



THE UNIVERSITY OF
CHICAGO



MPCS 51033 • AUTUMN 2017 • SESSION 8

BACKENDS FOR MOBILE APPLICATIONS

CLASS NEWS

CLASS NEWS

- Office hours Thursday from 10-11:30 in Young 308

CLASS NEWS

- Week 7 - CloudKit
 - Assignment 5 assigned
- Week 8 - CloudKit Server Extensions; Swift on the Server
 - Case Study assigned
 - Finish server component of Assignment 5
- Week 9 - Case Studies in Class (Assignment 5 Due)
- Week 10 - Cloud Roundup (Realm, Serverless, etc.)

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.)
 - 30 minutes

CLASS NEWS

- Week 7 - CloudKit
 - Assignment 5 assigned
- Week 8 - CloudKit Server Extensions; Swift on the Server
 - Case Study assigned
 - Finish server component of Assignment 5
- Week 9 - Case Studies in Class (Assignment 5 Due)
- Week 10 - Cloud Roundup (Realm, Serverless, etc.)

MAKE-UP CLASS
MATERIALS AFTER
PRESENTATIONS

CLASS NEWS

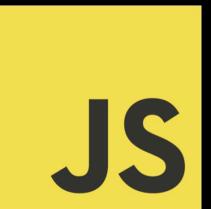
- Final Projects
 - Opportunity for you to apply what you learned in class to a project you care about
 - Can use old iOS project or extend assignment
 - Talk me about your ideas early

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" />
```



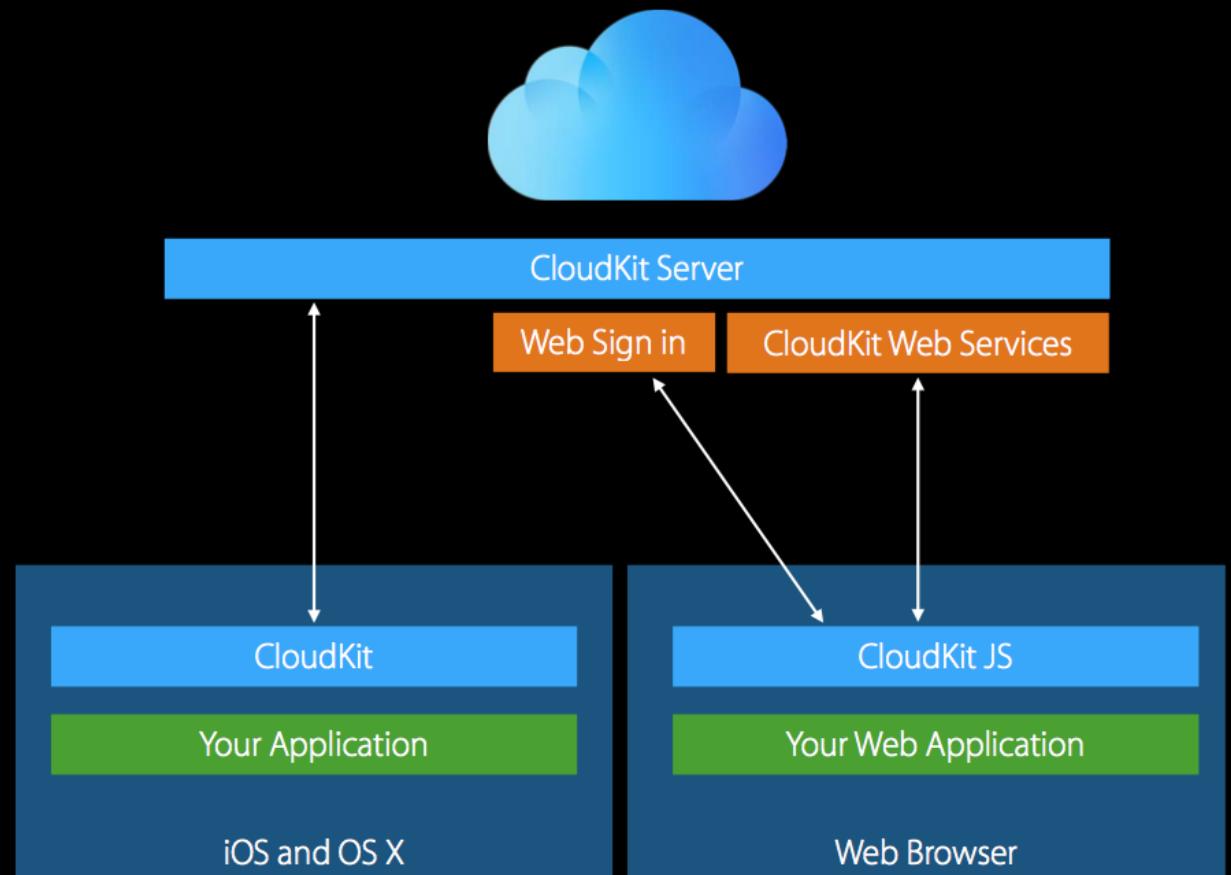
WEB SERVICE API

- 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

- An API token is used to allow a web front-end to send requests to this container and environment
- A server-to-server key allows a server-side application to read and write data in the public database of this container & environment

The screenshot shows the iCloud CloudKit developer portal interface. At the top, there's a navigation bar with a cloud icon and the text "iCloud.cloud.uchicago.CloudyWithAChanceOfEr". Below the navigation bar, there are two main sections: "DEVELOPMENT API ACCESS" and "API TOKENS". Under "API TOKENS", there are two entries: "Javascript Token" and "Token". The "Token" entry is highlighted with a blue background and has the sub-label "API Token". Below these, there's a section titled "SERVER TO SERVER KEYS" with an entry "NewKey" and a sub-label "Server-to-Server Key". There are two buttons at the bottom of this section: "New Token" and "New Key", both of which are highlighted with a blue border. To the right of the "Token" entry, there are several columns with placeholder text: "API Token", "Name", "Sign In Callback", "Allowed Origins", "Discoverability", "Notes", and a red "Cancel" button at the bottom.

CLOUDKIT JS

- Create a new token

The screenshot shows the iCloud Development API Access interface for creating a new API token. The left sidebar lists 'API TOKENS' and 'SERVER TO SERVER KEYS'. Under 'API TOKENS', 'Javascript Token' is selected, and 'Token' is highlighted. Below the sidebar, a note explains that an API token allows a web front-end to send requests to the container and environment, while a server-to-server key allows a server-side application to read and write data in the public database. Two buttons, 'New Token' and 'New Key', are available. The main right panel is titled 'Development API Access' and contains fields for 'Name' (set to 'Token'), 'Sign In Callback' (set to 'postMessage'), 'Allowed Origins' (set to 'Any domain'), and 'Discoverability' (set to 'Request user discoverability at sign in'). A 'Notes' section is also present. A 'Cancel' button is at the bottom.

DEVELOPMENT API ACCESS

API TOKENS

Javascript Token

API Token

Token

API Token

SIGN IN CALLBACK

API Token Shown after the token has been created.

Name

Sign In Callback postMessage

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

https://

Redirects the browser back to this URL after the Apple ID sign in has completed.

ALLOWED ORIGINS

Allowed Origins Any domain

Specific domains:

Discoverability Request user discoverability at sign in

Notes

Cancel

CLOUDKIT JS

CloudKit Catalog

Run Code

Unauthenticated User

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 `whenUserSignsOut` 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() {
```



CLOUDKIT JS

- Token is used to communicate with iCloud

API Token 1b20bfb72a908993605ca3662ef31c120851405732013b7a012a93019e

Name Javascript Token

Sign In Callback postMessage

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

https://

Redirects the browser back to this URL after the Apple ID sign in has completed.

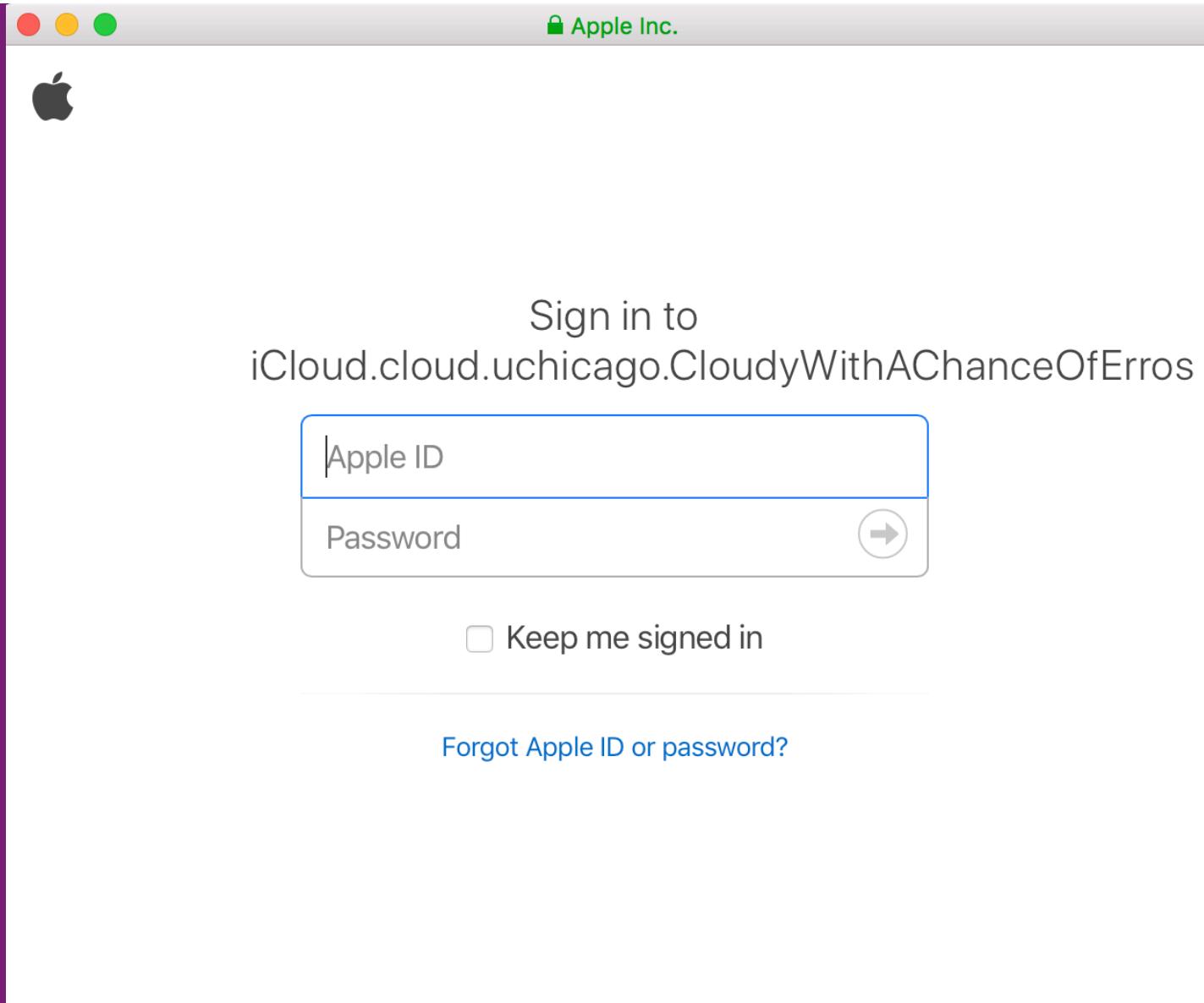
Allowed Origins Any domain

Specific domains:

Discoverability Request user discoverability at sign in

CLOUDKIT JS

- Authentication happens with Apple provided UI



CLOUDKIT JS

- Test directly in CloudKitCatalog web app

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

- Test directly in CloudKitCatalog web app

The screenshot shows the CloudKit Catalog web application interface. At the top, there is a navigation bar with the title "CloudKit Catalog" and a "Run Code" button. Below the navigation bar is a sidebar containing several sections: "README", "Authentication" (which is highlighted with a blue background), "Discoverability", "Query", "Zones", "Records", "Sync", "Sharing", "Subscriptions", and "Notifications" (with a note "Disconnected"). To the right of the sidebar, the main content area displays the result of a code run. It shows the container name "iCloud.cloud.uchicago.CloudyWithAChanceOfErrors" and the environment "development". The result itself is a simple text output: "Result" followed by the name "Andrew Binkowski" and 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 using a server-to-server key
- The missing link in data processing and server side logic 🤔



SERVER-TO-SERVER

- API calls can only be made to the public database
- Calls run as with the inherited privileges of the creator of the key



SERVER-TO-SERVER

- Apple provides demo for node.js



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

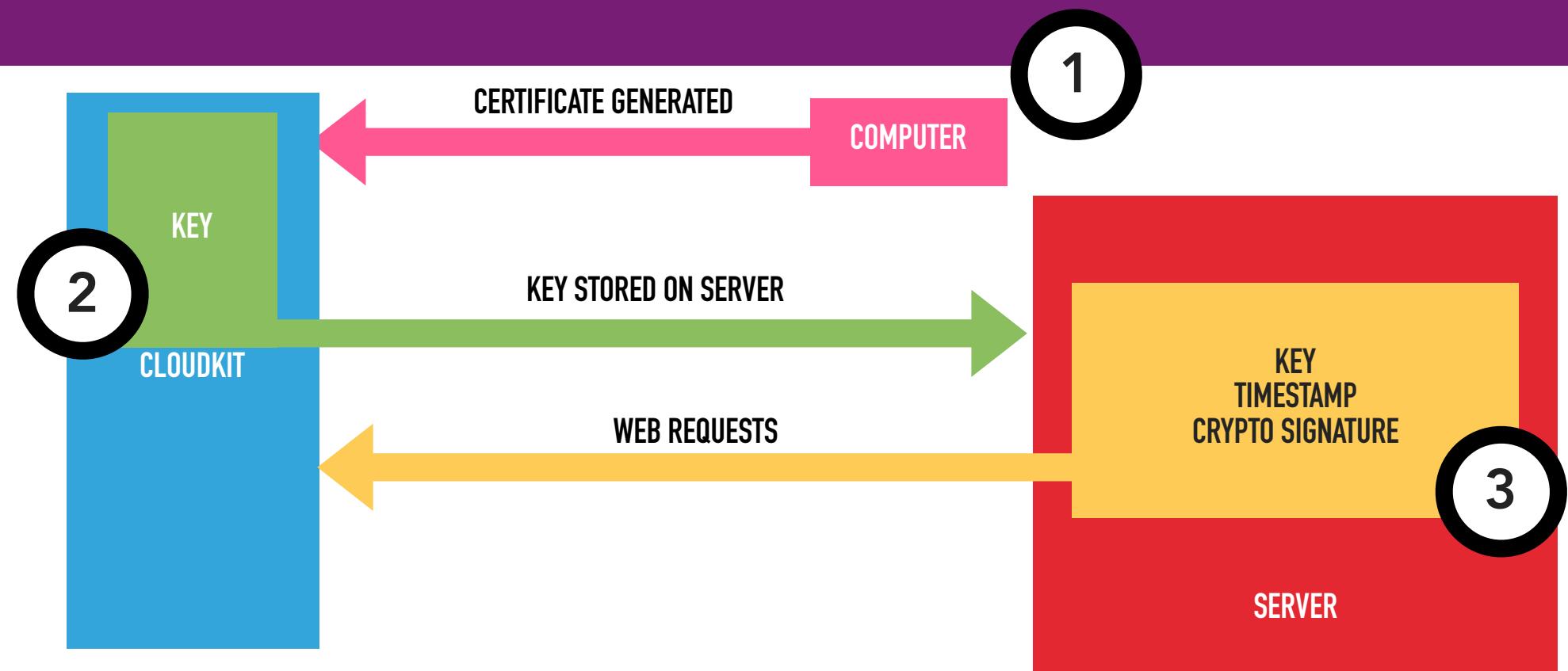
In order to use CloudKit JS server-side you will need a `fetch` implementation such as `node-fetch` which you can install via NPM. You must also download CloudKit JS itself from Apple's CDN.

```
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

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 as part of project
- Can be revoked if loose track of it

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
```

