ClearPass and Microsoft Intune Integration TechNote

a Hewlett Packard Enterprise company

ClearPass

TechNote

Change Log

Version	Date	Modified By	Comments
0.1 & 0.2 & 0.3	June 2016	Danny Jump	Draft checked by D Wilson, M Adjali and Microsoft
1.0	Oct 2016	Danny Jump	Initial Restricted-Access Published Version
1.1	Dec 2016	Danny Jump	Initial GA Published Version
1.2	May 2017	Josh Santomieri	Updates for new extension version (3.0.0)
2.0	May 2017	Danny Jump	Minor updates from TAC/ERT and new TechNote Template

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

This TechNote covers the setup, configuration, and monitoring of the Microsoft Intune ClearPass Extension within ClearPass. 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. Configuration and control of ClearPass Extensions is through the ClearPass REST API framework.

Installation of the Microsoft Intune ClearPass Extension is performed via the REST API interface. ClearPass REST APIs were introduced in ClearPass 6.5.0 and have consistently been enhanced, access to the APIs is through the following URL https://<Your-ClearPass-Server>/api-docs.

Prior to accessing the ClearPass REST APIs, you need to complete some pre-configuration steps, which is covered in the Installation, Configuration and Setup section below.

What's new in this ClearPass Extension V3

Externally we skipped releasing V2. V2 was an internal only release. In V3 we have added support for a new Intune endpoint attribute, Ownership. This allows ClearPass to differentiate between Corporatly enrolled and personally owned devices. Think of this as a corporate asset vs a BYOD device. Knowing the difference could be critical in how you allow these devices on the network. Additionally the process to collect the Microsoft Intune and Azure AD attributes to complete the configuration has changed, these changes are documented within this document.

Software Requirements

The minimum software version required for ClearPass is 6.6.0. At the time of writing, ClearPass 6.6.2 is the latest available and recommended release. ClearPass runs on a hardware appliance with pre-installed software or as a Virtual Machine under the following hypervisors.

- VMware ESXi 5.0, 5.1, 5.5, 6.0, or higher
- Microsoft Hyper-V Server 2012 R2
- Hyper-V on Microsoft Windows Server 2012 R2

Hypervisors that run on a client computer such as VMware Player are not supported.

Microsoft Intune can manage the following device platforms:

- Apple iOS 8.0 and later
- Google Android 4.0 and later (including Samsung KNOX SDK 4.0 and higher)
- Google Android for Work (requirements)

- Windows Phone 8.1 and later
- Windows 8.1 RT
- PCs running Windows 8.1
- PCs running Windows 10 (Home, Pro, Education, and Enterprise versions)
- Devices running Windows 10 IoT Enterprise (x86, x64)
- Devices running Windows 10 IoT Mobile Enterprise
- Windows Holographic & Windows Holographic Enterprise
- Mac OS X 10.9 and later

Microsoft maintains an up to date version of this list located here:

https://docs.microsoft.com/en-us/intune/get-started/supported-mobile-devices-and-computers

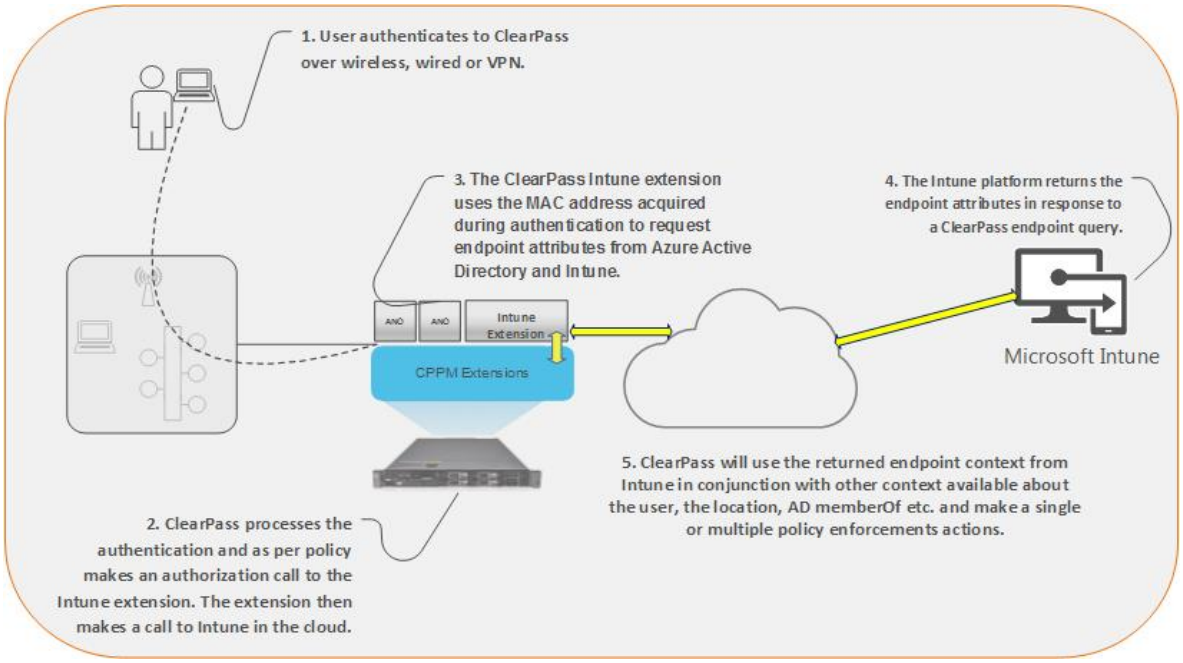
Installation and Deployment Guide

The document assumes your ClearPass environment is already configured and operational. If you require assistance with basic deployment refer to the following deployment guide located here: http://www.arubanetworks.com/techdocs/ClearPass/Aruba_DeployGd_HTML/Default.htm

Pictorial view of the Integration

The diagram below shows an overview of the components and how they interact together.

Figure 1: *Pictorial view of ClearPass's integration with Microsoft Intune and Azure AD*

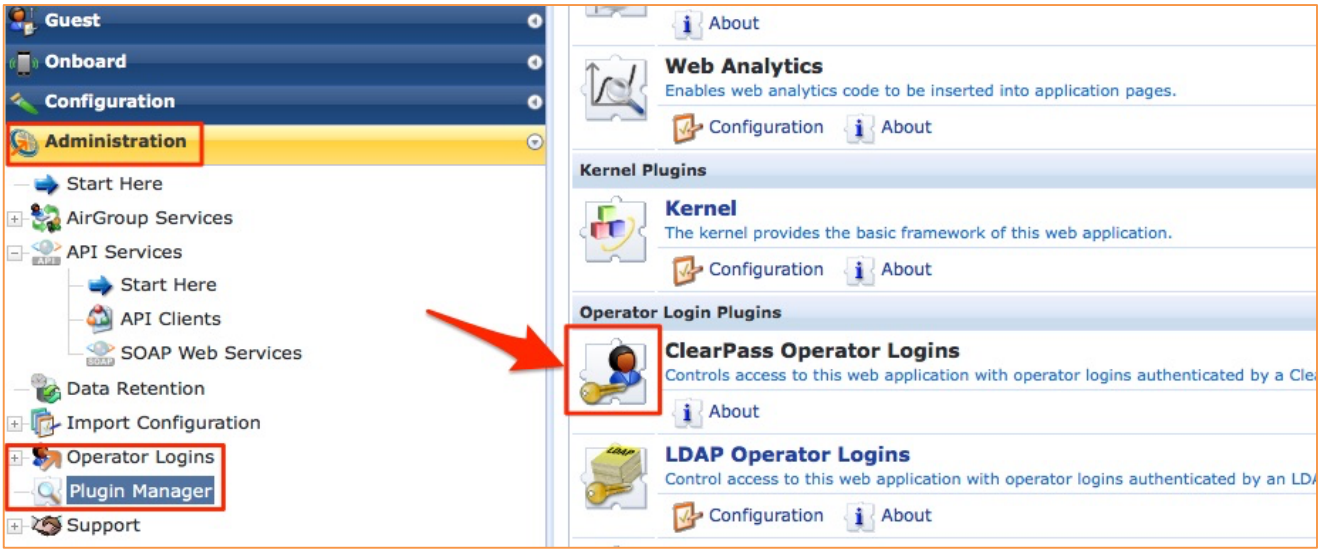


Installation, configuration, and setup

Before you can access the ClearPass platform and configuration APIs, you need to set up an API client. An API client provides the authentication and authorization for the REST APIs.
Authentication is performed utilizing OAUTH2, which is an authorization framework that enables applications to obtain limited access to data over a HTTP service without sharing their private credentials.
Before accessing the the API its necessary to configure an Operator Profile that will be associated with the API client used. This new Operator profile will be used in the next section when creating the API Client.

Login to Guest, go to **Administration -> Plugin Manager -> [click on] Local Operator Logins**

Figure 2: *Configuring Local Operator Profiles*



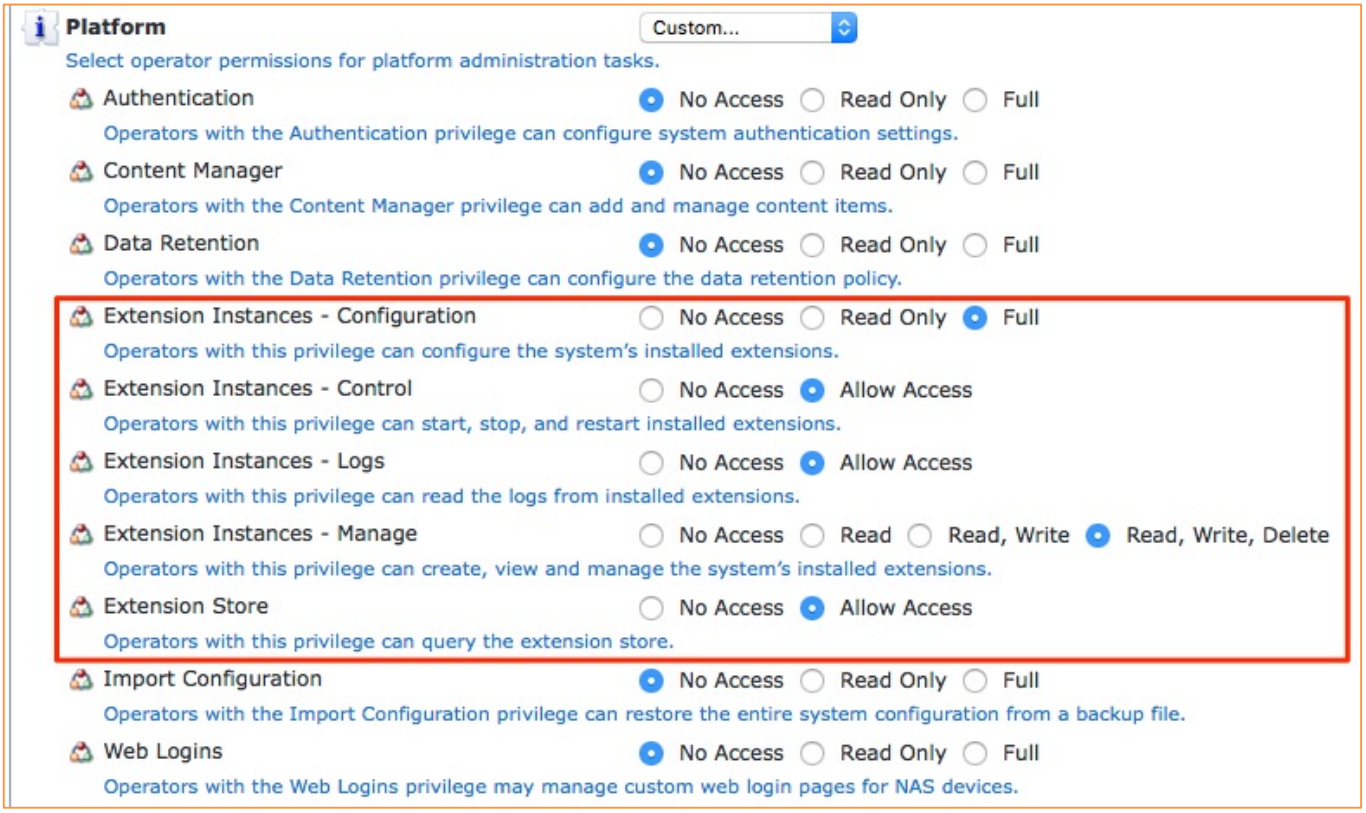
Click on Manage Operator Profiles

Figure 3: *Manage Operator Profiles*

Next click on '**API Guest Operator'** and select '**Duplicate'**. ClearPass will copy the profile and call it '**API Guest Operator (2)**'. Now edit and rename it to be '**API Extension Profile**'.

