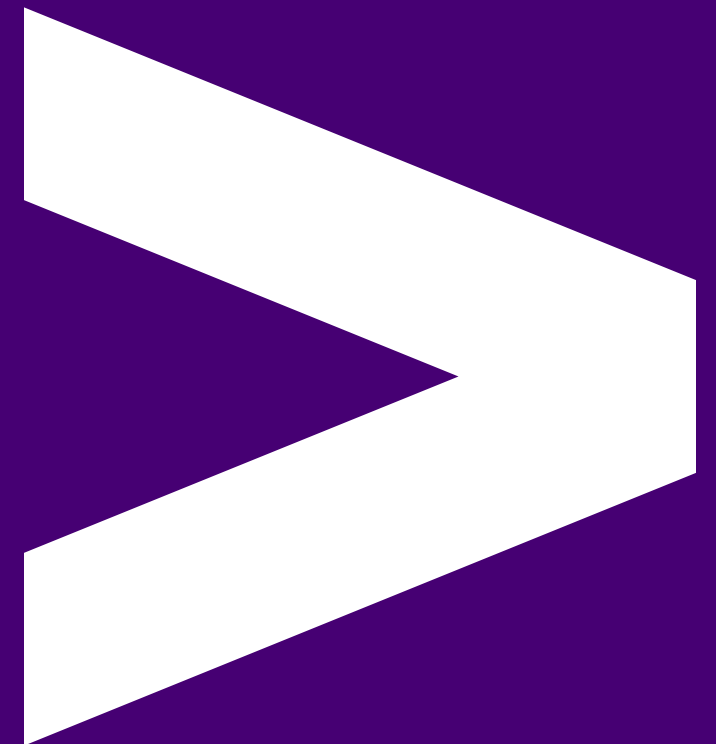


# AWS Cloud Front with CDK

CF and getting files out of S3








# Overview

- Continue with IAC (Infrastructure as Code)
- Use AWS CDK
- Put CloudFront (CF) in front of S3 to access our files
- Add Route53 to give CF a friendly DNS name
- Make and run an example. Twice.

# Objectives - recap

- Understand what CloudFront (CF) is
- Use the CF CDK construct in JS
- Understand what Route53 does
- Use Route53 to give a CF distribution a nice name
- Access files in S3 via CF

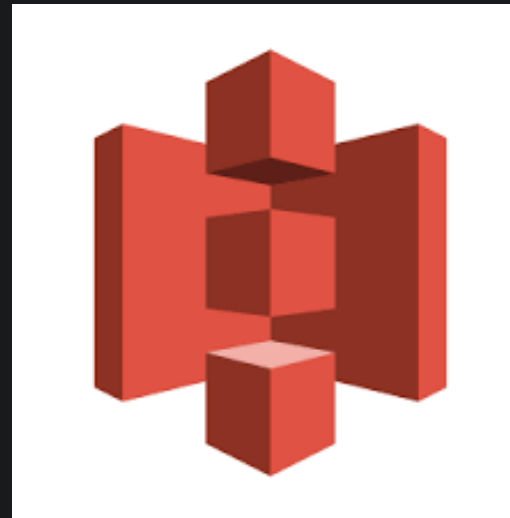
# AWS sessions list

- 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 Cloudfront?

"Amazon CloudFront is a content delivery network (CDN) service built for high performance, security, and developer convenience."

<https://aws.amazon.com/cloudfront/>



No really - What is Cloudfront?

# No really - What is Cloudfront?

Q) What does "Content Delivery Network" mean?

# No really - What is Cloudfront?

Q) What does "Content Delivery Network" mean?

A) A service that shares your files around the world for easy access



# No really - What is Cloudfront?

Q) What does "Content Delivery Network" mean?

A) A service that shares your files around the world for easy access

Hmm but what does that mean?

# What is Cloudfront?

Here's a slack question we asked of our colleagues:

Mark Matthews Today at 15:28

So. Cloudfront. The AWS docs are really waffly about what it actually is. So: How would you describe what CloudFront is to newbies in a couple of punchy sentences?

# What is Cloudfront?

And some responses:

Chris Mills Today at 16:02

Yeah but say CDN to a newbie and they'll just nod and agree.

