



THE UNIVERSITY OF  
CHICAGO



MPCS 51033 • SPRING 2017 • SESSION 6

---

# BACKENDS FOR MOBILE APPLICATIONS

# CLASS NEWS

## CLASS NEWS

- Office hours tomorrow from 10-11:30 in Young 308
- Make-up class Saturday, May 20, 9AM-12PM in Young 101(?)
  - iOS Performance and Security
  - ...

## CLASS NEWS

- Week 6 - Assignment 4 (CloudKit)
- Week 7 - Finish assignment 4 and work on case study and final project
- Week 8 - Assignment 5 Assigned (Swift on Server)
- Week 9 - Case Studies in Class; Assignment 5 Due
- Week 10 -
- Week 11 - Final Project Presentations

# CLASS NEWS

- Case Studies
  - Find an area of interest in mobile/cloud computing and present to class
  - Ideas:
    - New service (serverless, azure, aws lambda)
    - Technique (sharding :), mysql with app engine)
    - Case study (Snapchat on App Engine, what does XX company use)
    - Deeper dive on topic covered (eg. Google Vision API, etc.)
  - ~15 minutes

## CLASS NEWS

- Final Projects
  - Opportunity for you to apply what you learned in class to a project you care about

CLOUDKIT



# CLOUDKIT

- CloudKit  
QuickStart

CloudKit Quick Start

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

[Next](#)

## 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

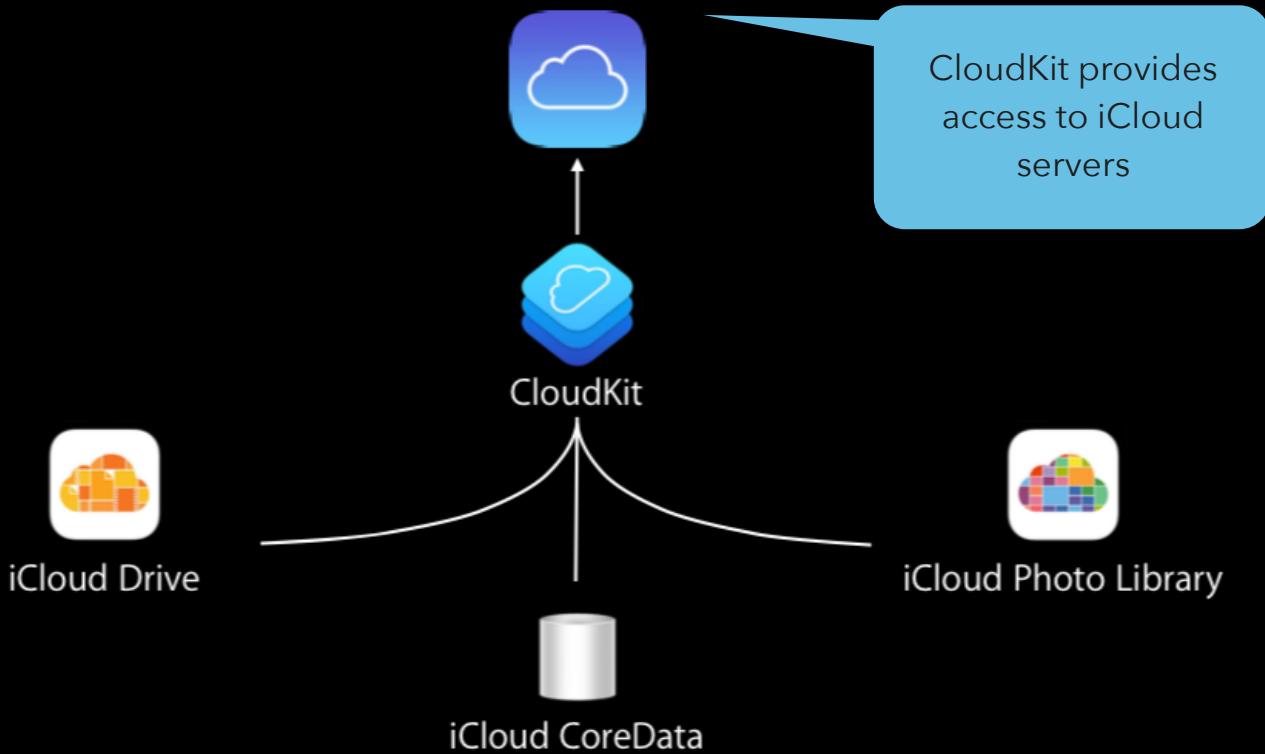
See [Glossary](#) for the definition of database terms used in this book.

## 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.

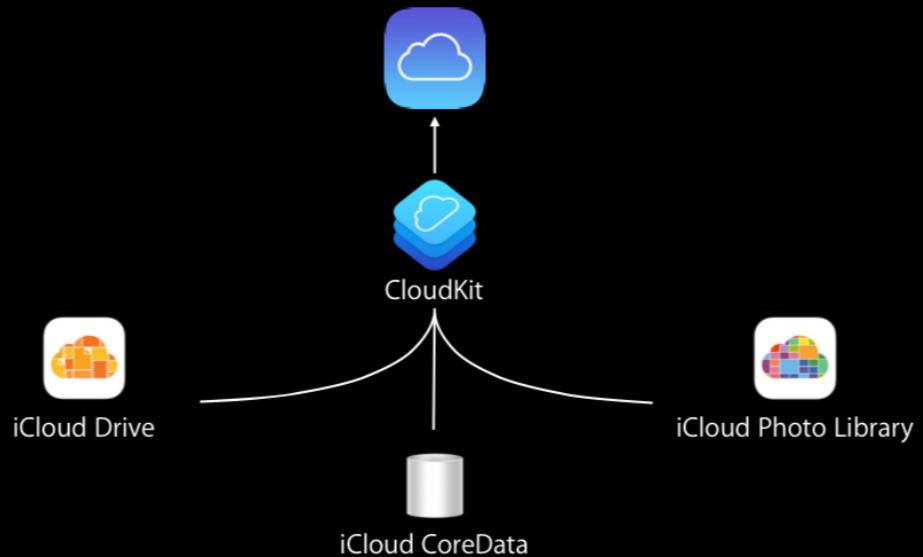
# CLOUDKIT



# 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.

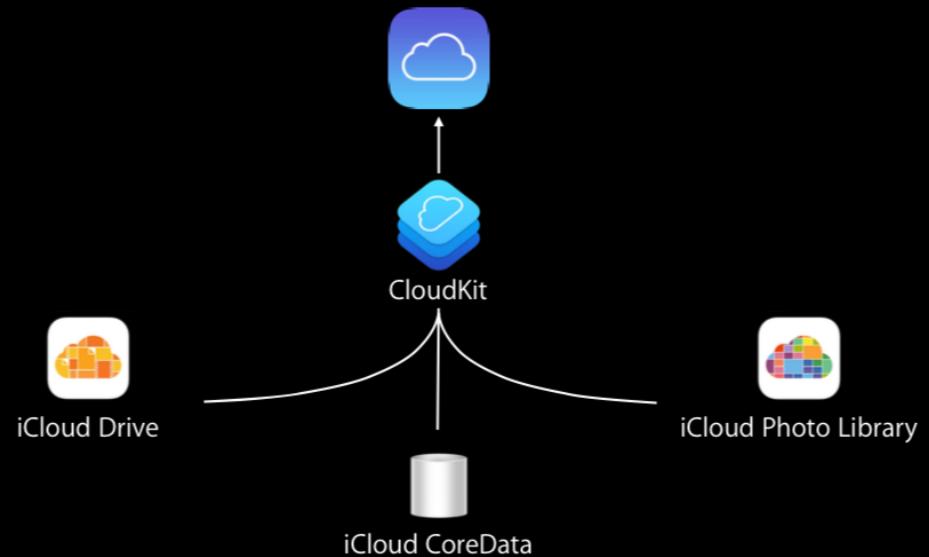
Based on:  
4 per 100,000 users

**\$0**

Total Cost

# CLOUDKIT

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



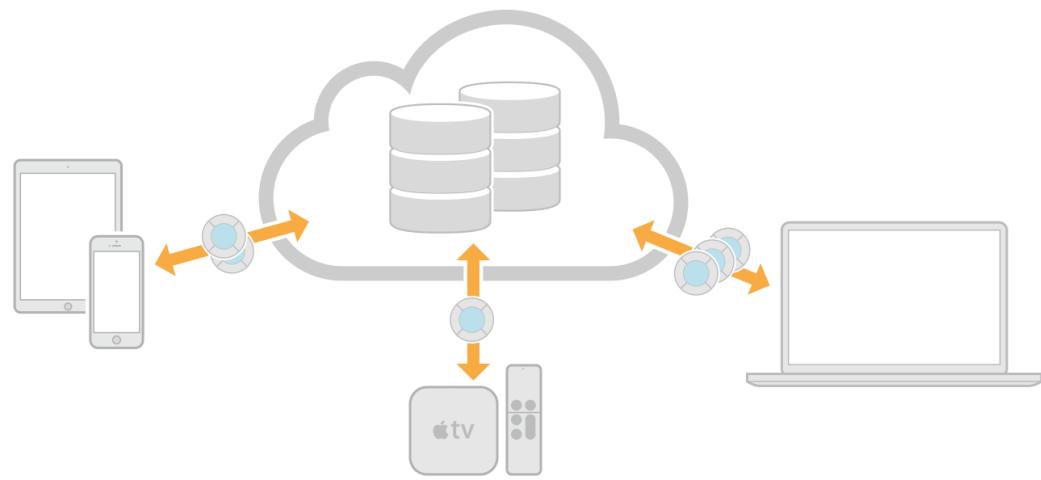
# CLOUDKIT

- Databases
  - Public
  - Private
  - Shared



# CLOUDKIT

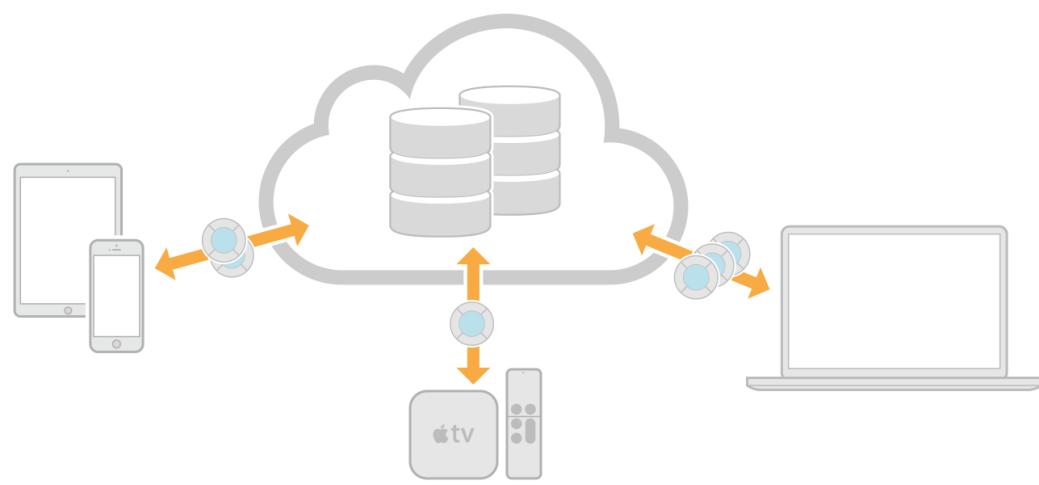
- Structured and bulk data
  - Data with types
  - Blobs



# CLOUDKIT

## CORE OBJECT

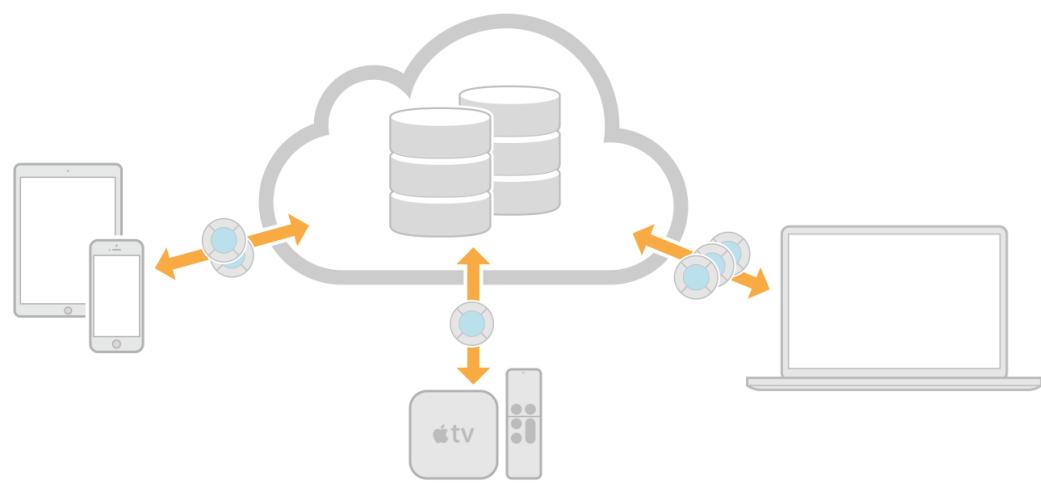
- Containers
- Databases
- Records
- Record Zones
- Record Identifiers
- Shares
- References
- Assets



# CLOUDKIT

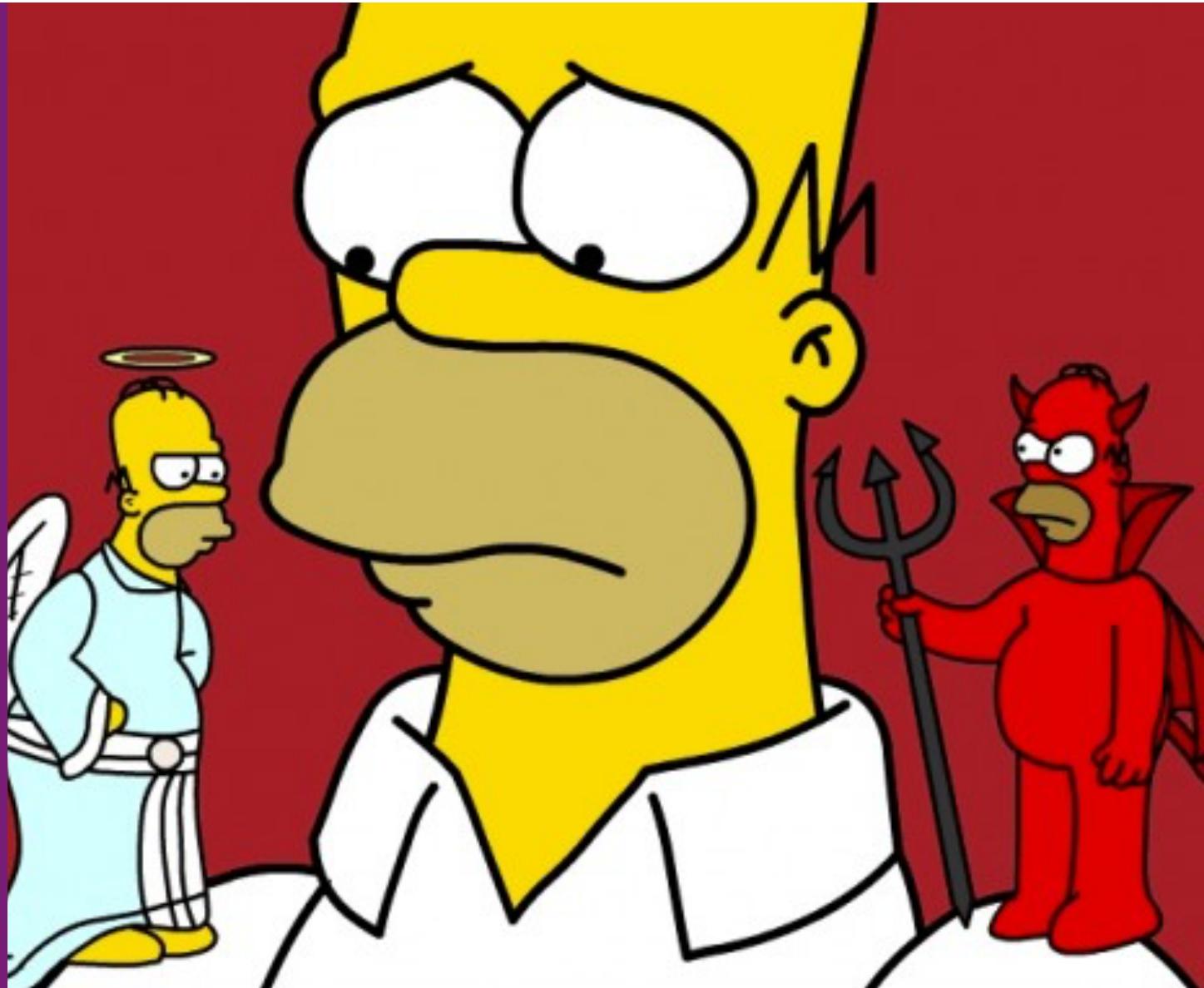
- Transport layer, not local persistence
- You still need to figure out what to do with the data once you have it on your device
  - Cache
  - Mirror

