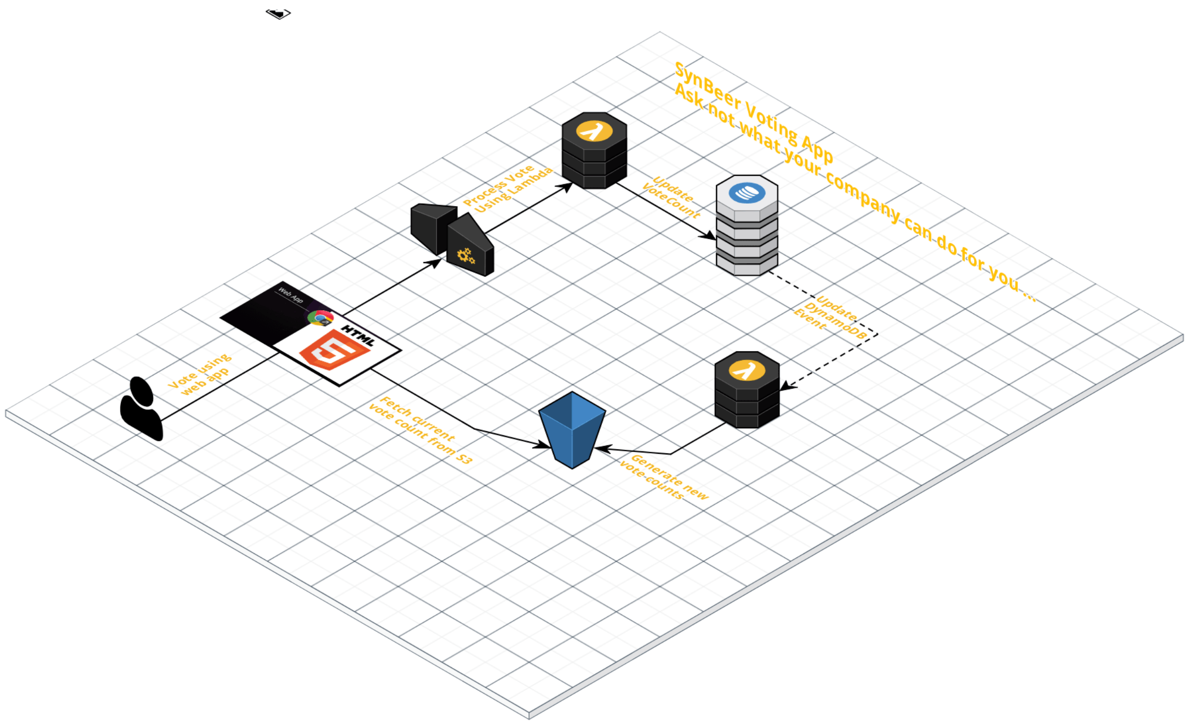
## SynBeerVoting App

JFK was not entirely right - he should have said “Ask not what your company can do for you, but what you can do for your company”.

The SynBeer Vote is the most important vote you will cast this year, as it will (hypothetically) determine the kind of beer style we will be brewing this year. However, we will need to build a web application first to allow people to vote and to see what the current score (votes per beer style) is.

As we are quite well equipped with AWS Lambda now, the beer gods have decided the web app should be built as a static AWS S3 website, implementing the relevant interactions with AWS Lambda functions.

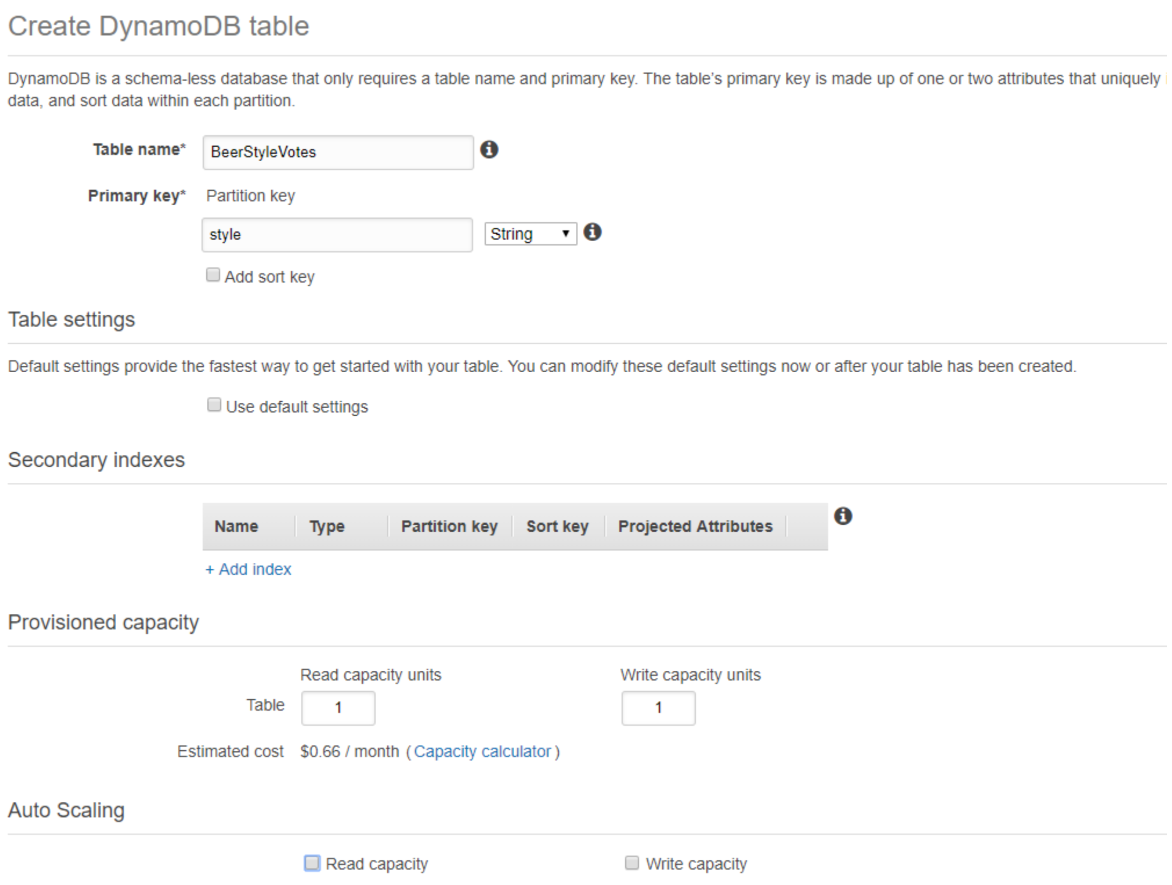


## NoSQL: Create a table in DynamoDB

First we need to setup a NoSQL database table to store the votes; navigate to Amazon’s DynamoDB console and define a simple table to hold the votes: name the table **BeerStyleVotes**, the primary key is to be called style and is a string.

This is all DynamoDB at its simplest require to store data, a table name and a primary key. DynamoDB does not care what else is stored inside the document … Hey, it’s NoSQL – no structure required (except for the primary key).

Make sure to **uncheck** the autoscaling on both read and write and to reduce the provision read and write capacity to 1:



Create 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
data, and sort data within each partition.

Table name* BeerStyleVotes

Primary key* Partition key

style String

☐ Add sort key

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

Secondary indexes

| Name | Type | Partition key | Sort key | Projected Attributes |
|---|---|---|---|---|

+ Add index

Provisioned capacity

| | Read capacity units | Write capacity units |
|---|---|---|
| Table | 1 | 1 |

Estimated cost $0.66 / month ( Capacity calculator )

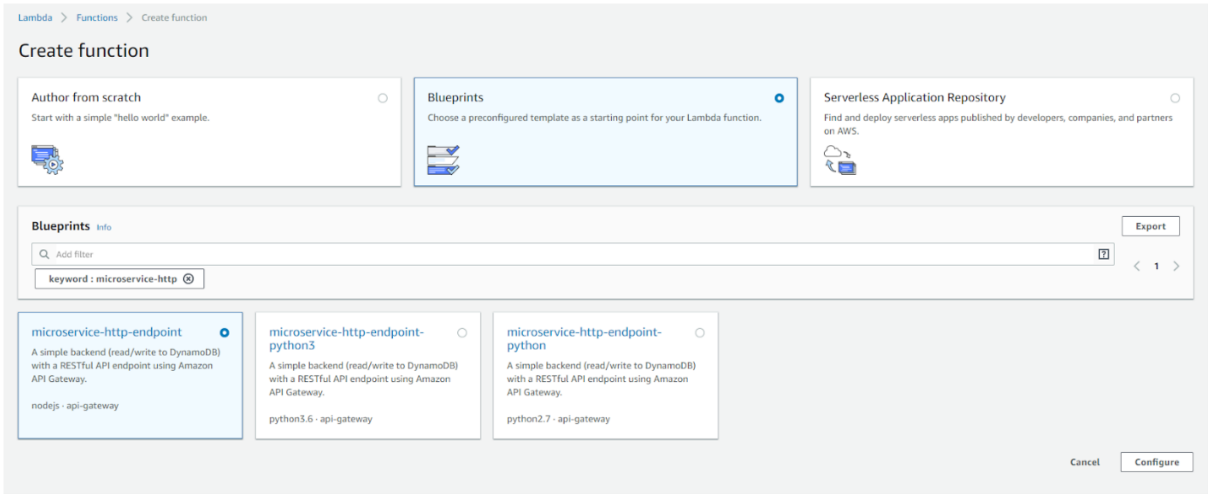
Auto Scaling

☐ Read capacity ☐ Write capacity

## Lambda: function SyntouchBeerVoter

Next up is a lambda function to process the incoming vote, i.e. to register a vote in the DynamoDB table.

Create the function from the AWS Lambda console, by using a standard blueprint: microservice-http-endpoint:



Lambda > Functions > Create function

Create function

Author from scratch
Start with a simple "hello world" example.

[✓ Blueprints]
Choose a preconfigured template as a starting point for your Lambda function.

Serverless Application Repository
Find and deploy serverless apps published by developers, companies, and partners on AWS.

Blueprints Info

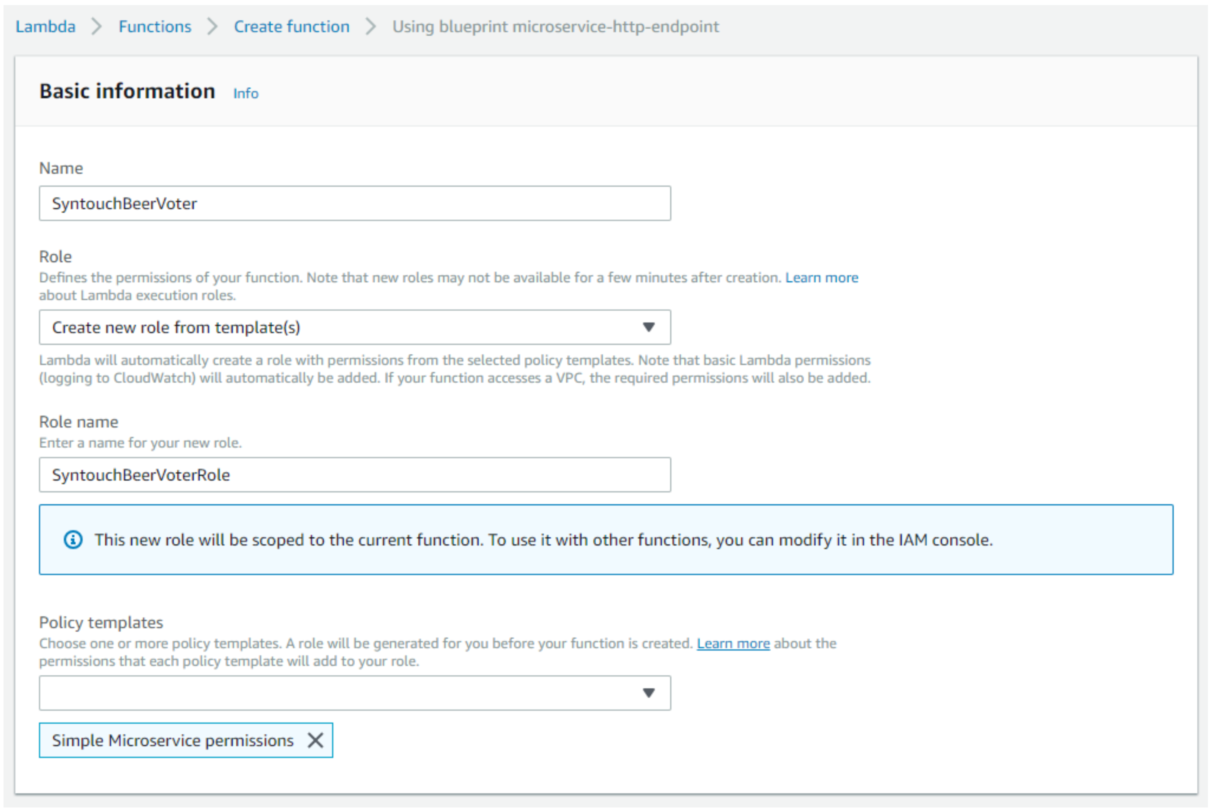
Add filter

keyword : microservice-http

[✓ microservice-http-endpoint]
A simple backend (read/write to DynamoDB) with a RESTful API endpoint using Amazon API Gateway.
nodejs · api-gateway

microservice-http-endpoint-python3
A simple backend (read/write to DynamoDB) with a RESTful API endpoint using Amazon API Gateway.
python3.6 · api-gateway

microservice-http-endpoint-python
A simple backend (read/write to DynamoDB) with a RESTful API endpoint using Amazon API Gateway.
python2.7 · api-gateway


Lambda > Functions > Create function > Using blueprint microservice-http-endpoint

**Basic information** Info

Name

SyntouchBeerVoter

Role

Defines the permissions of your function. Note that new roles may not be available for a few minutes after creation. Learn more about Lambda execution roles.

Create new role from template(s)

Lambda will automatically create a role with permissions from the selected policy templates. Note that basic Lambda permissions (logging to CloudWatch) will automatically be added. If your function accesses a VPC, the required permissions will also be added.

Role name

Enter a name for your new role.

SyntouchBeerVoterRole

This new role will be scoped to the current function. To use it with other functions, you can modify it in the IAM console.

