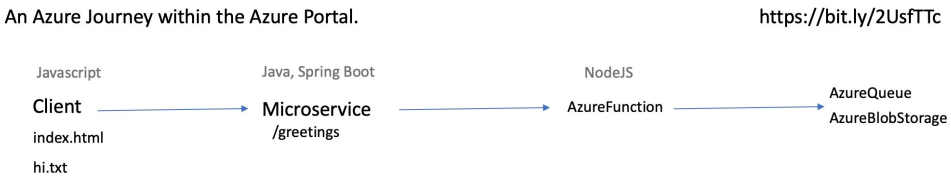
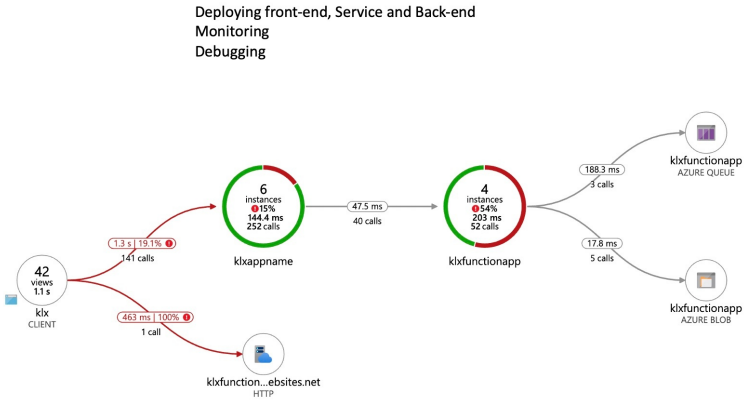
Created by Lomax, Ken, last modified by Vasconcellos Gomes, Marcelo on Apr 08, 2021

Journey

In this cloud lab you will do the following, all from within the Azure Portal:

- build and deploy
  - a simple Java, Spring-boot microservice
  - a simple web page that calls this microservice,
  - an Azure Function that writes data to an Azure Queue and Azure Blob
  - modify the microservice to call this Azure Function
- monitor the running system with Application Insights
- resolve a CORS issue and deliberate javascript error





Video of Journey

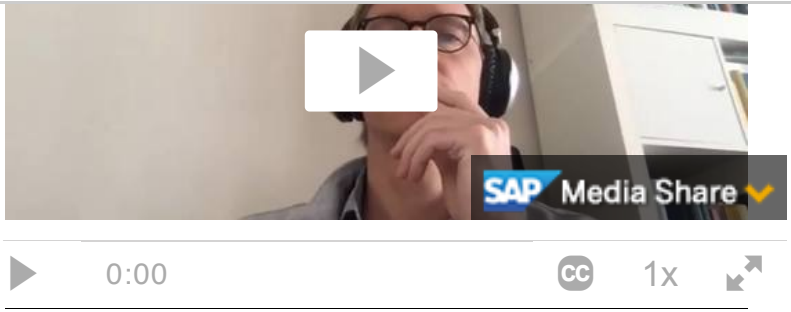
We want to keep this Cloud Labs correct. Please report any problems you have to ken.lomax@sap.com"

Useful Links:

- Azure Portal
- Balance of those with sponsorship accounts
- Azure for Everyone YouTube channel
- Microsoft's Github training labs material

Here is a video screencast of me going through this journey

up with the cloud"



Prerequisties

In this lab you need:

- a subscription on https://portal.azure.com/ - this can also be a free trial subscription
- a short, unique prefix to distinguish your azure deployment from others - in this journey I use **klx**, which you should replace with your own.
- occasionally I refer to path **/home/ken**. You should assume that is /home/<yourAzureName>
- we assume you do not yet have file storage mounted to your Azure CLI. If you have, unmount it before starting this lab, (with the CLI command "clouddrive unmount")

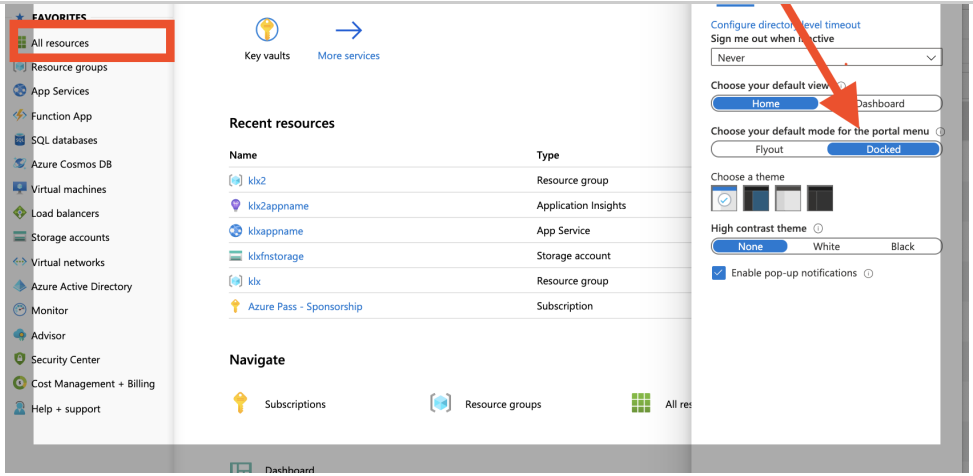
Cloud Lab Steps

Pertinent Links

Open and set up your Azure Portal

up with the cloud"

Select "Docked" so that the menus on the left stay there, rather than constantly hiding away.



The "All resources" link (or "blade") in the left panel is the main starting point where you will often navigate from.

Mount a storage to your cloud shell

Open Azure's Cloud Shell

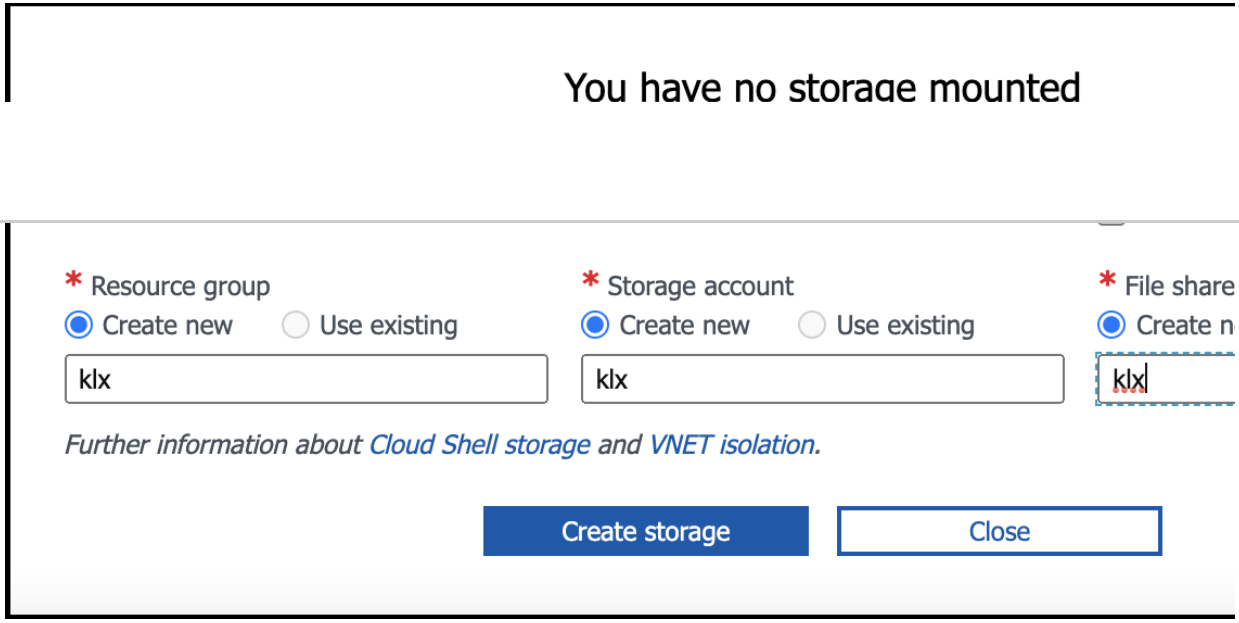
- A pop up will tell you "You have no storage mounted". This makes sense. The CLI needs some storage to store the files that you create..
- -> "Show advanced settings" and create
  a new Resource group,
  a new Storage account and
  a new File Share,
  all with the name **klx.**
- -> "Create Storage"

Link straight to an Azure Cloudshell @ https://shell.azure.com/

Mounting, Unmounting clouddrives

Azure CLI for running outside of the portal.

up with the cloud"



The naming can be arbitrary, but it helps to name your resources with care, to help you keep an overview of what belongs to what.

Create a skeletal microservice

List the files your CLI now sees.

> ls -la

Note that one of the entries is **clouddrive.** This is a link to the fileshare you just created. Change into that directory.

> cd clouddrive

Create a new spring boot microservice, called javaspringboot101, using spring's start.spring.io curl endpoint:

> curl https://start.spring.io/starter.zip -d dependencies=web,devtools -d bootVersion=2.4.3.RELEASE -o javaspringboot101.zip

Unzip this new skeletal microservice and cd into it:

Spring initializr

https://start.spring.io/

```
> unzip javaspringboot101.zip -d javaspringboot101
> cd javaspringboot101
```

up with the cloud"

and use it to navigate to the file
src/main/java/com/example/demo/DemoApplication.java

Replace the existing code in DemoApplication.java with the following, which will add a RestController with endpoint "/greetings". There are also some commented out lines, that we will be using later.

src/main/java/com/example/demo/DemoApplication.java

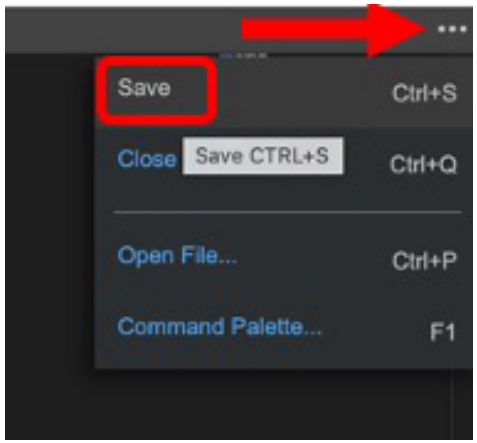
```
package com.example.demo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestPara
import org.springframework.web.bind.annotation.RestControl
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootAp
import org.springframework.http.ResponseEntity;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
public class DemoApplication {
  public static void main(String[] args) {
    SpringApplication.run(DemoApplication.class, args);
  }
```

```
@RestController
```

up with the cloud"

```
    public String greeting(@RequestParam(value = "name",
        return "Hi there "+name;

        /*
        RestTemplate restTemplate = new RestTemplate();
        String functionUrl = "...";
        ResponseEntity<String> response = restTemplate.get
        return "Hi there "+name+" Function response: "+res
        */
    }
}
}
```

And save it:



Create a new file src/main/resources/public/index.html that will contain a simple web page, to call this endpoint:

up with the cloud"

and add this code to it:

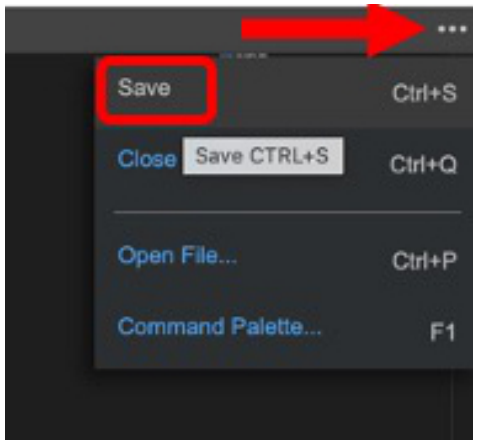
src/main/resources/public/index.html

```
<html>
<head>
  <script src='https://ajax.aspnetcdn.com/ajax/jQuery/jque
</head>
<body>
  <button id='callmyservice'>CallMyService</button>

  <div class='alert alert-primary' role='alert' id='messag
  <script>
    $( '#callmyservice' ).click(function() {
      var url = 'greetings';
      $.get(url)
      .done(function(data) {
        $( '#message' ).text(data).show();
      })
      .fail(function(error) {
        $( '#message' ).text(JSON.stringify(error)).show()
      })
    });
</script>
<br>
```

```
<img src="AzureJourney.jpg" width=806 height=508>
</body>
```

up with the cloud"

And save it:

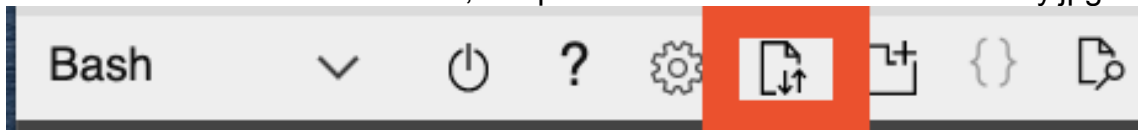


This code has a button that when clicked, calls your endpoint. The resulting response is displayed in the "message" div.

Upload the missing image

The index.html also displays an image, which we need to place in javaspringboot101/src/main/resources/public:

- Download this jpg to your laptop
- Then use the file transfer button, to upload it to /home/ken/AzureJourney.jpg



- Move it from /home/ken to javaspringboot101/src/main/resources/public:
  > mv /home/ken/AzureJourney.jpg src/main/resources/public

Run the microservice and access it via Web Preview, and via Curl.

