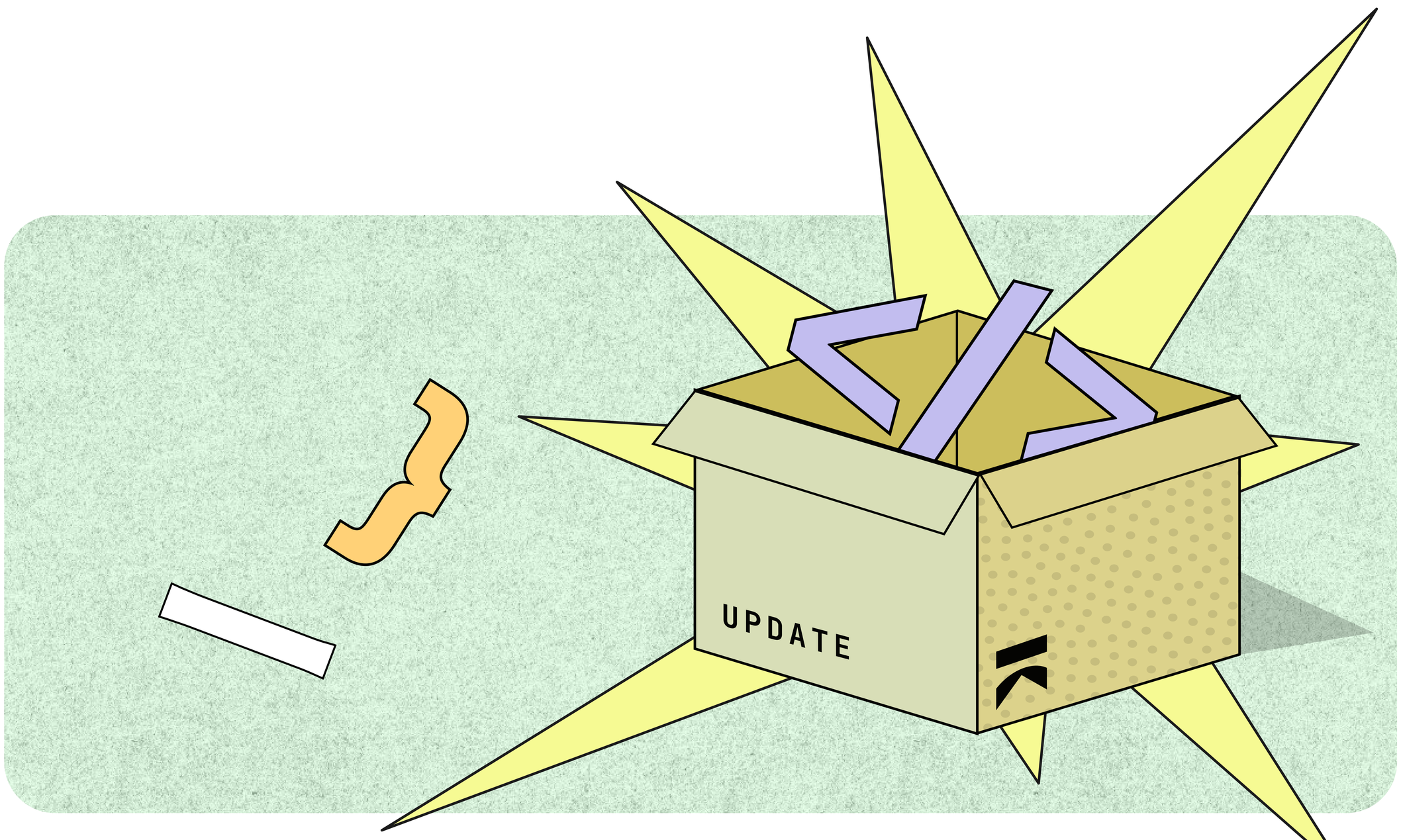
  Scan for issues using tools like Syft Grype Trivy to catch vulnerabilities inside your container layers.

