Next Gen Engineering
School of Tech - INDIA

AWS API Gateway with CDK

Putting a HTTP front on Lambda

Overview

- What is an API Gateway
- AWS API Gateway
- Running Lambda via API GW
- Accessing API directly
- Accessing API GW via CloudFront

Objectives

- Understand what an API Gateway is
- How to use it in front of Lambda
- Invoking Lambdas via HTTP
- Wiring CF in front of API GW
- Invoking Lambdas via CF & API GW

AWS sessions list

- AWS + Cloud intro 01 ✅ 1.5hrs
- AWS + Cloud intro 02 ✅ 1.5hrs
- AWS 01 S3 - storage (manual) ✅ 1.5hrs
- AWS 02 CDK intro - with S3 ✅ 3.0hrs
- AWS 03 Cloudfront - get files out of s3 ✅ 1.5hrs
- AWS 04 Lambda - running code ✅ 3.0hrs
- AWS 05 Api Gateway - put an API in front of Lambda ⬅ 3.0hrs
- AWS 06 Aurora Serverless Postgres - relational db 3.0hrs
- AWS 07 DynamoDB - non-relational db 3.0hrs

What is an API Gateway?

Say we have many useful lambdas that we want to use in our client code (like get gigs list, save new gig, and so on). We can't directly invoke those lambdas from our JavaScript code in our browser.

So, for example we want to have a nice set of urls to call to work with our backend system:

- http://infinigigs.ngei-sot.academy
    - /api/healthcheck
    - /api/gigs
    - /api/users
    - /api/tickets

Well, API Gateways let us do that.

What is an API Gateway?

API Gateway itself gives us urls like this:

- https://abcde.apigw.ap-south-1.amazon.com
    - `/api/healthcheck` (GET)
    - `/api/gigs` (GET and POST)
    - `/api/users` (GET and POST)
    - `/api/tickets` (GET and POST)

So that when our client code calls our backend (i.e. Lambdas) there is only one root address to go to.

We can put CloudFront over it for the nice *.ngei-sot.academy domains - which we will also do in this session.

What is an API Gateway?

Via redhat:

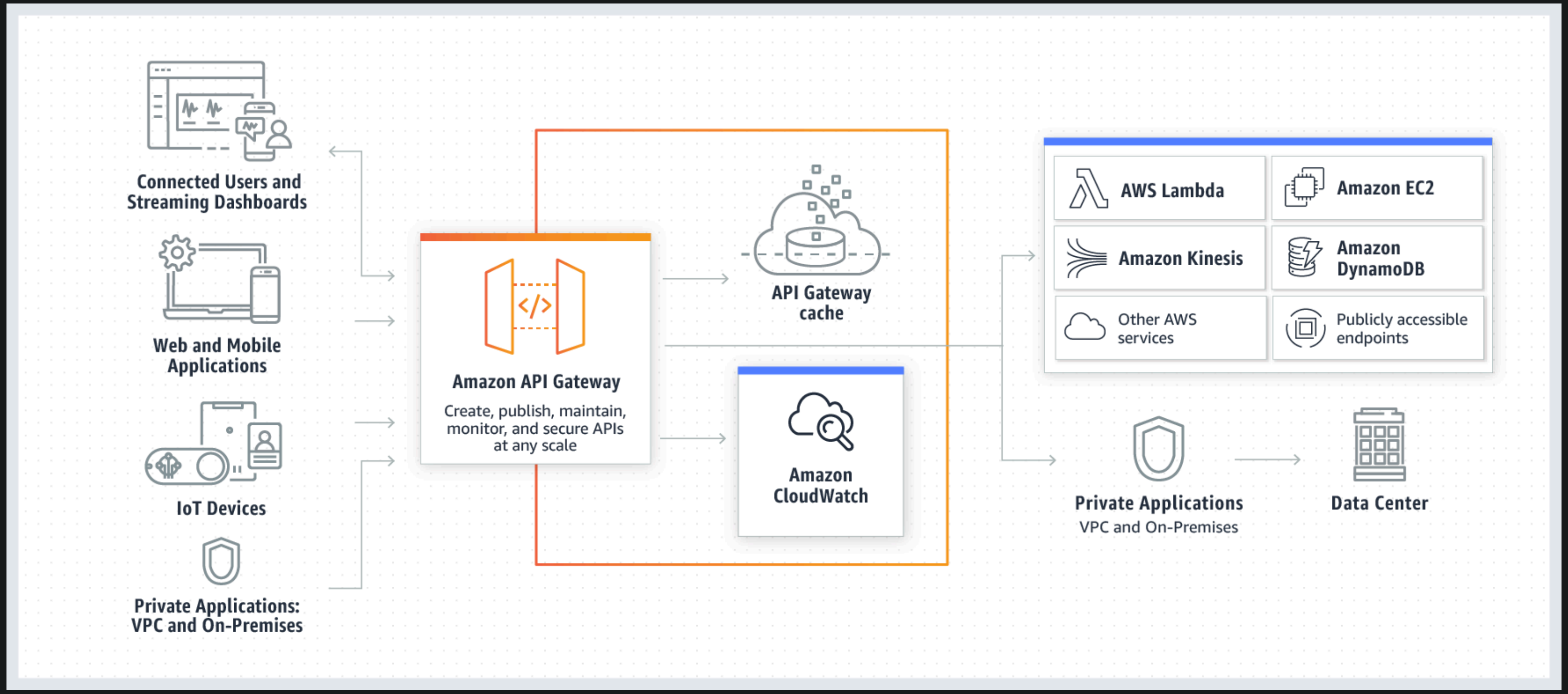
> An API gateway is an API management tool that sits between a client and a collection of backend services.
>
> An API gateway acts as a reverse proxy to accept all application programming interface (API) calls, aggregate the various services required to fulfil them, and return the appropriate result.