#2 consideration  
on choosing over  
other services



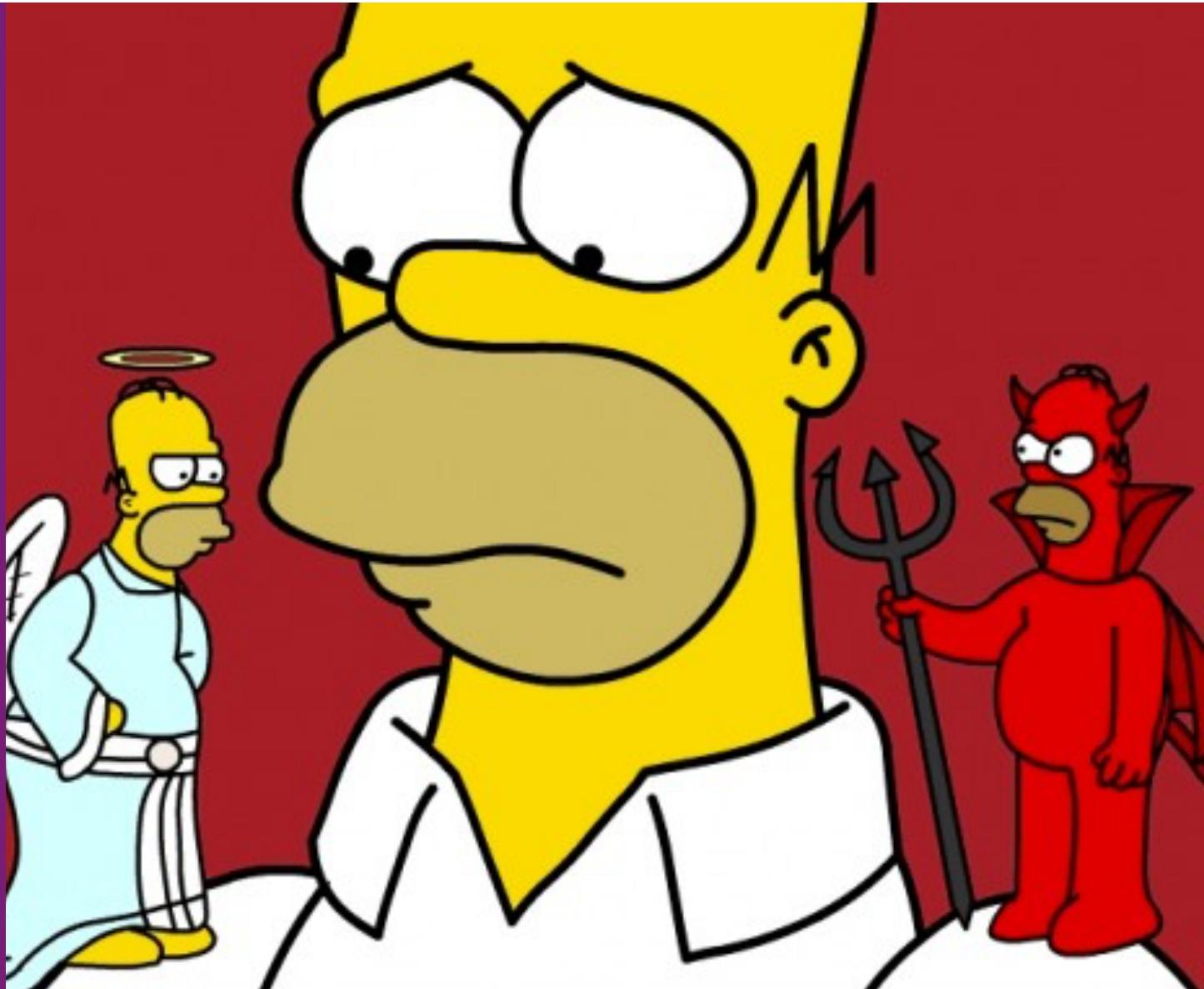
## CLOUDKIT

- 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 bad
  - No local storage
  - Platform lock  
(account & device)
  - No server logic



# CLOUDKIT

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

We will discuss how these may be less of an issue than they seem



# ENABLING CLOUDKIT IN YOUR APPLICATION

# ENABLING CLOUDKIT IN YOUR APPLICATION

The screenshot shows the Xcode interface with the project "CloudyWithAChanceOfErros" selected. The target "CloudyWithAChanceOfErros" is highlighted in the Targets list. The "Capabilities" tab is active, displaying the iCloud section. Under "Services", "Key-value storage" and "CloudKit" are checked. Under "Containers", "Use default container" is selected, and a list of available containers is shown, including "iCloud.cloud.uchicago.CloudyWithAChanceOfErros". A blue callout bubble points from the text "Custom containers can be shared between apps" to the "Containers" list.

Cloudy...OfErros > iPhone 7 Plus CloudyWithAChanceOfErros: Ready | Today at 9:51 PM

CloudyWithAChanceOfErros

CloudyWithAChanceOfErros

- CloudyWithACh...ros.entitlements
- AppDelegate.swift
- ViewController.swift
- Main.storyboard
- Assets.xcassets
- LaunchScreen.storyboard
- Info.plist

Products

Frameworks

General Capabilities Resource Tags Info Build Settings Build Phases Build Rules

PROJECT TARGETS

iCloud

ON

Services:

- Key-value storage
- iCloud Documents
- CloudKit

Containers:

- Use default container
- Specify custom containers

- iCloud.cloud.uchicago.CloudyWithAChanceOfErros
- iCloud.com.alicechicago.Instawatch
- iCloud.com.bennetth.loopyQ2

+ CloudKit Dashboard

Custom containers can be shared between apps

Steps:

- ✓ Add the iCloud feature to your App ID.
- ✓ Add iCloud Containers to your App ID
- ✓ Add the iCloud entitlement to your entitlements file
- ✓ Link CloudKit.framework

# ENABLING CLOUDKIT IN YOUR APPLICATION

The screenshot shows the CloudKit Configuration interface on a Mac. The sidebar on the left lists sections: SCHEMA, PUBLIC DATA, PRIVATE DATA, and SHARED DATA. Under SCHEMA, 'Record Types' is selected, highlighted with a blue arrow pointing from the title bar. The main pane displays 'Record Types' with a single entry: 'Users' (1 Public Record). A large blue callout box with rounded corners is positioned over the 'Users' section, containing the text 'Default container is created on iCloud'. To the right of the main pane, there's a preview area showing a table for 'Users' with columns for Created, Modified, Security, Indexes, Metadata Indexes, and Records. The preview also includes a 'Field Name' and 'Field Type' section with a 'Add Field...' button.

CloudyWithA... ▾

Record Types

SCHEMA

- Record Types
- Security Roles
- Subscription Types

PUBLIC DATA

- User Records
- Default Zone
- Usage

PRIVATE DATA

- No private zones

SHARED DATA

- No shared zones

Apple Inc.

Sort by Name ▾

Users

1 Public Record

Created: May 2 2017 9:55 PM Modified: May 2 2017 9:55 PM Security: Custom

Indexes: 0 Metadata Indexes: 0 Records: 1

Field Name Field Type Index

Add Field...

Andrew Binkowski ▾

Default container is created on iCloud

# 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

The screenshot shows the CloudKit Dashboard interface. On the left, a sidebar lists various sections: SCHEMA (Record Types, Security Roles, Subscription Types), PUBLIC DATA (User Records, Default Zone, Usage), PRIVATE DATA (No private zones), SHARED DATA (No shared zones), and ADMIN (Team, API Access, Deployment, Performance, Settings). The main area is titled "Record Types" and shows "Users" with "1 Public Record". A sub-section for "User" is displayed with fields for Created (May 2 2017 9:55 PM) and Indexes (0). There is also a section for adding new fields.

CloudyWithA... Record Types

SCHEMA

Sort by Name ▾

Record Types

Security Roles

Subscription Types

PUBLIC DATA

User Records

Default Zone

Usage

PRIVATE DATA

No private zones

SHARED DATA

No shared zones

ADMIN

Team

API Access

Deployment

Performance

Settings

Environment: DEVELOPMENT

Apple Inc.

Users

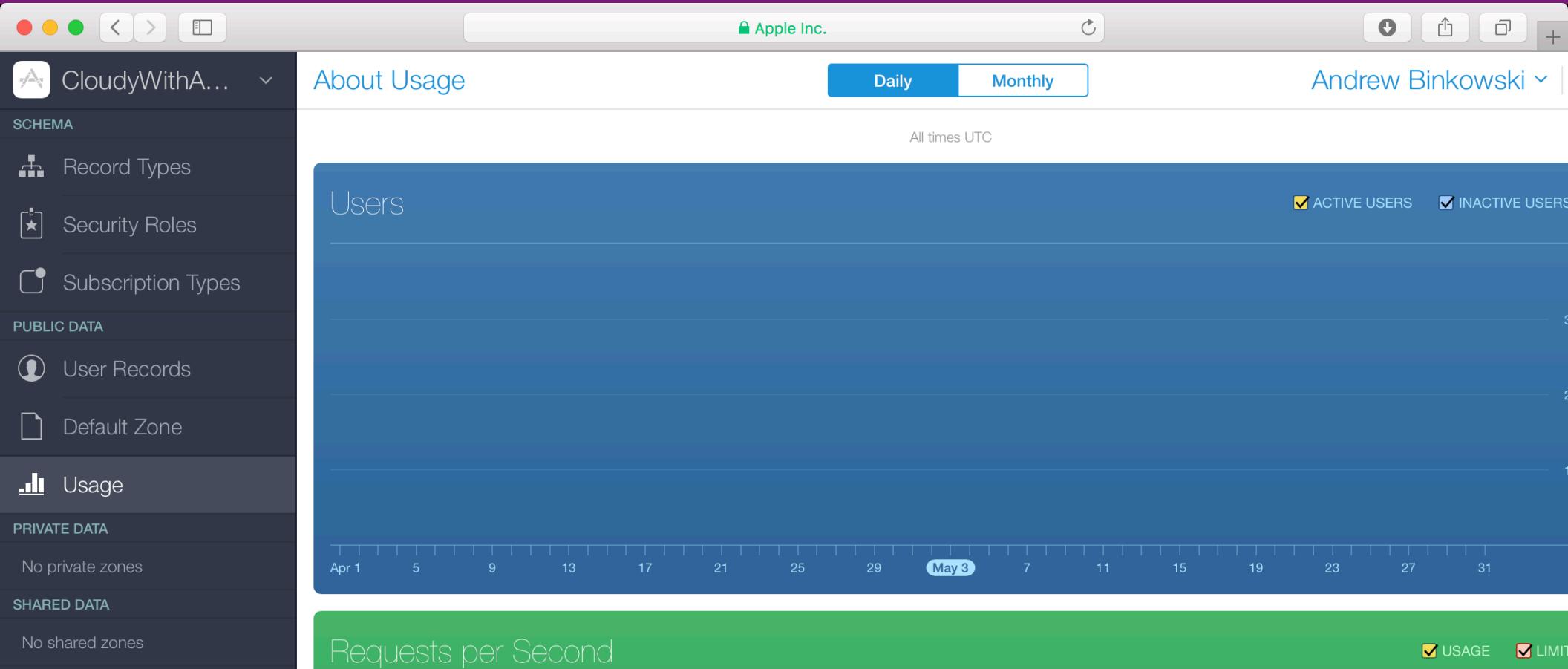
Created: May 2 2017 9:55 PM

Indexes: 0

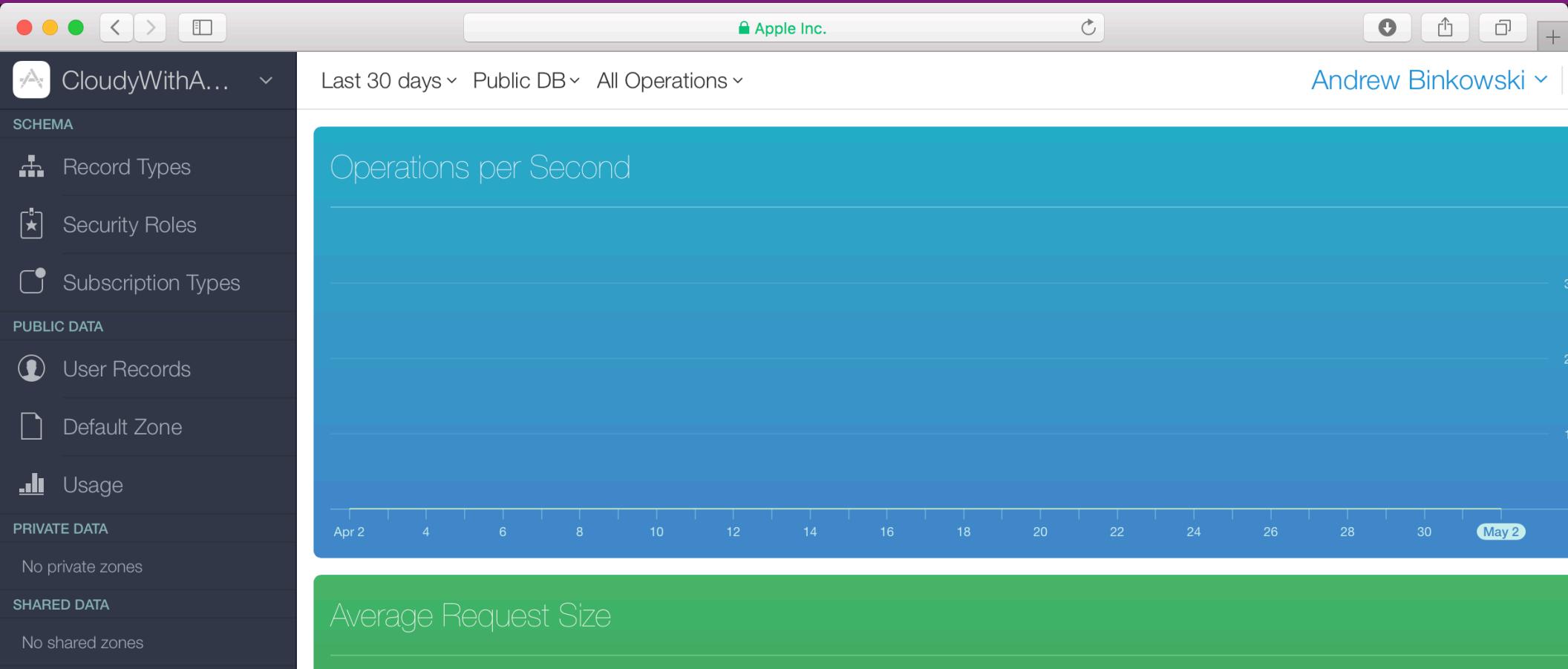
Field Name

Add Field...

