

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

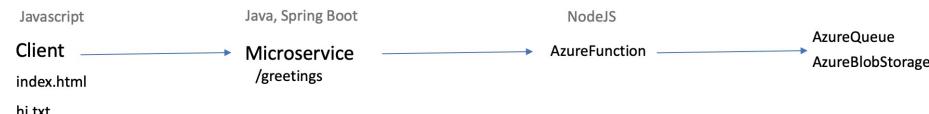
## Journey

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

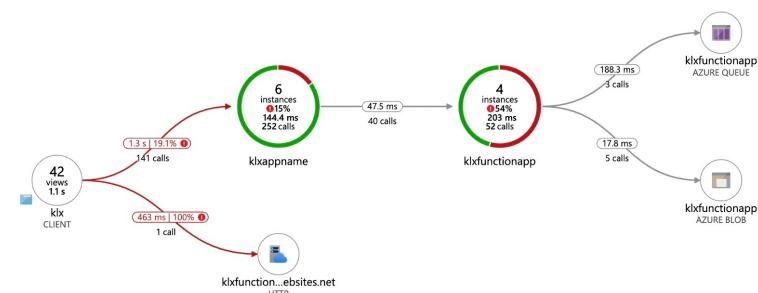
- build and deploy
  - a simple Java, Spring-boot microservice
  - a simple web page that calls this microservice,
  - an Azure Function that writes data to an Azure Queue and Azure Blob
  - modify the microservice to call this Azure Function
- monitor the running system with Application Insights
- resolve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

An Azure Journey within the Azure Portal.

<https://bit.ly/2UsfTTc>



Deploying front-end, Service and Back-end  
Monitoring  
Debugging



## Video of Journey

We want to keep this Cloud Labs correct. Please report any problems you have to [ken.lomax@sap.com](mailto: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



"I up with the cloud"

A screenshot of a video player interface. At the top is a thumbnail image of a person wearing glasses and headphones, looking thoughtful with a hand to their chin. Below the thumbnail is a large white play button icon. To the right of the play button is the text "SAP Media Share" with a yellow dropdown arrow. Below the thumbnail is a horizontal control bar. From left to right, it includes a play/pause button, the time "0:00", a closed captioning (CC) button, a "1x" speed button, and a share icon.

## Prerequisites

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 **k1x**, 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**

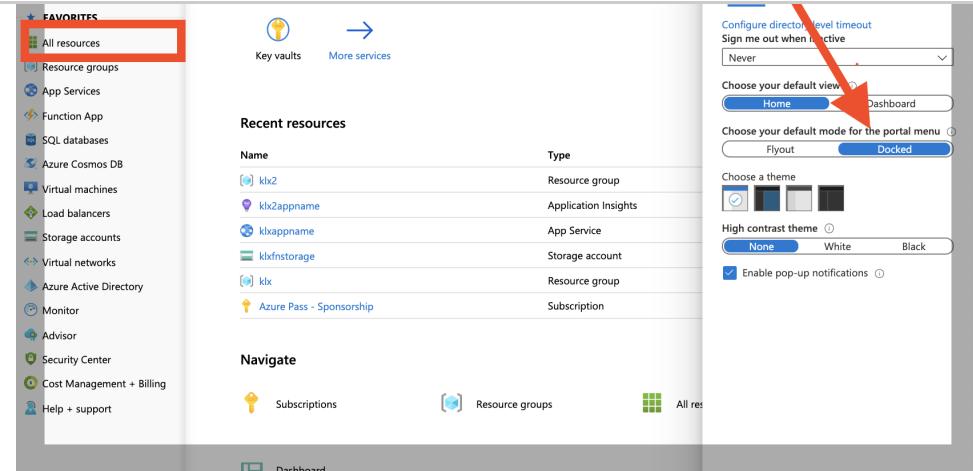
**Pertinent  
Links**

## Open and set up your Azure Portal

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



"Up with the cloud"



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



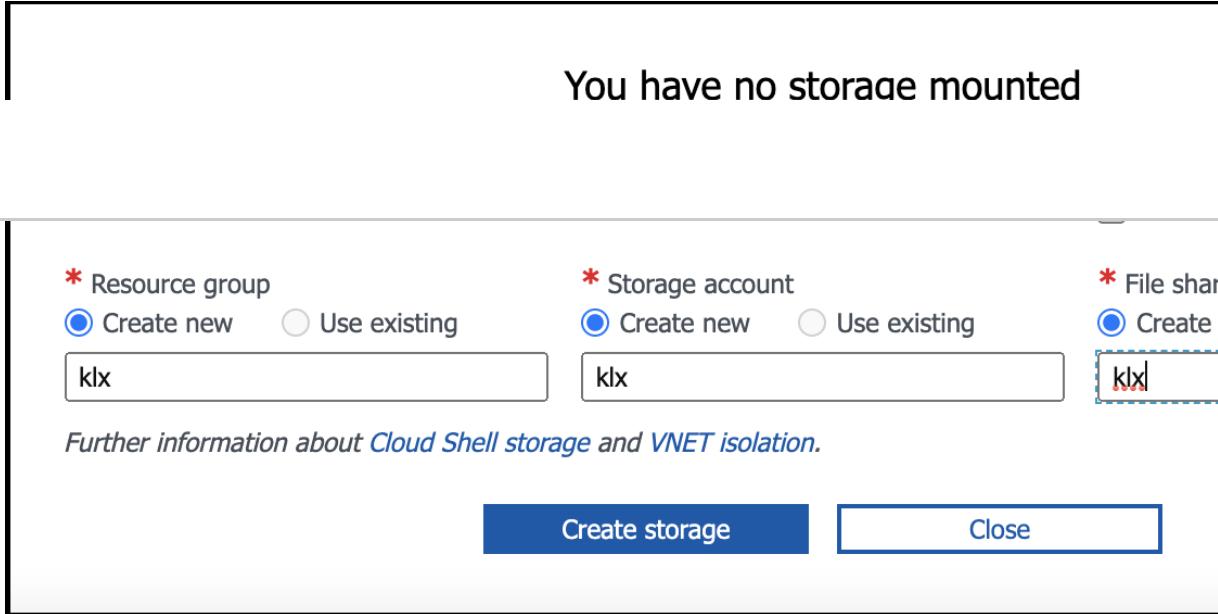
- A pop up will tell you "You have no storage mounted". This makes sense. The CLI needs some storage to store the files that you create..
- -> "Show advanced settings" and create a new Resource group, a new Storage account and a new File Share, all with the name **klx**.
- -> "Create Storage"

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

[Mounting, Unmounting clouddrives](#)

[Azure CLI](#) for running outside of the portal.

