


Testing in production

Losing fear, one Javascript line at a time

Jorge Marín

September 26th, 2019 – JSConf Budapest 

Hidden slide

Welcome to JSConf Budapest and thanks for coming to my talk :D

I know it is still a bit early for some of you but I promise this talk won't be too heavy

breathe

no, srsly, breathe

About me


My name is Jorge, and I am an engineer 🖐️ 🇪🇸

- Worked on autonomous indoor navigation for UAVs
- Got dragged to the cloud space ☁️
- 3 years working at Bitnami
- Currently working at Dyson

<https://jorgemarin.xyz>

[@chipironcin](#)

Poll time - Raise your hands if you...

- Know what Javascript is
- Are here for the city, the food and the party (and a bit of Javascript)
- Know what testing is
- Know about the testing pyramid or any other way of classifying different testing levels
- Have a production environment
- Are afraid of testing in production
- Are testing your services in production
- Prefer spaces to tabs 

What is this talk about

- Mostly answers to Frequently Asked Questions
- Not going to be too technical
- Based on the experience gained working at Dyson

Frequent Asked Questions (#6)

- Why do you need tests in production?
- What is the right testing level against production systems?
- How can Javascript/NodeJS help with that?
- How to avoid disrupting real users?
- How to keep your tests out of statistics reports?
- Cleanup
 - Should you clean after yourself? What if you don't?

Let's dive in!



Why do you need testing in production? #1

Why do we mean by “testing in production”?

#0

- When you deploy your application
 - Application
 - Service
 - Lambda
- To guarantee it continues to work consistently after the release/deployment
- Automatic scheduled tests to guarantee it is working as expected after the release and no external factor is negatively affecting the results it should yield

Why do you need testing in production? #1

- Imagine a lambda function that returns the current date
 - BUT it doesn't cater for the Year 2038 problem
 - Deployment is fine. All tests are fine.
 - But after a handful of years...
 - Dynamic input/output that depends on the context it is going to be executed
- Or imagine one of your stateful services becomes inconsistent and starts failing for all customers

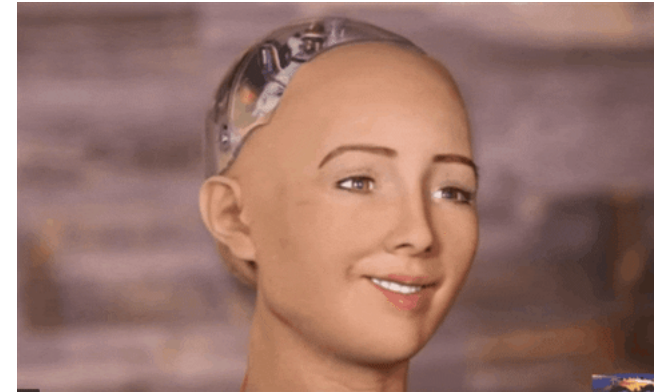
Why do you need testing in production? #1

- What do you prefer:

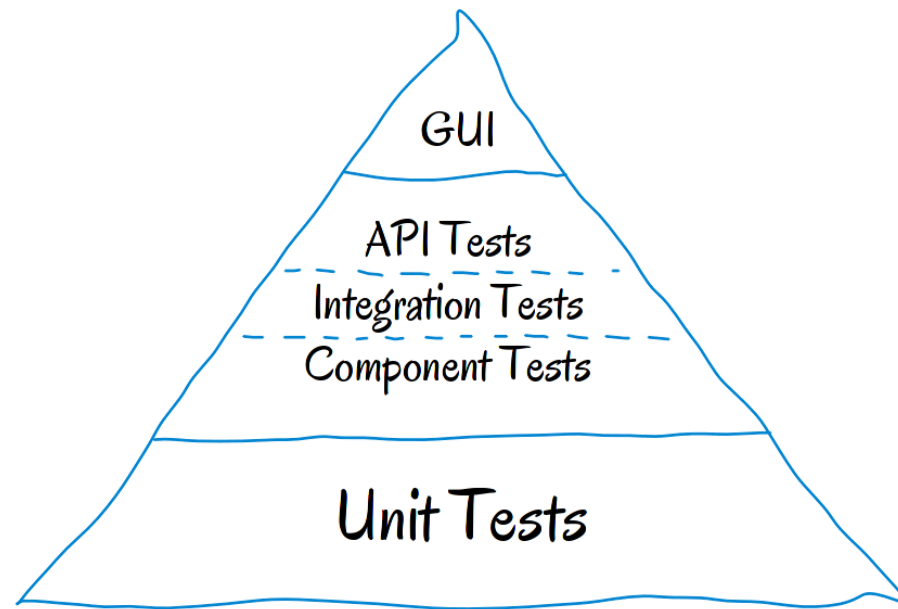
Real angry **paying** customers calling you because what they paid for is not working



