THE UNIVERSITY OF CHICAGO

MPCS 51033 • SPRING 2017 • SESSION 6

BACKENDS FOR MOBILE APPLICATIONS

© 2016 T.A. Binkowski

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

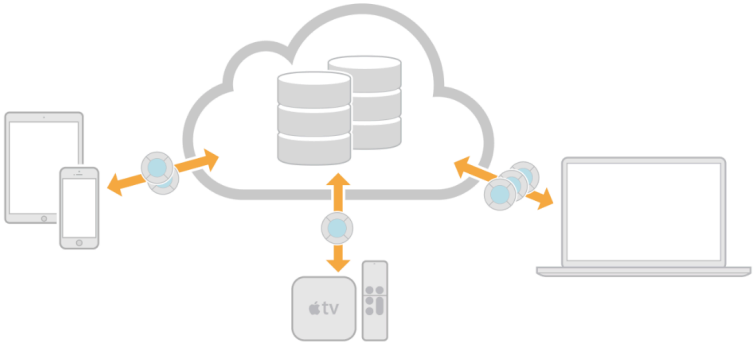
CLOUDKIT ☁️

CLOUDKIT

- CloudKit QuickStart

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

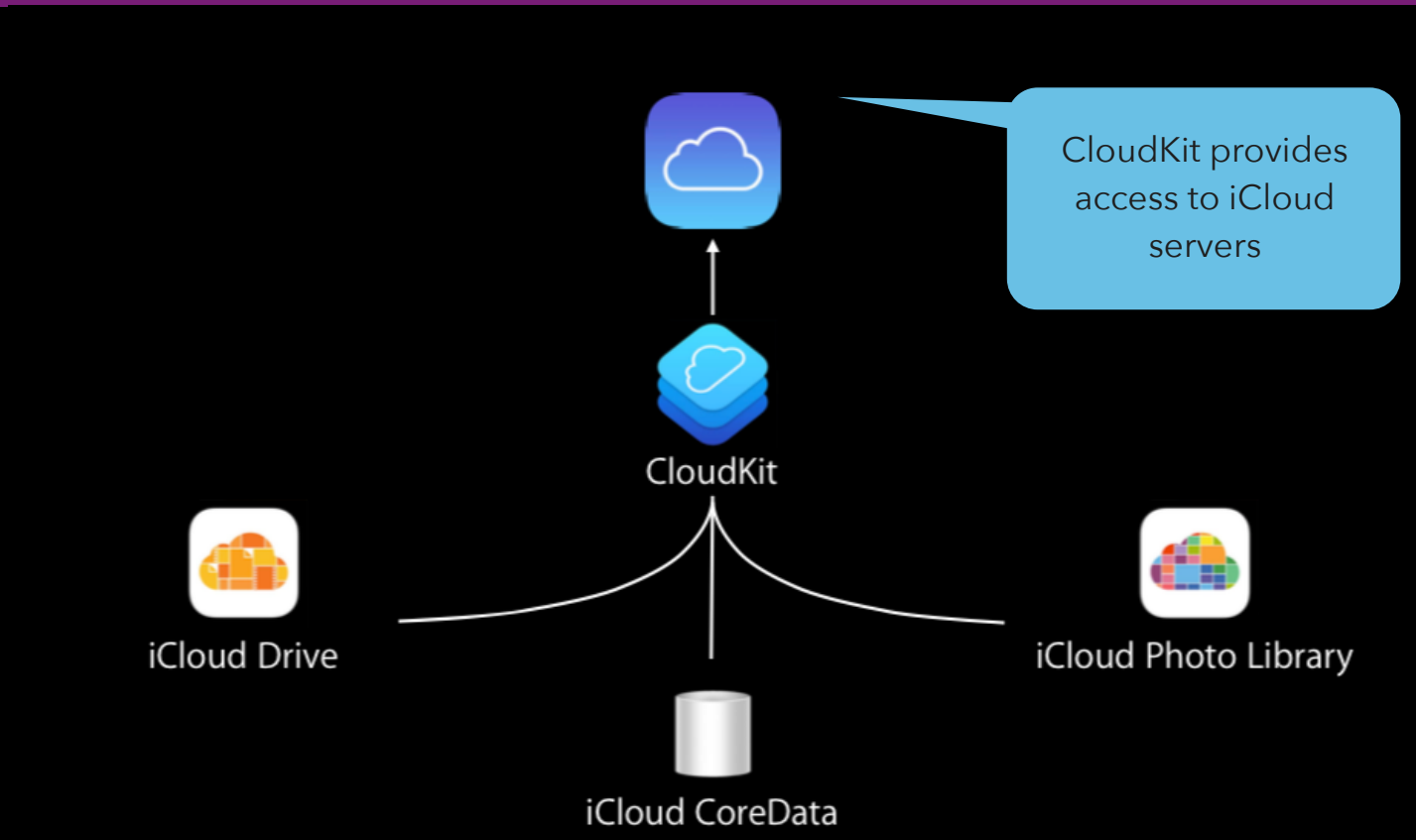
See Glossary for the definition of database terms used in this book.

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.

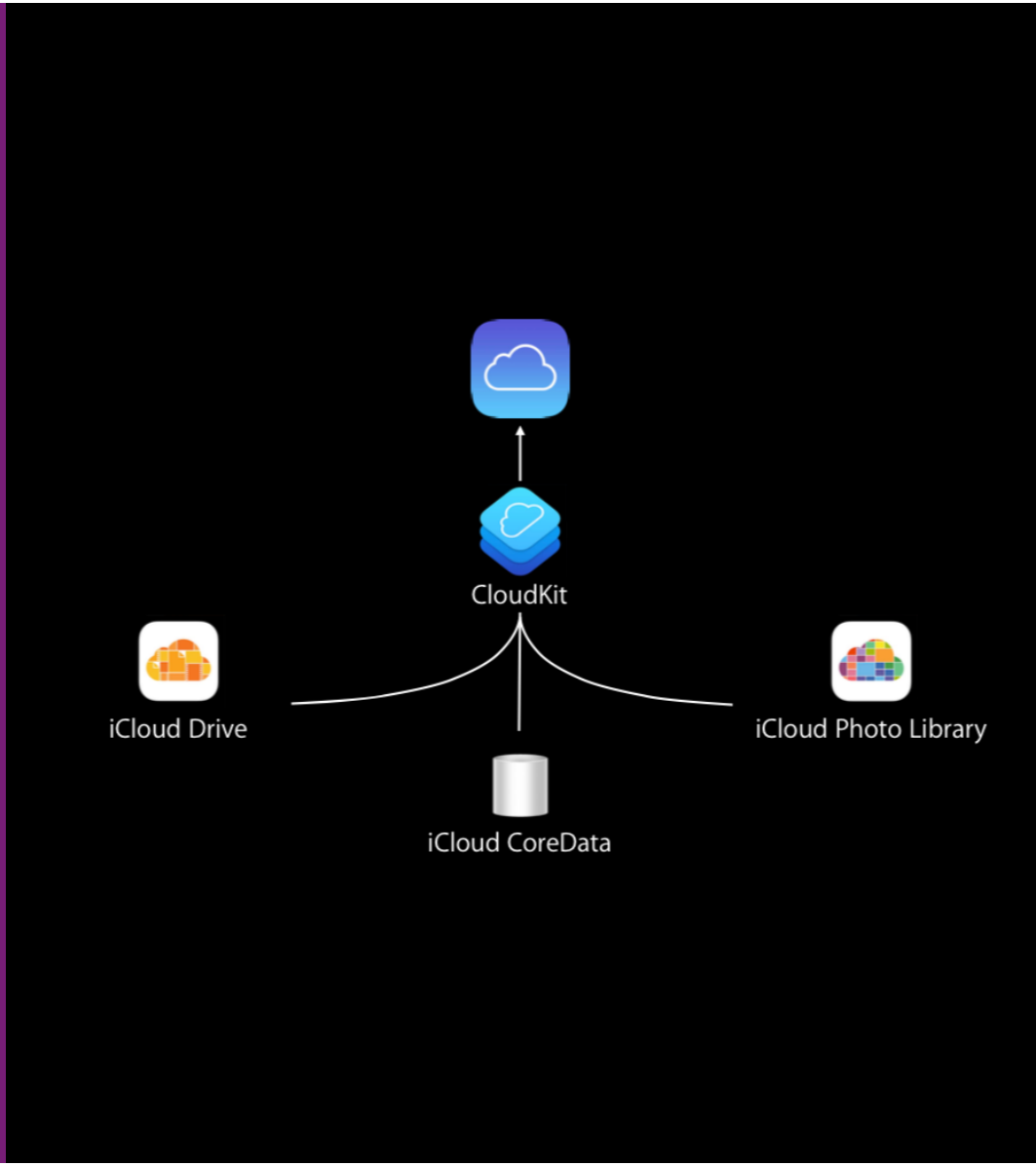
CLOUDKIT


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

CLOUDKIT

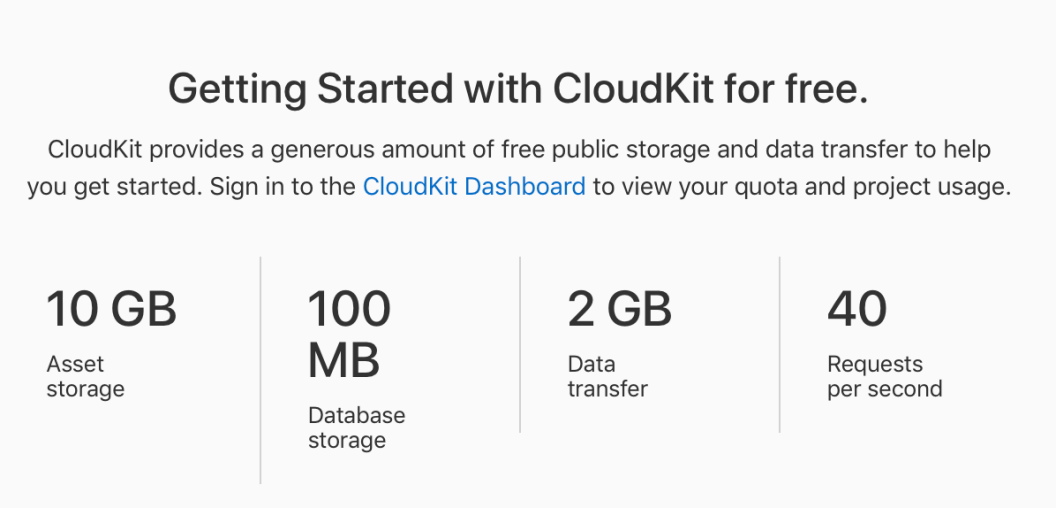
- Access to iCloud servers
- Supported on macOS, tvOS, iOS, watchOS and web (JS API)

#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

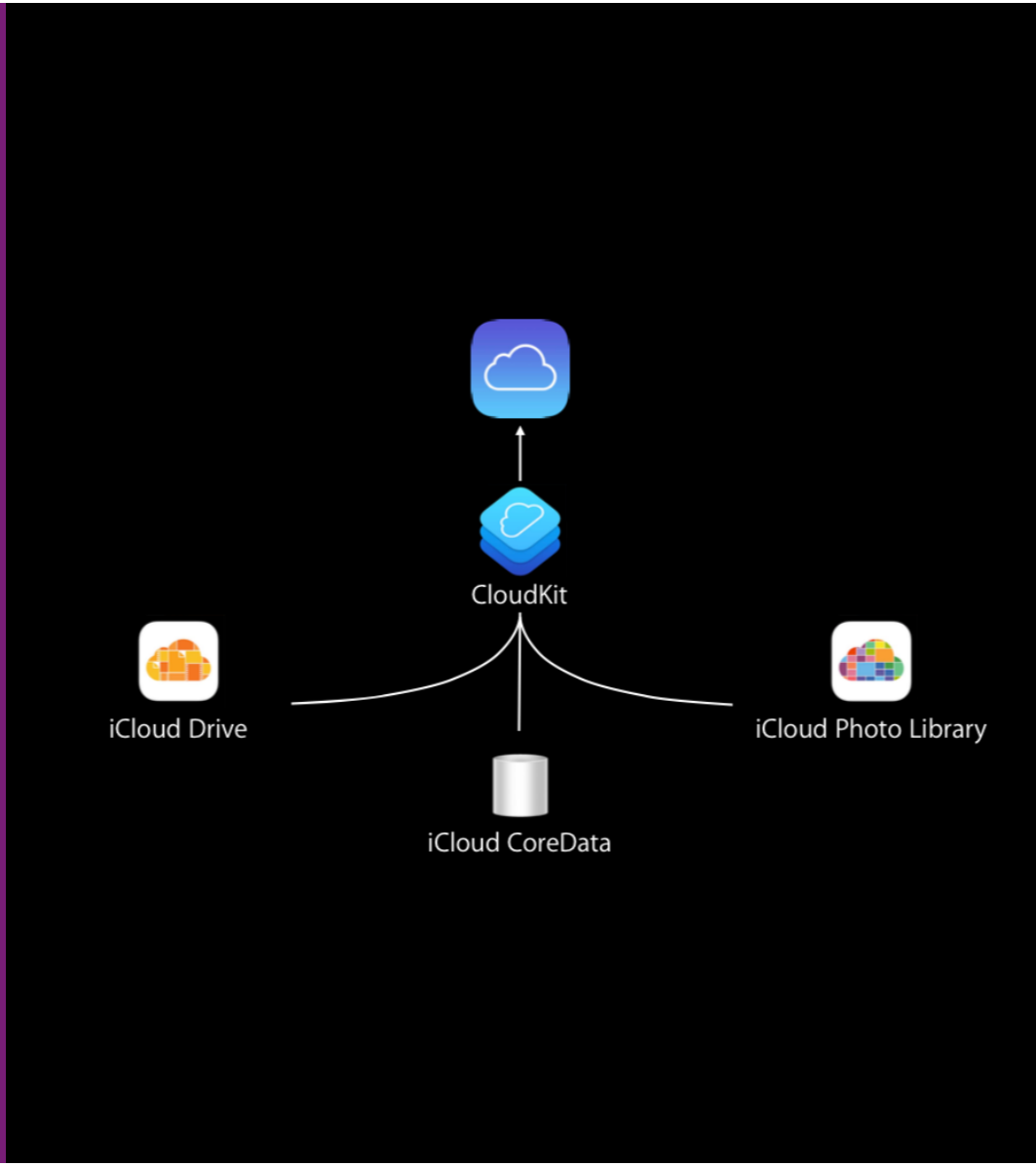
1 PB	10 TB	200 TB	400
Asset storage	Database storage	Data transfer	Requests per sec.
Based on: 100 MB per user	Based on: 1 MB per user	Based on: 20 MB per user	Based on: 4 per 100,000 users

$0

Total Cost

CLOUDKIT

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

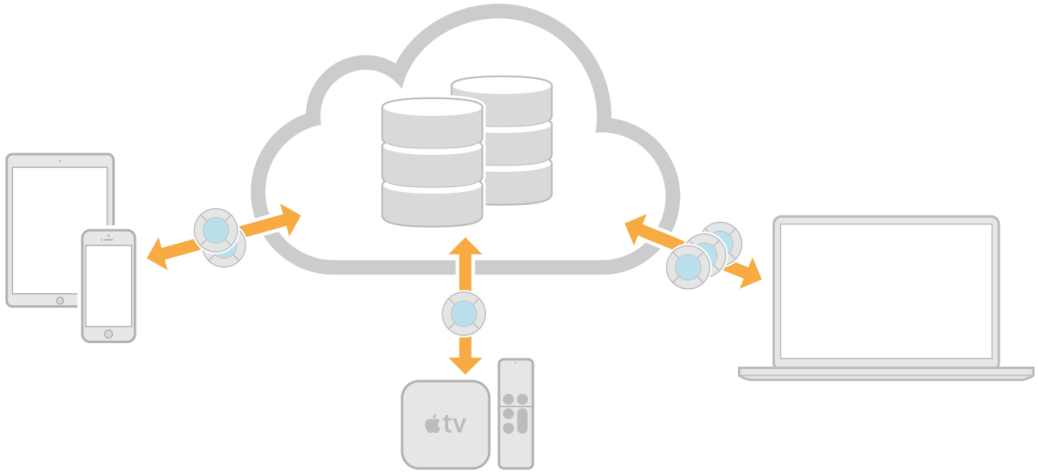


CLOUDKIT

- Databases
  - Public
  - Private
  - Shared

CLOUDKIT

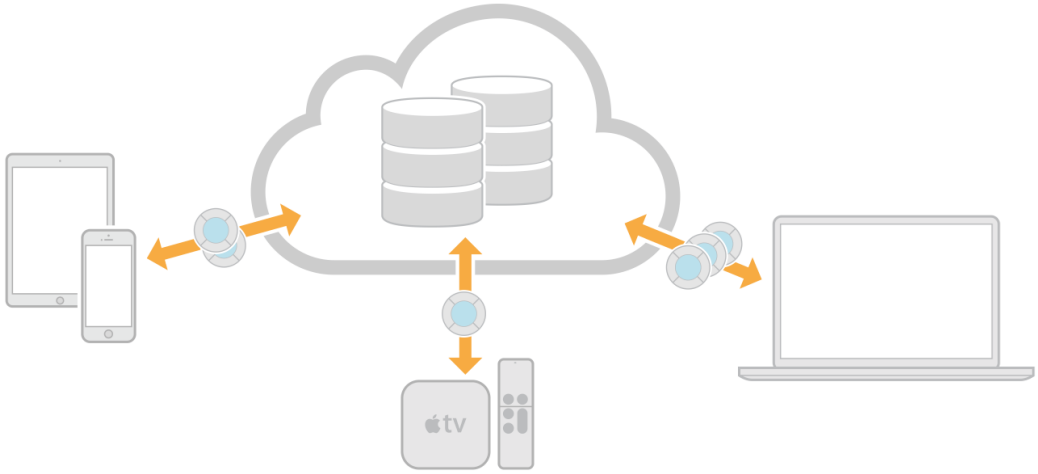
- Structured and bulk data
  - Data with types
  - Blobs



CLOUDKIT

CORE OBJECT

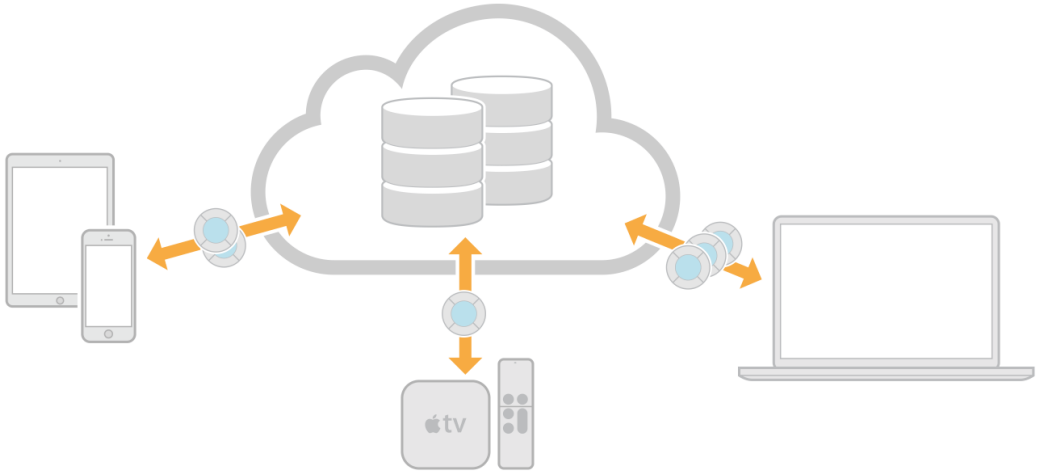
- Containers
- Databases
- Records
- Record Zones
- Record Identifiers
- Shares
- References
- Assets



CLOUDKIT

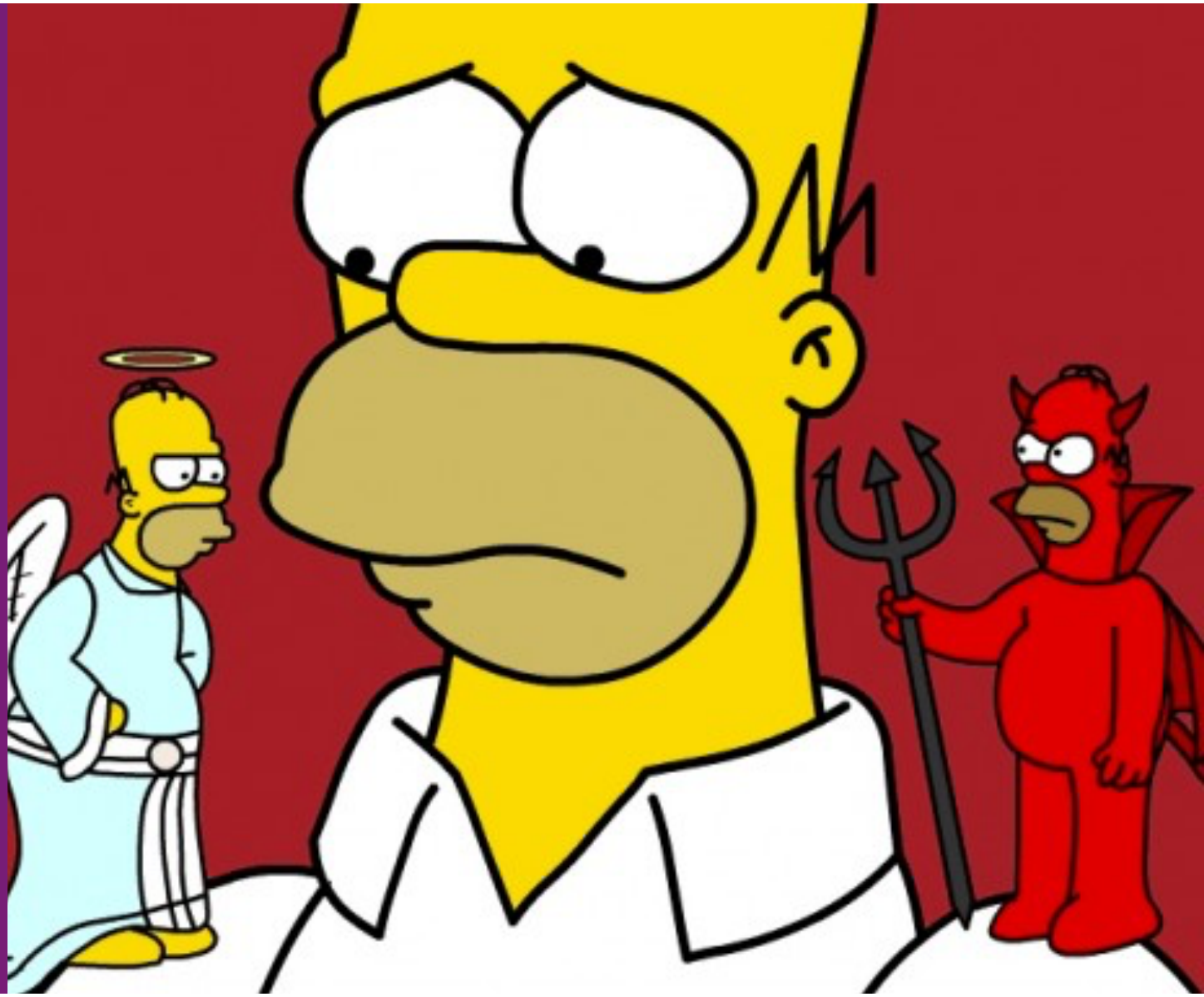
- Transport layer, not local persistence
- You still need to figure out what to do with the data once you have it on your device
  - Cache
  - Mirror

