CTD309

AWS re:INVENT

Building Serverless Websites with Lambda@Edge

George John
Product Manager
Amazon CloudFront/Lambda@Edge

Manigandan Radhakrishnan
Senior Software Development Engineer
Amazon CloudFront/Lambda@Edge

December 1, 2017

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

What Is Covered in This Session

- Why run websites at the edge?
- What is Lambda@Edge?
- How can Lambda@Edge help?

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

We will start up with a fully hosted website, we will deconstruct it as we go along, and finally get to a state where the same site is now hosted in a Serverless manner at an edge location. We will see why running websites at the edge might be beneficial.

In the Beginning ...

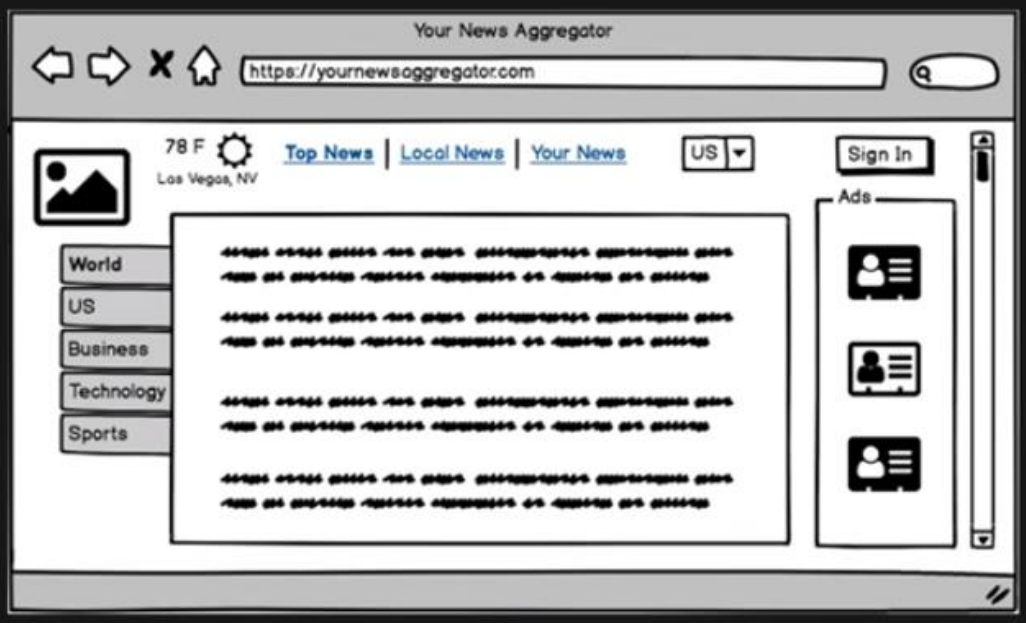
- You have a breakthrough idea

- You decide to build a MVP

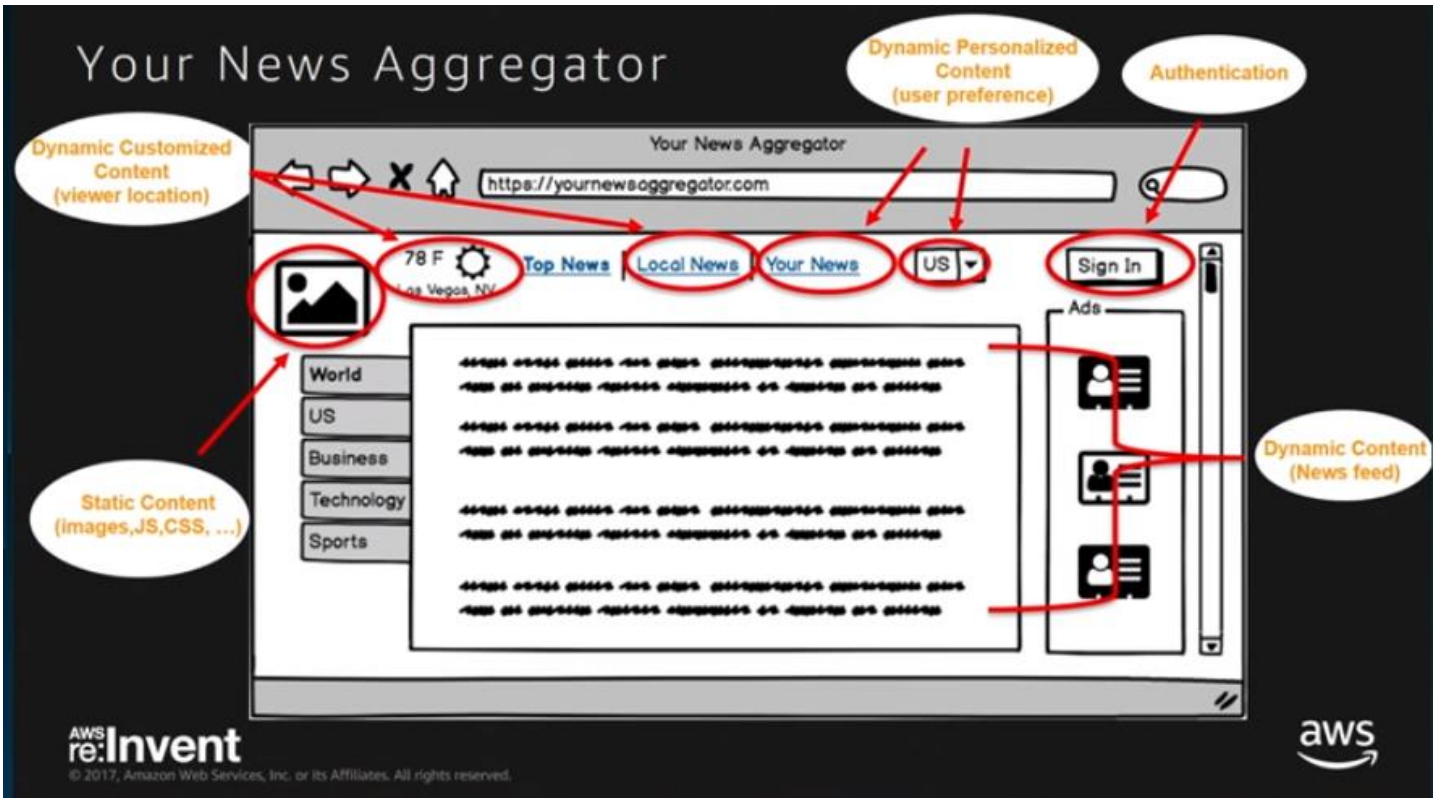
- You choose AWS as your platform

AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

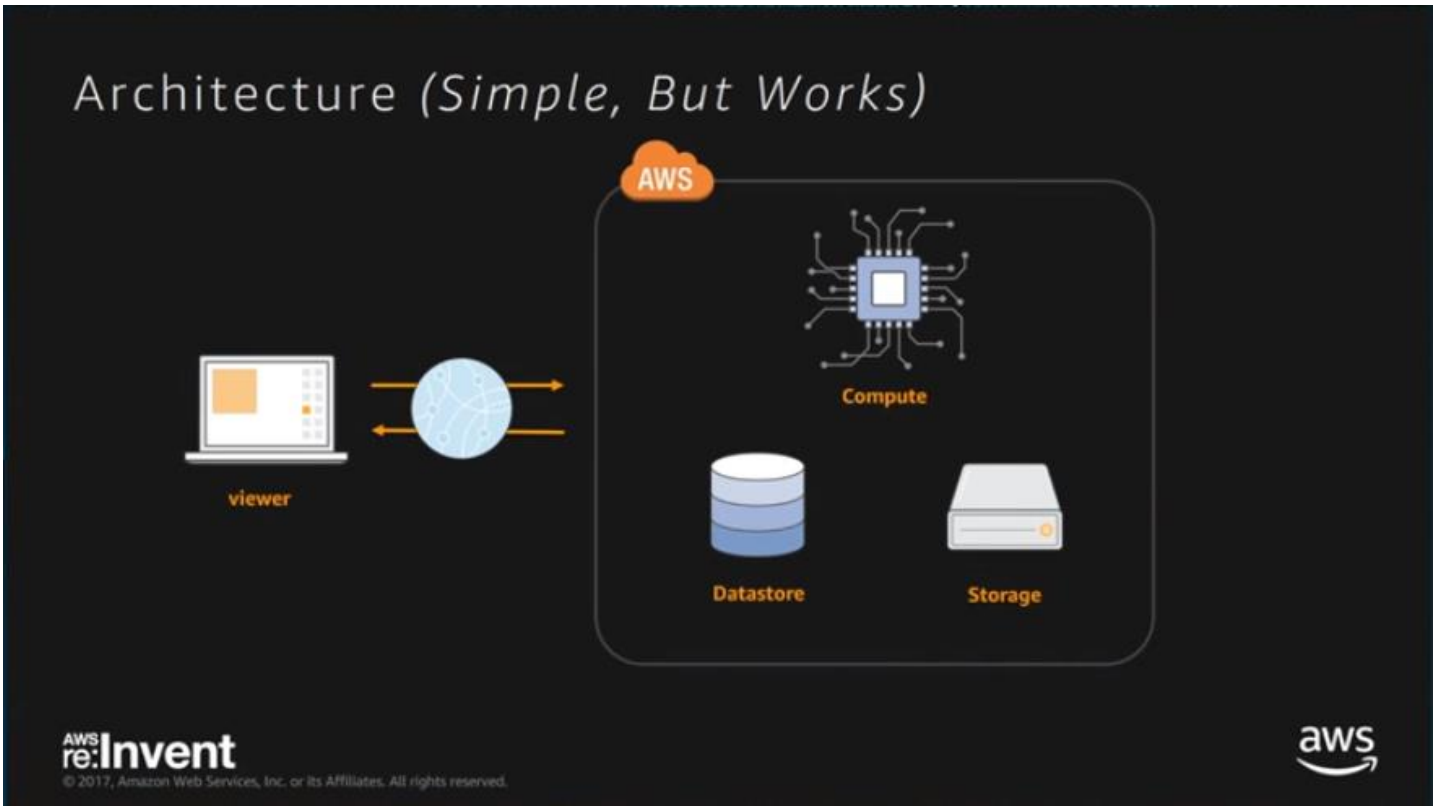
Your News Aggregator



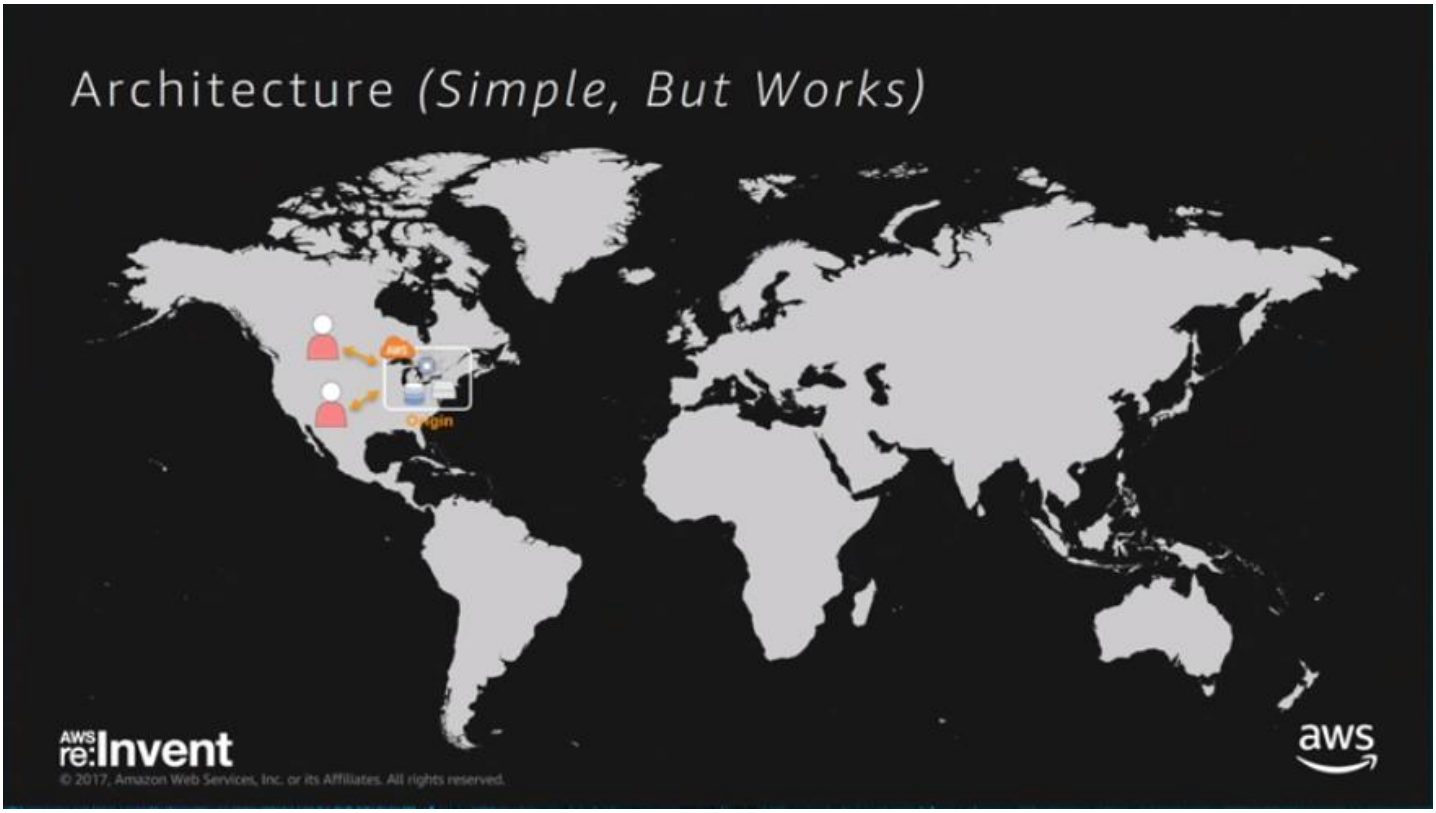
AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



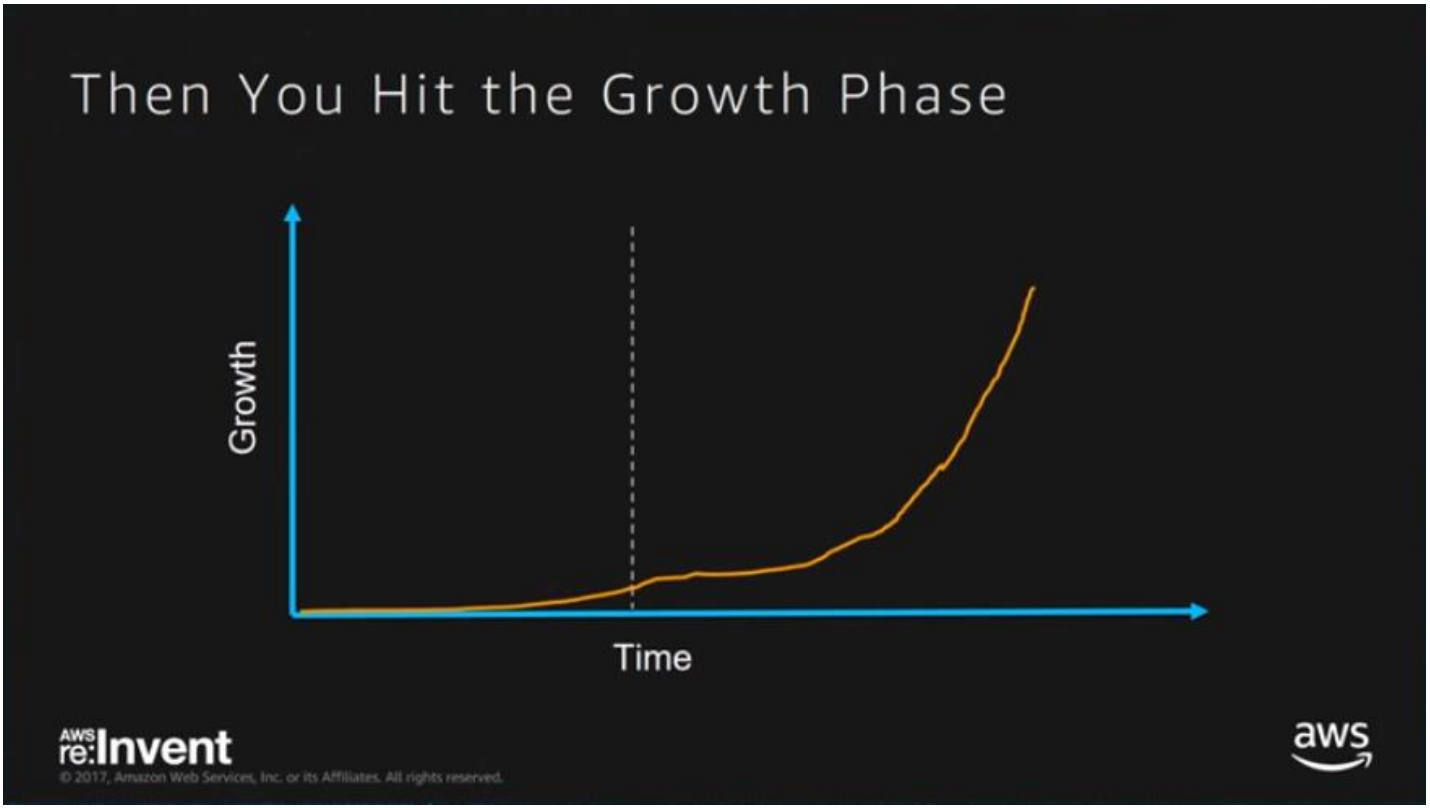
Things on the website like the images, JS, HTML, CSS files can be stored and cached in a location like S3. Then you have dynamic contents that will be changing on your site. You also have another type of dynamic contents that is very customized to the viewer depending on where the user is or what attributes they have like their names, location, whether the user is on mobile or desktop, etc.