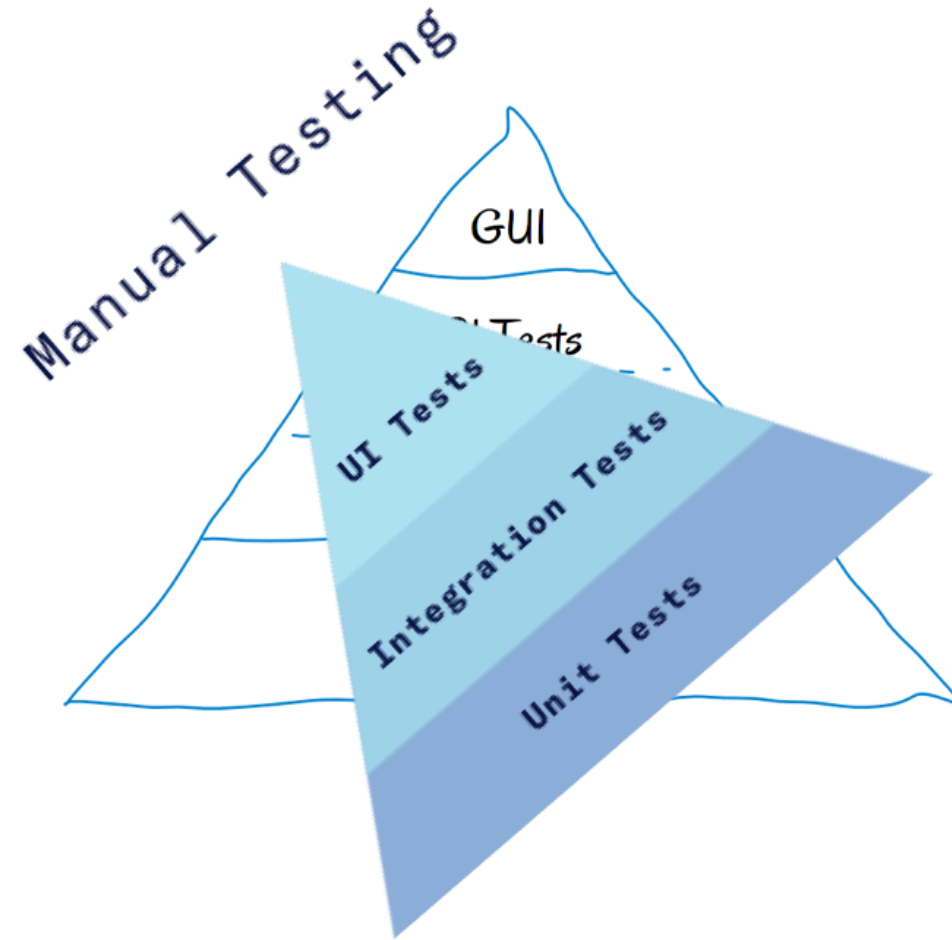
An automated process acting as a real user notifying you that your application/service started behaving wrongly



