



THE UNIVERSITY OF  
CHICAGO



MPCS 51033 • SPRING 2017 • SESSION 7

---

# BACKENDS FOR MOBILE APPLICATIONS

# CLASS NEWS

## CLASS NEWS

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

## CLASS NEWS

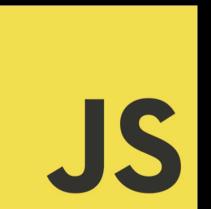
- Week 6 - Assignment 4 (CloudKit)
- Week 7 - Finish assignment 4 and work on case study and final project
- Week 8 - Assignment 5 Assigned (Swift on Server)
- Week 9 - Case Studies in Class; Assignment 5 Due
- Week 10 -
- Week 11 - Final Project Presentations

# CLOUDKIT API

## WEB SERVICE API

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

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



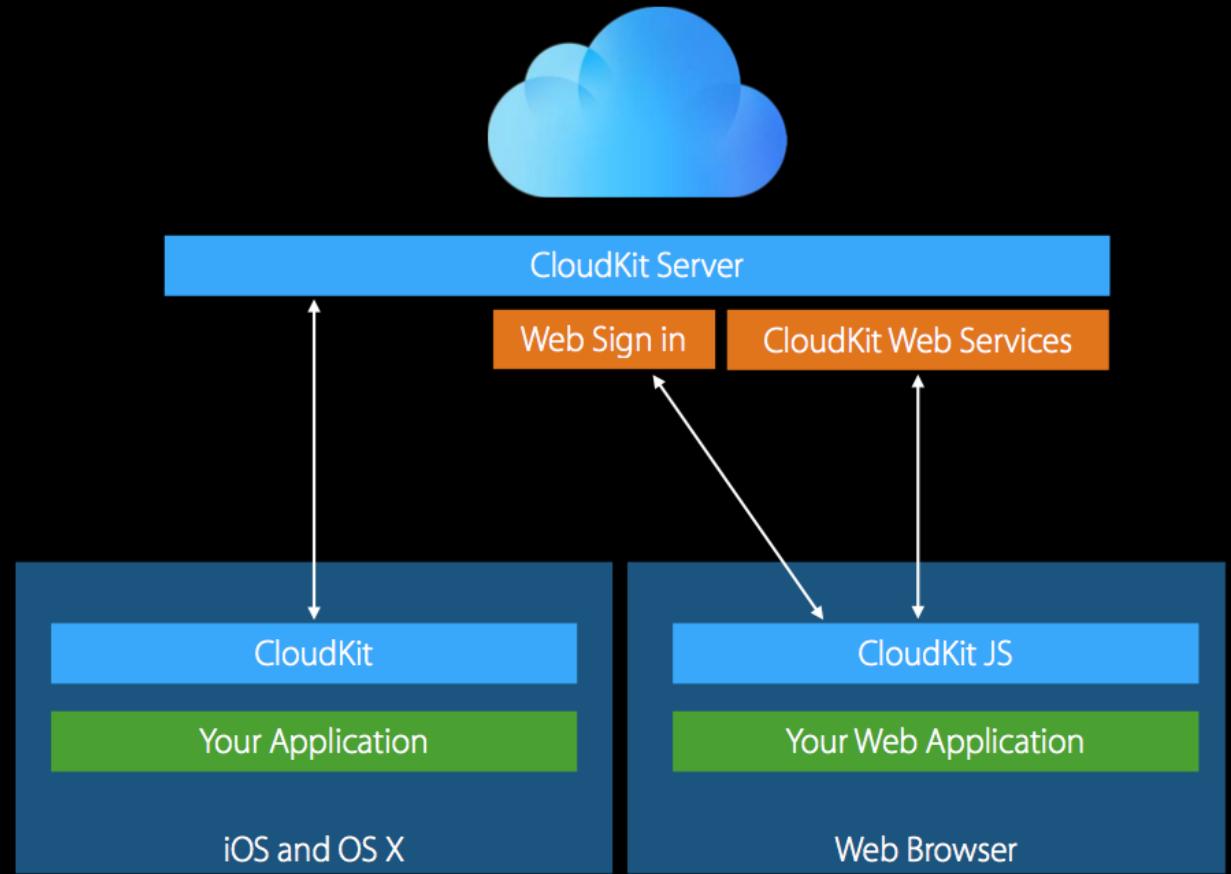
## WEB SERVICE API

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



## WEB SERVICES

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



# CLOUDKIT JS

CloudyWithAC... API Tokens ▾ Andrew Binkowski ▾ ?

SCHEMA

- Record Types
- Security Roles
- Subscription Types

PUBLIC DATA

- User Records
- Default Zone
- Usage

PRIVATE DATA

- No private zones

SHARED DATA

- No shared zones

ADMIN

- Team
- API Access
- Deployment
- Performance

API Tokens ▾ Sort by Name ▾

trash +

“CloudyWithAChanceOfErros”  
has no API Tokens

Add API Token



# CLOUDKIT JS

CloudyWithAC... API Tokens Sort by Name Javascript Token Andrew Binkowski ?

SCHEMA Record Types Security Roles Subscription Types

PUBLIC DATA User Records Default Zone Usage

PRIVATE DATA No private zones

SHARED DATA No shared zones

ADMIN Team API Access Deployment Performance Settings

**Javascript Token**

**Sign In Callback**

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

https:// Custom URL  
Redirects the browser back to this URL after the Apple ID sign in has completed.

**Allowed Origins**

Any domain

Specific domains:  
https:// Add Domain...

**Discoverability**

Request user discoverability at sign in

**Notes**

# CLOUDKIT JS

CloudKit Catalog

- README
- CloudKit on the web
- Server-side CloudKit with node.js

Authentication

- Discoverability
- Query
- Zones
- Records
- Sync
- Sharing
- Subscriptions

Run Code

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

Class: Container

## .setUpAuth()

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

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

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

```
function demoSetUpAuth() {
```

Unauthenticated User

# CLOUDKIT JS

- Token is used to communicate with iCloud

Javascript Token

Created: May 9 2017 7:29 PM      Created By: Andrew Binkowski      Modified: May 9 2017 7:29 PM

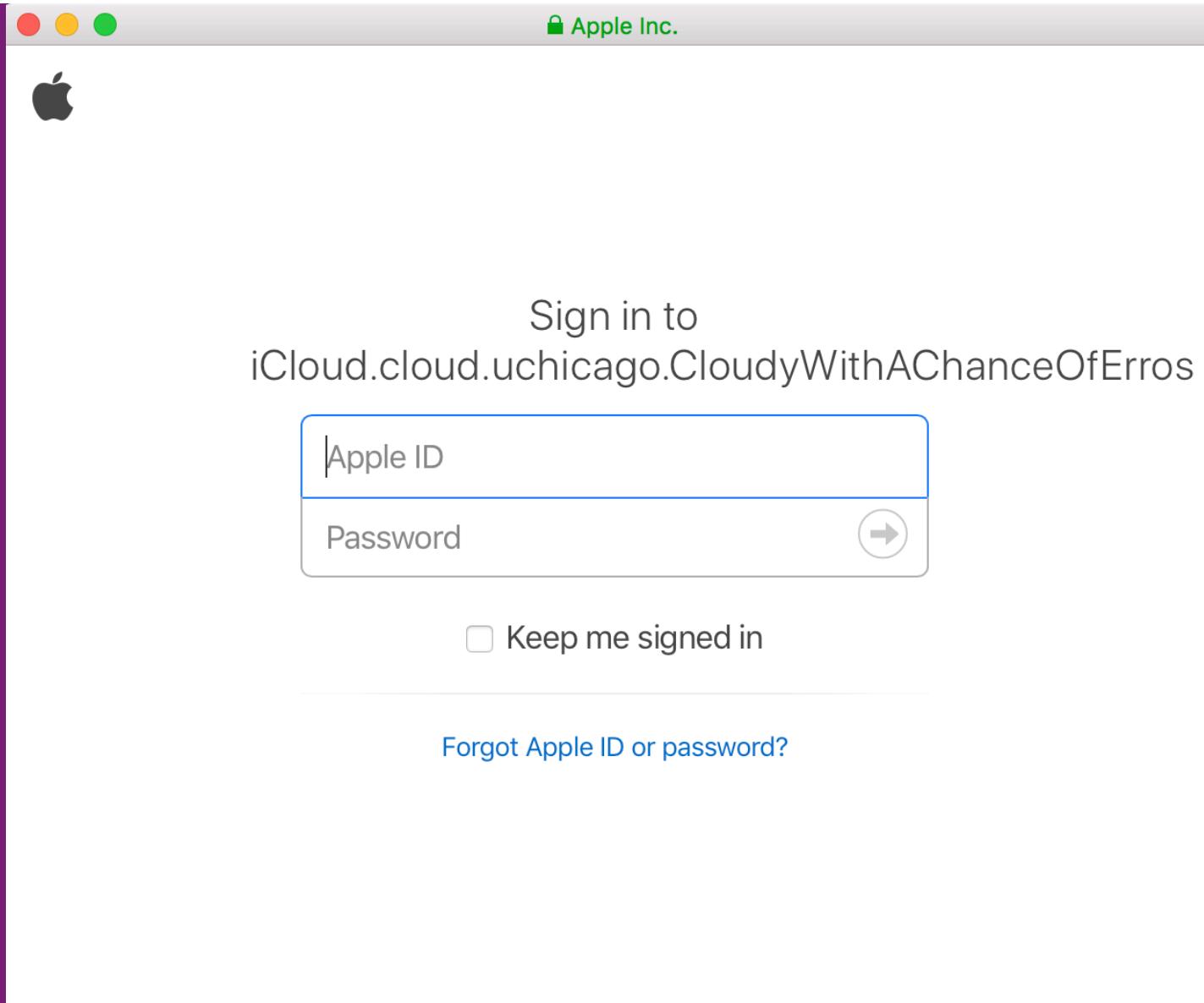
Token: 1b20bfb72a908993605ca3662ef31c120851405732013b7a012a93019ef9bde6

[Test with CloudKit Catalog](#)

Sign In Callback	<input checked="" type="radio"/> postMessage Allows the Apple ID sign in browser window to communicate the a web application using the JavaScript postMessage API.  <input type="radio"/> https://   ▾ Custom URL Redirects the browser back to this URL after the Apple ID sign in h
Allowed Origins	<input checked="" type="radio"/> Any domain  <input type="radio"/> Specific domains: https://   Add Domain...
Discoverability	<input type="checkbox"/> Request user discoverability at sign in

## CLOUDKIT JS

- Authentication happens with Apple provided UI



## CLOUDKIT JS

- Token is used to communicate with iCloud

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



# Look Me Up By Email

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

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

Don't Allow | **Allow**

# CLOUDKIT JS

- Token is used to communicate with iCloud

The screenshot shows the CloudKit Catalog interface. At the top left is the CloudKit logo and the title "CloudKit Catalog". To the right is a "Run Code" button with a play icon. Below the title is a sidebar with several sections: "README" (with a checkmark icon), "Authentication" (which is highlighted with a blue background), "Discoverability", "Query", "Zones", "Records", "Sync", "Sharing", "Subscriptions", and "Notifications" (with a note "Disconnected"). On the right side, under the "Authentication" section, it says "Container: iCloud.cloud.uchicago.CloudyWithAChanceOfErrors Environment: development". Below this, the word "Result" is displayed, followed by the name "Andrew Binkowski". At the bottom right is a "Sign out" button.

