

COMSE6998: Modern Serverless Cloud Applications

Lecture 6: Some Patterns, OAuth2, Social Media, Static Website, Graph DB

Dr. Donald F. Ferguson
Donald.F.Ferguson@gmail.com

Contents

- Introduction
 - Homework status, and some questions.
 - Could of Comments: Avoid direct model → code mapping.
 - Q&A
- New technical topics:
 - OAuth2
 - Social Media Integration (Review)
 - Building a Web Site – Serverless
 - Graph Databases (Intro).
 - ~~Content/Asset Management~~
- 3rd Project

Introduction

Contents

- Introduction
 - Homework status, and some questions.
 - Could of Comments: Avoid direct model → code mapping.
 - Q&A
- New technical topics:
 - OAuth2
 - Social Media Integration (Review)
 - Building a Web Site – Serverless
 - Graph Databases (Intro).
 - ~~Content/Asset Management~~
- 3rd Project

Some Comments

The screenshot shows the AWS Lambda API configuration interface. On the left, a sidebar lists resources and actions. A PUT action under the /customers resource is selected. The main panel displays the following:

- No header parameters exist for this method.
- No stage variables exist for this method.
- Request Body:

```
1 - {  
2   "email": "randomest@gmail.com",  
3   "address": "1234 awesome st",  
4   "firstname": "Jim",  
5   "lastname": "Rando",  
6   "phone_number": "1234567891"  
7 }
```
- A bulleted list of points related to the PUT request:

- PUT, DEL, GET → /customers/{email}
- /customers/{email}/address path,
- Shallow mapping is a problem when we start adding “context.”

Incoming/Outgoing “Event”

```
public class RESTMessage implements Cloneable {  
  
    private static final org.slf4j.Logger logger = LoggerFactory.getLogger(RESTMessage.class);  
  
    private Context context;  
    private Body body;  
    private Header header;  
    private Environment environment;  
  
    public RESTMessage() {  
    }  
  
    public Context getContext() {  
        return context;  
    }  
  
    public void setContext(Context context) {  
        this.context = context;  
    }  
  
    public Body getBody() {  
        return body;  
    }  
  
    public void setBody(Body body) {  
        this.body = body;  
    }  
  
    public Header getHeader() {  
        return header;  
    }  
  
    public void setHeader(Header header) {  
        this.header = header;  
    }  
  
    public Environment getEnvironment() {  
        return environment;  
    }  
  
    public void setEnvironment(Environment environment) {  
        this.environment = environment;  
    }  
  
    public String getOpId() {  
        return opId;  
    }  
  
    public void setOpId(String opId) {  
        this.opId = opId;  
    }  
  
    public Integer getMessageType() {  
        return messageType;  
    }  
  
    public void setMessageType(Integer messageType) {  
        this.messageType = messageType;  
    }  
  
    public String getVerb() {  
        return verb;  
    }  
  
    public void setVerb(String verb) {  
        this.verb = verb;  
    }  
  
    public String getResourcePath() {  
        return resourcePath;  
    }  
  
    public void setResourcePath(String resourcePath) {  
        this.resourcePath = resourcePath;  
    }  
  
    public MessageStatus getMessageStatus() {  
        return messageStatus;  
    }  
  
    public void setMessageStatus(MessageStatus messageStatus) {  
        this.messageStatus = messageStatus;  
    }  
}  
  
public class Header implements Cloneable {  
  
    private static final org.slf4j.Logger logger = LoggerFactory.getLogger(Header.class);  
  
    // The API gateway request mapping will set the operation ID.  
    // Otherwise, different invocation channels, if used, may set the ID. If not  
    // set,  
    // we should generate.  
    private String opId;  
  
    // The message could be a REQUEST or a RESPONSE  
    private Integer messageType;  
  
    // The verb is part of the operation, not the context.  
    private String verb;  
  
    // We use this to figure out which REST service to call. We should change  
    // this to the  
    // "path" to the resource, e.g. /api/customers, /api/customer/sociallogin.  
    private String resourcePath;  
  
    // MessageStatus
```

```
public class Context extends JSONUtil implements Cloneable{  
  
    private static final org.slf4j.Logger logger = LoggerFactory.getLogger(Context.class);  
  
    protected HashMap<String, String> headers;  
    protected HashMap<String, String> queryParams;  
    protected HashMap<String, String> pathParams;  
  
    public Context() {  
    }  
  
    public void addHeader(String key, String value) {  
        headers.put(key, value);  
    }  
  
    public void removeHeader(String key) {  
        headers.remove(key);  
    }  
  
    public void addQueryParam(String key, String value) {  
        queryParams.put(key, value);  
    }  
  
    public void removeQueryParam(String key) {  
        queryParams.remove(key);  
    }  
  
    public void addPathParam(String key, String value) {  
        pathParams.put(key, value);  
    }  
  
    public void removePathParam(String key) {  
        pathParams.remove(key);  
    }  
  
    public String getHeader(String key) {  
        return headers.get(key);  
    }  
  
    public String getQueryParam(String key) {  
        return queryParams.get(key);  
    }  
  
    public String getPathParam(String key) {  
        return pathParams.get(key);  
    }  
}
```

Technical Topics

OAuth2

Demos

- Facebook
- Twitter

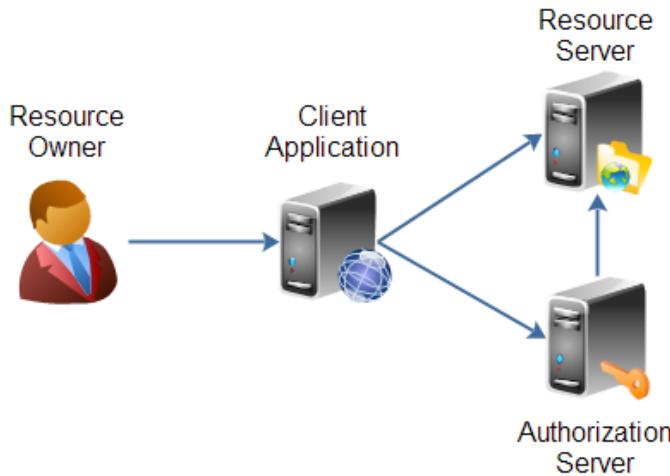
Overview (<http://tutorials.jenkov.com/oauth2/index.html>)

- Resource Owner
 - Controls *access* to the “data.”
 - Facebook **user**, LinkedIn **user**, ...
- Resource Server
 - The website that holds/manages info.
 - Facebook, LinkedIn, ...
 - And provides access API.
- Client Application
 - “The product you implemented.”
 - Wants to read/update
 - “On your behalf”
 - The data the data that the Resource Server maintains, e.g. posts, status, tweets, ...
- Authorization Server
 - Grants/rejects authorization
 - Based on Resource Owner decisions.
 - Usually (logically) the same as Resource Server.

OAuth 2.0 defines the following roles of users and applications:

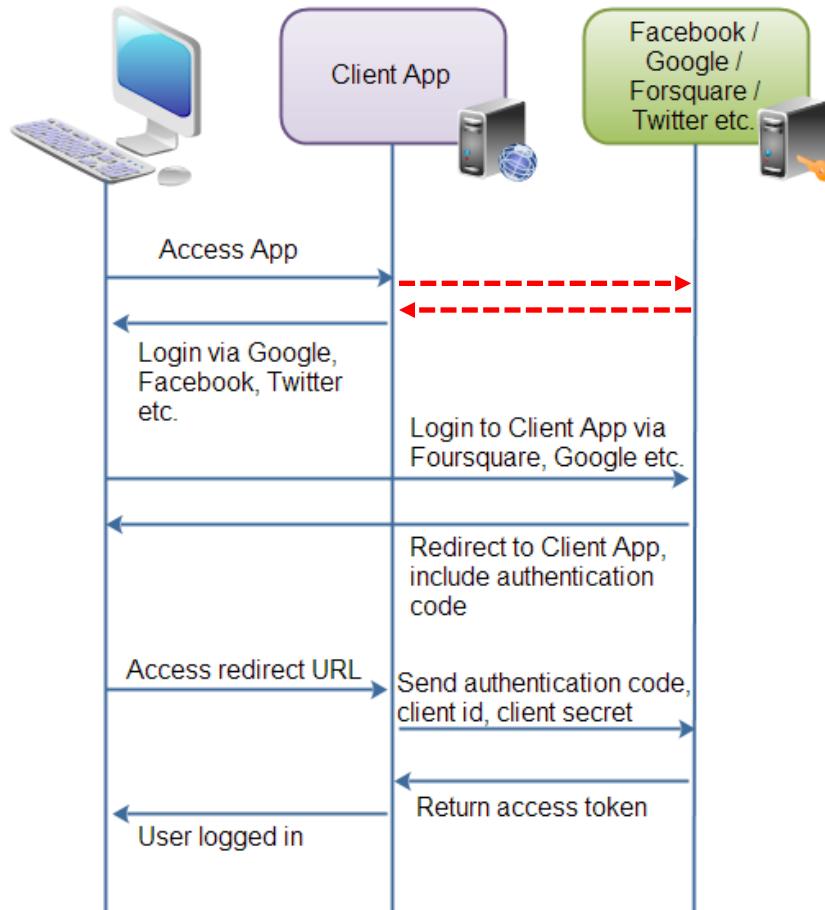
- Resource Owner
- Resource Server
- Client Application
- Authorization Server

These roles are illustrated in this diagram:



Roles/Flows

- User Clicks “Logon with XXX”
 - Redirect user to XXX
 - With Client App application ID.
 - And permissions app requests.
 - Code **MAY** have to call Resource Server to get a token to include in redirect URL.
- Browser redirected to XXX
 - Logon on to XXX prompt.
 - Followed by “do you grant permission to ...”
 - Clicking button drives a redirect back to Client App.
URL contains a temporary token.
- User/Browser
 - Redirected to Client App URL with token.
 - Client App calls XXX API
 - Obtains access token.
 - Returns to User.
- Client App can use access token on API calls.



