THE UNIVERSITY OF CHICAGO

MPCS 51033 • AUTUMN 2017 • SESSION 7

BACKENDS FOR MOBILE APPLICATIONS

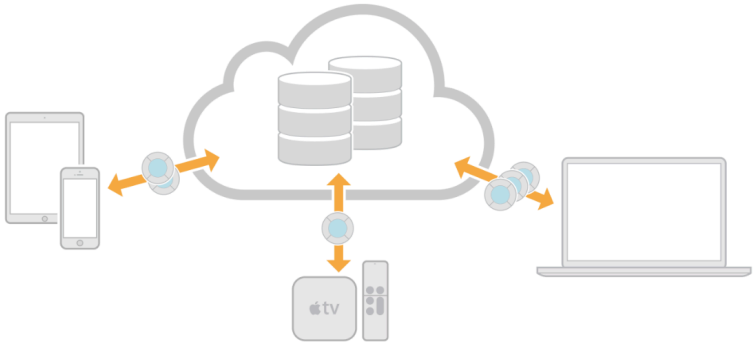
CLOUDKIT ☁️

CLOUDKIT

- CloudKit QuickStart

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

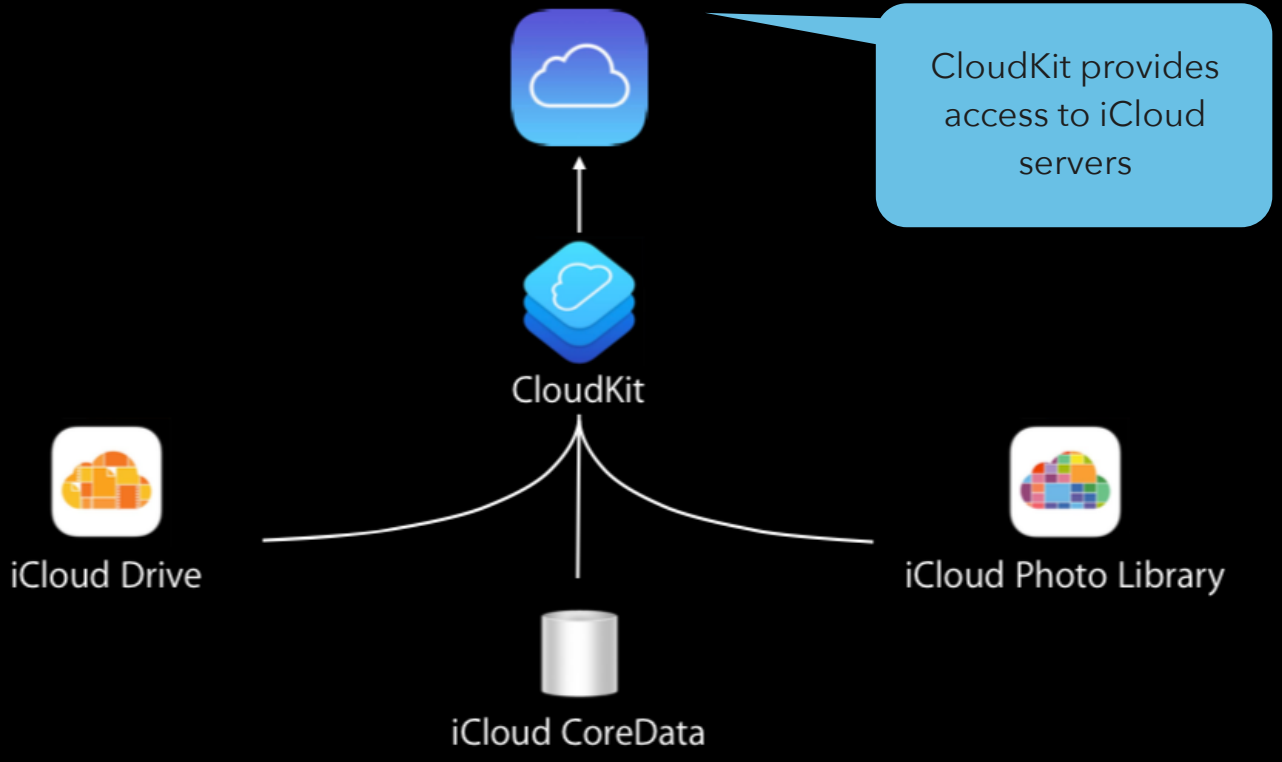
See Glossary for the definition of database terms used in this book.

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.

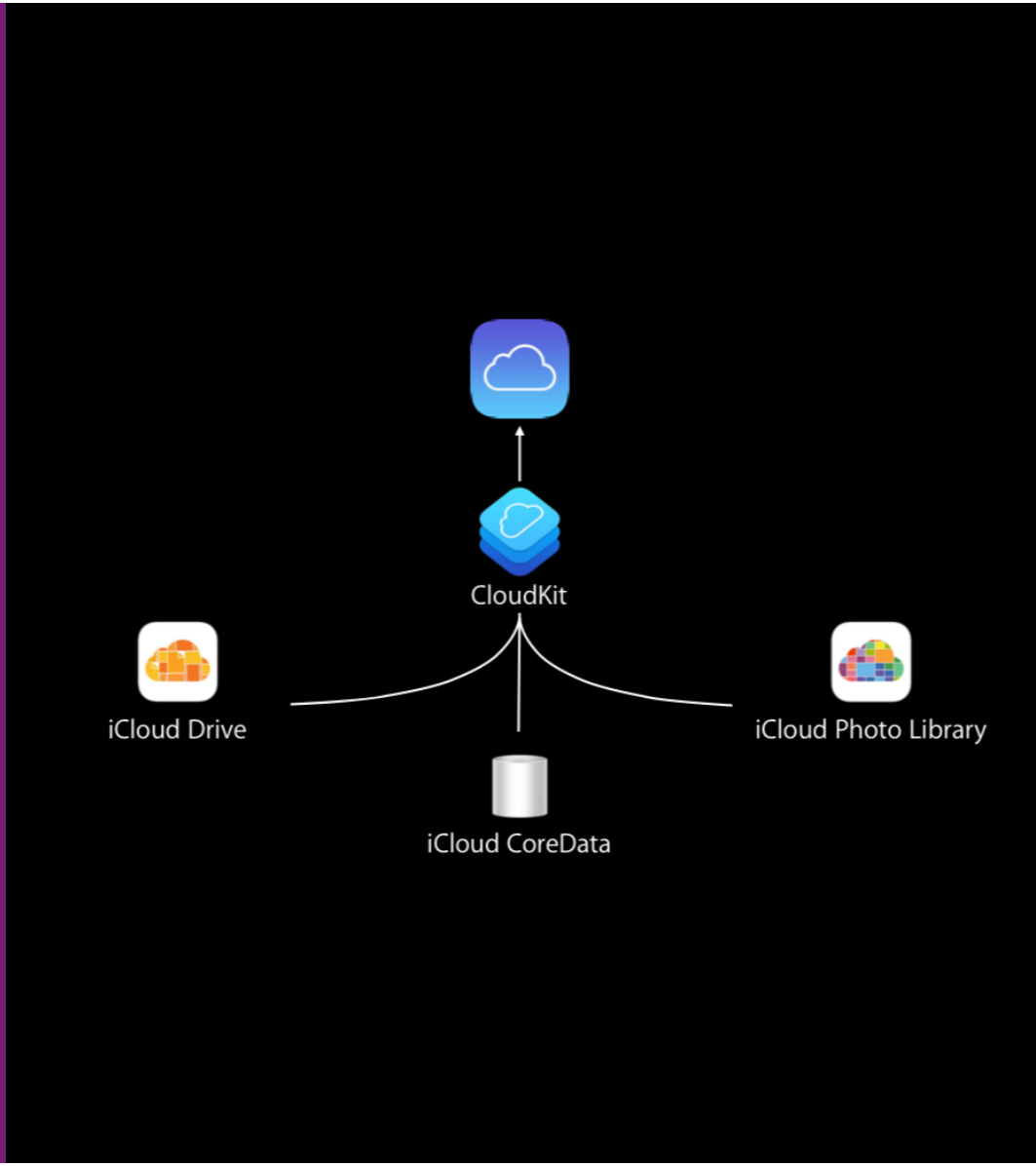
CLOUDKIT


CloudKit provides access to iCloud servers
CloudKit
iCloud Drive
iCloud Photo Library
iCloud CoreData

CLOUDKIT

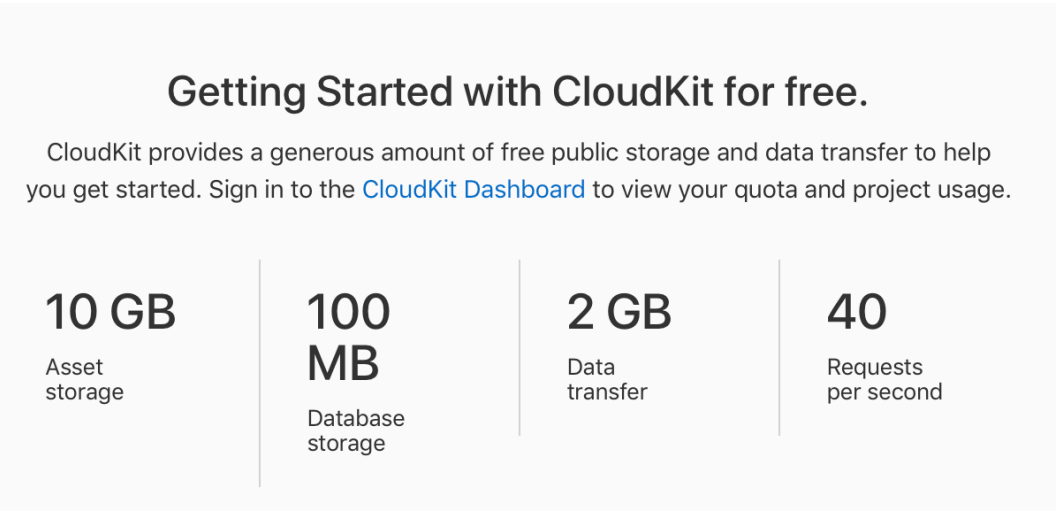
- Access to iCloud servers
- Supported on macOS, tvOS, iOS, watchOS and web (JS API)

#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



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

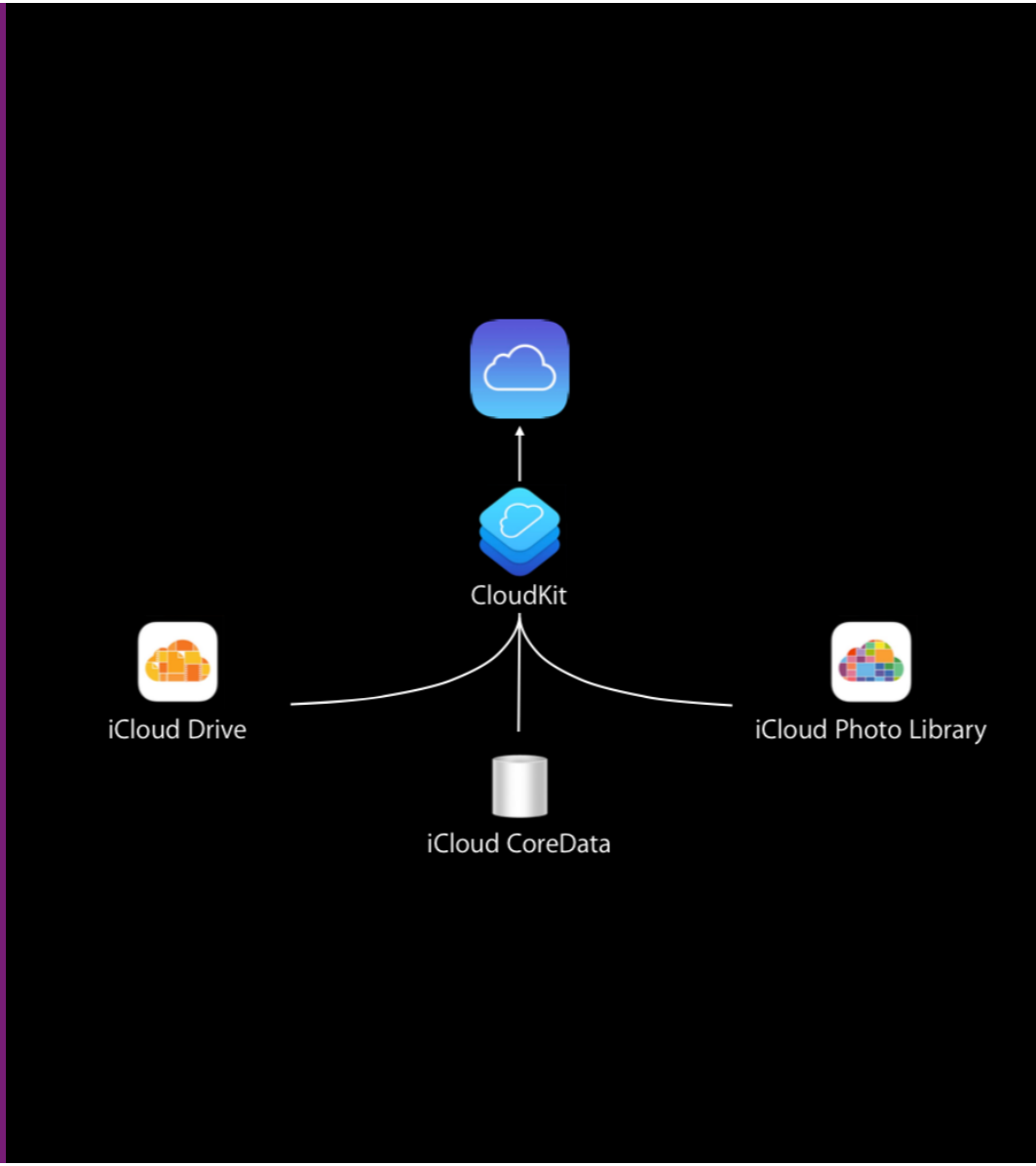
1 PB	10 TB	200 TB	400
Asset storage	Database storage	Data transfer	Requests per sec.
Based on: 100 MB per user	Based on: 1 MB per user	Based on: 20 MB per user	Based on: 4 per 100,000 users

$0

Total Cost

CLOUDKIT

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

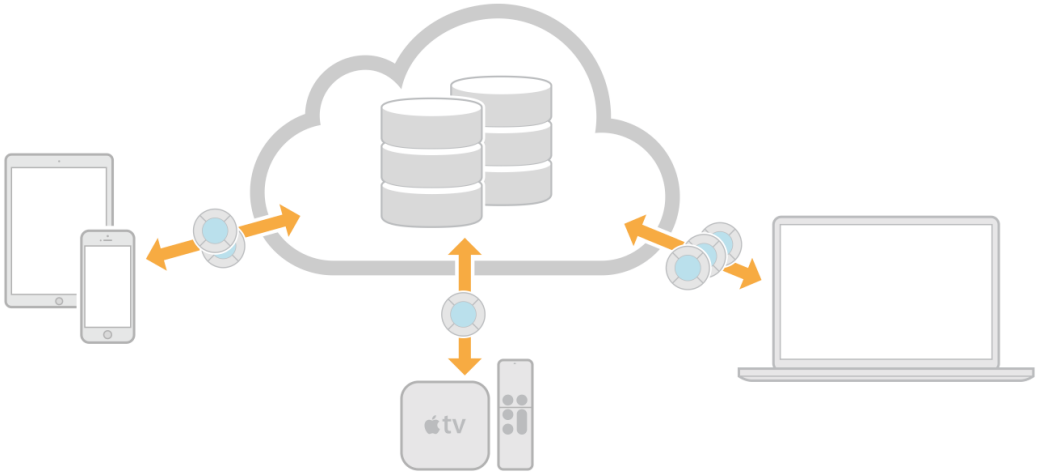


CLOUDKIT

- Databases
  - Public
  - Private
  - Shared

CLOUDKIT

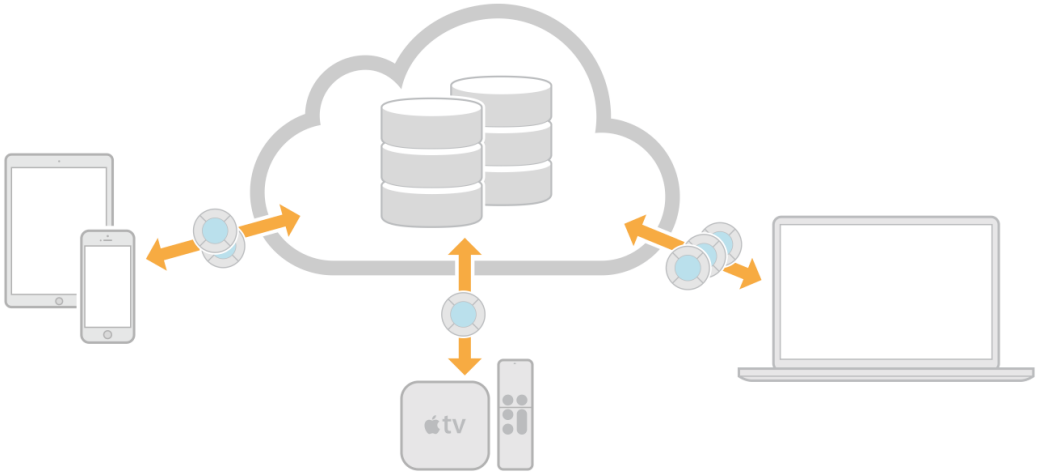
- Structured and bulk data
  - Data with types
  - Blobs



CLOUDKIT

CORE OBJECT

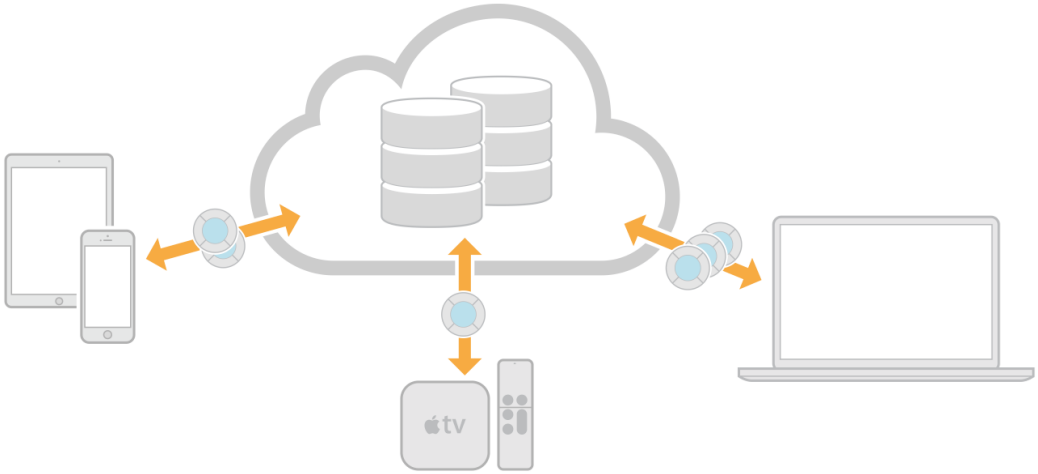
- Containers
- Databases
- Records
- Record Zones
- Record Identifiers
- Shares
- References
- Assets



CLOUDKIT

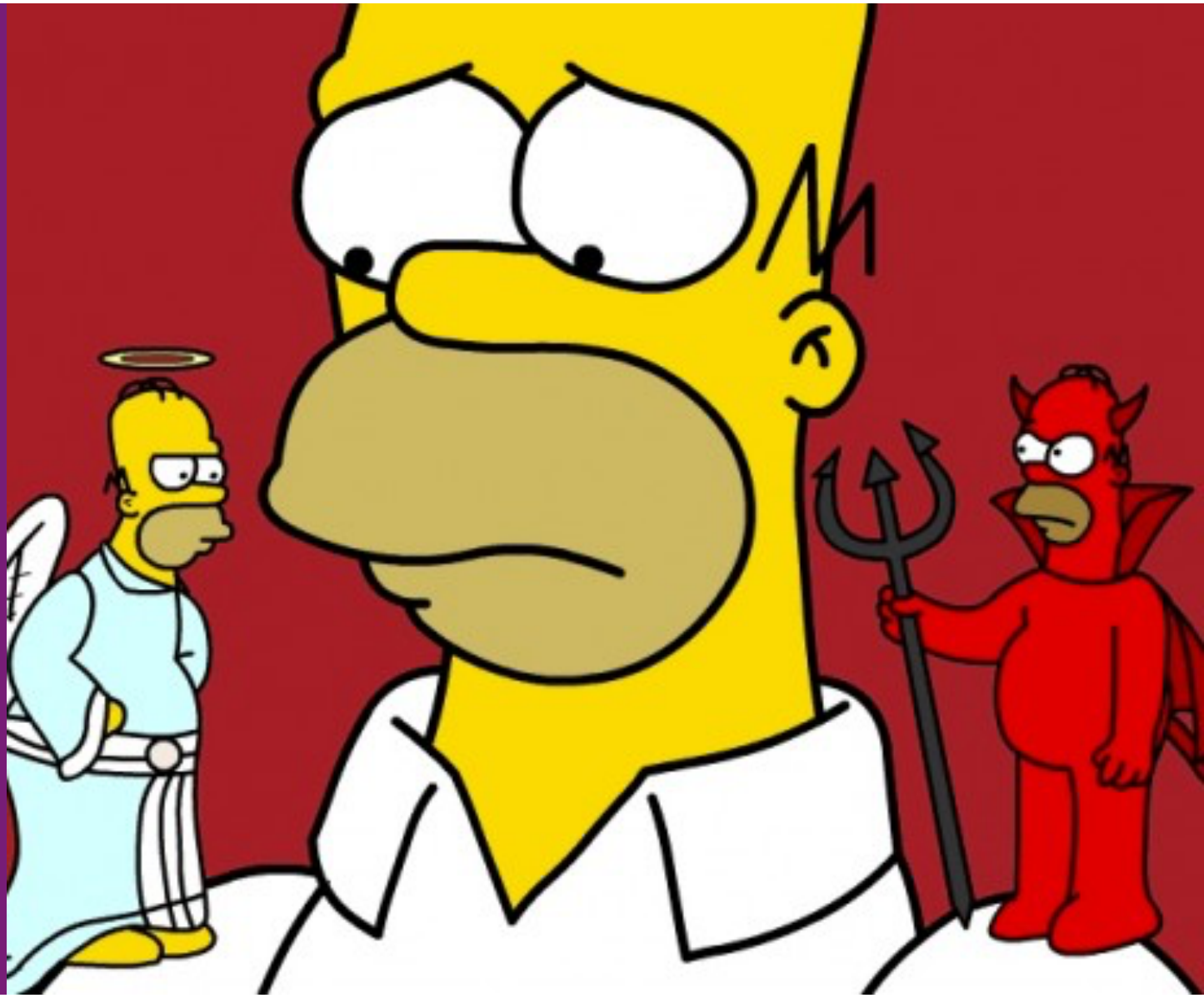
- Transport layer, not local persistence
- You still need to figure out what to do with the data once you have it on your device
  - Cache
  - Mirror

#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)

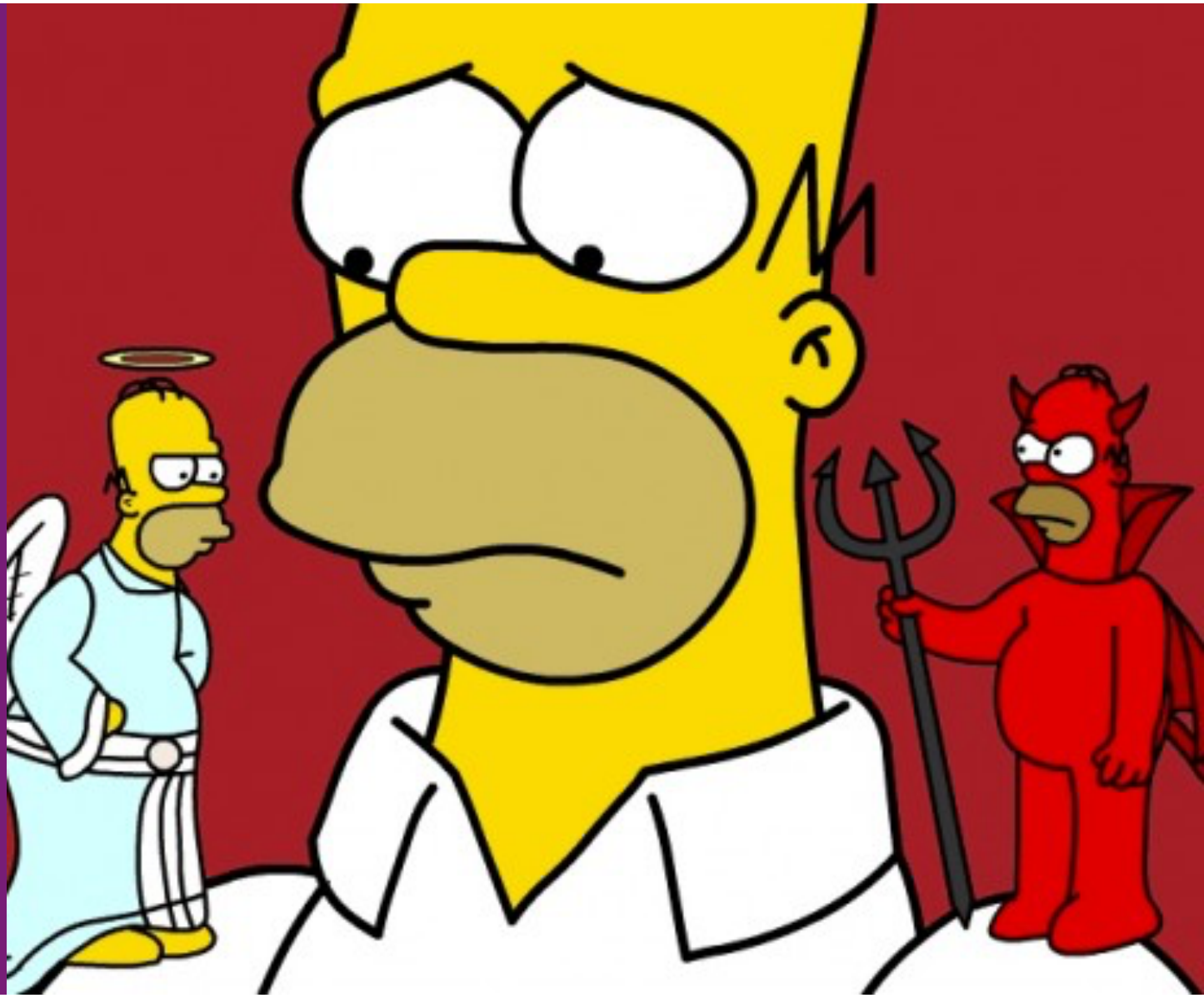


CLOUDKIT

- 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



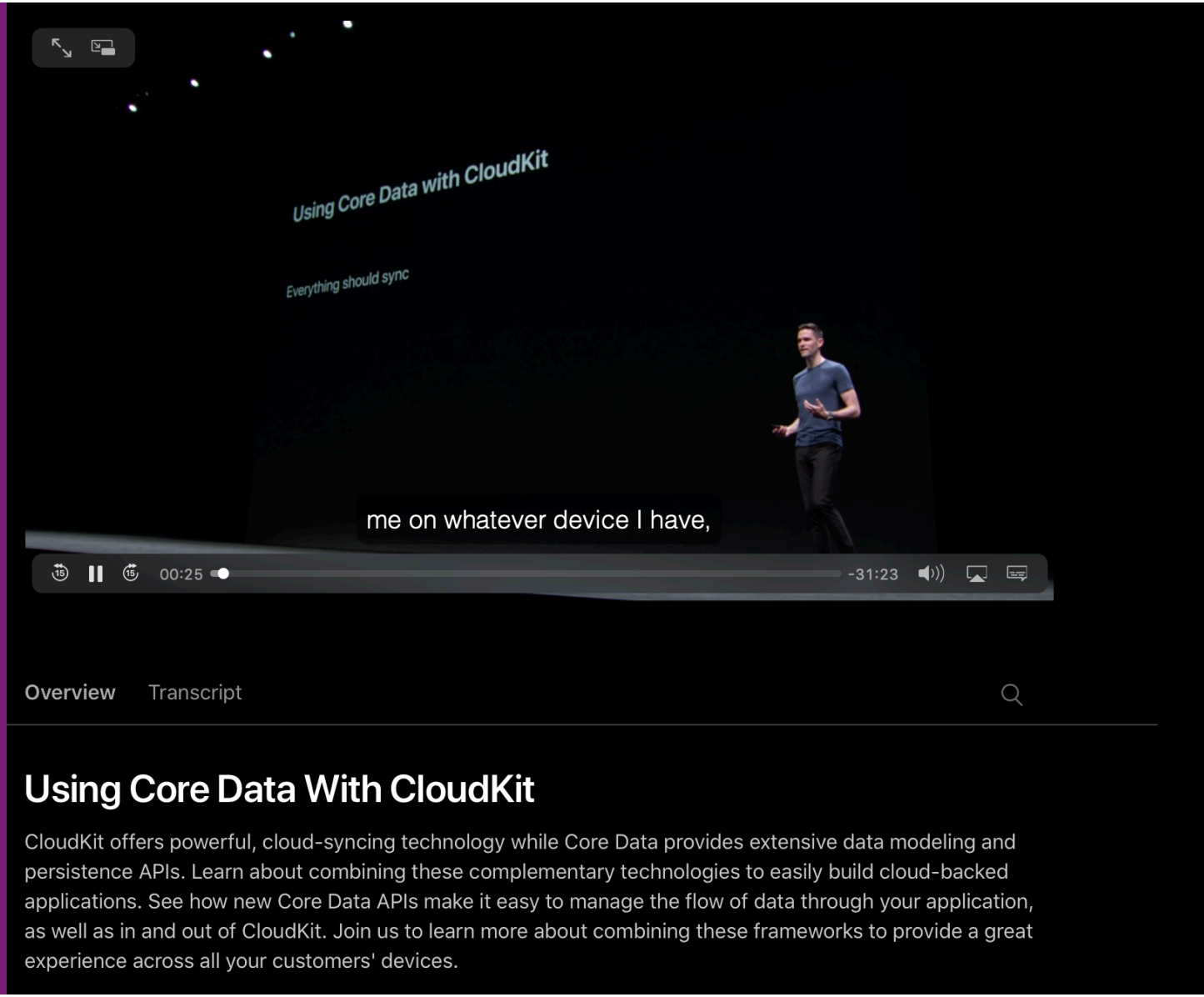
CLOUDKIT

- 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



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.

