Contents

What is Azure API for FHIR®?

5 minutes to read • Edit Online

Azure API for FHIR enables rapid exchange of data through Fast Healthcare Interoperability Resources (FHIR®) APIs, backed by a managed Platform-as-a Service (PaaS) offering in the cloud. It makes it easier for anyone working with health data to ingest, manage, and persist Protected Health Information PHI in the cloud:

- Managed FHIR service, provisioned in the cloud in minutes
- Enterprise-grade, FHIR®-based endpoint in Azure for data access, and storage in FHIR® format
- High performance, low latency
- Secure management of Protected Health Data (PHI) in a compliant cloud environment
- SMART on FHIR for mobile and web implementations
- Control your own data at scale with Role-Based Access Control (RBAC)
- Audit log tracking for access, creation, modification, and reads within each data store

Azure API for FHIR allows you to create and deploy a FHIR service in just minutes to leverage the elastic scale of the cloud. You pay only for the throughput and storage you need. The Azure services that power Azure API for FHIR are designed for rapid performance no matter what size datasets you're managing.

The FHIR API and compliant data store enable you to securely connect and interact with any system that utilizes FHIR APIs. Microsoft takes on the operations, maintenance, updates, and compliance requirements in the PaaS offering, so you can free up your own operational and development resources.

The following video presents an overview of Azure API for FHIR:

Leveraging the power of your data with FHIR

The healthcare industry is rapidly transforming health data to the emerging standard of FHIR® (Fast Healthcare Interoperability Resources). FHIR enables a robust, extensible data model with standardized semantics and data exchange that enables all systems using FHIR to work together. Transforming your data to FHIR allows you to quickly connect existing data sources such as the electronic health record systems or research databases. FHIR also enables the rapid exchange of data in modern implementations of mobile and web development. Most importantly, FHIR can simplify data ingestion and accelerate development with analytics and machine learning tools.

Securely manage health data in the cloud

The Azure API for FHIR allows for the exchange of data via consistent, RESTful, FHIR APIs based on the HL7 FHIR specification. Backed by a managed PaaS offering in Azure, it also provides a scalable and secure environment for the management and storage of Protected Health Information (PHI) data in the native FHIR format.

Free up your resources to innovate

You could invest resources building and running your own FHIR service, but with the Azure API for FHIR, Microsoft takes on the workload of operations, maintenance, updates and compliance requirements, allowing you to free up your own operational and development resources.

Enable interoperability with FHIR

Using the Azure API for FHIR enables to you connect with any system that leverages FHIR APIs for read, write, search, and other functions. It can be used as a powerful tool to consolidate, normalize, and apply machine learning with clinical data from electronic health records, clinician and patient dashboards, remote monitoring programs, or with databases outside of your system that have FHIR APIs.

Control Data Access at Scale

You control your data. Role-Based Access Control (RBAC) enables you to manage how your data is stored and accessed. Providing increased security and reducing administrative workload, you determine who has access to the datasets you create, based on role definitions you create for your environment.

Audit logs and tracking

Quickly track where your data is going with built-in audit logs. Track access, creation, modification, and reads within each data store.

Secure your data

Protect your PHI with unparalleled security intelligence. Your data is isolated to a unique database per API instance and protected with multi-region failover. The Azure API for FHIR implements a layered, in-depth defense and advanced threat protection for your data.

Applications for a FHIR Service

FHIR servers are key tools for interoperability of health data. The Azure API for FHIR is designed as an API and service that you can create, deploy, and begin using quickly. As the FHIR standard expands in healthcare, use cases will continue to grow, but some initial customer applications where Azure API for FHIR is useful are below:

- **Startup/IOT and App Development:** Customers developing a patient or provider centric app (mobile or web) can leverage Azure API for FHIR as a fully managed backend service. The Azure API for FHIR provides a valuable resource in that customers can managing data and exchanging data in a secure cloud environment designed for health data, leverage SMART on FHIR implementation guidelines, and enable their technology to be utilized by all provider systems (for example, most EHRs have enabled FHIR read APIs).
- **Healthcare Ecosystems:** While EHRs exist as the primary 'source of truth' in many clinical settings, it is not uncommon for providers to have multiple databases that aren't connected to one another or store data in different formats. Utilizing the Azure API for FHIR as a service that sits on top of those systems allows you to standardize data in the FHIR format. This helps to enable data exchange across multiple systems with a consistent data format.
- **Research:** Healthcare researchers will find the FHIR standard in general and the Azure API for FHIR useful as it normalizes data around a common FHIR data model and reduces the workload for machine learning and data sharing. Exchange of data via the Azure API for FHIR provides audit logs and access controls that help control the flow of data and who has access to what data types.