Configuring Your Application at XXX

Details Settings Keys and Access Tokens Permissions

Application Details

Name *
SparqTVTest
Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description *
SparqTV Test Application
Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website *
Your Product Site
Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL
Redirect from Twitter
Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

Privacy Policy URL

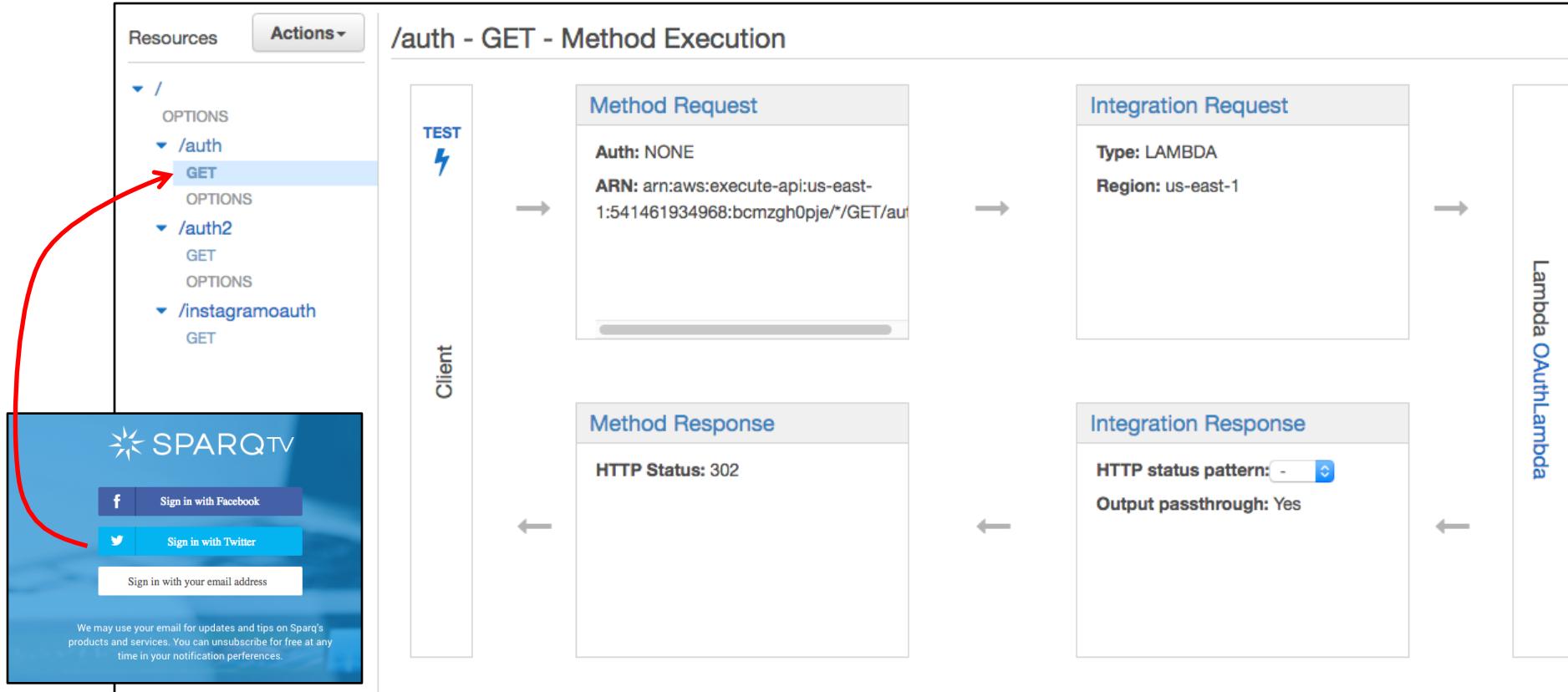
The URL for your application or service's privacy policy. The URL will be shared with users authorizing this application.

Terms of Service URL

The URL for your application or service's terms of service. The URL will be shared with users authorizing this application.

Enable Callback Locking (It is recommended to enable callback locking to ensure apps cannot overwrite the callback url)
 Allow this application to be used to Sign in with Twitter

Twitter Example – Step 1



Twitter Example - Step 1 (Cont)

First, declare response types using [Method Response](#). Then, map the possible responses from the backend to this method's response types.

Lambda Error Regex	Method response status	Output model	Default mapping
▶ 200	200		No
▼ -	302		Yes

Map the output from your Lambda function to the headers and output model of the 302 method response.

Lambda Error Regex ⓘ

Method response status ⓘ

Cancel Save

▼ Header Mappings

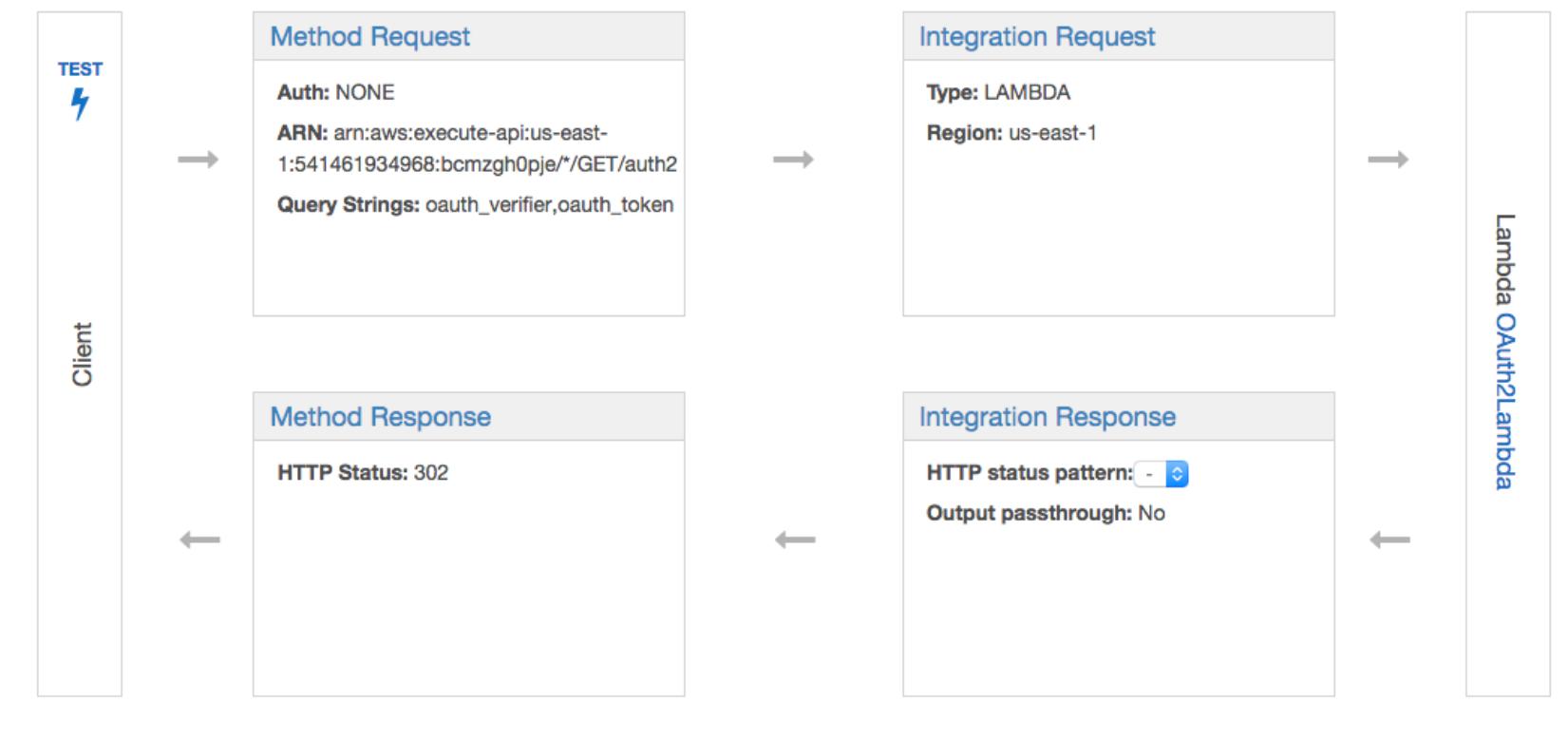
Response header	Mapping value ⓘ
Location	integration.response.body.location

▼ Body Mapping Templates

Content-Type	application/json
application/json	Generate template:
+ Add mapping template	<pre>1 #set(\$inputRoot = \$input.path('\$')) 2 { 3 \$input.json('\$.context.headers.redirect') 4 }</pre>

Step 2 – Same Basic Pattern

/auth2 - GET - Method Execution



```
public class TwitterConnector {  
  
    private static final String S3_BUCKET = "Base for our Web Site";  
    // Consumer Key (API Key)  
    private static final String TWITTER_CONSUMER_KEY = "Key (ID for App)";  
    // Consumer Secret (API Secret)  
    private static final String TWITTER_CONSUMER_SECRET = "Password";  
    // Info: Callback URL = S3_BUCKET + "/dev/auth2"  
  
    /*  
     * Interactions with Twitter require a RequestToken. There are two forms: 1.  
     * Just information identifying the application. 2. Also including user  
     * authentication information. This class represents the simple version.  
     */  
    public static class SimpleRequestToken {  
        String token;  
        String secret;  
  
        public SimpleRequestToken(String t, String s) {  
            this.token = t;  
            this.secret = s;  
        }  
    }  
  
    /*  
     * This is the persistent OAuth2 credential that we get back from completing  
     * the successful, second step in the Twitter login process. We save this  
     * for subsequent interactions.  
     *  
     * TODO I am not sure if we should embed the class inside the connector.  
     */  
    public class TwitterAuth {  
        public String oauth_token;  
        public String oauth_verifier;  
    }  
}
```

```
public class TwitterConnector {  
  
    private static final String S3_BUCKET = "Base for our Web Site";  
    // Consumer Key (API Key)  
    private static final String TWITTER_CONSUMER_KEY = "Key (ID for App)";  
    // Consumer Secret (API Secret)  
    private static final String TWITTER_CONSUMER_SECRET = "Password";  
    // Info: Callback URL = S3_BUCKET + "/dev/auth2"  
  
    /*  
     * Interactions with Twitter require a RequestToken. There are two forms: 1.  
     * Just information identifying the application. 2. Also including user  
     * authentication information. This class represents the simple version.  
     */  
    public static class SimpleRequestToken {  
        String token;  
        String secret;  
  
        public SimpleRequestToken(String t, String s) {  
            this.token = t;  
            this.secret = s;  
        }  
    }  
  
    /*  
     * This is the persistent OAuth2 credential that we get back  
     * the successful, second step in the Twitter login process. We  
     * for subsequent interactions.  
     *  
     * TODO I am not sure if we should embed the class inside the connector.  
     */  
    public class TwitterAuth {  
        public String oauth_token;  
        public String oauth_verifier;  
    }  
}
```

Hard coding these as strings is in the code is a bad idea.
But, I was lazy. Lazy is how security breaches occur.

```

private static boolean initializeConnector(String access_id, String access_token) {

    // Turns out that every Lambda function request does not get a fresh
    // JVM. So, consecutive requests were reusing the wrong request token.
    // so for now we just reinitialize every time.
    // TODO Fix this later.
    //
    // if (TwitterConnector.connectorInitialized == false) {

        logger.debug("Initializing the Twitter Connector");

        try {
            TwitterConnector.setup();
            TwitterConnector.connectorInitialized = true;

            logger.debug("Classes initialized.");

            // TODO We need to move this configuration information to somewhere
            // else.
            //
            twitterProperties = new Properties();
            twitterProperties.setProperty("oauth.consumerKey", TWITTER_CONSUMER_KEY);
            twitterProperties.setProperty("oauth.consumerSecret", TWITTER_CONSUMER_SECRET);
            if (access_id != null) {
                twitterProperties.setProperty("oauth.accessToken", access_id);
            }
            if (access_token != null) {
                twitterProperties.setProperty("oauth.accessTokenSecret", access_token);
            }

            logger.debug("Initialized properties.");

            PropertyConfiguration pc = new PropertyConfiguration(twitterProperties);
            twitter = new TwitterFactory(pc).getInstance();
            logger.debug("Initialized Twitter4J.");

        } catch (Exception e) {
            logger.warn("Exception while initializing Twitter Connector, e = " + e);
            connectorInitialized = false;
            twitterProperties = null;
            twitter = null;
        }

        logger.debug("Successfully initialized Twitter Connector.");
        // }

        return connectorInitialized;
    }
}

