



# ADVANCED iOS APPLICATION DEVELOPMENT

---

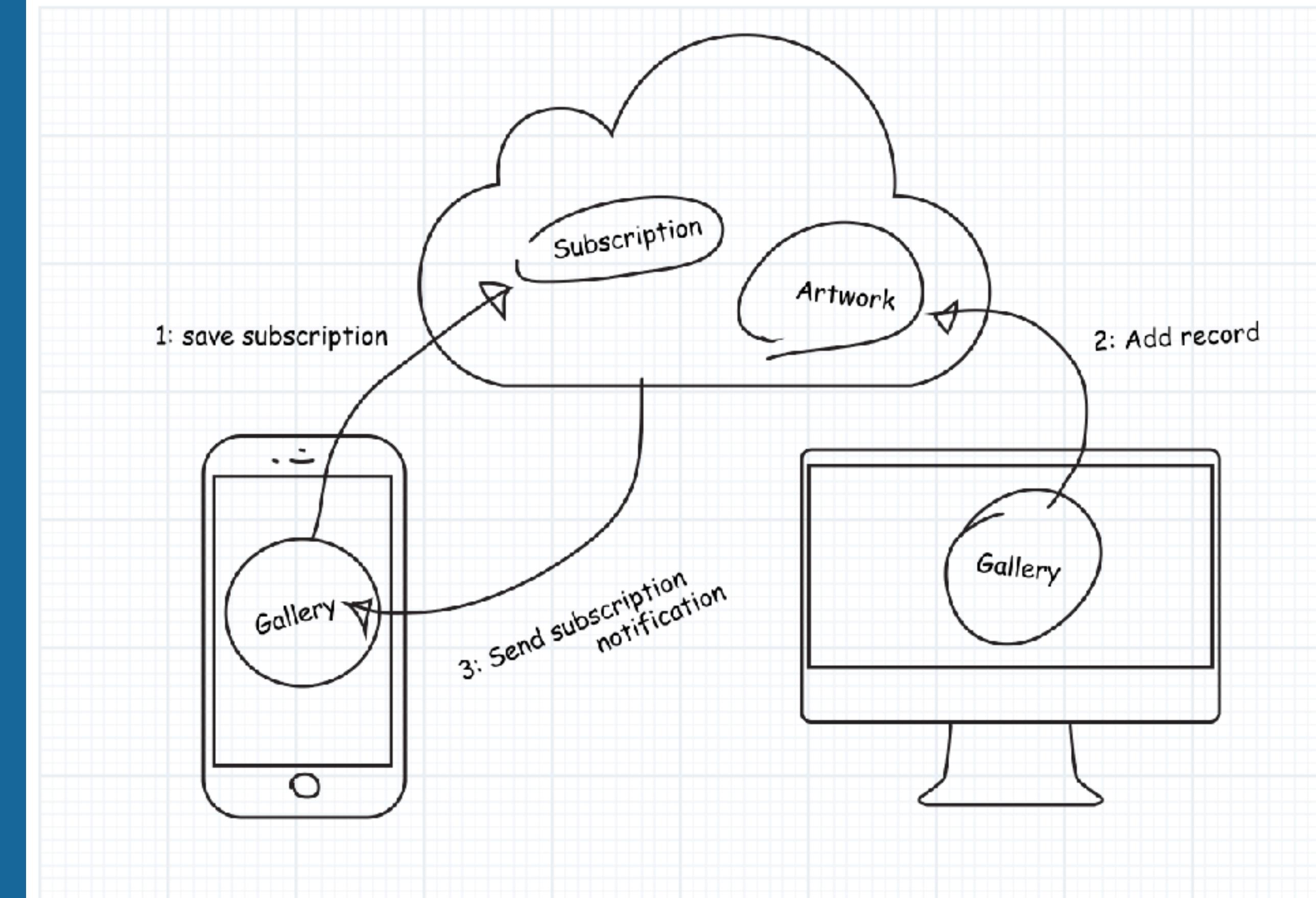
MPCS 51032 • SPRING 2020 • SESSION 3

# SUBSCRIPTIONS

# SUBSCRIPTIONS

SUBTITLE

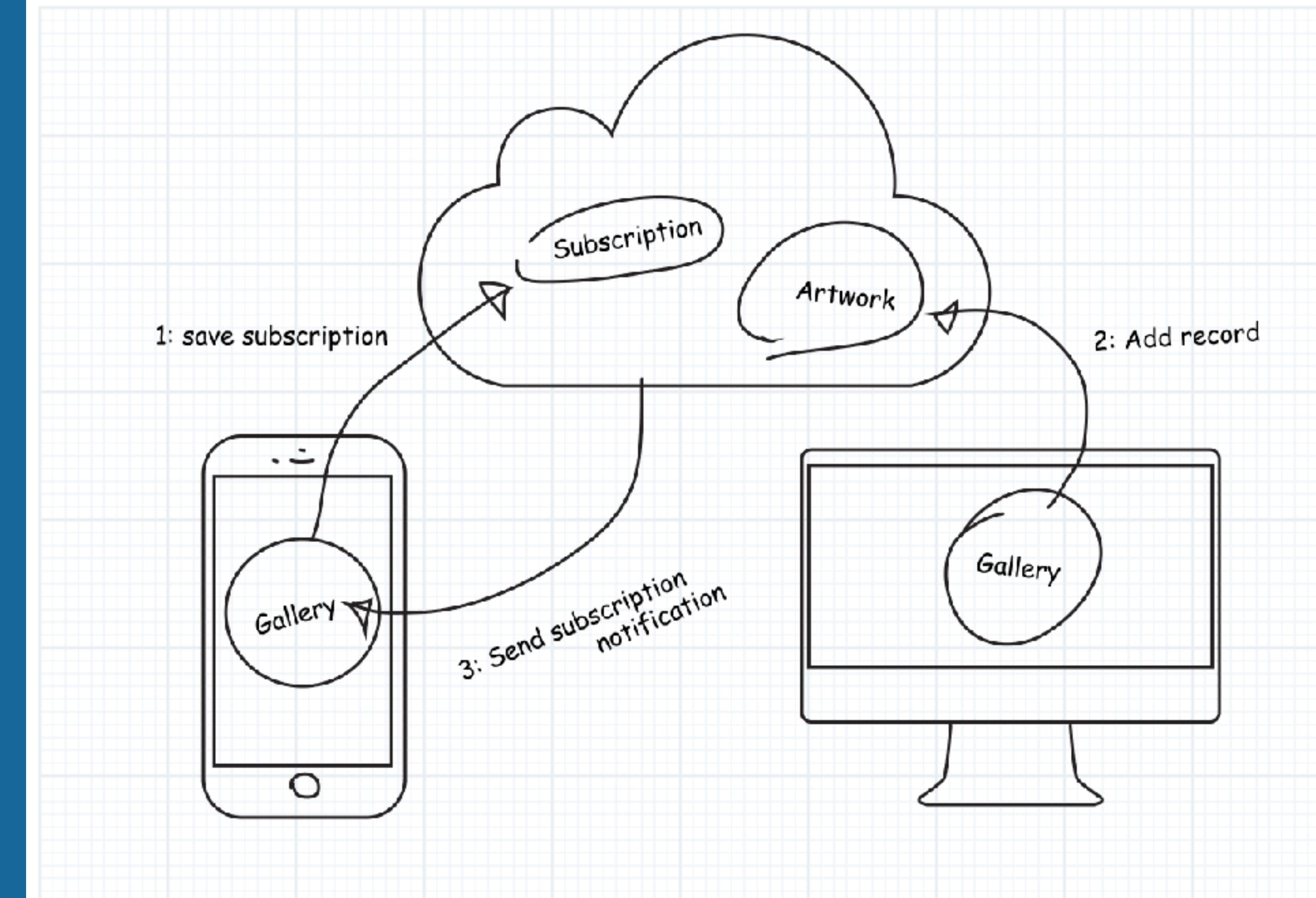
- Queries are polls
- Great for slicing through large server data
- Bad for large, mostly static data set
  - Battery life
  - Networking traffic
  - User experience



