



THE UNIVERSITY OF  
CHICAGO



MPCS 51033 • AUTUMN 2017 • SESSION 8

---

# BACKENDS FOR MOBILE APPLICATIONS

# CLASS NEWS

## CLASS NEWS

- Office hours Thursday from 10-11:30 in Young 308

## CLASS NEWS

- Week 7 - CloudKit
  - Assignment 5 assigned
- Week 8 - CloudKit Server Extensions; Swift on the Server
  - Case Study assigned
  - Finish server component of Assignment 5
- Week 9 - Case Studies in Class (Assignment 5 Due)
- Week 10 - Cloud Roundup (Realm, Serverless, etc.)

# CLASS NEWS

- Case Studies
  - Find an area of interest in mobile/cloud computing and present to class
  - Ideas:
    - New service (serverless, azure, aws lambda)
    - Technique (sharding :), mysql with app engine)
    - Case study (Snapchat on App Engine, what does XX company use)
    - Deeper dive on topic covered (eg. Google Vision API, etc.)
  - 30 minutes

## CLASS NEWS

- Week 7 - CloudKit
  - Assignment 5 assigned
- Week 8 - CloudKit Server Extensions; Swift on the Server
  - Case Study assigned
  - Finish server component of Assignment 5
- Week 9 - Case Studies in Class (Assignment 5 Due)
- Week 10 - Cloud Roundup (Realm, Serverless, etc.)

MAKE-UP CLASS  
MATERIALS AFTER  
PRESENTATIONS

## CLASS NEWS

- Final Projects
  - Opportunity for you to apply what you learned in class to a project you care about
  - Can use old iOS project or extend assignment
  - Talk me about your ideas early

REVIEW

# NOTIFICATIONS

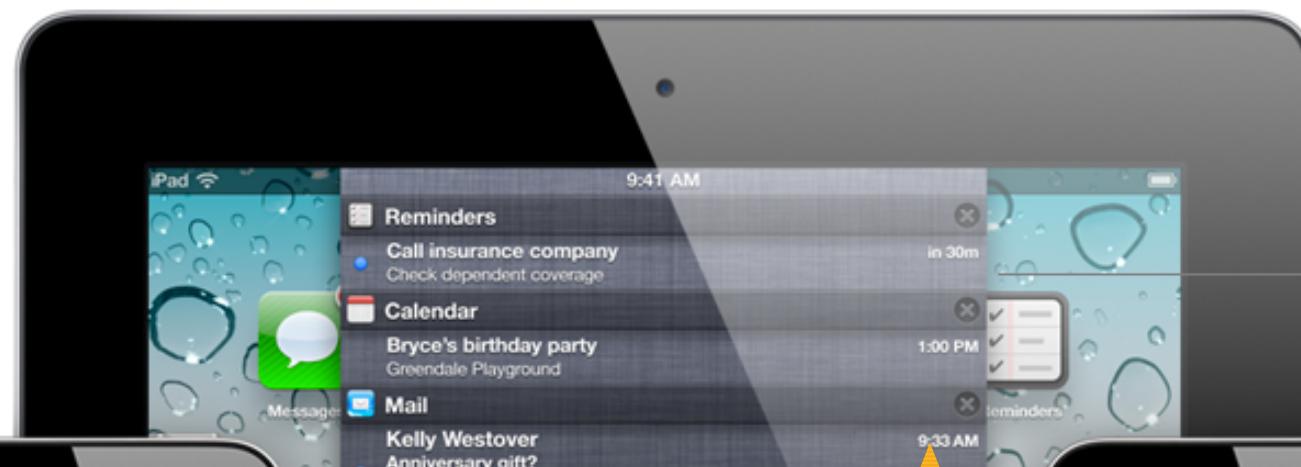
# NOTIFICATIONS



# NOTIFICATIONS

## No more interruptions

Notifications appear at the top of your screen and disappear quickly.



**One-swipe access**  
Swipe down to reveal Notification Center.

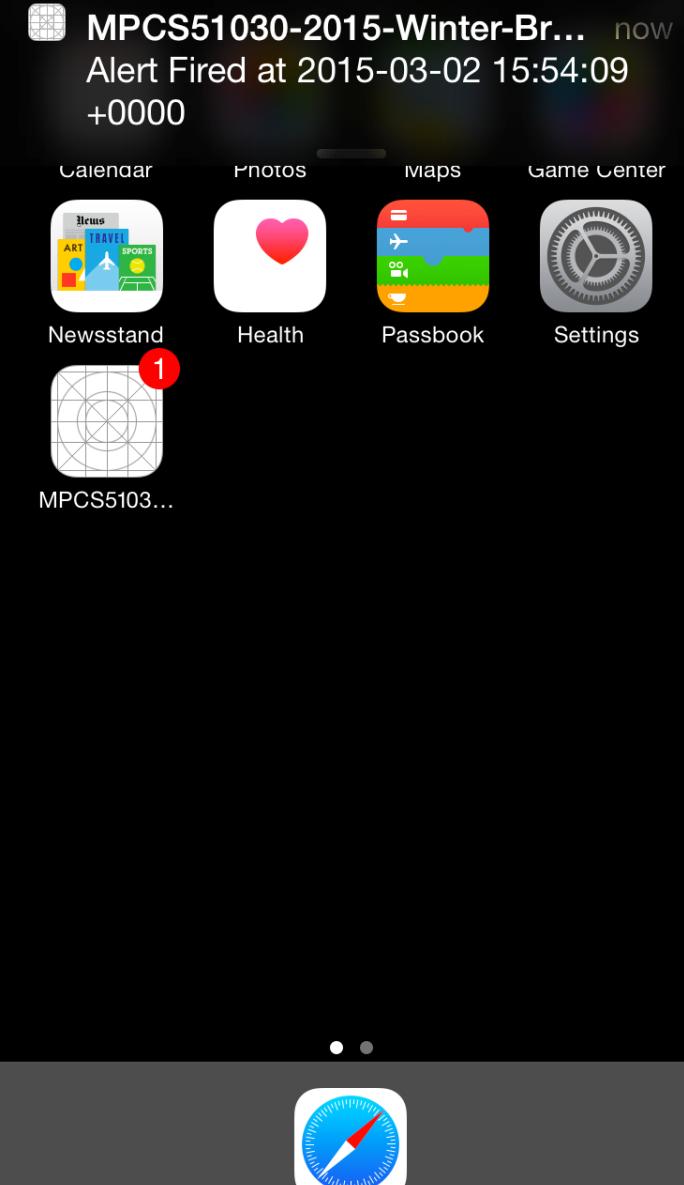
"User Notification"  
not broadcast  
notifications

**See more  
at a glance**  
View stocks and  
weather, too.



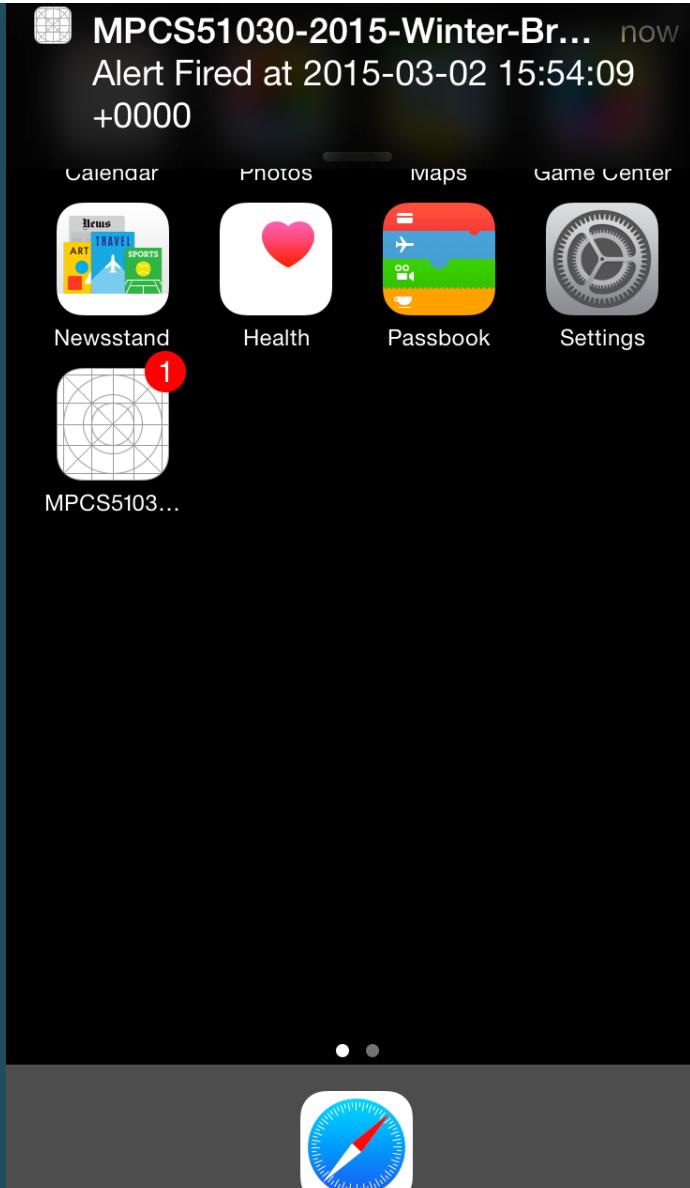
# NOTIFICATIONS

- Only one application can run in the foreground
- Notifications give you flexibility in how to interact with users when they're not running your app
  - Messages
  - Game turn
  - Updates available



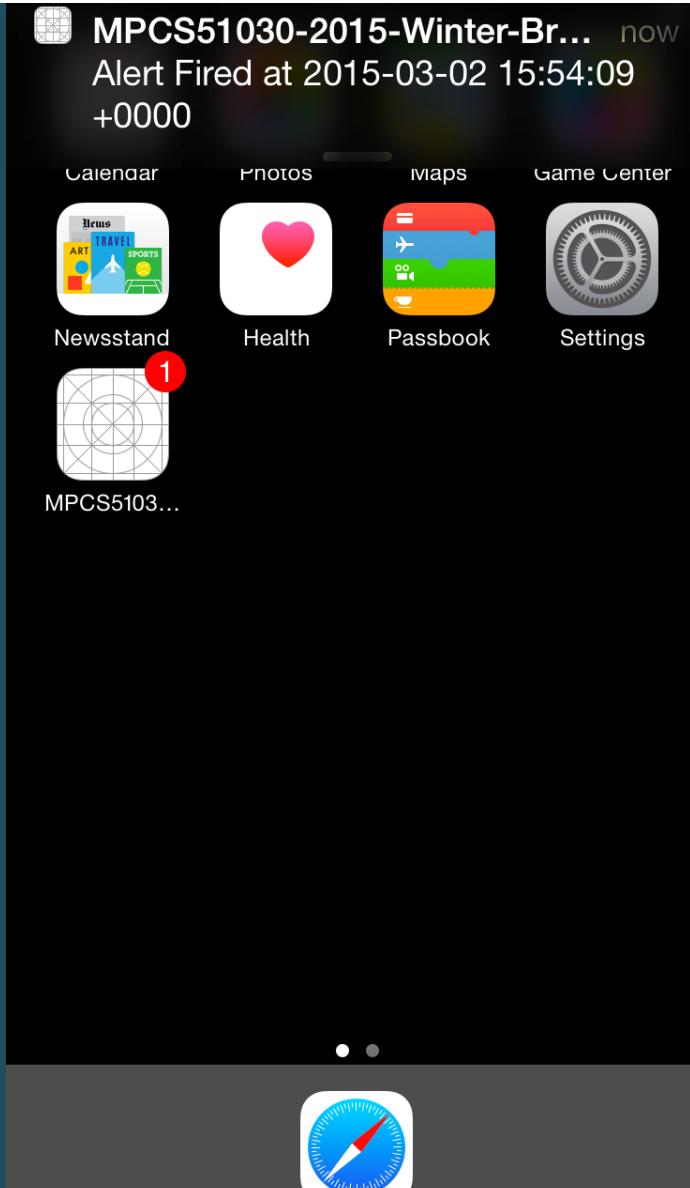
# NOTIFICATIONS

- Re-engage your users
  - Notifications can launch your app



# NOTIFICATIONS

- Notification styles
  - Modal
  - Alert
  - Sounds (default, custom)
  - Badges
  - Silent
    - Happens in background
    - “Trigger event”



# NOTIFICATIONS

- Local Notifications
  - Scheduled by an application
  - Delivered by the iOS on the device
  - "Schedule a notification"

Carrier 9:56 AM

[Back](#) Notifications

Show in Notification Center 5 >

Sounds

Badge App Icon

Show on Lock Screen

Show alerts on the lock screen, and in Notification Center when it is accessed from the lock screen.

ALERT STYLE WHEN UNLOCKED

None Banners Alerts

Alerts require an action before proceeding.  
Banners appear at the top of the screen and go away automatically.

# NOTIFICATIONS

- Push Notifications
  - Remote notifications
  - Sent by application's remote server to Apple Push Notification Service (APNS)
  - "Register a notification"

Carrier 9:56 AM

[Back](#) Notifications

Show in Notification Center 5 >

Sounds

Badge App Icon

Show on Lock Screen

Show alerts on the lock screen, and in Notification Center when it is accessed from the lock screen.

ALERT STYLE WHEN UNLOCKED

None Banners Alerts

Alerts require an action before proceeding.  
Banners appear at the top of the screen and go away automatically.

# NOTIFICATIONS

Sounds

Badge App Icon

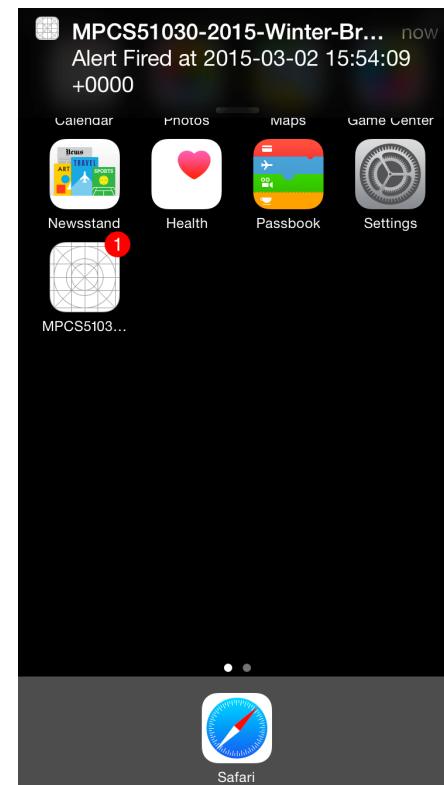
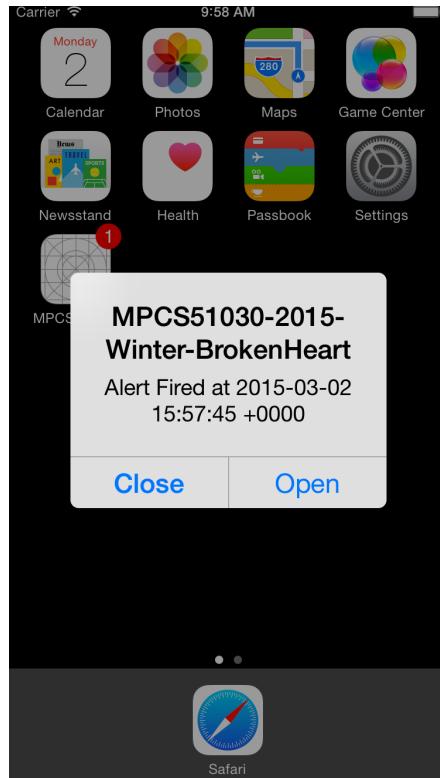
Show on Lock Screen

Show alerts on the lock screen, and in Notification Center when it is accessed from the lock screen.

ALERT STYLE WHEN UNLOCKED

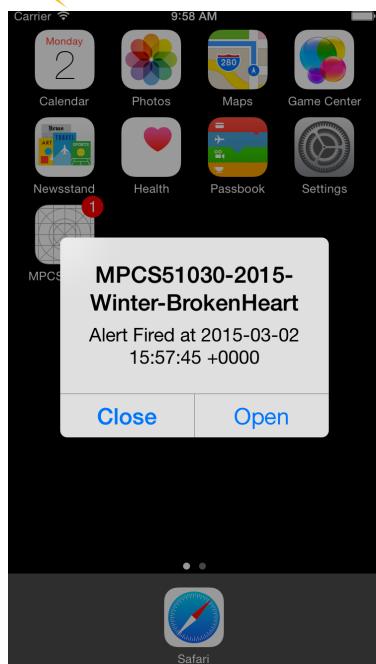
 None     Banners     Alerts

Alerts require an action before proceeding.  
Banners appear at the top of the screen and go away automatically.

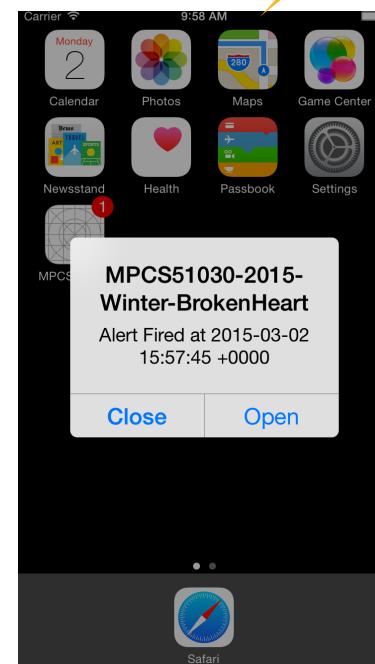


# NOTIFICATIONS

Local



Push



Appears the same on devices

# NOTIFICATIONS

- Handling notifications in app delegate
  - application:didFinishLaunchingWithOptions:
  - application:didReceiveRemoteNotification:
  - application:didReceiveLocalNotification:

Options: local or push

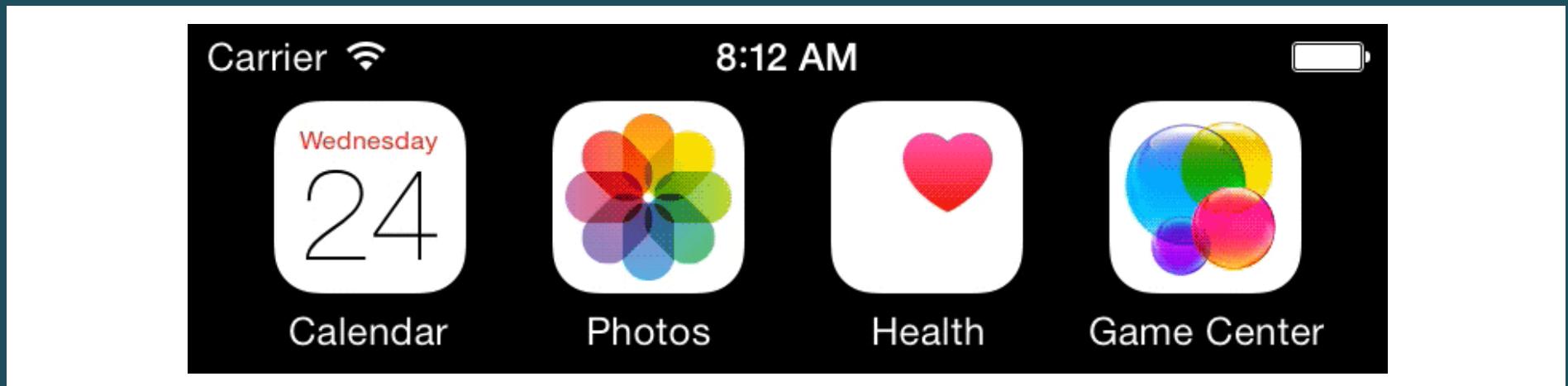
`UIApplicationLaunchOptionsLocalNotificationKey`

# NOTIFICATIONS

- iOS8 introduced interactive notifications
  - Engage with application without opening it



# NOTIFICATIONS



- iOS8 introduced interactive notifications
  - Engage with application without opening it
- iOS9 allows text input (eg. messages) from interactive notifications
- iOS10 unified the API for working with UserNotification Framework

# PERMISSIONS

# PERMISSIONS

- Remember that users are always in control
  - Keep them informed
  - Do not bother them
    - Of course you are trying to get their permission to bombard them with notifications)
  - Make sure your app still functions without notifications
    - Inform the user of consequences
    - "To be informed on the latest info, enable notifications in Settings" don't forget to show a link to settings

Carrier 9:56 AM

[Back](#) Notifications

Show in Notification Center 5 >

Sounds

Badge App Icon

Show on Lock Screen

Show alerts on the lock screen, and in Notification Center when it is accessed from the lock screen.

ALERT STYLE WHEN UNLOCKED

None

Banners

Alerts

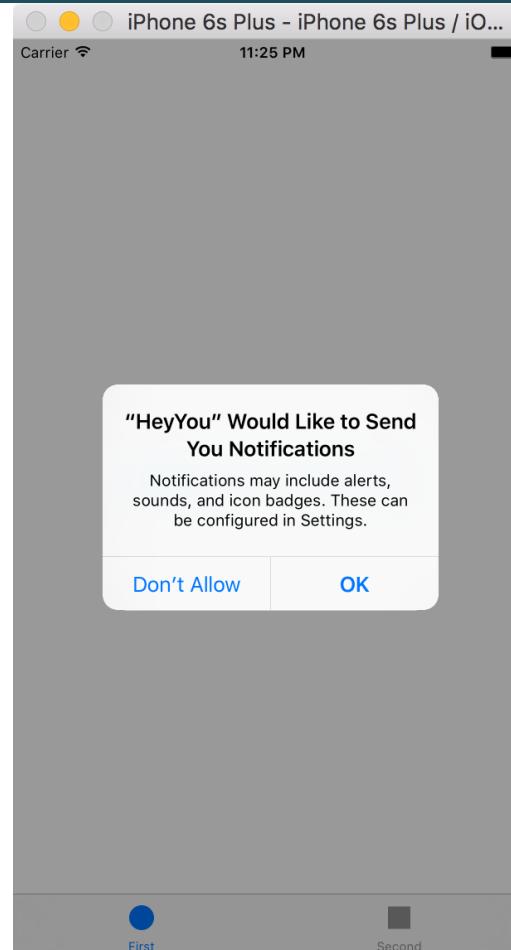
Alerts require an action before proceeding.  
Banners appear at the top of the screen and go away automatically.

# PERMISSIONS

```
let options: UNAuthorizationOptions = [.alert, .sound];
let center = UNUserNotificationCenter.current()
center.requestAuthorization(options: options) {
    (granted, error) in
    if !granted {
        print("Something went wrong")
    }
}
```

- Registering for notifications

# PERMISSIONS



Standard  
Dialogue

# PERMISSIONS

- Consideration on notifications
  - Users may not accept to receive notifications
  - Users can change preferences in settings
- Be prepared to handle all potential use cases

```
center.getNotificationSettings { (settings) in
    if settings.authorizationStatus != .authorized {
        // Notifications not allowed
    }
}
```

# PERMISSIONS

- The old way
  - "Go to Settings > Privacy > Location > OUR\_APP"

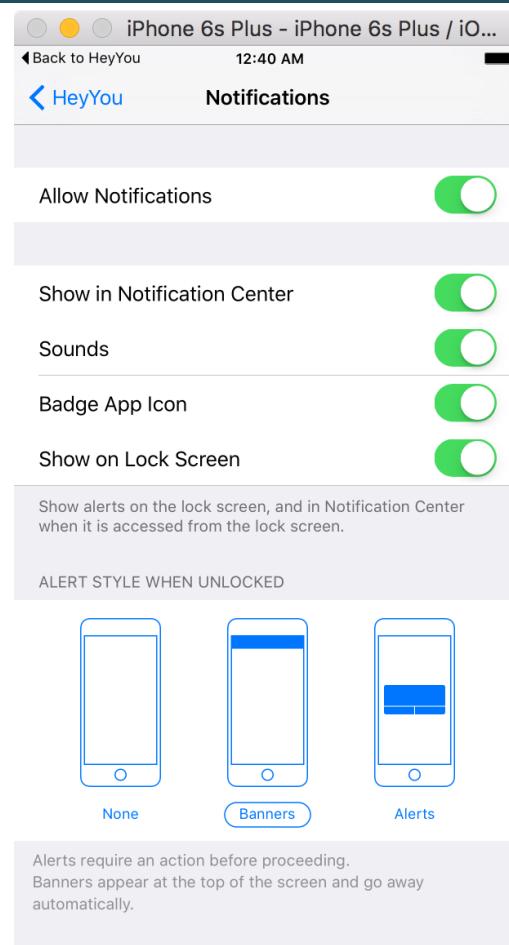
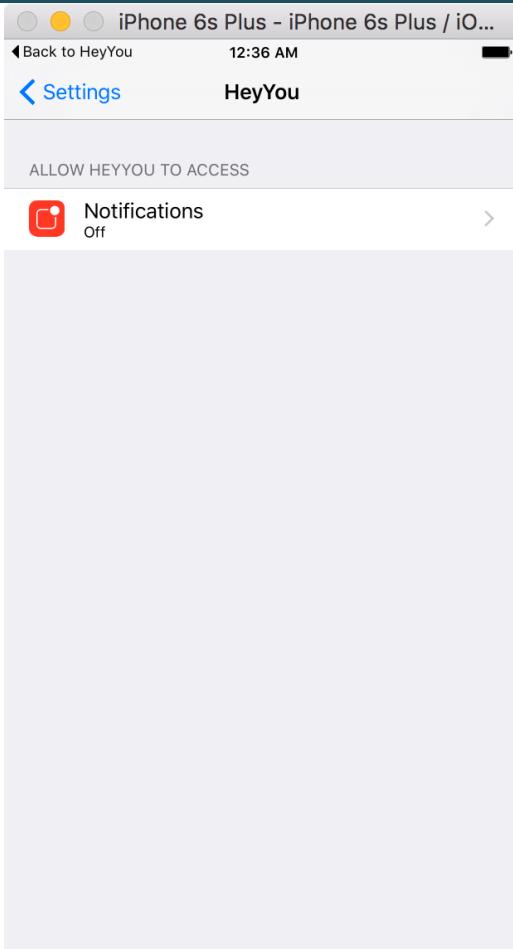


Settings      Swarm

ALLOW SWARM TO ACCESS

	Location	While Using >
	Contacts	<input type="checkbox"/>
	Photos	<input checked="" type="checkbox"/>
	Camera	<input checked="" type="checkbox"/>
	Notifications	>
	Background App Refresh	<input checked="" type="checkbox"/>
	Use Cellular Data	<input checked="" type="checkbox"/>

# PERMISSIONS



# PERMISSIONS

```
// Check if we have authorization to send notifications
guard let settings = UIApplication.sharedApplication().currentUserNotificationSettings() else { return }

// If we don't, let the user know and give them the option of going directly
// to the Settings screen for the application
if settings.types == .None {

    // Create an alert view controller
    let alertController = UIAlertController(title: "⚠️",
                                            message: "The notification permission was not authorized. Please enable it in Settings",
                                            preferredStyle: .Alert)

    // Create to go to settings
    let settingsAction = UIAlertAction(title: "Settings", style: .Default) { (alertAction) in
        if let appSettings = NSURL(string: UIApplicationOpenSettingsURLString) {
            UIApplication.sharedApplication().openURL(appSettings)
        }
    }
    alertController.addAction(settingsAction)

    // Create cancel action that does nothing
    let cancelAction = UIAlertAction(title: "Cancel", style: .Cancel, handler: nil)
    alertController.addAction(cancelAction)

    // Show the alert and exit early
    presentViewController(alertController, animated: true, completion: nil)
    return
}
```

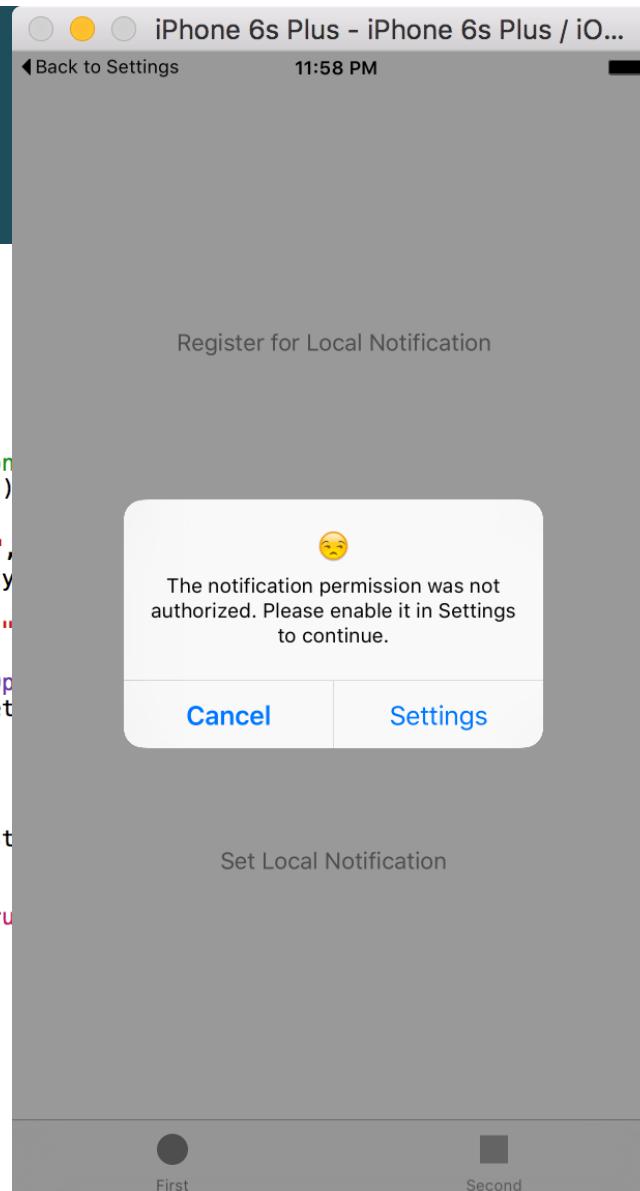
New  
Way

# PERMISSIONS

```
// Check if we have authorization to send notification
guard let settings = UIApplication.sharedApplication()
if settings.types == .None {
    let alertController = UIAlertController(title: ":(", preferredStyle: .Alert)
    let settingsAction = UIAlertAction(title: "Settings", style: .Default) { (action) in
        if let appSettings = NSURL(string: UIApplicationOpenSettingsURLString) {
            UIApplication.sharedApplication().openURL(appSettings)
        }
    }
    alertController.addAction(settingsAction)

    let cancelAction = UIAlertAction(title: "Cancel", style: .Default) { (action) in
        alertController.addAction(cancelAction)

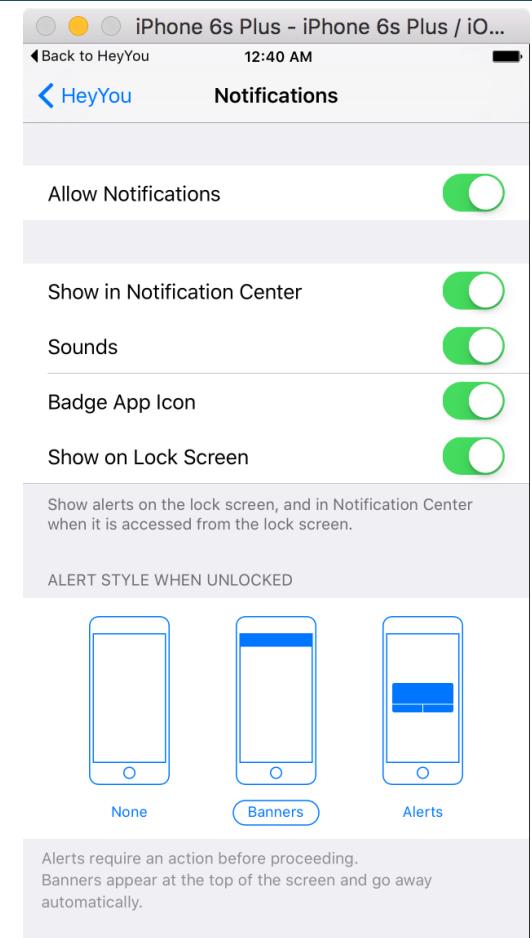
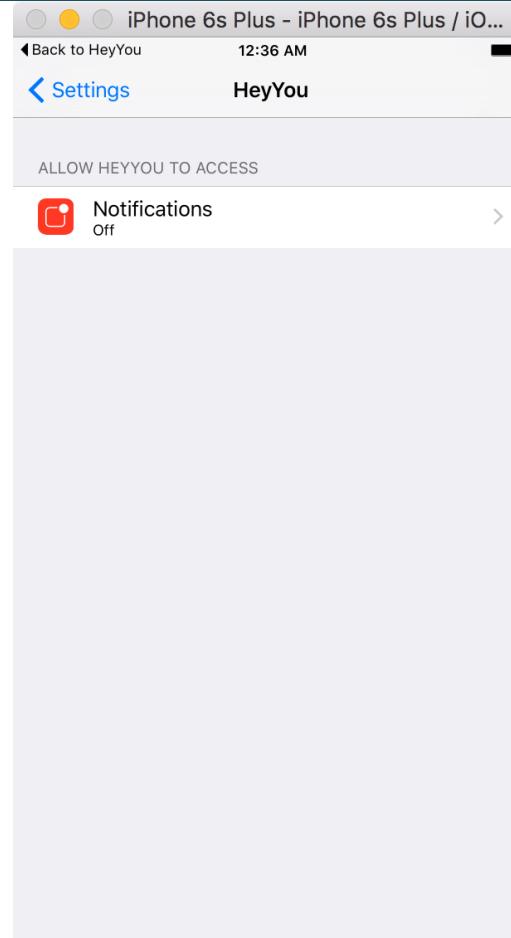
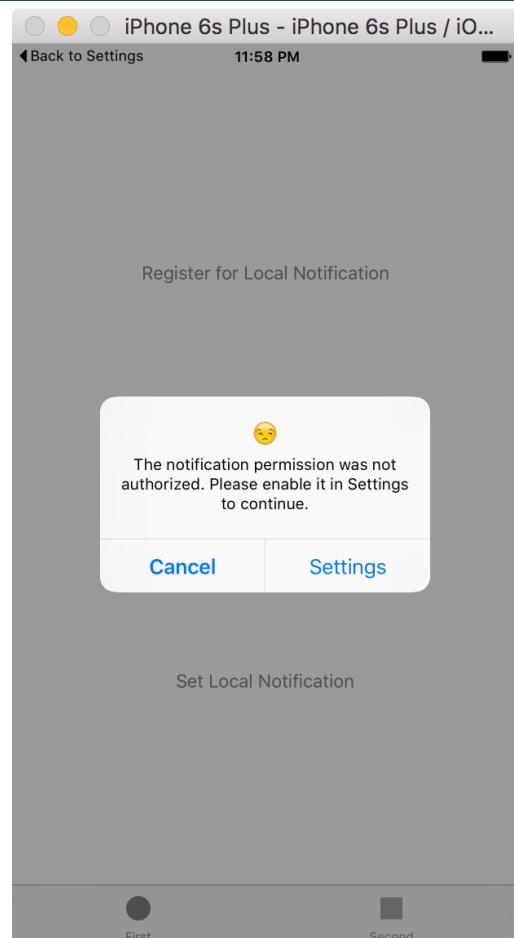
        presentViewController(alertController, animated: true, completion: nil)
        return
    }
}
```



```
}
```

