Deploying Oracle Digital Assistant Custom Component APIs to Oracle Kubernetes Cluster – Part 2 of 3. – Deploying a sample custom component API to Oracle Kubernetes Cluster on OCI

Abhay Bhavsar, September 2023

Part 1 of this series of 3 articles is a prerequisite for this article. Part 1 describes how to setup an Oracle Kubernetes Engine Cluster (i.e. OKE). Part 2 guides on how to deploy Oracle Digital Assistant custom components to OKE.

Disclaimer

The article will list OCI resources that were automatically configured upon completion of the terraform scripts from Part 1 of this series. Please use the OCI Cost Estimator to perform cost calculation to understand the cost associated with this setup. The author is not responsible for the costs incurred by the setup created.

Cost Estimator - https://www.oracle.com/cloud/costestimator.html

1 Summary and Outline

1.1 Before you begin

The article assumes that the OCI administrator and the developers have a basic understanding of OCI resources like Virtual Cloud Network (VCN), Load Balancers, Oracle Cloud Infrastructure Registry (OCIR), OCI Compute, OCPU and Memory. Knowledge about OCIR Oracle API Gateway Cloud, OCI Vault is not necessary, but it would help. An OCI Administrator access is required to complete the setup. This article creates a new IAM user and adds that user to the Administrator group on OCI.

1.2 Resources

With this article you will find a resources-part2.zip file. Download and unzip the file. Here are the included files and their purpose that are applicable to this article.

- Sample-CC.zip: This is a completed sample custom component API. The articles suggest to copy certain files from this into your custom component API.
- OKE-ARTICLE.postman_collection.json : Postman collection to test the APIs through Oracle API Gateway.
- SampleCCSkill(1.0).zip: This is a sample skill that calls an external custom component on the OKE cluster.
- part2-commands.txt: file containing commands used in this article.

1.3 Access Requirements

This article assumes that the person executing steps in this article is an OCI Administrator. In one of the steps in the subsequent sections of this article an OCI administrator account is created. This account is then used throughout this article and other parts of the series of this articles to run various scripts and OCI commands.

Note: Configuration steps in this article require OCI administrator role access

2 Creating a sample custom component

Create a new custom component by following these steps

2.1 Install NodeJS

Download and Install node by following https://nodejs.org/en/download

Run the following command from your windows laptop/pc/mac to ensure node in installed

```
node –version
```

This should show the version of the node installed on your machine.

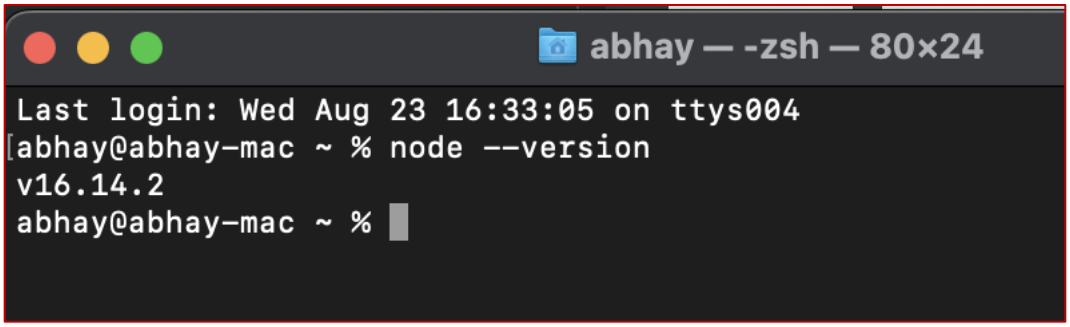

Figure 1: npm installed

2.2 Install bots-node-sdk

Install the bots-node-sdk using the following command

```
npm install -g @oracle/bots-node-sdk
```

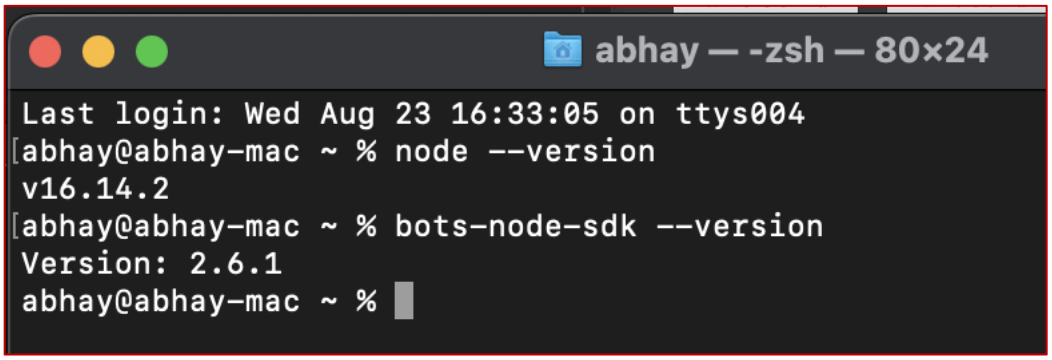

Figure 2: Oracle Bots Node SDK installed

More information about the Oracle Bots Node SDK is available at https://github.com/oracle/bots-node-sdk

2.3 Create new custom component

Create a new custom component in a new directory e.g. NEW-CC

```
bots-node-sdk init Sample-CC --name sample-cc
```

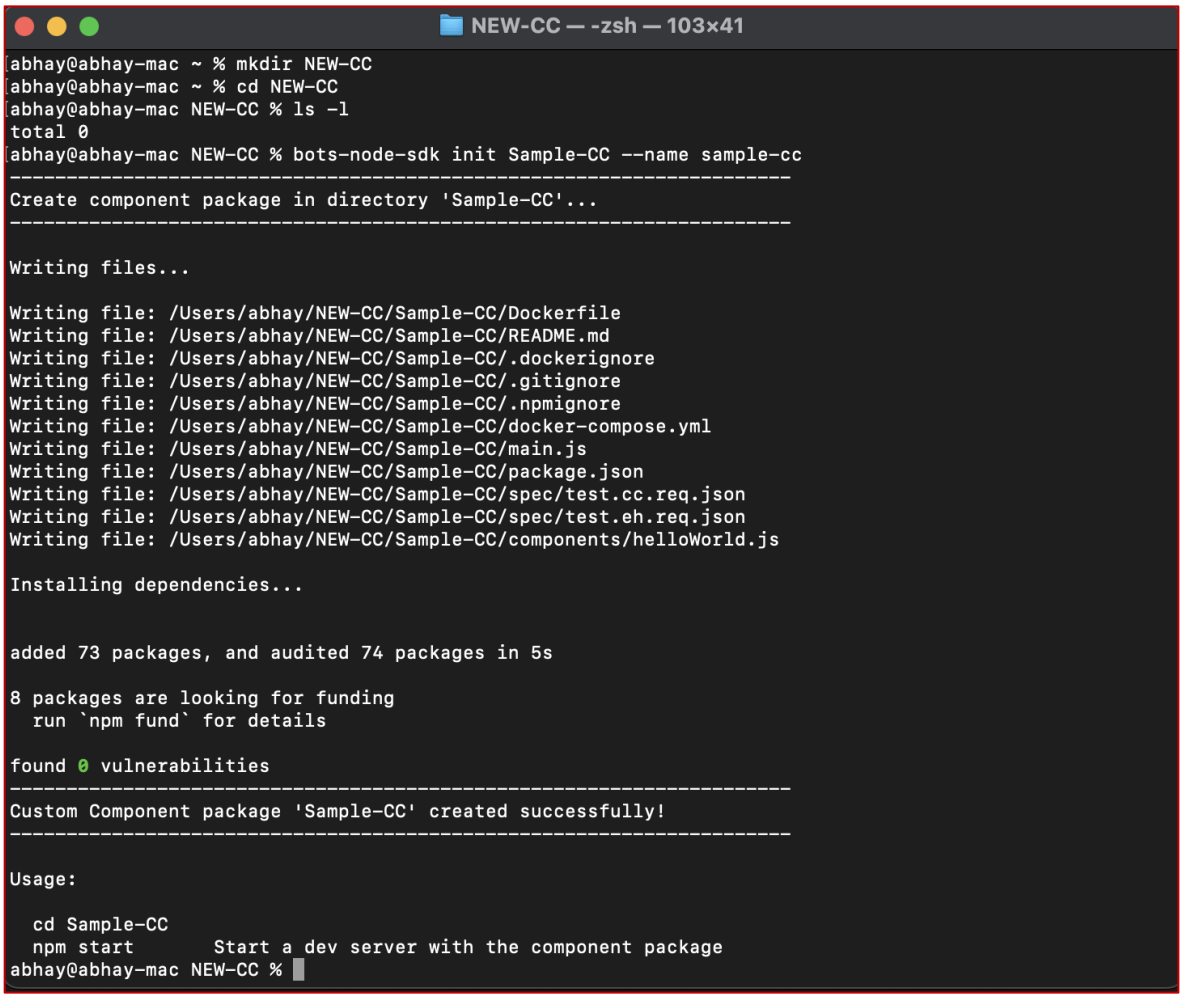

Figure 3: Sample Custom Component API created

3 Configure Custom Component API for OKE

From the supplied `resources-part2.zip` extract the `Sample-CC.zip` into a (new) separate directory. Copy the following files from this directory to the extracted Sample-CC directory. This will override the existing files.

1. `package.json`
2. `docker-compose.yml`
3. `deploy` folder including all files and subdirectories

Open the newly created sample custom component API (Sample-CC) in your preferred code editor. This article uses Visual Code as the default code editor.

Review the content of the files within the Sample-CC directory

1. `package.json`

   Note the following

   - Line number 2, name of the custom component api: `oda-sample-cc`
   - Line number 12, docker build command
   - Line number 13: undeploy – command to undeploy the custom component API from OKE
   - Line number 14: deploy – command to deploy the custom component API to OKE

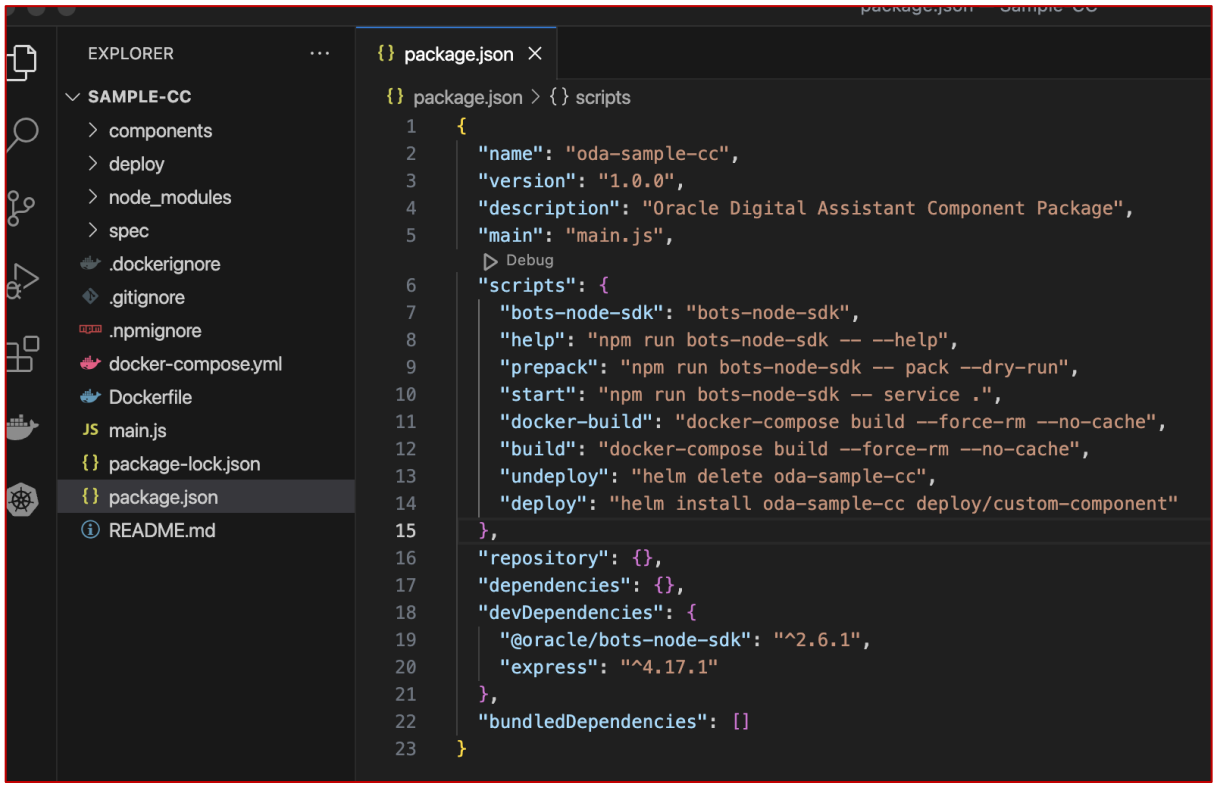

Figure 4: Review package.json

2. `docker-compose.yml`

The following screenshot shows more information about the various parts of the image repository.