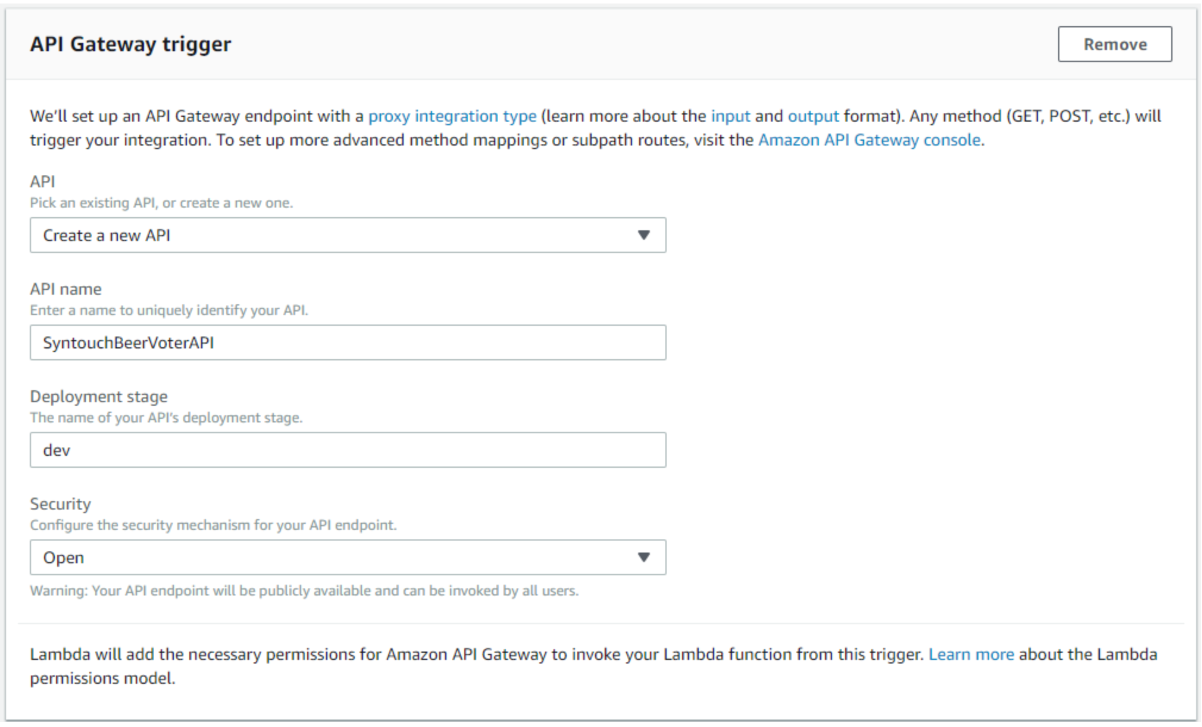
Policy templates

Choose one or more policy templates. A role will be generated for you before your function is created. Learn more about the permissions that each policy template will add to your role.

Simple Microservice permissions

Check what permissions you're adding by using the standard set of "Simple Microservice permissions" (https://docs.aws.amazon.com/lambda/latest/dg/policy-templates.html#MicroServiceExecutionRole)

Also define the trigger (API Gateway definition) from the same form:


**API Gateway trigger**

We'll set up an API Gateway endpoint with a proxy integration type (learn more about the input and output format). Any method (GET, POST, etc.) will trigger your integration. To set up more advanced method mappings or subpath routes, visit the Amazon API Gateway console.

API

Pick an existing API, or create a new one.

Create a new API

API name

Enter a name to uniquely identify your API.

SyntouchBeerVoterAPI

Deployment stage

The name of your API's deployment stage.

dev

Security

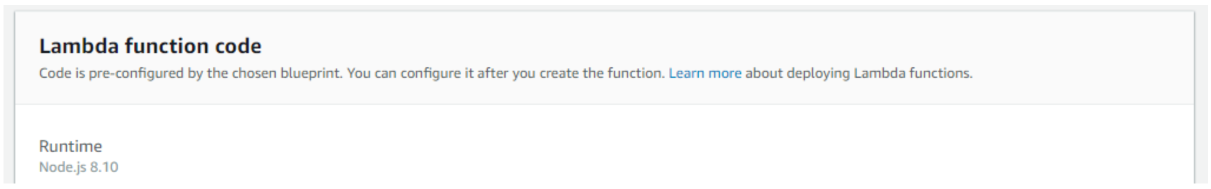
Configure the security mechanism for your API endpoint.

Open

Warning: Your API endpoint will be publicly available and can be invoked by all users.

Lambda will add the necessary permissions for Amazon API Gateway to invoke your Lambda function from this trigger. Learn more about the Lambda permissions model.

At this stage, you cannot define the function's code - you first need to create the function and the edit the function code inline:



**Lambda function code**

Code is pre-configured by the chosen blueprint. You can configure it after you create the function. Learn more about deploying Lambda functions.

Runtime

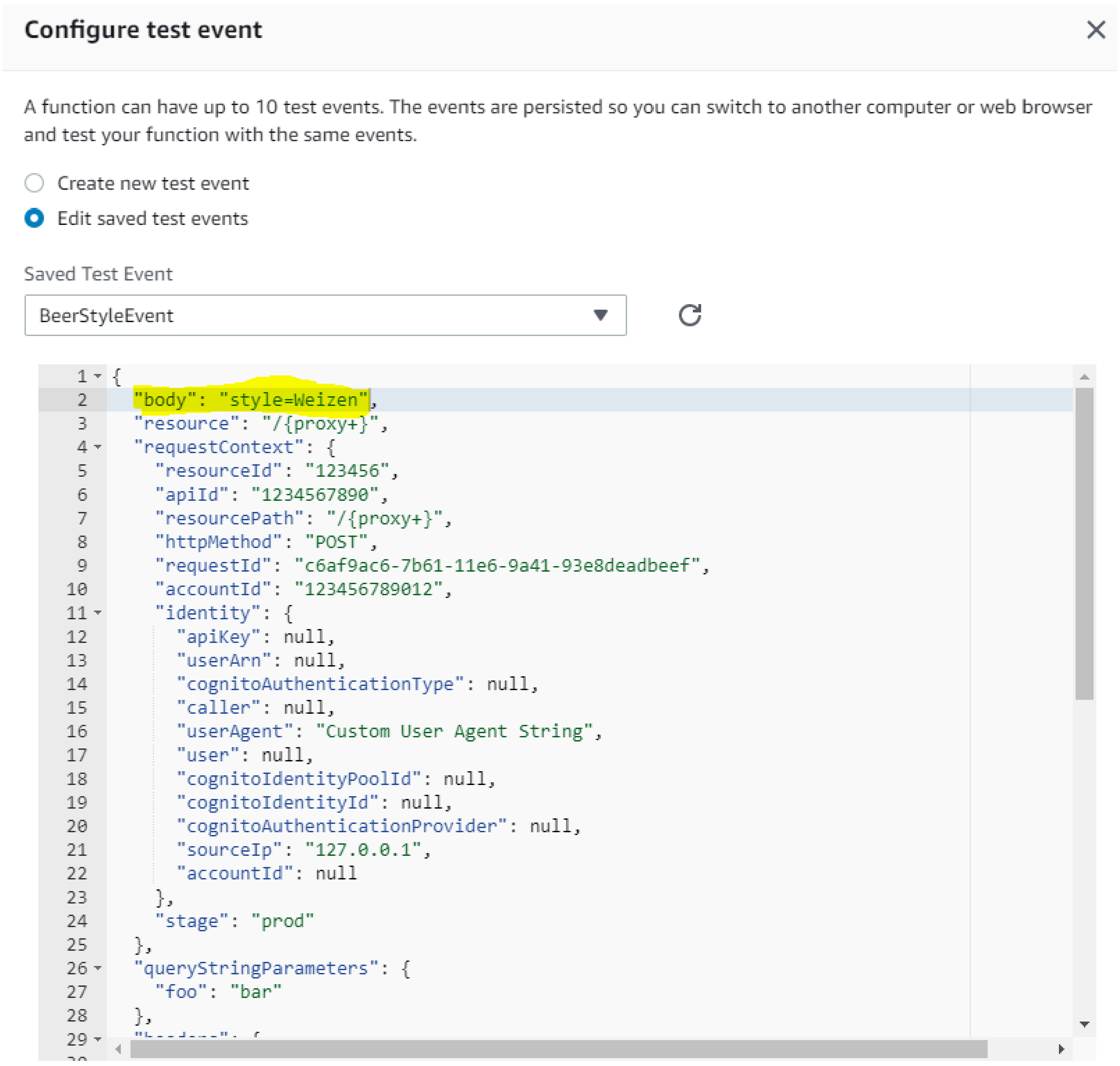
Node.js 8.10

Implement the code to trigger and update on the DynamoDB table:

```
'use strict';

console.log('Loading function');

const doc = require('dynamodb-doc');
const dynamo = new doc.DynamoDB();

const beervote = (payload, callback) => {

    var params = {
        Key: {
            style: payload.style
        },
        TableName: 'BeerStyleVotes',
            AttributeUpdates: {
                counter: {
                    Action: 'ADD',
                    Value: 1
                }
            }
    };

    console.log('Casting Vote:', JSON.stringify(params, null, 2));

    dynamo.updateItem(params, (err, data) => {
        if (err) console.log(err, err.stack); // an error occurred
        else     console.log(data);           // successful response

        callback(err, data);
    });
}

exports.handler = (event, context, callback) => {
    console.log('Received event:', JSON.stringify(event, null, 2));

    const done = (err, res) => callback(null, {
        statusCode: err ? '400' : '204',
        body: err ? err.message : JSON.stringify(res),
        headers: {
            'Content-Type': 'application/json',
        },
    });


    switch (event.httpMethod) {
        case 'POST':
            // body is sending "style=Weizen" objects, so just need to
            // parse out the value after the equal sign!
            beervote({
                    "style" : event.body.split('=')[1]
                }, done);
            break;
        default:
            done(new Error(`Unsupported method "${event.httpMethod}"`));
    }
};
```

## Testing, testing

Save the function and create a test event; the event structure is now an API Gateway proxy event type, the body should contain an attribute "style" with a value of your preference:



**Configure test event**

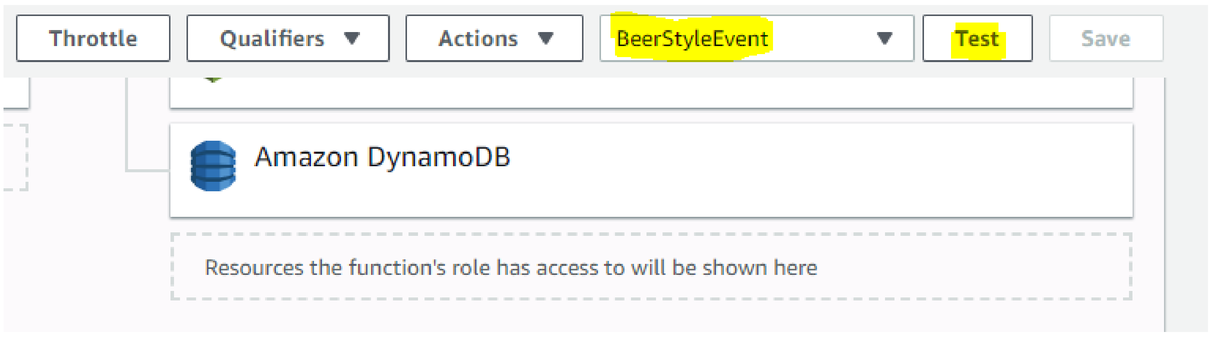
A function can have up to 10 test events. The events are persisted so you can switch to another computer or web browser and test your function with the same events.

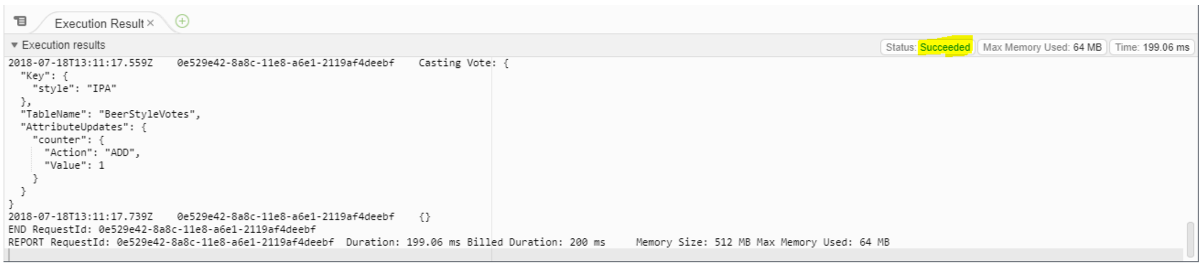
Create new test event [✓ Edit saved test events]

Saved Test Event: BeerStyleEvent

```
{
  "body": "style=Weizen",
  "resource": "/{proxy+}",
  "requestContext": {
    "resourceId": "123456",
    "apiId": "1234567890",
    "resourcePath": "/{proxy+}",
    "httpMethod": "POST",
    "requestId": "c6af9ac6-7b61-11e6-9a41-93e8deadbeef",
    "accountId": "123456789012",
    "identity": {
      "apiKey": null,
      "userArn": null,
      "cognitoAuthenticationType": null,
      "caller": null,
      "userAgent": "Custom User Agent String",
      "user": null,
      "cognitoIdentityPoolId": null,
      "cognitoIdentityId": null,
      "cognitoAuthenticationProvider": null,
      "sourceIp": "127.0.0.1",
      "accountId": null
    },
    "stage": "prod"
  },
  "queryStringParameters": {
    "foo": "bar"
  },
```

Perform the test and verify it completes successfully:

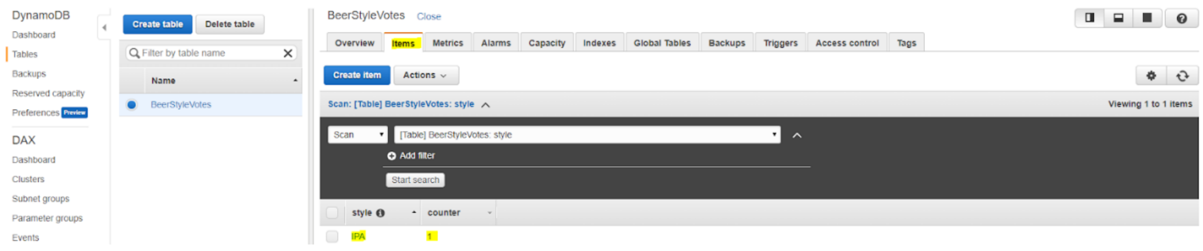

Throttle
Qualifiers
Actions
BeerStyleEvent
Test
Save
Amazon DynamoDB
Resources the function's role has access to will be shown here



```
2018-07-18T13:11:17.559Z    0e529e42-8a8c-11e8-a6e1-2119af4deebf    Casting Vote: {
  "Key": {
    "style": "IPA"
  },
  "TableName": "BeerStyleVotes",
  "AttributeUpdates": {
    "counter": {
      "Action": "ADD",
      "Value": 1
    }
  }
}
2018-07-18T13:11:17.739Z    0e529e42-8a8c-11e8-a6e1-2119af4deebf    {}
END RequestId: 0e529e42-8a8c-11e8-a6e1-2119af4deebf
REPORT RequestId: 0e529e42-8a8c-11e8-a6e1-2119af4deebf  Duration: 199.06 ms Billed Duration: 200 ms     Memory Size: 512 MB Max Memory Used: 64 MB
```

### *Did it work?*

Navigate to your table and verify the document has been inserted properly:



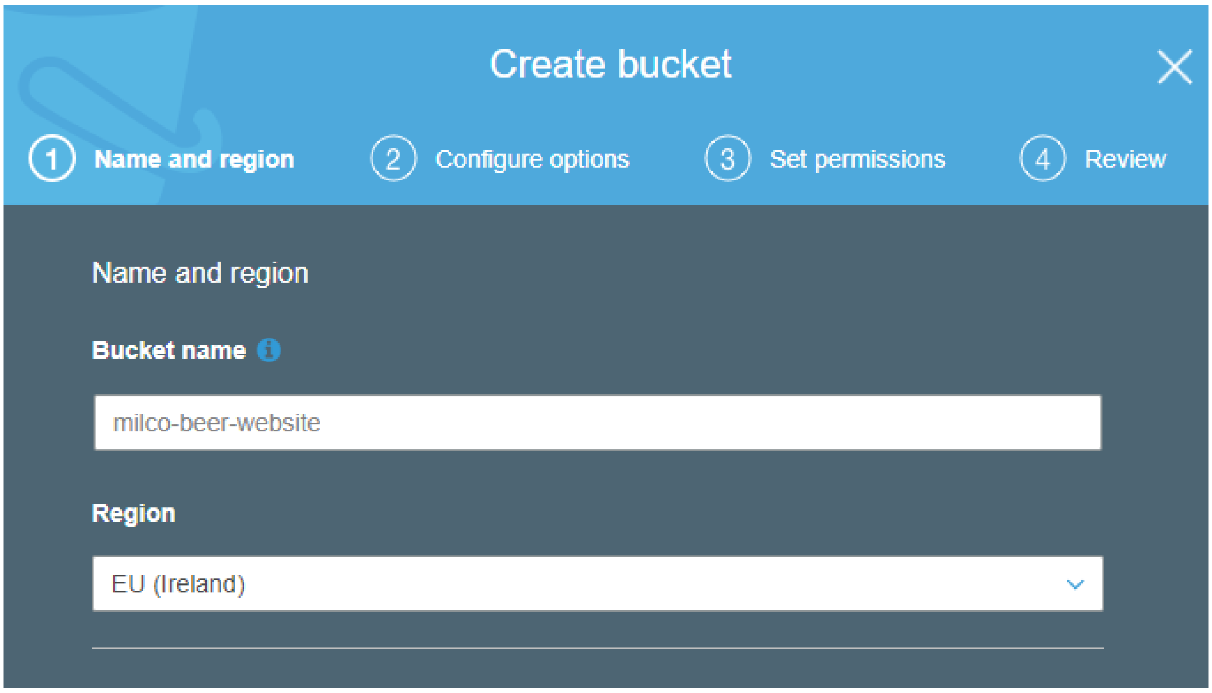
| style | counter |
|---|---|
| IPA | 1 |

Repeat the testing and verify that the counter was actually updated to 2!

## Hosting a website

Amazon S3 is not only an object store, but can also be used to host (static) website items, like HTML pages, stylesheets, JavaScript files, images and the like.

First we'll create a new bucket for the beerwebsite (bucketnames must be **globally unique** - prefix with your name or initials):



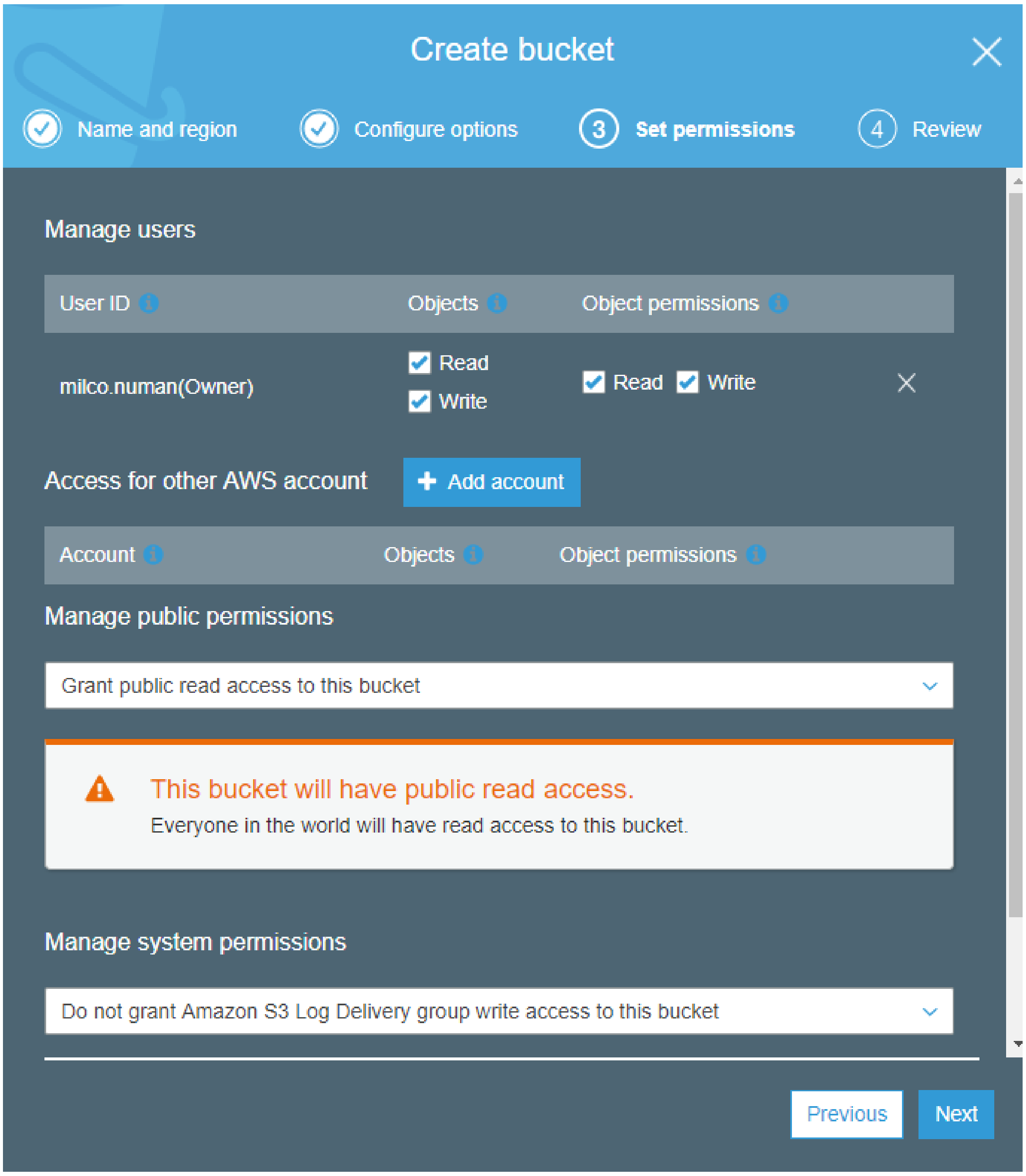
Name and region

Bucket name: milco-beer-website

Region: EU (Ireland)

No options

Set for public read access:


Create bucket

✓ Name and region ✓ Configure options 3 Set permissions 4 Review

Manage users

| User ID | Objects | Object permissions |
|---|---|---|
| milco.numan(Owner) | ✓ Read ✓ Write | ✓ Read ✓ Write |

Access for other AWS account

| Account | Objects | Object permissions |
|---|---|---|

Manage public permissions

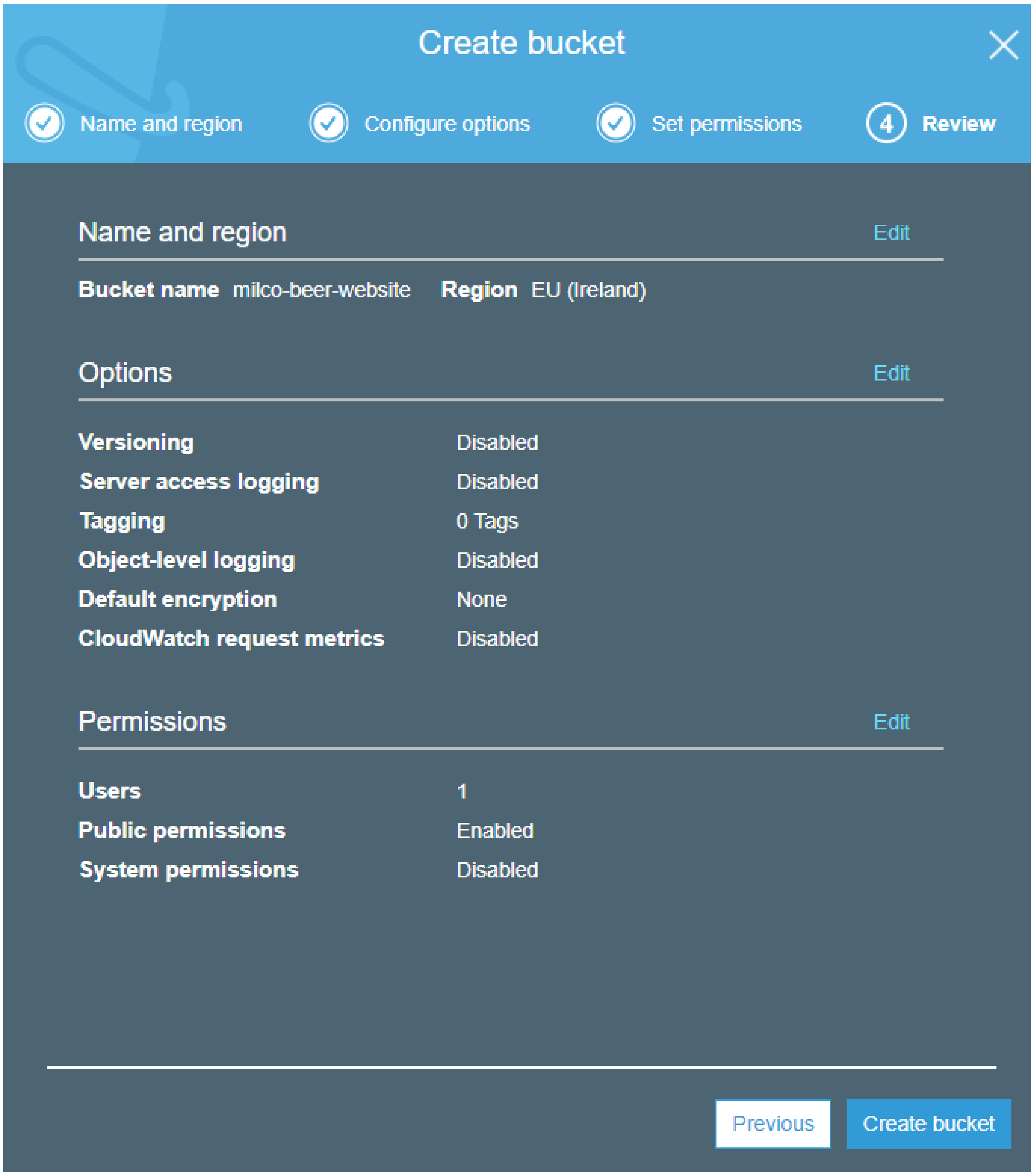
Grant public read access to this bucket

This bucket will have public read access.
Everyone in the world will have read access to this bucket.

Manage system permissions

Do not grant Amazon S3 Log Delivery group write access to this bucket

Review and create  bucket:



Name and region

Bucket name milco-beer-website Region EU (Ireland)

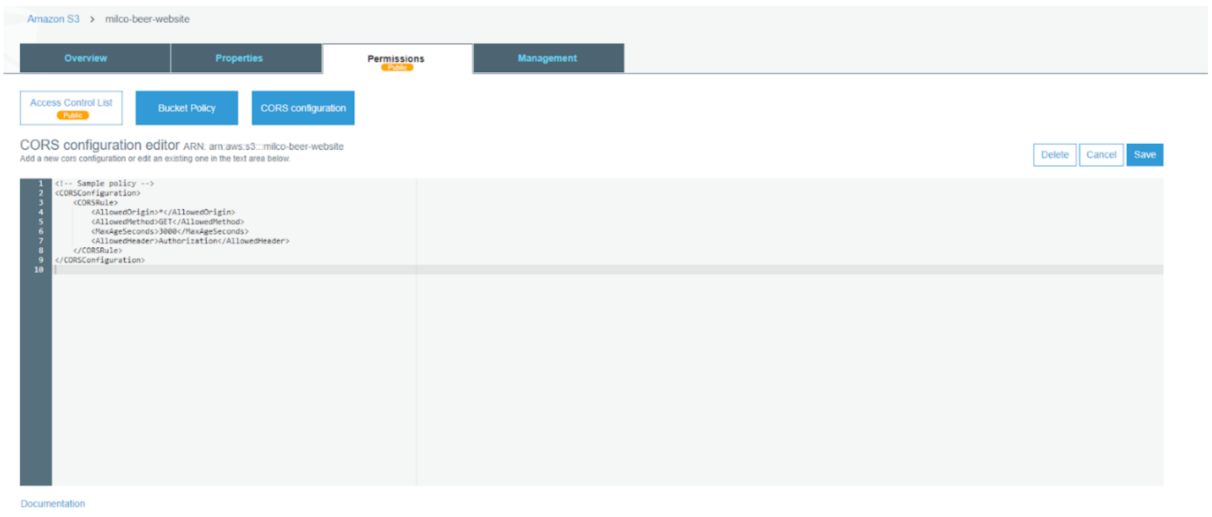
Options

| | |
|---|---|
| Versioning | Disabled |
| Server access logging | Disabled |
| Tagging | 0 Tags |
| Object-level logging | Disabled |
| Default encryption | None |
| CloudWatch request metrics | Disabled |

Permissions

| | |
|---|---|
| Users | 1 |
| Public permissions | Enabled |
| System permissions | Disabled |