authorized. Please enable it in Settings to continue.",

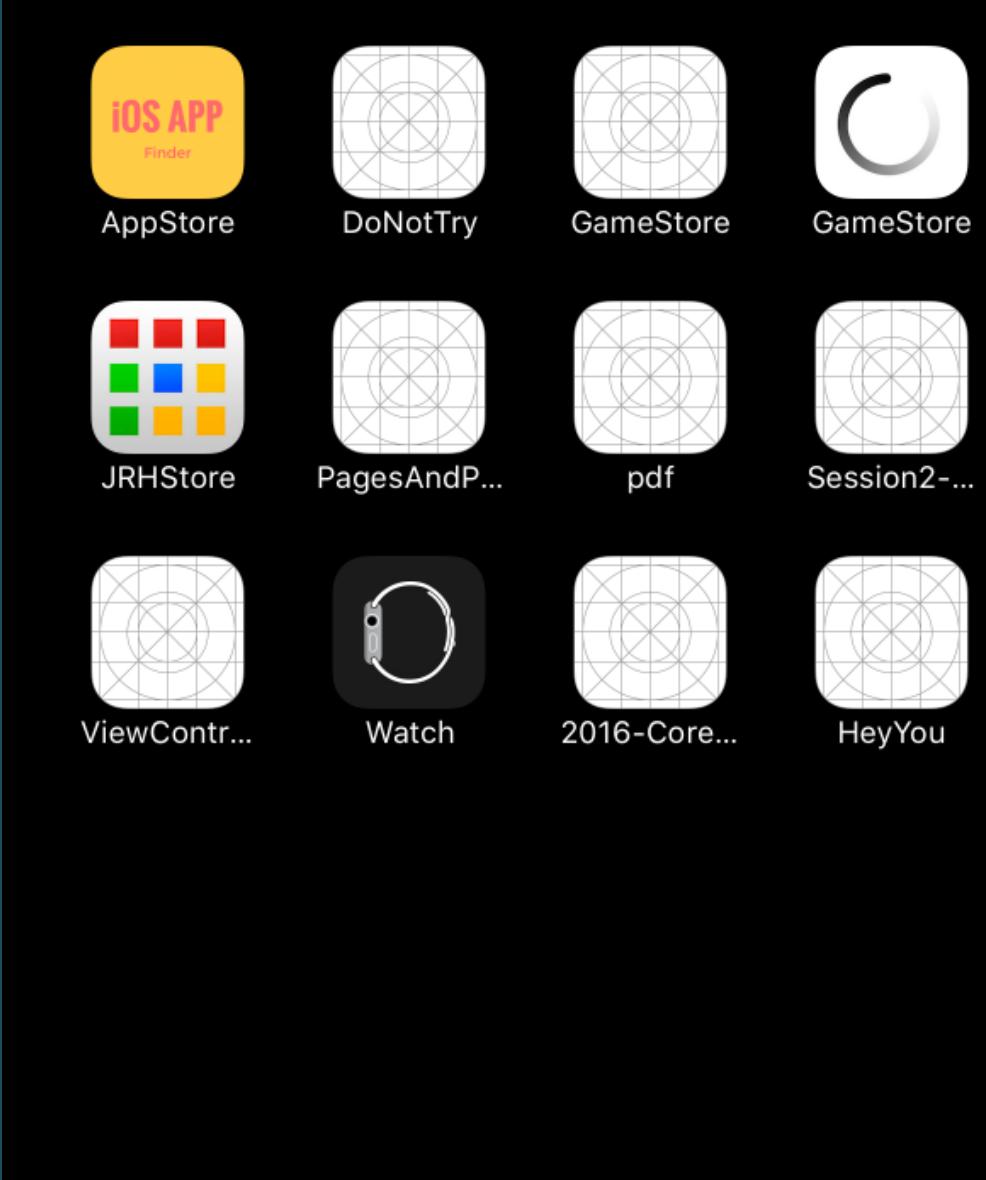
# PERMISSIONS



# LOCAL NOTIFICATIONS

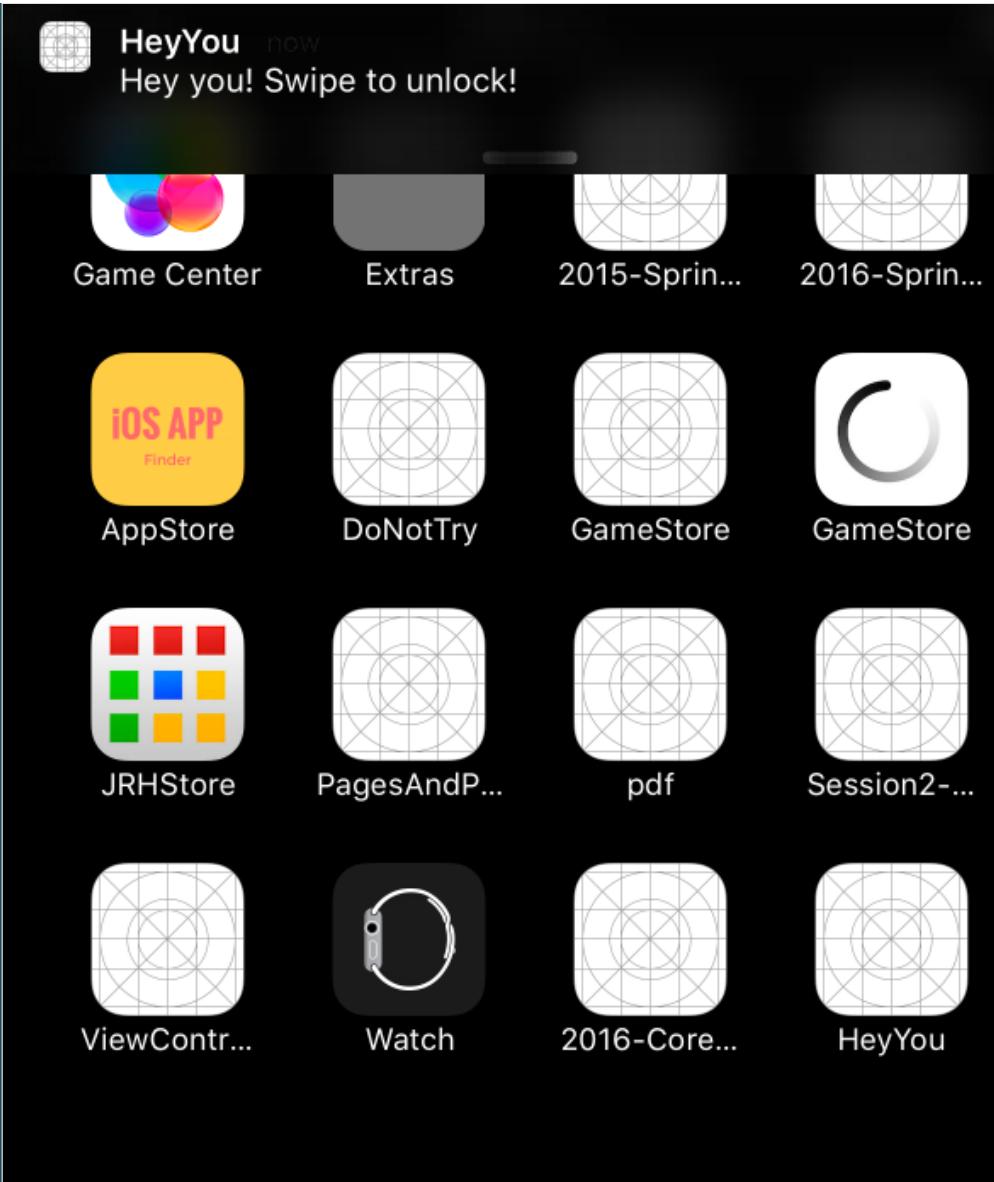
# LOCAL NOTIFICATIONS

- Local notifications are delivered by the iOS device



# LOCAL NOTIFICATIONS

- Messages are handled differently depending on the state of the application
  - Running
  - Background
  - Terminated
- Developer responsibility to handle each case



# LOCAL NOTIFICATIONS

Launch app from notification

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {  
    if let notification: UILocalNotification = launchOptions?[UIApplicationLaunchOptionsLocalNotificationKey] as? UILocalNotification {  
        print("Launch from local notification: \(notification)")  
    }  
    return true  
}
```

- Test for type of notification when launching from terminated state
- Different behavior
  - Go to a view controller
  - Set some date

# LOCAL NOTIFICATIONS

Handle  
notification from  
suspended

```
/// Called when the app is in the background
func userNotificationCenter(_ center: UNUserNotificationCenter,
                           didReceive response: UNNotificationResponse,
                           withCompletionHandler completionHandler: @escaping () -> Void) {
    print(response)
    if response.actionIdentifier == UNNotificationDismissActionIdentifier {
        // The user dismissed the notification without taking action
    }
    else if response.actionIdentifier == UNNotificationDefaultActionIdentifier {
        // The user launched the app
        print("AppDelegate: Did Receive: \(response)")
    }
    completionHandler()
}
```



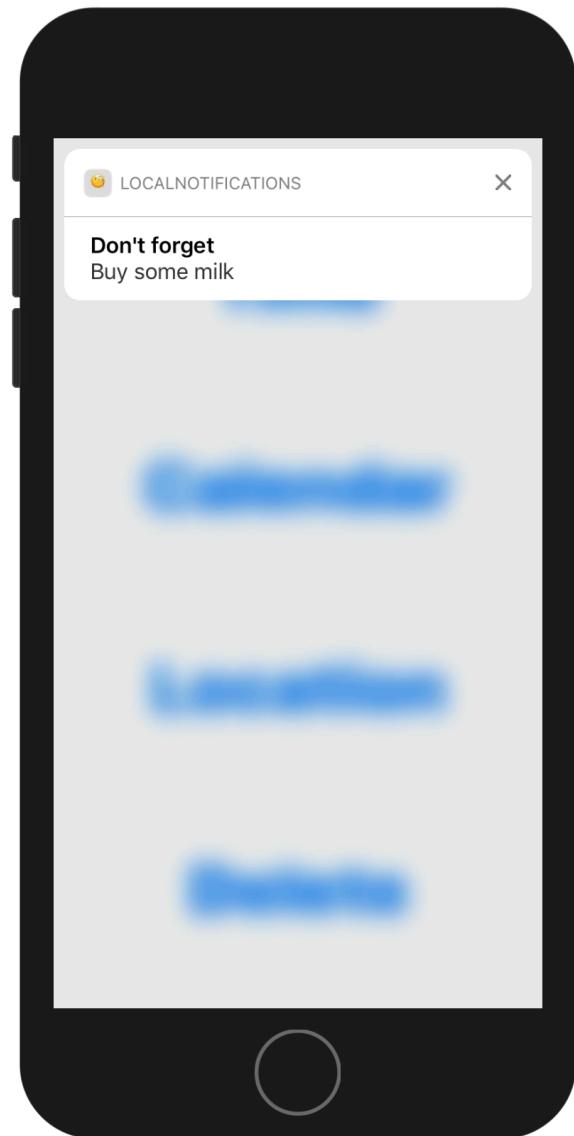
# LOCAL NOTIFICATIONS

Have to  
set  
delegate  
early

```
/// Called when the app is in the background
func userNotificationCenter(_ center: UNUserNotificationCenter,
                           didReceive response: UNNotificationResponse,
                           withCompletionHandler completionHandler: @escaping () -> Void) {
    print(response)
    if response.actionIdentifier == UNNotificationDismissActionIdentifier {
        // The user dismissed the notification without taking action
    }
    else if response.actionIdentifier == UNNotificationDefaultActionIdentifier {
        // The user launched the app
        print("AppDelegate: Did Receive: \(response)")
    }
    completionHandler()
}
```

# LOCAL NOTIFICATIONS

- Remember to handle all the states (if necessary)
  - App in foreground
  - App suspended
  - App terminated



# SCHEDULING LOCAL NOTIFICATIONS

## SCHEDULING NOTIFICATIONS

- Notification request contain content and a trigger

- `title` : String containing the primary reason for the alert.
- `subtitle` : String containing an alert subtitle (if required)
- `body` : String containing the alert message text
- `badge` : Number to show on the app's icon.
- `sound` : A sound to play when the alert is delivered. Use `UNNotificationSound.default()` or create a custom sound from a file.
- `launchImageName` : name of a launch image to use if your app is launched in response to a notification.
- `userInfo` : A dictionary of custom info to pass in the notification
- `attachments` : An array of `UNNotificationAttachment` objects. Use to include audio, image or video content.

Content sent with  
notification

## SCHEDULING NOTIFICATIONS

```
let content = UNMutableNotificationContent()
content.title = "Don't forget"
content.body = "Buy some milk"
content.sound = UNNotificationSound.default()
```

- Notification request contain content and a trigger

# SCHEDULING NOTIFICATIONS

- Triggers
  - Time
  - Calendar
  - Location

## SCHEDULING NOTIFICATIONS

- Schedule a notification for a number of seconds later

```
let content = UNMutableNotificationContent()
content.title = "Don't forget"
content.body = "Buy some milk"
content.sound = UNNotificationSound.default()

_ = UNTimeIntervalNotificationTrigger(timeInterval: 300,
                                      repeats: false)
```

Time trigger

# SCHEDULING NOTIFICATIONS

- Trigger at a specific date and time
  - The trigger is created using a date components object which makes it easier for certain repeating intervals

```
let content = UNMutableNotificationContent()
content.title = "Don't forget"
content.body = "Buy some milk"
content.sound = UNNotificationSound.default()

let date = Date(timeIntervalSinceNow: 3600)
let triggerDate = Calendar.current.dateComponents([.year,.month,.day,.hour,.minute,.second,.millisecond], from: date)

let trigger = UNCalendarNotificationTrigger(dateMatching: triggerDate, repeats: false)
```

## Calendar trigger

# SCHEDULING NOTIFICATIONS

```
let triggerDaily = Calendar.current.dateComponents([.hour,.minute,.second], from: date)
let trigger = UNCalendarNotificationTrigger(dateMatching: triggerDaily, repeats: true)

let triggerWeekly = Calendar.current.dateComponents([.weekday,.hour,.minute,.second], from: date)
let trigger = UNCalendarNotificationTrigger(dateMatching: triggerWeekly, repeats: true)
```

- Repeating notifications

# SCHEDULING NOTIFICATIONS

Location trigger

```
let trigger = UNLocationNotificationTrigger(triggerWithRegion:region, repeats:false)
```

- Trigger when a user enters or leaves a geographic region
- The region is specified through a CoreLocation `CLRegion`
- You need to get permissions just like `CoreLocation`



## SCHEDULING NOTIFICATIONS

```
UNUserNotificationCenter.current().  
    getPendingNotificationRequests {  
        requests in  
        print(requests)  
    }
```

- List all the current notifications

# SCHEDULING NOTIFICATIONS

```
let center = UNUserNotificationCenter.current()
center.removeAllPendingNotificationRequests()
```

- Delete notifications

## SCHEDULING NOTIFICATIONS

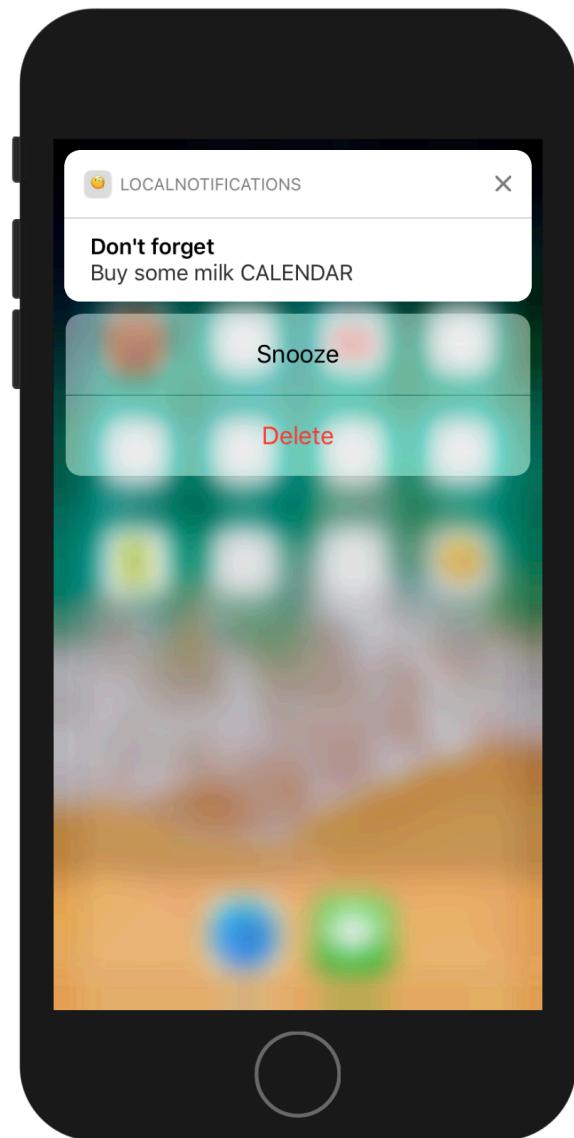
```
@IBAction func tapDelete(_ sender: Any) {  
    listNotification()  
  
    let center = UNUserNotificationCenter.current()  
    center.removeAllPendingNotificationRequests()  
  
    listNotification()  
}
```

- Delete notifications

# NOTIFICATION ACTIONS

# NOTIFICATION ACTIONS

- Define custom actions for your notifications



# NOTIFICATION ACTIONS

```
// Actions
let snoozeAction = UNNotificationAction(identifier: "Snooze", title: "Snooze", options: [])
let deleteAction = UNNotificationAction(identifier: "DeleteAction", title: "Delete", options: [.destructive])
let category = UNNotificationCategory(identifier: "CalendarCategory",
                                      actions: [snoozeAction, deleteAction],
                                      intentIdentifiers: [], options: [])
UNUserNotificationCenter.current().setNotificationCategories([category])
```

# NOTIFICATION ACTIONS

```
let content = UNMutableNotificationContent()
content.title = "Don't forget"
content.body = "Buy some milk CALENDAR"
content.sound = UNNotificationSound.default()
content.categoryIdentifier = "CalendarCategory"

let date = Date(timeIntervalSinceNow: 10)
let triggerDate = Calendar.current.dateComponents([.year,.month,.day,.hour,.minute,.second], from:
    date)
let trigger = UNCalendarNotificationTrigger(dateMatching: triggerDate, repeats: false)

// Actions
let snoozeAction = UNNotificationAction(identifier: "Snooze", title: "Snooze", options: [])
let deleteAction = UNNotificationAction(identifier: "DeleteAction", title: "Delete", options:
    [.destructive])
let category = UNNotificationCategory(identifier: "CalendarCategory",
    actions: [snoozeAction,deleteAction],
    intentIdentifiers: [], options: [])
UNUserNotificationCenter.current().setNotificationCategories([category])
```

# NOTIFICATION ACTIONS

```
// Called when the app is in the background
func userNotificationCenter(_ center: UNUserNotificationCenter,
                           didReceive response: UNNotificationResponse,
                           withCompletionHandler completionHandler: @escaping () -> Void) {

    print("AppDelegate: Did Receive: \(response)")

    // Determine the user action
    switch response.actionIdentifier {
        case UNNotificationDismissActionIdentifier:
            print("Dismiss Action")
        case UNNotificationDefaultActionIdentifier:
            print("Default")
        case "Snooze":
            print("Snooze")
        case "DeleteAction":
            print("Delete")
        default:
            print("Unknown action")
    }
    completionHandler()
}
```

Switch on the  
actionIdentifier

# PUSH NOTIFICATIONS

# PUSH NOTIFICATIONS

- Sent from Apple's servers to device
- Apple Push Notification Service (APNS) is the gateway for push notification
  - SSL certificate
  - Provisioning
  - Networking code
  - Device tokens

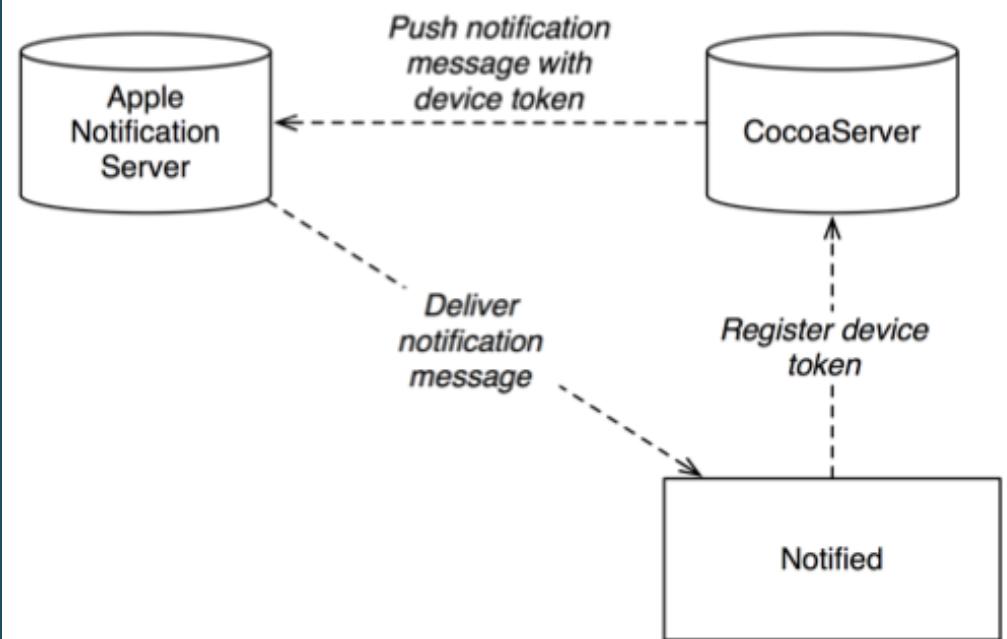
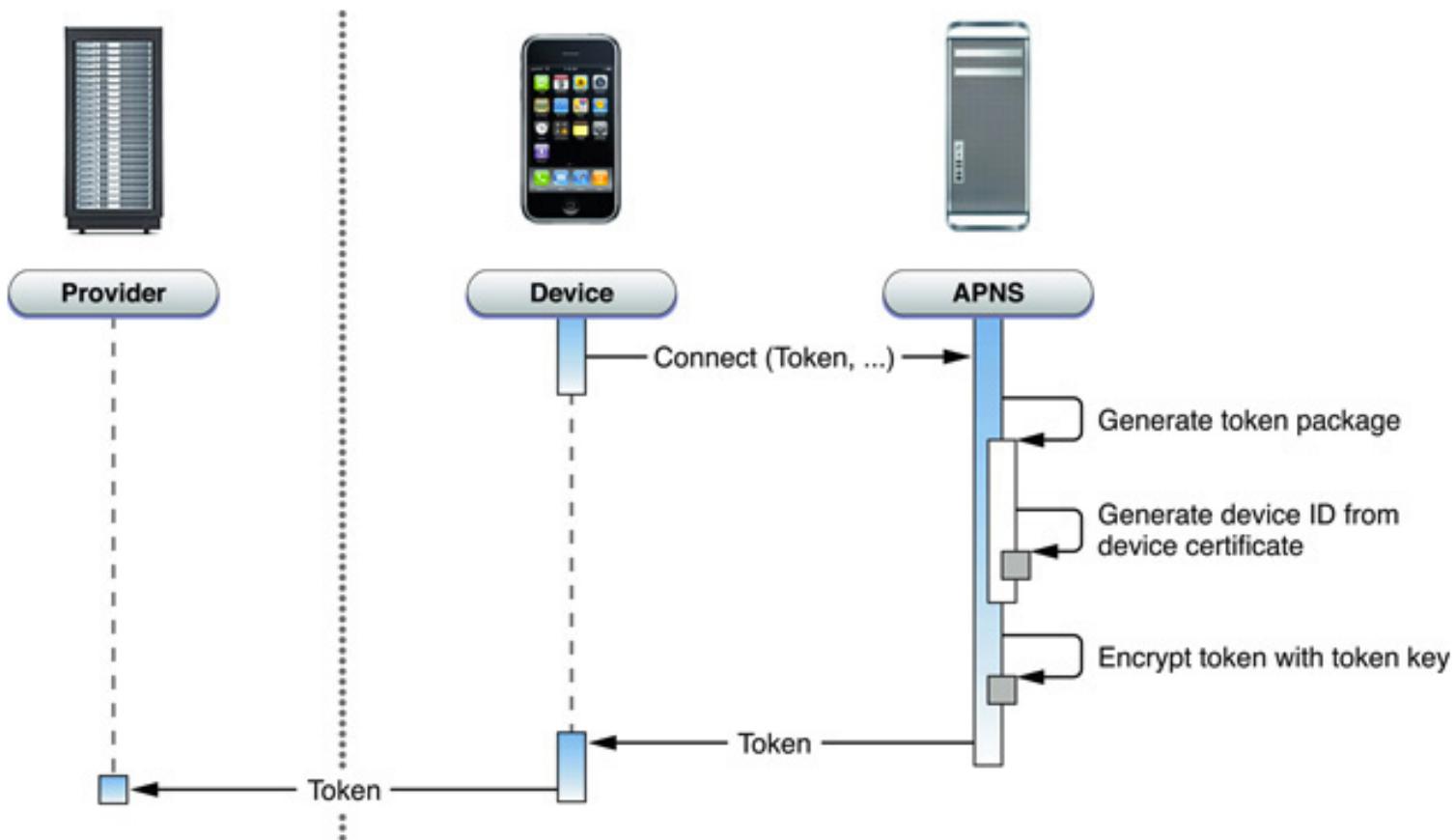


image via Big Nerd Ranch iOS Programming

# PUSH NOTIFICATIONS

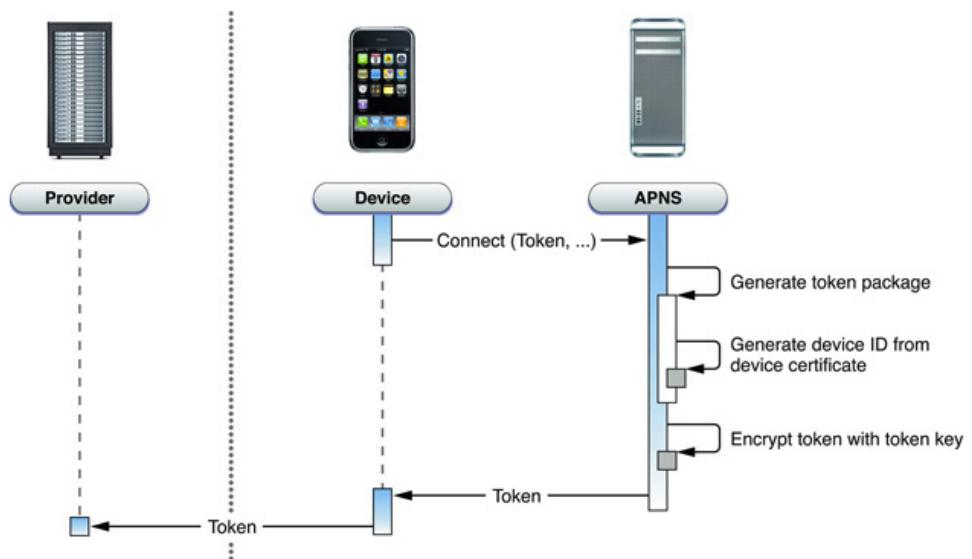
1. An app enables push notification (the user has to confirm that he wishes to receive these notifications)
2. The app receives a “device token”
3. The app sends the device token to your server
4. The server sends a push notification to the Apple Push Notification Service
5. APNS sends the push notification to the user’s device

# PUSH NOTIFICATIONS



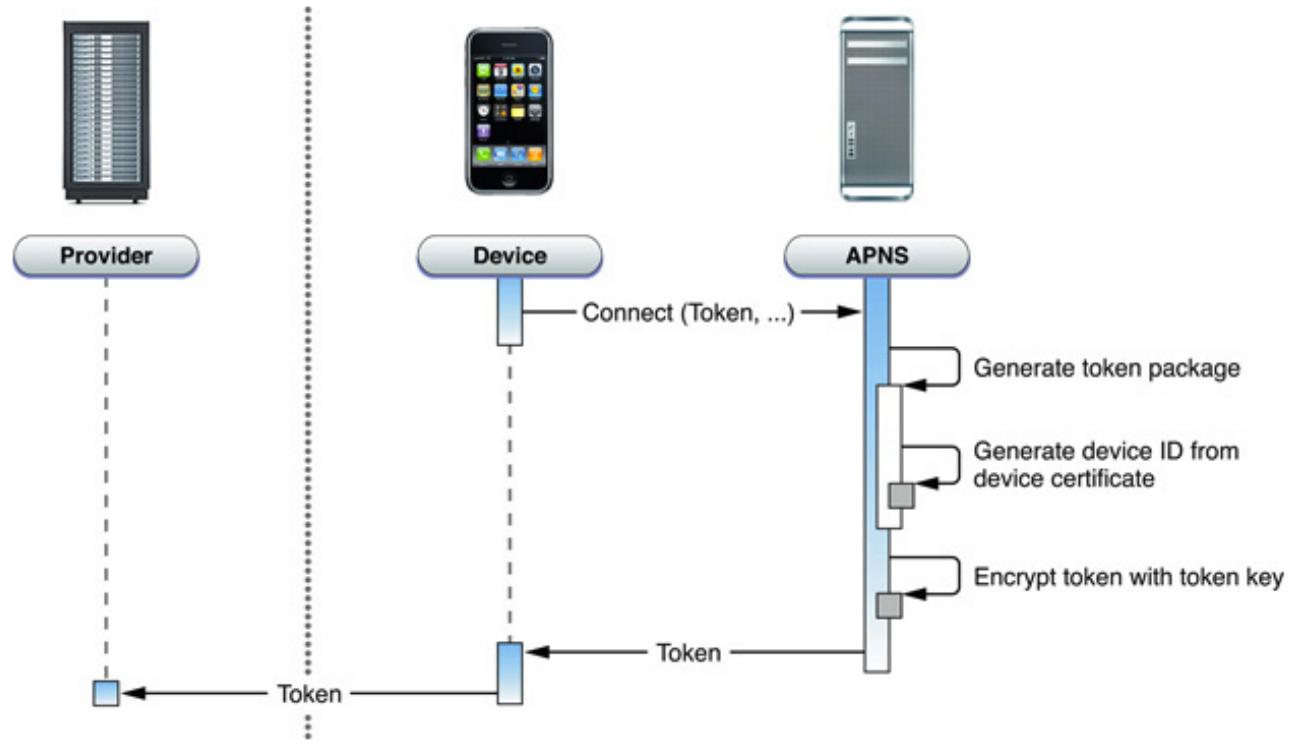
# PUSH NOTIFICATION

- Requirements for push notifications
  - iOS device
  - Paid iOS Developer membership
  - Server
    - Ability to run background processes
    - Install SSL certificates
    - Make outgoing TLS connections on non-standard ports



# PUSH NOTIFICATION

- Many third party solutions for push notifications
  - Parse
  - Firebase
  - App Engine
  - ...



# PUSH NOTIFICATIONS

```
{"aps": {  
    "badge": 100,  
    "alert": "Push Notification Test",  
    "sound": "SoundFile.aif"},  
    "type": "websiteUrl",  
    "data": "http://mySite.com"  
}
```

Manage the badges in your code (not incremental)

Include sound files in your bundle

- Push notification payload
  - JSON dictionary
  - <256 bytes

Pass any key-values in the notification

# PUSH NOTIFICATION

- Receiving notification require app delegate methods to be implemented
- Code path is the same for local notification

```
/// Called when the app is in the background
func userNotificationCenter(_ center: UNUserNotificationCenter,
                           didReceive response: UNNotificationResponse,
                           withCompletionHandler completionHandler: @escaping () -> Void) {
    print(response)
    if response.actionIdentifier == UNNotificationDismissActionIdentifier {
        // The user dismissed the notification without taking action
    }
    else if response.actionIdentifier == UNNotificationDefaultActionIdentifier {
        // The user launched the app
        print("AppDelegate: Did Receive: \(response)")
    }
    completionHandler()
}
```

# PUSH NOTIFICATION

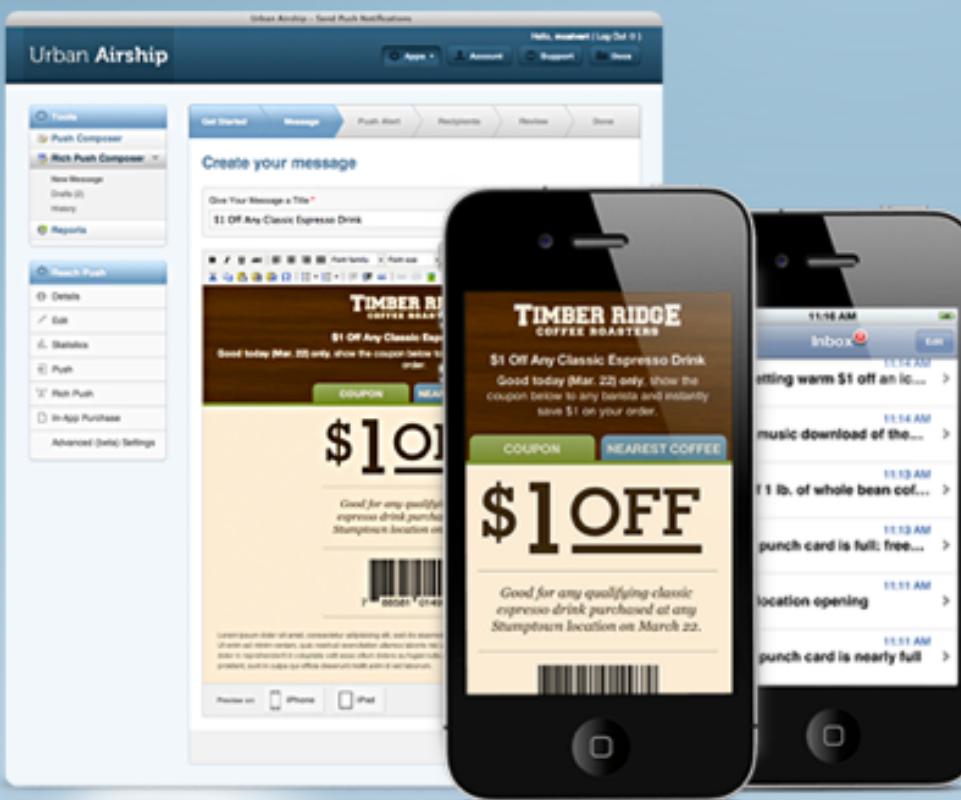
- There are many third party services for setting up push notifications

Urban Airship  
..... powering modern mobile

Products

Customers

Resou



## Rich Push

It just got easier to experiences. Use WYSIWYG editor to surveys and more

Explore Rich Pu

# PUSH NOTIFICATION

- There are many third party services for setting up push notifications

nomad / houston

Code Issues 27 Pull requests 5 Projects 0 Wiki Insights

Apple Push Notifications; No Dirigible Required <http://nomad-cli.com>

houston notifications ruby cli nomad apns

157 commits 1 branch 19 releases 32 contributors MIT

Branch: master New pull request Create new file Upload files Find file Clone or download

mpvosseller authored and dankimio committed on Jan 24 Set default passphrase to nil instead of the empty string. (#158) Latest commit 592bbeb on Jan 24

File	Description	Time Ago
bin	Set default passphrase to nil instead of the empty string. (#158)	a month ago
lib	Set default passphrase to nil instead of the empty string. (#158)	a month ago
spec	Mutable content: update README, add spec (#137)	a year ago
.gitignore	Update .gitignore	a year ago
Gemfile	Update code style (#134)	a year ago
LICENSE	Updating copyright	3 years ago
README.md	README: use unregistered_devices instead of devices (#156)	5 months ago
Rakefile	Rakefile: add Bundler tasks, add default task	a year ago
houston.gemspec	Update dependencies	a year ago

README.md



HOUSTON

# NOTIFICATIONS WRAP-UP

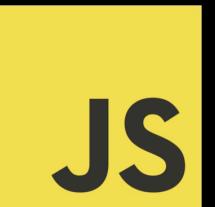
- Notifications keep users engage by opening your app
- Notifications are not guaranteed to be delivered by Apple
- Tips and tricks:
  - Push was important before multi-tasking in iOS4, but many uses can be implemented as local notifications
  - Third-party option for remote notifications
  - Use CloudKit for remote notifications via subscriptions
- Don't abuse notifications...this is one thing that Apple seems to regularly enforce in the App Store Guidelines
  - "Push notifications can not be used for advertising"

# CLOUDKIT API

## WEB SERVICE API

- 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" />
```



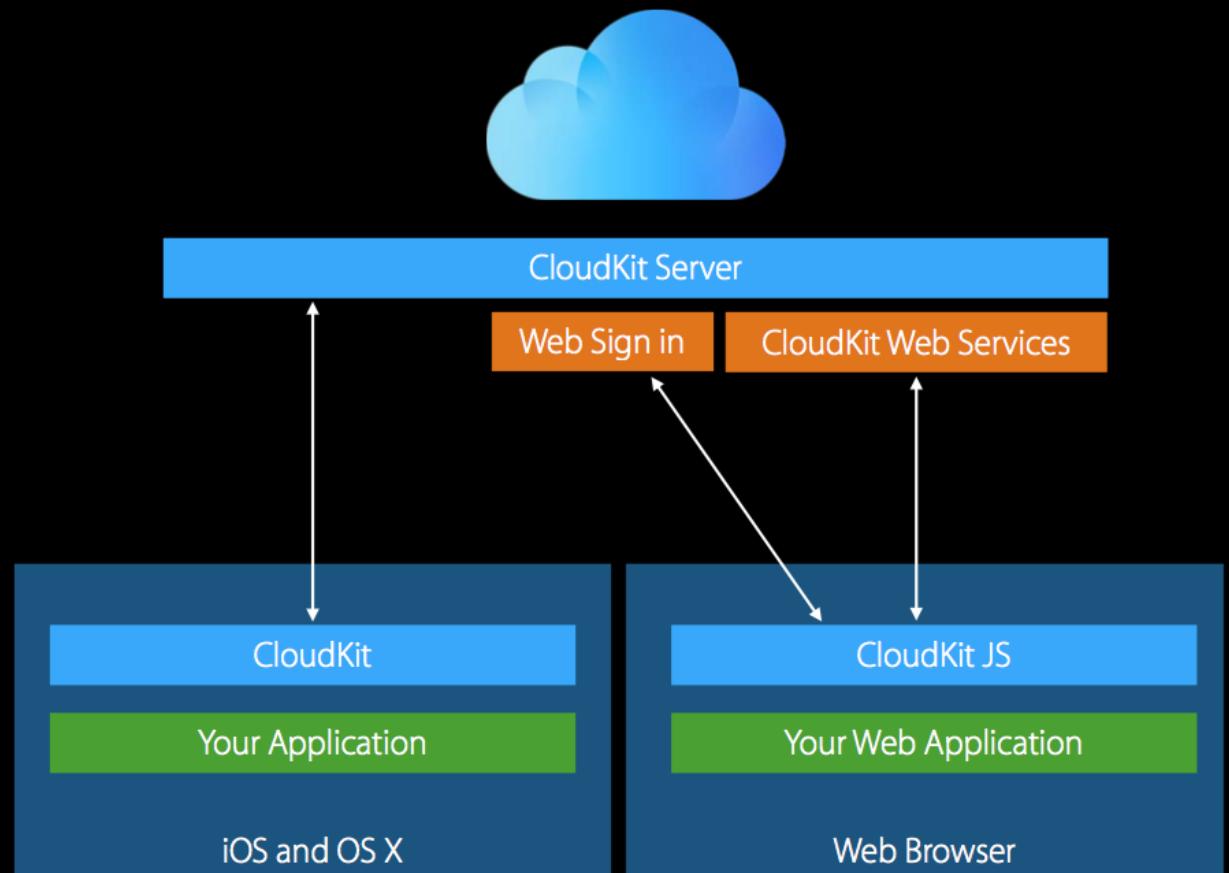
## WEB SERVICE API

- 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



# CLOUDKIT JS

- An API token is used to allow a web front-end to send requests to this container and environment
- A server-to-server key allows a server-side application to read and write data in the public database of this container & environment

## API Access

### API Tokens

#### Token

API Tokens allow a web frontend to send requests to this container and environment

#### Server-to-Server Keys

A server-to-server key allows a server-side application to read and write data in the public database of this container and environment.



