Serverless Vector Tiles on AWS

Mark Varley <mark@addresscloud.com>

Steps

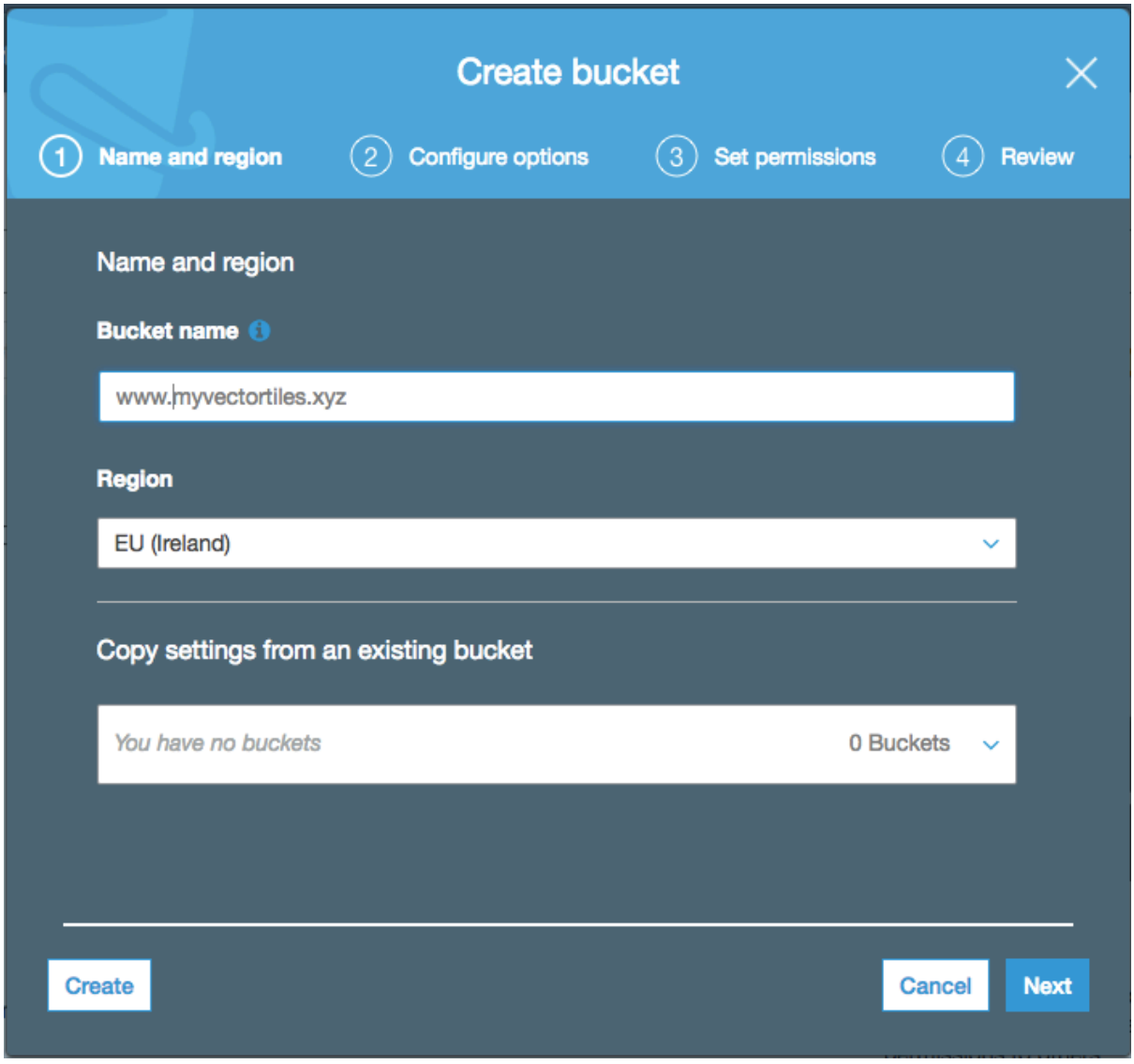
1. Host a simple webpage on AWS (Hello S3!)
2. Host the webpage from our own domain (Hello Route 53!)
3. Create a free HTTPS cert for our domain (Hello ACM!)
4. Publish our simple webpage to a CDN with SSL support (Hello CloudFront!)
5. Add a map (Hello Vector Tiles!)
6. Publish ready-made OSM tiles to S3/CloudFront
7. Publish our own geo data to S3/CloudFront

Pre-Requisites

- An AWS account
- A domain name that can be used for the exercise (this will need to be re-pointed so make sure it is not being used for anything else!). namecheap.com have domains starting at $0.48 per year but make sure you switch off auto-renew!
- Patience – there is a lot of AWS config to make this work and no sign of a map until Step 5 but this will be worth it, trust me

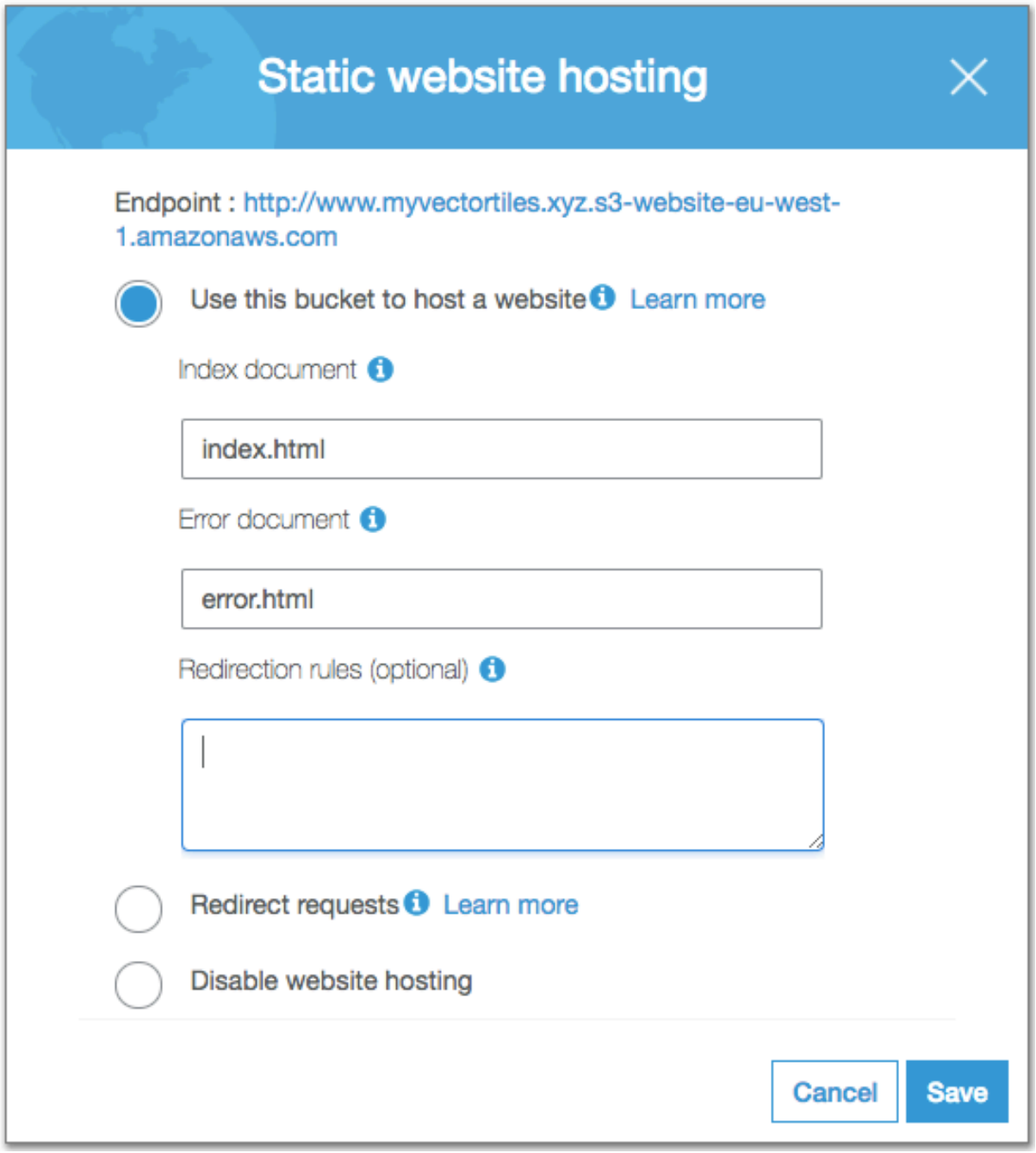
1. Host a simple webpage on AWS (Hello S3!)

First we need to create our bucket, log on to the AWS console and select S3 (make sure you are in the correct region – check the top hand corner of the screen). Click Create Bucket. We need to give the bucket the same name as our domain name, we will prefix ours with www. as we are going to create some more subdomains later, ours will be www.myvectortiles.xyz



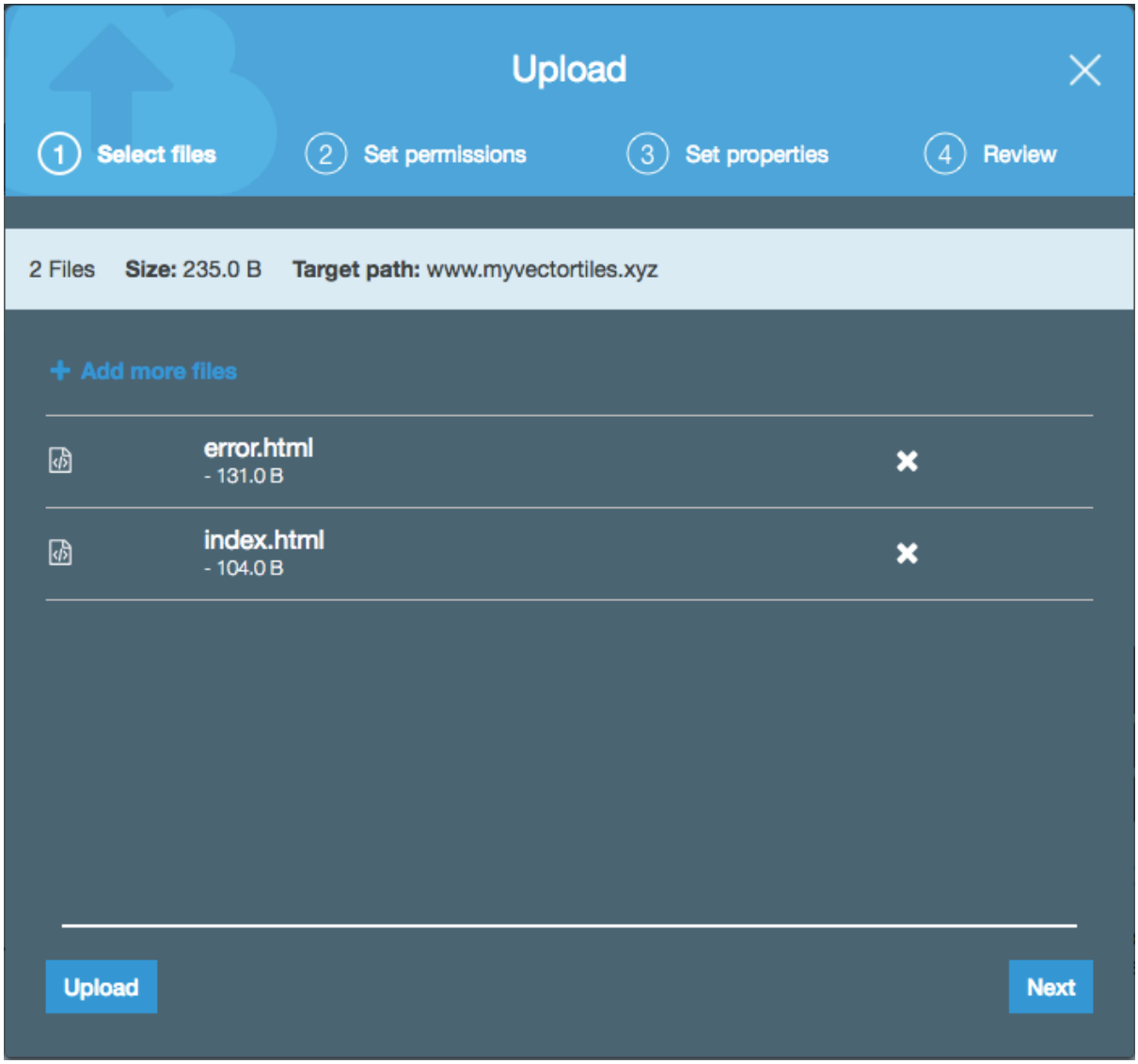
Accept all default options and then click Create.

We now need to tell Amazon that we want to serve a webpage out of our bucket. Click on the newly created bucket, click Properties and then click Static Website Hosting and configure as below.

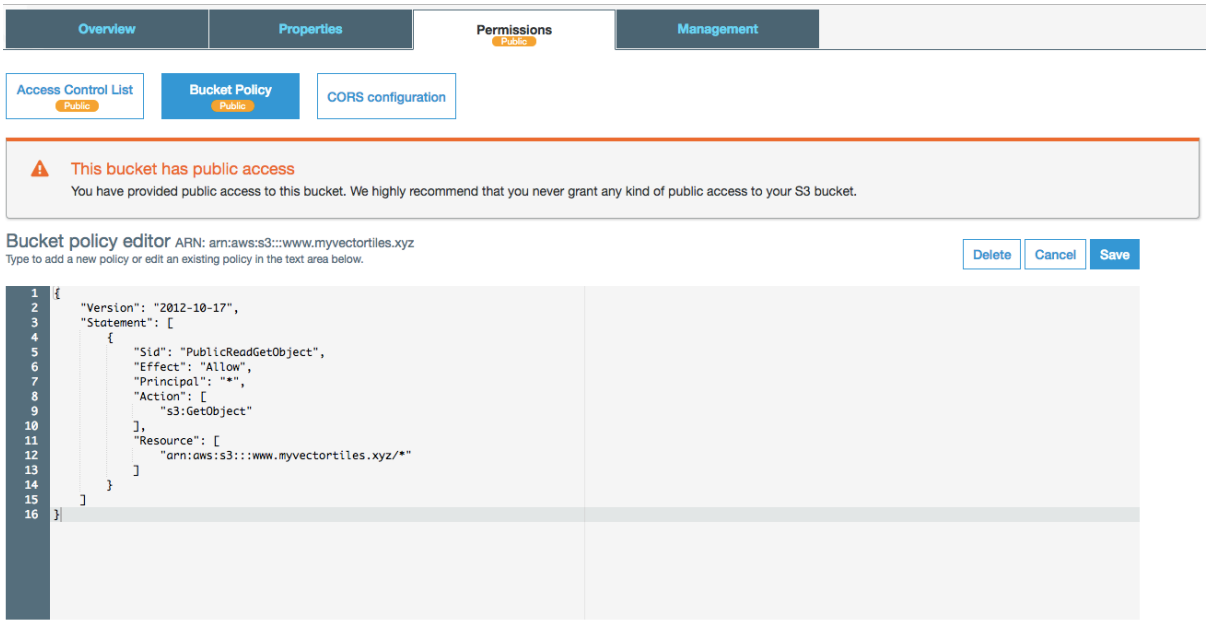


We now need to upload some content to our bucket. You can create your own index.html and error.htm files or use the simple examples in the repo at https://github.com/addresscloud/serverless-tiles/tree/master/01_HelloS3

Upload the files to the bucket:



Finally we need to open up our bucket to the outside world. We will set a public read access policy, it is advisable to refine this using AWS best practices for production use cases. Click on the bucket again, select permissions and configure a Bucket Policy as per the one below which can be found at https://github.com/addresscloud/serverless-tiles/blob/master/01_HelloS3/Policy.txt - be sure to amend to your own domain name



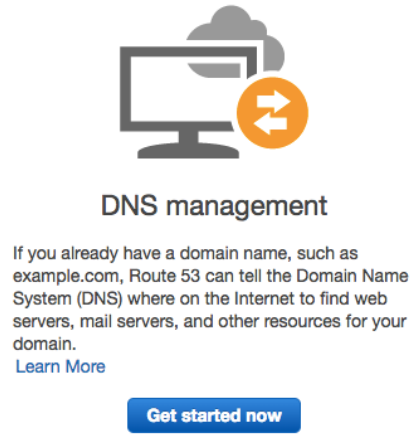
We should now have our publicly facing masterpiece of a website up and running. To test it go to http://www.YOUR_DOMAIN_NAME.s3-website.YOUR_REGION.amazonaws.com/, ours is http://www.myvectortiles.xyz.s3-website.eu-west-1.amazonaws.com/

Hello S3

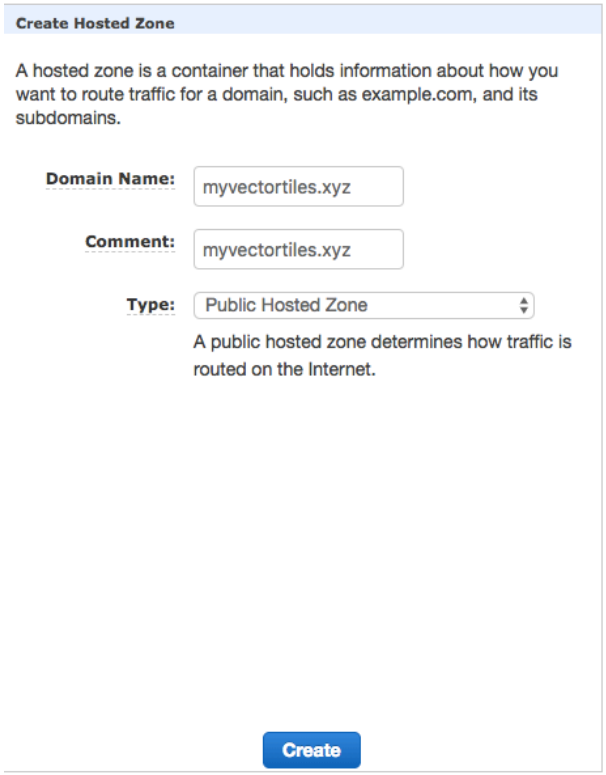
Wow! We can party like it's 1994

2. Host the webpage from our own domain (Hello Route 53!)

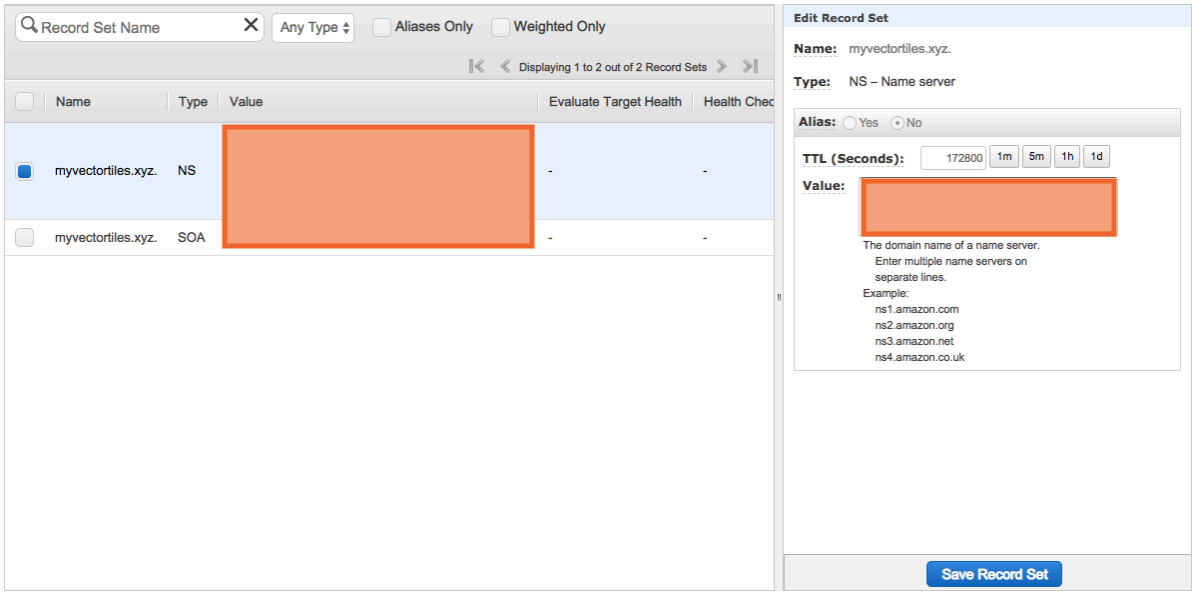
OK, we have a website which the world can see but we want to use our own domain. There are a number of options here but the most elegant makes use of Amazon's excellent Route 53 service. To use Route 53 we need to tell Amazon about our domain, let's do that now. Navigate to the Route 53 page in the Amazon console, if you are a new user you might see a fancy splash screen. We are going to be managing DNS so select that option