Change the java version in your **pom.xml** to use java version 1.8:

```
<properties>
    <java.version>1.8</java.version>
```

Deploying a Spring Boot app to Azure

https://spring.io/guides/gs/spring-boot-for-azure/

```
</properties>
```

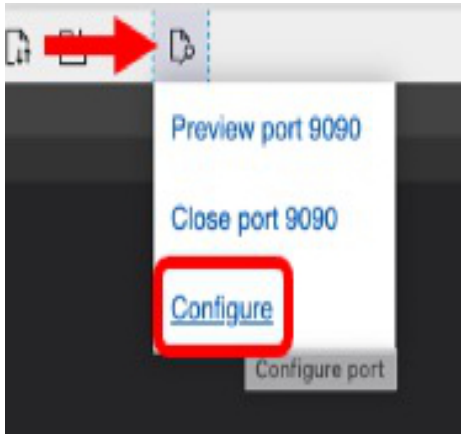
up with the cloud"

Build and run the microservice from the command line. As port 80 and 8080 are often taken, use 9090 instead:

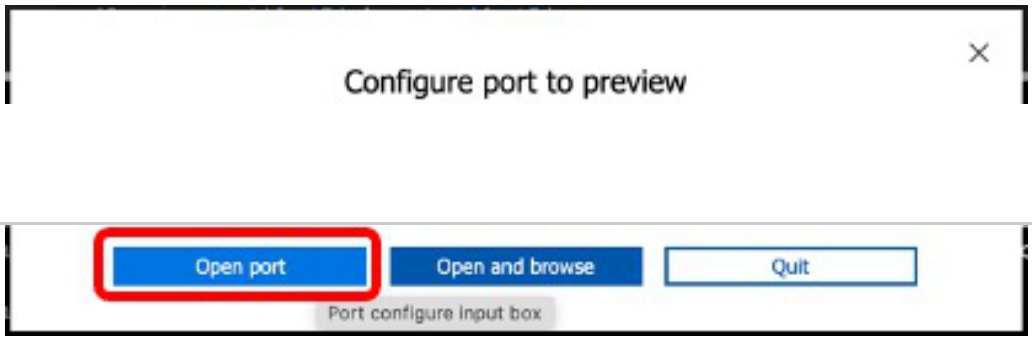
> mvn clean install

> mvn spring-boot:run -Dspring-boot.run.arguments=--server.port=9090

Click on Configure to open the port 9090:

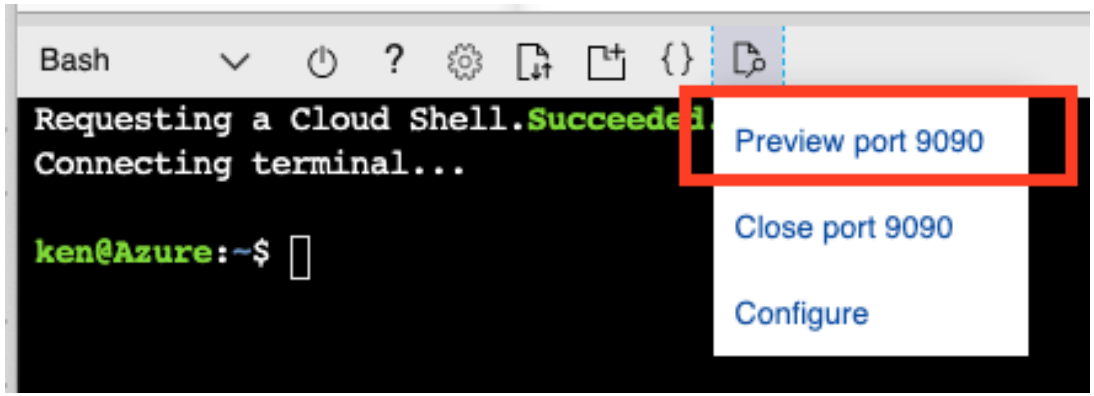


Type 9090 and confirm it:



up with the cloud"

Verify you can access your endpoint via Web preview:



Also, also try accessing via curl in a second CLI window:

> curl localhost:9090

Deploy the microservice into Azure's App Services

Until now, the microservice has been running locally via the CLI within Azure. We now want to deploy it into an Azure App Service. This will give us access to all the benefits AppService offers: scalability, security etc. There are several ways to do that. One is to enhance your pom.xml file to allow Azure Deployments from the command line:

>./mvnw com.microsoft.azure:azure-webapp-maven-plugin:1.8.0:config

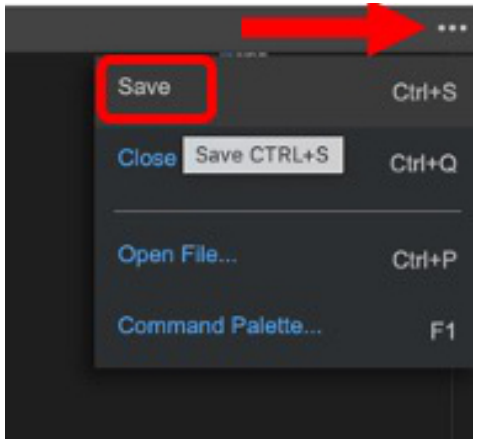
Accept the defaults (linux, java8)

This adds a new element to your **pom.xml**. Adjust the resourceGroup and appName:

up with the cloud"

```
...
<resourceGroup>klx</resourceGroup>
<appName>klxwebapp</appName>
...
```

And save it:



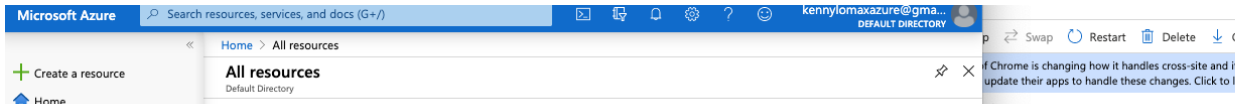
You can now build and deploy the microservice from the command line:

> mvn clean install

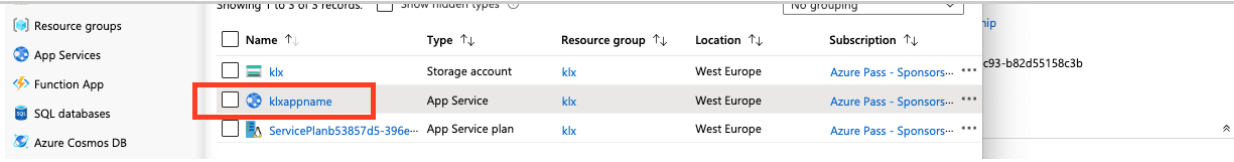
> mvn azure-webapp:deploy

Find and access the deployed webservice

Find the deployed web app in your list of resources and find its url endpoint:

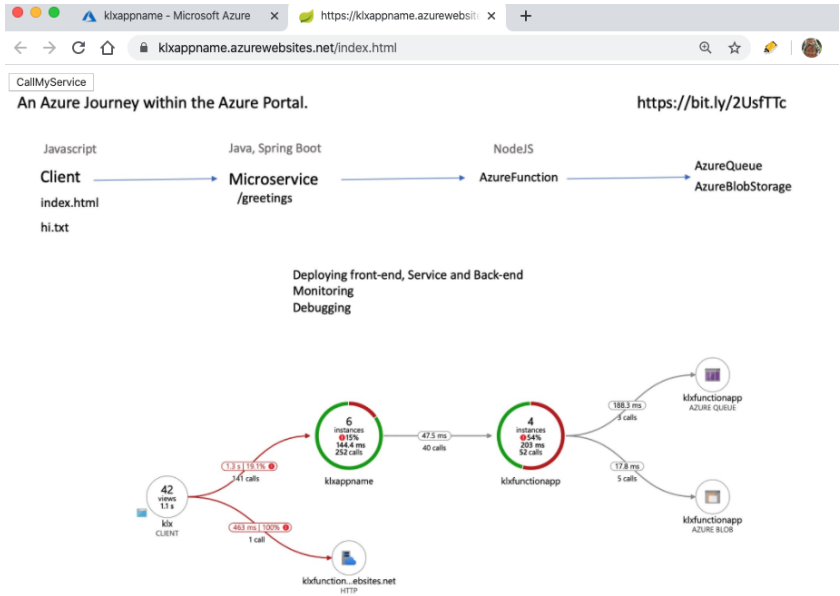


up with the cloud"



Verify that you can access your website from a browser:

- <your webapps url>/greetings?name=bod
- <your webapps url>/index.html
- Click the button in the website and confirm it invokes the /greetings endpoint.

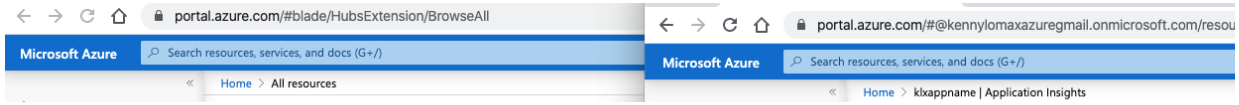


Gain monitoring insights..

Create an Application Insights resource, that will monitor your deployment:

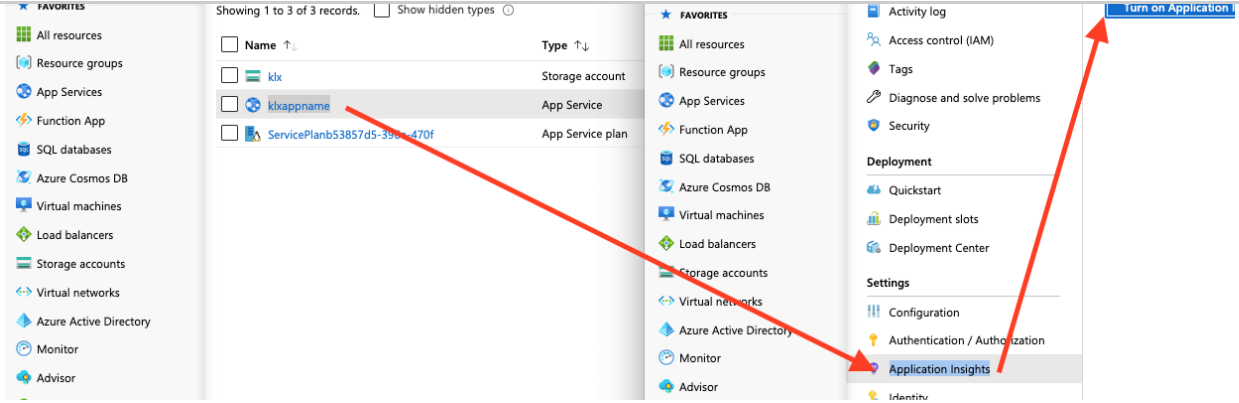
Persist files in Azure Cloud Shell

https://docs.microsoft.com/en-us/azure/cloud-

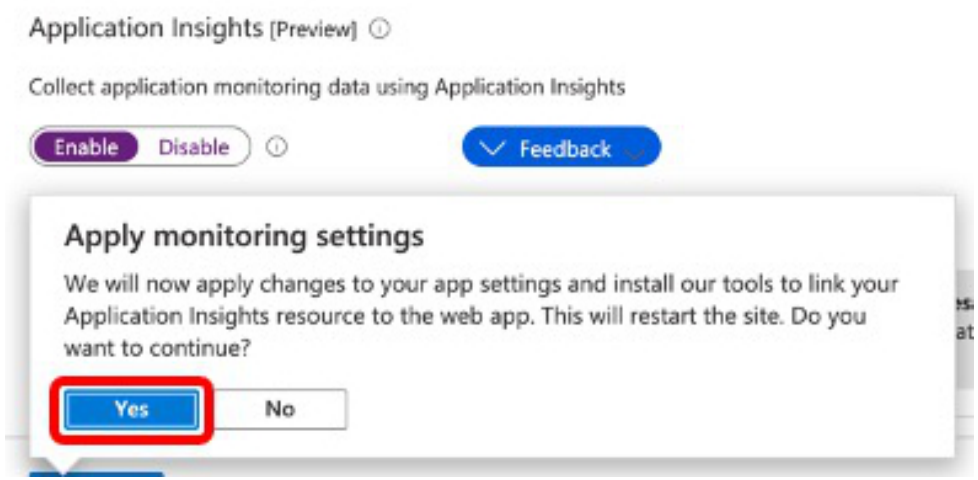


shell/persisting-shell-storage

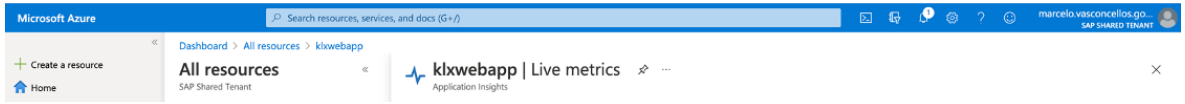
up with the cloud"



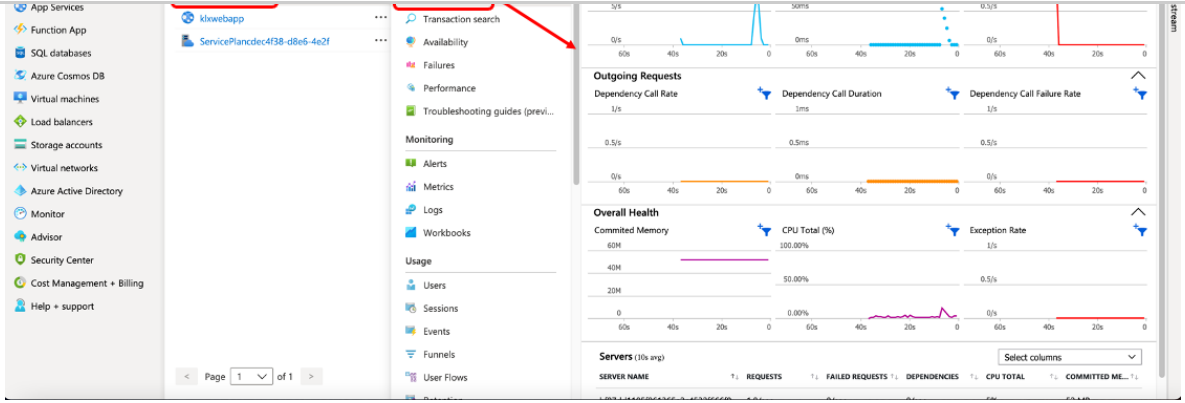
And apply it:



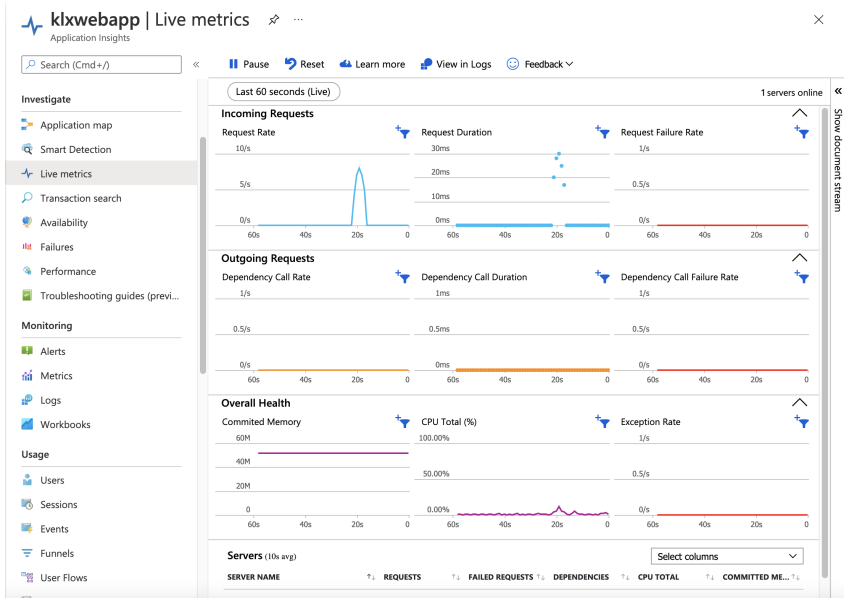
Once your Application Insights is visible in your resource list, select it, and then select "Live Metrics":



up with the cloud"



Hit your web site a few times, and confirm you can see the incoming requests appearing in the graphs.



Also monitor your web

The Application Insights is currently monitoring your microservice, but not the front

Application Insights for

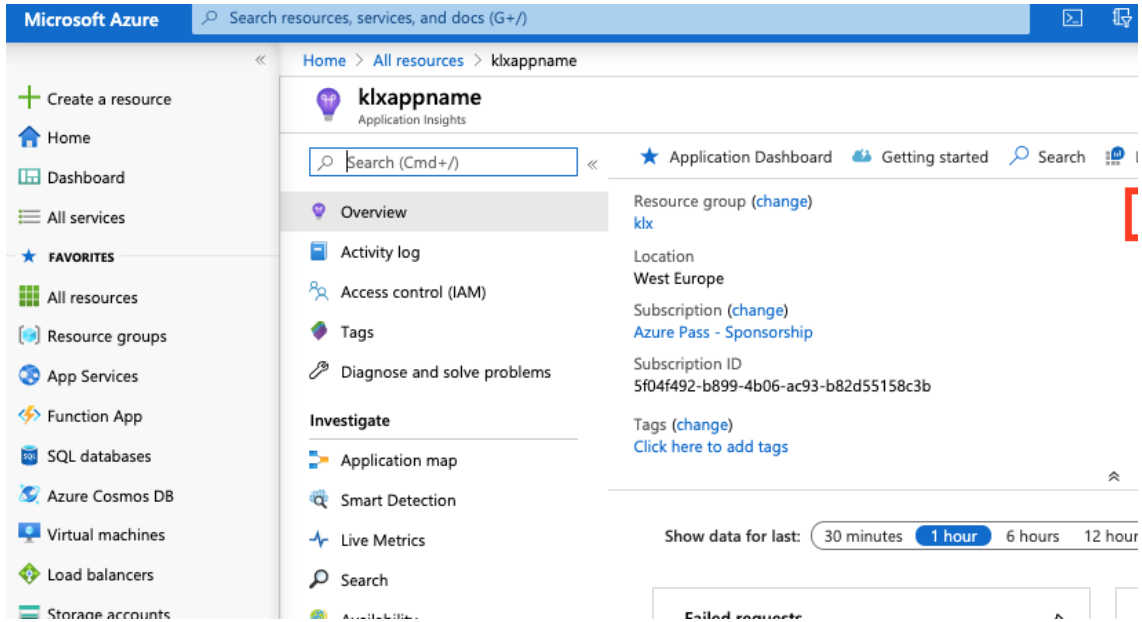
page webpage itself. web pages

Instrumentalize your **index.html** page so it can too be monitored: https://docs.microsoft.co

up with the cloud"

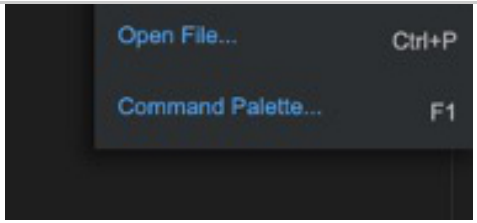
```
<script type="text/javascript">
var sdkInstance="appInsightsSDK";window[sdkInstance]="appI
{
  instrumentationKey:"INSTRUMENTATION_KEY"
}
);(window[aiName]=aisdk).queue&&0===aisdk.queue.length&&ai
</script>
```

- Replace the INSTRUMENTATION_KEY string with your app insight's own key:



And save it:

up with the cloud"



Redeploy the service with the newly instrumentalized front page:

> mvn clean install

> mvn azure-webapp:deploy

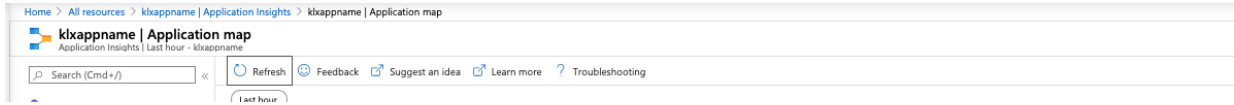
View your Application Map

The application map gives you a cool overview of your deployment.

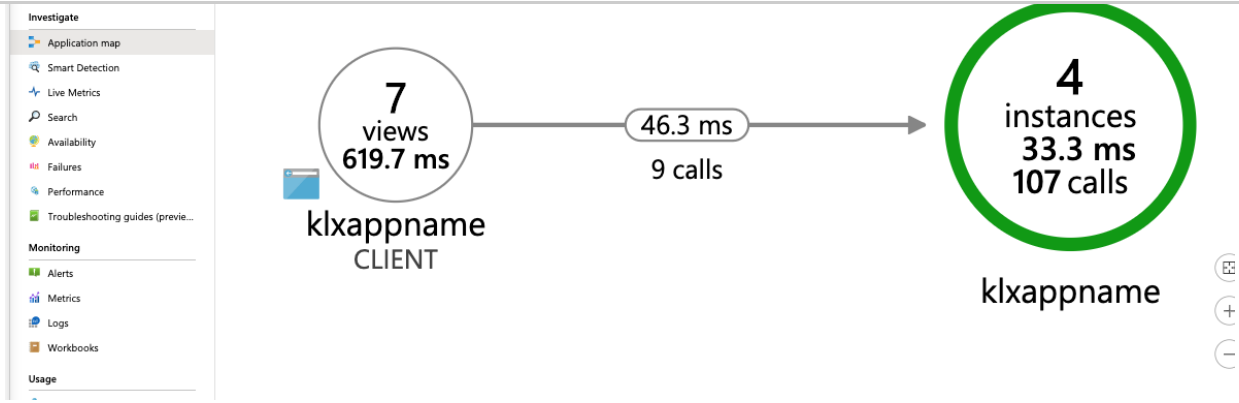
Wait for the new deployment and be sure your browser reloads this latest version (with the instrumentation key in the front page). Refresh your website several times, and click your website's "CallMyService" button a few times, so that Azure can track these calls.

Select the "Application Map" blade (above the "Live Metrics" blade) to see a map of your azure deployment.

After a few minutes you should see a 2 node network:

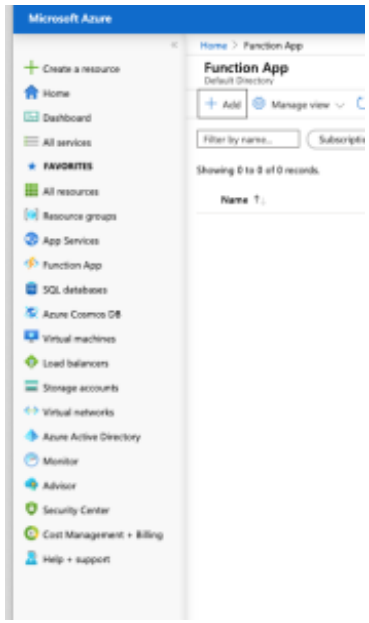


up with the cloud"



Extend your microserive logic with an Azure Function

Select "Function App" → "+Add"



App Service on Linux

https://docs.microsoft.com/en-us/azure/app-service/containers/app-service-linux-intro#limitations

This takes you thru several tabs to create a function, and it might be easiest to watch the video for this step. Use the following details when creating your function:

up with the cloud"

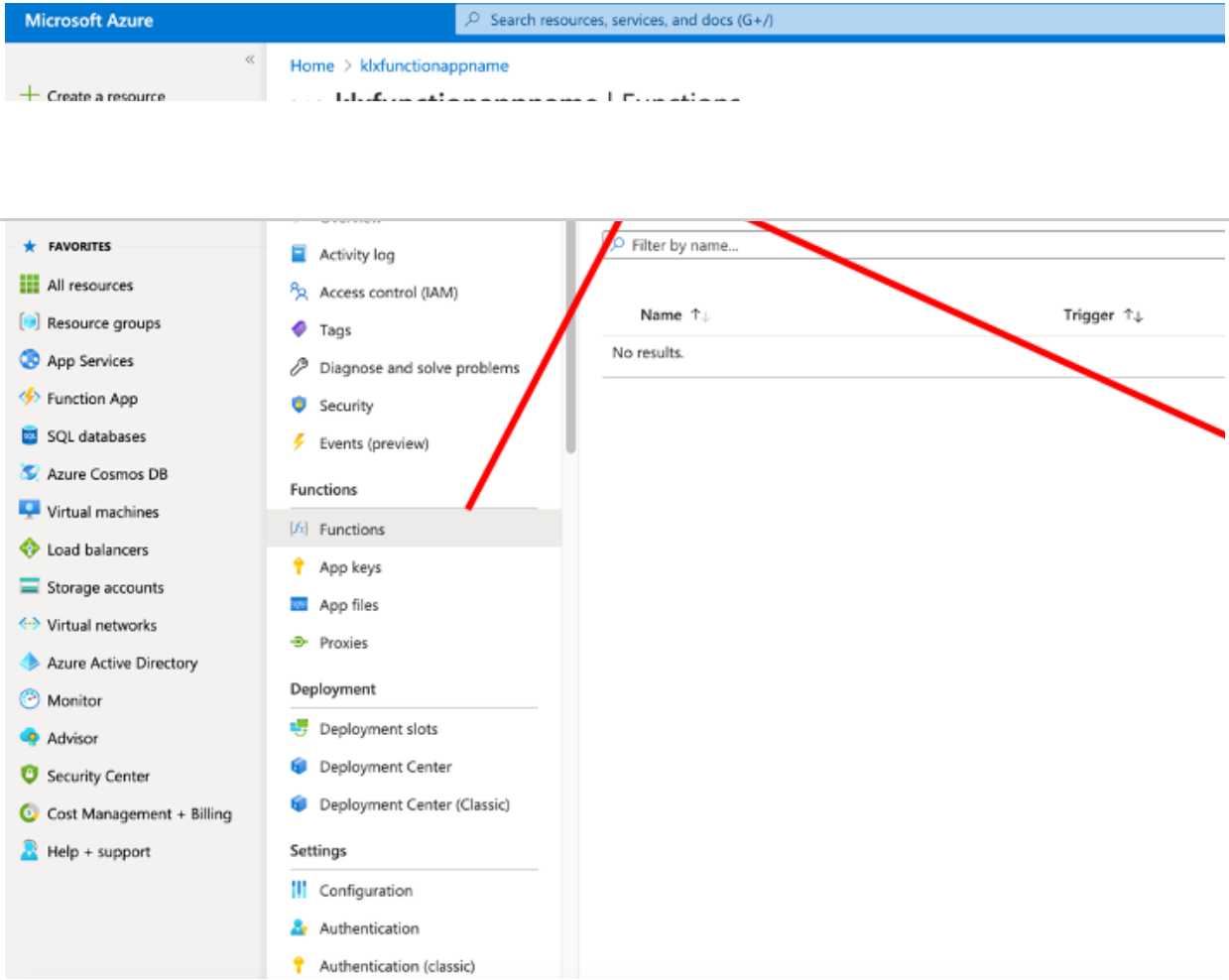
- Function App name: klxfunctionappname
- Publish: Code
- Runtime stack: Node.js
- Version: 12 LTS
- Region: <your region>

- Hosting
  - Storage account: klx
  - Operating System: Windows (because Azure supports in-portal function coding for Windows deployments)
  - Plan type: Consumption (Serveless)
- Monitoring
  - Enable Application Insights: Yes
  - Application Insights: klxwebapp (it is **important** that you choose the existing application insights you created already)
  - Region: <your region>

Click on "Review + create" and after on "Create" to confirm. Wait for your Function App to deploy, and then we can code a function within this FunctionApp.

