Getting to know the Amazing Amazon Web Services

Luigi Libero Lucio Starace
luigi.starace@gmail.com

June 4, 2018

University of Naples, Federico II

Outline

1 A little bit of context

2 An AWS bestiarium

3 Serverless architectures

4 Demo: a serverless web application

5 Take Home Messages

A little bit of context

Cloud Computing

> Cloud computing is the **on-demand delivery** of computing resources through a cloud services platform via the internet with **pay-as-you-go** pricing.

Cloud Computing: Service Models

- **Software as a Service (SaaS)**
  The service vendor provides the user with a completed product that is run and managed by the service provider.
- **Platform as a Service (PaaS)**
  The service vendor provides the user with a set of API which can be used to build, test and deploy applications.
- **Infrastructure as a Service (IaaS)**
  The service vendor provides users access to computing resources such as servers, storage and networking.

Service Models: a visual comparison

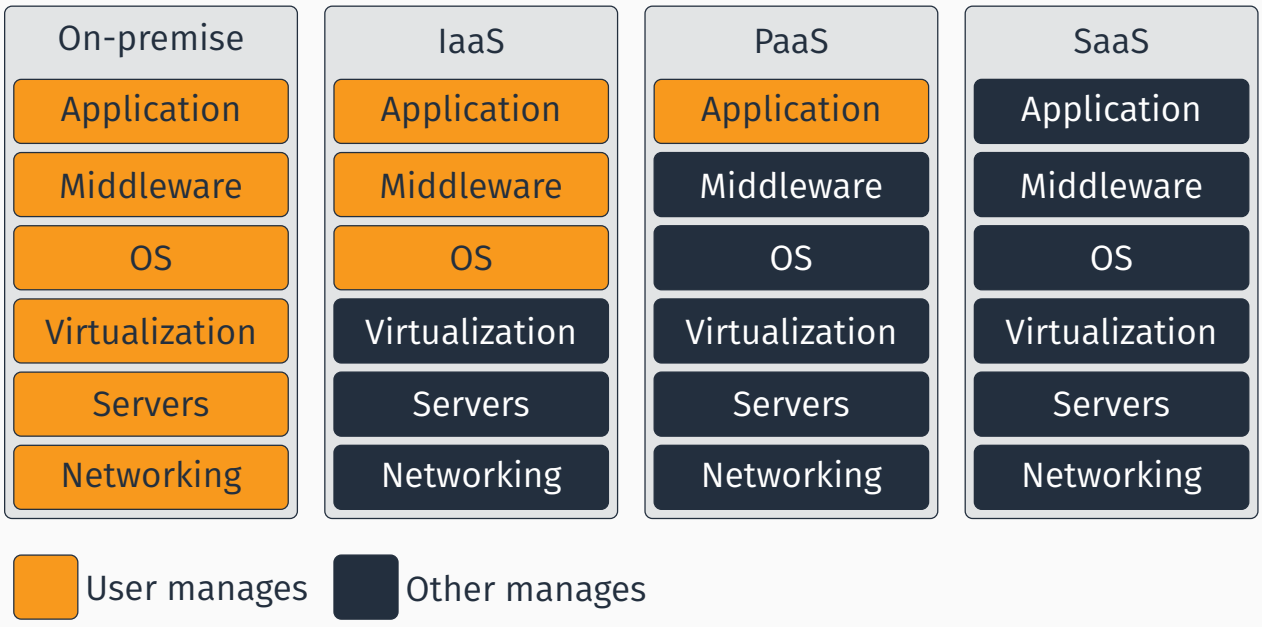
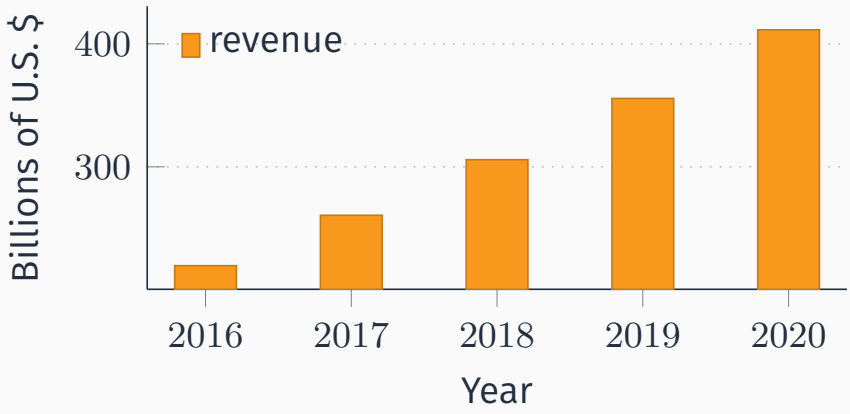


Figure 1: A service models comparison

Some stats

Worldwide Public Cloud Services Revenue Forecast (Billions of U.S. Dollars) [Gar17]

2016	2017	2018	2019	2020
219,6	260,6	305,8	355,6	411,4



The bigwigs

- Google
- IBM
- Microsoft
- Alibaba
- Oracle
- Amazon

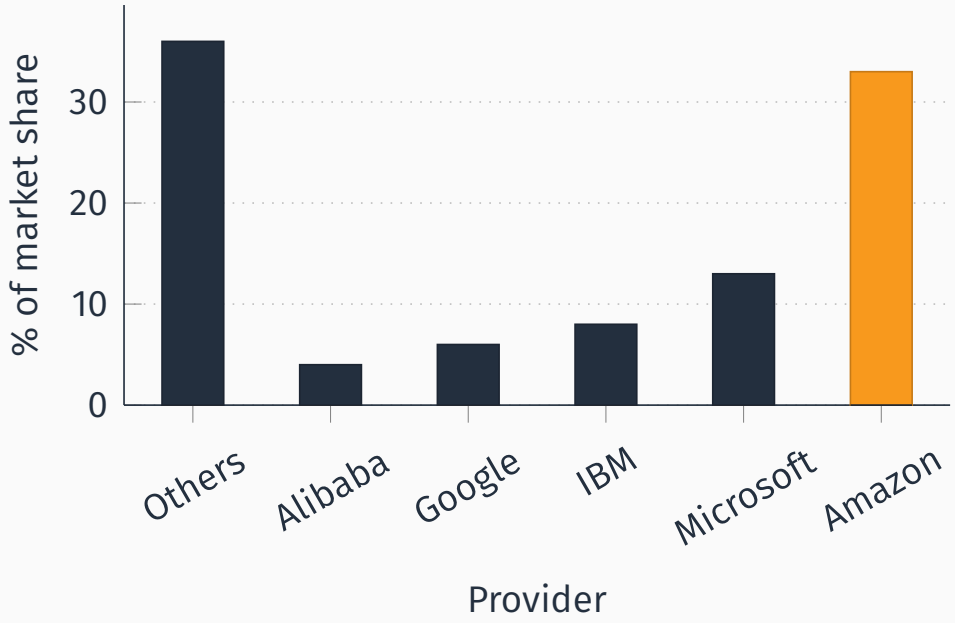


Figure 2: Market share in Q4 2017 (IaaS, PaaS, Hosted Private Cloud) [Syn18]

An AWS bestiarium

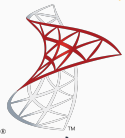
Amazon Web Services is a collection of cloud-based services.
A very big one.

An AWS bestiarium

Database services

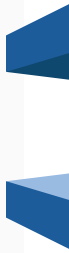
Relational Database Service (RDS)

- Set up, operate a relational database in the cloud.
- Takes care of backups, patching.
- Supports:
  - MySQL, PostgreSQL, MariaDB
  - Oracle, MS SQL Server
  - Amazon Aurora



Non Relational Database Services

- DynamoDB
  - *Fast and flexible NoSQL database service for any scale.*
- ElastiCache
  - In memory data store.
  - Supports memcached, Redis
- Neptune
  - Graph database service
  - Supports RDF, SPARQL, ...

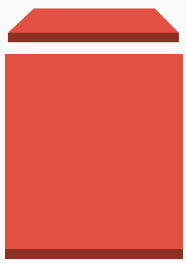


An AWS bestiarium

Cloud storage

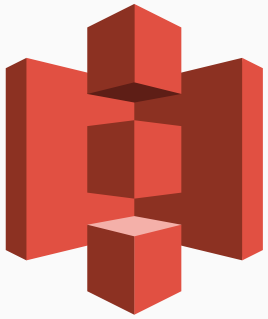
Cloud storage products

- Elastic Block Storage (EBS)
  - Persistent local storage for EC2 instances.
- Elastic File System (EFS)
  - File system interface to share data between EC2 instances.
- Simple Storage Service (S3)
- Glacier
  - Durable and cheap long-term storage.



Amazon Simple Storage Service (S3)

- *store and retrieve any amount of data from anywhere*
- 99.999999999% durability (nine nines!)
- Data is distributed across a *minimum* of three availability zones
- A logical unit of storage is a *bucket*
- Multiple storage classes
  - Standard
  - Infrequent Access
  - One zone-Infrequent Access
  - Amazon Glacier



Amazon Simple Storage Service (S3) - more

Multiple storage classes

Storage class	Storage	Retrieval (per 1K req.)
Standard	$0.022 per GB	$0.0004
Infrequent access	$0.0125 per GB	$0.001
IA single zone	$0.01 per GB	$0.001

Table 1: S3 pricing (Ireland)

- Well-integrated with other services
  - Machine Learning
  - Big Data Analysis
- REST API
- Can be used to host static websites

An AWS bestiarium

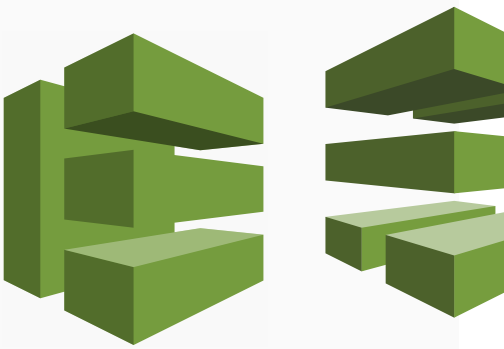
Developer Tools

Developer Tools

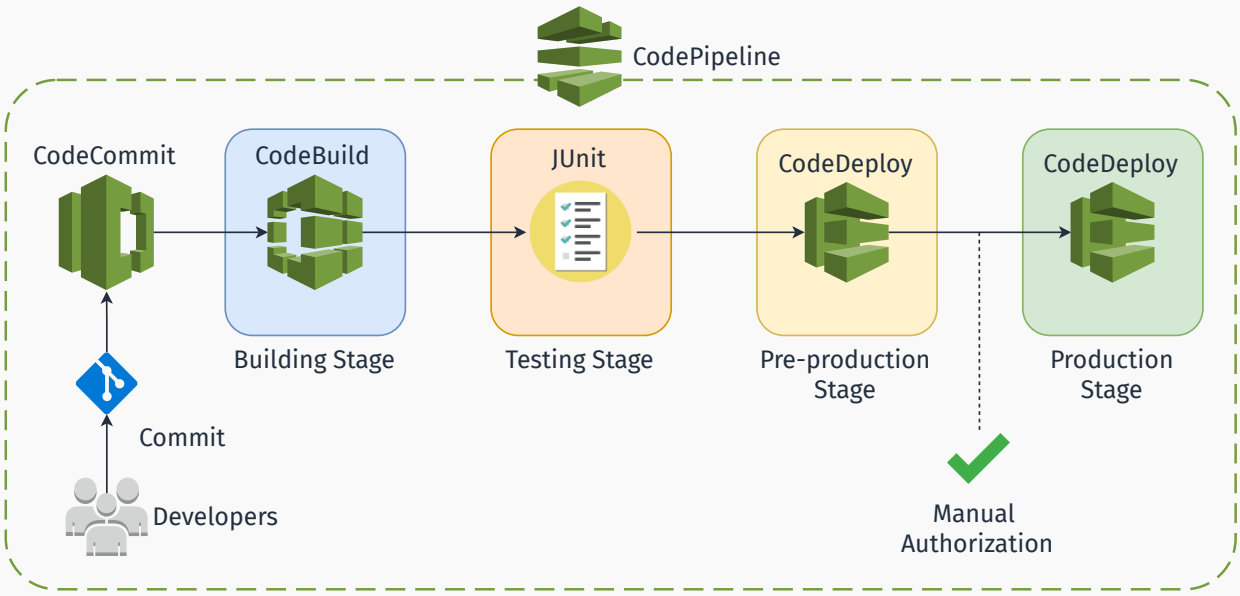
- CodeCommit
  - Managed, scalable, private `git` server
  - Pricing based on active users (5 free, 1$ for each additional user)
- CodeBuild
  - Managed, scalable build server
  - Pay-per-minute spent building your code

Developer Tools - more

- CodeDeploy
  - Automates deployment to computing services (also to instances running on-premise)
  - Tries to avoid downtime
  - 0.02$ per-on-premise deployment
- CodePipeline
  - Continuous integration e continuous delivery
  - Define your own workflow and stages
  - 1$ per-month per active pipeline



CodePipeline

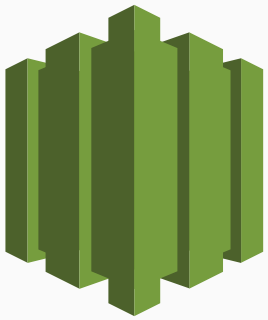


CodePipeline - more

Interested in CI/CD on AWS? Check these out:

- *Practicing Continuous Integration and Continuous Delivery on AWS* (whitepaper) [Ama17]
- *Set up a Continuous Deployment Pipeline using AWS CodePipeline* [Amab]
- *Tutorial: Create a Four-Stage Pipeline* [Amac]

- CodeStar
  - Wrapper around developer tools to simplify setup
  - Templates
  - Team Management
  - Central Project Dashboard
  - Free of charge
- Cloud9
  - Cloud-based full-fledged IDE
  - Runs in a web browser
  - Collaborative editing and chat
  - Greatly-integrated with AWS
  - Free of charge

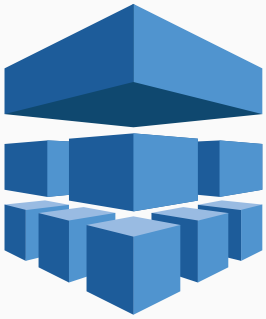


An AWS bestiarium

Machine Learning

Machine Learning: platform services

- Amazon SageMaker
  - Preconfigured for Tensorflow, MXNet...
  - Build, Train and Deploy phases
  - Pay based on build time, train time and hosting time



Machine Learning: application services

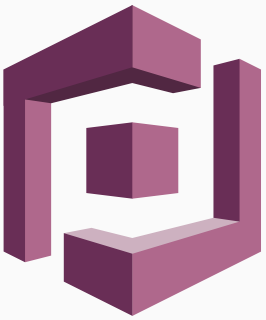
- Comprehend (for NLP) Dashboard
- Rekognition (Visual Analysis) Dashboard
- Translate
- Polly (text-to-speech)
- Transcribe (speech-to-text)

An AWS bestiarium

Miscellanea

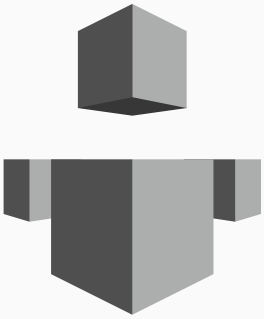
Miscellanea

- Cognito
  - Sign-up and authentication
  - Federated identities
- CloudFront
  - Content Delivery Network
  - 116 Points of Presence in 56 cities across 24 countries
- Mechanical Turk



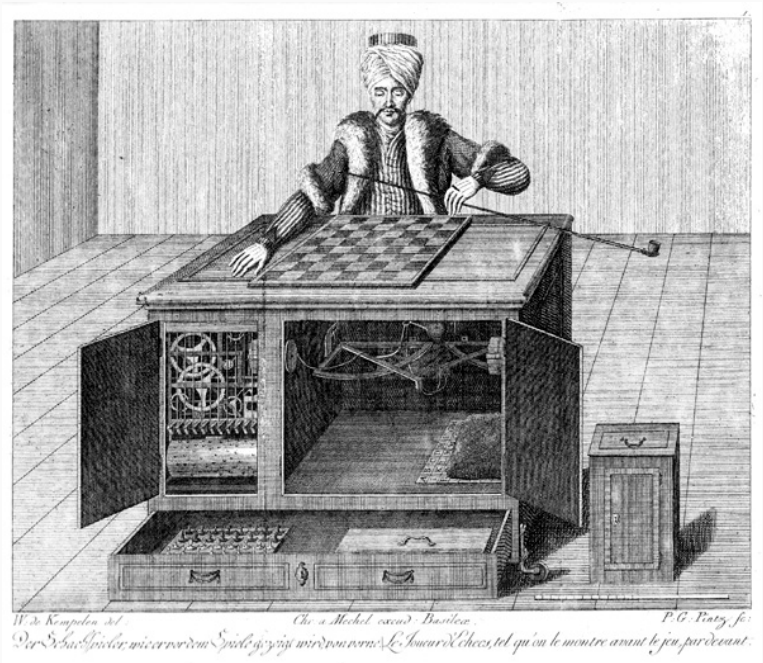
Amazon MechanicalTurk

- ???



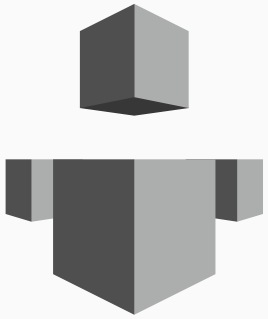
The Turk

The Turk was a chess-playing automaton built in 1770. Obviously it was a fraud.



- ???

- Human Intelligence through an API
- Create HIT (Human Intelligence Task)
- Elastic, on-demand workforce
- Available 24/7



An AWS bestiarium

Computing

Amazon Elastic Compute Cloud (EC2)

- (Virtual) Servers on demand
- Different types of instances to suit computing needs
- Per-second (or per-hour) billing
- Data transfer **not** included!
- Persistent storage **not** included!
  - ⏮ EBS/EFS
- Scaling **not** included!

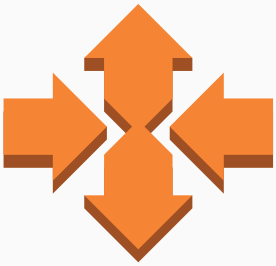
Azure: Virtual Machines web
Google Cloud: Compute Engine web

Amazon Elastic Compute Cloud (EC2) Notes

- Need more space? Amazon Elastic Block Storage provides persistent block storage volumes for use with Amazon EC2 instances in the AWS Cloud

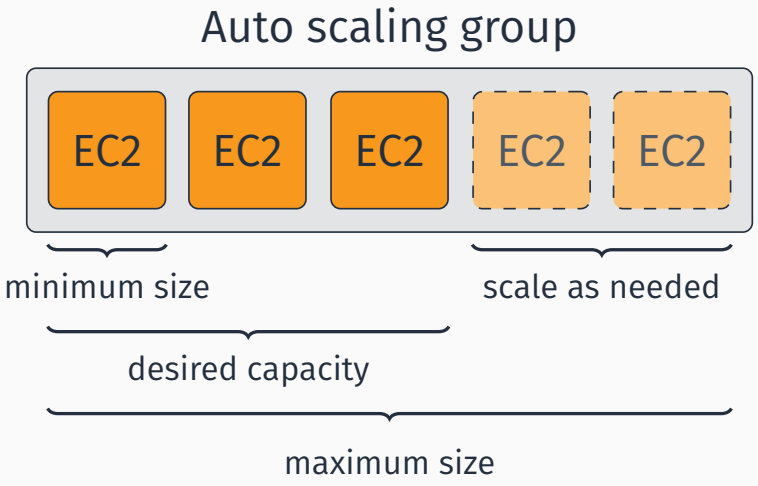
Amazon EC2 Auto Scaling

- *Scaling is the ability to increase or decrease the compute capacity of your application*
- Scale your application manually, on a scheduled basis or on demand



Azure: Virutal Machine Scale Sets web
Google Cloud: Load Balancing web

Amazon EC2 Auto Scaling: details

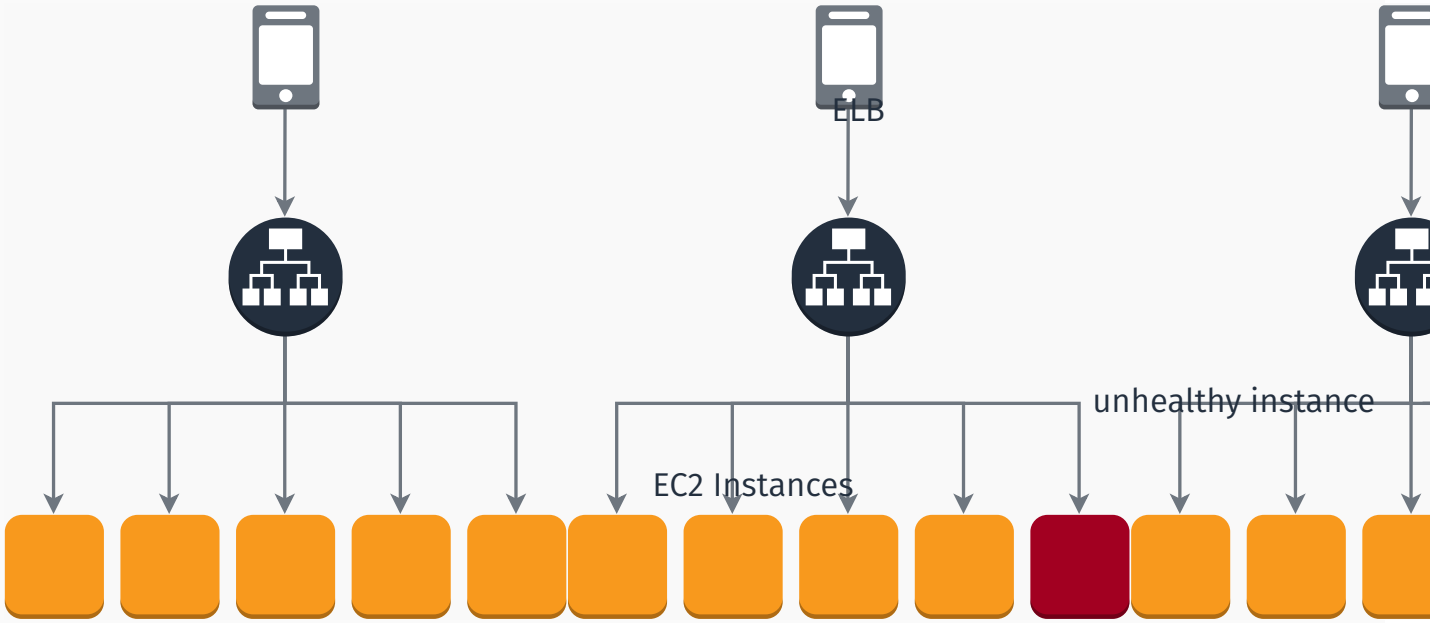


Amazon Elastic Load Balancing (ELB)

- Distributes incoming traffic across multiple EC2 instances
- Pay-per-use billing
  - Execution time
  - Number of requests / traffic

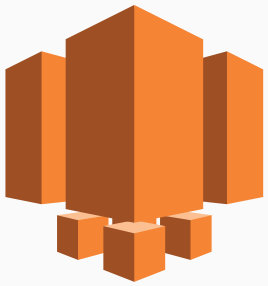
Azure: Load Balancer web

Amazon Elastic Load Balancing (ELB)



Amazon Lightsail

- A lightweight, simplified offer
- Bundles computing, storage, and networking capacity
- Preconfigured instances for
  - Debian, Windows Server, ...
  - Wordpress, Magento, Redmine, ...
  - LAMP stack, Nginx, ...
- Low and **predictable** monthly costs



Websites: EC2 Lightsail

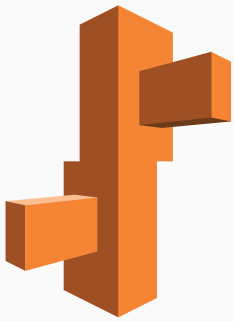
Amazon Lightsail

Notes

- Stress out how it is simplified
- Show the websites: EC2 Lightsail
- Point out how it's not immediate to predict monthly cost for EC2 + Storage (Elastic Block Store) + Autoscaling + ELB

Amazon Elastic Beanstalk

- "*Easy to begin, impossible to outgrow*"
- Easy-to-use service to deploy web apps
- Supports Apache, Nginx, IIS and more
- Supports Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker
- Manages auto-scaling, load balancing, health monitoring
- Customizable
- Free of charge. Pay only for the AWS resources you use.

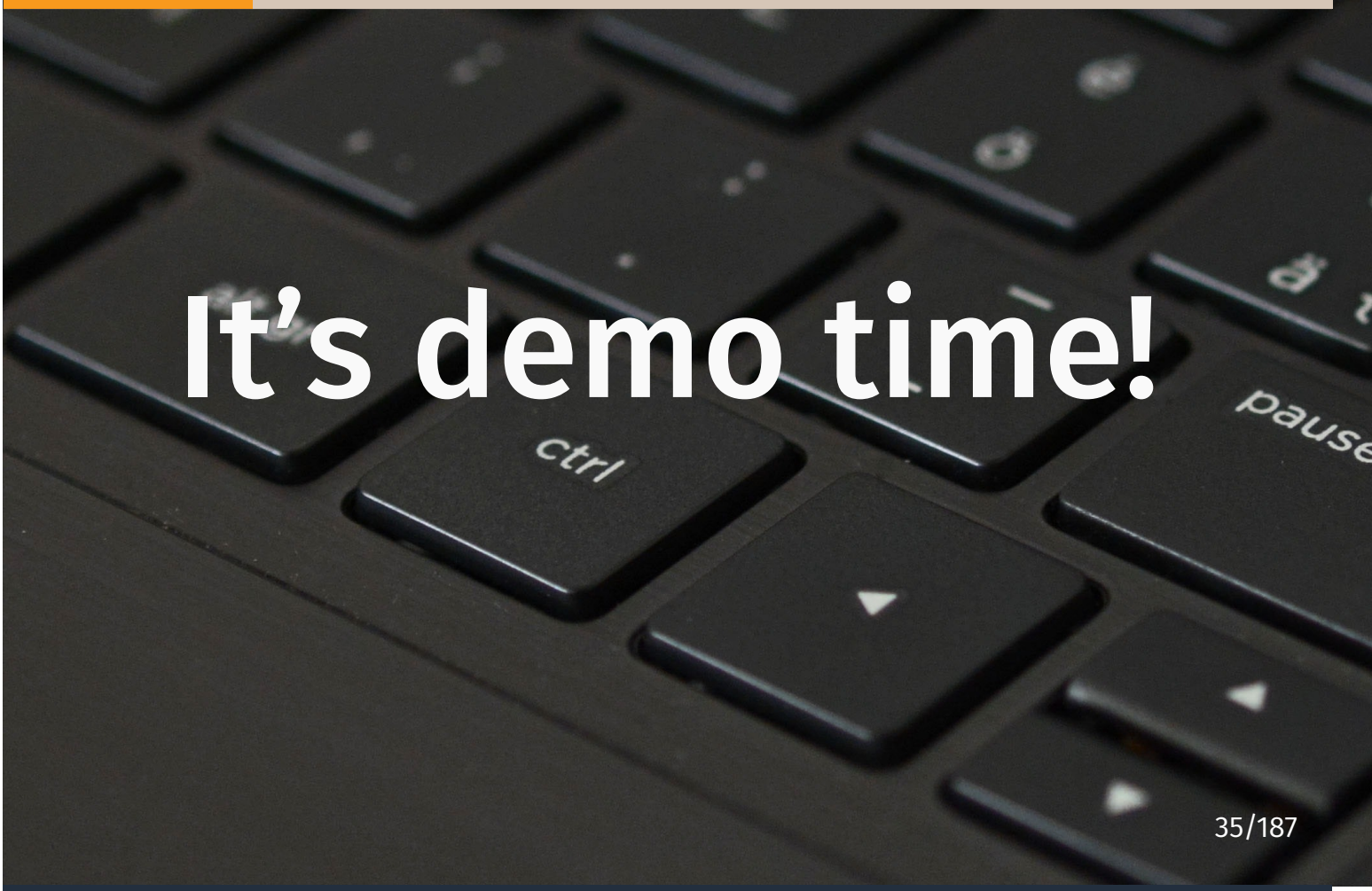


A little recap

So far we've seen:

- Elastic Compute Cloud (EC2)
  - Auto-scaling, Elastic Load Balancing
- Lightsail
- Elastic Beanstalk

We have to (somewhat) care about the infrastructure!



What we’re going to do in this demo

- Checkout a very simple web application written in PHP
- Run it locally (optional)
- Deploy it to the cloud using Amazon Elastic Beanstalk
- Doable in 30 minutes at home.

About the web app

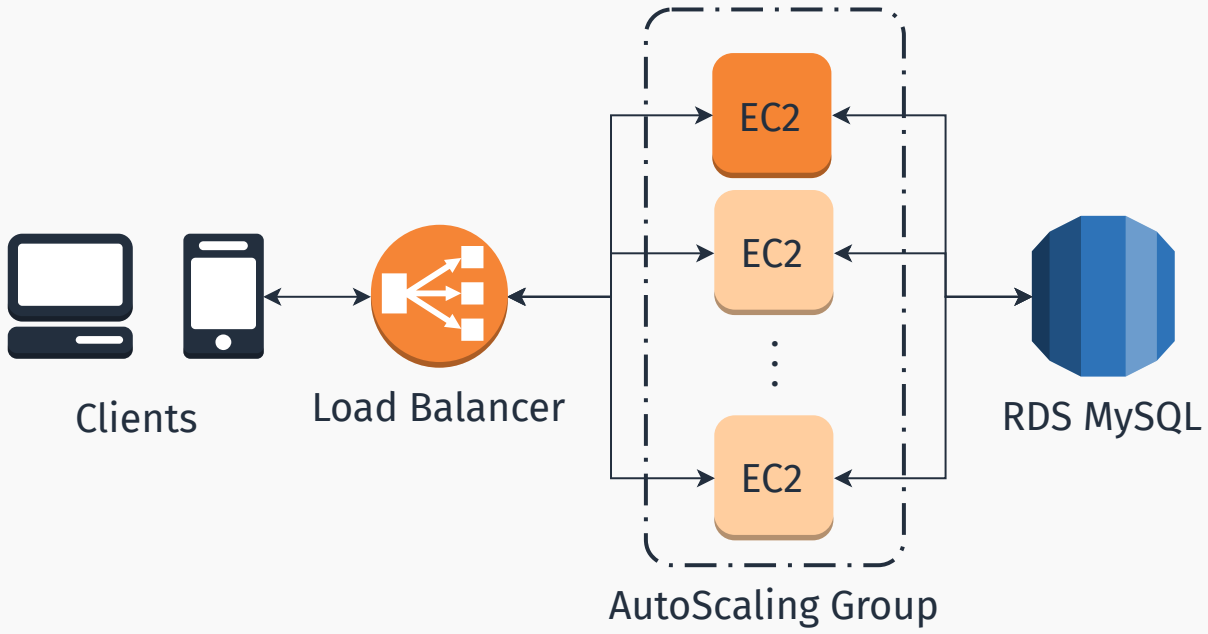
We'll deploy a very simple website for this very talk. The web app has two pages:

- a substantially static **homepage**
- a **comment** page allowing users to leave feedbacks.