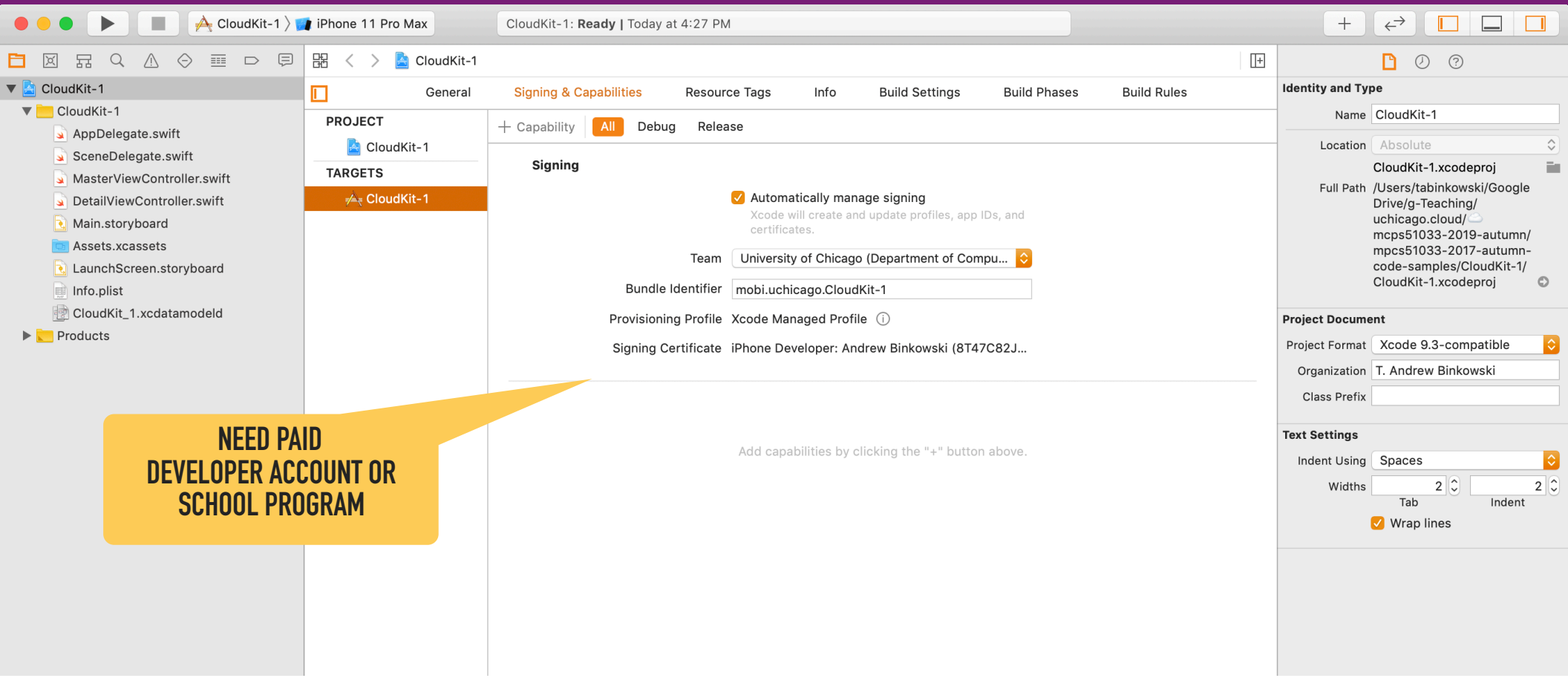
1. Choose File > New > Project to create a new project.
2. Select a project template to use as the starting point for your project, and click Next.
3. Select the Use Core Data and Use CloudKit checkboxes.
4. Enter any other project details and click Next.
5. Specify a location for your project and click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

THIS METHOD IS MORE CORE DATA THAN

Frame
Core D
On Th
Overvi
See Al

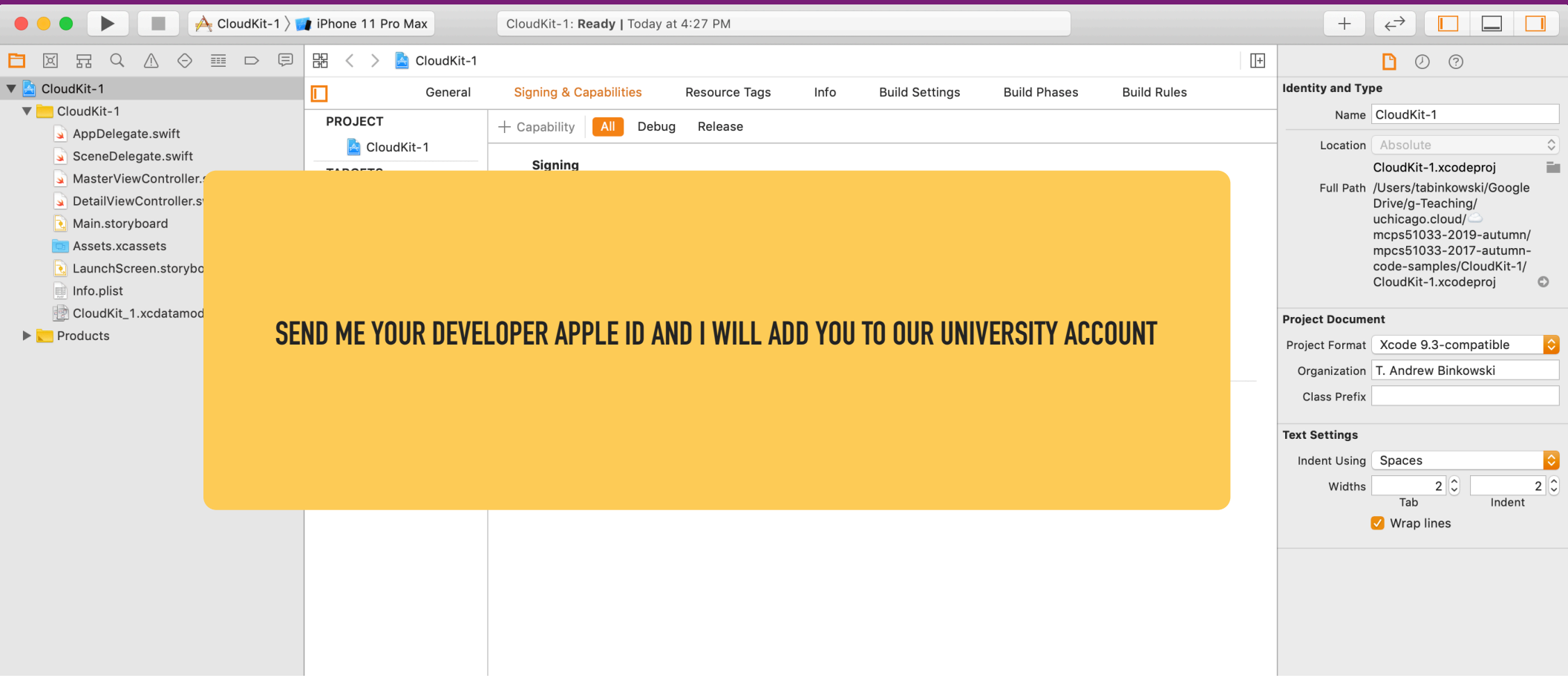
ENABLING CLOUDKIT IN YOUR APPLICATION

ENABLING CLOUDKIT IN YOUR APPLICATION



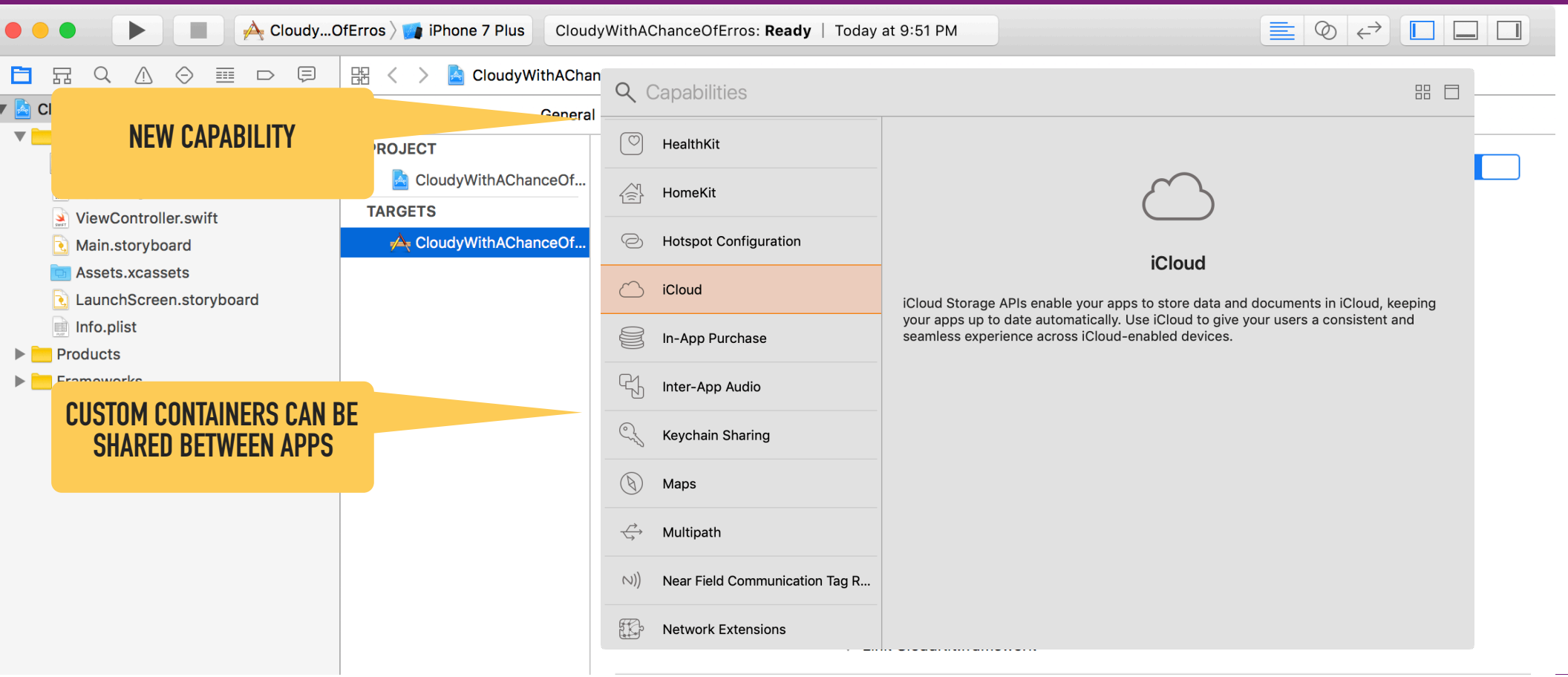
NEED PAID DEVELOPER ACCOUNT OR SCHOOL PROGRAM

ENABLING CLOUDKIT IN YOUR APPLICATION

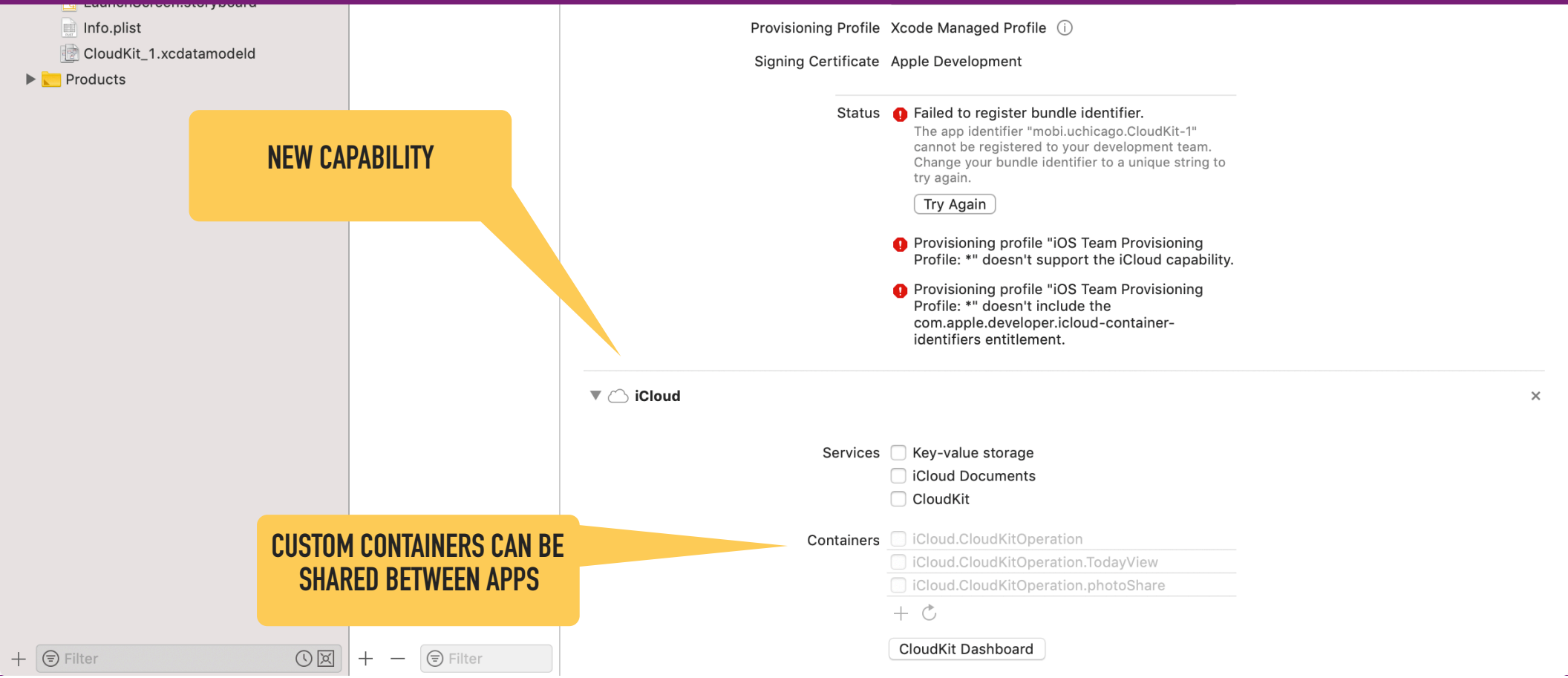


SEND ME YOUR DEVELOPER APPLE ID AND I WILL ADD YOU TO OUR UNIVERSITY ACCOUNT

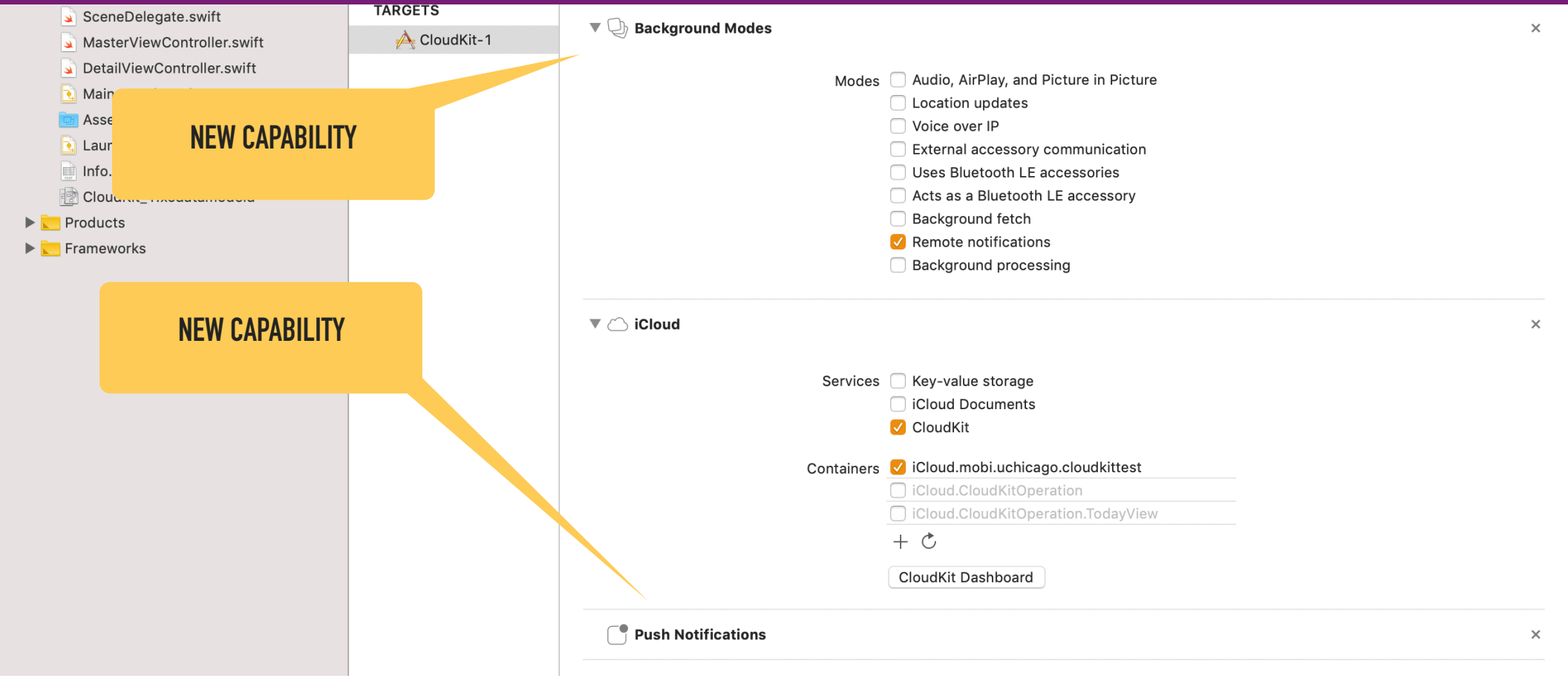
ENABLING CLOUDKIT IN YOUR APPLICATION



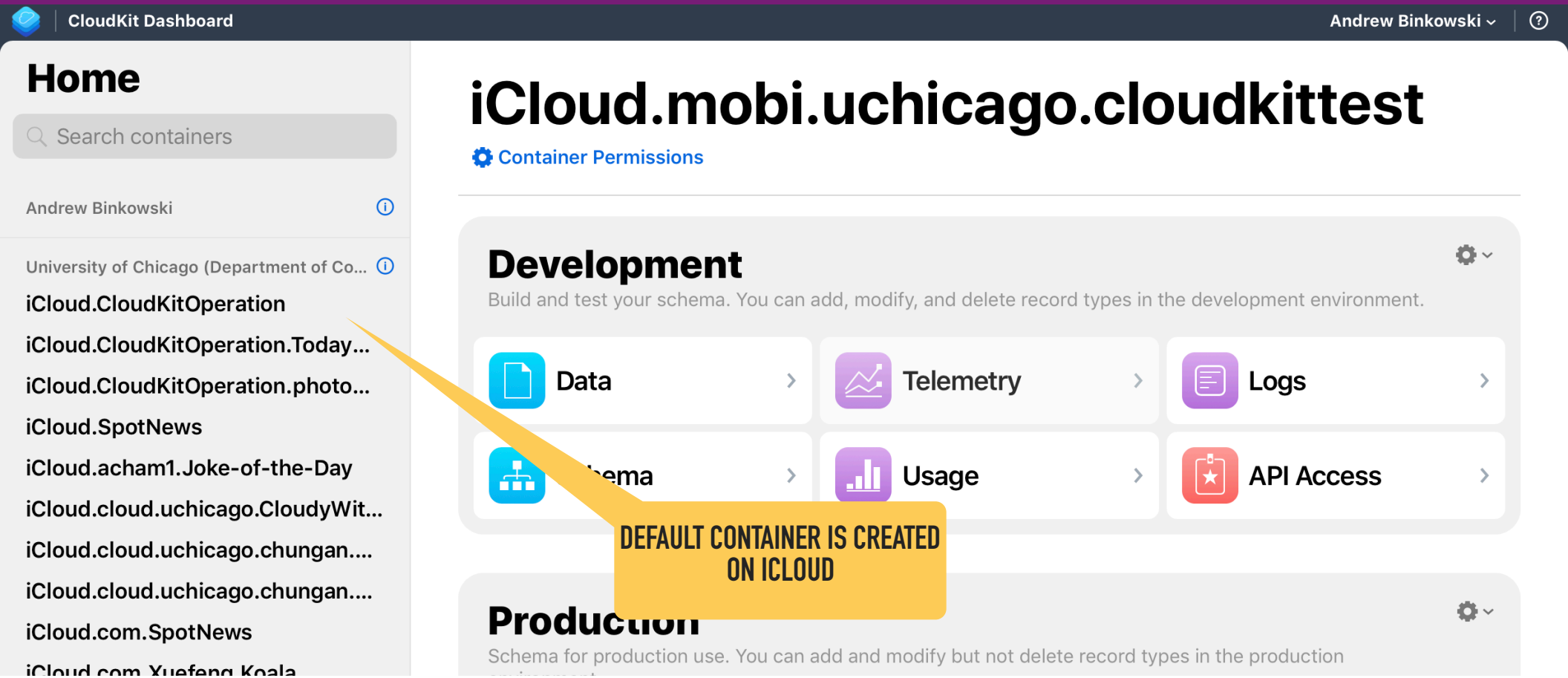
ENABLING CLOUDKIT IN YOUR APPLICATION



ENABLING CLOUDKIT IN YOUR APPLICATION

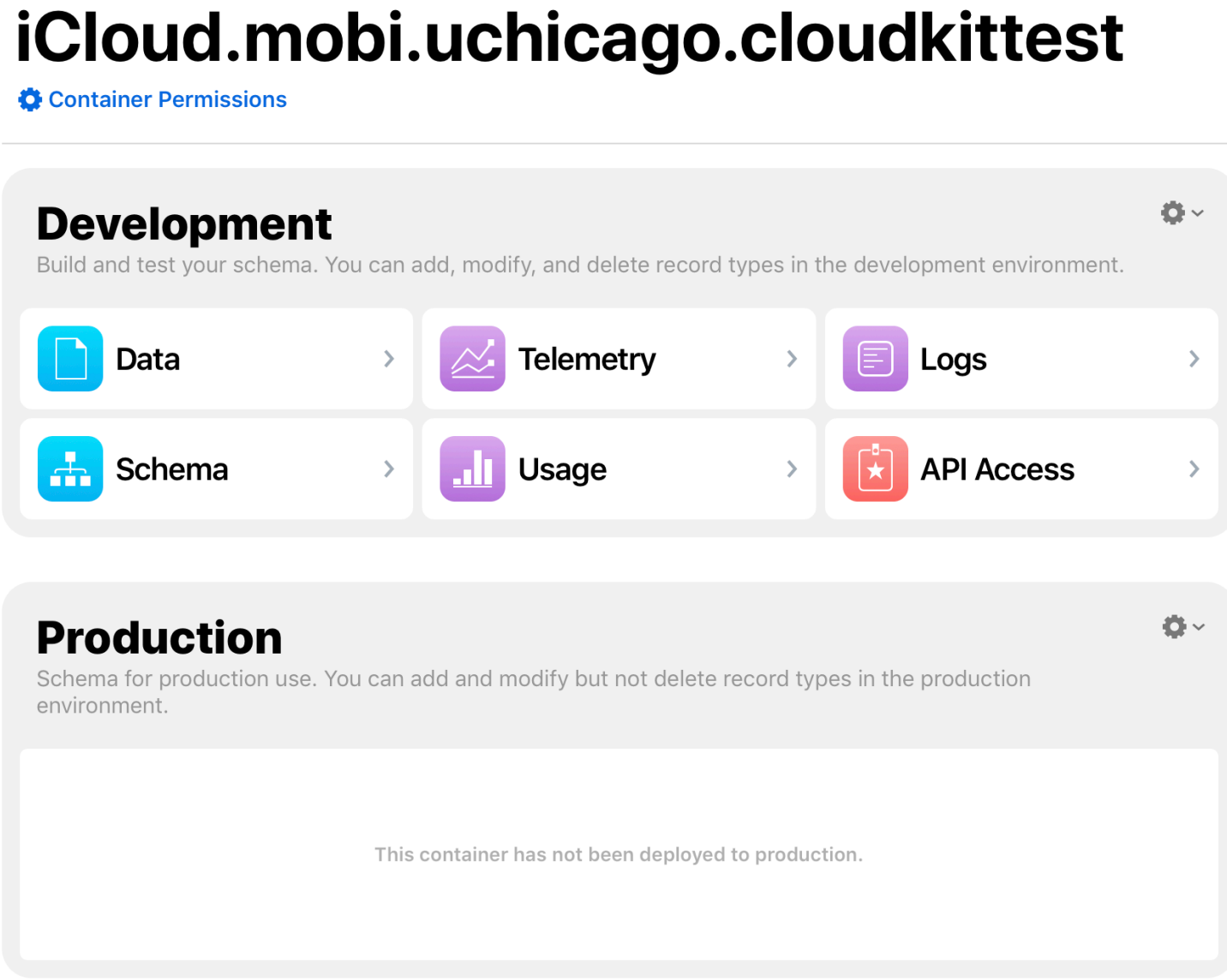


ENABLING CLOUDKIT IN YOUR APPLICATION



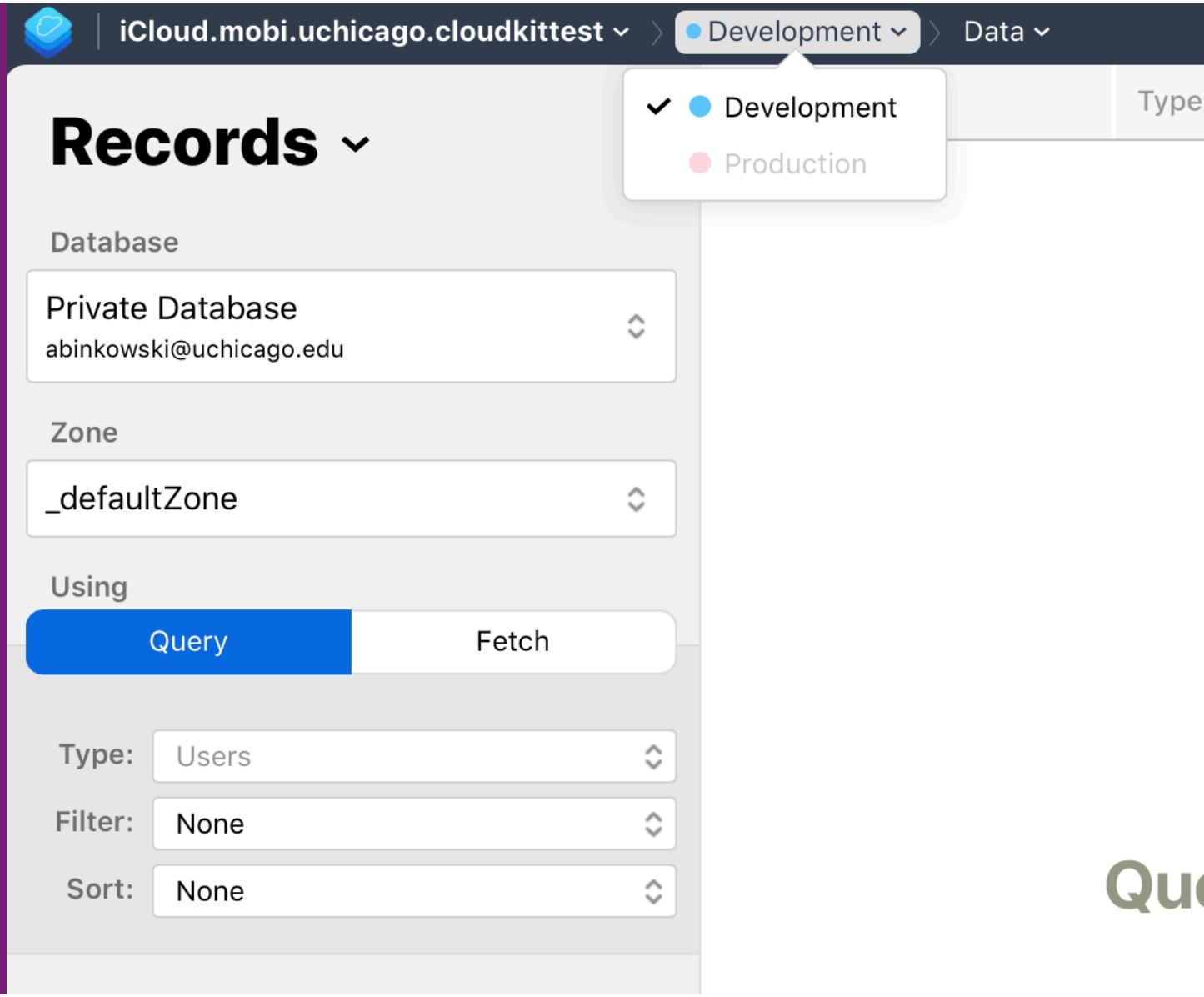
ENABLING CLOUDKIT IN YOUR APPLICATION

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

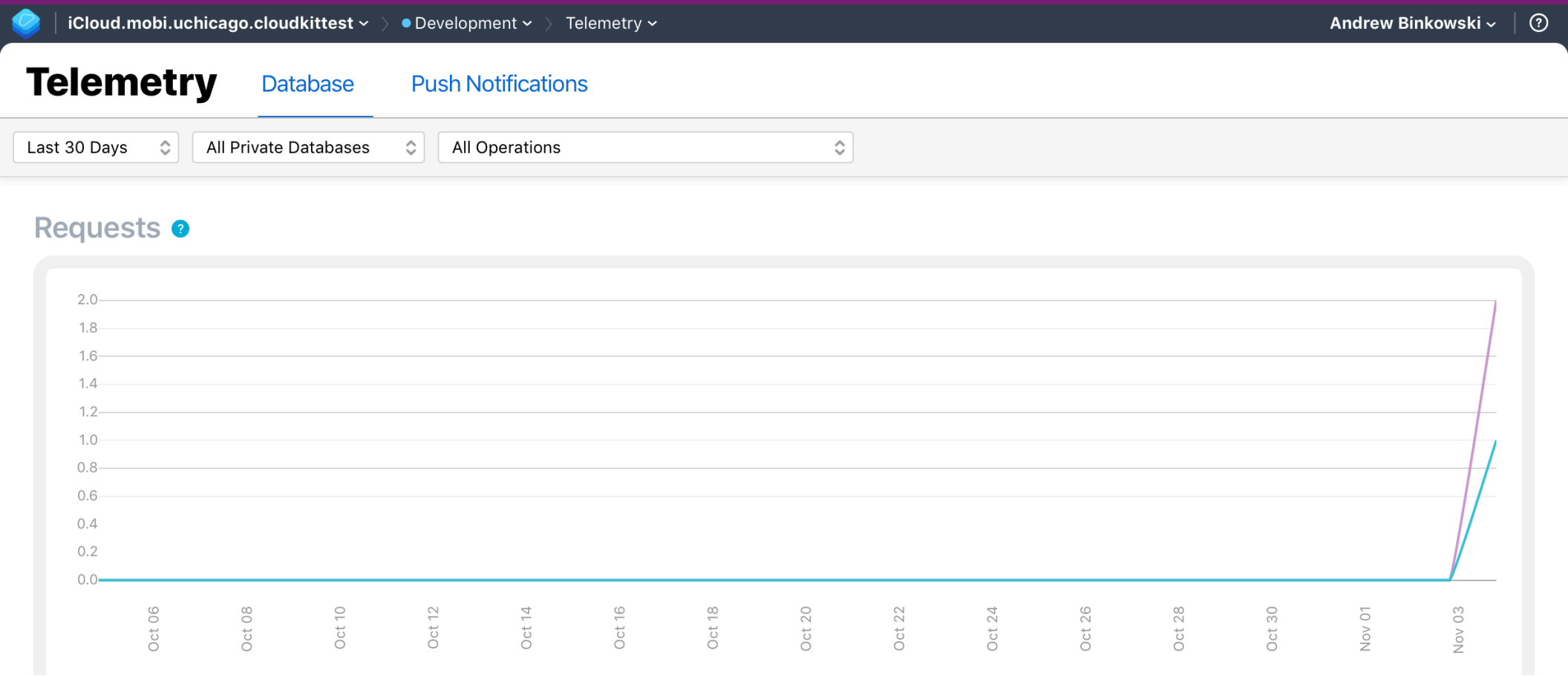


ENABLING CLOUDKIT IN YOUR APPLICATION

- "Flip a switch" to go to production mode



ENABLING CLOUDKIT IN YOUR APPLICATION

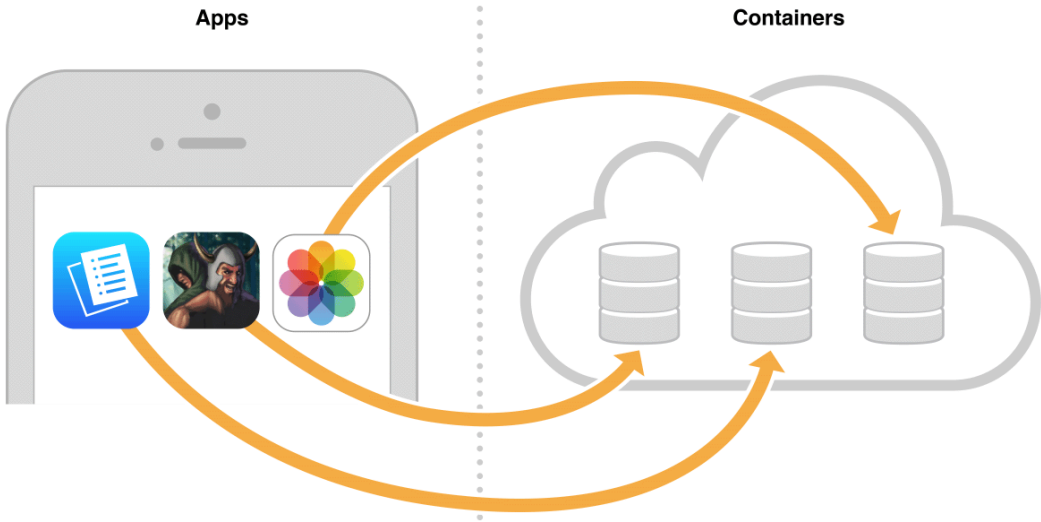


CONTAINER

CLOUDKIT OBJECTS

CONTAINER

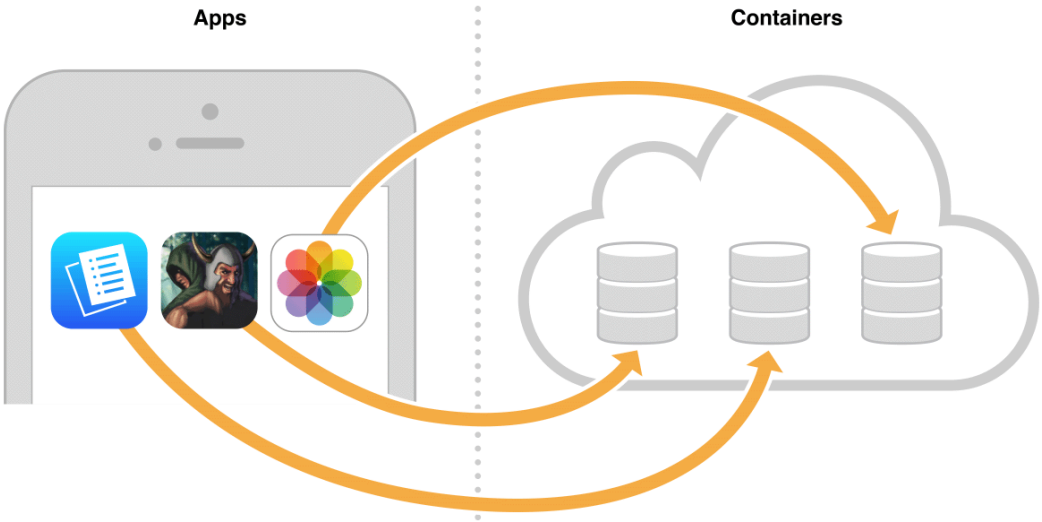
- CKContainer
  - One default container per application
  - Data segregation
  - User encapsulation
  - Managed by the developer via portal