```

```
public static HTTPRedirect twitterRedirect1() {  
    HTTPRedirect result = null;  
    UUID statusUUID = null;  
  
    try {  
        logger.debug("Step 1 in Twitter OAuth2 login.");  
  
        TwitterConnector.initializeConnector();  
  
        logger.debug("Attempting to get Twitter request token.");  
        RequestToken requestToken = twitter.getOAuthRequestToken();  
  
        // TODO Writing this info to a log file is probably a security  
        // exposure.  
        logger.debug("After getOAuthRequestToken = " + "token=" + requestToken.getToken() + ", " + "secret=" +  
            + requestToken.getToken());  
  
        // Save the request token, which we will need for the second phase.  
        SimpleRequestToken st = new TwitterConnector.SimpleRequestToken(requestToken.getToken(),  
            requestToken.getTokenSecret());  
        statusUUID = saveState(st);  
  
        if (statusUUID != null) {  
            String authorizationUrl = requestToken.getAuthenticationURL();  
            result = new HTTPRedirect(authorizationUrl, HTTPRedirect.success);  
        } else {  
            logger.warn("Could not save SimpleRequestToken in SessionStorage");  
        }  
    } catch (Exception e) {  
        logger.warn("twitterRedirect1 failed with exception", e);  
    }  
  
    if (result == null) {  
        logger.warn("Could not get redirect URL. Returning error URL.");  
        result = new HTTPRedirect(failureUrl, HTTPRedirect.failure);  
    }  
  
    return result;  
}
```

```

public static HTTPRedirect twitterRedirect2(TwitterAuth accessToken) {
    long id;
    twitter4j.User u = null;
    AccessToken authorizationToken = null;
    String s = null;

    try {

        logger.debug("Attempting step 2 in Twitter Login. accessToken = " + JSONUtil.toPrettyJSON(accessToken));

        TwitterConnector.initializeConnector();

        logger.debug("Attempting to load state from step 1.");
        SimpleRequestToken st = TwitterConnector.loadState(accessToken.oauth_token);

        if (st != null) {
            logger.debug("Got simple request token = " + JSONUtil.toPrettyJSON(st));
            RequestToken rt = new RequestToken(st.token, st.secret);

            logger.debug("Attempting to get Authorization Token");
            authorizationToken = twitter.getOAuthAccessToken(rt, accessToken.oauth_verifier);
            logger.debug("Got Authorization Token = " + JSONUtil.toPrettyJSON(authorizationToken));
            s = JSONUtil.toSimpleJSON(authorizationToken);

            logger.debug("Trying to get info from Twitter.");
            id = twitter.getId();
            logger.debug("ID = " + id);
            u = twitter.showUser(id);
            logger.debug("User = " + prettyGson.toJson(u));
        }
    } catch (Exception e) {
        logger.debug("Exception = " + e);
        e.printStackTrace();
        u = null;
    }

    HTTPRedirect r;
    if (u != null) {
        r = new HTTPRedirect("http://" + S3_BUCKET + "#/twitterintegration?name=" + u.getName() + "&id=" +
            + u.getId() + "&token=" + authorizationToken.getToken() + "&tokenSecret=" +
            + authorizationToken.getTokenSecret() + "&result=true", HTTPRedirect.success);
    } else {
        r = new HTTPRedirect(TwitterConnector.failureUrl, HTTPRedirect.success);
    }
    logger.debug("returning " + JSONUtil.toPrettyJSON(r));
    return r;
}

```

OAuth Process Allows Us to

- Get user profile info

- Name
 - Email
 - Location
 -

- Get user Social network

- User follows ...
 - Follows user...

- Tweet on user's behalf
when something happens
in my application.

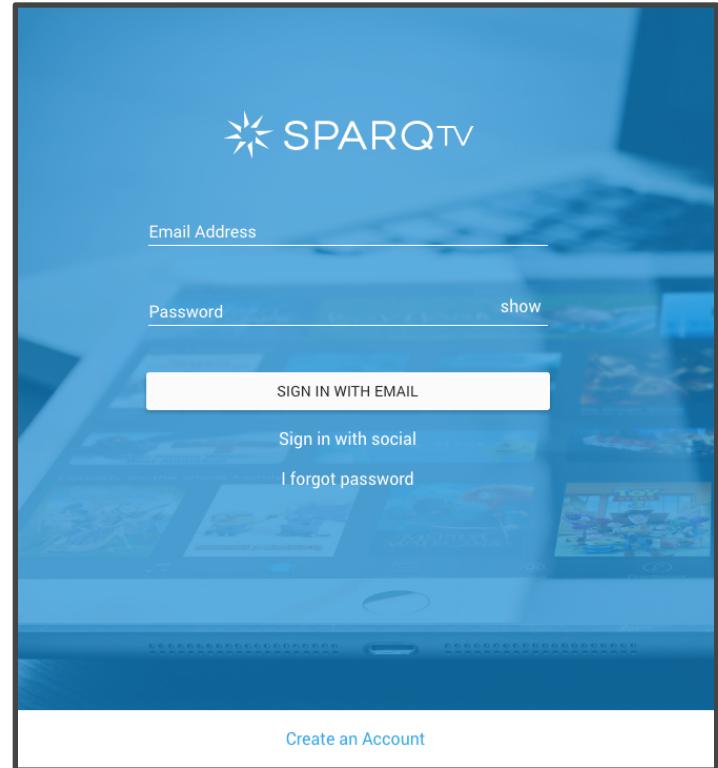
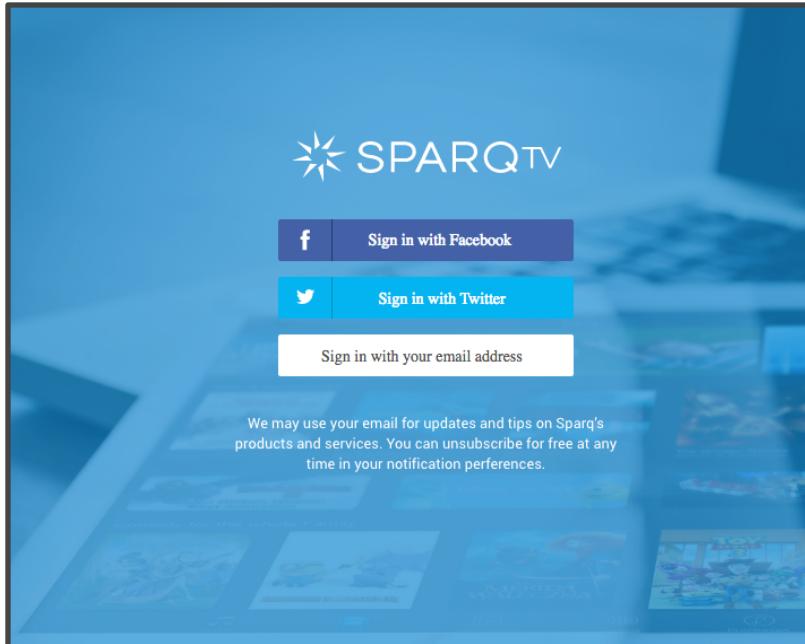
```
public static int getRetweets(String access_id, String access_token, String post_id) {  
    TwitterConnector.initializeConnector(access_id, access_token);  
    Status result = null;  
    int retweets = 0;  
    try {  
        long id = Long.parseLong(post_id);  
        result = twitter.showStatus(id);  
        retweets = result.getRetweetCount();  
    } catch (Exception e) {  
        e.printStackTrace();  
    } catch (OutOfMemoryError e) {  
        e.printStackTrace();  
    }  
    return retweets;  
}
```

```
public static List<Object> getFollowers(String access_id, String access_token) {  
    TwitterConnector.initializeConnector(access_id, access_token);  
    ArrayList<Object> followers = new ArrayList<Object>();  
    PagableResponseList<User> all = null;  
    long userId = Long.parseLong(access_id.substring(0, access_id.indexOf('-')));  
    try {  
        all = twitter.getFollowersList(userId, -1);  
        followers.addAll(all);  
        return followers;  
    } catch (Exception e) {  
        e.printStackTrace();  
    } catch (OutOfMemoryError e) {  
        e.printStackTrace();  
    }  
    return followers;  
}
```

Social Media

Register and Logon

Simplifies and expedites logon and registering for your product.



Federate Social Experience

Expands reach of your product to attract new users.

Simplifies your customers “sharing” what they are doing (no “cut and paste”)

The screenshot shows a social media feed interface. At the top, there is a blue header bar with the text "Some Friend". Below this, a profile picture of a couple is visible. The main content area displays a post from "Some Friend" with the text "Join us @ Polo" and "Polo SPARQ.tv: Polo". Below the post, there is a large white box containing the text "Sparq" and "Sparq TV Video Application". At the bottom of the feed, there is a blue button labeled "Some URL". Below the feed, there are standard social media interaction buttons for "Like", "Comment", and "Share", along with a "Write a comment..." input field and a camera icon.

The screenshot shows a social media feed interface, similar to the one on the left. At the top, there is a blue header bar with the text "Some Friend". Below this, a profile picture of a person is visible. The main content area displays a post from "Some Friend" with the text "SparqTV" and "cheese" @time 00:08:28". Below the post, there is a large white box containing the text "Sparq" and "Sparq TV Video Application". At the bottom of the feed, there is a blue button labeled "Some URL". Below the feed, there are standard social media interaction buttons for "Like", "Comment", and "Share", along with a "Write a comment..." input field and a camera icon.

Graph API

Provides user insights to enable customized, optimized interactions and support.

Graph API Explorer

Application: [?] Graph API Explorer ▾

Access Token: Access Token

GET → /v2.7/me?fields=id,name,email,devices,about,bio,education Learn more about the Graph API syntax

Node: me

- id
- name
- email
- devices
- about
- bio
- education

+ Search for a field

```
{
  "id": "10153132236678693",
  "name": "Donald Ferguson",
  "email": "donff2@aol.com",
  "devices": [
    {
      "hardware": "iPhone",
      "os": "iOS"
    },
    {
      "hardware": "iPad",
      "os": "iOS"
    },
    {
      "os": "Android"
    }
  ],
  "education": [
    {
      "school": {
        "id": "110154005680773",
        "name": "Cardinal Spellman High School"
      },
      "type": "High School",
      "id": "10153471493923693"
    },
    {
      "school": {
        "id": "109136799104262",
        "name": "Cardinal Spellman High School"
      },
      "type": "High School",
      "year": {
        "id": "138792749476094",
        "name": "1978"
      }
    }
  ]
}
```

Graph API

*Simplifies
inviting and finding
friends to use/
already using
the product.*

Graph API Explorer

Application: [?]

