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

JS

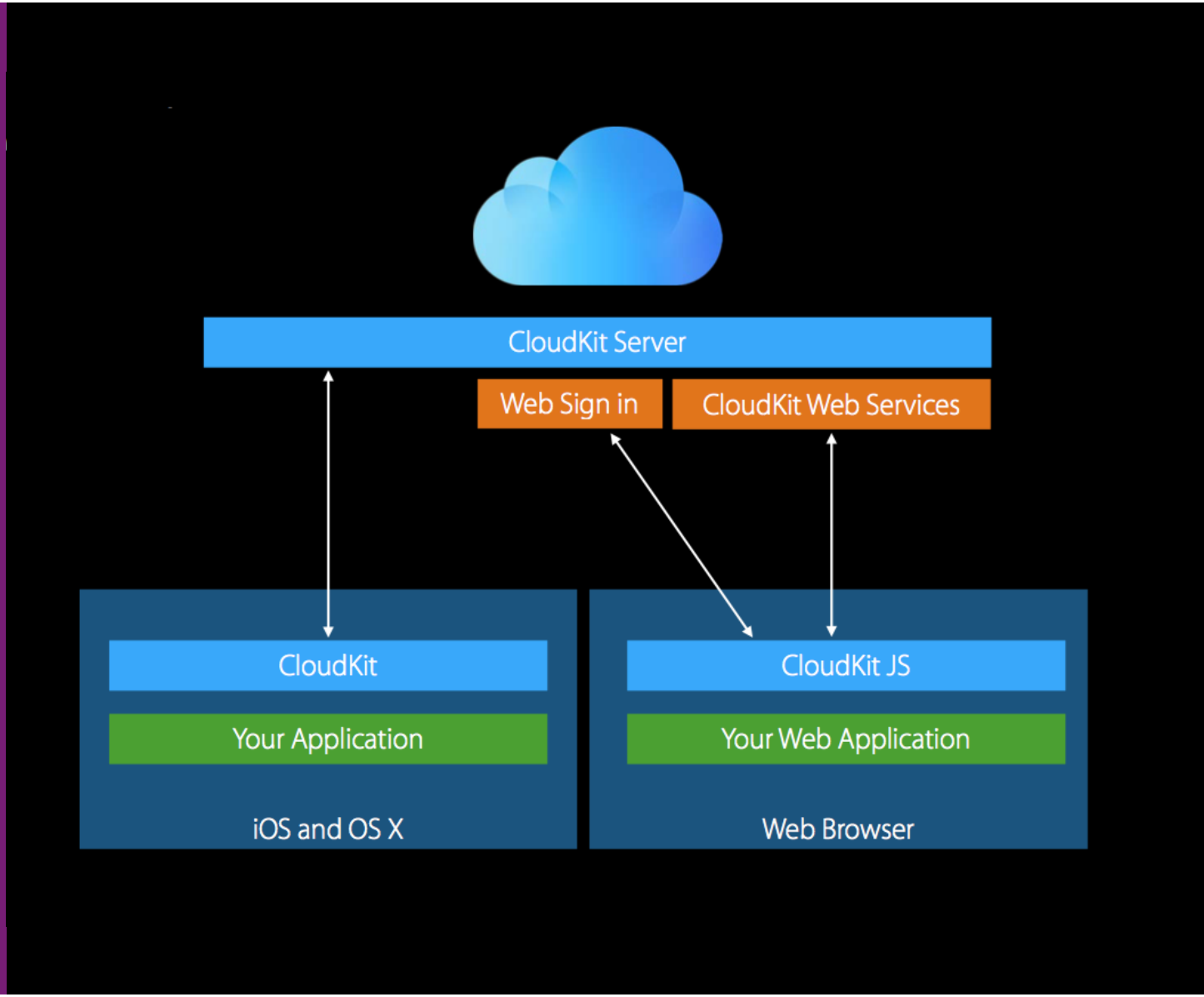
```
<SCRIPT SRC="HTTPS://CDN.APPLE-CLOUDKIT.COM/CK/1/CLOUDKIT.JS" />
```

WEB SERVICE API

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

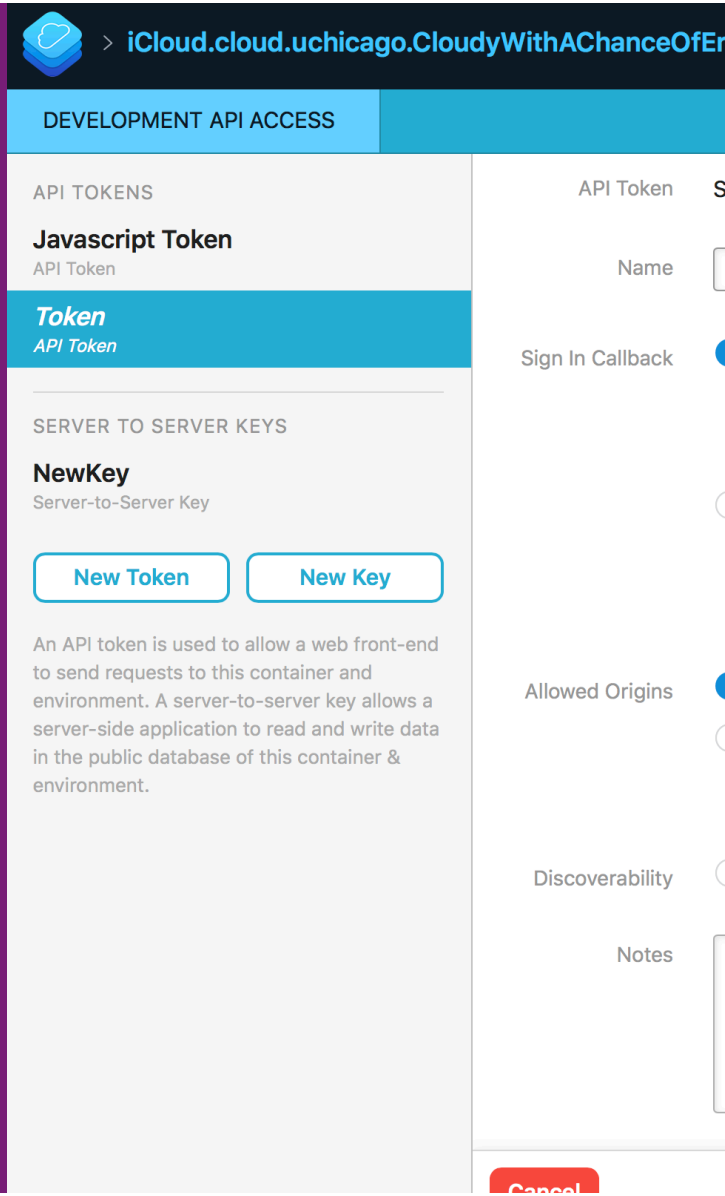
WEB SERVICES

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



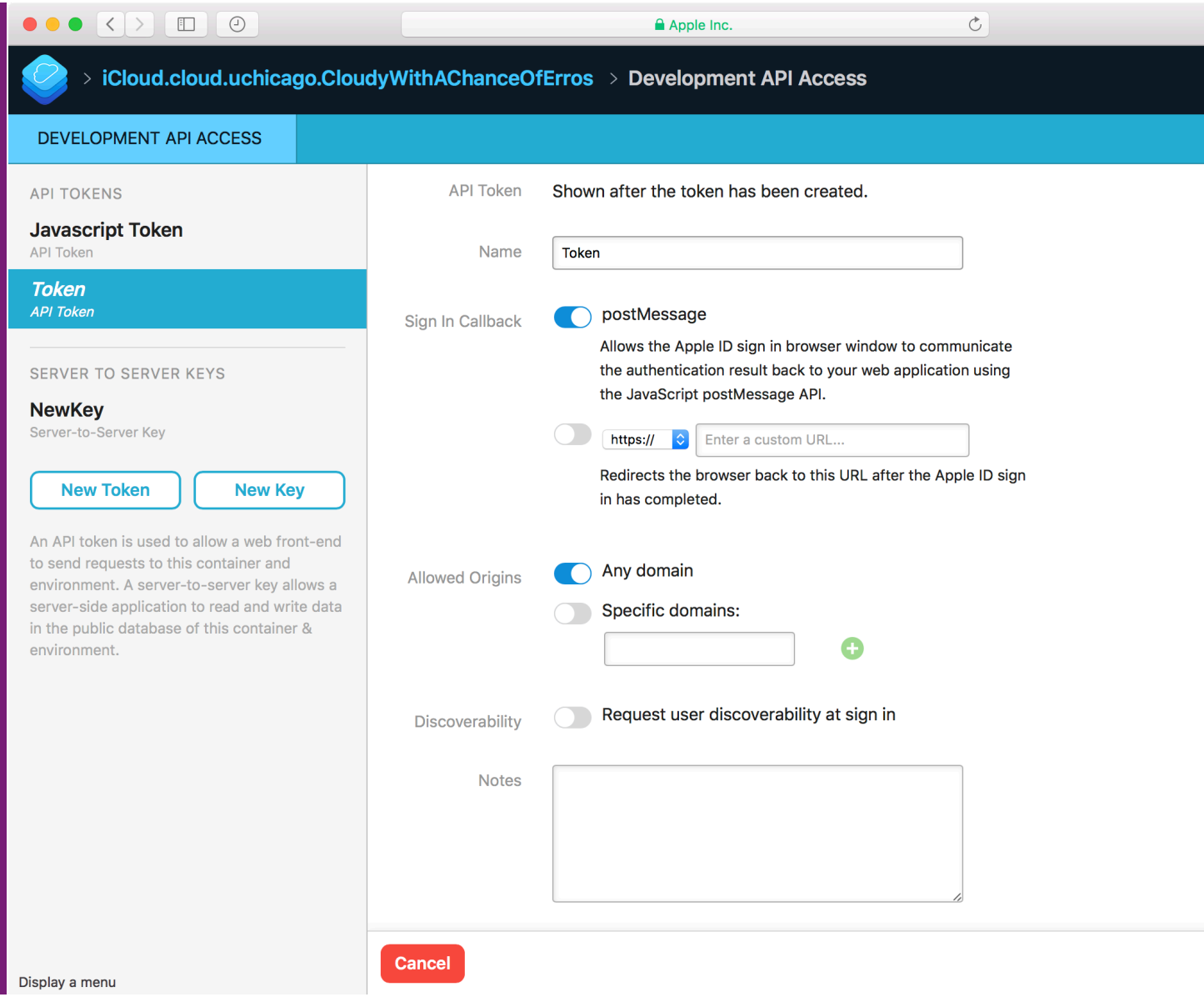
CLOUDKIT JS

- 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

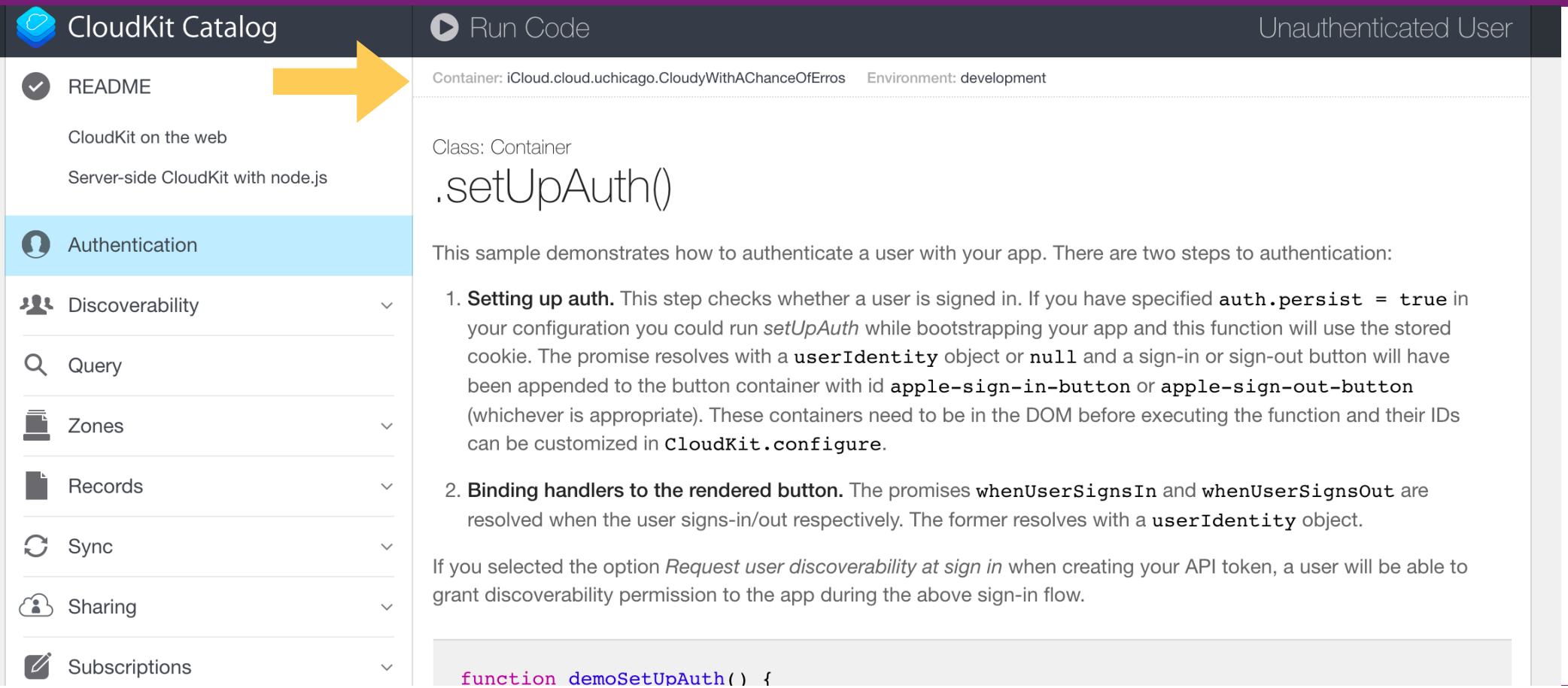


CLOUDKIT JS

- Create a new token

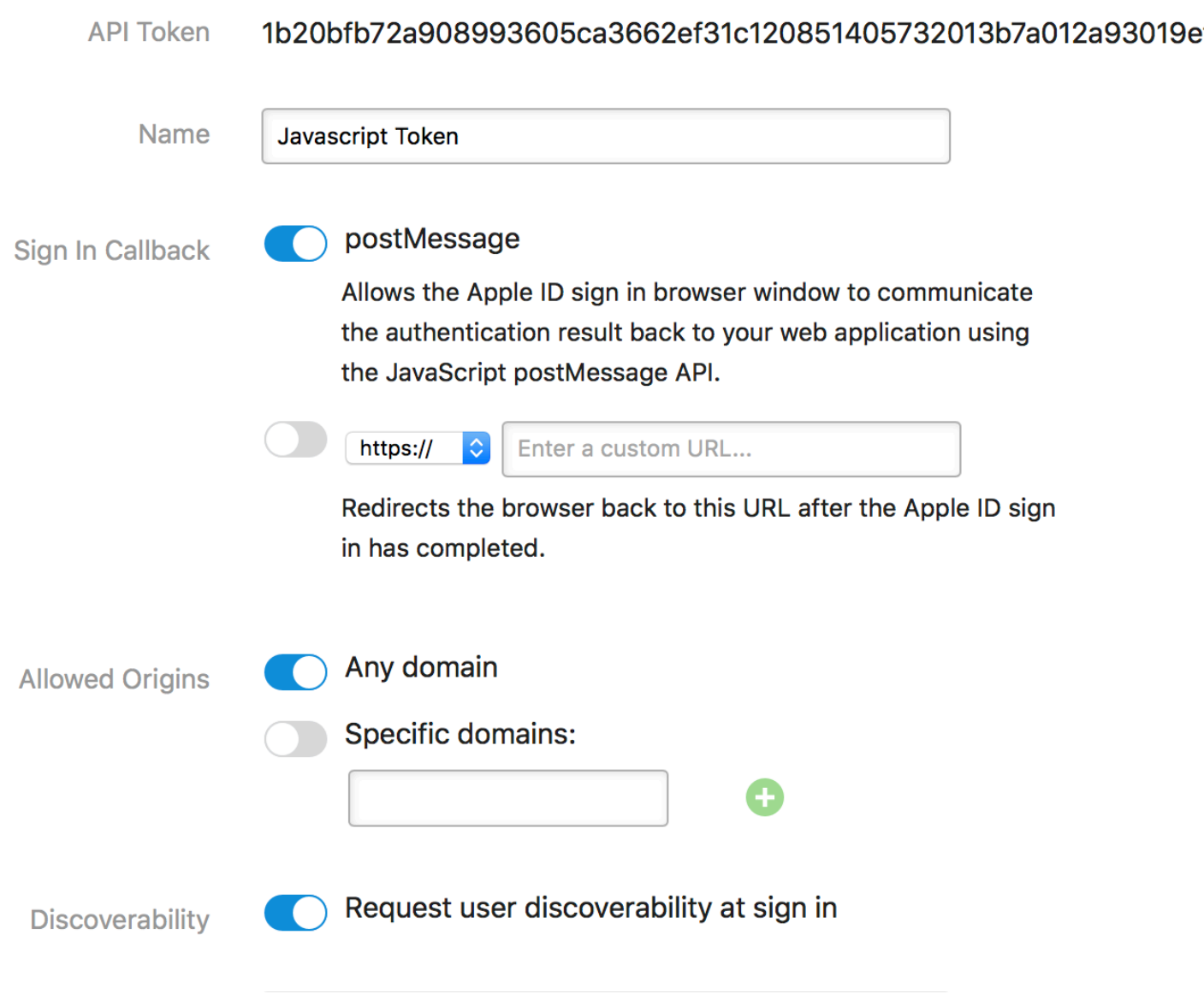


CLOUDKIT JS


CloudKit Catalog
Run Code
Unauthenticated User
README
CloudKit on the web
Server-side CloudKit with node.js
Authentication
Discoverability
Query
Zones
Records
Sync
Sharing
Subscriptions
Container: iCloud.cloud.uchicago.CloudyWithAChanceOfErros
Environment: development
Class: Container
.setUpAuth()
This sample demonstrates how to authenticate a user with your app. There are two steps to authentication:
1. 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.
2. Binding handlers to the rendered button. The promises whenUserSignsIn 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() {

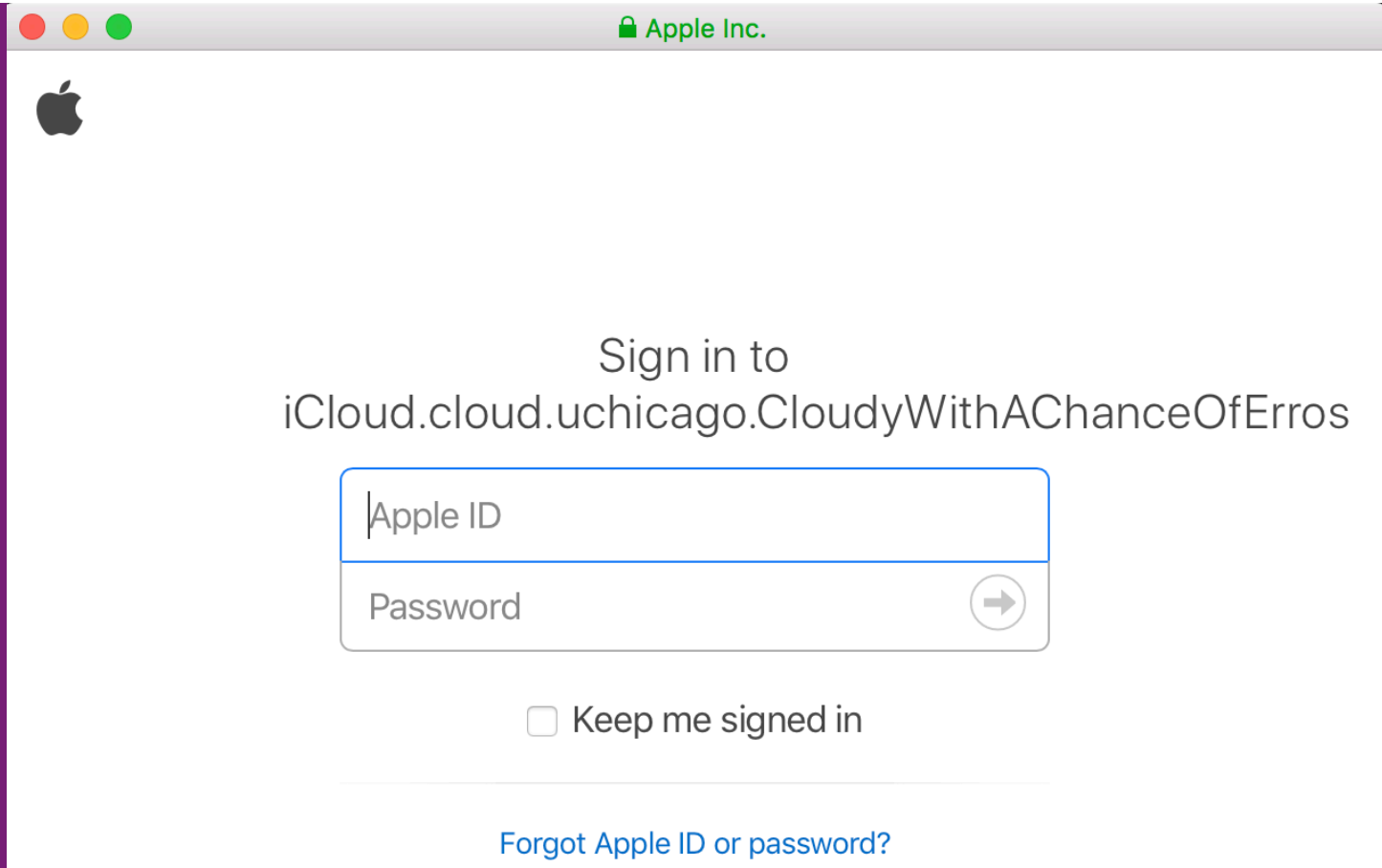
CLOUDKIT JS

- Token is used to communicate with iCloud



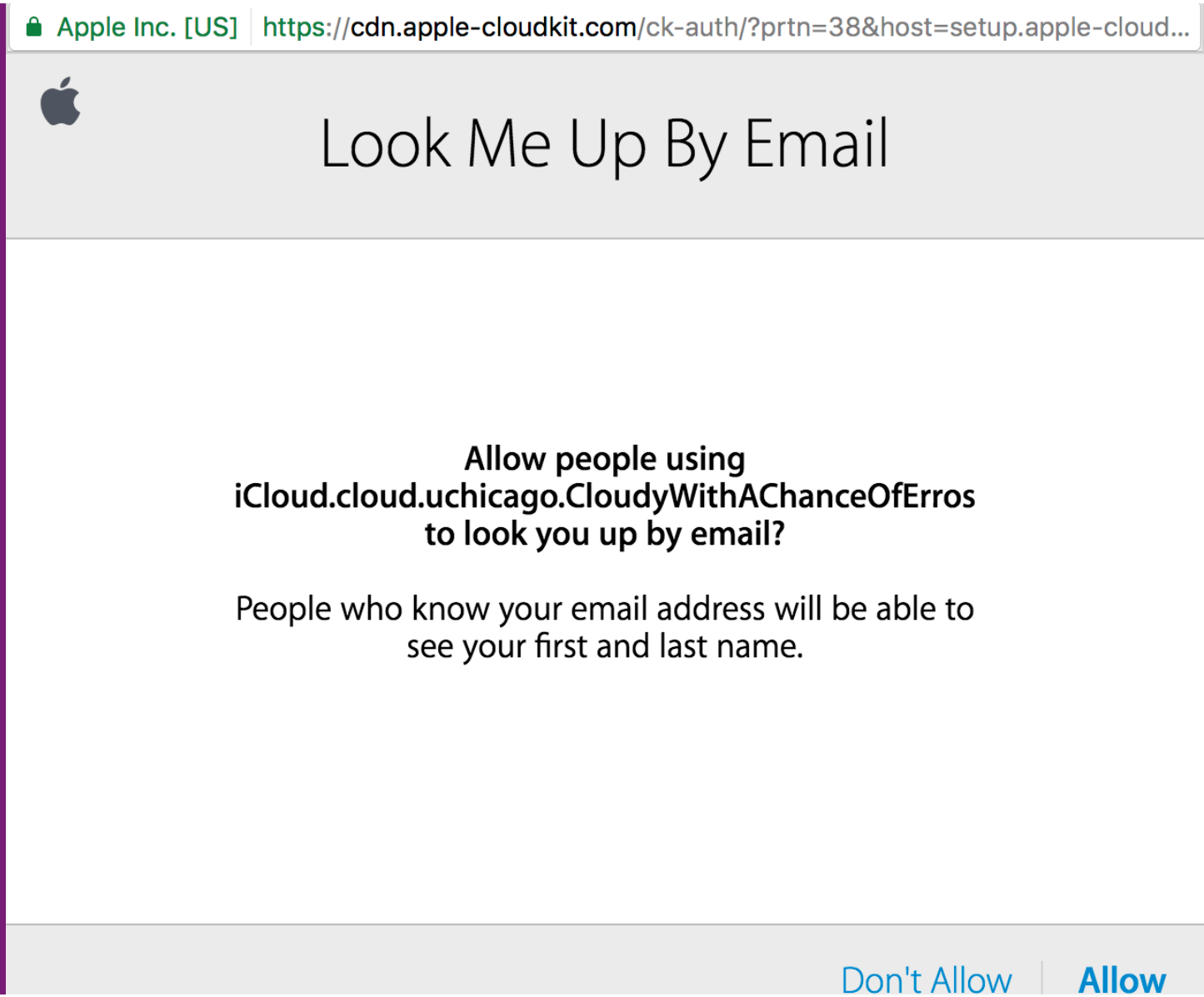
CLOUDKIT JS

- Authentication happens with Apple provided UI



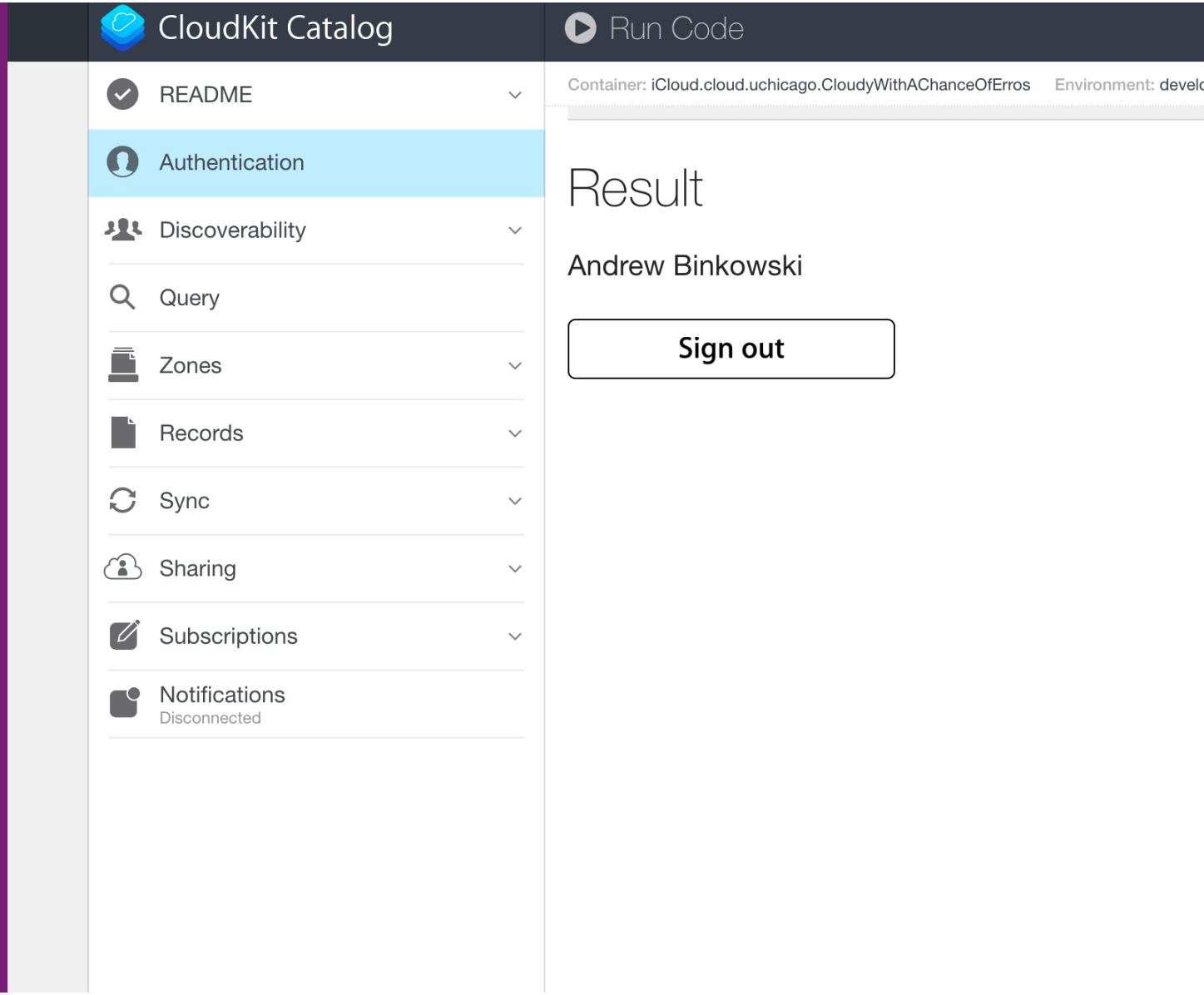
CLOUDKIT JS

- Test directly in CloudKitCatalog web app



CLOUDKIT JS

- Test directly in CloudKitCatalog web app



SERVER-TO-SERVER WITH CLOUDIKIT

SERVER-TO-SERVER

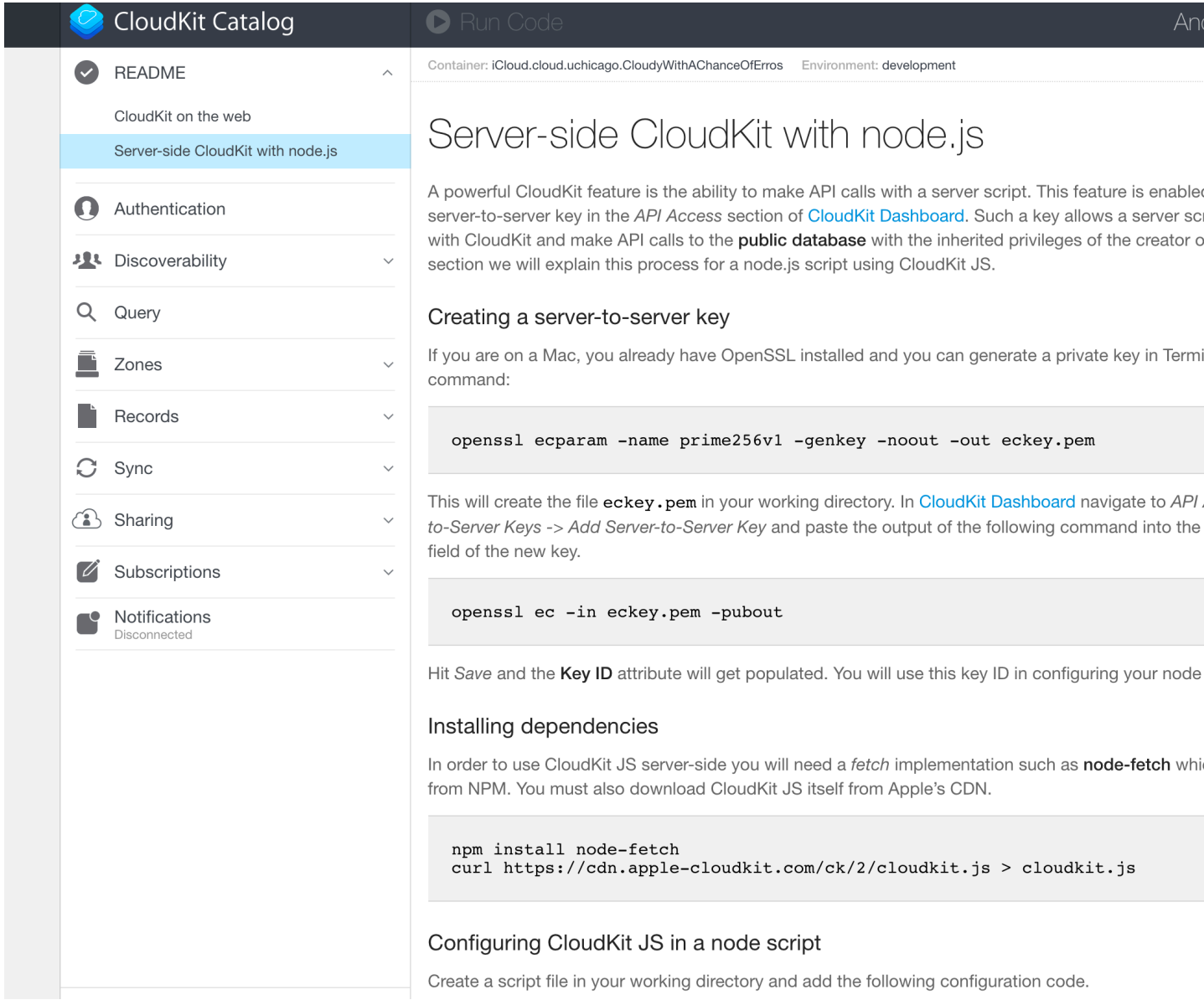
- Late 2016 Apple opened server-to-server communication
- Allows a server to communicate with CloudKit using a server-to-server key
- The missing link in data processing and server side logic 🤔