#2 consideration on choosing over other services



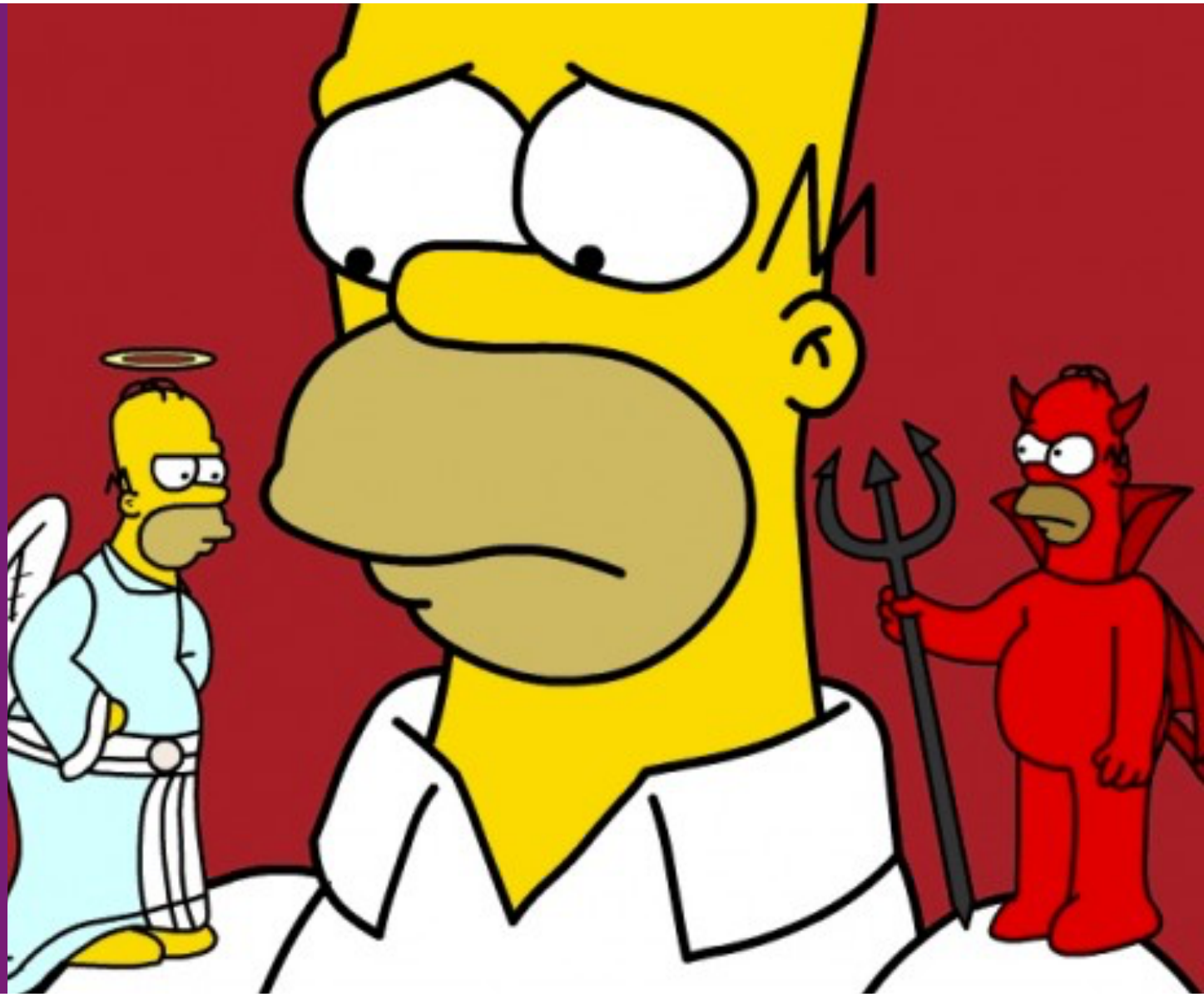
CLOUDKIT

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



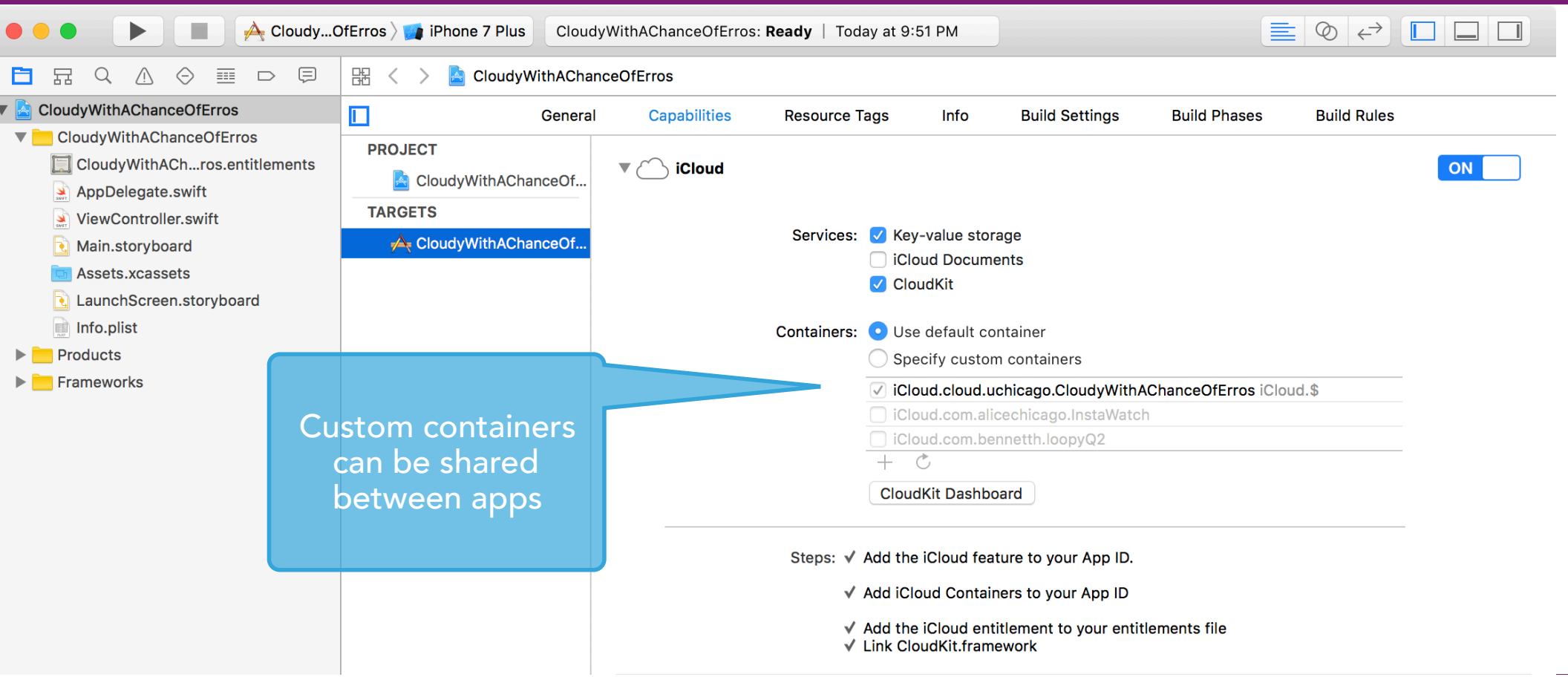
CLOUDKIT

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

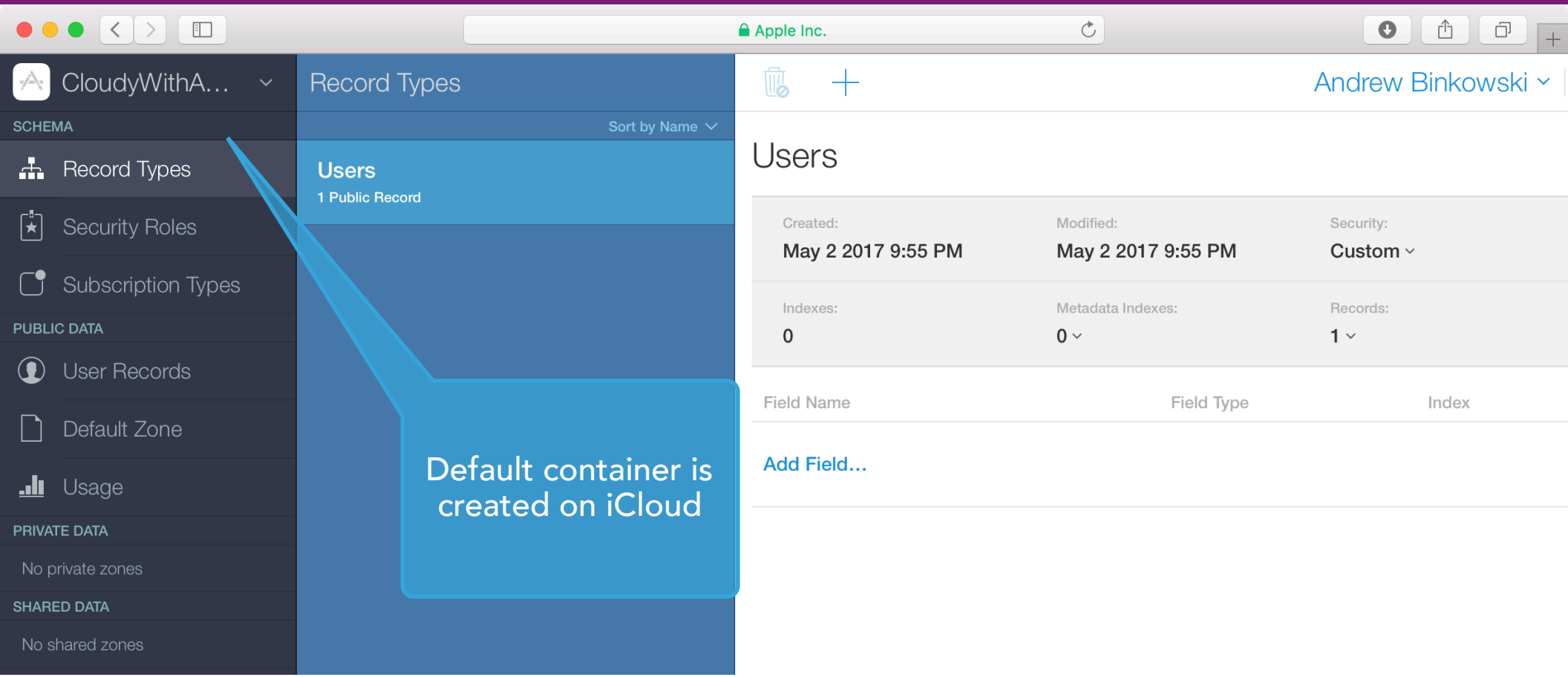
We will discuss how these may be less of an issue than they seem

ENABLING CLOUDKIT IN YOUR APPLICATION

ENABLING CLOUDKIT IN YOUR APPLICATION

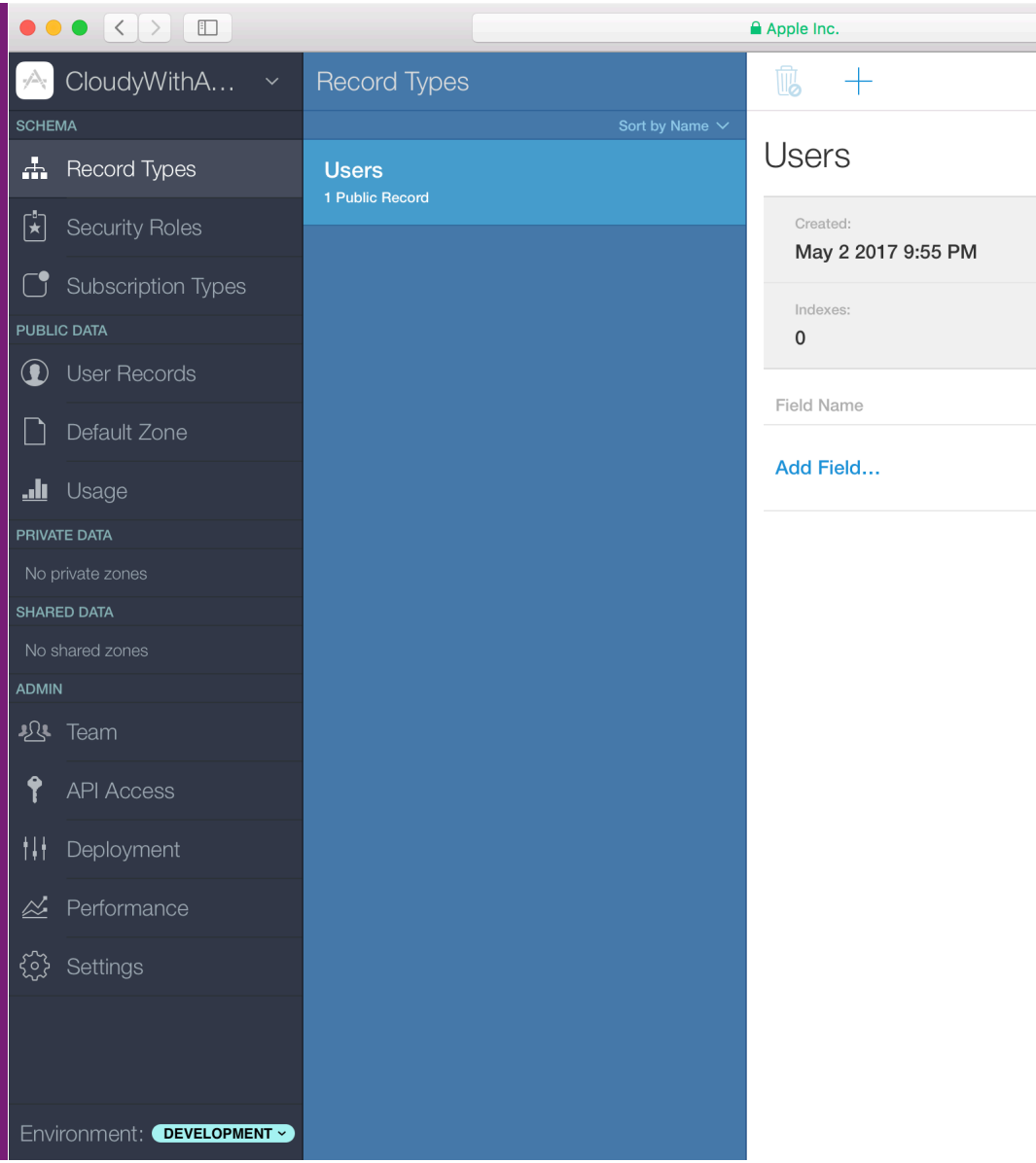


ENABLING CLOUDKIT IN YOUR APPLICATION

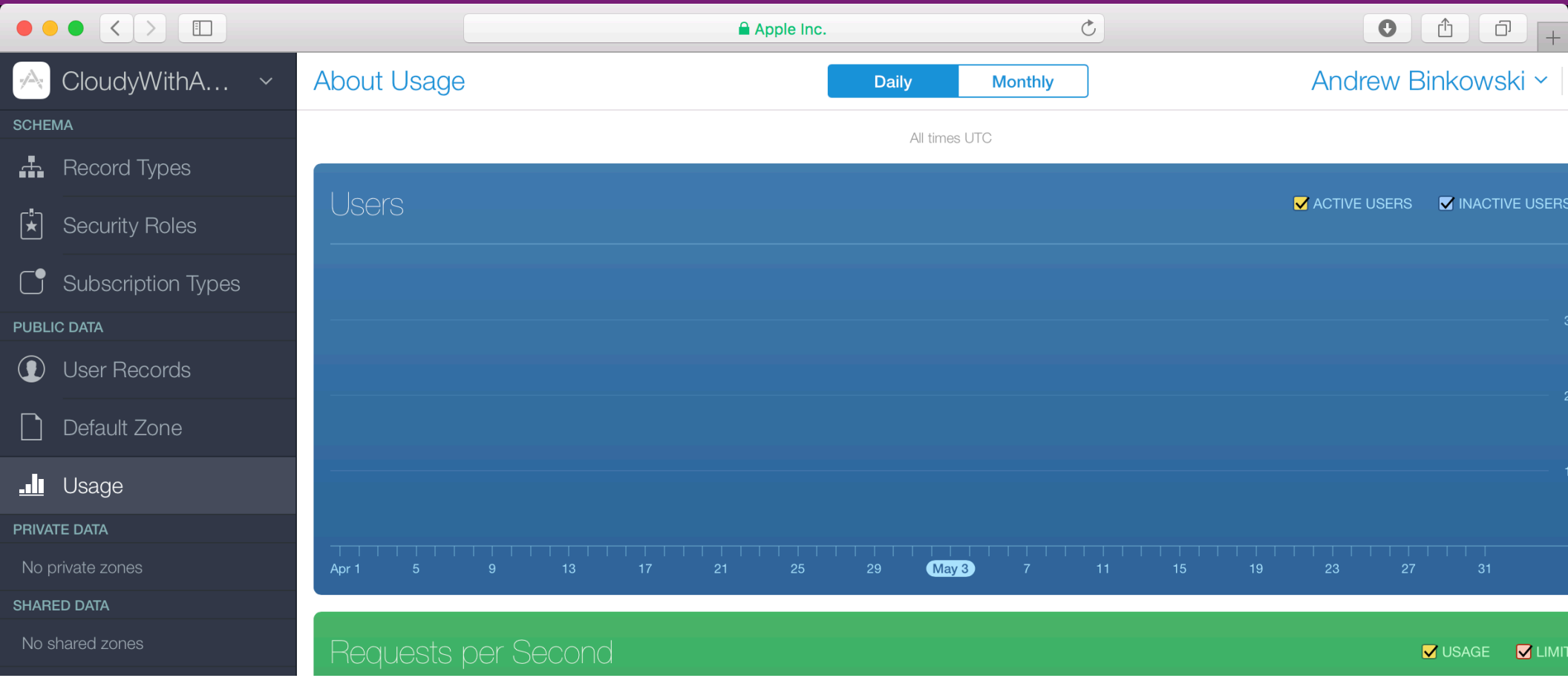


ENABLING CLOUDKIT IN YOUR APPLICATION

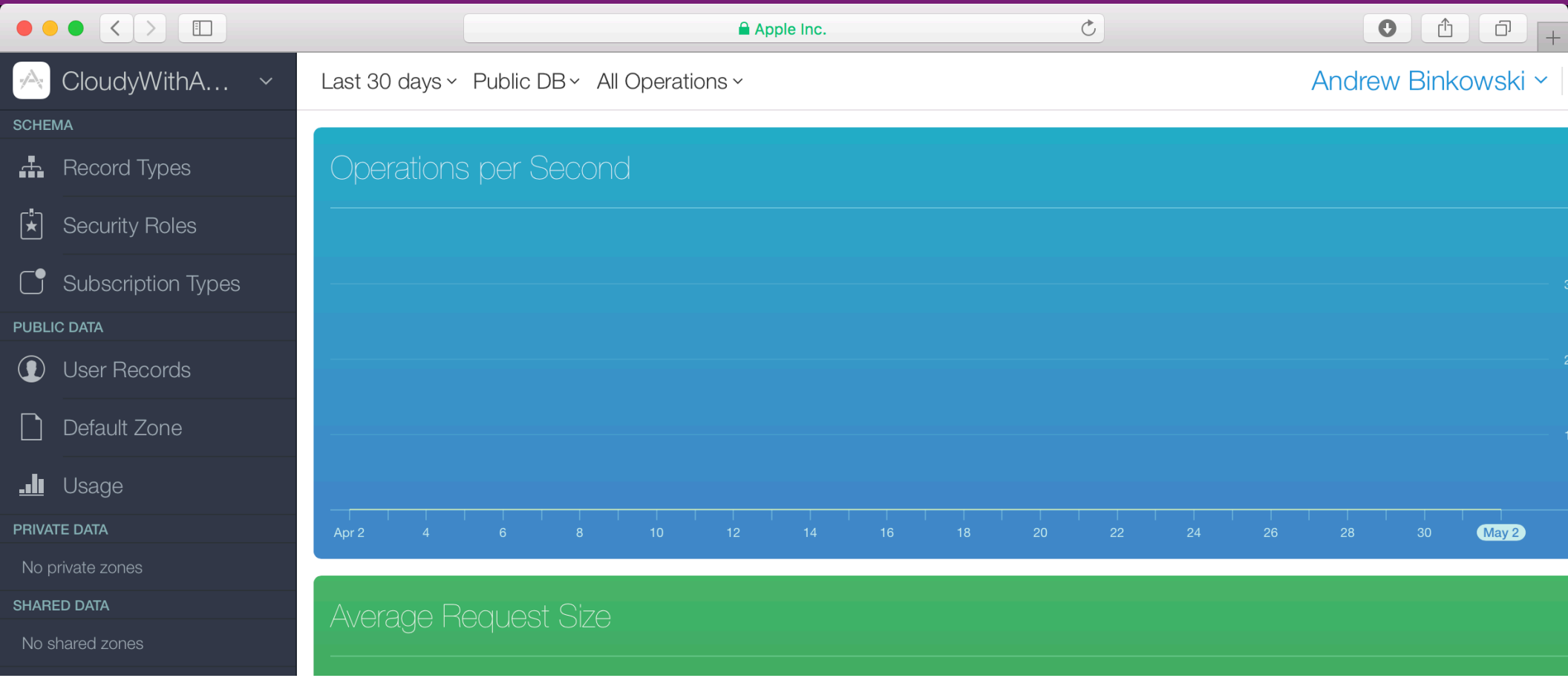
- Dashboard is only way to access permissions
- Data can be created, manipulated
- View analytic information about your data, users and operations



ENABLING CLOUDKIT IN YOUR APPLICATION



ENABLING CLOUDKIT IN YOUR APPLICATION

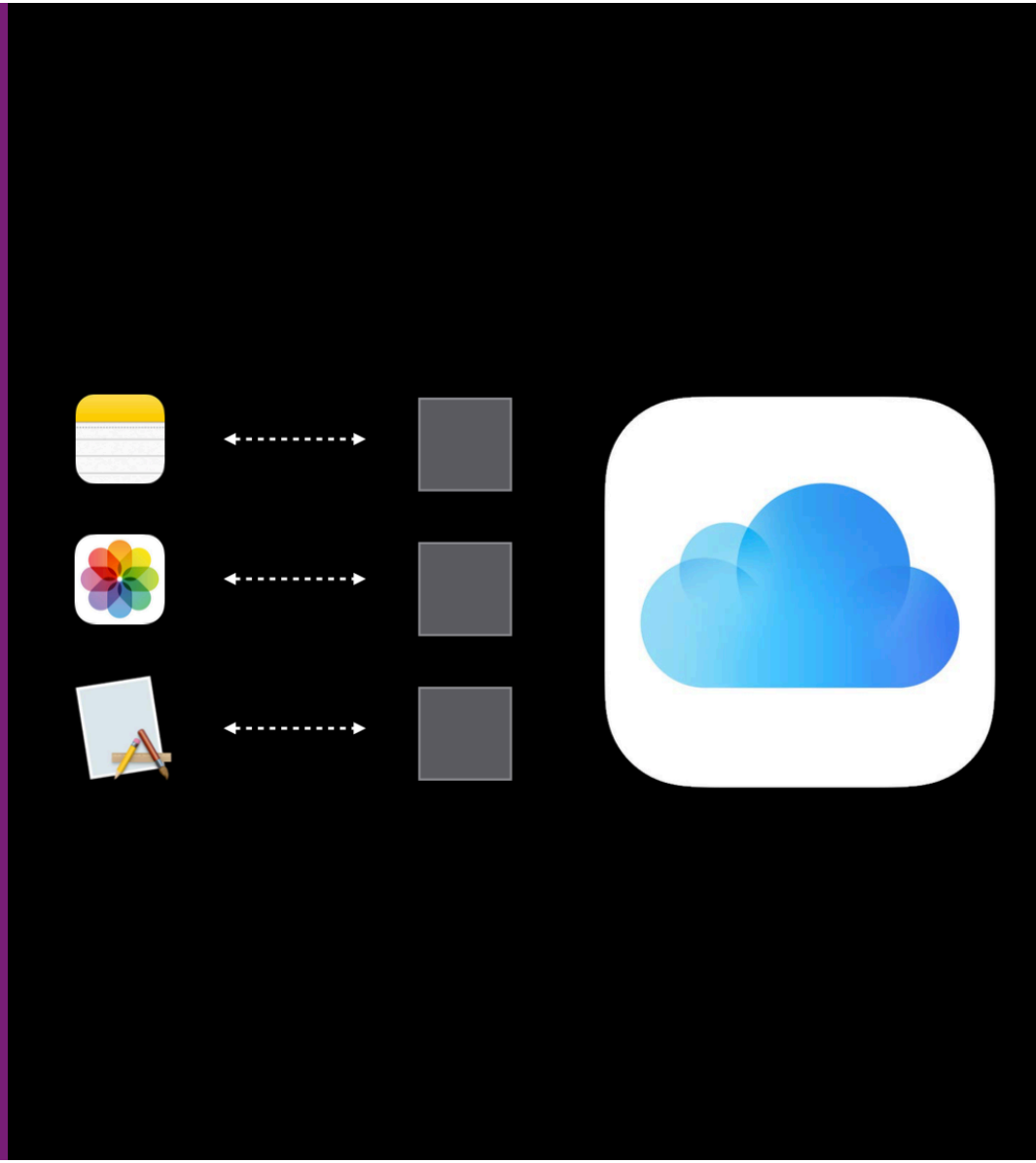


CONTAINER

CLOUDKIT OBJECTS

CONTAINER

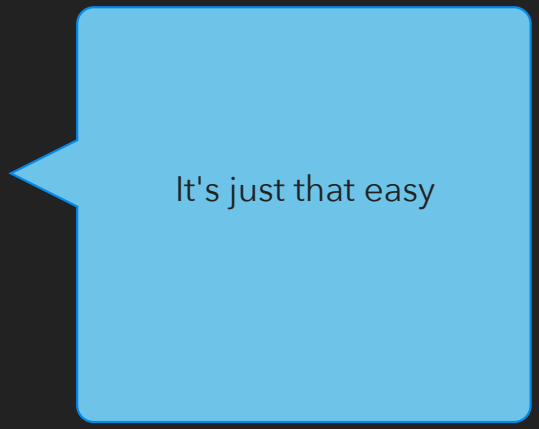
- CKContainer
  - One container per application
  - Data segregation
  - User encapsulation
  - Managed by the developer via portal
  - Can be shared between apps



