



AWS Serverless

Andreas Becker Bertelsen

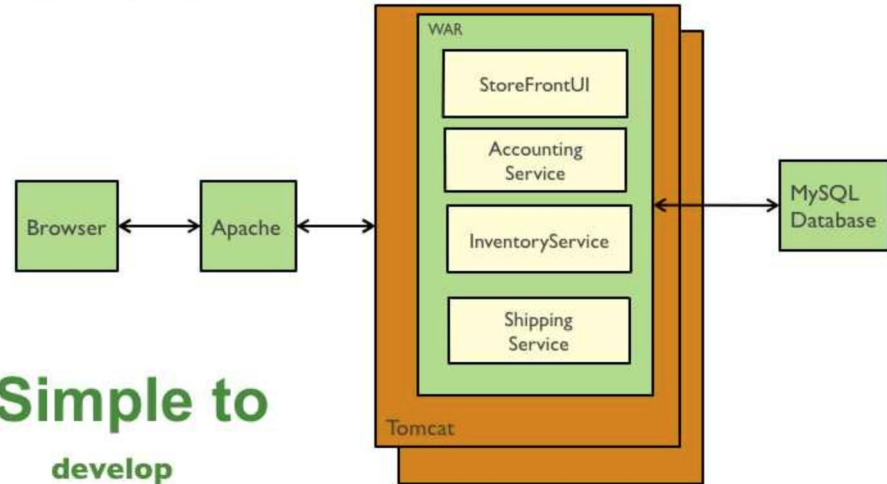


Agenda

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

Monolithic approach

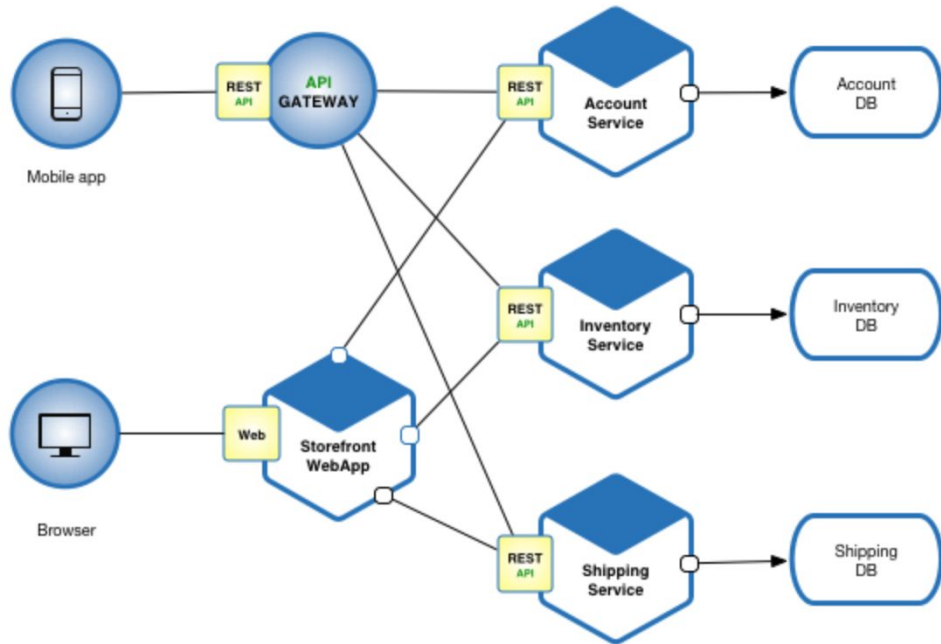
Traditional web application architecture



Simple to

**develop
test
deploy
scale**

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?

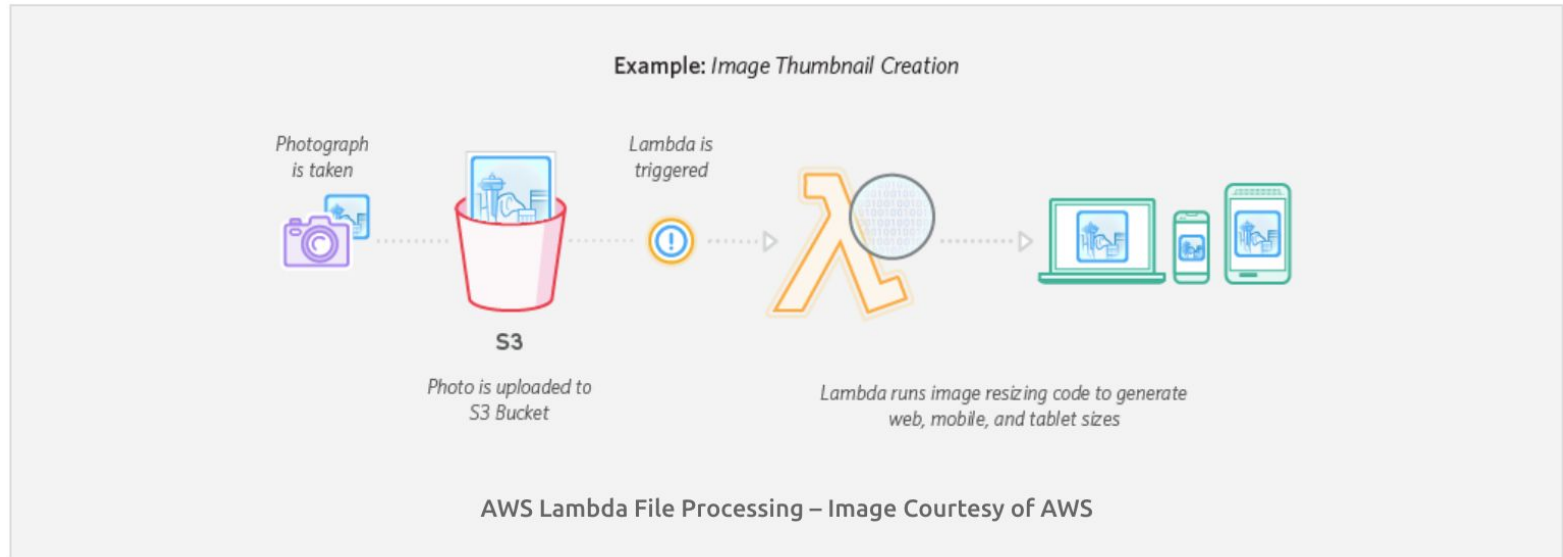
The first benefit of containers is their portability. The main draw of a container is that you can combine the application and all of its dependencies into a neat little package and run it anywhere. This provides an unprecedented level of flexibility and portability, and allows you to stay cloud vendor-agnostic.



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/>
- 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

Find Services

You can enter names, keywords or acronyms.

×

IAM
Manage User Access and Encryption Keys

▼ Recently visited services

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

▼ Set permissions



Add user to group



Copy permissions from
existing user



Attach existing policies
directly


Create policy



Filter policies ▼

Q Search

Showing 438 results

		Policy name ▼	Type	Used as	Description
<input checked="" type="checkbox"/>	▶	 AdministratorAccess	Job function	Permissions policy (2)	Provides full access to AWS services and re...

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





Success

You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: <https://p2p-fancypants.signin.aws.amazon.com/console>

 Download .csv

	User	Access key ID	Secret access key	Email login instructions
▶ 	my_user	AKIAINTOS3UB6H2E2IGQ	***** Show	Send email 

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

Find Services

You can enter names, keywords or acronyms.



Lambda

Run Code without Thinking about Servers

Function name

Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)

Choose the language to use to write your function.





Test your lambda

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

The area below shows the result returned by your function execution. [Learn more](#) about returning results from your function.

```
{
  "statusCode": 200,
  "body": "\"Hello from Lambda!\""
}
```

Summary

Code SHA-256

5pzL2d6zLJi9kHk6xQjMb2JSWieCbdH1tXK7n4nTifs=

Duration

54.47 ms

Resources configured

128 MB

Request ID

74fdffa7-afaf-4f40-9c79-2fa6ace6cbe1

Billed duration

100 ms

Max memory used

71 MB

Log output

The section below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. [Click here](#) to view the CloudWatch log group.

```
START RequestId: 74fdffa7-afaf-4f40-9c79-2fa6ace6cbe1 Version: $LATEST
END RequestId: 74fdffa7-afaf-4f40-9c79-2fa6ace6cbe1
REPORT RequestId: 74fdffa7-afaf-4f40-9c79-2fa6ace6cbe1  Duration: 54.47 ms      Billed Duration: 100 ms      Memory Size: 128 MB
Max Memory Used: 71 MB
```

**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



Introducing AWS DynamoDB! A NOSQL database



Create a DynamoDB table

Find Services

You can enter names, keywords or acronyms.

Q Dynamo|



DynamoDB

Managed NoSQL Database

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 name* ⓘ

Primary key* Partition key

String ⓘ

☒ Add sort key

String ⓘ

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**

Find Services

You can enter names, keywords or acronyms.

API Gateway
Build, Deploy and Manage APIs

Select whether you would like to create a REST API or a WebSocket API.

☒ REST ☐ WebSocket

Create new API

In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

☒ New API ☐ Clone from existing API ☐ Import from Swagger or Open API 3 ☐ Example API

Settings

Choose a friendly name and description for your API.

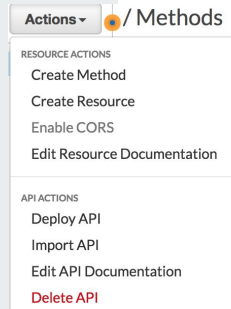
API name*	<input type="text" value="My_API"/>
Description	<input type="text"/>
Endpoint Type	<div>Regional </div>

Create an API Gateway with default settings!

What does this API need?

Create four methods for our four lambdas

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



- 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

▼ /

GET

Choose the integration point for your new method.

Integration type

☒ Lambda Function ⓘ

☐ HTTP ⓘ

☐ Mock ⓘ

☐ AWS Service ⓘ

☐ VPC Link ⓘ

Use Lambda Proxy integration

☐ ⓘ

Lambda Region

eu-west-1

Lambda Function

todo_get ⓘ

Use Default Timeout

☒ ⓘ

Save

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

Find Services

You can enter names, keywords or acronyms.

🔍 cloud9



Cloud9

A Cloud IDE for Writing, Running, and Debugging Code

▼ Recently visited services

The screenshot displays the AWS Cloud9 IDE interface. At the top, a search bar says "Go to Anything (⌘ P)". Below it, a file explorer shows a folder named "todo" containing a "README.md" file. The main workspace area has a dark header with "Developer Tools" and a large welcome message: "AWS Cloud9 Welcome to your development environment". Below this, it states: "AWS Cloud9 allows you to write, run, and debug your code with just a browser. You can [tour the IDE](#), write code for [AWS Lambda and Amazon API Gateway](#), share your IDE with others in real time, and much more."

In the center, there's a section titled "AWS Cloud9 for AWS Lambda" with a description: "AWS Lambda is a compute service that lets you run code without provisioning or managing servers. AWS Lambda executes your code". To the right of this, a "Getting started" panel offers links for "Create File", "Upload Files...", and "Clone from GitHub".

At the bottom, a terminal window is open with the prompt "bash - 'ip-172-31' x Immediate" and the user "todo_user:~/environment".

On the right side, a sidebar shows a list of Lambda functions under the "eu-west-1" region. The functions listed are: "IoD_sensors_location", "IoD_sensors_location_get", "IoD_sensors_location_get_day", "IoD_sensors_rotation", "IoD_sensors_rotation_get", "IoD_sensors_sound", "IoD_sensors_sound_get", "IoD_sensors_steps", "IoD_sensors_velocity", "IoD_sensors_velocity_get", "IoD_sensors_velocity_get_stats", "IoD_sensors_velocity_get_today_", "IoD_sensors_velocity_get_total_r", "IoD_sensors_velocity_today", and "IoD_test_lambda". A context menu is open over the "IoD_test_lambda" function, showing options: "Edit Function", "Edit Config", "Convert to SAM...", "Link to CloudFormation Stack...", "Show in Tree", "Run", "Create Here...", "Deploy", and "Import...". The "Run" option is highlighted.

Import your lambda functions

The image shows a development environment with two main panels. The left panel is a code editor with a file named `index.js`. It contains the following JavaScript code:

```
1 exports.handler = (event, context, callback) => {
2   // TODO implement
3   const response = {
4     statusCode: 200,
5     body: JSON.stringify('Hello from Lambda!'),
6   };
7   callback(null, response);
8 };
9
```

The right panel is a Lambda function test interface for the function `todo_web`. It shows the test payload as an empty object `{}`. Below the payload, the execution results are displayed, showing a successful response:

Execution results Max memory used: 30 MB Time: 2.04 ms

Response

```
{
  "statusCode": 200,
  "body": "\"Hello from Lambda!\""
}
```

Function Logs

Request ID
b1ecbd31-90b8-1310-e5e4-7b19959a625a

At the bottom of the code editor, the status bar indicates: 9:1 JavaScript Spaces: 4

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 ▼

Q dynamo

Showing 7 results

	Policy name ▼	Used as	Description
<input checked="" type="checkbox"/>	 AmazonDynamoDBFullAccess	Permissions policy (3)	Provides full access to Amazon Dynamo...

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

Use an existing role ▼

Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

lambda_for_dynamodb ▼



[View the lambda_for_dynamodb role](#) on the IAM console.

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

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

Manage public permissions

Grant public read access to this object(s)

 **This object(s) has public read access.**
Everyone in the world will have read access to this object(s).

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

Static website hosting ✕

Endpoint : <http://mycoolbucketfortodos.s3-website-eu-west-1.amazonaws.com>

☒ Use this bucket to host a website [Learn more](#)

Index document [i](#)

Error document [i](#)

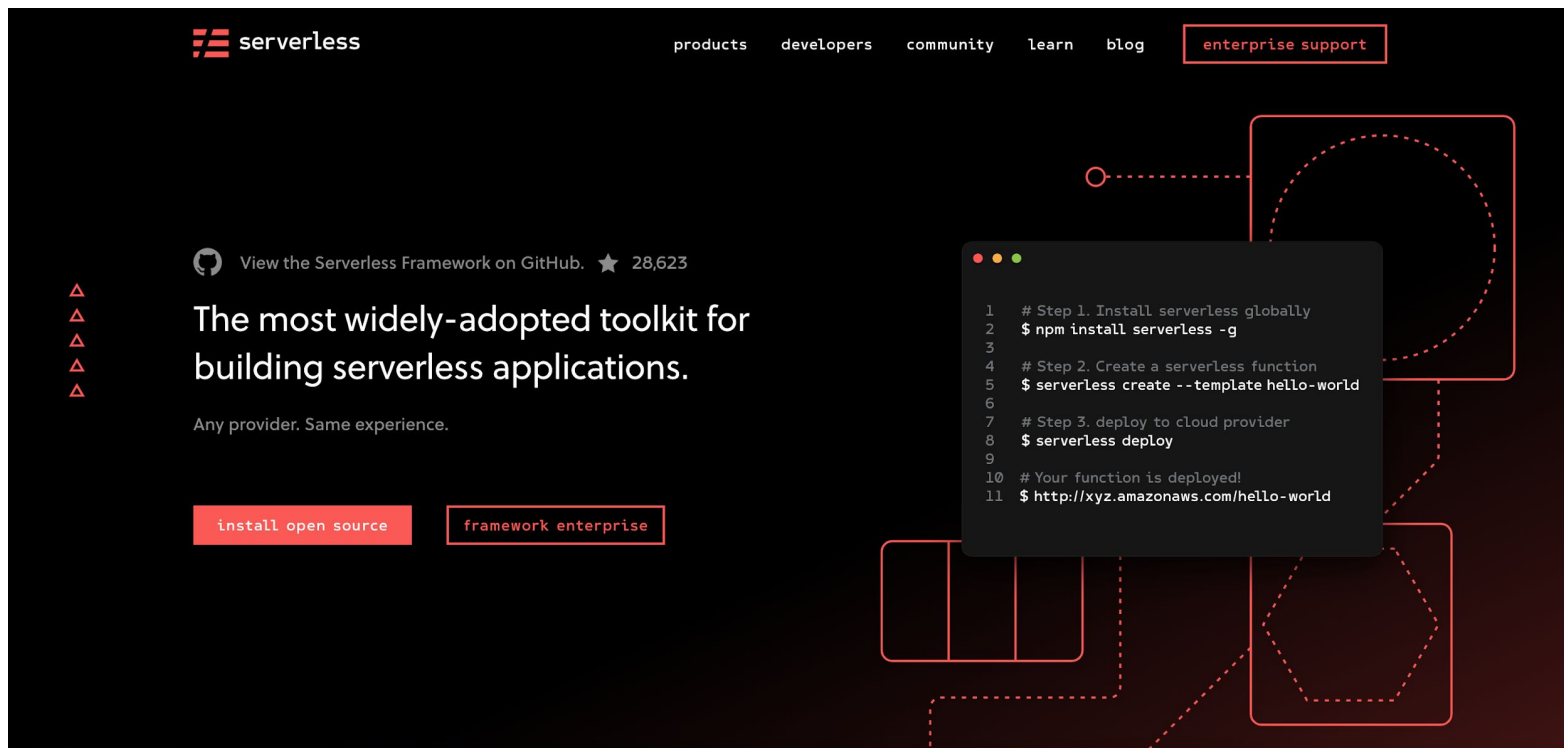
Redirection rules (optional) [i](#)

☐ Redirect requests [Learn more](#)

☐ Disable website hosting

Fast development! Not really..

—

The image shows the Serverless Framework website landing page. The background is dark with abstract orange geometric shapes and dashed lines. The navigation bar at the top includes links for products, developers, community, learn, blog, and enterprise support. The main content area features the Serverless logo, a GitHub link, a headline, a sub-headline, and two buttons. A code window is overlaid on the right side, showing a three-step installation and deployment process.

serverless

products developers community learn blog enterprise support

View the Serverless Framework on GitHub. ★ 28,623

The most widely-adopted toolkit for building serverless applications.

Any provider. Same experience.

install open source framework enterprise

```
1 # Step 1. Install serverless globally
2 $ npm install serverless -g
3
4 # Step 2. Create a serverless function
5 $ serverless create --template hello-world
6
7 # Step 3. deploy to cloud provider
8 $ serverless deploy
9
10 # Your function is deployed!
11 $ http://xyz.amazonaws.com/hello-world
```

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_ACCESS_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