Next go down to '**Platform'** and change '**No Access**' to '**Custom**' and then for the '**Extension**' options shown below ensure the configuration matches, then scroll to the bottom of the page and click on save.

Figure 4: *Modifying Operator Profile permissions for extensions*



The profile created above will be used in the next step as the Operator profile when generating the OAUTH2 API Client.

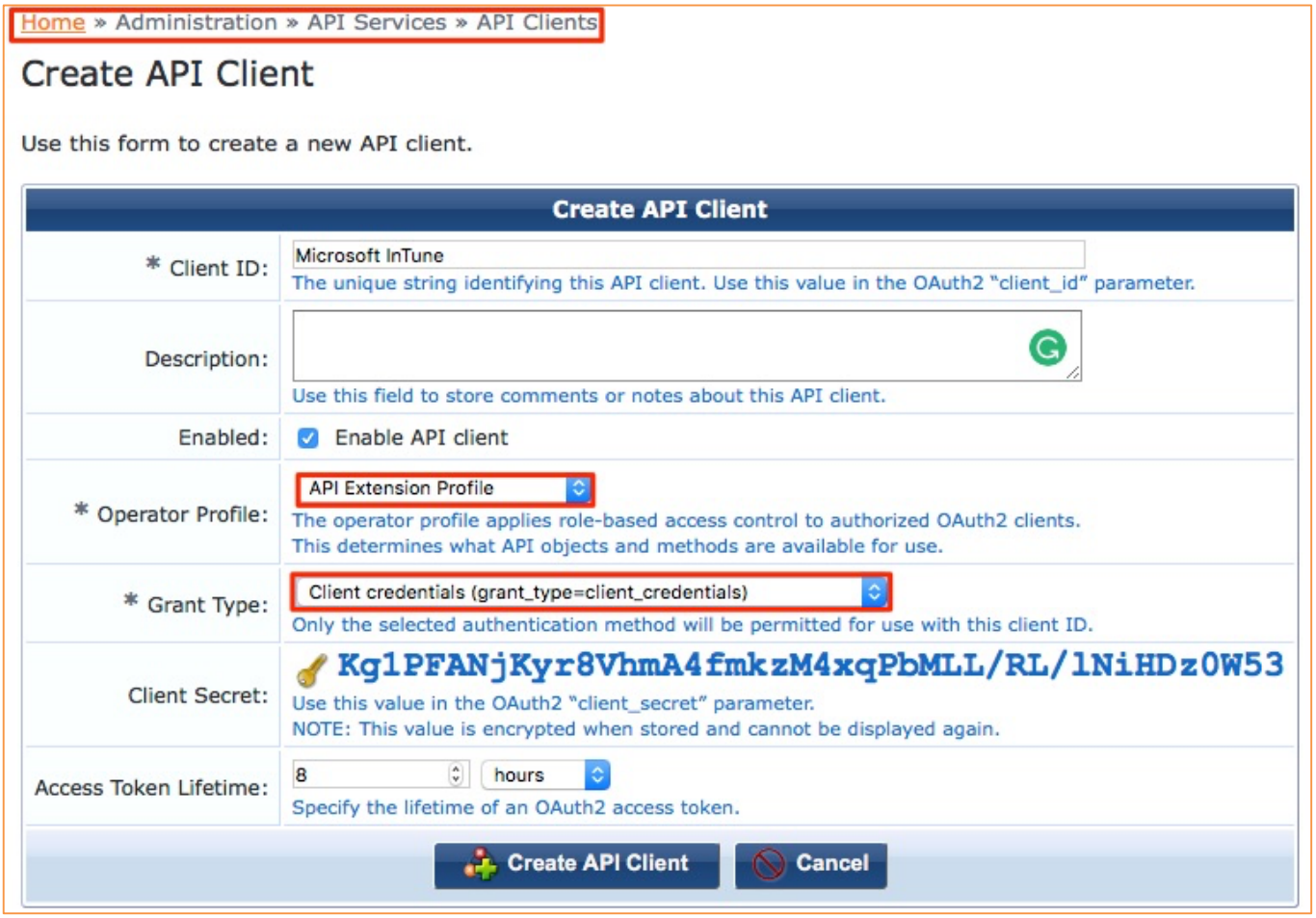
Setup of API Client

The first step in installing and enabling the Microsoft Intune ClearPass Extension is to create an API Client. Log into ClearPass Guest at **https://<Your-ClearPass-Server>/guest**

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

Client ID: Your choice (in this example we chose Microsoft Intune)
Operator Profile: API Extension Profile
Grant Type: Client credentials

Figure 5: *Creating your API client*

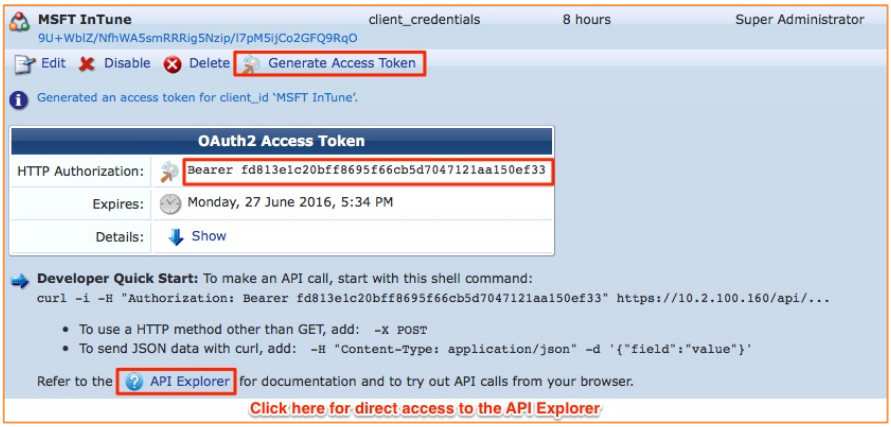


Click on '**Create API Client**' to save and create the API Client.

Generate an Access Token

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

Figure 6: *Generate the Access Token*



This will pre-populate the Authorization header in the API Explorer.

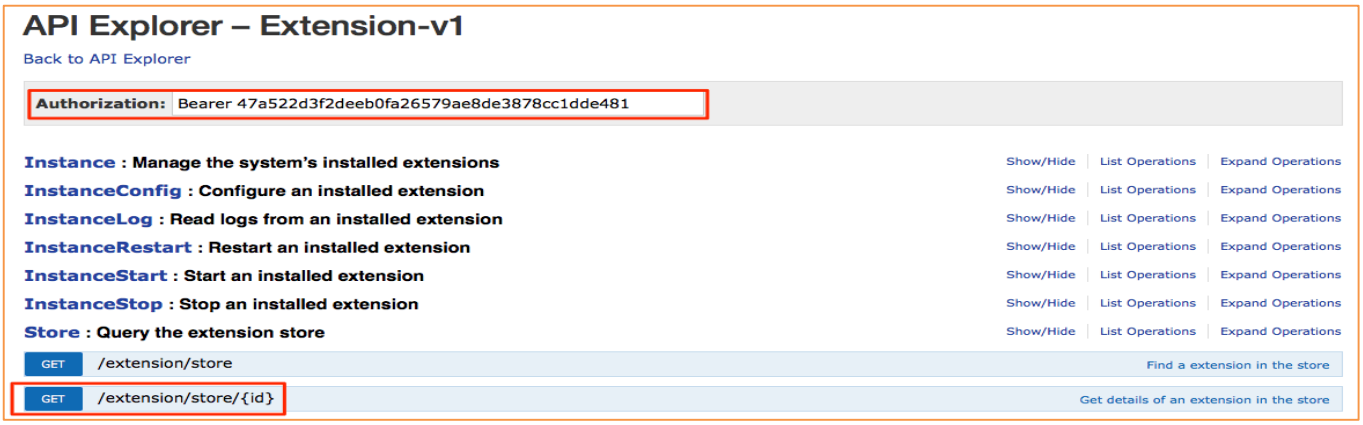
Go to the Extension APIs

Click on **Extension > Store.**

Figure 7: *API Explorer UI*

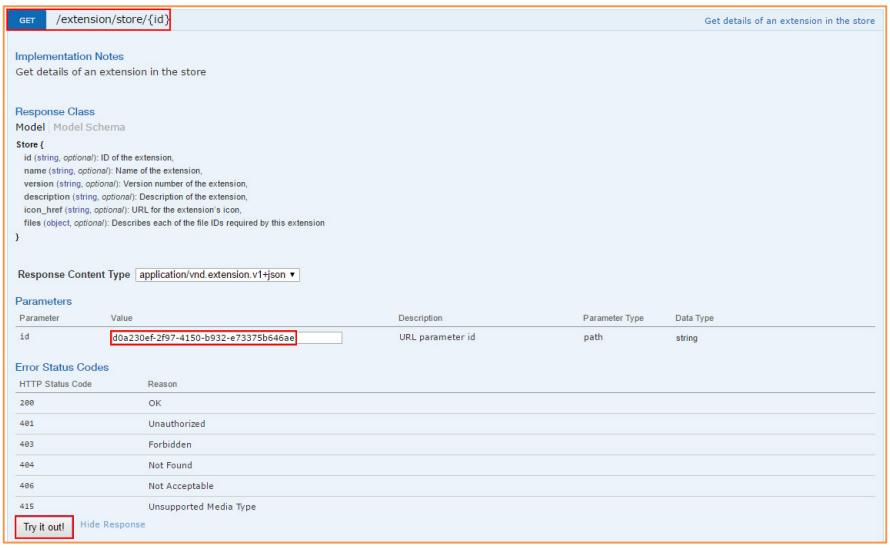
Under Store, click **GET /extension/store/{id}**

Figure 8: *Clicking on 'Store'*



Notice that the Authorization header is automatically populated. This is populated from creating the token above in the previous step. Now we need to expand the **GET /extension/store/{id}** and paste in the extension's store ID. The store ID for the Intune V3 extension is a fixed value of: **d8d419a8-6a1a-4d7e-b564-81d11a75105a**. Click 'Try it out!'

