



THE UNIVERSITY OF
CHICAGO



MPCS 51033 • AUTUMN 2017 • SESSION 7

BACKENDS FOR MOBILE APPLICATIONS

CLOUDKIT



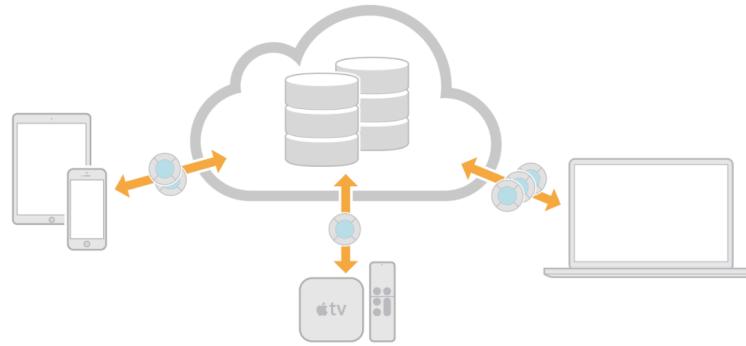
CLOUDKIT

- CloudKit
QuickStart

Table of Contents	
Introduction	
▼	Enabling CloudKit in Your App
About Containers and Databases	
Setup	
Enable iCloud and Select CloudKit	
Access CloudKit Dashboard	
▼	Share Containers Between Apps
Add Containers to an App	
Create Custom Containers	
Verify Your Steps	
Create an iCloud Account for Development	
Recap	
▼	Creating a Database Schema by Saving Records
▼ About Designing Your Schema	
Separate Data into Record Types	
Decide on Names for Your Records	
Create Records Programmatically	
Save Records	
Enter iCloud Credentials Before Running Your App	
Alert the User to Enter iCloud Credentials	
Run Your App	
► Verify Your Steps	
Recap	
▼	Using CloudKit Dashboard to Manage Databases
About the Development and Production Environments	
Select Your Container	
Reset the Development Environment	
Create and Delete Record Types	
Add, Modify, and Delete Records	
Search Records	
Sort Records	
Recap	
▼	Fetching Records
Fetch Records by Identifier	
Fetch and Modify Records	
Query for Records Using Predicates	
Recap	
►	Using Asset and Location Fields
▼	Adding Reference Fields

About This Document

This document gets you started creating a CloudKit app that stores structured app and user data in iCloud. Using CloudKit, instances of your app—launched by different users on different devices—have access to the records stored in the app's database. Use CloudKit if you have model objects that you want to persist and share between multiple apps running on multiple devices. These model objects are stored as records in the database and can be provided by you or authored by the user.



You'll learn how to:

- Enable CloudKit in your Xcode project and create a schema programmatically or with CloudKit Dashboard
- Fetch records and subscribe to changes in your code
- Use field types that are optimized for large data files and location data
- Subscribe to record changes to improve performance
- Test your CloudKit app on multiple devices before uploading it to the App Store, Mac App Store, or Apple TV App Store.
- Deploy the schema to production and keep it current with each release of your app

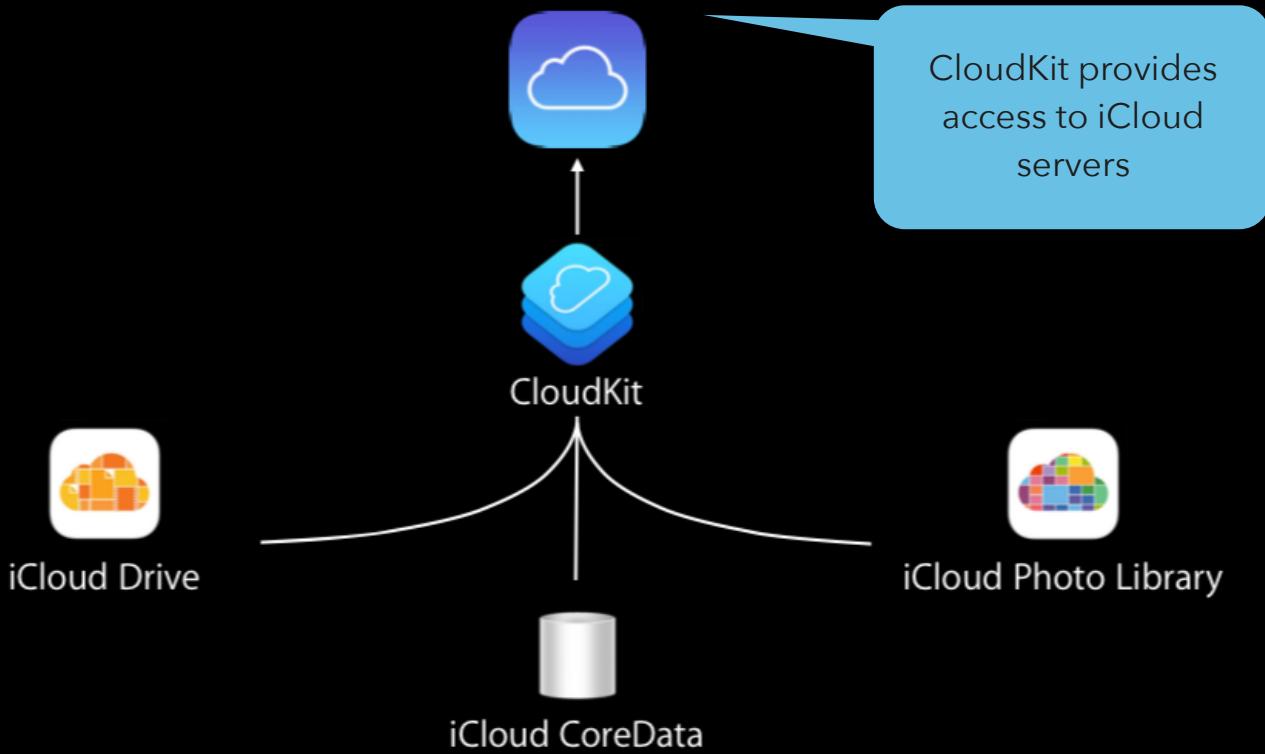
See [Glossary](#) for the definition of database terms used in this book.

See Also

The following WWDC sessions provide more CloudKit architecture and API details:

- [WWDC 2014: Introducing CloudKit](#) introduces the basic architecture and APIs used to save and fetch records.
- [WWDC 2014: Advanced CloudKit](#) covers topics such as private data, custom record zones, ensuring data integrity, and effectively modeling your data.
- [WWDC 2015: CloudKit Tips and Tricks](#) explore some of its lesser-known features and best practices for subscriptions and queries.
- [WWDC 2016: What's New with CloudKit](#) covers the new sharing APIs that lets you share private data between iCloud users.
- [WWDC 2016: CloudKit Best Practices](#) best practices from the CloudKit engineering team about how to take advantage of the APIs and push notifications in order to provide your users with the best experience.

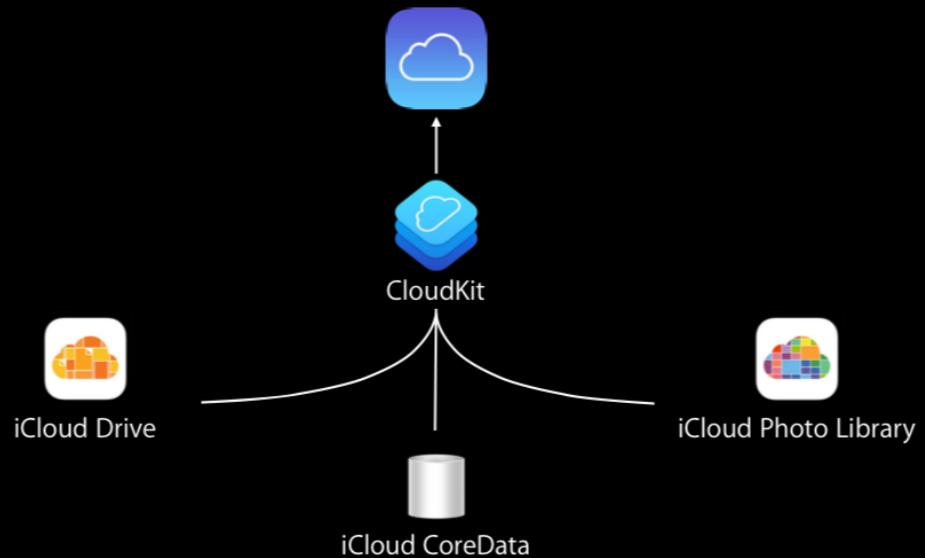
CLOUDKIT



CLOUDKIT

- Access to iCloud servers
- Supported on macOS, tvOS, iOS, watchOS and web (JS API)

#1 consideration
on choosing over
other services



CLOUDKIT

- Basically Free
- Public scales with users to PB
- Private db is charged against users quota
 - Permission can make anything private

Getting Started with CloudKit for free.

CloudKit provides a generous amount of free public storage and data transfer to help you get started. Sign in to the [CloudKit Dashboard](#) to view your quota and project usage.

10 GB

Asset
storage

**100
MB**

Database
storage

2 GB

Data
transfer

40

Requests
per second

CLOUDKIT

- Nice problem to have



Capacity scales with your users.

The amount of public storage and data transfer allocated to your apps will grow with every new active user—all for free with very high limits. Calculate the amount of storage you'll gain as your number of active users grows.

10,000,000

Active Users



1 PB

Asset
storage

Based on:
100 MB per user

10 TB

Database
storage

Based on:
1 MB per user

200 TB

Data
transfer

Based on:
20 MB per user

400

Requests
per sec.

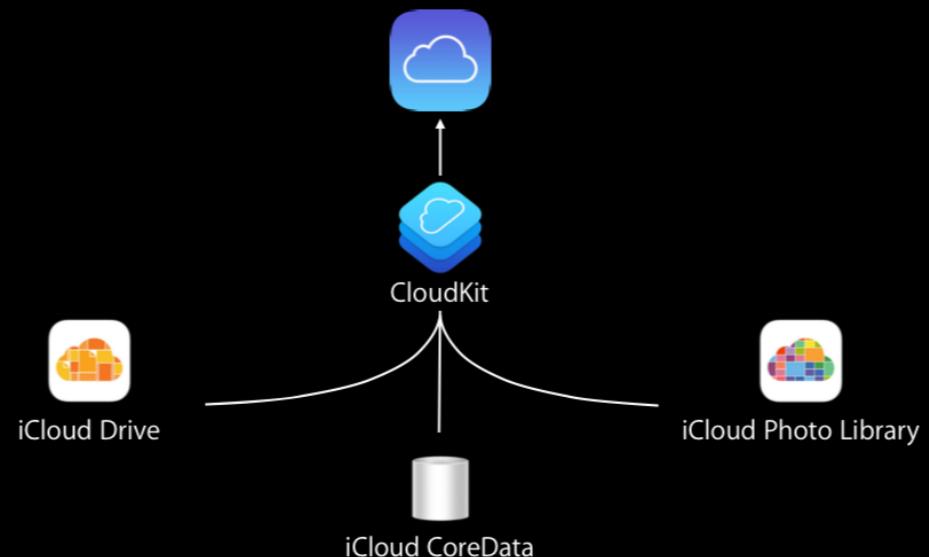
Based on:
4 per 100,000 users

\$0

Total Cost

CLOUDKIT

- Uses iCloud accounts
 - Logged in accounts used to identify user
 - Not logged in users can have read only anonymous access



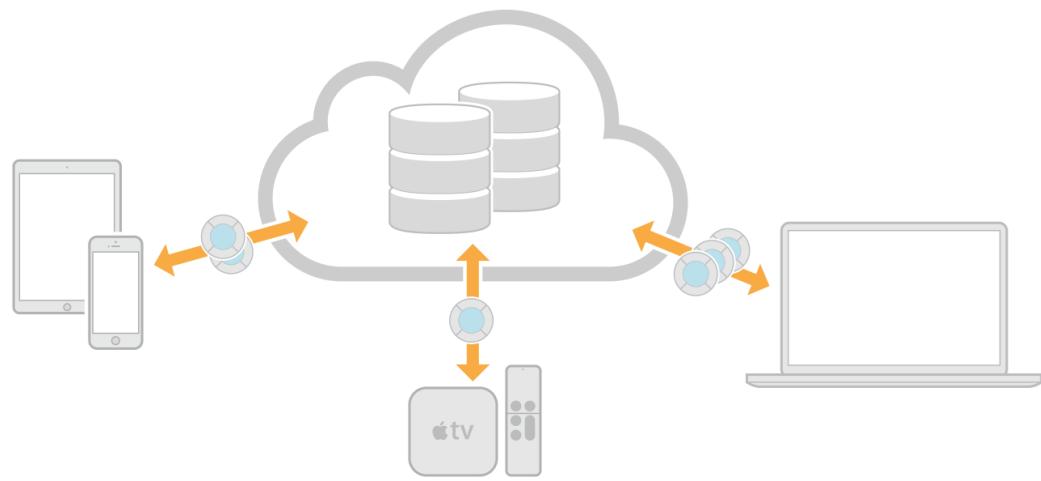
CLOUDKIT

- Databases
 - Public
 - Private
 - Shared



CLOUDKIT

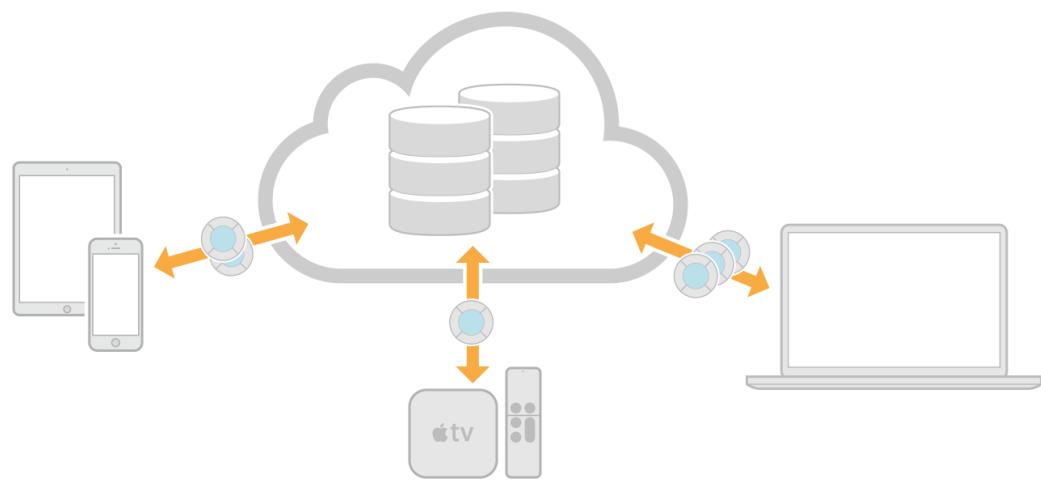
- Structured and bulk data
 - Data with types
 - Blobs



CLOUDKIT

CORE OBJECT

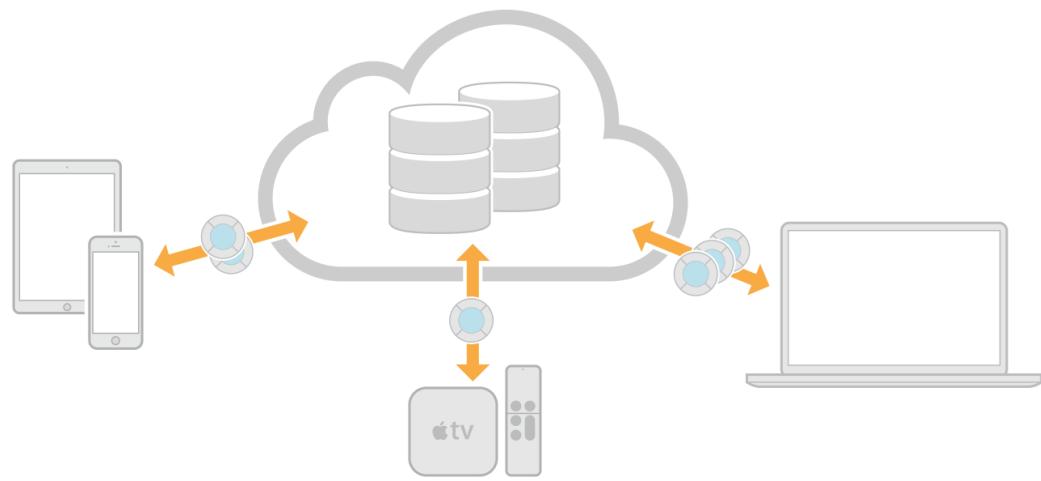
- Containers
- Databases
- Records
- Record Zones
- Record Identifiers
- Shares
- References
- Assets



CLOUDKIT

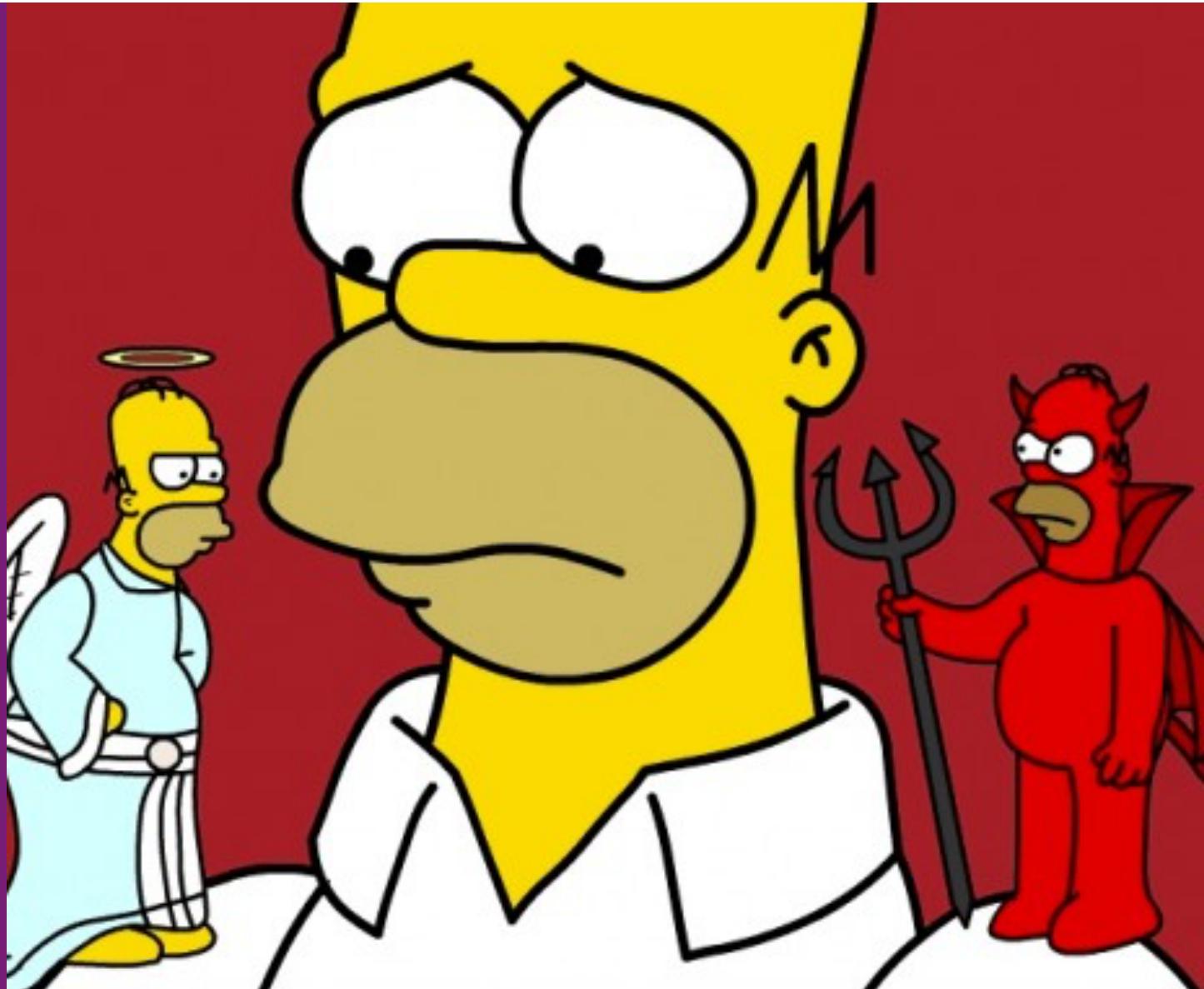
- Transport layer, not local persistence
- You still need to figure out what to do with the data once you have it on your device
 - Cache
 - Mirror

#2 consideration
on choosing over
other services



CLOUDKIT

- CloudKit's place in mobile backends -
The good
 - Convenient
 - Free
 - Zero authentication
 - Push notifications
 - Proven (Apple actually uses it)



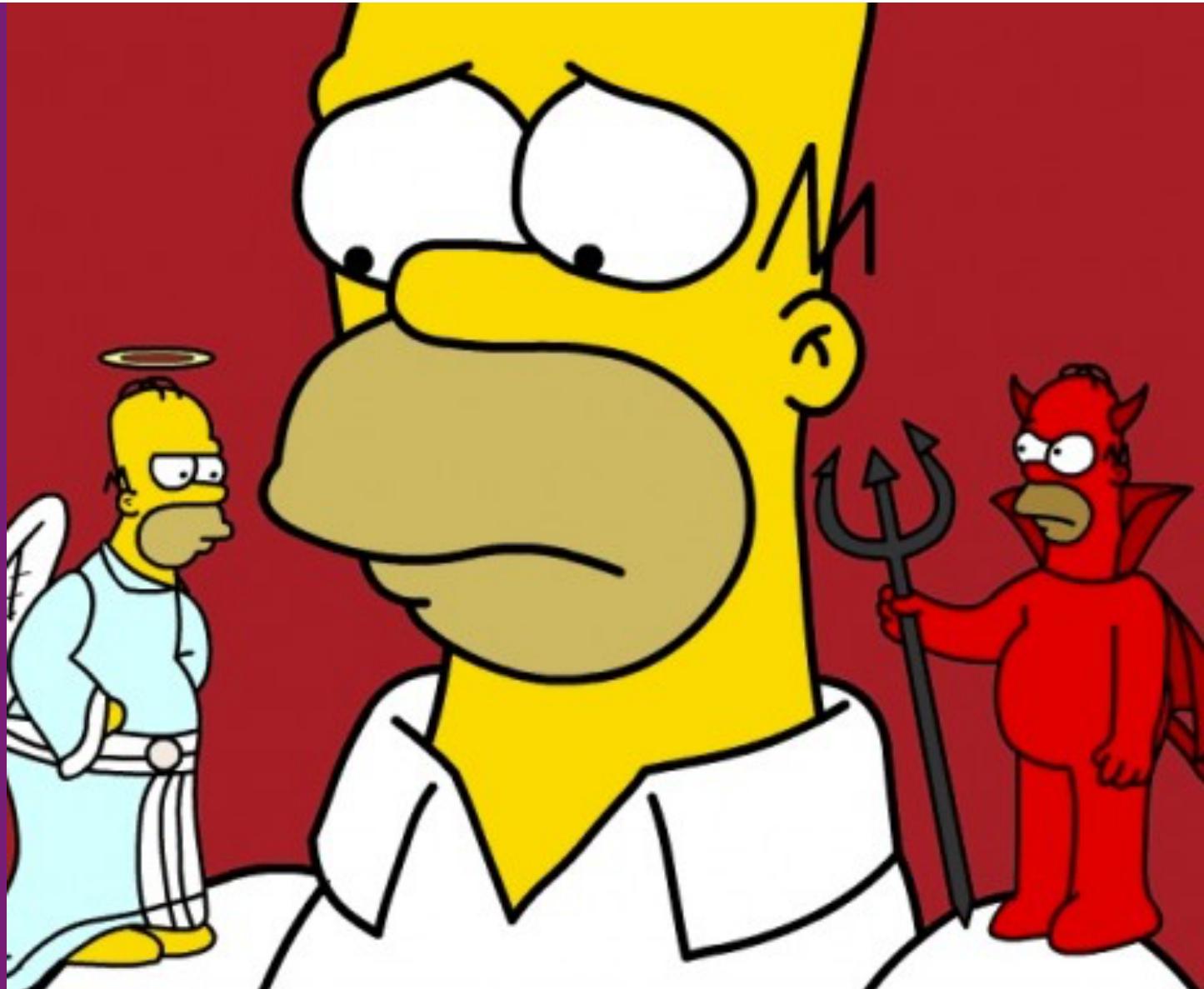
CLOUDKIT

- CloudKit's place in mobile backends -
The good
 - Convenient
 - Free
 - Zero authentication
 - Push notifications
 - Proven (Apple actually uses it)



CLOUDKIT

- CloudKit's place in mobile backends -
The bad
 - No local storage
 - Platform lock (account & device)
 - No native server logic



CLOUDKIT

- CloudKit's place in mobile backends - The bad
 - No local storage
 - Platform lock (account & device)
 - No native server logic



BUT WAIT...

