
Design
SDK
DEVCON
SDK

RWDevCon 2018 Vault

By the raywenderlich.com Tutorial Team

Notice of Rights

Notice of Liability

Trademarks

Table of Contents: Overview

Table of Contents: Extended

Prerequisites

Some tutorials and workshops at RWDevCon 2018 have prerequisites.

If you plan on attending any of these tutorials or workshops, **be sure to complete the steps below before attending the tutorial**.

If you don't, you might not be able to follow along with those tutorials.

> **Note**: All talks require **Xcode 9.2** installed, unless specified otherwise (such as in the ARKit tutorial and workshop).

15: App Development Workflow

- Install CocoaPods
- You will need a GitHub account
- You will need to sign up for Fabric: https://get.fabric.io/
- You will need an Apple Developer account and the private key and your distribution certificate installed in the keychain
- You will need to sign up for a Mixpanel account: https://mixpanel.com/

15: App Development Workflow

Building an iOS app is easy. Building a successful one however needs more effort. This session will focus on automating your builds, using continuous integration to test and deploy them, and finally integrating analytics and tracking once yourapp is released to prepare for the next iteration. You will walk away with a toolset for building an efficient app development workflow.

42 App Development Workflow: Demo 1

By Namrata Bandekar

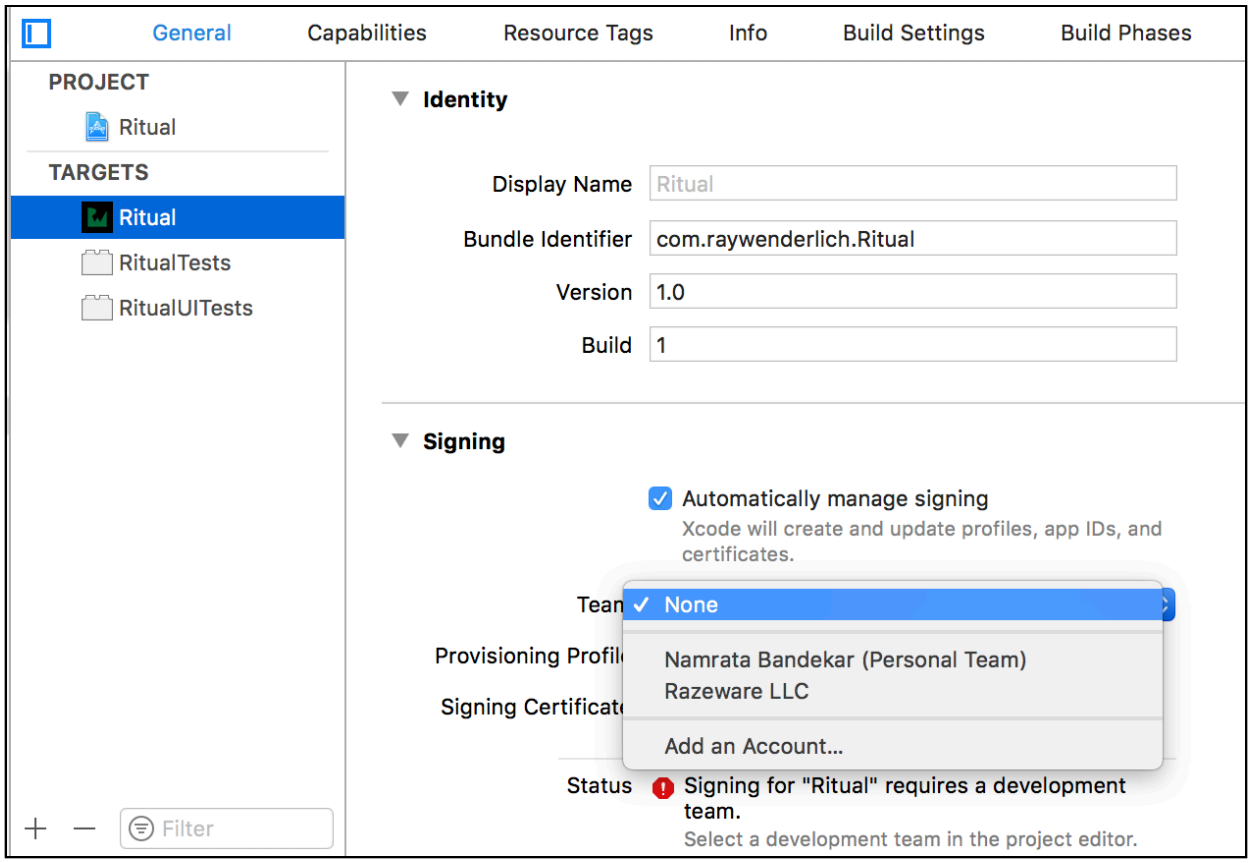
In this demo, you will learn how to write automated tests, and configure an Xcode bot to perform integrations on your project.

The steps here will be explained in the demo, but here are the raw steps in case you miss a step or get stuck.

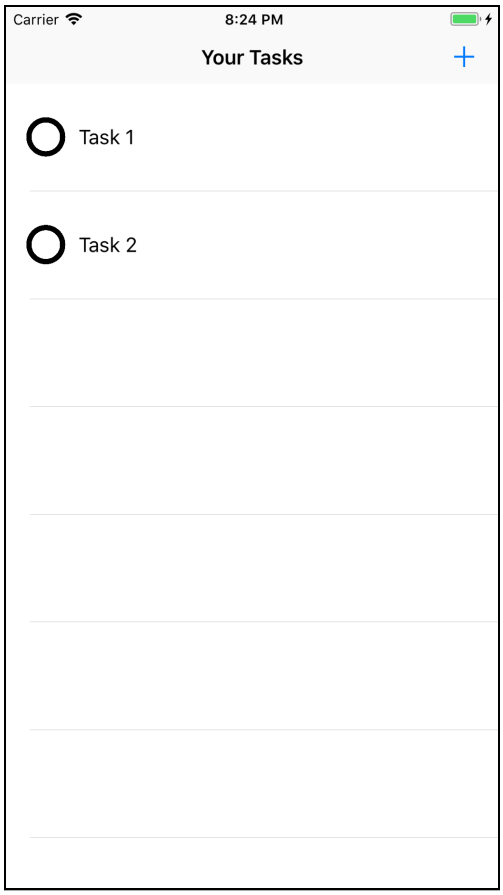
> **Note**: Begin work with the starter project in **Demo1\starter**.

1) Getting Started

Go to the Project Navigator, select Ritual and click on the **General** tab. Select a team under the **Signing** section.



Now build and run the starter project. You will see a screen with a pre-populated list of tasks.



2) Adding Tests

Adding A UI Test

The bot can integrate all your functional and UI tests. Let's add a UI test to your project.

Open **RitualUITests.swift**, and add the following method in the **RitualUITests** class:

```
    func testAddAndCancel() {
      let app = XCUIApplication()
      app.navigationBars["Your Tasks"].buttons["Add"].tap()
      XCTAssert(app.navigationBars["New Task"].exists)
      app.navigationBars["New Task"].buttons["Cancel"].tap()
      XCTAssert(app.navigationBars["Your Tasks"].exists)
    }
```

Here you add a UI test to tap the **+** button on your app's landing page. This pushes the "New Task" view controller on top of the existing view controller in the navigation controller. The test checks this by comparing the titles. Then it selects the **Cancel** button to go back to the "Your Tasks" view controller.

Run the test and make sure it passes.

Adding A Unit Test

Open **RitualTests.swift** and add the following.

```
    func testTaskInitialization() {
      let emptyTask = Task.init(description: "", notes: nil)
      XCTAssertNil(emptyTask)
    }
```

Here you are testing the failable initializer that is defined for your Task class. If the description for the task is empty, the initializer should fail and return nil.

Run the test and ensure it passes.

3) Setting Up Xcode Source Control

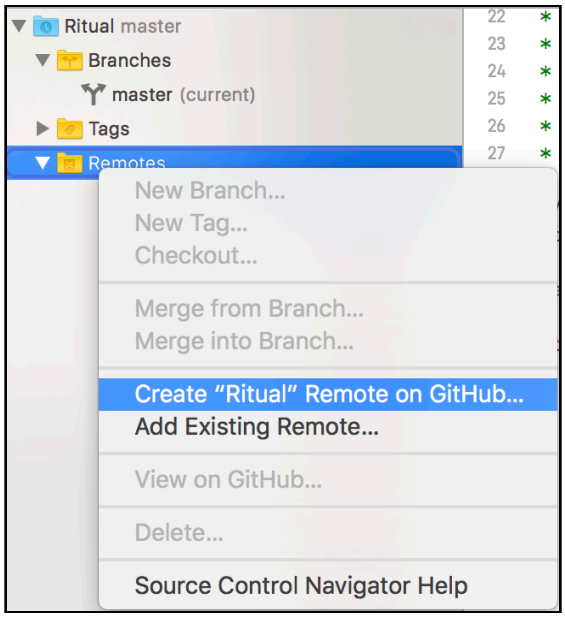
The Xcode Server bot needs access to source code and every time it performs an integration it pulls the latest code from the remote Git repo. This is why you will now hook up your starter project to Xcode's source control.

Navigate to **Source Control > Create Git Repositories...** in the menu. Select your project, if not already selected, and click **Create**. Xcode 9 only supports Git version control.

This automatically adds all your source files to a local Git repository on your workstation. To see your source tree, open the **Source Control Navigator** with **Command+2**. You can view all your branches, tags and remotes here.

> **Note**: If your project already resides in a remote repository, you can use the **Source Control > Clone** feature and pull a local copy on to your development machine. This will also set it up for using Xcode's source control. Xcode 9 only supports remote repositories hosted on GitHub.

To create a remote and push your project, go to the Source Control Navigator and right-click on **Remotes**. Select **Create "Ritual" Remote on GitHub**.



Enter your GitHub account credentials, adjust the options for the remote repo and click **Create**.

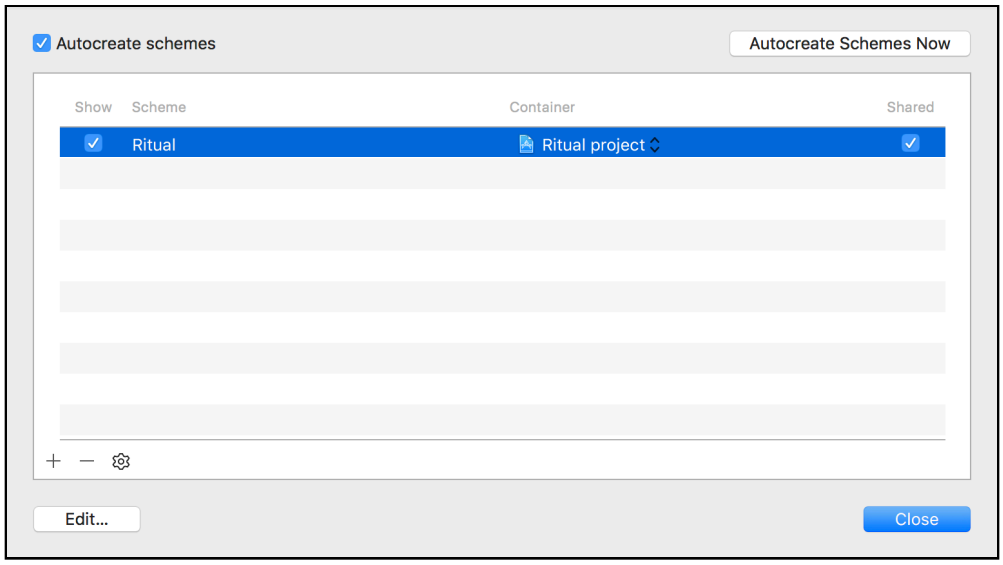
4) Shared Schemes And Xcode Server

Creating A Shared Scheme

A scheme is used by Xcode to figure out which targets to build, which build configuration to use and which executable environment to use when the app is launched. Xcode creates a default scheme when you create your project. This includes settings to perform static analysis, execute tests and build an archive of the app.

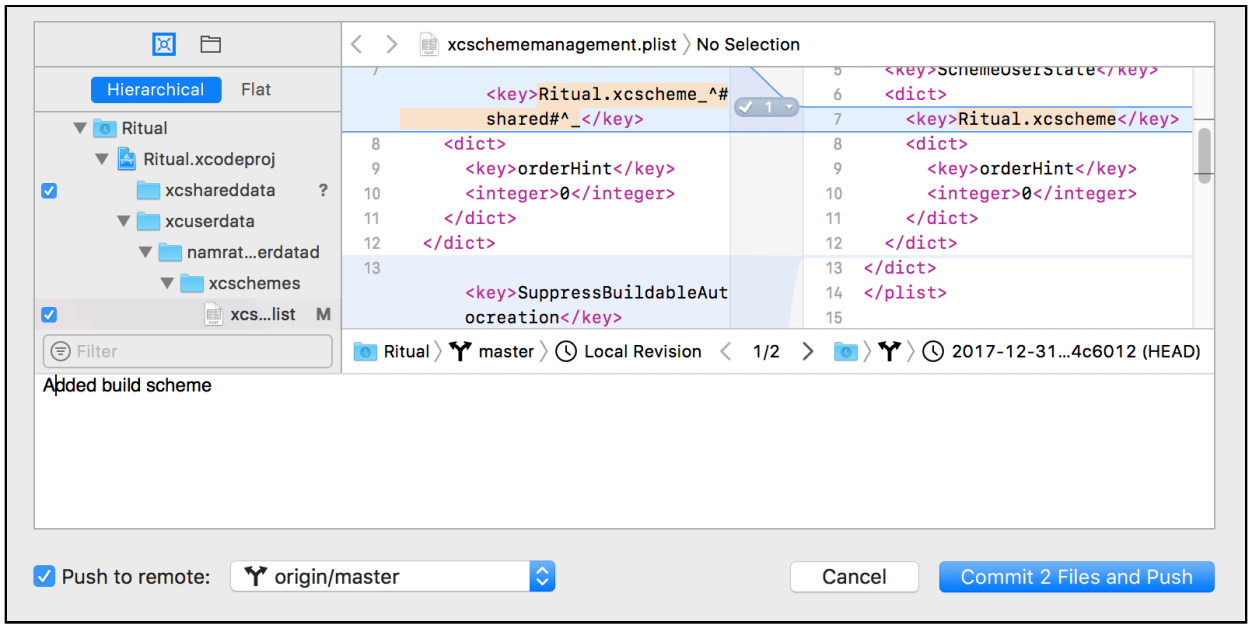
In order for Xcode Server to perform these actions, you need to share the project's scheme by publishing it to your repository.

Go to **Product > Scheme > Manage Schemes**. Select the default Ritual scheme. Check the **Shared** checkbox and click **Close**



Now, commit these changes to remote.

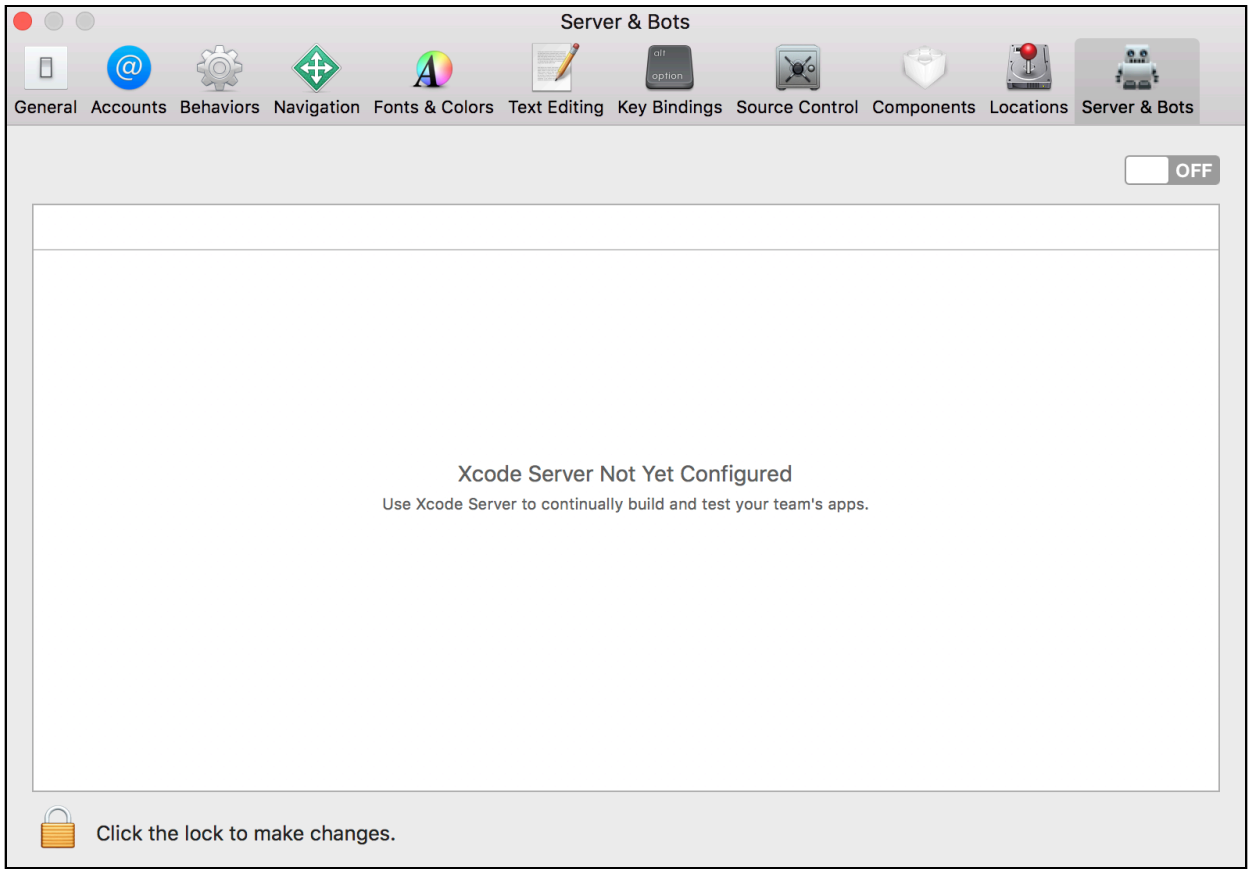
Go to **Source Control > Commit**. Select the **xcshareddata** folder and **xcschememanagement.plist**, add a commit message and check the **Push to remote** box. Now click **Commit 2 Files and Push** button.



This allows you to share the scheme with the build machine.