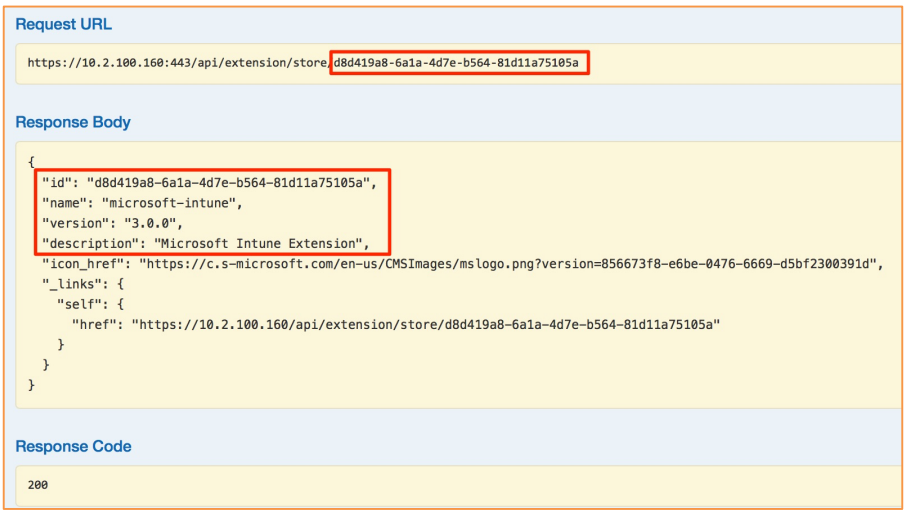
Figure 9: *Checking in the extension store for a particular extension ID*



The store ID of the extension **d8d419a8-6a1a-4d7e-b564-81d11a75105a** is unique to each released version. As the underlying feature and functions are enhanced, the store ID may change as the version of the extension increments.

This should return the details for the Microsoft Intune Extension, and confirm that your ClearPass has access to the extension.

Figure 10: *Details on the extension*

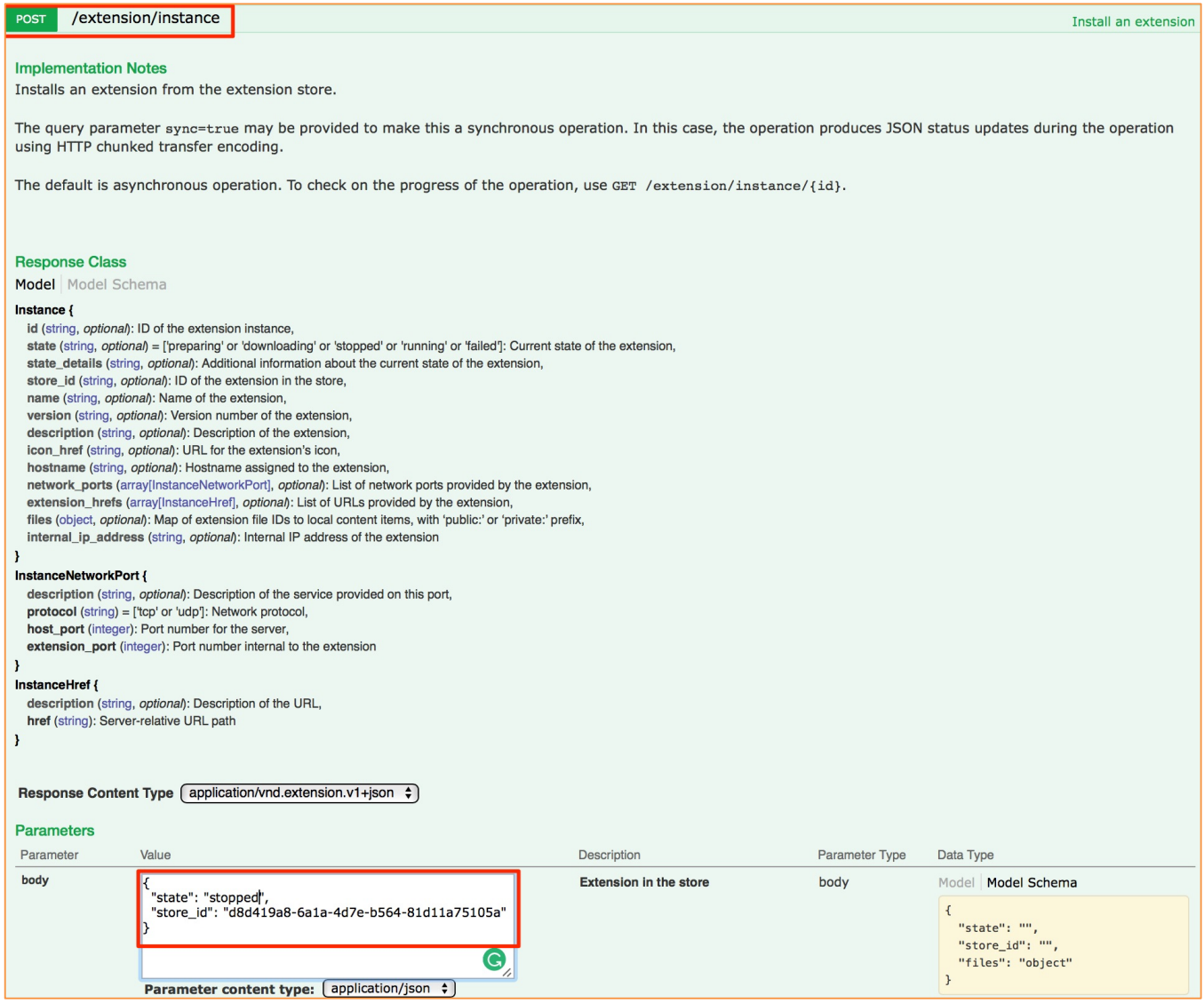


Install the extension

Under **Instance > POST /extension/instance**, paste in the body as shown below:

{"state":"stopped","store_id":"d8d419a8-6a1a-4d7e-b564-81d11a75105a"} and click 'Try it out!'

Figure 11: *Installing the extension direct from the extension store.*



NOTE

This will return an extension ID (this is different from the store ID). Make a copy of this ID as it will be required later.

Notice in the extension configuration above, it is set to install "**stopped**" and not to start, below the state of the extension as "**preparing**", this shows it is downloading, installing and deploying.

Figure 12: The Extension preparing for installation

From the above, we see the ID to be **03b11007-cd5a-482f-9fc3-8ea6a2a96bf8**. Let's query the state of the install using this ID, we'll call this the run-time ID.

NOTE

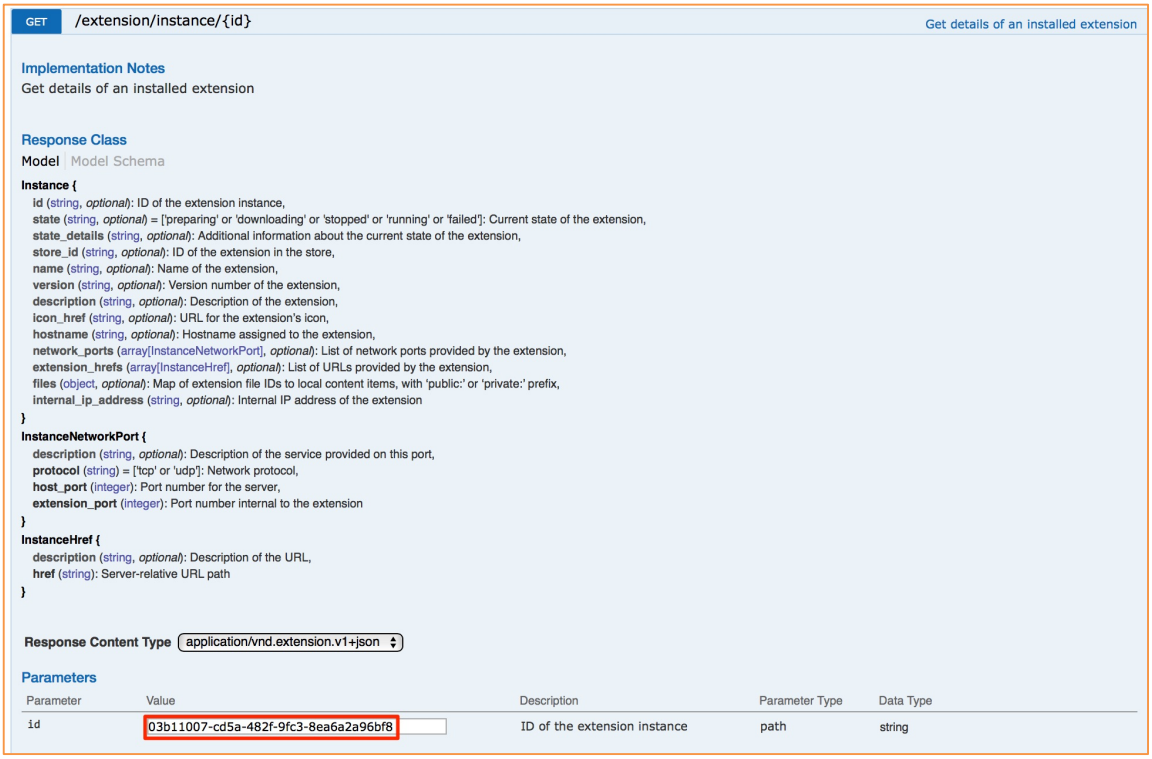
Remember your run-time extension "ID" will be different from that documented here in this TechNote.

Wait for Extension to Install

Under **Instance > GET /extension/instance/{id}** paste in the ID from the previous step:

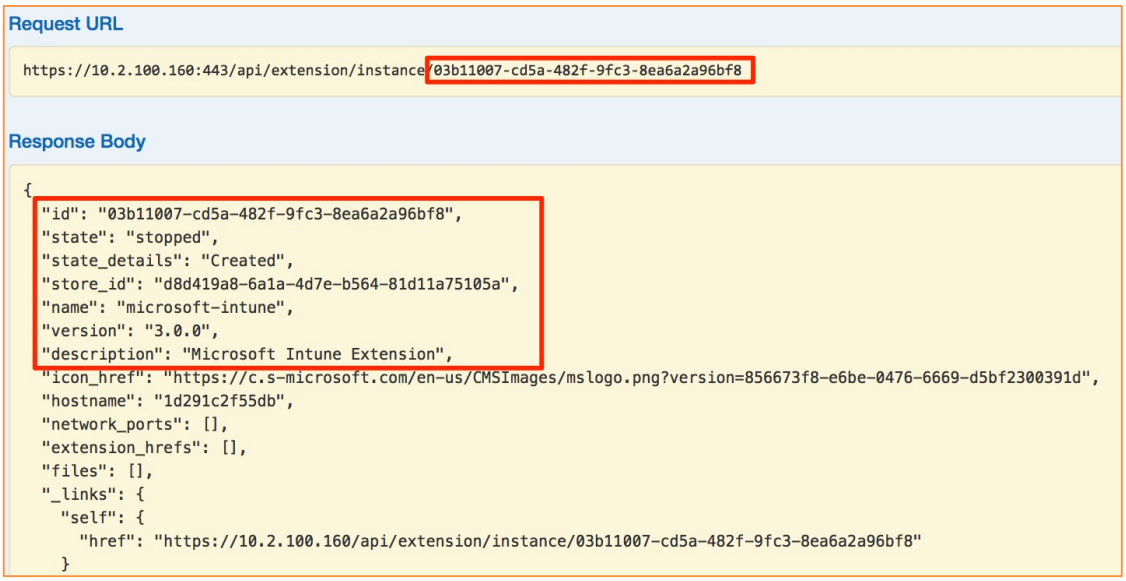
03b11007-cd5a-482f-9fc3-8ea6a2a96bf8 and click 'Try it out!'.

Figure 13: Checking in extension installation progress



Within the response body from this GET we see the following:

Figure 14: *Response to check on progress of extension installation*



The details of the extension will be displayed (could be "downloading", etc.). Eventually the state will change to either "stopped" or "failed". In our case we can see the installation is Created and stopped.

Collecting Information from Microsoft to Configure Intune

Below we will cover the process of adding a 'ClearPass App' into Azure as an application and enabling the necessary application level permissions. Think of this as the gateway between the ClearPass on-premises environment and Microsoft Azure Intune.

It is assumed you have your Intune/Azure environment already setup and configured. The setup of these environments is beyond the scope of this TechNote.