I used the Netflix example to give them a better mental model: You've got a film hosted on a server in New York (or wherever), but you don't want to stream it from New York to London a million times over. So you have an edge server where it's cached - now they just stream from [i.e. London]. Repeat for other areas of the world.

Ditto for static resources like images etc.

# What is Cloudfront?

And:

Johannes Geiger Today at 16:11

You don't want to drive to a farm miles away for a bottle of milk? Yeah, that's why there's corner shops. (Cloudfront taking care of distributing the milk from the farm to the shops in my bad example 😊)

# What is Cloudfront?

So then we can get to:

Lewis Richardson Today at 16:23

Caches content at locations in closer proximity to the user than the original source, in terms of CDN

And now hopefully these make sense?

# What else does it do?

As well as acting as a local cache in worldwide regions, it can also

- Serve static and dynamic content from backends like S3
- Have Route 53 DNS attached for friendly names
- Have security limits set (itself and connected to a web application firewall, or WAF)
- Have lambda functions directly attached to process requests
- Be set to allow or deny traffic from locations and other parts of AWS (see also CORS)
- Allow or deny query parameter pass-through to backend APIs
- Serve backends like API Gateway for complex processing

# Infini-Gigs to the rescue?

Perhaps there's an example in the Infini Gigs architecture?

# Infini-Gigs - Revisited

Lets looks at the Cloudfront + R53 part of the [Infini Gigs](#) architecture:

See the basic and extended architecture images in the README of [IW-Academy/infini-gigs-demo-project](#)

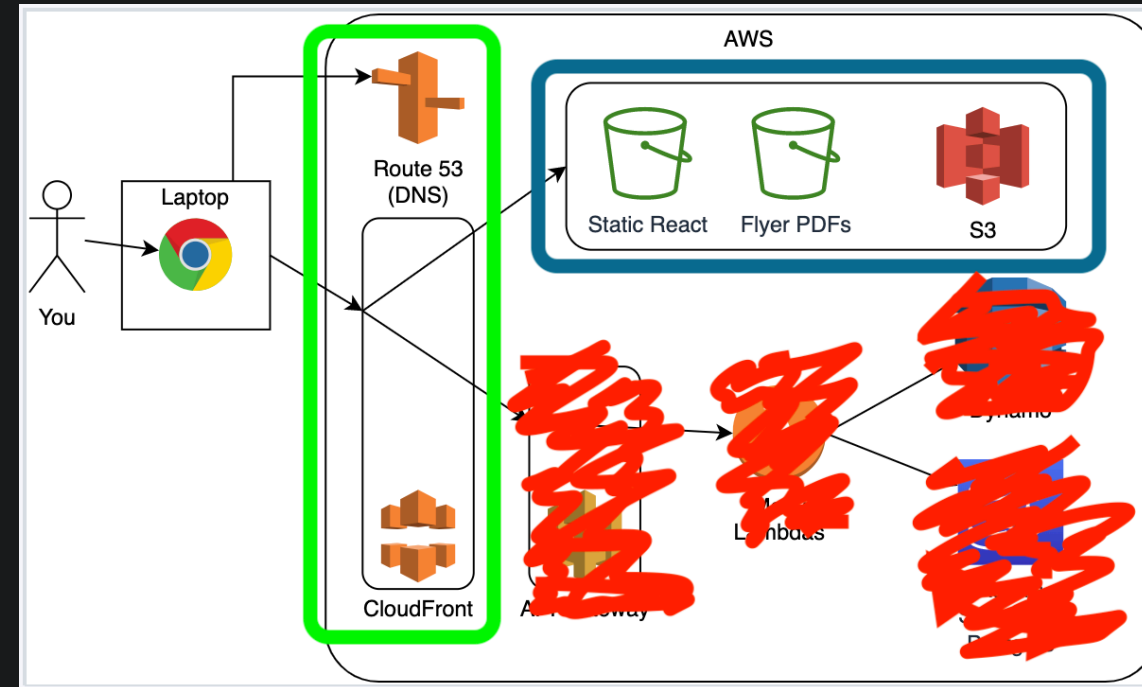


# Infini Gigs - Serving Flyers

In this session we are looking at putting CloudFront (CF) and Route53 (R53) in front of the files in S3, which is this part from [Infini Gigs](#):