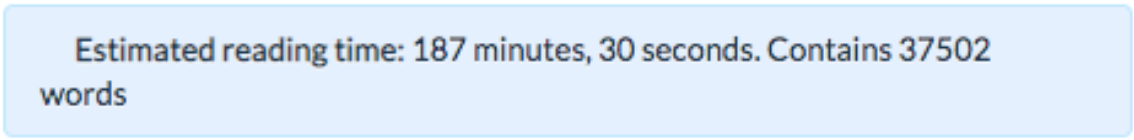
You should now see a screen with a bunch of options on the left, select Hosted Zones and then Create Hosted Zone and give it the name of your domain (without the www bit):



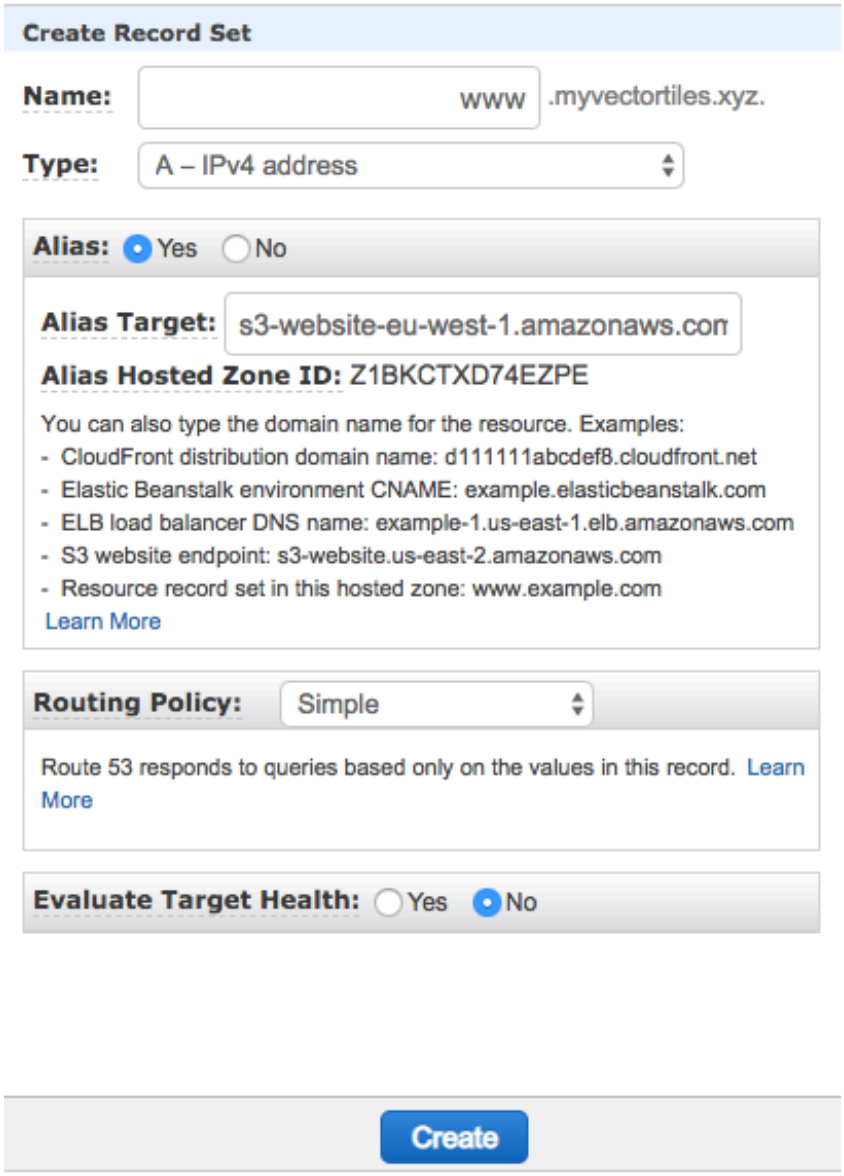
Amazon creates us a new "zone" domain and generates some nameserver entries for us. You will need to copy and paste these and follow the guidelines from whoever you bought your domain from to update the nameservers.



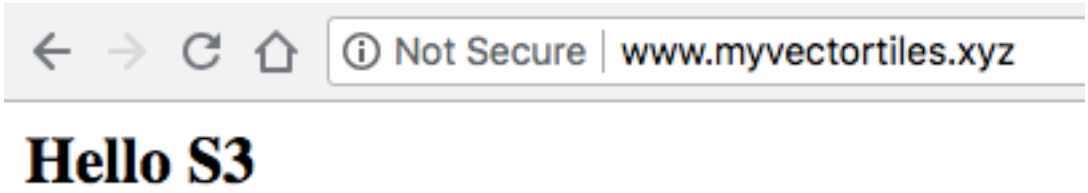
Be very careful here, if you are using that domain for anything else it will stop working once you update the domain servers! Once you have made that change it will take some time (in some cases several hours) for the changes to propagate (be sent to name servers around the world). If you are looking for a way to pass several hours I highly recommend the AWS Service Terms which are not only a riveting read but should take you the best part of 3 hours https://aws.amazon.com/service-terms/



We now need to associate the S3 bucket we created earlier with our Route 53 domain name. To do this we will create a Record Set in our Hosted Zone. Click Create Record Set and enter www as our Name, leave Type as “A” and click Yes next to Alias. This will use a special AWS routing to associate the subdomain with an AWS service, click in the Alias Target box and you should see a list of Target options, one of these should be the S3 website we created earlier.

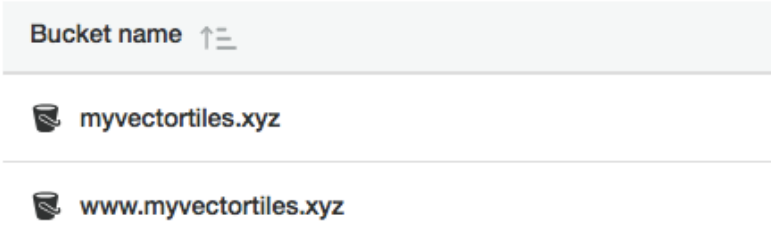


If, by this point your changes have propagated you should be able to navigate to your domain name (don't forget the www) and see your webpage:

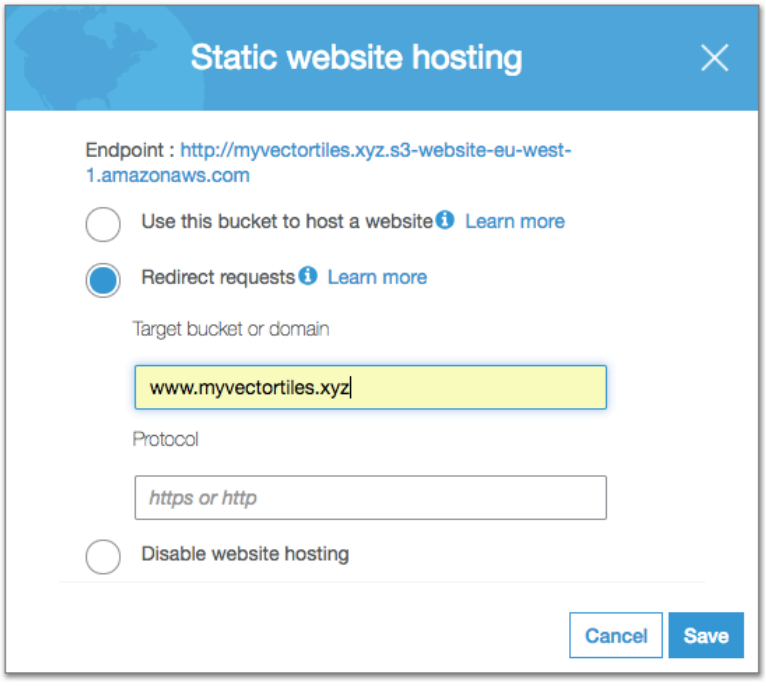


Magic! But what if we forget to type www?

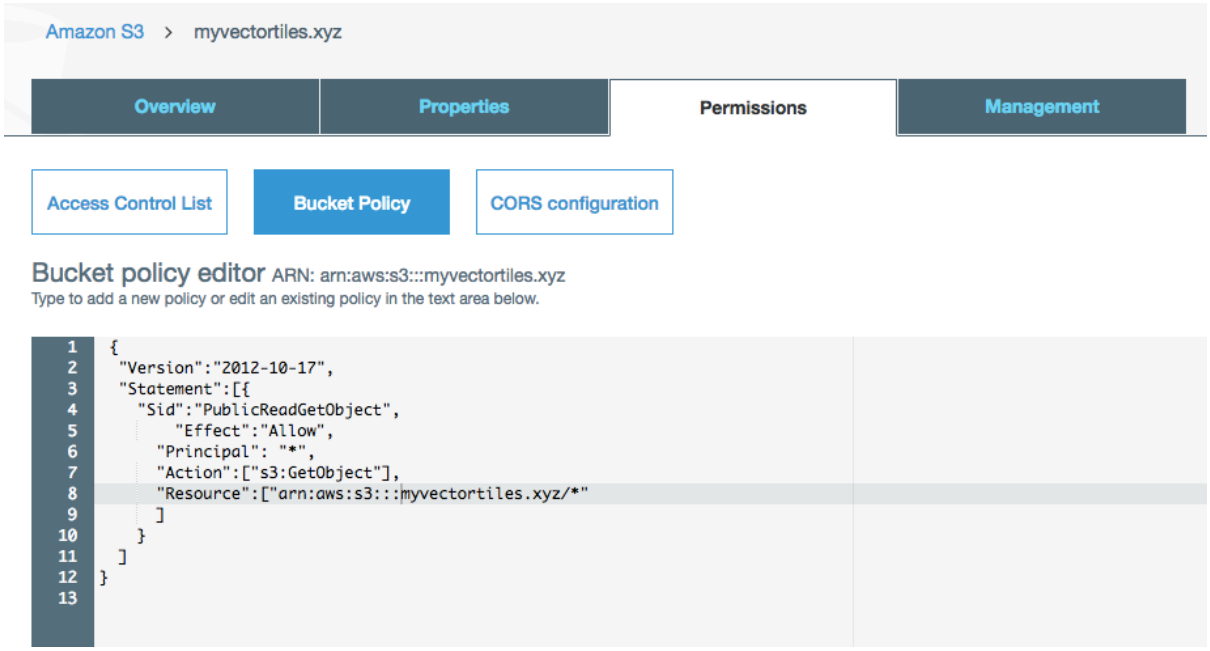
Ah, not so good. We need to re-direct calls to our root domain to our www subdomain, easy! We go back to S3 and create a new bucket, this time as our domain name without the www.



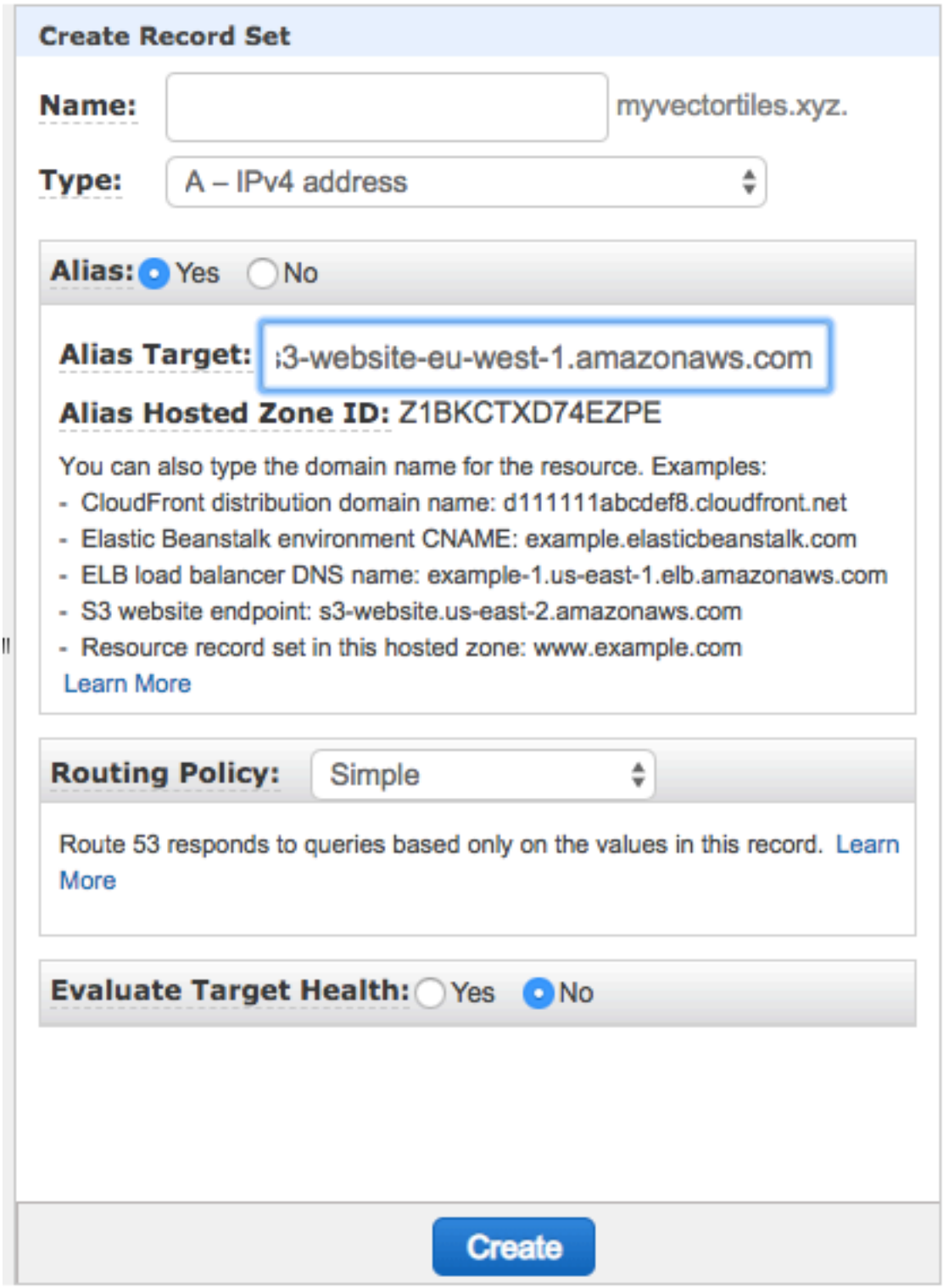
We click on this new bucket and select Properties, then under Static Website Hosting setup to redirect requests to the other bucket:



We need to make the bucket public as we did before, we can use the same policy but be sure to delete the www. from the policy record:



Finally we need to tell Route 53 to go to that domain when someone makes a request to the root domain so we create new Record Set with details as follows (we leave the Name blank) (Note: you may need to refresh the Route 53 screen to pickup the S3 website you just created):



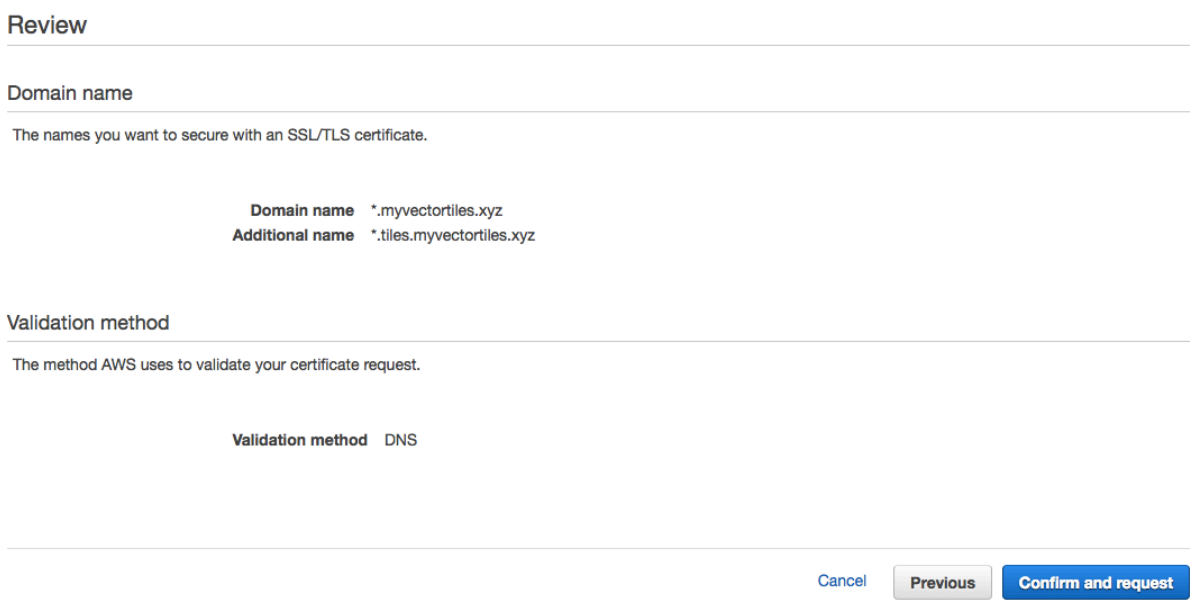
Now when we hit our root domain e.g. http://myvectortiles.xyz/ it should redirect us to http://www.myvectortiles.xyz/ - it may take an hour or so for this rule to start working as DNS information is cached so be patient if it doesn't work straight away.