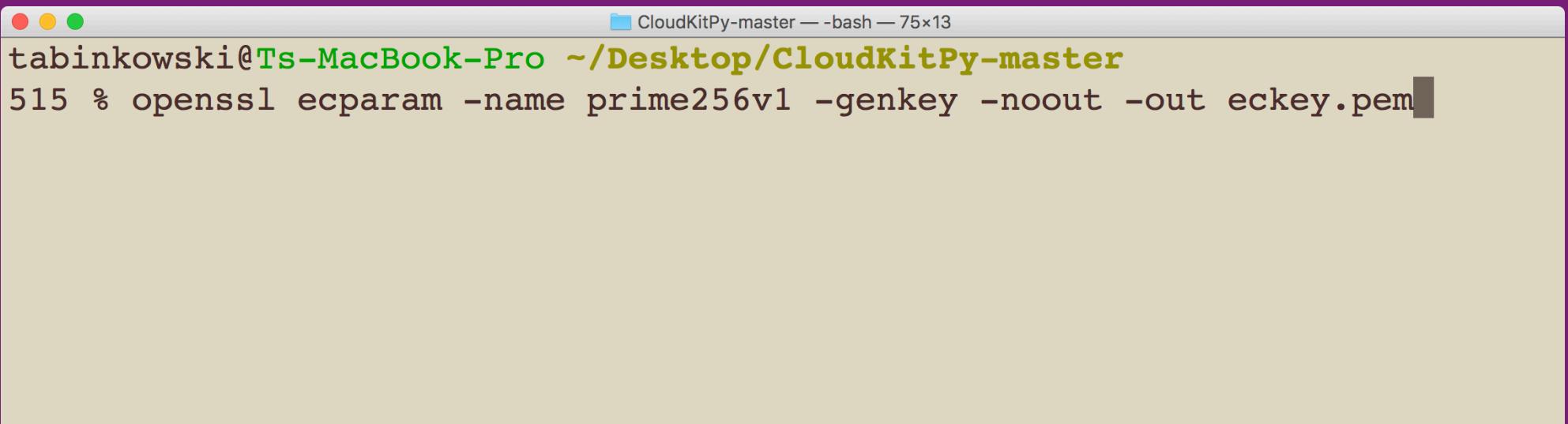
The public key appears in the output.

```
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



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 the user "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

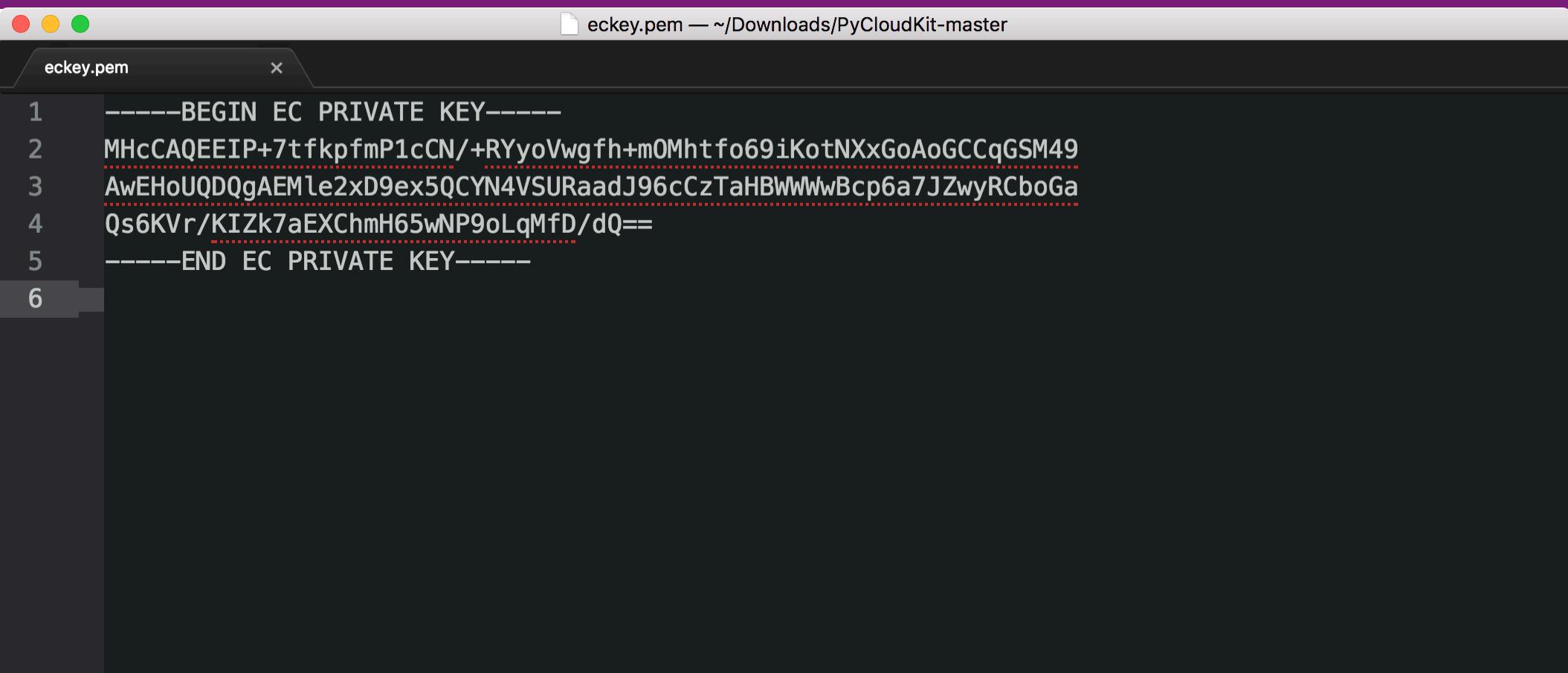
[Downloads: Source code](#)

[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

 > iCloud.cloud.uchicago.CloudyWithAChanceOfErros > Development API Access ANDREW BINKOWSKI ▾

DEVELOPMENT API ACCESS

API TOKENS

Javascript Token
API Token

SERVER TO SERVER KEYS

NewKey
Server-to-Server Key

Key
Server-to-Server Key

CREATE A NEW KEY

New Token New Key

Key ID Shown after the key has been created.

Name

Public Key

1. 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 public database.

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

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

SERVER-TO-SERVER KEY

API TOKENS

Javascript Token

API Token

SERVER TO SERVER KEYS

NewKey

Server-to-Server Key

Key

Server-to-Server Key

[New Token](#)

[New Key](#)

An API token is used to allow a web front-end to send requests to this container and environment. A server-to-server key allows a server-side application to read and write data in the public database of this container &

INSTRUCTIONS

Key ID Shown after the key has been created.

Name

Key

Public Key

1. 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 public database.

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

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

Paste your public key here (only the value between the BEGIN and END text)...

SERVER-TO-SERVER KEY

DEVELOPMENT API ACCESS

API TOKENS

Javascript Token

API Token

SERVER TO SERVER KEYS

NewKey

Server-to-Server Key

Key ID ffdb30c23add822e851e15cdd30ea819251d2fbb9bc59c4803f006dcc900191

Name

Public Key MFkwEwYHKOZIzj0CAQYIKoZIzj0DAQcDQgAEMle2xD9ex5QCYN4VSURaadJ96cCz
TaHBWWWwBcp6a7JZwyRCboGaQs6KVr/KIZk7aEXChmH65wNP9oLqMfD/dQ==



```
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 %
```

Created May 9 2017 9:41 PM by Andrew Binkowski

WEB SERVICE REQUESTS

WEB SERVICE REQUESTS

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

- **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 \(records/lookup\)](#).

- API Endpoints for CloudKit
- Look at CloudKit catalog for API examples

WEB SERVICE REQUEST 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 Zone ID Dictionary .
resultsLimit	The maximum number of records to fetch. The default is the maximum number of records in a response that is allowed, described in Data Size Limits .
query	The query to apply, described in Query Dictionary . 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 Response . 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 operations

Record Operation Dictionary

The operation dictionary keys are:

Key	Description
operationType	The type of operation. Possible values are described in Operation Type Values . This key is required.
record	A dictionary representing the record to modify, as described in Record Dictionary . 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 Record Dictionary . If unsuccessful, the result dictionary contains the keys described in Record Fetch Error Dictionary .
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)
- `continuationMarker` for the next batch

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

URL:

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

API URL

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

QUERY JSON

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 Record Operation Dictionary . See Data Size Limits for maximum number of operations allowed. This key is required.
zoneID	A dictionary that identifies a record zone in the database, described in Zone ID Dictionary .
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 AS UUID

- 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

ADD A RECORD

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

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 Record Dictionary . If unsuccessful, the result dictionary contains the keys described in Record Fetch Error Dictionary .

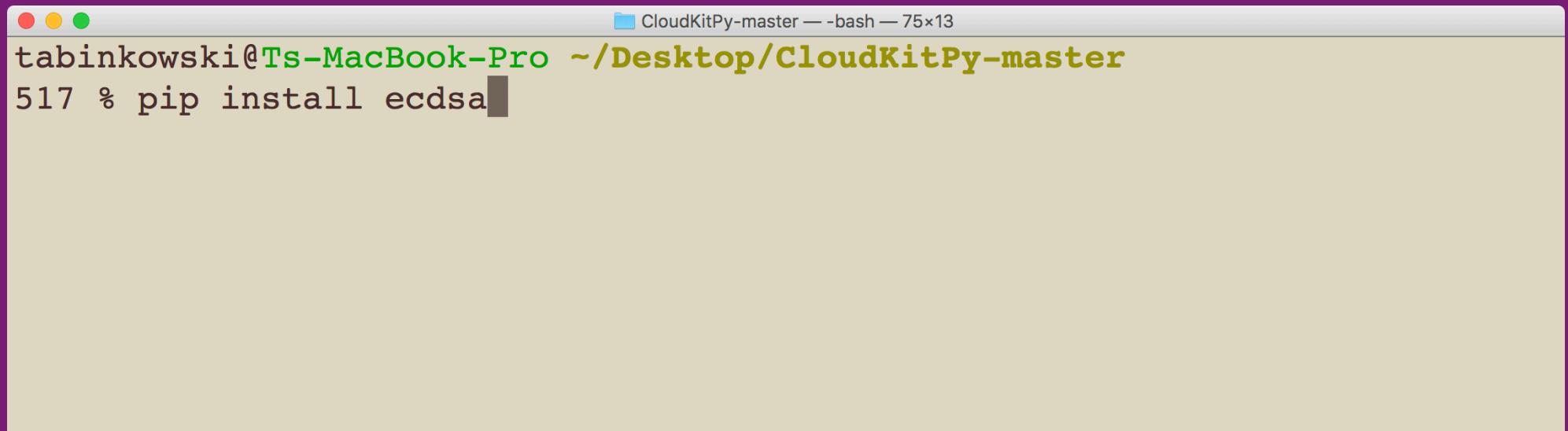
- Response is a dictionary with more information than you probably need

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. The title bar shows 'CloudKitPy-master — bash — 75x13'. The command line displays the user's path: 'tabinkowski@Ts-MacBook-Pro ~/Desktop/CloudKitPy-master'. Below the path, the command '517 % pip install ecdsa' is being typed. The background of the terminal is light beige.

- Elliptic Curve Digital Signature Algorithm used to sign requests
- Cryptography approach for public/private key agreement

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

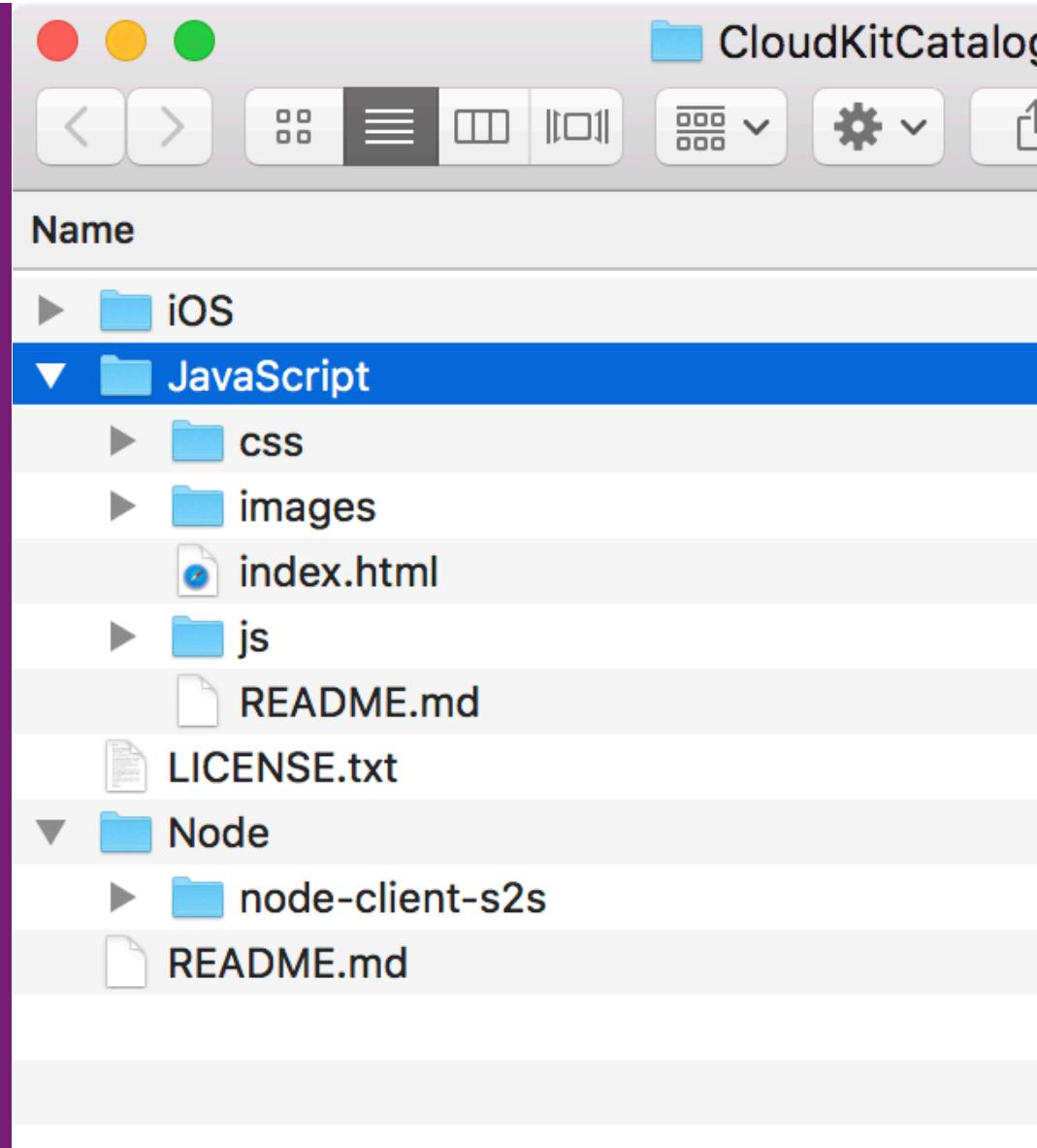
requests_cloudkit	bump version to 0.1.5
.gitignore	first commit
.travis.yml	add install script
LICENSE	first commit
Makefile	first commit
README.rst	Update README.rst
circle.yml	do python setup.py install for circle
requirements.txt	first commit
setup.cfg	first commit
setup.py	fix setup script
test_requests_cloudkit.py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

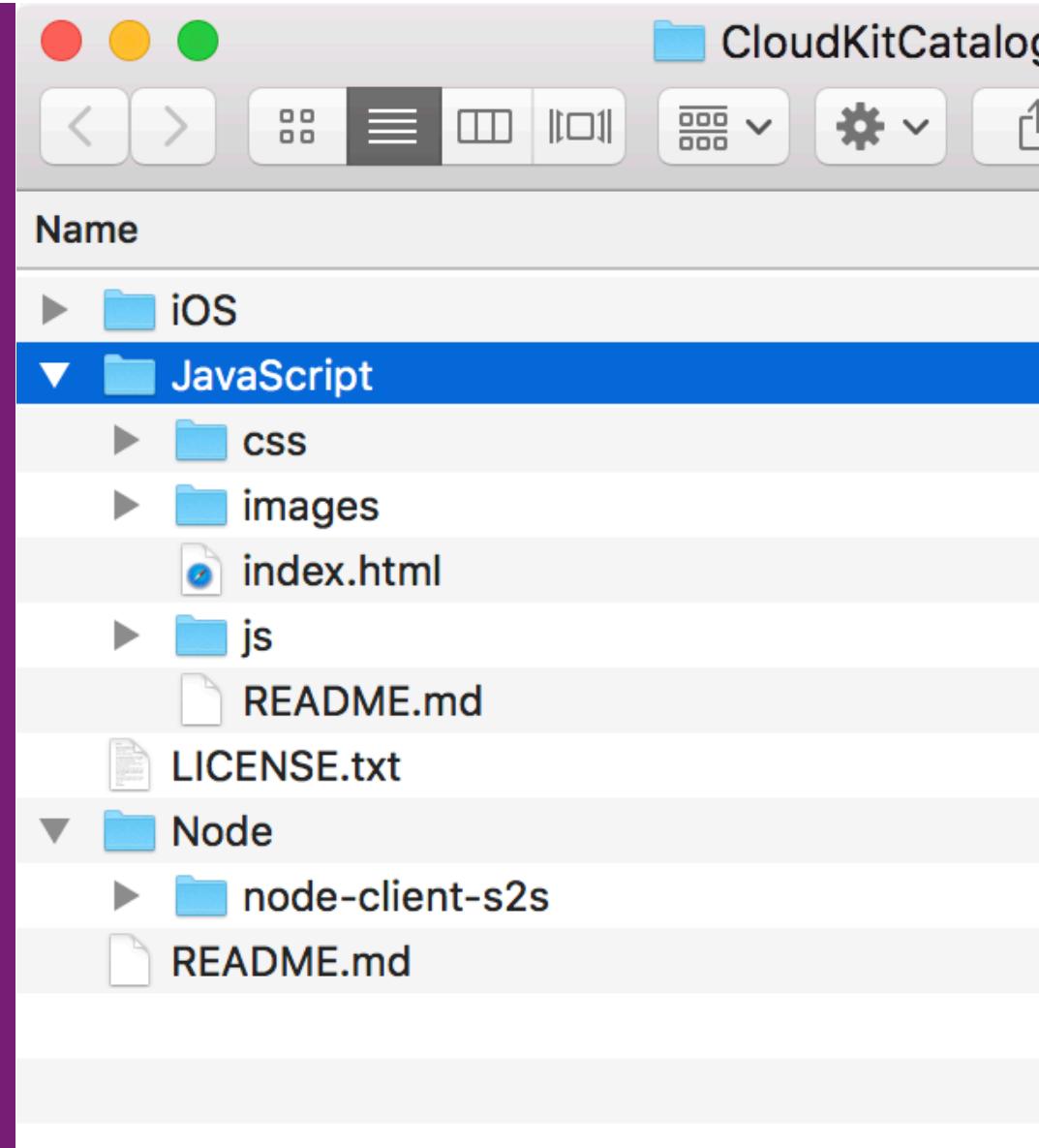
  var CloudKit = require('./cloudkit');
  var containerConfig = require('./config');

  // 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

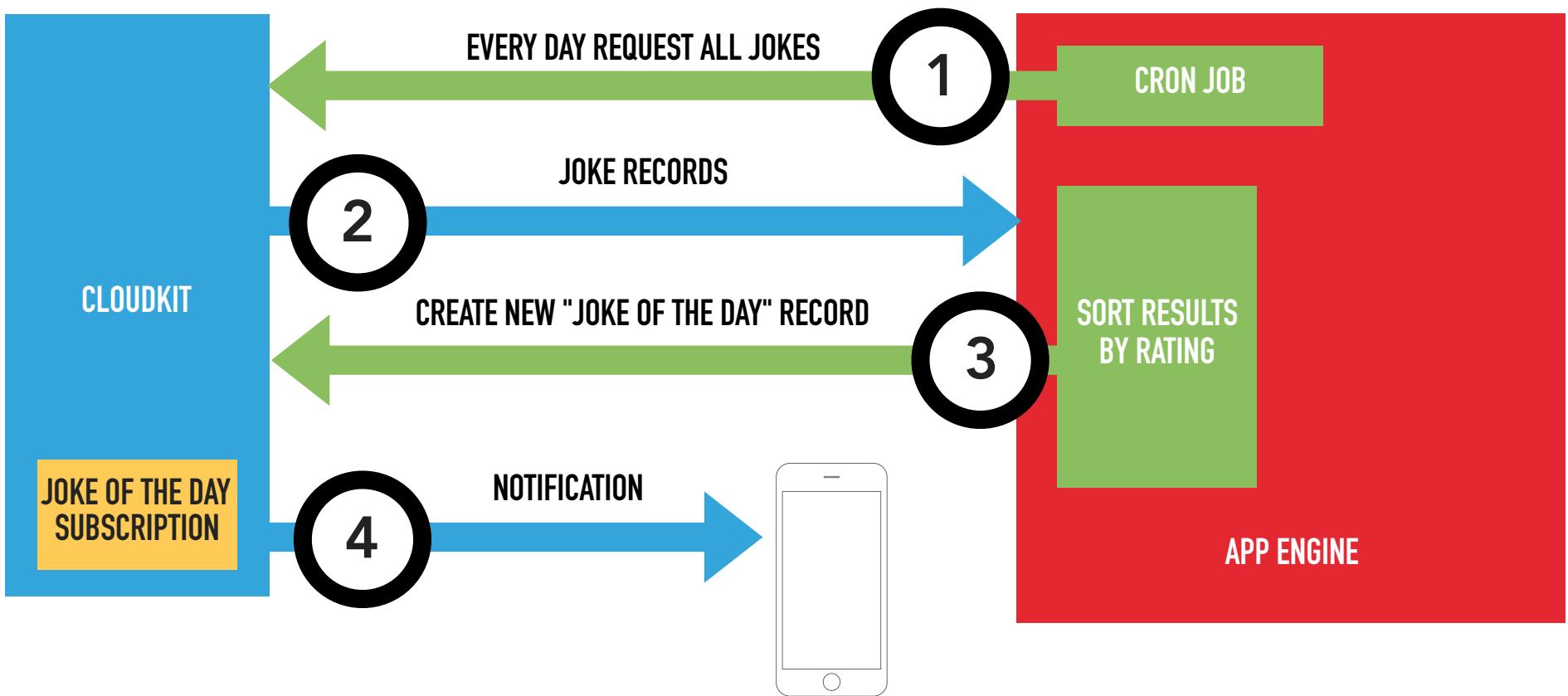
- Python so that we can use App Engine



CLOUDKIT HELPER



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

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

ZONES	RECORDS	RECORD TYPES	INDEXES	SUBSCRIPTIONS	SUBSCRIPTION TYPES	SECURITY ROLES
_defaultZone				Record Name Record T... Fields Ch. T... Created Modified		
USING:				▶ 08fa1c1e-7905... Daily day, joke j34o... Thu May 25 2... Thu May 25 2...		
				▶ 1B96F0DE-31E... Daily day j2ifp... Tue May 09 2... Tue May 09 2...		
				▶ 4063C911-1111... Daily day j34n... Thu May 25 2... Thu May 25 2...		
				▶ 4063C911-1111-4... Daily day j2ifo... Tue May 09 2... Tue May 09 2...		
				▶ 4063C911-1111-4... Daily day j34n... Thu May 25 2... Thu May 25 2...		
				▶ 4063C911-1111-4... Daily day j34n... Thu May 25 2... Thu May 25 2...		

SET UP A SUBSCRIPTION FOR THE DAILY "JOKE OF THE DAY"

Query Records

IOS

- Setup subscription for joke of the day
- Remember subscriptions are tied to individual users