CLOUDKIT OBJECTS

CONTAINER

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



CLOUDKIT

CONTAINER

```
import CloudKit

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

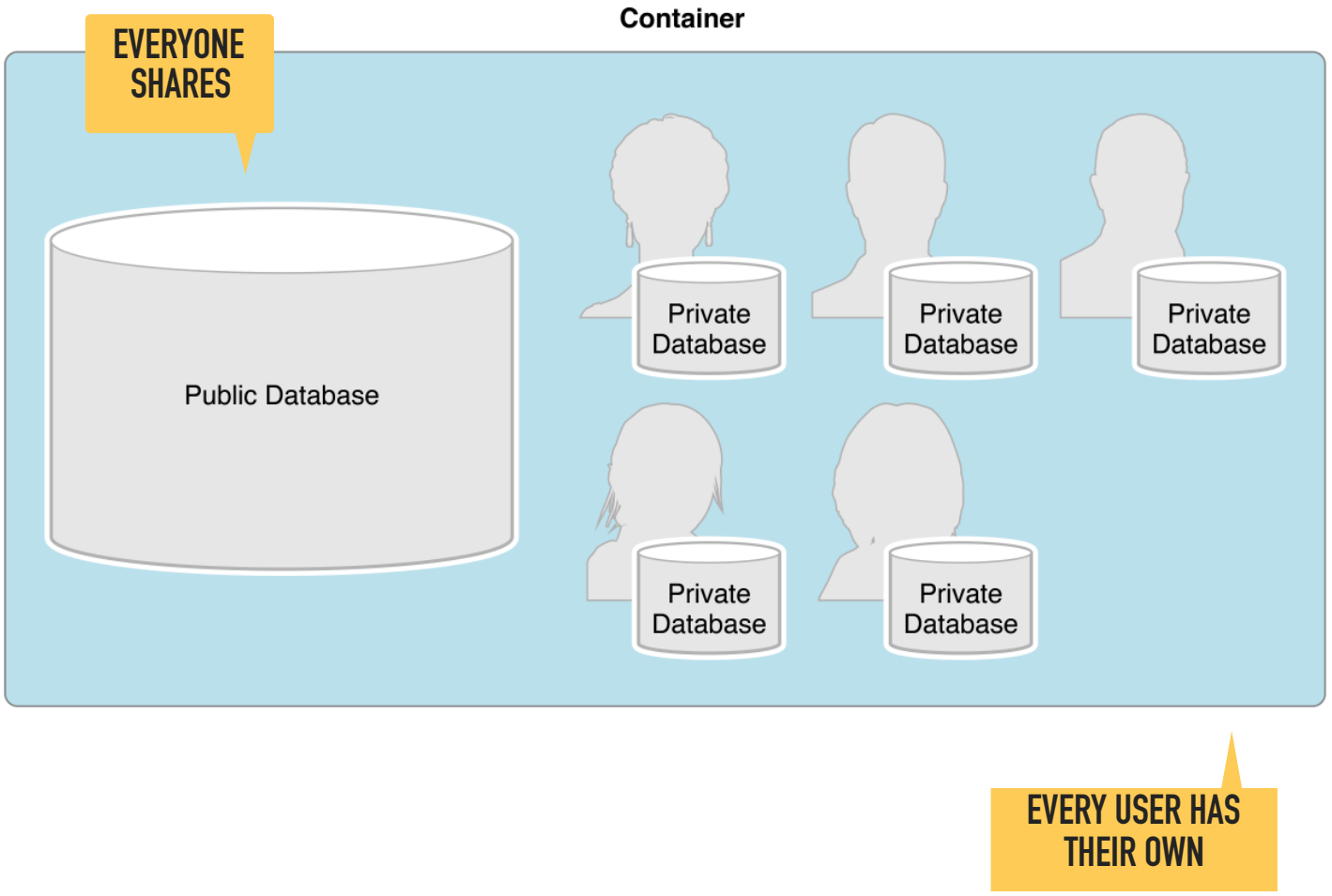
IT'S JUST THAT EASY

DATABASE

CLOUDKIT OBJECTS

DATABASE

- CKDatabase
  - Public
  - Private
  - Shared



CLOUDKIT OBJECTS

DATABASE

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

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

CLOUDKIT OBJECTS

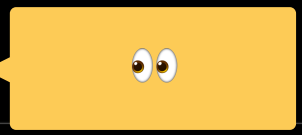
DATABASE

	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

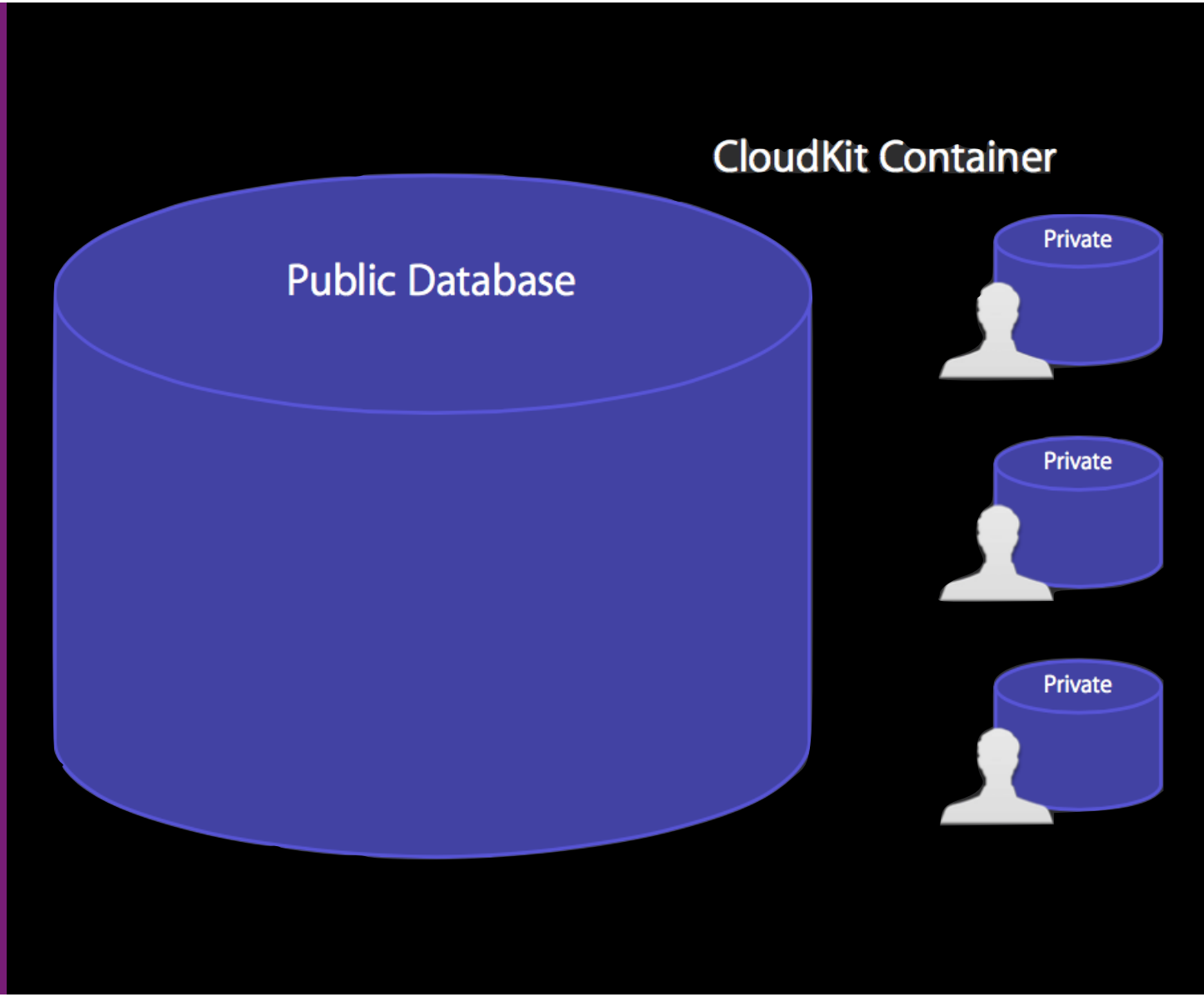
	Public Database	Private Database
Data Type	Shared Data	Current User's Data
Account	Required for Writing	Required
Quota	Developer	User
Default Permissions	World Readable	User Readable
Editing Permissions	iCloud Dashboard Roles	N/A



CLOUDKIT OBJECTS

DATABASE

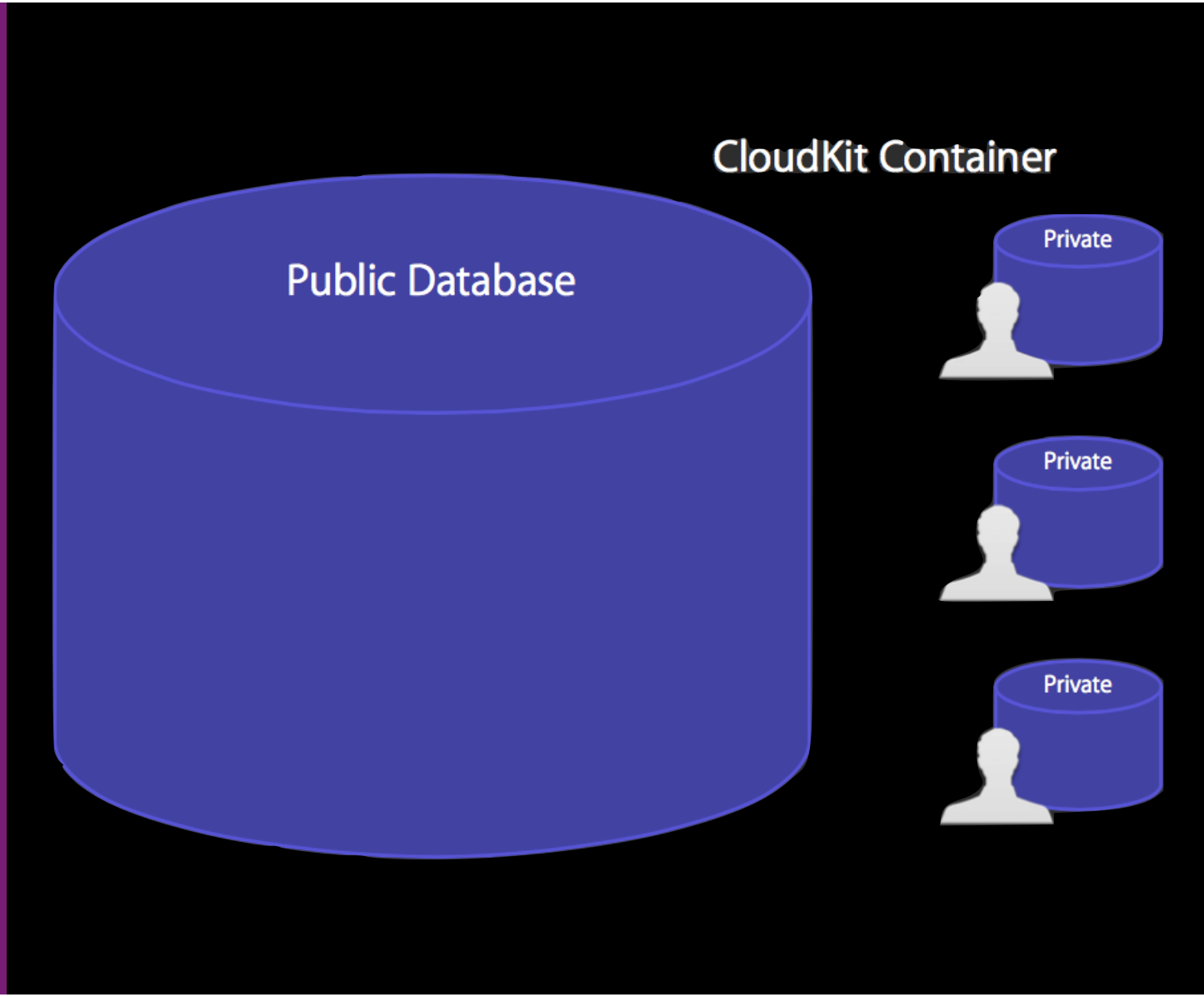
- Public databases
  - No atomic writes
  - No delta downloads
  - Cross zone references



CLOUDKIT OBJECTS

DATABASE

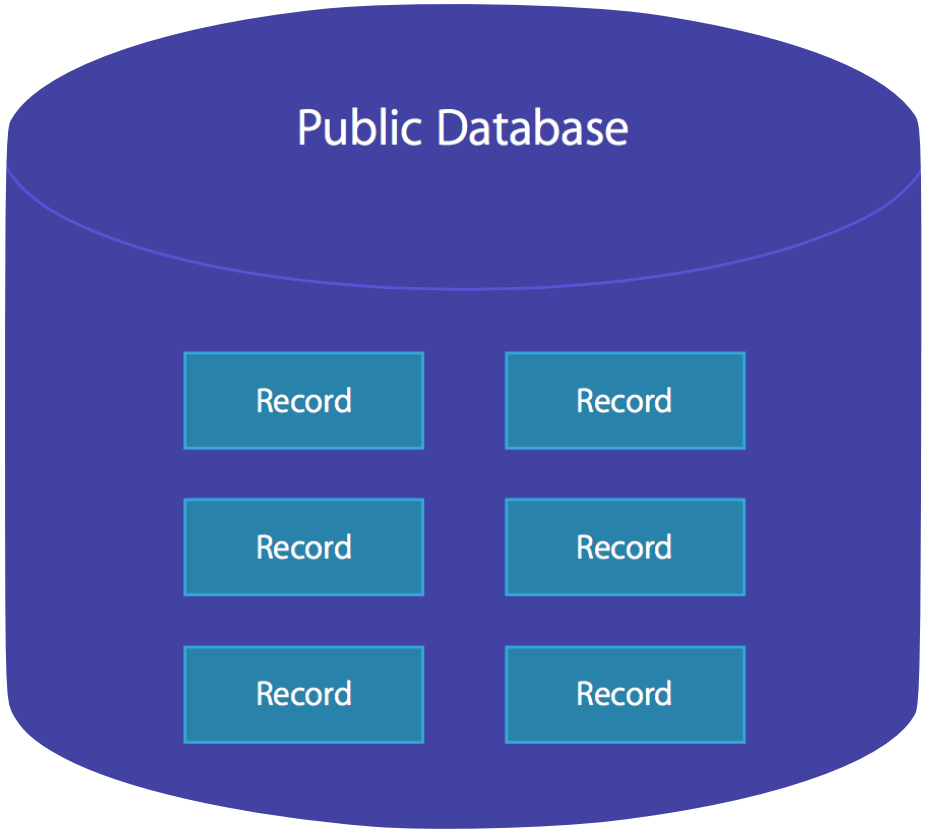
- Private databases
  - Atomic writes
  - Delta downloads
  - No cross-zone references



CLOUDKIT OBJECTS

DATABASE

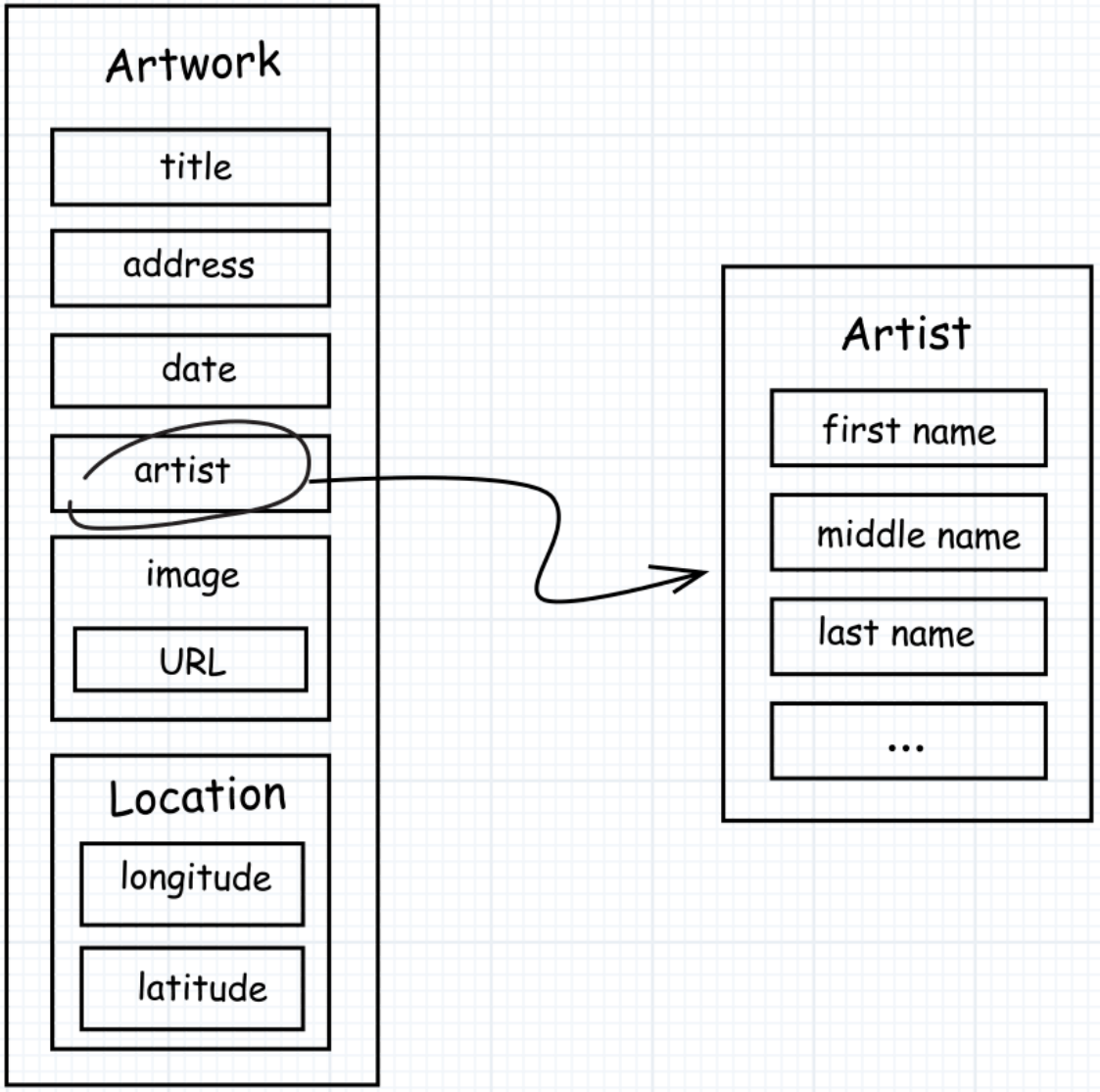
- Structured Data
  - Wraps key/value pairs
  - Types and references
  - Just-in-time schema
  - Metadata
    - Created, edited, etc.



CLOUDKIT OBJECTS

DATABASE

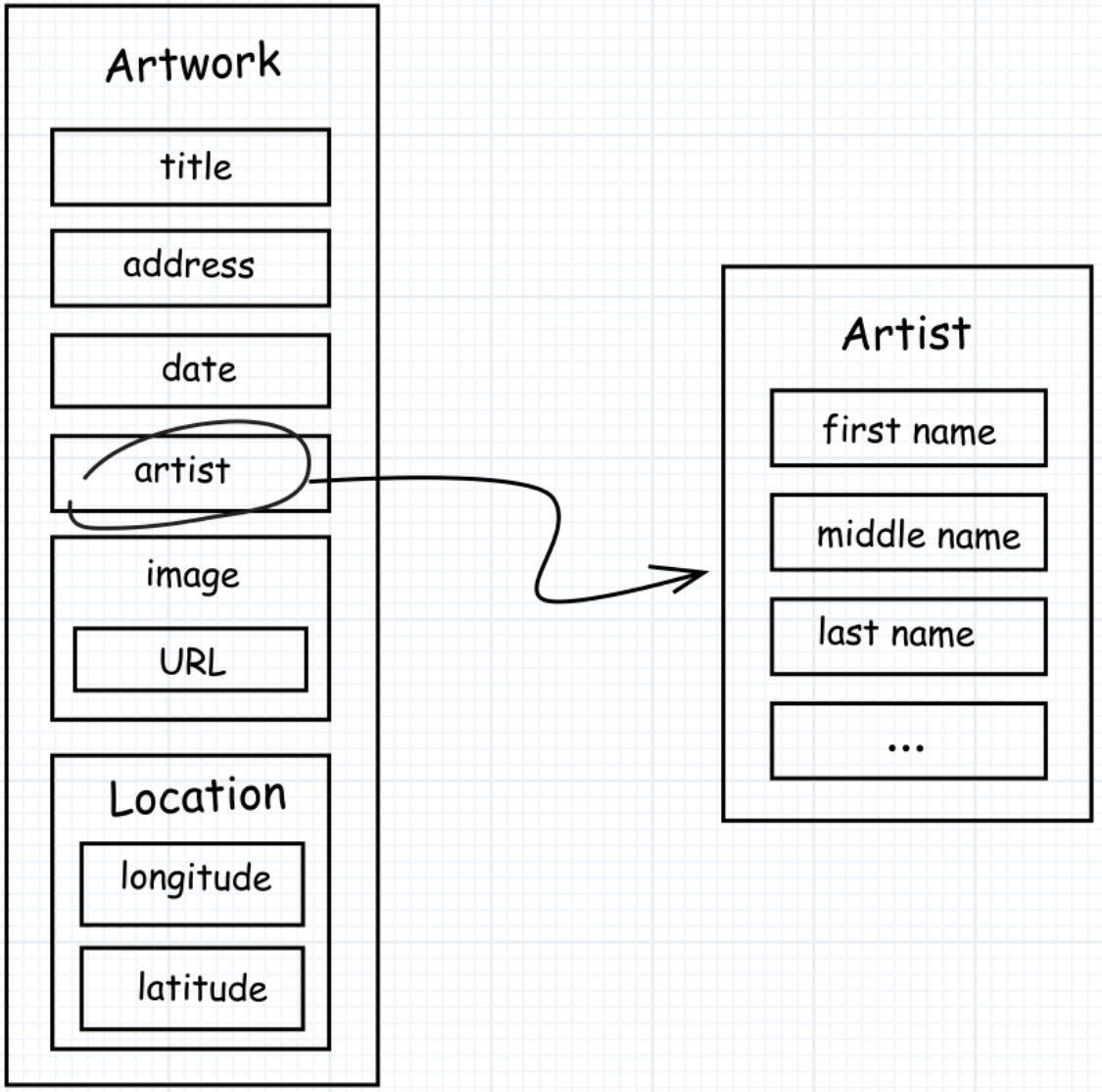
- Schema is built on-demand
- You can edit it in the console



CLOUDKIT OBJECTS

DATABASE

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

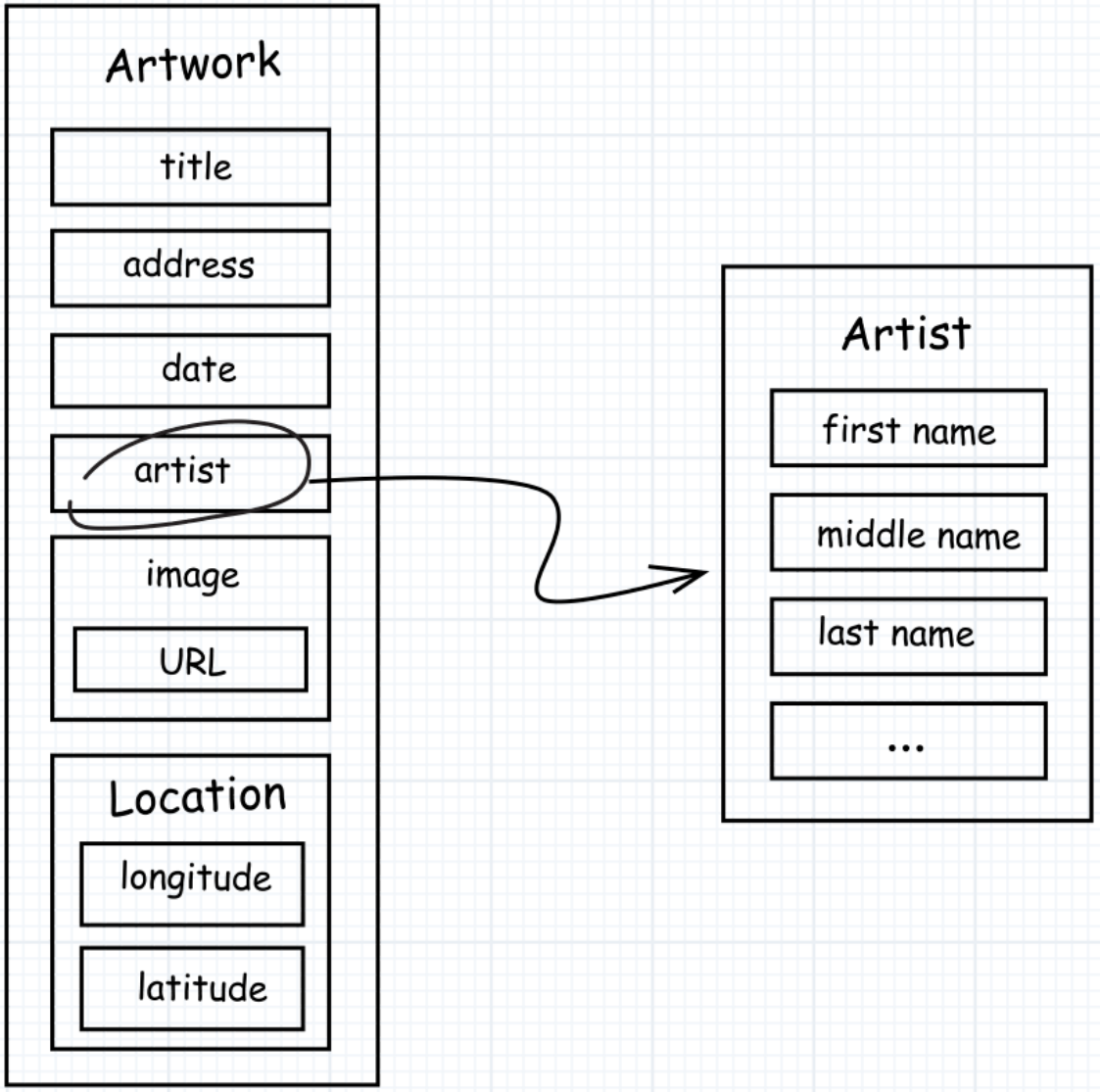


RECORDS

CLOUDKIT OBJECTS

RECORDS

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

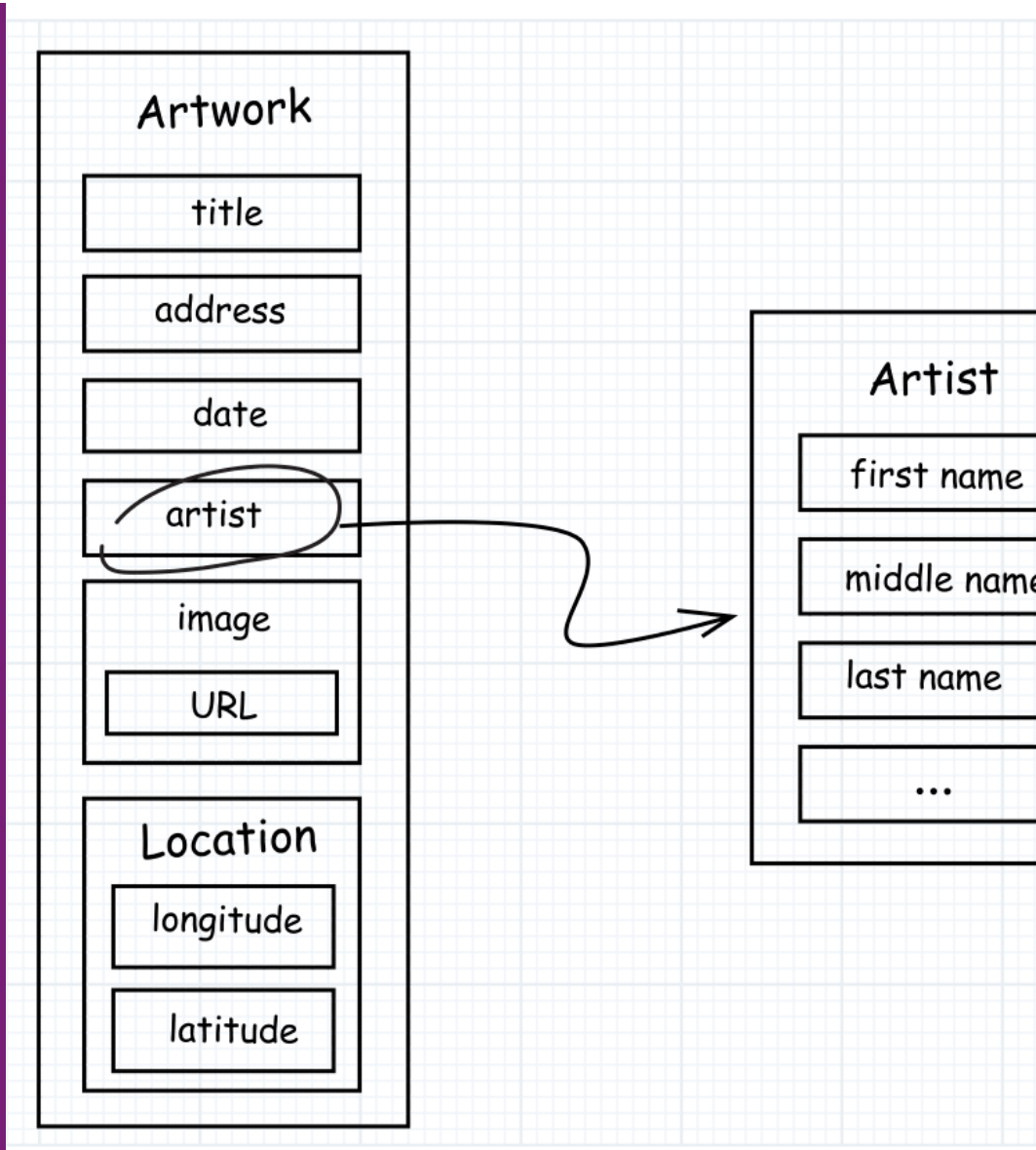


CLOUDKIT OBJECTS

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

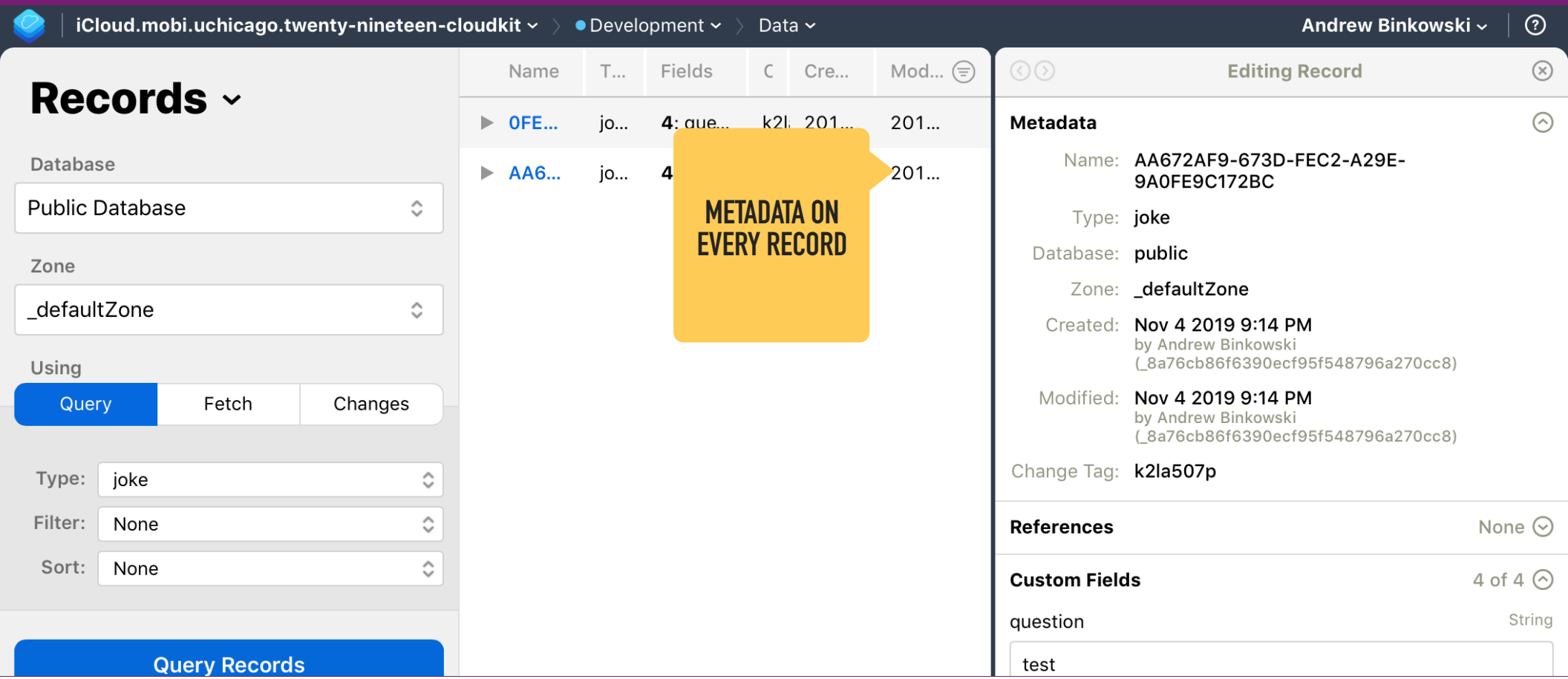
RECORDS

- 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] = valu
        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?
```

