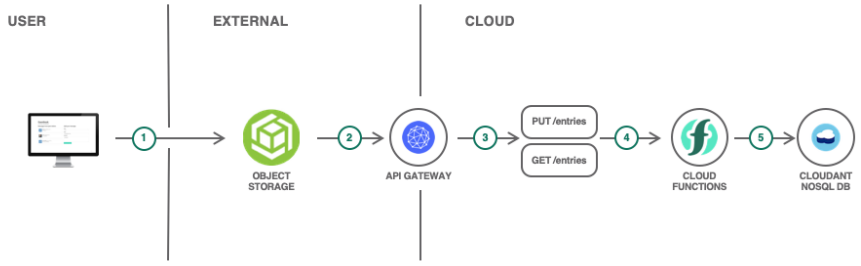
Serverless Web Application and API

Estimated time needed: 1 hour 30 minutes

You will have the opportunity to take your serverless and microservices skills and put them into action in a project that combines these technologies into a single web application. You will create a serverless web application – a simple guestbook website where users can post messages – and you will host the website in two different ways.

Below is the architecture for one version which uses object storage to host the site's files.



Learning Objectives

- Deploy a serverless backend and a database
- Expose a rest API
- Host a static website using object storage
- Deploy the static website as a microservice on Red Hat OpenShift

Troubleshooting

If at any point the IBM Cloud Functions UI doesn't seem up to date, perform a hard refresh of the webpage. In many browsers this is accomplished by clickng on the refresh button while holding the shift key.

Step 1: Deploy a Cloudant database

The guestbook entries will be stored in a Cloudant database for persistence. IBM Cloudant is a fully managed JSON document database built upon and compatible with Apache CouchDB.

1. Go to the IBM Cloud catalog and create an instance of Cloudant.
   Keep in mind the following things when creating your instance:

- The environment should be **multitenant**
- The authentication method should be **IAM and legacy credentials**

Once the Cloudant instance is provisioned (the status is **Active**), click on it to go to the service instance page.

2. Click on **Launch Dashboard** to open the dashboard in a new browser tab.
3. In the upper right, click on **Create Database**. Enter ***guestbook*** as the database name and select **Non-Partitioned** under **Partitioning**. Click **Create** to create the database.
4. Switch back to the browser tab with the service dashboard page. Go to **Service credentials**.
5. Click **New Credential**.
6. Set the name **for-guestbook**, and leave the role as **Manager**. Click **Add** to add the new credential.
7. Expand the newly created credentials and review them. These credentials will allow Cloud Functions actions to read/write to your Cloudant service.

Step 2: Create actions to save guestbook entries

In order for the guestbook to write entries to the Cloudant database and subsequently read entries from the database, you will use Cloud Functions. In this section, you need to create actions with IBM Cloud Functions to write the guestbook entries to Cloudant. The code will be given to you, but you need to create the appropriate actions and sequences to accomplish this task.

A sequence of two actions will be used to create the entries in Cloudant. Given a name, email address, and comment, the sequence will create a document to be persisted and store that document in your Cloudant database.

1. Go to IBM Cloud Functions and create a namespace for this project.
2. Create new Node.js action called `prepare-entry-for-save` in a new package called `guestbook`.
3. The code for this action should be the following:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

```
/**
 * Prepare the guestbook entry to be persisted
 */
function main(params) {
  if (!params.name || !params.comment) {
    return Promise.reject({ error: 'no name or comment'});
  }

  return {
    doc: {
      createdAt: new Date(),
       name: params.name,
       email: params.email,
       comment: params.comment
    }
  };
```

```
}
```

Copied!

Take note of what this action does. It first verifies the parameters passed to this function – it ensures that there is both a name and a comment. Then it returns a JSON string with a document that contains the date as well as the three parameters passed into the function.