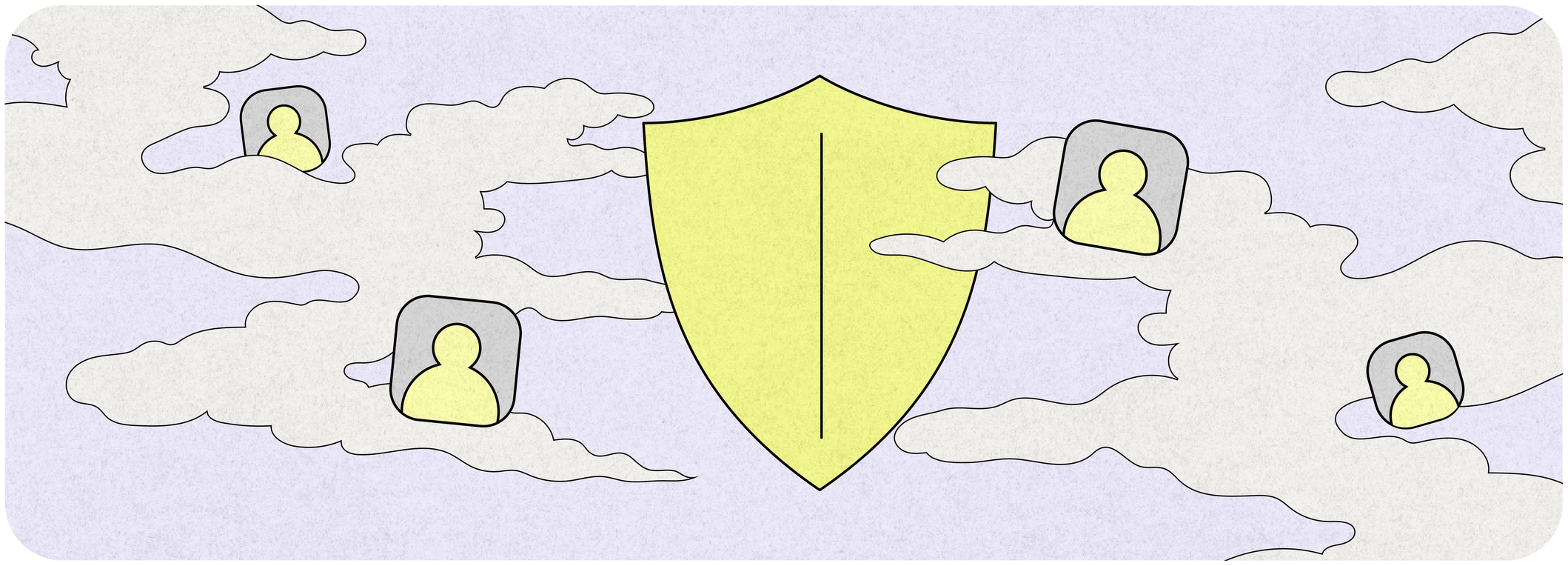
  Set it up in seconds with Aikido

- **Restrict privileges:** Avoid running containers as root. Keeping privileges low stops attackers from taking over your host.

- **Restrict privileges:** Avoid running containers as root. Keeping privileges low stops attackers from taking over your host.

- **Watch EOL packages:** Upgrade before libraries reach end of life. Aikido's container scanning can automate this.





2 Secure your cloud accounts

- **Separate environments:** Use different cloud accounts for dev, staging, and production. It's simpler and safer than trying to split everything with virtual networks.

- **Use CSPM tools:** Cloud platforms have endless options. A simple misconfig can expose your data. Tools like Cloudsploit or AWS Inspector scan for these mistakes.

  CSPM Tools: Aikido, Cloudsploit , AWS Inspector

  Set it up in seconds with Aikido

- **Set budget alerts:** If attackers hijack your account for crypto mining, budget alerts will warn you before the bills explode. Your cloud provider should offer built-in alerts for this, or you can set up cloud scanning with Aikido to check for budget concerns and risky misconfigurations.

3 Test your LLMs against known exploits

If your app uses LLMs - like a chatbot or onboarding assistant - test them against known exploits. The OWASP Top 10 for LLMs is a solid starting checklist so you don't accidentally expose your customers.

4 Build security into your development process

Security shouldn't be an afterthought. Shift left by adding checks early: follow security checklists, look for typical flaws in code reviews, and enforce security checks on pull requests.