In order to complete the integration, you need to collect multiple pieces of information from Intune and the Azure platform that are required to allow us to complete the extension configuration. The goal is to collect information to complete the highlighted attributes below in red:

```
{
"tokenEndpoint": "<tokenEndpoint>",
"tenantId": "<tenantId>",
"clientId": "<clientId>",
"clientSecret": "<clientSecret>",
"resourceUri": "https://api.manage.Microsoft.com/",
"apiVersion": "1.1",
"verifySSLCerts": true,
"logLevel": "INFO"
}
```

To start, open up your favorite text editor, and copy and paste the above text block into it. You'll be editing several lines for this JSON payload.

The first piece of information you need to update is the "**tokenEndpoint**". This is the URL that ClearPass uses to create OAuth2 Tokens that provide access to Azure Active Directory and Graph services.

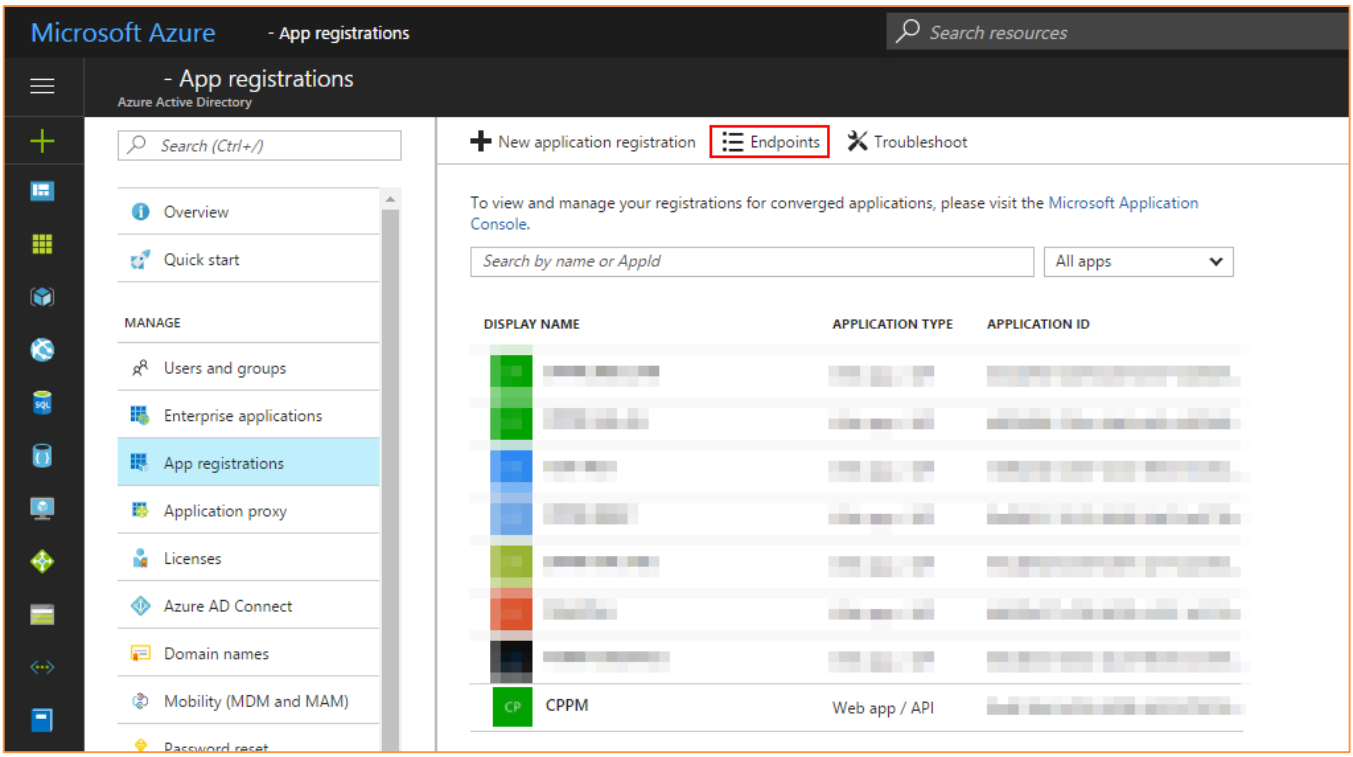
To get the "**tokenEndpoint**" value, first log into the Azure Portal. Point your browser to **https://portal.azure.com**. Log in using your Intune Tenant Admin account. We assume here you have already identified and configured at least one of your Intune accounts with Administrator rights. You can see below where we've logged in with a "onmicrosoft.com" account.

You may have to accept permissions for the account to use the API Explorer features.

Once logged in, open Azure Active Directory and select "**App Registrations**".

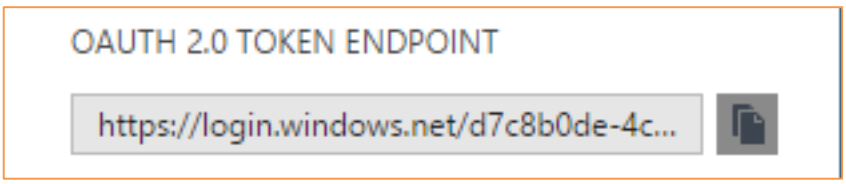
In “**App Registrations**”, click on the “**Endpoints**” menu option to view your Azure endpoints.

Figure 15: *Azure Application registrations*



From the list of endpoints, copy the OAUTH 2.0 TOKEN ENDPOINT value. This is the value you will use as the “**tokenEndpoint”** in the configuration.

Figure 16*: Capturing the Oauth2 token endpoint value*



Paste the copied endpoint URL in your ID string into the **tokenEndpoint** configuration item.

Next, we need the “**tenantId**” value. To get this, simply copy out the ID portion of the OAuth 2.0 Endpoint. For example, our token endpoint is,

https://login.windows.net/9b84bb69-c703-4cac-8db3-20414b0bb8bc/oauth2/token

From this URL, the highlighted portion is your Tenant ID. Copy this value into the **tenantId** setting of your configuration.

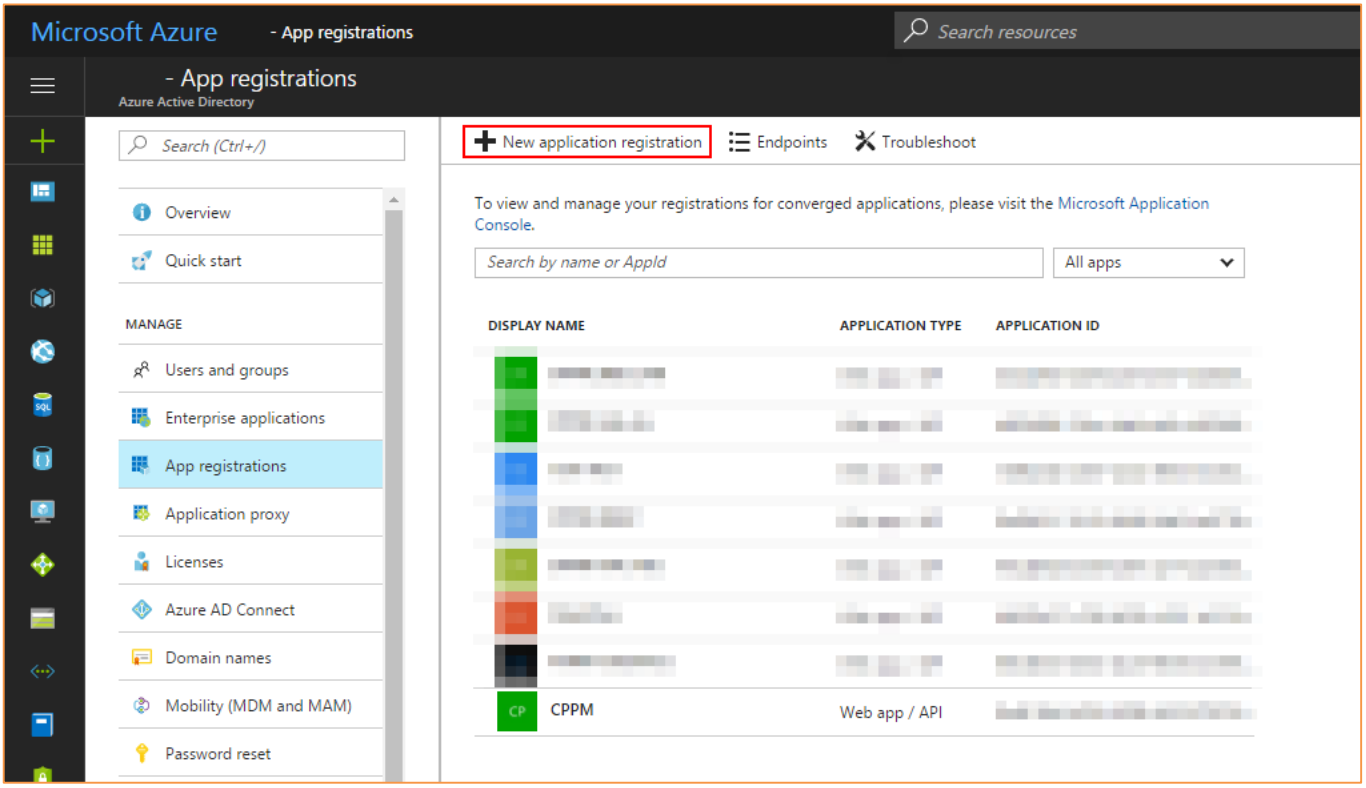
If you already have an Intune Application Registration in Azure Active Directory, you may use that for the rest of the configuration. If you do not have an Application registered in Azure Active Directory, follow the following steps to create one.

These next steps will be use to collect the **clientId** and **clientSecret** settings.

The next step is to create a new App Registration in Azure Active Directory. This is done from **https://portal.azure.com**. You must login with an account that has Administrative access to Azure Active Directory and Intune.

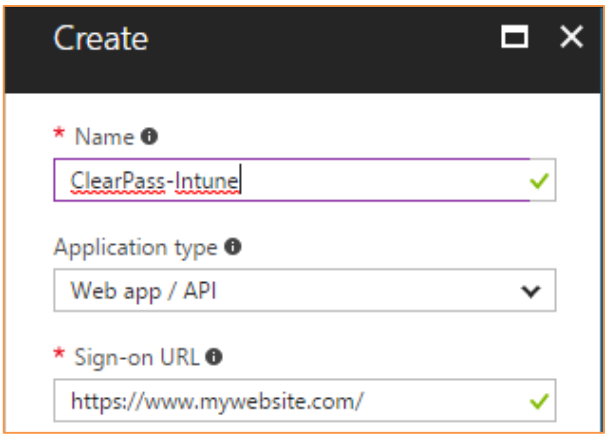
Once logged into the Azure Portal, navigate to Azure Active Directory, select "**App Registrations**" and then click on "**New application registration**", as shown below.