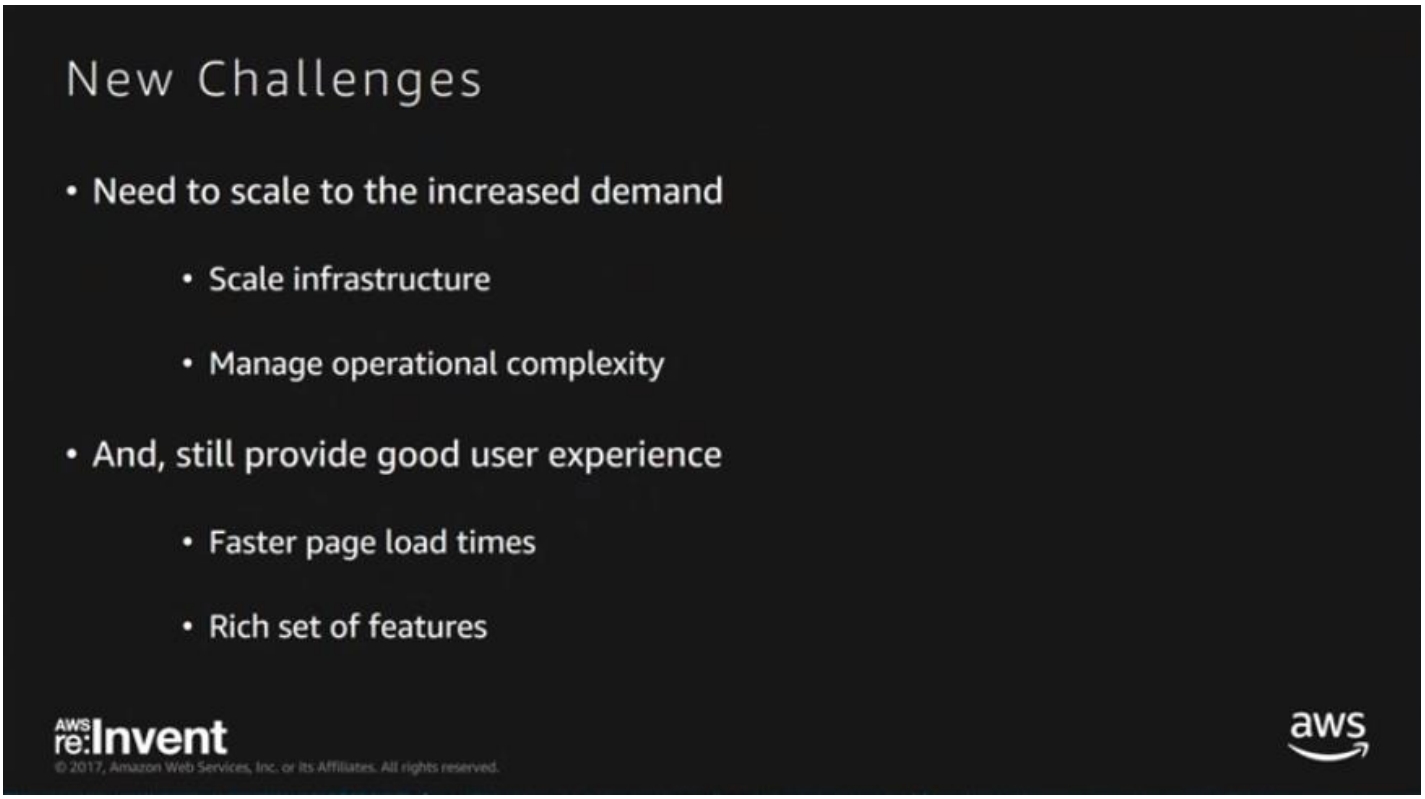
A typical website application will have some Compute like an EC2 instance, some storage like S3, a datastore like a RDBMS or a NoSQL database.

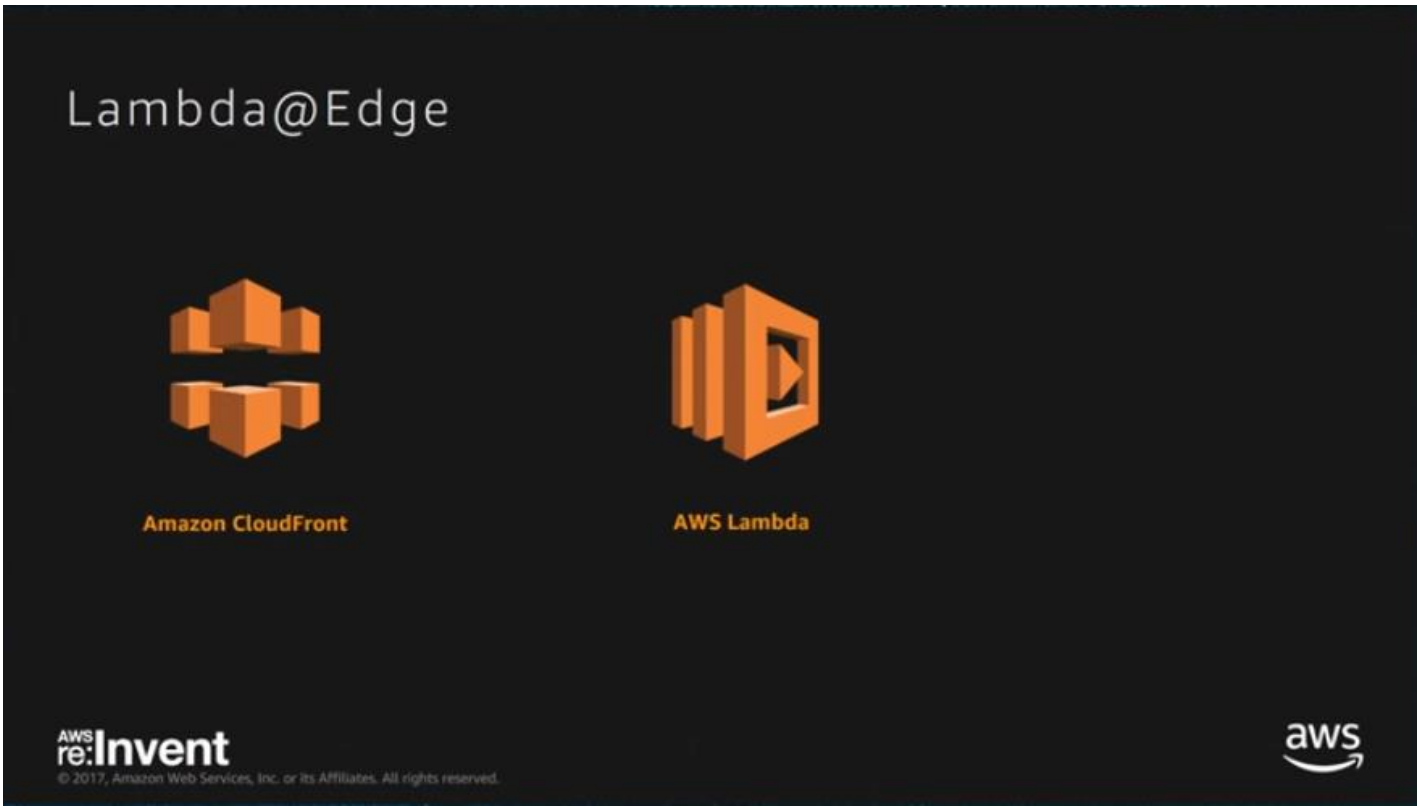


This works fine in the initial stages of your app

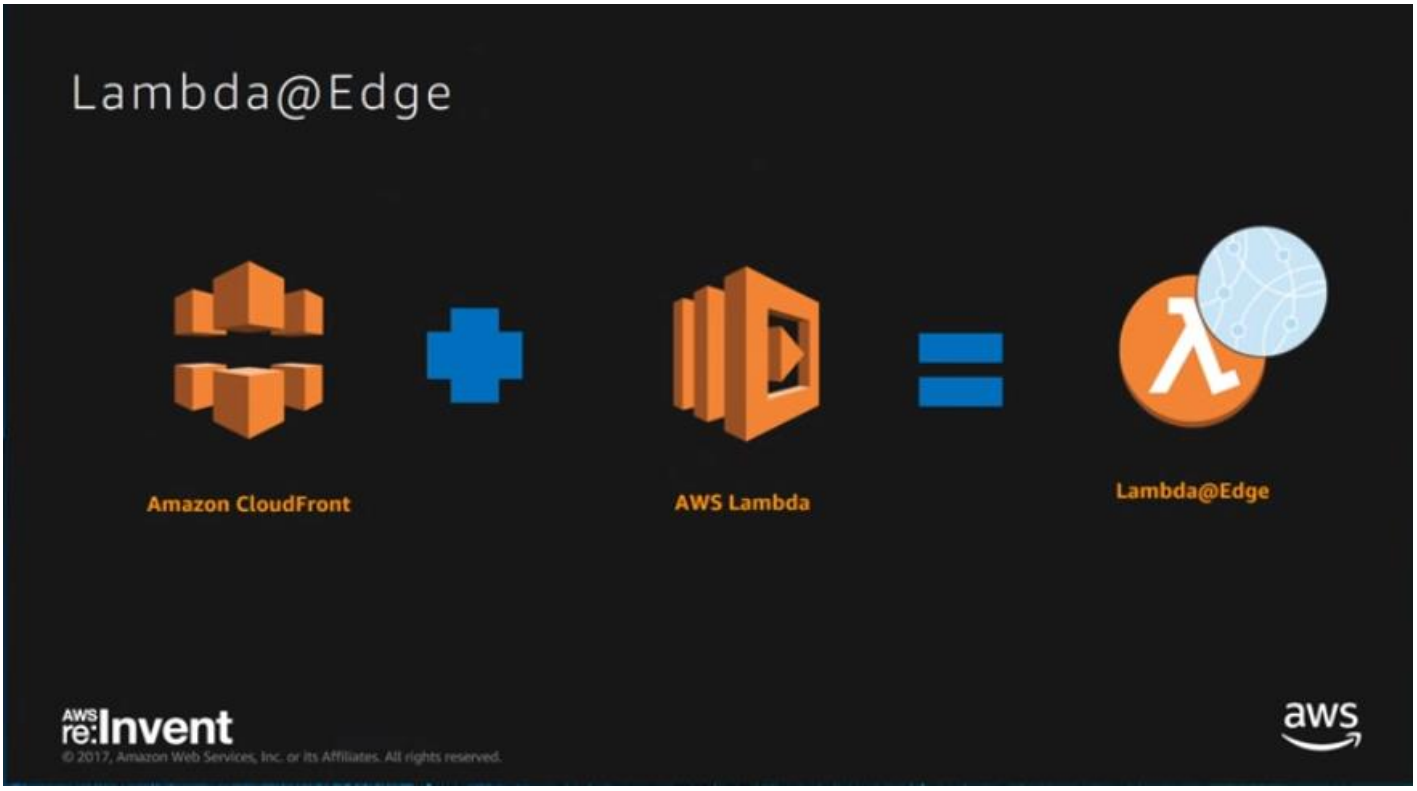


Now you have users across the globe all trying to access your website, the performance starts to degrade slowly





AWS today has services that can help with these issues. ***Amazon CloudFront*** is a Content Distribution Network service (CDN) that can be used to cache a lot of your static contents, and the AWS Lambda service can help address the scaling and operational overhead challenges you are having because lambda is serverless compute.



The AWS Lambda@Edge service is an amalgamation of the capabilities of both CloudFront and lambda.

Amazon CloudFront
Content Delivery Network

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

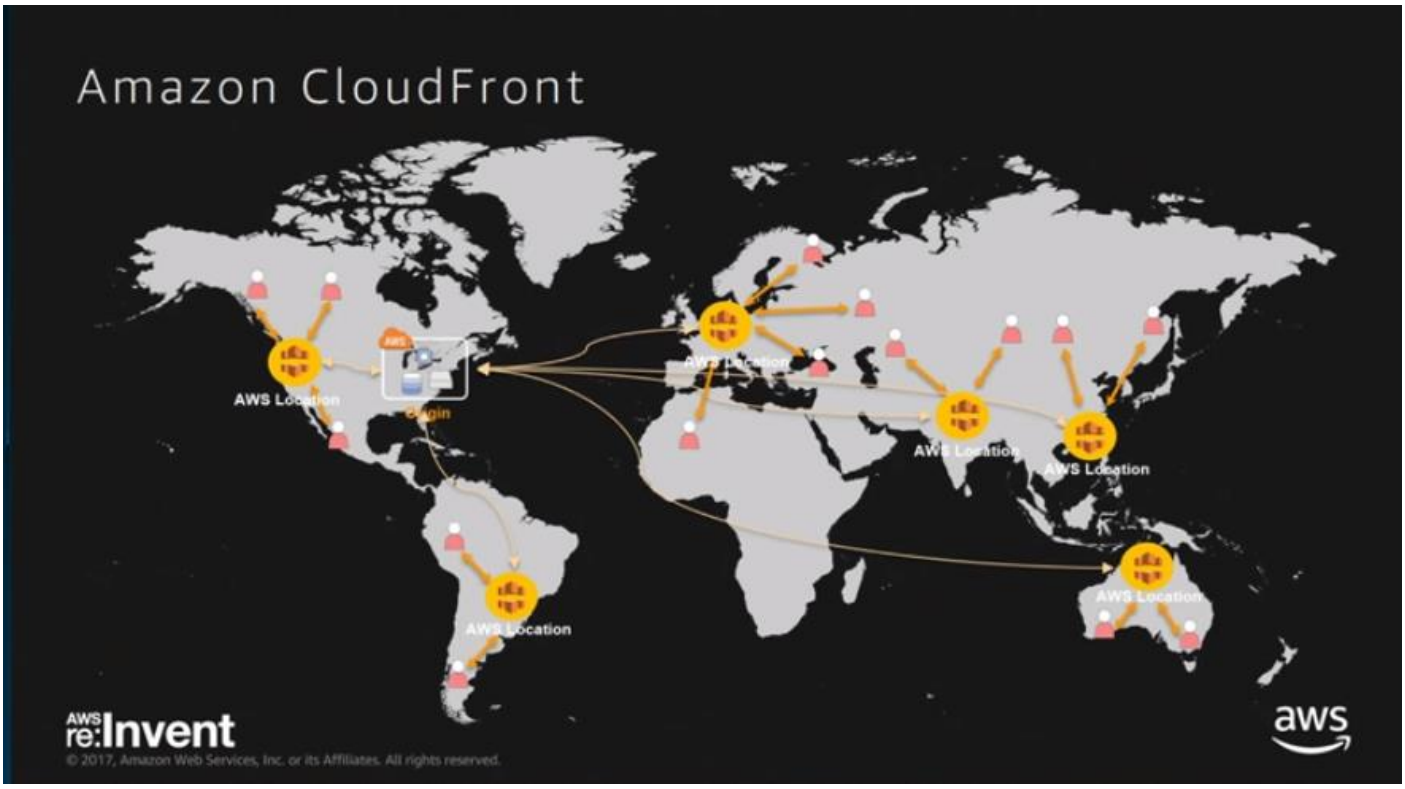
aws

Amazon CloudFront Global Content Delivery Network
107 PoPs (96 Edge Locations + 11 Regional Edge Caches)