# SUBSCRIPTIONS

SUBTITLE

- The server runs your query
  - In the background
  - After every record save
  - Push results to other other devices



# SUBSCRIPTIONS

iCloud.mobi.uchicago.twenty-nineteen-cloudkit ~ > ● Development > Data > Andrew Binkowski | ?

## Subscriptions ▾

Public Database

Fetch Subscriptions

Delete Subscription

ID	type	filterBy	firesOn	firesOnce	shouldSen...	shouldBad...	alertBody
joke-of-the-day-creation	query (joke)	[]	update, delete, create	—	true	true	The Joke of the Day is here! 😂

# SUBSCRIPTIONS

- CKQuerySubscription
  - RecordType
  - NSPredicate
  - Push
- Push delivered via Apple Push Notification Service (APNS) augmented payload
  - JSON

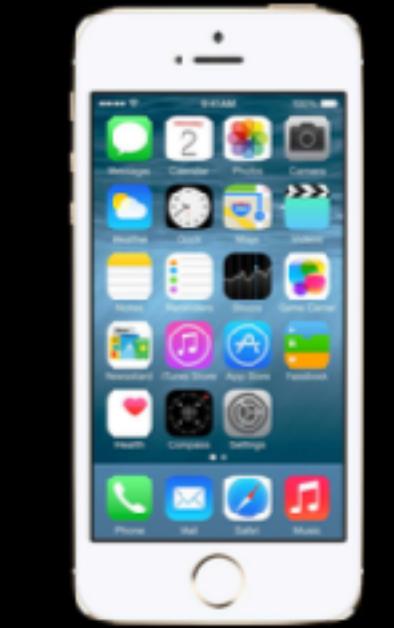
The screenshot shows the 'Subscriptions' section of the iCloud developer portal. A single subscription is listed:

ID	type	filterBy	firesOn	firesOnce
joke-of-the-day-creation	query (joke)	[]	update, delete, create	—

A yellow callout bubble points from the bottom right towards the 'Delete Subscription' button, containing the text: "CHEAPEST, EASIEST WAY TO ADD PUSH NOTIFICATIONS TO AN APP".

# SUBSCRIPTIONS

- Subscription in action



New parties  
In the future  
Alert with "Party Time!"



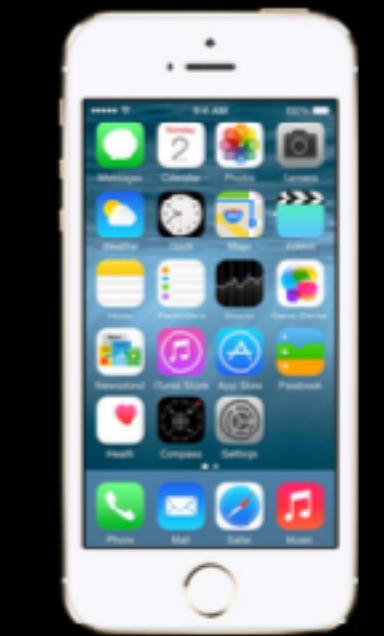
APPLE PARTY NOTIFICATION  
EXAMPLE



# SUBSCRIPTIONS

- Subscription in action

APPLE PARTY NOTIFICATION  
EXAMPLE



New parties  
In the future  
Alert with "Party Time!"



Party  
Tonight  
E31970FB



# SUBSCRIPTIONS

- Subscription in action



New parties  
In the future  
Alert with "Party Time!"



Party  
Tonight  
E31970FB



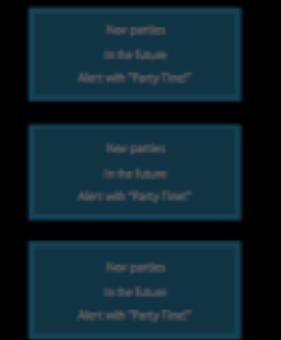
APPLE PARTY NOTIFICATION  
EXAMPLE

# SUBSCRIPTIONS

- Subscription in action



New parties  
In the future  
Alert with "Party Time!"



Party  
Tonight  
E31970FB



APPLE PARTY NOTIFICATION EXAMPLE

# SUBSCRIPTIONS

- Subscription in action



New parties  
In the future  
Alert with "Party Time!"