"Token" De

API Access

Name

Sign In Callba

Allowed Origin

Discoverability

Notes

# CLOUDKIT JS

- Create a new token

## API Access

### API Tokens

#### Token

API Tokens allow a web frontend to send requests to this container and environment

#### Server-to-Server Keys

A server-to-server key allows a server-side application to read and write data in the public database of this container and environment.



New ▾



Revoke

### "Token" Details

#### API Access

Shown after the token has been created.

#### Name

Token

#### Sign In Callback

postMessage

Allows the Apple ID sign in browser window to communicate the authentication result to the web application using the CloudKit JS postMessage API.

URL Redirect

Redirects the browser back to this URL after the Apple ID sign in has completed.

https://

Enter a custom URL...

#### Allowed Origins

Any domain

Only the following domain(s):

https://



#### Discoverability

Request user discoverability at sign in.

#### Notes

Save

# CLOUDKIT JS

CloudKit Catalog

Run Code

Unauthenticated User

Container: iCloud.cloud.uchicago.CloudyWithAChanceOfErrors Environment: development

Class: Container

`.setUpAuth()`

This sample demonstrates how to authenticate a user with your app. There are two steps to authentication:

- Setting up auth.** This step checks whether a user is signed in. If you have specified `auth.persist = true` in your configuration you could run `setUpAuth` while bootstrapping your app and this function will use the stored cookie. The promise resolves with a `userIdentity` object or `null` and a sign-in or sign-out button will have been appended to the button container with id `apple-sign-in-button` or `apple-sign-out-button` (whichever is appropriate). These containers need to be in the DOM before executing the function and their IDs can be customized in `CloudKit.configure`.
- Binding handlers to the rendered button.** The promises `whenUserSignIn` and `whenUserSignsOut` are resolved when the user signs-in/out respectively. The former resolves with a `userIdentity` object.

If you selected the option *Request user discoverability at sign in* when creating your API token, a user will be able to grant discoverability permission to the app during the above sign-in flow.

```
function demoSetUpAuth() {
```



# CLOUDKIT JS

- Token is used to communicate with iCloud

API Token 1b20bfb72a908993605ca3662ef31c120851405732013b7a012a93019e

Name Javascript Token

Sign In Callback  postMessage

Allows the Apple ID sign in browser window to communicate the authentication result back to your web application using the JavaScript postMessage API.

https://  Enter a custom URL...

Redirects the browser back to this URL after the Apple ID sign in has completed.

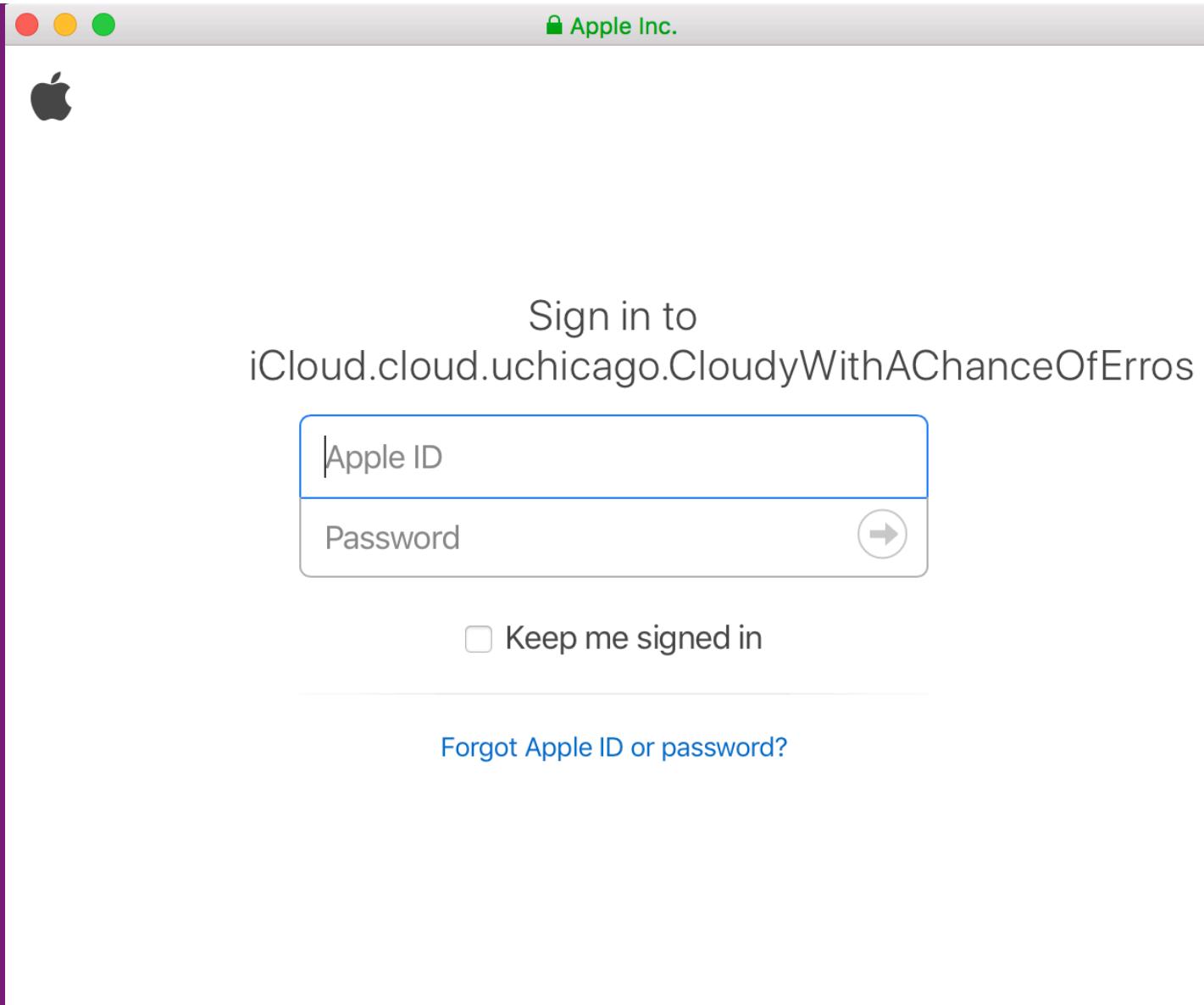
Allowed Origins  Any domain

Specific domains:

Discoverability  Request user discoverability at sign in

## CLOUDKIT JS

- Authentication happens with Apple provided UI



## CLOUDKIT JS

- Test directly in CloudKitCatalog web app

🔒 Apple Inc. [US] | <https://cdn.apple-cloudkit.com/ck-auth/?prtn=38&host=setup.apple-cloud...>



# Look Me Up By Email

Allow people using  
**iCloud.cloud.uchicago.CloudyWithAChanceOfErrors**  
to look you up by email?

People who know your email address will be able to see your first and last name.

Don't Allow | **Allow**

# CLOUDKIT JS

- Test directly in CloudKitCatalog web app

The screenshot shows the CloudKit Catalog web application interface. At the top, there's a navigation bar with the title "CloudKit Catalog" and a "Run Code" button. Below the navigation bar is a sidebar containing several sections: "README", "Authentication" (which is highlighted with a blue background), "Discoverability", "Query", "Zones", "Records", "Sync", "Sharing", "Subscriptions", and "Notifications" (with a note "Disconnected"). To the right of the sidebar, the main content area displays the result of a code run. It shows the container name "iCloud.cloud.uchicago.CloudyWithAChanceOfErrors" and environment "development". The result itself is a simple text output: "Result" followed by the name "Andrew Binkowski" and a "Sign out" button.

CloudKit Catalog

Run Code

Container: iCloud.cloud.uchicago.CloudyWithAChanceOfErrors Environment: development

Result

Andrew Binkowski

Sign out

README

Authentication

Discoverability

Query

Zones

Records

Sync

Sharing

Subscriptions

Notifications  
Disconnected

# SERVER-TO-SERVER WITH CLOUDKIT

## SERVER-TO-SERVER

- Late 2016 Apple opened server-to-server communication
- Allows a server to communicate with CloudKit using a server-to-server key
- The missing link in data processing and server side logic 🤔



## SERVER-TO-SERVER

- API calls can only be made to the public database
- Calls run as with the inherited privileges of the creator of the key



# SERVER-TO-SERVER

- Apple provides demo for node.js (supported by App Engine Flex)



## CloudKit Catalog

- README
- CloudKit on the web
- Server-side CloudKit with node.js
- Authentication
- Discoverability
- Query
- Zones
- Records
- Sync
- Sharing
- Subscriptions
- Notifications  
Disconnected

## Run Code

Container: iCloud.cloud.uchicago.CloudyWithAChanceOfErrors Environment: development

### Server-side CloudKit with node.js

A powerful CloudKit feature is the ability to make API calls with a server script. This feature is enabled by adding a server-to-server key in the API Access section of [CloudKit Dashboard](#). Such a key allows a server script to interact with CloudKit and make API calls to the **public database** with the inherited privileges of the creator of the key. In this section we will explain this process for a node.js script using CloudKit JS.

#### Creating a server-to-server key

If you are on a Mac, you already have OpenSSL installed and you can generate a private key in Terminal using the following command:

```
openssl ecparam -name prime256v1 -genkey -noout -out eckey.pem
```

This will create the file **eckey.pem** in your working directory. In [CloudKit Dashboard](#) navigate to API Access > Server-to-Server Keys -> Add Server-to-Server Key and paste the output of the following command into the **Key ID** field of the new key.

```
openssl ec -in eckey.pem -pubout
```

Hit Save and the **Key ID** attribute will get populated. You will use this key ID in configuring your node.js script.

#### Installing dependencies

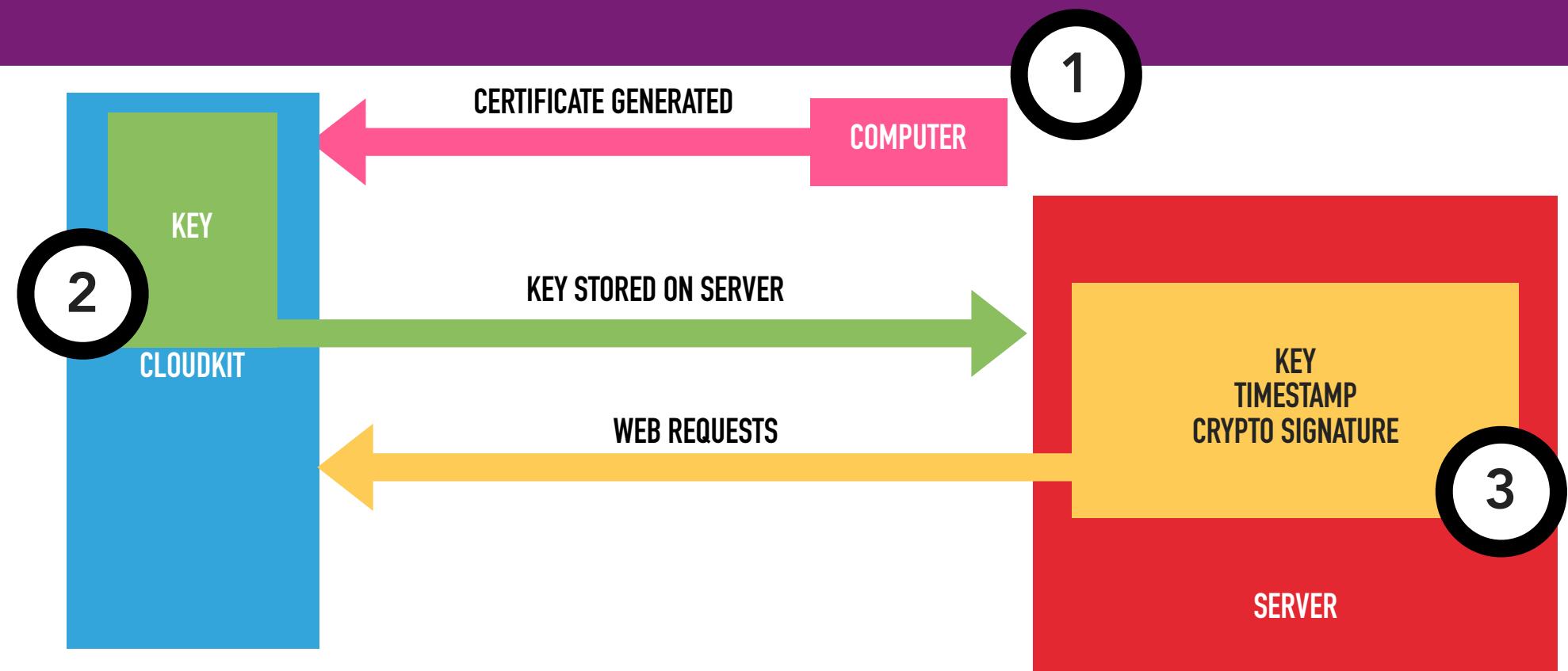
In order to use CloudKit JS server-side you will need a *fetch* implementation such as **node-fetch** which you can install via NPM. You must also download CloudKit JS itself from Apple's CDN.

```
npm install node-fetch
curl https://cdn.apple-cloudkit.com/ck/2/cloudkit.js > cloudkit.js
```

#### Configuring CloudKit JS in a node script

Create a script file in your working directory and add the following configuration code.

## SERVER-TO-SERVER



# SERVER-TO-SERVER KEY

# SERVER-TO-SERVER KEY

- Create a public/private key pair to communicate with the iCloud servers

Guides and Sample Code

CloudKit Web Services Reference

Table of Contents

- Introduction
- Composing Web Service Requests
- Modifying Records (records/modify)
- Fetching Records Using a Query (records/query)
- Fetching Records by Record Name (records/lookup)
- Fetching Record Changes (records/changes)
- Fetching Record Information (records/resolve)
- Accepting Share Records (records/accept)
- Uploading Assets (assets/upload)
- Referencing Existing Assets (assets/refererence)
- Fetching Zones (zones/list)
- Fetching Zones by Identifier (zones/lookup)
- Modifying Zones (zones/modify)
- Fetching Database Changes (changes/database)
- Fetching Record Zone Changes (changes/zone)
- Fetching Zone Changes (zones/changes)
- Fetching Current User Identity (users/caller)
- Discovering User Identities (POST users/discover)
- Discovering All User Identities (GET users/discover)
- Fetching Current User (users/current)
- Fetching Contacts (users/lookup/contacts)
- Fetching Users by Email (users/lookup/email)
- Fetching Users by Record Name (users/lookup/id)
- Modifying Subscriptions (subscriptions/modify)
- Fetching Subscriptions (subscriptions/list)
- Fetching Subscriptions by Identifier (subscriptions/lookup)
- Creating APNs Tokens (tokens/create)
- Registering Tokens (tokens/register)

## Accessing CloudKit Using a Server-to-Server Key

Use a server-to-server key to access the public database of a container as the developer who created the key. You create the server-to-server certificate (that includes the private and public key) locally. Then use CloudKit Dashboard to enter the public key and create a key ID that you include in the subpath of your web services requests.

See [CloudKit Catalog: An Introduction to CloudKit \(Cocoa and JavaScript\)](#) for a JavaScript sample that uses a server-to-server key.

### Creating a Server-to-Server Certificate

You create the certificate, containing the private and public key, on your Mac. The certificate never expires but you can revoke it.

**To create a server-to-server certificate**

1. Launch Terminal.
2. Enter this command:

```
openssl ecparam -name prime256v1 -genkey -noout -out eckey.pem
```

A `eckey.pem` file appears in the current folder.

You'll need the public key from the certificate to enter in CloudKit Dashboard later.

**To get the public key for a server-to-server certificate**

1. In Terminal, enter this command:

```
openssl ec -in eckey.pem -pubout
```

The public key appears in the output.

```
read EC key
writing EC key
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEExnKj6w8e3pxjtaOUfaNNjsnXHgWH
nQA3TzMt5P32tK8PjLHzpPm6doaDvGKZcs99YAXj0+u5pe9PtsmBKWTuWA==
-----END PUBLIC KEY-----
```

**Important:** Protect your private key as you would an account password. The private key is stored locally on your Mac and can't be retrieved if deleted. If someone else has your private key, that person may be able to impersonate you. Therefore, keep a secure backup of your public-private key pair.

### Storing the Server-to-Server Public Key and Getting the Key Identifier

# SERVER-TO-SERVER KEY

- [https://developer.apple.com/library/content/documentation/DataManagement/Conceptual/CloudKitWebServicesReference/SettingUpWebServices/SettingUpWebServices.html#/apple\\_ref/doc/uid/TP40015240-CH24-SW6](https://developer.apple.com/library/content/documentation/DataManagement/Conceptual/CloudKitWebServicesReference/SettingUpWebServices/SettingUpWebServices.html#/apple_ref/doc/uid/TP40015240-CH24-SW6)

## Guides and Sample Code

Apple Developer

### CloudKit Web Services Reference

#### Table of Contents

- ▶ Introduction
- ▶ Composing Web Service Requests
- ▶ Modifying Records (records/modify)
- ▶ Fetching Records Using a Query (records/query)
- ▶ Fetching Records by Record Name (records/lookup)
- ▶ Fetching Record Changes (records/changes)
- ▶ Fetching Record Information (records/resolve)
- ▶ Accepting Share Records (records/accept)
- ▶ Uploading Assets (assets/upload)
- ▶ Referencing Existing Assets (assets/referrence)
- ▶ Fetching Zones (zones/list)
- ▶ Fetching Zones by Identifier (zones/lookup)
- ▶ Modifying Zones (zones/modify)
- ▶ Fetching Database Changes (changes/database)
- ▶ Fetching Record Zone Changes (changes/zone)
- ▶ Fetching Zone Changes (zones/changes)
- ▶ Fetching Current User Identity (users/caller)
- ▶ Discovering User Identities (POST users/discover)
- ▶ Discovering All User Identities (GET users/discover)
- ▶ Fetching Current User (users/current)
- ▶ Fetching Contacts (users/lookup/contacts)
- ▶ Fetching Users by Email (users/lookup/email)
- ▶ Fetching Users by Record Name (users/lookup/id)
- ▶ Modifying Subscriptions (subscriptions/modify)
- ▶ Fetching Subscriptions (subscriptions/list)
- ▶ Fetching Subscriptions by Identifier (subscriptions/lookup)
- ▶ Creating APNs Tokens (tokens/create)
- ▶ Registering Tokens (tokens/register)

## Accessing CloudKit Using a Server-to-Server Key

Use a server-to-server key to access the public database of a container as the developer who created the key. You create the server-to-server certificate (that includes the private and public key) locally. Then use CloudKit Dashboard to enter the public key and create a key ID that you include in the subpath of your web services requests.

See [CloudKit Catalog: An Introduction to CloudKit \(Cocoa and JavaScript\)](#) for a JavaScript sample that uses a server-to-server key.

### Creating a Server-to-Server Certificate

You create the certificate, containing the private and public key, on your Mac. The certificate never expires but you can revoke it.

#### To create a server-to-server certificate

1. Launch Terminal.
2. Enter this command:

```
openssl ecparam -name prime256v1 -genkey -noout -out eckey.pem
```

A eckey.pem file appears in the current folder.

You'll need the public key from the certificate to enter in CloudKit Dashboard later.

#### To get the public key for a server-to-server certificate

1. In Terminal, enter this command:

```
openssl ec -in eckey.pem -pubout
```

The public key appears in the output.

```
read EC key
writing EC key
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAExnKj6w8e3pxjtaOUfaNNjsnXHgWH
nQA3TzMt5P32tK8PjLHzpPm6doaDvGKZcs99YAXj0+u5pe9PtsmBKWTuWA==
-----END PUBLIC KEY-----
```

**Important:** Protect your private key as you would an account password. The private key is stored locally on your Mac and can't be retrieved if deleted. If someone else has your private key, that person may be able to impersonate you. Therefore, keep a secure backup of your public-private key pair.

### Storing the Server-to-Server Public Key and Getting the Key Identifier

# SERVER-TO-SERVER KEY

- Protect your key
  - Back it up
  - Don't post on GitHub as part of project
- Can be revoked if loose track of it

Guides and Sample Code

CloudKit Web Services Reference

Table of Contents

- Introduction
- Composing Web Service Requests
- Modifying Records (records/modify)
- Fetching Records Using a Query (records/query)
- Fetching Records by Record Name (records/lookup)
- Fetching Record Changes (records/changes)
- Fetching Record Information (records/resolve)
- Accepting Share Records (records/accept)
- Uploading Assets (assets/upload)
- Referencing Existing Assets (assets/refererence)
- Fetching Zones (zones/list)
- Fetching Zones by Identifier (zones/lookup)
- Modifying Zones (zones/modify)
- Fetching Database Changes (changes/database)
- Fetching Record Zone Changes (changes/zone)
- Fetching Zone Changes (zones/changes)
- Fetching Current User Identity (users/caller)
- Discovering User Identities (POST users/discover)
- Discovering All User Identities (GET users/discover)
- Fetching Current User (users/current)
- Fetching Contacts (users/lookup/contacts)
- Fetching Users by Email (users/lookup/email)
- Fetching Users by Record Name (users/lookup/id)
- Modifying Subscriptions (subscriptions/modify)
- Fetching Subscriptions (subscriptions/list)
- Fetching Subscriptions by Identifier (subscriptions/lookup)
- Creating APNs Tokens (tokens/create)
- Registering Tokens (tokens/register)

## Accessing CloudKit Using a Server-to-Server Key

Use a server-to-server key to access the public database of a container as the developer who created the key. You create the server-to-server certificate (that includes the private and public key) locally. Then use CloudKit Dashboard to enter the public key and create a key ID that you include in the subpath of your web services requests.

See [CloudKit Catalog: An Introduction to CloudKit \(Cocoa and JavaScript\)](#) for a JavaScript sample that uses a server-to-server key.

### Creating a Server-to-Server Certificate

You create the certificate, containing the private and public key, on your Mac. The certificate never expires but you can revoke it.

**To create a server-to-server certificate**

1. Launch Terminal.
2. Enter this command:

```
openssl ecparam -name prime256v1 -genkey -noout -out eckey.pem
```

A eckey.pem file appears in the current folder.

You'll need the public key from the certificate to enter in CloudKit Dashboard later.

**To get the public key for a server-to-server certificate**

1. In Terminal, enter this command:

```
openssl ec -in eckey.pem -pubout
```

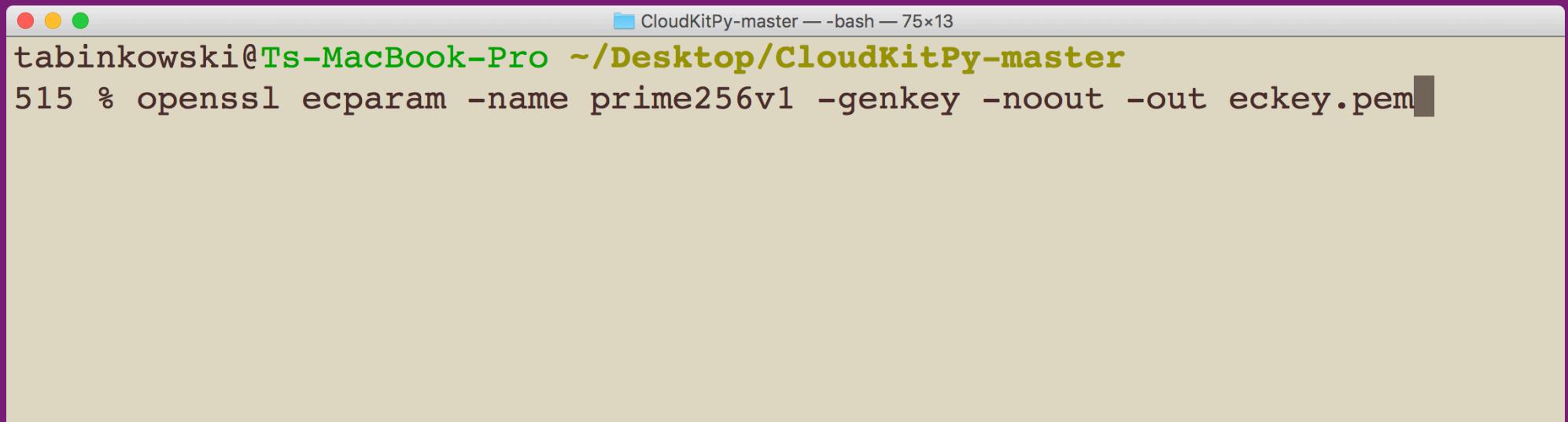
The public key appears in the output.

```
read EC key
writing EC key
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAExnKj6w8e3pxjtaOUfaNNjsnXHgWH
nQA3TzMt5P32tK8PjLHzpPm6doaDvGKZcs99YAXj0+u5pe9PtsmBKWTuWA==
-----END PUBLIC KEY-----
```

**Important:** Protect your private key as you would an account password. The private key is stored locally on your Mac and can't be retrieved if deleted. If someone else has your private key, that person may be able to impersonate you. Therefore, keep a secure backup of your public-private key pair.

### Storing the Server-to-Server Public Key and Getting the Key Identifier

## SERVER-TO-SERVER KEY



A screenshot of a macOS terminal window titled "CloudKitPy-master — bash — 75x13". The window shows the command: "tabinkowski@Ts-MacBook-Pro ~/Desktop/CloudKitPy-master 515 % openssl ecparam -name prime256v1 -genkey -noout -out eckey.pem". The terminal has a light beige background and a dark grey header bar.

```
CloudKitPy-master — bash — 75x13
tabinkowski@Ts-MacBook-Pro ~/Desktop/CloudKitPy-master
515 % openssl ecparam -name prime256v1 -genkey -noout -out eckey.pem
```

- Create the server-to-server certificate
- A eckey.pem file will be generated

SERVER-TO-SERVER KEY

# OpenSSL

Cryptography and SSL/TLS Toolkit

[Home](#) | [Blog](#) | [Downloads](#) | [Docs](#) | [News](#) | [Policies](#) | [Community](#) | [Support](#)

## Welcome to OpenSSL!

OpenSSL is an open source project that provides a robust, commercial-grade, and full-featured toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. It is also a general-purpose

### Home

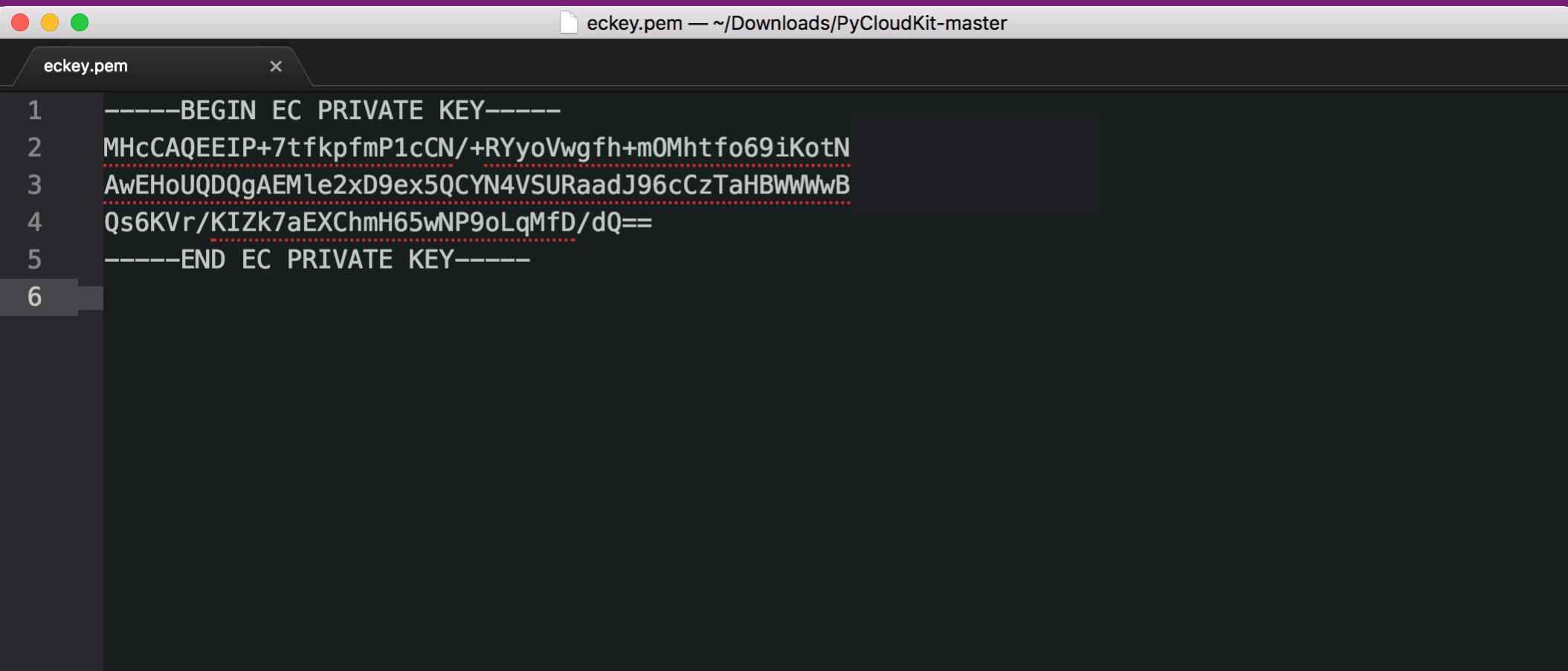
[Downloads: Source code](#)

[Docs: FAQ, FIPS, manpages, ...](#)

[News: Latest information](#)

[Policies: How we operate](#)

# SERVER-TO-SERVER KEY



A screenshot of a terminal window on a Mac OS X system. The window title is "eckey.pem — ~/Downloads/PyCloudKit-master". The file content is displayed in a monospaced font. The text shows a PEM-formatted EC private key, starting with "-----BEGIN EC PRIVATE KEY-----", followed by a long string of base64-encoded data, and ending with "-----END EC PRIVATE KEY-----". The terminal has a dark theme with red selection highlights.

```
1 -----BEGIN EC PRIVATE KEY-----
2 MhCAQEEIP+7tfkpfmP1cCN/+RYyoVwgfh+m0Mhtfo69iKotN
3 AwEHoUQDQgAEMle2xD9ex50CYN4VSURaadJ96cCzTaHBWwWwB
4 Qs6KVr/KIZk7aEXChmH65wNP9oLqMfD/dQ==
5 -----END EC PRIVATE KEY-----
6
```

## SERVER-TO-SERVER KEY

```
CloudKitPy-master — bash — 75x13
tabinkowski@Ts-MacBook-Pro ~/Desktop/CloudKitPy-master
[523 % openssl ec -in eckey.pem -pubout
read EC key
writing EC key
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEMle2xD9ex
TaHBWWWwBcp6a7JZwyRCboGaQs6KVr/KIZk7aEXChmH65
-----END PUBLIC KEY-----
tabinkowski@Ts-MacBook-Pro ~/Desktop/CloudKitPy-master
524 %
```

- Generate the public key
- Enter in CloudKit console

# SERVER-TO-SERVER KEY

## API Access

### API Tokens

### Javascript Token

API Tokens allow a web frontend to send requests to this container and environment

### Server-to-Server Keys

#### App Engine Keys

CREATE A NEW KEY

A server-side key is a server-side key that allows your app to access data in the CloudKit database. It's stored in the container and environment.



New ▾



Revoke

### "App Engine Key" Details

Key ID Shown after the key has been created.

Name App Engine Key

Public Key 1. Generate a private key in Terminal using:

```
openssl ecparam -name prime256v1 -genkey -noout -out eckey.pem
```

Note: Never expose or share the above private key with anyone, including Apple, as it has unrestricted access to your public database.

2. Output a public key to be stored with the CloudKit server:

```
openssl ec -in eckey.pem -pubout
```

```
MFkwEwYHKOZlZj0CAQYIKoZlZj0DAQcDQgAETwxGHYZEuNnb  
5dwIPJHon1MH3fV7  
OMQOCR3p2WANz2OaewWO4S+5LP+H9JIG8jCIWy2Dq5eSh
```

# SERVER-TO-SERVER KEY

## API Access

### API Tokens

#### Javascript Token

API Tokens allow a web frontend to send requests to this container and environment

### Server-to-Server Keys

#### App Engine Key

A server-to-server key allows a server-side application to read and write data in the public database of this container and environment.

## "App Engine Key" Details

Key ID 99e04a281f141f4bce822833bd8e87827b914

Name App Engine Key

Public Key MFkwEwYHKOZIzj0CAQYIKoZIzj0DAQcDQgAE  
OMQOCR3p2WANz2OaewWO4S+5LP+H9JIG

### Notes

# WEB SERVICE REQUESTS

# WEB SERVICE REQUESTS

- When using a server-to-server key, you sign the web service request
- Key is authentication
- Developer allowed access

## To create a signed request using the server-to-server certificate in your keychain

1. Concatenate the following parameters and separate them with colons.

```
[Current date]:[Request body]:[Web service URL subpath]
```

### *Current date*

The ISO8601 representation of the current date (without milliseconds)—for example, 2016-01-25T22:15:

### *Request body*

The base64 string encoded SHA-256 hash of the body.

### *Web service URL subpath*

The URL described in [The Web Service URL](#) but without the [path] component, as in:

```
/database/1/iCloud.com.example.gkumar.MyApp/development/public/records/query
```

**Note:** If you include the API token subpath, described in [Accessing CloudKit Using an API Token](#), the requ

2. Compute the ECDSA signature of this message with your private key.

3. Add the following request headers.

```
X-Apple-CloudKit-Request-KeyID: [keyID]  
X-Apple-CloudKit-Request-ISO8601Date: [date]  
X-Apple-CloudKit-Request-SignatureV1: [signature]
```

### *keyID*

The identifier for the server-to-server key obtained from CloudKit Dashboard, described in [Storing the Server-to-Server Key](#) and [Getting the Key Identifier](#).

### *date*

The ISO8601 representation of the current date (without milliseconds).

### *signature*

The signature created in Step 2.

## WEB SERVICE REQUESTS

- All requests must have proper headers
- Only valid for 10 minutes

```
X-Apple-CloudKit-Request-KeyID: [keyID]  
X-Apple-CloudKit-Request-ISO8601Date: [date]  
X-Apple-CloudKit-Request-SignatureV1: [signature]
```

### *keyID*

The identifier for the server-to-server key obtained from CloudKit Data [Getting the Key Identifier](#).

### *date*

The ISO8601 representation of the current date (without milliseconds).

### *signature*

The signature created in Step 2.

# WEB SERVICE REQUESTS

QUERY

URL:

`https://api.apple-cloudkit.com/database/1/  
iCloud.cloud.uchicago.CloudyWithAChanceOfErros/development/  
public/records/query`

Data: `{"query": {"recordType": "joke"} }`

Headers: {

`'X-Apple-CloudKit-Request-ISO8601Date': '2017-05-10T04:33:33Z',  
'X-Apple-CloudKit-Request-SignatureV1':u'MEUCIQ...TfA=',  
'X-Apple-CloudKit-Request-KeyID':'ffdb30c23add...00191'}`

# WEB SERVICE REQUESTS

- **records/resolve:** Fetches information about records using GUIDs, described in [Fetching Record Information \(records/resolve\)](#).
- **records/accept:** Accepts a share on behalf of the current user, described in [Accepting Share Records \(records/accept\)](#).
- **changes/database:** Fetches changed zones in a database, described in [Fetching Database Changes \(changes/database\)](#).
- **changes/zone:** Fetches changed records in the specified zones, described in [Fetching Record Zone Changes \(changes/zone\)](#).
- **users/caller:** Fetches information about the current user, described in [Fetching Current User \(users/current\)](#).
- **GET users/discover:** Fetches all user identities in the current user's address book, described in [Discovering All User Identities \(GET users/discover\)](#).
- **POST users/discover:** Fetches all users in the specified array, described in [Discovering User Identities \(POST users/discover\)](#).

Updated endpoints to support sharing records:

- **records/changes:** Added `shared` as database value, described in [Fetching Record Changes \(records/changes\)](#).
- **records/modify:** Added `shared` as database value, described in [Modifying Records \(records/modify\)](#).
- **records/lookup:** Added `shared` as database value, described in [Fetching Records by Record Name \(records/lookup\)](#).

- API Endpoints for CloudKit
- Look at CloudKit catalog for API examples

# WEB SERVICE REQUEST QUERY

# WEB SERVICE REQUESTS

- API for fetching records

## Fetching Records Using a Query (records/query)

You can fetch records using a query.

### Path

POST [path]/database/[version]/[container]/[environment]/[database]/records/query

### Parameters

#### *path*

The URL to the CloudKit web service, which is <https://api.apple-cloudkit.com>.

#### *version*

The protocol version—currently, 1.

#### *container*

A unique identifier for the app's container. The container ID should begin with `iCloud..`

#### *environment*

The version of the app's container. Pass `development` to use the environment that is not accessible by apps available on the store. environment that is accessible by development apps and apps available on the store.