# ENABLING CLOUDKIT IN YOUR APPLICATION



# ENABLING CLOUDKIT IN YOUR APPLICATION

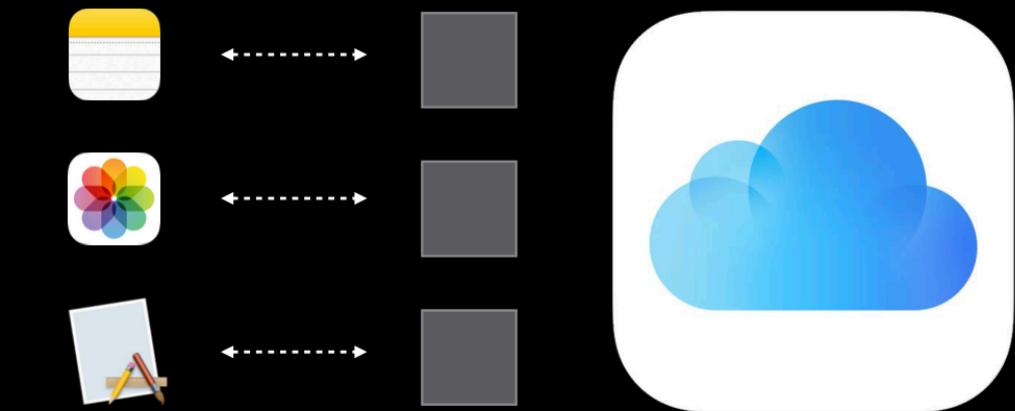


# CONTAINER

# CLOUDKIT OBJECTS

## CONTAINER

- CKContainer
  - One container per application
  - Data segregation
  - User encapsulation
  - Managed by the developer via portal
  - Can be shared between apps



# CLOUDKIT

## CONTAINER

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



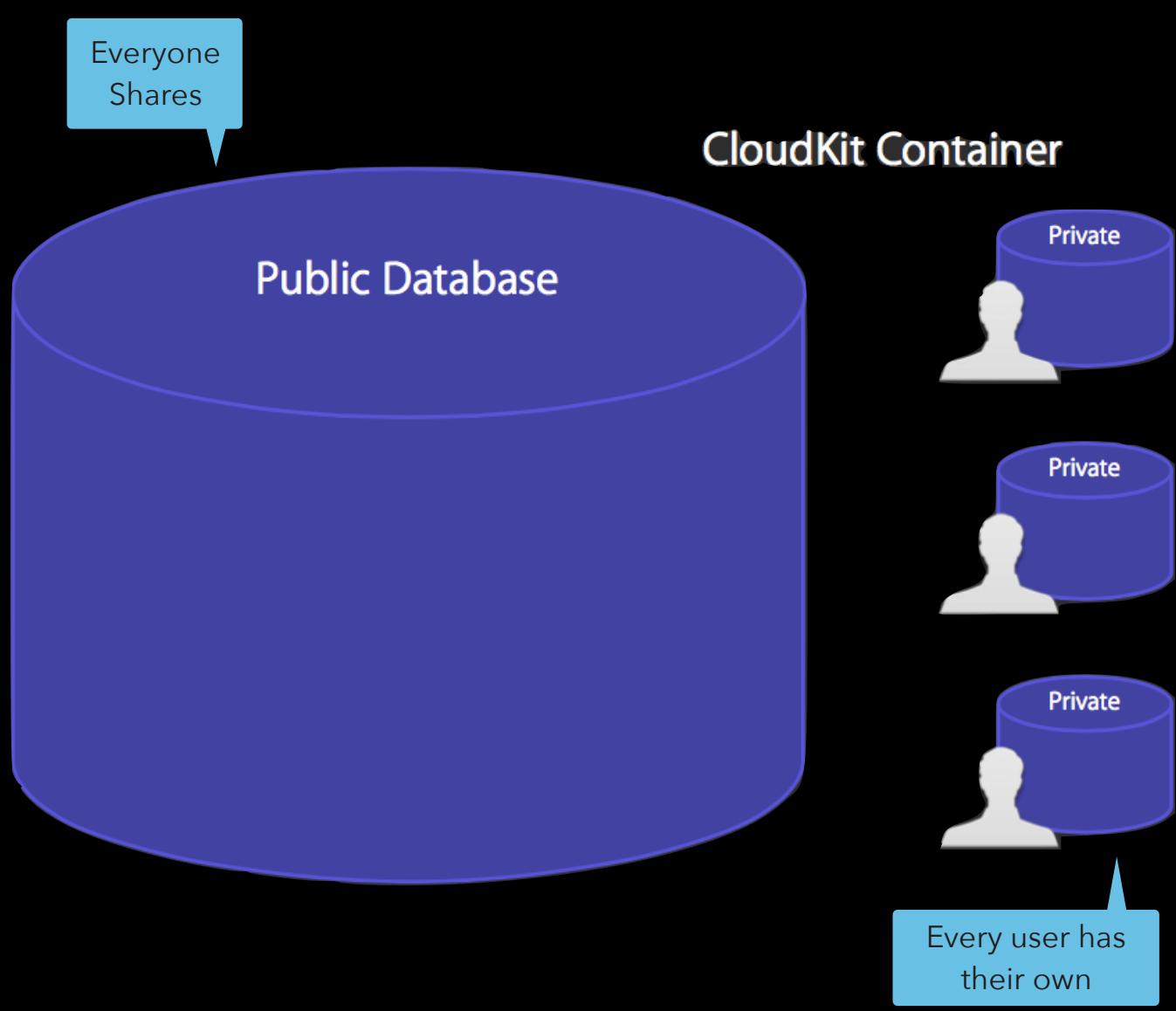
It's just that easy

# DATABASE

# CLOUDKIT OBJECTS

## DATABASE

- CKDatabase
  - Public
  - Private
  - Shared



# 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

```
// MARK: - CloudKit

let container: CKContainer = CKContainer.default()

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

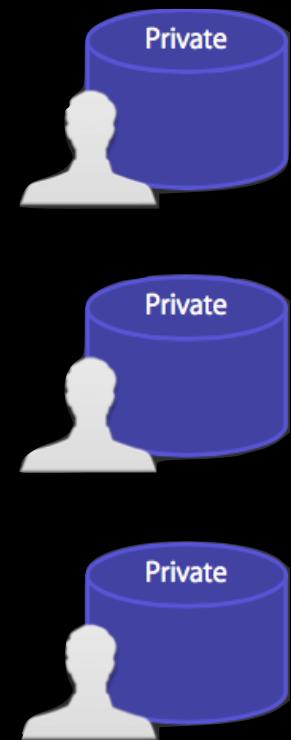
## CLOUDKIT OBJECTS

### DATABASE

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

Public Database

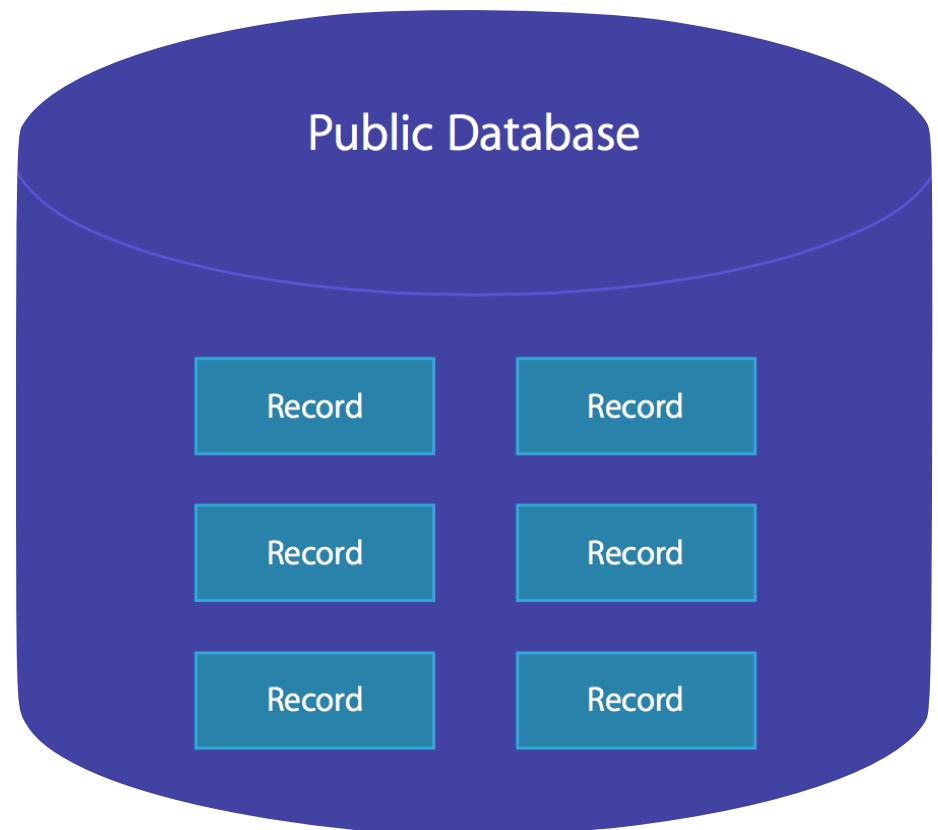
CloudKit Container



# CLOUDKIT OBJECTS

## DATABASE

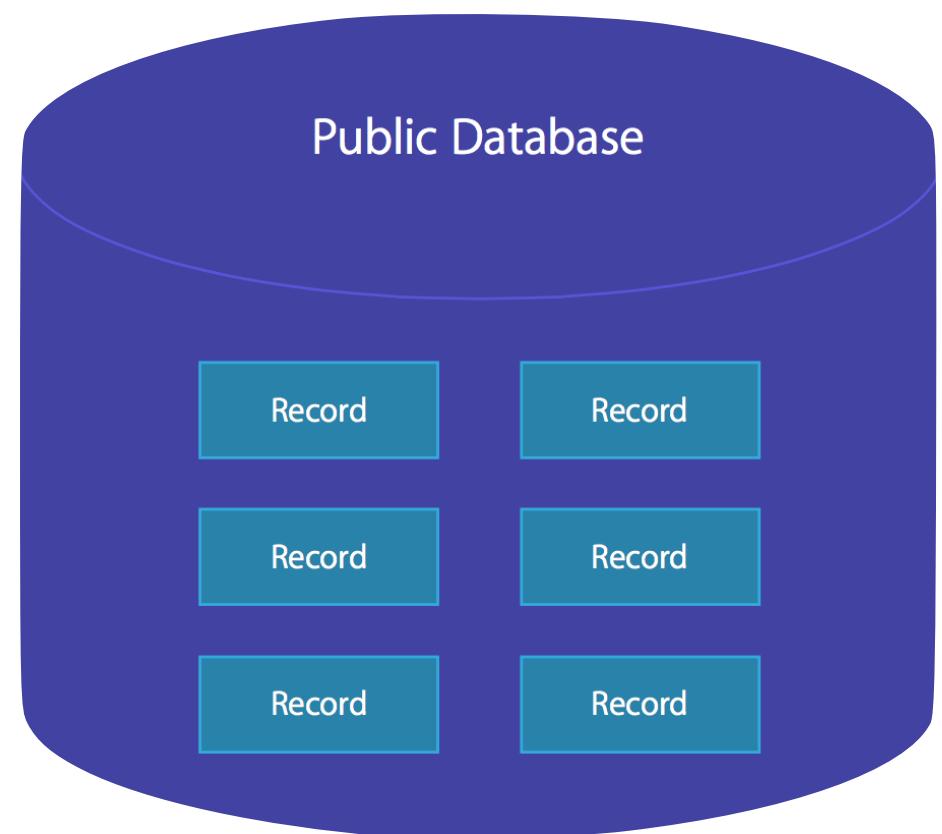
- Structured Data
  - Wraps key/value pairs
  - Types and references
  - Just-in-time schema
  - Metadata
    - Created, edited, etc.



# CLOUDKIT OBJECTS

## DATABASE

- Schema is built on-demand
- You can edit it in the console
- Changing it may be painful
  - Requires on-device work

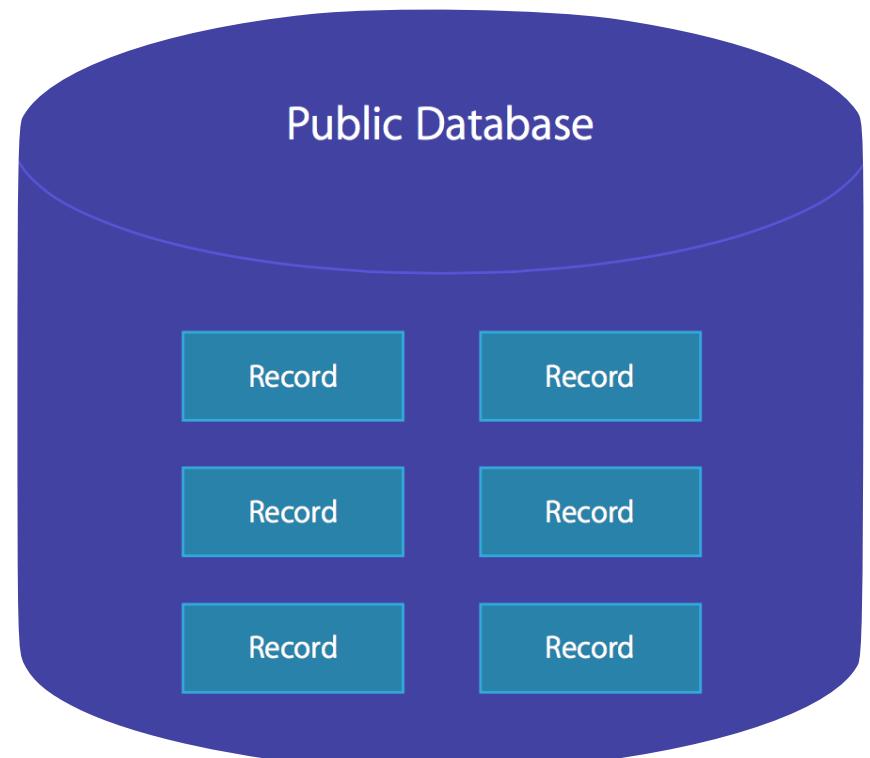


# RECORDS

# CLOUDKIT OBJECTS

## RECORDS

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



## CLOUDKIT OBJECTS RECORDS

- Meta data is used heavily

```
/* 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 object(forKey key: String) -> CKRecordValue?
```

# CLOUDKIT OBJECTS

## RECORDS

The screenshot shows the CloudKit Record Types interface. On the left, a sidebar navigation includes sections for SCHEMA (Record Types, Security Roles, Subscription Types), PUBLIC DATA (User Records, Default Zone, Usage), PRIVATE DATA (No private zones), SHARED DATA (No shared zones), ADMIN (Team, API Access, Deployment, Performance), and a search bar for "Andrew Binkowski". The main area is titled "Record Types" with a "Sort by Name" dropdown. It lists one record type, "joke", which has 17 unused indexes. Below the record types is a section for "Users" with 1 public record.

**joke**

Field Name	Field Type	Index
question	String	<input checked="" type="checkbox"/> Sort <input checked="" type="checkbox"/> Query <input checked="" type="checkbox"/> Search
rating_negative	Int(64)	<input checked="" type="checkbox"/> Sort <input checked="" type="checkbox"/> Query
rating_positive	Int(64)	<input checked="" type="checkbox"/> Sort <input checked="" type="checkbox"/> Query
response	String	<input checked="" type="checkbox"/> Sort <input checked="" type="checkbox"/> Query <input checked="" type="checkbox"/> Search

[Add Field](#)

# CLOUDKIT OBJECTS

## RECORDS

Screenshot of the CloudKit Records interface in a web browser.

The title bar shows the URL [CloudyWithAChanceOfErrors](#) and the page title **NewRecordType**.

The sidebar on the left contains the following sections:

- SCHEMA**: Record Types, Security Roles, Subscription Types.
- PUBLIC DATA**: User Records, Default Zone (selected), Usage.
- PRIVATE DATA**: No private zones.
- SHARED DATA**: No shared zones.
- ADMIN**: Team, API Access, Deployment.

The main content area displays a search bar with the placeholder "Sort by question" and a search icon. It features a large blue header bar with the text "NewRecordType". Below this, there is a large empty white area with a small blue plus sign icon in the top right corner. A large gray document icon is centered in the middle of the screen. To the right of the document icon, the text "CloudyWithAChanceOfErrors has no public “joke” records" is displayed. At the bottom right, a blue "New Record" button is visible.

# 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

Save the record

## CLOUDKIT OBJECTS

### RECORDS

- Create a joke record in our view controller