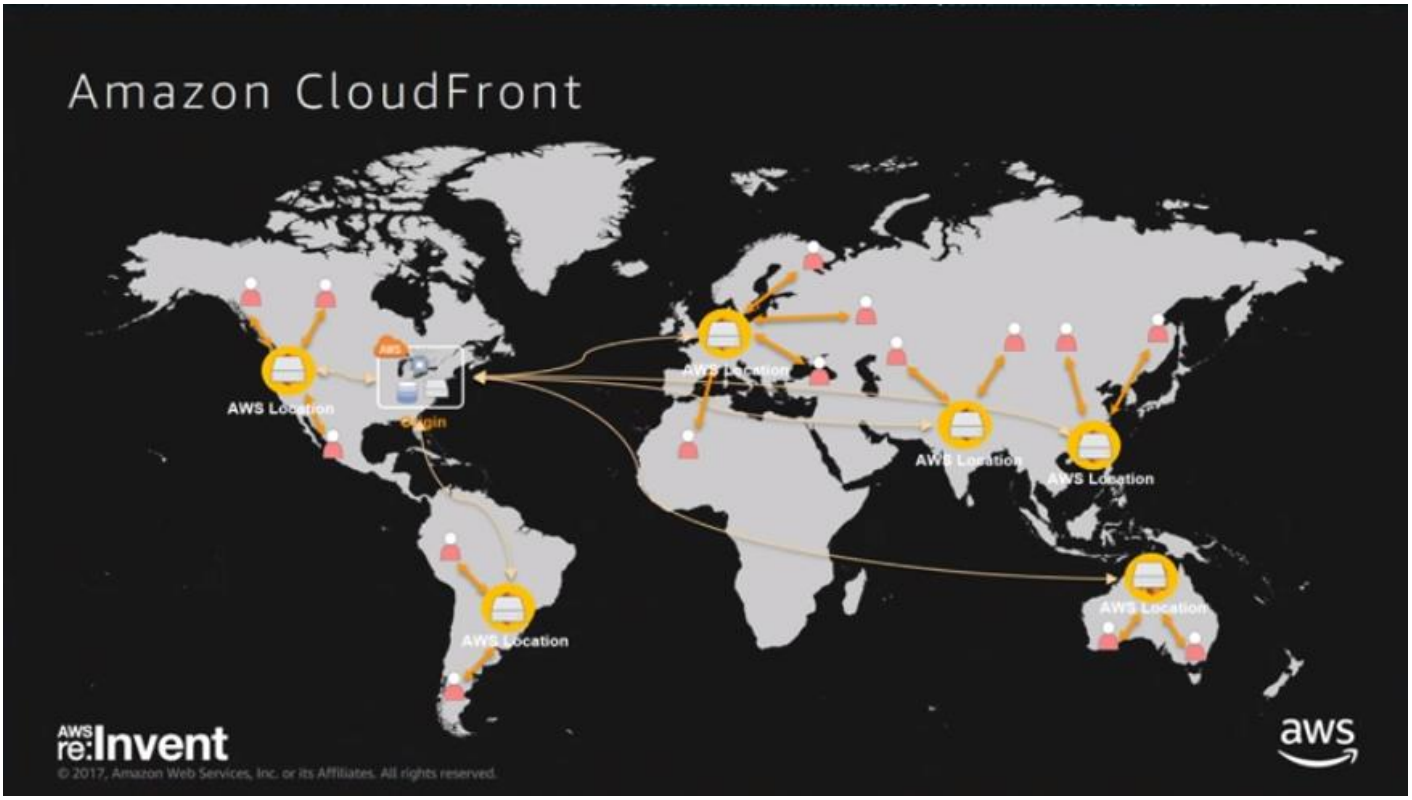
AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

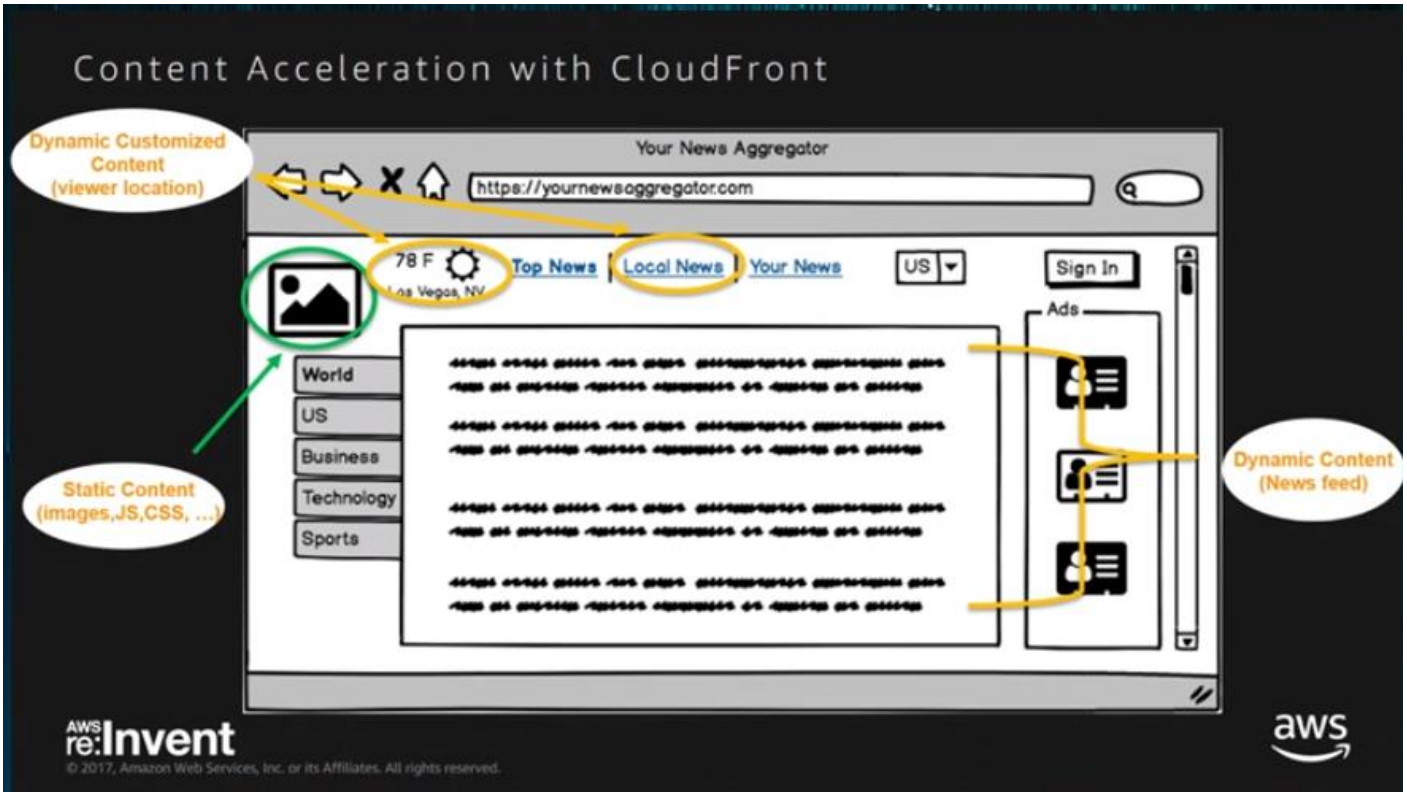
aws



CloudFront will redirect user requests to a CloudFront location that is closest to them from a latency standpoint using some latency-based algorithm.



You can now extend your cache to the edge to have the static contents closer to the user.



All the static contents are now being cached and closer to the viewers, the dynamic contents are also a little better just by introducing CloudFront because CloudFront has capabilities for optimizing dynamic content. We have moved some functionalities to the edge in CloudFront, mainly the storage of static assets.

AWS Lambda

No servers to provision or manage

Scales with usage

Never pay for idle

Built-in availability and fault tolerance

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Lambda@Edge

No servers to provision or manage

Scales with usage

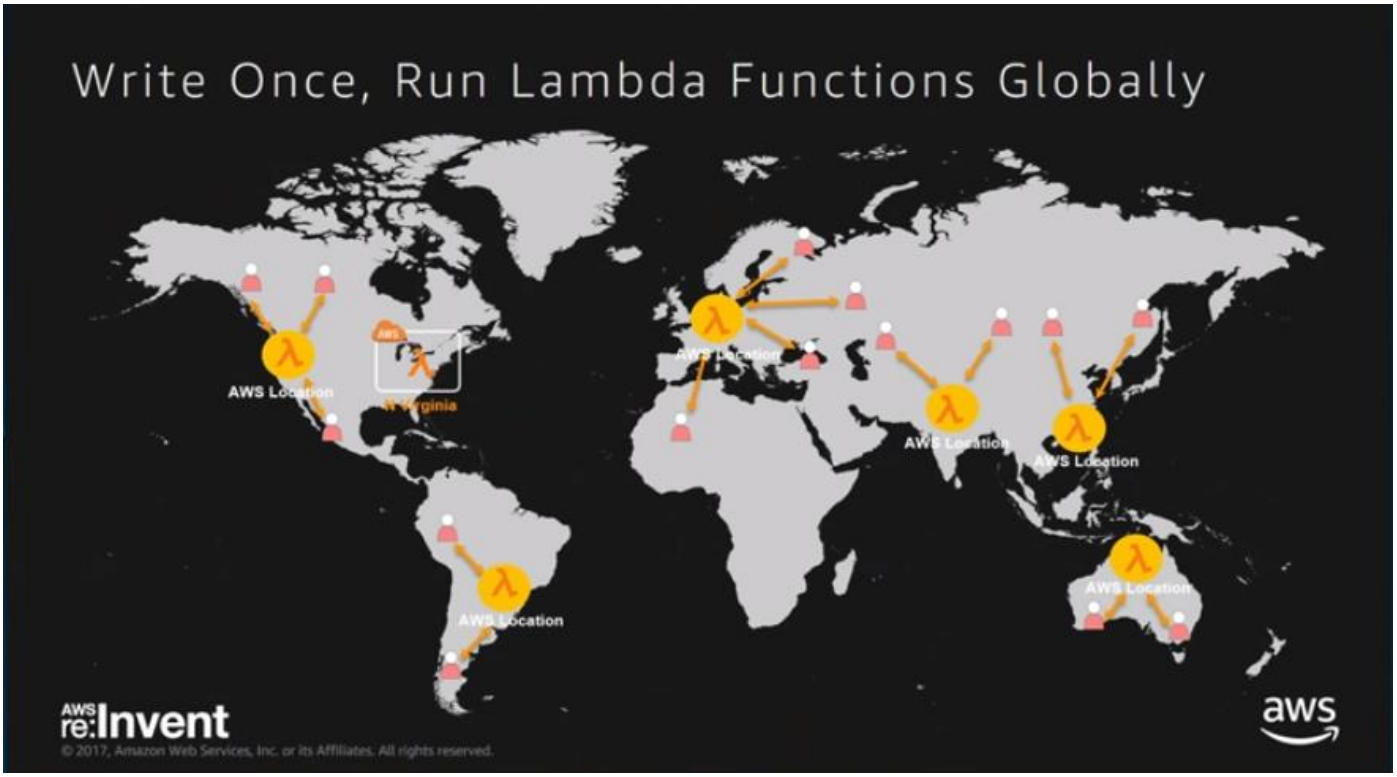
Never pay for idle

Built-in availability and fault tolerance

Globally distributed

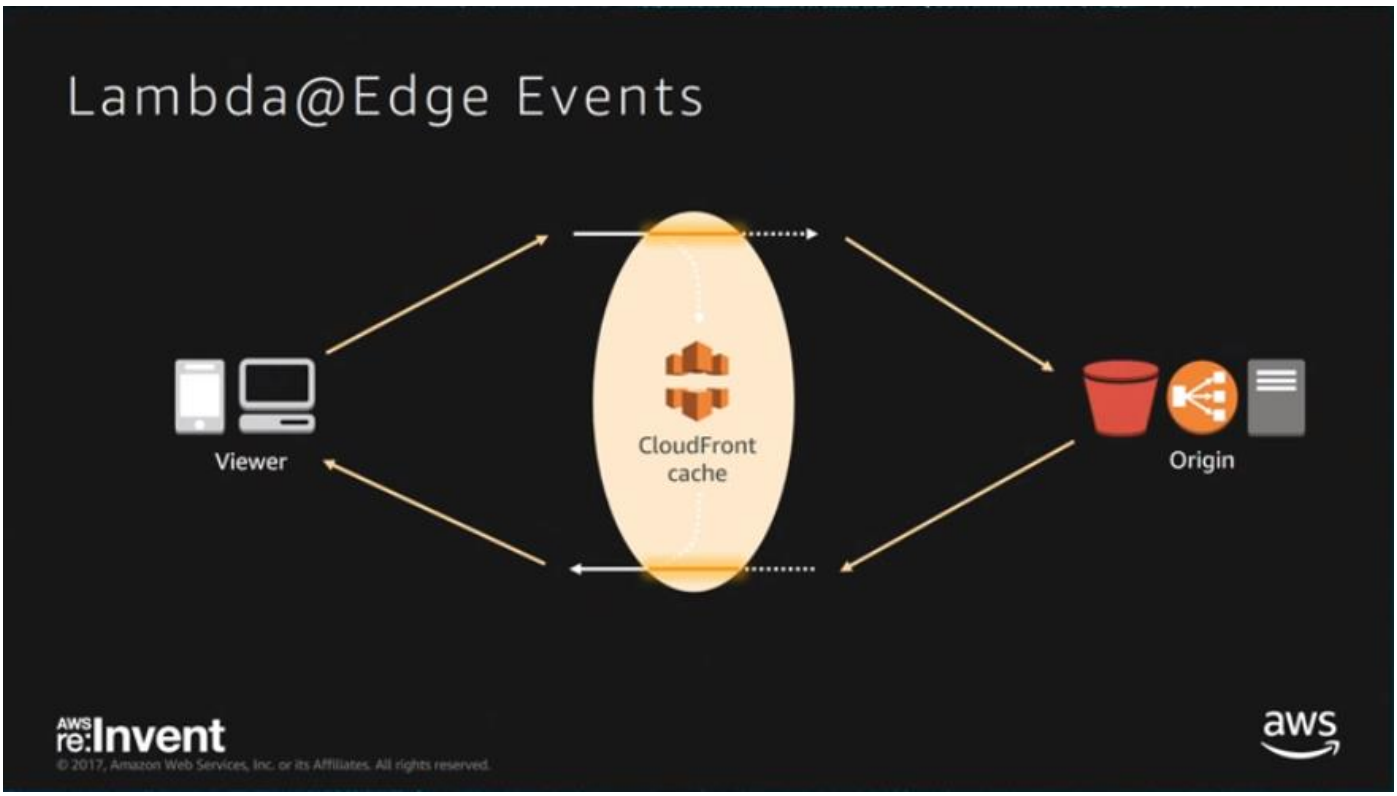
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

This is your current scenario with the static contents moved to the edge via CloudFront