CLOUDKIT AND CORE DATA

- WWDC 2019 announcement

The screenshot shows a video player interface. At the top, there's a navigation bar with icons for back, forward, and search. The main title 'Using Core Data with CloudKit' is displayed diagonally above the video frame. Below the title, a subtitle 'Everything should sync' is visible. In the video frame, a man in a blue t-shirt and dark pants is standing on a stage, gesturing with his hands. A subtitle at the bottom of the frame reads 'me on whatever device I have,'. The video player has a progress bar showing '00:25' and a total duration of '-31:23'. Below the video frame, there are buttons for 'Overview' and 'Transcript' and a magnifying glass icon for search.

Using Core Data With CloudKit

CloudKit offers powerful, cloud-syncing technology while Core Data provides extensive data modeling and persistence APIs. Learn about combining these complementary technologies to easily build cloud-backed applications. See how new Core Data APIs make it easy to manage the flow of data through your application, as well as in and out of CloudKit. Join us to learn more about combining these frameworks to provide a great experience across all your customers' devices.

CLOUDKIT AND CORE DATA

- There are actually 2 ways to use CloudKit
 - Manually
 - Fully managed with Core Data Sync

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.
4. Enter any other project details and click Next.
5. Specify a location for your project and click Create.

THIS METHOD IS MORE CORE DATA THAN

ENABLING CLOUDKIT IN YOUR APPLICATION

ENABLING CLOUDKIT IN YOUR APPLICATION

The screenshot shows the Xcode interface with a project named "CloudKit-1" selected. The "Signing & Capabilities" tab is active. In the "Targets" section, "CloudKit-1" is selected. Under the "Signing" heading, the "Automatically manage signing" checkbox is checked, and the "Team" dropdown is set to "University of Chicago (Department of Compu...)".

NEED PAID DEVELOPER ACCOUNT OR SCHOOL PROGRAM

Add capabilities by clicking the "+" button above.

Identity and Type

- Name: CloudKit-1
- Location: Absolute
- CloudKit-1.xcodeproj
- Full Path: /Users/tabinkowski/Google Drive/g-Teaching/uchicago.cloud/mcps51033-2019-autumn/mpcs51033-2017-autumn-code-samples/CloudKit-1/CloudKit-1.xcodeproj

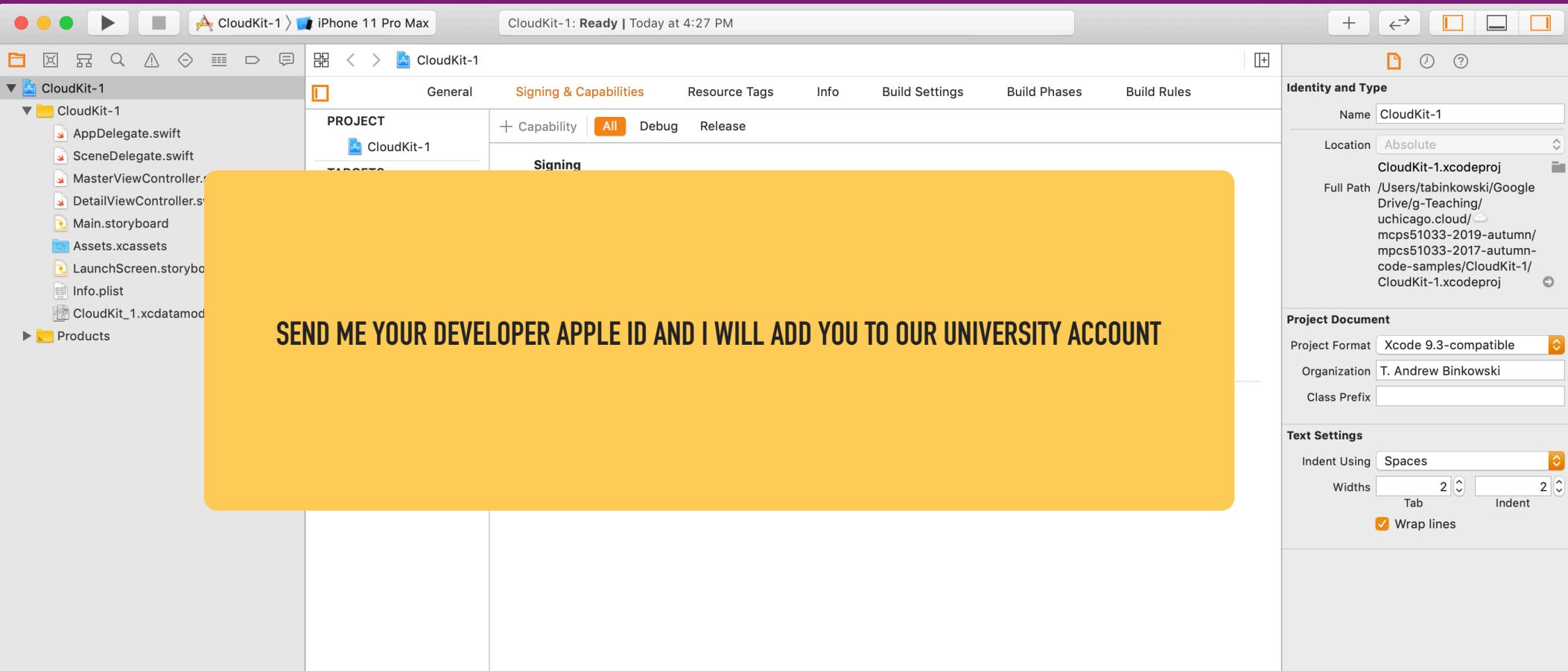
Project Document

- Project Format: Xcode 9.3-compatible
- Organization: T. Andrew Binkowski
- Class Prefix:

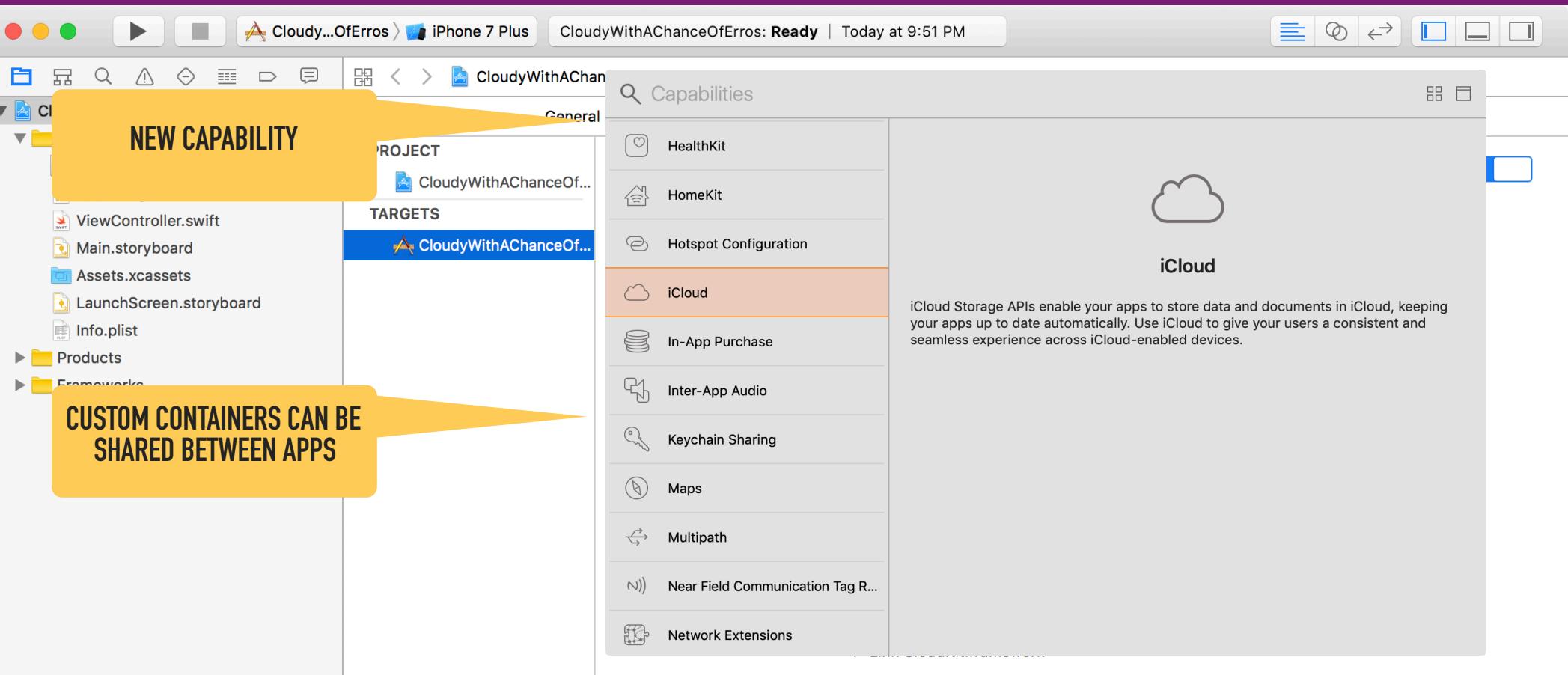
Text Settings

- Indent Using: Spaces
- Widths: Tab 2 | Indent 2
- Wrap lines:

ENABLING CLOUDKIT IN YOUR APPLICATION



ENABLING CLOUDKIT IN YOUR APPLICATION



ENABLING CLOUDKIT IN YOUR APPLICATION

The screenshot shows the Xcode interface for managing CloudKit configurations. On the left, there's a sidebar with files like `CloudKit.storyboard`, `Info.plist`, `CloudKit_1.xcdatamodeld`, and a `Products` folder. The main area has a yellow callout bubble containing the text "NEW CAPABILITY". Below it, another yellow callout bubble contains the text "CUSTOM CONTAINERS CAN BE SHARED BETWEEN APPS".

At the top right, there are sections for "Provisioning Profile" (Xcode Managed Profile), "Signing Certificate" (Apple Development), and "Status". The "Status" section displays three error messages:

- Failed to register bundle identifier. The app identifier "mobi.uchicago.CloudKit-1" cannot be registered to your development team. Change your bundle identifier to a unique string to try again. [Try Again](#)
- Provisioning profile "iOS Team Provisioning Profile: **" doesn't support the iCloud capability.
- Provisioning profile "iOS Team Provisioning Profile: **" doesn't include the com.apple.developer.icloud-container-identifiers entitlement.

Below these errors, there are sections for "Services" (checkboxes for Key-value storage, iCloud Documents, and CloudKit), "Containers" (checkboxes for iCloud.CloudKitOperation, iCloud.CloudKitOperation.TodayView, and iCloud.CloudKitOperation.photoShare), and a "CloudKit Dashboard" button.

ENABLING CLOUDKIT IN YOUR APPLICATION

The screenshot shows the Xcode Capabilities tab for a target named "CloudKit-1". The tab is divided into several sections:

- Background Modes**: A list of modes with checkboxes:
 - Audio, AirPlay, and Picture in Picture
 - Location updates
 - Voice over IP
 - External accessory communication
 - Uses Bluetooth LE accessories
 - Acts as a Bluetooth LE accessory
 - Background fetch
 - Remote notifications
 - Background processing
- iCloud**: A list of services and containers:
 - Services**:
 - Key-value storage
 - iCloud Documents
 - CloudKit
 - Containers**:
 - iCloud.mobi.uchicago.cloudkittest
 - iCloud.CloudKitOperation
 - iCloud.CloudKitOperation.TodayView
- Push Notifications**: A section with a checkbox and a "CloudKit Dashboard" button.

Two yellow callout boxes with the text "NEW CAPABILITY" point to the "Background Modes" and "iCloud" sections.

ENABLING CLOUDKIT IN YOUR APPLICATION

The screenshot shows the CloudKit Dashboard interface. At the top, there's a navigation bar with the title "CloudKit Dashboard" and user information "Andrew Binkowski". Below the navigation bar is the main content area.

The main content area has a large heading "iCloud.mobi.uchicago.cloudkittest". Below it is a section titled "Container Permissions".

The dashboard is divided into two main sections: "Development" and "Production".

Development: This section is currently active. It contains several buttons: "Data", "Telemetry", "Logs", "Schema", "Usage", and "API Access". A yellow arrow points from the "Schema" button towards a yellow callout box containing the text "DEFAULT CONTAINER IS CREATED ON ICLOUD".

Production: This section is shown below the Development section. It contains the text "Schema for production use. You can add and modify but not delete record types in the production" and a "Logs" button.

ENABLING CLOUDKIT IN YOUR APPLICATION

- Dashboard is only way to access permissions
- Data can be created, manipulated
- View analytic information about your data, users and operations

iCloud.mobi.uchicago.cloudkittest

 Container Permissions

Development

Build and test your schema. You can add, modify, and delete record types in the development environment.



Data



Telemetry



Logs



Schema



Usage



API Access

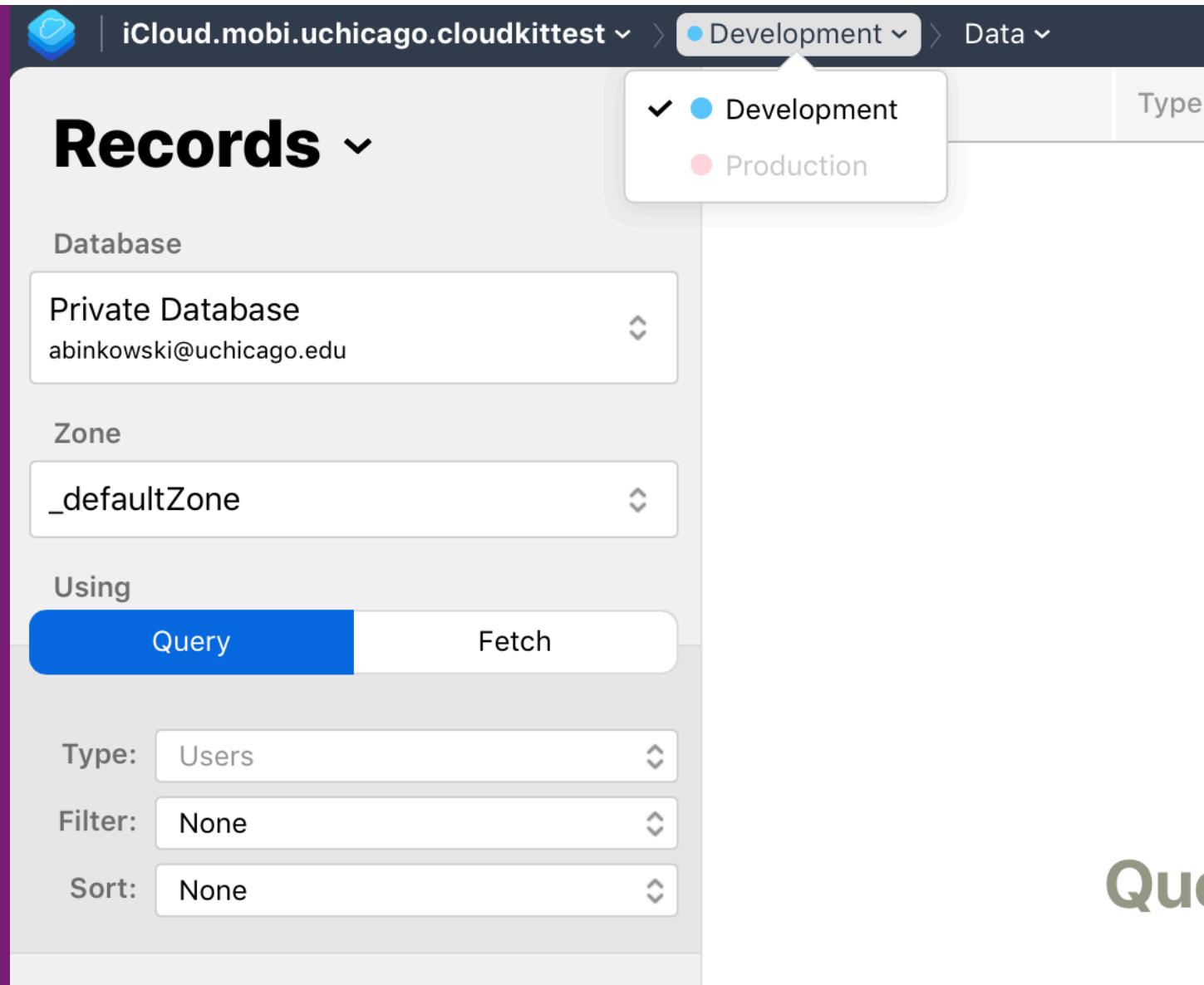
Production

Schema for production use. You can add and modify but not delete record types in the production environment.

This container has not been deployed to production.

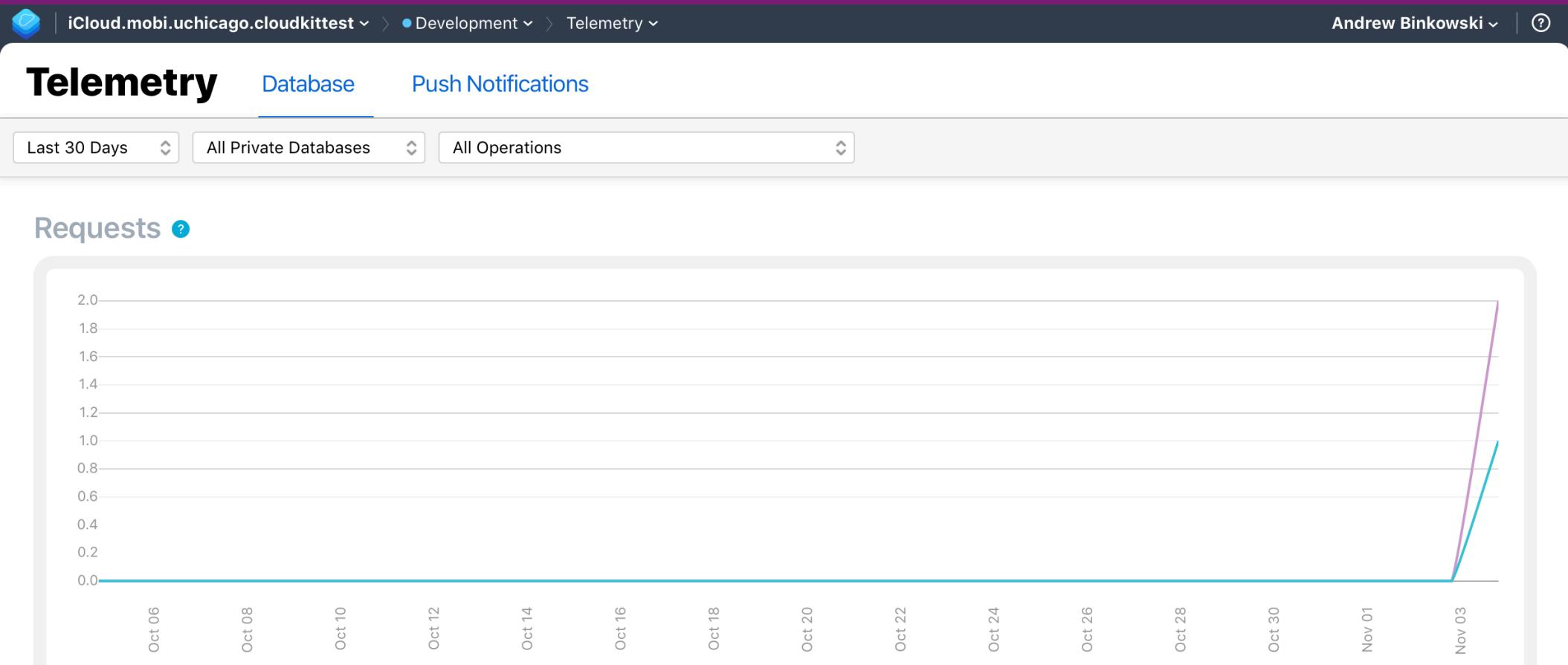
ENABLING CLOUDKIT IN YOUR APPLICATION

- "Flip a switch" to go to production mode



The screenshot shows the CloudKit Records interface. At the top, the URL is `iCloud.mobi.uchicago.cloudkittest`, followed by dropdown menus for `Development` and `Data`. A tooltip indicates that `Development` is selected. Below this, the main title is **Records**. The interface includes sections for **Database** (set to `Private Database` and `abinkowski@uchicago.edu`), **Zone** (`_defaultZone`), and **Using** (`Query` is selected, while `Fetch` is also available). Below these are filters for **Type** (`Users`), **Filter** (`None`), and **Sort** (`None`). The bottom right corner of the slide has the word **Query**.

