FASTLANE IOS BUILD AND AZURE DEVOPS
DOCUMENTATION BY:
SHANZAY GAUHAR

FASTLANE AND AZURE DEVOPs **2**

IOS BUILD - PIPELINE **2**

Fastlane commands and their setup on pipeline 2

IOS BUILD - FASTFILE / GEMFILE **5**

APP_CREATION LANE **5**

create_app_online 5

SIGN LANE: **6**

sync_code_signing 6

update_code_signing_setting 7

Beta_build LANE: **7**

cocoapods 7

create_keychain 8

disable_automatic_code_signing 8

update_project_provisioning && update_project_team 8

gym/build_ios_app 9

Automatic_code_signing 10

increment_build_number_in_plist 10

upload_to_testflight 10

commit_version_bump 11

push_to_git_remote 11

private lane :update_version 11

before/after_all action 11

Project Code Reference: https://github.com/shanzaygauhar/Fastlane 12

Errors I faced: 12

With fastlane, it is recommended to keep most of the actions in the fastfile and only the necessary/command line actions in the pipeline i.e. yaml file.

Add --verbose command in the BASH SCRIPT to track the process and spot error if any.

FASTLANE AND AZURE DEVOPs

I implemented the fastlane using the concept of environment variables. The variables are added in the azure pipeline using the variable tab. The variable corresponding to the certain action would have the prefix of that action name. For instance, if I'm using match and I need to specify the type for match, I would do so by MATCH_TYPE.

Moreover, the secret variables in the pipeline are not accessed by the build process unless they are explicitly mentioned or stated where required.

THE DETAILS OF ANY FASTLANE ACTION CAN BE OBTAINED BY RUNNING ON TERMINAL:

fastlane action action_name

IOS BUILD - PIPELINE

The actions that pipeline does is it specifies the virtual image that is required. The common used are "macOS-latest" or "macOS 10.14".

Fastlane commands and their setup on pipeline

1. **SSH Keys for Fastlane sync_code_signing**

   Fastlane doc link: https://docs.fastlane.tools/actions/sync_code_signing/
   The provisioning profile and certificates are kept in a private repository and match downloads and uses those to sign the ipa. In order to enable the match command to install certificates successfully, we used ssh keys where the repository was hosted on azureDevOps. SSH keys were created manually on desktop and then configured for Azure devOps.

   SSH KEYS
   Commands to generate ssh keys

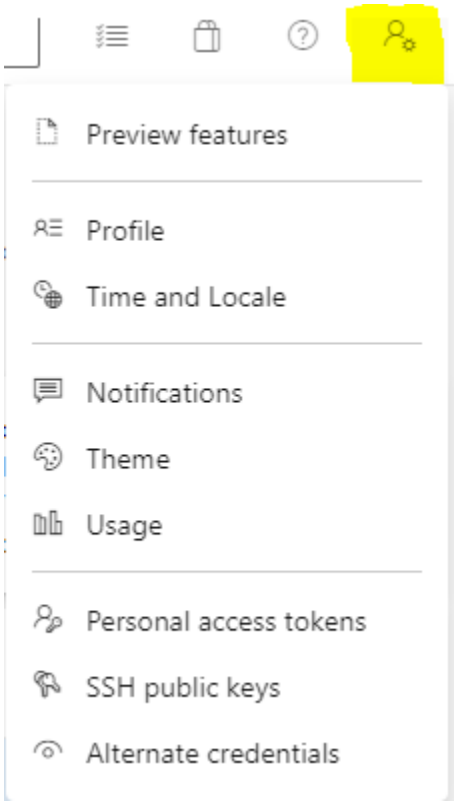
   https://docs.microsoft.com/en-us/azure/devops/repos/git/use-ssh-keys-to-authenticate?view=azure-devops

https://docs.github.com/en/github/authenticating-to-github/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

Ssh-keygen -p keyname → to change the passphrase of the private ssh key

LEAVING PASSPHRASE FOR SSH KEY EMPTY/ SKIPPING THAT STEP LED TO UNEXPECTED ERRORS. DEFINE PASSPHRASE AND STORE IT IN *MATCH_PASSWORD* VARIABLE.

These commands generate private/public key pairs. The private key is uploaded to library/securefiles and then the task InstallSSHKeys is added. The private key is added as following



Choose the SSH public key and then add a new key and in it copy paste the contents of .pub file - i.e corresponding public key.