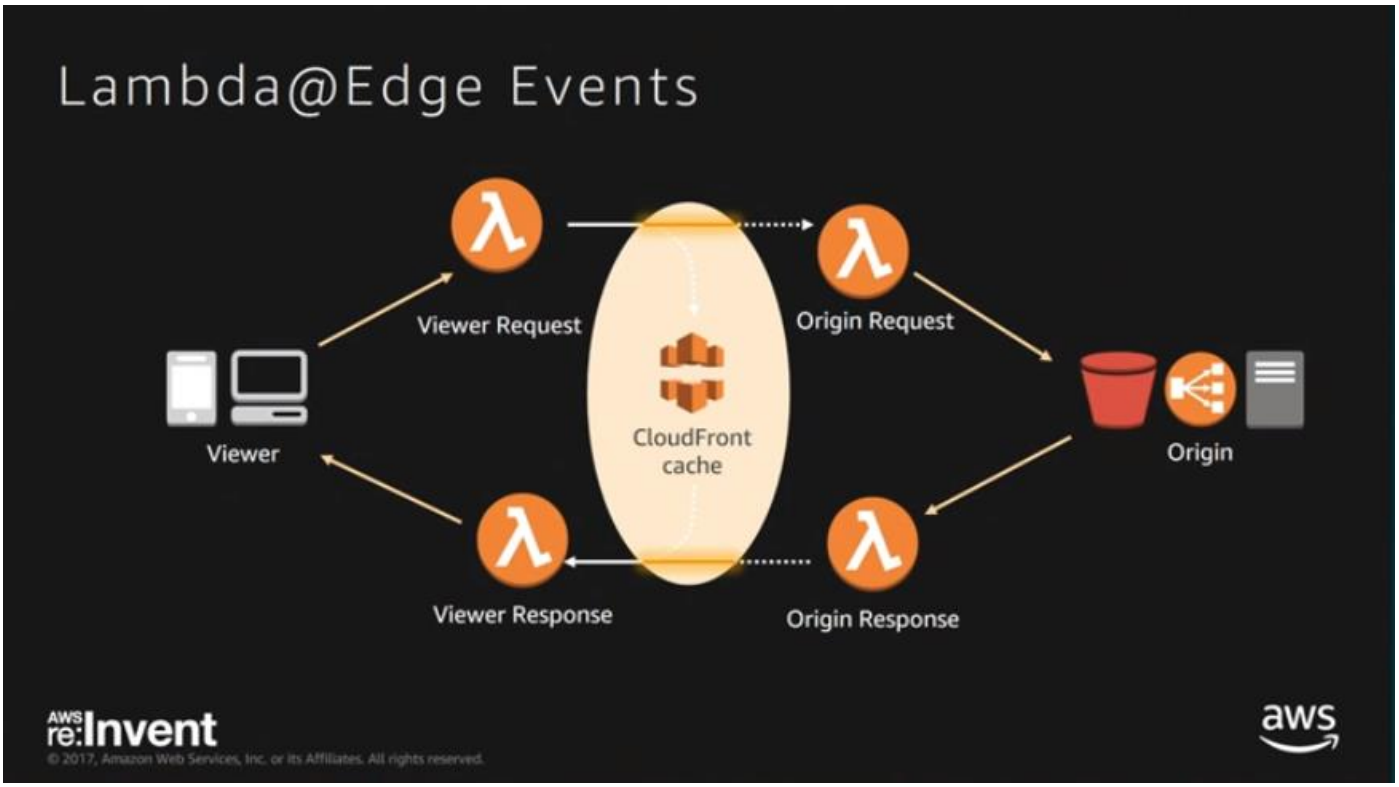


This means that the same way CloudFront is able to move storage to the edge locations, Lambda@Edge is also able to move the compute to the edge locations.

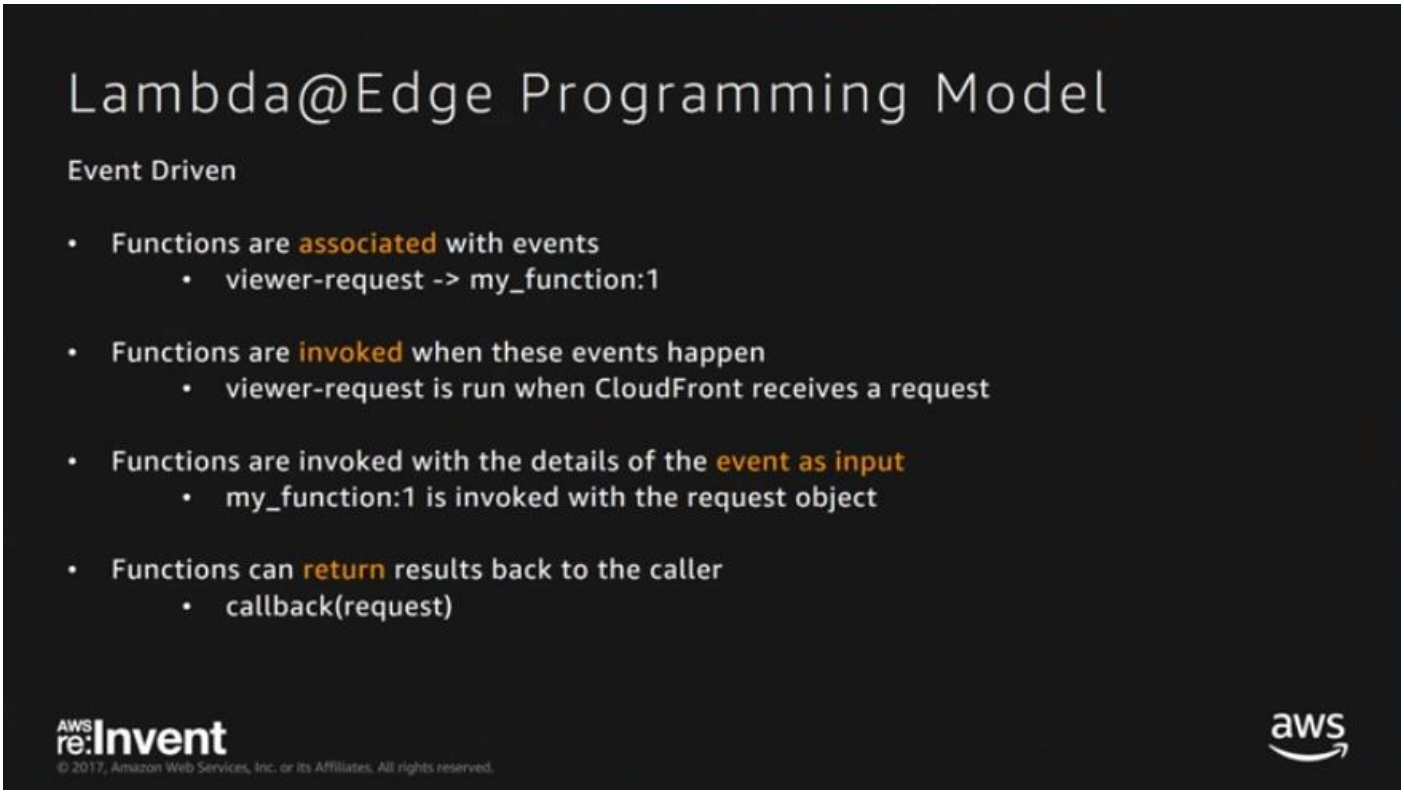
Let us now see how to move some of the remaining things not done at the edge to the edge using Lambda@Edge.



Let us see what happens in an edge location when CloudFront (CF) gets a request. The request lands at the CF cache which is like a big blob/giant cache, it the request is there it goes back to the caller, otherwise it goes on to the location to get it



Using Lambda@Edge, we can now modify properties of the Viewer requests right before they get to the CF cache itself. We can also modify requests missed from the cache to the origin. Likewise, we can modify responses before they get cached in CF and in their way back to the Viewer. Based on your use case, you can decide which of the 4 points is the best way to implement the properties modification logic you have.



Lambda@Edge Programming Model

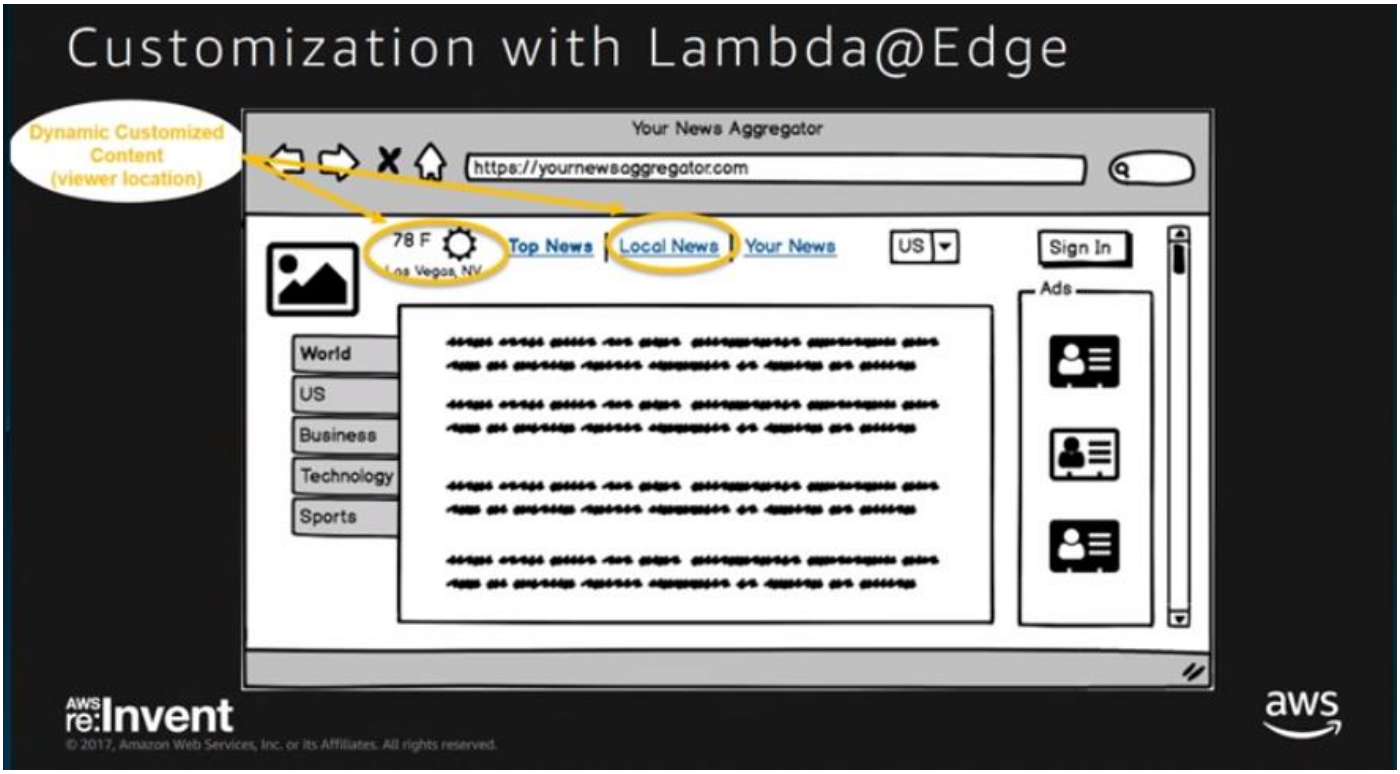
```
exports.handler = (event, context, callback) => {

    /* viewer-request and origin-request events
     * have the request as input */
    const request = event.Records[0].cf.request;

    /* viewer-response and origin-response events
     * have the response as input */
    /* const response = event.Records[0].cf.response; */

    /* Do the processing - say add a header */

    /* When I am done I let CloudFront what to do next */
    callback(null, request);
}
```

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

This is probably how this would look in code, the function typically takes 3 parameters, the event name, the context and the callback. You can access the request or response as above and do the manipulations before letting the requests go



L@E can help with the customizations we need to do in our scenario

Customization - Inputs

Non-personal artifacts (*customization*) – Request attributes

- Geography
  - Local weather, traffic updates, local/regional news, local events, and so on
- Device Type
  - Mobile vs. Desktop
  - QHD phone vs. 720p phone
- User Agent
  - Crawler vs. Actual user
- Referer

Personalization

- User's identity + Information related to that user

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

Customizations - Setup

With origin	Serverless
• Using CloudFront cache - URL, Header, and Query parameters	• Using CloudFront cache - URL, Header, and Query parameters
• Using origin selection	• Generating content at the edge

Your particular setup will also add to the way you customize things using L@E. Using the CF cache when you have contents that is highly cacheable like 2 images versions for a mobile device or a desktop device. Using the ***origin selection*** option means that you can dynamically change the origin that a request gets sent to right before it leaves the CF cache after a cache fail using the ***originSelect*** event. Generating content at the edge is a serverless case where you generate the entire response inside one of the events typically a Viewer request event or an origin request event, and you don't reach out to the origin at all.

Customization - Examples

URL modification

- Serve different content based on viewer's country
  - E.g.: "/index.html" -> "/au/index.html" or "/uk/index.html" or "/de/index.html")
- Serve different web assets from same distribution
  - E.g.: Use Host header to rewrite URL.
  - "/index.html" + "Host: foo.com"-> URL: "/foo_com/index.html"
  - "/index.html" + "Host: bar.com" -> URL: "/bar_com/index.html"
- Serve different content by device type
  - E.g.: Use CloudFront-Is-Mobile-Viewer -> URL: "/mobile/index.html" or "/desktop/index.html"

This is a CF cache based example. When all the client calls out for the same object, say the index.html request, then internally on the server side it includes both L@E and CF. You decide which combination to serve to the user by user country.

Customization - Examples

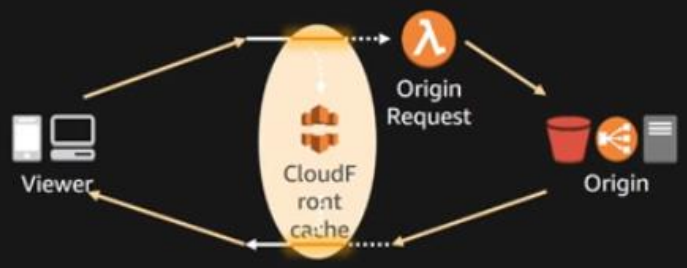
Header Add/Modify/Delete

- Security Headers
  - Ex: Insert HSTS, CORS headers for all responses
- Setting Cache-control headers for S3 objects
  - Ex: Be able to do it for a larger group at once
  - /images/* or /*.jpg
- Expose the true Client IP to the origin
  - Ex: copy X-Forwarded-For to a header (True-Client-IP) that the origin understands

This is for both request and response side. You can include security headers right before the response or request get generated, modify the cache control headers, or even expose true client IPs as an additional header.