Graph API Explorer ▾

Access Token: [EAACED...ZC137Hpkv](#) [Get Token](#)

GET → /v2.7/me/friends

Edge: me/friends

+ Search for a field

1 Debug Message (Show)

```
[{"name": "John Doe", "id": "123456789"}, {"name": "Jane Smith", "id": "987654321"}, {"name": "Mike Johnson", "id": "345678901"}, {"name": "Sarah Williams", "id": "567890123"}, {"name": "David Lee", "id": "789012345"}, {"name": "Emily Davis", "id": "901234567"}, {"name": "Kevin Brown", "id": "135790246"}, {"name": "Linda Green", "id": "357902468"}, {"name": "Brian Wilson", "id": "579024680"}, {"name": "Amy Hill", "id": "790246812"}]
```

"paging": {
 "cursors": {
 "before": "QVFIUmtVRktsaXJRU3BpQ2tCVTR0S3BJVTY4T01paWpQdUp4ajZAoRkdKdEZAIIY01U0U5Ob2taVTJY0WgxMj1KbFnF1",
 "after": "QVFIUjBXckJzcW00ZAm1iUlZAtVUU2LXpyanpGcC1aMGY5Nkgwa3NGUzVNTE4LWxHNk1JdFnTcE1PVWZATQnBVchDc
 }
},
"summary": {
 "total_count": 633
}

Graph API Permissions

Graph API Explorer

Access Token: EAACED

Select Permissions

v2.7

Edge: me/friends
+ Search for a field

User Data Permissions

email ✕
 publish_actions ✕
 user_about_me ✕
 user_birthday ✕
 user_education_history ✕
 user_friends ✕
 user_games_activity ✕
 user_hometown ✕
 user_likes ✕
 user_location ✕
 user_photos ✕
 user_posts ✕
 user_relationship_details ✕
 user_relationshipships ✕
 user_religion_politics ✕
 user_status ✕
 user_tagged_places ✕
 user_videos ✕
 user_website ✕
 user_work_history ✕

Events, Groups & Pages

ads_management
 ads_read
 business_management ✕
 manage_pages ✕
 pages_manage_cta ✕
 publish_pages ✕
 pages.messaging
 pages.messaging_payments
 pages.messaging_phone_number
 pages.messaging_subscriptions
 pages.show_list ✕
 read_page_mailboxes
 rsvp_event
 user_events ✕
 user_managed_groups ✕
 pages.manage_instant_articles

Open Graph Actions

user_actions.books ✕
 user_actions.fitness ✕
 user_actions.music ✕
 user_actions.news ✕
 user_actions.video ✕

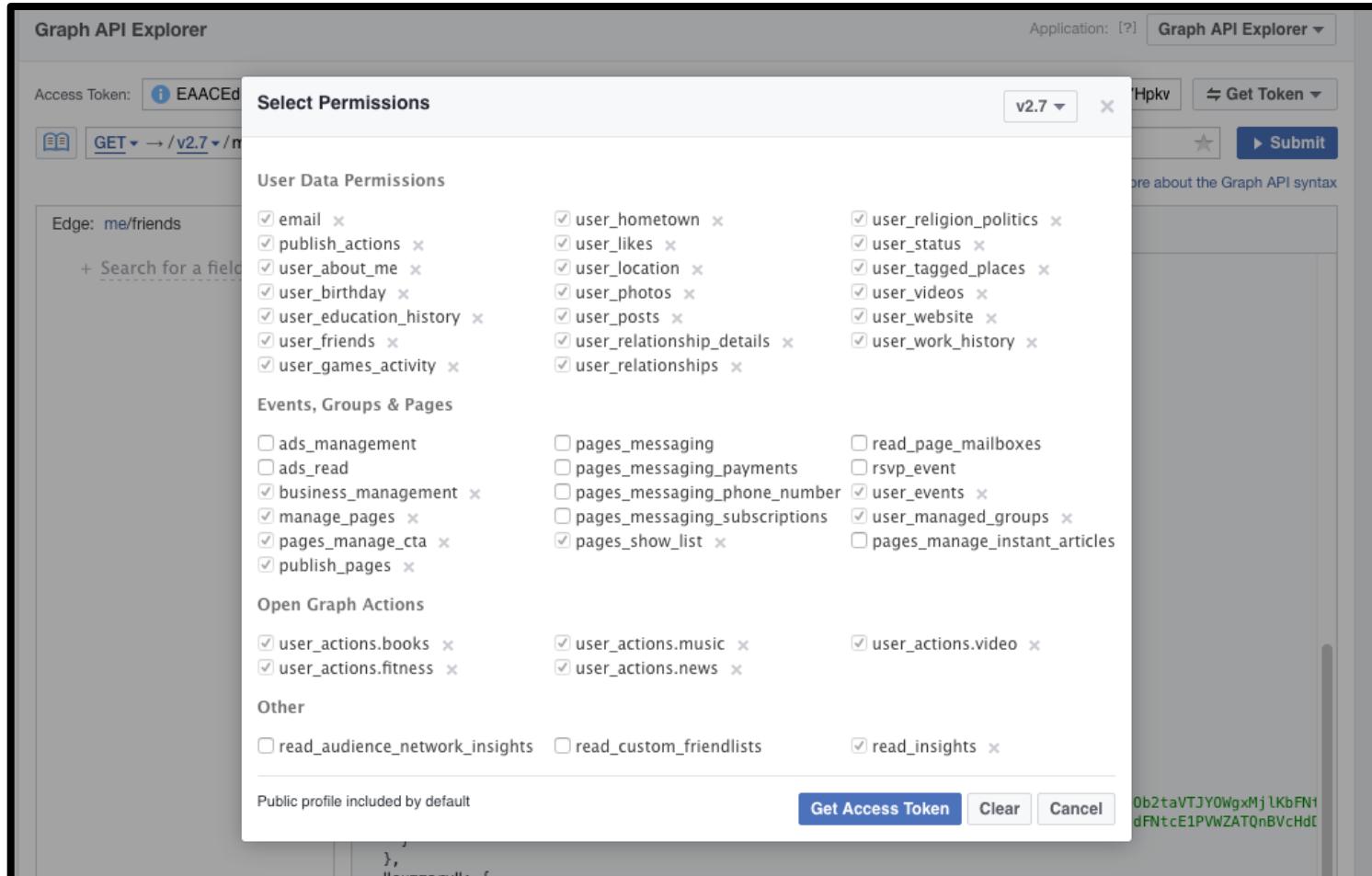
Other

read_audience_network_insights
 read_custom_friendlists
 read_insights ✕

Public profile included by default

Get Access Token Clear Cancel

Ob2taVTJY0WgxMj1KbFN1dFNtcE1PVWZATQnBVcHdC



Define an App

The screenshot shows the SparqTV app dashboard within the CourseWorks platform. The top navigation bar includes links for Apps, CourseWorks power..., Settings, MGRS Coordinates..., Center Harbor, NH..., body-parser, Best Online Courses..., An interactive, mod..., Tools & Support, and Docs. The user's profile picture is in the top right.

The left sidebar menu for the SparqTV app includes: Dashboard, Settings, Roles, Alerts, App Review, PRODUCTS, Facebook Login, and + Add Product.

The main dashboard area displays the following information:

- SparqTV •** (Green status icon)
- This app is public and available to all users.
- API Version: v2.5
- App ID: **App ID**
- App Secret: [REDACTED] [Show](#)

A red box highlights a call-to-action button: **Choose Platform**.

Below this, sections include:

- Get Started with the Facebook SDK**: Use our quick start guides to set up the Facebook SDK for your iOS or Android app, Canvas game or website. [Choose Platform](#)
- Facebook Analytics for Apps**: Set up Analytics. Analytics for Apps helps you grow your business and learn about the actions people take in your app. It only takes 5 minutes to set up. [Try Demo](#) [View Quickstart Guide](#)
- Facebook Login**: Active Login Users Trend. A line chart showing active login users from September 1st to October 1st. The chart shows three trends: Monthly Active, Weekly Active, and Daily Active users. The Daily Active user count is consistently the highest, peaking around 10.

Graph API Explorer Demo

Static Website

Web Application Basic Concepts

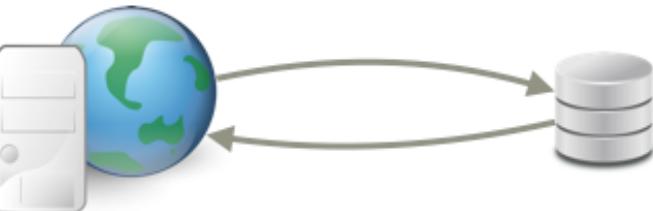
1. User performs an action that requires data from a database to be displayed.



2. A request is formed and sent from the client to the web server.



3. The request is processed and the database is queried.



6. Information is displayed to the user.

Application User

5. An appropriate response is generated and sent back.

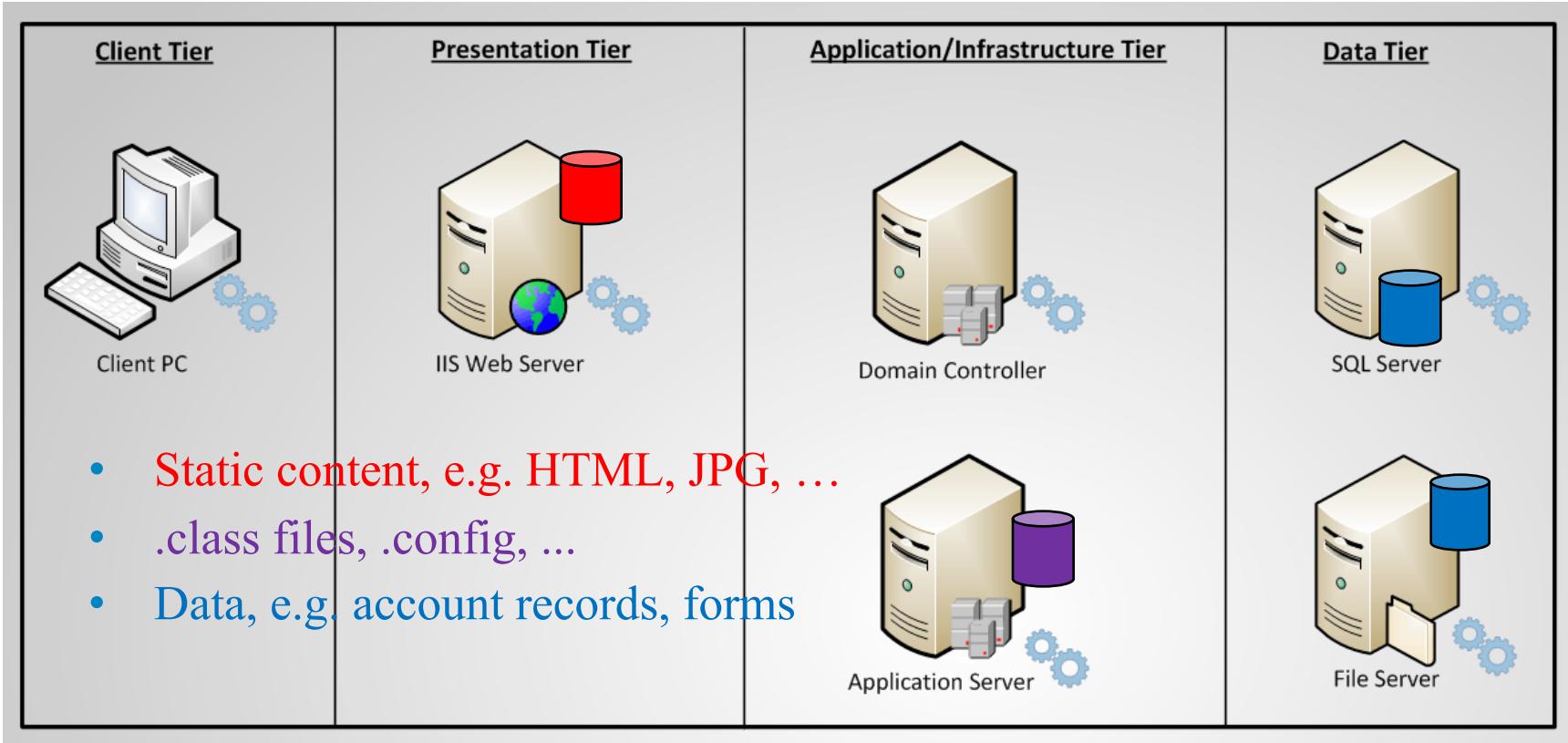
Web Client
(Presentation Tier)

