Azure Deployment

Table of Contents

1. Folder Layout
2. Deployment
    1. Manual
    2. Automatic
3. Usage Examples

1. Folder contents

- docker: all contents for API endpoint server specific code and build code
- templates: contains `endpoint` and `registry` sub-folders with the parameters and template files for the Azure Container Registry and the Azure API App

2. Deployment Steps

2(i). Manual

The steps to deploy manually are broken into three main sections, the Azure Container Registry, building the docker image, and the API App.

Registry To deploy the registry, the only parameter that needs to be set is the name of the registry. This can be set in the `parameters.json` file in the `templates/registry` directory.

A name for the deployment, as well as a resource group to deploy into are also needed, but not in the `parameters.json` file.

Once a resource group is ready, a deployment name is chosen, and the `parameters.json` file is filled out, the registry can be deployed by running the following command from the `templates/registry` folder

```
az deployment group create \
     --name <deployment_name> \
     --resource-group <resource_group> \
     --template-file template.json \
     --parameters @parameters.json
```

Once this deploys, run the following commands to get the login information for the registry:

Username:

```
az acr credential show -n <deployment_name> --query username
```

Password:

```
az acr credential show -n <deployment_name> --query passwords[0].value
```

These values are needed for pushing to the docker registry.

Docker Before building the docker image, set the `connection_strings` list in `config/doc_cls_config.yaml` to all the connection strings for any storage accounts that the API App will have access to.

To build the docker image, run the following command from the root of the repo

```
docker build -t <registry>".azurecr.io/"<docker_image_name> -f azure/docker/dockerfile .
```

where `<registry>` is the name of the registry that was deployed earlier, and `<docker_image_name>` is the name of the image to build

Now that the image is built, it must be pushed to the Azure Container Registry. To push, the local docker client must be logged in to the registry. This can be done with the following command:

```
az acr login --name <registry> --user <registry_username> --password <registry_password>
```

where all of the values in `<>` are from the registry deployment section above.

Once logged in, the docker image can be pushed with `docker push <registry>".azurecr.io/"<docker_image_name>` where the values for `<registry>` and `<docker_image_name>` are the same as before.

API App To deploy the API App, there are some parameters that need to be changed from their default values in the `parameters.json` file in the `templates/endpoint` directory:

- `name` : This is the name of the API App, and must be globally unique for DNS purposes.
- `dockerRegistryUsername` : This is the username from the registry section
- `dockerRegistryPassword` : This is the password from the registry section
- `dockerRegistryUrl` : This can be made following the pattern: `https://<registry>.azurecr.io` where `<registry>` is the same as from the docker section
- `linuxFxVersion` : This can be made following the pattern: `DOCKER|<registry>.azurecr.io/<docker_i` with the same values from the docker section
- `hostingPlanName` : This is the name of the hosting plan that will be created, it is best to name it similar to the API name

There is also the whitelisted CIDR range in the `template.json` file in the `templates/endpoint` directory, in the top `variables` section. This will need to be changed to any CIDR range that is intended to access the endpoint.

Once the `parameters.json` and `template.json` files are filled out, the API App can be deployed by running the following command from the `templates/endpoint` folder

```
az deployment group create \
     --name <deployment_name> \
```

```
--resource-group <resource_group> \
--template-file template.json \
--parameters @parameters.json
```

The API App is now deployed.

2(ii). Automated

In this directory is a file `deploy.sh` that can be ran to deploy the whole setup. This will perform all of the same steps covered in the manual deployment, but using bash to automate the deployment.

Requirements for running `deploy.sh`

- the `az` Azure CLI tool is set up, and signed in to the Azure environment that will be deployed into
- The following are installed:
  - `bash`
  - `cut`
  - `tr`
  - `jq`
  - `cat`
  - `docker`
  - `uuid`

First, from the root of the repo, set the `connection_strings` list in `config/doc_cls_config.yaml` to all the connection strings for any storage accounts that the API App will have access to.

Next, open `deploy.sh`, at the top of the file is a section that looks like:

```
...

# declare all inputs in this script
# Azure inputs
resource_group="CerberusTestingDE"
# API and docker inputs
docker_image_name="docclass"
registry_name=$docker_image_name"Registry"
# CIDR whitelist
cidr_whitelist="73.219.252.215/32"

...
```

Here change the `resource_group` to the resource group to deploy into, and the `docker_image_name` to the base name for the API endpoint, this must be short enough that it does not go over Azures character limit with a UUID appended to it (ex `docclass` -> `docclass-5e3203ce-a9f1-11ec-8b3f-349971d41f7c`).

The `cidr_whitelist` is a single CIDR range that will be allowed to access the API Endpoint.

Once inputs are set, the script can be ran from the `azure` directory with

```
./deploy.sh
```

API Usage

To use the API, it can be sent POST requests with the following parameters for inputs:

- `input_path` : Required; the URL of the Azure Storage container that will be processed as a data room
- `output_path` : Optional; the URL of the Azure Storage container that the output will be saved to, can be a container/folder or a file name ending in `.json`
- `num_clusters` : Optional; manual setting for the number of clusters to sort the documents into

The API URL will follow the pattern `https://<name>.azurewebsites.net/` where `<name>` is either the manual deployment name, or the docker image name with a UUID added.

Once a POST request is made, the API Endpoint will give a response that looks like:

```
{"time_estimate":155.0,"size_estimate":0.25,"output_file_name":"2022-03-22_16-58-27_results.
```

This has a time estimate in seconds for how long processing will take, a size estimate in `mb` for how large the output JSON file will be, and the name of the output file that will be stored in Azure Blob Storage.

To view currently running data processors, and get the estimated time remaining for them, as well as see system utilization, a GET request can be made to the endpoint. Further input and output examples are documented below.

The output that the API Endpoint creates will be located at the same location as the `input_path`, unless `output_path` was specified. There will also be a `latest_results.json` output to the same location as the primary output, but this will overwrite any other `latest_results.json` file that is at that location.

The only exception in the response that is returned directly from the POST request is if the API Endpoint does not have access to the data room that was provided, and will look like:

```
{"error":"API does not have access to data room <data room name>"}
```

If the API encounters any errors while processing data, the exception will be output to the results location, in place of results themselves.

```
{'exception thrown': 'division by zero', 'traceback': '[\'  File "/app/./main.py", line 234,
```

POST Input Examples

Minimum required inputs:

```
api_url="https://test-2357deae-a492-11ec-b98d-349971d41f7c.azurewebsites.net/"

data='{"input_path":"https://testdataroomnyx.blob.core.windows.net/demodataroom"}'

curl -X POST \
     -H "Content-Type: application/json" \
     -d $data \
     $api_url
```

Custom output path and output file name:

```
api_url="https://test-2357deae-a492-11ec-b98d-349971d41f7c.azurewebsites.net/"

data='{"input_path":"https://testdataroomnyx.blob.core.windows.net/testdataroooooom","output_

curl -X POST \
     -H "Content-Type: application/json" \
     -d $data \
     $api_url
```

Custom output path and output file name and number of clusters:

```
api_url="https://test-2357deae-a492-11ec-b98d-349971d41f7c.azurewebsites.net/"

data='{"input_path":"https://testdataroomnyx.blob.core.windows.net/testdataroooooom","output_

curl -X POST \
     -H "Content-Type: application/json" \
     -d $data \
     $api_url
```

Custom number of clusters and default output path:

```
api_url="https://test-2357deae-a492-11ec-b98d-349971d41f7c.azurewebsites.net/"

data='{"input_path":"https://testdataroomnyx.blob.core.windows.net/demodataroom","num_cluste

curl -X POST \
     -H "Content-Type: application/json" \
     -d $data \
     $api_url
```

Data room that API Endpoint does not have access to:

```
api_url="https://test-2357deae-a492-11ec-b98d-349971d41f7c.azurewebsites.net/"
```

```
data='{"input_path":"https://containerthatapidoesnthaveaccessto.blob.core.windows.net/doesnt

curl -X POST \
     -H "Content-Type: application/json" \
     -d $data \
     $api_url
```

GET Input Examples To view running data processors and system utilization, any GET request will work.

```
api_url="https://test-2357deae-a492-11ec-b98d-349971d41f7c.azurewebsites.net/"

curl $api_url
```

will return an output similar to


```
{"running_data_processors":[{"started at":"2022/03/31 16:40:51","hours left":0.67},{"started
```


or


```
{"running_data_processors":[],"cpu_usage_percent":2.7,"memory_usage_percent":44.2}
```