# SERVER-TO-SERVER WITH CLOUDKIT

## SERVER-TO-SERVER

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



# SERVER-TO-SERVER

- Apple provides demo

## CloudKit Catalog

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

## Run Code

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

### Server-side CloudKit with node.js

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

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

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

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

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

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

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

#### Installing dependencies

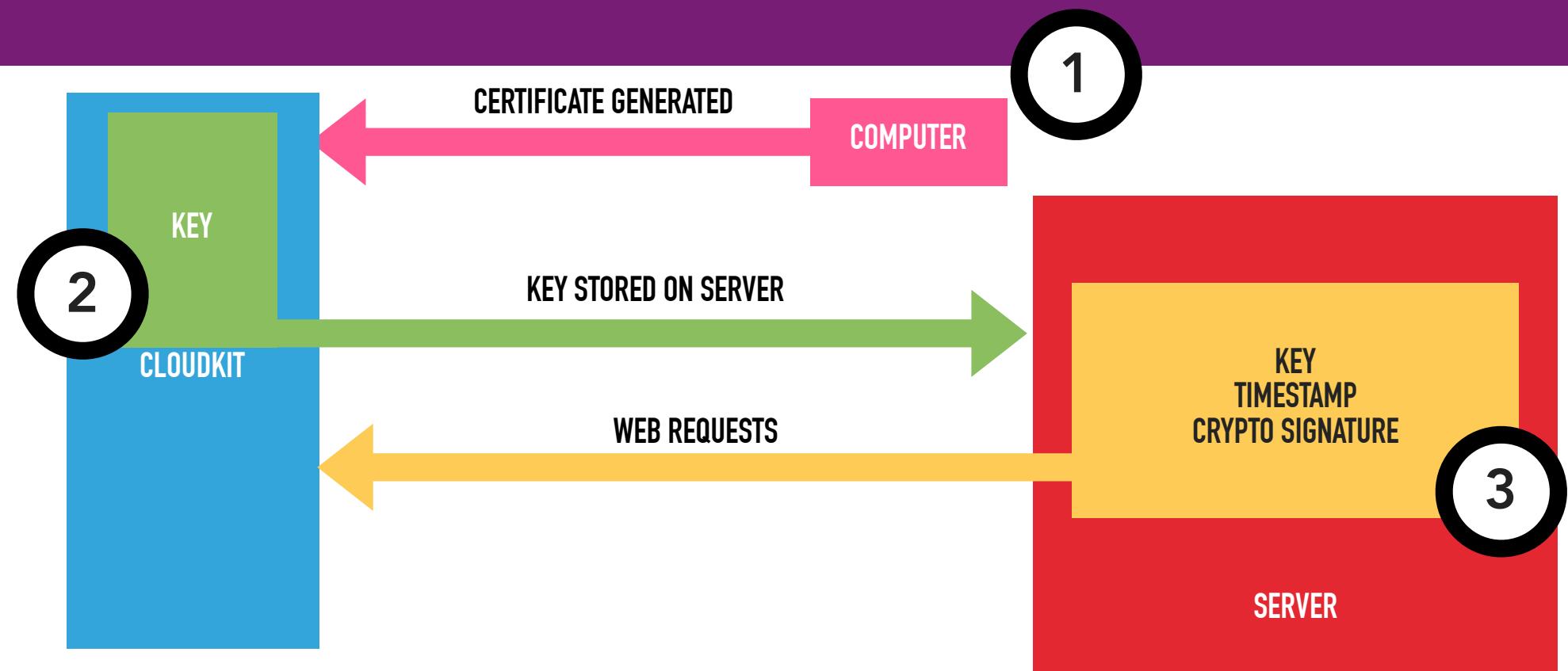
In order to use CloudKit JS server-side you will need a `fetch` implementation such as `node-fetch` which you can install via NPM. You must also download CloudKit JS itself from Apple's CDN.

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

#### Configuring CloudKit JS in a node script

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

## SERVER-TO-SERVER



# SERVER-TO-SERVER KEY

# SERVER-TO-SERVER KEY

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

Guides and Sample Code

CloudKit Web Services Reference

Table of Contents

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

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

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

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

### Creating a Server-to-Server Certificate

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

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

1. Launch Terminal.
2. Enter this command:

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

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

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

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

1. In Terminal, enter this command:

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

The public key appears in the output.

```
read EC key
writing EC key
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEExnKj6w8e3pxjtaOUfaNNjsnXHgWH
nQA3TzMt5P32tK8PjlHzpPm6doaDvGKZcS99YAXj0+u5pe9PtmsBKWTuWA==
-----END PUBLIC KEY-----
```

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

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

# SERVER-TO-SERVER KEY

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

Guides and Sample Code

CloudKit Web Services Reference

Table of Contents

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

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

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

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

### Creating a Server-to-Server Certificate

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

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

1. Launch Terminal.
2. Enter this command:

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

A eckey.pem file appears in the current folder.

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

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

1. In Terminal, enter this command:

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

The public key appears in the output.

```
read EC key
writing EC key
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEExnKj6w8e3pxjtaOUfaNNjsnXHgWH
nQA3TzMt5P32tK8PjLHzpPm6doaDvGKZcS99YAXj0+u5pe9PtmsBKWTuWA==
-----END PUBLIC KEY-----
```

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

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

# SERVER-TO-SERVER KEY

- Protect your key
  - Back it up
  - Don't post on GitHub
- Can be revoked

Guides and Sample Code

CloudKit Web Services Reference

Table of Contents

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

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

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

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

### Creating a Server-to-Server Certificate

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

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

1. Launch Terminal.
2. Enter this command:

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

A eckey.pem file appears in the current folder.

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

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

1. In Terminal, enter this command:

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

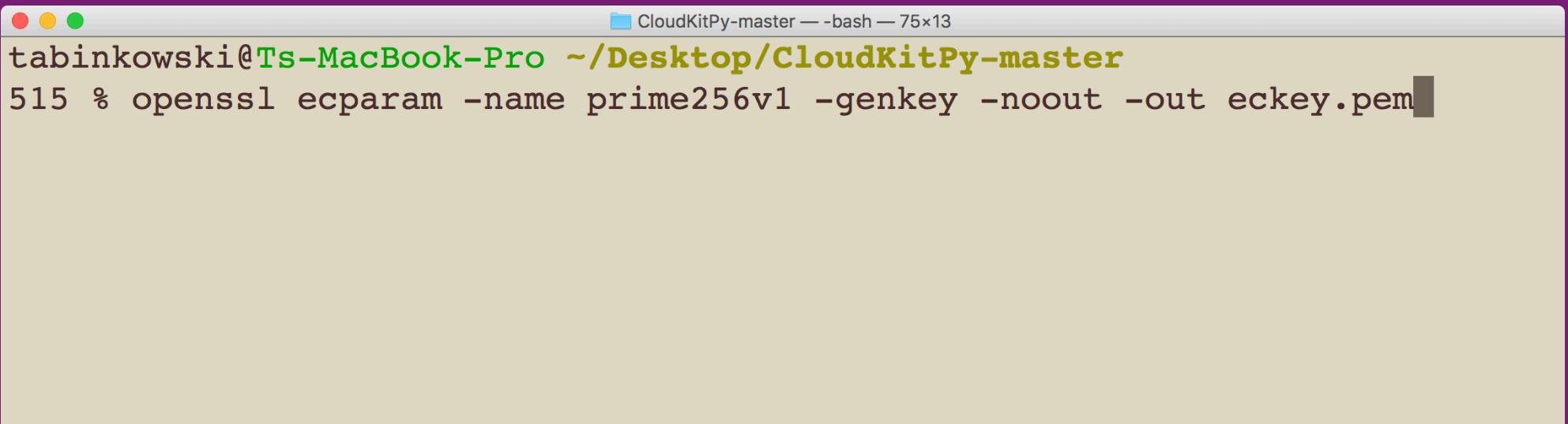
The public key appears in the output.

```
read EC key
writing EC key
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEExnKj6w8e3pxjtaOUfaNNjsnXHgWH
nQA3TzMt5P32tK8PjLHzpPm6doaDvGKZcS99YAXj0+u5pe9PtmsBKWTuWA==
-----END PUBLIC KEY-----
```

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

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

## SERVER-TO-SERVER KEY



A screenshot of a Mac OS X terminal window titled "CloudKitPy-master — bash — 75x13". The window shows the command "openssl ecparam -name prime256v1 -genkey -noout -out eckey.pem" being typed by a user named "tabinkowski". The command is partially completed, with the file name "eckey.pem" visible at the end of the line.