SERVER-TO-SERVER

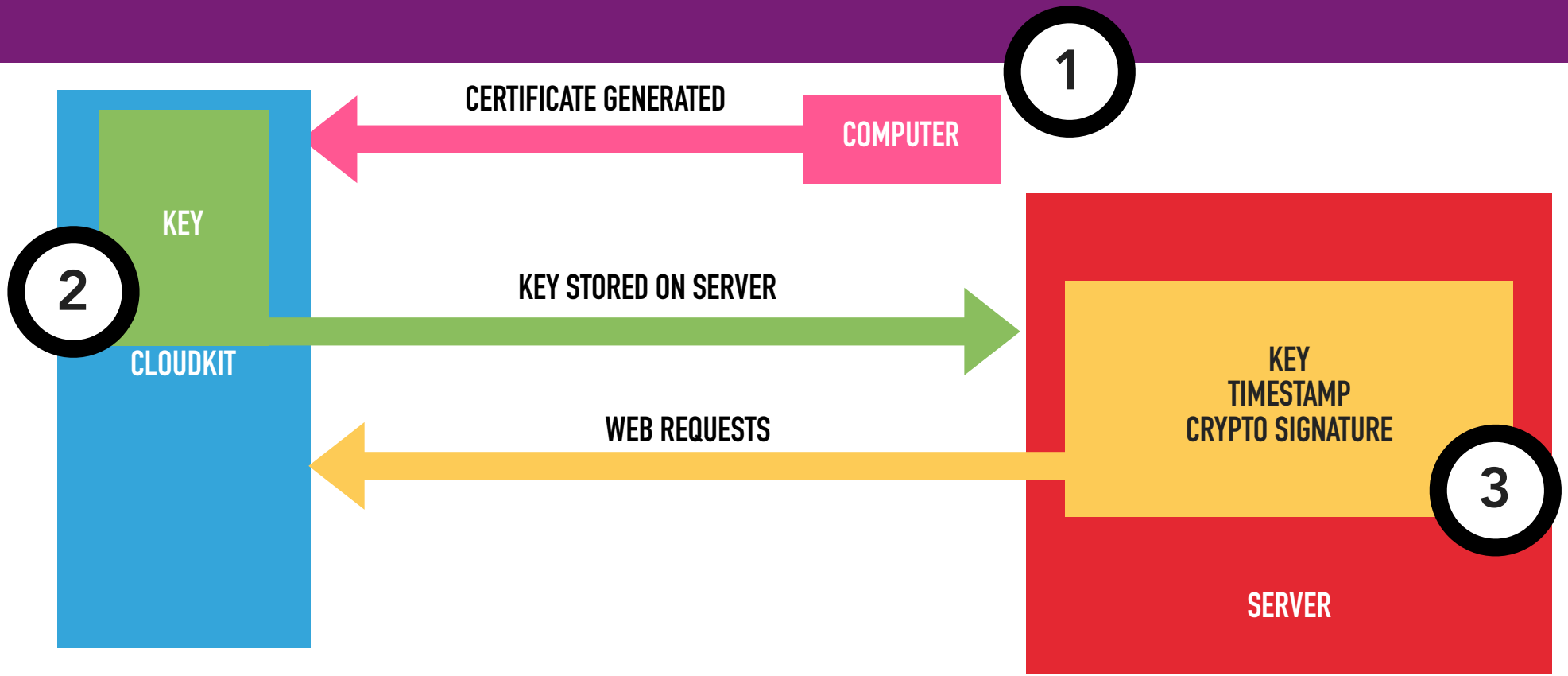
- API calls can only be made to the public database
- Calls run as with the inherited privileges of the creator of the key

SERVER-TO-SERVER

- Apple provides demo for node.js 👎



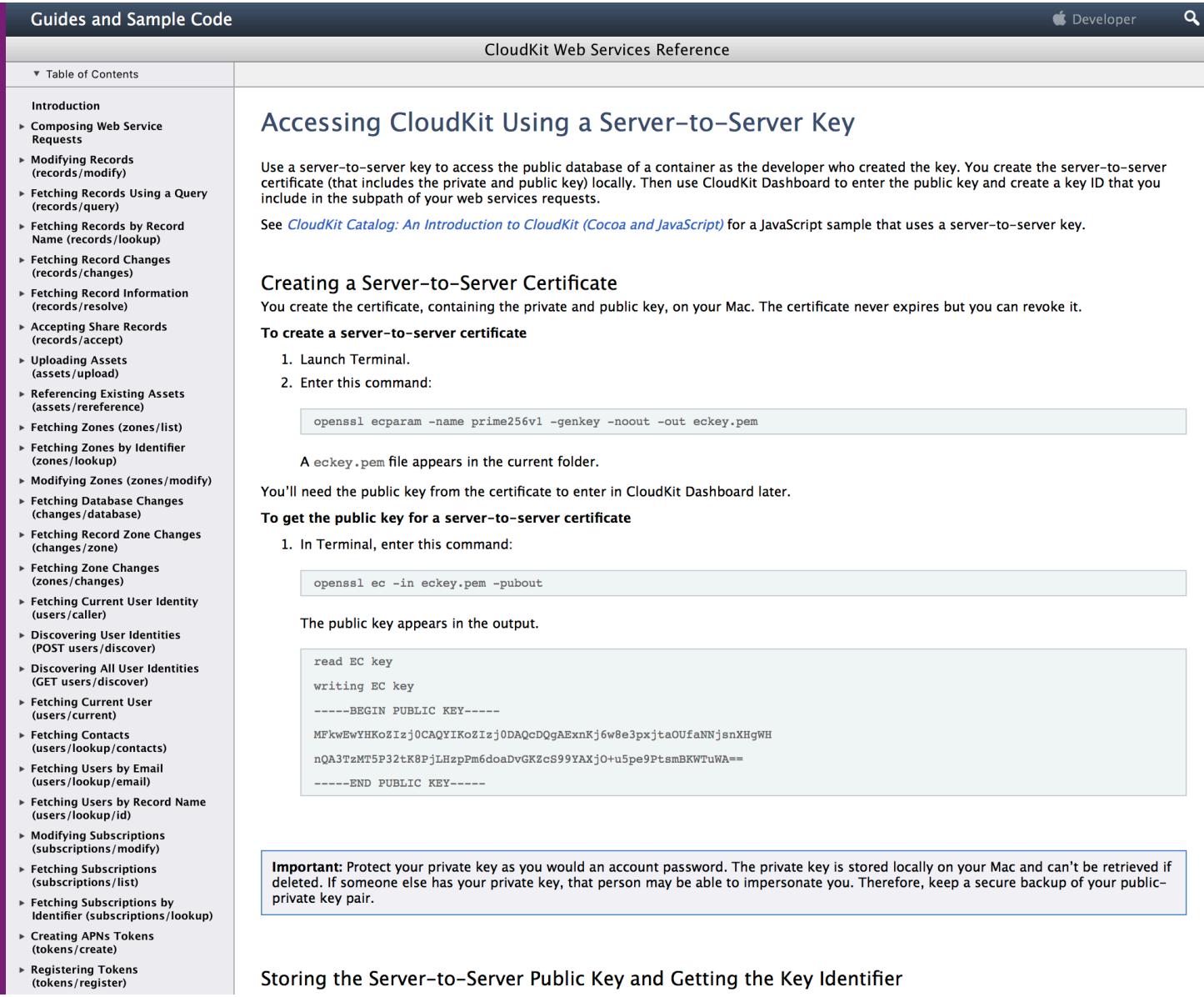
SERVER-TO-SERVER



SERVER-TO-SERVER KEY

SERVER-TO-SERVER KEY

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



Guides and Sample Code

Developer

CloudKit Web Services Reference

Table of Contents

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-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAExnKj6w8e3pxjtaOUfaNNjsnXHgWH
nQA3TzMT5P32tK8PjLHzpPm6doaDvGKZcS99YAXjO+u5pe9PtsmBKWTuWA==
-----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

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.

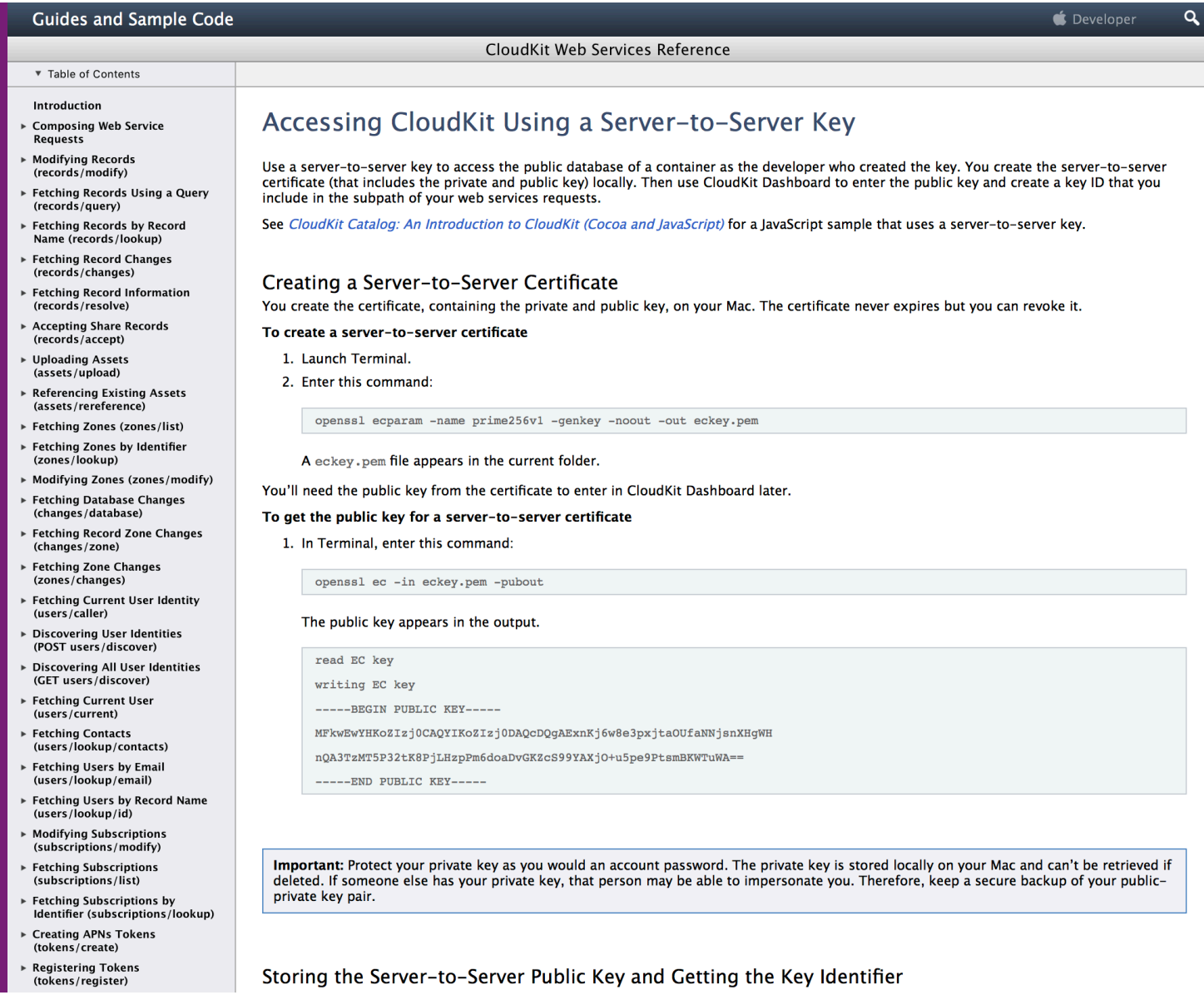
```
read EC key
writing EC key
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAExnKj6w8e3pxjtaOUfaNNjsnXHgWH
nQA3TzMT5P32tK8PjLHzpPm6doaDvGKZcS99YAXjO+u5pe9PtsmBKWTuWA==
-----END PUBLIC KEY-----
```

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

Developer

CloudKit Web Services Reference

Table of Contents

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-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAExnKj6w8e3pxjtaOUfaNNjsnXHgWH
nQA3TzMT5P32tK8PjLHzpPm6doaDvGKZcS99YAXjO+u5pe9PtsmBKWTuWA==
-----END PUBLIC KEY-----
```

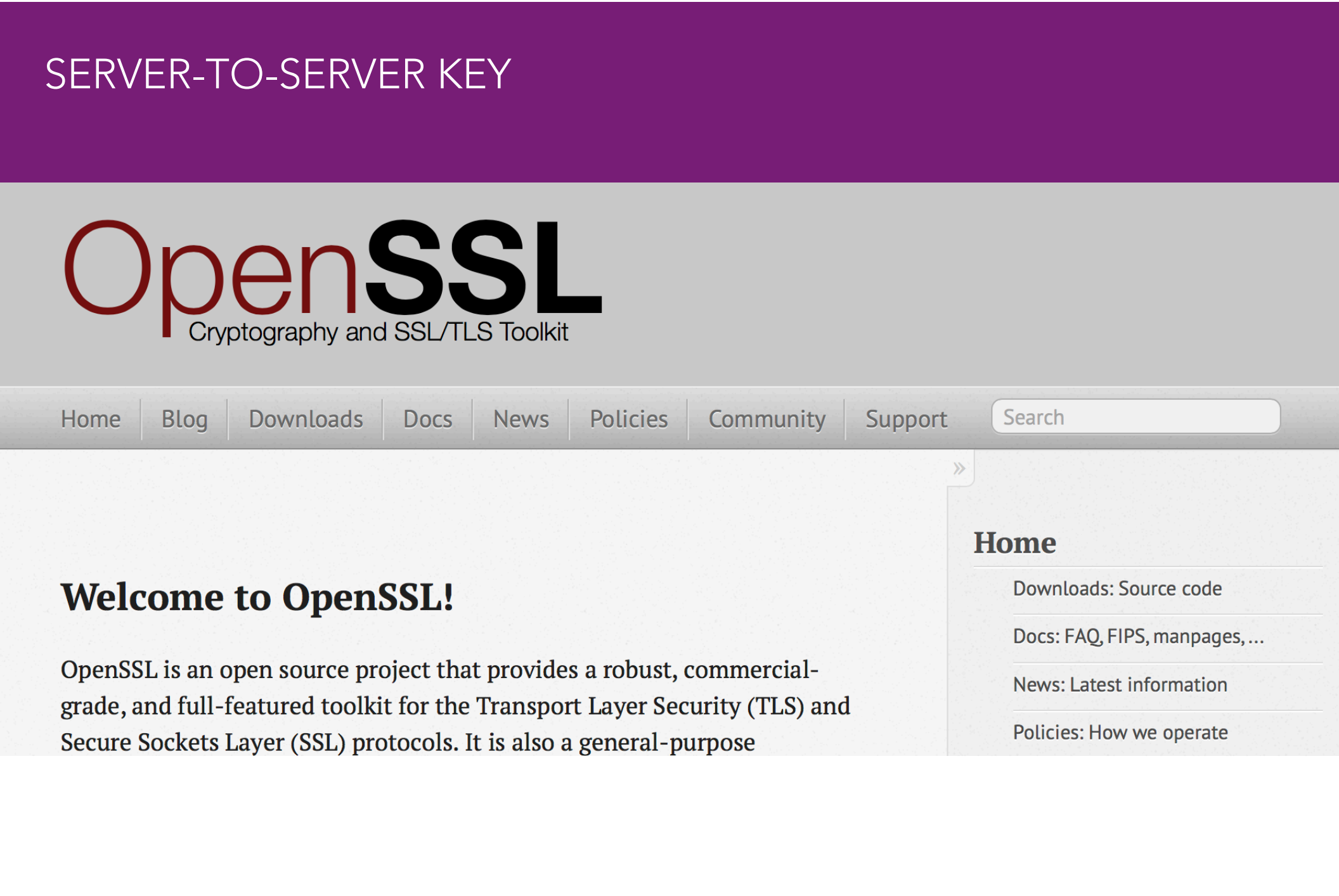
Important: Protect your private key as you would an account password. The private key is stored locally on your Mac and can't be retrieved if deleted. If someone else has your private key, that person may be able to impersonate you. Therefore, keep a secure backup of your public-private key pair.

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

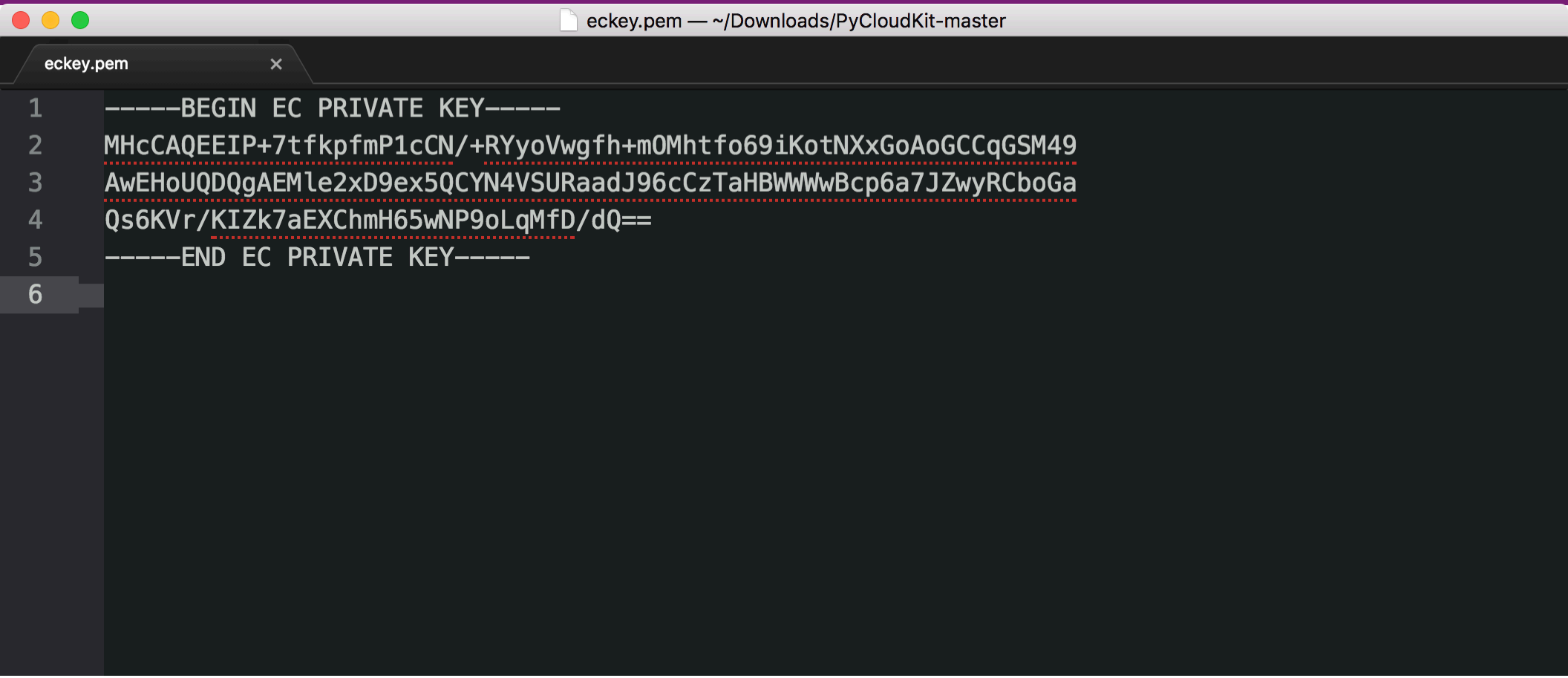
SERVER-TO-SERVER KEY

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

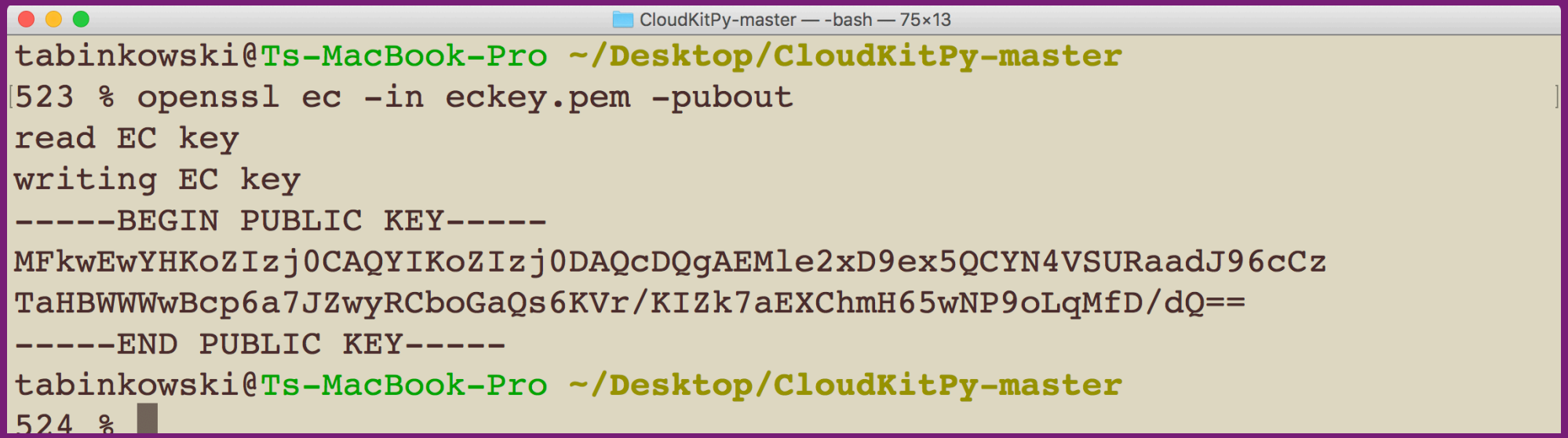
SERVER-TO-SERVER KEY



SERVER-TO-SERVER KEY

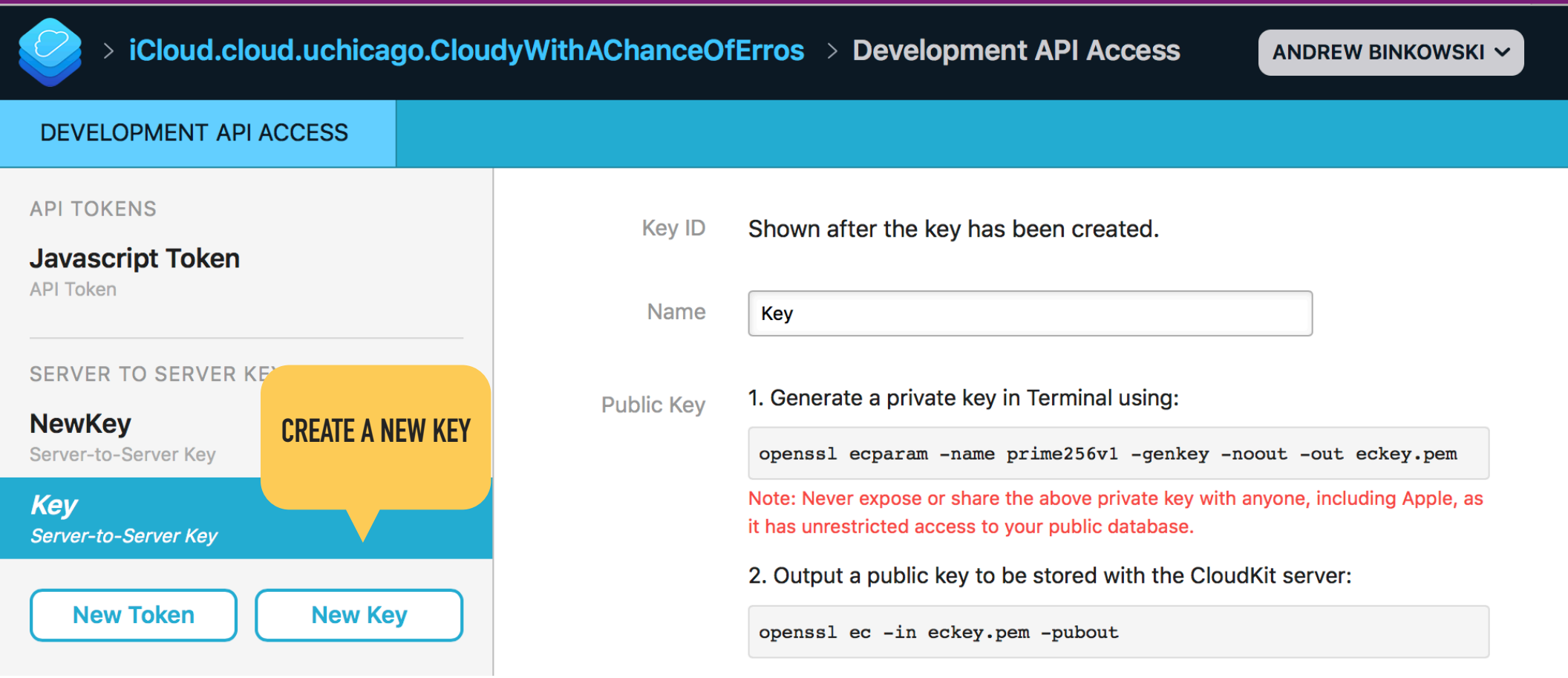


SERVER-TO-SERVER KEY



- Generate the public key
- Enter in CloudKit console

SERVER-TO-SERVER KEY



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 ffdb30c23addd822e851e15cdd30ea819251d2fbb9bc59c4803f006dcc900191

Name NewKey

Public Key MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEMle2xD9ex5QCYN4VSURaadJ96cCz
TaHBWWWwBcp6a7JZwyRCboGaQs6KVr/KIZk7aEXChmH65wNP9oLqMfD/dQ==

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 Ser
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 Da
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. 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 `null`.
`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 `null`, which fetches all record fields.
`zoneWide`	A Boolean value determining whether all zones should be searched. This key is ignored if `zoneID` is non-`null`. To search all zones, set to `true`. To search the default zone only, set to `false`.
`numbersAsStrings`	A Boolean value indicating whether number fields should be represented by strings. The default value is `false`.

- 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 `null`, which fetches all record fields. This `desiredKeys` setting overrides the `desiredKeys` 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 `null`.
`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 `continuationMarker` key as the value of the `continuationMarker` 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

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'}
```

API URL

QUERY JSON

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 `true`, the entire request fails if one operation fails. If `false`, some operations may succeed and others may fail. The default value is `true`. 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 `null`, which fetches all record fields.
`numbersAsStrings`	A Boolean value indicating whether number fields should be represented by strings. The default value is `false`.

- 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":
{"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.