```
import UIKit
import CloudKit

class ViewController: UIViewController {

    // MARK: - CloudKit
    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: "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

```
import CloudKit

class ViewController: UIViewController {

    // MARK: - CloudKit
    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?")
    }

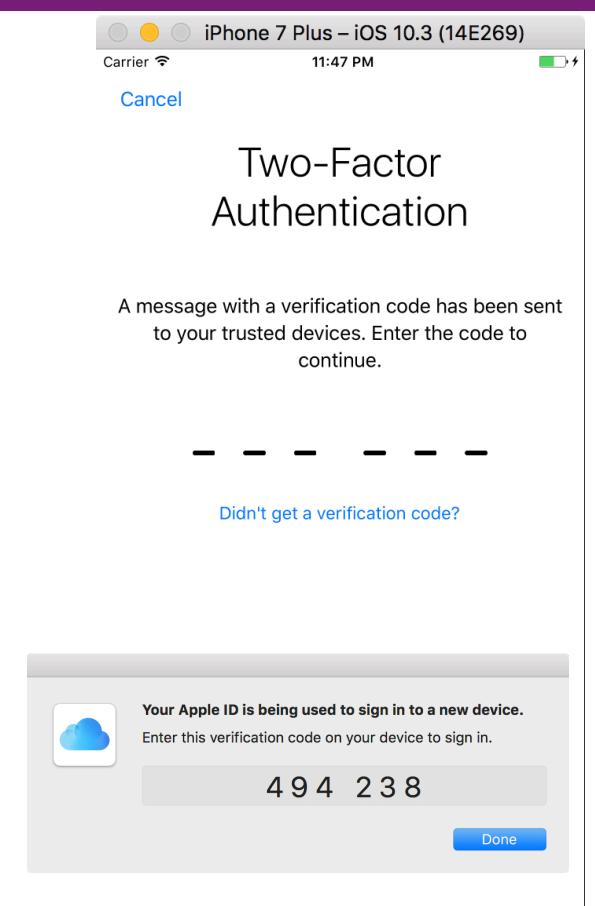
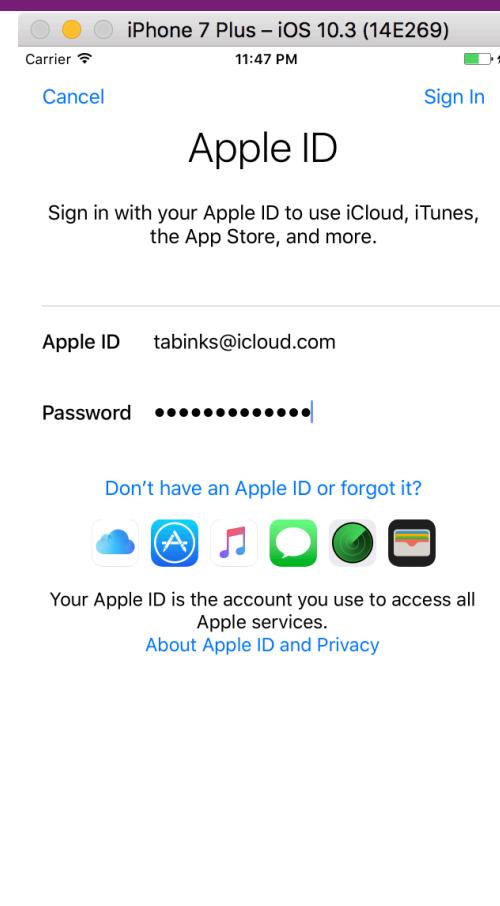
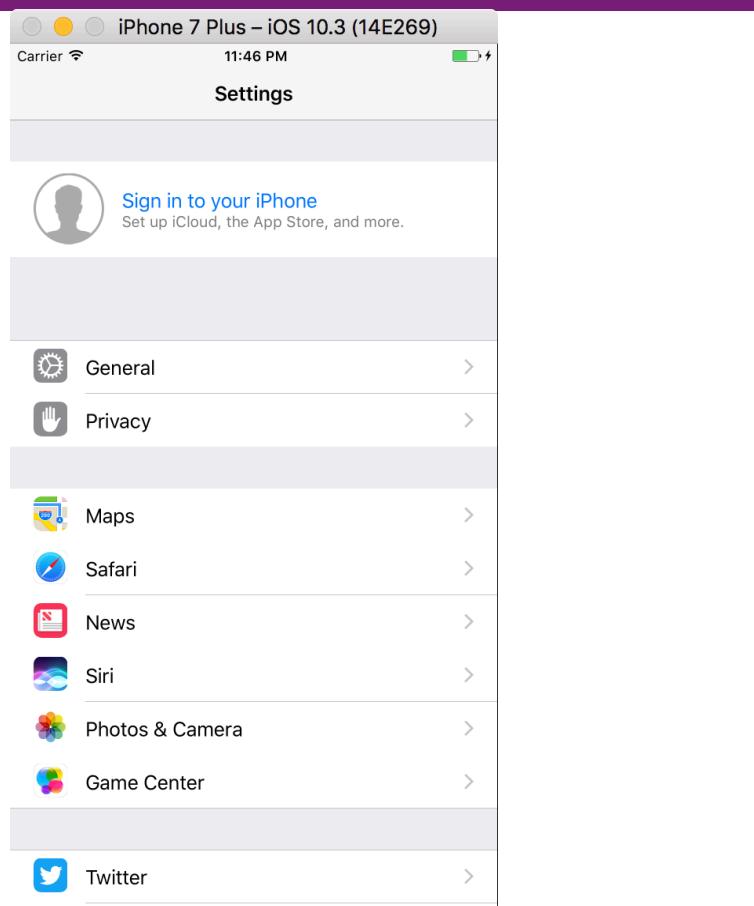
    /// Create a joke record and save to iCloud using CloudKit
    /// - parameter joke: Funny string
    /// - remark: Error handling leaves something to be desired
    func saveJoke(_ joke: String) {
        let record = CKRecord(recordType: "joke")
        record.setValue(joke, forKey: "text")
      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

Your first CloudKit  
error

🐞: This request requires an authenticated account

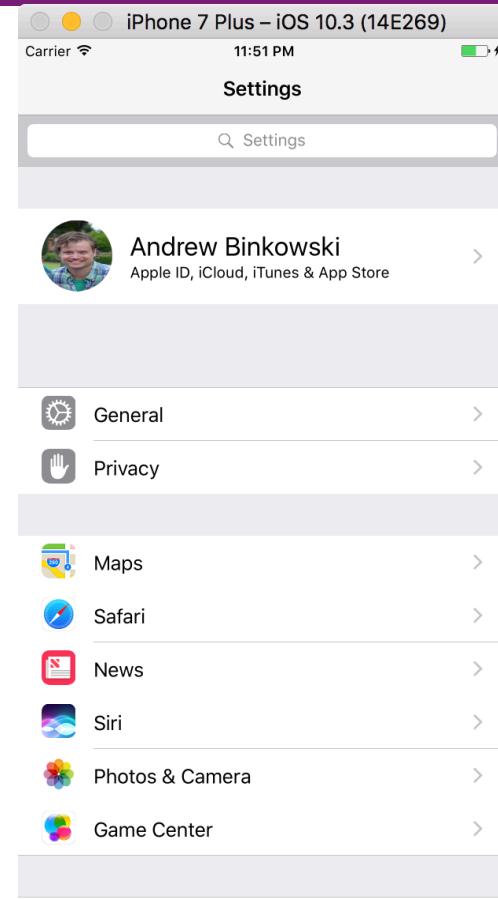
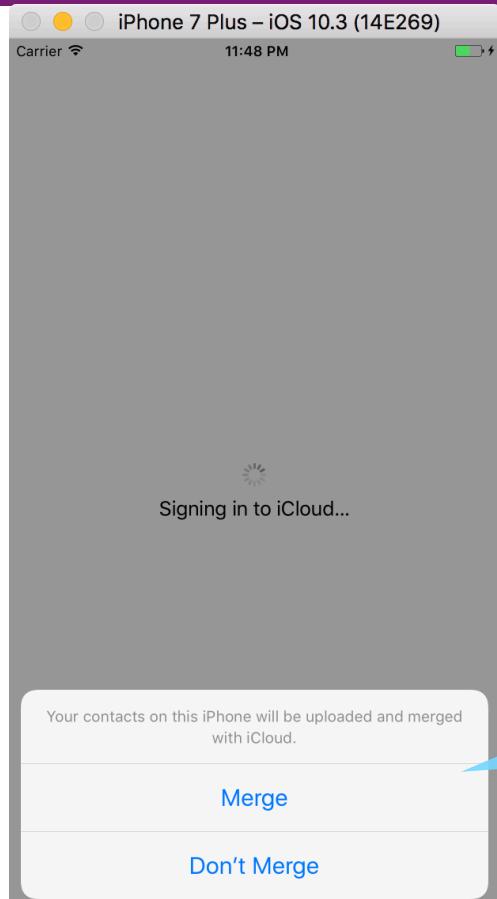
# CLOUDKIT OBJECTS

## RECORDS



# 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 Swift code editor and a terminal window below it.

**Code Editor:**

```
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)")
}
}
```

**Terminal Output:**

```
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

The screenshot shows the CloudKit Dashboard interface on a Mac. The left sidebar contains navigation links for Schema, Record Types, Security Roles, Subscription Types, Public Data (User Records, Default Zone, Usage), Private Data (No private zones), Shared Data (No shared zones), Admin (Team, API Access, Deployment), and Deployment.

The main area displays a record for a "joke" type. The search bar shows "joke". The record title is "Why did the chicken cross the road?". The record name is "1275F7C8-05DF-456B-B804-E91E458C6393".

Record details:

Created:	Created By:	Modified:	Modified By:
May 3 2017 12:05 AM	_23d865322080d4185ad95d...	May 3 2017 12:08 AM	_7b892f2a3eeadd6afc77ff...

Record fields:

Name	Type
question	String
rating_negative	Int(64)
rating_positive	Int(64)
response	String

The "question" field contains "Why did the chicken cross the road?". The "rating\_negative" field contains "0". The "rating\_positive" field contains "0". The "response" field contains "To get to the other side."

# CLOUDKIT OBJECTS

## RECORDS

The screenshot shows the CloudKit Record Editor interface. On the left, a sidebar lists various sections: SCHEMA, PUBLIC DATA, PRIVATE DATA, SHARED DATA, ADMIN, and DEPLOYMENT. Under SCHEMA, Record Types is selected. In the main area, a record named "joke" is displayed. The record contains the question "Why did the chicken cross the road?" and the response "To get to the other side." The record was created on May 3, 2017, at 12:05 AM by a user with a specific UUID, and modified on the same day at 12:08 AM by another user.

CloudyWithA... joke Sort by question

Record Name:  
1275F7C8-05DF-456B-B804-E91E458C6393

Created: May 3 2017 12:05 AM  
Created By: \_23d865322080d4185ad95d...  
Modified: May 3 2017 12:08 AM  
Modified By: \_7b892f2a3eeadd6afc77ff...

question	Why did the chicken cross the road?	String
rating_negative	0	Int(64)
rating_positive	0	Int(64)
response	To get to the other side.	String

Change schema from console or in code

# CLOUDKIT OBJECTS

## RECORDS

The screenshot shows the CloudKit Record Editor interface. On the left is a sidebar with navigation links for Record Types, Security Roles, Subscription Types, PUBLIC DATA (User Records, Default Zone, Usage), PRIVATE DATA (No private zones), SHARED DATA (No shared zones), ADMIN (Team, API Access, Deployment, Performance, Settings), and Environment (set to DEVELOPMENT). The main area displays a record named "Joke" with 1 Public Record. The "Users" section shows 2 Public Records. The "Joke" record details are as follows:

Created:	Modified:	Security:
May 2 2017 11:34 PM	May 3 2017 12:09 AM	Default ▾
Indexes:	Metadata Indexes:	Records:
12	7 ▾	1 ▾

The schema table lists fields and their properties:

Field Name	Field Type	Index
question	String	<input checked="" type="checkbox"/> Sort <input checked="" type="checkbox"/> Query <input checked="" type="checkbox"/> Search
rating_negative	Int(64)	<input checked="" type="checkbox"/> Sort <input checked="" type="checkbox"/> Query
rating_positive	Int(64)	<input checked="" type="checkbox"/> Sort <input checked="" type="checkbox"/> Query
response	String	<input checked="" type="checkbox"/> Sort <input checked="" type="checkbox"/> Query <input checked="" type="checkbox"/> Search
comments	String List ▾	<input checked="" type="checkbox"/> Query <input checked="" type="checkbox"/> Search

A blue callout bubble with an arrow points from the bottom-left towards the schema table, containing the text: "Change schema from console or in code".

# CLOUDKIT OBJECTS

## RECORDS

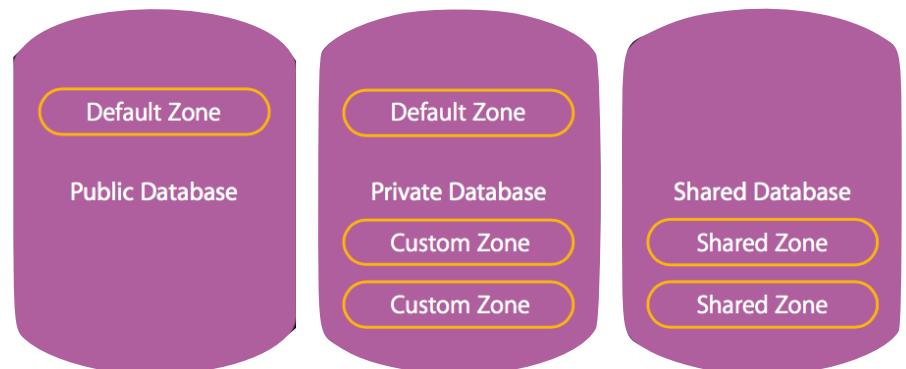
The screenshot shows the CloudKit Dashboard interface. On the left, there's a sidebar with various navigation options like Schema, Record Types, Security Roles, etc. A large blue callout bubble points from the bottom-left towards the 'Record Types' section, containing the text "Added new field 🤘". The main area displays the 'Record Types' page for the 'joke' record type. The 'joke' record type has 18 unused indexes and 1 public record. It was created on May 2, 2017, at 11:34 PM, and modified on May 3, 2017, at 12:15 AM. The security level is set to 'Default'. The table below lists the fields: 'comments' (String List, indexed for Query and Search), 'question' (String, indexed for Sort, Query, and Search), 'rating\_negative' (Int(64), indexed for Sort and Query), 'rating\_positive' (Int(64), indexed for Sort and Query), and 'response' (String, indexed for Sort and Query).

Field Name	Field Type	Index
comments	String List	<input checked="" type="checkbox"/> Query <input checked="" type="checkbox"/> Search
question	String	<input checked="" type="checkbox"/> Sort <input checked="" type="checkbox"/> Query <input checked="" type="checkbox"/> Search
rating_negative	Int(64)	<input checked="" type="checkbox"/> Sort <input checked="" type="checkbox"/> Query
rating_positive	Int(64)	<input checked="" type="checkbox"/> Sort <input checked="" type="checkbox"/> Query
response	String	<input checked="" type="checkbox"/> Sort <input checked="" type="checkbox"/> Query

# 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



# 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.
// CKRecordZoneID

#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
                           zone:(CKRecordZoneID *)zone;

@property (nonatomic, readonly, strong) NSString *recordName;
@property (nonatomic, readonly, strong) CKRecordZoneID *zone;