Party  
Time!  
E31970FB

Party  
Tonight



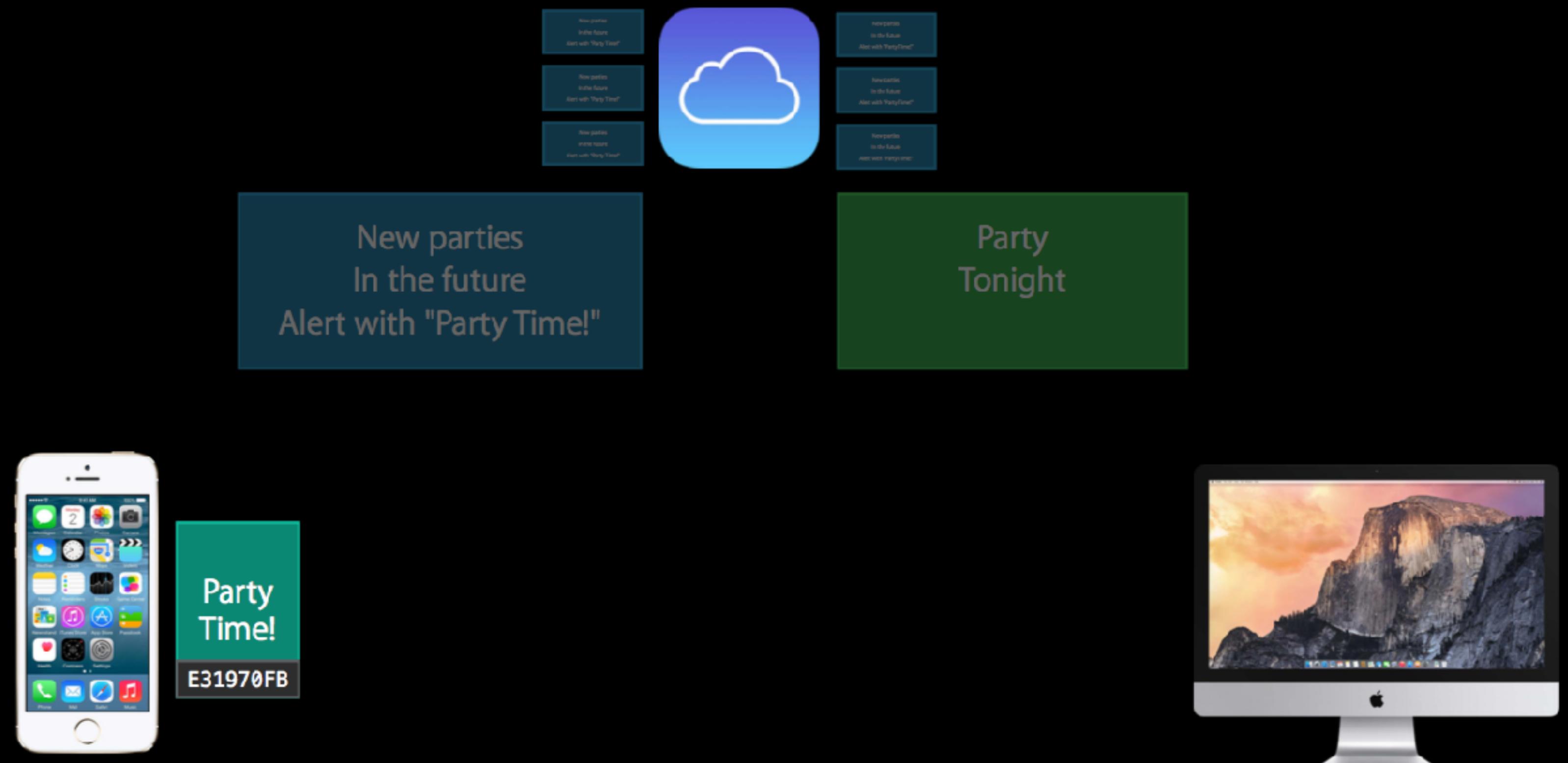
APPLE PARTY NOTIFICATION  
EXAMPLE



# SUBSCRIPTIONS

- Subscription in action

APPLE PARTY NOTIFICATION EXAMPLE



# SUBSCRIPTIONS

- Subscriptions are user based
- Require a unique id

```
func registerJokeOfTheDaySubscriptions() {  
    // Unique identifier for the subscription  
    let identifier = "joke-of-the-day-creation"  
  
    // Create the notification that will be delivered  
    let notificationInfo = CKSubscription.NotificationInfo()  
    notificationInfo.alertBody = "The Joke of the Day is here! 😂"  
    notificationInfo.shouldBadge = true  
    notificationInfo.shouldSendContentAvailable = true  
    //notificationInfo.desiredKeys = ["joke"]  
  
    // Create the subscription object  
    let subscription = CKQuerySubscription(recordType: "joke",  
                                            predicate: NSPredicate(value: true),  
                                            subscriptionID: identifier,  
                                            options: [  
        CKQuerySubscription.Options.firesOnRecordCreate,  
        CKQuerySubscription.Options.firesOnRecordUpdate,  
        CKQuerySubscription.Options.firesOnRecordDelete]  
    )  
  
    subscription.notificationInfo = notificationInfo  
  
    // Save subscription  
    CKContainer.default().publicCloudDatabase.save(subscription, completionHandler: {  
        if let err = error {  
            print("JOTD: subscription failed \(err.localizedDescription)")  
        } else {  
            print("JOTD: subscription set up")  
        }  
    })  
}
```

# SUBSCRIPTIONS

- Create a generic notification
- Some tricks to make your notifications seem more dynamic
  - Alert view when received

```
func registerJokeOfTheDaySubscriptions() {  
    // Unique identifier for the subscription  
    let uuid: UUID = UUID()  
    let identifier = "\((uuid))-joke-of-the-day-any-key"  
  
    // Create the notification that will be delivered  
    let notificationInfo = CKSubscription.NotificationInfo()  
    notificationInfo.alertBody = "The Joke of the Day is here! 😂"  
    notificationInfo.shouldBadge = true  
    notificationInfo.shouldSendContentAvailable = true  
    notificationInfo.desiredKeys = ["question"]  
  
    // Create the subscription object  
    let subscription = CKQuerySubscription(recordType: "joke",  
                                             predicate: NSPredicate(value: true),  
                                             subscriptionID: identifier,  
                                             options: [  
        CKQuerySubscription.Options.firesOnRecordCr  
  
    subscription.notificationInfo = notificationInfo  
  
    // Save subscription  
    currentDB.save(subscription, completionHandler: {returnRecord, error in  
        if let err = error {  
            print("JOTD: subscription failed \(err.localizedDescription)")  
        } else {  
            print("JOTD: subscription set up")  
        }  
    })  
}
```

# SUBSCRIPTIONS

- Create a subscription

```
func registerJokeOfTheDaySubscriptions() {  
    // Unique identifier for the subscription  
    let uuid: UUID = UUID()  
    let identifier = "\u{uuid}-joke-of-the-day-any-key"  
  
    // Create the notification that will be delivered  
    let notificationInfo = CKSubscription.NotificationInfo()  
    notificationInfo.alertBody = "The Joke of the Day is here! 😂"  
    notificationInfo.shouldBadge = true  
    notificationInfo.shouldSendContentAvailable = true  
    notificationInfo.desiredKeys = ["question"]  
  
    // Create the subscription object  
    let subscription = CKQuerySubscription(recordType: "joke",  
                                            predicate: NSPredicate(value: true),  
                                            subscriptionID: identifier,  
                                            options: [  
                                                CKQuerySubscription.Options.firesOnRecordCreate])  
  
    subscription.notificationInfo = notificationInfo  
  
    // Save subscription  
    currentDB.save(subscription, completionHandler: {returnRecord, error in  
        if let err = error {  
            print("JOTD: subscription failed \(err.localizedDescription)")  
        } else {  
            print("JOTD: subscription set up")  
        }  
    })  
}
```

# SUBSCRIPTIONS

- Save a subscription
  - Need to do some local work to remember state of subscription preferences

```
func registerJokeOfTheDaySubscriptions() {  
    // Unique identifier for the subscription  
    let uuid: UUID = UUID()  
    let identifier = "\u{uuid}-joke-of-the-day-any-key"  
  
    // Create the notification that will be delivered  
    let notificationInfo = CKSubscription.NotificationInfo()  
    notificationInfo.alertBody = "The Joke of the Day is here! 😂"  
    notificationInfo.shouldBadge = true  
    notificationInfo.shouldSendContentAvailable = true  
    notificationInfo.desiredKeys = ["question"]  
  
    // Create the subscription object  
    let subscription = CKQuerySubscription(recordType: "joke",  
                                            predicate: NSPredicate(value: true),  
                                            subscriptionID: identifier,  
                                            options: [  
                                                CKQuerySubscription.Options.firesOnRecordCreate])  
  
    subscription.notificationInfo = notificationInfo  
  
    // Save subscription  
    currentDB.save(subscription, completionHandler: {returnRecord, error in  
        if let err = error {  
            print("JOTD: subscription failed \(err.localizedDescription)")  
        } else {  
            print("JOTD: subscription set up")  
        }  
    })  
}
```

# SUBSCRIPTIONS

SHOW ALL SUBSCRIPTIONS

iCloud.mobi.uchicago Development > Data Andrew Binkowski ?