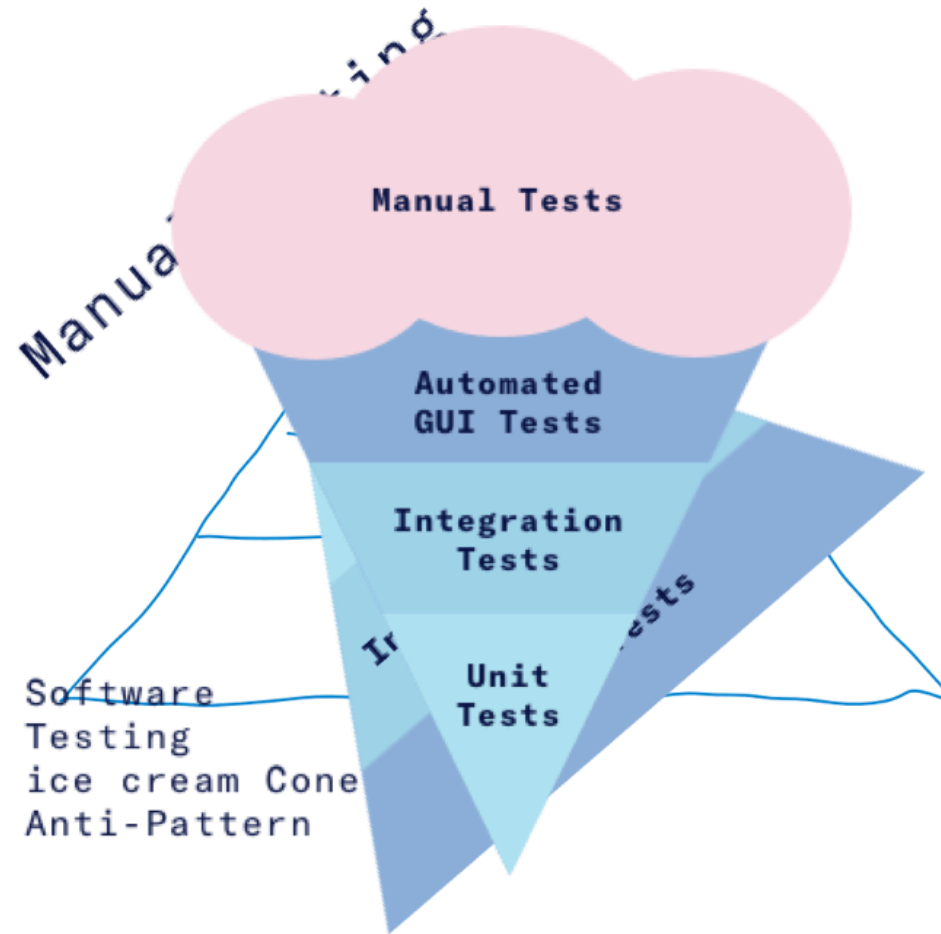
What is the right testing level against production systems? #2



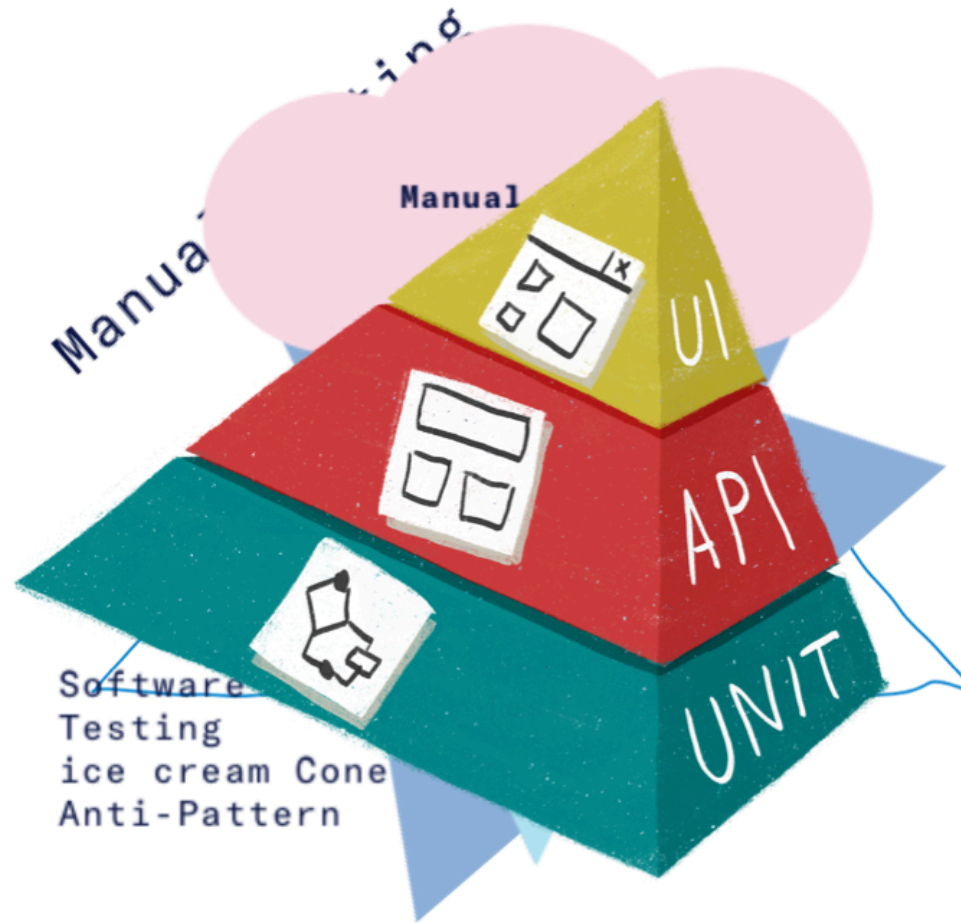
What is the right testing level against production systems? #2