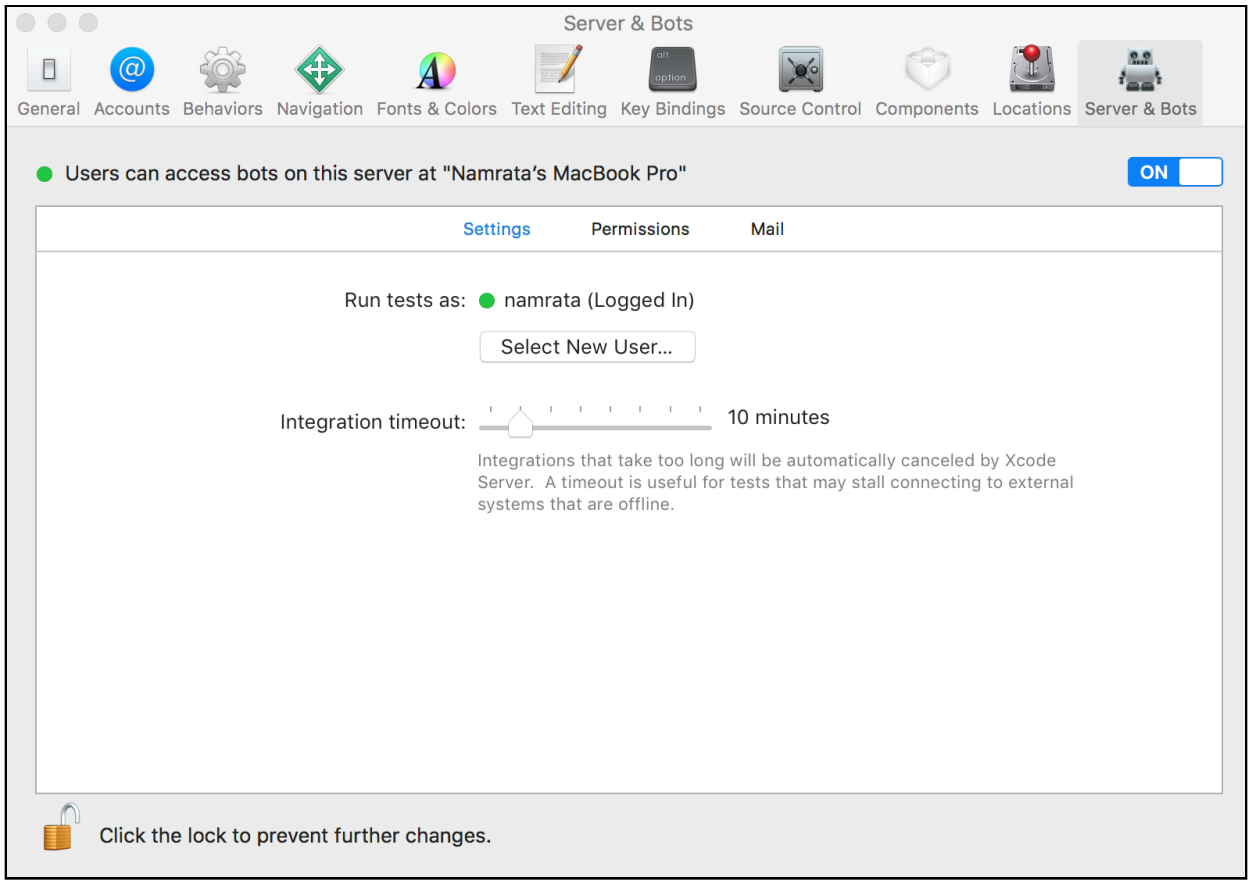
Xcode Server

To create a bot, you first need an Xcode Server set up. Navigate to **Xcode > Preferences...**, select the **Server & Bots** tab on the far right and switch the toggle to **ON**.

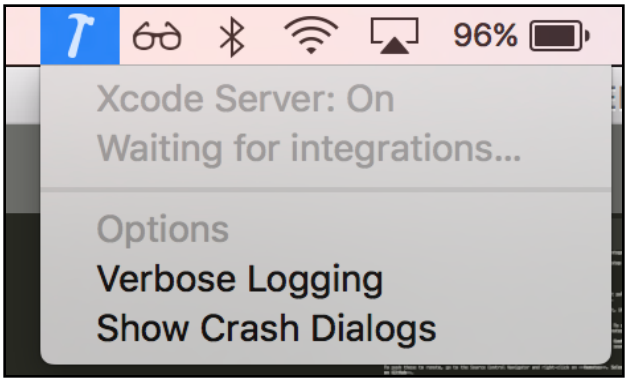


Note: You would normally setup your server on your build machine. However, for the purpose of this demo you will use your personal machine to do both automated builds and development.

Xcode instructs you to use a dedicated, non-administrative user. For this demo, select your current logged in account even if it is an administrator. Xcode then configures the server automatically. You will need to give accessibility permissions to the Xcode Helper. Once it successfully configures, you will see a green dot next to your server name. On this screen you can select your integration timeout. It defaults to ten minutes. You can configure this based on your average build times.

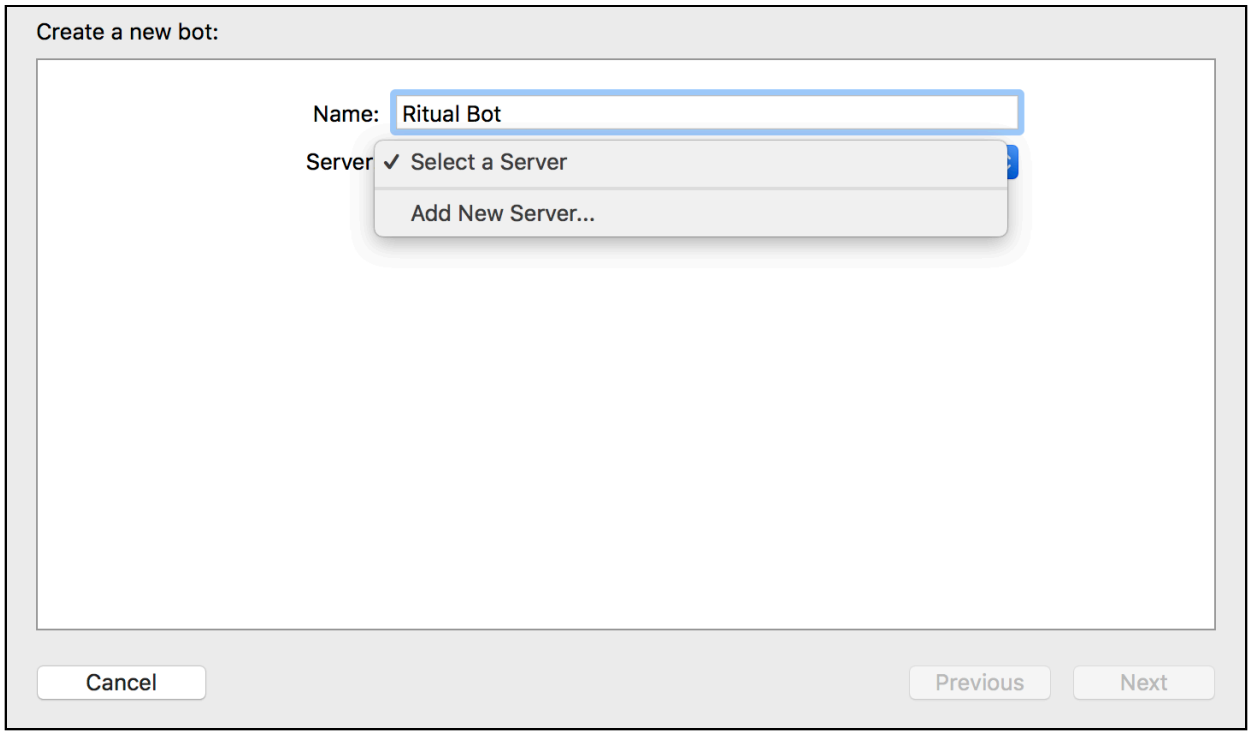


On your build machine's status bar you will see a hammer icon indicating that Xcode Server is running.

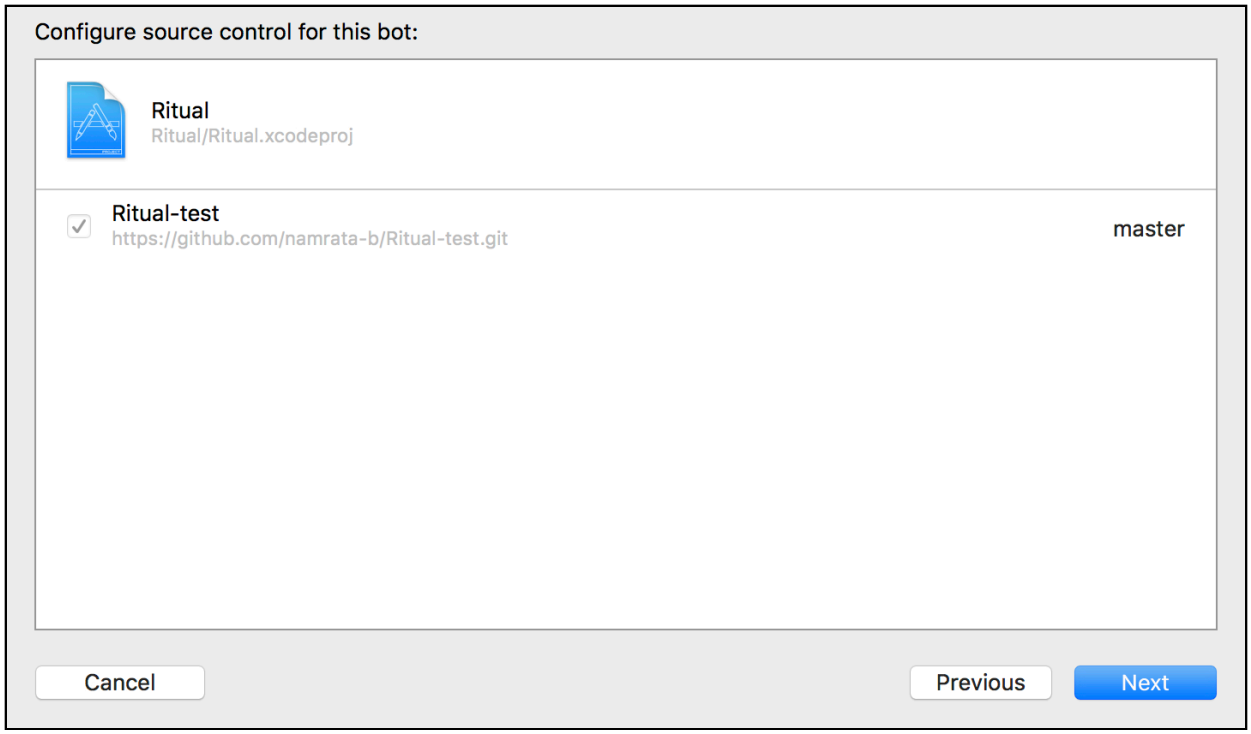


5) Creating A Bot

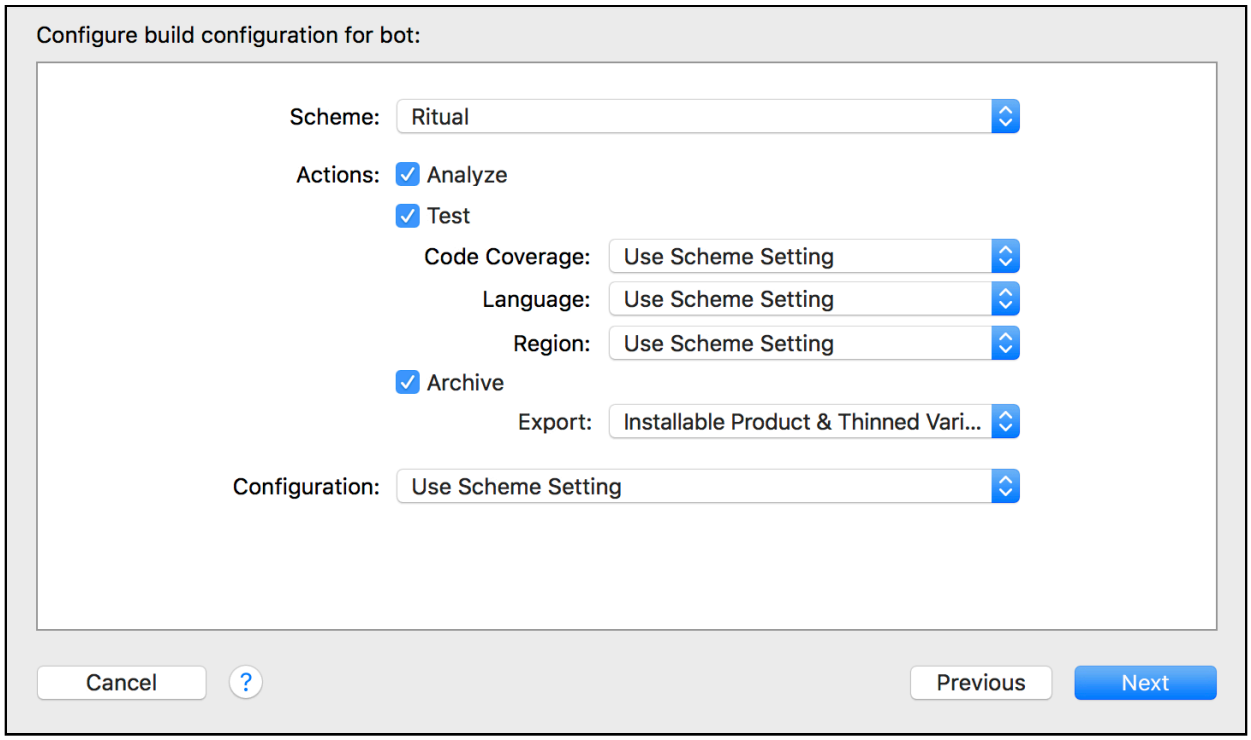
Go to **Product > Create Bot**. Enter the name for the bot and select the server that you just set up in Step 3. You will need to login to the Xcode Server account to add it.



Click **Next**.



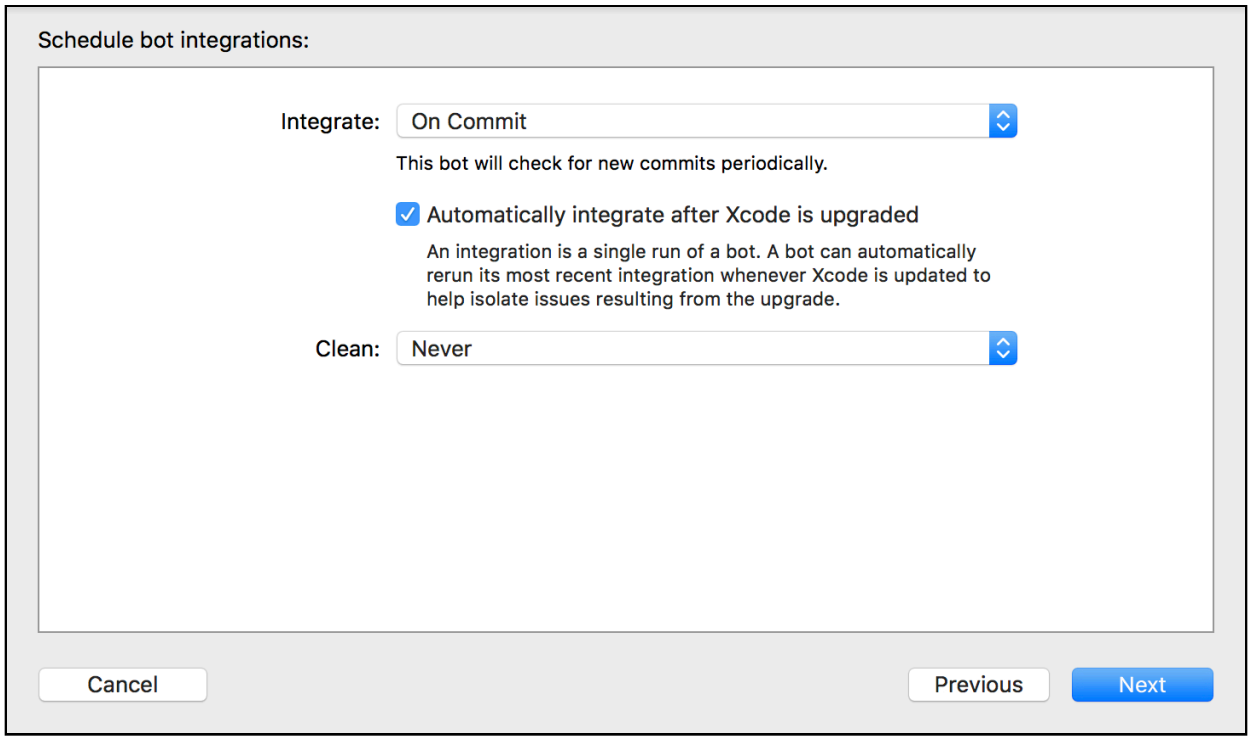
This screen prompts you to configure source control for your bot. In this case, we have already setup the GitHub repo on this machine. If you are using a separate build machine, you will need to login and provide credentials to access your remote repository. Click **Next**.