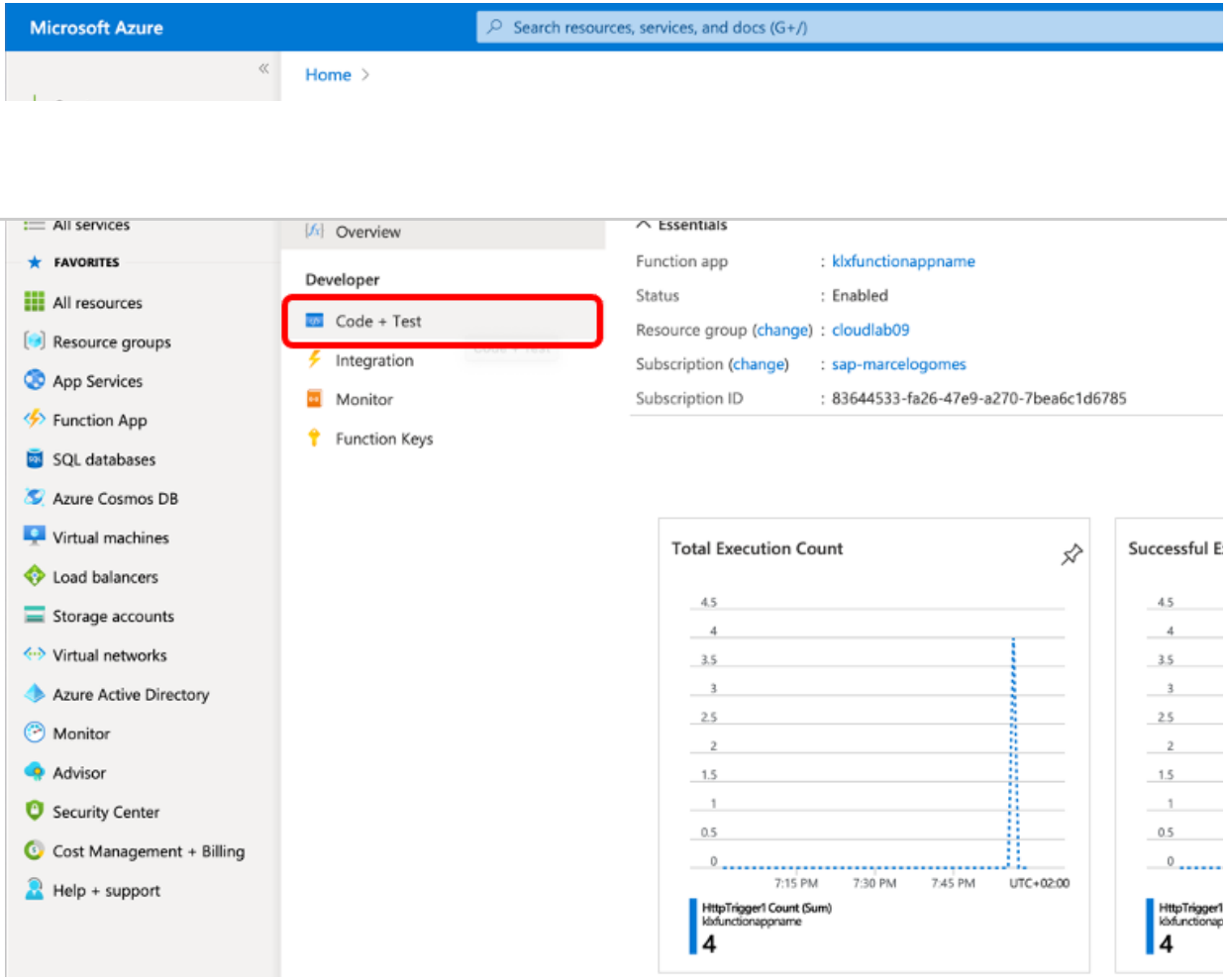
Write the function code itself

Follow this clickpath to create your function:



up with the cloud"

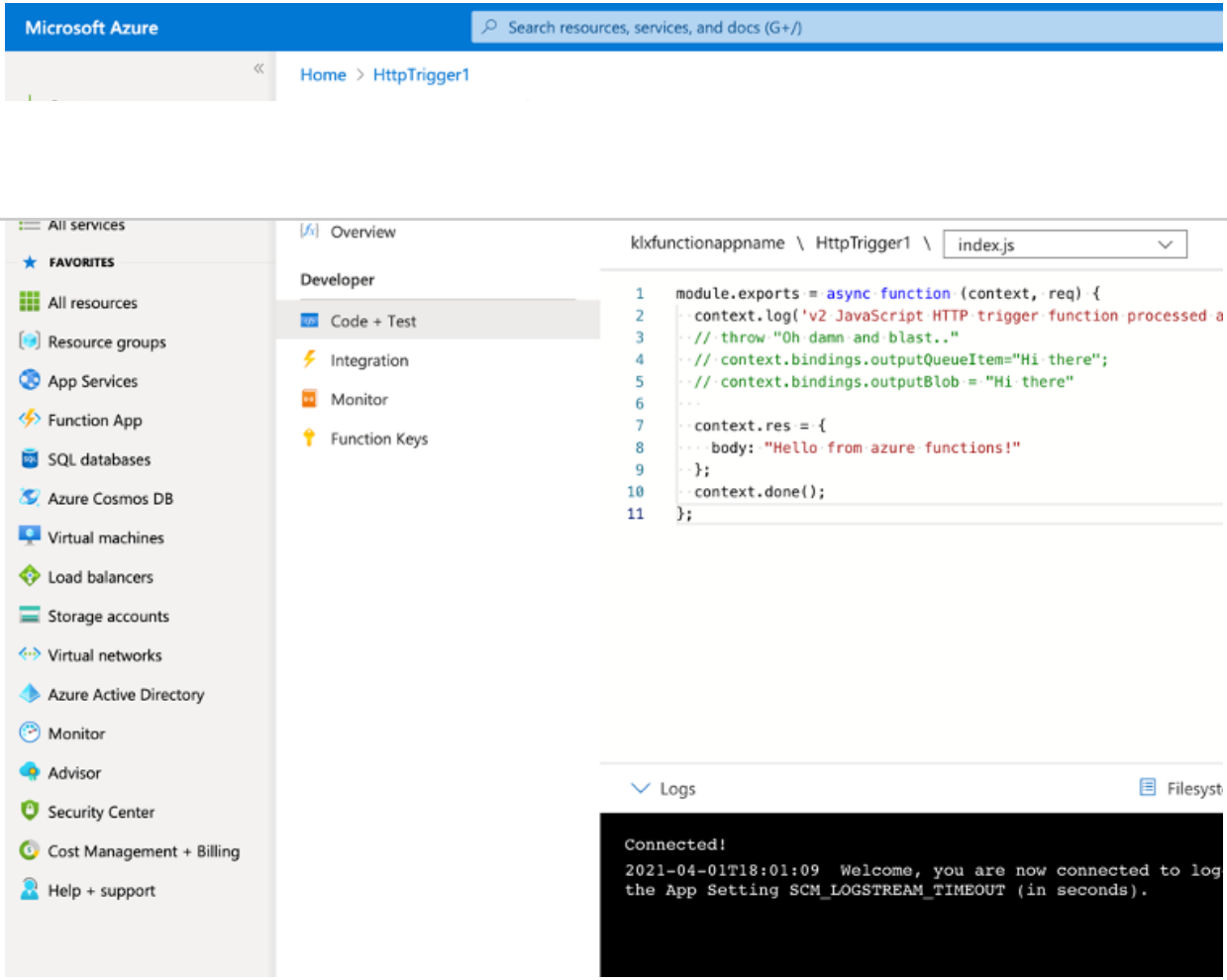
up with the cloud"



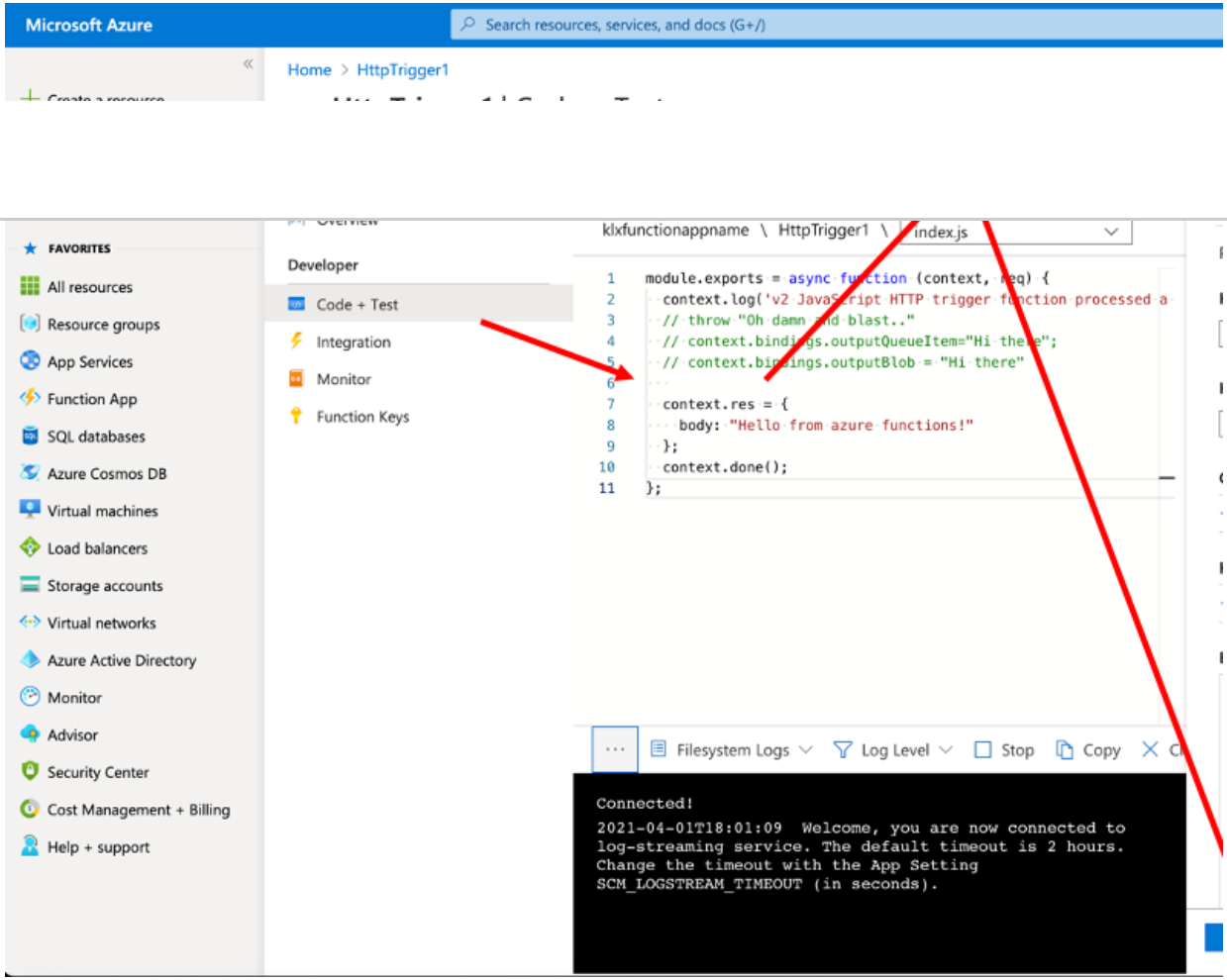
It might help to watch the video for that part.

Put this code into the Index.js text block that then appears:

up with the cloud"

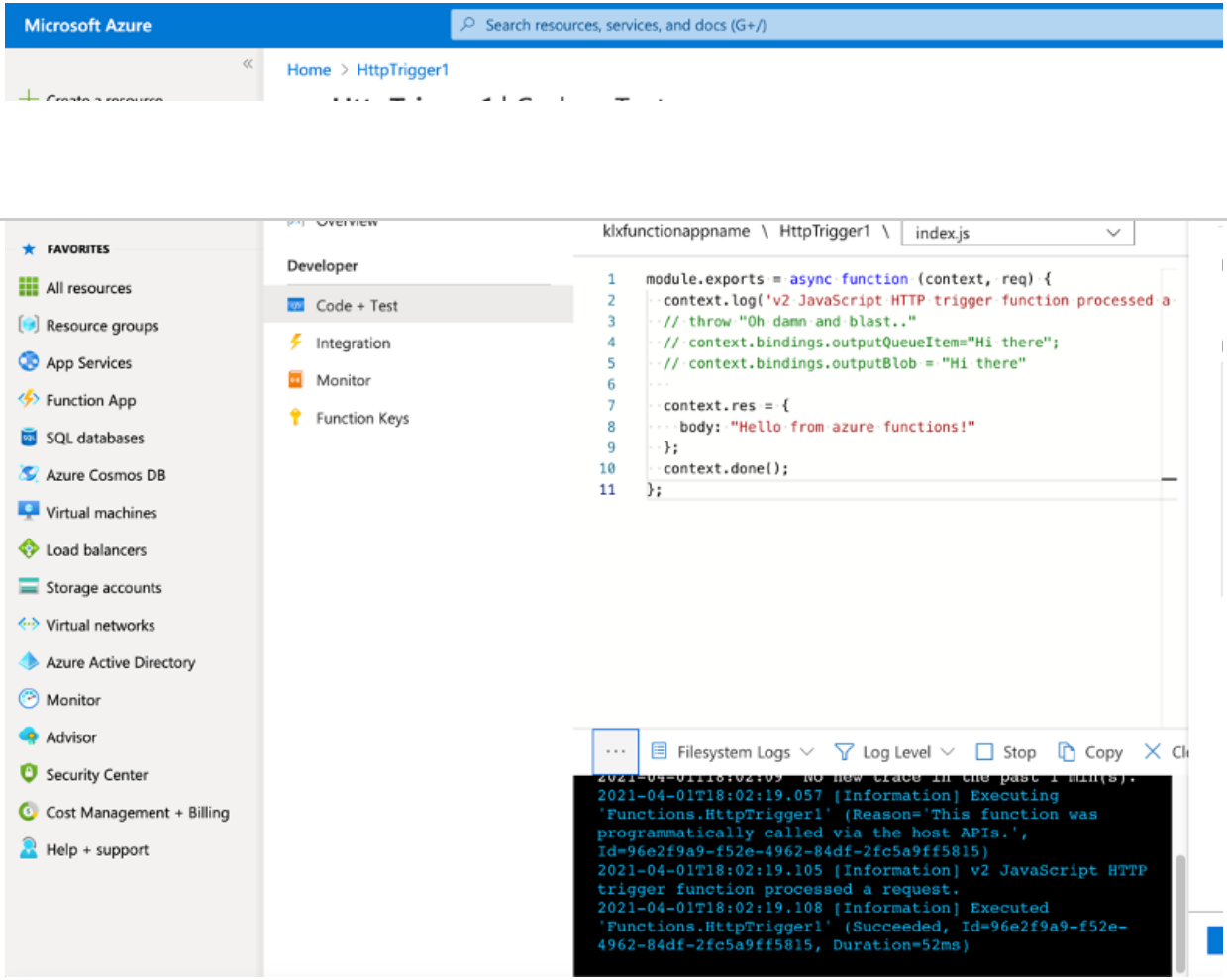


Click the "Save" button, after the "Test/Run" button and then the "Run" button to run the function a few times, and confirm you see "Status 200 OK" and the HTTP response content "Hello from azure functions!" on the right.



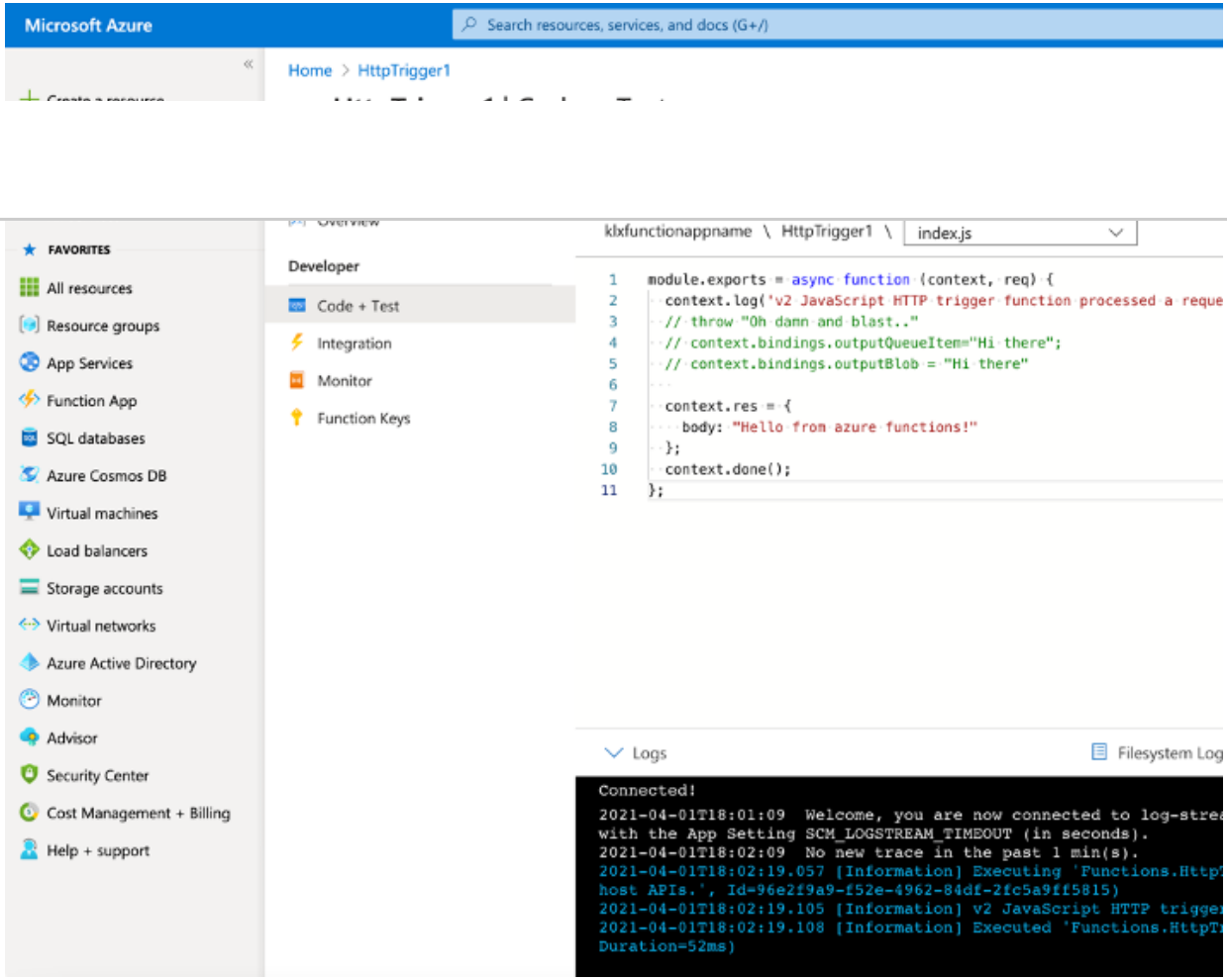
up with the cloud"

up with the cloud"



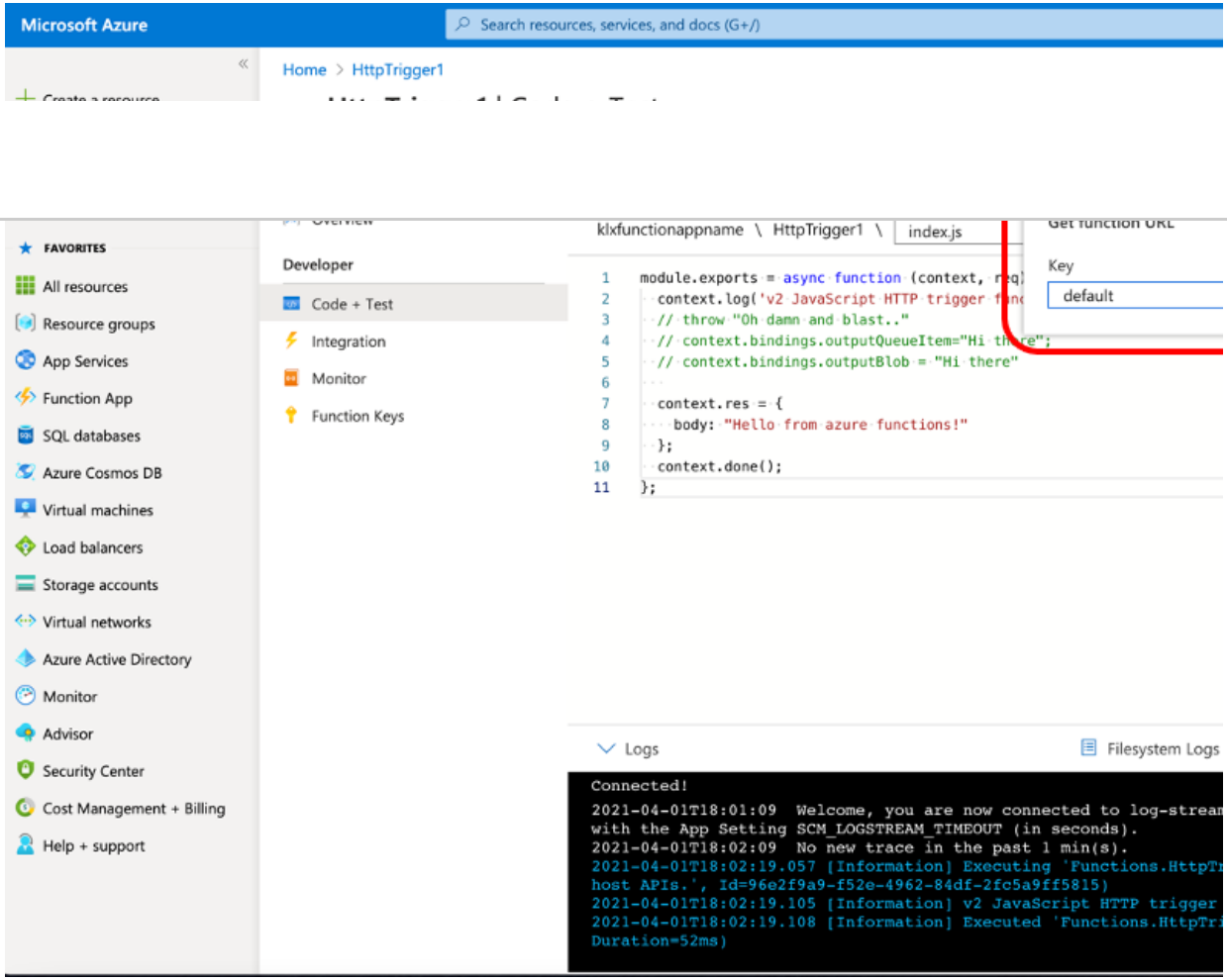
Call this function from your webservice

Find the function URL endpoint, which you will use below.



up with the cloud"

up with the cloud"



Adapt your **DemoApplication.java** to call your Azure Function.

Replace the YOUR FUNCTION ENDPOINT with your function's url:

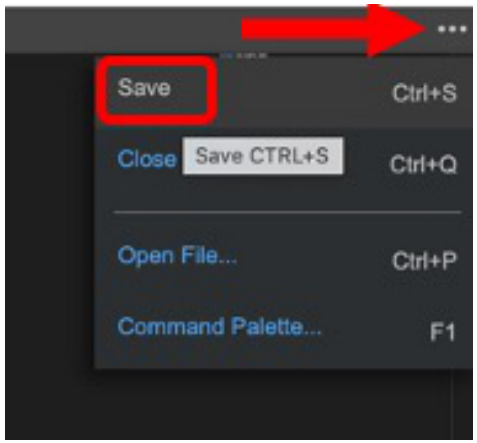
DemoApplication.java

```
public String greeting(@RequestParam(value = "name", defau
        //return "Hi there "+name;
        RestTemplate restTemplate = new RestTemplate();
```

```
String functionUrl = "YOUR FUNCTION ENDPOINT"; //
ResponseEntity<String> response = restTemplate.get
```

up with the cloud"

And save it:

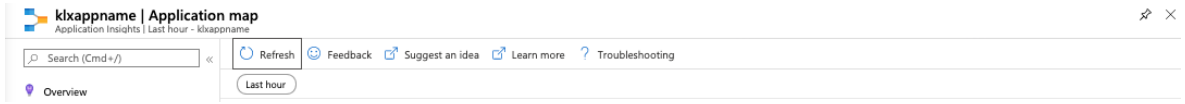


Rebuild and redeploy your microservice:

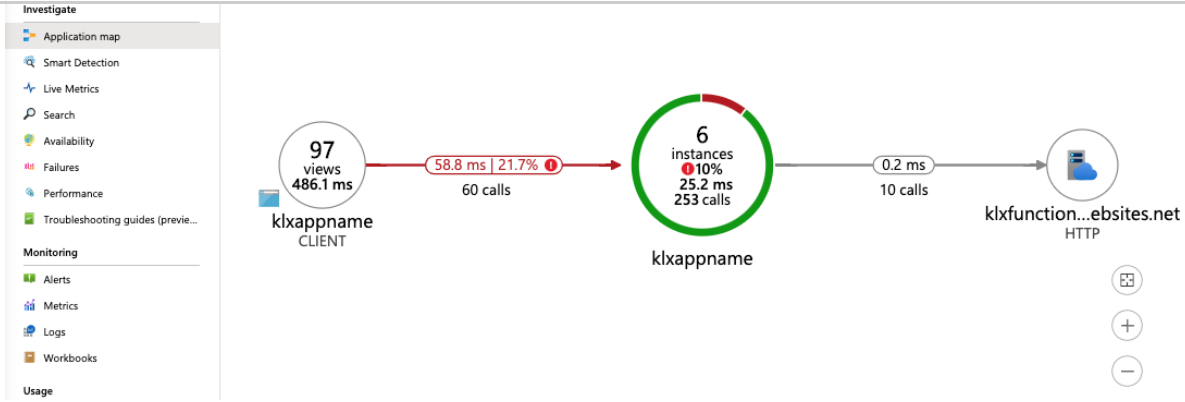
> mvn clean install

> mvn azure-webapp:deploy

Once deployed, refresh your webservice, and click your "CallMyService" button. After a while you should see a modified Application Map, now with 3 layers: the front page, the service and the function:

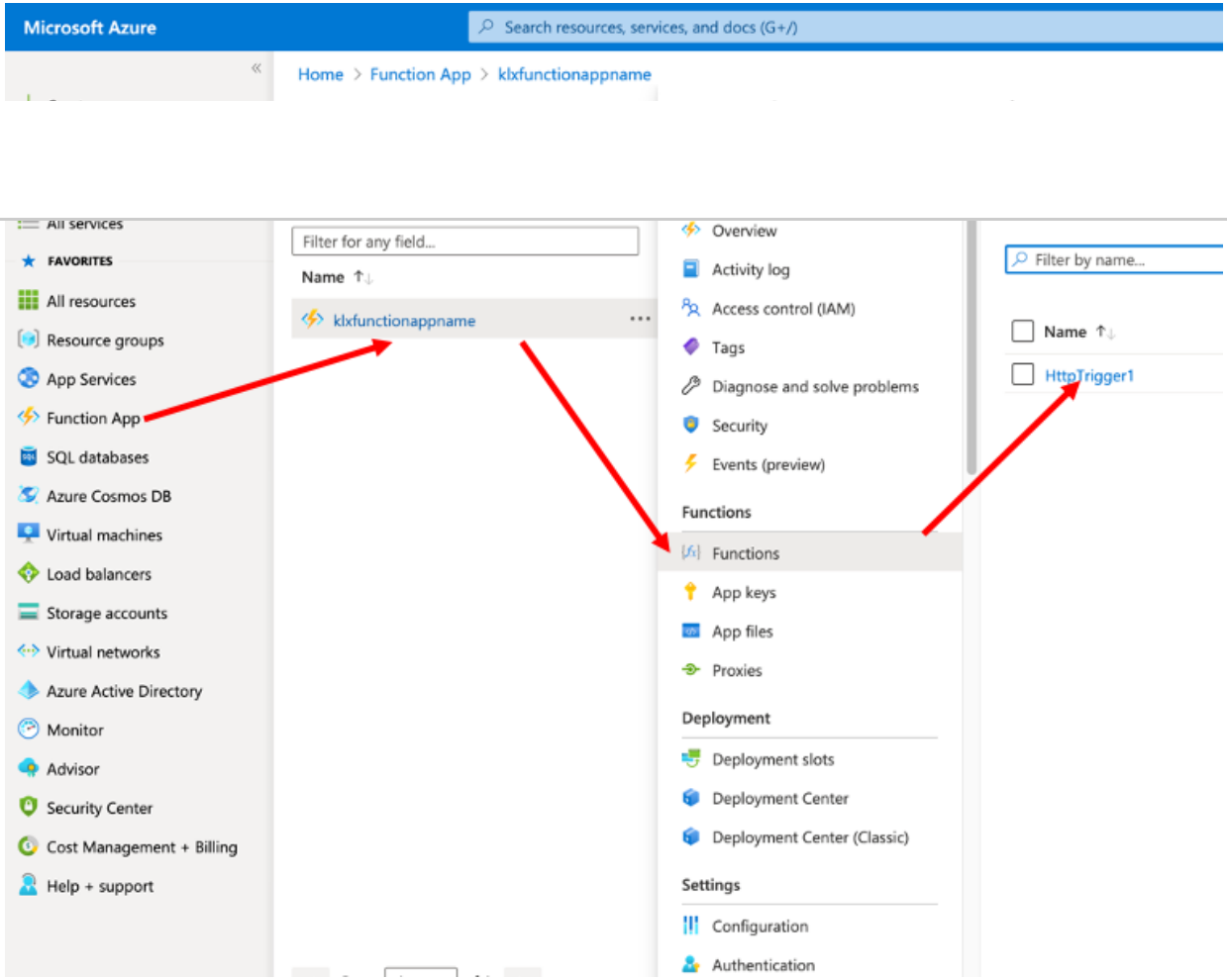


up with the cloud"

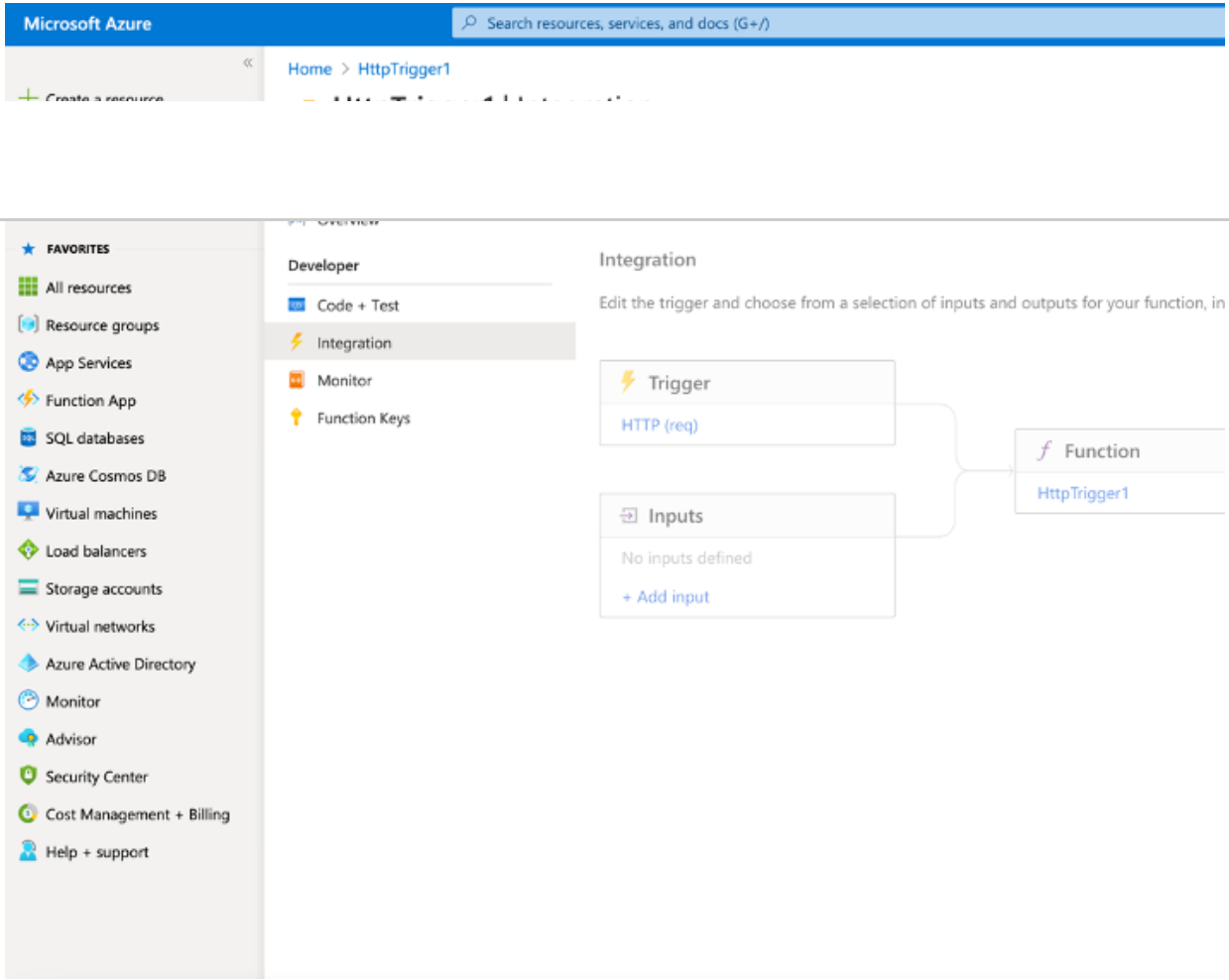


Add some logic to the function

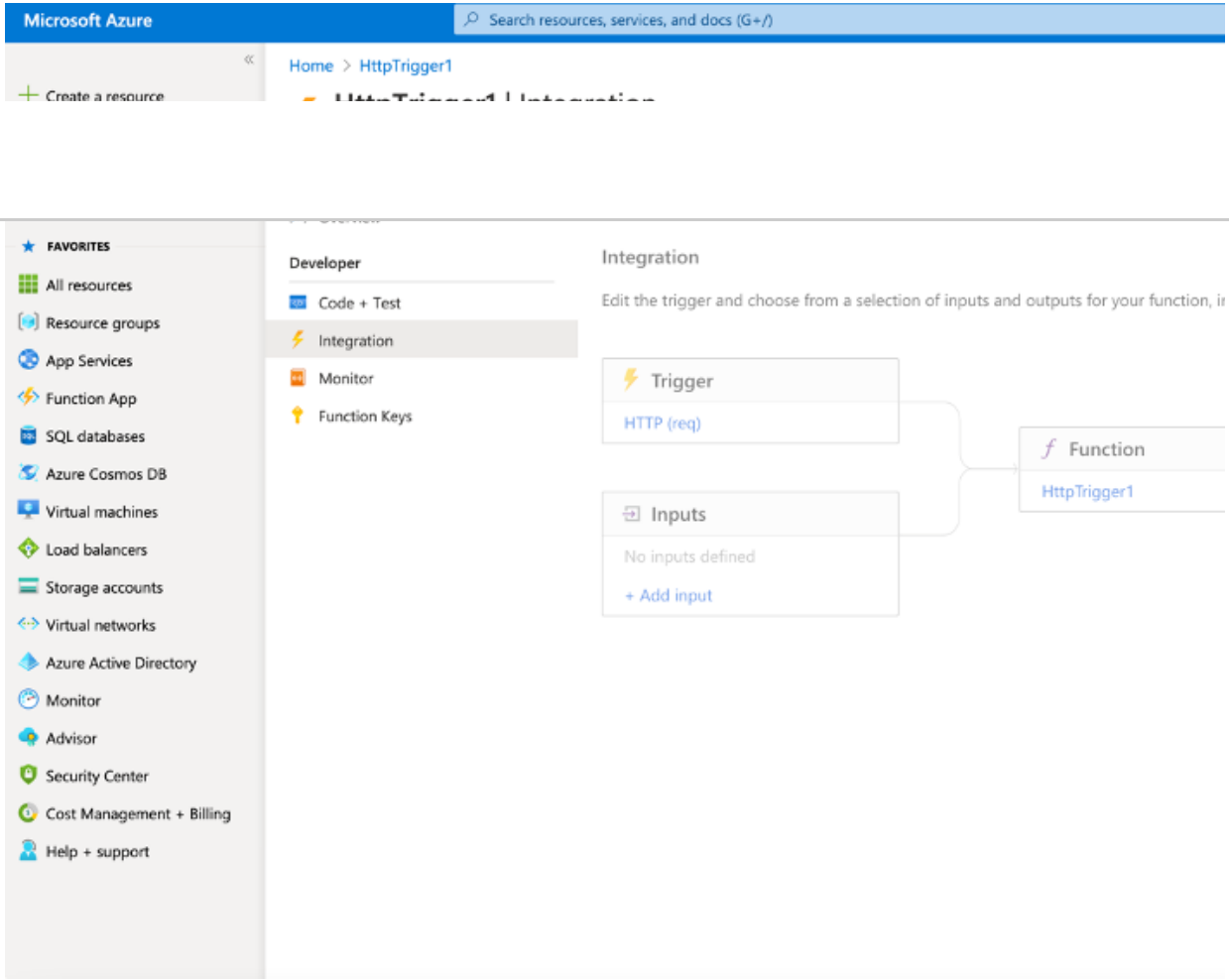
We will modify the function to output data to an Azure Blob and an Azure Queue:



up with the cloud"



up with the cloud"

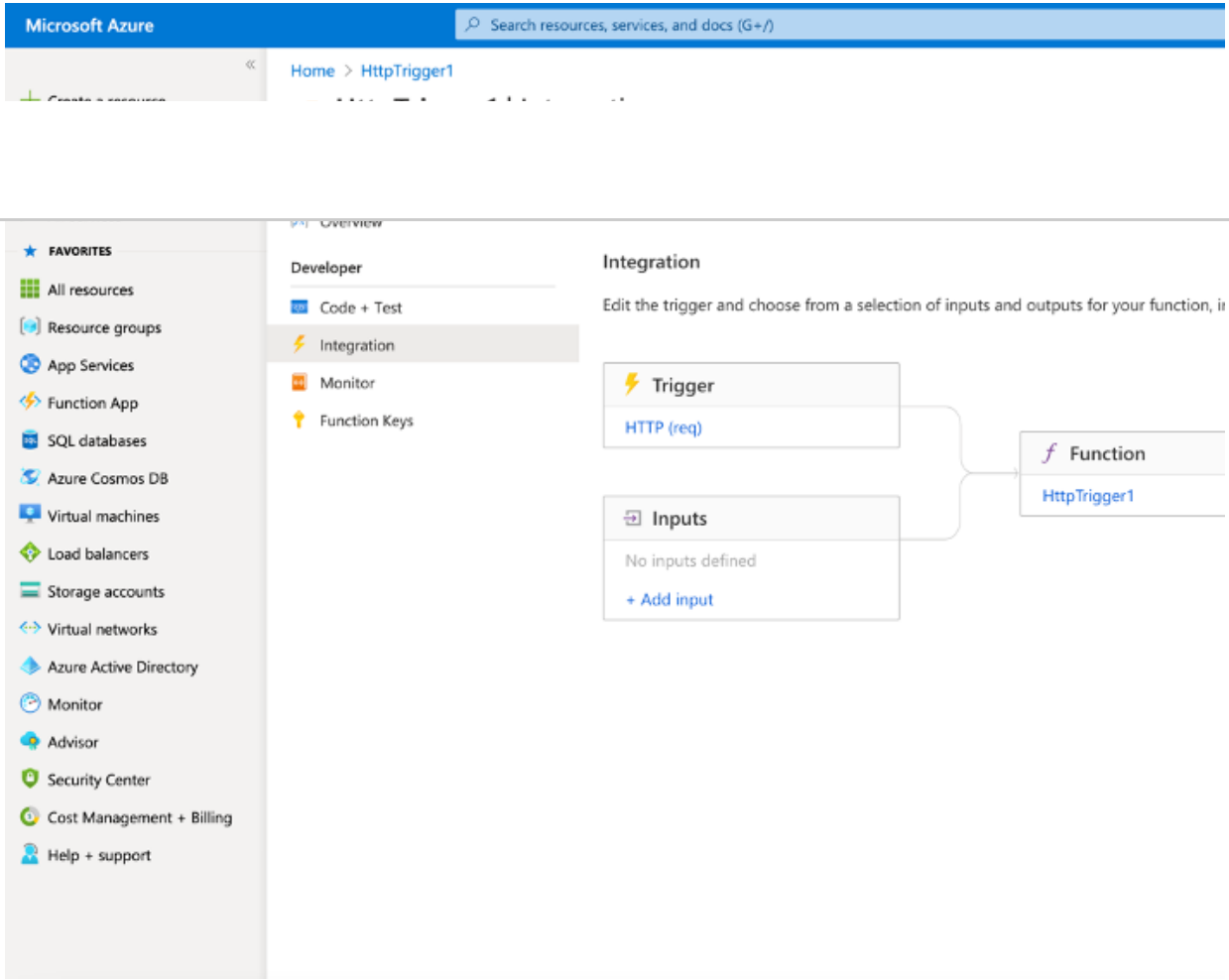


up with the cloud"

And accept the default settings.

Similarly add a new output to an an Azure Queue Storage, and accept the defaults. You should now see 3 outputs listed for your function:

up with the cloud"



Modify your functions code to output a message to this blob and to the queue:

Azure function

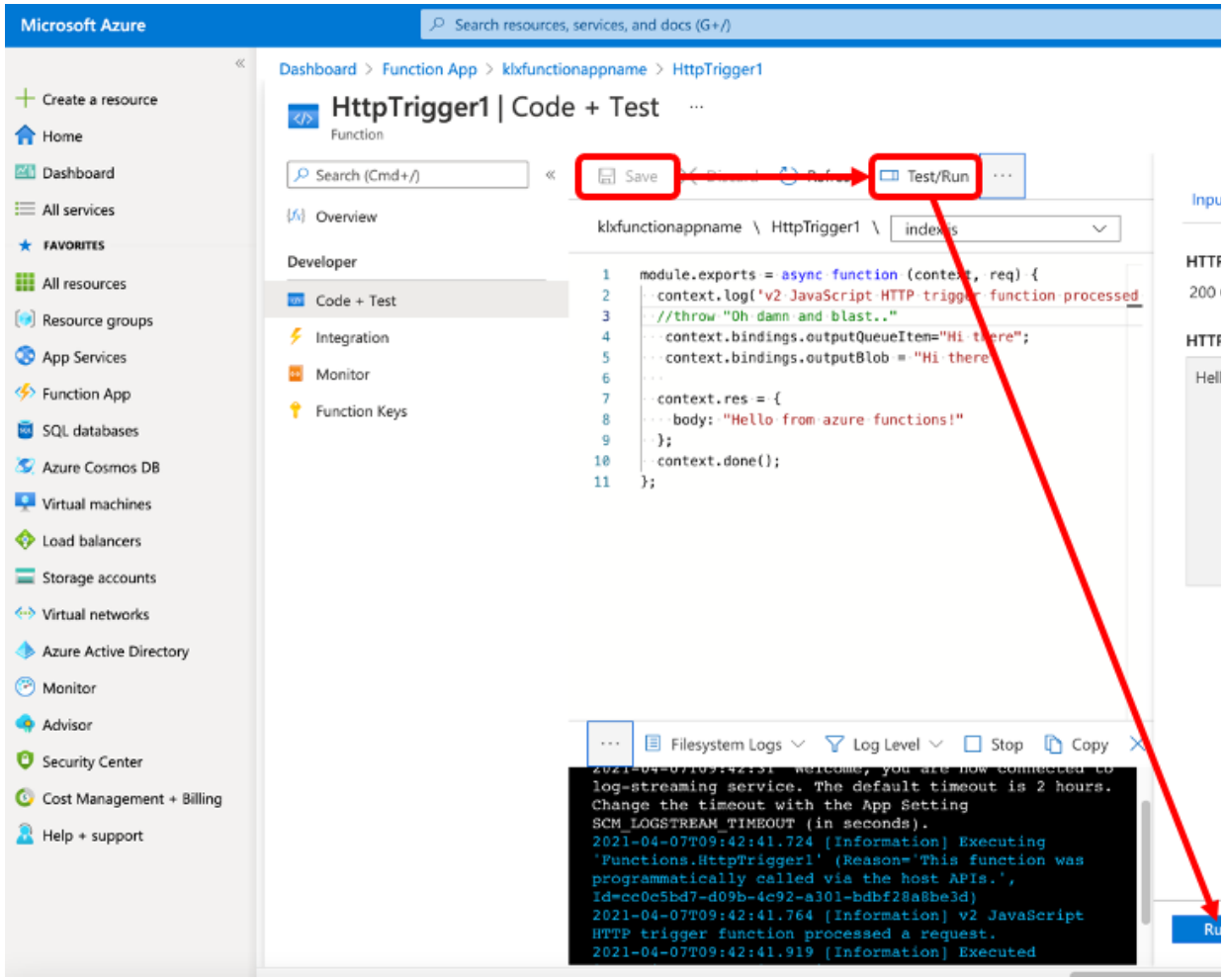
```
module.exports = async function (context, req) {
  context.log('v2 JavaScript HTTP trigger function process
  // throw "Oh damn and blast.."
   context.bindings.outputQueueItem="Hi there";
   context.bindings.outputBlob = "Hi there"
```

```
    context.res = {
```

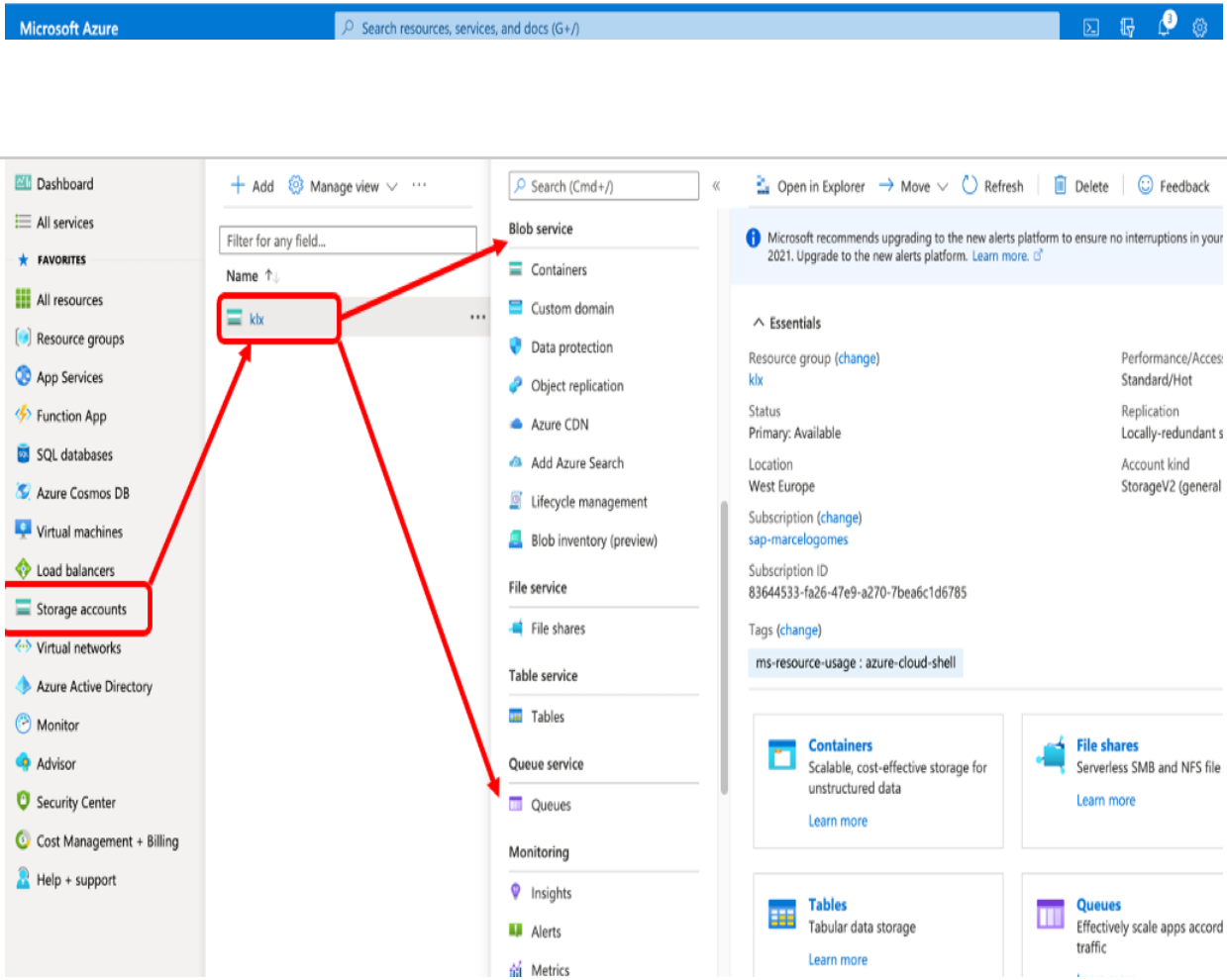
up with the cloud"

```
    context.done();
};
```

Save and run this function a few times.



The queue and blob are found in the "Storage accounts" blade.

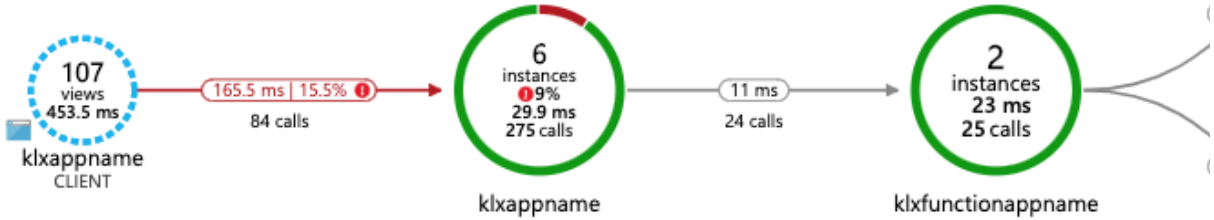


Navigate to the blob and queue and confirm you can see new items in there.

Note that you do not need to redeploy your service to use the new functionality - it is available immediately 🙂

Click the "CallMyServce" button in your webpage a few times, and in a few minutes you will see an updated application map, that now includes the queue and blob storage.

up with the cloud"



Introduce and debug a javascript error

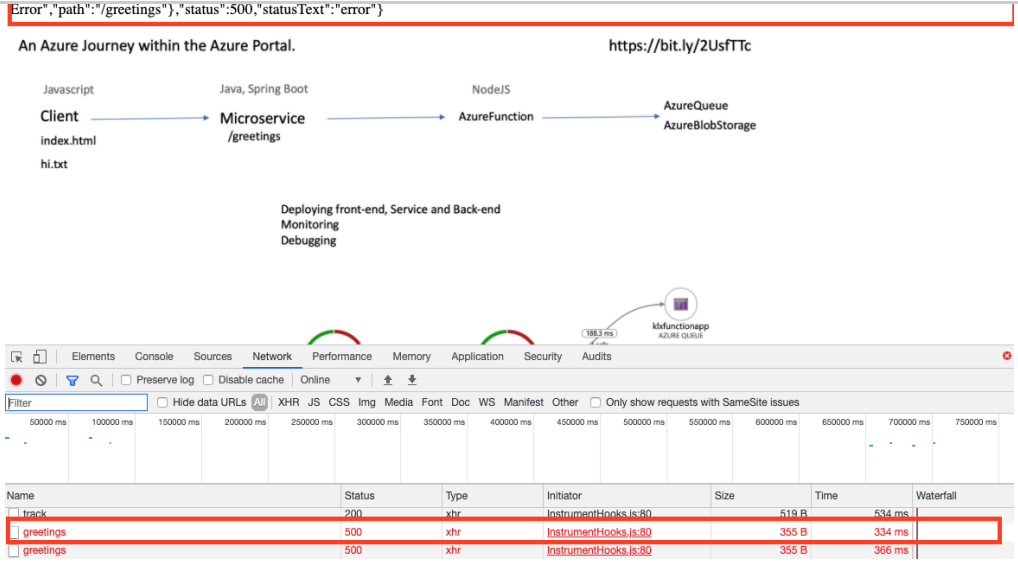
Introduce an error in your azure function

Azure function

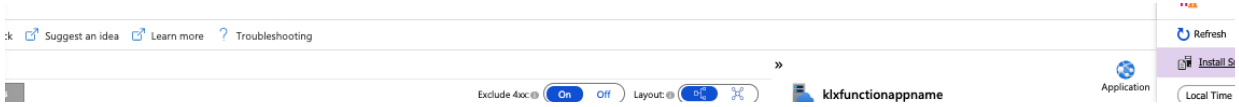
```
module.exports = async function (context, req) {
  context.log('v2 JavaScript HTTP trigger function process
   context.bindings.outputQueueItem="Hi there";
   context.bindings.outputBlob = "Hi there"
   throw "Oh damn and blast.."

  context.res = {
    body: "Hello from azure functions!"
  };
  context.done();
};
```

Invoke the function as before from the "CallMyService" in your applications index.html.