CLOUDKIT

CONTAINER

```
import CloudKit

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

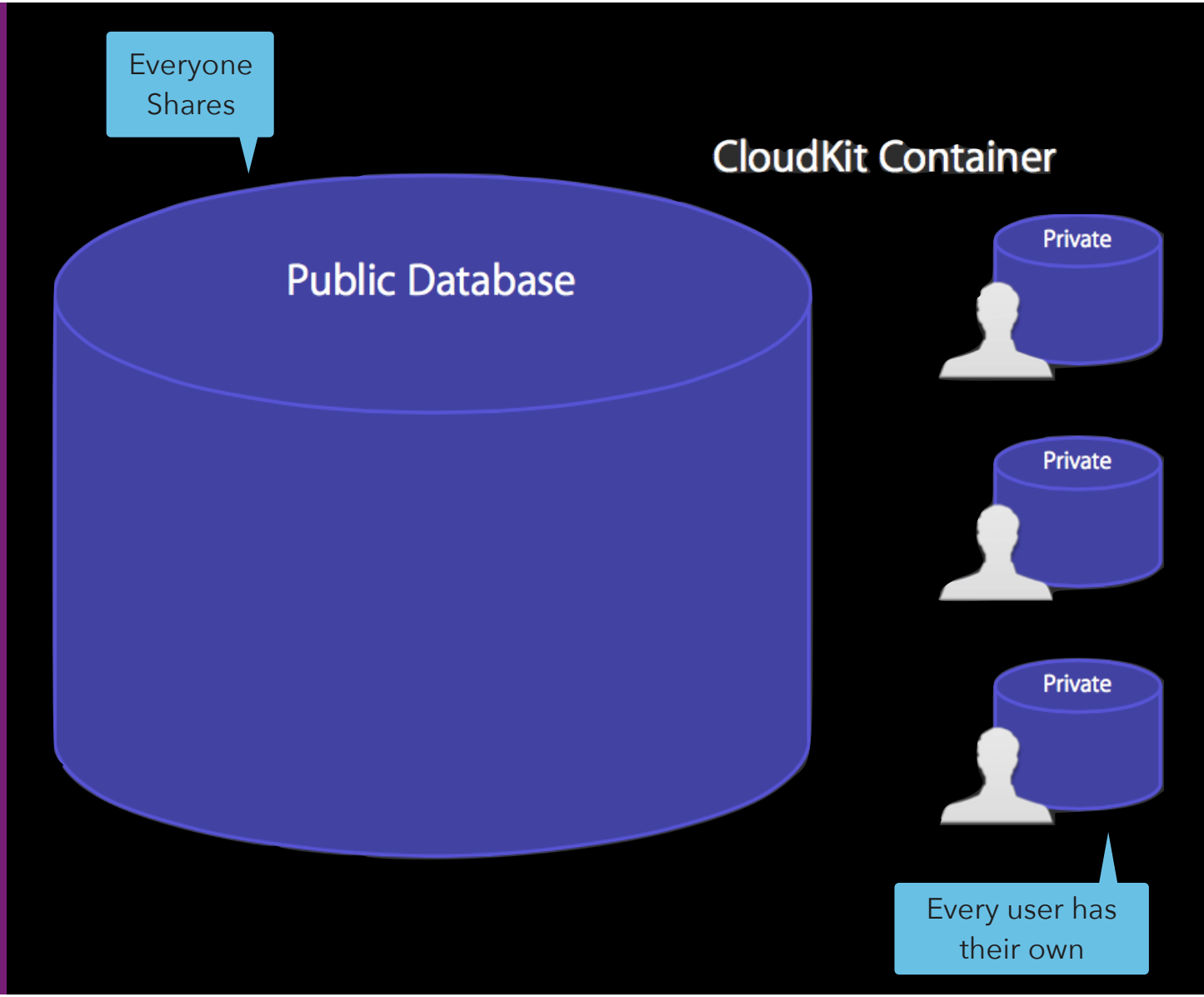


DATABASE

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

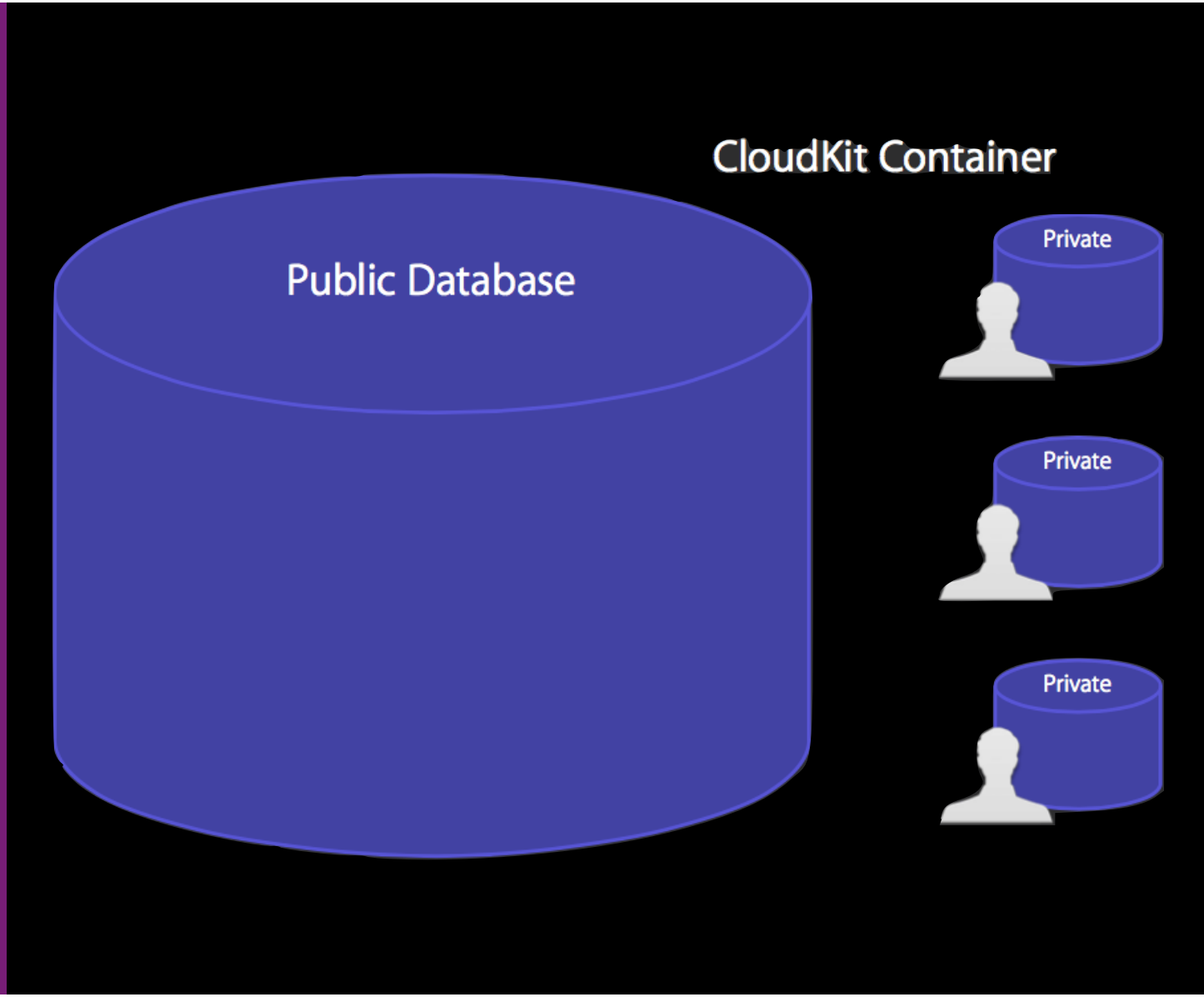
DATABASE

	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

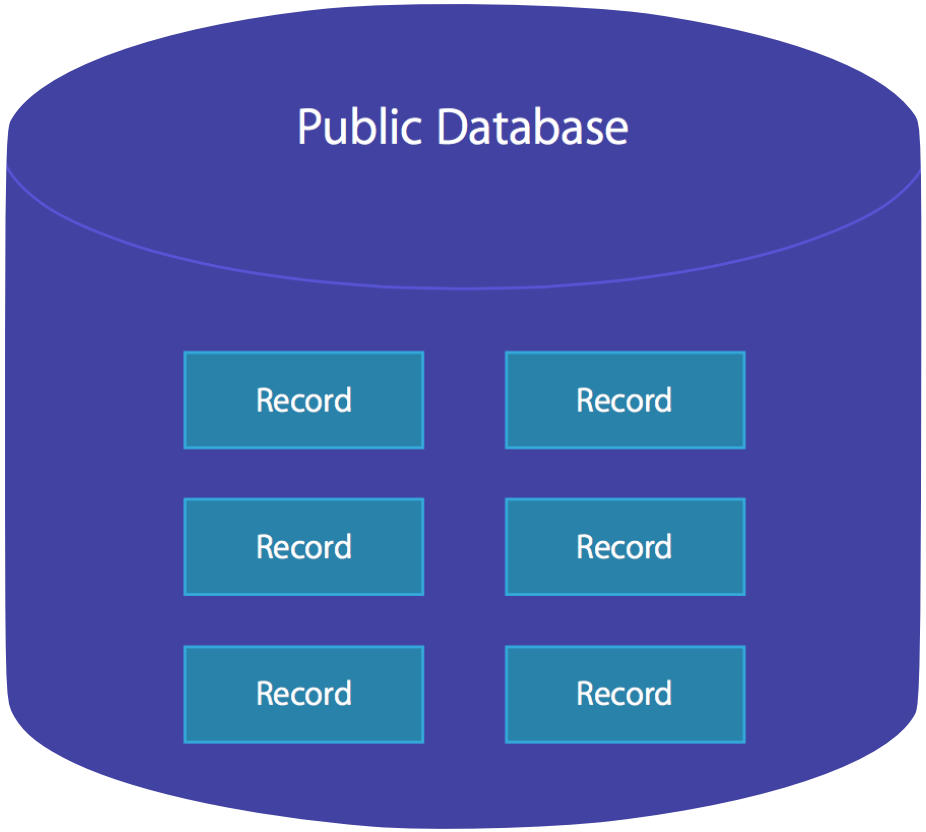
- Private databases
  - Atomic writes
  - Delta downloads
  - No cross-zone references
- Public databases
  - No atomic writes



CLOUDKIT OBJECTS

DATABASE

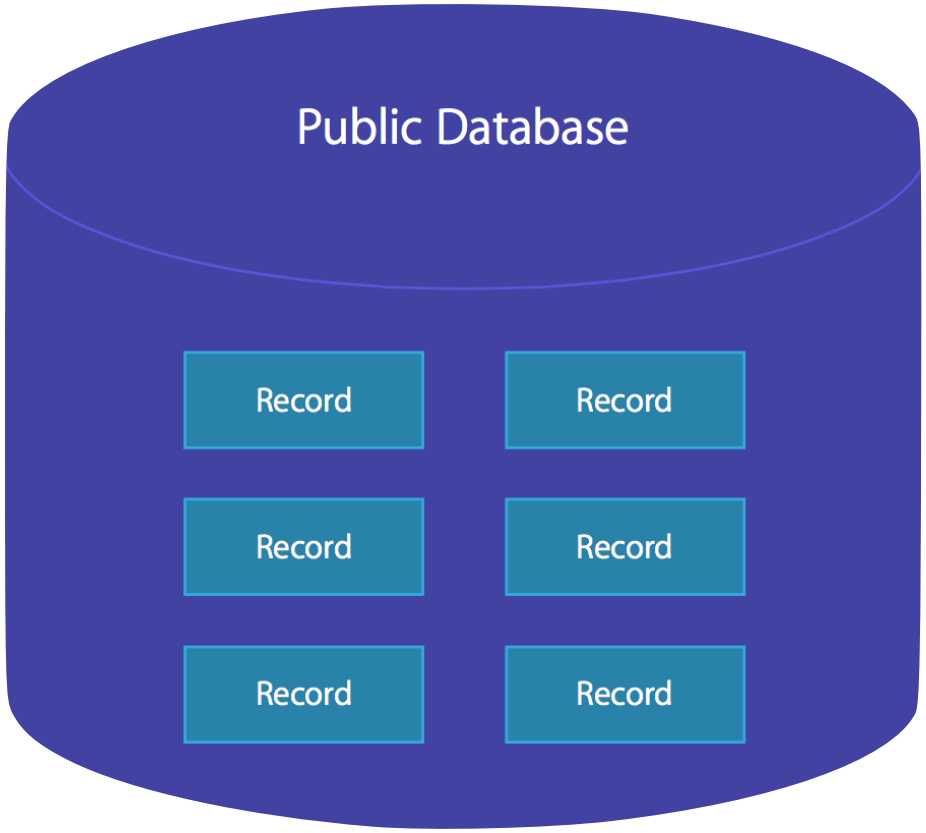
- Structured Data
  - Wraps key/value pairs
  - Types and references
  - Just-in-time schema
  - Metadata
    - Created, edited, etc.



CLOUDKIT OBJECTS

DATABASE

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

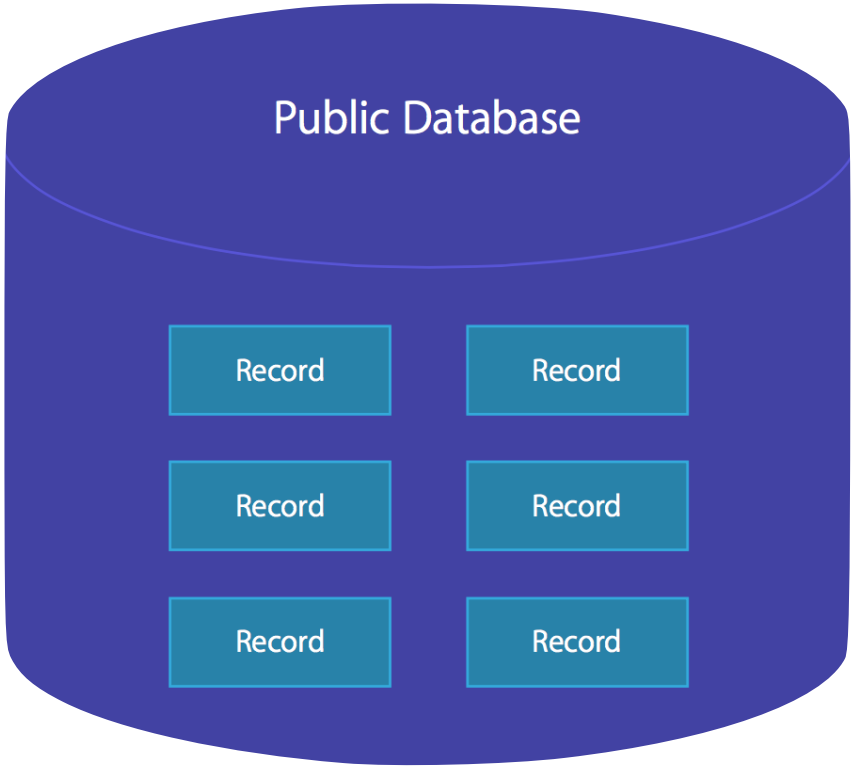


RECORDS

CLOUDKIT OBJECTS

RECORDS

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



CLOUDKIT OBJECTS

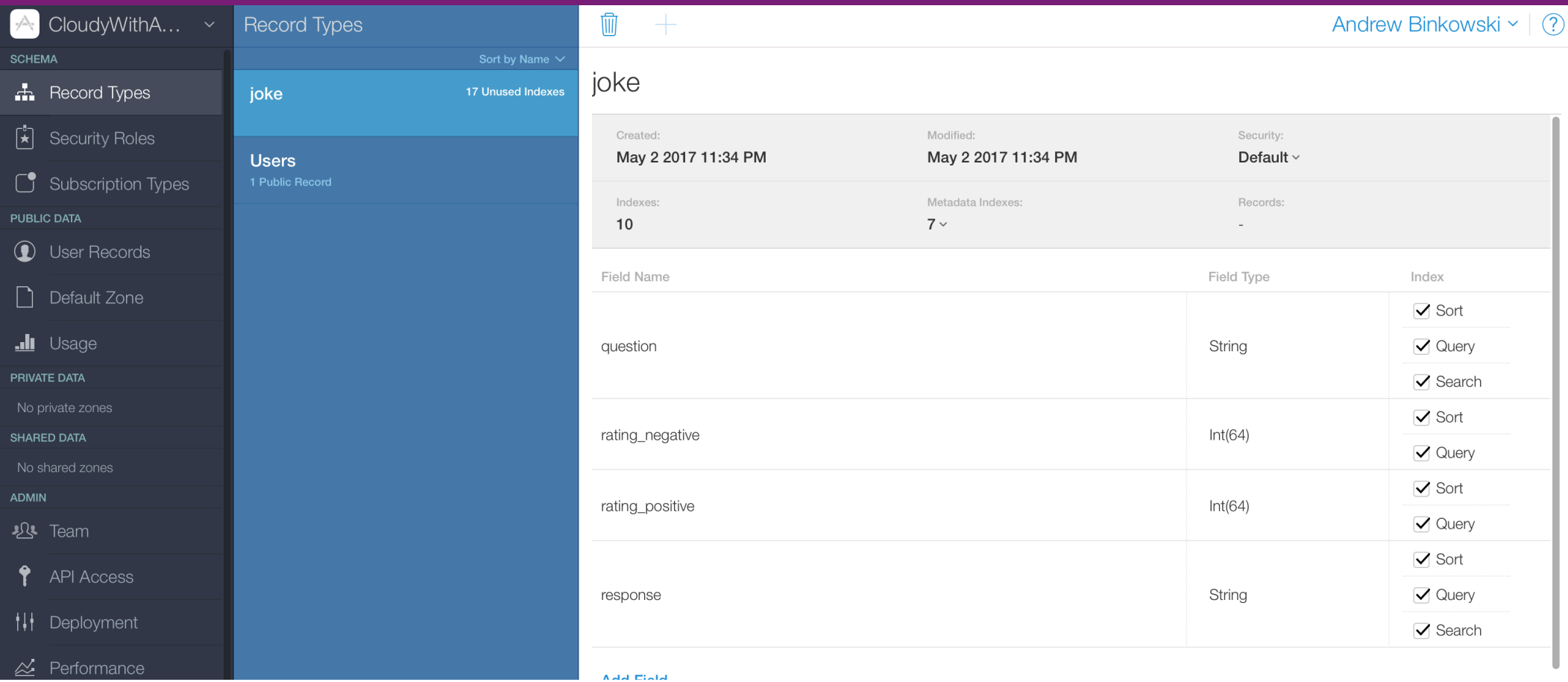
RECORDS

- 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] = 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?
```

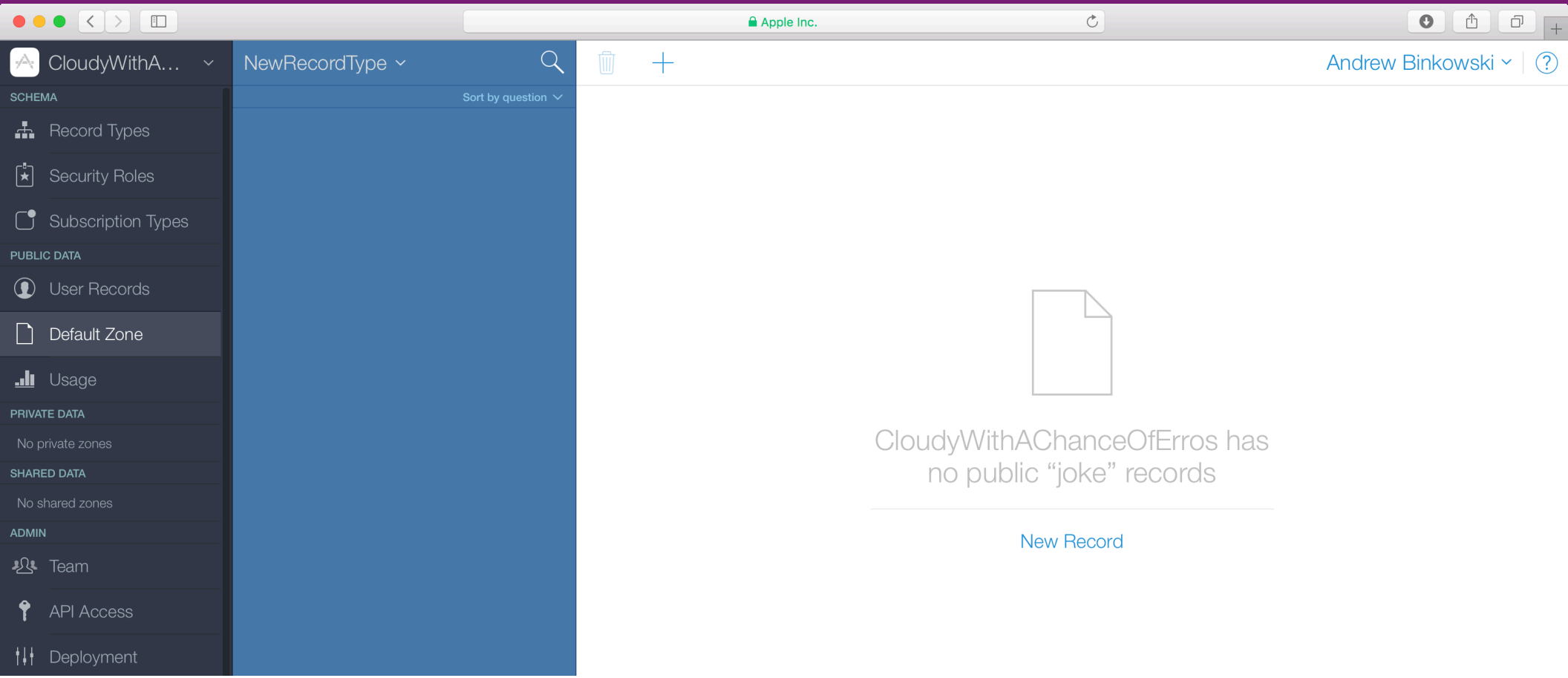
CLOUDKIT OBJECTS

RECORDS



CLOUDKIT OBJECTS

RECORDS



CLOUDKIT OBJECTS

RECORDS

```
/// 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

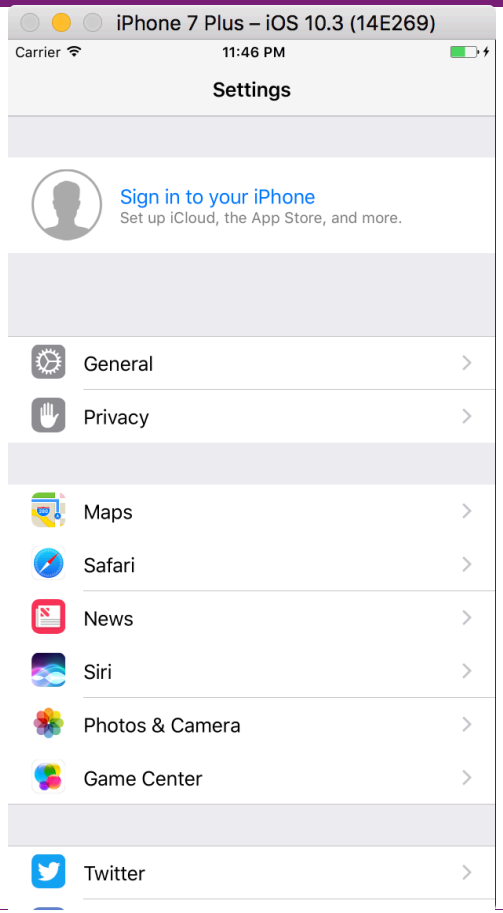
```swift
import CloudKit