What is an API Gateway?

Via Amazon (our italics):

> Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale.
>
> APIs act as the "front door" for applications to access data, business logic, or functionality from your backend services.
>
> Using API Gateway, you can create RESTful APIs and WebSocket APIs that enable real-time two-way communication applications. API Gateway supports containerized and serverless workloads, as well as web applications.

Amazon's image:


Connected Users and Streaming Dashboards
Web and Mobile Applications
IoT Devices
Private Applications: VPC and On-Premises
Amazon API Gateway
Create, publish, maintain, monitor, and secure APIs at any scale
API Gateway cache
Amazon CloudWatch
AWS Lambda
Amazon EC2
Amazon Kinesis
Amazon DynamoDB
Other AWS services
Publicly accessible endpoints
Private Applications
VPC and On-Premises
Data Center

What does an API Gateway do?

What does an API Gateway do?

Can front many different backends for you

What does an API Gateway do?

Can front many different backends for you

Handles the processing of requests into your infrastructure

What does an API Gateway do?

Can front many different backends for you

Handles the processing of requests into your infrastructure

Can handle and route hundreds of thousands of requests per minute

What does an API Gateway do?

Can front many different backends for you

Handles the processing of requests into your infrastructure

Can handle and route hundreds of thousands of requests per minute

Can manage a REST api for you

What does an API Gateway do?

Can front many different backends for you

Handles the processing of requests into your infrastructure

Can handle and route hundreds of thousands of requests per minute

Can manage a REST api for you

Can manage a WebSockets api for you

And also...

And also...

Has CORS support to manage access from different sources

And also...

Has CORS support to manage access from different sources

Can be configured for varied authorization and access control methods

And also...

Has CORS support to manage access from different sources

Can be configured for varied authorization and access control methods

Can have throttling limits configured (to prevent DDOS etc)

And also...

Has CORS support to manage access from different sources

Can be configured for varied authorization and access control methods

Can have throttling limits configured (to prevent DDOS etc)

Handles monitoring

And also...

Has CORS support to manage access from different sources

Can be configured for varied authorization and access control methods

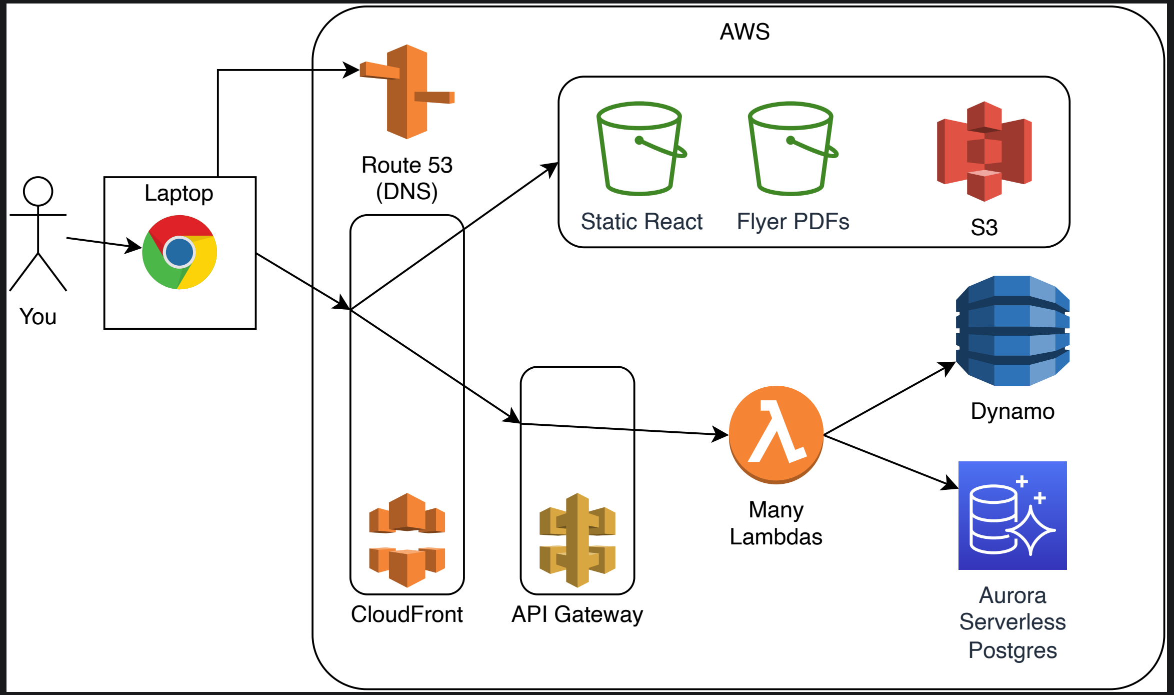
Can have throttling limits configured (to prevent DDOS etc)

Handles monitoring

Has API version management

InfiniGigs revisited

In the InfiniGigs app we have a gateway fronting the Lambdas:

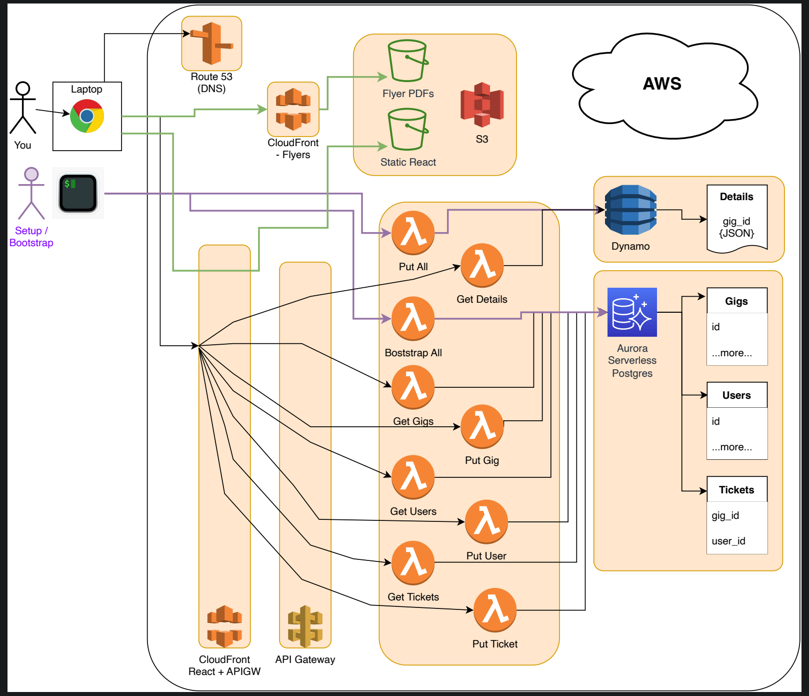


This allows us to make http(s) calls to a single service to access all the different features of our system.

Next Gen Engineering
School of Tech - INDIA

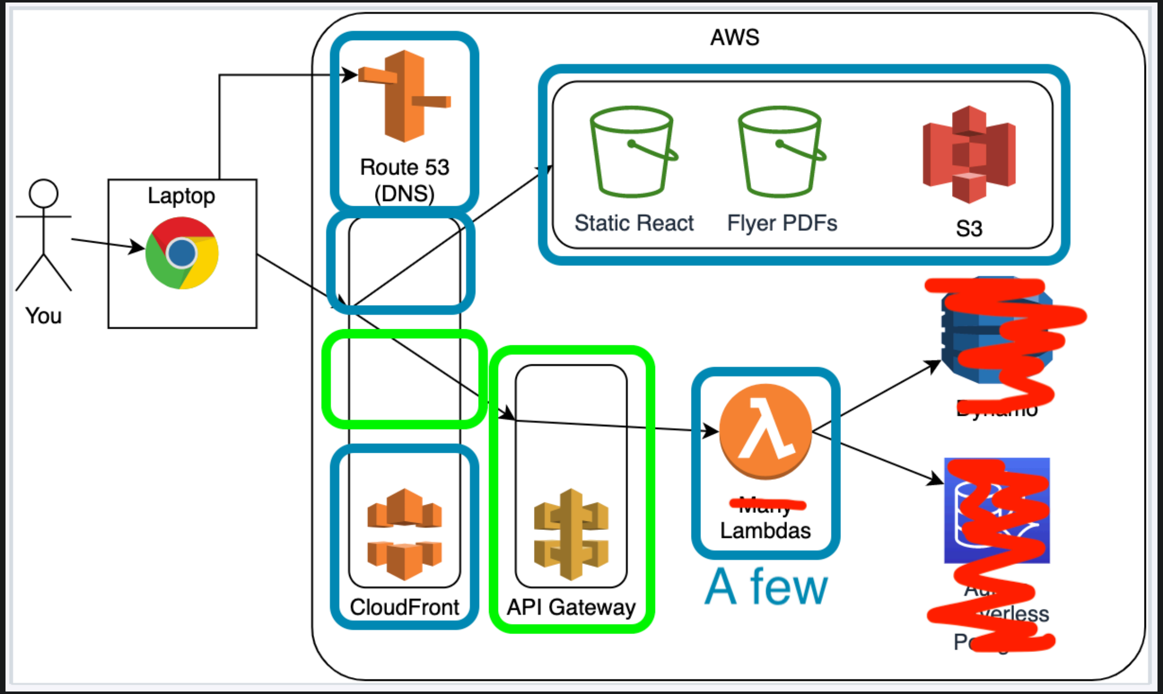
Infini Gigs gateway

All the lambda traffic is routed through the gateway:



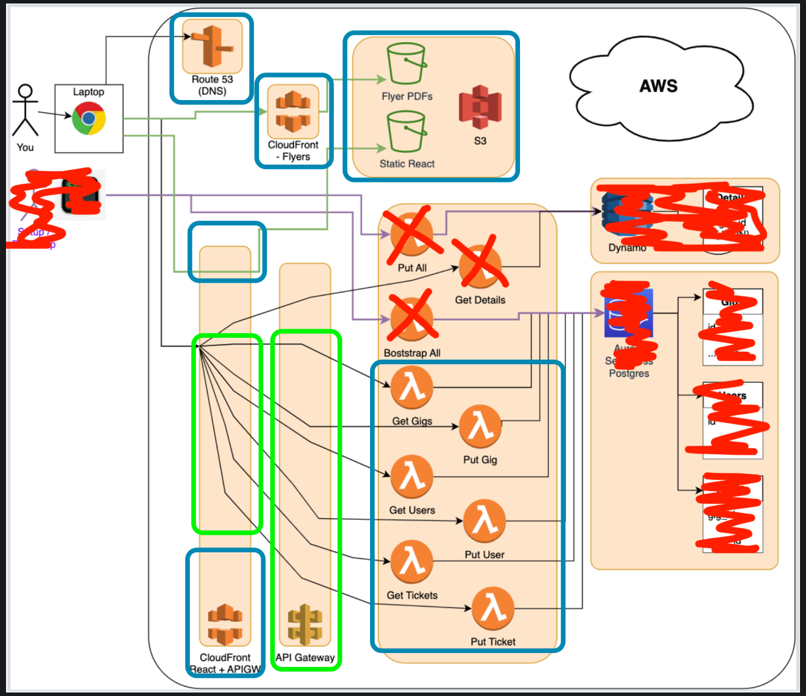
InfiniGigs revisited

In this session we are going to look only at the Gateway:



Well... there's a bit of wiring needed to get the unfriendly urls of API GW to be fronted by the nice ones (DNS) we have on CF... So we'll do that too.