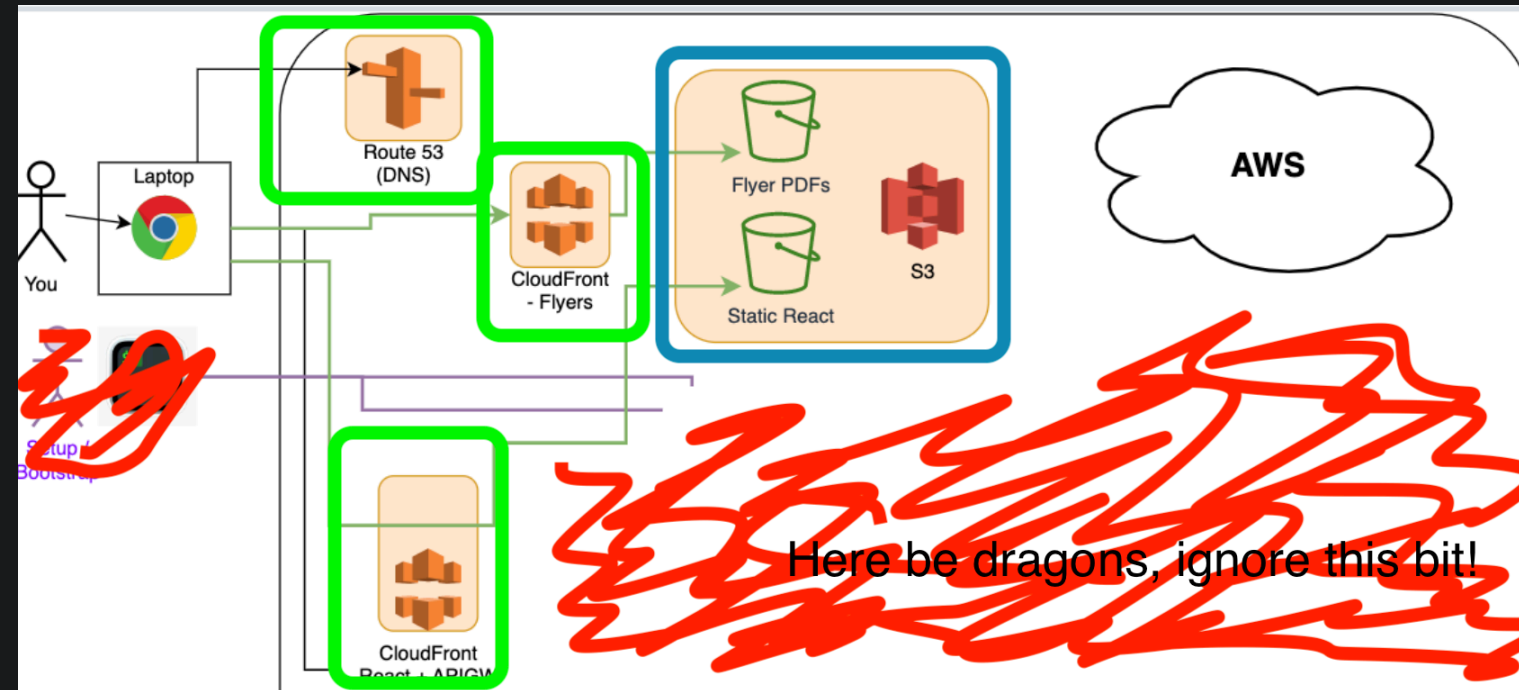
From now on: Always with code, not manually!

I.e, from the overview diagram:



We will now expose those file to the internet. The blue parts were done previously.

And from the more detailed one:



The blue parts were done previously. There's still lots of this we don't need yet.

# Infini Gigs - other stuff

We don't yet need a backend to get these files out.

Our react client won't "work" yet as there's lots needed to support the complexity of a web app - but we can make sure it is reachable from the internet.

Then in following sessions we can add that middle layer (code for processing) and backend (storage).

## Emoji Check:

On a high level, do you think you understand the target 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

# The Infini-Gigs front end

In this session, we have a bare-bones Infini Gigs front end - until we add more in following sessions, it wouldn't fully "work" anyway!