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

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

SERVER-TO-SERVER KEY

# OpenSSL

Cryptography and SSL/TLS Toolkit

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

## Welcome to OpenSSL!

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

### Home

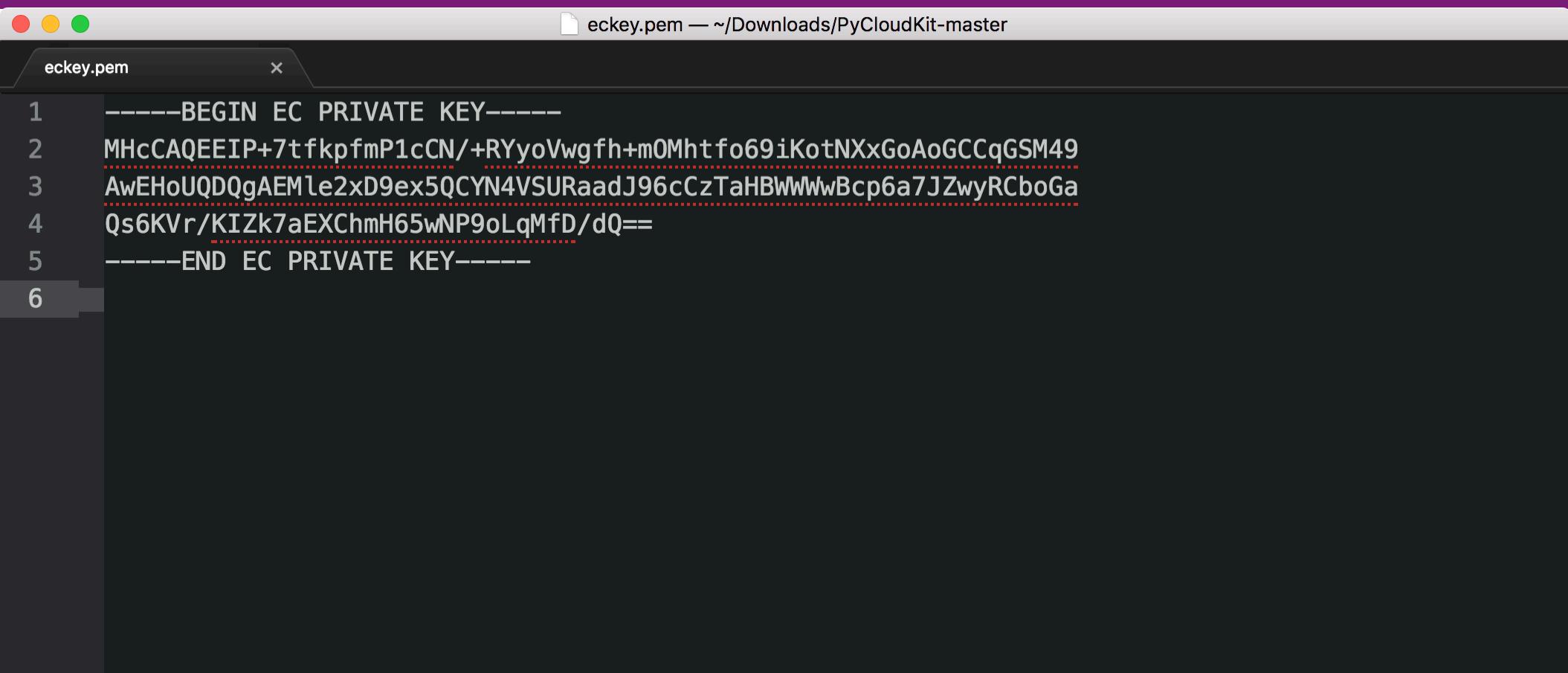
[Downloads: Source code](#)

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

[News: Latest information](#)

[Policies: How we operate](#)

# SERVER-TO-SERVER KEY



A screenshot of a terminal window on a Mac OS X system. The window title is "eckey.pem — ~/Downloads/PyCloudKit-master". The file content is displayed in a monospaced font. The text shows a PEM-formatted EC PRIVATE KEY, starting with "-----BEGIN EC PRIVATE KEY-----" and ending with "-----END EC PRIVATE KEY-----". The key itself is a long string of characters, including letters, numbers, and punctuation marks, separated by red dotted lines.

```
1 -----BEGIN EC PRIVATE KEY-----
2 MHcCAQEEIP+7tfkpfmP1cCN/+RYyoVwgfh+m0Mhtfo69iKotNXxGoAoGCCqGSM49
3 AwEHoUQDQgAEMle2xD9ex5QCYN4VSURaadJ96cCzTaHBWwWwBcp6a7JZwyRCboGa
4 Qs6KVr/KIZk7aEXChmH65wNP9oLqMfD/dQ==
5 -----END EC PRIVATE KEY-----
6
```

## SERVER-TO-SERVER KEY

```
CloudKitPy-master — bash — 75x13
tabinkowski@Ts-MacBook-Pro ~/Desktop/CloudKitPy-master
[523 % openssl ec -in eckey.pem -pubout
read EC key
writing EC key
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEMle2xD9ex5QCYN4VSURaadJ96cCz
TaHBWWWwBcp6a7JZwyRCboGaQs6KVr/KIZk7aEXChmH65wNP9oLqMfD/dQ==
-----END PUBLIC KEY-----
tabinkowski@Ts-MacBook-Pro ~/Desktop/CloudKitPy-master
524 %
```

- Generate the public key
- Enter in CloudKit console

# SERVER-TO-SERVER KEY

CloudyWithAC... ▾ Server-to-Server Keys ▾ Andrew Binkowski ▾ | ?

SCHEMA

- Record Types
- Security Roles
- Subscription Types

PUBLIC DATA

- User Records
- Default Zone
- Usage

PRIVATE DATA

- No private zones

SHARED DATA

- No shared zones

ADMIN

- Team
- API Access

Sort by Name ▾

Create a new key

 “CloudyWithAChanceOfErros” has no Server-to-Server Keys

Add Server-to-Server Key

# SERVER-TO-SERVER KEY

The screenshot shows the CloudKit dashboard interface. On the left, a sidebar lists various management sections: Record Types, Security Roles, Subscription Types, PUBLIC DATA (User Records, Default Zone, Usage), PRIVATE DATA (No private zones), SHARED DATA (No shared zones), and ADMIN (Team, API Access, Deployment). The main area is titled "NewKey" and displays a form for creating a new key. The form includes fields for Created, Created By, Modified, Modified By, and Key ID. Below the form, there is a "Public Key" section with a copy button and an information icon. A large blue callout bubble with the text "Instructions" points to this section. To the right of the callout, there are two code snippets and a note:

Generate a private key in Terminal using:

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

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

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

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

# SERVER-TO-SERVER KEY

The screenshot shows a CloudKit dashboard interface with a sidebar on the left containing various navigation items. The main area displays a key creation screen and a terminal session.

**CloudKit Dashboard Sidebar:**

- Record Types
- Security Roles
- Subscription Types
- PUBLIC DATA**
- User Records
- Default Zone
- Usage
- PRIVATE DATA**
- No private data found
- SHARING
- No sharing found
- ADMINISTRATORS
- No administrators found
- Deployment

**Main View - Key Creation:**

**NewKey**  
Created: May 9 2017 9:41 PM

**Details:**

Created:	Created By:	Modified:	Modified By:
May 9 2017 9:41 PM	Andrew Binkowski	May 9 2017 9:41 PM	Andrew Binkowski

Key ID: ffdb30c23addd822e851e15cdd30ea819251d2fbb9bc59c4803f006dcc900191

**Public Key:** MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEMle2xD9ex5QCYN4VSURaadJ96cCzTaHBWWWwBcp6a7JZwyRCboGaQs6KVr/KIZk7aEXChmH65wNP9oLqMfD/dQ==

**Notes:**

**Terminal Session:**

```
CloudKitPy-master ~$ openssl ec -in eckey.pem -pubout  
read EC key  
writing EC key  
-----BEGIN PUBLIC KEY-----  
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEMle2xD9ex5QCYN4VSURaadJ96cCz  
TaHBWWWwBcp6a7JZwyRCboGaQs6KVr/KIZk7aEXChmH65wNP9oLqMfD/dQ==  
-----END PUBLIC KEY-----  
CloudKitPy-master ~$
```

# WEB SERVICE REQUESTS

# WEB SERVICE REQUESTS

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

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

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

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

*Current date*

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

*Request body*

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

*Web service URL subpath*

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

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

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

2. Compute the ECDSA signature of this message with your private key.
3. Add the following request headers.

```
X-Apple-CloudKit-Request-KeyID: [keyID]
```

```
X-Apple-CloudKit-Request-ISO8601Date: [date]
```

```
X-Apple-CloudKit-Request-SignatureV1: [signature]
```

*keyID*

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

*date*

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

*signature*

The signature created in Step 2.

## WEB SERVICE REQUESTS

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

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

### *keyID*

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

### *date*

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

### *signature*

The signature created in Step 2.

# WEB SERVICE REQUESTS

Query

URL:

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

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

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

# WEB SERVICE REQUESTS

Added new endpoints:

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

Updated endpoints to support sharing records:

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

- API Endpoints for CloudKit

# QUERY

# WEB SERVICE REQUESTS

- API for fetching records

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

You can fetch records using a query.

### Path

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

### Parameters

#### *path*

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

#### *version*

The protocol version—currently, 1.

#### *container*

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

#### *environment*

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

#### *database*

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

##### `public`

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

##### `private`

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

##### `shared`

The database that contains records shared with the current user.

# WEB SERVICE REQUESTS

## Request

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

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

- POST request consists of a JSON dictionary

# WEB SERVICE REQUESTS

- Example query

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

# WEB SERVICE REQUESTS

- Example query

## Record Operation Dictionary

The operation dictionary keys are:

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

## Operation Type Values

The possible values for the `operationType` key are:

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

# WEB SERVICE REQUESTS

## Response

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

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

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

# WEB SERVICE REQUESTS

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

## Data Size Limits

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

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

# WEB SERVICE REQUESTS

Query

URL:

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

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

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

# MODIFY RECORDS

# WEB SERVICE REQUESTS

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

## Path

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

## Parameters

### *path*

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

### *version*

The protocol version—currently, 1.

### *container*

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

### *environment*

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

### *database*

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

#### *public*

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

#### *private*

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

#### *shared*

The database that contains records shared with the current user.

# WEB SERVICE REQUESTS

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

- POST request consists of a JSON dictionary

## WEB SERVICE REQUESTS

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

Your custom fields

Omit recordName and it will be automatically generated

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

## WEB SERVICE REQUESTS

Add a record

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

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

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

# WEB SERVICE REQUESTS

## Response

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

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

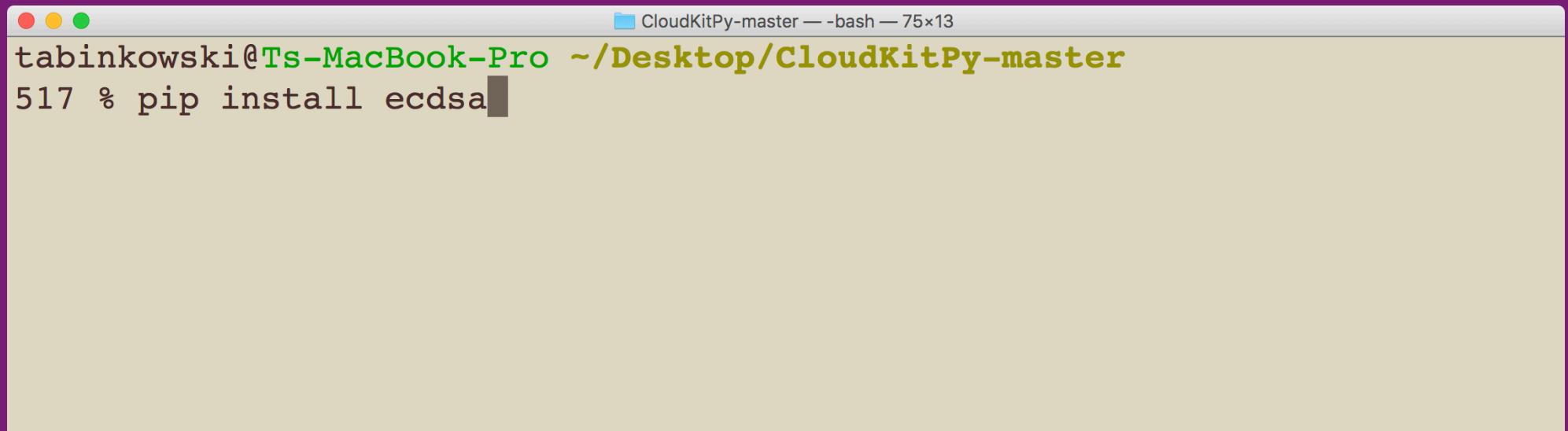
- Response is a dictionary with more information than you probably need