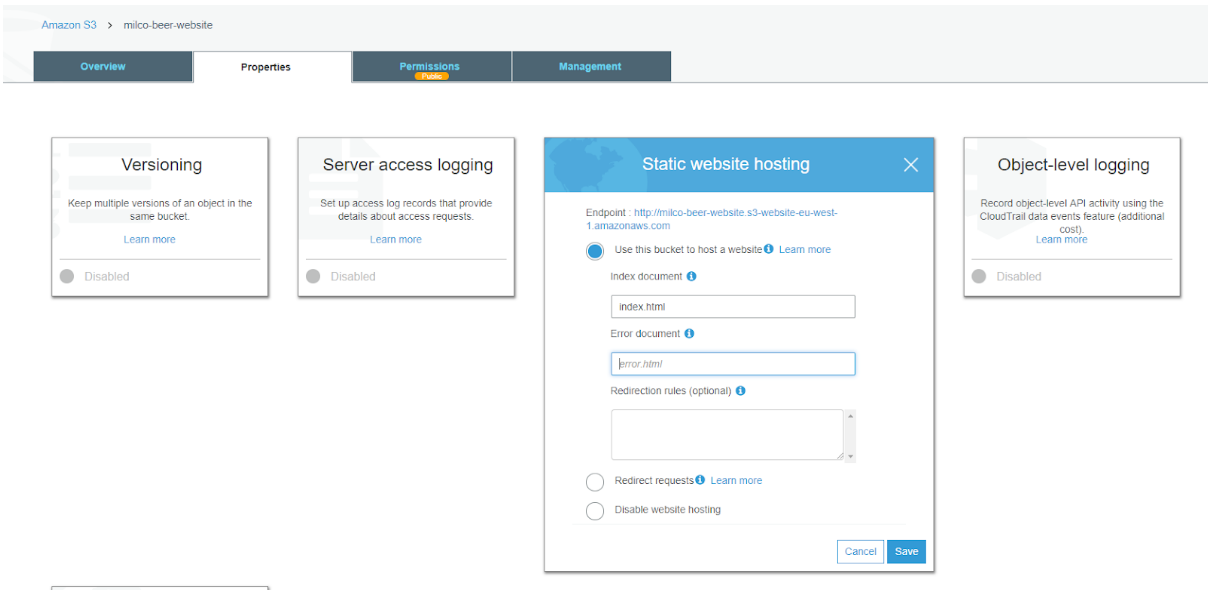
Allow CORS (Cross Origin Resource Sharing) - the configuration is prefilled and can be accepted, just click save:



CORS configuration editor ARN: arn:aws:s3:::milco-beer-website
Add a new cors configuration or edit an existing one in the text area below.

```
<!-- Sample policy -->
<CORSConfiguration>
    <CORSRule>
        <AllowedOrigin>*</AllowedOrigin>
        <AllowedMethod>GET</AllowedMethod>
        <MaxAgeSeconds>3000</MaxAgeSeconds>
        <AllowedHeader>Authorization</AllowedHeader>
    </CORSRule>
</CORSConfiguration>
```

Enable this bucket for hosting a static website; specify the index document to be the standard index.html, leave the error document blank:



Keep multiple versions of an object in the same bucket.
Disabled

Set up access log records that provide details about access requests.
Disabled

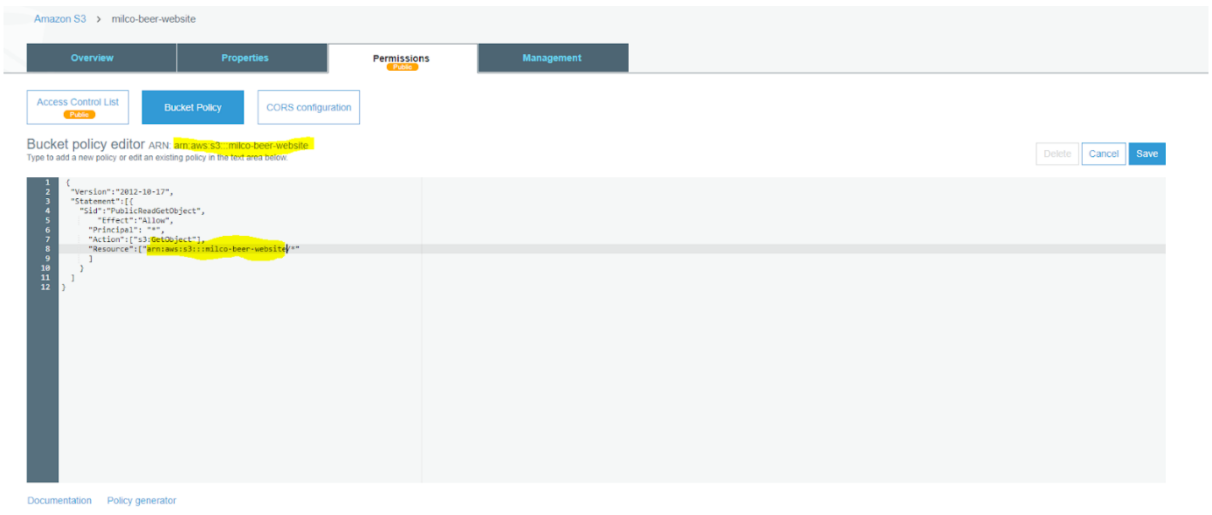
Endpoint : http://milco-beer-website.s3-website-eu-west-1.amazonaws.com
[✓ Use this bucket to host a website]
Index document: index.html
Error document: error.html
Redirection rules (optional)
Redirect requests
Disable website hosting

Record object-level API activity using the CloudTrail data events feature (additional cost).
Disabled

Navigate back to the bucket's permission's tab, select bucket policy and copy the bucket policy from the AWS documentation:

https://docs.aws.amazon.com/AmazonS3/latest/dev/WebsiteAccessPermissionsReqd.html

**Be sure to overwrite the resource (that is the bucketname) with your bucket name:**

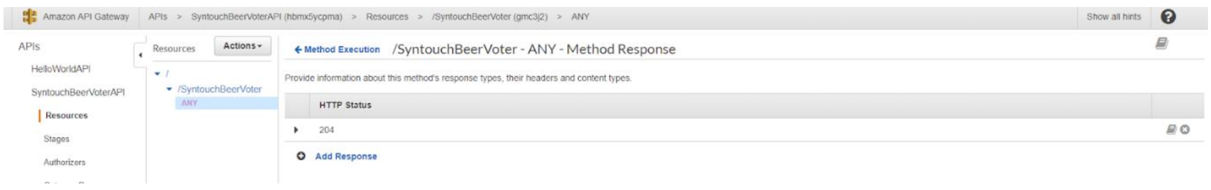


Bucket policy editor ARN: arn:aws:s3:::milco-beer-website
Type to add a new policy or edit an existing policy in the text area below.

```
{
  "Version":"2012-10-17",
  "Statement":[{
    "Sid":"PublicReadGetObject",
        "Effect":"Allow",
      "Principal": "*",
      "Action":["s3:GetObject"],
      "Resource":["arn:aws:s3:::milco-beer-website/*"
      ]
    }
  ]
}
```

## API Gateway – API response

Navigate to API gateway, remove the default HTTP/200 response code and add an HTTP/204 (No Content) response code:

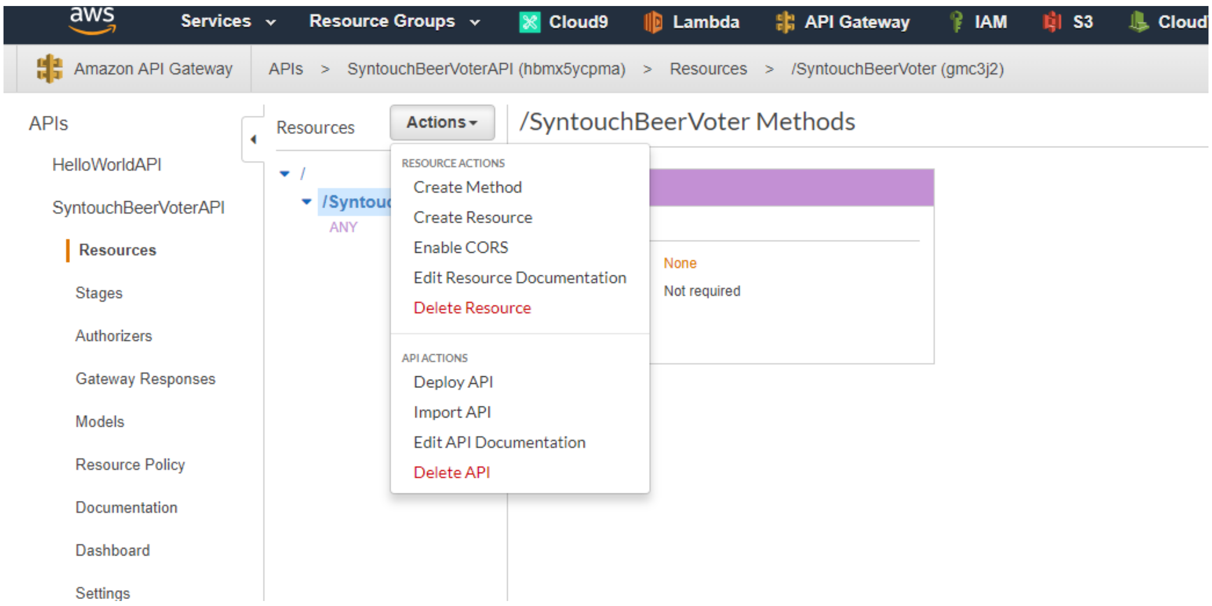


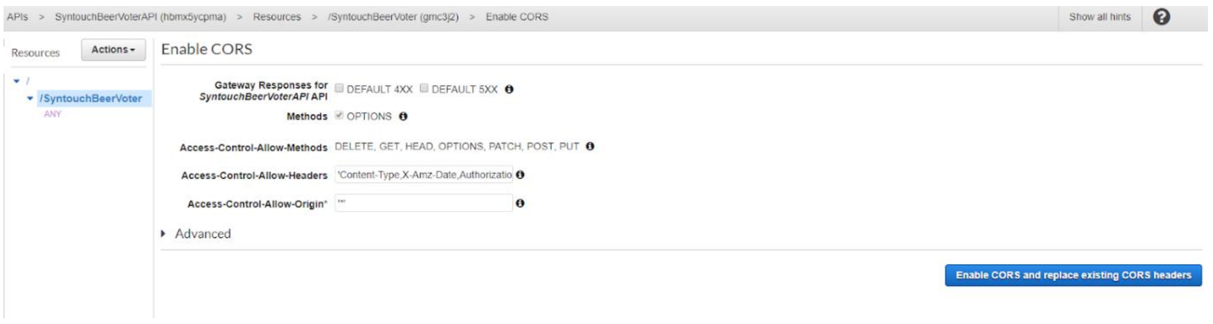
/SyntouchBeerVoter - ANY - Method Response

Provide information about this method's response types, their headers and content types.

| HTTP Status |
|---|
| 204 |

Now, enable CORS on the API as well:




APIs > SyntouchBeerVoterAPI (hbmx5ycpma) > Resources > /SyntouchBeerVoter (gmc3j2) > Enable CORS
Show all hints
Resources
Actions
/
/SyntouchBeerVoter
ANY
Enable CORS
Advanced
Enable CORS and replace existing CORS headers

Gateway Responses for *SyntouchBeerVoterAPI* API: DEFAULT 4XX, DEFAULT 5XX
Methods: [✓ OPTIONS]
Access-Control-Allow-Methods: DELETE, GET, HEAD, OPTIONS, PATCH, POST, PUT
Access-Control-Allow-Headers: 'Content-Type,X-Amz-Date,Authorizatio
Access-Control-Allow-Origin*: '*'

Confirm to replace existing values when asked.

## Lambda Updating data file

Whenever an update to the DynamoDB table occurs, we need to  generate a new data.json file as the votes have changed.

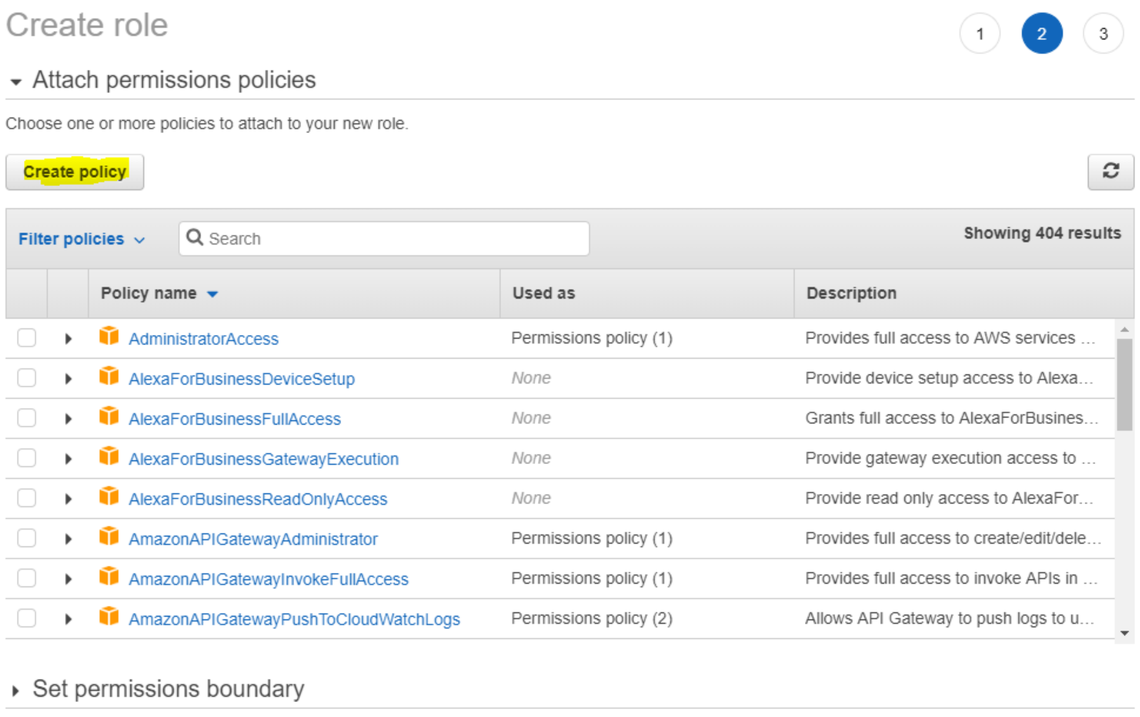
### First IAM

First, let's create a new role that allows the BeerCounter lambda function to create log events and also allows it to create or modify files in the bucket used for our website.

Navigate to the IAM console and create a new role.

