A picture containing clipart

Description automatically generated

Dos Equis Project



# Problem summary

Releasing new code to our customers is a slow process. Currently, we have 3 environments on our path to Production. The two staging environments have high contention. They are less stable, resulting in test failures which add considerable time to our build and deploy procedures and is unnecessary for UI applications. We also don't have an easy way to allow both internal and external users to see different versions of the UI to serve their specific needs.

# Solution summary

By separating the deployment and release of UI applications, we can deploy code changes safely to Production without impacting our customers.

Testing would be done post-deployment against production services and infrastructure. Only once we were happy with that version of the UI, we would be marked as the active version and available to all users.

This solution provides benefit to a multitude of end-users

* Engineers can deploy quickly and test their changes against Production services
* Beta customers can opt in to test our latest and greatest changes
* Sales teams can get an advance look at features before they are rolled out to customers
* Support teams can work with customers to check that the fix to that tricky CSI works for them
* When setting up an evaluation for state adoption, I can relax safe in the knowledge that the locked district I have provided to TX for review will not change during the process

## Technical solution overview

A UI application was created using React. This application is given a different version at build time - for the Hackathon, we used npm scripts to apply semver to the app and store it in a versioned build folder. Versioning using semver is not sufficiently safe for a production environment, using something more opaque like a guid or commit hashes would be better.

We used Git as our file system, but any scalable static serve file system could be used for a production system, e.g. S3.

We also created a simple login application and some user personas.

A NodeJS server was built to handle routing. If a specific version of the application has been requested the server redirects the user to that version. If a version is not passed, the server looks the user up in a simple rules engine to see if they should be directed to a specific version. If no rule for that user is found, they are redirected to the current version.

For the Hackathon, we used a simple JSON file located on the server as our 'database'. For a production system, this would need to be changed to a robust database.

We believe it would be practical to implement this for the future. It would necessitate changing how we build and deploy UI applications. The service or application used to route users would have to be extremely performant - perhaps something like a gateway could be used? Features would need to be implemented to automate deployments and rollback procedures.

We have an open question as to how we would manage to version with a large team of UI engineers, but we believe this to be solvable.

Considerations would have to be given to usages of the INT and Cert environments by other stakeholders.