Technologies involved:

- Symfony framework
- Doctrine ORM
- Webpack, Sass

Architecture



⏭ Skip tutorial

What you'll need

- An AWS account (a free one will suffice)
- `git` version control (recommended)
- If you want to build and run the app locally:
  - An AMP (Apache, PHP $\geq$ 7.1.3, MySQL $\geq$ 5.7) stack
  - Composer package manager
  - Node.js

Step 1: get the app

Clone the git repository Github

```
D:\Desktop> git clone https://github.com/luistar/
    serverful-webapp.git serverful-webapp
```

Step 2: install dependencies

Install dependencies with composer

```
D:\Desktop> cd serverful-webapp

D:\Desktop\serverful-webapp> composer install
```

Then install Node.js dependecies

```
D:\Desktop\serverful-webapp> npm install
```


Step 3: Configuration parameters

Start your database instance and create an user for the webapp. Once you are done, update the configuration file `config/packages/database-config.php` accordingly.

```
2 //get parameter from environment or fallback to defaults
3 $db_host = (
4     (isset($_SERVER['RDS_HOST'])) ?
5     ($_SERVER['RDS_HOST']) : ('localhost')
6 );
7 /* And following lines */
```

In `config/services.yaml` replace the dummy text with your Google Maps API Key.

```
1 parameters:
2     locale: 'en'
3     app.gmaps_api_key: '<YOUR GMAPS API KEY HERE>'
```

Step 4: build assets and create database schema

Build assets with

```
D:\Desktop\serverful-webapp> npm run webpack-dev
```

Then create the database and the data schema by running

```
D:\Desktop\serverful-webapp> npm run drop-database
D:\Desktop\serverful-webapp> npm run create-database
D:\Desktop\serverful-webapp> npm run create-schema
```


Step 5: run the app

Now you can start the dev server anche check out the app.

```
D:\Desktop\serverful-webapp> npm run serve
```

Once the server started, visit the webapp at localhost:8000

Step 6: create a source bundle

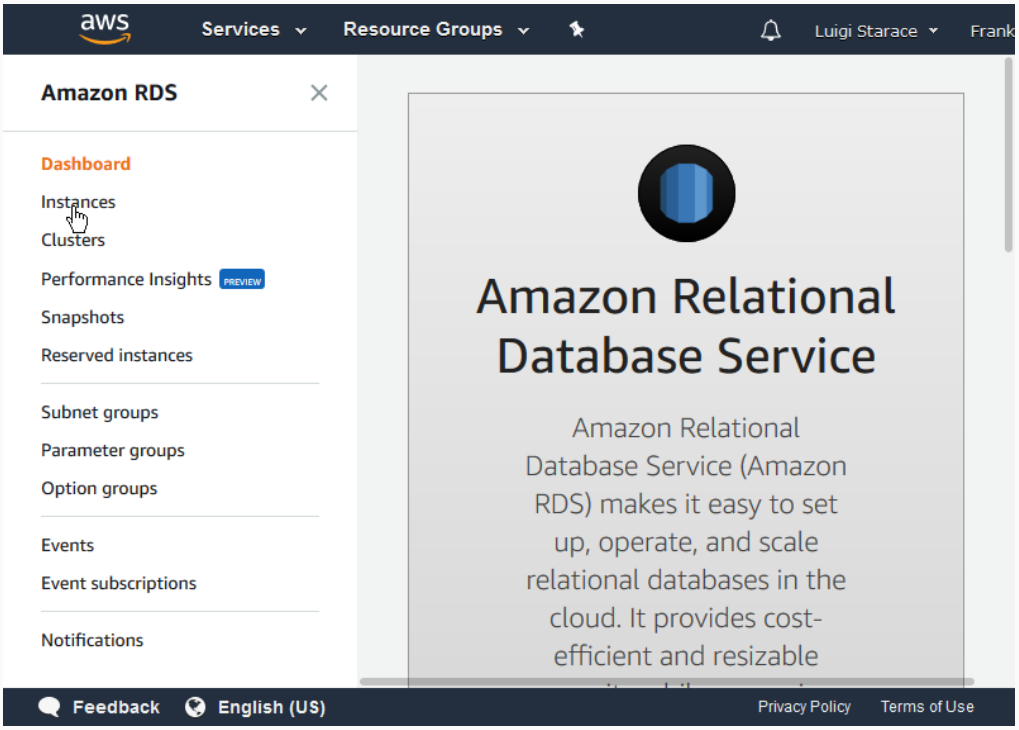
Elastic Beanstalk requires a single WAR or ZIP archive containing you app. To create a source bundle for our app, run

```
D:\Desktop\serverful-webapp> npm run create-source-
    bundle
```

A `serverful-app.zip` (our source bundle) archive will be created in the app root.

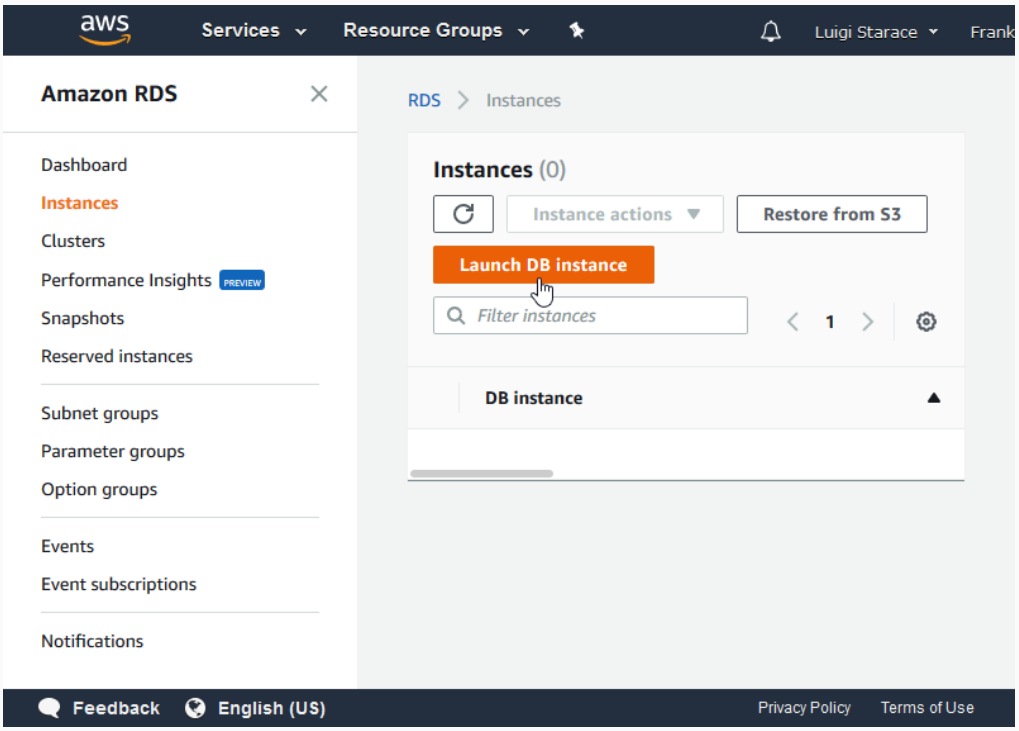
Step 7: Create a Database Instance

Go to the RDS Console and select "instances".



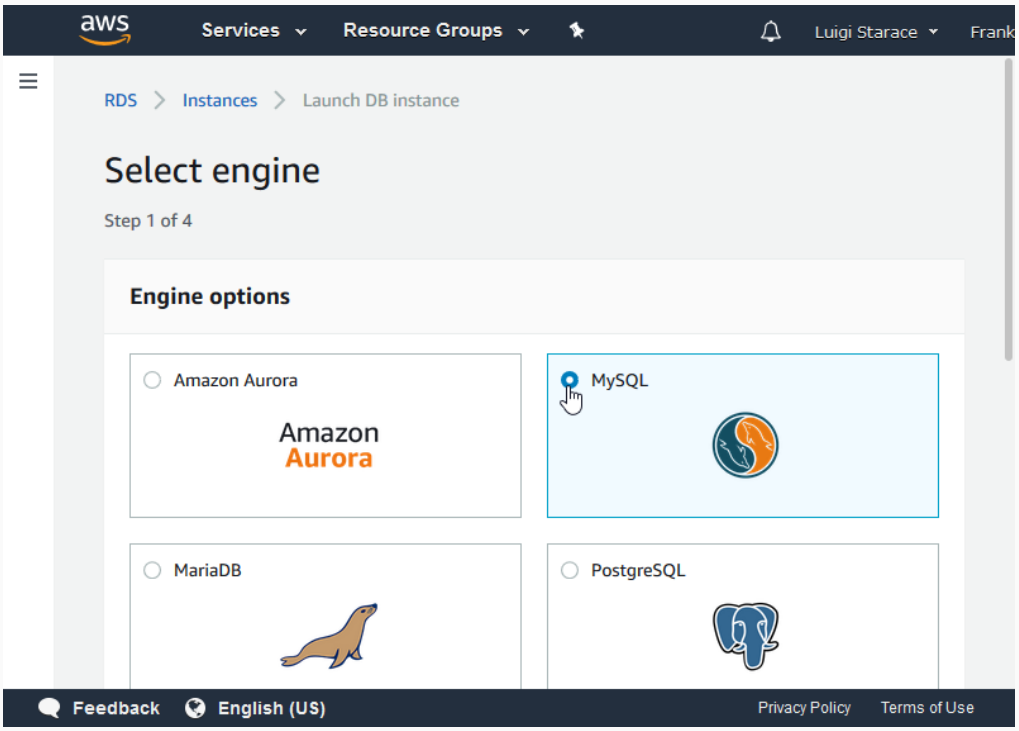
Step 7: Create a Database Instance

Select “Launch DB instance” .



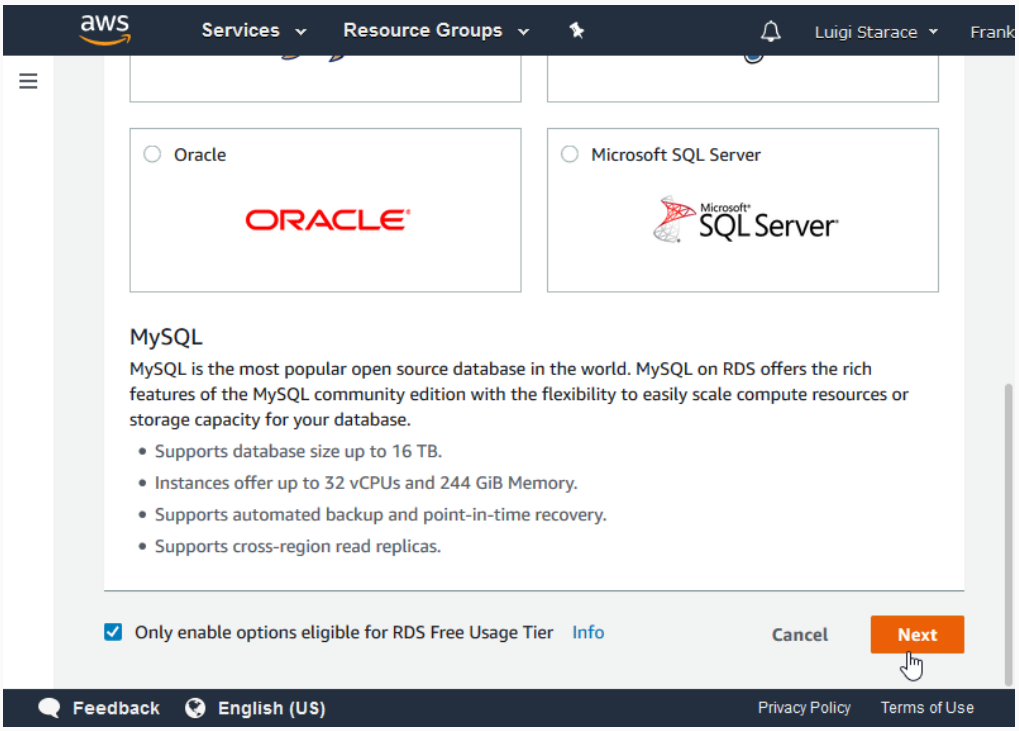
Step 7: Create a Database Instance

Select MySQL DBMS.



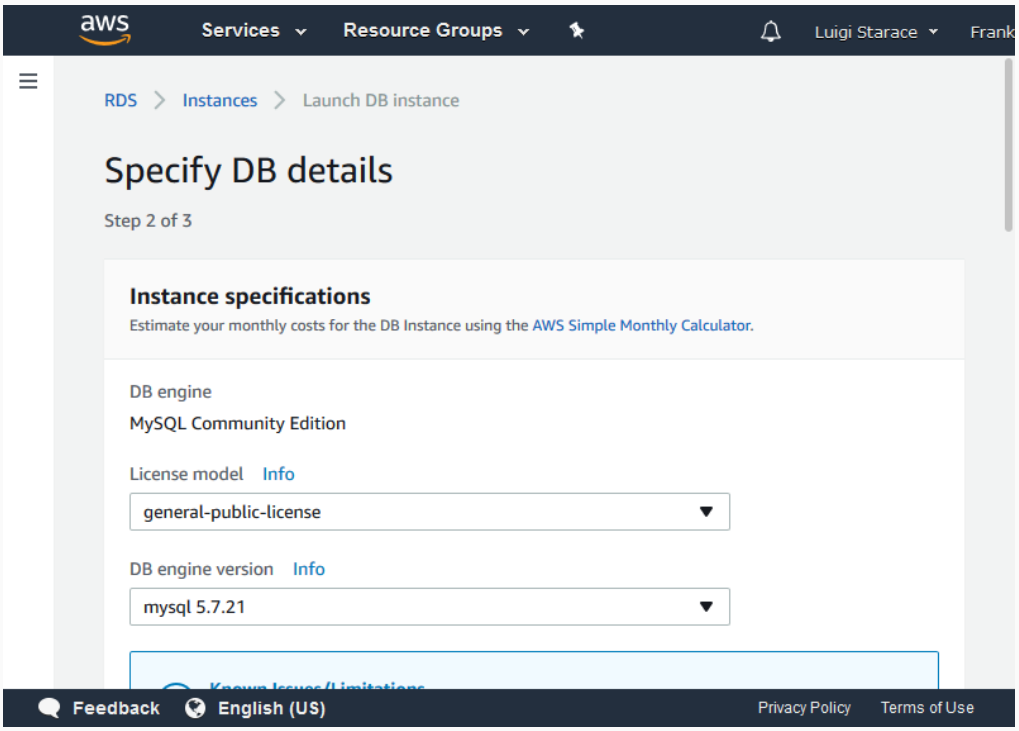
Step 7: Create a Database Instance

Enable only free-tier options and continue.



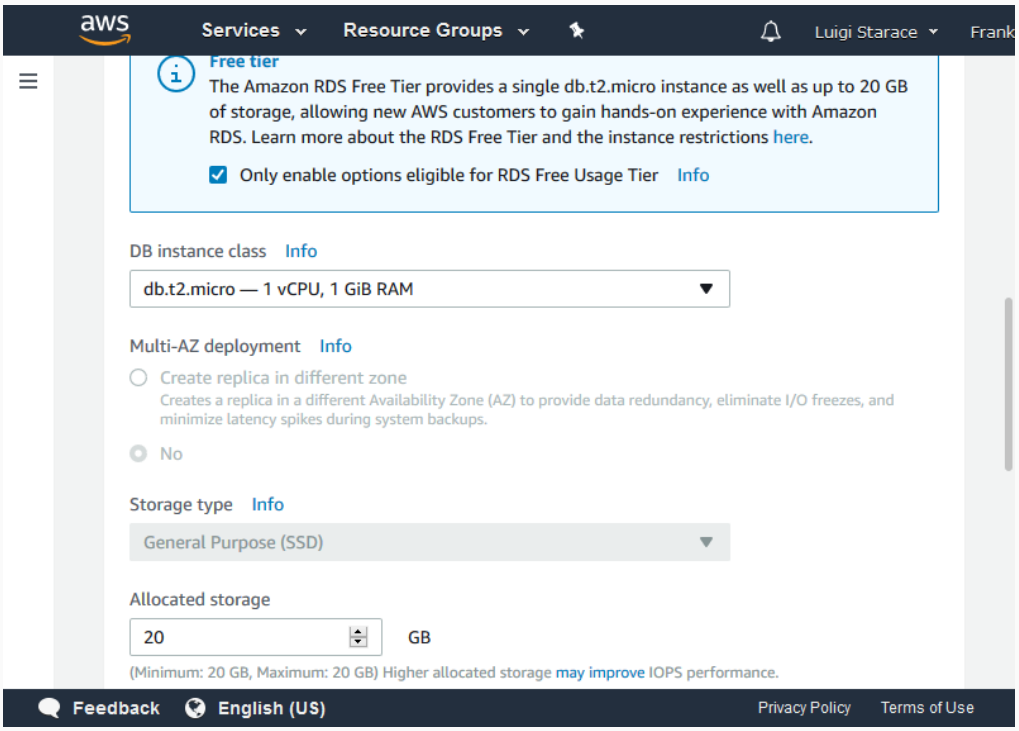
Step 7: Create a Database Instance

Select MySQL version `5.7.21`



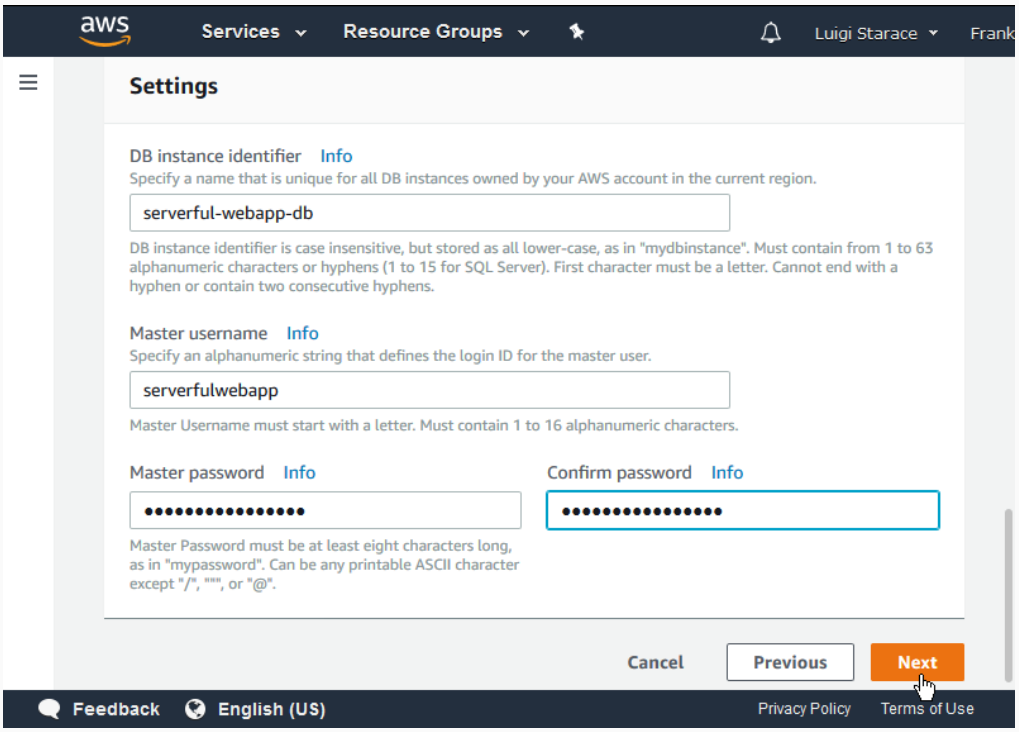
Step 7: Create a Database Instance

Select `db.t2.micro` instance.



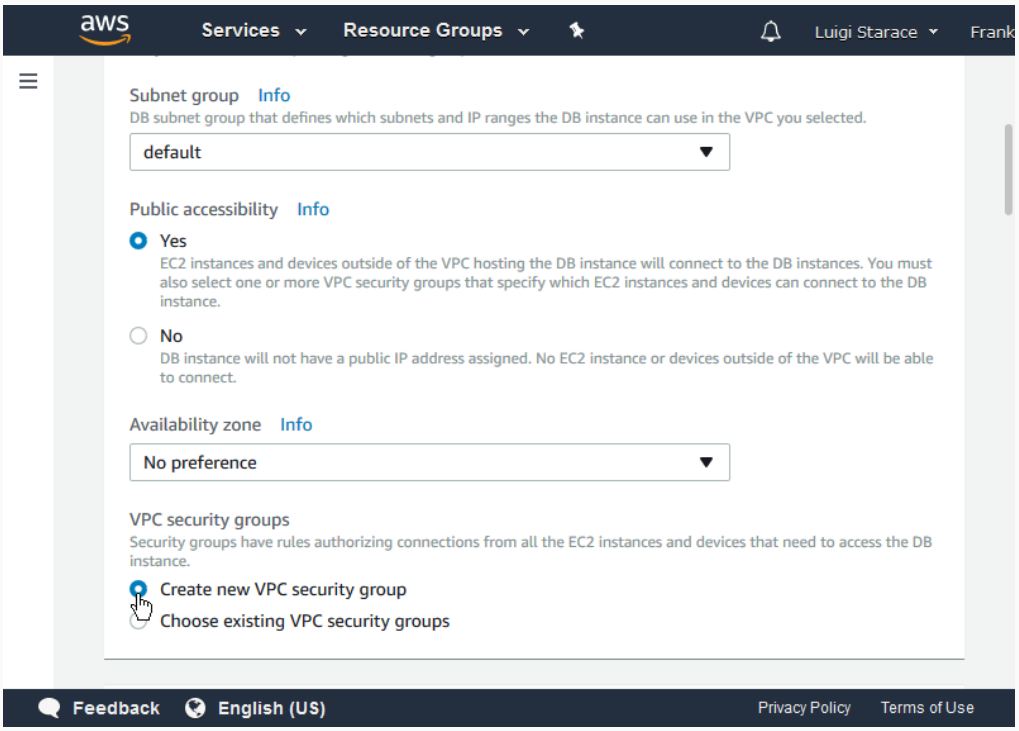
Step 7: Create a Database Instance

Enter your desidered settings (**remember the password!** ⚠).



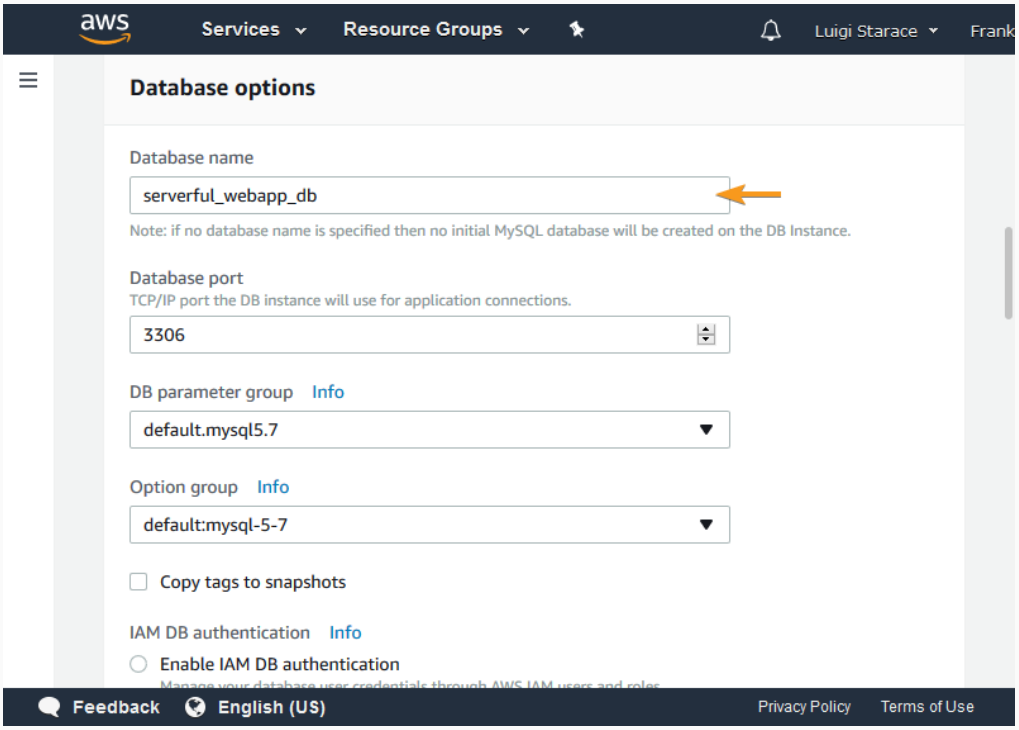
Step 7: Create a Database Instance

Be sure to select “create a new security group” .



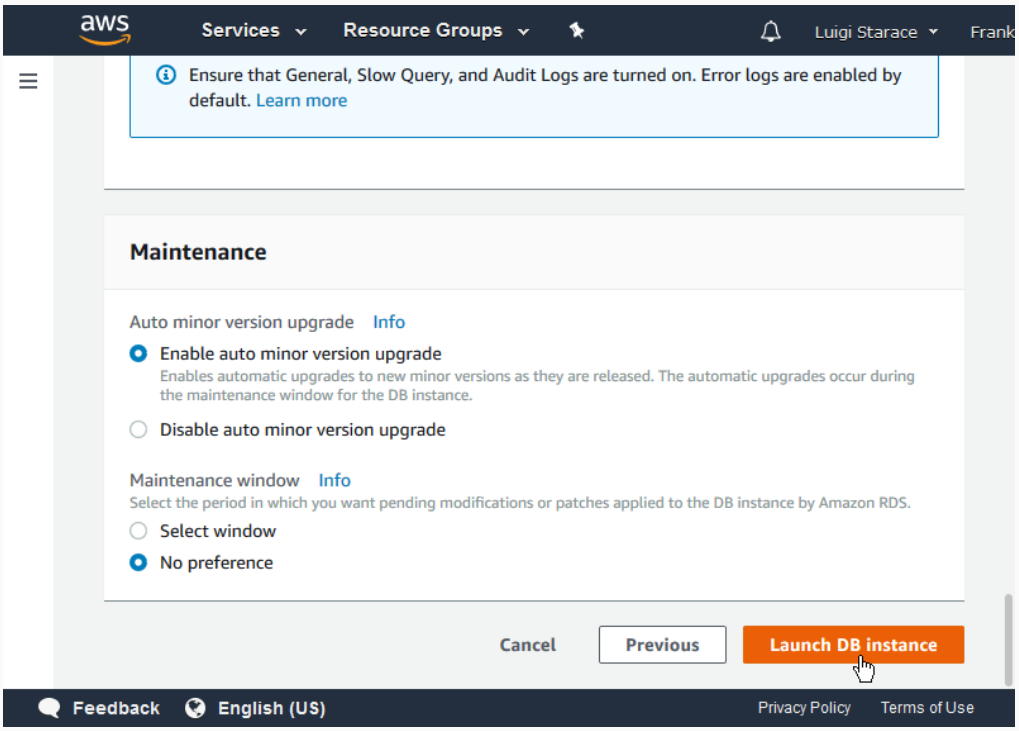
Step 7: Create a Database Instance

Enter a database name for the instance (**important!** ⚠) and leave the rest as is.



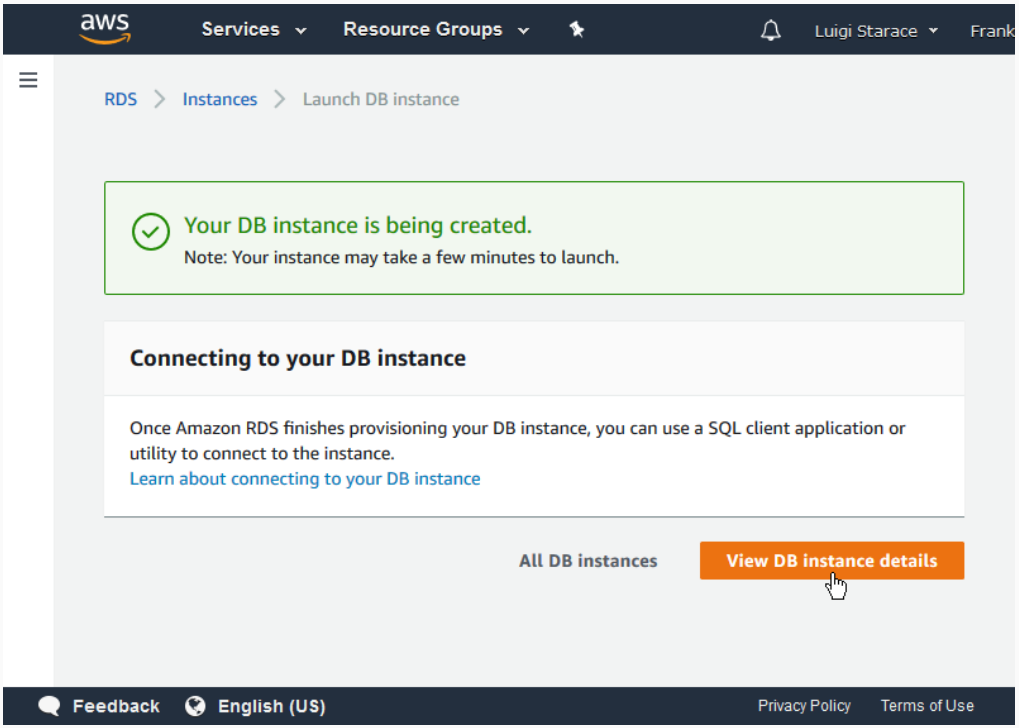
Step 7: Create a Database Instance

Click on "Launch DB Instance".



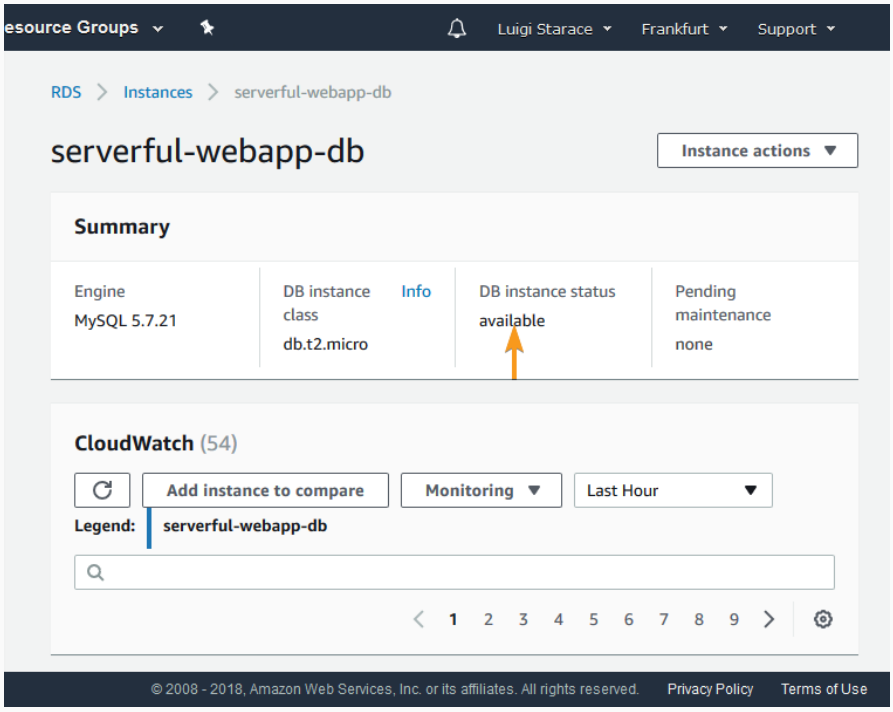
Step 7: Create a Database Instance

The creation process takes around 15 minutes. Click on "View DB Instance Details" to visit the detail page for the instance you just created.



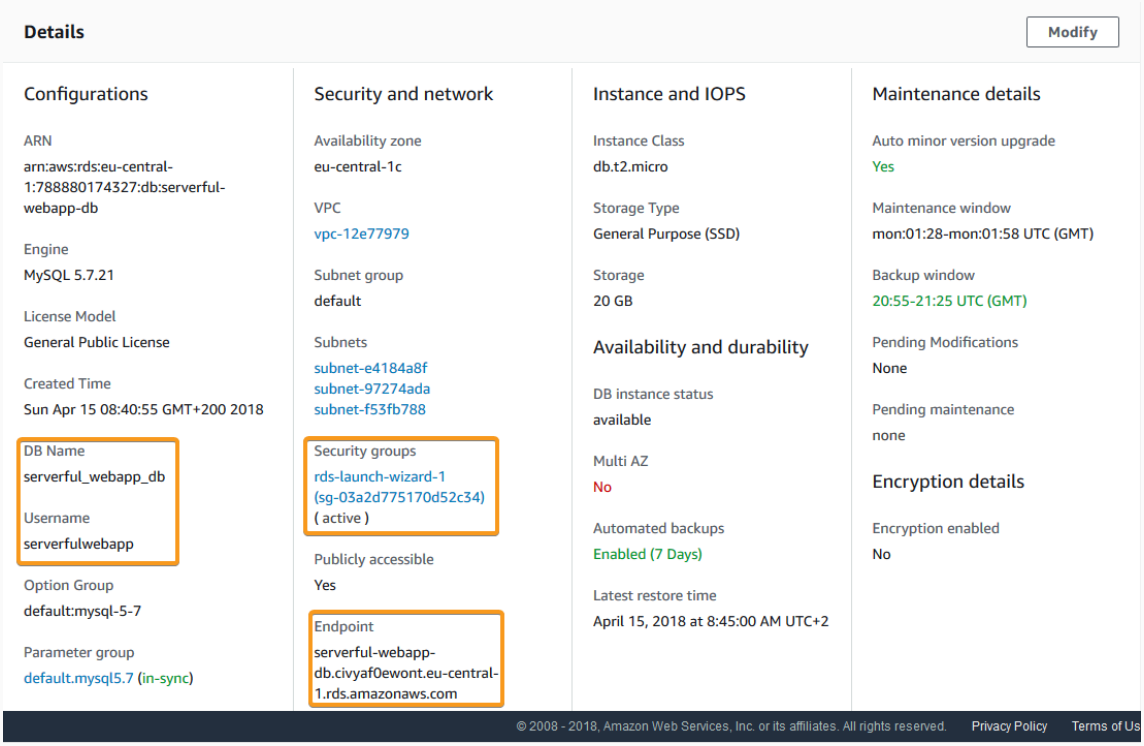
Step 7: Create a Database Instance

When done, the status in your instance detail page will change to “available” .



Step 7: Create a Database Instance

Notice a few important elements in the details section. We’re going to need these later.

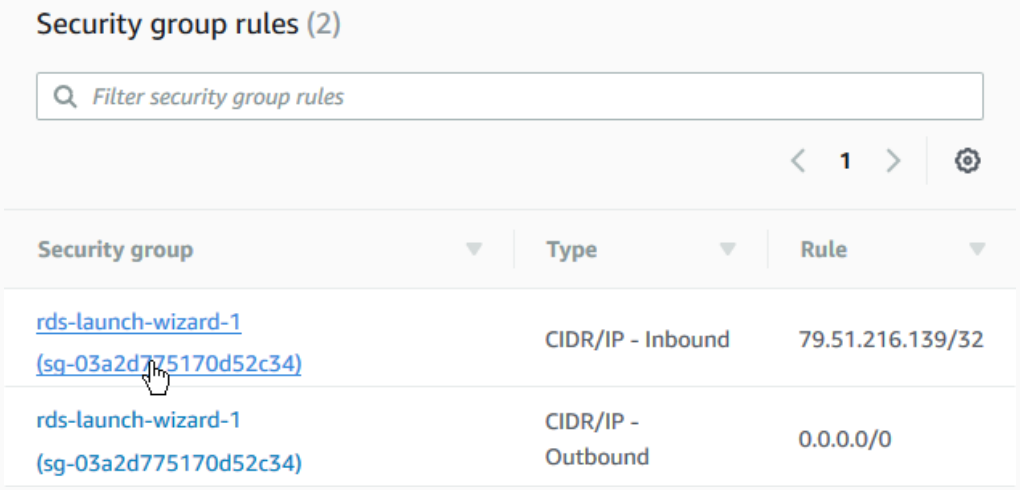


Step 7: Create a Database Instance

We'll need this instance to be accessible by our web application. To do so we're going to add a new rule to allow all instances in the same security group to access the database instance.

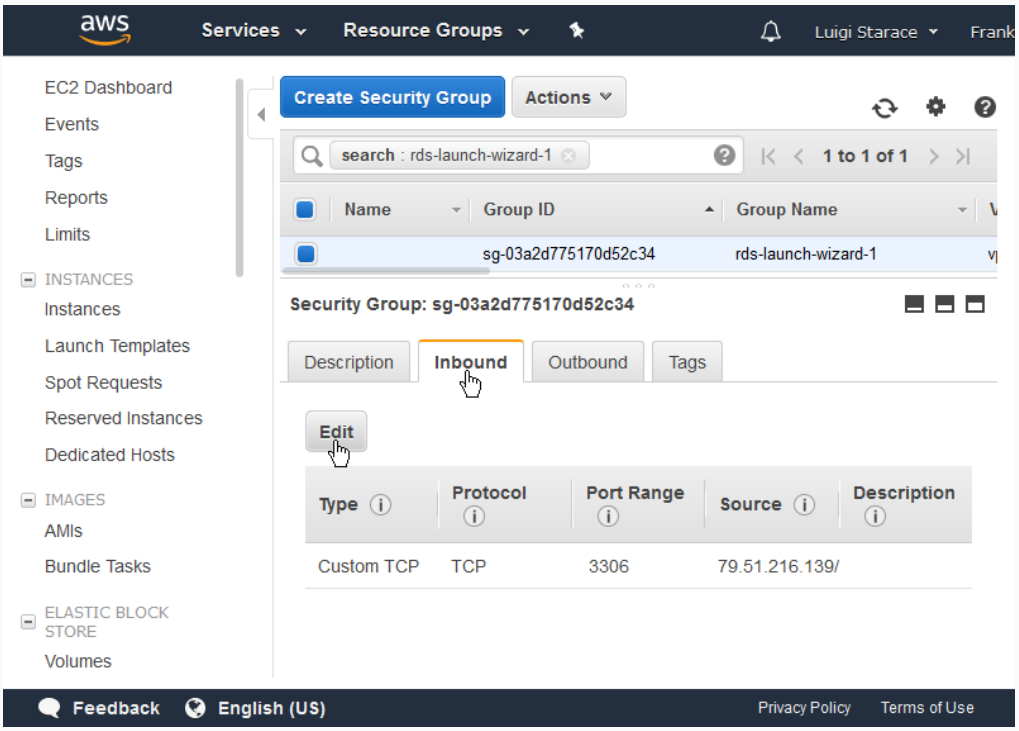
Step 7: Create a Database Instance

Click on the security group in the section *Security Group Rules*.



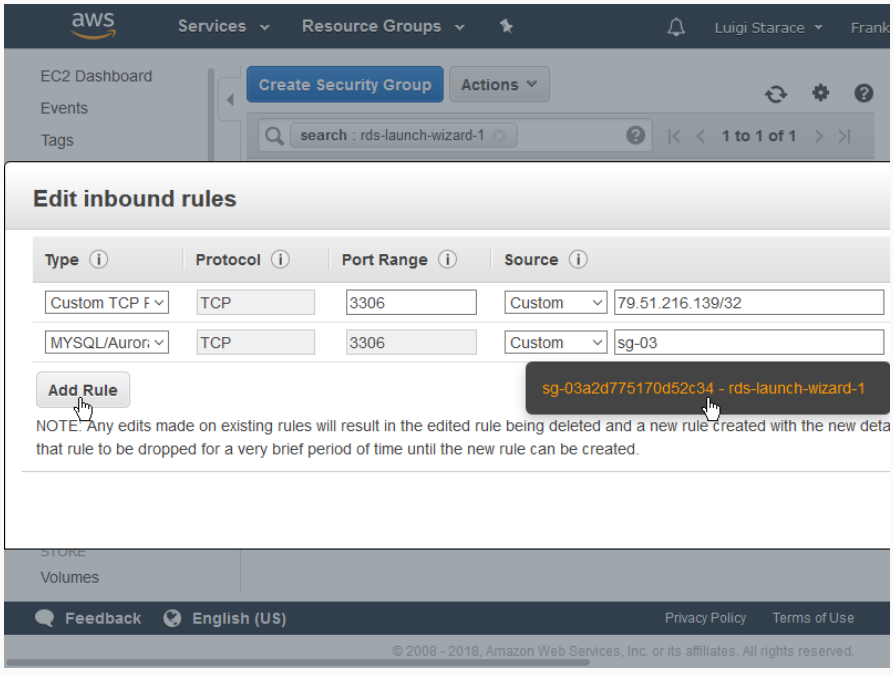
Step 7: Create a Database Instance

Select the *Inbound* tab then click on the Edit button.



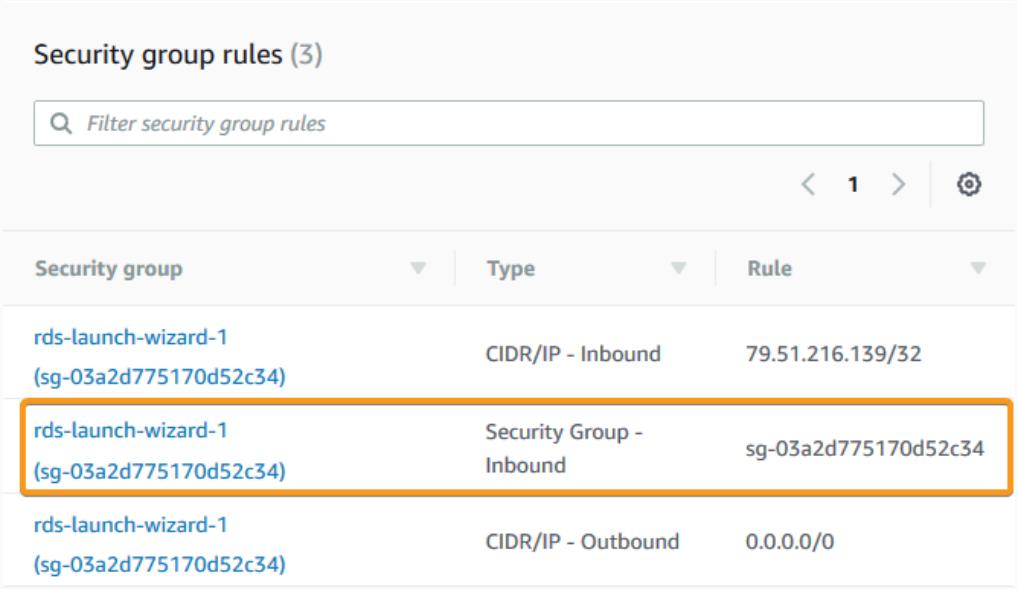
Step 7: Create a Database Instance

Add a new rule as shown in the picture. Be sure to select the same security group of the database instance. Then save and return to the RDS instance detail page.



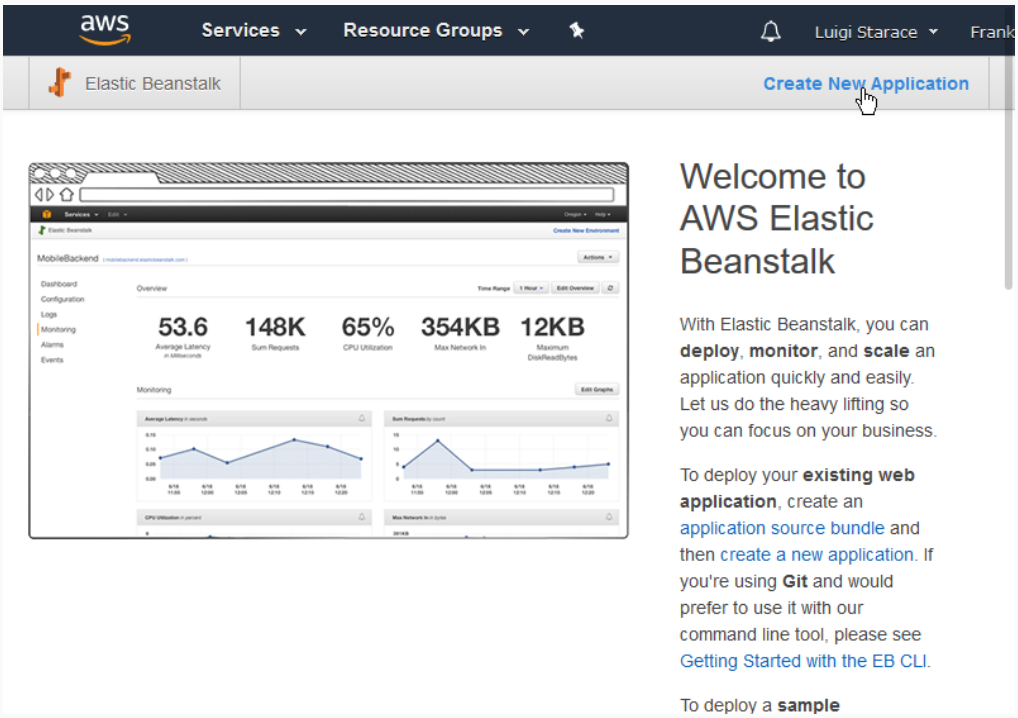
Step 7: Create a Database Instance

The rule you just added should be displayed among the other two.



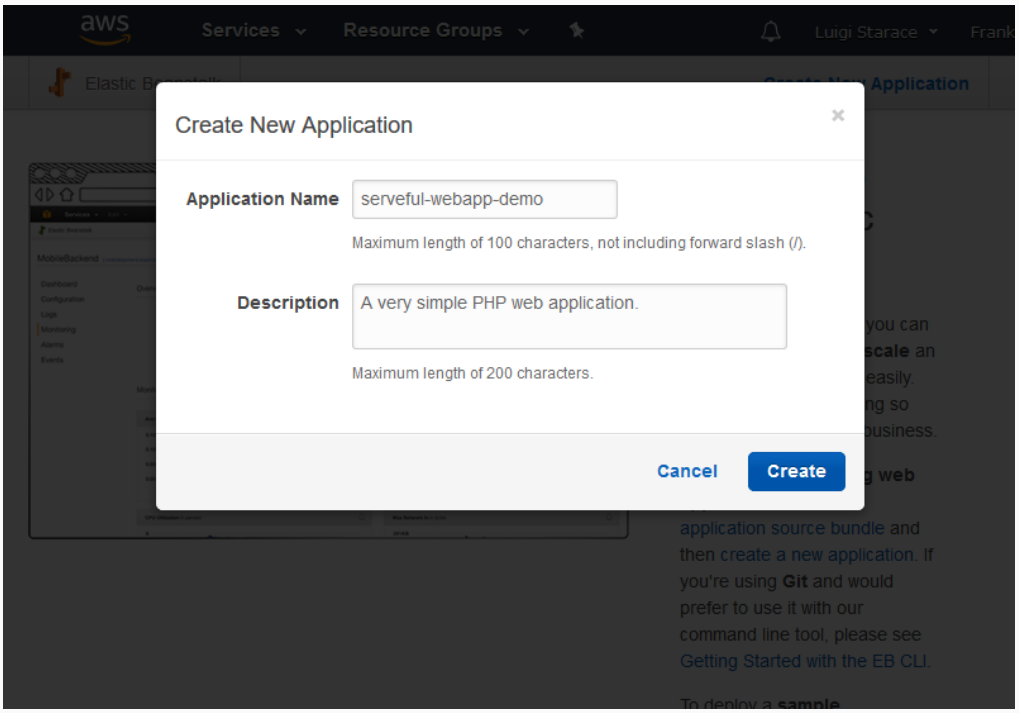
Step 7: Create a Beanstalk application

Go to the Beanstalk console and select *Create New Application*.



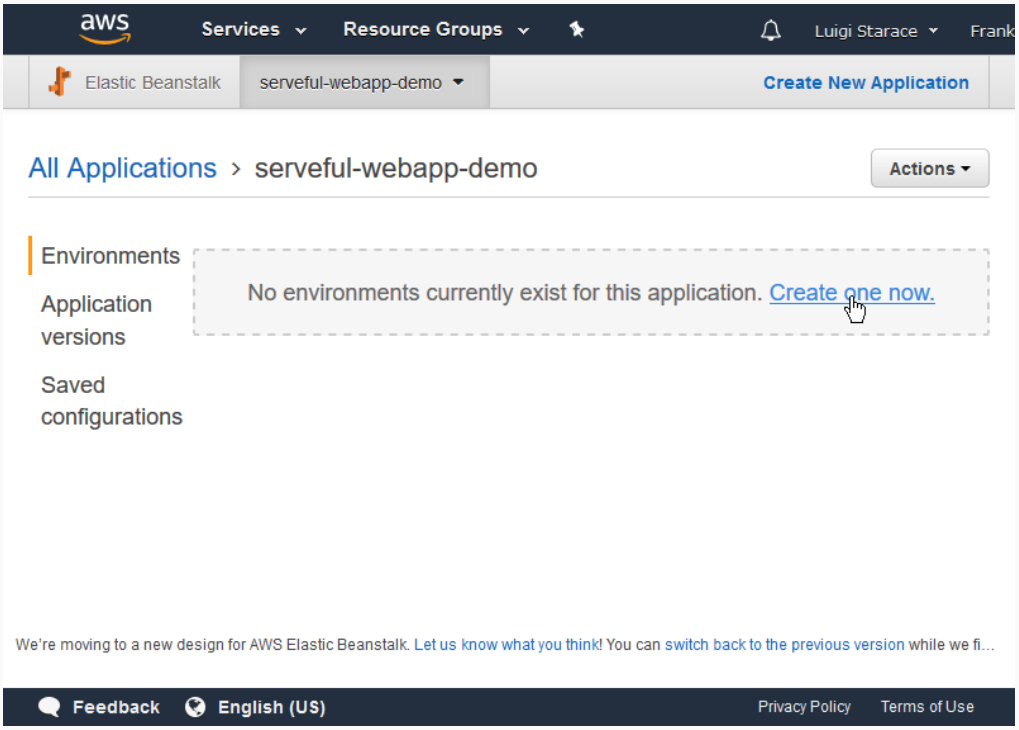
Step 7: Create a Beanstalk application

Fill the form with your application information and continue.



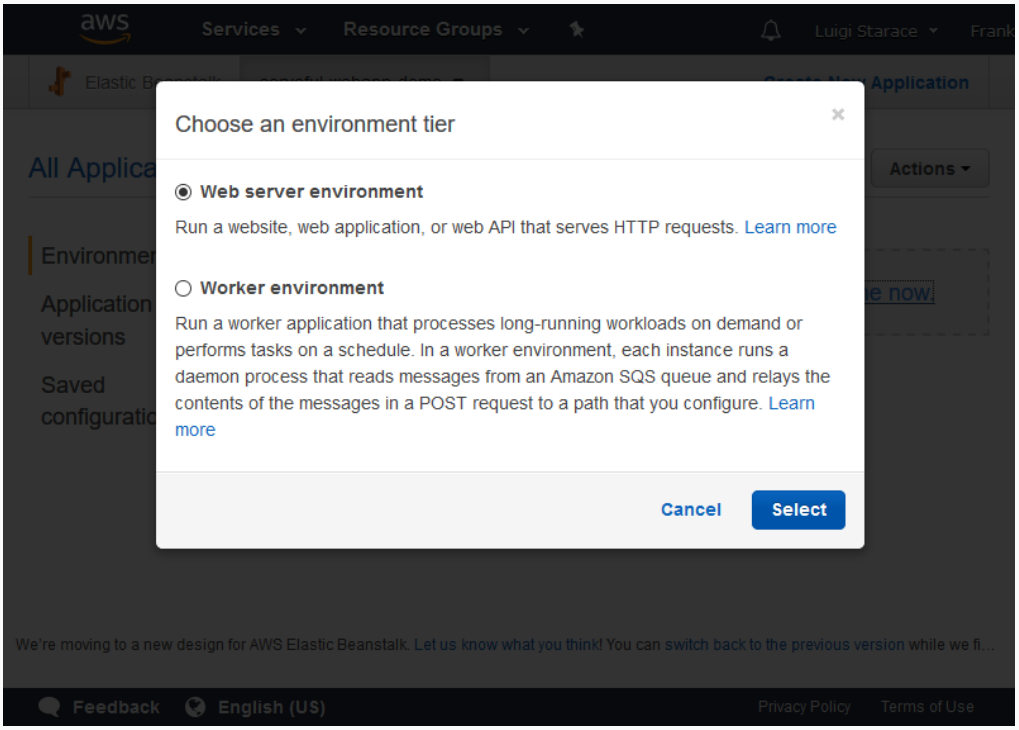
Step 7: Create a Beanstalk application

Then select *Create one now* to create a new environment for your application.



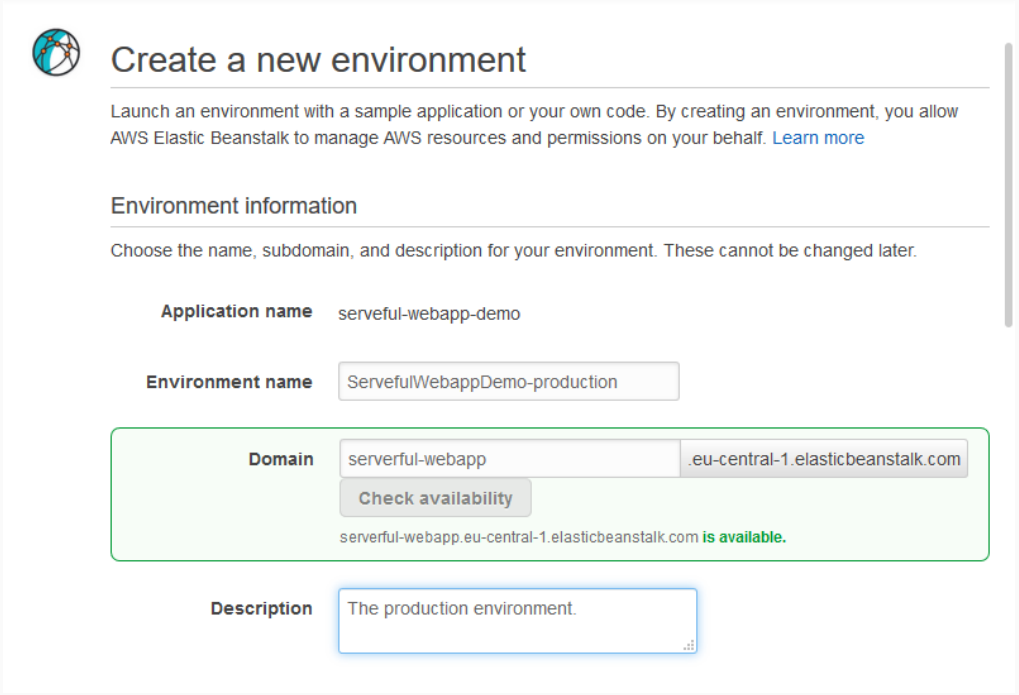
Step 7: Create a Beanstalk application

Select *Web Server Environment*, as we are going to deploy a web application.



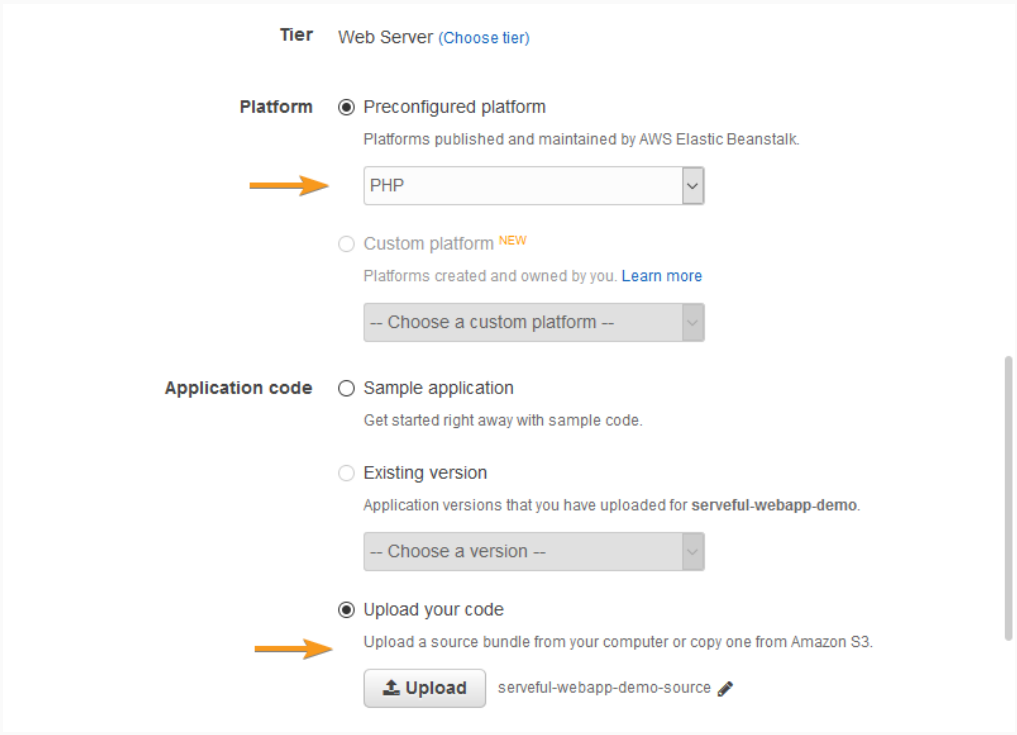
Step 7: Create a Beanstalk application

Fill the form with information about your environment.



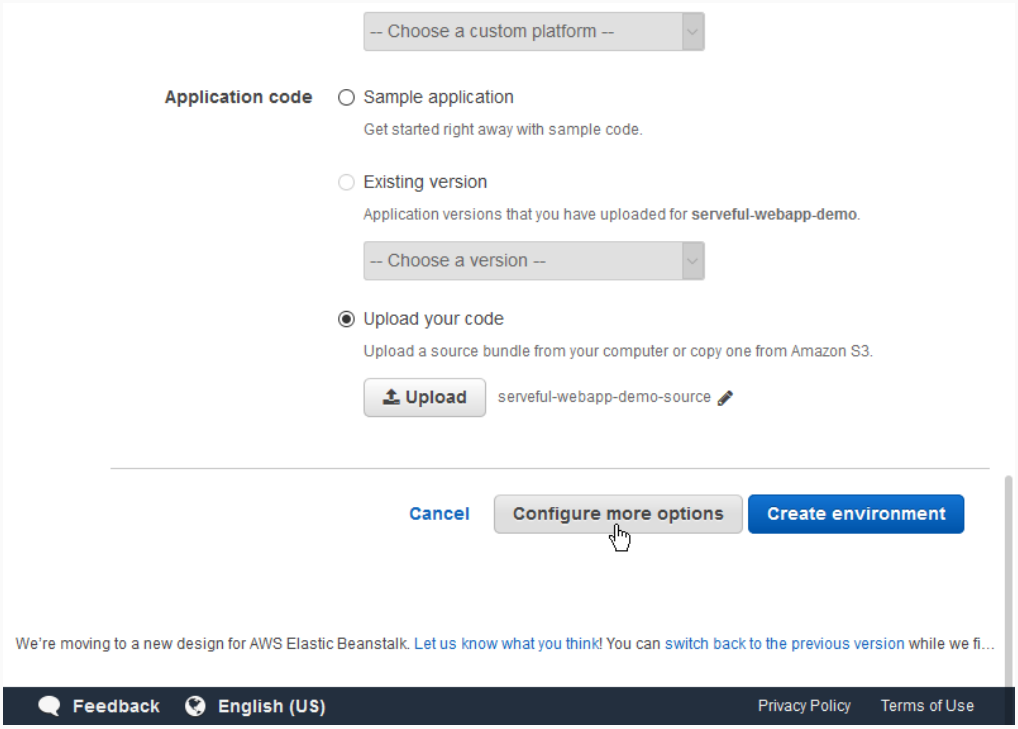
Step 7: Create a Beanstalk application

Select PHP as preconfigured platform and upload the source bundle you previously prepared.



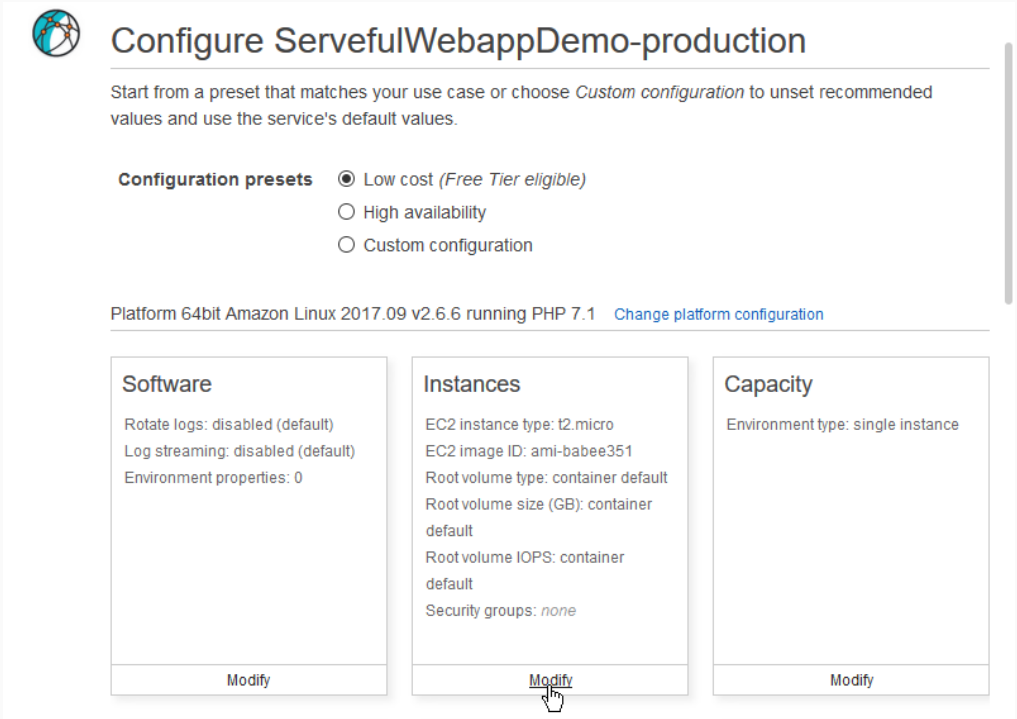
Step 7: Create a Beanstalk application

Select *Configure More Options* and continue.



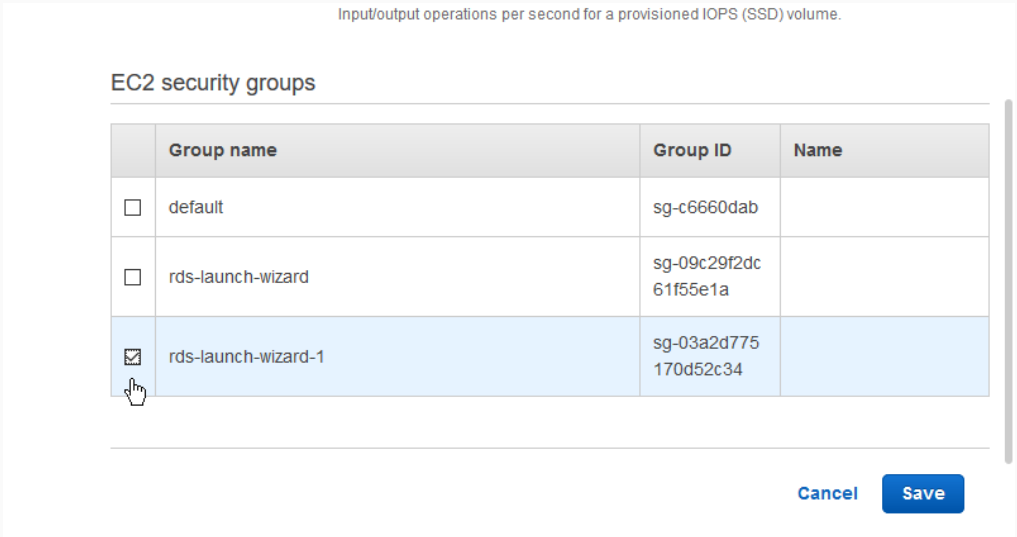
Step 7: Create a Beanstalk application

In the configuration page, select *modify* con the *Instances* card.