```
//  
// 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 sub = 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 (trigger subscription)

```
# app.yaml
#
# Configuration for Google App Engine
# 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 AN EXTERNAL LIBRARY
IN YOUR PROJECT

APP ENGINE

- Installed in full in the same directory
- Allows you to use it as pure python code

```
— 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
|   |   — six.py
|   |   — six.pyc
|   |   — test_pyecdsa.py
|   |   — test_pyecdsa.pyc
|   |   — util.py
|   |   — util.pyc
|   — ecdsa-0.13.dist-info
|       — DESCRIPTION.rst
|       — INSTALLER
|       — METADATA
|       — RECORD
|       — WHEEL
|       — metadata.json
|       — top_level.txt
— main.py
```

APP ENGINE

- Add a new file:

appengine_config.py

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

APP ENGINE

- main.py
- Handlers
 - Processing
 - Posting record

```
import webapp2

import sys
import json

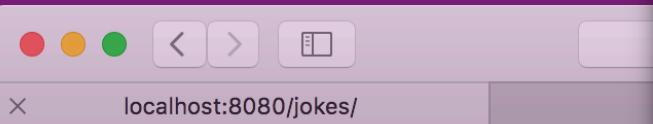
import cloudkit_helper as ck
from joke_of_the_day import JokeOfTheDay

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

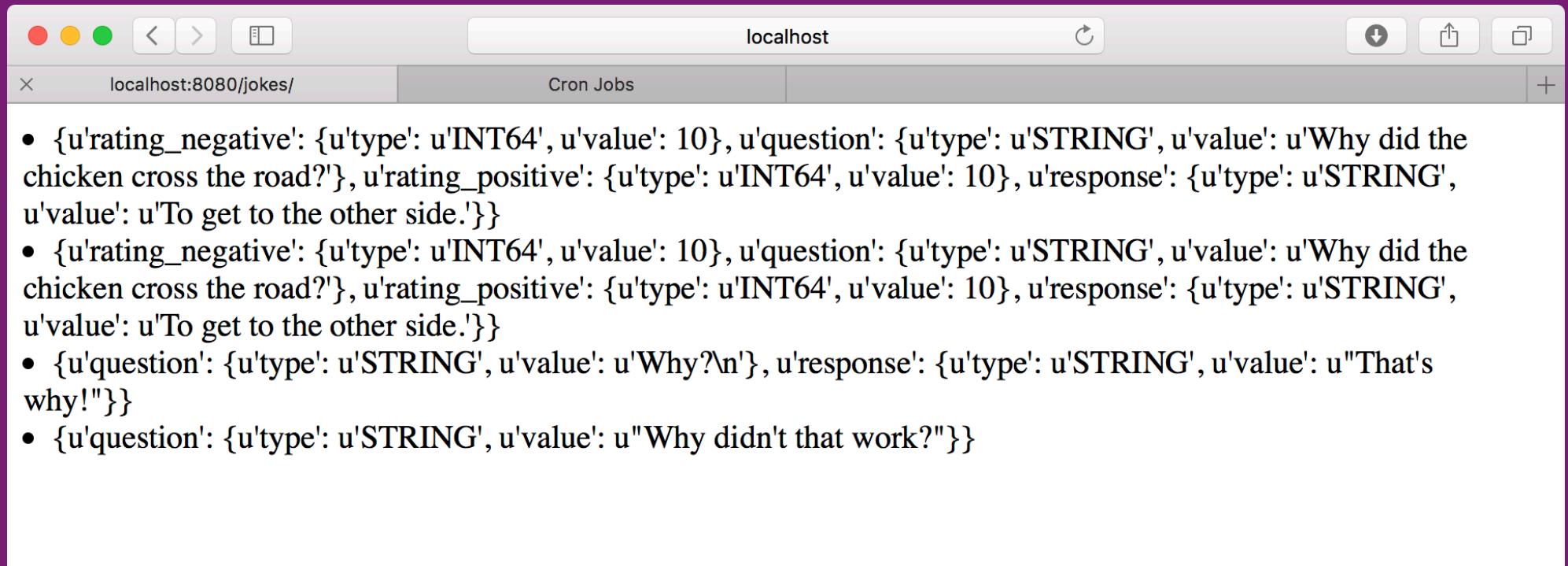


- {u'rating_negative': {u'type': u'INT64', u'value': 10}, u'question': {u'type': u'String', u'value': u'Why did the chicken cross the road?'}, u'rating_positive': {u'type': u'INT64', u'value': 10}, u'response': {u'type': u'String', u'value': u'To get to the other side.'}}
- {u'rating_negative': {u'type': 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"])
```

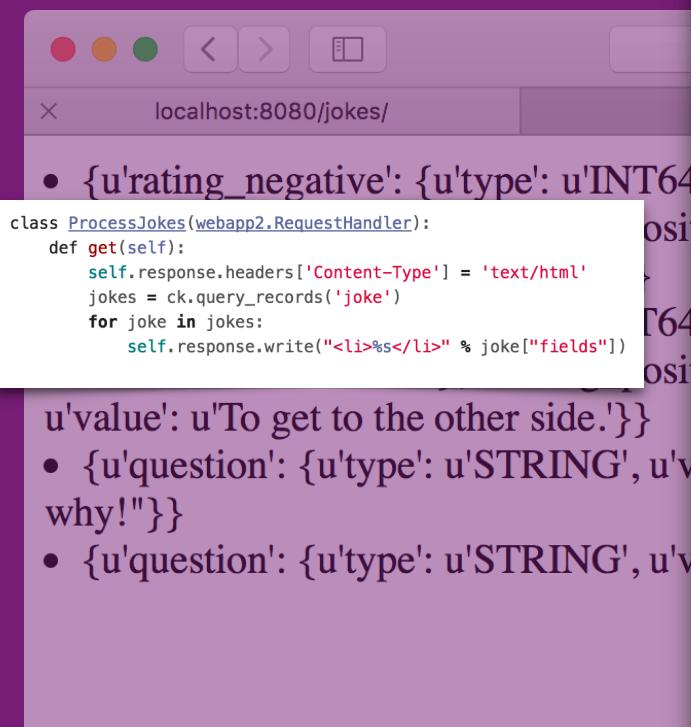
- <http://localhost:8080/jokes/>

APP ENGINE



- <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 HTML list format. The first item in the list is: "• {u'rating_negative': {u'type': u'INT64', u'value': u'To get to the other side.'}}

```
• {u'rating_negative': {u'type': u'INT64', u'value': u'To get to the other side.'}}
```

```
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/

```
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
```

REQUEST WITH
CURSORS

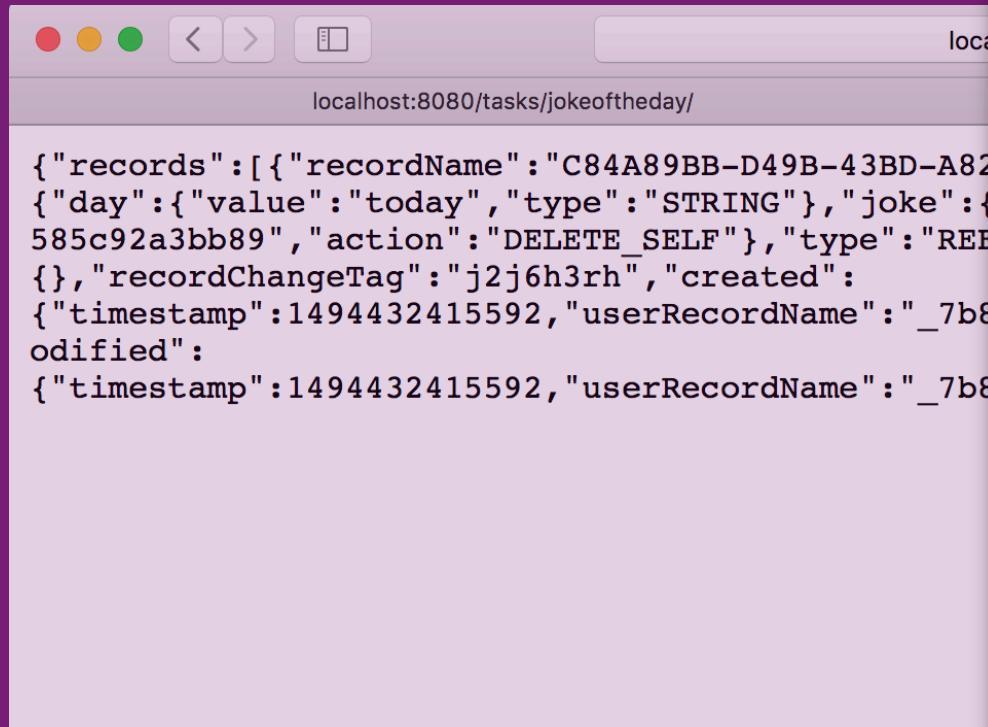
APP ENGINE

```
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



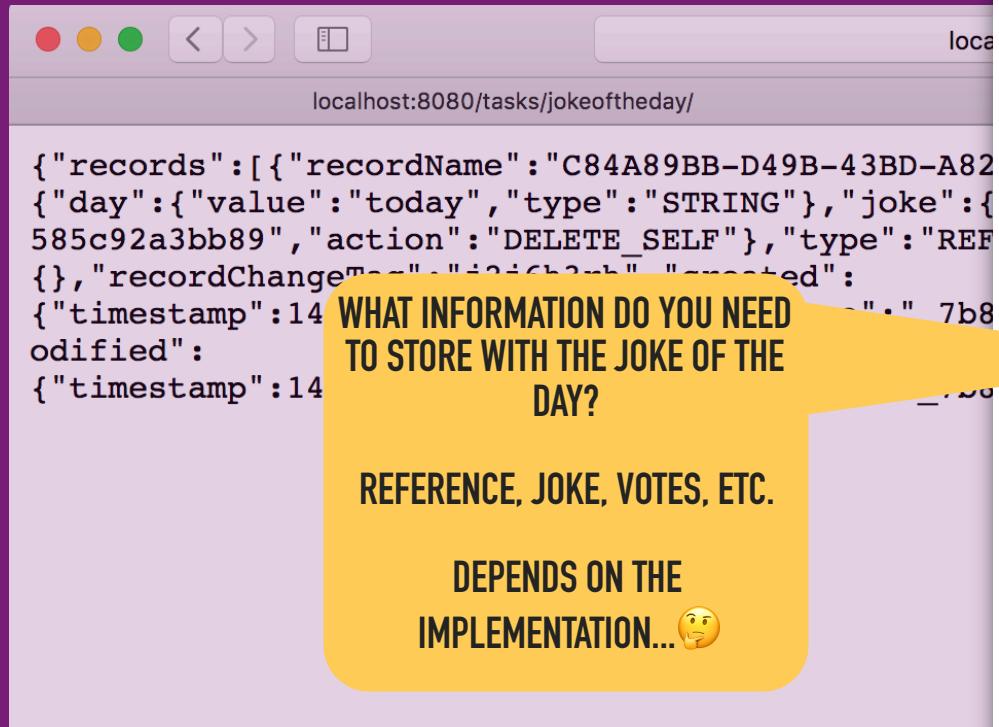
A screenshot of a web browser window. The address bar shows the URL `localhost:8080/tasks/jokeoftheday/`. The page content displays a JSON object with several records, including record IDs, day values, joke content, and timestamp information.

```
{"records": [{"recordName": "C84A89BB-D49B-43BD-A825-85c92a3bb89", "day": {"value": "today", "type": "STRING"}, "joke": {"text": "Why did the tomato turn red? Because it saw the salad dressing!"}, "recordChangeTag": "j2j6h3rh", "created": {"timestamp": 1494432415592}, "userRecordName": "_7b8odified": {"timestamp": 1494432415592, "userRecordName": "_7b8odified"}]}
```

- <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

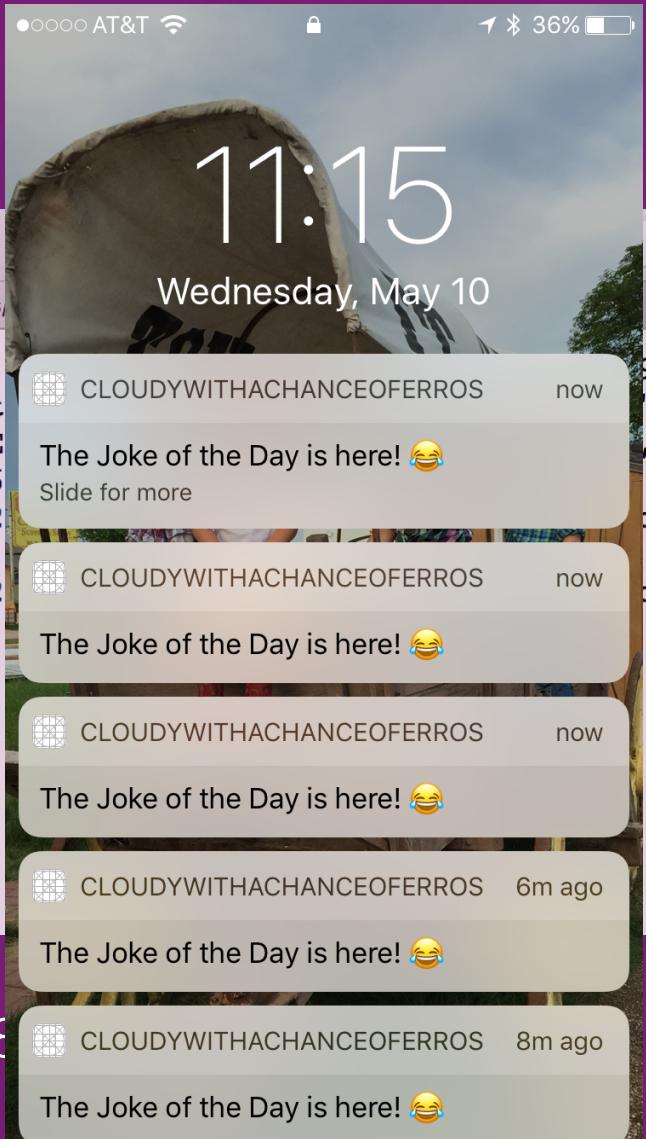


- `http://localhost:8080/tasks/jokeoftheday/`

```
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

```
localhost:8080/tasks/  
  
{"records": [{"recordName": "CloudKitCloud", "value": "Cloud"}, {"recordName": "CloudKitCloud", "value": "Cloud"}]}
```



- <http://localhost:8080>

```
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.  
    unsigned_data = ':' . join([date, body, web_service_url]).encode('utf-8')  
  
    # Sign payload via certificate.  
    signed_data = signing_key.sign(unsigned_data,  
                                    hashfunc=hashlib.sha256,  
                                    sigencode=ecdsa.util.sigencode_der)  
  
    signature = base64.b64encode(signed_data).decode('utf-8')  
  
    headers = {  
        'X-Apple-CloudKit-Request-KeyID': key_id,  
        'X-Apple-CloudKit-Request-ISO8601Date': date,  
        'X-Apple-CloudKit-Request-SignatureV1': signature  
    }  
  
    if data:  
        req_type = 'POST'  
    else:  
        req_type = 'GET'  
  
    result = curl('https://api.apple-cloudkit.com' + web_service_url,  
                 req_type=req_type,  
                 data=data,  
                 headers=headers)  
  
    return result
```