InfiniGigs revisited



In later sessions we will plug in more backend behind the Lambdas - for this session they all return fake data. We'll also ignore the Dynamo ones in this session.

InfiniGigs API Gateway

To repeat from the previous slides - API Gateway itself gives us urls like this to the backends we link behind it:

- https://abcde.apigw.ap-south-1.amazon.com
    - `/api/healthcheck` (GET)
    - `/api/gigs` (GET and POST)
    - `/api/users` (GET and POST)
    - `/api/tickets` (GET and POST)

So that when our client code calls our backend (i.e. Lambdas) there is only one root address to go to.

InfiniGigs API Gateway

And to further recap from previous slides - If we have CloudFront linked to R53, then we can use that to give the API (Gateway) friendly names:

- http://infinigigs.ngei-sot.academy
    - /api/healthcheck
    - /api/gigs
    - /api/users
    - /api/tickets

Emoji Check:

On a high level, understand that we are about to put a "front" on our lambdas?

1. 😢 Haven't a clue, please help!
2. 🙁 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

Express equivalent - some code

Lets take a look at an implementation using `express` to get a frame of reference.

- Open the `./examples/express` folder in vscode.

Express equivalent - example app

This is a small example app to link the ideas of the APIS sessions to what we are doing in API Gateway.

> In AWS we don't need to run a whole server - we can have bits of code in Lambdas - and then front that with API Gateway so it looks like an express app (has a variety of urls) and behaves like an express app (has code in it to do different things).

Code along - install packages

In the terminal in the `./exercises` folder for this session:

- Run `./install.sh`

This will run `npm i` in the following locations:

- `client`
- `cdk`
- `cdk/functions`

Code Along - Check your env vars

Run the following in your terminal:

```
echo $GIGS_STACK_NAME
```

And make sure it gives you some output.

Speak now if it does not - we need this!

Code Along - Build react

Let's build our react application so it's up to date, and uses our own `GIGS_STACK_NAME`:

> Make sure you use Git bash for this

```
cd client
sh ./build.sh
```

This version has lots of the UI in it, ready and waiting for us to wire in the API!

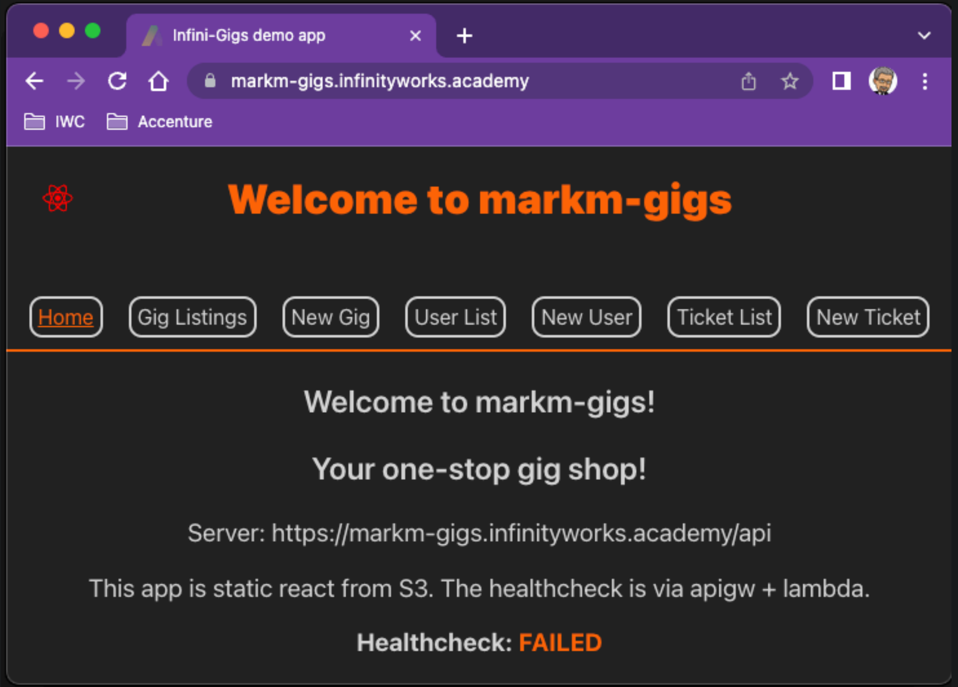
Code Along - initial deploy

To get everyone's stacks to the same starting point let's deploy now from the `./exercises/cdk` folder:

```
aws-logon # or use the full version
npx cdk deploy --profile iw-academy
```

Lets check it!

Browse to your UI on the homepage... You should see a UI up with 6 new data tabs (gigs, users, tickets, with a list and new tab for each)?



Welcome to markm-gigs

Welcome to markm-gigs!

Your one-stop gig shop!

Server: https://markm-gigs.infinityworks.academy/api

This app is static react from S3. The healthcheck is via apigw + lambda.

Healthcheck: FAILED

> And does it have your own name in it? If not things won't work later!

Oh no!

> The healthcheck has failed. Sad times :-(

- Examine the url the client JS called in the console - our client code wants a friendly url to talk to, i.e
    - https://markm-gigs.ngei-sot.academy/api/healthcheck

Emoji Check:

Did everyone get a UI up with 6 new data tabs (gigs, users, tickets, list and new tab for each) and a failing healthcheck?

