Service Registry Broker

A quick and easy way to expose pre-built, custom, existing or legacy services on the Cloud Foundry Marketplace without wrapping each service with its own service broker.

Overview

Most applications don't run in a vacuum or isolated mode. They are dependent on other applications and services running on the same or different platforms. Applications running on Cloud Foundry use the service bindings to associate service metadata with the consumer application in form of environment variables to relay information about the service, its endpoint, credentials or any other information relevant to invoke or consume a service.

Application developers can either use Cloud Foundry's user provided services to bind a service metadata to an application (and repeat this for each space the service needs to be made available) or build a service broker that would expose the same information in Cloud Foundry Marketplace and let the user bind the consuming application to the service.

The service broker approach to encapsulate the service metadata becomes a bit too tedious when there are large number of services, with each service requiring a service broker interface to be exposed in the Marketplace.

The Spring Cloud Service Registry is meant for microservices built using Spring Cloud connectors that use Netflix OSS Eureka service registry to control and manage client side load balancing calls to other associated services and does not solve the problem of exposing existing services via a service broker interface to any consuming application.

Lets look into an external Service Registry with Broker interface that can help Cloud Foundry customers address this specific problem.

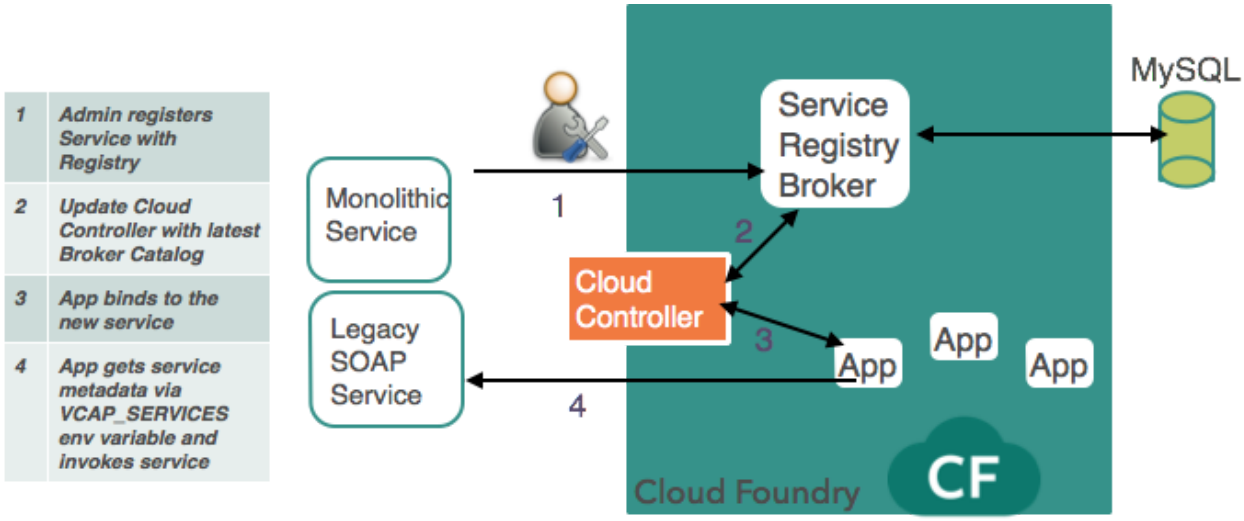
Service Registry Broker

Service Registry Broker is a prototype Cloud Foundry Service Broker exposing external service registry functionality to applications running on Cloud Foundry.

The Service Registry acts as a central repository to publish and consume information about pre-existing services like legacy SOAP Services or other user managed services like Databases, Custom Integration layers, etc. It does not manage, provision or monitor the services, only acts as a placeholder of information about services.

The registry exposes the registered services to Cloud Foundry via the Service Broker interface. A Cloud Foundry application bound to the underlying service would be able to consume information about the

underlying service (like endpoint and any credentials information) in the form of a VCAP_SERVICES environment variable when pushed to CF. The client/consumer app needs to parse the VCAP_SERVICES env variable and use the appropriate spring cloud connectors or other methods to use the endpoint configuration and invoke the related service.



1	*Admin registers Service with Registry*
2	*Update Cloud Controller with latest Broker Catalog*
3	*App binds to the new service*
4	*App gets service metadata via VCAP_SERVICES env variable and invokes service*

Each service definition is expected to contain a plan and associated credentials. The credentials encapsulate the information about the service and can be in the form of name value pairs like urls or endpoints about the service (can be behind a firewall or load balancer) or authentication tokens or just provide any arbitrary set of data (like cert name, locations, type, keys). Plans are mapped to environments, like Dev, Test, SIT, Prod and each would have its own credential.

```
{
  "bindable": true,
  "description": "TestCredsContentRetrievalSystem",
  "name": "TestCredsContentRetreiveInterface",
  "metadata": {
      "displayName": "TestCreds Content",
                                                                   "imageUrl":
"https://d1fto35gcfffzn.cloudfront.net/images/brandimages/P_WhiteOnTeal_RGB300.png",
      "longDescription": "Pull TestCreds content from content management system",
```

```
    "providerDisplayName": "Pivotal CloudFoundry Platform Eng",
    "supportUrl": "http://support.pivotal.io"
  },
"plans": [
  {
    "description": "Basic Plan throttled to 5 connections per second",
    "name": "basic",
    "free": true,
    "metadata": {
      "bullets": [ "TestCreds Content", "Free service" ]
    },
    "credentials": {
      "uri": "http://test-uri.10.10.10.10:8080",
      "username": "testuser",
      "password": "testuser",
      "certLocation" : "http://certLocation....",
      "certFormat" : "pem",
      "additionaltag1" : "additionalval1",
      "additionaltag2" : "additionalval2"
    }
  },
  {
    "description": "Premium Plan throttled to 25 connections per second",
```

```
    "name": "premium",
    "free": false,
    "metadata": {
      "bullets": [ "TestCreds Content", "Premium service" ]
    },
    "credentials": {
      "uri": "http://test-uri.20.20.20.20:8080",
      "username": "testuser",
      "password": "testuser",
      "certFormat" : "pem",
      "certLocation" : "http://certLocation....",
      "additionaltag1" : "additionalval1"          }
  }   ]   }
```

Configuration of a service can include multiple plans and associated credentials per plan (such as a bronze plan that will allow only 5 connections to a development instance endpoint while gold will allow 50 connections to a production instance endpoint). The data about the services, plans and credentials is persisted inside a database (using MySQL service binding). Administrator or allowed users can publish or update information about the services via a REST interface on the Service Registry.

Note:

The Service Broker does not create or provision any new set of service instances, or spin off new apps or containers or vms. The backend services are expected to be up and running at some endpoint as specified in the credentials. The Service Registry Broker does not manage or monitor the health or life cycle of underlying services and only acts as a bridging layer to provide information about the service to any consuming application, unlike Eureka or other services. No calls are ever made to the Service Registry at runtime when a consumer application invokes a bound service.

Configuring the service-registry-broker

- Deploy the backend or any test/simulation service. A sample simulation service is available at [document-service]
- Push the app to CF using manifest.yml. Edit the manifest to bind to a MySQL Service instance.

```
App service-registry-broker was started using this command `SERVER_PORT=$PORT $PWD/.java-buildpack/open_jdk_jre/bin/java -cp $PW
_jdbc-1.1.8.jar:$PWD/.java-buildpack/spring_auto_reconfiguration/spring_auto_reconfiguration-1.7.0_RELEASE.jar -Djava.io.tmpdir=
pack/open_jdk_jre/bin/killjava.sh -Xmx382293K -Xms382293K -XX:MaxMetaspaceSize=64M -XX:MetaspaceSize=64M -Xss995K org.springfram

Showing health and status for app service-registry-broker in org sampleOrg1 / space development as admin...
OK

requested state: started
instances: 1/1
usage: 512M x 1 instances
urls: service-registry-broker.classic.coke.cf-app.com
last uploaded: Fri Aug 28 16:22:10 UTC 2015

     state     since                    cpu    memory           disk           details
#0   running   2015-08-28 09:22:57 AM   0.0%   376.8M of 512M   131.1M of 1G
```

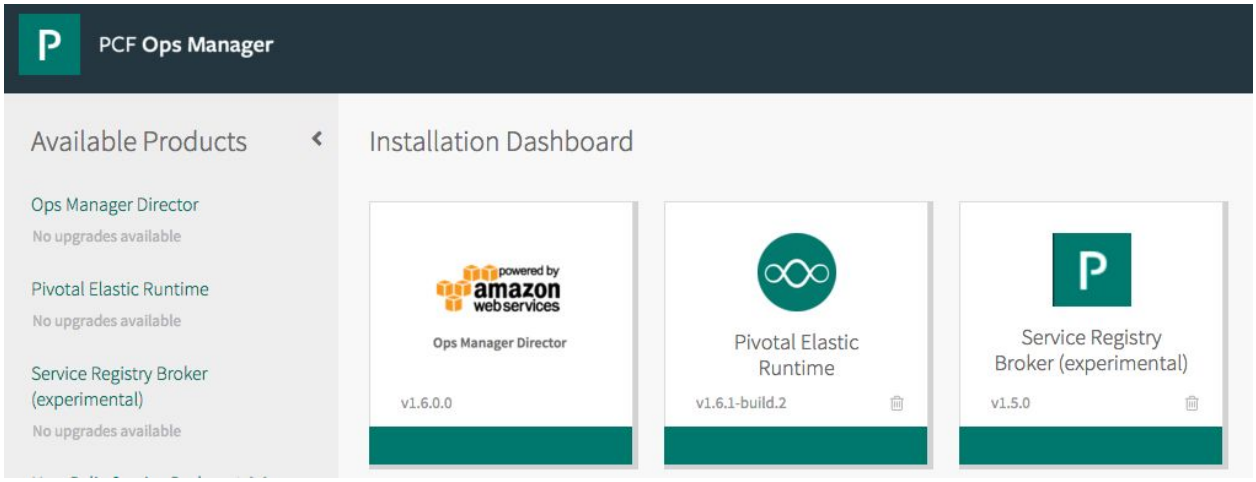
- Register the app as a service broker (this requires admin privileges on the CF instance) against Cloud Foundry (use create-service-registry call).

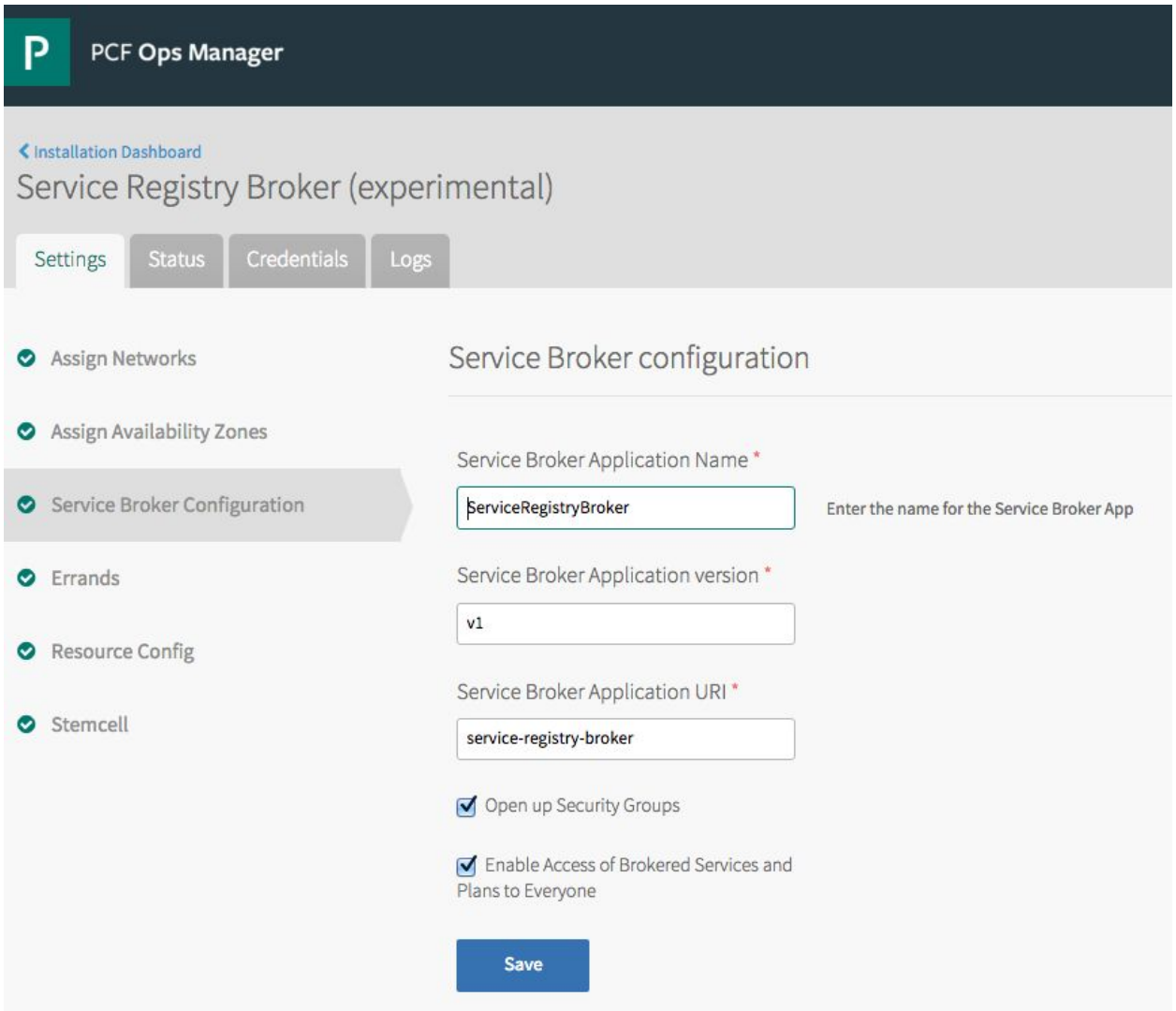
```
hammerkop:service-registry-broker sparameswaran$ cf create-service-broker service-registry-broker testuser testuser http://service-registry-broker.classic
.coke.cf-app.com
Creating service broker service-registry-broker as admin...
OK
hammerkop:service-registry-broker sparameswaran$ cf m
Getting services from marketplace in org sampleOrg1 / space development as admin...
OK

service          plans                              description
app-autoscaler   bronze, gold                       Scales bound applications in response to load
appdynamics      Plan1                              See How Your Apps Are Performing
newrelic         ecs, hybris, cloudfoundry, nrtravel  Manage and monitor your apps
p-mysql          400mb-dev                          MySQL service for application development and testing
```

- The Service Registry Broker Tile automatically deploys the app and register it as a service broker with CF and opening up access to embedded test services on the Marketplace.

Import the .pivotal Tile file into the Ops Mgr and add it. Minimal configuration is required.

Prerequisite: The service registry broker does require MySQL service to persists its service plans configurations.


PCF Ops Manager
Available Products
Ops Manager Director
No upgrades available
Pivotal Elastic Runtime
No upgrades available
Service Registry Broker (experimental)
No upgrades available
Installation Dashboard
powered by amazon web services
Ops Manager Director
v1.6.0.0
Pivotal Elastic Runtime
v1.6.1-build.2
Service Registry Broker (experimental)
v1.5.0


PCF Ops Manager
Installation Dashboard
Service Registry Broker (experimental)
Settings
Status
Credentials
Logs
Assign Networks
Assign Availability Zones
Service Broker Configuration
Errands
Resource Config
Stemcell
Service Broker configuration
Service Broker Application Name *
ServiceRegistryBroker
Enter the name for the Service Broker App
Service Broker Application version *
v1
Service Broker Application URI *
service-registry-broker
Open up Security Groups
Enable Access of Brokered Services and Plans to Everyone
Save

- Expose the services/plans within a specific org or publicly accessible.

```
# Then enable service-access for the service against the org/space or for everyone.
cf enable-service-access PolicyInterface
```

```
broker: service-registry-broker
   service                          plan      access   orgs
   PolicyInterface                  basic     none
   PolicyInterface                  premium   none
   TestCredsContentRetreiveInterface premium  none
   TestCredsContentRetreiveInterface basic    none
   EDMSRetreiveInterface            basic     none
   EDMSRetreiveInterface            premium   none

hammerkop:service-registry-broker sparameswaran$ cf enable-service-access TestCredsContentRetreiveInterface
Enabling access to all plans of service TestCredsContentRetreiveInterface for all orgs as admin...
OK
hammerkop:service-registry-broker sparameswaran$
```

Consuming the services

- Lookup available service plans from command line (cf marketplace) or Marketplace via Apps Manager.

```
cf marketplace
```

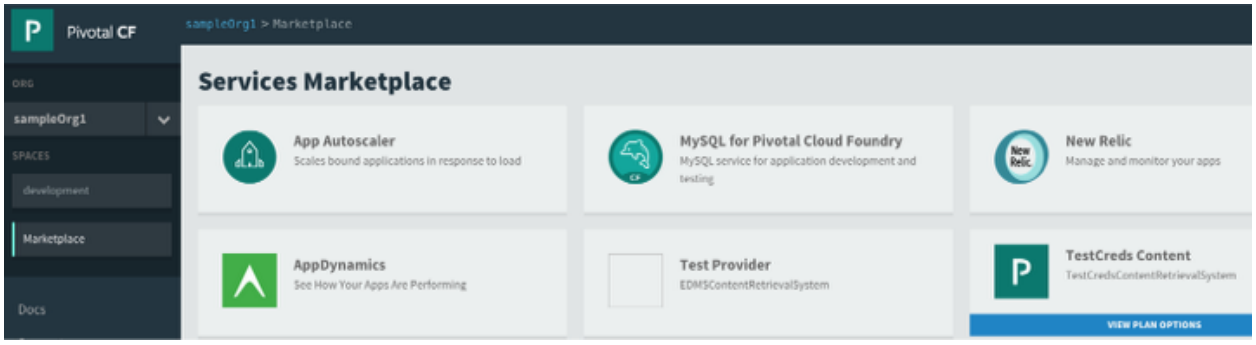
```
hammerkop:service-registry-broker sparameswaran$ cf m
Getting services from marketplace in org sampleOrg1 / space development as admin...
OK

service                          plans                              description
EDMSRetreiveInterface            basic, premium*                    EDMSContentRetrievalSystem
TestCredsContentRetreiveInterface premium, basic                    TestCredsContentRetrievalSystem
app-autoscaler                   bronze, gold                       Scales bound applications in response to load
appdynamics                      Plan1                              See How Your Apps Are Performing
newrelic                         ecs, hybris, cloudfoundry, nrtravel Manage and monitor your apps
p-mysql                          400mb-dev                          MySQL service for application development and testing
```

- Create a service based on the plan from command line or marketplace.

Create a service based on a service defn and plan
cf create-service EDMSRetreiveInterface basic EDMSRetreiveInterface-basic

```
hammerkop:service-registry-broker sparameswaran$ cf create-service EDMSRetreiveInterface basic EDMSRetreiveInterface-basic
Creating service EDMSRetreiveInterface-basic in org sampleOrg1 / space development as admin...
OK
hammerkop:service-registry-broker sparameswaran$ cf services
Getting services in org sampleOrg1 / space development as admin...
OK

name                        service                 plan       bound apps                    last operation
TestCredsService1           EDMSRetreiveInterface   basic      dora                          create succeeded
EDMSRetreiveInterface-basic EDMSRetreiveInterface   basic                                    create succeeded
mysql-service               p-mysql                 400mb-dev  push, service-registry-broker create succeeded
```

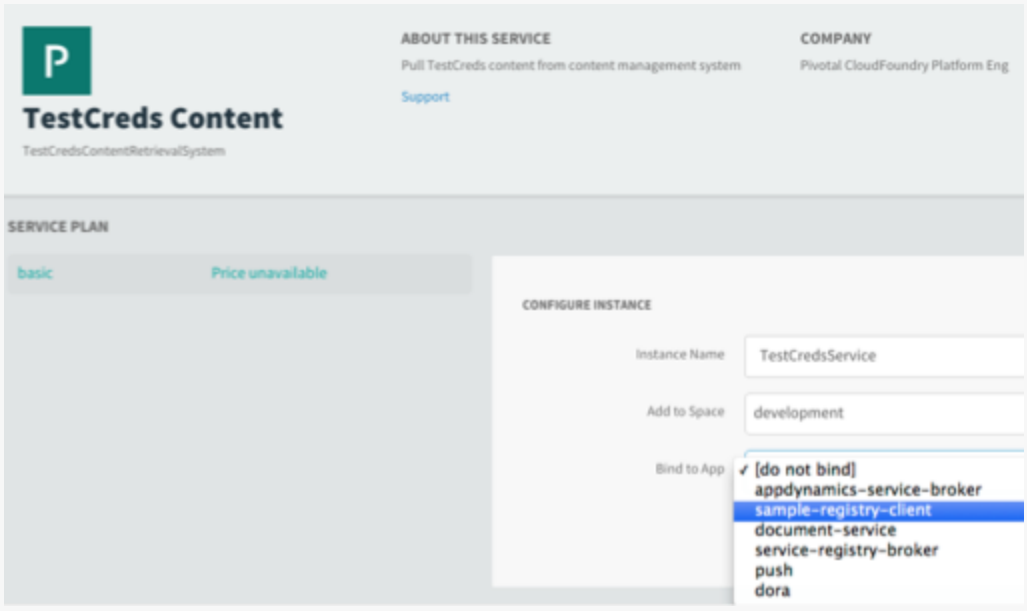
or from Marketplace on Apps Manager



- Deploy the client app that would bind to the service and consume the service. A sample client app is available on github at [sample-doc-retrieve-gateway client]

Now push a sample client app that can bind to the newly created service

cf bind-service sample-registry-client EDMSRetreiveInterface-basic



Add/Edit Services and plans from Web UI

The service registry broker application also bundles a web interface to interact and manipulate services and plan definitions. Use the security credentials for the application (default is testuser/testuser, if pushed manually and check the Ops Mgr for generated credentials if the application was installed as Tile) to access the default web page (ie. http://service-registry-broker.appdomain.cf...).

Using the web interface, users can add new services (in json format), edit service or plan definitions, add new services, plans, tweak the set of attributes that goes as part of the credentials section for a given service binding.

Figure: Service Registry Broker Web UI

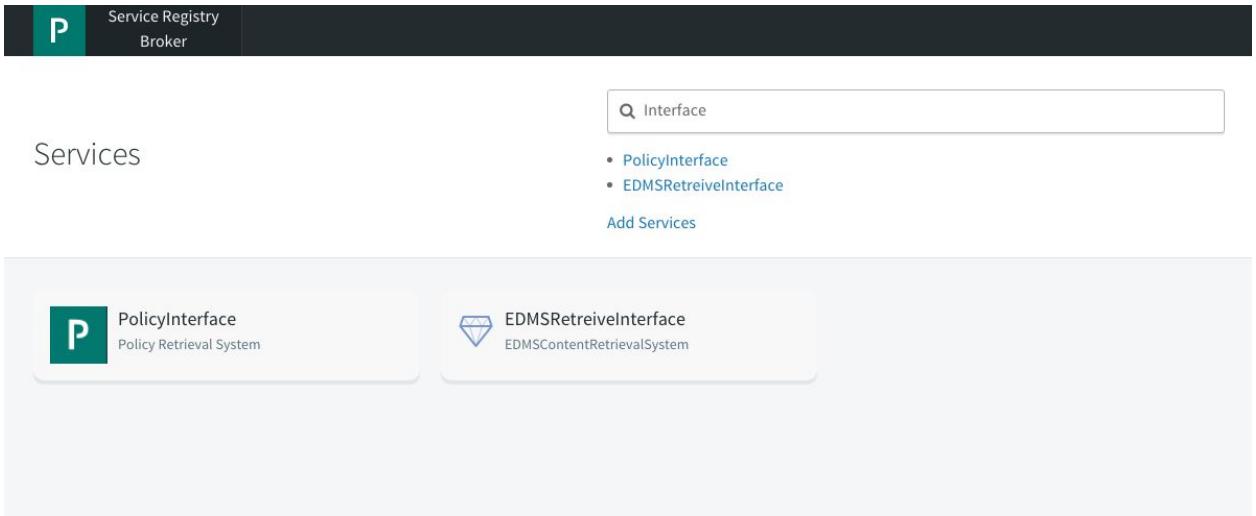


Figure: Service Definition Page

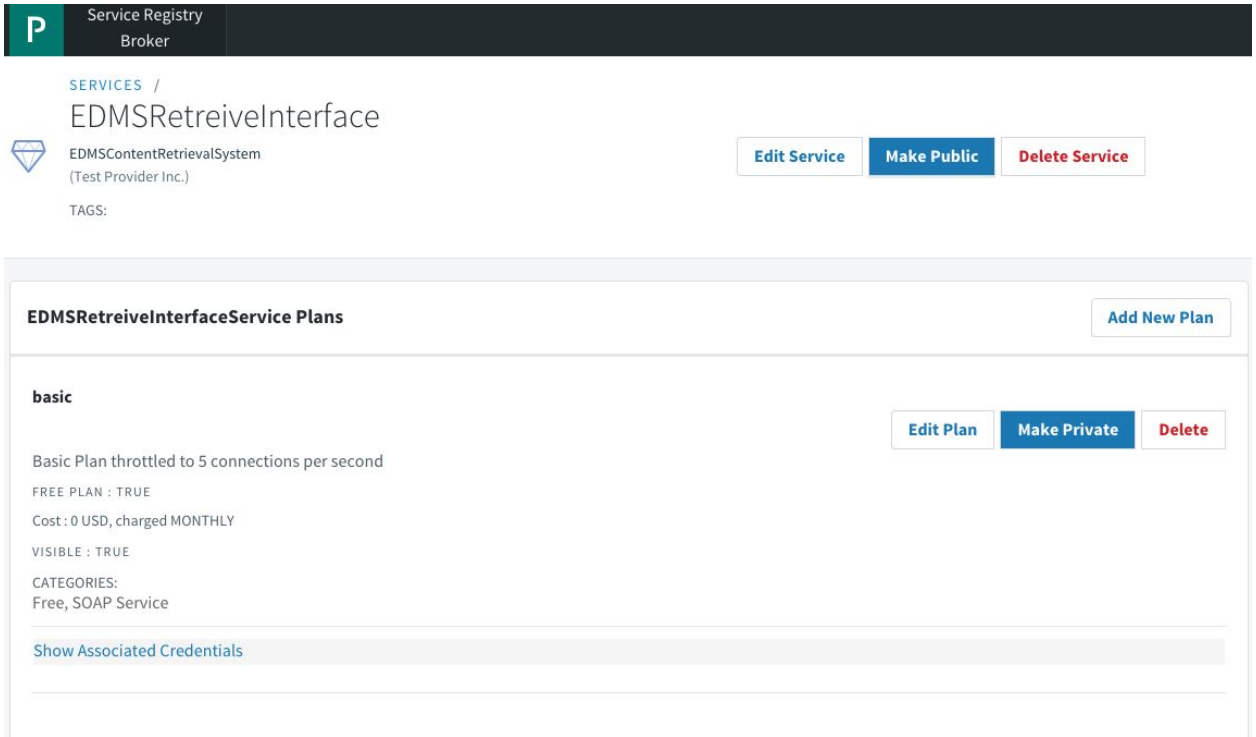


Figure: Service Plan Details

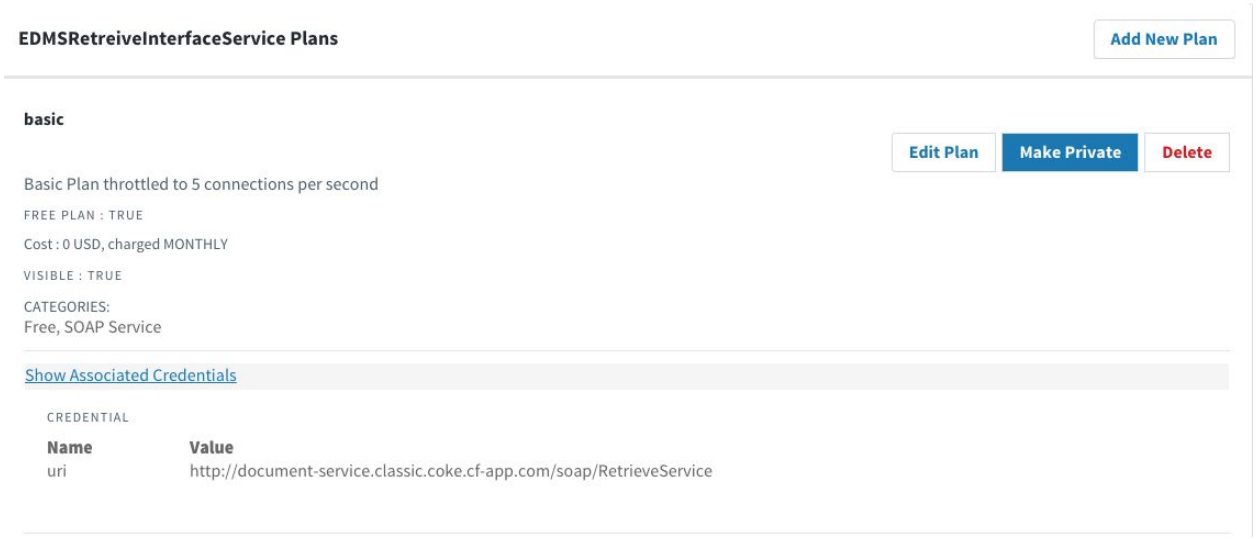


Figure: Change Service Visibility in Marketplace

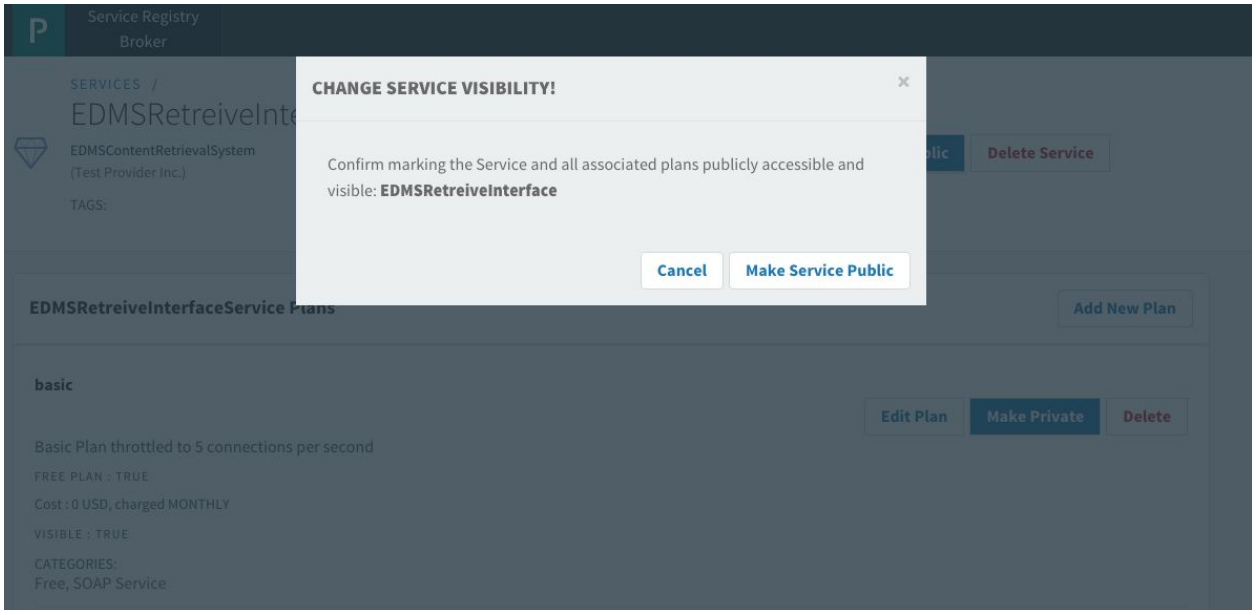
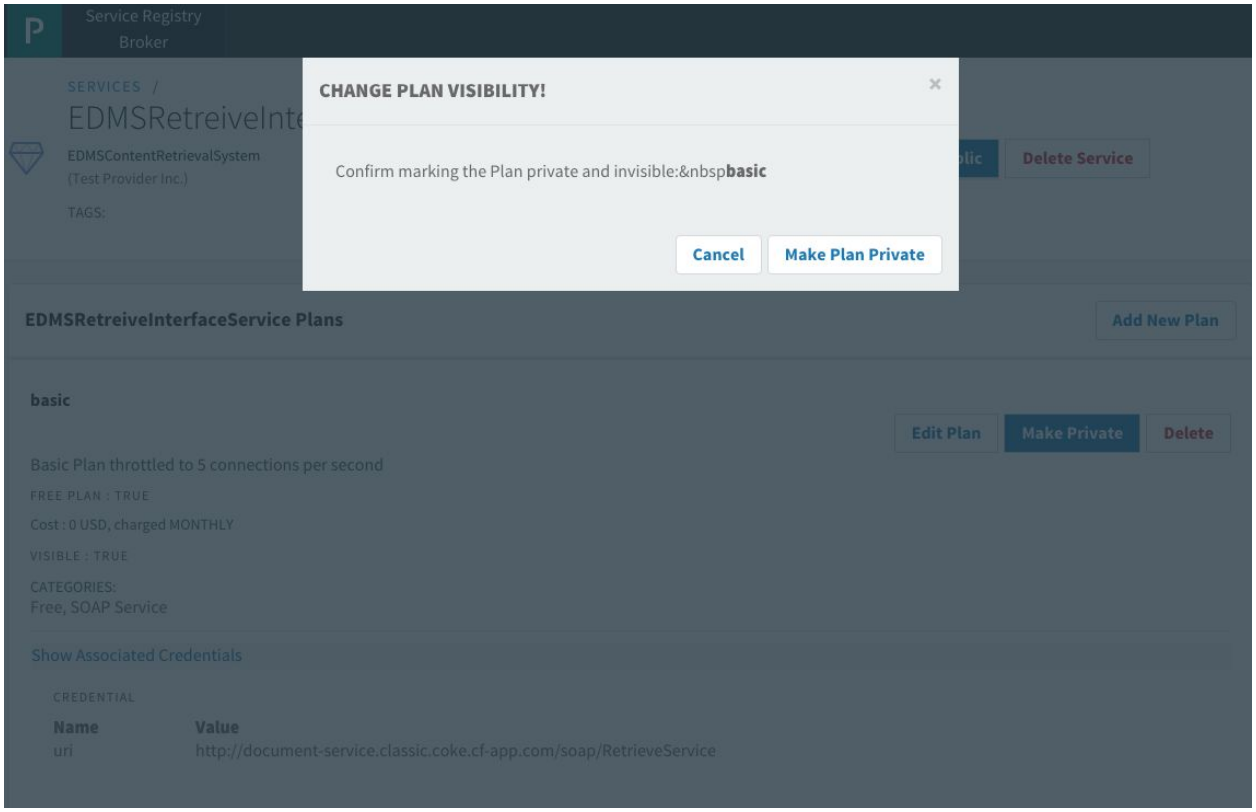


Figure: Change Plan Visibility



Service Editor

Service Name

PolicyInterface

Service Description

Policy Retrieval System

Display name

Service Provider

Long Description

Enter Long Description

Provider Description

Service Provider Inc.

Image Url

/images/pivotal-img.png

Documentation Url

Enter Documentation Url

Support Url

Enter Support Url

Submit

Figure: Edit Service

Plan Editor

Plan Name

basic

Plan Description

Basic Plan throttled to 5 connections per second

Free Plan

Costs

Amount (in double)

0

Note: PCF Console (Apps Manager) will not show the plan in Marketplace for an Organization, if cost is set to non-zero amount until the org quota is modified to allow access to non-basic (free) services

Currency

usd

Units

MONTHLY

Tags

Name	Action
Free, SOAP Service	Remove

Add a tag

Tag

Add Tags

Credentials

Name	Value	Action
uri	http://policy-service.classic.coke.cf-app.com/soap/RetrieveService	Remove

Add an entry

Name

Value

Add Entry

Submit

Figure: Edit Plan

Service Registry Broker

Add one or more Service definitions

Modify the JSON template below...!!


```
[
 {
  "name": "ServiceName1",
  "description": "Sample Description 1",
  "bindable": true,
  "metadata": {
   "displayName": "Short display name..",
   "imageUrl": "/images/pivotal-img.png",
   "longDescription": "... Long description....",
   "providerDisplayName": "...Eng team or Org .....",
   "supportUrl": "http://CompanyUrl..."
  },
  "plans": [
   {
    "description": "Description of Plan.....",
    "name": "SomeSampleQAPlan1",
    "free": true,
    "metadata": {
     "bullets": [
      "Dev plan for some service",
      "Service Type",
      "Free service"
     ],
     "costs": [
      {
       "amount": {
        "usd": 0
       },
       "unit": "MONTHLY"
      }
     ]
    },
    "credentials": {
     "uri": "http://dev-service.services1.xyz.com:8080",
     "username": "testuser",
     "password": "testpassword",
     "certLocation": "http://certLocation.xyz.com/.....",
     "certFormat": "pem",
     "additionaltag1": "additionalval1",
     "additionaltag2": "additionalval2"
    }
   },
   {
    "description": "Description of Additional Plan like Premium, SIT, Prod....",
    "name": "SomeSampleSITPlan",
    "free": false,
```

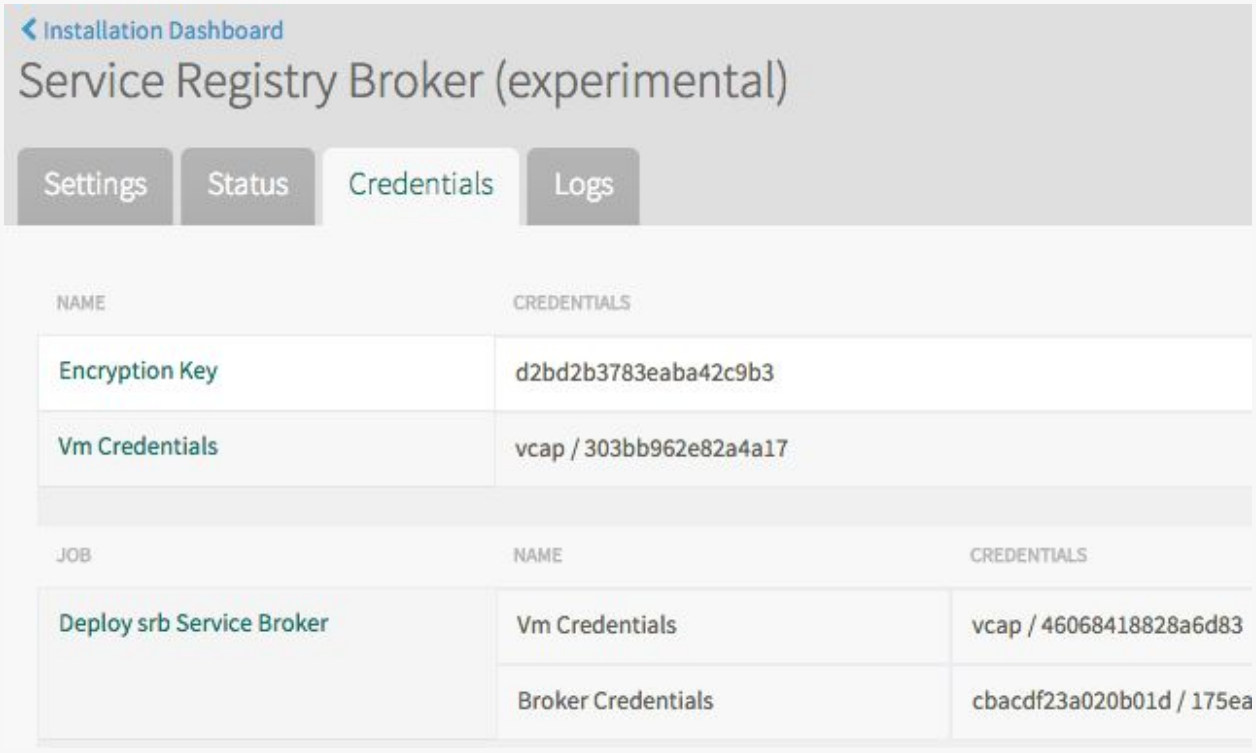

Figure: Add Services

Note: Auto-update and service access enabling is supported in this experimental build of the application.

Updating the catalog

After a new service has been registered or after updates on the service registry, update of the catalog with the Cloud Foundry controller is automatically handled by the Service Registry Broker.

Note: If using tile to deploy/install the service registry broker, get the Broker Credentials from the Ops Mgr Credentials page for the tile and use that to communicate with the service broker app.



Using the Service Registry REST interface

Refer to the README in the repo to get an overview of the REST api supported by the service registry to create/read/update/delete service definitions, plans and credentials.

Notes

- There can be multiple set of services, each with unlimited plans.
- Each plan in any service would be associated with one and only credentials row. The Plan can be space or env specific and allow the service instance to use the set of credentials associated with the plan.
- Users can provide additional parameters during service creation or binding time (using cf cli command line) to override/add additional metadata to the binding credentials.
- The Web UI interface allows either opening up access to all plans or just a specific plan within a service. For more fine grained control of plan access to specific orgs/spaces, refer to Service Plans Access control.

Summary

In conclusion, the Service Registry Broker allows any new, existing or legacy applications or services to surface as readily available services in the Cloud Foundry Marketplace via the Service Broker interface, thereby allowing application developers to leverage the services in a easy manner, without rewriting or creating yet another service broker for every service.