4. Data is retrieved.

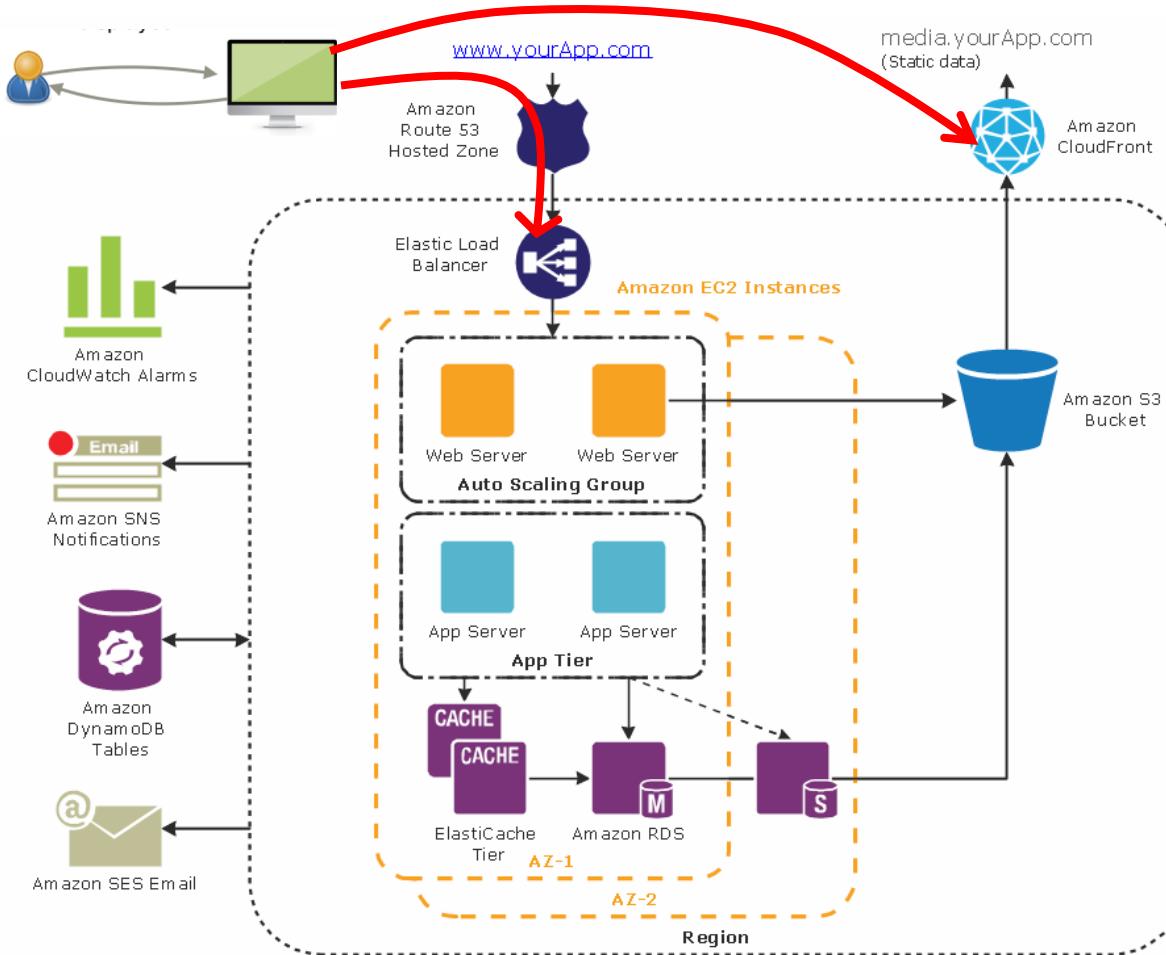
Web Server
(Application Tier)

Database
(Data Tier)