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

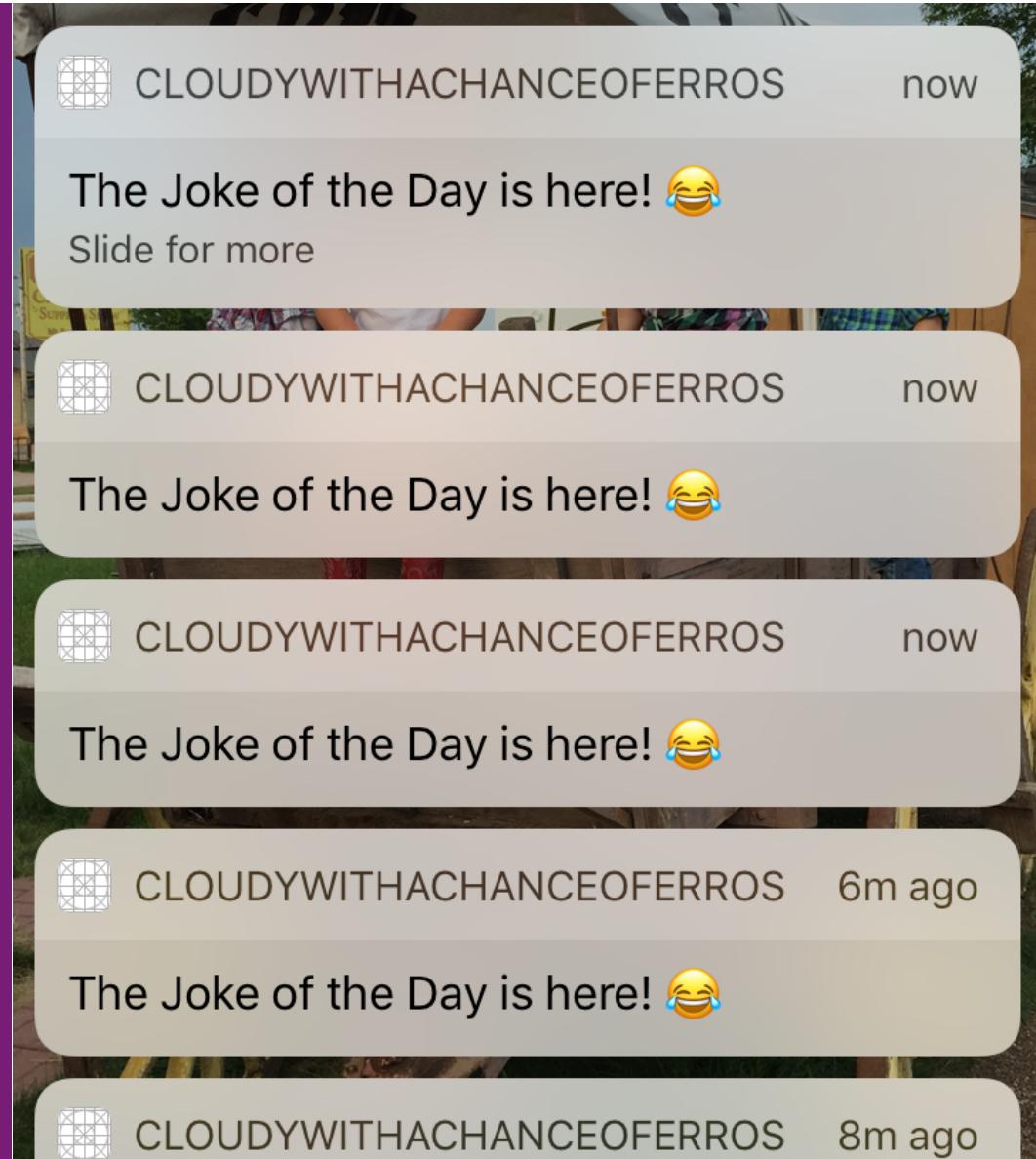
Request to /tasks/jokeoftheday/ succeeded!

Cron Job	Schedule	
/tasks/jokeoftheday/	every day 08:00	<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	

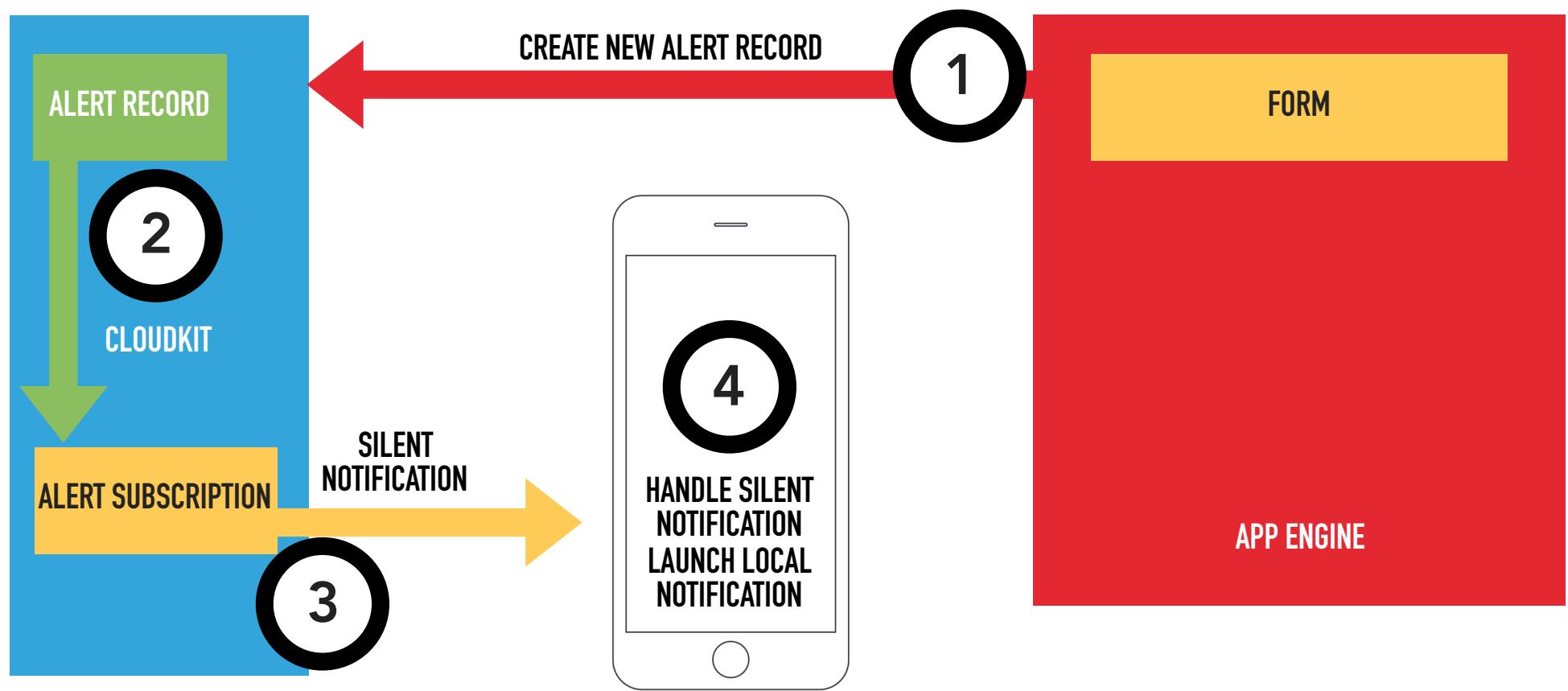
CUSTOM NOTIFICATIONS FROM SUBSCRIPTIONS

CUSTOM NOTIFICATIONS

- Generic notifications from subscriptions
- This may not be the great user experience we want



CUSTOM NOTIFICATIONS



CUSTOM NOTIFICATIONS



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

ZONES	RECORDS	RECORD TYPES	INDEXES	SUBSCRIPTIONS	SUBSCRIPTION TYPES	SECURITY ROLES	
DEFAULT TYPES		Field Name		Field Type	Indexes		
Users		message		String	Queryable, Searchable, ...		X
CUSTOM TYPES		recordName	SYSTEM FIELD	Reference	Queryable		
Alert		createdBy	SYSTEM FIELD	Reference	Queryable		
Daily joke		createdAt	SYSTEM FIELD	Date/Time	Queryable, Sortable		
		modifiedBy	SYSTEM FIELD	Reference	Queryable		
		modifiedAt	SYSTEM FIELD	Date/Time	Queryable, Sortable		
		changeTag	SYSTEM FIELD	String			

Record types will be automatically created in the development environment when you use the native API to create records of that type.

[Create New Type](#)

CUSTOM NOTIFICATIONS

- Set up a new route

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

WEB APP TO SEND
NOTIFICATIONS

CUSTOM NOTIFICATIONS

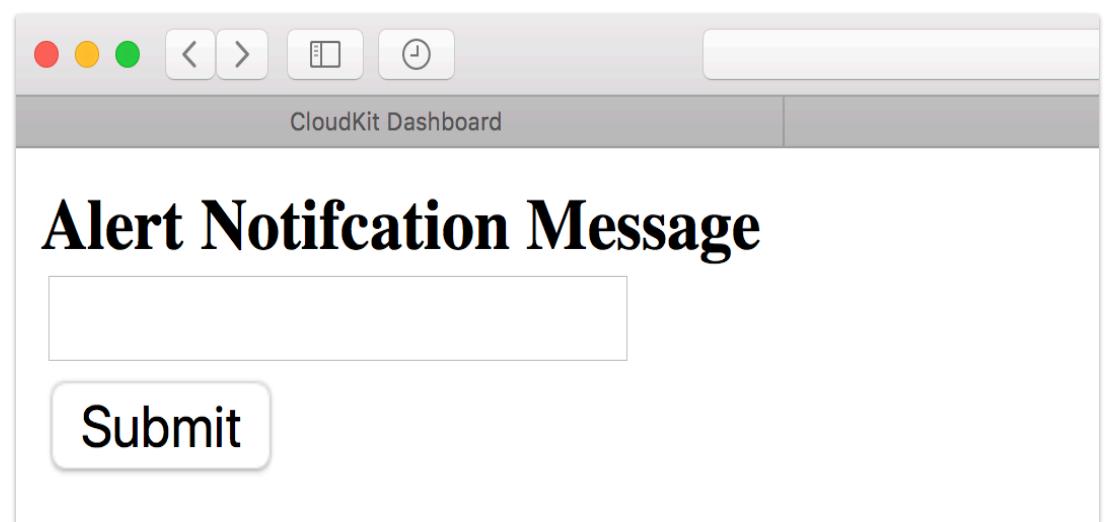
- Create a new Alerts record with the passed in message

```
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 that takes some text for the custom message

```
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)
```

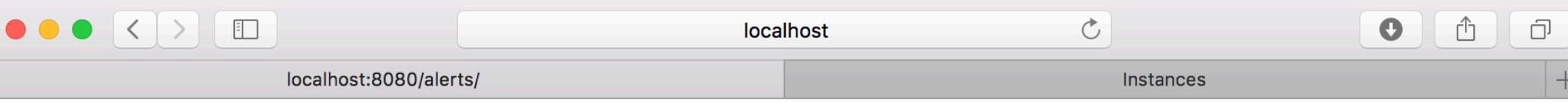


CUSTOM NOTIFICATIONS

- Create a new Alerts record
- Message (text from form) is passed as a field

```
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'])
```

CUSTOM NOTIFICATIONS



A screenshot of a web browser window titled "localhost". The address bar shows "localhost:8080/alerts/". The main content area displays a JSON object:

```
{"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"}}]} 
```

The code block below shows the Python code used to generate this response:

```
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']) 
```

A yellow callout bubble with an arrow pointing to the right contains the text "DUMP RESPONSE TO SCREEN".

CUSTOM NOTIFICATIONS

```
{  
  "records" : [ {  
    "recordName" : "AF528050-E146-4B5D-95C3-37289C6FD60B",  
    "recordType" : "Alert",  
    "fields" : {  
      "message" : {  
        "value" : "Does this work &#128520;",  
        "type" : "STRING"  
      }  
    },  
    "pluginFields" : { },  
    "recordChangeTag" : "j9ykilht",  
    "created" : {  
      "timestamp" : 1510600468839,  
    }  
  }]
```

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.Cloudy";  
    fo = 1; // chanceOfErrors  
    nid = "2b661f39-5617-4a1c-a880-cd9eae7acd9eae7a";  
    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)
}
}
```

BREAK TIME



OPEN SOURCE SWIFT

OPEN SOURCE SWIFT

Dec 3, 2015

Swift is Open Source

Swift is now open source. Today Apple launched the open source Swift community, as well as amazing new tools and resources including:

- [Swift.org](#) – a site dedicated to the open source Swift community
- Public source code repositories at [github.com/apple](#)
- A new Swift package manager project for easily sharing and building code
- A Swift-native core libraries project with higher-level functionality above the standard library
- Platform support for all Apple platforms as well as Linux

Now anyone can download the code and in-development builds to see what the team is up to. More advanced developers interested in contributing to the project can file bugs, participate in the community, and contribute their own fixes and enhancements to make Swift even better. For production App Store development you should always use the stable releases of Swift included in Xcode, and this remains a requirement for app submission.

- It all started here

Swift.org

Swift.org is an entirely new site dedicated to open source Swift. This site hosts resources for the community of developers that want to help evolve Swift, contribute fixes, and most importantly, interact with each other. Swift.org hosts:

- A bug reporting and tracking system
- Mailing lists
- A blog dedicated to the engineering of Swift
- Community guidelines
- Getting started tutorials
- Contributing instructions
- Documentation on Swift

[View the site](#)

OPEN SOURCE SWIFT

- The promise
 - Same code runs in both places
 - Reduce development time by sharing code
 - Leverage (some) frameworks and APIs



OPEN SOURCE SWIFT



- Something is missing...

OPEN SOURCE SWIFT

- Linux and Mac Platform support
- Standard Library Foundation, Dispatch, and XCTest Compiler
- Command Line Tools



Swift

ABOUT SWIFT

BLOG

DOWNLOAD

GETTING STARTED

DOCUMENTATION

SOURCE CODE

COMMUNITY

CONTRIBUTING

CONTINUOUS
INTEGRATION

PROJECTS

COMPILER AND
STANDARD LIBRARY

PACKAGE MANAGER

CORE LIBRARIES

REPL AND DEBUGGER

Welcome to Swi

Swift is now open source!

We are excited by this new chapter in the story of Swift. When Apple unveiled the Swift programming language, it quickly became one of the fastest growing languages in history. Swift is a modern, safe, and efficient software that is incredibly fast and safe by design. Now that Swift is open source, you can help make the best general-purpose programming language available everywhere.

For students, learning Swift has been a great introduction to modern programming concepts and best practices. Now that Swift is open source, their Swift skills will be able to be applied to a broader range of platforms, from mobile devices to the web to the cloud.

Welcome to the Swift community. Together we can build a better programming language for everyone.

– **The Swift Team**

STATE OF SWIFT ON THE SERVER

STATE OF SWIFT ON THE SERVER

- Many Swift web frameworks in development
- Most popular (15,000+ stars)



STATE OF SWIFT ON THE SERVER

- Perfect
 - A complete framework
 - 11,448 ★
 - Rails inspired

The screenshot shows the homepage of perfect.org. At the top, there is a navigation bar with links for "WHAT IS PERFECT?", "DOCUMENTATION", "DEVELOPER RESOURCES", and "EVENTS". Social media icons for LinkedIn, Twitter, Facebook, YouTube, and GitHub are also present, along with a search icon. Below the navigation, there are four main sections: "Get Started" (with an orange bird icon), "Documentation" (with an orange open book icon), "Assistant" (with an orange laptop icon), and "Commercial" (with an orange hexagonal icon). Each section has a brief description and a call-to-action button at the bottom. A large orange banner at the bottom features a play button icon and the text "TAKE ME TO THE LEARNING CENTRE". Another banner to the right encourages users to "Show me Ray Wenderlich's video tutorials >>". The central part of the page contains the heading "What is Perfect?" and a brief description of the service.

perfect.org

WHAT IS PERFECT? DOCUMENTATION DEVELOPER RESOURCES EVENTS

Get Started Documentation Assistant Commercial

Access Perfect's feature set and start your project.

Get support from our comprehensive set of documentation.

Meet the Perfect Assistant to help you do more with Perfect and Swift, more easily.

Use Perfect to build apps for business.

Get Started >> Documentation >> Meet the Assistant >> Find Out How >>

TAKE ME TO THE LEARNING CENTRE

Show me Ray Wenderlich's video tutorials >>

What is Perfect?

