Mofluid Magento™ Open Source eCommerce

User Guide

Plugin Installation Guide

Magento Plugin are simply great way of adding new features and functionality to your Magento powered stores. To install our Mofluid push notification feature plugin from magento connect you simply need a plugin .tgz file.

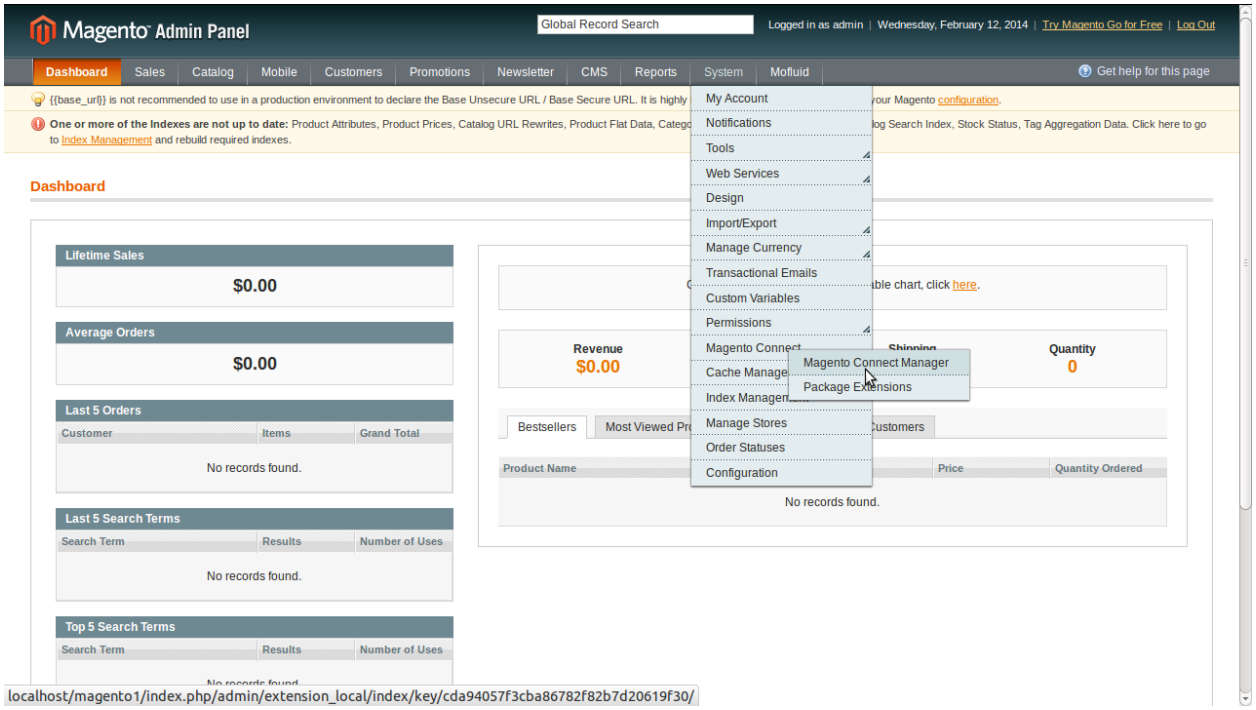
Follow the instructions below to install the plugin to your site.

Note: Before installing this plugin make sure that Mofluid_Mobile_App_Builder_V1.10.0 is already installed with in your Magento store.

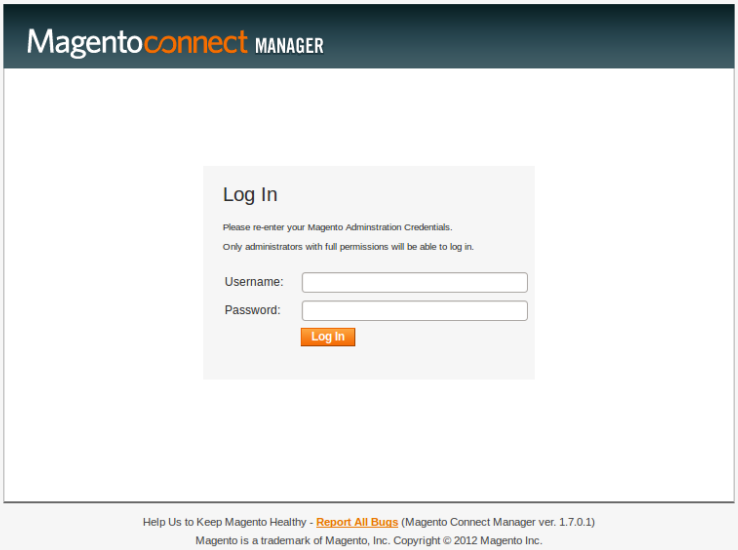
Steps to install Magento Extension Using Magento Connect

1. Sign into your Magento store admin panel.

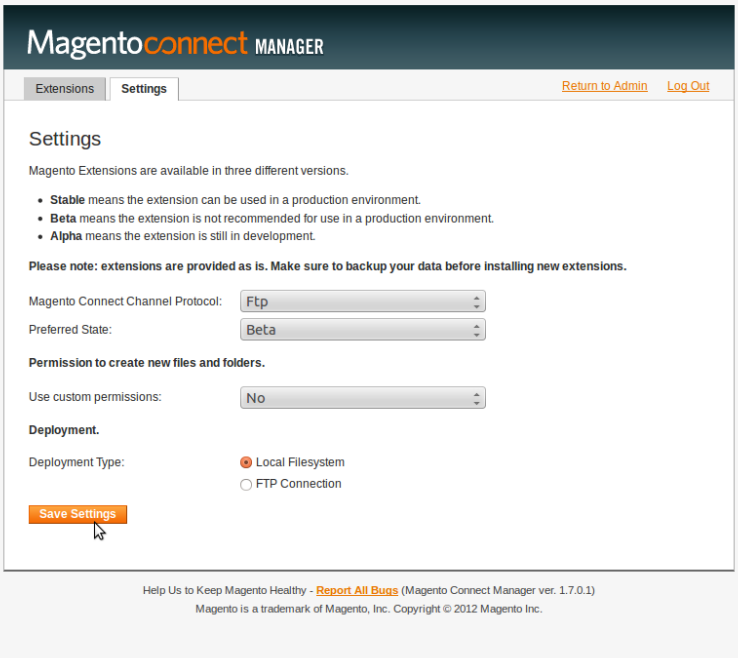
2. Navigate to System > Magento Connect > Magento Connect Manager.



3. Now Sign into Magento Connect Manager.



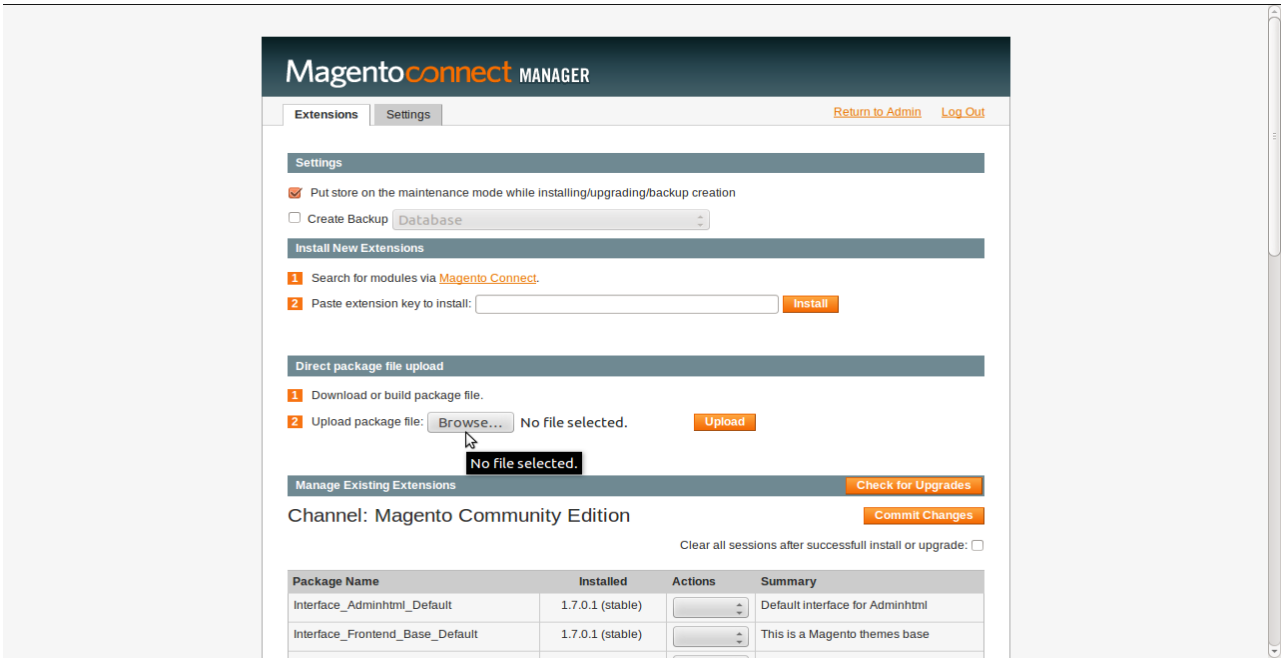
4. A window will open having two tabs Extensions and Settings.Now go to Setting select Magento Connect Channel Protocol as Ftp, Preferred State as Beta, Use custom permissions as No and Deployment Type as Local Filesystem, if Local file system Radio button is disabled then just give the write permission to your Magento Store whole folder Now just Click on Save Settings Button.

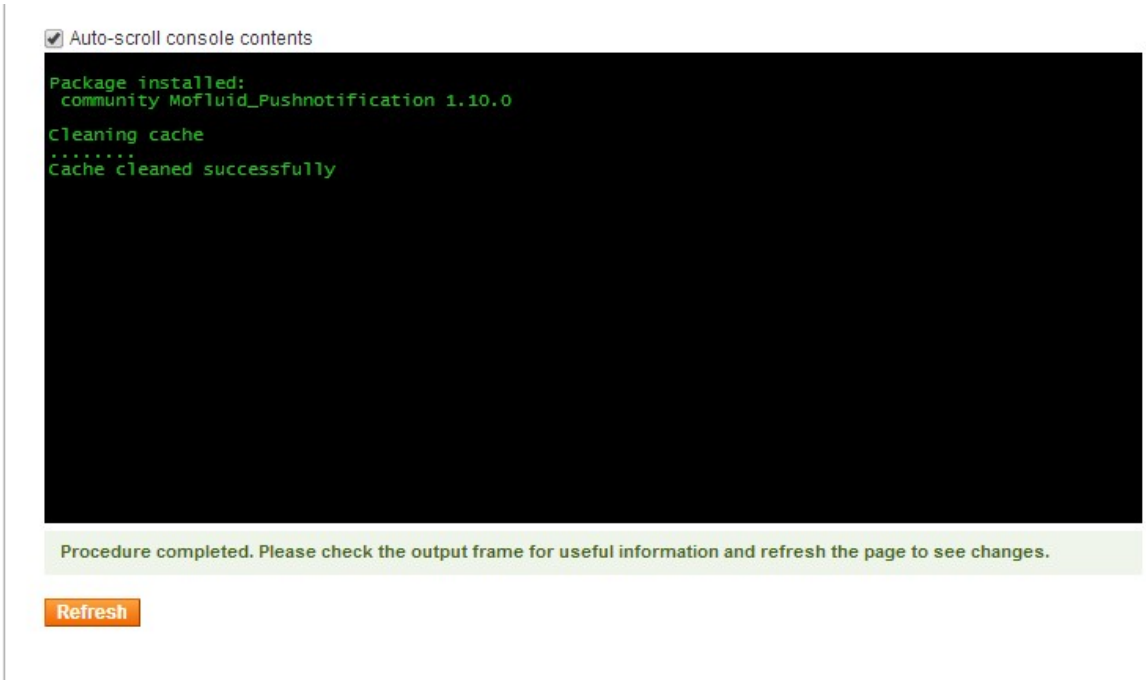


5.If you want to put your store on the maintenance mode while installing the Extension then

check the Put store on the maintenance mode while installing/upgrading/backup creation checkbox(Optional Step).

