ClearPass Extension BlackBerry Unified Endpoint Manager

ClearPass

TechNote

Change Log

Version	Date	Modified By	Comments
0.1	May 2017	Danny Jump	Initial Draft Version for internal review
0.2	June 2017	Robert Filer	Wordsmith and minor edits
1.0	June 2017	Danny Jump	First Published Version

Copyright

Open Source Code

This product includes code licensed under the GNU General Public License, the GNU Lesser General Public License, and/or certain other open source licenses. A complete machine-readable copy of the source code corresponding to such code is available upon request. This offer is valid to anyone in receipt of this information and shall expire three years following the date of the final distribution of this product version by Hewlett- Packard Company. To obtain such source code, send a check or money order in the amount of US $10.00 to:

Hewlett-Packard Company
Attn: General Counsel
3000 Hanover Street
Palo Alto, CA 94304
USA

Please specify the product and version for which you are requesting source code. You may also request a copy of this source code free of charge at HPE-Aruba-gplquery@hpe.com.

Contents

Figures

Introduction and Overview

This TechNote describes how to use ClearPass Policy Manager and the ClearPass Unified Endpoint Management Extension to interoperate with the BlackBerry on-premise or cloud-deployed Unified Endpoint Management Extension [UEM]. UEM is the new device management platform from BlackBerry, previously known as BlackBerry Enterprise Server or BES where they have streamlined the management of mobile [Android, BlackBerry, iOS, Windows Phone] and desktop operating system such as macOS and Windows 10. UEM is the new platform that also enables the consolidation of Good technologies and extends the BES platform to MDM, MAM and MCM.

Software Requirements

The minimum software version required is ClearPass 6.6.1. At the time of writing, ClearPass 6.6.5 is the latest available and recommended release. Any subsequent ClearPass software release will support this integration. ClearPass runs on either hardware appliances with pre-installed software, or as a Virtual Machine on the hypervisors shown below. Hypervisors that run on a client computer such as VMware Player are not supported.

- VMware ESXi 5.0, 5.1, 5.5, 6.0, 6.5 or higher
- Microsoft Hyper-V Server 2012 or 2016 R2
- Hyper-V on Microsoft Windows Server 2012 or 2016 R2
- KVM on CentOS 6.6, 6.7, or 6.8.
- BlackBerry UEM 12.6 MR1

NOTE

The API's utilized in the UEM 12.6 MR1 existed in UEM 12.5. At this time, only version 12.6 MR1 has been tested and certified.

ClearPass Installation and Deployment Guide

This document assumes your ClearPass environment is already configured and operational. If assistance with basic deployment is required, refer to the following deployment guide:
http://www.arubanetworks.com/techdocs/ClearPass/Aruba_DeployGd_HTML/Default.htm

a Hewlett Packard Enterprise company

www.arubanetworks.com
3333 Scott Blvd
Santa Clara, CA 95054
Phone: 1-800-WIFI-LAN (+800-943-4526)
Fax 408.227.4550

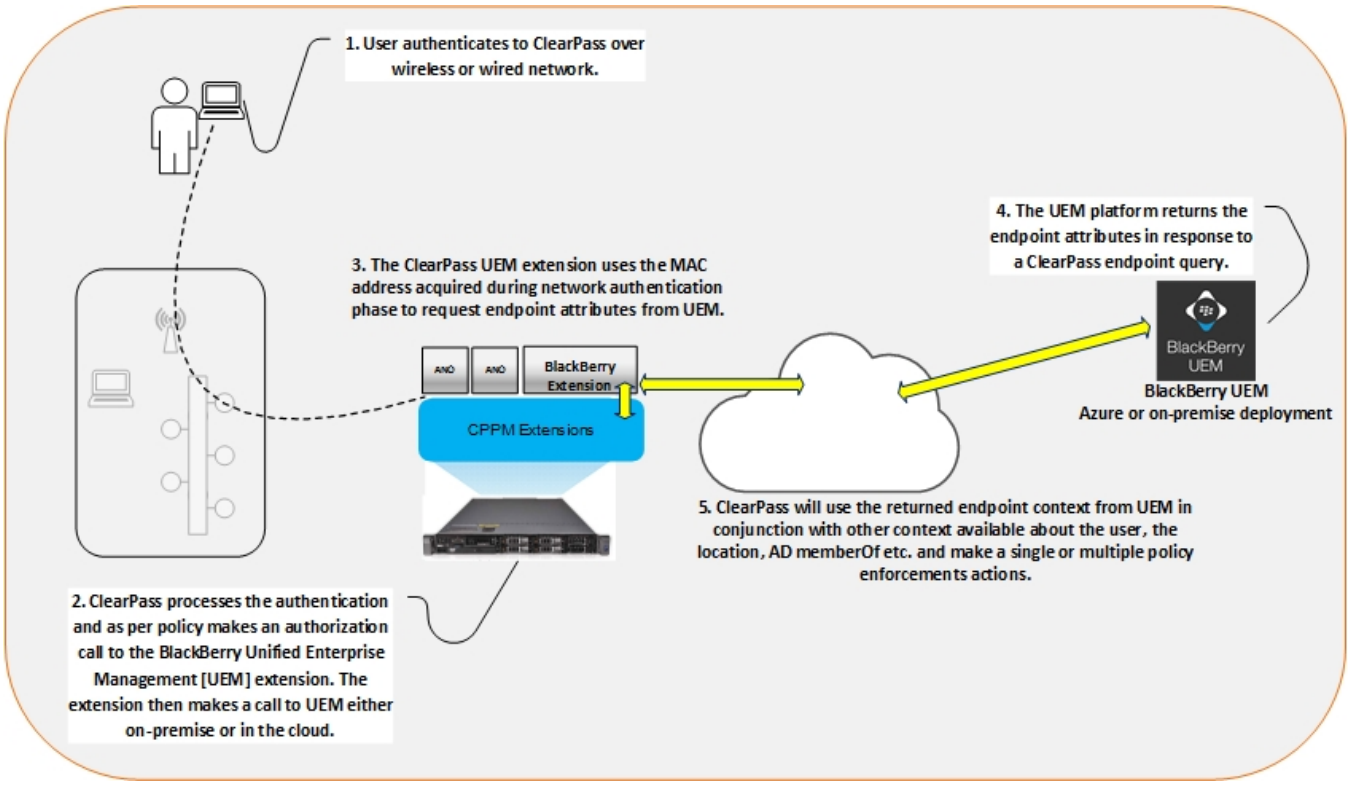
ClearPass Extensions

The integration between ClearPass Policy Manager and UEM is driven through a ClearPass capability known as Extensions, a sub-component of the ClearPass Exchange Integration framework. ClearPass Extensions are micro-services running on top of the base ClearPass platform. These micro-services enable Aruba to deliver new features outside of the main software release cycle and facilitate a faster time to market for specific features and integrations. Configuration and control of ClearPass Extensions is accomplished through the ClearPass REST APIs which are covered later in this document.

Pictorial View of the Integration

Below is a pictorial view of the integration components between ClearPass Policy Manager and BlackBerry UEM. In the below diagram, BlackBerry UEM is shown running in the Cloud. UEM is also supported as an on-premise deployment.

Figure 1: Pictorial view of the ClearPass and BlackBerry Solution



UEM Extension Installation

Installation of the UEM ClearPass Extension is performed via the REST API interface as it is for any Extension. ClearPass REST APIs were initially introduced in CPPM 6.5 and have continued to be enhanced in subsequent releases. Access to the APIs is through the following URL: **https://<ClearPass_IP>/api-docs**. Admin credentials are required to access the API interface.

The APIs to support ClearPass Extensions were initially released in ClearPass version 6.6.

Several components, and multiple steps, are required to complete the ClearPass configuration:

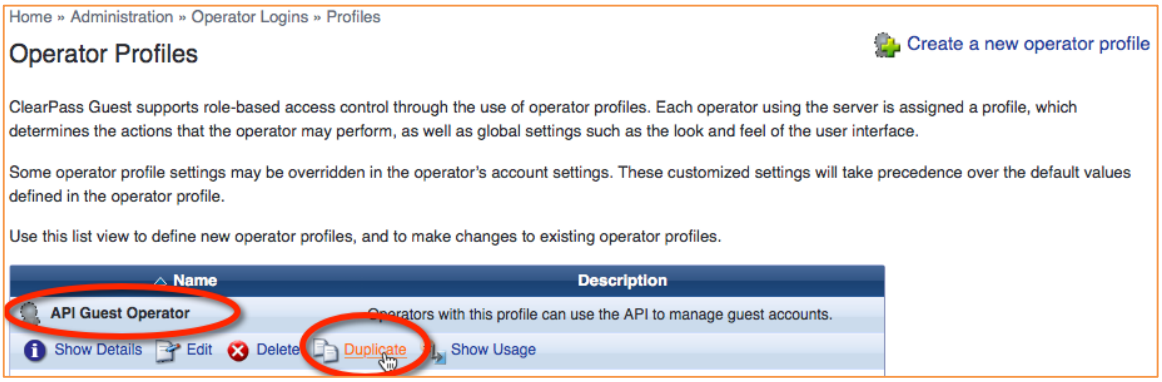
- Extension Installation
- Extension Configuration
- Policy Manager HTTP Authorization Source
- Policy Manager Enforcement Policy and Profiles

The ClearPass extension installation is shown below. The steps to create the Operator Profile and install the actual extension are the same for all Extensions; the only unique change will be the specific Extension ID.

Create an Extension API Operator Profile

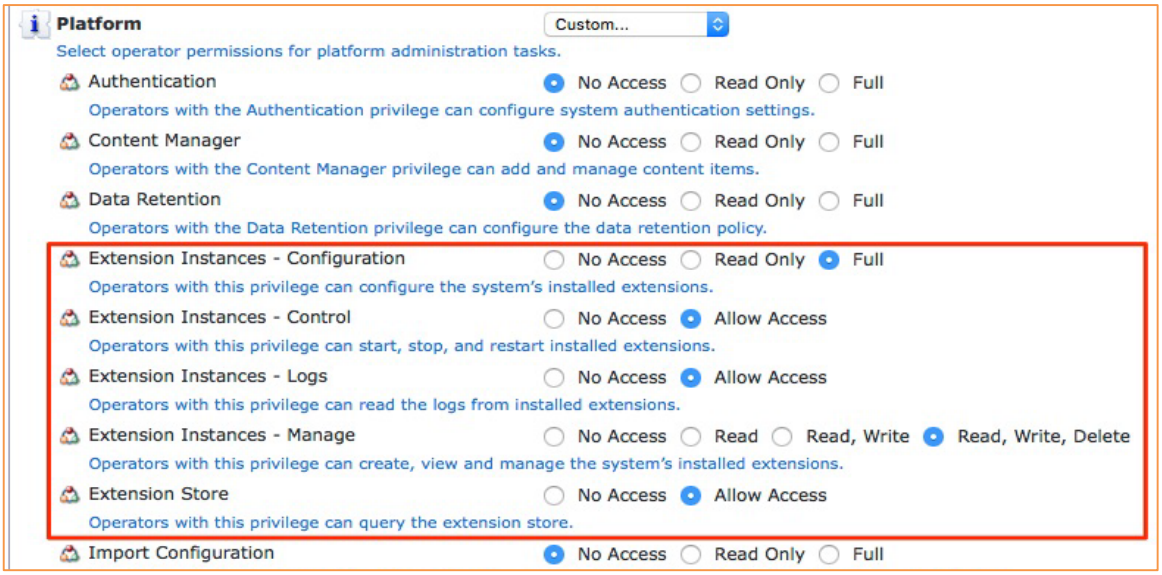
Before setting up the API access, configure an **Operator Profile** that will be associated with the configured API client. This new **Operator Profile** will be used in the next section when creating the **API Client**. Login to ClearPass Guest and go to **Administration > Operator Logins > Profiles**. Next, **Click** on **API Guest Operator** and select **Duplicate**.