ENABLING CLOUDKIT IN YOUR APPLICATION

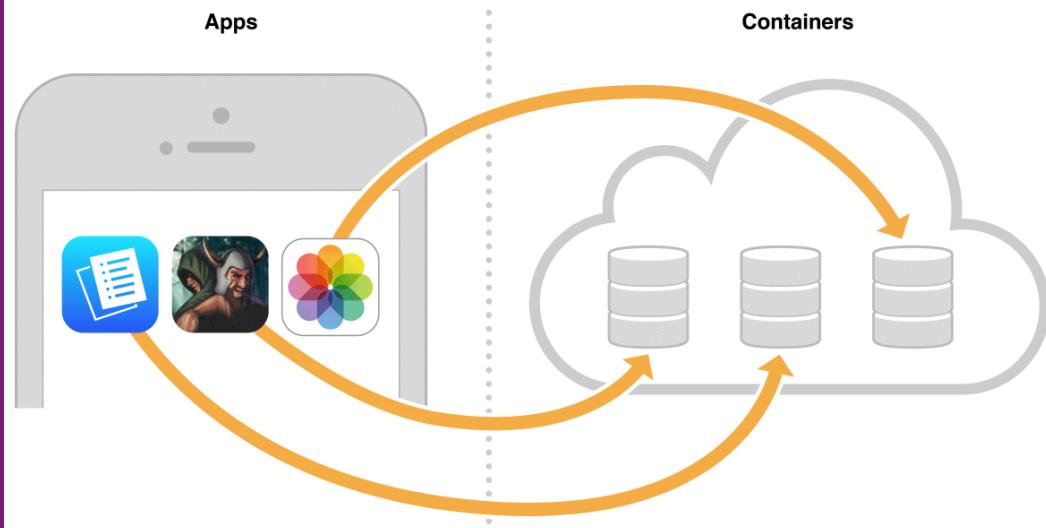


CONTAINER

CLOUDKIT OBJECTS

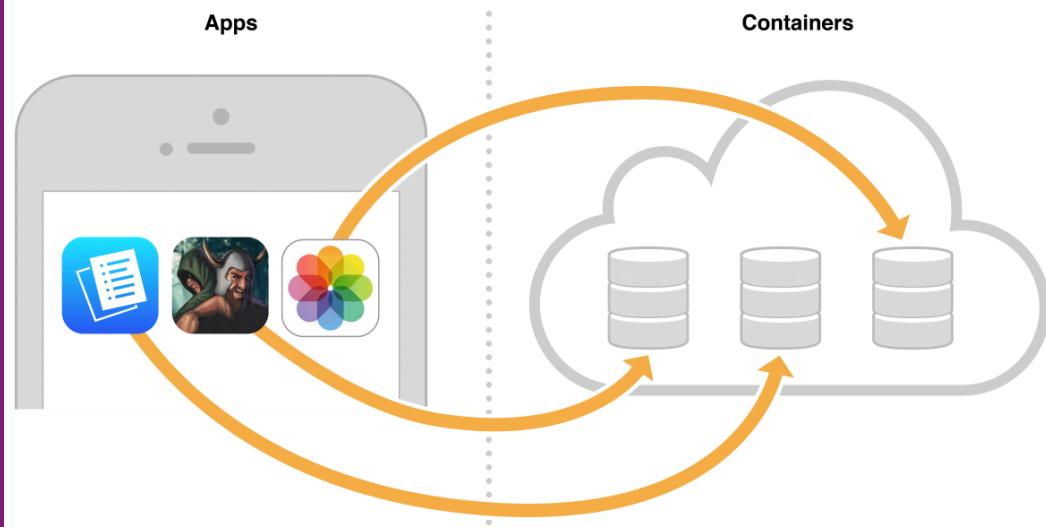
CONTAINER

- CKContainer
 - One default container per application
 - Data segregation
 - User encapsulation
 - Managed by the developer via portal



CLOUDKIT OBJECTS CONTAINER

- CKContainer can be shared between apps from the same developer
- You can create additional custom containers for your app



CLOUDKIT CONTAINER

```
import CloudKit  
  
let container: CKContainer = CKContainer.default()
```

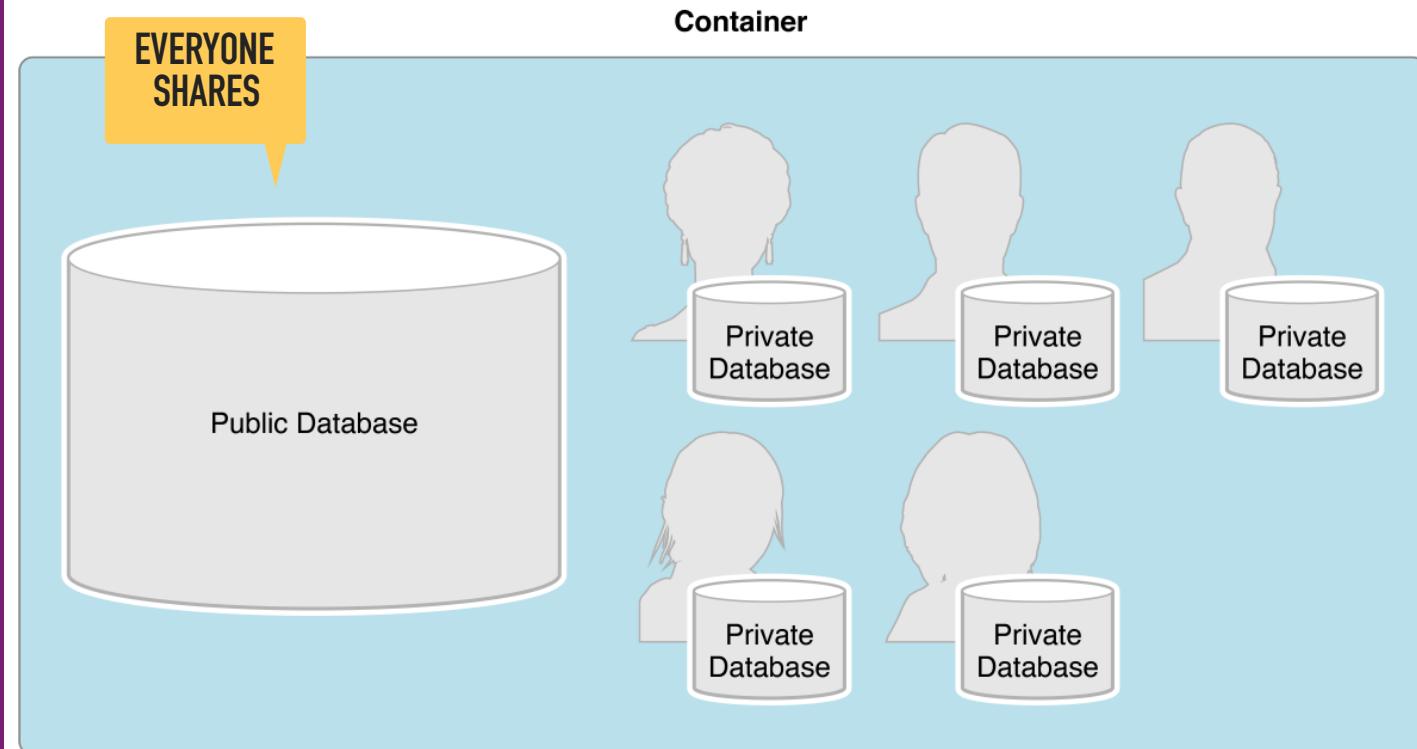
IT'S JUST THAT EASY

DATABASE

CLOUDKIT OBJECTS

DATABASE

- CKDatabase
 - Public
 - Private
 - Shared



EVERY USER HAS
THEIR OWN

CLOUDKIT OBJECTS

DATABASE

```
// Container
let container: CKContainer = CKContainer.default()

// Databases
let publicDB: CKDatabase = CKContainer.default().publicCloudDatabase
let privateDB: CKDatabase = CKContainer.default().privateCloudDatabase
```

CLOUDKIT OBJECTS

DATABASE

	Public Database	Private Database
Data Type	Shared Data	Current User's Data
Account	Required for Writing	Required
Quota	Developer	User
Default Permissions	World Readable	User Readable
Editing Permissions	iCloud Dashboard Roles	N/A

CLOUDKIT OBJECTS

DATABASE

	Public Database	Private Database
Data Type	Shared Data	Current User's Data
Account	Required for Writing	Required
Quota	Developer	User
Default Permissions	World Readable	User Readable
Editing Permissions	iCloud Dashboard Roles	N/A 

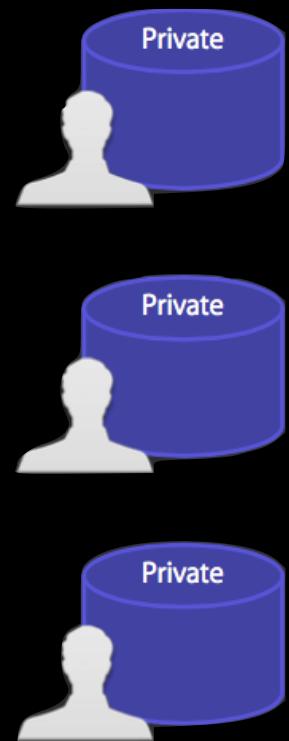
CLOUDKIT OBJECTS

DATABASE

- Public databases
 - No atomic writes
 - No delta downloads
 - Cross zone references

Public Database

CloudKit Container



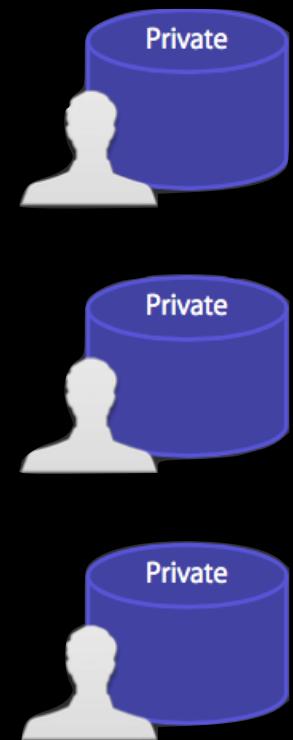
CLOUDKIT OBJECTS

DATABASE

- Private databases
 - Atomic writes
 - Delta downloads
 - No cross-zone references

Public Database

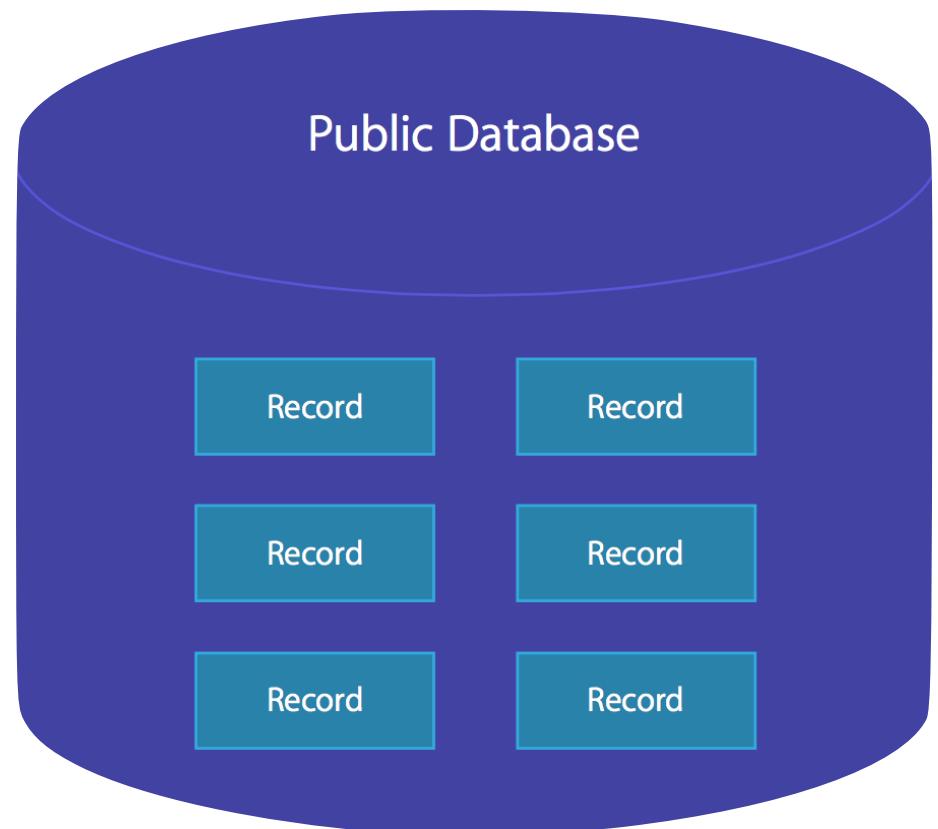
CloudKit Container



CLOUDKIT OBJECTS

DATABASE

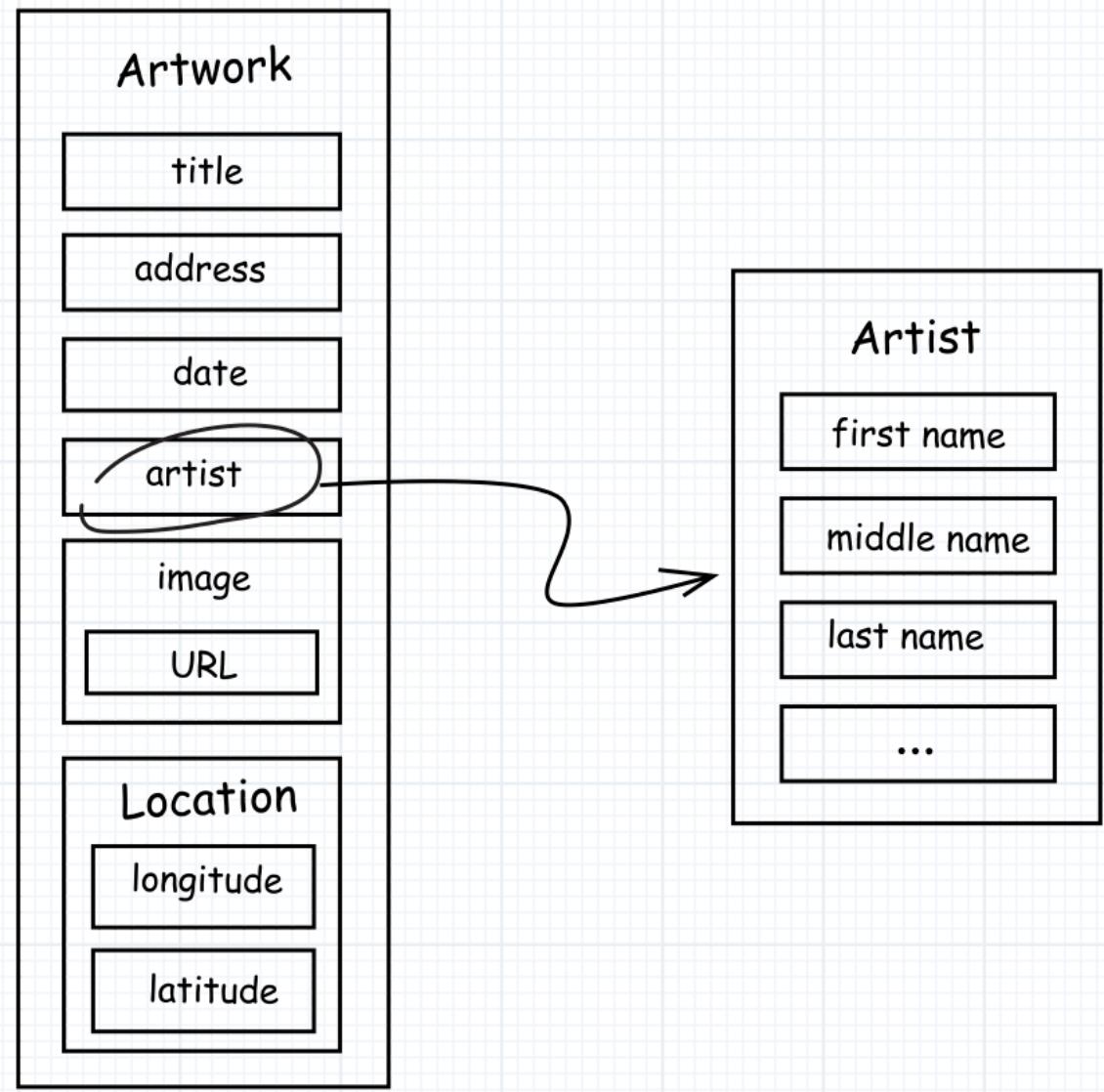
- Structured Data
 - Wraps key/value pairs
 - Types and references
 - Just-in-time schema
 - Metadata
 - Created, edited, etc.



CLOUDKIT OBJECTS

DATABASE

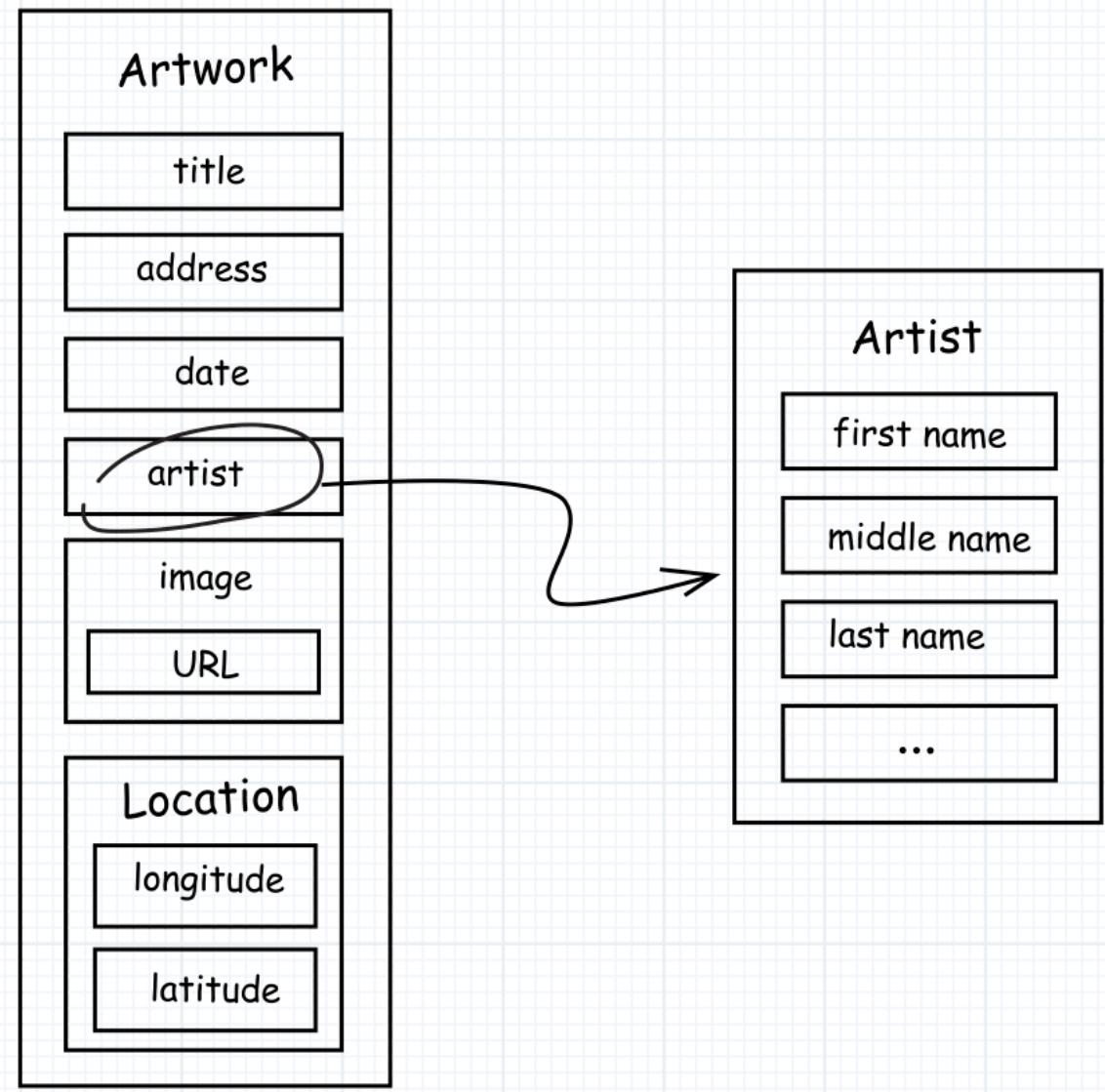
- Schema is built on-demand
- You can edit it in the console



CLOUDKIT OBJECTS

DATABASE

- Changing it may be painful
 - Requires coordination between device and console

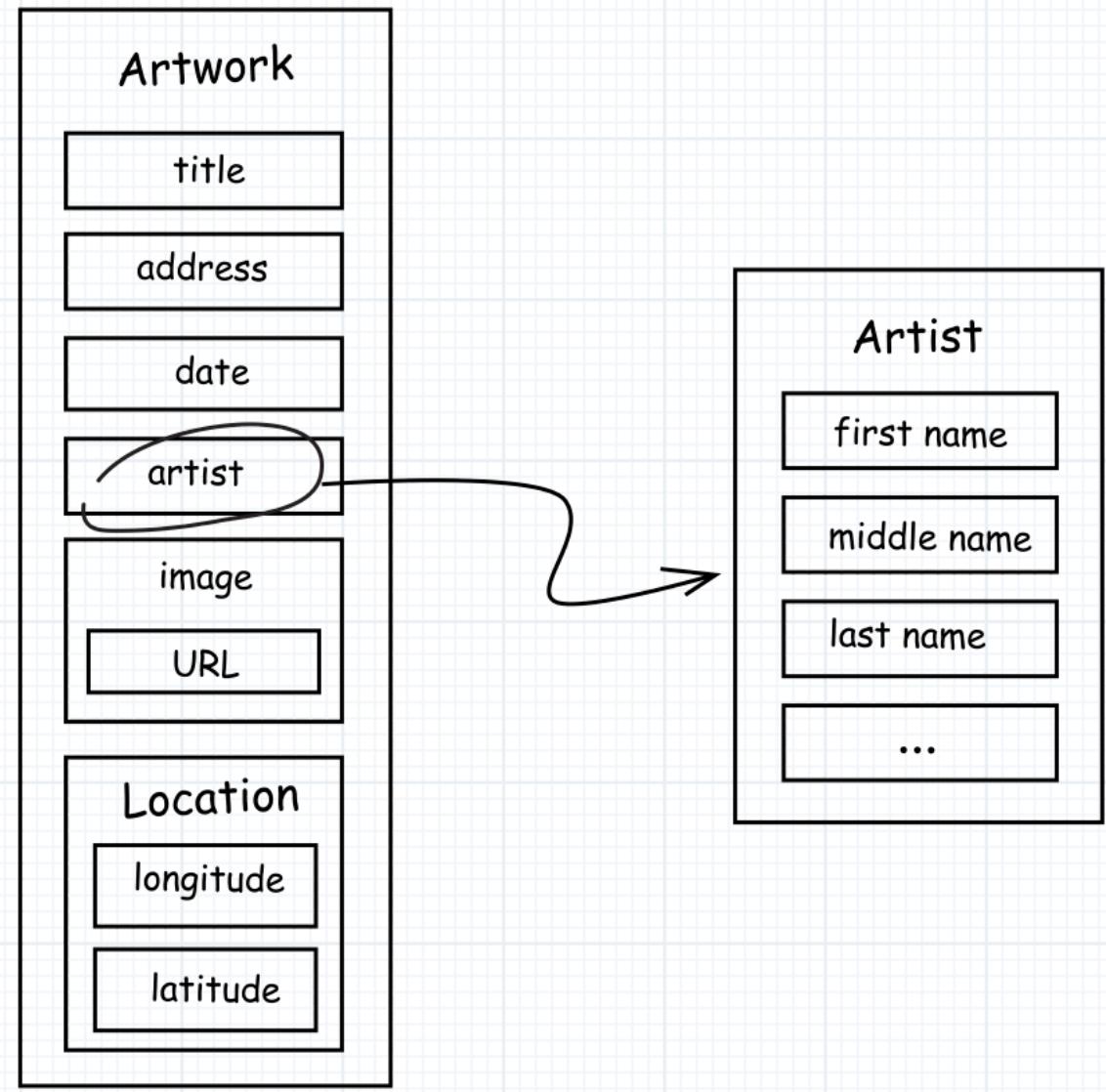


RECORDS

CLOUDKIT OBJECTS

RECORDS

- Record Values
 - String
 - Number
 - Data
 - Date
 - CLLocation
 - CKReference
 - CKAsset
 - Arrays of the above

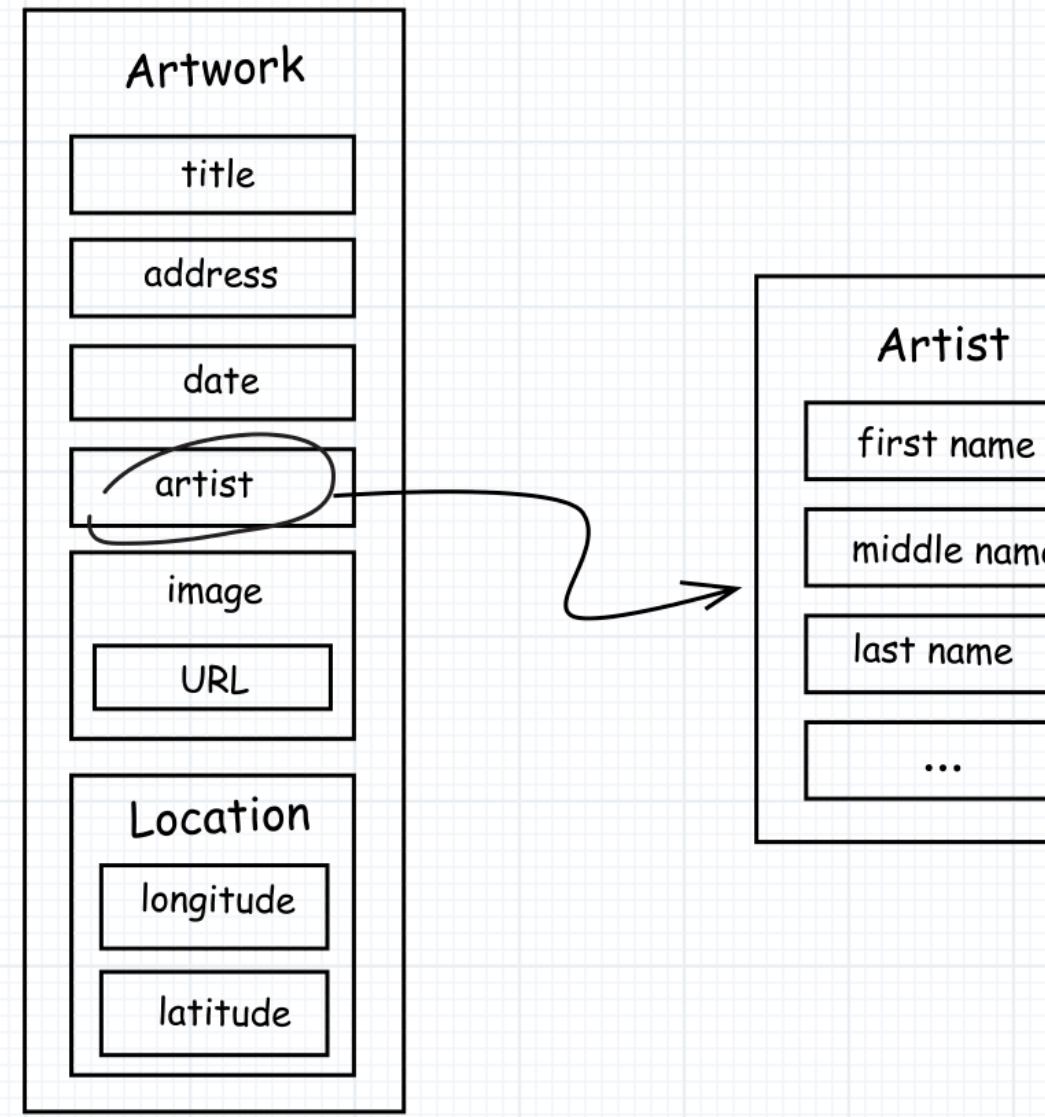


CLOUDKIT OBJECTS

RECORDS

- Records are uniquely stored by a record identifier made up of a combination of the "Zone+Name"