I had issues with known_host and details of known_host can be found in the following link i.e. https://www.ssh.com/ssh/host-key

The input parameter for the pipeline `InstallSSHKey@0` task includes:

```
knownHostsEntry: //specify the known_hosts that hosts the
repository
sshKeySecureFile: //specify the name of private key uploaded
to secure file
```

In the first run readonly parameter is set to false, and this creates and uploads the certificates to be used later. This step needs to be done by the admin of the developer account and in my case, this step was performed manually through homebrew.

In the later runs, the readonly parameter is set to true because we don't want the match command to create the certificates itself.

DO NOT MANUALLY ADD CERTIFICATES AS IT CAN LEAD TO POTENTIAL ERRORS. This option is available but is often discouraged.

2. **Ruby Version**

   Specify the ruby version to greater than equal to 2.4 to successfully run/compile fastfile.

3. **Node - npm tool**
   Node modules are required to accommodate the built of app and can be done using azure provided task as displayed below:

```
task: NodeAndNpmTool@1
  inputs:
    versionSpec: '10.x'
  displayName: 'Install Node'
- script: npm install
  displayName: 'Install dependencies'
```

4. **Bash Script**
   Add the following lines to BASH Script to ensure latest xcode tools are installed and bundle update ensures that it's dependencies and all the other dependencies that are in our gemfile are up to date.

```
xcode-select --install
bundle update --bundler
```

IOS BUILD - FASTFILE / GEMFILE

“A Gemfile describes the gem dependencies required to execute associated Ruby code. Place the Gemfile in the root of the directory containing the associated code. “

iOS built requires cocoapods and we require fastlane as well. In our gemfile, we will add along with all the dependencies we need for our ruby code:

```
gem(“fastlane”)
gem(“gym”)  //Didn’t try this. Read it on some document
gem(“cocoapods”)
```

`Bundle init` → creates the gemfile.
`Bundle install` → installs the gems listed in gemfile that are associated with ruby.
`Bundle update --bundler` → updates all the gems listed in gemfile along with their dependencies

These above commands can be included on a pipeline in the BASH Script task or manually in the local machine to create gemfile.

APP_CREATION LANE

create_app_online

Fastlane docs: https://docs.fastlane.tools/actions/create_app_online/

First of all, we need to provide the details of our apple account so we can start with the process of fastlane for ios. The following variable allows us to do so:

1. FASTLANE_USER → Apple ID
2. FASTLANE_PASSWORD → Apple account password
3. FASTLANE_TEAM_NAME
4. FASTLANE_ITC_TEAM_NAME

In case, the account uses two way authentication, we may declare a variable `FASTLANE_SESSION` in our pipeline to tackle this situation. In order to generate the value for this variable run the following command on your terminal/ homebrew:

```
fastlane spaceauth -u “yourappleID”
```

This command will generate a value for you which you will assign to FASTLANE_SESSION in your pipeline.

Moreover, this is an alias for the “produce” action of fastlane. According to documentation, “*produce* creates new iOS apps on both the Apple Developer Portal and App Store Connect with the minimum required information.” The parameter I used includes the following:

1. PRODUCE_APP_IDENTIFIER
2. PRODUCE_APP_NAME
3. PRODUCE_PLATFORM
4. PRODUCE_SKU → declare any unique number yourself

This will check if the profile already exists else it’ll create one.

SIGN LANE:

sync_code_signing

Fastlane docs: https://docs.fastlane.tools/actions/sync_code_signing/

It syncs the certificates and profiles across the team via match. The git storage for the match command on azure devOps is discussed above.

Matchfile is generated by command match init. It creates the matchfile which can be configured in the following manner:

```
git_url("SPECIFY SSH LINK HERE")
storage_mode("git")
type("appstore") # The default type, can be: appstore, adhoc, enterprise or development
```

The rest of the parameters can be set or accessed through environment variables on the pipeline. The environment variables include:

1. MATCH_APP_IDENTIFIER
2. MATCH_CLONE_BRANCH_DIRECTLY
3. MATCH_GIT_URL
4. MATCH_GIT_USER_EMAIL
5. MATCH_KEYCHAIN_NAME
6. MATCH_PASSWORD
7. MATCH_READONLY
8. MATCH_TEAM_NAME
9. MATCH_TYPE

The actions can create lane variables that facilitate the communication between actions. Hence in our fastfile, after we call the function sync_code_signing we might call: `mapping=Actions.lane_context[SharedValues::MATCH_PROVISIONING_PROFILE_MAPPING]`