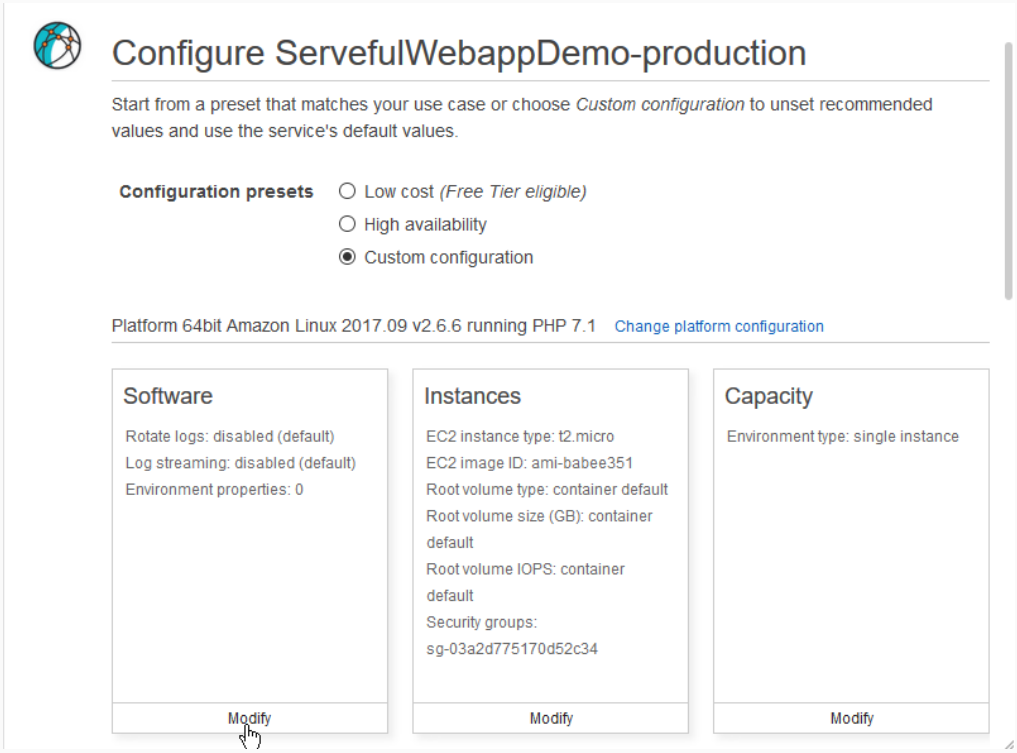
Step 7: Create a Beanstalk application

In the instances configuration page, add the t2 instance to the same security group as the DB instance. Then save and continue.



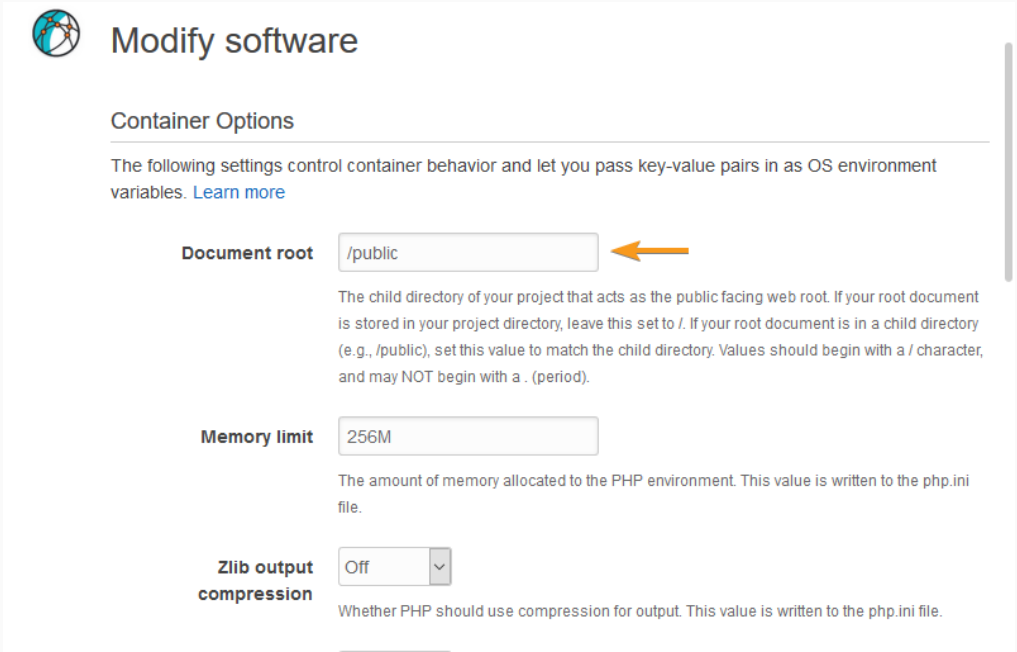
Step 7: Create a Beanstalk application

In the configuration page, select *modify* con the *Software* card.



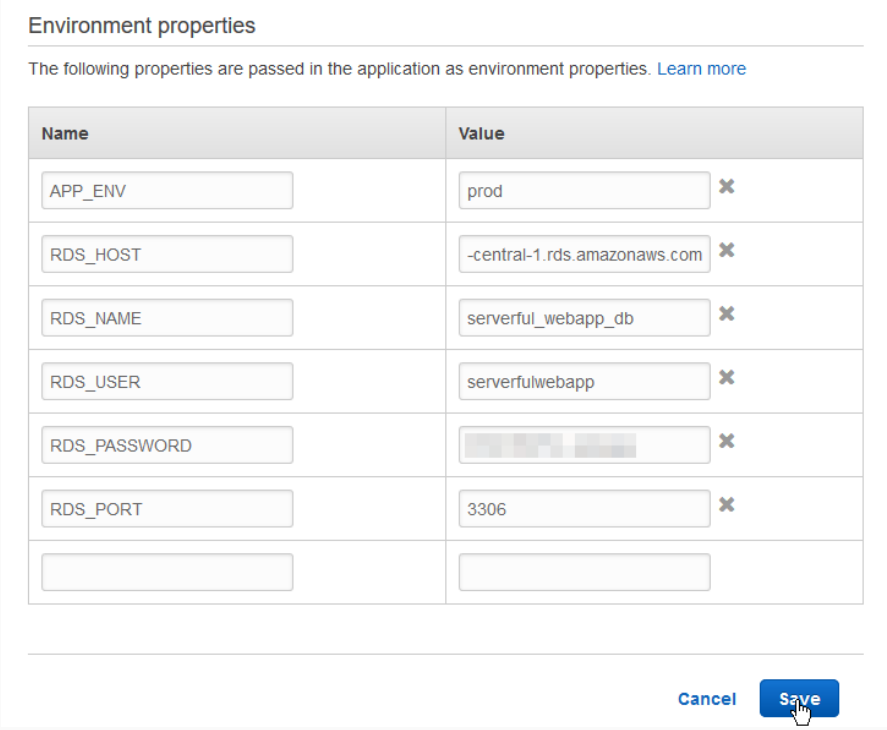
Step 7: Create a Beanstalk application

Enter "/public" as the document root and scroll down.



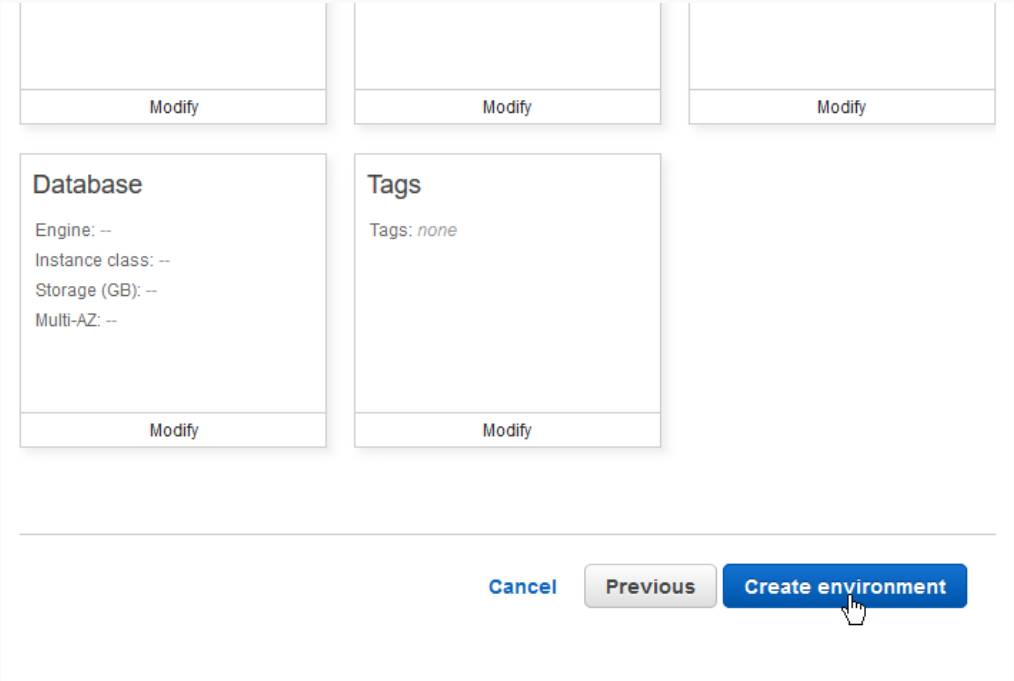
Step 7: Create a Beanstalk application

Enter the required enviroment parameters are show in the picture. Be careful, deployment might fail if you mess up! ⚠



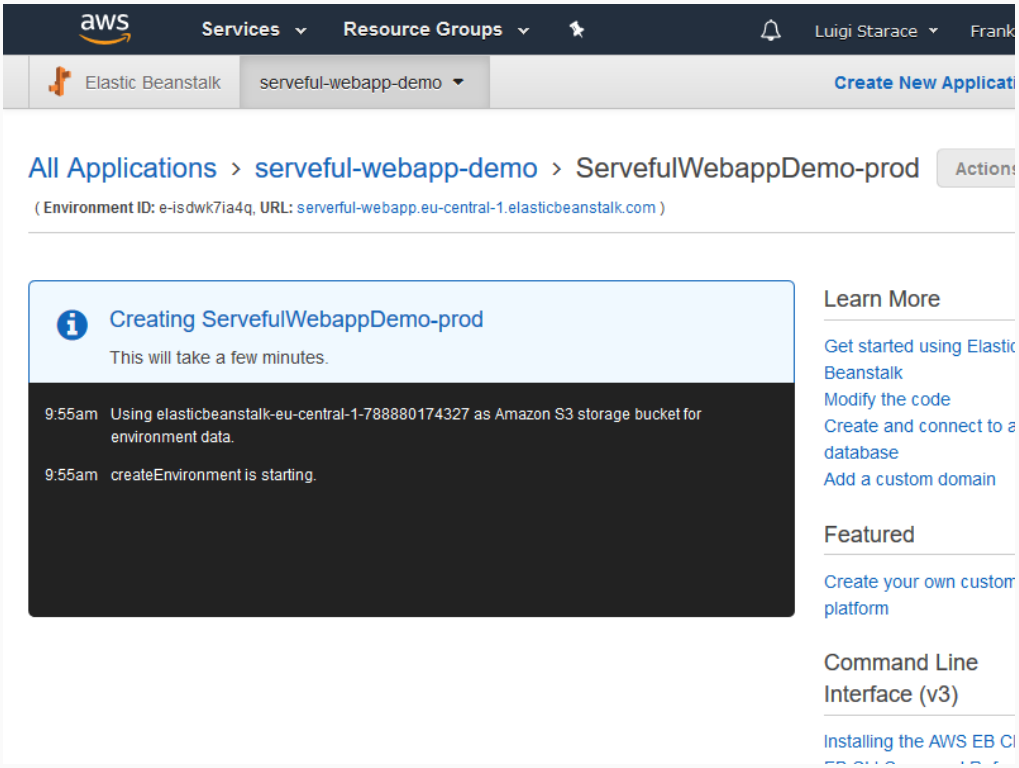
Step 7: Create a Beanstalk application

Click on *Create Enviroment* and continue.



Step 7: Create a Beanstalk application

Wait for the enviroment to be created. This takes about 10 minutes.

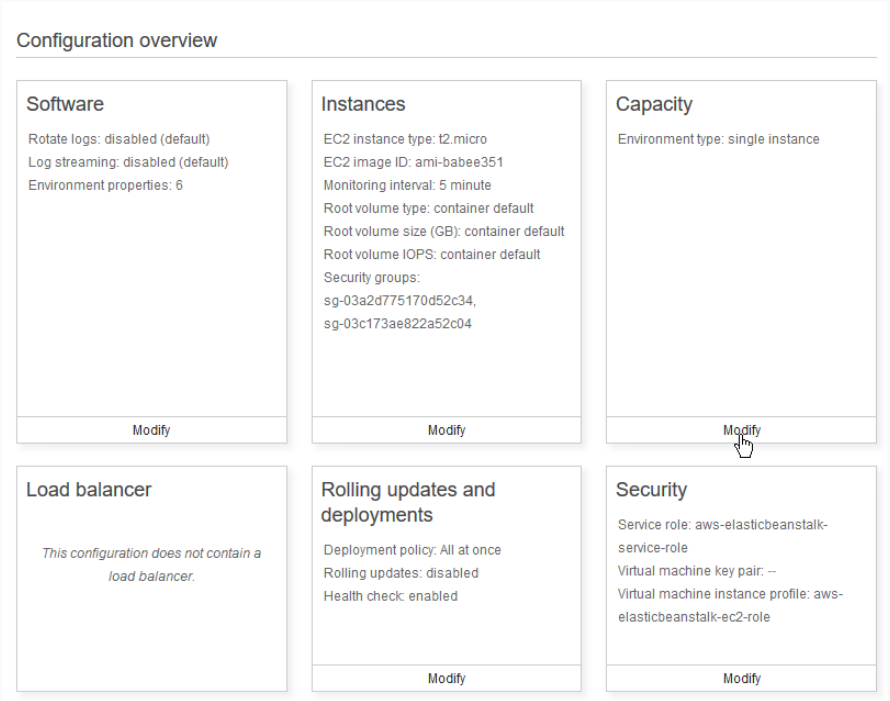


Step 8: Load balancing

Right now we have our application running on a single (virtual) web server. That's not scaling at all. Let's take advantage of the cloud and make the web application load balanced.

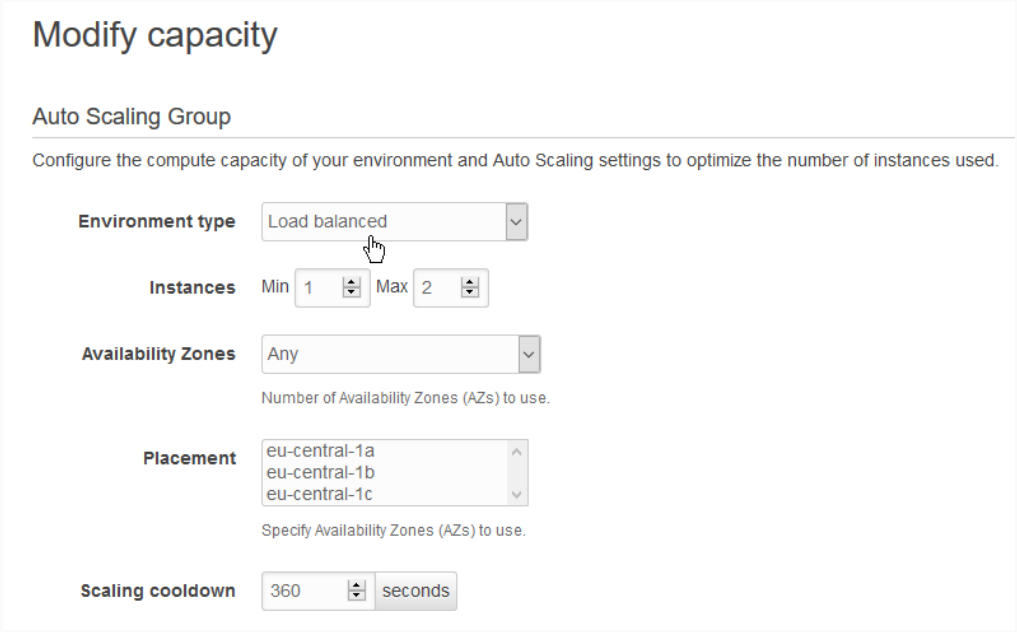
Step 8: Load balancing

Select the environment's configuration view, then select the Capacity card.



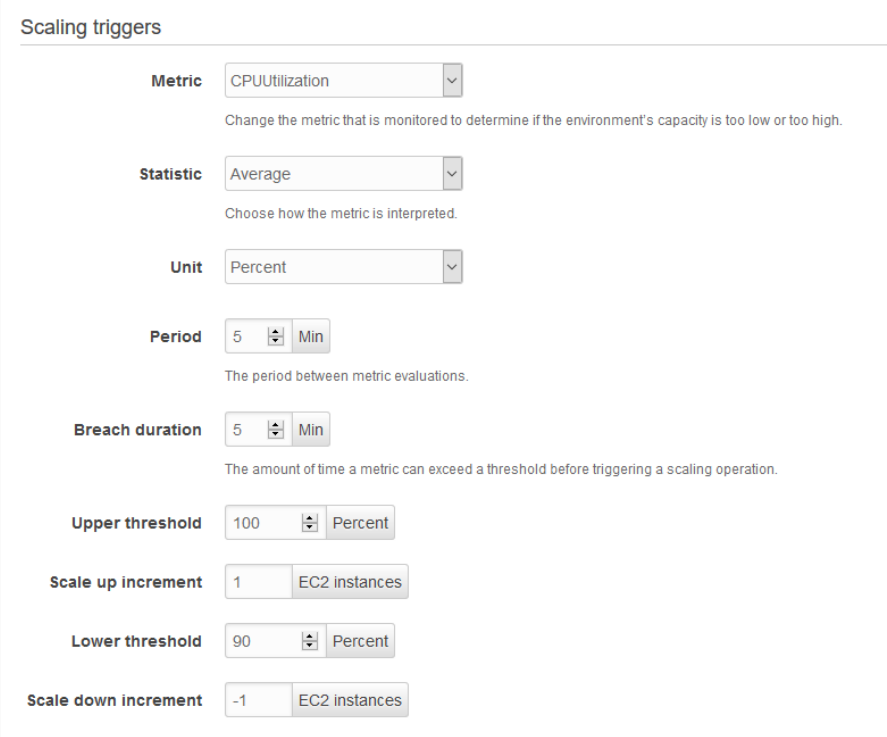
Step 8: Load balancing

Select "Load balanced" as the environment type and customize the Auto Scaling Group.



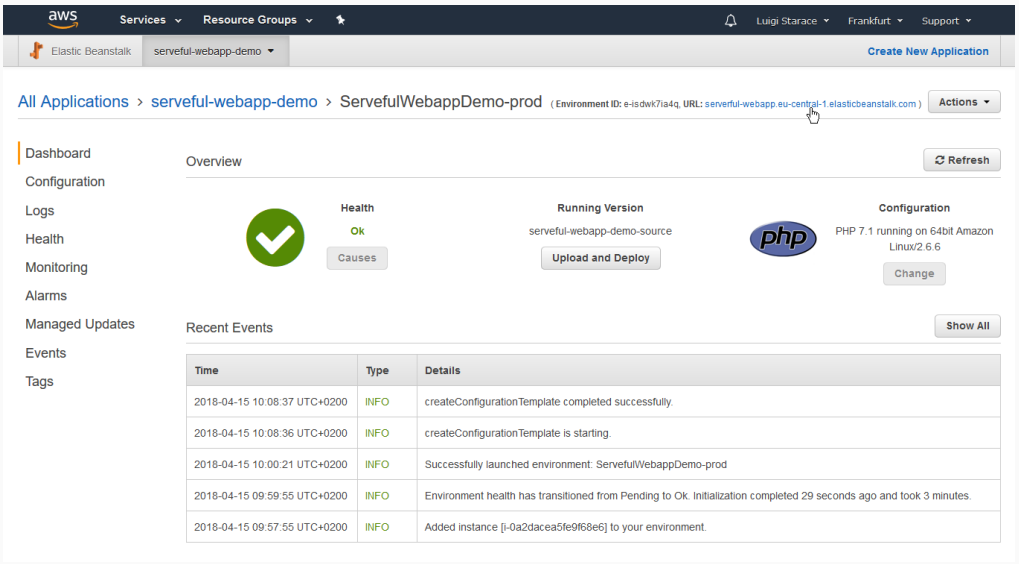
Step 8: Load balancing

Select some triggers (you can even setup time based ones), then save your changes.



Step 9: Enjoy your web app

When it's done you should see something like this. Click on the URL to visit the load-balanced web application you just deployed on Beanstalk!



Step 9: Enjoy your web app

Sweet!

Let's get back to computing services

Amazon Lambda

- You provide the code and say when to run it.
- Execution is triggered by events
  - S3, Cognito, DynamoDB
  - CodeCommit, Scheduled Event
- Support for Java, Node.js, C# e Python (more to come).
- Pay only for **actual** execution time.
- Run your code without thinking about infrastructure
  - No need to worry about provisioning, load balancing, scaling...

Amazon Lambda — Notes

- Named after anonymous (lambda) functions?
- Stress the *actual* execution time part.
- Give some examples of events triggering lambda execution
  - API Gateway call
  - File uploaded, record updated/added, chron events, ecc...

Amazon Lambda: limits

AWS Lambda imposes some limits

- Max 300 seconds execution time.
- Max 3008 MB memory allocation.
- Deployment package must be smaller than 50 MB (negotiable).
- No more than 10000 concurrent invocation of a Lambda function in a given region (negotiable).
- For a complete list: Lambda docs

Amazon Lambda: FaaS

FaaS (Functions as a Service)

- Functions are the unit of deployment
- Executed in ephemeral, stateless containers
- Event driven
- No provisioning, scales automatically
- Azure: Functions web
- Google Cloud: Functions web
- IBM: Cloud Functions web
  - Based on Apache OpenWhisk web

Amazon Step Functions

- Orchestrating Lambda functions
- Define a state machine

Amazon Step Functions: sample I

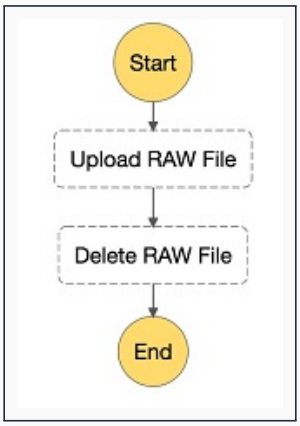


Figure 3: Sequential steps, from [AWS]

Amazon Step Functions: sample II

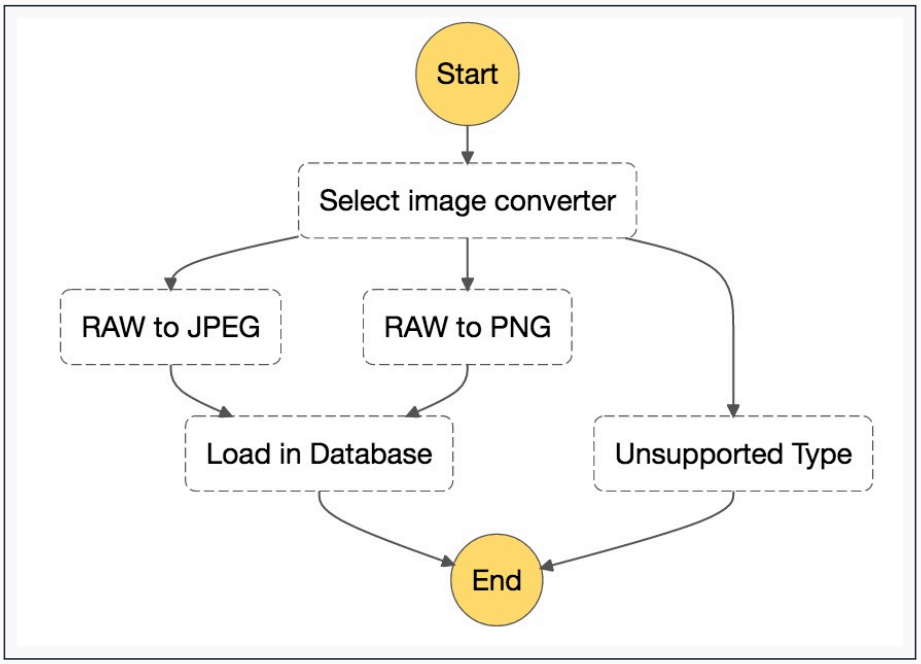


Figure 4: Branching, from [AWS]

Amazon Step Functions: sample III

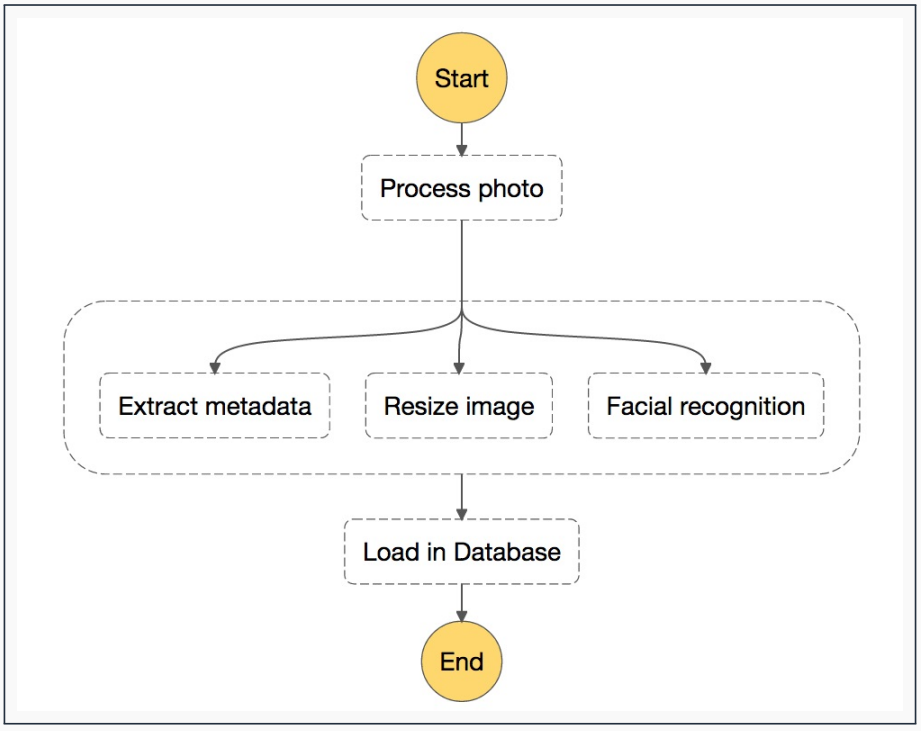


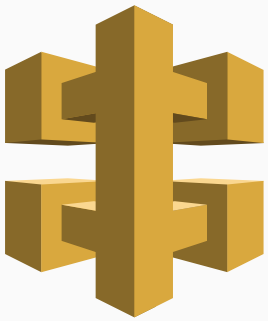
Figure 5: Parallel execution, from [AWS]

Amazon Step Functions: sample IV

Figure 6: Monitoring executions, from [AWS]

Amazon API Gateway

- Create, publish, and secure APIs at any scale
- Authorizers (Cognito)



Serverless?

What's all the **FaaS** about?

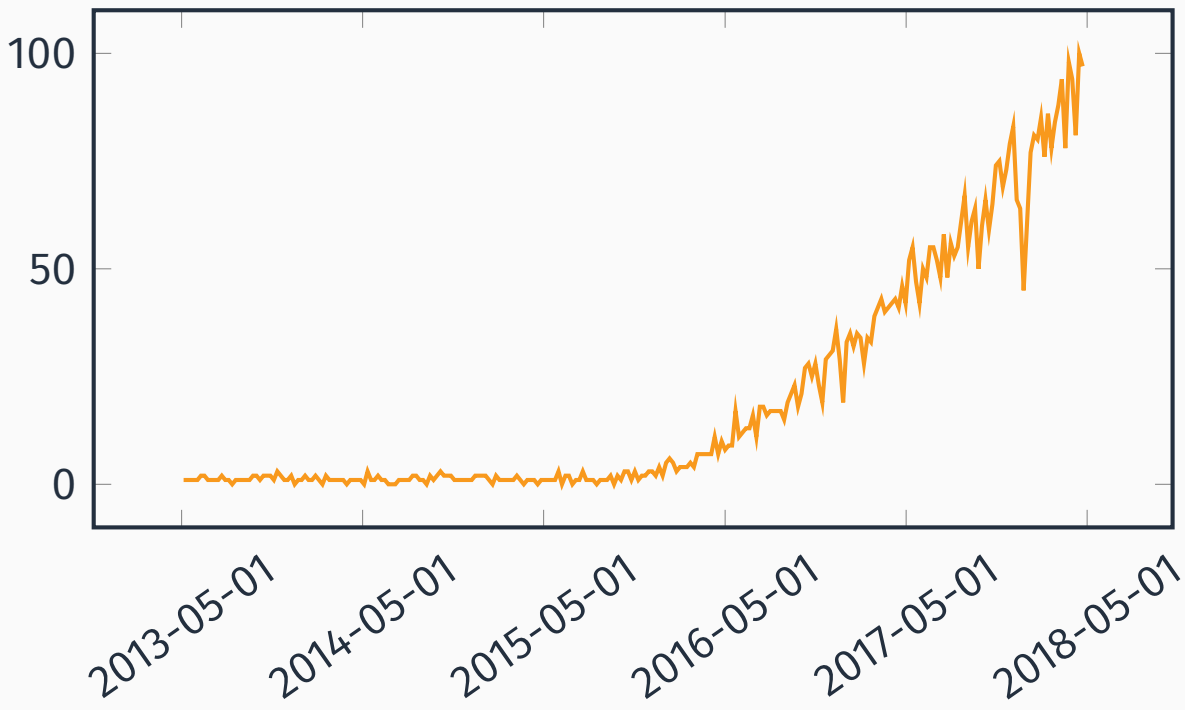


Figure 7: Last five years trend on Google for "serverless"

- No, they're not *actually* serverless...
- Rely on FaaS and third-party services so that traditional always-on servers are no longer needed
- No worries about provisioning and scaling
- "Smarter" clients

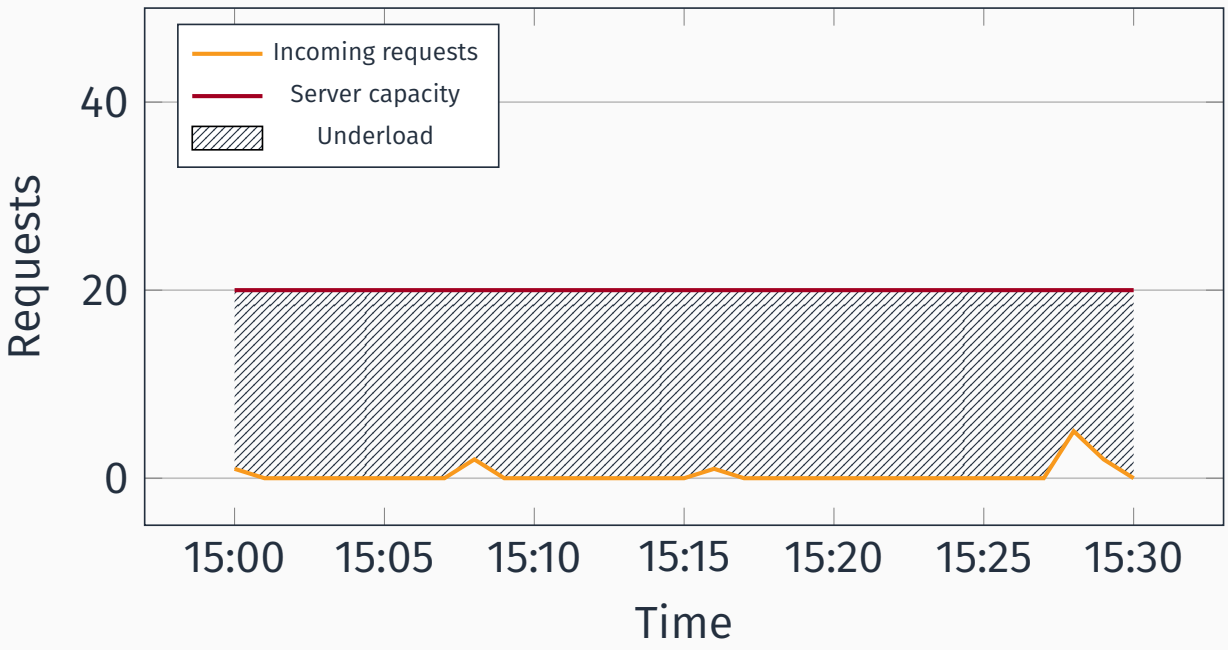


Figure 8: Sporadic requests example

Serverless use cases: inconsistent requests

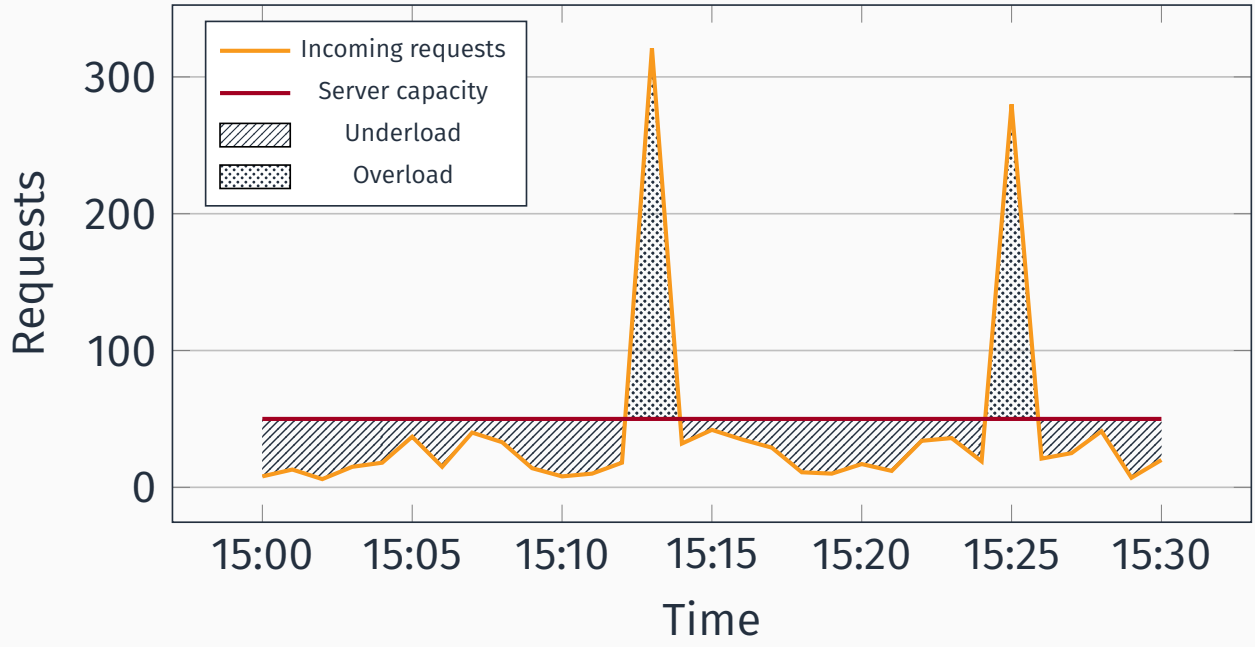


Figure 9: Inconsistent requests example

Serverless Architectures: trade-offs

Pros

- Reduce costs servers.LOL
- No worries about provisioning, scaling
- Less time to market

Cons

- Limits
- Vendor lock-in
- Testing

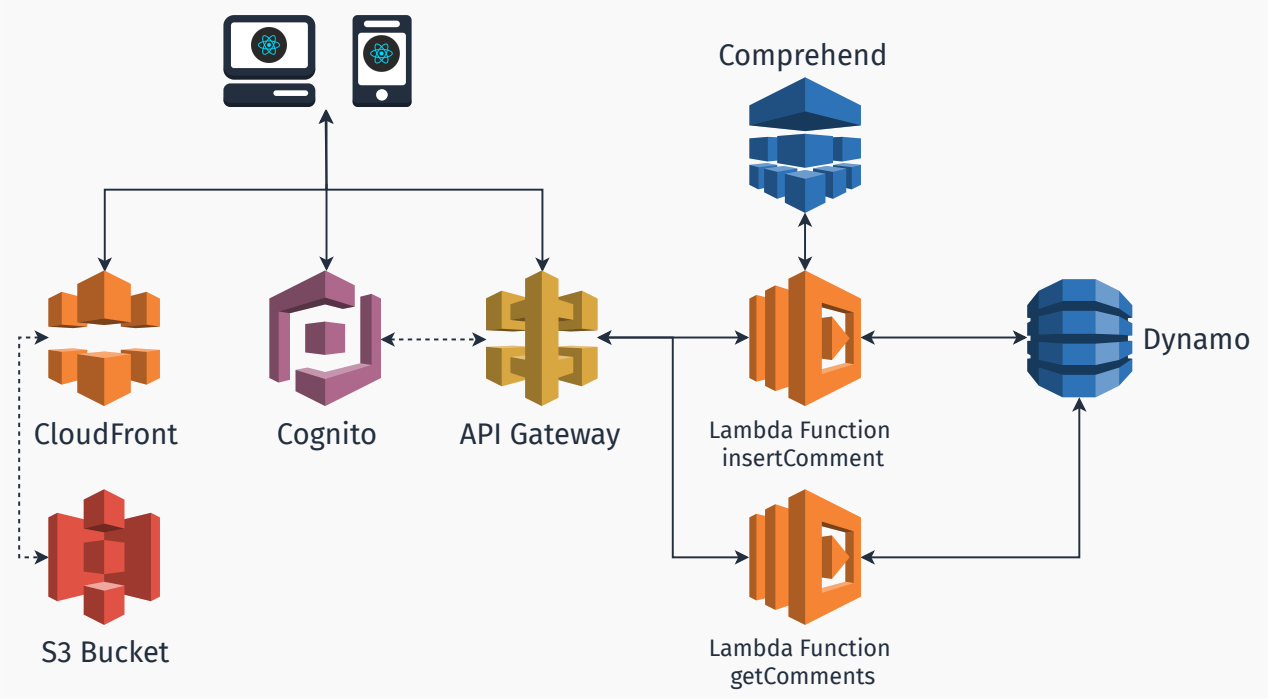
- Serverless Framework web
  - “toolkit to deploy and operate serverless architecture” .
  - Works with AWS, Google, Microsoft, IBM.
- APEX web
- AWS SAM web
  - Serverless Application Model
  - “Define serverless applications with a simple and clean syntax”
  - SAM Local: CLI tool for local development and testing

It’s demo time, again!

What we're going to do in this demo

- Remember the web application for this very talk we deployed earlier?
- Now we'll make it **serverless**, and add more features:
  - Sign-up and Authentication (Amazon Cognito)
  - Language detection and **sentiment analysis** on comments (Amazon Comprehend)
  - Deploy it on a global CDN to minimize latency (Amazon CloudFront)

Architecture



Skip tutorial

What you'll need

- An AWS account (a free one will suffice)
- `git` version control (recommended)
- Node.js
- Python (recommended)

Step 1: get the app

Clone the git repository Github

```
D:> git clone https://github.com/luistar/serverless-
    webapp.git serverless-webapp
```

Step 2: install dependencies

```
D:> cd serverless-webapp
D:\serverless-webapp> npm install
```

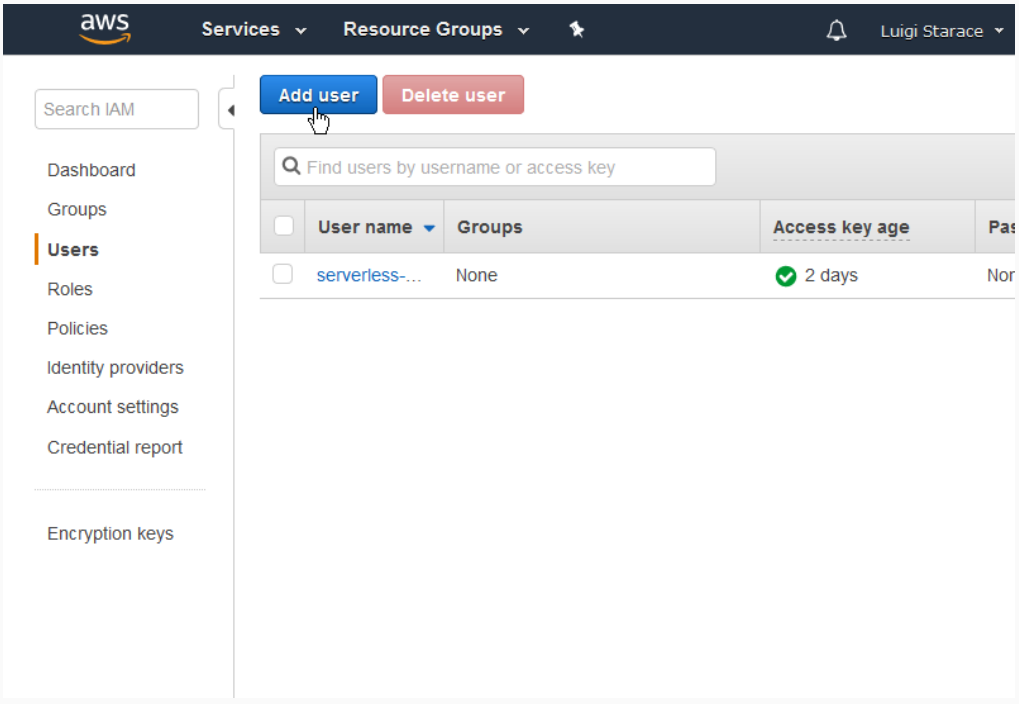

Step 3: install aws-mobile cli tool

```
D:\serverless-webapp> npm -g install awsmobile-cli
```

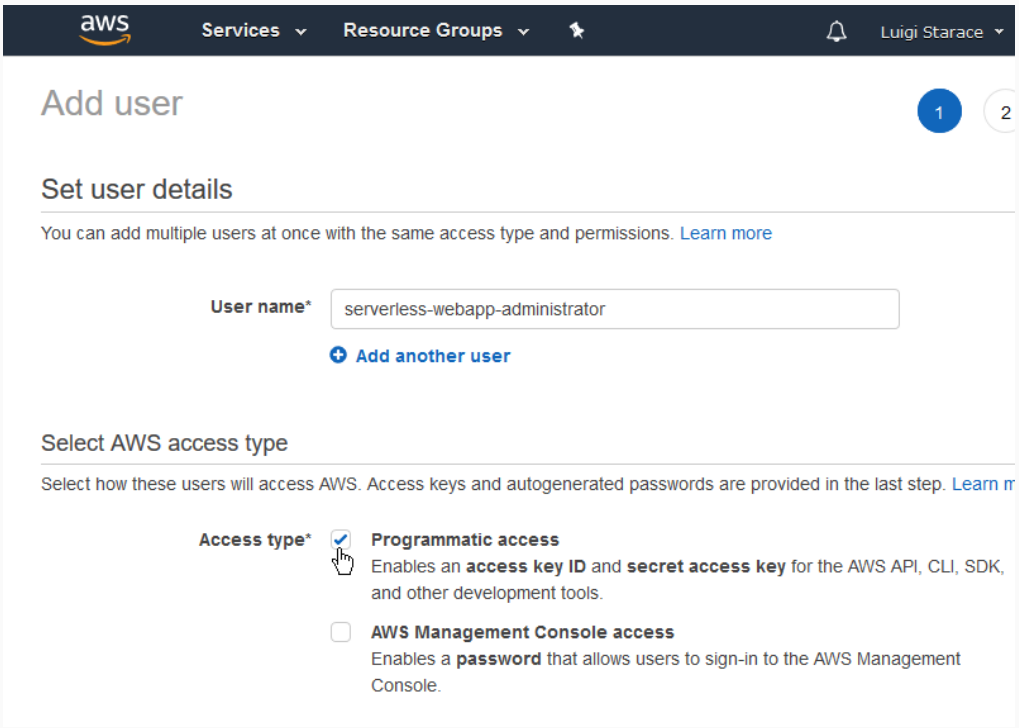
Step 4: create a new user on IAM

In order to use awsmobile-cli you're gonna need an access key id and a secret access key. If you don't already have one, go the IAM console and create a new user.

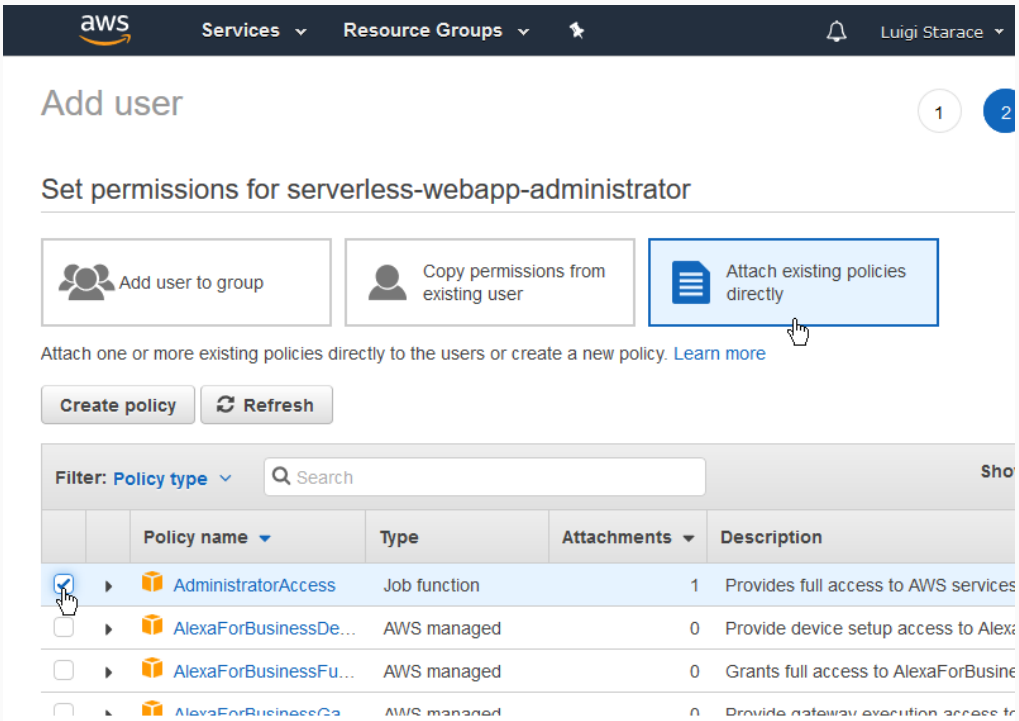
Step 4: create a new user on IAM



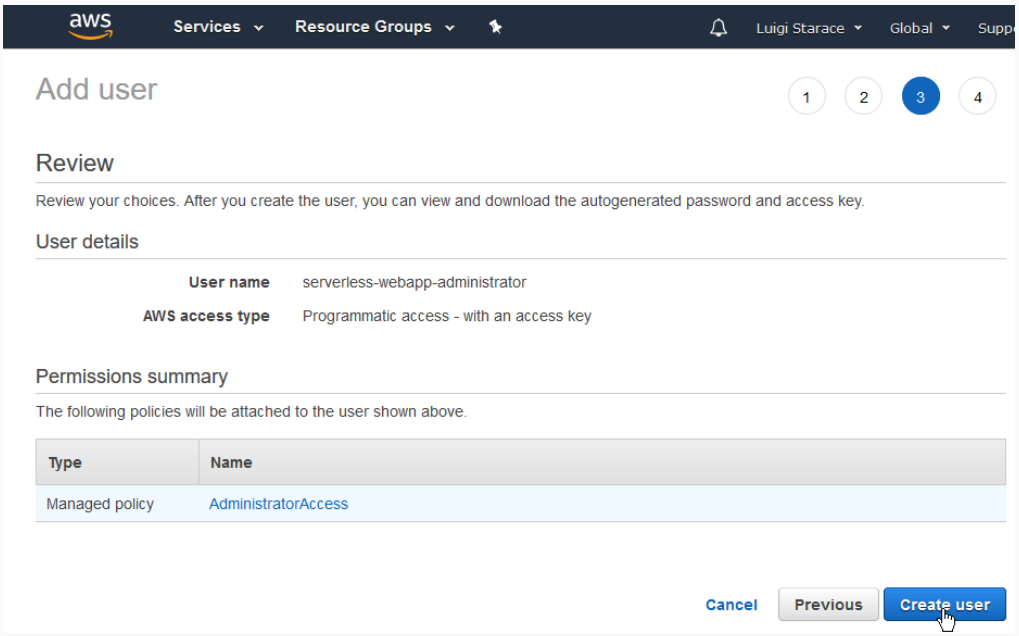
Step 4: Create a new user on IAM



Step 4: Create a new user on IAM



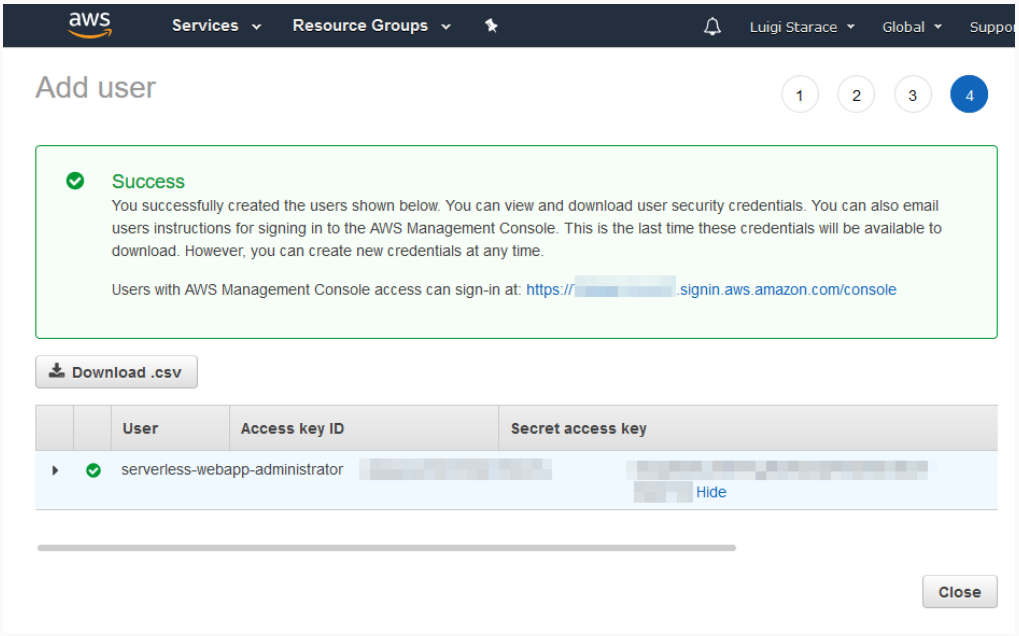
Step 4: create a new user on IAM



Step 4: create a new user on IAM

Be sure to write up your keys and to keep them safe!

Read more about security



Step 5: configure awsmobile-cli

Configure AWS Mobile CLI.

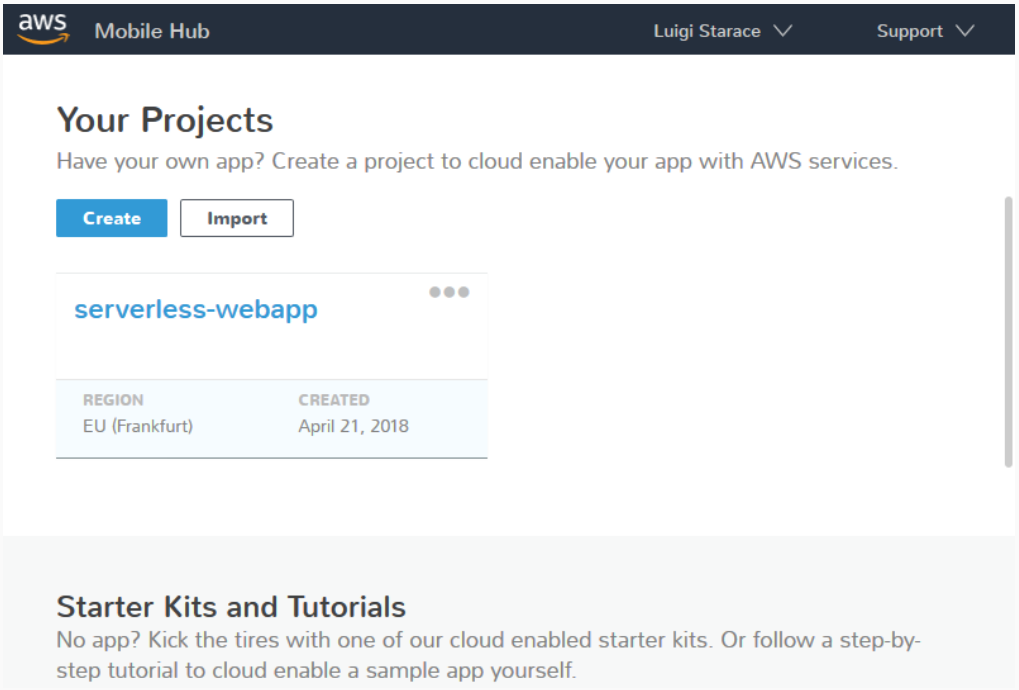
```
D:\serverless-webapp> awsmobile configure

configure aws
? accessKeyId:  <YOUR_ACCESS_KEY_ID>
? secretAccessKey:  <YOUR_SECRET_ACCESS_KEY>
? region:  eu-central-1
```


Step 6: initialize a new AWS Mobile project

```
D:\serverless-webapp> awsmobile init

Please tell us about your project:
? Where is your project's source directory:  src
? Where is your project's distribution directory that
    stores build artifacts:  build
? What is your project's build command:  npm.cmd run-
    script build
? What is your project's start command for local test
    run:  npm.cmd run-script start

? What awsmobile project name would you like to use:
    serverless-webapp
```

Step 6: initialize a new AWS Mobile project

Visit the AWS Mobile Console. Your newly created project should be waiting for you there.

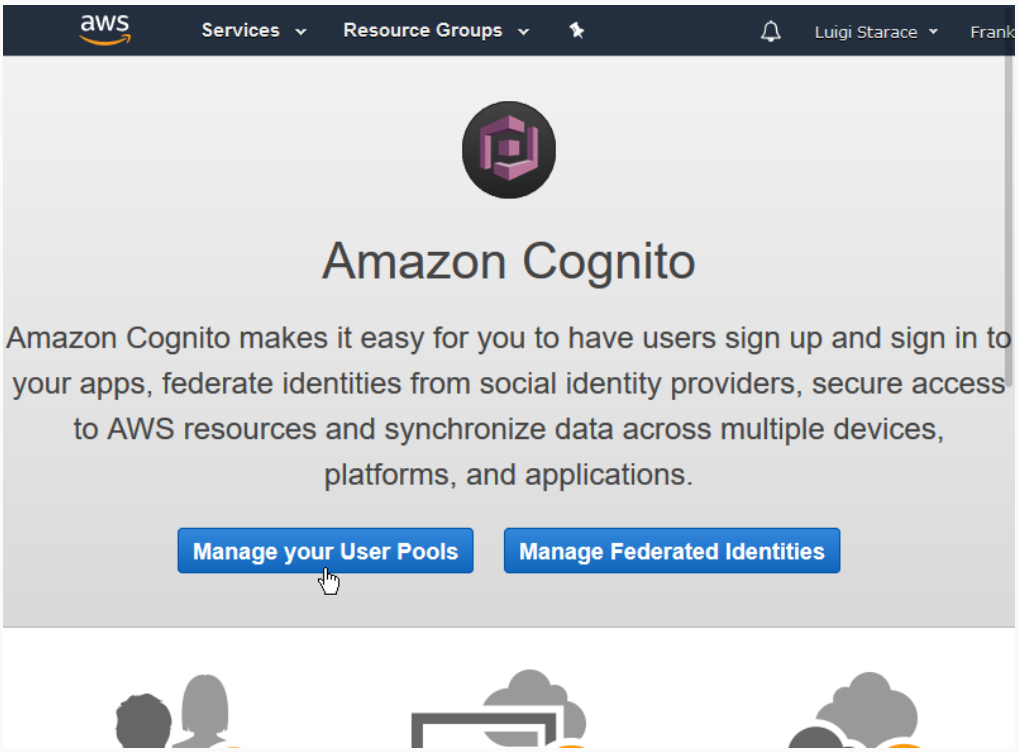


Step 7: create a Cognito User Pool

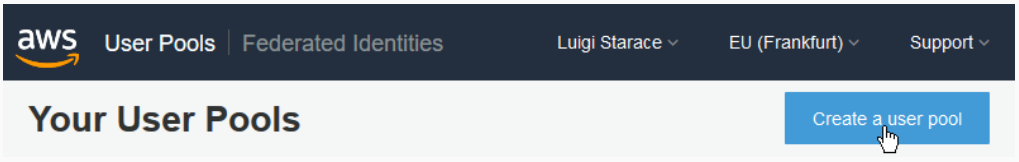
First we're gonna need a Cognito User Pool to authenticate our users. Let's create one.

Step 7: create a Cognito User Pool

Visit the Cognito Console.

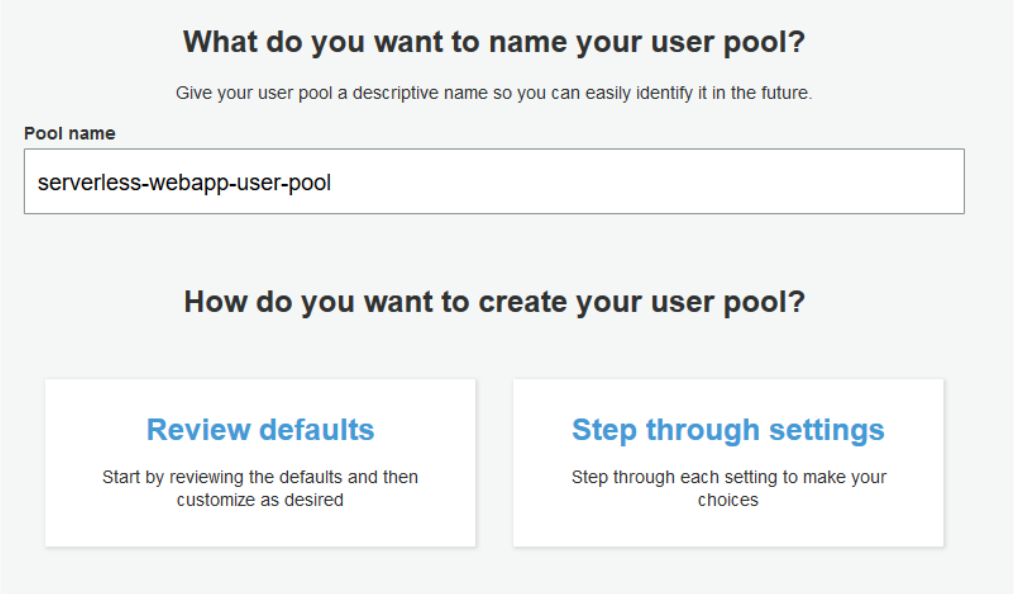


Step 7: create a Cognito User Pool



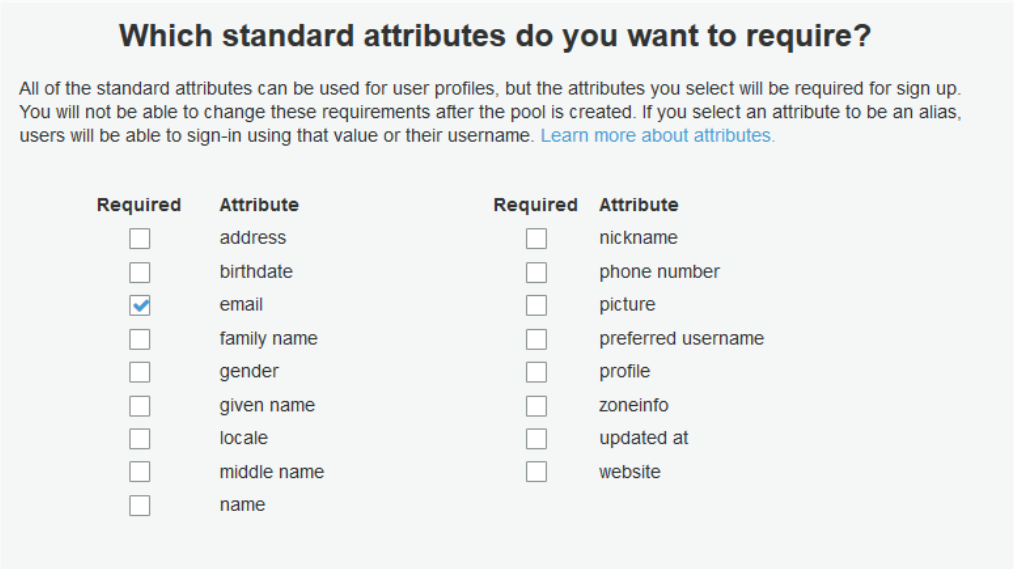
Step 7: create a Cognito User Pool

Insert a name for your user pool.



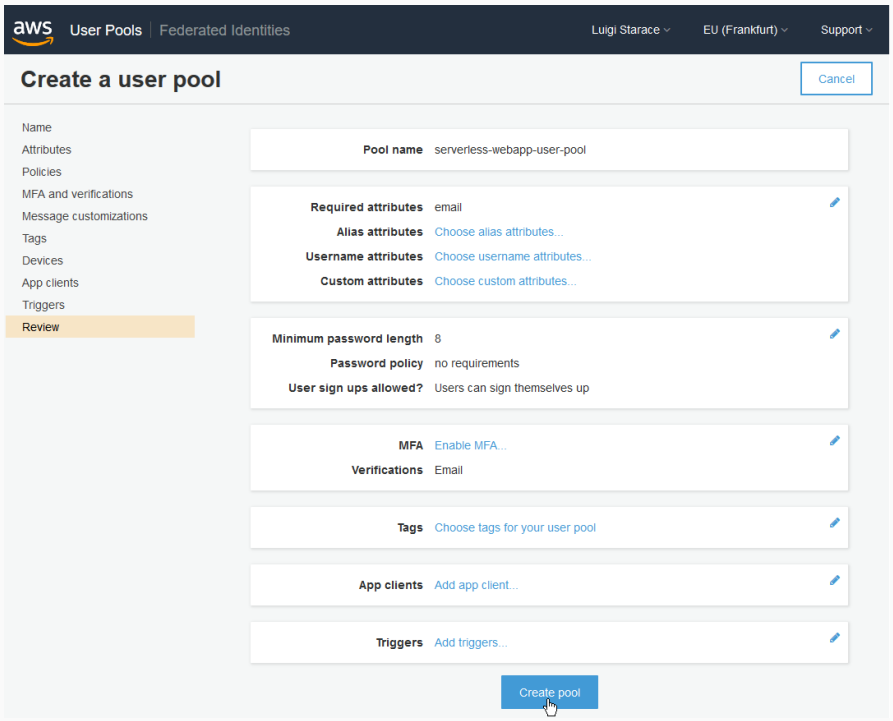
Step 7: create a Cognito User Pool

Make sure only an email is required.



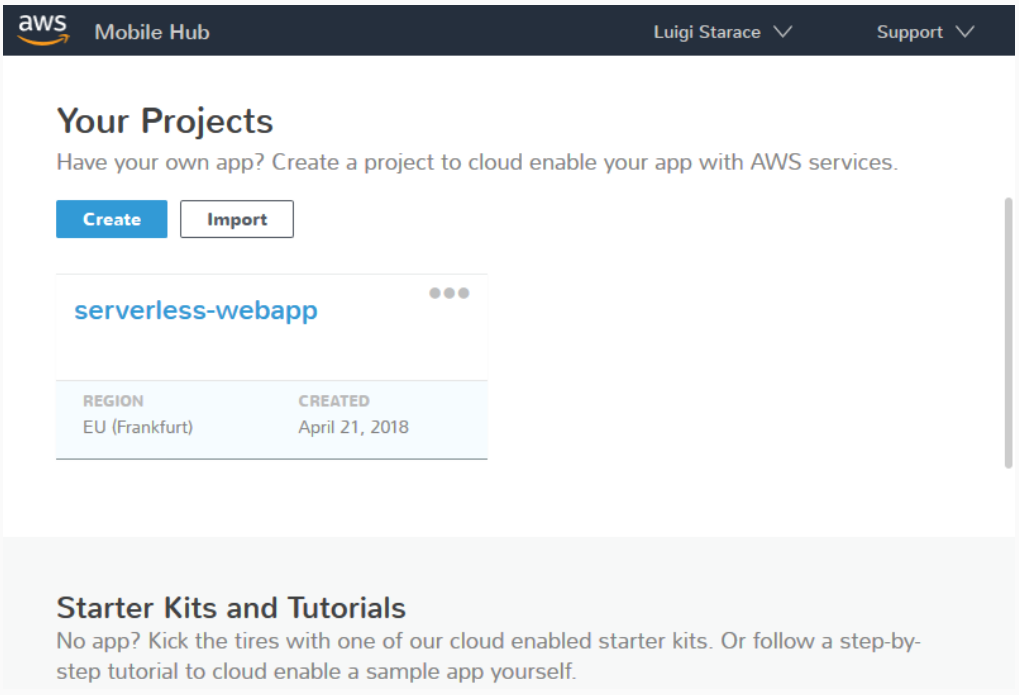
Step 7: Create a Cognito User Pool

Review your configuration and create the pool.



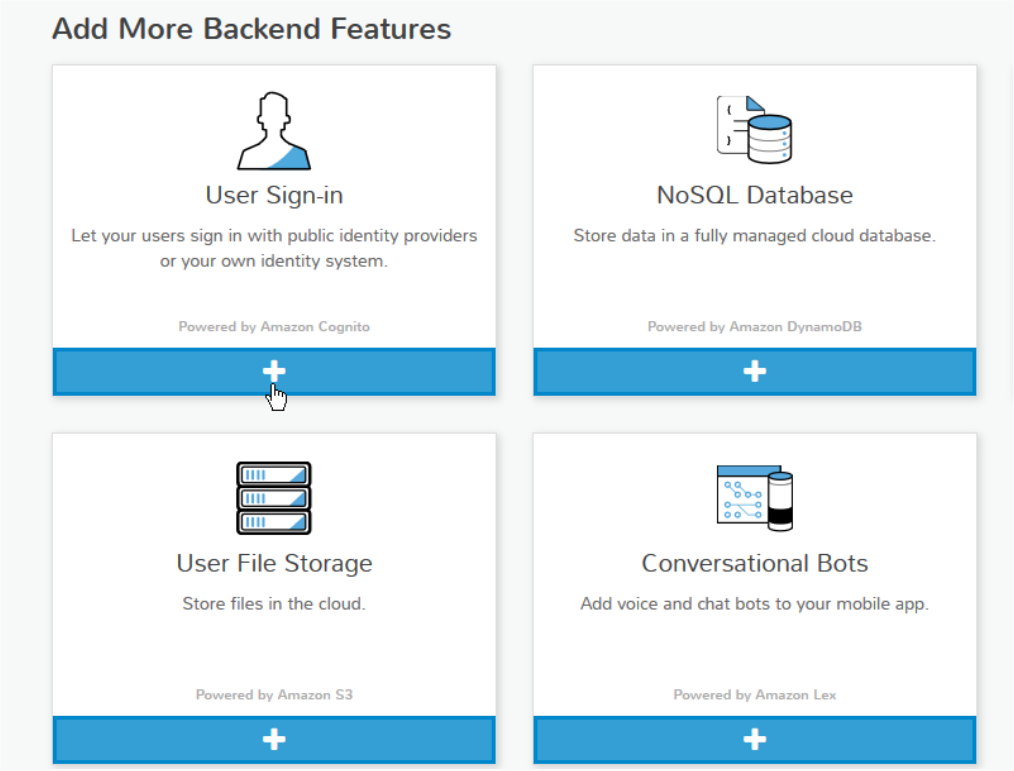
Step 8: Configure User Sign-In in the Mobile App Console

Return to the AWS Mobile Console and open your project.



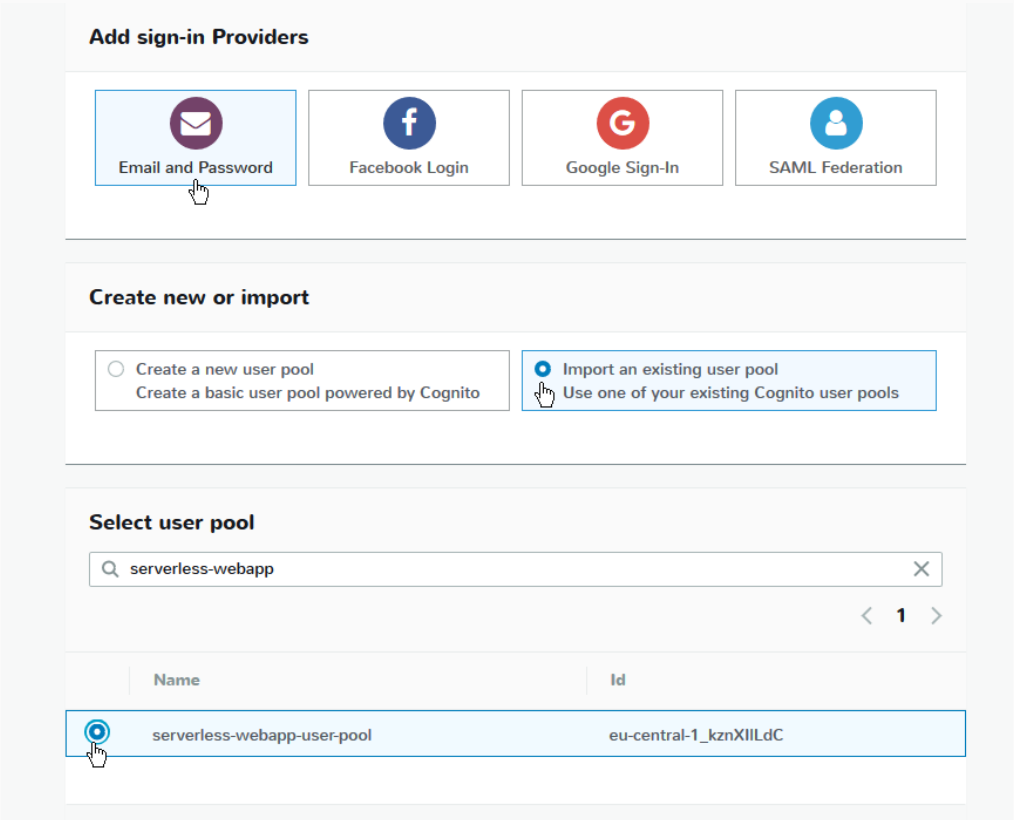
Step 8: Configure user sign-in in the mobile app console

Add user sign in to the project.



Step 8: Configure user sign-in in the mobile app console

Import your newly created user pool.



Step 8: configure user sign-in in the mobile app console

Pull your new project configuration with

```
D:\serverless-webapp> awsmobile pull
```

If you were to start the application locally with

```
D:\serverless-webapp> npm start
```

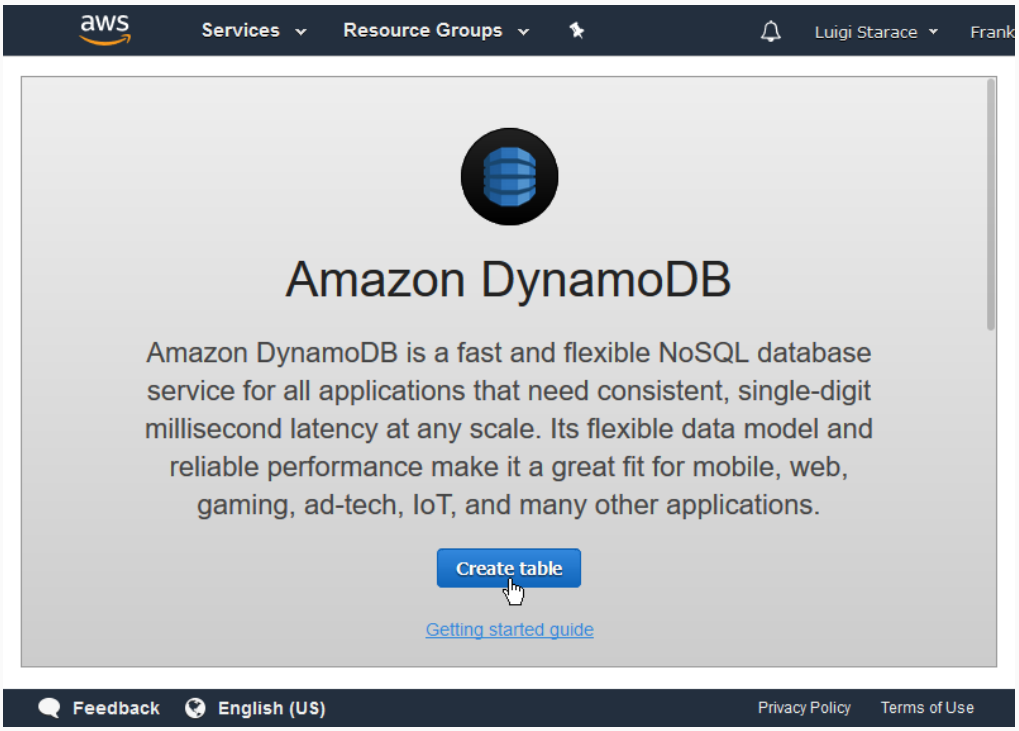
The authentication will now work.

Step 9: configure Amazon Dynamo Database

Next thing we're gonna need is a database to store the comments. In this tutorial we'll use the NoSQL database Amazon Dynamo.

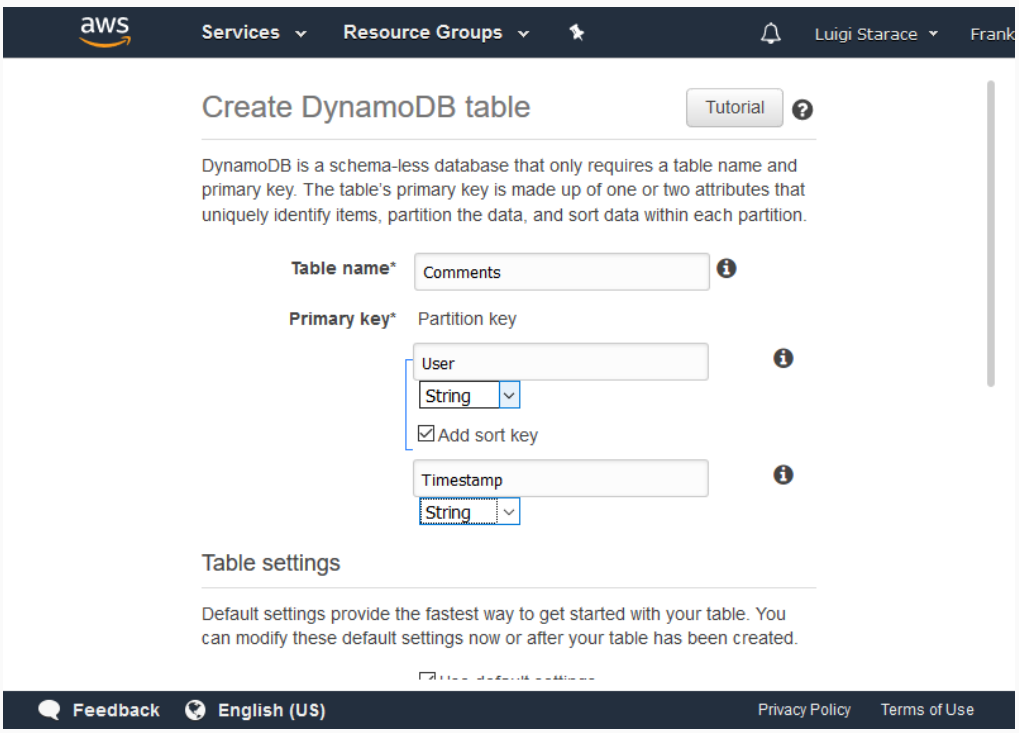
Step 9: configure Amazon Dynamo Database

Visit the Dynamo Dashboard.



Step 9: configure Amazon Dynamo Database

Create a new Comments table as shown. Leave other fields with their default values.

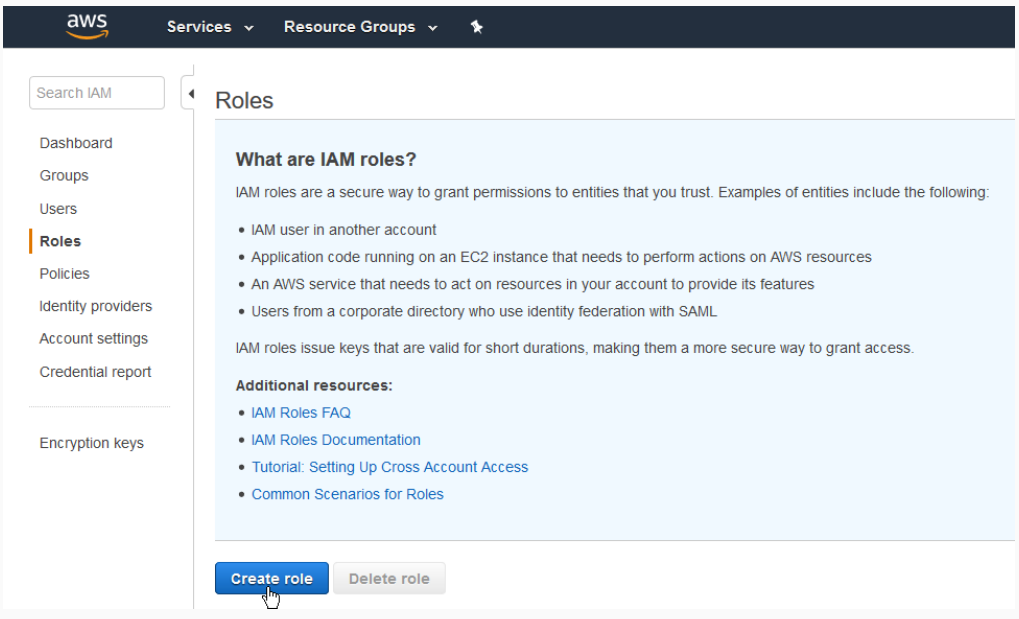


Step 10: create a role for the Lambda functions

Before we create our Lambda functions, let's create a new role defining the authorizations we want them to have.

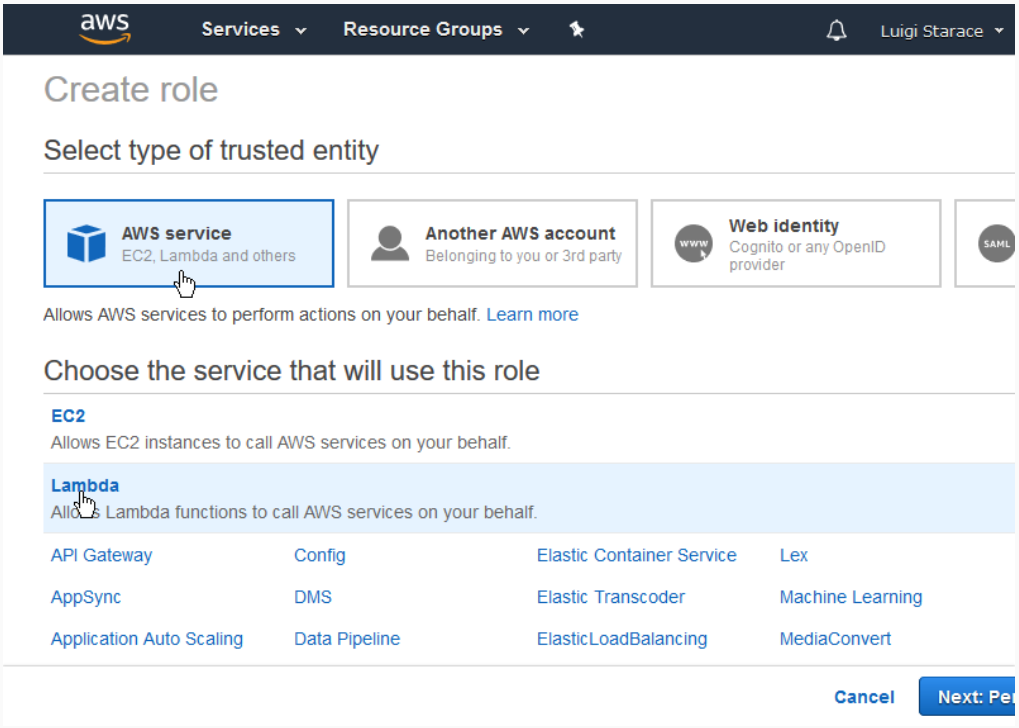
Step 10: create a role for the Lambda functions

Return to the IAM console and select the "role" tab, then the "create role" button.



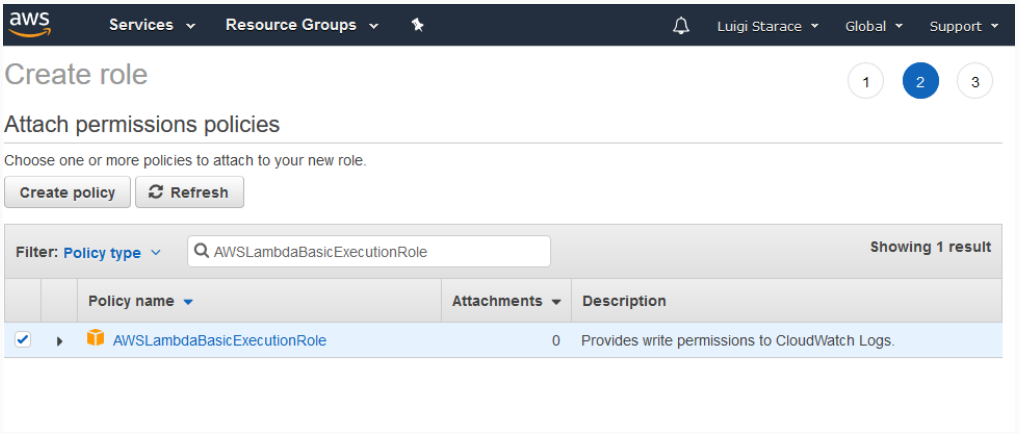
Step 10: create a role for the Lambda functions

Select “AWS Service” and “Lambda” in the wizard, as shown in the picture.



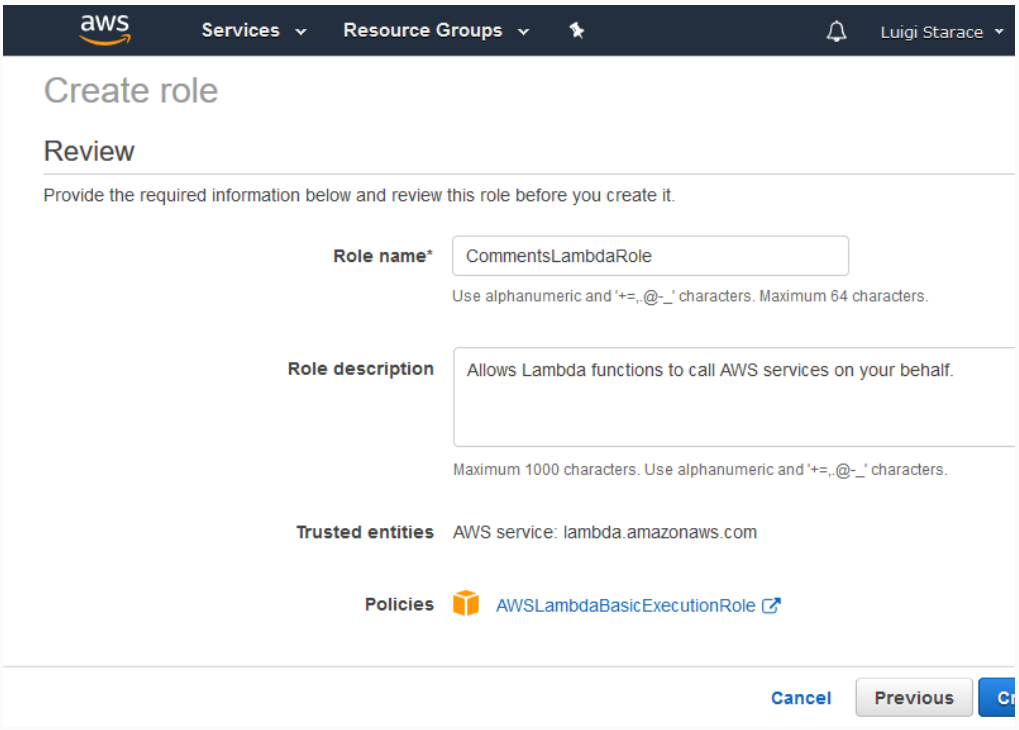
Step 10: create a role for the Lambda functions

Add the AWSLambdaBasicExecutionRole, as shown.



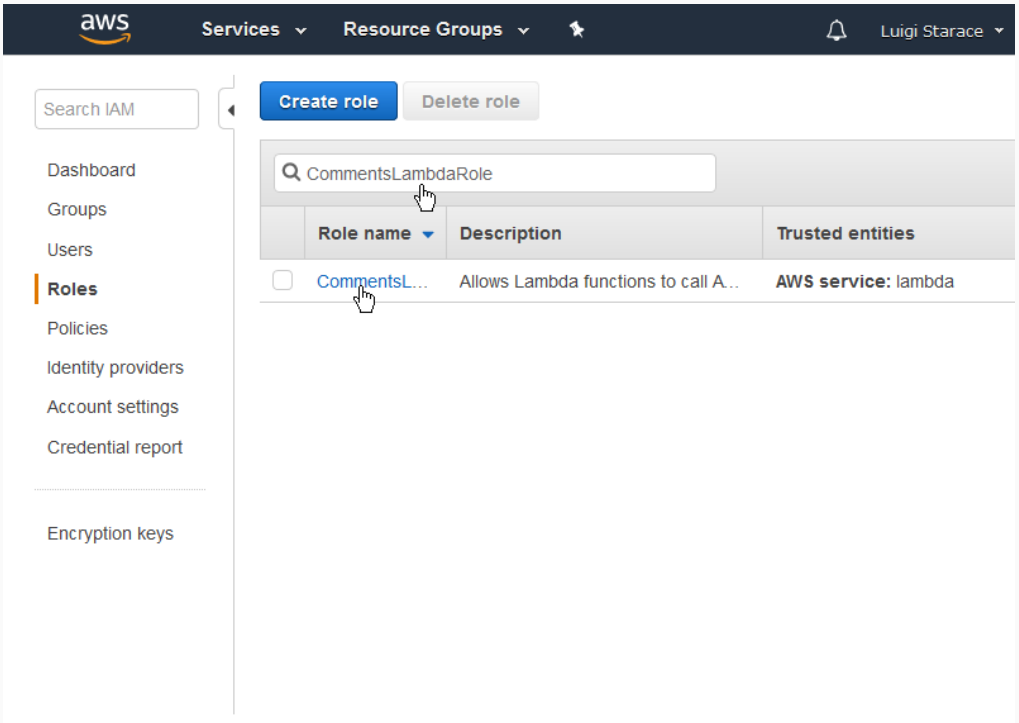
Step 10: create a role for the Lambda functions

Insert a name and a description and create the role.



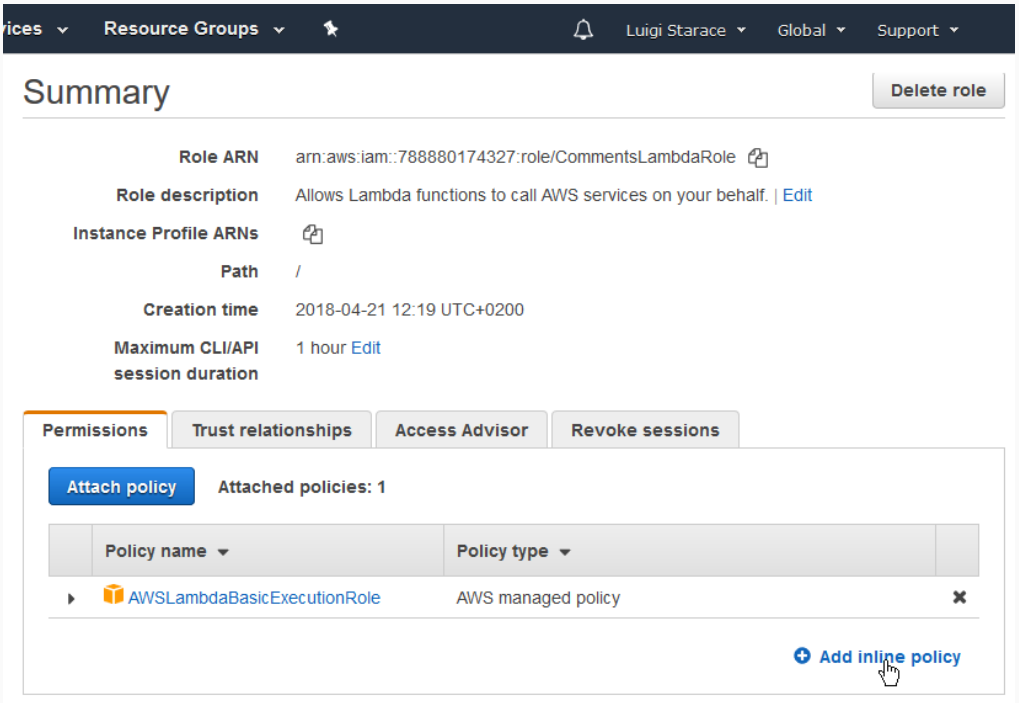
Step 10: create a role for the Lambda functions

Go back to the roles tab in the IAM Dashboard and select your newly created role.



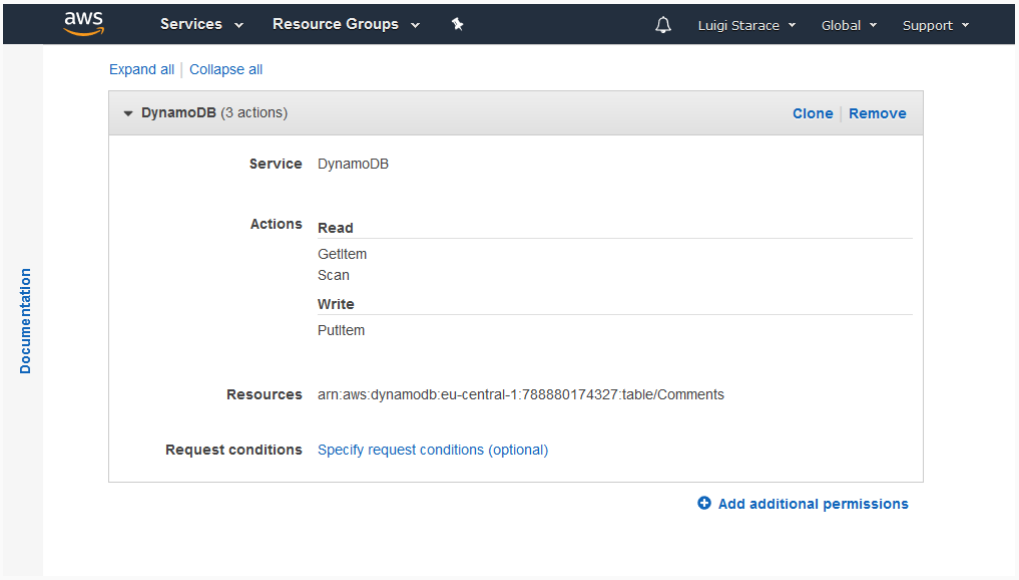
Step 10: create a role for the Lambda functions

Add inline policies to allow the role to access Dynamo DB and Comprehend.



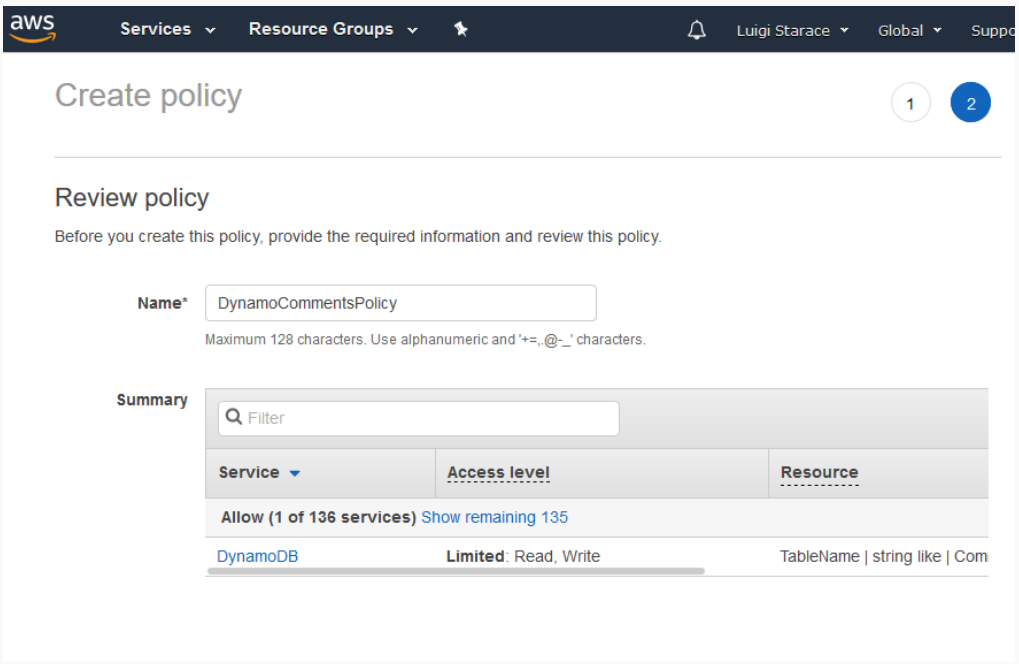
Step 10: create a role for the Lambda functions

Add an inline policy to allow this role to access Dynamo DB tables.



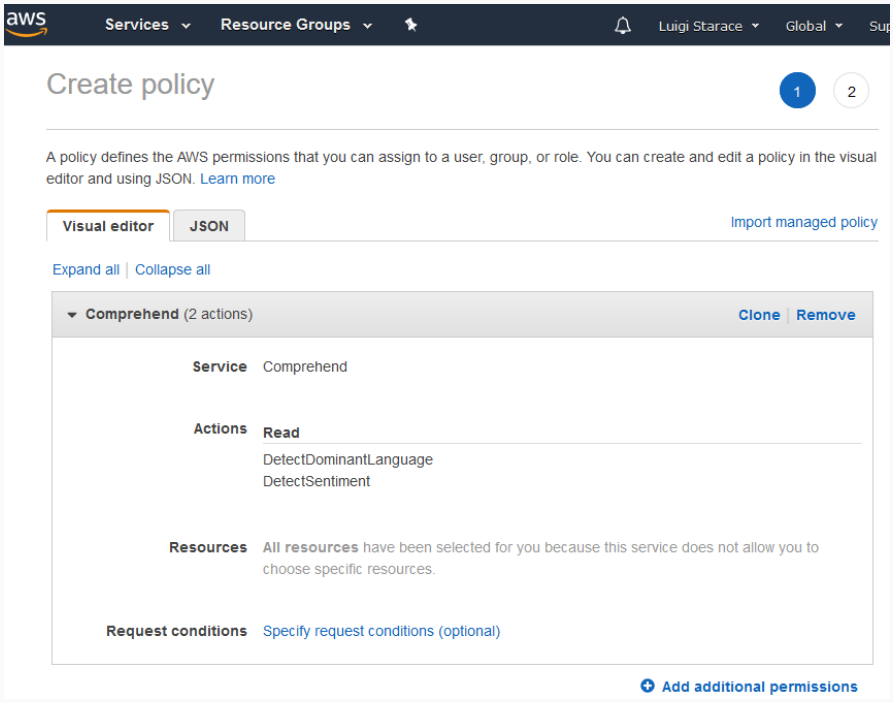
Step 10: create a role for the Lambda functions

Save the inline policy.



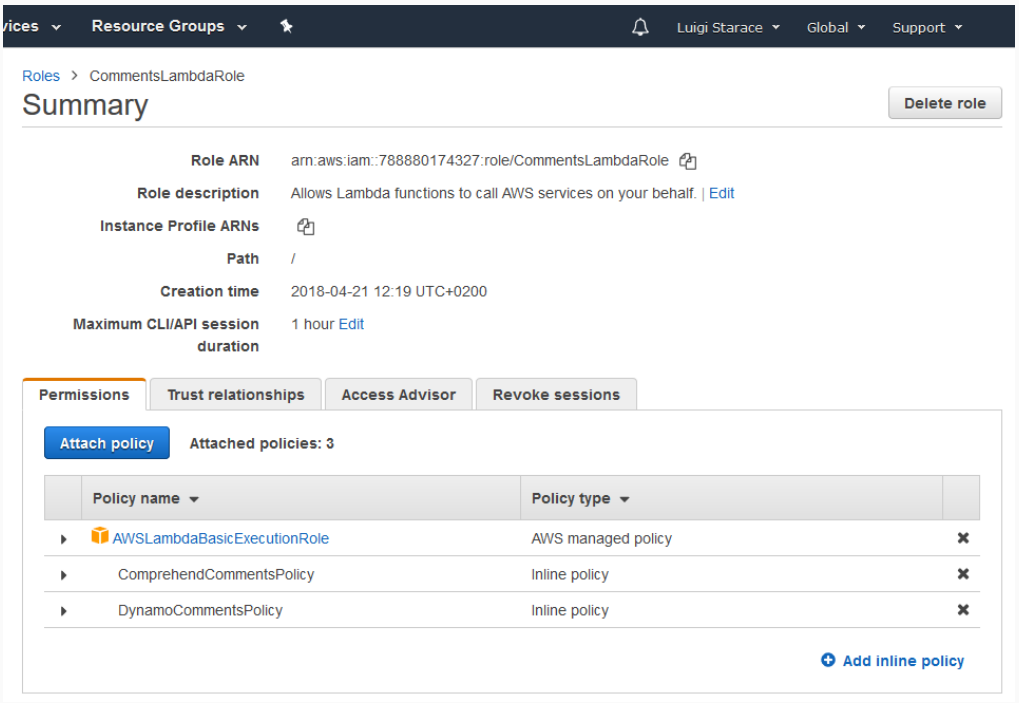
Step 10: create a role for the Lambda functions

Now add another inline policy to allow this role to access Comprehend's detect language and detect sentiment features.



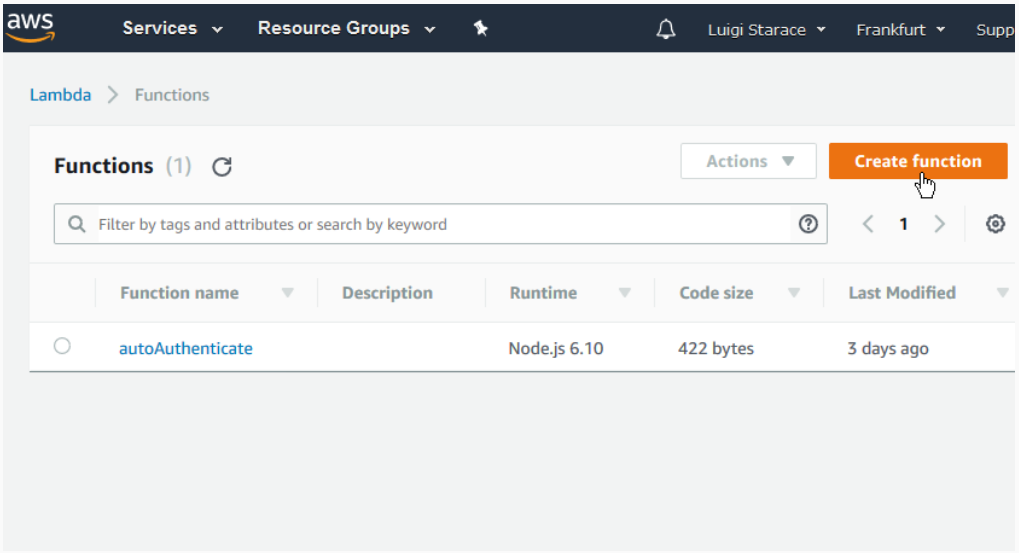
Step 10: create a role for the Lambda functions

Your role should look like this.



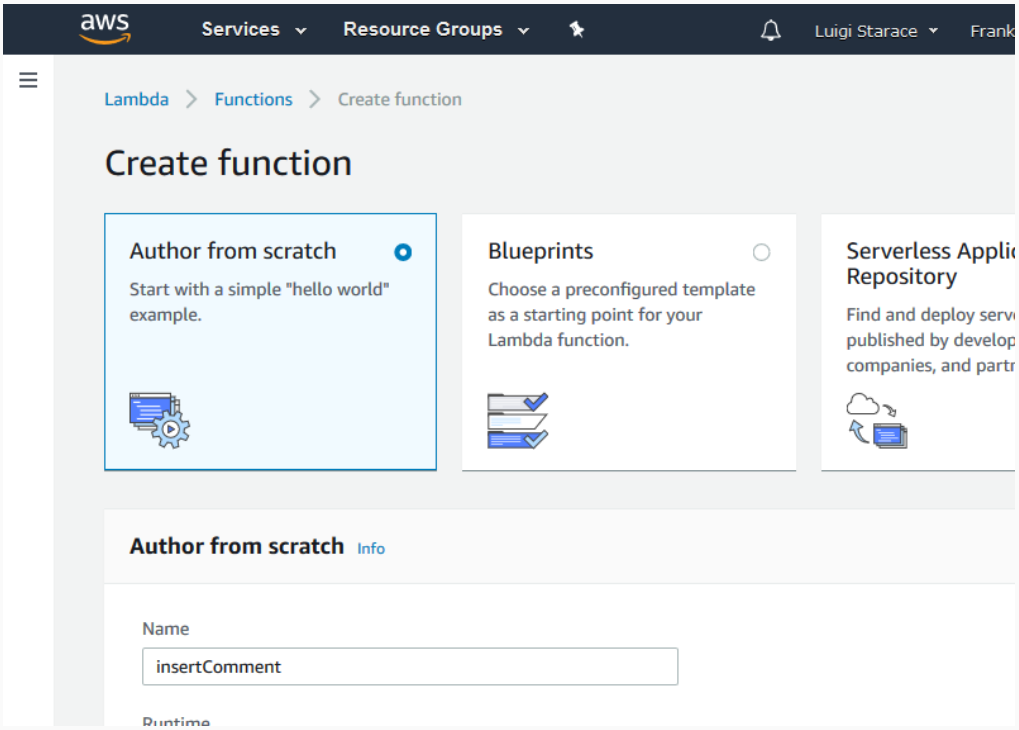
Step 11: create the Lambda functions

Go to the Lambda Dashboard and click on the "Create function" button.



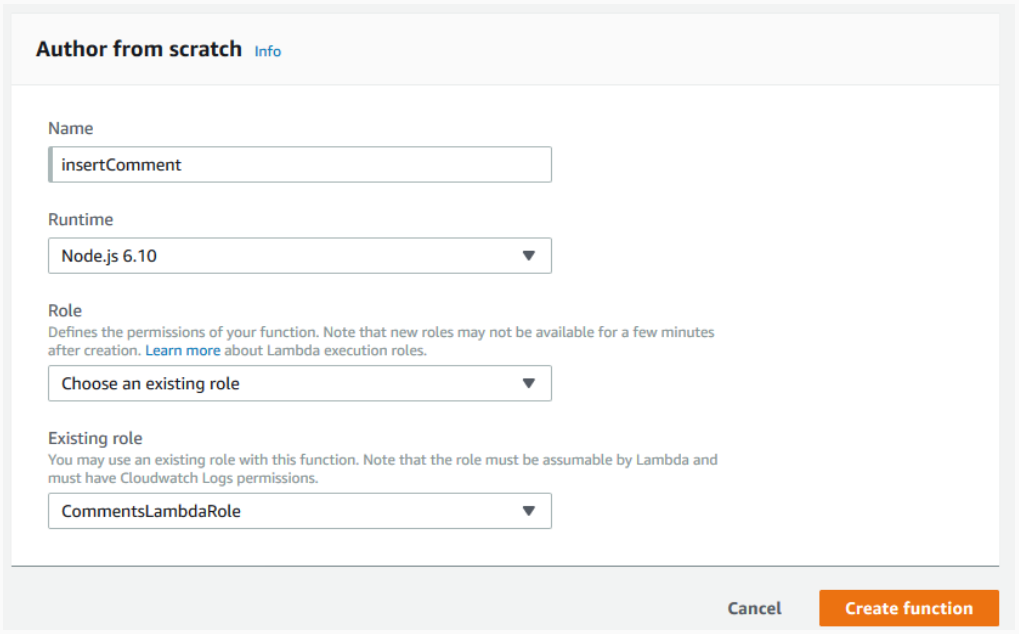
Step 11: create the Lambda functions

Select “Author from scratch”



Step 11: create the Lambda functions

Name the function `insertComment` and select Node.js 6.10 as the runtime and the role we created earlies as the role.



Step 11: create the Lambda functions

Insert the code provided in the `lambda/insertComment.js` file in the next screen, then save the lambda function.

Step 11: create the Lambda functions

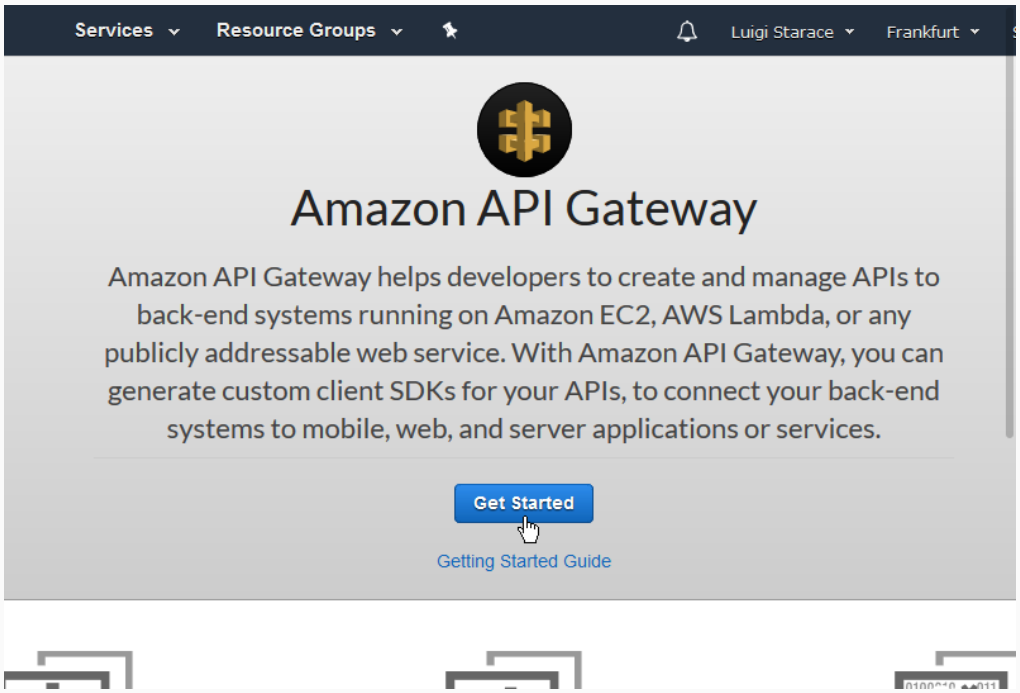
Proceed similarly and create the getComments Lambda function.

Step 12: create the APIs to expose the Lambda functions

Once we have our Lambda functions, let's hook 'em up with an API our web app can rely upon.

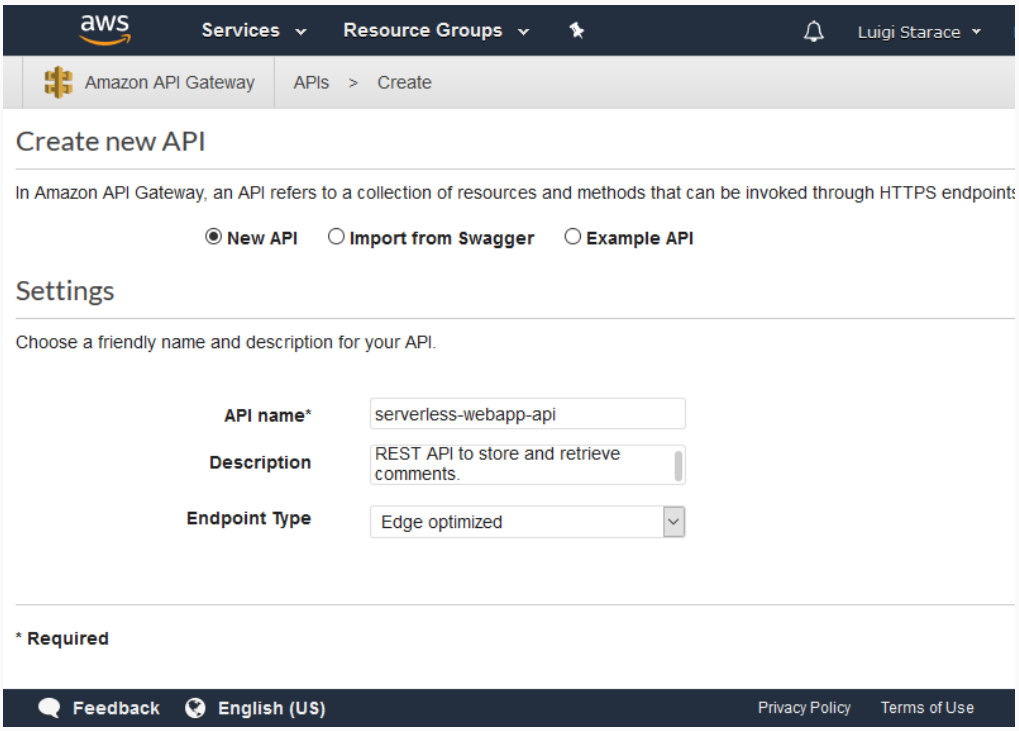
Step 12: create the APIs to expose the Lambda functions

Visit the API Gateway Dashboard



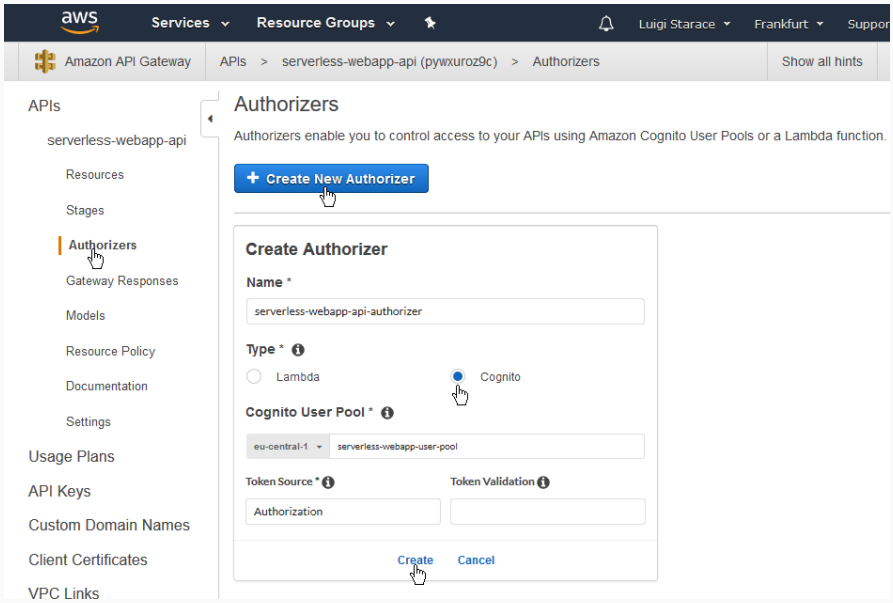
Step 12: create the APIs to expose the Lambda functions

Create a new API and select a name and a description.



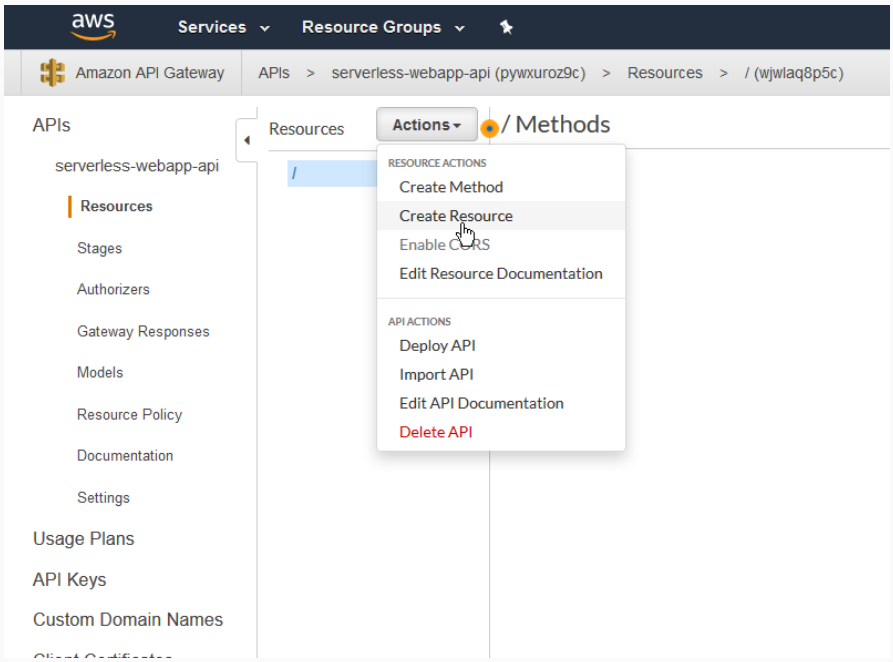
Step 12: create the APIs to expose the Lambda functions

Select the Authorizers tab and create a new Authorizer for your API. Give it a name, select the user pool we created earlier, and enter "Authorization" in the "Token Source" field.



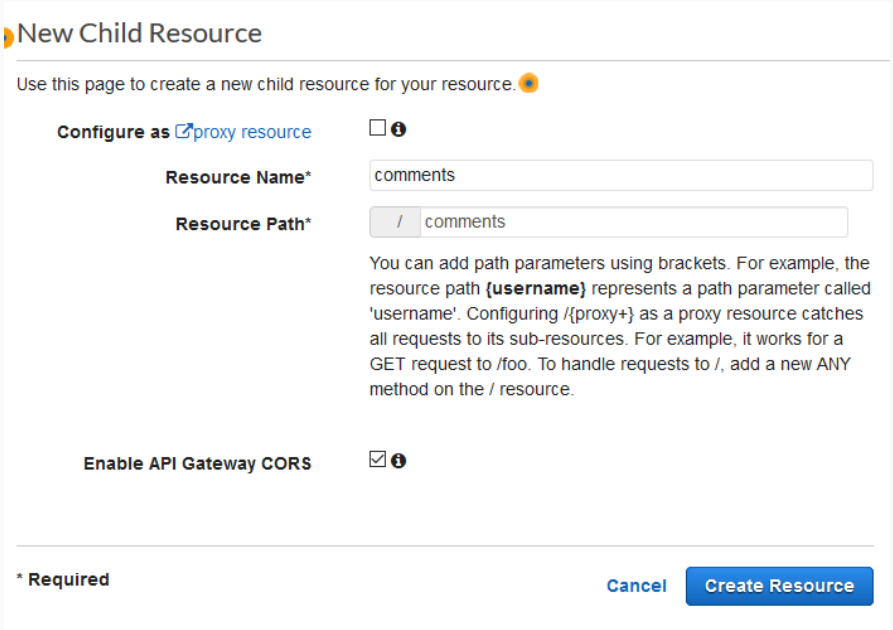
Step 12: create the APIs to expose the Lambda functions

Select the resources tab and create a new Resource



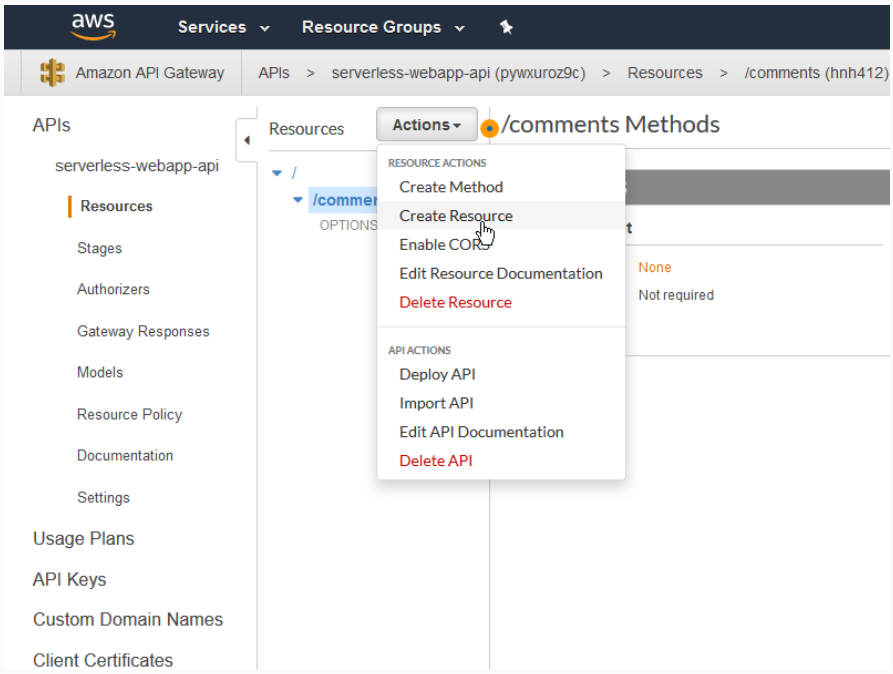
Step 12: create the APIs to expose the Lambda functions

Name the resource `comments`, enable CORS and continue.



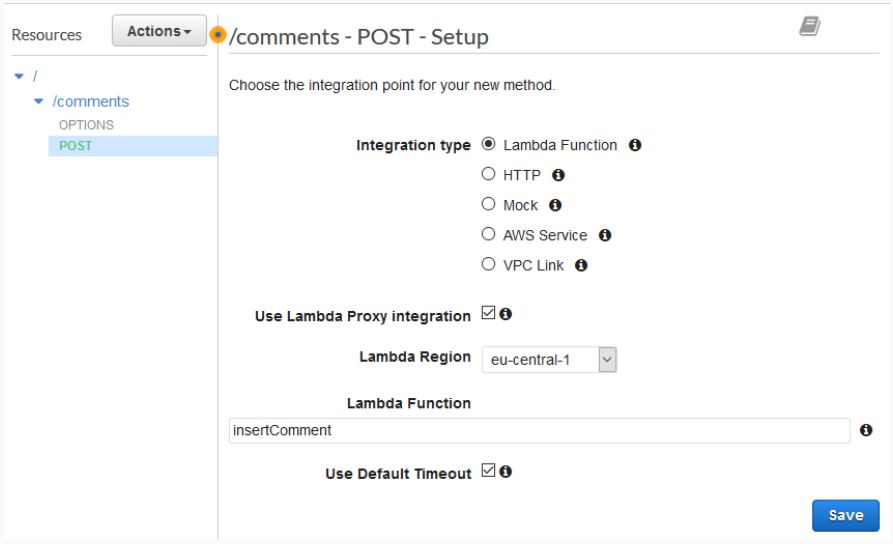
Step 12: create the APIs to expose the Lambda functions

Select the comments resource and create a new POST method.



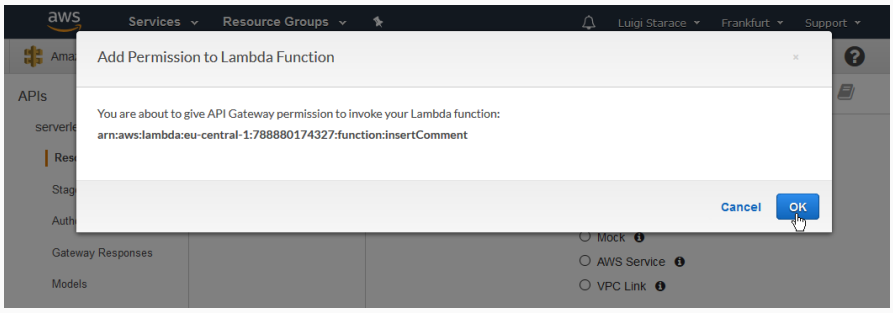
Step 12: create the APIs to expose the Lambda functions

As shown, select the insertComment Lambda function you created earlier as the integration point.



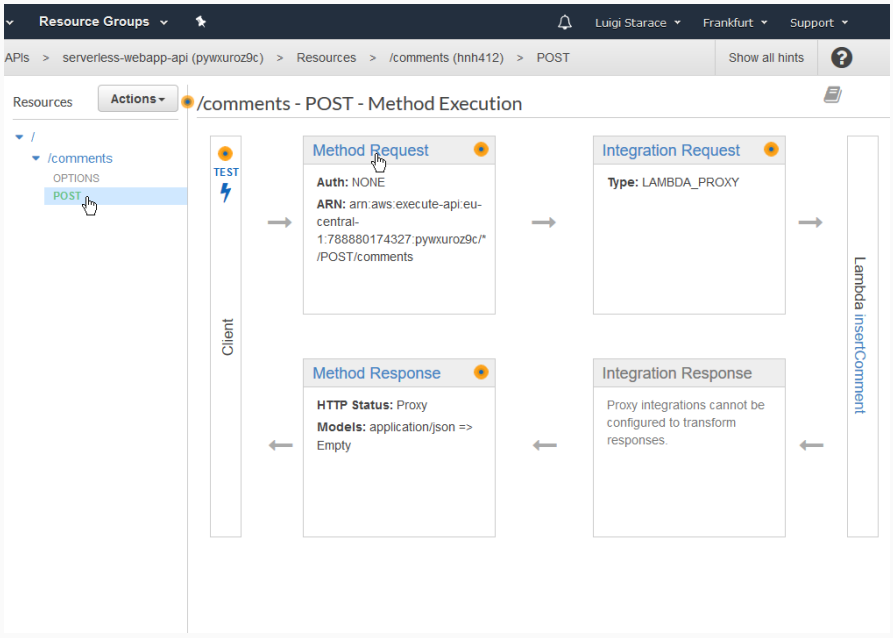
Step 12: create the APIs to expose the Lambda functions

Give API Gateway the permission to invoke the Lambda function



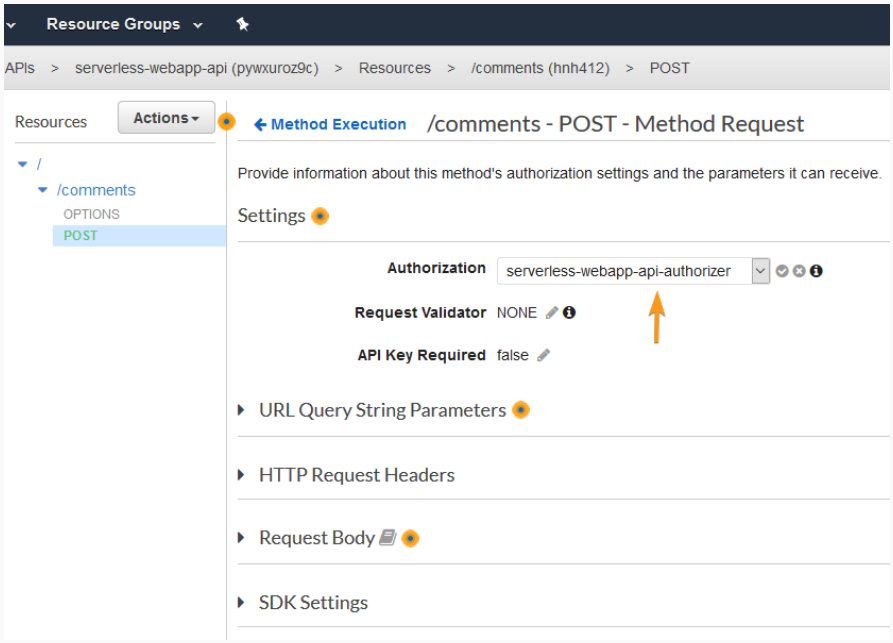
Step 12: create the APIs to expose the Lambda functions

Select the POST method on the comments resource, then select the Method Request card.



Step 12: create the APIs to expose the Lambda functions

Select the authorizer you created earlies for the Authorization field in the Settings section.