#### *database*

The database to store the data within the container. The possible values are:

##### `public`

The database that is accessible to all users of the app.

##### `private`

The database that contains private data that is visible only to the current user.

##### `shared`

The database that contains records shared with the current user.

# WEB SERVICE REQUESTS

## Request

The POST request is a JSON dictionary containing the following keys:

Key	Description
zoneID	A dictionary that identifies a record zone in the database, described in <a href="#">Zone ID Dictionary</a> .
resultsLimit	The maximum number of records to fetch. The default is the maximum number of records in a response that is allowed, described in <a href="#">Data Size Limits</a> .
query	The query to apply, described in <a href="#">Query Dictionary</a> . This key is required.
continuationMarker	The location of the last batch of results. Use this key when the results of a previous fetch exceeds the maximum. See <a href="#">Response</a> . The default value is <code>null</code> .
desiredKeys	An array of strings containing record field names that limits the amount of data returned in this operation. Only the fields specified in the array are returned. The default is <code>null</code> , which fetches all record fields.
zoneWide	A Boolean value determining whether all zones should be searched. This key is ignored if <code>zoneID</code> is non- <code>null</code> . To search all zones, set to <code>true</code> . To search the default zone only, set to <code>false</code> .
numbersAsStrings	A Boolean value indicating whether number fields should be represented by strings. The default value is <code>false</code> .

- POST request consists of a JSON dictionary

# WEB SERVICE REQUESTS

- Example query

```
"{
  "zoneID": {
    "zoneName": "myCustomZone",
  },
  "query": {
    "recordType": "myRecordType",
    "filterBy": [
      {
        "systemFieldName": "createdUserRecordName",
        "comparator": "EQUALS",
        "fieldValue": {
          "value": {
            "recordName": "recordA",
          },
          "type": "REFERENCE"
        }
      }
    ],
    "sortBy": [
      {
        "systemFieldName": "createdTimestamp",
        "ascending": false
      }
    ]
  }
}"
```

# WEB SERVICE REQUESTS

- Example query operations

## Record Operation Dictionary

The operation dictionary keys are:

Key	Description
operationType	The type of operation. Possible values are described in <a href="#">Operation Type Values</a> . This key is required.
record	A dictionary representing the record to modify, as described in <a href="#">Record Dictionary</a> . This key is required.
desiredKeys	An array of strings containing record field names that limits the amount of data returned in this operation. Only the fields specified in the array are returned. The default is <code>null</code> , which fetches all record fields. This <code>desiredKeys</code> setting overrides the <code>desiredKeys</code> setting in the enclosing dictionary.

## Operation Type Values

The possible values for the `operationType` key are:

Value	Description
create	Create a new record. This operation fails if a record with the same record name already exists.
update	Update an existing record. Only the fields you specify are changed.
forceUpdate	Update an existing record regardless of conflicts. Creates a record if it doesn't exist.
replace	Replace a record with the specified record. The fields whose values you do not specify are set to <code>null</code> .
forceReplace	Replace a record with the specified record regardless of conflicts. Creates a record if it doesn't exist.
delete	Delete the specified record.
forceDelete	Delete the specified record regardless of conflicts.

# WEB SERVICE REQUESTS

## Response

An array of dictionaries describing the results of the operation. The dictionary contains the following keys:

Key	Description
records	An array containing a result dictionary for each record requested. If successful, the result dictionary contains the keys described in <a href="#">Record Dictionary</a> . If unsuccessful, the result dictionary contains the keys described in <a href="#">Record Fetch Error Dictionary</a> .
continuationMarker	If included, indicates that there are more results matching this query. To fetch the other results, pass the value of the <code>continuationMarker</code> key as the value of the <code>continuationMarker</code> key in another query.

- Response contains dictionary of records
- Returned results are limited by data (unless specified)

# WEB SERVICE REQUESTS

- Returned results are limited by data (unless specified by the `resultsLimit` key in request)
- `continuationMarker` for the next batch

## Data Size Limits

These are the limits on the size of data sent to and from the CloudKit server.

Property	Value
Maximum number of operations in a request	200
Maximum number of records in a response	200
Maximum number of tokens in a request	200
Maximum record size (not including Asset fields)	1 MB
Maximum file size of an Asset field	50 MB
Maximum number of source references to a single target where the action is delete self	750

# WEB SERVICE REQUESTS

URL:

`https://api.apple-cloudkit.com/database/1/  
iCloud.cloud.uchicago.CloudyWithAChanceOfErros/development/  
public/records/query`

API URL

Data: `{"query": {"recordType": "joke"} }`

QUERY JSON

Headers: `{  
'X-Apple-CloudKit-Request-ISO8601Date': '2017-05-10T04:33:33Z',  
'X-Apple-CloudKit-Request-SignatureV1': u'MEUCIQ...TfA=',  
'X-Apple-CloudKit-Request-KeyID': 'ffdb30c23add...00191'}`

# MODIFY RECORDS

# WEB SERVICE REQUESTS

- Apply multiple types of operations
  - Creating
  - Updating
  - Replacing
  - Deleting
- Multiple records can be modified in a single request

## Path

POST [path]/database/[version]/[container]/[environment]/[database]/records/modify

## Parameters

### *path*

The URL to the CloudKit web service, which is <https://api.apple-cloudkit.com>.

### *version*

The protocol version—currently, 1.

### *container*

A unique identifier for the app's container. The container ID begins with iCloud..

### *environment*

The version of the app's container. Pass `development` to use the environment that is not accessible by apps available on the store. environment that is accessible by development apps and apps available on the store.

### *database*

The database to store the data within the container. The possible values are:

#### *public*

The database that is accessible to all users of the app.

#### *private*

The database that contains private data that is visible only to the current user.

#### *shared*

The database that contains records shared with the current user.

# WEB SERVICE REQUESTS

Key	Description
operations	An array of dictionaries defining the operations to apply to records in the database. The dictionary keys are described in <a href="#">Record Operation Dictionary</a> . See <a href="#">Data Size Limits</a> for maximum number of operations allowed. This key is required.
zoneID	A dictionary that identifies a record zone in the database, described in <a href="#">Zone ID Dictionary</a> .
atomic	A Boolean value indicating whether the entire operation fails when one or more operations fail. If <code>true</code> , the entire request fails if one operation fails. If <code>false</code> , some operations may succeed and others may fail. The default value is <code>true</code> . Note this property only applies to custom zones.
desiredKeys	An array of strings containing record field names that limit the amount of data returned in the enclosing operation dictionaries. Only the fields specified in the array are returned. The default is <code>null</code> , which fetches all record fields.
numbersAsStrings	A Boolean value indicating whether number fields should be represented by strings. The default value is <code>false</code> .

- POST request consists of a JSON dictionary

## WEB SERVICE REQUESTS

```
{  
  "operationType" : "create",  
  "record" : {  
    "recordType" : "Artist",  
    "fields" : {  
      "firstName" : {"value" : "Mei"},  
      "lastName" : {"value" : "Chen"}  
    }  
    "recordName" : "Mei Chen"  
  },  
}
```

YOUR CUSTOM FIELDS

OMIT RECORDNAME AND IT WILL BE  
AUTOMATICALLY  
GENERATED AS UUID

- POST request consists of a JSON dictionary
- Schema is not on-demand for JS API

# WEB SERVICE REQUESTS

ADD A RECORD

URL: <https://api.apple-cloudkit.com/database/1/icloud.cloud.uchicago.CloudyWithAChanceOfErros/development/public/records/modify>

Data: {"operations": [ {"record": {"fields": {"joke": {"value": "A Cloudy Day in Chicago"}, "recordName": "60d82b7d-32ef-4e91-9c12-585c92a3bb89"}, "action": "DELETE\_SELF", "zoneID": {"zoneName": "DefaultZone"}}, "day": {"value": "today"}, "recordType": "Daily"}, {"operationType": "create"} ]}

Headers: {  
'X-Apple-CloudKit-Request-ISO8601Date': '2017-05-10T04:33:33Z',  
'X-Apple-CloudKit-Request-SignatureV1': u'MEUCIQ...TfA=',  
'X-Apple-CloudKit-Request-KeyID': 'ffdb30c23add...00191'}

# WEB SERVICE REQUESTS

ADD A RECORD

```
{"operations": [  
  {"record":  
    {"fields":  
      {"joke":  
        {"value":  
          {"recordName": "60d82b7d-32ef-4e91-9c12-585c92a3bb89",  
           "action": "DELETE_SELF",  
           "zoneID": {"zoneName": "_defaultZone"}  
        }  
      },  
      "day": {"value": "today"}},  
      "recordType": "Daily"},  
      "operationType": "create"}]  
}
```

# WEB SERVICE REQUESTS

## Response

The response is an array of dictionaries describing the results of the operation. The dictionary contains a single key:

Key	Description
records	An array containing a result dictionary for each operation in the request. If successful, the result dictionary contains the keys described in <a href="#">Record Dictionary</a> . If unsuccessful, the result dictionary contains the keys described in <a href="#">Record Fetch Error Dictionary</a> .

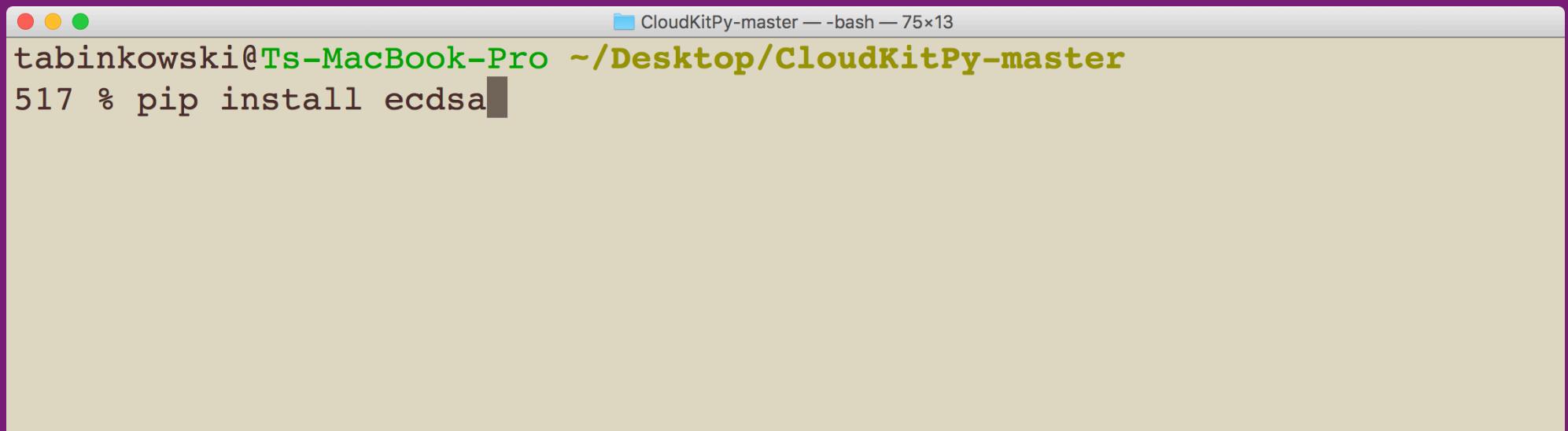
- Response is a dictionary with more information than you probably need

# WEB SERVICE REQUESTS

- Response dictionary

```
{  
    "created": {  
        "deviceID": "2",  
        "timestamp": 1494288550303,  
        "userRecordName": "_7b892f2a3eeadd6afc77ff7158f69d9"  
    },  
    "fields": {  
        "question": {  
            "type": "STRING",  
            "value": "Why?\n"  
        },  
        "response": {  
            "type": "STRING",  
            "value": "That's why!"  
        }  
    },  
    "modified": {  
        "deviceID": "2",  
        "timestamp": 1494386749697,  
        "userRecordName": "_7b892f2a3eeadd6afc77ff7158f69d9"  
    },  
    "pluginFields": {},  
    "recordChangeTag": "j2gstkiq",  
    "recordName": "60d82b7d-32ef-4e91-9c12-585c92a3bb89",  
    "recordType": "joke",  
    "zoneID": {  
        "ownerRecordName": "_7b892f2a3eeadd6afc77ff7158f69d9",  
        "zoneName": "_defaultZone"  
    }  
}
```

## SERVER-TO-SERVER KEY



A screenshot of a macOS terminal window titled "CloudKitPy-master — bash — 75x13". The window shows the command "pip install ecdsa" being typed by the user "tabinkowski@Ts-MacBook-Pro ~/Desktop/CloudKitPy-master". The terminal has a light beige background and a dark grey header bar.

```
CloudKitPy-master — bash — 75x13
tabinkowski@Ts-MacBook-Pro ~/Desktop/CloudKitPy-master
517 % pip install ecdsa
```

- Elliptic Curve Digital Signature Algorithm used to sign requests
- Cryptography approach for public/private key agreement

# SERVER-TO-SERVER DEMO

# SERVER-TO-SERVER KEY

- CloudKit API is JSON based so any language can be used
- Surprisingly few number of existing libraries
  - Even less that actually work

lionheart / requests-cloudkit Watch ▾

Code Issues 1 Pull requests 0 Pulse Graphs

This project provides Apple CloudKit server-to-server support for the requests Python library.

python rest-api cloudkit python-library

19 commits 1 branch 0 releases

Branch: master New pull request Create new file Upload file

dlo committed on GitHub Update README.rst

requests\_cloudkit bump version to 0.1.5

.gitignore first commit

.travis.yml add install script

LICENSE first commit

Makefile first commit

README.rst Update README.rst

circle.yml do python setup.py install for circle

requirements.txt first commit

setup.cfg first commit

setup.py fix setup script

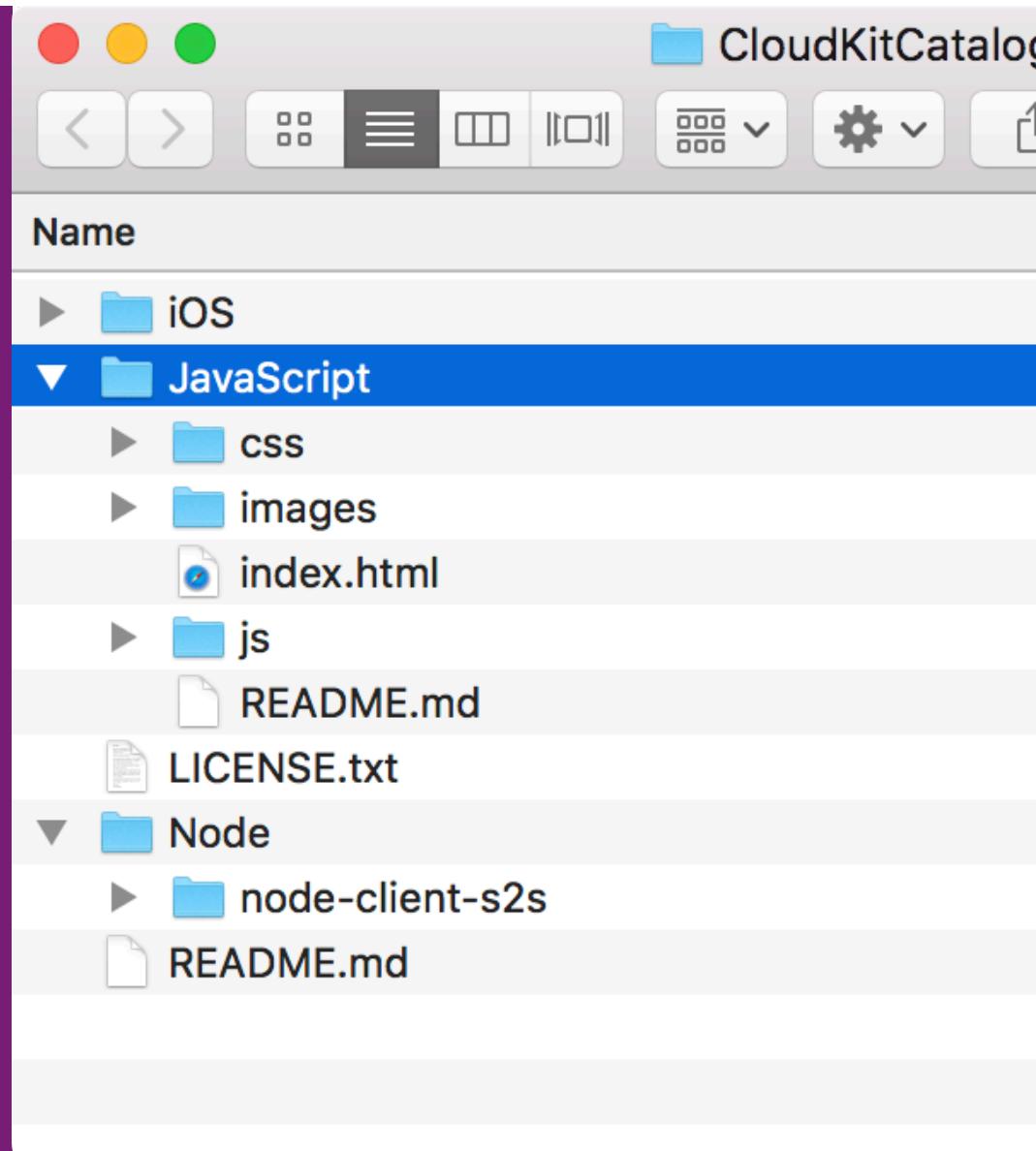
test\_requests\_cloudkit.py first commit

README.rst

**Requests-CloudKit** build passing pypi v0.1.5

## SERVER-TO-SERVER KEY

- Apple has a sample project showing iOS, JS, and Node sharing a CloudKit container
- CloudKitCatalogAnlroduction  
ToCloudKitCocoaJavaScript



# SERVER-TO-SERVER KEY

```
/*
Copyright (C) 2016 Apple Inc. All Rights Reserved.
See LICENSE.txt for this sample's licensing information

Abstract:
This node script uses a server-to-server key to make public database calls with CloudKit JS
*/

process.env.NODE_TLS_REJECT_UNAUTHORIZED = "0";

(function() {
  var fetch = require('node-fetch');

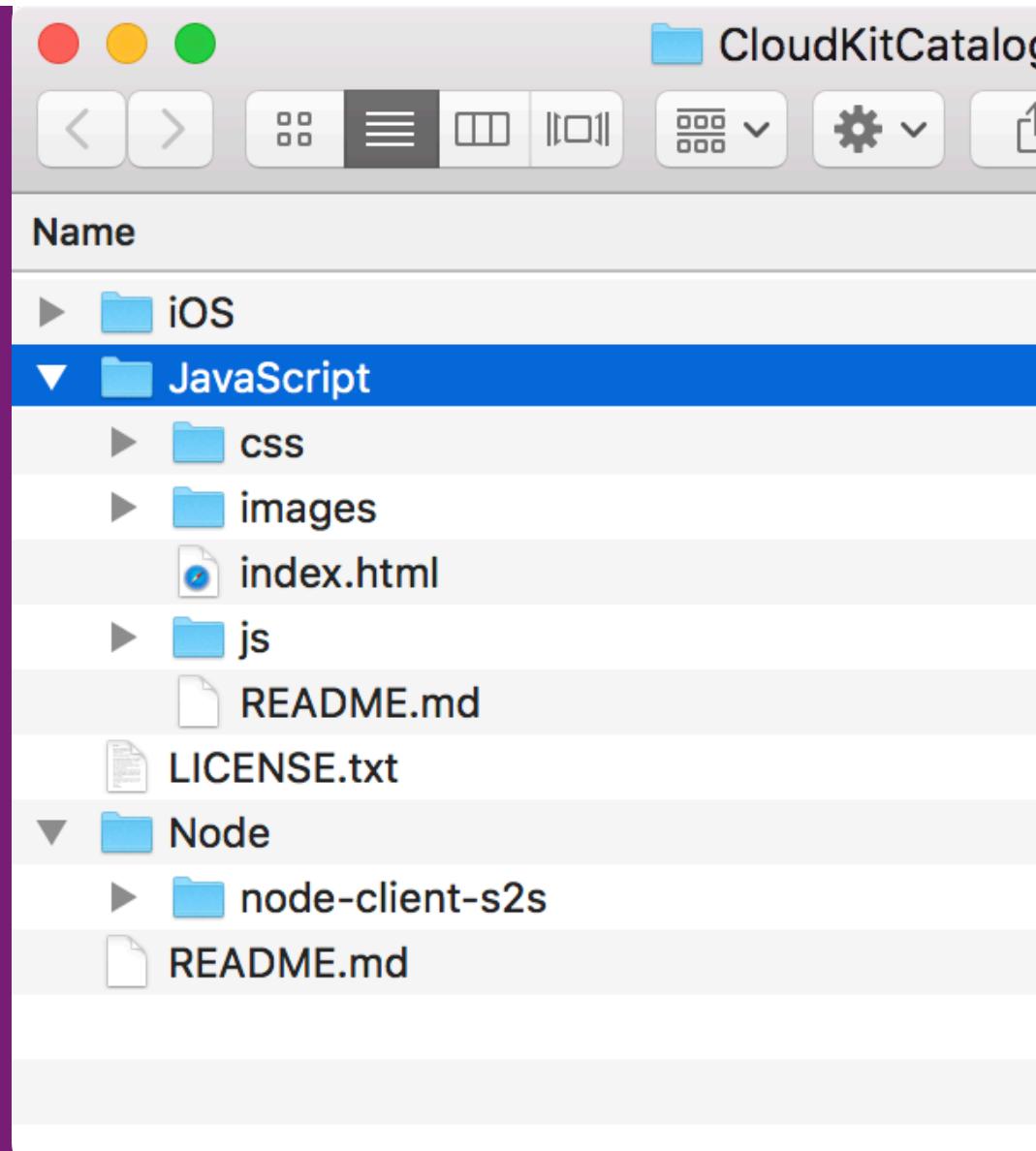
  var CloudKit = require('./cloudkit');
  var containerConfig = require('./config');

  // A utility function for printing results to the console.
  var println = function(key,value) {
    console.log("--> " + key + ":");
    console.log(value);
    console.log();
  };

  //CloudKit configuration
  CloudKit.configure({
    containerName: 'CloudKit-Sample',
    serverURL: 'https://cloudkit-test.firebaseio.com'
  });
})()
```

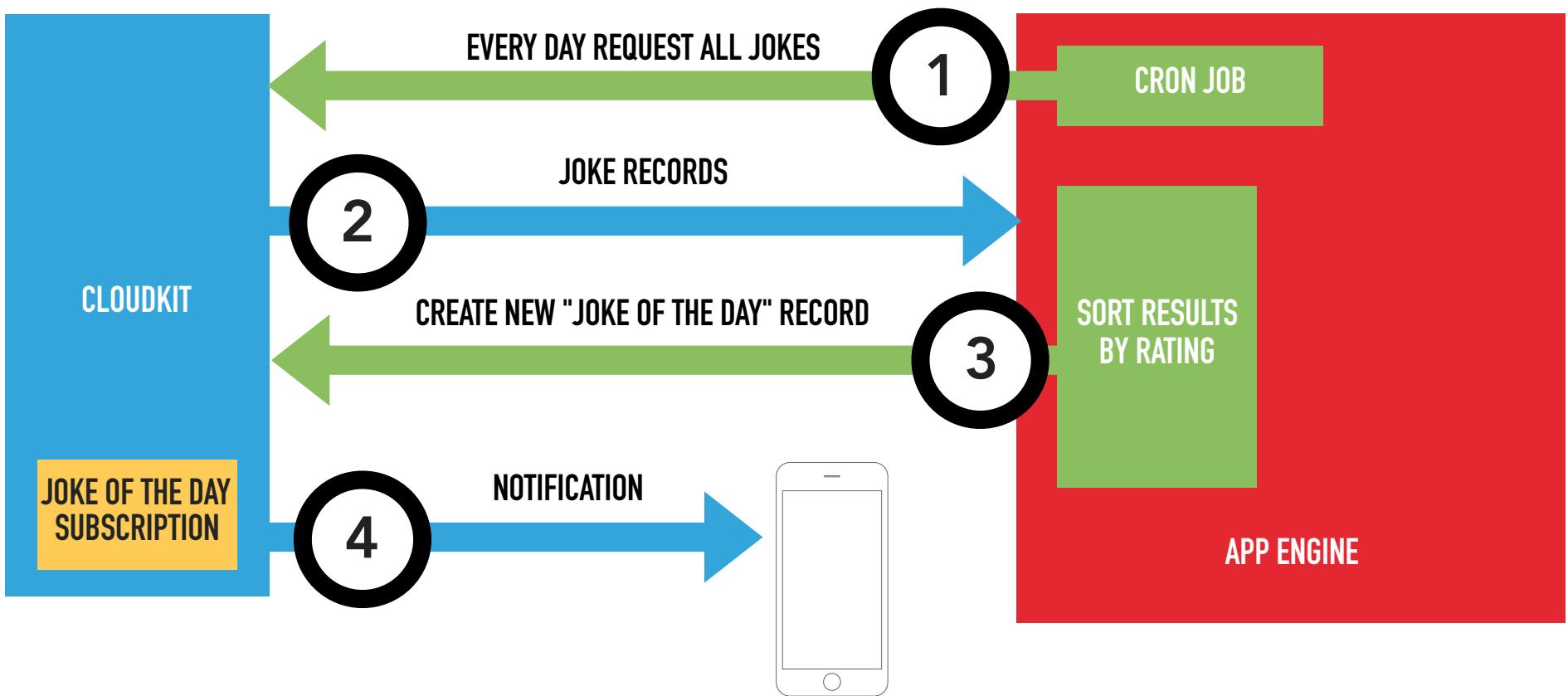
## SERVER-TO-SERVER KEY

- Python so that we can use App Engine



# CLOUDKIT HELPER

## SERVER-TO-SERVER KEY



# SERVER-TO-SERVER KEY

- Python so that we can use App Engine

```
#  
#  
# Based off the following  
# - CloudKitCatalog (c) 2016, Apple  
# - PyCloudKit. Created on 09.02.2016 (c) 2015 Andreas Schulz  
# - requests-cloudkit Copyright 2016 Lionheart Software LLC  
  
from __future__ import print_function  
import ecdsa  
import base64  
import hashlib  
import datetime  
import sys  
import json  
  
from urllib2 import HTTPPasswordMgrWithDefaultRealm, HTTPBasicAuthHandler, Request, build_opener  
from urllib import urlencode  
  
KEY_ID = 'ffdb30c23addd822e851e15cdd30ea819251d2fbb9bc59c4803f006dcc900191'  
CONTAINER = 'iCloud.cloud.uchicago.CloudyWithAChanceOfErrors'  
  
def cloudkit_request(cloudkit_resource_url, data):  
    """Uses HTTP GET or POST to interact with CloudKit. If data is empty, Uses  
    GET, else, POSTs the data.  
    """  
  
    # Get ISO 8601 date, cut milliseconds.  
    date = datetime.datetime.utcnow().isoformat()[:-7] + 'Z'  
  
    # Load JSON request from config.  
    _hash = hashlib.sha256(data.encode('utf-8')).digest()  
    body = base64.b64encode(_hash).decode('utf-8')  
  
    # Construct URL to CloudKit container.  
    web_service_url = '/database/1/' + CONTAINER + cloudkit_resource_url  
  
    # Load API key from config.  
    key_id = KEY_ID  
  
    # Read out certificate file corresponding to API key.  
    with open('eckey.pem', 'r') as pem_file:  
        signing_key = ecdsa.SigningKey.from_pem(pem_file.read())  
  
    # Construct payload.
```

IOS

IOS

## Record Types ▾

System Types

Users

Custom Types

joke

joke\_of\_the\_day



New Type



Delete Type

Field Name	Field Type	Indexes
<b>System Fields</b>		
recordName	Reference	None
createdBy	Reference	None
createdAt	Date/Time	None
modifiedAt	Reference	None
changedBy	Date/Time	None
changeNotes	String	None
joke	Reference	None

SET UP A SUBSCRIPTION FOR  
THE DAILY "JOKE OF THE DAY"

⊕ Add Field

L Edit Indexes

# IOS

- Setup subscription for joke of the day
- Remember subscriptions are tied to individual users

```
func registerJokeOfTheDaySubscriptions() {  
    // Unique identifier for the subscription  
    let uuid: UUID = UUID()  
    let identifier = "\(uuid)-joke-of-the-day"  
  
    // 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_of_the_day",  
                                            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")  
        }  
    })  
}
```

# IOS

```
// 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 = CKSubscription(recordType: "joke_of_the_day",
                                   predicate: NSPredicate(value: true),
                                   subscriptionID: identifier,
                                   options: [
                                       CKQuerySubscription.Options.firesOnRecordCreation
                                   ])
```

- Could send the joke (or make it generic)

# IOS

```
// Create the notification that will
let notificationInfo = CKSubscription.NotificationInfo()
notificationInfo.alertBody =
notificationInfo.shouldBadge = true
notificationInfo.shouldSendContentAvailable = true
notificationInfo.desiredKeys = ["joke"]
```

```
}, AnyHashable("ck"): {
    ce = 2;
    cid = "iCloud.mobi.uchicago.twenty-nineteen-cloudkit";
    ckuserid = "_c364d6b1fb322f17cd2346d1b82667a2";
    nid = "257010ef-4903-4116-873a-bfc3b619137";
    qry = {
        af = {
            joke = "3CCB6936-49BD-4CDF-8FAB-63F362C5D1B8";
        };
        "joke_of_the_day" = {
            zid = "_defaultZone";
            zoid = "_defaultOwner";
        };
    };
}
```

KEYS TO SEND WITH RECORD TYPE

```
(recordType: "joke_of_the_day",
predicate: NSPredicate(value: true),
subscriptionID: identifier,
options: [
    CKQuerySubscription.Options.firesOnRecordCreation]
```

KEYS TO SEND WITH RECORD TYPE

- Could send the joke (or make it generic)

# APP ENGINE

## APP ENGINE

- App Engine microservice to run cron job to process the data
  - Request all jokes
  - Process to find top rated joke
  - Add a new record to CloudKit representing joke of the day (trigger subscription)

```
# app.yaml
#
# Configuration for Google App Engine
# See https://cloud.google.com/appengine/docs/python/config/appref

runtime: python27
api_version: 1
threadsafe: true

handlers:
- url: /*
  script: main.app
```

## APP ENGINE

- We don't need to store anything on App Engine to accomplish this functionality

```
# app.yaml
#
# Configuration for Google App Engine
# See https://cloud.google.com/appengine/docs/python/config/appref

runtime: python27
api_version: 1
threadsafe: true

handlers:
- url: /*
  script: main.app
```

# APP ENGINE

- Adapt our cloudkit\_helper for app engine
  - Replace print()
  - ...

```
from __future__ import print_function
import ecdsa
import base64
import hashlib
import datetime
import sys
import json

from urllib2 import HTTPPasswordMgrWithDefaultRealm, HTTPBasicAuthHandler, Request, build_opener
from urllib import urlencode

KEY_ID = 'ffdb30c23add822e851e15cdd30ea819251d2fbb9bc59c4803f006dcc900191'
CONTAINER = 'iCloud.cloud.uchicago.CloudyWithAChanceOfErros'

def cloudkit_request(cloudkit_resource_url, data):
    """Uses HTTP GET or POST to interact with CloudKit. If data is empty, Uses
    GET, else, POSTs the data.
    """

    # Get ISO 8601 date, cut milliseconds.
    date = datetime.datetime.utcnow().isoformat()[:-7] + 'Z'

    # Load JSON request from config.
    _hash = hashlib.sha256(data.encode('utf-8')).digest()
    body = base64.b64encode(_hash).decode('utf-8')

    # Construct URL to CloudKit container.
    web_service_url = '/database/1/' + CONTAINER + cloudkit_resource_url

    # Load API key from config.
    key_id = KEY_ID
```

ecdsa is not a standard module on app engine

# APP ENGINE

```
cd gae_dir
```

```
mkdir lib
```

```
# pip install -t lib/ <library_name>
pip install -t lib/ ecdsa
```



INSTALL THE AN EXTERNAL LIBRARY  
IN YOUR PROJECT

# APP ENGINE

- Installed in full in the same directory
- Allows you to use it as pure python code

```
— app.yaml
— appengine_config.py
— cloudkit_helper.py
— cron.yaml
— eckey.pem
— joke_of_the_day.py
— lib
|   — ecdsa
|   |   — __init__.py
|   |   — __init__.pyc
|   |   — _version.py
|   |   — _version.pyc
|   |   — curves.py
|   |   — curves.pyc
|   |   — der.py
|   |   — der.pyc
|   |   — ecdsa.py
|   |   — ecdsa.pyc
|   |   — ellipticcurve.py
|   |   — ellipticcurve.pyc
|   |   — keys.py
|   |   — keys.pyc
|   |   — numbertheory.py
|   |   — numbertheory.pyc
|   |   — rfc6979.py
|   |   — rfc6979.pyc
|   |   — six.py
|   |   — six.pyc
|   |   — test_pyecdsa.py
|   |   — test_pyecdsa.pyc
|   |   — util.py
|   |   — util.pyc
|   — ecdsa-0.13.dist-info
|       — DESCRIPTION.rst
|       — INSTALLER
|       — METADATA
|       — RECORD
|       — WHEEL
|       — metadata.json
|       — top_level.txt
— main.py
```

# APP ENGINE

- Add a new file:

appengine\_config.py

```
# appengine_config.py
#
# from google.appengine.ext import vendor
#
# Add any libraries install in the "lib" folder.
vendor.add('lib')
```

# APP ENGINE

- main.py
- Handlers
  - Processing
  - Posting record

```
import webapp2

import sys
import json

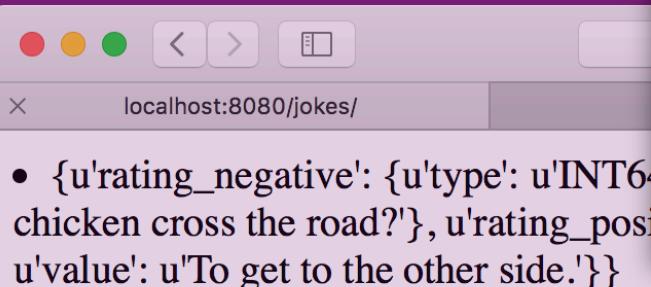
import cloudkit_helper as ck
from joke_of_the_day import JokeOfTheDay

class MainPage(webapp2.RequestHandler):
    def get(self):
        self.response.headers['Content-Type'] = 'text/html'
        self.response.write("home")

class ProcessJokes(webapp2.RequestHandler):
    def get(self):
        self.response.headers['Content-Type'] = 'text/html'
        jokes = ck.query_records('joke')
        for joke in jokes:
            self.response.write("<li>%s</li>" % joke["fields"])

app = webapp2.WSGIApplication([
    ('/', MainPage),
    ('/jokes/', ProcessJokes),
    ('/tasks/jokeoftheday/', JokeOfTheDay),
], debug=True)
```

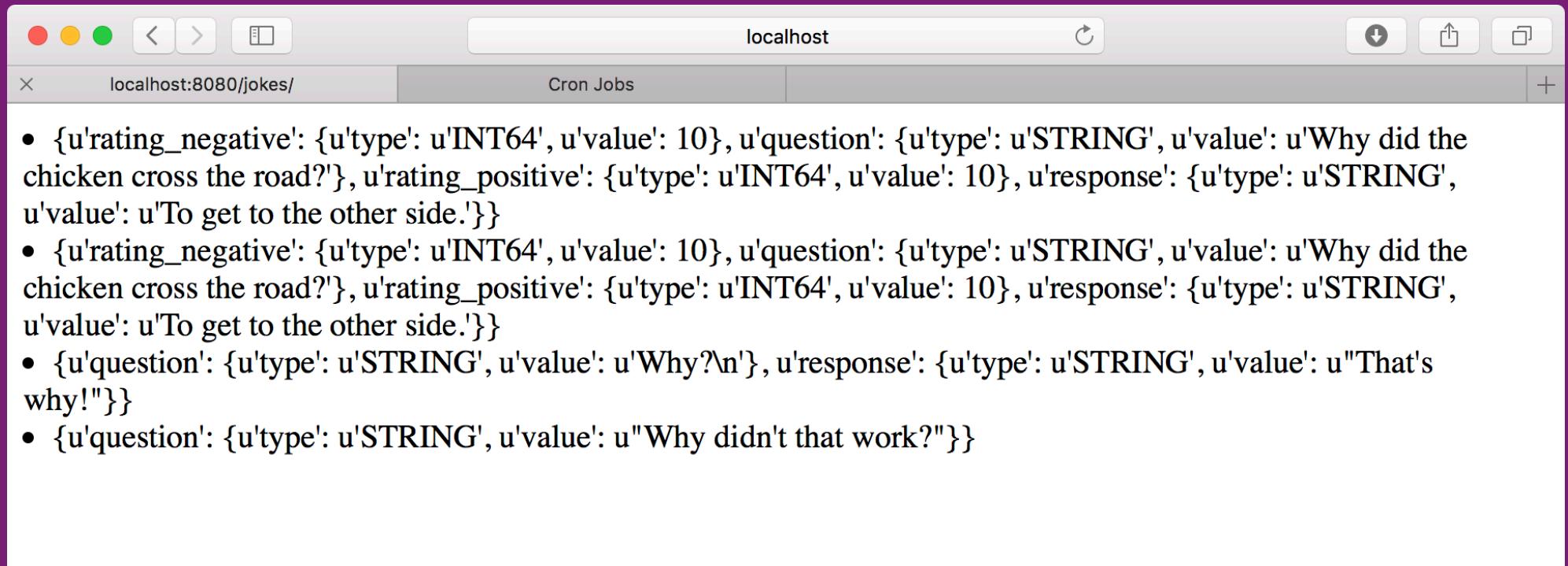
# APP ENGINE



```
class ProcessJokes(webapp2.RequestHandler):
    def get(self):
        self.response.headers['Content-Type'] = 'text/html'
        jokes = ck.query_records('joke')
        for joke in jokes:
            self.response.write("<li>%s</li>" % joke["fields"])
```

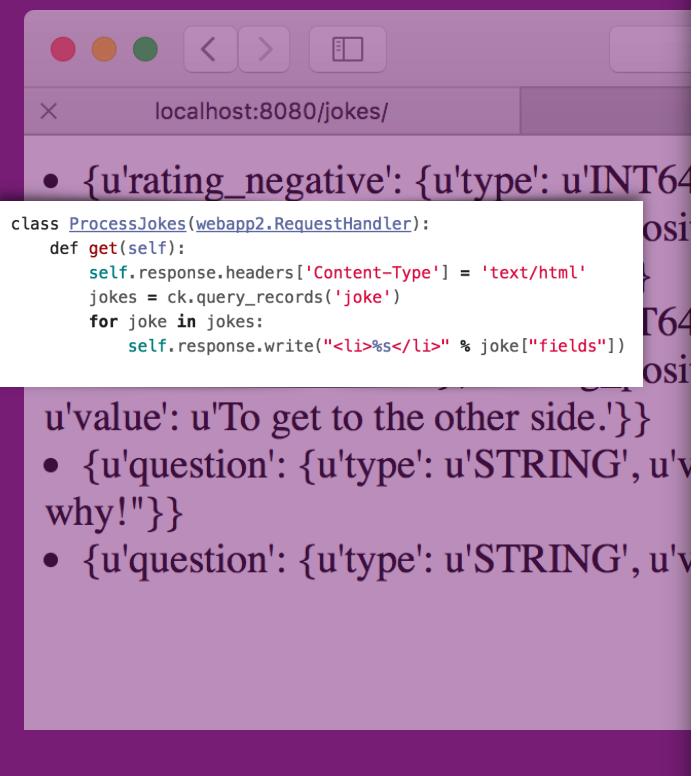
- <http://localhost:8080/jokes/>

# APP ENGINE



- <http://localhost:8080/jokes/>

# APP ENGINE



```
def query_records(record_type):
    """Queries CloudKit for all records of type record_type."""
    json_query = {
        'query': {
            'recordType': record_type
        }
    }

    records = []
    while True:
        result_query_authors = cloudkit_request(
            '/development/public/records/query',
            json.dumps(json_query))
        result_query_authors = json.loads(result_query_authors['content'])

        records += result_query_authors['records']

        if 'continuationMarker' in result_query_authors.keys():
            json_query['continuationMarker'] = \
                result_query_authors['continuationMarker']
        else:
            break
```

REQUEST WITH  
CURSORS

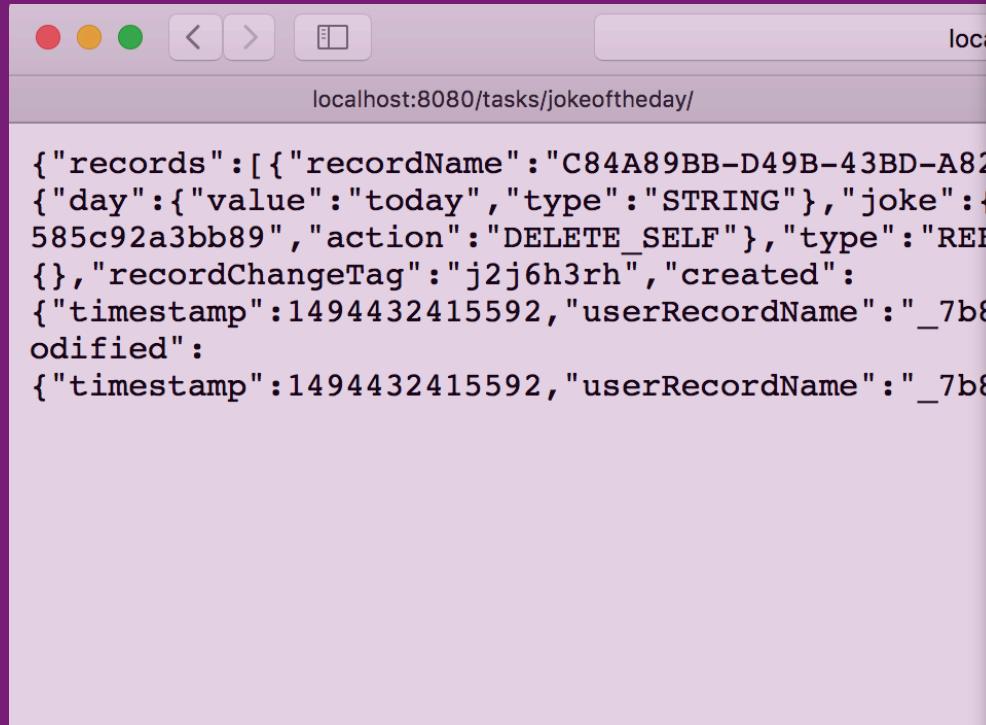
## APP ENGINE

```
def get(self):
    self.response.headers['Content-Type'] = 'text/html'
    jokes = ck.query_records('joke')
    for joke in jokes:
        self.response.write("<li>%s</li>" % joke["fields"])

app = webapp2.WSGIApplication([
    ('/', MainPage),
    ('/jokes/', ProcessJokes),
    ('/tasks/jokeoftheday/', JokeOfTheDay),
], debug=True)
```

- <http://localhost:8080/tasks/jokeoftheday/>

# APP ENGINE



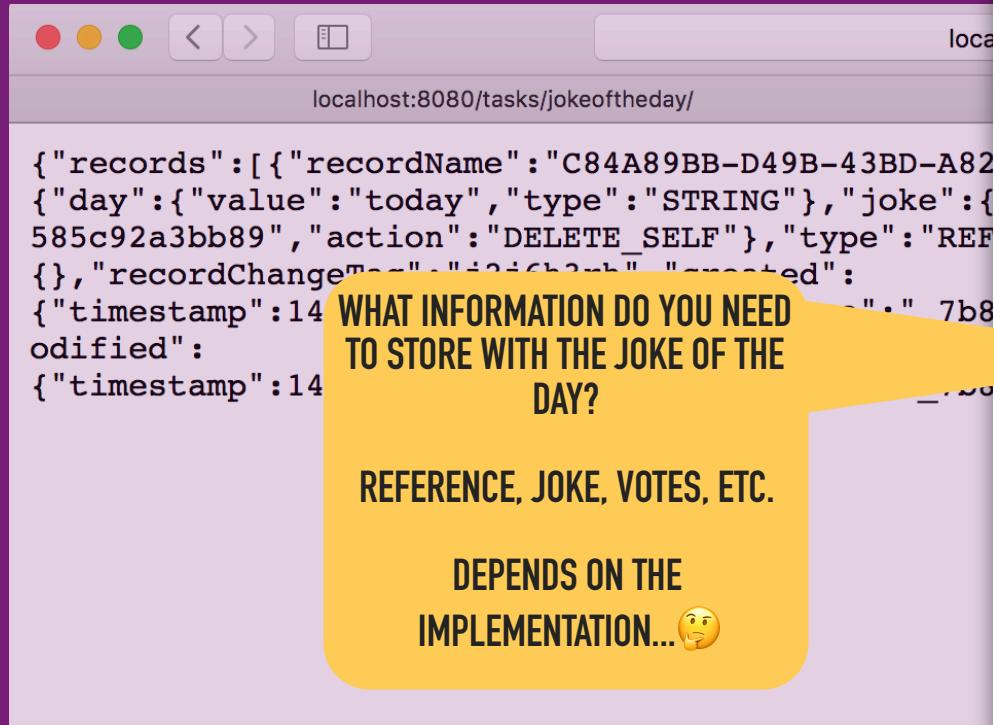
A screenshot of a web browser window. The address bar shows the URL `localhost:8080/tasks/jokeoftheday/`. The page content displays a JSON object with the following structure:

```
{"records": [{"recordName": "C84A89BB-D49B-43BD-A825-8D82b7d32ef4", "day": {"value": "today", "type": "STRING"}, "joke": {"recordName": "585c92a3bb89", "action": "DELETE_SELF"}, "type": "REFERENCE", "recordChangeTag": "j2j6h3rh", "created": {"timestamp": 1494432415592}, "userRecordName": "_7b8odified": {"timestamp": 1494432415592, "userRecordName": "_7b8odified"}]}
```

- <http://localhost:8080/tasks/jokeoftheday/>

```
class JokeOfTheDay(webapp2.RequestHandler):  
    #  
    # Create a new joke of the day record and post it to iCloud  
    # Note: We are hard coding the reference here, you should get  
    # if from the processing of the daily joke ratings  
    def get(self):  
  
        new_joke_of_the_day_data = {  
            'operations': [{  
                'operationType': 'create',  
                'record': {  
                    'recordType': 'Daily',  
                    'fields': {  
                        'day': {'value': 'today'},  
                        'joke': {  
                            'value': {  
                                'recordName': '60d82b7d-32ef-4e91-9c12-585c92a3bb89',  
                                'zoneID': {  
                                    'zoneName': '_defaultZone'  
                                },  
                                'action': 'DELETE_SELF'  
                            }  
                        }  
                    }  
                }  
            }  
        }  
  
        #print('Posting operation to create quote...')  
        result_modify_jokes = ck.cloudkit_request(  
            '/development/public/records/modify',  
            json.dumps(new_joke_of_the_day_data))  
  
        self.response.headers['Content-Type'] = 'text/json'  
        self.response.write(result_modify_jokes['content'])
```

# APP ENGINE

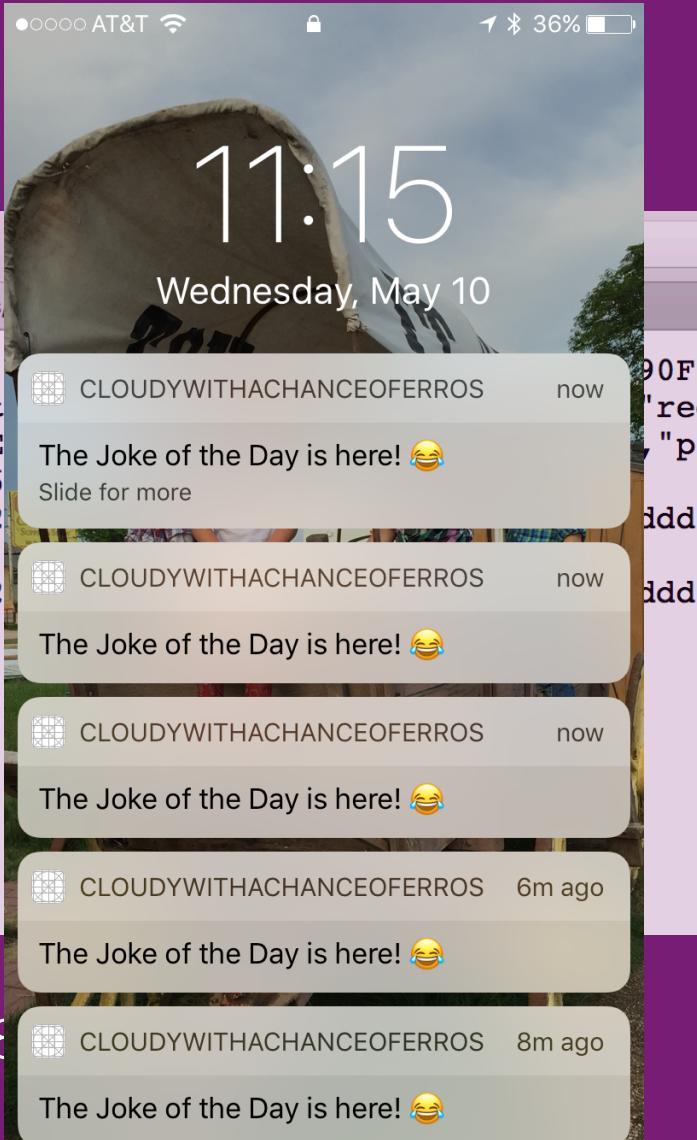


- <http://localhost:8080/tasks/jokeoftheday/>

```
class JokeOfTheDay(webapp2.RequestHandler):  
    #  
    # Create a new joke of the day record and post it to iCloud  
    # Note: We are hard coding the reference here, you should get  
    # if from the processing of the daily joke ratings  
    def get(self):  
  
        new_joke_of_the_day_data = {  
            'operations': [{  
                'operationType': 'create',  
                'record': {  
                    'recordType': 'Daily',  
                    'fields': {  
                        'day': {'value': 'today'},  
                        'joke': {  
                            'value': {  
                                'recordName': '60d82b7d-32ef-4e91-9c12-585c92a3bb89',  
                                'zoneID': {  
                                    'zoneName': '_defaultZone'  
                                },  
                                'action': 'DELETE_SELF'  
                            }  
                        }  
                    }  
                }  
            }  
        }  
  
        #print('Posting operation to create quote...')  
        result_modify_jokes = ck.cloudkit_request(  
            '/development/public/records/modify',  
            json.dumps(new_joke_of_the_day_data))  
  
        self.response.headers['Content-Type'] = 'text/json'  
        self.response.write(result_modify_jokes['content'])
```

# APP ENGINE

```
localhost:8080/tasks  
  
{"records": [{"recordName": "CloudyWithAChanceOfFerros", "day": {"value": "today", "text": "585c92a3bb89"}, "action": "DELETE", "recordChangeTag": "j2j6"}, {"timestamp": 1494432415592}, {"modified": {"timestamp": 1494432415592}}]
```



- <http://localhost:8080>

```
def cloudkit_request(cloudkit_resource_url, data):  
    """Uses HTTP GET or POST to interact with CloudKit. If data is empty, Uses  
    GET, else, POSTs the data.  
    """  
  
    # Get ISO 8601 date, cut milliseconds.  
    date = datetime.datetime.utcnow().isoformat()[:-7] + 'Z'  
  
    # Load JSON request from config.  
    _hash = hashlib.sha256(data.encode('utf-8')).digest()  
    body = base64.b64encode(_hash).decode('utf-8')  
  
    # Construct URL to CloudKit container.  
    web_service_url = '/database/1/' + CONTAINER + cloudkit_resource_url  
  
    # Load API key from config.  
    key_id = KEY_ID  
  
    # Read out certificate file corresponding to API key.  
    with open('eckey.pem', 'r') as pem_file:  
        signing_key = ecdsa.SigningKey.from_pem(pem_file.read())  
  
    # Construct payload.  
    unsigned_data = ':'.join([date, body, web_service_url]).encode('utf-8')  
  
    # Sign payload via certificate.  
    signed_data = signing_key.sign(unsigned_data,  
                                    hashfunc=hashlib.sha256,  
                                    sigencode=ecdsa.util.sigencode_der)  
  
    signature = base64.b64encode(signed_data).decode('utf-8')  
  
    headers = {  
        'X-Apple-CloudKit-Request-KeyID': key_id,  
        'X-Apple-CloudKit-Request-ISO8601Date': date,  
        'X-Apple-CloudKit-Request-SignatureV1': signature  
    }  
  
    if data:  
        req_type = 'POST'  
    else:  
        req_type = 'GET'  
  
    result = curl('https://api.apple-cloudkit.com' + web_service_url,  
                 req_type=req_type,  
                 data=data,  
                 headers=headers)  
  
    return result
```

# APP ENGINE

```
# cron.yaml
#
# To test in development server goto http://localhost:8000/cron
#
cron:
- description: daily joke summary and notification
  url: /tasks/jokeoftheday/
  schedule: every day 08:00
  retry_parameters:
    min_backoff_seconds: 60
    max_doublings: 5
```

- Add a new file: cron.yaml

# APP ENGINE

## CRON JOB

# Google App Engine

Development SDK 1.9.50

dev~None

Instances

Datastore Viewer

Datastore Indexes

Datastore Stats

Interactive Console

Memcache Viewer

Blobstore Viewer

Task Queues

Cron Jobs

### Cron Jobs

Request to /tasks/jokeoftheday/ succeeded!

Cron Job	Schedule
/tasks/jokeoftheday/	<b>every day 08:00</b>
daily joke summary and notification	In production, this would run at these times: 2017-05-11 08:00:00Z 16:20:41.791990 from now
	2017-05-12 08:00:00Z 1 day, 16:20:41.791990 from now
	2017-05-13 08:00:00Z 2 days, 16:20:41.791990 from now

Run now

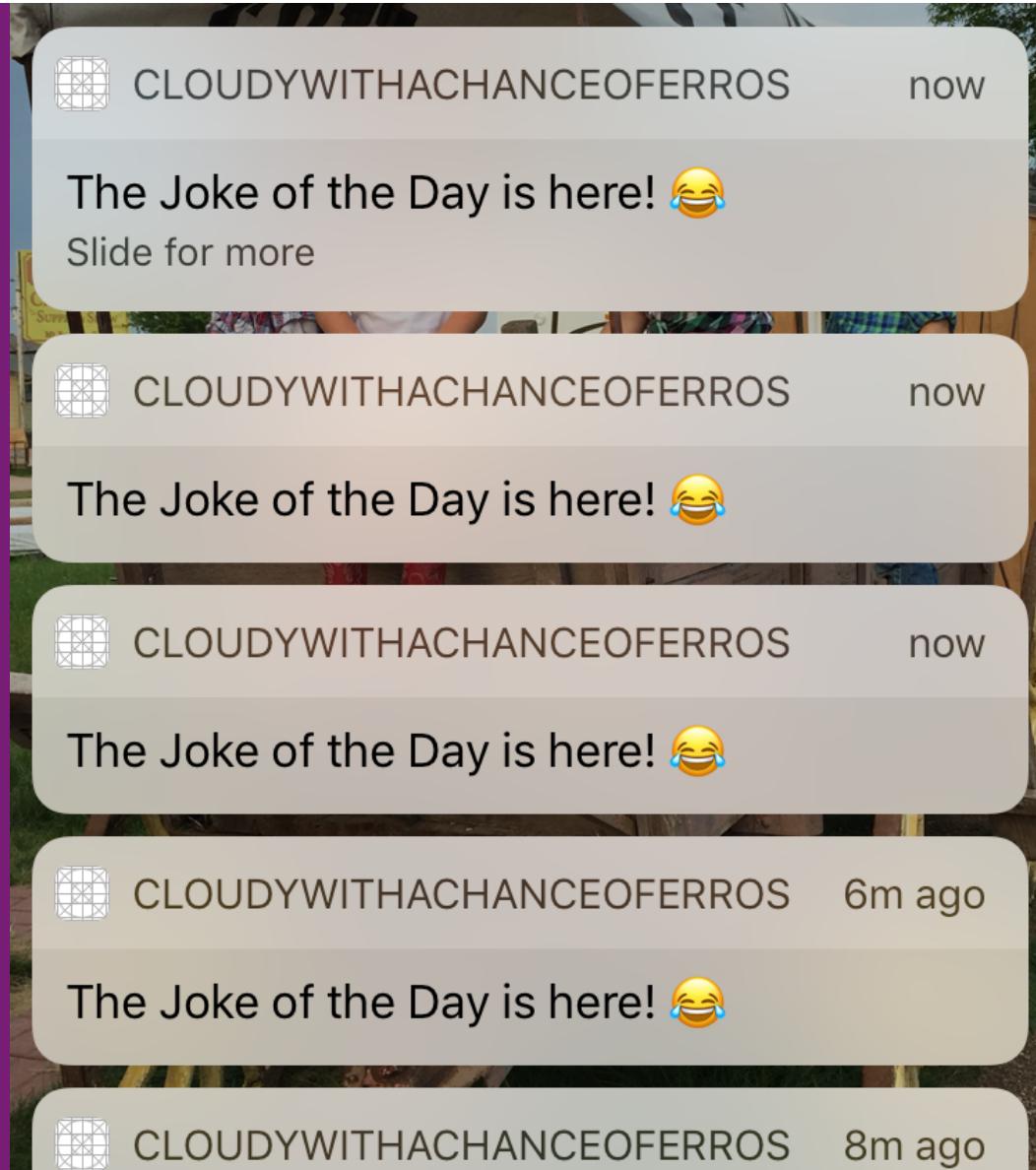
BREAK TIME



# CUSTOM NOTIFICATIONS FROM SUBSCRIPTIONS

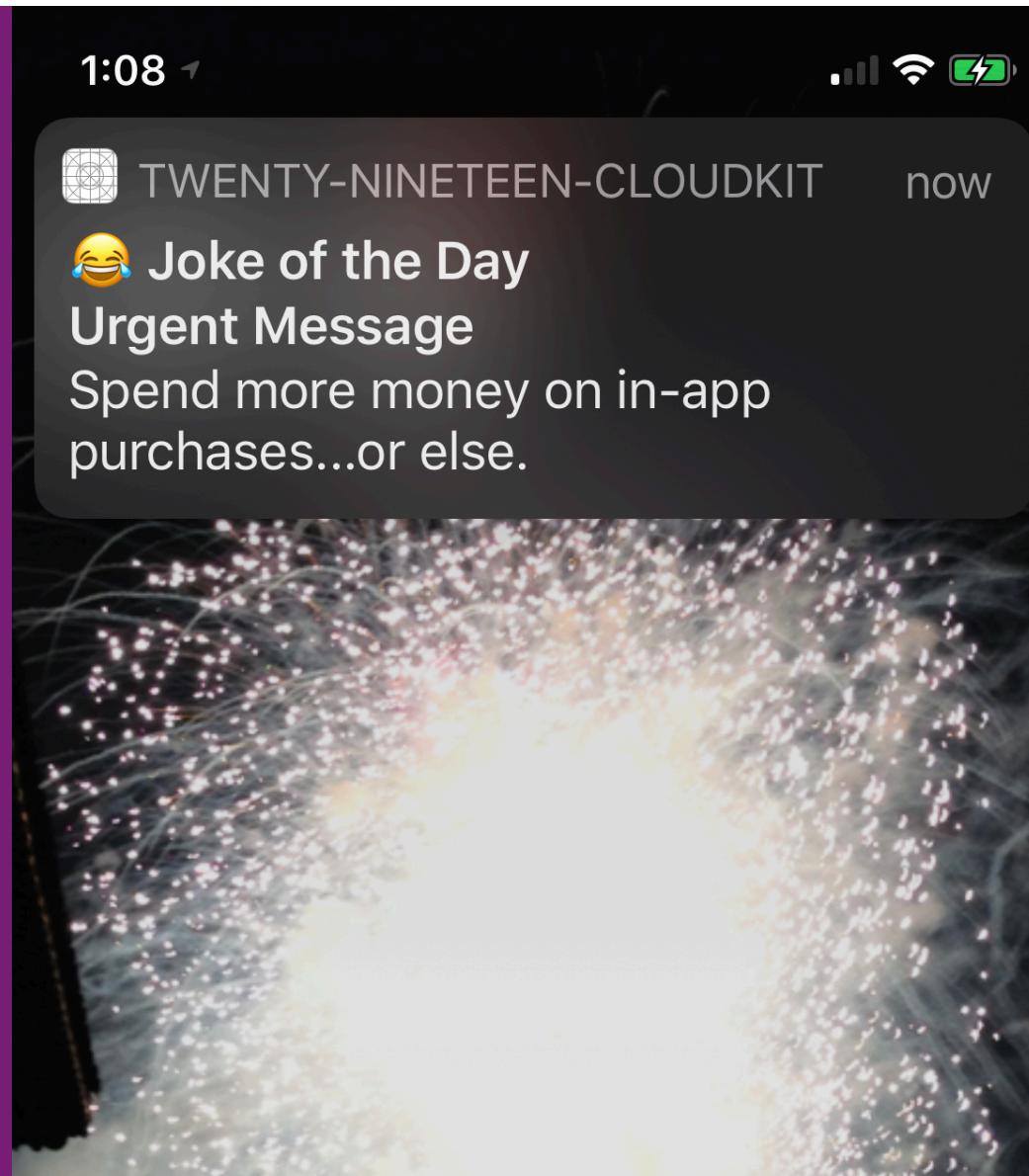
# CUSTOM NOTIFICATIONS

- Generic notifications from subscriptions
- This may not be the great user experience we want



## CUSTOM NOTIFICATIONS

- Generic notifications from subscriptions
- This may not be the great user experience we want



# CUSTOM NOTIFICATIONS



# CUSTOM NOTIFICATIONS

## Record Types ▾

System Types

Users

Custom Types

alert

joke

joke\_of\_the\_day



New Type



Delete Type

Field Name	Field Type	Indexes
<strong>System Fields</strong>		
recordName	Reference	None
createdBy	Reference	None
createdAt	Date/Time	None
modifiedBy	Reference	None
modifiedAt	Date/Time	None
changeTag	String	None
<strong>Custom Fields</strong>		
message	String	None
<a href="#">+ Add Field</a>		<a href="#">Edit Indexes</a>

# CUSTOM NOTIFICATIONS

```
func registerSilentAlertSubscription() {
    let uuid: UUID = UUID()
    let identifier = "\(uuid)-alert"

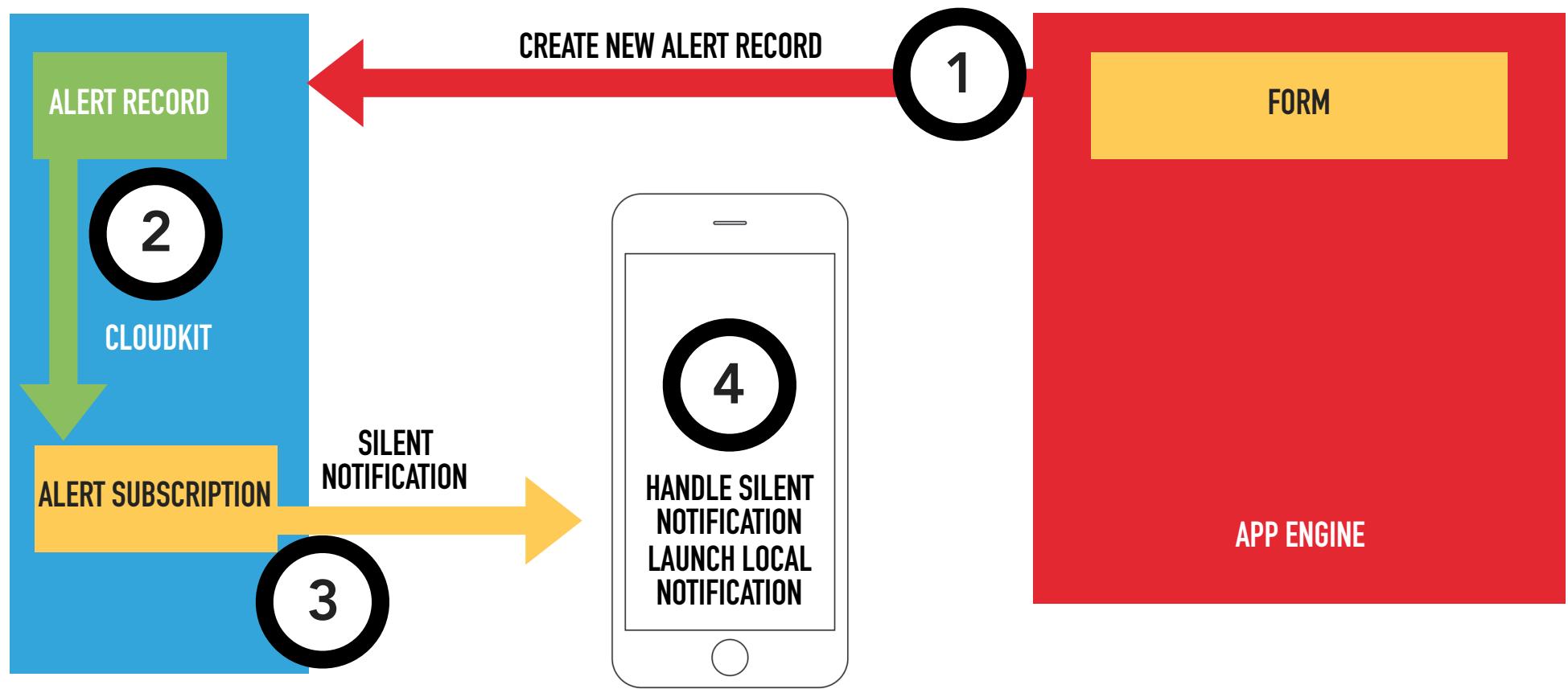
    // Create the notification that will be delivered
    let notificationInfo = CKSubscription.NotificationInfo()
    notificationInfo.shouldSendContentAvailable = true
    notificationInfo.desiredKeys = ["message"]

    // Create the subscription
    let subscription = CKQuerySubscription(recordType: "alert",
                                             predicate: NSPredicate(value: true),
                                             subscriptionID: identifier,
                                             options: [CKQuerySubscription.Options.firesOnRecordCreation])

    subscription.notificationInfo = notificationInfo
    CKContainer.default().publicCloudDatabase.save(subscription,
                                                    completionHandler: {returnRecord, error in
        if let err = error {
            print("ALERT: subscription failed \(err.localizedDescription)")
        } else {
            print("ALERT: subscription set up")
        }
    })
}
```

MESSAGE IS PASSED ALL THE WAY THROUGH TO THE IOS APP

# CUSTOM NOTIFICATIONS



# CLOUDKIT HELPER

# CUSTOM NOTIFICATIONS

- alert.py
- Prompt user for a message and then adds a new record to the public database
  - Triggers the alert

```
import sys
import json

import cloudkit_helper as ck

def add_alert(message):
    """Create a new alert with a custom message"""

    data = {
        'operations': [
            {
                'operationType': 'create',
                'record': {
                    'recordType': 'alert',
                    'fields': {
                        'message': {
                            'value': message,
                        }
                    }
                }
            }
        ]
    }

    print('Posting operation to create quote...')
    result = ck.cloudkit_request(
        '/development/public/records/modify', json.dumps(data))
    print(result['content'])

def main():
    message = input("What would you like to say to your users?\n")
    add_alert(message)

if __name__ == '__main__':
    main()
```

# CUSTOM NOTIFICATIONS

```
def add_alert(message):
    """Create a new alert with a custom message"""

    data = {
        'operations': [
            {
                'operationType': 'create',
                'record': {
                    'recordType': 'alert',
                    'fields': {
                        'message': {
                            'value': message,
                        }
                    }
                }
            }
        ]
    }

    print('Posting operation to create quote...')
    result = ck.cloudkit_request(
        '/development/public/records/modify', json.dumps(data))
    print(result['content'])
```

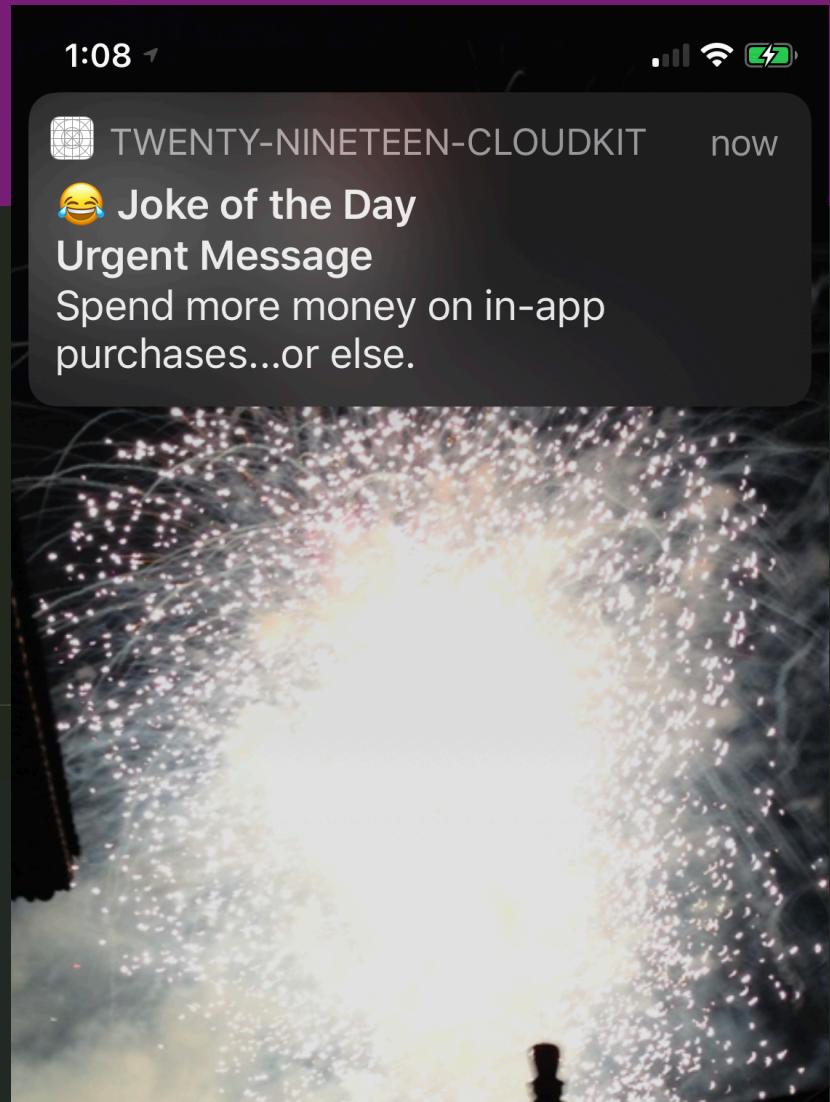
# CUSTOM NOTIFICATIONS

```
12     record : [
13       {
14         'recordType': 'alert',
15         'fields': {
16           'message': {
17             'value': message,
18           }
19         }
20       }
21     ]
22
23   print('Posting operation to create quote...')
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

3-2

What would you like to say to your users?  
Buy more in app purchases...or else



# CUSTOM NOTIFICATIONS

What would you like to say to your users?  
☞ You can even send emojis!!!!!!

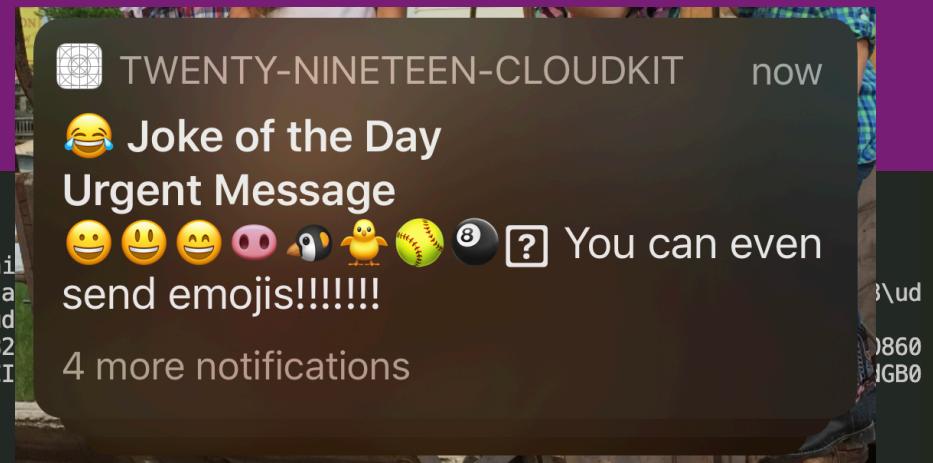
Posting operation to create quote...

URL: <https://api.apple-cloudkit.com/database/1/iCloud.mobi.uchicago.twenty-nineteen-cloudkit/records>  
Data: {"operations": [{"operationType": "create", "record": {"recordType": "quote", "fields": {"message": {"value": "\ud83d\udc04\ud83d\udc3d\ud83d\udc27\ud83d\udc25\ud83e\udd4e\ud83c\udfb1\ud83e\udc80"}, "type": "STRING"}}, "pluginFields": {}, "recordChangeTag": "k2w91wdk", "created": {"timestamp": 1573586969406, "userRecordName": "\_8a76cb86f6390ecf95f548796a270cc8", "userRecordID": "8a76cb86f6390ecf95f548796a270cc8"}]}]

Params: None

```
{  
  "records": [ {  
    "recordName": "0B361BB6-5E9D-4D10-8531-F607A96C480B",  
    "recordType": "alert",  
    "fields": {  
      "message": {  
        "value": "\ud83d\udc04\ud83d\udc3d\ud83d\udc27\ud83d\udc25\ud83e\udd4e\ud83c\udfb1\ud83e\udc80"},  
        "type": "STRING"  
      }  
    },  
    "pluginFields": {},  
    "recordChangeTag": "k2w91wdk",  
    "created": {  
      "timestamp": 1573586969406,  
      "userRecordName": "_8a76cb86f6390ecf95f548796a270cc8",  
      "userRecordID": "8a76cb86f6390ecf95f548796a270cc8"  
    }  
  }]
```

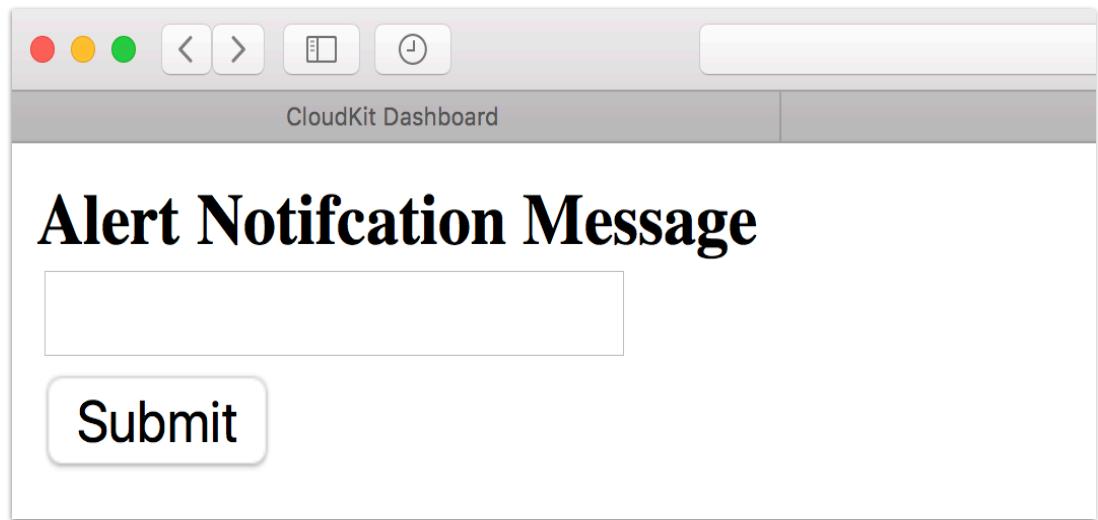
EMOJIS SURVIVE



# CUSTOM NOTIFICATIONS

- Create an amazing form that takes some text for the custom message

```
def get(self):
    self.response.headers['Content-Type'] = 'text/html'
    html = """
        <form action="/alerts/" method="post">
        <b>Alert Notifcation Message</b><br>
        <input type="text" name="message" value=""><br>
        <input type="submit" value="Submit">
        </form>
    """
    self.response.write(html)
```



IOS APP

# CUSTOM NOTIFICATIONS

- Handle the remote notification
  - Read the APNS data
  - Extract the message
  - Create local notification

```
/// Called for push notifications
/// In this case, we are getting the changed record from cloudkit and then creating a new notification
func application(_ application: UIApplication,
                  didReceiveRemoteNotification userInfo: [AnyHashable : Any],
                  fetchCompletionHandler completionHandler: @escaping (UIBackgroundFetchResult) -> Void) {
    // Get the APS notification data
    print(userInfo)
    let aps = userInfo["aps"] as! [String: AnyObject]

    // Do Something with the content available data
    let contentAvailable = aps["content-available"] as! Int
    if contentAvailable == 1 {
        print("APS: \(aps)")
        let cloudKitInfo = userInfo["ck"] as! [String: AnyObject]
        let recordId = cloudKitInfo["qry"]?["rid"] as! String
        let field = cloudKitInfo["qry"]?["af"] as! [String: AnyObject]

        // A message indicates that we have received a silent notification
        if let message = field["message"] as? String {
            local_notification_from_silent(message: message, recordId: recordId)
        }
        completionHandler(.newData)
    } else {
        completionHandler(.noData)
    }
}
```

# CUSTOM NOTIFICATIONS

SILENT NOTIFICATION CONTENT

```
[AnyHashable("aps"): {  
    "content-available" = 1;  
}, AnyHashable("ck"): {  
    ce = 2;  
    cid = "iCloud.cloud.uchicago.Cloudy";  
    fo = 1;  
    nchanceOfErrors;  
    nid = "2b661f39-5617-4a1c-a880-cd9eae7a";  
    qry = {  
        af = {  
            message = "Where are my emojis?";  
        };  
        dbs = 2;  
        fo = 1;  
        rid = "A0915CF5-E4E7-4C76-856E-48CE01ADCACE";  
        sid = "14725FE6-9C8F-4BDC-96FF-3815E5FE1E3B-alert";  
        zid = "_defaultZone";  
        zoid = "_defaultOwner";  
    };  
};  
]
```

FIELD I REQUESTED

RECORD THAT MATCHED  
THE QUERY

# CUSTOM NOTIFICATIONS

```
func local_notification_from_silent(message: String, recordId: String) {  
    print("Creating local notification")  
  
    // Create notification content  
    let content = UNMutableNotificationContent()  
    content.title = "😂 Joke of the Day"  
    content.subtitle = "Urgent Message"  
    content.body = message  
    content.sound = UNNotificationSound.default  
  
    // Set up trigger  
    let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 1, repeats: false)  
  
    // Create the notification request  
    let center = UNUserNotificationCenter.current()  
    let identifier = recordId  
    let request = UNNotificationRequest(identifier: identifier,  
                                         content: content, trigger: trigger)  
    center.add(request, completionHandler: { (error) in  
        if let error = error {  
            print("Something went wrong: \(error)")  
        }  
    })
```

CREATE A  
NOTIFICATION

# CUSTOM NOTIFICATIONS

```
func local_notification_from_silent(message: String, recordId: String) {  
    print("Creating local notification")  
  
    // Create notification content  
    let content = UNMutableNotificationContent()  
    content.title = "😂 Joke of the Day"  
    content.subtitle = "Urgent Message"  
    content.body = message  
    content.sound = UNNotificationSound.default  
  
    // Set up trigger  
    let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 1, repeats: false)  
  
    // Create the notification request  
    let center = UNUserNotificationCenter.current()  
    let identifier = recordId  
    let request = UNNotificationRequest(identifier: identifier,  
                                         content: content, trigger: trigger)  
    center.add(request, completionHandler: { (error) in  
        if let error = error {  
            print("Something went wrong: \(error)")  
        }  
    })
```

CREATE A TRIGGER

TIME, 1 SEC

# CUSTOM NOTIFICATIONS

```
func local_notification_from_silent(message: String, recordId: String) {  
    print("Creating local notification")  
  
    // Create notification content  
    let content = UNMutableNotificationContent()  
    content.title = "😂 Joke of the Day"  
    content.subtitle = "Urgent Message"  
    content.body = message  
    content.sound = UNNotificationSound.default  
  
    // Set up trigger  
    let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 1, repeats: false)  
  
    // Create the notification request  
    let center = UNUserNotificationCenter.current()  
    let identifier = recordId  
    let request = UNNotificationRequest(identifier: identifier,  
                                         content: content, trigger: trigger)  
    center.add(request, completionHandler: { (error) in  
        if let error = error {  
            print("Something went wrong: \(error)")  
        }  
    })
```

ADD TO  
NOTIFICATION  
CENTER

# CORE DATA

# CORE DATA

Technique	Pros	Cons
Archiving	Allows ordered relationships (arrays, not sets). Easy to deal with versioning.	Reads all the objects in (no faulting). No incremental updates.
Web Service	Makes it easy to share data with other devices and applications.	Requires a server and a connection to the internet.
Core Data	Lazy fetches by default. Incremental updates.	Versioning is awkward (but can certainly be done using an <b>NSModelMapping</b> ). No real ordered relationships (at the time this is being written).

- Options for data storage and persistence

# CORE DATA

- A framework that provides object-relational mapping
  - Converts objective-c objects into data stored on disk
- Creates a visual mapping between database and objects
- An object-oriented API that conforms to MVC paradigm
- API for constructing detailed queries
- Access the data using **properties** on the objects

## Introduction to Core Data Programming Guide

The Core Data framework provides generalized and automated solutions to common tasks associated with object graph management, including persistence.

### Who Should Read This Document

You should read this document to gain an understanding of the Core Data framework. *You must be experienced in Cocoa development, including the Objective-C language and memory management.*

**Important:** Although this document provides a thorough treatment of the fundamentals of the Core Data framework, simply reading from start to finish is not a good strategy for learning how to use the technology effectively. You should typically augment your understanding by following the related tutorials provided in the Reference section. For a description of the recommended learning path, see *Core Data Starting Point*.

### Organization of This Document

The following articles explain the problems the Core Data Framework addresses, the solutions it provides, its functionality, and common tasks you might perform:

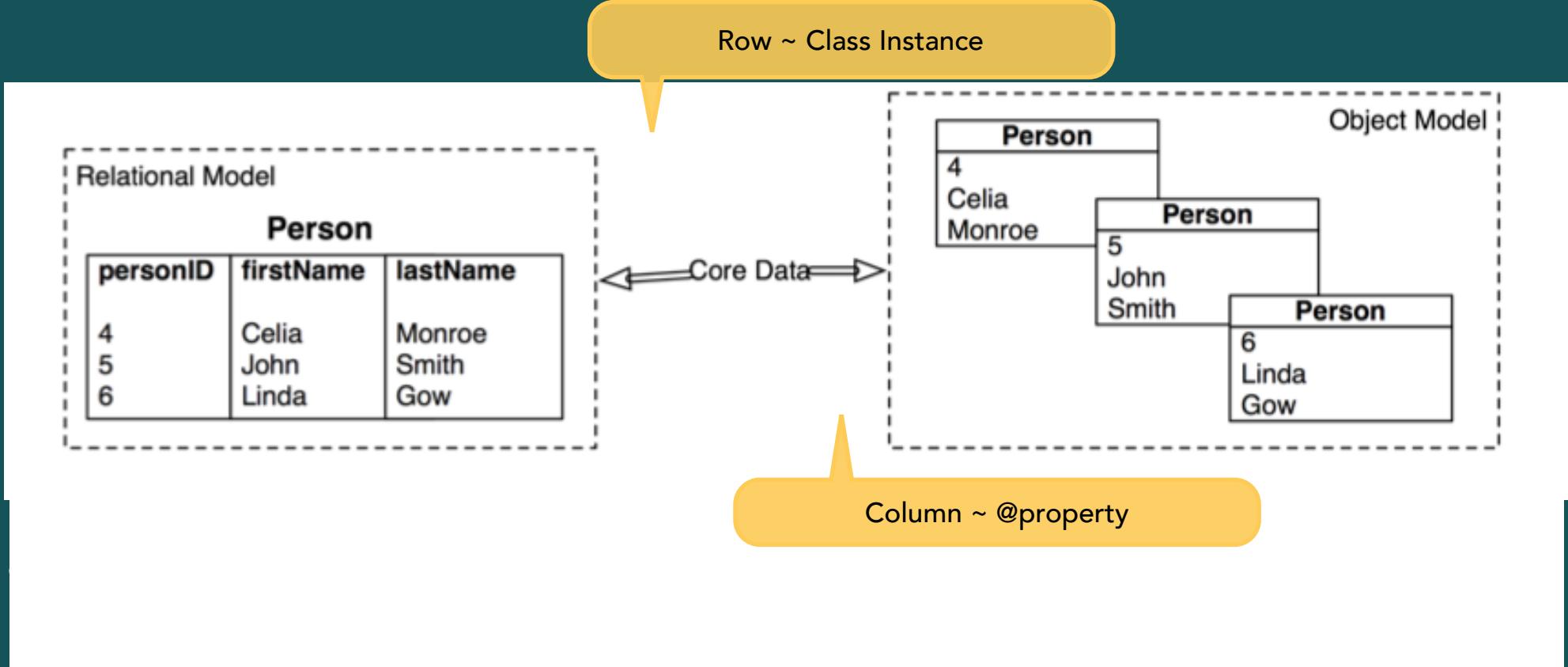
- [Technology Overview](#) describes what Core Data is and why you might choose to use it.
- [Core Data Basics](#) describes the fundamental architecture of the technology.
- [Managed Object Models](#) describes the features of a managed object model.
- [Using a Managed Object Model](#) describes how you use a managed object model in your application.
- [Managed Objects](#) describes the features of a managed object, the `NSManagedObject` class, and how to work with them.

# CORE DATA

- Core Data is an object graph management and persistence framework
- Core Data is not a database!



# CORE DATA



- Row =~ an instance of the class
- Core Data model file is the description of every entity and their attributes in your application

# CORE DATA

## WHAT YOU SHOULD KNOW ABOUT CORE DATA

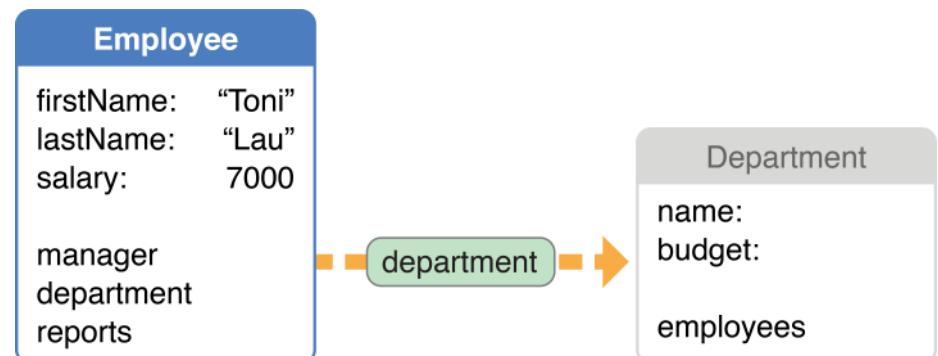
- Object graph store
  - Model layer represent state of your app
- Persistence
  - Read/write state of model objects to disk
- Querying
  - SQL-like parameter based queries
- Versioning
  - Lightweight migration



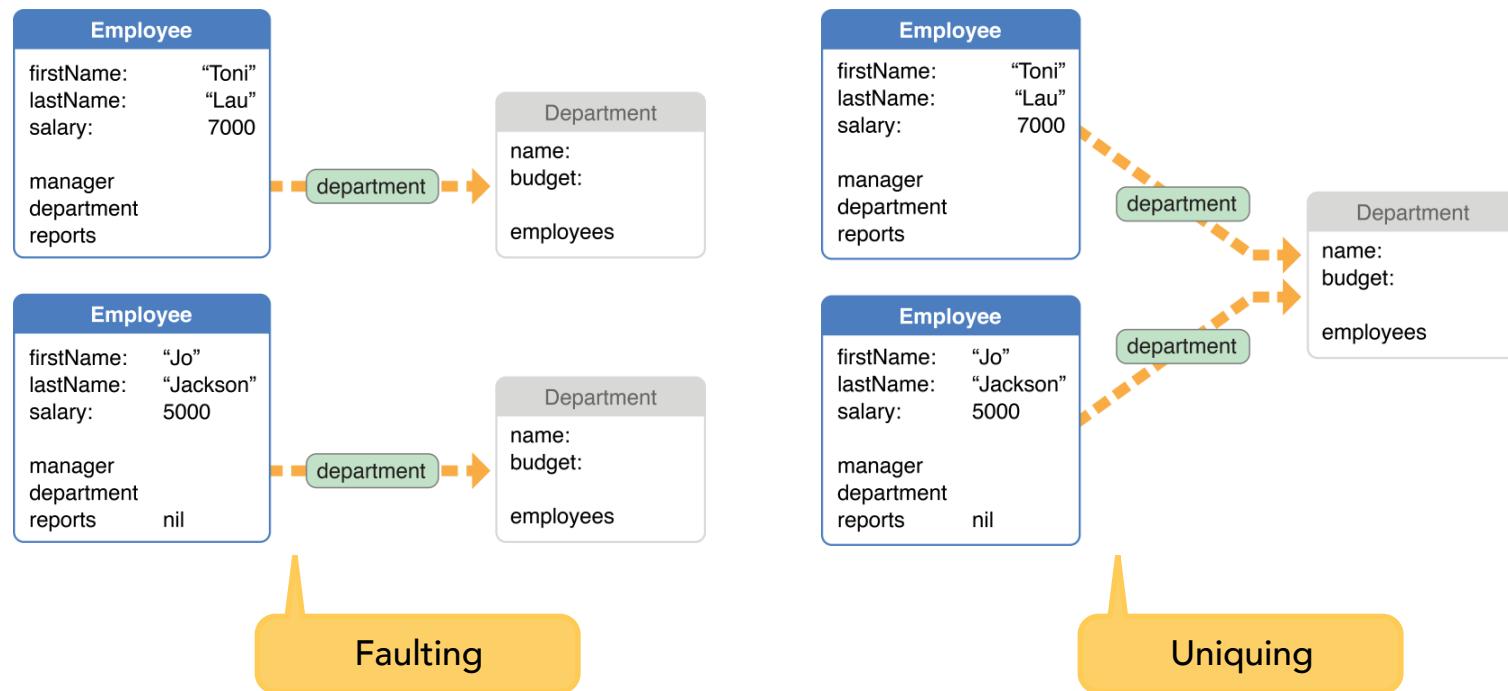
# CORE DATA

## WHAT YOU PROBABLY DO NOT KNOW

- Change tracking
- Optimize data work flows while in memory
- Faulting
  - Reduce application's memory usage
- Uniquing
  - Ensures that, in a given managed object context, you never have more than one managed object to represent a given record

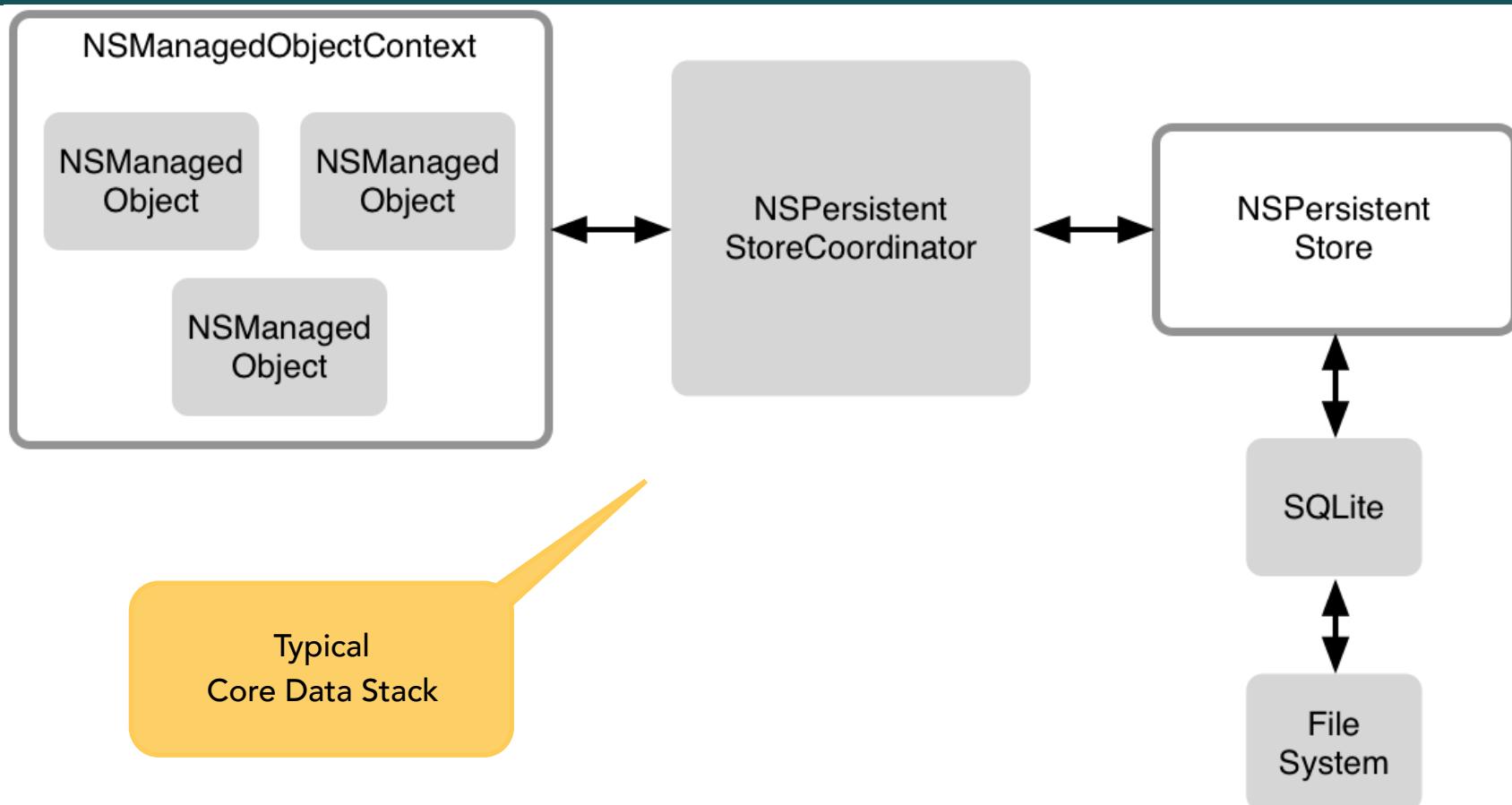


# CORE DATA

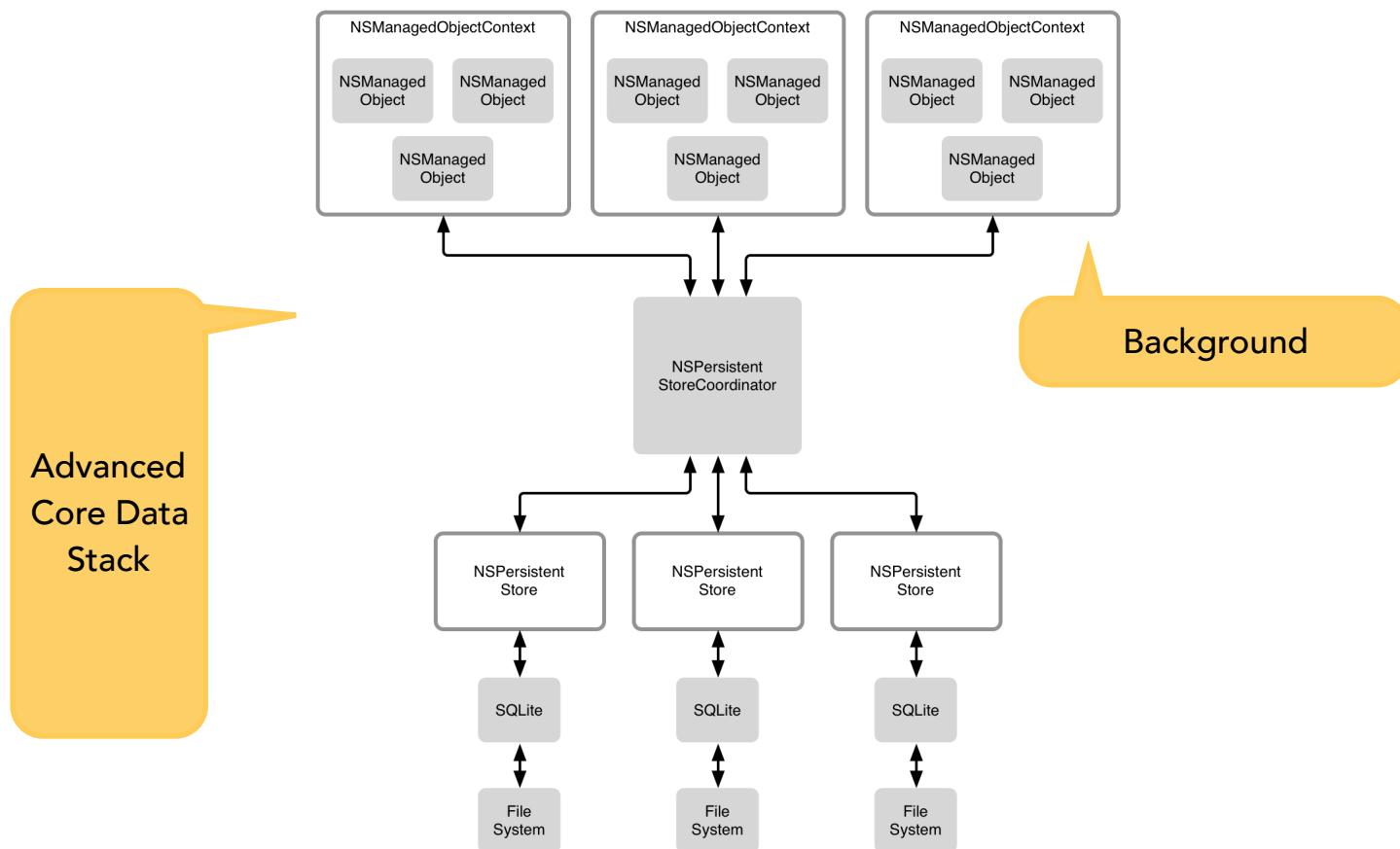


Faulting is a mechanism Core Data employs to reduce your application's memory usage  
Uniquing ensures that you never have more than one managed object to represent a given record

# CORE DATA

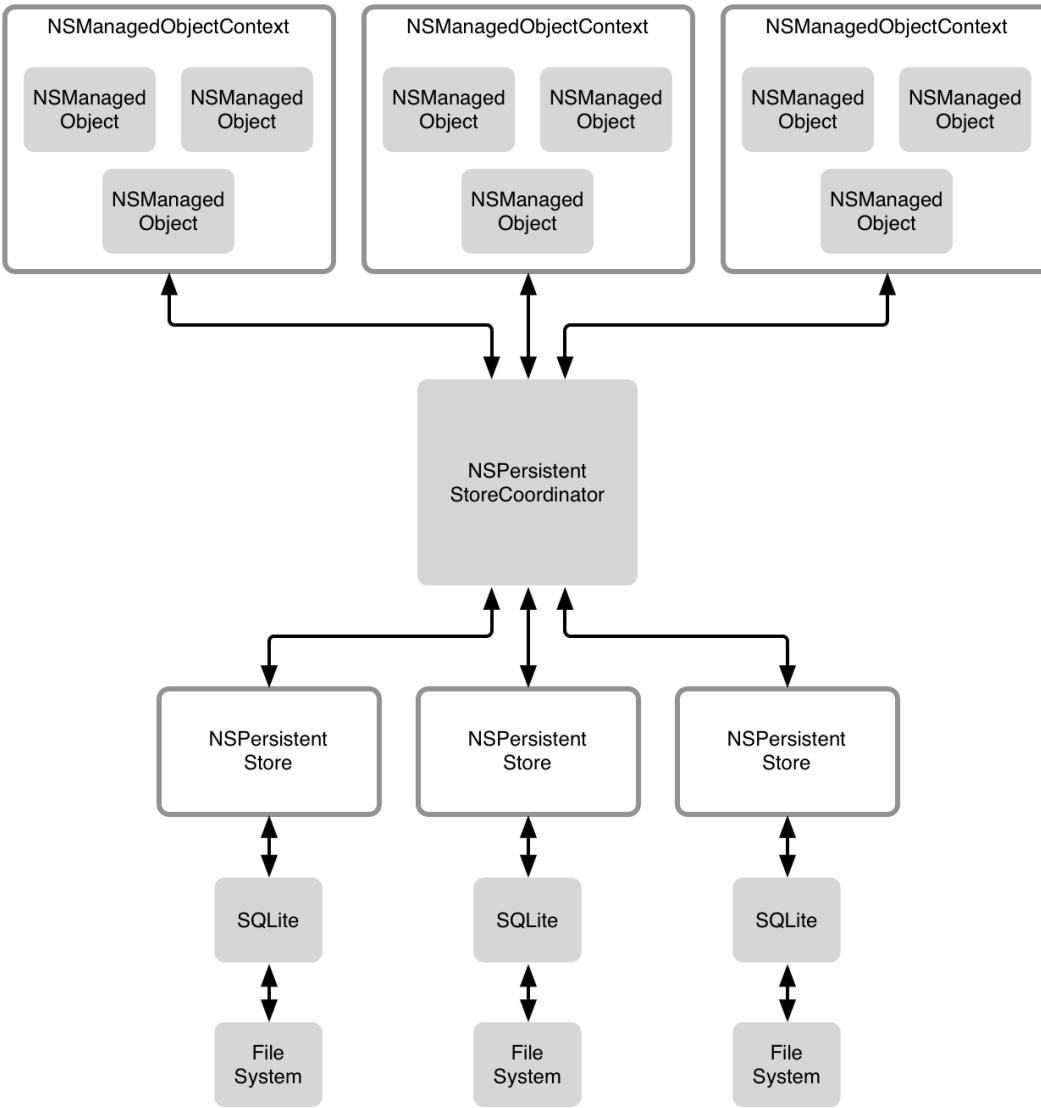


# CORE DATA



# CORE DATA

- Multiple Managed Object Context (MOC) setups
  - MOC only know about their objects
  - Objects only know about their MOC
- Persistent store doesn't care about what its storing



# CORE DATA

## APPLICATION FLOW

- Setup Core Data Stack
  - Object model
  - Persistent store
  - MOC

```
lazy var persistentContainer: NSPersistentContainer = {
    /*
        The persistent container for the application. This implementation
        creates and returns a container, having loaded the store for the
        application to it. This property is optional since there are legitimate
        error conditions that could cause the creation of the store to fail.
    */
    let container = NSPersistentContainer(name: "_017_CoreDataBasics")
    container.loadPersistentStores(completionHandler: { (storeDescription, error) in
        if let error = error as NSError? {
            // Replace this implementation with code to handle the error app
            // fatalError() causes the application to generate a crash log a
            // in a shipping application, although it may be useful during
            /*
                Typical reasons for an error here include:
                * The parent directory does not exist, cannot be created, or di
                * The persistent store is not accessible, due to permissions or
                * The device is out of space.
                * The store could not be migrated to the current model version.
                Check the error message to determine what the actual problem wa
            */
            fatalError("Unresolved error \(error), \(error.userInfo)")
        }
    })
    return container
}()

// MARK: - Core Data Saving support

func saveContext () {
    let context = persistentContainer.viewContext
    if context.hasChanges {
        do {
            try context.save()
        } catch {
            // Replace this implementation with code to handle the error app
            // fatalError() causes the application to generate a crash log a
            // in a shipping application, although it may be useful during
            let nserror = error as NSError
            fatalError("Unresolved error \(nserror), \(nserror.userInfo)")
        }
    }
}
```

# CORE DATA

## APPLICATION FLOW

The screenshot shows the Xcode interface with the Core Data editor open. The left sidebar displays the project structure for "2017-CoreDataBasics". The main editor area is focused on the "Person" entity within the "ENTITIES" section of the xcdata model.

**Attributes:**

Attribute	Type
N age	Integer 16
S name	String
D timestamp	Date

**Relationships:**

Relationship	Destination	Inverse
(empty)	(empty)	(empty)

**Fetched Properties:**

Fetched Property	Predicate
(empty)	(empty)

# CORE DATA

## APPLICATION FLOW

- Create an instance of your entity and set the property values

```
var person = Person(context: context!)
person.name = "Homer"
person.age = 42
person.timestamp = Date() as NSDate

person = Person(context: context!)
person.name = "Marge2"
person.age = 41
person.timestamp = Date() as NSDate
```

# CORE DATA

## APPLICATION FLOW

- Changes are kept in memory until you explicitly call 'save'
- What happens?
  - MOC figures out what has changed
  - The managed object context then passes these changes on to the persistent store coordinator
  - The persistent store coordinator coordinates with the store (SQLite3 db) to write into the SQL database on disk

```
func saveContext () {
    if managedObjectContext.hasChanges {
        do {
            try managedObjectContext.save()
        } catch {
            // Replace this implementation with code to handle the error
            // appropriately.
            // abort() causes the application to generate a crash log and
            // terminate. You should not use this function in a shipping
            // application, although it may be useful during development.
            let nserror = error as NSError
            NSLog("Unresolved error \(nserror), \(nserror.userInfo)")
            abort()
        }
    }
}
```

# CORE DATA

## APPLICATION FLOW

- Make a fetch request
- Core Data brings back entity objects
- Fine grain control on the performance

```
// Fetch our person objects
let personFetchRequest: NSFetchRequest<Person> = Person.fetchRequest()

do {
    let fetchedEntities = try context?.fetch(personFetchRequest)

    for user in fetchedEntities! {
        print("> \(user.name!) #####")
    }
} catch {
    // Do something in response to error condition
}
```

# CORE DATA

## Why Should You Use Core Data?

There are a number of reasons why it may be appropriate for you to use Core Data. One of the simplest metrics is that, with Core Data, the amount of code you write to support the model layer of your application is typically 50% to 70% smaller as measured by lines of code. This is primarily due to the features listed above—the features Core Data provides are features you don't have to implement yourself. Moreover they're features you don't have to test yourself, and in particular you don't have to optimize yourself.

Core Data has a mature code base whose quality is maintained through unit tests, and is used daily by millions of customers in a wide variety of applications. The framework has been highly optimized over several releases. It takes advantage of information provided in the model and runtime features not typically employed in application-level code. Moreover, in addition to providing excellent security and error-handling, it offers best memory scalability of any competing solution. Put another way: you could spend a long time carefully crafting your own solution optimized for a particular problem domain, and not gain any performance advantage over what Core Data offers for free for any application.

In addition to the benefits of the framework itself, Core Data integrates well with the OS X tool chain. The model design tools allow you to create your schema graphically, quickly and easily. You can use templates in the Instruments application to measure Core Data's performance, and to debug various problems. On OS X desktop, Core Data also integrates with Xcode Interface Builder to allow you to create user interfaces from your model. These aspects help to further shorten integration application design, implementation, and debugging cycles.

Already done  
for you

It's a mature  
technology

# CORE DATA

## What Core Data Is Not

NOT A RDB

Having given an overview of what Core Data is and does, and why it may be useful, it is also useful to correct some common misperceptions and state what it is not.

- Core Data is not a relational database or a relational database management system (RDBMS).

Core Data provides an infrastructure for change management and for saving objects to and retrieving them from storage. It can use SQLite as one of its persistent store types. It is not, though, in and of itself a database. (To emphasize this point: you could for example use just an in-memory store in your application. You could use Core Data for change tracking and management, but never actually save any data in a file.)

- Core Data is not a silver bullet.

Core Data does not remove the need to write code. Although it is possible to create a sophisticated application solely using the Xcode data modeling tool and Interface Builder, for more real-world applications you will still have to write code.

- Core Data does not rely on Cocoa bindings.

Core Data integrates well with Cocoa bindings and leverages the same technologies—and used together they can significantly reduce the amount of code you have to write—but it is possible to use Core Data without bindings. You can readily create a Core Data application without a user interface (see *Core Data Utility Tutorial*).

"real-world"?

# CORE DATA

## DOWNFALLS

```
NSManagedObjectModel *model = <#Get a model#>;
NSFetchRequest *requestTemplate = [[NSFetchRequest alloc] init];
NSEntityDescription *publicationEntity =
    [[model entitiesByName] objectForKey:@"Publication"];
[requestTemplate setEntity:publicationEntity];

NSPredicate *predicateTemplate = [NSPredicate predicateWithFormat:
    @"(mainAuthor.firstName like[cd] $FIRST_NAME) AND \
    (mainAuthor.lastName like[cd] $LAST_NAME) AND \
    (publicationDate > $DATE)"];
[requestTemplate setPredicate:predicateTemplate];
[model setFetchRequestTemplate:requestTemplate
    forName:@"PublicationsForAuthorSinceDate"];
```

- Verbose (simple fetch)
- Versioning
- Historical misunderstanding

# CORE DATA STACK IN IOS10

# CORE DATA

## APPLICATION FLOW

- This was tremendously simplified in iOS10
- Most criticism of CoreData it probably is outdated

```
lazy var persistentContainer: NSPersistentContainer = {
    /*
        The persistent container for the application. This implementation
        creates and returns a container, having loaded the store for the
        application to it. This property is optional since there are legitimate
        error conditions that could cause the creation of the store to fail.
    */
    let container = NSPersistentContainer(name: "_017_CoreDataBasics")
    container.loadPersistentStores(completionHandler: { (storeDescription, error) in
        if let error = error as NSError? {
            // Replace this implementation with code to handle the error app
            // fatalError() causes the application to generate a crash log a
            // in a shipping application, although it may be useful during
            /*
                Typical reasons for an error here include:
                * The parent directory does not exist, cannot be created, or di
                * The persistent store is not accessible, due to permissions or
                * The device is out of space.
                * The store could not be migrated to the current model version.
                Check the error message to determine what the actual problem wa
            */
            fatalError("Unresolved error \(error), \(error.userInfo)")
        }
    })
    return container
}()

// MARK: - Core Data Saving support

func saveContext () {
    let context = persistentContainer.viewContext
    if context.hasChanges {
        do {
            try context.save()
        } catch {
            // Replace this implementation with code to handle the error app
            // fatalError() causes the application to generate a crash log a
            // in a shipping application, although it may be useful during
            let nserror = error as NSError
            fatalError("Unresolved error \(nserror), \(nserror.userInfo)")
        }
    }
}
```

# CORE DATA

## APPLICATION FLOW

- Setup Core Data Stack
  - Object model
  - Persistent store
  - MOC

Old way had to set up all parts of the stack manually

```
// MARK: - Core Data stack

lazy var applicationDocumentsDirectory: NSURL = {
    // The directory the application uses to store the Core Data store file. This code uses a directory named "mobi.uchicago.016_CoreDataStack"
    let urls = NSFileManager.defaultManager().URLsForDirectory(.DocumentDirectory, inDomains: .UserDomainMask)
    return urls.urls[urls.count-1]
}()

lazy var managedObjectModel: NSManagedObjectModel = {
    // The managed object model for the application. This property is not optional. It is a fatal error for the application not
    let modelURL = NSBundle.mainBundle().URLForResource("_016_CoreDataStack", withExtension: "momd")!
    return NSManagedObjectModel(contentsOfURL: modelURL)!
}()

lazy var persistentStoreCoordinator: NSPersistentStoreCoordinator = {
    // The persistent store coordinator for the application. This implementation creates and returns a coordinator, having added
    // Create the coordinator and store
    let coordinator = NSPersistentStoreCoordinator(managedObjectModel: self.managedObjectModel)
    let url = self.applicationDocumentsDirectory.URLByAppendingPathComponent("SingleViewCoreData.sqlite")
    var failureReason = "There was an error creating or loading the application's saved data."
    do {
        try coordinator.addPersistentStoreWithType(NSSQLiteStoreType, configuration: nil, URL: url, options: nil)
    } catch {
        // Report any error we got.
        var dict = [String: AnyObject]()
        dict[NSLocalizedDescriptionKey] = "Failed to initialize the application's saved data"
        dict[NSLocalizedFailureReasonErrorKey] = failureReason

        dict[NSErrorUnderlyingErrorKey] = error as NSError
        let wrappedError = NSError(domain: "YOUR_ERROR_DOMAIN", code: 9999, userInfo: dict)
        // Replace this with code to handle the error appropriately.
        // abort() causes the application to generate a crash log and terminate. You should not use this function in a shipping
        NSLog("Unresolved error \(wrappedError), \(wrappedError.userInfo)")
        abort()
    }
}

return coordinator
}()

lazy var managedObjectContext: NSManagedObjectContext = {
    // Returns the managed object context for the application (which is already bound to the persistent store coordinator for you).
    let coordinator = self.persistentStoreCoordinator
    var managedObjectContext = NSManagedObjectContext(concurrencyType: .MainQueueConcurrencyType)
    managedObjectContext.persistentStoreCoordinator = coordinator
    return managedObjectContext
}()
```

# CORE DATA

```
_managedObjectModel = [[NSManagedObjectModel alloc] initWithContentsOfURL:[self modelURL]];
_persistentStoreCoordinator = [[NSPersistentStoreCoordinator alloc]
                               initWithManagedObjectModel:self.managedObjectModel];

_managedObjectContext = [[NSManagedObjectContext alloc]
                        initWithConcurrencyType:NSMainQueueConcurrencyType];

self.managedObjectContext.persistentStoreCoordinator = self.persistentStoreCoordinator;
[self.managedObjectContext.persistentStoreCoordinator addPersistentStoreWithType:NSSQLiteStoreType
                      configuration:nil
                            URL:[self storeURL]
                          options:nil
                        error:&error];
```

- Application Flow: Setup Core Data Stack

# CORE DATA

```
|let entity = NSEntityDescription.entityForName("Standard", inManagedObjectContext: context)
self.init(entity: entity!, insertIntoManagedObjectContext: context)
self.identifier = identifier
self.category = data["category"] as? String
self.stateStandard = data["stateStandard"] as? String
self.subCategory = data["subCategory"] as? String
self.uuid = data["uuid"] as? String
```

- Create/Edit entities

# CORE DATA

## APPLICATION FLOW

```
guard self._fetchedResultsController == nil else {
    return self._fetchedResultsController!
}
let fetchRequest = NSFetchedResultsController(entityName: "Lesson")
let sectionSortDescriptor = NSSortDescriptor(key: "currentIndexSection", ascending: true)
let rowSortDescriptor = NSSortDescriptor(key: "currentIndexRow", ascending: true)
fetchRequest.sortDescriptors = [sectionSortDescriptor, rowSortDescriptor]
if let cp = currentPredicate {
    fetchRequest.predicate = cp
} else {
    fetchRequest.predicate = defaultPredicate
}

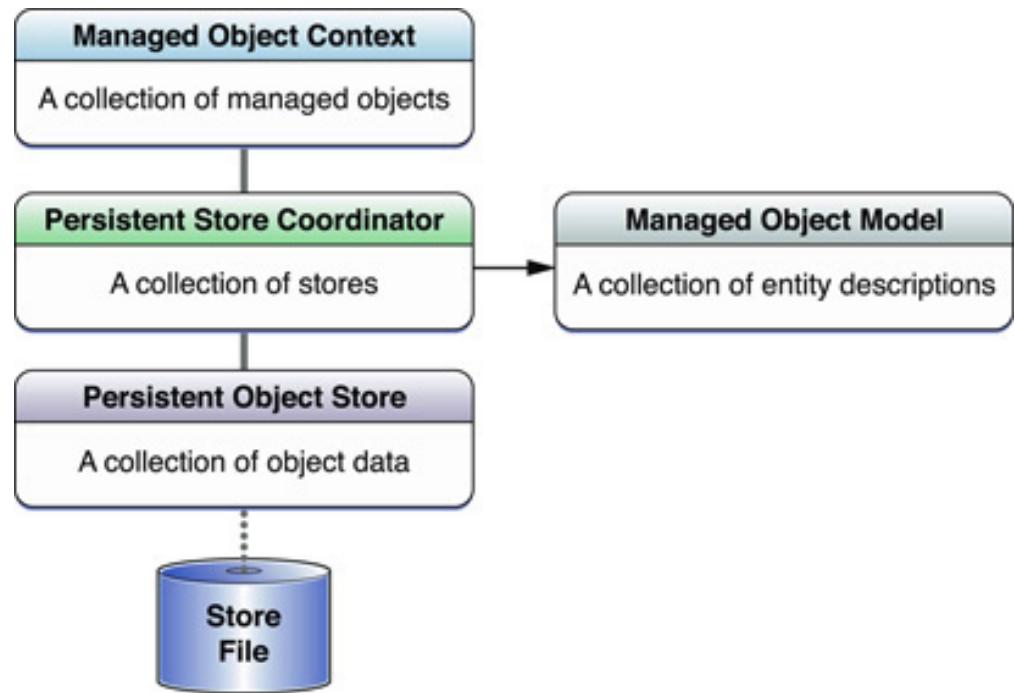
let frc = NSFetchedResultsController(fetchRequest: fetchRequest, managedObjectContext: self.context, sectionNameKeyPath: "currentIndexSection", cacheName: nil)
frc.delegate = self
self._fetchedResultsController = frc
return self._fetchedResultsController!
```

- Fetch

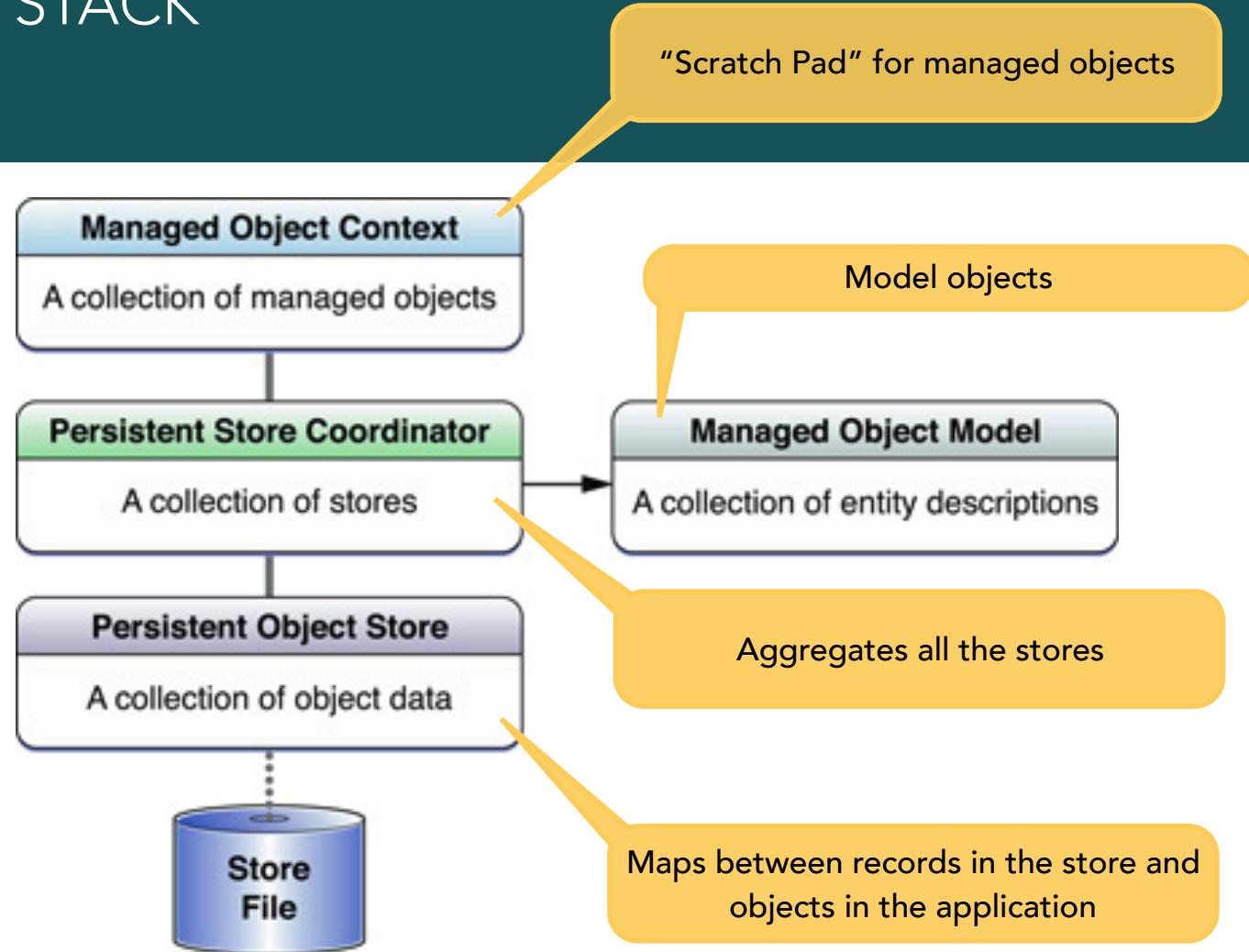
# CORE DATA STACK

# CORE DATA STACK

- Core Data stack
  - One or more managed object contexts
  - A single persistent store coordinator
  - Persistent store

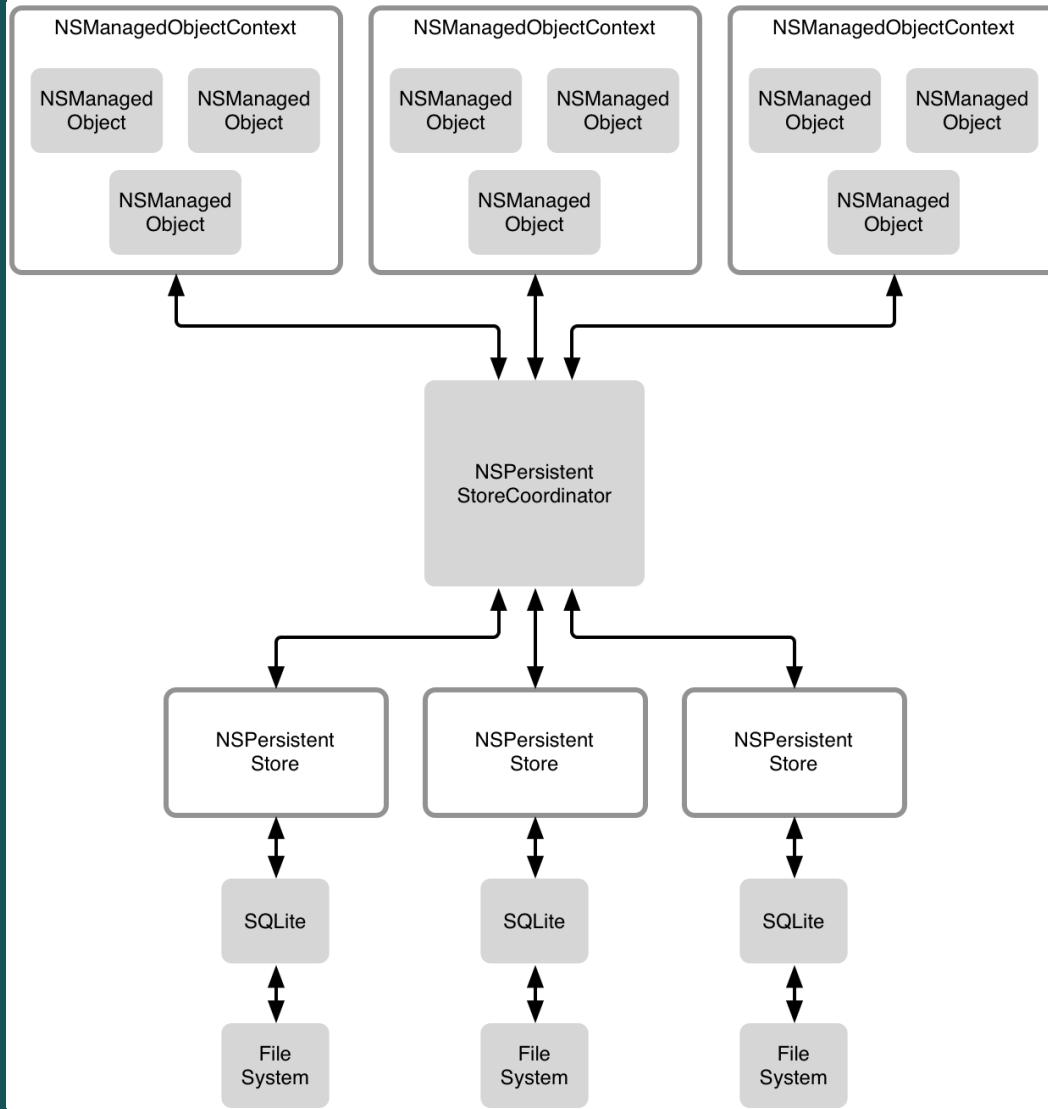


# CORE DATA STACK



# CORE DATA STACK

- Persistent Store Coordinator defines stack
- Can be multiple contexts, stores, etc.



# CORE DATA STACK

## XCODE TEMPLATE

- Xcode will create code stubs for working the persistent store
- AppDelegate with have a property called `persistentContainer`
- These are fairly complete and will not need to be dramatically altered

```
lazy var persistentContainer: NSPersistentContainer = {
    /*
        The persistent container for the application. This implementation
        creates and returns a container, having loaded the store for the
        application to it. This property is optional since there are legitimate
        error conditions that could cause the creation of the store to fail.
    */
    let container = NSPersistentContainer(name: "_017_CoreDataBasics")
    container.loadPersistentStores(completionHandler: { (storeDescription, error) in
        if let error = error as NSError? {
            // Replace this implementation with code to handle the error appropriately.
            // fatalError() causes the application to generate a crash log and terminate. You should not use this f
            // useful during development.

            /*
                Typical reasons for an error here include:
                * The parent directory does not exist, cannot be created, or disallows writing.
                * The persistent store is not accessible, due to permissions or data protection when the device is loc
                * The device is out of space.
                * The store could not be migrated to the current model version.
                Check the error message to determine what the actual problem was.
            */
            fatalError("Unresolved error \(error), \(error.userInfo)")
        }
    })
    return container
}()


```

# CORE DATA STACK

## XCODE TEMPLATE

- Provides a function to save the current store

```
// MARK: - Core Data Saving support

func saveContext () {
    print("Saving...")
    let context = persistentContainer.viewContext
    if context.hasChanges {
        do {
            try context.save()
            dump()

        } catch {
            // Replace this implementation with code to handle the error appropriately.
            // fatalError() causes the application to generate a crash log and terminate.
            // This behavior may look different in Swift 4.2 and later versions.
            let nserror = error as NSError
            fatalError("Unresolved error \(nserror), \(nserror.userInfo)")
        }
    }
}
```

# CORE DATA STACK

## WORKING WITH DATA

- Keys are NSString (i.e. 'thumbnailData', 'date', 'name')
- The value is whatever is stored
  - Numbers and booleans are NSNumber type objects
  - “To-many” relationships are NSSet
  - Single relationships are NSManagedObjects
  - Binary data is NSData

# CORE DATA STACK

## WORKING WITH DATA

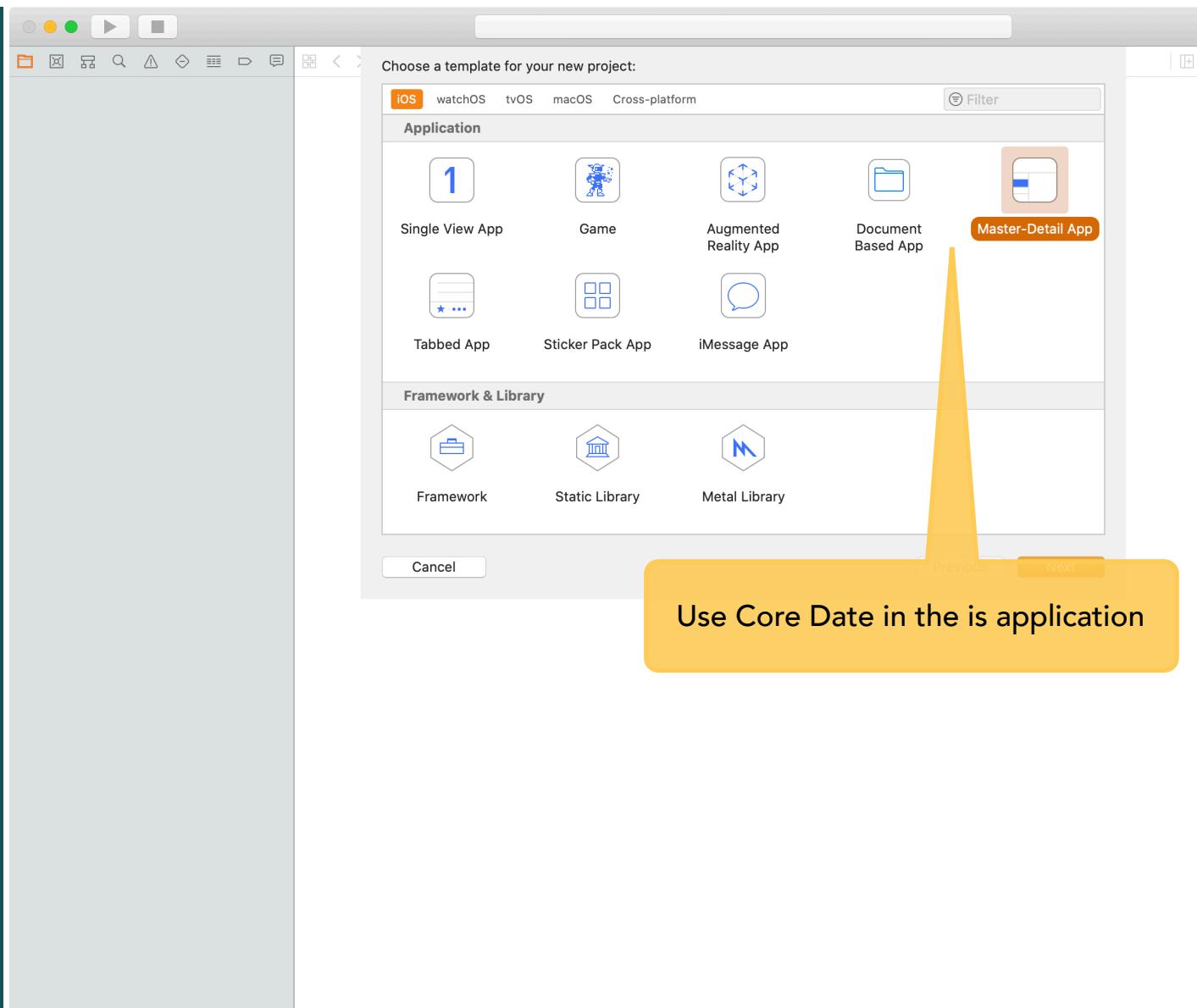
- Behind the scenes
  - SQL is generated
  - Results are fetched (lazily)
- Writes to database are only done in memory
  - Need to save them or you will lose data

```
func saveContext () {  
    if managedObjectContext.hasChanges {  
        do {  
            try managedObjectContext.save()  
        } catch {  
            // Replace this implementation with code to handle the error  
            // appropriately.  
            // abort() causes the application to generate a crash log and  
            // terminate. You should not use this function in a shipping  
            // application, although it may be useful during development.  
            let nserror = error as NSError  
            NSLog("Unresolved error \(nserror), \(nserror.userInfo)")  
            abort()  
        }  
    }  
}
```

# CREATE A CORE DATA MODEL

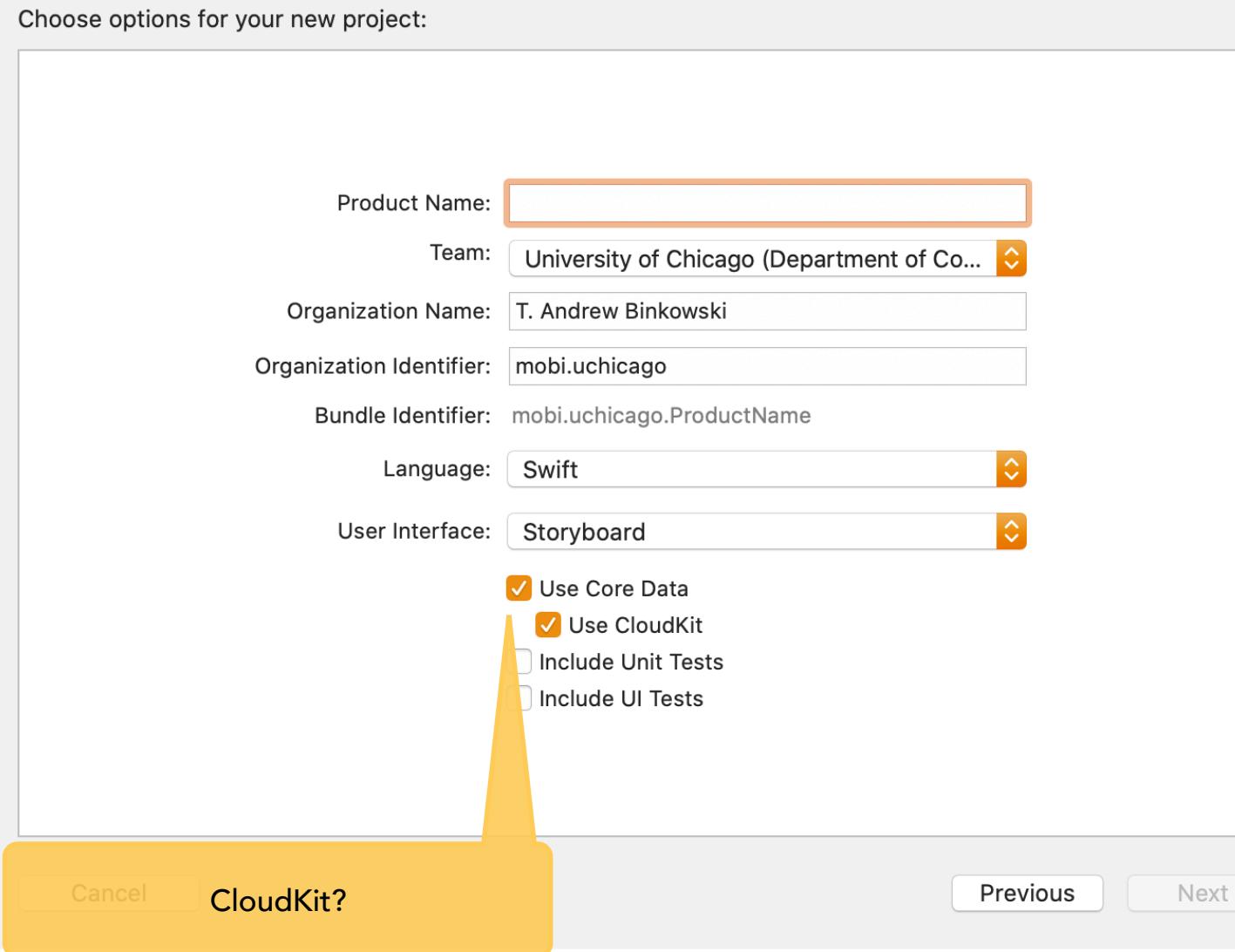
# CORE DATA MODEL

- Provide a simple data model (.xcdatamodel)
- Xcode will generate the boiler plate methods



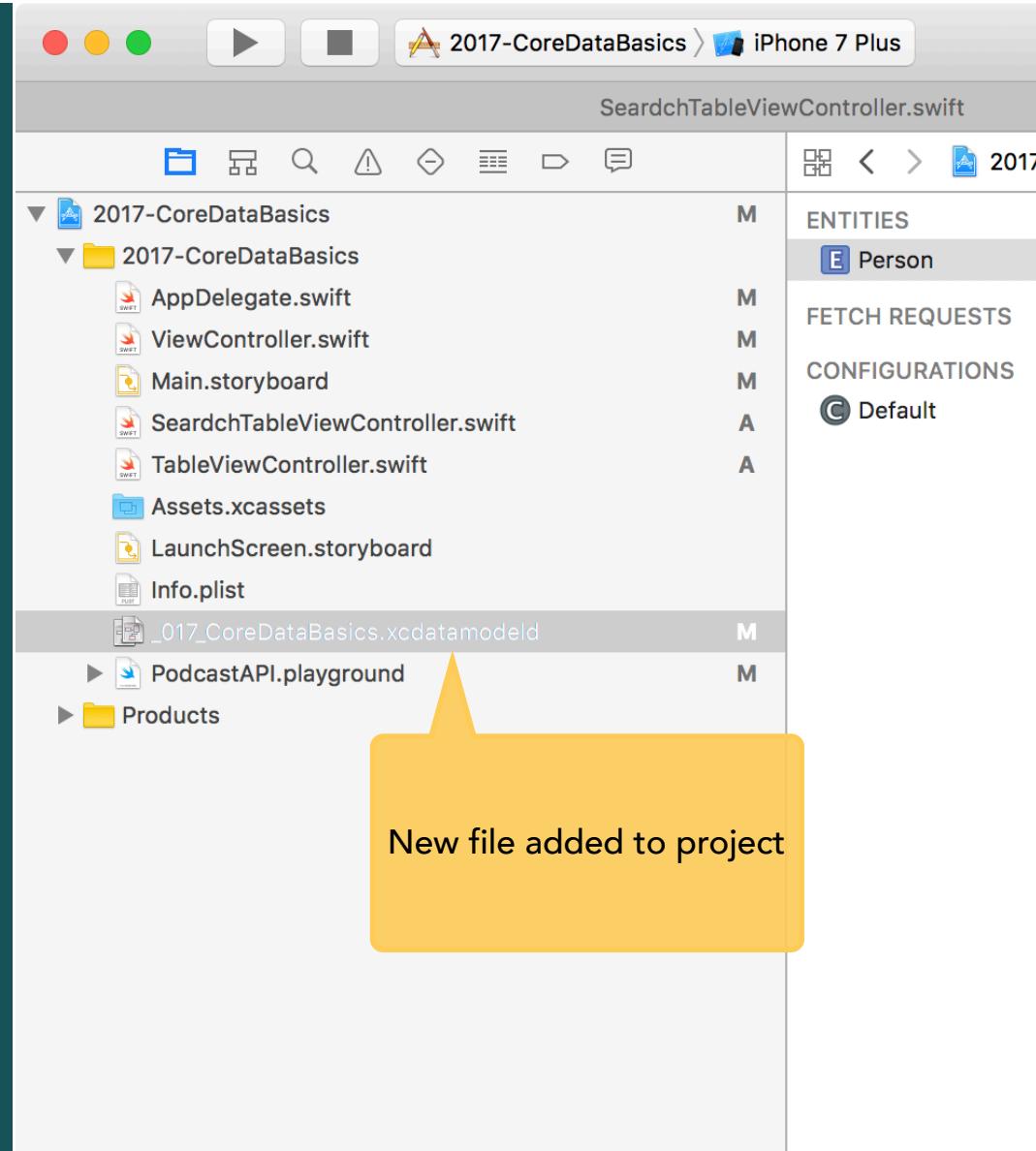
# CORE DATA MODEL

- Provide a simple data model (.xcdatamodel)
- Xcode will generate the boiler plate methods



# CORE DATA

- .xcdatamodeld is a directory
  - Stores data model
  - Provides built-in versioning
- Changing your data model will break your app
  - Think about the changes you may make



# CORE DATA

- Model is .xcdatamodel file
- Tasks
  - Add entities
  - Define attributes
- Attributes and relationships of an entity equivalent to @properties of a class

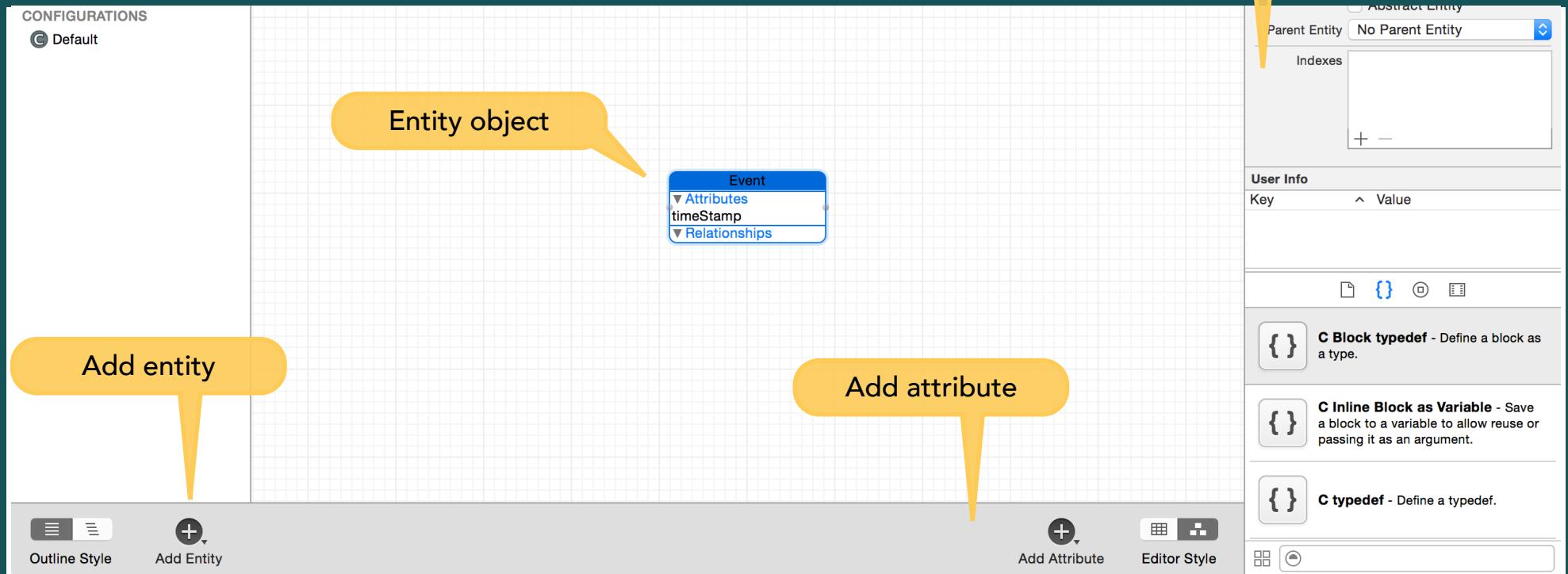
The screenshot shows the Xcode Core Data editor interface. The left sidebar lists 'ENTITIES', 'FETCH REQUESTS', and 'CONFIGURATIONS'. The 'Person' entity is selected, indicated by a blue highlight. The main area is divided into sections: 'Attributes', 'Relationships', and 'Fetched Properties'. Under 'Attributes', three attributes are defined: 'age' (Integer 16), 'name' (String), and 'timestamp' (Date). Under 'Relationships' and 'Fetched Properties', there are currently no entries.

Attribute	Type
N age	Integer 16
S name	String
D timestamp	Date

Relationship	Destination	Inverse

Fetched Property	Predicate

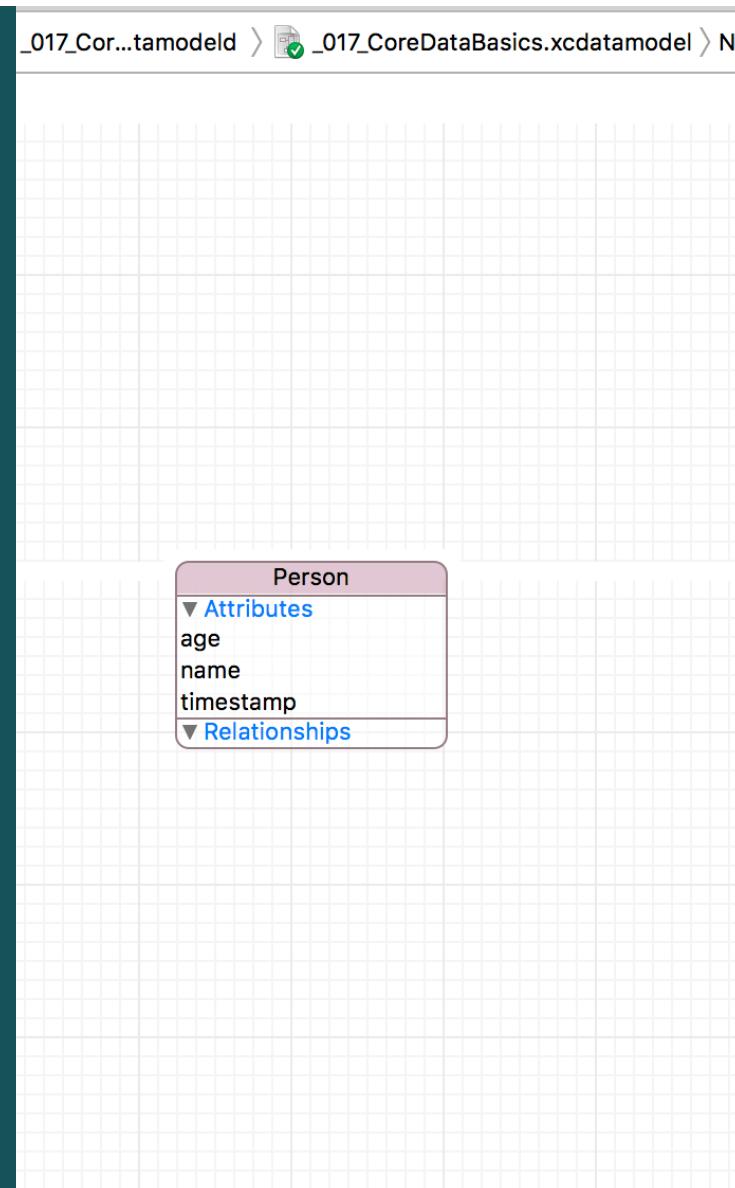
# CORE DATA



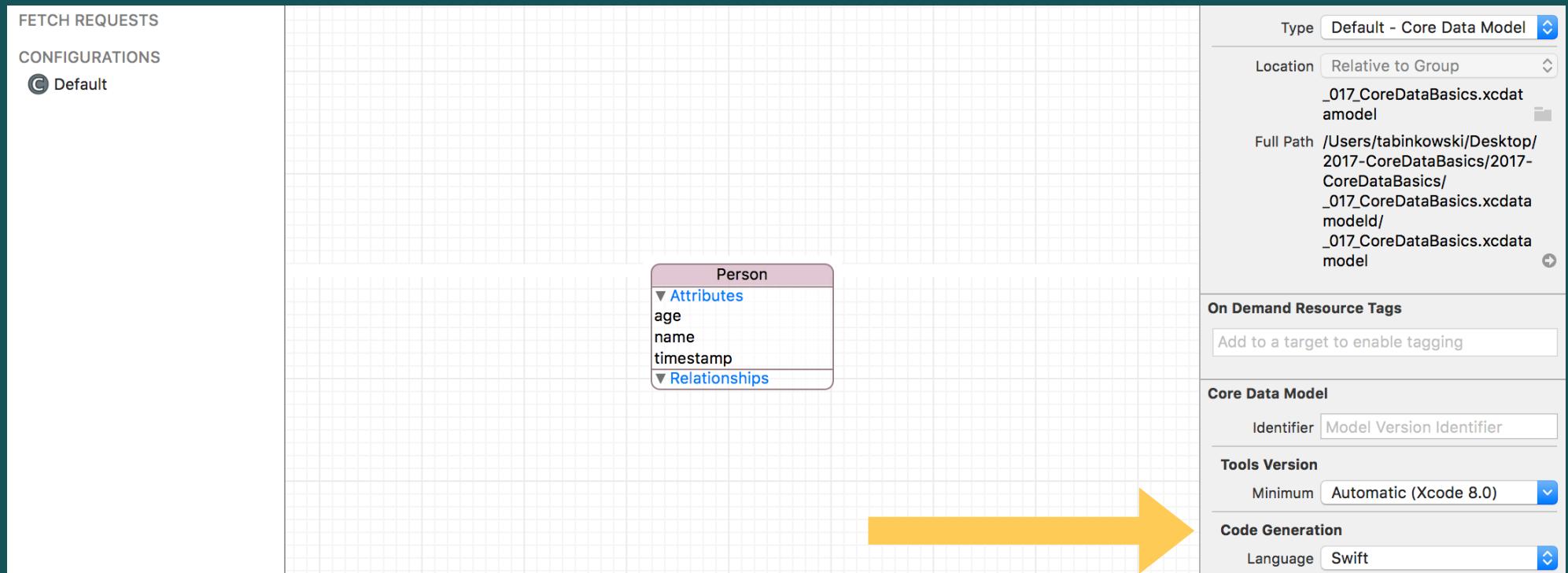
- Entity maps a database table to an object in our code

# CORE DATA

- NSManagedObject
  - Subclass of NSObject
  - Access your Core Data model and data
  - The returned object of a Core Data query
  - Key-Value pairs represent the properties of an entity
  - Not thread safe (i.e. do work on main thread)
- Subclass NSManagedObject to access the properties
  - Can also get them using key-value coding with a subclass (this is uncommon)



# CORE DATA



- Xcode will generate subclass files of NSManagedObject for you

# CORE DATA

The screenshot shows the Xcode Core Data editor with the following details:

- Entity List:** Shows 'ENTITIES' and 'Person' is selected.
- Attribute List:** Shows three attributes: 'age' (Integer 16, Optional), 'name' (String), and 'timestamp' (Date).
- Relationship List:** Shows no relationships.
- Fetched Properties List:** Shows no fetched properties.
- Attribute Inspector (Right Panel):**
  - Name:** age
  - Properties:**  Transient,  Optional,  Indexed
  - Attribute Type:** Integer 16
  - Validation:** No Value, Minimum, No Value, Maximum, 0, Default
  - Advanced:**  Index in Spotlight,  Store in External Record File
  - Use Scalar Type:**
- User Info (Right Panel):** Shows a table for key-value pairs.

- Attributes map to a database column
- Will be able to access via @property in code

# CORE DATA

- Attributes
  - Type
    - Maps to foundation object
  - Optional
  - Transient
    - Create a property that exist on object, but have no backing
  - Indexed
    - Create an index in the SQLite backend
    - Only what you will search on

The screenshot shows the Xcode Attribute inspector for an attribute named "age". The top bar includes standard icons for file, help, and navigation. The main area is titled "Attribute" and contains the following fields:

- Name:** age
- Properties:**  Transient  Optional  Indexed
- Attribute Type:** Integer 16
- Validation:** No Value  Minimum  
No Value  Maximum  
0  Default  Use Scalar Type
- Advanced:**  Index in Spotlight  Store in External Record File
- User Info:** (This section is currently empty)

# CORE DATA

- Attribute types
  - Foundation types
    - NSData for everything else
    - Binary Data for holding images
    - Transformable for nonstandard object types

The screenshot shows the Xcode Core Data editor interface. At the top, it displays the project name "SimpleCoreData" and target "iPhone 6". The main window title is "SimpleCoreData.xcdatamodel". On the left, there's a sidebar with sections for "ENTITIES", "FETCH REQUESTS", and "CONFIGURATIONS". Under "ENTITIES", the entity "Session" is selected. In the main pane, there are three expandable sections: "Attributes", "Relationships", and "Fetched Properties".

**Attributes**

Attribute ^	Type
B active	Boolean
D endTime	Date
D startTime	Date
S uuid	String

**Relationships**

Relationship ^	Destination	Inverse
(empty)	(empty)	(empty)

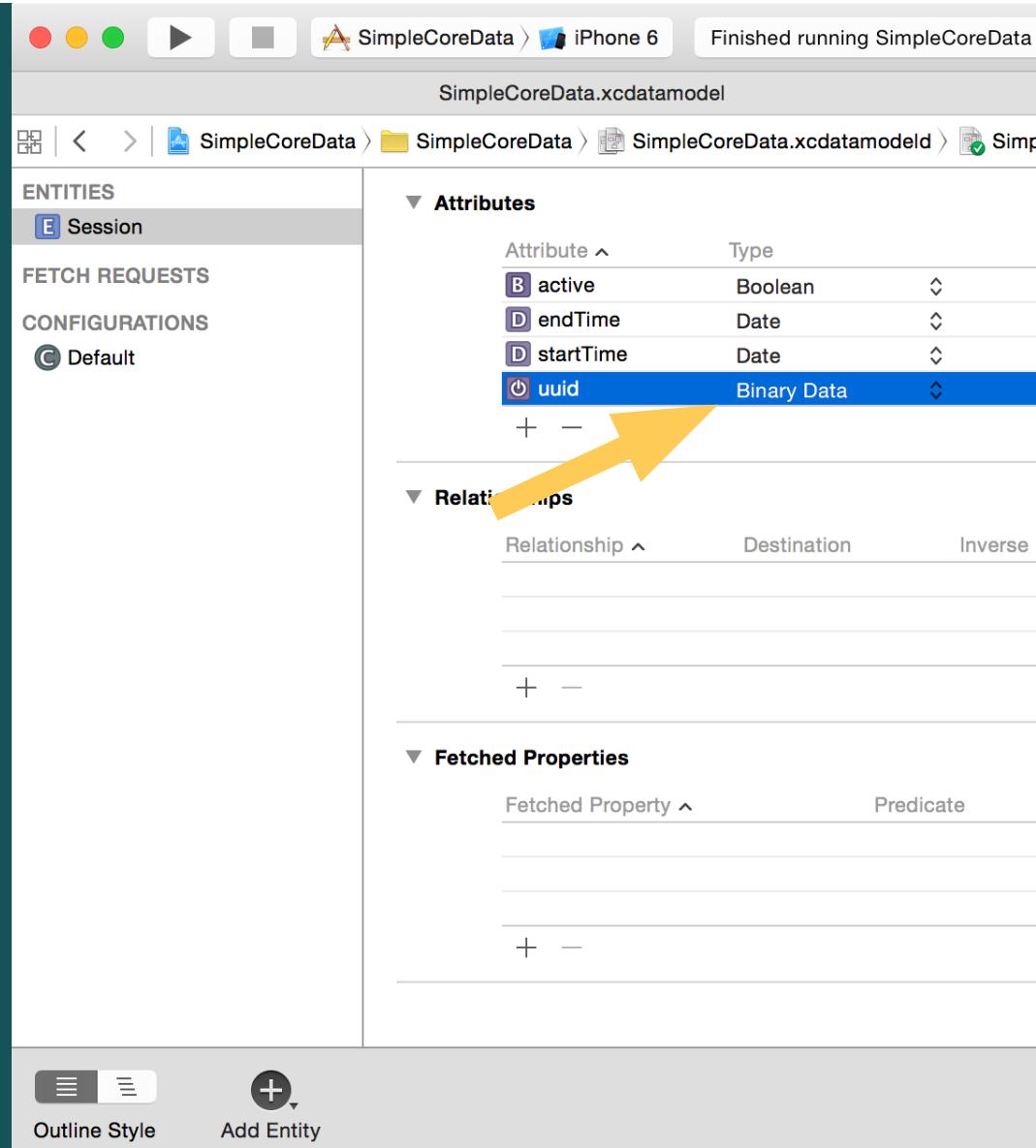
**Fetched Properties**

Fetched Property ^	Predicate
(empty)	(empty)

At the bottom of the editor, there are buttons for "Outline Style" and "Add Entity".

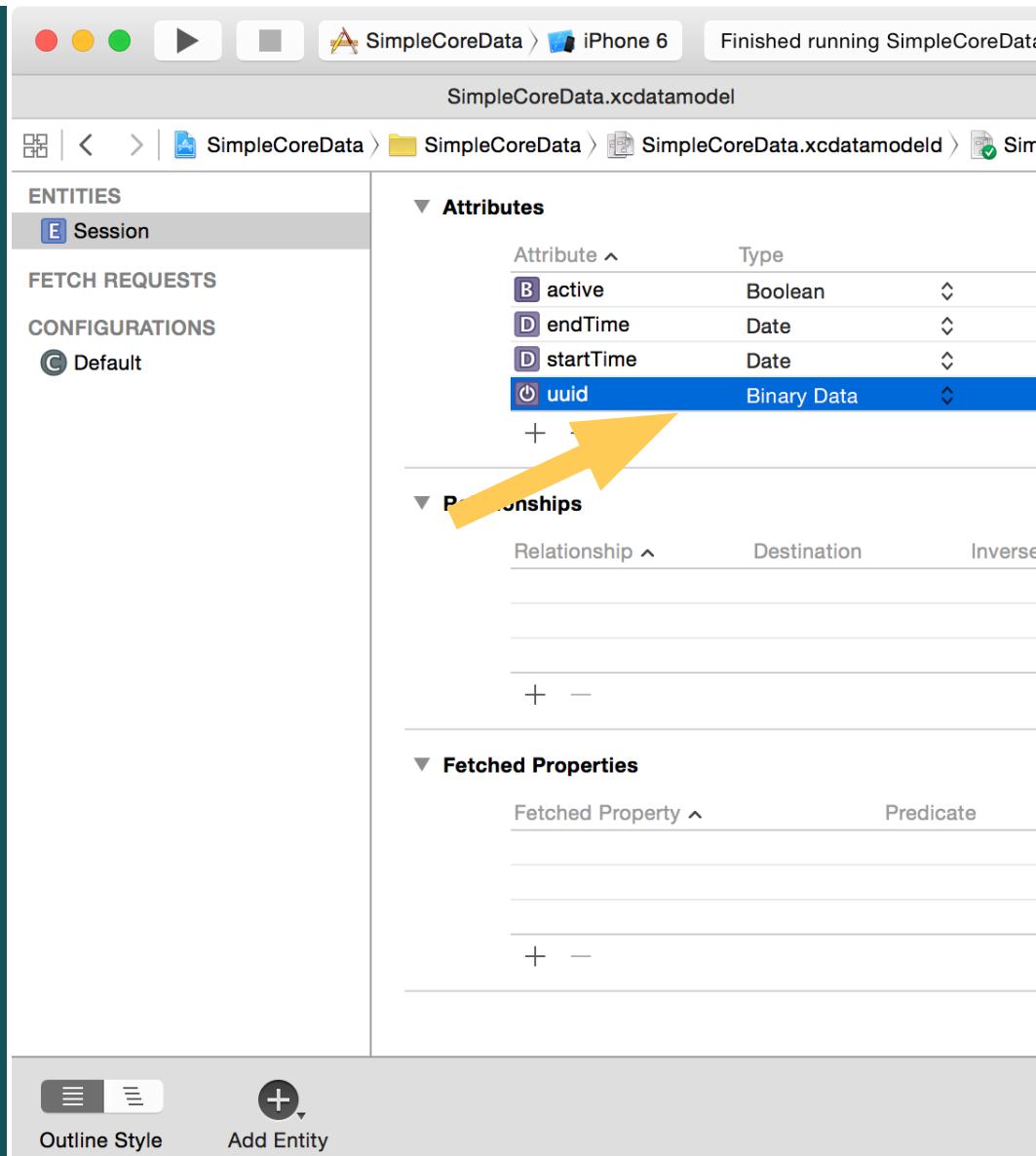
# CORE DATA

- External Binary Data
  - New in iOS5
- When enabled, Core Data heuristically decides on a per-value basis
  - Save the data directly in the database
  - Store a URI to a separate file



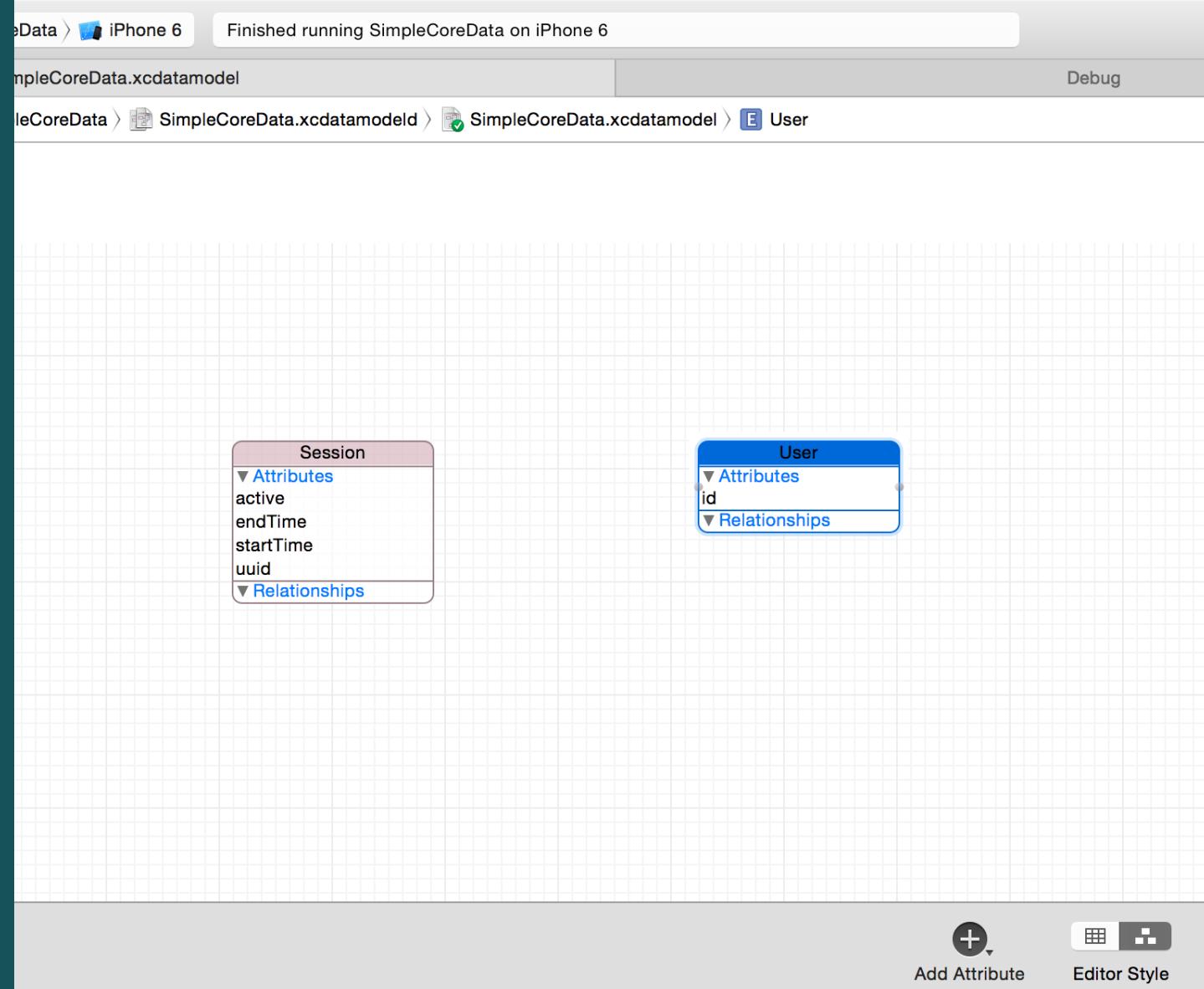
# CORE DATA

- Image storage guidelines
  - < 100 kb store in entity
  - < 1 mb store in a separate entity to avoid performance issues
  - > 1 mb store on disk and reference the path storing nonstandard object types within Core Data



# CORE DATA

- Entities can have relationships
- Example: Users and Session
  - A user can have multiple sessions
  - A session has only one user



# CORE DATA

The screenshot shows the Xcode Core Data editor with the User entity selected in the sidebar.

**Attributes**

Attribute	Type
S id	String

**Relationships**

Relationship	Destination	Inverse
M sessions	Session	user

**Fetched Properties**

Fetched Property	Predicate
+	-

**User Entity Configuration (Right Panel)**

- Name: sessions
- Properties:
  - Transient
  - Optional
- Destination: Session
- Inverse: user
- Delete Rule: Nullify
- Type: To Many
- Arrangement:
  - Ordered
  - Count:
    - Unbounded
    - Minimum
    - Unbounded
    - Maximum
- Advanced:
  - Index in Spotlight
  - Store in External Record File

**User Info**

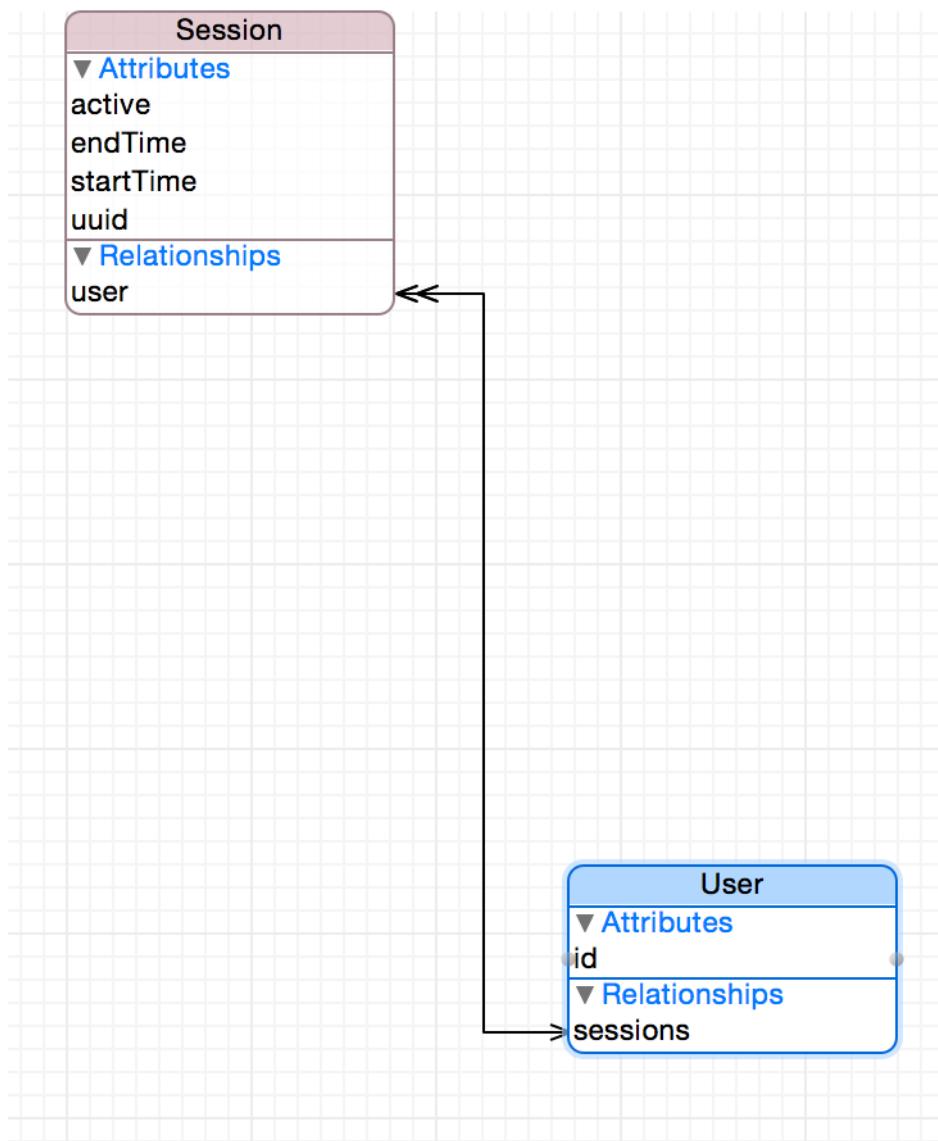
Icons:

**C Block typedef** - Define a block as a type.

**C Inline Block as Variable** - Save a block to a variable to allow reuse or passing it as an argument.

# CORE DATA

- “To Many” relationship
  - Each user can have many different sessions
- Maps to an `NSSet` of `NSManagedObjects`
  - Remember that sets have no order guarantee



# CORE DATA

- Inverse relationship from user to sessions
- Each session can have only 1 user

The screenshot shows the Xcode Core Data editor interface. At the top, it displays "Finished running SimpleCoreData on iPhone 6". The main window title is "SimpleCoreData.xcdatamodel". The sidebar on the left lists "ENTITIES" (Session, User), "FETCH REQUESTS", and "CONFIGURATIONS" (Default). The main content area is divided into sections: "Attributes", "Relationships", and "Fetched Properties".

**Attributes**

Attribute	Type
B active	Boolean
D endTime	Date
D startTime	Date
S uuid	String

**Relationships**

Relationship	Destination	Inverse
O user	User	◊ sessions ◊

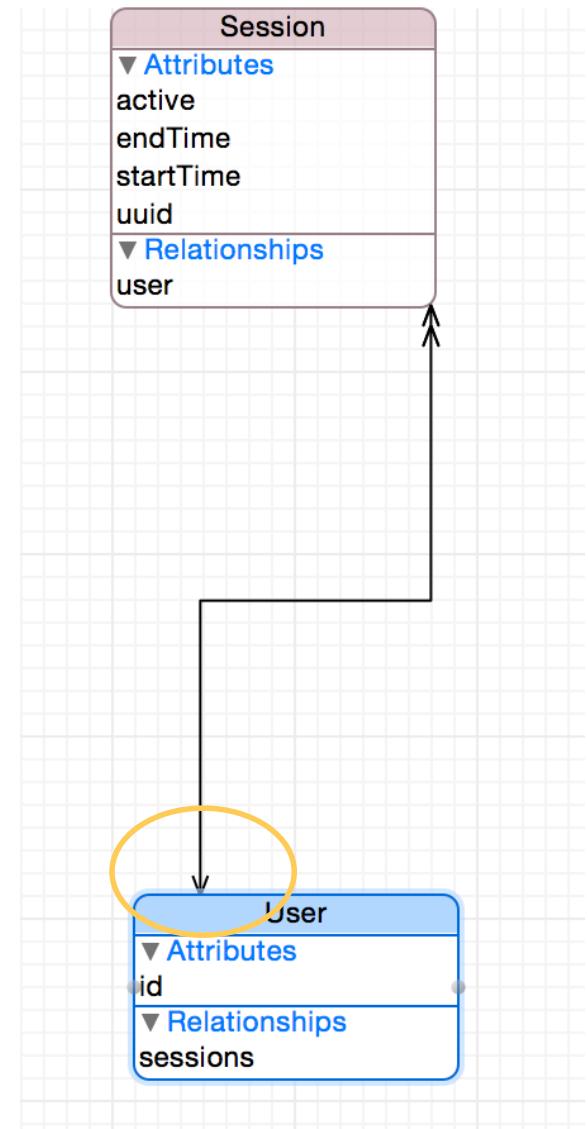
**Fetched Properties**

Fetched Property	Predicate

At the bottom of the editor, there are buttons for "Outline Style", "Add Entity", "Add Attribute", "Editor Style", and icons for "User Info" and "Parent Entity".

# CORE DATA

- Inverse goes only one way
- Maps to a single NSManagedObject



# CORE DATA

**E Session**

**E User**

**FETCH REQUESTS**

**CONFIGURATIONS**

**C Default**

**Attributes**

Entity	Attribute ^	Type
Session	B active	Boolean
Session	D endTime	Date
User	S id	String
Session	D startTime	Date
Session	S uuid	String

+ -

**Relationships**

Entity	Relationship ^	Destination	Inverse
User	M sessions	Session	◊ user ◊
Session	O user	User	◊ sessions ◊

+ -

**Fetched Properties**

Entity	Fetched Property ^	Predicate
--------	--------------------	-----------

**Name** Multiple Values

**Class** NSManagedObject

Abstract Entity

**Parent Entity** No Parent Entity

**Indexes**

+ -

**User Info**

Key ^ Value

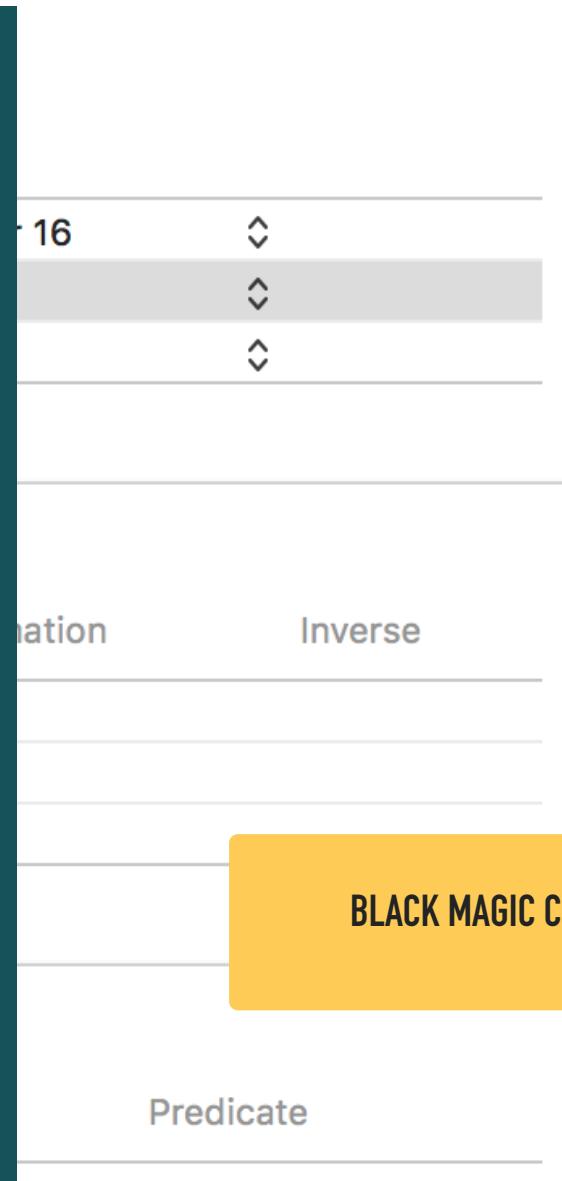
**C Block typedef** - Define a block as a type.

**C Inline Block as Variable** - Save a block to a variable to allow reuse or passing it as an argument

# NSMANAGEDOBJECT SUBCLASS IN IOS10

## NSMANAGEDOBJECT SUBCLASS IN IOS10

- Xcode 8+  
`codegen` will  
create subclasses  
for you at compile  
time
  - You don't ever  
see them



**Entity**

Name	Person
<input type="checkbox"/> Abstract Entity	
Parent Entity	No Parent Entity

**Class**

Name	Person
Module	Global namespace
Codegen	Class Definition

**Indexes**

No Content	
+	-

**Constraints**

No Content	
+	-

**User Info**

Key	Value
-----	-------

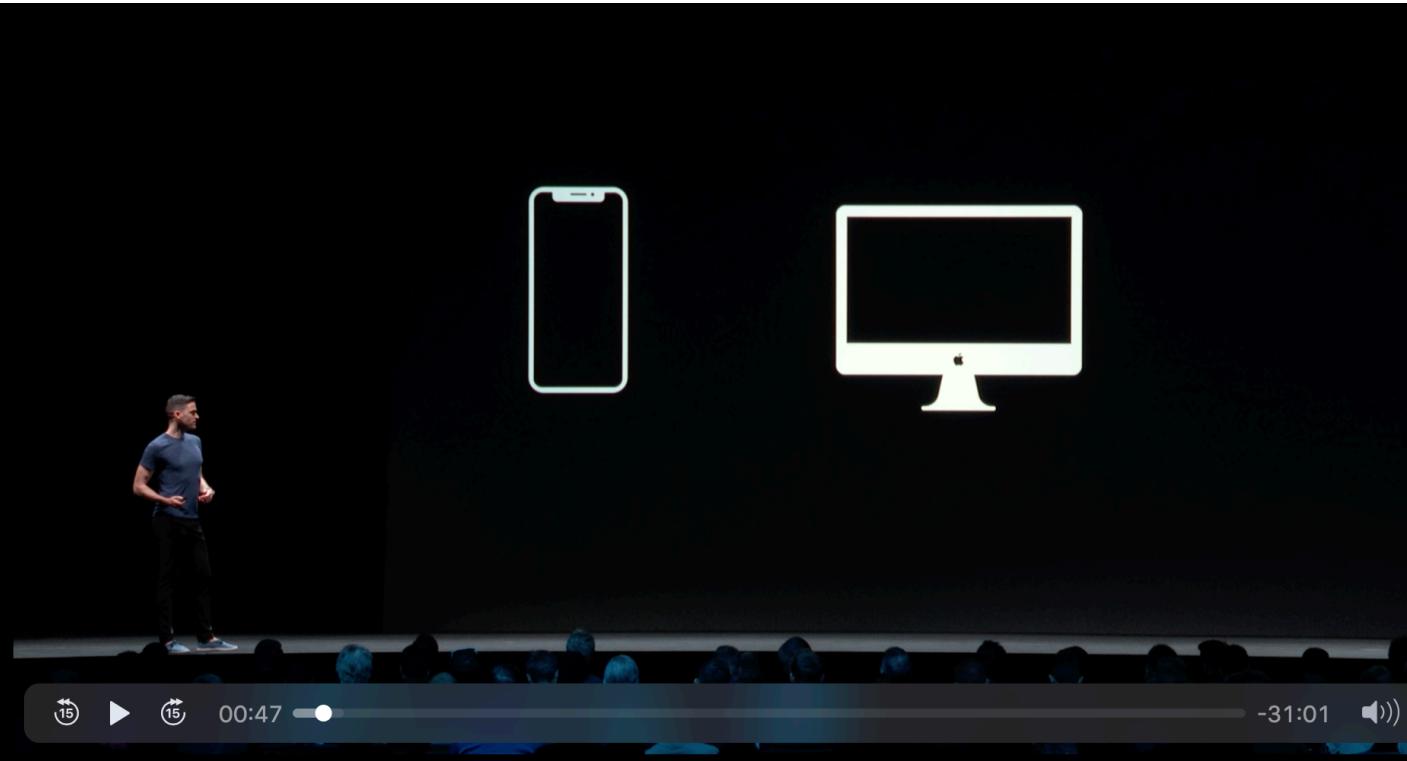
## NSMANAGEDOBJECT SUBCLASS IN IOS10

- Thats it 😊

CLOUDKIT + COREDATA  
TOGETHER AT LAST

# CLOUDKIT + COREDATA

- WWDC 2019 the dream becomes reality



[Overview](#)   [Transcript](#)

## Using Core Data With CloudKit

CloudKit offers powerful, cloud-syncing technology while Core Data provides extensive data model persistence APIs. Learn about combining these complementary technologies to easily build cloud

# CLOUDKIT + COREDATA

- WWDC 2019 the dream becomes reality

## Setting Up Core Data with CloudKit

Set up the classes and capabilities that sync your store to CloudKit.

---

### Overview

To sync your Core Data store to CloudKit, you enable the CloudKit capability for your app. You also set up the Core Data stack with a persistent container that is capable of managing one or more local persistent stores that are backed by a CloudKit private database.

### Configure a New Xcode Project

When you create a new project, you specify whether you want to add support for Core Data with CloudKit directly from the project setup interface. The resulting project instantiates an `NSPersistentCloudKitContainer` in your app's delegate. Once you enable CloudKit in your project, you use this container to manage one or more local stores that are backed with a CloudKit database.

1. Choose File > New > Project to create a new project.
2. Select a project template to use as the starting point for your project, and click Next.
3. Select the Use Core Data and Use CloudKit checkboxes.

# CLOUDKIT + COREDATA

- Replicates data model in CloudKit
- Schedule notifications to sync data

Article

## Setting Up Core Data with CloudKit

Set up the classes and capabilities that sync your data between iCloud and your app.

### Overview

To sync your Core Data store to CloudKit, you enable the CloudKit capability for your app. You also set up the Core Data stack with a persistent container that includes one or more local persistent stores that are backed by a CloudKit database.

### Configure a New Xcode Project

When you create a new project, you specify whether you want to include CloudKit directly from the project setup interface. The resulting `NSPersistentCloudKitContainer` in your app's delegate handles the setup for you. In your project, you use this container to manage one or more local persistent stores that are backed by a CloudKit database.

1. Choose File > New > Project to create a new project.
2. Select a project template to use as the starting point.
3. Select the Use Core Data and Use CloudKit checkboxes.
4. Enter any other project details and click Next.

# CLOUDKIT + COREDATA

	Core Data	CloudKit
Objects	NSManagedObject	CKRecord
Models	NSManagedObjectModel	Schema
Stores	NSPersistentStore	CKRecordZone / CKDatabase

# CLOUDKIT + COREDATA

- WWDC 2019  
Making App With  
Core Dat
  - <https://developer.apple.com/videos/play/wwdc2019/230>

The screenshot shows a video player interface with a presentation slide. The slide has a dark background and features the following content:

- Configuring Managed Object Contexts**
- Query generations provide stability
- `try container.viewContext.setQueryGenerationFrom(.current)`
- Automatic merging provides freshness
- `context.automaticallyMergesChangesFromParent = true`

At the bottom of the slide, there is a navigation bar with icons for back, forward, and search, along with a timestamp of 06:00. Below the slide, the video player interface includes controls for volume, brightness, and a progress bar. At the very bottom, there are links for "Overview" and "Transcript" and a search icon.

**Making Apps with Core Data**

Core Data helps manage the flow of data throughout your app. Hear about new features in Core Data that make your code simpler and more powerful, including derived attributes, history tracking, change

# CLOUDKIT + COREDATA

The screenshot shows the Xcode interface with the Core Data editor open. The title bar indicates the project has finished running on an iPhone 11 Pro Max. The left sidebar lists project files, and the main area displays the Core Data model for the 'Event' entity.

**Entities:** Event

**Attributes:**

Attribute	Type
D timestamp	Date

**Relationships:**

Relationship	Destination	Inverse

**Fetched Properties:**

# CLOUDKIT + COREDATA

running mpc51033-2019-autumn-cloudkit-coredata on iPhone 11 Pro Max

The screenshot shows the Xcode Core Data editor for the 'mpcs51033\_2019\_autumn\_cloudkit\_coredata.xcdatamodel' file. The left sidebar lists entities: 'Event' (selected), 'FETCH REQUESTS', and 'CONFIGURATIONS'. The main area displays the 'Event' entity configuration with three sections: 'Attributes', 'Relationships', and 'Fetched Properties'. The 'Attributes' section contains one attribute: 'timestamp' (Date type). The 'Relationships' section is currently empty. The 'Fetched Properties' section is also currently empty.

iCloud.mpc51033-2019-autumn-cloudkit-coredata ▾ > ● Development ▾ > Schema ▾

## Record Types ▾

System Types  
Users

Custom Types  
**CD\_Event**

**New Type** **Delete Type**

Field Name	Field Type
<b>System Fields</b>	
recordName	Reference
createdBy	Reference
createdAt	Date/Time
modifiedBy	Reference
modifiedAt	Date/Time
changeTag	String
<b>Custom Fields</b>	
CD_entityName	String
CD_timestamp	Date/Time
<b>Add Field</b>	

# CLOUDKIT + COREDATA

iCloud.mpcs51033-2019-autumn-cloudkit-coredata ~ > Development > Data ~

## Records

Database  
Private Database  
abinkowski@uchicago.edu

Zone  
com.apple.coredata.cloudkit.zone

Using  
Query   Fetch   Changes

Type: CD\_Event

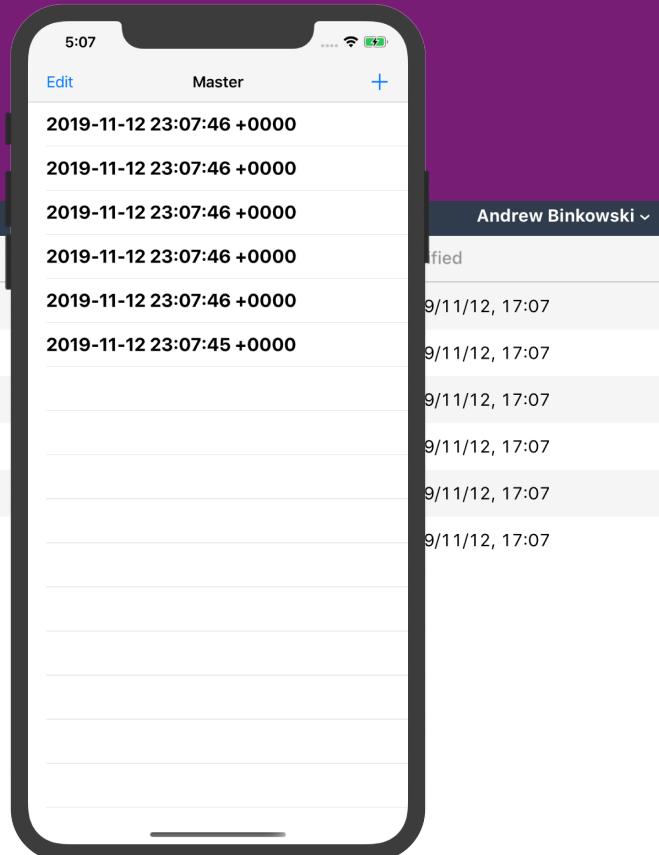
Filter: None

Sort: None

**Query Records**

Query for records of the type "CD\_Event" that are in the "com.apple.coredata.cloudkit.zone" zone of the "private" database.

New   Accept Shared   Delete



# CLOUDKIT + COREDATA

```
// MARK: - Core Data stack

lazy var persistentContainer: NSPersistentCloudKitContainer {
    /*
        The persistent container for the application. This implementation
        creates and returns a container, having traded the store for the
        application to it. This property is optional since there are legitimate
        error conditions that could cause the creation of the store to fail.
    */
    let container = NSPersistentCloudKitContainer(name:
        "mpcs51033_2019_autumn_cloudkit_coredata")
    container.loadPersistentStores(completionHandler: { (storeDescription, error) in
        if let error = error as NSError? {
            // Replace this implementation with code to handle the error appropriately.
            // fatalError() causes the application to generate a crash log and terminate.
            // You should not use this function in a shipping application, although it may
            // be useful during development.
        }
    })
}
```

CLOUD

# CLOUDKIT + COREDATA



- Leave it alone
- Create another store to self manage other data

# CLOUDKIT + COREDATA

- Reactions
  - Great when it works
  - Terrible when it doesn't

Article

## Setting Up Core Data with CloudKit

Set up the classes and capabilities that sync your local Core Data store with the CloudKit database.

### Overview

To sync your Core Data store to CloudKit, you enable the CloudKit capability for your app. You also set up the Core Data stack with a persistent container that contains one or more local persistent stores that are backed by a CloudKit database.

### Configure a New Xcode Project

When you create a new project, you specify whether you want to include CloudKit directly from the project setup interface. The resulting `NSPersistentCloudKitContainer` in your app's delegate handles the setup for you. When you add CloudKit to an existing project, you use this container to manage one or more local persistent stores that are backed by a CloudKit database.

1. Choose File > New > Project to create a new project.
2. Select a project template to use as the starting point.
3. Select the Use Core Data and Use CloudKit checkboxes.
4. Enter any other project details and click Next.



THE UNIVERSITY OF  
CHICAGO



MPCS 51033 • AUTUMN 2019 • SESSION 7

---

# BACKENDS FOR MOBILE APPLICATIONS