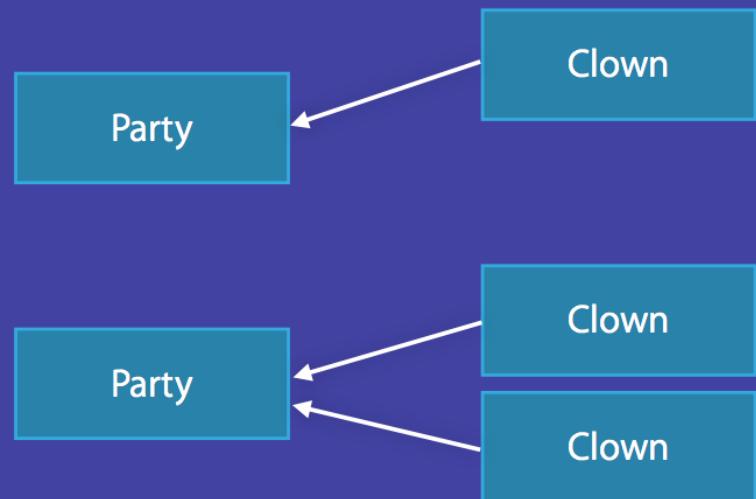
@end
```

# CLOUDKIT OBJECTS

## RECORDS

- CKReference
  - Server Understands Relationship
  - Cascade Deletes
  - Dangling Pointers
  - Back References

Public Database



# CLOUDKIT API

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

- CKReferenceAction
  - None
  - Self-delete

What happens on delete

# CLOUDKIT API

ANSWER

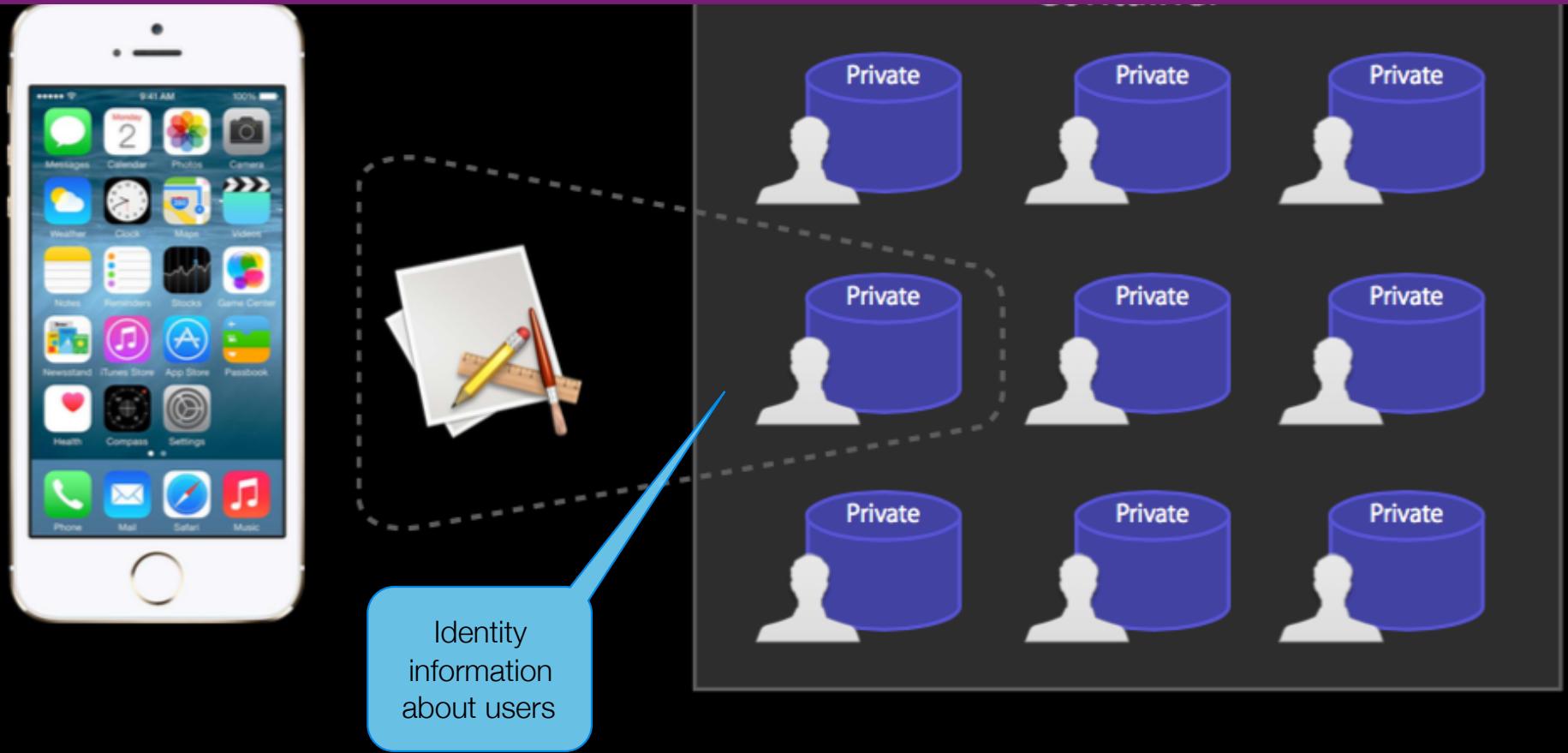
Question		Attributes			
Users	3 Records	Indexes:	Metadata Indexes:	Index Size if Deployed:	Cost
	9 Unused Indexes	Created: <b>Mar 11 2015 13:55</b>	Modified: <b>Mar 11 2015 13:55</b>	Security: <b>Default</b> ▾	
	9 Unused Indexes	Indexes: <b>6</b>	Metadata Indexes: <b>7</b> ▾	Index Size if Deployed: <b>0 bytes</b>	
Attribute Name		Attribute Type	Index	Cost	
Affirmative		Int(64)	<input checked="" type="checkbox"/> Sort <input checked="" type="checkbox"/> Reference <input checked="" type="checkbox"/> Query	+105%	+105%
Location		Location	<input checked="" type="checkbox"/> Sort <input checked="" type="checkbox"/> Query	+105%	+105%
Question		Reference	<input checked="" type="checkbox"/> Query	+105%	+105%
User		Reference	<input checked="" type="checkbox"/> Query	+105%	+105%

# USER RECORDS

## USER RECORDS

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

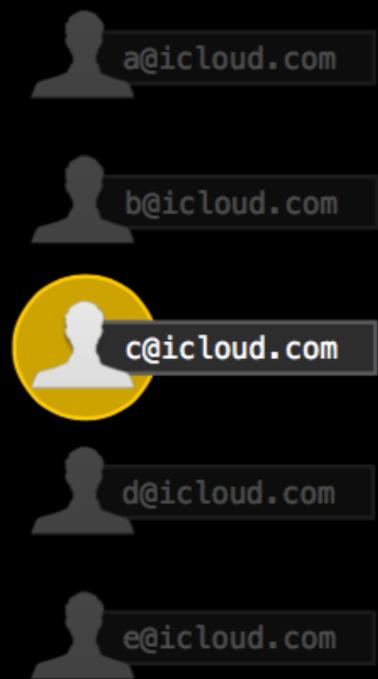
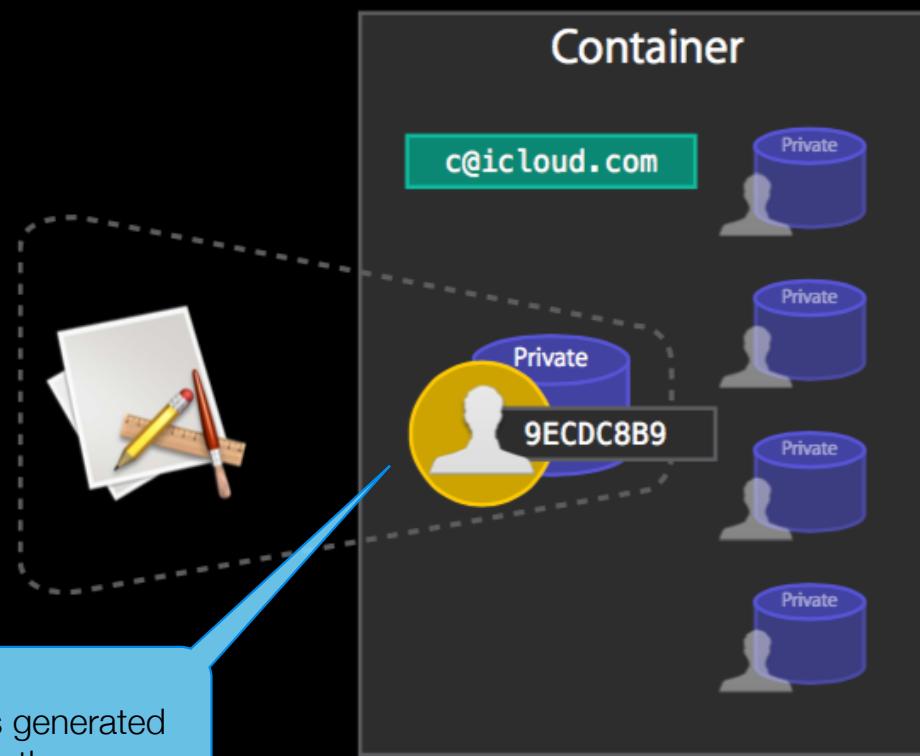
# USER RECORDS



# USER RECORDS



A unique ID is generated  
representing the user



# USER RECORDS

The screenshot shows the CloudKit Record Types interface. On the left, a sidebar lists various sections: SCHEMA (Record Types, Security Roles, Subscription Types), PUBLIC DATA (User Records, Default Zone, Usage), PRIVATE DATA (No private zones), SHARED DATA (No shared zones), and ADMIN (Team, API Access, Deployment, Performance). The main area displays 'Record Types' with a table header 'Sort by Name'. It lists two record types: 'joke' (3 Public Records) and 'Users' (2 Public Records, highlighted in blue). Below the table, a tooltip says 'User record is already created'. On the right, the 'Users' record type details are shown: Created: May 2 2017 9:55 PM, Modified: May 3 2017 12:27 AM, Security: Custom, Indexes: 0, Metadata Indexes: 1, Records: 2. An 'Add Field...' button is present. Another tooltip on the right says 'Add additional fields to customize your data; not recommended by Apple'.

CloudKit Record Types

SCHEMA

Record Types

Security Roles

Subscription Types

PUBLIC DATA

User Records

Default Zone

Usage

PRIVATE DATA

No private zones

SHARED DATA

No shared zones

ADMIN

Team

API Access

Deployment

Performance

Sort by Name

joke 16 Unused Indexes  
3 Public Records

Users 2 Public Records

User record is already created

Created: May 2 2017 9:55 PM Modified: May 3 2017 12:27 AM Security: Custom

Indexes: 0 Metadata Indexes: 1 Records: 2

Add Field...

Add additional fields to customize your data; not recommended by Apple

# USER RECORDS

- To associate a User with other data
  - Create a User record type that has a reference to the UserId
  - Store any other data user



The screenshot shows a database management interface. On the left, a sidebar titled "Record Types" lists two entries: "joke" (16 Unused Indexes, 3 Public Records) and "Users" (2 Public Records). The main area displays a "Users" record. At the top right of this area are icons for delete (-) and add (+). Below these are fields for "Created" (May 2 2017 9), "Indexes" (0), and "Field Name" (an empty input field). A blue "Add Field..." button is located at the bottom right of the main area.

Record Types	
joke	16 Unused Indexes 3 Public Records
Users	2 Public Records

Users

Created: May 2 2017 9

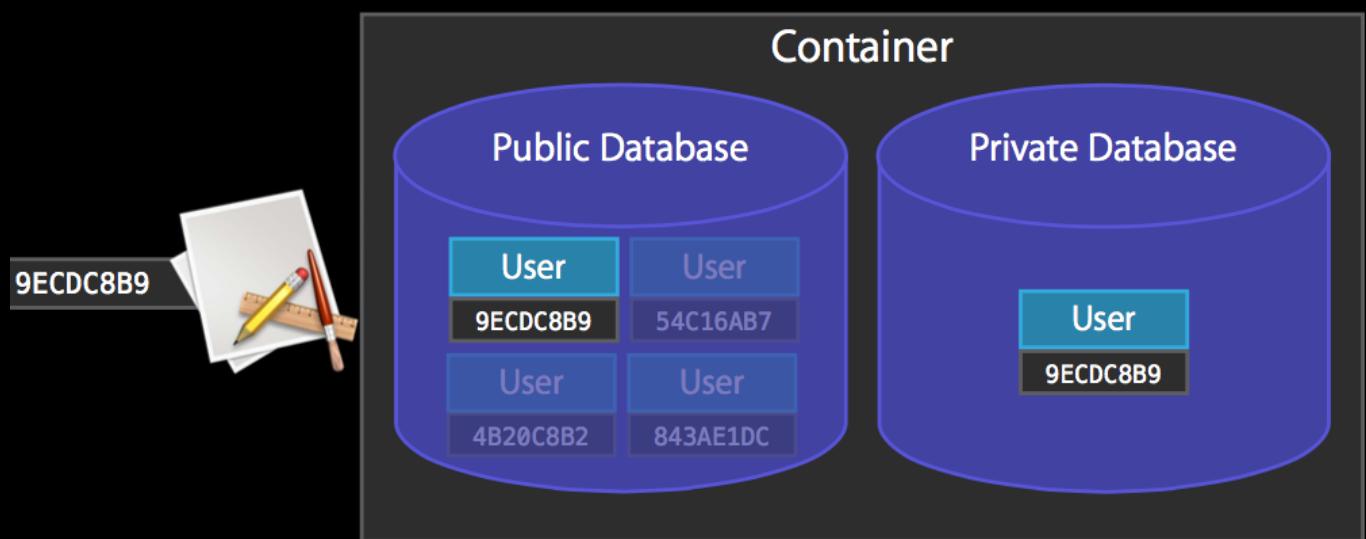
Indexes: 0

Field Name

Add Field...

# CloudKit User Accounts

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



## USER RECORDS

```
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.fetchUserRecordID(recordID, error:in
    "23d865322080d4185ad95db121c80663"

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

The screenshot shows a web-based application interface for managing user records. On the left, a sidebar menu lists various sections: SCHEMA (Record Types, Security Roles, Subscription Types), PUBLIC DATA (User Records, Default Zone, Usage), PRIVATE DATA (No private zones), SHARED DATA (No shared zones), and ADMIN (Team, API Access, Deployment, Performance). A blue callout bubble points from the 'User Records' item in the PUBLIC DATA section towards the central content area. The main content area has a header 'User Records' with a search icon. Below the header, two user records are listed, both titled 'No Name'. The first record's ID is '\_23d865322080d4185ad95db121c80663'. The second record's ID is '\_7b892f2a3eeadd6afc77ff7158f69d9'. To the right of the records, a detailed view is open for the first record, showing its ID as 'No Name' and its creation and modification dates as 'May 2 2017 11:51 PM'. The entire screenshot is framed by a thick blue border.

CloudyWithA... ▾

User Records

No Name  
\_23d865322080d4185ad95db121c80663

No Name  
\_7b892f2a3eeadd6afc77ff7158f69d9

No Name

Record Name:  
\_23d865322080d4185ad95db121c80663

Created:  
May 2 2017 11:51 PM

Modified:  
May 2 2017 11:51 PM

"\_23d865322080d4185ad95db121c80663"

Andrew Binkowski ▾

User Records

## 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  
“cloud.uchicago.CloudyWith  
AChanceOfErros” 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  
“cloud.uchicago.CloudyWith  
AChanceOfErros” 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)
    })
}

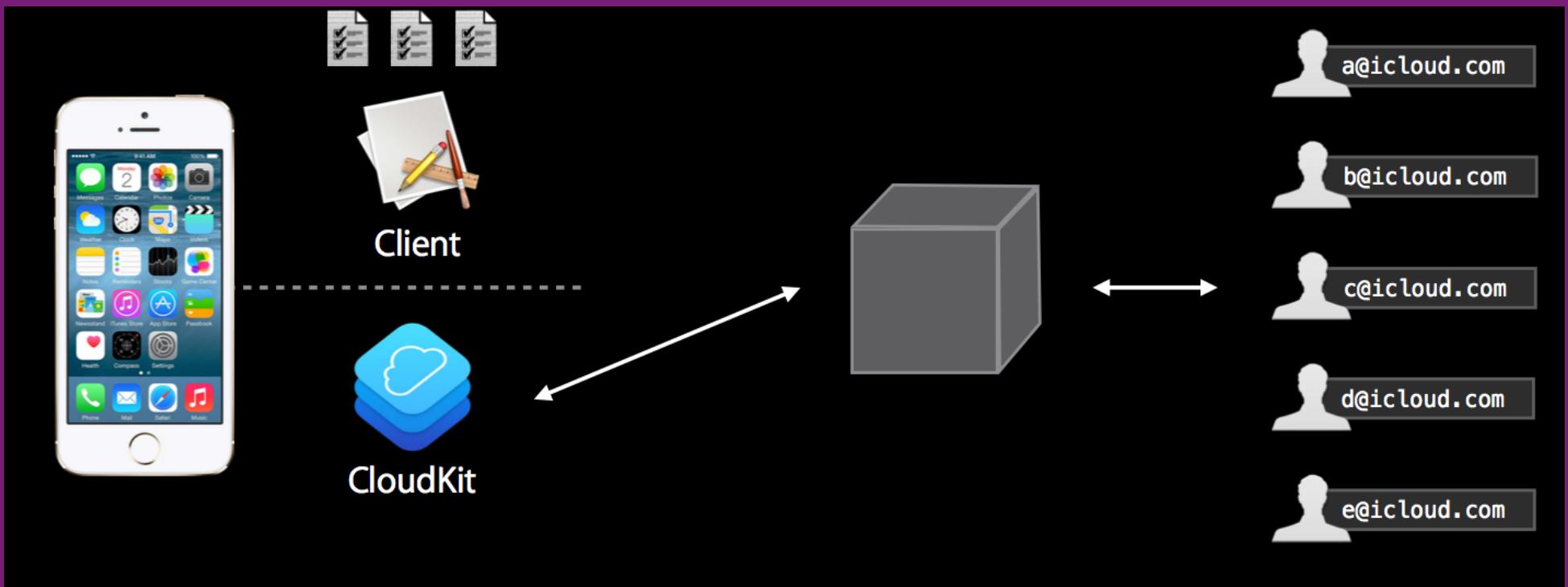
CKContainer.default().requestApplicationPermission(.userDiscoverability) { (status, error) in
    CKContainer.default().fetchUserRecordID { (record, error) in
        CKContainer.default().discoverUserIdentity(withUserRecordID: record!, completionHandler: { (userID, error) in
            userName = (userID?.nameComponents?.givenName)!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