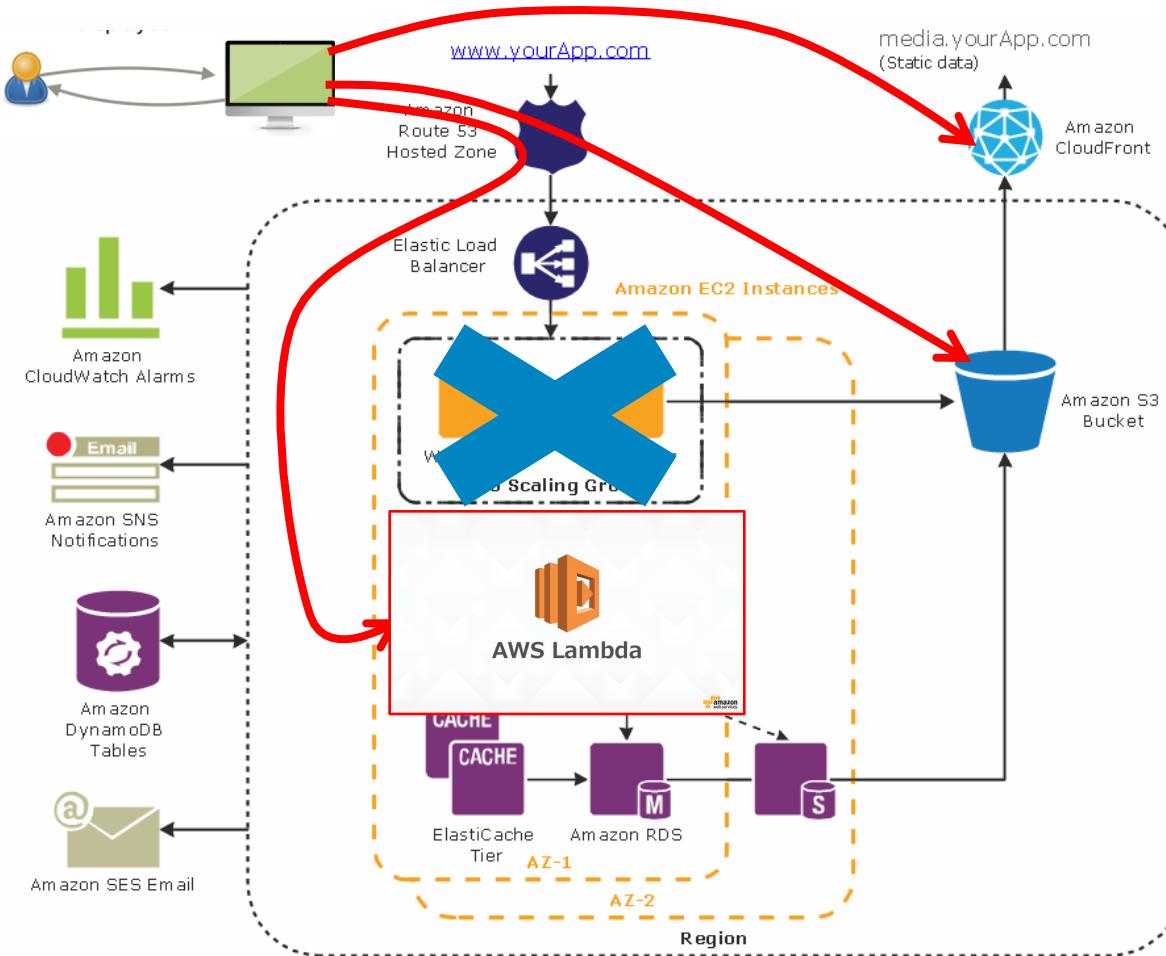
Physical Topology



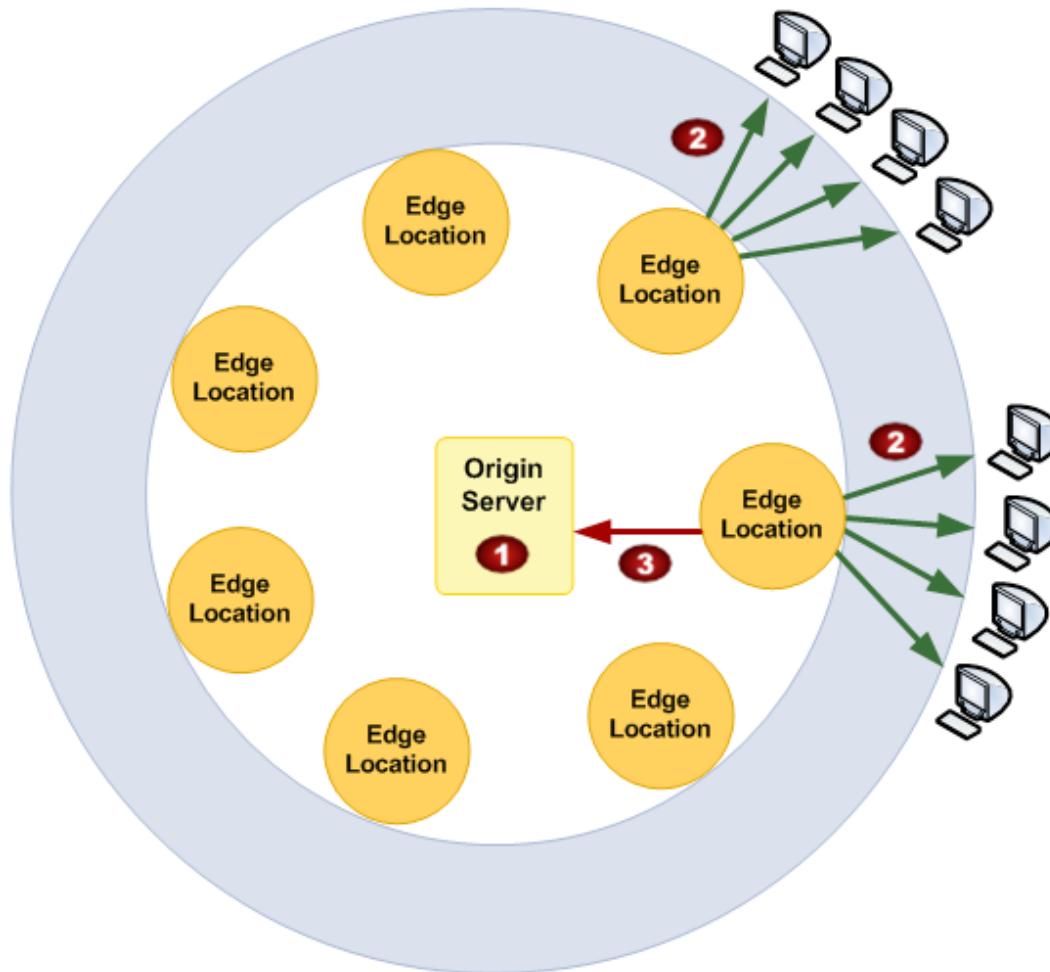
“Traditional” AWS Topology



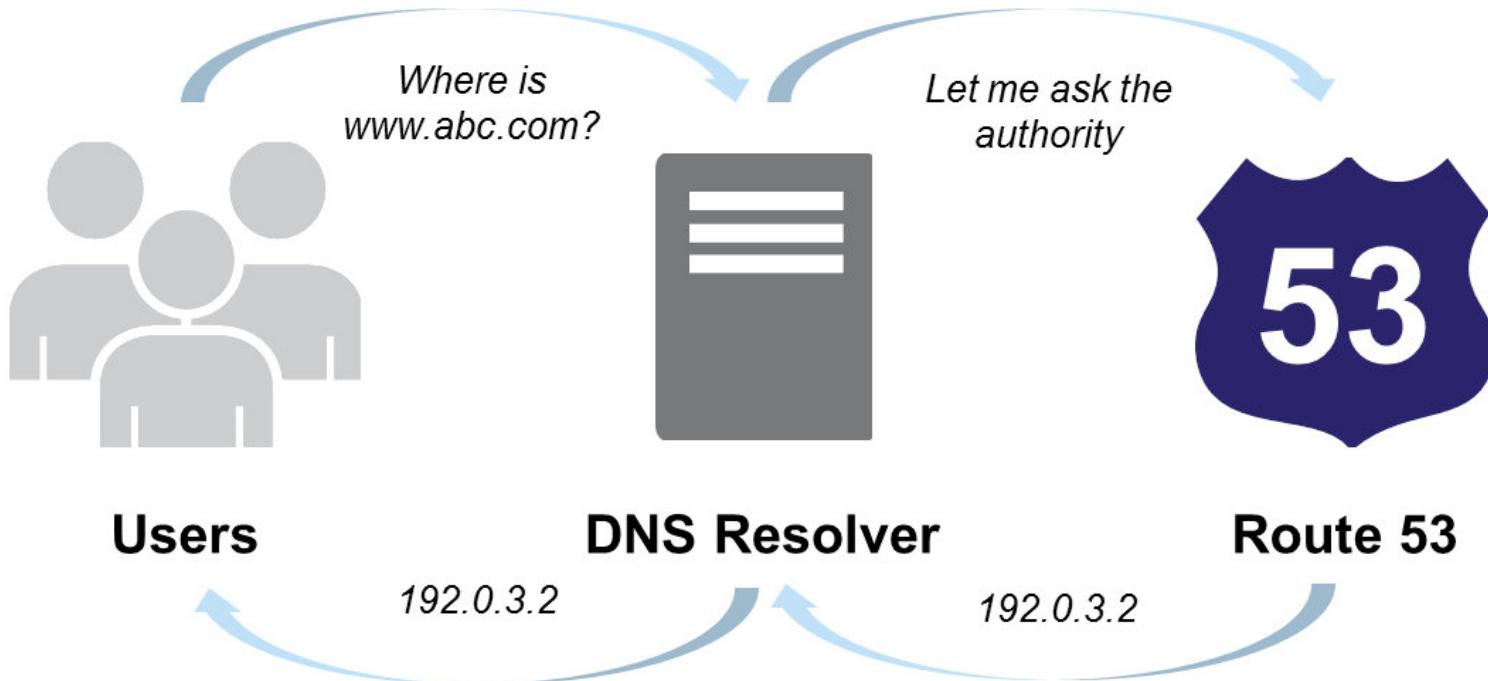
Our Model



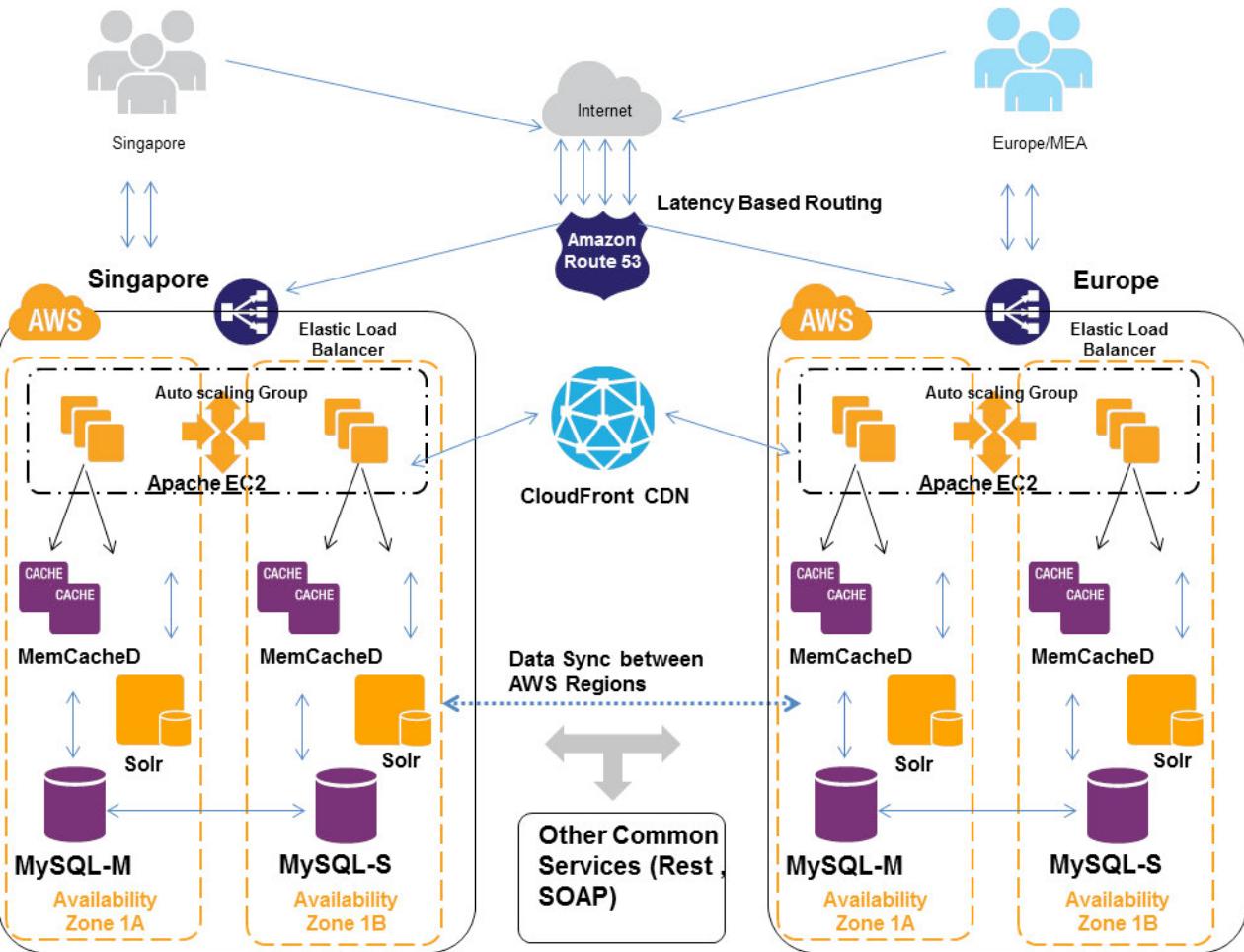
AWS CloudFront



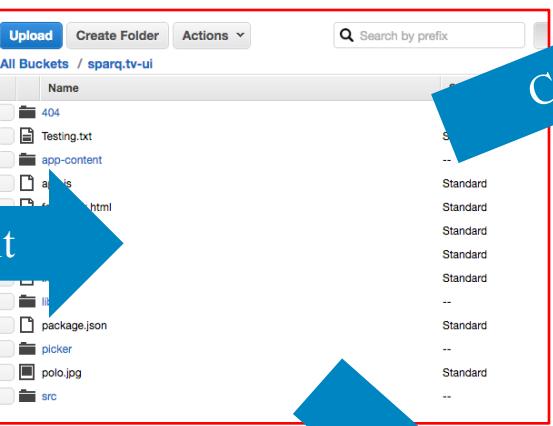
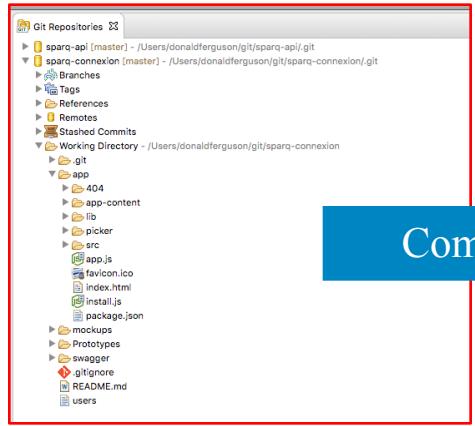
AWS Route 53



Example



Putting it Together



CloudFront Distributions

CloudFront Distributions		
Create Distribution		
Distribution Settings		
Viewing :	Any Delivery Method	Any State
Delivery Method	ID	Domain Name
Web	E11JPZG9QK06XL	d1qg8h4l6x6ma7.cloudfront.net
Web	E1VOXR00UTSTW6	d2hjlc013sr2bd.cloudfront.net
Web	E3J0JTRHR300ZS	d17lrqjhtsop1j.cloudfront.net
Web	EKBL9QAD2W9VF	d2kovksorxicly.cloudfront.net

Config

Commit

Config

CloudFront Hosted Zones			
Create Hosted Zone			
Go to Record Sets			
Search all fields	X	All Types	Displaying 1 to 1 of 1
Domain Name	Type	Record Set Count	Comment
sparq.tv.	Public	2	

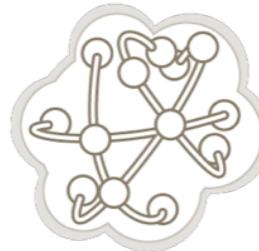
Graph Database (Intro)

Four NOSQL Categories

Key-Value



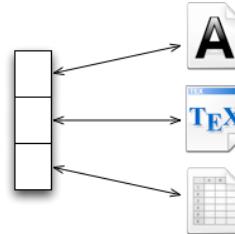
Graph DB



BigTable

A sparse matrix icon representing a BigTable, showing a grid of cells with some filled with the number '1'.

Document



www.ideal.ece.utexas.edu/courses/ee380l.../ppt/Neo4j.ppt

Pros and Cons

- Strengths
 - Powerful data model. Ideal, and simpler, for scenarios
 - Social networks
 - Relationships between people and things, which are also related.
 - etc.
 - Fast
 - For connected data, can be many orders of magnitude faster than RDBMS
- Weaknesses:
 - Sharding
 - Though they *can* scale reasonably well
 - And for some domains you can shard too!