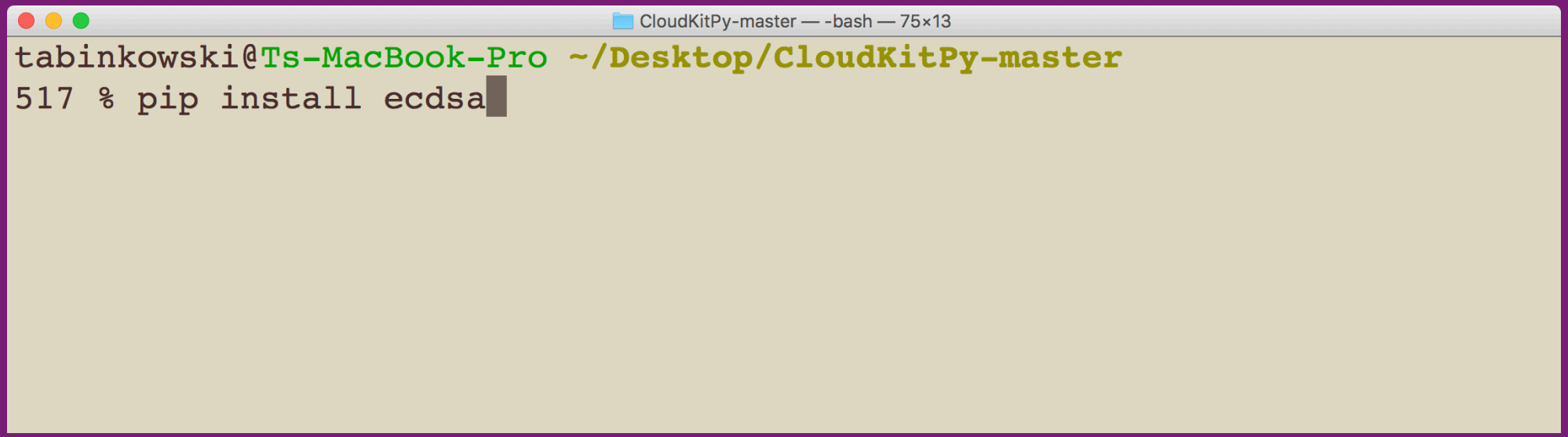
- Response is a dictionary with more information than you probably need

WEB SERVICE REQUESTS

- Response dictionary


```
{
    "created": {
        "deviceID": "2",
        "timestamp": 1494288550303,
        "userRecordName": "_7b892f2a3eeaddd6afc77ff7158f69d9"
    },
    "fields": {
        "question": {
            "type": "STRING",
            "value": "Why?\n"
        },
        "response": {
            "type": "STRING",
            "value": "That's why!"
        }
    },
    "modified": {
        "deviceID": "2",
        "timestamp": 1494386749697,
        "userRecordName": "_7b892f2a3eeaddd6afc77ff7158f69d9"
    },
    "pluginFields": {},
    "recordChangeTag": "j2gstkiq",
    "recordName": "60d82b7d-32ef-4e91-9c12-585c92a3bb89",
    "recordType": "joke",
    "zoneID": {
        "ownerRecordName": "_7b892f2a3eeaddd6afc77ff7158f69d9",
        "zoneName": "_defaultZone"
    }
```

SERVER-TO-SERVER KEY

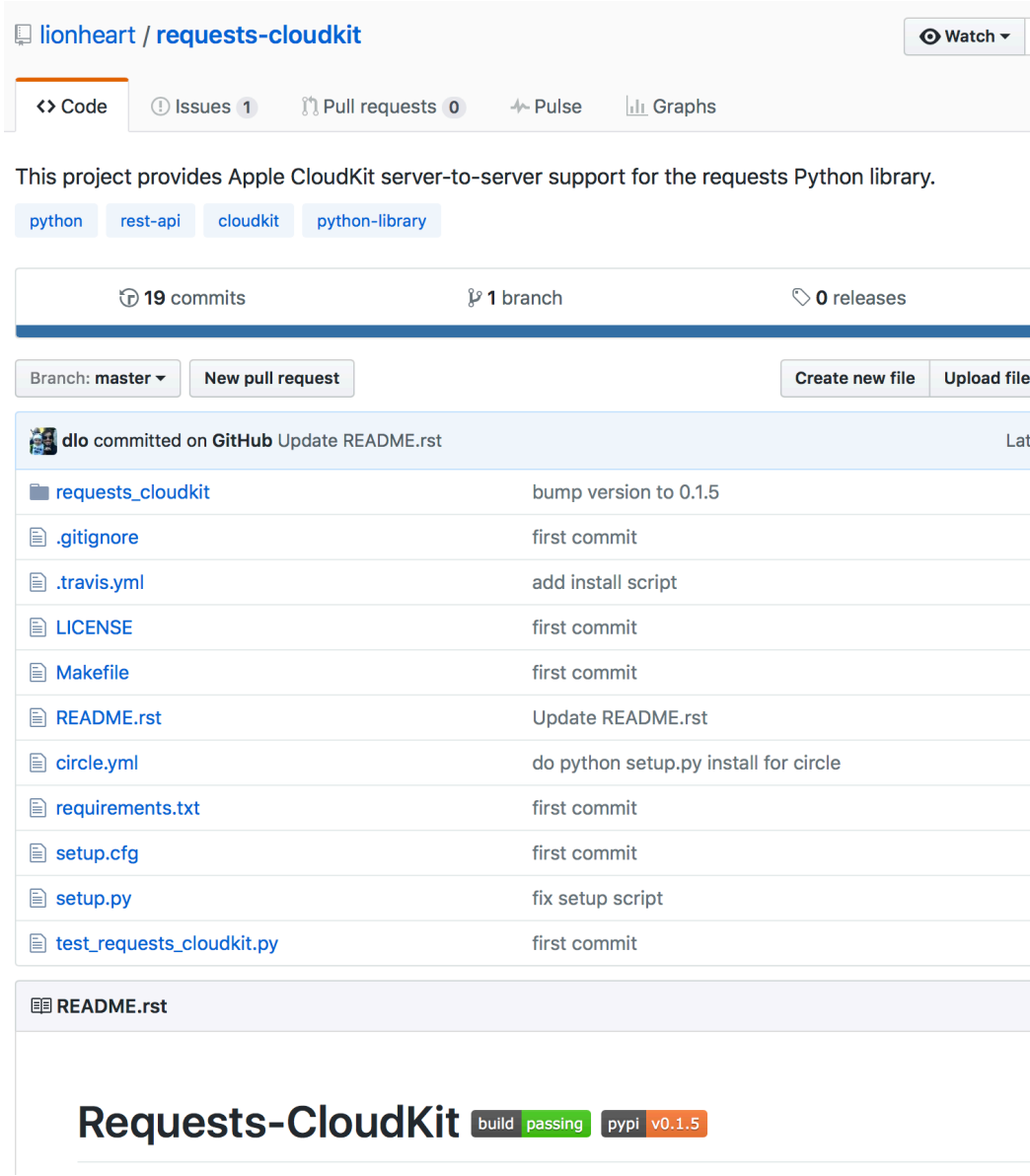


- Elliptic Curve Digital Signature Algorithm used to sign requests
- Cryptography approach for public/private key agreement

SERVER-TO-SERVER DEMO

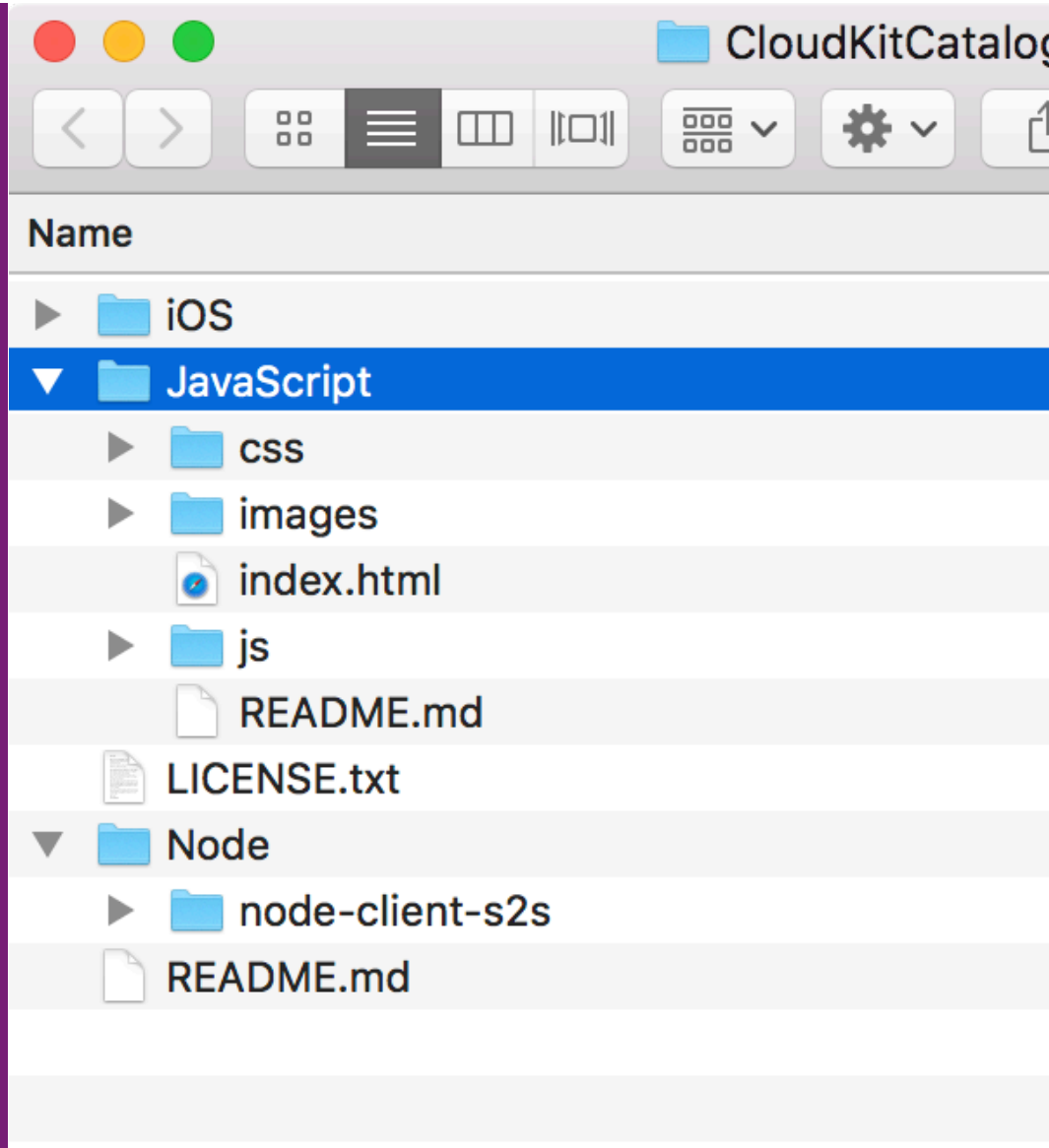
SERVER-TO-SERVER KEY

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



SERVER-TO-SERVER KEY

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

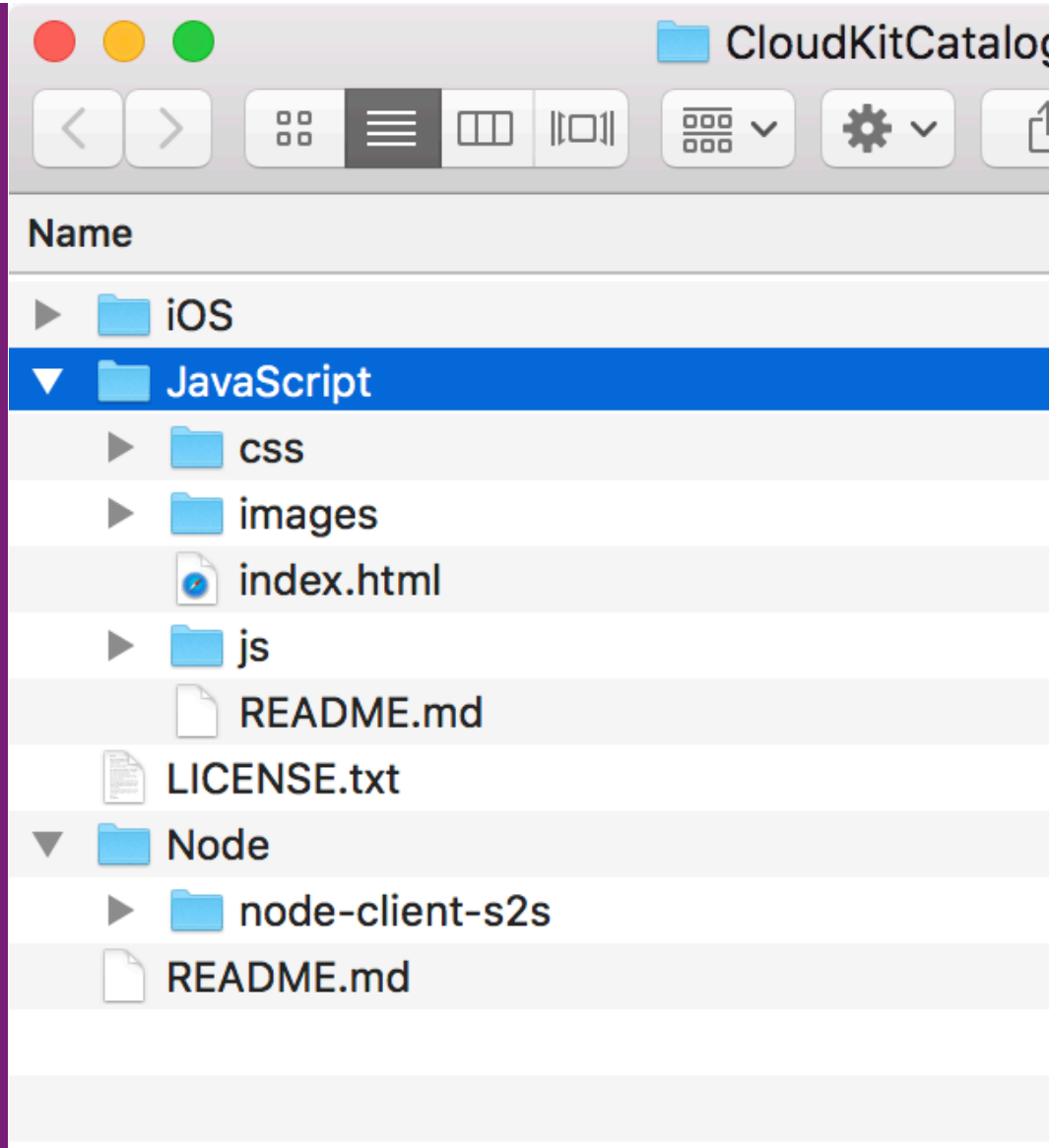


SERVER-TO-SERVER KEY

```
/*
 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({
```

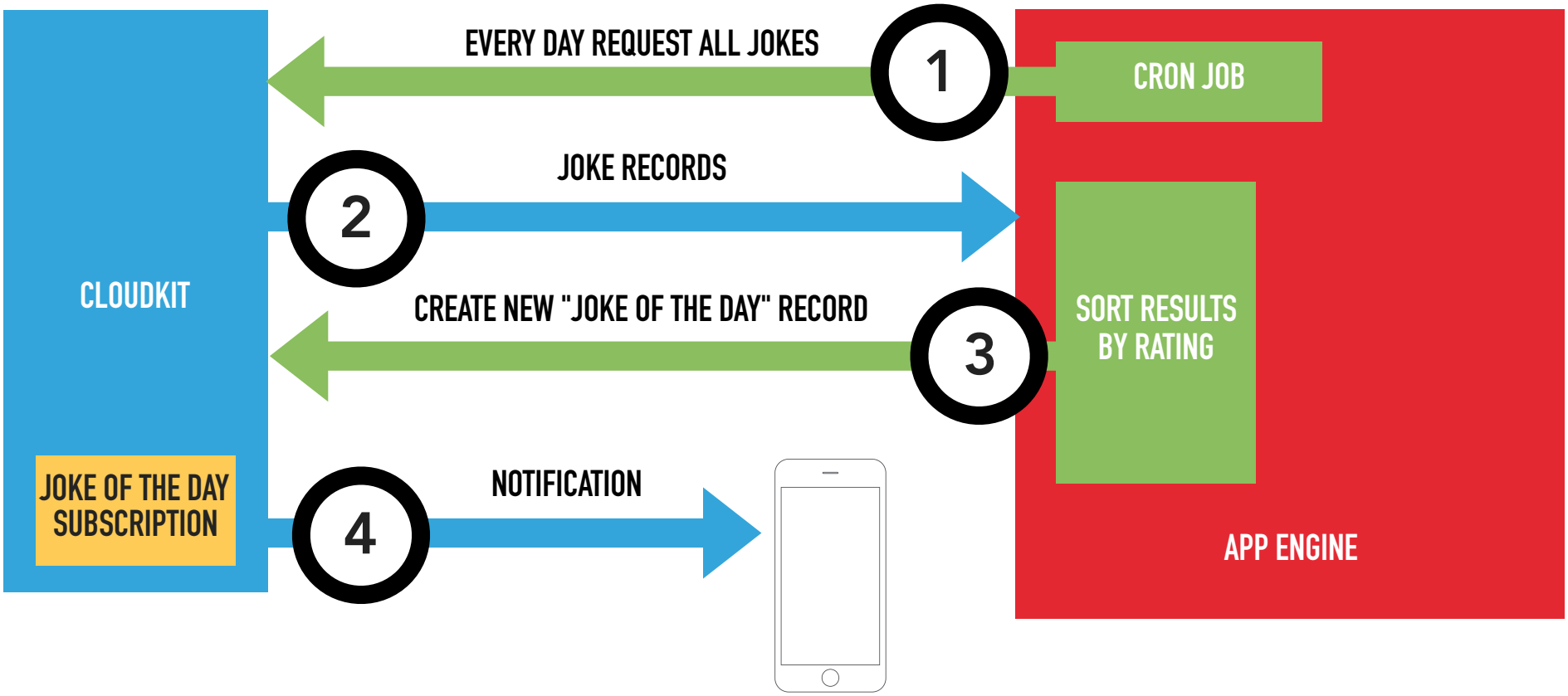
SERVER-TO-SERVER KEY

- Python so that we can use App Engine



CLOUDKIT HELPER

SERVER-TO-SERVER KEY



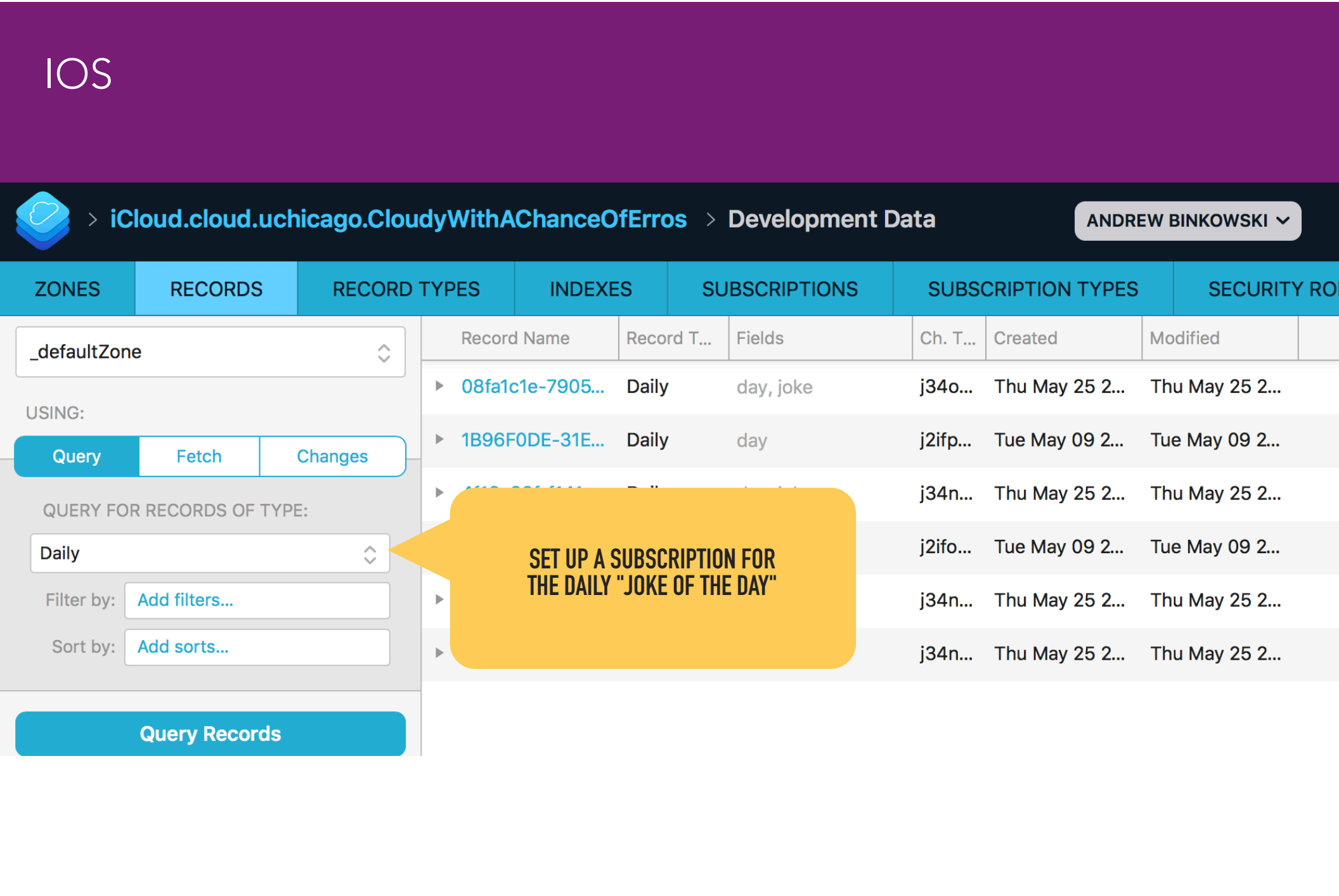
SERVER-TO-SERVER KEY

- 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.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

    # 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.CloudyWithAChanceOfErros
Development Data
ANDREW BINKOWSKI
ZONES
RECORDS
RECORD TYPES
INDEXES
SUBSCRIPTIONS
SUBSCRIPTION TYPES
SECURITY RO
_defaultZone
USING:
Query
Fetch
Changes
QUERY FOR RECORDS OF TYPE:
Daily
Filter by:
Add filters...
Sort by:
Add sorts...
Query Records
Record Name
Record T...
Fields
Ch. T...
Created
Modified
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...
j34n...
Thu May 25 2...
Thu May 25 2...
j2ifo...
Tue May 09 2...
Tue May 09 2...
j34n...
Thu May 25 2...
Thu May 25 2...
j34n...
Thu May 25 2...
Thu May 25 2...
SET UP A SUBSCRIPTION FOR
THE DAILY "JOKE OF THE DAY"

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 = "\(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"]

// Crea
let sub                                    n(recordType: "Daily",
                                             predicate: NSPredicate(value:
                                             subscriptionID: identifier,
                                             options: [.firesOnRecordCreat
subscription.notificationInfo = notificationInfo
```

KEYS TO SEND WITH RECORD TYPE

- 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
#

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
#

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 = 'ffdb30c23addd822e851e15cdd30ea819251d2fbb9bc59c4803f006dcc900191'
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

INSTALL THE AN EXTERNAL LIBRARY
IN YOUR PROJECT

```
cd gae_dir

mkdir lib

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

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')
```

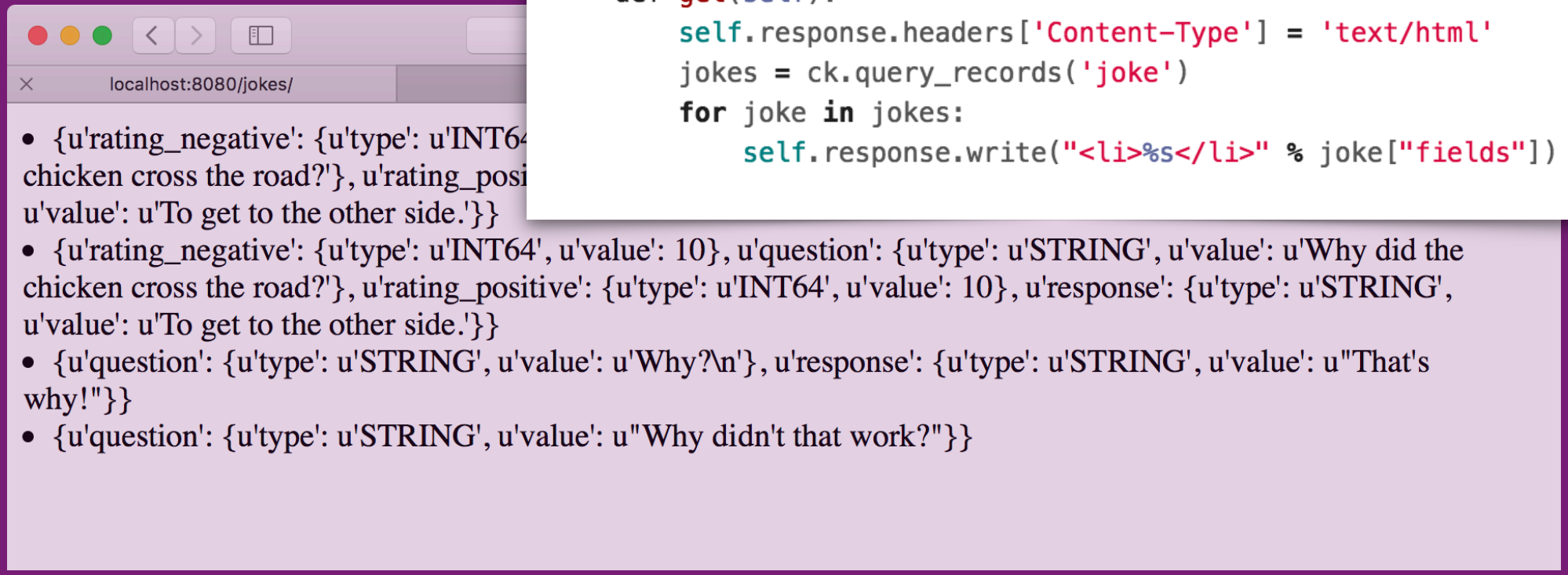
APP ENGINE

- 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)
```

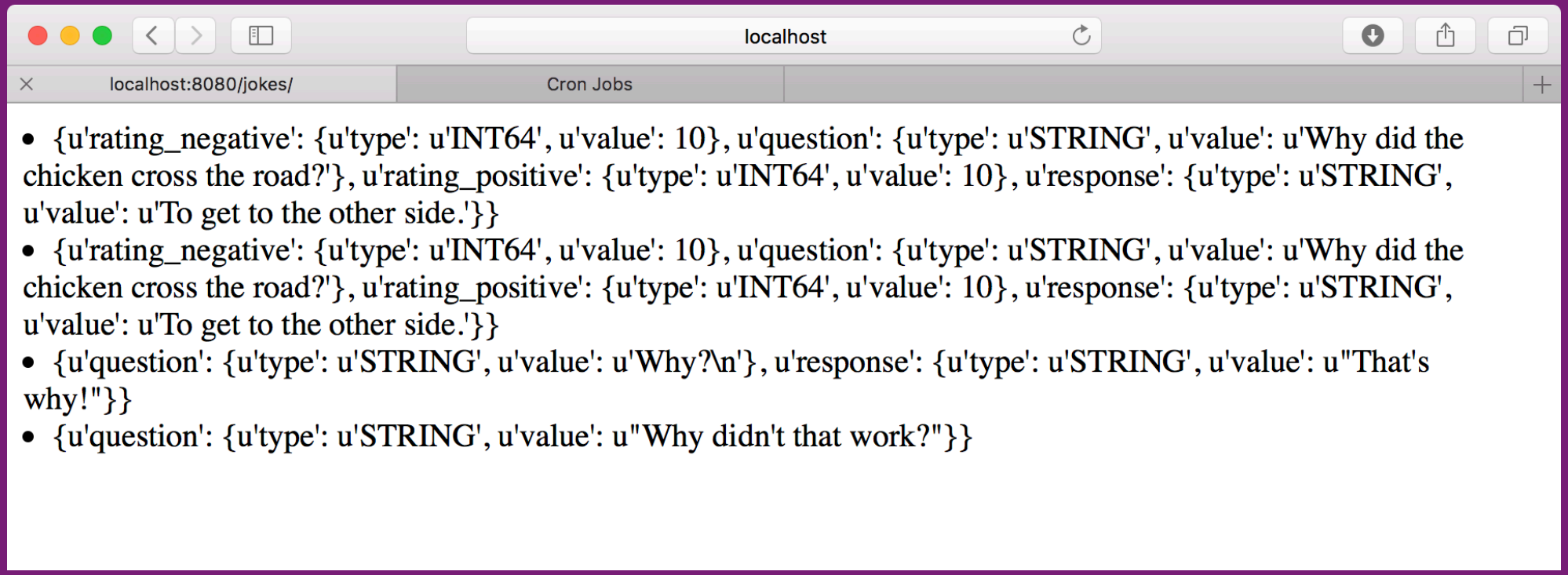
APP ENGINE

```
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"])
```



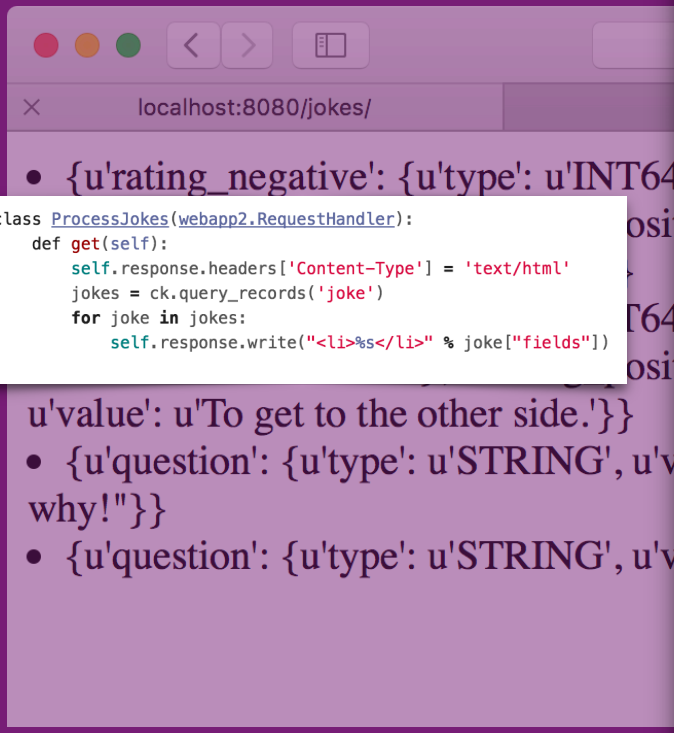
- http://localhost:8080/jokes/

APP ENGINE



- http://localhost:8080/jokes/

APP ENGINE



- http://localhost:8080/jok

```
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
```

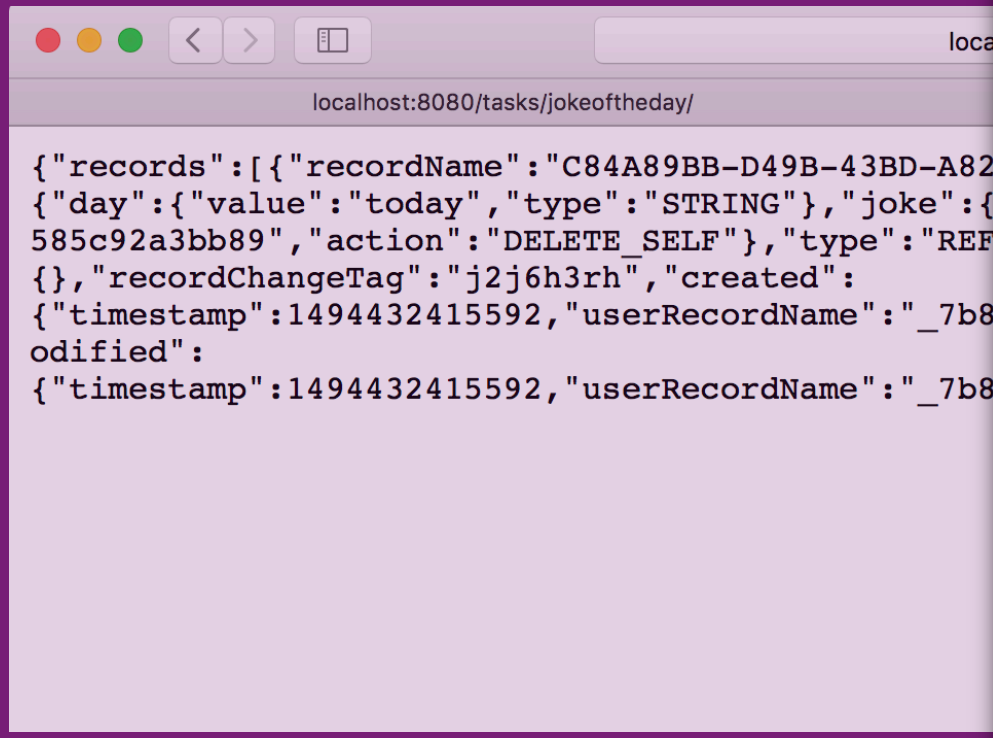
REQUEST WITH CURSORS

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/

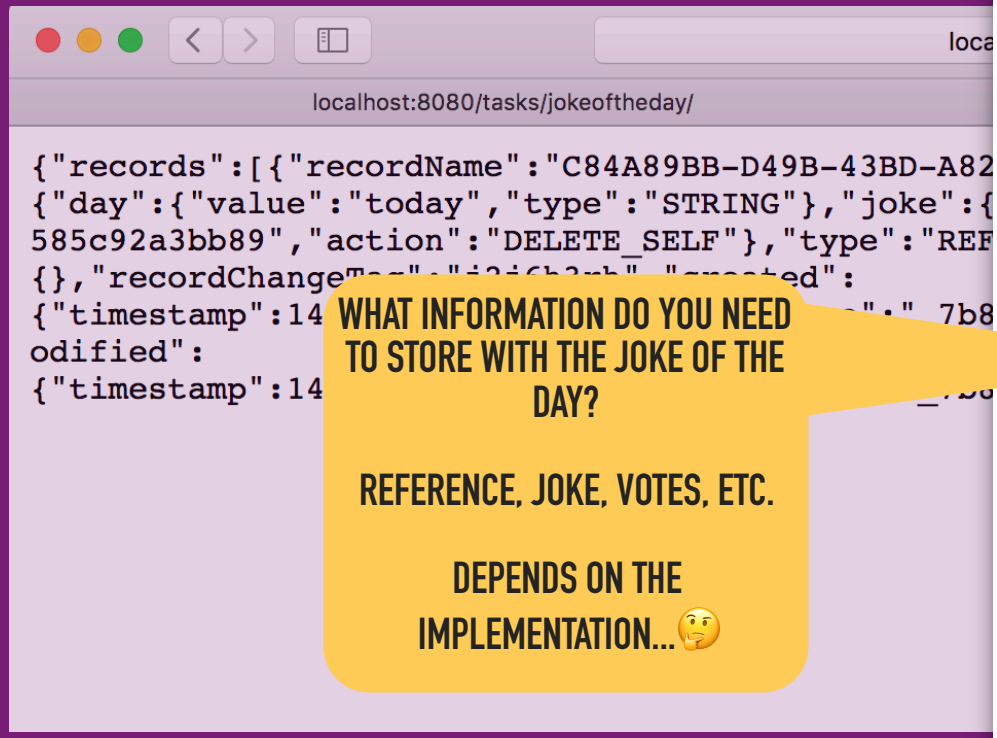
APP ENGINE



- http://localhost:8080/tasks/jokeofthe

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


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

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

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

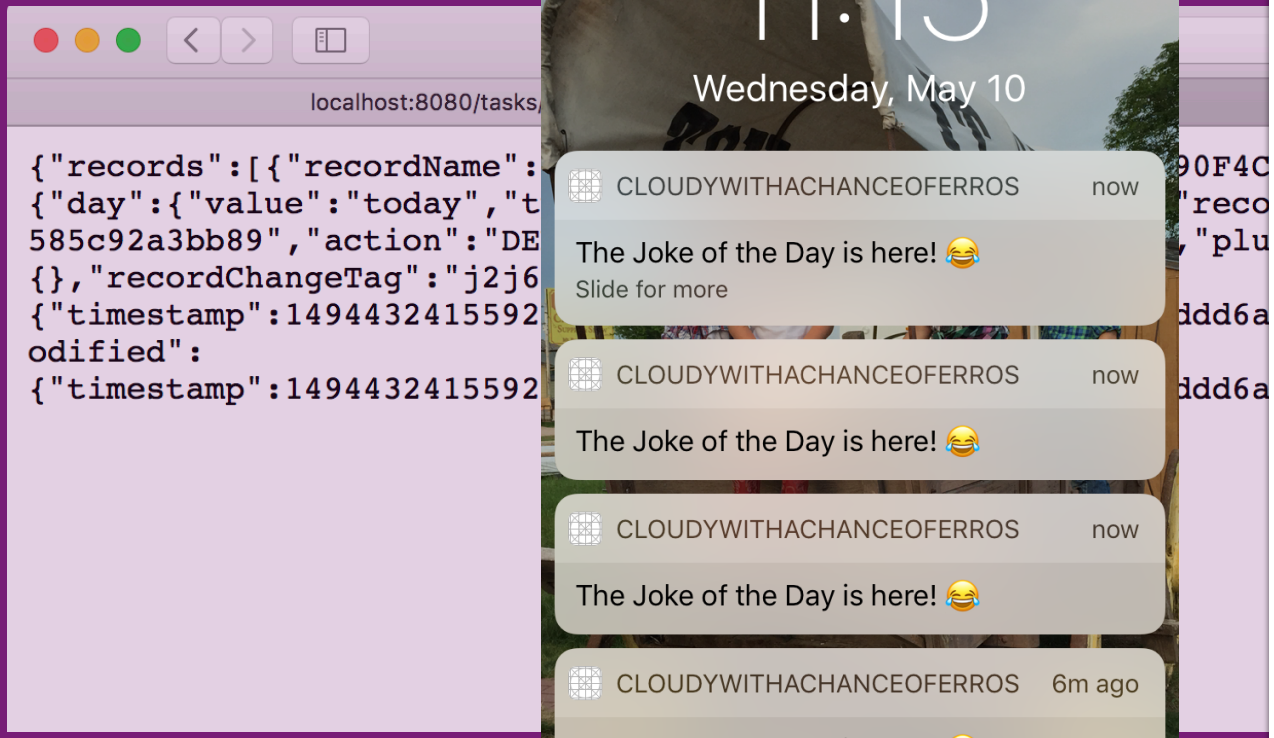
APP ENGINE




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


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

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

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

- http://localhost:8080/tasks/jokeofthe

APP ENGINE




```python
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
```