To build it, change to the `aws-iac-03-cdk-cloudfront/exercises/client` folder and run this:

```
chmod a+x build.sh  
./build.sh  
ls ./dist/*
```

## Demo - Thus far

So far we have put the gig flyers and a cut-down react front end in S3.

See the `./exercises` folder - this is our starting point today.

Make sure you have the `aws-iac-03-cdk-cloudfront` files!

In here we have the same `gig-flyers` and `client` that we completed last time, with two S3 deployments.

Now we need to add more!

# Code-along: get ready

Open a terminal on the `./exercises` folder for this session.

- Run `pwd`
  - you should see the right `aws-iac-03-cdk-cloudfront/exercises` folder path
- Run `ls -la`
  - You should see the `gig-flyers`, `client` and `cdk` folders

From now on we will start with a generic `cdk` folder.



# Code-along: get ready

Why a generic `cdk` folder?

This has the dynamic stack name, so if we did the previous session correctly, this and each following session will update your own stack as per the `GIGS_STACK_NAME` env var 😊

# Emoji Check:

Are you definitely in the `./exercises` folder for THIS session, `aws-iac-03-cdk-cloudfront`?

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: get ready part 2

- Run `cd cdk`
- Run `ls -la` - you should see the `package.json` file plus others
- Log into AWS with your `aws-logon` alias, or the full command
- Run `npx cdk synth` to check it compiles

## Code-along: get ready part 3

- Run `npx cdk deploy`
- There should be no changes from the previous session
  - but if there are and you get a `Y/N` prompt then respond `Y` to bring your stack up to date

Now all our stacks are at the same starting point.

# CloudFront construct

Now we can add the CF construct into our code, and point it at an S3 bucket.

I.e. Cloudfront sits in front of the S3 buckets.

See next slide

## More settings needed

We will need some more settings to make our stack work with CloudFront.

We need Certificates for the Route 53 DNS.

And a root domain name for us to link to i.e. `markm-gigs.ngei-sot.academy`

## Code along - more settings

Update the `settings` object in your lib file (`./exercises/cdk/lib/cdk-stack.ts`) to add some new things like so:

```
export interface GigsSettings extends cdk.StackProps {  
  certArn: string, // new  
  permissionsBoundaryPolicyName: string,  
  domainName: string, // new  
  subDomain: string,  
}
```

This adds `certArn` and `domainName`. What do you think we will use them for?

## Code Along - get env var

Let's pull our `GIGS_DOMAIN` env var into the `bin/*.ts` file so we can use it.

Near the top of the file, after the `GIGS_STACK_NAME` env var is used, add the following:

```
const domainName: string = process.env.GIGS_DOMAIN || ''
if (!(domainName && domainName.trim() && domainName.trim().length)) {
  console.error(`PARAMETER $GIGS_DOMAIN NOT SET, got: '${domainName}'`)
  process.exit(1)
}
```



# Code along - more settings

Update the matching `settings` object in your bin  
(`./exercises/cdk/bin/cdk.ts`) file to specify the values for `certArn` and  
`domainName`:

```
const settings: GigsSettings = {  
  // Inherited from cdk.StackProps  
  env: {  
    account: process.env.CDK_DEFAULT_ACCOUNT || 'NOT_SET',  
    region: process.env.CDK_DEFAULT_REGION || 'NOT_SET',  
  },  
  stackName: stackName,  
  // Our own properties  
  certArn: cdk.Fn.importValue('SoTSharedCertArn'), // new  
  permissionsBoundaryPolicyName: 'ScopePermissions',  
  domainName: domainName.toLowerCase(), // new  
  subDomain: stackName.toLowerCase(),  
}
```

# Sharing values between stacks

What do we think this bit of code is doing?

```
cdk.Fn.importValue( 'SoTSharedCertArn' )
```

## Code along - imports

We will need to use some more parts of CDK, so let's import those in our `./exercises/lib/cdk-stack.ts` file:

```
// at the top //
import * as cloudfront from 'aws-cdk-lib/aws-cloudfront'
import * as origins from 'aws-cdk-lib/aws-cloudfront-origins'
import * as acm from 'aws-cdk-lib/aws-certificatemanager'
import * as route53 from 'aws-cdk-lib/aws-route53'
```

## Code along - domains

We will need to have some domain names for DNS. Also in the `lib/cdk-stack.ts` file, add this at the very top of the `constructor` method:

```
// Domain variables go here – to use in route 53 etc
const fullDomain = `${props.subDomain}.${props.domainName}`
const flyerDomain = `flyers-${props.subDomain}.${props.domainName}`
```

## Code along - domains

We have a wildcard cert for a `*.ngei-sot.academy` domain. Under this we can add many other sub-domains like `yourname.ngei-sot.academy`.