3. Create a free HTTPS cert for our domain (Hello ACM!)

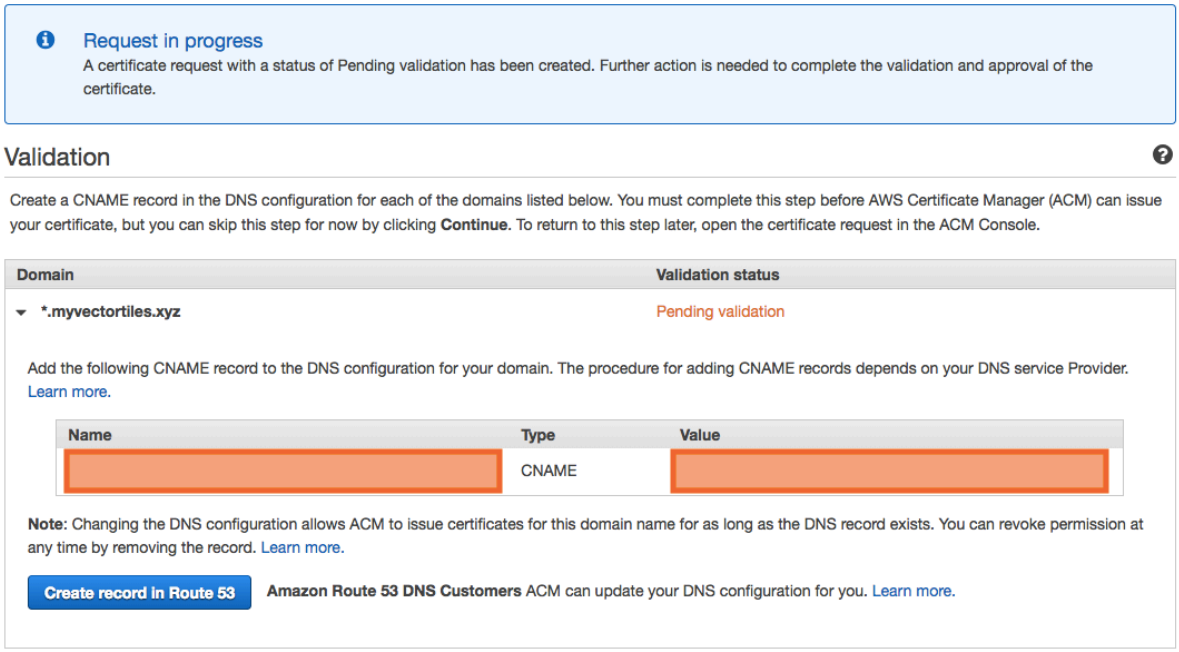
We have our website, hosted on our own domain but it's not secure. We all know HTTPS certs cost a fortune, especially wildcard root certs that we can use for any subdomain on our fancy domain name, right? Wrong! AWS changed all that when they launched Amazon Certificate Manager (ACM) a couple of years back. Now all we need to do is request a cert, click a few buttons and let AWS do the rest, they even handle renewals for us. All this will cost you absolutely nothing.

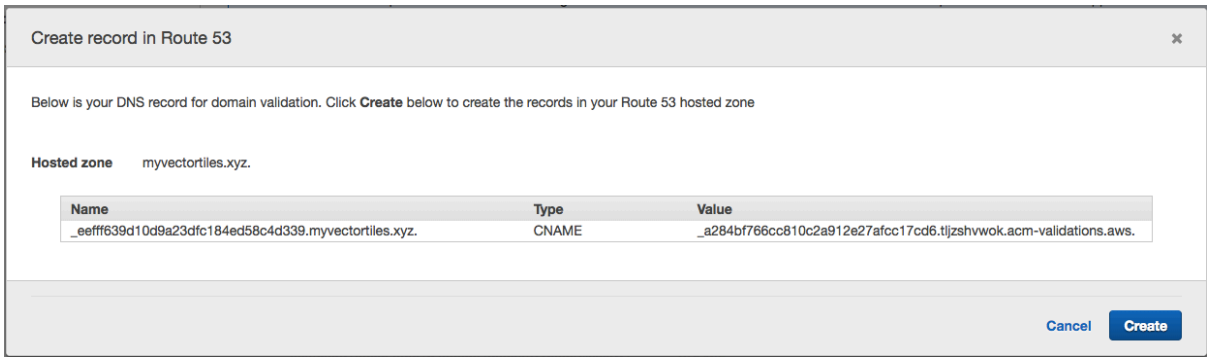
Go to the AWS Console, **make sure your region is set to US East regardless of what region your normally use!** Now select Certificate Manager then, Provision Certificates. Select "**Request a public certificate**" then continue. Enter the domain name as *.YOUR_DOMAIN so for us *.myvectortiles.xyz. We also need to add another domain name, the reason for this will be explained further on. We add *.tiles.myvectortiles.xyz.

Click next and leave DNS validation as the default. Click Review then Confirm and Request and then Continue:

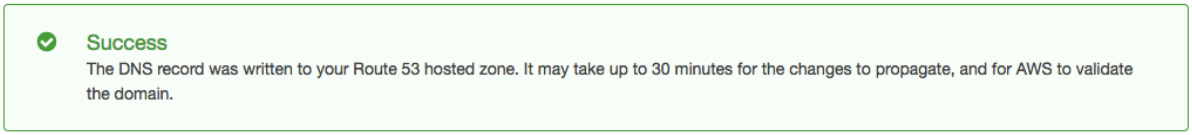


We now need to setup our DNS to tell AWS that we own this Domain. Click the expand arrow and there is a load of information about CNAME records, we can skip this and just click the big blue Create record in Route 53 button and we should be all set.

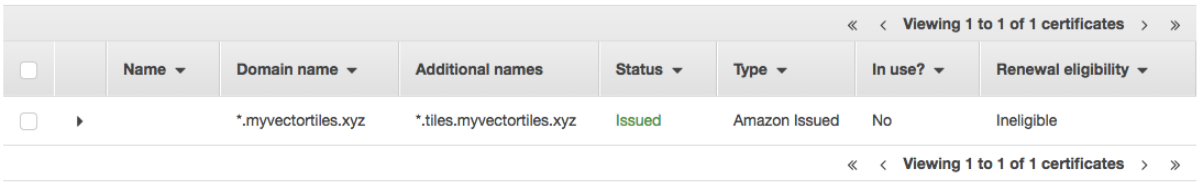




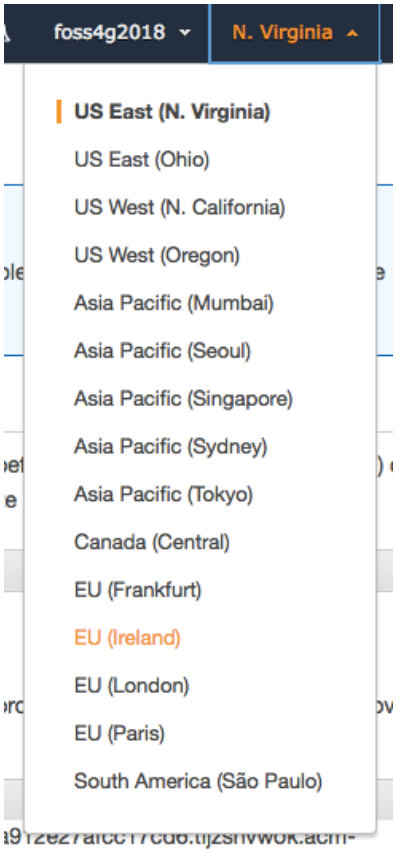
Awesome. We are in business!



Repeat this process for *.tiles.myvectortiles.xyz. Once the certificate is issued we will see the status below:



Now before we get let's go back from Virgina to where we were, in our case Dublin:


foss4g2018
N. Virginia
US East (N. Virginia)
US East (Ohio)
US West (N. California)
US West (Oregon)
Asia Pacific (Mumbai)
Asia Pacific (Seoul)
Asia Pacific (Singapore)
Asia Pacific (Sydney)
Asia Pacific (Tokyo)
Canada (Central)
EU (Frankfurt)
EU (Ireland)
EU (London)
EU (Paris)
South America (São Paulo)

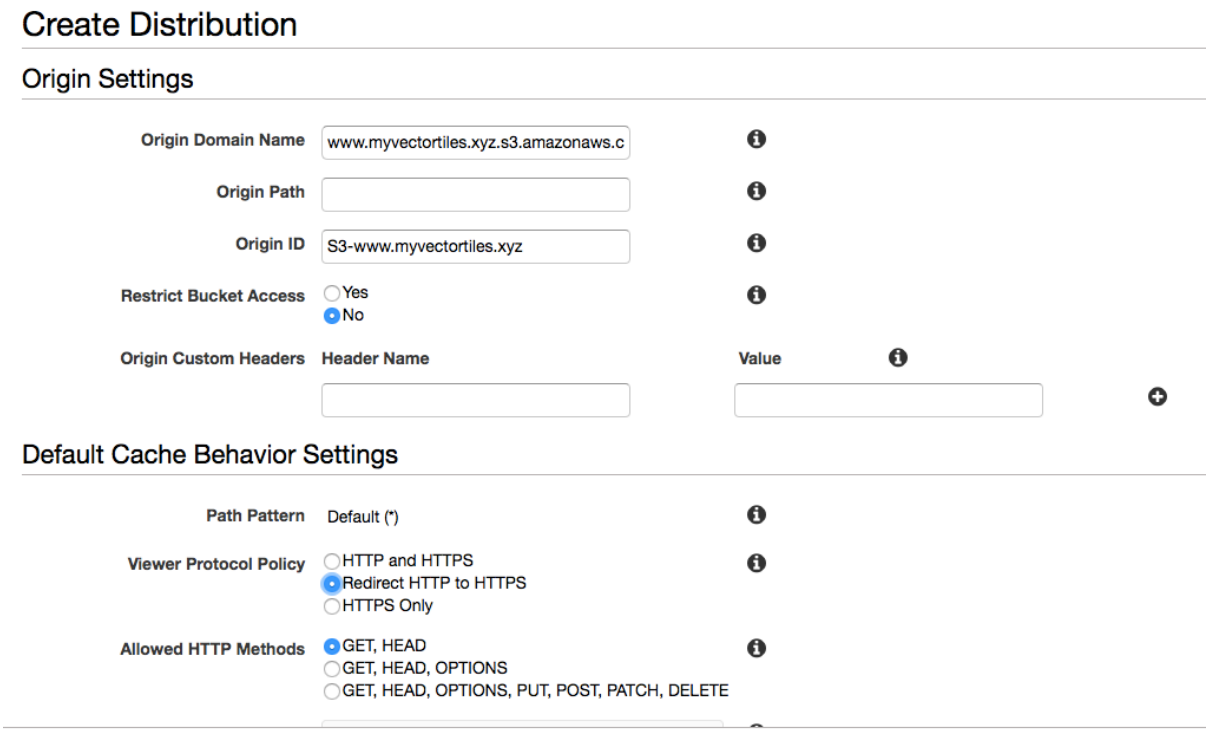
4. Publish our simple webpage to a CDN with SSL support (Hello CloudFront!)

So we have a website, hosted out of an AWS bucket (in our case in Dublin) on our own domain. If someone hits our root domain AWS knows to redirect to www. We have an SSL cert but we've not used it yet. Also, we have users all over the world who want to see out content but at the moment all calls go to our home region Dublin. So we want speed and we want security and we don't want to pay much / anything for it. Enter Cloudfront!

CloudFront is a Content Delivery Network (CDN) which takes content from our S3 buckets and publishes it to "Edge" locations (local caches) around the world. When a request goes to www.myvectortiles.xyz we want to serve the content from the nearest cache rather than going to our home region to get it. So if our user was on Bondi Beach we would want to serve them from Sydney and not Dublin. We also want SSL, CloudFront takes care of that too.

Let's go back to the AWS console and select CloudFront as our service. Notice that the region says Global as CloudFront like Route 53 is a global service. We click Create Distribution to get started. We aren't streaming media so we click Get Started under Web. We can complete the config as follows (make sure you use your own domain!).

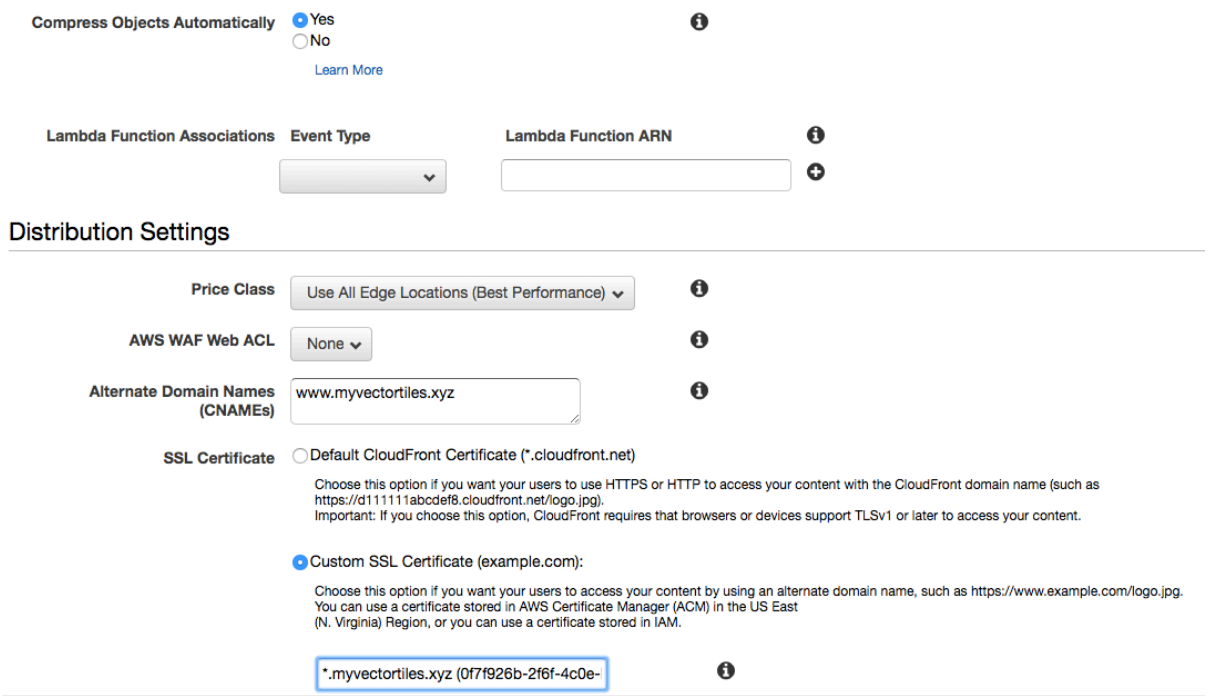
We click in Origin Domain Name and it should give us our list of buckets, we want to use the www bucket where we have our files. We also want to Redirect any non-secure HTTP calls to HTTPS:



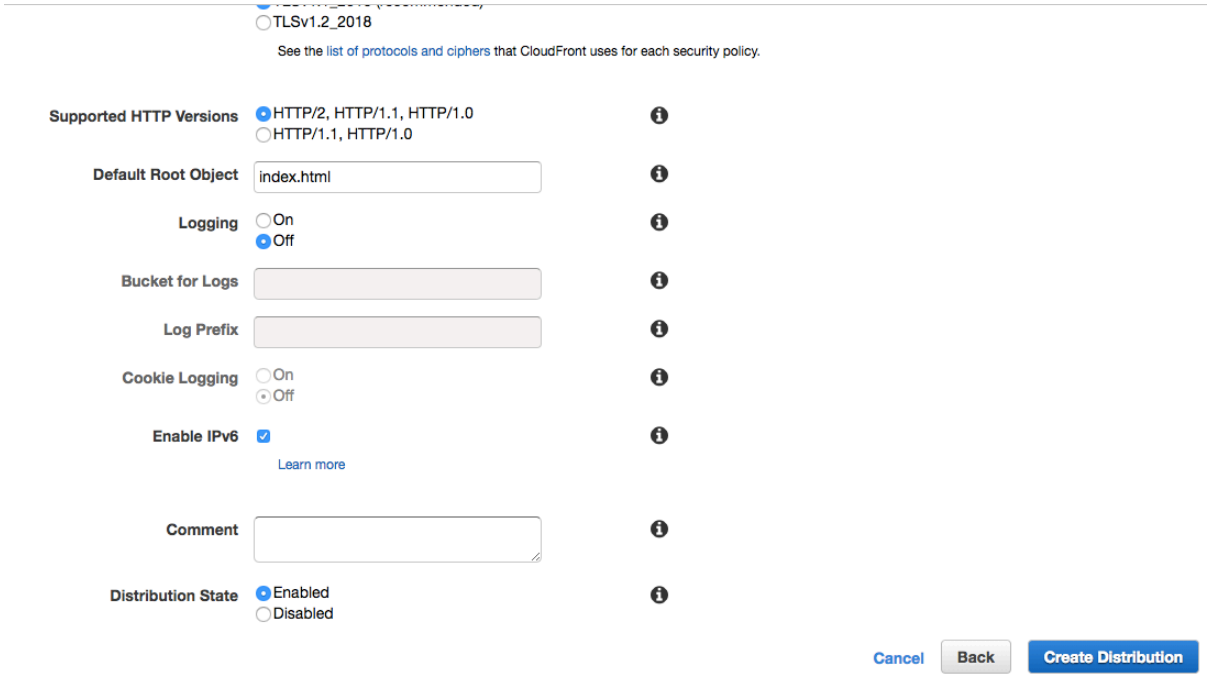
We want to compress objects automatically as this will come in handy once we start serving up big javascript files. Under Price Class we can choose to only host our content in part of the world to save money but we have a small application with a global user base so will leave as-is.

We want to use our Domain Name so we enter it as a CNAME.

We finally want to use that cert we created earlier so we select Custom SSL Certificate and then select it from the list:

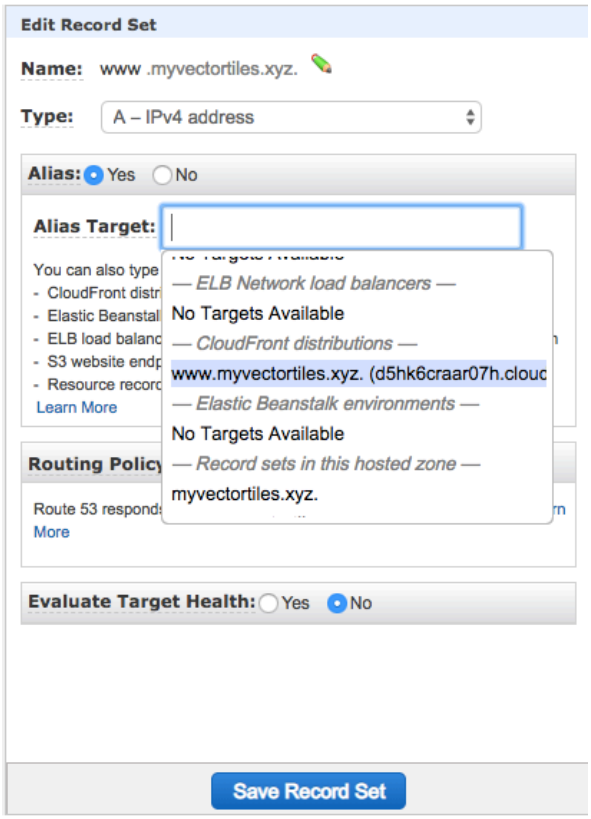


We want our default object to be our index page but we can leave everything else as defaults.