### Create a Role for AWS Lambda

On the next page, create a new policy (a new tab will appear):

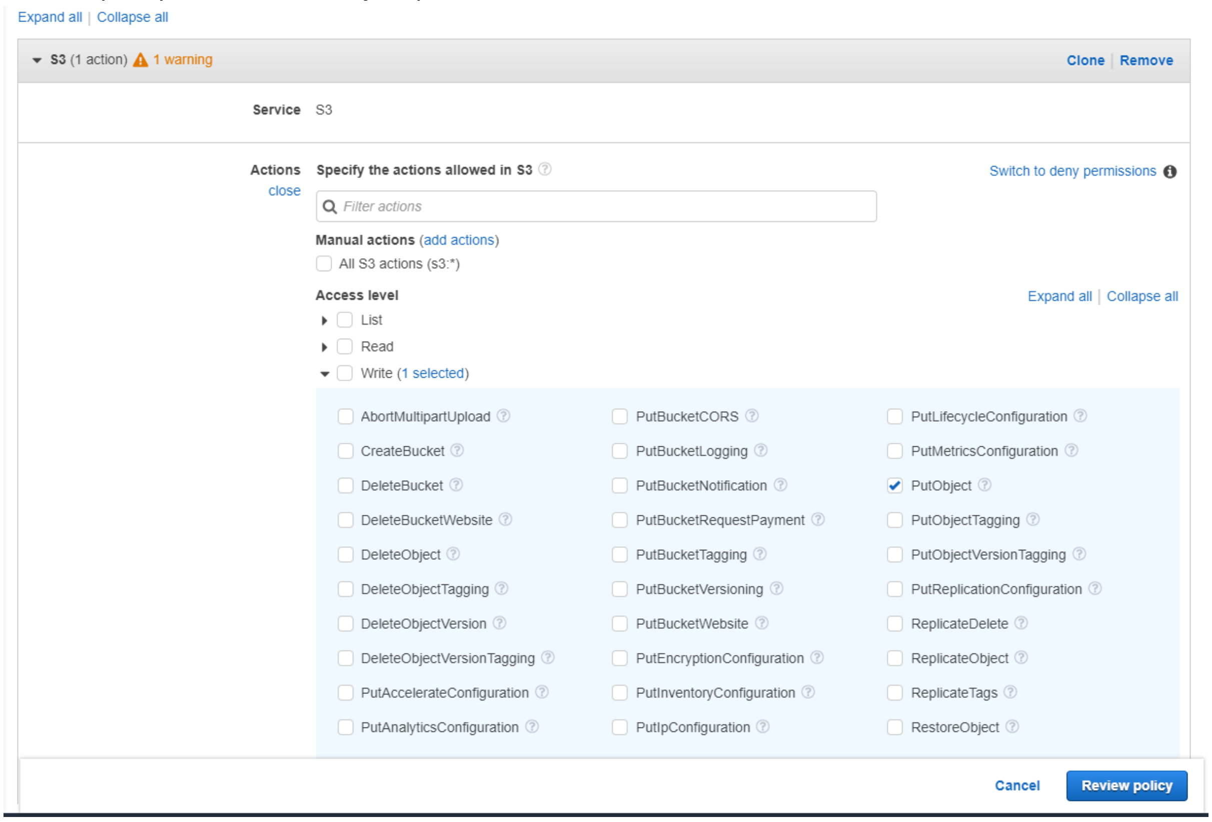


Choose one or more policies to attach to your new role.

Showing 404 results

| Policy name | Used as | Description |
|---|---|---|
| AdministratorAccess | Permissions policy (1) | Provides full access to AWS services ... |
| AlexaForBusinessDeviceSetup | *None* | Provide device setup access to Alexa... |
| AlexaForBusinessFullAccess | *None* | Grants full access to AlexaForBusines... |
| AlexaForBusinessGatewayExecution | *None* | Provide gateway execution access to ... |
| AlexaForBusinessReadOnlyAccess | *None* | Provide read only access to AlexaFor... |
| AmazonAPIGatewayAdministrator | Permissions policy (1) | Provides full access to create/edit/dele... |
| AmazonAPIGatewayInvokeFullAccess | Permissions policy (1) | Provides full access to invoke APIs in ... |
| AmazonAPIGatewayPushToCloudWatchLogs | Permissions policy (2) | Allows API Gateway to push logs to u... |

For this policy, allow S3 PutObject permissions:



Expand all | Collapse all

S3 (1 action) 1 warning — Clone | Remove

Service S3

Actions (close) — Specify the actions allowed in S3 — Switch to deny permissions

Filter actions

Manual actions (add actions)

- [ ] All S3 actions (s3:*)

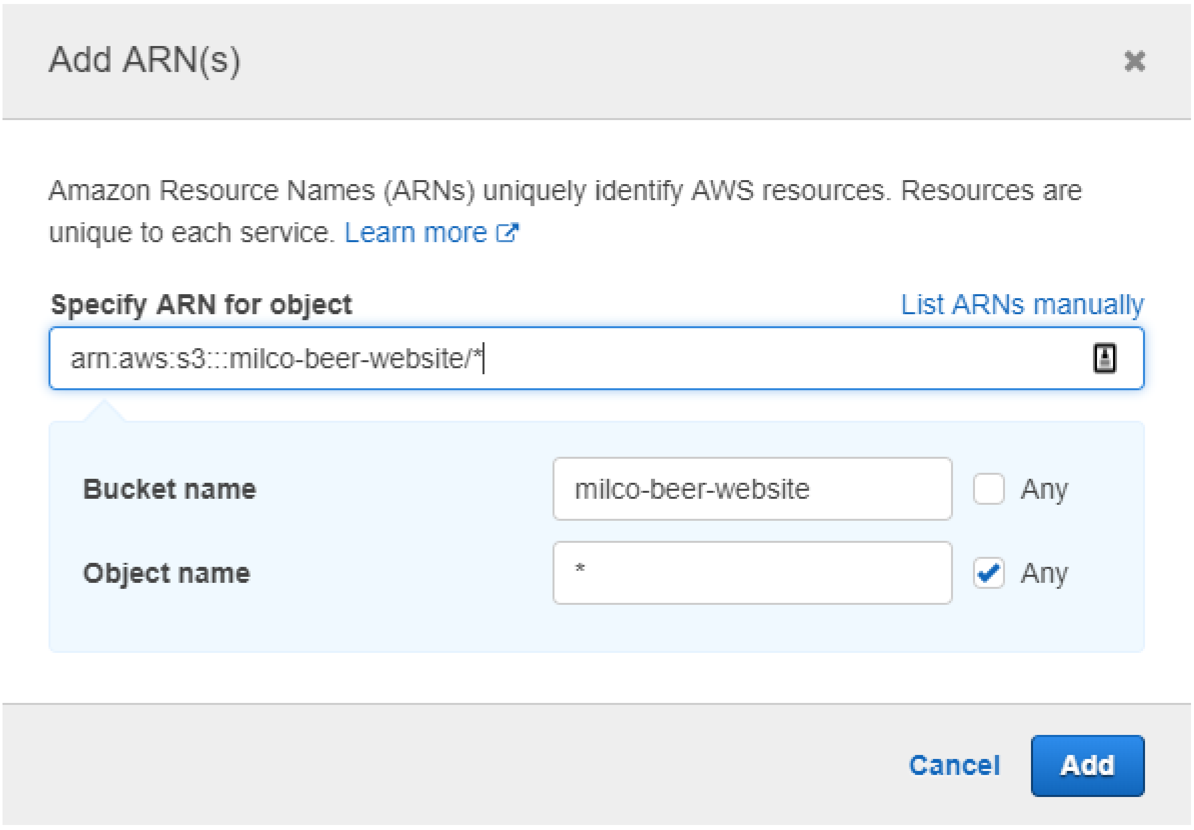
Access level — Expand all | Collapse all

- [ ] List
- [ ] Read
- [ ] Write (1 selected)

| | | |
|---|---|---|
| [ ] AbortMultipartUpload | [ ] PutBucketCORS | [ ] PutLifecycleConfiguration |
| [ ] CreateBucket | [ ] PutBucketLogging | [ ] PutMetricsConfiguration |
| [ ] DeleteBucket | [ ] PutBucketNotification | [✓] PutObject |
| [ ] DeleteBucketWebsite | [ ] PutBucketRequestPayment | [ ] PutObjectTagging |
| [ ] DeleteObject | [ ] PutBucketTagging | [ ] PutObjectVersionTagging |
| [ ] DeleteObjectTagging | [ ] PutBucketVersioning | [ ] PutReplicationConfiguration |
| [ ] DeleteObjectVersion | [ ] PutBucketWebsite | [ ] ReplicateDelete |
| [ ] DeleteObjectVersionTagging | [ ] PutEncryptionConfiguration | [ ] ReplicateObject |
| [ ] PutAccelerateConfiguration | [ ] PutInventoryConfiguration | [ ] ReplicateTags |
| [ ] PutAnalyticsConfiguration | [ ] PutIpConfiguration | [ ] RestoreObject |

Cancel | Review policy

Specify the ARN for your bucket:



Add ARN(s)

Amazon Resource Names (ARNs) uniquely identify AWS resources. Resources are unique to each service. Learn more

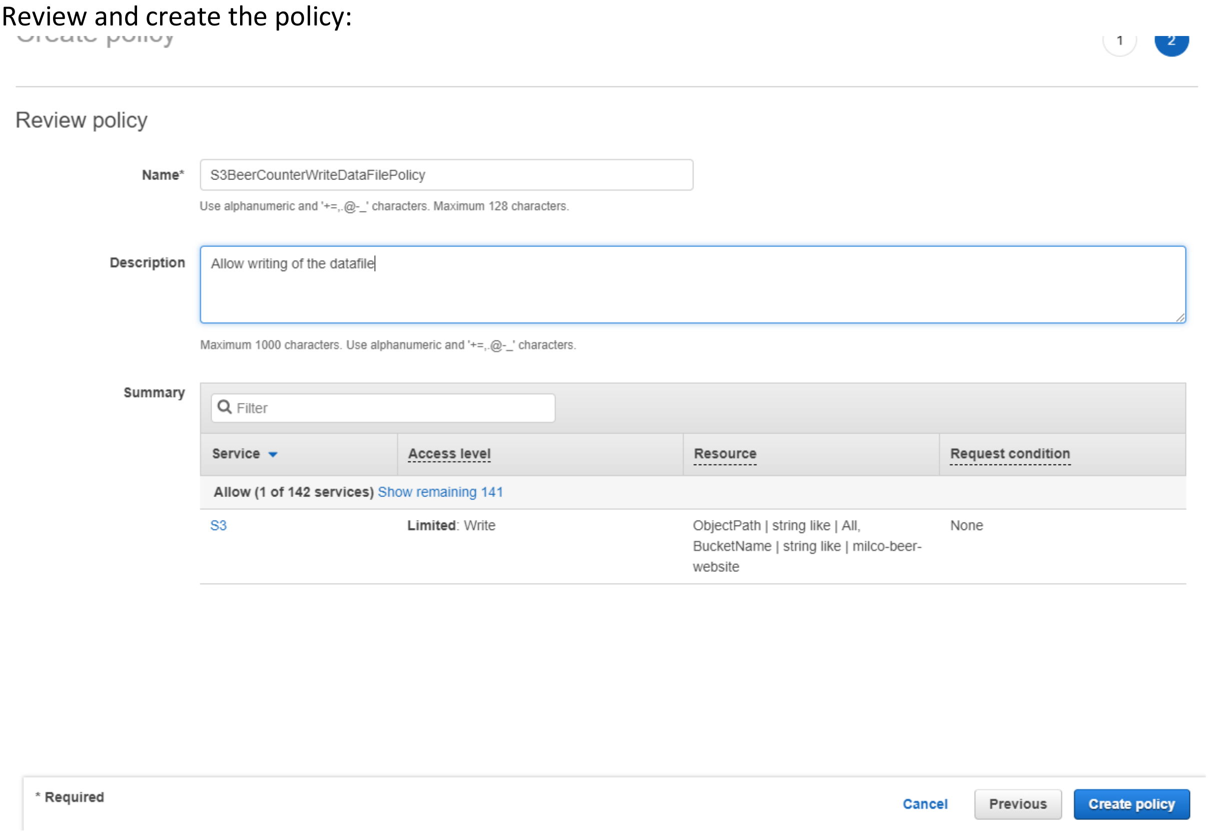
Specify ARN for object — List ARNs manually

arn:aws:s3:::milco-beer-website/*

| | | |
|---|---|---|
| Bucket name | milco-beer-website | [ ] Any |
| Object name | * | [✓] Any |

Cancel | Add

Review and create the policy:



Create policy

1 2

Review policy

Name* S3BeerCounterWriteDataFilePolicy

Use alphanumeric and '+=,.@-_' characters. Maximum 128 characters.

Description Allow writing of the datafile

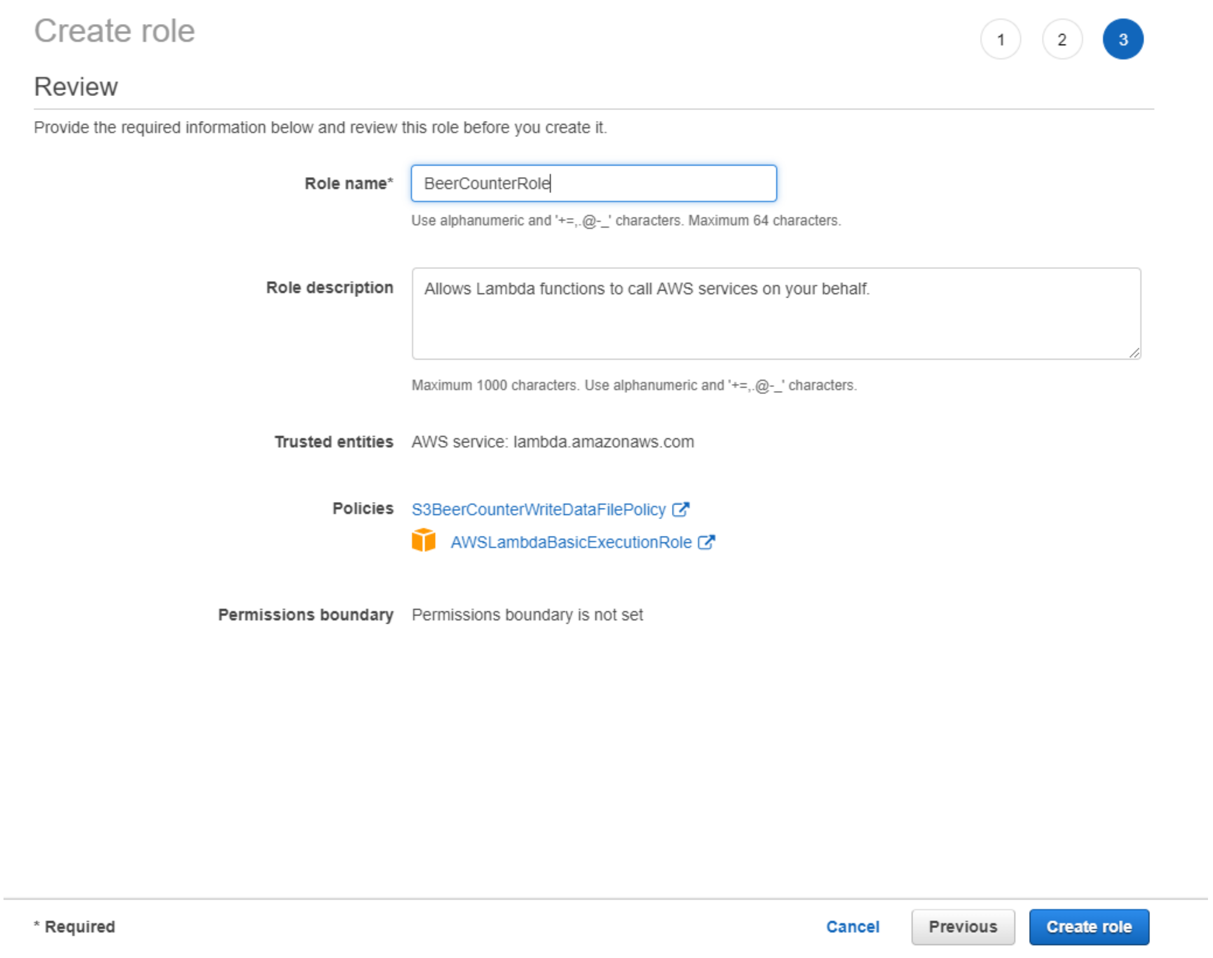
Maximum 1000 characters. Use alphanumeric and '+=,.@-_' characters.