After the permissions boundary and before your flyers bucket, add this:

```
// Lookup cert for domain *.ngei-sot.academy
const cert = acm.Certificate.fromCertificateArn(
  this,
  `cert`,
  props.certArn,
)
```

# Code along - CF Construct

Now we can add the CF construct - do this at the end of the **constructor**:

```
// Cloudfront sits in front of S3 bucket
const flyersDistribution = new cloudfront.Distribution(this,
  `flyers-distribution`,
  {
    defaultBehavior: {
      origin: new origins.S3Origin(flyersBucket),
      viewerProtocolPolicy:
        cloudfront.ViewerProtocolPolicy.REDIRECT_TO_HTTPS,
    },
    priceClass: cloudfront.PriceClass.PRICE_CLASS_100,
    defaultRootObject: 'index.html',
    domainNames: [ flyerDomain ],
    certificate: cert,
  }
)
```

## Code along - outputs

Let's also output the DNS of our distribution, so we can try and get our files out of the right URL - do this at the end of the `constructor`:

```
// Flyers outputs
new cdk.CfnOutput(this, 'FlyersDistributionDNS', {
  value: flyersDistribution.distributionDomainName
})
new cdk.CfnOutput(this, 'FlyersSampleDNS', {
  value: `https://${flyersDistribution.distributionDomainName}`
})
```

## Are we there yet? (Code along)

Very nearly. We should now be able to get our files out of S3 via the CloudFront domain!

Let's deploy and check the outputs - Run this:

```
aws-login # or the full command  
npx cdk deploy
```

We should all get an output like this:

```
markm-gigs-CdkStack.FlyersSampleDNS  
    = https://d2dsnl6ji8djrm.cloudfront.net/gig-001-flyer.pdf
```



# Testing time! (Code along)

Open the outputted URL in your browser - did it work?

- I.e. <https://d2dsnl6ji8djrm.cloudfront.net/gig-001-flyer.pdf>

But what about a nice name?

## But what about a nice name?

Q1) What about a nice name? Can anyone think why we are not using it yet?

## But what about a nice name?

Q1) What about a nice name? Can anyone think why we are not using it yet?

Q2) Didn't we set the "domain" on the CF construct

## But what about a nice name?

Q1) What about a nice name? Can anyone think why we are not using it yet?

Q2) Didn't we set the "domain" on the CF construct

Answer: 1+2... We didn't wire in a "proper" Route53 DNS yet

## Code along - add a R53 record

Now we can wire in a "nice" domain name using the `flyerDomain` we set before. Add this to the end of your `constructor`:

```
// Route 53 glue between DNS and CloudFront
const zone = route53.HostedZone.fromLookup(this, 'zone', {
  domainName: props.domainName,
})
new route53.CnameRecord(this, 'flyers-record', {
  zone,
  domainName: flyersDistribution.domainName,
  recordName: flyerDomain,
})
```

This registers the "nice name" with R53 and links it to the CF distribution.

## Code along - DNS outputs

And we can also output the nice DNS names too, also at the end of your **constructor**:

```
// Route 53 outputs
new cdk.CfnOutput(this, 'FlyersUrl', {
  value: `https://${flyerDomain}`,
})
new cdk.CfnOutput(this, 'FlyersExampleUrl', {
  value: `https://${flyerDomain}/gig-001-flyer.pdf`,
})
```

# Code along - it's ready!

Now if we redeploy we should be able to get our flyers from the nice DNS!

