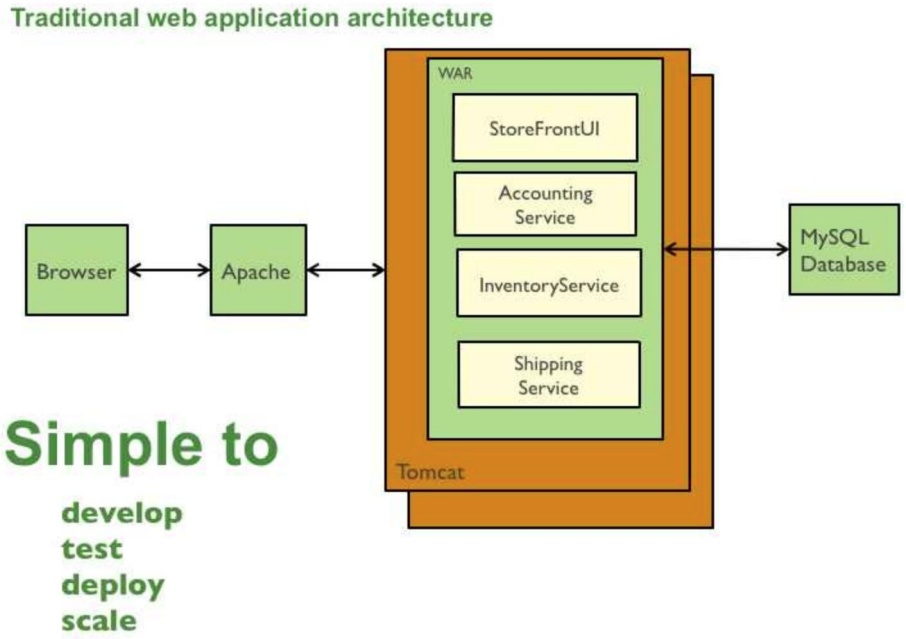
AWS Serverless

Andreas Becker Bertelsen

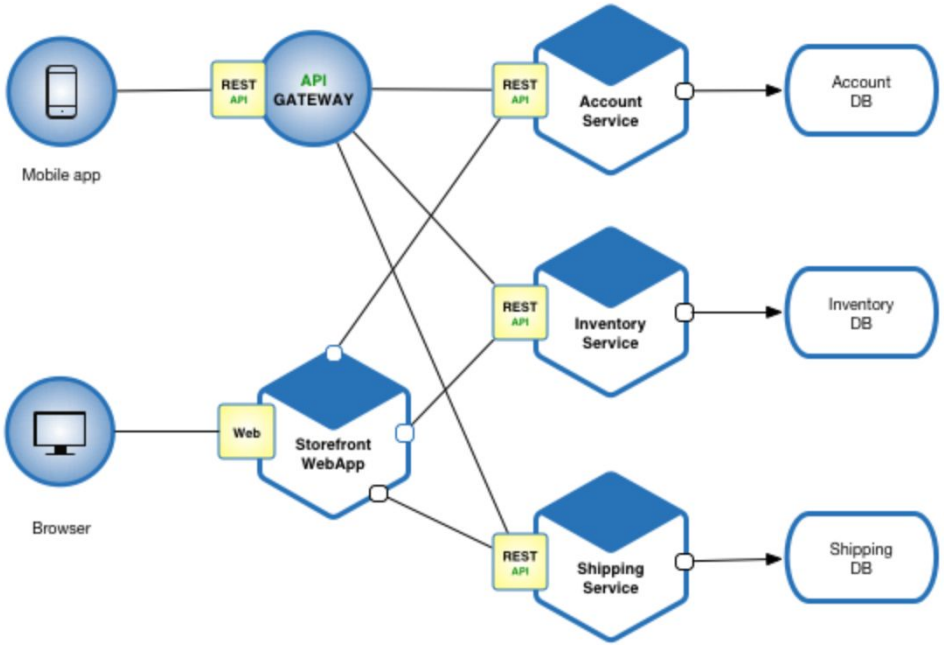
Agenda

- Introduction to serverless
- What are we going to build?
- Building a serverless application in AWS
- Using the serverless framework

Monolithic approach



Microservice approach



What are containers?

A container is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, and settings.

Docker

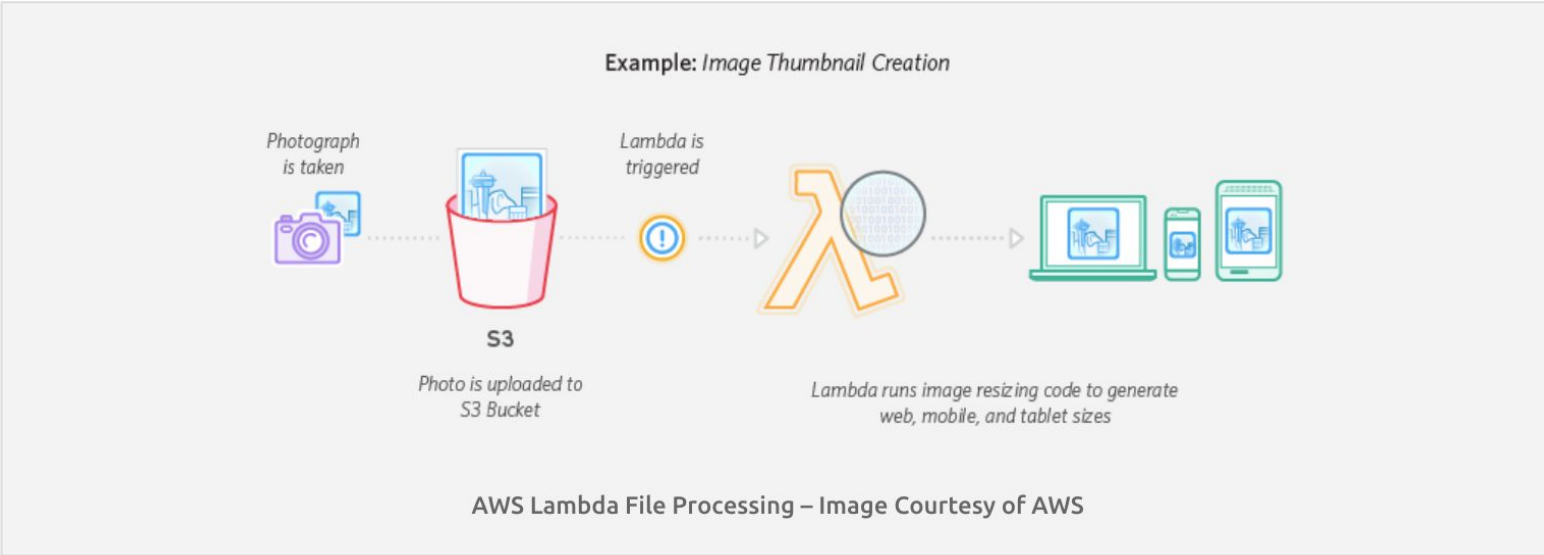
Why containers?

The first benefit of containers is their portability. The main draw of a container is that you can combine the application and all of its dependencies into a neat little package and run it anywhere. This provides an unprecedented level of flexibility and portability, and allows you to stay cloud vendor-agnostic.

What is serverless?

Serverless architecture replaces long-running virtual machines with ephemeral compute power that comes into existence on request and disappears immediately after use.

Thoughtworks 2016



Serverless is based on events, meaning once a condition is triggered a lambda can respond

Who uses serverless?

- Netflix was one of the pioneers
  - Encode media files
  - Back up files for disaster recovery
  - Secure their assets
  - Monitor their environment

Containers vs Serverless

- Both are portable
- Serverless monitoring improving vs Containers full control over monitoring
- Serverless pay for what you use vs Containers pay for CPU & memory pr. minute
- Serverless no infrastructure vs Containers full control
- Serverless cold starts vs Containers always ready
- Serverless cannot handle long processes vs Containers are great at long processes

What are we going to build?

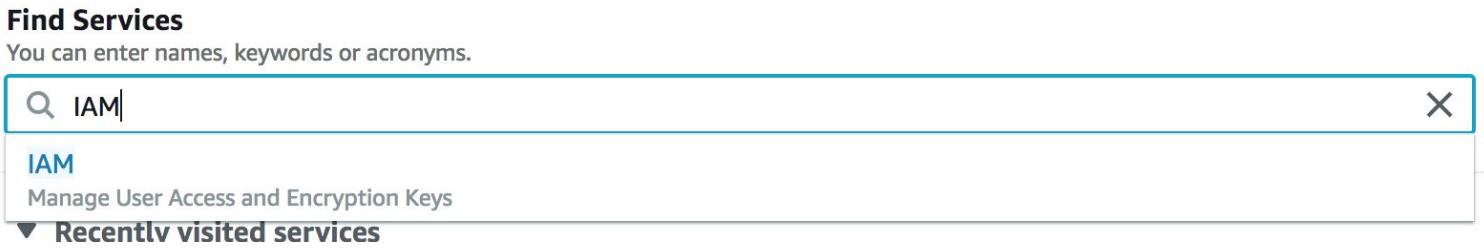
- A Todo application
  - http://todo-webpage.s3-website-eu-west-1.amazonaws.com/
  - Get the index.html file from here: https://github.com/Andyandpandy/aws-serverless-workshop/tree/workshop_start

- A simple application that adds, updates, deletes, and gets todos.
  - When a todo is added it is updated with the prefix 'Todo: '

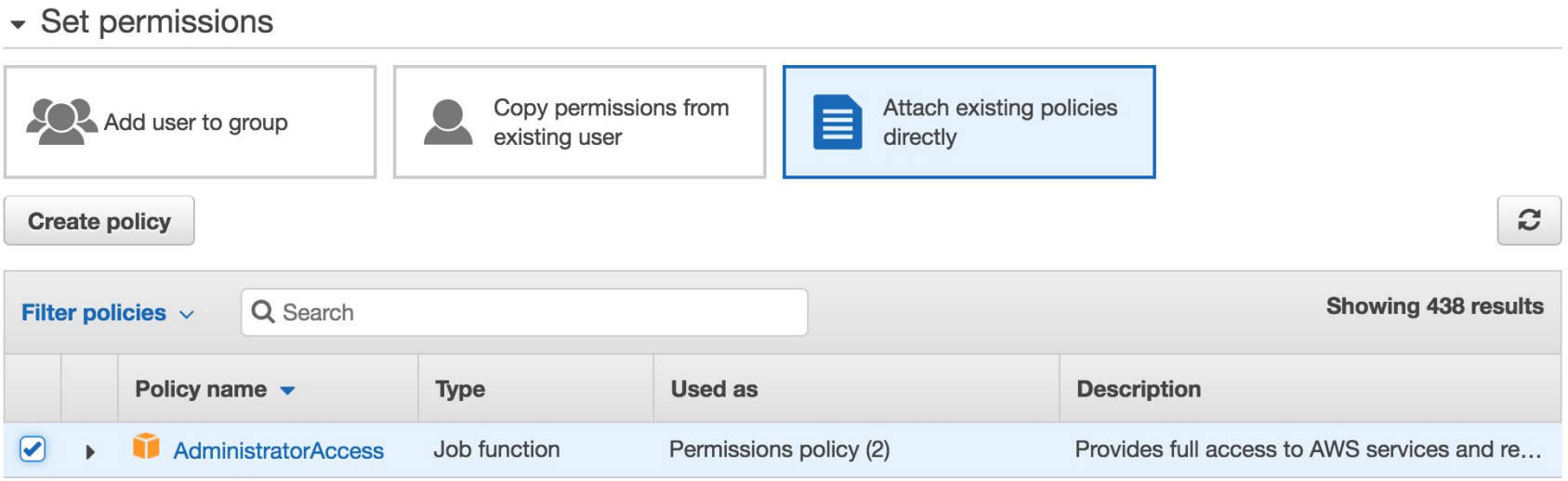
Let’s get started

Overview of AWS

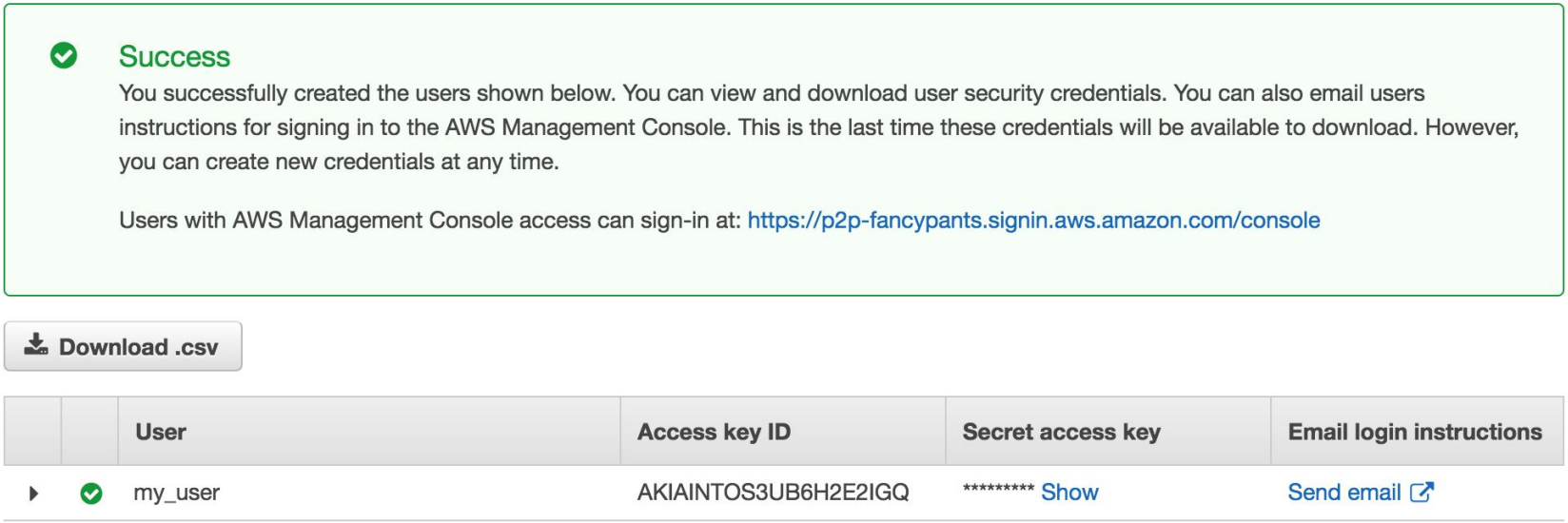
Create an IAM user



- We would like both programmatic access and AWS Managemant console access
-



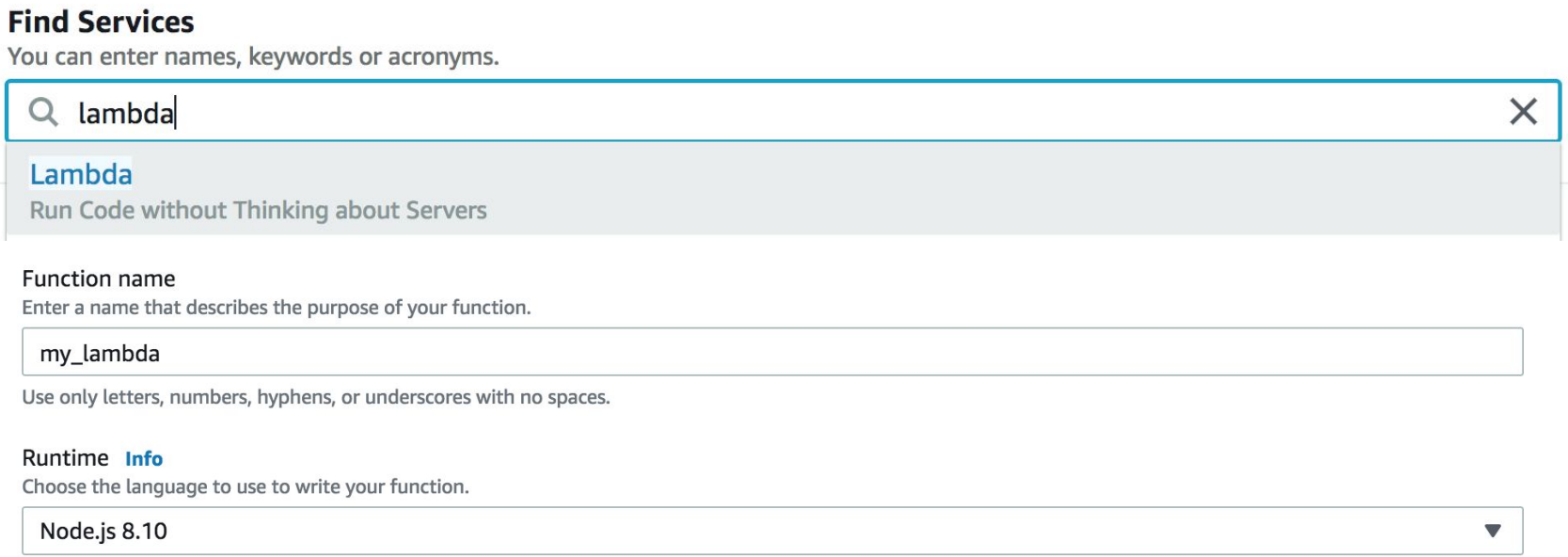
Bad practice, but for this exercise we will be super important users! Now create this user



Important! Download the .csv file! We will need this later. Click on the console access sign-in link!

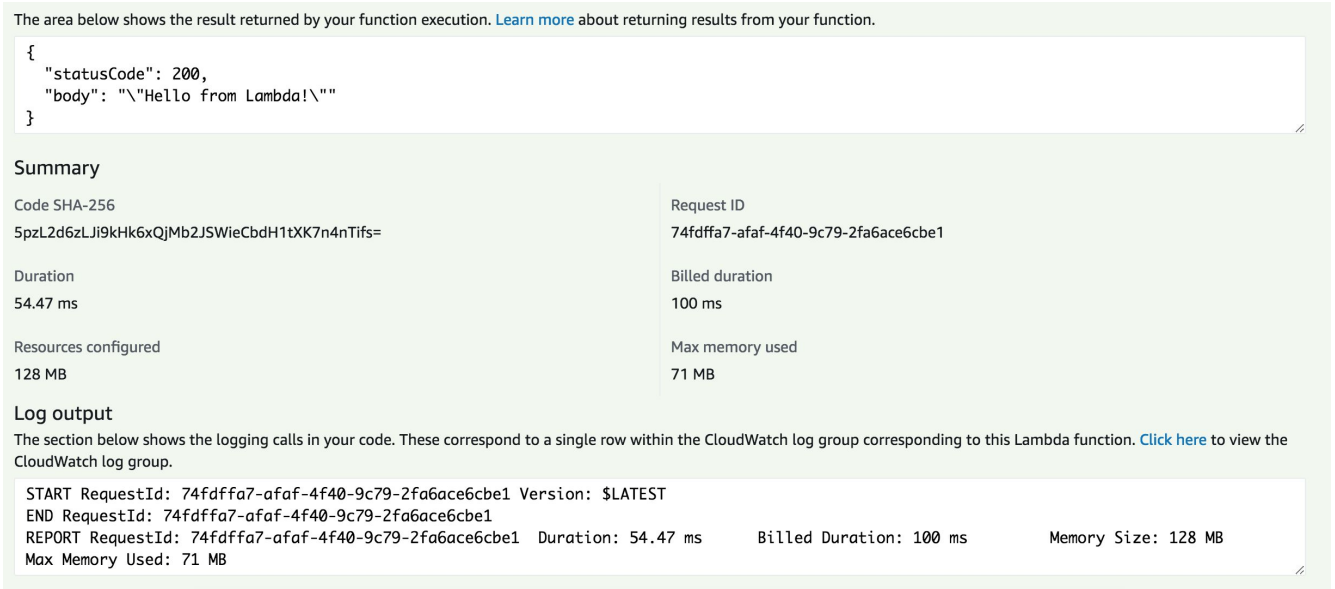
Login with your new user

Create your first nodejs lambda



Test your lambda

- Click on Test
- Enter a name
- Click on create
- Click test again



Let’s create the lambda’s needed for the todo application

What do we need?

Create four lambda functions

- A FETCH (GET) method for getting all todos
- An ADD (POST) method for adding a todo
- An UPDATE (PATCH) method for updating a todo
- A REMOVE (DELETE) method for deleting a todo

We need somewhere to store our todos

DynamoDB

amazon web services™

Introducing AWS DynamoDB! A NOSQL database

Create a DynamoDB table

DynamoDB is a schema-less database that only requires a table name and primary key. The table's primary key is made up of one or two attributes that uniquely identify items, partition the data, and sort data within each partition.

Table settings

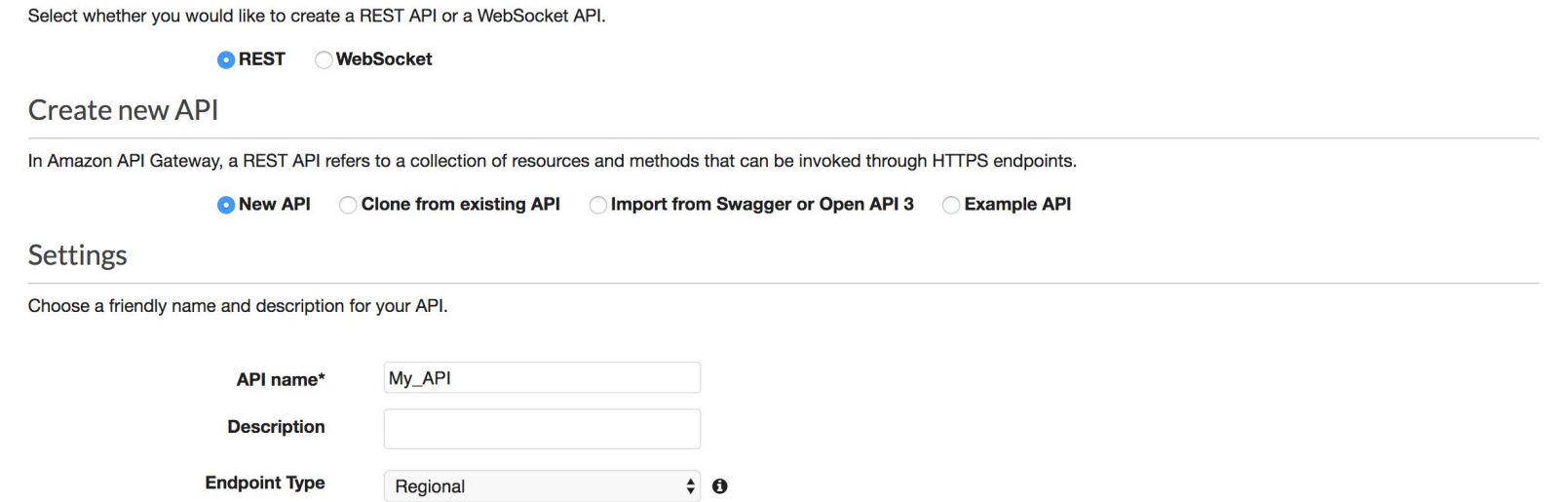
Default settings provide the fastest way to get started with your table. You can modify these default settings now or after your table has been created.

Use default settings

- No secondary indexes.
- Provisioned capacity set to 5 reads and 5 writes.
- Basic alarms with 80% upper threshold using SNS topic "dynamodb".
- Encryption at Rest with DEFAULT encryption type NEW!

Name your table and set the Partition key to an id and sort key to a timestamp with default settings

Let’s create our API with AWS API Gateway



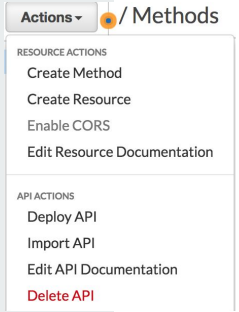
Create an API Gateway with default settings!

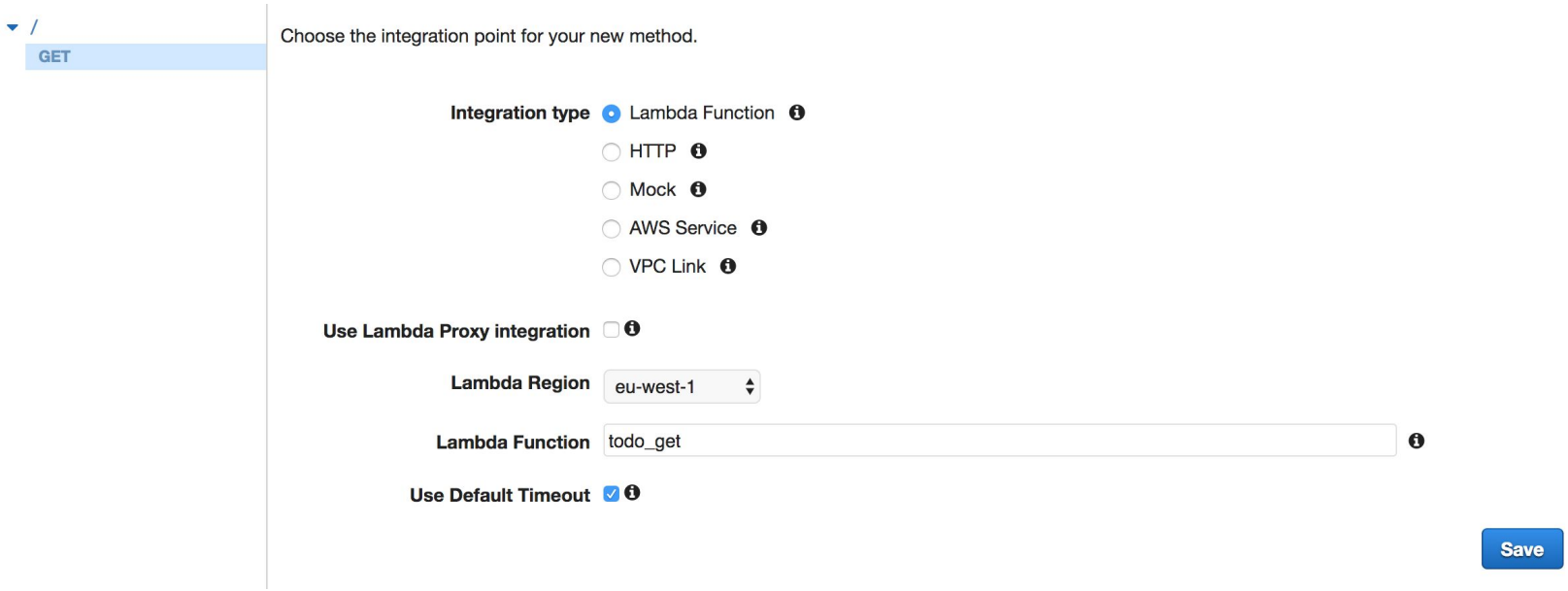
What does this API need?

- A FETCH (GET) method for getting all todos
- An ADD (POST) method for adding a todo
- An UPDATE (PATCH) method for updating a todo
- A REMOVE (DELETE) method for deleting a todo

Create four methods for our four lambdas

Match the method to the lambda created for it e.g. todo_get -> GET





Example of matching get method to lambda get function

Final touches for the API

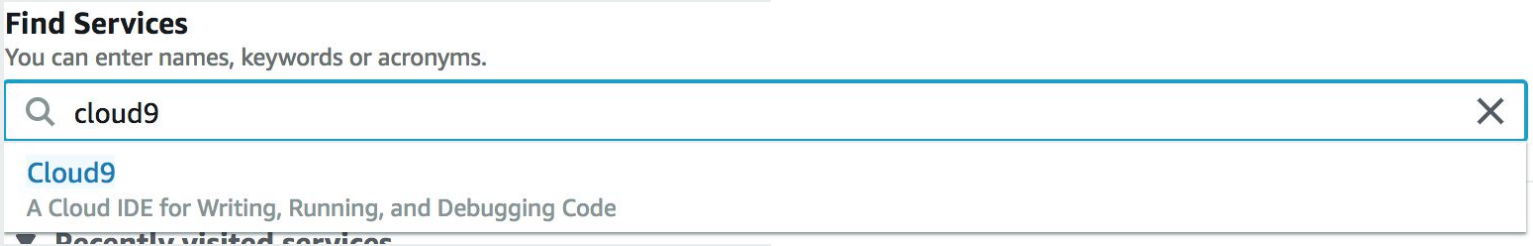
- Enable cors with the default settings for this path ‘/’
- Deploy the api.
- Create a new stage.
- Click on the new link and see your get lambda respond

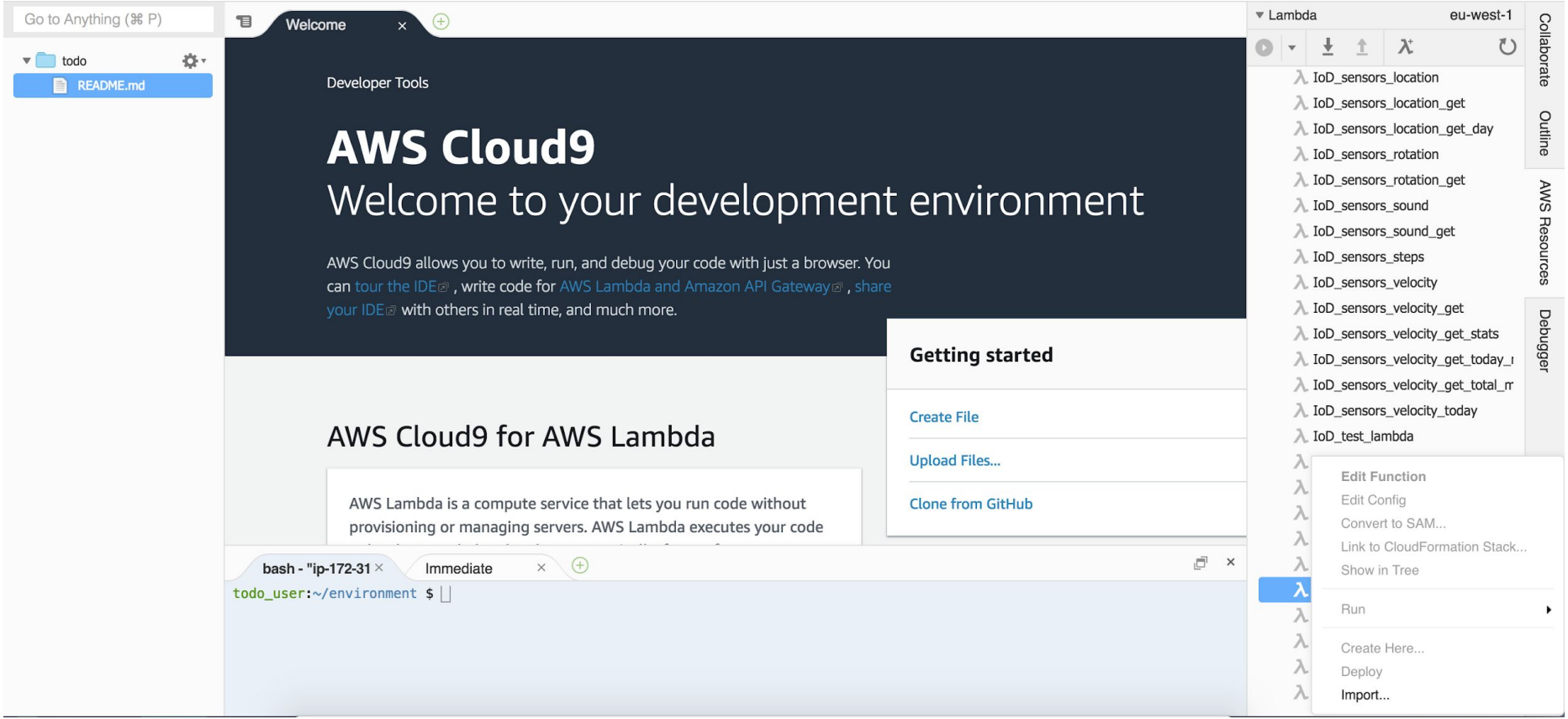
Let’s get the lambda’s working

Create a cloud9 environment

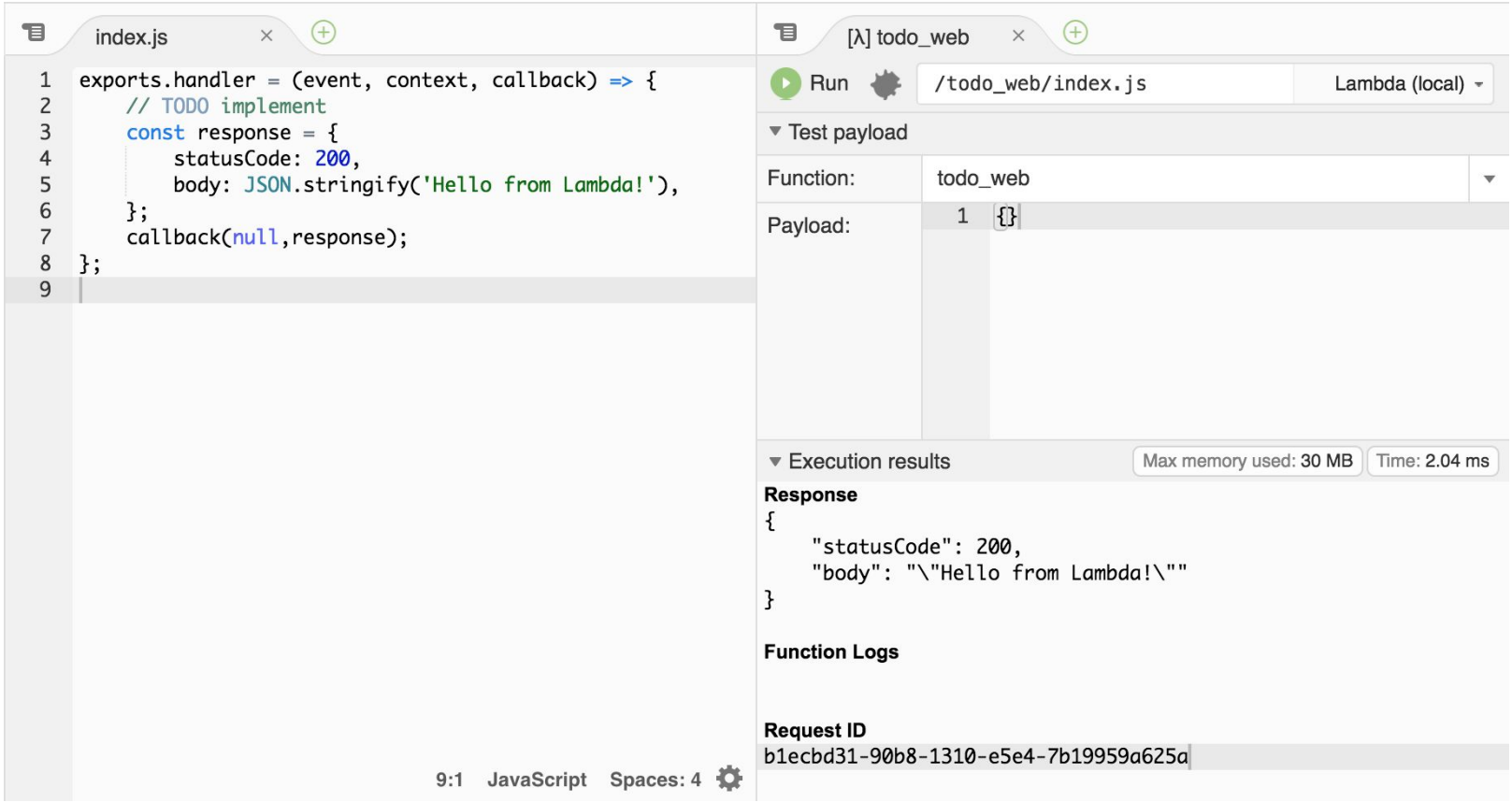
We need somewhere to create the code for our project

- Enter a name for your IDE
- Create the environment with all default settings





Import your lambda functions



Change the code to have 3 parameters and call the callback parameter. Then run it!

Time to get to work

For the add lambda you can use uniqid for generating id's and Date.now() for timestamps

```
// dependencies
var AWS = require('aws-sdk');

// Get reference to AWS dynamodb
var dynamodb = new AWS.DynamoDB.DocumentClient();
```

```
todo_user:~/environment $ cd todo_add
todo_user:~/environment/todo_add $ npm init
todo_user:~/environment/todo_add $ npm install uniqid
```

```
var uniq = require('uniqid');
var id = uniq();
var timestamp = Date.now();
```

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStarted.NodeJs.03.html

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStarted.NodeJs.04.html

You can find the code solutions here:

Let’s now try it with the API

{"errorMessage":"Error in get: AccessDeniedException: User: arn:aws:sts::183094096906:assumed-role/lambda_basic_execution/todo_get is not authorized to perform: dynamodb:Scan on resource: arn:aws:dynamodb:eu-west-1:183094096906:table/todos"}

So what went wrong? Why doesn't it work anymore?

Lambda execution roles

- Right now the lambda has a basic execution role
  - This means it only logs what happens in the lambda
  - It does not have permissions for using other services
- But it worked with cloud9?
  - Yes it works locally on cloud9 because then it is the cloud9 that takes over
  - Try running the lambda remotely in cloud9
- So how do we fix it?
  - Create a new Lambda IAM Role with full access to dynamodb

Select type of trusted entity

AWS service
EC2, Lambda and others

Another AWS account
Belonging to you or 3rd party

Web identity
Cognito or any OpenID provider

SAML 2.0 federation
Your corporate directory

Allows AWS services to perform actions on your behalf. Learn more

Choose the service that will use this role

EC2
Allows EC2 instances to call AWS services on your behalf.

Lambda
Allows Lambda functions to call AWS services on your behalf.

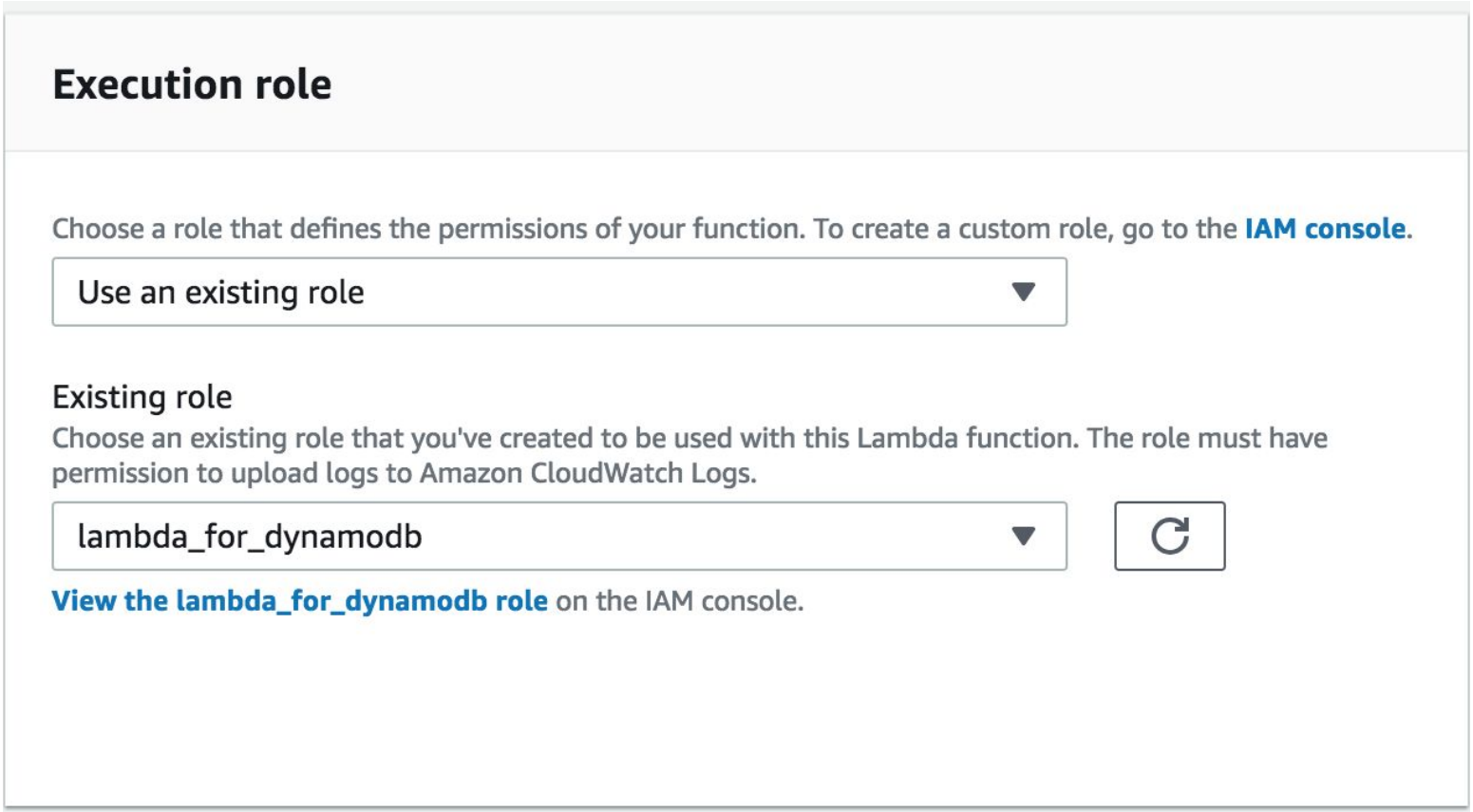
Attach permissions policies

Choose one or more policies to attach to your new role.

Create policy

Filter policies | dynamo | Showing 7 results

Policy name	Used as	Description
AmazonDynamoDBFullAccess	Permissions policy (3)	Provides full access to Amazon Dynamo…



Set each lambda to use the role you just created and test your API again

Let’s make it official

Find Services

You can enter names, keywords or acronyms.

s3

S3

Scalable Storage in the Cloud

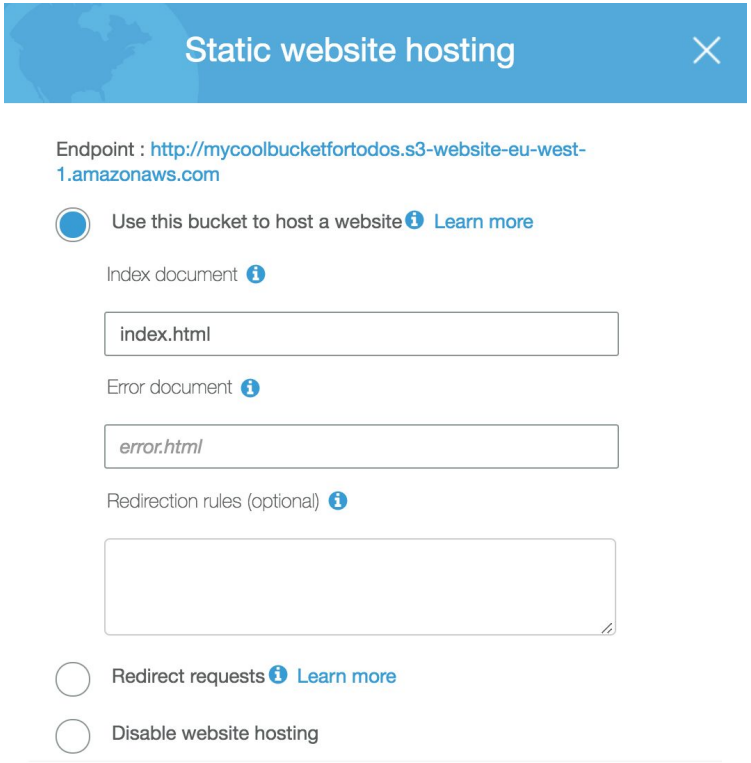
Manage public access control lists (ACLs) for this bucket

- [ ] Block new public ACLs and uploading public objects *(Recommended)*
- [ ] Remove public access granted through public ACLs *(Recommended)*

Manage public bucket policies for this bucket

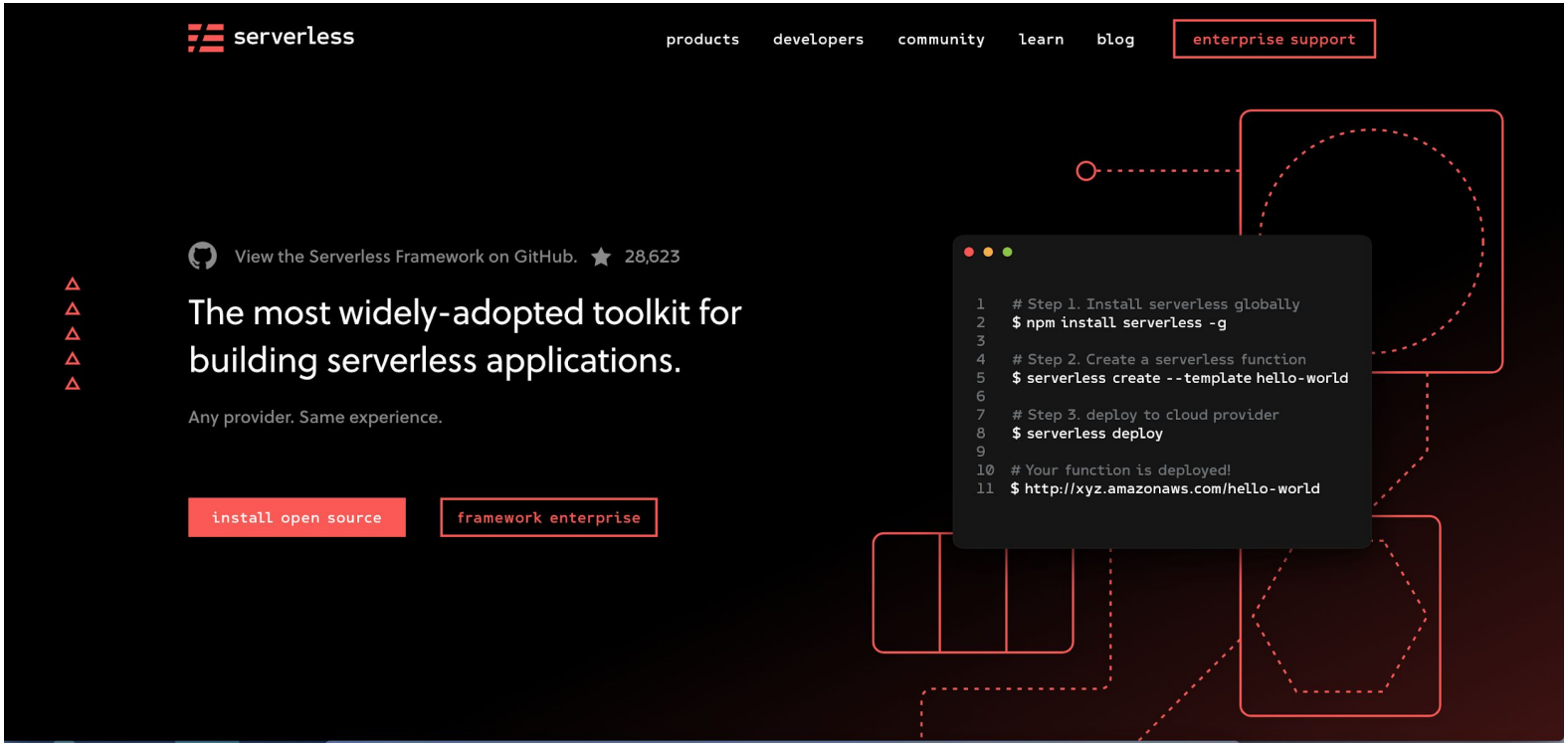
- [ ] Block new public bucket policies *(Recommended)*
- [ ] Block public and cross-account access if bucket has public policies *(Recommended)*

Create an s3 bucket and set permissions to false



Upload the index.html file to the bucket and grant public read access and enable static website hosting on the buckets properties

Fast development! Not really..



Introducing the Serverless framework - https://serverless.com/framework/docs/providers/aws/guide/intro/

Let’s create a serverless application

- Get Serverless
  - Command: npm install -g serverless
  - Their github: https://github.com/serverless/serverless
- Set credentials (This might not work on windows, try using set instead of export)
  - Command: export AWS_ACCES_KEY_ID=<key>
  - Command: export AWS_SECRET_ACCESS_KEY=<secret-key>
  - If this doesn’t work:
    - https://github.com/serverless/serverless/blob/master/docs/providers/aws/guide/credentials.md
- Create aws nodejs application in a folder
  - Command: serverless create --template aws-nodejs --path my-cool-folder

Deploy to AWS

- Open the serverless.yml and add under the provider the correct region
  - region: eu-west-1
- Deploy development API to AWS
  - Command: serverless deploy -v
- Deploy production API to AWS
  - Command: serverless deploy -v --stage prod
- Invoke your function
  - Command: serverless invoke -f function_name -l
- Change your function and upload only the function
  - Command: serverless deploy -f function_name
- Get logs of a function
  - Command: serverless logs -f function_name -t

Adapt the website

- Edit the url variable to your new API
- Use your previous lambda functions in the serverless application
- You must enable cors.
- Add a specific iamRoleStatement

```
functions:
  get_my_todo:
    handler: todo_get.handler
    events:
      - http:
          path: /
          method: get
          cors: true
```

```
iamRoleStatements:
  - Effect: "Allow"
    Action:
    - "dynamodb:*"
    Resource: "*"
```

Lambda's changes

```
const body = JSON.parse(event.body)
const tid = body.tid;
const timestamp = body.timestamp;
```

```
callback(null, { statusCode: 200,
 headers: {
   "Access-Control-Allow-Origin" : "*",
   "Access-Control-Allow-Credentials" : true
 },
 body: JSON.stringify(result)});
```


FEEDBACK