This gives the provisioning profile obtained from the match command so it can be used with other actions like done below with match.

update_code_signing_setting

Fastlane docs: https://docs.fastlane.tools/actions/update_code_signing_settings/

```
update_code_signing_settings(
      profile_name: mapping[ENV['MATCH_APP_IDENTIFIER']]
      use_automatic_signing: false,
      path: "fastlane/ios/Project.xcodeproj"
)
```

This function updates and configures the xcode signing configurations:
The environment variables in the pipeline includes:

1. FL_CODE_SIGN_IDENTITY
2. FL_OUTPUT_DIR
3. FL_PROJECT_SIGNING_BUILD_CONFIGURATIONS
4. FL_PROJECT_SIGNING_PROJECT_PATH
5. FL_PROJECT_SIGNING_TARGETS

profile_name: provisioning profile name to use for code signing. For instance type is development and identifier is xxx.com.xxx, this command "`mapping[ENV['MATCH_APP_IDENTIFIER']]`" uses the mapping variable declared above sets it to match Development xxx.xom.xxx in the XCODE project configuration.

Beta_build LANE:

cocoapods

Fastlane docs: https://docs.fastlane.tools/actions/cocoapods/

```
cocoapods(
        clean_install: true,
        podfile: "./ios"
)
```

This runs pod install and creates the podfile required for iOS built. This function is added in fastfile in either before_all block or in the build lane. Many suggest adding it in before_all in platform: ios but I implemented it in this lane.

create_keychain

Fastlane docs:https://docs.fastlane.tools/actions/create_keychain/

setup_ci() is an alternative method to declare a keychain for a pipeline but in effort to resolve some other error I came across this method. We can declare our own keychain to store match certificates and profiles. The environment variable set into the pipeline includes the following:

1. KEYCHAIN_NAME
2. KEYCHAIN_PASSWORD
3. KEYCHAIN_UNLOCK ---------→ true so we can install and use our certificates
4. KEYCHAIN_LOCK_WHEN_SLEEPS
5. KEYCHAIN_DEFAULT_KEYCHAIN
6. KEYCHAIN_ADD_TO_SEARCH_LIST

In order to use this keychain, set `MATCH_KEYCHAIN_NAME` equal to the value of `KEYCHAIN_NAME` and `MATCH_KEYCHAIN_PASSWORD` equal to `KEYCHAIN_PASSWORD`.

Add the following lines to the Fastfile,

```
unlock_keychain(path:"PATH TO KEYCHAIN", password:"PASSWORD")
```

```
sh "security set-keychain-settings -t 10000 -l
~/Library/Keychains/NAME_OF_KEYCHAIN.keychain"
```

disable_automatic_code_signing

This only requires a path to the xcode project.

This is deprecated version and this can be done with update_code_signing_setting use_automatic_signing set to false

update_project_provisioning && update_project_team

This generally is not used when we use match and gym because it automatically maps but pipeline built was not picking up profile or the team ID so I added the following to my Fastfile after setting up a keychain.

```
update_project_provisioning(
        xcodeproj: "ios/Project.xcodeproj",
        profile: ENV["sigh_com.gamechefs.vs_development_profile-path"],
        build_configuration: "Release"
)
update_project_team(
        path: "ios/Project.xcodeproj",
        teamid: "HQ6CGYLV6L"
)
```

ENV["sigh_com.gamechefs.vs_development_profile-path"] → this environment variable is generated by match command and contains the path to the profile.

Syntax - not sure - sigh_yourBundleID_yourMatchCommandType_profile-path

gym/build_ios_app

Now we may call a gym or build_ios_app command that will generate the required signed .ipa file which we will later upload to our App Store account. The parameters of the gym command or build_ios_app command includes the following:

```
gym(scheme: "Project", workspace: "ios/Project.xcworkspace",export_method: "app-store",output_directory: "ios/output",clean: "true", export_options:{signingStyle: "manual"})
```

Export method basically is the method that is being used to export the archive. Possible values include: app-store, ad-hoc, package, enterprise, development, developer-id.

SharedValue	Description
`SharedValues::IPA_OUTPUT_PATH`	The path to the newly generated ipa file
`SharedValues::PKG_OUTPUT_PATH`	The path to the newly generated pkg file
`SharedValues::DSYM_OUTPUT_PATH`	The path to the dSYM files
`SharedValues::XCODEBUILD_ARCHIVE`	The path to the xcodebuild archive

These sharedValue variables can be utilized similarly as was displayed above for match. This can also be used to communicate an ipa path to upload_to_testflight action or upload_to_app_store or any other action that requires an ipa file.

