
ibexa

Building bot-resilient websites through proactive testing

“Breaking things on purpose: testing your website before the bots do”

ibexa

Hello, I’m Lucas

- I’ve joined Upsun (then Platform.sh) almost 8 years ago (!) to be a solutions architect
- My background is in systems and networks engineering
- I am now the manager of the onboarding team, teaching *you* how to make the best use of Ibexa PaaS and Upsun in general

ibexa

Did you know?

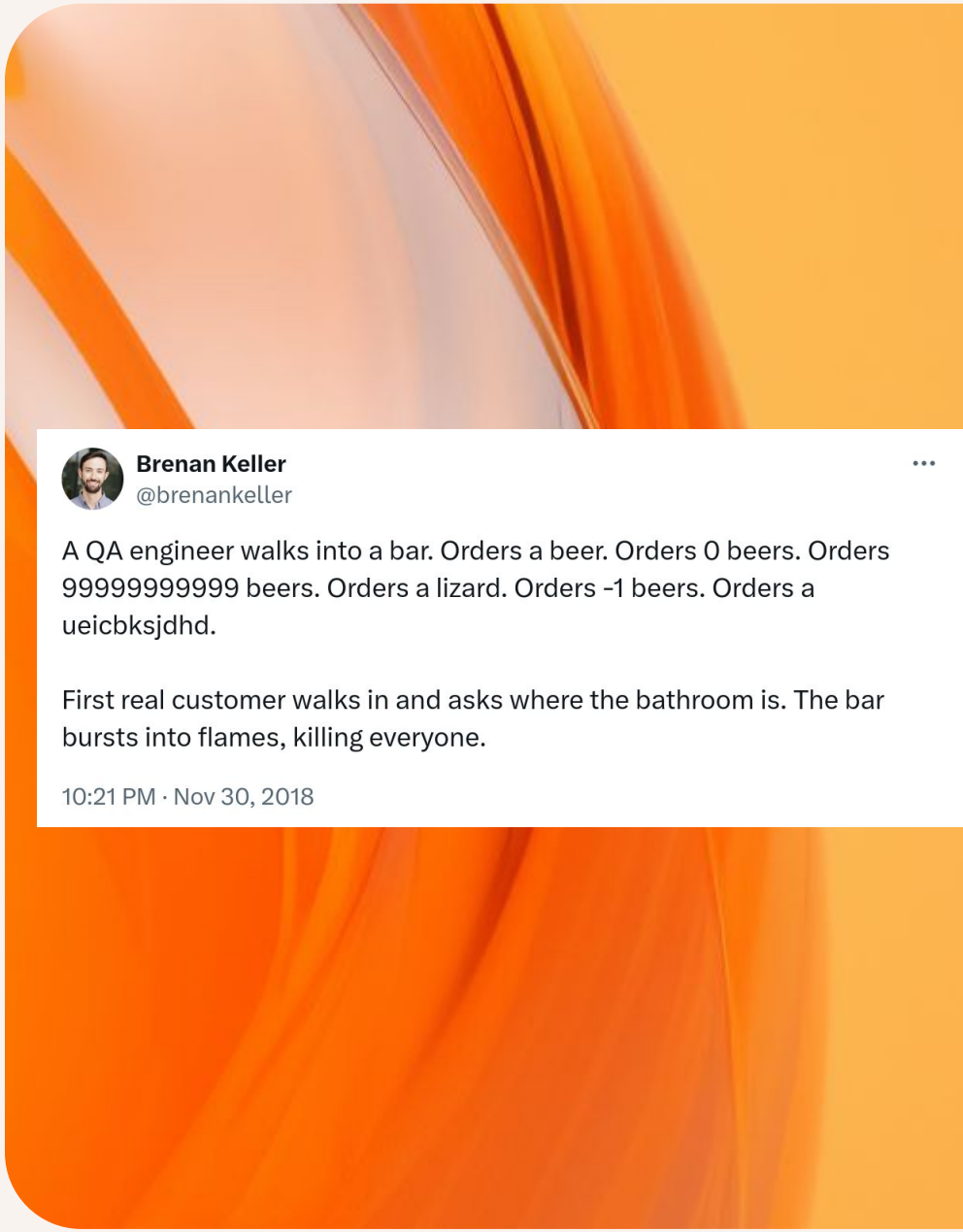
- In April 2025, Imperva measured that 51% of global internet traffic was made by bots.
  - This doesn't mean fewer human traffic, quite the opposite in fact; this only means more traffic, with different users
  - This is good and bad bots; Google is a bot you (probably) want
  - This is also industry-dependant; it is up to 61% in the travel industry

https://cpl.thalesgroup.com/about-us/newsroom/2025-imperva-bad-bot-report-ai-internet-traffic


Why even test?