# WEB SERVICE REQUESTS

- Response dictionary

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

## SERVER-TO-SERVER KEY



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

- Elliptic Curve Digital Signature Algorithm
- Cryptography approach for public/private key agreement

# BREAK TIME



# SERVER-TO-SERVER DEMO

# SERVER-TO-SERVER KEY

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

lionheart / requests-cloudkit Watch ▾

Code Issues 1 Pull requests 0 Pulse Graphs

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

python rest-api cloudkit python-library

19 commits 1 branch 0 releases

Branch: master New pull request Create new file Upload file

dlo committed on GitHub Update README.rst

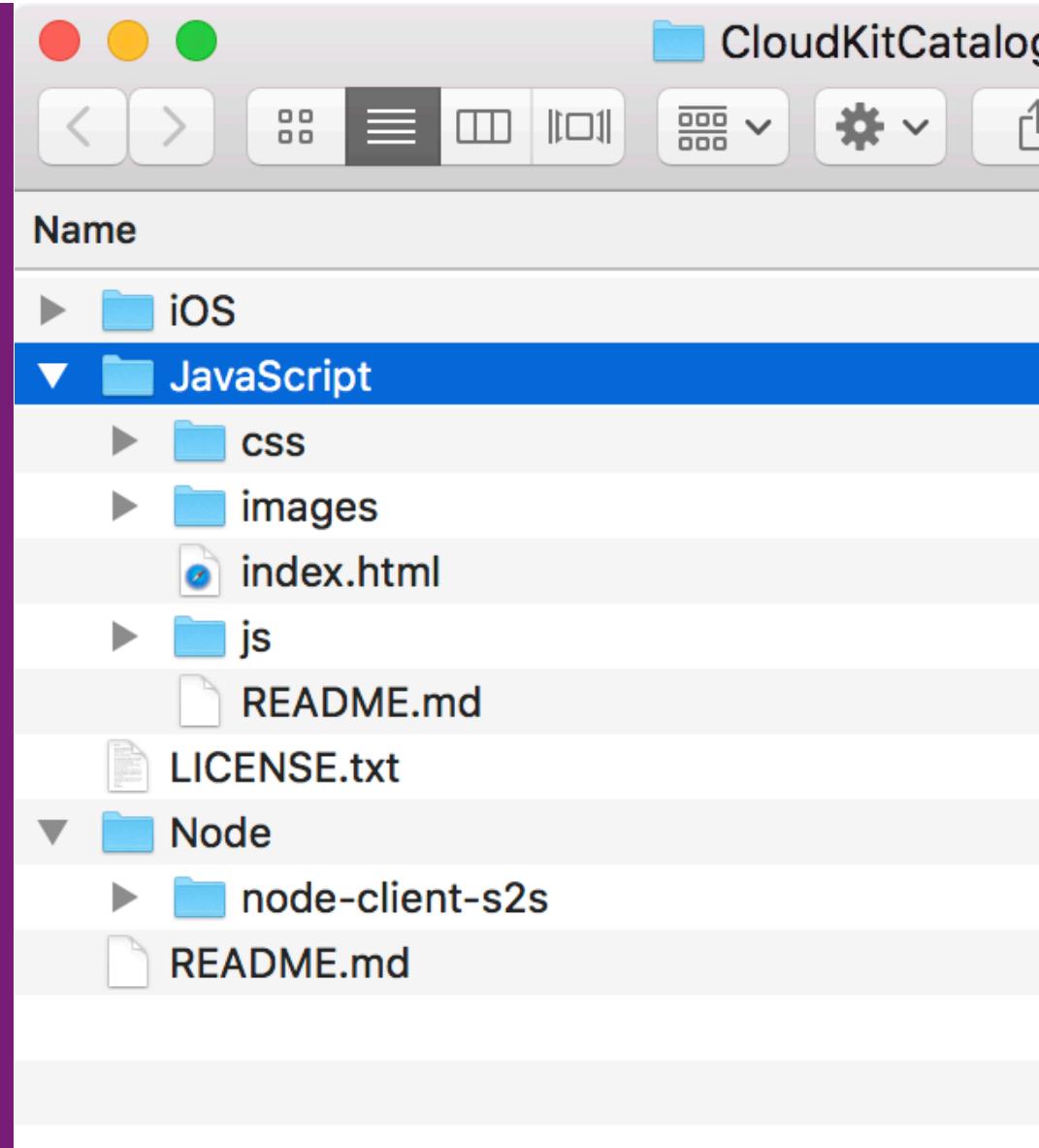
requests_cloudkit	bump version to 0.1.5
.gitignore	first commit
.travis.yml	add install script
LICENSE	first commit
Makefile	first commit
README.rst	Update README.rst
circle.yml	do python setup.py install for circle
requirements.txt	first commit
setup.cfg	first commit
setup.py	fix setup script
test_requests_cloudkit.py	first commit

README.rst

Requests-CloudKit [build passing](#) [pypi v0.1.5](#)

## SERVER-TO-SERVER KEY

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



# SERVER-TO-SERVER KEY

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

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

process.env.NODE_TLS_REJECT_UNAUTHORIZED = "0";

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

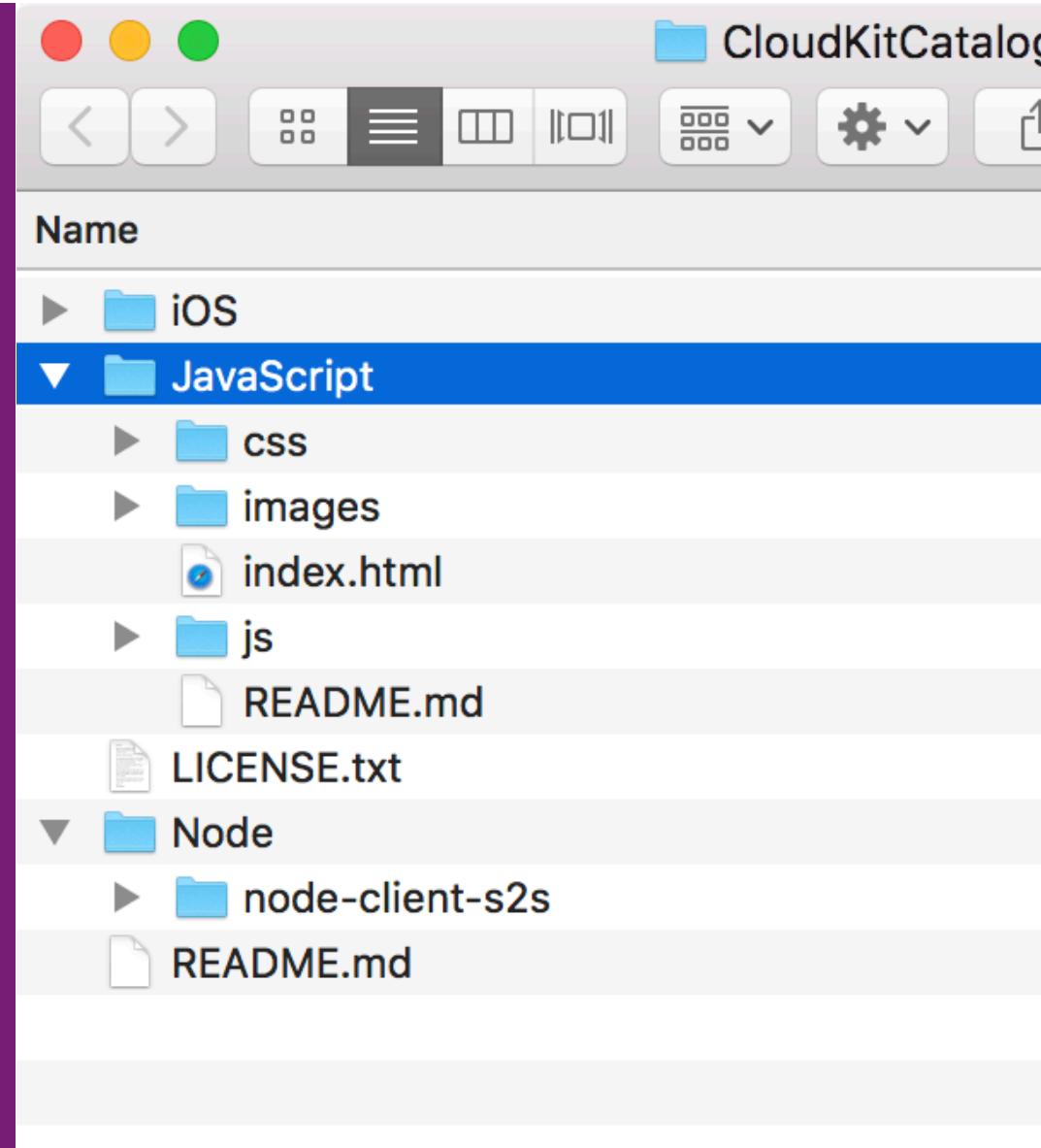
  var CloudKit = require('./cloudkit');
  var containerConfig = require('./config');