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)")
    }

  }

}
```

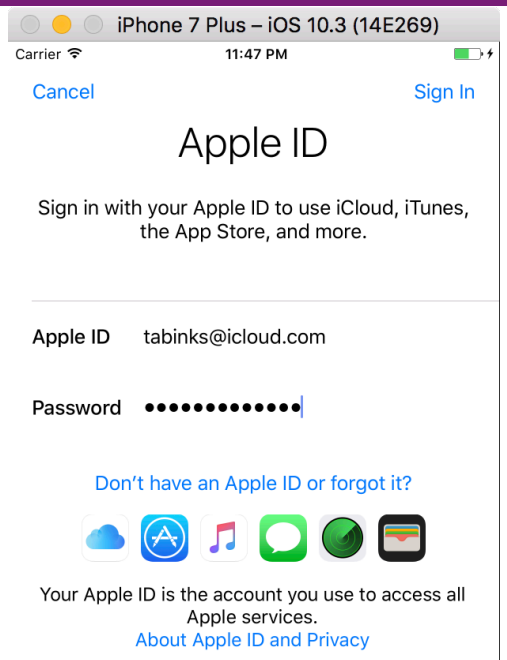
Your first CloudKit error

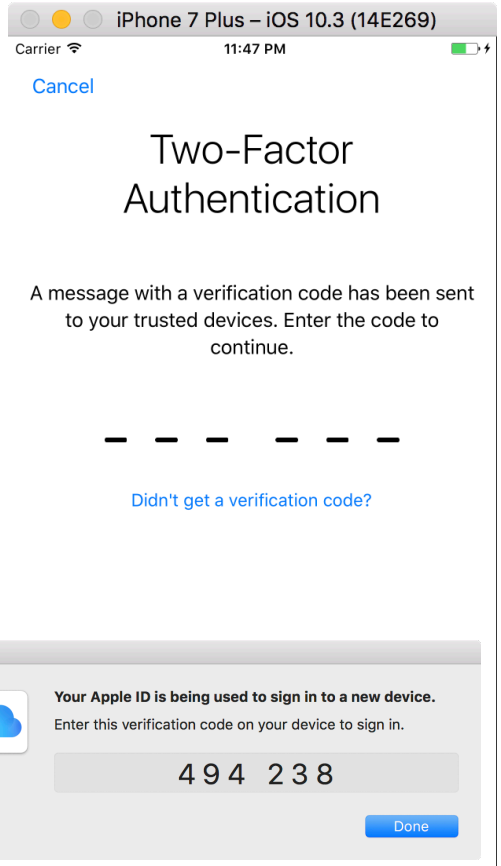
```
🐞: This request requires an authenticated account
```

CLOUDKIT OBJECTS

RECORDS

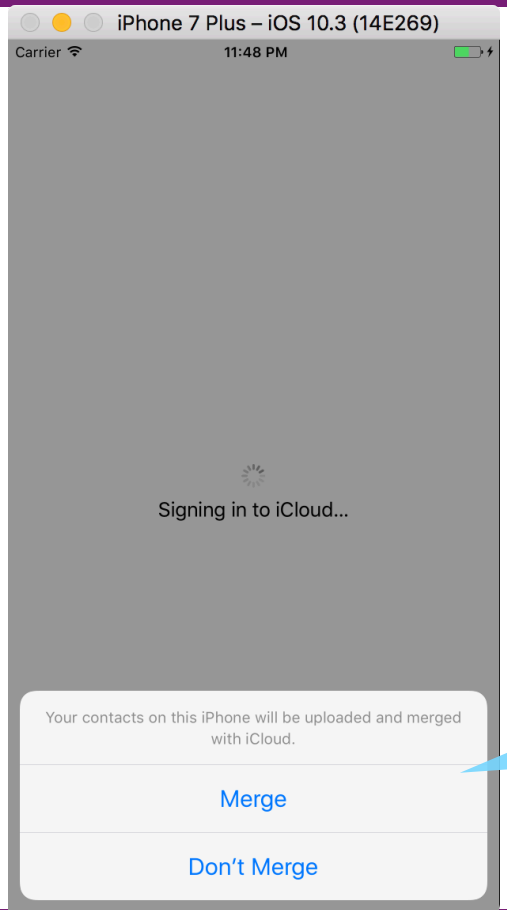




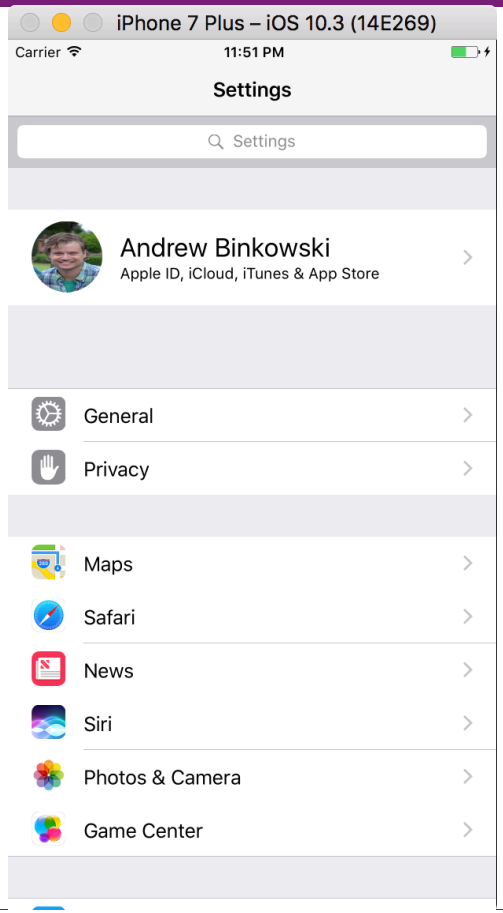


CLOUDKIT OBJECTS

RECORDS



Be careful...you probably don't want to merge



CLOUDKIT OBJECTS

RECORDS

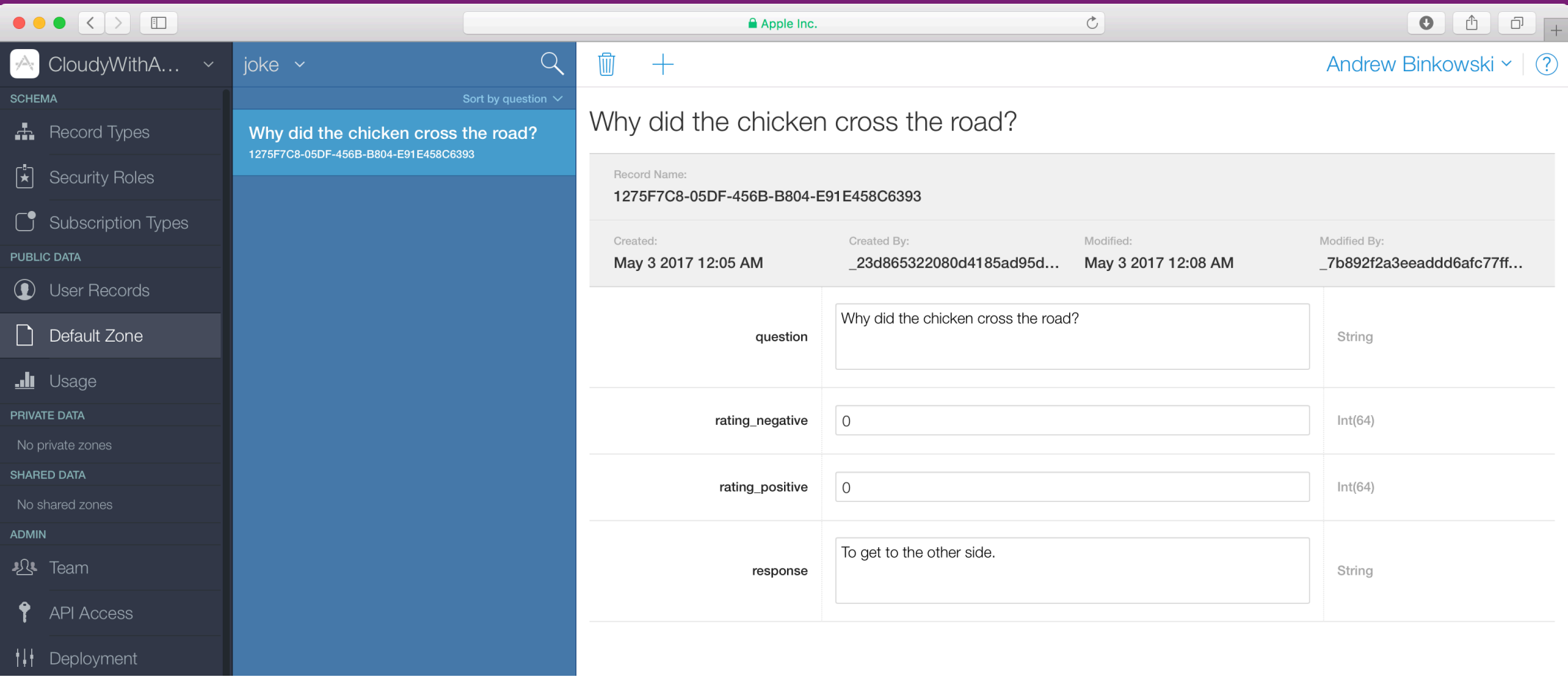
```
    let record = CKRecord(recordType: "joke")
    record.setValue(joke, forKey: "question")
    record["response"] = "To get to the other side." as CKRecordValue

    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?"
})
```

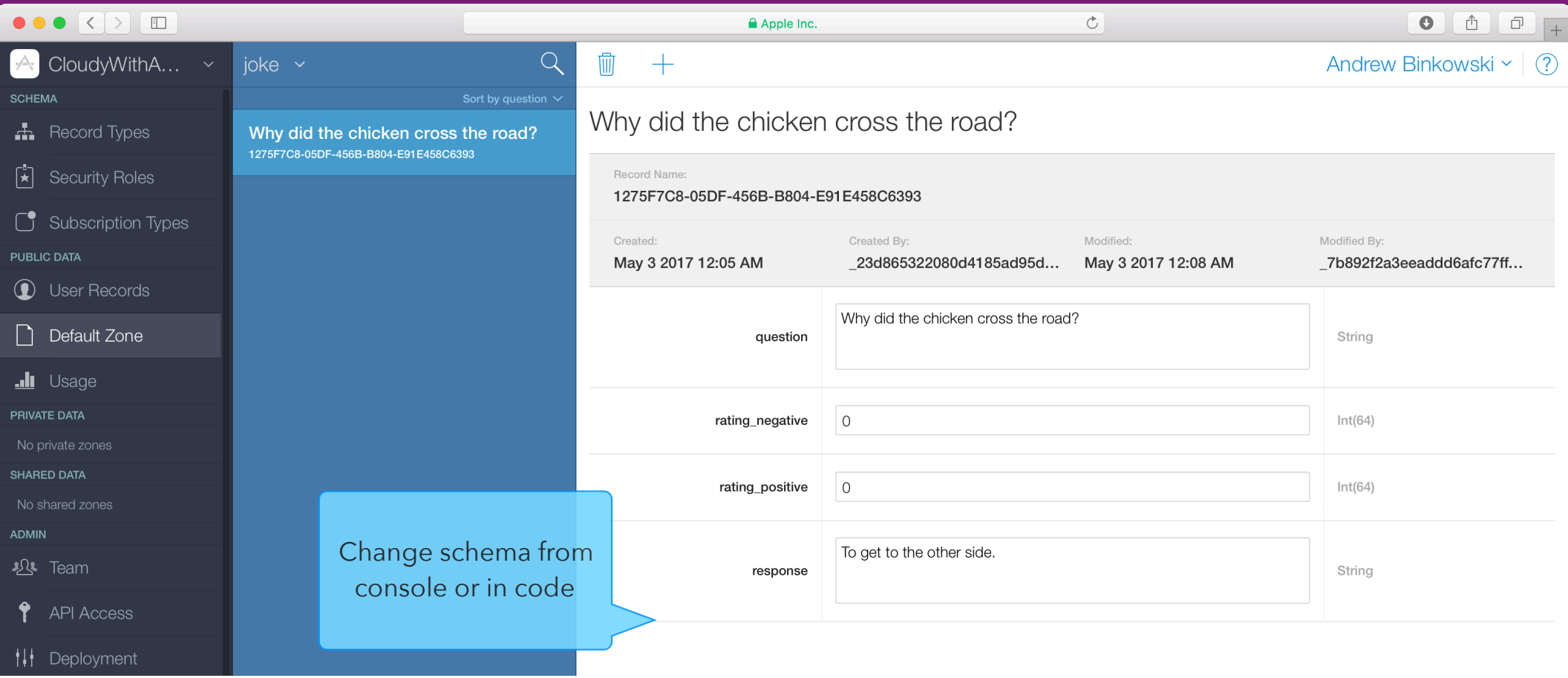
CLOUDKIT OBJECTS

RECORDS



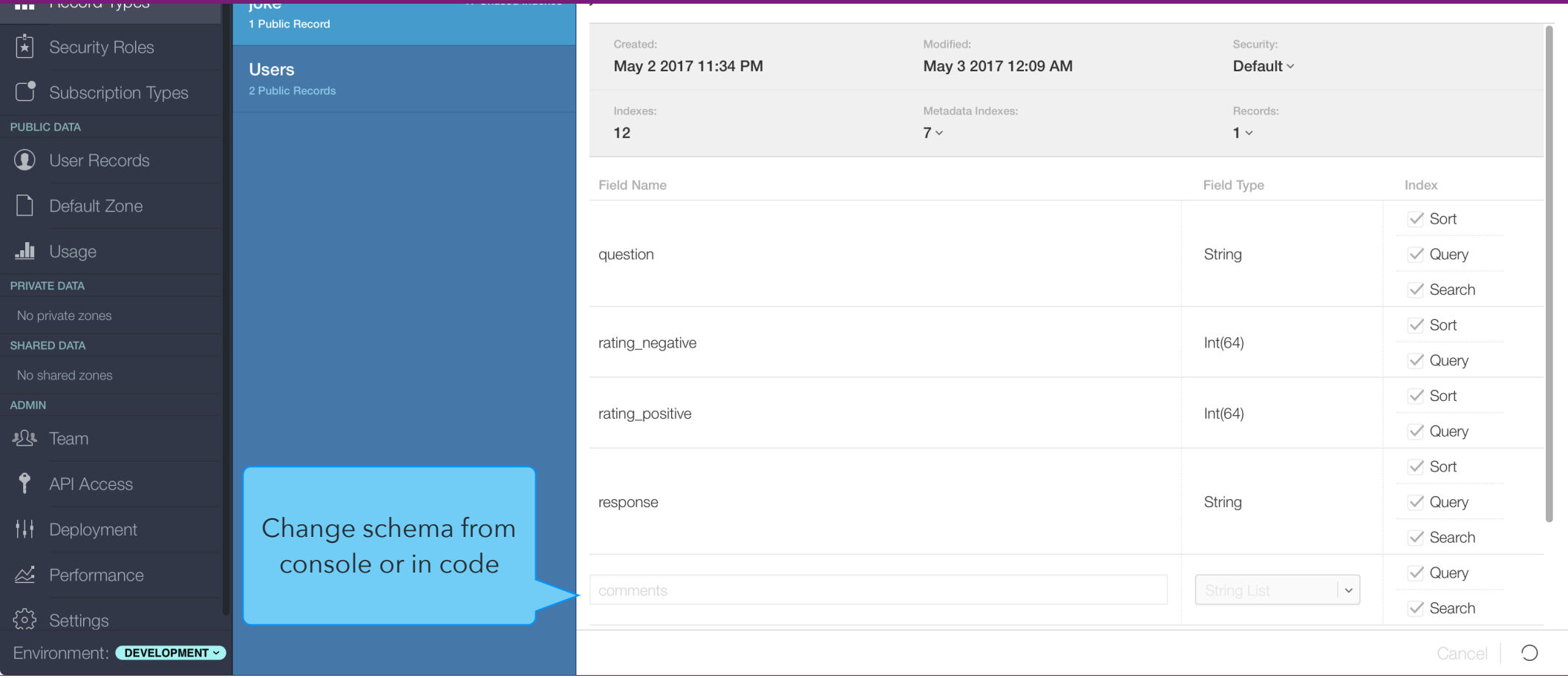
CLOUDKIT OBJECTS

RECORDS



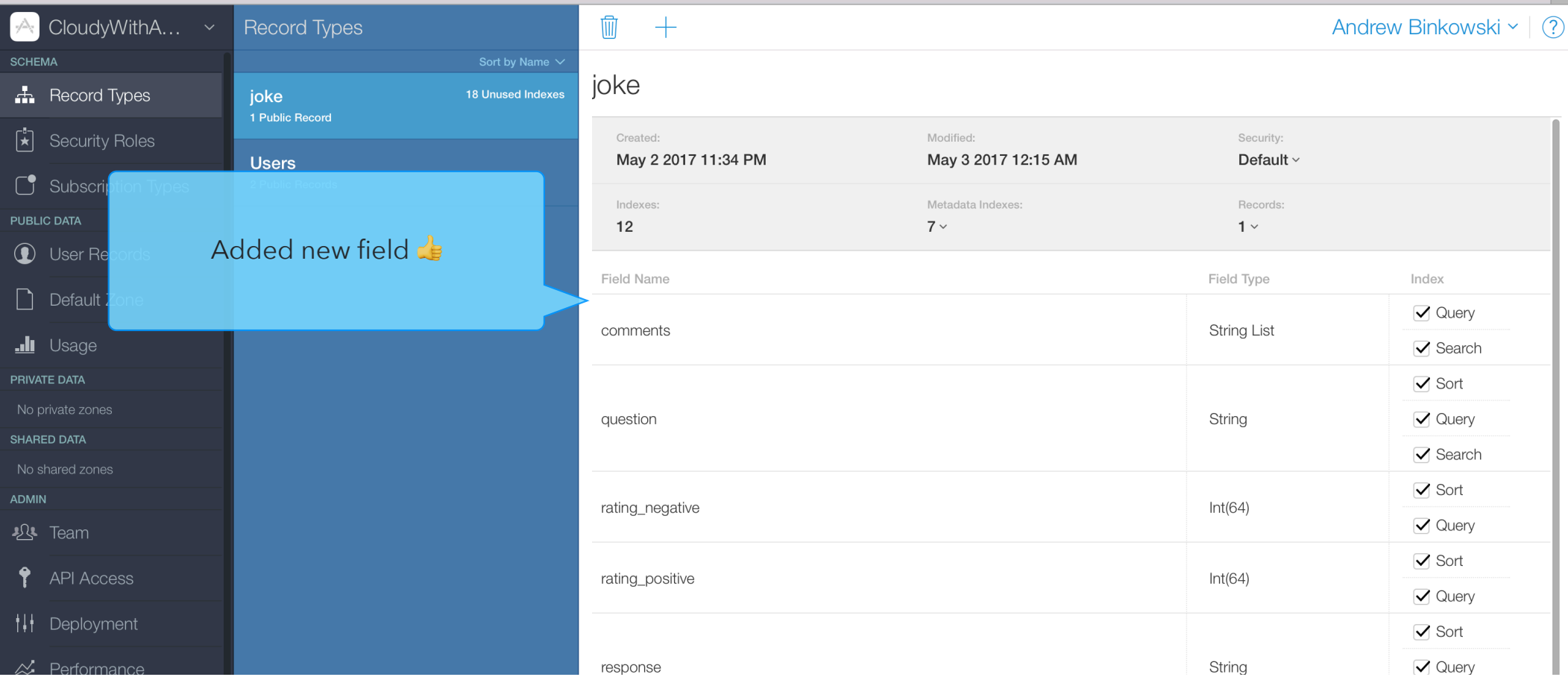
CLOUDKIT OBJECTS

RECORDS



CLOUDKIT OBJECTS

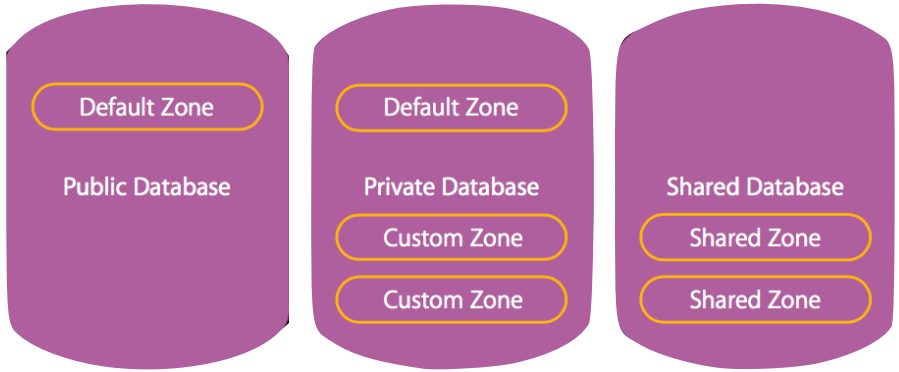
RECORDS



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

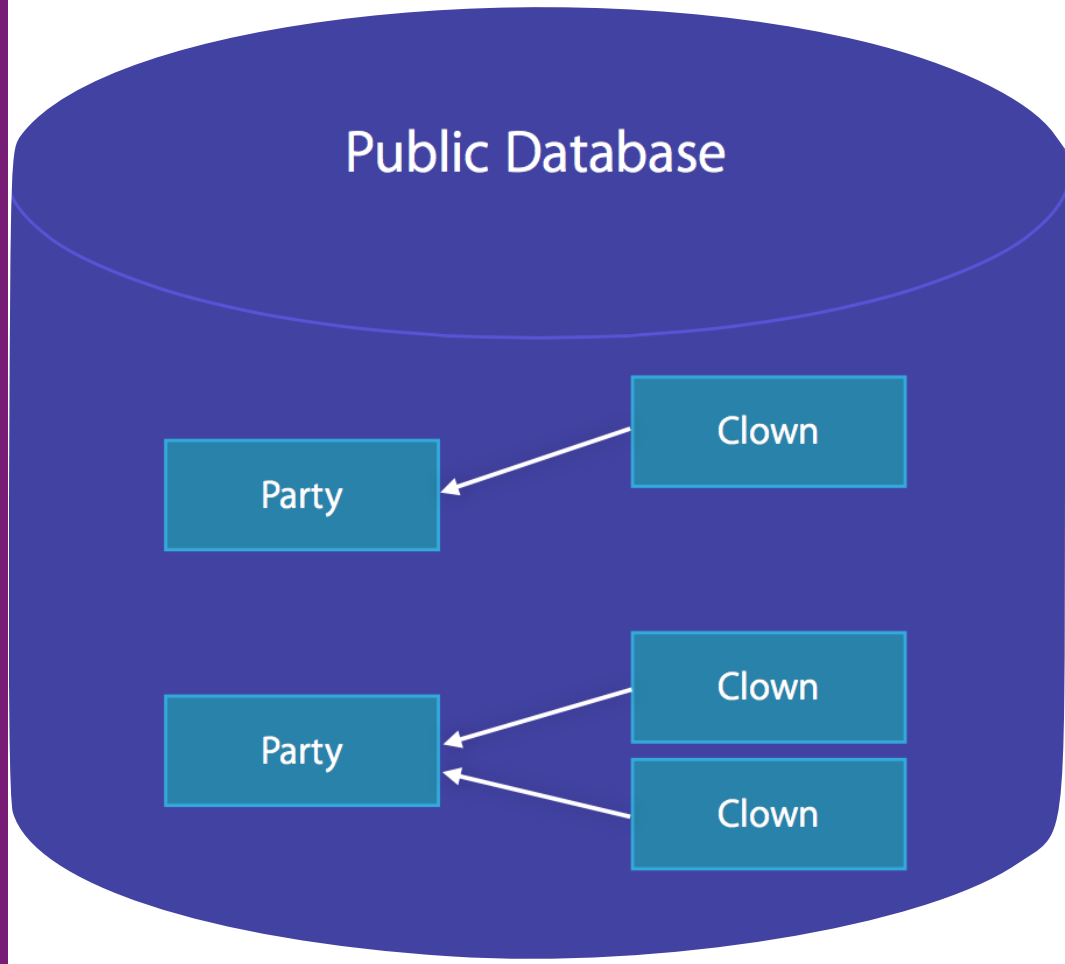
```
//
//  CKRecordID.h
//  CloudKit
//
//  Copyright (c) 2014 Apple Inc. All rights r
//