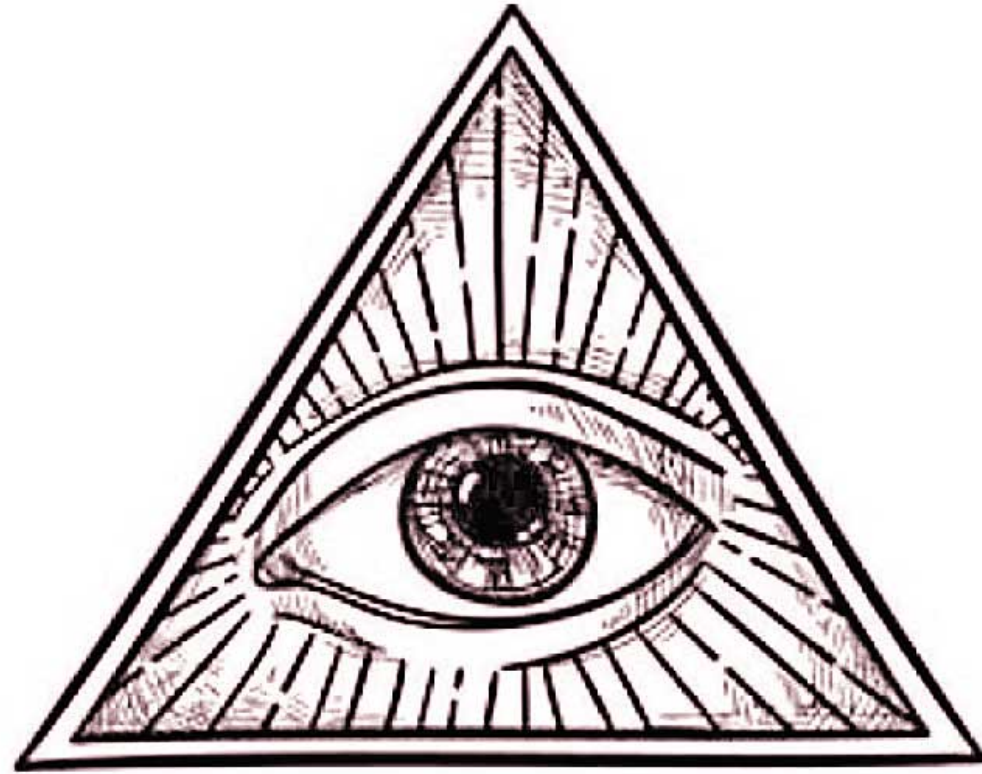


What is the right testing level against production systems? #2

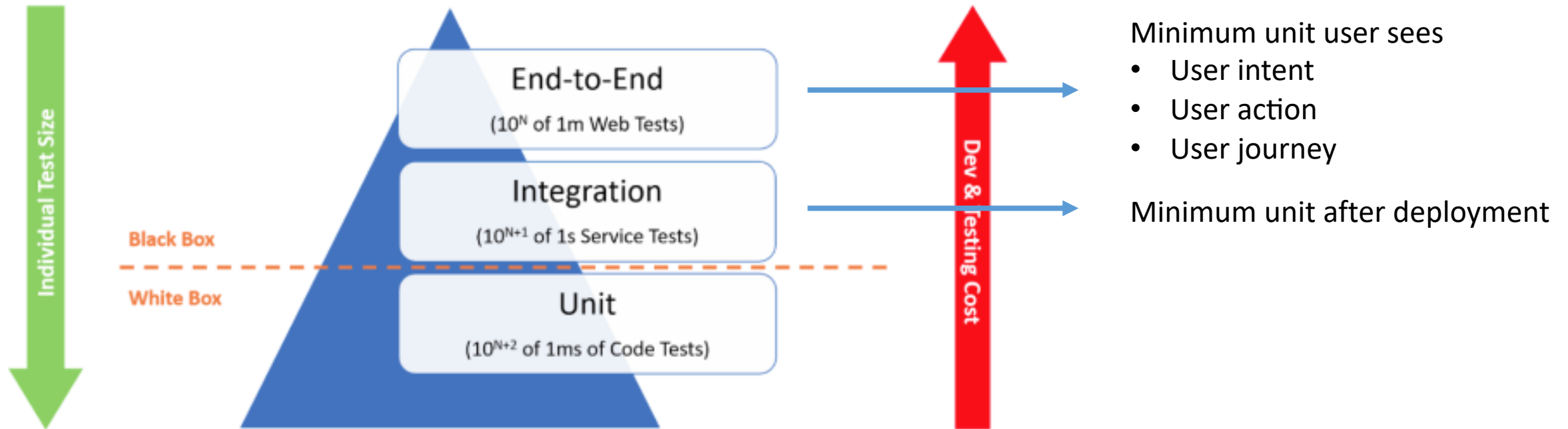


What is the right testing level against production systems? #2



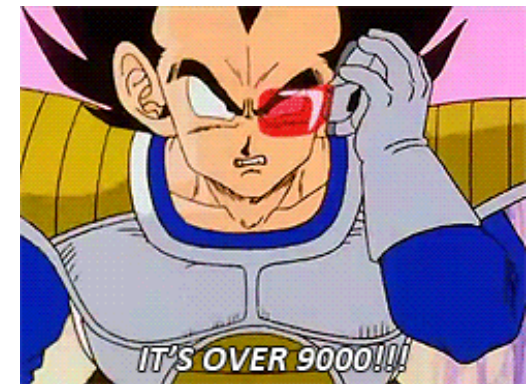


What is the right testing level against production systems? #2



How can Javascript/NodeJS help with that? #3

- More than 1 million connected machines performing automated actions like scheduled cleans
- Users all over the world
 - actions as simple as renaming your cleaning robot from 'Dyson 360 Eye' to 'MegaSucker 9000'
- Fast
- Easy to run
- Extendable
- Ability to write BDD tests



How can Javascript/NodeJS help with that? #3

- Combination of:
 - CucumberJS
 - Request library
 - Assert library
 - Lots of single steps defining calls to APIs and data transformations
 - Mocking machine (virtual machine)
 - Mocking user (virtual user)
 - But going to the cloud and back
 - Grouped together under user journeys or scenarios

