

Video conference

01/06/2020

Wellington Vieira Menezes

Webmotors S/A

Rua, Gomes de Carvalho, 1996

São Paulo, SP, 04547-905

Overview

With the onset of the coronavirus pandemic and restrictions imposed on commerce by the government, car stores needed to have a video conferencing service to show interested customers the vehicle online, to help increase vehicle sales while stores remain closed by the pandemic.


Objectives

1. Allow the merchant to schedule a video conference from his CRM for a previously registered buyer.
2. Allow buyers to accept and access the video conferencing platform.
3. Notificar o comprador de uma nova video conferência e disponibilizar link de acesso na conferência.
4. Notify the buyer of a new video conference and provide an access link to the conference.
5. Notify seller when buyer accesses video conference if the seller is not accessing video conference
6. Limit the amount of video conference minutes for the seller in the settings.
7. Store the video with audio of the video conference.
8. Allow the seller to access the video of the conference video from the CRM.

Architecture

I was responsible for the team as an architect and the team was composed of:

- 1 architect/scrum master
- 3 backend developers
- 2 frontend react developers
- 2 ios/android frontend developers
- 3 QA
- 1 PO
- 1 Manager



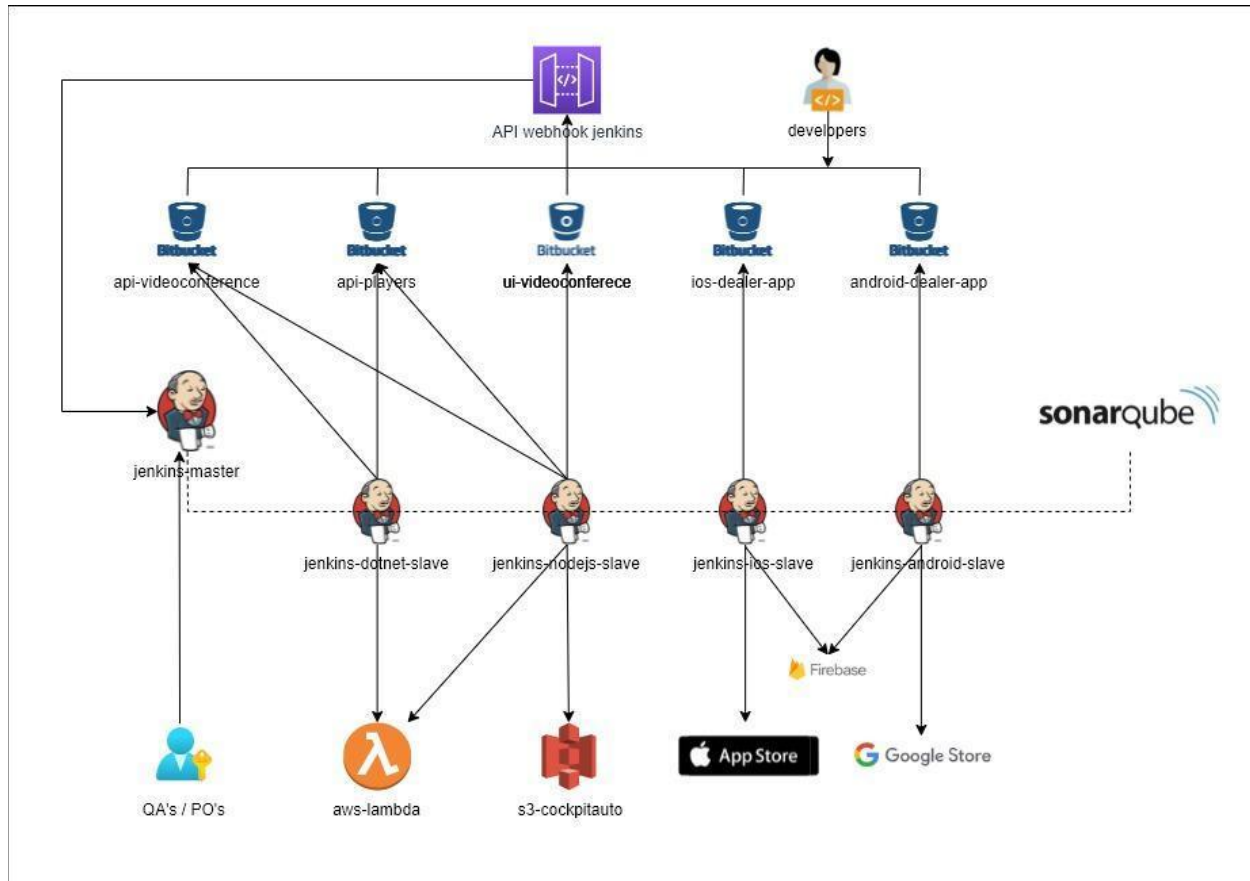
We decided as a team that the project architecture would be ServerLess, if we look at the application design we will see that the current retailers CRM system uses AutoScaling with EC2 on AWS and it is the only part of the system that is not ServerLess the database used was the DynamoDB and was chosen because of the ease of querying by key and sortkey, the cost is due to the use of operations/storage and we separated the project into 3 API's to separate context and security::