Example – Customize Based on Device Type

All clients request https://example.com/index.html



- CloudFront-Is-Mobile-Viewer: true -> https://example.com/mobile/index.html
- Every one else is unchanged or gets sent to https://example.com/desktop/index.html
- Content is highly cachable – so run logic in origin-request event
- To correctly cache the content add CloudFront-Is-Mobile-Viewer to Forward headers

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

you can modify based on the device type,

Example – Customize Based on Device Type

```
exports.handler = (event, context, callback) => {
    const request = event.Records[0].cf.request;
    const headers = request.headers;

    /* Direct to a different part of the website based on
     * the device type */
    const desktopPath = '/dektop';
    Const mobilePath = '/mobile';

    if (headers['cloudfront-is-desktop-viewer'] && headers['cloudfront-is-desktop-viewer'][0].value === 'true') {
        request.uri = desktopPath + request.uri;
    } else if (headers['cloudfront-is-mobile-viewer'] && headers['cloudfront-is-mobile-viewer'][0].value === 'true') {
        request.uri = mobilePath + request.uri;
    }

    console.log(`Request uri set to "${request.uri}"`);
    callback(null, request);
}
```

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

The code in blue are things we have talked about earlier; the orange text is about looking at the headers and adding the right prefix before going to the origin

Customization Using Origin Selection

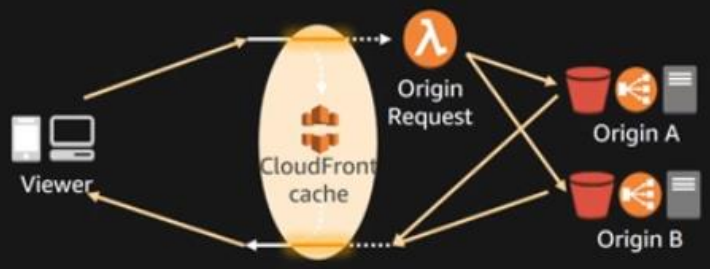
- Multiple origin setup
  - Latency: Talk to the origin closest to the viewer
  - Load balance across origins
- Controlled rollout of changes at origin
  - A/B Testing of new features
  - Blue/Green origin deploys
- Migrating between origins
  - Including on-premise to cloud
- Search Engine Optimization
  - Serve human and web crawler traffic from separate origins

AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

You can detect whether the caller is a crawler and just send them to a rendered version of the resource

Origin Selection – A/B Testing

Example: You want to test a new feature. It is only deployed to one of your origins



In the function:

1. Check to see if this is a active session. (Say, using a cookie.)
2. For active sessions, set the origin based on the value in the cookie
3. For a new session, decide whether to show A or B variant. And set the origin accordingly

AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

Origin Selection

- Origin is present as part of request
  - `event.Records[0].cf.request.origin`
- Modified Origin should also be part of the request structure returned

```
"s3": {
  "domainName": "green-bucket.s3.amazonaws.com",
  "path": "/originPath",
  "authMethod": "origin-access-identity",
  "region": "us-east-1",
  "customHeaders": {
      "my-custom-origin-header": [
          {
            "key": "My-Custom-Origin-Header",
            "value": "test-value"
          }
      ]
  }
}
```

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

This is a way to do this within the origin request event in code, request.origin is where you have the details of the origin request you want to manipulate

Example – A/B Testing

```
exports.handler = (event, context, callback) => {
    const request = event.Records[0].cf.request;

    desiredOrigin = decide(request);

    /* Set custom origin fields*/
    request.origin = {
                custom: {
                    domainName: desiredOrigin,
                    port: 443,
                    protocol: 'https',
                }
     };
     request.headers['host'] = [{ key: 'host',
                                  value: desiredOrigin }];

     callback(null, request);
};
```

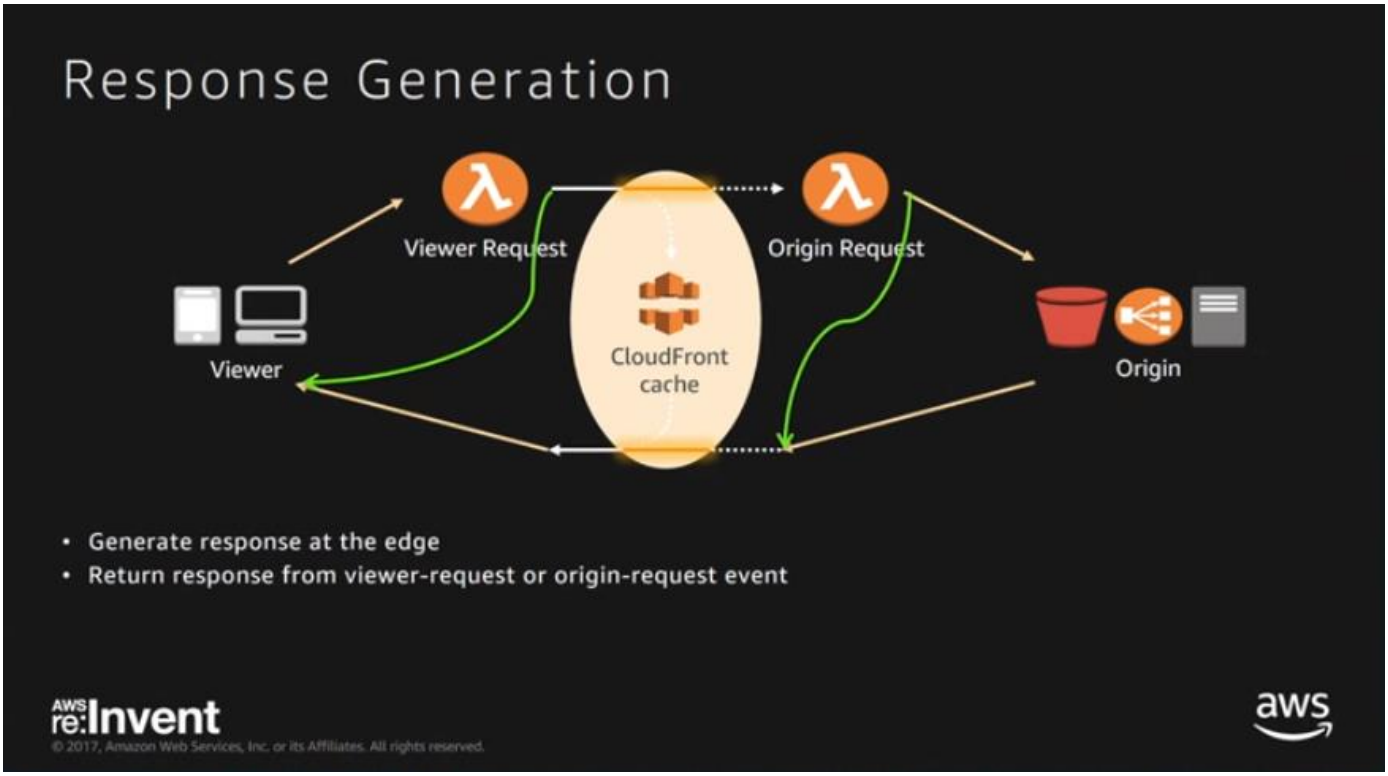
```
function decide(request) {
    if (request.headers['my-session-cookie']) {
        cookie = request.headers['my-session-
cookie'].value;
        return decodeOrigin(cookie);
    } else {
        return chooseOrigin(request);
    }
};
```

AWS re:Invent

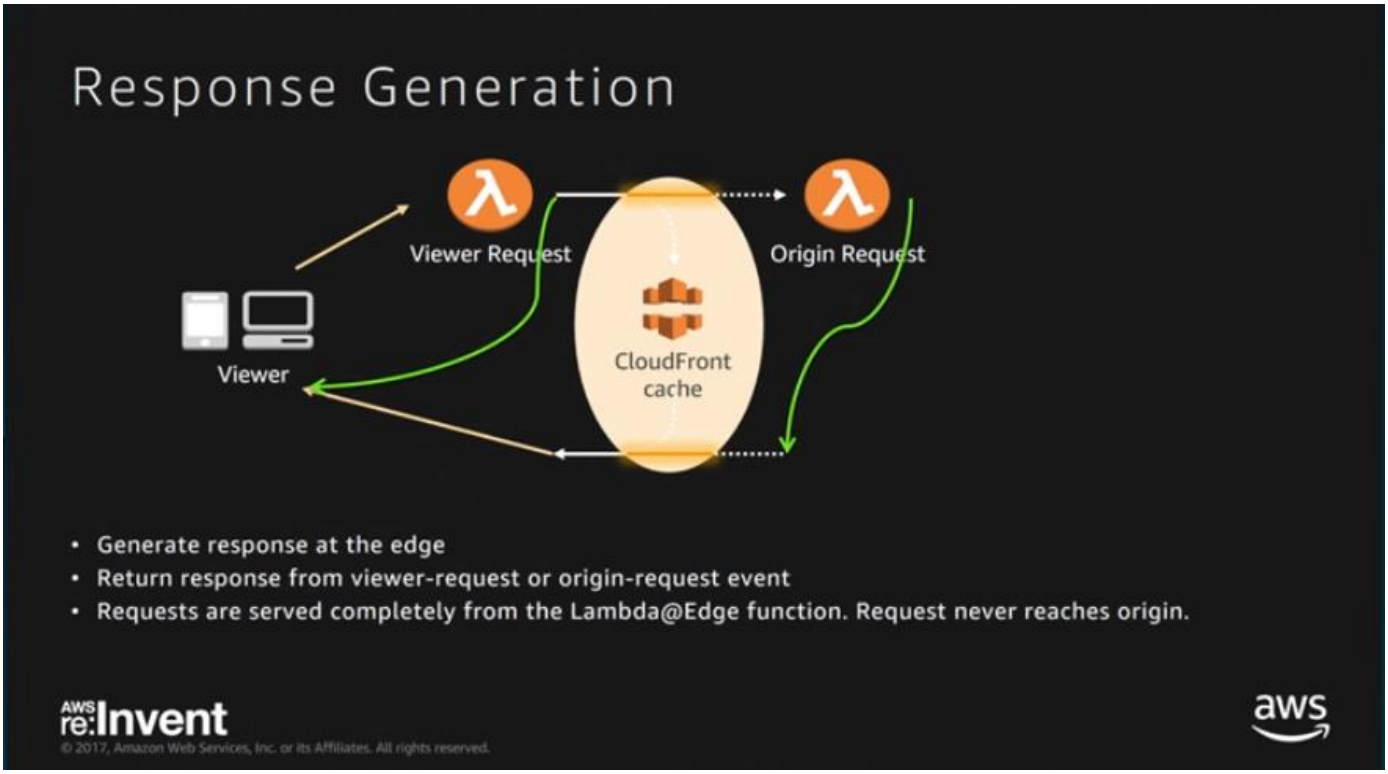
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

This is an example of A/B testing where we have the ***decide() logic***, the domain name is the only property we want to change within the cf.request object



For response generation at the edge,



Response Generation

- Response generated at the edge location
- Can be done in viewer-request or origin-request events
- Generate a complete HTTP response in the function
- Return this response as part of callback

```
callback(null, response);
```

- Request never reaches out to the origin

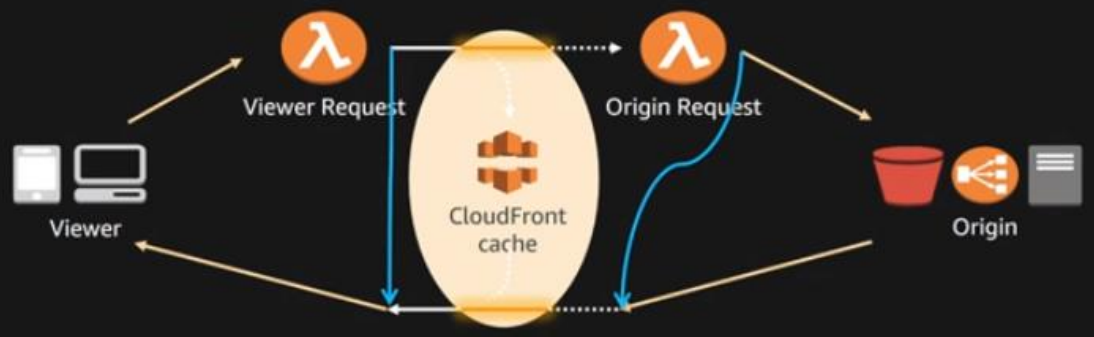
AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

Customization Using Response Generation

- Without body (only headers)
  - Redirect
  - May use external source for decision making
- With body
  - Constituent elements are highly cacheable
  - Generated response may be unique to the viewer
  - Using input from the request
    - Ex: Weather App – landing page
      - List of cities varies by user. Request has list of cities.
      - Each city's weather is highly cacheable
  - Using user-specific data (not part of the request)
    - Ex: My Prime Music page
      - Shows list of albums I own
      - Recommended playlists
      - Input often comes from a datastore

AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

Example - Redirect



- Viewer-request event and Origin-request event
- Redirect to a login page for non-authenticated users
    - In viewer-request function validate the cookie
    - If cookie is expired or not present, redirect to login page

AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

There is no more origin in this case anymore

Example - Redirect

```
'use strict';

exports.handler = (event, context, callback) => {
    /*
     * Generate HTTP redirect response with 302 status code and Location header.
     */
    const response = {
        status: '302',
        statusDescription: 'Found',
        headers: {
            location: [{
                key: 'Location',
                value: 'http://mydomain.com/login.html',
            }],
        },
    };
    callback(null, response);
};
```

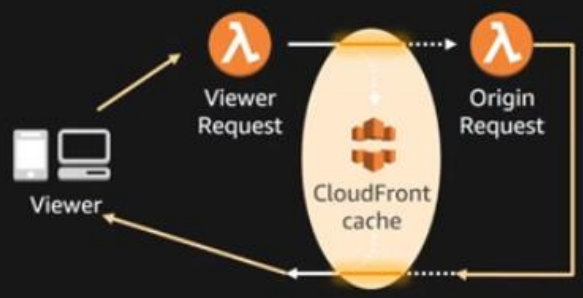
AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

Note that you don’t set a body here

Example - Content Aggregation

Example: Weather App Landing page
Client: Each user has a set of cities.

http://example.com/weather?cities=Seattle;NYC



Function:
- Parses the URL
- Fetches the relevant data
- Sends the aggregated response to the client (app)

AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

AWS has a sample code or blueprint for this function on the website

Example - Content Aggregation

```
function getCityForecast(request) {
    return new Promise((resolve, reject) => {
        https.get(request.uri, (response) => {
            let content = '';
            response.setEncoding('utf8');
            response.on('data', (chunk) => { content += chunk; });
            response.on('end', () => resolve({ city: request.city, forecast: content }));
        }).on('error', e => reject(e));
    });
}

const uriSplit = uri.split('/');
const cities = uriSplit[2].split(':');

const forecasts = [];
cities.forEach((cityName) => {
    const cityForecastUri = citiesBaseUri + cityName;
    forecasts.push({ city: cityName, uri: cityForecastUri })
});
```

AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

There are 2 parts, how you fetch the content and then how you query for content

Example - Content Aggregation

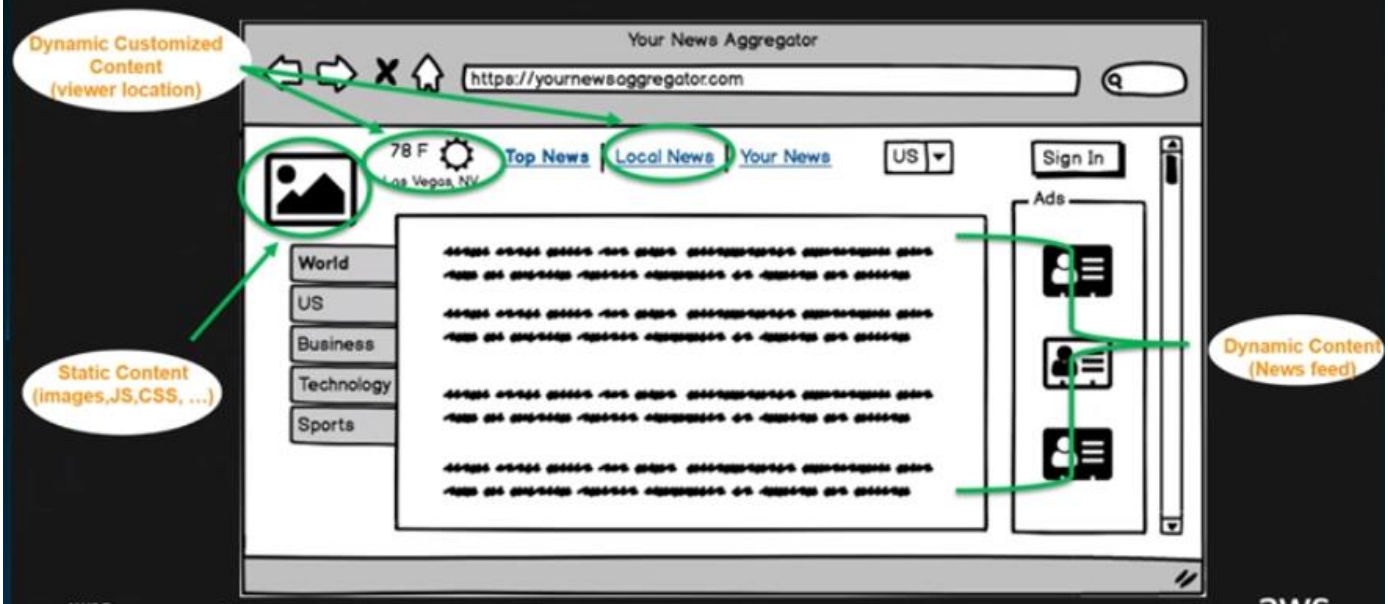
```
Promise.all(forecasts.map(getCityForecast)).then((ret) => {
        console.log('Aggregating the responses:\n', ret);
        const response = {
            status: '200',  /* Status signals this is a generated response */
            statusDescription: 'OK',
            headers: {
                'content-type': [{
                    key: 'Content-Type',
                    value: 'application/json',
                }],
                'content-encoding': [{
                    key: 'Content-Encoding',
                    value: 'UTF-8',
                }],
            },
            body: JSON.stringify(ret, null, '\t'),
        };

        console.log('Generated response: ', JSON.stringify(response));
        callback(null, response);
```

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

What Else Can You Move to the Edge?