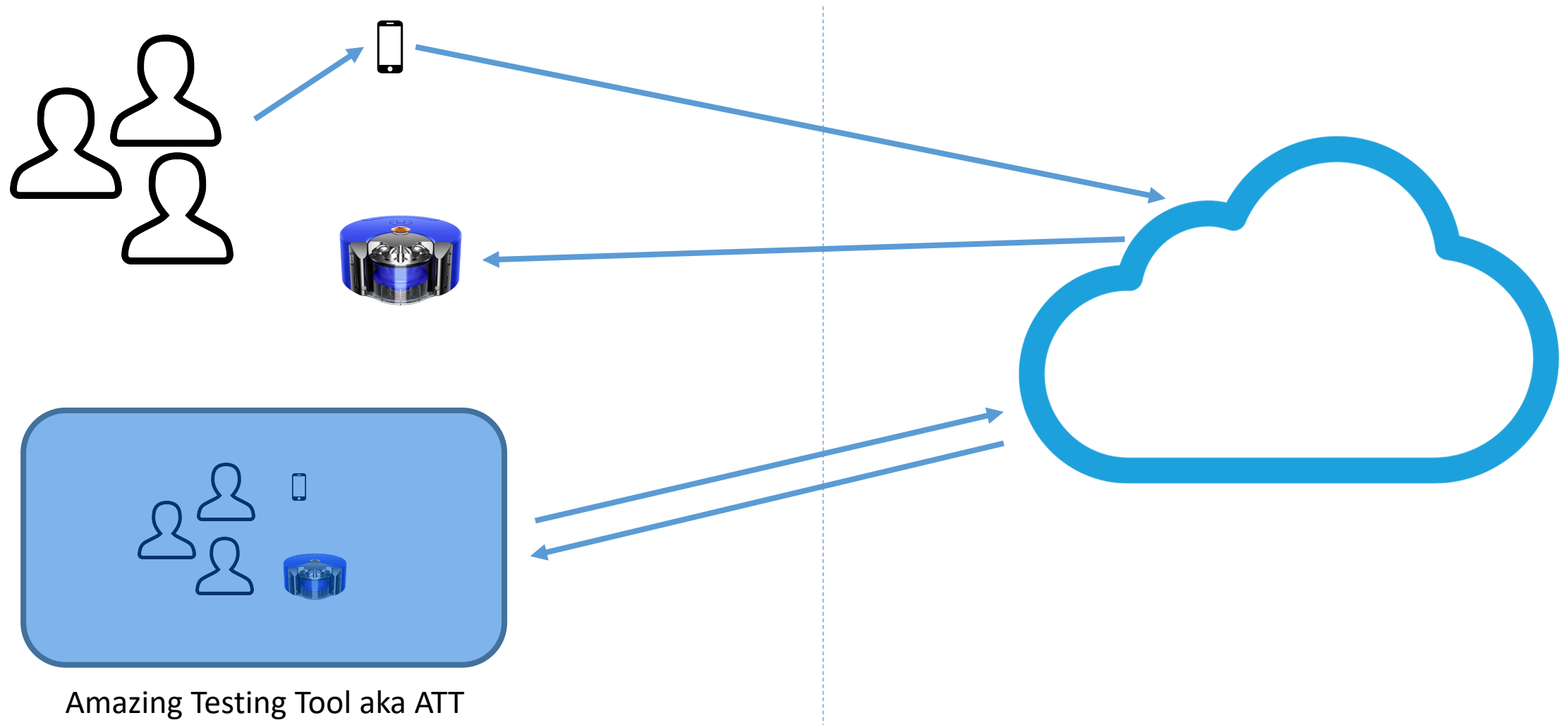


How can Javascript/NodeJS help with that? #3



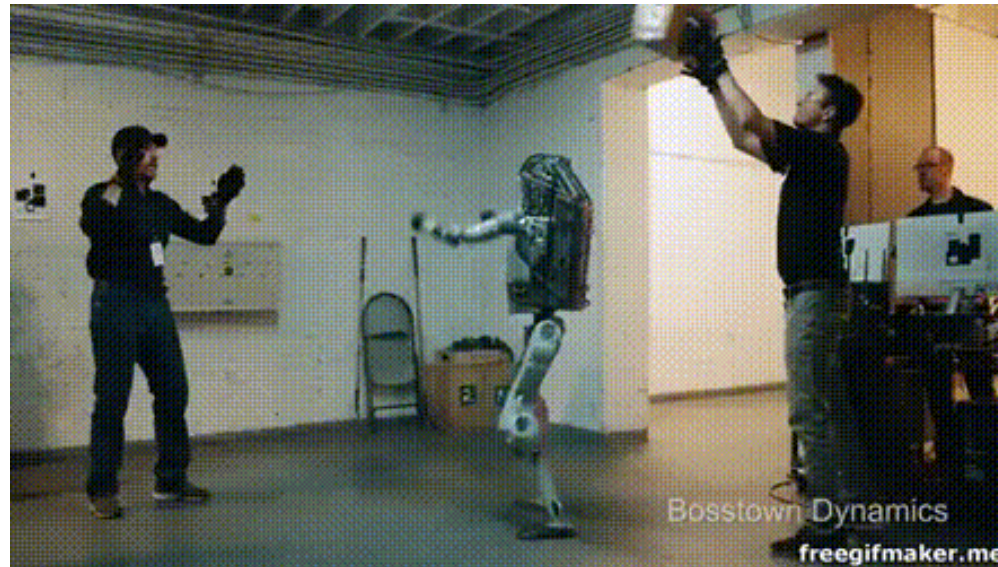
- Given an existing user with this configuration
- And the user logs in
- And the user changes robot name to 'UltraSucker9000'
- Then the robot name has changed to 'UltraSucker9000'