6.After saving settings click on Extension tab here in Direct package file Upload section you just have to upload a tgz file package of the extension and then click on Upload button.





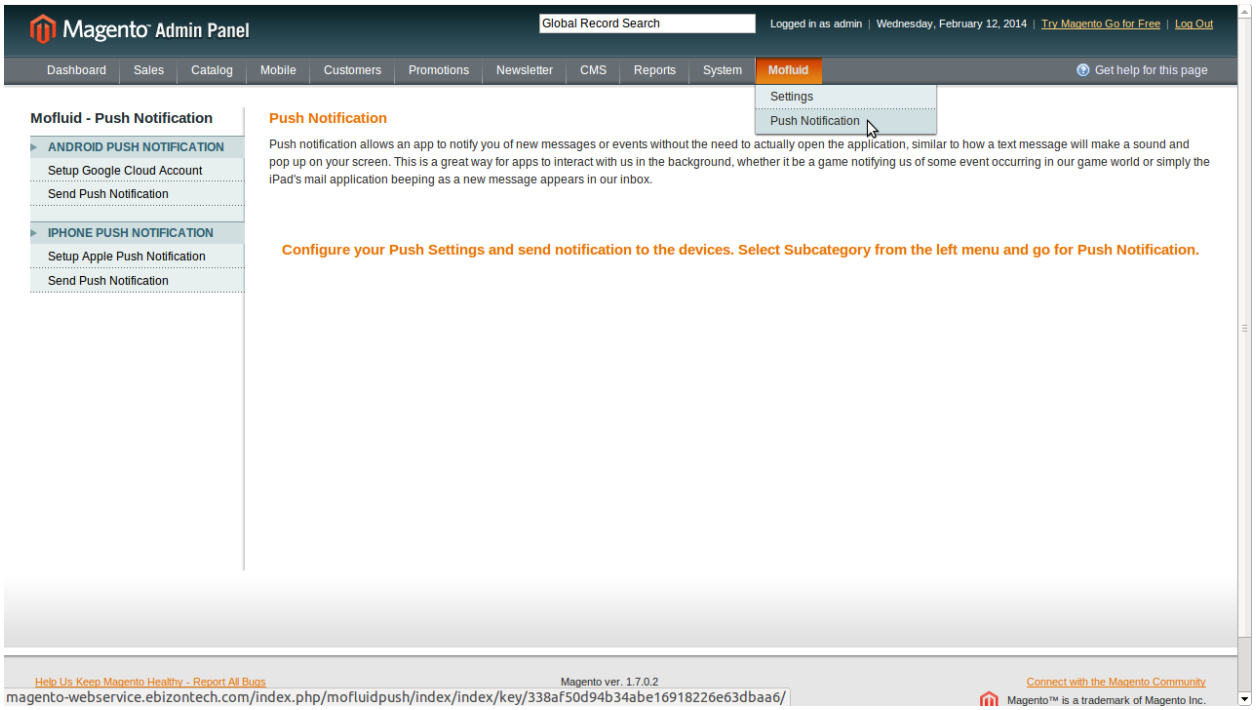
Above screen will be displayed after successful installation completed.

Now Return to admin panel you will see that a new extension with Push notification feature is installed on your Magento store.It will be displayed in main navigation bar under Mofluid>> Push notification.

Steps to enable Push notification for your app:

Push notification allows an app to notify you of new messages or events without the need to actually open the application, similar to how a text message will make a sound and pop up on your screen. This is a great way for apps to interact with us in the background, whether it be a game notifying us of some event occurring in our game world or simply the iPad's mail application beeping as a new message appears in our inbox.

Go to Mofluid menu and select Push Notification



Platform of Push Notification:

There are two platforms of Push Notification available according to your mobile app.They are

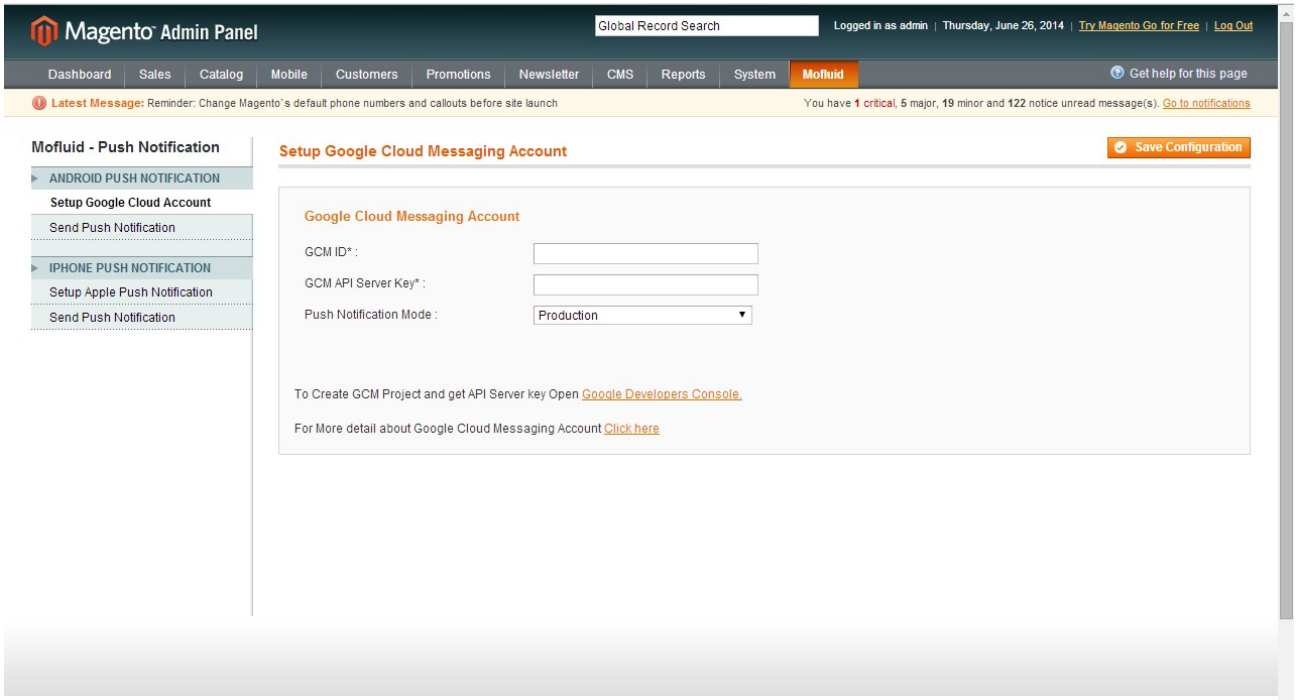
- ANDROID PUSH NOTIFICATION
- IPHONE PUSH NOTIFICATION

ANDROID PUSH NOTIFICATION

Google Cloud Messaging for Android (GCM) is a service that allows you to send data from your server to your users **Android-powered device**, and also to receive messages from devices on the same connection. The **GCM** service handles all aspects of queueing of messages and delivery to the target Android application running on the target device. **GCM** is completely free no matter how big your messaging needs are, and there are no quotas.

1. On the left side of the Push Notification page there is Android Push Notification here there is two option one is **Setup Google Cloud Account** and second is **Send Push Notification.**

2. Click on **Setup Google Cloud Account** .



3. Now fill the **GCM ID** and **GCM API Server Key** (**How to get your GCM ID and GCM API Server Key description is given below**) and select the mode between **Production** and **Development** and click on **Save Configuration**.

Production Mode :
If your app is already released in the market then you can use production mode for sending the push notification to the registered users.
Note : If your app is not released in market and you will use the production mode then push notification will not sent to the registered users.

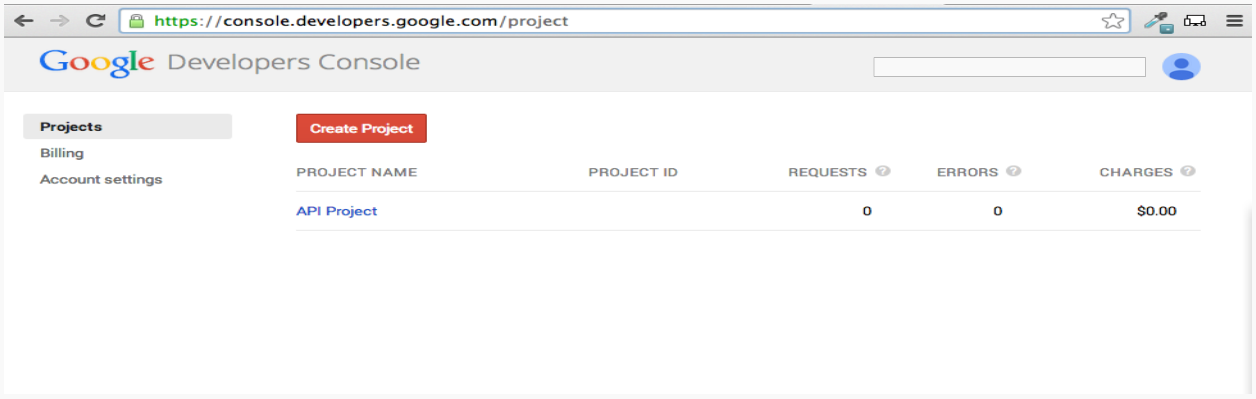
Development Mode :
If your app is not released in the market then you will use the development mode for sending the push notification to the registered users.