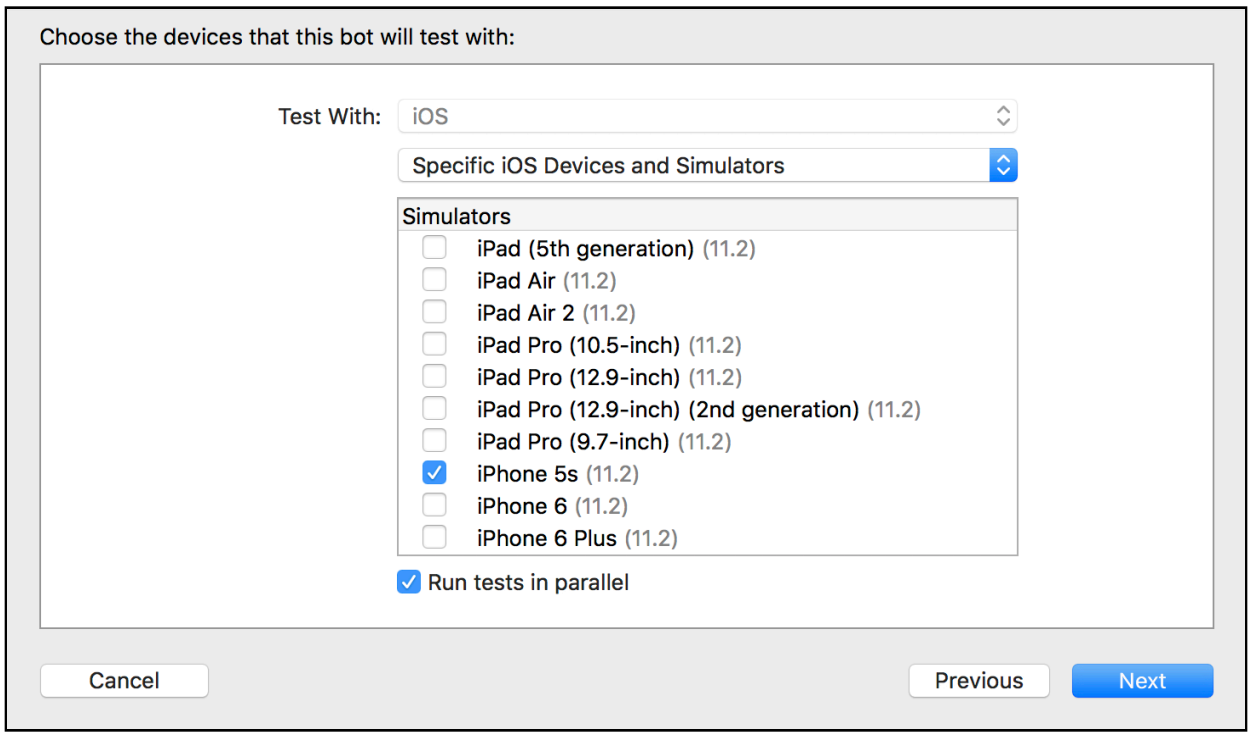
Here you configure your builds for the bot.

- You choose the scheme for your build.
- Select the actions that the integration should perform. You can enable static analysis, testing, and product archiving.
- Select either a debug or release build configuration.

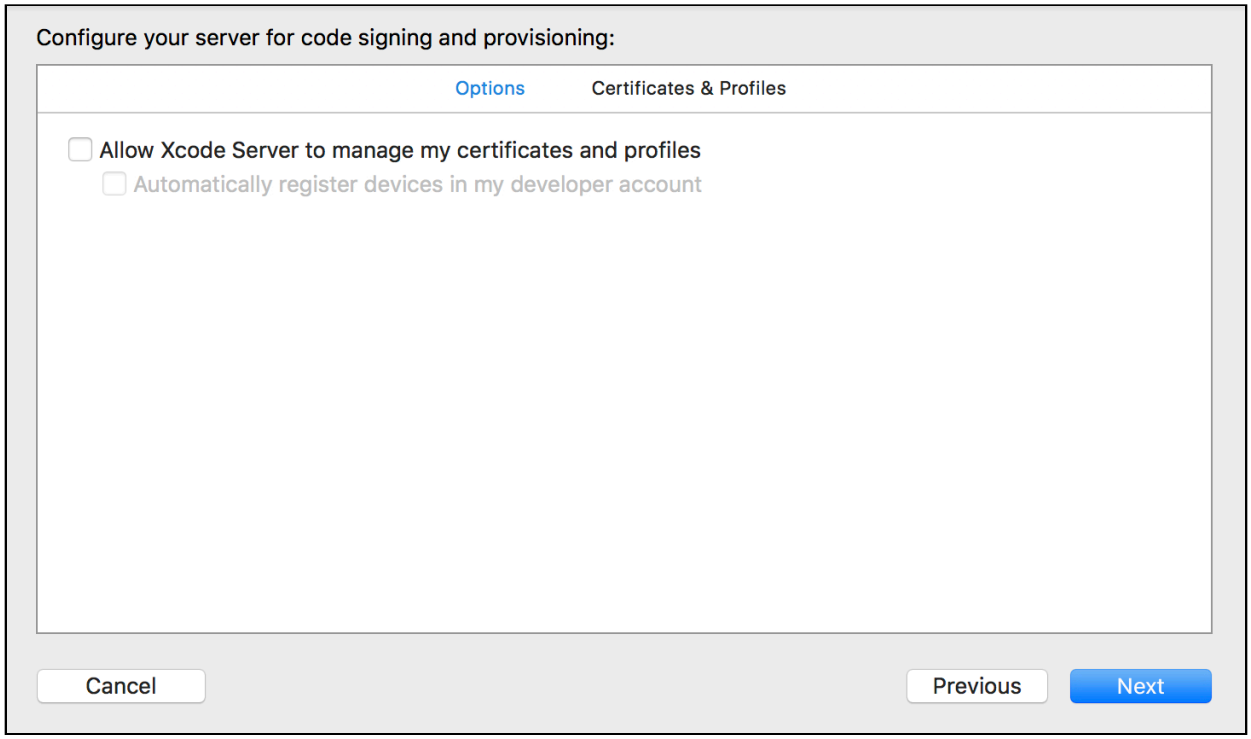
Click **Next**.



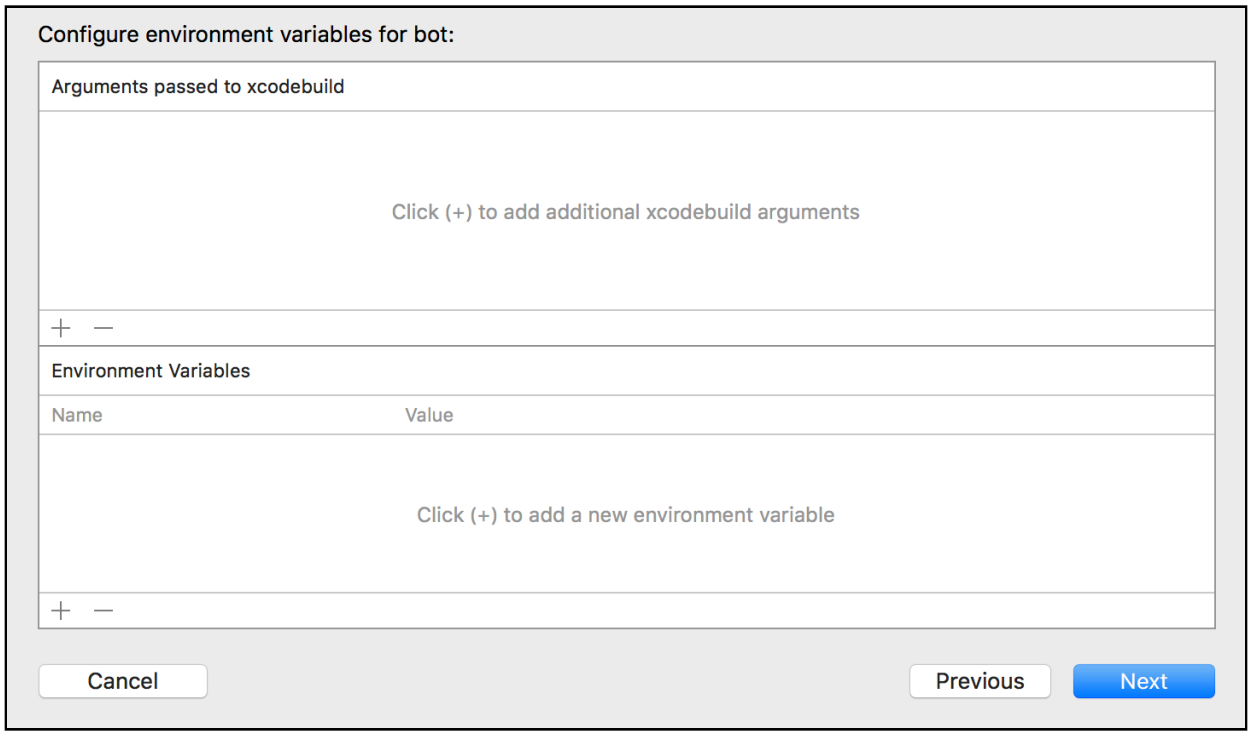
You can schedule the bot to perform integrations periodically (hourly, daily, or weekly), on every commit, or manually. Select the **On Commit** option from the dropdown. Check the box to automatically run an integration whenever Xcode is upgraded to help isolate issues resulting from an update. Use the **Clean** pop-up menu to specify the frequency with which to do a clean build. Click **Next**.



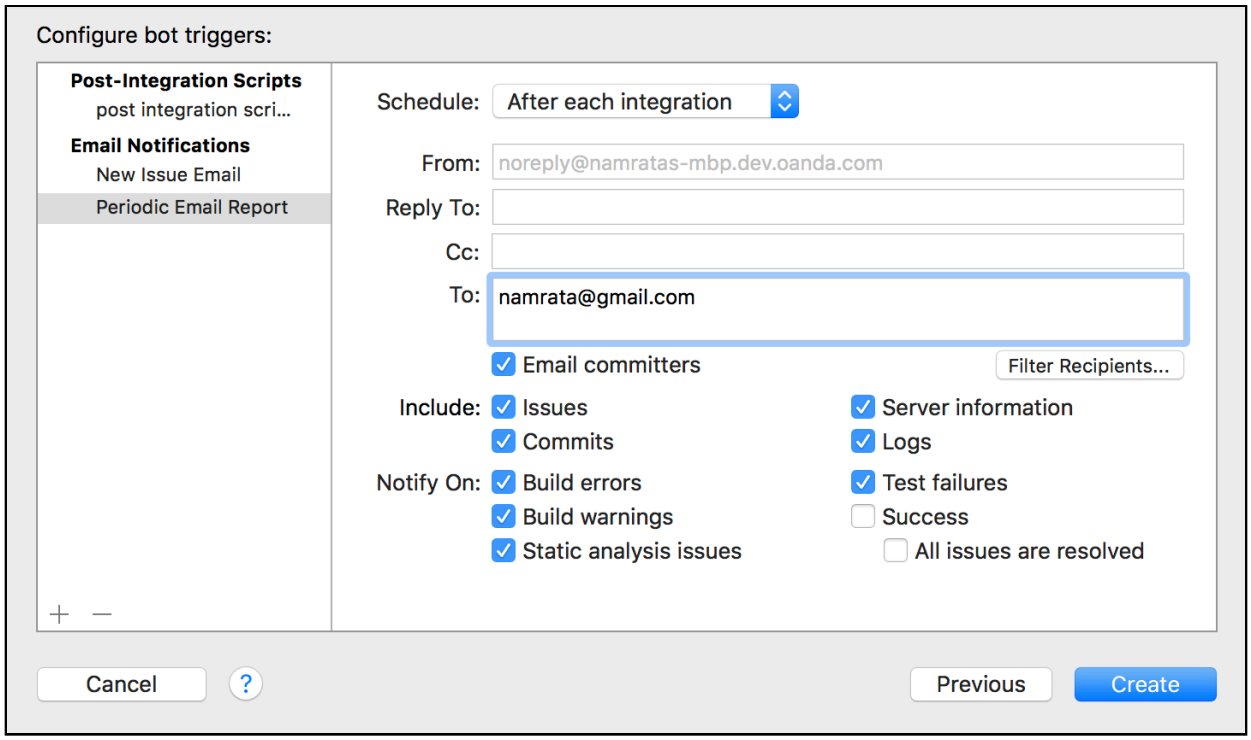
Here you can choose what kinds of devices or simulators the app will be tested on. Since you are running the server on your laptop and want to do a quick test, you will only select iPhone 5s from the list. You can run all the tests in parallel so that they finish faster. Click **Next**.



Configure your server for code signing and provisioning. Click **Next**.



Here you define any environment variables needed by Run Script build phases that execute as part of your integration, or for your pre-integration and post-integration triggers, then click **Next**.



Triggers are actions that a bot can perform before and after an integration. A trigger can run custom shell scripts and/or send email reports. Click on **+** and select **Periodic Email Report**. Select the information to include and the conditions under which reports are sent. Select **Create**.

You are done setting up the bot!

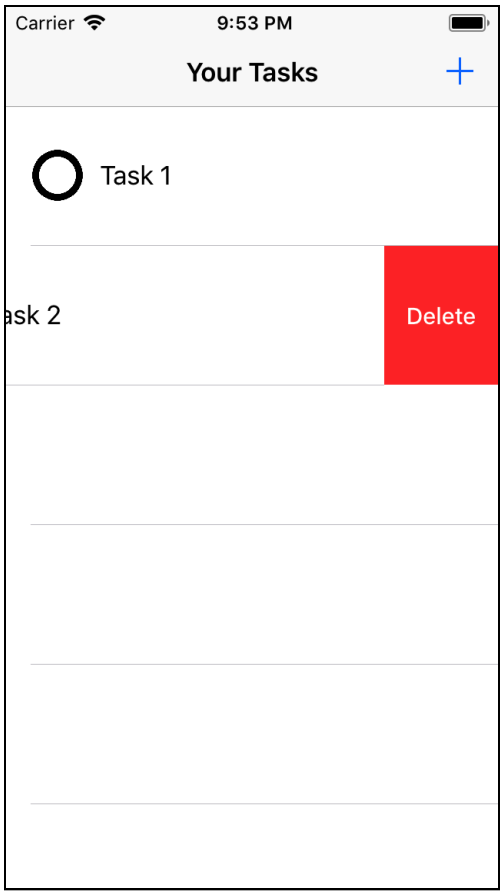
6) Test The Integration

Let's make a small change to your app to test our bot. Wouldn't it be nice to be able to delete the tasks? Open **TaskTableViewController** and add the following.

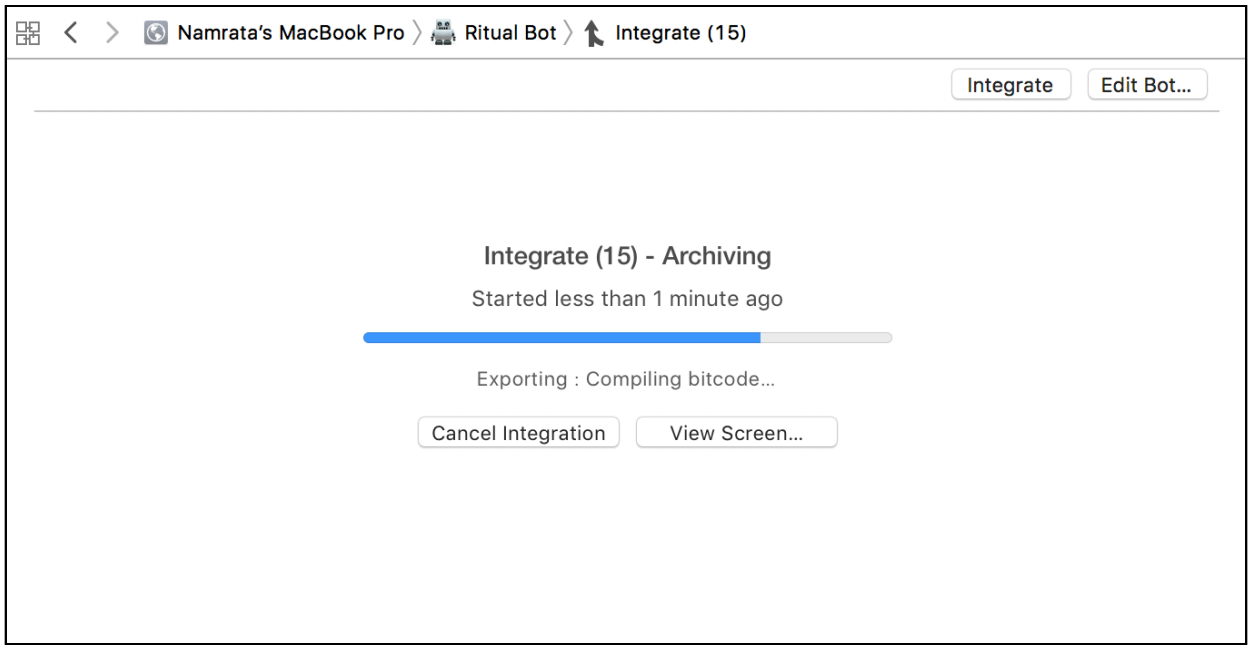
```
override func tableView(_ tableView: UITableView, canEditRowAt indexPath:
IndexPath) -> Bool {
  return true
}

override func tableView(_ tableView: UITableView, commit editingStyle:
UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {
  if editingStyle == .delete {
    tasks.remove(at: indexPath.row)
    tableView.deleteRows(at: [indexPath], with: .fade)
  }
```

```
}
```

Here you enable editing of the rows in your table view. Then if the user edits a row with the delete gesture, you remove the corresponding task from the list and delete that row with a fade animation. Build and run. Test deleting a task by swiping left on a row.



Commit the changes to remote. You will notice your bot starts an integration when it detects there is a new commit in the remote repository.



7) That's it!

Congrats, at this time you should have a good understanding of how to set up your project for continuous integration with Xcode Bots! It's time to move onto Demo 2.

43 App Development Workflow: Demo 2

By Namrata Bandekar

In this demo, you will explore using Crashlytics for crash reporting. The steps here will be explained in the demo, but here are the raw steps in case you miss a step or get stuck.

> **Note**: Begin work with the starter project in **Demo2\starter**.

1) Installing Crashlytics

Go to Terminal and navigate to your project directory. In the Ritual directory type the following command to initialize your Podfile.

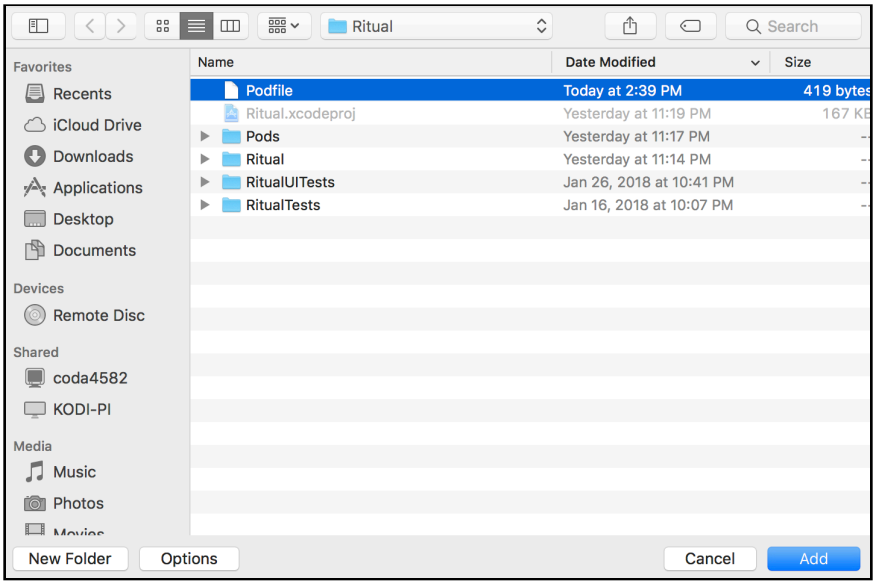
```
pod init
```

In the terminal run `cat Podfile`. You will see a Podfile is generated with the following contents.

```
# Uncomment the next line to define a global platform for your project
# platform :ios, '9.0'

target 'Ritual' do
  # Comment the next line if you're not using Swift and don't want to use
dynamic frameworks
  use_frameworks!

  # Pods for Ritual

  target 'RitualTests' do
    inherit! :search_paths
    # Pods for testing
  end

  target 'RitualUITests' do
```

```
    inherit! :search_paths
    # Pods for testing
  end

end
```

Now add the Podfile to your project in Xcode.