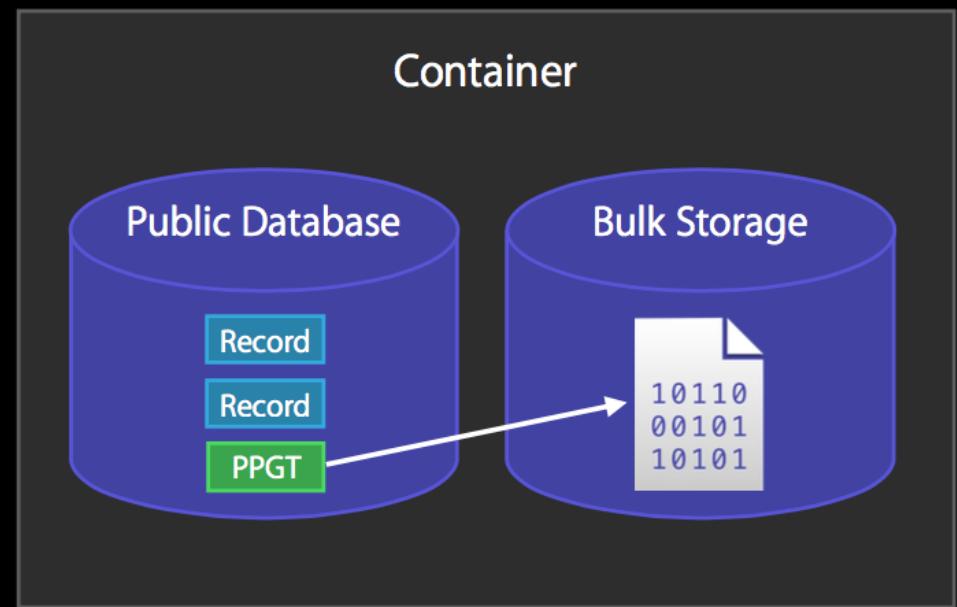


- You can get all friends who use the same app (discover users api)

# ASSETS

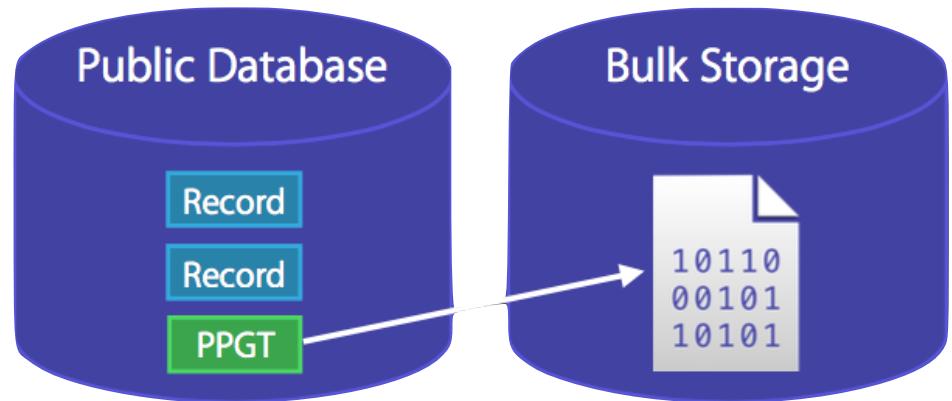
# CLOUDKIT API

- CKAssets is bulk storage for CloudKit



# CLOUDKIT API

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



CloudyWithAC... ▾ Andrew Binkowski ▾ | ?

SCHEMA

- Record Types
- Security Roles
- Subscription Types

PUBLIC DATA

- User Records
- Default Zone
- Usage

PRIVATE DATA

No private zones

SHARED DATA

No shared zones

ADMIN

- Team
- API Access

## Record Types

Sort by Name ▾

Record Type	Indexes
joke	17 Unused Indexes 16 Public Records
Users	3 Public Records

trash +

**joke**

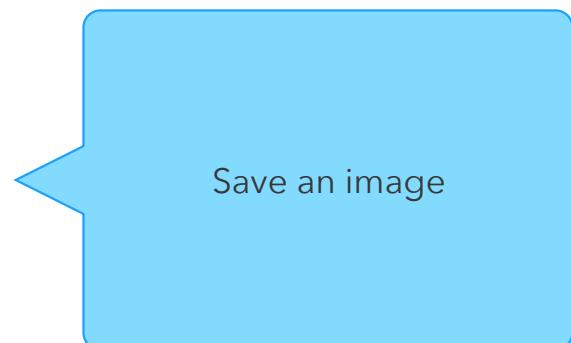
Created: May 2 2017 11:34 PM Modified: May 3 2017 3:01 PM Security: Default ▾

Indexes: 12 Metadata Indexes: 7 Records: 16

Field Name	Field Type	Index
audioFile	Asset	None
comments	String List	<input checked="" type="checkbox"/> Query <input checked="" type="checkbox"/> Search
question	String	<input checked="" type="checkbox"/> Sort <input checked="" type="checkbox"/> Query
rating_negative	Int(64)	<input checked="" type="checkbox"/> Search <input checked="" type="checkbox"/> Sort <input checked="" type="checkbox"/> Query

# CLOUDKIT API

```
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)
}
```



Save an image

# CLOUDKIT API

```
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
- NSOperation based API
  - Customizable and fully featured
- Convenience API for single record interactions
  - May be all you ever need

# 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
  - Saving a modified record

# 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

- 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 `CKRecordID` of the user
open func getJokesByCurrentUser(_ recordID: CKRecordID) {
    // The user is a reference, so we need to fetch it
    let reference = CKReference(recordID: recordID,
                                 database: nil)
    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: \(String(describing: error))")
    }
}
```

# 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: \(String(describing: 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
        //}
    }
}
```

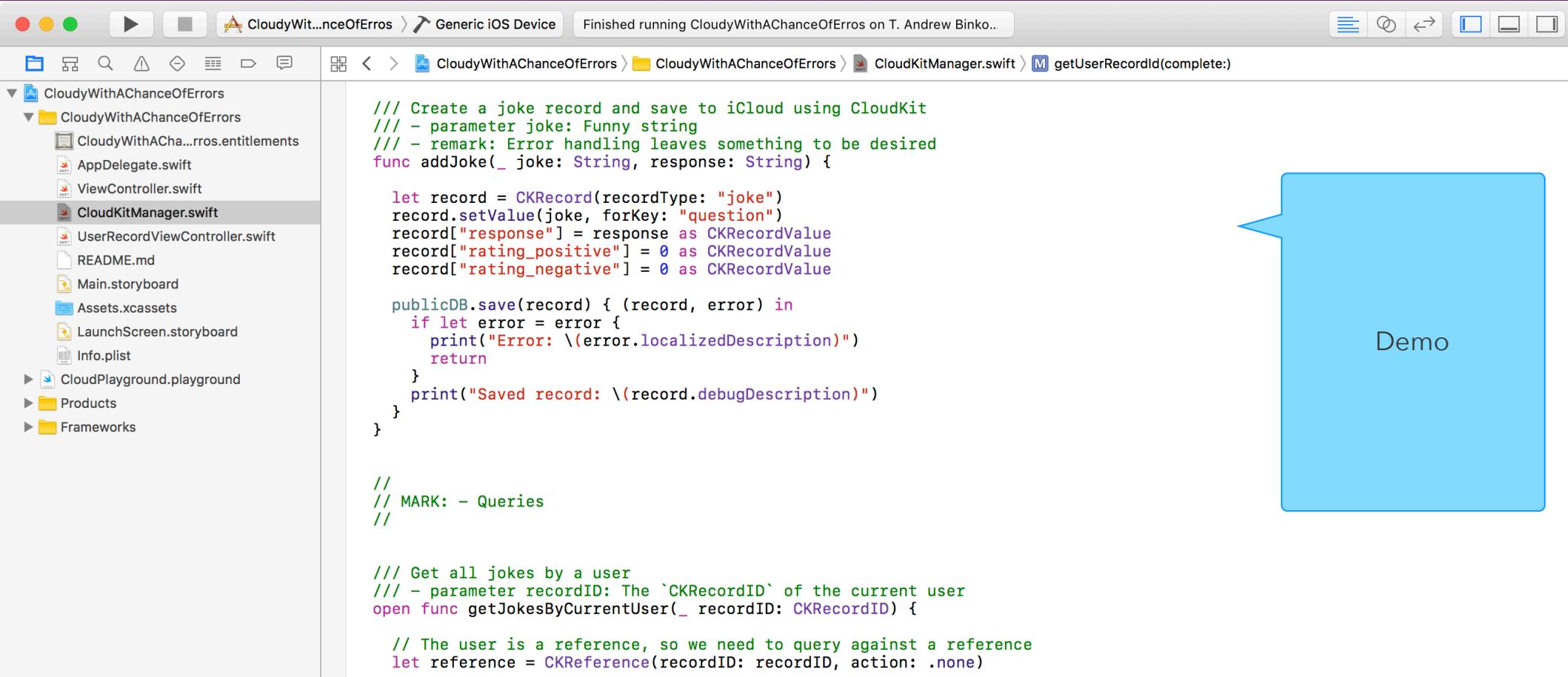
# QUERIES

```
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

CloudyWithACh...rros.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) {

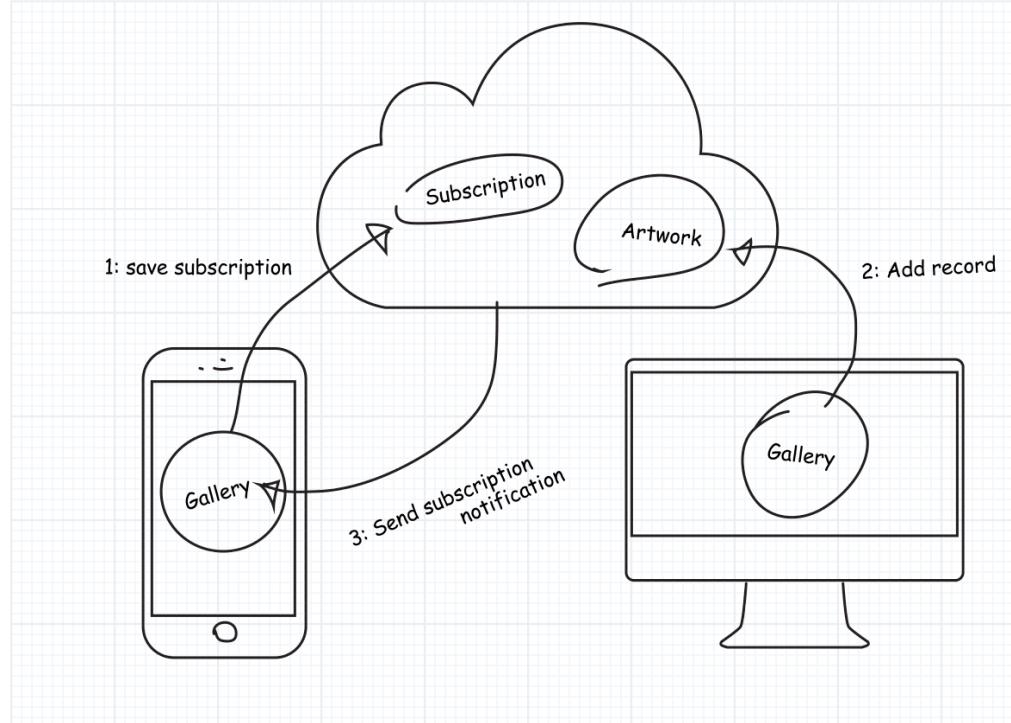
    // The user is a reference, so we need to query against a reference
    let reference = CKReference(recordID: recordID, action: .none)
```

Demo

# SUBSCRIPTIONS

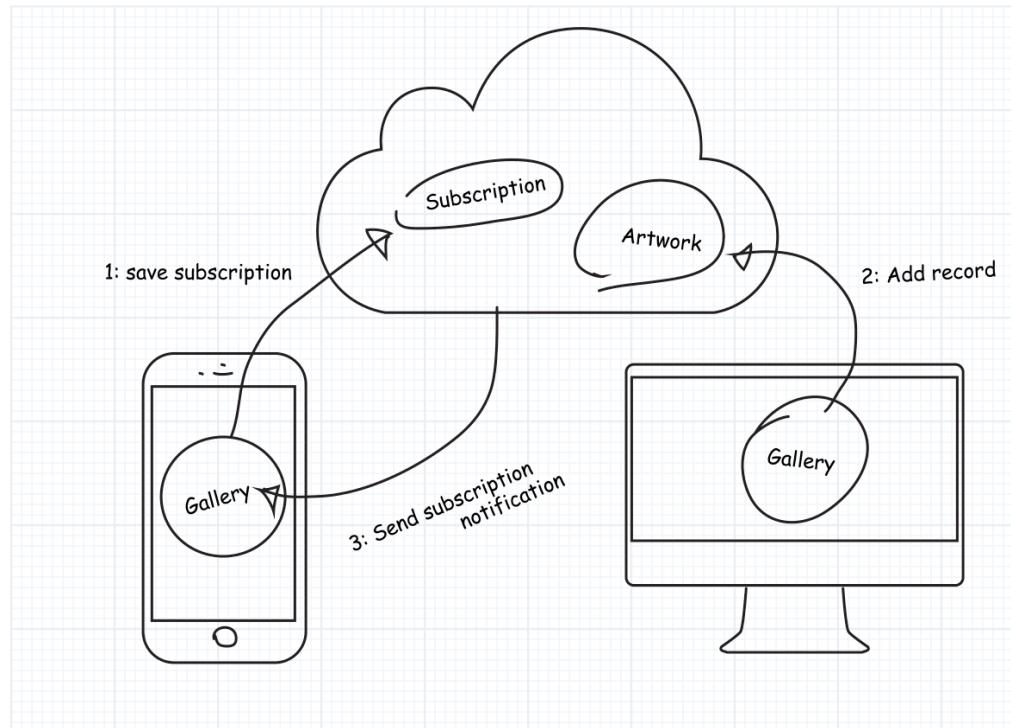
# SUBSCRIPTIONS

- 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 running your query
  - in the background
  - after every record save
  - push results



# SUBSCRIPTIONS

- The server running your query
  - in the background
  - after every record save
  - push results

The screenshot shows the CloudyWithAC... application interface. On the left is a sidebar with the following menu items:

- SCHEMA
  - Record Types
  - Security Roles
  - Subscription Types
- PUBLIC DATA
  - User Records
  - Default Zone
  - Usage
- PRIVATE DATA
  - No private zones
- SHARED DATA
  - No shared zones
- ADMIN
  - Team
  - API Access
  - Deployment
  - Performance
  - Settings

The main content area is titled "Subscription Types" and shows two entries:

Subscription Types	
Sort by Record Type ▾	
joke	Signature: 686bfb79f17b47aef6f7ee1a7fb7bc82
joke	Signature: 7c401b3dddc25d4979ef61e55040e5da

A modal window is open for the first "joke" entry, showing the following details:

Signature:	686bfb79f17b47aef6f7ee1a7fb7bc82
RecordType	joke
Trigger	INSERT
Criteria	[empty]

# SUBSCRIPTIONS

- CKQuerySubscription
  - RecordType
  - NSPredicate
  - Push
- Push delivered via Apple Push Service Augmented payload

Subscription Types	
Sort by Record Type ▾	
joke	Signature: 686fb79f17b47aef6f7ee1a7fb7bc82
joke	Signature: 7c401b3dddc25d4979ef61e55040e5da
RecordType	joke
Trigger	INSERT
Criteria	

CHEAPEST, EASIEST WAY TO ADD PUSH TO AN APP

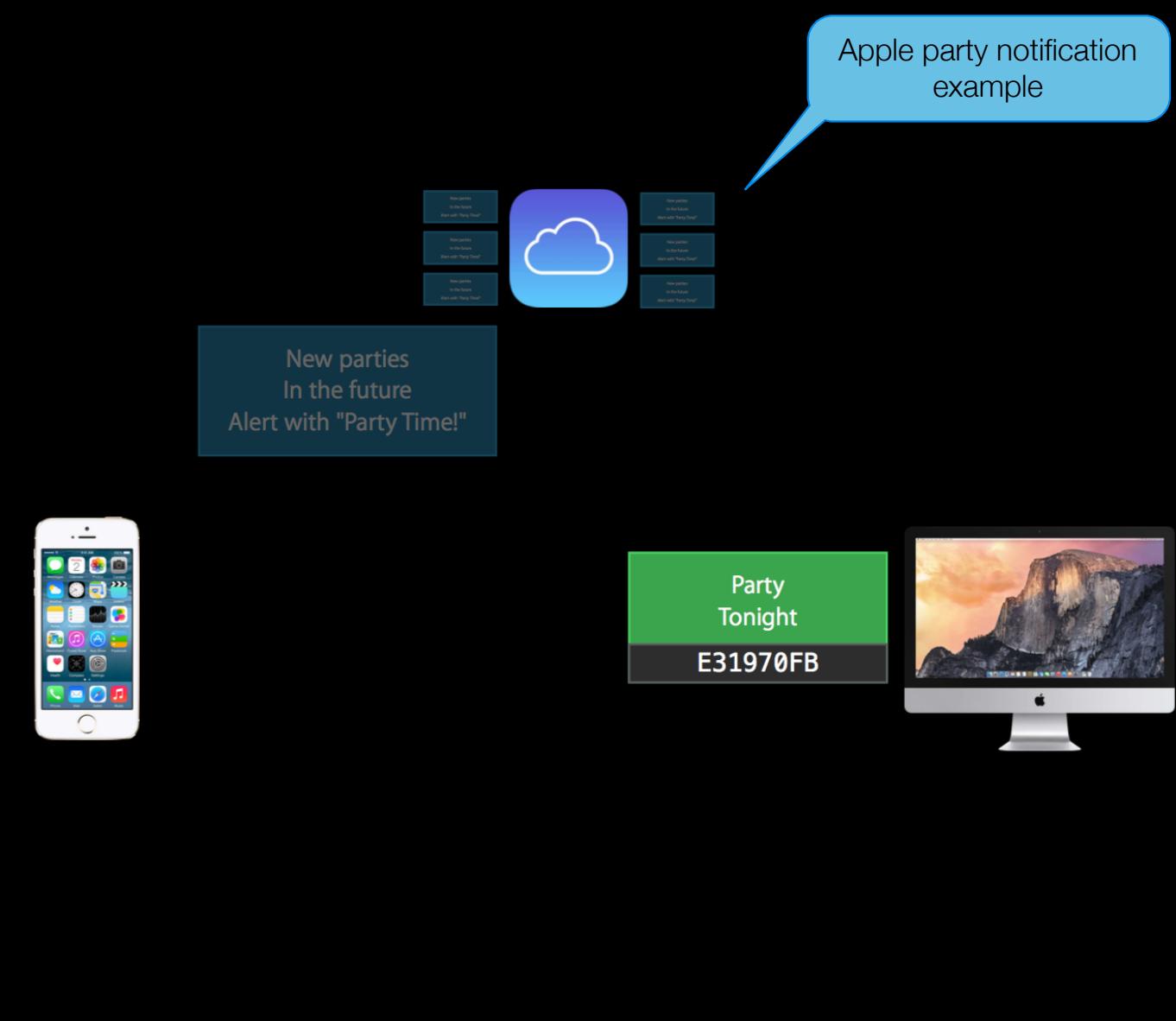
# SUBSCRIPTIONS

- Subscription in action



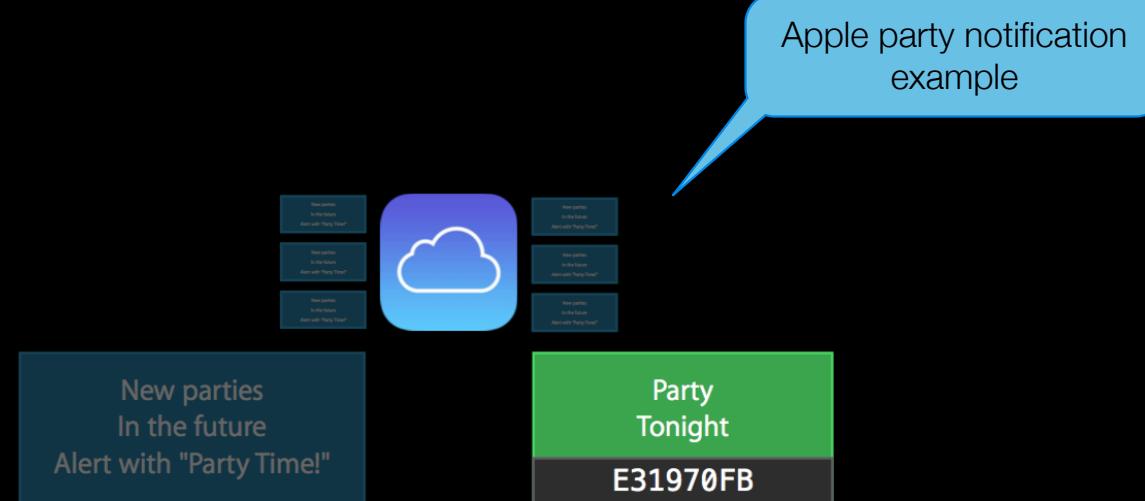
# SUBSCRIPTIONS

- Subscription in action



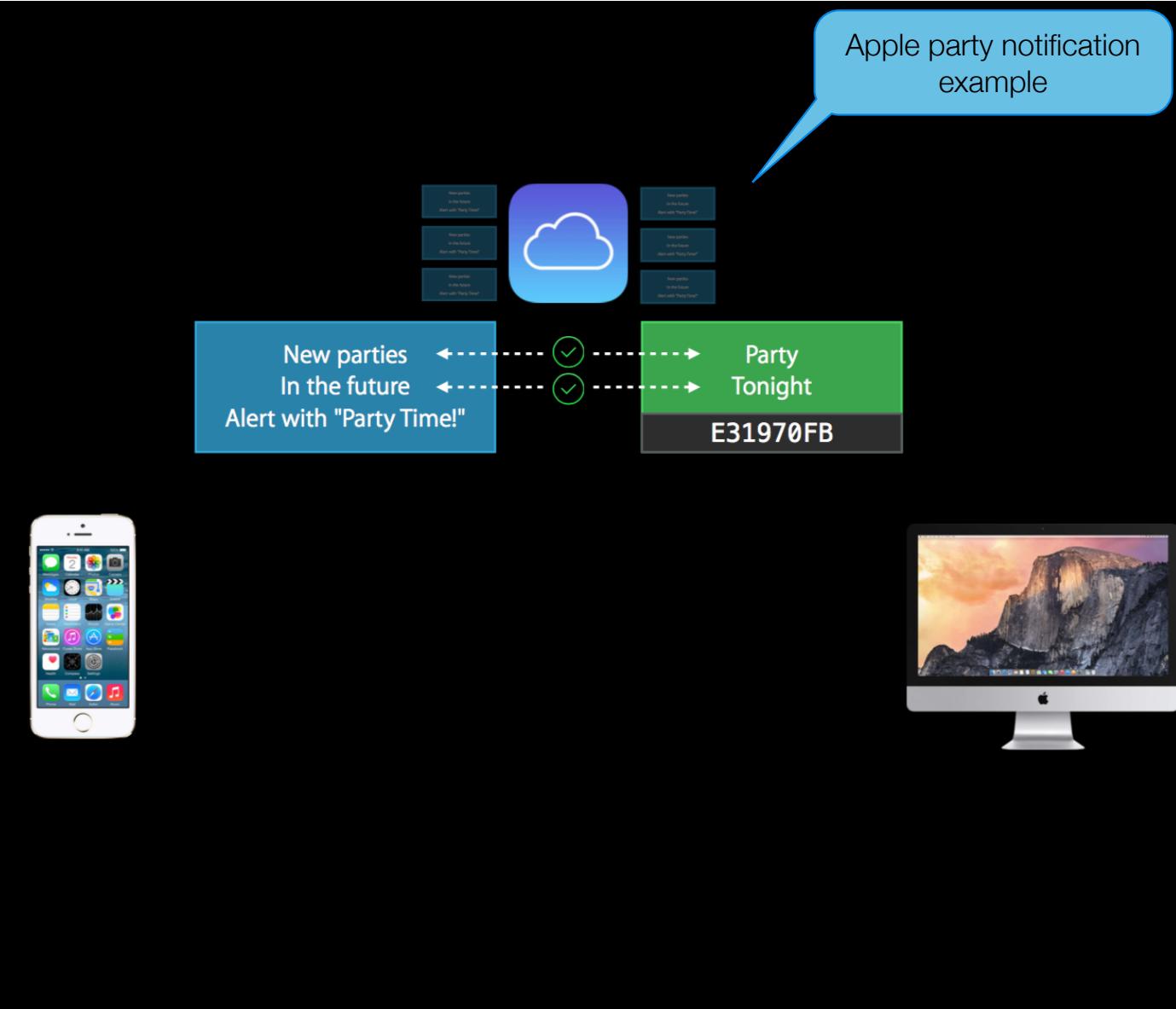
# SUBSCRIPTIONS

- Subscription in action



# SUBSCRIPTIONS

- Subscription in action



# SUBSCRIPTIONS

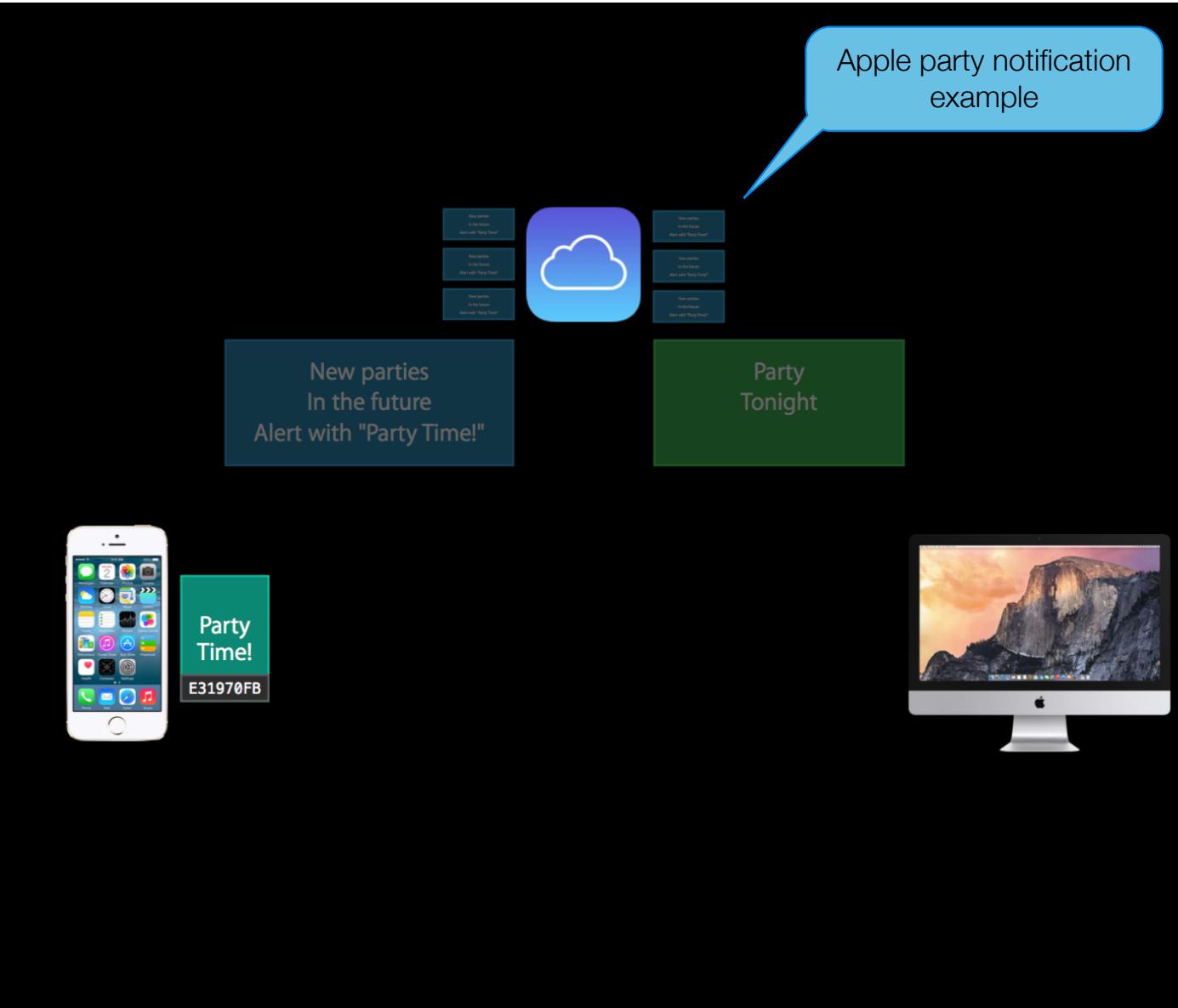
- Subscription in action

Apple party notification example



# SUBSCRIPTIONS

- Subscription in action



# SUBSCRIPTIONS

- Subscriptions need a unique id

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

# SUBSCRIPTIONS

- Create a generic notification
- Some tricks to make your notifications seem more dynamic
  - Fake alert

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

# SUBSCRIPTIONS

- Create a subscription

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

# SUBSCRIPTIONS

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

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

# SUBSCRIPTIONS

The screenshot shows a user interface for managing cloud subscriptions. On the left, a sidebar menu includes options like 'Record Types', 'Subscription Types', 'User Records', 'Default Zone', 'Usage', 'Team', 'API Access', and 'Deployment'. A blue callout bubble points to the 'Subscription Types' option with the text 'Show all subscriptions'. The main pane displays a list of 'Subscription Types' with a single item selected: 'joke'. The 'joke' entry has a 'Signature' field containing the value '686bfb79f17b47aef6f7ee1a7fb7bc82'. The right pane provides detailed information about the 'joke' subscription, including its RecordType ('joke'), Trigger ('INSERT'), and Criteria (empty). A blue callout bubble points to the 'See details' link in the top right corner of the right-hand panel.

Show all subscriptions

CloudyWithA...

Subscription Types

SCHEMA

Record Types

Security Roles

Subscription Types

PUBLIC DATA

User Records

Default Zone

Usage

PRIVATE DATA

No private zones

SHARED DATA

No shared zones

ADMIN

Team

API Access

Deployment

Sort by Record Type

joke

Signature: 686bfb79f17b47aef6f7ee1a7fb7bc82

RecordType joke

Trigger INSERT

Criteria

See details

Apple Inc.

Andrew Binkowski

# SETUP YOUR APP TO HANDLE NOTIFICATIONS

# SUBSCRIPTIONS

- Enable push notifications

The screenshot shows the Xcode interface with the project "CloudyWithAChanceOfErrors" selected. The "Capabilities" tab is active. Under the "Push Notifications" section, the switch is set to "ON". Below this, there are sections for Game Center, Wallet, Siri, Apple Pay, In-App Purchase, Maps, Personal VPN, Network Extensions, and Background Modes. The Background Modes section is expanded, showing a list of modes with checkboxes. The "Background fetch" and "Remote notifications" checkboxes are checked. A note at the bottom says "Steps: ✓ Add the Required Background Modes key to your info.plist file".

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

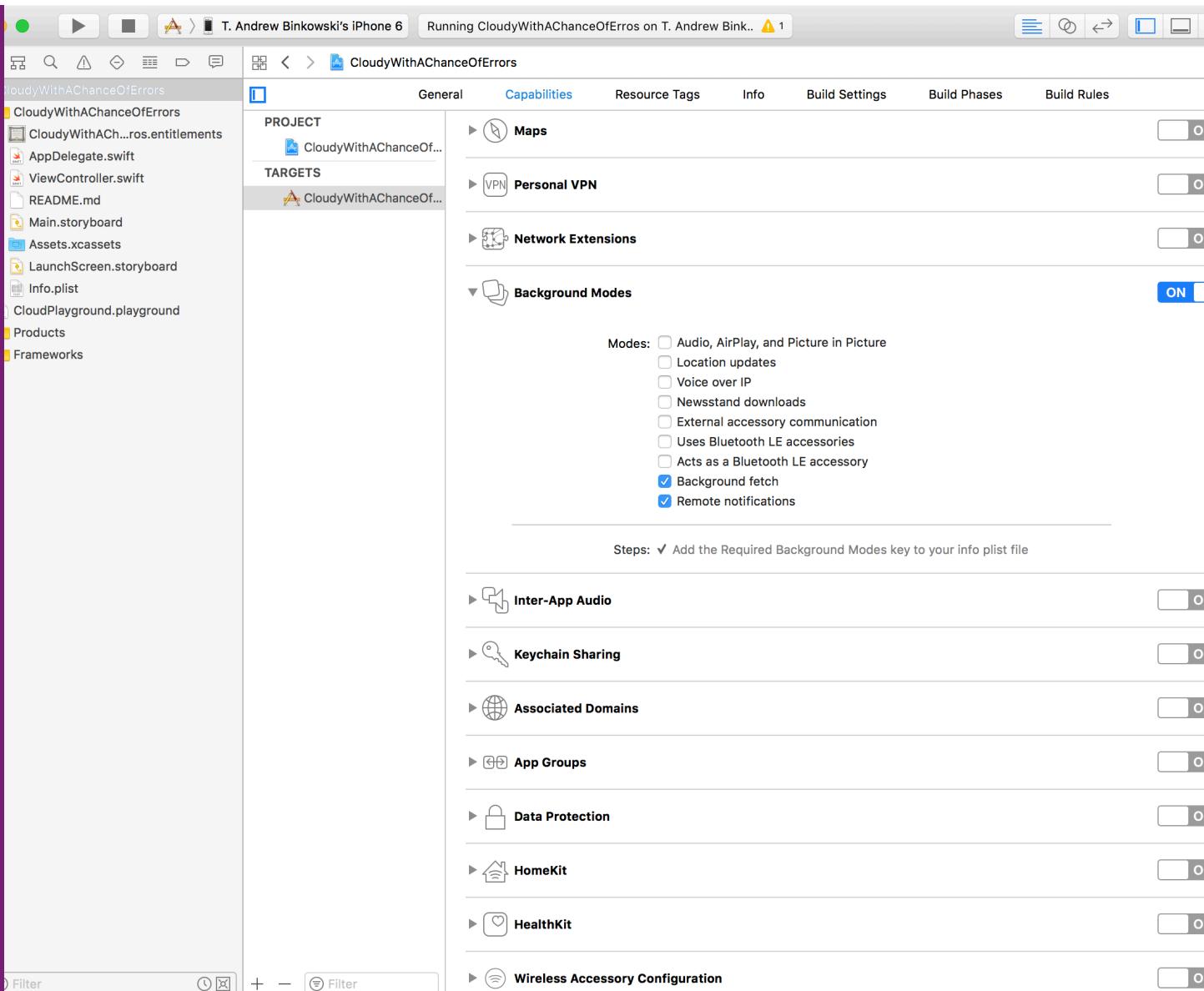
Modes:

- 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 Required Background Modes key to your info.plist file

# SUBSCRIPTIONS

- Request background modes



# SUBSCRIPTIONS

Request permissions

```
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 without permission

**"CloudyWithAChanceOfErrors  
" Would Like to Send You  
Notifications**

Notifications may include alerts, sounds, and icon badges. These can be configured in Settings.

