Create a Custom Service Broker

Goal

Create a simple custom Service Broker. The Service Broker will be implemented as a Cloud Foundry application, and will include the service itself.

Approximate time: 60 minutes

Note: This lab assume you have logged in as admin and targeted the development space of your Pivotal Cloud Foundry installation.

Exercises

Create a simple service

1. Inside of the directory of your choice (like your home directory), create a `myservice` directory. The myservice directory will contain the service itself as well as the Service Broker.
2. For the service itself, get the time of today’s sunset in Paris from the Yahoo Weather API. Curl is enabled by default in the PHP buildpack. Place the following code in an index.php file **directly in the myservice directory**:

```
<?php
$BASE_URL = "http://query.yahooapis.com/v1/public/yql";
$yql_query = 'select astronomy.sunset from weather.forecast where woeid in (select woeid from geo.places(1)
where text="paris")';
$yql_query_url = $BASE_URL . "?q=" . urlencode($yql_query) . "&format=json";

$session = curl_init($yql_query_url);
curl_setopt($session, CURLOPT_RETURNTRANSFER,true);
echo curl_exec($session);
curl_close($session);
?>
```

3. From the myservice directory, run `cf push` , naming your app `myservice` and giving it 128M of memory.
4. Using a browser, verify that your service returns JSON containing the sunset time of Paris.

Create a custom Service Broker catalog

1. Inside of your myservice directory, create the following directory structure: `v2/catalog` . The v2 directory will contain the Service Broker endpoints.
2. Inside of the v2/catalog directory, create an `index.php` file with the JSON shown below. You will use that JSON as a starting point for your Service Broker.

```
{
  "services": [{
    "id": "service-guid-here",
    "name": "mysql",
    "description": "A MySQL-compatible relational database",
    "bindable": true,
    "plans": [{
      "id": "plan1-guid-here",
      "name": "small",
      "description": "A small shared database with 100mb storage quota and 10 connections"
    }],
    "dashboard_client": {
      "id": "client-id-1",
      "secret": "secret-1",
      "redirect_uri": "https://dashboard.service.com"
    }
  }]
}
```

3. Change the contents of the JSON so that this Service Broker is unique. Specify any unique “id” under “services”, under “plans” and under "dashboard_client". Also give your Service Broker a unique name and description. You can call it a hello world service.
4. **Directly in the myservice directory**, place a file named `.htaccess` containing the following text. This file is used to avoid redirection when accessing a directory and to make sure that any request to `v2/catalog*` (where the * represents any string of characters) resolves to the `v2/catalog/index.php` page.

```
RewriteEngine on
DirectorySlash off

RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^v2/catalog(.*)$ v2/catalog/index.php [L]
```

5. From the myservice directory, run `cf push`, naming your app as you did before. **Notice that your service app is now also a Service Broker.**
6. Append `v2/catalog` to the url of your app and verify that you can see the JSON. Also ensure that `v2/catalog/` and `v2/catalog/anything` resolves to the JSON. You have created the endpoint for the `catalog` method of the Service Broker.
7. Register your Service Broker. Note that for simplicity we have not used http authentication, so any user name and password can be used. This of course is not recommended for the final Service Broker.

```
cf create-service-broker my_sb someuser password [path to your service app]
```

For example:

```
cf create-service-broker my_sb someuser password http://myservice.apps.sX.edu.pcfdemo.com
```

8. Using the cf CLI, use the `cf service-brokers` command to view the Service Brokers in your installation. You should see your new Service Broker.
9. Use the `cf marketplace` command view the services in your installation. Notice that your service is not listed. Use the `cf service-access` command to list the access for your services. Notice that your service is set to none.
10. Use the `cf enable-service-access <myservicename>` command to make the service accessible in the Marketplace. The <myservicename> is the name you specified near the beginning of your catalog JSON, and is visible from the `cf service-access` command.
11. Verify that your service is now available when using `cf marketplace`. Also verify that your service is visible in the Marketplace of Apps Manager.

Congratulations, you have implemented and tested the `catalog` method of a new Service Broker.

Implement the v2/service_instances Service Broker endpoint for provisioning

To provision a service instance, the Service Broker implements the v2/service_instances endpoint.

1. Inside of your `myservice/v2/` directory, create a `service_instances` directory.
2. Inside of the service_instances directory, create an `index.php` file with the following simple contents. Note that this Service Broker doesn't actually provision any resources.

```
{}
```

3. In the myservice directory, modify the contents of the file named `.htaccess` to contain the following text. This file is used to avoid redirection when accessing a directory and to make sure that any request to `v2/service_instances*` (where the * represents any string of characters) resolves to the `v2/service_instances/index.php` page.

```
RewriteEngine on
DirectorySlash off

RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^v2/catalog(.*)$ v2/catalog/index.php [L]

RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^v2/service_instances(.*)$ v2/service_instances/index.php [L]
```

4. From the myservice directory, run `cf push`, naming your app the same as you did previously. Note that your app is now a service that gives the sunset time as well as a Service Broker for that service.

5. Append `v2/service_instances` to the url of your app and verify that the result of provisioning the service is empty JSON. Also ensure that `v2/service_instances/` and `v2/service_instances/anything` resolves to the same content.
6. Use the `cf create-service` command to create an instance of your service in your development space.

```
cf create-service [service name from marketplace] [plan name from marketplace] myserviceinstance
```

7. Use `cf services` to verify that your service instance has been created. That simple service can now be used by applications in the development space.
8. (Optional) Bind an instance of this service to any app and view VCAP_SERVICES. This service is now behaving like other services in the marketplace.

Deprovisioning

1. View the Deprovisioning section of the Service Broker API documentation at https://docs.cloudfoundry.org/services/api.html. Since we did not actually provision any resources that we would need to reclaim, there is no deprovisioning necessary.
2. Execute the `cf delete-service` command to delete your service instance.

Implement the v2/service_instances/:instance_id/service_bindings Service Broker endpoint for binding

To bind an app instance to the service, the Service Broker implements the v2/service_instances/:instance_id/service_bindings endpoint.

1. We will use the existing service_instances/index.php file as a "controller" to read the request URI and determine if this is a provision request or a binding request. Replace the contents of service_instances/index.php with the following:

```
<?php
//check for 'service_bindings' in the request URI (binding calls will contain this)
$pos = strpos($_SERVER['REQUEST_URI'], 'service_bindings');

if ($pos ===false){
  //request did not contain 'service_bindings'
  //assume this is a provisioning request and return empty JSON
  echo '{}';
} else {
  //assume this is a binding request and return fake credentials
  echo '{
      "credentials": {
        "uri": "http://fakedomain.fake",
        "username": "fakeuser",
        "password": "fakepassword"
        }
      }';
}
?>
```

2. Notice that applications that bind to this service will just pass fake credentials. This is because the Yahoo API doesn't require credentials. If the service actually required credentials (like GitHub), this is how you would pass them.
3. From the myservice directory, run `cf push` , naming your app the same as you did previously. This app now contains a bindable Service Broker.
4. Use the `cf create-service` command (or Apps Manager) to create an instance of your service in your development space.

5. Select any existing application and bind this new service to that existing application. Use `cf bind-service` or Apps Manager to do this.
6. Using `cf env` or Apps Manager, view the environment variables for the binding application and notice that the fake credentials are included. These credentials can be used when calling the service from the binding application.

Congratulations! You have created a simple Service Broker.

Last updated 2016-01-09 09:34:53 PST