FHIR from Microsoft

FHIR capabilities from Microsoft are available in two configurations:

- Azure API for FHIR – A PaaS offering in Azure, easily provisioned in the Azure portal and managed by Microsoft.
- FHIR Server for Azure – an open-source project that can be deployed into your Azure subscription, available on GitHub at https://github.com/Microsoft/fhir-server.

For use cases that requires extending or customizing the FHIR server or require access the underlying services—such as the database—without going through the FHIR APIs, developers should choose the open-source FHIR Server for Azure. For implementation of a turn-key, production-ready FHIR API and backend service where persisted data should only be accessed through the FHIR API, developers should choose the Azure API for FHIR

Next Steps

To start working with the Azure API for FHIR, follow the 5-minute quickstart to deploy the Azure API for FHIR.

Deploy Azure API for FHIR

FHIR is the registered trademark of HL7 and is used with the permission of HL7.

Quickstart: Deploy Azure API for FHIR using Azure portal

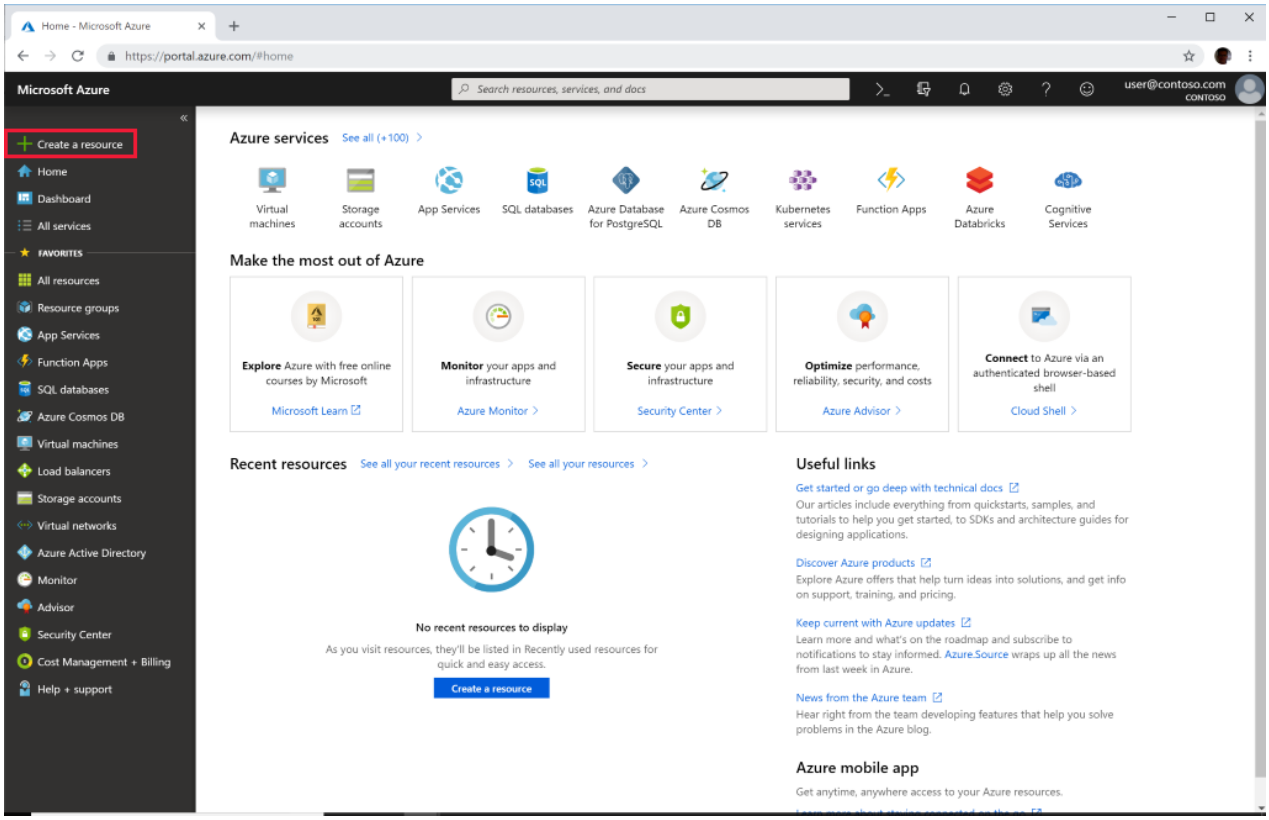
2 minutes to read • Edit Online

In this quickstart, you'll learn how to deploy Azure API for FHIR using the Azure portal.

If you don't have an Azure subscription, create a free account before you begin.