"Get 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

```
ken@Azure:~$ ls -l
total 0
lrwxrwxrwx 1 ken ken 22 Apr  3 10:11 clouddrive -> /usr/csuser
```

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.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
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

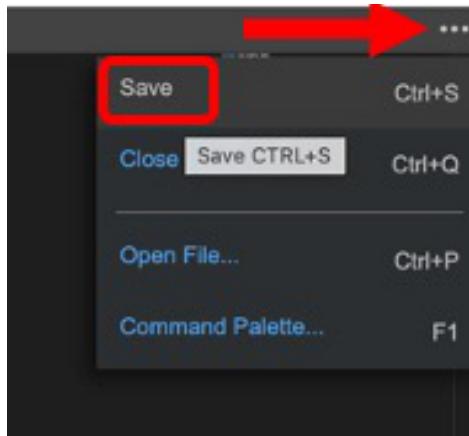
<html>
<head>
    <script src='https://ajax.aspnetcdn.com/ajax/jQuery/jque
</head>
<body>
    <button id='callmyservice'>CallMyService</button>

    <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>
```

```
  
</body>
```

"up with the cloud"

And save it:

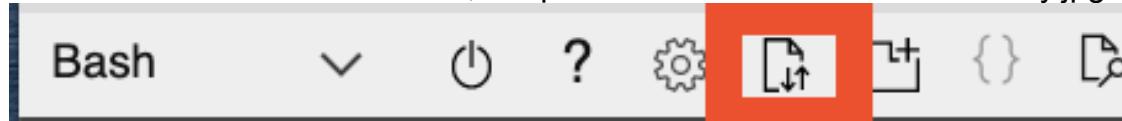


This code has a button that when clicked, calls your endpoint. The resulting response is displayed in the "message" div.

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

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

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

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

Deploying a Spring Boot app to Azure

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

```
</properties>
```

"up with the cloud"

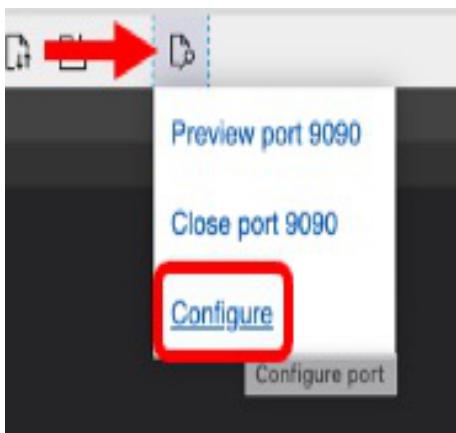


Build and run the microservice from the command line. As port 80 and 8080 are often taken, use 9090 instead:

> mvn clean install

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

Click on Configure to open the port 9090:

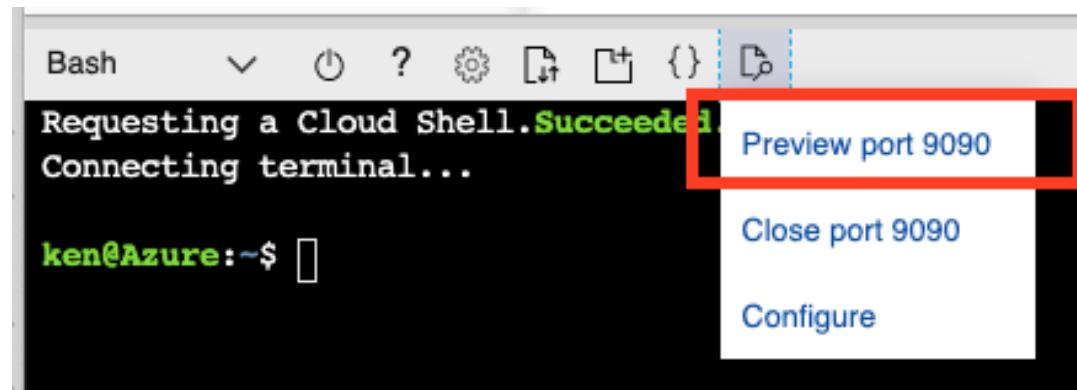


Type 9090 and confirm it:

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

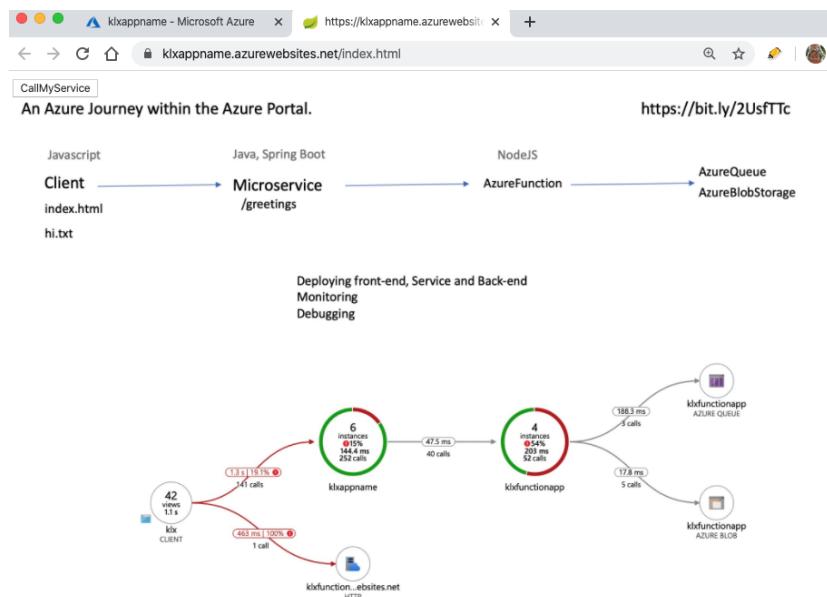
Find and access the  
deployed webservice

The screenshot shows the Microsoft Azure portal's 'All resources' page. At the top, there's a search bar and a navigation bar with options like 'Swap', 'Restart', 'Delete', and a note about browser changes. Below the navigation is a sidebar with links for 'Resource groups', 'App Services', 'Function App', 'SQL databases', and 'Azure Cosmos DB'. The main area lists resources with columns for 'Name', 'Type', 'Resource group', 'Location', and 'Subscription'. One resource, 'kxappname', is highlighted with a red box.

"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/>

"I up with the cloud"

The screenshot shows two side-by-side views of the Microsoft Azure portal. The left view is the 'All resources' blade, displaying a list of resources including Storage accounts, App Services, Function Apps, and others. The right view is the 'Application Insights' blade, showing the settings for an Application Insights resource named 'kxappname'. A red arrow points from the 'Application Insights' blade back to the 'All resources' blade, indicating where to select the resource for configuration.

And apply it:

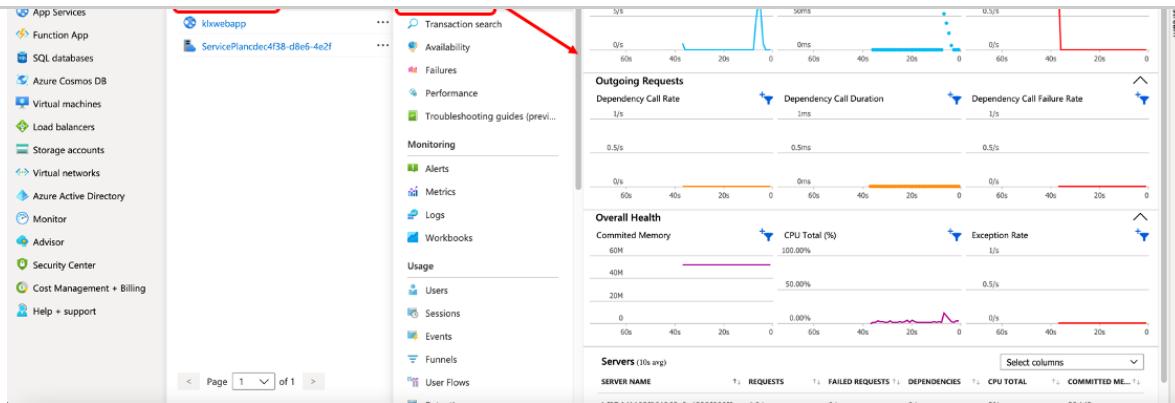
The screenshot shows the 'Application Insights [Preview]' blade. It has a 'Enable' button (which is highlighted with a red box) and a 'Disable' button. Below these are sections for 'Apply monitoring settings' and 'Feedback'. A modal dialog box is open, asking if the user wants to continue applying changes. The 'Yes' button in this dialog is also highlighted with a red box.

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

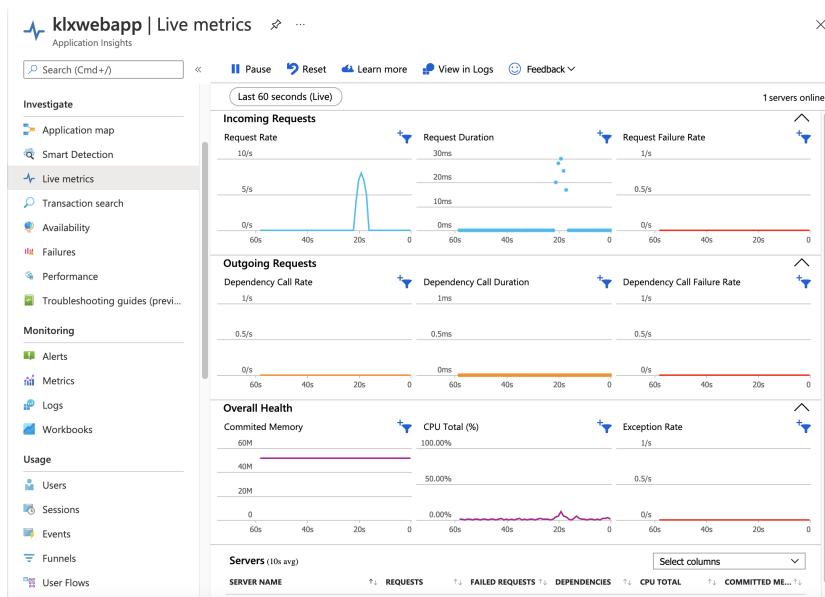
The screenshot shows the Microsoft Azure Application Insights dashboard for a resource named 'klixwebapp'. The left sidebar lists various Azure services. The main area displays 'Live metrics' for 'klixwebapp'. Key sections include:

- Outgoing Requests:** Shows Dependency Call Rate (1/s), Dependency Call Duration (1ms), and Dependency Call Failure Rate (0.5/s).
- Overall Health:** Shows Committed Memory (60M), CPU Total (%) (100.00%), and Exception Rate (1/s).
- Servers:** Shows REQUESTS, FAILED REQUESTS, DEPENDENCIES, CPU TOTAL, and COMMITTED MEMORY over time.

"I 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

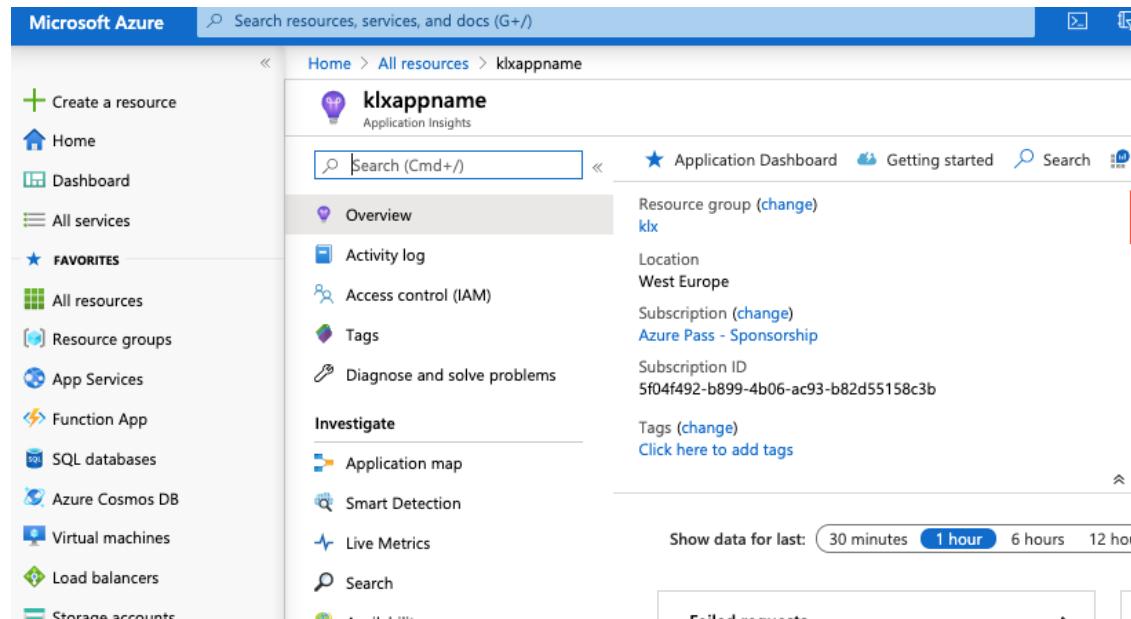
Instrumentalize your **index.html** page so it can too be monitored:

<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:



The screenshot shows the Microsoft Azure portal interface. On the left, there is a sidebar with various service icons: Create a resource, Home, Dashboard, All services, FAVORITES (with All resources selected), Resource groups, App Services, Function App, SQL databases, Azure Cosmos DB, Virtual machines, Load balancers, and Storage accounts. The main content area is titled "klxapppname" under the Application Insights category. It displays the following details:

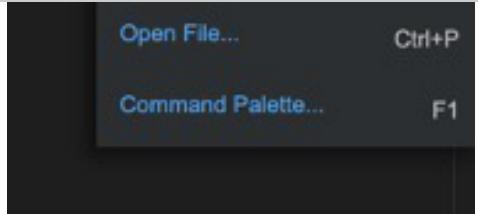
- Resource group: (change) klx [highlighted by a red bracket]
- Location: West Europe
- Subscription: (change) Azure Pass - Sponsorship
- Subscription ID: 5f04f492-b899-4b06-ac93-b82d55158c3b
- Tags: (change) Click here to add tags

At the bottom, there is a "Show data for last:" dropdown set to "1 hour".

And save it:



"Get up with the cloud"



Redeploy the service with the newly instrumentalized front page:

> mvn clean install

> mvn azure-webapp:deploy

