Twitter Thread by Jack Ellis

Jack Ellis
@JackEllis

We just launched a fun little tool called Phantom Analyzer. It's a 100% serverless tool that scans websites for hidden tracking pixels.

I want to talk about how we built it ■

The idea came about early this year after @mijustin gave us an idea about badges/certification, which developed into "what if we could scan websites for Google Analytics?!". But we left the idea in Basecamp for many months.

Fast forward to Halloween, we're thinking about fun ideas we could do to entertain people. We discussed "Phantom Analytics" in response to people getting confused with our product name, and had various ideas. And then we landed on the URL analyzer idea.

Once we'd finalized the spec, Paul got to work on our Halloween themed product and then coded up the HTML/CSS for it all. I then took it, put it into a Laravel application. Nice and easy.

Off the bat, I already knew the base stack I was using.

> Laravel Vapor
> ChipperCI for deployment
> SQS for queues
> DynamoDB for the database

We went with DynamoDB as we don't want to worry about our database scaling!

So with our infrastructure known, we had a few challenges left to solve:

> How will we scan websites for tracking pixels?
> How will we utilize the queue and check the job is done?
> How will we validate the URL?

For scanning websites, the first thing I did was write out a complex, multi-level regex matching, guzzle executing scanner. But the problem was that it didn't automatically run the javascript, which often includes additional requests, so the results weren't accurate.

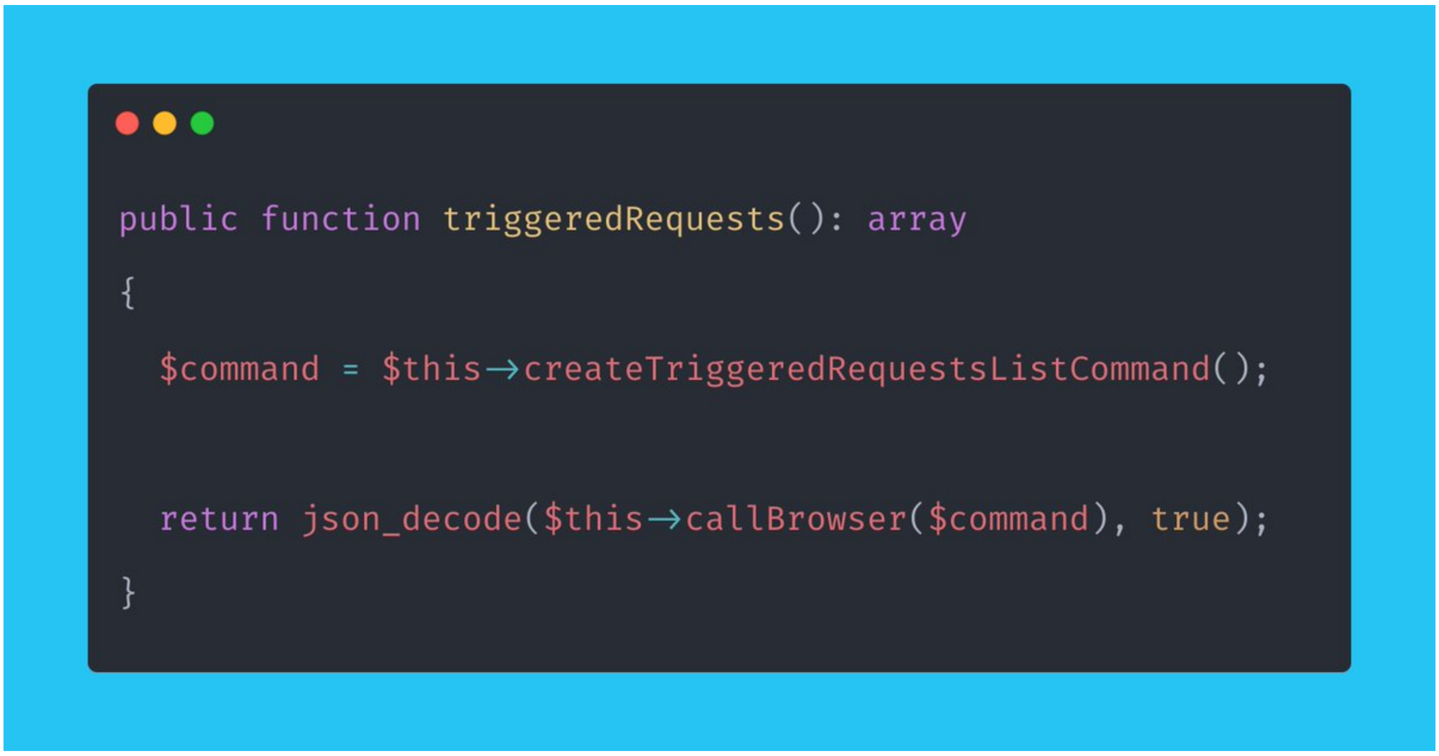
After spending a long time on that, I was searching for website crawlers and came across Puppeteer, which is a headless Chrome Node.js API. I then searched for how to get it running on Laravel Vapor and saw that someone had already solved that challenge!