Summary

| Service | Access level | Resource | Request condition |
|---|---|---|---|
| **Allow (1 of 142 services)** Show remaining 141 | | | |
| S3 | **Limited**: Write | ObjectPath \| string like \| All, BucketName \| string like \| milco-beer-website | None |

* Required

For your new role, attach both the new custom policy you just created (for allowing the Lambda function to write to the bucket) and the LambdaBasicExecution role for sending events to CloudWatch:



Create role

1 2 3

Review

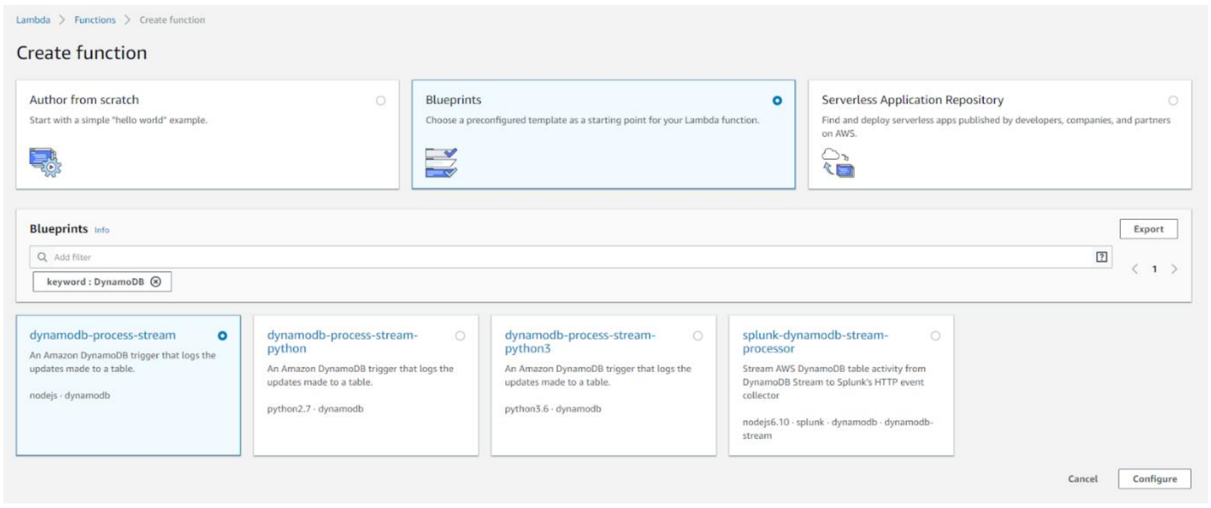
Provide the required information below and review this role before you create it.

| | |
|---|---|
| Role name* | BeerCounterRole |
| | Use alphanumeric and '+=,.@-_' characters. Maximum 64 characters. |
| Role description | Allows Lambda functions to call AWS services on your behalf. |
| | Maximum 1000 characters. Use alphanumeric and '+=,.@-_' characters. |
| Trusted entities | AWS service: lambda.amazonaws.com |
| Policies | S3BeerCounterWriteDataFilePolicy<br>AWSLambdaBasicExecutionRole |
| Permissions boundary | Permissions boundary is not set |

* Required

## Create the lambda implementation

Create a new nodejs lambda function based on the dynamodb process stream  blueprint:



Lambda > Functions > Create function

Create function

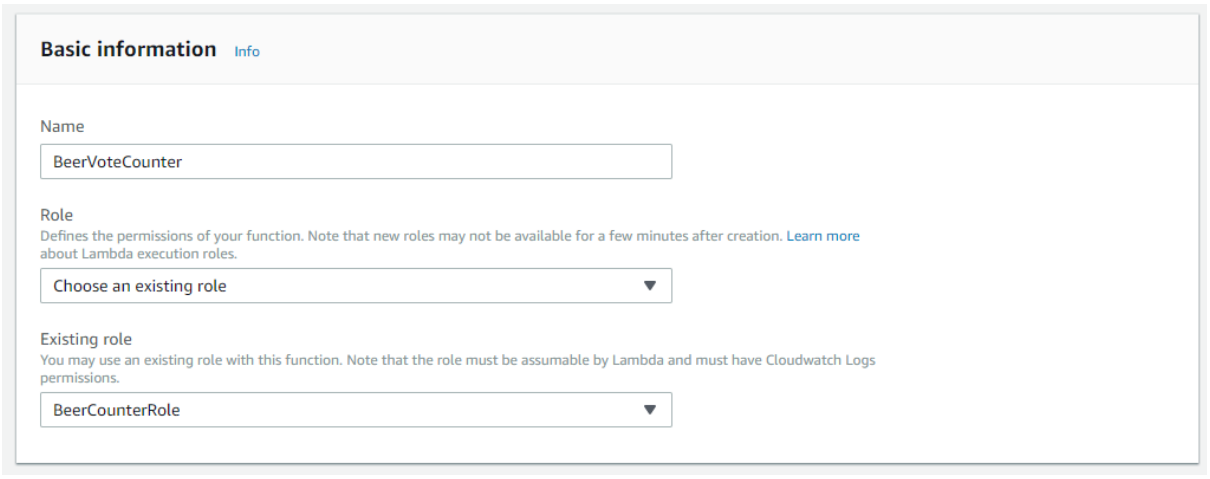
- Author from scratch — Start with a simple "hello world" example.
- [✓ Blueprints] — Choose a preconfigured template as a starting point for your Lambda function.
- Serverless Application Repository — Find and deploy serverless apps published by developers, companies, and partners on AWS.

Blueprints Info

Add filter

keyword : DynamoDB

- [✓ dynamodb-process-stream] — An Amazon DynamoDB trigger that logs the updates made to a table. — nodejs · dynamodb
- dynamodb-process-stream-python — An Amazon DynamoDB trigger that logs the updates made to a table. — python2.7 · dynamodb
- dynamodb-process-stream-python3 — An Amazon DynamoDB trigger that logs the updates made to a table. — python3.6 · dynamodb
- splunk-dynamodb-stream-processor — Stream AWS DynamoDB table activity from DynamoDB Stream to Splunk's HTTP event collector — nodejs6.10 · splunk · dynamodb · dynamodb-stream



**Basic information** Info

Name

BeerVoteCounter

Role

Defines the permissions of your function. Note that new roles may not be available for a few minutes after creation. Learn more about Lambda execution roles.

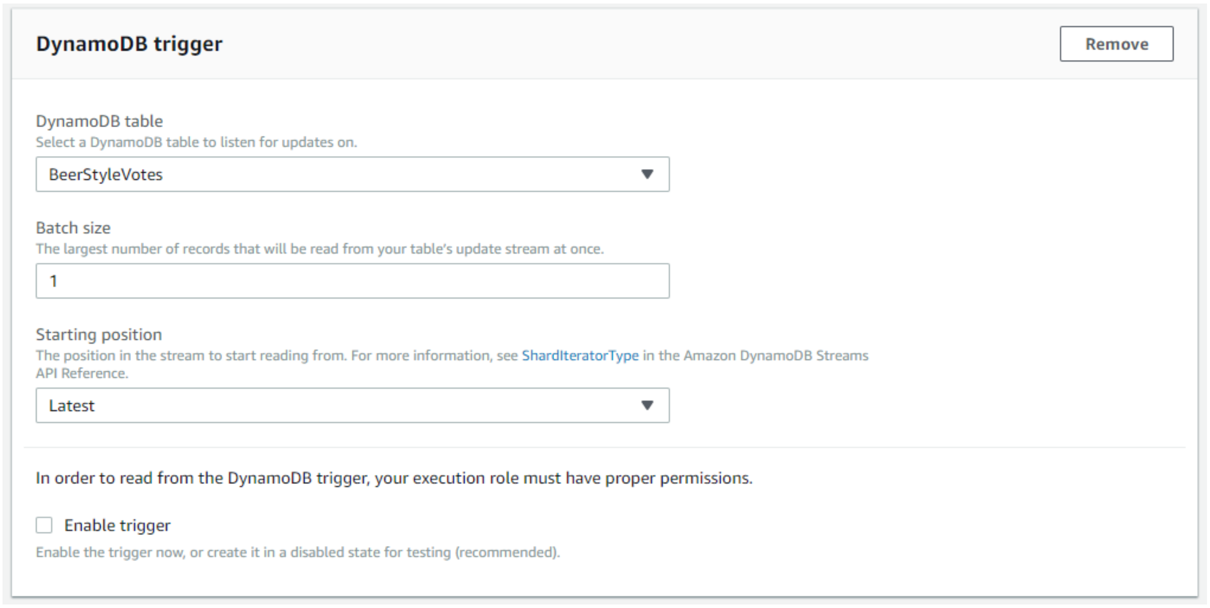
Choose an existing role ▼

Existing role

You may use an existing role with this function. Note that the role must be assumable by Lambda and must have Cloudwatch Logs permissions.

BeerCounterRole ▼

Updates to the BeerStyleVotes trigger the lambda function:



**DynamoDB trigger**

DynamoDB table

Select a DynamoDB table to listen for updates on.

BeerStyleVotes ▼

Batch size

The largest number of records that will be read from your table's update stream at once.

1

Starting position

The position in the stream to start reading from. For more information, see ShardIteratorType in the Amazon DynamoDB Streams API Reference.

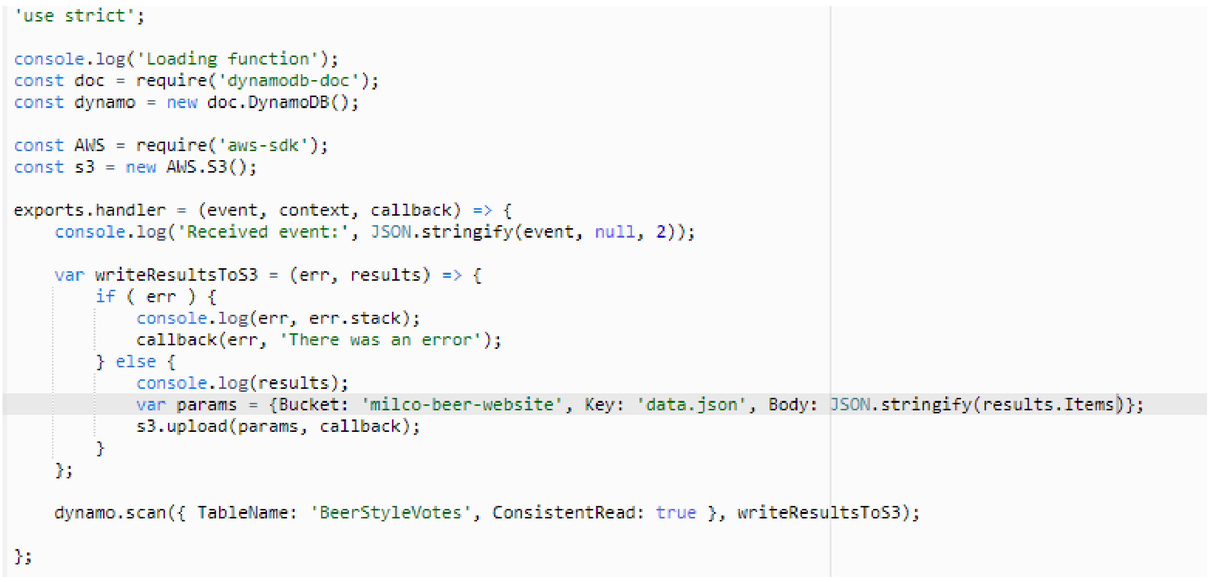
Latest ▼

In order to read from the DynamoDB trigger, your execution role must have proper permissions.

☐ Enable trigger

Enable the trigger now, or create it in a disabled state for testing (recommended).

Create the function and modify the implementation; be sure to refer to your Bucket in the write call and to your table name in the table scan operation:

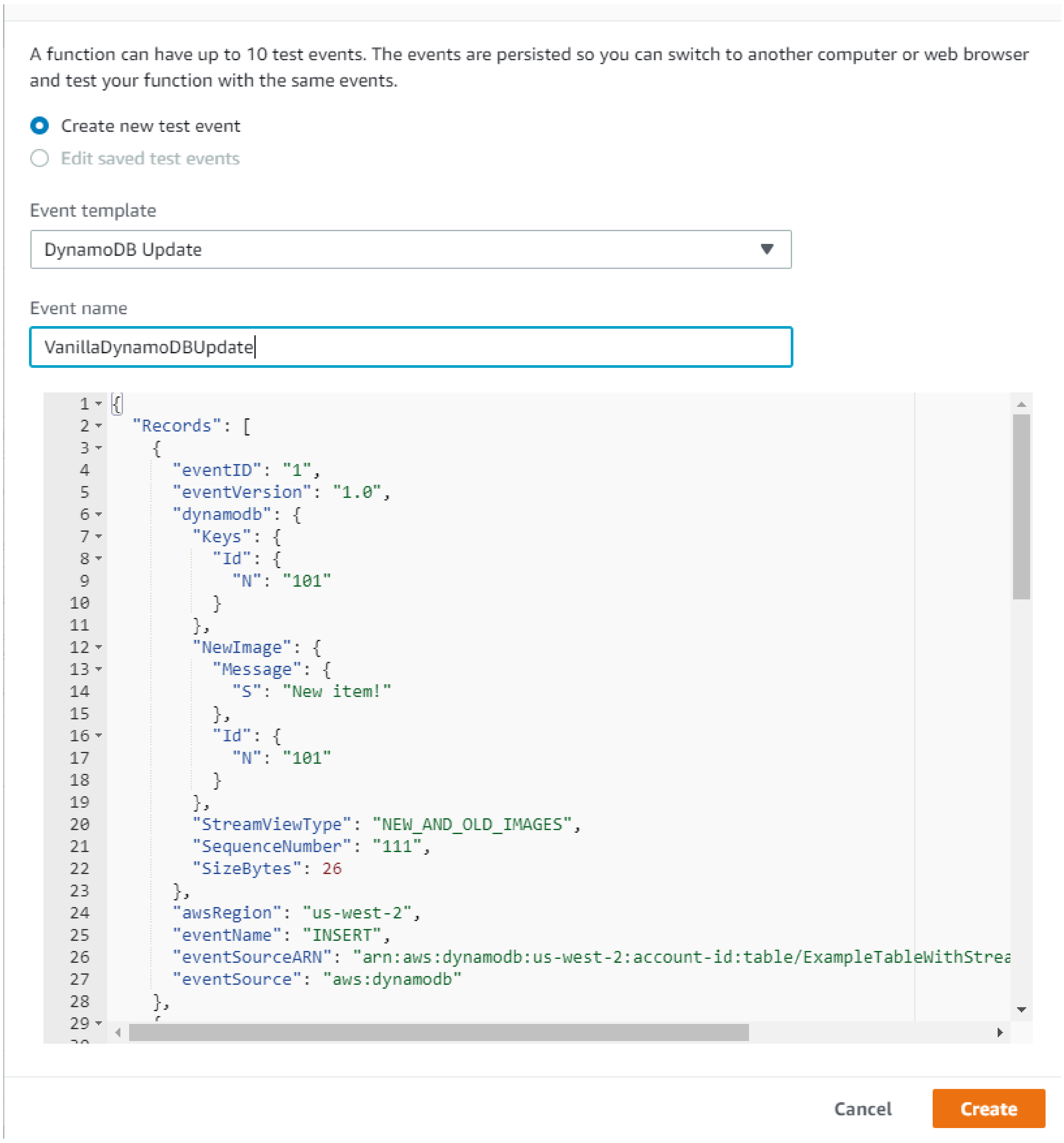


```
'use strict';

console.log('Loading function');
const doc = require('dynamodb-doc');
const dynamo = new doc.DynamoDB();

const AWS = require('aws-sdk');
const s3 = new AWS.S3();

exports.handler = (event, context, callback) => {
    console.log('Received event:', JSON.stringify(event, null, 2));

    var writeResultsToS3 = (err, results) => {
        if ( err ) {
            console.log(err, err.stack);
            callback(err, 'There was an error');
        } else {
            console.log(results);
            var params = {Bucket: 'milco-beer-website', Key: 'data.json', Body: JSON.stringify(results.Items)};
            s3.upload(params, callback);
        }
    };

    dynamo.scan({ TableName: 'BeerStyleVotes', ConsistentRead: true }, writeResultsToS3);

};
```

```
'use strict';

console.log('Loading function');
const doc = require('dynamodb-doc');
const dynamo = new doc.DynamoDB();

const AWS = require('aws-sdk');
const s3 = new AWS.S3();

exports.handler = (event, context, callback) => {
    console.log('Received event:', JSON.stringify(event, null, 2));

    var writeResultsToS3 = (err, results) => {
        if ( err ) {
            console.log(err, err.stack);
            callback(err, 'There was an error');
        } else {
            console.log(results);
            var params = {Bucket: 'milco-beer-website', Key: 'data.json',
Body: JSON.stringify(results.Items)};
            s3.upload(params, callback);
        }
    };


    dynamo.scan({ TableName: 'BeerStyleVotes', ConsistentRead: true },
writeResultsToS3);



};
```