HOW TO GET YOUR GCM ID and GCM API Server Key :

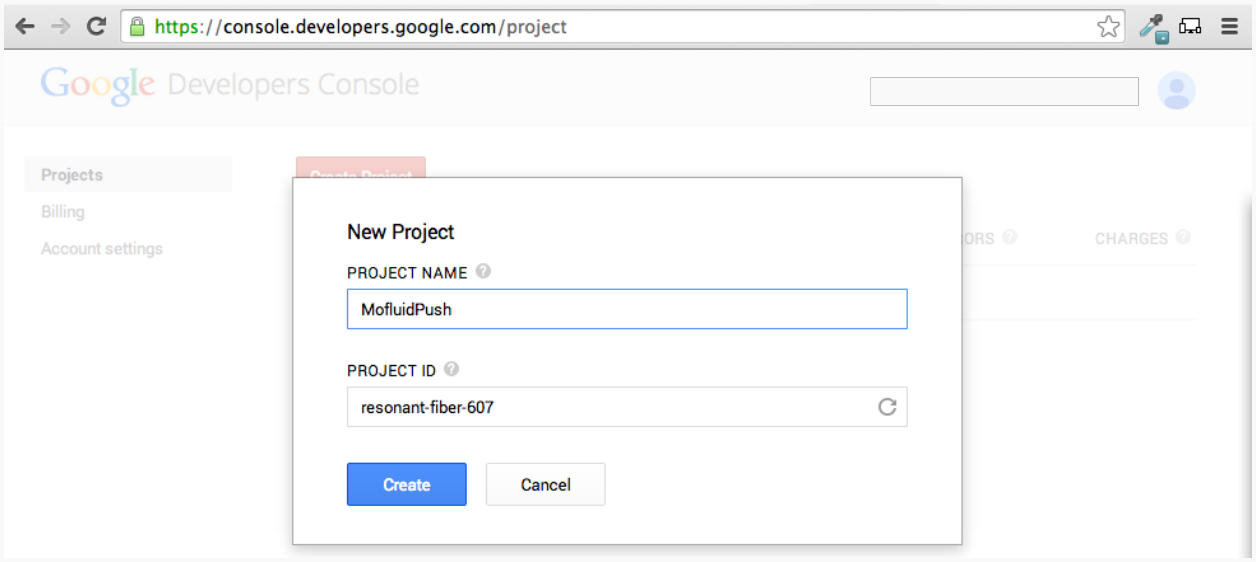
There are some steps to get the GCM ID which are given below :

1. **Creating a Google API Project :**

(i) Click on https://console.developers.google.com/projects link for open the Google Developer Console.

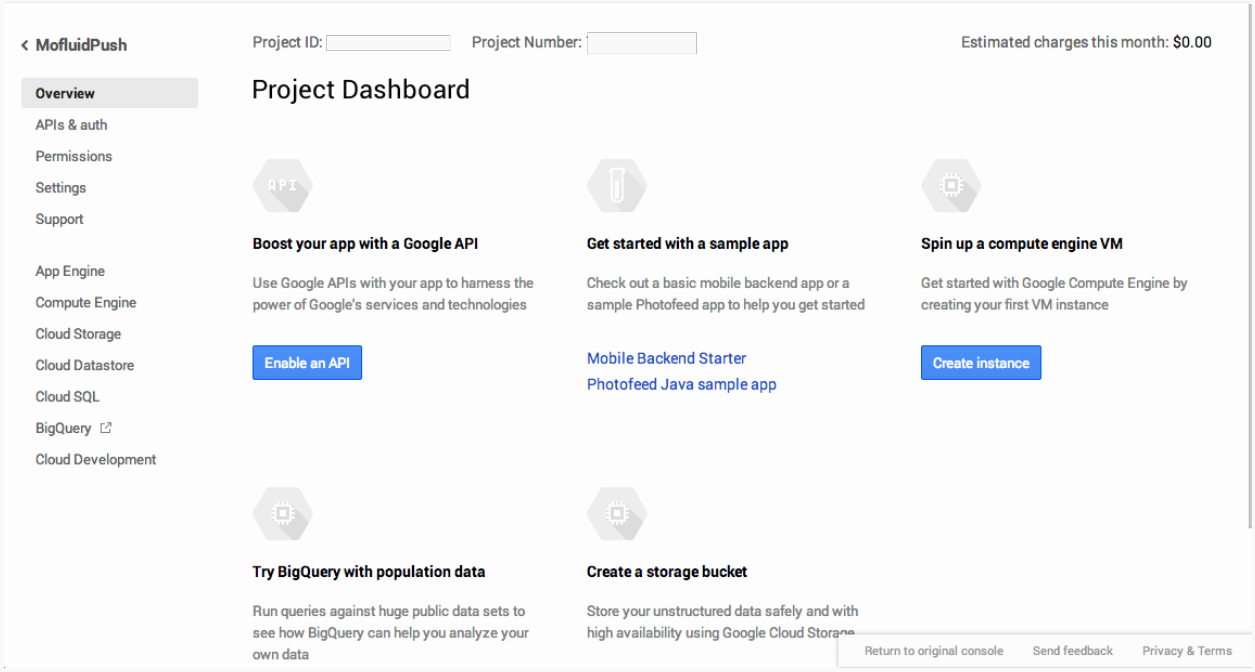


(ii) If you haven't created an API project yet, click **Create Project**.



(iii) Enter a project name and click **Create**.

Once the project has been created, a page appears that displays your project ID and project number. For example, Project Number: 123456789012.



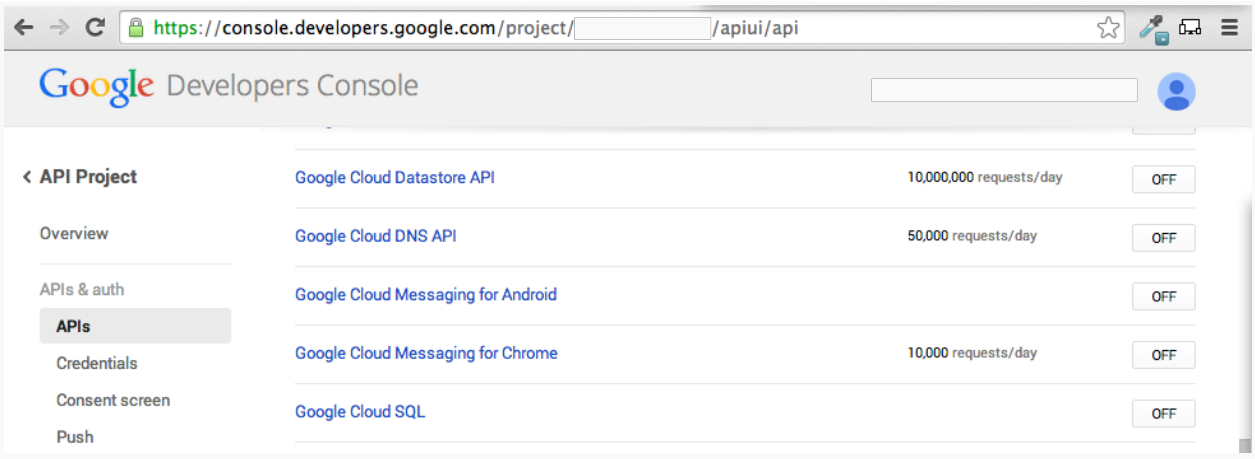
(iv) Copy down your project number. You will use it later on as the **GCM ID.**

2. Enabling the GCM Service :

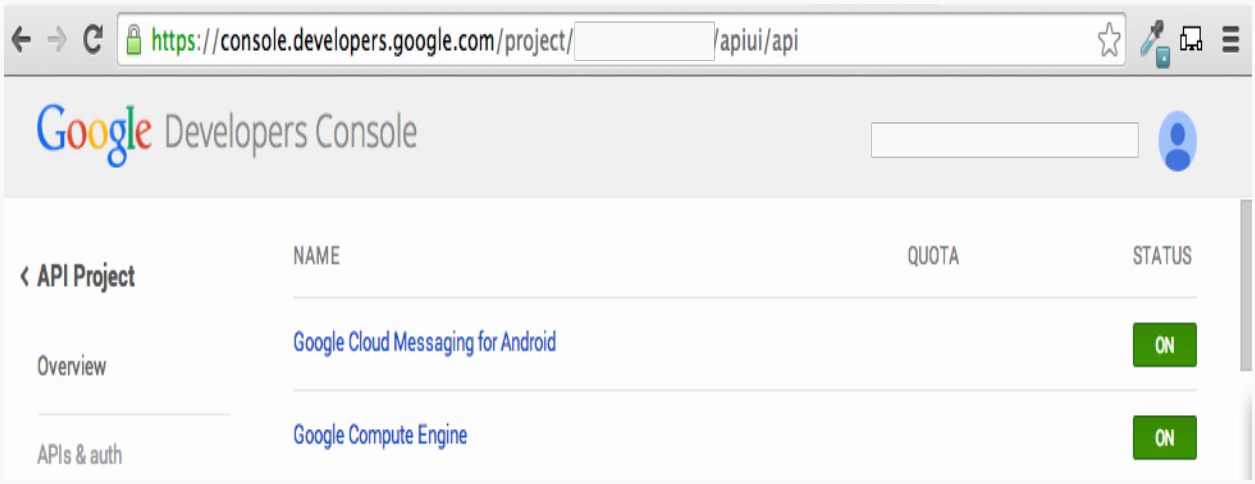
To enable the GCM service:

(i) In the sidebar on the left, select **APIs & auth**.

(ii) In the displayed list of APIs, **Google Cloud Messaging for Android** toggle is **OFF**.

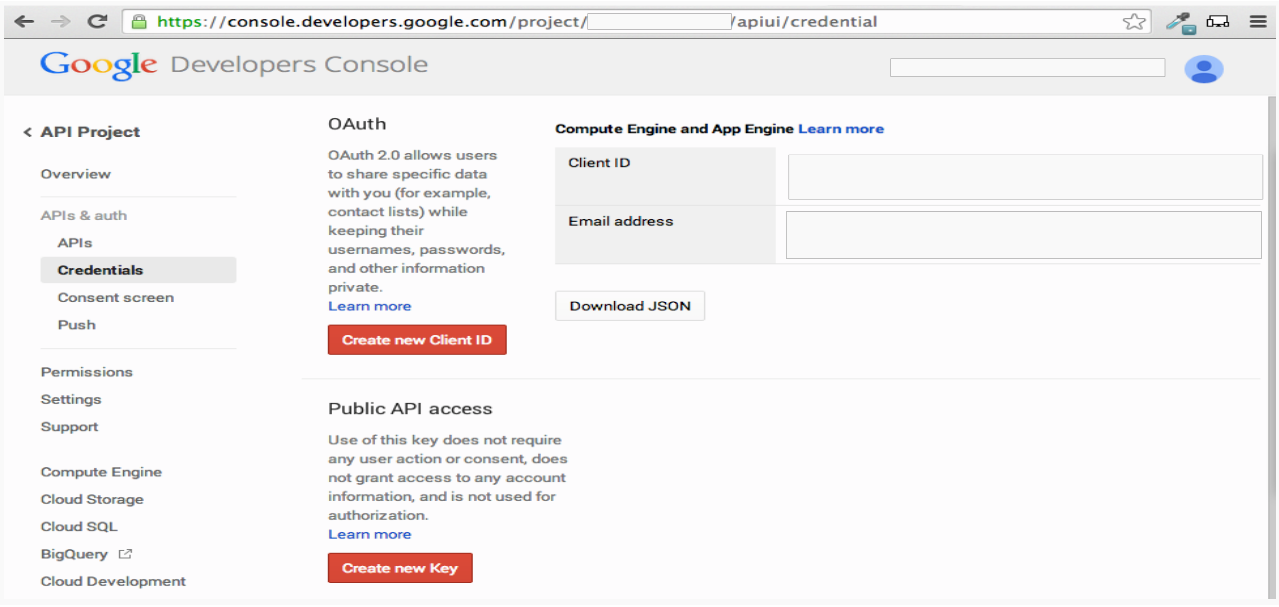


(iii) Now turn the **Google Cloud Messaging for Android** toggle to **ON**.