SIMILAR TO APP ENGINE PARENT KEY



CLOUDKIT OBJECTS RECORDS

- Meta data is automatically assigned to all records
- Used for many tasks

```
/* These create the record in the default zone. */
public init(recordType: String)

public init(recordType: String, recordID: CKRecordID)

public init(recordType: String, zoneID: CKRecordZoneID)

open var recordType: String { get }

@NSCopying open var recordID: CKRecordID { get }

/* Change tags are updated by the server to a unique value every time a record is modified.
A different change tag necessarily means that the contents of the record are different. */
open var recordChangeTag: String? { get }

/* This is a User Record recordID, identifying the user that created this record. */
@NSCopying open var creatorUserRecordID: CKRecordID? { get }

open var creationDate: Date? { get }

/* This is a User Record recordID, identifying the user that last modified this record. */
@NSCopying open var lastModifiedUserRecordID: CKRecordID? { get }

open var modificationDate: Date? { get }

/*
In addition to objectForKey: and setObject:forKey:, dictionary-style subscripting (record[key] and record[key] = value)
used to get and set values.
Acceptable value object classes are:
CKReference
CKAsset
CLLocation
NSData
NSDate
NSNumber
NSString
NSArray containing objects of any of the types above

Any other classes will result in an exception with name NSInvalidArgumentException.

Derived field keys are prefixed with '_'. Attempting to set a key prefixed with a '_' will result in an error.

Key names roughly match C variable name restrictions. They must begin with an ASCII letter and can contain ASCII
letters and numbers and the underscore character.
The maximum key length is 255 characters.
*/
open func objectForKey(key: String) -> CKRecordValue?
```

CLOUDKIT OBJECTS

RECORDS

The screenshot shows the CloudKit Records interface. On the left, there's a sidebar with filters for Database (Public Database), Zone (_defaultZone), Using (Query selected), Type (joke), Filter (None), and Sort (None). At the bottom is a blue "Query Records" button. The main area displays a table of records with columns: Name, T..., Fields, C, Cre..., and Mod.... Two records are visible: one named OFE... and another named AA6... Both records have a value of 4 in the Fields column. A yellow callout box with the text "METADATA ON EVERY RECORD" points to the first record's row. To the right, a detailed view of the AA6... record is shown under "Editing Record". The record has the following metadata:

Name:	AA672AF9-673D-FEC2-A29E-9A0FE9C172BC
Type:	joke
Database:	public
Zone:	_defaultZone
Created:	Nov 4 2019 9:14 PM by Andrew Binkowski (_8a76cb86f6390ecf95f548796a270cc8)
Modified:	Nov 4 2019 9:14 PM by Andrew Binkowski (_8a76cb86f6390ecf95f548796a270cc8)
Change Tag:	k2la507p

Below the metadata, there are sections for References (None) and Custom Fields. The custom field "question" has a value of "test".

CLOUDKIT OBJECTS

RECORDS

iCloud.mobi.uchicago.cloudkittest ▾ Development ▾ Schema ▾ Andrew Binkowski ▾ ?

Security Roles ▾

Default Roles

World

Authenticated

Creator

Custom Roles

Security roles allow you to control permissions in the public database.

 Create New Role  Delete Role

Record Type	Permissions
Users	Create <input checked="" type="checkbox"/> Read <input checked="" type="checkbox"/> Write

NO WRITE BY DEFAULT

CLOUDKIT OBJECTS

RECORDS

CloudKit Schema Editor

Navigation: iCloud.mobi.uchicago.twenty-nineteen-cloudkit > Development > Schema

User: Andrew Binkowski

Record Types

- System Types
- Users
- Custom Types
 - joke

New Type **Delete Type**

Field Name	Field Type	Indexes
System Fields		
recordName	Reference	Queryable
createdBy	Reference	Queryable
modifiedAt	Date/Time	Queryable
modifiedBy		None
modifiedAt		None
changeTag		None
Custom Fields		
question		Queryable, Searchable, Sort...
rating_negative	Int(64)	Queryable, Sortable
rating_positive	Int(64)	Queryable, Sortable
response	String	Queryable, Searchable, Sort...

RECORDS ARE DEFINED PROGRAMMATICALLY OR IN THE CONSOLE

CLOUDKIT OBJECTS

RECORDS

```
/// Create a joke record and save to iCloud using CloudKit
/// - parameter joke: Funny string
/// - remark: Error handling leaves something to be desired
func saveJoke(_ joke: String) {

    let record = CKRecord(recordType: "joke")
    record.setValue(joke, forKey: "question")
    record["response"] = "To get to the other side." as CKRecordValue
    record["rating_positive"] = 0 as CKRecordValue
    record["rating_negative"] = 0 as CKRecordValue

    publicDB.save(record) { (record, error) in
        if let error = error {
            print("Error: \(error.localizedDescription)")
            return
        }
        print("Saved record: \(record.debugDescription)")
    }
}
```

CREATE A RECORD

CLOUDKIT OBJECTS

RECORDS

```
/// Create a joke record and save to iCloud using CloudKit
/// - parameter joke: Funny string
/// - remark: Error handling leaves something to be desired
func saveJoke(_ joke: String) {

    let record = CKRecord(recordType: "joke")
    record.setValue(joke, forKey: "question")
    record["response"] = "To get to the other side." as CKRecordValue
    record["rating_positive"] = 0 as CKRecordValue
    record["rating_negative"] = 0 as CKRecordValue

    publicDB.save(record) { (record, error) in
        if let error = error {
            print("Error: \(error.localizedDescription)")
            return
        }
        print("Saved record: \(record.debugDescription)")
    }
}
```

SAVE THE
RECORD

CLOUDKIT OBJECTS RECORDS

- Create a Joke record in our view controller
- Data is key-value pairs

```
// MARK: - CloudKit
let container: CKContainer = CKContainer.default()
let publicDB: CKDatabase = CKContainer.default().publicCloudDatabase
let privateDB: CKDatabase = CKContainer.default().privateCloudDatabase

/// Create a joke record and save to iCloud using CloudKit
/// - parameter joke: Funny string
/// - remark: Error handling leaves something to be desired
func saveJoke(_ joke: String) {
    let record = CKRecord(recordType: "joke")
    record.setValue(joke, forKey: "question")
    record["response"] = "To get to the other side." as CKRecordValue

    publicDB.save(record) { (record, error) in
        if let error = error {
            print("🐞: \(error.localizedDescription)")
            return
        }
        print("Saved record: \(record.debugDescription)")
    }
}
```

CLOUDKIT OBJECTS

RECORDS

```
let container: CKContainer = CKContainer.default()
let publicDB: CKDatabase = CKContainer.default().publicCloudDatabase
let privateDB: CKDatabase = CKContainer.default().privateCloudDatabase

override func viewDidAppear(_ animated: Bool) {
    saveJoke("Why did the chicken cross the road?")
}

/// Create a joke record and save to iCloud using CloudKit
/// - parameter joke: Funny string
/// - remark: Error handling leaves something to be desired
func saveJoke(_ joke: String) {
    let record = CKRecord(recordType: "joke")
    record.setValue(joke, forKey: "text")
    publicDB.save(record) { (record, error) in
        if let error = error {
            print("🐞: \(error.localizedDescription)")
            return
        }
        print("Saved record: \(record.debugDescription)")
    }
}

🐞: This request requires an authenticated account
```

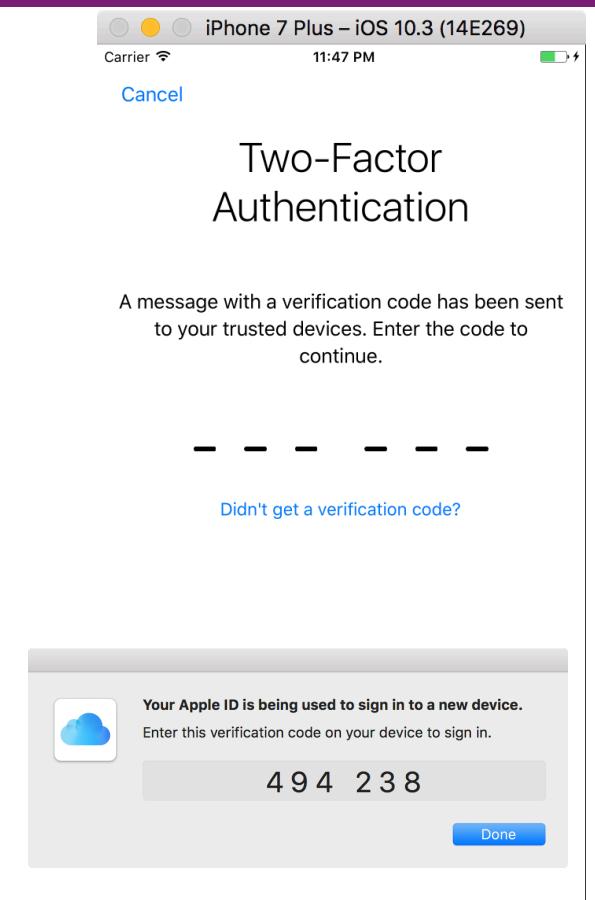
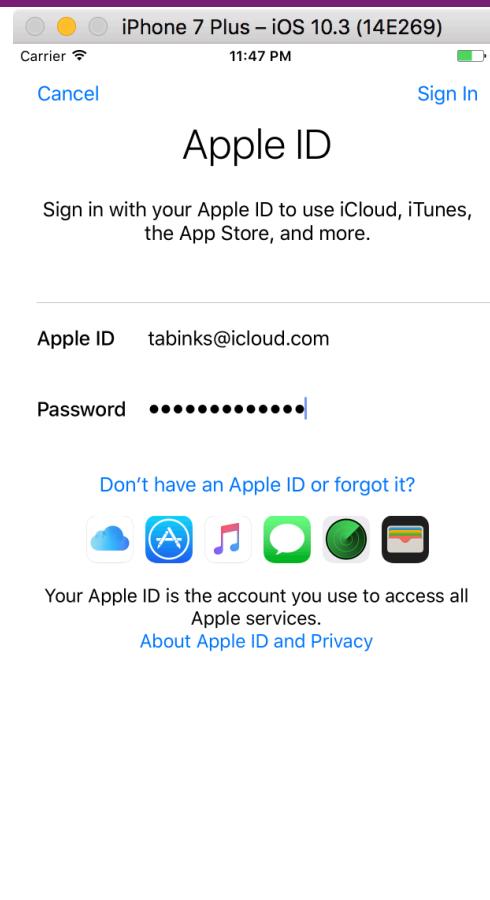
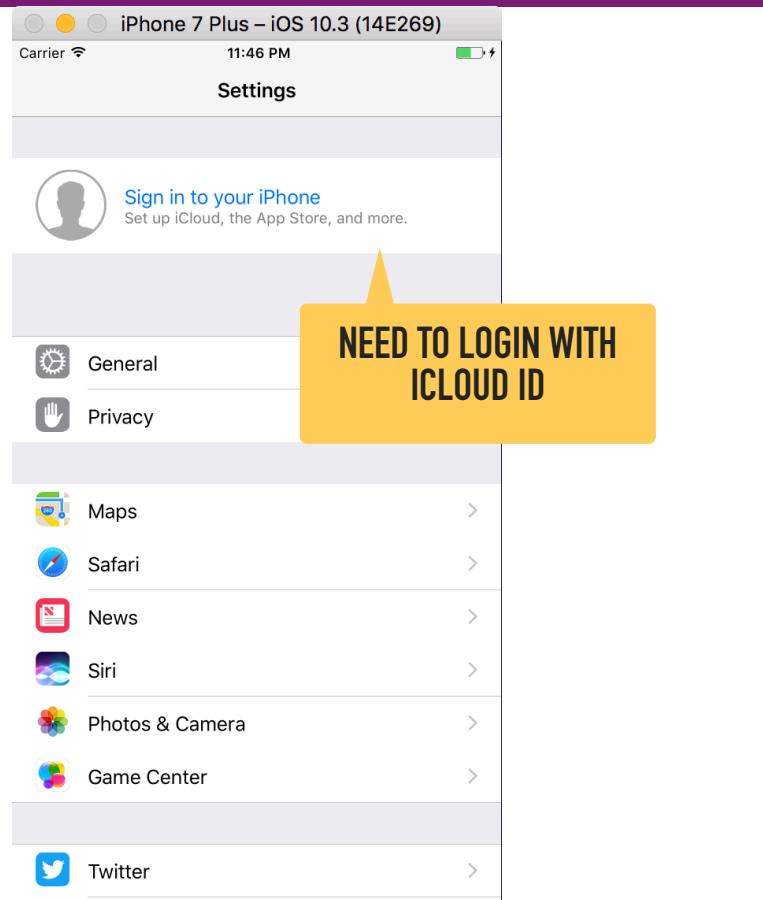
YOUR FIRST CLOUDKIT
ERROR



CLOUDKIT OBJECTS

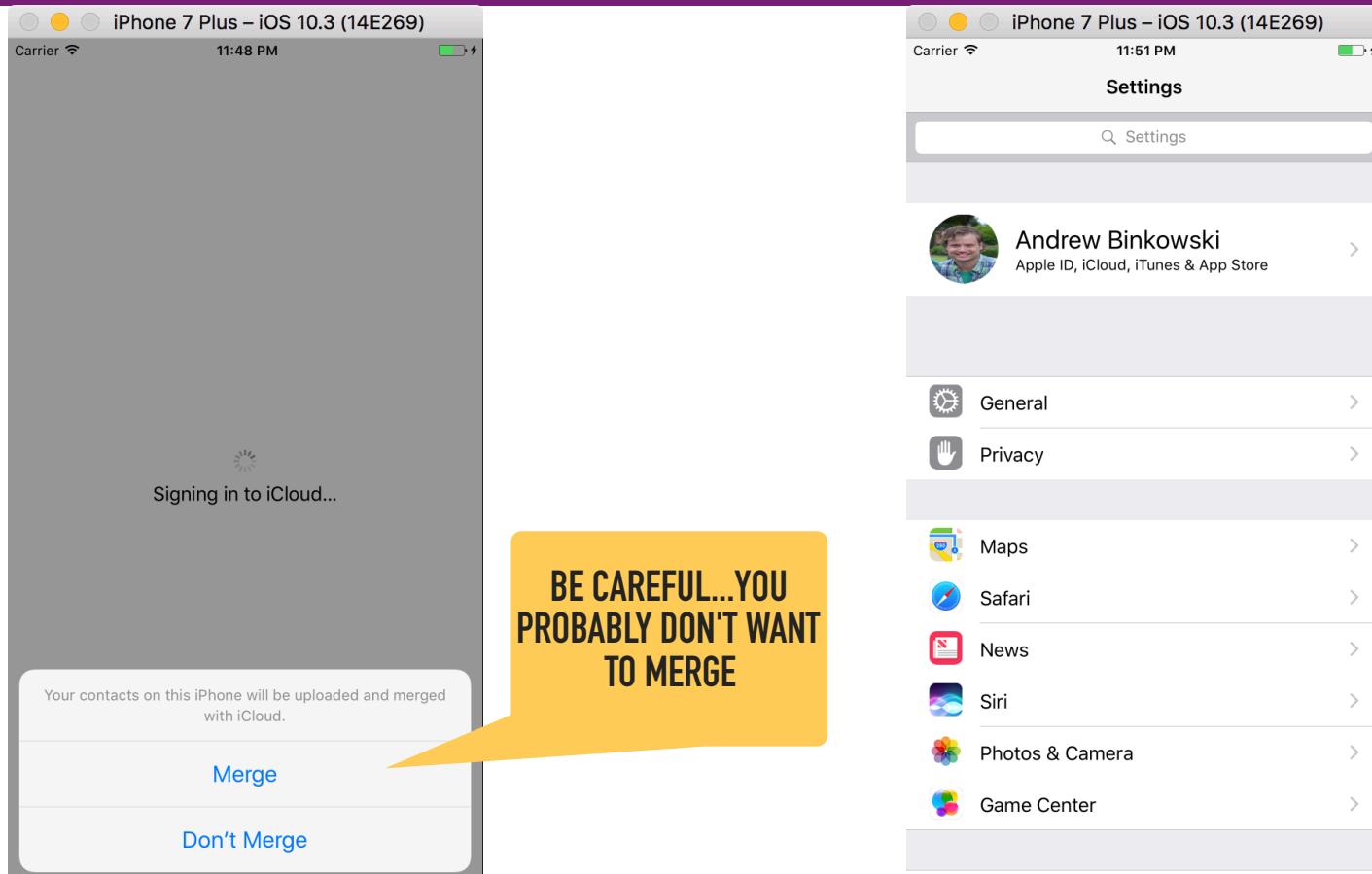
RECORDS

FULLY SUPPORTED BY
THE SIMULATOR



CLOUDKIT OBJECTS

RECORDS

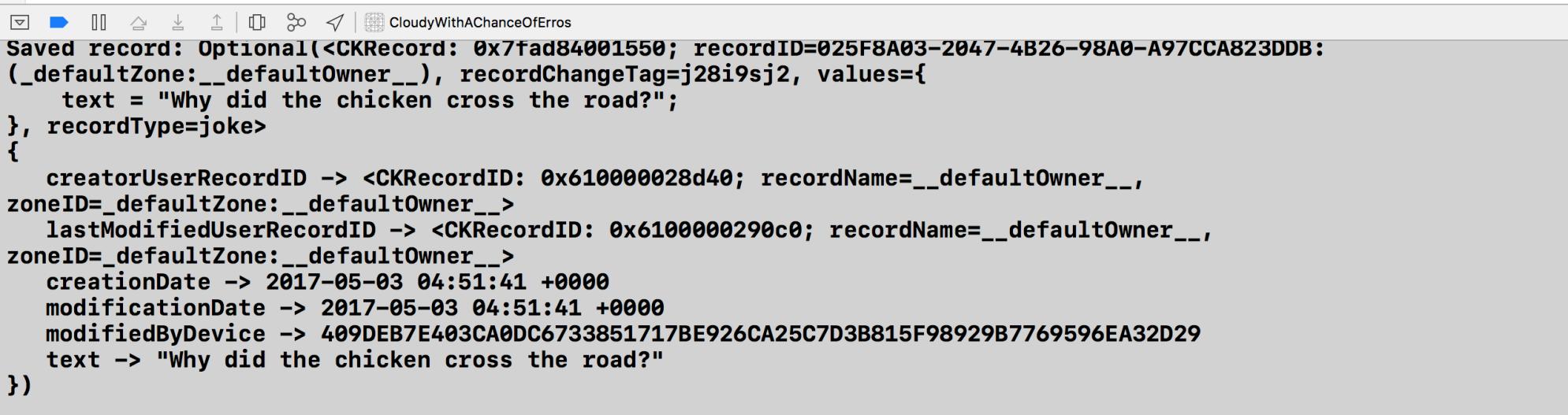


CLOUDKIT OBJECTS

RECORDS

```
let record = CKRecord(recordType: "joke")
record.setValue(joke, forKey: "question")
record["response"] = "To get to the other side." as CKRecordValue

publicDB.save(record) { (record, error) in
    if let error = error {
        print("Error: \(error.localizedDescription)")
        return
    }
    print("Saved record: \(record.debugDescription)")
}
}
```



The screenshot shows a Xcode interface with a code editor and a terminal window. The code editor contains Swift code for saving a CloudKit record. The terminal window shows the output of running the code, which prints the saved record's debug description. The output includes the record ID, change tag, values (text), record type, and various metadata fields like creation and modification dates.

```
CloudyWithAChanceOfErrors
Saved record: Optional(<CKRecord: 0x7fad84001550; recordID=025F8A03-2047-4B26-98A0-A97CCA823DDB: {_defaultZone:__defaultOwner__}, recordChangeTag=j28i9sj2, values={ text = "Why did the chicken cross the road?"; }, recordType=joke>
{
    creatorUserRecordID -> <CKRecordID: 0x610000028d40; recordName=__defaultOwner__,
    zoneID=_defaultZone:__defaultOwner__
    lastModifiedUserRecordID -> <CKRecordID: 0x6100000290c0; recordName=__defaultOwner__,
    zoneID=_defaultZone:__defaultOwner__
    creationDate -> 2017-05-03 04:51:41 +0000
    modificationDate -> 2017-05-03 04:51:41 +0000
    modifiedByDevice -> 409DEB7E403CA0DC6733851717BE926CA25C7D3B815F98929B7769596EA32D29
    text -> "Why did the chicken cross the road?" })
```

CLOUDKIT OBJECTS

RECORDS

Records ▾

Database
Public Database

Zone
_defaultZone

Using
Query Fetch Changes

Type: joke

Filter: None

Sort: None

Query Records

Query for records of the type "joke" that are in the "_defaultZone" zone of the "public" database.

Name	Type	Fields	Ch. Tag	Created	Modified
▶ OFEB50A6-282E-4...	joke	4: question, response, ...	k2lacelf	2019/11/04, 21:20	2019/11/04, 21:20
▶ AA672AF9-673D-F...	joke	4: question, response, ...	k2la5...	2019/11/04, 21:14	2019/11/04, 21:14

CLOUDKIT OBJECTS

RECORDS

CloudKit | iCloud.mobi.uchicago.twenty-nineteen-cloudkit | Development | Schema | Andrew Binkowski | ?

Record Types ▾

System Types
Users
Custom Types
joke

New Type **Delete Type**

Field Name	Field Type	Indexes
System Fields		
recordName	Reference	Queryable
createdBy	Reference	Queryable
createdAt	Date/Time	Queryable
modifiedBy	Reference	None
modifiedAt	Date/Time	None
changeTag	String	None
Custom Fields		
question		Queryable, Searchable, Sort...
rating_negative)	Queryable, Sortable
rating_positive	Int(64)	Queryable, Sortable
response	String	Queryable, Searchable, Sort...