## Testing, testing

Create a new test event for a DynamoDB update (payload does not exactly matter, since we do not use any elements from it … we're triggered by the mere fact the table as been updated)



A function can have up to 10 test events. The events are persisted so you can switch to another computer or web browser and test your function with the same events.

[✓ Create new test event]

Edit saved test events

Event template

DynamoDB Update

Event name

VanillaDynamoDBUpdate

```
{
  "Records": [
    {
      "eventID": "1",
      "eventVersion": "1.0",
      "dynamodb": {
        "Keys": {
          "Id": {
            "N": "101"
          }
        },
        "NewImage": {
          "Message": {
            "S": "New item!"
          },
          "Id": {
            "N": "101"
          }
        },
        "StreamViewType": "NEW_AND_OLD_IMAGES",
        "SequenceNumber": "111",
        "SizeBytes": 26
      },
      "awsRegion": "us-west-2",
      "eventName": "INSERT",
      "eventSourceARN": "arn:aws:dynamodb:us-west-2:account-id:table/ExampleTableWithStrea
      "eventSource": "aws:dynamodb"
    },
```

Test the event agains your lambda function and watch it fail. Can you figure out what goes wrong and how to solve it?

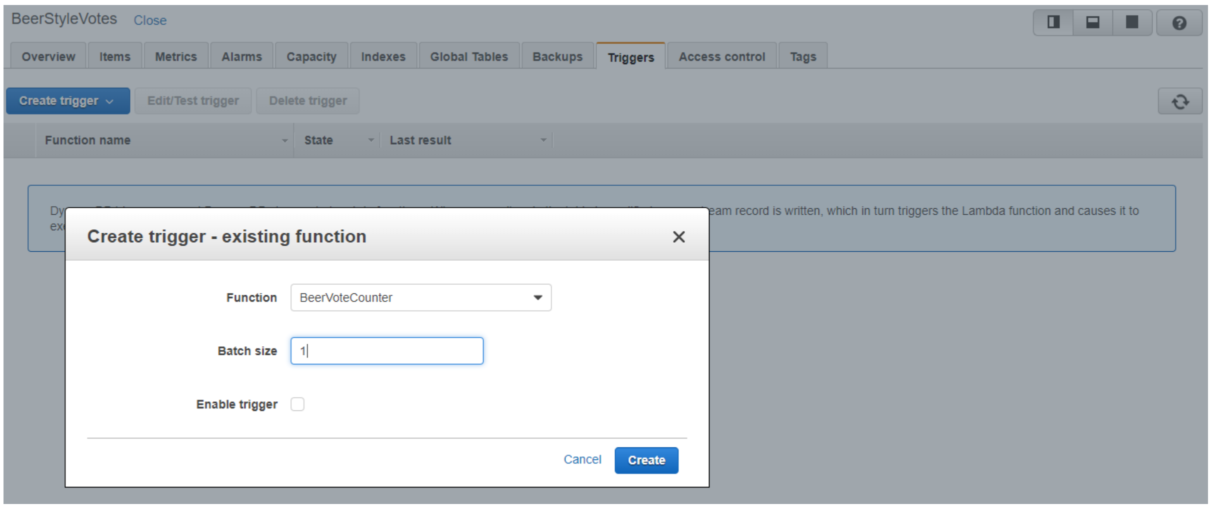
Are you peeking the solution??

**Reason: we have not yet authorized the Lambda function to SCAN the table we use for our data !**

Solution: *Add an additional policy to your role that allows Scans on your table for the DynamoDB server.*

## Trigger-Happy

Let’s hook up the DynamoDB table to the lambda function, from the DynamoDB console:



Dy… ex… eam record is written, which in turn triggers the Lambda function and causes it to

Create trigger - existing function

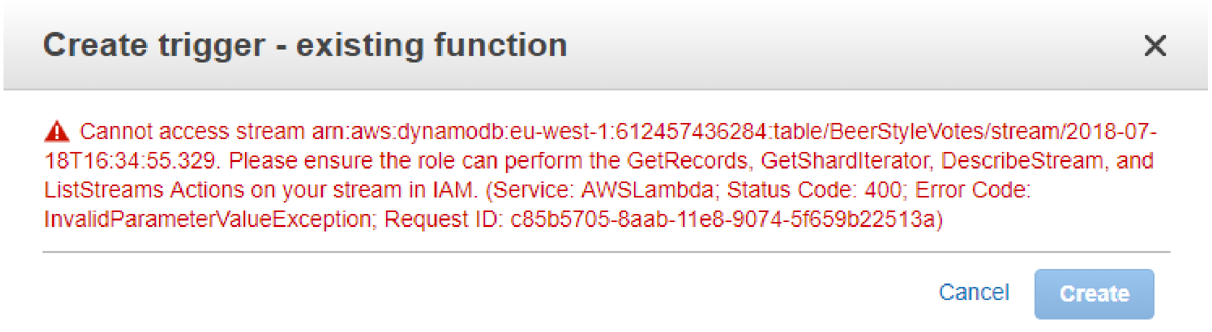
Function: BeerVoteCounter

Batch size: 1

Enable trigger: ☐

Cancel Create

Again, we have not granted sufficient access permissions …

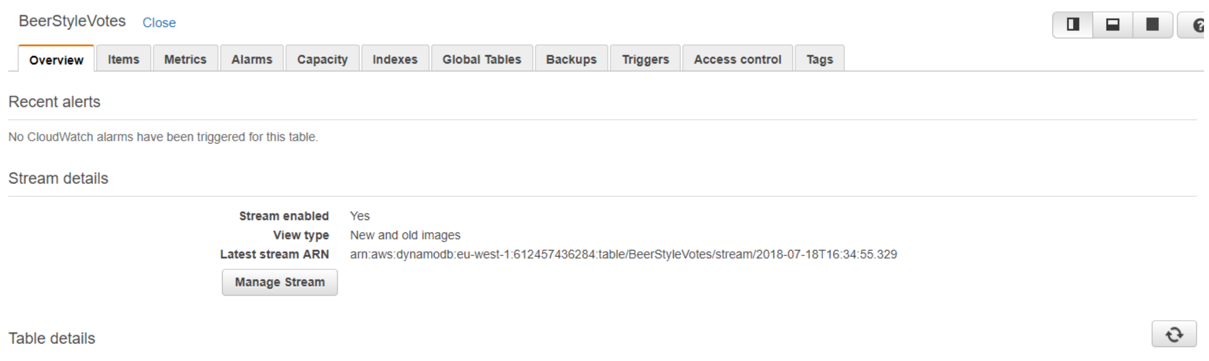


Create trigger - existing function

Cannot access stream arn:aws:dynamodb:eu-west-1:612457436284:table/BeerStyleVotes/stream/2018-07-18T16:34:55.329. Please ensure the role can perform the GetRecords, GetShardIterator, DescribeStream, and ListStreams Actions on your stream in IAM. (Service: AWSLambda; Status Code: 400; Error Code: InvalidParameterValueException; Request ID: c85b5705-8aab-11e8-9074-5f659b22513a)

Cancel Create

Go back to the overview page and copy your stream's ARN:


BeerStyleVotes Close

Recent alerts

No CloudWatch alarms have been triggered for this table.

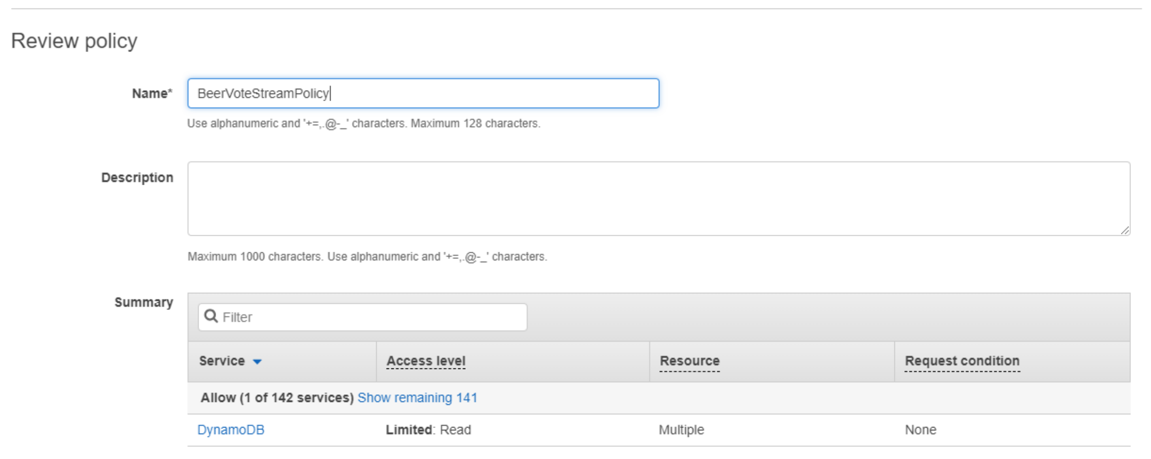
Stream details

| | |
|---|---|
| Stream enabled | Yes |
| View type | New and old images |
| Latest stream ARN | arn:aws:dynamodb:eu-west-1:612457436284:table/BeerStyleVotes/stream/2018-07-18T16:34:55.329 |

Table details

My ARN is arn:aws:dynamodb:eu-west-1:612457436284:table/BeerStyleVotes/stream/2018-07-18T16:34:55.329

Add the required access to a new policy and attach the policy to your role:


Review policy

Name* BeerVoteStreamPolicy

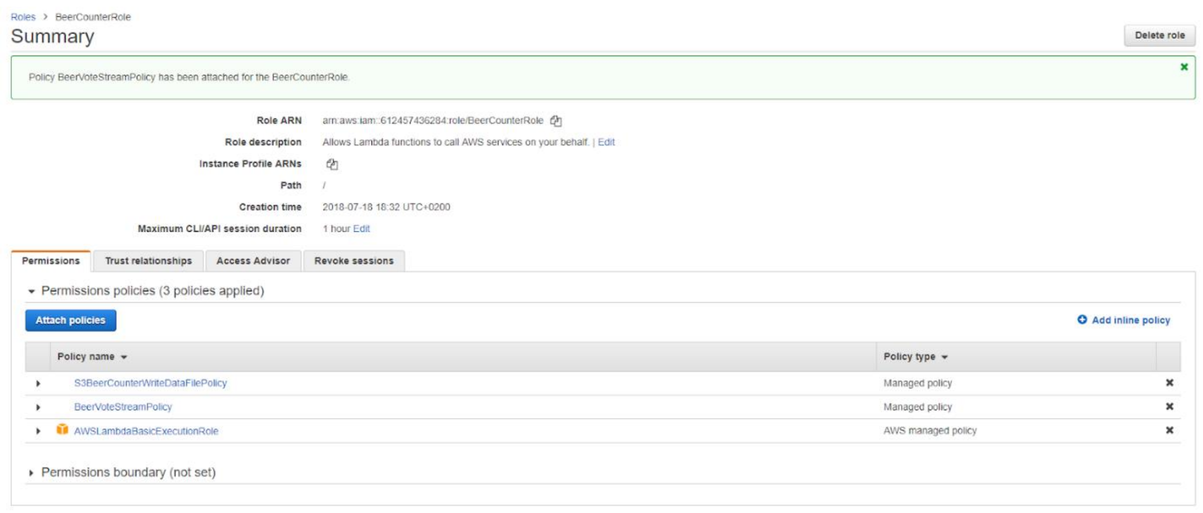
Use alphanumeric and '+=,.@-_' characters. Maximum 128 characters.

Description

Maximum 1000 characters. Use alphanumeric and '+=,.@-_' characters.

Summary

| Service | Access level | Resource | Request condition |
|---|---|---|---|
| Allow (1 of 142 services) Show remaining 141 | | | |
| DynamoDB | Limited: Read | Multiple | None |


Roles > BeerCounterRole

Summary

Policy BeerVoteStreamPolicy has been attached for the BeerCounterRole.