I then spent 8 hours trying to start from scratch with Puppeteer, copying from @spatie_be's Browsershot code, but I just couldn't get it working. I went to bed, woke up the next morning, and decided that I'd start with Browsershot and simply modify it to what I needed.

I woke up the next day, and within 15 minutes I get a screenshot generated on Laravel Vapor, hooray! I then start to modify Browsershot…

Wait a minute...

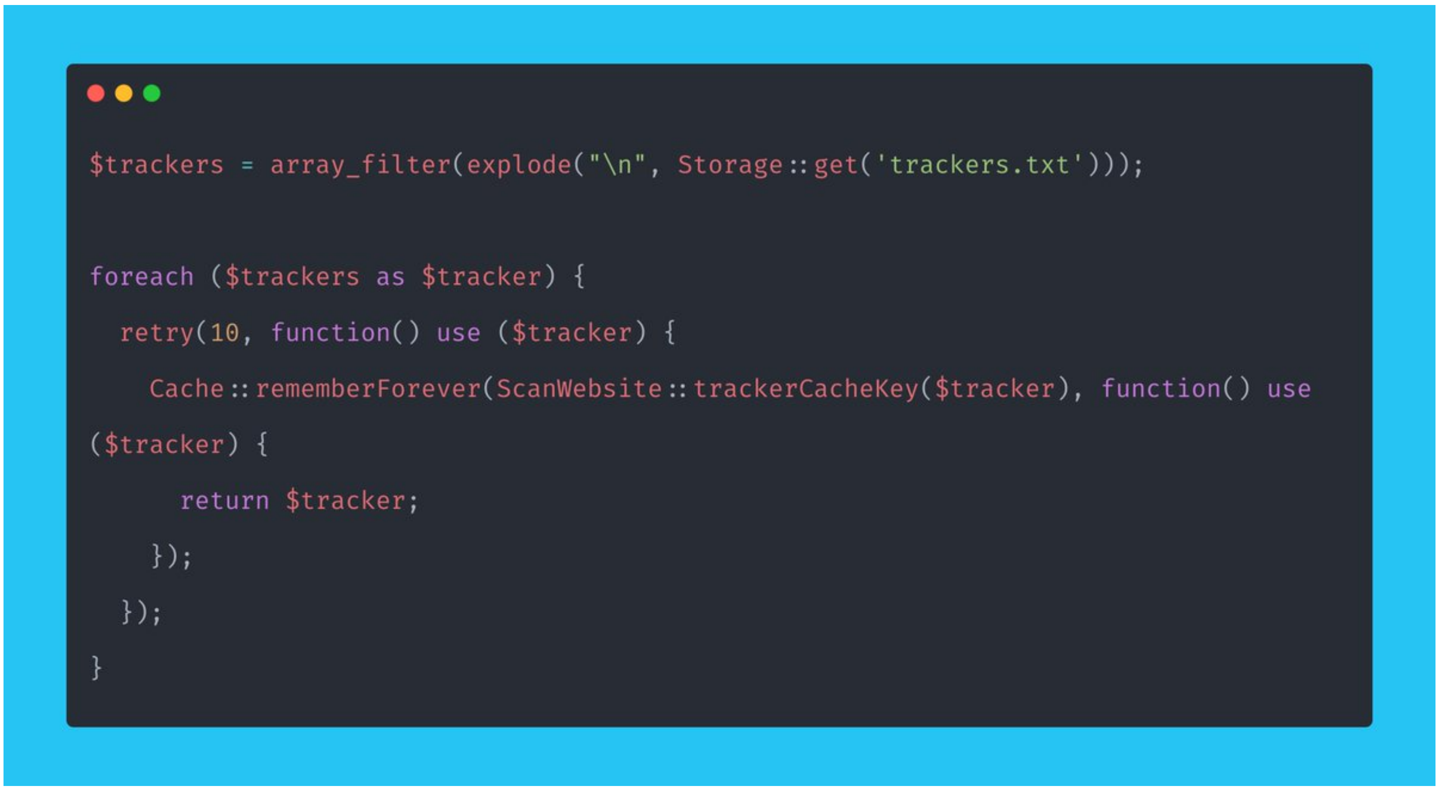
Yes, out of the box, Browsershot already had what I needed. Are you kidding me?



```php
public function triggeredRequests(): array
{
  $command = $this->createTriggeredRequestsListCommand();

  return json_decode($this->callBrowser($command), true);
}
```