We can now go ahead and create our bucket, it will take 10-15 minutes to push out all that amazing content around the world!

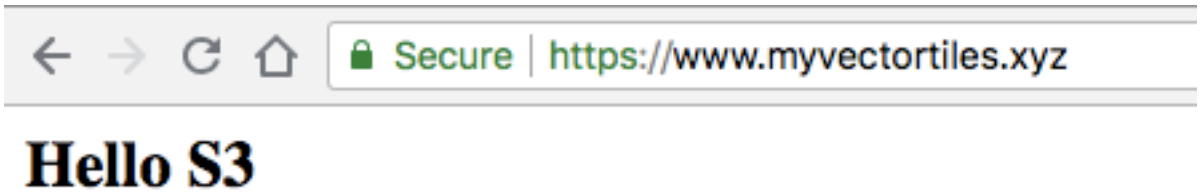
In the meantime we can configure Route 53 to point out www subdomain to our CloudFront endpoint rather than our S3 bucket. This is quite straightforward, go back to Route 53, refresh the page, and click on the www. myvectortiles.xyz Record Set. Click on the Alias Target box and scroll down the list and we should see a new entry for our CloudFront distribution:



We select this and save the record set.

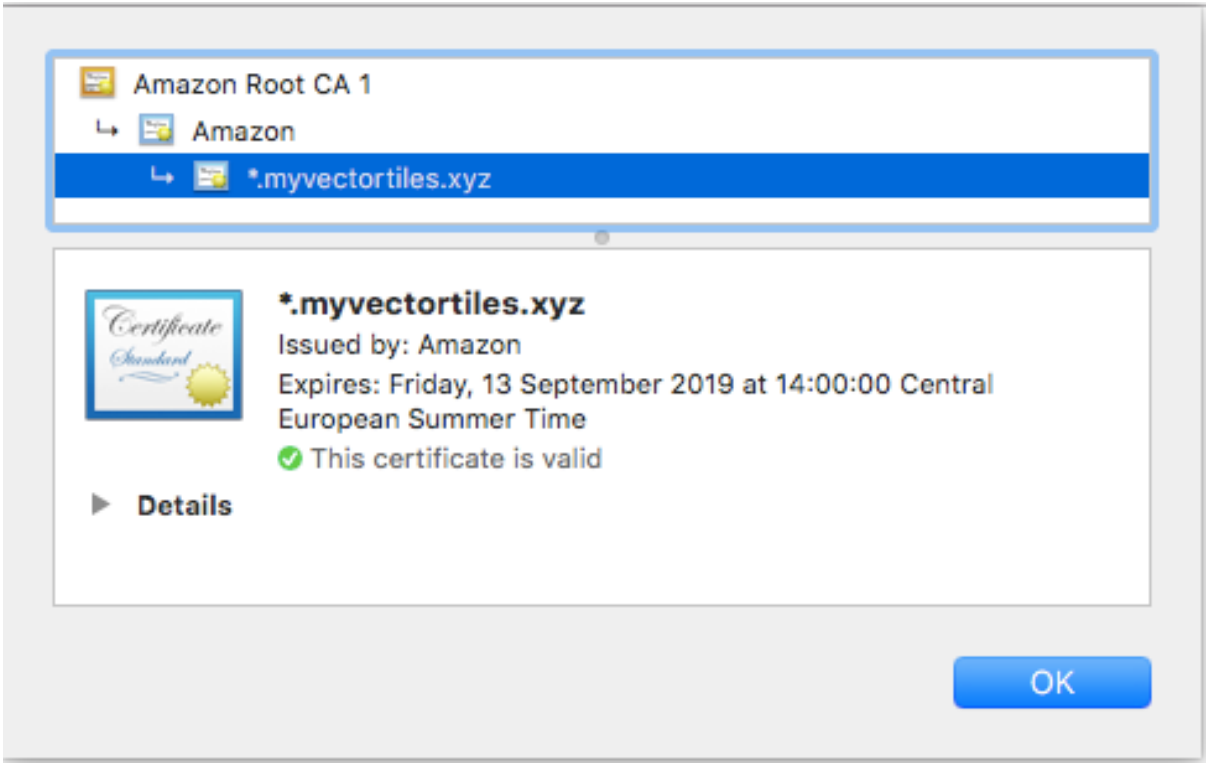
Let's see how our CloudFront distribution is getting on. Once it is ready we should see the Status update from In Progress to Deployed.

Let's go back and check our URL, hmm our DNS has not updated yet. We will give it some time and try again:



Magic, we have SSL! If we try http://myvectortiles.xyz/ or http://www.myvectortiles.xyz/ we see the same thing, our CloudFront is working.

Let's check the cert:



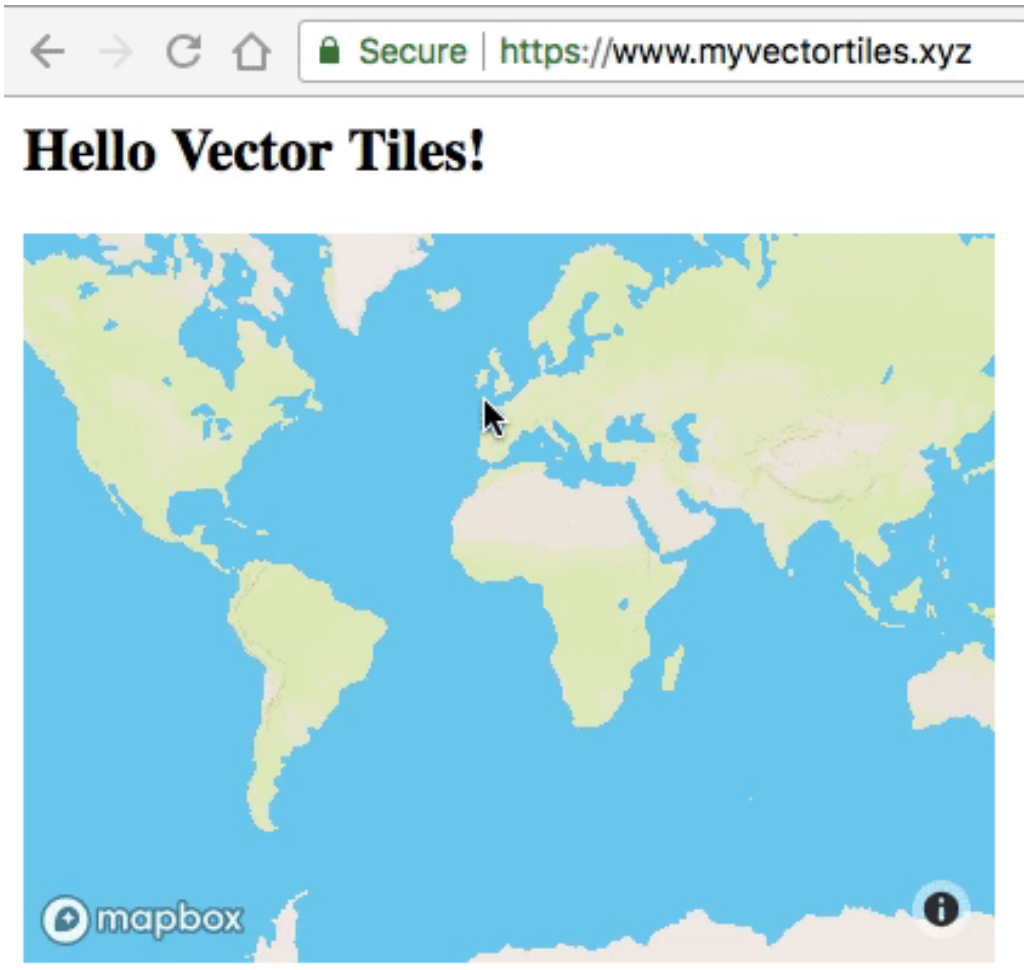
All good, Amazon has come through with a fully trusted HTTPS root certificate and it cost us nothing

5. Add a map (Hello Vector Tiles!)

Finally a map, what we came here for. OK with the AWS homework out of the way let's deploy our first map. This is a vector tiles tutorial so we need a client that is capable of rendering them. Mapbox popularised the use of vector tiles which had been around for some time when they published and open sourced their vector tile specification https://www.mapbox.com/vector-tiles/specification/ naturally they have excellent support with their Mapbox GL JS https://github.com/mapbox/mapbox-gl-js library (they also have native mobile versions). We could also use OpenLayers or a Leaflet plugin but the Mapbox library is excellent and has a very active community and we can use it commercially so it is a great fit.

Let's update our Hello S3 page with a map, we will follow the Mapbox quickstart at https://www.mapbox.com/mapbox-gl-js/api/. The new code can be copied from https://github.com/addresscloud/serverless-tiles/blob/master/05_HelloVectorTiles/index.html. We upload it to our S3 bucket replacing our original page.

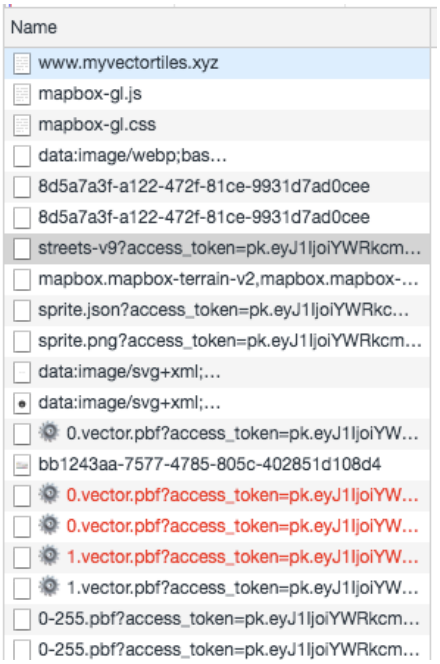
We have a map!



Not just any map but a map based on vector tiles, which means:

- It should be quicker to render
- We can style the tiles client side to support multiple customer requirements
- We can do lots of clever client side tricks such as 3D rendering
- We can "over-zoom" tiles to support more zoom levels and zoom in and out without having to re-query the server at every zoom level
- Our tiles are smaller and will take less space on our server and we only have to serve one set of tiles which can be used for multiple client side use cases

So what is actually happening here? Let's look at the Network Traffic:



1. We download our own client side code, which references Mapbox JS and CSS in the header, and a Mapbox style ()
2. We download the Mapbox client JS library and accompanying CSS
3. We download Style JSON file, this tells the client what to render, how to render it and where to find it, it must follow the Mapbox Style Specification: https://www.mapbox.com/mapbox-gl-js/style-spec/
4. We download the data source files for each source specified in the Style JSON file above, the data source files must conform to the TileJSON spec https://github.com/mapbox/tilejson-spec
5. The client then downloads any supporting images and tiles associated with the current style and viewport
6. As we pan and zoom around our map more tiles are pulled down but not for every zoom level, just where we need new data. If we look at a typical request we can see the https://a.tiles..../1/0/0.vector.pbf traditional Z/X/Y directory structure is being used
7. If we look at a few requests we notice that the subdomain flips between a and b e.g. https://a.tiles..../1/0/0.vector.pbf, https://b.tiles..../1/0/1.vector.pbf this is common technique to squeeze extra performance out of a service to get around browser parallel call limits

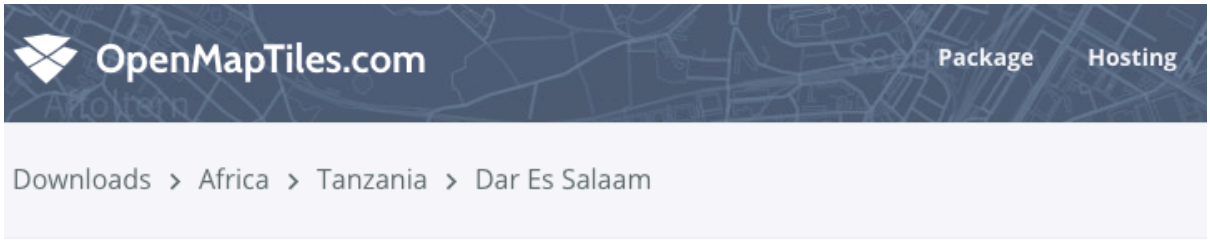
So, our requirements are as follows:

- We need to host some pbf (binary) files using a known directory structure
- We need to host some JSON config files (style.json and a tileset.json file)
- We need to support HTTPS
- We need to support multiple subdomains to optimise performance

Well, we already know that CloudFront can cope with all of these requirements so let's get started!

6. Publish ready-made OSM tiles to S3/CloudFront

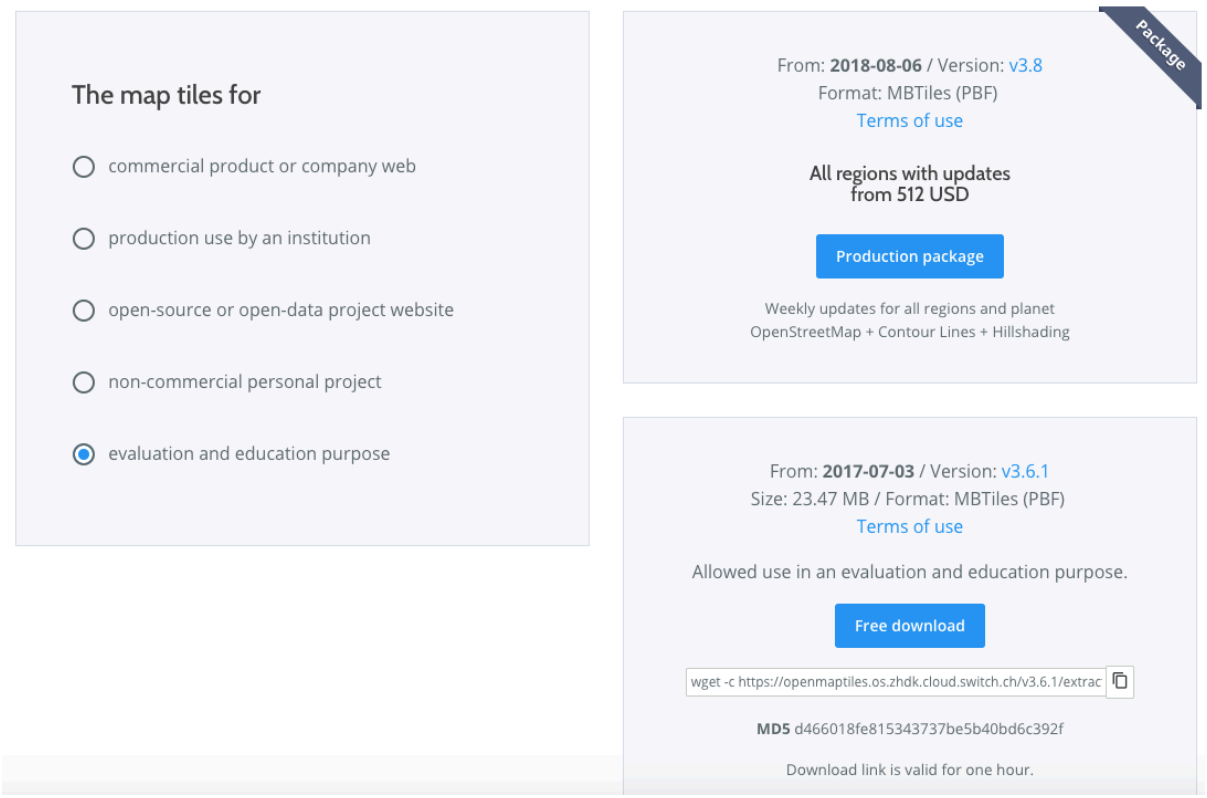
To prove our concept we will start with a ready made set of vector tiles. OpenMapTiles.org provides a fantastic resource of pre-packaged OpenStreetMap vector tiles and sample styles. There is a free-tier for non commercial use such as ours however for commercial use there is a very reasonable fee. We are going to download the OpenStreetMap tiles for the city of Dar Es Salaam.



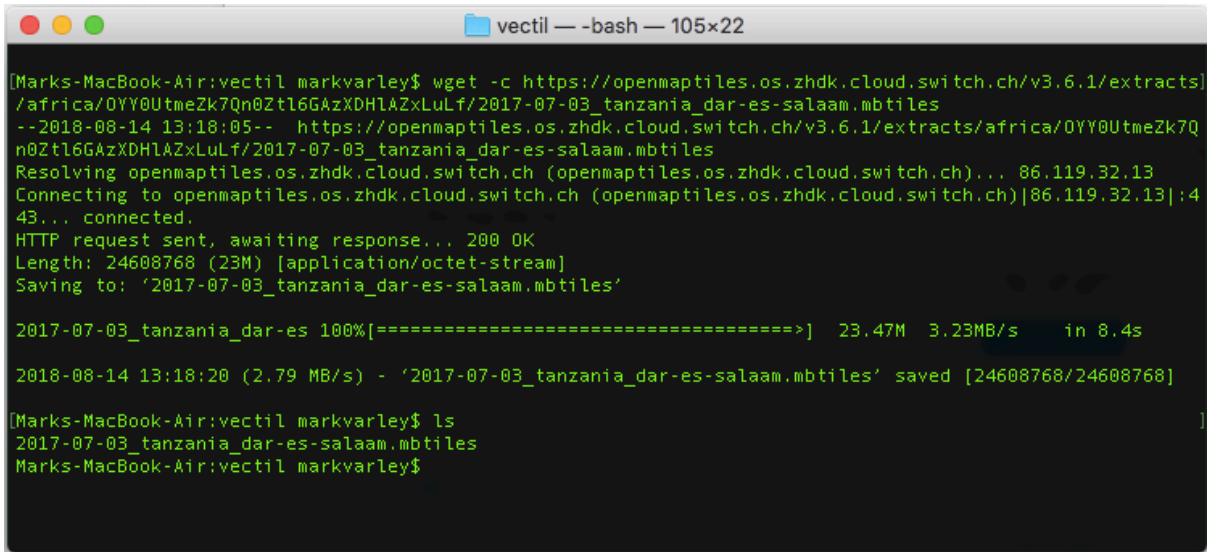
Dar Es Salaam
Vector and raster map tiles

Bounds 38.894, -7.12, 39.661, -6.502
OpenStreetMap | Contour lines | Hillshade | Satellite

We choose evaluation and educational purposes, we get a link to download the file once we have signed in. Note that a commercial license for this data is only 10 USD, a very fair and affordable price you will agree!