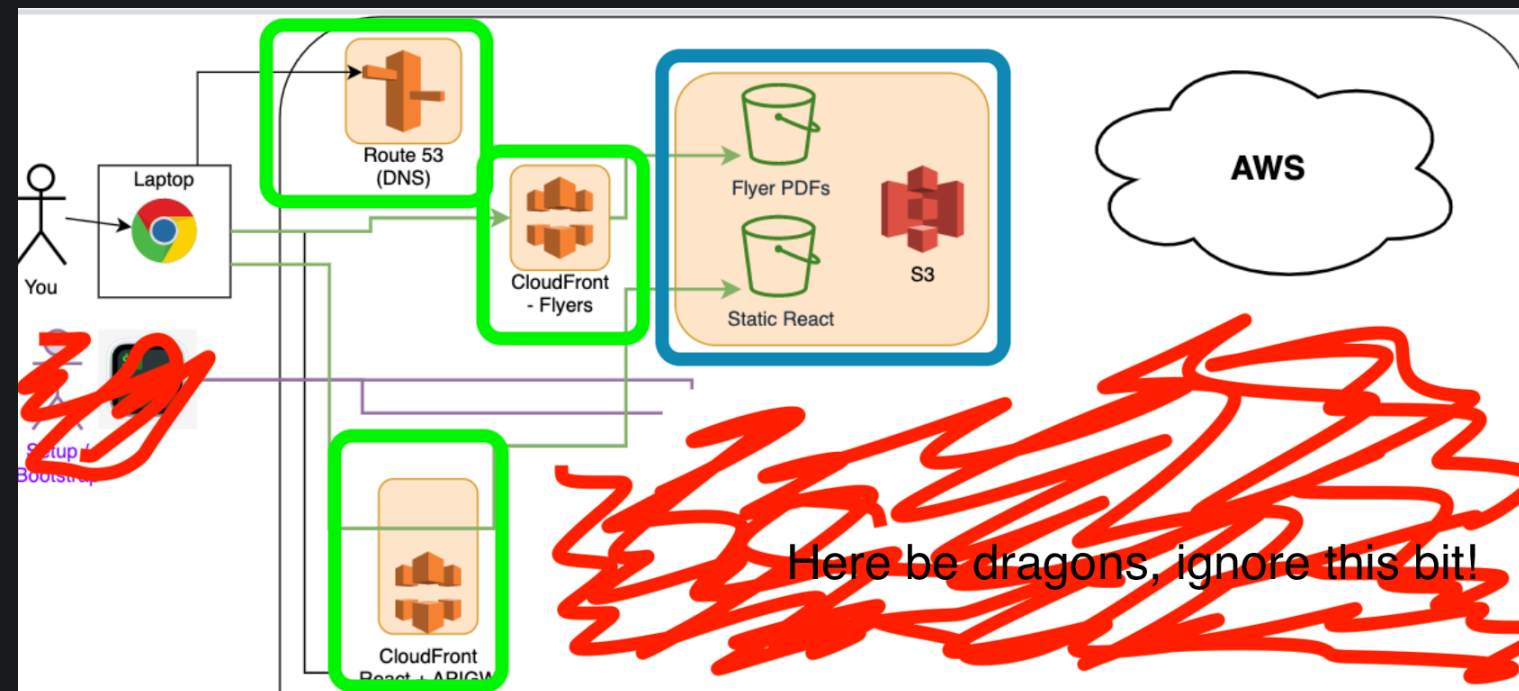
- Re-deploy
- You should get an output like this:
  - <https://flyers-markm-gigs.ngei-sot.academy/gig-001-flyer.pdf>
- Test the URL!
- Try different flyer numbers too!

Also

- What happens if you miss out the file name and just browse to the root of the bucket (the CF DNS)?



# What we just made work



Any questions about the pieces?

## Emoji Check:

Cool huh! On a high level do you get the building we have just done / the wiring we have used?

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

## Stale stuff

If we redeploy now, CDK will both (a) upload new files but (b) not tell CloudFront the files have changed - CF is already wired up.

For our flyers this maybe ok, but for a new React build that would be a pain!

We need to tell CF to make an "invalidation" - i.e. to invalidate the cache of filenames it has and check again.

# Cloudfront invalidation

Move the S3 Deployment for the Flyers (`flyersDeployment` construct) to AFTER the CloudFront `flyersDistribution`. Careful, the names are similar!