Testing individual components is “easy”

- Test individual components, of course
- Use a hint from Elixir: “fail fast”;
  - Fail as early as you can
  - Leave logs and *nothing else*
  - Error gracefully, be kind to your user
- “Nothing’s easy until it’s done”



ibexa

How those components work *together* is difficult

- Interactions are where most bugs are
- Your app does a lot, works with a lot of services and external parts
- If any breaks or throws unexpected errors or content, are you ready?

Components working together *at scale* is why *you* (yes, you) are the best at what you do

- Like every kind of accident, you're not always to blame
  - Even Formula One drivers wear a seatbelt
  - Testing is your seatbelt; it can't prevent every accident, but it will limit damages
- Testing as many things as you can, in production-like conditions, as soon as you can, is essential.

What's testing?
What should I test?

ibexa

Testing is breaking in a controlled fashion

- Breaking things is *fun*!
- Ask for help, anyone can break things
  - Reporting framework:
    - What were you trying to do when it broke?
    - What were you expecting?
    - What did the breakage look like?
    - Can it be easily reproduced? Provide steps
- If someone reports something broken, fix it, then *write a test for it*
- Don’t test in production (I know, right?)

The two kinds of testing: “shallow” testing

- Shallow testing is what you could call “at scale”
  - Many users doing the same easy thing (cached front page...)
  - Many users doing the same difficult thing (logging in, buying something...)
    - Don’t overdo it! Oversizing tests will waste resources in more ways than one
  - Many users doing multiple things (checking a bunch of maybe-uncached articles at once)
    - Same here; don’t overdo it! "Clean cache then full test" is usually not an accurate real-world scenario

The two kinds of testing: “deep” testing

- Deep testing is what you could call “interactions”
  - As complete of a path as you can
    - http cache → crazy routing → database interactions → app cache
  - “What if”s…
    - What if app cache explodes because you have too many objects?
    - What if the database is slow?
    - What if the cache caches *too much*?


When and where
should I test?

Test early, often and everywhere you can

- Test as soon and often as you can, small and early is better than *not*
  - Just like working out, regularity is what makes testing good
- Individual tests on your machine (in a git hook…) are good
- You can run even more tests in a remote pipeline (GitHub Actions, GitLab pipelines…) for each merge request, if you use them
- Test the app in prod-like conditions as soon as you can
  - Avoids the “it works on my machine”®
  - Ibexa Cloud can automatically, completely and quickly reproduce your production environment, including data and sizing (soon)

How about the bots? What will *they* test?

There are two kinds of bots; the good bots and the bad bots:

- Good bots will respect your robots.txt
  - They could even follow `content-signal` directions
- Good bots will respect your rate limits (slow down on 421)
- Good bots won't spam uncached pages
  - Think search pages, logins pages, facets, account creation…
- Bad bots will do the opposite of ^, and sometimes even pretend to not be a bot by faking their user-agent

More often than not, bot issues are scale issues.

Alright, I'll test (more)

Tools to get through “shallow” testing

- K6 by Grafana is very good
  - It uses JS as its language, making it easy to understand and automate
- Octoperf is an example of a hosted load-test tool
- With Ibexa PaaS Flex (not yet available), you’ll be able to reproduce your production 1:1, with sizing if you wish; you’ll even be able to make changes on the fly

Tools to get through “deep” testing

- Remember Postman?
  - Hoppscotch
  - Bruno (usebruno.com)
- Curl is still *very* relevant
  - It’s the base for most tools
  - It’s likely already installed (unless you use Debian)
  - It’s amazing at reproducing simple tests
- Hurl (Orange-OpenSource/hurl) is very neat to share your tests
  - Based on Curl
  - Allows for complex tests, even chaining requests
  - Still uses regular, git-friendly text files
- With Ibexa PaaS, you’ll be able to reproduce your entire production environment, complete with services and data, in just a few minutes