Figure 17: *Creating a new application in Azure*



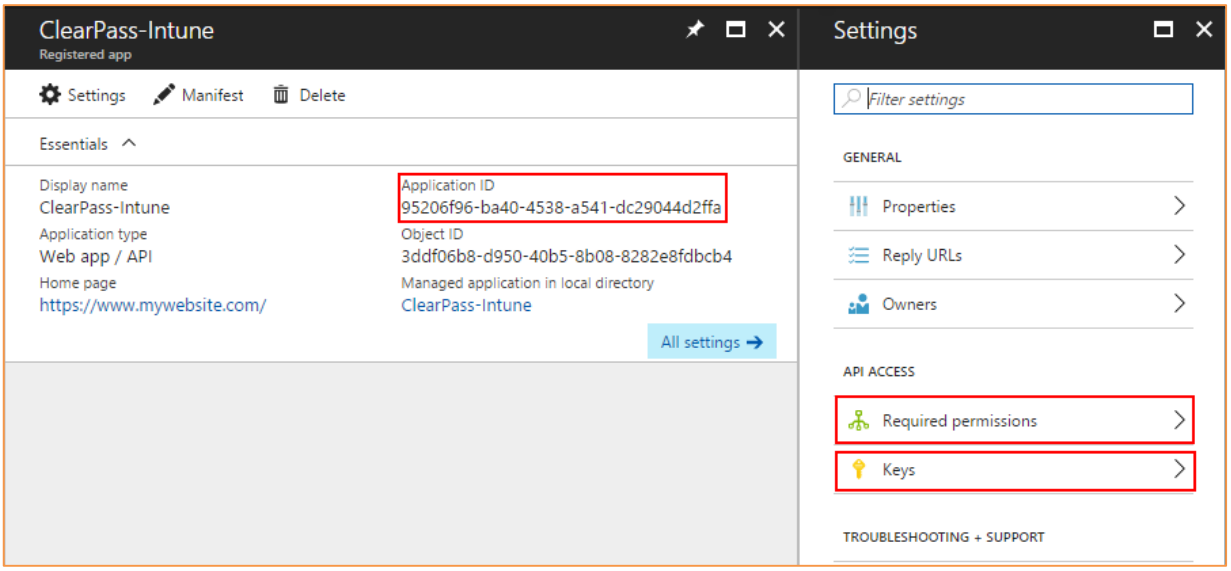
The next step is to create a new application registration. We suggest using the name ClearPass, or something that will clearly identify what the application registration is for. The application type should be set to "**Web app / API**" and Sign-on URL should be set to a valid URL. After entering your settings, click on **Create**.

Figure 18: *Creating a new application registration in Azure*



Once your application has created, select it from the "**App registration**" list to view the application properties and configure the application. The important areas are highlighted below.

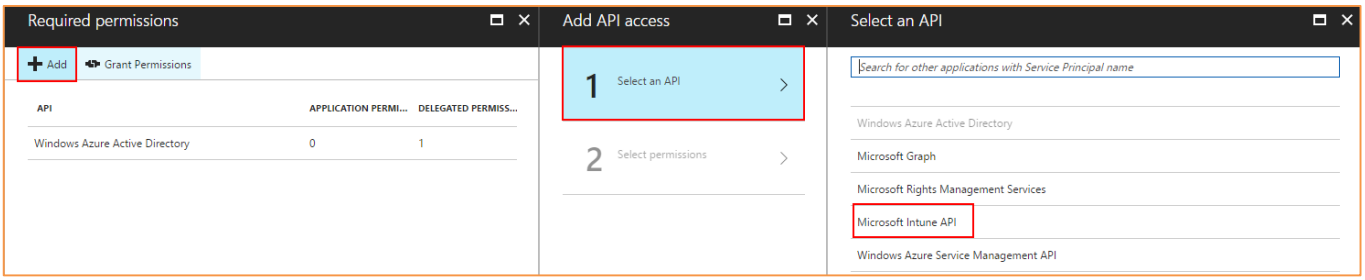
Figure 19: *Capturing important data from your Azure application*



Copy the Application ID: The **Application ID** is the value required for the **clientId** configuration in the extension. You can copy and paste that value to your extension configuration now.

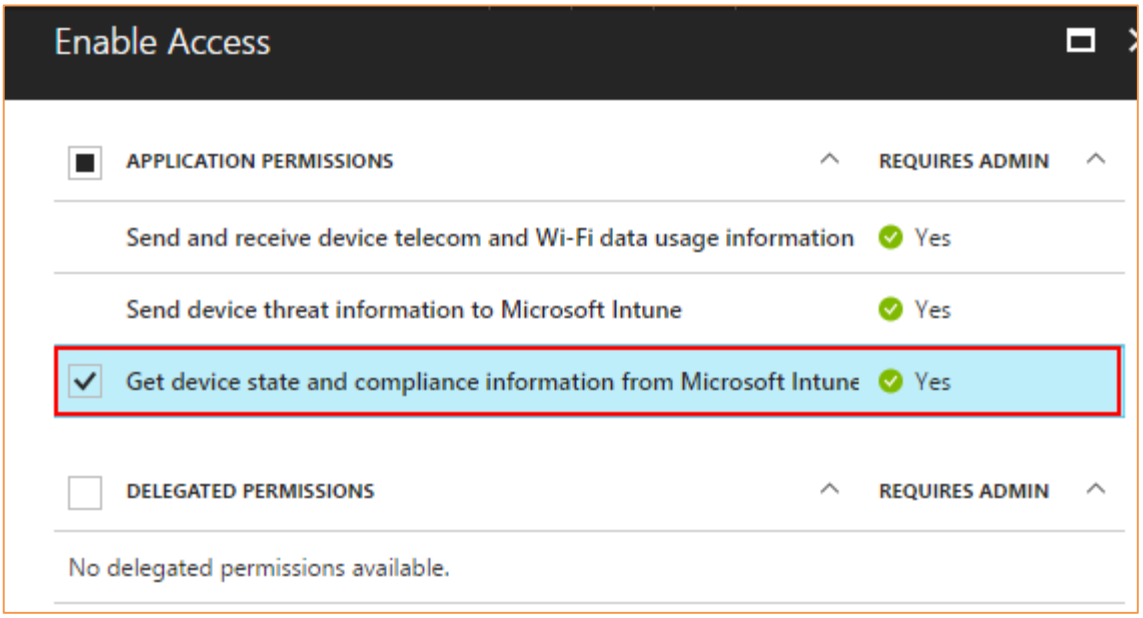
Next set the required permissions for the App Registration. To do this click on the "**Required permissions**" option in the settings panel. Next select "**Add**", then "**Select an API**" finally followed by "**Microsoft Intune API**". Once you have completed that, click on "**select**" to create the permissions.

Figure 20: *Setting application permissions – part1*



After clicking "**select**", you must enable access to "**Get device state and compliance information from Microsoft Intune**" then click "**Select**" followed by "**Done**". Your permissions will now be added.

Figure 21: *Setting application permissions – part2*



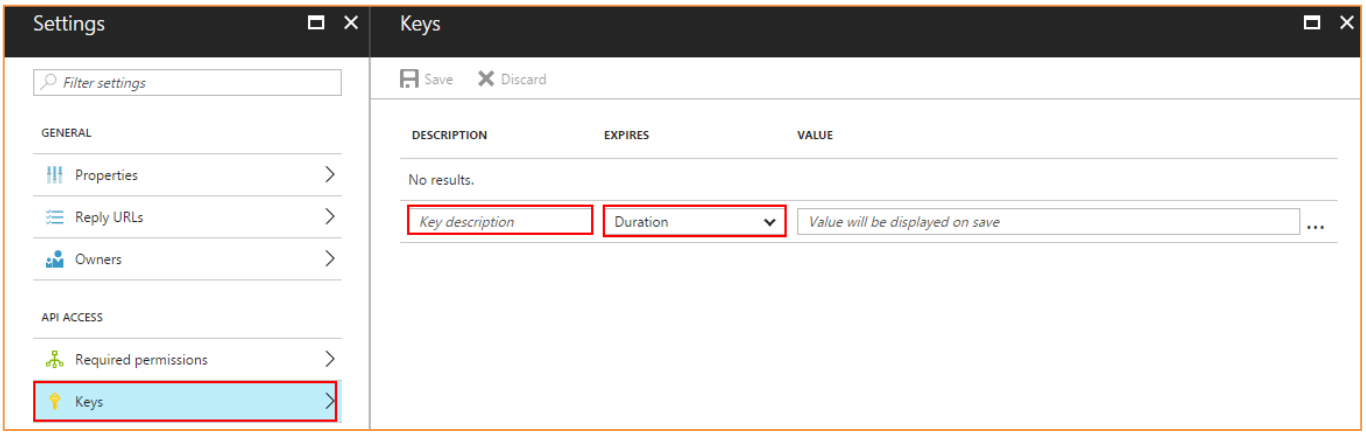
The next and final step is to capture the "**clientSecret**", this currently is a fixed value and maps to the registered Microsoft Intune ClearPass Extension.

NOTE

When you register the Azure AD (AAD) App, the "" will be displayed, you must capture it at this time as it can't be displayed in the future, this is covered below in the following Azure configuration. Follow these steps carefully.

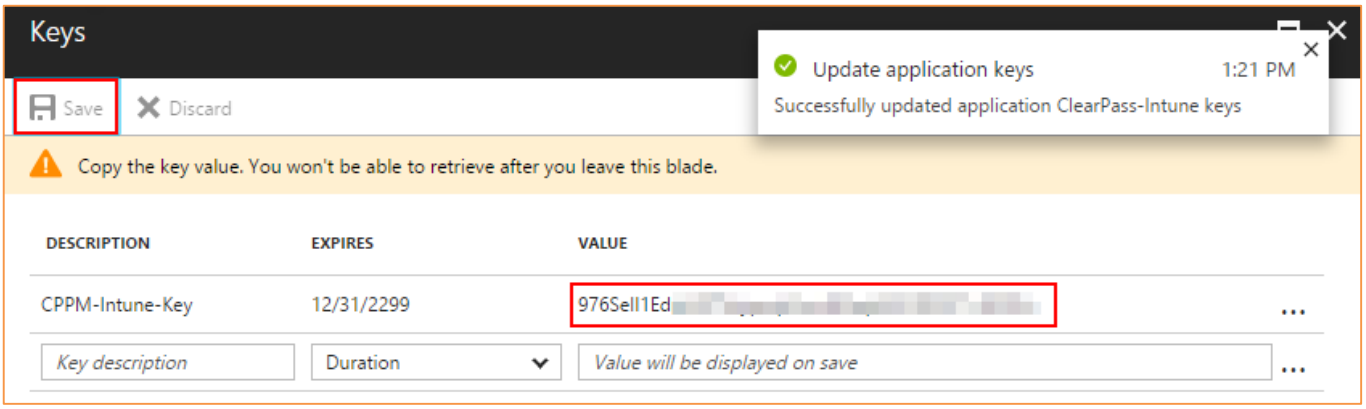
After setting permissions, navigate back to the Application settings and select "**Keys**". In the Keys settings, enter a key description. Use something appropriate to identify the keys for Intune. Then select the duration, we recommend "**Never Expires**" otherwise you will be forced to update the extension configuration when the key expires.

Figure 22: *Creating application clientSecret keys*



After entering your desired information, click "**Save**". This will save your settings and generate the **clientSecret**. Copy the "**value**" to the **clientSecret** setting in the Intune Extension configuration.

Figure 23*: Copying the application clientSecret keys*



WARNING

Remember to save these keys, as the warning above shows, once you exit this screen you are unable to see the keys again.

Finally, you can easily build the string for "**resourceURI**" line (if needed). It should simply be **https://api.manage.microsoft.com.**

These three remining lines are unchanged and should only be modified if directed by Aruba TAC.

- **"verifySSLCerts": true**
- **"apiVersion": "1.1"**
- **"logLevel": "INFO"**

The apiVersion above refers to the Microsoft Intune API version, **not** the ClearPass Extension version.

Use the Intune Data to Configure the Extension

Configure the extension

Under **InstanceConfig > GET /extension/instance/{id}/config** paste in the run-time ID **03b11007-cd5a-482f-9fc3-8ea6a2a96bf8** and click 'Try it out!', this returns a copy of the current configuration, remember this ID is unique to the example in this document, yours will differ.