Perfect is a web server and toolkit for developers using the Swift programming language to build applications and other REST services. It lets developers build using only Swift to program both the client-facing and server-side of their

STATE OF SWIFT ON THE SERVER

- Perfect
 - A complete framework
 - Runs its own server or as a fastCGI module for Apache or NGINX

```
import PerfectLib

public func PerfectServerModuleInit() {
    Routing.Handler.registerGlobally()

    Routing.Routes["/] = { _ in return HelloWorldHandler() }

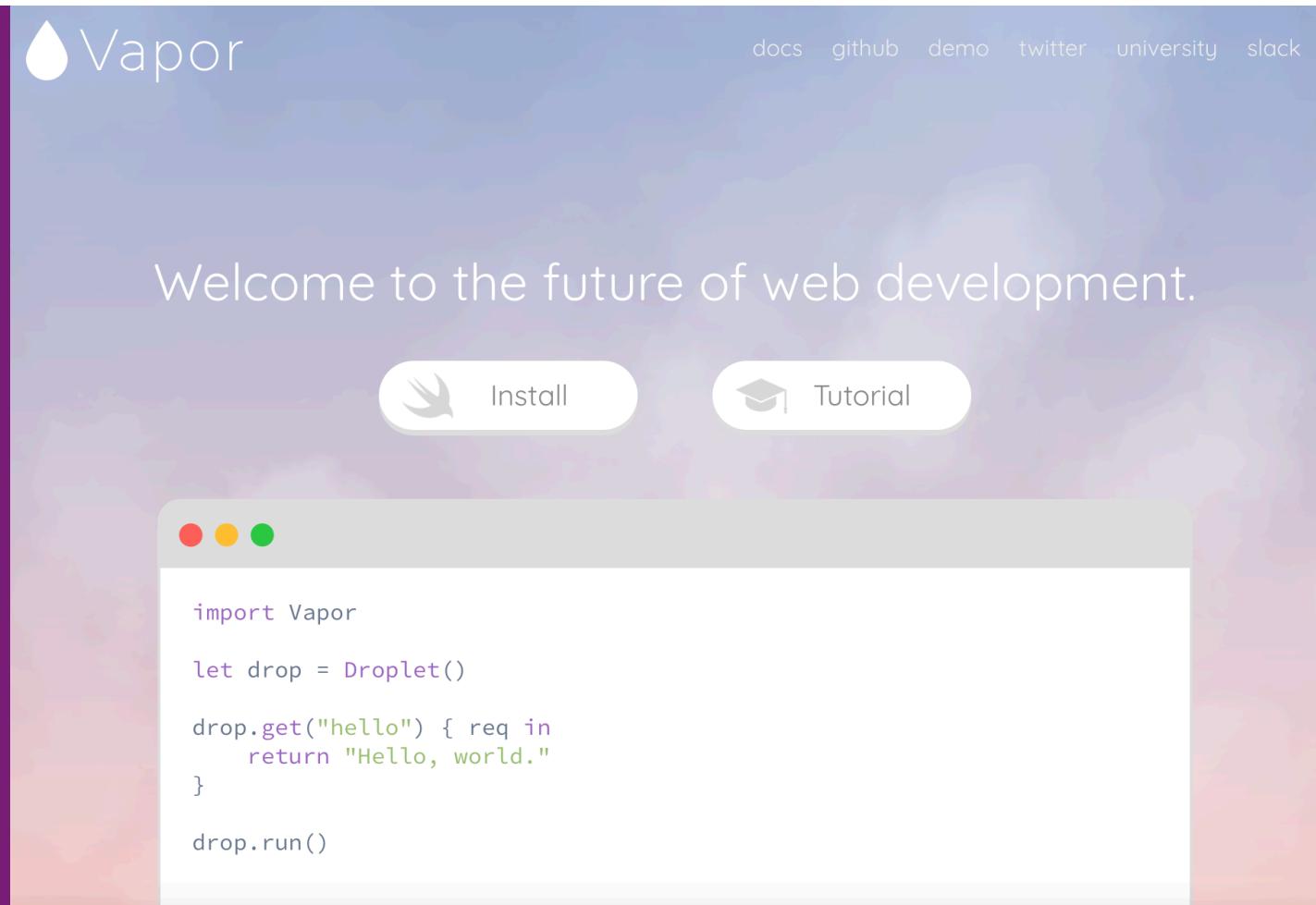
}

class HelloWorldHandler: RequestHandler {
    func handleRequest(request: WebRequest, response: WebResponse) {
        response.appendBodyString("Hello, World!")
        response.requestCompletedCallback()
    }
}
```

STATE OF SWIFT ON THE SERVER

- Vapor

- 9,479 ★
- Laravel (php)
inspired



The image shows the Vapor website homepage. At the top left is the Vapor logo (a white water droplet icon) followed by the word "Vapor". To the right are links for "docs", "github", "demo", "twitter", "university", and "slack". Below the header is a large, semi-transparent text box containing the slogan "Welcome to the future of web development." In the center of this box are two buttons: "Install" with a bird icon and "Tutorial" with a graduation cap icon. At the bottom of the page is a code editor window displaying Swift code:

```
import Vapor

let drop = Droplet()

drop.get("hello") { req in
    return "Hello, world."
}

drop.run()
```



STATE OF SWIFT ON THE SERVER

- Perfect
 - A complete framework
 - Runs its own server or as a fastCGI module for Apache or NGINX

```
import Vapor

let app = Application()

app.get("/") { request in
    return "Hello, World!"
}

app.start()
```

STATE OF SWIFT ON THE SERVER

- Kitura
 - Swift@IBM
 - 5,686 ★
 - Express.js inspired

The image shows a person's hands typing on a laptop keyboard. In the background, a computer monitor displays the official Kitura website. The website features a logo of a bird in flight over clouds. The navigation bar includes links for "Getting started", "Tutorials", "API reference", and "Support". Below the navigation, there is a screenshot of a Mac OS X application window titled "Cloud Runtimes" showing three running projects: "Kitura-Runtime", "Bluepic-Sample-Runtime", and "ToDo-Server", each with a green "Running" status indicator. A large blue button at the bottom right of the website says "Get started with Kitura".

STATE OF SWIFT ON THE SERVER

- Kitura
 - Swift@IBM
 - 5,686 ★

```
import Kitura

let router = Router()

router.get("/") { request, response, next in
    response.send("Hello, World!")
    next()
}

Kitura.addHTTPServer(onPort: 8090, with: router)
Kitura.run()
```

STATE OF SWIFT ON THE SERVER

- Server Side Swift Standards (S4) project
- Protocols and standards

open-swift / S4

Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs

HTTP standards for Swift

264 commits 2 branches 27 releases 12 contributors

Branch: master New pull request Create new file Upload files Find file Clone

noppoMan committed with paulofaria Updating to September 10, 2016 (#78) Latest commit a544ad7 on

File	Last Commit	Author(s)
Sources	Updating to September 10, 2016 (#78)	8
Tests	Updating to August 4, 2016	9
.gitignore	Update to C7 0.4.0	
.swift-version	Updating to September 10, 2016 (#78)	8
.travis.yml	Updated Swift to 07-25 (#74)	10
LICENSE	initial setup	
Package.swift	Updating to August 4, 2016	9
README.md	Updated Swift to 07-25 (#74)	10
S4.podspec	add cocoapods support	

README.md

S4 - HTTP

STATE OF SWIFT ON THE SERVER

- Server Side Swift Standards (S4) project
 - Protocols and standards
- Apple has a working group dedicated to swift on the server

VAPOR

VAPOR

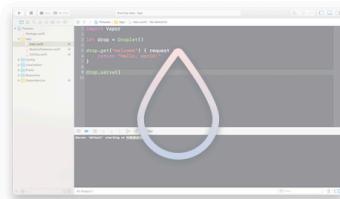
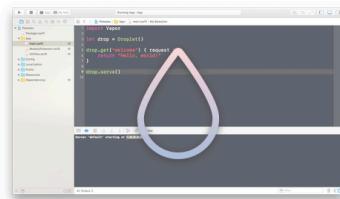
- Web framework and server for Swift
- Works on macOS and Ubuntu

The screenshot shows the GitHub repository page for 'vapor / vapor'. The repository has 3,196 commits, 6 branches, 172 releases, and 77 contributors. The latest commit was made 6 hours ago. The repository uses the MIT license.

File	Description	Time Ago
.Documents	Adding another break line	3 months ago
.Sources	Sessions Middleware doesn't require Foundation anymore	10 hours ago
.Tests	Merge branch 'master' into sessionsMiddlewareImprovements	8 hours ago
.Utilities	allow xcode 8.1	8 months ago
.codecov.yml	remove unnecessary files	a day ago
.gitignore	Make sure sessions cookie is HTTP only	14 hours ago
.travis.yml	prevent double provider boot	a month ago
ISSUE_TEMPLATE.md	Update ISSUE_TEMPLATE.md	a month ago
LICENSE	add MIT License	a year ago
Package.swift	remove perf target	15 days ago
README.md	text updates	13 hours ago
circle.yml	prevent double provider boot	a month ago

VAPOR

- Great documentation and resources



vapor.university



Look Maa! Server Side Swift Using Vapor

A detailed introduction to using Vapor and a SQLite database.

15m • Easy • Mohammad Azam

Social Authentication

Learn how to implement user authentication using Facebook and Google.

8m • Intermediate • Caleb Kleveter

User Authentication

Learn how to authenticate users using Vapor's Auth module and PostgreSQL.

10m • Intermediate • Caleb Kleveter

GETTING STARTED

[Install Swift 3: macOS](#)[Install Swift 3: Ubuntu](#)[Install Toolbox](#)[Hello, World](#)[Manual](#)[Xcode](#)

GUIDE

[Droplet](#)[Folder Structure](#)[JSON](#)[Config](#)[Views](#)[Leaf](#)

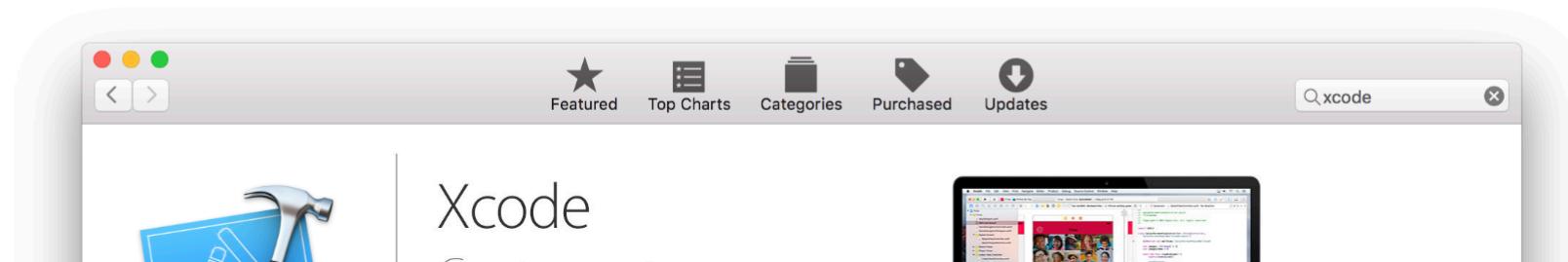
Install Swift 3: macOS

[!\[\]\(084f060b1338fbad40d1b9ab097af275_img.jpg\) Edit on GitHub](#)

To use Swift 3 on macOS, you just need to have Xcode 8 installed.

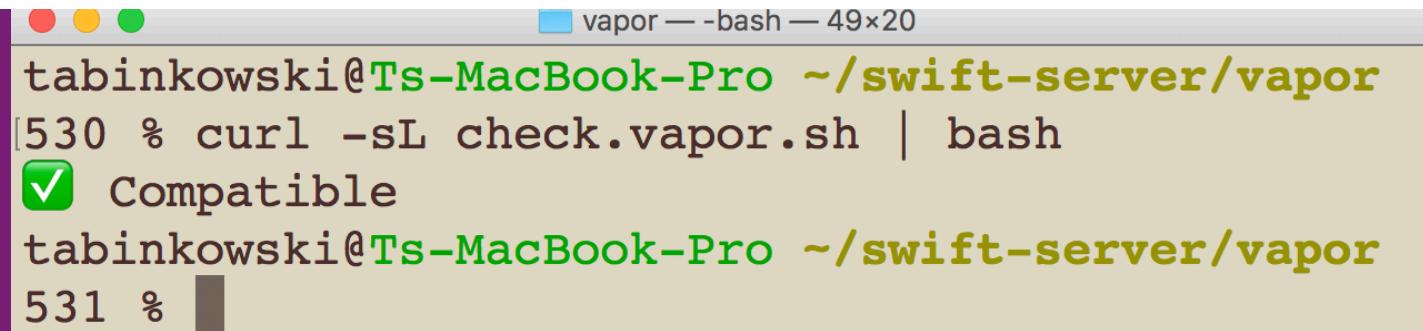
Install Xcode

Install [Xcode 8](#) from the Mac App Store.



VAPOR

- Install Xcode
- Check version for vapor



A screenshot of a macOS terminal window titled "vapor — -bash — 49x20". The window shows the following command and its output:

```
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor
[530 % curl -sL check.vapor.sh | bash
✓ Compatible
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor
531 % ]
```

The terminal uses color coding: green for host and path, grey for command, brown for output, and black for the prompt.

VAPOR

- Toolbox provides command line and shortcuts for common tasks
- Install with homebrew 🍺

```
vapor — git-remote-https • brew install vapor/tap/vapor — 49×20
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor
534 % brew install vapor/tap/vapor
```

VAPOR

- Toolbox provides command line and shortcuts for common tasks
- Install with homebrew 

```
vapor -- bash -- 50x20
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor
542 % vapor --help
[Usage: vapor <new|build|update|run|fetch|clean|test|xcode|version|self|heroku>
Join our Slack if you have questions, need help,
or want to contribute: http://vapor.team
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor
543 % ]
```

VAPOR

```
vapor — -bash — 80x20
[ tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor
542 % vapor --help
Usage: vapor <new|build|update|run|fetch|clean|test|xcode|version|self|heroku>
Join our Slack if you have questions, need help,
or want to contribute: http://vapor.team
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor
543 % ]
```

VAPOR

```
# The toolbox can update itself. This may be useful if # you
experience any issues in the future.
vapor self update

# Templates
# The toolbox can create a project from the Vapor basic-
template or any other git repo.
vapor new <name> [--template=<repo-url-or-github-path>]
```

VAPOR

- `vapor new vapor-hello-world`
 - Build a new project from basic templates
 - Can specify other templates

```
vapor — bash — 50x20
* * ~~~~~ * *
* * ~~~~~ * *
* * ~~~~~ * *
* * ~~~~~ * *
* * ~~~~~ * *
* * ~~~~~ * *
* * ~~~~~ * *
* * ~~~~~ * *
* * ~~~~~ +++++ * *
* * ~~~~~ +++++ * *
*** ~~~~~ +++++ *** *
**** ~~~~~ +++++ **** *
***** ~~~~~ ***** *
***** ~***** *
```

\ \ / / \ \ / \ [P) / \ \ / [R)
a web framework for Swift

VAPOR

- `vapor new vapor-hello-world`
- Managing vapor projects
 - Xcode
 - Swift Package Manager for other builds

```
vapor-hello-world — swift-package ▾ vapor xcode — 50×20
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor
[557 % cd vapor-hello-world/
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
[558 % vapor xcode
No .build folder, fetch may take a while...
Fetching Dependencies [• ]
```

VAPOR

BUILDS A FRAMEWORK AND APP

The screenshot shows a Mac OS X desktop with an Xcode project window open. The title bar says "vapor-hello-world: Ready | Today at 9:47 PM". The left sidebar shows the project structure for "vapor-hello-world" with files like Package.swift, main.swift, and various models and controllers. The main editor area displays the following Swift code:

```
import PackageDescription

let package = Package(
    name: "vapor-hello-world",
    dependencies: [
        .Package(url: "https://github.com/vapor/vapor.git", majorVersion: 1, minor: 5)
    ],
    exclude: [
        "Config",
        "Database",
        "Localization",
        "Public",
        "Resources",
    ]
)
```

A yellow speech bubble points from the word "BUILDS" in the header to the ".Package" line in the code. Another yellow speech bubble points from the word "USES" in the footer to the "Dependencies" section of the code.

VAPOR

The screenshot shows the Xcode interface with the following details:

- Project Navigator:** Shows the project structure for "vapor-hello-world". It includes a "Sources" folder containing "App" (with "main.swift", "Controllers", and "Models"), "Tests", "Config", "Localization", "Public", and "Resources". A yellow callout bubble labeled "DEPENDENCIES" points to the "Dependencies" folder.
- File Navigator:** Shows the "Package.swift" file selected.
- Editor:** Displays the contents of "Package.swift":

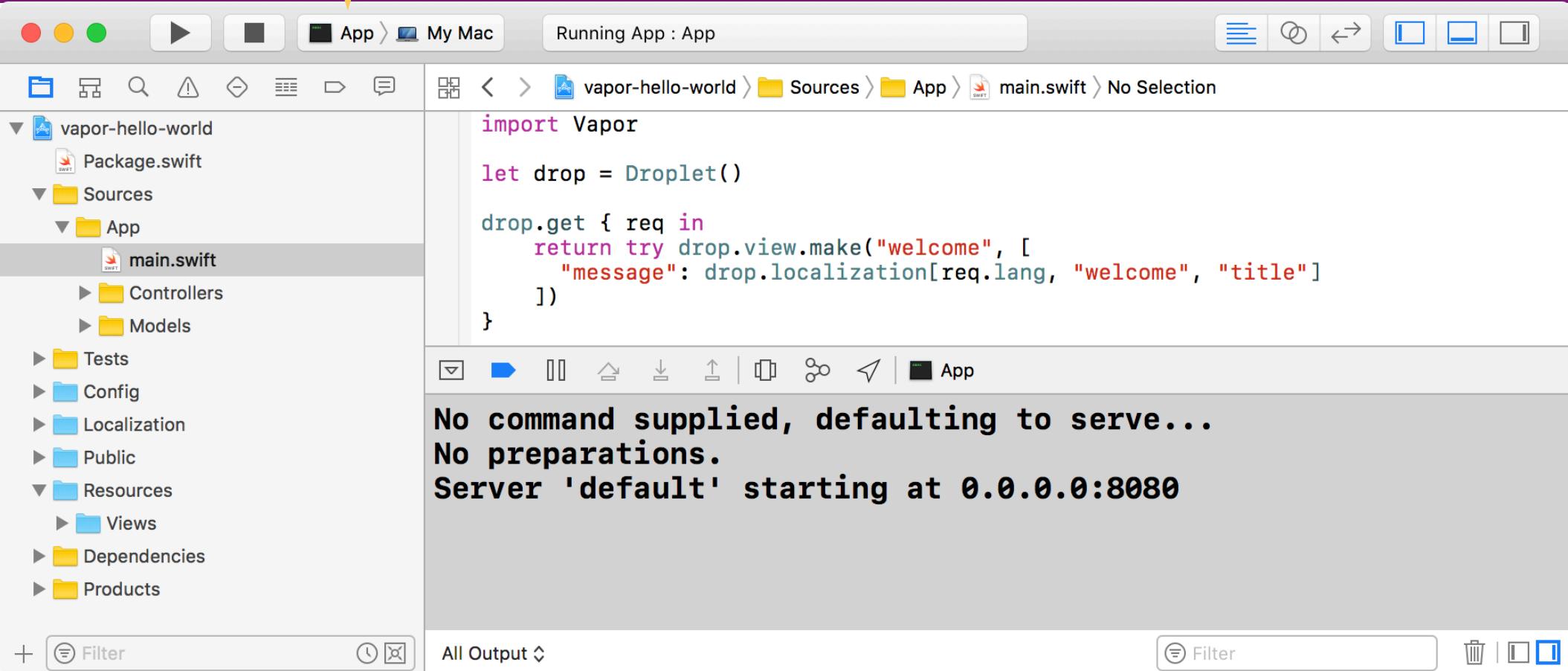
```
import PackageDescription

let package = Package(
    name: "vapor-hello-world",
    dependencies: [
        .Package(url: "https://github.com/vapor/vapor.git", majorVersion: 1, minor: 5)
    ],
    exclude: [
        "Config",
        "Database",
        "Localization",
        "Public",
        "Resources",
    ]
)
```

The status bar at the top indicates "Building vapor-hello-world: Cipher | Copying swiftdocs".