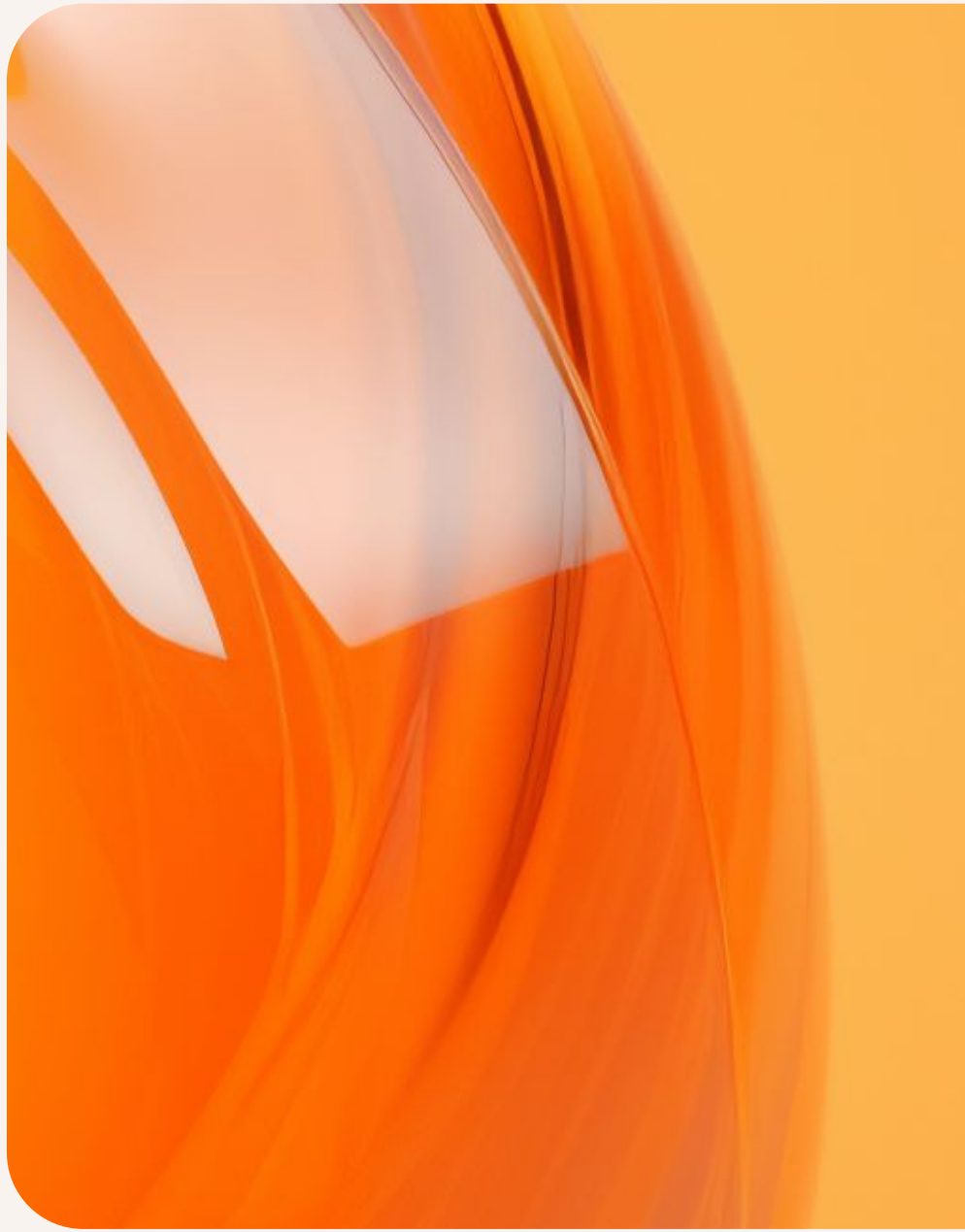
ibexa

Fighting fire with fire: getting robots to help with your robot problems

- AI is an amazing excuse to write good documentation
  - `AGENTS.md` is great to get your agent friend started
- From good documentation, AI can generate good tests
- Of course, good tests aren't *great* tests
  - you're still on the hook for those <3
  - Pro-tip: record yourself explaining the test, send the transcript to your fav agent
- AI is amazing at writing boilerplate
- AI is also a great cyber-rubberduck
  - This one doesn't float though, it drinks your water


Thanks a lot!
Questions?
ibexa


ibexa




ibexa