## Subscriptions ▾

Public Database

Fetch Subscriptions

Delete Subscription

ID	type	filterBy	firesOn	firesOnce	shouldS...	shouldB...	alertBody
95231B4... A939- 4317- A23F- EA932C... joke-of- the-day- any-key	query (joke)	[]	create	—	true	true	The Joke of the Day is here! 😂

SEE DETAILS

SETUP YOUR APP TO  
HANDLE NOTIFICATIONS

# SUBSCRIPTIONS

- Required capabilities
  - Push notifications

dyWit...nceOfErrors > Generic iOS Device    Finished running CloudyWithAChanceOfErrors on T. Andrew Binko..

CloudyWithAChanceOfErrors

General Capabilities Resource Tags Info Build Settings Build Phases Build Rules

PROJECT iCloud ON

TARGETS CloudyWithAChanceOf... Push Notifications ON

Steps: ✓ Add the Push Notifications feature to your App ID.  
✓ Add the Push Notifications entitlement to your entitlements file

Game Center OFF

Wallet OFF

Siri OFF

Apple Pay OFF

In-App Purchase OFF

Maps OFF

Personal VPN OFF

Network Extensions OFF

Background Modes ON

Modes:

- Audio, AirPlay, and Picture in Picture
- Location updates
- Voice over IP
- Newsstand downloads
- External accessory communication
- Uses Bluetooth LE accessories
- Acts as a Bluetooth LE accessory
- Background fetch
- Remote notifications

Steps: ✓ Add the Required Background Modes key to your info plist file

# SUBSCRIPTIONS

- Required capabilities
  - Remote notification
  - Background fetch

## ▼ Background Modes

- Modes:
- Audio, AirPlay, and Picture in Picture
  - Location updates
  - Voice over IP
  - Newsstand downloads
  - External accessory communication
  - Uses Bluetooth LE accessories
  - Acts as a Bluetooth LE accessory
  - Background fetch
  - Remote notifications

Steps: ✓ Add the Required Background Modes key to your info plist f

---

## ► Inter-App Audio

---

## ► Keychain Sharing

---

## ► Associated Domains

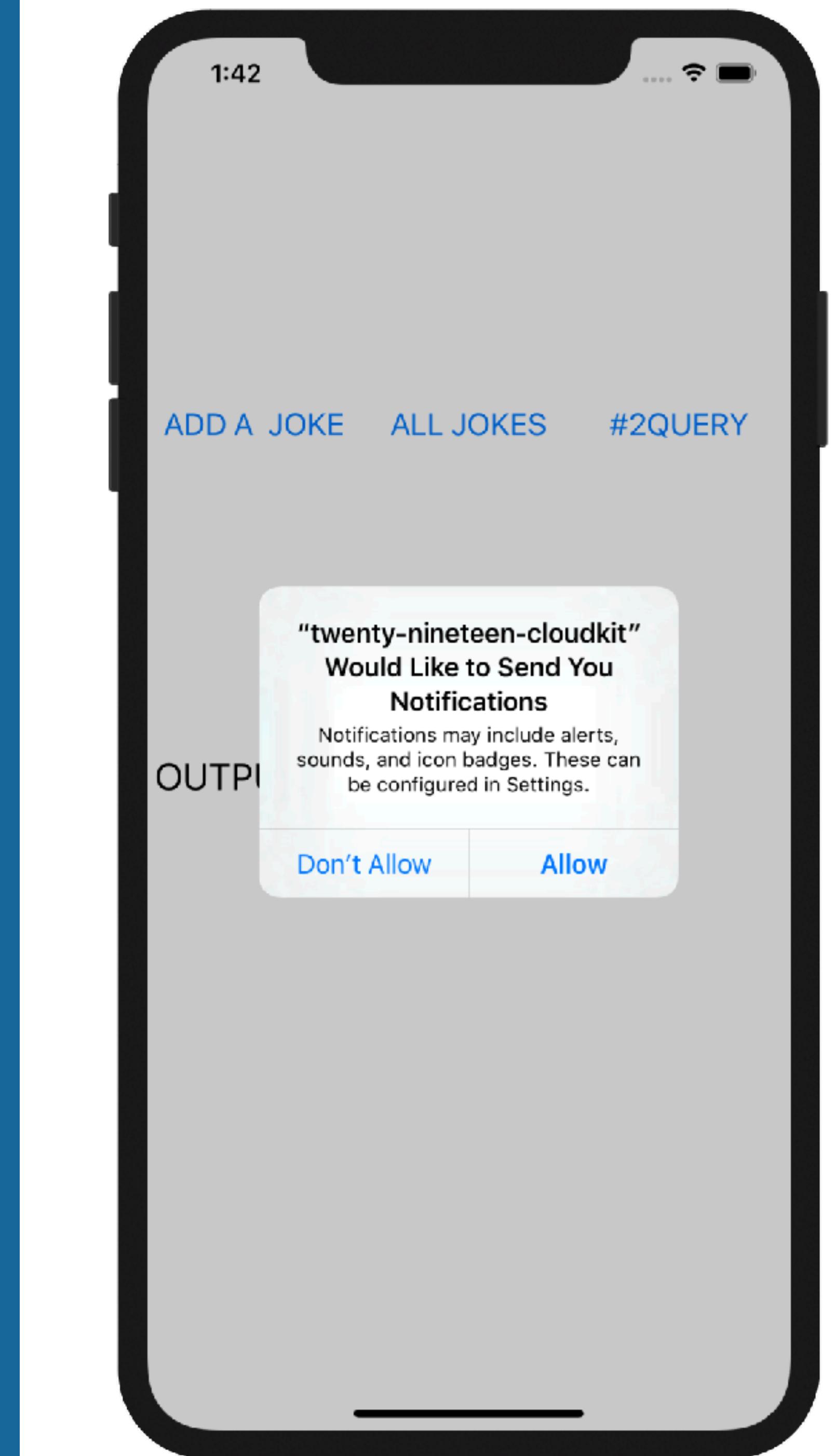
# SUBSCRIPTIONS

REQUEST PERMISSIONS TO RECEIVE  
NOTIFICATIONS

```
func application(_ application: UIApplication,  
                 didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {  
  
    // Set the notification delegate  
    UNUserNotificationCenter.current().requestAuthorization(options: [.alert, .sound, .badge]) { granted, error in  
        if let error = error {  
            print("Error: \(error.localizedDescription)")  
        } else {  
            application.registerForRemoteNotifications()  
        }  
    }  
    UNUserNotificationCenter.current().delegate = self  
  
    // Register subscriptions  
    CloudKitManager.sharedInstance.registerSubscriptionsWithIdentifier("id2")  
    CloudKitManager.sharedInstance.registerSilentSubscriptionsWithIdentifier("id3")  
    //configureUserNotificationsCenter()  
  
    if let notification = launchOptions?[UIApplicationLaunchOptionsKey.remoteNotification] as? [String: AnyObject] {  
        // 2  
        let aps = notification["aps"] as! [String: AnyObject]  
        print("APS: \(aps)")  
        // 3  
        //((window?.rootViewController as? UITabBarController)?.selectedIndex = 1  
    }  
    return true  
}
```