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

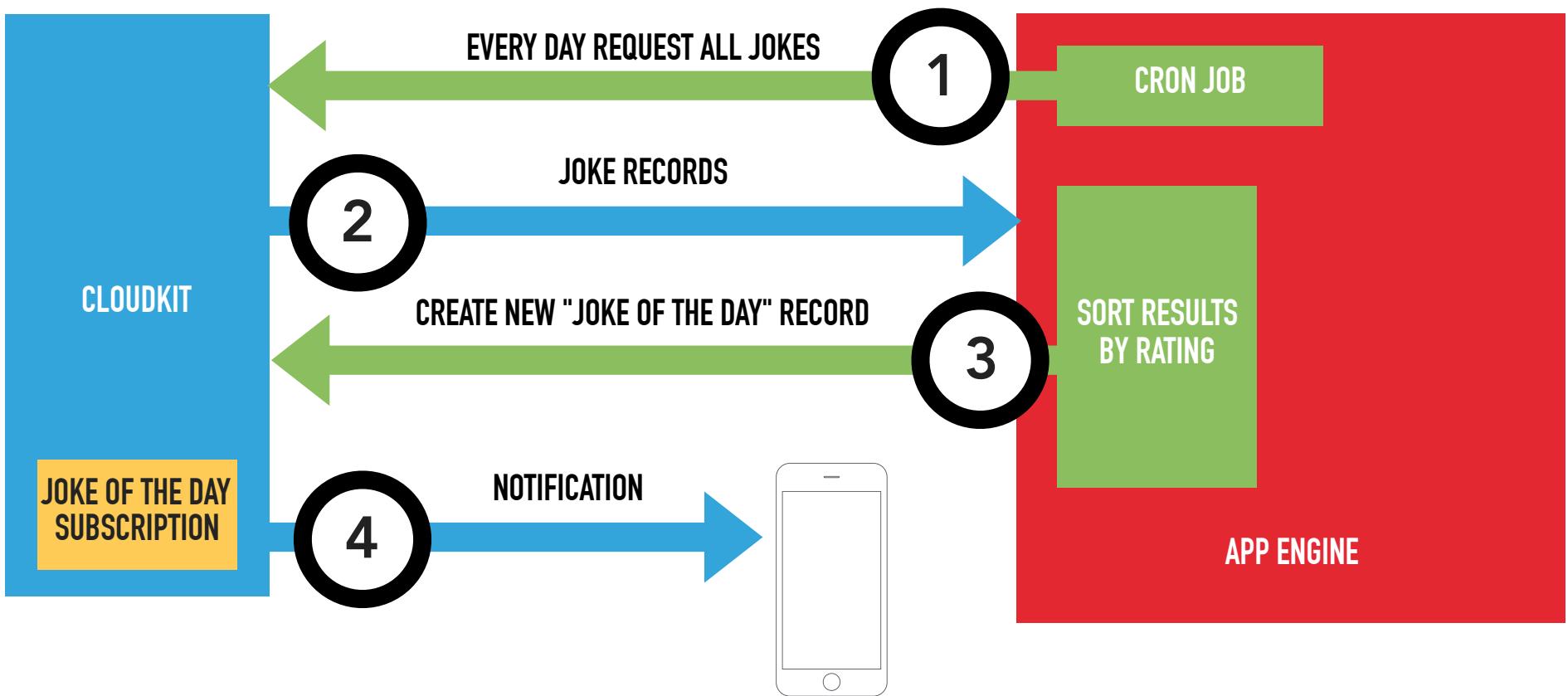
  //CloudKit configuration
  CloudKit.configure({
    containerName: 'ServerToServerKey',
    serverURL: 'https://api.icloud.com',
    applicationUsername: 'icloud.com',
    applicationPassword: 'yourpassword'
  });
});
```

## SERVER-TO-SERVER KEY

- Python so that we can use App Engine



## SERVER-TO-SERVER KEY



# SERVER-TO-SERVER KEY

- Python so that we can use App Engine

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

IOS

# IOS

CloudyWithAC... ▾ Andrew Binkowski ▾ | ?

SCHEMA

- Record Types
- Security Roles
- Subscription Types

PUBLIC DATA

- User Records
- Default Zone
- Usage

PRIVATE DATA

- Default Zone  
For abinkowski@uchicago.edu

SHARED DATA

- No shared zones

ADMIN

- Team
- API Access

## Subscription Types

Sort by Record Type ▾

	Daily
	Signature: b995870e8254b265cce02bfd2c090ec4
	joke
	Signature: 686fb79f17b47aef6f7ee1a7fb7bc82

**Daily**

Signature: b995870e8254b265cce02bfd2c090ec4

RecordType	Daily
Trigger	INSERT
Criteria	

Set up a subscription for the Joke of the Day

# IOS

- Setup subscription for joke of the day

```
//  
// MARK: - Subscriptions  
  
func registerJokeOfTheDaySubscriptions() {  
    let uuid: UUID = UIDevice().identifierForVendor!  
    let identifier = "\u{(uuid)-joke-of-the-day"  
  
    // Create the notification that will be delivered  
    let notificationInfo = CKNotificationInfo()  
    notificationInfo.alertBody = "The Joke of the Day is here! 😂"  
    notificationInfo.shouldBadge = true  
    notificationInfo.shouldSendContentAvailable = true  
    notificationInfo.desiredKeys = ["joke"]  
  
    // Create the subscription  
    let subscription = CKQuerySubscription(recordType: "Daily",  
                                            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("JOTD: subscription failed \(err.localizedDescription)")  
        } else {  
            print("JOTD: subscription set up")  
        }  
    })  
}
```

## IOS

```
// Create the notification that will be delivered
let notificationInfo = CKNotificationInfo()

notificationInfo.alertBody = "The Joke of the Day is here! 😂"
notificationInfo.shouldBadge = true
notificationInfo.shouldSendContentAvailable = true
notificationInfo.desiredKeys = ["joke"]

// Create the subscription
let subscription = CKSubscription(recordType: "Daily",
                                   predicate: NSPredicate(value: true),
                                   subscriptionID: identifier,
                                   options: [.firesOnRecordCreation])
subscription.notificationInfo = notificationInfo
```

- Could send the joke (or make it generic)

# APP ENGINE

## APP ENGINE

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

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

runtime: python27
api_version: 1
threadsafe: true

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

## APP ENGINE

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

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

runtime: python27
api_version: 1
threadsafe: true

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

# APP ENGINE

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

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

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

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

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

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

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

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

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

ecdsa is not a standard module on app engine

# APP ENGINE

```
cd gae_dir
```

```
mkdir lib
```

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



Install the library in your  
project

# APP ENGINE

- Installed in the same directory

```
|── app.yaml  
|── appengine_config.py  
|── cloudkit_helper.py  
|── cron.yaml  
|── eckey.pem  
|── joke_of_the_day.py  
|── lib  
    ├── ecdsa  
    |   ├── __init__.py  
    |   ├── __init__.pyc  
    |   ├── _version.py  
    |   ├── _version.pyc  
    |   ├── curves.py  
    |   ├── curves.pyc  
    |   ├── der.py  
    |   ├── der.pyc  
    |   ├── ecdsa.py  
    |   ├── ecdsa.pyc  
    |   ├── ellipticcurve.py  
    |   ├── ellipticcurve.pyc  
    |   ├── keys.py  
    |   ├── keys.pyc  
    |   ├── numbertheory.py  
    |   ├── numbertheory.pyc  
    |   ├── rfc6979.py  
    |   └── rfc6979.pyc
```

## APP ENGINE

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

- Add a new file: appengine\_cofig.py

# APP ENGINE

- main.py
- Handlers
  - Processing
  - Posting record

```
import webapp2

import sys
import json

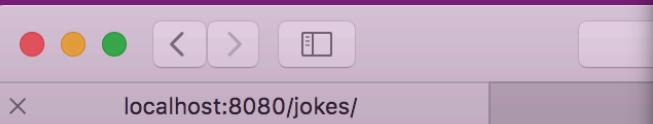
import cloudkit_helper as ck
from joke_of_the_day import JokeOfTheDay

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

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

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

# APP ENGINE

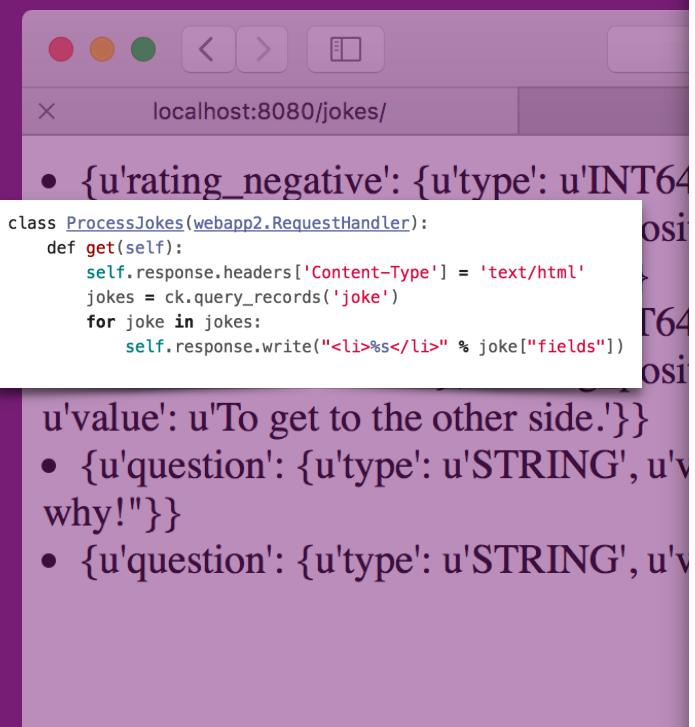


- {u'rating\_negative': {u'type': u'INT64', u'value': 10}, u'question': {u'type': u'String', u'value': u'Why did the chicken cross the road?'}, u'rating\_positive': {u'type': u'INT64', u'value': 10}, u'response': {u'type': u'String', u'value': u'To get to the other side.'}}
- {u'rating\_negative': {u'type': u'INT64', u'value': 10}, u'question': {u'type': u'String', u'value': u'Why did the chicken cross the road?'}, u'rating\_positive': {u'type': u'INT64', u'value': 10}, u'response': {u'type': u'String', u'value': u'To get to the other side.'}}
- {u'question': {u'type': u'String', u'value': u'Why?\n'}, u'response': {u'type': u'String', u'value': u"That's why!"}}
- {u'question': {u'type': u'String', u'value': u"Why didn't that work?"}}

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

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

# APP ENGINE



A screenshot of a web browser window titled "localhost:8080/jokes/". The page displays a list of jokes in an unordered list:

- {u'rating\_negative': {u'type': u'INT64', u'value': u'To get to the other side.'}}
- {u'question': {u'type': u'String', u'value': u'why!'}}
- {u'question': {u'type': u'String', u'value': u'what's the best way to get to the other side?'}}

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

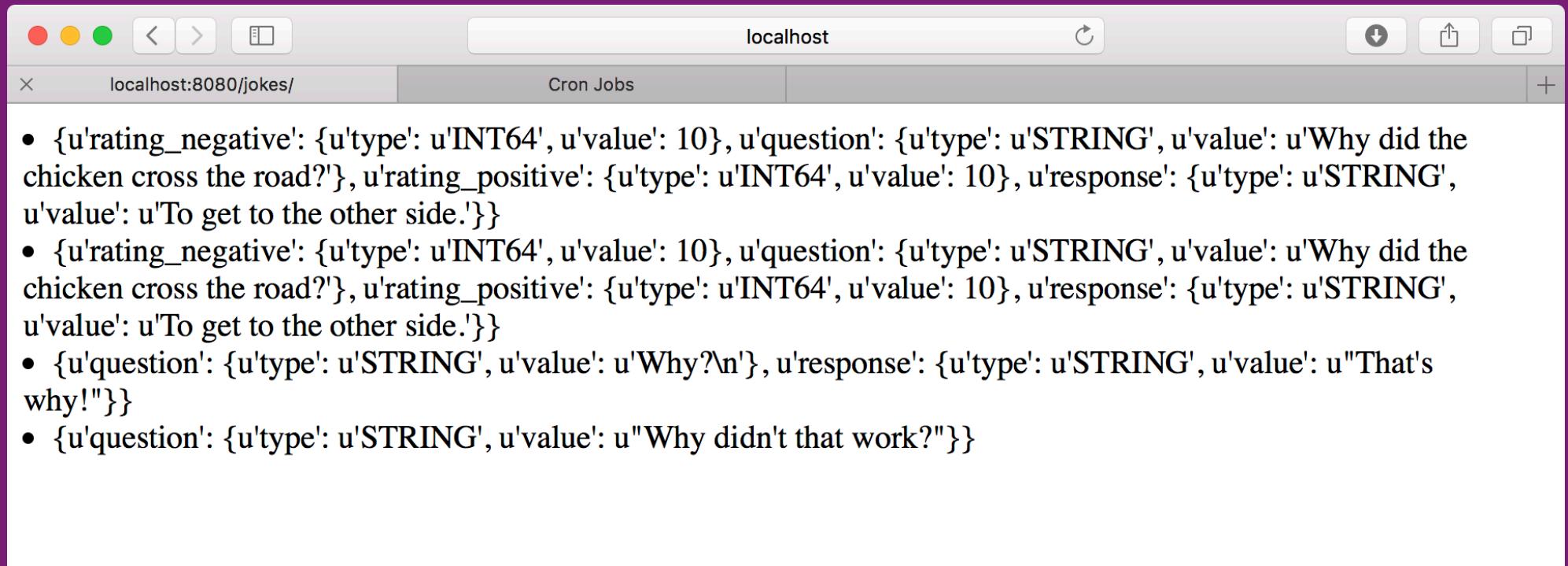
    records = []
    while True:
        result_query_authors = cloudkit_request(
            '/development/public/records/query',
            json.dumps(json_query))
        result_query_authors = json.loads(result_query_authors['content'])

        records += result_query_authors['records']