CHANGE SCHEMA FROM CONSOLE OR IN CODE

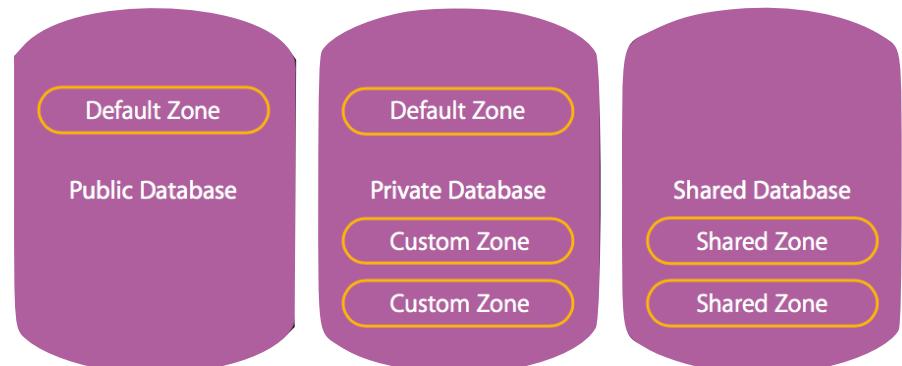
RECORD ZONES

CLOUDKIT OBJECTS

RECORD ZONES

- CKRecordZones
 - Zones are a useful way to arrange a discrete group of records
 - Supported only in Private and Shared database
 - Atomic group writes

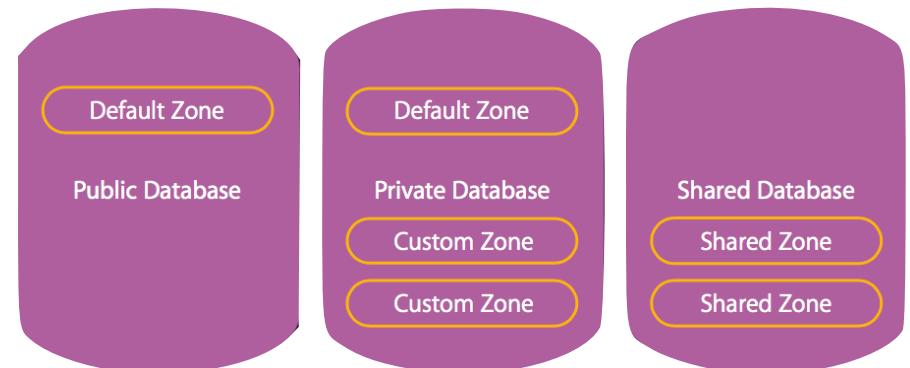
PUBLIC HAS A DEFAULT ZONE



CLOUDKIT OBJECTS

RECORD ZONES

- CKRecordZones
 - Zones are a useful way to arrange a discrete group of records
 - Supported only in Private and Shared database
 - Atomic group writes



This seems familiar

CLOUDKIT OBJECTS

RECORDS

- Created by the client
- They represent the location of the record
- External data set foreign key

```
//  
//  CKRecordID.h  
//  CloudKit  
//  
//  Copyright (c) 2014 Apple Inc. All rights reserved.  
  
#import <Foundation/Foundation.h>  
  
@class CKRecordZoneID;  
  
NS_CLASS_AVAILABLE(10_10, 8_0)  
@interface CKRecordID : NSObject <NSSecureCoding>  
  
- (instancetype)init NS_UNAVAILABLE;  
  
/* Record names must be 255 characters or less */  
/* This creates a record ID in the default zone */  
- (instancetype)initWithRecordName:(NSString *)name  
- (instancetype)initWithRecordName:(NSString *)name  
  
@property (nonatomic, readonly, strong) NSString *name;  
@property (nonatomic, readonly, strong) CKRecordZoneID *zone;  
  
@end
```

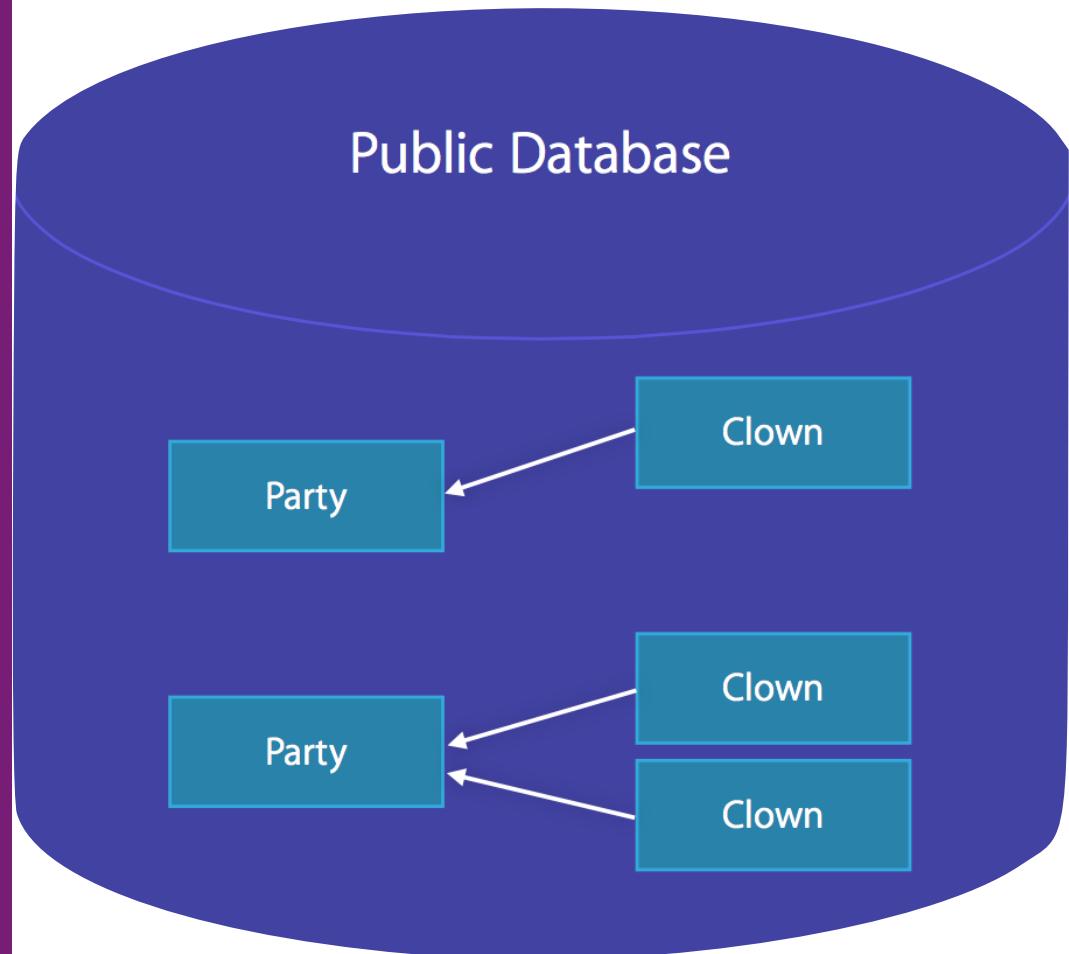
CKRECORDZONEID

REFERENCES

CLOUDKIT OBJECTS

RECORDS

- CKReference
 - Points to a another record
 - Server understands relationship (takes some responsibility for managing; if you want)

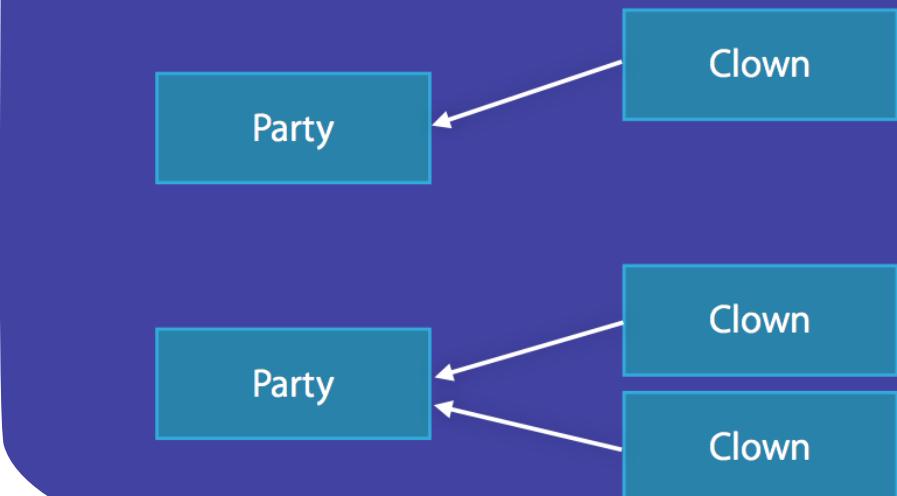


CLOUDKIT OBJECTS

RECORDS

- Cascade Deletes
 - What happens when you delete something with a reference?
 - Dangling Pointers
- Use "Back References" strategy

Public Database



CLOUDKIT API

```
// Create a reference with a record  
// Set the delete 'action' preference  
let reference = CKReference(recordID: recordID,  
                           action: .none)
```

WHAT HAPPENS ON DELETE

CLOUDKIT API

iCloud.mobi.uchicago.twenty-nineteen-cloudkit › Development › Schema › Binkowski › ?

Record Types ▾

System Types
Users
Custom Types
joke

New Type **Delete Type**

Field Name	Field Type	
System Fields		
recordName	Reference	Queryable
createdBy	Reference	Queryable
createdAt	Date/Time	Queryable
modifiedBy	Reference	None
modifiedAt	Date/Time	None
changeTag	String	None
Custom Fields		
question	String	Queryable, Searchable, Sort... -
rating_negative	Int(64)	Queryable, Sortable -
rating_positive	Int(64)	Queryable, Sortable -
response	String	Queryable, Searchable, Sort... -

REFERENCE

CLOUDKIT API

iCloud.mobi.uchicago.twenty-nineteen-cloudkit › Development › Schema › Andrew Binkowski › ?

Record Types ▾

System Types
Users
Custom Types
joke

New Type **Delete Type**

Field Name	Field Type	
System Fields		
recordName	Reference	Queryable
createdBy	Reference	Queryable
createdAt	Date/Time	Queryable
modifiedBy	Reference	None
modifiedAt	Date/Time	None
changeTag	String	None
Custom Fields		
question	String	Queryable, Searchable, Sort... -
rating_negative	Int(64)	Queryable, Sortable -
rating_positive	Int(64)	Queryable, Sortable -
response	String	Queryable, Searchable, Sort... -

USER REFERENCE TO RECORD IS BUILT IN

DATA MODELING

DATA MODELING

- How should 1 to many relationships be handled?

Album

PhotoArray

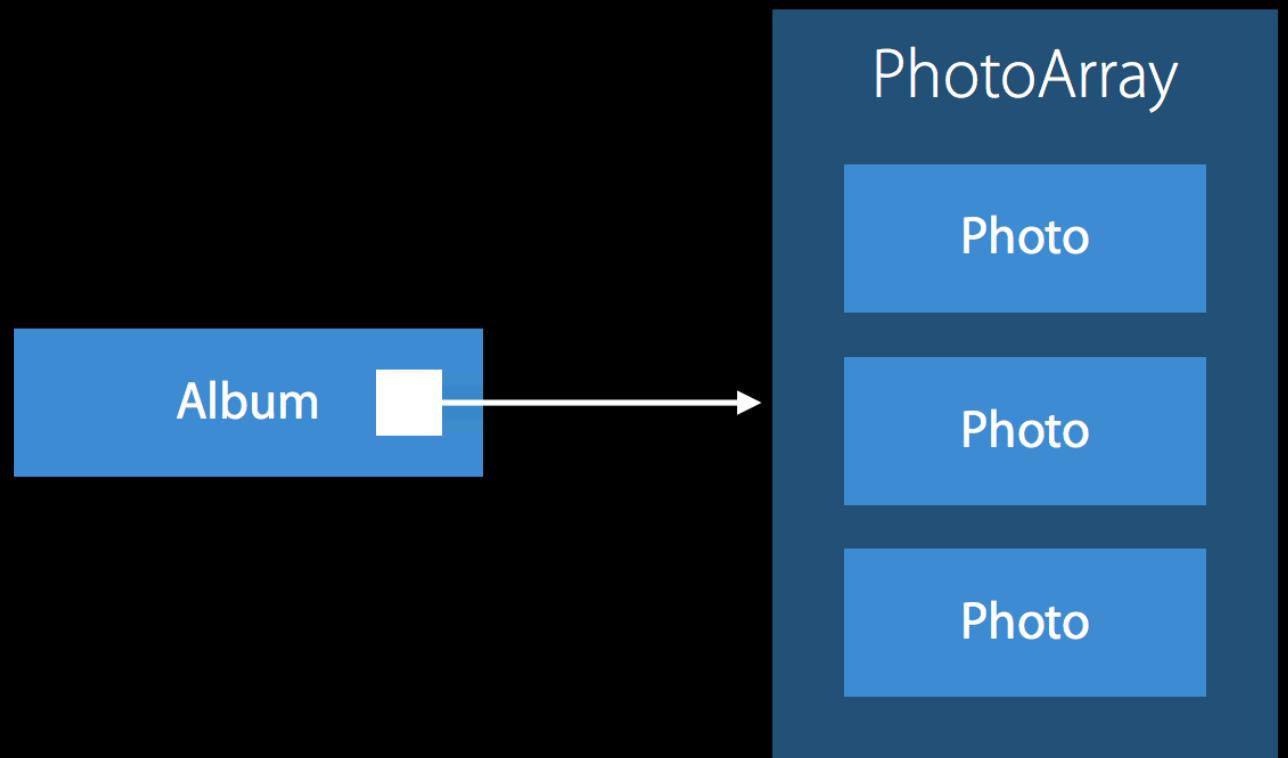
Photo

Photo

Photo

DATA MODELING

- Typical setup to model

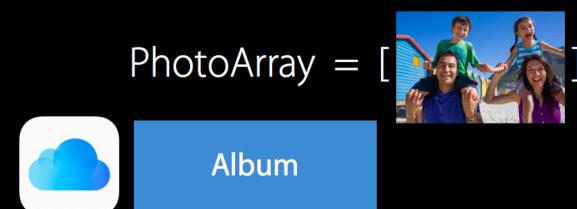


DATA MODELING

PhotoArray = []



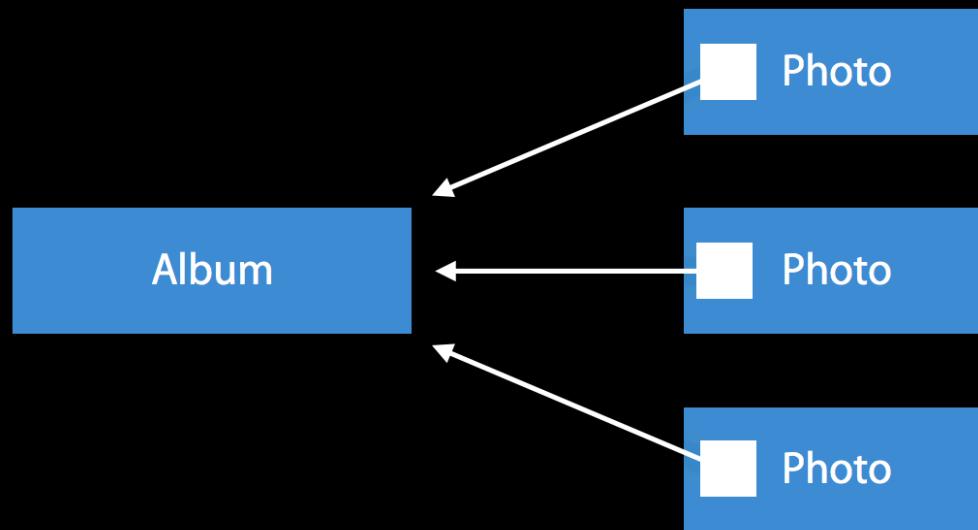
DATA MODELING



serverRecordChanged



DATA MODELING

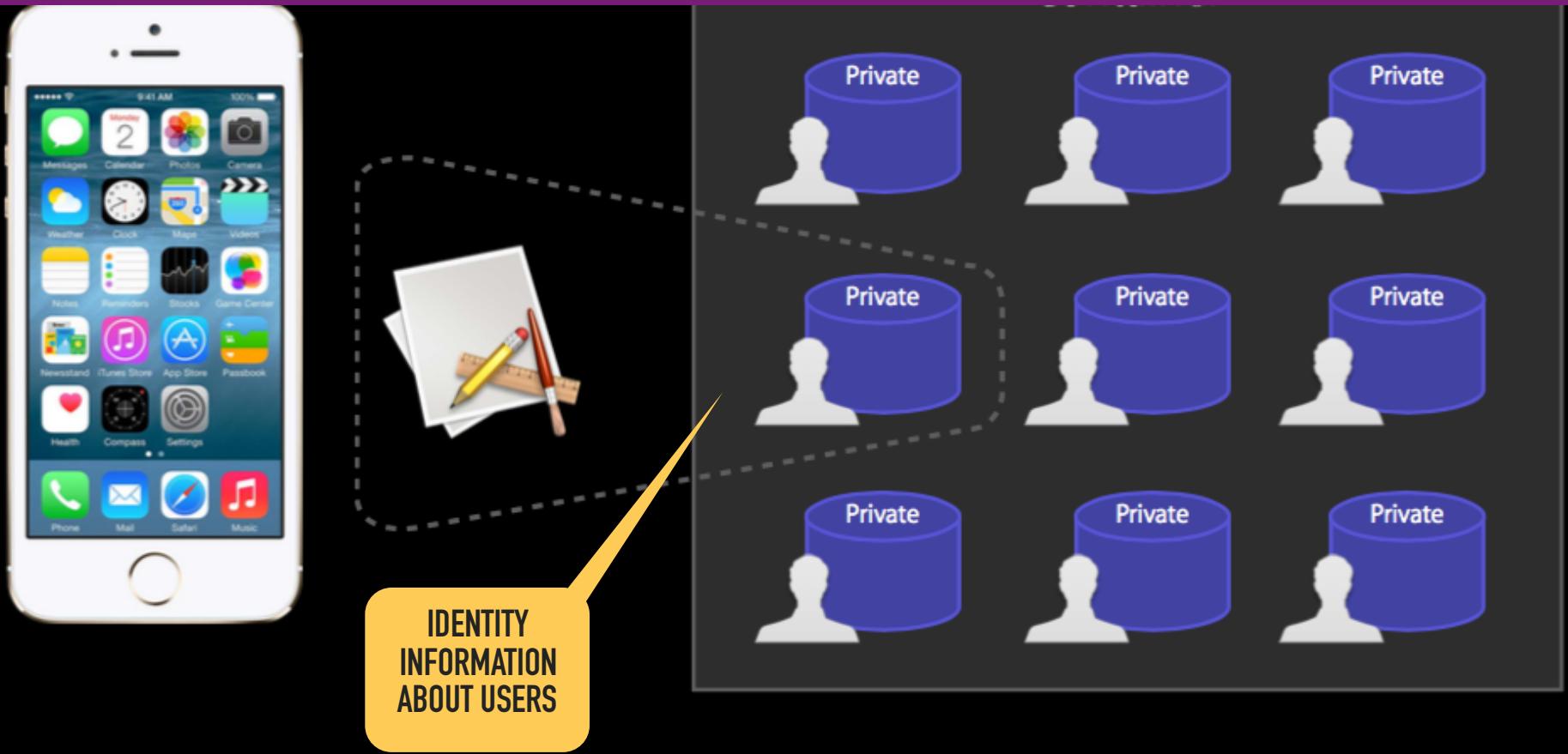


USER RECORDS

USER RECORDS

- Identity
- Metadata
- Privacy
- Discovery
 - Users can discover friends who use the application

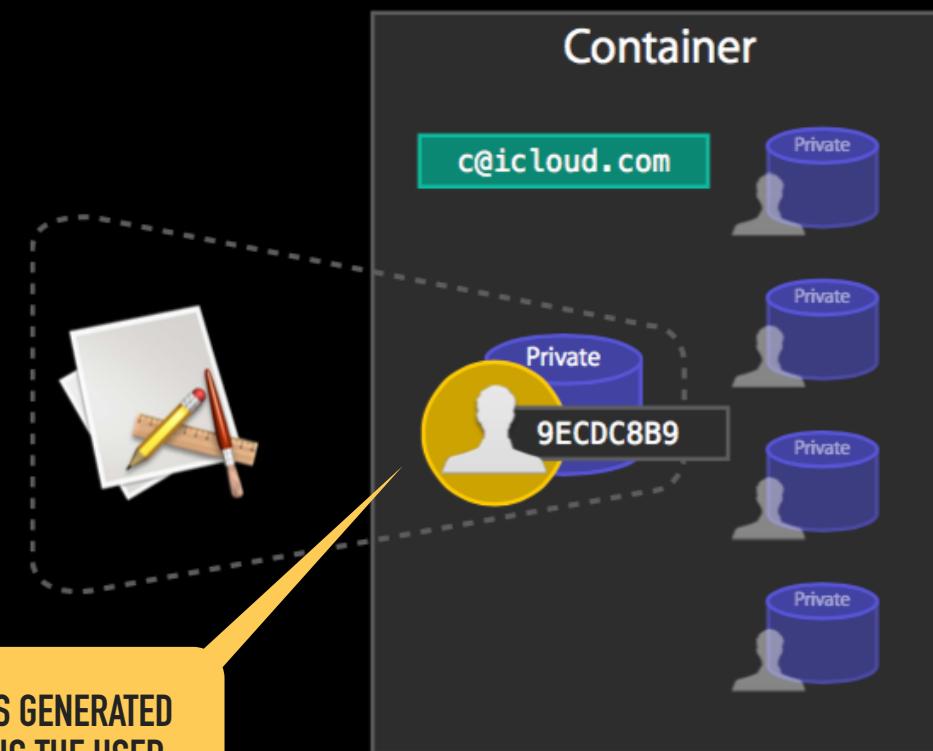
USER RECORDS



USER RECORDS



A UNIQUE ID IS GENERATED
REPRESENTING THE USER



- a@icloud.com
- b@icloud.com
- c@icloud.com
- d@icloud.com
- e@icloud.com

USER RECORDS

iCloud.mobi.uchicago.twenty-nineteen-cloudkit › Development › Schema › Andrew Binkowski › ?

Record Types ▾

System Types

Users

Custom Types

joke

USER RECORD IS ALREADY CREATED

Field Name	Field Type	Indexes
System Fields		
recordName	Reference	Queryable
createdBy	Reference	Queryable
createdAt	Date/Time	Queryable
modifiedBy	Reference	Queryable
modifiedAt	Date/Time	Queryable
changeTag	String	None
Custom Fields		
question	String	Queryable, Searchable, Sort...
rating_negative	Int(64)	Queryable, Sortable
rating_positive	Int(64)	Queryable, Sortable
response	String	Queryable, Searchable, Sort...

ADD ADDITIONAL FIELDS TO CUSTOMIZE YOUR DATA; NOT RECOMMENDED BY APPLE

USER RECORDS

- Instead, to associate a User with other data
 - Create a `User` record type on other record that has a reference to the `UserId`
 - Store any other data



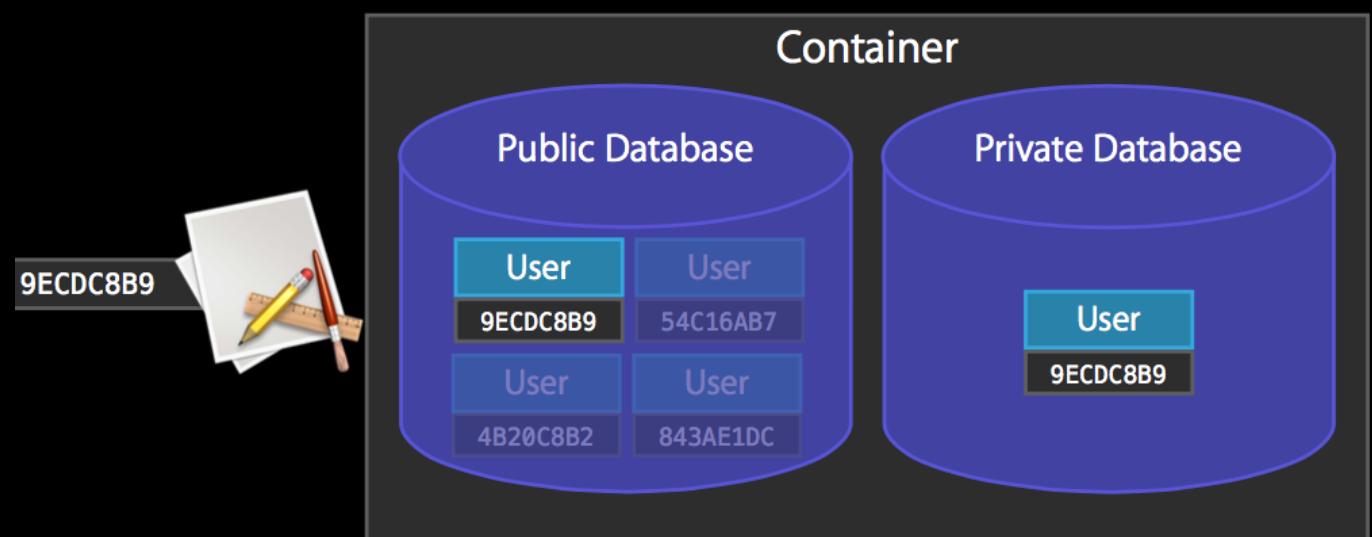
A screenshot of the iCloud Cloud interface, specifically the Record Types section. The top navigation bar shows the path: iCloud > iCloud.cloud.uchicago.CloudyWithACh... . The main area is titled "RECORD TYPES". Below it, there are sections for "DEFAULT TYPES" and "CUSTOM TYPES".