# SUBSCRIPTIONS

- Users will have to grant permissions for notifications that you see
- You can send silent notifications without permission



# SUBSCRIPTIONS

```
func application(_ application: UIApplication,  
                 didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {  
  
    // Set the notification delegate  
    UNUserNotificationCenter.current().requestAuthorization(options: [.ale  
        if let error = error {  
            print("Error: \(error.localizedDescription)")  
        } else {  
            application.registerForRemoteNotifications()  
        }  
    }  
    UNUserNotificationCenter.current().delegate = self  
  
    // Register subscriptions  
    CloudKitManager.sharedInstance.registerSubscriptionsWithIdentifier("id2")  
    CloudKitManager.sharedInstance.registerSilentSubscriptionsWithIdentifier("id3")  
    //configureUserNotificationsCenter()  
  
    if let notification = launchOptions?[UIApplicationLaunchOptionsKey.remoteNotification] as? [String: AnyObject] {  
        // 2  
        let aps = notification["aps"] as! [String: AnyObject]  
        print("APS: \(aps)")  
        // 3  
        //((window?.rootViewController as? UITabBarController)?.selectedIndex = 1  
    }  
    return true  
}
```

REGISTER SUBSCRIPTIONS

error in

# SUBSCRIPTIONS

- Register a subscription

```
func registerSubscriptionsWithIdentifier(_ identifier: String) {  
  
    let uuid: UUID = UIDevice().identifierForVendor!  
    let identifier = "\(uuid)-creation"  
  
    // Create the notification that will be delivered  
    let notificationInfo = CKNotificationInfo()  
    notificationInfo.alertBody = "A new joke was added."  
    notificationInfo.shouldBadge = true  
    notificationInfo.shouldSendContentAvailable = true  
  
  
    let subscription = CKQuerySubscription(recordType: "joke",  
                                            predicate: NSPredicate(value: true),  
                                            subscriptionID: identifier,  
                                            options: [.firesOnRecordCreation])  
    subscription.notificationInfo = notificationInfo  
    CKContainer.default().publicCloudDatabase.save(subscription, completionHandler: {returnRecord, error in  
        if let err = error {  
            print("subscription failed \(err.localizedDescription)")  
        } else {  
            print("subscription set up")  
        }  
    })  
}
```

HANDLE NOTIFICATIONS  
(THERE ARE 3 WAYS)

# SUBSCRIPTIONS

- Handle notifications depending on what you want to do
  - Do nothing (just show notifications)
  - Receive information about what has changed

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {  
    // Set the notification default behavior  
    UNUserNotificationCenter.current().delegate = self  
    if let error = error {  
        print("Error: \(error)")  
    } else {  
        application.registerForRemoteNotifications()  
    }  
    UNUserNotificationCenter.current().requestAuthorization(options: [.alert, .sound]) {  
        // Register subscriptions  
        CloudKitManager.sharedInstance.registerForCloudKitSubscriptions()  
        CloudKitManager.sharedInstance.registerForPushNotifications()  
        //configureUserNotificationSettings()  
  
        if let notification = launchOptions?[.localNotification] as? UILocalNotification {  
            let aps = notification.userInfo![.aps] as! [String: Any]  
            print("APS: \(aps)")  
        }  
        return true  
    }  
}
```

# SUBSCRIPTIONS

- Do nothing
- Handle CloudKit notifications in two places  
(depending on the state of the app)
  - applicationDidFinishLaunching
  - didReceiveRemoteNotification

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {  
    // Set the notification default behavior  
    UNUserNotificationCenter.current().delegate = self  
    if let error = error {  
        print("Error: \(error)")  
    } else {  
        application.registerForRemoteNotifications()  
    }  
    UNUserNotificationCenter.current().requestAuthorization(options: [.alert, .sound]) {  
        granted, error in  
        if granted {  
            print("User granted notifications")  
        } else {  
            print("User denied notifications")  
        }  
    }  
  
    // Register subscriptions  
    CloudKitManager.sharedInstance.registerForCloudKitNotifications()  
    CloudKitManager.sharedInstance.registerForCloudKitPushNotifications()  
    //configureUserNotifications(launchOptions: launchOptions)  
  
    if let notification = launchOptions?[.remoteNotification] as? [String: Any]  
        let aps = notification[.aps] as? [String: Any]  
        print("APS: \(aps)")  
    }  
    return true  
}
```

# SUBSCRIPTIONS

- Handle CloudKit notifications in two places  
(depending on the state of the app)
  - applicationsDidFinishLaunching
  - didReceiveRemoteNotification

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {  
    // If this is a fresh launch, register for notifications.  
    if launchOptions?[.remoteNotification] != nil {  
        let notification = launchOptions![.remoteNotification] as! UNNotification  
        let aps = notification.notification.request.content.userInfo["aps"] as! [String: Any]  
        print("APS: \(aps)")  
    }  
    return true  
}  
  
func application(_ application: UIApplication, didReceiveRemoteNotification notification: UNNotification, fetchCompletionHandler completionHandler: @escaping (UIBackgroundFetchResult) -> Void) {  
    // If this is a fresh launch, register for notifications.  
    if application.applicationState == .background {  
        let notification = notification as! UNNotification  
        let aps = notification.notification.request.content.userInfo["aps"] as! [String: Any]  
        print("APS: \(aps)")  
    }  
    completionHandler(.newData)  
}  
  
// Register subscriptions  
CloudKitManager.sharedInstance.registerForCloudKitNotifications()  
CloudKitManager.sharedInstance.registerForUserNotifications()  
//configureUserNotifications()  
  
if let notification = launchOptions?[.remoteNotification] as? UNNotification {  
    let aps = notification.notification.request.content.userInfo["aps"] as! [String: Any]  
    print("APS: \(aps)")  
}  
return true  
}
```

# SUBSCRIPTIONS

- Handle CloudKit notifications in two places  
(depending on the state of the app)
  - applicationsDidFinishLaunching
  - didReceiveRemoteNotification

ANY OTHER STATE

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {  
    // Set the notification default settings  
    UNUserNotificationCenter.current().delegate = self  
    if let error = error {  
        print("Error: \(error)")  
    } else {  
        application.registerForRemoteNotifications()  
  
        // Register for push notifications  
        UNUserNotificationCenter.current().requestAuthorization(options: [.alert, .sound, .badge]) {  
            granted, error in  
            if granted {  
                print("User granted notifications permission")  
            } else {  
                print("User denied notifications permission")  
            }  
        }  
  
        // Fetch existing subscriptions  
        CloudKitManager.sharedInstance.fetchCloudKitSubscriptions {  
            CloudKitManager.sharedInstance.fetchUserNotifications {  
                //Configure User Notifications  
  
                if let notification = launchOptions?[.remoteNotification] as? UNNotification {  
                    letaps = notification.notificationPayload.userInfo["aps"]  
                    print("APS: \(aps)")  
                }  
                return true  
            }  
        }  
    }  
}
```

# SUBSCRIPTIONS

```
func application(_ application: UIApplication, didReceiveRemoteNotification userInfo: [AnyHashable : Any],  
    fetchCompletionHandler completionHandler: @escaping (UIBackgroundFetchResult) -> Void) {  
    let aps = userInfo["aps"] as! [String: AnyObject]  
  
    if (aps["content-available"] as? NSString)?.integerValue == 1 {  
        // Pull data  
        completionHandler(.newData)  
    } else {  
        completionHandler(.newData)  
    }  
}
```

## Description

Tells the app that a remote notification arrived that indicates there is data to be fetched.