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

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

# APP ENGINE



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

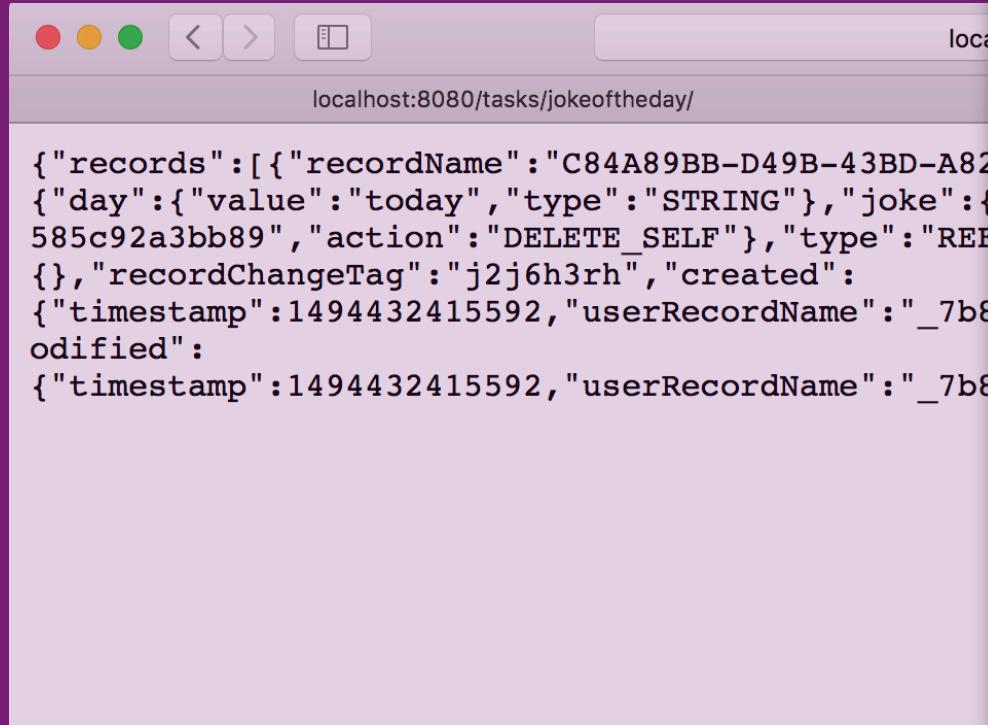
## APP ENGINE

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

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

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

# APP ENGINE



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

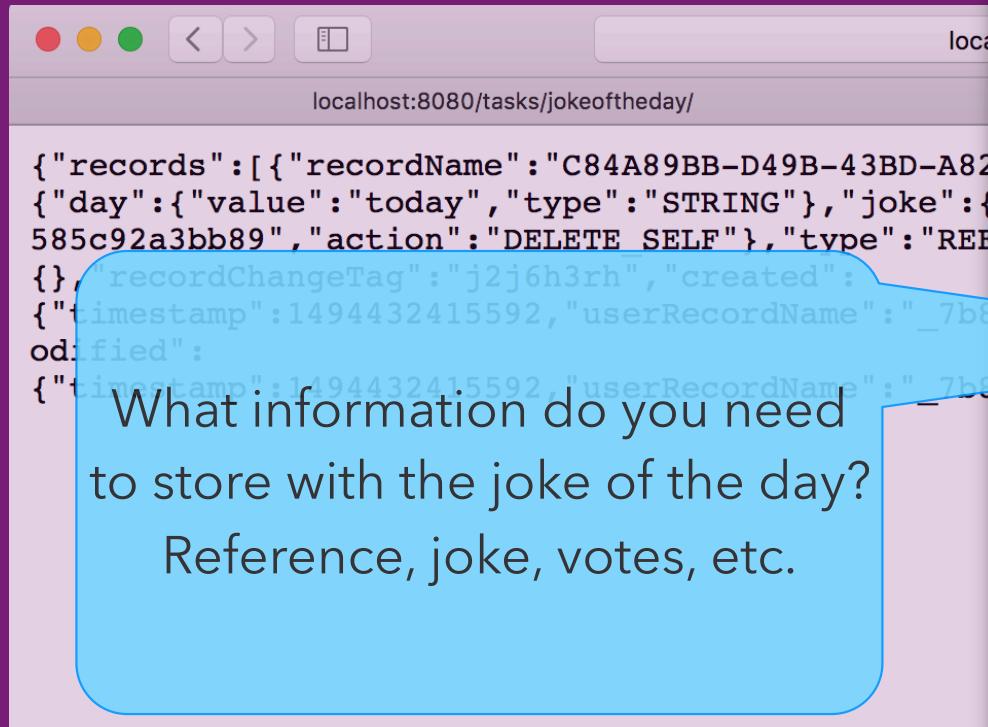
```
class JokeOfTheDay(webapp2.RequestHandler):
    #
    # Create a new joke of the day record and post it to iCloud
    # Note: We are hard coming the reference here, you should get
    # if from the processing of the daily joke ratings
    def get(self):

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

        #print('Posting operation to create quote...')
        result_modify_jokes = ck.cloudkit_request(
            '/development/public/records/modify',
            json.dumps(new_joke_of_the_day_data))

        self.response.headers['Content-Type'] = 'text/json'
        self.response.write(result_modify_jokes['content'])
```