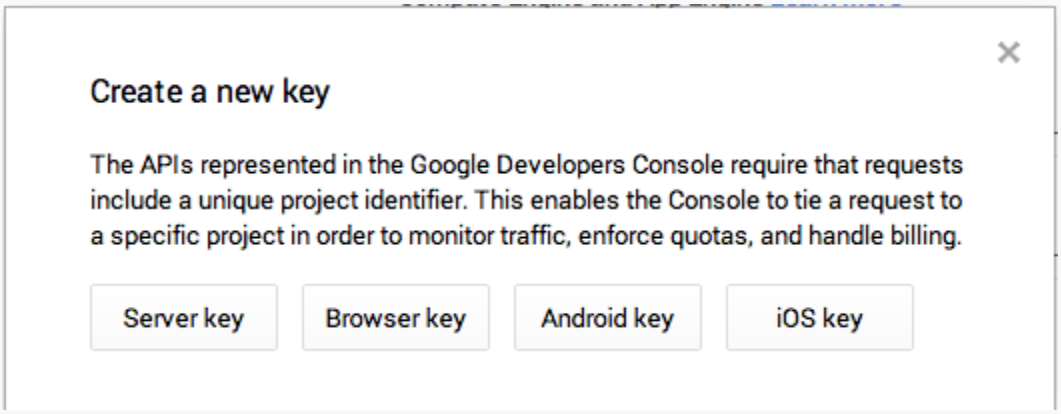


3. Obtaining an API Key :

(i) In the sidebar on the left, select **APIs & Auth >> Credentials**.

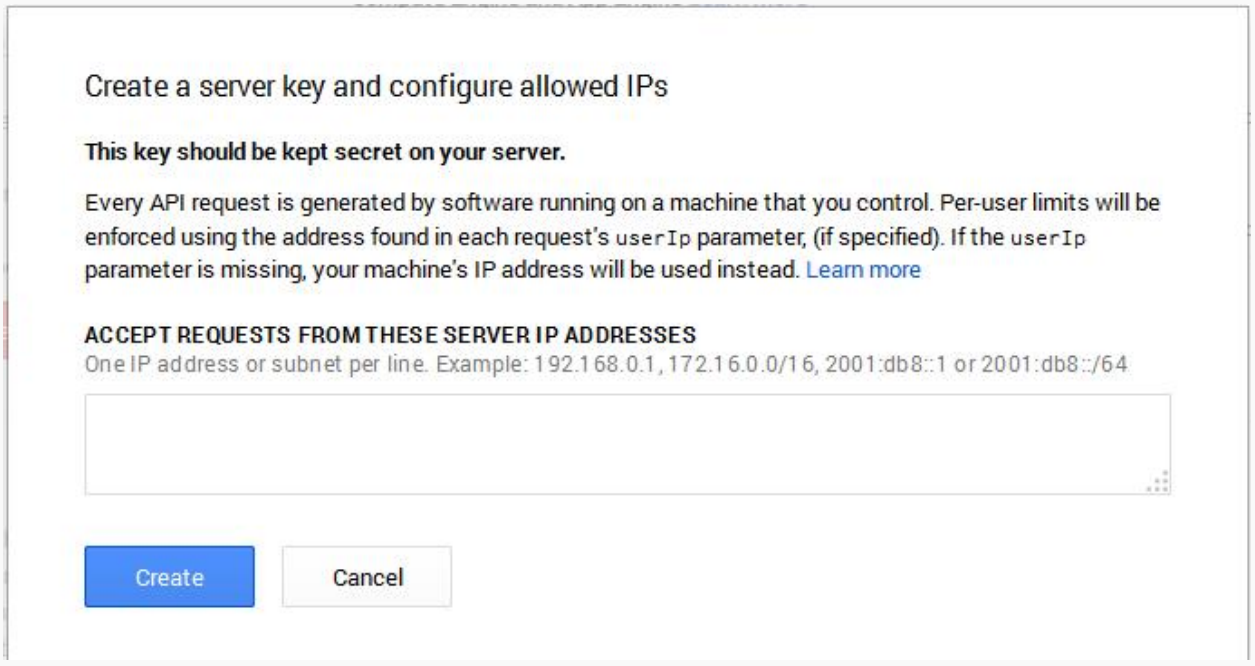


(ii) Under **Public API access**, click **Create new key**.



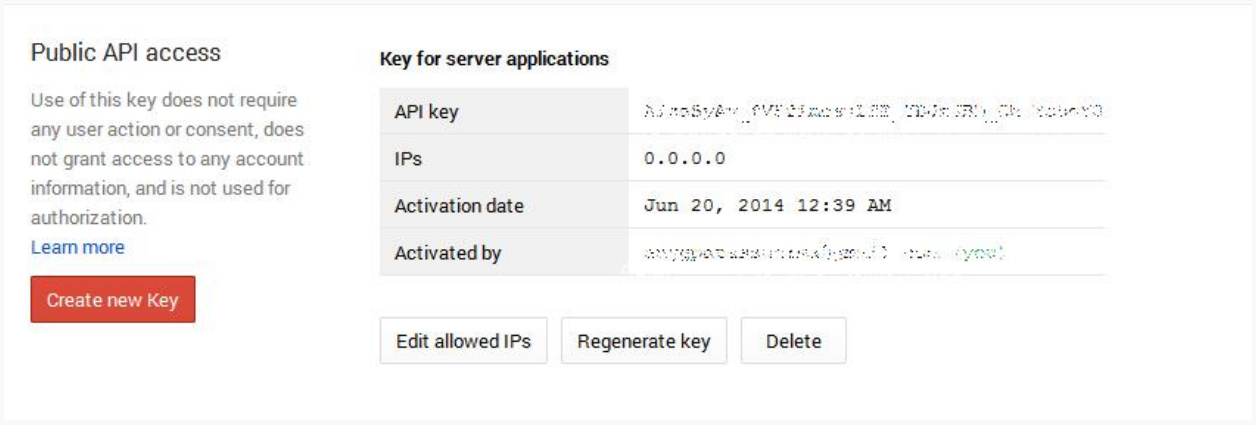
(iii) In the **Create a new key** dialog, click **Server key**.

(iv) In the resulting configuration dialog, supply your server's IP address. For testing purposes, you should leave it blank to allow any IPs.



(v) Click **Create**.

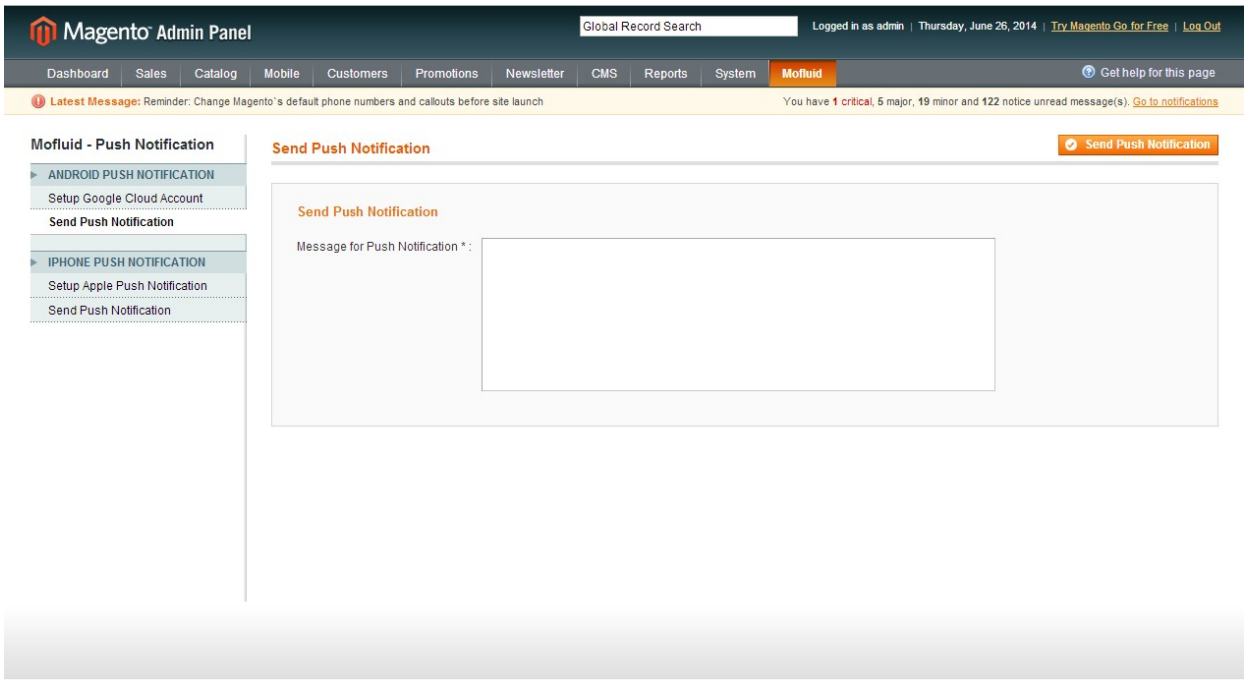
(vi) In the refreshed page, copy the API key. You will need the API key later on to perform authentication in your application server.



(vii) All steps for getting GCM ID and GCM API Server Key are completed. Fill GCM ID block and GCM API Server Key block corresponding to your GCM ID and GCM API Server Key.

4. Click on **Send Push Notification.** On clicking send Push Notification tab from left menu a page will open containing field **Message of Push Notification** here you enter message for Push Notification. When you click **Send Push Notification** Button a Notification will be send to all registered users.

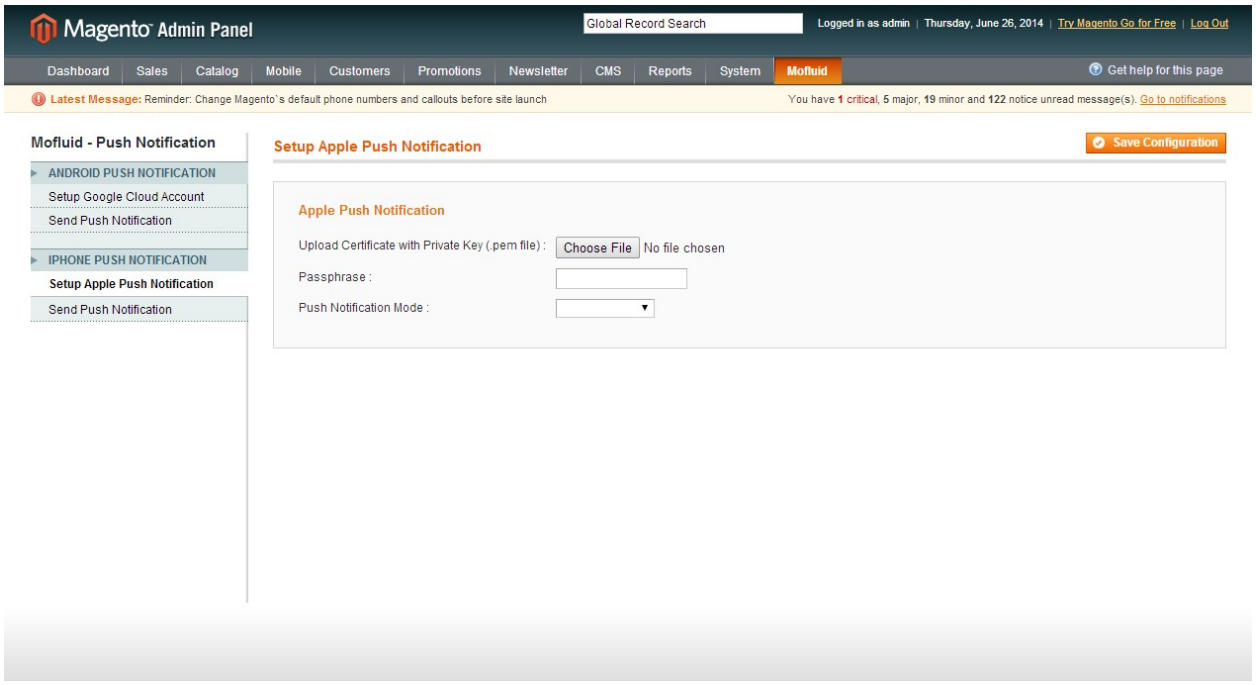
Note: Push notification should be enable in magento store using GCM ID and GCM API Server Key before generating android app for devices . Otherwise user will not receive any push notification send through magento store.