Figure 24: *Get extension configuration*

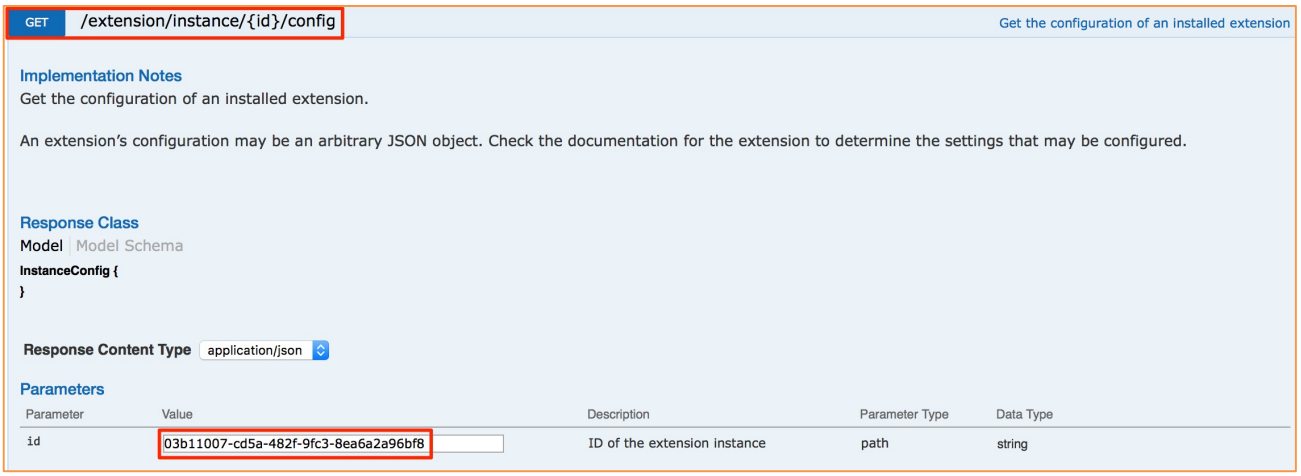
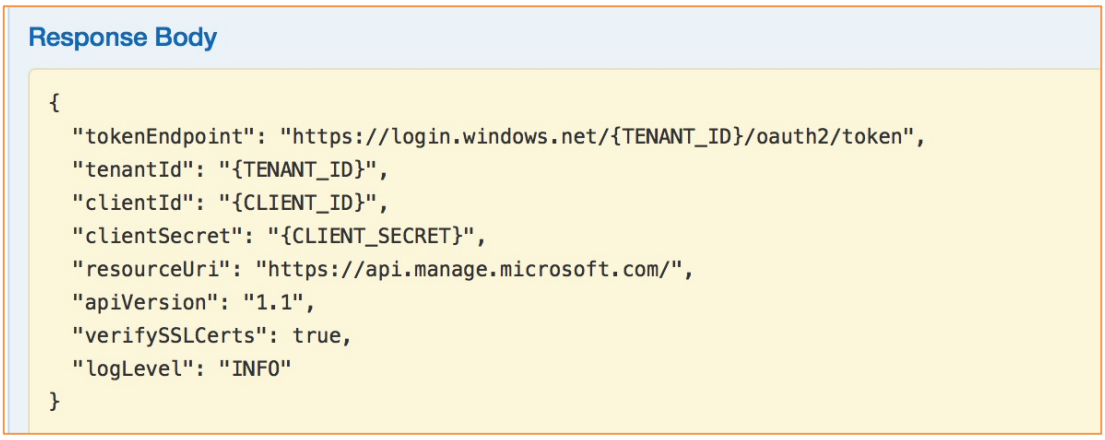


Figure 25: *Response to a request for the ClearPass extension configuration*

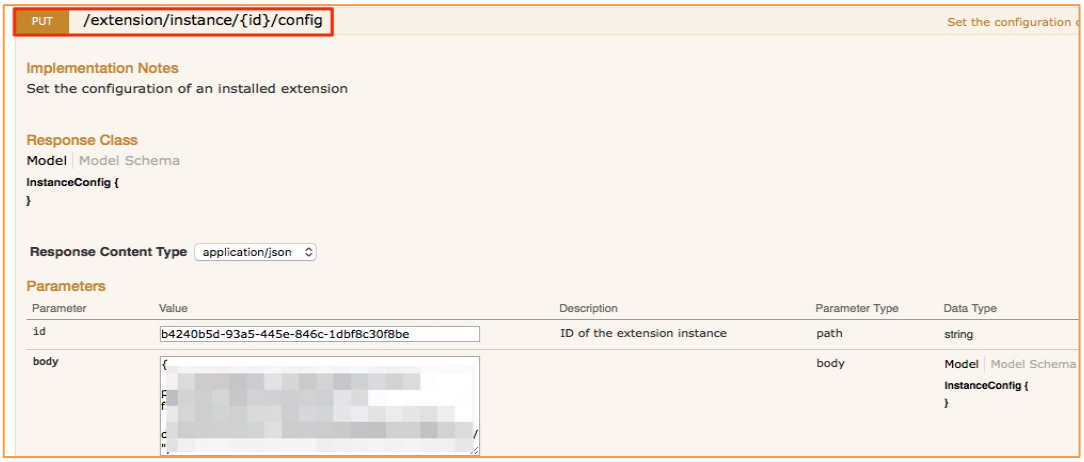


This is an example of the base config after the extension is insalled. The process just completed in collecting the TENEANT_ID, CLIENT_ID etc. will be used to set this configuration.

Notice that debugging is currently set to INFO, this should only be changed if you or Aruba TAC need to DEBUG the extension. In the extension we verify the SSL-certificate presented.

To set the actual configuration under **InstanceConfig > PUT /extension/instance/{id}/config**, copy and paste your run-time extension ID and configuration collected in the previous steps to the PUT method and Click 'Try it out!'.

Figure 26: *Setting the extension configuration*



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

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

Figure 27: *HTTP 204 response to the configuration PUT*

It's important to ensure you format the HTTP body correctly when configuring the Microsoft Intune ClearPass Extension. Below is an example of the parameters that are needed to complete the extension configuration. As covered previous ensure you have the correct formatting of the JSON payload.

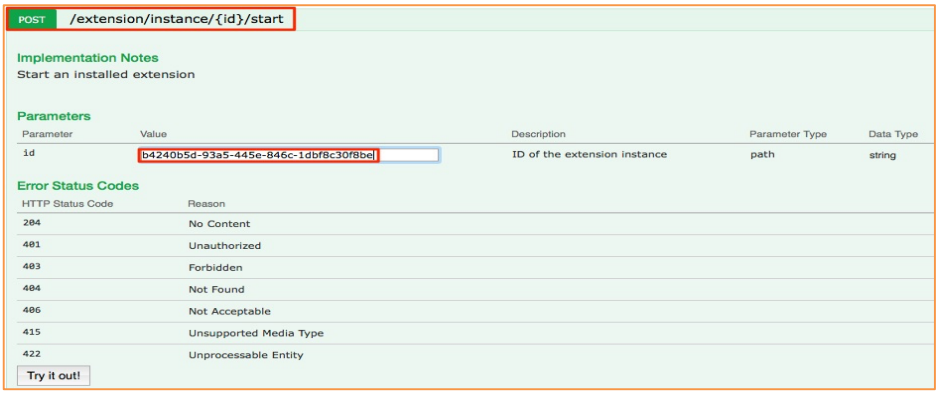
Figure 28: *Example of JSON formatted extension configuration payload*

```
{
 "tokenEndpoint": "https://login.windows.net/47f09275-5bc0-4807-8aae-f35cb0341329/oauth2/token",
 "tenantId": "47f09275-5bc0-4807-8aae-f35cb0341329",
 "clientId": "ed706345-b83f-4af5-9c96-84622b2799f8",
 "clientSecret": "Q92ye91w5EGCe/cHK6fOyLu1pHI0D7ZRTn92MV0w1SI=",
 "resourceUri": "https://api.manage.microsoft.com/",
 "apiVersion": "1.1",
 "verifySSLCerts": true,
 "logLevel": "INFO"
}
```

Starting the extension

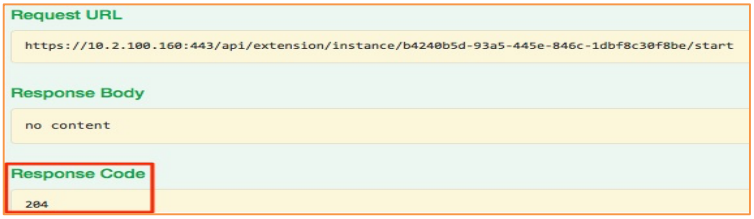
Under **InstanceStart > POST /extension/instance/{id}/start** paste in the extension ID and click 'Try it out!'.

Figure 29: *Starting the extension*



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

Figure 30: *Expected HTTP response to InstanceStart*



Verify the extension is running

Under **Instance > GET /extension/instance/{id}** copy and paste the extension 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 31: *Detailed information on the running extension*

Response Body


```
{
  "id": "9a6360a9-4b18-4b1e-afa0-b67a93a7e9dd",
  "state": "running",
  "state_details": "Started 12 days ago",
  "store_id": "d8d419a8-6a1a-4d7e-b564-81d11a75105a",
  "name": "microsoft-intune",
  "version": "3.0.0",
  "description": "Microsoft Intune Extension",
  "icon_href": "https://c.s-microsoft.com/en-us/CMSImages/mslogo.png?version=856673f8-e6be-0476-6669-d5bf2300391d",
  "hostname": "cd121b191b6a",
  "network_ports": [],
  "extension_hrefs": [],
  "files": [],
  "internal_ip_address": "172.17.0.4",
  "_links": {
    "self": {
      "href": "https://10.2.100.160/api/extension/instance/9a6360a9-4b18-4b1e-afa0-b67a93a7e9dd"
    }
```

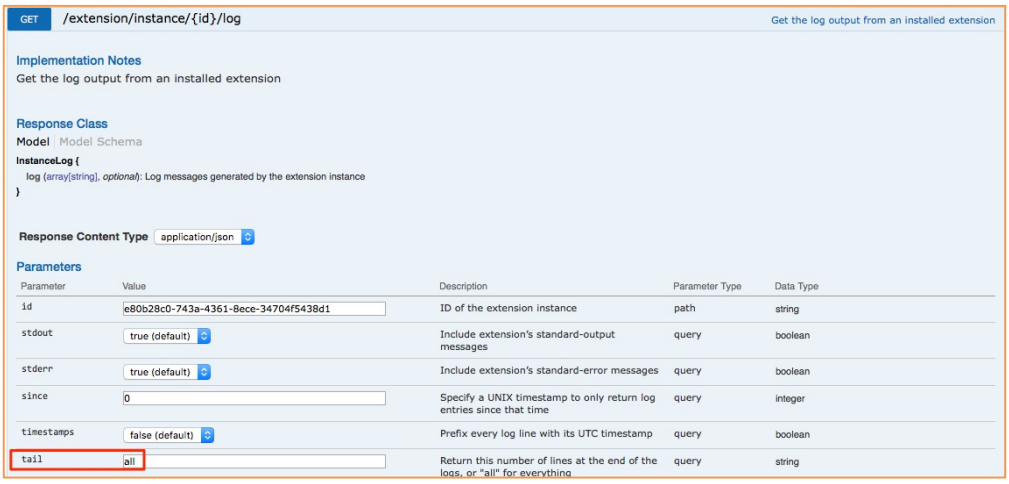

NOTE

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 extension ID. Enter a value for "tail", e.g. 100 will show the last 100 lines of output and then click 'Try it out!'. Note that other settings are applicable when getting logs, e.g. timestamps.