# APP ENGINE

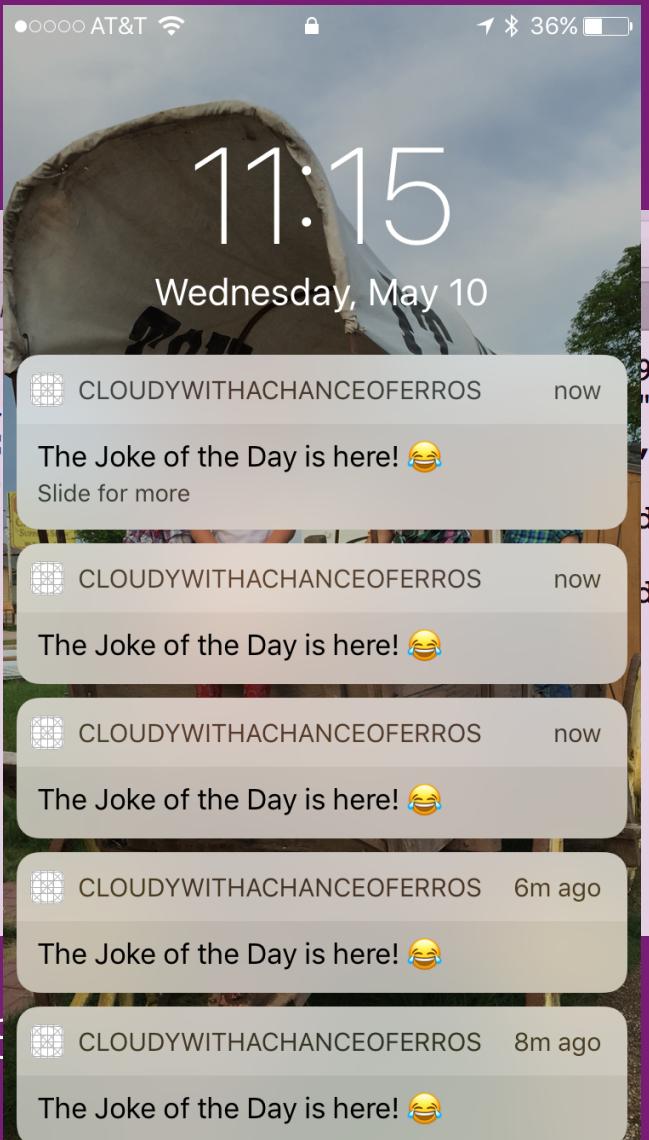


- <http://localhost:8080/tasks/jokeoftheday/>

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

# APP ENGINE

```
{ "records": [ { "recordName": "today", "action": "DELETED", "timestamp": 1494432415592 } ] }
```



- <http://localhost:8080>

# APP ENGINE

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

- Add a new file: cron.yaml

# APP ENGINE

## CRON JOB



Development SDK 1.9.50

dev~None

Instances

Datastore Viewer

Datastore Indexes

Datastore Stats

Interactive Console

Memcache Viewer

Blobstore Viewer

Task Queues

Cron Jobs

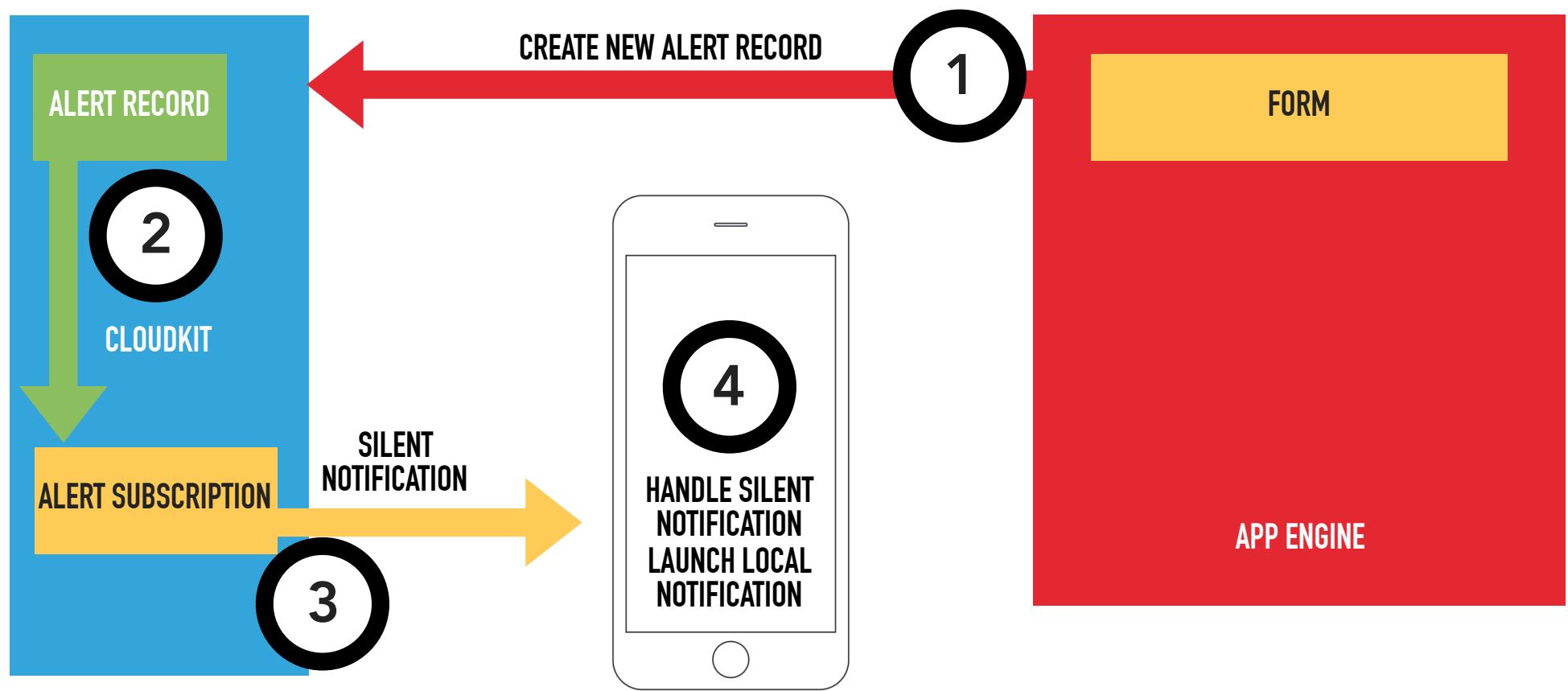
### Cron Jobs

Request to /tasks/jokeoftheday/ succeeded!

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

# CUSTOM NOTIFICATIONS

# CUSTOM NOTIFICATIONS



# CUSTOM NOTIFICATIONS

CloudyWithAC... ▾

SCHEMA

- Record Types
- Security Roles
- Subscription Types

PUBLIC DATA

- User Records
- Default Zone
- Usage

PRIVATE DATA

- Default Zone  
For abinkowski@uchicago.edu

SHARED DATA

No shared zones

ADMIN

- Team
- API Access

Record Types

Sort by Name ▾

<b>Alert</b>	10 Unused Indexes
<b>Daily</b>	9 Unused Indexes
<b>joke</b>	15 Unused Indexes
<b>Users</b>	3 Public Records, 1 Private Record

trash +

## Alert

Created: **May 10 2017 11:42 AM** Modified: **May 10 2017 11:42 AM** Security: **Default**

Indexes: **3** Metadata Indexes: **7** Records: **-**

Field Name	Field Type	Index
message	String	<input checked="" type="checkbox"/> Sort <input checked="" type="checkbox"/> Query <input checked="" type="checkbox"/> Search

[Add Field...](#)

Andrew Binkowski ▾ | ?

# CUSTOM NOTIFICATIONS

- Set up a new route

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

# CUSTOM NOTIFICATIONS

- Create a new Alerts record with the passed in message

```
class AlertNotifications(webapp2.RequestHandler):  
    def post(self):  
        message = self.request.get('message')  
        new_joke_of_the_day_data = {  
            'operations': [{  
                'operationType': 'create',  
                'record': {  
                    'recordType': 'Alert',  
                    'fields': {  
                        'message': {'value': message},  
                    }  
                }  
            }]  
        }  
  
        result_modify_jokes = ck.cloudkit_request(  
            '/development/public/records/modify',  
            json.dumps(new_joke_of_the_day_data))  
  
        self.response.headers['Content-Type'] = 'text/json'  
        self.response.write(result_modify_jokes['content'])  
        #self.redirect('/alerts/')  
  
    def get(self):  
        self.response.headers['Content-Type'] = 'text/html'  
        html = """  
        <form action="/alerts/" method="post">  
        <b>Alert Notifcation Message</b><br>  
        <input type="text" name="message" value=""><br>  
        <input type="submit" value="Submit">  
        </form>  
        """  
        self.response.write(html)
```

# CUSTOM NOTIFICATIONS

- Create an amazing form

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

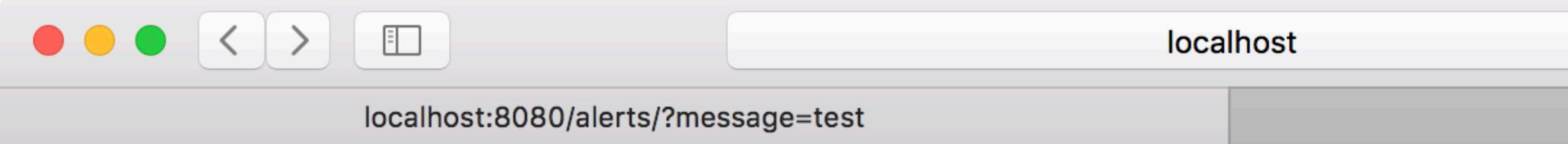
    self.response.write(html)
```

# CUSTOM NOTIFICATIONS

- Create a new Alerts record with the passed in message

```
class AlertNotifications(webapp2.RequestHandler):  
    def post(self):  
        message = self.request.get('message')  
        new_joke_of_the_day_data = {  
            'operations': [{  
                'operationType': 'create',  
                'record': {  
                    'recordType': 'Alert',  
                    'fields': {  
                        'message': {'value': message},  
                    }  
                }  
            }]  
        }  
  
        result_modify_jokes = ck.cloudkit_request(  
            '/development/public/records/modify',  
            json.dumps(new_joke_of_the_day_data))  
  
        self.response.headers['Content-Type'] = 'text/json'  
        self.response.write(result_modify_jokes['content'])  
        #self.redirect('/alerts/')
```

# CUSTOM NOTIFICATIONS

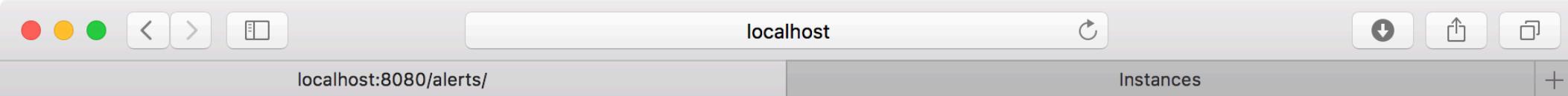


## Alert Notification Message

I can't believe this works! 😊

**Submit**

# CUSTOM NOTIFICATIONS



```
{"records": [{"recordName": "7836D815-D22B-4440-8797-F8DBDE1D9574", "recordType": "Alert", "fields": {"message": {"value": "I can't believe this works! \ud83d\udcbb", "type": "STRING"}}, "pluginFields": {}, "recordChangeTag": "j2j9ajfb", "created": {"timestamp": 1494437148145, "userRecordName": "_7b892f2a3eeadd6afc77ff7158f69d9", "deviceID": "2"}, "modified": {"timestamp": 1494437148145, "userRecordName": "_7b892f2a3eeadd6afc77ff7158f69d9", "deviceID": "2"}}]}]
```

# CUSTOM NOTIFICATIONS

The screenshot shows a database management interface with a sidebar and a main query results area.

**Left Sidebar:**

- CloudyWithAC... ▾
- SCHEMA**
  - Record Types
  - Security Roles
  - Subscription Types
- PUBLIC DATA**
  - User Records
  - Default Zone
  - Usage
- PRIVATE DATA**
  - Default Zone  
For abinkowski@uchicago.edu
- SHARED DATA**
  - No shared zones
- ADMIN**
  - Team

**Main Area:**

Alert ▾ Sort by message ▾

**Record Details:**

I can't believe this works! &#128521;  
A2774EA7-A1A3-49B1-87A7-297A16DC699F

This is custom  
4974951B-9639-4895-8A04-29EACBC20DCA

This is custom  
62226496-60E3-47D2-8D6D-19CD4FB9830F

**Record View:**

Andrew Binkowski ▾ ( )

Record Name:  
A2774EA7-A1A3-49B1-87A7-297A16DC6...

Created: May 10 2017 12:1... Created By: \_7b892f2a3eeadd... Modified: May 10 2017 12:1... Modified By: \_7b892f2a3eeadd...

message I can't believe this works! &#128521; String

**Callout:**

Emojis did not survive 😞

# CUSTOM NOTIFICATIONS

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

```
/// Called for push notifications
/// In this case, we are getting the changed record from cloudkit and then creating a new notification
func application(_ application: UIApplication,
                  didReceiveRemoteNotification userInfo: [AnyHashable : Any],
                  fetchCompletionHandler completionHandler: @escaping (UIBackgroundFetchResult) -> Void) {

    // Get the APS notification data
    let aps = userInfo["aps"] as! [String: AnyObject]
    print("APS: \(aps)")

    let contentAvailable = aps["content-available"] as! Int

    if contentAvailable == 1 {
        // Pull data
        let cloudKitInfo = userInfo["ck"] as! [String: AnyObject]
        let recordId = cloudKitInfo["qry"]?[["rid"] as! String]
        let field = cloudKitInfo["qry"]?[["af"] as! [String: AnyObject]]
        let message = field["message"] as! String

        // Create notification content
        let content = UNMutableNotificationContent()
        content.title = "Joke of the Day"
        content.subtitle = "Local from Silent"
        content.body = message
        content.sound = UNNotificationSound.default()

        // Set up trigger
        let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 5, repeats: false)

        // Create the notification request
        let center = UNUserNotificationCenter.current()
        let identifier = recordId
        let request = UNNotificationRequest(identifier: identifier,
                                            content: content, trigger: trigger)
        center.add(request, withCompletionHandler: { (error) in
            if let error = error {
                print("Something went wrong: \(error)")
            }
        })

        completionHandler(.newData)
    } else {
        completionHandler(.noData)
    }
}
```

## CUSTOM NOTIFICATIONS

Silent notification content

```
[AnyHashable("aps"): {  
    "content-available" = 1;  
}, AnyHashable("ck"): {  
    ce = 2;  
    cid = "iCloud.cloud.uchicago.CloudyWithAChanceOfErrors";  
    nid = "2b661f39-5617-4a1c-a889-ee09cd9eae7a";  
    qry = {  
        af = {  
            message = "Where are my emojis?";  
        };  
        dbs = 2;  
        fo = 1;  
        rid = "A0915CF5-E4E7-4C76-856E-48CE01ADCACE";  
        sid = "14725FE6-9C8F-4BDC-96FF-3815E5FE1E3B-alert";  
        zid = "_defaultZone";  
        zoid = "_defaultOwner";  
    };  
};  
]
```

Field I requested

Record that  
matched the  
query

# CUSTOM NOTIFICATIONS

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

    let contentAvailable = aps["content-available"] as! Int

    if contentAvailable == 1 {
        // Pull data
        let cloudKitInfo = userInfo["ck"] as! [String: AnyObject]
        let recordId = cloudKitInfo["qry"]?["rid"] as! String
        let field = cloudKitInfo["qry"]?["af"] as! [String: AnyObject]
        let message = field["message"] as! String
```

# CUSTOM NOTIFICATIONS

```
// Create notification content
let content = UNMutableNotificationContent()
content.title = "Joke of the Day"
content.subtitle = "Local from Silent"
content.body = message
content.sound = UNNotificationSound.default()

// Set up trigger
let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 5,repeats: false)

// Create the notification request
let center = UNUserNotificationCenter.current()
let identifier = recordId
let request = UNNotificationRequest(identifier: identifier,
                                    content: content, trigger: trigger)
center.add(request, completionHandler: { (error) in
    if let error = error {
        print("Something went wrong: \(error)")
    }
})

completionHandler(.newData)

} else {
    completionHandler(.noData)
}
}
```

# DATA MODELING

# DATA MODELING

- How should 1 to many relationships be handled?