Use this method to process incoming remote notifications for your app. Unlike the `application(_:didReceiveRemoteNotification:)` method, which is called only when your app is running in the foreground, the system calls this method when your app is running in the foreground or background. In addition, if you enabled the remote notifications background mode, the system launches your app (or wakes it from the suspended state) and puts it in the background state when a remote notification arrives. However, the system does not automatically launch your app if the user has force-quit it. In that situation, the user must relaunch your app or restart the device before the system attempts to launch your app automatically again.

# SUBSCRIPTIONS

- Notification can be silent
  - Push only data
- Pay attention to state of application when receiving notifications

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:  
    [UIApplication.LaunchOptionsKey: Any]?) -> Bool {  
  
    // Register a subscription for this device  
    CloudKitManager.sharedInstance.registerJokeOfTheDaySubscriptions()  
  
    // Request authorization for notifications  
    UNUserNotificationCenter.current().requestAuthorization(options: [.alert, .sound, .badge])  
        granted, error in  
        if let error = error {  
            print("D'oh: \(error.localizedDescription)")  
        } else {  
            DispatchQueue.main.async {  
                application.registerForRemoteNotifications()  
            }  
        }  
    }  
  
    // Set delegate to receive notifications in all cases  
    UNUserNotificationCenter.current().delegate = self  
  
    // Parse the launch notification so that we can get the payload  
    if let notification = launchOptions?[UIApplication.LaunchOptionsKey.remoteNotification] as  
        [String: AnyObject] {  
        let aps = notification["aps"] as! [String: AnyObject]  
        print("APS: \(aps)")  
    }  
  
    return true  
}
```

# SUBSCRIPTIONS

- Subscriptions have advanced behavior over notifications
- Delivery behavior
- Can contain record references

```
func registerJokeOfTheDaySubscriptions() {  
    // Unique identifier for the subscription  
    let uuid: UUID = UUID()  
    let identifier = "\u{uuid}-joke-of-the-day-any-key"  
  
    // Create the notification that will be delivered  
    let notificationInfo = CKSubscription.NotificationInfo()  
    notificationInfo.alertBody = "The Joke of the Day is here! 😂"  
    notificationInfo.shouldBadge = true  
    notificationInfo.shouldSendContentAvailable = true  
    notificationInfo.desiredKeys = ["question"]  
  
    // Create the subscription object  
    let subscription = CKQuerySubscription(recordType: "joke",  
                                             predicate: NSPredicate(value: true),  
                                             subscriptionID: identifier,  
                                             options: [  
        CKQuerySubscription.Options.firesOnRecordCre  
  
    subscription.notificationInfo = notificationInfo  
  
    // Save subscription  
    currentDB.save(subscription, completionHandler: {returnRecord, error in  
        if let err = error {  
            print("JOTD: subscription failed \u{err.localizedDescription}")  
        } else {  
            print("JOTD: subscription set up")  
        }  
    })  
}
```

# SUBSCRIPTIONS

```
[AnyHashable("aps"): {
    alert = "The Joke of the Day is here! \Ud83d\Ude02";
    badge = 39;
    "content-available" = 1;
}, AnyHashable("ck"): {
    ce = 2;
    cid = "iCloud.mobi.uchicago.twenty-nineteen-cloudkit";
    ckuserid = "_c364d6b1fb322f17cd2346d1b82667a2";
    nid = "90a87393-1eb3-403c-bff9-73ad00c44820";
    qry = {
        dbs = 2;
        fo = 1;
        rid = "DCE4A6EF-A6A0-47C9-AE60-4022E13D511B";
        sid = "joke-of-the-day-creation";
        zid = "_defaultZone";
        zoid = "_defaultOwner";
    };
}]
```

# SUBSCRIPTIONS

```
[AnyHashable("aps"): {
    alert = "The Joke of the Day is here! \Ud83d\Ude02";
    badge = 39;
    "content-available" = 1;
}, AnyHashable("ck"): {
    ce = 2;
    cid = "iCloud.mobi.uchicago.twenty-nineteen-cloudkit";
    ckuserid = "_c364d6b1fb322f17cd2346d1b82667a2";
    nid = "90a87393-1eb3-403c-bff9-73ad00c44820";
    qry = {
        dbs = 2;
        fo = 1;
        rid = "DCE4A6EF-A6A0-47C9-AE60-4022E13D511B";
        sid = "joke-of-the-day-creation";
        zid = "_defaultZone";
        zoid = "_defaultOwner";
    };
}]]
```

# SILENT NOTIFICATIONS

**DEBUGGING  
NOTIFICATION  
HANDLING**

# SUBSCRIPTIONS

Guides and Sample Code

Developer



Debugging issues with CloudKit subscriptions

Technical Q&A QA1917

## Debugging issues with CloudKit subscriptions

### Q: My CloudKit subscriptions don't trigger notifications after relevant changes are made. How to debug that?

A: Most CloudKit subscription issues are either due to incorrect assumptions about when and where notifications should fire or improperly configured CloudKit containers or subscriptions. This document will introduce you some cases that aren't expected to trigger CloudKit notifications, following with how to verify the state of your iCloud container and how to avoid some common issues related to CloudKit subscriptions.

### Cases that aren't expected to trigger CloudKit notifications

When working with CloudKit subscriptions, be aware that:

- Notifications won't be delivered to your device if the notification settings for your app are off. CloudKit relies on the Apple Push Notification service (APNs) to deliver notifications. If your app is not allowed for push notifications on the device, you won't see them. To turn the settings on, go to `Settings > Notifications`, then navigate to your app's setting screen.
- CloudKit notifications won't be delivered to the device on which the relevant changes are made.
- The Simulators don't support push notifications. To test push notifications you must be running directly on the target platform.
- CloudKit notifications won't be delivered to your app if the notifications' `shouldSendContentAvailable` property is `true` and meanwhile your app is force quit. On iOS, users can force quit an app by double-tapping the home button and swiping it away from the multitasking UI.
- CloudKit generates a notification for every relevant change. However, notifications can be co-delivered, so you can retrieve the unhandled ones with `CKFetchNotificationChangesOperation` and process them from there.

[HTTPS://DEVELOPER.APPLE.COM/LIBRARY/CONTENT/QA/QA1917/\\_INDEX.HTML](https://developer.apple.com/library/content/qa/QA1917/_index.html)

# SUBSCRIPTIONS