IPHONE PUSH NOTIFICATION

1. On the left side of the Push Notification page there is iPhone Push Notification. Here you will find two options within it one is **Setup Apple Push Notification** and other **Send Push Notification**.

2. Click on **Setup Apple Push Notification.**



3. On click that link that page will open with three fields. **Upload Certificate with Private Key (.pem file)** here you have to upload Certificate with Private Key a .pem file, **Passphrase** here you entered your passphrase (**How to get .pem file and passphrase description is given below after this point**) and **Push Notification mode** here you select your Push Notification mode between **Production** and **Developer** and then click on **save configuration button**.

How to get .pem file and passphrase :

Provisioning Profiles and Certificates:

To enable push notifications in your app, it needs to be signed with a provisioning profile that is configured for push. In addition, your server needs to sign its communications to APNS with an SSL certificate.

The provisioning profile and SSL certificate are closely tied together and are only valid for a single App ID. This is a protection that ensures only your server can send push notifications to instances of your app, and no one else.

As you know, apps use different provisioning profiles for development and distribution. There are also two types of push server certificates:

1. **Development** : If your app is running in Debug mode and is signed with the Development provisioning profile (Code Signing Identity is “iPhone Developer”), then your server must be using the Development certificate.

2. Production : Apps that are distributed as Ad Hoc or on the App Store (when Code Signing will Identify is “iPhone Distribution”) must talk to a server that uses the Production certificate. If there is a mismatch between these, push notifications cannot be delivered to your app.

Generating the Certificate Signing Request (CSR):

Remember how you had to go to the iOS Provisioning Portal and make a Development Certificate after you signed up for the iOS Developer Program? If so, then these next steps should be familiar. Still, I advise you to follow them exactly. Most of the problems people have with getting push notifications to work are due to problems with the certificates.

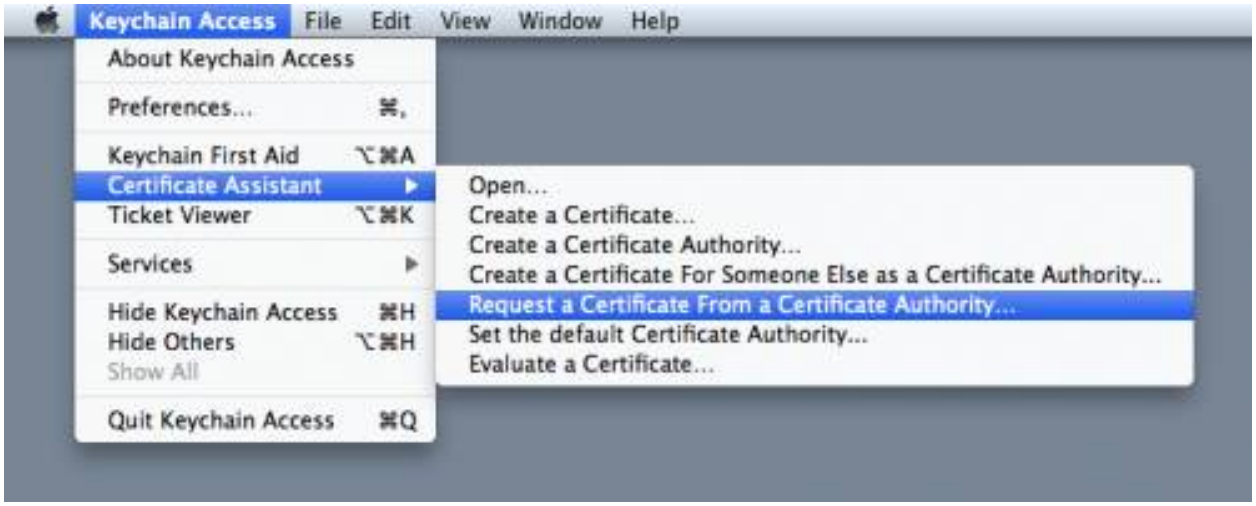
Digital certificates are based on public-private key cryptography. You don’t need to know anything about cryptography to use certificates, but you do need to be aware that a certificate always works in combination with a private key.

The certificate is the public part of this key pair. It is safe to give it to others, which is exactly what happens when you communicate over SSL. The private key, however, should be kept… private. It’s a secret. Your private key is nobody’s business but your own. It’s important to know that you can’t use the certificate if you don’t have the private key.

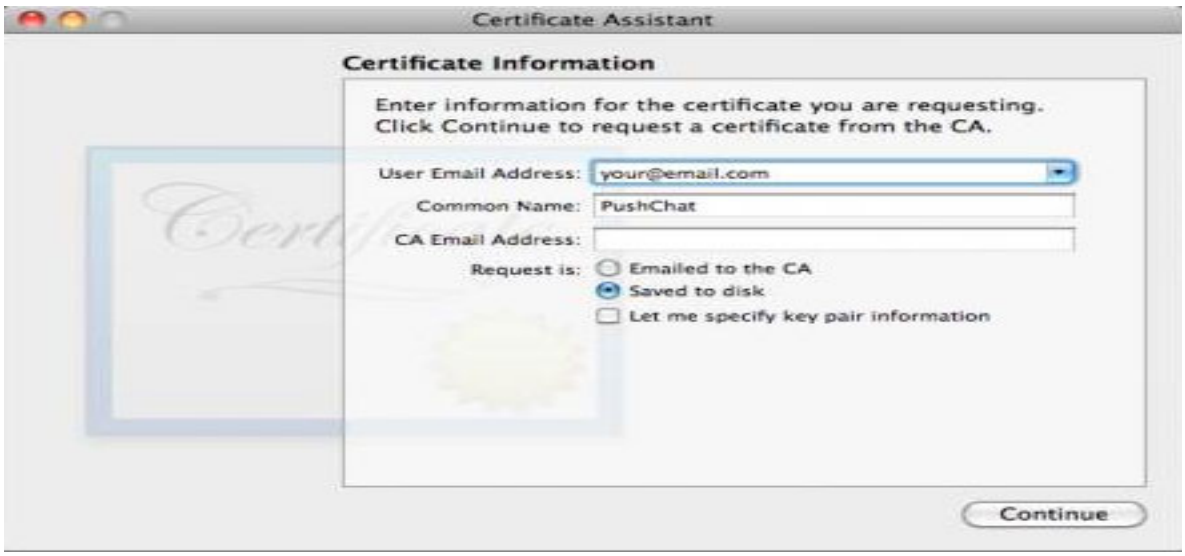
Whenever you apply for a digital certificate, you need to provide a Certificate Signing Request, or

CSR for short. When you create the CSR, a new private key is made that is put into your keychain. You then send the CSR to a certificate authority (in this case that is the iOS Developer Portal), which will generate the SSL certificate for you based on the information in the CSR.
Open Keychain Access on your Mac (it is in Applications/Utilities) and choose the menu option

Request a Certificate from a Certificate Authority….



If you do not have this menu option or it says “Request a Certificate from a Certificate Authority with key”, then download and install the WWDR Intermediate Certificate first. Also make sure no private key is selected in the main Keychain Access window.
You should now see the following window:

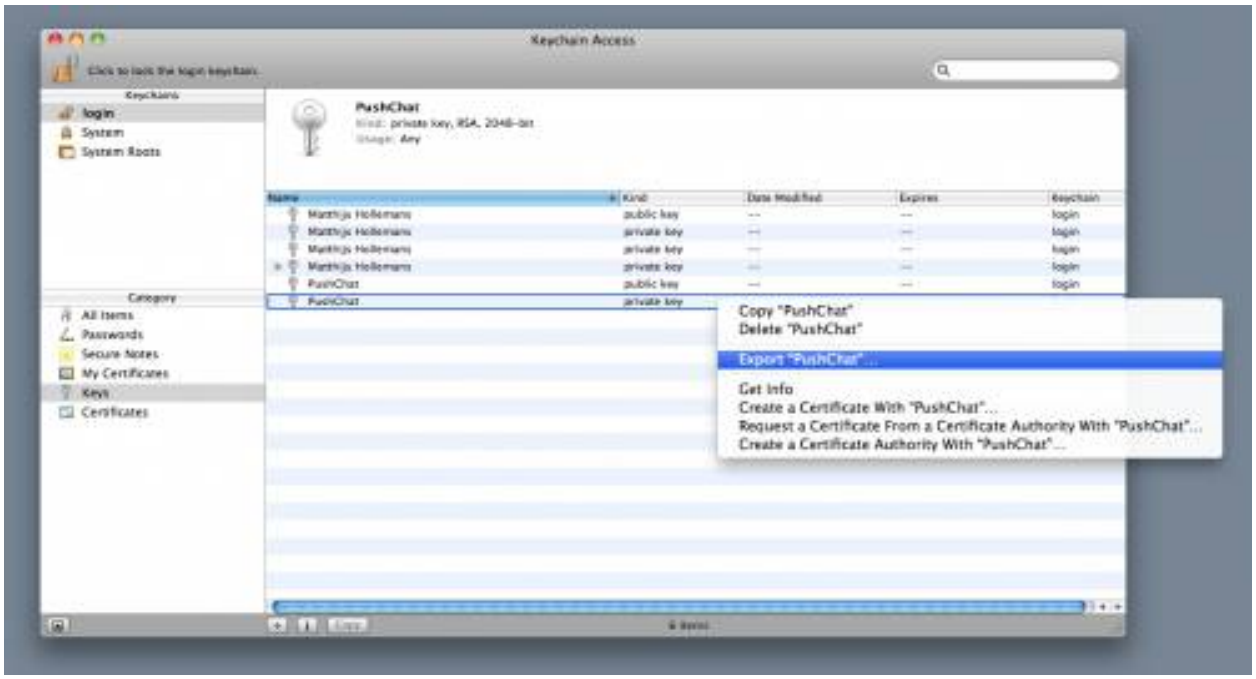


Enter your email address here. I've heard people recommended you use the same email address that you used to sign up for the iOS Developer Program, but it seems to accept any email address just fine.