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

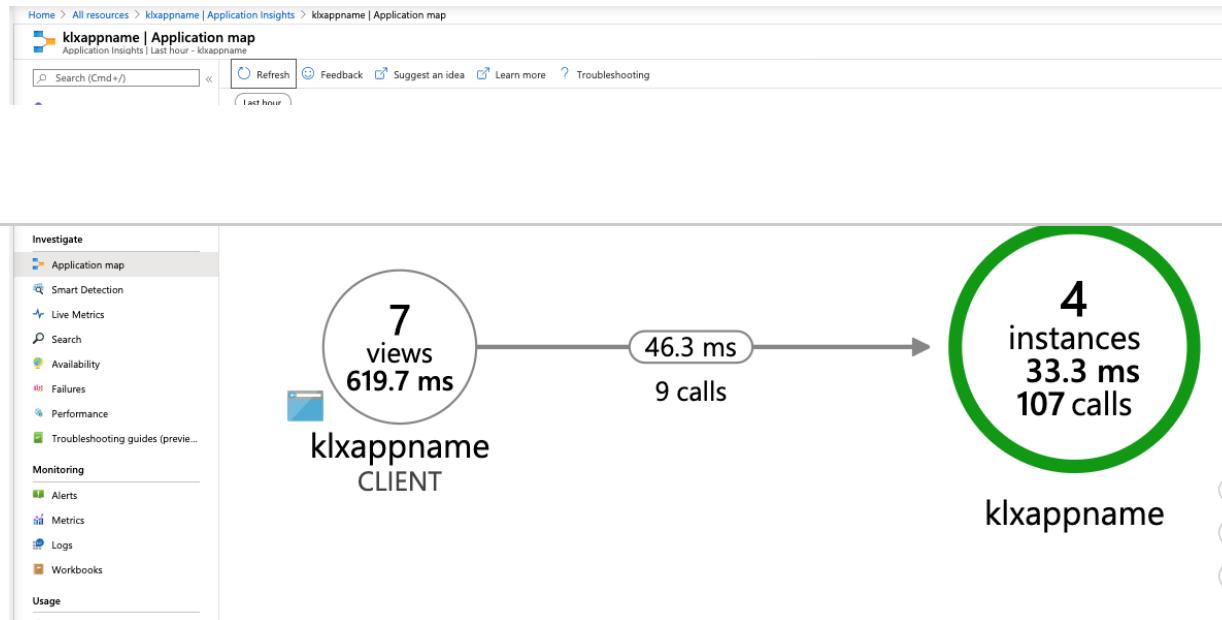
Wait for the new deployment and be sure your browser reloads this latest version (with the instrumentation key in the front page). Refresh your website several times, and click your website's "CallMyService" button a few times, so that Azure can track these calls.

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:

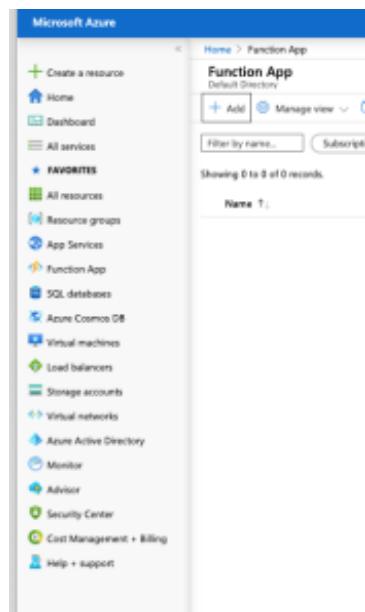
View your Application  
Map

"up with the cloud"



Extend your microservice 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: kixfunctionappname
  - 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 (Serverless)
  - 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:

"Get up with the cloud"

The screenshot shows the Microsoft Azure portal interface. At the top, there's a blue header bar with the text "Microsoft Azure" and a search bar that says "Search resources, services, and docs (G+/)". Below the header, the breadcrumb navigation shows "Home > klxfunctionappname". On the left, there's a sidebar titled "FAVORITES" containing links to various Azure services like All resources, Resource groups, App Services, Function App, SQL databases, etc. The main content area is titled "klxfunctionappname" and shows the "Functions" blade. This blade includes sections for Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, Events (preview), and Functions. The "Functions" link in the sidebar is highlighted with a red arrow pointing to it. Below these sections is a table with columns for "Name" and "Trigger", both with an upward arrow icon. The table displays the message "No results.".

"Get up with the cloud"

The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with various service icons like All services, Favorites, All resources, Resource groups, App Services, Function App, SQL databases, Azure Cosmos DB, Virtual machines, Load balancers, Storage accounts, Virtual networks, Azure Active Directory, Monitor, Advisor, Security Center, Cost Management + Billing, and Help + support. The 'Function App' icon is selected. In the center, the 'Overview' tab is active under the 'Developer' section. A red box highlights the 'Code + Test' button. To the right, the 'Essentials' panel displays the following details:

Function app	:	klxfunctionappname
Status	:	Enabled
Resource group (change)	:	cloudlab09
Subscription (change)	:	sap-marcelogomes
Subscription ID	:	83644533-fa26-47e9-a270-7bea6c1d6785

Below this, there are two charts: 'Total Execution Count' and 'Successful Execution Count'. Both charts show a single data point at 7:45 PM UTC+02:00 with a value of 4.

Total Execution Count

4.5
4
3.5
3
2.5
2
1.5
1
0.5
0

7:15 PM    7:30 PM    7:45 PM    UTC+02:00

HttpTrigger1 Count (Sum)  
klxfunctionappname  
4

Successful Execution Count

4.5
4
3.5
3
2.5
2
1.5
1
0.5
0

HttpTrigger1  
klxfunctionappname  
4

It might help to watch the video for that part.

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

"Get up with the cloud"

The screenshot shows the Microsoft Azure portal interface. At the top, there's a blue header bar with the text "Microsoft Azure" and a search bar that says "Search resources, services, and docs (G+/)". Below the header, the breadcrumb navigation shows "Home > HttpTrigger1".

The main area has a sidebar on the left titled "FAVORITES" containing links like All services, All resources, Resource groups, App Services, Function App, SQL databases, Azure Cosmos DB, Virtual machines, Load balancers, Storage accounts, Virtual networks, Azure Active Directory, Monitor, Advisor, Security Center, Cost Management + Billing, and Help + support.

The central area is titled "Overview" and shows the path "klxfunctionappname \ HttpTrigger1 \ index.js". It has tabs for "Developer" (which is selected), "Code + Test" (highlighted in blue), "Integration", "Monitor", and "Function Keys".

The "Developer" tab displays the code for "index.js":

```
1 module.exports = async function (context, req) {
2   context.log('v2 JavaScript HTTP trigger function processed a
3   // throw "Oh damn and blast.."
4   // context.bindings.outputQueueItem="Hi there";
5   // context.bindings.outputBlob = "Hi there"
6   ...
7   context.res = {
8     body: "Hello from azure functions!"
9   };
10  context.done();
11 }
```

Below the code editor, there are two tabs: "Logs" (selected) and "Filesystem". The "Logs" tab shows the output:

```
Connected!
2021-04-01T18:01:09 Welcome, you are now connected to log
the App Setting SCM_LOGSTREAM_TIMEOUT (in seconds).
```

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.

"I'm up with the cloud"

The screenshot shows the Microsoft Azure portal interface. At the top, there's a navigation bar with 'Microsoft Azure' and a search bar. Below the navigation bar, the URL 'Home > HttpTrigger1' is visible. On the left, there's a sidebar titled 'FAVORITES' containing various Azure service icons. The main area is titled 'Developer' and has tabs for 'Code + Test' (which is selected), 'Integration', 'Monitor', and 'Function Keys'. The 'Code + Test' tab shows the file path 'kixfunctionappname \ HttpTrigger1 \ index.js'. The code editor displays the following JavaScript code:

```
1 module.exports = async function (context, req) {
2   context.log('v2 JavaScript HTTP trigger function processed a request');
3   // throw "Oh damn and blast.."
4   // context.bindings.outputQueueItem="Hi there";
5   // context.bindings.outputBlob = "Hi there"
6   ...
7   context.res = {
8     ... body: "Hello from azure functions!"
9   };
10  context.done();
11 }
```

A large red arrow points from the code editor down to the log stream at the bottom. The log stream window has a title bar with '... Filesystem Logs Log Level Stop Copy Close' and a message area. The message area contains the following text:

Connected!  
2021-04-01T18:01:09 Welcome, you are now connected to log-streaming service. The default timeout is 2 hours.  
Change the timeout with the App Setting  
SCM\_LOGSTREAM\_TIMEOUT (in seconds).

"Get up with the cloud"

The screenshot shows the Microsoft Azure portal interface. At the top, there's a blue header bar with the text "Microsoft Azure" and a search bar that says "Search resources, services, and docs (G+)". Below the header, the URL "Home > HttpTrigger1" is visible. On the left, there's a sidebar titled "FAVORITES" containing various Azure service icons. The main area is titled "Developer" and has tabs for "Code + Test" (which is selected), "Integration", "Monitor", and "Function Keys". To the right of the tabs, there's a code editor window showing the file "index.js" with the following code:

```
1 module.exports = async function (context, req) {
2   context.log('v2 JavaScript HTTP trigger function processed a'
3   // throw "Oh damn and blast.."
4   // context.bindings.outputQueueItem="Hi there";
5   // context.bindings.outputBlob = "Hi there"
6   ...
7   context.res = {
8     body: "Hello from azure functions!"
9   };
10  context.done();
11 }
```

Below the code editor is a log viewer window titled "Filesystem Logs". It displays the following log entries:

```
2021-04-01T18:02:09 [Information] Executing
'Functions.HttpTrigger1' [Reason='This function was
programmatically called via the host APIs.'],
Id=96e2f9a9-f52e-4962-84df-2fc5a9ff5815)
2021-04-01T18:02:19.105 [Information] v2 JavaScript HTTP
trigger function processed a request.
2021-04-01T18:02:19.108 [Information] Executed
'Functions.HttpTrigger1' (Succeeded, Id=96e2f9a9-f52e-
4962-84df-2fc5a9ff5815, Duration=52ms)
```

Call this function from  
your webservice

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

"Get up with the cloud"

The screenshot shows the Microsoft Azure portal interface. At the top, there's a blue header bar with the text "Microsoft Azure" and a search bar that says "Search resources, services, and docs (G+/)". Below the header, the URL "kxfunctionappname \ HttpTrigger1 \ index.js" is visible. On the left, there's a sidebar titled "FAVORITES" containing links to various Azure services like All resources, Resource groups, App Services, Function App, etc. The main content area is titled "Overview" and has a "Developer" section with tabs for "Code + Test" (which is selected), Integration, Monitor, and Function Keys. Under "Code + Test", the file "index.js" is displayed with the following code:

```
1 module.exports = async function (context, req) {
2   context.log('v2 JavaScript HTTP trigger function processed a request');
3   // throw "Oh damn and blast."
4   // context.bindings.outputQueueItem="Hi there";
5   // context.bindings.outputBlob = "Hi there"
6   ...
7   context.res = {
8     body: "Hello from azure functions!"
9   };
10  context.done();
11 }
```

Below the code editor, there's a "Logs" section with a "Connected!" message and a log stream showing several entries:

```
Connected!
2021-04-01T18:01:09 Welcome, you are now connected to log-stream...
with the App Setting SCM_LOGSTREAM_TIMEOUT (in seconds).
2021-04-01T18:02:09 No new trace in the past 1 min(s).
2021-04-01T18:02:19.057 [Information] Executing 'Functions.HttpTrigger1' host APIs., Id=96e2f9a9-f52e-4962-84df-2fc5a9ff5815
2021-04-01T18:02:19.105 [Information] v2 JavaScript HTTP trigger...
2021-04-01T18:02:19.108 [Information] Executed 'Functions.HttpTrigger1' Duration=52ms
```

"I'm up with the cloud"

The screenshot shows the Microsoft Azure portal interface. At the top, there's a navigation bar with 'Microsoft Azure' and a search bar. Below it, the URL 'Home > HttpTrigger1' is visible. A 'Create a resource' button is on the left. The main area has a sidebar titled 'FAVORITES' with various service icons like All resources, Resource groups, App Services, Function App, etc. The 'Developer' section is selected, showing 'Code + Test' as the active tab. To its right is the code editor for 'index.js' in the 'HttpTrigger1' function. The code is as follows:

```
1 module.exports = async function (context, req) {
2   context.log('v2 JavaScript HTTP trigger function');
3   // throw "Oh dear, an error occurred";
4   // context.bindings.outputQueueItem="Hi there";
5   // context.bindings.outputBlob = "Hi there"
6   ...
7   context.res = {
8     body: "Hello from azure functions!"
9   };
10  context.done();
11 }
```

To the right of the code editor is a 'Get function URL' section with a 'Key' input field containing 'default'. A red arrow points to this field. Below the code editor is the 'Logs' panel, which displays log messages:

```
Connected!
2021-04-01T18:01:09 Welcome, you are now connected to log-stream
with the App Setting SCM_LOGSTREAM_TIMEOUT (in seconds).
2021-04-01T18:02:09 No new trace in the past 1 min(s).
2021-04-01T18:02:19.057 [Information] Executing 'Functions.HttpTrigger1'
host APIs.', Id=96e2f9a9-f52e-4962-84df-2fc5a9ff5815)
2021-04-01T18:02:19.105 [Information] v2 JavaScript HTTP trigger
2021-04-01T18:02:19.108 [Information] Executed 'Functions.HttpTrigger1'
Duration=52ms)
```

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

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

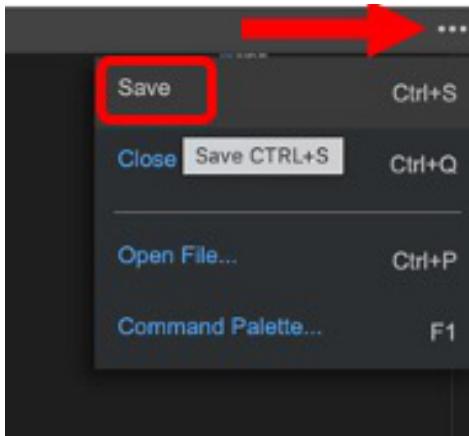
### DemoApplication.java

```
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"

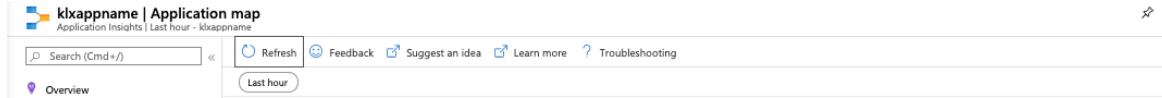
And save it:



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:

"I'm up with the cloud"

The screenshot shows the Microsoft Azure portal interface. At the top, there's a blue header bar with the text "Microsoft Azure" and a search bar that says "Search resources, services, and docs (G+/-)". Below the header, the breadcrumb navigation shows "Home > Function App > kxfunctionappname".

The left sidebar lists various Azure services under "All services" and "FAVORITES". The "Function App" service is selected, indicated by a red arrow pointing to it.

The main content area displays the "kxfunctionappname" function app. It includes a "Filter for any field..." search bar and a "Name ↑↓" filter. The function name "kxfunctionappname" is highlighted with a red arrow. To the right of the function name is a three-dot menu icon.

The right pane contains several sections:

- Overview**: Includes links to Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, and Events (preview).
- Functions**: This section is expanded, showing the "Functions" list with "HttpTrigger1" highlighted by a red arrow. Other items in this list include App keys, App files, and Proxies.
- Deployment**: Includes links to Deployment slots, Deployment Center, and Deployment Center (Classic).
- Settings**: Includes Configuration and Authentication.

"Get up with the cloud"

Microsoft Azure

Search resources, services, and docs (G+/)

Home > HttpTrigger1

Create a resource

**FAVORITES**

- All resources
- Resource groups
- App Services
- Function App
- SQL databases
- Azure Cosmos DB
- Virtual machines
- Load balancers
- Storage accounts
- Virtual networks
- Azure Active Directory
- Monitor
- Advisor
- Security Center
- Cost Management + Billing
- Help + support

Developer

Integration

Edit the trigger and choose from a selection of inputs and outputs for your function, in

**Trigger**

HTTP (req)

**Function**

HttpTrigger1

**Inputs**

No inputs defined

+ Add input

```
graph LR; Trigger[Trigger HTTP (req)] --- Function[Function HttpTrigger1];
```

"Get up with the cloud"

Microsoft Azure

Search resources, services, and docs (G+/)

Create a resource

Home > HttpTrigger1

FAVORITES

- All resources
- Resource groups
- App Services
- Function App
- SQL databases
- Azure Cosmos DB
- Virtual machines
- Load balancers
- Storage accounts
- Virtual networks
- Azure Active Directory
- Monitor
- Advisor
- Security Center
- Cost Management + Billing
- Help + support

Developer

Integration

Trigger

HTTP (req)

Function

HttpTrigger1

Inputs

No inputs defined

+ Add input

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:

"Get up with the cloud"

The screenshot shows the Microsoft Azure portal interface. At the top, there's a blue header bar with the text "Microsoft Azure" and a search bar that says "Search resources, services, and docs (G+/-)". Below the header, the URL "Home > HttpTrigger1" is visible. On the left, there's a sidebar titled "FAVORITES" containing links to various Azure services like All resources, Resource groups, App Services, Function App, SQL databases, etc. The main content area is titled "Overview" and has tabs for "Developer" (selected), "Integration" (highlighted with a grey background), "Code + Test", "Monitor", and "Function Keys". Under the "Integration" tab, there's a diagram showing a "Trigger" (HTTP (req)) connected to a "Function" (HttpTrigger1). Below the trigger, there's a box labeled "Inputs" with the sub-label "No inputs defined" and a "+ Add input" button. To the right of the "Function" box, there's some descriptive text: "Edit the trigger and choose from a selection of inputs and outputs for your function, if any are available." The overall layout is clean and modern, typical of the Azure developer portal.

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 = {
```

"I'm up with the cloud"

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

Save and run this function a few times.

The screenshot shows the Microsoft Azure portal interface for a Function App named "HttpTrigger1".

- Left Sidebar:** Lists various Azure services like Home, Dashboard, All services, Favorites, All resources, Resource groups, App Services, Function App, SQL databases, Azure Cosmos DB, Virtual machines, Load balancers, Storage accounts, Virtual networks, Azure Active Directory, Monitor, Advisor, Security Center, Cost Management + Billing, and Help + support.
- Top Bar:** Shows the search bar "Search resources, services, and docs (G+ /)" and the current location "Dashboard > Function App > klxfunctionappname > HttpTrigger1".
- Code Editor:** Displays the function code:

```
module.exports = async function (context, req) {
  context.log('v2 JavaScript HTTP trigger function processed');
  //throw "Oh damn and blast.."
  context.bindings.outputQueueItem="Hi there";
  context.bindings.outputBlob = "Hi there";
  ...
  context.res = {
    ...body: "Hello from azure functions!"
  };
  context.done();
};
```
- Toolbar:** Includes "Save" and "Test/Run" buttons, both of which are highlighted with red boxes. A large red arrow points from the "Test/Run" button down to the log viewer.
- Log Viewer:** Shows the function logs:

```
2021-04-07T09:42:51 Welcome, you are now connected to
log-streaming service. The default timeout is 2 hours.
Change the timeout with the App Setting
SCM_LOGSTREAM_TIMEOUT (in seconds).
2021-04-07T09:42:41.724 [Information] Executing
'Functions.HttpTrigger1' (Reason='This function was
programmatically called via the host APIs.', 
Id=cc0e5bd7-d09b-4c92-a301-bdbf28a8be3d)
2021-04-07T09:42:41.764 [Information] v2 JavaScript
HTTP trigger function processed a request.
2021-04-07T09:42:41.919 [Information] Executed
```

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

"I'm up with the cloud"

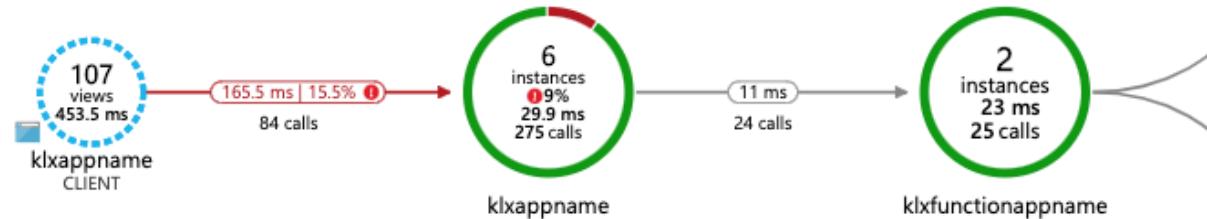
The screenshot shows the Microsoft Azure portal interface. On the left, the navigation menu is visible with various service categories like Dashboard, All services, Favorites, All resources, Resource groups, App Services, Function App, SQL databases, Azure Cosmos DB, Virtual machines, Load balancers, Storage accounts (which is selected and highlighted with a red box), Virtual networks, Azure Active Directory, Monitor, Advisor, Security Center, Cost Management + Billing, and Help + support. In the center, the main content area is titled "Storage accounts" with a sub-section for "Blob service". A search bar at the top of this section contains the placeholder "Filter for any field..." and has a red box around it. Below the search bar, there is a list of blob-related items: Containers, Custom domain, Data protection, Object replication, Azure CDN, Add Azure Search, Lifecycle management, Blob inventory (preview), and a "... more" button. To the right of the blob service, there is a sidebar titled "Essentials" containing resource group information (Resource group: kx, Status: Primary: Available, Location: West Europe, Subscription: sap-marcelogomes, Subscription ID: 83644533-fa26-47e9-a270-7bea6c1d6785, Tags: ms-resource-usage : azure-cloud-shell), and links to Container, File shares, Table, Queue, Monitoring, Insights, Alerts, and Metrics services. At the bottom of the central content area, there are four cards: Containers (Scalable, cost-effective storage for unstructured data), File shares (Serverless SMB and NFS file), Tables (Tabular data storage), and Queues (Effectively scale apps according to traffic).

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 "CallMyService" 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

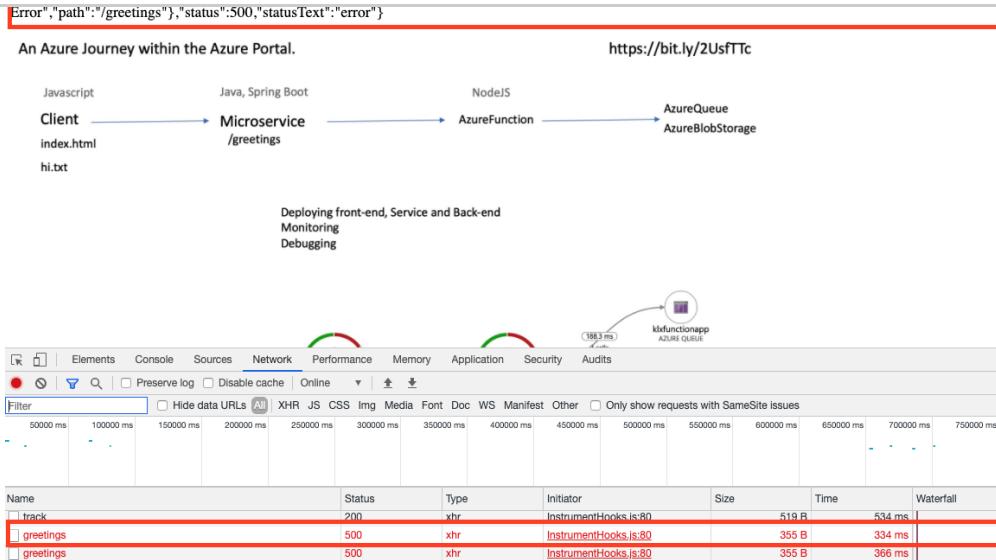
### Azure function

```
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.."

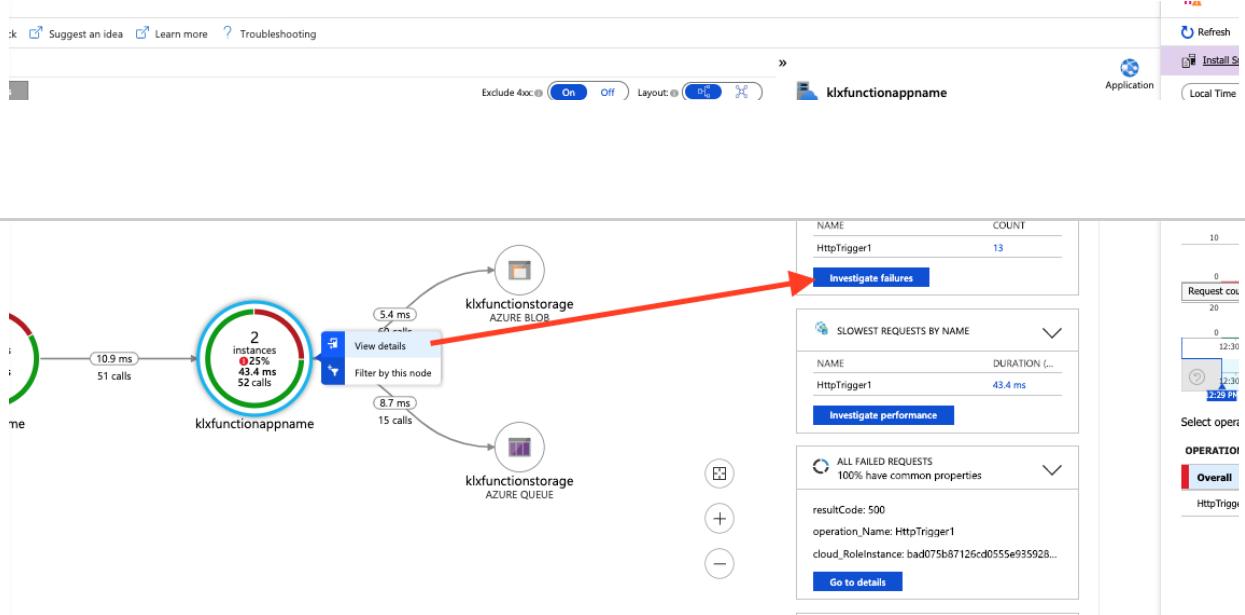
    context.res = {
        body: "Hello from azure functions!"
    };
    context.done();
};
```

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.



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.

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

---

"I'm up with the cloud"

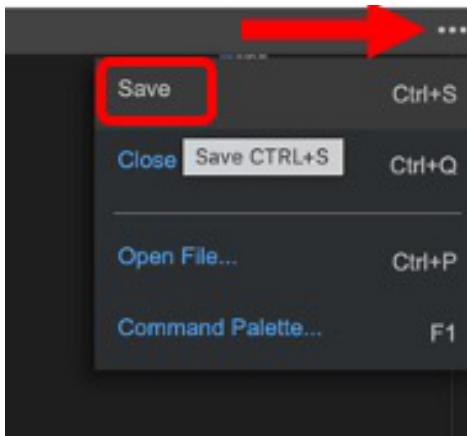
Try calling the Azure function directly from your web page.

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

"up with the cloud"

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

And save it:

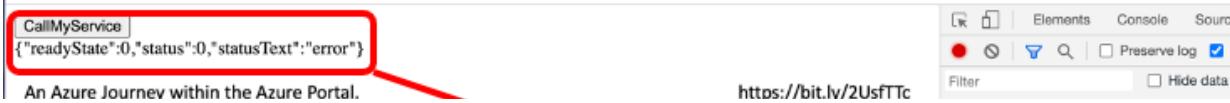


Build and deploy the microservice from the command line:

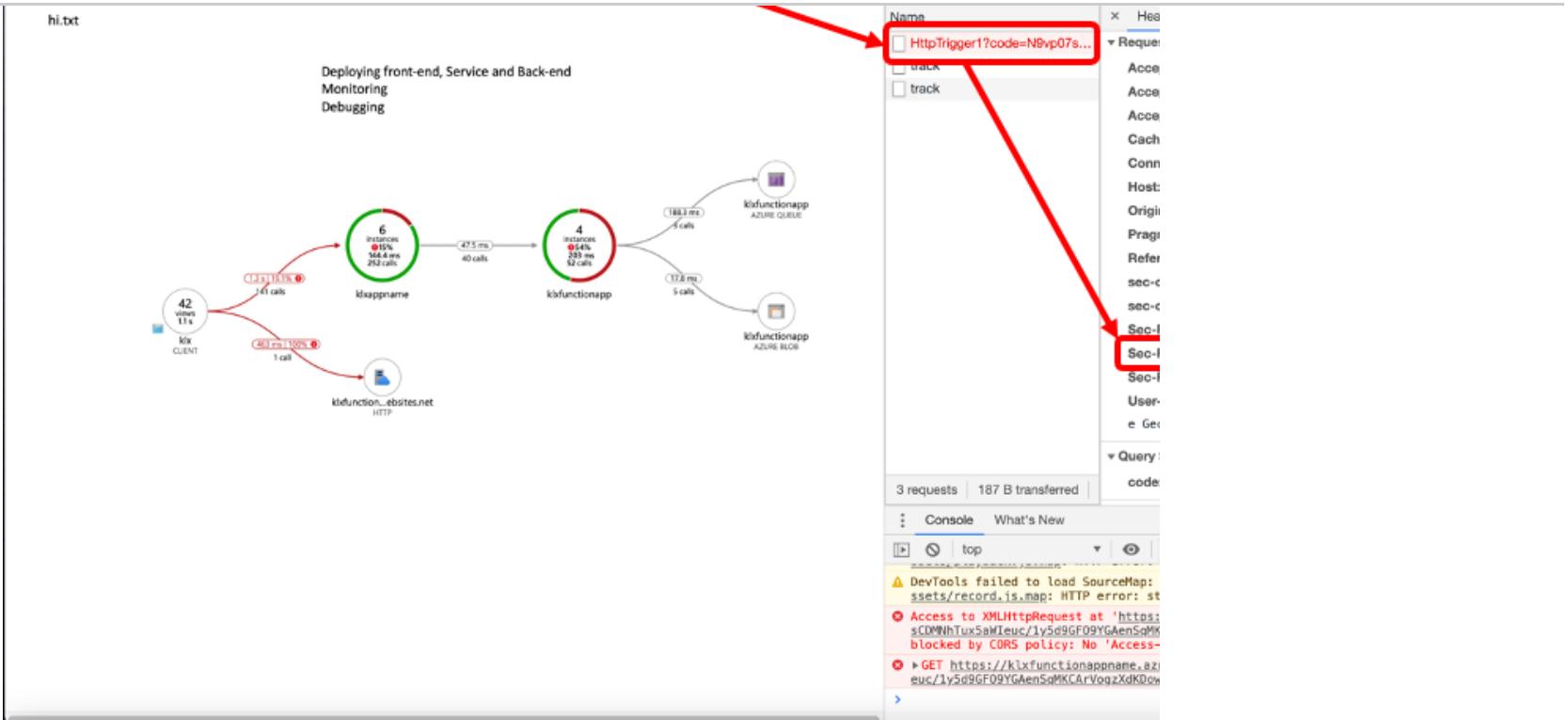
```
> mvn clean install  
> mvn azure-webapp:deploy
```

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"



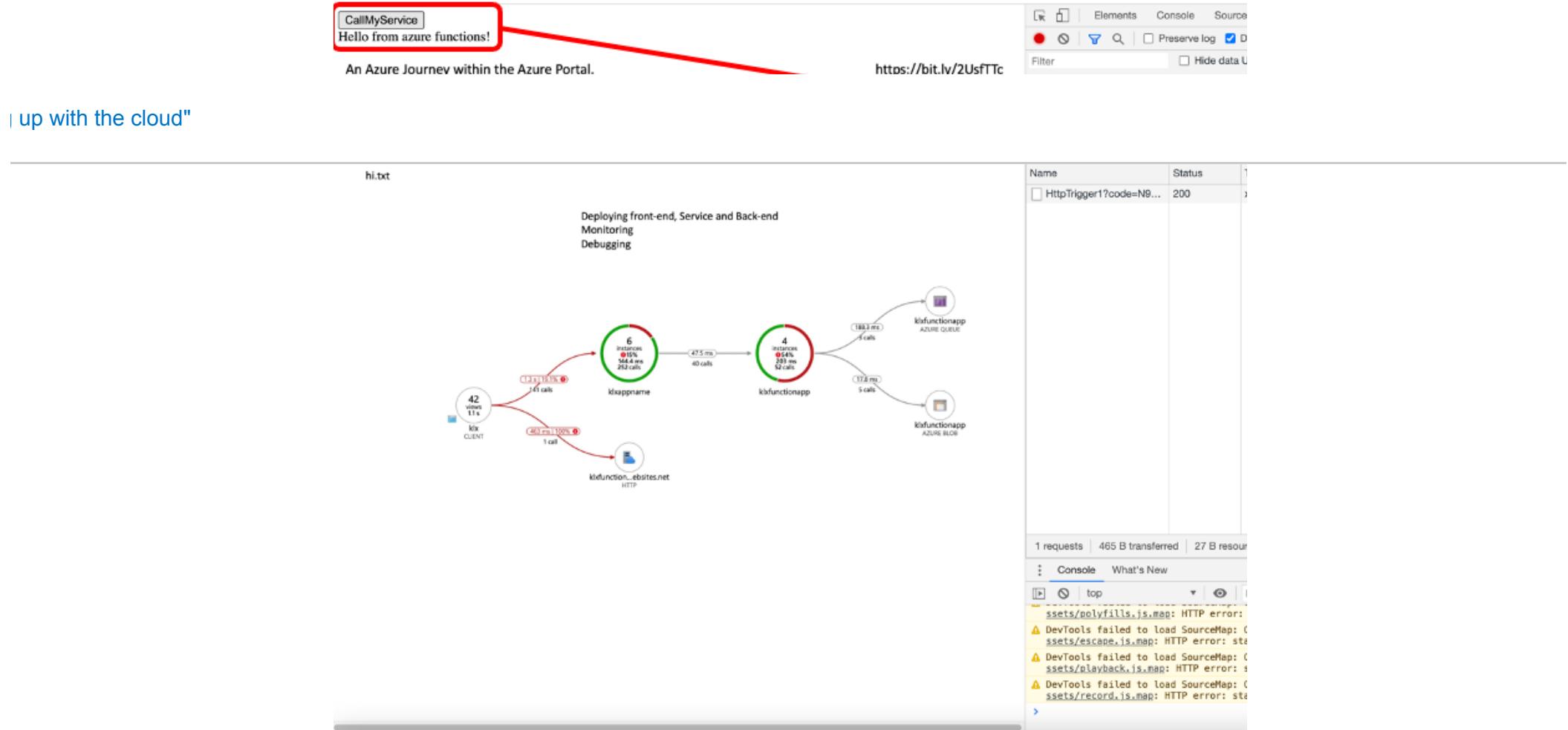
To fix that, add your site's address to the list of permitted CORS sites:

"Get up with the cloud"

The screenshot shows the Microsoft Azure portal interface. On the left, the navigation menu is visible with various service categories like All resources, Resource groups, App Services, and Function App. The 'Function App' item is highlighted with a red box and has a red arrow pointing to the search bar at the top of the main content area. The search bar contains the text 'klxfunctionappname'. The main content area displays a list of items under