```
// Alert that will show over the entire application to debug the notifications
func alert(title: String, message: String) {
    let alert = UIAlertController(title: title, message: message, preferredStyle: .alert)
    alert.addAction(UIAlertAction(title: "OK",
                                 style: .`default`,
                                 handler: { _ in
                                    NSLog("The \"OK\" alert occurred.")
        }))
    let rvc = (window?.rootViewController as? UITabBarController)
    rvc?.selectedViewController?.present(alert, animated: true, completion: nil)
}
```

# SUBSCRIPTIONS

```
// Throw a local notification now to test the launch cycle
func now(message: String, time: Double = 2) {
    let notificationContent = UNMutableNotificationContent()
    notificationContent.title = "👋"
    notificationContent.body = message

    // Add Trigger
    let notificationTrigger = UNTimeIntervalNotificationTrigger(timeInterval: TimeInterval(time),
                                                                repeats: false)

    // Create Notification Request
    let notificationRequest = UNNotificationRequest(identifier: UUID.init().debugDescription,
                                                    content: notificationContent,
                                                    trigger: notificationTrigger)

    // Add Request to User Notification Center
    UNUserNotificationCenter.current().add(notificationRequest) { (error) in
        if let error = error {
            print("Unable to Add Notification Request (\(error), \(error.localizedDescription))")
        }
    }
}
```

# SUBSCRIPTIONS



Cloudy...OfErrors > T. Andrew Binkowski's iPhone 6 Finished running CloudyWithAChanceOfErrors on T. Andrew Binkowski's iPhone 6 6

Buildtime (6) Runtime

CloudyWithAChanceOfErrors (6)

- Auto Layout Localization
- Fixed width constraints may cause clipping. Main.storyboard width = 200
- Fixed width constraints may cause clipping. Main.storyboard width = 200
- Fixed width constraints may cause clipping. Main.storyboard width = 200
- Unsupported Configuration
- Prototype table cells must have reuse identifiers Main.storyboard
- Dependency Analysis Warnings
- The use of Swift 3 @objc inference in Swift 4 mode is deprecated. Please ...
- Unsupported Configuration
- Prototype table cells must have reuse identifiers Main.storyboard

```
style: .`default`,
handler: { _ in
    NSLog("The \"OK\" alert occurred.")
})
let rvc = (window?.rootViewController as? UITabBarController)
rvc?.selectedViewController?.present(alert, animated: true, completion: nil)
}

/// Throw a local notification now to test the launch cycle
func now(message: String, time: Double = 2) {
    let notificationContent = UNMutableNotificationContent()
    notificationContent.title = "🕒"
    notificationContent.body = message

    // Add Trigger
    let notificationTrigger = UNTimeIntervalNotificationTrigger(timeInterval: TimeInterval(time), repeats: false)

    // Create Notification Request
    let notificationRequest = UNNotificationRequest(identifier: UUID.init().debugDescription,
                                                    content: notificationContent,
                                                    trigger: notificationTrigger)

    // Add Request to User Notification Center
    UNUserNotificationCenter.current().add(notificationRequest) { (error) in
        if let error = error {
            print("Unable to Add Notification Request (\(error), \(error.localizedDescription))")
        }
    }
}
```

Identity and Type

- Name AppDelegate.swift
- Type Default - Swift Source
- Location Relative to Group AppDelegate.swift
- Full Path /Users/tabinowski/Google Drive/g-Teaching/uchicago.cloud/mpcs51033-2017-autumn/mpcs51033-2017-autumn-code-samples/cloudkit/CloudyWithAChanceOfErrors/

View Controller - A controller that manages a view.

Storyboard Reference - Provides a placeholder for a view controller in an external storyboard.

Navigation Controller - A controller that manages navigation through a hierarchy of views.

Table View Controller - A controller that manages a table view.

Collection View Controller - A controller that manages a collection view.

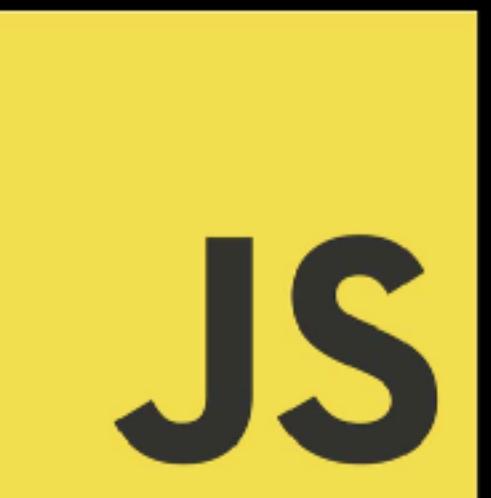
# WEB SERVICES

# WEB SERVICE API

SUBTITLE

- Javascript API
- Matches native CloudKit API
- No intermediate servers
- New notes web app built with CloudKit JS