VAPOR

RUN APP



A screenshot of a Mac OS X desktop environment showing a terminal window. The window title is "Running App : App". The terminal output is as follows:

```
import Vapor

let drop = Droplet()

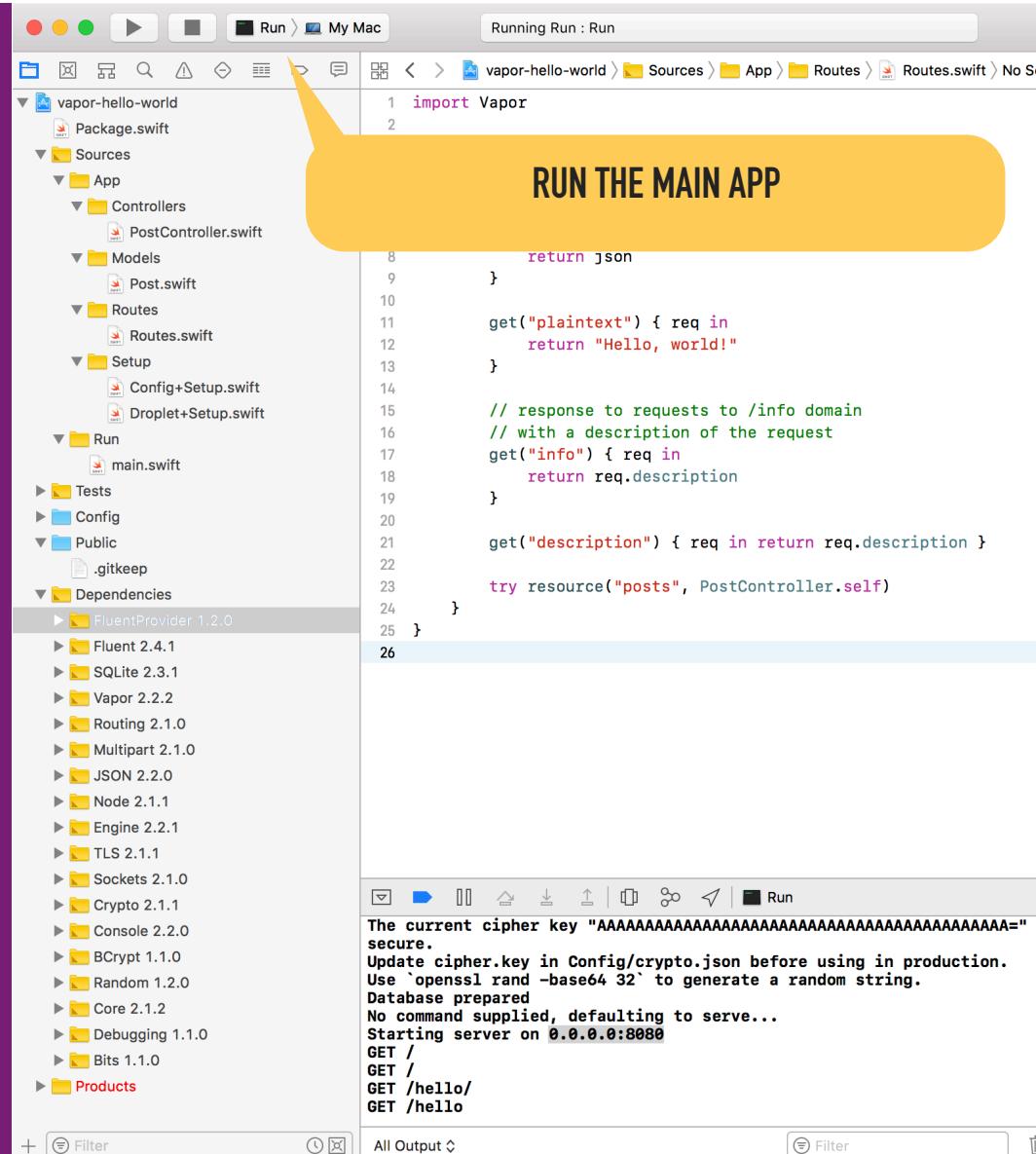
drop.get { req in
    return try drop.view.make("welcome", [
        "message": drop.localization[req.lang, "welcome", "title"]
    ])
}

No command supplied, defaulting to serve...
No preparations.
Server 'default' starting at 0.0.0.0:8080
```

The left sidebar shows the project structure of "vapor-hello-world" with files like Package.swift, main.swift, and various models/controllers.

VAPOR

- Builds all the package dependencies and links them



The current cipher key "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=" is secure.
Update cipher.key in Config/crypto.json before using in production.
Use `openssl rand -base64 32` to generate a random string.
Database prepared
No command supplied, defaulting to serve...
Starting server on 0.0.0.0:8080
GET /
GET /
GET /hello/
GET /hello

RUN THE MAIN APP

```
import Vapor

get("plaintext") { req in
    return "Hello, world!"
}

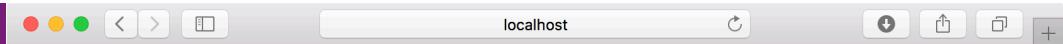
// response to requests to /info domain
// with a description of the request
get("info") { req in
    return req.description
}

get("description") { req in
    return req.description
}

try resource("posts", PostController.self)
```

VAPOR

- Server runs on [http://localhost:
8080](http://localhost:8080)



VAPOR

- File structure is designed to make testing easier 🙌

The image shows a file browser view of a Vapor project named "vapor-hello-world". The project structure is as follows:

- Project root: vapor-hello-world
 - Package.swift
 - Sources
 - App
 - Controllers (PostController.swift)
 - Models (Post.swift)
 - Routes (Routes.swift)
 - Setup (Config+Setup.swift, Droplet+Setup.swift)
 - Run (main.swift)
 - Tests
 - Config
 - Public (.gitkeep)
 - Dependencies
 - Products

On the right, a snippet of the "Package.swift" file is shown, highlighting the "Sources" section:

```
1 import Vapo
2
3 extension D
4     func se
5         get
6
7
8
9
10
11     get
12
13 }
14
15 // 
16 // 
17 get
18
19 }
20
21 get
22
23 try
24
25 }
26
```

VAPOR

The screenshot shows a Mac OS X desktop with an Xcode window open. The title bar says "Running App : App". The file browser sidebar shows a project structure for "vapor-hello-world" with files like "Package.swift", "main.swift", "Controllers", "Models", "Tests", "Config", "Localization", "Public", "Views", "Dependencies", and "Products". The main editor area contains the following Swift code:

```
import Vapor

let drop = Droplet()

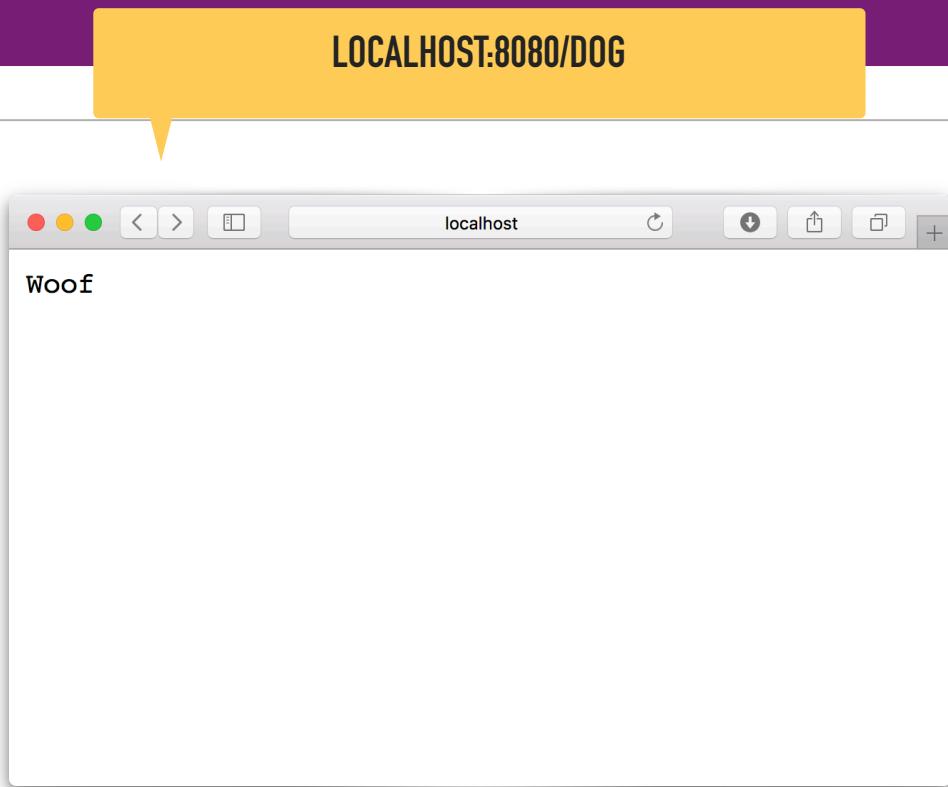
drop.get { req in
    return try JSON(node:
        ["message": "Hello Vapor"]
    )
}

drop.run()
```

A small preview window in the bottom right corner shows the output of the application: {"message": "Hello Vapor"}. The status bar at the bottom of the Xcode window shows various icons and the text "localhost".

VAPOR

```
vapor-hello-world > Sources > App > main.swift > No Selection  
import Vapor  
  
let drop = Droplet()  
  
drop.get { req in  
    return try JSON(node:  
        ["message": "Hello Vapor"]  
    )  
}  
  
drop.get("dog") { req in  
    return "Woof"  
}  
  
drop.get("dog", "speak") { req in  
    return "Bark! Bark!"  
}  
  
drop.run()
```



VAPOR

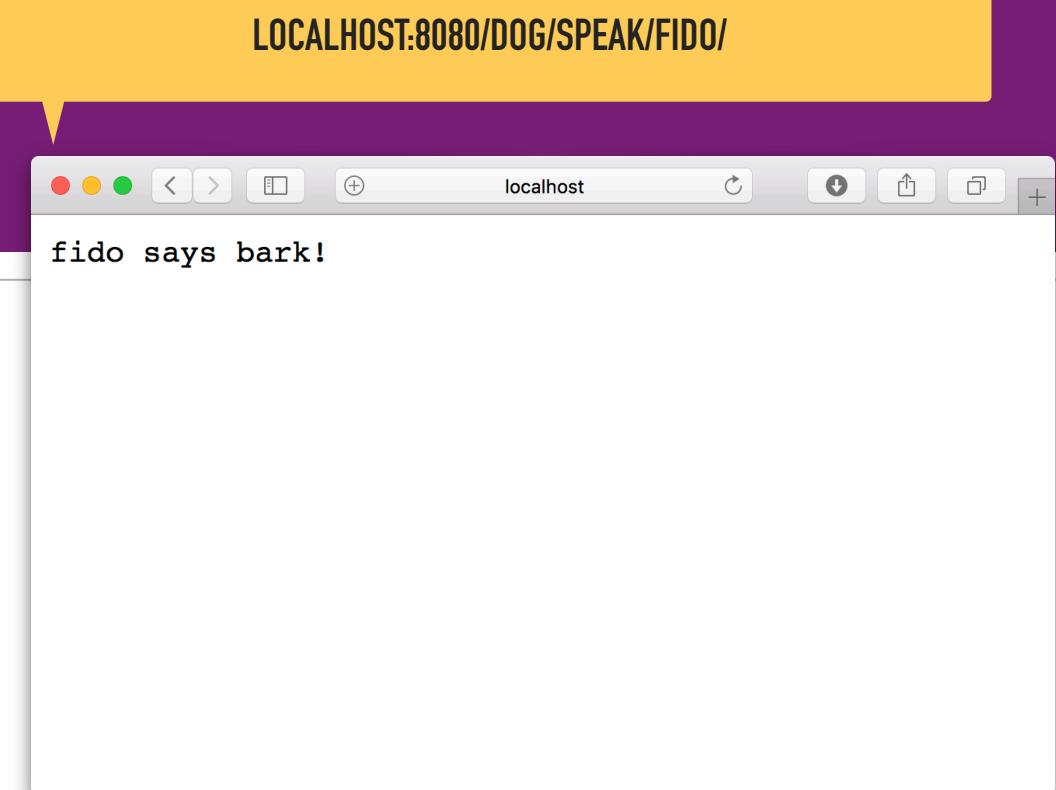
```
drop.get { req in
    return try JSON(node:
        ["message": "Hello Vapor"]
    )
}

drop.get("dog") { req in
    return "Woof"
}

drop.get("dog", "speak") { req in
    return "Bark! Bark!"
}

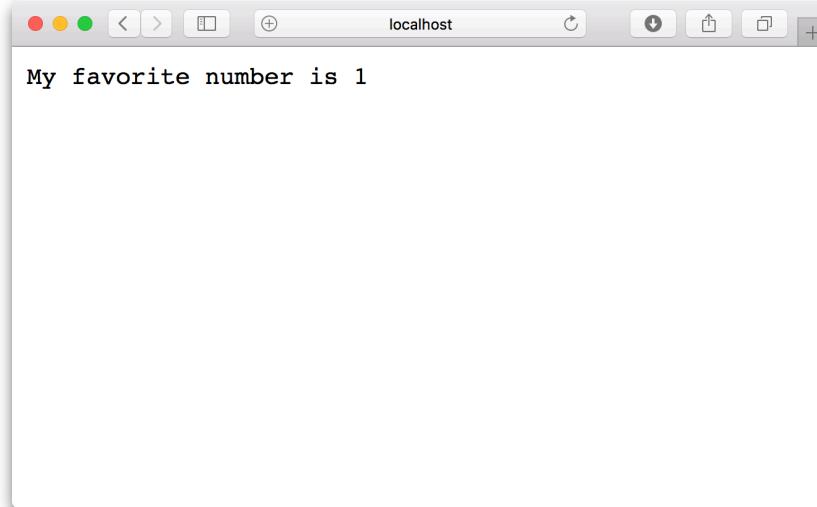
drop.get("dog", "speak", String.self) { req, name in
    return "\(name) says bark!"
}

drop.run()
```

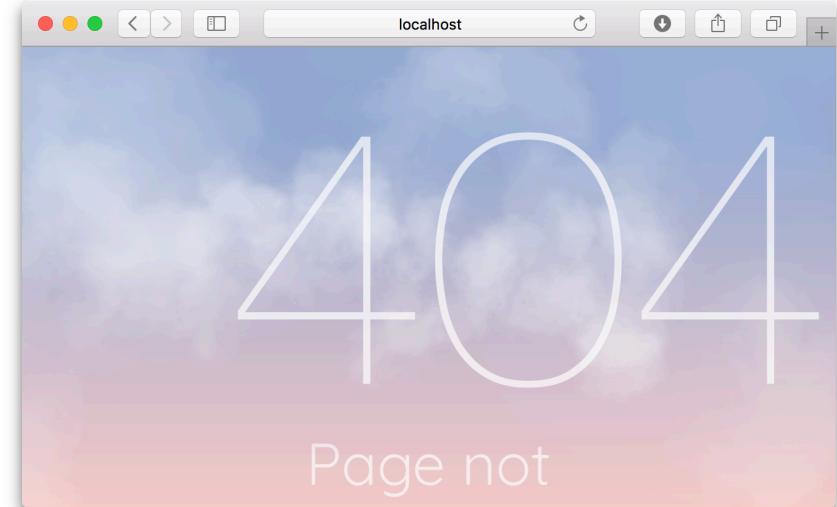


VAPOR

LOCALHOST:8080/DOG/SPEAK/1/



LOCALHOST:8080/DOG/SPEAK/FIDO/



```
drop.get("dog", "speak", Int.self) { req, number in
    return "My favorite number is \(number)"
}
```

DEPLOY VAPOR APP TO
HEROKU

DEPLOY VAPOR APP TO HEROKU

- Heroku is a platform as a service company
- Free 'hobby' tier to play around



Log in to your account

Email address

Password

Log In

New to Heroku? [Sign Up](#)

[Log in via SSO](#) [Forgot your password?](#)

DEPLOY VAPOR APP TO HEROKU

```
# Install tools  
brew install heroku
```

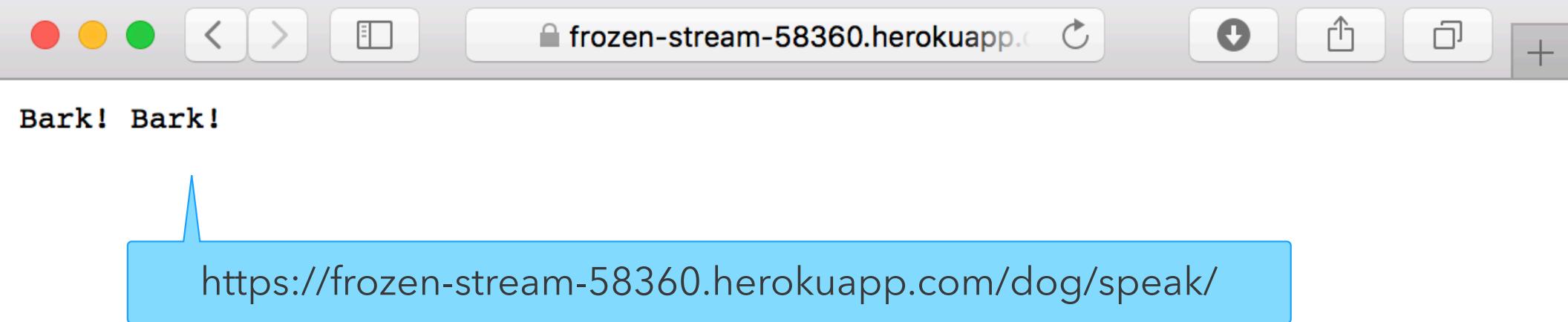
```
# Set credentials  
heroku login
```

```
# Upload  
vapor heroku init
```

DEPLOY VAPOR APP TO HEROKU

```
Building on Heroku ... ~5-10 minutes [D] •
Building on Heroku ... ~5-10 minutes [D]
Building on Heroku ... ~5-10 minutes [Do]
Building on Heroku ... ~5-10 minutes [Do]
Building on Heroku ... ~5-10 minutes [Do] •
Building on Heroku ... ~5-10 minutes [Done]
Spinning up dynos [Done]
Visit https://dashboard.heroku.com/apps/
App is live on Heroku, visit
https://frozen-stream-58360.herokuapp.com/ | https://git.heroku.com/frozen-stream-58360.git
```

DEPLOY VAPOR APP TO HEROKU



VAPOR WITH POSTGRESQL

VAPOR WITH POSTGRESQL

- Vapor support many different databases
 - MySQL
 - MongoDB
 - PostgreSQL
- Databases run independently of the web app

The world's most advanced open source database.