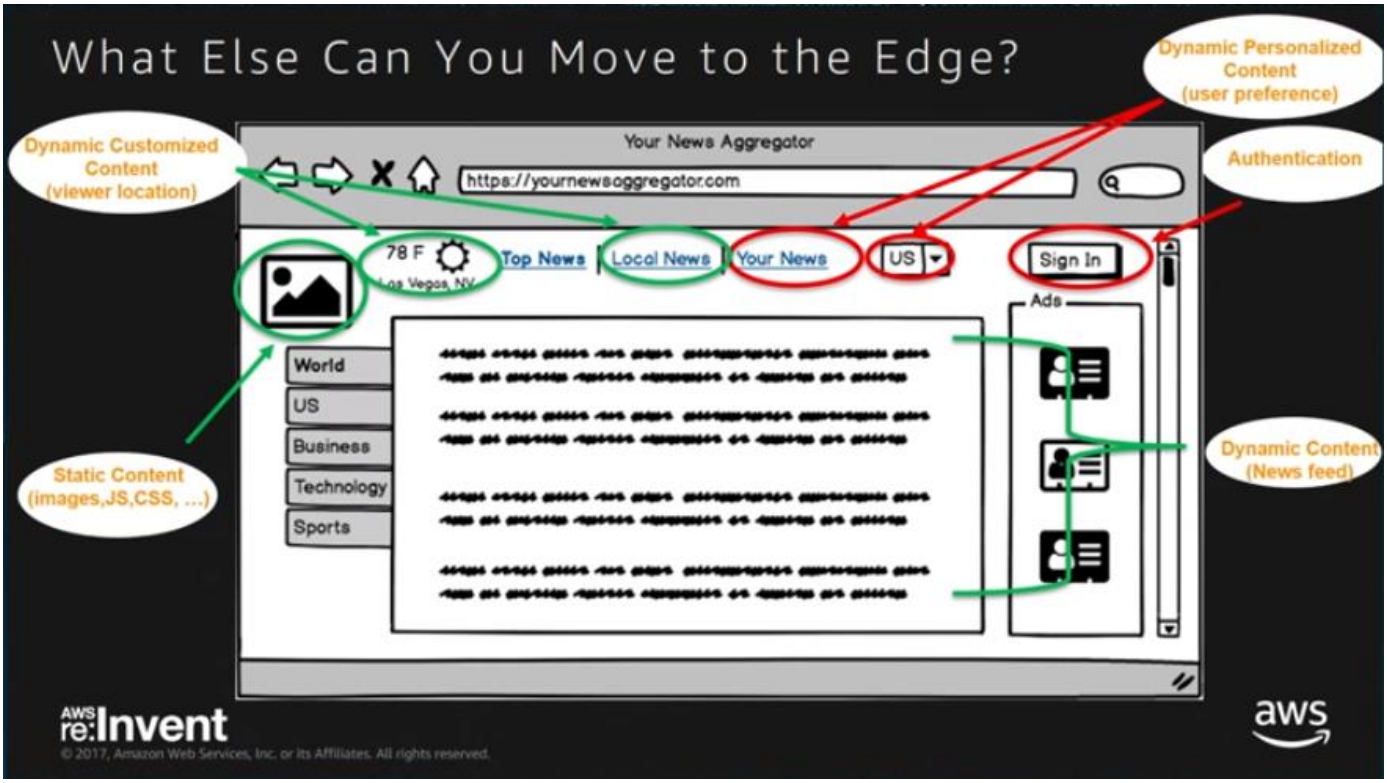


AWS re:Invent

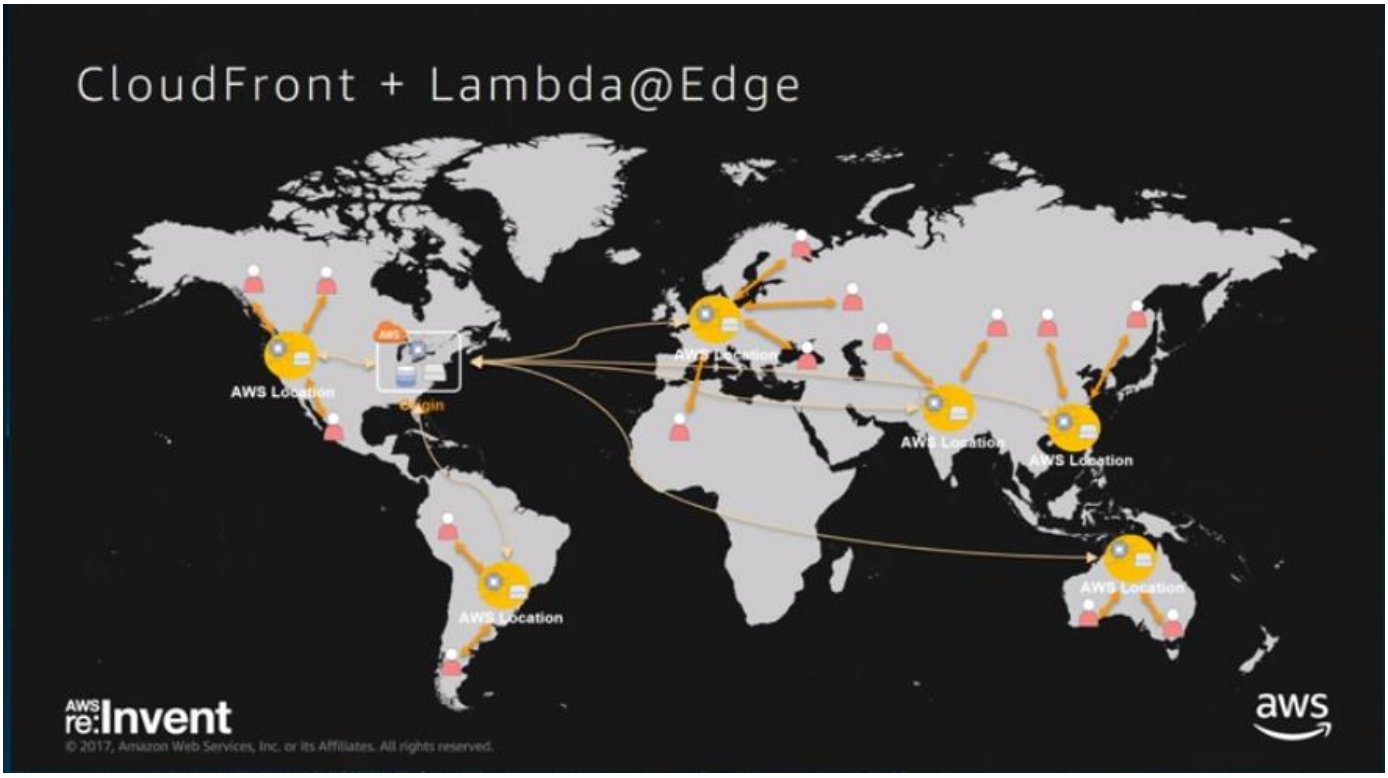
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

Some of the dynamic contents can now be generated at the edge and use the combination of the highly cacheable items to generate some unique data for that user.



Now let us get into how can we take advantage of knowing who the user is, to knowing more about that particular user



We have moved compute closer to the user for stateless compute.

Personalization

- Identity: Authentication – *know who the customer is*
- Information: *knowledge about the customer*
    - *Preferences, Browsing/buying history …*
- Combine Identity and Information to create content personalized to that user

Authentication

A mechanism to decide whether to serve this request or not:

- Authentication using Trusted Signers (CloudFront feature)
- Authentication - validated at the edge in the function
- Authentication using remote servers

Accessing Data from Functions

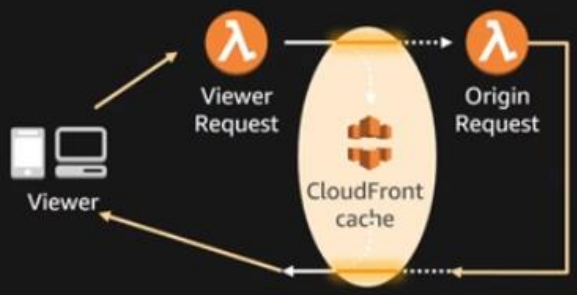
- Bundle with the code
    - Easy to use/access. Fast
    - Read only. Limited by size of deployment package. Change requires a deploy.
- CloudFront cache
    - Close to the function (often in the same location). Periodically refreshed (TTL).
    - Read only
- AWS DB offering (DDB, Aurora)
    - Read-write
    - Often in one region farther away from the function
- Amazon DynamoDB Global Tables
    - Multi-Region, Multi-Master tables
    - Read-write, closer to functions

Content Generation - Personalization

Example: We want to recommend new "items" when a user logs in

Client:
- https://example.com/index.html
- User is logged in

Function:
- Fetch user's history and preference from DDB
- Generate response using the fetched data as input
- Ex:
  - News articles based on my reading history
  - Music or Movie recommendations based on my listening/watching
  - Recommendations based on people you follow



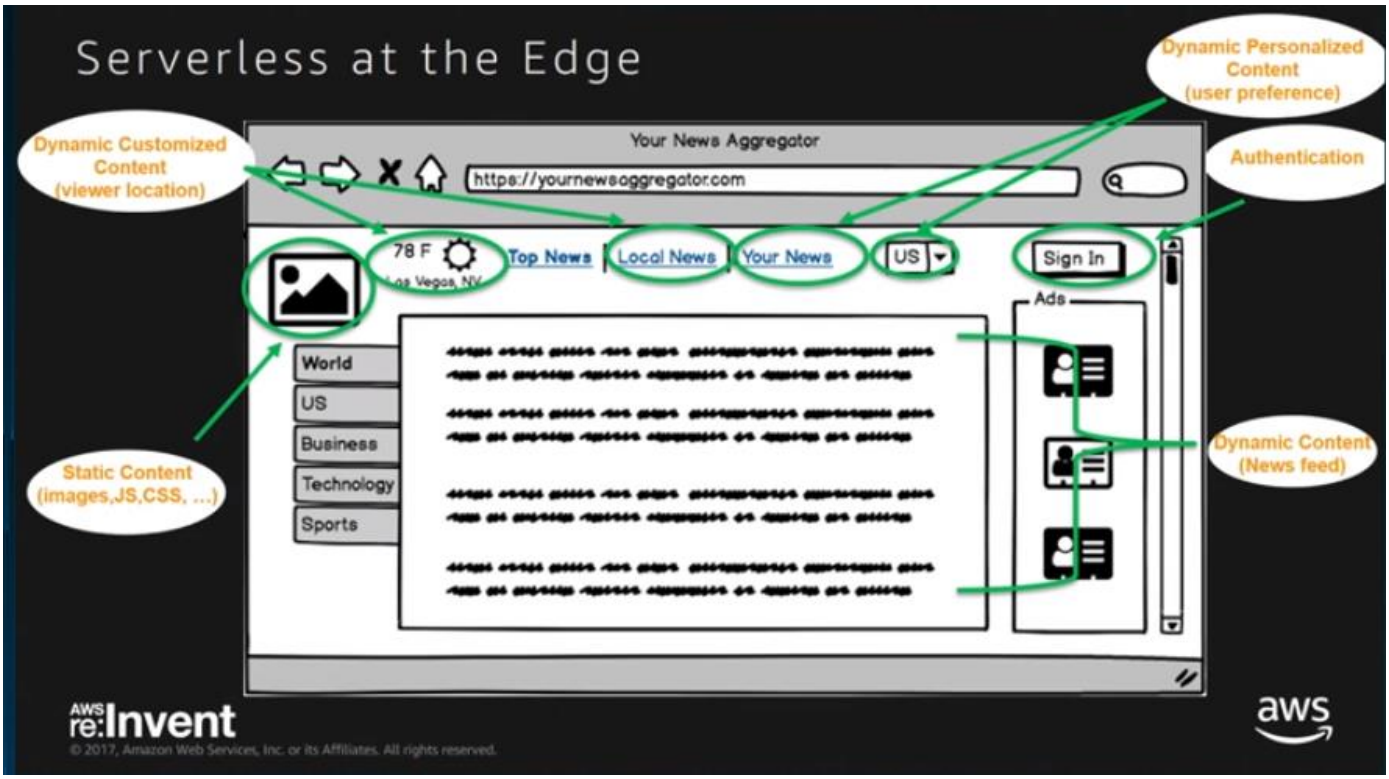
AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

We will be accessing DynamoDB to get content from within the lambda function

Content Generation - Recommendations

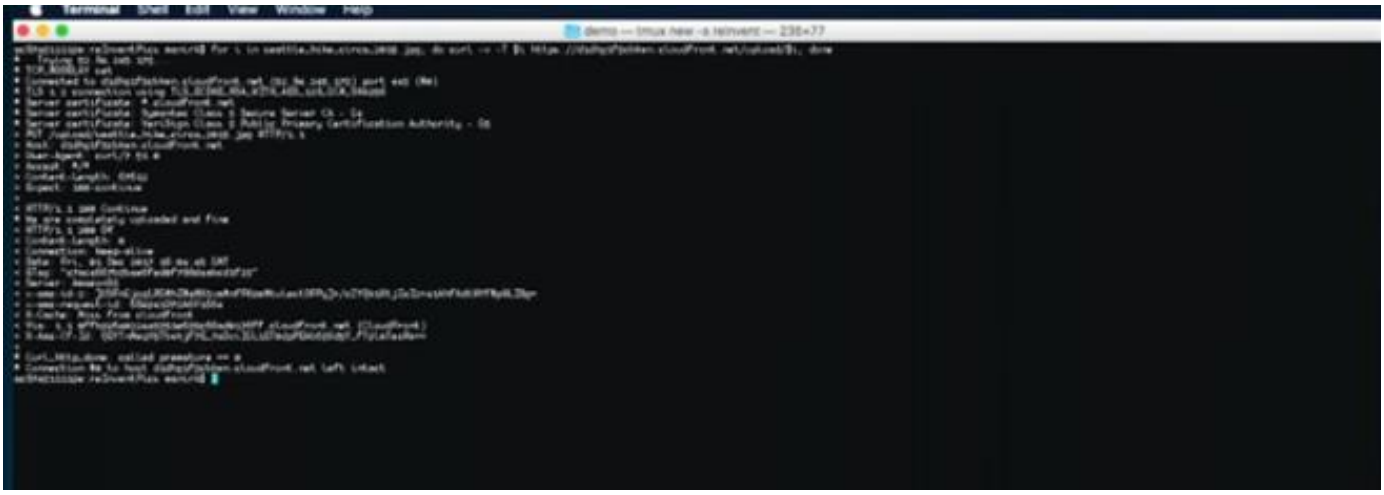
```
/* Access DDB */
var aws = require('aws-sdk');
var ddb = new aws.DynamoDB().DocumentClient({apiversion: '2012-10-08', region: 'us-east-1'});;

/* Get the item based on the user */
function getItems(request, context, callback) {
    const params = {
        TableName: "recommendations",
        Key: { "User": { "S": user } }
    };
    ddb.getItem(params, function(err, data) {
        if (err) {
            console.log(err, err.stack);
            generateErrorResponse(request, callback);
        } else {
            console.log("Received output from ddb: " + data)
            generateResponse(request, data, callback);
        }
    });
}
```

AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

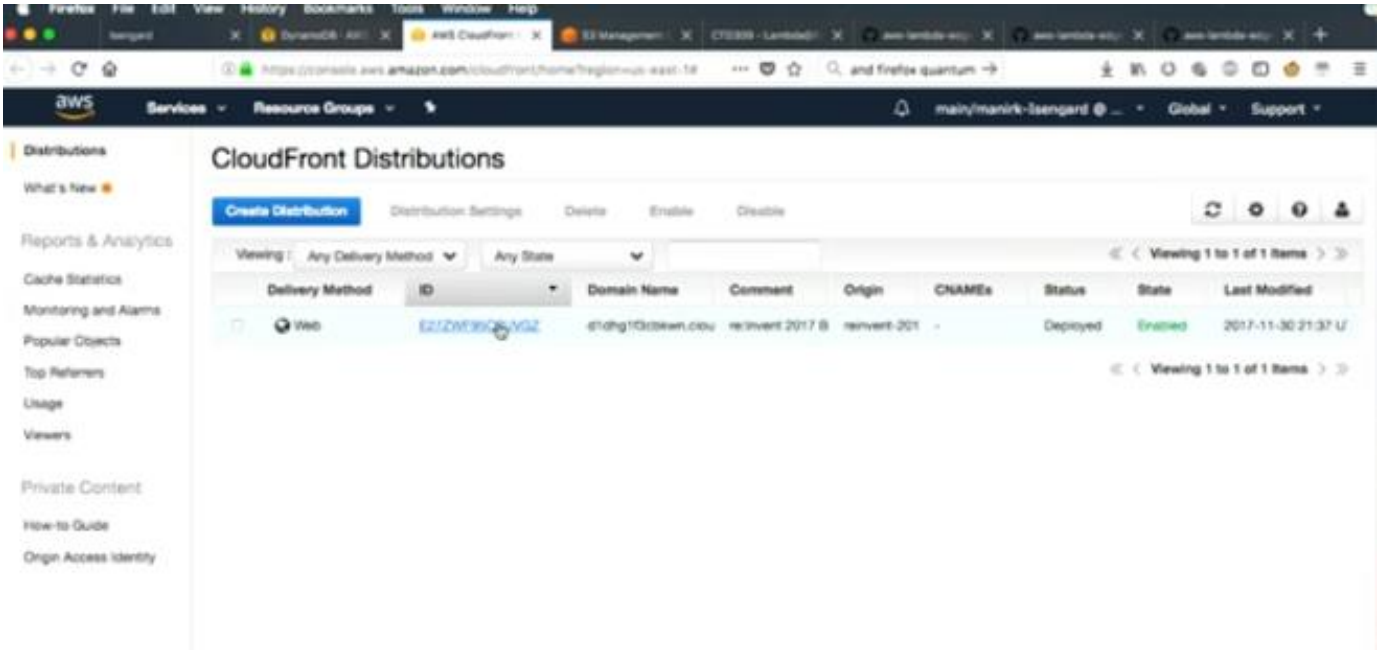


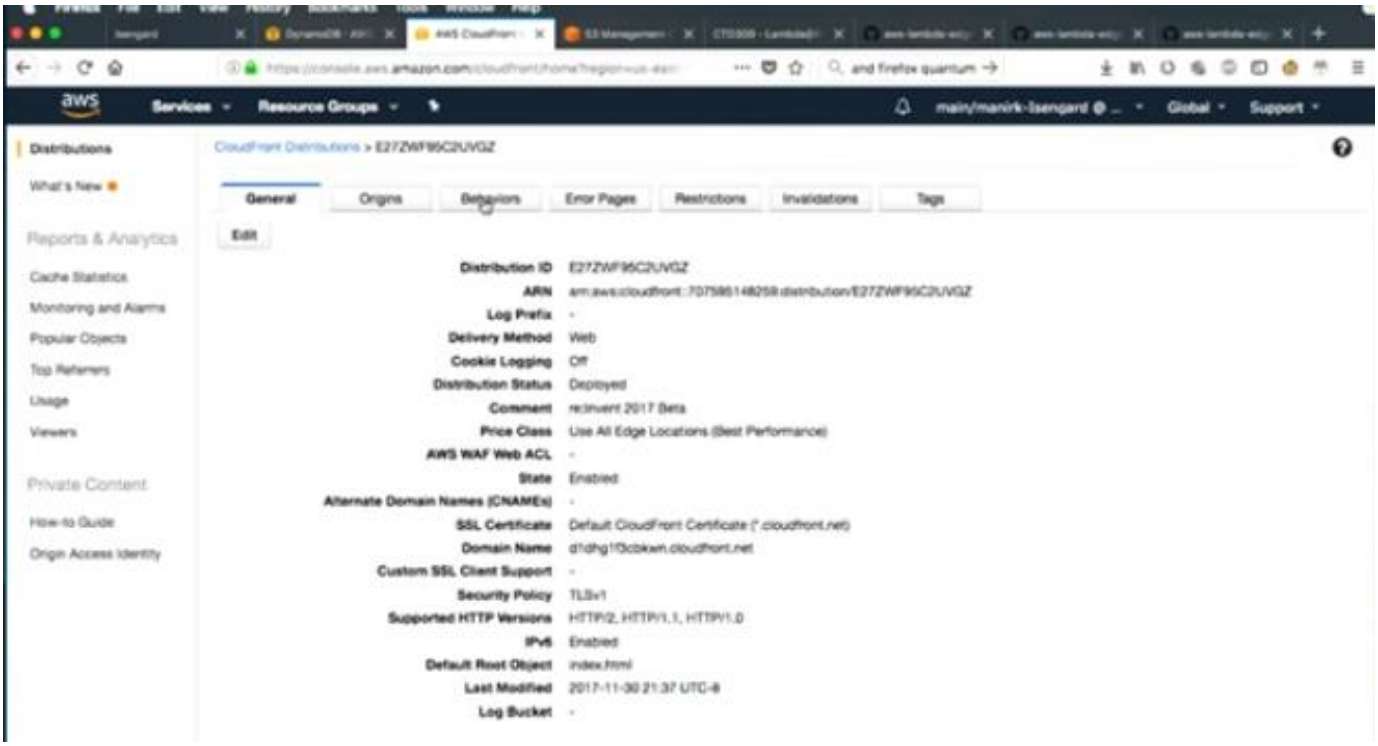
Now we have a fully serverless website application using Lambda@Edge

We have a way for people to share come photos using a serverless website that uses S3, CloudFront, Lambda@edge and DynamoDB. This index.html is served from the origin request event

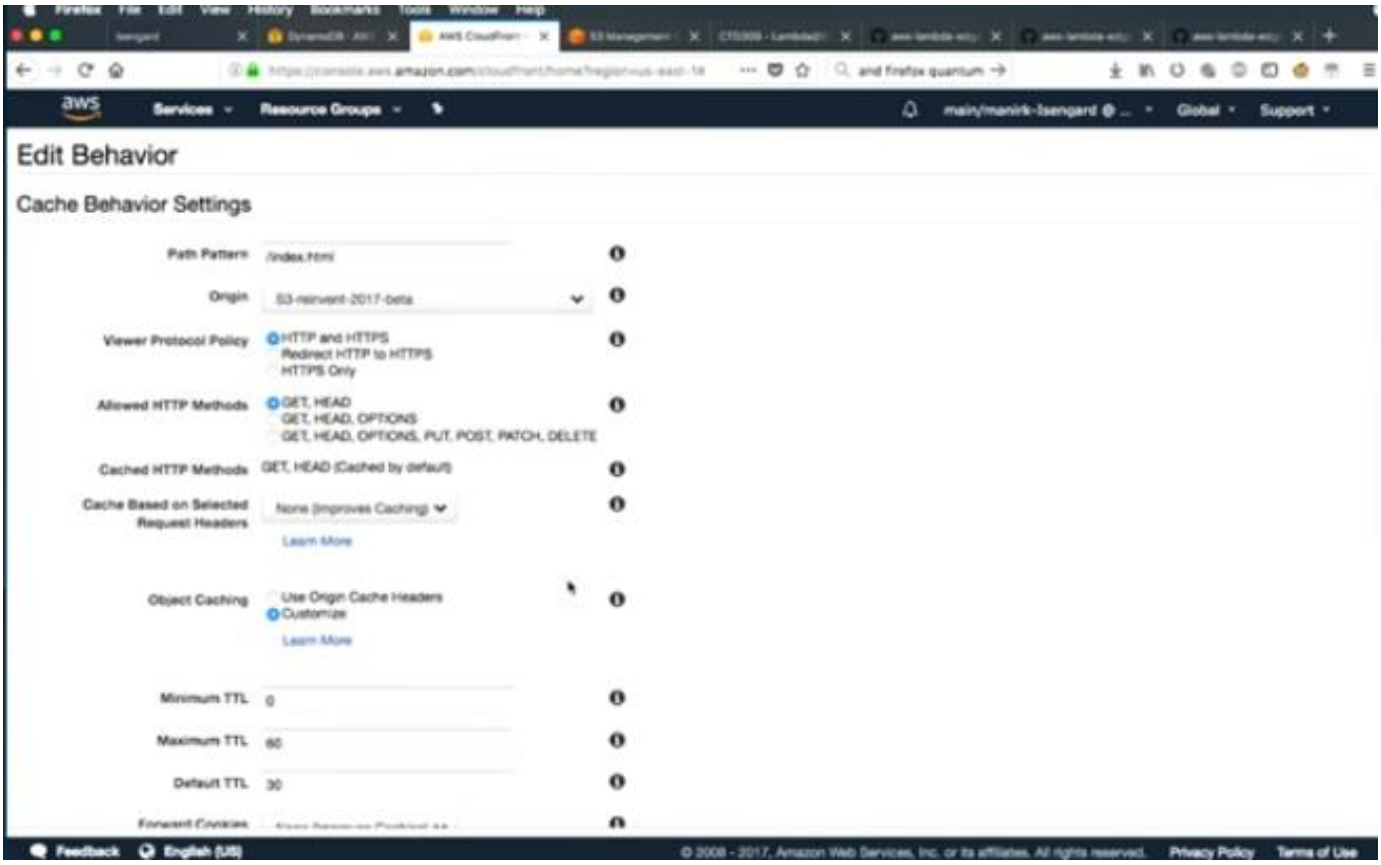


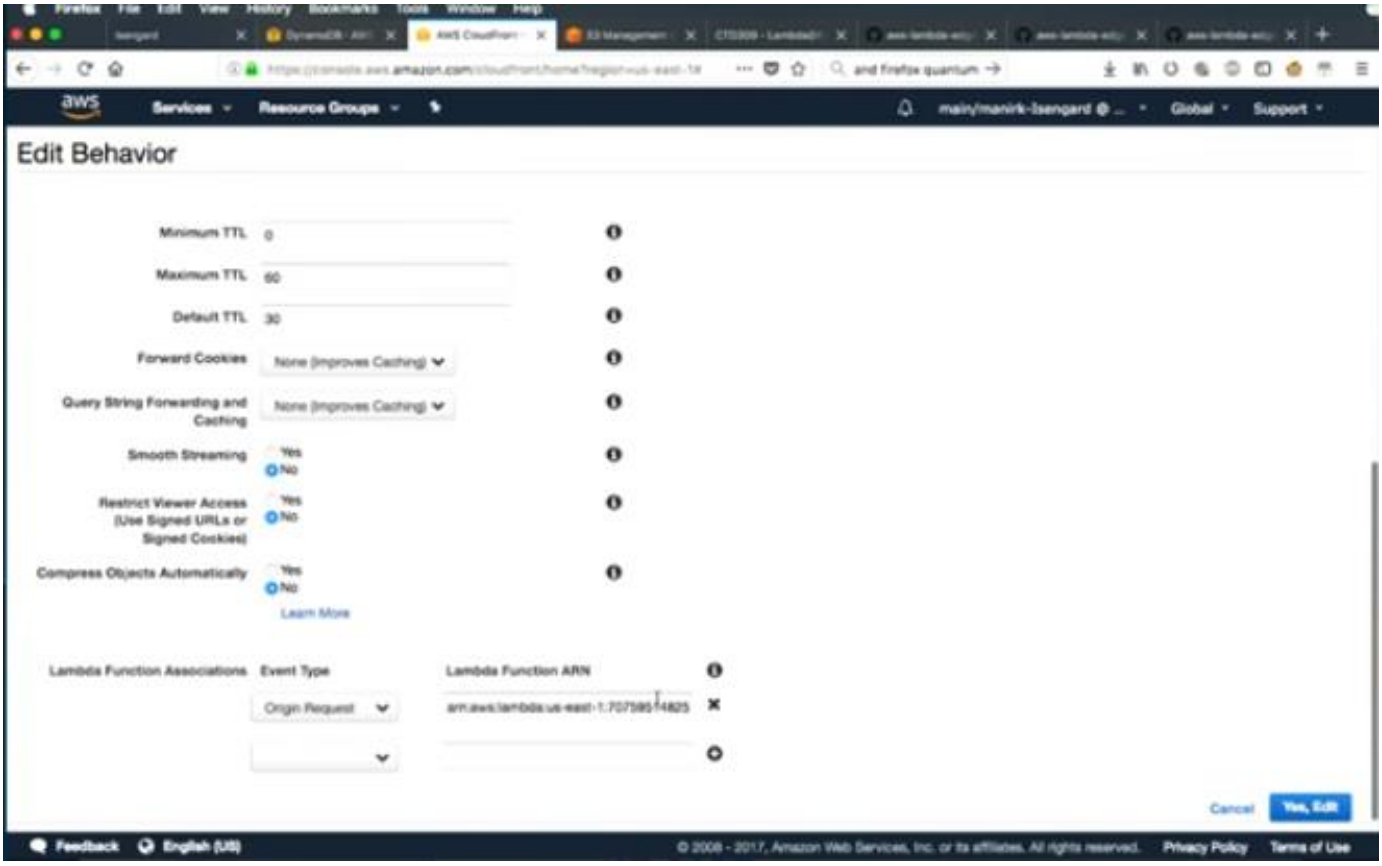
We can upload more images using code





Our CF has 3 cache behaviors in it, one for the index.html page, one for the API that we are using to upload content, and the default behavior that we are using to serve static contents.





For the index.html behavior, we have it associated with a lambda function for the origin request function that does the page rendering.