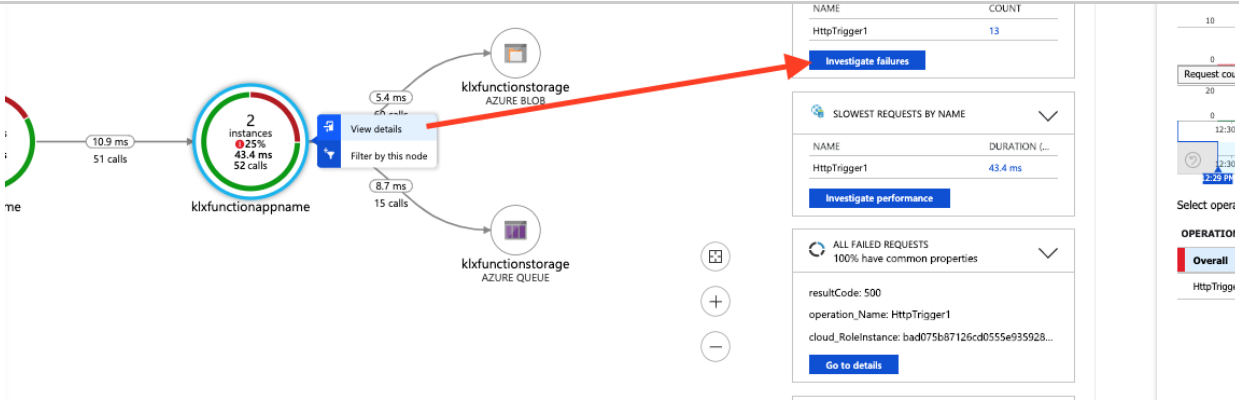
up with the cloud"



Give the application map a few minutes to catch up and show a red error status at the function node.



up with the cloud"



You need to search around the failures, when you can find the "Damn and blast" message 🙂 It will also show you the exact line of javascript where this was thrown.

Try calling the Azure function directly from your web page.

Comment out the error from your Azure Function

```
//throw "Oh damn and blast.."
```

Modify your **index.html** to call your function directly (rather than "greetings").

Replace the YOUR FUNCTION ENDPOINT with your function's url:

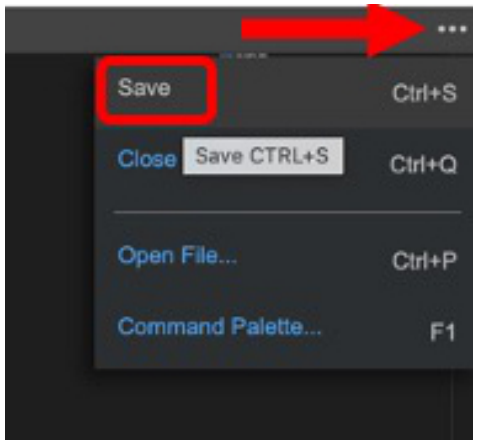
```
<script>
    $( '#callmyservice' ).click(function() {
      var url = "YOUR FUNCTION ENDPOINT";// For example "h
      $.get(url)
      .done(function(data) {
        $( '#message' ).text(data).show();
```

```
        })
        .fail(function(error) {
```

up with the cloud"

```
    });
</script>
```

And save it:



Build and deploy the microservice from the command line:

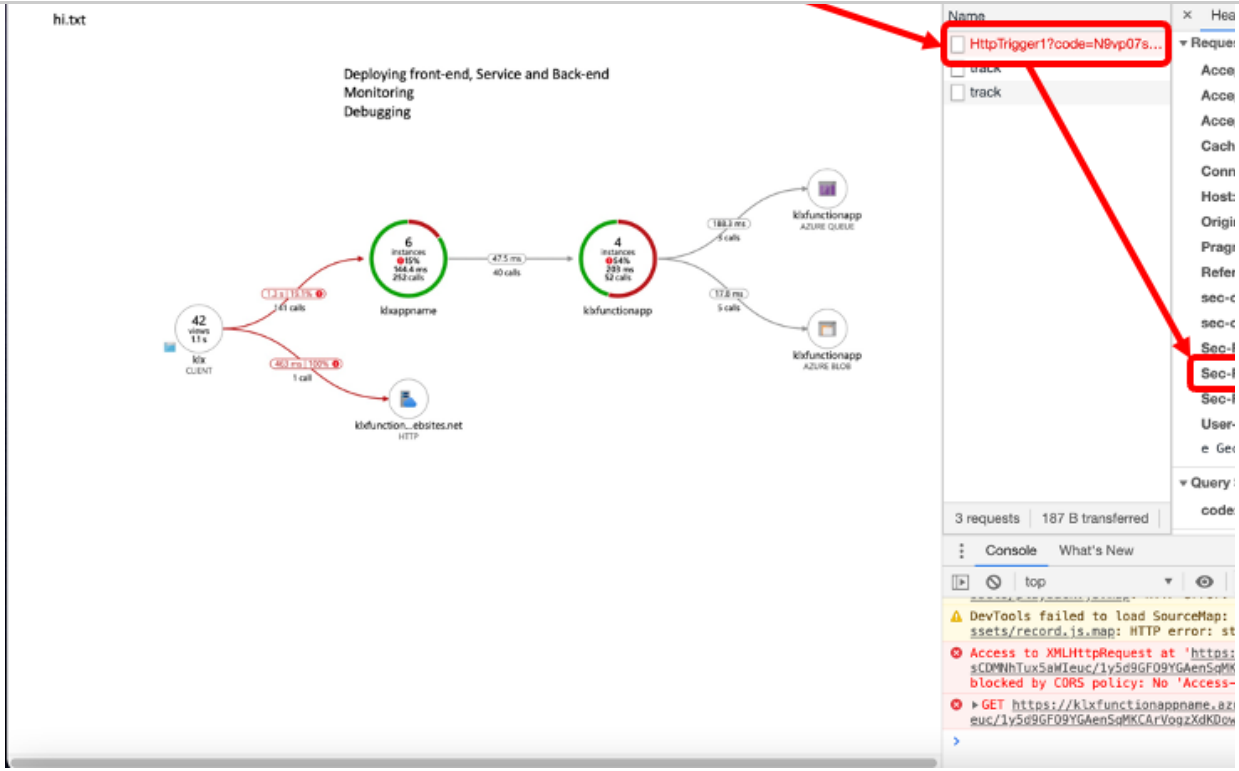
> mvn clean install

> mvn azure-webapp:deploy

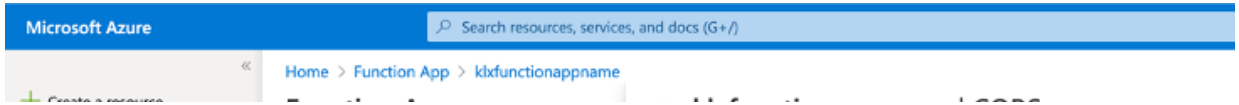
Once deployed, refresh your webservice, open View → Developer → Developer Tools → Network and click your "CallMyService" button.

Notice that "CallMyService" button now results in a CORS problem - the function is not accepting calls from your website.

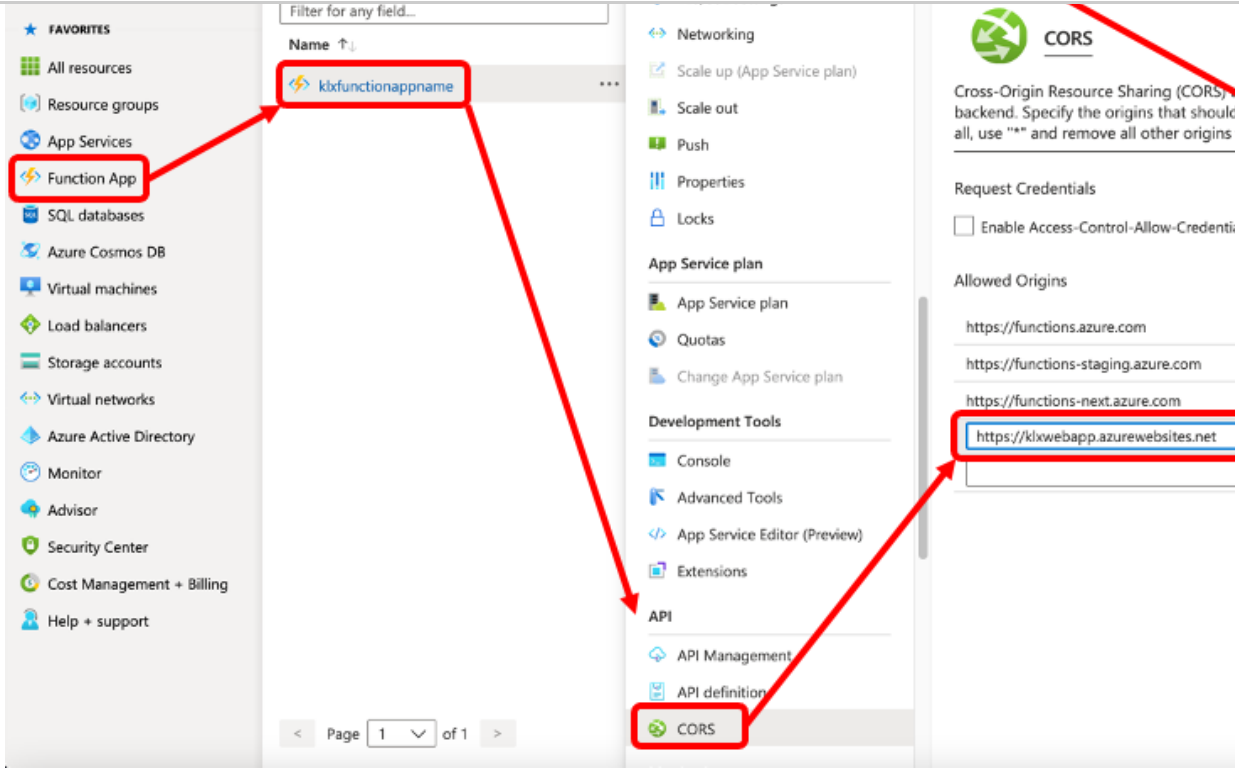
up with the cloud"



To fix that, add your site's address to the list of permitted CORS sites:



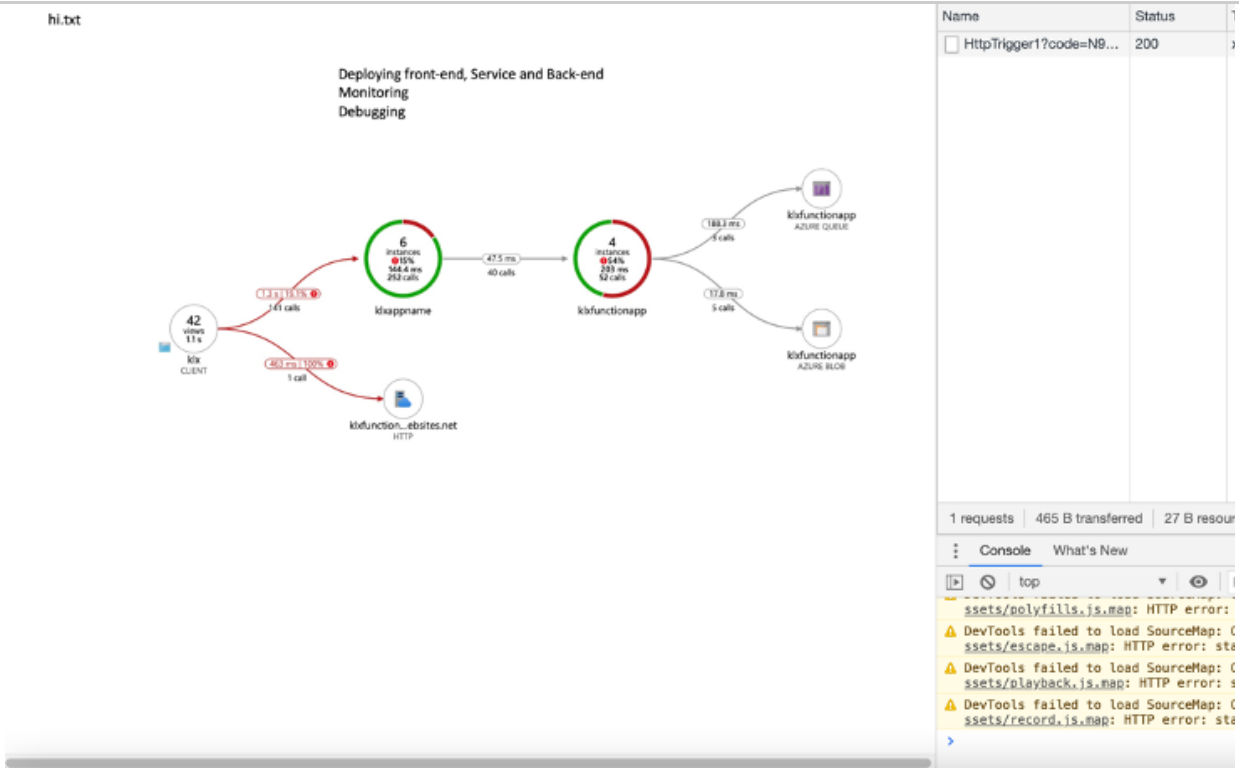
up with the cloud"



Once fixed, refresh your webservice, open View → Developer → Developer Tools → Network and click your "CallMyService" button.

Notice that "CallMyService" button now is calling your function normally.

up with the cloud"



Finally, delete your resourceGroup, which will delete all its resources 🙂

That's it for now.

The next Cloud Lab will build upon this one, with further important Azure features.

Feedback please to ken.lomax@sap.com

Merci!

Ken

up with the cloud"

Over 100 events a year: Jam | Wiki | Blog | Forum
Get involved, expand your skillset, increase your impact.
Missing the invites? Follow this step

No labels