1. Public API that makes video-conference access methods available for the merchant and buyer, video-conference completion methods, and Twilio's return webhook for storing the conference video/audio in S3 to make available to the retailer in CRM.
2. Private API that provides database methods and project-specific rules.
3. Players API (private also accessed only from within the VPC) which provides the methods of using the players where the video conference actually takes place ("AWS Chime" and "Twilio") from the client's settings and made flexibly to add other players in the future such as "zoom".

Lambdas were made in Dotnet Core 80% and NodeJS 20%.

The frontEnd was made in react and made available to the customer by S3 as a CloudFront origin, or in the seller's App available on the Apple Store and Google Store.

Deploy




I. FrontEnd React

In AWS CloudFront the default behavior has an origin pointing to S3 and a lambda edge that validates by the CloudFront Distribution ID for which folder gets the HTML, images, and the application bundle.

The separation of environments is done by the creation of AWS CloudFront and the Build is done whenever there is a Pull request merge of the new feature with the master and the master branch always updates the approval folder with the result of the build.

An environment we call blue was made and it simulates a blue/green environment to have a pre-test before sending it to production when running the UI-frontend-blue pipeline, this pipeline is the responsibility of the QA's and the PO that tested and validated the approval environment.



When validating the blue environment with integrated tests made by the QA's team and within the programmed window, we updated the S3 folder for a productive environment with the Html files, images, and application bundle.

II. FrontEnd APP Android/IOS

After merging the Pull request of the new feature branch with the master branch the APP's approval environment update pipeline runs and updates firebase with the build binaries made by Jenkins the QA team accesses the binary for testing and installs the application across multiple devices to do built-in testing of applications.

Shipping to the Google store and Apple store is done with the binaries that are in firebase by the team responsible for the applications.

III. BackEnd lambdas

In AWS Cloudfront, by the environment, it has an origin of `{{url_api_gateway}}/{{environment}}` (hml/blue/prd environments) and the API gateway has in stage variables which is the environment and the link between the resource of the API gateway and the lambda insert the environment of the stage variables.

In lambda, it has 3 aliases one for each environment (hml/blue/prd) the alias hml always points to the last version of the lambda the first deployment of each lambda creates a first version of the lambda and points the environment blue and prd to version 1.

Whenever the master branch is updated after the merge of a Pull request that needs to go through code review and be approved by at least 3 other developers, the lambdas are updated and the approval environment that is associated with the last version of the lambdas is automatically updated.

To update the blue/green environment there is an update pipeline that generates a new version of all lambdas and updates the alias to the generated version, from that point on the blue/green environment is updated for integrated tests by the QA's team to make it available the productive environment.

With the new features updated and the application's basic validation tests done, we can run the production pipeline that only inserts the lambda alias blue version in the prd alias and all platform users start using the new platform version.