#import <Foundation/Foundation.h>

@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

CLOUDKIT OBJECTS

RECORDS

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



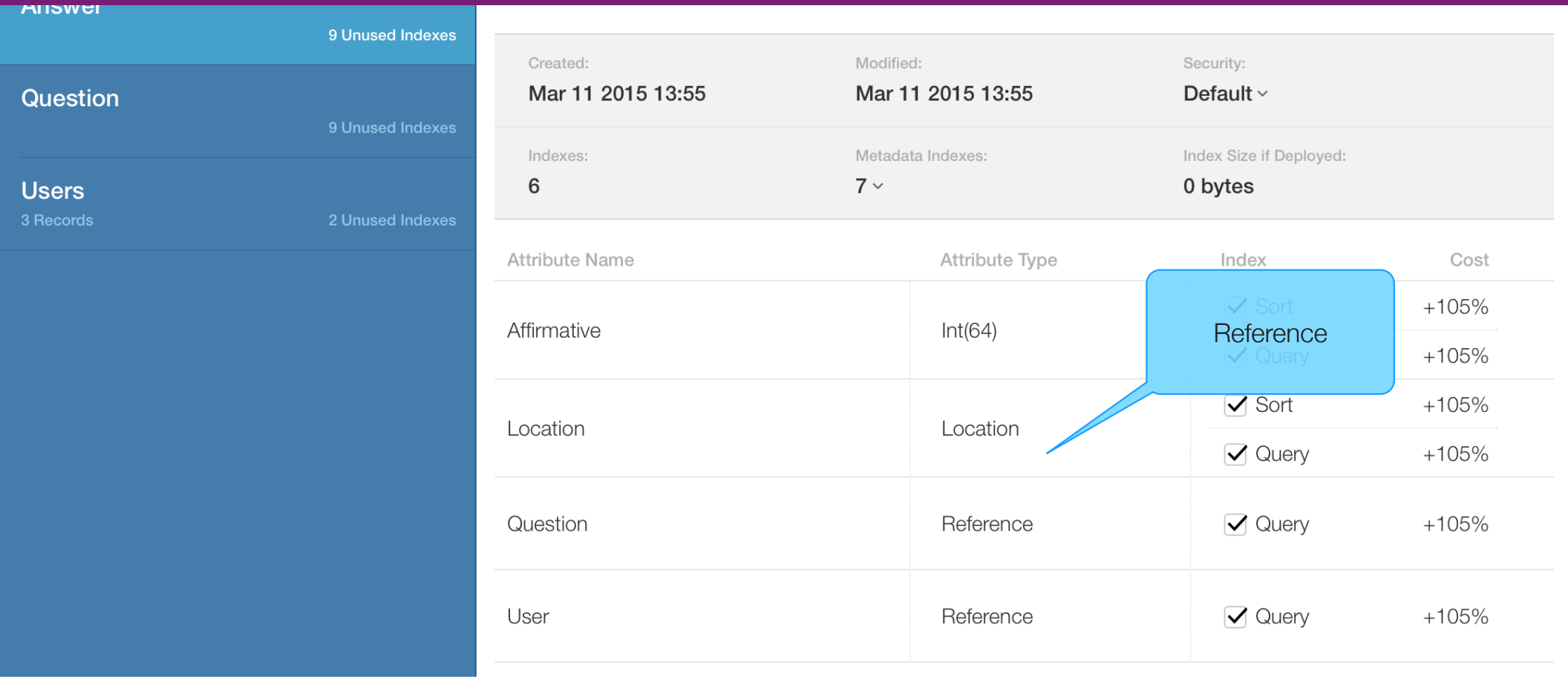
CLOUDKIT API

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

What happens on delete

- CKReferenceAction
  - None
  - Self-delete

CLOUDKIT API

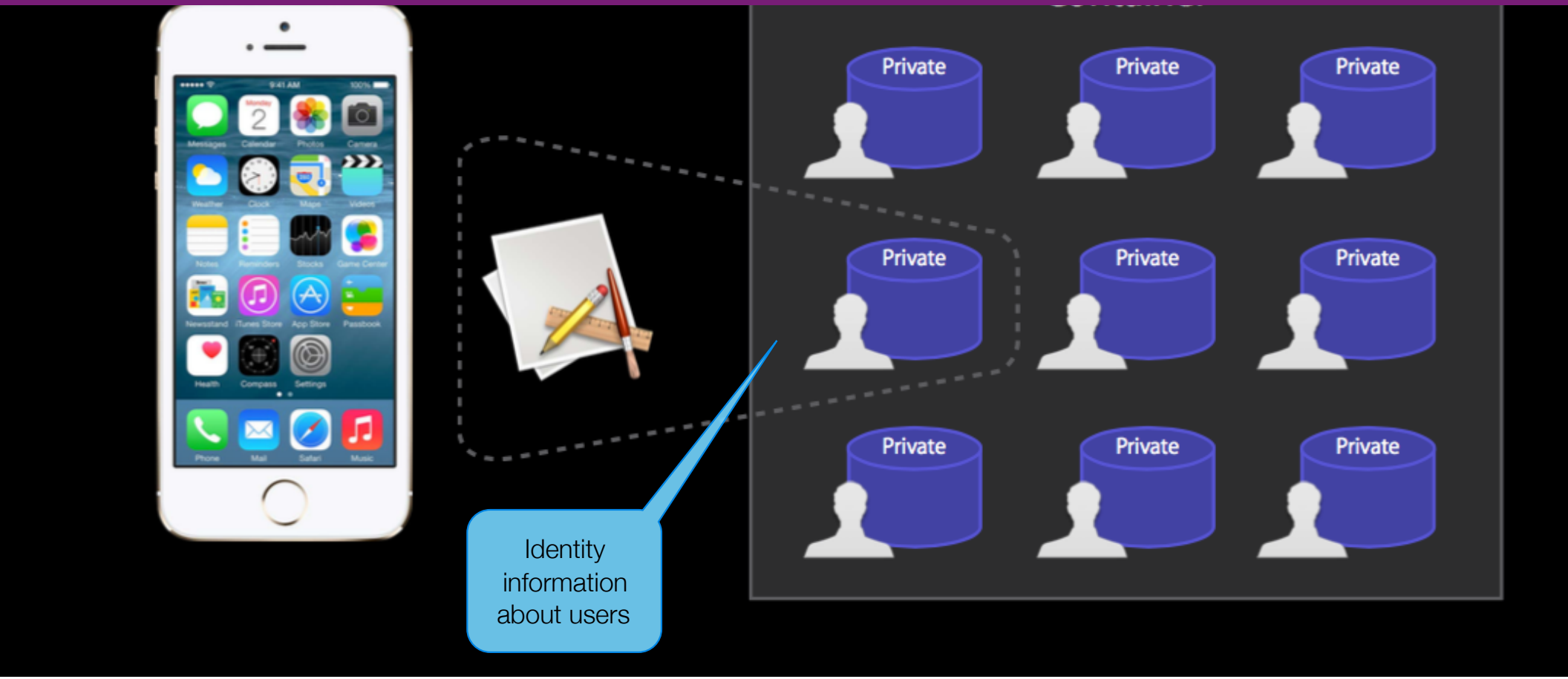


USER RECORDS

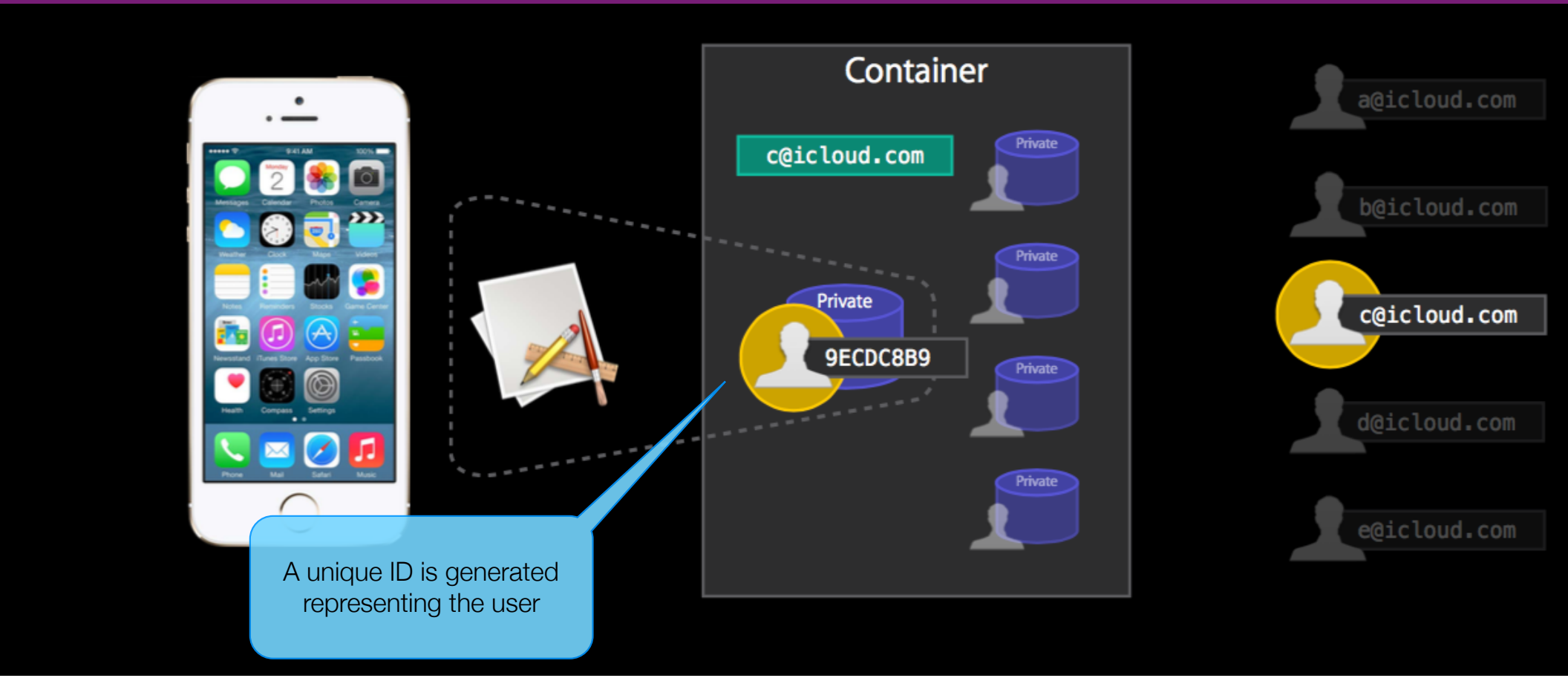
USER RECORDS

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

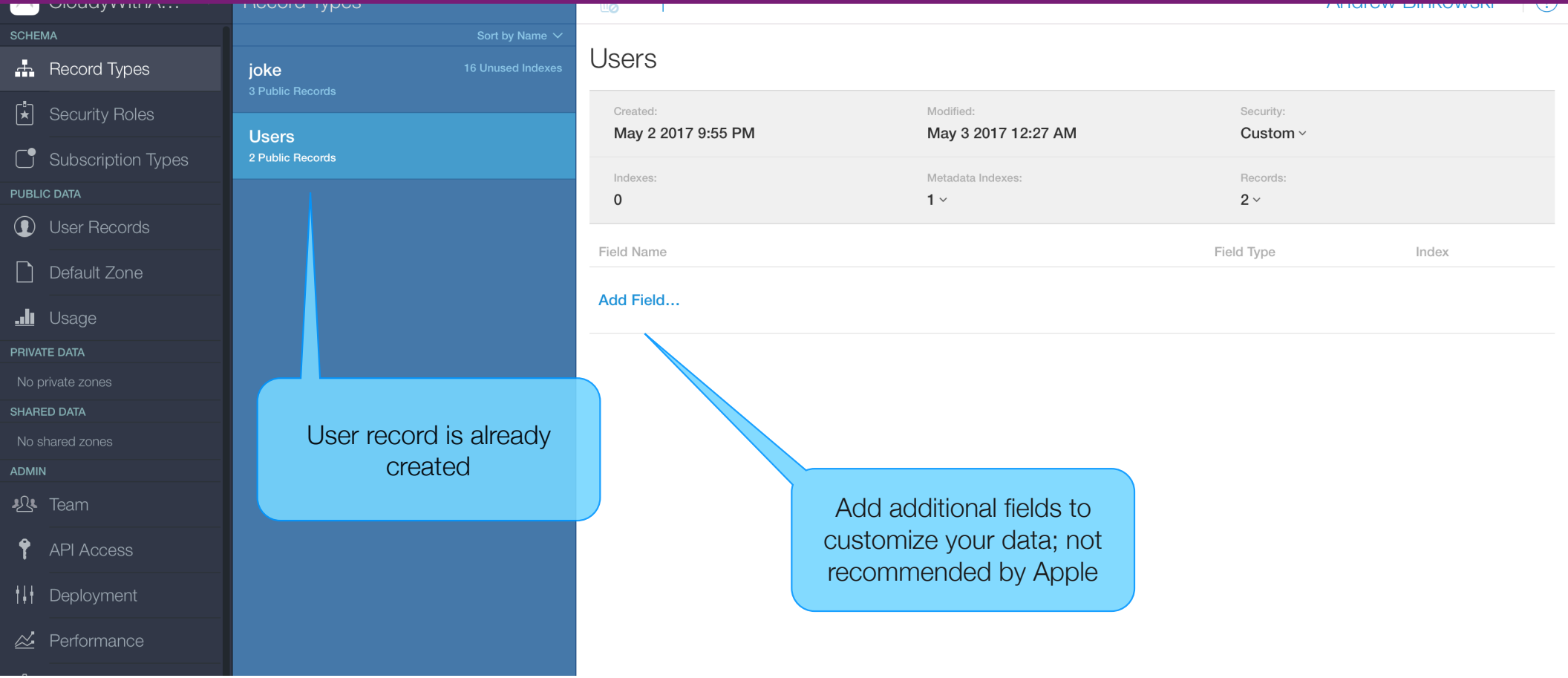
USER RECORDS



USER RECORDS

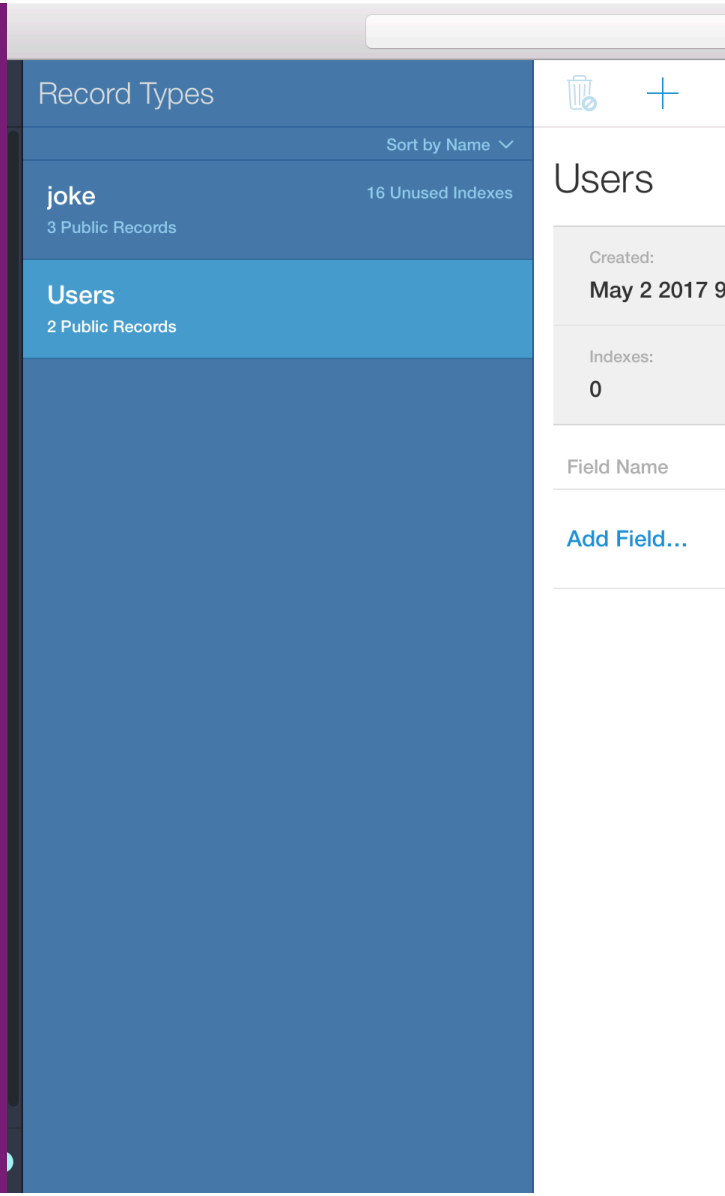


USER RECORDS



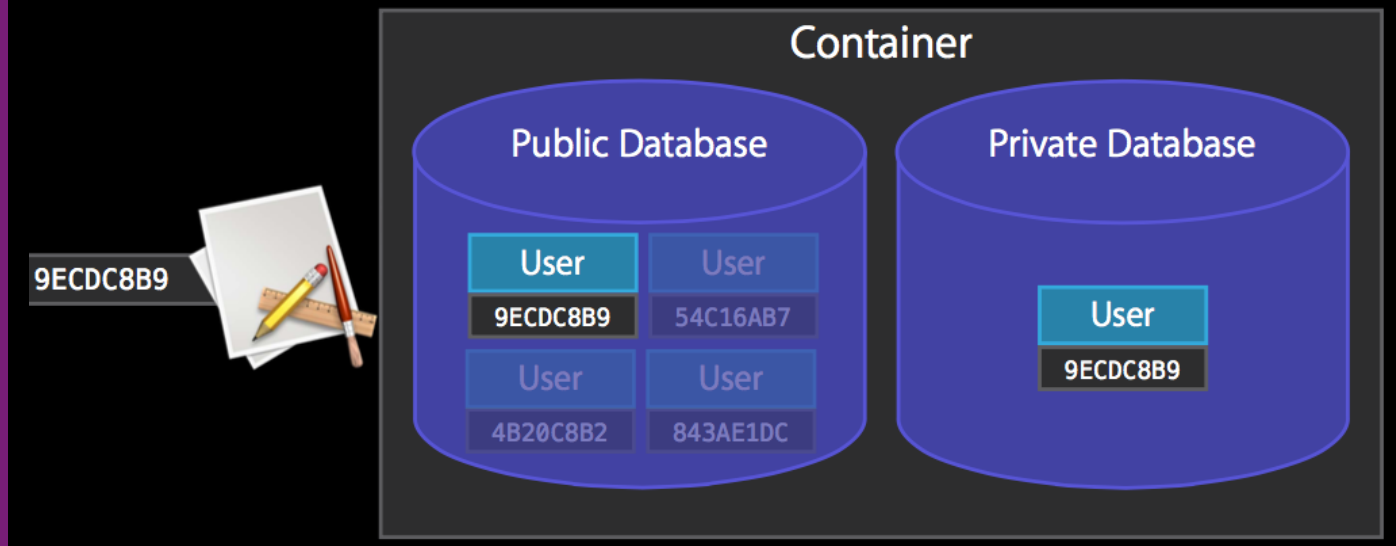
USER RECORDS

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



CloudKit User Accounts

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



USER RECORDS

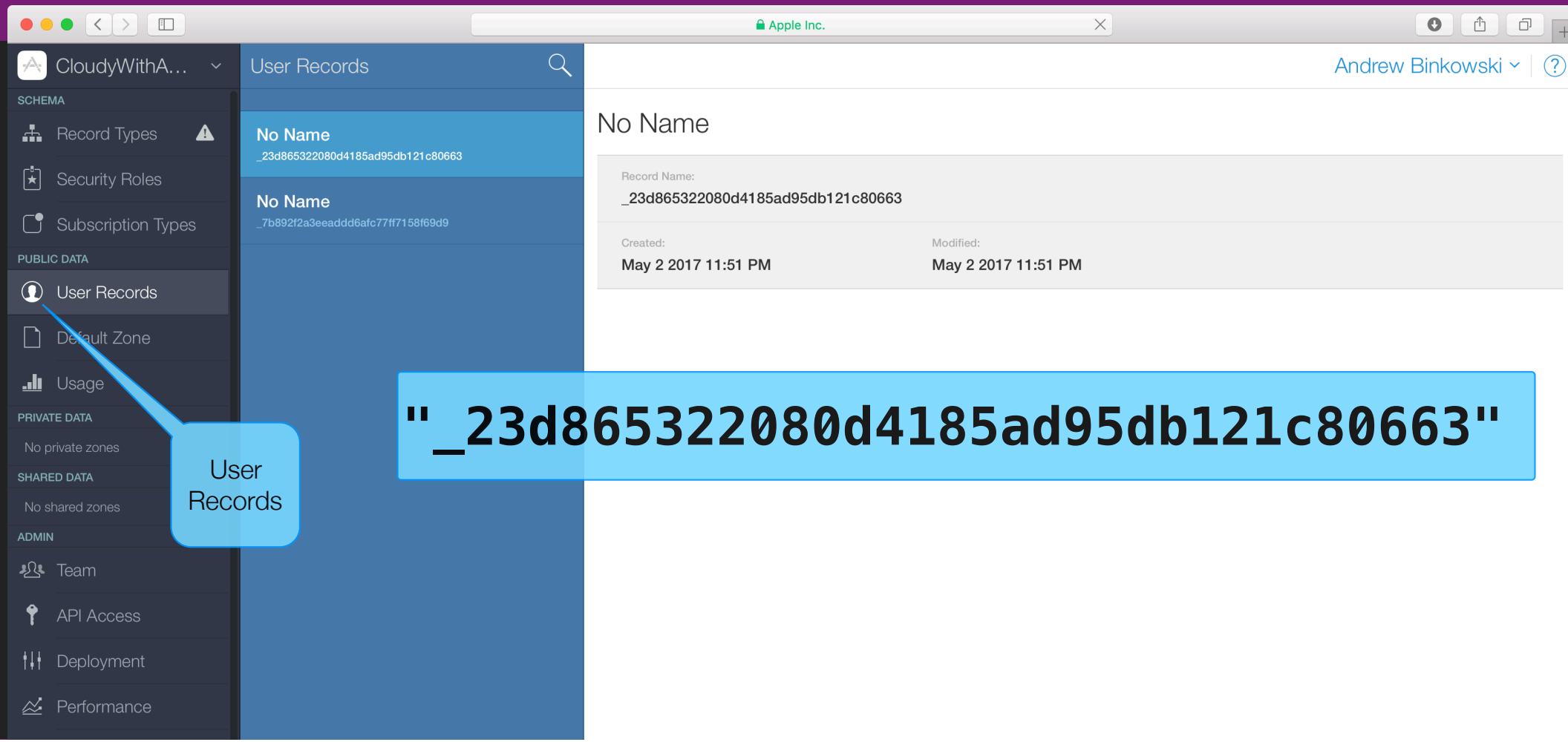
```
let container = CKContainer.default()

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

      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)
      }
}
```

"_23d865322080d4185ad95db121c80663"

USER RECORDS



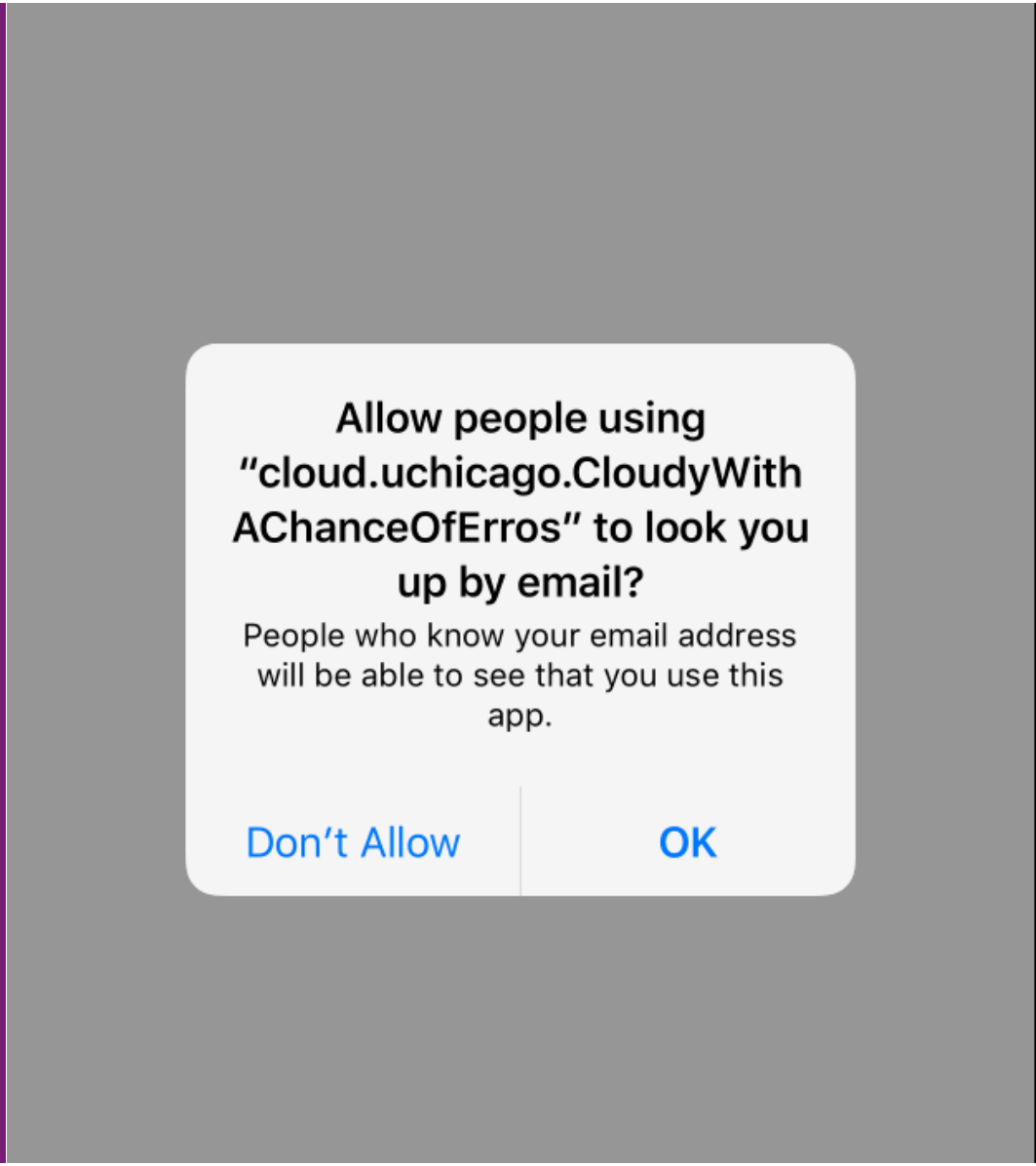
USER RECORDS

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

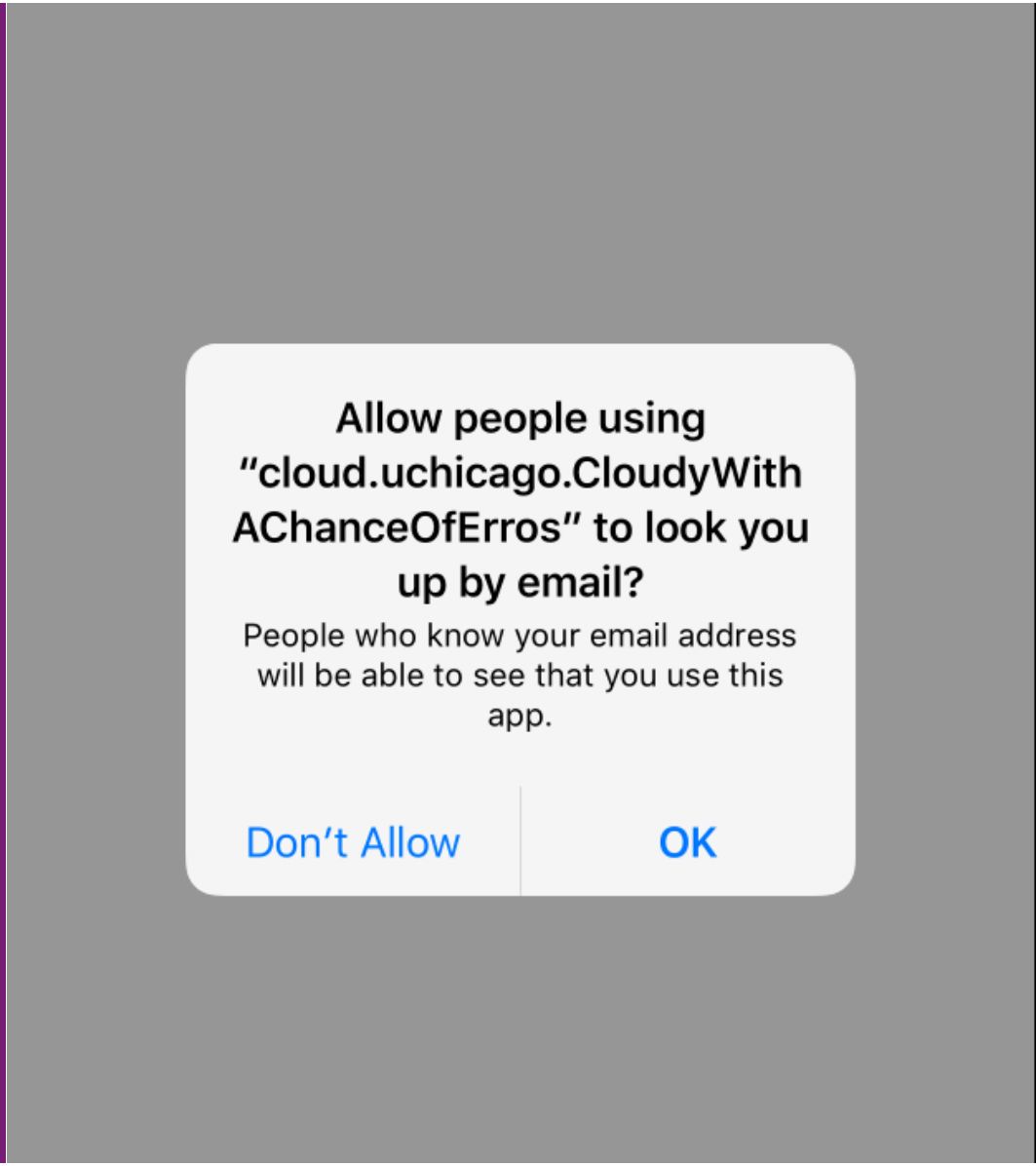
USER RECORDS

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



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)
          })
        }
      }
```

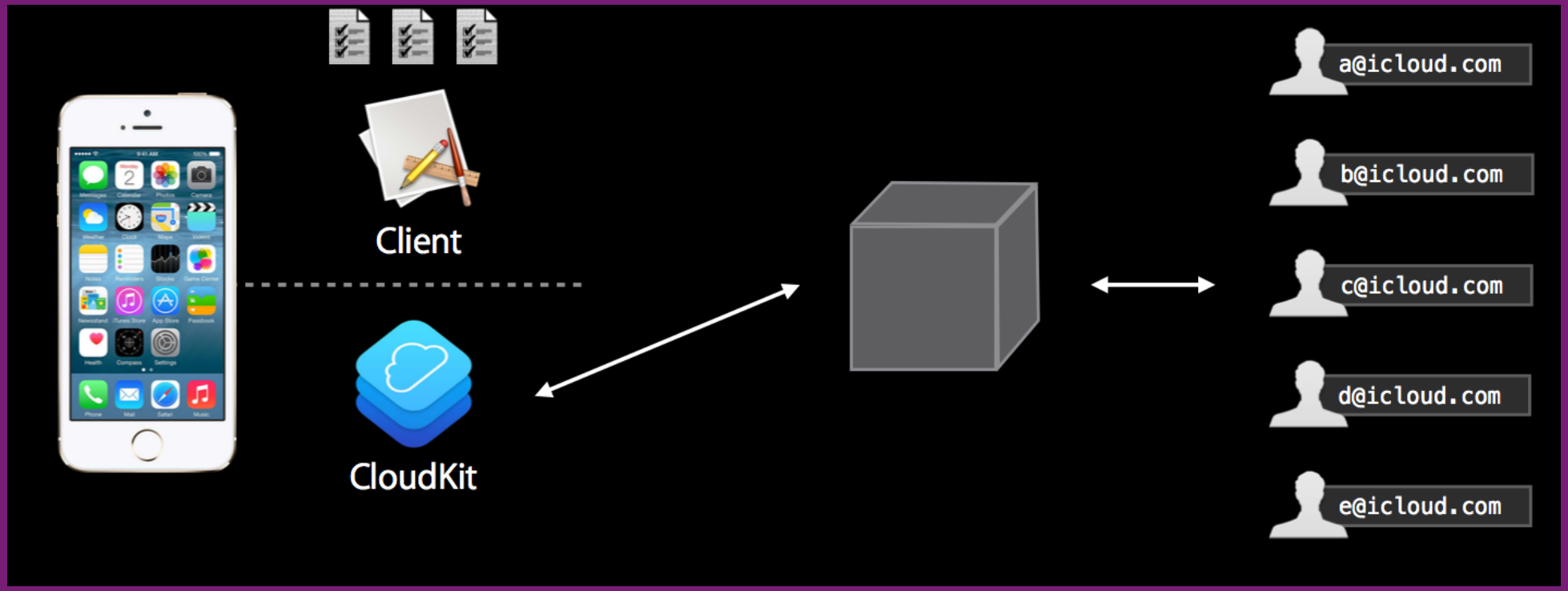
```
CKContainer.default().requestApplicationPermission(.userDiscoverability) { (status, error) in
  CKContainer.default().fetchUserRecordID { (record, error) in
    CKContainer.default().discoverUserIdentity(withUserRecordID: record!, completionHandler: { (userID, error) in
      userName = (userID?.nameComponents?.givenName)!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