- http://localhost:808

APP ENGINE

```yaml
# cron.yaml
#
# To test in developement 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

Google App Engine

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

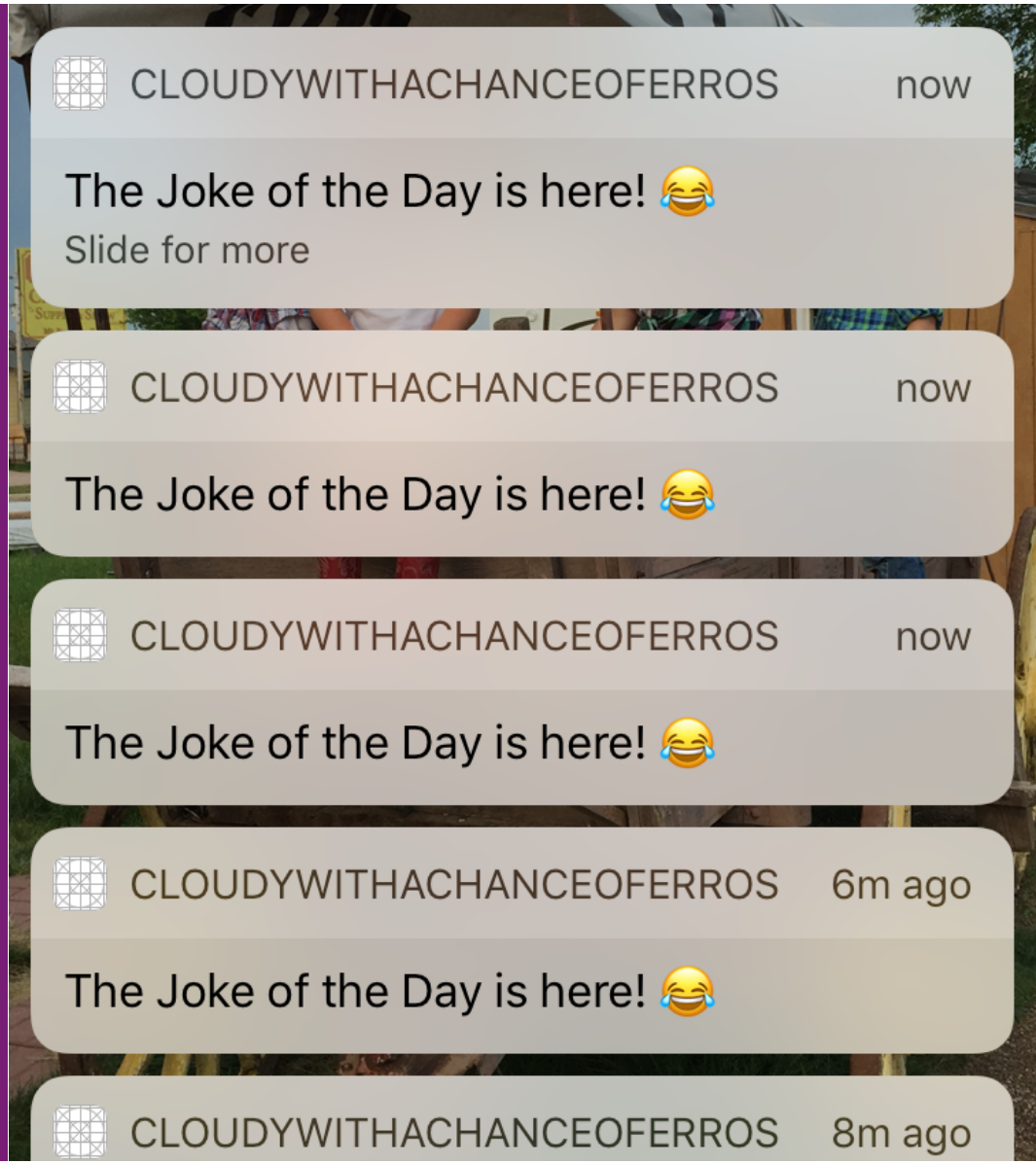
Request to /tasks/jokeoftheday/ succeeded!

Cron Job	Schedule
/tasks/jokeoftheday/ daily joke summary and notification	**every day 08:00** Run now 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

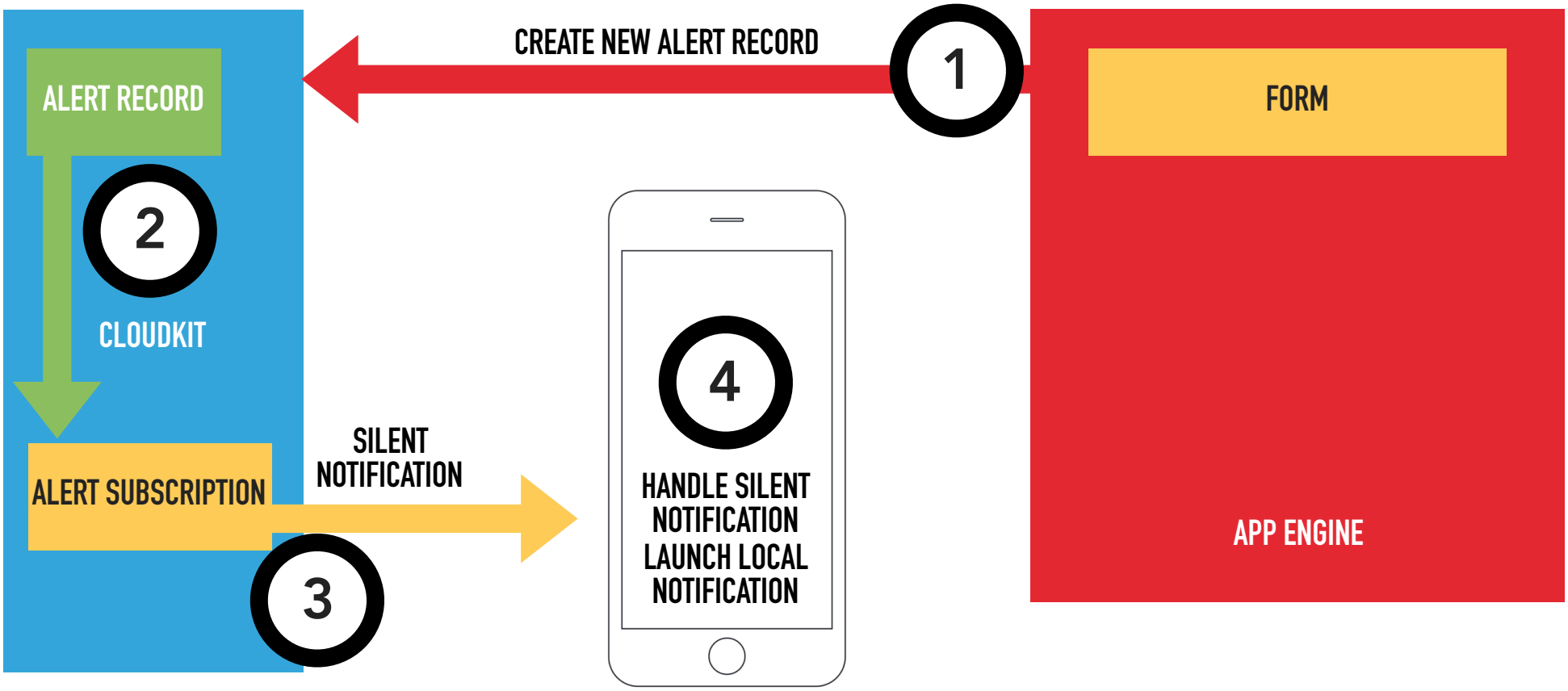
CUSTOM NOTIFICATIONS FROM SUBSCRIPTIONS

CUSTOM NOTIFICATIONS

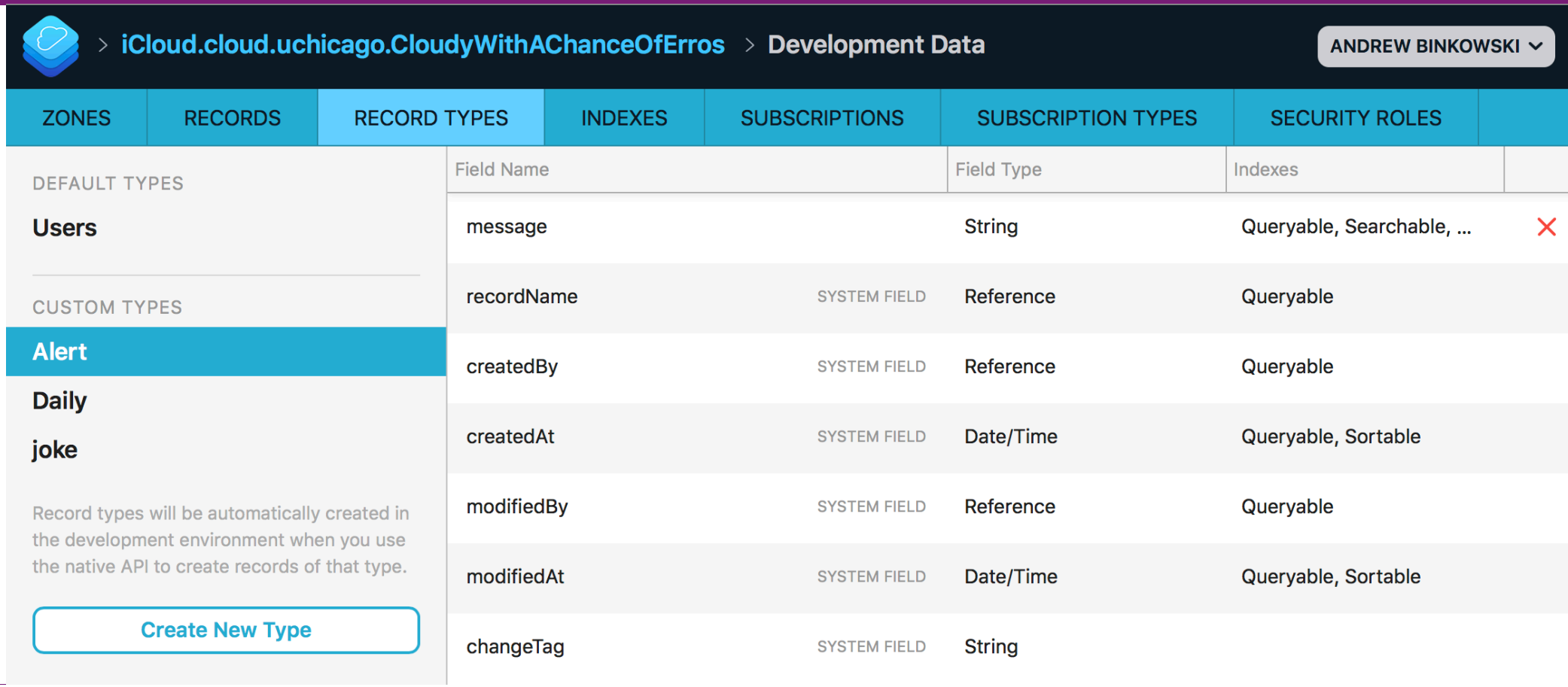
- Generic notifications from subscriptions
- This may not be the great user experience we want



CUSTOM NOTIFICATIONS



CUSTOM NOTIFICATIONS



CUSTOM NOTIFICATIONS

- Set up a new route

WEB APP TO SEND NOTIFICATIONS

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

], debug=True)
```

CUSTOM NOTIFICATIONS

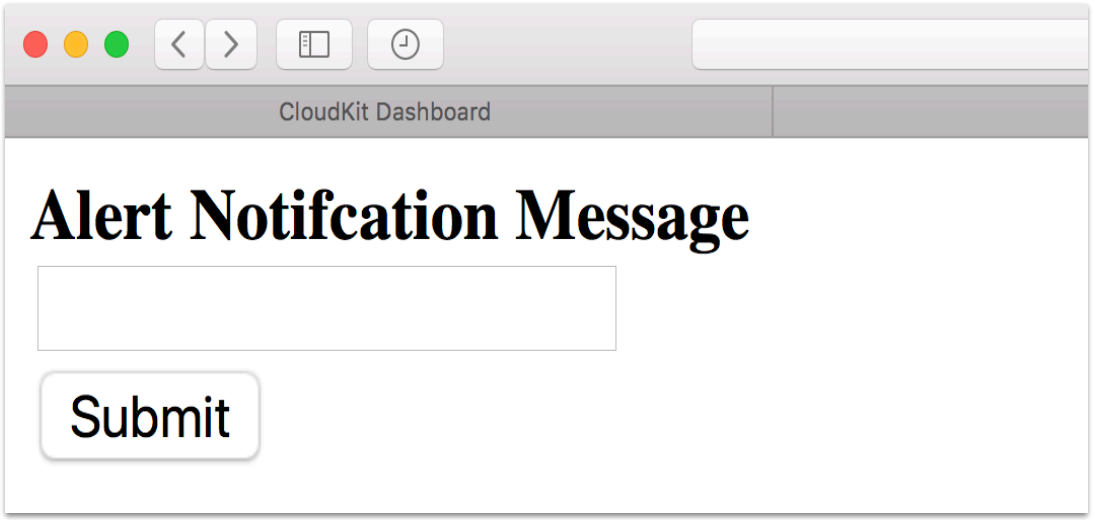
- Create a new Alerts record with the passed in message

```
class AlertNotifications(webapp2.RequestHandler):
    def post(self):
        message = self.request.get('message')
        new_joke_of_the_day_data = {
        'operations': [{
            'operationType': 'create',
            'record': {
                'recordType': 'Alert',
                'fields': {
                    'message': {'value': message},
                }
            }
        }]
        }

        result_modify_jokes = ck.cloudkit_request(
        '/development/public/records/modify',
        json.dumps(new_joke_of_the_day_data))

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

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

CUSTOM NOTIFICATIONS

- Create an amazing form 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)
```

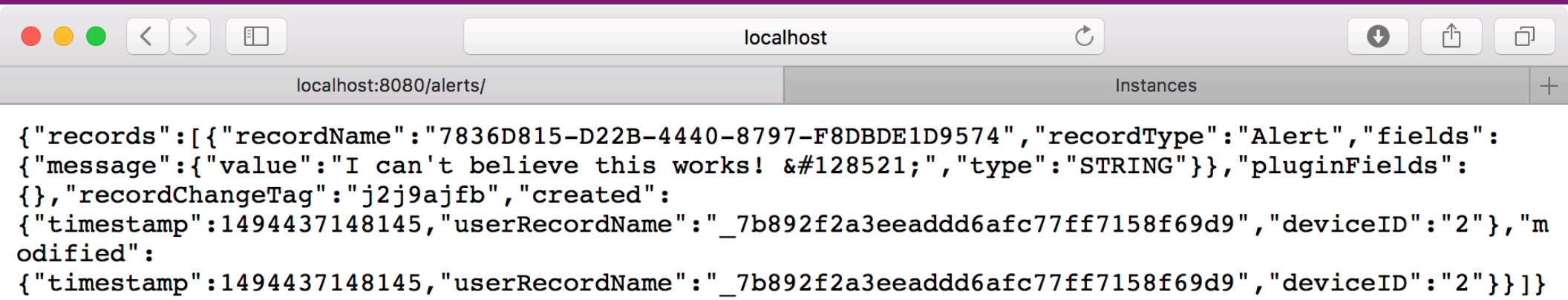


CUSTOM NOTIFICATIONS

- 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




```
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'])
```

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...AChanceOfErros";
    nid = "2b661f39-5617-4a1c-a88...09cd9eae7a";
    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 (UIBackgroundFet

  // 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, withCompletionHandler: { (error) in
      if let error = error {
        print("Something went wrong: \(error)")
      }
    })

    completionHandler(.newData)

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

BREAK TIME 

OPEN SOURCE SWIFT

OPEN SOURCE SWIFT

- It all started here

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.

Swift.org

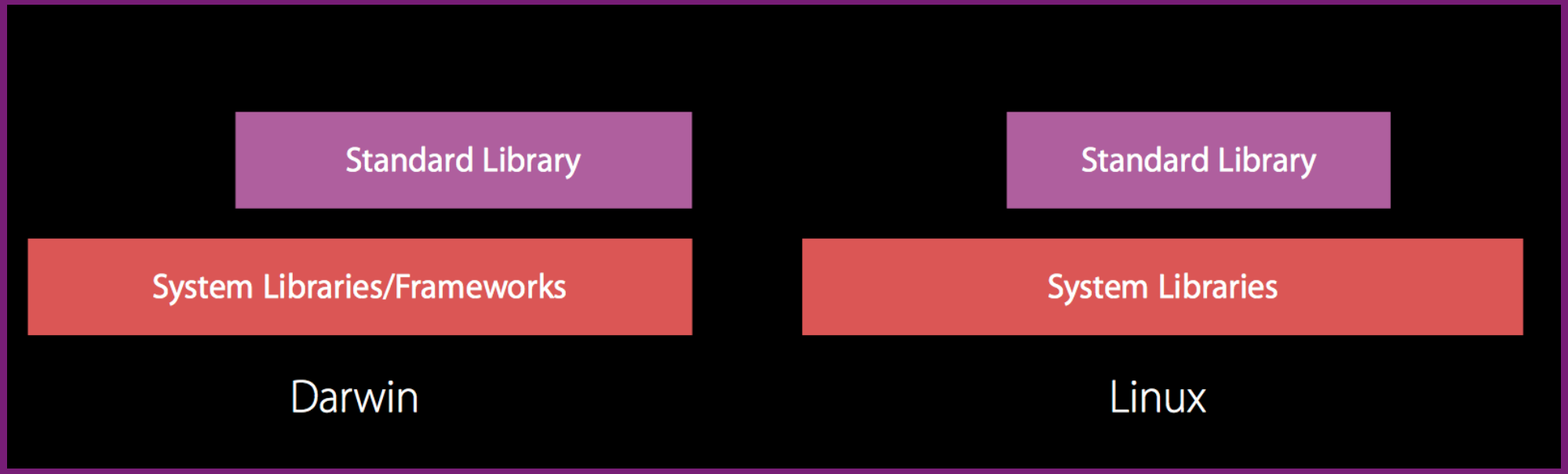
Swift.org is an entirely new site dedicated to open source Swift. This site hosts resources for the community of developers that want to help evolve Swift, contribute fixes, and most importantly, interact with each other. Swift.org hosts:

- 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

OPEN SOURCE SWIFT

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

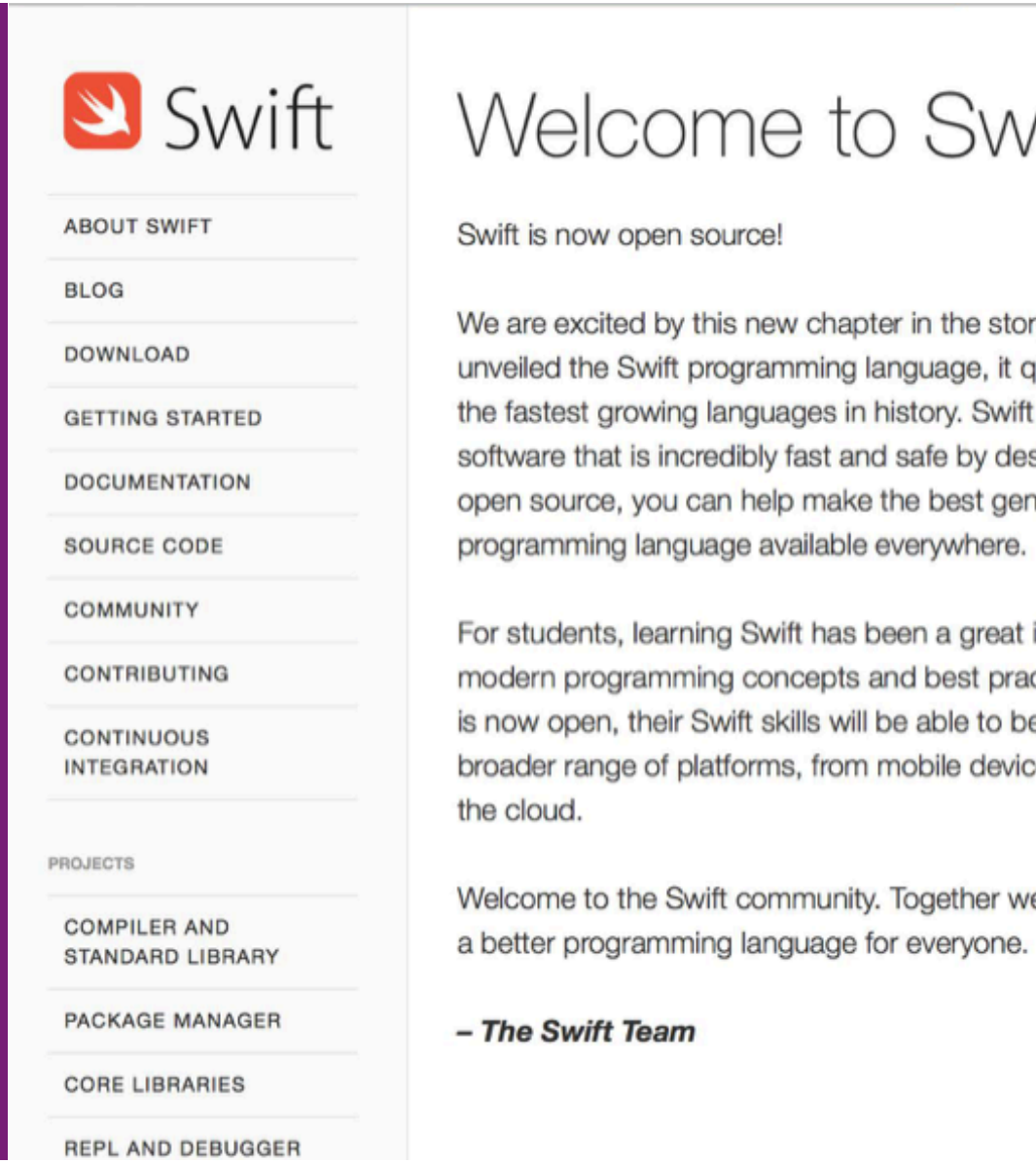
OPEN SOURCE SWIFT



- Something is missing...

OPEN SOURCE SWIFT

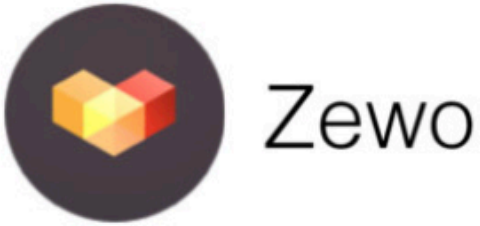
- Linux and Mac Platform support
- Standard Library Foundation, Dispatch, and XCTest Compiler
- Command Line Tools



STATE OF SWIFT ON THE SERVER

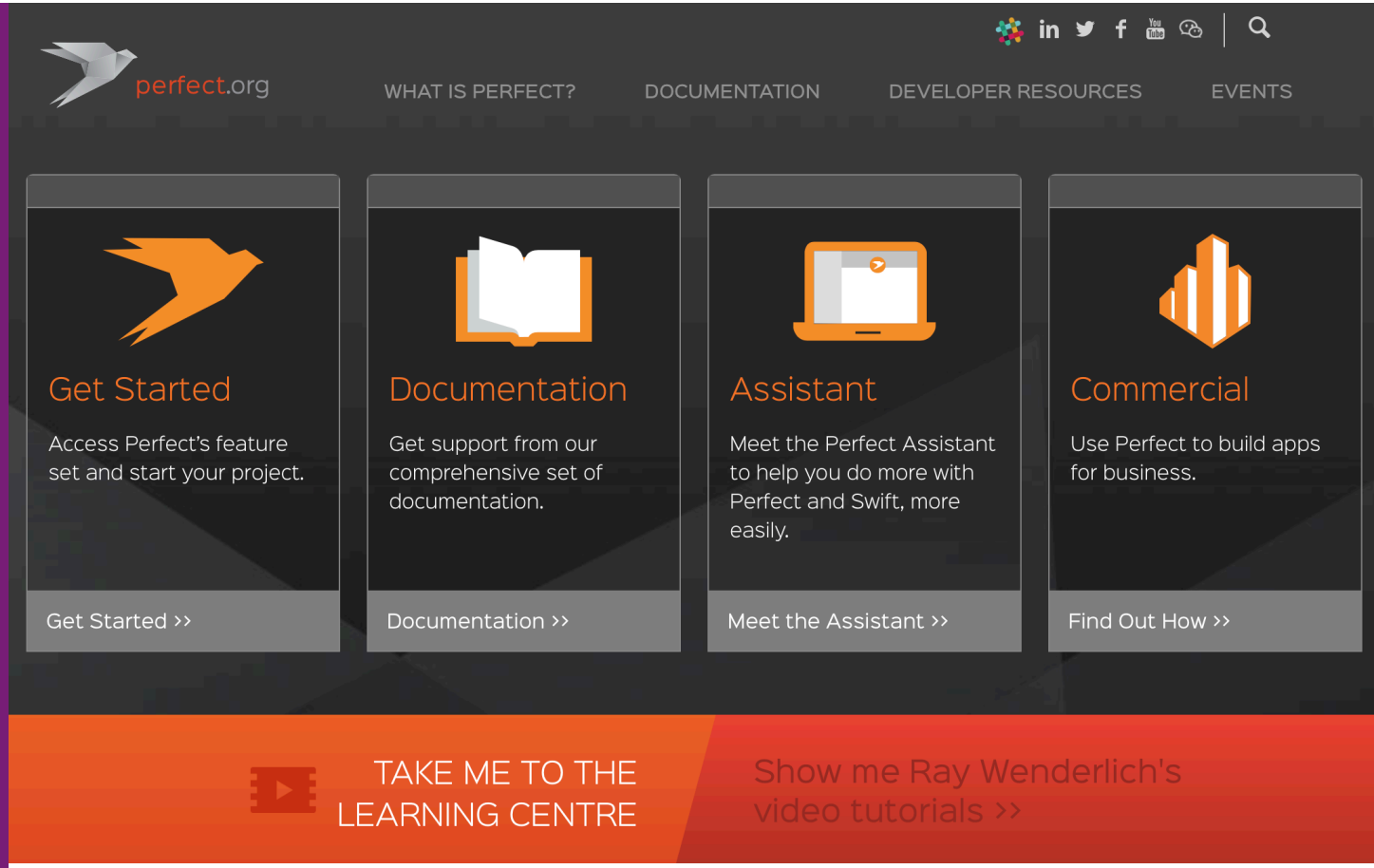
STATE OF SWIFT ON THE SERVER

- 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



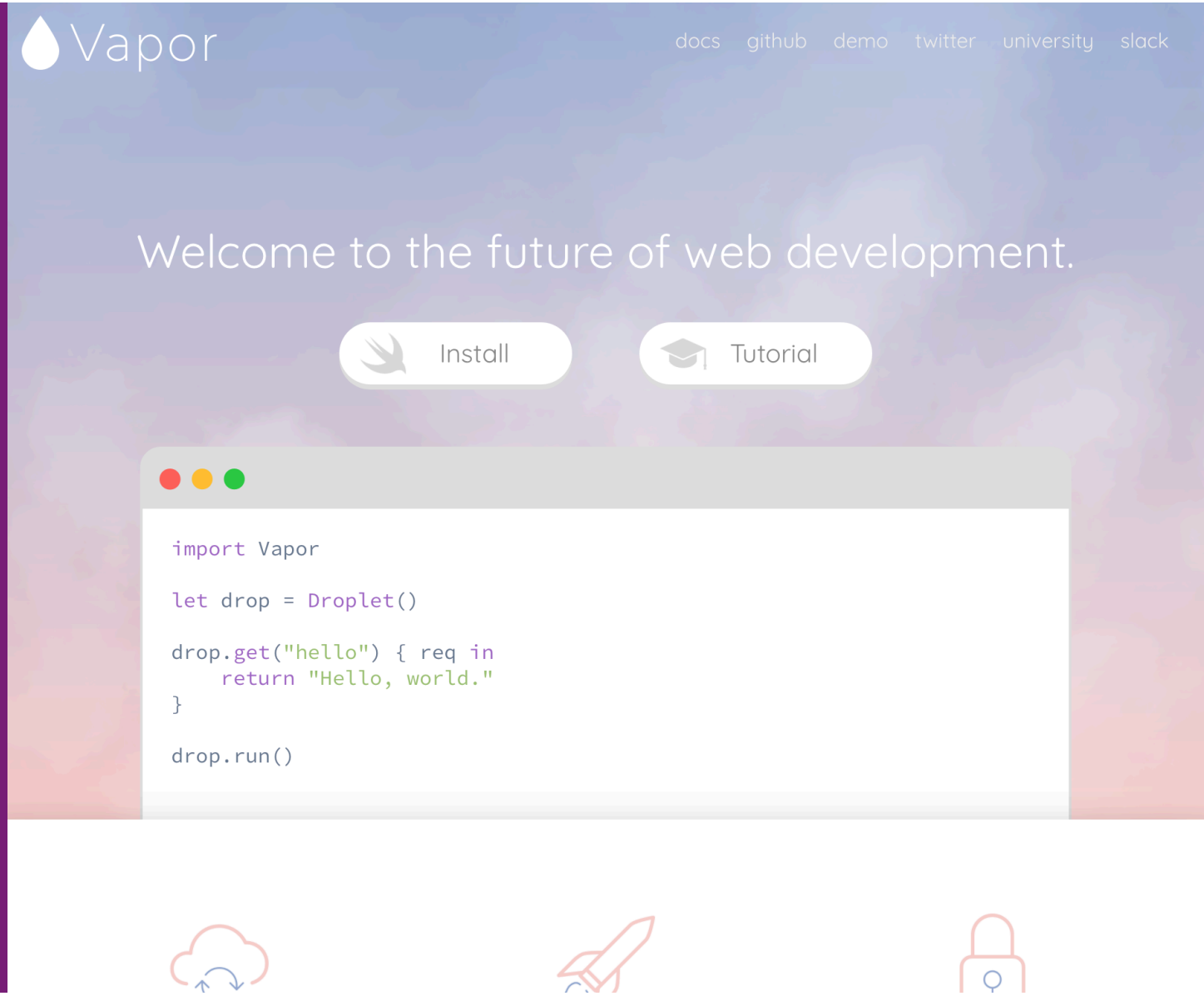
STATE OF SWIFT ON THE SERVER

- Perfect
  - A complete framework
  - Runs it 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