Then update it as so, adding the `distribution` & `distributionPaths` lines where it says `//TODO`:

```
1 // Copy flyers into bucket – depends on both Bucket and Cloud
2 const flyersDeployment = new s3Deployment.BucketDeployment(th
3   'flyers-deployment', {
4     destinationBucket: flyersBucket,
5     sources: [ s3Deployment.Source.asset('../gig-flyers') ], //
6     retainOnDelete: false,
7     distribution: flyersDistribution, // invalidate this
8     distributionPaths: [ '/*' ], // invalidate everything
9     prune: true,
10    memoryLimit: 256, // in case folder is big
11  })
```

# Demo code review

Before we move on, lets check we all have about the same code!

Instructor to show the content of `./examples/cdk/lib/cdk-stack.ts`

## Extra things for CF

CloudFront can do many things. We already have it redirecting from HTTP to HTTPS.

One thing we can set is the default object to serve from S3. This is less relevant for the flyers, but will be important for React! When serving react we always want `index.html` to be the default thing.

But also - what if the URL we are given is incorrect? We can add handler functions to CF so that it corrects this for us. Very handy!

# Code along - Attaching functions

First we define a CF function - add this to the end of the `constructor`:

```
// Redirects lambda to send folks to the right place in cloudf
const redirectsFunction = new cloudfront.Function(this,
  'redirects-function',
  {
    code: cloudfront.FunctionCode.fromFile({
      filePath: "functions/redirects.js",
    }),
  },
)
// CLIENT DISTRIBUTION + R53 FOR REACT WOULD GO HERE //
```

# Code along - Client Distribution

We need the Client Distribution to be at the end.

Luckily the last slide gave us a hint where to put it.

```
// CLIENT DISTRIBUTION + R53 FOR REACT WOULD GO HERE //
```



# Code along - Client Distribution

Add this at the comment we just found:

```
// CLIENT DISTRIBUTION + R53 FOR REACT WOULD GO HERE //
// Cloudfront sits in front of S3 bucket
const clientDistribution = new cloudfront.Distribution(this,
  'client-distribution',
  {
    defaultBehavior: {
      origin: new origins.S3Origin(clientBucket),
      viewerProtocolPolicy: cloudfront.ViewerProtocolPolicy.RE
    },
    priceClass: cloudfront.PriceClass.PRICE_CLASS_100,
    defaultRootObject: 'index.html',
    domainNames: [ fullDomain ],
    certificate: cert,
  }
)
```

# How to set CF extras

We can now extend the `defaultBehavior` settings of the CF distribution (in addition to all the other settings), to handle react nicely:

```
const clientDistribution = new cloudfront.Distribution(this,
  `flyers-distribution`,
  {
    defaultBehavior: {
      // -- other stuff then... -- //
      functionAssociations: [ // extra bit here
        { // redirects handler
          eventType: cloudfront.FunctionEventType.VIEWER_REQUEST
          function: redirectsFunction,
        },
      ],
    },
  },
);
```

# CF and query parameters

We can also tell CF to pass query parameters to our React code, like so:

```
const clientDistribution = new cloudfront.Distribution(this,
  `client-distribution`,
  {
    defaultBehavior: {
      // -- other stuff then -- //
      // allow query params from CF into the lambdas
      originRequestPolicy: // extra bit here
        new cloudfront.OriginRequestPolicy(
          this,
          `request-policy`,
          {
            queryStringBehavior:
              cloudfront.OriginRequestQueryStringBehavior.all(
            )),
          },
    })
```

# Sanity check - CF Client Distribution

Check your `clientDistribution` against that shown by the instructor from `solutions/cdk/lib/cdk-stack.ts`

# Breakouts - 20 mins

In your breakouts finish the `clientXXX` constructs and deploy your stack:

- Move `clientDeployment` after `clientDistribution`
  - Then add invalidation settings (`distribution` and `distributionPaths`)
- Add a `route53.CnameRecord` for the `clientDistribution`
  - Use the `fullDomain`
  - You do not need another `zone` lookup
- Add an output called `ClientUrl` for `https://${fullDomain}`
- Logon to aws and re-deploy
- Test you can see the frontend at <https://yourname-gigs.ngai-sot.academy> !