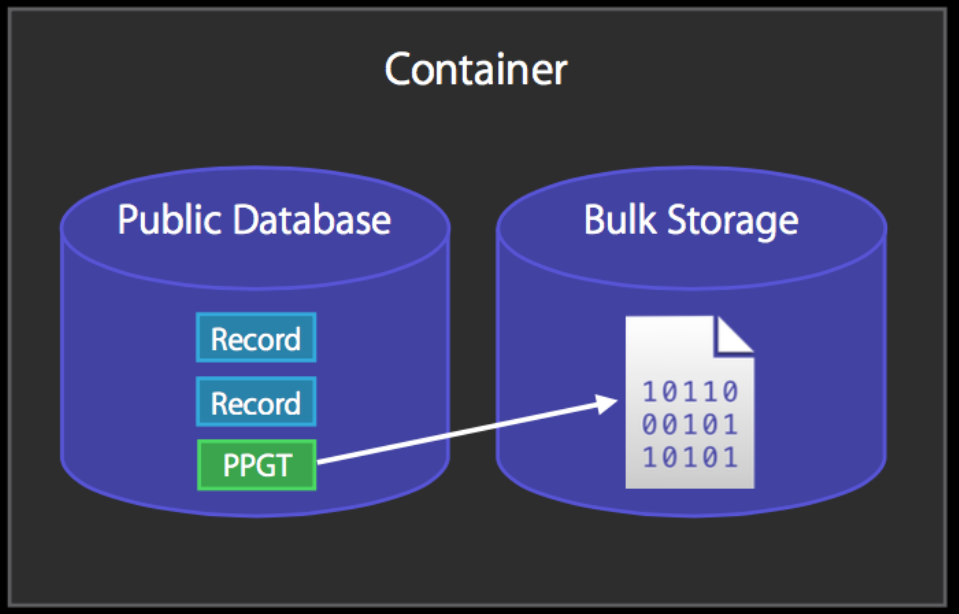


- You can get all friends who use the same app (discover users api)

ASSETS

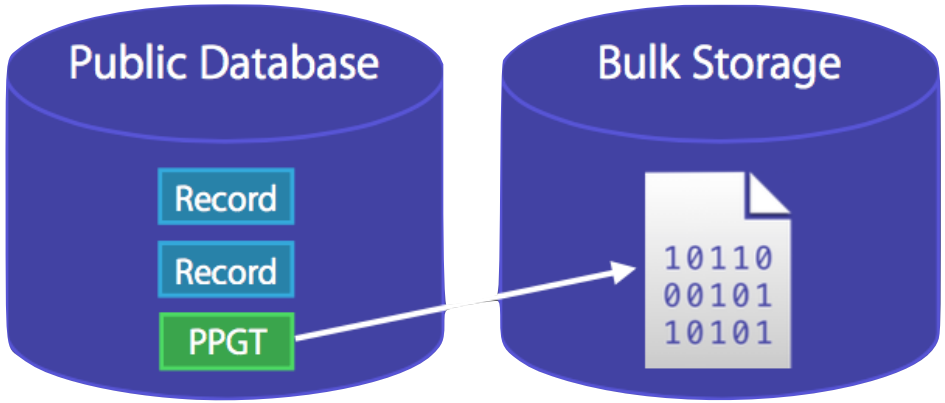
CLOUDKIT API

- CKAssets is bulk storage for CloudKit

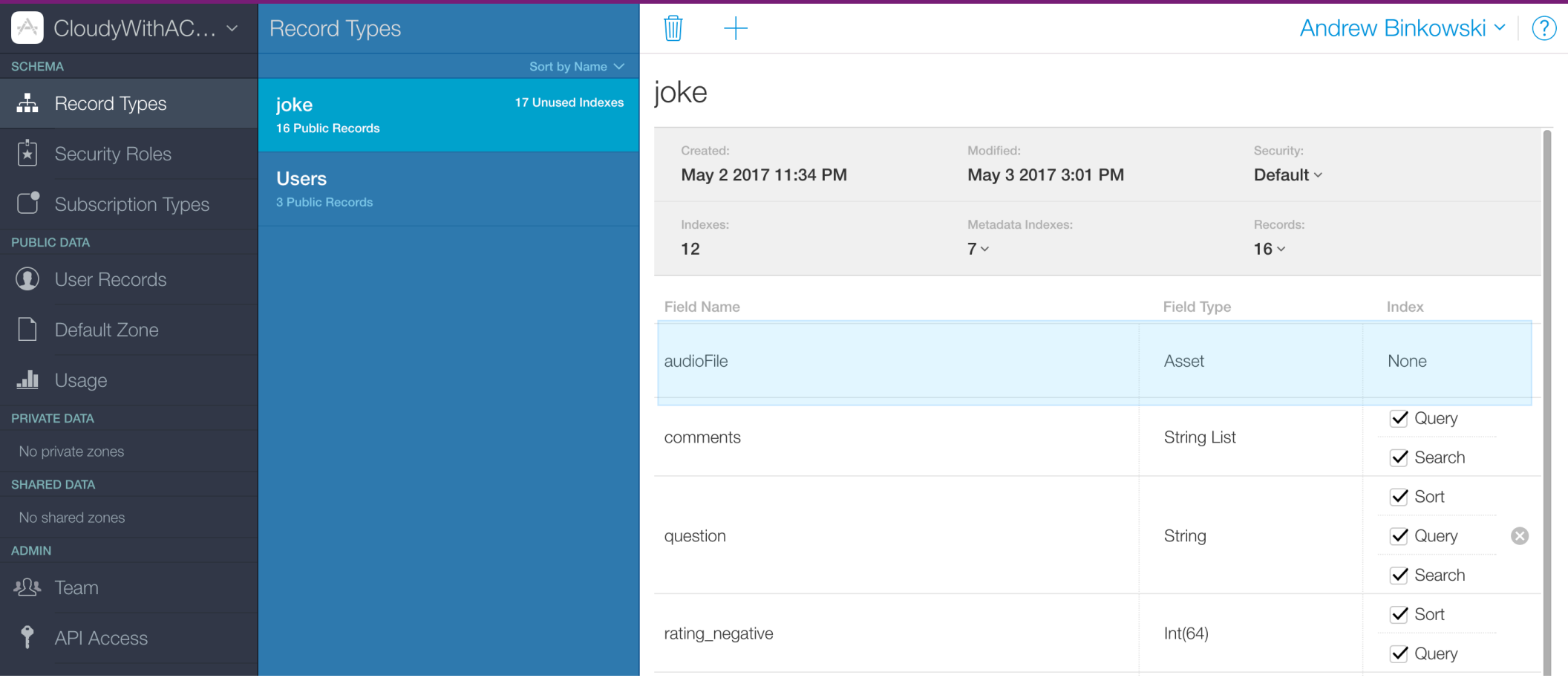


CLOUDKIT API

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



\


CloudyWithAC...
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
joke
17 Unused Indexes
16 Public Records
Users
3 Public Records
Andrew Binkowski
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
Query
Search
question
String
Sort
Query
Search
rating_negative
Int(64)
Sort
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)
}
```

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:@"%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

- 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 `CKRecor
open func getJokesByCurrentUser(_ reco

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

  let predicate = NSPredicate(format:

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

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) {
    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) {
```