Field Name
recordName
createdBy
createdAt
modifiedBy
modifiedAt
roles
changeTag

Under "CUSTOM TYPES", several record types are listed: "Users", "Alert", "Daily", and "joke". A note states: "Record types will be automatically created in the development environment when you use the native API to create records of that type." At the bottom right is a button labeled "Create New Type".

CloudKit User Accounts

- User metadata is a CKRecordTypeUser record
 - One for each database



UNIQUE TO EACH APPLICATION; WILL NOT BE THE SAME FOR SAME USER IN DIFFERENT APPLICATIONS FROM THE SAME DEVELOPER

USER RECORDS

GET A USER RECORD

```
let container = CKContainer.default()

container.fetchUserRecordID() {
    recordID, error in

    if error != nil {
        print(error!.localizedDescription)
        complete(nil, error as NSError?)
    } else {
        // We have access to the user's record
        print("fetched ID \(recordID?.recordName ?? "")")
        complete(recordID, nil)
    }
}
```

USER RECORDS

```
let container = CKContainer.default()

container.fetchUserRecord(recordID: "23D865322080D4185AD95DB121C80663") { record, error in
    if let error = error {
        print("Error fetching record: \(error.localizedDescription)")
        complete(nil, error as NSError?)
    } else {
        // We have access to the user's record
        print("fetched ID \(record?.recordName ?? "")")
        complete(record, nil)
    }
}
```

USER RECORDS

"_23D865322080D4185AD95DB121C80663"

ZONES	RECORDS	RECORD TYPES	INDEXES	SUBSCRIPTIONS	SUBSCRIPTION TYPES	SECURITY R	EDITING RECORD	X
LOAD RECORDS FROM:		Record Name	Record T...	Fields	Ch. T...	Created	Modified	
Public Database		_f84642634df...	Users		j29c...	Wed May 03 ...	Wed May 03 ...	
_defaultZone		_23d865322080	Users		j28i9s	Tue May 02 201	Tue May 02 201	X
		d4185ad95db12			gv	7 23:51:41 GMT	7 23:51:41 GMT	
		1c80663				-0500 (CDT)	-0500 (CDT)	
USING:		_7b892f2a3ee...	Users		j28e...	Tue May 02 2...	Tue May 02 2...	
		_7f991400b15...	Users		j2q5...	Mon May 15 2...	Mon May 15 2...	
QUERY FOR RECORDS OF TYPE:								
Users								
Filter by: Add filters...								
Sort by: Add sorts...								
Query Records								
Query for records of the type "Users" that are in the "_defaultZone" zone of the "public" database.								
Create New Record...								
USER RECORDS								
▼ Metadata								
Name: _23d865322080d4185ad95db121c80663								
Type: Users								
Database: public								
Zone: _defaultZone								
Created: May 2 2017 11:51 PM by _23d865322080d4185ad95db121c80663								
Modified: May 2 2017 11:51 PM by _23d865322080d4185ad95db121c80663								
Change Tag: j28i9sgv								
▼ References 0								
There are no records referenced by this record.								
▼ Security Roles								
This environment has no custom security roles.								
▼ Fields 0 of 0								

USER RECORDS

```
/// Get the users `RecordId`  
/// - parameters complete: A completion block passing two parameters  
func getUserRecordId(complete: @escaping (CKRecordID?, NSError?) -> ()) {  
  
    let container = CKContainer.default()  
    container.fetchUserRecordID() {  
        recordID, error in  
  
        if error != nil {  
            print(error!.localizedDescription)  
            complete(nil, error as NSError?)  
        } else {  
            // We have access to the user's record  
            print("fetched ID \(recordID?.recordName ?? "")")  
            complete(recordID, nil)  
        }  
    }  
}
```

```
getUserRecordId { (recordID, error) in  
    if let userID = recordID?.recordName {  
        print("iCloudID: \(userID)")  
        self.getJokesByCurrentUser(recordID!)  
    } else {  
        print("iCloudID: nil")  
    }  
}
```

USER RECORDS

- User privacy
 - No disclosure by default
 - Disclosure requested by application
- Opting in allows you to be discoverable by your friends

Allow people using "twenty-nineteen-cloudkit" to look you up by email?

People who know your email address will be able to see that you use this app.

[Don't Allow](#)

[OK](#)

USER RECORDS

- Input
 - User RecordID
 - Email address
 - Entire address book
- Output
 - User RecordID
 - First and last names
- Personally identifying information
 - Requires opt-in

Allow people using “twenty-nineteen-cloudkit” to look you up by email?

People who know your email address will be able to see that you use this app.

Don't Allow

OK

USER RECORDS

GET USER NAME ASSOCIATED
WITH ICLOUD ACCOUNT

```
/// Get the users `RecordId`  
/// - parameters complete: A completion block passing two parameters  
open func getUserIdentity(complete: @escaping (String?, NSError?) -> ()) {  
  
    container.requestApplicationPermission(.userDiscoverability) { (status, error) in  
        self.container.fetchUserRecordID { (record, error) in  
  
            if error != nil {  
                print(error!.localizedDescription)  
                complete(nil, error as NSError?)  
  
            } else {  
                //print("fetched ID \(record?.recordName ?? "")")  
                self.container.discoverUserIdentity(withUserRecordID: record!, completionHandler: { (userID, error) in  
                    let userName = (userID?.nameComponents?.givenName)! + " " + (userID?.nameComponents?.familyName)!  
                    print("CK User Name: " + userName)  
                    complete(userName, nil)  
                })  
            }  
        }  
    }  
}
```

USER RECORDS

```
        print("CK User Name: " + username)
    }
}

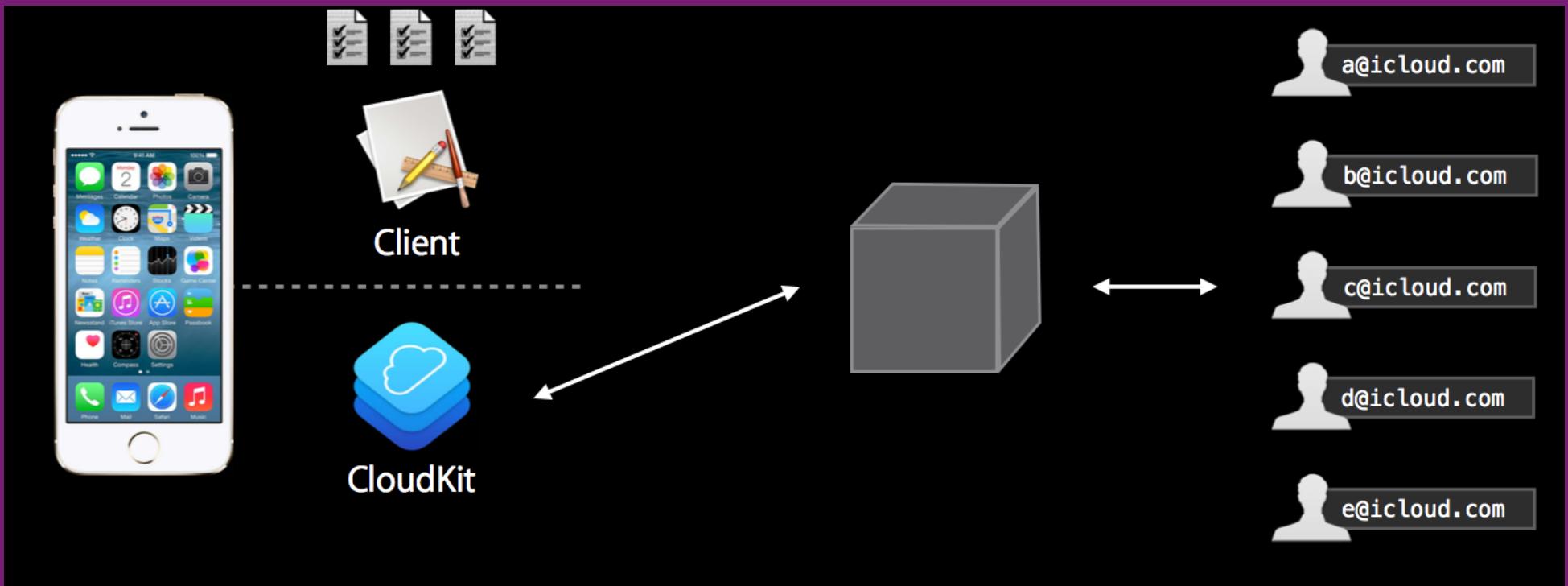
CKContainer.default().requestApplicationPermission(.userDiscoverability) { (status, error) in
    CKContainer.default().fetchUserRecordID { (record, error) in
        CKContainer.default().discoverUserIdentity(withUserRecordID: record!, completionHandler: { (userID, error) in
            userName = (userID?.nameComponents?.givenName)! + " " + (userID?.nameComponents?.familyName)!
            print("CK User Name: " + userName)
        })
    }
}

func getJokesByCurrentUser(_ recordID: CKRecordID) {
    // The user is a reference, so we need to query against a reference
    let reference = CKReference(recordID: recordID, action: .none)
    let predicate = NSPredicate(format: "creatorUserRecordID == %@", reference)
```

CloudyWithAChanceOfErrors

```
fetched ID _23d865322080d4185ad95db121c80663
iCloudID: _23d865322080d4185ad95db121c80663
CK User Name: Andrew Binkowski
```

USER RECORDS

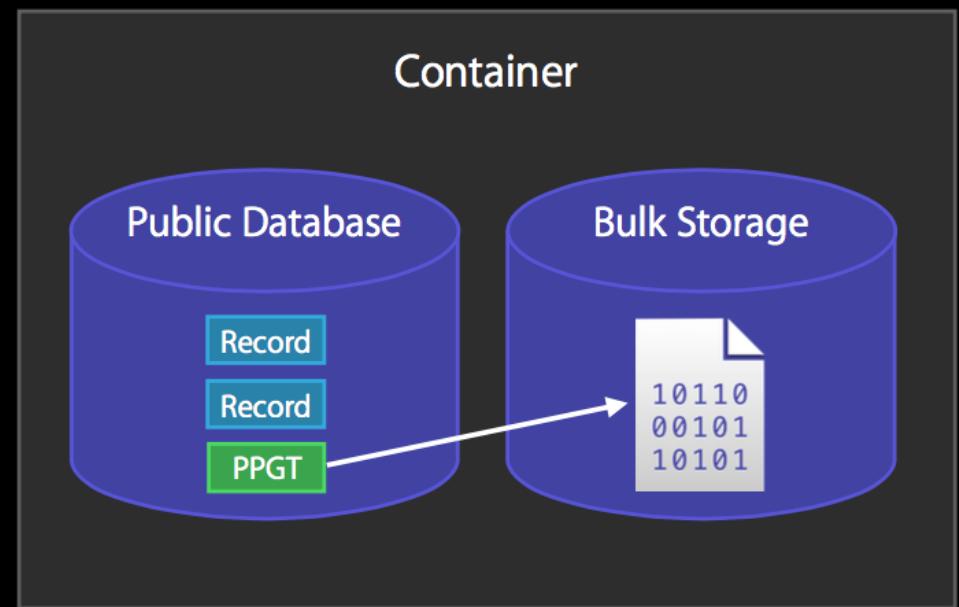


- You can get all friends who use the same app (discover users api)

ASSETS

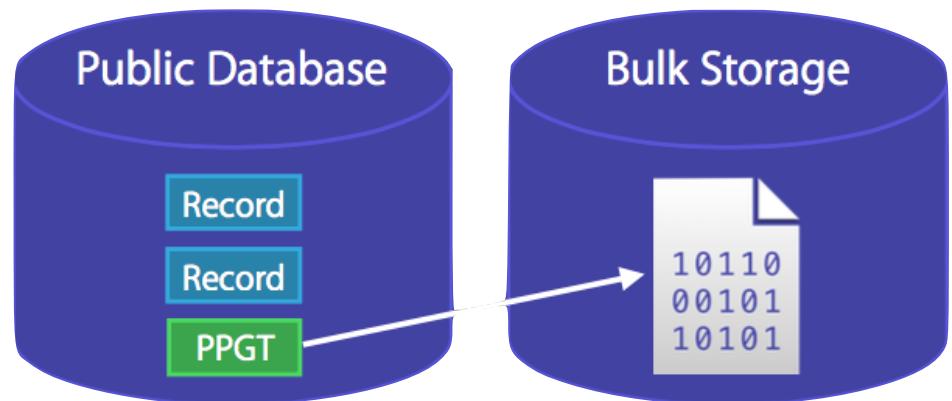
ASSETS

- CKAssets is bulk storage for CloudKit



ASSETS

- CKAsset
 - Large, unstructured data
 - Files on disk
 - Owned by a CKRecord
 - Garbage collected
 - Deletes with a record
 - Efficient uploads and downloads



ASSETS

 > iCloud.cloud.uchicago.CloudyWithAChanceOfErrors > Development Data ANDREW BINKOWSKI ▾

ZONES	RECORDS	RECORD TYPES	INDEXES	SUBSCRIPTIONS	SUBSCRIPTION TYPES	SECURITY R	◀ ▶	EDITING RECORD	X
LOAD RECORDS FROM:		Record Name	Record T...	Fields	Ch. T...	Created	Modified		
Public Database _defaultZone		977D8136-0A78-430E-93C2-513B7B0674ED joke	question Why did the student throw a clock out the window? response She wanted to see time fly. rating_positive 0 rating_negative 0	j9pri30j	Tue Nov 07 2017 09:20:06 GM T-0600 (CST)	Tue Nov 07 2017 09:20:06 GM T-0600 (CST)	X	Metadata Name: 977D8136-0A78-430E-93C2-513B7B0674ED Type: joke Database: public Zone: _defaultZone Created: Nov 7 2017 9:20 AM by _23d865322080d4185ad95db121c80663 Modified: Nov 7 2017 9:20 AM by _23d865322080d4185ad95db121c80663 Change Tag: j9pri30j	
USING: Query Fetch Changes		ASSETS There are no records referenced by this record.							
QUERY FOR RECORDS OF TYPE: joke Filter by: Add filters... Sort by: Add sorts...		ASSETS There are no records referenced by this record.							
Query Records Query for records of the type "joke" that are in the "_defaultZone" zone of the "public" database.									

ASSETS

```
do {  
    let data = UIImagePNGRepresentation(myImage)!  
    try data.writeToURL(tempURL, options: NSDataWritingOptions.AtomicWrite)  
    let asset = CKAsset(fileURL: tempURL)  
    record["myImageKey"] = asset  
}  
catch {  
    print("Error writing data", error)  
}
```

NEED TO CONVERT FILE TO A
CKASSET AND THEN ASSOCIATE
WITH A RECORD

ASSETS

```
if let asset = record["myImageKey"] as? CKAsset,  
    data = NSData(contentsOfURL: asset.fileURL),  
    image = UIImage(data: data)  
{  
    // Do something with the image  
}
```

RETRIEVE AN IMAGE ASSOCIATED WITH A RECORD;
RETURNED A FILE THAT NEEDS TO BE ACTED ON IMMEDIATELY

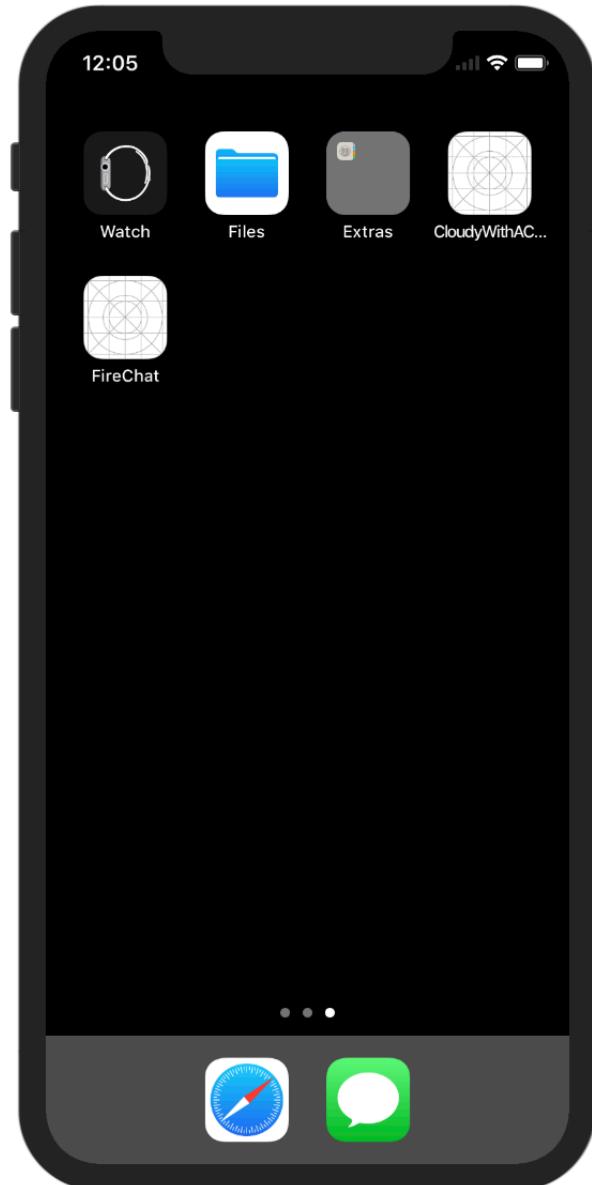
BREAK TIME



QUERIES

QUERIES

- Big Data, Tiny Phone
 - Keep your large data in the cloud
 - Client views slice of that data
 - Client view can change
 - Clients use queries to focus their viewpoint



CLOUDKIT API

- Two ways to interact with CloudKit
- Convenience API for single record interactions
 - May be all you ever need
- NSOperation based API
 - Customizable and fully featured

QUERIES

```
publicDB.save(record) { (record, error) in
    if let error = error {
        print("Error: \(error.localizedDescription)")
        return
    }
    print("Saved record: \(record.debugDescription)")
}
```

- Records are always live when they return
- Saving modified records directly

QUERIES

- CKQuery
 - RecordType
 - NSPredicate
 - CloudKit supports a subset of NSPredicate
 - NSSortDescriptors

Operation	Supported operators
Basic comparisons	=, ==
Boolean value predicates	TRUEPREDICATE
Basic compound predicates	AND, &&
String comparisons	BEGINSWITH
Aggregate Operations	IN
Functions	distanceToLocation:fromLocation:

QUERIES

```
[NSPredicate predicateWithFormat:@"name = %@", partyName];  
  
[NSPredicate predicateWithFormat:@"%@", dynamicKey, value];  
  
[NSPredicate predicateWithFormat:@"start > %@", [NSDate date]];  
  
CLLocation *location = [[CLLocation alloc] initWithLatitude:37.783 longitude:-122.404];  
[NSPredicate predicateWithFormat:@"distanceToLocation:fromLocation:(Location, %@) < 100",  
    location];  
  
[NSPredicate predicateWithFormat:@"ALL tokenize(@\", 'Cdl') IN allTokens",  
    @"after session"];  
  
[NSPredicate predicateWithFormat:@"name = %@ AND startDate > %@",  
    partyName, [NSDate date]];
```

QUERIES

CONNIVENCE API

```
/// Get all jokes in the public database
open func getJokes() {
    let predicate = NSPredicate(format: "TRUEPREDICATE")

    let query = CKQuery(recordType: "joke", predicate: predicate)
    publicDB.perform(query, inZoneWith: nil) { (records, error) -> Void in
        if let error = error {
            print("Error: \(String(describing: error.localizedDescription))")
            return
        }
        for record in records! {
            print("😂: \(record["question"] as! String)")
        }
    }
}
```

DON'T FORGOT TO HANDLE
ERRORS

QUERIES

```
/// Get all jokes by a user
/// - parameter recordID: The `CKRecordID` of the current user
open func getJokesByCurrentUser(_ recordID: CKRecordID) {

    // The user is a reference, so we need to query against a reference
    let reference = CKReference(recordID: recordID, action: .none)

    let predicate = NSPredicate(format: "creatorUserRecordID == %@", reference)

    let query = CKQuery(recordType: "joke", predicate: predicate)
    publicDB.perform(query, inZoneWith: nil) { (records, error) -> Void in
        if let error = error {
            print("Error: \(String(describing: error.localizedDescription))")
            return
        }
        for record in records! {
            print(record["question"] as! String)
        }
    }
}
```

QUERIES

```
/// Get all jokes by a user
/// - parameter recordID: The `CKRecordID` of the current user
open func getJokesByCurrentUser(_ recordID: CKRecordID) {

    // The user is a reference, so we need to query against a reference
    let reference = CKReference(recordID: recordID, action: .none)

    let predicate = NSPredicate(format: "creatorUserRecordID == %@", reference)

    let query = CKQuery(recordType: "joke", predicate: predicate)
    publicDB.perform(query, inZoneWith: nil) { (records, error) -> Void in
        if let error = error {
            print("Error: \(String(describing: error.localizedDescription))")
            return
        }
        for record in records! {
            print(record["question"] as! String)
        }
    }
}
```

NOTE WE HAVE
TO CREATE A
REFERENCE
FROM THE
RECORD

QUERIES

- Potentially have a problem with this query
- Async retrieval of user record required to make query
 - When will the value return?
 - Which order?

```
/// Get all jokes by a user
/// - parameter recordID: The `CKRecord`'s ID
open func getJokesByCurrentUser(_ recordID: CKRecord.ID) {
    // The user is a reference, so we need to convert it to a record
    let reference = CKReference(recordID: recordID, type: .user)
    let predicate = NSPredicate(format: "jokeAuthor == %@", reference)
    let query = CKQuery(recordType: "joke", predicate: predicate)
    publicDB.perform(query, inZoneWith: nil) { [weak self] (records, error) in
        if let error = error {
            print("Error: \(String(describing: error))")
            return
        }
        for record in records! {
            print(record["question"] as! String)
        }
    }
}
```

QUERIES

- NSOperation based query
 - More control of responses
 - Results cursor
 - Guarantee the order of operations

Class

CKQueryOperation

A CKQueryOperation object is a concrete operation that you can use to execute queries against a database. A query operation takes the query parameters you provide and applies those parameters to the specified database and zone, delivering any matching records asynchronously to the blocks that you provide.

Overview

To perform a new search:

1. Initialize a CKQueryOperation object with a [CKQuery](#) object containing the search criteria and sorting information for the records you want.
2. Assign a block to the [queryCompletionBlock](#) property so that you can process the results and execute the operation.

If the search yields many records, the operation object may deliver a portion of the total results to your blocks.