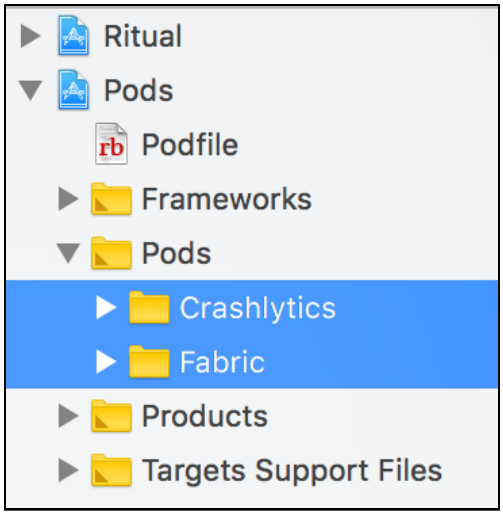
Change the Podfile contents to the following.

```
platform :ios, '9.0'
use_frameworks!

target 'Ritual' do
  # Pods for Ritual
  pod 'Fabric'
  pod 'Crashlytics'
end
```

Then run `pod install` in the terminal in your project directory. You will see the following message if the installation is successful.

```
Analyzing dependencies
Downloading dependencies
Installing Crashlytics (3.10.1)
Installing Fabric (1.7.5)
Generating Pods project
Integrating client project

[!] Please close any current Xcode sessions and use `Ritual.xcworkspace`
for this project from now on.
Sending stats
Pod installation complete! There are 2 dependencies from the Podfile and
2 total pods installed.
```

Now close the Xcode project window. Navigate to your project directory and open the `Ritual.xcworkspace` workspace file. You will notice that the `Crashlytics` and `Fabric` pods appears in the Pods folder.



In the Project Navigator, click on your project and add a new run script build phase

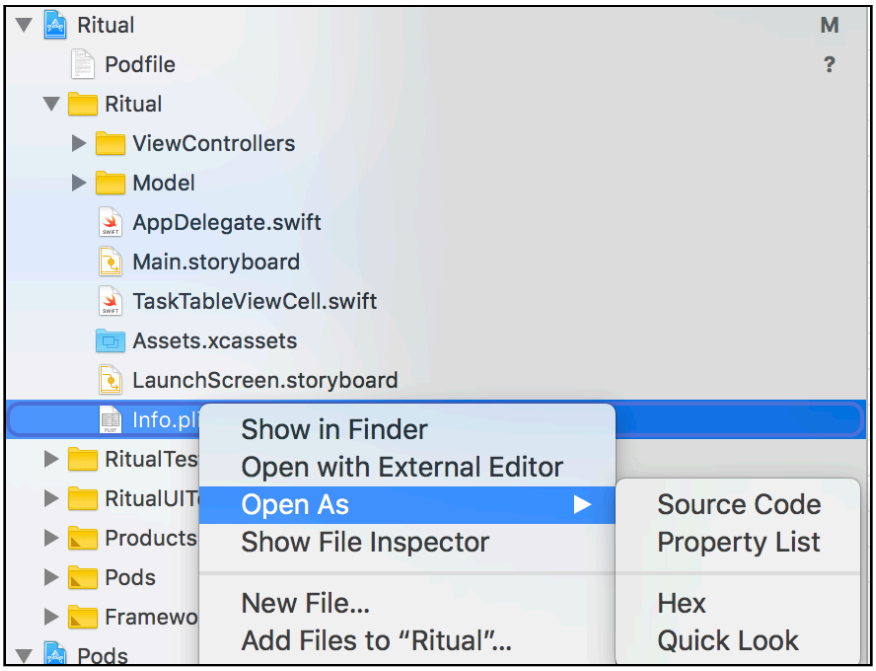
```
"${PODS_ROOT}/Fabric/run" <FABRIC_API_KEY> <BUILD_SECRET>
```

Replace `FABRIC_API_KEY` and `BUILD_SECRET` with your keys from Fabric.

> **NOTE:** If you have an existing Fabric account linked with an organization, you can get your API key and build secret by navigating to **Settings->Organizations**. Click on the organization you want the app to be part of. You will find the necessary keys below the organization name.

> If you have created a new Fabric account, go to https://fabric.io/kits/ios/crashlytics/install which gives you all the instructions listed here. You can also choose to just copy the API key and build secret strings from that page and follow this tutorial.

In the Project Navigator, right click on **Info.plist**, and click **Open As -> Source Code**



Add the following key value pair under `<dict>`

```
  <key>Fabric</key>
  <dict>
    <key>APIKey</key>
    <string><FABRIC_API_KEY></string>
    <key>Kits</key>
    <array>
      <dict>
        <key>KitInfo</key>
        <dict/>
        <key>KitName</key>
        <string>Crashlytics</string>
      </dict>
    </array>
  </dict>
```

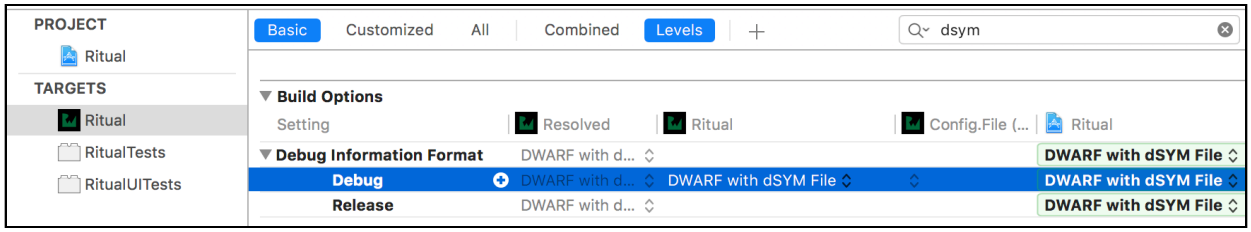
2) Initializing Crashlytics

Next initialize Crashlytics by replacing the contents of your **AppDelegate.swift** file with the following.

```
import UIKit
import Fabric
import Crashlytics

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

  var window: UIWindow?
```

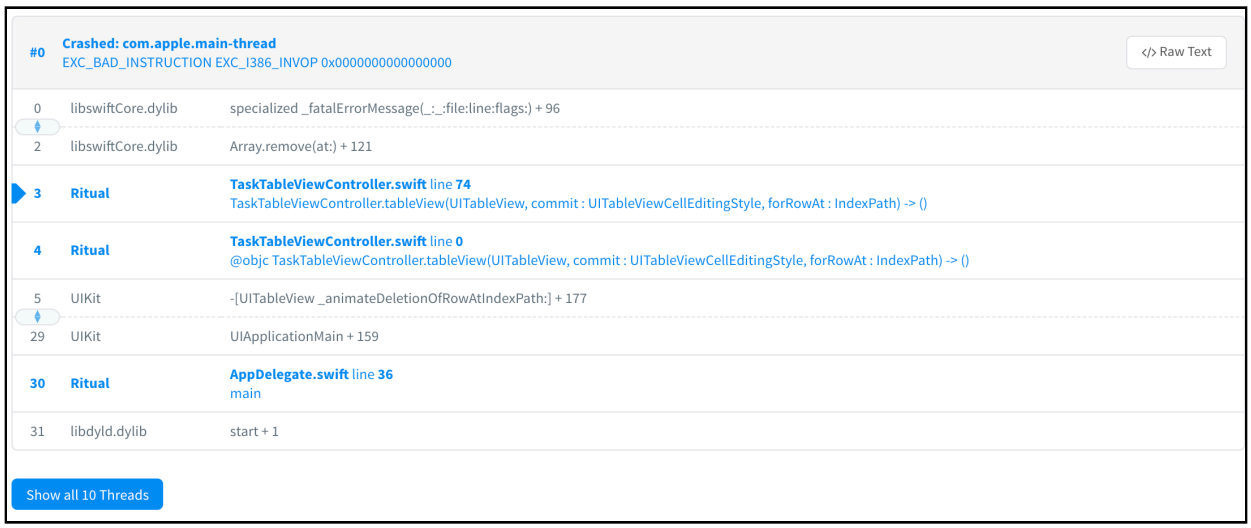
```
  func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplicationLaunchOptionsKey: Any]?) -> Bool {
    Fabric.with([Crashlytics.self])
    return true
  }
}
```

In the Project Navigator, click on Ritual and select **Build Settings**. Search for "dSYM" in the search bar. Ensure the **Debug Information Format** is set to "DWARF with dSYM File" for both Debug and Release



Build and run the app. Since Crashlytics doesn't capture crashes if a debugger is attached at launch, hit the stop button. Now launch the app in the simulator. Delete all the tasks in the list. You will notice the app crashes.

To ensure the crash report is delivered to Crashlytics, launch the app once more. Now check your Fabric dashboard by navigating to https://fabric.io/home. Click on your project. You will see the crash report. Click on the crash to see a detailed stack trace.



3) Resolving the Crash

Let's fix the crash. Based on the stack trace in the report, the crash is happening at

line 74 in **TaskTableViewController.swift** and is related to **Array.remove(at:)**. On closer inspection you will find that when a task is being deleted from the array, the index is offset by 1 leading to an array out of bounds exception. Replace line 74 with the following code.

```
  tasks.remove(at: indexPath.row)
```

Build and run the app. Delete all tasks and ensure that the app does not crash.

4) That's it!

Congrats, by now you should be familiar with monitoring your app's performance through crash reports! Take a break, and then it's time to move onto Demo 3.

44 App Development Workflow: Demo 3

By Namrata Bandekar

In this demo, you will learn how to acquire data related to user behaviour and how to use this to iterate on your app. For this demo, you will be using Mixpanel, a product analytics platform.

The steps here will be explained in the demo, but here are the raw steps in case you miss a step or get stuck.

> **Note**: Begin work with the starter project in **Demo3\starter**. As mentioned in the pre-requisites, you should already have a mixpanel account which lets you use their free plan. You should also have cocoapods installed and setup.

1) Getting Started With Mixpanel

Installing Mixpanel

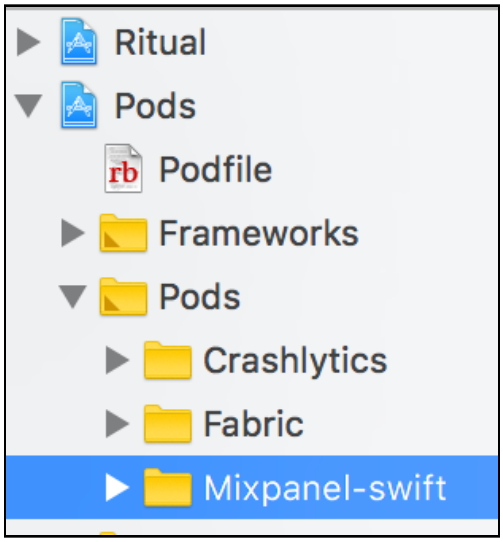
Open Podfile in your starter project. Change the Podfile contents to the following.

```
platform :ios, '9.0'
use_frameworks!

target 'Ritual' do
  # Pods for Ritual
  pod 'Fabric'
  pod 'Crashlytics'
  pod 'Mixpanel-swift'
end
```

You just added `pod 'Mixpanel-swift'` to your target. Run `pod install` in the terminal in your project directory. You will see the following message if the installation is successful.

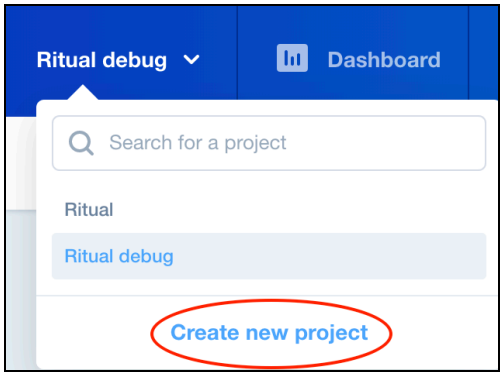
```
Analyzing dependencies
Downloading dependencies
Installing Crashlytics (3.10.1)
Installing Fabric (1.7.5)
Using Mixpanel-swift (2.3.3)
Generating Pods project
Integrating client project
Sending stats
Pod installation complete! There are 3 dependencies from the Podfile and
3 total pods installed.
```

Make sure you are using the `Ritual.xcworkspace` workspace file. You will notice that the `Mixpanel-swift` pod appears in the Pods folder.



Initializing Mixpanel

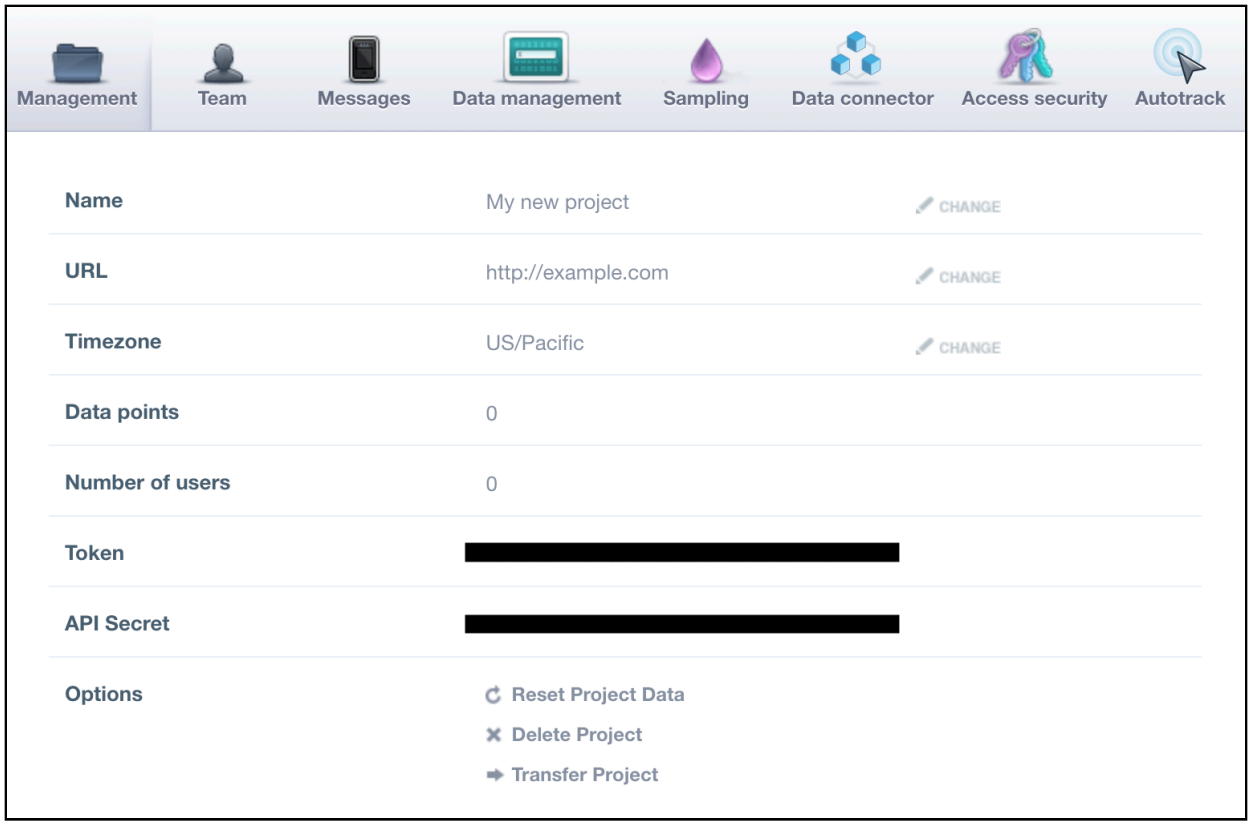
To start tracking with the Mixpanel Swift library, you must first initialize it with your project token. Sign in to your Mixpanel account and click the project dropdown on the top left. Click `Create new project`.



Set the project name to Ritual debug. Now create another project and set the name to Ritual production. You will use "Ritual debug" to test your Mixpanel events in debug mode and "Ritual production" to see your users events once you release the

app in the App Store.

To obtain the token for your project, Click on settings icon next to your name in the upper righthand corner and select `Project settings`. You will find your project token by under the `Management` tab.



Next create a new class file in your project called TrackingManager.swift and replace the contents of the file with the following.

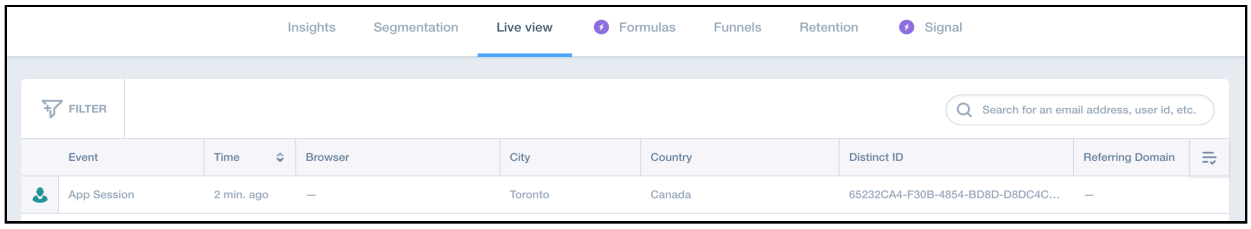
```
import Mixpanel

class TrackingManager {

  static func initializeMixpanel() {
    #if DEBUG
      Mixpanel.initialize(token: "INSERT_RITUAL_DEBUG_TOKEN_HERE")
    #else
      Mixpanel.initialize(token: "INSERT_RITUAL_PRODUCTION_TOKEN_HERE")
    #endif
    Mixpanel.mainInstance().loggingEnabled = true
  }

}
```

Replace the strings `INSERT_RITUAL_DEBUG_TOKEN_HERE` and `INSERT_RITUAL_PRODUCTION_TOKEN_HERE` with the respective Mixpanel tokens in the above code snippet.