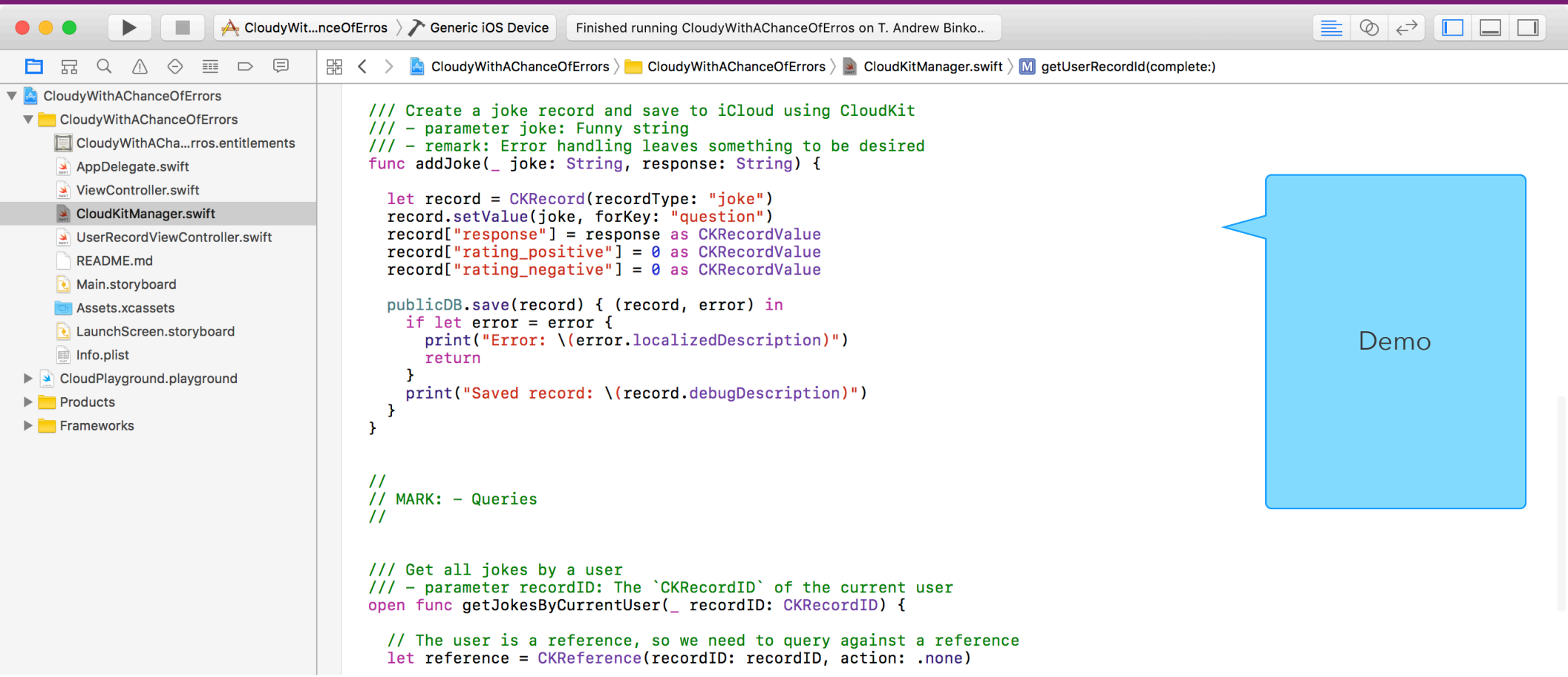
QUERIES

```
// 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

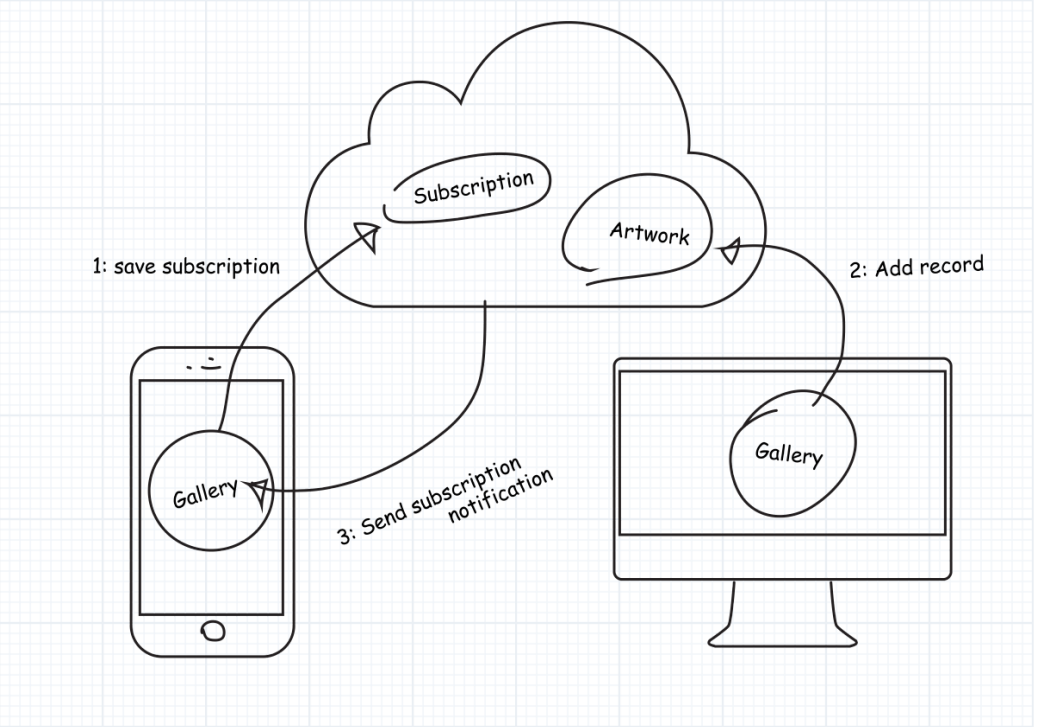


Demo

SUBSCRIPTIONS

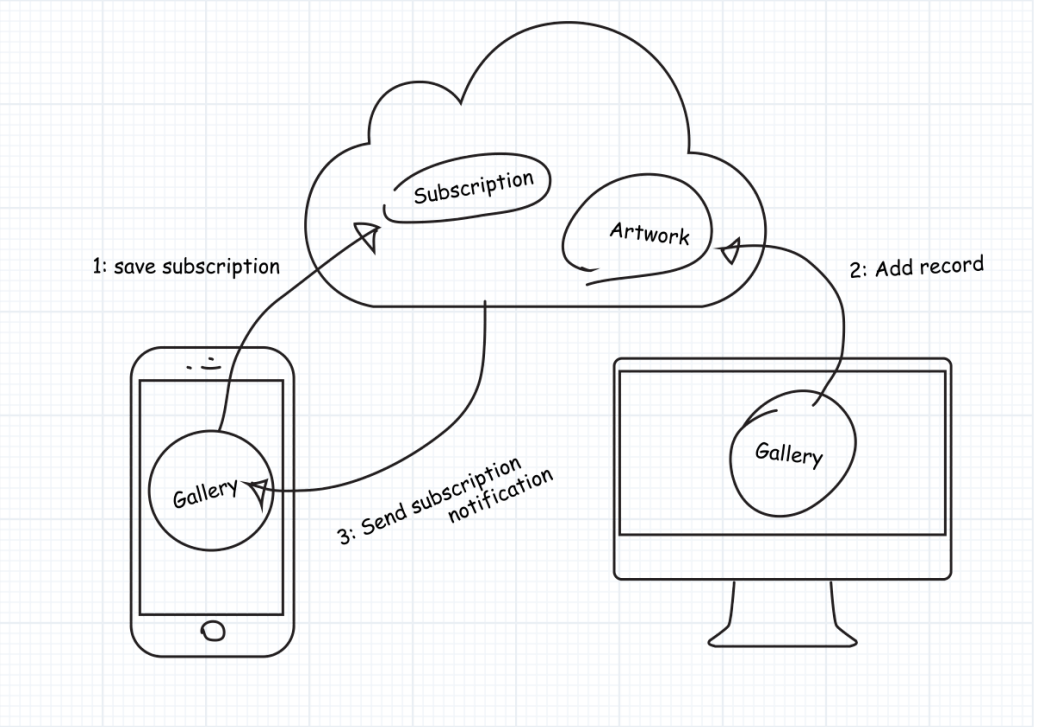
SUBSCRIPTIONS

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



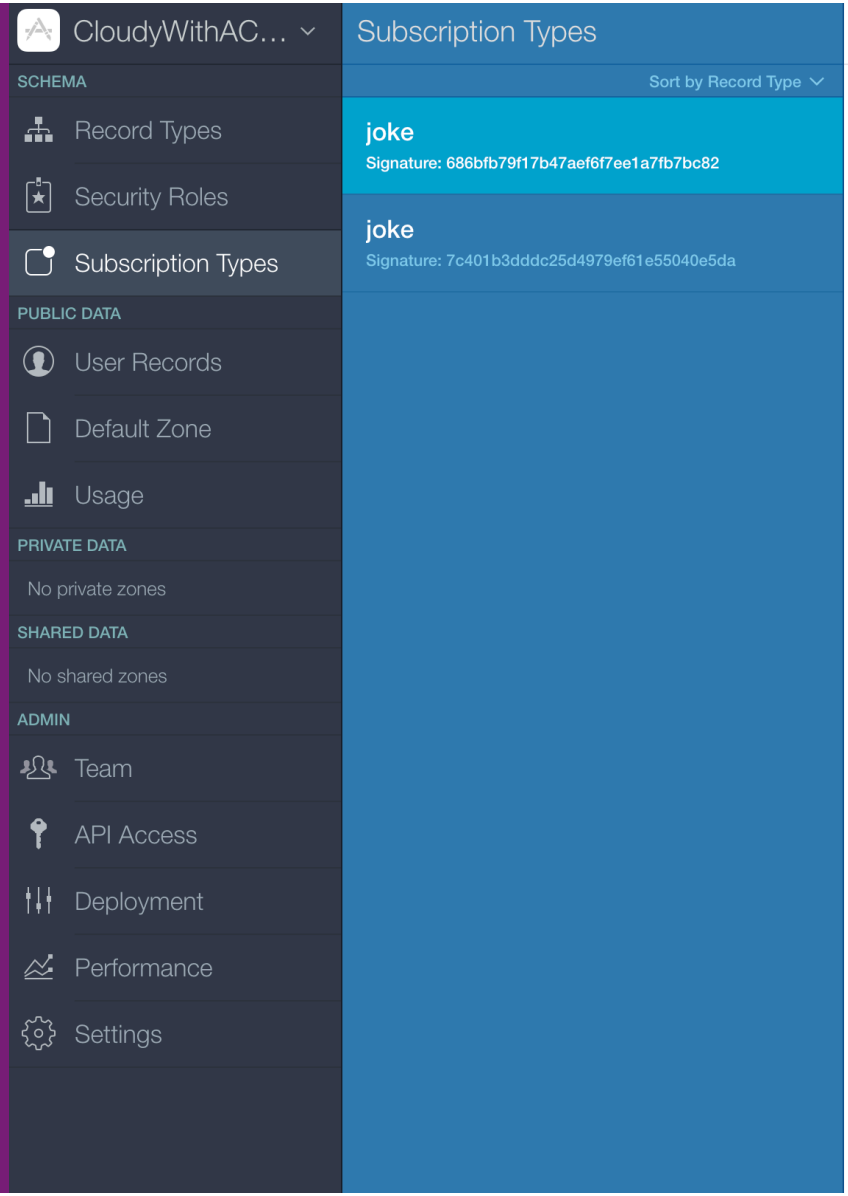
SUBSCRIPTIONS

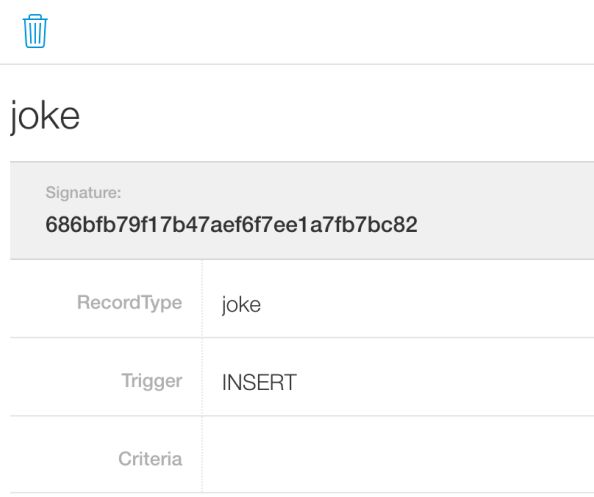
- The server running your query
  - in the background
  - fter every record save
  - push results



SUBSCRIPTIONS

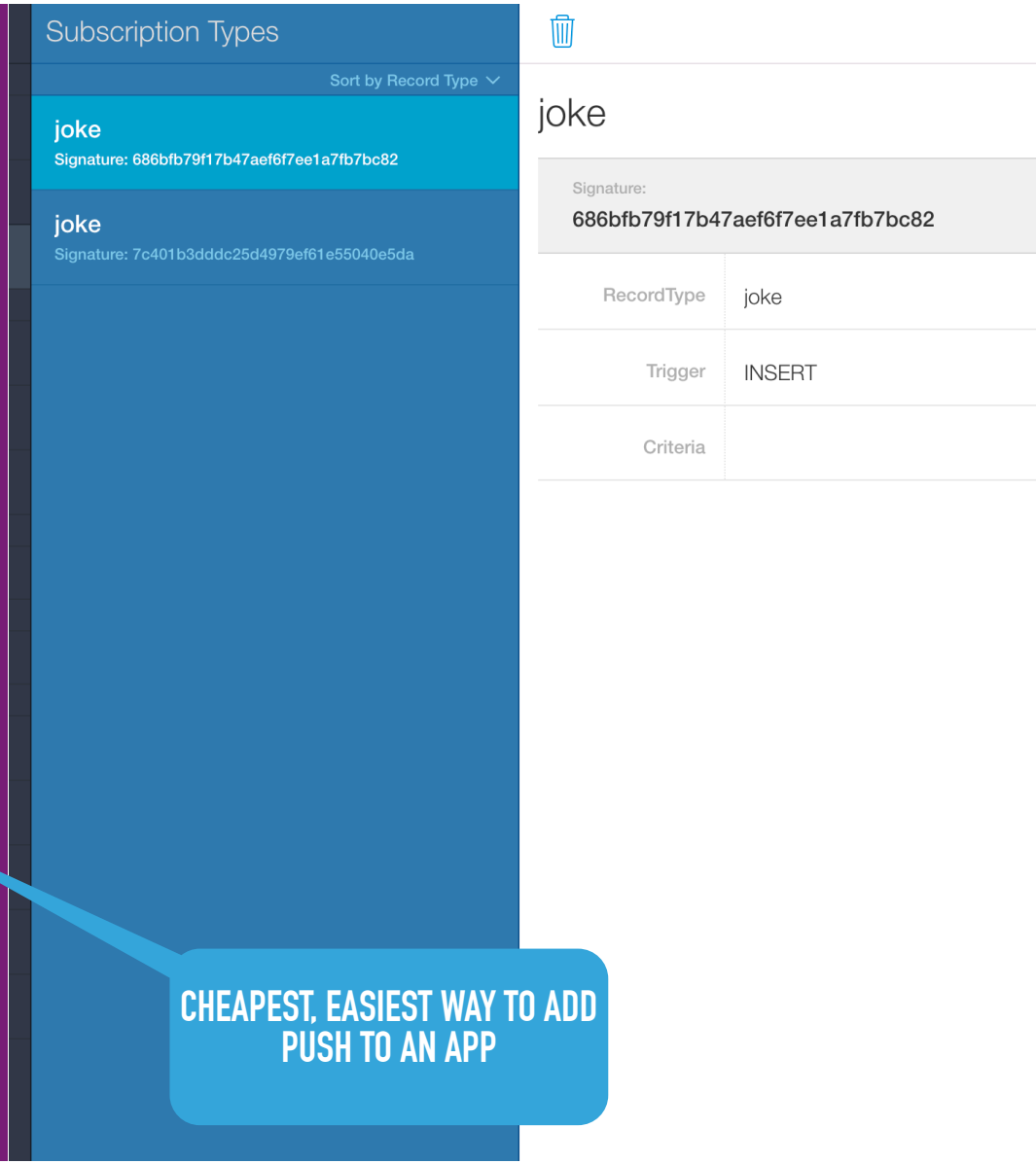
- The server running your query
  - in the background
  - fter every record save
  - push results





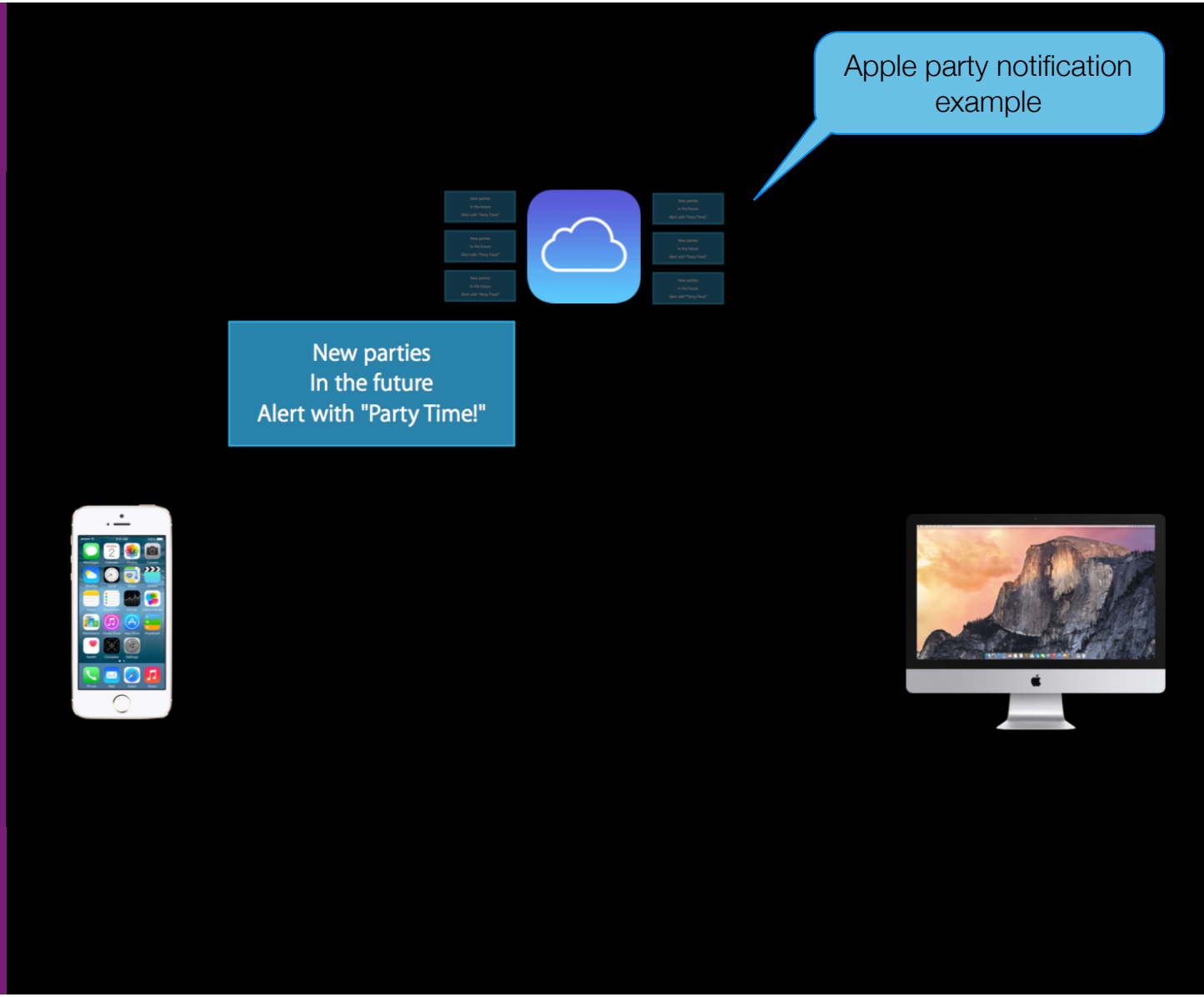
SUBSCRIPTIONS

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



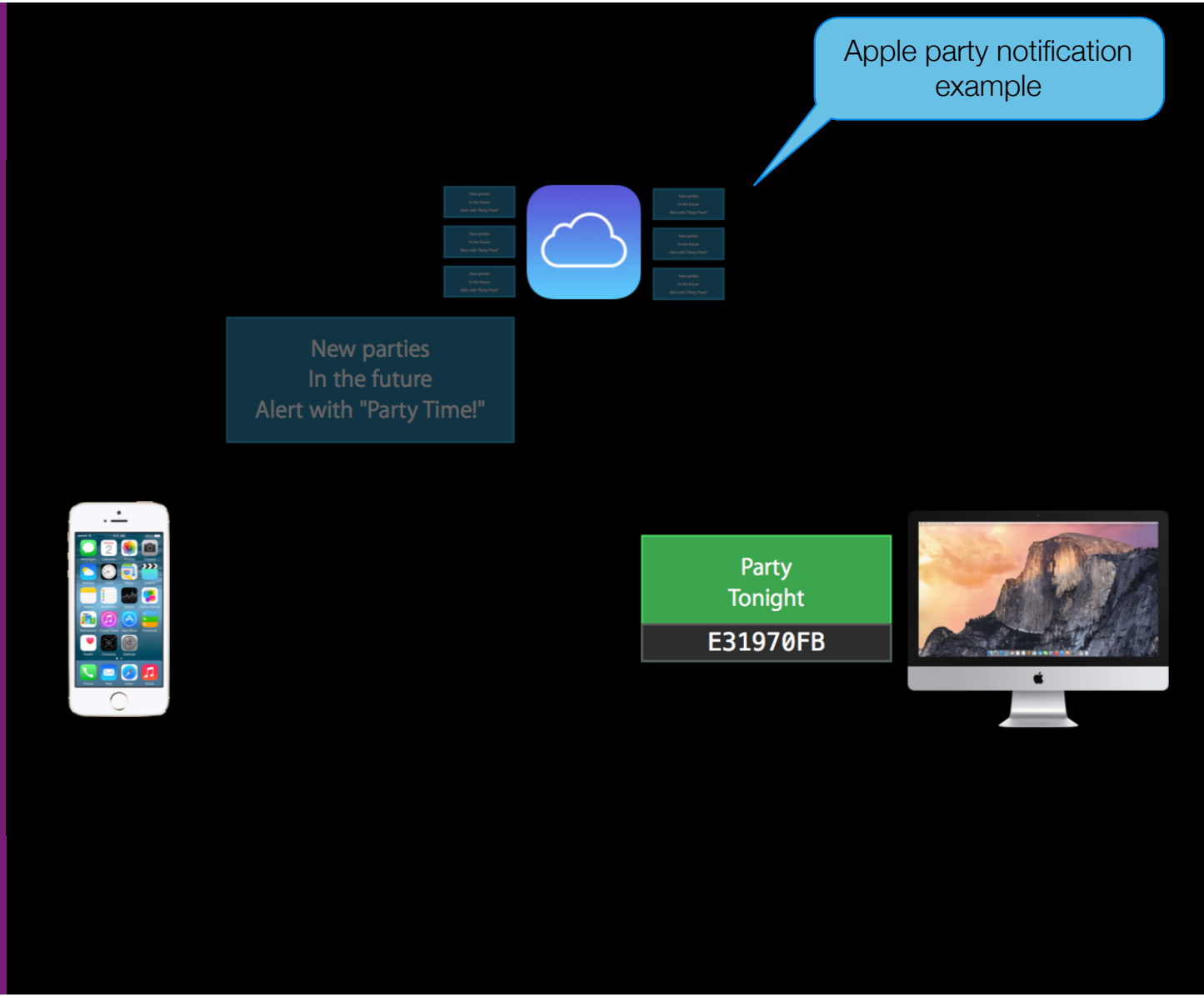
SUBSCRIPTIONS

- Subscription in action



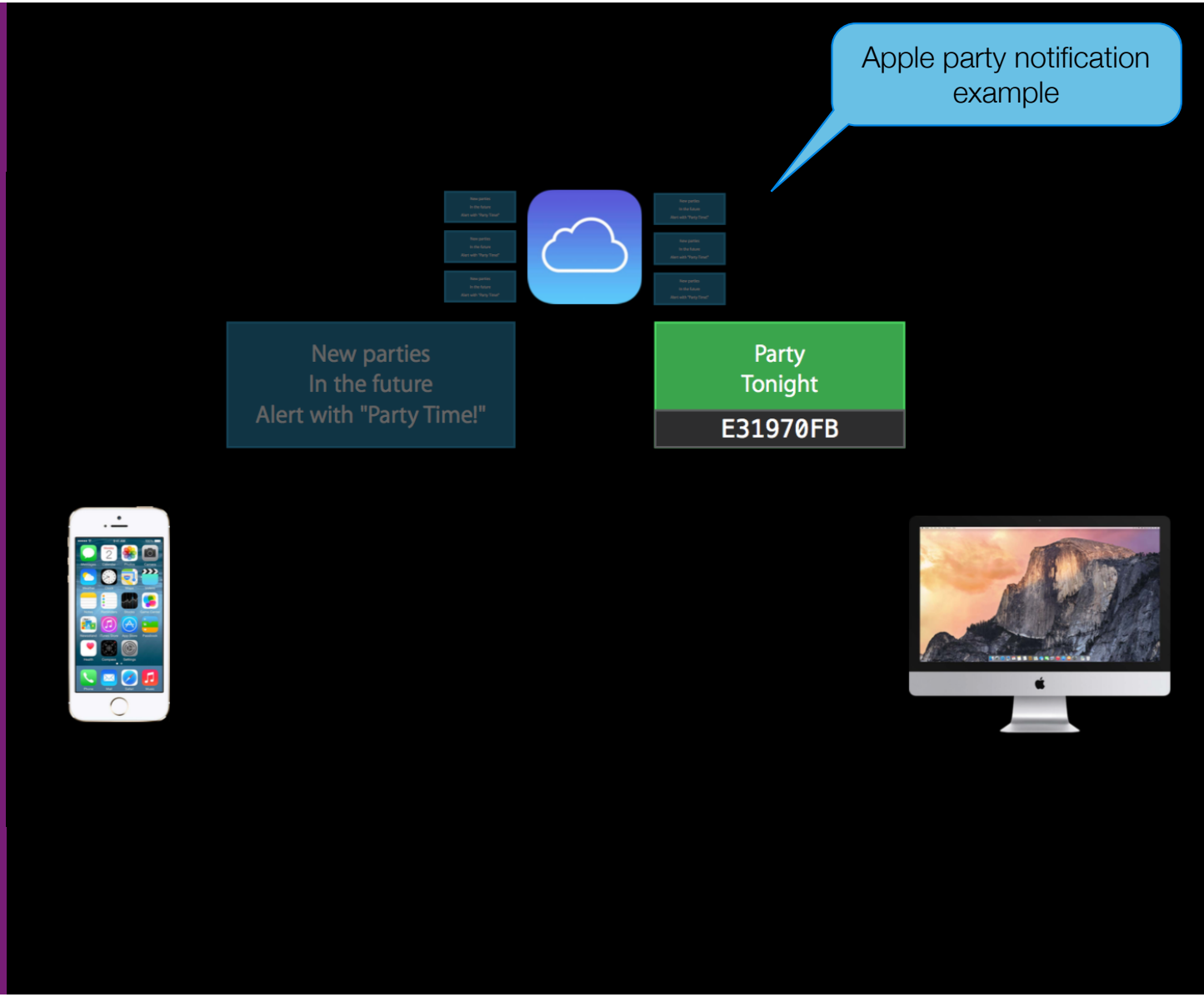
SUBSCRIPTIONS

- Subscription in action



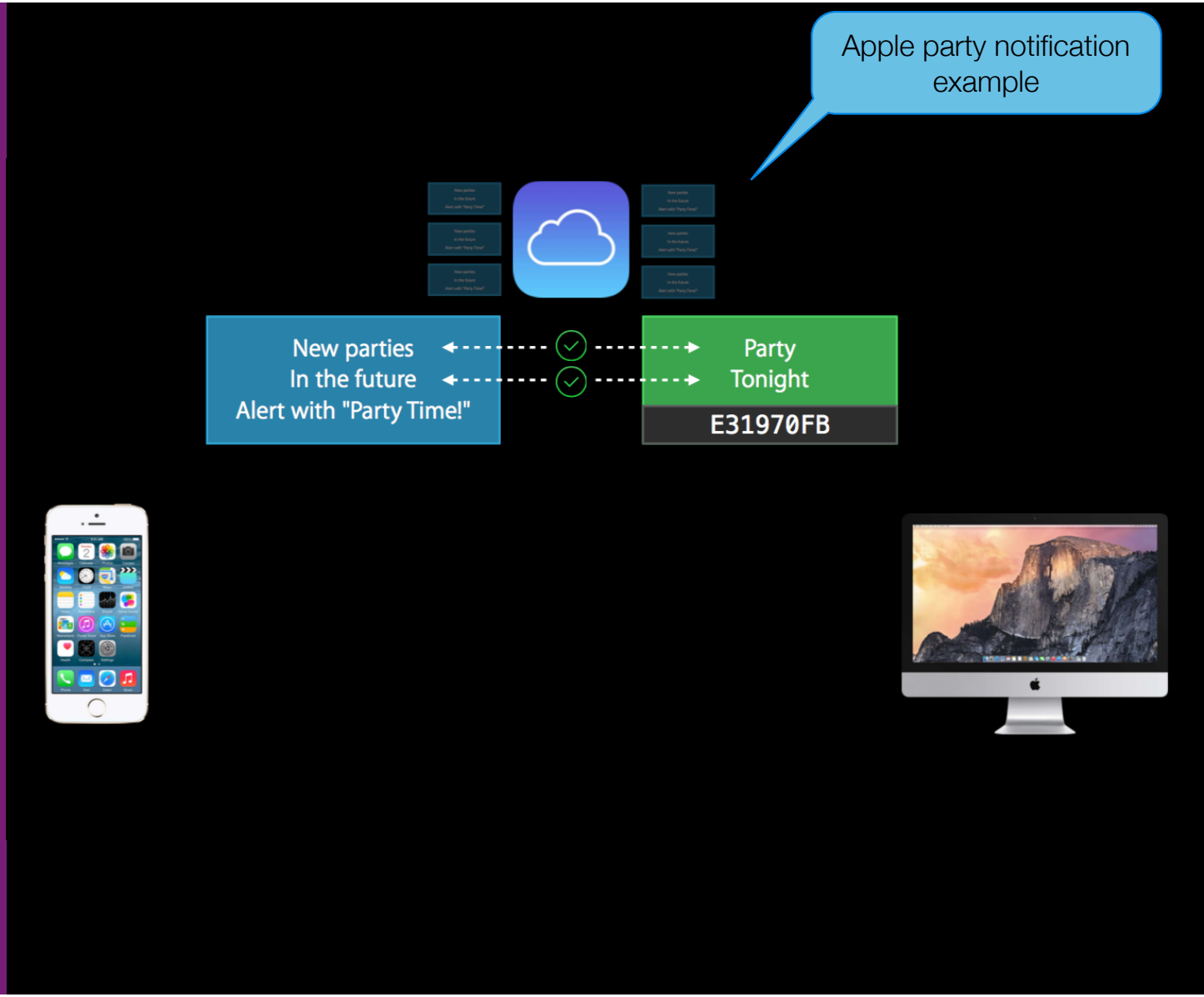
SUBSCRIPTIONS

- Subscription in action



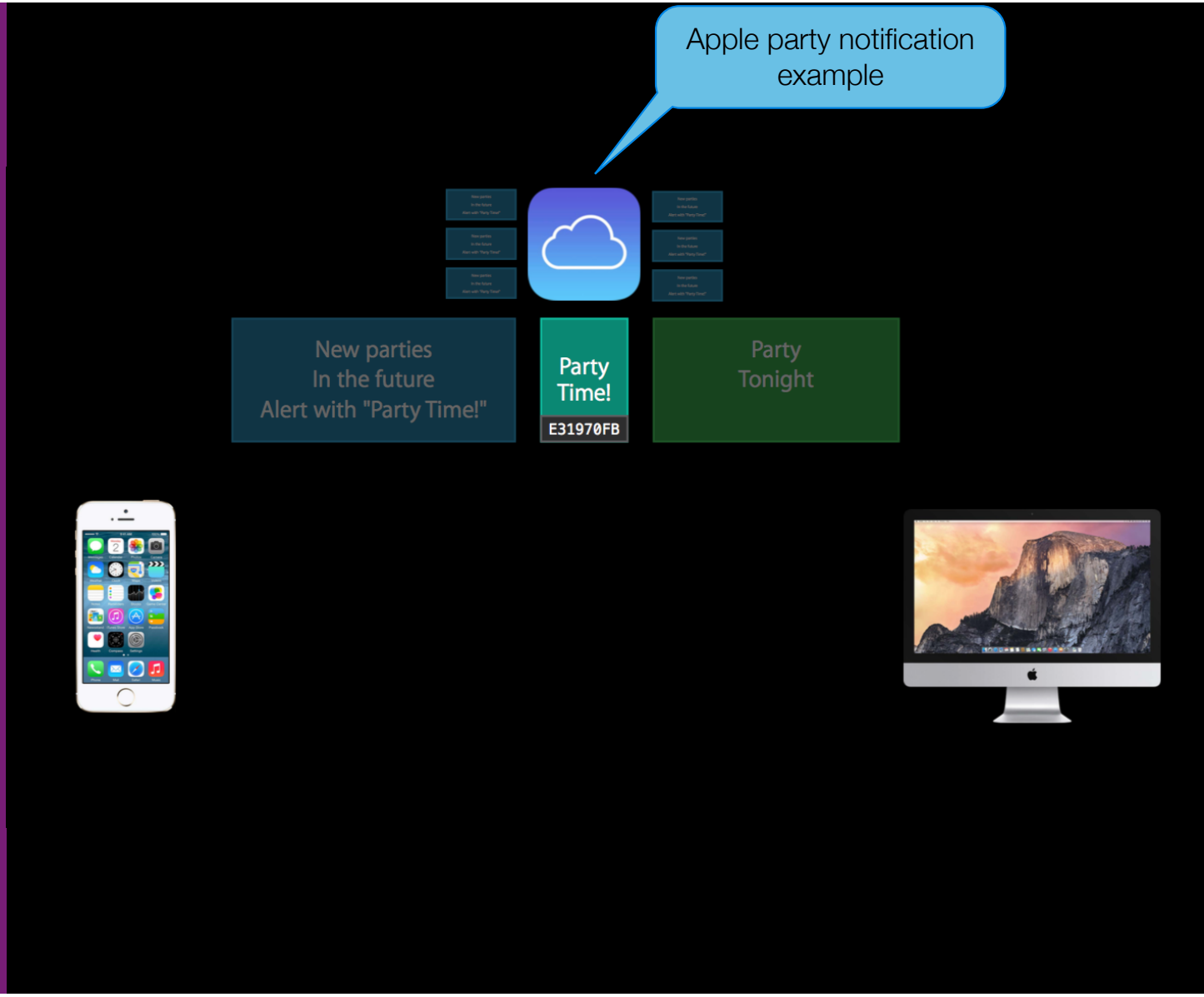
SUBSCRIPTIONS

- Subscription in action



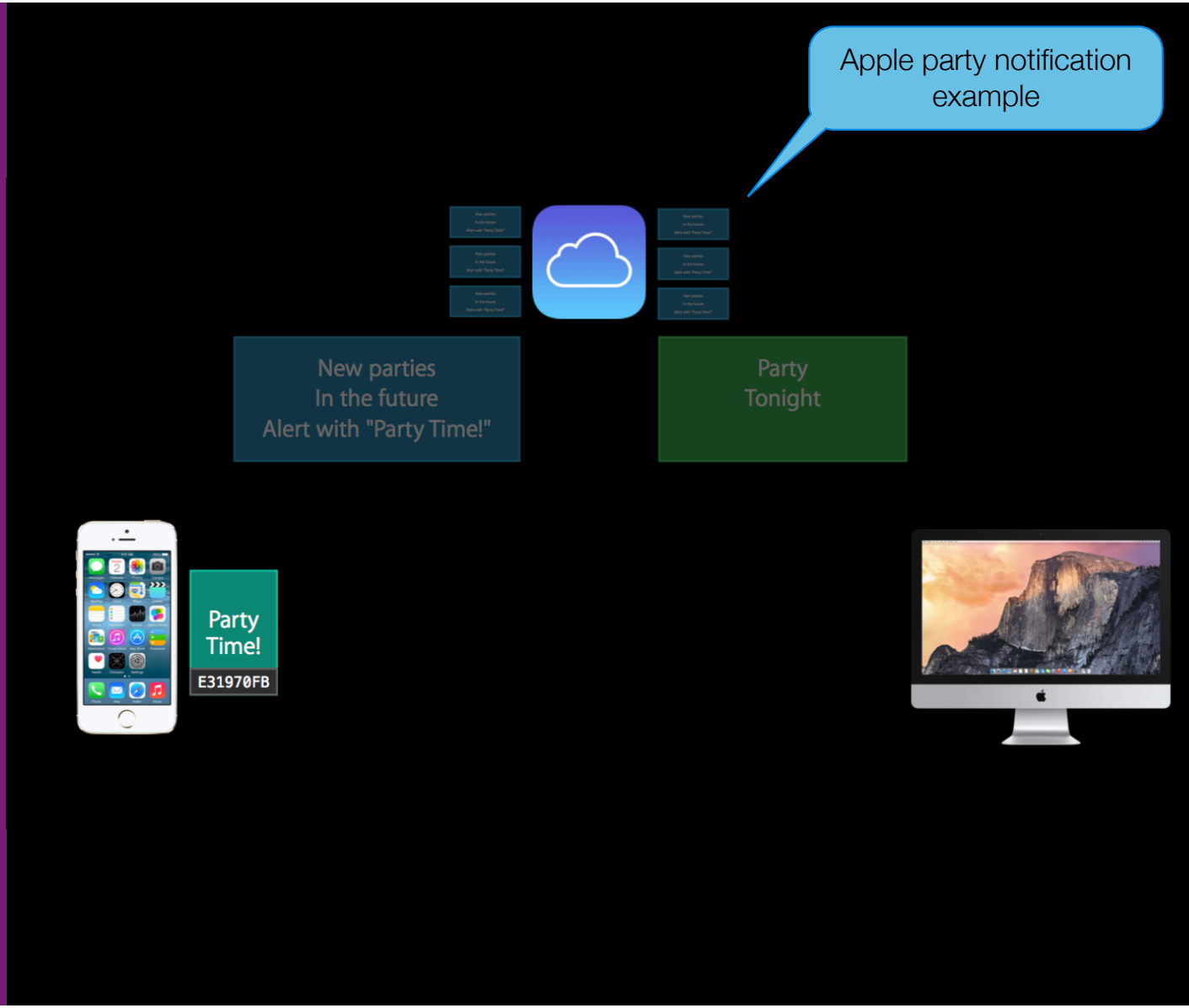
SUBSCRIPTIONS

- Subscription in action



SUBSCRIPTIONS

- Subscription in action



SUBSCRIPTIONS

- Subscriptions 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 you 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

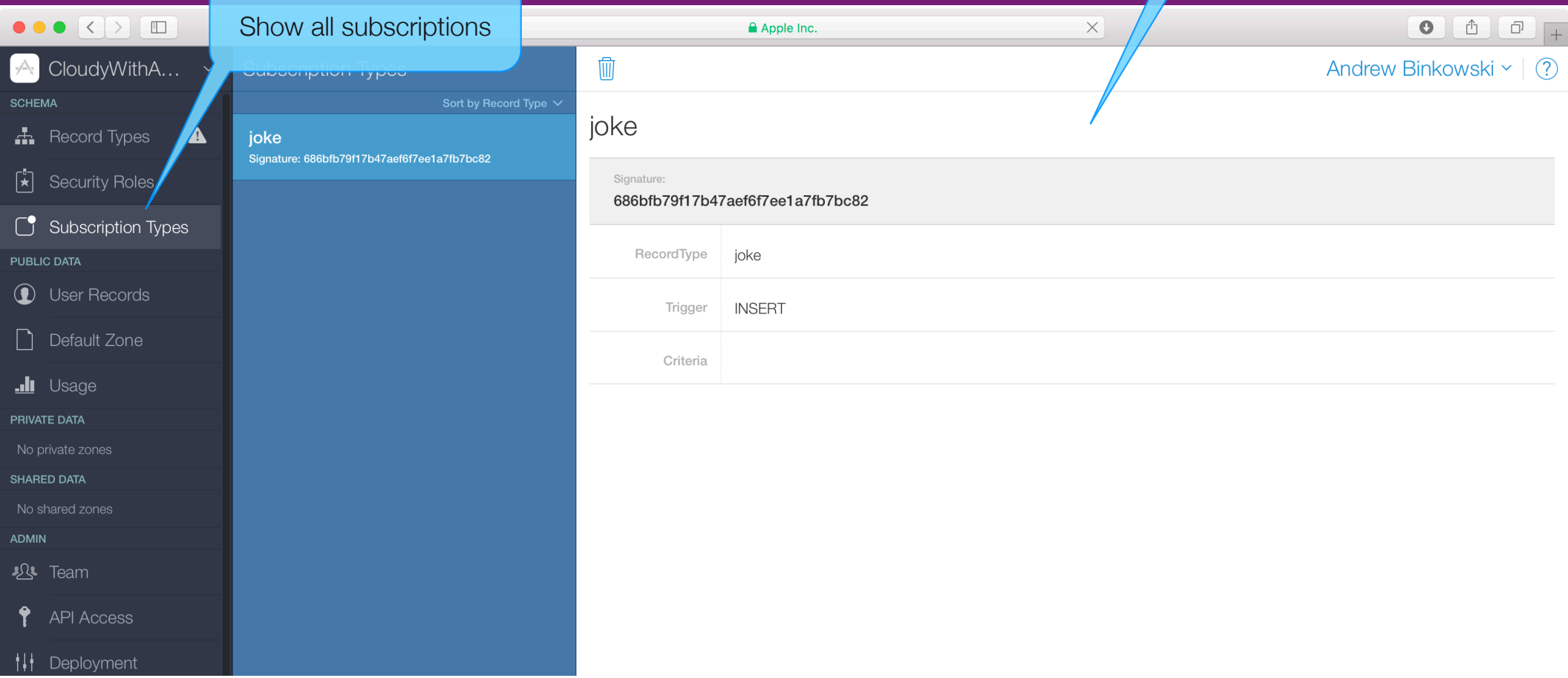
```
func registerSubscriptionsWithIdentifier(_ identifier: String) {