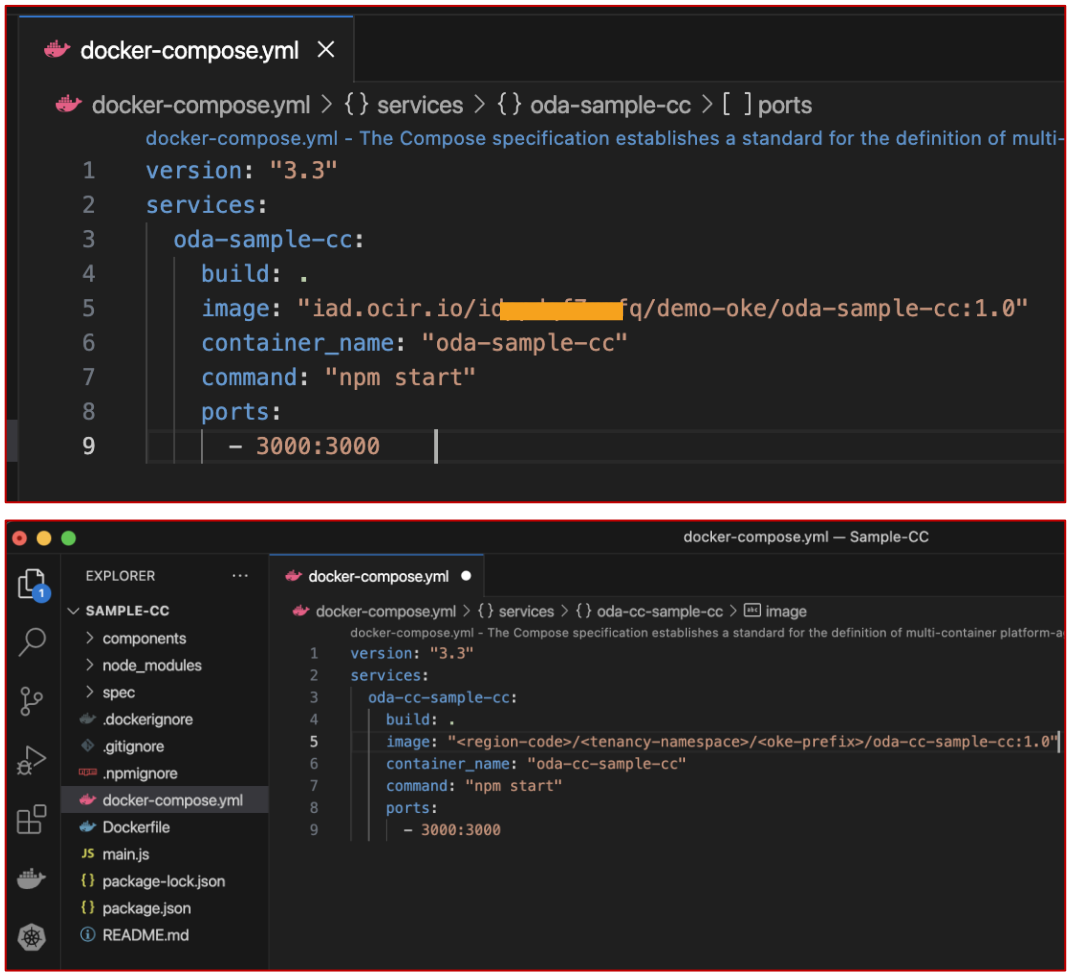

Figure 5: Review `docker-compose.yml`

The format of the image property is very important. Find the region code from the following documentation https://docs.oracle.com/en-us/iaas/Content/General/Concepts/regions.htm. The screenshot example shows: iad.ocir.io where iad is the region code for us-ashburn-1.

The tenancy-namespace can be found by referring to Part 1 (Step 5) in the Deploying Oracle Digital Assistant Custom Component APIs to Oracle Kubernetes Cluster article series, where the steps explain how to find the tenancy namespace from the Object Storage service in the OCI console. The <oke-prefix> is the same prefix name that was configured in the `terraform.tfvars` file.

3. In the Sample-CC | deploy/custom-component | templates directory edit the values in the `values.yaml` file.

   Ensure that the region code, object storage namespace and that the repository name is same as the prefixName.

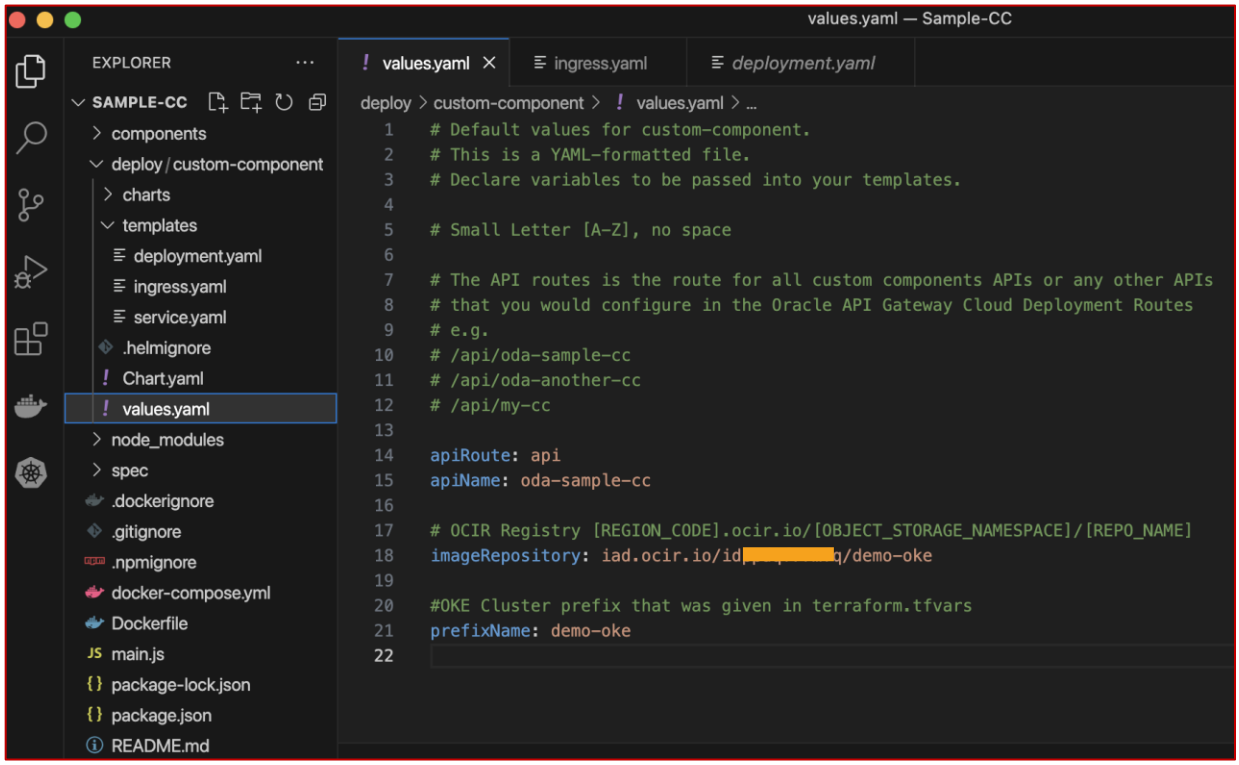

Figure 6: Review `values.yaml` and update it.

The apiRoute is the prefix used in the Oracle API Gateway cloud. A single route in the API gateway cloud would be needed for all the subsequent APIs. In the subsequent section of this article, there will be instructions on how to configure new routes for the API gateway cloud deployment.

4 Upload the custom component API to cloud shell

After reviewing and verifying the various configuration files, create an archive of the Sample-CC directory using zip command or an appropriate archive tool, giving the zip file a name like `Sample-CC.zip`.

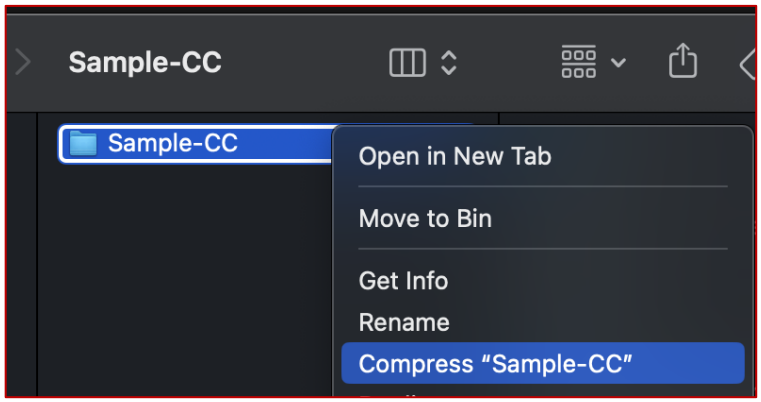

Figure 6: Create zip file e.g, `Sample-CC.zip`

Log into the Cloud Shell using the OKEAdmin account. Select the Cloud Shell's upload menu option. From the File Upload dialog, drag and drop the Sample-CC.zip file into the drop section.

https://cloud.oracle.com/?region=us-ashburn-1

Check your region and change the URL accordingly.

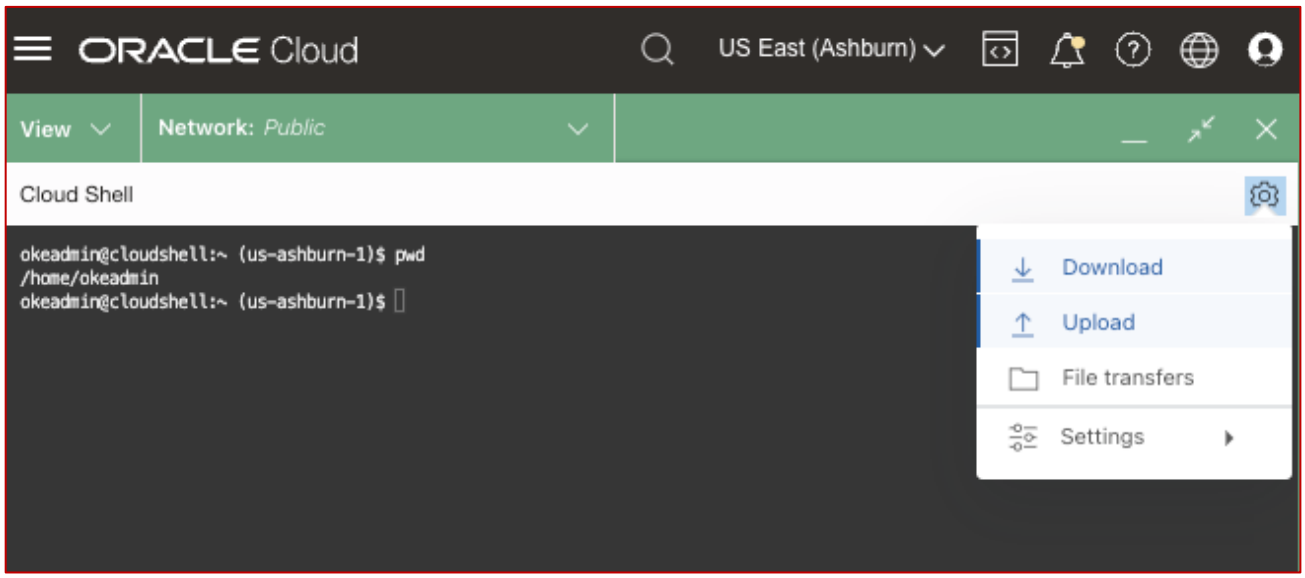

Figure 7: Upload to the cloud shell

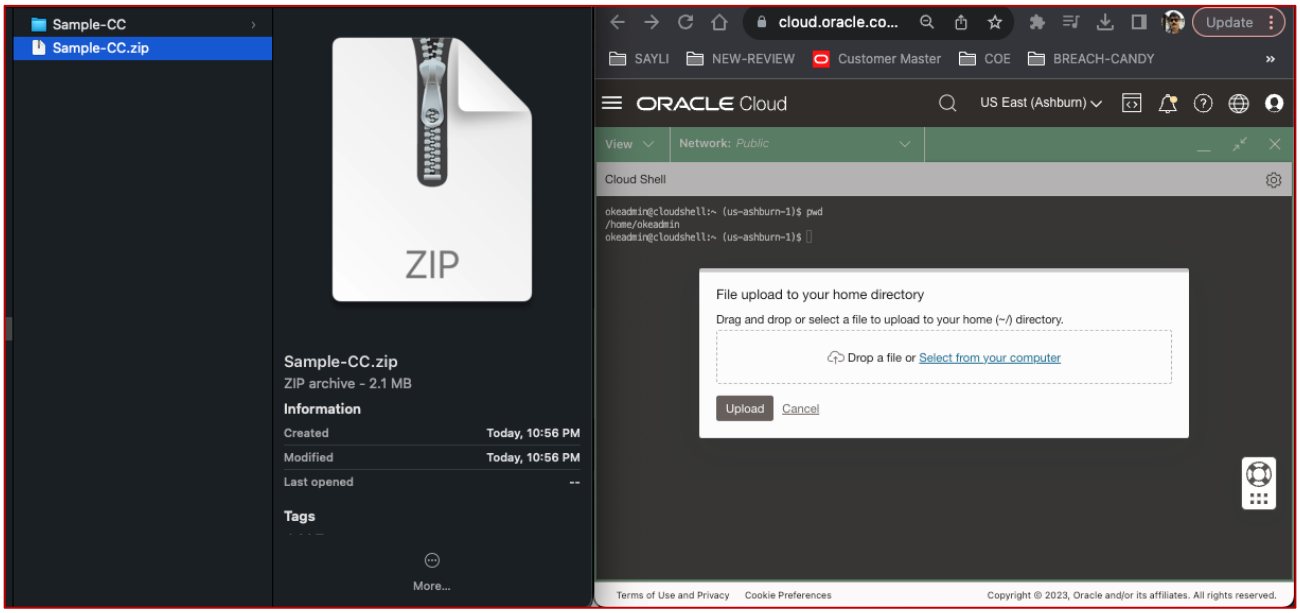

Figure 8: `Sample-CC.zip` uploaded to cloud shell

Once the file has been uploaded successfully, locate the file in the home directory. Unzip this file using the following command:

```
unzip Sample-CC.zip
```

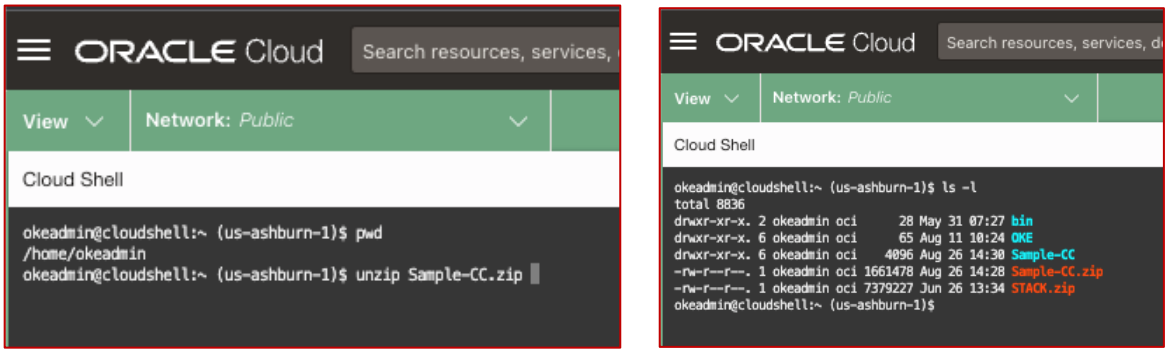

Figure 9: Locate and unzip the Sample-CC.zip

5 Deploy the custom component to OKE cluster

This section provide step by step directions to deploy the sample custom component API to the OKE cluster. The steps involves

- Creating a docker image using the docker compose command
- Push the image to the OCI registry
- Use the kubectl utility to deploy this image to the OKE cluster.

Although the OCI Code Editor has an in-built code editor, this article does use Visual Code as the editor of choice in some examples

5.1 Verify configuration files

In the OCI Code Editor, locate the Sample-CC in the directory as shown in the screenshot below. Review all of the files `package.json`, `docker-compose.yml` and `deploy | custom-component |values.yaml` that were edited in your local environment.

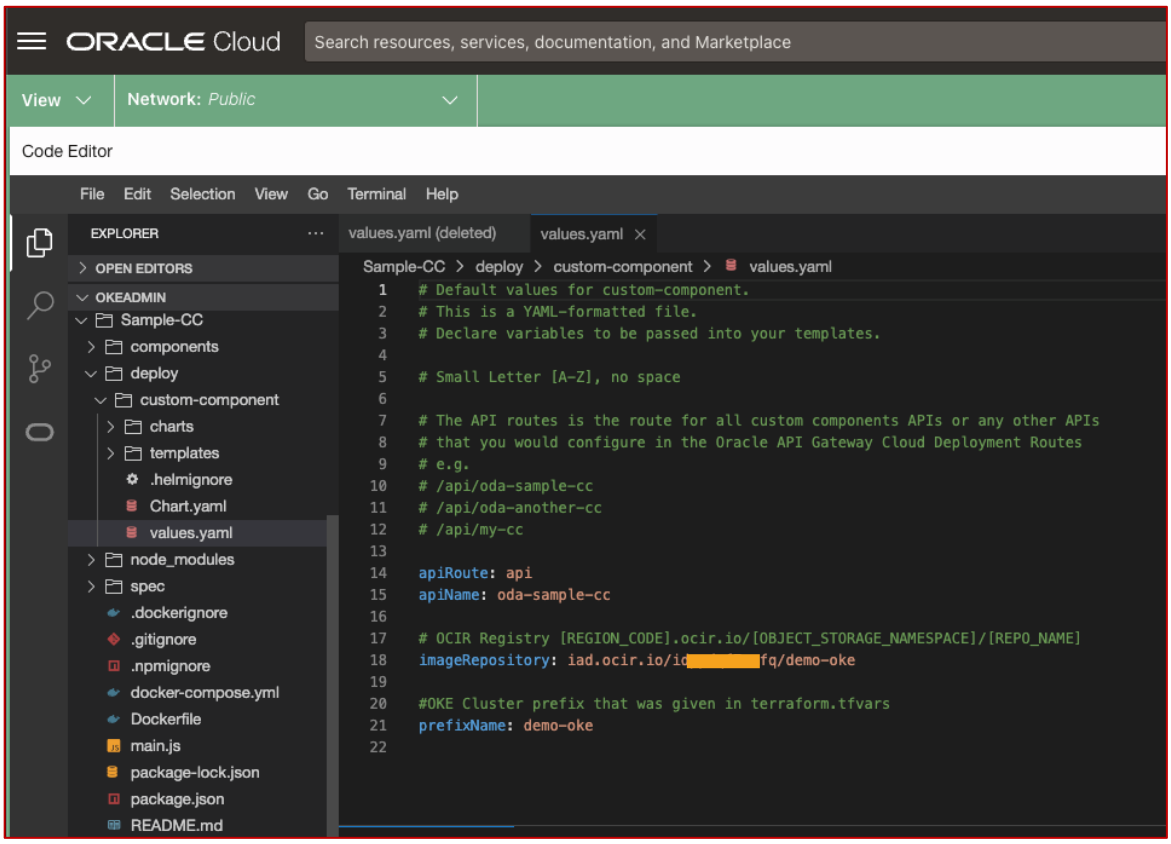

Figure 10: Sample-CC in code editor, showing `values.yaml`

5.2 NPM Install

Refresh the NPM modules. Open the Cloud Shell. Run the following command.

```
cd ~/Sample-CC
```

```
npm install
```

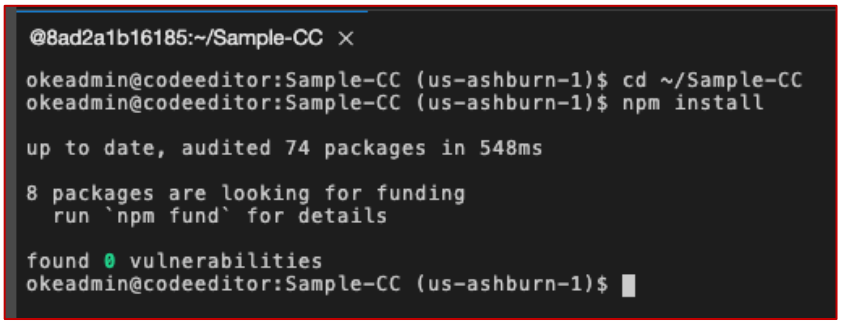

Figure 11: `npm install`

5.3 Build using docker-compose

Execute the package.json's **build** command from Cloud Shell. Ensure that the current directory is Sample-CC.

```
npm run build
```

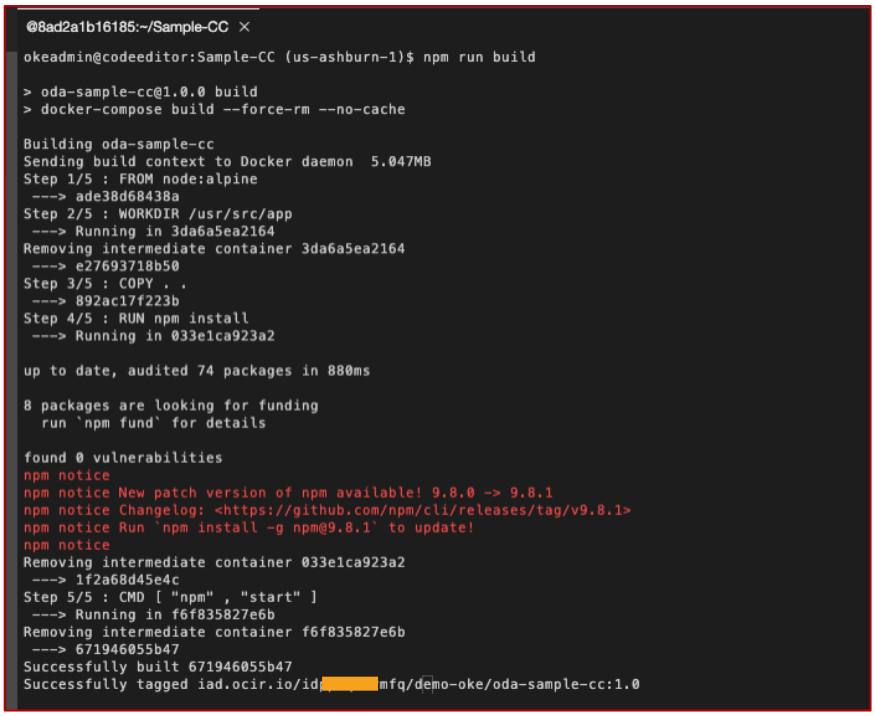

Figure 12: `npm run build`

This command basically calls the `docker-compose` to build the image.

For the docker-compose to run you should have the docker-compose be installed an available in the path of the OCI account OKEAdmin. In Part 1 of the Deploying Oracle Digital Assistant Custom Component APIs to Oracle Kubernetes Cluster series, section 2.5.2, there is information about how to configure this.

After this execution of the command, the last line of the output shows that the image was tagged:

Successfully tagged iad.ocir.io/idpXXXXmfq/demo-oke/oda-sample-cc:1.0
Copy this tag and save it. The tag will be used in the next step, which is to push the image to OCI registry.

5.4 Create a new OCI repository

The next step is to create a public image repository in the OCI registry. Access the container registry using the following URL (check your region and change accordingly).

https://cloud.oracle.com/compute/registry/containers?region=us-ashburn-1

Select the compartment that was created by the terraform scripts in Part 1 of this series (e.g. `OKE-DEMO`).

Select the **Create repository** link, choose **Public** Access and provide the repository name using this pattern: `<prefixName>/oda-sample-cc`

Select **Create** to complete.

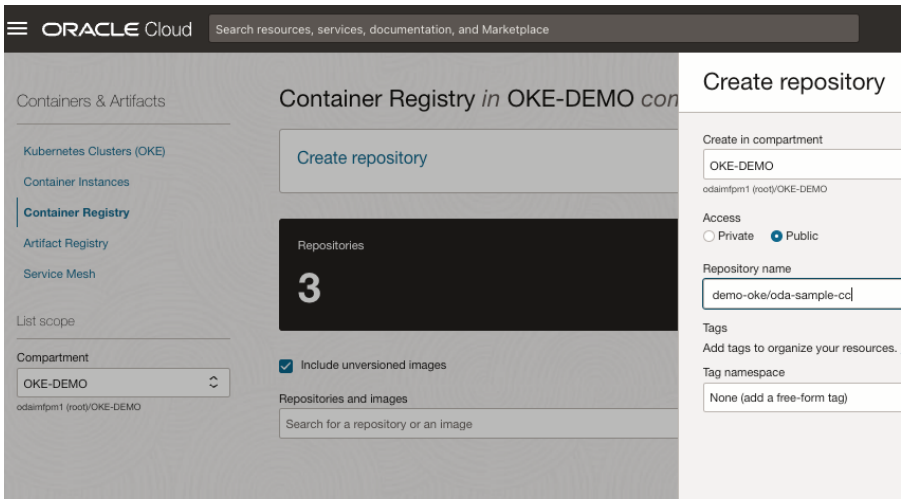

Figure 13: Create new repository in OCI registry

Notice that there are 3 existing repositories that can be seen under “Repositories and Images”. Follow the same format. The OKE cluster prefix name e.g. demo-oke in this case is used as the prefix for the repository.

From Part 1 of this series, the terraform script created 3 repositories that can also be viewed under Repositories and images drop down. The repository name basically is following this same format. If the repository is private the subsequent docker push command would fail.

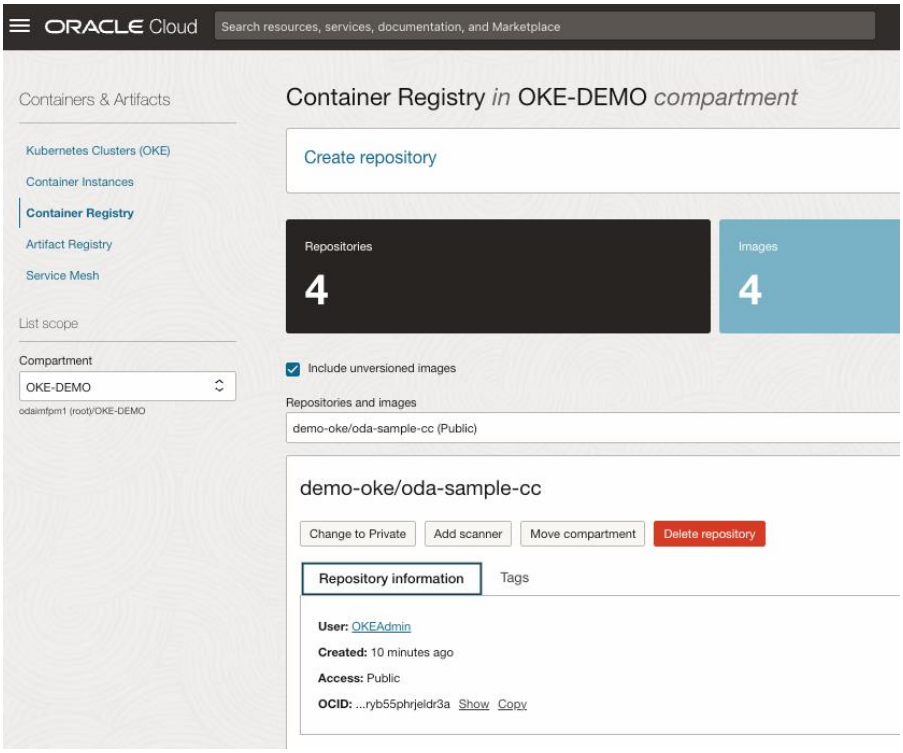

Figure 14: Create a new repository for your new custom component image.

5.5 Push docker image to OCI repository

Next step is to push the image to the repository created in the OCI registry. Run the following command, which uses the saved tag from the npm build command. Update the command to reflect the correct object storage namespace (e.g. idpXXXXmfq) and OKE cluster prefix name (e.g. iad).

```
docker push iad.ocir.io/idpXXXXmfq/demo-oke/oda-sample-cc:1.0
```

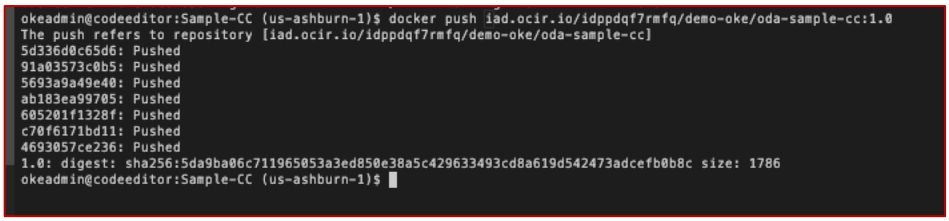

Figure 15: `docker push`

The complete tag maps to the repository created. Ensure all values are correct: region code (**`iad.`**`ocir.io`), object storage namespace (`idpXXXXmfq`), OKE cluster prefix (**`demo-oke`**) and the sample custom component name (`oda-sample-cc`) and version (`1.0`).

Confirm that the image is deployed to OCI registry by refreshing the browser page showing the container registry.

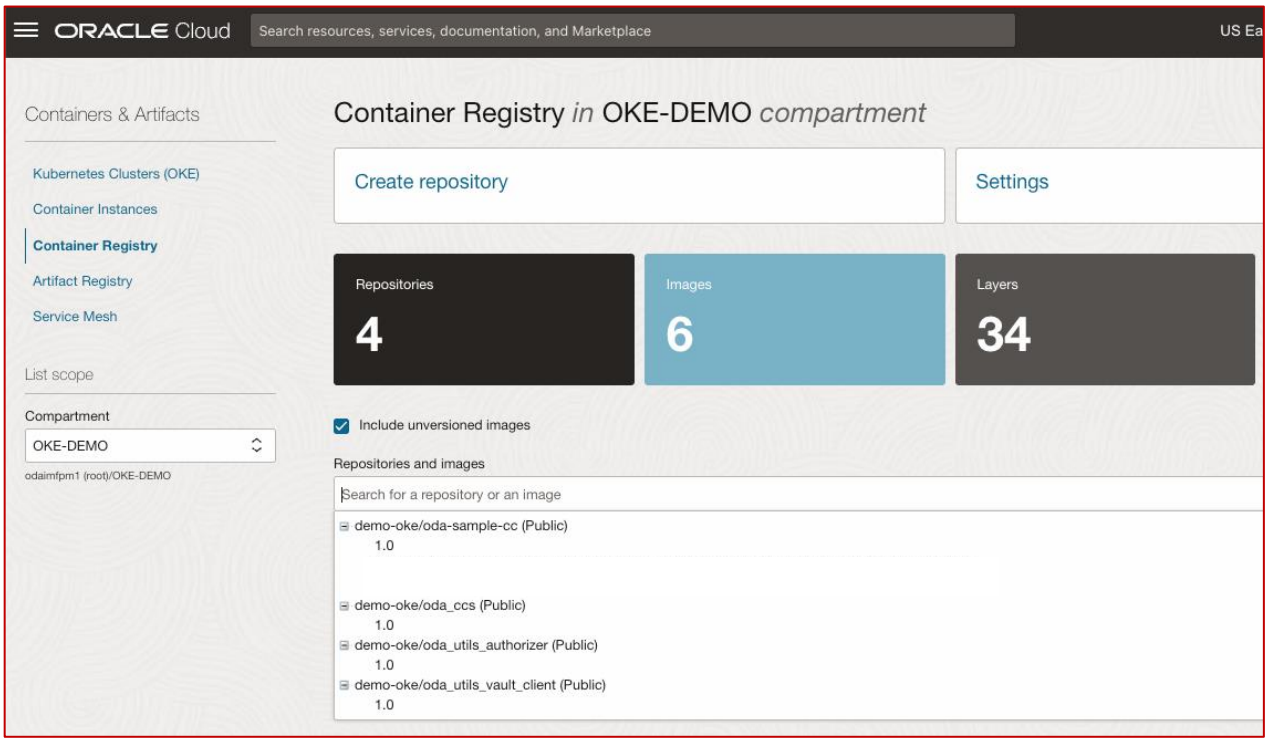

Figure 16: oda-sample-cc pushed to OCI registry

5.6 Deploy the image to OKE

Next step is deploy this image to the OKE cluster. As part of the deployment a pod gets created and gets exposed to the OKE node compute and then gets configured to be accessed through the load balancer. In the subsequent steps the API Gateway cloud can be configured to make the API available to public by wiring it up with the load balancer.

In the Cloud Shell run the following command.

```
npm run deploy
```

```
okeadmin@codeeditor:Sample-CC (us-ashburn-1)$ npm run deploy

> oda-sample-cc@1.0.0 deploy
> helm install oda-sample-cc deploy/custom-component

WARNING: Kubernetes configuration file is group-readable. This is insecure. Location: /home/okeadmin/.kube/config
WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /home/okeadmin/.kube/config
NAME: oda-sample-cc
LAST DEPLOYED: Sat Aug 26 15:00:56 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
okeadmin@codeeditor:Sample-CC (us-ashburn-1)$
```

Figure 17: Deploy to OKE using `npm run deploy`

5.7 Verify the pod

Run the following command to check the status of the pod. The pod will have a prefix `oda-sample-cc`.

```
kubectl get pods
```

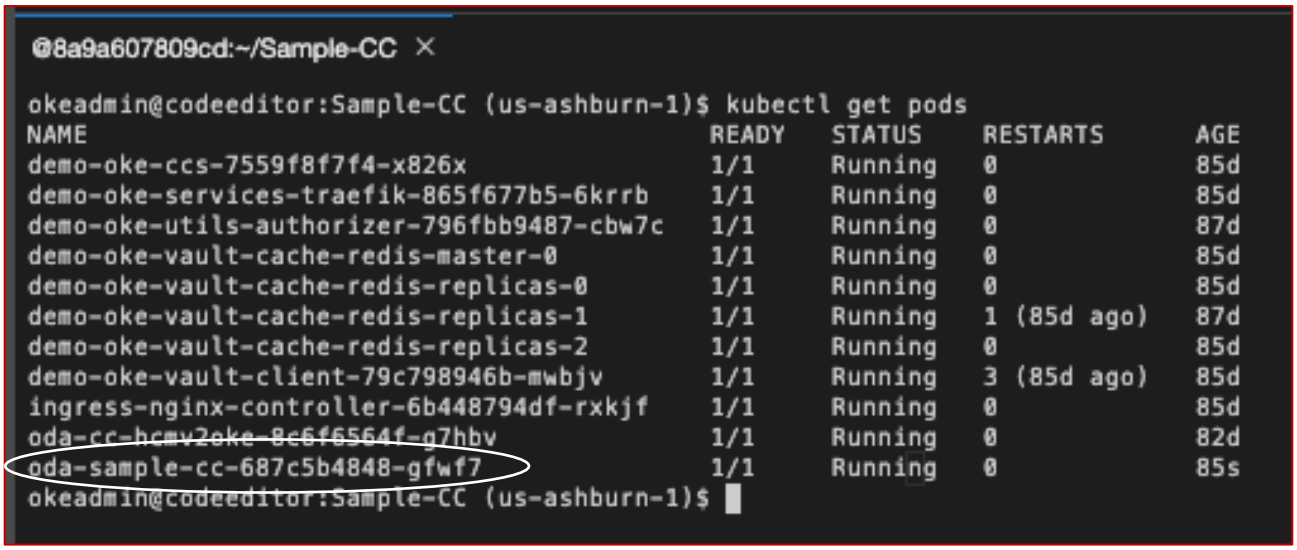

Figure 18: Locate the pod

The pod should be in a Running status.

In case the pod is not in a running status check the following

- The OCI Repository created exists and is public
- Check the complete tag and the corresponding OCI repository created.
- Confirm that the image has been pushed to OCI registry using the `docker push command`
- From the Cloud Shell run the following command to check if the custom component has no errors. See the following screenshot and compare.

  ```
  npm start
  ```

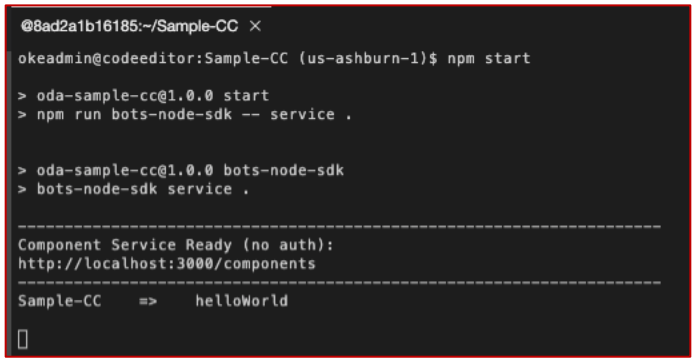

Figure 19: Validate that the custom component has no errors

To check the log of the pod running the custom component run the following commands

```
kubectl get pods
```

Locate the pod corresponding to the `oda-sample-cc` custom component and copy that in your clipboard. This would help to check the log using the following command.

```
kubectl logs -f oda-sample-cc-<random-number>
```

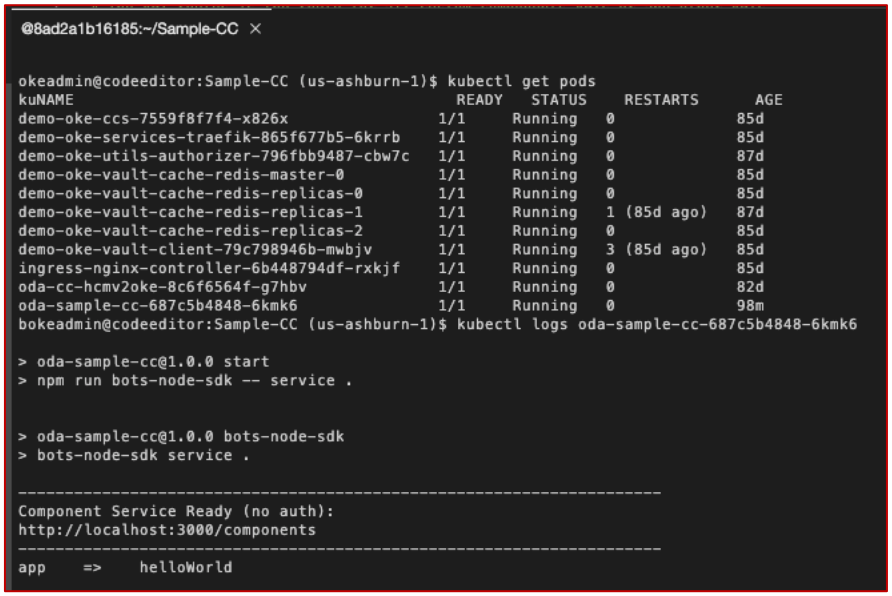

Figure 20: custom component running inside the OKE pod.

5.8 Re-deploy new changes

There will be times most likely that changes will need to be done on an already deployed custom component. Since there are 2 possible ways that new code changes can be introduced to the custom component, the section will cover both.

- Option 1: For small and/or urgent changes, developers can use the OCI Cloud Editor to modify the appropriate files
- Option 2: Developers would make the code changes locally, repackage (e.g. zip) and then proceed to upload the modified zip via the Cloud Shell

The process of creating a zip file and how to upload the zip has been explained in section 4 of this article.

Whichever process is chosen to update the custom component, there is only a single path to ensure that the changes are deployed correctly. The following will cover these steps.

All commands used in these steps are executed from the root of the custom component directory. Using the sample custom component used in this article, the root directory would be Sample-CC

1. Perform undeploy
    a. Once the updated zip has been uploaded via the Cloud Shell or the updates where completed within the (OCI) Code Editor, the first step is to **undeploy** the existing deployment. This is important, since if this step is skipped, the deployment would fail.

    b. Run the following command from the root of the custom component directory.

    ```
    npm run undeploy
    ```

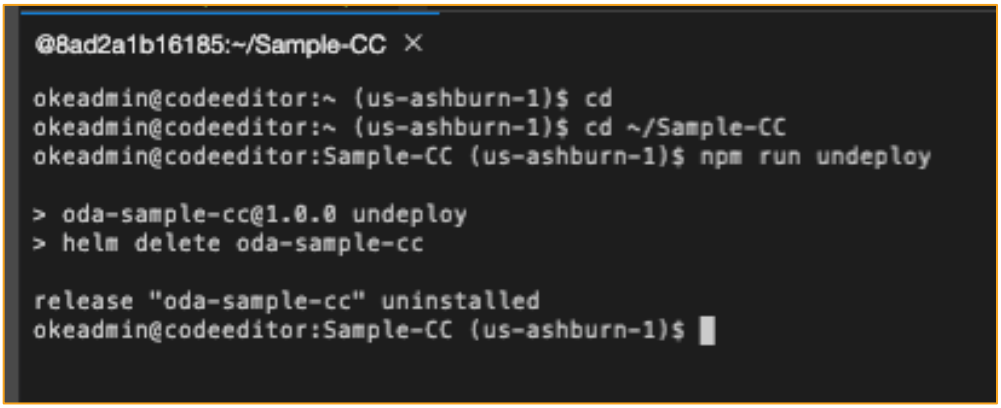

Figure 21: npm run undeploy

2. Perform the Build
   Once the pod and the necessary services have been undeployed, the next step is to build the new image. Run the following command. This step is explained in section 5.3 of this article.

3. Perform the Push
   As before, the build will produce a tag. Note the tag and then push the (new) image to the existing repository using the following command. This step explained in section 5.5 of this article.

   `docker push` [tag]

Note that the repository need not be deleted and recreated. Every new push created necessary layers of changes in the existing OCI repository e.g. demo-oke/oda-sample-cc

4. Perform the Deploy
   The last step is to deploy the new image. This will create a new pod and other necessary services. Run the following command. This has been explained in section 5.6 of this article.

6 Configure the Oracle API Gateway Cloud

In order to consume the custom component API externally from postman or from an Oracle Digital Assistant Skill, an Oracle API Gateway Cloud will need to be configured.

6.1 Login to Oracle API Gateway

Access the Oracle API Gateway Cloud by using the word **gateway** in the OCI Console's search field (located on top by the Orcle Cloud logo) or by using the following URL. (change your region if necessary.

https://cloud.oracle.com/api-gateway/gateways?region=us-ashburn-1

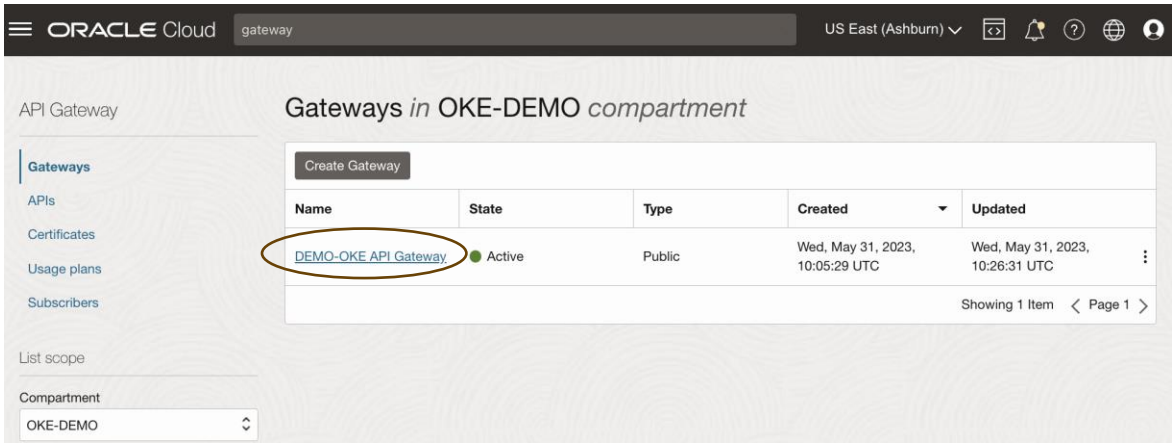

Figure 21: API Gateway

Select the gateway, which was created in Part 1 of this series, (e.g. DEMO-OKE API Gateway). Within the proceeding UI, select **Deployments**. Again, from the previous Part 1 deployment, there will be an existing deployment as shown below. In addition, ensure you are in the right compartment.

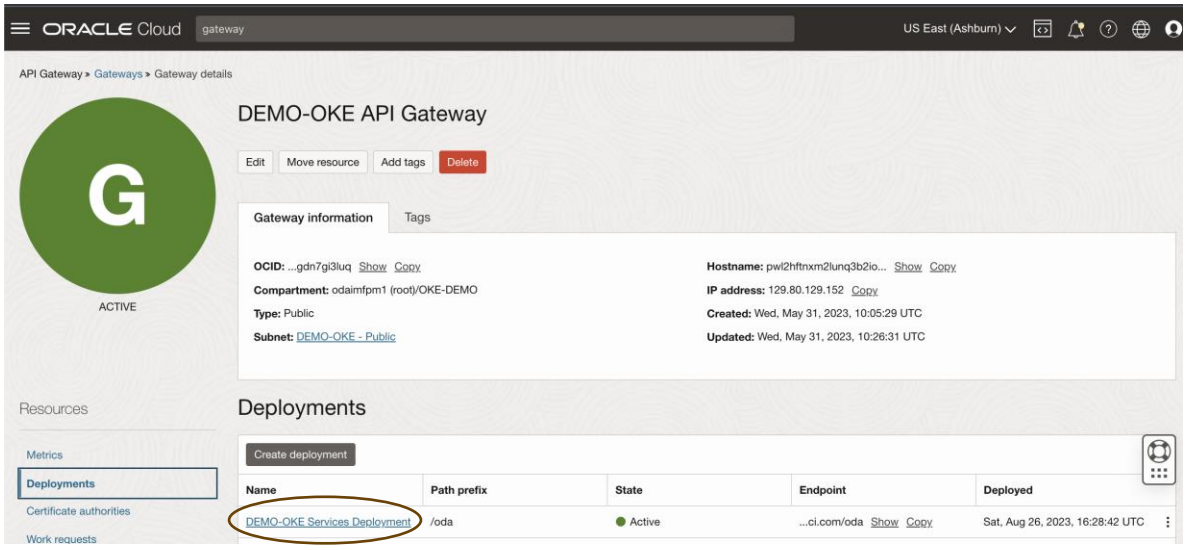

Figure 22: Deployment

6.2 Edit deployment and configure new routes

This section provides the instructions to add new routes to the API Gateway Cloud deployment. This step will make the new custom component API publicly available using the Oracle API Gateway Cloud. The route just needs to be created once, so that any subsequent deployments, including any new custom components, can leverage this same configuration.

Select the Deployment (e.g. **DEMO-OKE Services Deployment**)

Select the **Edit** button to edit deployments

Figure 23: Edit Deployment

In the Edit Deployment screen, select (3) **Routes**. In Part 1 of this series, there are existing routes (e.g. Route1, 2..), which were created by the terraform script. Collapse these existing routes or scroll down to the bottom of the page to locate the **+ Another Route** button.

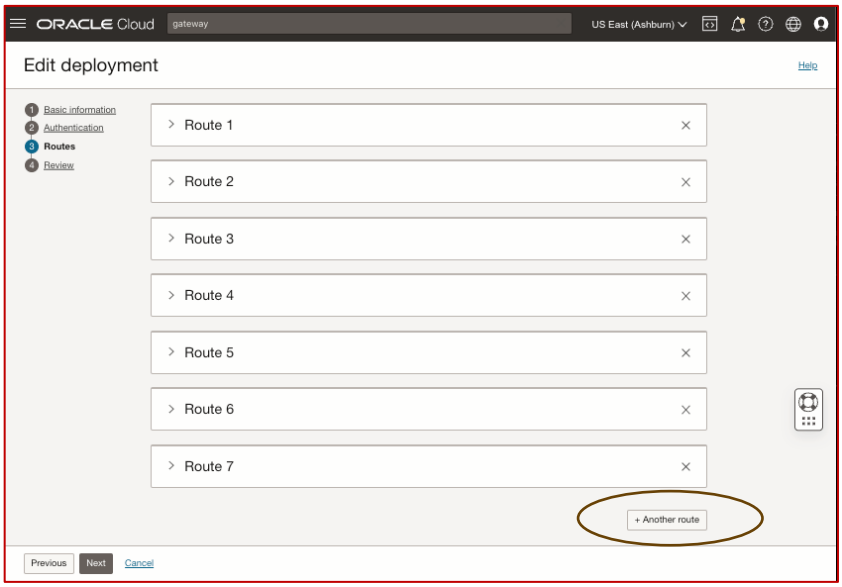

Figure 24: See existing Routes, add new routes

The following table lists the values that are necessary to complete the route for the new custom component. The input fields for URL and others will be available once the Backend Type of HTTP is selected.

Property	Value	Description
Path	/api/{cc}/components	A parameterized route to support custom components API
Methods	GET	
Backend Type	HTTP	
URL	http://x.x.x.x/api/${request.path[cc]}/components	Maps the incoming request to the load balancer.
Connection established timeout	60	
Request transit timeout	10	
Reading response timeout	10	
Disable SSL Verification	Checked	

Replace the IP address in the URL with the IP address of the current load balancer. For example, copy this value from one of the existing routes (e.g. Route 1).

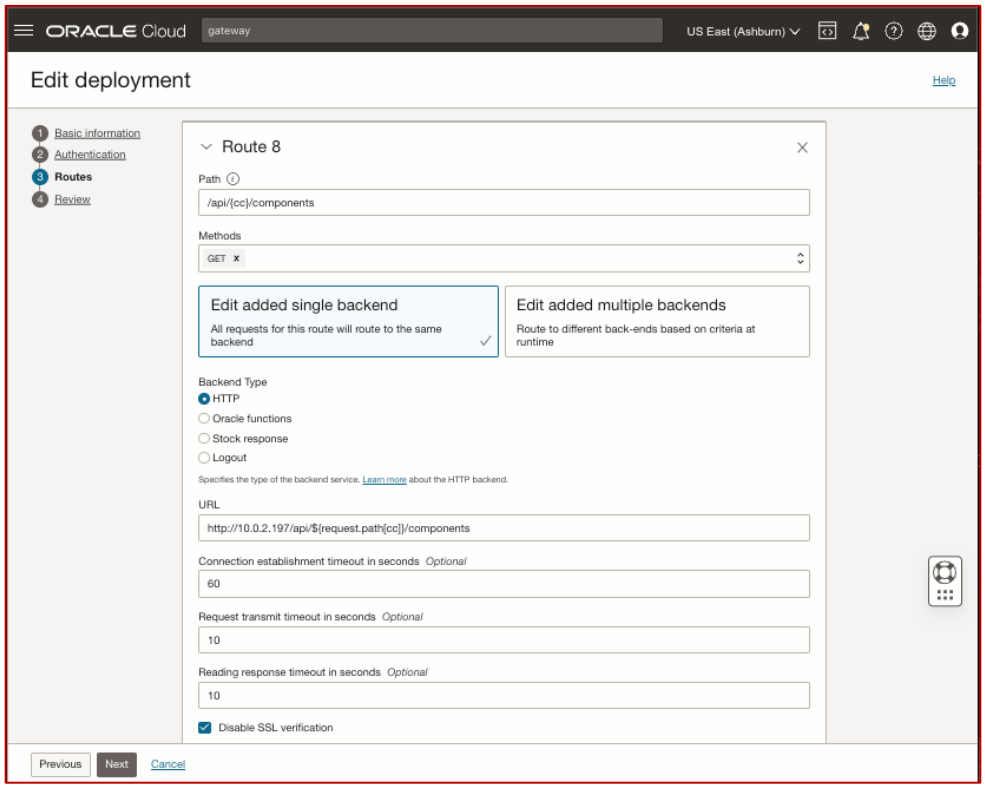

Figure 25: Add new route for the custom component APIs

Create another route for the POST method.

Here is the table that provides values to be entered for this particular route.

Property	Value	Description
Path	/api/{cc}/components/{componentName}	A parameterized route to the custom components API
Methods	POST	
Backend Type	HTTP	
URL	http://x.x.x.x/api/${request.path[cc]}/components/${request.path[componentName]}	Maps to load balancer resource.
Connection established timeout	60	
Request transit timeout	10	
Reading response timeout	10	
Disable SSL Verification	Checked	

Replace the IP address in the URL with the IP address of the current load balancer. For example, copy this value from one of the existing routes (e.g. Route 1).

In the URL, note that are 2 variables `cc` and `componentName` that are dynamically added from the incoming request.

For example, the resulting URL would look something like shown below.

https://pwl2hftnxXXXXXXo773gsi.apigateway.us-ashburn-1.oci.customer-oci.com/oda/api/oda-sample-cc/components

where the following values would be added in the URL

`/oda` – This is configured as the base route path for the current deployement

`/oda/api` – This is the route path to support multiple custom components

`/oda/api/oda-sample-cc` – Specific Custom component API path

`/oda/api/oda-sample-cc/components` – This complete the custom component URL.

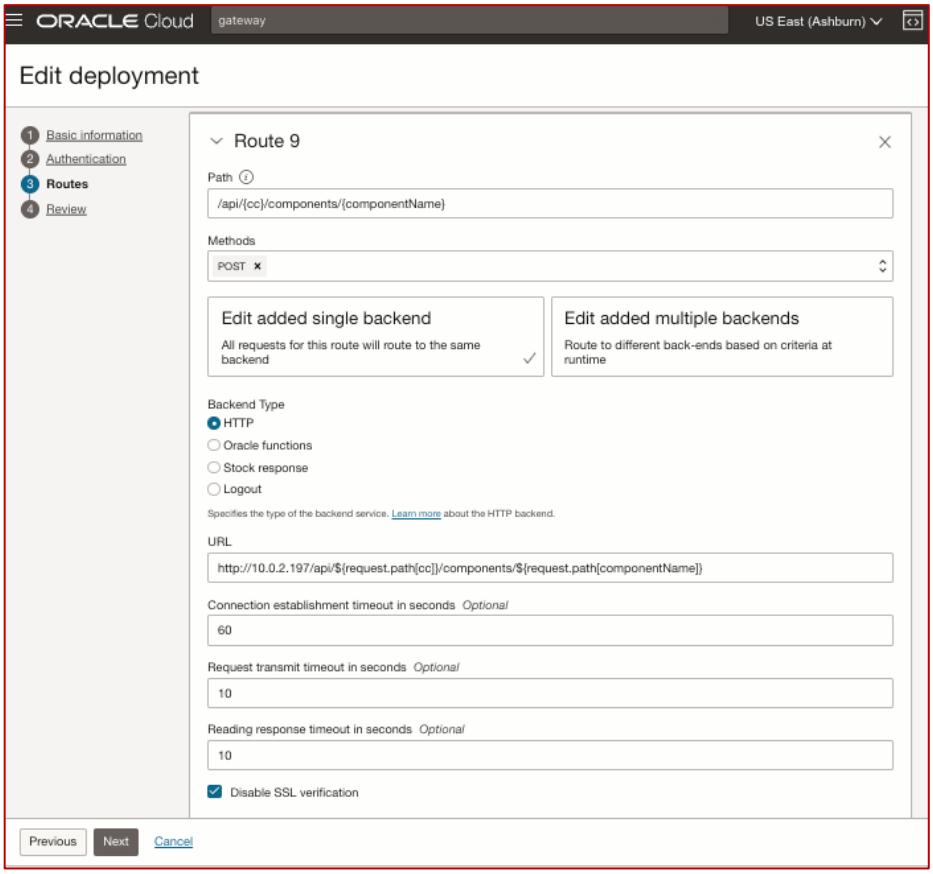

Figure 26: Add Route 9 for POST

Select the **Next** button to complete the configuration of the 2 new routes and then select **Save changes**. Once the new routes changes have been activated, the screen switches back to the Geteway Deployment details page.

6.3 Test the API from postman

As before in Part 1 or this series, Postman will be used to test the deployed custom component.

The (downloaded) `resources-part2.zip` contains a file: `OKE-ARTICLE.postman_collection.json` that can be imported into Postman.

Locate the **OKE Demo CC** in the OKE-ARTICLE collection. Select the View More Actions (…) and from the context menu, select **Duplicate**.

The collection in the resources-part2.zip is similar to the collection that was included in resources-part1.zip. Moreover, in Part 1 of this series, the URL for the verification step, was configured with the Oracle API Gateway value. This same Gate value can be used in the (part2) URL. Otherwise, the API Gateway URL value can be manually modified.

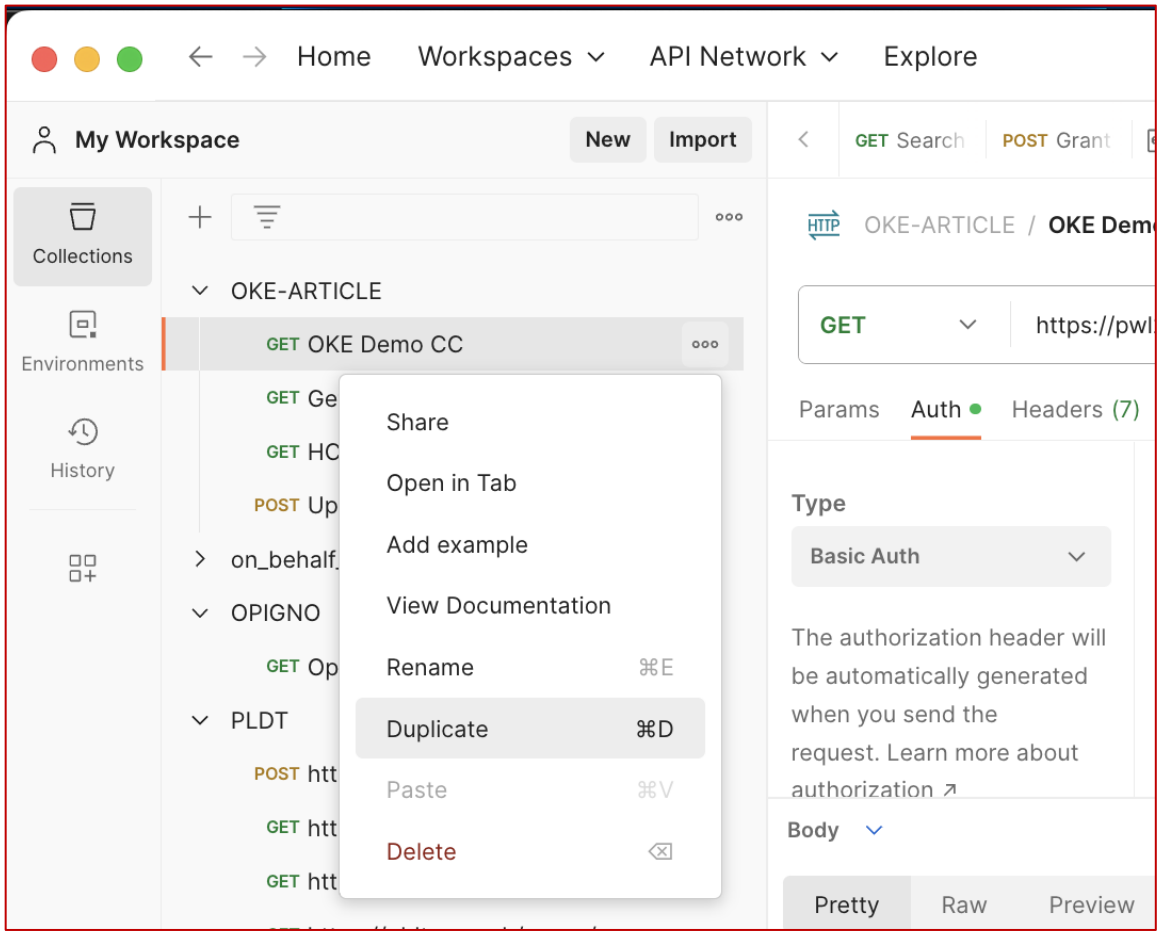

Figure 27: Locate the OKE Demo CC and duplicate it

The Postman URL will look similar to this:

https://pwl2hftnxm2XXXXXXgsi.apigateway.us-ashburn-1.oci.customer-oci.com/oda/api/oda-sample-cc/components

Next, configure Basic Authorization with Username as **oda** and Password as **oda**

Select the **Send** button. Verify that there is a HTTP 200 response with the response Body displaying the custom components meta-data.

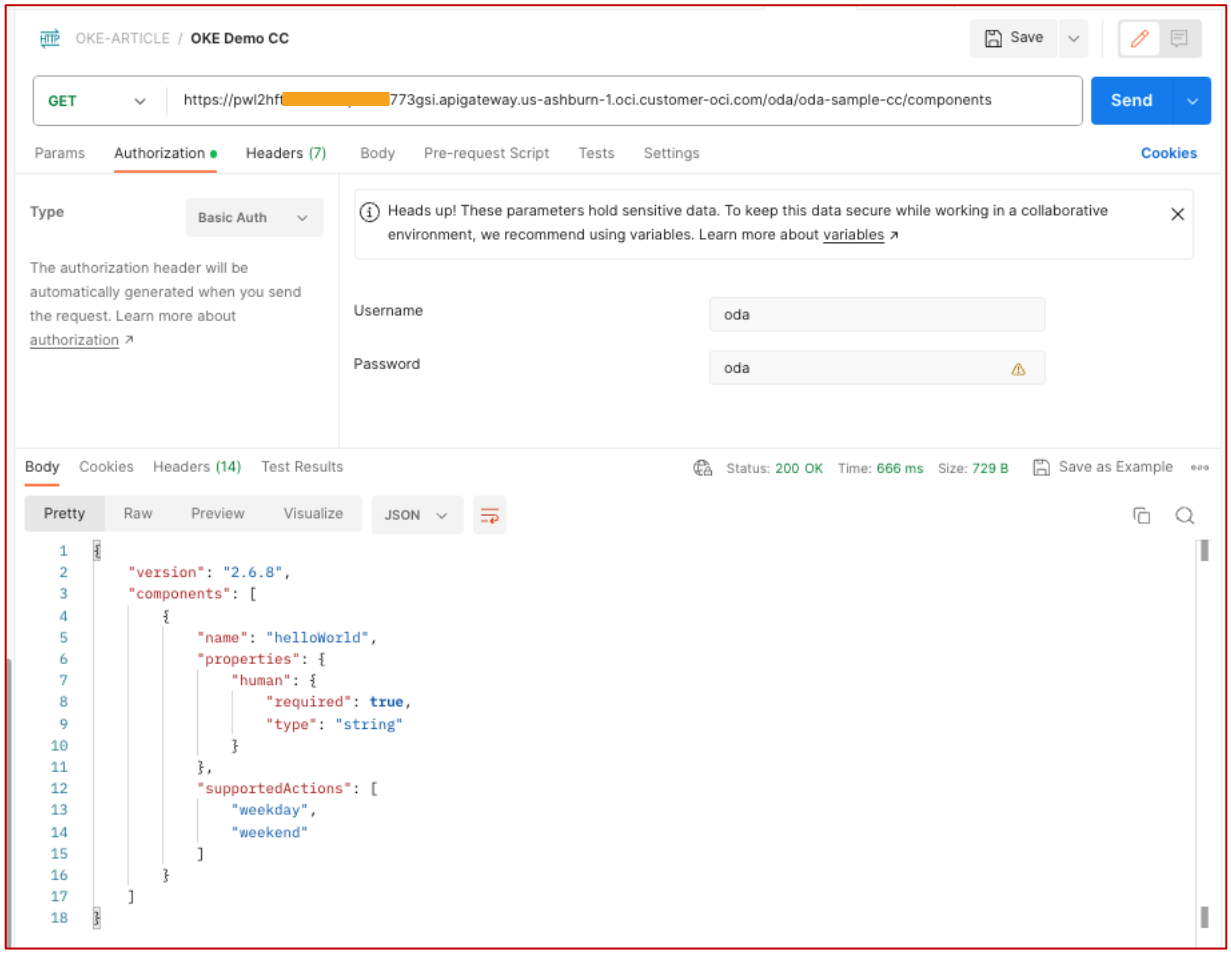

Figure 28: Status: 200 OK

A 502 error means there is an issue with the setup. There is a complete appendix in this article that can help troubleshoot the issue

6.4 Configure the skill with external custom component

Now that the custom component can be reached, the next step is to test in an Oracle Digital Assistant (ODA) Skill. Within the resources-part2.zip there is also a `SampleCCSkill(1.0).zip` provided.

Once logged into the Oracle Digital Assistant user interface, select Skills from the left menu pane. If the left menu bar is hidden, select the ☰ ORACLE Cloud menu icon to display the menu pane. In the Skills UI, select the Import Skills link, which is located on the top right of the main UI. Navigate to where the SampleCCSkill(1.0).zip is located and select **Open** to import the skill.

Note during the import process, the ODA skill will prompt a warning about the custom component import. This warning can be ignored, since the correct values will be provided in the next steps.

Open the imported SampleCCSkill.

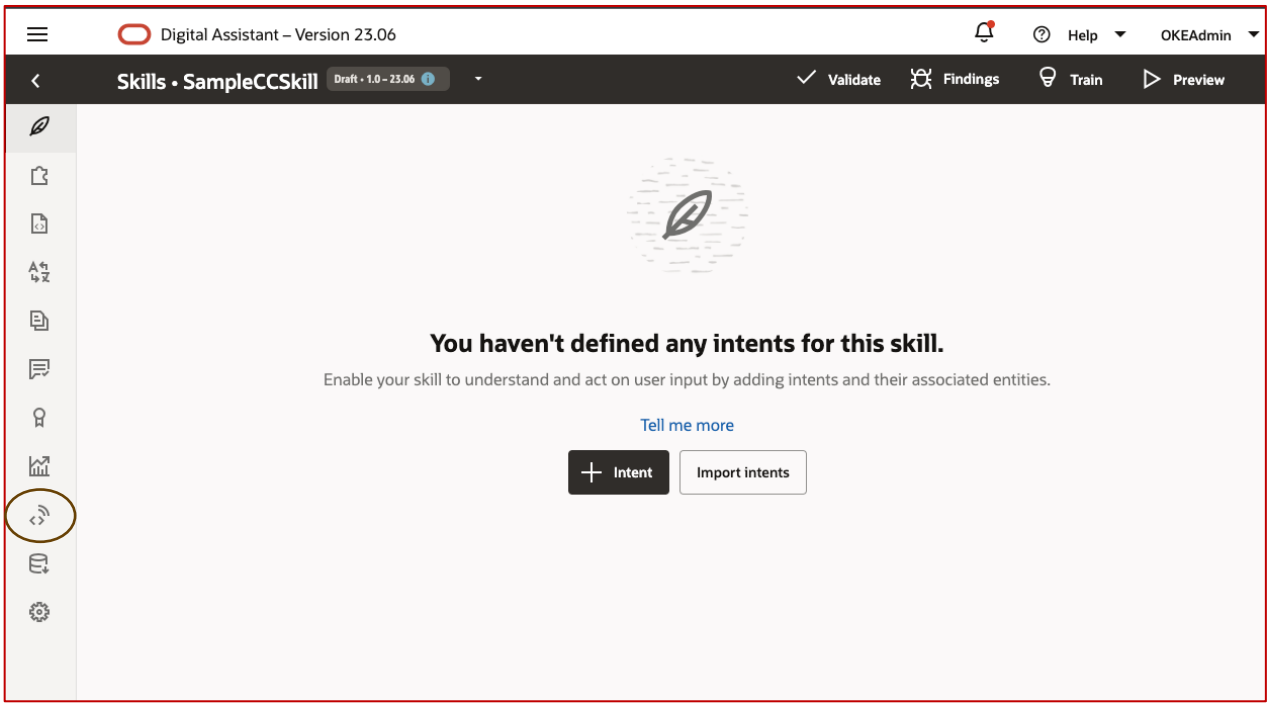

Figure 29: Open the SampleCCSkill

From the Skill's left menu pane, select the Custom Components service. There will be already a custom component service created named ExternalCCService.

Update the Metadata URL with the URL that was used to verify the end-point in Postman and also add in the Password: **oda**. After the auto save has happened, the Status will change from Error to **Ready**.