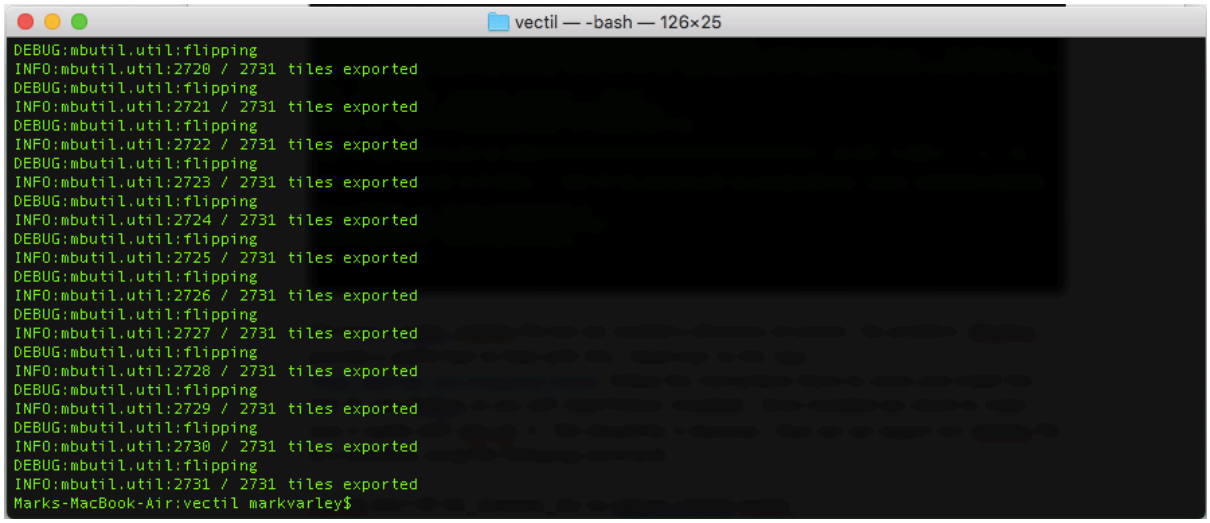


We download the file using the supplied wget command (on a mac you can install wget using homebrew (https://brew.sh/) with `brew install wget`)

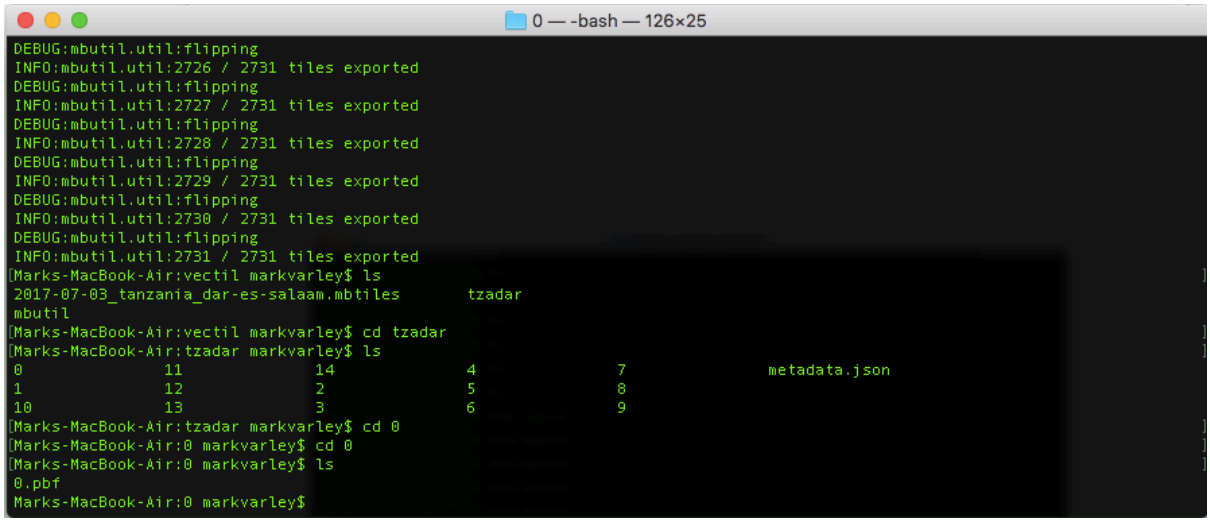


We have a single .mbtiles file but we needed a directory structure. No problem, Mapbox provide a useful tool to help with this. Head over to the repo https://github.com/mapbox/mbutil, follow the instructions there to clone and install the tool (it uses Python so you will need Python installed). Once installed we check to make sure it works with `mb-util -h`. We should be in business. Now we can export our mbtiles file to a directory using the following command:

```
mb-util 2017-07-03_tanzania_dar-es-salaam.mbtiles tzadar --
image_format=pbf
```

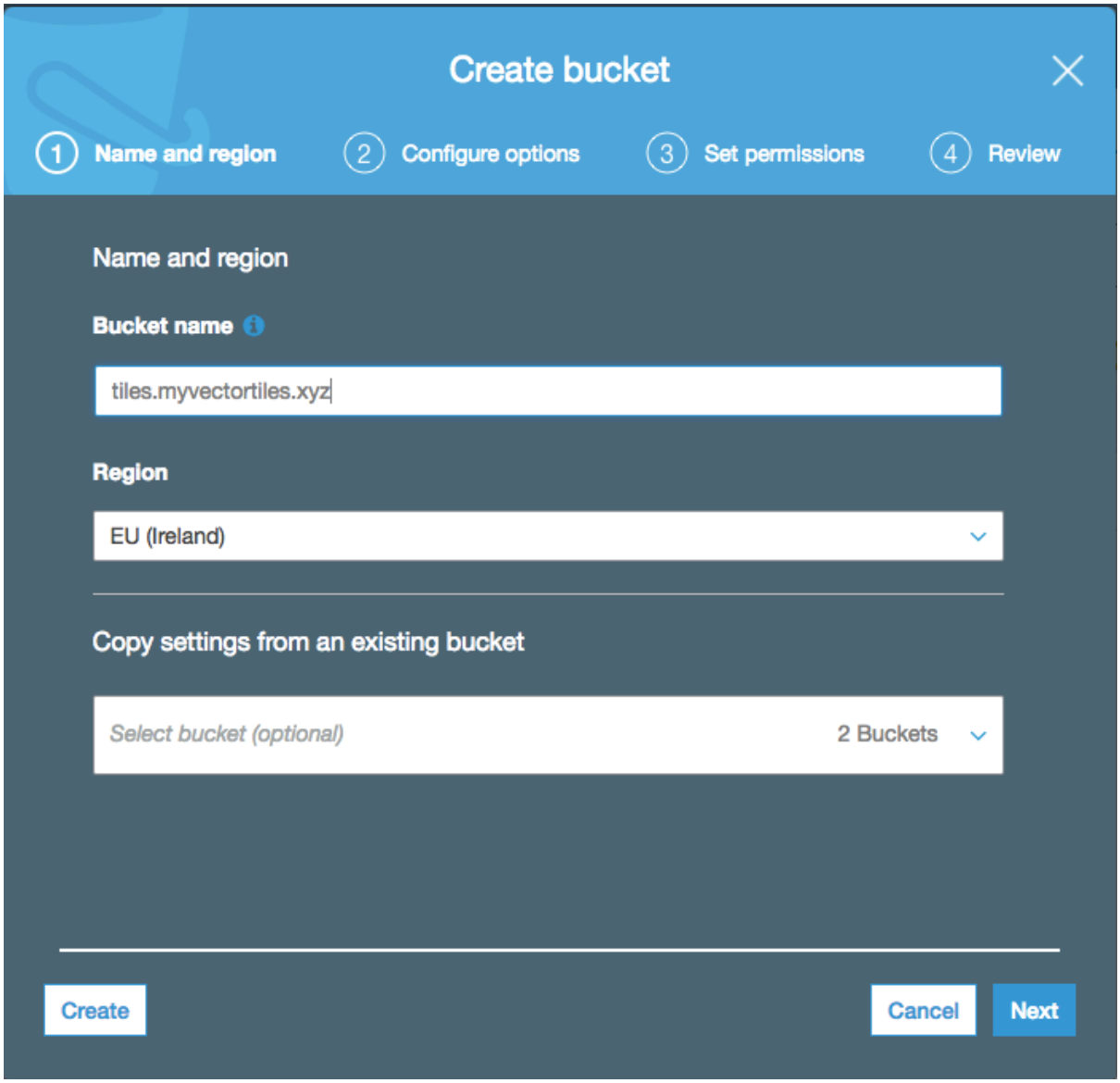


We get lots of encouraging progress messages. Let's see what we have just created:

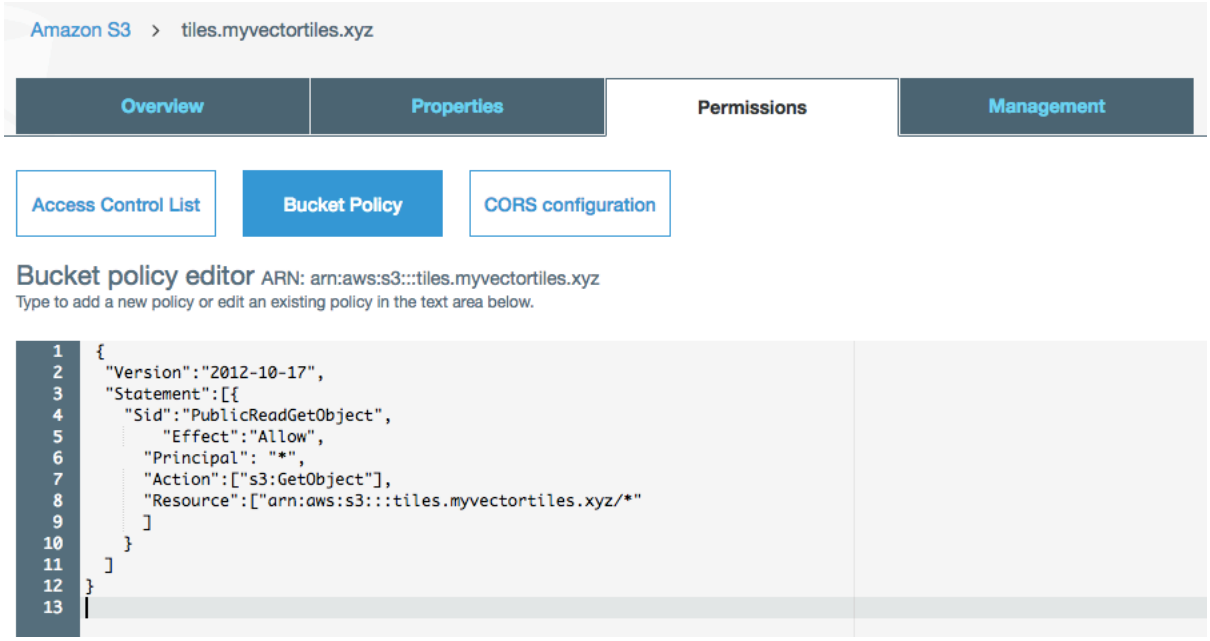


This looks very promising, we have a directory structure that looks like what we need, if we drill through the 3 tiers we found a pbf file as expected. It looks like we are in business.

We need to create a new S3 bucket to host our tiles. This is going to be a CloudFront website so we follow the same process as before:

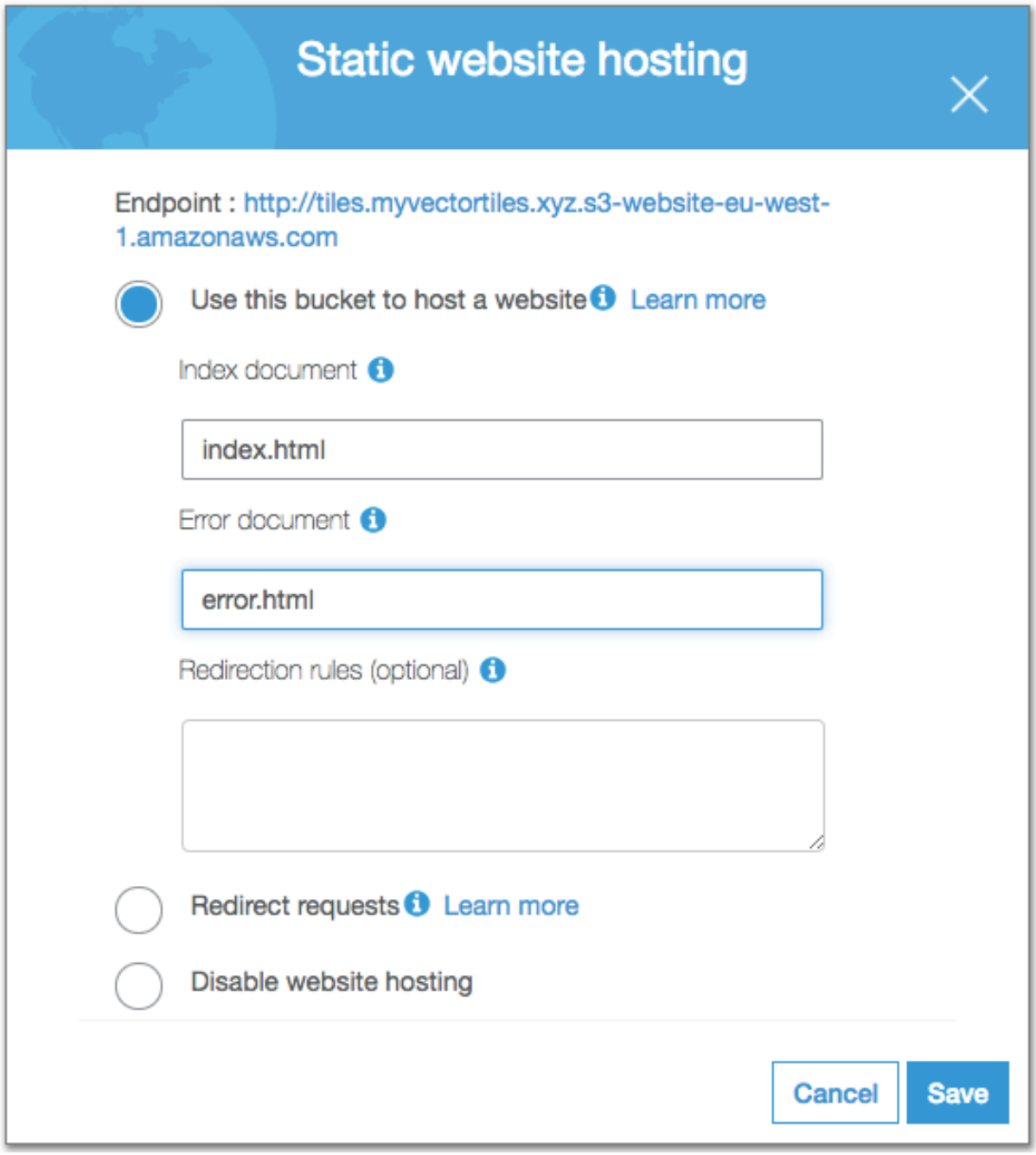


As before we also need to configure public access:



We also need to setup CORS configuration, this is to allow us to make requests for files from within our mapping application. The config file can be found at https://raw.githubusercontent.com/addresscloud/serverless-tiles/master/06_OSMVectorTilesAWS/CORS.txt :

Now we need to tell Amazon that this is going to be a website:

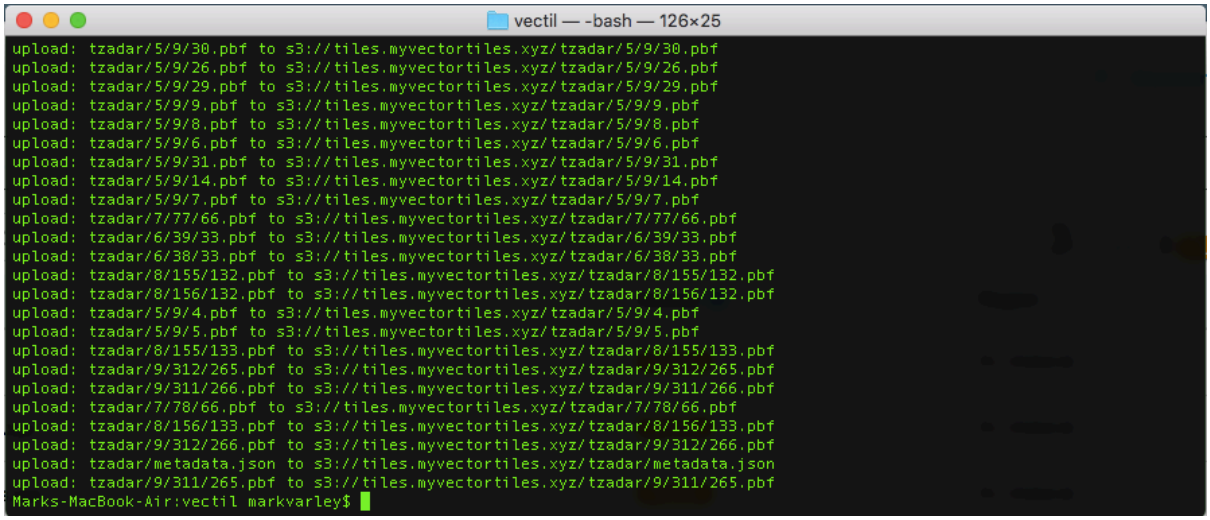


Let's go ahead and upload this directory to our new bucket in S3. We have a lot of files and a complex directory structure to upload so rather than using the console we will use the AWS command line interface. Instructions for installing and configuring the interface can be found here https://aws.amazon.com/cli/. You will also need to create an amazon user for your account using the Identity and Access Management (IAM) with permissions to write to S3. Instructions for doing this are here https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users_create.html

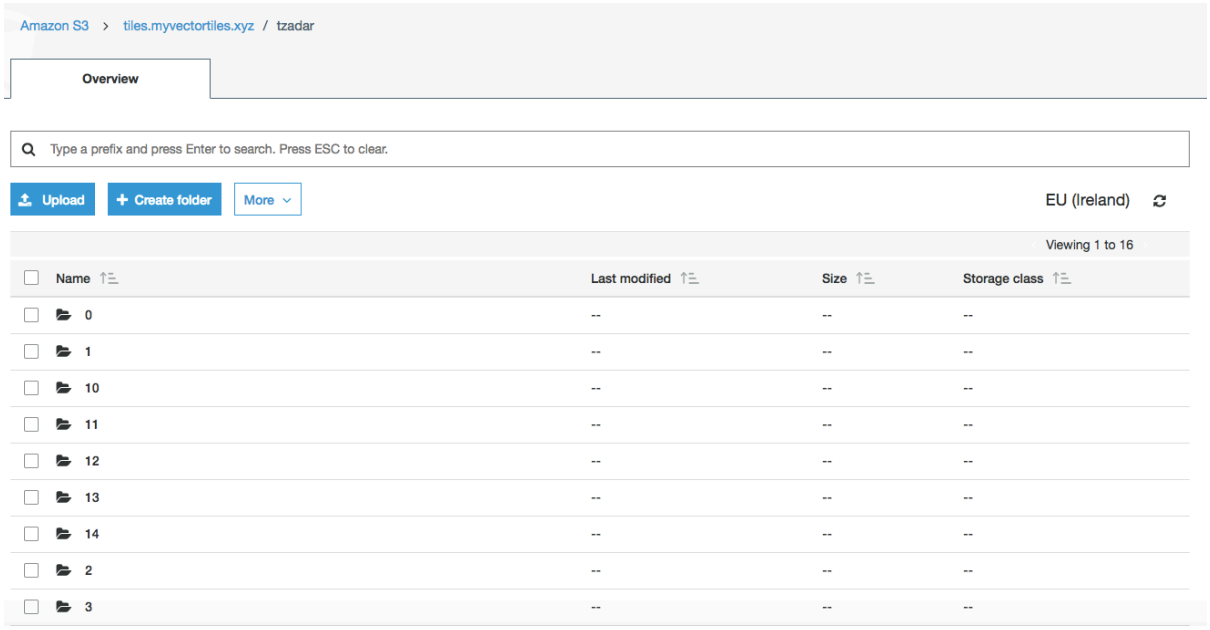
Before we upload the files let's think back to what we said earlier, the format for the vector tiles is .pbf (Protocol Buffers) which are compressed binary files. We need to tell Amazon that our files are already compressed so it does not try and compress them again and we also need to let it know the file type. We can do all that with a single command:

```
aws s3 cp tzadar s3://tiles.myvectortiles.xyz/tzadar/ --
recursive --content-type application/x-protobuf --content-
encoding 'gzip'
```