Open **AppDelegate.swift** and replace `application(_:didFinishLaunchingWithOptions:)` with the following.

```
func application(_ application: UIApplication,
                 didFinishLaunchingWithOptions launchOptions:
  [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
  Fabric.with([Crashlytics.self])
  TrackingManager.initializeMixpanel()
  return true
}
```

Build and run the app. Now, on MixPanel, navigate to the `Live view` tab in the Ritual debug project. You will notice, Mixpanel has already started tracking your actions.



Now your project is ready to start using Mixpanel!

2) Tracking Events

Open **TrackingManager.swift** and add the following method.

```
static func trackEvent(name: String, eventProperties: [String:
MixpanelType]? = nil) {
  Mixpanel.mainInstance().track(event: name,
                                properties: eventProperties)
}
```

Mixpanel tracks events with an associated event name. Each event can also have a properties dictionary. Property keys must be String objects and the supported value types need to conform to `MixpanelType`. `MixpanelType` can be either `String`, `Int`, `UInt`, `Double`, `Float`, `Bool`, `[MixpanelType]`, `[String: MixpanelType]`, `Date`, `URL`, or `NSNull`.

Open TaskTableViewController.swift and add the following to the end of the class.

```
private func trackCreateTask(_ task: Task) {
  TrackingManager.trackEvent(name: "Create Task",
                             eventProperties:
["isReminderSet":task.isReminderSet])
}
```

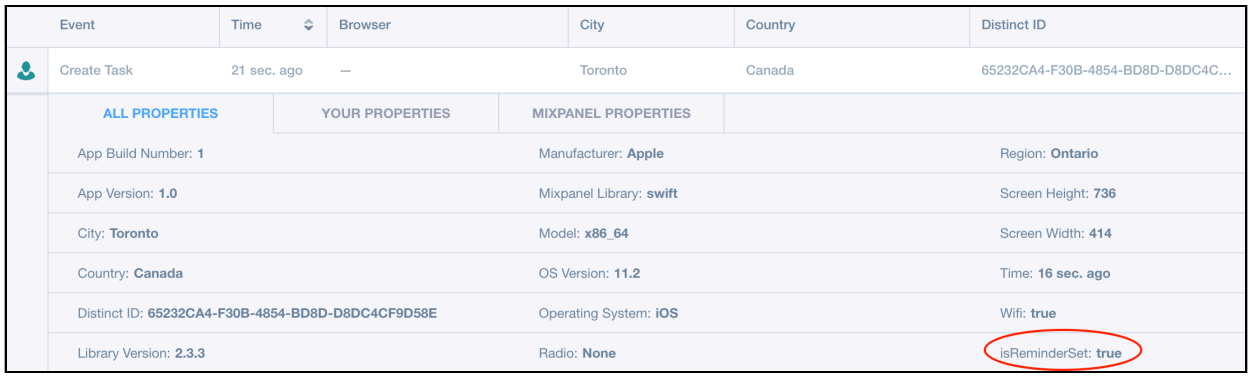
Replace `unwindToTaskList()` with the following.

```
@IBAction func unwindToTaskList(sender: UIStoryboardSegue) {
```

```
  if let sourceViewController = sender.source as?
TaskDetailsViewController,
    let task = sourceViewController.task {
    if let selectedIndexPath = tableView.indexPathForSelectedRow {
      tasks[selectedIndexPath.row] = task
      tableView.reloadRows(at: [selectedIndexPath], with: .none)
    } else {
      let newIndexPath = IndexPath(row: tasks.count, section: 0)
      tasks.append(task)
      tableView.insertRows(at: [newIndexPath], with: .automatic)
      // This line is new
      self.trackCreateTask(task)
    }
    self.saveTasks()
  }
}
```

Here you add a call to `trackCreateTask()` when a new task is created.

Now build and run the app. Create a new task. Go to your Mixpanel project and see that the event `Create Task` shows up in the live view (it may take a few seconds for it to show up).



3) Timed Events

With Mixpanel, you can track the time it took for an action to occur.

Open **TrackingManager.swift** and add the following method.

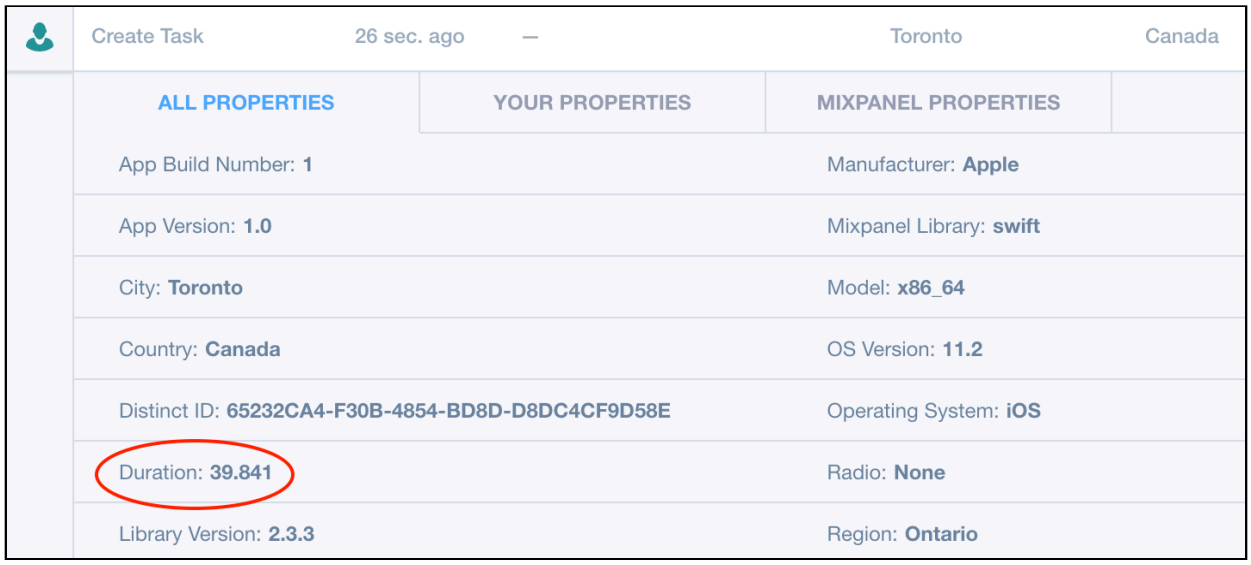
```
static func startTrackingTimedEvent(name: String) {
  Mixpanel.mainInstance().time(event: name)
}
```

Add the following to **TaskTableViewController** at the end of `viewDidLoad()`.

```
TrackingManager.startTrackingTimedEvent(name: "Create Task")
```

Build and run the app and create another new task. Check the live view tab and you will see another `Create Task` event. This time the event has a new property called

`Duration` which is the time elapsed in seconds since the timed tracking for the event started and the event was triggered.



4) User Profiles

The Mixpanel library assigns a default unique identifier called "distinct ID" to each unique user who installs your application. This distinct ID is saved to device storage so that it will persist across sessions.

You can store user profiles which are persistent sets of properties that describe a user. A profile can consist of properties like name, email address, and signup date.

Add the following methods to **TrackingManager.swift**.

```
  static func identifyUser() {
    Mixpanel.mainInstance().identify(distinctId:
Mixpanel.mainInstance().distinctId)
  }

  static func storeUserProperties(userProperties: [String: MixpanelType])
{
    Mixpanel.mainInstance().people.set(properties: userProperties)
  }
```

Add the following at the end of `initializeMixpanel()`.

```
  identifyUser()
```

Open **TaskTableViewController.swift** and add the following code.

```
  private func trackNumberOfTasks(_ numberOfTasks: Int) {
    TrackingManager.storeUserProperties(userProperties: ["Number of
Tasks": numberOfTasks])
```

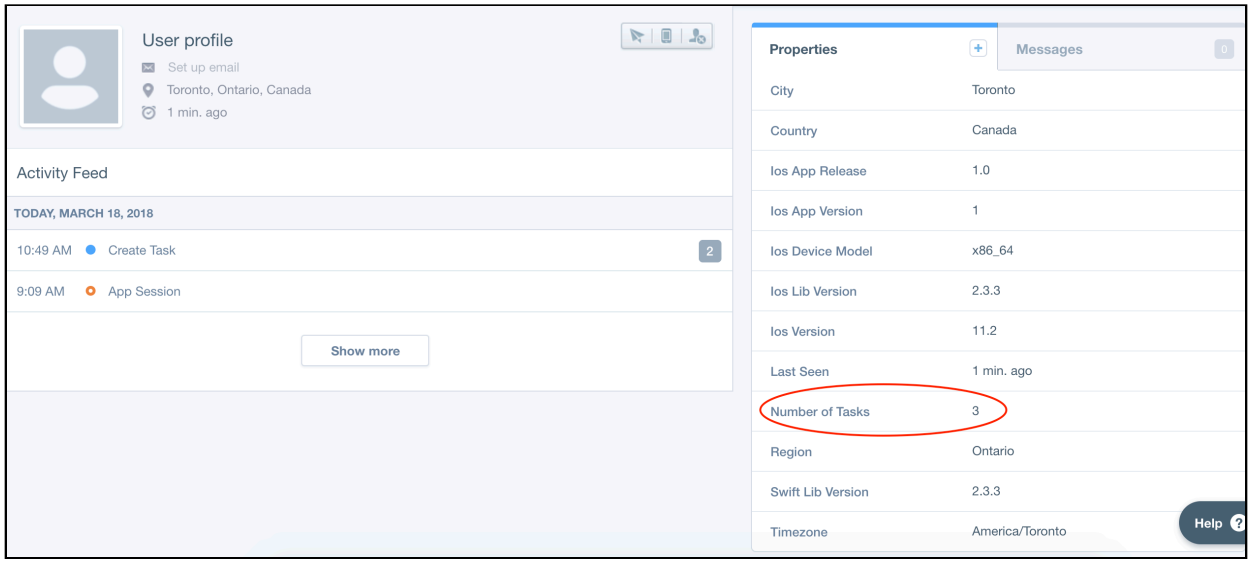
```
    }
```

At the end of `saveTasks()` add a call to `trackNumberOfTasks()`

```
  trackNumberOfTasks(tasks.count)
```

Here you are tracking the number of tasks in the user's list as a user property.

Build and run the app and add another new task. Open Mixpanel and navigate to **Users->Explore**. Select the user. You will see a detailed user profile with your new property and all the events showing the user's activity in your app. This might take some time to show up.



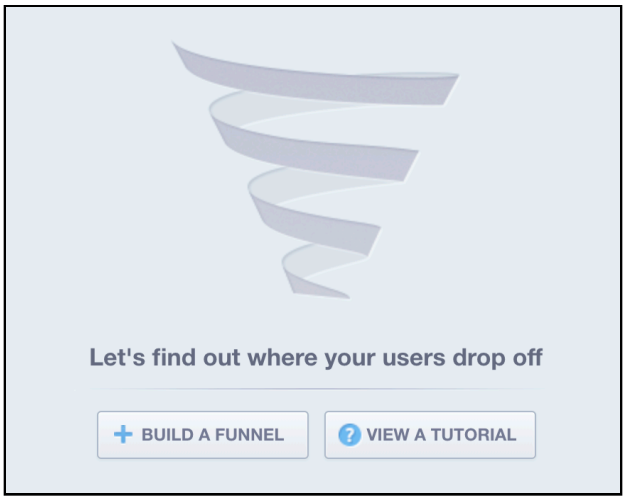
5) Funnels

Add the following to `application(_:didFinishLaunchingWithOptions:)` in your **AppDelegate.swift** file just before the return statement.

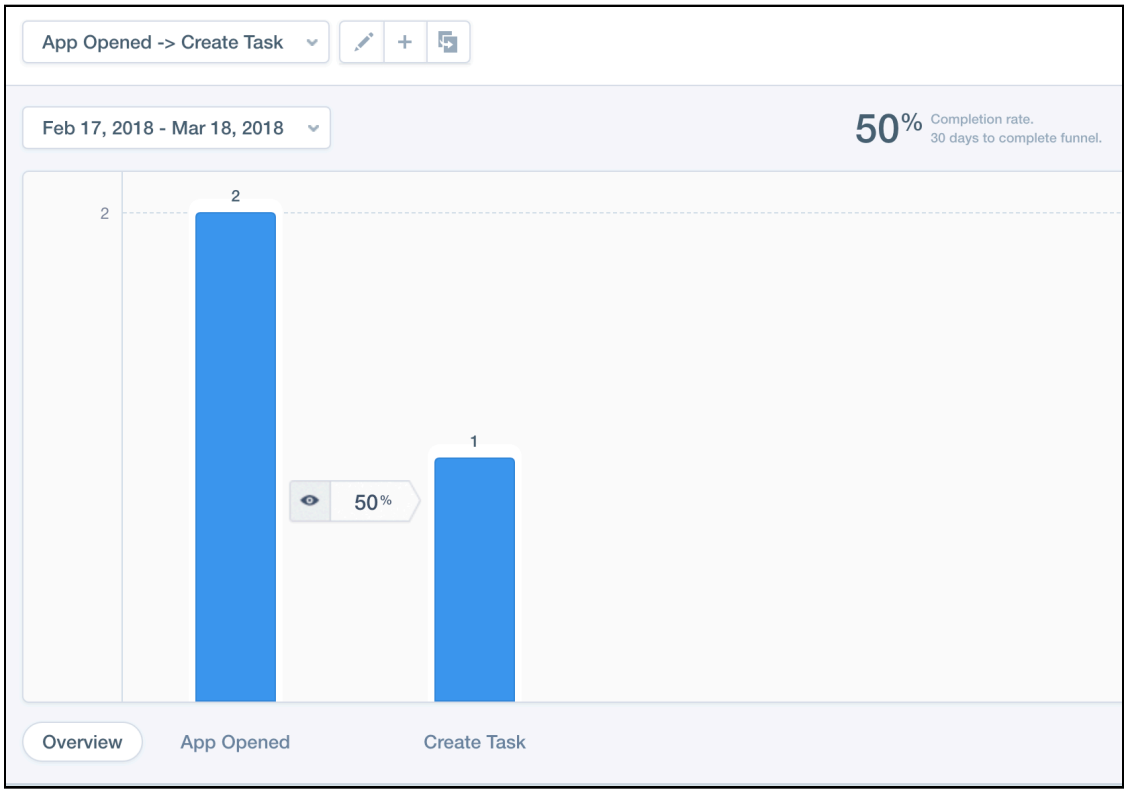
```
  TrackingManager.trackEvent(name: "App Opened")
```

Here you trigger the "App Opened" event every time the user launches Ritual. Build and Run so Mixpanel knows about your new event.

You can analyze how users navigate through your app with Funnel analysis. In your Mixpanel project navigate to **Analysis->Funnels**. Click **Build A Funnel**.



Name your funnel "App Opened -> Create Task". Assign the "App Opened" event for Step 1 and the "Create Task" event for Step 2. Click Save. You will see a chart with the number of times these events were triggered in succession.



6) That's it!

Congrats, at this time you should have a good understanding of how to get started with user tracking in your iOS apps! There are lot of other analytics features you could explore such as Segmentation, Cohorts, and Retention. You can also compare

what Mixpanel offers to other product analytics tools like Amplitude and Flurry.

45 App Development Workflow: Extras

By Namrata Bandekar

In this section, you will learn how to distribute your builds on iTunes Connect and use TestFlight for beta testing.

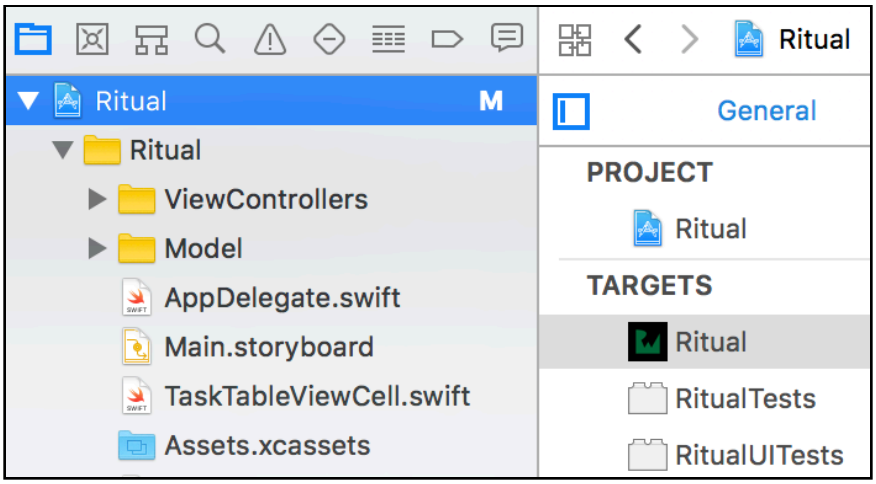
1) Prepare App for Distribution

Now that your app is setup to record crash reports and user behaviour, you will prepare it for distribution. In order to upload your app to iTunes Connect, you will need to make sure your app has the required information.

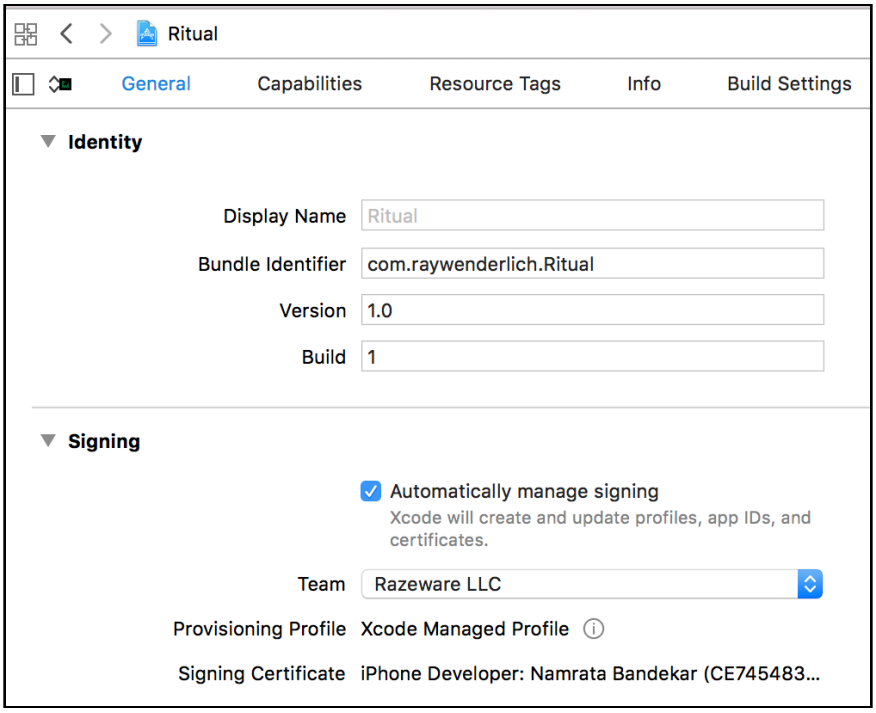
Set Required Metadata

You need to set the app's metadata such as bundle identifier, build information and other necessary assets.