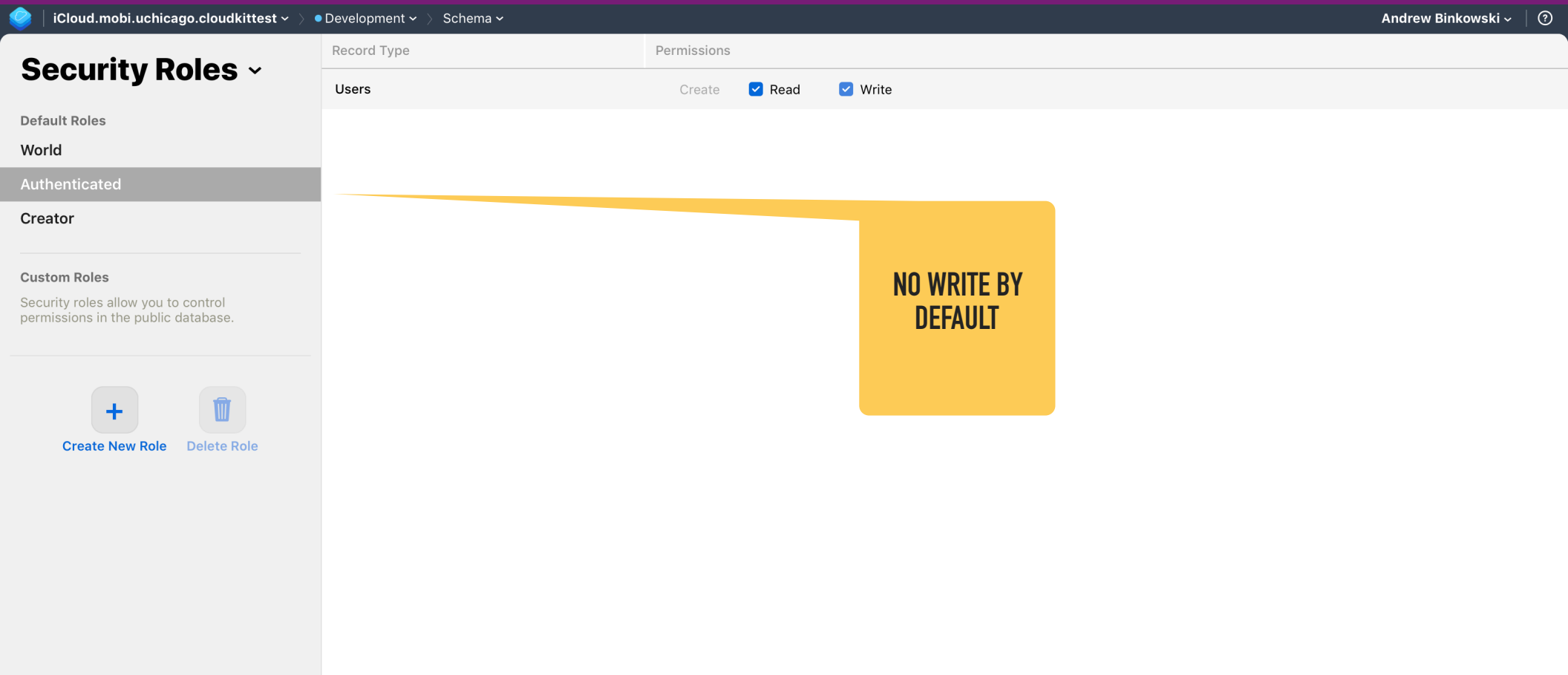
CLOUDKIT OBJECTS

RECORDS



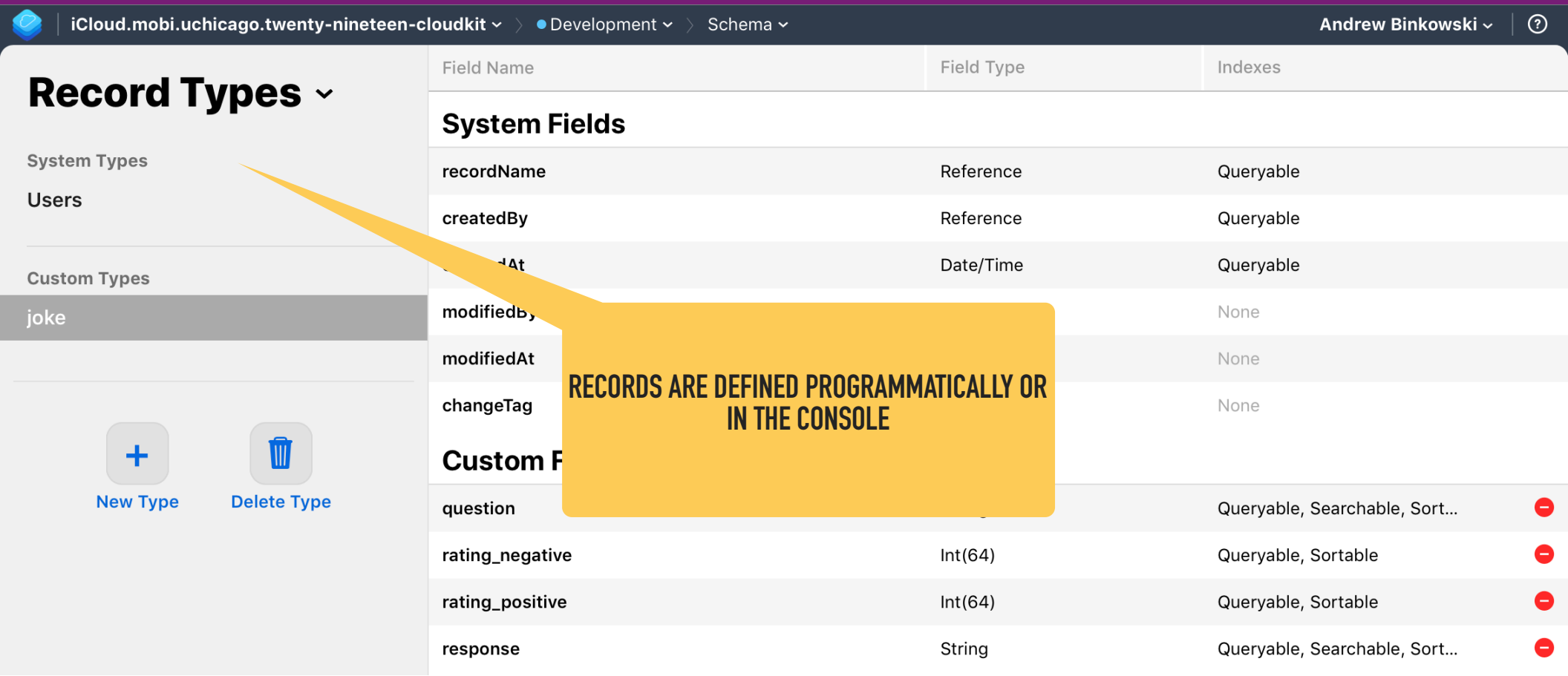
CLOUDKIT OBJECTS

RECORDS



CLOUDKIT OBJECTS

RECORDS



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().privateCloudDatabas
```

```
/// 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)")
  }

}

}
```

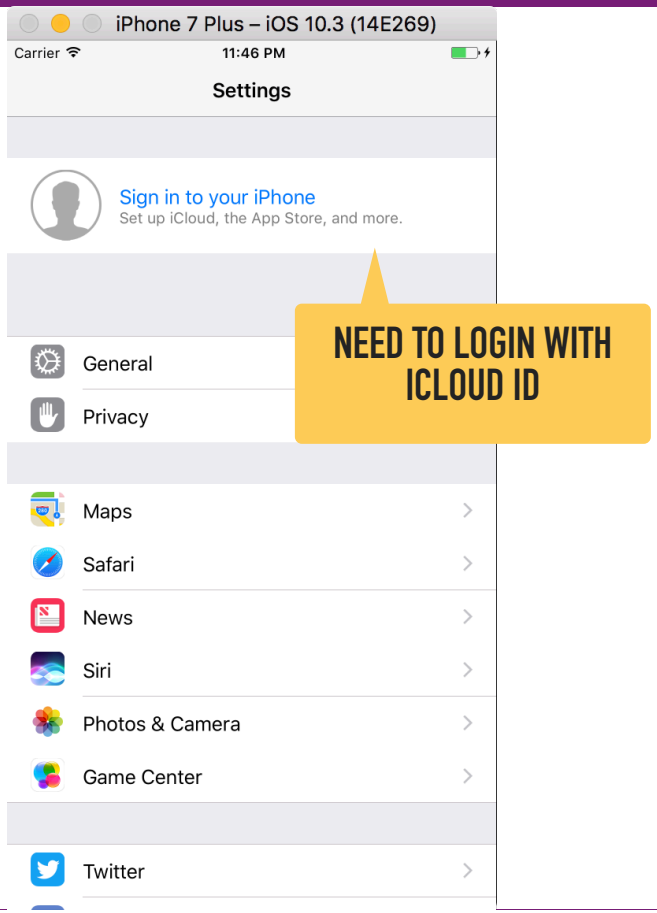
YOUR FIRST CLOUDKIT ERROR

```
🐞: This request requires an authenticated account
```

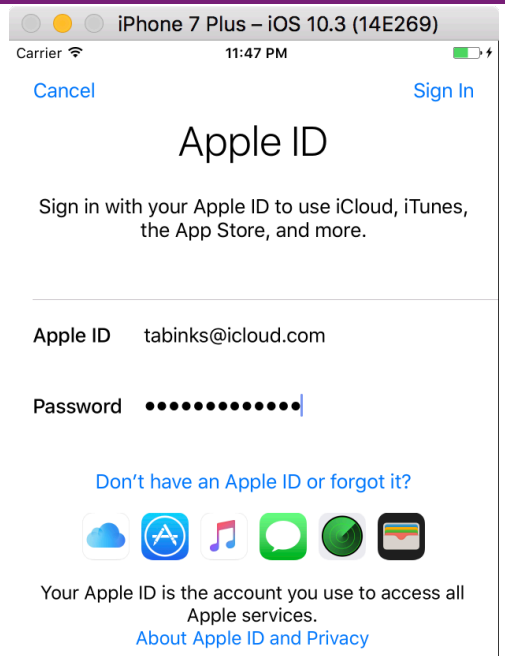
CLOUDKIT OBJECTS

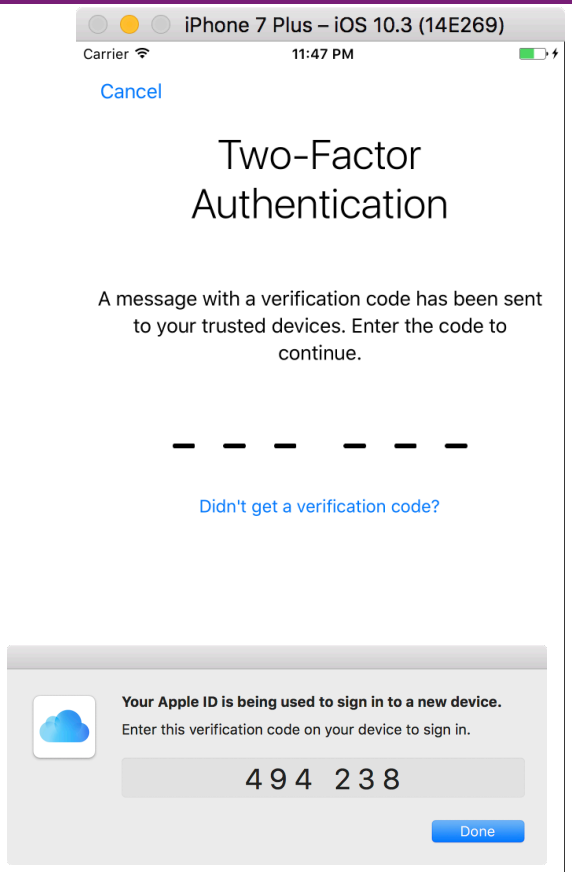
RECORDS

FULLY SUPPORTED BY THE SIMULATOR



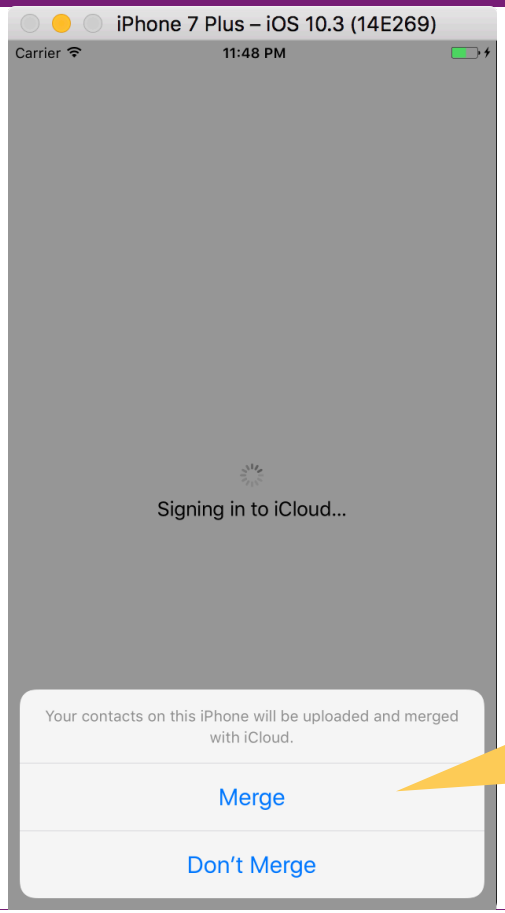
NEED TO LOGIN WITH ICLOUD ID



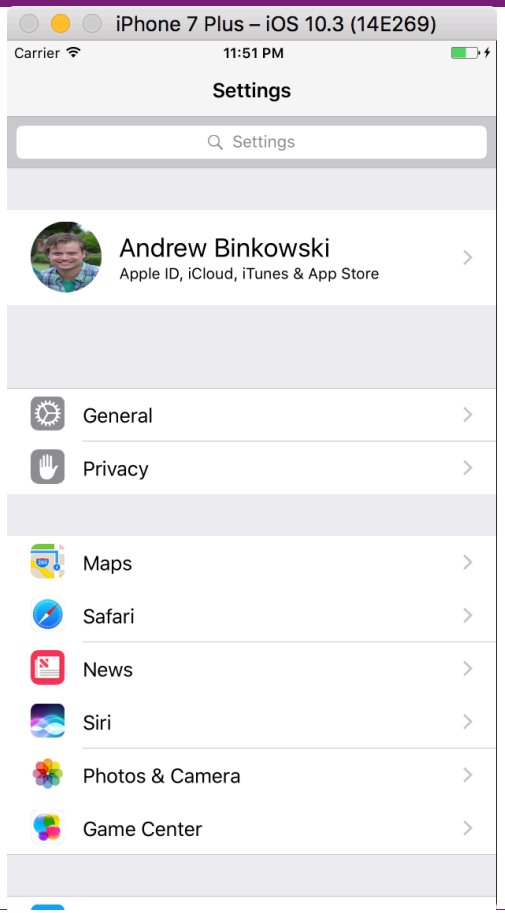


CLOUDKIT OBJECTS

RECORDS



BE CAREFUL...YOU PROBABLY DON'T WANT TO MERGE



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)")
    }
  }

}
```

CloudyWithAChanceOfErros

```
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

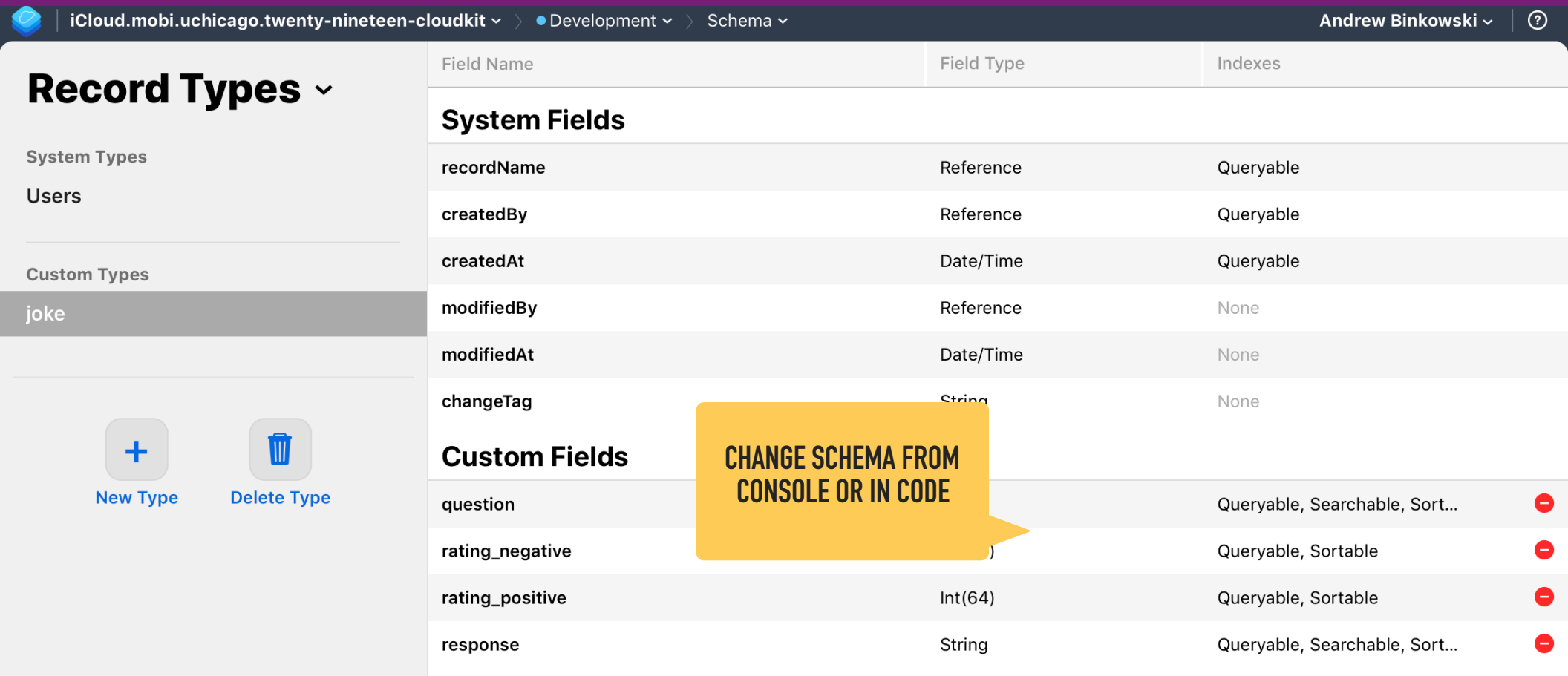
Query Records

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

Name	Type	Fields	Ch. Tag	Created	Modified
0FEB50A6-282E-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

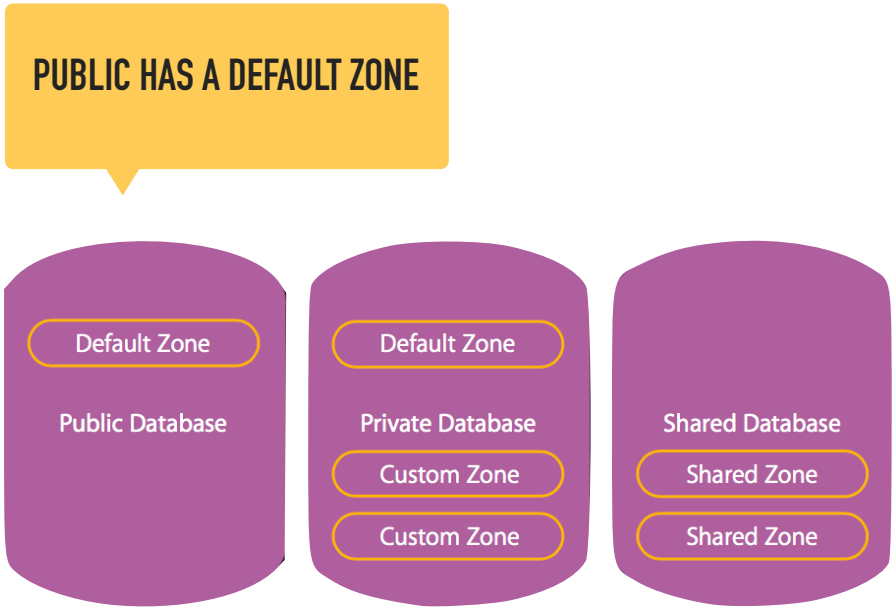


RECORD ZONES

CLOUDKIT OBJECTS

RECORD ZONES

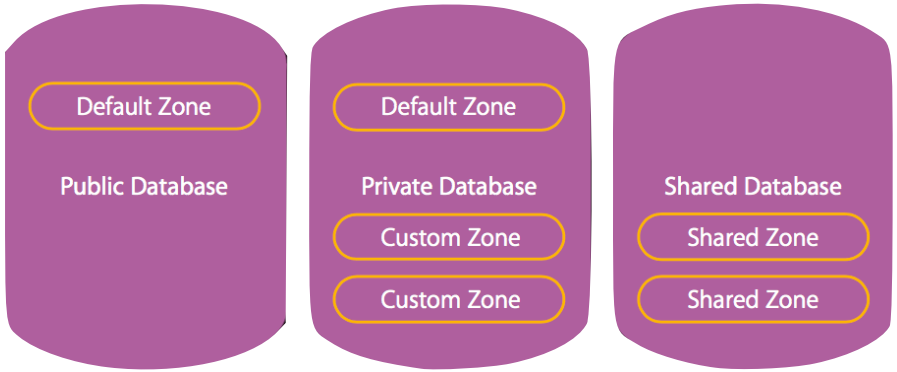
- CKRecordZones
  - Zones are a useful way to arrange a discrete group of records
  - Supported only in Private and Shared database
  - Atomic group writes



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
//

#import <Foundation/Foundati

@class CKRecordZoneID;

NS_CLASS_AVAILABLE(10_10, 8_0)
@interface CKRecordID : NSObject <NSSecureCodi

- (instancetype)init NS_UNAVAILABLE;

/* Record names must be 255 characters or less
/* This creates a record ID in the default zon
- (instancetype)initWithRecordName:(NSString *
- (instancetype)initWithRecordName:(NSString *

@property (nonatomic, readonly, strong) NSStri
@property (nonatomic, readonly, strong) CKReco

@end
```

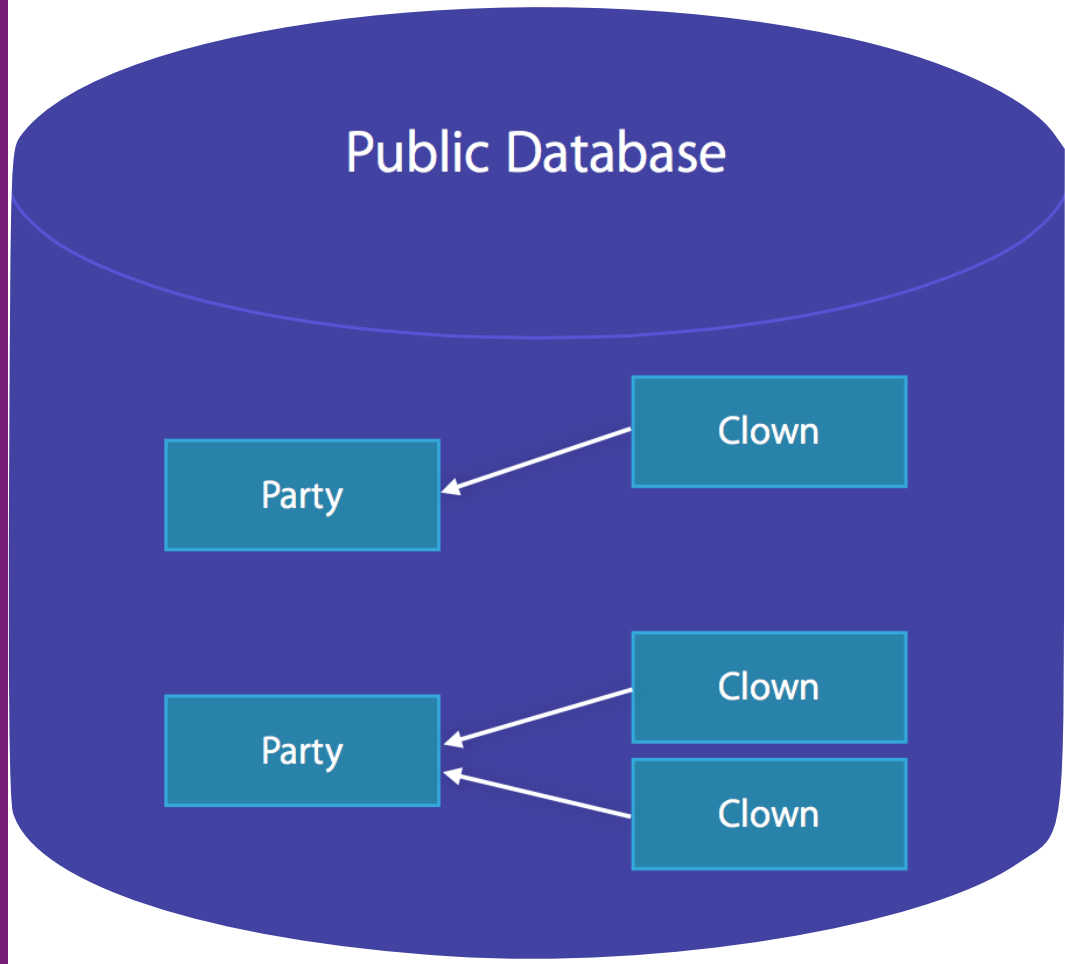
CKRECORDZONEID

REFERENCES

CLOUDKIT OBJECTS

RECORDS

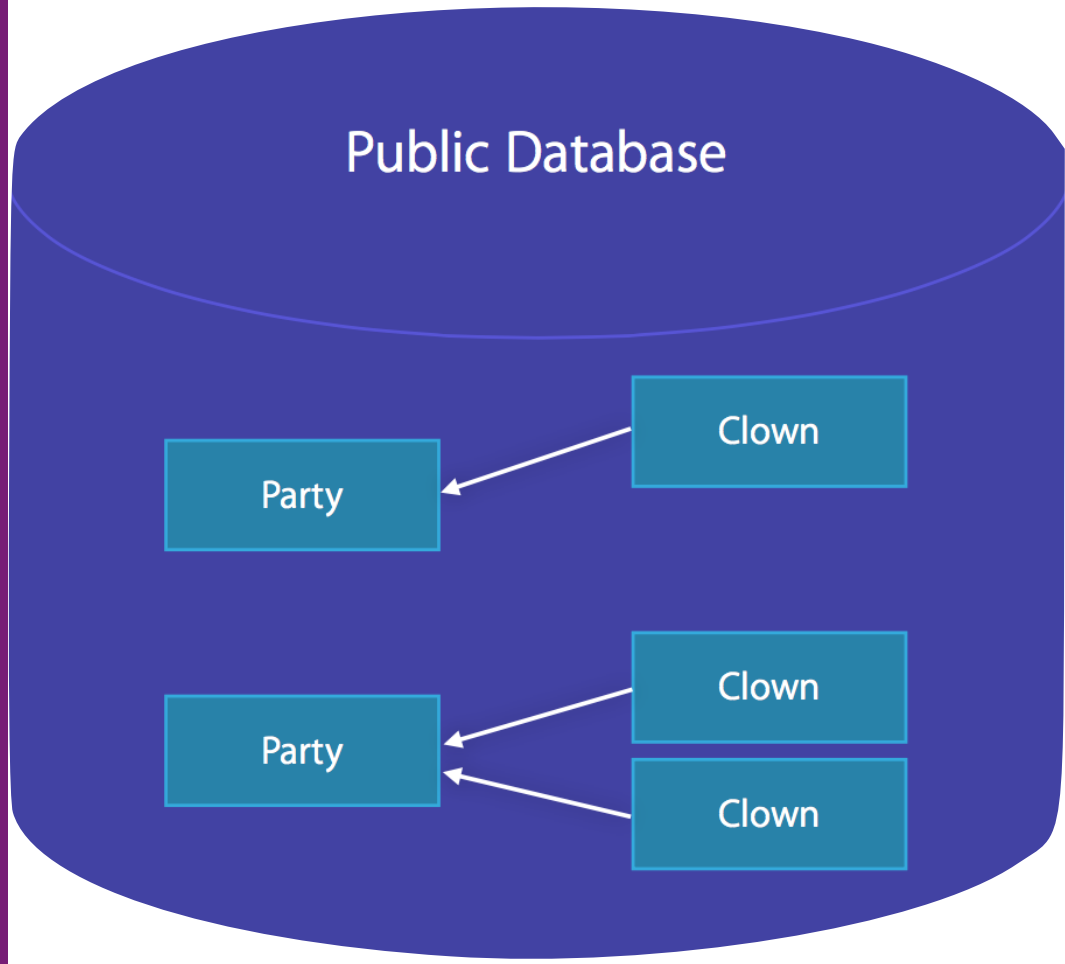
- CKReference
  - Points to a another record
  - Server understands relationship (takes some responsibility for managing; if you want)



CLOUDKIT OBJECTS

RECORDS

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

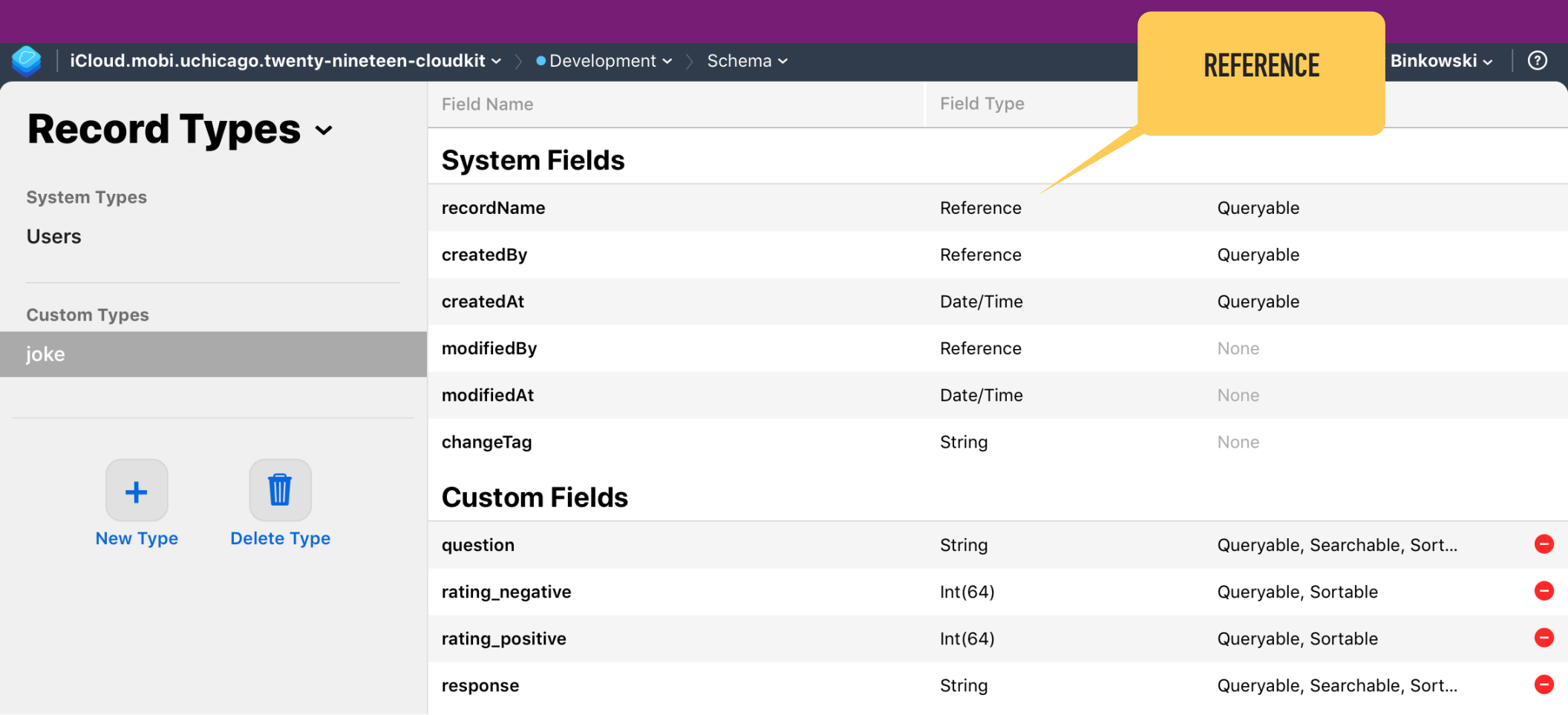


CLOUDKIT API

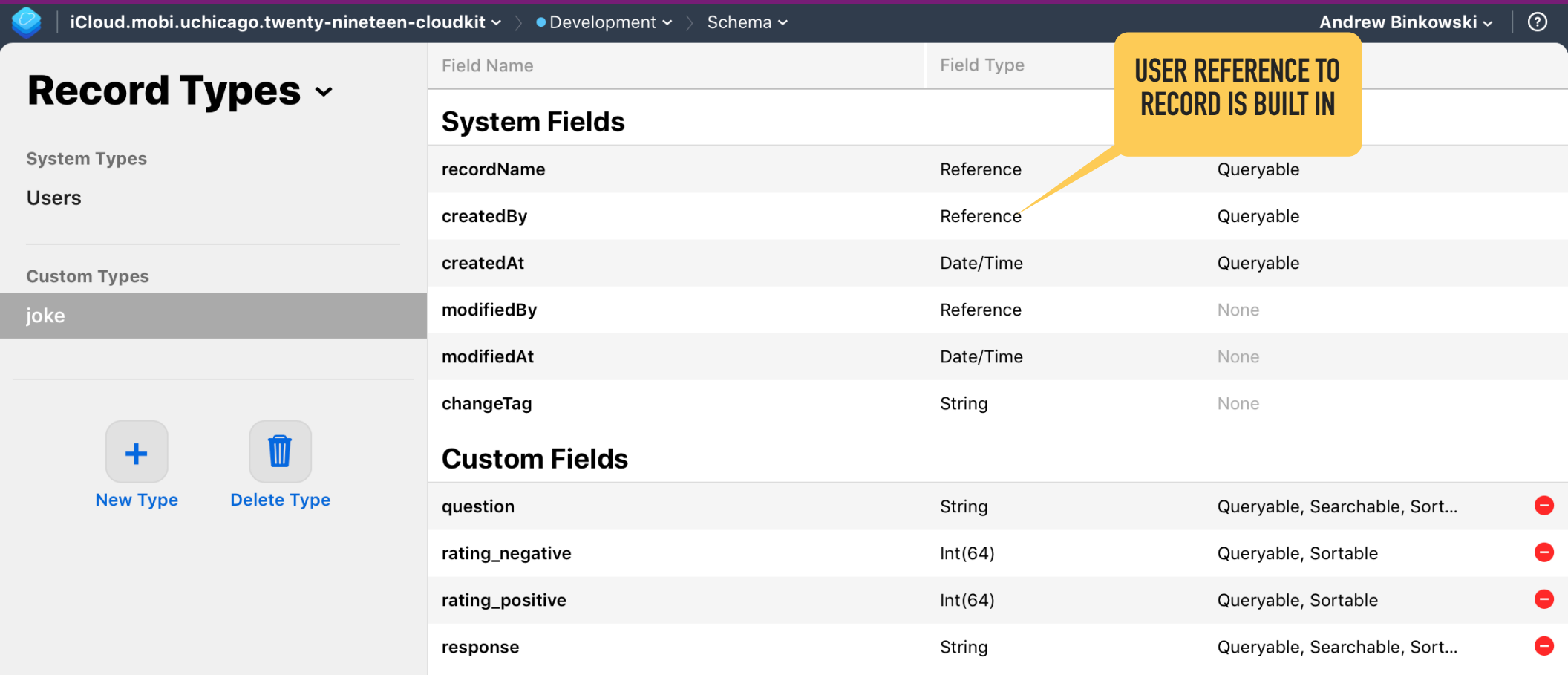
```
// Create a reference with a record
// Set the delete 'action' preference
let reference = CKReference(recordID: recordID,
                            action: .none)
```

WHAT HAPPENS ON DELETE

CLOUDKIT API



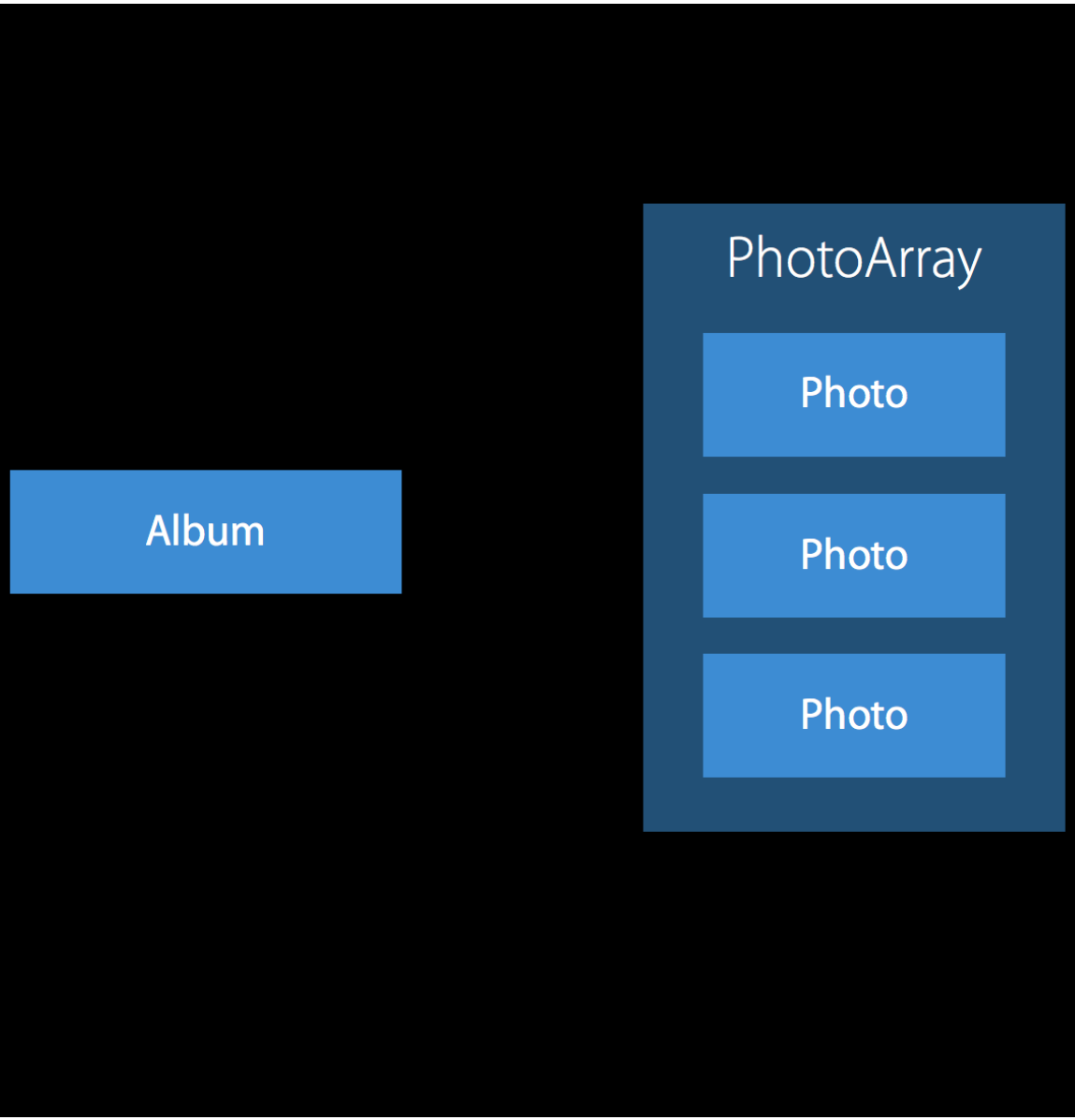
CLOUDKIT API



DATA MODELING

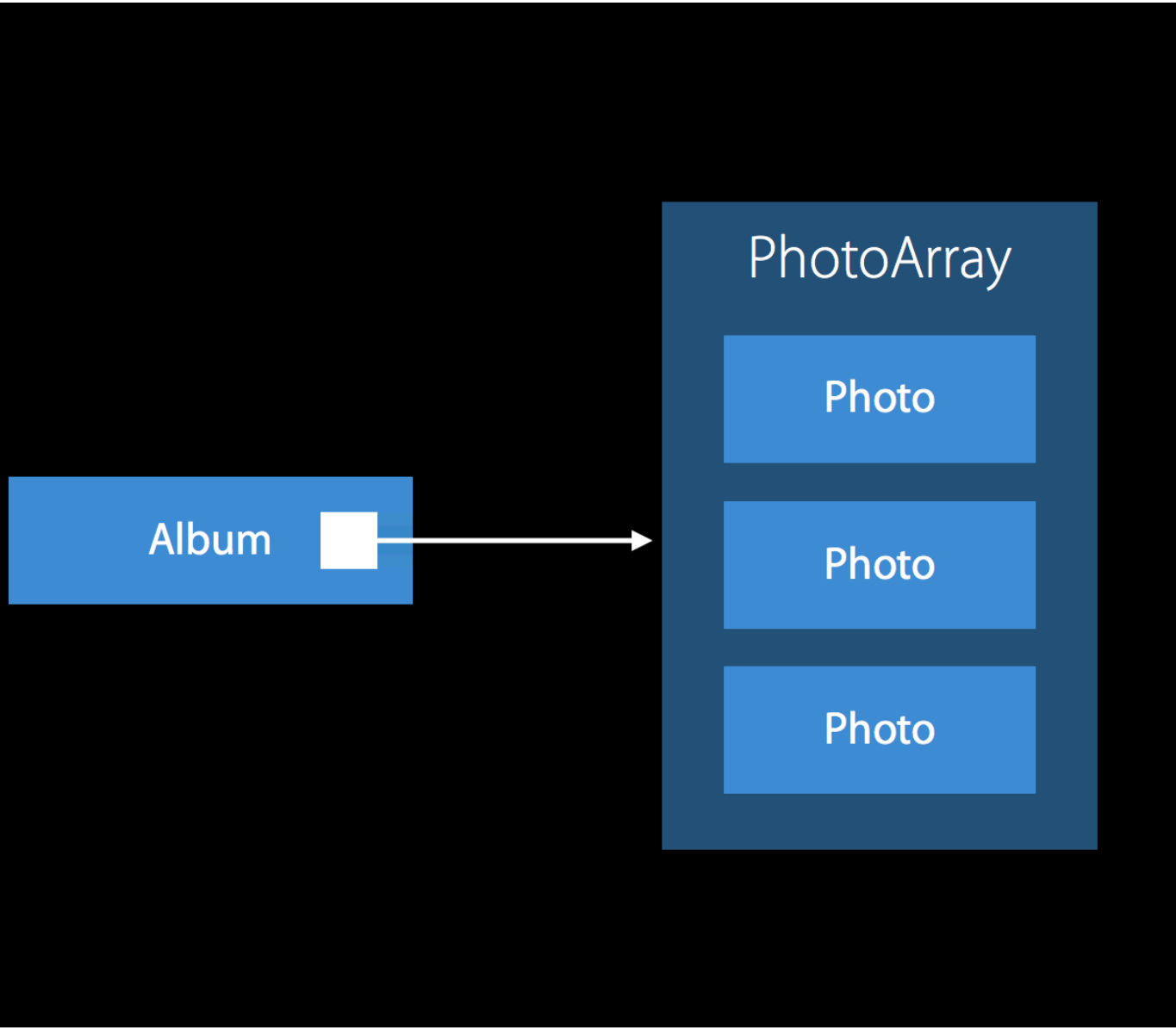
DATA MODELING

- How should 1 to many relationships be handled?



DATA MODELING

- Typical setup to model



DATA MODELING

PhotoArray = []

Album

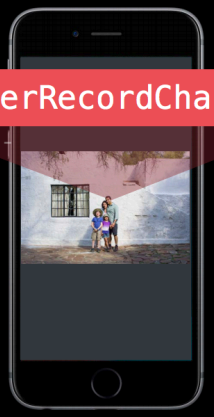
DATA MODELING

PhotoArray = []

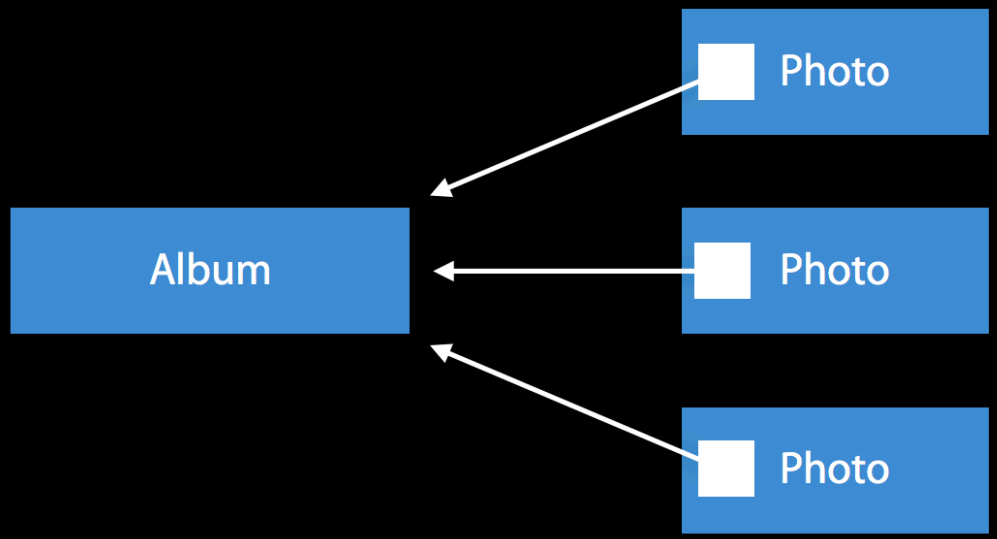
Album

serverRecordChanged

serverRecordChanged



DATA MODELING

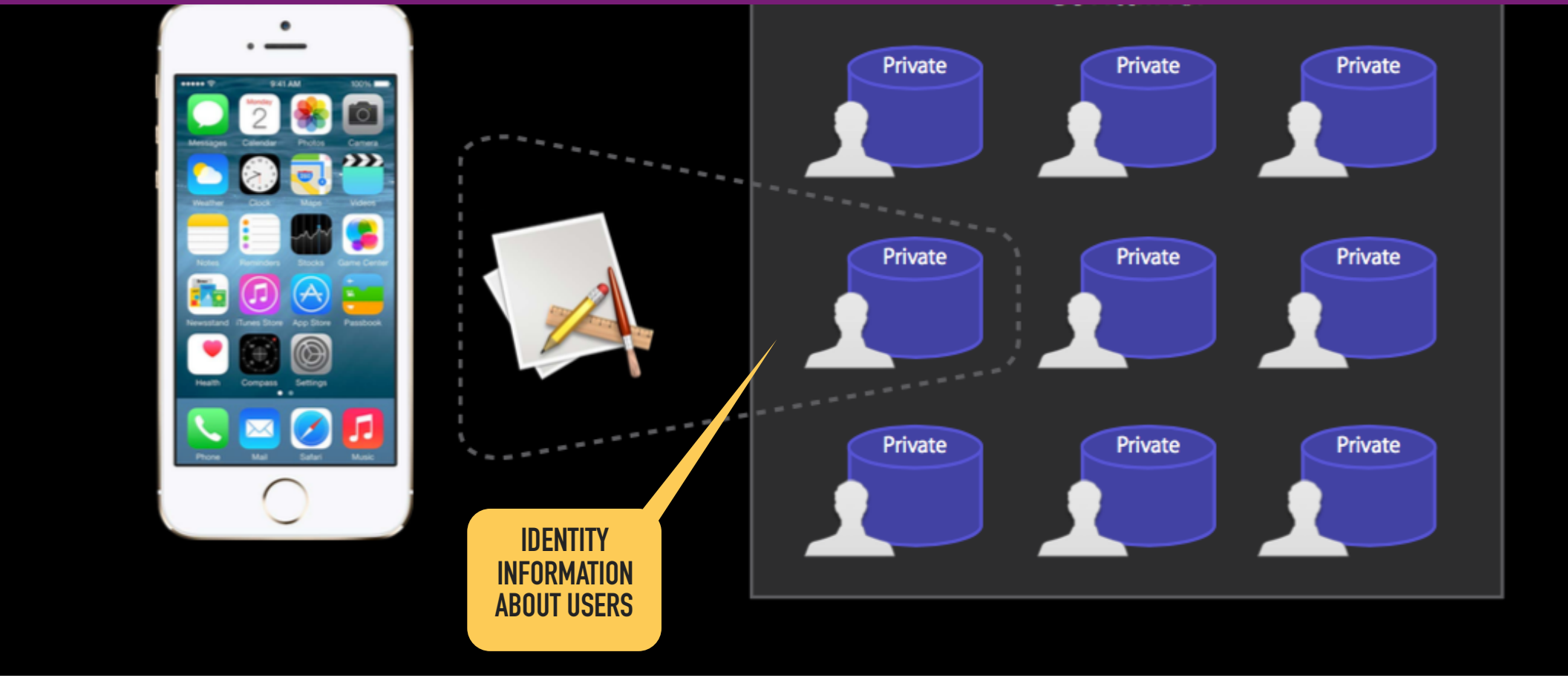


USER RECORDS

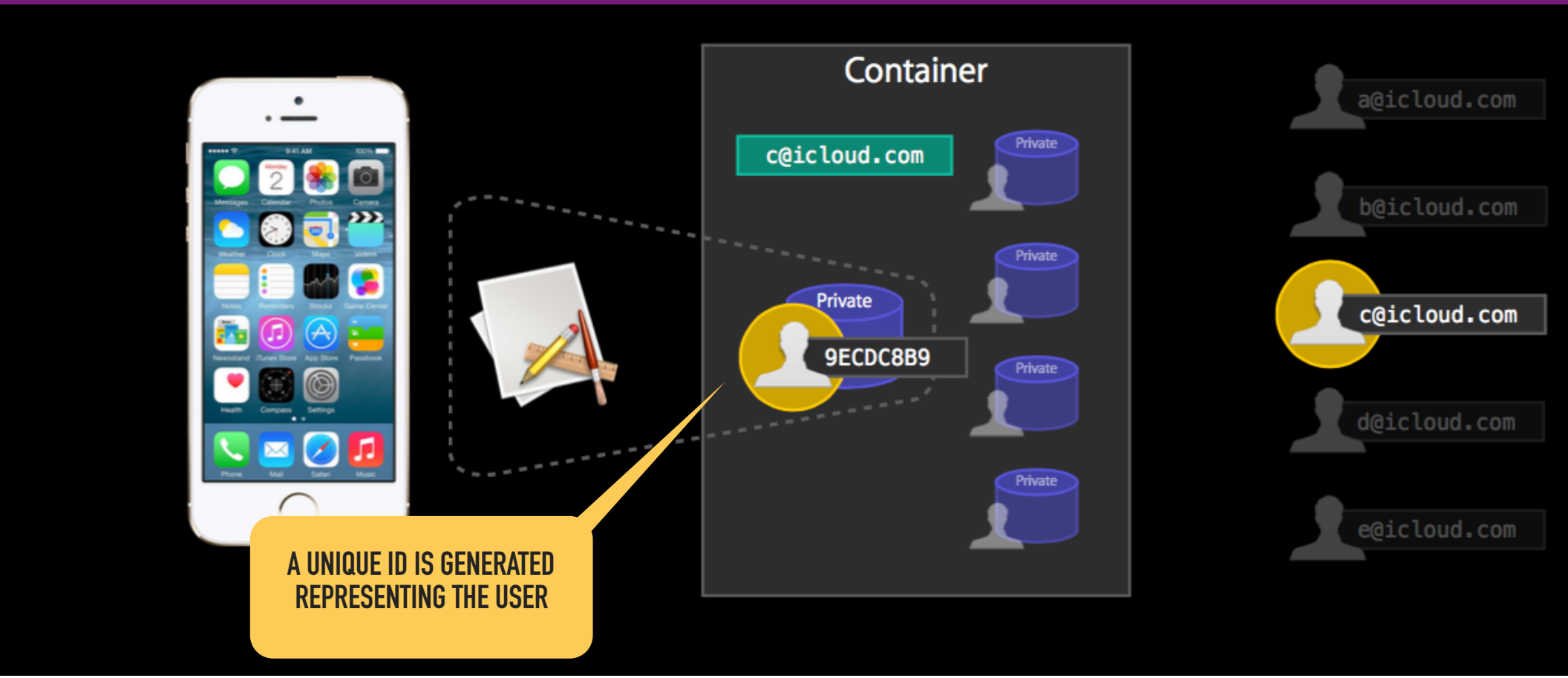
USER RECORDS

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

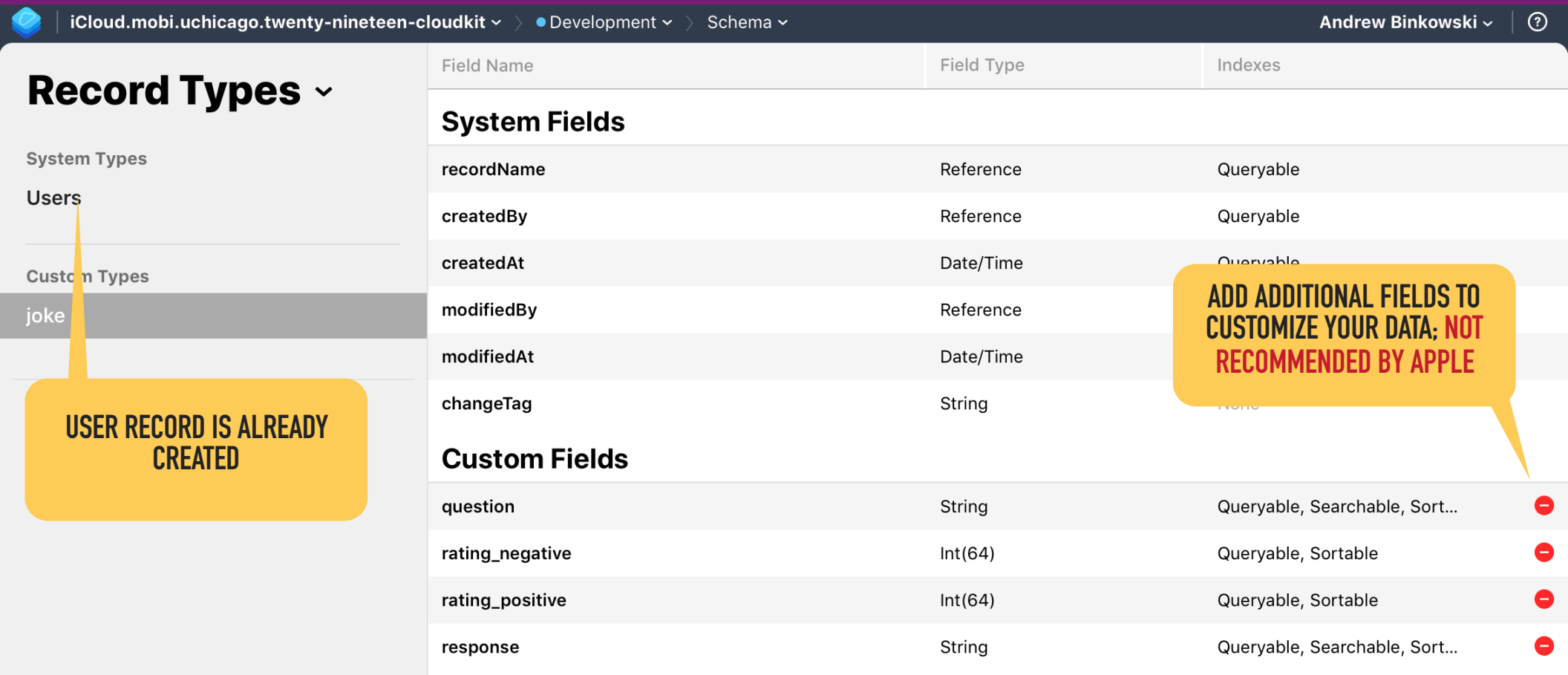
USER RECORDS



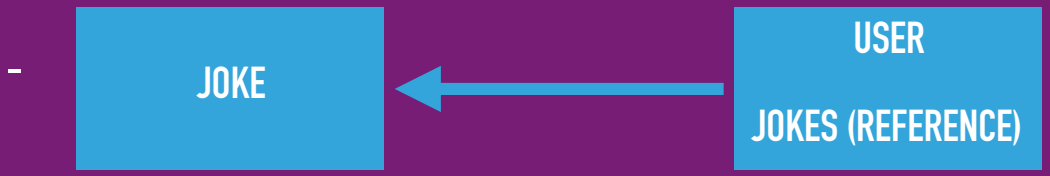
USER RECORDS

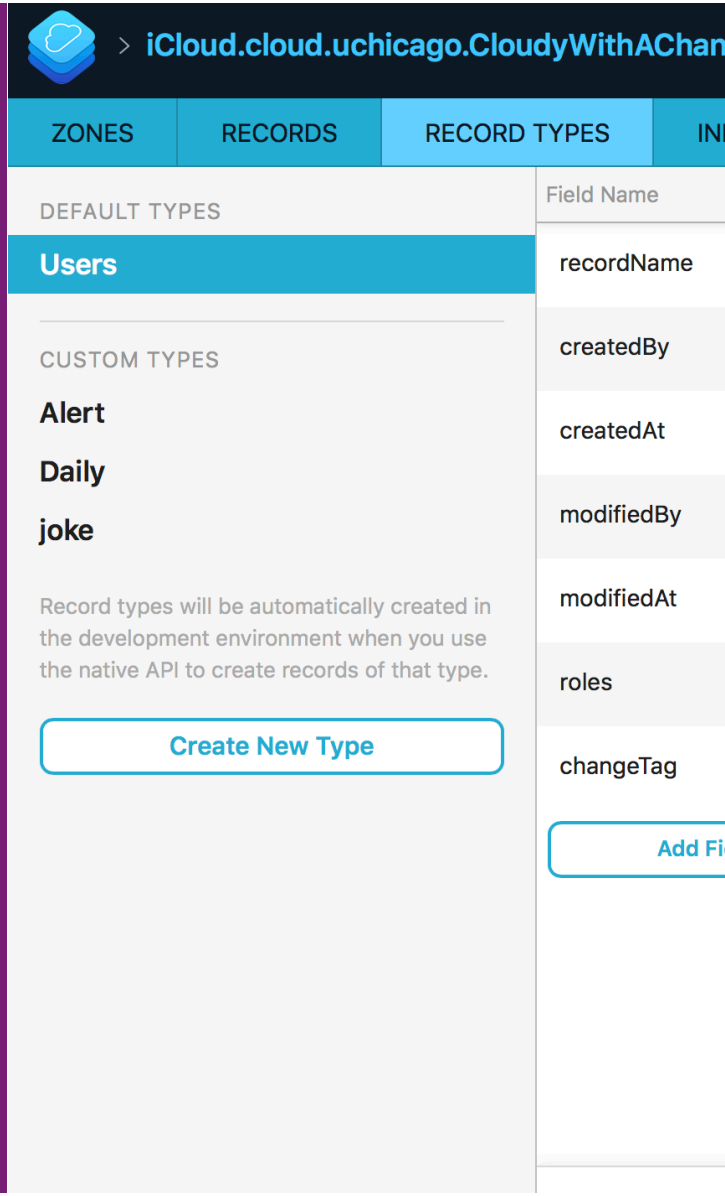


USER RECORDS



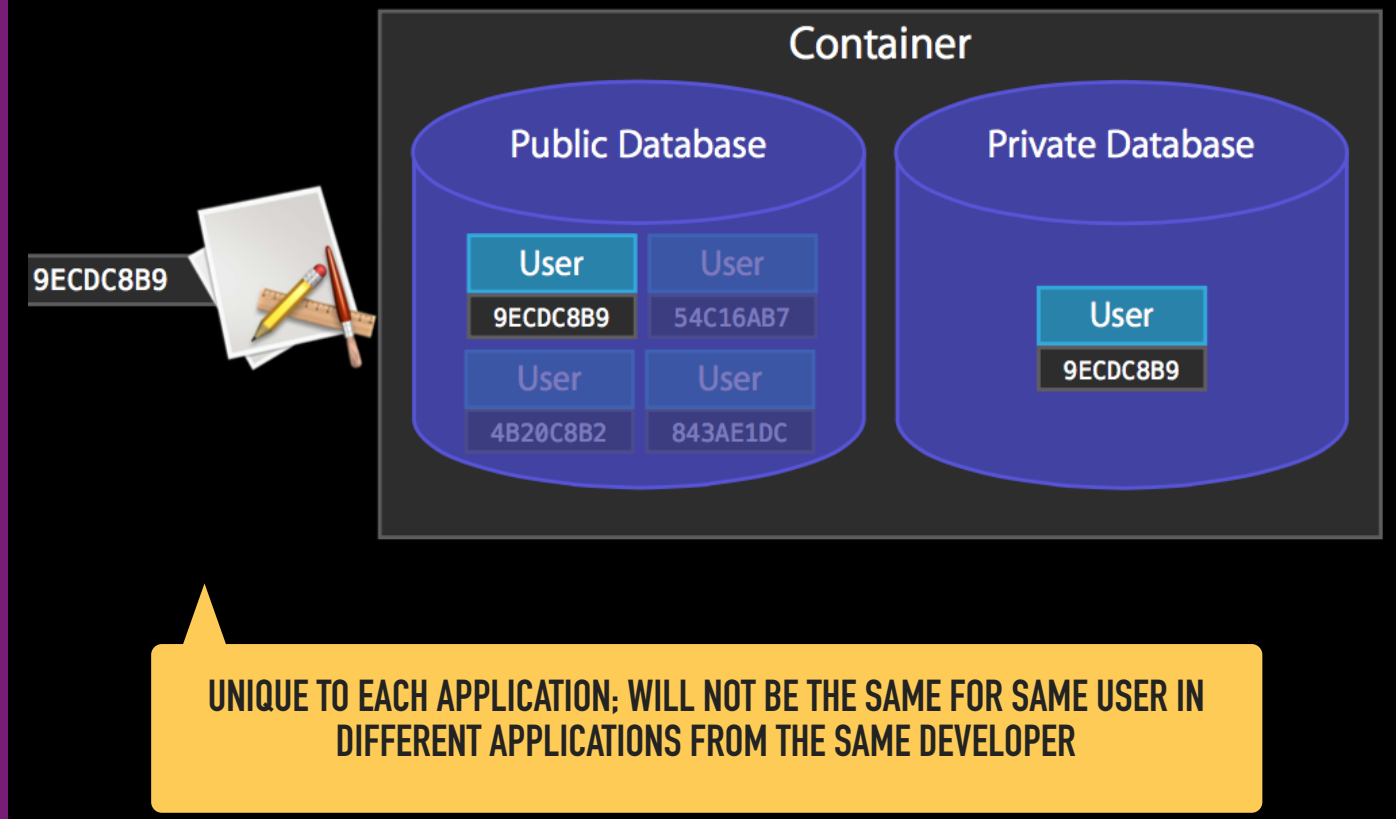
USER RECORDS

- 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
  - 



CloudKit User Accounts

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



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)
      }
}
```

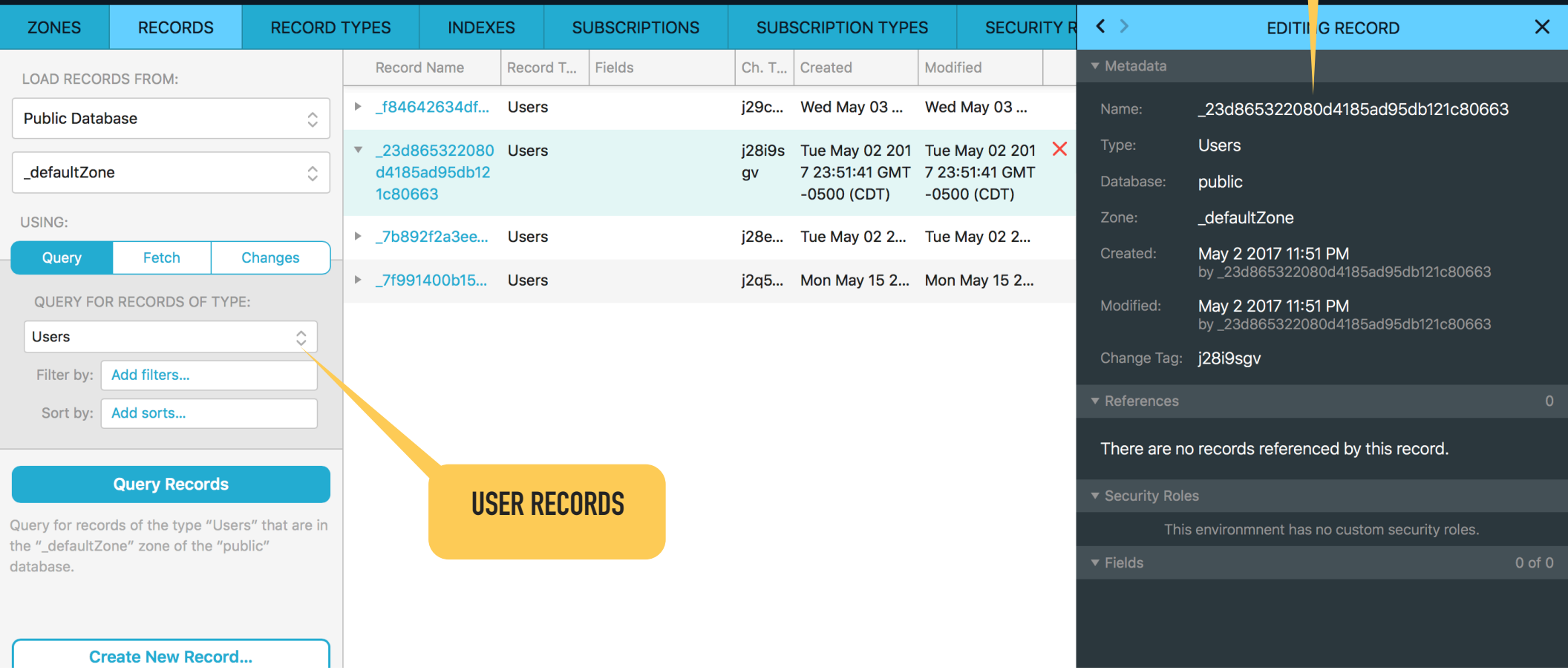
USER RECORDS

```
let container = CKContainer.default()

container.fetc
      recordID

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

"_23D865322080D4185AD95DB121C80663"

USER RECORDS

"_23D865322080D4185AD95DB121C80663"



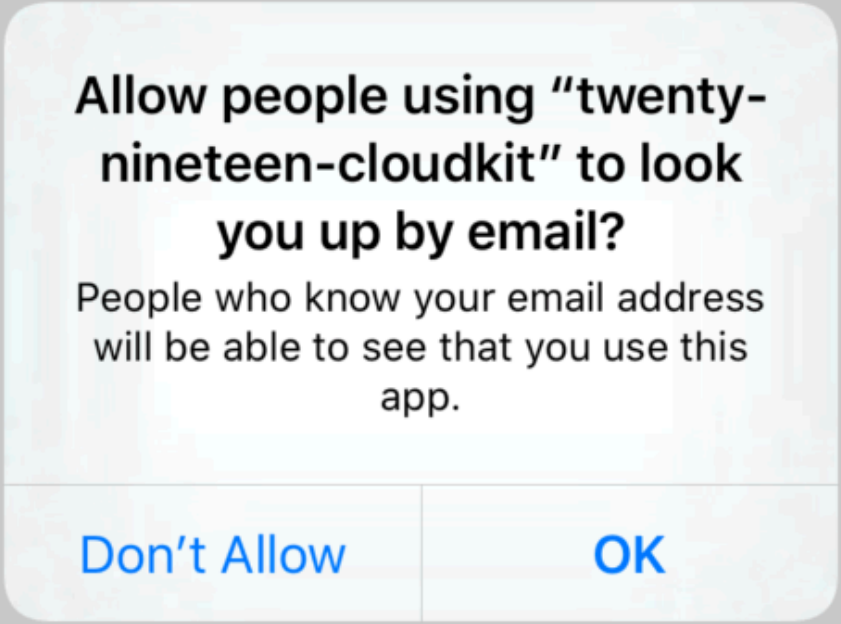
USER RECORDS

```
/// 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")
  }
}
```

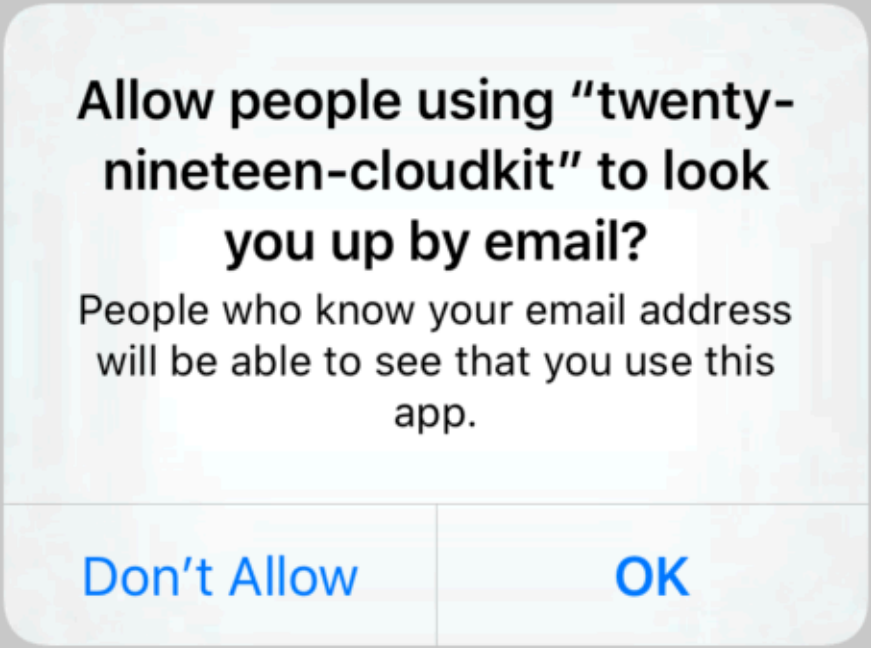
USER RECORDS

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



USER RECORDS

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



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)
        })
      }
    }
```

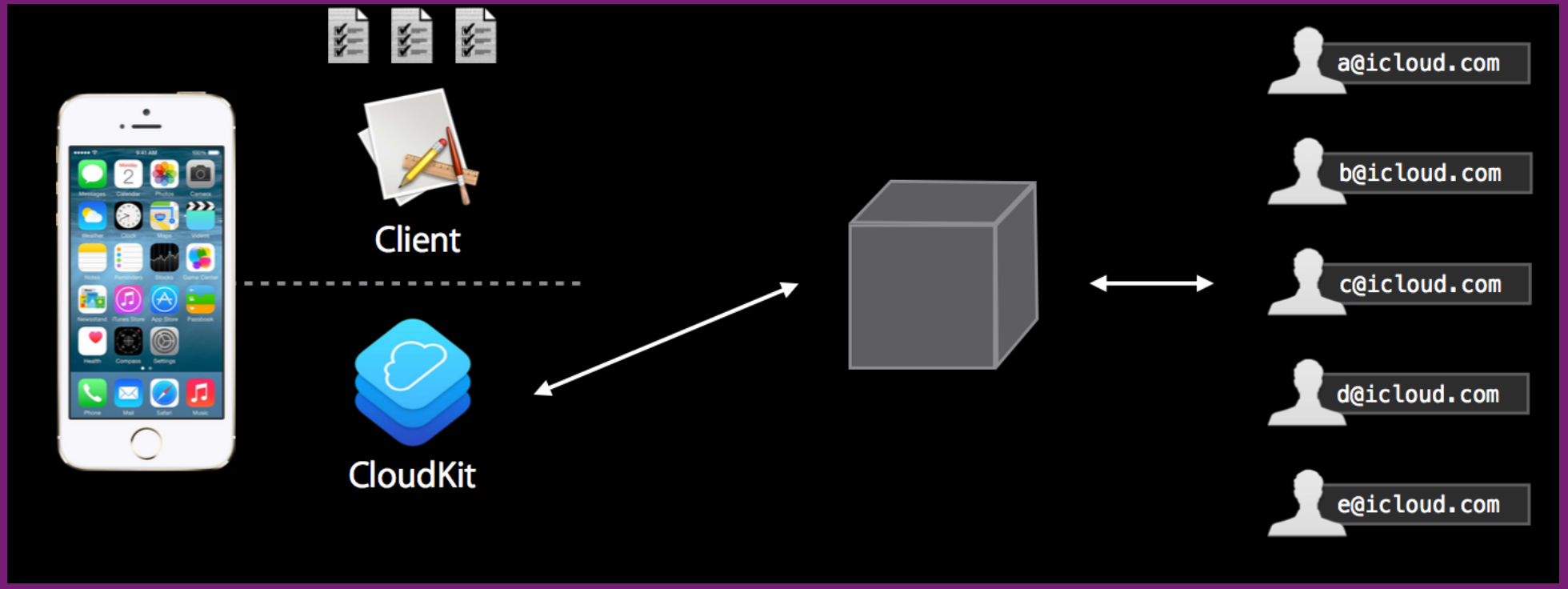
```
CKContainer.default().requestApplicationPermission(.userDiscoverability) { (status, error) in
  CKContainer.default().fetchUserRecordID { (record, error) in
    CKContainer.default().discoverUserIdentity(withUserRecordID: record!, completionHandler: { (userID, error) in
      userName = (userID?.nameComponents?.givenName)!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

CloudyWithAChanceOfErros

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

USER RECORDS

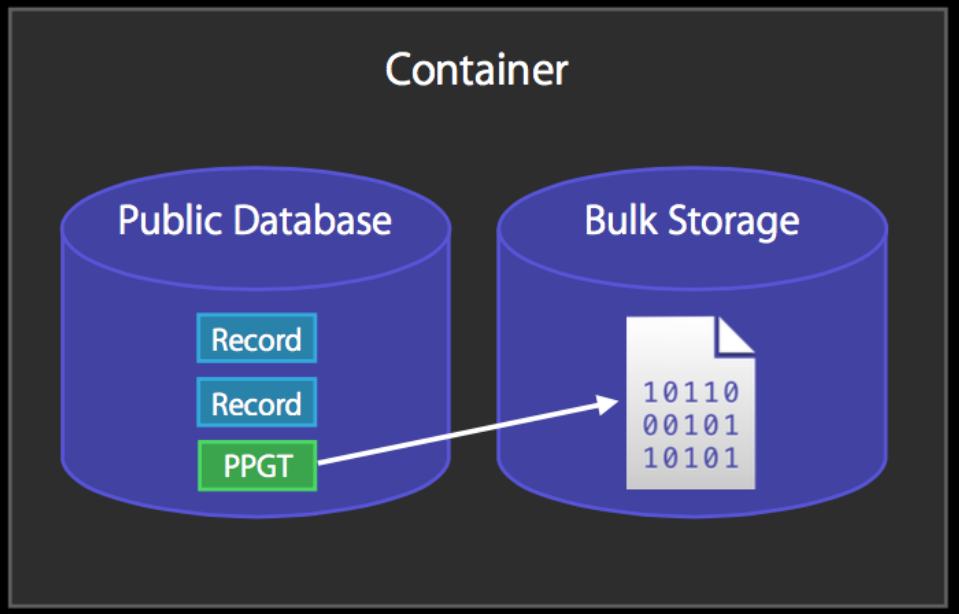


- You can get all friends who use the same app (discover users api)

ASSETS

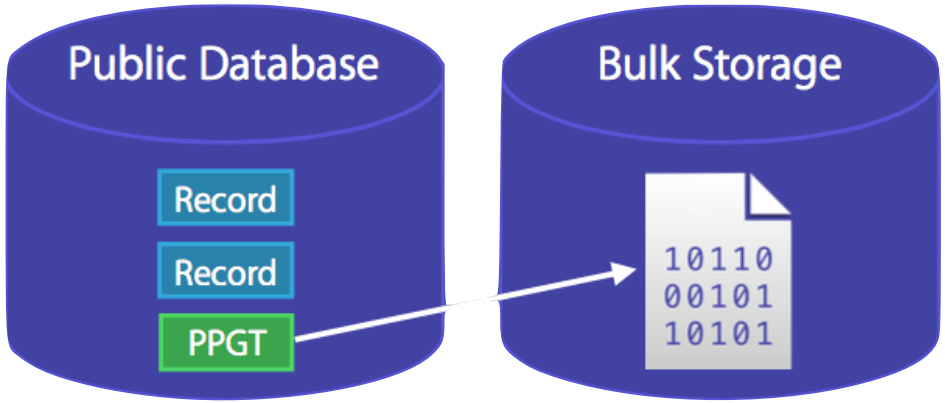
ASSETS

- CKAssets is bulk storage for CloudKit

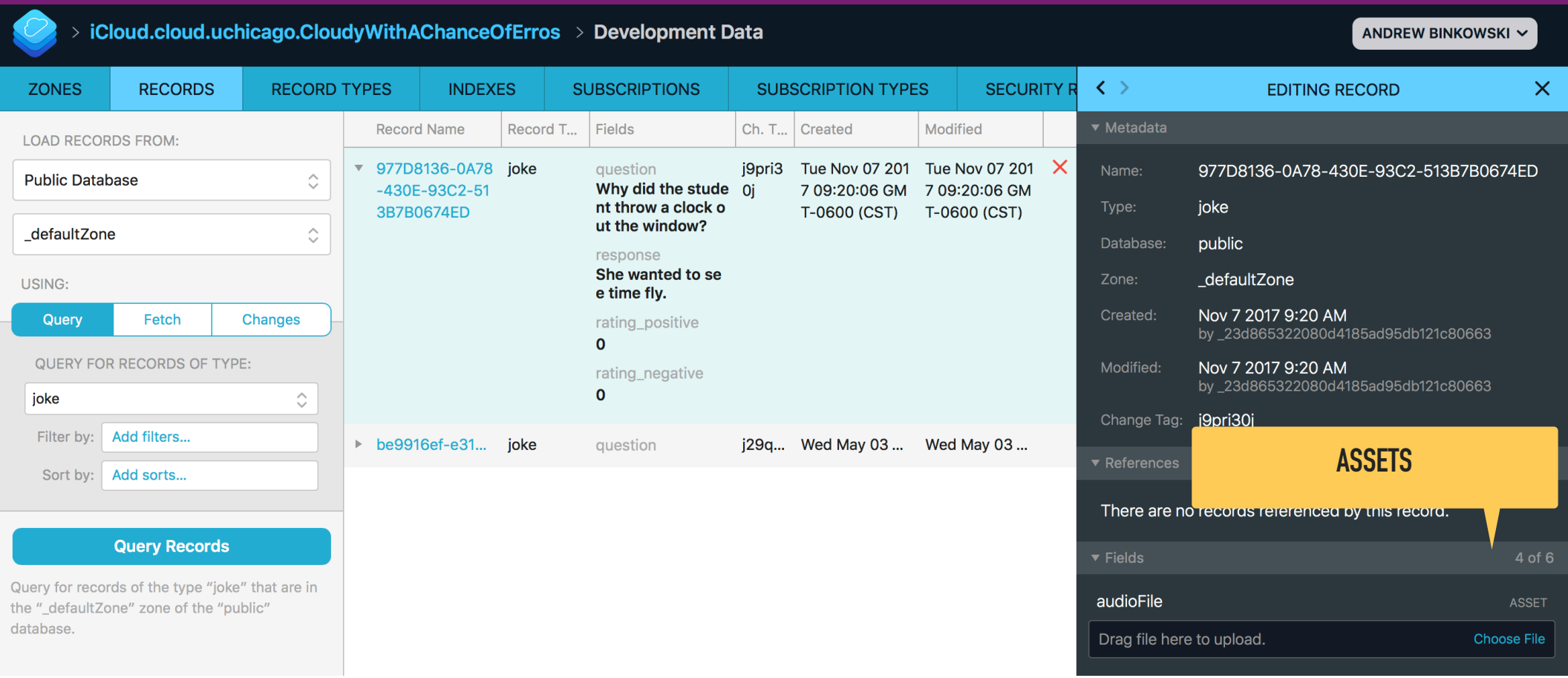


ASSETS

- 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.CloudyWithAChanceOfErros
Development Data
ANDREW BINKOWSKI
ZONES
RECORDS
RECORD TYPES
INDEXES
SUBSCRIPTIONS
SUBSCRIPTION TYPES
SECURITY R
LOAD RECORDS FROM:
Public Database
_defaultZone
USING:
Query
Fetch
Changes
QUERY FOR RECORDS OF TYPE:
joke
Filter by:
Add filters...
Sort by:
Add sorts...
Query Records
Query for records of the type "joke" that are in the "_defaultZone" zone of the "public" database.
Record Name
Record T...
Fields
Ch. T...
Created
Modified
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 GMT-0600 (CST)
Tue Nov 07 2017 09:20:06 GMT-0600 (CST)
be9916ef-e31...
joke
question
j29q...
Wed May 03 ...
Wed May 03 ...
EDITING RECORD
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
References
ASSETS
There are no records referenced by this record.
Fields
4 of 6
audioFile
ASSET
Drag file here to upload.
Choose File

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
}
```

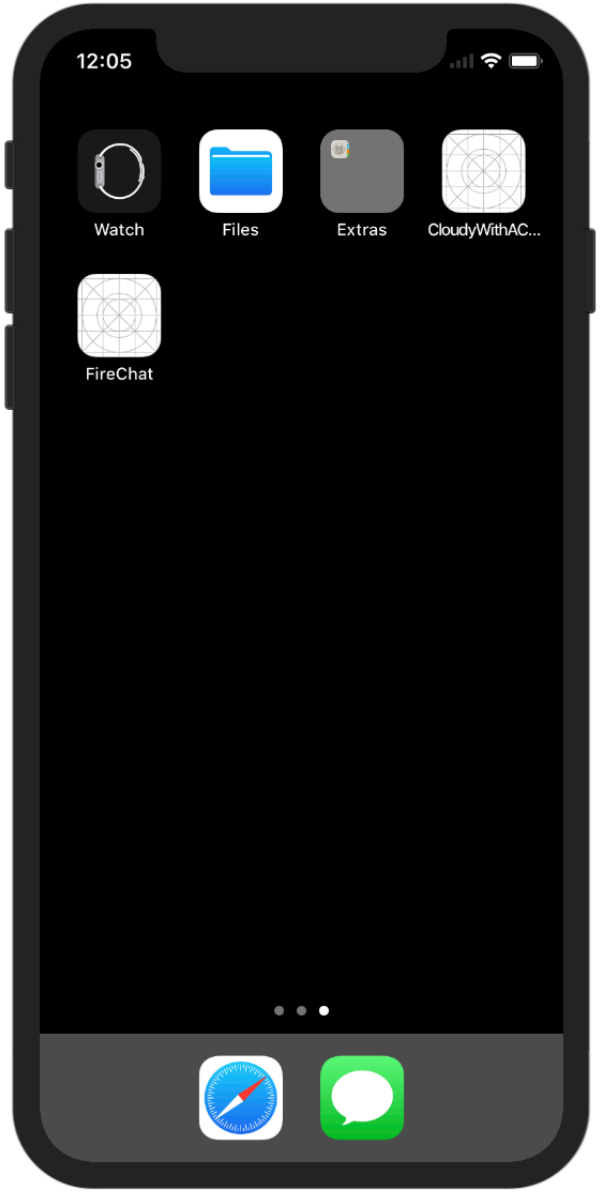
RETRIEVE AN IMAGE ASSOCIATED WITH A RECORD;
RETURNED A FILE THAT NEEDS TO BE ACTED ON IMMEDIATELY

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:@"%K = %@", 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 `CKReco
open func getJokesByCurrentUser(_ reco

  // The user is a reference, so we ne
  let reference = CKReference(recordID


  let predicate = NSPredicate(format:

  let query = CKQuery(recordType: "jo
  publicDB.perform(query, inZoneWith:
    if let error = error {
      print("Error: \(String(describin
      return
    }
    for record in records! {
      print(record["question"] as! St
    }
  }
}
```

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 folow 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 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

```
/// Use the operation API to make a query and use cursors
/// to control the folow 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) {
```

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) {
```

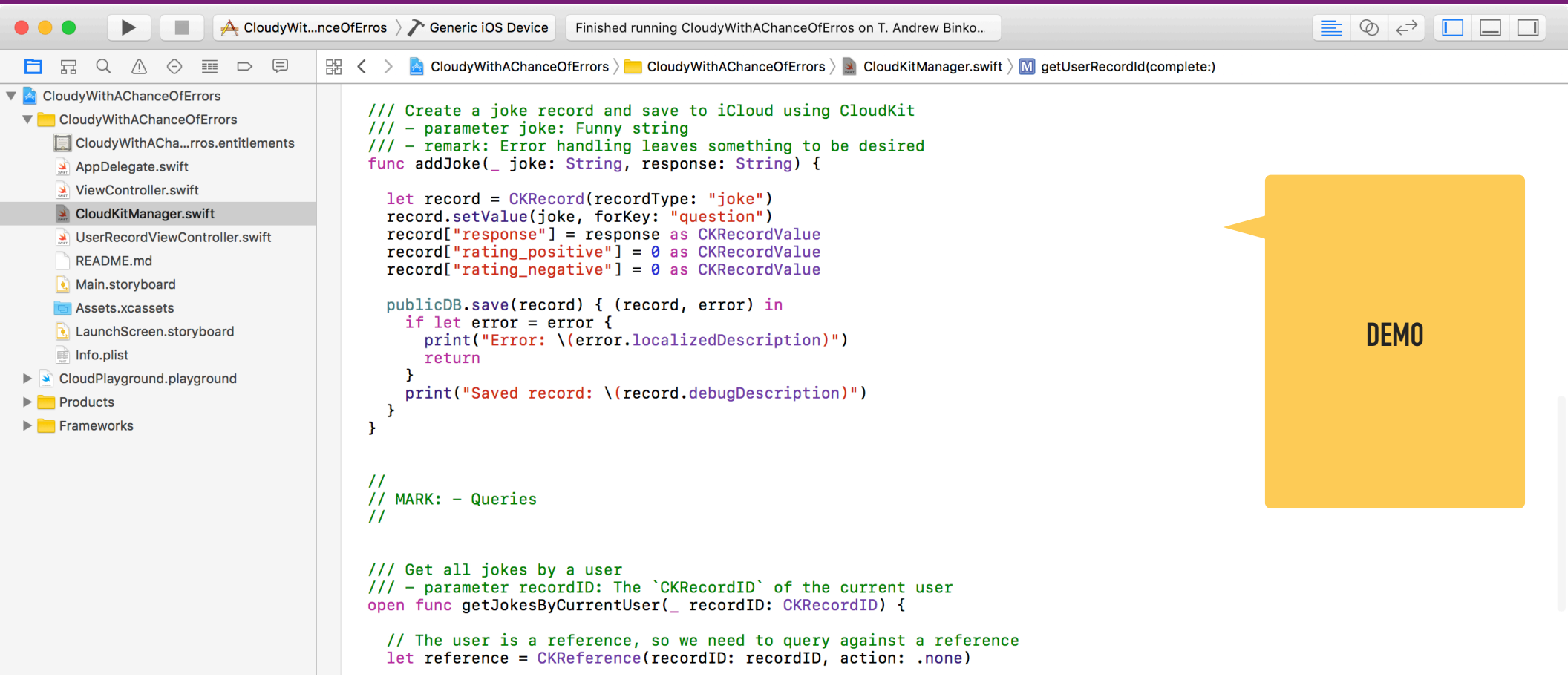
QUERIES

```
// 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 opeartion...")
    //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

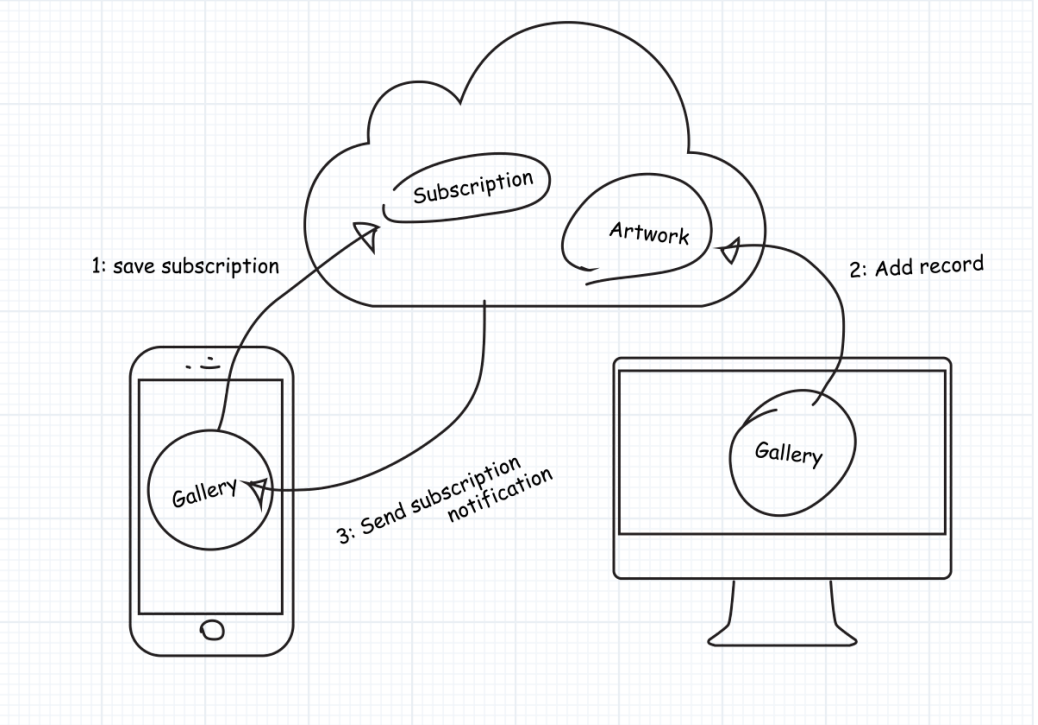


DEMO

SUBSCRIPTIONS

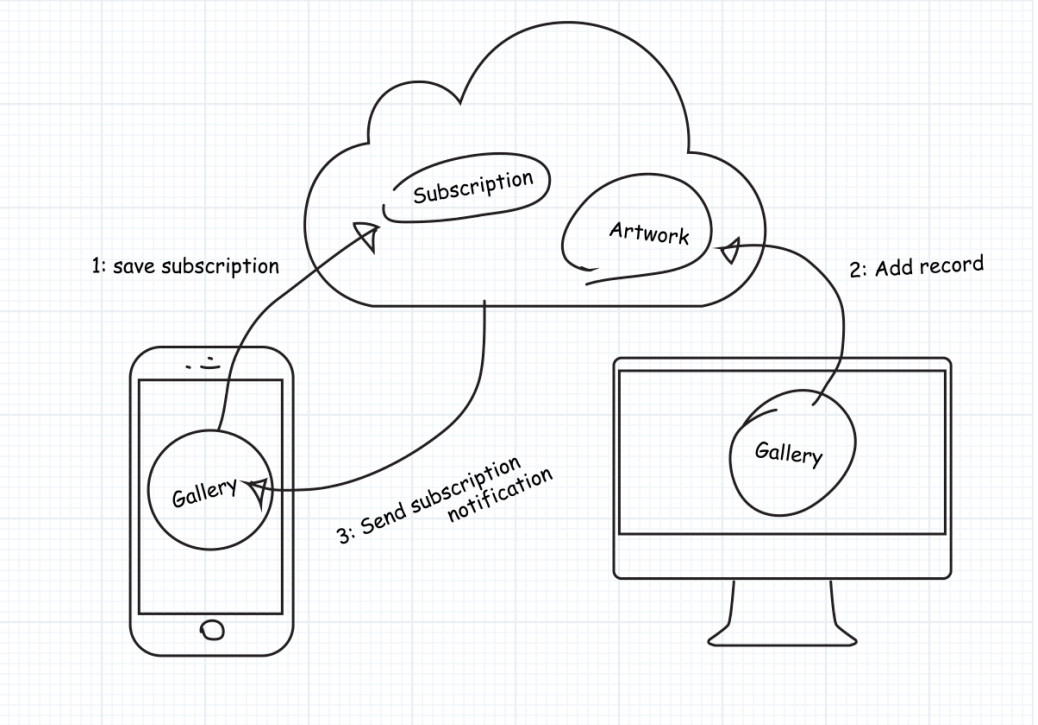
SUBSCRIPTIONS

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

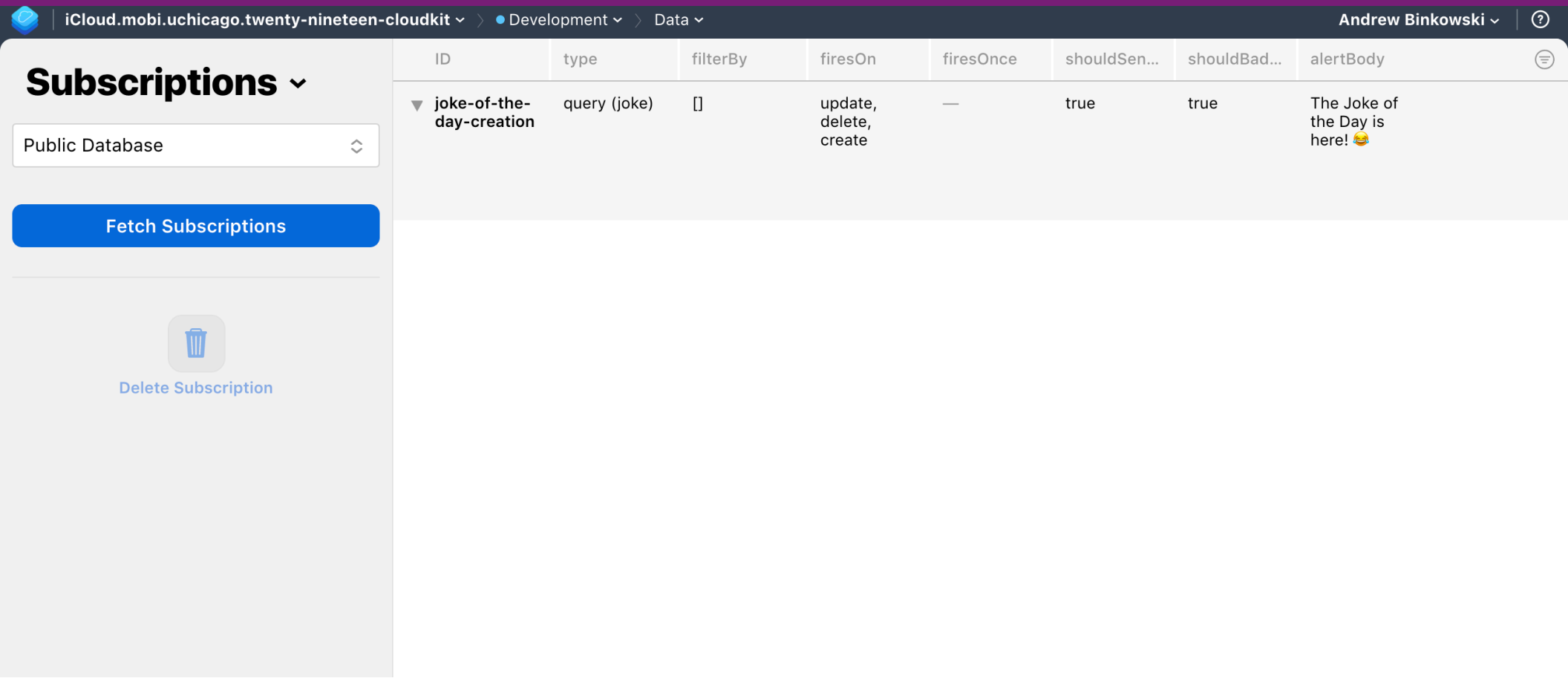


SUBSCRIPTIONS

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

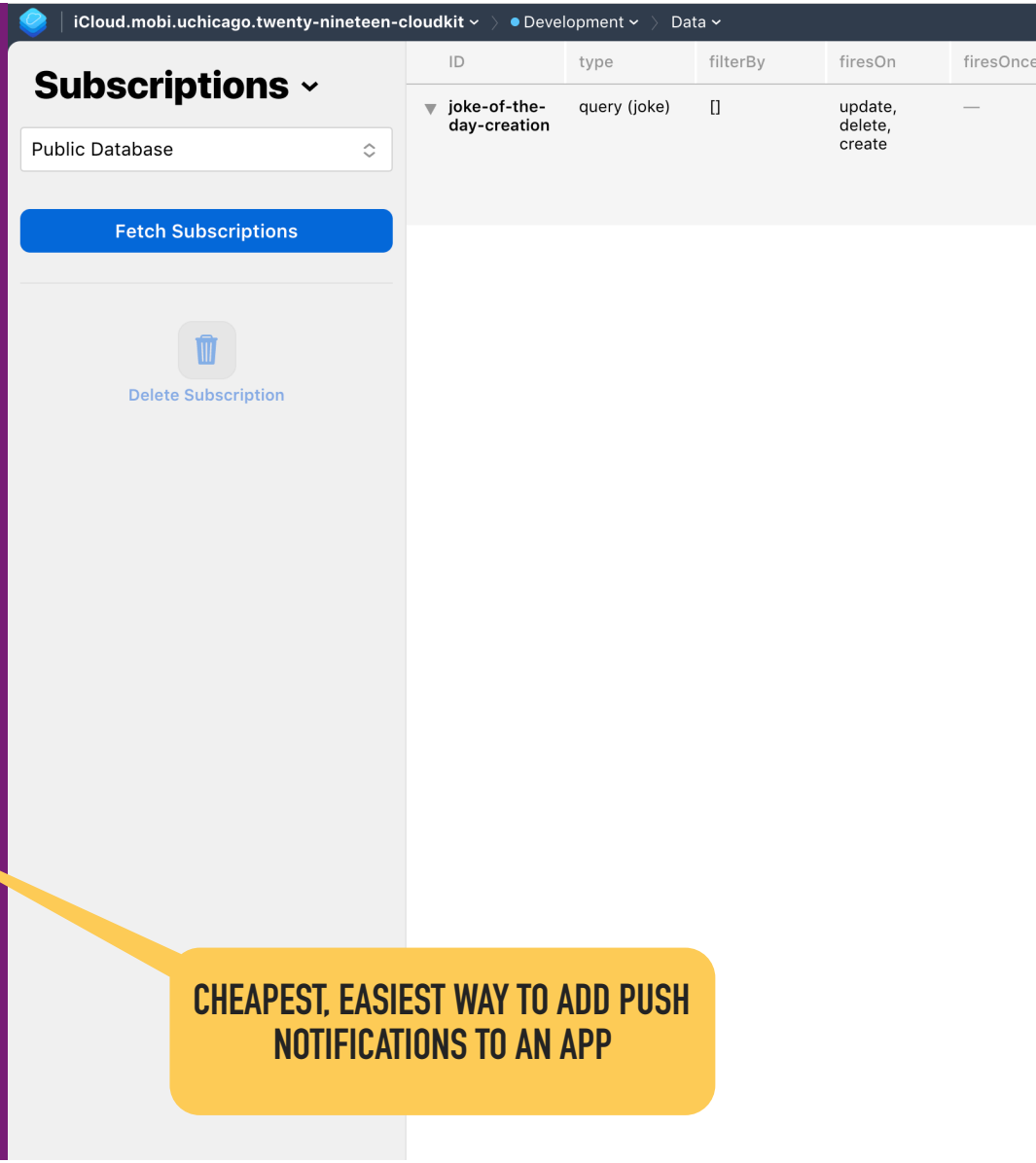


SUBSCRIPTIONS



SUBSCRIPTIONS

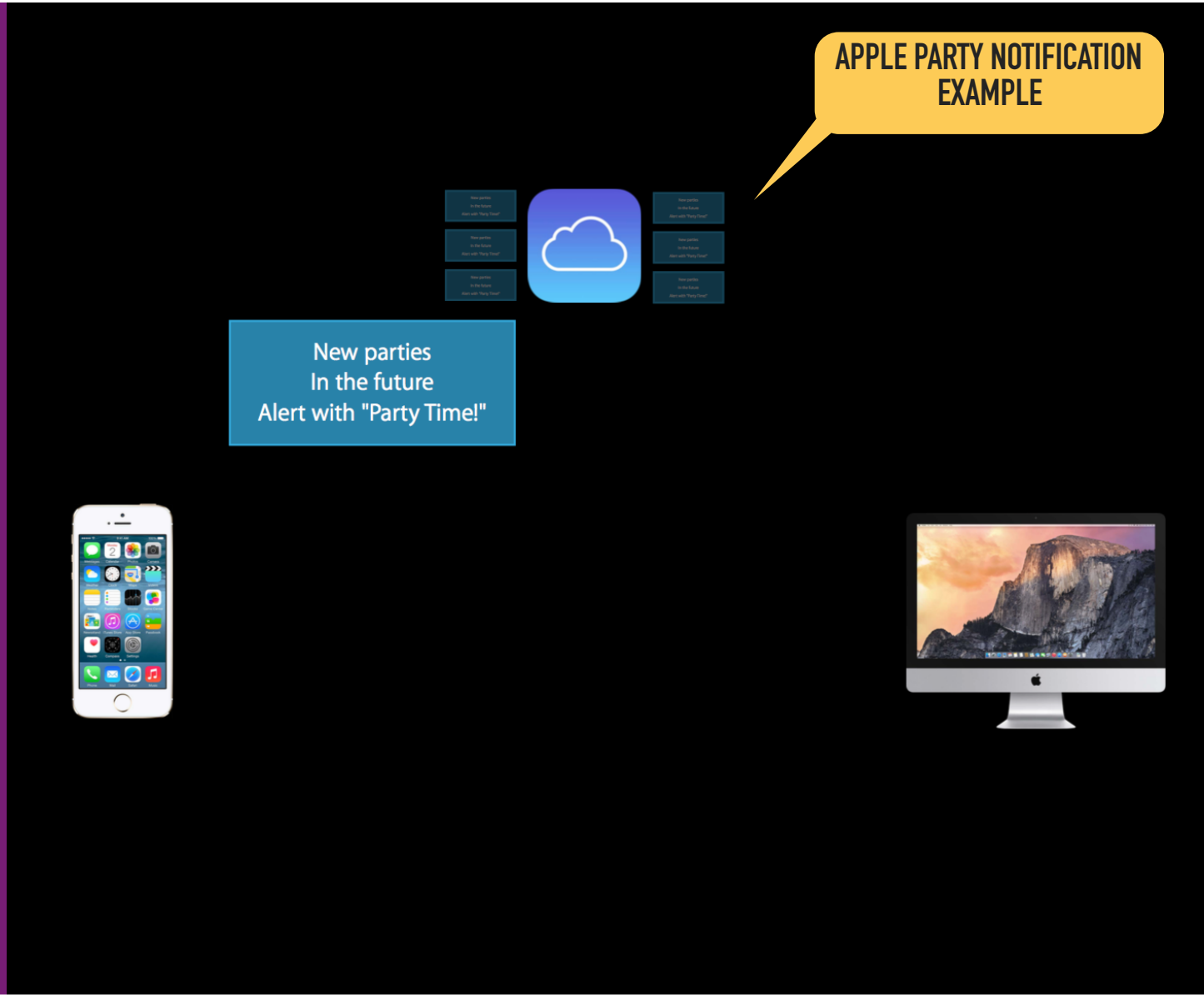
- CKQuerySubscription
  - RecordType
  - NSPredicate
  - Push
- Push delivered via Apple Push Notification Service (APNS) augmented payload
  - JSON



CHEAPEST, EASIEST WAY TO ADD PUSH NOTIFICATIONS TO AN APP

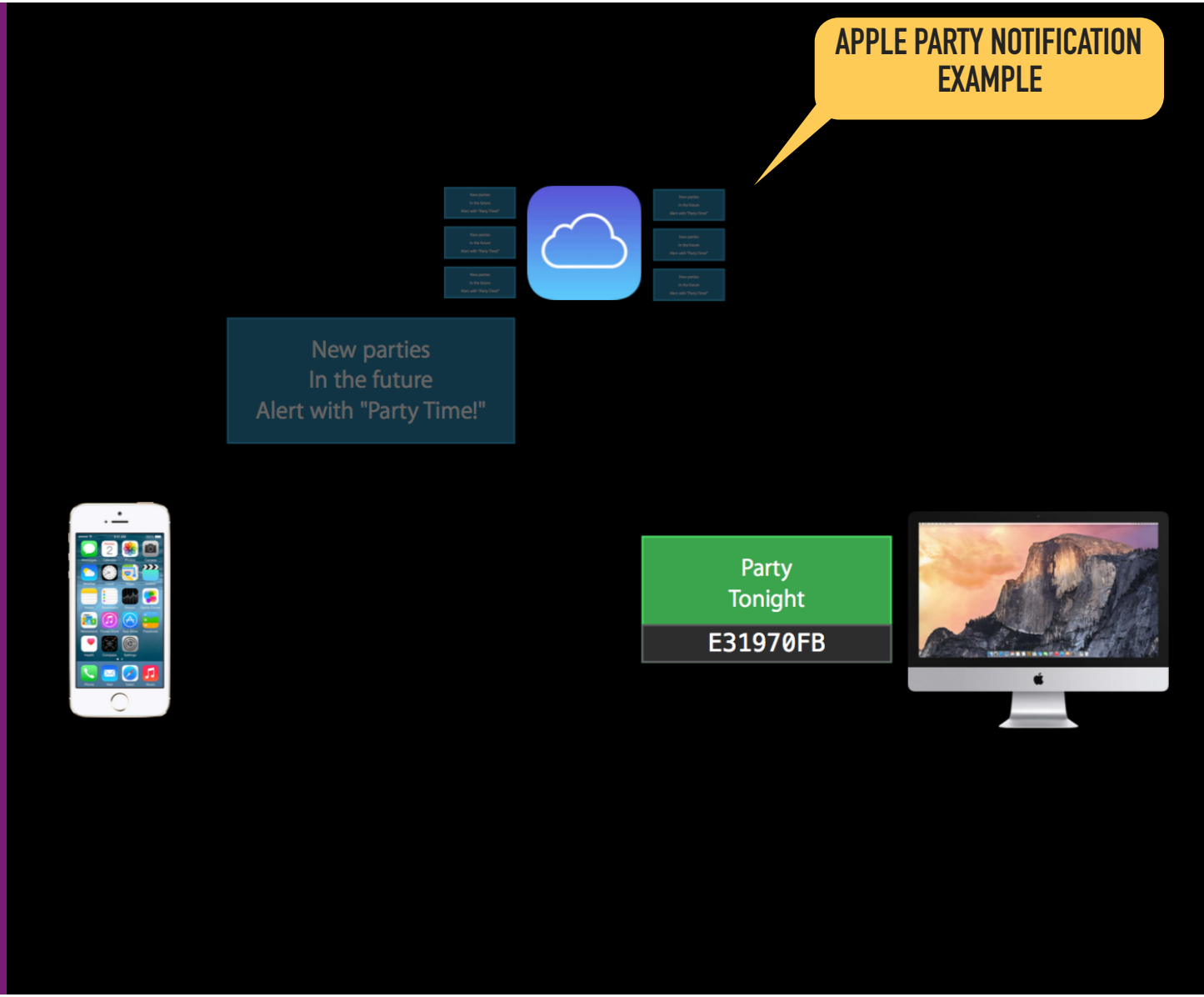
SUBSCRIPTIONS

- Subscription in action



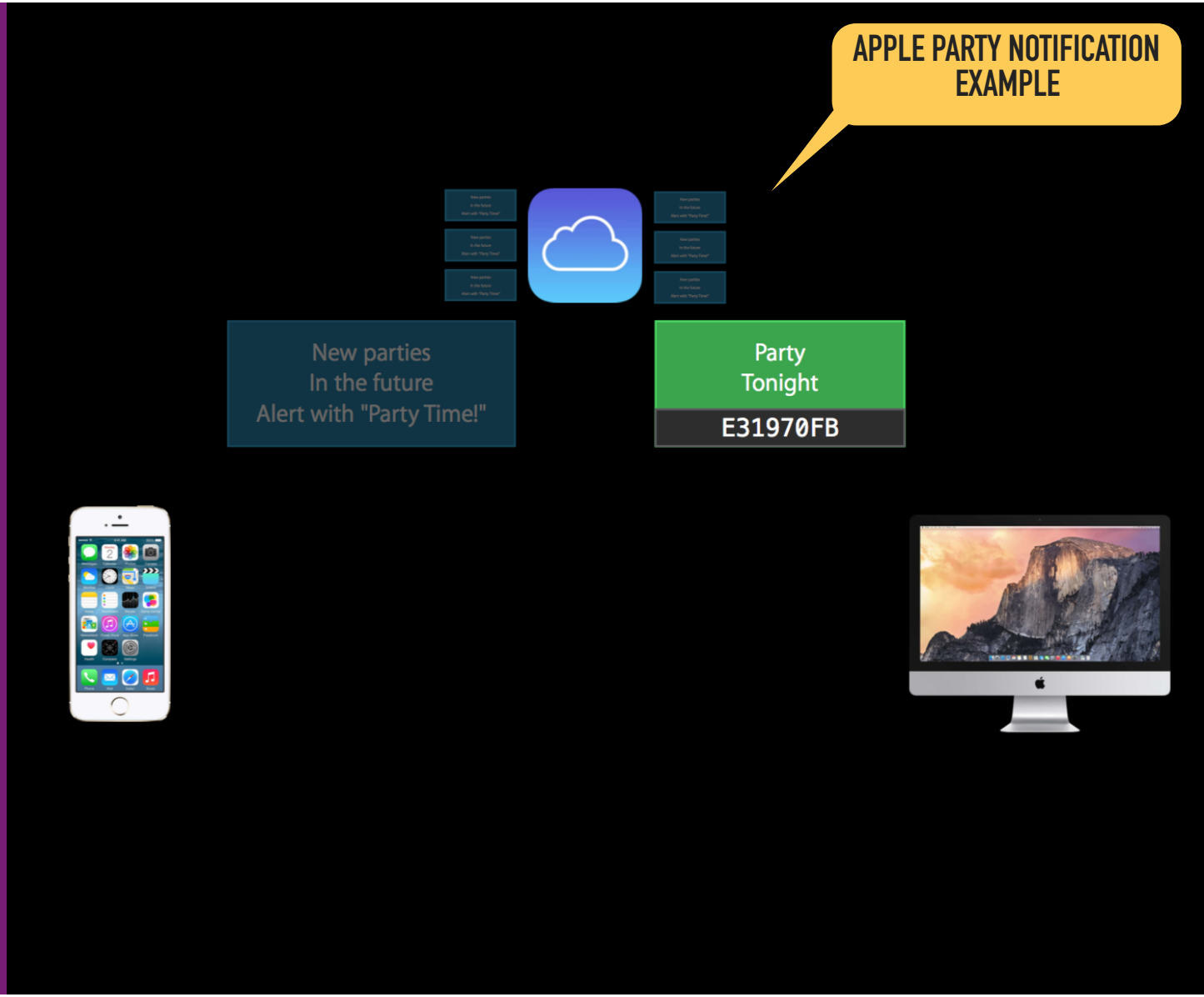
SUBSCRIPTIONS

- Subscription in action



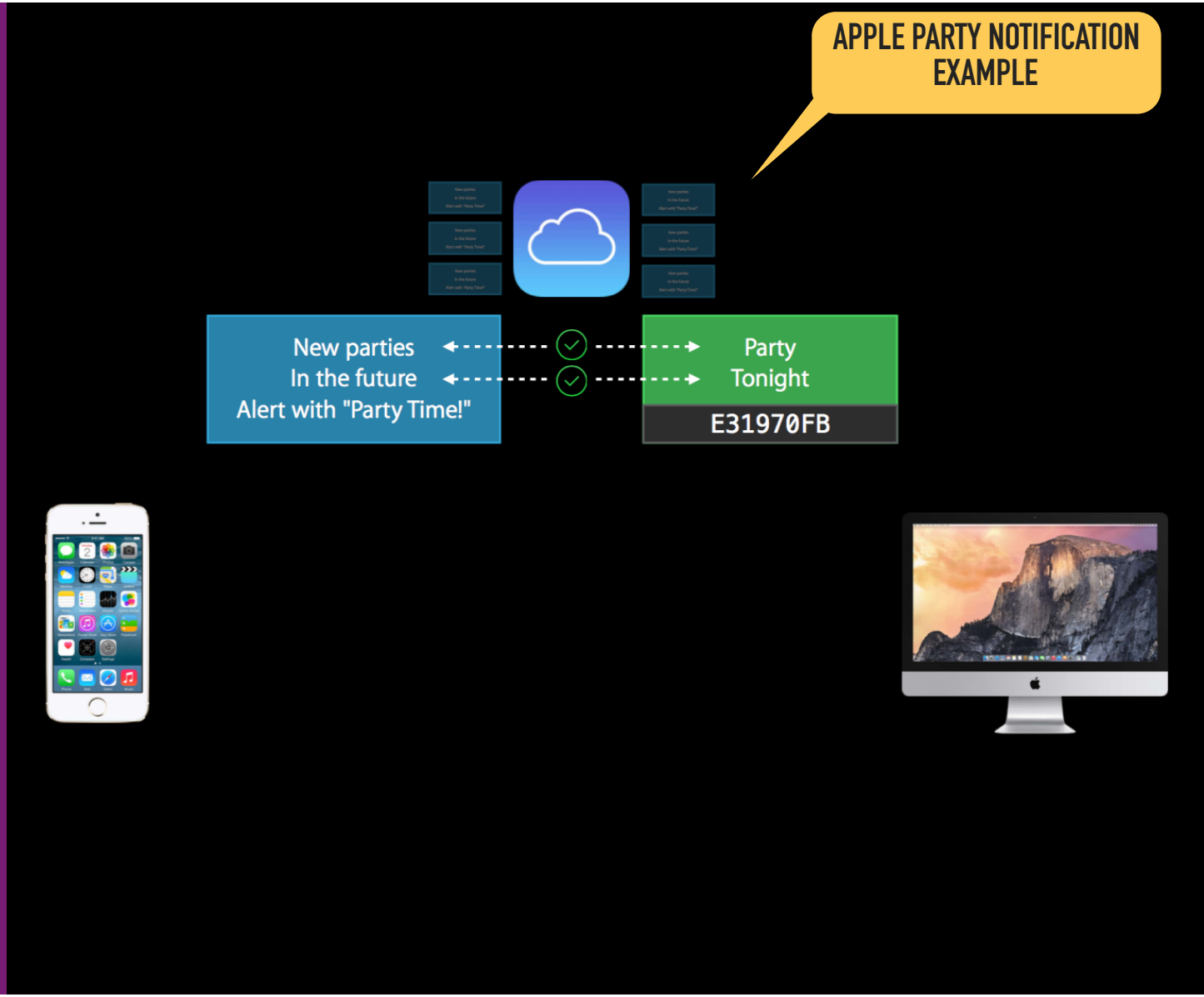
SUBSCRIPTIONS

- Subscription in action



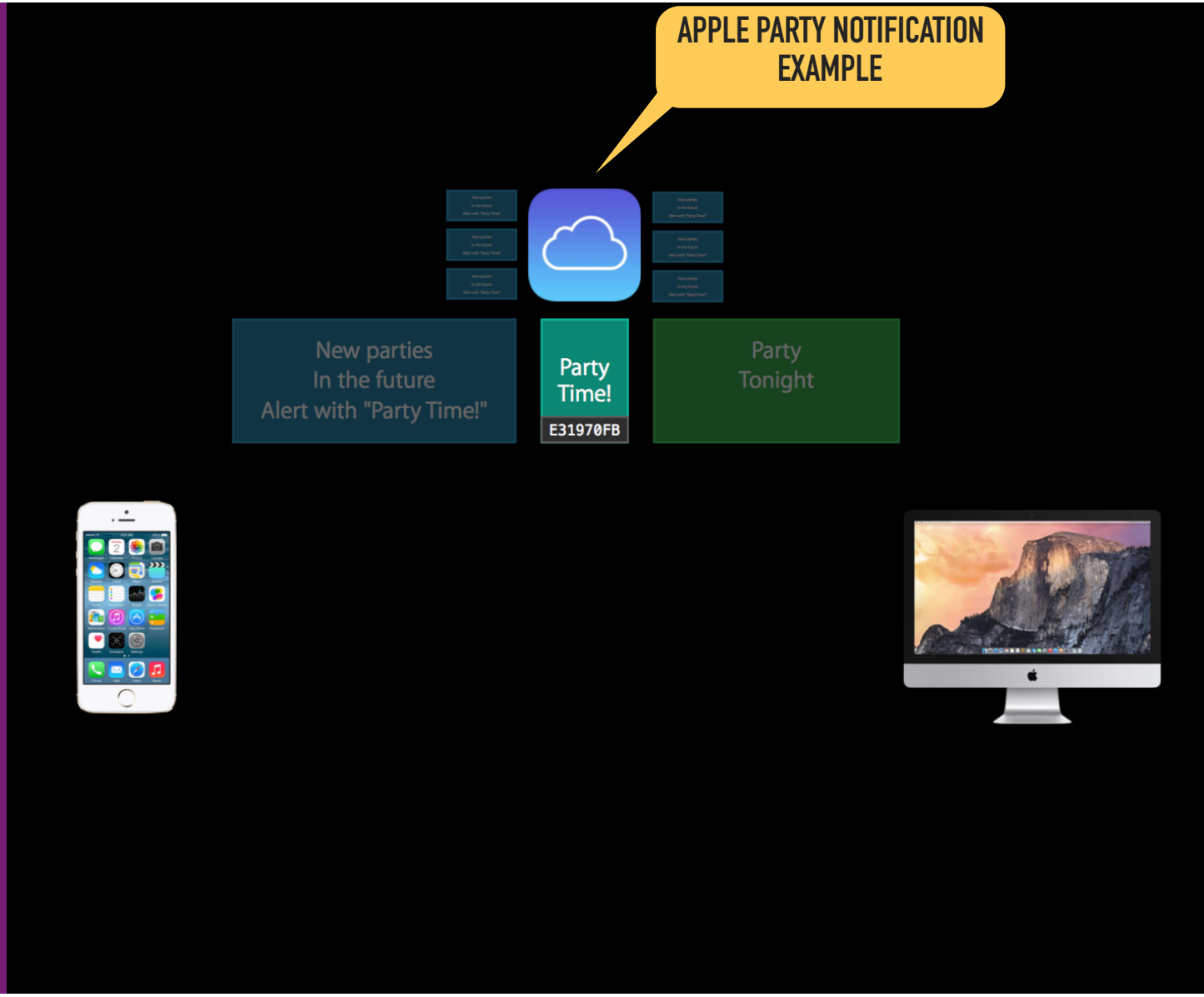
SUBSCRIPTIONS

- Subscription in action



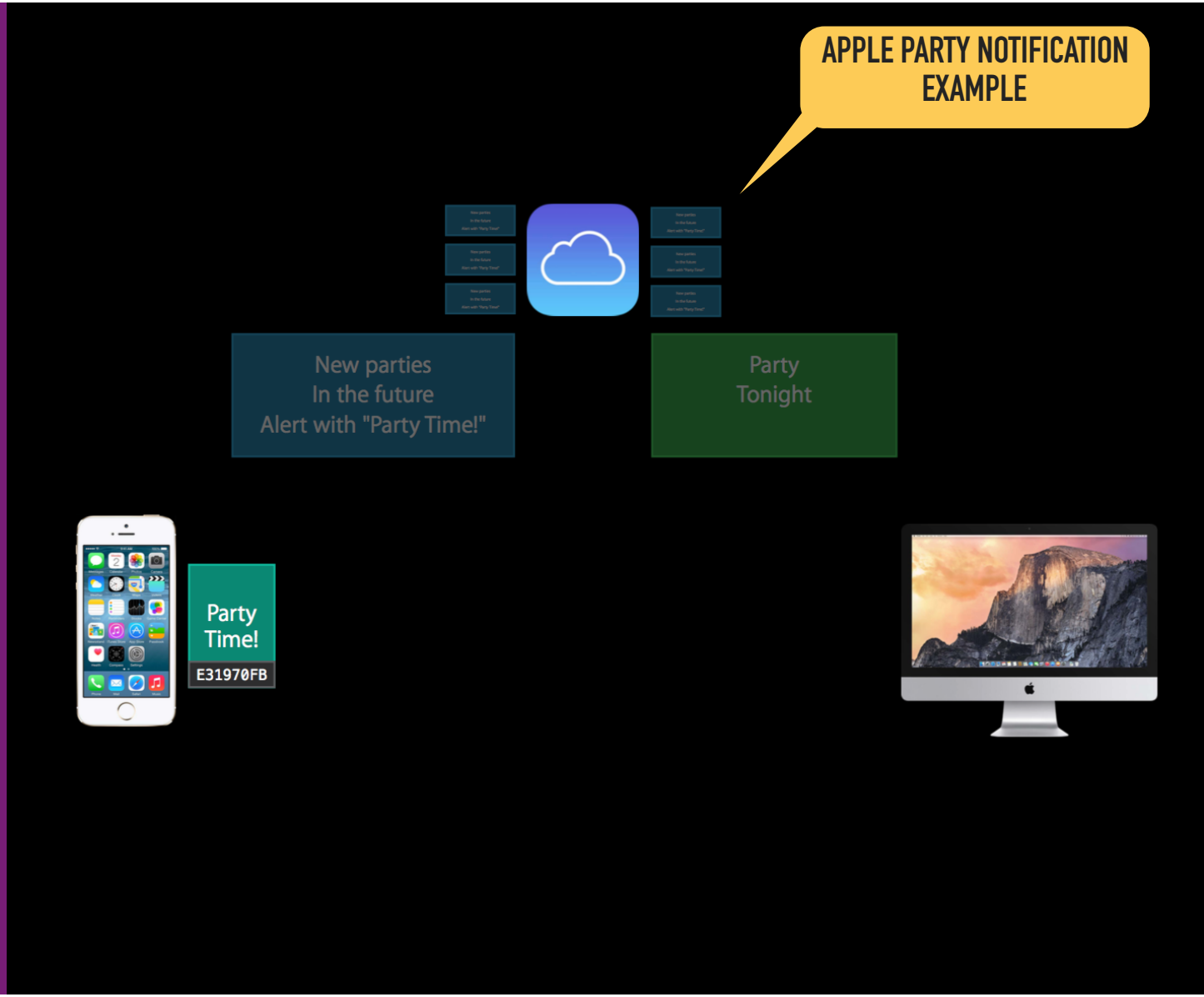
SUBSCRIPTIONS

- Subscription in action



SUBSCRIPTIONS

- Subscription in action



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.firesOnRecordCre
                                          CKQuerySubscription.Options.firesOnRecordUpd
                                          CKQuerySubscription.Options.firesOnRecordDel

  subscription.notificationInfo = notificationInfo

  // Save subscription
  CKContainer.default().publicCloudDatabase.save(subscription, completionHandler: ({re
    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

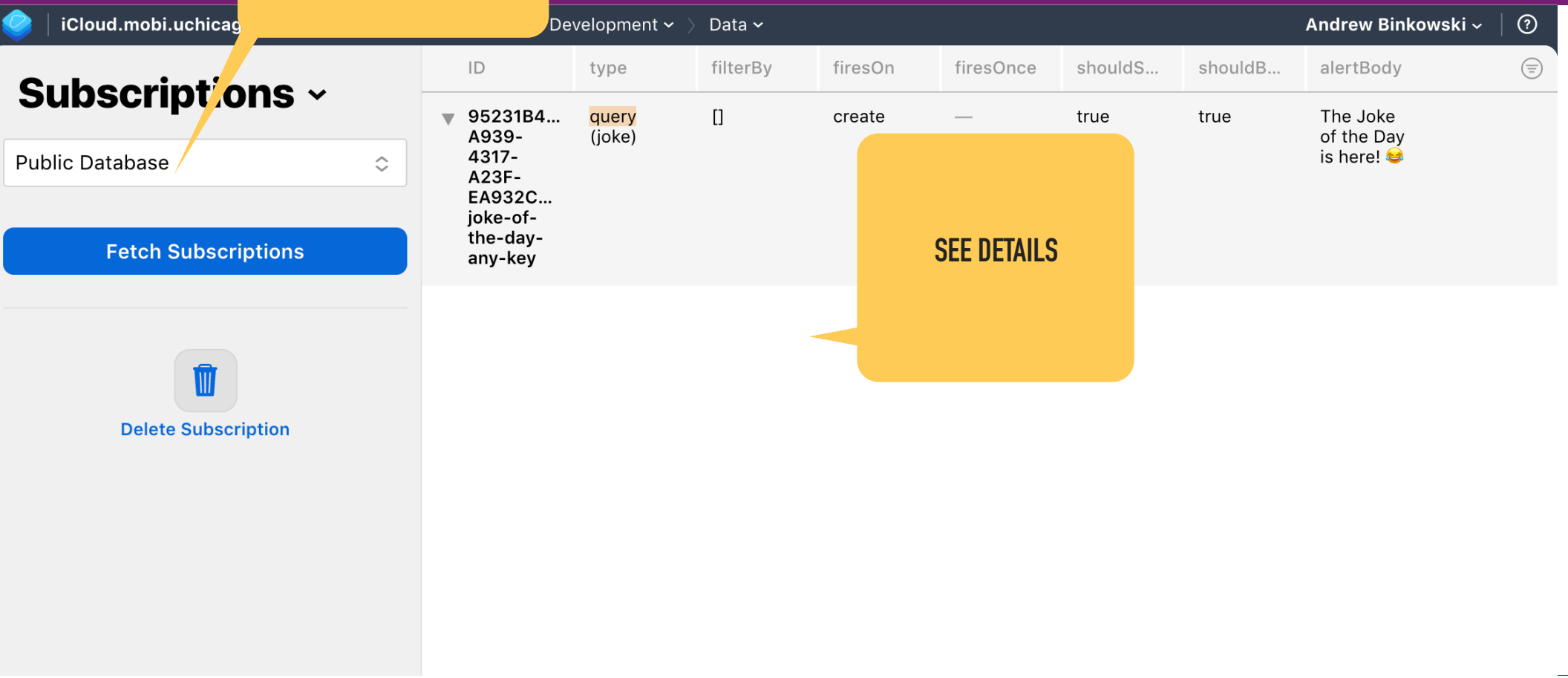
```
func registerJokeOfTheDaySubscriptions() {
  // Unique identifier for the subscription
  let uuid: UUID = UUID()
  let identifier = "\(uuid)-joke-of-the-day-any-key"

  // 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.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

- 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.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")
    }
  }))
}
```