Overview <http://graphdatabases.com/>

The Labeled Property Graph Model --

- A labeled property graph is made up of nodes, relationships, properties, and labels.
- Nodes contain properties. Think of nodes as documents that store properties in the form of arbitrary key-value pairs.
 - In Neo4j, the keys are strings and the values are the Java string and primitive data types, plus arrays of these types.
 - Nodes can be tagged with one or more labels.
 - Labels group nodes together, and indicate the roles they play within the dataset.
- Relationships connect nodes and structure the graph.
 - A relationship always has a direction, a single name, and a start node and an end node—there are no dangling relationships
 - Like nodes, relationships can also have properties.

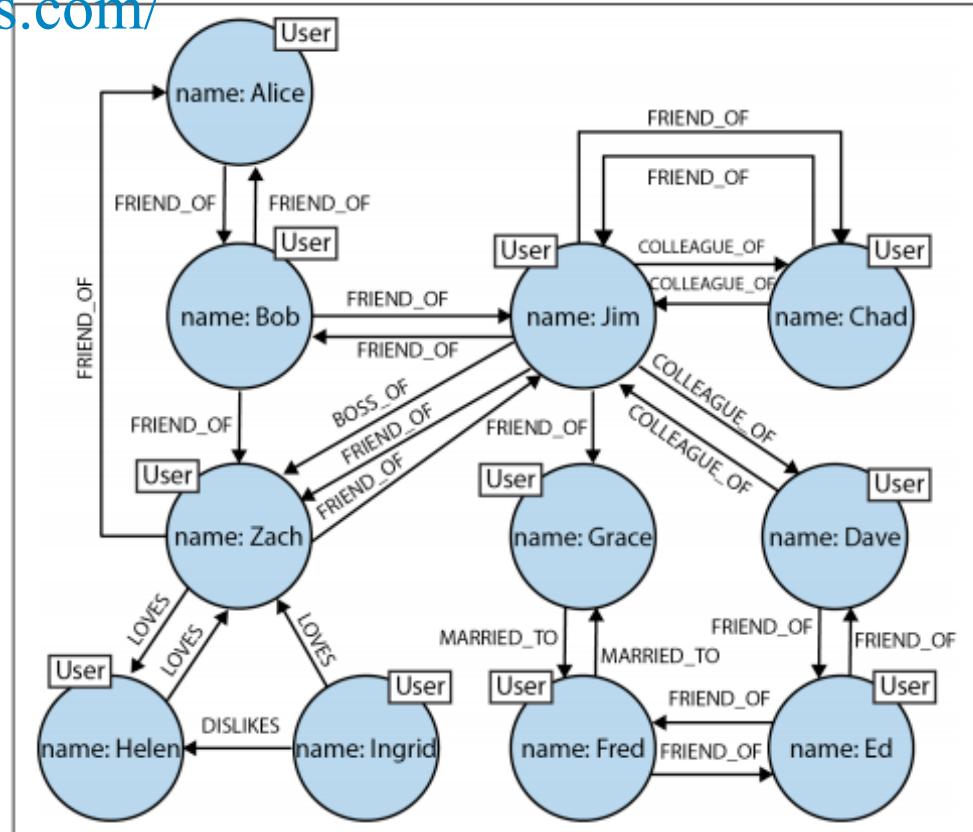


Figure 2-5. Easily modeling friends, colleagues, workers, and (unrequited) lovers in a graph

Social Network “path exists” Performance

- Experiment:
 - ~1k persons
 - Average 50 friends per person
 - `pathExists(a, b)` limited to depth 4
 - Caches warm to eliminate disk IO

	# persons	query time
Relational database	1000	2000ms
Neo4j	1000	2ms
Neo4j	1000000	2ms

www.ideal.ece.utexas.edu/courses/ee380l.../ppt/Neo4j.ppt

Neo4J— Cypher DDL

```
CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, tagline:'Welcome to the Real World'})  
CREATE (Keanu:Person {name:'Keanu Reeves', born:1964})  
CREATE (Carrie:Person {name:'Carrie-Anne Moss', born:1967})  
CREATE (Laurence:Person {name:'Laurence Fishburne', born:1961})  
CREATE (Hugo:Person {name:'Hugo Weaving', born:1960})  
CREATE (AndyW:Person {name:'Andy Wachowski', born:1967})  
CREATE (LanaW:Person {name:'Lana Wachowski', born:1965})  
CREATE (JoelS:Person {name:'Joel Silver', born:1952})  
  
CREATE  
    (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrix),  
    (Carrie)-[:ACTED_IN {roles:['Trinity']}]->(TheMatrix),  
    (Laurence)-[:ACTED_IN {roles:['Morpheus']}]->(TheMatrix),  
    (Hugo)-[:ACTED_IN {roles:['Agent Smith']}]->(TheMatrix),  
    (AndyW)-[:DIRECTED]->(TheMatrix),  
    (LanaW)-[:DIRECTED]->(TheMatrix),  
    (JoelS)-[:PRODUCED]->(TheMatrix)
```

Cypher DDL

```
CREATE (AFewGoodMen:Movie {title:"A Few Good Men", released:1992, tagline:"In the heart of the nation's capital")
CREATE (TomC:Person {name:'Tom Cruise', born:1962})
CREATE (JackN:Person {name:'Jack Nicholson', born:1937})
CREATE (DemiM:Person {name:'Demi Moore', born:1962})
CREATE (KevinB:Person {name:'Kevin Bacon', born:1958})
CREATE (KieferS:Person {name:'Kiefer Sutherland', born:1966})
CREATE (NoahW:Person {name:'Noah Wyle', born:1971})
CREATE (CubaG:Person {name:'Cuba Gooding Jr.', born:1968})
CREATE (KevinP:Person {name:'Kevin Pollak', born:1957})
CREATE (JTW:Person {name:'J.T. Walsh', born:1943})
CREATE (JamesM:Person {name:'James Marshall', born:1967})
CREATE (ChristopherG:Person {name:'Christopher Guest', born:1948})
CREATE (RobR:Person {name:'Rob Reiner', born:1947})
CREATE (AaronS:Person {name:'Aaron Sorkin', born:1961})
CREATE
```

Cypher DDL

CREATE

```
(TomC)-[:ACTED_IN {roles:['Lt. Daniel Kaffee']}]->(AFewGoodMen),  
(JackN)-[:ACTED_IN {roles:['Col. Nathan R. Jessup']}]->(AFewGoodMen),  
(DemiM)-[:ACTED_IN {roles:['Lt. Cdr. JoAnne Galloway']}]->(AFewGoodMen),  
(KevinB)-[:ACTED_IN {roles:['Capt. Jack Ross']}]->(AFewGoodMen),  
(KieferS)-[:ACTED_IN {roles:['Lt. Jonathan Kendrick']}]->(AFewGoodMen),  
(NoahW)-[:ACTED_IN {roles:['Cpl. Jeffrey Barnes']}]->(AFewGoodMen),  
(CubaG)-[:ACTED_IN {roles:['Cpl. Carl Hammaker']}]->(AFewGoodMen),  
(KevinP)-[:ACTED_IN {roles:['Lt. Sam Weinberg']}]->(AFewGoodMen),  
(JTW)-[:ACTED_IN {roles:['Lt. Col. Matthew Andrew Markinson']}]->(AFewGoodMen),  
(JamesM)-[:ACTED_IN {roles:['Pfc. Louden Downey']}]->(AFewGoodMen),  
(ChristopherG)-[:ACTED_IN {roles:['Dr. Stone']}]->(AFewGoodMen),  
(AaronS)-[:ACTED_IN {roles:['Man in Bar']}]->(AFewGoodMen),  
(RobR)-[:DIRECTED]->(AFewGoodMen),  
(AaronS)-[:WRITED]->(AFewGoodMen)
```

Cypher DML

```
$ MATCH (cloudAtlas {title: "Cloud Atlas"}) RETURN cloudAtlas
```



:play movie graph



The Movie Graph

Find

Example queries for finding individual nodes.

1. Click on any query example
2. Run the query from the editor
3. Notice the syntax pattern
4. Try looking for other movies or actors

Find the actor named "Tom Hanks"...

```
MATCH (tom {name: "Tom Hanks"}) RETURN tom
```

Find the movie with title "Cloud Atlas"...

```
MATCH (cloudAtlas {title: "Cloud Atlas"}) RETURN cloudAtlas
```

Find 10 people...

```
MATCH (people:Person) RETURN people.name LIMIT 10
```

Find movies released in the 1990s...

```
MATCH (nineties:Movie) WHERE nineties.released > 1990 AND nineties.released < 2000 RETURN nineties.title
```

Cypher DML

\$ MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED_IN]->(tomHanksMovies) RETURN tom,tomHanksMovies

Graph

*(13) Movie(12) Person(1)

*(13) ACTED_IN(12) DIRECTED(1)

Rows

A Text

</> Code

```
graph TD; TomHanks((Tom Hanks)) -- ACTED_IN --> CloudAtlas((Cloud Atlas)); TomHanks -- ACTED_IN --> DaVinciCode((The Da Vinci Code)); TomHanks -- ACTED_IN --> GreenMile((The Green Mile)); TomHanks -- ACTED_IN --> Apollo13((Apollo 13)); TomHanks -- ACTED_IN --> CastAway((Cast Away)); TomHanks -- ACTED_IN --> LeagueOfTheirOwn((A League of Their Own)); TomHanks -- ACTED_IN --> PolarExpress((The Polar Express)); TomHanks -- ACTED_IN --> CharliesWomansWar((Charlie Wilson's War)); TomHanks -- ACTED_IN --> YouveGotMail((You've Got Mail)); TomHanks -- ACTED_IN --> Sleepless((Sleepless)); TomHanks -- ACTED_IN --> JoeVersusTheVolcano((Joe Versus the Volcano)); TomHanks -- ACTED_IN --> ThatThingYouDo((That Thing You Do)); TomHanks -- DIRECTED --> JoeVersusTheVolcano;
```

Displaying 13 nodes, 13 relationships (completed with 13 additional relationships).

AUTO-COMPLETE

Cypher DML

```
$ MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[ :ACTED_IN]-(coActors) RETURN coActors.name
```



coActors.name

Rows
Julia Roberts

A
Text
Philip Seymour Hoffman

Geena Davis

</>
Code
Lori Petty

Rosie O'Donnell

Madonna

Bill Paxton

Helen Hunt

Kevin Bacon

Gary Sinise

Ed Harris

Bill Paxton

Michael Clarke Duncan

Patricia Clarkson

Gary Sinise

Bonnie Hunt

Returned 39 rows in 74 ms.

Cypher DML

\$ MATCH (people:Person)-[relatedTo]-(:Movie {title: "Cloud Atlas"}) RETURN people.name, Type(relatedTo), relatedTo

*(11) Movie(1) Person(10)

*(10) ACTED_IN(4) DIRECTED(3) PRODUCED(1) REVIEWED(1) WROTE(1)

Graph Rows Text Code



Displaying 11 nodes, 10 relationships.