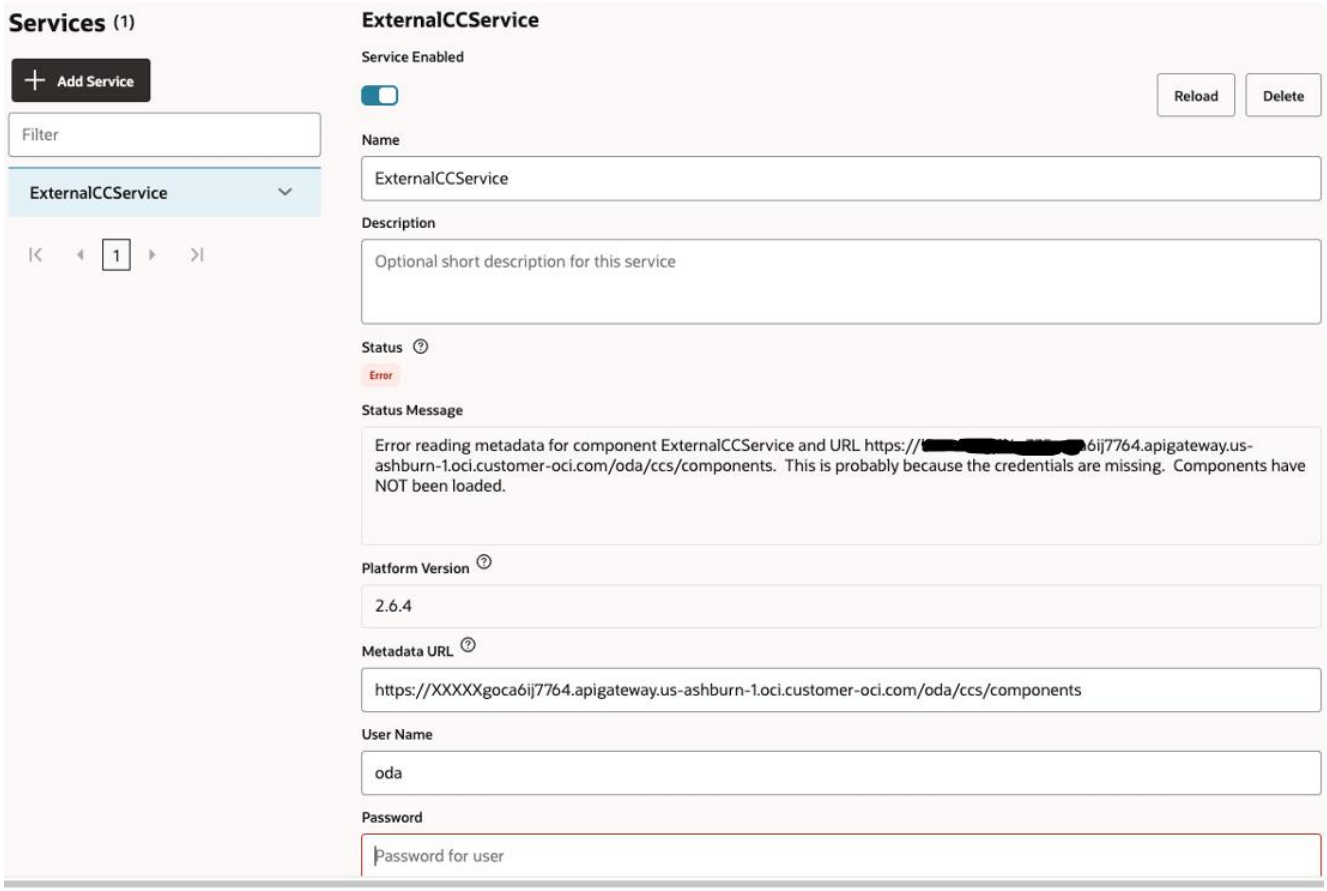

Figure 30: Configure External Service in ODA

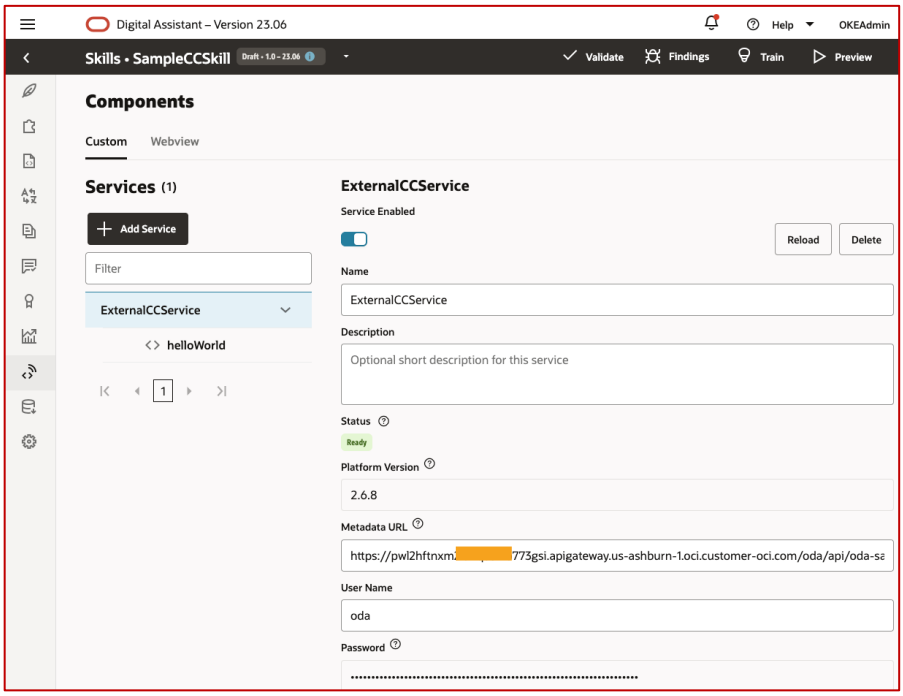

Figure 31: External service ready to be consumed.

Now ODA can call and invoke the external custom component. Before that can be done, the Sample CC requires a few modifications.

In the left menu pane, select the Flow Designer. In the Flow Designer UI select the UnresolvedIntent flow.

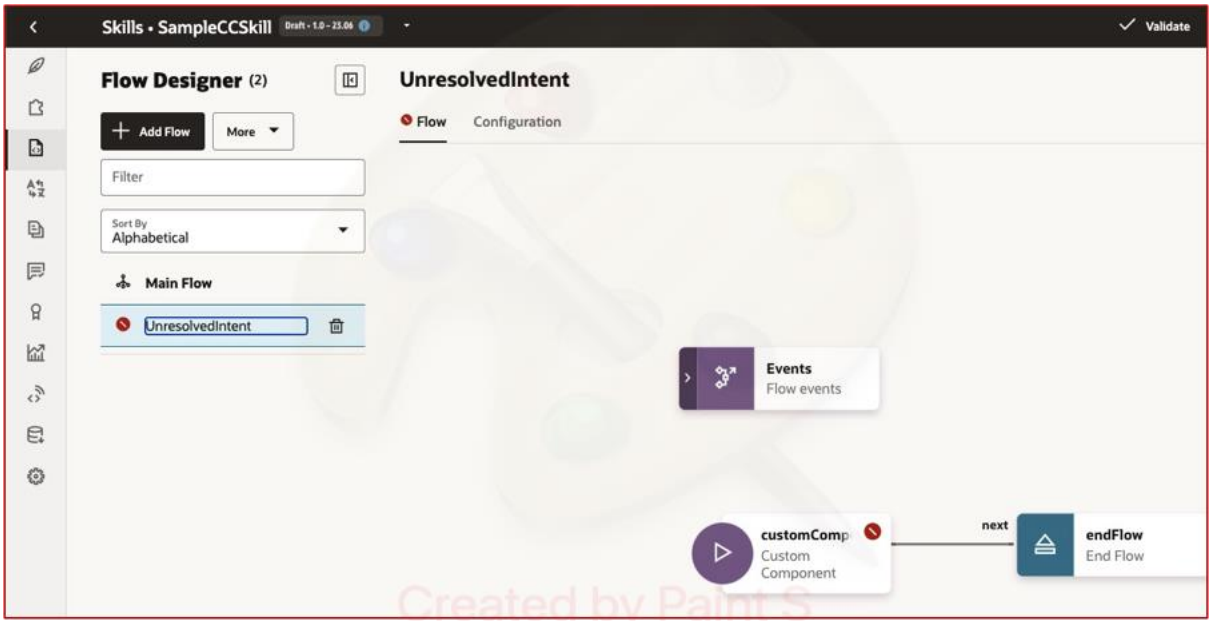

Figure 32: UnresolvedIntent flow

Notice that the custom Component state has a red warning. Select this state to open the properties dialog. Under the General tab, ensure that the Custom Component is selected from the drop down option.

Next, select the Component tab, where the red warning is. The is an attribute name **human**, which requires a value. This value is passed to the custom component, which in turn will be used as part of the response. Review the meta data of the custom component to locate this required value.

Next, input a value, for example, *Welcome to ODA*. Clicking out of the input field will result in ODA saving the value, which will also remove the alert warning.

Now the Skill can be tested. In ODA UI, select the **Preview** button, located in the top right. When the Bot Tester appears, send a message by typing **Hi**. The flow will be executed, which in turns calls the (external) custom component in OKE. The response will then return similar to what is shown below.

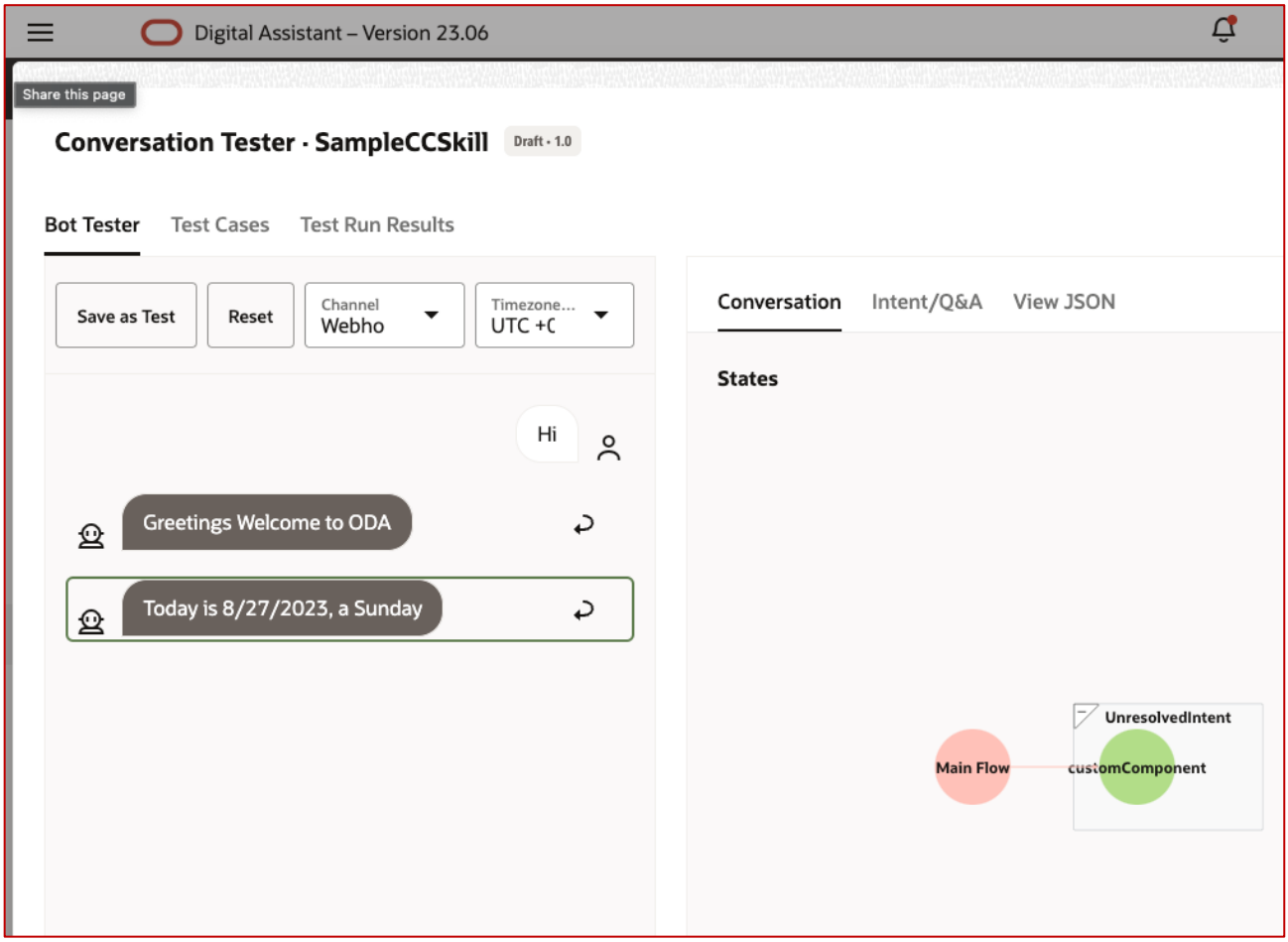

Figure 33: Run the test

This completes the deployment of custom components to the OKE cluster and testing from ODA. Part 3 will conclude this series of articles and will shows how a non admin user account can be configured with the necessary permissions to deploy custom components to API.

Conclusion

Deploying a custom component to OKE requires the custom component be made both Docker and Kubernetes ready. While this article covered deploying custom components to an OKE cluster, this same cluster can be used as an API hosting platform. For example, deployment of APIs to support backend systems such as PeopleSoft and/or CRM. These external APIs can also generate events to trigger Application Initiated Conversations with MS Teams or Slack through ODA.

Resources

Design Camp Session - Unleash The Power of OCI for ODA Part 1 - Custom Component Enterprise Deployment
https://videohub.oracle.com/media/Oracle+Digital+Assistant+Design+CampA+Unleash+The+Power+of+OCI+for+ODA+Part+1+-+Custom+Component+Enterprise+Deployment/1_rio2lqzq