CASE STUDY

About the Client

A global digital business transformation company that has spent nearly three decades utilizing the disruptive power of technology and ingenuity to help digitally enable their clients' business.

The Problem

The company ran the customer's websites. The end client was disappointed in the amount of time the websites took to load as well as the overall page size. The previous architecture relied on Grunt. Grunt is a JavaScript task runner, a tool used to automatically perform frequent tasks such as minification, compilation, unit testing, and linting. At the time, we recommend and implemented Gulp which proved to be much faster. Today, we would recommend Webpack.

About MobiCycle

We hold certifications in DevOps. We leverage our expertise in CI/CD pipelines. We run unit tests, linting, deploy to the staging environment on merge , etc. We are comfortable on AWS, GCP and Azure.

WEBPAGES LOADED 70% FASTER

The Solutions

The core component was a front-end build pipeline written in Node.js. Developers would run it on their local machines as well in the cloud to turn the client's custom framework into minified, efficient JavaScript. Business users would then validate the updates in the staging environment before releasing to production.

The pipeline loaded the latest version of the code (or the version on the developer's local machine) then ran "npm ci" to install the most efficient versions of the dependencies.

Any failed builds could easily be discarded and logged locally or in AWS for further investigation. As we ran blue-green deployment, the websites would always be live even if there was a problem with the build.

The build pipeline was integrated into a wider pipeline along with unit tests and regression tests. In the event of those tests failing, the build pipeline could be run repeatedly and still produce the same result.

We used Terraform templates to rollback our infrastructure deployments where necessary. In the event of only one section of the infrastructure failing, we were able to do partial rollbacks as well.

Multiple build tasks were run in parallel to reduce build time. We used Node.js threading capabilities to ensure that tasks and IO resources were properly managed between each workstream.

All our code was checked by the security and testing teams before reaching production. We utilized the tool "npm audit" to audit our dependency graph before deploying. We required to give a clean bill of health before deployment.

Code was stored in BitBucket with the representative tickets in Jira. The pipeline could automatically detect any codebase changes then rebuild the websites.

The build pipeline was for a group of websites with a over 1 million daily visitors. We created monitoring statistics within the pipeline to monitor build times across all sites and noticed significant decreases.

Right before release, we conducted a number of architectural review and training sessions with other teams, explaining how they could upgrade to the new build system.

Summary

We eliminated race conditions in the pipeline which were slowing down the build time. We swapped using a static analyzer for

The code base was written in Node.js. The new pipeline would reduce the build time and page load. We finished the task ahead of schedule.

Based on the successful completion of the initial 3 month project, we were offered other projects and extensions. We delved further into the issues causing the slow load times. .

Summary

2

Multiple contract extensions

3

Asked to do backend development, blockchain, AI

4

Worked with a diverse team from Sales, Marketing, etc

WE ARE AWS CERTIFIED IN DEVOPS