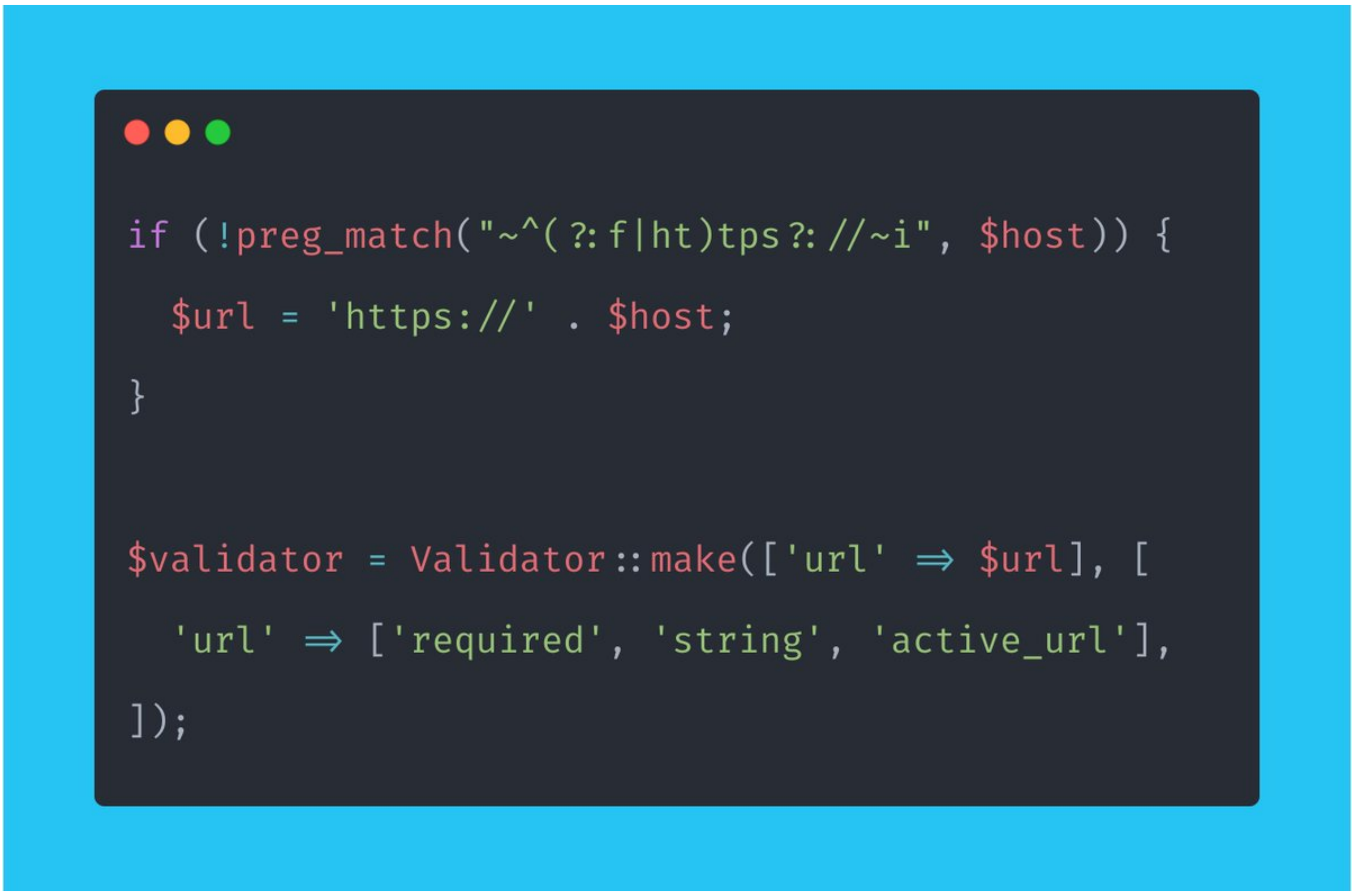
So I modified my job and had it all working within minutes. Browsershot returns a list of network requests when loading a web page and returns them, bloody perfect. I then simply compared them against a list of around 10,000 known third party trackers that we had.