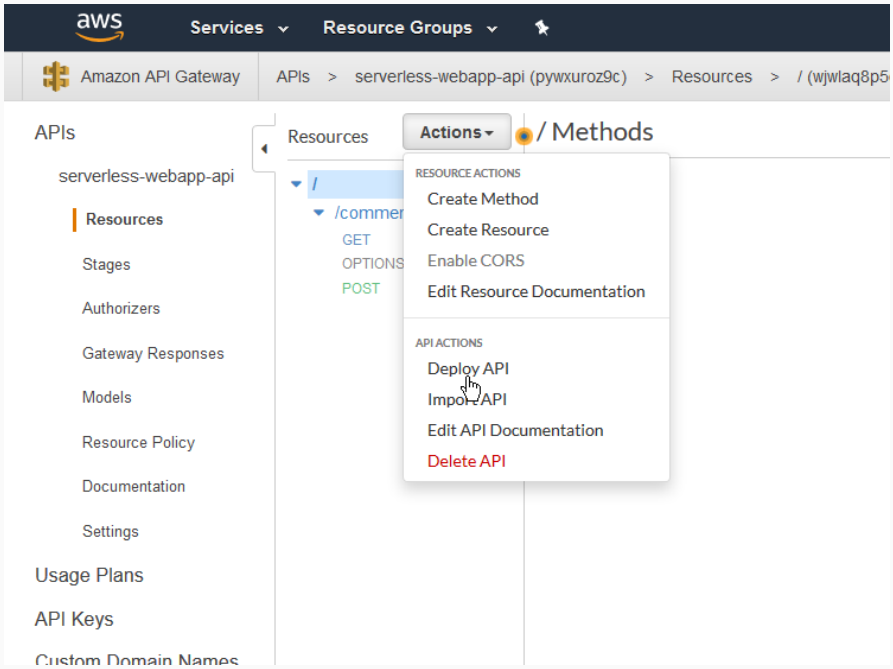


Step 12: create the APIs to expose the Lambda functions

Proceed similarly to hook up the GET method with the getComments Lambda function. This time authorization is not needed. We want non-authenticated users to be able to fetch the comments.

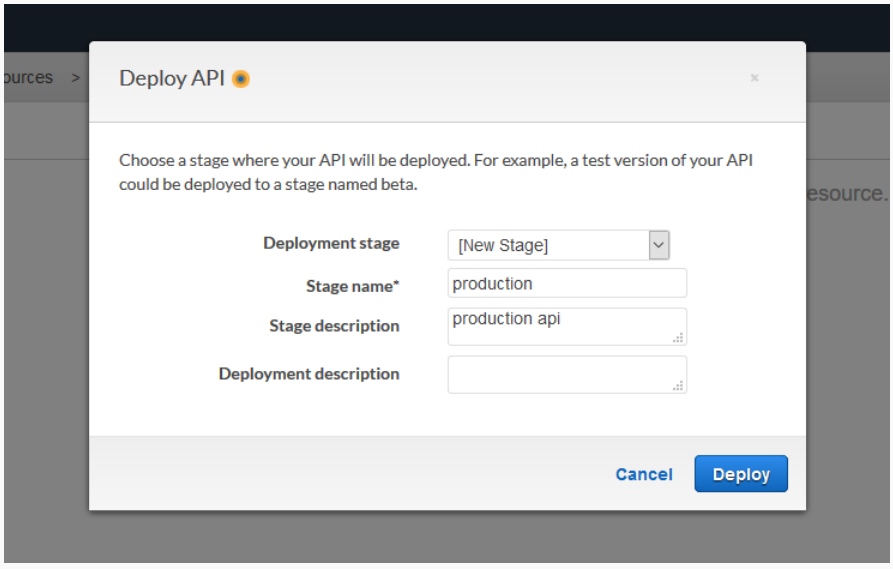
Step 12: create the APIs to expose the Lambda functions

Once you are done setting up the GET method, select the root resource, then Deploy API.



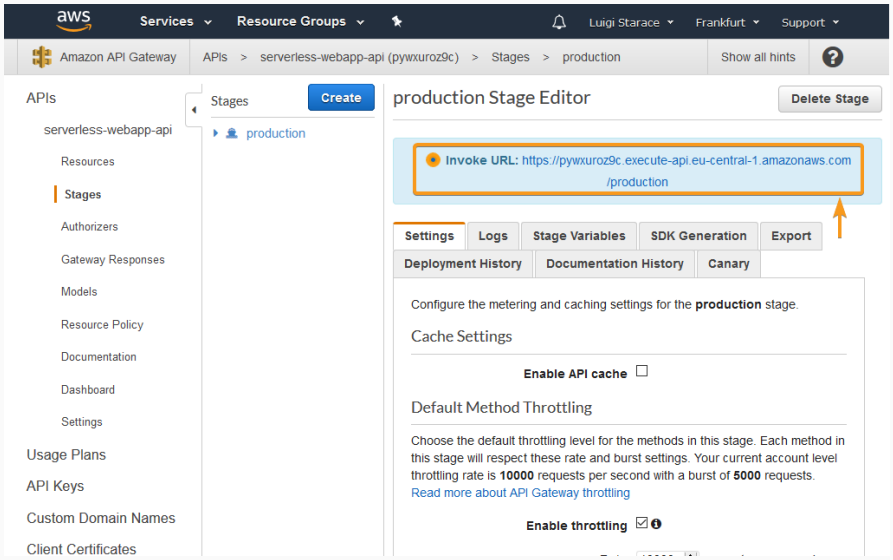
Step 12: create the APIs to expose the Lambda functions

Insert deployment stage informations and deploy.



Step 12: create the APIs to expose the Lambda functions

Select the stages tab and note the invoke url.



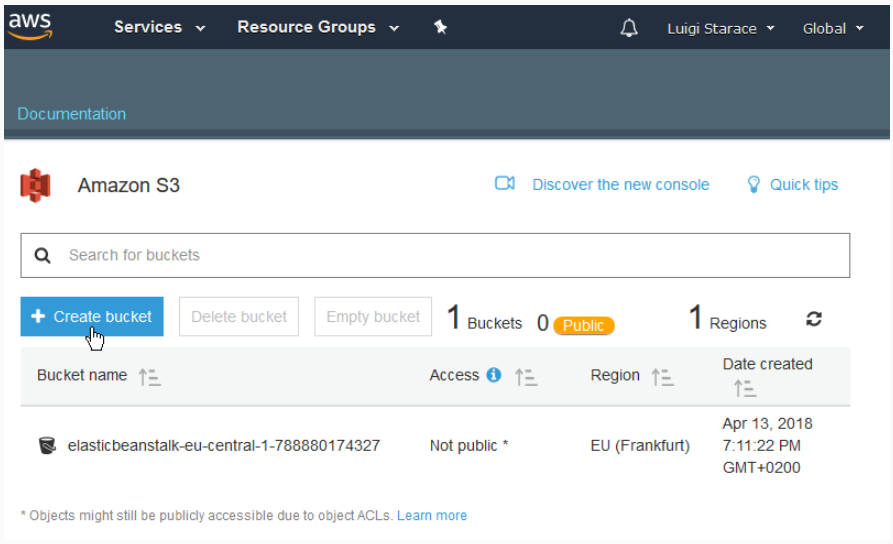
Step 12: create the APIs to expose the Lambda functions

Change the CommentsAPI class accordingly in `src/API/CommentsAPI.js`.

```
class CommentsAPI {

    constructor(){
        this.endpoint = '<YOUR_INVOKE_URL_HERE>';
    }

    // ...

}
```

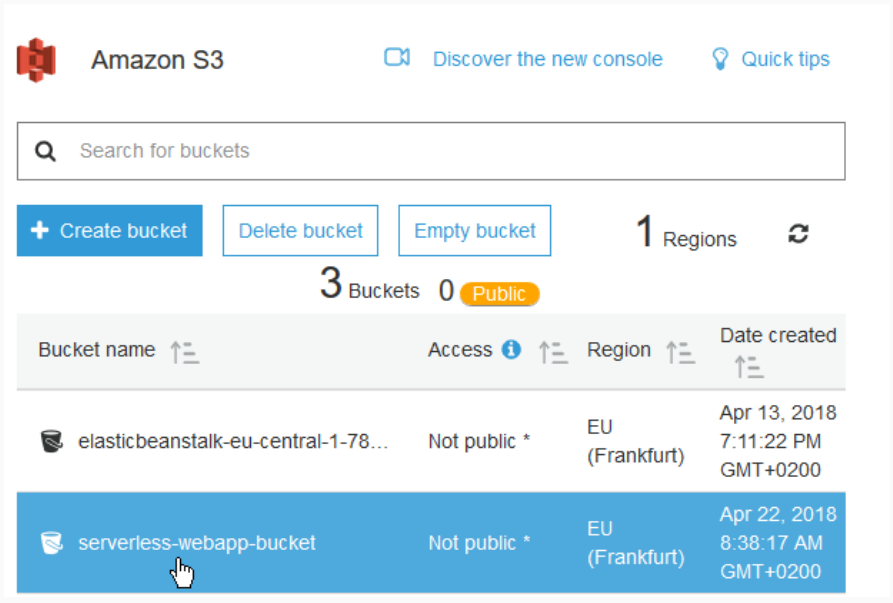
Step 13: Host the static files with S3

Visit the S3 Dashboard and create a new bucket.



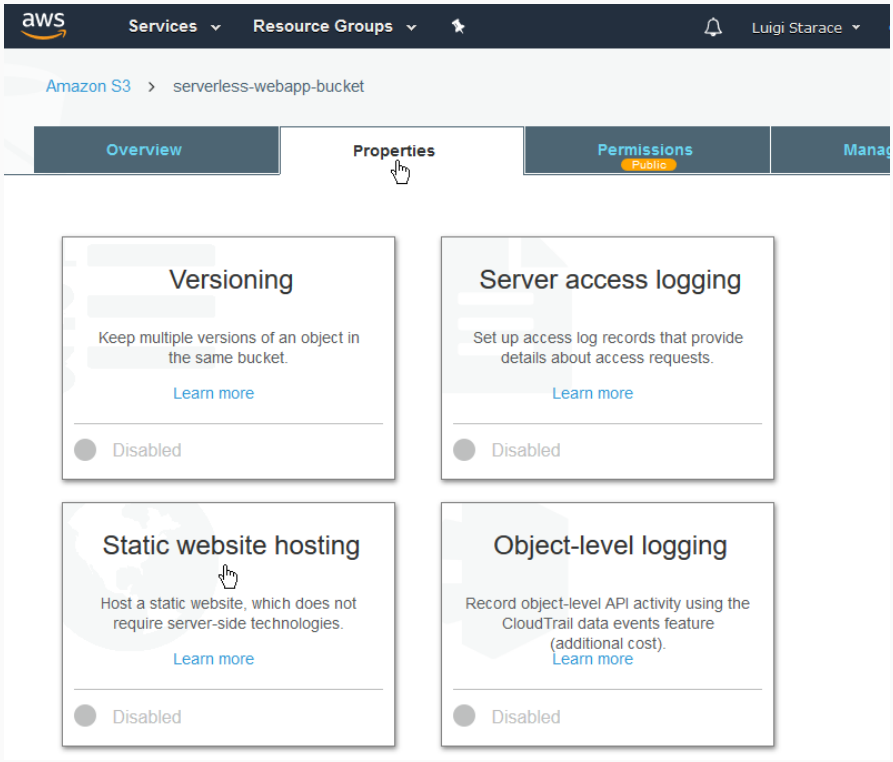
Step 13: Host the static files with S3

Select your newly created S3 bucket.



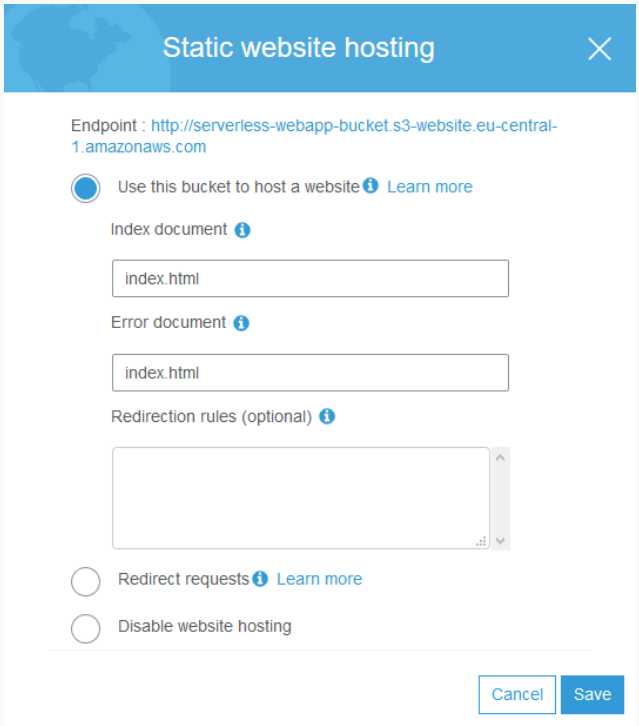
Step 13: Host the static files with S3

Under the properties tab, select the Static website hosting card.



Step 13: Host the static files with S3

Fill the form as shown in the picture. Note the Endpoint, as it will be the URL of the website!



Step 13: host the static files with S3

Now we'll show how to upload the static website via AWS CLI. This operation can also be performed via the web interface of the bucket.

Step 13: host the static files with S3

Install AWS CLI

```
D:\serverless-webapp> pip install awscli --upgrade
```

Then configure it

```
D:\serverless-webapp> aws configure
AWS Access Key ID [None]: <YOUR_ACCESS_KEY_ID>
AWS Secret Access Key [None]: <YOUR_SECRET_ACCESS_KEY>
Default region name [None]: eu-central-1
Default output format [None]: json
```

Step 13: Host the static files with S3

Build the website

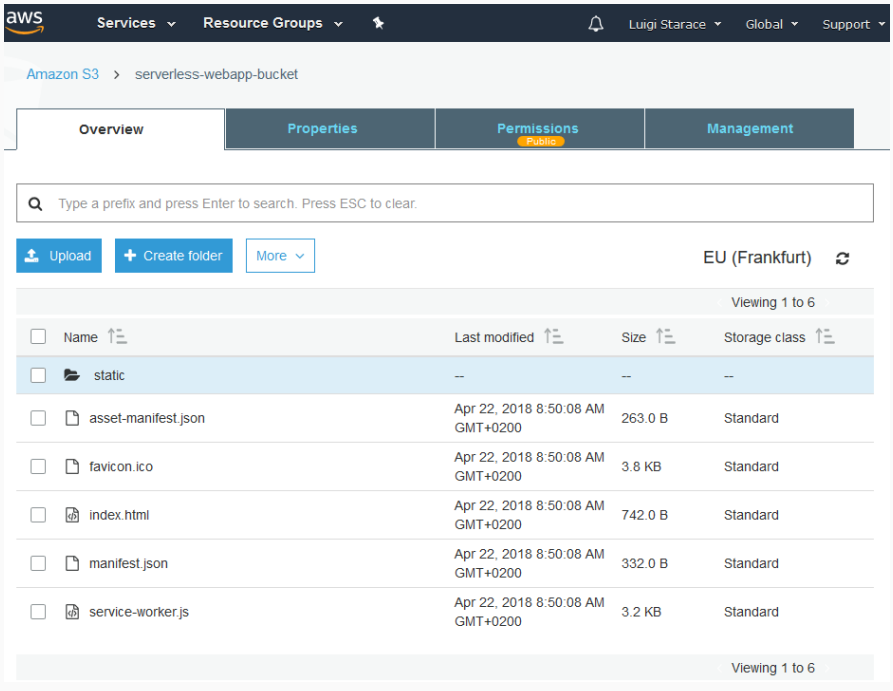
```
D:\serverless-webapp> npm run build-css
D:\serverless-webapp> npm run build
```

Then upload the files with

```
D:\serverless-webapp> aws s3 sync ./build s3://
    serverless-webapp-bucket --acl public-read
```

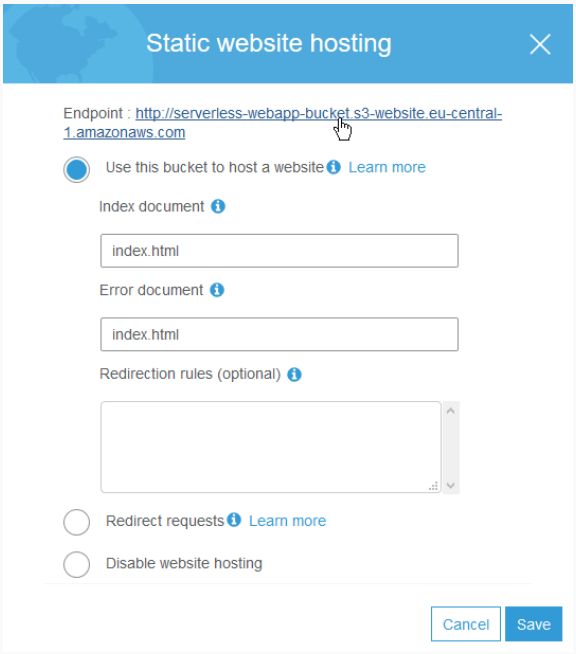

Step 13: Host the static files with S3

After the upload is done, your bucket should look like this.



Step 13: Host the static files with S3

Visit the Static website hosting card again under the properties tab, then click con the endpoint URL.



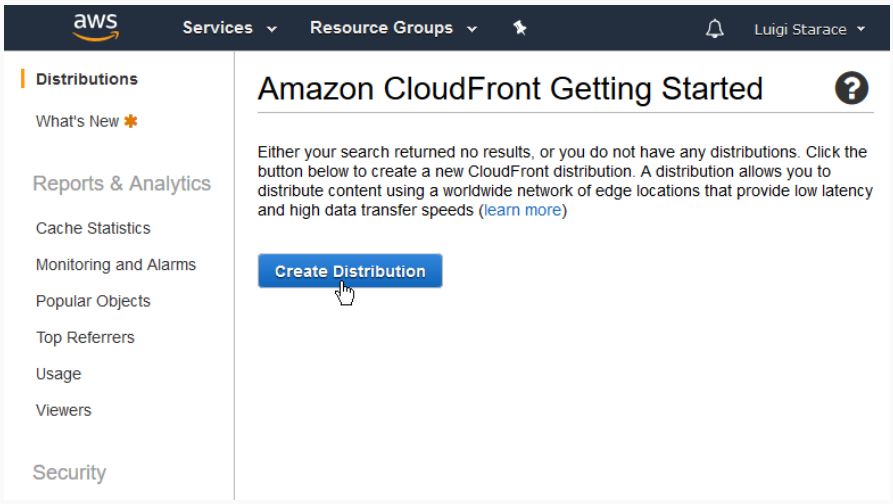
Step 13: Host the static files with S3

You should see a very nice single-page serverless web application!

But wait, there's more!

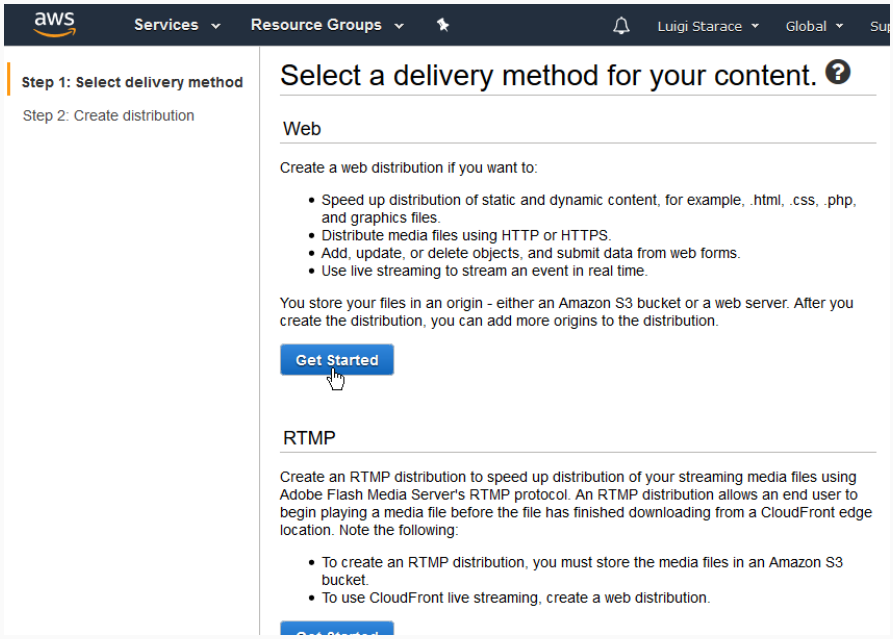
Such a nice web application wouldn't be complete without a global CDN to speed up load times. Se we'll now set up Amazon CloudFront to distribute the static files all over the globe.

Visit the CloudFront Dashboard and create a new distribution.



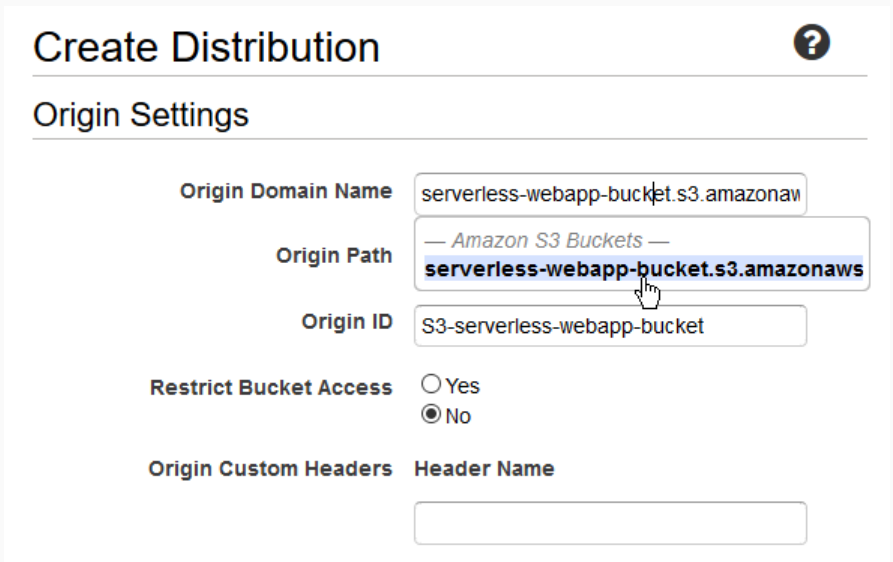
Step 14: optimize latency with CloudFront

Select web delivery method.



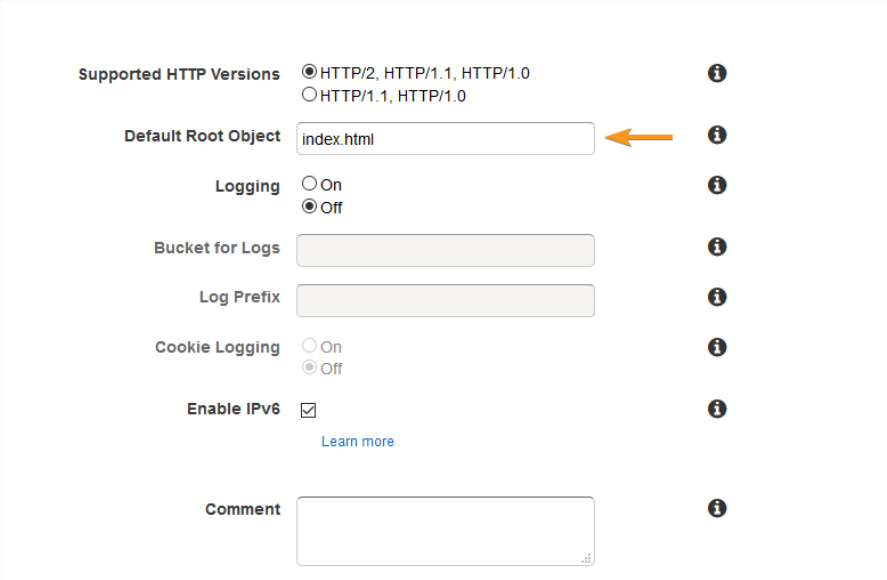
Step 14: optimize latency with CloudFront

Select your website bucket as the origin.



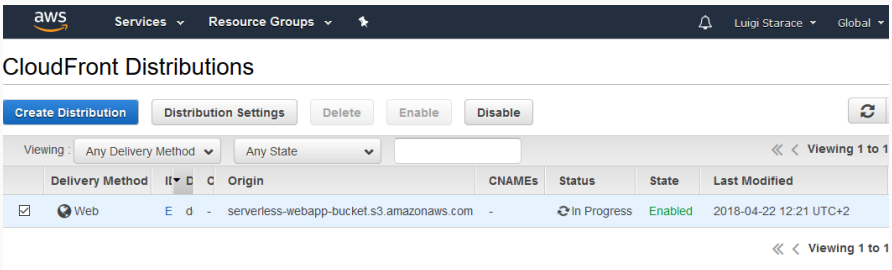
Step 14: optimize latency with CloudFront

Select index.html as the default root object.



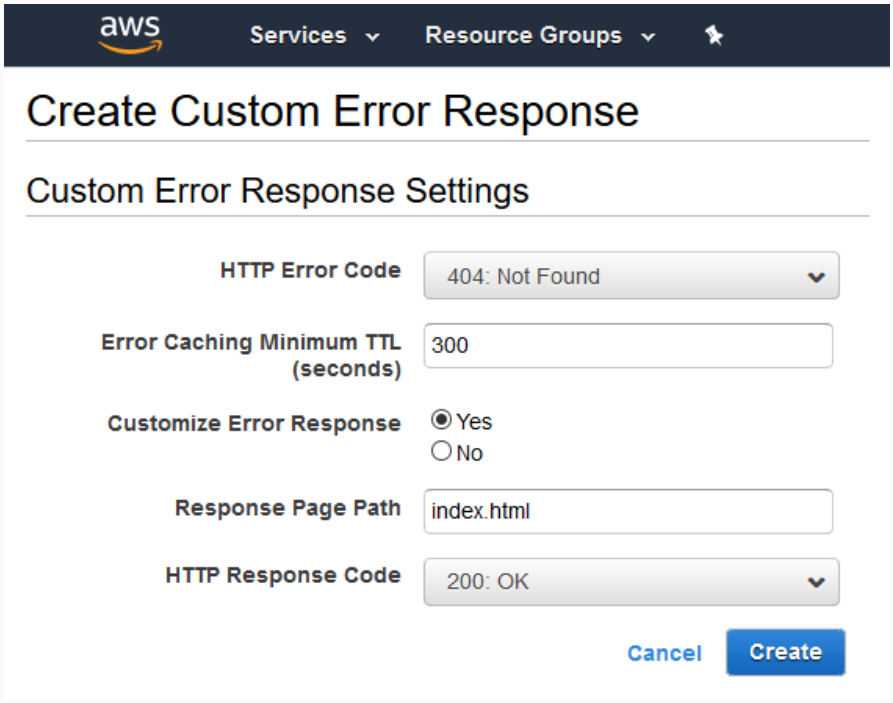
Step 14: optimize latency with CloudFront

It takes a few minutes to setup the distribution. When it's done the status will change to Deployed.



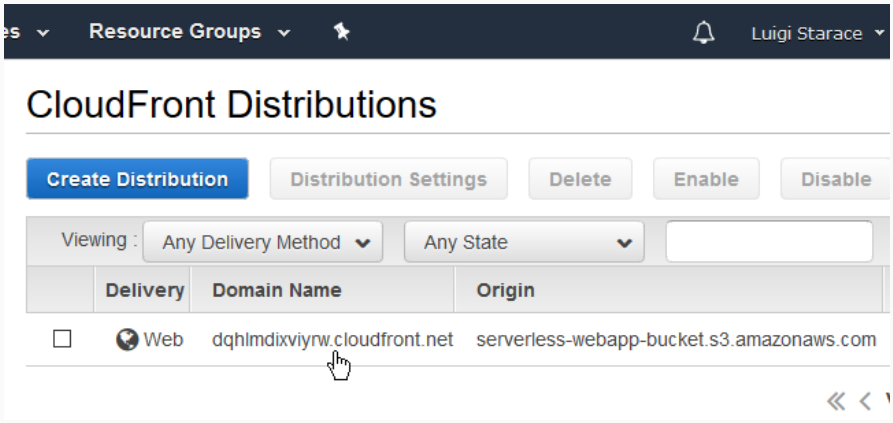
Step 14: optimize latency with CloudFront

In the distribution detail page add a custom error response as shown in the picture below to make sure 404 errors are handled by the application.



Step 14: optimize latency with CloudFront

Now you can visit the application from the cloudfront URL



Take Home Messages

Take Home Messages

- Cloud computing and service models
- AWS
- Deploy a “classic” web application on AWS
- FaaS and serverless computing
- Build and deploy a serverless one-page web application on AWS

Security recommendations

- Be **very** careful not to expose your IAM credentials;
- Enforce the least privilege principle: each user should only be able to access the minimum resources necessary to fulfill its purpose.

Security recommendations

- The very second you expose your credentials to the public, some bot may use them to spin up large numbers of EC2 instances. If that happens, the billing might be a scary surprise!
- Tools like the AWS-developed `git-secrets` help avoiding the exposure of IAM credentials

Back to the tutorial

IAM best practices web
Git Secrets - Github repository web

Nice readings I

[Jan16] Badri Janakiraman. *Serverless*. June 20, 2016. URL: https://martinfowler.com/bliki/Serverless.html (visited on 05/21/2018).

[LF14] James Lewis and Martin Fowler. *Microservices: a definition of this new architectural term*. Mar. 25, 2014. URL: https://martinfowler.com/articles/microservices.html (visited on 05/21/2018).

[Rob16] Mike Roberts. *Serverless Architectures*. Apr. 6, 2016. URL: https://martinfowler.com/articles/serverless.html (visited on 05/21/2018).

Nice readings II

[Rus16] Mark Russinovich. *Microservices: An application revolution powered by the cloud*. Mar. 17, 2016. URL: https://azure.microsoft.com/it-it/blog/microservices-an-application-revolution-powered-by-the-cloud/ (visited on 05/21/2018).

[Ser] Serverless inc. *Serverless guide*. URL: https://github.com/serverless/guide (visited on 05/21/2018).

References I

[Amaa] Amazon Web Services. *What is cloud computing?*. URL: https://aws.amazon.com/what-is-cloud-computing/ (visited on 03/30/2018).

[Amab] Inc. Amazon Web Services. *Set up a Continuous Deployment Pipeline using AWS CodePipeline*. URL: https://aws.amazon.com/it/getting-started/tutorials/continuous-deployment-pipeline/ (visited on 06/10/2018).

References II

[Amac] Inc. Amazon Web Services. *Tutorial: Create a Four-Stage Pipeline*. URL: https://docs.aws.amazon.com/codepipeline/latest/userguide/tutorials-four-stage-pipeline.html (visited on 06/10/2018).

[Ama17] Inc. Amazon Web Services. *Practicing Continuous Integration and Continuous Delivery on AWS*. Tech. rep. June 2017. URL: https://d1.awsstatic.com/whitepapers/DevOps/practicing-continuous-integration-continuous-delivery-on-AWS.pdf (visited on 06/01/2018).

References III

[AWS] AWS. *AWS Step Functions*. URL: https://aws.amazon.com/step-functions/?nc1=f_ls (visited on 05/01/2018).

[Gar17] Gartner. *Gartner Forecasts Worldwide Public Cloud Services Revenue to Reach $260 Billion in 2017*. Oct. 12, 2017. URL: https://www.gartner.com/newsroom/id/3815165 (visited on 03/30/2018).

References IV

[Syn18] Synergy Research Group. *Cloud Growth Rate Increases; Amazon, Microsoft & Google all Gain Market Share*. Feb. 2, 2018. URL: https://www.srgresearch.com/articles/cloud-growth-rate-increases-amazon-microsoft-google-all-gain-market-share (visited on 03/30/2018).