In the project navigator, click on the project and select **Ritual** under Targets. Click the **General** tab.

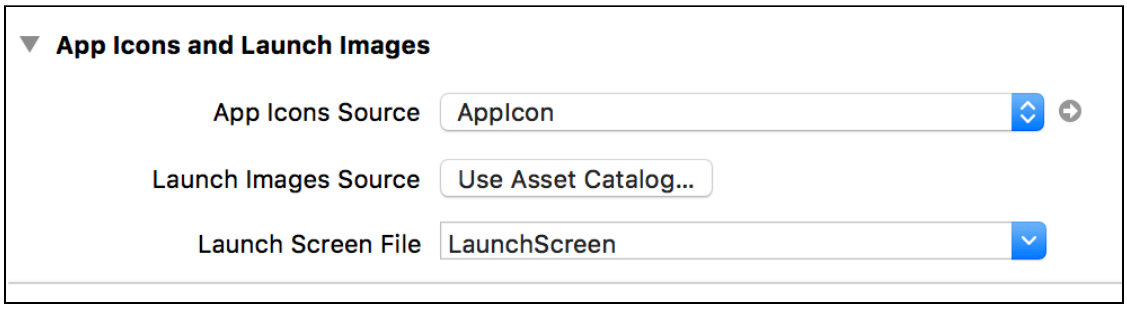


Under the **Identity** section, make sure the Bundle Identifier, Version and Build are set. Under signing you can check **Automatically manage signing**. Assign the target to your team.

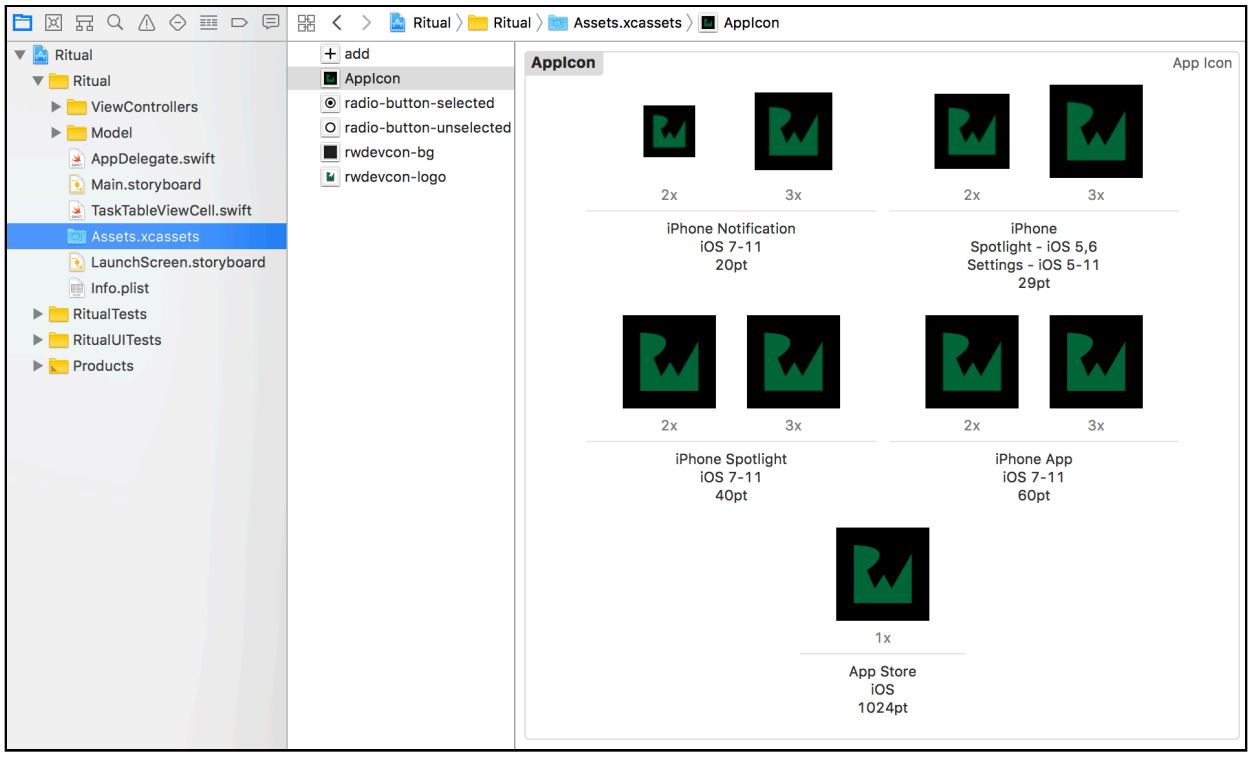


Set Required Assets

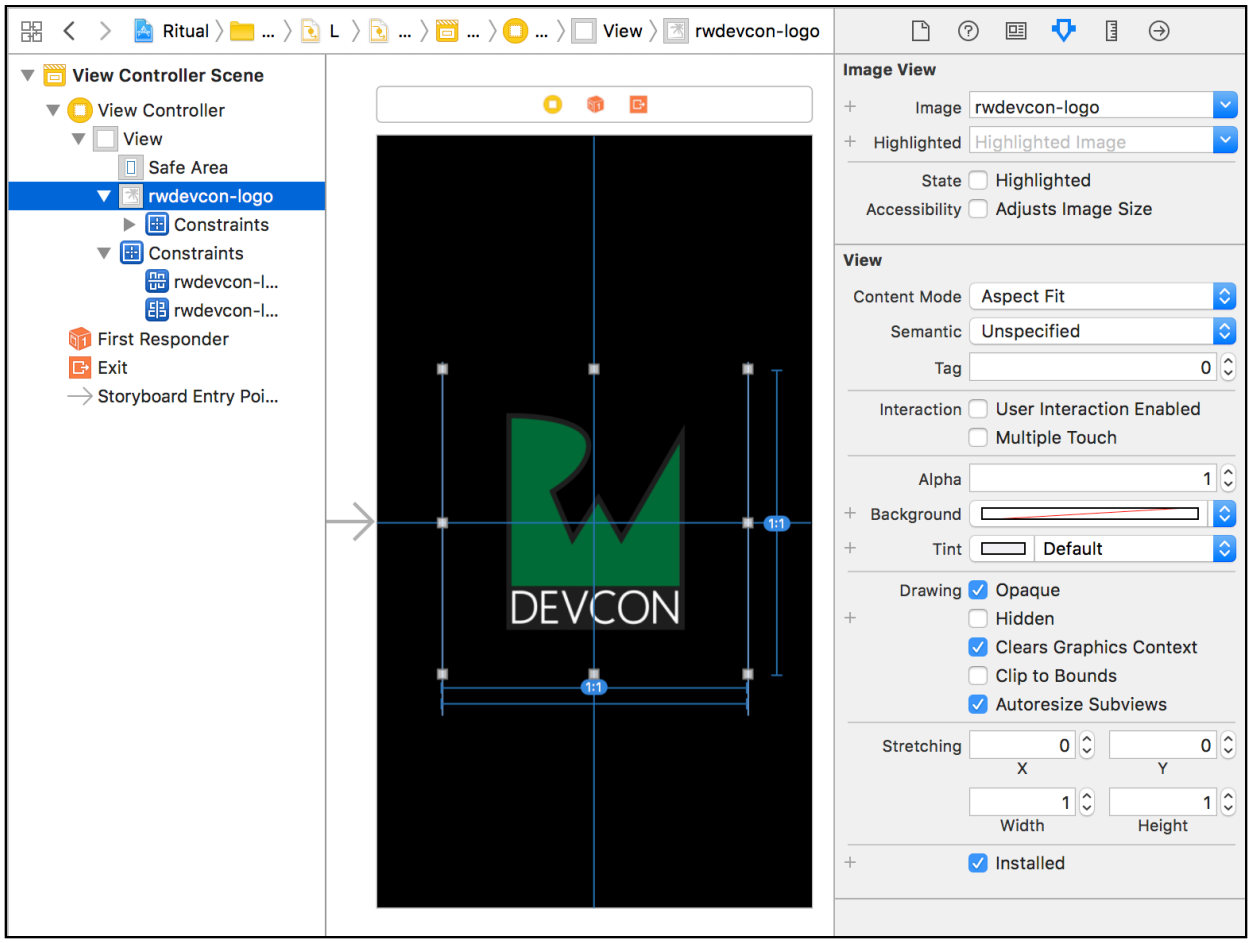
You need to include certain assets in your app in order to prepare it for release. Select the Ritual target and navigate to **General**. By default Xcode sets the files for App Icon and Launch Screen File under **App Icon and Launch Images**.



The App Store icon has to be included in the bundle starting with Xcode 9. In the project navigator, select **Assets.xcassets**. Select **Appicon** and notice the App Store icon 1024 pt has been added.



In the project navigator, select **LaunchScreen.storyboard**. This file has already been changed to include a UIImageView with the RWDevCon logo.



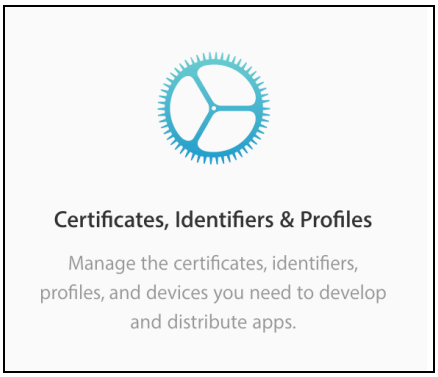
2) Add New App on App Store

Your Bundle ID is a unique identifier for your app in the App store.

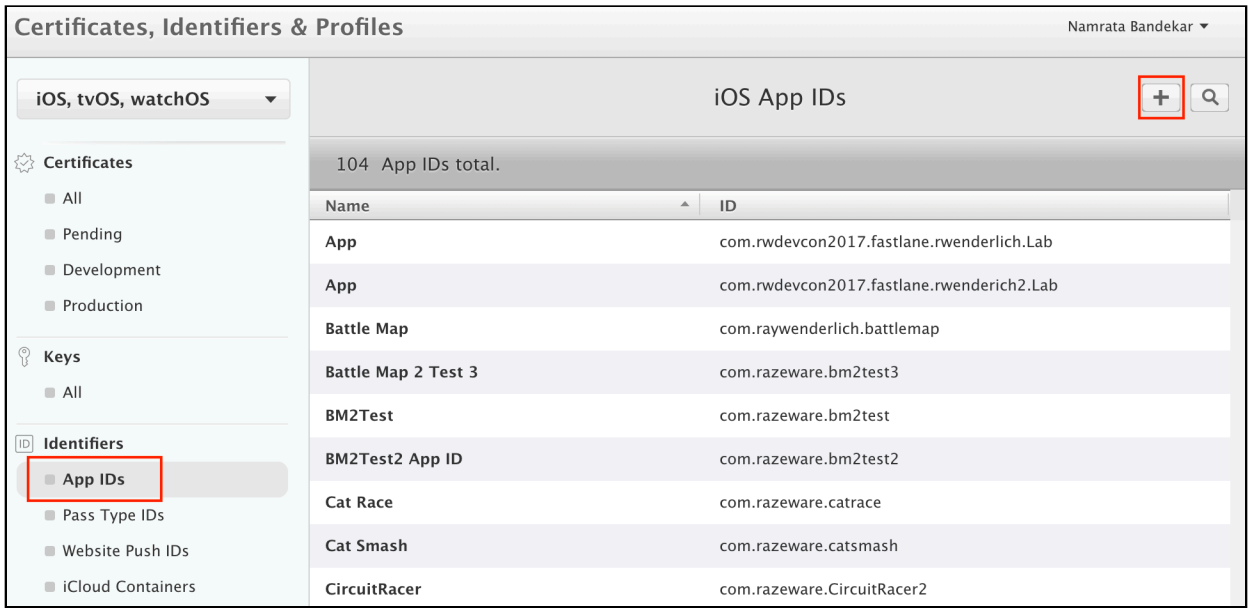
> **Note**: The bundle ID string must contain only alphanumeric characters (A-Z,a-z,0-9), hyphen (-), and period (.). Bundle IDs are case-sensitive.

In Xcode, select the Ritual target and navigate to **General**. Change the Bundle Identifier string to something **unique**. You can add your name to it (com.raywenderlich.Ritual.namratabandekar).

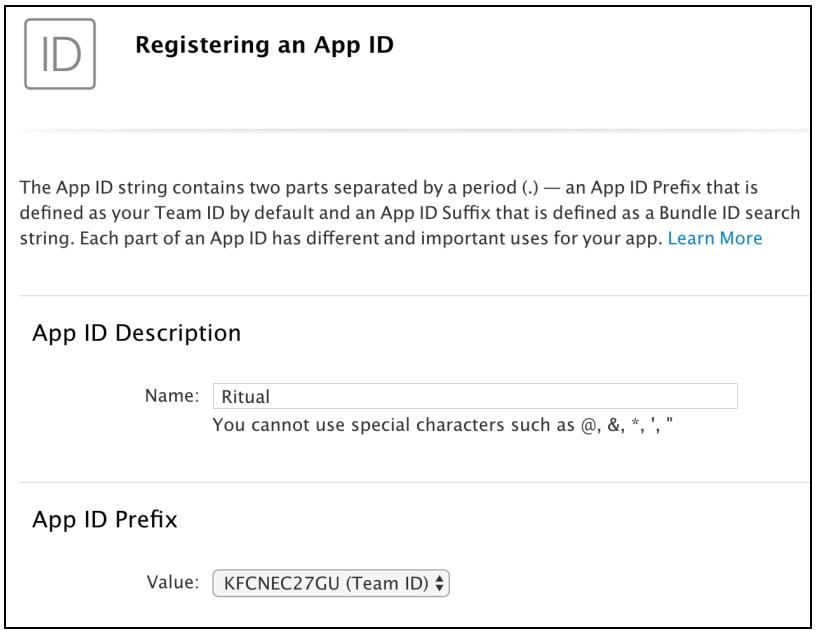
Login to your Apple Developer portal at `https://developer.apple.com/account`. Select **Certificates, Identifiers & Profiles**.



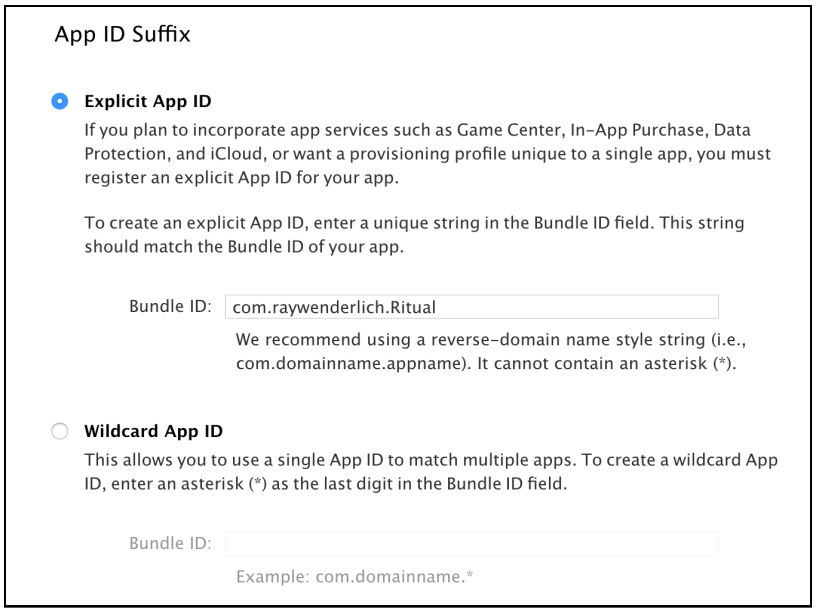
Under the **Identifiers** section on the left, choose **App IDs** and click the **+** button.



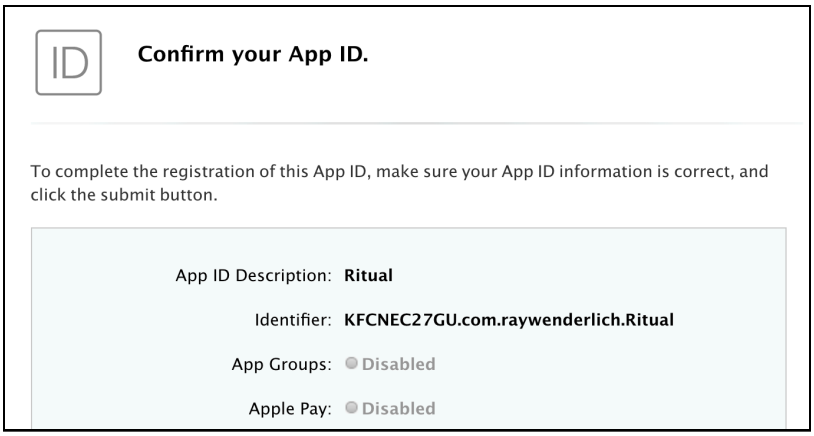
Enter Ritual in the **Name** field and select the appropriate **App ID Prefix** from the dropdown.



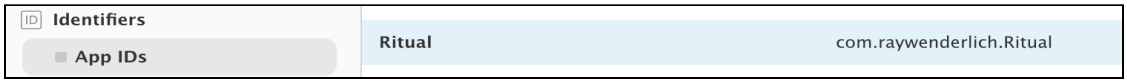
Under App ID Suffix choose **Explicit App ID** and enter the bundle ID as **com.raywenderlich.ritual.<YOUR_NAME>**. This matches the bundle ID of your app in Xcode.



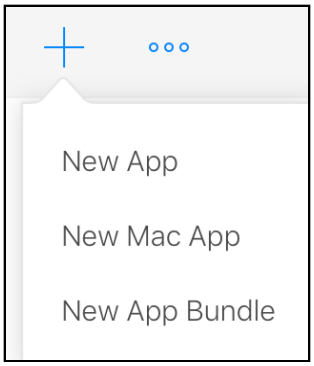
Scroll to the bottom and click **Continue**.



Click **Register** to confirm the App ID. Now go to the App IDs section. Notice the newly created app ID on the right.