Home | About | Download | Documentation | Community | Developers | Support | Your account

11th May 2017

PostgreSQL 9.6.3, 9.5.7, 9.4.12, 9.3.17 and 9.2.21 Released!

The PostgreSQL Global Development Group is pleased to announce the availability of PostgreSQL 9.6.3, 9.5.7, 9.4.12, 9.3.17 and 9.2.21.

These new releases contain bug fixes over previous releases. All users should plan to upgrade their systems as soon as possible.

» [Release Announcement](#)
» [Release Notes](#)
» [Download](#)



FEATURED USER

... mission-critical technology tasks can continue to depend on the power, flexibility and robustness of PostgreSQL.

Ram Mohan, CTO, [Afiliias](#)
» [Case Studies](#) | [More Quotes](#) | [Featured Users](#)

LATEST NEWS

2017-05-11
[2017-05-11 Security Update Release](#)

2017-05-07
[pg_chameleon 1.0 released](#)

2017-05-04
[Announcing The Release Of pglogical 2.0](#)

2017-04-25
[DB Doc 4.1 released](#)

2017-04-19
[New version of MySQL-to-PostgreSQL has been released](#)

2017-04-07
[PgComment for PostgreSQL released](#)

2017-03-20
[Announcing AgensGraph v1.1 Release](#)

» [More](#) | [Submit News](#) | [RSS](#)

UPCOMING EVENTS

2017-05-18 – 2017-05-20
[PostgreSQL Booth at PyCon 2017](#)
(Portland, Oregon, United States)

2017-05-23 – 2017-05-26
[PGCon 2017](#)
(Ottawa, Ontario, Canada)

2017-06-08
[Pg Day France 2017](#)
(Toulouse, France)

2017-06-09
[PgDay Argentina 2017](#)
(Santa Fe, Santa Fe, Argentina)

2017-06-26 – 2017-06-28
[Postgres Vision 2017](#)
(Boston, MA, United States)

» [More](#) | [Submit Event](#) | [RSS](#)

UPCOMING TRAINING

There are 11 training events in 3 countries scheduled over the next six months from PostgresCourse.com,

LATEST RELEASES

9.6.3 · May 11, 2017 · [Notes](#)
9.5.7 · May 11, 2017 · [Notes](#)
9.4.12 · May 11, 2017 · [Notes](#)
9.3.17 · May 11, 2017 · [Notes](#)
9.2.21 · May 11, 2017 · [Notes](#)

[Download](#) | [RSS](#)
Why should I upgrade?
Upcoming releases

SHORTCUTS

» [Security](#)
» [International Sites](#)
» [Mailing Lists](#)
» [Wiki](#)
» [Report a Bug](#)
» [FAQs](#)

SUPPORT US

PostgreSQL is free. Please support our work by making a [donation](#).

PLANET POSTGRES

2017-05-16
[Joshua Drake: PGConf US Local: Seattle Call for Papers](#)

2017-05-16
[Michael Paquier: Postgres 10 highlight - Incompatible changes](#)

2017-05-15
[Bruce Momjian: RAID Controllers and SSD's](#)

2017-05-15
[Holly Orr: DevOps: Can't we just all get along?](#)

2017-05-13
[Federico Campoli: pg_chameleon v1.1 out](#)

2017-05-13
[Leo Hsu and Regina Obe: PostgreSQL JQuery extension Windows binaries](#)

2017-05-12
[Bruce Momjian: Postgres Window Magic](#)

» [More](#) | [RSS](#)

VAPOR WITH POSTGRESQL

- Importing Vapor also imports the `Fluent` framework
- ORM tool for Swift
 - Works with different databases
- Can be used on app for consistency

vapor / fluent

Code Issues 0 Pull requests 0 Wiki Pulse Graphs

Swift models, relationships, and querying for NoSQL and SQL databases.

vapor orm swift database sql nosql

608 commits 3 branches 97 releases 31 contributors

Create new file

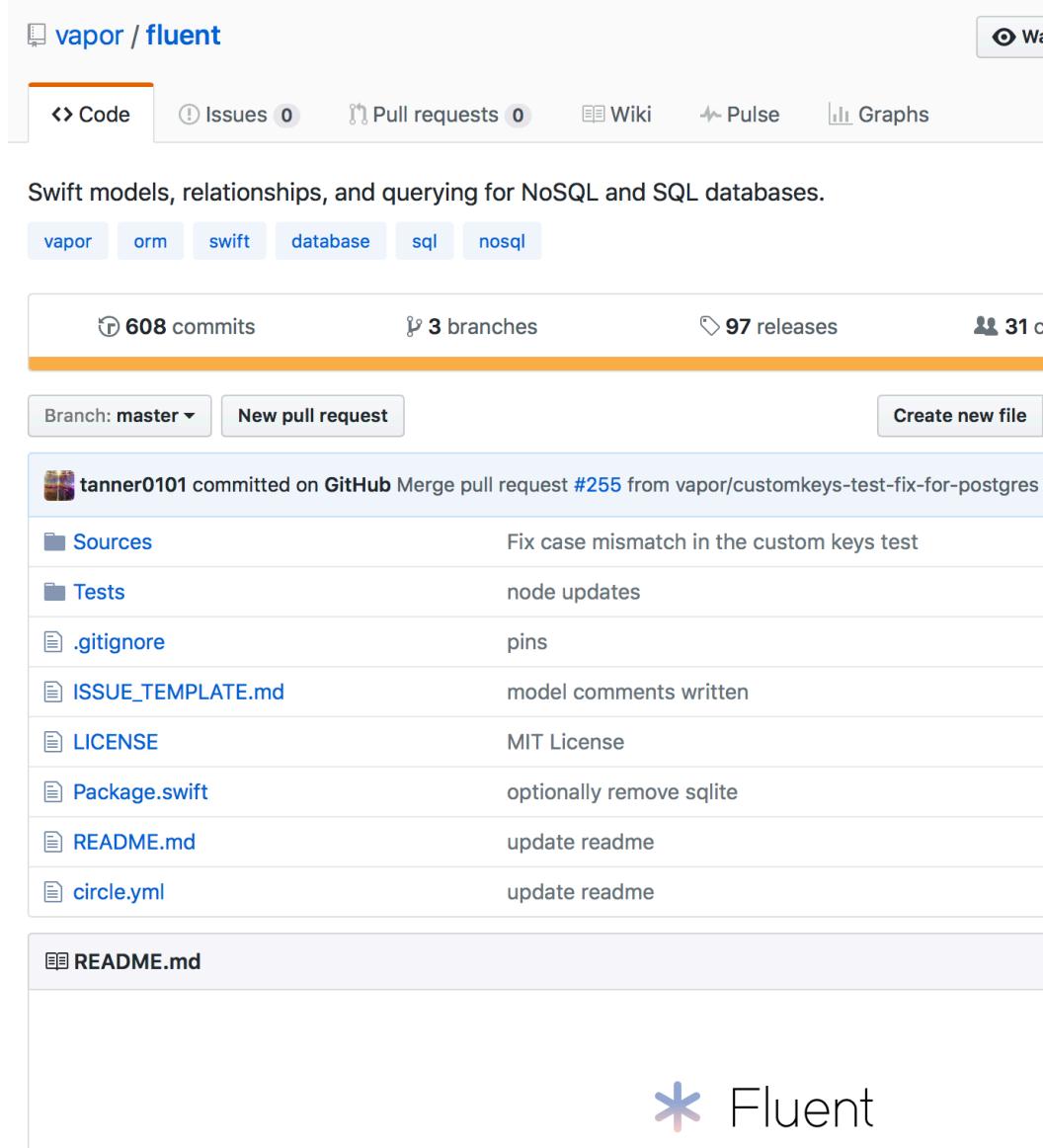
Branch: master New pull request

tanner0101 committed on GitHub Merge pull request #255 from vapor/customkeys-test-fix-for-postgres

Sources	Fix case mismatch in the custom keys test
Tests	node updates
.gitignore	pins
ISSUE_TEMPLATE.md	model comments written
LICENSE	MIT License
Package.swift	optionally remove sqlite
README.md	update readme
circle.yml	update readme

README.md

* Fluent



VAPOR WITH POSTGRESQL

- Import package
- Configure droplet
- Add configuration file

The world's most advanced open source database.

Home | About | Download | Documentation | Community | Developers | Support | Your account

11th May 2017

PostgreSQL 9.6.3, 9.5.7, 9.4.12, 9.3.17 and 9.2.21 Released!

The PostgreSQL Global Development Group is pleased to announce the availability of PostgreSQL 9.6.3, 9.5.7, 9.4.12, 9.3.17 and 9.2.21.

These new releases contain bug fixes over previous releases. All users should plan to upgrade their systems as soon as possible.

» [Release Announcement](#)
» [Release Notes](#)
» [Download](#)



FEATURED USER

... mission-critical technology tasks can continue to depend on the power, flexibility and robustness of PostgreSQL.

Ram Mohan, CTO, Afilias
» [Case Studies](#) | [More Quotes](#) | [Featured Users](#)

LATEST NEWS

2017-05-11 [2017-05-11 Security Update Release](#)
2017-05-07 [pg_chameleon 1.0 released](#)
2017-05-04 [Announcing The Release Of pglogical 2.0](#)
2017-04-25 [DB Doc 4.1 released](#)
2017-04-19 [New version of MySQL-to-PostgreSQL has been released](#)
2017-04-07 [PgComment for PostgreSQL released](#)
2017-03-20 [Announcing AgensGraph v1.1 Release](#)
» [More](#) | [Submit News](#) | [RSS](#)

UPCOMING EVENTS

2017-05-18 – 2017-05-20 [PostgreSQL Booth at PyCon 2017](#)
(Portland, Oregon, United States)
2017-05-23 – 2017-05-26 [PGCon 2017](#)
(Ottawa, Ontario, Canada)
2017-06-08 [PG Day France 2017](#)
(Toulouse, France)
2017-06-09 [PgDay Argentina 2017](#)
(Santa Fe, Santa Fe, Argentina)
2017-06-26 – 2017-06-28 [Postgres Vision 2017](#)
(Boston, MA, United States)
» [More](#) | [Submit Event](#) | [RSS](#)

UPCOMING TRAINING

There are 11 training events in 3 countries scheduled over the next six months from PostgresCourse.com,

LATEST RELEASES

9.6.3 · May 11, 2017 · [Notes](#)
9.5.7 · May 11, 2017 · [Notes](#)
9.4.12 · May 11, 2017 · [Notes](#)
9.3.17 · May 11, 2017 · [Notes](#)
9.2.21 · May 11, 2017 · [Notes](#)

[Download](#) | [RSS](#)
[Why should I upgrade?](#)
[Upcoming releases](#)

SHORTCUTS

» [Security](#)
» [International Sites](#)
» [Mailing Lists](#)
» [Wiki](#)
» [Report a Bug](#)
» [FAQs](#)

SUPPORT US

PostgreSQL is free. Please support our work by making a [donation](#).

PLANET POSTGRES

2017-05-16 [Joshua Drake: PGConf US Local: Seattle Call for Papers](#)
2017-05-16 [Michael Paquier: Postgres 10 highlight - Incompatible changes](#)
2017-05-15 [Bruce Momjian: RAID Controllers and SSDs](#)
2017-05-15 [Holly Orr: DevOps: Can't we just all get along?](#)
2017-05-13 [Federico Campoli: pg_chameleon v1.1 out](#)
2017-05-13 [Leo Hsu and Regina Obe: PostgreSQL JQuery extension Windows binaries](#)
2017-05-12 [Bruce Momjian: Postgres Window Magic](#)
» [More](#) | [RSS](#)

VAPOR WITH POSTGRESQL

```
# Install postgres
brew install postgres

# Start postgres server
postgres -D /usr/local/var/postgres/

# Create a database
createdb dogs
```

VAPOR WITH POSTGRESQL

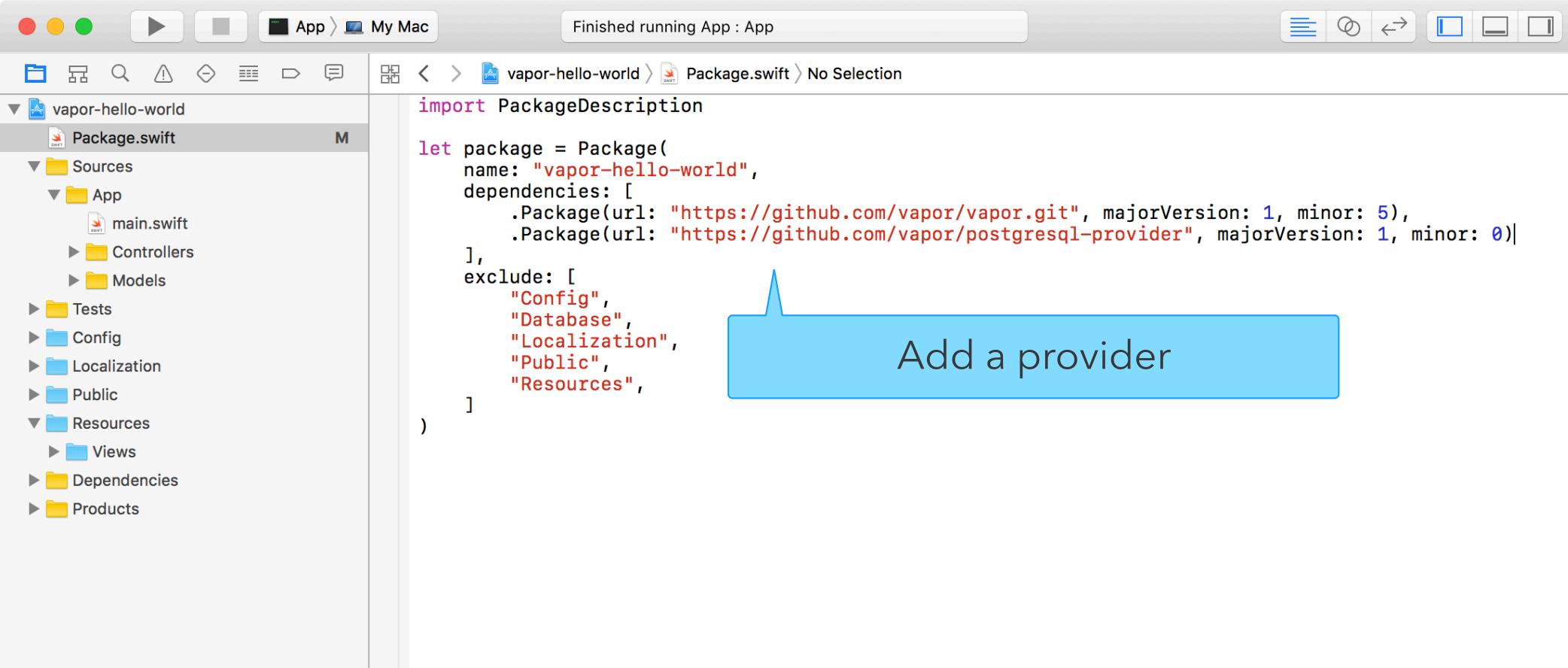
```
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
[592 % psql
psql (9.6.3)
Type "help" for help.

[tabinkowski=# \l
                                         List of databases
  Name | Owner | Encoding | Collate | Ctype | Access privileges
-----+-----+-----+-----+-----+-----+
dogs   | tabinkowski | UTF8    | en_US.UTF-8 | en_US.UTF-8 |
postgres | tabinkowski | UTF8    | en_US.UTF-8 | en_US.UTF-8 |
tabinkowski | tabinkowski | UTF8    | en_US.UTF-8 | en_US.UTF-8 |
template0 | tabinkowski | UTF8    | en_US.UTF-8 | en_US.UTF-8 | =c/tabinkowski          +
                                         |                   tabinkowski=CTc/tabinkowski
template1 | tabinkowski | UTF8    | en_US.UTF-8 | en_US.UTF-8 | =c/tabinkowski          +
                                         |                   tabinkowski=CTc/tabinkowski
(5 rows)

tabinkowski=# ]
```

- Vapor support many different databases
 - MySQL

VAPOR WITH POSTGRESQL



Finished running App : App

vapor-hello-world

Package.swift

Sources

App

main.swift

Controllers

Models

Tests

Config

Localization

Public

Resources

Views

Dependencies

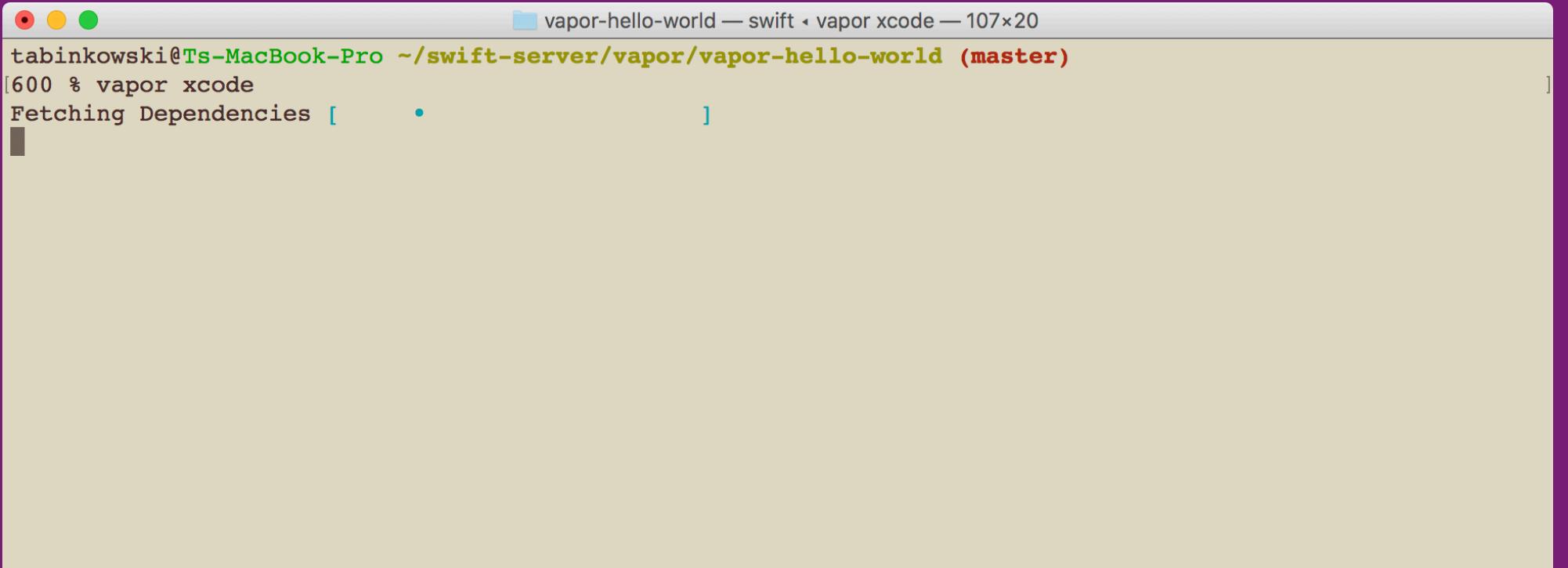
Products

```
import PackageDescription

let package = Package(
    name: "vapor-hello-world",
    dependencies: [
        .Package(url: "https://github.com/vapor/vapor.git", majorVersion: 1, minor: 5),
        .Package(url: "https://github.com/vapor/postgresql-provider", majorVersion: 1, minor: 0)
    ],
    exclude: [
        "Config",
        "Database",
        "Localization",
        "Public",
        "Resources",
    ]
)
```

Add a provider

VAPOR WITH POSTGRESQL



A screenshot of a macOS terminal window titled "vapor-hello-world — swift ▾ vapor xcode — 107x20". The window shows the command "tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)" followed by "[600 % vapor xcode". Below this, the text "Fetching Dependencies [•]" is displayed, indicating a progress bar.