1. 😢 Haven't a clue, please help!
2. 🙁 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

Time to fix it

To make the healthcheck happy we will need a bunch of new wiring in AWS.

That is - a new `RestApi` construct, with the relevant `properties` to configure it.

As usual for CDK this will take a few steps - but once we get to attaching the Lambdas to it, those steps are then shorter.

Code along - Import a gateway

Add the import to the top of your `./exercises/cdk/lib/cdk-stack.ts` file:

```
import * as apigw from 'aws-cdk-lib/aws-apigateway'
```

Code along - A gateway shaped hole

We've left a hint of where to put it in the `./lib/cdk-stack.ts` file:

```
// API GW.
// TODO
```

This is placed before the CloudFront construct so we can attach them together later.

Locate this code now; If you can't - you probably have the wrong `./exercises` folder open :-)

Code along - Add a gateway

Add this:

```
const api = new apigw.RestApi(this, 'apigw',
  {
    description: `${props.subDomain}-apigw`,
    restApiName: `${props.subDomain}-apigw`,
    deployOptions: {
      stageName: 'api', // to match CF later
    },
    deploy: true, // always deploy
  })
```

Here we set the main properties of the gateway. Very importantly the `stageName` must match what we do later in CloudFront, and we want CDK to always re-deploy it when we touch the settings.

Code along - open a security hole

We want to be able to test this from local and remote machines - so we have to open the CORS security hole. Add these settings after the `deploy: true,` flag:

```
defaultCorsPreflightOptions: {
  allowHeaders: [ 'Content-Type',
    'Access-Control-Allow-Origin',
    'Access-Control-Request-Method',
    'Access-Control-Request-Headers'
  ],
  allowMethods: [ '*' ], // Allow all
  allowCredentials: true,
  allowOrigins: [ '*' ], // Allow all
},
```

Rate limiting

API Gateway supports hundreds of thousands of requests per minute. We don't need that level of performance.

But also - it starts to cost as we have to pay AWS for invocations and data transferred - it starts off cheap but can quickly go up.

Also, there is a thing called a Distributed Denial Of Service (DDOS) attack, where people try and flood your site with bogus traffic. You'll still pay for your use of AWS!

Fortunately this is one of the many things we can set on API GW. There are many other things we can do but this one is a good example to use this session.

Code along - rate limiting

"Because DDOS costs dollars", add this after the API GW definition:

```
// API GW rate limiting
api.addUsagePlan('apigw-rate-limits', {
  name: `${props.subDomain}-apigw-rate-limits`,
  throttle: {
    rateLimit: 10,
    burstLimit: 5
  }
})
```

Code along - API raw url

We can now output the default url of the api; Find the `//Raw api url` comment at the end the file in the `// OUTPUTS` section and update it like so:

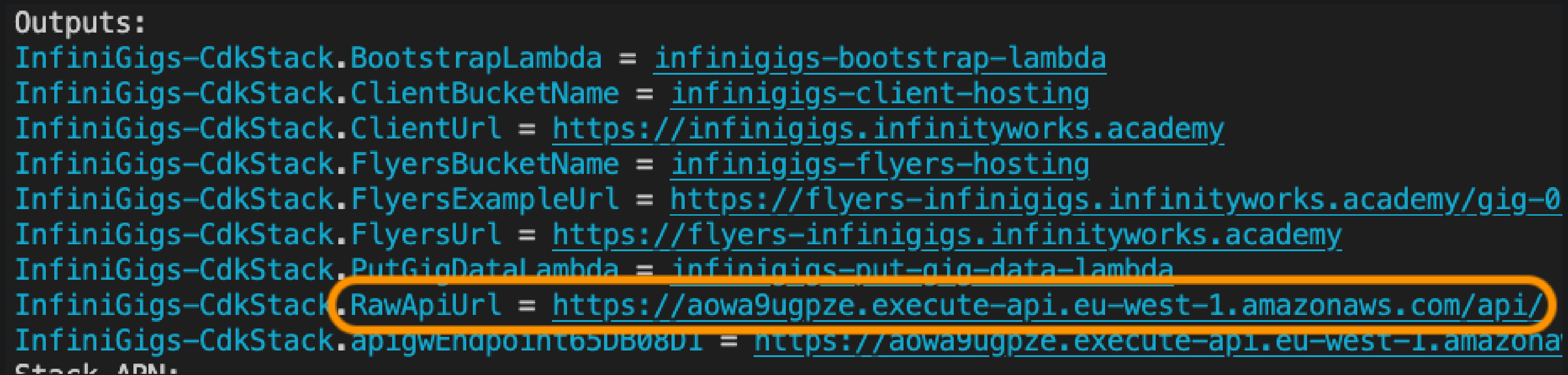
```
// Raw api url
new cdk.CfnOutput(this, 'RawApiUrl', {
  value: api.url ?? 'NO_URL',
})
```

Progress!

We've now done all the boring "boilerplate" things we need. Soon we can get to the real fun bit (of attaching this to Lambdas, CloudFront, and calling nice URLs!).

To check our code is correct so far lets deploy from our `./exercises/cdk` folder. You should see the raw api url in the output.

```
npx cdk deploy --profile iw-academy
```



```
Outputs:
InfiniGigs-CdkStack.BootstrapLambda = infinigigs-bootstrap-lambda
InfiniGigs-CdkStack.ClientBucketName = infinigigs-client-hosting
InfiniGigs-CdkStack.ClientUrl = https://infinigigs.infinityworks.academy
InfiniGigs-CdkStack.FlyersBucketName = infinigigs-flyers-hosting
InfiniGigs-CdkStack.FlyersExampleUrl = https://flyers-infinigigs.infinityworks.academy/gig-0
InfiniGigs-CdkStack.FlyersUrl = https://flyers-infinigigs.infinityworks.academy
InfiniGigs-CdkStack.PutGigDataLambda = infinigigs-put-gig-data-lambda
InfiniGigs-CdkStack.RawApiUrl = https://aowa9ugpze.execute-api.eu-west-1.amazonaws.com/api/
InfiniGigs-CdkStack.apigwEndpoint65DB08D1 = https://aowa9ugpze.execute-api.eu-west-1.amazona
Stack ARN:
```

Emoji Check:

Did everyone get deployed? Say so if not!