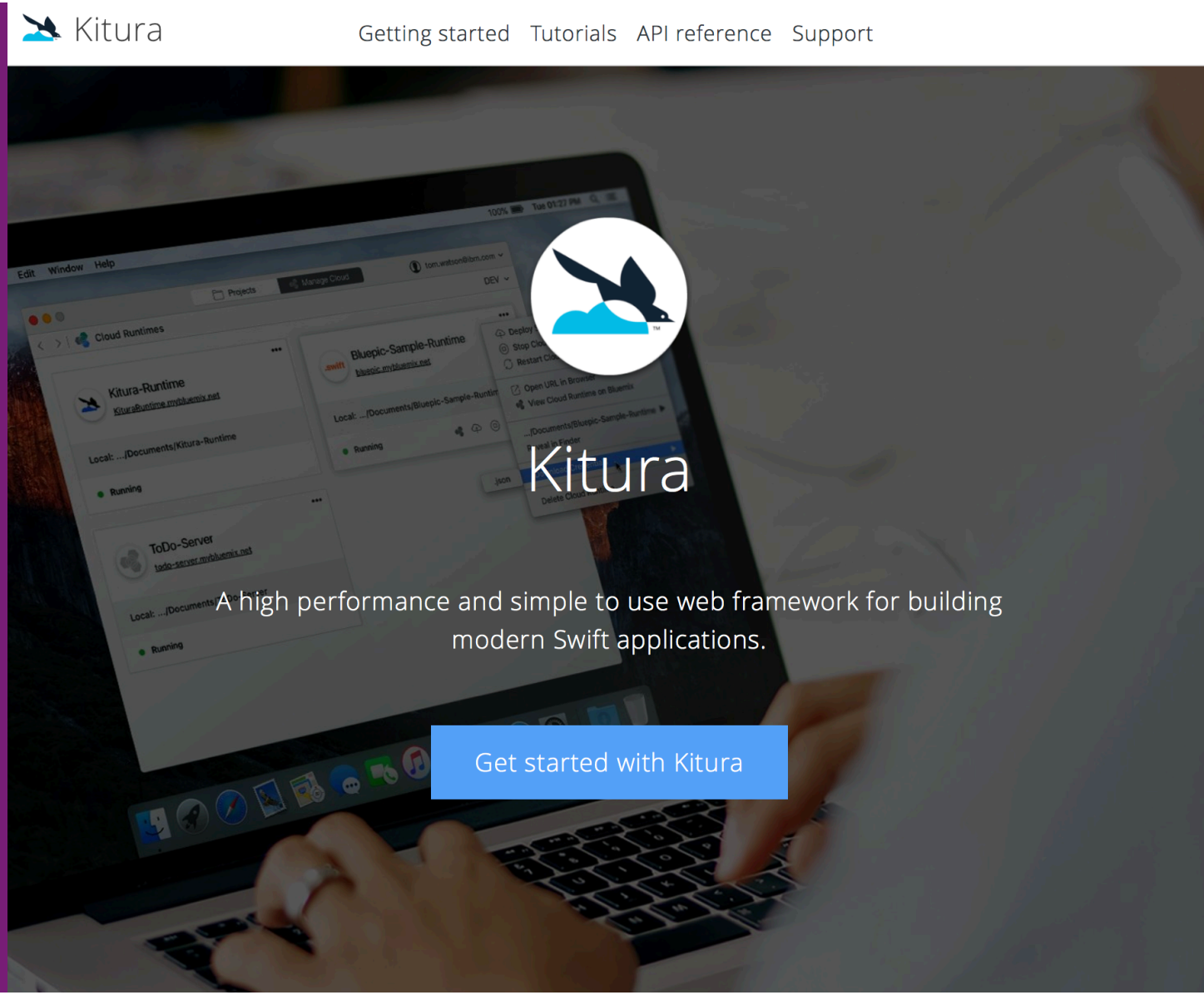
STATE OF SWIFT ON THE SERVER

- Perfect
  - A complete framework
  - Runs it 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



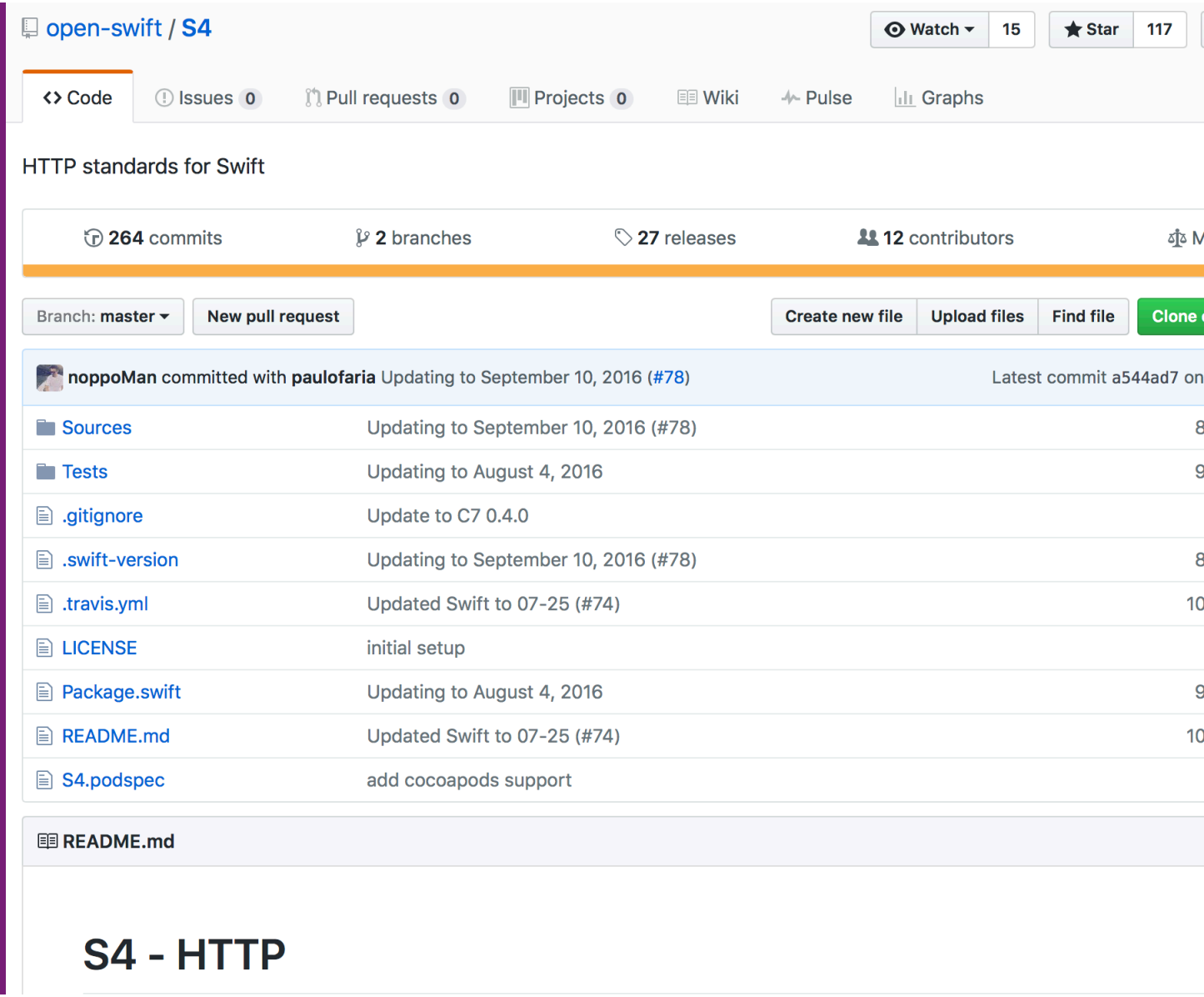
STATE OF SWIFT ON THE SERVER

- 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



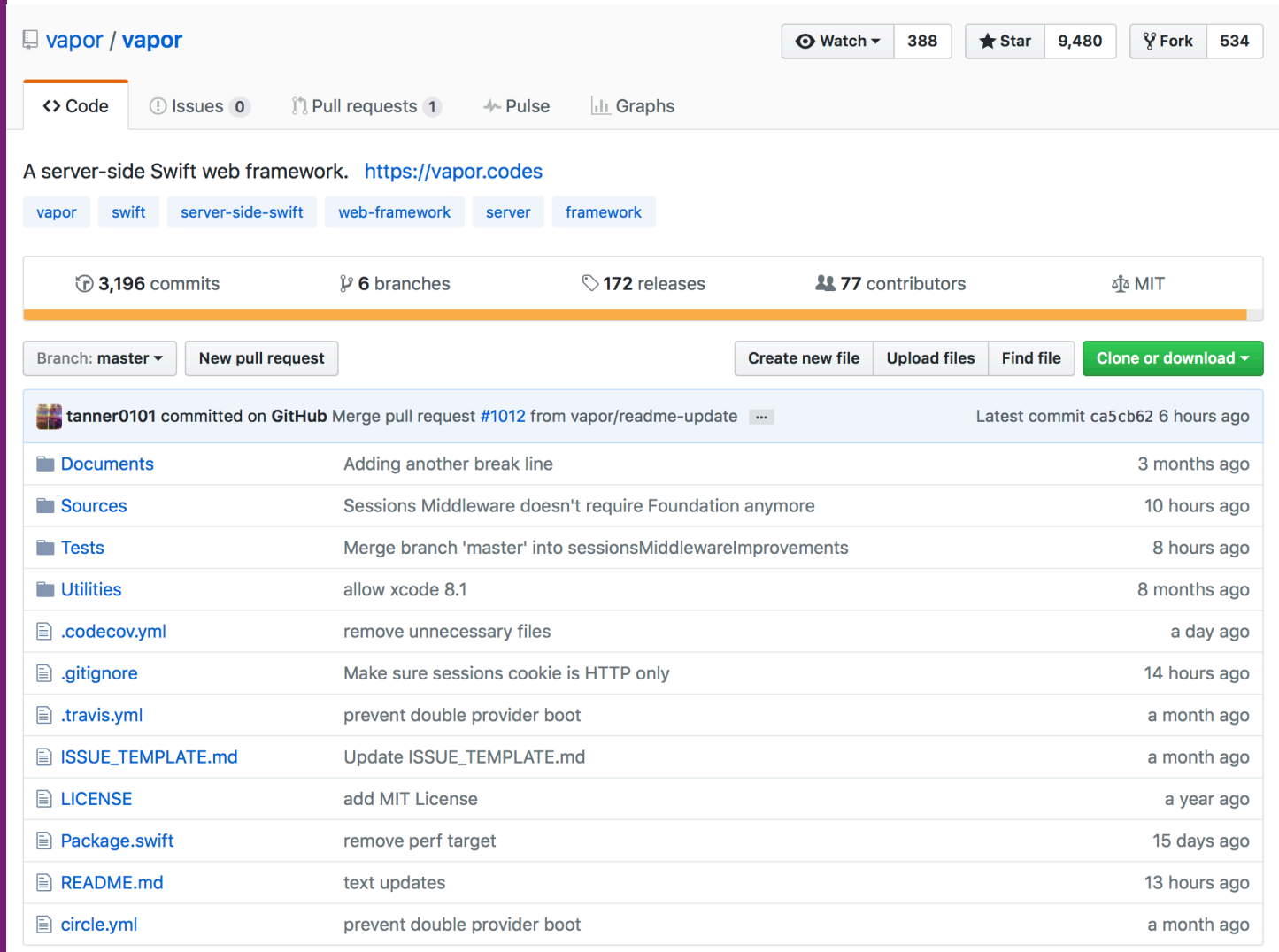
STATE OF SWIFT ON THE SERVER

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

VAPOR

VAPOR

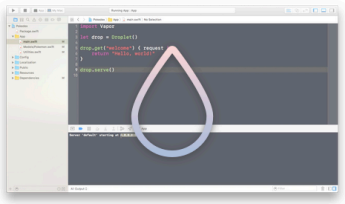
- Web framework and server for Swift
- Works on macOS and Ubuntu



VAPOR

- Great documentation and resources

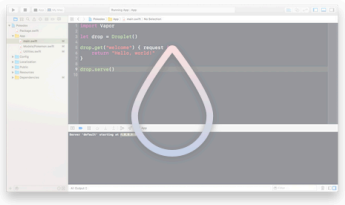
vapor.university



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

VAPOR

VAPOR Docs

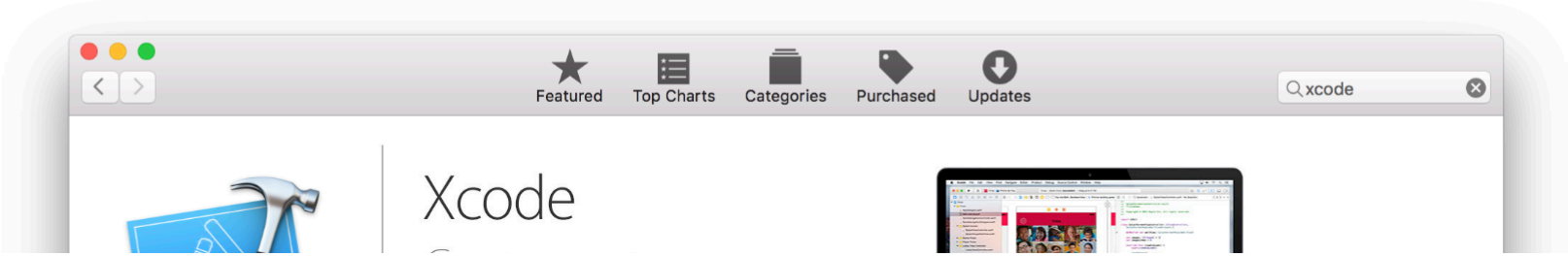
✎ Edit on GitHub

Install Swift 3: macOS

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

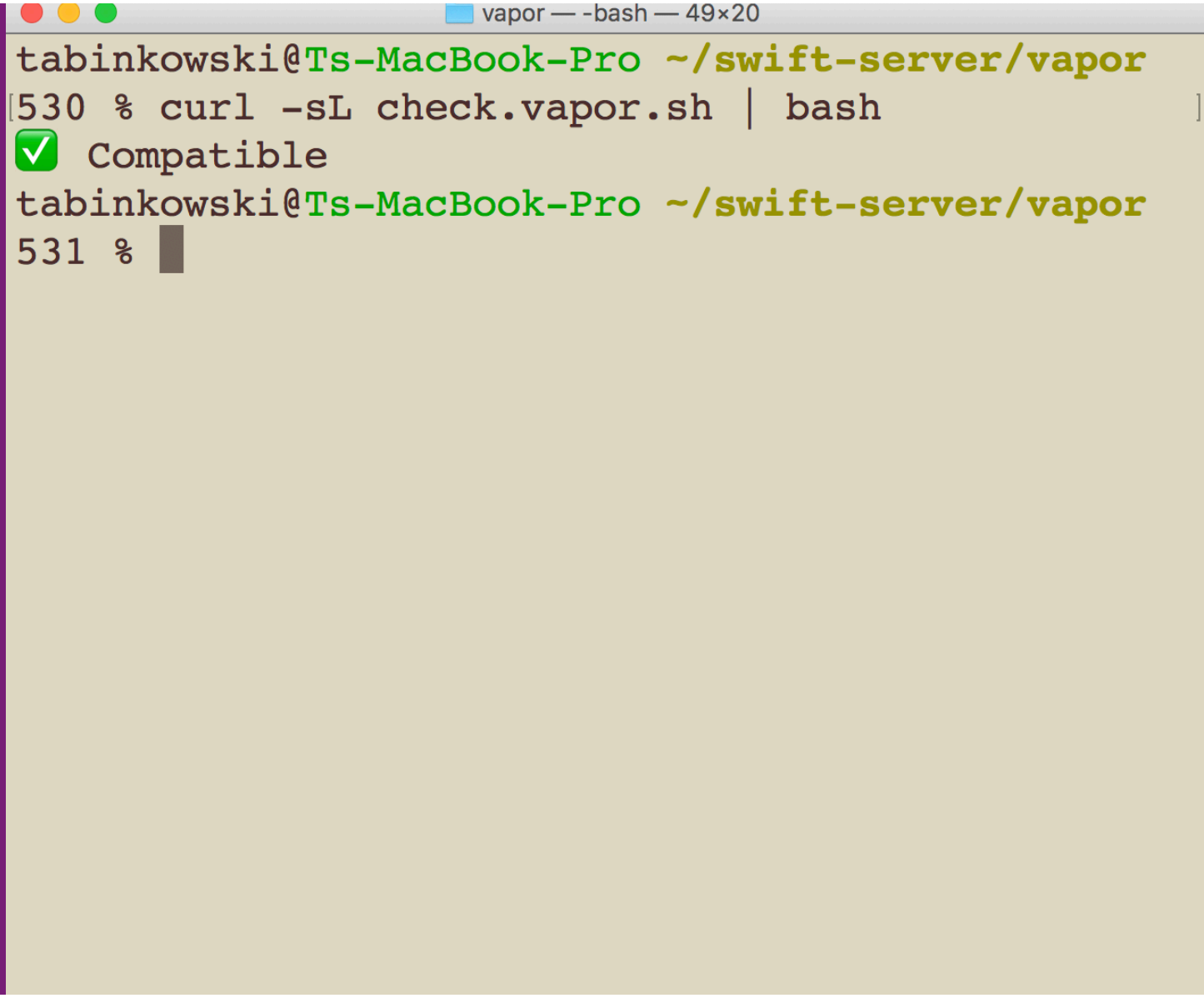
Install Xcode

Install Xcode 8 from the Mac App Store.



VAPOR

- Install Xcode
- Check version for vapor

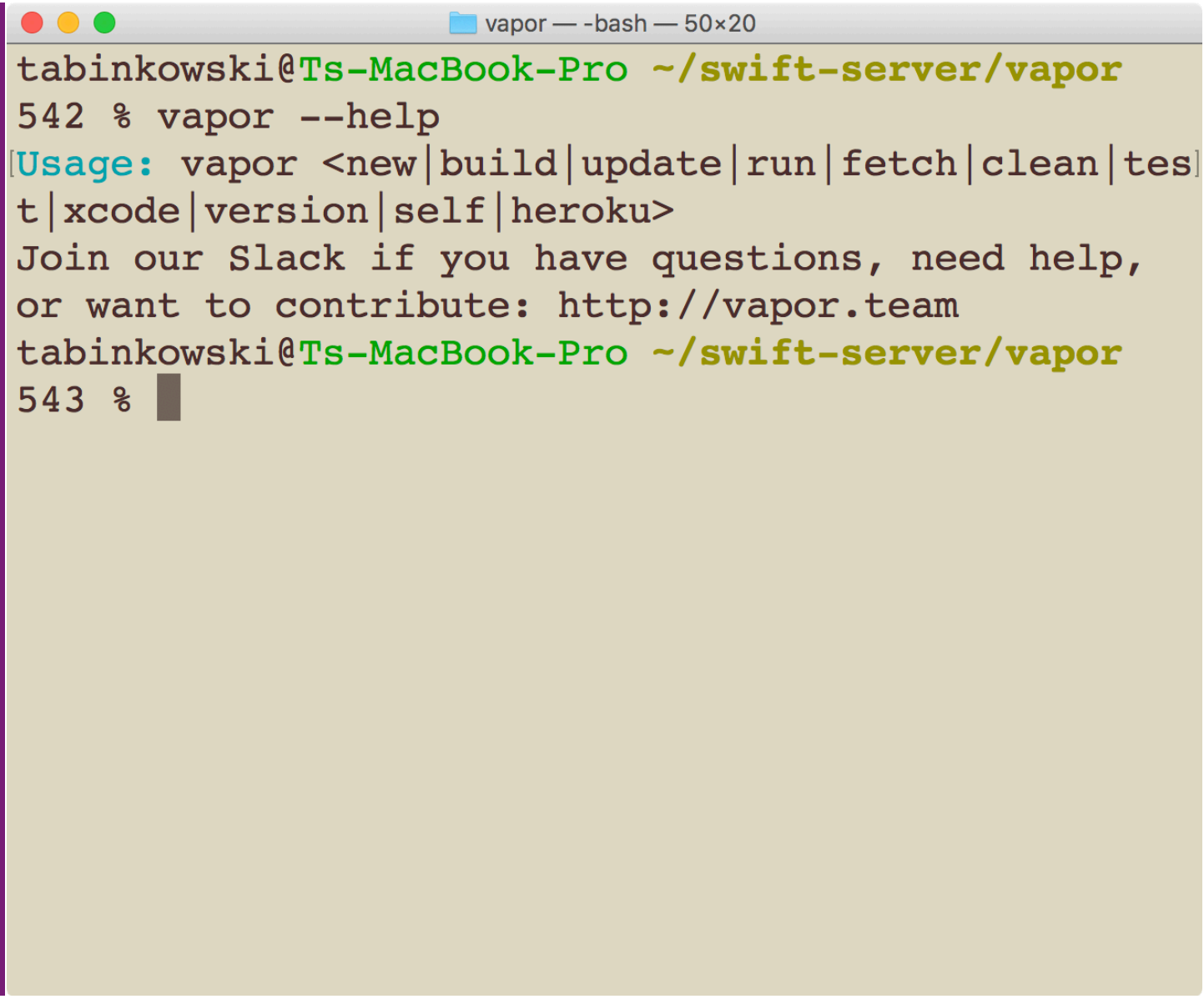


VAPOR

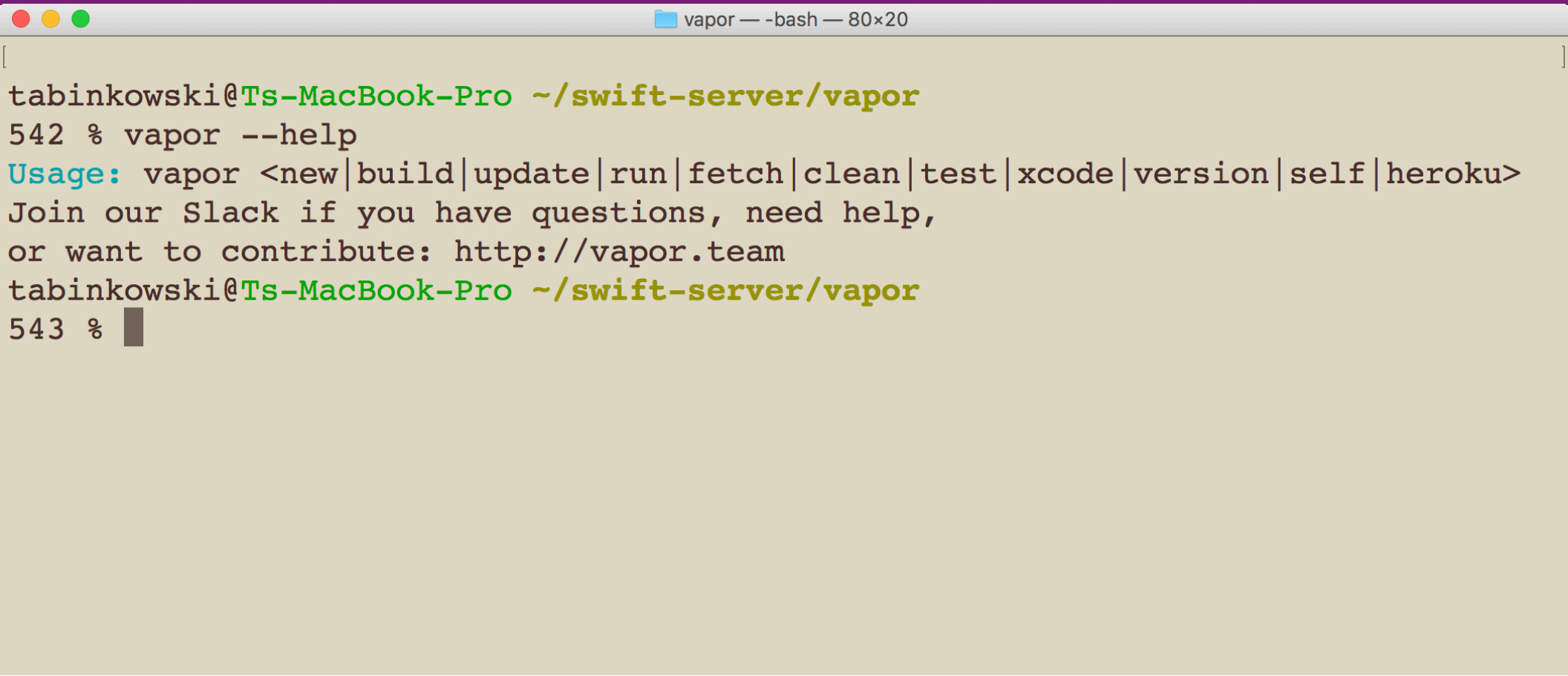
- Toolbox provides command line and shortcuts for common tasks
- Install with homebrew 🍺

VAPOR

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



VAPOR

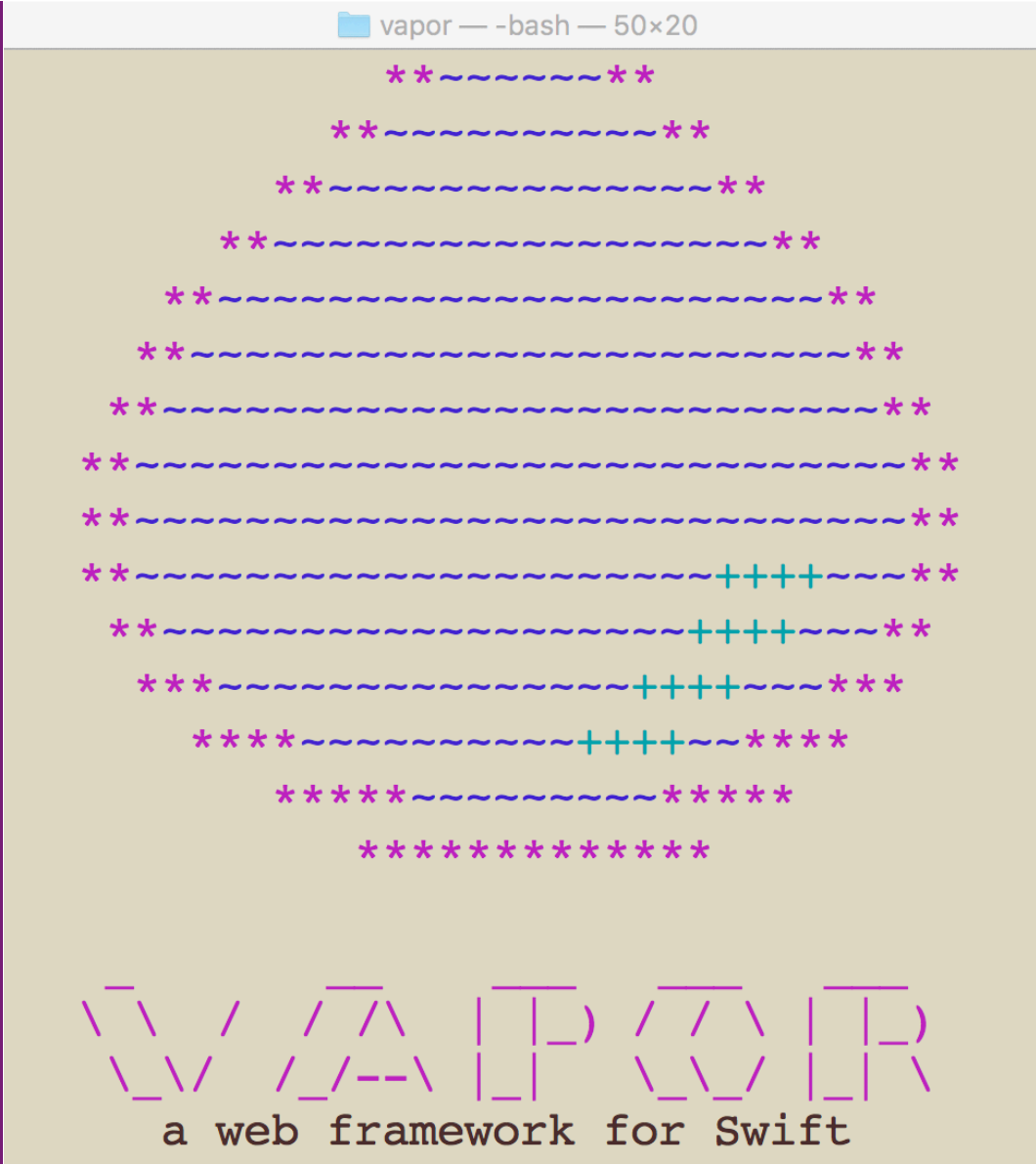

vapor — -bash — 80×20
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor
542 % vapor --help
Usage: vapor <new|build|update|run|fetch|clean|test|xcode|version|self|heroku>
Join our Slack if you have questions, need help,
or want to contribute: http://vapor.team
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor
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>]
```

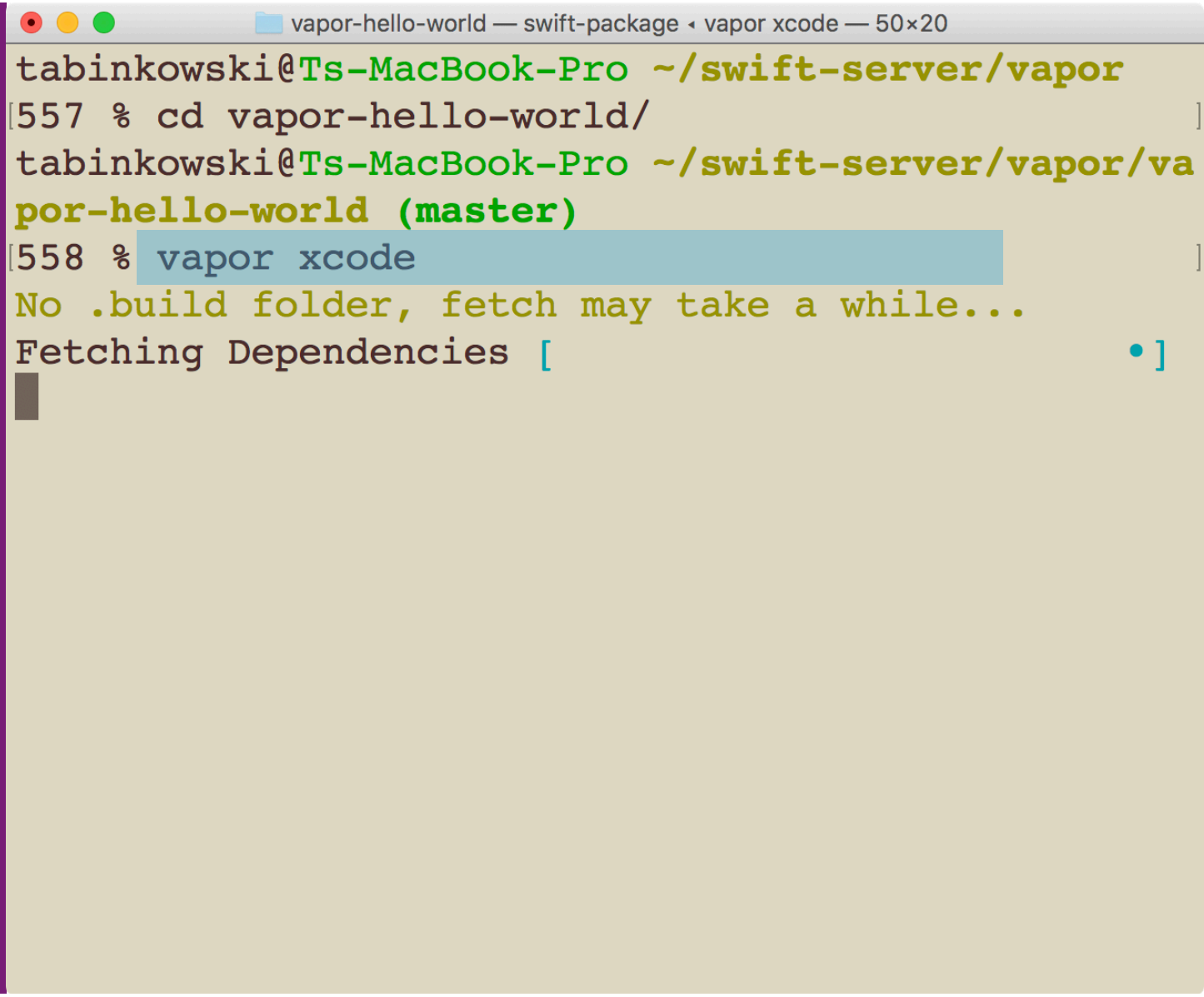
VAPOR

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



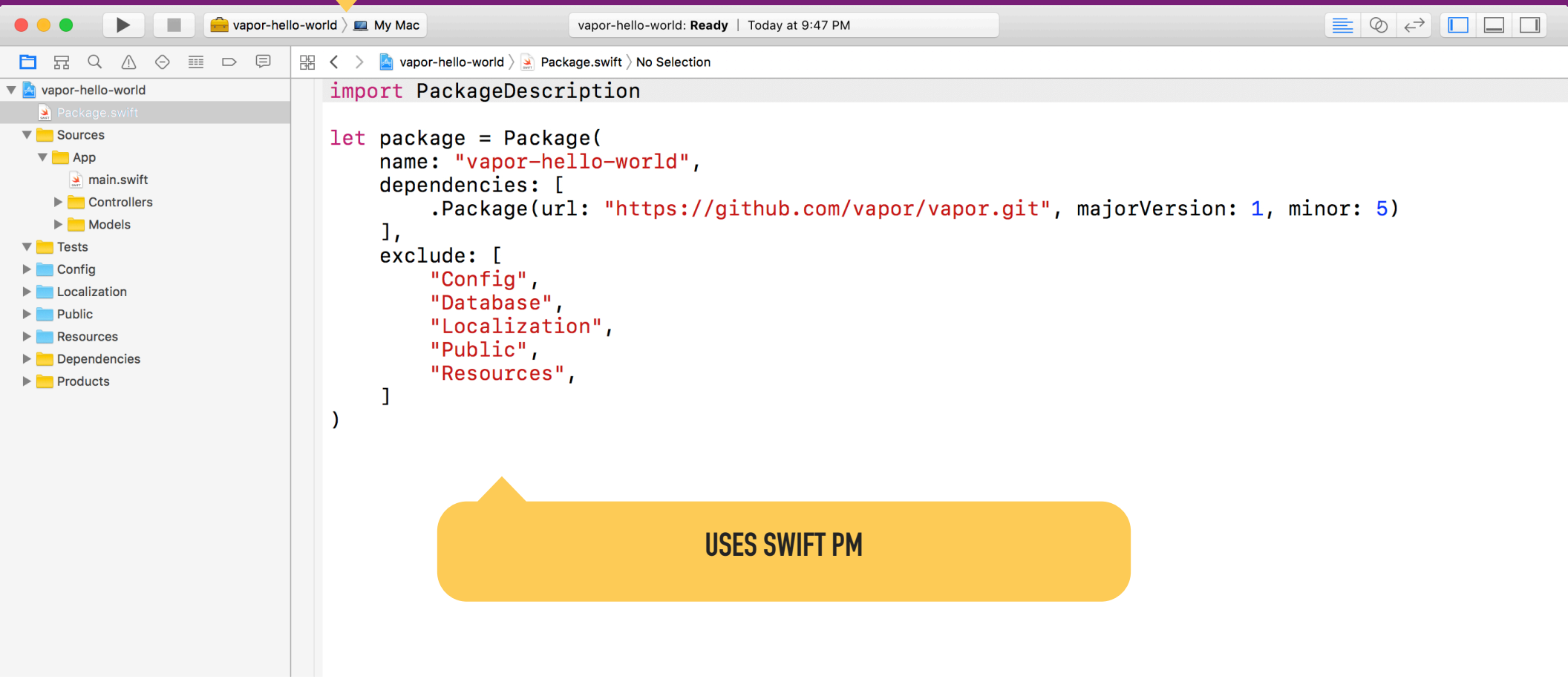
VAPOR

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



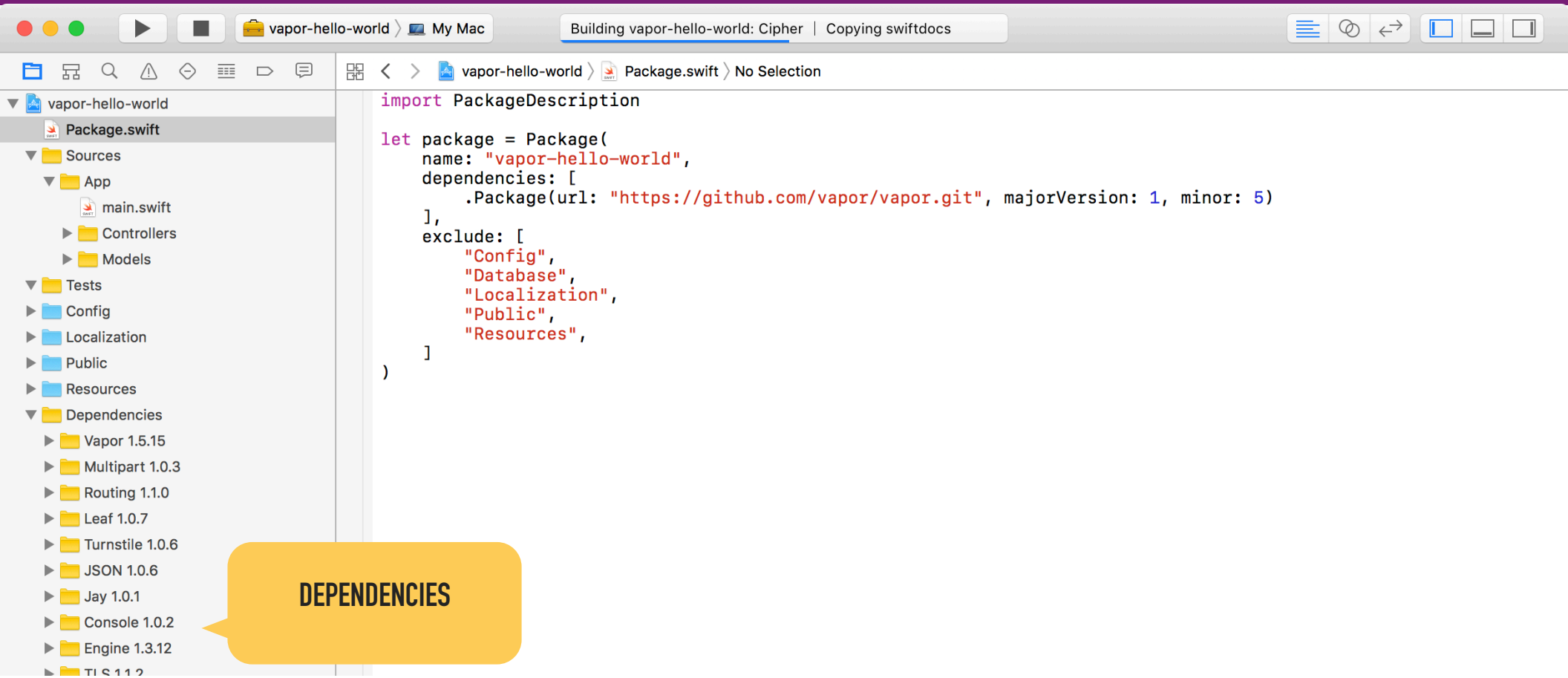
VAPOR

BUILDS A FRAMEWORK AND APP



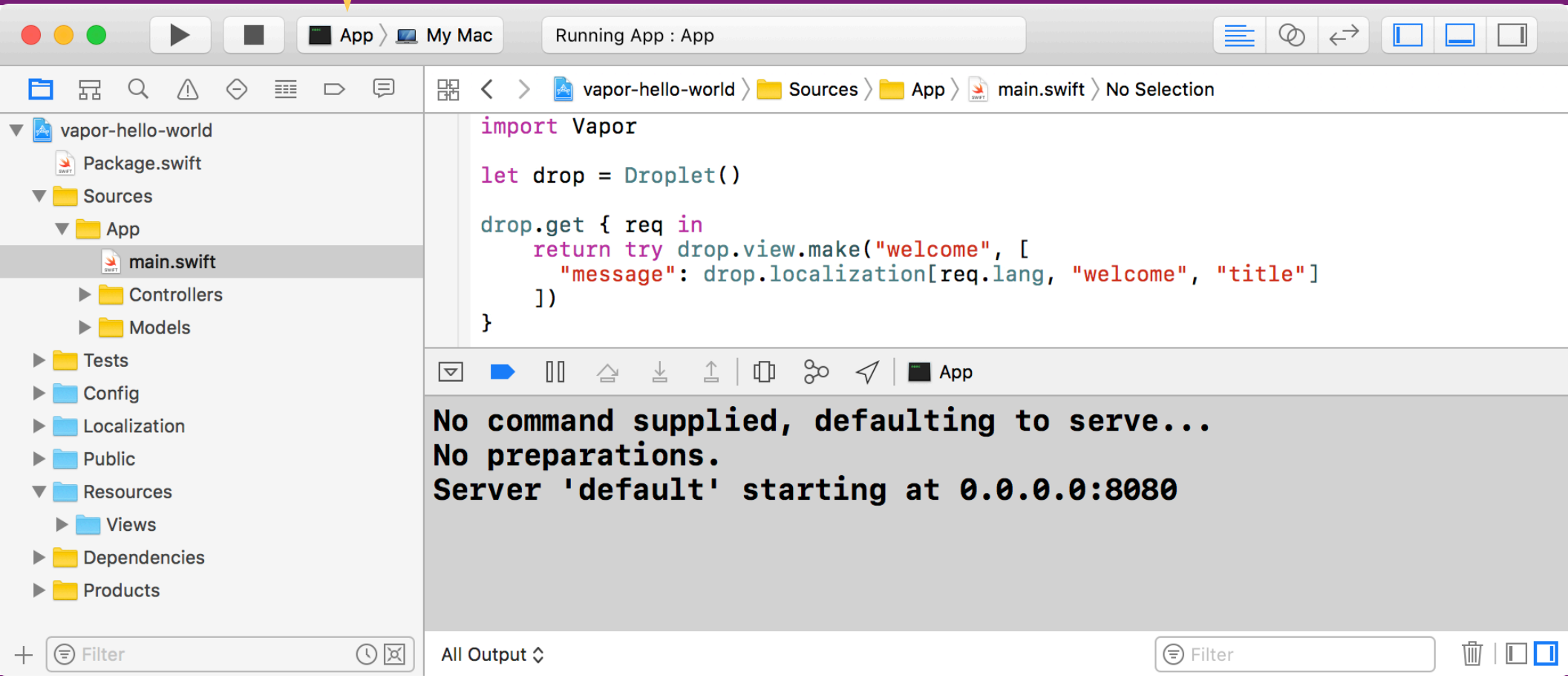
USES SWIFT PM

VAPOR


vapor-hello-world 〉 My Mac
Building vapor-hello-world: Cipher | Copying swiftdocs
vapor-hello-world 〉 Package.swift 〉 No Selection
vapor-hello-world
Package.swift
Sources
App
main.swift
Controllers
Models
Tests
Config
Localization
Public
Resources
Dependencies
Vapor 1.5.15
Multipart 1.0.3
Routing 1.1.0
Leaf 1.0.7
Turnstile 1.0.6
JSON 1.0.6
Jay 1.0.1
Console 1.0.2
Engine 1.3.12
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",
]
)
DEPENDENCIES

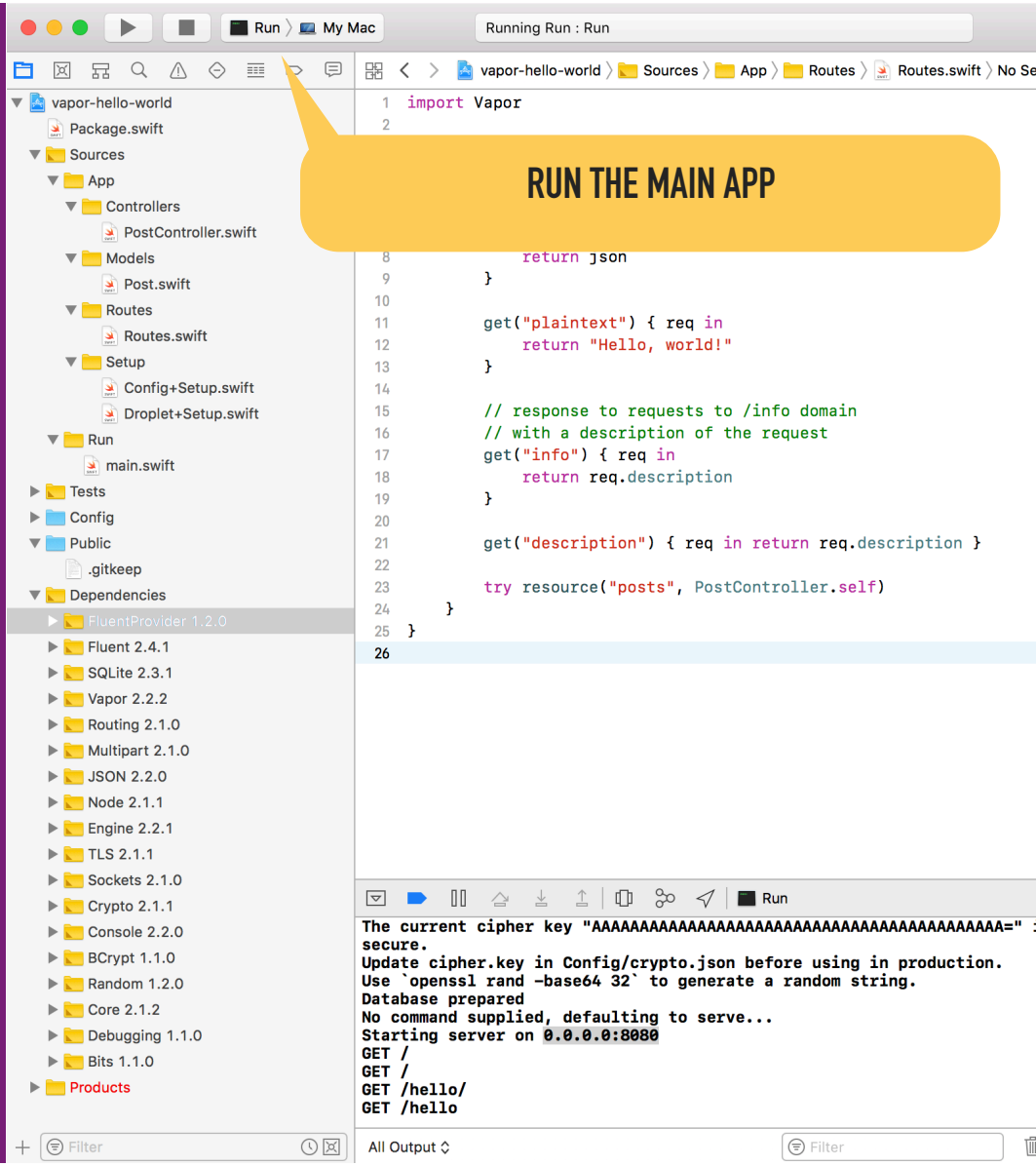
VAPOR

RUN APP



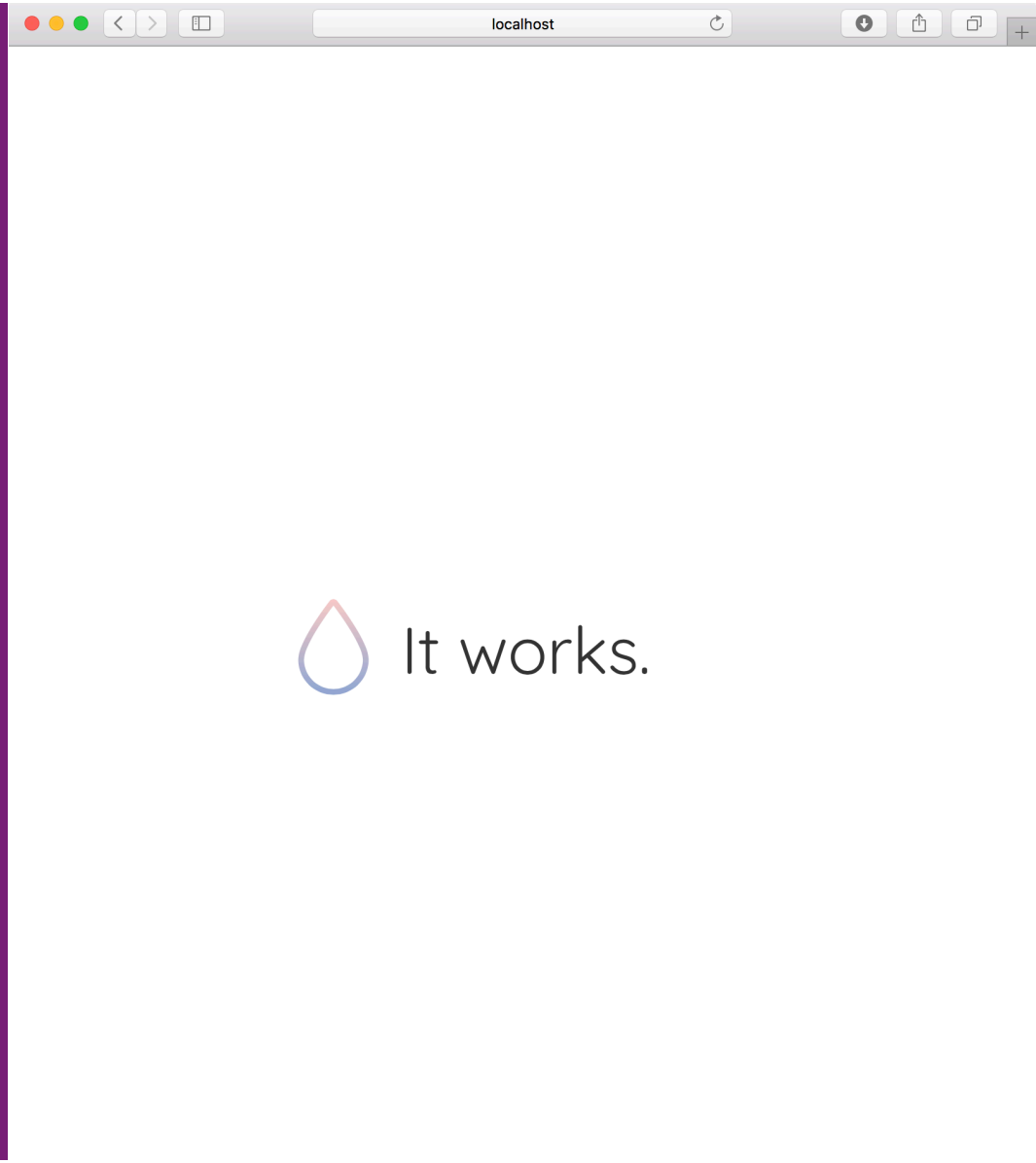
VAPOR

- Builds all the package dependencies and links them



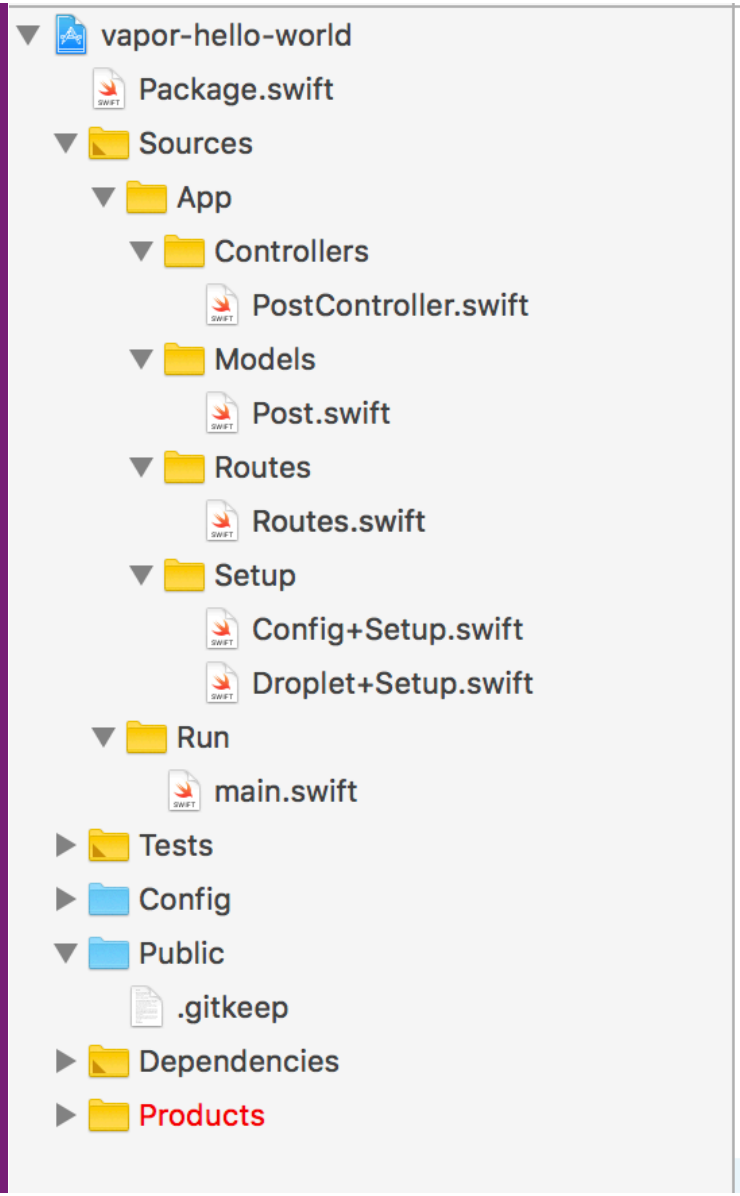
VAPOR

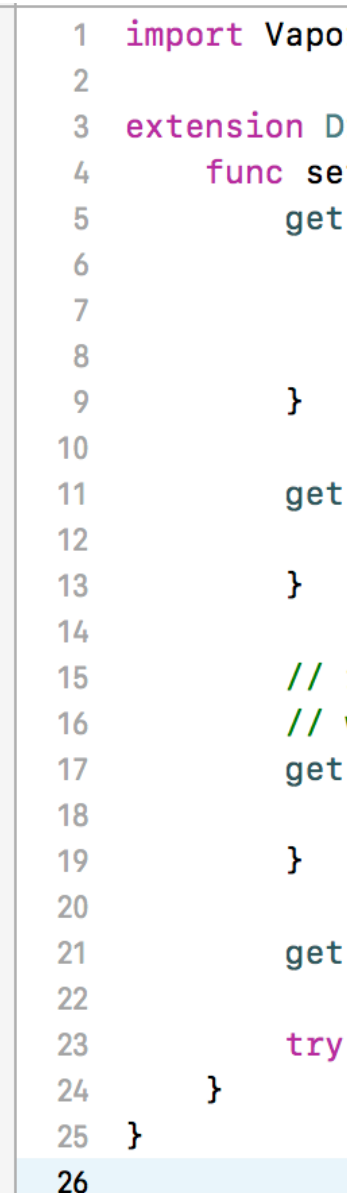
- Server runs on http://localhost:8080



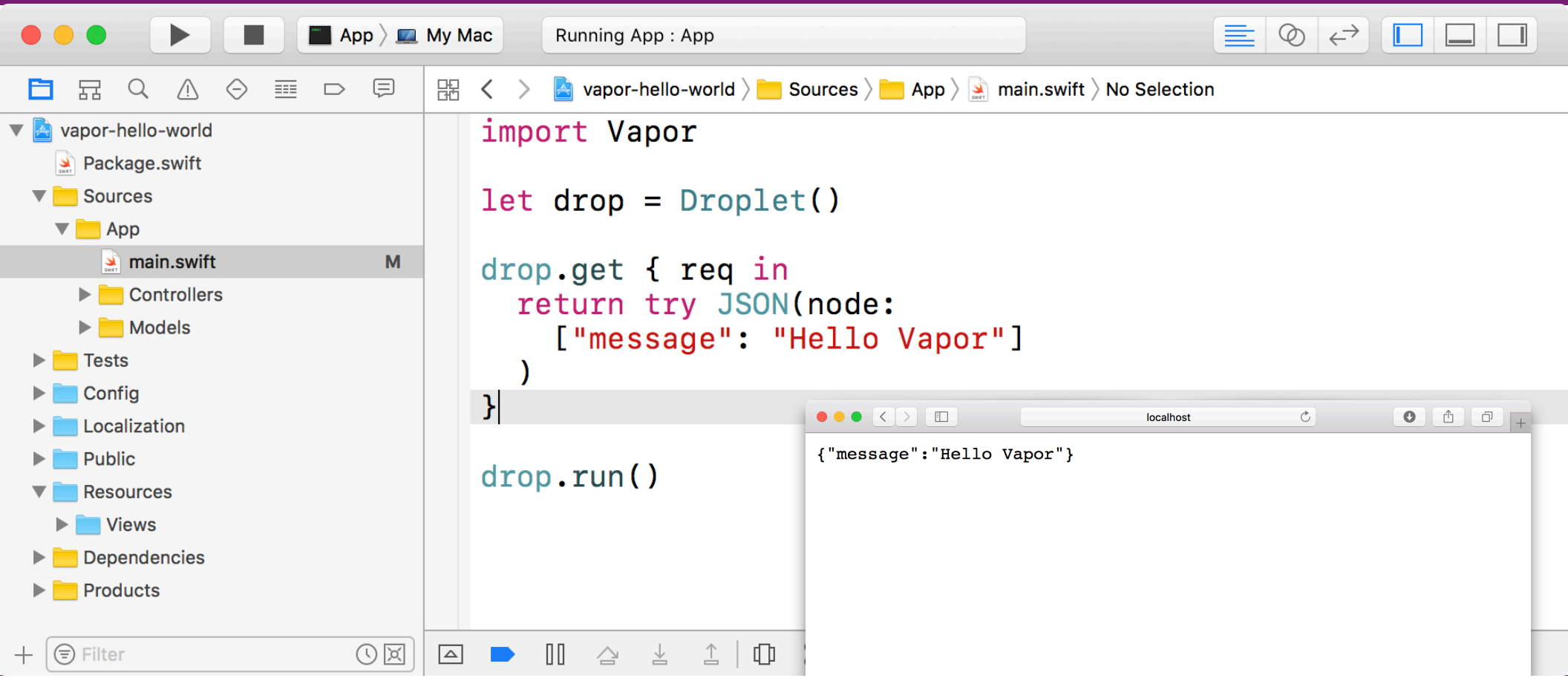
VAPOR

- File structure is designed to make testing easier 🤷‍♀️





VAPOR

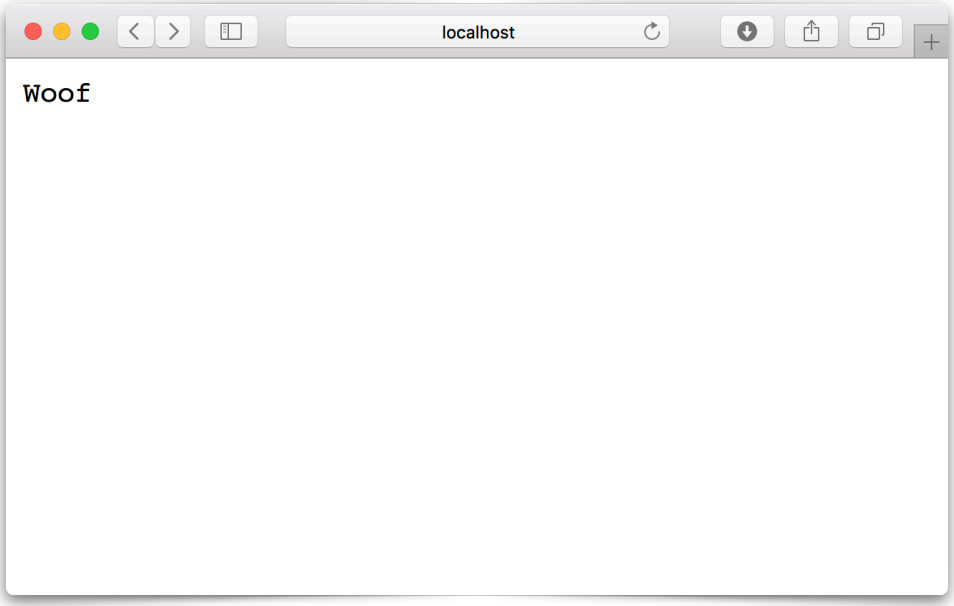

App
My Mac
Running App : App
vapor-hello-world
Sources
App
main.swift
No Selection
vapor-hello-world
Package.swift
Sources
App
main.swift M
Controllers
Models
Tests
Config
Localization
Public
Resources
Views
Dependencies
Products
Filter
import Vapor
let drop = Droplet()
drop.get { req in
return try JSON(node:
["message": "Hello Vapor"]
)
}
drop.run()
localhost
{"message":"Hello Vapor"}

VAPOR

LOCALHOST:8080/DOG

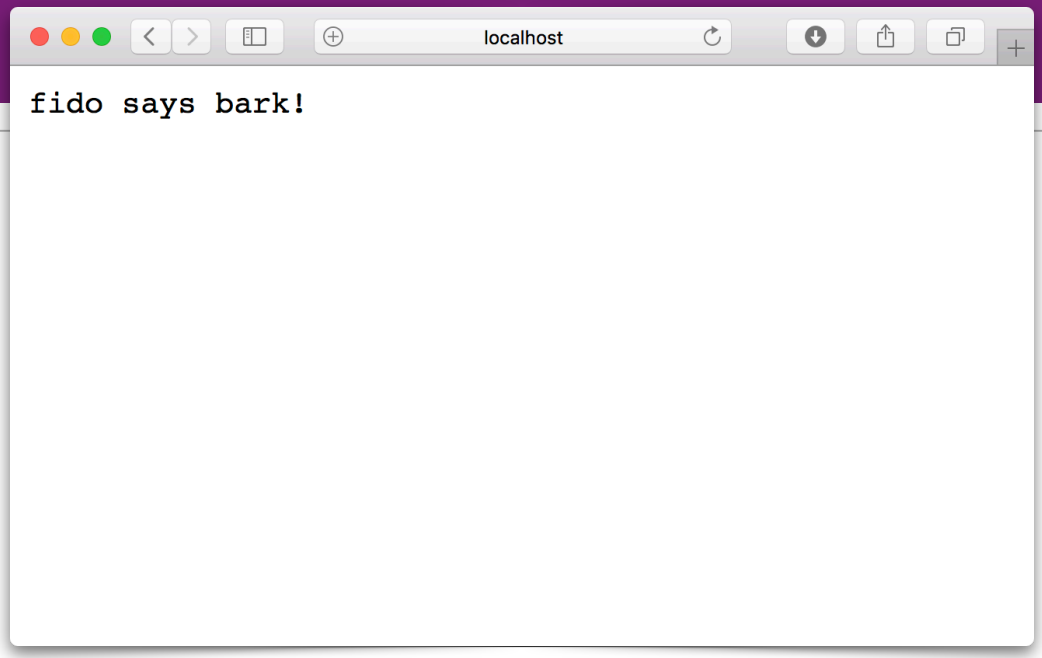
vapor-hello-world 〉 Sources 〉 App 〉 main.swift 〉 No Selection

```
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

LOCALHOST:8080/DOG/SPEAK/FIDO/

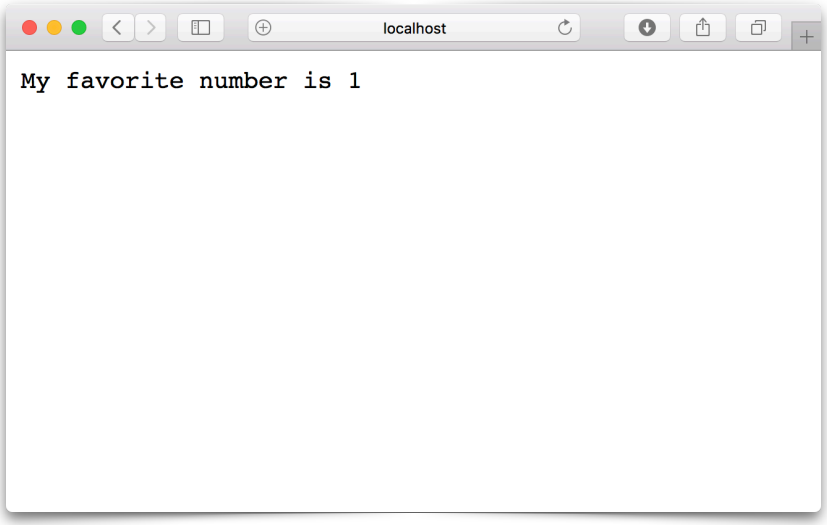


vapor-hello-world / Sources / App / main.swift / No Selection

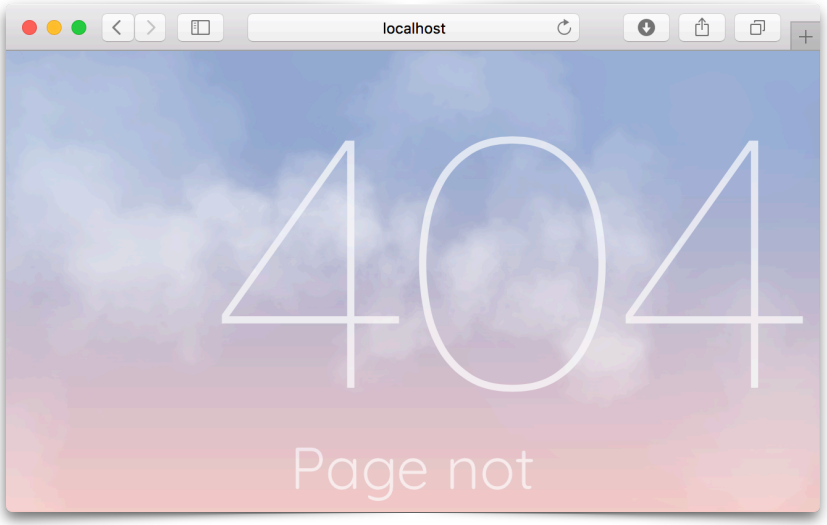
```
drop.get { req in
  return try JSON(node:
    ["message": "Hello Vapor"]
  )
}

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/

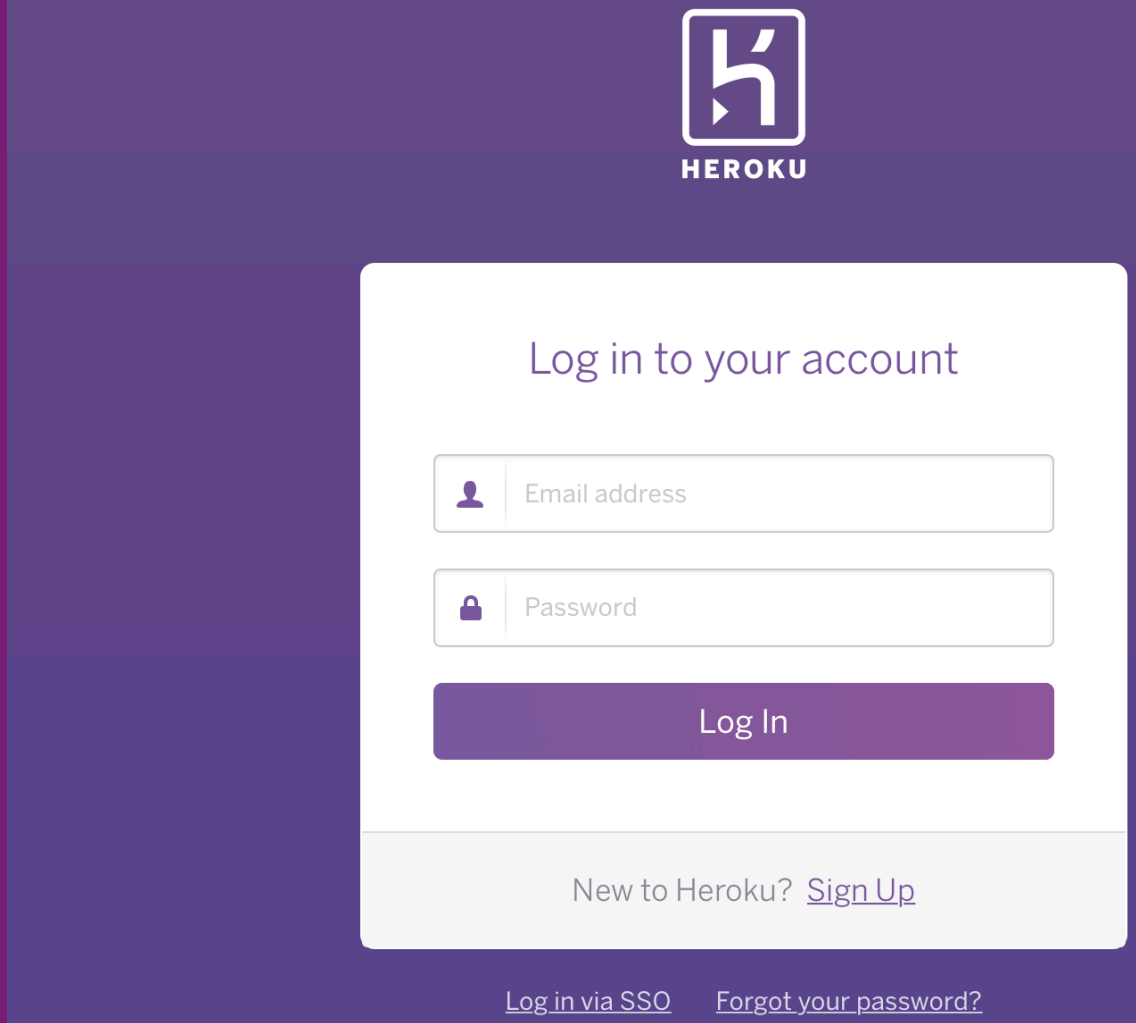



```
drop.get("dog","speak",Int.self) { req, number in
   return "My favorite number is \(number)"
}
```

DEPLOY VAPOR APP TO HEROKU

DEPLOY VAPOR APP TO HEROKU

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



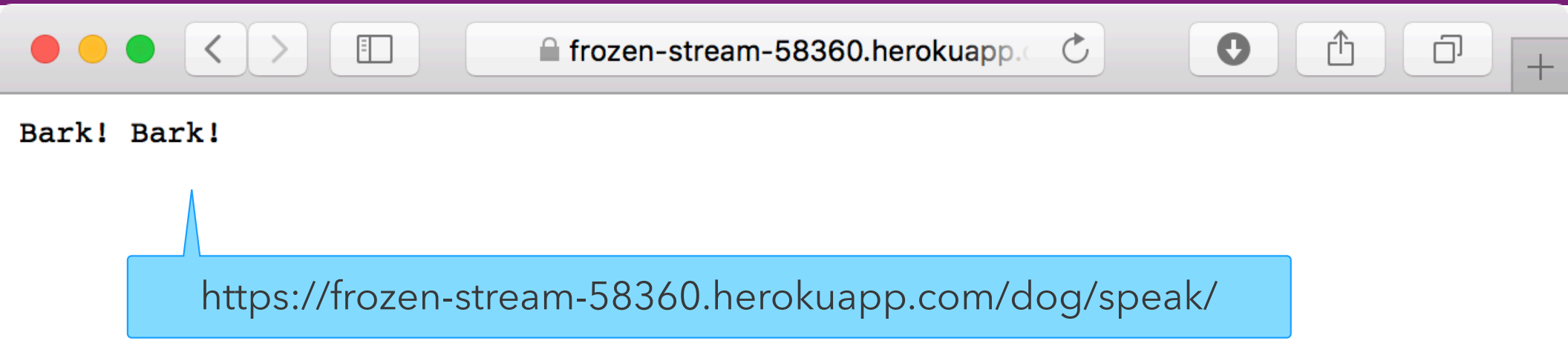
DEPLOY VAPOR APP TO HEROKU

```
# 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
```

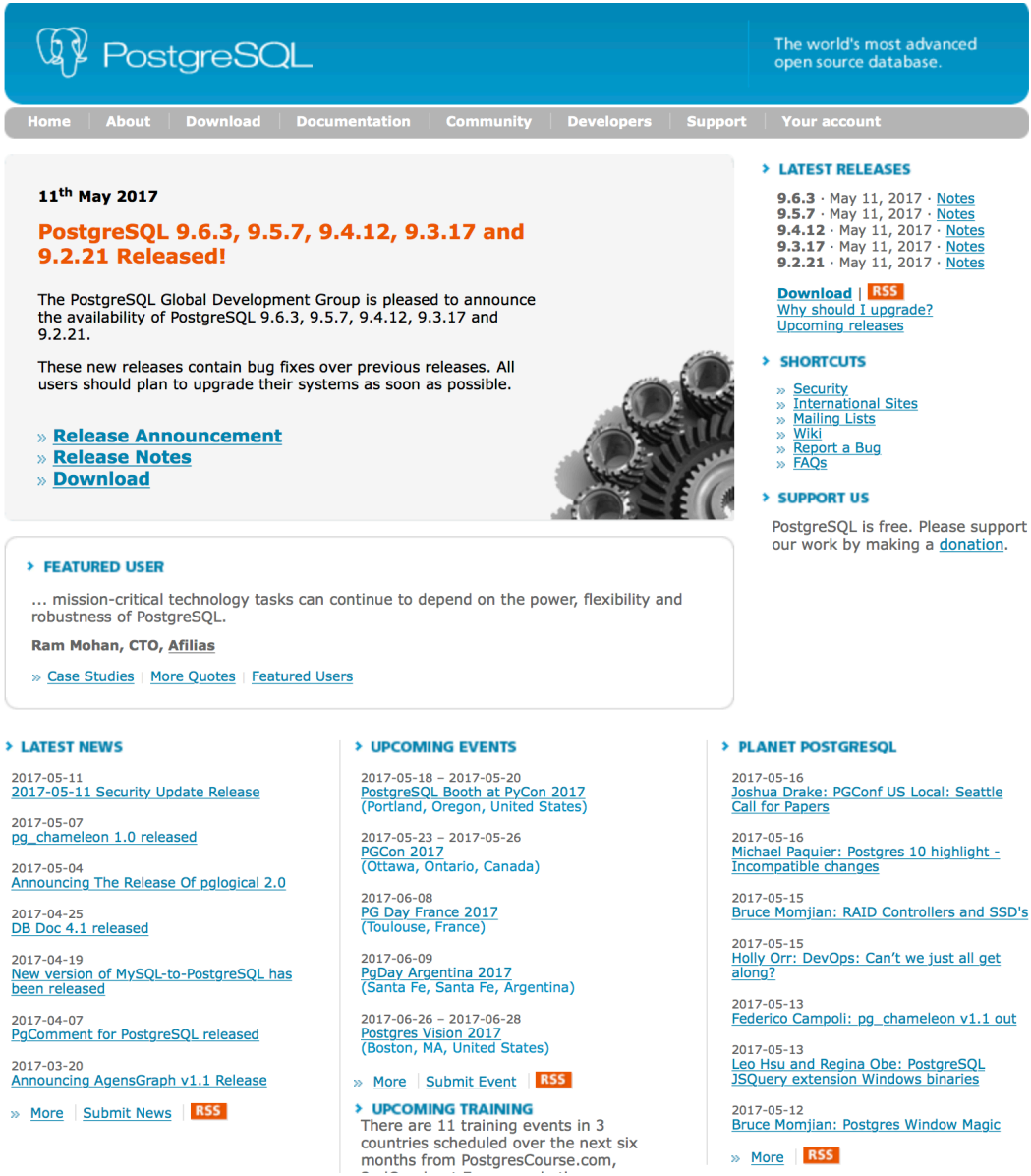
DEPLOY VAPOR APP TO HEROKU



VAPOR WITH POSTGRESQL

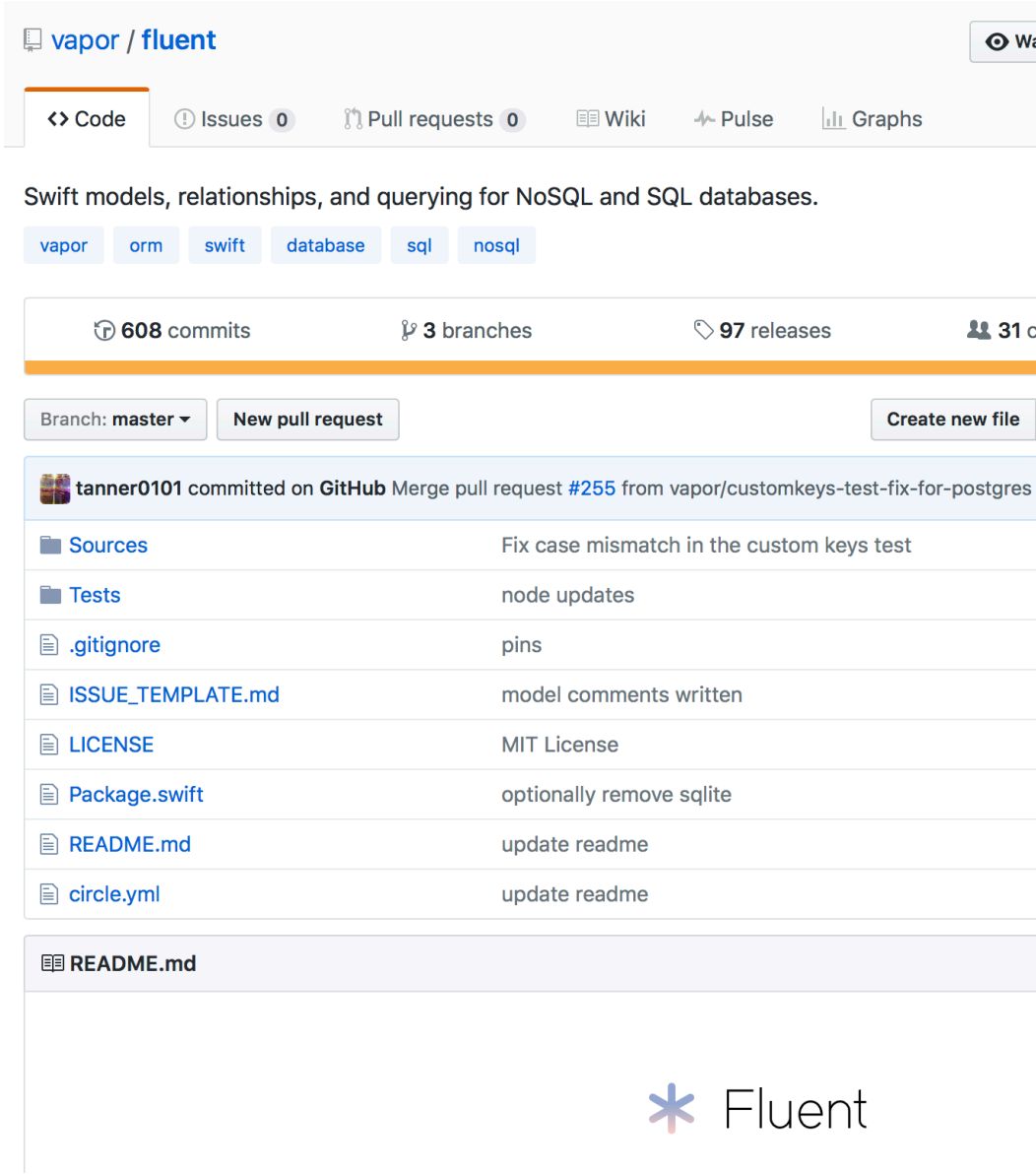
VAPOR WITH POSTGRESQL

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



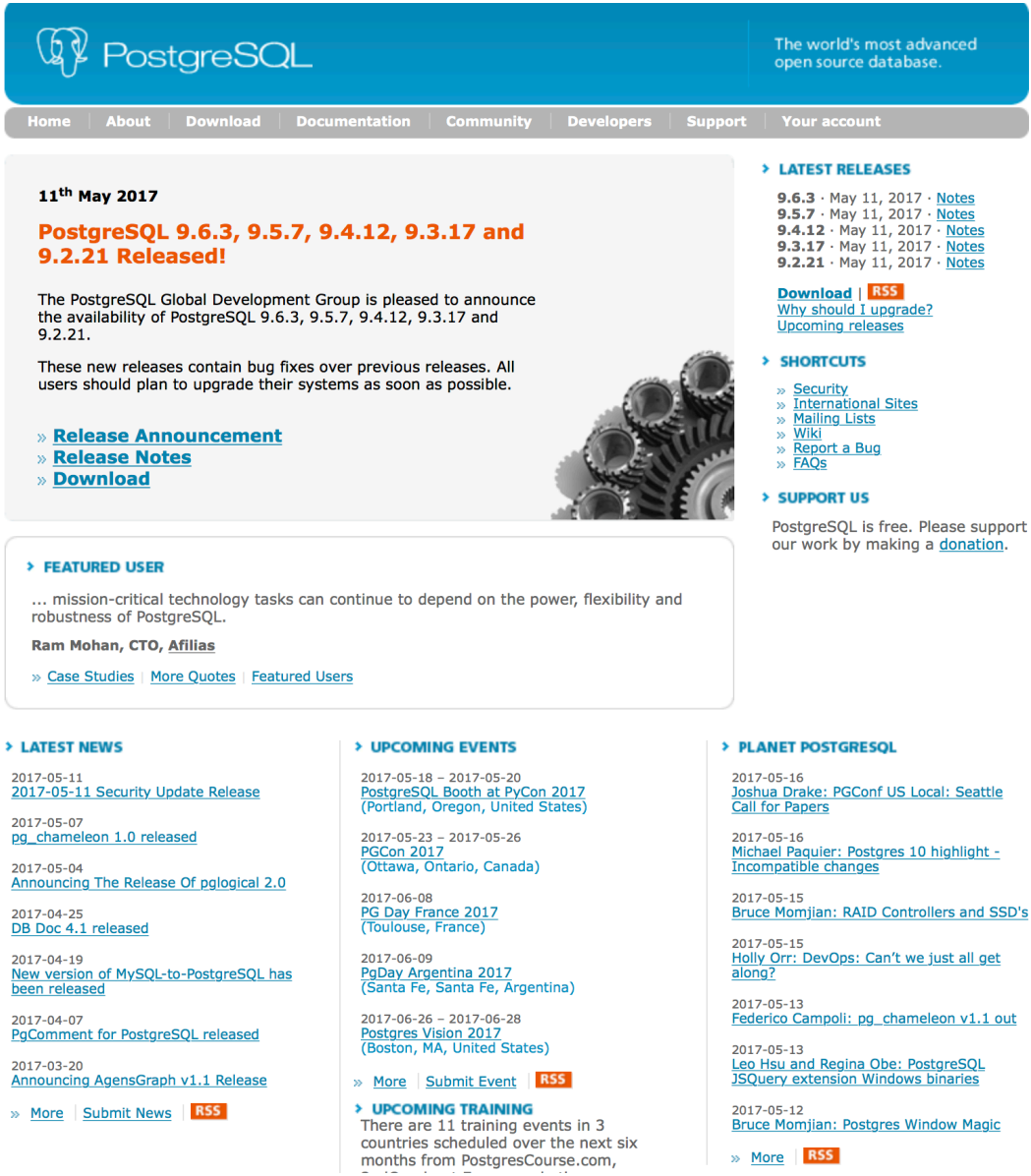
VAPOR WITH POSTGRESQL

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



VAPOR WITH POSTGRESQL

- Import package
- Configure droplet
- Add configuration file



VAPOR WITH POSTGRESQL

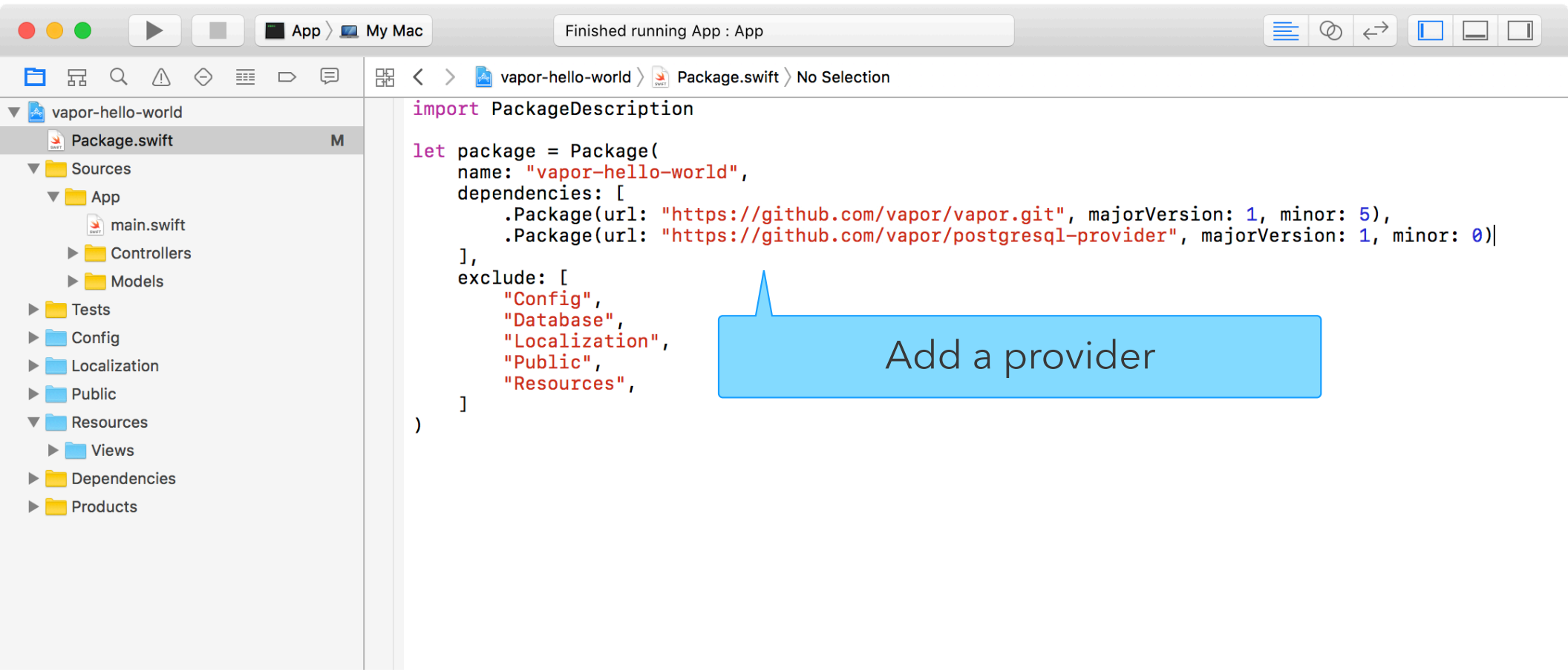
```
# Install postgres
brew install postgres

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

# Create a database
createdb dogs
```

VAPOR WITH POSTGRESQL

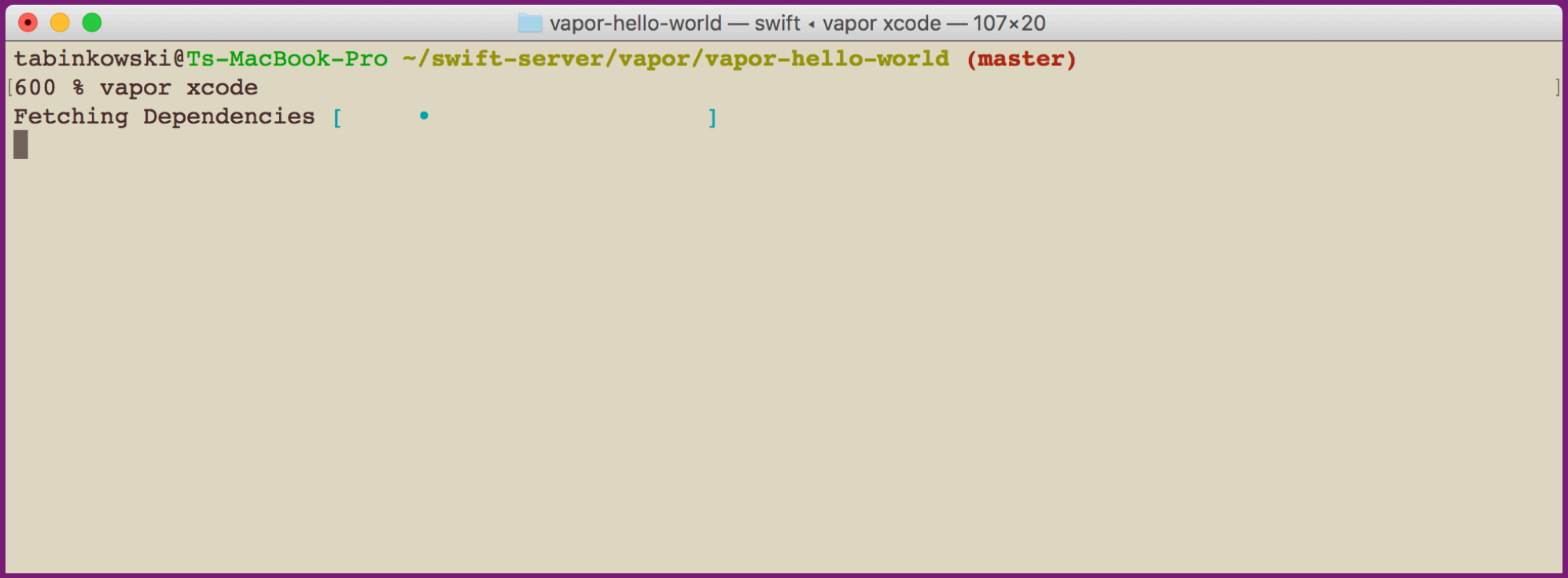
```
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
[592 % psql
psql (9.6.3)
Type "help" for help.

[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



VAPOR WITH POSTGRESQL



- Update the project and rebuild your framework in Xcode

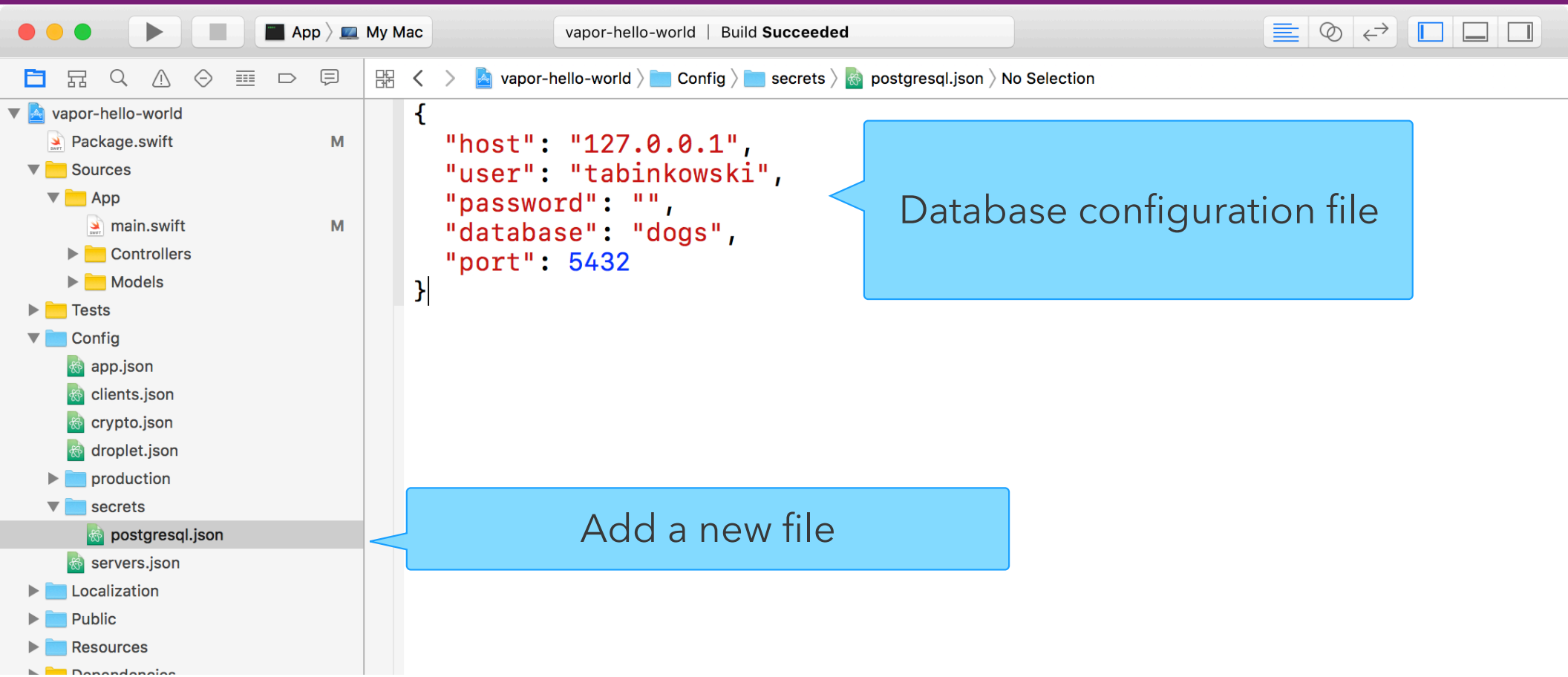
VAPOR WITH POSTGRESQL

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

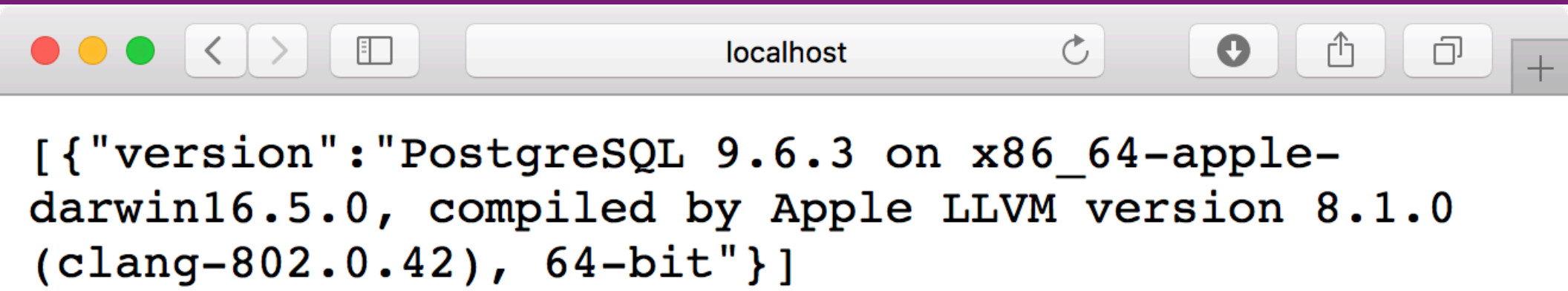


VAPOR WITH POSTGRESQL

```
// 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"
  }
}
```

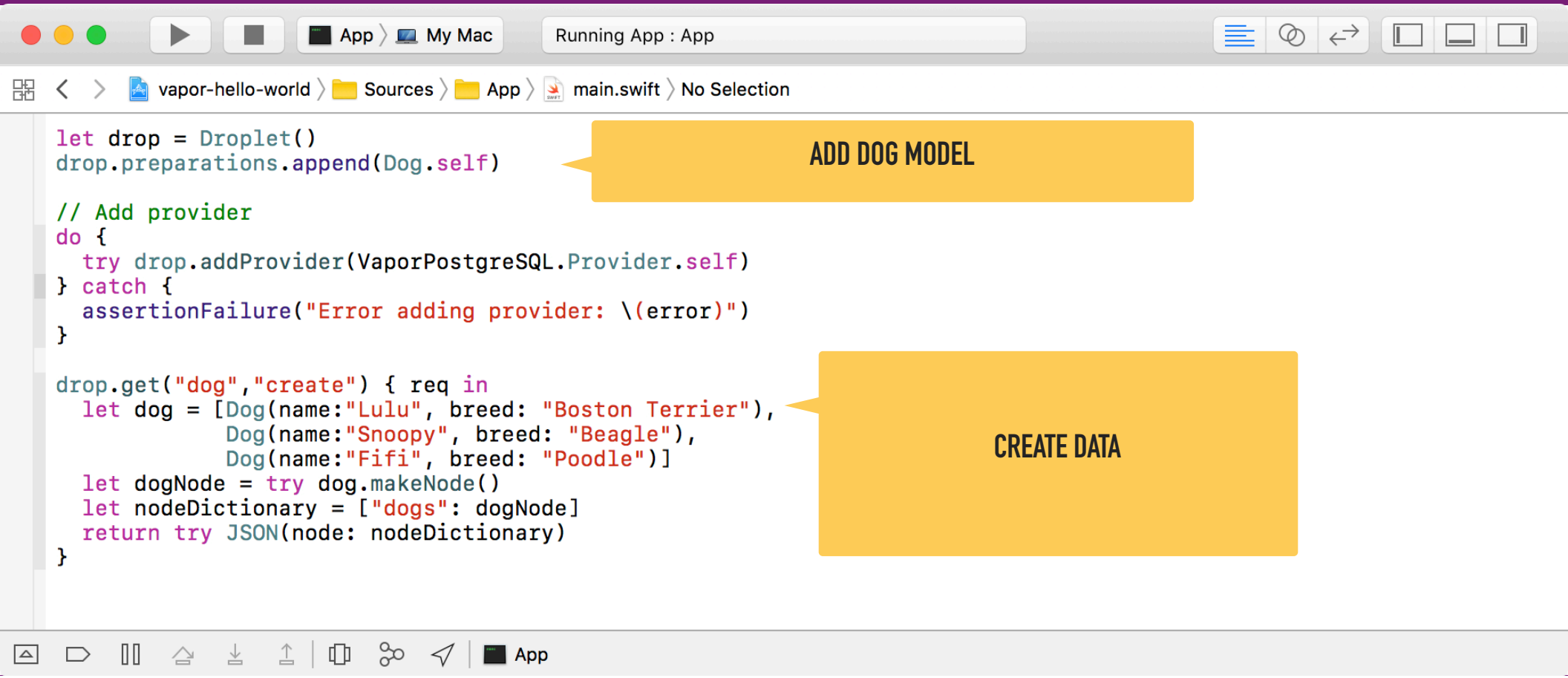
Print out database version

VAPOR WITH POSTGRESQL

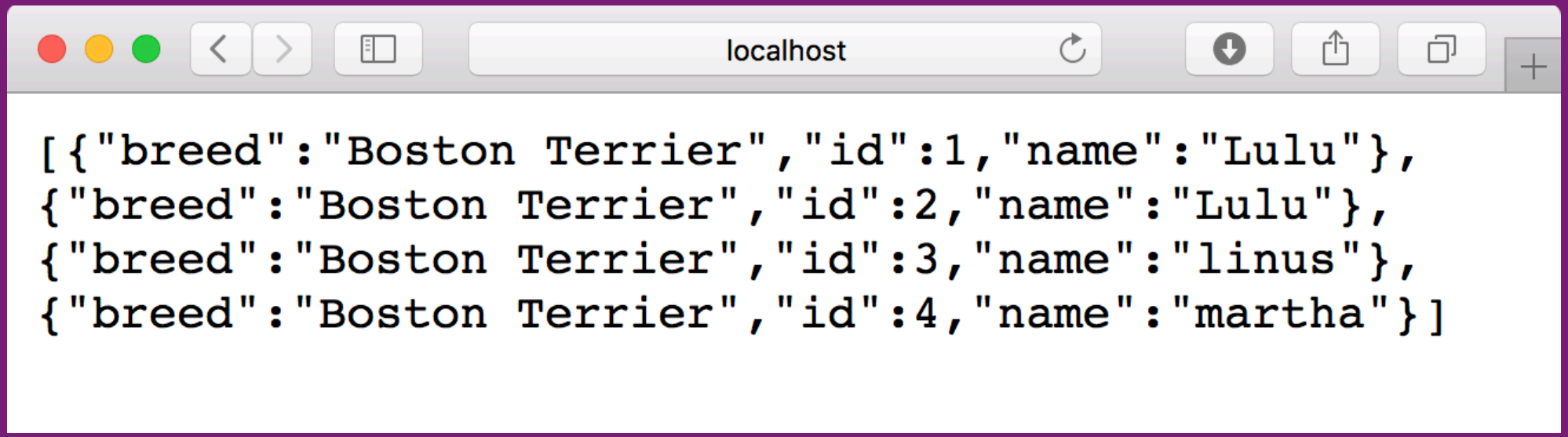


IT'S BORING, BUT IT WORKS

VAPOR WITH POSTGRESQL



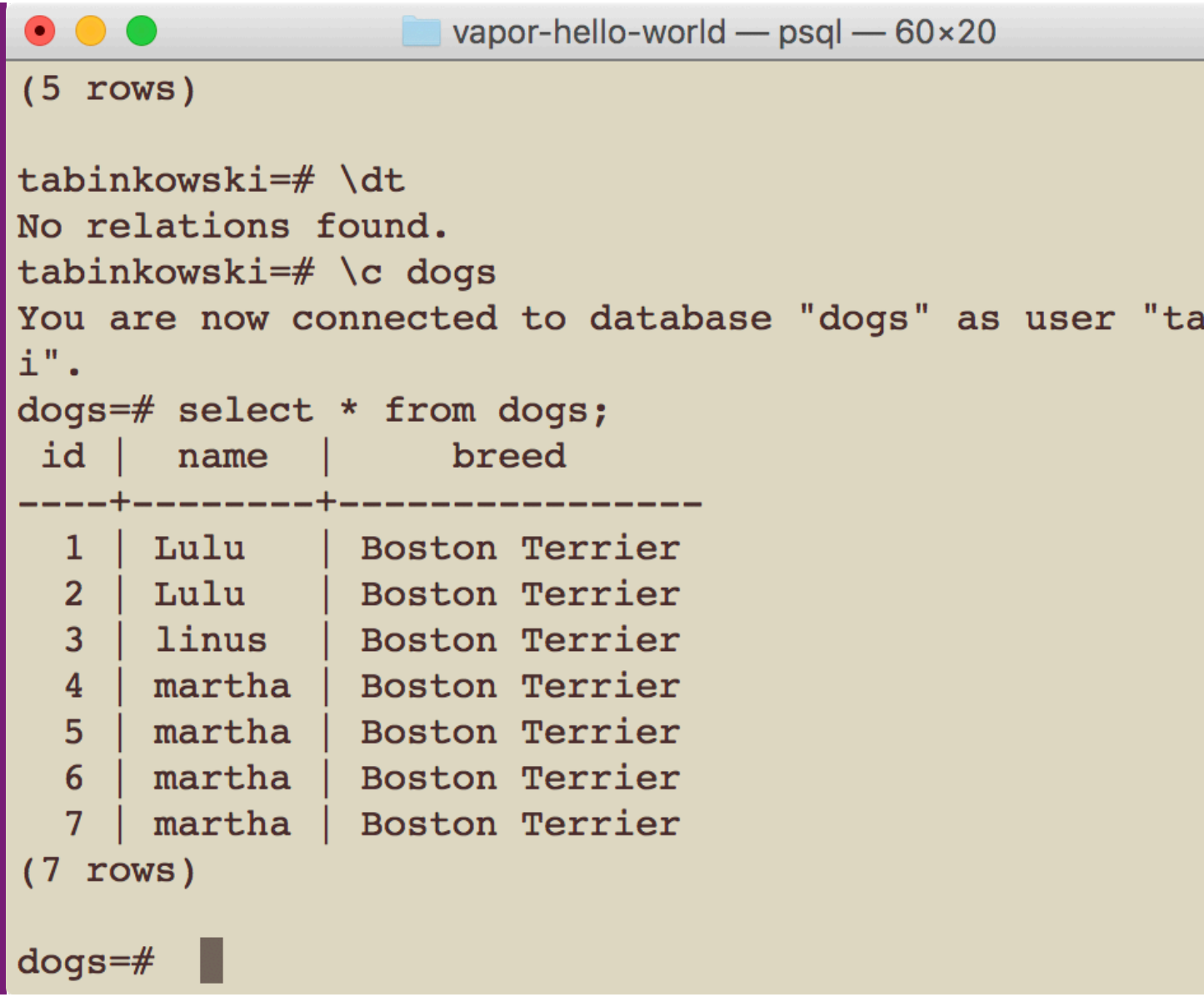
VAPOR WITH POSTGRESQL




```
// 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



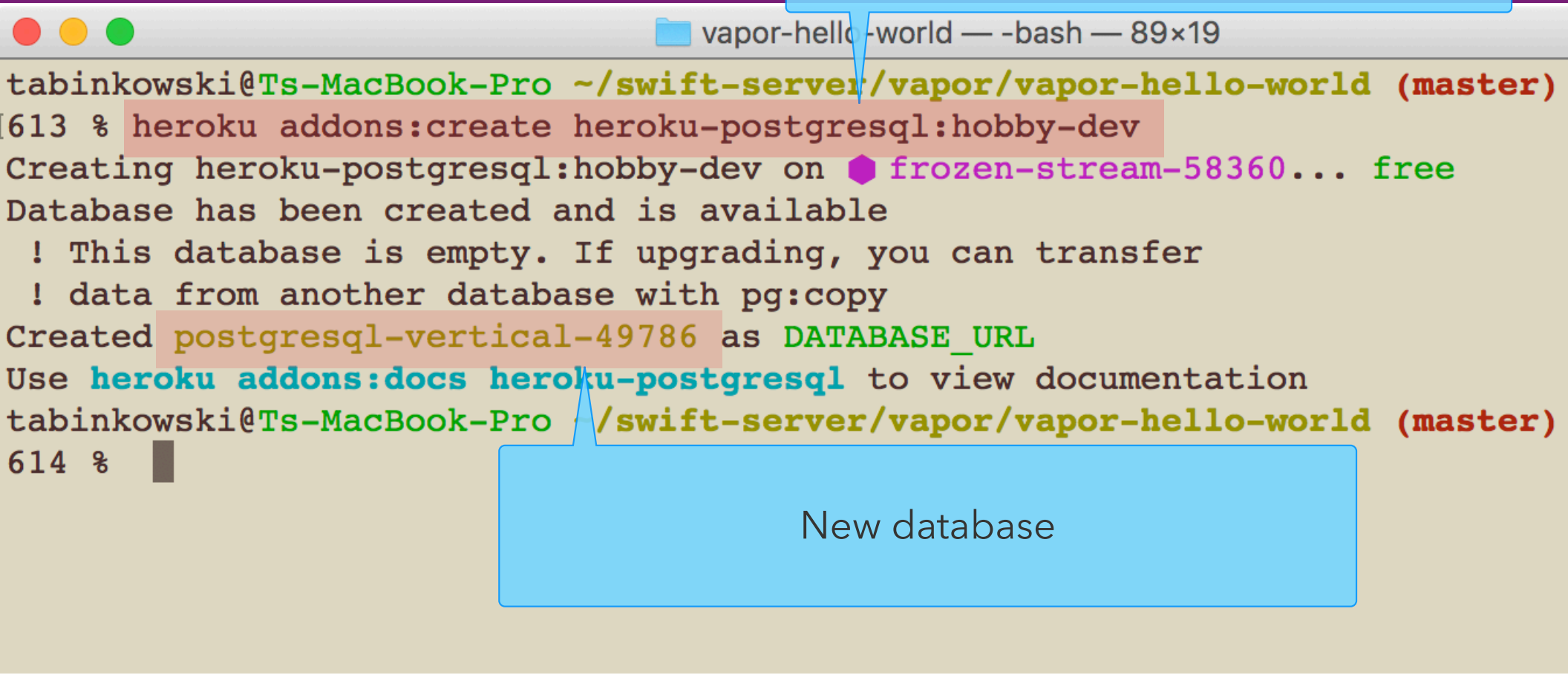
DEPLOY TO HEROKU WITH POSTGRESQL

DEPLOY TO HEROKU WITH POSTGRESQL

- Set up postgresql on heroku
  - Built-in, just need to add it
- Tell Postgresql where the database is locates
  - 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

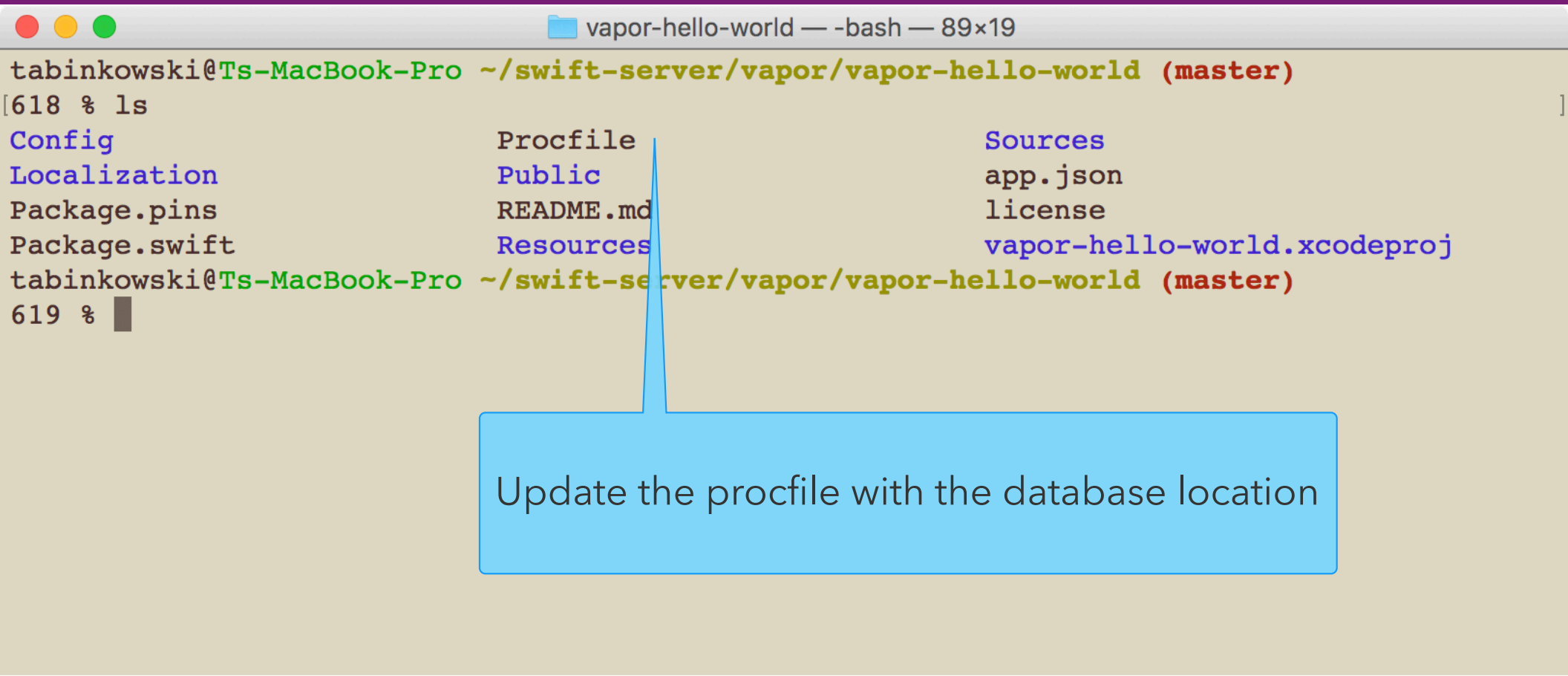


DEPLOY TO HEROKU WITH POSTGRESQL

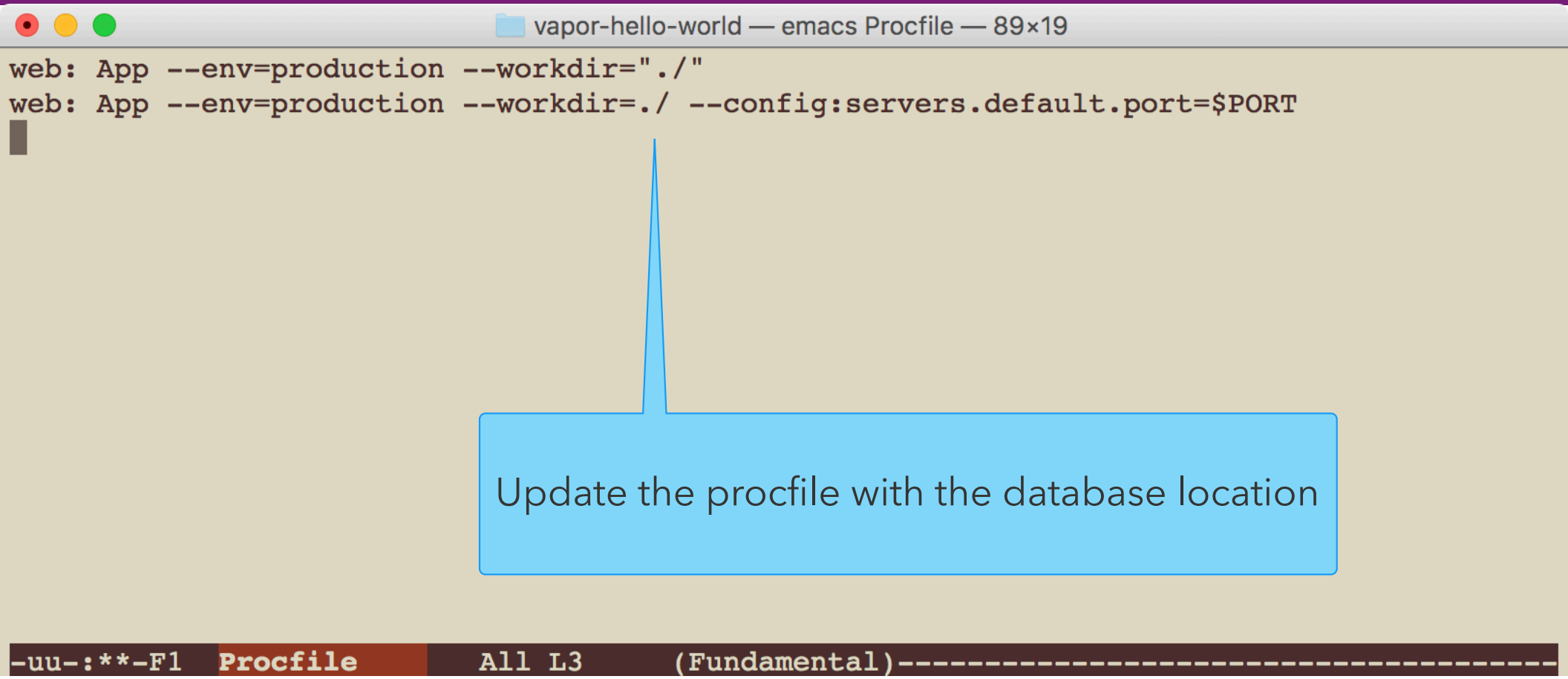
```
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
[613 % heroku addons:create heroku-postgresql:hobby-dev
Creating heroku-postgresql:hobby-dev on ⬢ frozen-stream-58360... free
Database has been created and is available
 ! This database is empty. If upgrading, you can transfer
 ! 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/d9hapt1pu1fgrf
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
615 %
```

Database location

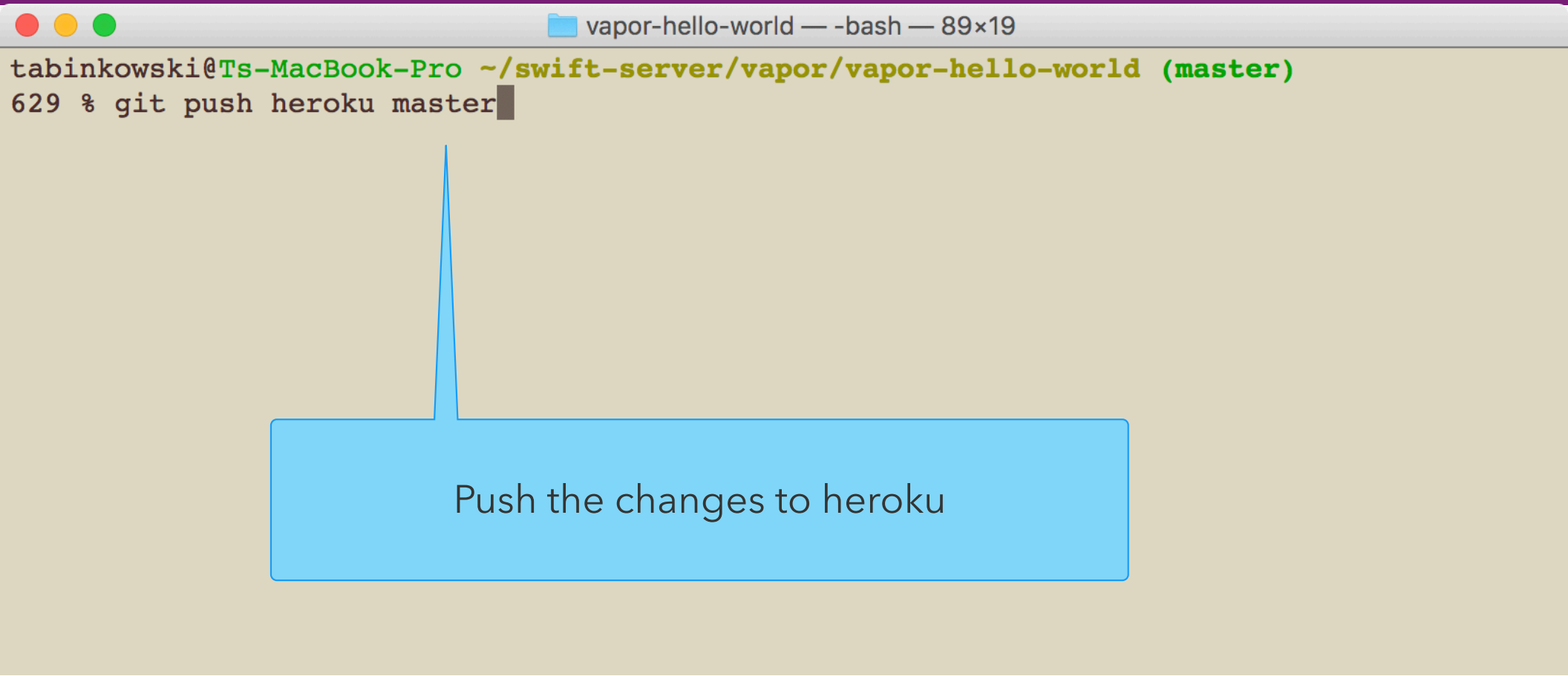
DEPLOY TO HEROKU WITH POSTGRESQL



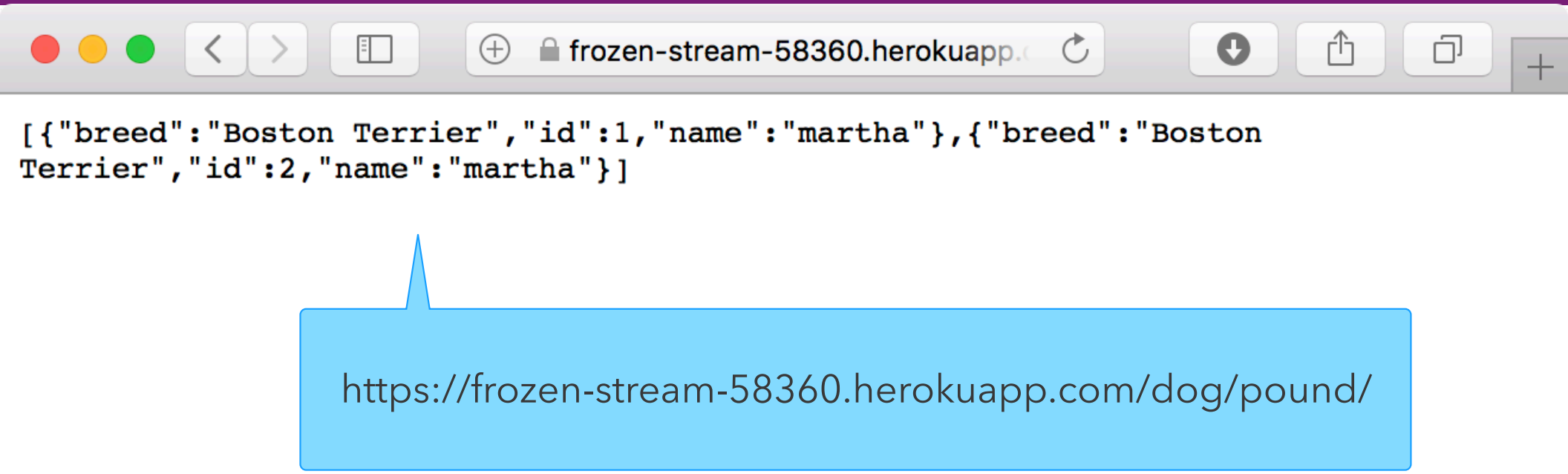
DEPLOY TO HEROKU WITH POSTGRESQL



DEPLOY TO HEROKU WITH POSTGRESQL



DEPLOY TO HEROKU WITH POSTGRESQL



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

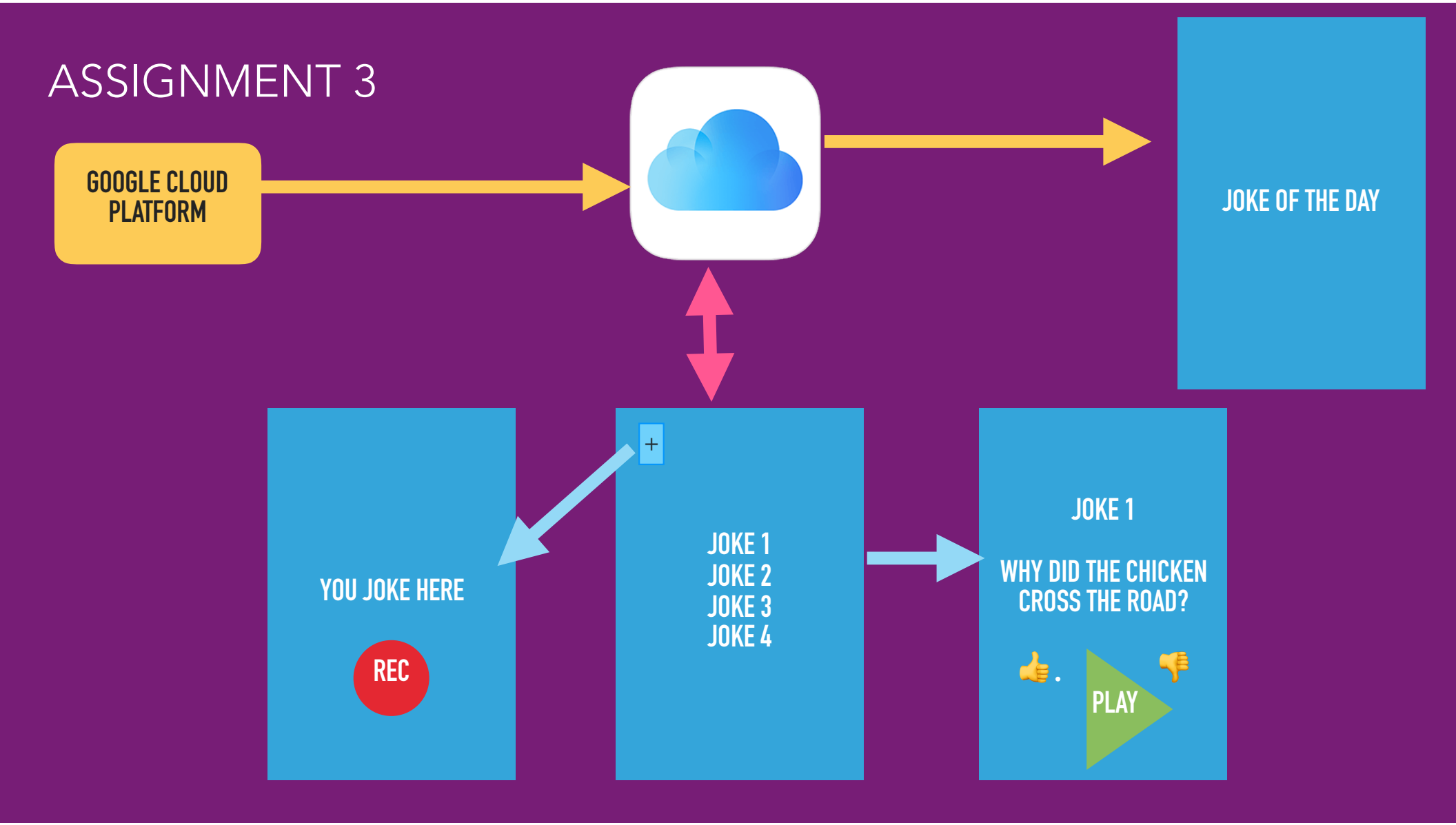
Save

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