Enter "PushChat" for Common Name. You can type anything you want here, but choose something descriptive. This allows us to easily find the private key later.

Check **Saved to disk** and click **Continue**. Save the file as "PushChat.certSigningRequest".

If you go to the Keys section of Keychain Access, you will see that a new private key has appeared in your keychain. Right click it and choose Export.

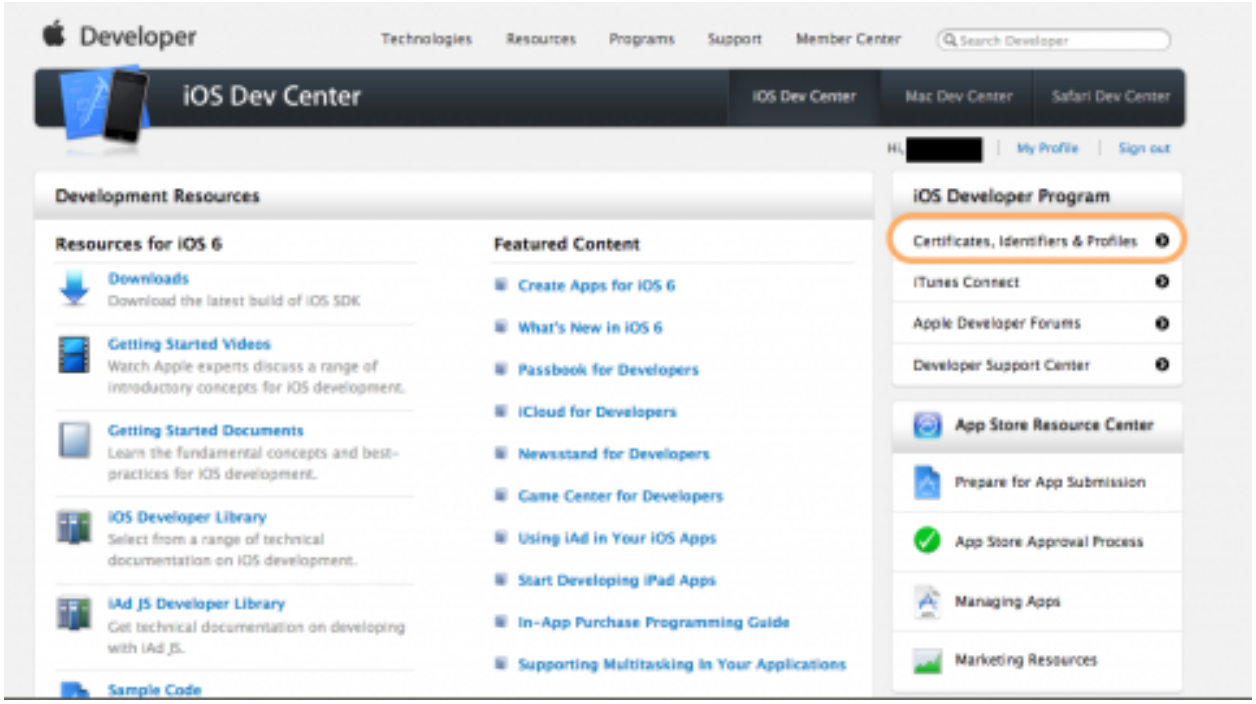


Save the private key as **PushChatKey.p12** and enter a passphrase.

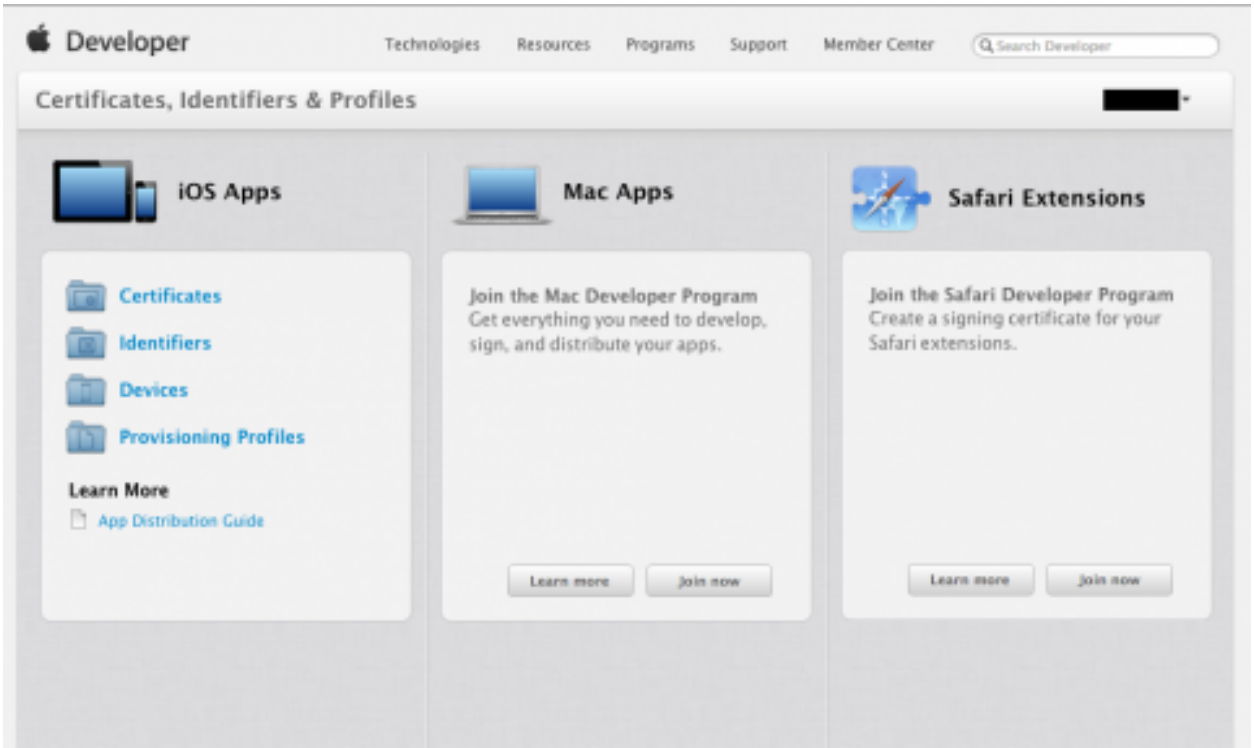
For the convenience of this tutorial, I used the passphrase "pushchat" to protect the p12 file but you should really choose something that is less easy to guess. The private key needs to be a secret, remember? Do choose a passphrase that you can recall, or you won't be able to use the private key later.

Making the App ID and SSL Certificate:

Log in to the iOS Dev Center and "Select the Certificates, Identifiers and Profiles" from the right panel.



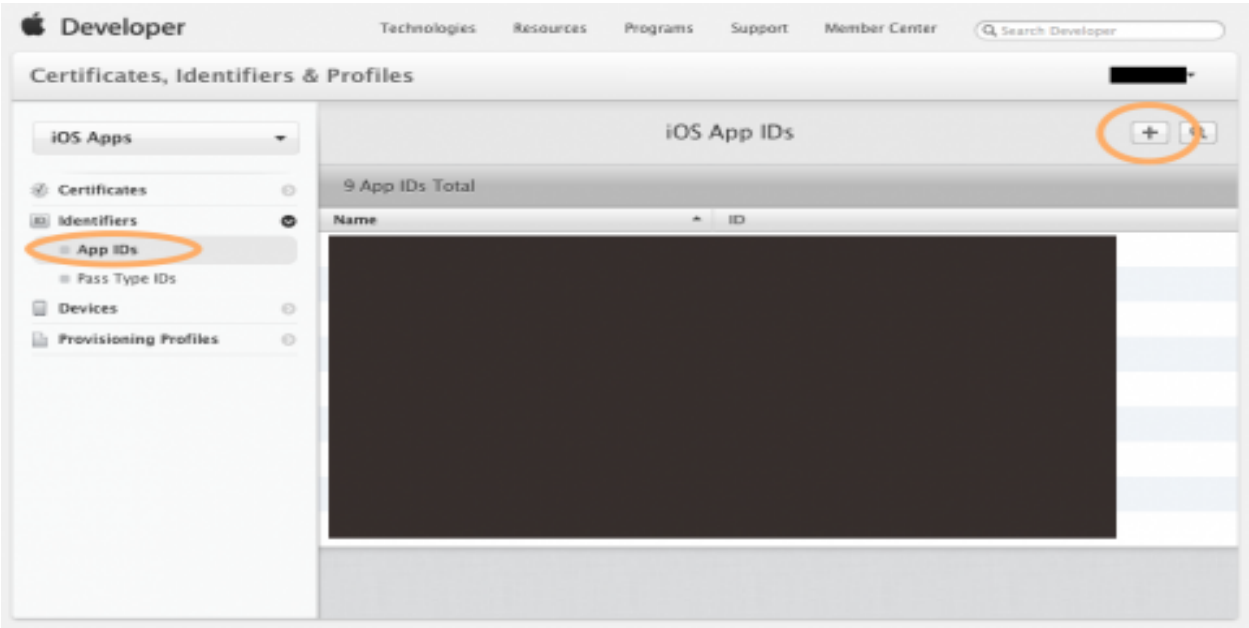
You will be presented with the following screen (Doesn't the new dev center UI look sleek)



Since you're making an iOS app select **Certificates** in the **iOS Apps** section.

Now, you are going to make a new App ID. Each push app needs its own unique ID because push notifications are sent to a specific application. (You cannot use a wildcard ID.)

Go to **App IDs** in the sidebar and click the **+** button.