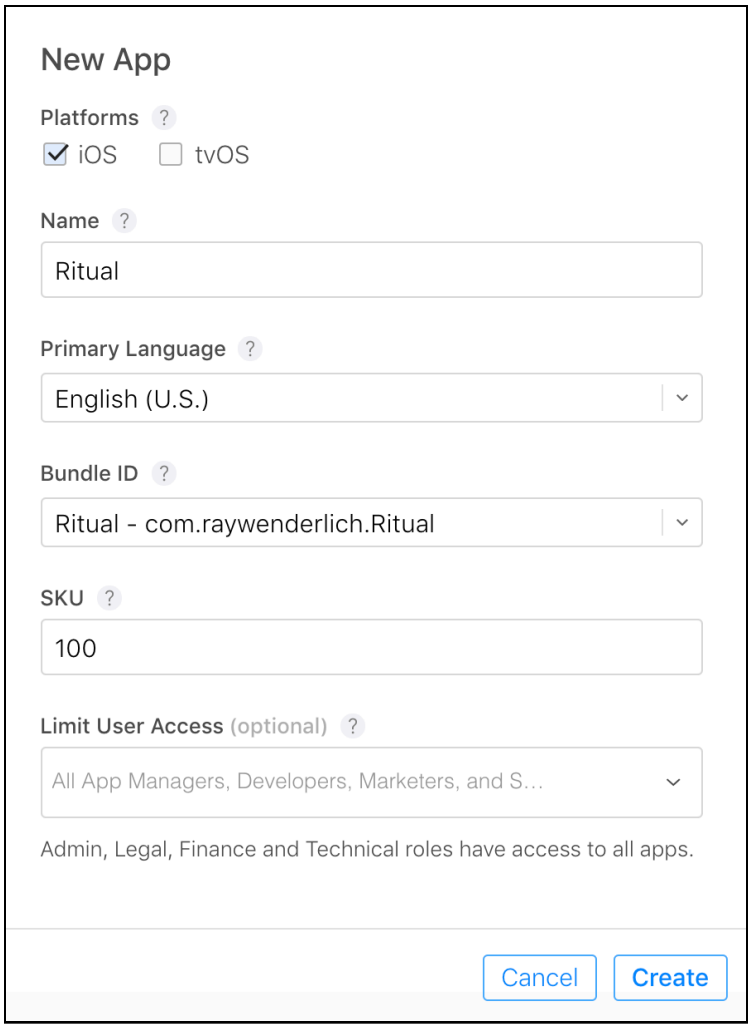


Go to your iTunes Connect account and select **My Apps**. Click **+** and select **New App** to add Ritual as a new app.



Add all the details on the next screen and click **Create**.

> **NOTE:** The name has to be unique and can't be longer than 30 characters. Use "Ritual-YOUR_FULL_NAME" for this demo.



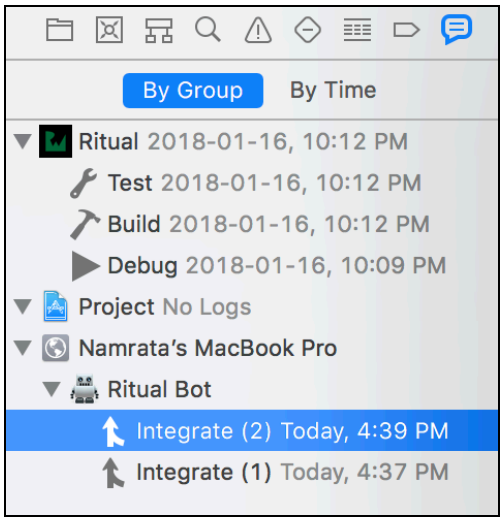
If applicable, commit any changes in your project to your remote repository so that the bot can create a new build or select the bot and click Integrate.

3) Upload Archives

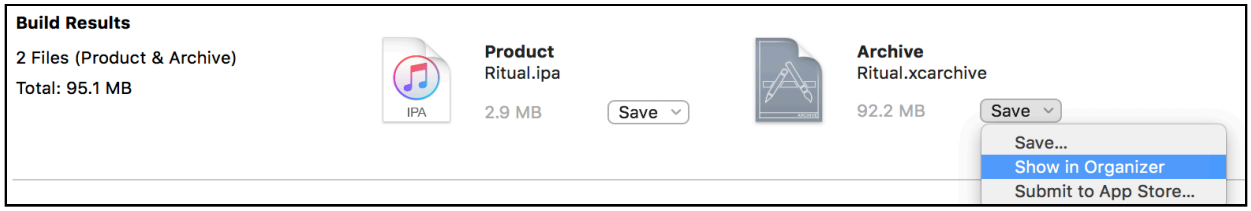
Now that you have created a new app on iTunes Connect, you can upload builds to it.

In Xcode, open **Info.plist**. Add the key `ITSAppUsesNonExemptEncryption` and value `false` to the Info.plist. Here you provide export compliance information with your build. This ensures that once you upload the app to TestFlight, you will not need to answer the compliance questions.

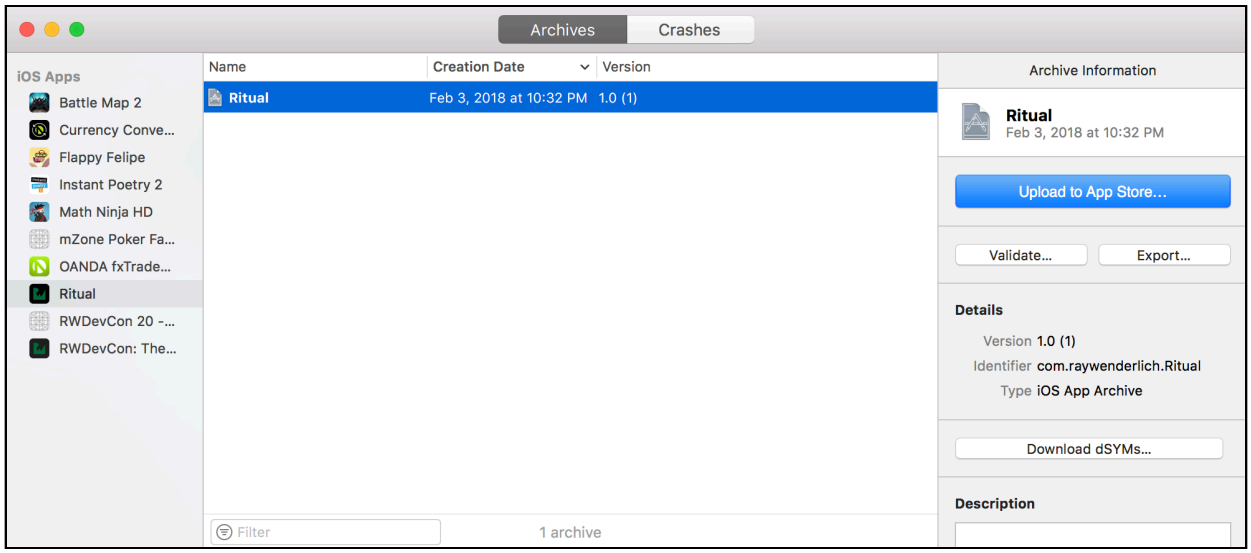
Next navigate to **Report Navigator** or press **Cmd+9**. Select the last integration done by the bot.



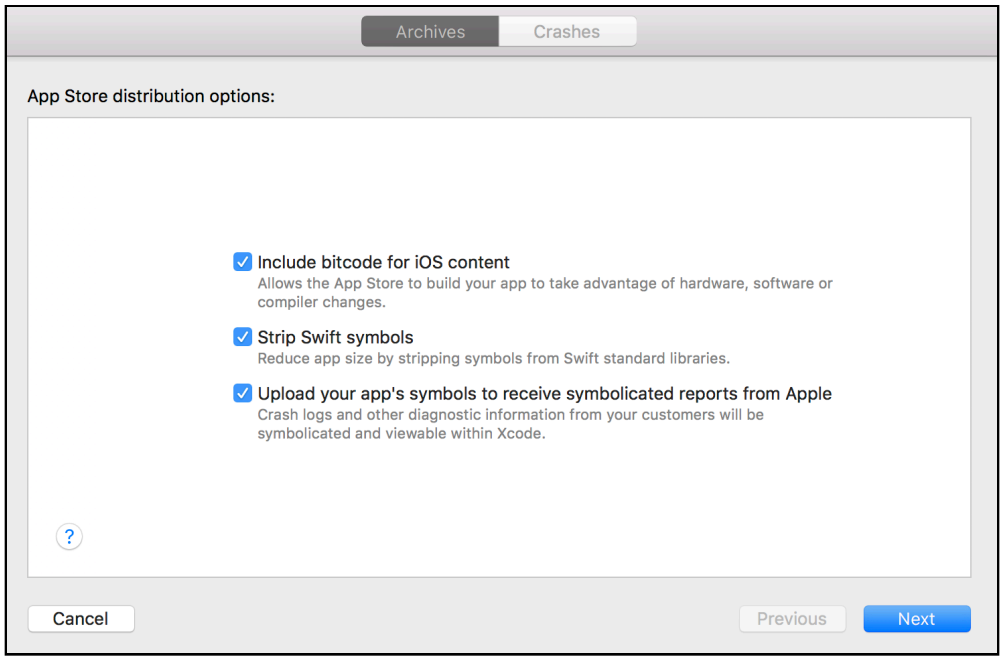
Since we had opted to create an archive with every integration, in the **Build Results** section, you see two files. Click the dropdown for **Ritual.xcarchive** and select **Show in organizer**.



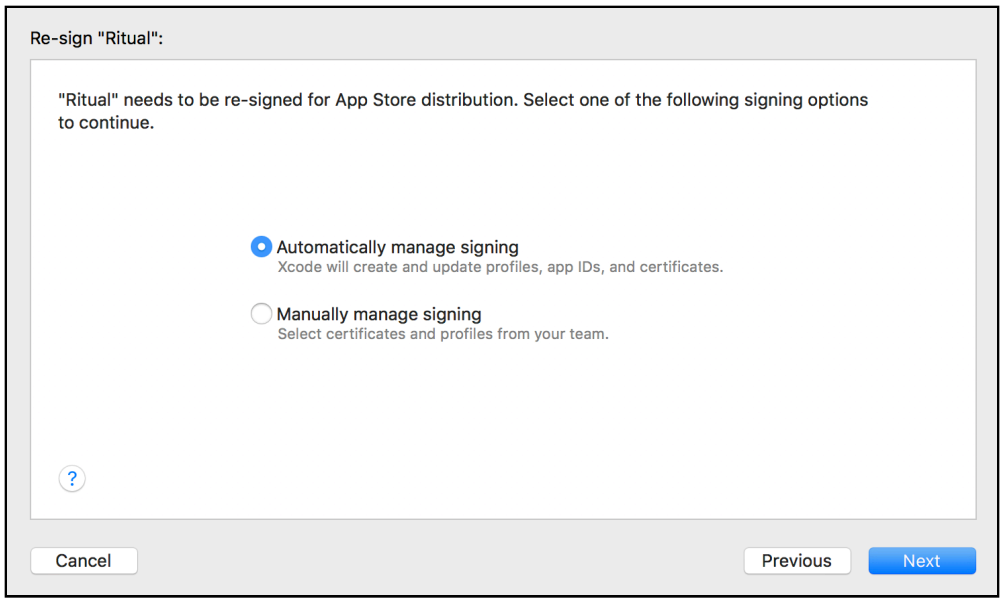
This will add a new archive to the organizer. Click **Upload to App Store**.



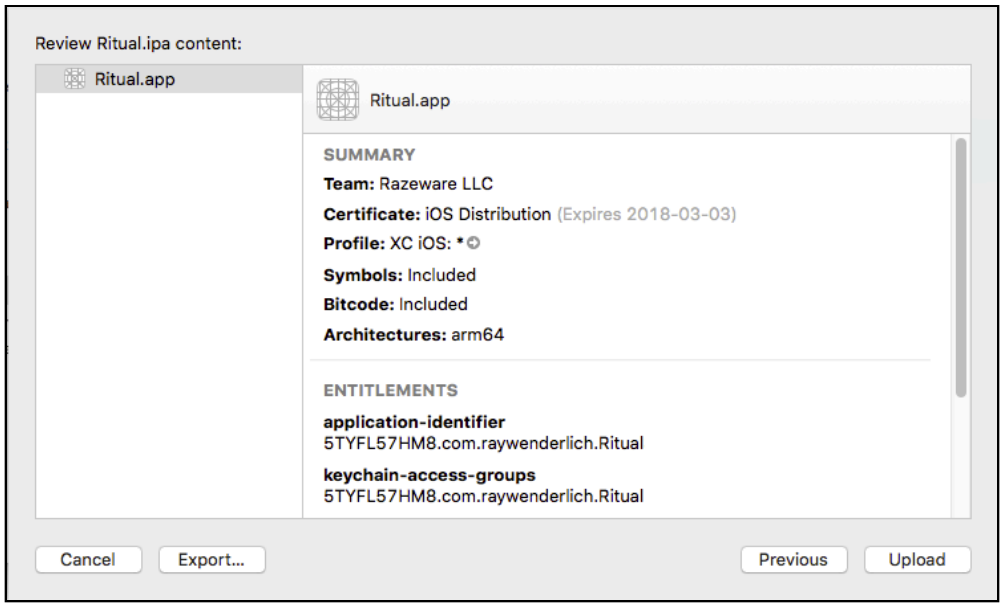
You are presented with a window to select App Store distribution options. Keep all of them selected and click **Next**.



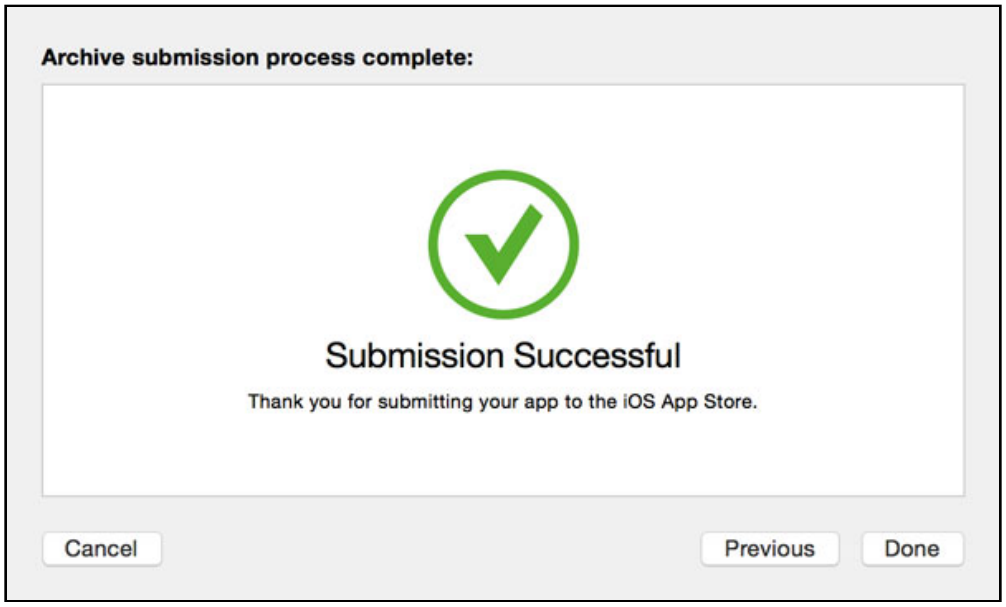
On the next screen select **Automatically manage signing** and click **Next**. This will let Xcode re-sign your app with your distribution certificate. Make sure you have the private key for your distribution certificate in your keychain.



Review the signing certificate, provisioning profile, and entitlements. Click **Upload**.



Hooray! Your app is successfully uploaded.



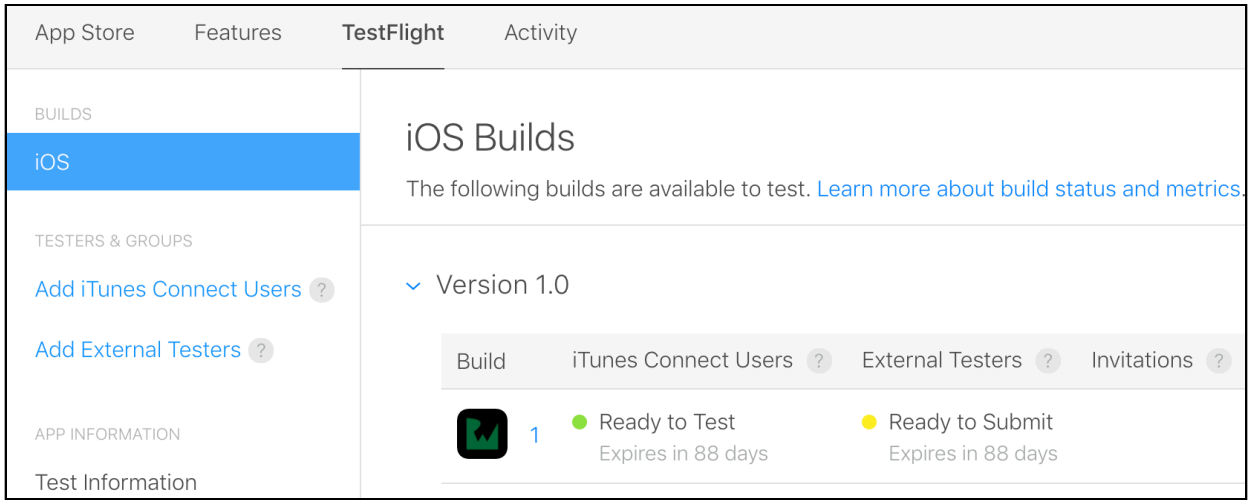
4) TestFlight

TestFlight is Apple's beta testing platform. There are two types of testers - internal and external. Your internal testers are all your iTunes Connect users. You can add up to 25 internal testers to test your app using TestFlight beta testing.