the heading 'Name ↑'. One item, 'klxfunctionappname', is also highlighted with a red box. To the right of the list, there is a sidebar with several sections: Networking, Scale up (App Service plan), Scale out, Push, Properties, Locks, App Service plan, Quotas, Change App Service plan, Development Tools, Console, Advanced Tools, App Service Editor (Preview), Extensions, API, API Management, API definition, and CORS. The 'CORS' section is highlighted with a red box. Below the 'CORS' section, the text reads: 'Cross-Origin Resource Sharing (CORS) backend. Specify the origins that should all, use "\*" and remove all other origins'. Under 'Request Credentials', there is a checkbox labeled 'Enable Access-Control-Allow-Credentials'. The 'Allowed Origins' section lists three URLs, each with a red box around it: 'https://functions.azure.com', 'https://functions-staging.azure.com', and 'https://functions-next.azure.com'. The bottom of the 'Allowed Origins' section has a text input field containing 'https://klxwebapp.azurewebsites.net', which is also highlighted with a red box. A red arrow points from the 'CORS' button in the sidebar down to the input field.

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.



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](mailto:ken.lomax@sap.com)

Merci!

Ken

"up with the cloud"

OVER 100 EVENTS A YEAR. SUMMITS | BLOG | FORUM

Get involved, expand your skillset, increase your impact.  
Missing the invites? Follow this step

No labels