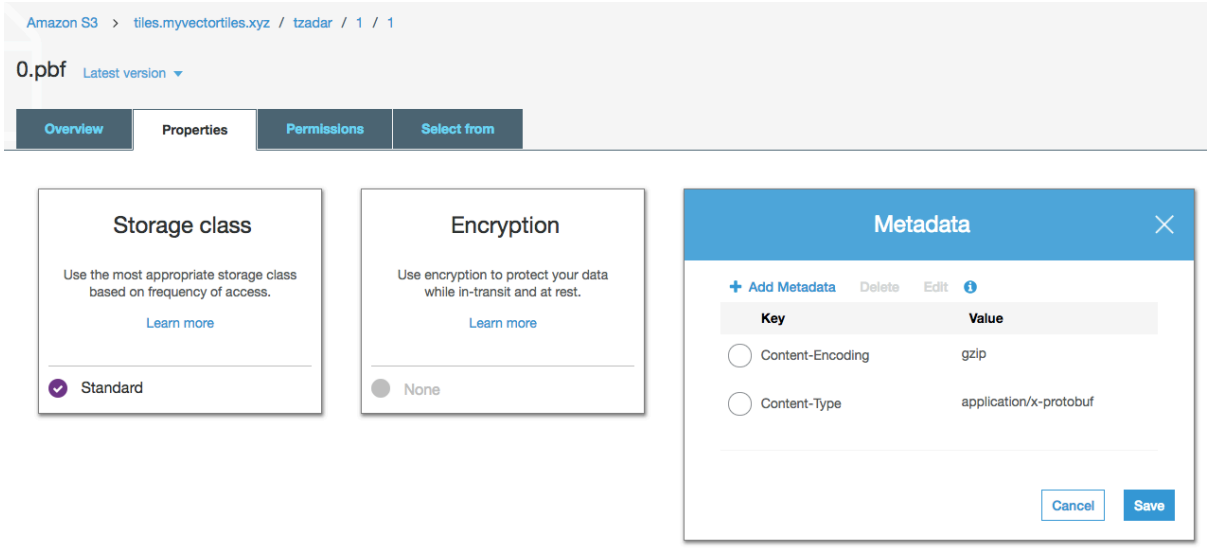
Depending on the size of the area you are uploading and your bandwidth this may take some time. One option to speed this up is to launch a micro EC2 instance on Amazon, in the region where your S3 bucket is, remote into via the command line and run the upload from there but since this is a serverless tutorial that would cheating!



With our upload complete we can head back to the AWS console and check everything looks OK:



Our directories have been created and if we navigate into them we can see pbf files. Let's check the format information is correct:

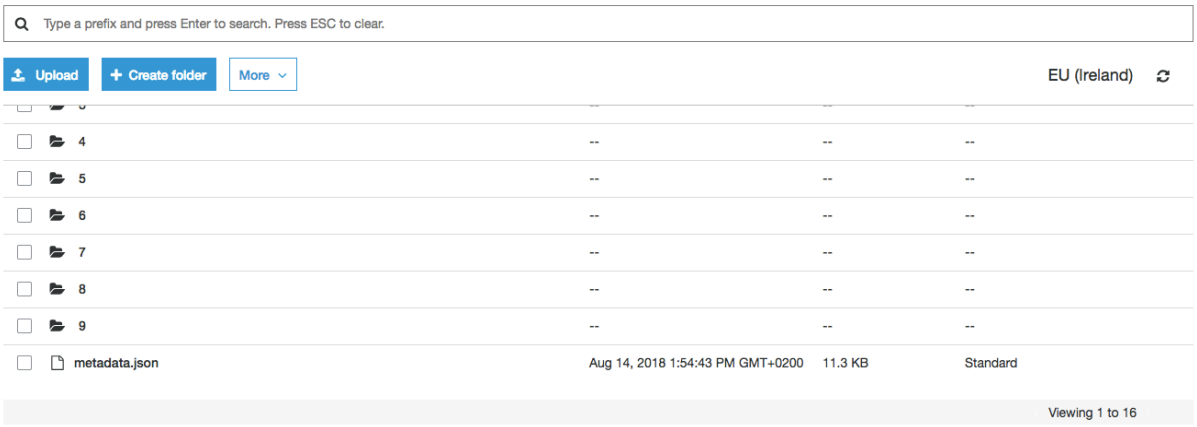


All looking good.

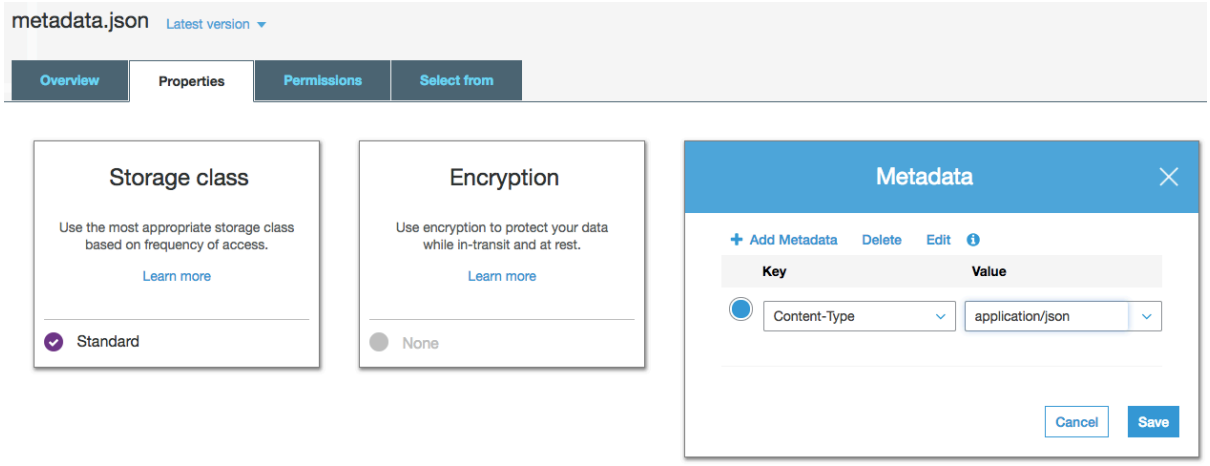
So let's revisit our requirements:

- ~~We need to host some pbf (binary) files using a known directory structure~~
- We need to host some JSON config files (style.json and a tileset.json file)
- We need to support HTTPS
- We need to support multiple subdomains to optimise performance

So what about the config. If we look in the root of our directory we see a metadata.json file.



It would be great if this corresponds to the TileJSON spec that Mapbox GL JS is expecting. Let's open it and take a look. We can't open it or download it, why?! Remember we told Amazon earlier that we were uploading pbf files so we need to delete the Content-Encoding and update the Content-Type:



With the Metadata updated and saved we can download the file:


```
{
"basename": "tanzania_dar-es-salaam.mbtiles",
"attribution": "<a href=\"http://www.openmaptiles.org/\" target=\"_blank\">&copy; OpenMapTiles</a> <a href=\"http://www.openstreetmap.org/about/\" target=\"_blank\">&copy;
OpenStreetMap contributors</a>",
"center": "39.2775,-6.811,14",
"format": "pbf",
"description": "Extract from https://openmaptiles.org",
"planettime": "1499040000000",
"bounds": "38.894,-7.12,39.661,-6.502",
"json": "{\"vector_layers\":[{\"maxzoom\":14,\"fields\":{\"class\":\"String\"},\"minzoom\":0,\"id\":\"water\",\"description\":\"\"},{\"maxzoom\":14,\"fields\":
{\"name:mt\":\"String\",\"name:pt\":\"String\",\"name:az\":\"String\",\"name:ka\":\"String\",\"name:rm\":\"String\",\"name:ko\":\"String\",\"name:kn\":\"String\",\"name:ar\":\"S
tring\",\"name:cs\":\"String\",\"name_de\":\"String\",\"name:ro\":\"String\",\"name:it\":\"String\",\"name_int\":\"String\",\"name:ru\":\"String\",\"name:pl\":\"String\",\"name:
ca\":\"String\",\"name:lv\":\"String\",\"name:bg\":\"String\",\"name:cy\":\"String\",\"name:fi\":\"String\",\"name:he\":\"String\",\"name:da\":\"String\",\"name:de\":\"String\",
\"name:tr\":\"String\",\"name:fr\":\"String\",\"name:mk\":\"String\",\"name:nonlatin\":\"String\",\"name:fy\":\"String\",\"name:be\":\"String\",\"name:zh\":\"String\",\"name:sr\
":\"String\",\"name:sl\":\"String\",\"name:nl\":\"String\",\"name:ja\":\"String\",\"name:lt\":\"String\",\"name:no\":\"String\",\"name:kk\":\"String\",\"name:ko_rm\":\"String\",
\"name:ja_rm\":\"String\",\"name:br\":\"String\",\"name:bs\":\"String\",\"name:lb\":\"String\",\"name:la\":\"String\",\"name:sk\":\"String\",\"name:uk\":\"String\",\"name:hy\":\
"String\",\"name:sv\":\"String\",\"name_en\":\"String\",\"name:hu\":\"String\",\"name:hr\":\"String\",\"class\":\"String\",\"name:sq\":\"String\",\"name:el\":\"String\",\"name:g
a\":\"String\",\"name:en\":\"String\",\"name\":\"String\",\"name:gd\":\"String\",\"name:ja_kana\":\"String\",\"name:is\":\"String\",\"name:th\":\"String\",\"name:latin\":\"Strin
g\",\"name:sr-Latn\":\"String\",\"name:et\":\"String\",\"name:es\":\"String\"},\"minzoom\":0,\"id\":\"waterway\",\"description\":\"\"},{\"maxzoom\":14,\"fields\":
{\"class\":\"String\",\"subclass\":\"String\"},\"minzoom\":0,\"id\":\"landcover\",\"description\":\"\"},{\"maxzoom\":14,\"fields\":
{\"class\":\"String\"},\"minzoom\":0,\"id\":\"landuse\",\"description\":\"\"},{\"maxzoom\":14,\"fields\":
{\"name:mt\":\"String\",\"name:pt\":\"String\",\"name:az\":\"String\",\"name:ka\":\"String\",\"name:rm\":\"String\",\"name:ko\":\"String\",\"name:kn\":\"String\",\"name:ar\":\"S
tring\",\"name:cs\":\"String\",\"rank\":\"Number\",\"name_de\":\"String\",\"name:ro\":\"String\",\"name:it\":\"String\",\"name_int\":\"String\",\"name:lv\":\"String\",\"name:pl\
":\"String\",\"name:de\":\"String\",\"name:ca\":\"String\",\"name:bg\":\"String\",\"name:cy\":\"String\",\"name:fi\":\"String\",\"name:he\":\"String\",\"name:da\":\"String\",\"e
le\":\"Number\",\"name:tr\":\"String\",\"name:fr\":\"String\",\"name:mk\":\"String\",\"name:nonlatin\":\"String\",\"name:fy\":\"String\",\"name:be\":\"String\",\"name:zh\":\"Str
ing\",\"name:sl\":\"String\",\"name:nl\":\"String\",\"name:ja\":\"String\",\"name:lt\":\"String\",\"name:no\":\"String\",\"name:kk\":\"String\",\"name:ko_rm\":\"String\",\"name:
ja_rm\":\"String\",\"name:br\":\"String\",\"name:bs\":\"String\",\"name:lb\":\"String\",\"name:la\":\"String\",\"name:sk\":\"String\",\"name:uk\":\"String\",\"name:hy\":\"String
\",\"name:ru\":\"String\",\"name:sv\":\"String\",\"name_en\":\"String\",\"name:hu\":\"String\",\"name:hr\":\"String\",\"name:sr\":\"String\",\"name:sq\":\"String\",\"name:el\":\
"String\",\"name:ga\":\"String\",\"name:en\":\"String\",\"name\":\"String\",\"name:gd\":\"String\",\"ele_ft\":\"Number\",\"name:ja_kana\":\"String\",\"name:is\":\"String\",\"osm
_id\":\"Number\",\"name:th\":\"String\",\"name:latin\":\"String\",\"name:sr-
Latn\":\"String\",\"name:et\":\"String\",\"name:es\":\"String\"},\"minzoom\":0,\"id\":\"mountain_peak\",\"description\":\"\"},{\"maxzoom\":14,\"fields\":
{\"class\":\"String\"},\"minzoom\":0,\"id\":\"park\",\"description\":\"\"},{\"maxzoom\":14,\"fields\":
{\"admin_level\":\"Number\",\"disputed\":\"Number\",\"maritime\":\"Number\"},\"minzoom\":0,\"id\":\"boundary\",\"description\":\"\"},{\"maxzoom\":14,\"fields\":
{\"class\":\"String\"},\"minzoom\":0,\"id\":\"aeroway\",\"description\":\"\"},{\"maxzoom\":14,\"fields\":
{\"brunnel\":\"String\",\"ramp\":\"Number\",\"class\":\"String\",\"service\":\"String\",\"oneway\":\"Number\"},\"minzoom\":0,\"id\":\"transportation\",\"description\":\"\"},
{\"maxzoom\":14,\"fields\":{\"render_min_height\":\"Number\",\"render_height\":\"Number\"},\"minzoom\":0,\"id\":\"building\",\"description\":\"\"},{\"maxzoom\":14,\"fields\":
{\"name:mt\":\"String\",\"name:pt\":\"String\",\"name:az\":\"String\",\"name:ka\":\"String\",\"name:rm\":\"String\",\"name:ko\":\"String\",\"name:kn\":\"String\",\"name:ar\":\"S
tring\",\"name:cs\":\"String\",\"name_de\":\"String\",\"name:ro\":\"String\",\"name:it\":\"String\",\"name_int\":\"String\",\"name:ru\":\"String\",\"name:pl\":\"String\",\"name:
```

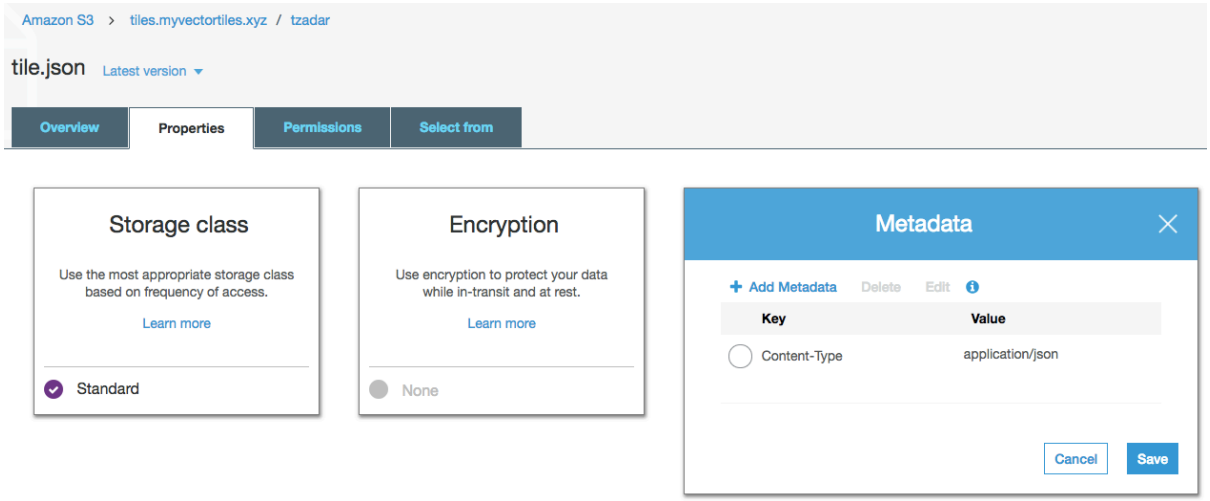

If we look at a simple Mapbox TileJSON file we see that the specs don't match:

```
{
    "attribution": "INSERT_ATTRIBUTION_HERE",
    "basename": "OUR_FILENAME.mbtiles",
    "bounds": [],
    "center": [],
    "description": "Extract from https://openmaptiles.org",
    "format": "pbf",
    "id": "ID_OF_TILESET",
    "maxzoom": "MAX_ZOOM",
    "minzoom": "MIN_ZOOM",
    "name": "NAME_OF_TILESET",
    "scheme": "xyz",
    "tiles": ["https://a.tiles.OUR_SERVER/OUR_TILE_DIR/{z}/{x}/{y}.pbf", "https://b.tiles.OUR_SERVER/OUR_TILE_DIR/{z}/{x}/
    "vector_layers": [],
    "version": "2.0.0"
```

However, with a little bit of copying, pasting and cleanup we can get what we need. We need to replace those \" for valid values. We can then copy everything between the square

brackets after vector_layers in the json section. For the bounds and center we can copy directly and just amend to the TileJSON format (an array of numeric values in square brackets) and then we can set our tiles sources to be the endpoint we are going to host from. We will end up with a file like this https://raw.githubusercontent.com/addresscloud/serverless-tiles/master/06_OSMVectorTilesAWS/tile.json

We can upload this file to our S3 bucket via the console into the tzadar folder at the root. It should look like this:



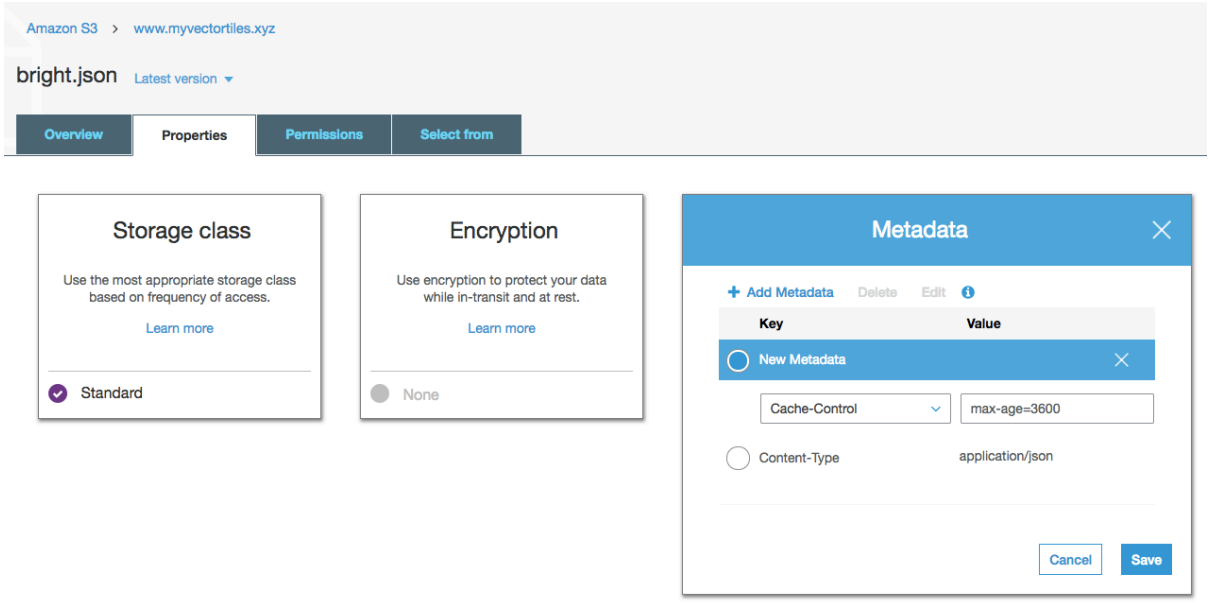
Amazon has detected and set the correct content type.