Automatic_code_signing

Fastlane docs: https://docs.fastlane.tools/actions/automatic_code_signing/

We enable automatic signing after the gym command.

increment_build_number_in_plist

This is one of fastlane plugins that might be used to update the build number. This function is added in fastfile.
Syntax:

increment_build_number_in_plist (xcodeproj:'ios/Project.xcodeproj', target: 'Project')

upload_to_testflight

Fastlane docs: https://docs.fastlane.tools/actions/upload_to_testflight/

This uploads the binary file - .ipa to AppStore Connect for Testflight beta testing.

If you are using *pilot* via the fastlane action, add the following to your `Fastfile`

```
ENV["DELIVER_ITMSTRANSPORTER_ADDITIONAL_UPLOAD_PARAMETERS"] = "-t DAV"
pilot...
```

It is recommended that password does not contain any special characters otherwise unexpected errors might occur.

Application specific passwords may be used to upload. For this inorder to avoid two way authentication or 2FA, these two variables must be properly set - skip_waiting_for_build_processing and apple_id. This password is set in the environment variable: FASTLANE_APPLE_APPLICATION_SPECIFIC_PASSWORD

The documents to perform this step can be located here:
https://docs.fastlane.tools/best-practices/continuous-integration/#application-specific-passwords

The several parameters can be located in the upload_to_testflight fastlane docs. The IPA path can be obtained similar to the method as demonstrated above in the document here https://docs.google.com/document/d/1CQnOwxQtn6f_OvmtDnMq8HkrMzFFgB7kC0v764vw9KY/edit#heading=h.gg278zx2m39a ----> In the link it is using the SharedValue lane variable to obtain path to profile.

commit_version_bump

Fastlane docs: https://docs.fastlane.tools/actions/commit_version_bump/
This command is run after we have incremented the built number. It then creates a version bump commit. It checks for changes in xcodeproj or plist file.

push_to_git_remote

Fastlane docs: https://docs.fastlane.tools/actions/push_to_git_remote/

This command pushes local changes to remote repositories. According to link above,

> Lets you push your local commits to a remote git repo. Useful if you make local changes such as adding a version bump commit (using `commit_version_bump`) or a git tag (using 'add_git_tag') on a CI server, and you want to push those changes back to your canonical/main repo.
> If this is a new branch, use the `set_upstream` option to set the remote branch as upstream.

private lane :update_version

Pre-Condition: Semantic versioning is chosen in the xcode/xcworkspace project.

1.2.3 → the leftmost is major, middle minor, rightmost patch.

This self declared private lane updates the version number by comparing the version number listed on appstore and plist and if they are the same, we increment the minor value else we increment the patch. If both versions are the same then it might be a major update and we increment the minor. If they are not the same versions, we might increment the patch.

```
private lane :update_version do
        app_store_version = get_app_store_version_number(bundle_id:"YOUR BUNDLE ID")
        plist_version = get_version_number_from_plist(xcodeproj:'./ios/Project.xcodeproj')

        if Gem::Version.new(plist_version.to_f) == Gem::Version.new(app_store_version.to_f)
                increment_version_number_in_plist(xcodeproj:'ios/Project.xcodeproj',bump_type:'minor')
        else
                 increment_version_number_in_plist(xcodeproj:'ios/Project.xcodeproj',bump_type:'patch')
```

before/after_all action

1. **before_all**
   ensure_git_status_clean → this is done to make sure that there is no uncommitted change or anything.
   ensure_git_branch → ensures this is the master branch or the branch you need to work on.
   git_pull → executes a simple git pull command to pull together all the changes

Fastlane docs: https://docs.fastlane.tools/actions/ensure_git_status_clean/
https://docs.fastlane.tools/actions/ensure_git_branch/
https://docs.fastlane.tools/actions/git_pull/

2. **after_all**
   reset_git_repo → discard all uncommitted changes

   Fastlane docs: https://docs.fastlane.tools/actions/reset_git_repo/

Project Code Reference: *https://github.com/shanzaygauhar/Fastlane*

Errors I faced:

MATCH TYPE: "DEVELOPMENT" → stuck at `Running script '[CP] Copy Pods Resources'

MATCH TYPE: " APPSTORE" → asked me for a development certificate.

Probable solution → update_project_team