Don't Allow

Allow

# SUBSCRIPTIONS

```
UNUserNotificationCenter.current().delegate = self

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

if let notification = launchOptions?[UIApplicationLaunchOptionsKey.remoteNotification] as? [String: AnyObject] {
    let aps = notification["aps"] as! [String: AnyObject]
    print("APS: \(aps)")
}
return true
}
```

- Handling cloud kit notifications
  - applicationWillFinishLaunching
  - didReceiveRemoteNotification

# 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)  
    }  
}
```

- Handling cloud kit notifications
  - applicationsDidFinishLaunching
  - didReceiveRemoteNotification

# SUBSCRIPTIONS

- Notification can be silent
  - Push only data
- Pay attention to state of application similar to other notifications

```
//  
// AppDelegate.swift  
// CloudyWithAChanceOfErrors  
//  
// Created by T. Andrew Binkowski on 5/2/17.  
// Copyright © 2017 T. Andrew Binkowski. All rights reserved.  
  
import UIKit  
import UserNotifications  
import CloudKit  
  
@UIApplicationMain  
class AppDelegate: UIResponder, UIApplicationDelegate, UNUserNotificationCenterDelegate {  
  
    var window: UIWindow?  
  
    func application(_ application: UIApplication,  
                     didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey] = [:]) -> 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("id1")  
        CloudKitManager.sharedInstance.registerSilentSubscriptionsWithIdentifier("id2")  
        //configureUserNotificationsCenter()  
  
        if let notification = launchOptions?[UIApplicationLaunchOptionsKey.remoteNotification] {  
            // 2  
            let aps = notification["aps"] as! [String: AnyObject]  
            print("APS: \(aps)")  
            // 3  
            //((window?.rootViewController as? UITabBarController)?.selectedIndex ?? 0) + 1  
        }  
        return true  
    }  
}
```

# SUBSCRIPTIONS

- Subscriptions can have advanced behavior over notifications
- Delivery behavior

```
/  
/ MARK: - Notifications Support  
  
// Create an alert to show if the application is active and receives a local  
// notification  
// - parameter notification: The `UILocalNotification` received  
unc showAlertForNotification(userInfo: [NSObject : AnyObject]) {  
  
    // Do not show unless the application is active  
    guard UIApplication.sharedApplication().applicationState == .Active else { return }  
  
    // Create the alert  
    let alertController = UIAlertController(title: "Received Notification",  
                                           message: userInfo.description,  
                                           preferredStyle: .Alert)  
  
    // Create cancel action that does nothing  
    let cancelAction = UIAlertAction(title: "Cancel", style: .Cancel, handler: nil)  
    alertController.addAction(cancelAction)  
  
    // Show the alert and exit early  
    self.window?.rootViewController?.presentViewController(alertController,  
                                                       animated: true,  
                                                       completion: nil)  
  
    //let vc = window?.rootViewController as? ViewController  
    //vc?.refreshTable()  
  
    // Alternative (more flexible way)  
unc showNotificationNowFromCloudKit(notification: UILocalNotification) {  
    // Create a notification  
    let notification = UILocalNotification()  
    notification.alertBody = "This was from a Now notification."  
    notification.alertAction = "Ok!"  
  
    // Schedule the notification  
    UIApplication.sharedApplication().presentLocalNotificationNow(notification)
```

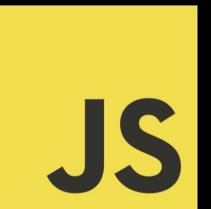
# WEB SERVICES

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" />
```



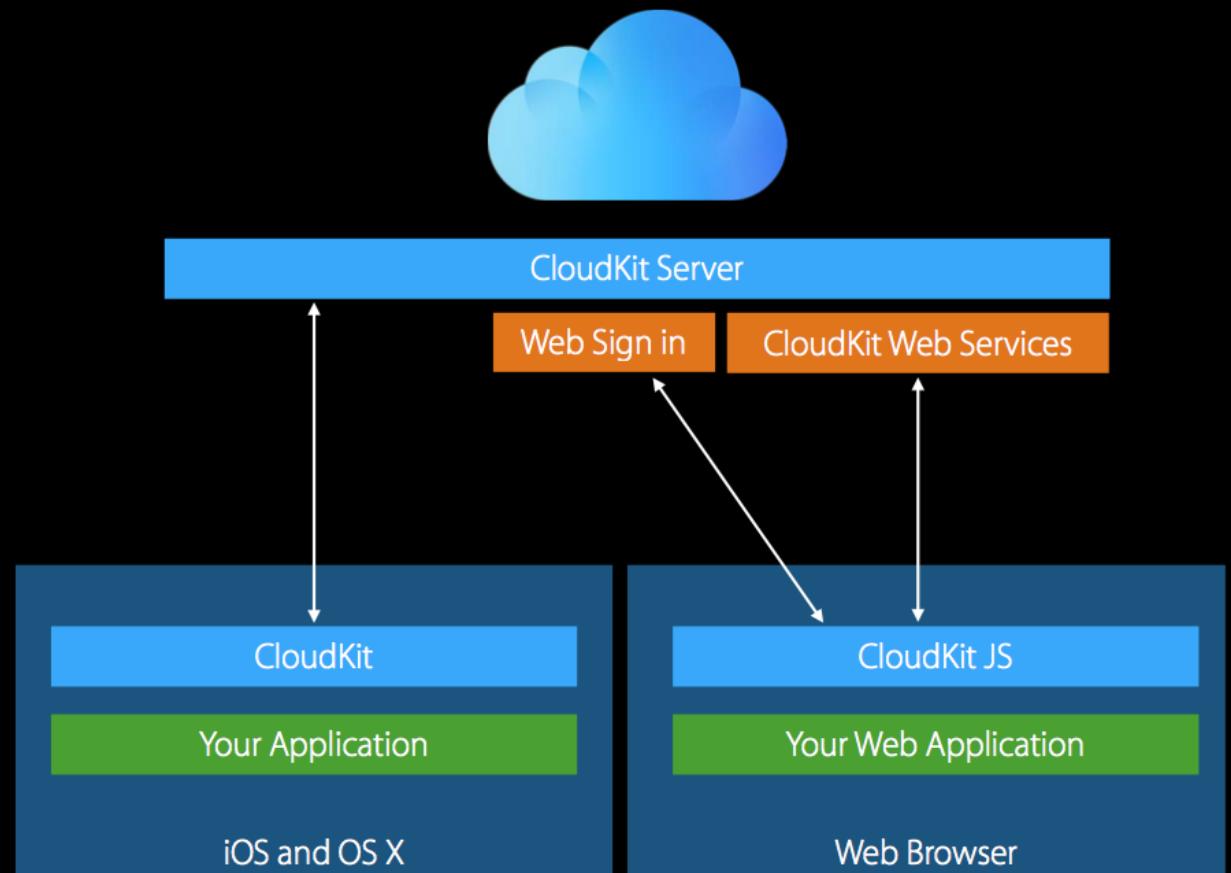
## WEB SERVICE API

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



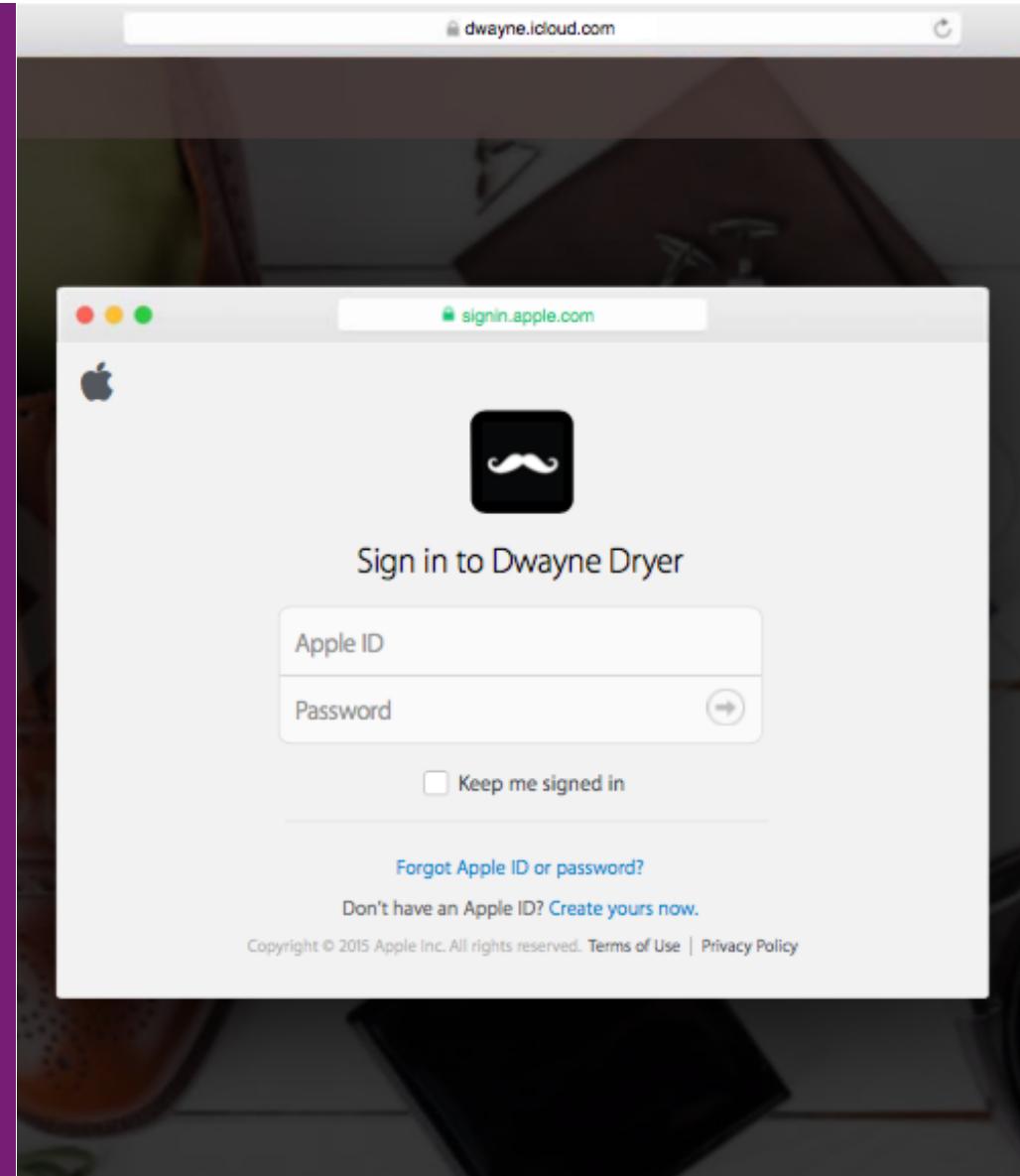
## WEB SERVICES

- 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

Run Code

Unauthenticated User

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>
```

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

# WHEN TO USE CLOUDKIT

## WHEN TO USE CLOUDKIT

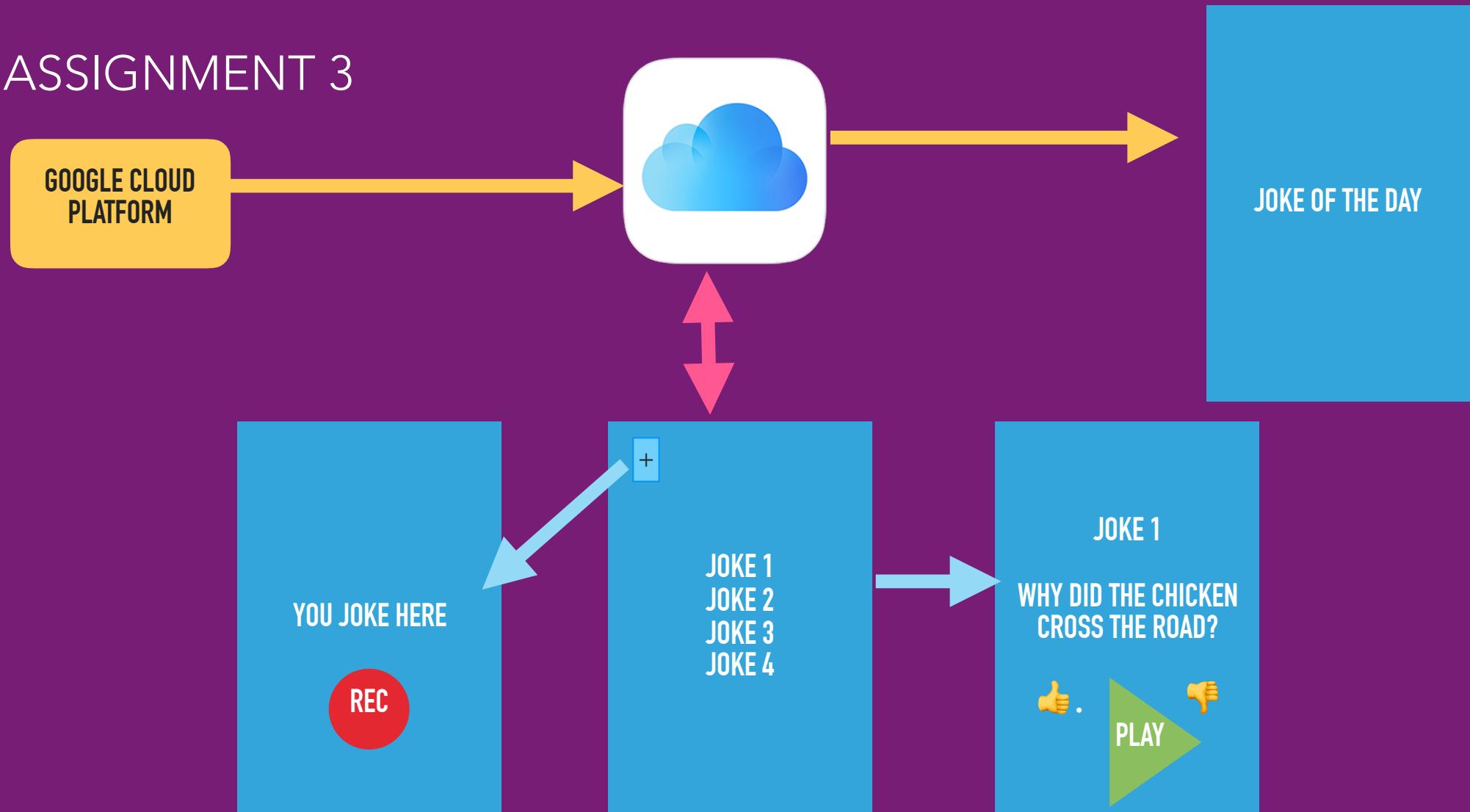
- You're developing exclusively for Apple platforms
- You use it as a partial solution

# ASSIGNMENT 4

# ASSIGNMENT 4

- 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 • SPRING 2017 • SESSION 6

---

# BACKENDS FOR MOBILE APPLICATIONS