1. 😢 Haven't a clue, please help!
2. 🙁 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

Code along - healthcheck

Finally, the sort-of point of using API GW - we can attach the healthcheck lambda "behind" it:

```
// healthcheck lambda method on API
const healthcheckApi = api.root.addResource('healthcheck')
healthcheckApi.addMethod(
  'GET',
  new apigw.LambdaIntegration(
    healthcheckLambda, { proxy: true }
  ),
)
```

Resources on an API

There are a few parts of that wiring, and they are worth a mention:

The first adds a `./healthcheck` url to the root (`.`) of the api's url:

```
const healthcheckApi = api.root.addResource('healthcheck')
```

Resources on an API

The next bit adds a HTTP `GET` to that url, and backs it with a integration to a lambda:

```
healthcheckApi.addMethod(
  'GET',
  new apigw.LambdaIntegration(
    healthcheckLambda,
    { proxy: true }
  ),
)
```

The `proxy: true` setting means that we have a Lambda Proxy Integration - There's a lot of detail for another time in that link.

Code along - deploy

We can now deploy the healthcheck integration:

```
npx cdk deploy --profile iw-academy
```

Code along - a url!

We can now see our healthcheck work from raw url GET in a browser

- Call your healthcheck with http://my-api-url/healthcheck
- E.g. https://aowa9ugpze.execute-api.ap-south-1.amazonaws.com/api/healthcheck

If we checked the lambda logs in CloudWatch, we should see successful invocations.

Urg, horrible URL!

> API Gateway gives you a horrid url like <ajgbjad.hjfefe.amazonaws.binbag.com> to your server code
>
> CloudFront then gives a friendly url like <mynicelittlepage.com/fairies> to your code instead

Ruby, Sept 2022

Urg, horrible URL!

That binbag.com URL is not the sort of thing we want to give customers though!

Now it's time to update the CloudFront client distribution with `additional behaviors` to send traffic onwards to the api-gw :-)

Why the Client distribution, not the Flyers one?

Well, because of the nice domain name on it e.g. `https://infinigigs.ngei-sot.academy` is the right logical place to have an associated api like `https://infinigigs.ngei-sot.academy/api/`.

Code Along - CF Additional Behaviour

First find the correct place.

We must extend the existing CF distribution with extra settings.

We have commented it in the code, like so:

```
// Additional CF behaviors to link to API
// TODO
```

Code Along - CF Additional Behaviour

Insert this extra code:

```
additionalBehaviors: {
  '/api/*': { // must be same as in ApiGW construct
    origin: new origins.HttpOrigin(
      `${api.restApiId}.execute-api.`
          +`${props.env.region}.amazonaws.com`,
      {  originPath: '/' }
    ),
    viewerProtocolPolicy:
      cloudfront.ViewerProtocolPolicy.REDIRECT_TO_HTTPS,
    allowedMethods: cloudfront.AllowedMethods.ALLOW_ALL,
    cachePolicy: cloudfront.CachePolicy.CACHING_DISABLED,
  },
},
```

Code Along - friendly url

In the outputs section, we can now find the `// TODO` for the `Pretty api url` output and update it:

```
// Pretty api url
new cdk.CfnOutput(this, 'PrettyApiUrl', {
  value: `https://${fullDomain}/api/`,
})
```

Code along - deploy

We can now deploy the CloudFont forwarding:

```
npx cdk deploy --profile iw-academy
```

A nice url!

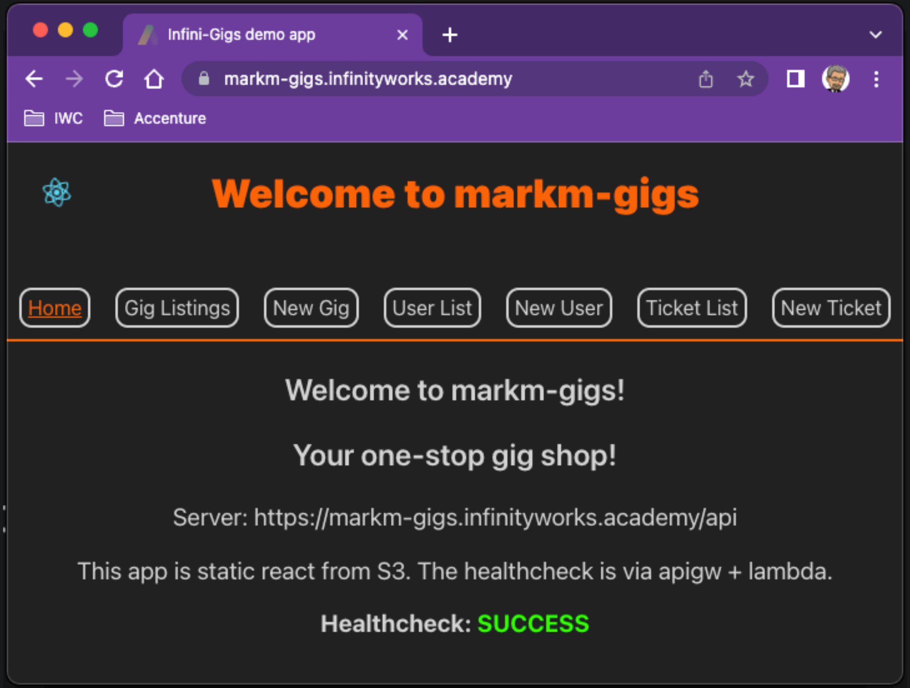
We can now see our healthcheck work from the nice url using a GET in a browser:

- Call your healthcheck with http://my-gigs/api/healthcheck
- E.g. https://markm-gigs.ngei-sot.academy/api/healthcheck

If we checked the lambda logs in CloudWatch, we would see the successful invocations.