The next step was working out how we would get permanent storage in DynamoDB without bringing in anything extra. I wanted to keep it simple. So with DynamoDB as the driver, and "resources" as the storage root, I wrote a command that cached the tracking pixels indefinitely.



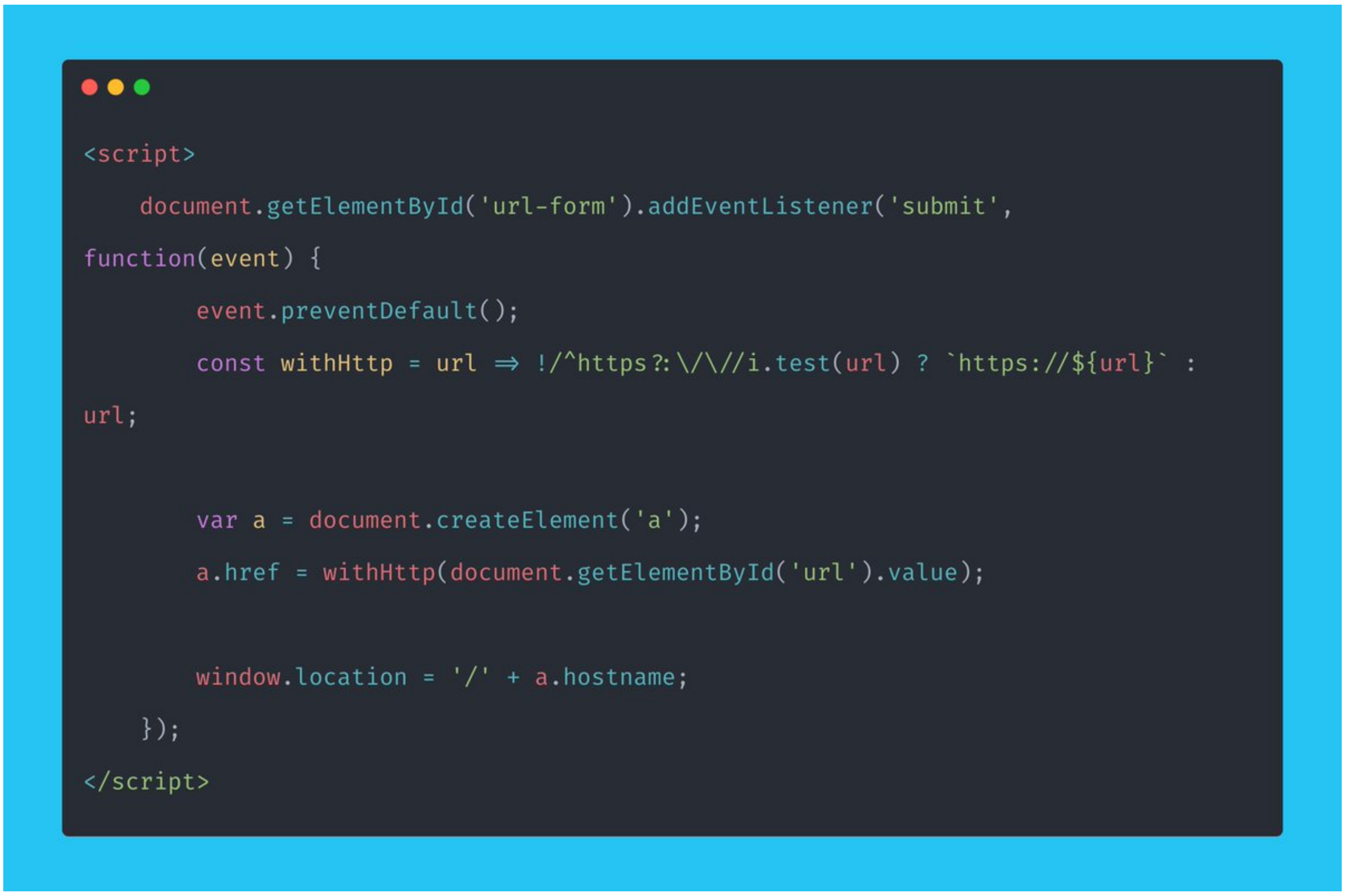
```php
$trackers = array_filter(explode("\n", Storage::get('trackers.txt')));


foreach ($trackers as $tracker) {
  retry(10, function() use ($tracker) {
    Cache::rememberForever(ScanWebsite::trackerCacheKey($tracker), function() use
($tracker) {
      return $tracker;
    });
  });
}
```