- Update the project and rebuild your framework in Xcode

VAPOR WITH POSTGRESQL

< > vapor-hello-world > Sources > App > main.swift > No Selection

```
import Vapor
import VaporPostgreSQL

let drop = Droplet()

// Add provider
do {
    try drop.addProvider(VaporPostgreSQL.Provider.self)
} catch {
    assertionFailure("Error adding provider: \(error)")
}
```

Add a provider

VAPOR WITH POSTGRESQL

vapor-hello-world | Build **Succeeded**

vapor-hello-world > Config > secrets > postgresql.json > No Selection

```
{  
    "host": "127.0.0.1",  
    "user": "tabinkowski",  
    "password": "",  
    "database": "dogs",  
    "port": 5432  
}
```

Database configuration file

Add a new file

vapor-hello-world

- Package.swift
- Sources
 - App
 - main.swift
 - Controllers
 - Models
- Tests
- Config
 - app.json
 - clients.json
 - crypto.json
 - droplet.json
 - production
 - secrets
 - postgresql.json
 - servers.json
- Localization
- Public
- Resources
- Dependencies

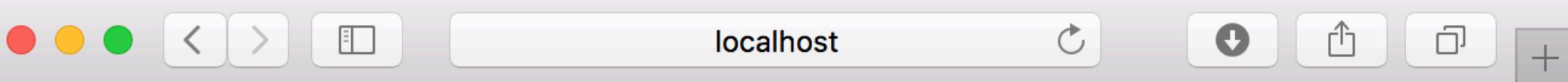
VAPOR WITH POSTGRESQL

```
// Add provider
do {
    try drop.addProvider(VaporPostgreSQL.Provider.self)
} catch {
    assertionFailure("Error adding provider: \(error)")
}

// Route to print out the db version
drop.get("version") { req in
    if let db = drop.database?.driver as? PostgreSQLDriver {
        let version = try db.raw("SELECT version()")
        return try JSON(node: version)
    } else {
        return "No database connection"
    }
}
```

Print out database version

VAPOR WITH POSTGRESQL



IT'S BORING, BUT IT WORKS

VAPOR WITH POSTGRESQL

The screenshot shows the Xcode interface with a Vapor application running. The title bar says "Running App : App". The file path in the sidebar is "vapor-hello-world > Sources > App > main.swift > No Selection". The code in main.swift is:

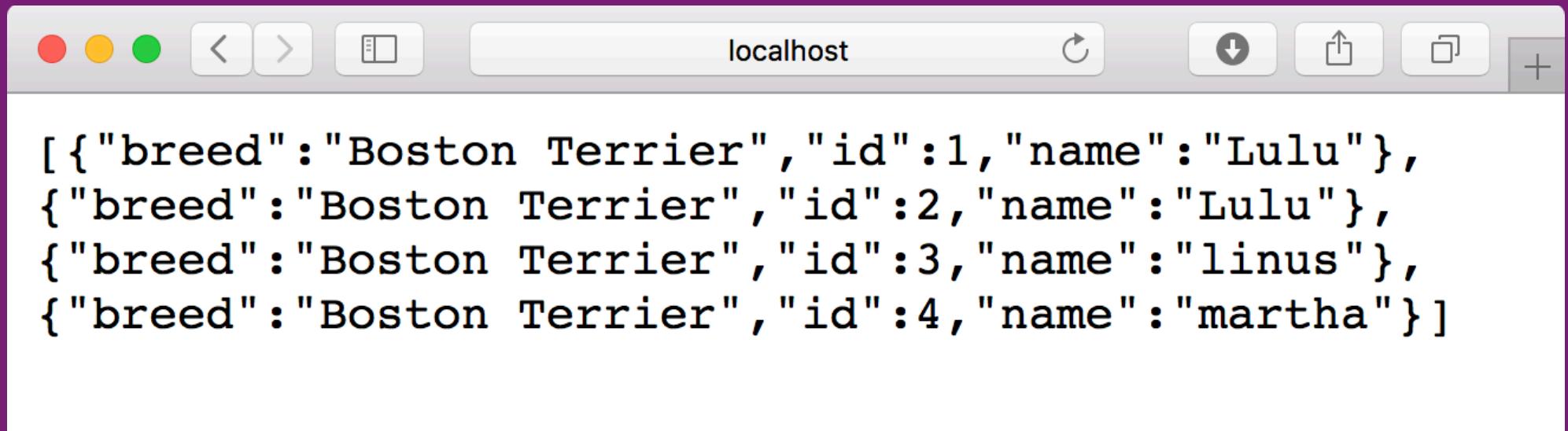
```
let drop = Droplet()
drop.preparations.append(Dog.self)

// Add provider
do {
    try drop.addProvider(VaporPostgreSQL.Provider.self)
} catch {
    assertionFailure("Error adding provider: \(error)")
}

drop.get("dog", "create") { req in
    let dog = [Dog(name:"Lulu", breed: "Boston Terrier"),
              Dog(name:"Snoopy", breed: "Beagle"),
              Dog(name:"Fifi", breed: "Poodle")]
    let dogNode = try dog.makeNode()
    let nodeDictionary = ["dogs": dogNode]
    return try JSON(node: nodeDictionary)
}
```

A yellow callout box points to the line `drop.preparations.append(Dog.self)` with the text "ADD DOG MODEL". Another yellow callout box points to the line `let dog = [Dog(name:"Lulu", breed: "Boston Terrier"), Dog(name:"Snoopy", breed: "Beagle"), Dog(name:"Fifi", breed: "Poodle")]` with the text "CREATE DATA".

VAPOR WITH POSTGRESQL



```
// List all dogs
drop.get("dog", "pound") { req in
    return try JSON(node: Dog.all().makeNode())
}
```

VAPOR WITH POSTGRESQL

- Check that data is being persisted using psql
 - \l - list dis
 - \c - switch db
 - \d - list tables

```
vapor-hello-world — psql — 60x20
(5 rows)

tabinkowski=# \dt
No relations found.
tabinkowski=# \c dogs
You are now connected to database "dogs" as user "ta
i".
dogs=# select * from dogs;
 id | name      |      breed
----+-----+-----
    1 | Lulu      | Boston Terrier
    2 | Lulu      | Boston Terrier
    3 | linus     | Boston Terrier
    4 | martha    | Boston Terrier
    5 | martha    | Boston Terrier
    6 | martha    | Boston Terrier
    7 | martha    | Boston Terrier
(7 rows)

dogs=#
```

DEPLOY TO HEROKU WITH POSTGRESQL

DEPLOY TO HEROKU WITH POSTGRESQL

- Set up postgresql on heroku
 - Built-in, just need to add it
- Tell Postgresql where the database is located
 - Do not upload the secrets/postgresql.json file

```
{  
  "host": "127.0.0.1",  
  "user": "tabinkowski",  
  "password": "",  
  "database": "dogs",  
  "port": 5432  
}
```

DEPLOY TO HEROKU WITH POSTGRESQL

Add postgres to your heorku app

```
vapor-hello-world — -bash — 89x19
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
613 % heroku addons:create heroku-postgresql:hobby-dev
Creating heroku-postgresql:hobby-dev on ⚙ frozen-stream-58360... free
Database has been created and is available
! This database is empty. If upgrading, you can transfer
! data from another database with pg:copy
Created postgresql-vertical-49786 as DATABASE_URL
Use heroku addons:docs heroku-postgresql to view documentation
tabinkowski@Ts-MacBook-Pro /swift-server/vapor/vapor-hello-world (master)
614 %
```

New database

DEPLOY TO HEROKU WITH POSTGRESQL

```
vapor-hello-world bash 89x15
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
[613 % heroku addons:create heroku-postgresql:hobby-dev
Creating heroku-postgresql:hobby-dev on ⚙️ frozen-stream-58360... free
Database has been created and is available
! This database is empty. If upgrading, you can transfer
! data from another database with pg:copy
Created postgresql-vertical-49786 as DATABASE_URL
Use heroku addons:docs heroku-postgresql to view documentation
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
[614 % heroku config
==== frozen-stream-58360 Config Vars
DATABASE_URL: postgres://hzgtyzqowaztwa:4ad9847249b30c97f1835744ca06b1bae2b9012422c21d32d
4dad5e5088def04@ec2-23-23-227-188.compute-1.amazonaws.com:5432/d9hapt1pulfgrf
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
615 %
```

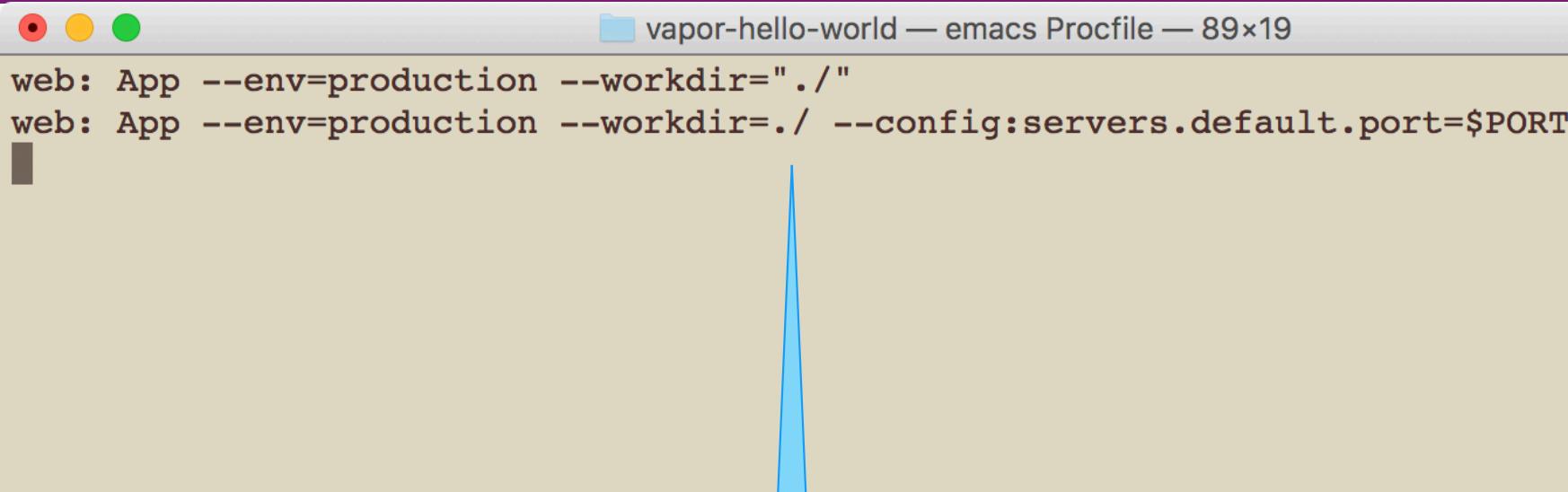
Database location

DEPLOY TO HEROKU WITH POSTGRESQL

```
vapor-hello-world — -bash — 89x19
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
[618 % ls
Config                               Procfile
Localization                         Public
Package.pins                          README.md
Package.swift                         Resources
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
619 % ]
```

Update the procfile with the database location

DEPLOY TO HEROKU WITH POSTGRESQL



```
vapor-hello-world — emacs Procfile — 89x19
web: App --env=production --workdir=". /"
web: App --env=production --workdir= ./ --config:servers.default.port=$PORT
```

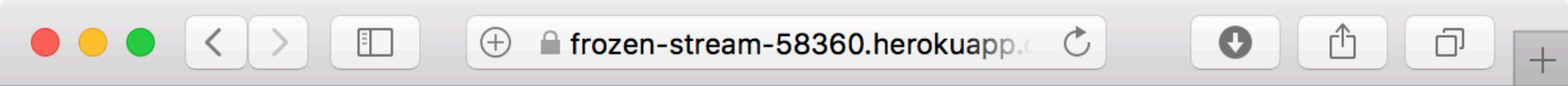
Update the procfile with the database location

DEPLOY TO HEROKU WITH POSTGRESQL

```
vapor-hello-world — -bash — 89x19
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
629 % git push heroku master
```

Push the changes to heroku

DEPLOY TO HEROKU WITH POSTGRESQL



```
[{"breed": "Boston Terrier", "id": 1, "name": "martha"}, {"breed": "Boston Terrier", "id": 2, "name": "martha"}]
```

<https://frozen-stream-58360.herokuapp.com/dog/pound/>

VAPOR WITH POSTGRESQL

```
// List all dogs
drop.get("dog", "pound") { req in
    return try JSON(node: Dog.all().makeNode())
}
```

Get all entries

VAPOR WITH POSTGRESQL

Filter

```
// Get all bostons
drop.get("query-boston") { req in
    return try JSON(node: Dog.query().filter("breed", .equals, "Boston Terrier").all().makeNode())
}
```

Query

VAPOR WITH POSTGRESQL

```
//  
drop.get("update-poodle") { req in  
    guard var dog = try Dog.query().first() else {  
        throw Abort.badRequest  
    }  
    dog.breed = "Poodle"  
    try dog.save()  
    return dog  
}
```

Get the first entry

Change values

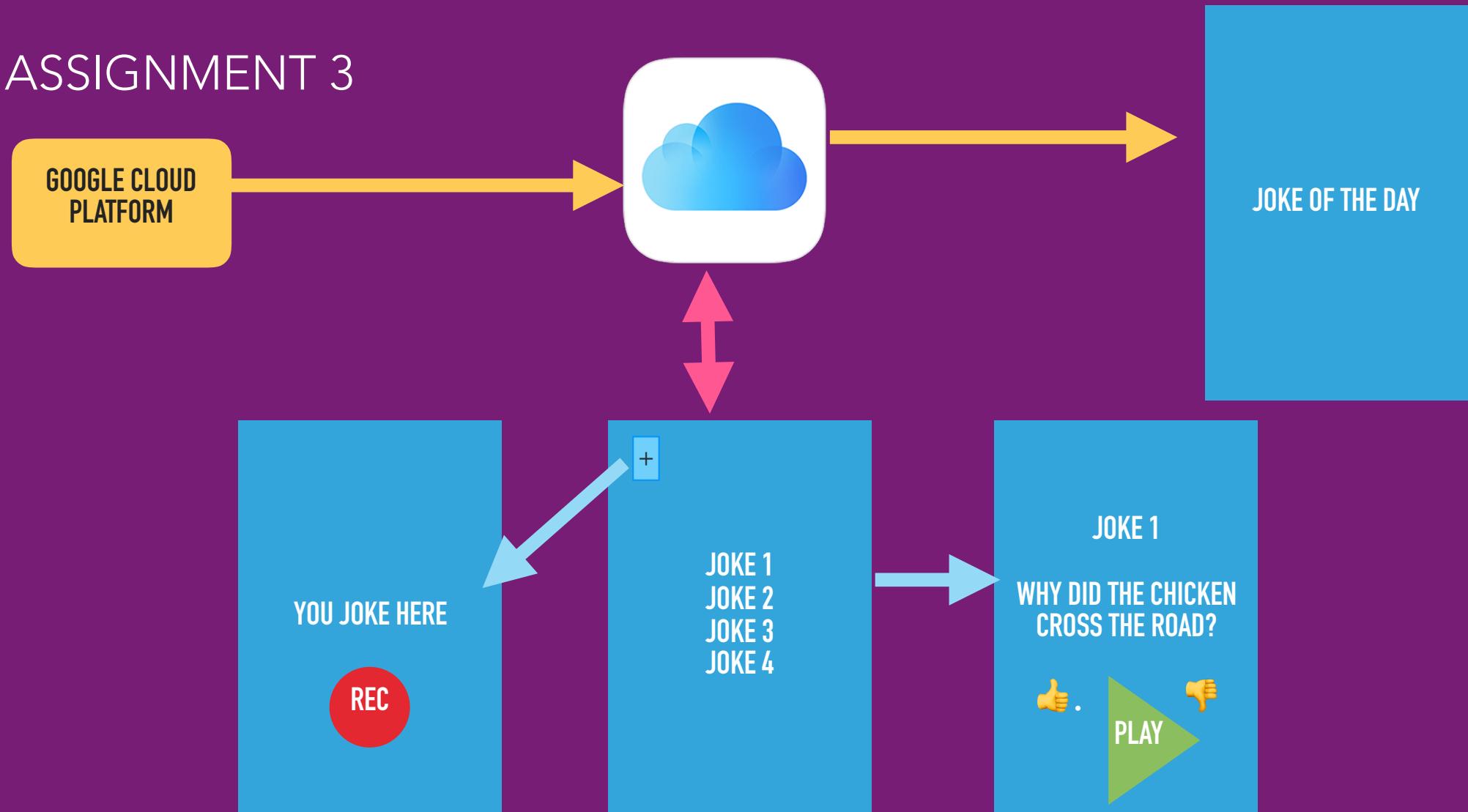
Save

ASSIGNMENT 4

ASSIGNMENT 5

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

ASSIGNMENT 3





THE UNIVERSITY OF
CHICAGO



MPCS 51033 • AUTUMN 2017 • SESSION 8

BACKENDS FOR MOBILE APPLICATIONS