And there's more!

Our web page healthcheck should be ok now - the icon is blue, not red, and the text is green!



Welcome to markm-gigs!

Your one-stop gig shop!

Server: https://markm-gigs.infinityworks.academy/api

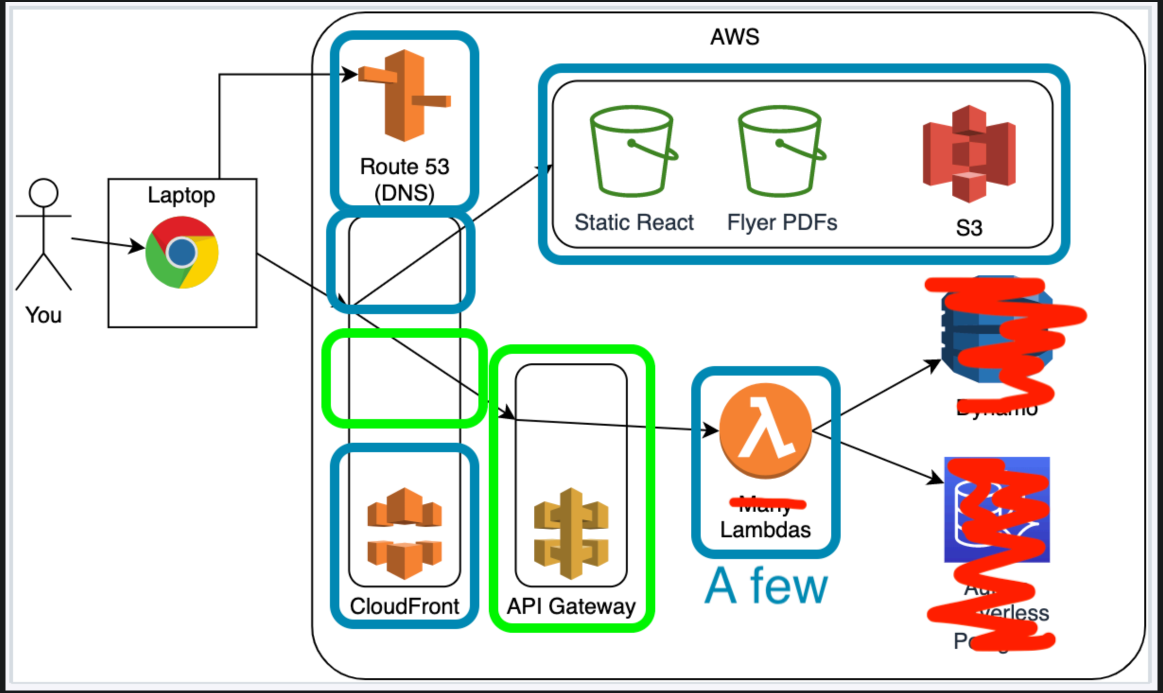
This app is static react from S3. The healthcheck is via apigw + lambda.

Healthcheck: SUCCESS

Next Gen Engineering
School of Tech - INDIA

Infini gigs revisited

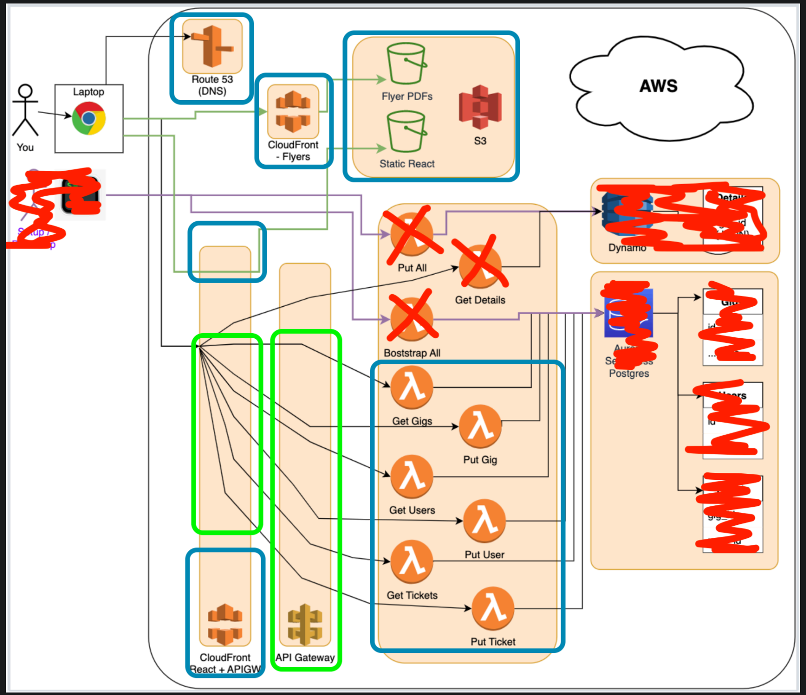
We've now filled in part of this wiring (the green bit):



We still need to wire in more Lambdas, but an important bit is done :-)

Infini gigs details

On the detailed view that is:



Emoji Check:

Are the pieces starting to come together? Do you, at a high level, get that we have put API GW in front of our Lambda, and CF in front of that?