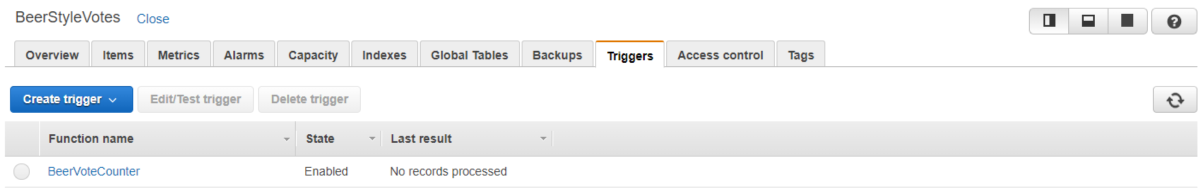
| | |
|---|---|
| Role ARN | arn:aws:iam::612457436284:role/BeerCounterRole |
| Role description | Allows Lambda functions to call AWS services on your behalf. \| Edit |
| Instance Profile ARNs | |
| Path | / |
| Creation time | 2018-07-18 18:32 UTC+0200 |
| Maximum CLI/API session duration | 1 hour Edit |

Permissions policies (3 policies applied)

| Policy name | Policy type |
|---|---|
| S3BeerCounterWriteDataFilePolicy | Managed policy |
| BeerVoteStreamPolicy | Managed policy |
| AWSLambdaBasicExecutionRole | AWS managed policy |

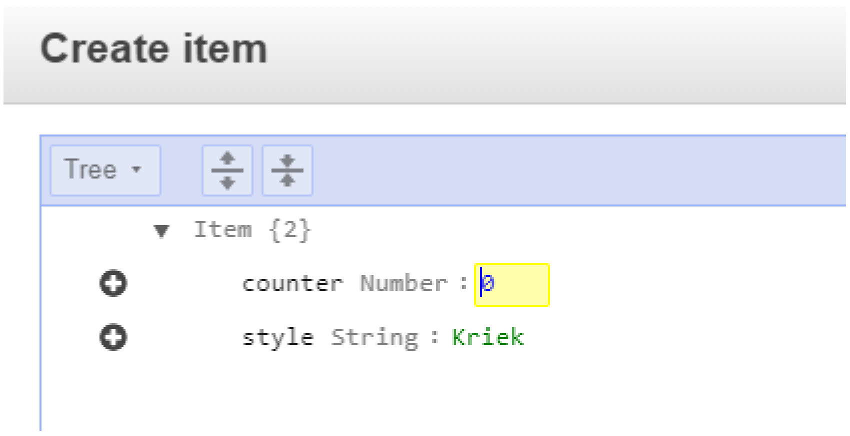
Permissions boundary (not set)

Again create the trigger, now no errors occur:


BeerStyleVotes Close
Overview Items Metrics Alarms Capacity Indexes Global Tables Backups Triggers Access control Tags
Create trigger
Edit/Test trigger
Delete trigger

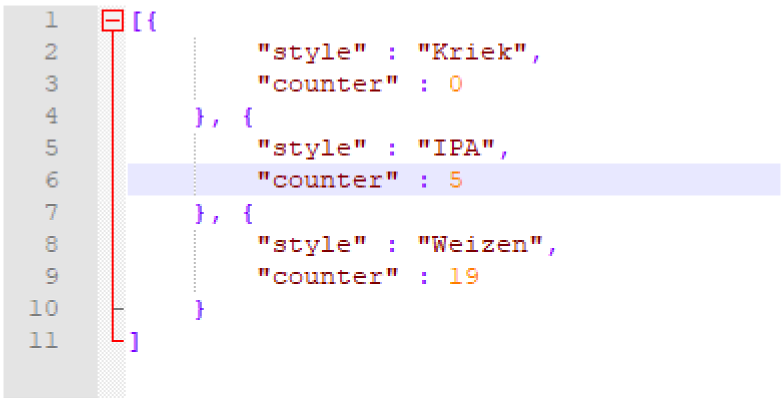
| Function name | State | Last result |
|---|---|---|
| BeerVoteCounter | Enabled | No records processed |

Create a new item in the table:



```
▼ Item {2}
    counter Number : 0
    style String : Kriek
```

Save the new record and verify that the data.json file in your bucket is now updated. The file should have a structure like this:

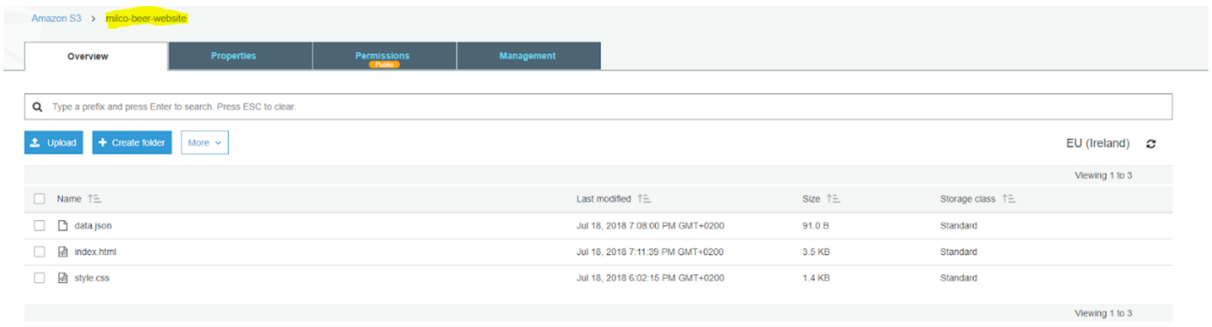


```
[{
        "style" : "Kriek",
        "counter" : 0
    }, {
        "style" : "IPA",
        "counter" : 5
    }, {
        "style" : "Weizen",
        "counter" : 19
    }
]
```

Now it is time to put all together: upload the index.html and the style.css files to your bucket.

## Open the webapp in browser

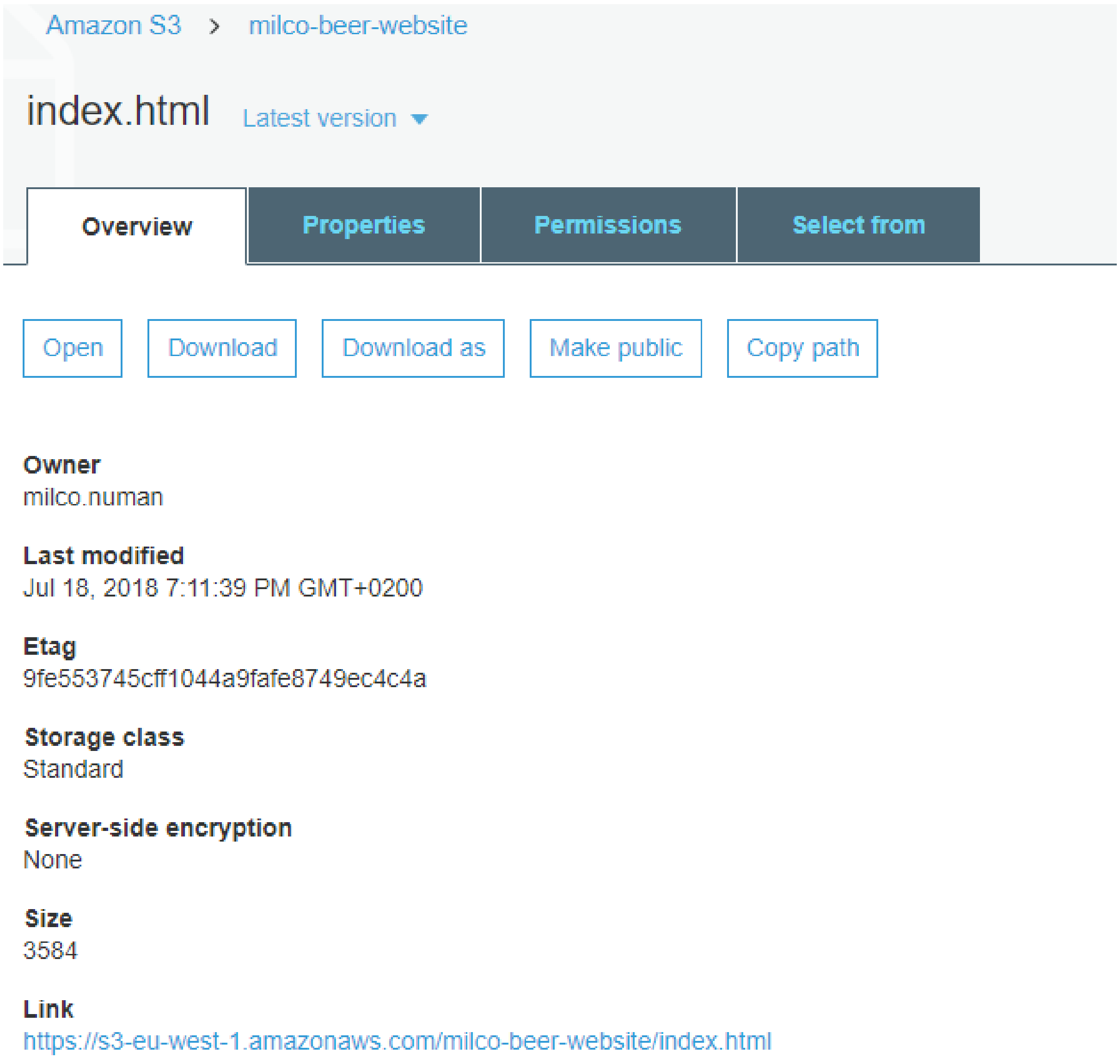
Open the S3 bucket and open the index.html file in your browser from the bucket:



| Name | Last modified | Size | Storage class |
|---|---|---|---|
| data.json | Jul 18, 2018 7:08:00 PM GMT+0200 | 91.0 B | Standard |
| index.html | Jul 18, 2018 7:11:39 PM GMT+0200 | 3.5 KB | Standard |
| style.css | Jul 18, 2018 6:02:15 PM GMT+0200 | 1.4 KB | Standard |

Viewing 1 to 3

On the index.html object you will see the link; open this from the S3 console:



**Owner**
milco.numan

**Last modified**
Jul 18, 2018 7:11:39 PM GMT+0200

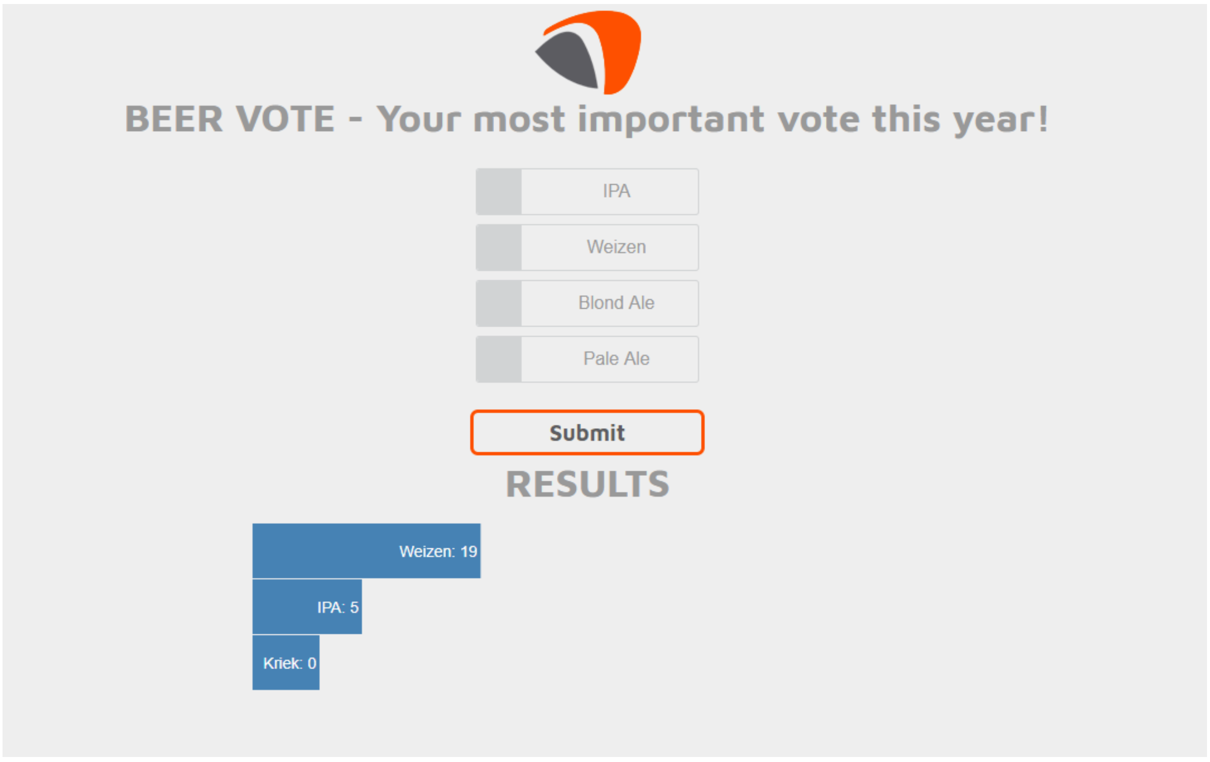
**Etag**
9fe553745cff1044a9fafe8749ec4c4a

**Storage class**
Standard

**Server-side encryption**
None

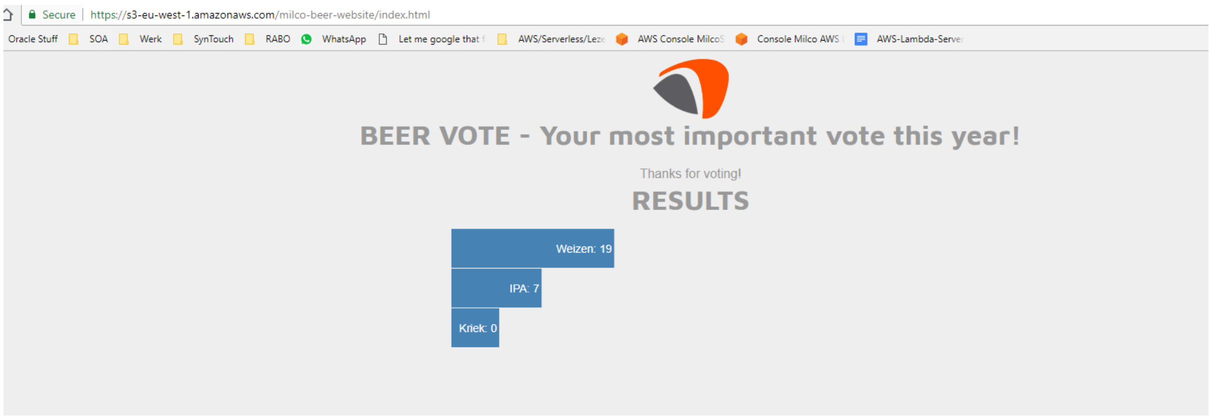
**Size**
3584

**Link**
https://s3-eu-west-1.amazonaws.com/milco-beer-website/index.html


BEER VOTE - Your most important vote this year!

RESULTS

Vote and wait for the refresh:


Secure | https://s3-eu-west-1.amazonaws.com/milco-beer-website/index.html

BEER VOTE - Your most important vote this year!

Thanks for voting!

RESULTS

## Cleanup

To avoid running into costs, remove the object your created - specifically:

DynamoDB table from the DynamoDB console

S3 bucket from the S3 console

Lambda functions

API gateway entries

(optionally) remove the log groups for the above from the CloudWatch console

This tutorial was based on [https://medium.com/head-in-the-clouds/how-to-create-a-serverless-website-with-aws-lambda-95bb5abfdbff](https://medium.com/head-in-the-clouds/how-to-create-a-serverless-website-with-aws-lambda-95bb5abfdbff)