4. Add this action to a new sequence called `save-guestbook-entry-sequence`. This sequence should also be in the `guestbook` package.
5. Add a second action to this sequence by clicking the **Add +** button on the sequence page. Use the public `create-document` action from the Cloudant package. You'll need a new package binding, which you should name `binding-for-guestbook`, to bind your Cloudant instance to this action. When creating the binding, make sure to choose your correct Cloudant instance, credentials, and database for this project. Also make sure to save the sequence after you add this action.
6. Test your work so far by going to the `save-guestbook-entry-sequence` and clicking **Invoke with parameters**. Use the following JSON parameters:

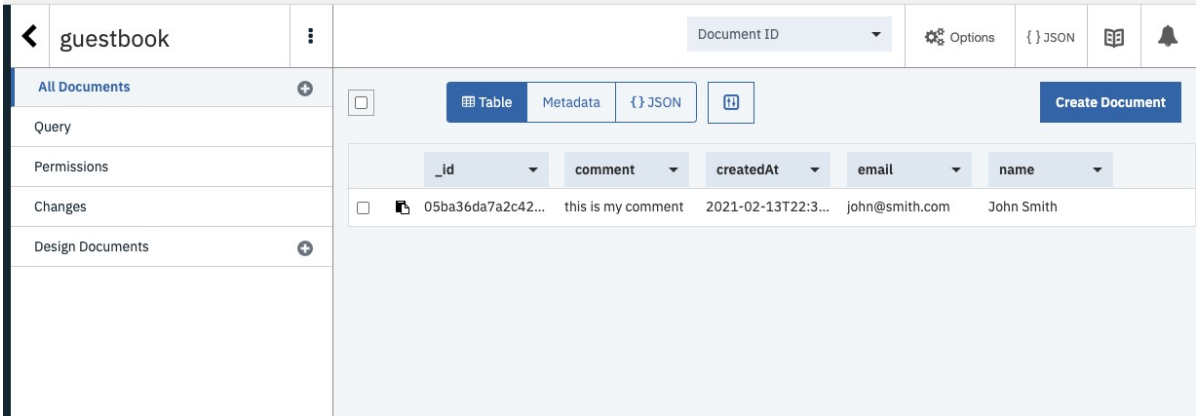
```
1
2
3
4
5
```

```
{
  "name": "John Smith",
  "email": "john@smith.com",
  "comment": "this is my comment"
}
```

Copied!

If this worked, you should not get an error, but you should see the ID and the revision of the document just added.

You should also be able to go to your Cloudant instance dashboard, open the **guestbook** database that you created, and see a newly created document there containing your parameters and the date.



Step 3: Create actions to retrieve guestbook entries

In this step, you need to create a sequence that again consists of two actions, but this sequence will be used to retrieve guestbook entries stored in the Cloudant database. This sequence will list all documents from the database, format the documents, and return them.

1. Create a Node.js action called `set-read-input` in the `guestbook` package.
2. The code for this action should be the following:

```
1
2
3
4
5
6
7
```

```
function main(params) {
  return {
    params: {
      include_docs: true
    }
  };
}
```

Copied!

This code creates the parameters to be passed to the next function. In this case, when we list the documents in the database, we want to include the documents in the result that is returned so that we can view their contents.

3. Add this action to a new sequence called `read-guestbook-entries-sequence`. This sequence should also be in the `guestbook` package.
4. Add a second action to this sequence by clicking the **Add +** button on the sequence page. Use the public `list-documents` action from the Cloudant package. Use your existing package binding since this action will also act upon your same database. If the package binding doesn't show up under **My bindings**, refresh the page. Make sure to save the sequence after you add this action.
5. Create another Node.js action in this sequence. Call this action `format-entries`.
6. The code for this action should be the following:

```
1
2
3
4
5
6
```

```
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
```

```
1. const md5 = require('spark-md5');
2.
3. function main(params) {
4.   return {
5.     entries: params.rows.map((row) => { return {
6.       name: row.doc.name,
7.       email: row.doc.email,
8.       comment: row.doc.comment,
9.       createdAt: row.doc.createdAt,
10.      icon: (row.doc.email ? `https://secure.gravatar.com/avatar/${md5.hash(row.doc.email.trim().toLowerCase())}?s=64` : null)
11.    }})
12.  };
13. }
```

Copied!

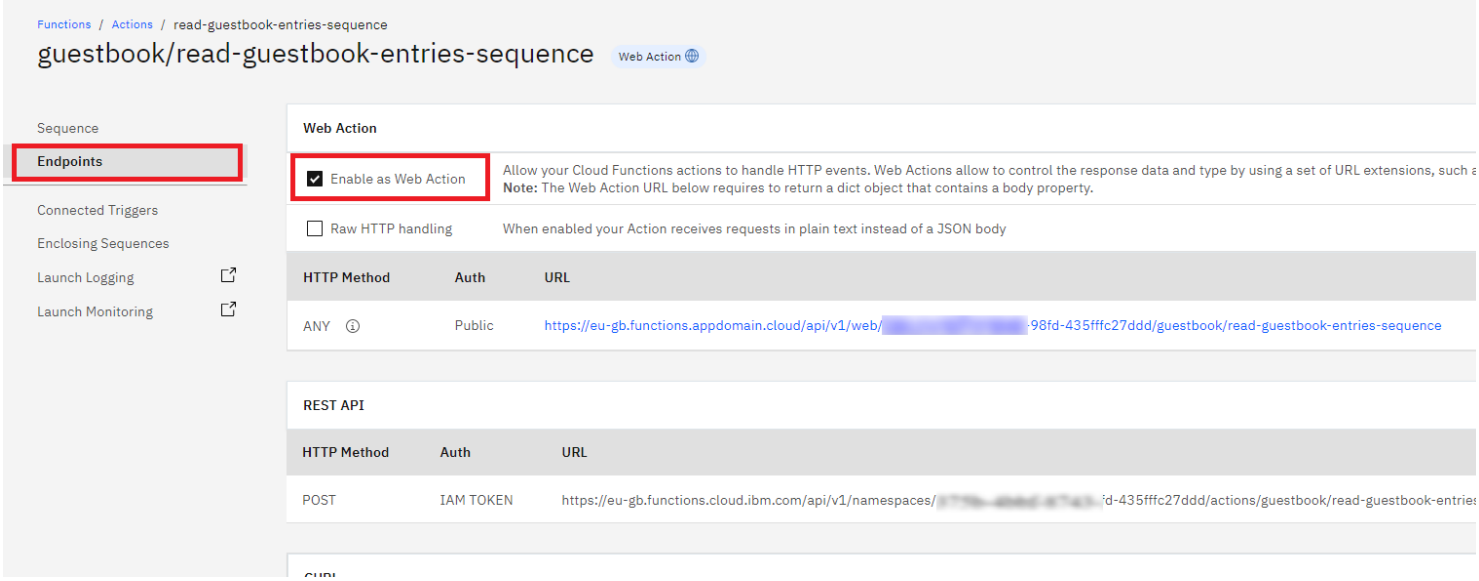
This code formats the data returned by Cloudant so that it can be used by the front end web application for the guestbook UI.

7. Test your work so far by going to the `read-guestbook-entries-sequence` and clicking **Invoke**. You should see this return the formatted comments from your database.

Step 4: Enable the web actions

In order for these functions to be utilized by the guestbook UI, we need to enable the web actions. In this step, you will expose enable your sequences as web actions and use the HTTP Method URL as API that responds to PUT and GET requests in your code.

1. For **each** sequence that you created, enable that sequence as a web action.
2. Make a note of the route URL for Later, as your web application will need it. Refer to the screenshot attached below and copy the URL for read-guestbook-entries-sequence and save-guestbook-entry-sequence.



Note: Please note that using Public URL is not recommended for Production. In real time scenarios please use the URL with the IAM Authentication for security purpose.

Step 5: Deploy the web app using OpenShift

You will deploy the web app in two different ways: as a microservice on OpenShift, and using Object Storage to host the static files. By using OpenShift, you can deploy your web app using as a microservice by using containers. Though this app is currently quite small and will only contain a single microservice, you could easily begin extending this application with additional microservices to enhance its features.

1. Fork this repository which contains code for the front end web application.
2. View the `guestbook.js` file. Notice that the `apiUrl` constant is blank. In the quotes, put the route to your API, which you should have noted in the last step. And change the endpoints too.
3. Insert the URL within the ‘ ‘ on line 4. Remove the name of the sequence from the end.

For example, if the URL is https://eu-gb.functions.appdomain.cloud/api/v1/web/395da288-6950-435fffc27ddd/guestbook/read-guestbook-entries-sequence, then shorten the URL by removing the /read-guestbook-entries-sequence. So now your URL should look something like this:

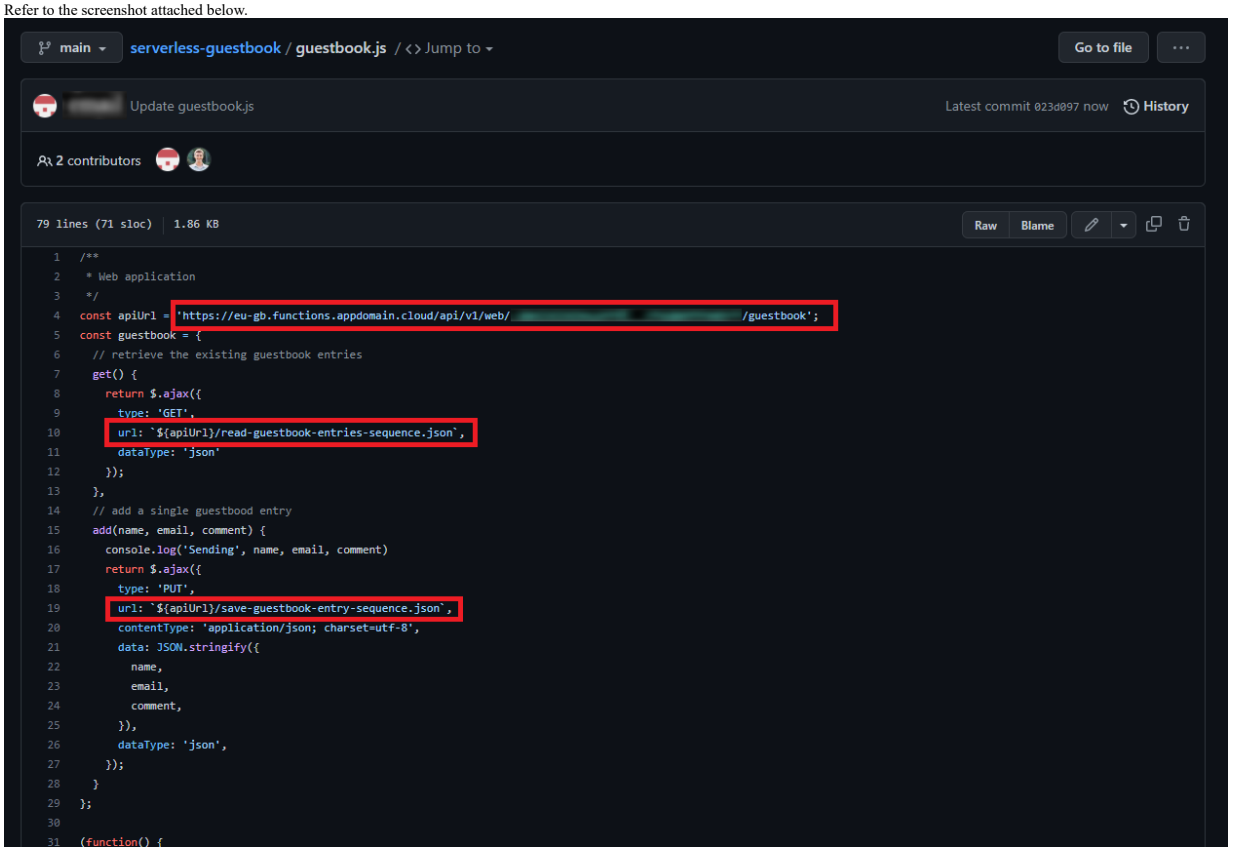
https://eu-gb.functions.appdomain.cloud/api/v1/web/395da288-6950-435fffc27ddd/guestbook

NOTE: The URL mentioned above is a sample URL. Do not use the same in your code.

4. Now replace the get method’s URL endpoint /entries with **/read-guestbook-entries-sequence.json** (line 10) and for post method (line 19) with **/save-guestbook-entry-sequence.json**

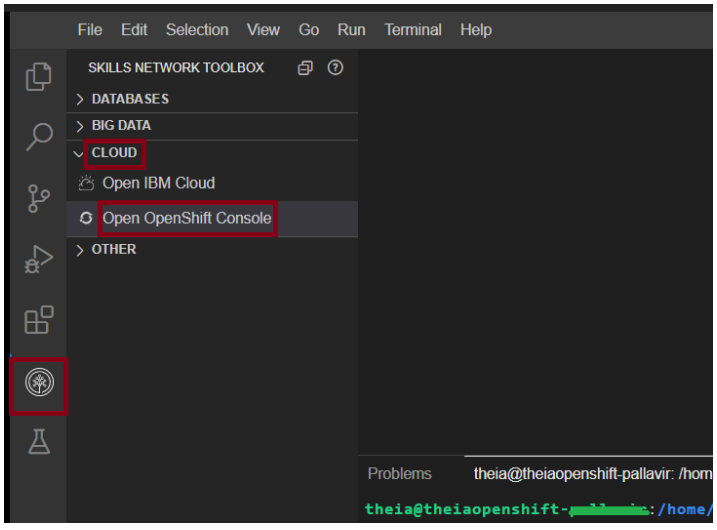
NOTE: Do not forget to add **.json** at the end of the endpoint.

Refer to the screenshot attached below.



Commit this change to your repository. If you're not familiar with GitHub, there is a pencil button the edit the file, and you after making your change you can leave the default commit message and click **Commit changes**.

3. To open openshift web console, click on the Skills Network button on the right, it will open the **"Skills Network Toolbox"**. Then click the **Cloud** then **Open OpenShift console** as shown in the following image.This will launch the OpenShift console in a new tab.



4. Change to the **Developer** perspective.
5. Click the **+Add** button to add a new application.
6. Choose **From Git** so that you can deploy your web app directly from your forked GitHub repository.
7. Paste your GitHub repo URL in the box.
8. Choose the **Httpd** builder image. This will build your web app as an Apache web server that serves static content. This is perfect since the web app consists of two simple files: an HTML file and a JavaScript file.
9. Click **Create**.

OpenShift will now create a build to build your repository into a container image to run on OpenShift.

10. Click on the deployment that was just created. The outer circle is the application, so click the inner circle with the OpenShift logo.
11. You should now see a pod, a build, a route. Once the build completes, the pod should stop crashing and should start running. At that point, click the route to view your web app.

If you've done this correctly and you tested your actions in previous steps, you should see your test comment already present in the guestbook. This is because that comment was persisted to the Cloudant database, and this web app is reading all the comments from that database.

12. Insert a new comment and ensure that it appears in your guestbook.

Step 6: Deploy the web app using Object Storage

You can also host static sites using IBM Cloud Object Storage. By uploading your web app's files to a bucket in Object Storage, you can host a static website. This is particularly useful if you don't feel that you need microservices and your app can run as a simple static site.

This method of deploying the web app will provide us with an entirely serverless solution: a cloud-managed Cloudant database to persist the guestbook entries, IBM Cloud Functions to save and retrieve entries from the database, and a static site hosted in Object Storage. This entire guestbook site will only require you to pay for what you use! (Of course, for this project it won't cost you anything since everything we are using is free!)

1. Create an instance of IBM Cloud Object Storage. Make sure to select the Lite plan.
2. Click **Create a bucket**.
3. Click the arrow to **Customize your bucket**.
4. Enter a bucket name that is unique is across all IBM accounts. Try `<yourinitials>-guestbook`.
5. Select **Regional** resiliency and **Smart Tier** storage class.
6. Scroll down to the Static website hosting and click **Add rule**.
7. Keep the Routing rules (individual) selected and add the Index document `index.html`.
8. Click Public access to **On**.
9. Click **Save**.
10. Scroll to the bottom and click **Create bucket**.

Now you need to add your files to this bucket.

11. From the main page of your repository in GitHub, download a zip file by clicking **Code** then **Download ZIP**.
12. Unzip the file and navigate to the `guestbook.js` and `index.html` files.
13. Open the bucket **Objects** view and drag and drop the `guestbook.js` and `index.html` files to the COS bucket.
14. Navigate to the **Configuration** tab for the bucket and scroll down to the **Static website hosting endpoints** section to copy the Public endpoint into a browser tab.
15. At this link, you should see another incarnation of your guestbook! Test it again to ensure that it works properly. Since the same database backs both the Object Storage and the OpenShift hosted sites, entries submitted in one will be reflected in the other.

Summary

In this final project, you have leveraged the serverless and microservices knowledge that you acquired throughout this course. You interacted with IBM Cloud to provision two types of services: a Cloudant database and Cloud Object Storage. You created actions and sequences to store entries in and retrieve entries from the Cloudant Database. You exposed those sequences using web actions. And finally, you hosted the front end for the guestbook web application as a microservice on OpenShift and as static files on Cloud Object Storage.

Next Steps

Remember to take the appropriate screenshots to submit for this final project!

Changelog

Date	Version	Changed by	Change Description
08.09.2022	1.1	Samaah Sarang	Instructions changed
13.01.2023	1.2	K Sundararajan	Added **Estimated time needed**