## Emoji Check:

Did you manage to create the `clientDistribution` and see the react online?

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

## More to do

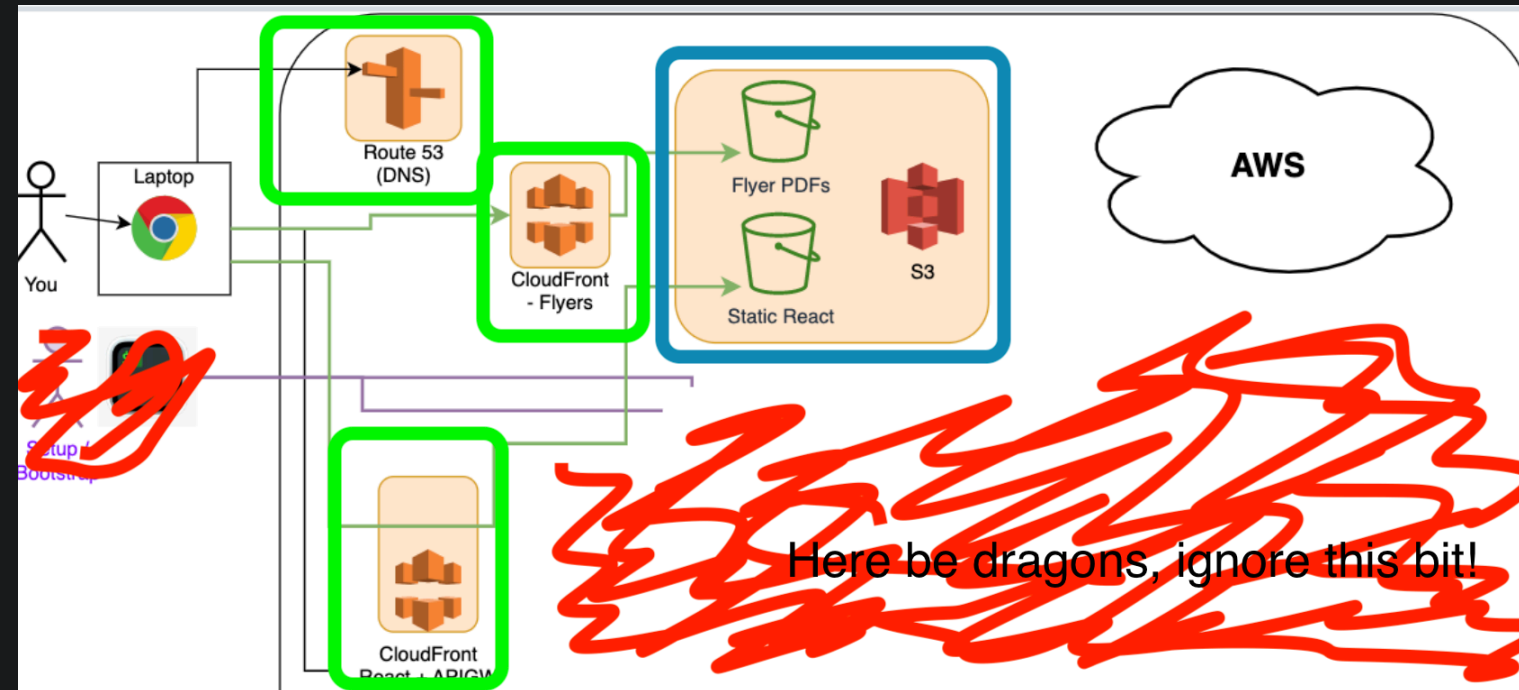
When you did see your react online, you may notice the UI can't connect to the backend?

Well, we have not made a middle layer yet, let alone the backend.

For that... we need more sessions 😊

# What we just made

Well done!





# Overview - recap

- Continue with IAC (Infrastructure as Code)
- Use AWS CDK
- Put CloudFront (CF) in front of S3 to access our files
- Add Route53 to give CF a friendly DNS name
- Make and run an example. Twice.

# Objectives - recap

- Understand what CloudFront (CF) is
- Use the CF CDK construct in JS
- Understand what Route53 does
- Use Route53 to give a CF distribution a nice name
- Access files in S3 via CF

## 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