  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")
    }
  }))
}
```

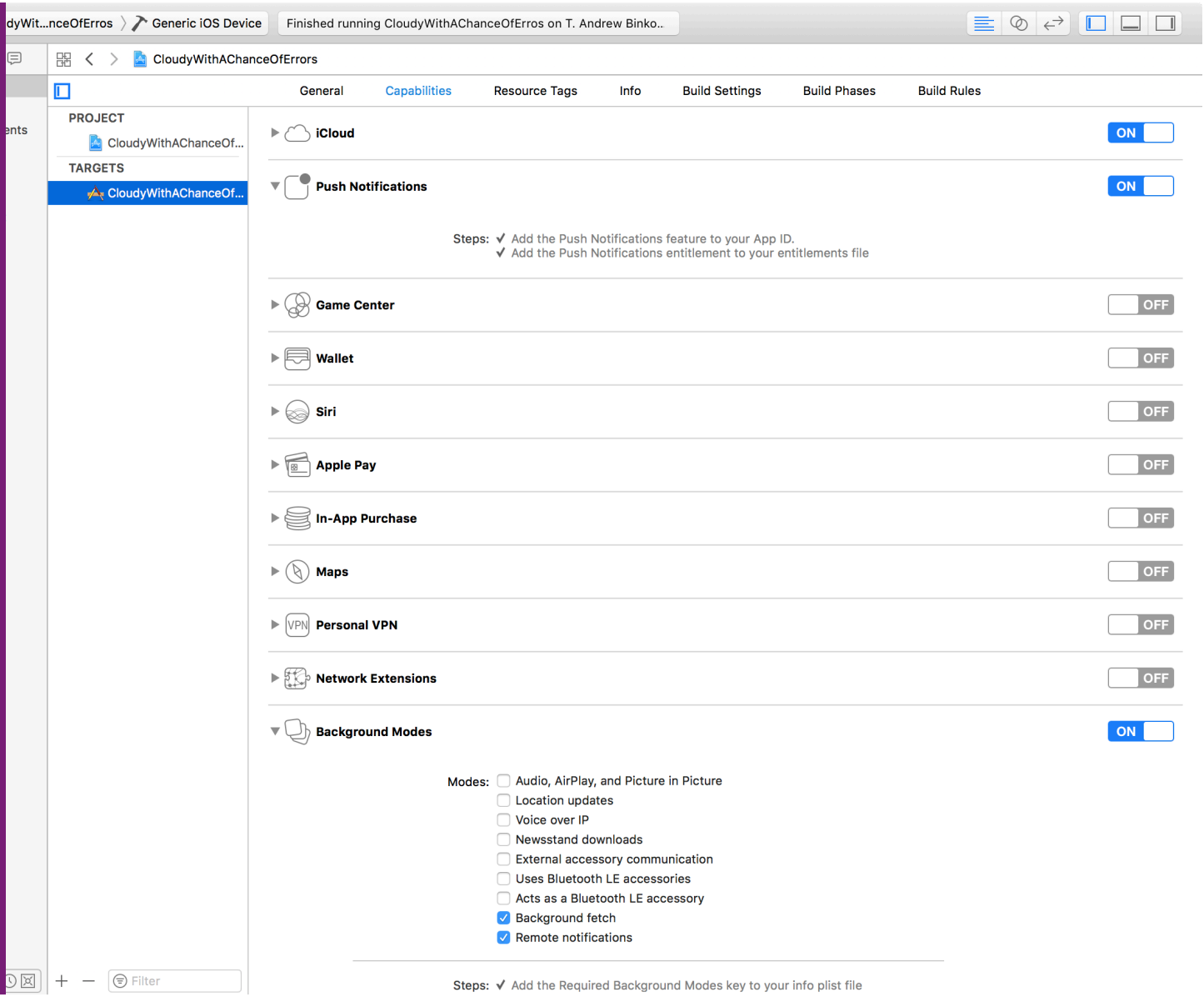
SUBSCRIPTIONS



SETUP YOUR APP TO HANDLE NOTIFICATIONS

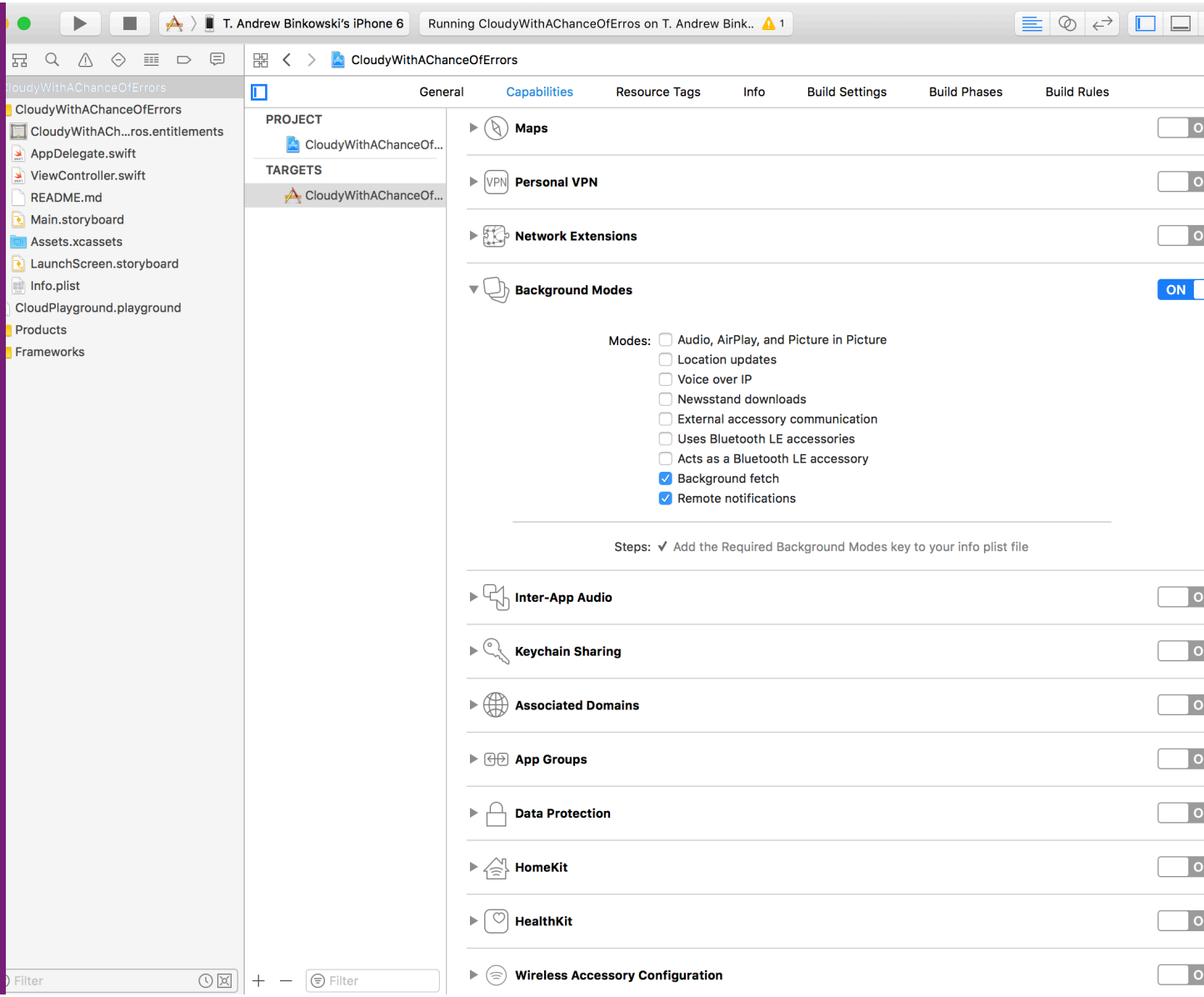
SUBSCRIPTIONS

- Enable push notifications



SUBSCRIPTIONS

- Request background modes



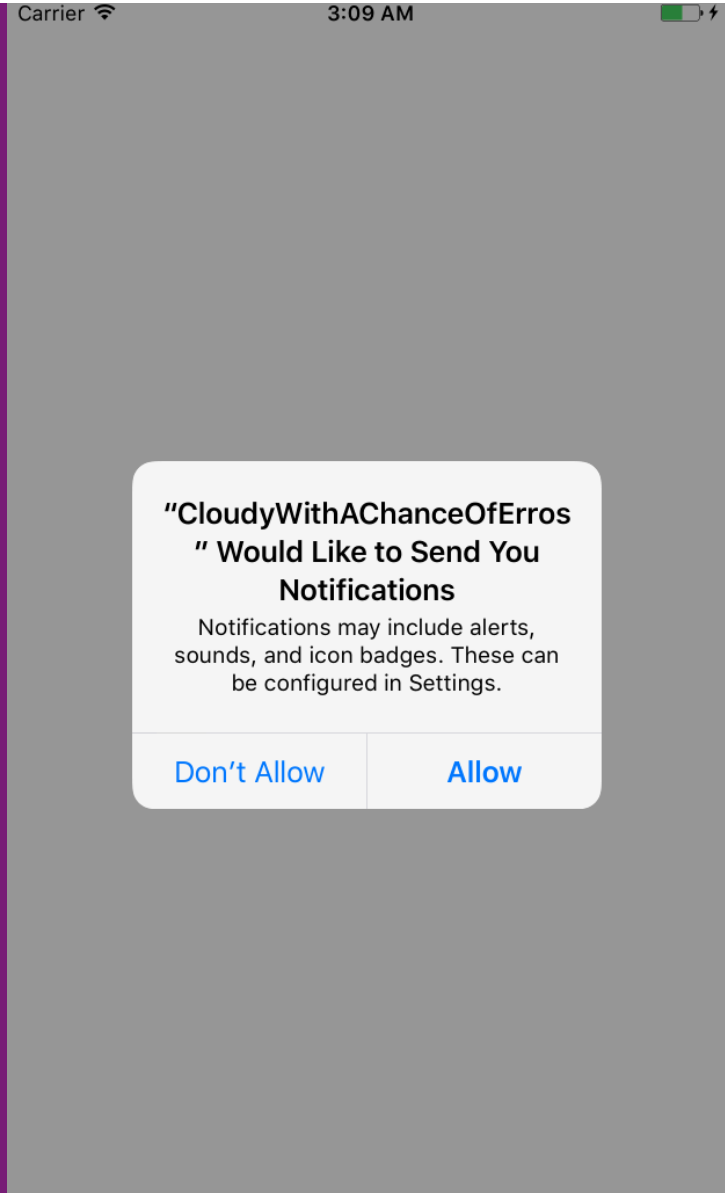
SUBSCRIPTIONS

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



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
  - applicationsDidFinishLaunching
  - 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
//  CloudyWithAChanceOfErros
//
//  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, UNUserNotificationC

  var window: UIWindow?


  func application(_ application: UIApplication,
                   didFinishLaunchingWithOptions launchOptions: [UIApplica

    // Set the notification delegate
    UNUserNotificationCenter.current().requestAuthorization(options: [.ale
      if let error = error {
        print("Error: \(error.localizedDescription)")
      } else {
        application.registerForRemoteNotifications()
      }
    }
    UNUserNotificationCenter.current().delegate = self

    // Register subscriptions
    CloudKitManager.sharedInstance.registerSubscriptionsWithIdentifier("id
    CloudKitManager.sharedInstance.registerSilentSubscriptionsWithIdentifi
    //configureUserNotificationsCenter()

    if let notification = launchOptions?[UIApplicationLaunchOptionsKey.rem
      // 2
      let aps = notification["aps"] as! [String: AnyObject]
      print("APS: \(aps)")
      // 3
      //(window?.rootViewController as? UITabBarController)?.selectedIndex
    }
    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
func 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: "Recieved 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)
func showNotificationNowFromCloudKit(notification: UILocalNotification) {
 //  Create a notofication
 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

JS

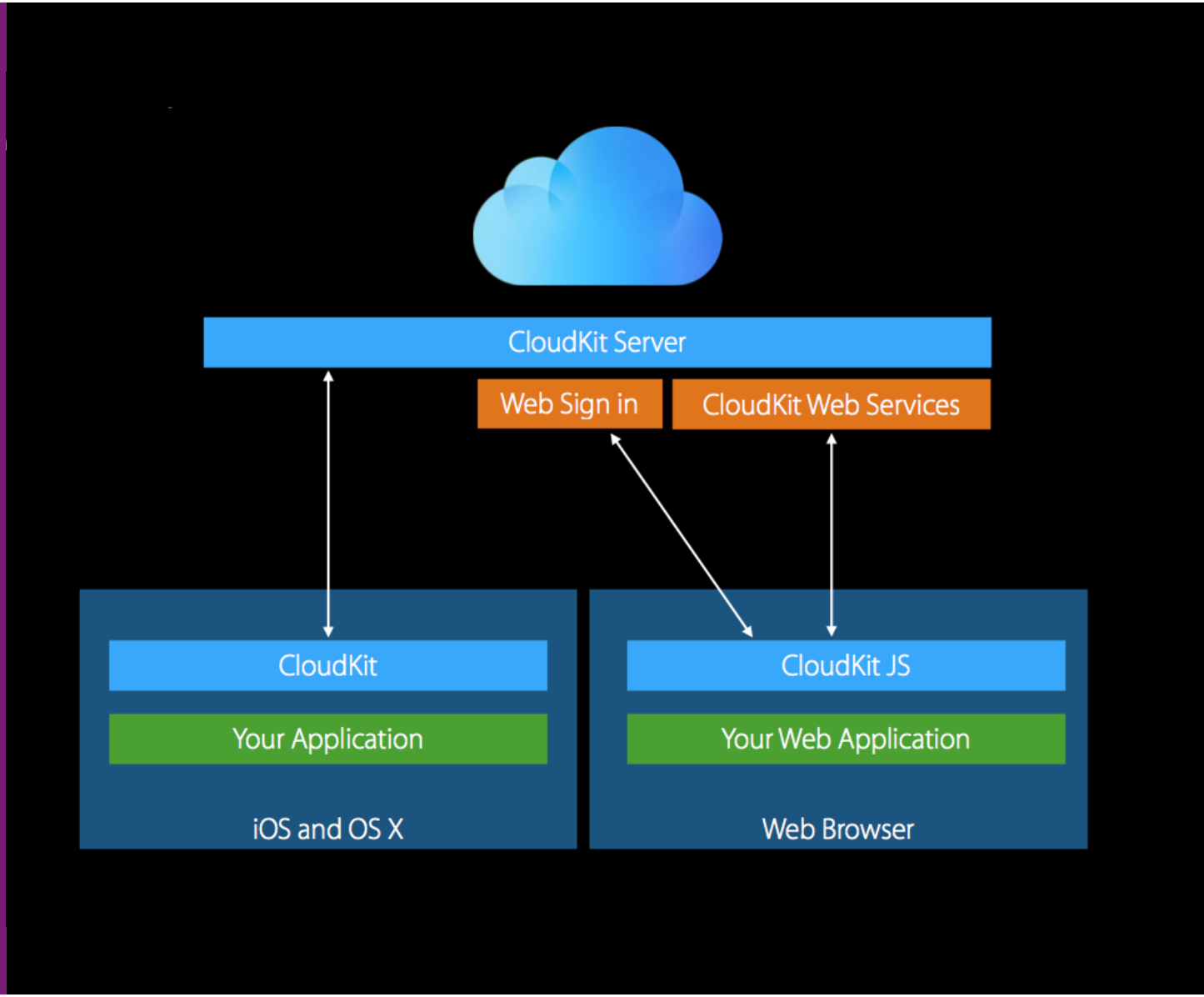
```
<SCRIPT SRC="HTTPS://CDN.APPLE-CLOUDKIT.COM/CK/1/CLOUDKIT.JS" />
```

WEB SERVICE API

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

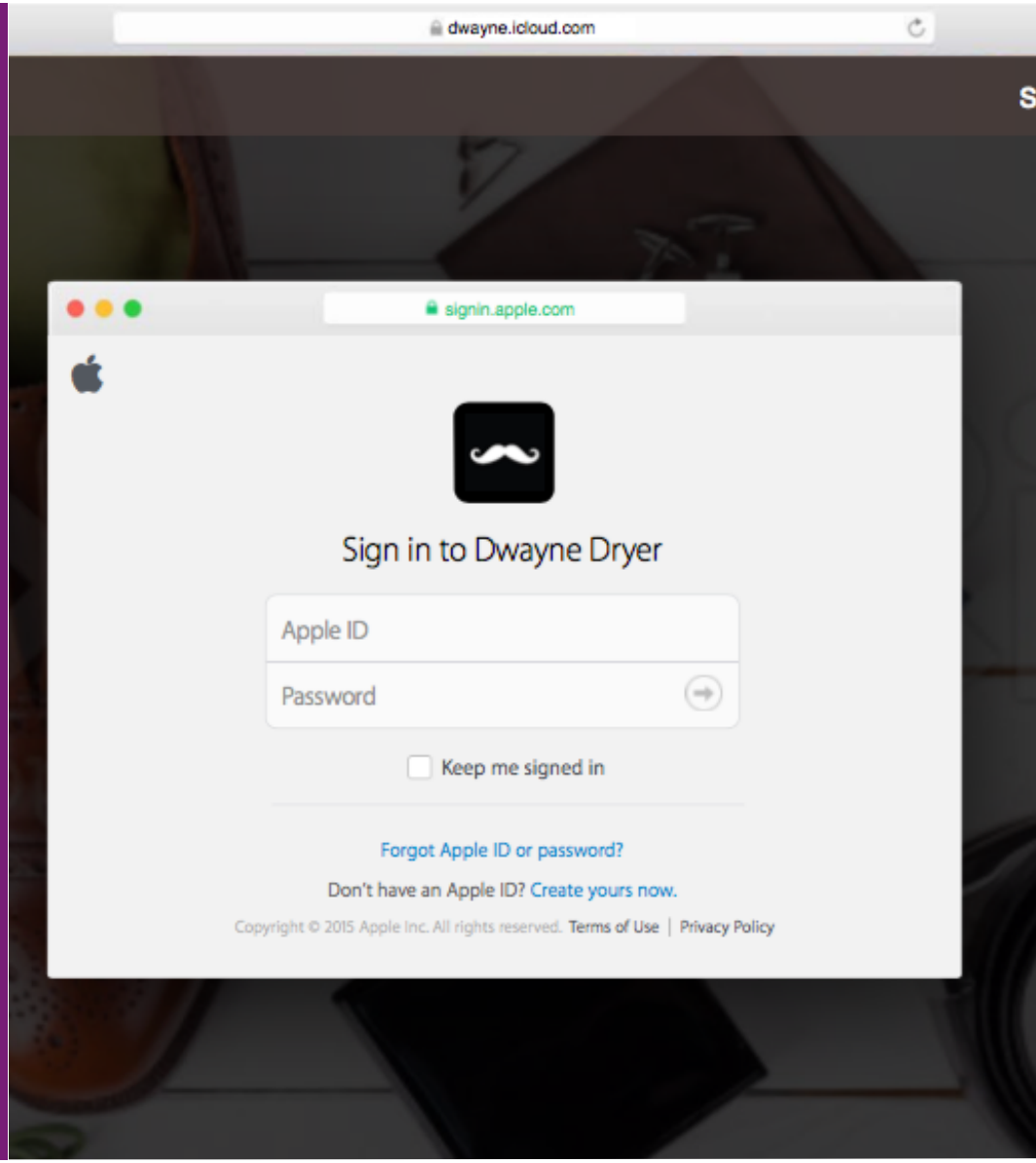
WEB SERVICES

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

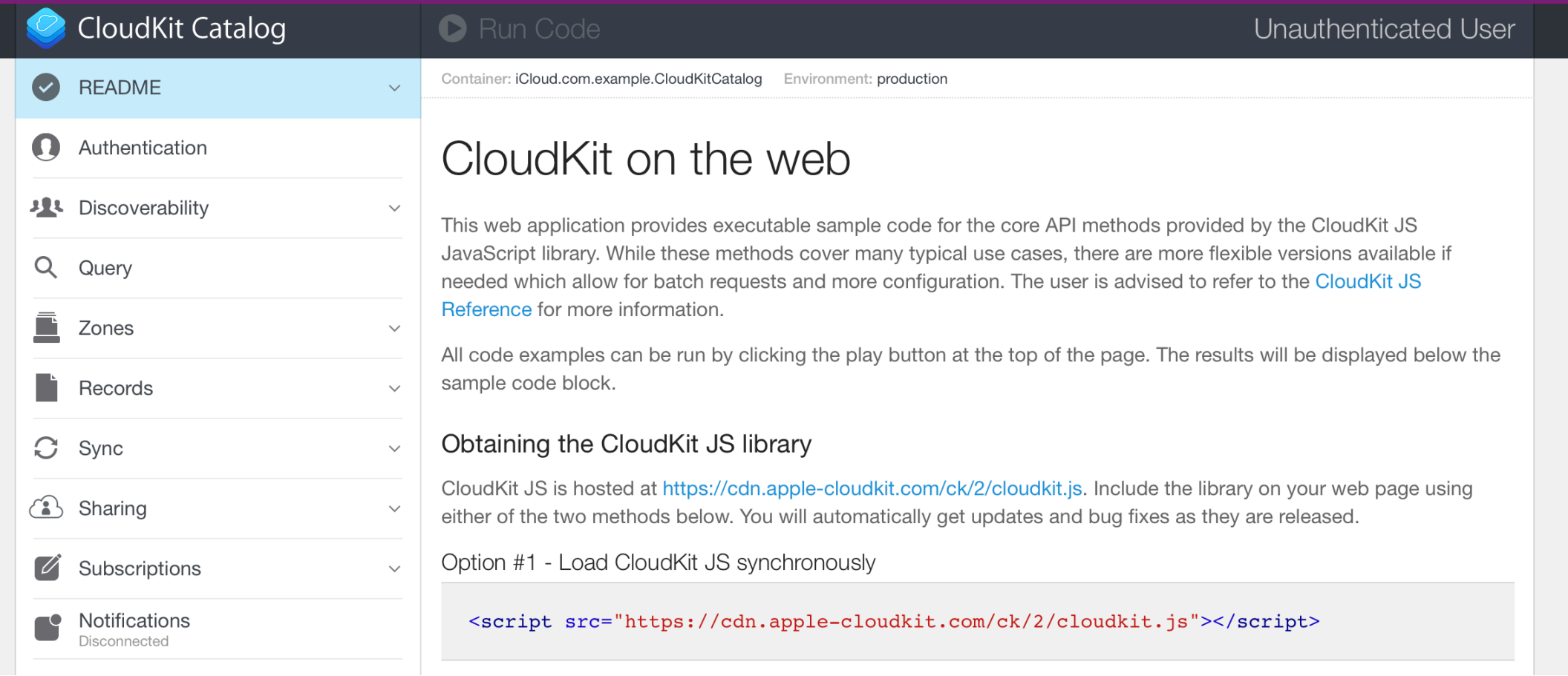


WEB SERVICE API

- 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

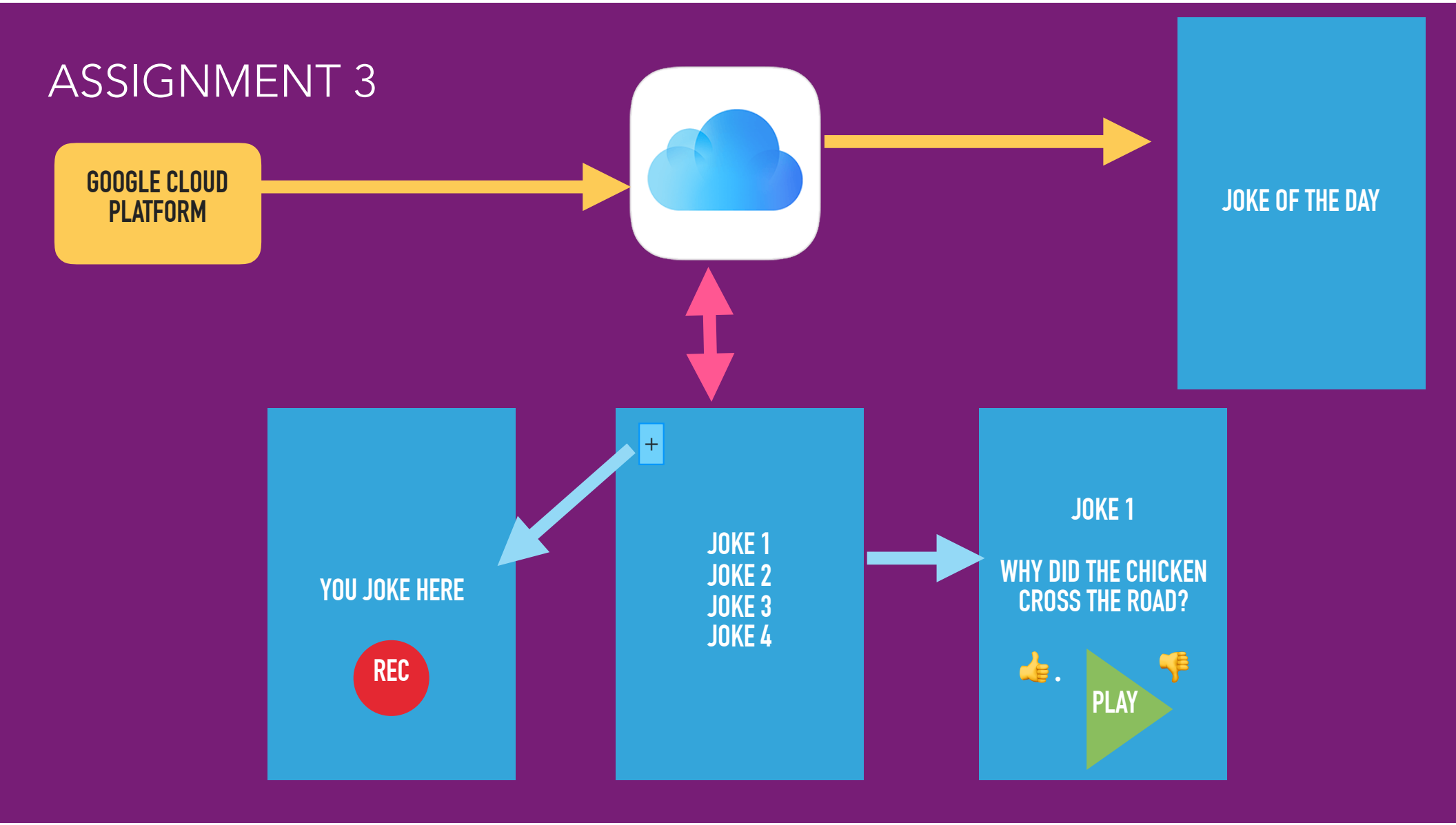
- You're developing exclusively for Apple platforms
- You use it as a partial solution

ASSIGNMENT 4

ASSIGNMENT 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

© 2016 T.A. Binkowski