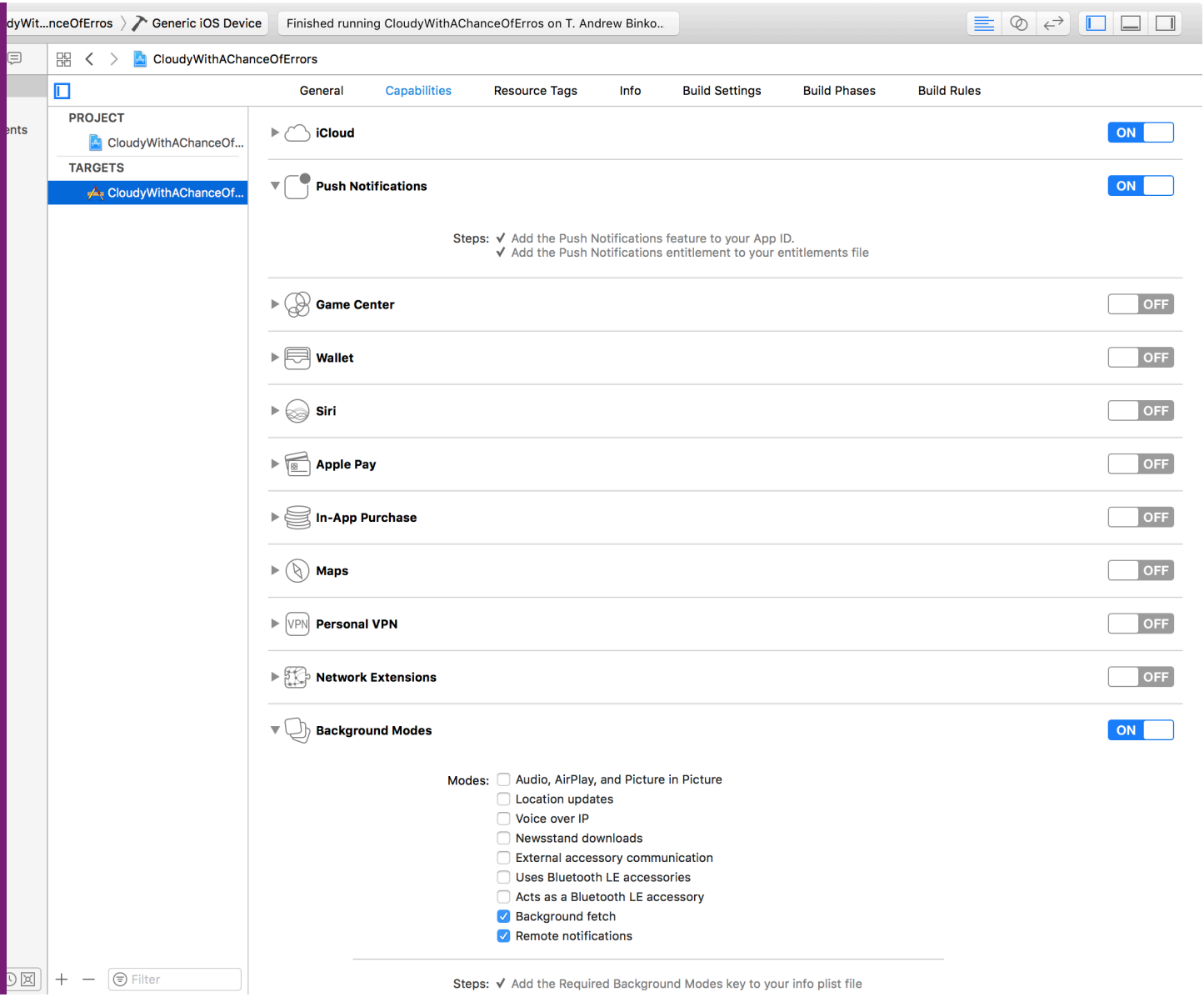
SUBSCRIPTIONS



SETUP YOUR APP TO HANDLE NOTIFICATIONS

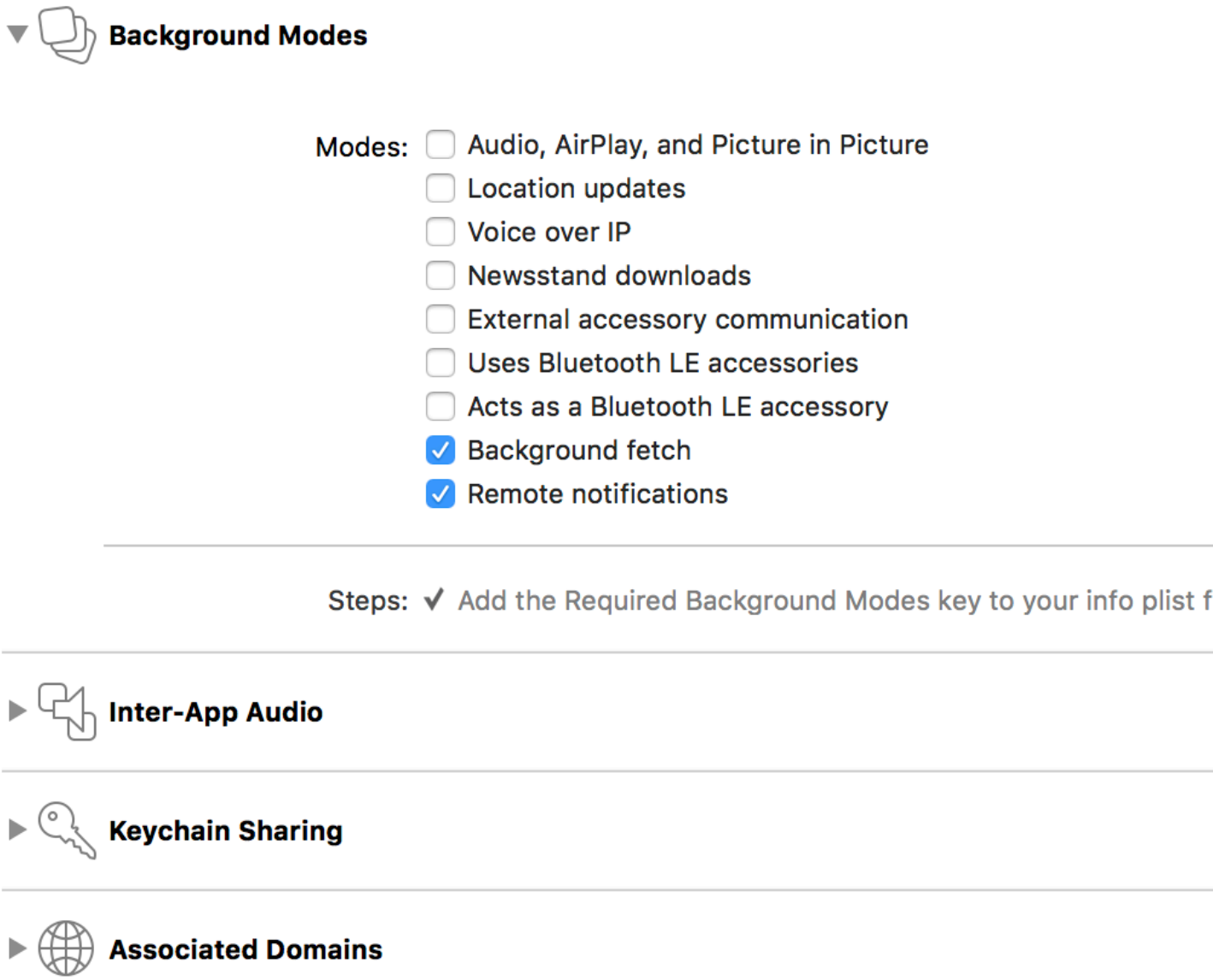
SUBSCRIPTIONS

- Required capabilities
  - Push notifications



SUBSCRIPTIONS

- Required capabilities
  - Remote notification
  - Background fetch



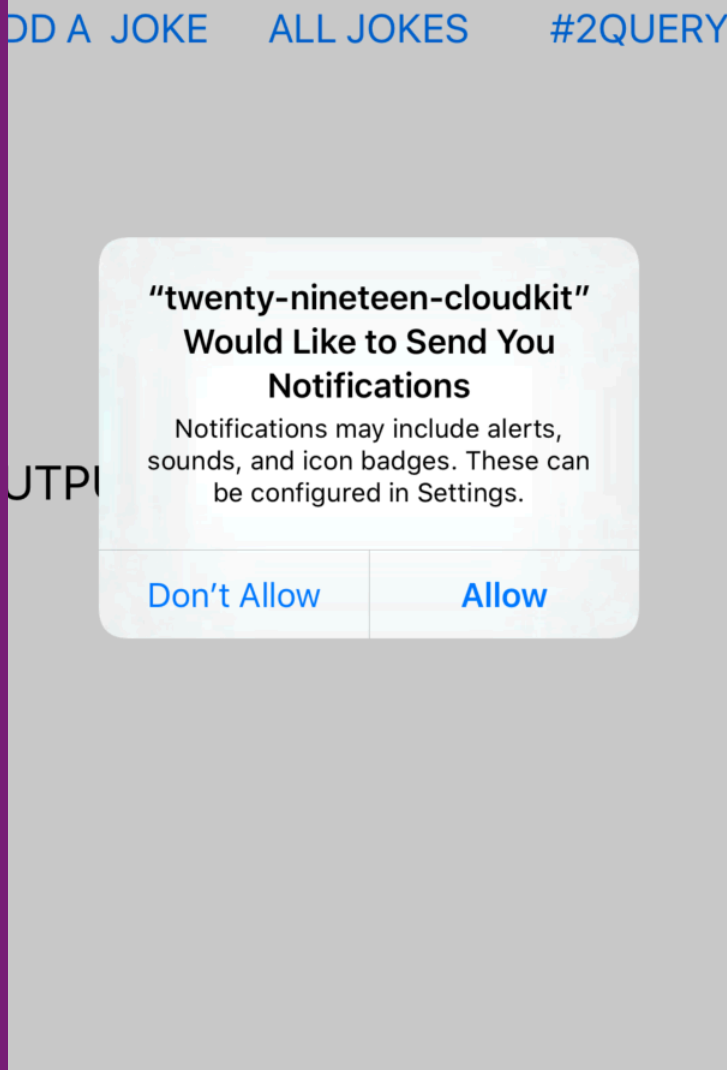
SUBSCRIPTIONS

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                    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
}
```

REGISTER SUBSCRIPTIONS

SUBSCRIPTIONS

- Register a subscription

```
func registerSubscriptionsWithIdentifier(_ identifier: String) {

  let uuid: UUID = UIDevice().identifierForVendor!
  let identifier = "\(uuid)-creation"

  // Create the notification that will be delivered
  let notificationInfo = CKNotificationInfo()
  notificationInfo.alertBody = "A new joke was added."
  notificationInfo.shouldBadge = true
  notificationInfo.shouldSendContentAvailable = true


  let subscription = CKQuerySubscription(recordType: "joke",
                                         predicate: NSPredicate(value: true),
                                         subscriptionID: identifier,
                                         options: [.firesOnRecordCreation])
  subscription.notificationInfo = notificationInfo
  CKContainer.default().publicCloudDatabase.save(subscription, completionHandler: ({returnRecord, error in
    if let err = error {
      print("subscription failed \(err.localizedDescription)")
    } else {
      print("subscription set up")
    }
  }))
}
```

HANDLE NOTIFICATIONS (THERE ARE 3 WAYS)

SUBSCRIPTIONS

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

```
func application(_ applicati
                 didFinishLa

  // Set the notification de
  UNUserNotificationCenter.c
    if let error = error {
      print("Error: \(error.
    } else {
      application.registerFo
    }
  }
  UNUserNotificationCenter.c

  // Register subscriptions
  CloudKitManager.sharedInst
  CloudKitManager.sharedInst
  //configureUserNotificatio

  if let notification = laun
    let aps = notification['
    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(_ applicati
                 didFinishLa

  // Set the notification de
  UNUserNotificationCenter.c
    if let error = error {
      print("Error: \(error.
    } else {
      application.registerFo
    }
  }
  UNUserNotificationCenter.c

  // Register subscriptions
  CloudKitManager.sharedInst
  CloudKitManager.sharedInst
  //configureUserNotificatio

  if let notification = laun
    let aps = notification['
    print("APS: \(aps)")
  }
  return true
}
```

SUBSCRIPTIONS

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

FRESH LAUNCH
(RARELY HAPPENS)

```
func application(_ applicati
                 didFinishLa

                 notification de
                 icationCenter.
                 ror = error {
                 Error: \(error.

                 tion.registerFo

  }
  UNUserNotificationCenter.

  // Register subscriptions
  CloudKitManager.sharedInst
  CloudKitManager.sharedInst
  //configureUserNotificatio

  if let notification = laun
    let aps = notification['
    print("APS: \(aps)")
  }
  return true
}
```

SUBSCRIPTIONS

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

ANY OTHER STATE

```
func application(_ applicati
                 didFinishLa

  // Set the notification de
  UNUserNotificationCenter.c
    if let error = error {
      print("Error: \(error.
    } else {
      lication.registerFo

      otificationCenter.c

      ster subscriptions
  CloudKitManager.sharedInst
  CloudKitManager.sharedInst
  //configureUserNotificatio

  if let notification = laun
    let aps = notification['
    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 NOTIFCATIONS

DEBUGGING NOTIFICATION HANDLING

SUBSCRIPTIONS

Technical Q&A QA1917

Debugging issues with CloudKit subscriptions

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

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

Cases that aren't expected to trigger CloudKit notifications

When working with CloudKit subscriptions, be aware that:

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

HTTPS://DEVELOPER.APPLE.COM/LIBRARY/CONTENT/QA/QA1917/_INDEX.HTML

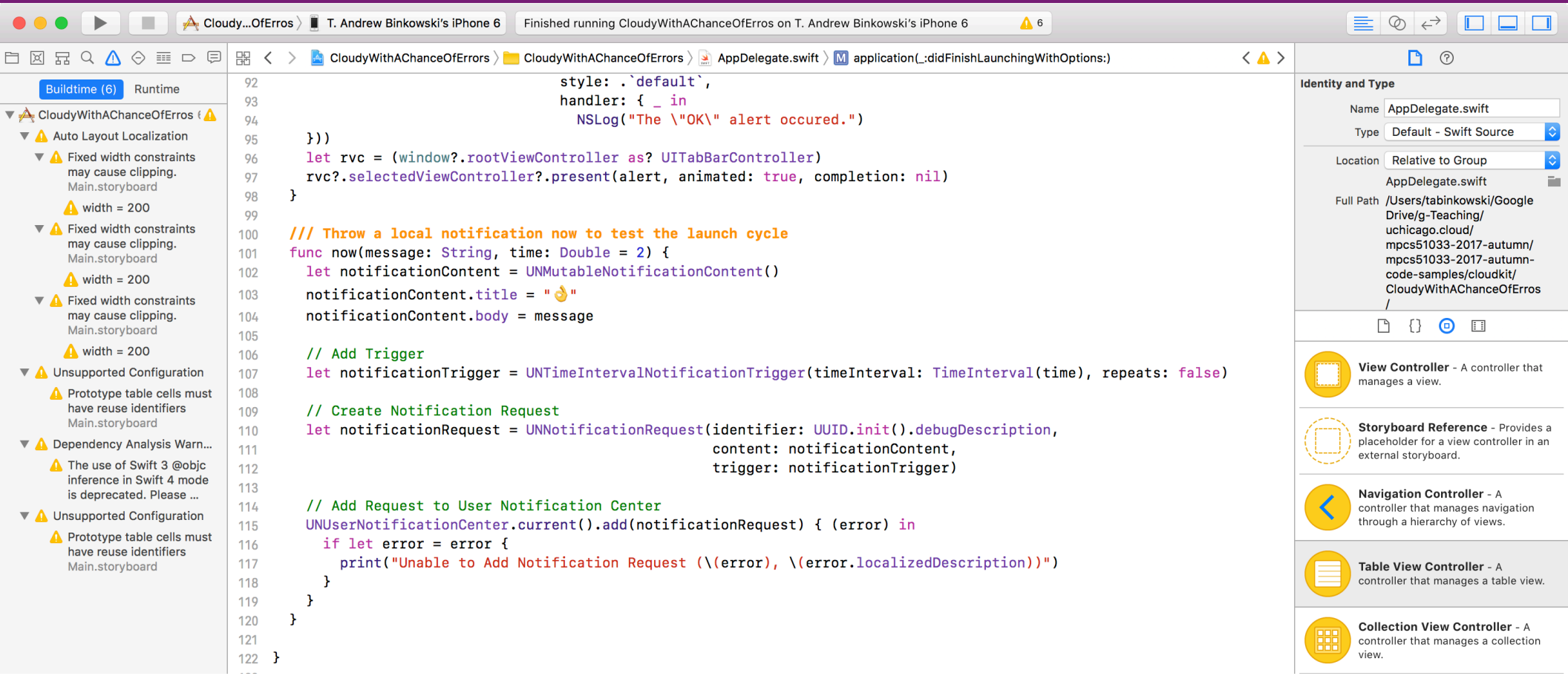
SUBSCRIPTIONS

```
/// 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 occured.")
  }))
  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



WEB SERVICES

PREVIEW

WEB SERVICE API

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

JS

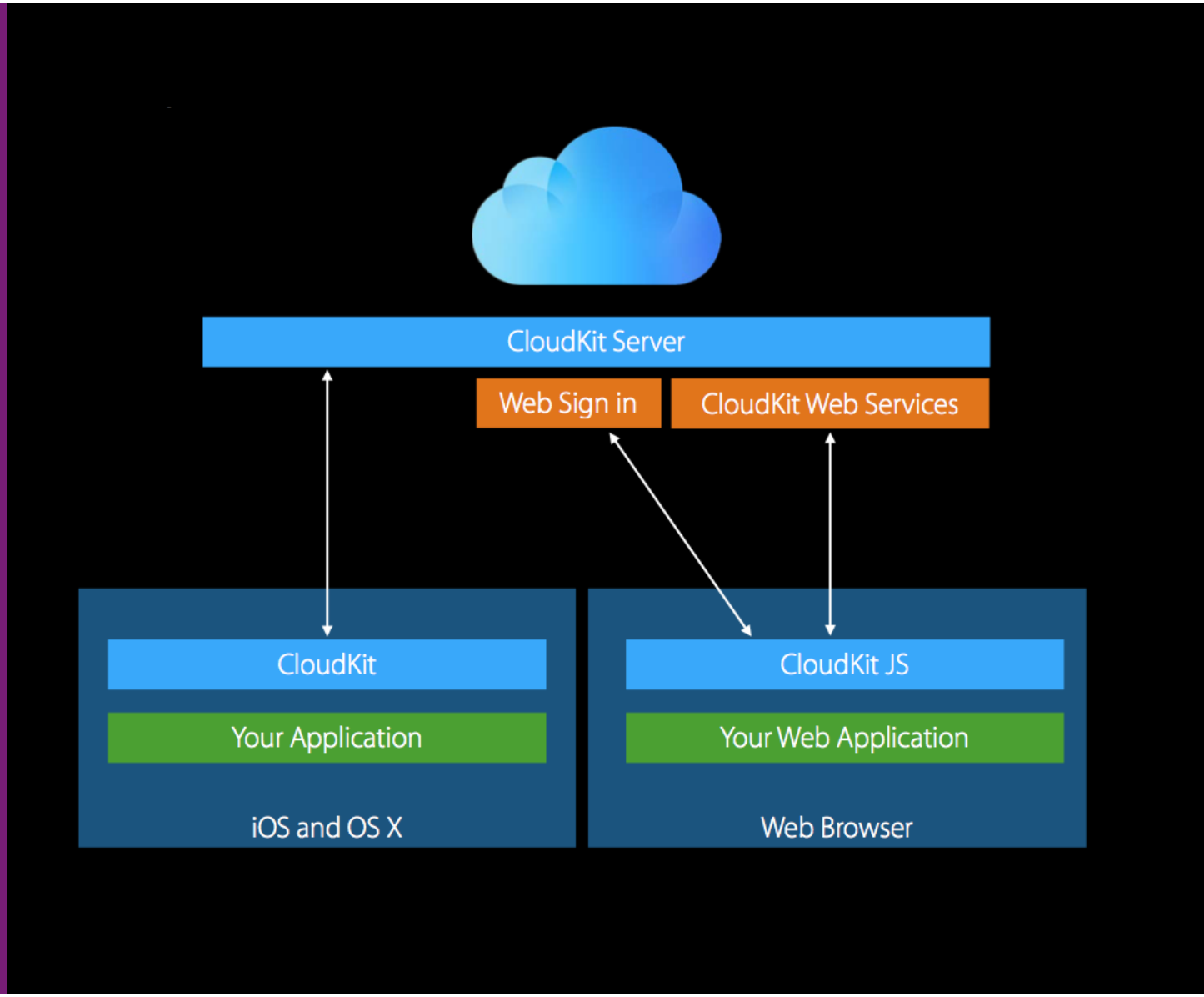
```
<SCRIPT SRC="HTTPS://CDN.APPLE-CLOUDKIT.COM/CK/1/CLOUDKIT.JS" />
```

WEB SERVICE API

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

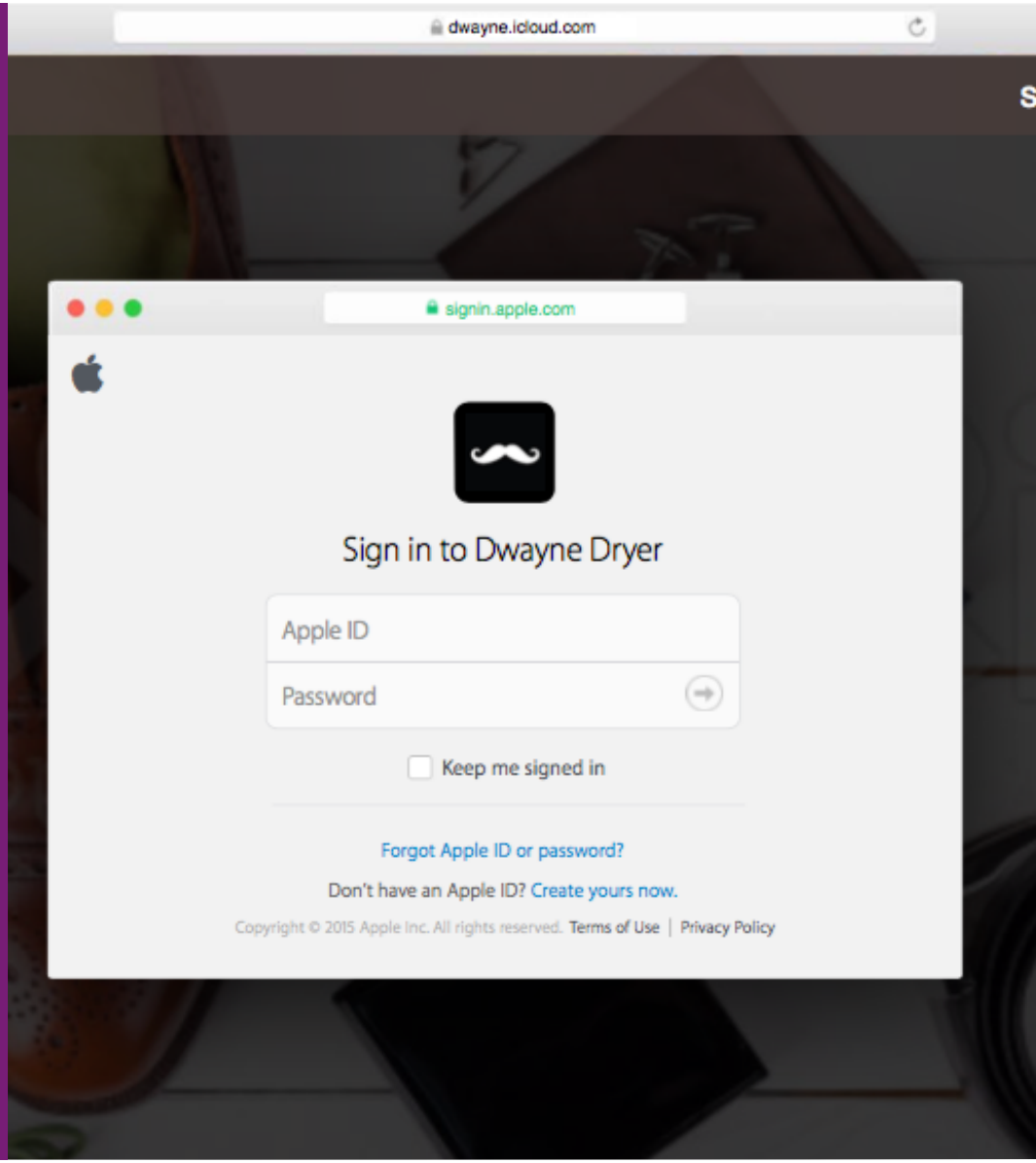
WEB SERVICES

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

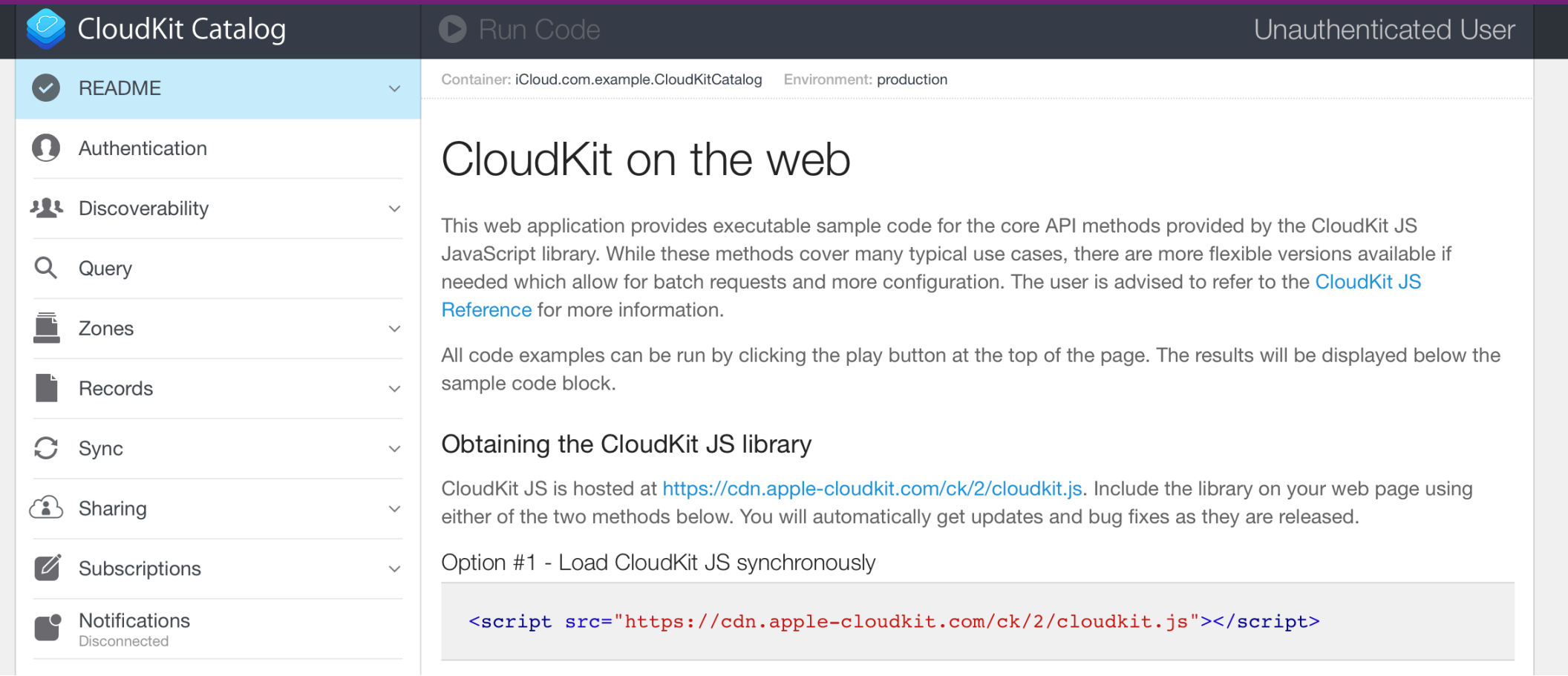


WEB SERVICE API

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



WEB SERVICE API



WHEN TO USE CLOUDKIT

WHEN TO USE CLOUDKIT

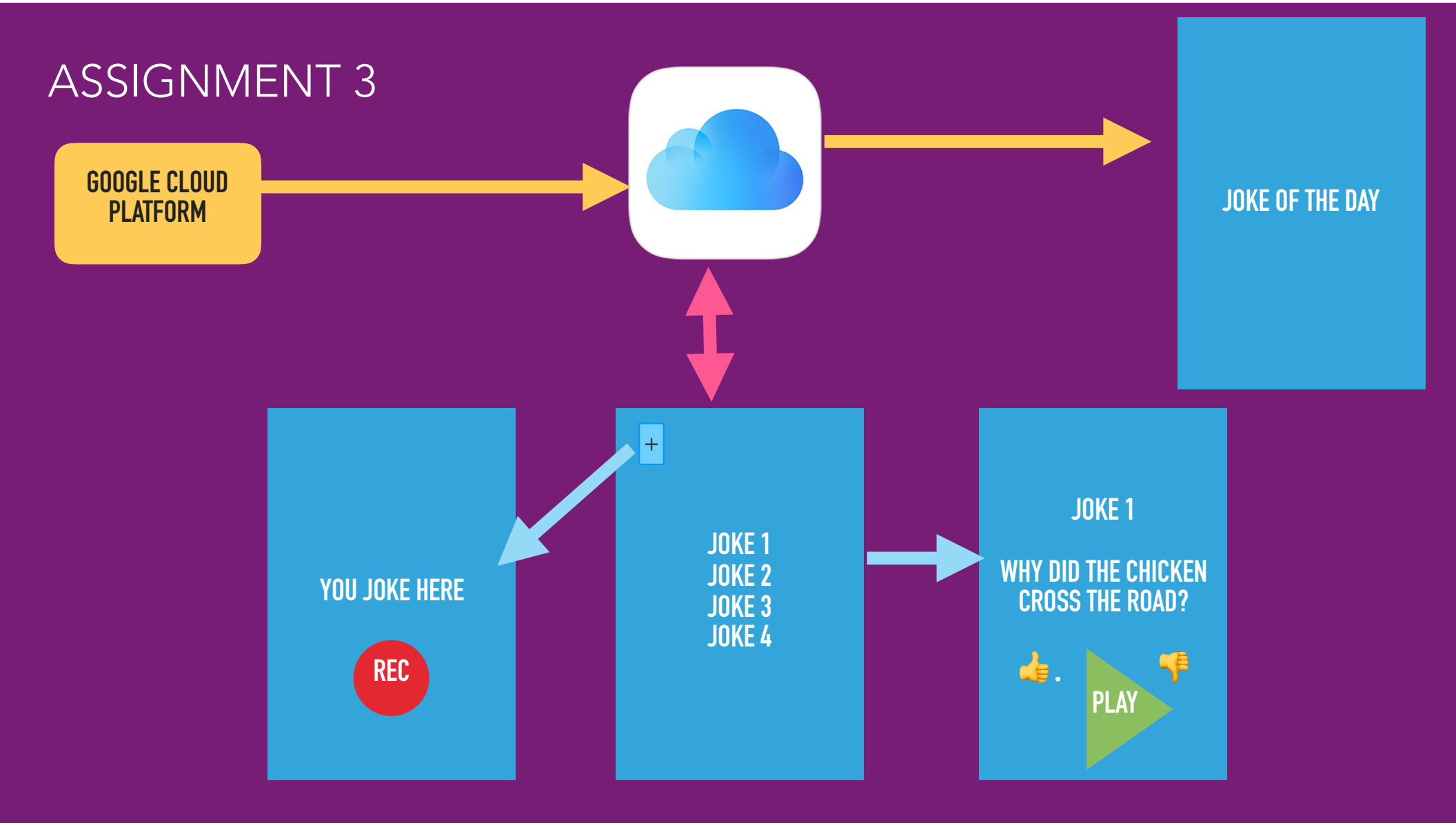
- You're developing exclusively for Apple platforms
- You use it as a partial solution

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