How can Javascript/NodeJS help with that? #3



How avoid disrupting real users? #4

- If your test is too close to real users and real actions...

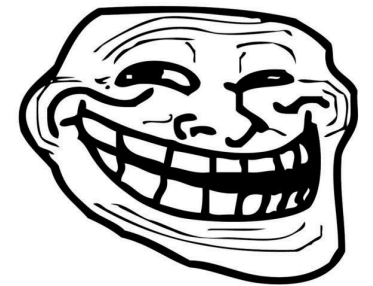


How avoid disrupting real users? #4

- Spacing these end to end tests
 - The more often you run them the faster you will notice an issue
 - Noise
 - Slowing down your production systems
 - Balance
- Virtual users registered in production like real users with their own configuration

How to keep your tests out of statistics? #5

- But testing in production can also skew some of the metrics you record like:
 - Number of users interacting with your system
 - API usage
 - Error count
- Even mixed logs
 - How to discern what was part of a test or part of a real interaction?



problem?

How to keep your tests out of statistics? #5

- Correlation id

e2effff-0000-0000-0000-000000000000

- User agent

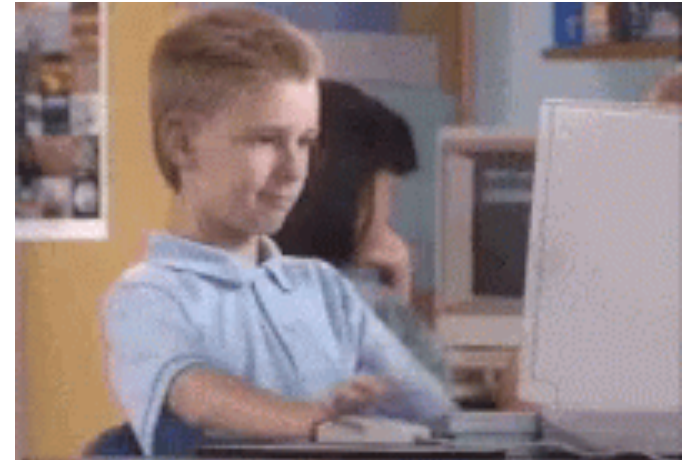
AppId/AppVersionId (Language=LanguageNameAndOptionallyVersion)

- Custom HTTP headers

TestToolVersion=1.1.2

Cleanup? #6

- Running tests in production 👍



Cleanup? #6

- But... what about the all the fake data they generated?



Cleanup? #6

- Automatically clean as much testing items as possible after each test
- Helps starting tests from a clean state
- Avoids cluttering your production databases
- Avoids exhausting other limited resources
- Avoids making real data more difficult to retrieve and search



Recap

- Testing in production is good and necessary
- Think like an user, act like an user
- CucumberJS is a good start
- Think about your system's capacity
- Mark test intents to differentiate them from real user interactions
- Clear your test data and reset connections/status after each test

Thank you (no applause yet)



One more thing

THE FINAL

BIT-BORING-TALK-BUT-WILL-SMILE-FOR-THE-GRAM

SELFIE WITH ATTENDEES!

and you?

are you brave enough to test in production?

and you?
are you brave enough to test in production?

(applause)