```
<SCRIPT SRC="HTTPS://CDN.APPLE-CLOUDKIT.COM/CK/1/CLOUDKIT.JS" />
```



# WEB SERVICE API

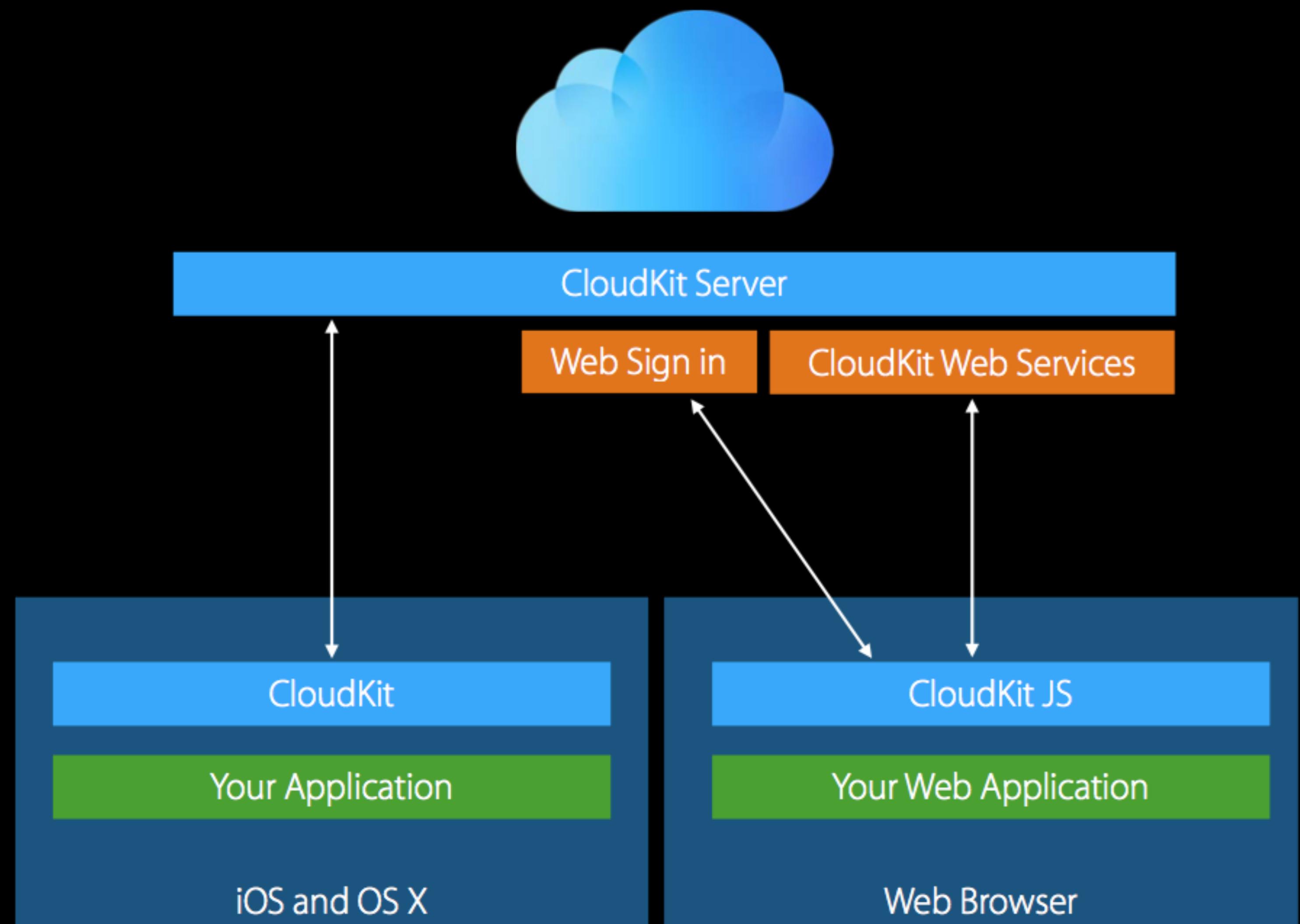
SUBTITLE

- Public and private database access
- Record operations
- Assets
- Query
- Subscriptions and notifications
- User discoverability
- Sync



# WEB SERVICES

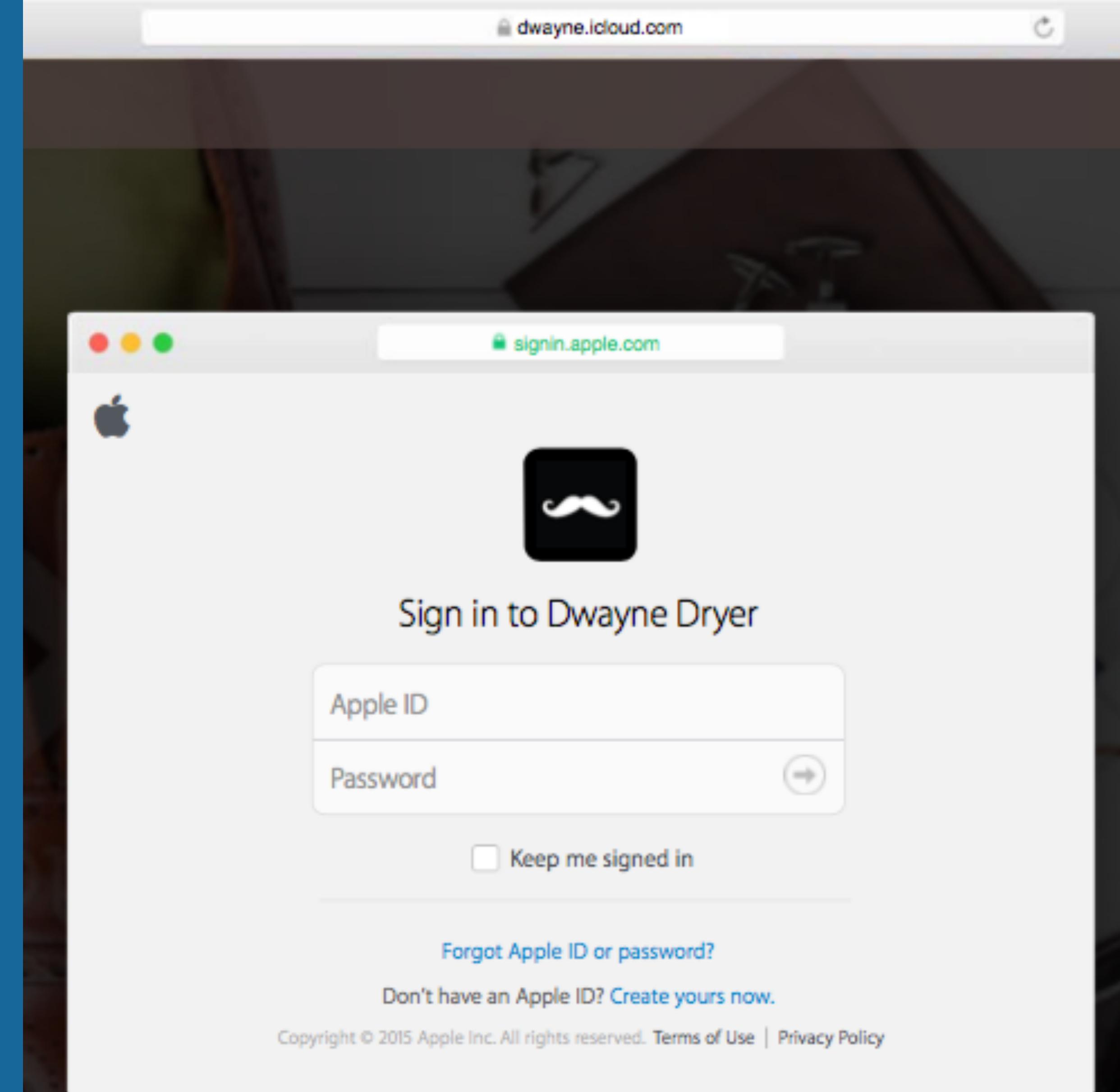
- New server-to-server capabilities
- Add servers to cover the areas that cloud kit doesn't cover



# WEB SERVICE API

SUBTITLE

- Web based authentication
- Need iCloud account
  - New users can sign-up for a free 1GB web only account



# WEB SERVICE API

 CloudKit Catalog

 README

 Authentication

 Discoverability

 Query

 Zones

 Records

 Sync

 Sharing

 Subscriptions

 Notifications  
Disconnected

 Run Code

Container: iCloud.com.example.CloudKitCatalog Environment: production

## CloudKit on the web

This web application provides executable sample code for the core API methods provided by the CloudKit JS JavaScript library. While these methods cover many typical use cases, there are more flexible versions available if needed which allow for batch requests and more configuration. The user is advised to refer to the [CloudKit JS Reference](#) for more information.

All code examples can be run by clicking the play button at the top of the page. The results will be displayed below the sample code block.

### Obtaining the CloudKit JS library

CloudKit JS is hosted at <https://cdn.apple-cloudkit.com/ck/2/cloudkit.js>. Include the library on your web page using either of the two methods below. You will automatically get updates and bug fixes as they are released.

Option #1 - Load CloudKit JS synchronously

```
<script src="https://cdn.apple-cloudkit.com/ck/2/cloudkit.js"></script>
```

Unauthenticated User



# ADVANCED iOS APPLICATION DEVELOPMENT

---

MPCS 51032 • SPRING 2020 • SESSION 3