AUTO-COMPLETE

Some Spooky Stuff

The Movie Graph

Solve

You've heard of the classic "Six Degrees of Kevin Bacon"? That is simply a shortest path query called the "Bacon Path".

1. Variable length patterns
2. Built-in shortestPath() algorithm

The Movie Graph

Recommend

Let's recommend new co-actors for Tom Hanks. A basic recommendation approach is to find connections past an immediate neighborhood which are themselves well connected.

For Tom Hanks, that means:

1. Find actors that Tom Hanks hasn't yet worked with, but his co-actors have.
2. Find someone who can introduce Tom to his potential co-actor.

Movies and actors up to 4 "hops" away from Kevin Bacon

```
MATCH (bacon:Person {name:"Kevin Bacon"})-[*1..4]-(hollywood)
RETURN DISTINCT hollywood
```

Bacon path, the shortest path of any relationships to Meg Ryan

```
MATCH p=shortestPath(
  (bacon:Person {name:"Kevin Bacon"})-[*]-(meg:Person {name:"Meg Ryan"})
)
RETURN p
```

Extend Tom Hanks co-actors, to find co-co-actors who haven't work with Tom Hanks...

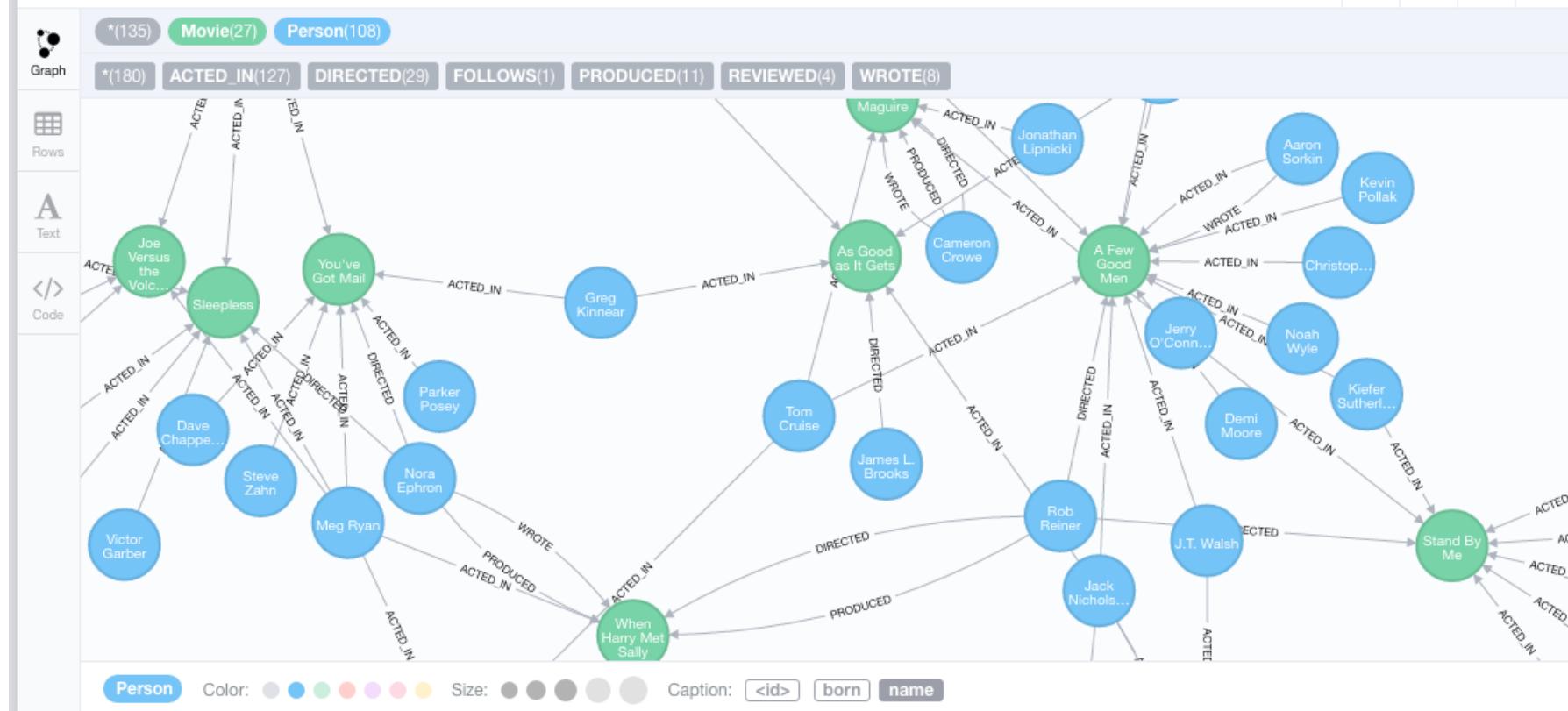
```
MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors),
      (coActors)-[:ACTED_IN]->(m2)<-[:ACTED_IN]-(cocoActors)
WHERE NOT (tom)-[:ACTED_IN]->(m2)
RETURN cocoActors.name AS Recommended, count(*) AS Strength ORDER BY Strength DESC
```

Find someone to introduce Tom Hanks to Tom Cruise

```
MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors),
      (coActors)-[:ACTED_IN]->(m2)<-[:ACTED_IN]-(cruise:Person {name:"Tom Cruise"})
RETURN tom, m, coActors, m2, cruise
```

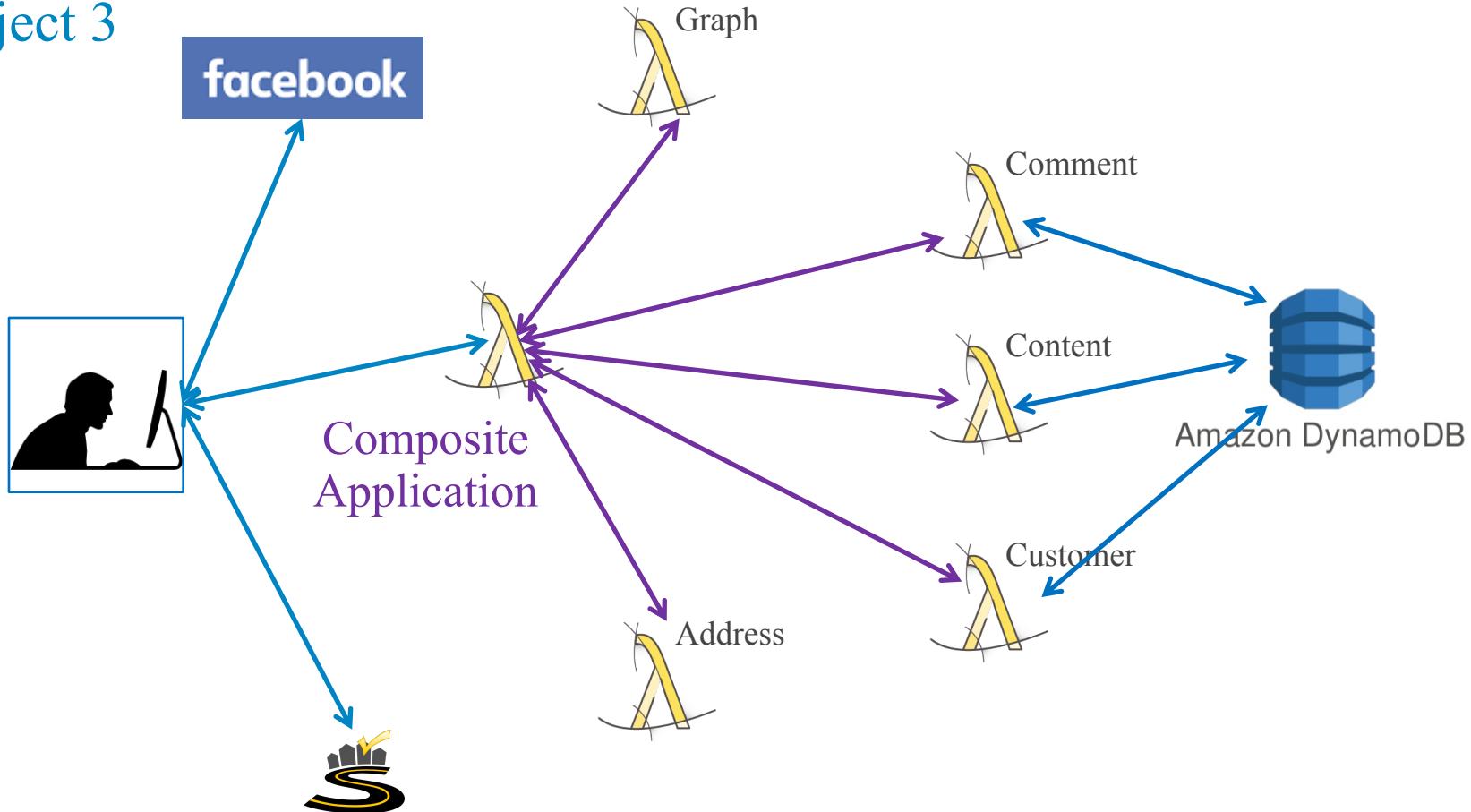
Some Spooky Stuff

```
$ MATCH (bacon:Person {name:"Kevin Bacon"})-[1..4]-(hollywood) RETURN DISTINCT hollywood
```



3rd Project

Project 3



3rd Project (Part 1)

- Define and provide a Lambda/REST interface to a “content catalog” in DynamoDB.
 - Model: Content Instance
 - Property (e.g. HBO, CBS) is a set of one or more Franchises.
 - A Franchise (e.g. Game of Thrones, Bing Bang Theory) is a set of one or more Series.
 - A Series (e.g. Season 1, Christmas Specials) is a set of one or more Episodes.
 - An Episode is an instance of a “TV show.”
 - Data for each type is just (ID, name).
 - Comment (ID, user, content instance, comment)
- Graph database
 - Signup for/begin to use REST callable Neo4J database (e.g. graphenedb.com)
 - Node type are Person, Content (one of Property, Franchise, ...) and Comment.
 - Relationships between people (Follows, Friends)
 - Relationships between people and comments (commented, responded, liked, ...)
 - Relationships between people and content (Likes, Rates, ...)
- Expand REST API and Lambda functions to provide integrated API for accessing data and relationships
 - Customer/person, Comment, Content Instance, ... in DynamoDB including property links.
 - CRUD on Customer, Content Instance, Comment create nodes and links in DynamoDB.
- Login and Register with Facebook
 - Register creates name, email, etc. From Facebook data.
 - User can change address, name, preferred email, etc.

3rd Project (Part 1)

- Define and provide full REST interface “as a service” in Python DB
 - Model
 - Create
 - Read
 - Update
 - Delete
- Welcome to the “Big Leagues,” or H*ll, depending on your point-of-view.
- This project is going to “force you” to
 - Think in terms of user stories, scenarios, ...
 - Do some basic high-level design diagrams.
 - Carefully think about API model
 - etc.
- Expand Project to include:
 - Customer
 - CRUD
- Three weeks completion, with interim reviews.
- Login and Registration
 - Register creates name, email, etc. From Facebook data.
 - User can change address, name, preferred email, etc.