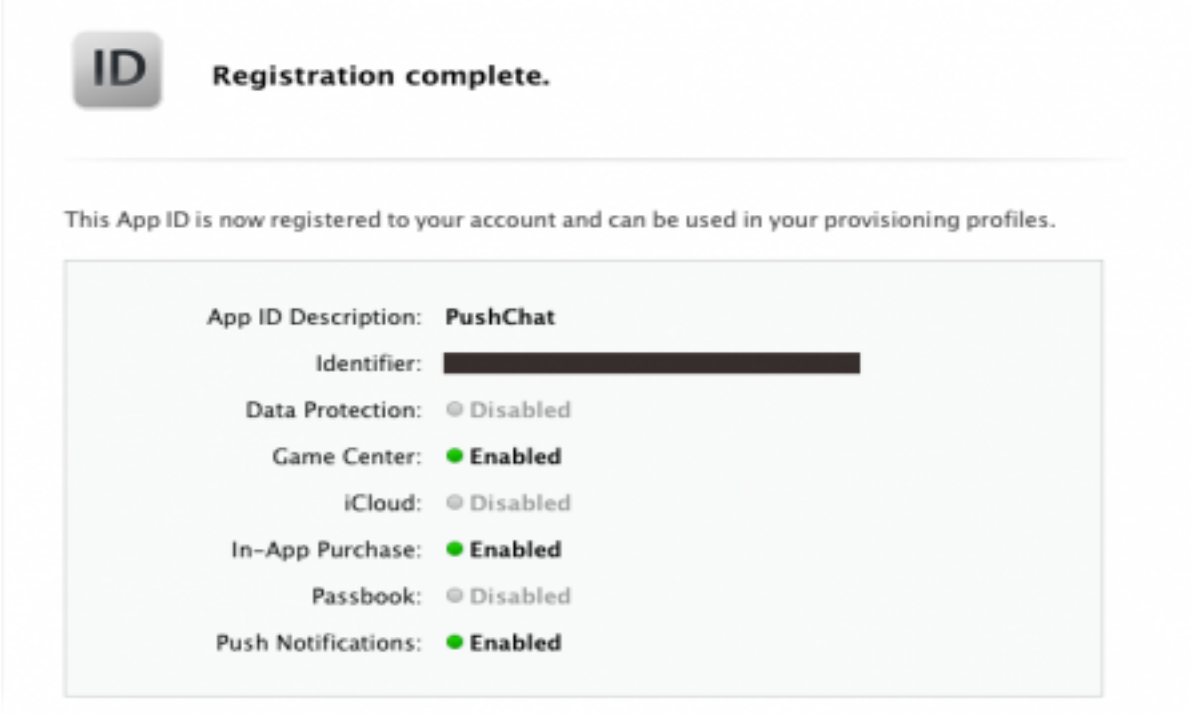


Fill the following details:

- **App ID Description**: PushChat
- **App Services** Check the Push Notifications Checkbox
- **Explicit App ID**: com.hollance.PushChat

It is probably best if you choose your own Bundle Identifier here – com.yoursite.PushChat – instead of using mine. You will need to set this same bundle ID in your Xcode project. After you're done filling all the details press the **Continue** button. You will be asked to verify the details of the app id, if everything seems okay click **Submit**

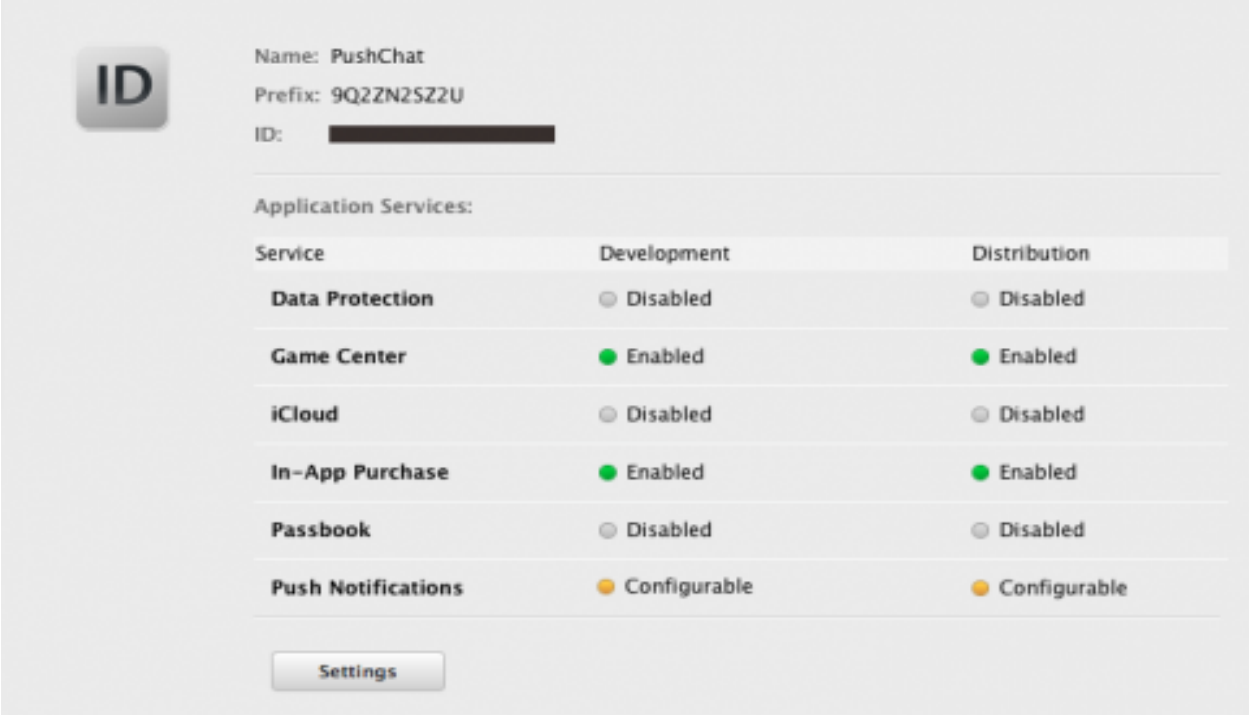
Hurray! You have successfully registered a new App ID.



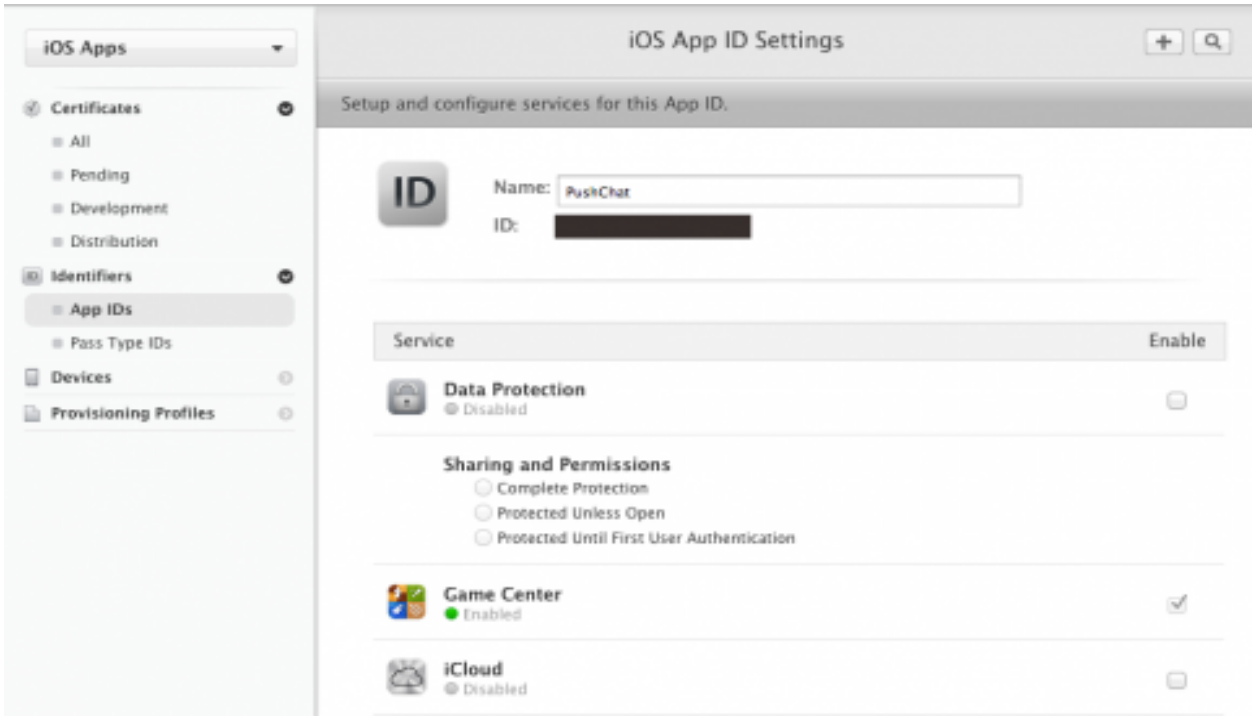
In a few moments, you will generate the SSL certificate that your push server uses to make a secure connection to APNS. This certificate is linked with your App ID. Your server can only send push notifications to that particular app, not to any other apps.

After you have made the App ID, it shows up like this in the list:

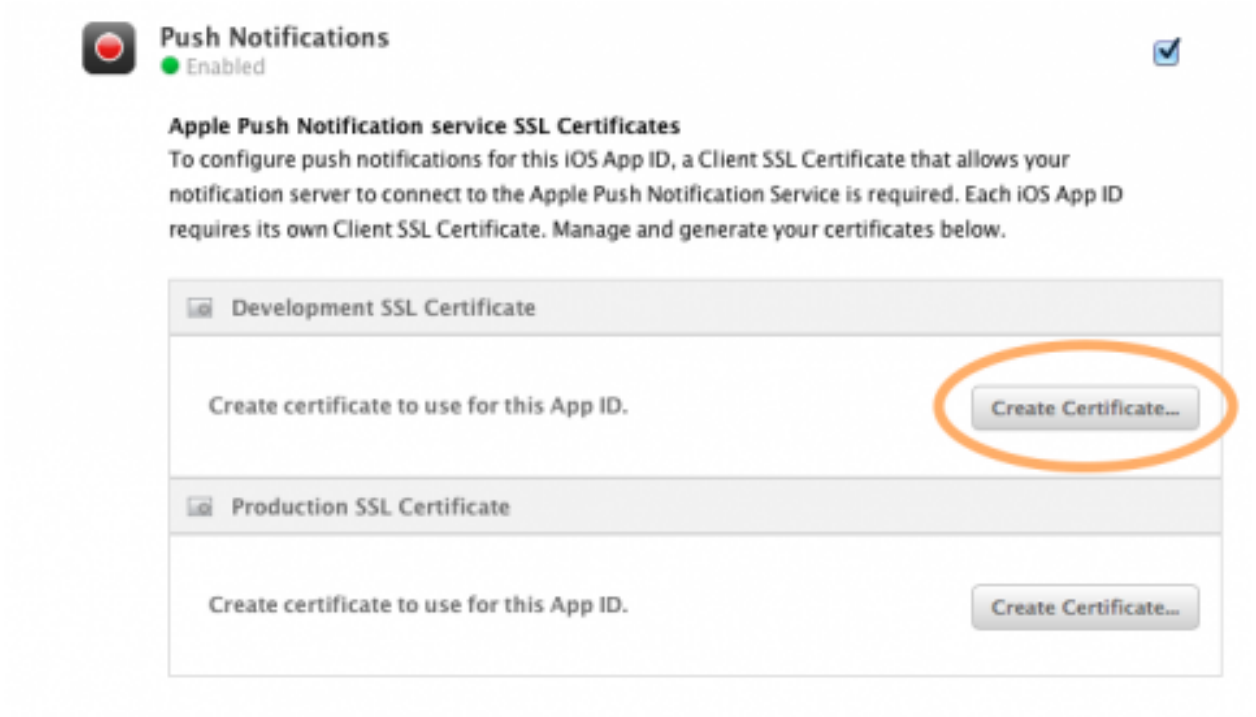
Select the **PushChat** app ID from the list. This will open up an accordion as shown below:



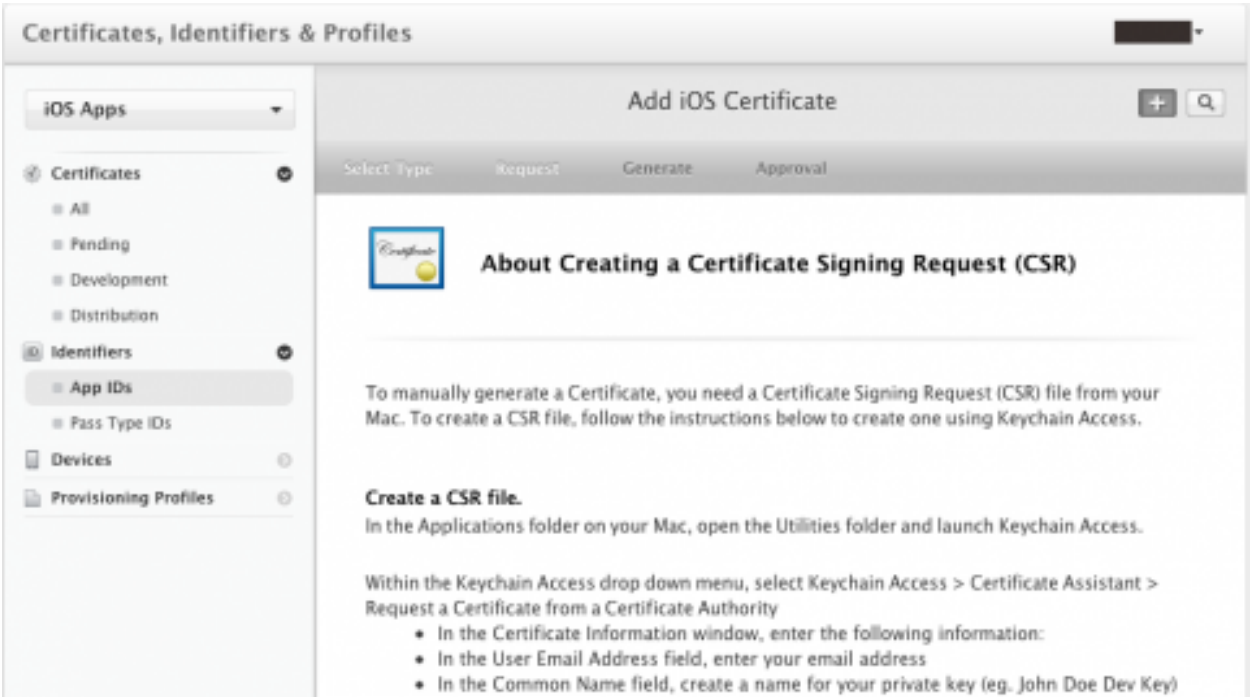
Notice in the “Push Notification” row, there are two orange lights that say “Configurable” in the Development and Distribution column. This means your App ID can be used with push, but you still need to set this up. Click on the **Setting** button to configure these settings.



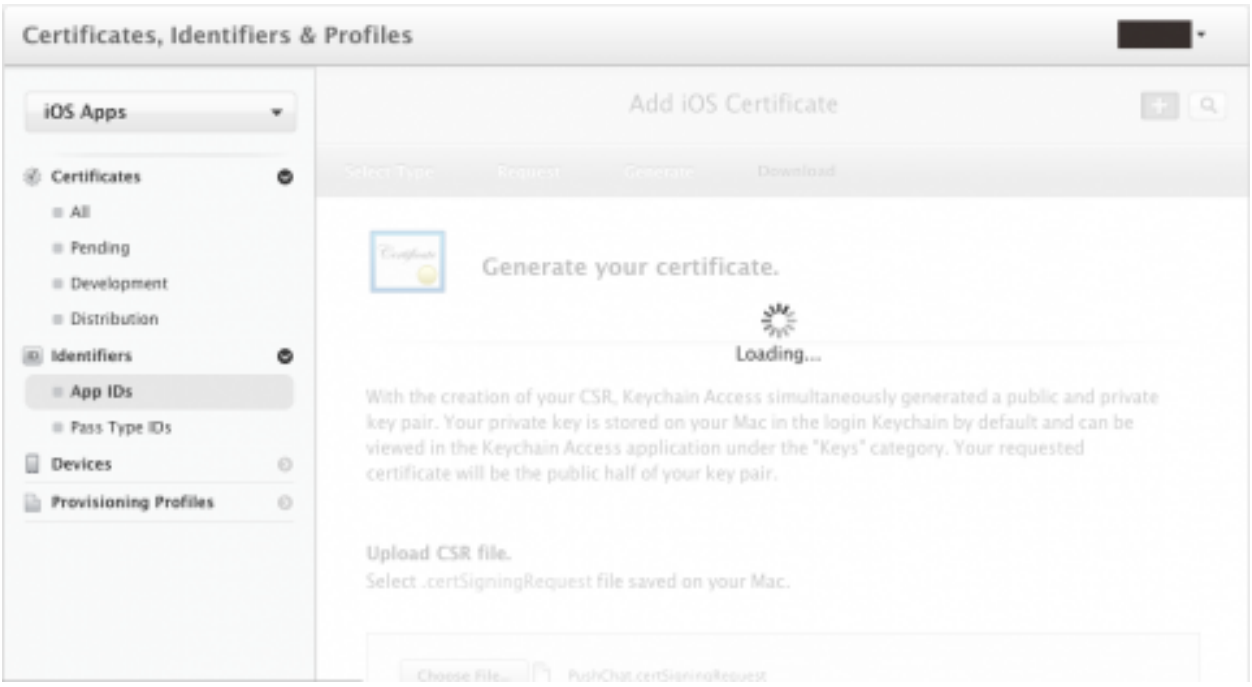
Scroll down to the Push Notifications section and select the **Create Certificate** button in the **Development SSL Certificate** section.



The “Add iOS Certificate” wizard comes up:

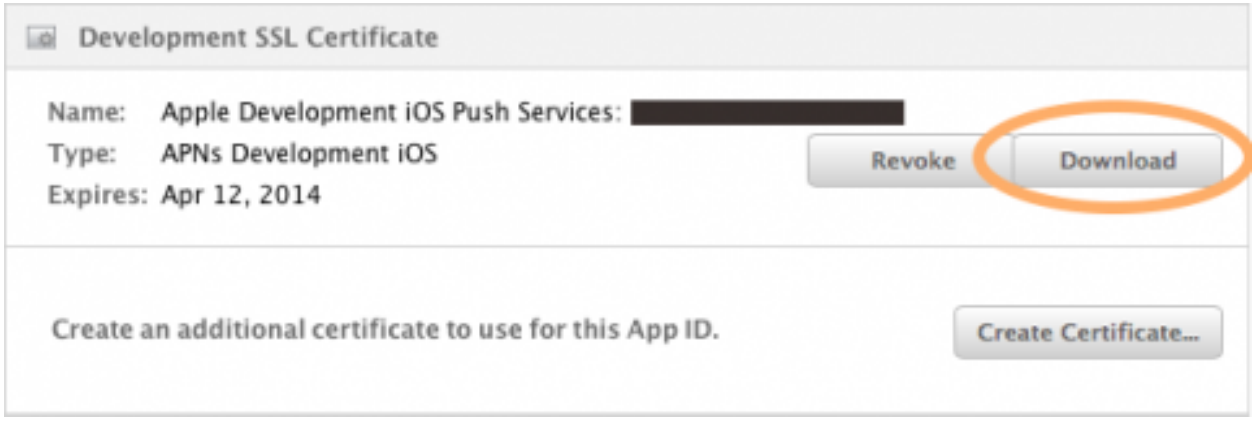


The first thing it asks you is to generate a Certificate Signing Request. You already did that, so click **Continue**. In the next step you upload the CSR. Choose the CSR file that you generated earlier and click **Generate**.



It takes a few seconds to generate the SSL certificate. Click **Continue** when it’s done.

Now click **Download** to get the certificate – it is named “aps_development.cer”.



As you can see, you have a valid certificate and push is now available for development. You can download the certificate again here if necessary. The development certificate is only valid for 3 months.

When you are ready to release your app, repeat this process for the production certificate. The steps are the same.

Note: The production certificate remains valid for a year, but you want to renew it before the year is over to ensure there is no downtime for your app.

You don’t have to add the certificate to your Keychain, although you could if you wanted to by double-clicking the downloaded **aps_development.cer** file. If you do, you’ll see that it is now associated with the private key.

Making a PEM File:

So now you have three files:

- The CSR
- The private key as a p12 file (PushChatKey.p12)
- The SSL certificate, aps_development.cer

Store these three files in a safe place. You could throw away the CSR but in my opinion it is easier to keep it. When your certificate expires, you can use the same CSR to generate a new one. If you were to generate a new CSR, you would also get a new private key. By re-using the CSR you can keep using your existing private key and only the .cer file will change.

You have to convert the certificate and private key into a format that is more usable. Because the

push part of our server will be written in PHP, you will combine the certificate and the private key into a single file that uses the PEM format.

The specifics of what PEM is doesn't really matter (in fact, I have no idea) but it makes it easier for PHP to use the certificate. If you write your push server in another language, these following steps may not apply to you.

You're going to use the command-line OpenSSL tools for this. Open a Terminal and execute the following steps.

Go to the folder where you downloaded the files, in my case the Desktop:

$ cd ~/Desktop/

Convert the .cer file into a .pem file:

$ openssl x509 -in aps_development.cer -inform der -out PushChatCert.pem

Convert the private key's .p12 file into a .pem file:

$ openssl pkcs12 -nocerts -out PushChatKey.pem -in PushChatKey.p12
Enter Import Password:
MAC verified OK
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:

You first need to enter the passphrase for the .p12 file so that openssl can read it. Then you need to enter a new passphrase that will be used to encrypt the PEM file. Again for this tutorial I used "pushchat" as the PEM pass phrase. You should choose something more secure.

Note: if you don't enter a PEM pass phrase, openssl will not give an error message but the generated .pem file will not have the private key in it.

Finally, combine the certificate and key into a single .pem file:

$ cat PushChatCert.pem PushChatKey.pem > ck.pem

At this point it's a good idea to test whether the certificate works. Execute the following command:

$ telnet gateway.sandbox.push.apple.com 2195
Trying 17.172.232.226...
Connected to gateway.sandbox.push-apple.com.akadns.net.
Escape character is '^]'.

This tries to make a regular, unencrypted, connection to the APNS server. If you see the above response, then your Mac can reach APNS. Press Ctrl+C to close the connection. If you get an error message, then make sure your firewall allows outgoing connections on port 2195.

Let's try connecting again, this time using our SSL certificate and private key to set up a secure connection:

$ openssl s_client -connect gateway.sandbox.push.apple.com:2195 -cert PushChatCert.pem -key PushChatKey.pem

Enter passphrase for PushChatKey.pem:

You should see a whole bunch of output, which is openssl letting you know what is going on under the hood.

If the connection is successful, you should be able to type a few characters. When you press enter, the server should disconnect. If there was a problem establishing the connection, openssl will give you an error message but you may have to scroll up through the output to find it.

Note: There are two different APNS servers: the "sandbox" server that you can use for testing, and the live server that you use in production mode. Above, we used the sandbox server because our certificate is intended for development, not production use.

4. Click on **Send Push Notification.** On selecting **Send Push Notification** a page will open containing Message of Push Notification here you should enter message of Push Notification. When you click Send Push Notification Button a Notification will be send to all registered users.