1. 😢 Haven't a clue, please help!
2. 🙁 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

What's next?

Now we could add all the `get` lambdas with `GET` integrations on the API, and we should be able to see some data in our UI!

...But ony if we use the same urls the React client is coded to use.

Breakouts - 15 mins

Copying what you did for `healthcheck`, do this:

- On the `api.root`, add a `gigs` resource. Name the variable `gigsApi`
- Using `gigsApi`, add a `GET` method proxy `LambdaIntegration` to the `getGigsLambda`

Then

- Repeat for a `users` resource integrated with `getUsersLambda`
- Repeat for a `tickets` resource integrated with `getTicketsLambda`

Then deploy, and test your UI's "xxx List" tabs!

To be sure it's worked you will want to check the CloudWatch logs for your `yourname-get-XYZ` lambdas.

Emoji Check:

Did you make your "Gig Listings", "User List" and "Ticket List" tabs work now?

Hopefully the speed at which we can now add integrations to the API was worth trudging through the boilerplate?

1. 😢 Haven't a clue, please help!
2. 🙁 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

Many methods

Each URL "resource" on our API can have different lambdas handling fetching data, creating it, updating it, and so on.

In InfiniGigs we have been good and made separate small lambdas for each of the listing (`GET`) and creating (`POST`) tabs on our website.

Many methods

For example:

```
const ticketsApi = api.root.addResource('tickets')
ticketsApi.addMethod( 'GET', /* integration here */ )
ticketsApi.addMethod( 'POST', /* integration here */ )
ticketsApi.addMethod( 'PUT', /* integration here */ )
ticketsApi.addMethod( 'DELETE', /* integration here */ )
```

Breakouts - 15 mins

Copying what you did for `healthcheck`:

- Using the `gigsApi` variable, add a `POST` method proxy `LambdaIntegration` to the `postGigLambda`.

Then assuming you were consistent with variable naming...

- Repeat for `usersApi` with a `POST` to the `postUserLambda`
- Repeat for `ticketsApi` with a `POST` to the `postTicketLambda`

Then deploy, and test your UI's "New xxx" tabs!

To be sure it's worked you will want to check the CloudWatch logs for your `yourname-post-XYZ` lambdas.

Emoji Check:

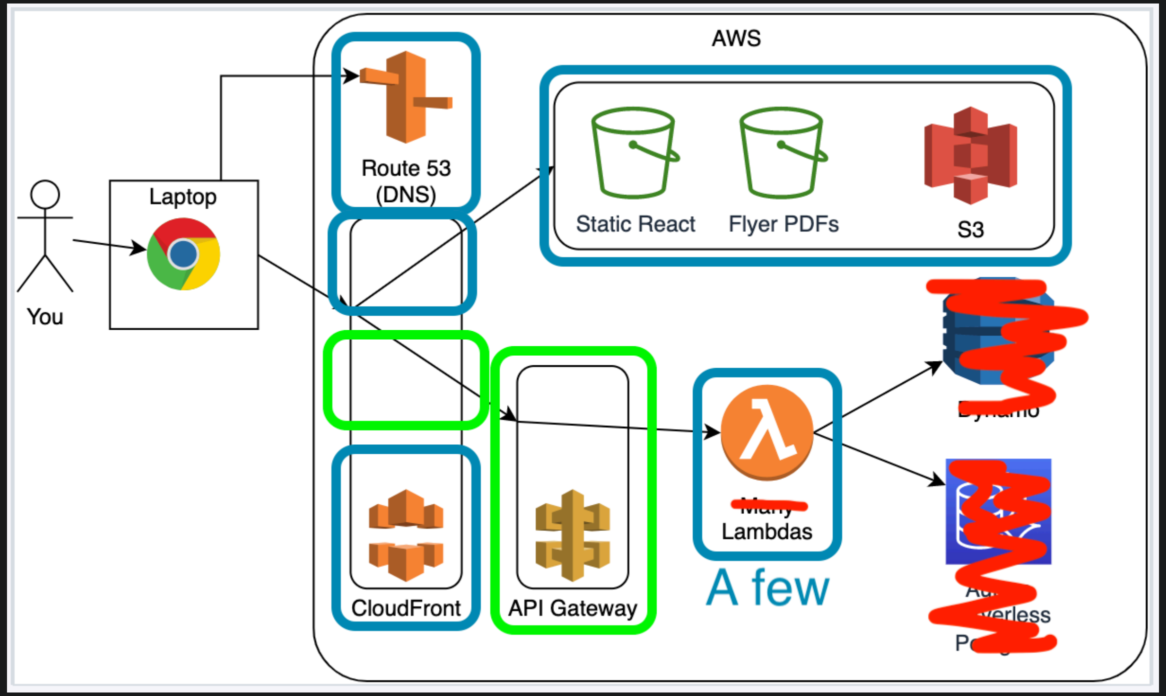
Did you make your "New Gig", "New User" and "New Ticket" tabs work now?

Hopefully the speed at which we can now add integrations to the API was worth trudging through the boilerplate?

1. 😢 Haven't a clue, please help!
2. 🙁 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

Look how much we've done now!!

Well done :-)



Overview - recap

- What is an API Gateway
- AWS API Gateway
- Running Lambda via API GW
- Accessing API directly
- Accessing API GW via CloudFront

Objectives - recap

- Understand what an API Gateway is
- How to use it in front of Lambda
- Invoking Lambdas via HTTP
- Wiring CF in front of API GW
- Invoking Lambdas via CF & API GW

Emoji Check:

On a high level, do you think you understand the main concepts of this session? Say so if not!

1. 😢 Haven't a clue, please help!
2. 🙁 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively