VMware vCloud® Architecture Toolkit™
for Service Providers

Extending VMware vCloud® API with vCloud Extensibility Framework

Version 2.9

January 2018

John Dwyer

VMware, Inc.
3401 Hillview Ave
Palo Alto, CA 94304
www.vmware.com

Contents

Introduction

As more customers embrace development and operations (DevOps), for automating the process of software delivery and infrastructure changes, it is important to provide users with the necessary tools to interact with your complete IaaS platform. By providing your services as an extension to the VMware vCloud® API, you provide a single point of integration for your user interface (custom or third-party) as well as your tenant customers automation tools.

The VMware vCloud Director® Extension Framework gives service providers the power to extend the standard API included with vCloud Director. Service providers can register a URL pattern that, when called, will be routed to custom code to execute through the Advanced Message Queuing Protocol (AMQP).

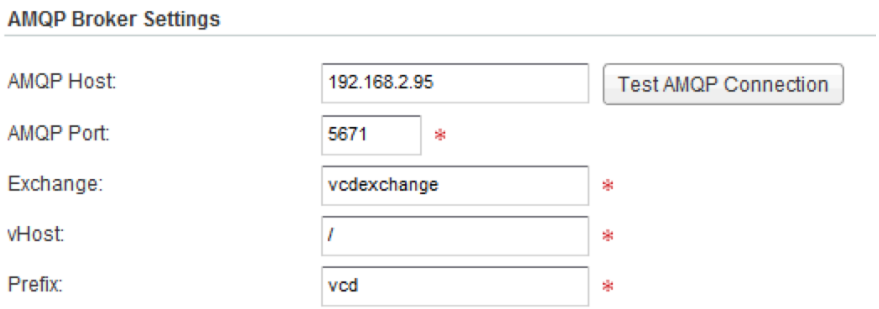
Some example services you might consider implementing using vCloud API calls include the following:

- Ability to order and manage services
  - Public IP space
  - Operating system support
- Ability to open, update, and close trouble ticket requests
- Ability to configure:
  - Anti-virus
  - Backups
  - Disaster recovery

1.1 vCloud Director API Extension Requirements

The requirements for using vCloud Director API extensions include the following:

- A vCloud Director user account with system administrator privileges.
- vCloud Director must be configured to use AMQP: **Administration** > **System Settings** > **Extensibility.**



  After configuring the AMQP settings, verify your AMQP connection.

- A programming language of your choice with bindings for RabbitMQ. (RabbitMQ provides the AMQP implementation that is used with vCloud Director.)
- (Optional) Ability to make command-line API calls. In the following examples, HTTPie is utilized.

Command-Line API Calls

To simplify making API calls from the command line, HTTPie is used for all demo examples. HTTPie is a simple client that provides cURL-like functionality.

All example code can be found on GitHub.

2.1 Log in to vCloud Director API using HTTPie

You must be logged in as a system administrator. This example uses a local system administrator account called `api-user`.

```
# http --verify no --session=vcloud-gcp-admin -a api-user@System POST
https://vcd.gcp.local/api/sessions 'Accept:application/*+xml;version=5.6'
```

This example uses a self-signed certificate on a vCloud Director instance, so the `–verify no` option is passed. After you log in, the session is stored in `vcloud-gcp-admin` to reuse on future requests.

The output resulting from execution of the example should look something like the following:

```
HTTP/1.1 200 OK

Content-Length: 1390

Content-Type: application/vnd.vmware.vcloud.session+xml;version=5.6

Date: Tue, 08 Dec 2015 19:49:08 GMT

Date: Tue, 08 Dec 2015 19:49:09 GMT

Set-Cookie: vcloud-token=304715fbb28e4ad19efeb6adfe850184; Secure; Path=/

X-VMWARE-VCLOUD-REQUEST-EXECUTION-TIME: 1076

X-VMWARE-VCLOUD-REQUEST-ID: b64552a4-054a-42e4-9ae1-0ea7fabb56a4

x-vcloud-authorization: 304715fbb28e4ad19efeb6adfe850184


<?xml version="1.0" encoding="UTF-8"?>

<ns0:Session xmlns:ns0="http://www.vmware.com/vcloud/v1.5"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" href="https://vcd.gcp.local/api/session"
org="System" type="application/vnd.vmware.vcloud.session+xml" user="api-user"
userId="urn:vcloud:user:44fbd6f9-7a76-4bca-b273-3536b181ad09"
xsi:schemaLocation="http://www.vmware.com/vcloud/v1.5
http://vcd.gcp.local/api/v1.5/schema/master.xsd">

    <ns0:Link href="https://vcd.gcp.local/api/org/" rel="down"
type="application/vnd.vmware.vcloud.orgList+xml" />

    <ns0:Link href="https://vcd.gcp.local/api/session" rel="remove" />

    <ns0:Link href="https://vcd.gcp.local/api/admin/" rel="down"
type="application/vnd.vmware.admin.vcloud+xml" />

    <ns0:Link href="https://vcd.gcp.local/api/admin/extension" rel="down"
type="application/vnd.vmware.admin.vmwExtension+xml" />

    <ns0:Link href="https://vcd.gcp.local/api/org/a93c9db9-7471-3192-8d09-a8f7eeda85f9"
name="System" rel="down" type="application/vnd.vmware.vcloud.org+xml" />

    <ns0:Link href="https://vcd.gcp.local/api/query" rel="down"
type="application/vnd.vmware.vcloud.query.queryList+xml" />

    <ns0:Link href="https://vcd.gcp.local/api/entity/" rel="entityResolver"
type="application/vnd.vmware.vcloud.entity+xml" />

    <ns0:Link href="https://vcd.gcp.local/api/extensibility" rel="down:extensibility"
type="application/vnd.vmware.vcloud.apiextensibility+xml" />

</ns0:Session>
```

To reuse the session, simply specify the `--session=vcloud-gcp-admin` parameter on all future calls.

The following example lists all organizations in the vCloud Director instance:

```
# http --verify no --session=vcloud-gcp-admin --pretty colors GET
https://vcd.gcp.local/api/org/
```

Returns:

```
<?xml version="1.0" encoding="UTF-8"?>
<OrgList xmlns="http://www.vmware.com/vcloud/v1.5" href="https://vcd.gcp.local/api/org/"
type="application/vnd.vmware.vcloud.orgList+xml" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.vmware.com/vcloud/v1.5
http://vcd.gcp.local/api/v1.5/schema/master.xsd">
    <Org href="https://vcd.gcp.local/api/org/a93c9db9-7471-3192-8d09-a8f7eeda85f9" name="System"
type="application/vnd.vmware.vcloud.org+xml"/>
    <Org href="https://vcd.gcp.local/api/org/9aee51e8-654e-49a8-8dab-3fdbf00a21ae" name="Coke"
type="application/vnd.vmware.vcloud.org+xml"/>
    <Org href="https://vcd.gcp.local/api/org/2ce0365d-4d7d-4c15-a603-9257ea338c99" name="Pepsi"
type="application/vnd.vmware.vcloud.org+xml"/>
</OrgList>
```

Extension Example

This example integrates a ticketing system with vCloud Director. This can be an integration into an open source project, an existing enterprise tool, or a "homegrown" system. The goal of the integration in this example is to allow tenants to open generic tickets at the customer level. This integration is done using custom scripts found at https://github.com/johnnycuse/vcd-api-examples/tree/master/ticketing.

3.1 Register an Extension

To create a custom extension, you must POST a *service* element. A service element describes the name, namespace, routing key, exchange to use, and URL patterns (APIFilters) and ServiceLink elements to be used as API filters.

The parameters used to create the extension are as follows:

- Name – Name of the entity.
  - For example: `gcp-ticketing`.
- Namespace – Service namespace, which must be unique among all registered extension services. Service namespace names must follow the naming convention used for Java packages.
  - This example uses the namespace `local.gcp.ticketing`.
- RoutingKey – AMQP routing key that vCloud Director must use when routing messages to the service.
  - This example uses the routing key `gcp-ticketing`.
- Exchange – AMQP exchange name that vCloud Director must use when routing messages to the service. The service must create the specified exchange on the AMQP service that vCloud Director uses.
  - This example uses the AMQP exchange name `vcdext`, which is different than the one that was configured with vCloud Director. The exchange used for extensions must be of the type Direct.
- APIFilters – One or more URL patterns that the vCloud Director REST service must treat as extension requests. URL patterns can include regular expressions recognizable by `java.util.regex.Pattern`.
  - For example: `(/api/org/.*/ticketing/*[0-9]*)`
    - This allows you to match on:

      `/api/org/org-uuid/ticketing`

      `/api/org/org-uuid/ticketing/`

      `/api/org/org-uuid/ticketing/12345`
    - In this case, the addition to org is limited. If you wanted to also allow this creation on individual VMs, you could specify:

      `(/api/org/.*/ticketing*[0-9]*)|(/api/vApp/vm-.*/ticketing*[0-9]*)`

The XML for the API extension is created in a file called `ext-reg.xml` (https://github.com/johnnycuse/vcd-api-examples/blob/master/ticketing/ext-reg.xml), and this will be passed to vCloud to register the extension:

```
<?xml version="1.0" encoding="UTF-8"?>
<vmext:Service xmlns="http://www.vmware.com/vcloud/v1.5"
xmlns:vmext="http://www.vmware.com/vcloud/extension/v1.5" name="gcp-
ticketing">
```

```
    <vmext:Namespace>local.gcp.ticketing</vmext:Namespace>
    <vmext:Enabled>true</vmext:Enabled>
    <vmext:RoutingKey>gcp-ticketing</vmext:RoutingKey>
    <vmext:Exchange>vcdext</vmext:Exchange>
    <vmext:ApiFilters>
       <vmext:ApiFilter>
          <vmext:UrlPattern>(/api/org/.*/ticketing/*[0-9]*)</vmext:UrlPattern>
       </vmext:ApiFilter>
    </vmext:ApiFilters>
</vmext:Service>
```

To register, POST to `/api/admin/extension/service`

with:

```
Content Type: application/vnd.vmware.admin.service+xml
'Accept:application/*+xml;version=5.6
```

To do this with HTTPie, execute the following command:

```
# http --verify no --session=vcloud-gcp-admin POST
https://vcd.gcp.local/api/admin/extension/service 'Content-type:
application/vnd.vmware.admin.service+xml'
'Accept:application/*+xml;version=5.6' < ext-reg.xml
```

You will receive the following output:

```
<?xml version="1.0" encoding="UTF-8"?>
<vmext:Service xmlns:vmext="http://www.vmware.com/vcloud/extension/v1.5"
xmlns:vcloud="http://www.vmware.com/vcloud/v1.5" name="gcp-test"
id="urn:vcloud:externalService:f79f562a-8d7a-44d6-9bab-82ddf40589f7"
href="https://vcd.gcp.local/api/admin/extension/service/f79f562a-8d7a-44d6-9bab-82ddf40589f7"
type="application/vnd.vmware.admin.service+xml" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.vmware.com/vcloud/extension/v1.5
http://vcd.gcp.local/api/v1.5/schema/vmwextensions.xsd http://www.vmware.com/vcloud/v1.5
http://vcd.gcp.local/api/v1.5/schema/master.xsd">
    <vcloud:Link rel="remove" href="https://vcd.gcp.local/api/admin/extension/service/f79f562a-
8d7a-44d6-9bab-82ddf40589f7"/>
    <vcloud:Link rel="edit" href="https://vcd.gcp.local/api/admin/extension/service/f79f562a-
8d7a-44d6-9bab-82ddf40589f7" type="application/vnd.vmware.admin.service+xml"/>
    <vcloud:Link rel="rights" href="https://vcd.gcp.local/api/admin/extension/service/f79f562a-
8d7a-44d6-9bab-82ddf40589f7/rights" type="application/vnd.vmware.admin.rights+xml"/>
    <vcloud:Link rel="down:serviceLinks"
href="https://vcd.gcp.local/api/admin/extension/service/f79f562a-8d7a-44d6-9bab-
82ddf40589f7/links" type="application/vnd.vmware.vcloud.query.records+xml"/>
    <vcloud:Link rel="bundle:upload"
href="https://vcd.gcp.local/api/admin/extension/service/localizationbundles"
type="application/vnd.vmware.admin.bundleUploadParams+xml"/>
    <vcloud:Link rel="add" href="https://vcd.gcp.local/api/admin/extension/service/f79f562a-8d7a-
44d6-9bab-82ddf40589f7/links" type="application/vnd.vmware.admin.serviceLink+xml"/>
    <vcloud:Link rel="down:apiFilters"
href="https://vcd.gcp.local/api/admin/extension/service/f79f562a-8d7a-44d6-9bab-
82ddf40589f7/apifilters" type="application/vnd.vmware.vcloud.query.records+xml"/>
    <vcloud:Link rel="add" href="https://vcd.gcp.local/api/admin/extension/service/f79f562a-8d7a-
44d6-9bab-82ddf40589f7/apifilters" type="application/vnd.vmware.admin.apiFilter+xml"/>
```

```
    <vcloud:Link rel="add" href="https://vcd.gcp.local/api/admin/extension/service/f79f562a-8d7a-
44d6-9bab-82ddf40589f7/apidefinitions" type="application/vnd.vmware.admin.apiDefinition+xml"/>

    <vcloud:Link rel="down:apiDefinitions"
href="https://vcd.gcp.local/api/admin/extension/service/f79f562a-8d7a-44d6-9bab-
82ddf40589f7/apidefinitions" type="application/vnd.vmware.vcloud.query.records+xml"/>

    <vcloud:Link rel="add" href="https://vcd.gcp.local/api/admin/extension/service/f79f562a-8d7a-
44d6-9bab-82ddf40589f7/resourceclasses" type="application/vnd.vmware.admin.resourceClass+xml"/>

    <vcloud:Link rel="down:resourceClasses"
href="https://vcd.gcp.local/api/admin/extension/service/f79f562a-8d7a-44d6-9bab-
82ddf40589f7/resourceclasses" type="application/vnd.vmware.vcloud.query.records+xml"/>

    <vcloud:Link rel="authorization:check"
href="https://vcd.gcp.local/api/admin/extension/service/f79f562a-8d7a-44d6-9bab-
82ddf40589f7/authorizationcheck"
type="application/vnd.vmware.admin.authorizationCheckParams+xml"/>

    <vmext:Namespace>local.gcp.ticketing</vmext:Namespace>

    <vmext:Enabled>true</vmext:Enabled>

    <vmext:AuthorizationEnabled>false</vmext:AuthorizationEnabled>

    <vmext:RoutingKey>gcp-ticketing</vmext:RoutingKey>

    <vmext:Priority>0</vmext:Priority>

    <vmext:Exchange>vcdext</vmext:Exchange>

</vmext:Service>
```

To retrieve a list of all extensions you can execute:

```
# http --verify no --session=vcloud-gcp-admin --pretty colors GET
https://vcd.gcp.local/api/admin/extension/service/query
```

To test that the extension allows calling the ticketing method for an organization, execute the following command:

```
# http --verify no --session=vcloud-gcp-admin --pretty colors GET
https://vcd.gcp.local/api/org/<ORG_UUID>/ticketing
```

When this test command is executed, the call will hang for 10 seconds, and eventually return a 504 error message, “External service ‘gcp-ticketing’ failed to respond in the specified timeout (10 SECONDS).” This verifies that the new extension was registered successfully, however, nothing was configured yet to handle the message that was added to the message queue.

3.2 Handling Extension AMQP Messages

When the API receives an extension request, the vCloud Director REST service creates a message and sends it to the system AMQP service, specifying the exchange and routing key registered by the extension service. The extension service retrieves the message from a queue bound to the exchange it registered, processes the request, and returns a response to the common reply exchange.

To handle the AMQP messages, you must use a language that has a client to communicate with RabbitMQ. There are tutorials for most major programming languages on the RabbitMQ site at https://www.rabbitmq.com/tutorials/tutorial-one-go.html. This example uses Python with the Pika client.

To listen for incoming messages, you must bind to the correct Exchange and Queue (also referred to as RoutingKey). This was defined when you registered the extension (see Section 3.1, Register an Extension).

- Exchange: `vcdext`
- Queue: `gcp-ticketing`

When new messages are received, the following information is provided: channel, method, properties and body.

The properties contain three important pieces of information that are required to respond to the request:

- Headers – These are the AMQP headers, not the API headers. The key piece of information required is the replyToExchange.
- Correlation_id – vCloud Director sets a unique correlation_id in every message sent to the extension service, and the extension must set the same value in the corresponding response.
- Reply_to – What queue to respond on.

The body is made up of many parts, but, for basic integration, the following is required:

- Body – The body of the API request.
- Headers – The API request headers. It is important to note the incoming Accept header to respond back with the correct content type.
- Method – The method used to call the API (GET, POST, PUT, DELETE).
- Id – The unique ID assigned to the incoming message. This is used to reply to the message.

For more information about the body, see the *VMware vCloud API Programming Guide for Service Providers* at http://pubs.vmware.com/vcd-80/topic/com.vmware.vcloud.api.sp.doc_90/GUID-DD339C3C-4304-4524-BD00-A76E47A9C731.html.

3.2.1 Basic Example

This example generically handles a request and responds back using Python. The following is an example of the code used. Refer to the source code on GitHub to make sure you have the most current version of the code.

On a server that can connect to the RabbitMQ instance, run the `basic_receive.py` program script (https://github.com/johnnycuse/vcd-api-examples/blob/master/ticketing/basic_receive.py).

```python
#!/usr/bin/env python
import base64, json, pika

# RabbitMQ Connection Information
RABBIT_HOST = 'rabbitmqhost'
RABBIT_USER = 'vcdextuser'
RABBIT_PASSWORD = 'vcdextpass'

# Exchange and Queue we will subscribe to
RABBIT_EXCHANGE = 'vcdext'
RABBIT_ROUTINGKEY = 'gcp-ticketing'


# Connect to RabbitMQ
connection = pika.BlockingConnection(pika.ConnectionParameters(\
  host=RABBIT_HOST, credentials=pika.PlainCredentials(RABBIT_USER, RABBIT_PASSWORD)))

# Create a channel to subscribe to the incoming messages.
sub_channel = connection.channel()
sub_channel.exchange_declare(exchange=RABBIT_EXCHANGE, type='direct', durable=True)
sub_channel.queue_declare(queue=RABBIT_ROUTINGKEY)
sub_channel.queue_bind(exchange=RABBIT_EXCHANGE, queue=RABBIT_ROUTINGKEY)

print ' [*] Waiting for messages on exchange %s. To exit press CTRL+C' % RABBIT_EXCHANGE

# Create a channel for publishing messages back to the client.
pub_channel = connection.channel()

def callback(ch, method, properties, body):
    """
    Function for handleing all messages received on the RabbitMQ Exchange
    """
    print ' [!] Received a message!'
    body = json.loads(body)[0]

    # The response body that we will sent back to the client.
    rsp_body = "Got your message!"
    # Build the response message to return
    rsp_msg = {'id':body['id'],
               'headers':{'Content-Type':body['headers']['Accept'],
                          'Content-Length':len(rsp_body)},
               'statusCode':200,
               'body':base64.b64encode(rsp_body),
               'request':False}

    # vCD sets unique correlation_id in every message sent to extension and the extension must set.
    # the same value in the corresponding response.
    rsp_properties = pika.BasicProperties(correlation_id=properties.correlation_id)

    print "\t Sending response..."
    # We send our response to the Exchange and queue that were specified in the received properties.
    pub_channel.basic_publish(properties.headers['replyToExchange'],
                              properties.reply_to,
                              json.dumps(rsp_msg),
                              rsp_properties)

    print ' [X] message handled.'

# Bind to the the queue we will be listening on with a callback function.
sub_channel.basic_consume(callback,
                          queue=RABBIT_ROUTINGKEY,
                          no_ack=True)

# Start to continuously monitor the queue for messages.
sub_channel.start_consuming()
```

A call is then made against the new API ticketing endpoint:

```
# http --verify no --session=vcloud-gcp-admin --pretty colors GET
https://vcd.gcp.local/api/org/<ORG_UUID>/ticketing
```

The call and response is similar to the following:

```
user@demoBox:~/ticketing$ http --session=vcloud-gpc-admin --pretty colors GE
T https://vcd.gcp.local/api/org/9aee51e8-654e-49a8-8dab-3fdbf00a21ae/ticketi
ng --verify no
HTTP/1.1 200 OK
Date: Tue, 15 Sep 2015 17:58:12 GMT
X-VMWARE-VCLOUD-REQUEST-ID: ab2ec1d8-dcf0-45ec-adba-251b4a8714ba
x-vcloud-authorization: d5b8b8a57bf74a7aaba02df02d0bf826
Set-Cookie: vcloud-token=d5b8b8a57bf74a7aaba02df02d0bf826; Secure; Path=/
Content-Type: application/*+xml;version=5.6
Content-Length: 17

Got your message!
```

The following output from `basic_receive.py` indicates that the message has been properly handled:

```
user@demoBox:~/ticketing$ python basic_receive.py
 [*] Waiting for messages on exchange vcdext. To exit press CTRL+C
 [!] Received a message!
         Sending response...
 [X] message handled.
```

3.3 Full Ticketing Example

Now that you have seen that the message has been successfully handled, you can implement a full ticketing solution. A python script, `ticketing.py`, that is meant to simulate working with a ticketing system, is available at https://github.com/johnnycuse/vcd-api-examples/blob/master/ticketing/ticketing.py.

To explore the API, execute `python ticketing.py`:

```
user@demoBox:~/ticketing$ python ticketing.py
Starting ticketing...
```

The following is a list of newly exposed functionality with examples showing how to make the corresponding API extension call:

- List tickets for an org – `GET: /api/org/<ORG_UUID>/ticketing`

  ```
  http --session=vcloud-gcp-admin --pretty colors GET
  https://vcd.gcp.local/api/org/<ORG_UUID>/ticketing --verify no
  ```

- Create a ticket – `POST: /api/org/<ORG_UUID>/ticketing`

  ```
  http --session=vcloud-gcp-admin --pretty colors POST
  https://vcd.gcp.local/api/org/<ORG_UUID>/ticketing --verify no <
  ticket_create.xml
  ```

- Get ticket details – `GET: /api/org/<ORG_UUID>/ticketing/<Ticket_ID>`

```
http --session=vcloud-gcp-admin --pretty colors GET
https://vcd.gcp.local/api/org/<ORG_UUID>/ticketing/1000 --verify no
```

- Update a ticket – `PUT: /api/org/<ORG_UUID>/ticketing/<Ticket_ID>`

```
http --session=vcloud-gcp-admin --pretty colors PUT
https://vcd.gcp.local/api/org/<ORG_UUID>/ticketing/1000 --verify no <
ticket_update.xml
```

- Delete a ticket – `DELETE: /api/org/<ORG_UUID>/ticketing/<Ticket_ID>`

```
http --session=vcloud-gcp-admin --pretty colors DELETE
https://vcd.gcp.local/api/org/<ORG_UUID>/ticketing/1000 --verify no
```

Using these example, you can explore how the new API endpoints are exposed and implemented.

3.4 Adding API Links

While the new ticketing extension is registered, it does not show up in the link list. By using the ServiceLink object, you can add link elements to the representation of vCloud API objects. This can be done by passing the ServiceLinks with the original service extension registration, or it can be added later. In this case, they are added after the extension has been implemented.

When you view an organization:

```
# http --verify no --session=vcloud-gcp-admin --pretty colors GET
https://vcd.gcp.local/api/org/<ORG_UUID>
```

there is no link element returned providing the path to the newly defined service. To add the new links, you must retrieve the UUID of the registered service:

```
# http --session=vcloud-gcp-admin --pretty colors GET
https://vcd.gcp.local/api/admin/extension/service/query --verify no
```

This provides the list of all registered extensions. After finding the ticketing extension, you can get the details of the extension you added:

```
# http --verify no --session=vcloud-gcp-admin --pretty colors GET
https://vcd.gcp.local/api/admin/extension/service/4c3eb8af-38ba-4ff9-8189-
52329d601a58
```

One of the links that is returned allows you to add the ServiceLinks:

```
<vcloud:Link rel="add"
href="https://vcd.gcp.local/api/admin/extension/service/4c3eb8af-38ba-4ff9-
8189-52329d601a58/links"
type="application/vnd.vmware.admin.serviceLink+xml"/>
```

POST this ServiceLink to this location. To build a ServiceLink, there are four required attributes:

- `LinkHref` – The value of the `href` attribute of this service Link. This can be any URI, and can include the variables `{baseUri}` and `{resourceId}`. When constructing the `href` value of the link, vCloud Director replaces `{baseUri}` with the vCloud Director REST API base URL, and replaces `{resourceId}` with the UUID portion of the `id` attribute value of the resource in which the link is inserted. The value must be less than or equal to 255 characters in length.
- `MimeType` – The value, specified as a MIME content type, of the type attribute of the link. The value must be less than or equal to 255 characters in length.
- `Rel` – The value of the `rel` attribute of the link. This value must be less than or equal to 255 characters in length.
- `ResourceType` – The object type, specified as a MIME content type, of the object in which the link appears. Must be less than or equal to 255 characters in length.

Example XML (https://github.com/johnnycuse/vcd-api-examples/blob/master/ticketing/ext-addLink.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<vmext:ServiceLink
   xmlns:vmext="http://www.vmware.com/vcloud/extension/v1.5"
   xmlns:vcloud="http://www.vmware.com/vcloud/v1.5"
   type="application/vnd.vmware.admin.serviceLink+xml">
   <vmext:LinkHref>{baseUri}org/{resourceId}/ticketing</vmext:LinkHref>
   <vmext:MimeType>application/vnd.example.vcd.tickets+xml</vmext:MimeType>
   <vmext:Rel>listTickets</vmext:Rel>

<vmext:ResourceType>application/vnd.vmware.vcloud.org+xml</vmext:ResourceType
>
</vmext:ServiceLink>
```

Add the link to the extension:

```
# http --verify no --session=vcloud-gcp-admin --pretty colors POST
https://vcd.gcp.local/api/admin/extension/service/4c3eb8af-38ba-4ff9-8189-
52329d601a58/links 'Content-type:
application/vnd.vmware.admin.servicelink+xml' < ext-addLink.xml
```

After the link is added, call the `org` object to verify that it has been added:

```
# http --verify no --session=vcloud-gcp-admin --pretty colors GET
https://vcd.gcp.local/api/org/<ORG_UUID>
```

The Org XML that is returned now includes the newly added ticketing link element:

```
<Link rel="listTickets" href="https://vcd.gcp.local/api/org/9aee51e8-654e-
49a8-8dab-3fdbf00a21ae/ticketing"
type="application/vnd.example.vcd.tickets+xml"/>
```

Summary

Now that you have created an extension, you might be wondering how you control access to your new API. By default, all extension requests must be authenticated through the vCloud API, but all users are implicitly granted permission to perform all actions. Through the use of the vCloud Director API Authorization Framework, you can create granular ACL-based control.

Also consider adding the schema files to provide documentation about your new API using Create or Update an Extension Service API Definition.