AWS Lambda

Dashboard

Functions

Functions (13)

Actions | Create function

Filter by tags and attributes or search by keyword

Function name	Description	Runtime	Code size	Last Modified
reinvent-demo-index-html-generator	Blueprint for generating a redirect response based on the viewer country. Triggered by an origin-request. Implemented in Node.js.	Node.js 6.10	1.1 kB	12 hours ago
reinvent-2017-pandademo-origin-request	Blueprint for generating aggregated response from multiple remote calls on origin-request trigger implemented in Node.js.	Node.js 6.10	1.3 kB	last month
origin-request-with-data-response-generation	Blueprint for generating aggregated response from multiple remote calls on origin-request trigger implemented in Node.js.	Node.js 6.10	835 bytes	4 days ago
origin-request-upload	Blueprint for generating a response from viewer-request trigger implemented in Node.js.	Node.js 6.10	756 bytes	18 hours ago
testingPolicy	test function to verify permissions and policy	Node.js 6.10	216 bytes	4 months ago
beFunction	Blueprint for returning HTTP redirect implemented in Node.js.	Node.js 6.10	380 bytes	3 months ago
redirect-demo-origin-request	Blueprint for returning HTTP redirect implemented in Node.js.	Node.js 6.10	380 bytes	6 days ago
reinvent-2017-demo-origin-request	Blueprint for generating aggregated response from multiple remote calls on origin-request trigger implemented in Node.js.	Node.js 6.10	1.3 kB	2 months ago
origin-request-no-data-response-generation	Blueprint for generating aggregated response from multiple remote calls on origin-request trigger implemented in Node.js.	Node.js 6.10	1.1 kB	5 days ago
myTestMonkeyPatchFn		Node.js 6.10	216 bytes	6 months ago

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

reinvent-demo-index-html-generator:7

Version: 7 | Actions | indexhtml | Test

Configuration | Monitoring

Add triggers

Click on a trigger from the list below to add it to your function

API Gateway

AWS IoT

Alexa Skills Kit

Alexa Smart Home

CloudFront

CloudWatch Events

CloudWatch Logs

CodeCommit

Cognito Sync Trigger

DynamoDB

Kinesis

S3

reinvent-demo-index-html-generator:7

CloudFront

Add triggers from the list on the left

AWS Lambda

Amazon DynamoDB

Data Pipeline

Amazon CloudWatch Logs

Amazon EC2

Identity And Access Management

Amazon SNS

Feedback English (US) © 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

reinvent-demo-index-html-generator:7

Node.js 6.10

index.handler

index.js

```
1  'use strict';
2
3  const AWS = require('aws-sdk');
4  AWS.config.update({region: 'us-east-1'});
5  const ddb = new AWS.DynamoDB({apiVersion: '2012-10-08'});
6
7  var htmlHeader = `
8  <!DOCTYPE html>
9  <html>
10 <head>
11     <meta charset="utf-8">
12     <title>CTD309 - LambdaEdge Demo</title>
13     <link rel="stylesheet" href="/style/style.css">
14 </head>
15 <body>
16
17 <p>Recent images</p>`;
18
19 var htmlFooter = `
20 </body>
21 </html>`;
22
23
24 function generateImageMatrix(data) {
25     var html = "";
26
```

Environment variables

reinvent-demo-index-html-generator:7

Node.js 6.10

index.handler

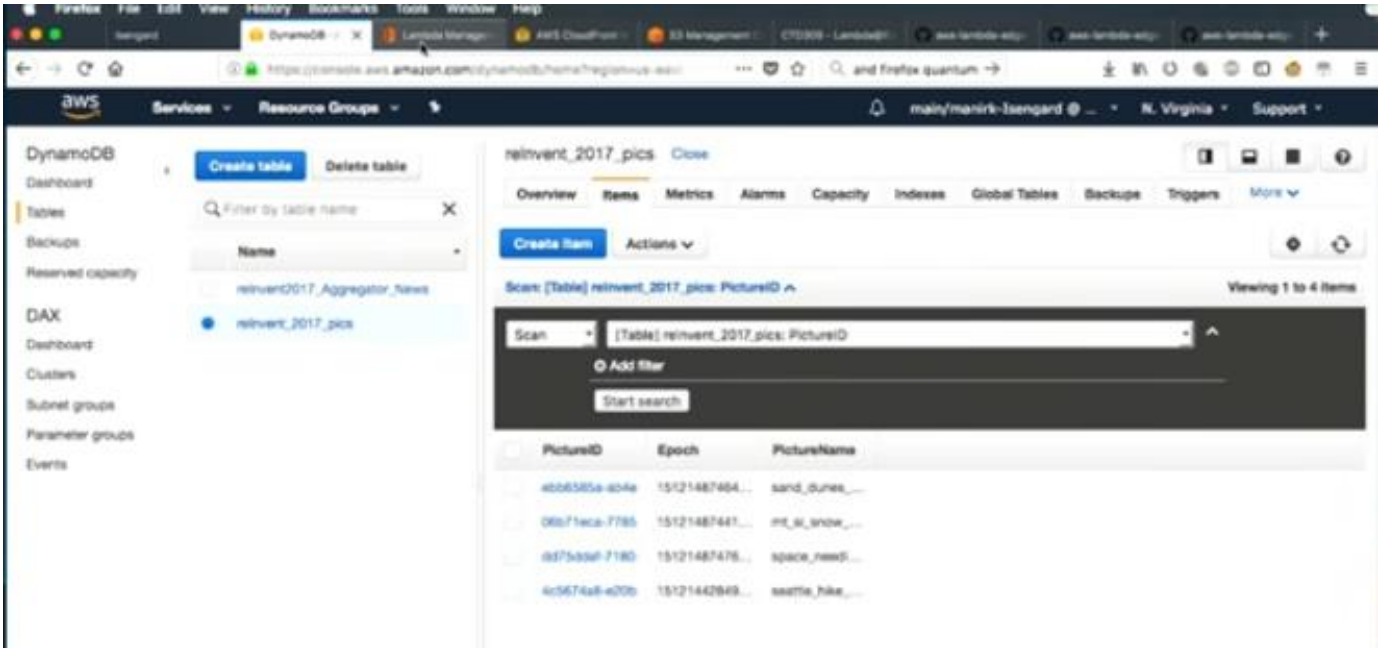
index.js

```
21 </html>`;
22
23
24 function generateImageMatrix(data) {
25     var html = "";
26
27     for (var i=0; i < data.Count; i++) {
28         const item = data.Items[i];
29         const ref = "/uploads/" + item.PictureID.S;
30         html += "<a href=\"" + ref + "\" class=\"card_wrap\">";
31         html += "<img class=\"card_img\" src=\"" + ref + "\" height=\"200px" width=\"200px">";
32         html += "</a>";
33         if (i % 2) {
34             html += "</br>";
35         }
36     }
37
38     return html;
39 }
40
41 function getDDBImages(callback) {
42     const params = {
43         TableName: "reinvent_2017_pics",
44         Limit: 6,
45     };
46
```

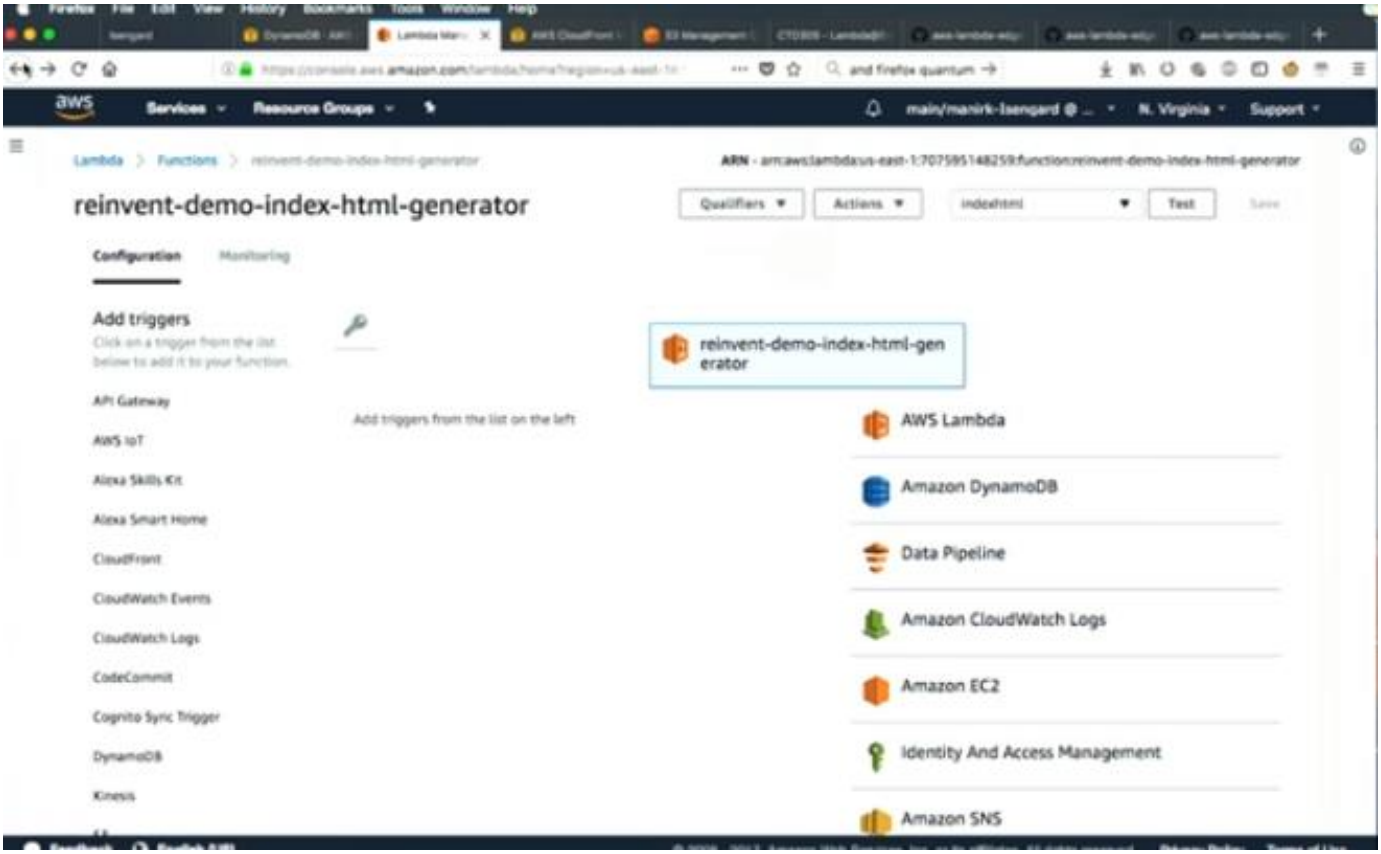
Environment variables

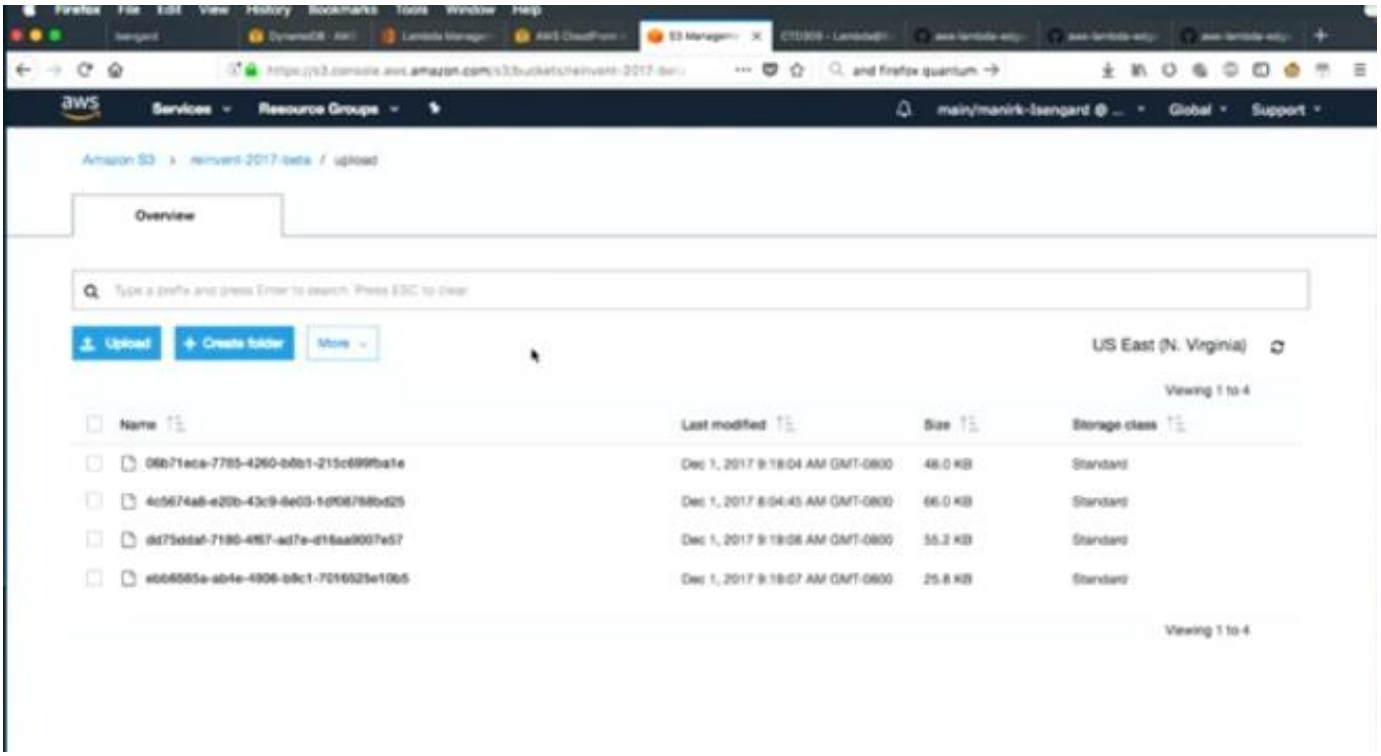
The code is looking at a DynamoDB table and scanning it for the 6 most recent items. We also could have picked a template to populate and then send out. There is a header section and a footer section

This is the pace where we are constructing the list of items. We then set some header values and send it back out.

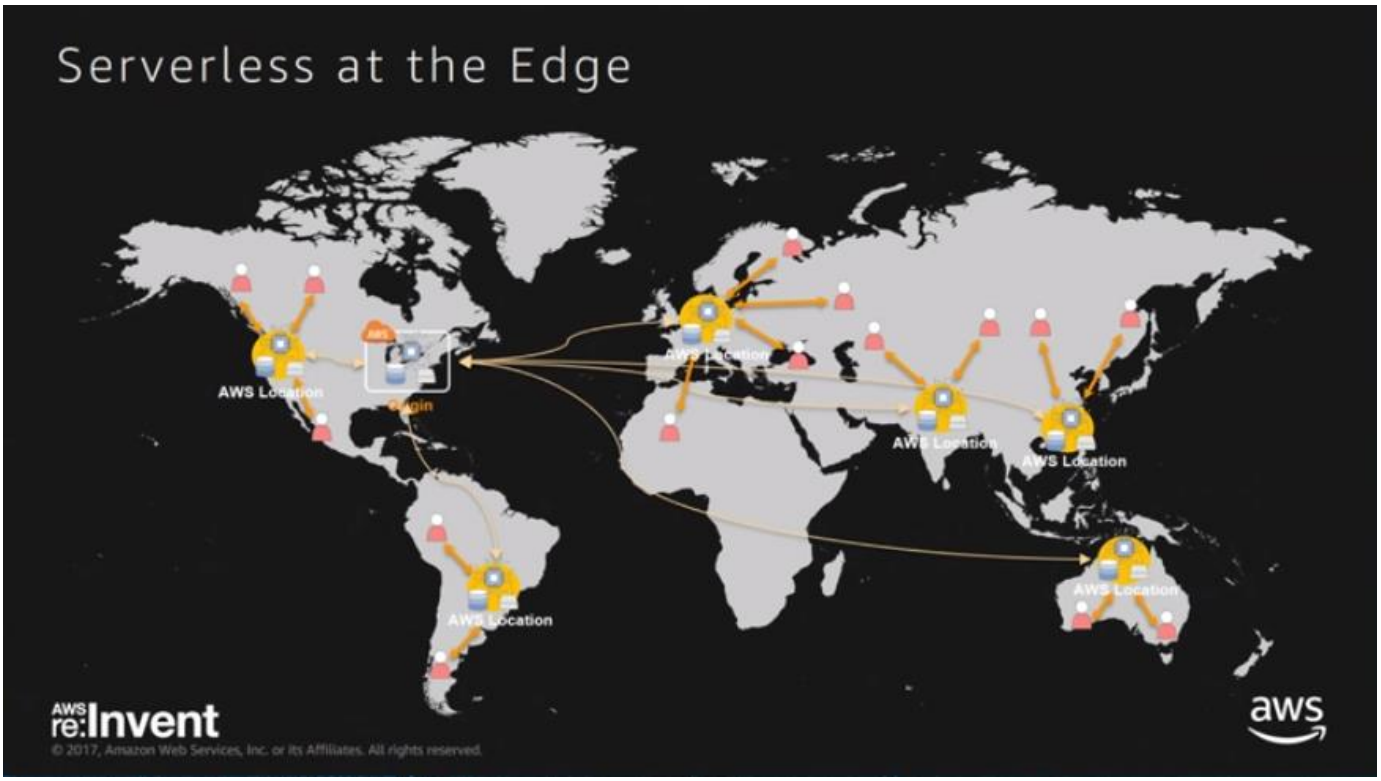


There are 2 things that happen on the upload function part, we write a file to S3 and then create an entry for the file in a DynamoDB table which is what the index.html file is going to read when trying to get the location of the file in S3 to display.





In S3, we change the file name into some guid value as metadata and set that as the URI of the image to display.



AWS re:Invent

Thank you!

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws