

# Consume a real-time inferencing service

5 minutes

After deploying a real-time service, you can consume it from client applications to predict labels for new data cases.

## Use the Azure Machine Learning SDK

For testing, you can use the Azure Machine Learning SDK to call a web service through the `run` method of a `WebService` object that references the deployed service. Typically, you send data to the `run` method in JSON format with the following structure:

JSON

Copy

```
{
  "data": [
    [0.1, 2.3, 4.1, 2.0], // 1st case
    [0.2, 1.8, 3.9, 2.1], // 2nd case,
    ...
  ]
}
```

The response from the `run` method is a JSON collection with a prediction for each case that was submitted in the data. The following code sample calls a service and displays the response:

Python

Copy

```
import json

# An array of new data cases
x_new = [[0.1, 2.3, 4.1, 2.0],
          [0.2, 1.8, 3.9, 2.1]]

# Convert the array to a serializable list in a JSON document
json_data = json.dumps({"data": x_new})

# Call the web service, passing the input data
response = service.run(input_data = json_data)

# Get the predictions
predictions = json.loads(response)
```

```
# Print the predicted class for each case.
for i in range(len(x_new)):
    print (x_new[i], predictions[i])
```

## Use a REST endpoint

In production, most client applications will not include the Azure Machine Learning SDK, and will consume the service through its REST interface. You can determine the endpoint of a deployed service in Azure Machine Learning studio, or by retrieving the `scoring_uri` property of the **WebService** object in the SDK, like this:

Python

 Copy

```
endpoint = service.scoring_uri
print(endpoint)
```

With the endpoint known, you can use an HTTP POST request with JSON data to call the service. The following example shows how to do this using Python:

Python

 Copy

```
import requests
import json

# An array of new data cases
x_new = [[0.1,2.3,4.1,2.0],
         [0.2,1.8,3.9,2.1]]

# Convert the array to a serializable list in a JSON document
json_data = json.dumps({"data": x_new})

# Set the content type in the request headers
request_headers = { 'Content-Type': 'application/json' }

# Call the service
response = requests.post(url = endpoint,
                        data = json_data,
                        headers = request_headers)

# Get the predictions from the JSON response
predictions = json.loads(response.json())

# Print the predicted class for each case.
for i in range(len(x_new)):
    print (x_new[i], predictions[i] )
```


## Authentication

In production, you will likely want to restrict access to your services by applying authentication. There are two kinds of authentication you can use:

- **Key:** Requests are authenticated by specifying the key associated with the service.
- **Token:** Requests are authenticated by providing a JSON Web Token (JWT).

By default, authentication is disabled for ACI services, and set to key-based authentication for AKS services (for which primary and secondary keys are automatically generated). You can optionally configure an AKS service to use token-based authentication (which is not supported for ACI services).

Assuming you have an authenticated session established with the workspace, you can retrieve the keys for a service by using the **get\_keys** method of the **WebService** object associated with the service:

Python	 Copy
<pre>primary_key, secondary_key = service.get_keys()</pre>	

For token-based authentication, your client application needs to use service-principal authentication to verify its identity through Azure Active Directory (Azure AD) and call the **get\_token** method of the service to retrieve a time-limited token.

To make an authenticated call to the service's REST endpoint, you must include the key or token in the request header like this:

Python	 Copy
<pre>import requests import json  # An array of new data cases x_new = [[0.1,2.3,4.1,2.0],           [0.2,1.8,3.9,2.1]]  # Convert the array to a serializable list in a JSON document json_data = json.dumps({"data": x_new})  # Set the content type in the request headers request_headers = { "Content-Type":"application/json",                     "Authorization":"Bearer " + key_or_token }  # Call the service response = requests.post(url = endpoint,                         data = json_data,                         headers = request_headers)  # Get the predictions from the JSON response</pre>	

```
predictions = json.loads(response.json())

# Print the predicted class for each case.
for i in range(len(x_new)):
    print (x_new[i], predictions[i] )
```

---

## Next unit: Troubleshoot service deployment

[Continue >](#)

---