Figure 32: *Getting Debug Logs from the extension.*



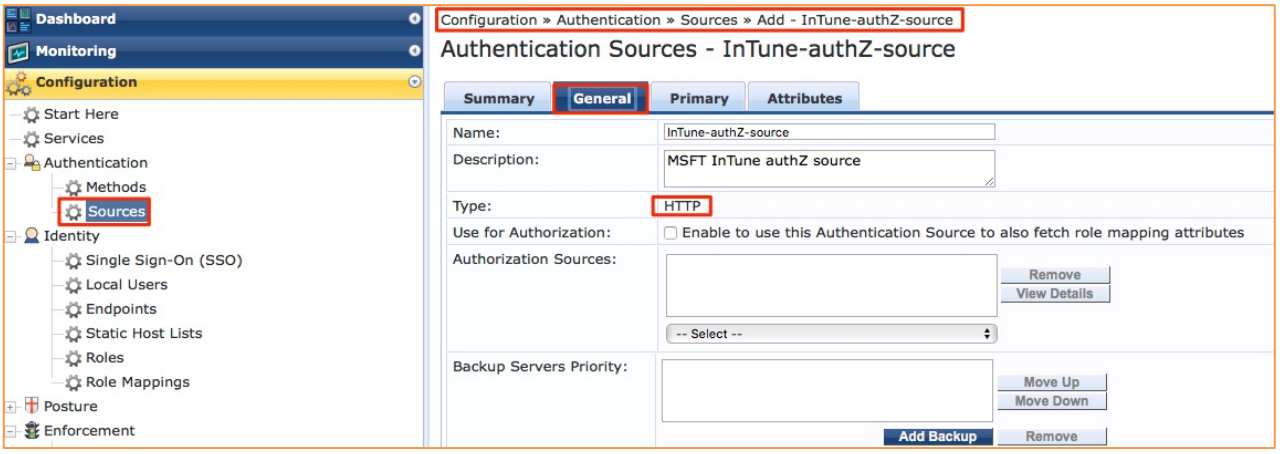
Configuring ClearPass Policy Manager

To complete the configuration, configure an authorization source within ClearPass. With Intune as an authorization source, ClearPass can check with Intune to see if the device is enrolled and managed by Intune before allowing it to connect. Other common use-cases are that ClearPass could any of the returned context such as the version of the installed operating system as the basis for applying specific access policy, or another popular use-case as supported in this latest version of the Intune Extension, is to use the ownership attribute to differentitate between a Corporate or Privately {BYOD} 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 33: *Adding an HTTP authorization source*

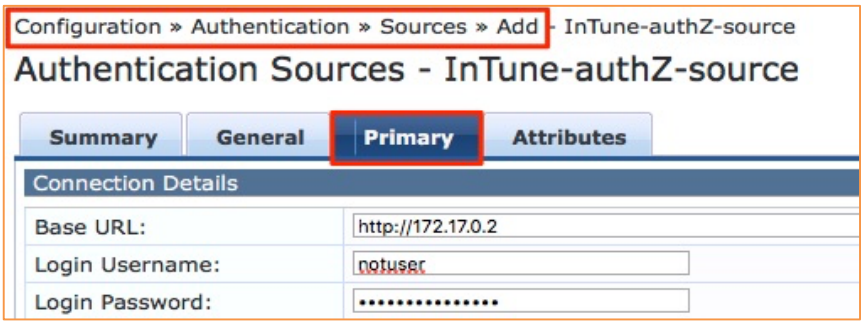


Click on Next. This will advance to the Primary Tab provide the connection details.

NOTE

The Base URL IP address is what you captured in Figure31 above.

Figure 34: *Adding HTTP authorization source credentials*



Its mandated that a Login Username/Password is entered, but is not used, this it can be anything.

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 Intune extension asking for context about the endpoint. The query is indexed off the mac-address of the authenticating endpoint. For completeness, the Filter Query is provided here, copy it carefully.

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

Next build out the definitions of the attributes that will be returned from the Filter Query. These attributes will subsequently be used within our policy-evaluation and ultimately the enforcement policy applied.

Figure 35: *Adding HTTP authorization source query string and returned field definitions*

Configure Filter

Configuration

Filter Name: intune-filter

Filter Query: ?macAddress=%{Connection:Client-Mac-Address-NoDelim}

	Name	Alias Name	Data type	Enabled As
1.	msft_imei	msft_imei	String	Attribute
2.	msft_deviceOwner	msft_deviceOwner	String	Attribute
3.	msft_azureDeviceId	msft_azureDeviceId	String	Attribute
4.	msft_complianceState	msft_complianceState	String	Attribute
5.	msft_isManaged	msft_isManaged	String	Attribute
6.	msft_lastContactTimeUtc	msft_lastContactTimeUtc	String	Attribute
7.	msft_meid	msft_meid	String	Attribute
8.	msft_macAddress	msft_macAddress	String	Attribute
9.	msft_manufacturer	msft_manufacturer	String	Attribute
10.	msft_model	msft_model	String	Attribute
11.	msft_osVersion	msft_osVersion	String	Attribute
12.	msft_serialNumber	msft_serialNumber	String	Attribute
13.	msft_udid	msft_udid	String	Attribute

Once the HTTP authorization source is defined you can use the returned attributes in your policy processing. Below we cover options on how to use the results from the authorization query in an enforcement policy.

NOTE

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

Using data from INTUNE in a ClearPass Enforcement Policy

Multiple use-cases exist for how the data that is returned from Intune can be used in your policy enforcement. In the example below, we are performing multiple checks:

1. Check the device is a Corporately issued and managed device. If true then update the Palo Alto and CheckPoint corporate firewall with context about this device.
2. Check that the device exists in Intune and that it's compliant. In addition to allowing access for this devices, we're also updating the endpoint with the authentication Date & Time so we can track the device's access to the network.
3. If the device is not in compliant then we will apply a Quarantine role.
4. If the device is running an OS that begins with 9.2 [assume iOS] then we flag it as an old-OS.
5. If the device is running an OS that begins with 9.3 [assume iOS] then we flag it as an approved-OS.
6. If the device is running Android OS then we attach a label of Android.
7. If the device is running Android OS then we attach a label of Apple.

Figure 36: *Example of an Enforcement Policy utilizing attributes returned from Intune*

Configuration » Enforcement » Policies » Edit - InTune Device-Policy

Enforcement Policies - InTune Device-Policy

Summary | Enforcement | Rules

Enforcement:

Name:	InTune Device-Policy
Description:	
Enforcement Type:	RADIUS
Default Profile:	[Deny Access Profile]

Rules:

Rules Evaluation Algorithm:	Evaluate all

	Conditions	Actions
1.	(Authorization:InTune-authZ-endpoint-check:msft_deviceOwner *EQUALS* Corporate)	InTune-AllowManaged, CheckPoint-update-node(18)-Enterprise, panw-user-device-update-cppm14
2.	(Authorization:InTune-authZ-endpoint-check:msft_isManaged *EQUALS* false)	InTune-unmanaged-device-ArubaRole, panw-user-device-update-cppm10
3.	(Authorization:InTune-authZ-endpoint-check:msft_isManaged *EQUALS* true) *AND* (Authorization:InTune-authZ-endpoint-check:msft_complianceState *EQUALS* true)	oauth-csa-profile, InTune_Compliant-ALLOW_ACCESS, InTune-update-endpoint authz time, panw-user-device-update-cppm10
4.	(Authorization:InTune-authZ-endpoint-check:complianceState *EQUALS* false)	InTune_Not-Compliant-DENY_ACCESS, panw-user-device-update-cppm10
5.	(Authorization:InTune-authZ-endpoint-check:msft_osVersion *BEGINS_WITH* 9.2)	InTune-old-OS-ArubaRole
6.	(Authorization:InTune-authZ-endpoint-check:msft_osVersion *BEGINS_WITH* 9.3)	InTune-approved-OS-ArubaRole
7.	(Authorization:InTune-authZ-endpoint-check:msft_manufacturer *CONTAINS* Android)	InTune-installed-OS-Android-ArubaRole
8.	(Authorization:InTune-authZ-endpoint-check:msft_manufacturer *CONTAINS* Apple)	InTune-installed-OS-Apple-ArubaRole

Different companies will have different enforcement profiles and policies. The key take away here is that we are using the authorization attributes received from Intune to drive the policy engine into making and taking different enforcement actions for the device as they authenticate on the network.

Appendix A – Authorization source XML configuration file

Below is an example of a HTTP XML configuration file you can copy into a file and Import into your ClearPass node. Before you import the file, you need to amend a couple of the attributes below in your preferred text editor.

your_IP_ADDRESS_goes_here

USERNAME_can_be_anything_goes_here

PASSWORD_can_be_anything_goes_here

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<TipsContents xmlns="http://www.avendasys.com/tipsapiDefs/1.0">
  <TipsHeader exportTime="Sun May 21 00:21:57 CEST 2017" version="6.6"/>
  <AuthSources>
    <AuthSource description="MSFT InTune authZ source" name="InTune-authZ-endpoint-check"
isAuthorizationSource="false" type="HTTP">
<NVPair value="http://your_IP_ADDRESS_goes_here" name="base_url"/>
      <NVPair value="USERNAME_can_be_anything_goes_here " name="username"/>
      <NVPair value="PASSWORD_can_be_anything_goes_here " name="password"/>
      <Filters>
        <Filter paramValues="" filterQuery="?macAddress=%{Connection:Client-Mac-Address-
NoDelim}" filterName="intune-filter">
          <Attributes>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" alias-
Name="msft_imei" attrName="msft_imei"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" alias-
Name="msft_deviceOwner" attrName="msft_deviceOwner"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" alias-
Name="msft_azureDeviceId" attrName="msft_azureDeviceId"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" alias-
Name="msft_complianceState" attrName="msft_complianceState"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" alias-
Name="msft_isManaged" attrName="msft_isManaged"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" alias-
Name="msft_lastContactTimeUtc" attrName="msft_lastContactTimeUtc"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" alias-
Name="msft_meid" attrName="msft_meid"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" alias-
Name="msft_macAddress" attrName="msft_macAddress"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" alias-
Name="msft_manufacturer" attrName="msft_manufacturer"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" alias-
Name="msft_model" attrName="msft_model"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" alias-
Name="msft_osVersion" attrName="msft_osVersion"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" alias-
Name="msft_serialNumber" attrName="msft_serialNumber"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" alias-
Name="msft_udid" attrName="msft_udid"/>
          </Attributes>
        </Filter>
      </Filters>
    </AuthSource>
  </AuthSources>
</TipsContents>
```

Appendix B – Additional diagnostics / support

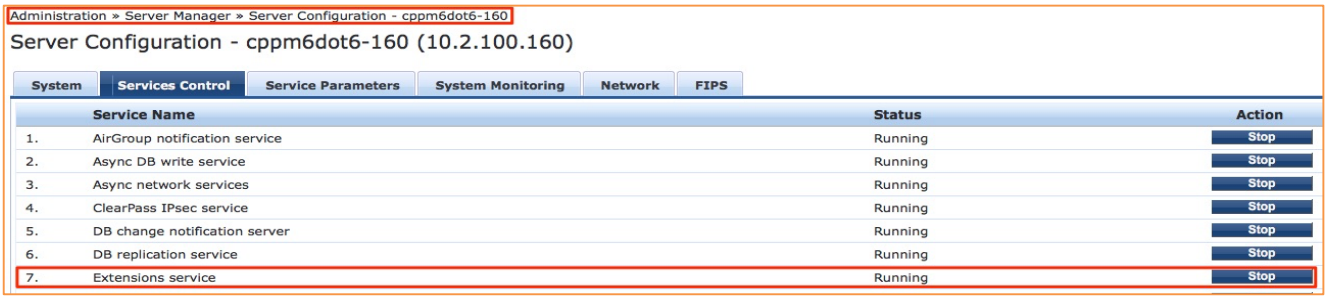
Extension Service

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

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 37: *Checking on extension service and how to start/stop the service*

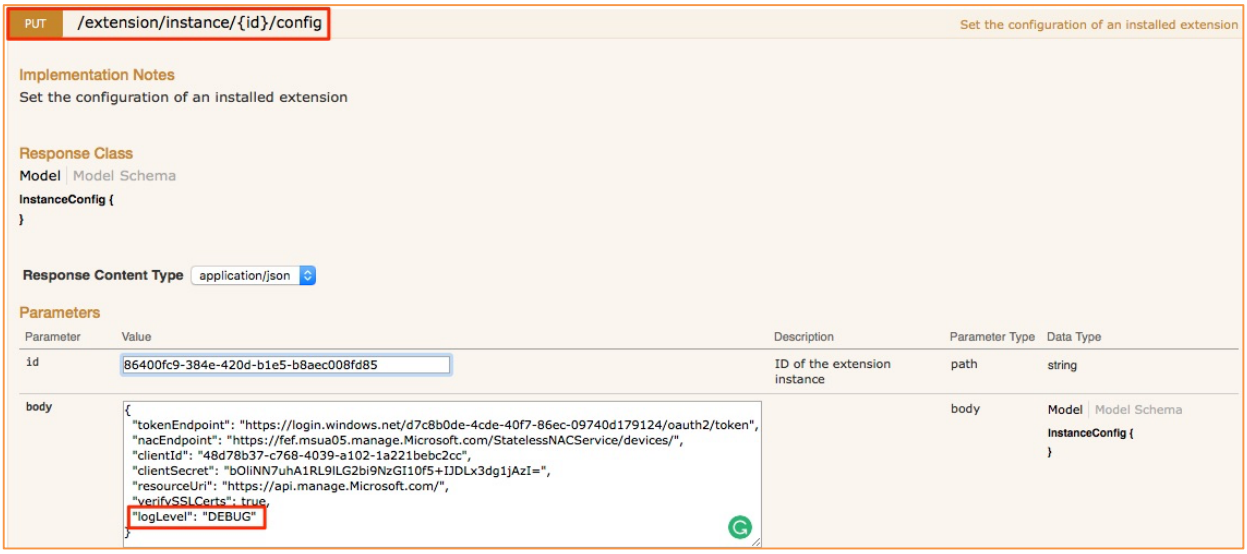


Extension Logs/Debugging

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 we previously used, adjust the "**logLevel**" to "**DEBUG**". Post this using the API Explorer as shown below.

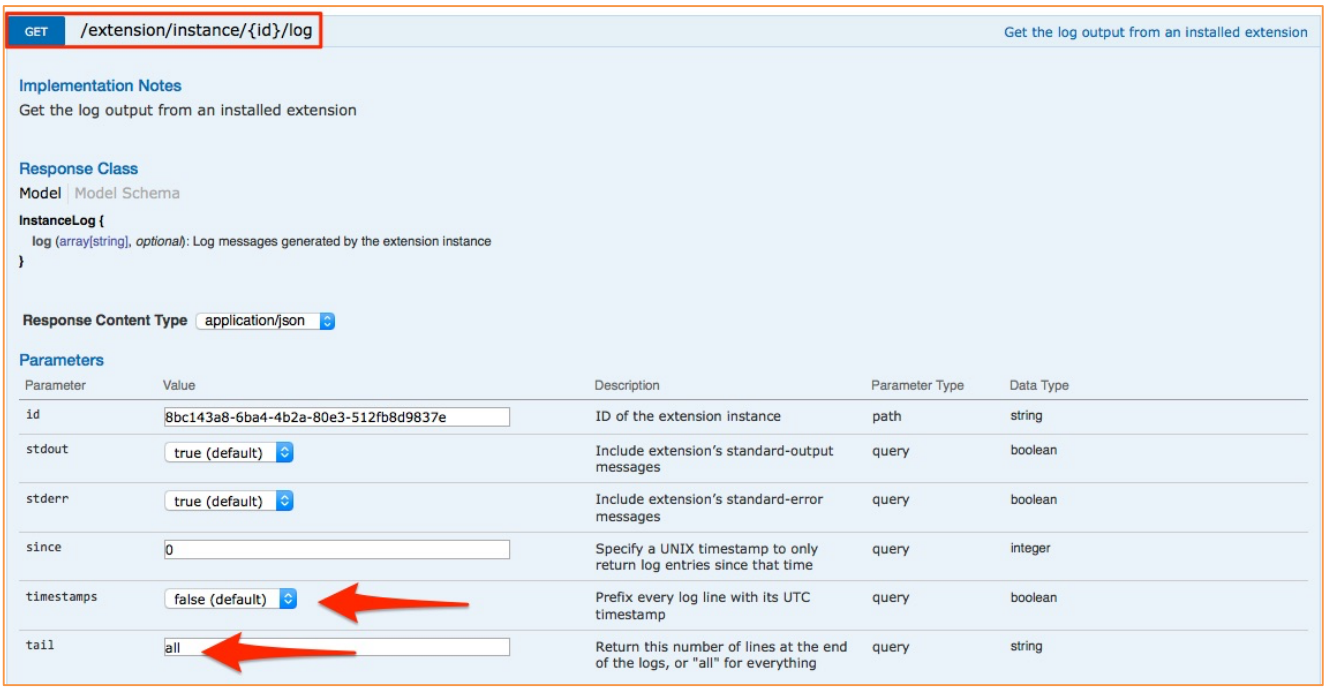
```
{
   "tokenEndpoint": "https://login.windows-ppe.net/47f09275-5bc0-4807-8aae-f35cb0341329/oauth2/token",
   "tenantId": "47f09275-5bc0-4807-8aae-ffffffffffffff"
   "clientId": "ed706345-b83f-4af5-9c96-abcabcabc",
   "clientSecret": "Q92ye91w5EGCe/cHK6fOyLu1pHI0D7ZRTnkdidy^^%$d",
   "resourceUri": "https://api.manage.microsoft.com/",
   "apiVersion":"1.1",
   "verifySSLCerts": true,
   "logLevel": "DEBUG"
}
```

Figure 38: *Turning on Debug logging on 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 39: *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 but 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 40: *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"
  ]
}
```


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

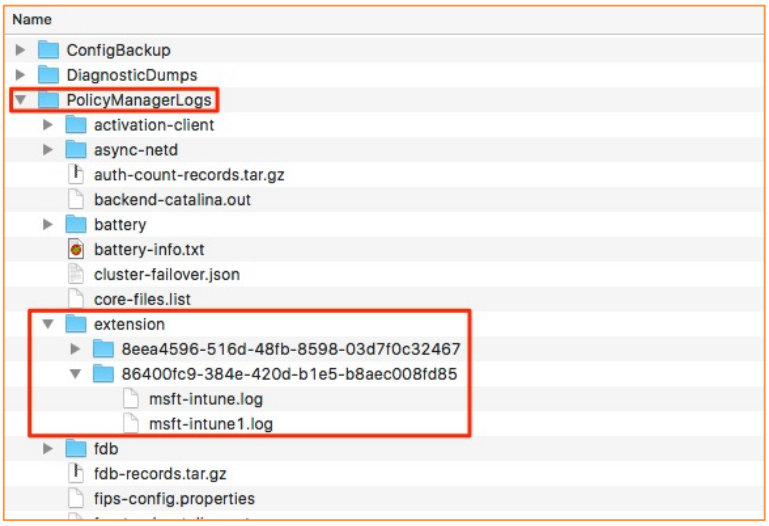
Accessing extension logs using 'Collect Logs'

In addition to viewing the logging of messages as shown above, we 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 our support team.

If there is a requirement for Aruba support 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, FATAL can also be set. Any of the levels will display the information for the selected state and lower… so 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.

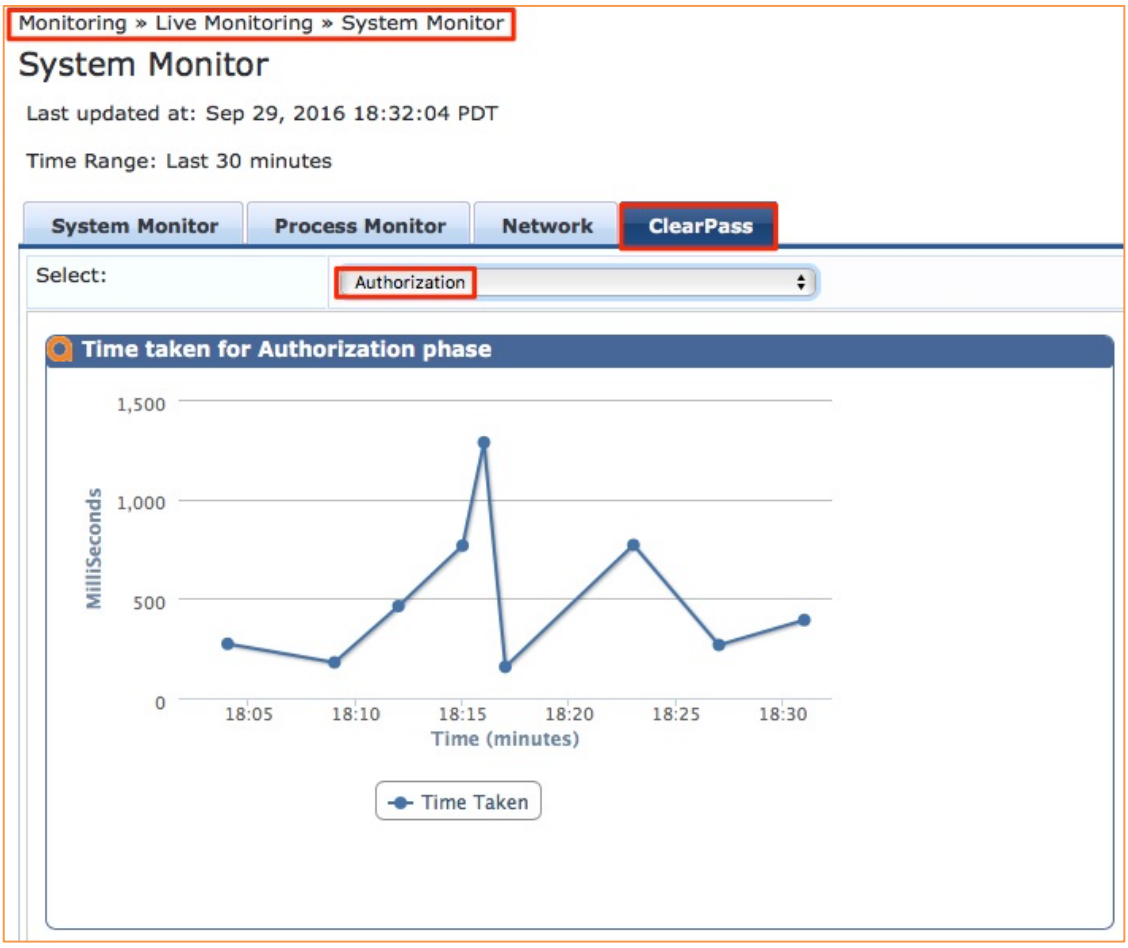
Figure 41: *Extension logs location in 'Collect Logs' diagnostic GZ file*



Monitoring authorization performance

Since we are authorizing against an external system, it is important to monitor the performance of these transactions as you setup and deploy. If you suspect there is a performance issue, ClearPass provides a way to monitor the authorization processing time. The graph below shows an example of this data, navigate to **Monitoring -> Live Monitor -> System Monitor** [click on ClearPass Tab, then select [Authorization]….

Figure 42: *Monitoring the performance of the authorization process*



ClearPass authorization throughput guidelines

Based upon scale & performance testing completed under ideal test conditions we have concluded that a ClearPass 25K Appliance is capable of sustaining 200 network authentications/second and ClearPass 5K Appliance is capable of sustaining 100 network authentications/second. The test conditions included a service categorization with an authorization check to the Microsoft Cloud based Intune MDM service, EAP-PEAP MS-CHAPv2 authentication between client and ClearPass and local user accounts in ClearPass.

a Hewlett Packard
Enterprise company

www.arubanetworks.com
1344 Crossman Avenue
Sunnyvale, CA 94089
Phone: 1-800-WIFI-LAN (+800-943-4526)
Fax 408.227.4550