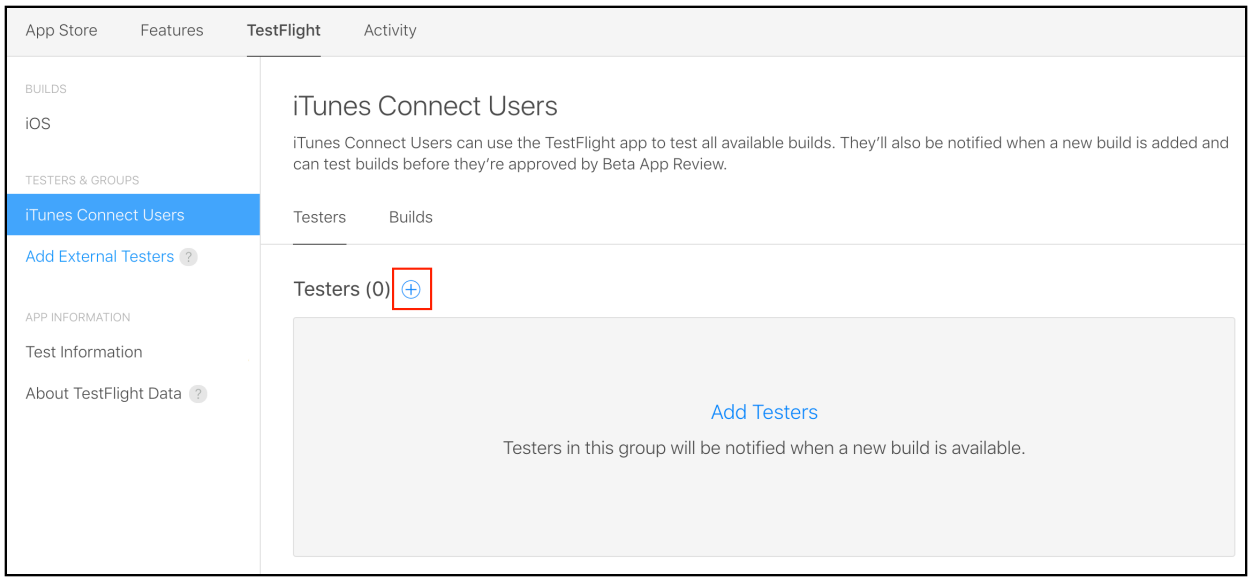
Adding Internal Testers

On the iTunes Connect homepage click **My Apps**, and select Ritual. Then in the

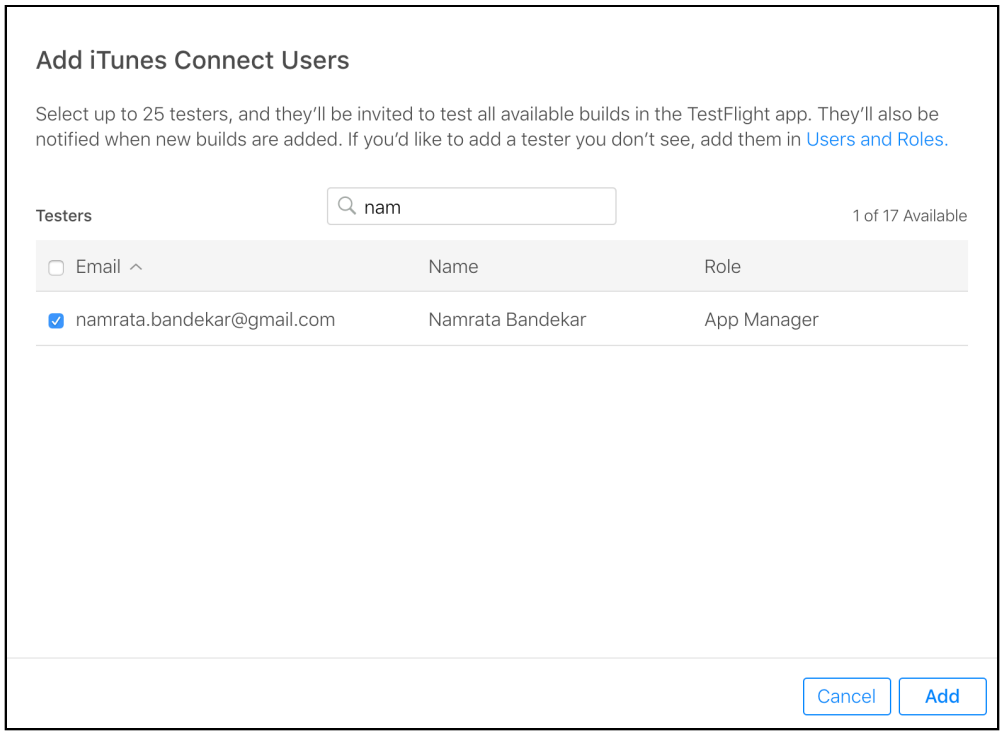
toolbar select **TestFlight**. Under Testers & Groups in the left menu, your first time in, this will say **Add iTunes Connect Users**. Subsequent times, it will say **iTunes Connect Users**.



Click on **Add iTunes Connect Users** and click the **+** button located next to Testers.

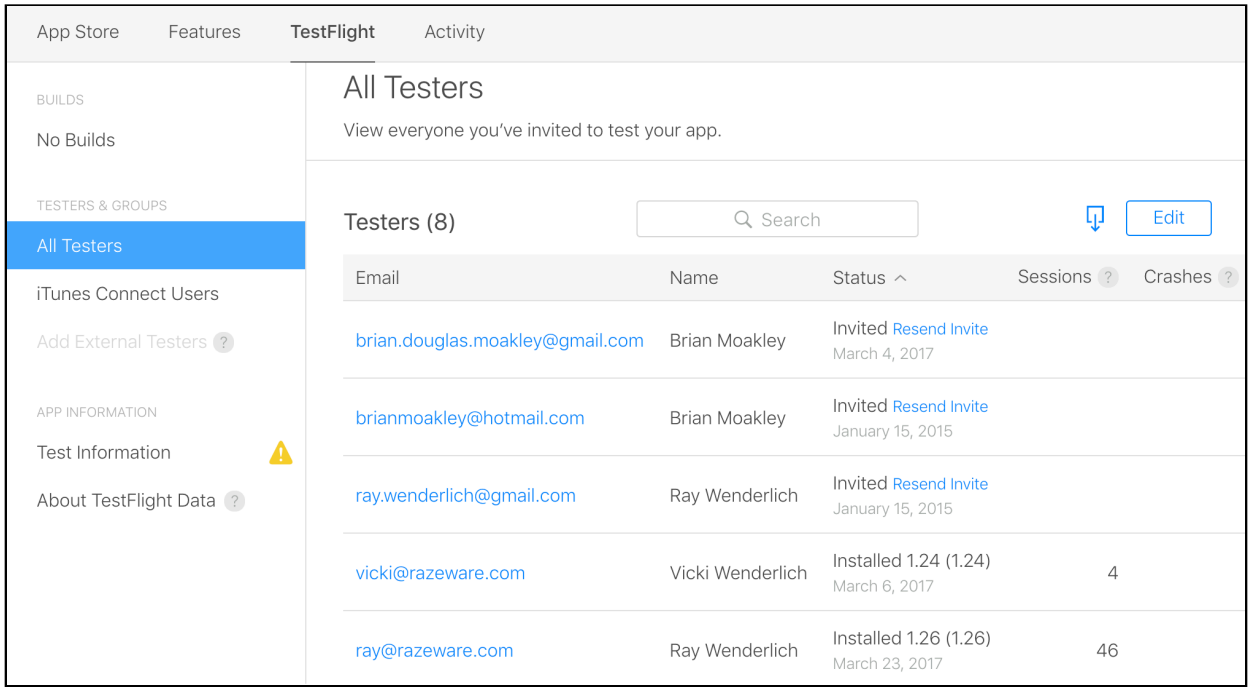


Search for a user you want to add to the internal testers group for this app, select the checkbox next to the user and click **Add**. You can only add users who have one of Admin, App Manager, Developer, or Marketer roles. You are going to add yourself as the internal tester.



Tester Information

In the TestFlight tab, select **All Testers** under the Testers & Groups section in the left column.



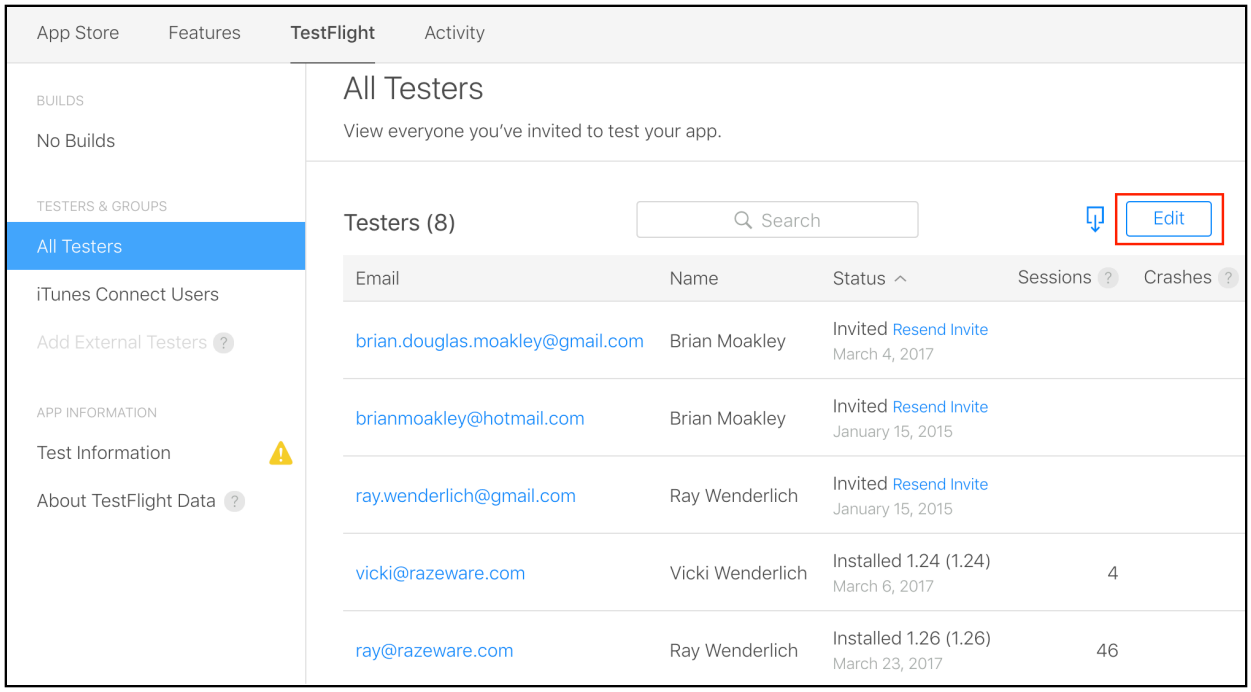
With the new enhancements in TestFlight, you can see who is actively testing and who is experiencing crashes. The current status is shown as **Invited** and it shows

the date on which the invite was sent. Notice the **Resend Invite** button next to the status. This lets you re-invite any of your testers in case they haven't joined yet.

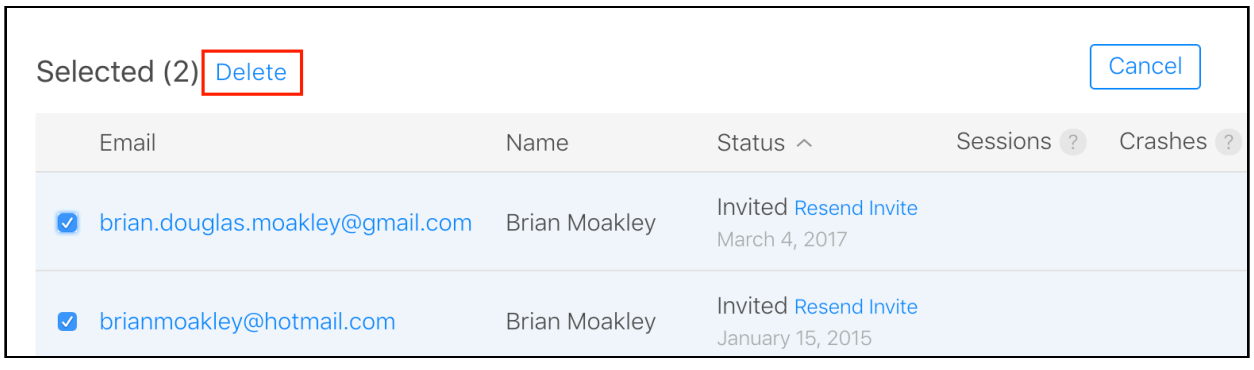
There is a session count for every tester which tells you how many times each tester has used the app. You can also see how many times a particular build has crashed for a user in the **Crashes** column.

Removing Testers

In the **All Testers** view, click **Edit**.



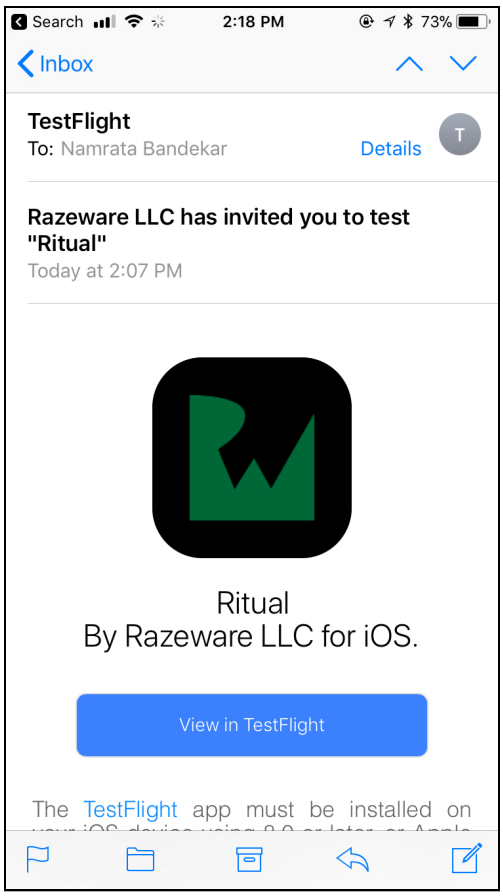
In this state, you can select all the testers you want to delete and then click **Delete**. The title indicates how many testers you have selected.



If you remove a tester from the **All Testers** section, they would be removed from all groups and would no longer have access to any builds of your app. If you remove a tester from a specific group, this would just remove their access to the builds in that group.

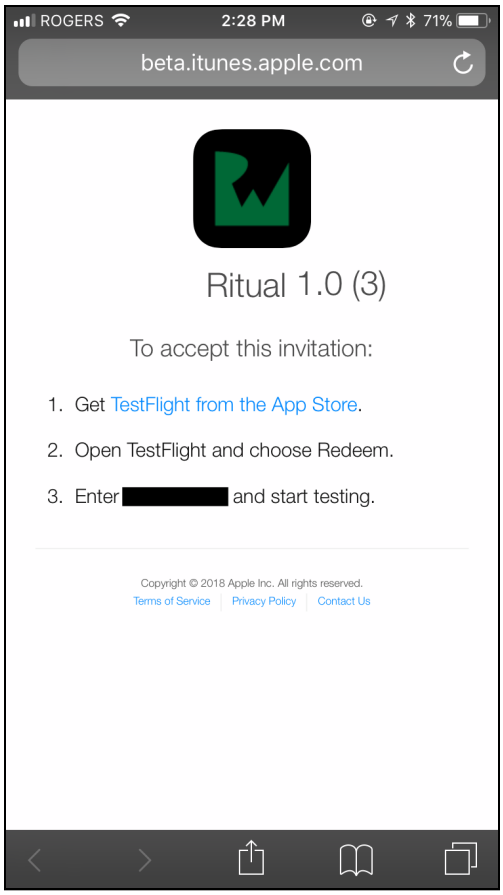
TestFlight app

After you add your email as an internal tester, you will receive an email invitation to test the app.

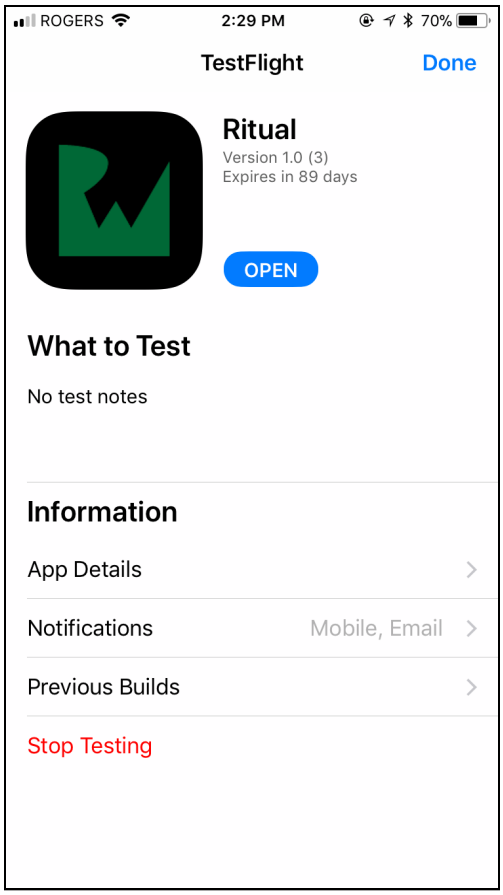


Every time you upload a build to TestFlight, your internal testers will get an email and they can see all previous unexpired builds in their TestFlight app. Builds live for 90 days before they expire.

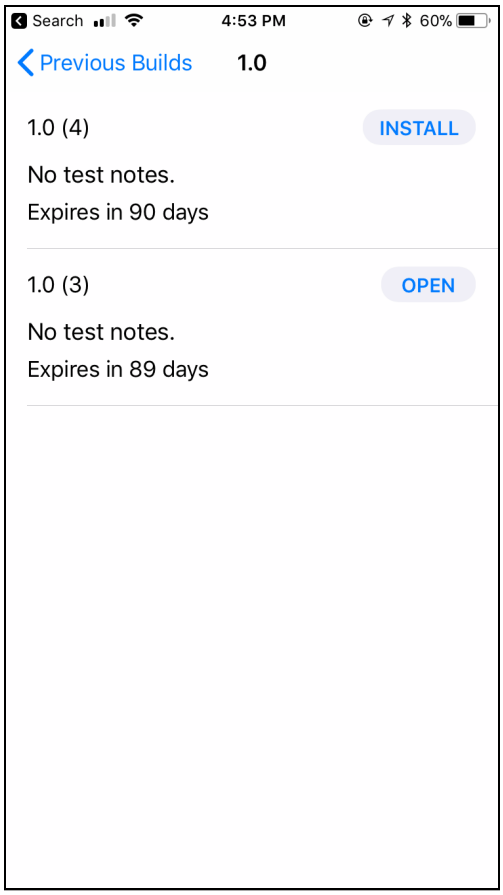
Download and open the TestFlight app on your phone. Make sure you are logged in to the same Apple id as your email that you added to the internal testers on iTunes Connect. Open the invitation email and tap on **View in TestFlight**. This will open your beta invite in Safari.



The invite gives you instructions to accept the testing invitation and provides you with a code. Once you redeem the code you will see your app's beta build.



You can view and install all previous builds that are within the 90 day expiration period.



5) That's it!

Congrats, by now you should be familiar with uploading builds to the App Store, testing your app through TestFlight!