One of the initial concerns I had was regarding security. We would be passing user input to the command line and that wouldn't be safe. I spoke with @marcelpociot and he gave me some great advice, and I added in some validation. The active_url rule is fantastic.



```php
if (!preg_match("~^(?:f|ht)tps?://~i", $host)) {
  $url = 'https://' . $host;
}


$validator = Validator::make(['url' => $url], [
  'url' => ['required', 'string', 'active_url'],
]);
```

I also wanted to have a way so that if a user entered a full URL (e.g. https://t.co/66d4eLDaOu) and not just "https://t.co/GA31muKcta", it still redirected them to the correct results page. Especially since our "tidying up" was opinionated. So we ran this code.



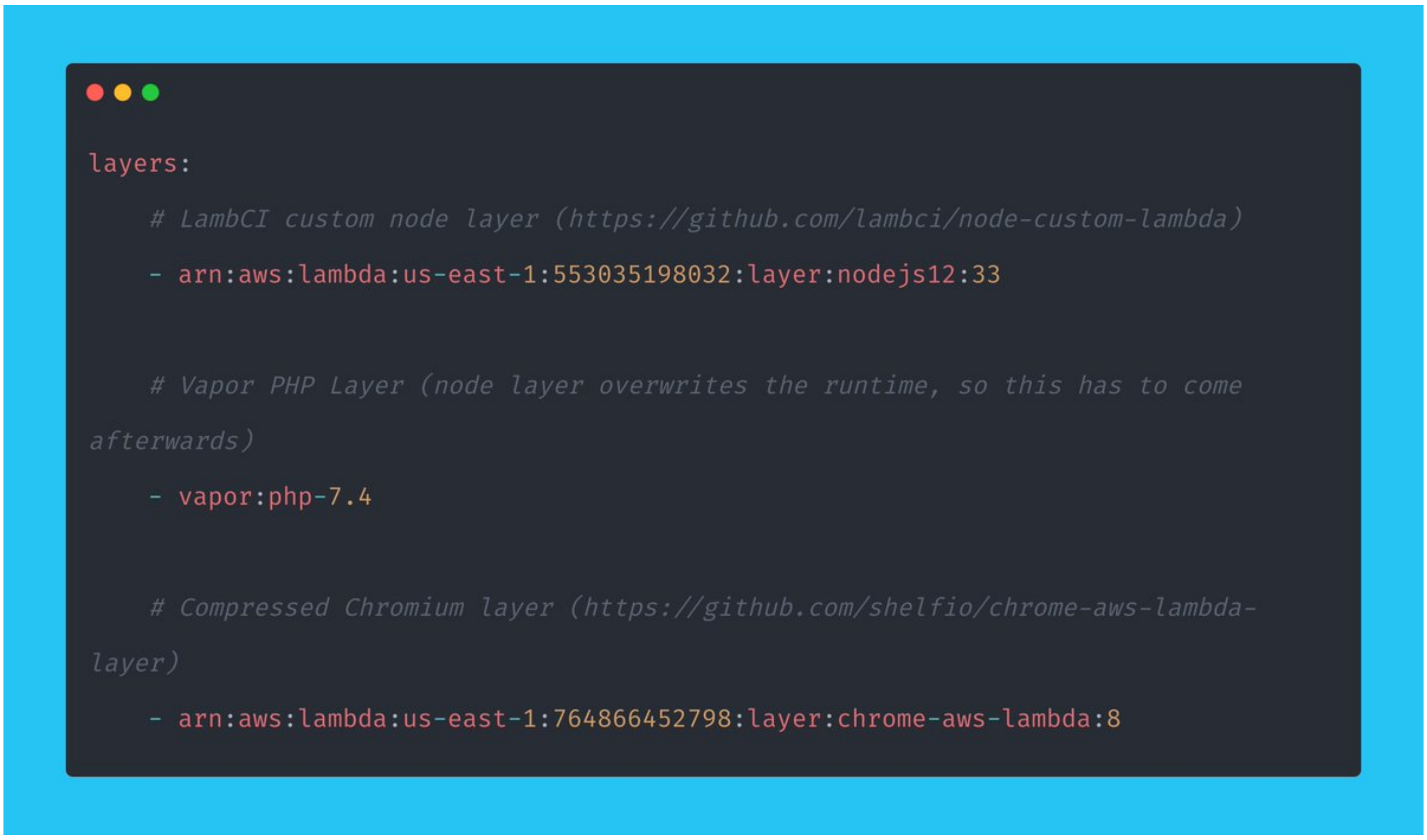
```
<script>
    document.getElementById('url-form').addEventListener('submit',
function(event) {
        event.preventDefault();
        const withHttp = url => !/^https?:\/\//i.test(url) ? `https://${url}` :
url;

        var a = document.createElement('a');
        a.href = withHttp(document.getElementById('url').value);

        window.location = '/' + a.hostname;
    });
</script>
```

We then had to think about how to configure our Vapor app, and I went with the following settings

> 1024MB of RAM
> 2048 of RAM for the queue (could likely reduce!)
> Warm of 500
> CLI Timeout of 180 seconds

Those settings all worked nicely.