Figure 2: Duplicate API Guest Operator profile



ClearPass will copy the profile and name it **API Guest Operator (2)**. **Click Edit** on the new profile and rename it to be **API Extension Profile**. For the **Platform** privilege, change **No Access** to **Custom** and then set all the **Extension** options as shown below (furthest option to the right). Scroll to the bottom of the page and **Click Save**.

Figure 3: Modifying Operator Profile permissions for extensions



In the next step, this profile will be used as the **Operator Profile** when generating the API Client.

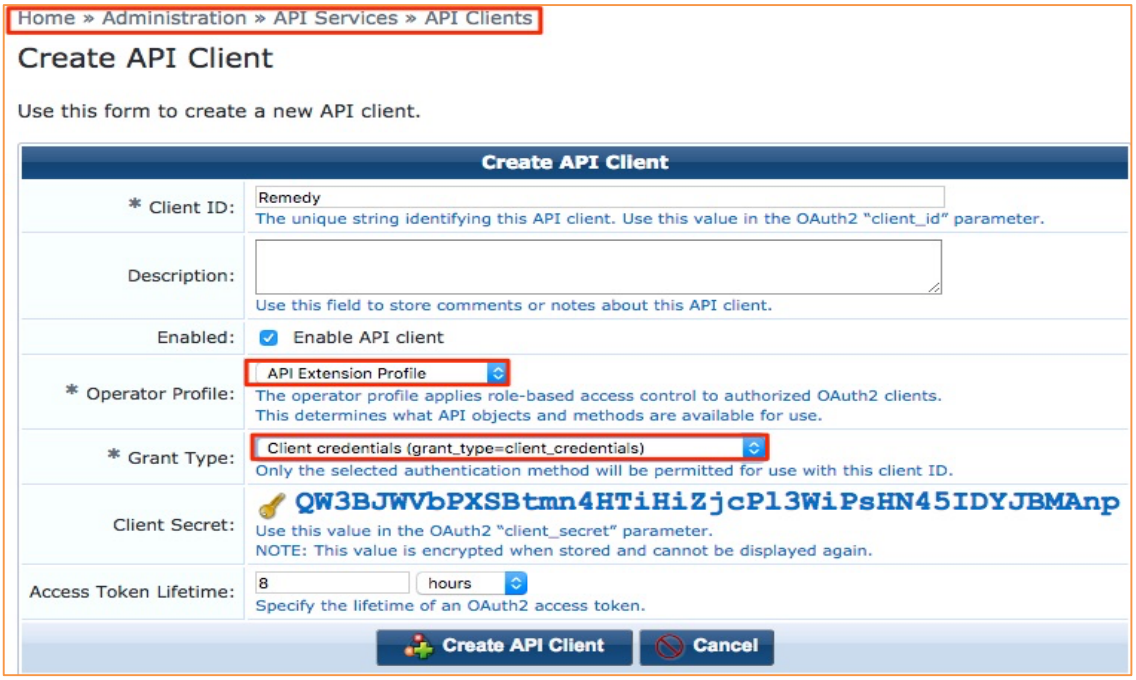
Create an API Client

The first step in installing and enabling the ClearPass Extension is to create an API Client. An API Client provides authentication and authorization for the REST APIs. Authentication is performed using OAuth2, which is an authorization framework that enables applications to obtain limited access to data over a HTTP service without sharing their private credentials. Log into ClearPass Guest at **https://<Your-ClearPass-Server>/guest**.

Navigate to **Guest > Administration > API Services > API Clients** and create an API Client by entering the following:

1. **Client ID**: Free choice
2. **Operator Profile**: API Extension Profile [just created]
3. **Grant Type**: Client credentials

Figure 4: Creating an API Client

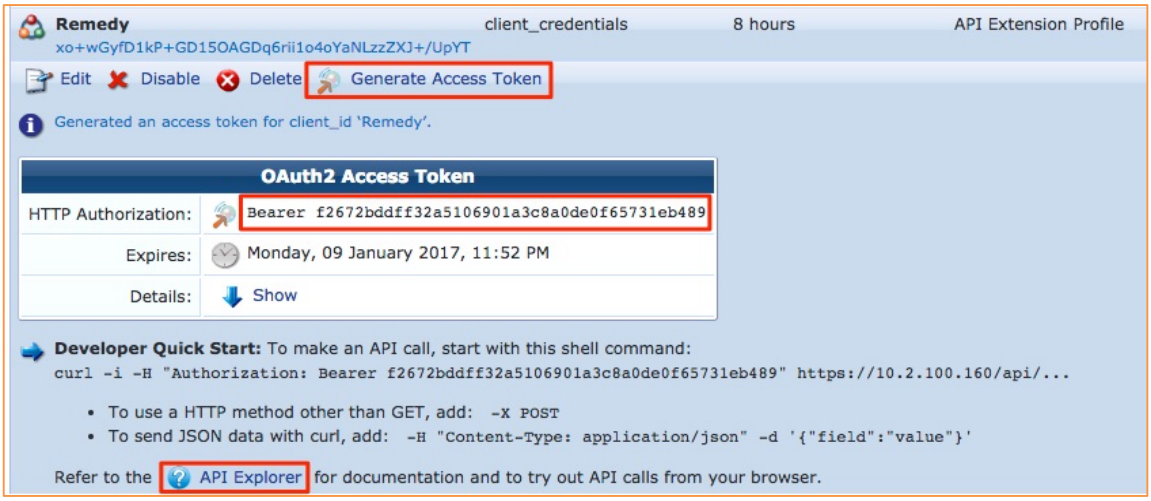


Click on **Create API Client** to save and create the API Client.

Generate an Access Token

Click Generate Access Token and then launch the **API Explorer**, as highlighted below at the bottom of the image.

Figure 5: Generate the Access Token

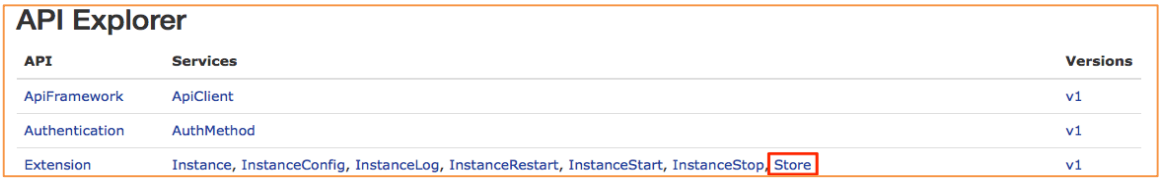


This will pre-populate the Authorization header in the API Explorer and permit commands to be run directly from the ClearPass UI.

Go to the Extension APIs

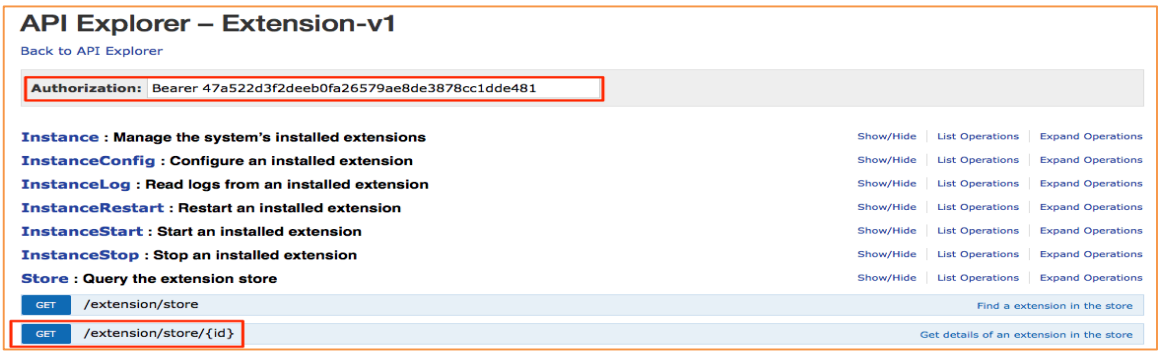
Next check whether ClearPass can communicate with the Extension store. This is an important step to ensure that Proxy-Servers and/or Firewalls are not blocking connectivity to the ClearPass Extension Store. **Click** on **Extension > Store**.

Figure 6: API Explorer UI



Under Store, **Click GET /extension/store/{id}**

Figure 7: Accessing the Store API

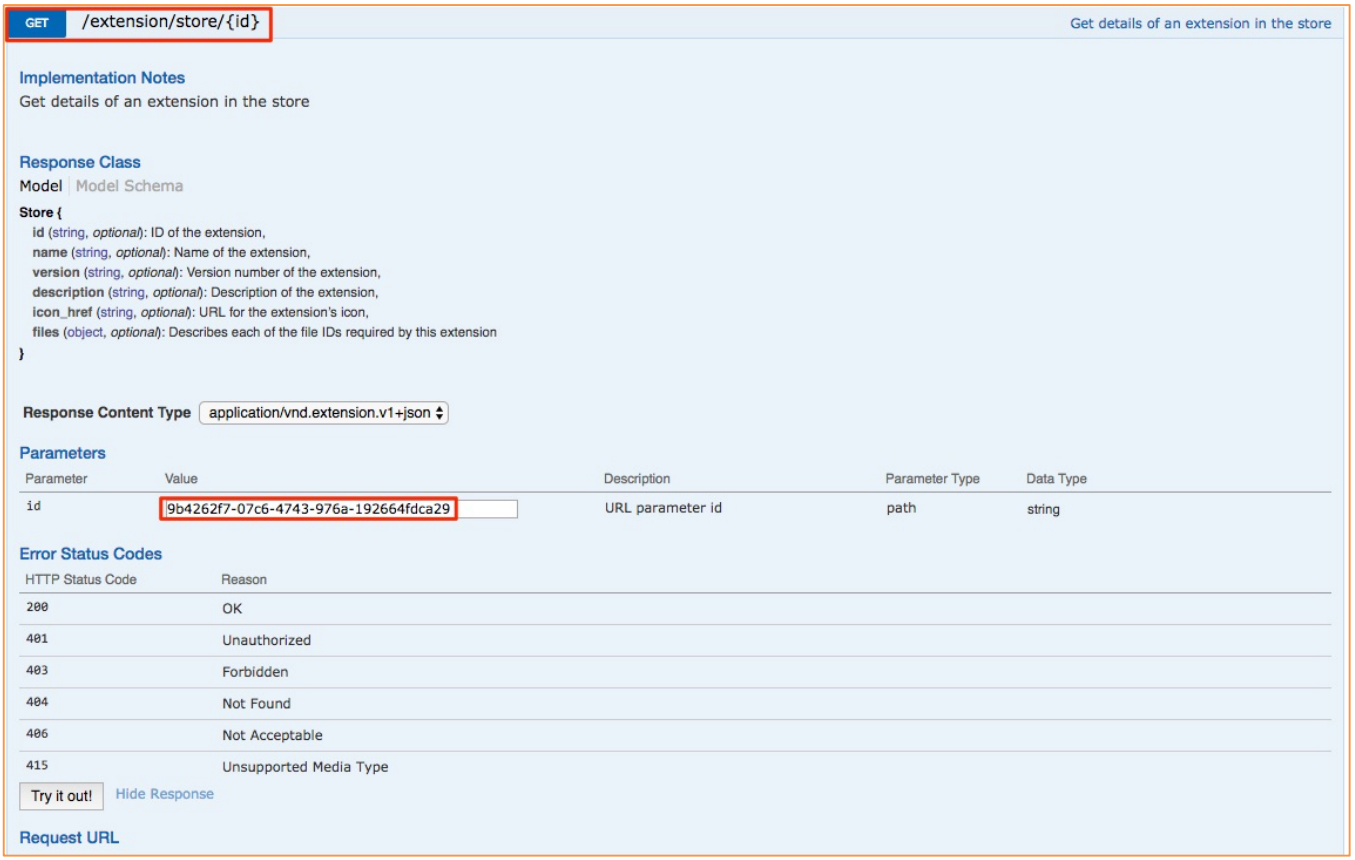


Notice that the Authorization header is populated. This is populated from creating the token in the previous step. Next expand the **GET /extension/store/{id}** and paste in the extension's store ID.

NOTE

As an example, let's use the store ID for the BlackBerry UEM Extension. It's a fixed value of **9b4262f7-07c6-4743-976a-192664fdca29. Click** on '**Try it out'!**

Figure 8: Checking the Extension store for a particular Extension ID

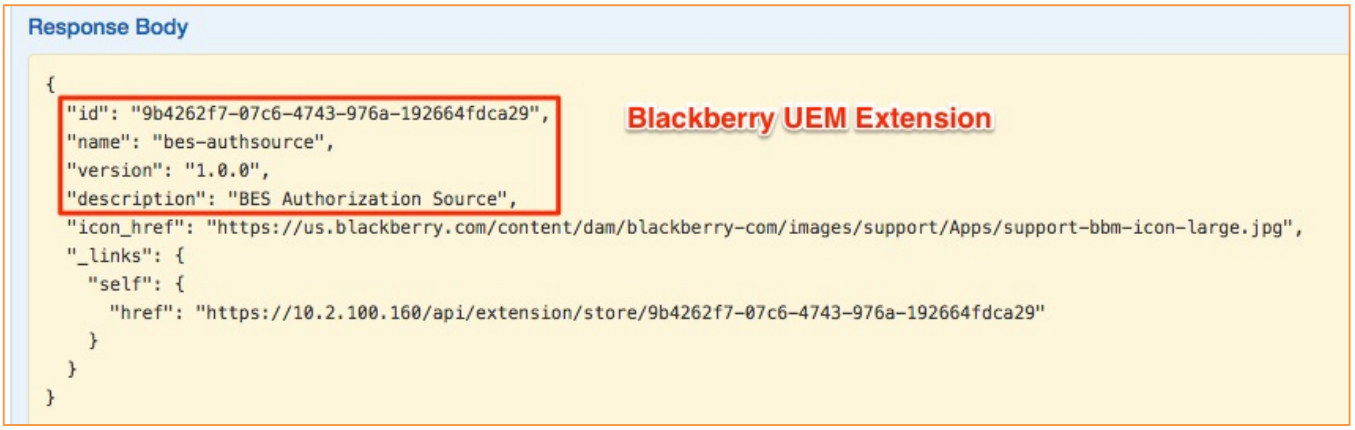


NOTE

Remember that the store ID **9b4262f7-07c6-4743-976a-192664fdca29** is unique to this version of the BlackBerry UEM Extension. The store ID will change as new versions of the Extension are published.

This will return details for the BlackBerry UEM Extension. This provides assurance that the correct authority is configured to allow access to the Store, and that this is the correct ID for the extension being installed.

Figure 9: Details of the Extension

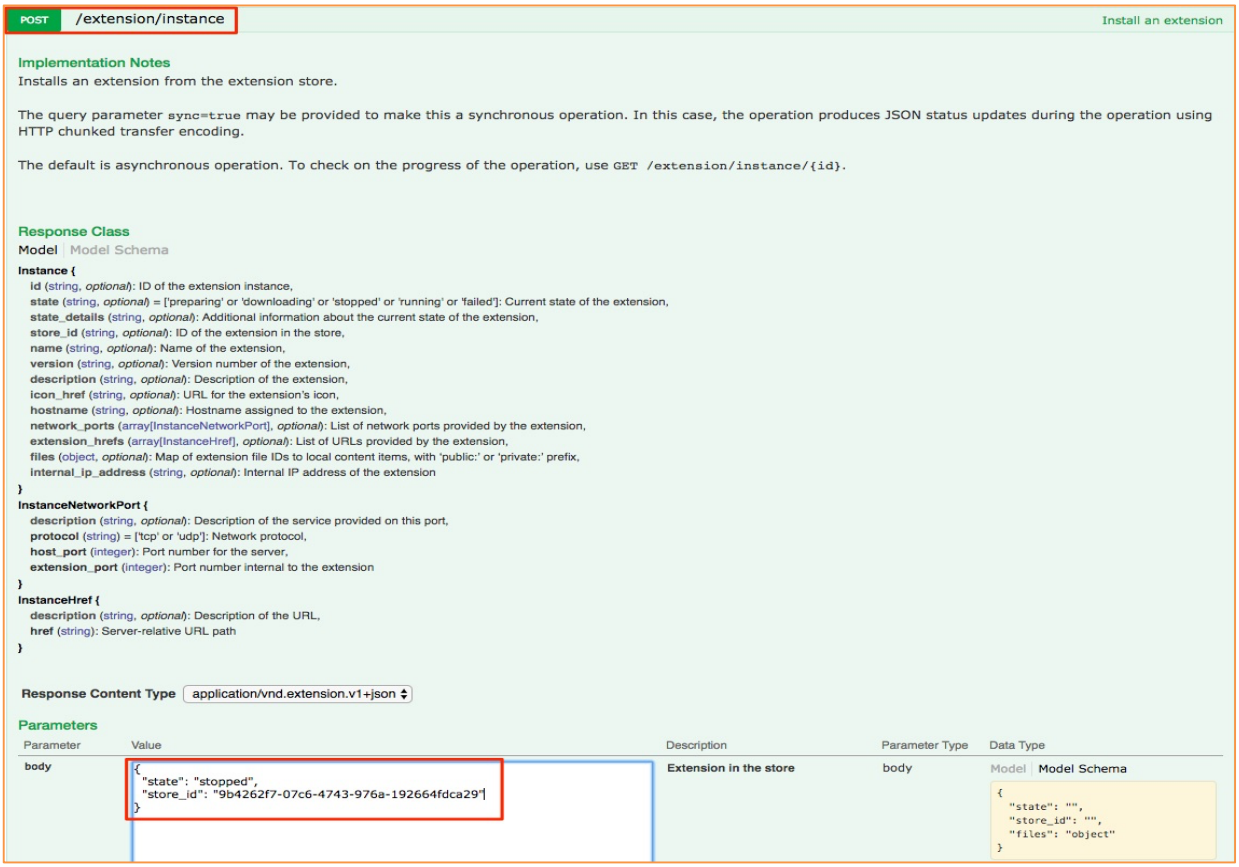


Install the Extension

After checking for extension visibility, the next step is to install it. Under **Instance > POST /extension/instance**, paste in the body as shown below:

{"state": "stopped", "store_id":"9b4262f7-07c6-4743-976a-192664fdca29"} and then **Click** on '**Try it out!'**

Figure 10: Installing the Extension directly from the Extension store



NOTE

This returns an **ID** (Note: this is different from the store ID. It will be referred to as the run-time ID). Make a copy of this ID as it will be required later.

The below shows the state of the Extension as **preparing**. This indicates the extension is in the process of being downloaded. A typical installation should take just a few seconds to complete.

Figure 11: Extension Status

Response Body


```
{
  "id": "c7edd79e-a9e5-4982-a02d-577454a089ac",
  "state": "preparing",
  "state_details": "",
  "store_id": "9b4262f7-07c6-4743-976a-192664fdca29",
  "name": "",
  "version": "",
  "description": "",
  "icon_href": "",
  "_links": {
    "self": {
      "href": "https://10.2.100.160/api/extension/instance/c7edd79e-a9e5-4982-a02d-577454a089ac"
    }
  }
}
```


From the above, you can see the ID is **c7edd79e-a9e5-4982-a02d-577454a089ac**.This ID (the run-time ID) will be used to query and configure the extension.

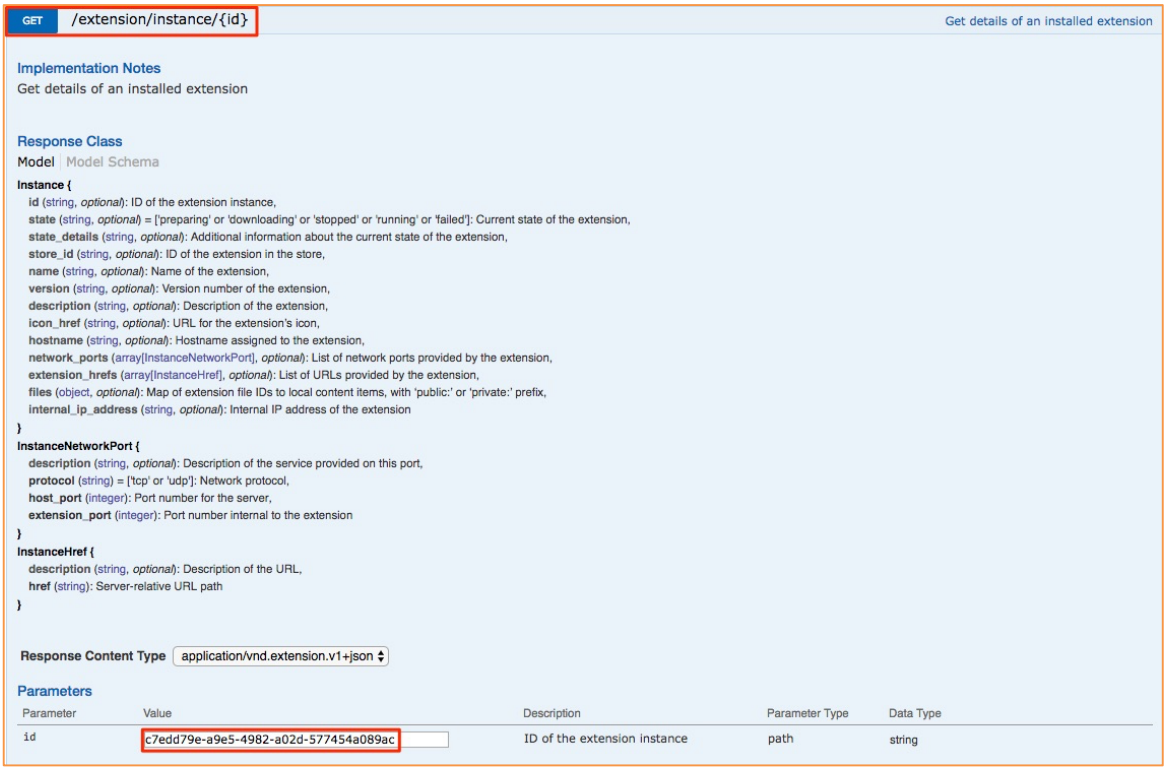
WARNING

Your run-time ID will differ from the one used here. It is generated per installation and is unique to each and every installation.

Query Extension after Installation

Under **Instance > GET /extension/instance/{id}** paste in the ID from the previous step: **c7edd79e-a9e5-4982-a02d-577454a089ac** and **Click 'Try it out!'**. Remember, this ID will be unique for every installation.

Figure 12: Checking Extension Installation Progress



GET /extension/instance/{id} — Get details of an installed extension

Implementation Notes

Get details of an installed extension

Response Class

Model | Model Schema

```
Instance {
  id (string, optional): ID of the extension instance,
  state (string, optional) = ['preparing' or 'downloading' or 'stopped' or 'running' or 'failed']: Current state of the extension,
  state_details (string, optional): Additional information about the current state of the extension,
  store_id (string, optional): ID of the extension in the store,
  name (string, optional): Name of the extension,
  version (string, optional): Version number of the extension,
  description (string, optional): Description of the extension,
  icon_href (string, optional): URL for the extension's icon,
  hostname (string, optional): Hostname assigned to the extension,
  network_ports (array[InstanceNetworkPort], optional): List of network ports provided by the extension,
  extension_hrefs (array[InstanceHref], optional): List of URLs provided by the extension,
  files (object, optional): Map of extension file IDs to local content items, with 'public:' or 'private:' prefix,
  internal_ip_address (string, optional): Internal IP address of the extension
}
InstanceNetworkPort {
  description (string, optional): Description of the service provided on this port,
  protocol (string) = ['tcp' or 'udp']: Network protocol,
  host_port (integer): Port number for the server,
  extension_port (integer): Port number internal to the extension
}
InstanceHref {
  description (string, optional): Description of the URL,
  href (string): Server-relative URL path
}
```

Response Content Type: application/vnd.extension.v1+json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	c7edd79e-a9e5-4982-a02d-577454a089ac	ID of the extension instance	path	string

Within the Body Response from this **GET** observe the following:

Figure 13: Response to check on progress of Extension installation


```
{
  "id": "c7edd79e-a9e5-4982-a02d-577454a089ac",
  "state": "stopped",
  "state_details": "Created",
  "store_id": "9b4262f7-07c6-4743-976a-192664fdca29",
  "name": "bes-authsource",
  "version": "1.0.0",
  "description": "BES Authorization Source",
  "icon_href": "https://us.blackberry.com/content/dam/blackberry-com/images/support/Apps/support-bbm-icon-large.jpg",
  "hostname": "82e927f1bdac",
  "network_ports": [],
  "extension_hrefs": [],
  "files": [],
  "_links": {
    "self": {
      "href": "https://10.2.100.160/api/extension/instance/c7edd79e-a9e5-4982-a02d-577454a089ac"
    }
```


The details of the extension will be displayed (could be **downloading**, etc.). Eventually the state will change to either **stopped** or **failed**. In this example, it is clear the that the installation is **created** and **stopped**.

UEM Extension Configuration

To complete the configuration of the BlackBerry UEM extension, there are a number of parameters which need to be collected and supplied to the Extension. Start with looking at the default configuration.

Below is an example of how to retrieve the current configuration of the extension. Under **InstanceConfig > GET /extension/instance/{id}/config**, paste in the run-time ID **c7edd79e-a9e5-4982-a02d-577454a089ac** and **Click 'Try it out!'** to return a copy of the current Extension configuration.

Figure 14: Get Extension Configuration

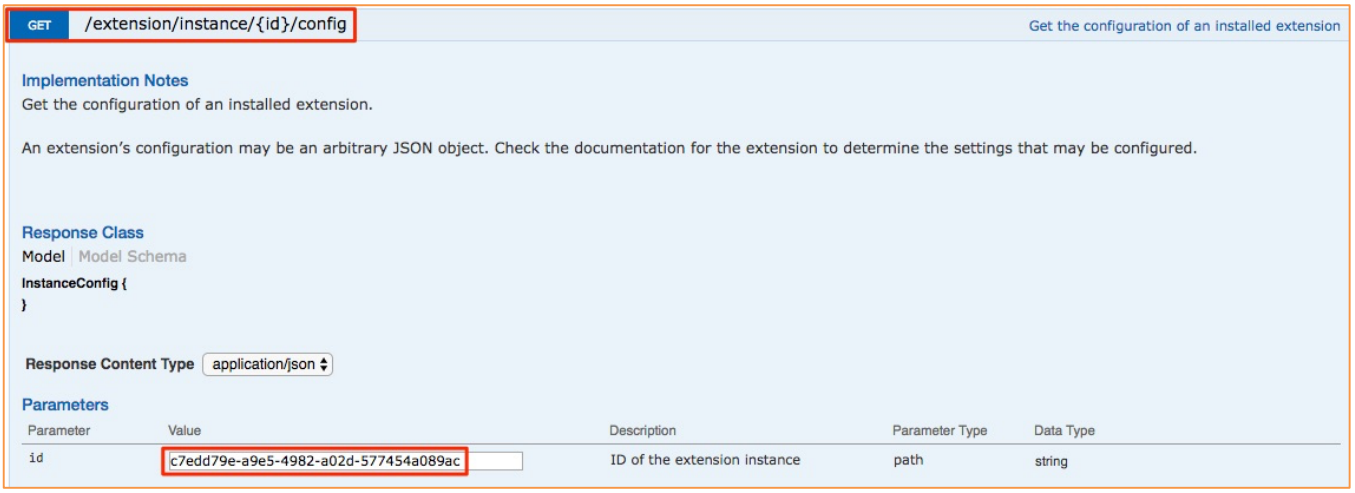


Figure 15: Response to a request for the current Extension configuration

```
{
  "logLevel": "INFO",
  "verifySSLCerts": true,
  "besEndpoint": "https://127.0.0.1:8095",
  "tenantExternalId": "SRP_ID",
  "provider": "LOCAL",
  "domain": "",
  "username": "admin.user1",
  "password": "admin.password1"
}
```

The above example shows the default shipping configuration of the BlackBerry UEM extension. This will need to be modified as mentioned above to match your BlackBerry UEM deployment.

Collecting Information to Configure the ClearPass UEM Extension

Installation, Deployment and Configuration of BlackBerry UEM is beyond the scope of this document, but to get the basics of a UEM Server running you will have already applied for and received your SRP-ID. BlackBerry has online resources for the deployment and configuration here: https://help.blackberry.com/en/blackberry-uem/12.6/

Based upon the default configuration, a number of attributes are required. The **logLevel** and **verifySSLCerts** should be left at their default values unless advised by TAC. This leaves the following:

- **besEndpoint**
- **username**
- **password**
- **provider**
- **tenantExternalId**

Of these remaining attributes, **besEndpoint** should be set to the FQDN or IP address of the BES UEM server.

The **username** and **password** reflect the credentials the Extension will use to communicate with the BlackBerry UEM Server.

The **provider** attribute refers to the authentication source of the above credential being used. This can be one of three values, **LOCAL, LDAP** or **AD**. If using **AD** then you need to update the **domain** attribute shown above in the default configuration with your appropriate setting.

If using **LDAP** setting of the **domain** attribute it's optional. If using **LOCAL** then "domain" is ignored.

Creating Roles in UEM to restrict communication

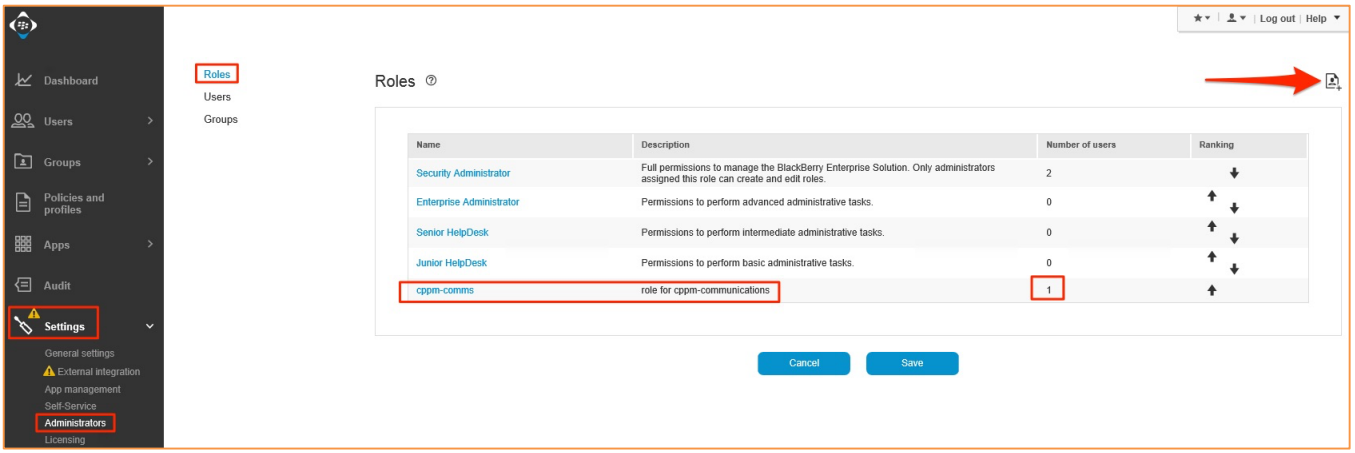
The account created and configure and used to communicate between the ClearPass Extension is a UEM application account. As discussed above this account can be a LOCAL, AD or LDAP account. If you use a LOCAL UEM account then the account needs minimum access rights as shown below.

When signed into the UEM console as a Security Administrator create a role to limit the capabilities of the account used to communicate with UEM.

Go to **Settings -> Administrators -> Roles** [click on the RHS Icon] to create a new role

An example of our role is shown below. It shows one single user assigned to the role-group.

Figure 16: UEM Role Summary

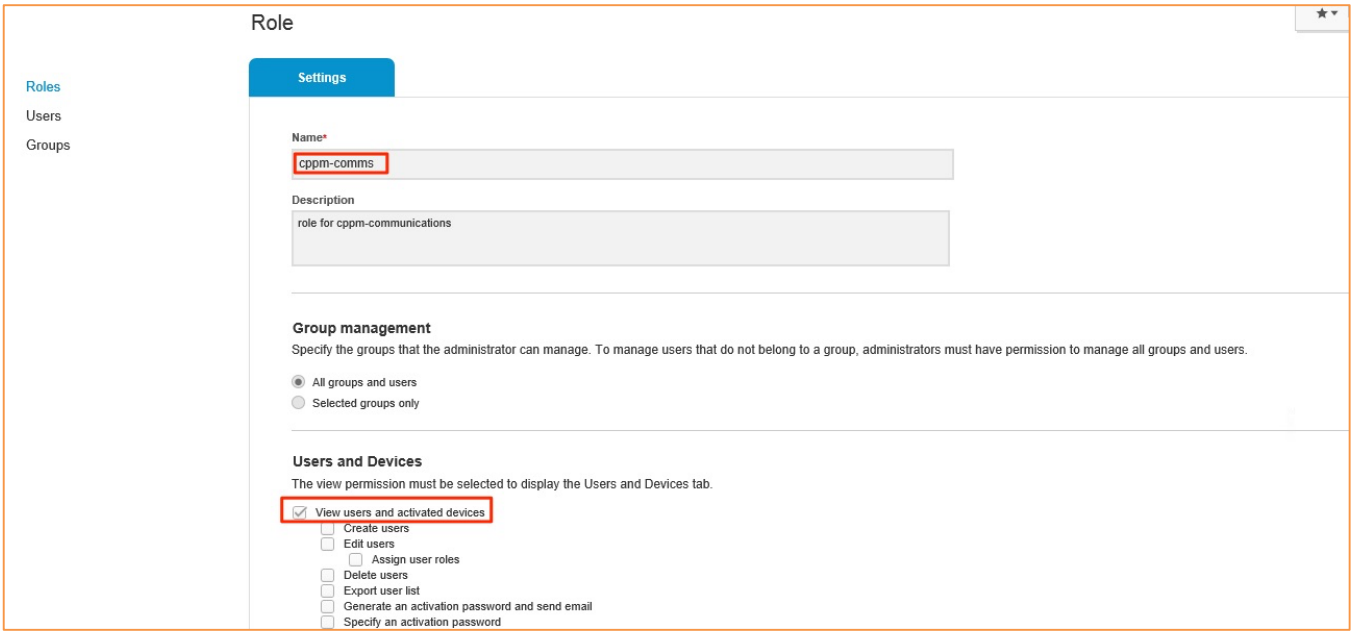


WARNING

This TechNote shows the use of a LOCAL account configured inside of the UEM application, not to be confused with a Windows LOCAL account.

When creating the role, set the minimum required permissions.

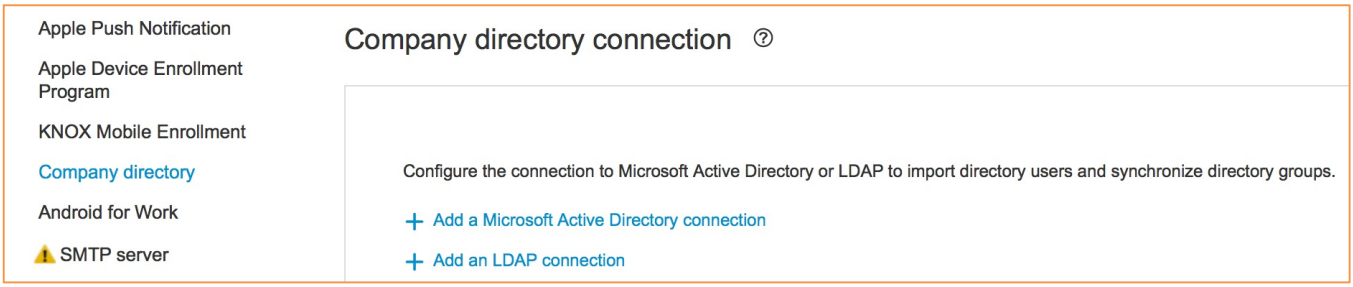
Figure 17: Creating the Admin Group with restricted permissions



One the role is created, assign this to your user. This is the user that will configured inside the ClearPass Extension and will be sued to authenticated communication between the ClearPass UEM Extension and the BlackBerry UEM Server.

If you need to integrate BlackBerry UEM with the underling Windows server/domain then under **Settings -> External Integration -> Company Directory** you can integrate UEM with either Microsoft AD or LDAP.

Figure 18: BlackBerry UEM Directory Connection Configuration



The remaining attribute, **tenantExternalId** is used to uniquely identify the deployed UEM Server instance. In BlackBerry speak it's called the Server Routing Protocol ID or SRP-ID for short. This is a value supplied directly by BlackBerry as you contract with them to use the BlackBerry MDM framework and services.

Once you have collected/generated all of the necessary attributes above, you need to update the Extension configuration as shown in the following steps.

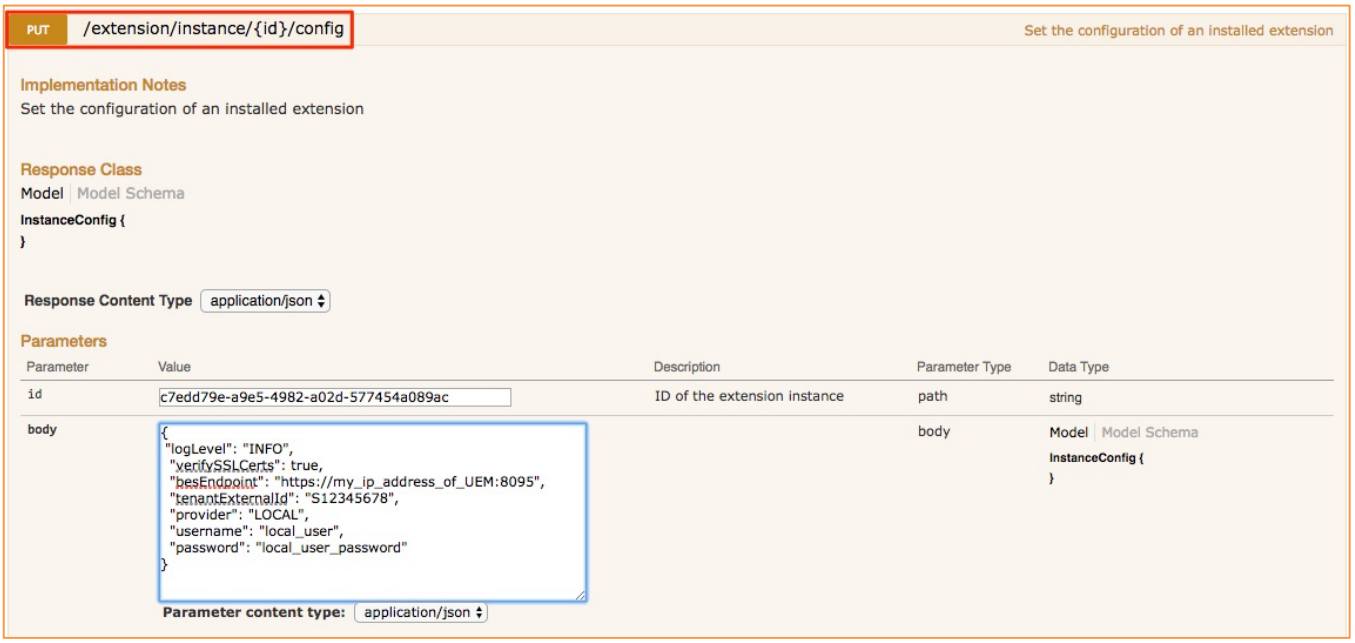
Configure the Extension

Under **InstanceConfig > PUT /extension/instance/{id}/config** paste in the run-time ID **c7edd79e-a9e5-4982-a02d-577454a089ac,** copy and paste the configuration collected in the previous steps to the PUT method and **Click 'Try it out!'**. Remember this ID is unique to the example in this document, yours will differ.

NOTE

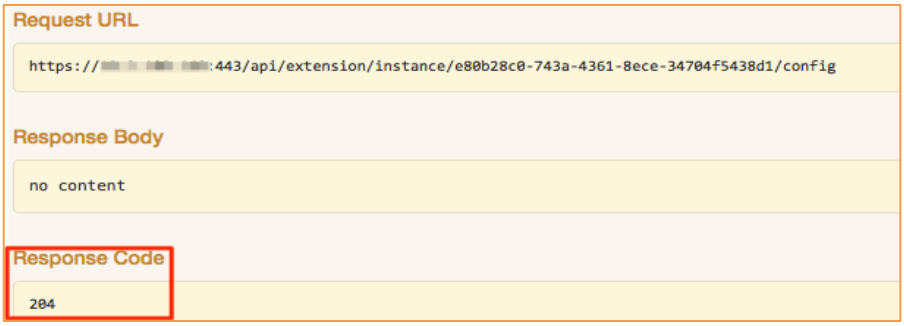
If changing the configuration, you should restart the extension via **InstanceRestart > POST /extension/instance/{id}/restart**

Figure 19: Setting the configuration in the Extension



Below is the Response to the PUT. A successful result is indicated by response code of 204.

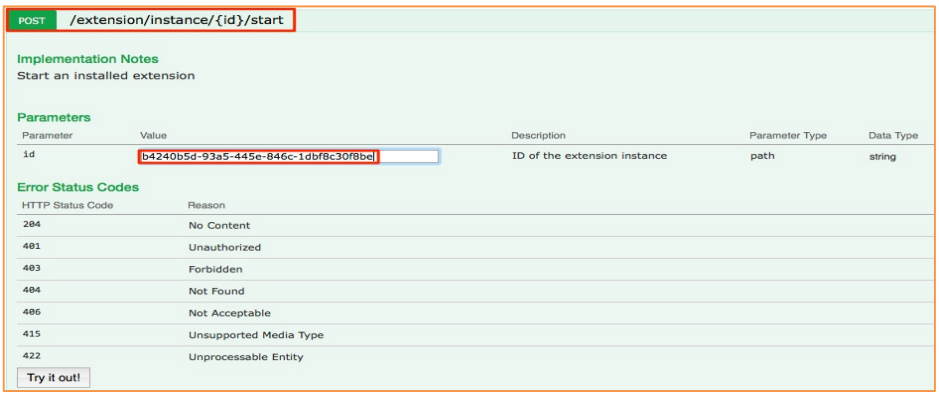
Figure 20: HTTP 204 response to the configuration PUT



Starting the Extension

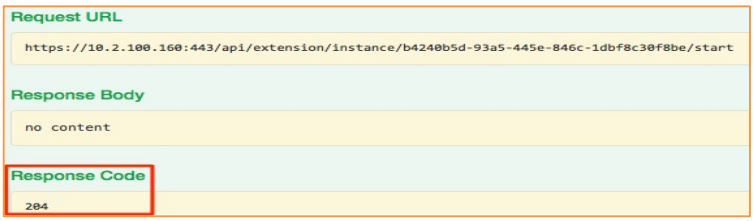
Under **InstanceStart > POST /extension/instance/{id}/start** paste in the extension ID and **Click** on **'Try it out!'**.

Figure 21: Starting the extension



A successful response is indicated by a 204 result as shown below.

Figure 22: Expected HTTP response to InstanceStart



Verify the Extension is Running

Under **Instance > GET /extension/instance/{id}** copy and paste the run-time ID and **Click 'Try it out!'**. The state of the extension should now be "running". An example of the HTTP response is shown below:

Figure 23: Detailed information on the running extension


```
{
  "id": "c7edd79e-a9e5-4982-a02d-577454a089ac",
  "state": "running",
  "state_details": "Started 12 seconds ago",
  "store_id": "9b4262f7-07c6-4743-976a-192664fdca29",
  "name": "bes-authsource",
  "version": "1.0.0",
  "description": "BES Authorization Source",
  "icon_href": "https://us.blackberry.com/content/dam/blackberry-com/images/support/Apps/support-bbm-icon-large.jpg",
  "hostname": "82e927f1bdac",
  "network_ports": [],
  "extension_hrefs": [],
  "files": [],
  "internal_ip_address": "172.17.0.12",
  "_links": {
    "self": {
      "href": "https://10.2.100.160/api/extension/instance/c7edd79e-a9e5-4982-a02d-577454a089ac"
    }
  }
}
```

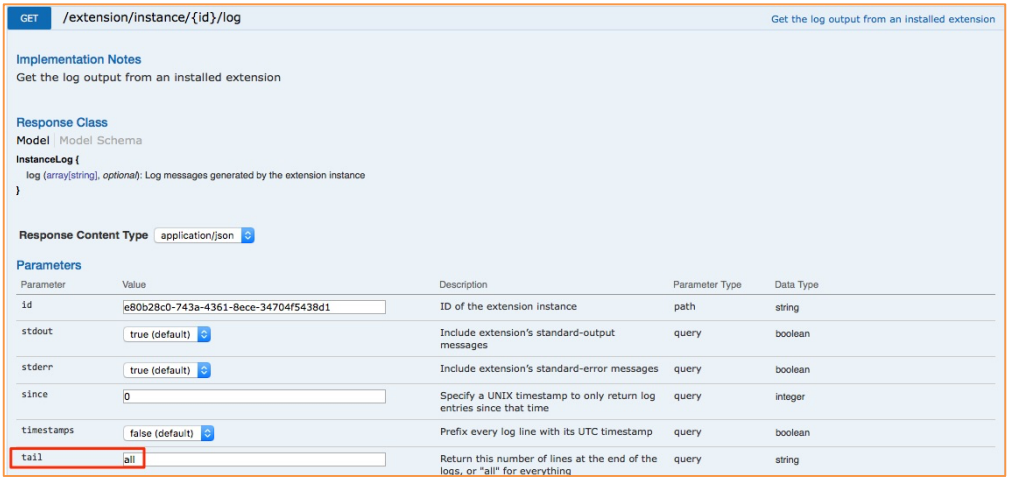

NOTE

The "**internal_ip_address**" of the extension will be set by the extension service. This will be used for configuring the authorization source later in Policy Manager.

Troubleshooting the Extension

Under **InstanceLog > GET /extension/instance/{id}/log**, paste in the run-time ID. Enter a value for "tail", e.g. 100 will show the last 100 lines of output and then **Click** on **'Try it out!'**. Note that other settings are applicable when getting logs, e.g. timestamps.

Figure 24: Getting Debug Logs from the extension.



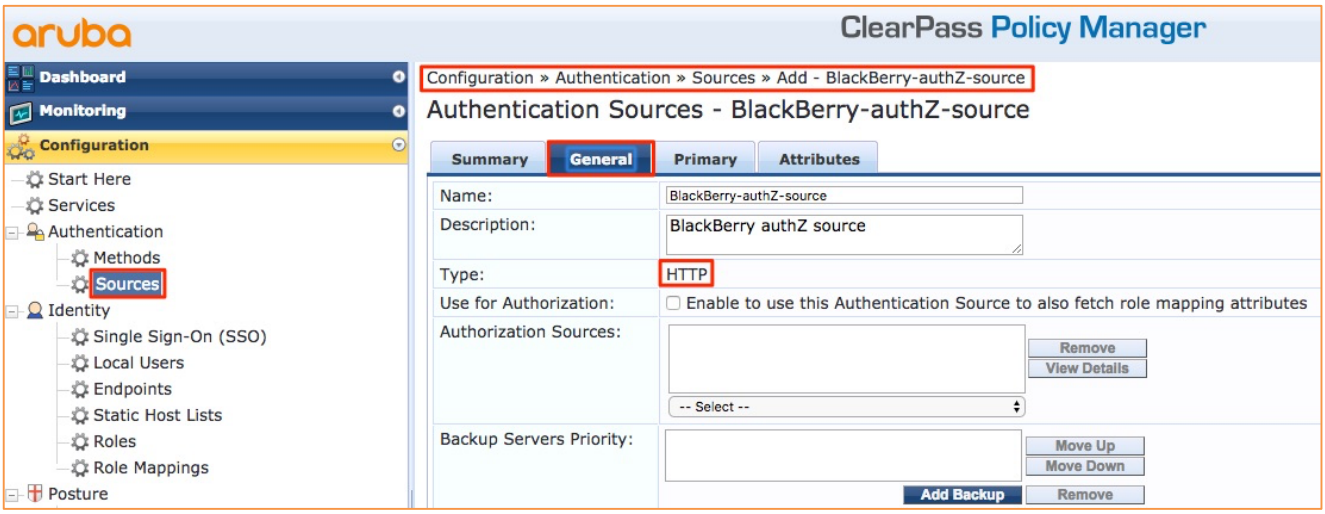
Configuring ClearPass Policy Manager

To complete the configuration, configure an authorization source within ClearPass. With BlackBerry UEM as an authorization source, ClearPass can check with UEM to see if the device is enrolled, managed, and in a valid compliance state before allowing it to connect. A common use case for ClearPass is to check the version of the installed operating system on the endpoint as the basis for applying specific access policy, if a known endpoint OS has recently been exposed to a OS vulnerability. Another popular use case is to use the ownership attribute to differentiate between a Corporate or Privately {BYOD} owned device. Finally, using context to know if the device is encrypted might be utilized to restrict access to financial applications for a user with an unencrypted device. These and/or other contextual attributes can be used to evaluate an endpoint at the time of network authentication.

Add HTTP Authorization Source

The first step is to add the authorization source. Under **Configuration > Authentication > Sources**, click **Add**.

Figure 25: Adding an HTTP authorization source

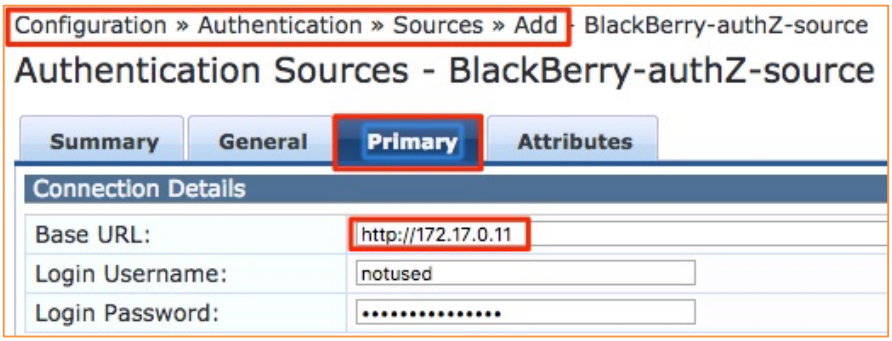


Click on **Next**. This will advance to the Primary Tab. Provide the connection details as mandated.

NOTE

The Base URL IP address is what you captured in Figure 21 above.

Figure 26: Adding HTTP authorization source credentials and Base URL



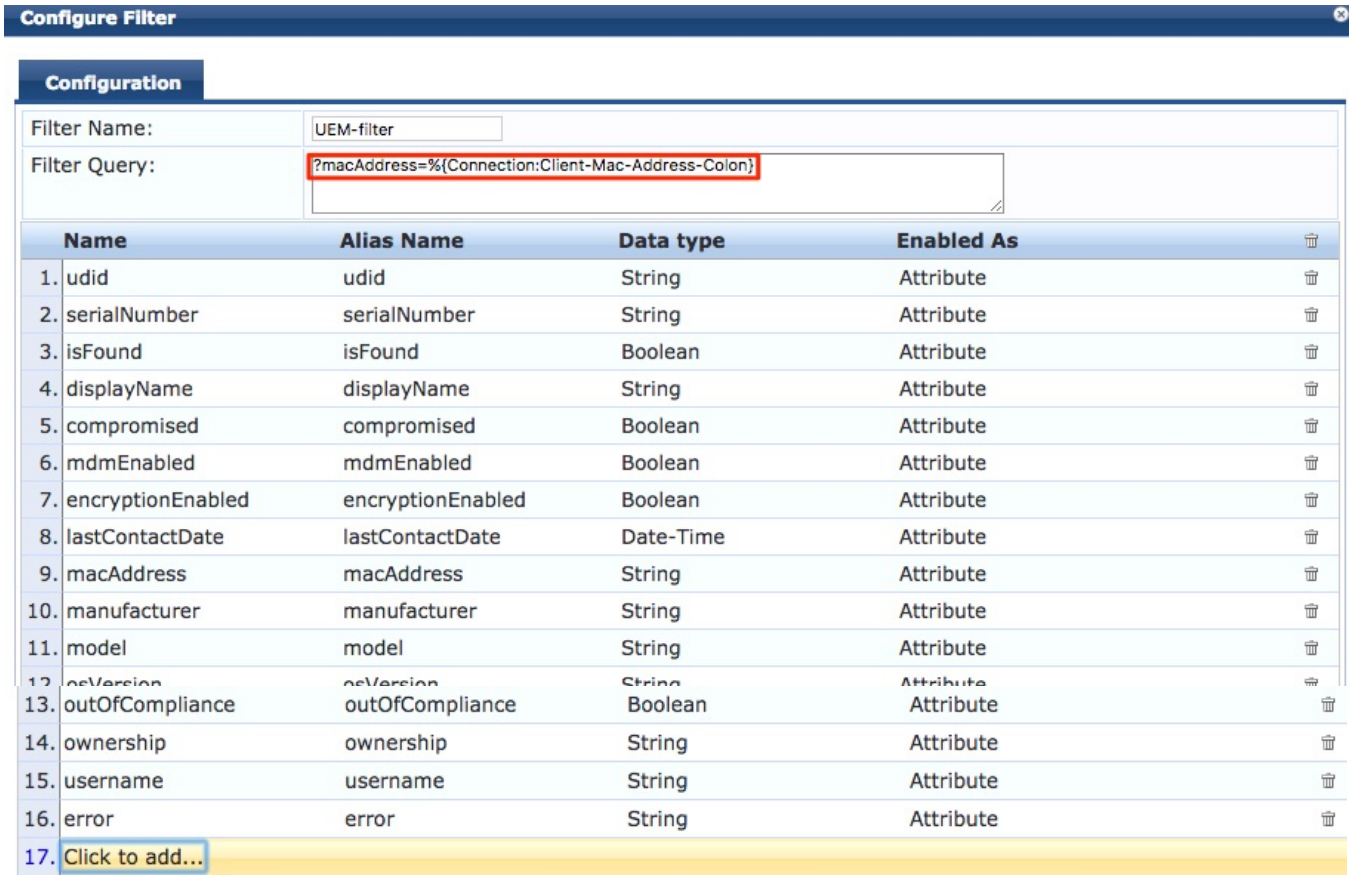
It's mandated that a Login Username and Password be entered, but they are not used and can be set to anything you wish.

Click on **Next**. This will advance you to the Attributes Tab where you need to provide the authorization attributes. **Click** on '**Add More Filters**'. Provide a Name for the filter and then a Filter Query. It's extremely important that the Filter Query is defined correctly. This is the query string that is sent to the BlackBerry UEM Extension asking for context about the endpoint. The query is indexed off the MAC Address of the authenticating endpoint. For ease in copying, the Filter Query is provided here.

?macAddress=%{Connection:Client-Mac-Address-Colon}

Next build out the definitions of the attributes that will be returned from the Filter Query. These attributes will subsequently be used within ClearPass policy evaluation and ultimately the enforcement policy applied. Below is a list of all the available fields. The list of fields you require could very likely just be a subset of that shown below. Only define what you need to expose in your enforcement policy.

Figure 27: Adding HTTP authorization source query string and returned field definitions



Once the HTTP authorization source is defined you can use the returned attributes in your policy processing. Several options on how to use the results from the authorization query in an enforcement policy are shown below.

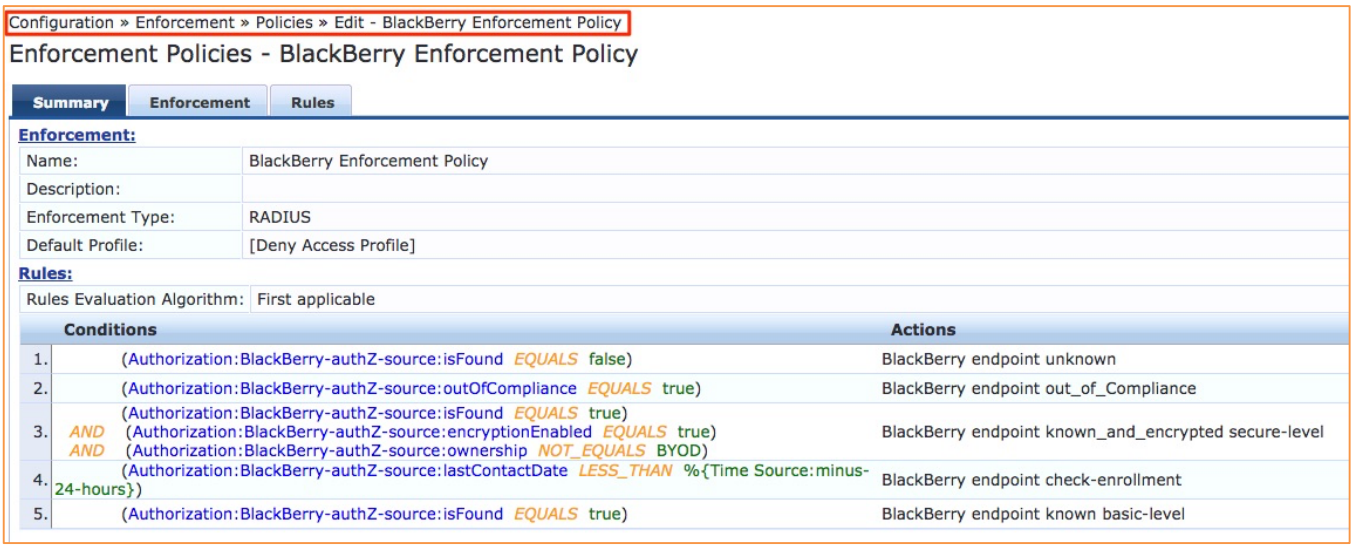
A copy of the above authorization source in XML format is in Appendix A.

Using data from UEM in a ClearPass Enforcement Policy

Multiple use cases exist for how the data that is returned from UEM can be used in your policy enforcement. In the example below, multiple checks are performed. Your enforcement policy will likely differ as required.

1. The first check is to ensure this is a known device and thus under management. This rule is looking for a boolean response of false, meaning this is **not** a managed device and treated accordingly.
2. If the device is not in compliance then apply an out_of_Compliance policy.
3. If the device is Corporately issued and managed, also ensure the device is encrypted. If all the above checks pass then in this example apply a privileged access policy.
4. Check the device has checked in to the BlackBerry UEM management platform in the last 24-hours. If not, then accordingly the health/posture of the endpoint can't be determined.
5. As a final check, as long as the device is known, thus managed, set a basic-level of access.

Figure 28: Example of an Enforcement Policy utilizing attributes returned from BlackBerry UEM



Configuration » Enforcement » Policies » Edit - BlackBerry Enforcement Policy

Enforcement Policies - BlackBerry Enforcement Policy

Summary | Enforcement | Rules

Enforcement:

Name:	BlackBerry Enforcement Policy
Description:	
Enforcement Type:	RADIUS
Default Profile:	[Deny Access Profile]

Rules:

Rules Evaluation Algorithm:	First applicable

	Conditions	Actions
1.	(Authorization:BlackBerry-authZ-source:isFound *EQUALS* false)	BlackBerry endpoint unknown
2.	(Authorization:BlackBerry-authZ-source:outOfCompliance *EQUALS* true)	BlackBerry endpoint out_of_Compliance
3.	(Authorization:BlackBerry-authZ-source:isFound *EQUALS* true) *AND* (Authorization:BlackBerry-authZ-source:encryptionEnabled *EQUALS* true) *AND* (Authorization:BlackBerry-authZ-source:ownership *NOT_EQUALS* BYOD)	BlackBerry endpoint known_and_encrypted secure-level
4.	(Authorization:BlackBerry-authZ-source:lastContactDate *LESS_THAN* %{Time Source:minus-24-hours})	BlackBerry endpoint check-enrollment
5.	(Authorization:BlackBerry-authZ-source:isFound *EQUALS* true)	BlackBerry endpoint known basic-level

Different companies will have different views on enforcement policies and how to grant or deny access. The key take-away here is that you are using the authorization attributes received from BlackBerry UEM to drive the policy engine into making and taking different enforcement actions for the device as they authenticate on the network.

Appendix A – BlackBerry UEM Authentication Source


```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<TipsContents xmlns="http://www.avendasys.com/tipsapiDefs/1.0">
  <TipsHeader exportTime="Wed May 31 15:52:27 PDT 2017" version="6.6"/>
  <AuthSources>
    <AuthSource description="BlackBerry authZ source" name="BlackBerry-authZ-source" isAuthorizationSource="false" type="HTTP">
      <NVPair value="http://172.17.0.11" name="base_url"/>
      <NVPair value="notused" name="username"/>
      <NVPair value="" name="password"/>
      <Filters>
        <Filter paramValues="" filterQuery="?macAddress=%{Connection:Client-Mac-Address-Colon}" filterName="UEM-filter">
          <Attributes>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" aliasName="udid" attrName="udid"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" aliasName="serialNumber" attrName="serialNumber"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="Boolean" aliasName="isFound" attrName="isFound"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" aliasName="displayName" attrName="displayName"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="Boolean" aliasName="compromised" attrName="compromised"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="Boolean" aliasName="mdmEnabled" attrName="mdmEnabled"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="Boolean" aliasName="encryptionEnabled" attrName="encryptionEnabled"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="Date-Time" aliasName="lastContactDate" attrName="lastContactDate"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" aliasName="macAddress" attrName="macAddress"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" aliasName="manufacturer" attrName="manufacturer"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" aliasName="model" attrName="model"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" aliasName="osVersion" attrName="osVersion"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="Boolean" aliasName="outOfCompliance" attrName="outOfCompliance"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" aliasName="ownership" attrName="ownership"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" aliasName="username" attrName="username"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" aliasName="error" attrName="error"/>
          </Attributes>
        </Filter>
      </Filters>
    </AuthSource>
  </AuthSources>
</TipsContents>
```

Appendix B – Additional Diagnostics / Support

Extension Service

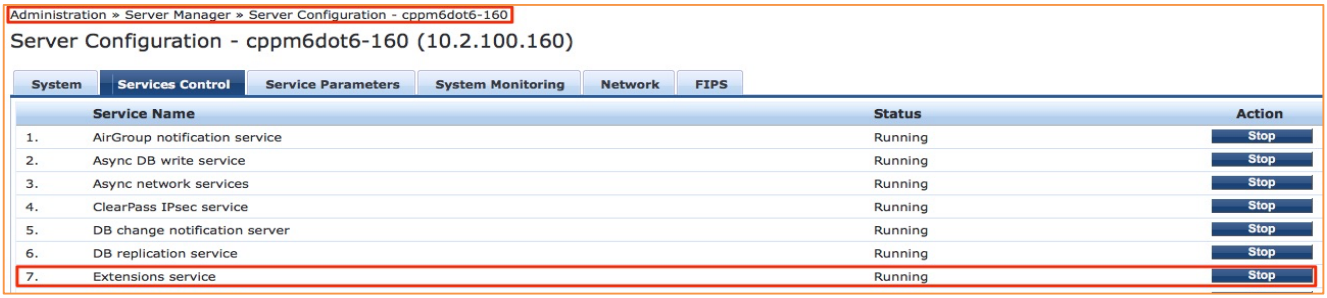
ClearPass Extensions are supported by a new system service that was added in ClearPass 6.6. This service should be running by default.

WARNING

Restarting this service will affect all deployed and running extensions.

To check on the state and make changes to the service navigate to **Administration > Server Manager > Server Configuration [select your ClearPass node] > Service Control**. You can also start/stop the extension service from here. By default, this service is automatically started.

Figure 29: Checking on extension service and how to start/stop the service



Extension Logs/Debugging

If you have a need to access the logs from inside the extension, you can turn on log collection from the API Explorer. Referencing the configuration previously used, adjust the "**logLevel**" to "**DEBUG**". Post this using the API Explorer as shown below.

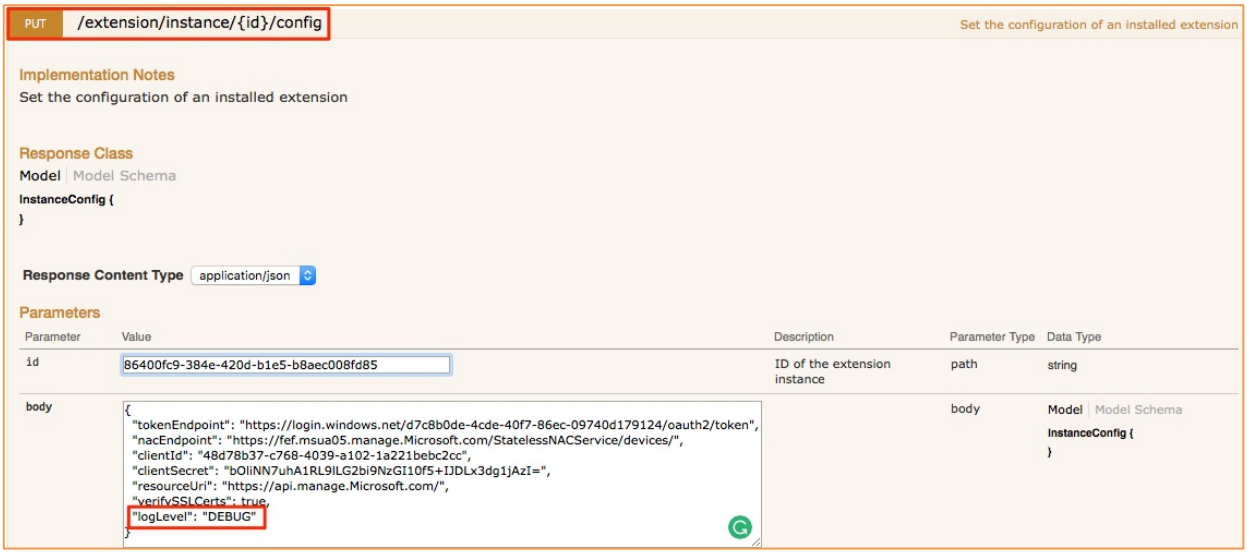
NOTE

Remember, any change to an extension configuration mandates a restart of the extension.

Figure 30: Changing logLevel to DEBUG

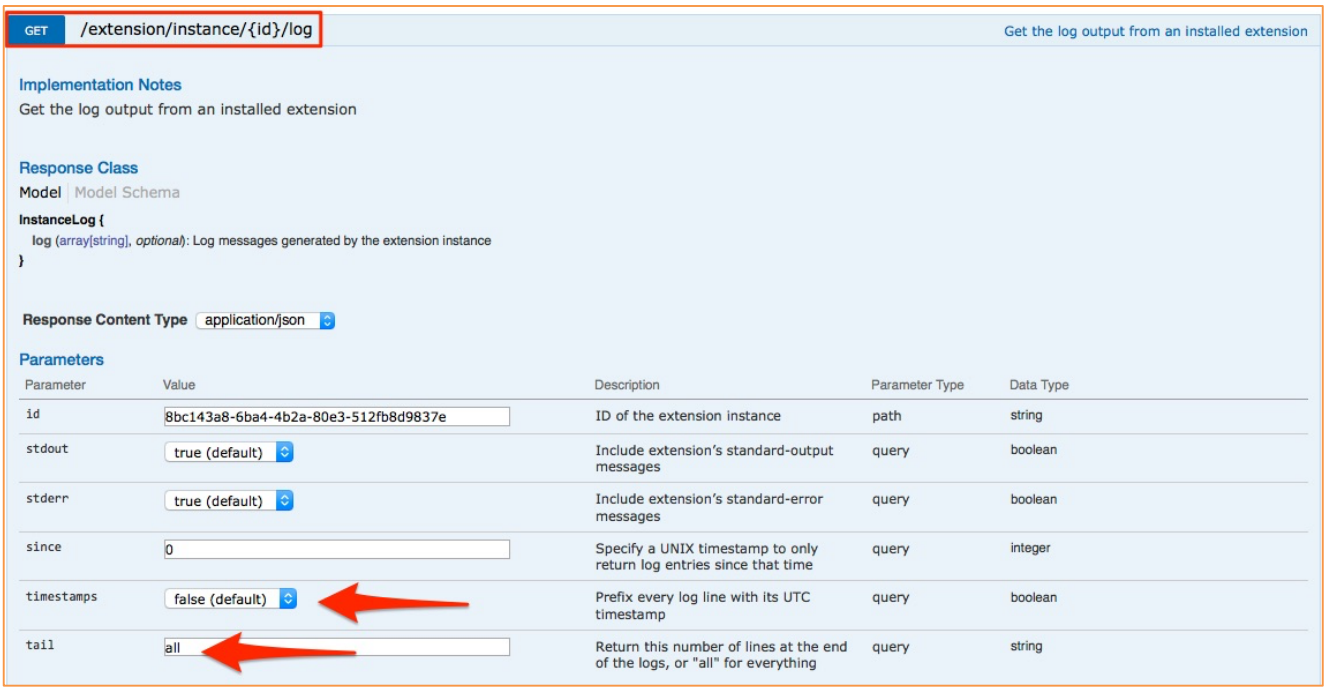
```
{
"logLevel": "DEBUG",
"verifySSLCerts": true",
"besEndpoint": https://my_ip_address_of_UEM:8095,
"tenantExternalId": "S12345678",
"provider": "LOCAL",
"username": "local_user",
"password": "local_user_password"
}
```

Figure 31: Turning on Debug logging for an extension



Once you have configured the extension to capture logs, there are two methods to access them. The first is directly through the API Explore and the second using the "**Collect Logs**" function.

Figure 32: Accessing Logs in an extension from API Explorer UI



NOTE

You can also turn on timestamps by flipping the timestamps option and optionally limit the number of logs returned to say the last 100 rather than 'all' logs by specifying a number in the tail parameter. By default, all logs are returned with no timestamps.

An example of the output from the UI is below.

Figure 33: An example of the logs from the extension in the API Explorer UI

Response Body


```
{
  "log": [
    "\u001b[32m[2016-10-19 22:08:17.444] [INFO] epo - \u001b[39mServer running at port 80\n",
    "\u001b[36m[2016-10-19 22:09:03.196] [DEBUG] epo - \u001b[39mRequest URL: /?macAddress=42987b054b5c\n",
    "\u001b[36m[2016-10-19 22:09:03.204] [DEBUG] epo - \u001b[39mePo Host/port: 10.2.100.122:8443\n",
    "\u001b[36m[2016-10-19 22:09:03.205] [DEBUG] epo - \u001b[39mePo URL: /remote/aruba.executeQuery?target=EPOLeafNode&where=(eq%20EPOComputerProperties.NetAddress%
20%2242987b054b5c%22)&joinTables=EPOComputerProperties&%3Aoutput=json\n",
    "\u001b[36m[2016-10-19 22:09:03.725] [DEBUG] epo - \u001b[39mSystem Not Found (42987b054b5c).\n",
    "\u001b[36m[2016-10-19 22:09:43.057] [DEBUG] epo - \u001b[39mRequest URL: /?macAddress=60672001E42A\n",
    "\u001b[36m[2016-10-19 22:09:43.060] [DEBUG] epo - \u001b[39mePo Host/port: 10.2.100.122:8443\n",
    "\u001b[36m[2016-10-19 22:09:43.060] [DEBUG] epo - \u001b[39mePo URL: /remote/aruba.executeQuery?target=EPOLeafNode&where=(eq%20EPOComputerProperties.NetAddress%
20%2260672001E42A%22)&joinTables=EPOComputerProperties&%3Aoutput=json\n"
  ]
}
```


WARNING

Remember after collecting logs or turning on DEBUG mode, please ensure you return it back to the INFO level. DEBUG mode should only be enabled under guidance from Aruba TAC.

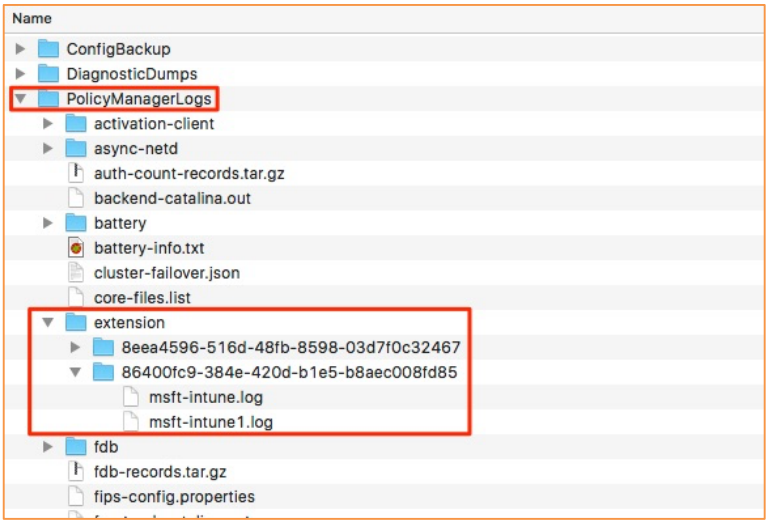
Accessing Extension logs using 'Collect Logs'

In addition to viewing the log messages as shown above, you can also configure the extension to log messages so that they can be collected and examined via the Policy Manager '**Collect Logs**' system function. This is extremely useful for Aruba TAC.

If there is a requirement for Aruba TAC to investigate a system issue, one of the items they regularly ask for is the system logs to aid with their diagnostic investigation. By default the "logLevel" is set to INFO, but TRACE, DEBUG, INFO, WARN, ERROR, or FATAL can also be set. Any of the levels will display the information for the selected state and lower. If INFO is selected, it will show messages for INFO, WARN, ERROR, FATAL.

After the logs have been collected and expanded, you can locate the extension logs in the following location '**PolicyManagerLogs->extension**' as shown below.

Figure 34: Extension logs location in 'Collect Logs' diagnostic GZ file



Appendix C – Considerations for Installing in a Cluster

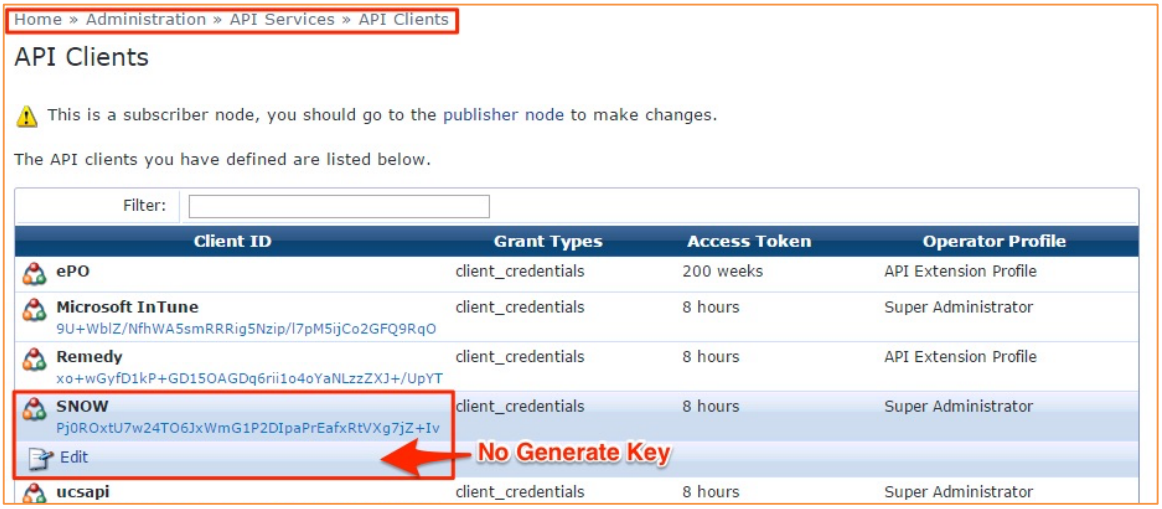
All previous information in this TechNote applies to a single ClearPass node (publisher-only) deployment. When installing the extension subscriber nodes within a cluster, there are a number of things that should be considered.

Extension Installation Procedure for a Subscriber

When installing an extension on a subscriber, the workflow is a slightly different as there is no process to configure API Clients, generate OAuth2 Access Tokens, etc., as all the configuration is performed on the Publisher.

Installing an extension on a subscriber requires that you use the API Explorer REST interface to install the extension, using the API Explorer requires access to an OAuth2 Access Token. The problem is that on the subscriber node there is no option to generate the Access Token, as shown below.

Figure 35: Subscriber with no "Generate Key" option



NOTE

The solution is to configure the API client on the Publisher, then generate an Oauth2 Access Token. Then COPY the HTTP Authorization key from the Publisher and use it on the Subscriber. Ensure you copy the "Bearer" portion of the key as well as shown below.

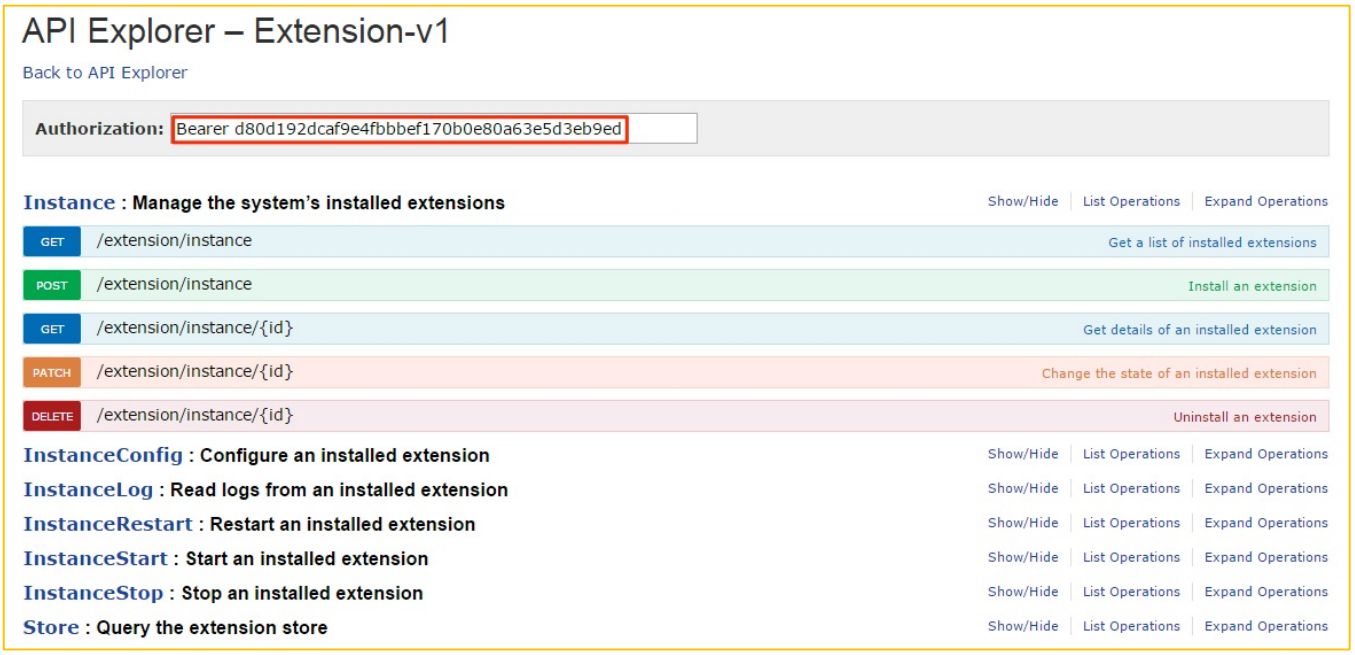
Figure 36: Generate API Key in Publisher and copy it

After you have copied the OAuth2 Access Token from the Publisher, go directly to the **https://<subscriber_node>/api-docs** URL on the subscriber.

NOTE

Ensure you enter the Authorization Header in the API Explorer before using trying to use the functions.

Figure 37: Entering the HTTP Authorization key into the Subscriber



Extension IDs in a Cluster

Note that the store-ID never changes and is specific to a published version of the extension. The **Extension ID** refers to the Store ID of the extension. Installing an extension on five subscriber nodes would result in a different **run-time ID** on each node.