Now we need some style! Remember that with vector tiles the styles are applied on the fly and can be swapped client side. OpenMapTiles provides some great styles https://openmaptiles.org/styles/ that can be further customised if required, Maputnik is a great tool for this https://maputnik.github.io/. We will use the OSM Bright style: https://github.com/openmaptiles/osm-bright-gl-style to get started. Let's clone the repo and have a look at the style.json file.

We need to update the URLs to point to our tile.json file that we just created. A sample style file is here: https://raw.githubusercontent.com/addresscloud/serverless-tiles/master/06_OSMVectorTilesAWS/bright.json

We can will upload this to our bucket www.myvectortiles.xyz along with our client side code. We can set the Cache-Control header with max-age=3600, this will help ensure that the config file is refreshed at least once an hour. Note that every time we upload this file we will need to remember to set this header. Or we can do this with the S3 CLI with:

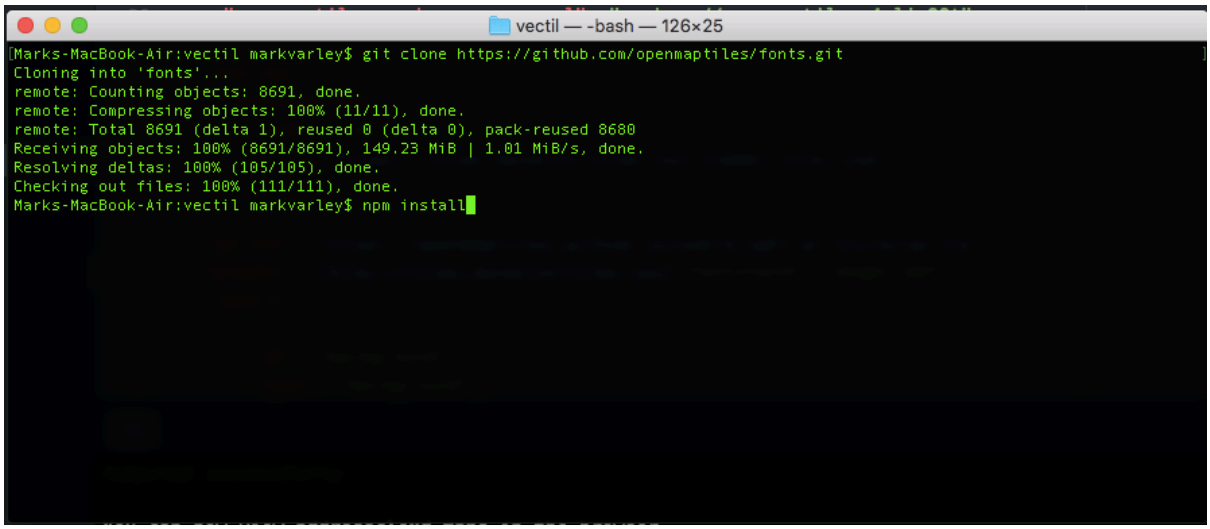
```
aws s3 cp bright.json s3://www.myvectortiles.xyz/bright.json --cache-control max-age=3600
```



Notice in our style file we also have a requirement to serve some fonts. We can either register for a key with https://openmaptiles.org/ or create our own and host out of our bucket. We will do the latter, the steps are as follows:

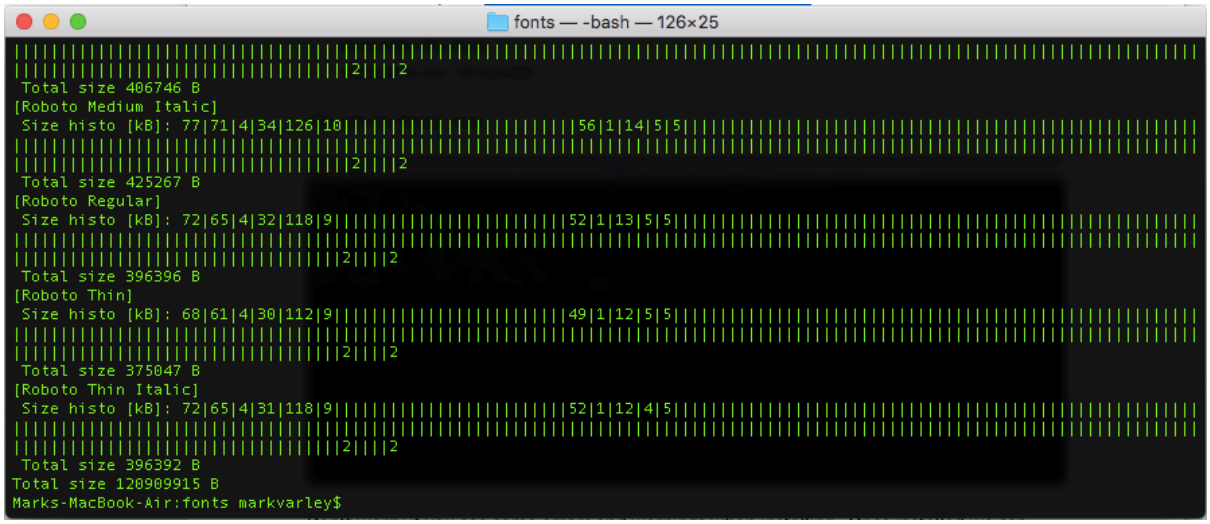
- Clone the repo at https://github.com/openmaptiles/fonts
- Install the package (nodejs is required as a dependency)
- Generate the fonts
- Upload to our S3 bucket

We will do those now:



Don't worry if you see some errors and warnings when installing. Once installed we can generate the font files we need and upload them to S3:

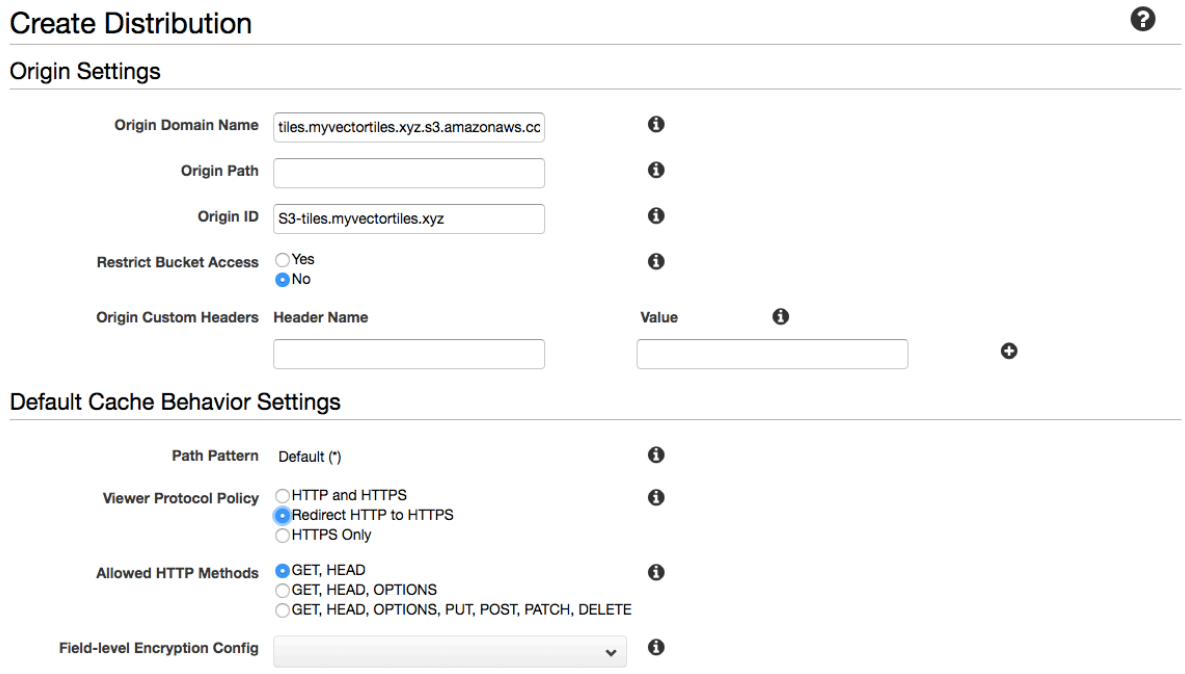
```
node ./generate.js
```



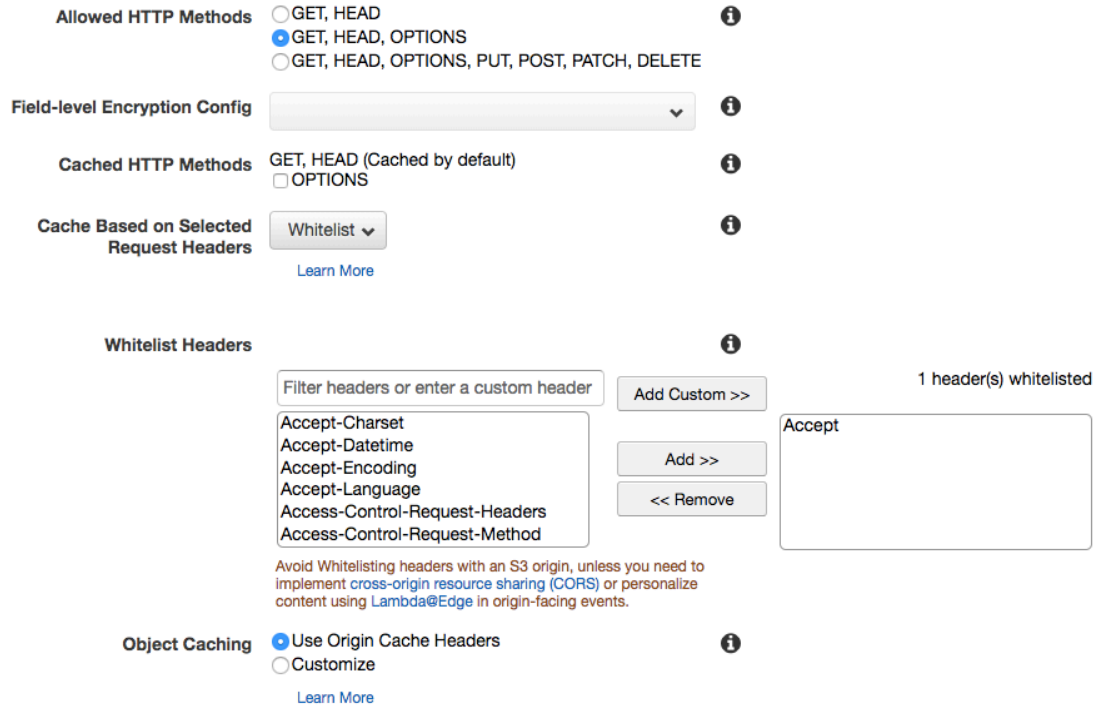

```
aws s3 cp _output s3://tiles.myvectortiles.xyz/fonts/ --
recursive --content-type application/x-protobuf
```

Note: we do not use the content-encoding header this time as the fonts are not compressed.

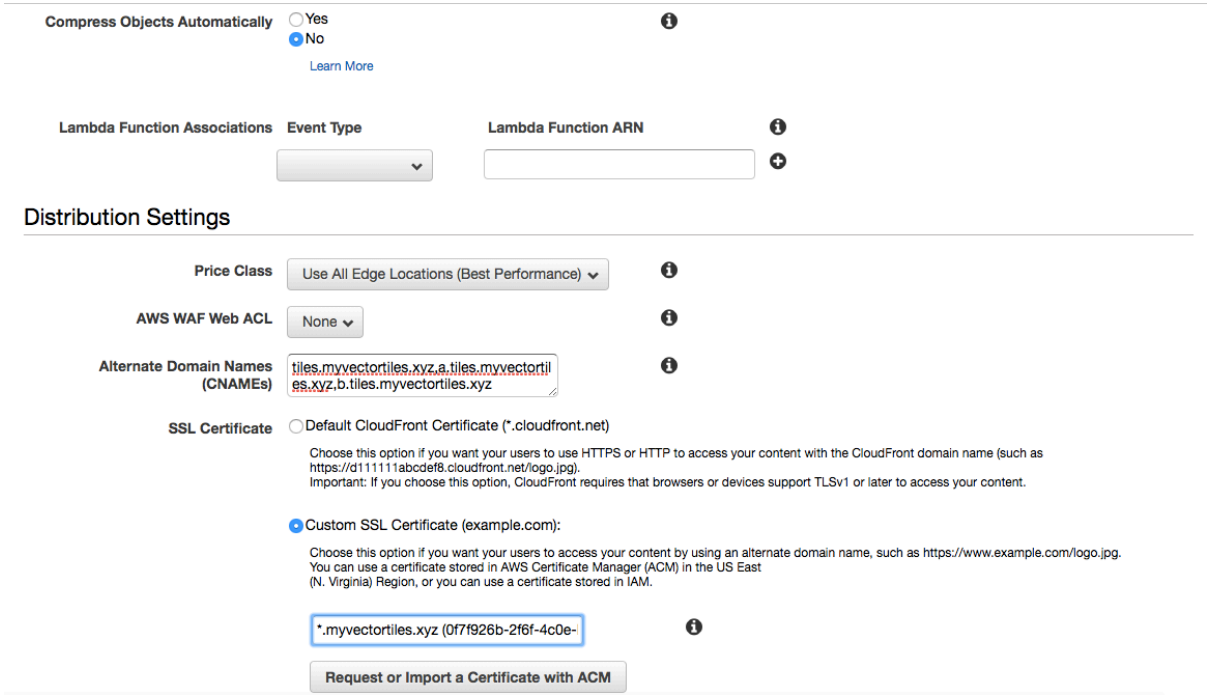
Now we should need to setup out CloudFront distribution to host all this. The process is as before but with a few small differences. We start by Creating a new CloudFront distribution and point it at our S3 bucket and set HTTP to redirect to HTTTPS:



We need to set the Allowed HTTP Methods to GET, HEAD, OPTIONS. We also need to adjust the caching options to Whitelist and then select Accept as a header to whitelist. This will ensure that we don't run into any CORS (https://en.wikipedia.org/wiki/Cross-origin_resource_sharing) errors later on.

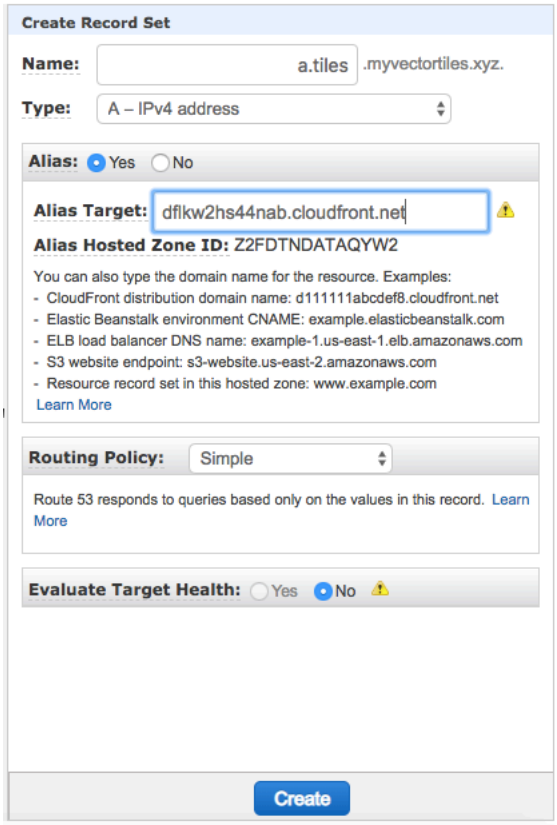


We don't want to compress content this time as our pbf files are already compressed. For our CNAME records we need 3: tiles.myvectortiles.xyz, a.tiles.myvectortiles.xyz, b.tiles.myvectortiles.xyz this is to allow us to serve content on the regular tiles. subdomain or use a and b aliases to improve performance on the client. We select our custom certificate as before:

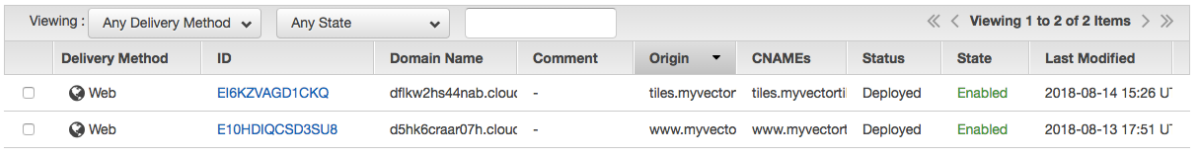


We don't need a default root object so we can go ahead and Create the Distribution. While this is being provisioned and our vector tiles being sent to edge locations all around the world we can setup our DNS, let's head back over to Route 53:

We will setup aliases for each of our endpoints: tiles, a.tiles and b.tiles as so:



Let's check the status of the Distribution, it should be deployed in around 10-15 minutes all being well:



Let's check our new URLs to make sure this is all working:

https://tiles.myvectortiles.xyz/tzadar/tile.json
https://a.tiles.myvectortiles.xyz/tzadar/tile.json
https://b.tiles.myvectortiles.xyz/tzadar/tile.json

We should see our JSON file and a green padlock for each to indicate they are secure:

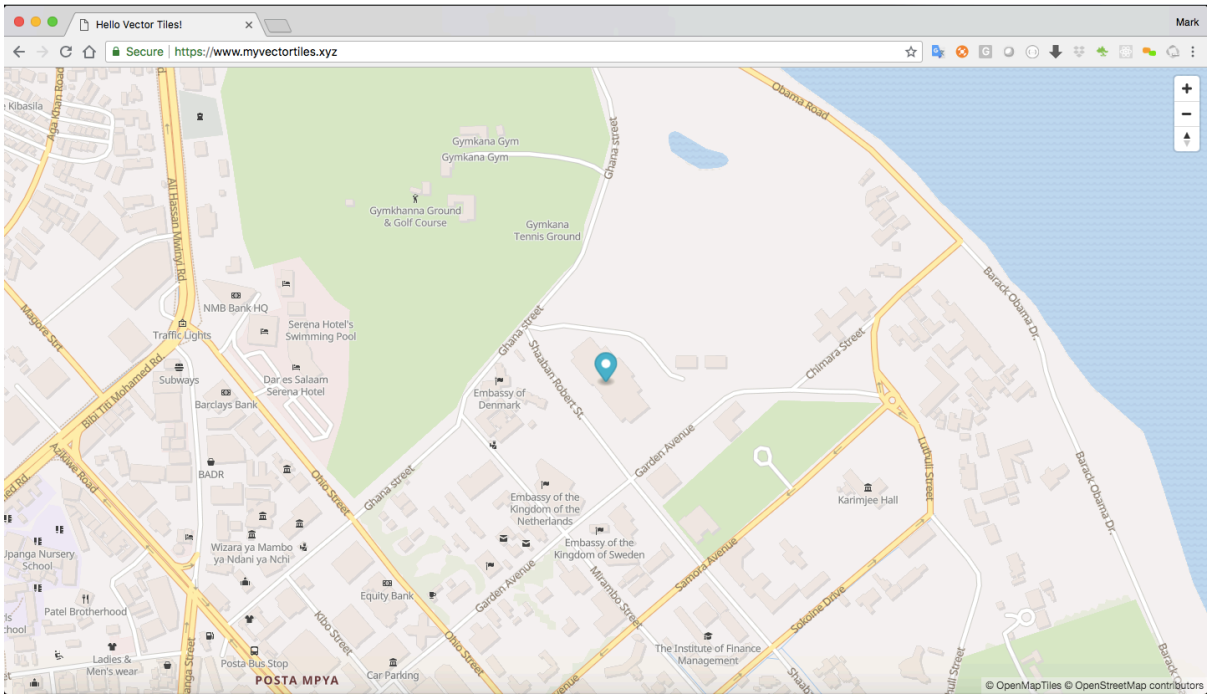
So let's revisit our requirements:

- ~~We need to host some pbf (binary) files using a known directory structure~~
- ~~We need to host some JSON config files (style.json and a tileset.json file)~~
- ~~We need to support HTTPS~~
- ~~We need to support multiple subdomains to optimise performance~~

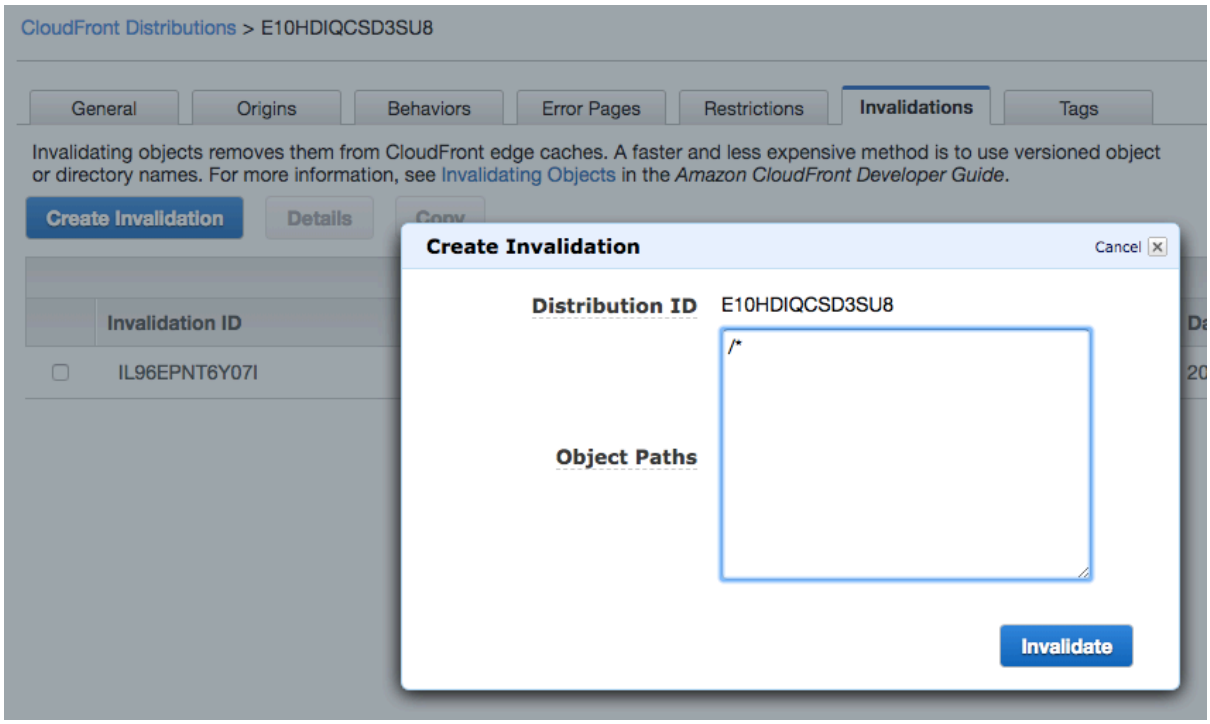
We are now ready to put all this into practice! We will update our index.html with details of the new service, let's make the map occupy the whole page as well so we can revel in our success! The new file should look like this https://raw.githubusercontent.com/addresscloud/serverless-tiles/master/06_OSMVectorTilesAWS/index.html

We will again upload it to our www.myvectortiles.xyz bucket.

Let's see what is happening at https://www.myvectortiles.xyz/ et voila!



Note: content can be cached for an hour or so, if we do not see a map straight away, we can force a refresh of the cache by going to the CloudFront console, selecting our distribution and create an invalidation as follows:

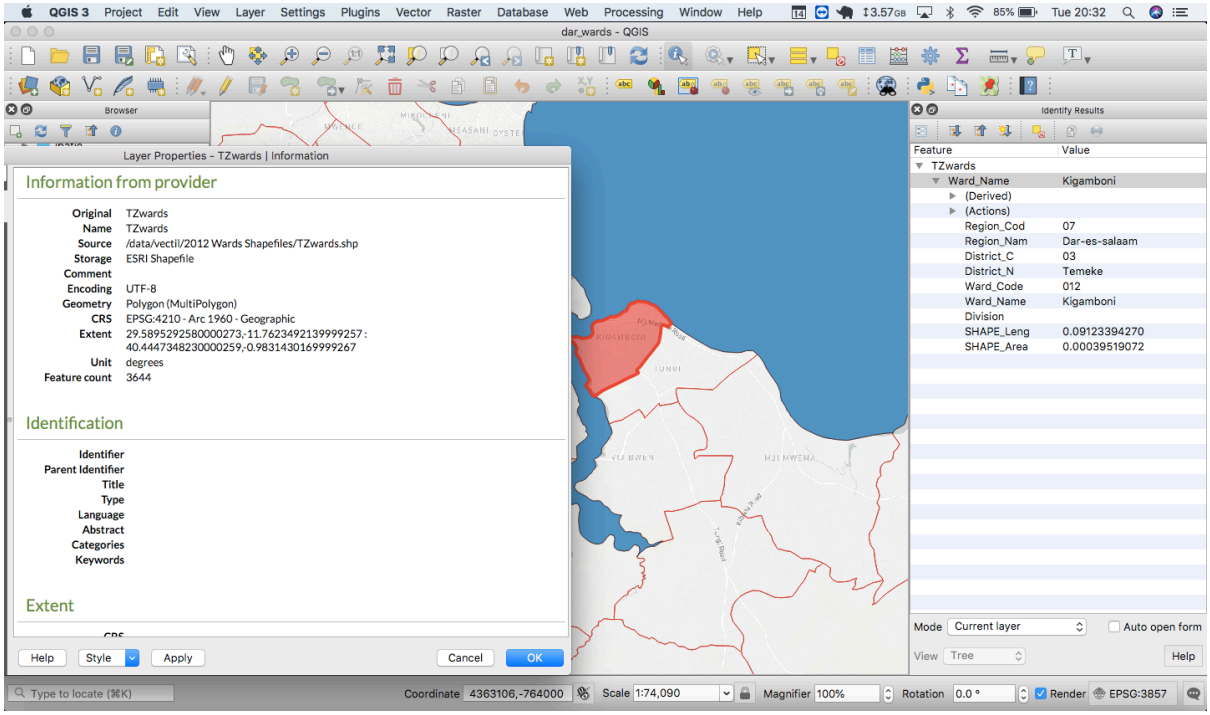


This process can take 15-20 minutes as all of the edge locations around the world need to re-sync to our master region.

7. Publish our own geo data to S3/CloudFront

We now come to our final section, publishing our own content. This is actually fairly simple thanks to the excellent tools from Mapbox. To get started though we will need some data to overlay on our map of Dar Es Salaam. A quick Google search shows that ward information is available for download for Tanzania can be downloaded from https://data.humdata.org/dataset/2012-census-tanzania-wards-shapefiles

We download the file and preview it in QGIS:



It looks good however looking at the Layer Properties and we can see that the CRS is EPSG:4210 we will need to convert this into EPSG:4326 to work with the Mapbox tools. We also need to convert the file from an ESRI Shapefile to GeoJSON. Luckily we can do both of these in a single operation with the excellent ogr2ogr https://www.gdal.org/ogr2ogr.html tool that is available as part of GDAL https://www.gdal.org/

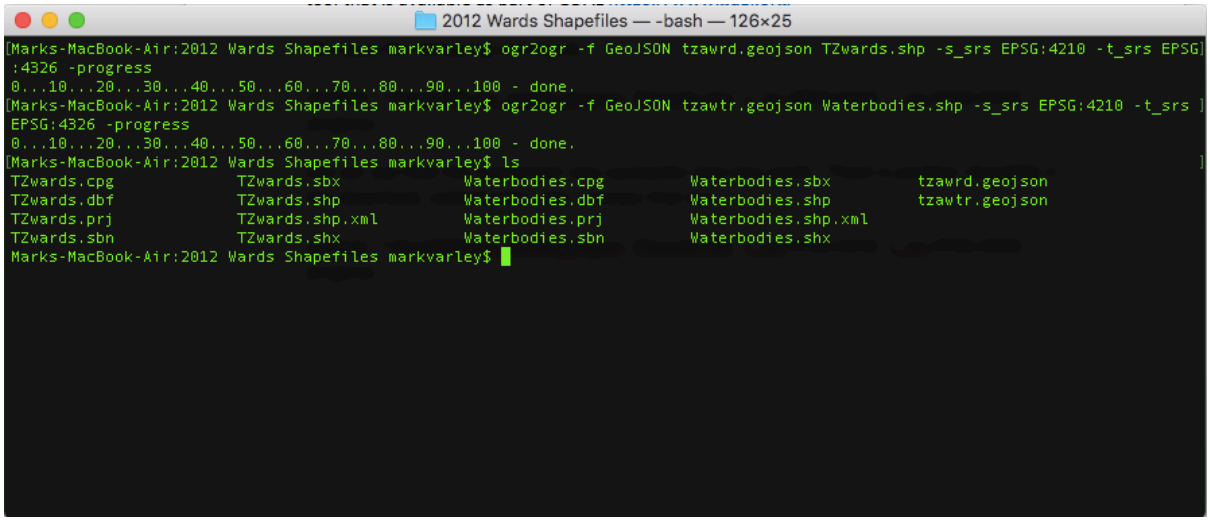
We issue the following command:

```
ogr2ogr -f GeoJSON tza_wards.geojson TZwards.shp -s_srs
EPSG:4210 -t_srs EPSG:4326 -progress
```

We can see that the conversion was successful and very quick. There is another shapefile in the download, waterbodies, vector tiles files can include multiple layers so let's process that too:

```
ogr2ogr -f GeoJSON tza_wards.geojson TZwards.shp -s_srs
EPSG:4210 -t_srs EPSG:4326 -progress
```

Note: if we wanted to generate our GeoJSON file from a spatial database such as PostGIS or practically any other spatial format ogr2ogr has drivers for almost all of them!

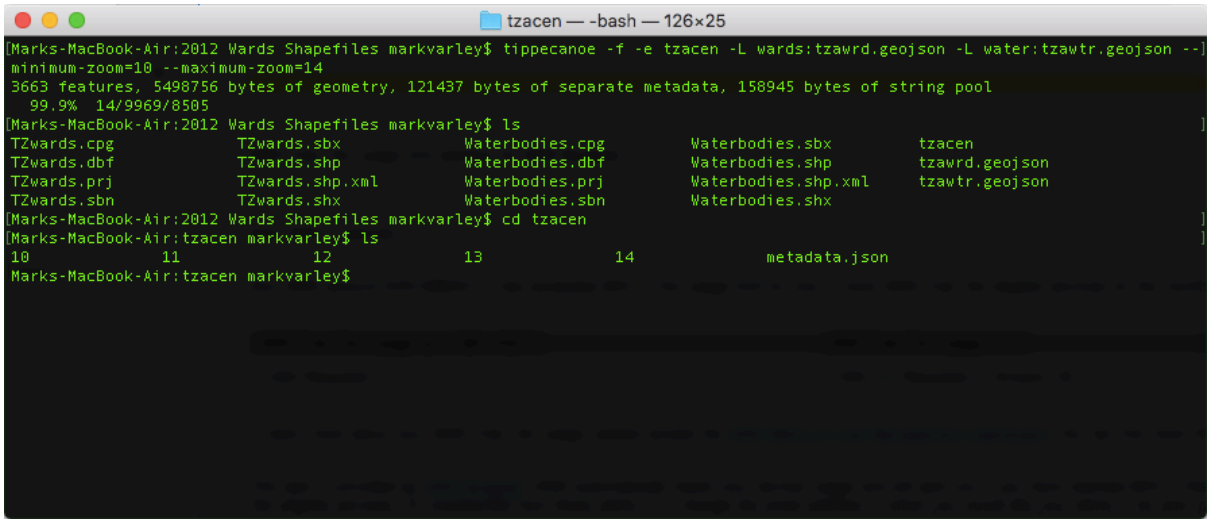


We now need to convert our geojson into vector tiles. For this we can use the excellent Mapbox Tippecanoe tool https://github.com/mapbox/tippecanoe

With Tippecanoe installed (use the instructions in the repo or on a Mac we can issue brew install Tippecanoe) we run the following command which will create a directory with our vector tiles in it:

```
tippecanoe -f -e tzacen -L wards:tzawrd.geojson -L
water:tzawtr.geojson --minimum-zoom=10 --maximum-zoom=14
```

Once complete we have a new directory tzacen with the familiar structure and a new metadata.json file



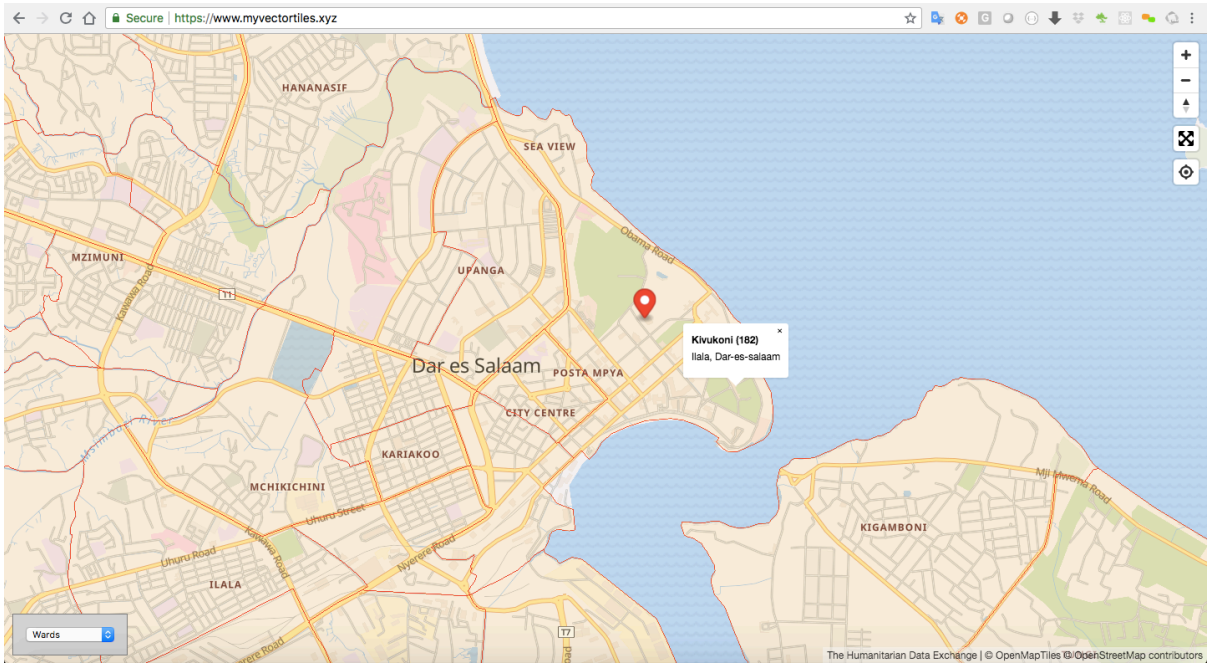
As before with the openmaptiles bundle we upload this directory to our S3 bucket and set the appropriate headers:

```
aws s3 cp tzacen s3://tiles.myvectortiles.xyz/tzacen/ --
recursive --content-type application/x-protobuf --content-
encoding 'gzip'
```

Once complete we need to configure our tile.json file using the contents of our metadata.json file. The approach is as before and we will end up with something like this https://raw.githubusercontent.com/addresscloud/serverless-tiles/master/07_CustomVectorTilesAWS/tile.json

We need to update our style file to add support for our new tileset and layers. Here is an example of some simple styling: https://raw.githubusercontent.com/addresscloud/serverless-tiles/master/07_CustomVectorTilesAWS/bright.json. Search for tzacen and you should be able to work out how this all hangs together.

Finally we will add a few extra controls and a new custom layer control using simple HTML to our client page: https://raw.githubusercontent.com/addresscloud/serverless-tiles/master/07_CustomVectorTilesAWS/index.html



Our final result looks like this, not bad at all for something we have written and hosted ourselves for almost free!

Conclusion

We have taken a simple web page and using the power of AWS cloud services we have distributed this around the world to edge locations so it always loads quickly, secured with HTTPS on our own domain. We applied the same technique to our map tiles and build two vector tile caches hosted on our subdomain, one using pre-prepared OpenStreetMap tiles and another using our own data. We built a map and a custom style that combined the two layers and add some simple client side interactions on the vector tiles.

This technique is very flexible and powerful and extending this example to use some Single Page Application technology such as the excellent ReactJS from Facebook https://reactjs.org/ we can build complex and beautiful applications and host them for next to nothing on AWS. If we need more advanced server side calls we can leverage hosted services such as AWS lambda for our server-side functions, RDS for hosting our PostGIS Database (a serverless version of RDS is expected to be released in 2018) and API Gateway for securing our endpoints and protecting us against DDoS attacks.

The best bit is that we can host this whole stack for free / next to nothing! I hope the techniques here will prove useful to others who, like I was 3 years ago, are looking to launch their own business from scratch with no resources other than a good idea and a determination to learn.

I would like to thank all those who give up their time to create the amazing open software that we all use as well as those who take time to write tutorials and blog posts on their own findings which are often clearer than any official docs.

Mark Varley
August 2018