QUERIES

- Operation query with cursor

```
/// Use the operation API to make a query and use cursors
/// to control the flow of data
/// - parameter query: A `CKQuery?`, most likely represents the initial query
open func getJokesWithOperation(query: CKQuery?, cursor: CKQueryCursor?) {
    var queryOperation: CKQueryOperation!

    if query != nil {
        let predicate = NSPredicate(value: true)
        let query = CKQuery(recordType: "joke", predicate: predicate)
        queryOperation = CKQueryOperation(query: query)
    } else if let cursor = cursor {
        print("== Cursor =====")
        queryOperation = CKQueryOperation(cursor: cursor)
    }

    // Query parameters
    //queryOperation.desiredKeys = ["", "", ""]
    queryOperation.queuePriority = .veryHigh
    queryOperation.resultsLimit = 2
    queryOperation.qualityOfService = .userInteractive

    // This gets called each time per record
    queryOperation.recordFetchedBlock = {
        (record: CKRecord!) -> Void in
        if record != nil {
            print("😂 operation: \(record["question"] as! String)")
        }
    }

    // This is called after all records are retrieved and iterated
    // on
    queryOperation.queryCompletionBlock = { cursor, error in
        if (error != nil) {
            print("Error:\(String(describing: error))")
            return
        }

        if let cursor = cursor {
            print("There is more data to fetch")
            self.getJokesWithOperation(query: nil, cursor: cursor)

            print("Done with operation...")
            //OperationQueue.main.addOperation() {
            // Do anything else with the record after downloaded that
            // needs to be on the main thread
            //}
        }
    }
    self.publicDB.add(queryOperation)
}
```

QUERIES

```
/// Use the operation API to make a query and use cursors
/// to control the flow of data
/// - parameter query: A `CKQuery?`, most likely represents the initial query
open func getJokesWithOperation(query: CKQuery?, cursor: CKQueryCursor?) {
    var queryOperation: CKQueryOperation!

    if query != nil {
        let predicate = NSPredicate(value: true)
        let query = CKQuery(recordType: "joke", predicate: predicate)
        queryOperation = CKQueryOperation(query: query)
    } else if let cursor = cursor {
        print("== Cursor =====")
        queryOperation = CKQueryOperation(cursor: cursor)
    }

    // Query parameters
    //queryOperation.desiredKeys = ["", "", ""]
    queryOperation.queuePriority = .veryHigh
    queryOperation.resultsLimit = 2
```

QUERIES

```
// Query parameters
//queryOperation.desiredKeys = ["", "", ""]
queryOperation.queuePriority = .veryHigh
queryOperation.resultsLimit = 2
queryOperation.qualityOfService = .userInteractive

// This gets called each time per record
queryOperation.recordFetchedBlock = {
    (record: CKRecord!) -> Void in
    if record != nil {
        print("😂 operation: \(record["question"] as! String)")
    }
}

// This is called after all records are retrieved and iterated
// on
queryOperation.queryCompletionBlock = { cursor, error in
    if (error != nil) {
        print("Error: \(error.localizedDescription) - \(error?.localizedDescription ?? "")")
    }
}
```

QUERIES

```
// Query parameters
//queryOperation.desiredKeys = ["", "", ""]
queryOperation.queuePriority = .veryHigh
queryOperation.resultsLimit = 2
queryOperation.qualityOfService = .userInteractive

// This gets called each time per record
queryOperation.recordFetchedBlock = {
    (record: CKRecord!) -> Void in
    if record != nil {
        print("😂 operation: \(record["question"] as! String)")
    }
}

// This is called after all records are retrieved and iterated
// on
queryOperation.queryCompletionBlock = { cursor, error in
    if (error != nil) {
        print("Error: \(error.localizedDescription) - \(error)")
    }
}
```

QUERIES

```
// This is called after all records are retrieved and iterated
// on
queryOperation.queryCompletionBlock = { cursor, error in
    if (error != nil) {
        print("Error:\(String(describing: error))")
        return
    }

    if let cursor = cursor {
        print("There is more data to fetch")
        self.getJokesWithOperation(query: nil, cursor: cursor)

        print("Done with operation...")
        //OperationQueue.main.addOperation() {
        // Do anything else with the record after downloaded that
        // needs to be on the main thread
        //}
    }
}
```

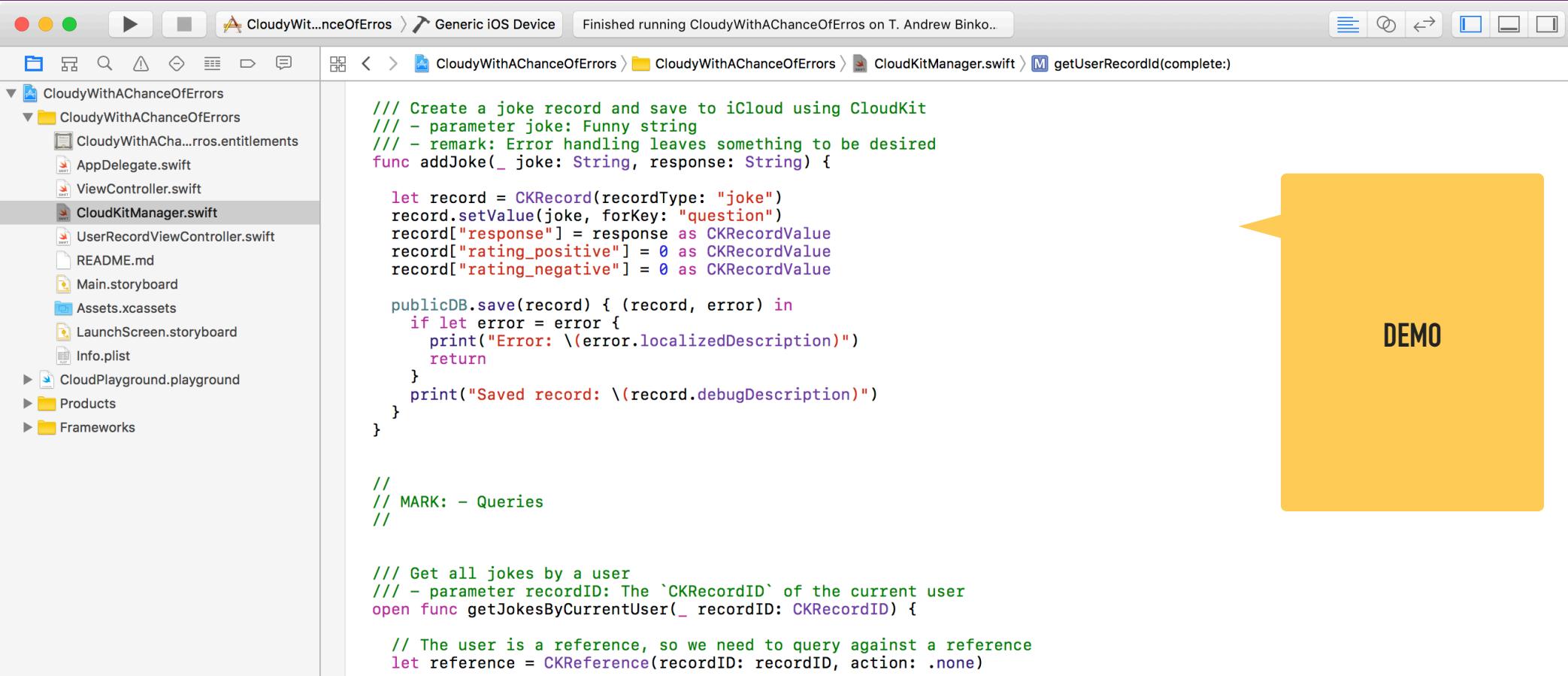
QUERIES

```
if let cursor = cursor {
    print("There is more data to fetch")
    self.getJokesWithOperation(query: nil, cursor: cursor)

    print("Done with opeartion...")
    //OperationQueue.main.addOperation() {
    // Do anything else with the record after downloaded that
    // needs to be on the main thread
    //}
}
}

self.publicDB.add(queryOperation)
}
```

QUERIES



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

CloudyWithAChanceOfErrors

CloudyWithAChanceOfErrors

CloudyWithAChanceOfErrors.entitlements

AppDelegate.swift

ViewController.swift

CloudKitManager.swift

UserRecordViewController.swift

README.md

Main.storyboard

Assets.xcassets

LaunchScreen.storyboard

Info.plist

CloudPlayground.playground

Products

Frameworks

CloudyWithAChanceOfErrors > CloudyWithAChanceOfErrors > CloudKitManager.swift > M getUserRecordId(complete:)

```
/// Create a joke record and save to iCloud using CloudKit
/// - parameter joke: Funny string
/// - remark: Error handling leaves something to be desired
func addJoke(_ joke: String, response: String) {

    let record = CKRecord(recordType: "joke")
    record.setValue(joke, forKey: "question")
    record["response"] = response as CKRecordValue
    record["rating_positive"] = 0 as CKRecordValue
    record["rating_negative"] = 0 as CKRecordValue

    publicDB.save(record) { (record, error) in
        if let error = error {
            print("Error: \(error.localizedDescription)")
            return
        }
        print("Saved record: \(record.debugDescription)")
    }
}

// MARK: - Queries

/// Get all jokes by a user
/// - parameter recordID: The `CKRecordID` of the current user
open func getJokesByCurrentUser(_ recordID: CKRecordID) {

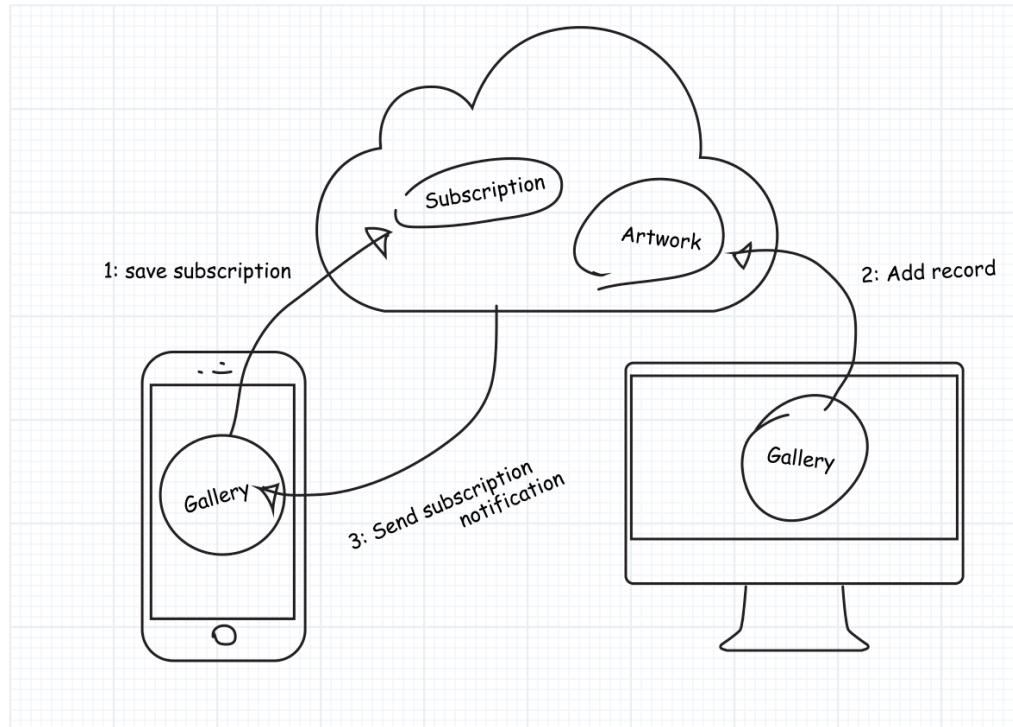
    // The user is a reference, so we need to query against a reference
    let reference = CKReference(recordID: recordID, action: .none)
```

DEMO

SUBSCRIPTIONS

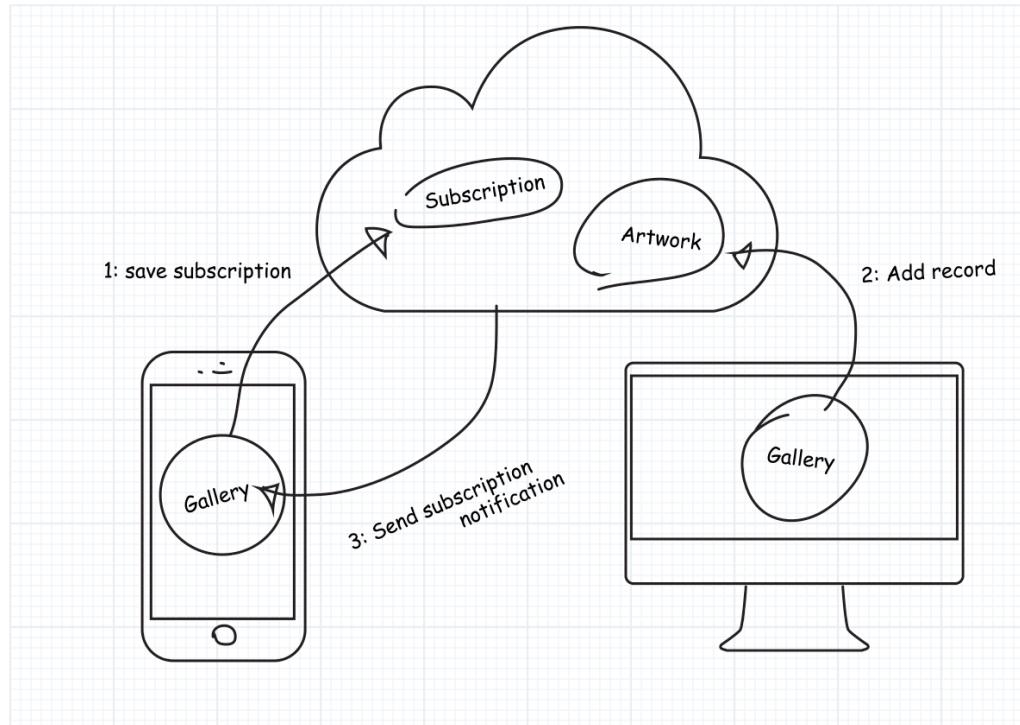
SUBSCRIPTIONS

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



SUBSCRIPTIONS

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



SUBSCRIPTIONS

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

Subscriptions ▾

Public Database

Fetch Subscriptions

Delete Subscription

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

SUBSCRIPTIONS

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

The screenshot shows the iCloud CloudKit Subscriptions interface. At the top, there's a navigation bar with the URL "iCloud.mobi.uchicago.twenty-nineteen-cloudkit" and dropdown menus for "Development" and "Data". Below the navigation is a section titled "Subscriptions" with a dropdown menu set to "Public Database". A blue button labeled "Fetch Subscriptions" is visible. On the right, a table lists a single subscription entry:

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

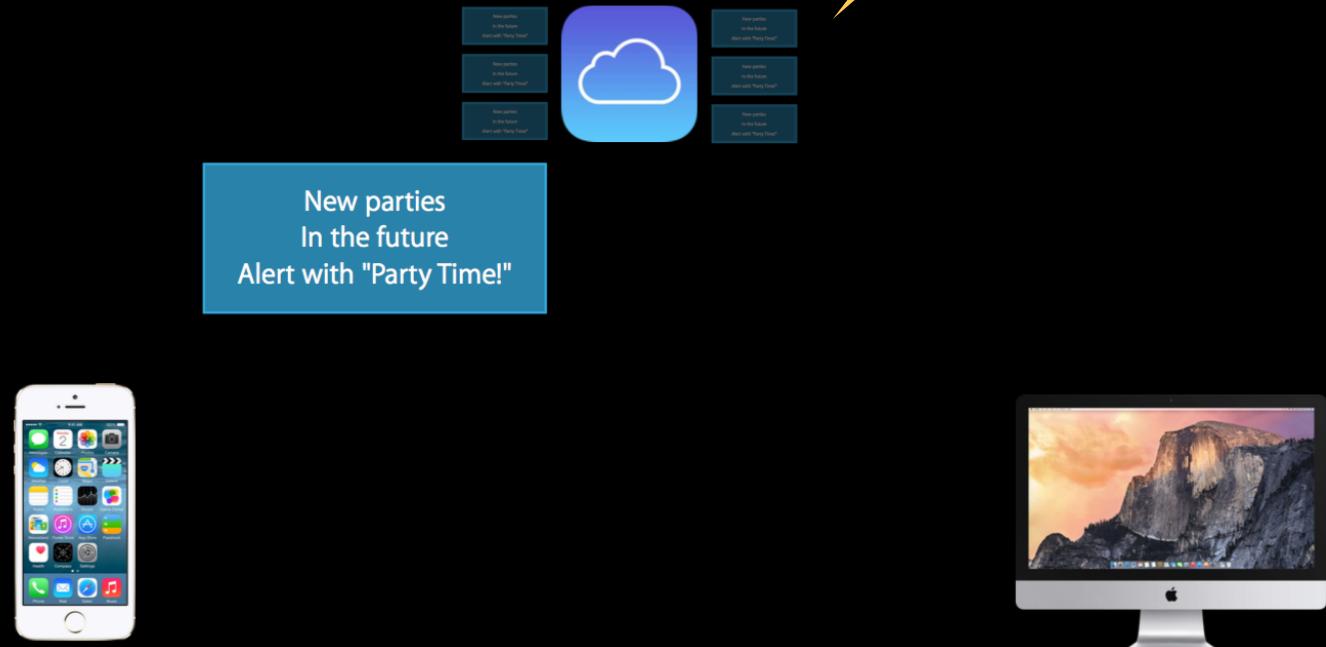
Below the table is a "Delete Subscription" button with a trash icon.

A yellow callout bubble with a black border and a black arrow points from the bottom right towards the "Subscriptions" section. The text inside the callout bubble reads: "CHEAPEST, EASIEST WAY TO ADD PUSH NOTIFICATIONS TO AN APP".

SUBSCRIPTIONS

- Subscription in action

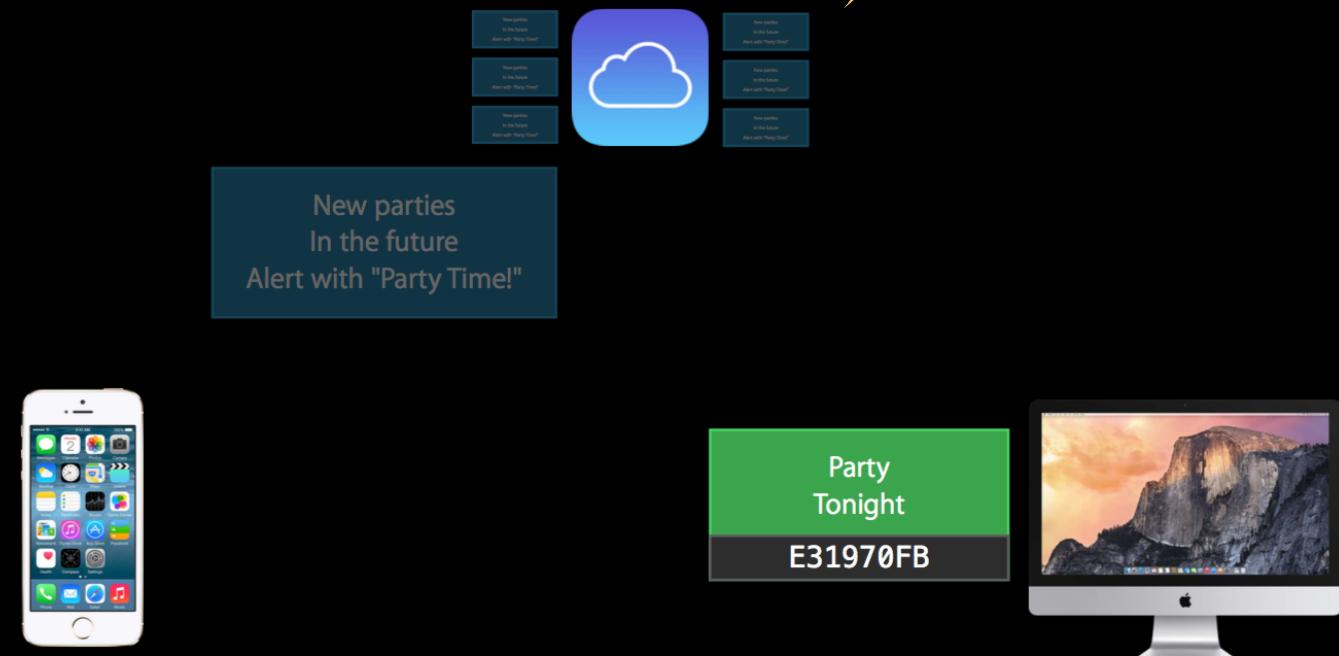
APPLE PARTY NOTIFICATION EXAMPLE



SUBSCRIPTIONS

- Subscription in action

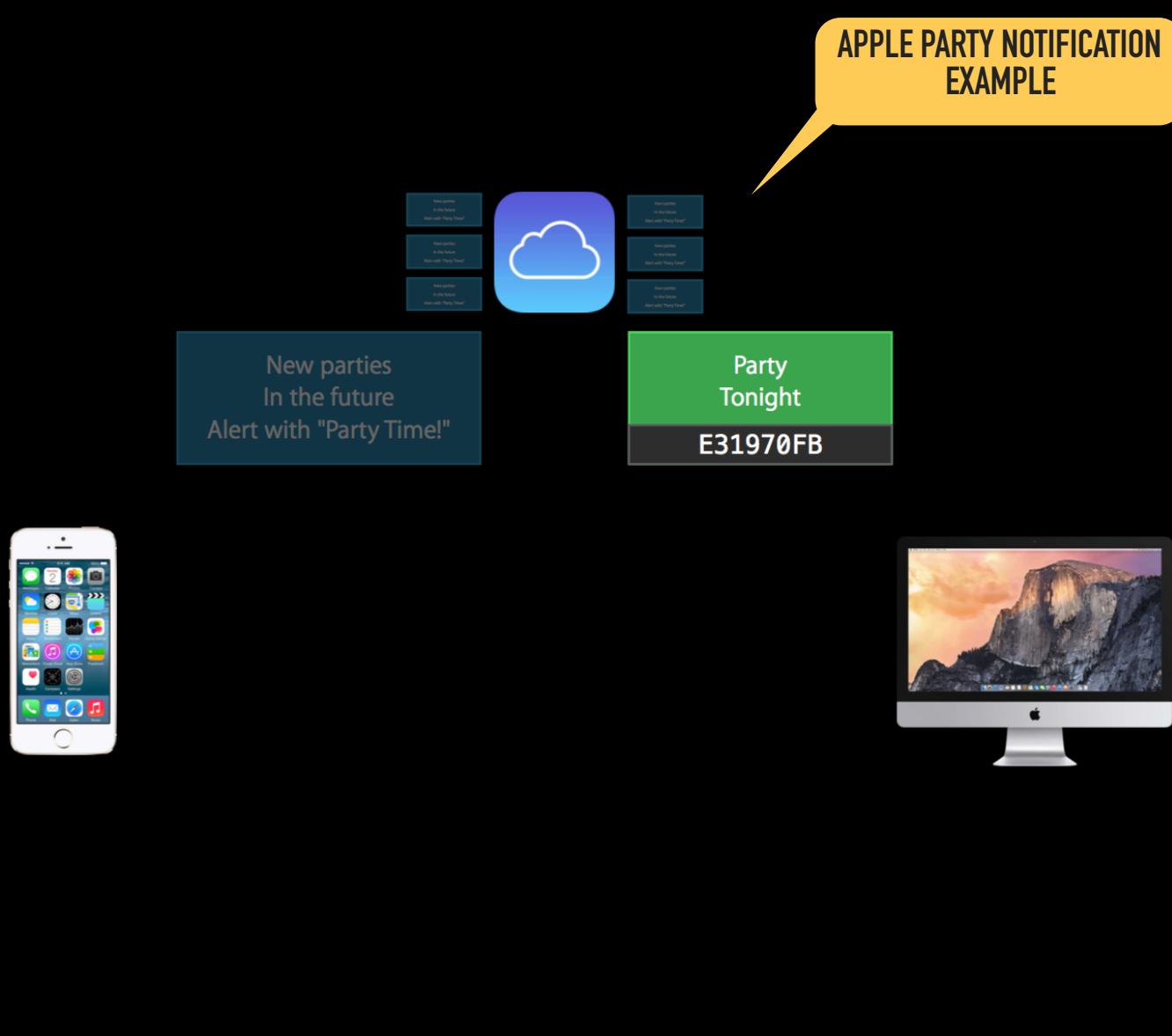
APPLE PARTY NOTIFICATION EXAMPLE



SUBSCRIPTIONS

- Subscription in action

APPLE PARTY NOTIFICATION EXAMPLE



SUBSCRIPTIONS

- Subscription in action

APPLE PARTY NOTIFICATION EXAMPLE



SUBSCRIPTIONS

- Subscription in action

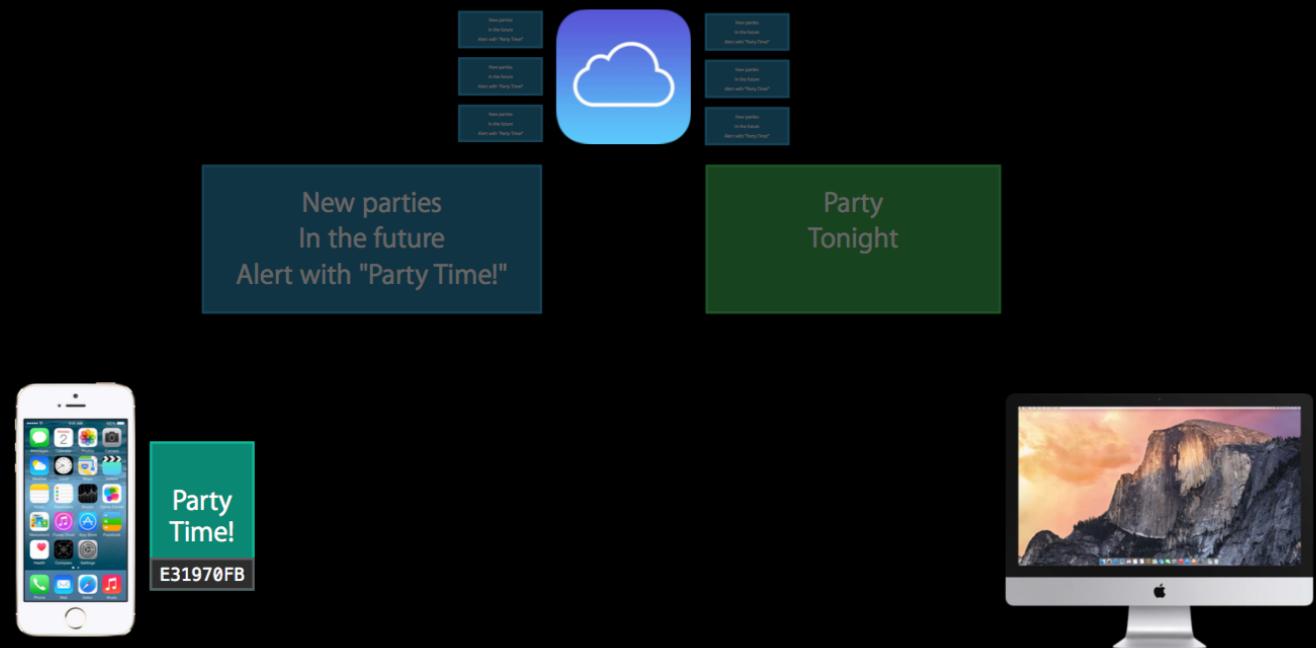
APPLE PARTY NOTIFICATION EXAMPLE



SUBSCRIPTIONS

- Subscription in action

APPLE PARTY NOTIFICATION EXAMPLE



SUBSCRIPTIONS

- Subscriptions are user based
- Require a unique id

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

SUBSCRIPTIONS

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

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

SUBSCRIPTIONS

- Create a subscription

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

SUBSCRIPTIONS

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

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

SUBSCRIPTIONS

SHOW ALL SUBSCRIPTIONS

Development > Data > Andrew Binkowski | ?