Album

PhotoArray

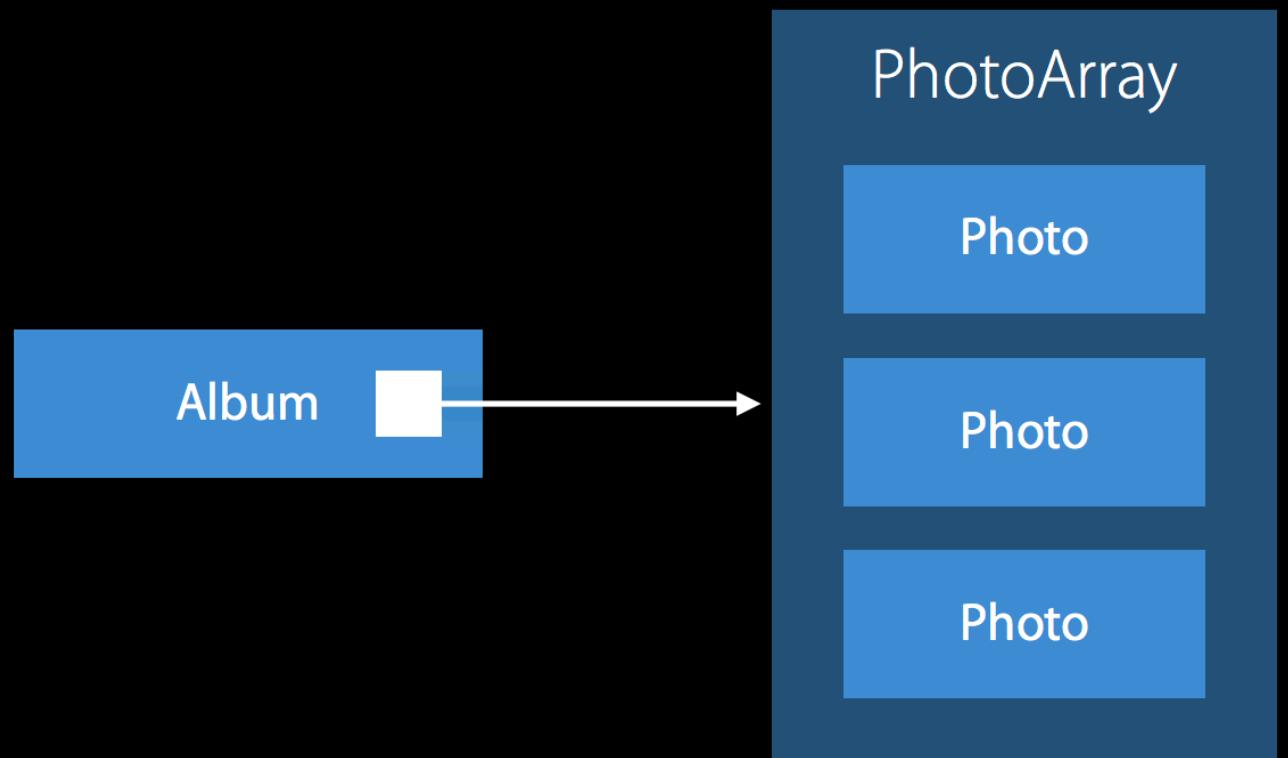
Photo

Photo

Photo

## DATA MODELING

- Typical setup to model

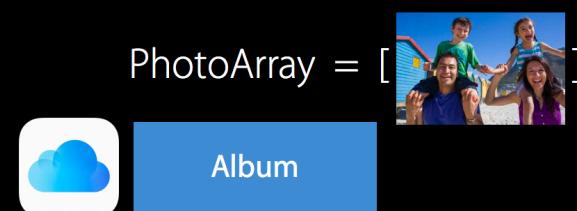


# DATA MODELING

PhotoArray = []



# DATA MODELING

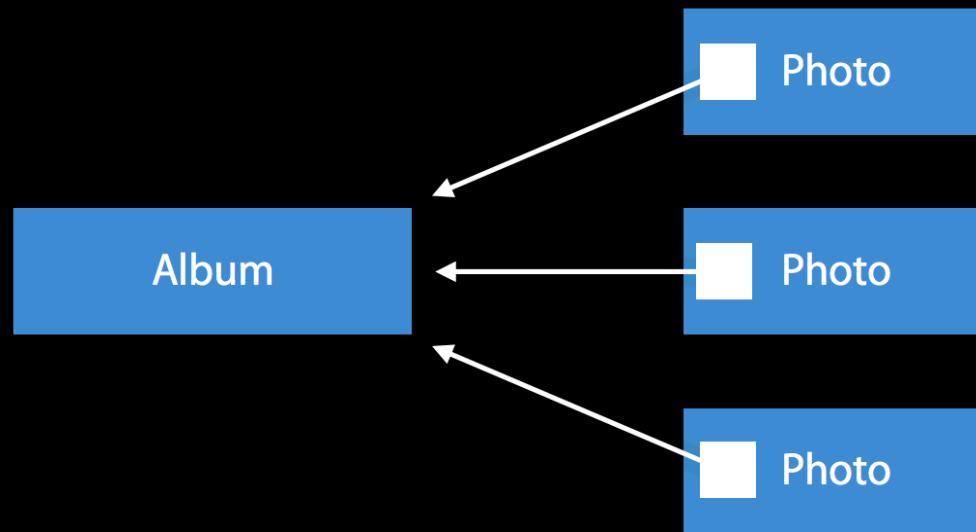


serverRecordChanged



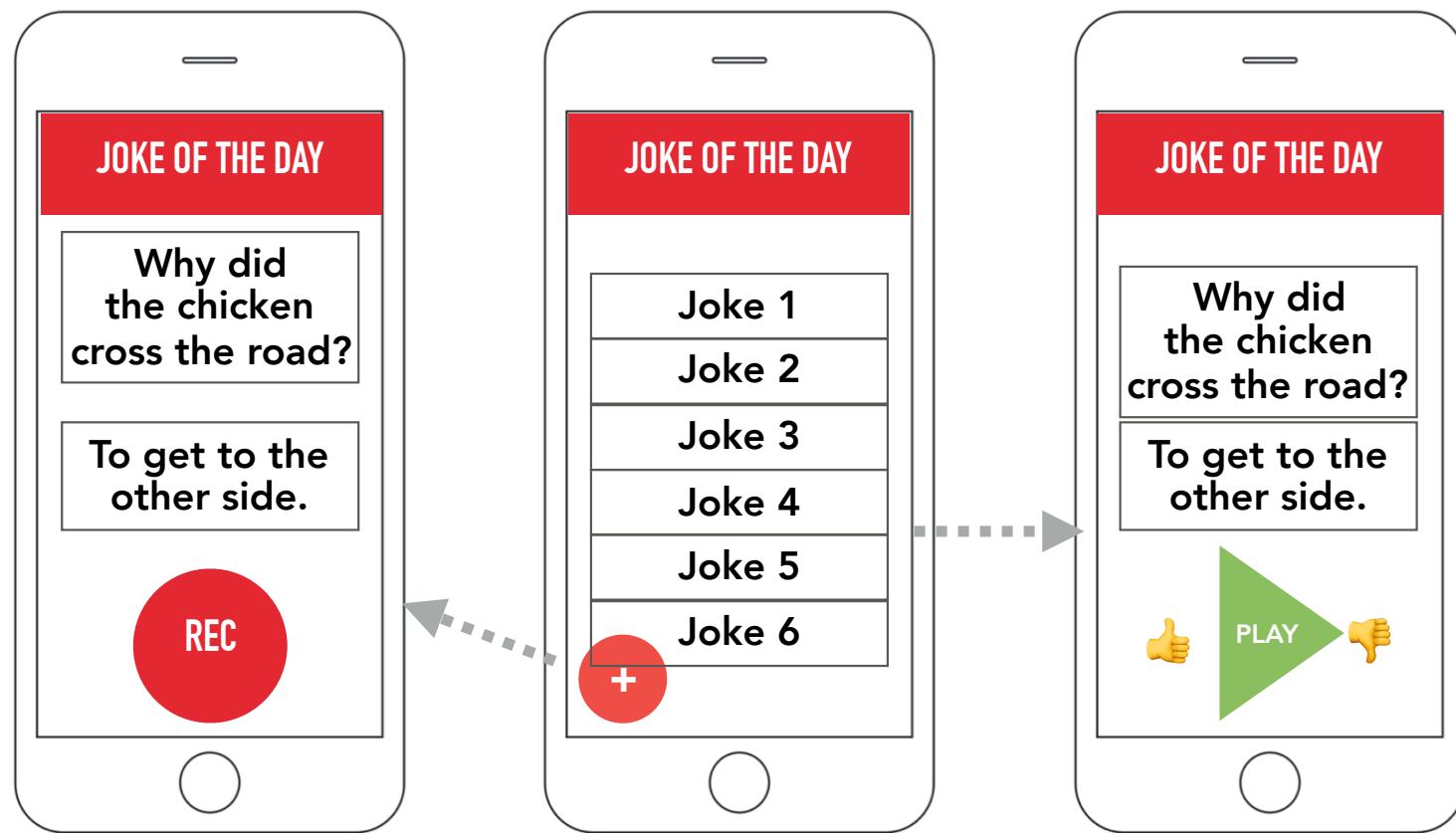
serverRecordChanged

# DATA MODELING



# ASSIGNMENT 4

# ASSIGNMENT 4





THE UNIVERSITY OF  
CHICAGO



MPCS 51033 • SPRING 2017 • SESSION 6

---

# BACKENDS FOR MOBILE APPLICATIONS