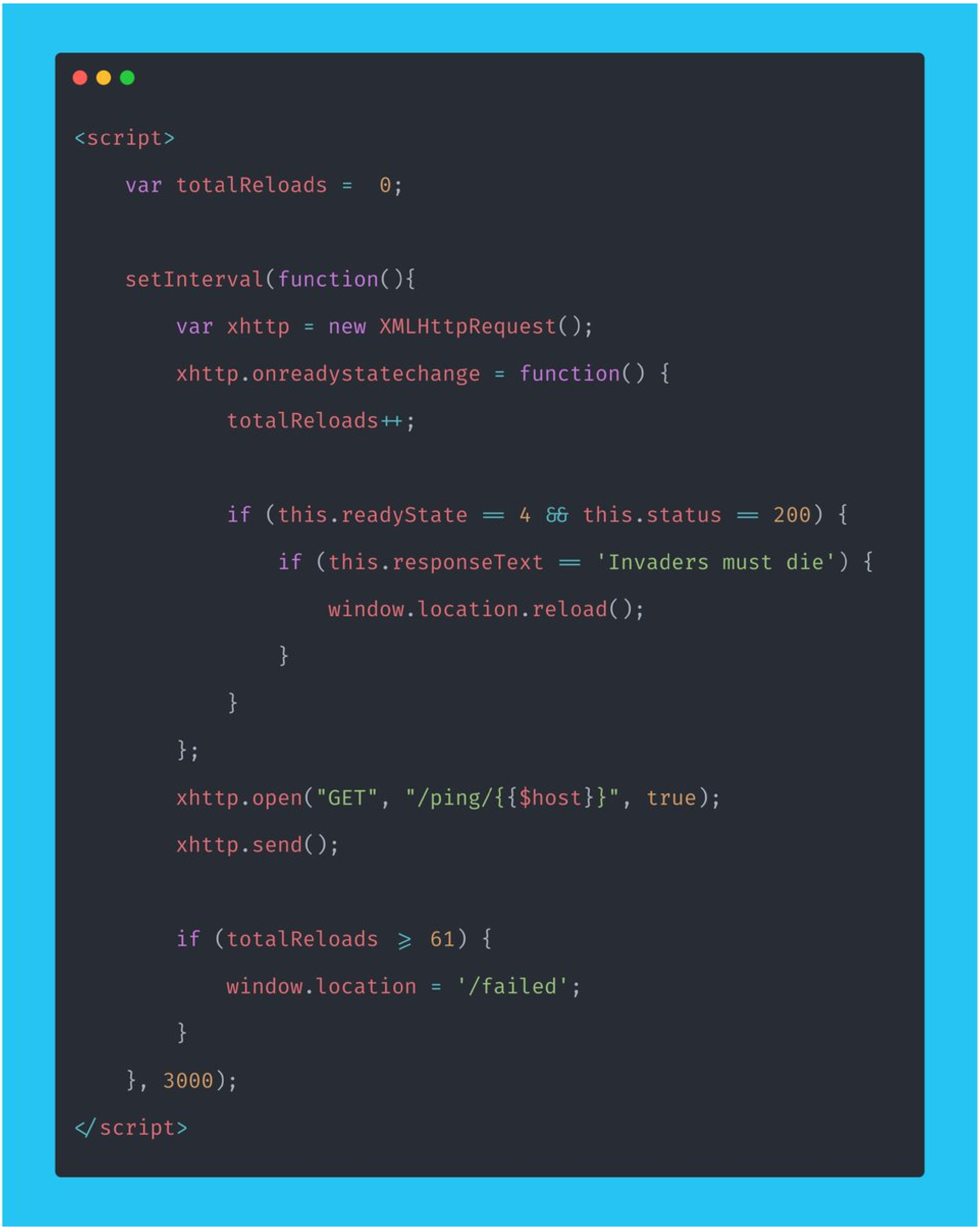
For our Vapor layers, we ran it like this. Very cool. My first time using Layers. Incredible work by the Vapor team (@themsaid @taylorotwell @enunomaduro).



```
layers:
    # LambCI custom node layer (https://github.com/lambci/node-custom-lambda)
    - arn:aws:lambda:us-east-1:553035198032:layer:nodejs12:33

    # Vapor PHP Layer (node layer overwrites the runtime, so this has to come
afterwards)
    - vapor:php-7.4

    # Compressed Chromium layer (https://github.com/shelfio/chrome-aws-lambda-
layer)
    - arn:aws:lambda:us-east-1:764866452798:layer:chrome-aws-lambda:8
```

For the "is it ready?" check, I debated using a UUID but I decided that we might have multiple users trying a website at the same time, and they should benefit from the same cache entry (we cache results for 5 minutes).

So for the ping, we went super old school. Interval and redirect when done. Very effective. And when it reloaded the page, it would hit the cache, see the entry and display it.



```
<script>
    var totalReloads =  0;

    setInterval(function(){
        var xhttp = new XMLHttpRequest();
        xhttp.onreadystatechange = function() {
            totalReloads++;

            if (this.readyState == 4 && this.status == 200) {
                if (this.responseText == 'Invaders must die') {
                    window.location.reload();
                }
            }
        };
        xhttp.open("GET", "/ping/{{$host}}", true);
        xhttp.send();

        if (totalReloads >= 61) {
            window.location = '/failed';
        }
    }, 3000);
</script>
```

All in all, this was a fun project to build. I love working with Paul. The only design addition I made was the bats & fade, Paul did everything else. Very grateful for that ■

I am still in awe over how quickly we deployed this with Vapor. I'm not kidding, it was all coded up and we just created it in the UI, deployed it and we were done. Remarkable experience. Infinite scale without any server work ■

Hope you all enjoy Phantom Analyzer!