Subscriptions ▾

Public Database

Fetch Subscriptions

Delete Subscription

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

SEE DETAILS

SETUP YOUR APP TO
HANDLE NOTIFICATIONS

SUBSCRIPTIONS

- Required capabilities
 - Push notifications

The screenshot shows the Xcode Capabilities tab for a project named "CloudyWithAChanceOfErrors". The "Push Notifications" capability is selected and turned "ON". Below it, other capabilities like Game Center, Wallet, Siri, Apple Pay, In-App Purchase, Maps, Personal VPN, Network Extensions, and Background Modes are listed with their status set to "OFF". Under the "Background Modes" section, a list of modes is provided, with "Background fetch" and "Remote notifications" checked.

PROJECT
CloudyWithAChanceOf...

TARGETS
CloudyWithAChanceOf...

General Capabilities Resource Tags Info Build Settings Build Phases Build Rules

iCloud ON

Push Notifications ON

Game Center OFF

Wallet OFF

Siri OFF

Apple Pay OFF

In-App Purchase OFF

Maps OFF

Personal VPN OFF

Network Extensions OFF

Background Modes ON

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

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

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

SUBSCRIPTIONS

- Required capabilities
 - Remote notification
 - Background fetch



Background Modes

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

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



Inter-App Audio



Keychain Sharing



Associated Domains

SUBSCRIPTIONS

REQUEST PERMISSIONS TO RECEIVE
NOTIFICATIONS

```
func application(_ application: UIApplication,
                 didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

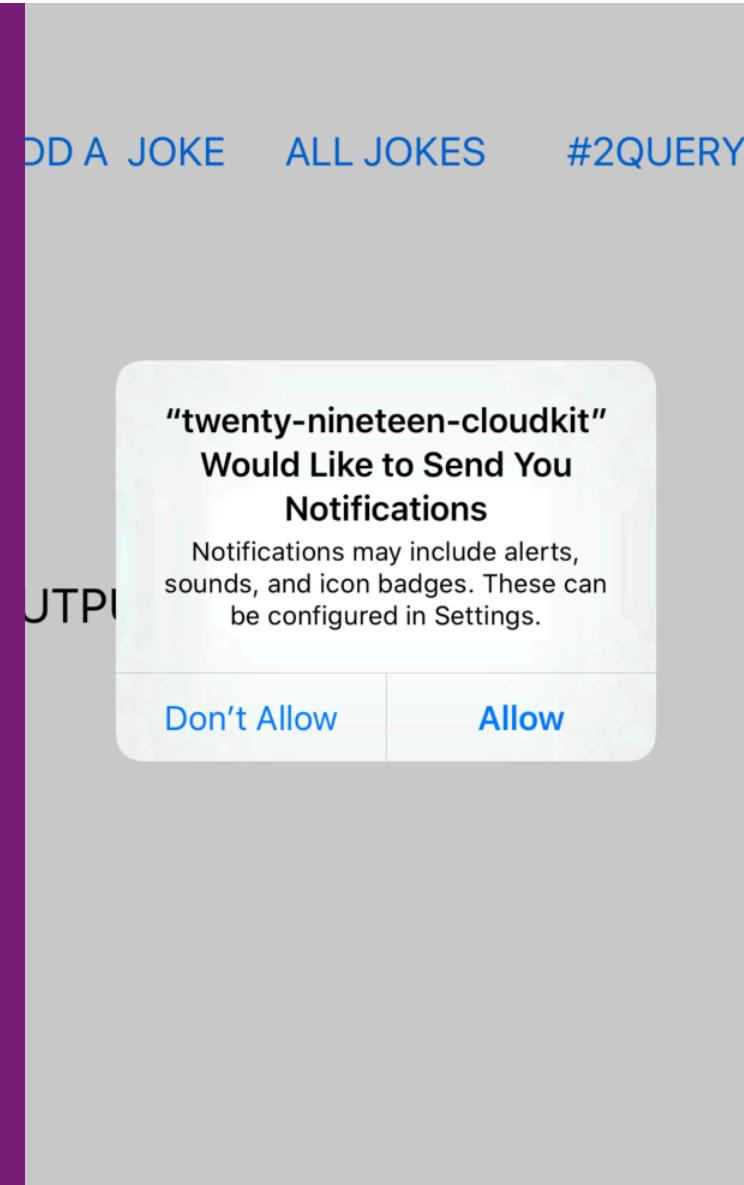
    // Set the notification delegate
    UNUserNotificationCenter.current().requestAuthorization(options: [.alert, .sound, .badge]) { granted, error in
        if let error = error {
            print("Error: \(error.localizedDescription)")
        } else {
            application.registerForRemoteNotifications()
        }
    }
    UNUserNotificationCenter.current().delegate = self

    // Register subscriptions
    CloudKitManager.sharedInstance.registerSubscriptionsWithIdentifier("id2")
    CloudKitManager.sharedInstance.registerSilentSubscriptionsWithIdentifier("id3")
    //configureUserNotificationsCenter()

    if let notification = launchOptions?[UIApplicationLaunchOptionsKey.remoteNotification] as? [String: AnyObject] {
        // 2
        let aps = notification["aps"] as! [String: AnyObject]
        print("APS: \(aps)")
        // 3
        //((window?.rootViewController as? UITabBarController)?.selectedIndex = 1
    }
    return true
}
```

SUBSCRIPTIONS

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



SUBSCRIPTIONS

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

REGISTER SUBSCRIPTIONS

error in

SUBSCRIPTIONS

- Register a subscription

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

HANDLE NOTIFICATIONS (THERE ARE 3 WAYS)

SUBSCRIPTIONS

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

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

SUBSCRIPTIONS

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

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

SUBSCRIPTIONS

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

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {  
    // Set up CloudKit  
    CloudKitManager.sharedInstance().setup()  
  
    // Set up UserNotifications  
    UNUserNotificationCenter.current().delegate = self  
  
    // Register subscriptions  
    CloudKitManager.sharedInstance().registerForCloudKitNotifications()  
    CloudKitManager.sharedInstance().registerForRemoteNotifications()  
  
    // Configure User Notifications  
    UNUserNotificationCenter.current().requestAuthorization(options: [.alert, .sound, .badge]) {  
        granted, error in  
        if let error = error {  
            print("Error: \(error.localizedDescription)")  
        }  
    }  
  
    // Handle CloudKit notifications  
    func application(_ application: UIApplication, didReceiveRemoteNotification notification: [AnyHashable: Any], fetchCompletionHandler completionHandler: @escaping (UIBackgroundFetchResult) -> Void) {  
        // If the notification was triggered by CloudKit  
        if let notification = launchOptions?[.remoteNotification] as? [AnyHashable: Any]  
            let aps = notification[.aps] as? [String: Any]  
            print("APS: \(aps ?? [:])")  
        }  
        completionHandler(.newData)  
    }  
}
```

FRESH LAUNCH
(RARELY HAPPENS)

SUBSCRIPTIONS

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



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

SUBSCRIPTIONS

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

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

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

SUBSCRIPTIONS

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

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

SUBSCRIPTIONS

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

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

SUBSCRIPTIONS

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

SUBSCRIPTIONS

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

SILENT NOTIFICATIONS

DEBUGGING NOTIFICATION HANDLING

SUBSCRIPTIONS

Guides and Sample Code

Developer



Debugging issues with CloudKit subscriptions

Technical Q&A QA1917

Debugging issues with CloudKit subscriptions

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

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

Cases that aren't expected to trigger CloudKit notifications

When working with CloudKit subscriptions, be aware that:

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

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

SUBSCRIPTIONS

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

SUBSCRIPTIONS

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

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

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

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

SUBSCRIPTIONS



The screenshot shows the Xcode interface with the following details:

- Project Navigator:** Shows the project structure with a warning icon.
- Editor:** Displays the `AppDelegate.swift` file content:

```
92         style: `.default`,
93         handler: { _ in
94             NSLog("The \"OK\" alert occurred.")
95         })
96     let rvc = (window?.rootViewController as? UITabBarController)
97     rvc?.selectedViewController?.present(alert, animated: true, completion: nil)
98 }
99
100 // Throw a local notification now to test the launch cycle
101 func now(message: String, time: Double = 2) {
102     let notificationContent = UNMutableNotificationContent()
103     notificationContent.title = "🎉"
104     notificationContent.body = message
105
106     // Add Trigger
107     let notificationTrigger = UNTimeIntervalNotificationTrigger(timeInterval: TimeInterval(time), repeats: false)
108
109     // Create Notification Request
110     let notificationRequest = UNNotificationRequest(identifier: UUID.init().debugDescription,
111                                                 content: notificationContent,
112                                                 trigger: notificationTrigger)
113
114     // Add Request to User Notification Center
115     UNUserNotificationCenter.current().add(notificationRequest) { (error) in
116         if let error = error {
117             print("Unable to Add Notification Request (\(error), \(error.localizedDescription))")
118         }
119     }
120 }
121
122 }
```

- Identity and Type:** Shows the file is named `AppDelegate.swift`, type is `Default - Swift Source`, and location is `Relative to Group`.
- Document Outline:** Lists various warnings related to Auto Layout Localization, Fixed width constraints, Unsupported Configuration, and Dependency Analysis Warnings.
- Utilities:** Shows icons for View Controller, Storyboard Reference, Navigation Controller, Table View Controller, and Collection View Controller.

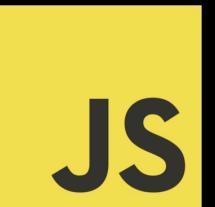
WEB SERVICES

PREVIEW

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" />
```



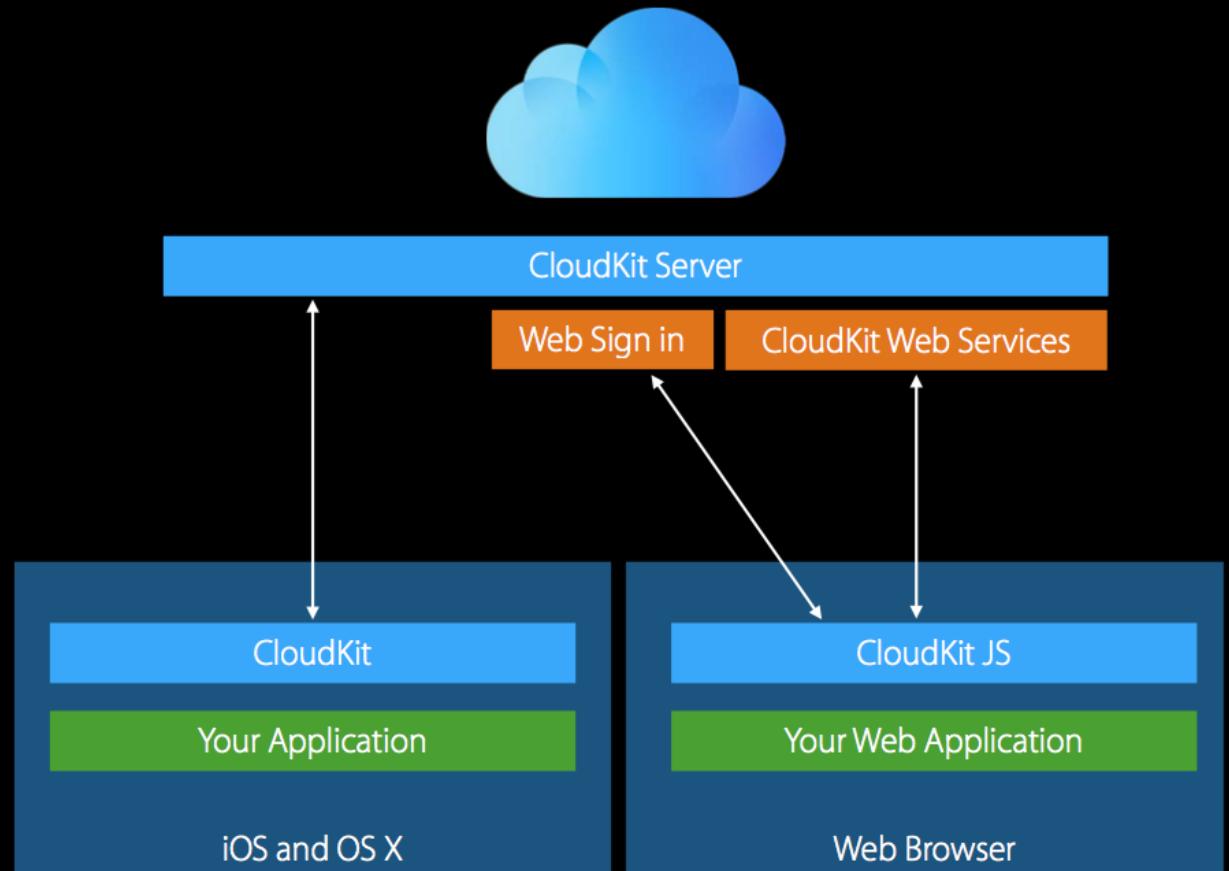
WEB SERVICE API

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



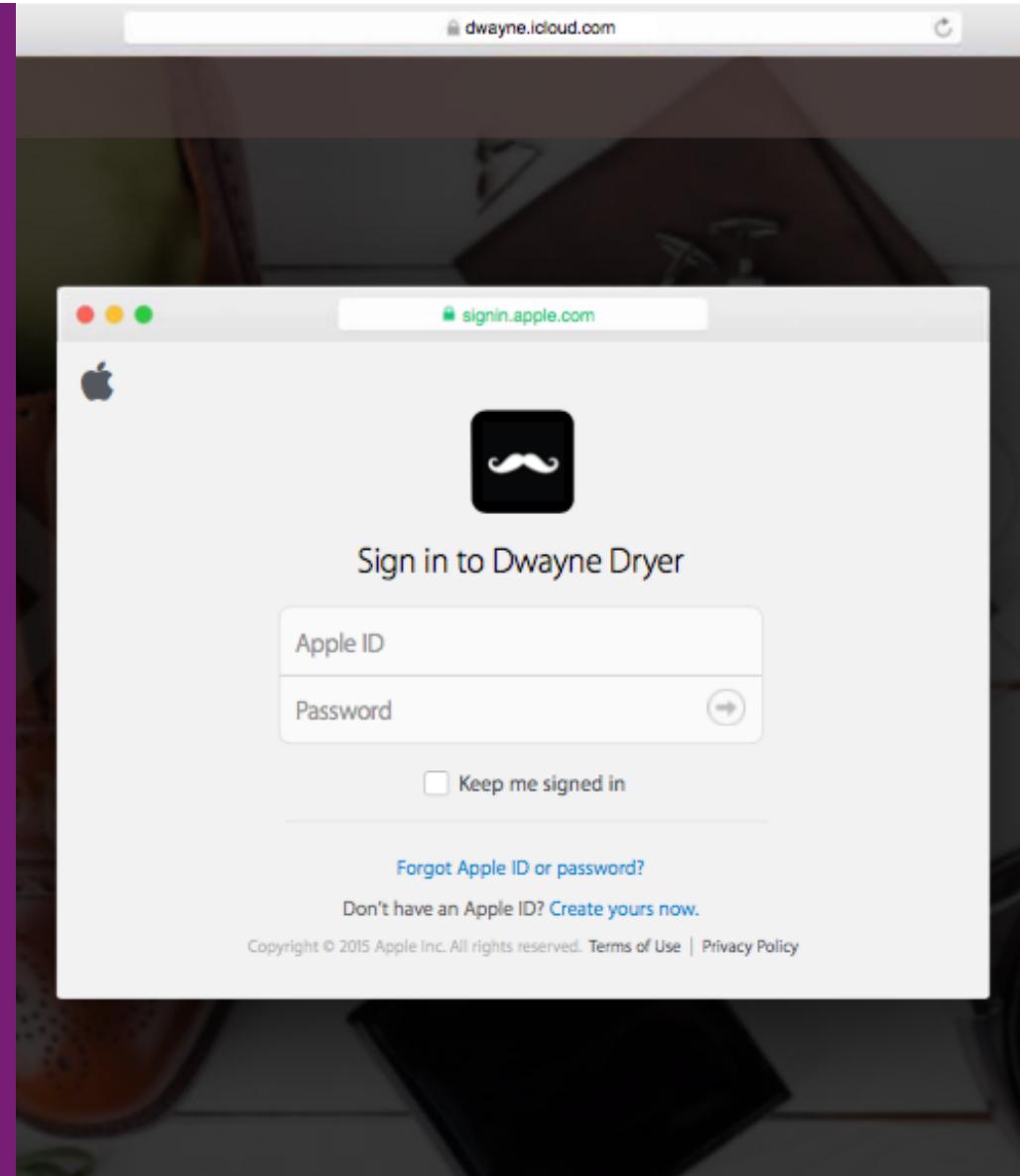
WEB SERVICES

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



WEB SERVICE API

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



WEB SERVICE API

 CloudKit Catalog

-  README
-  Authentication
-  Discoverability
-  Query
-  Zones
-  Records
-  Sync
-  Sharing
-  Subscriptions
-  Notifications
Disconnected

 Run Code

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

CloudKit on the web

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

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

Obtaining the CloudKit JS library

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

Option #1 - Load CloudKit JS synchronously

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

Unauthenticated User

WHEN TO USE CLOUDKIT

WHEN TO USE CLOUDKIT

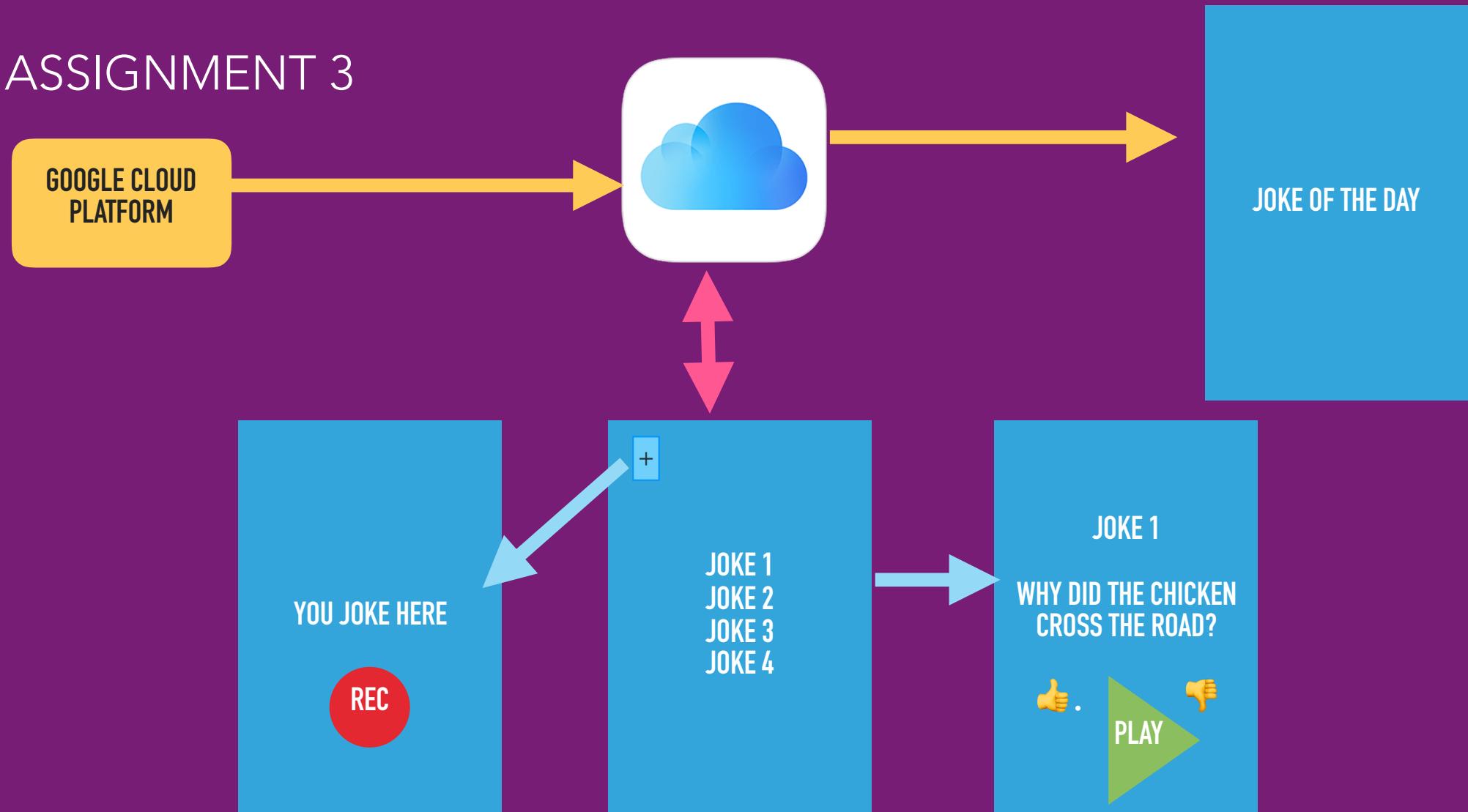
- You're developing exclusively for Apple platforms
- You use it as a partial solution

ASSIGNMENT 4

ASSIGNMENT 5

- Create a "Joke of the Day" application
- Users submit jokes (text and audio)
- Users can view/listen to all jokes and rate them
 - New jokes trigger a silent push notification to update local data
- Use the JS API to trigger a daily "Joke of the Day" notification sent to all users from a different service
- Create a simple interface to send push notifications to users
 - iOS, macOS or web app

ASSIGNMENT 3





THE UNIVERSITY OF
CHICAGO



MPCS 51033 • AUTUMN 2019 • SESSION 6

BACKENDS FOR MOBILE APPLICATIONS