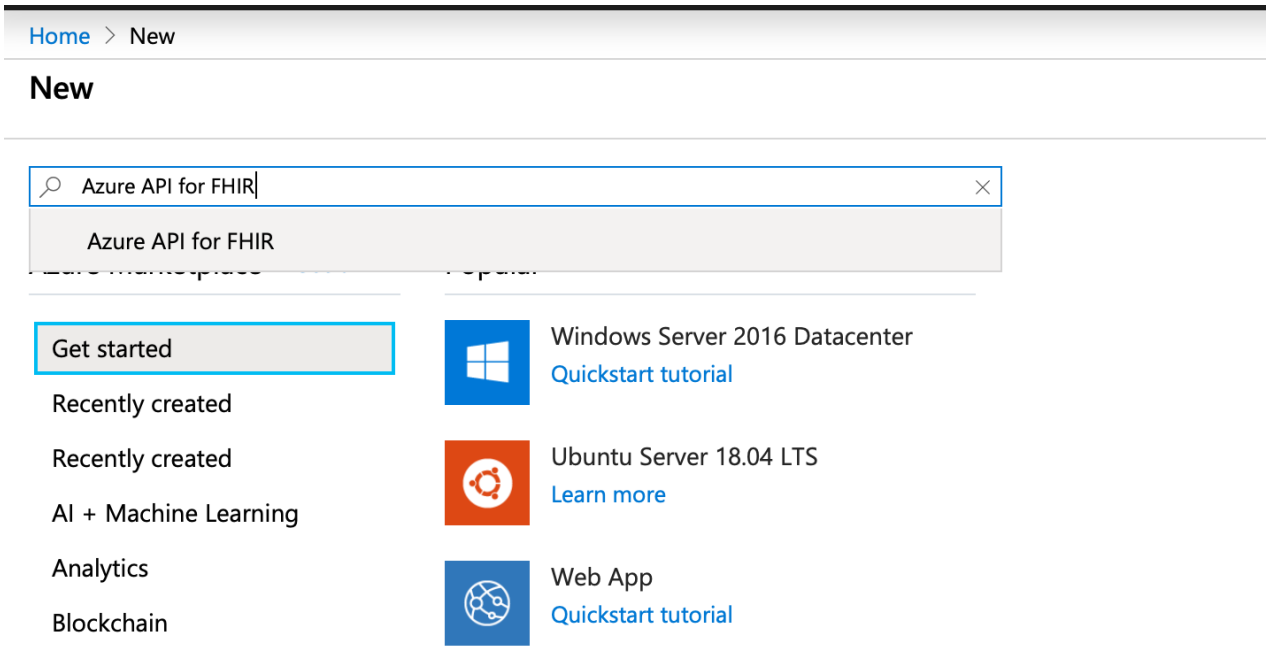
Create new resource

Open the Azure portal and click **Create a resource**



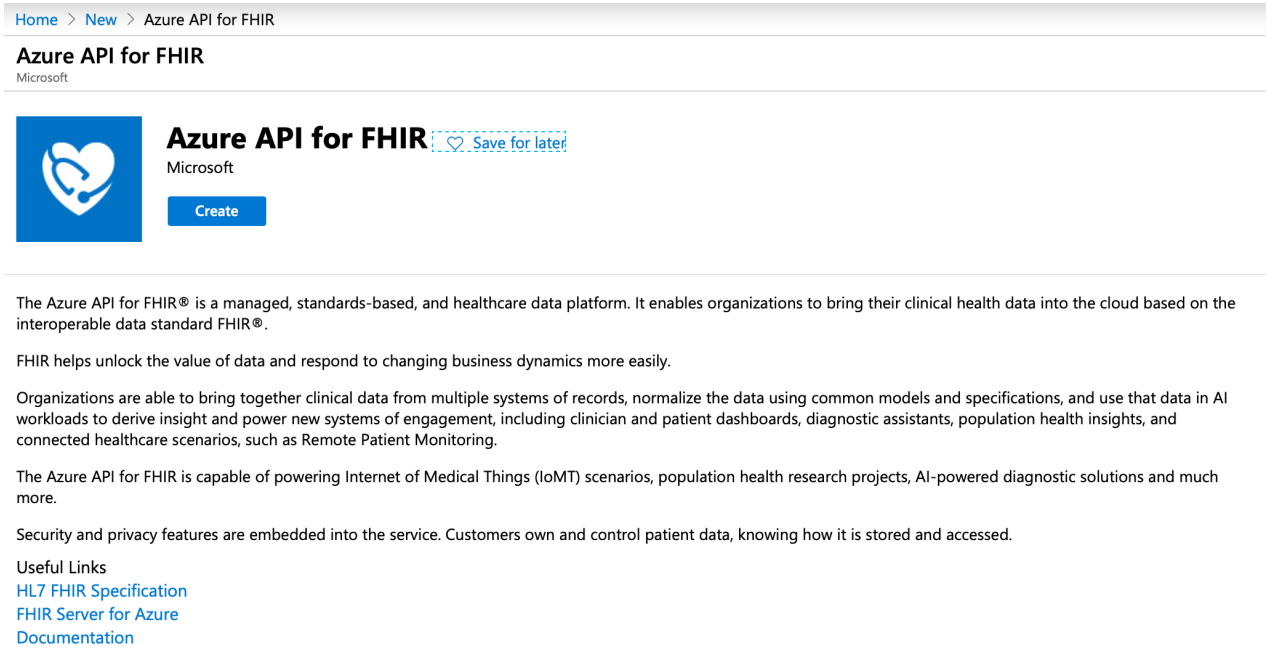
Search for Azure API for FHIR

You can find Azure API for FHIR by typing "FHIR" into the search box:



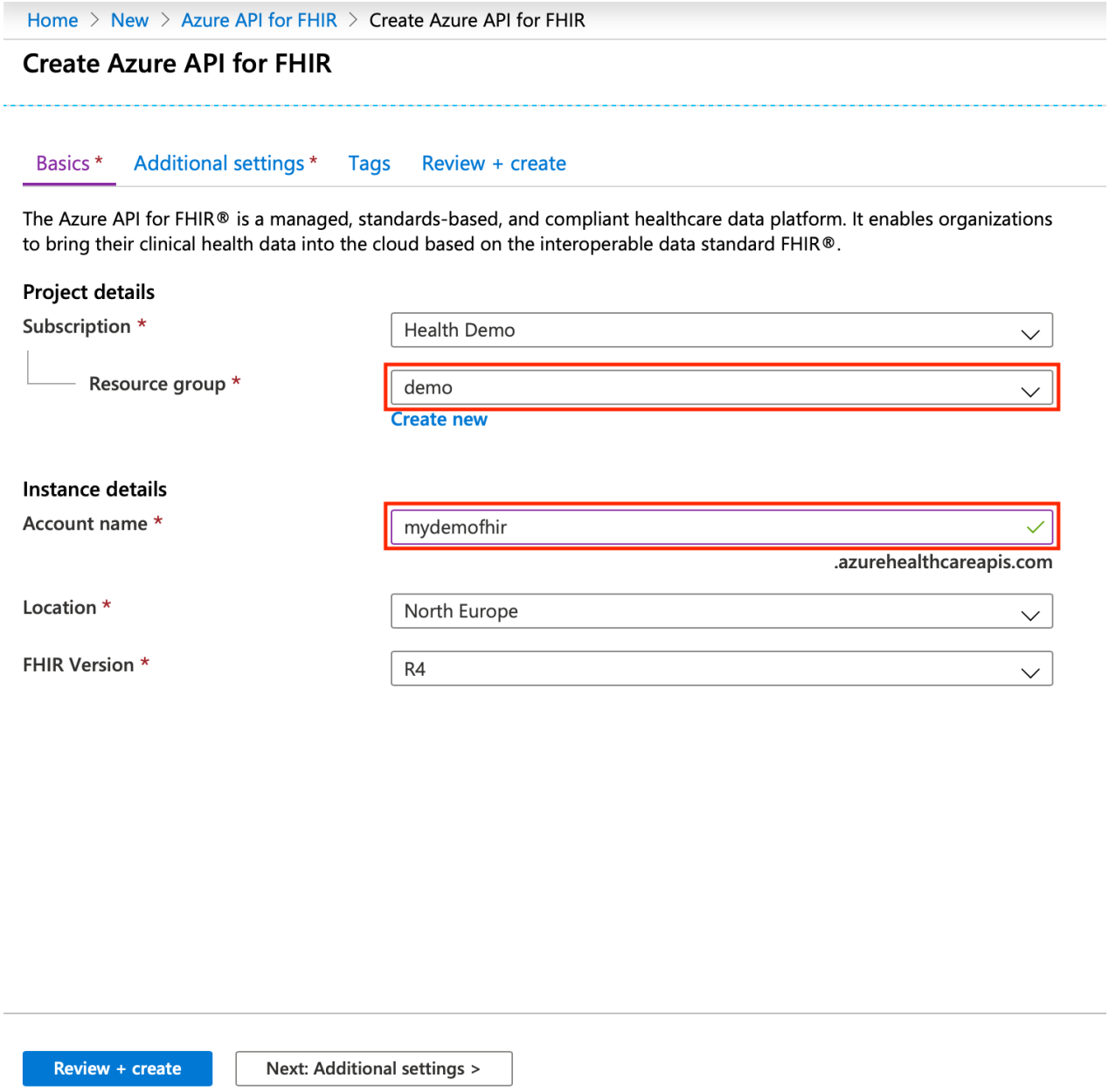
Create Azure API for FHIR account

Select **Create** to create a new Azure API for FHIR account:



Enter account details

Select an existing resource group or create a new one, choose a name for the account, and finally click **Review + create**:

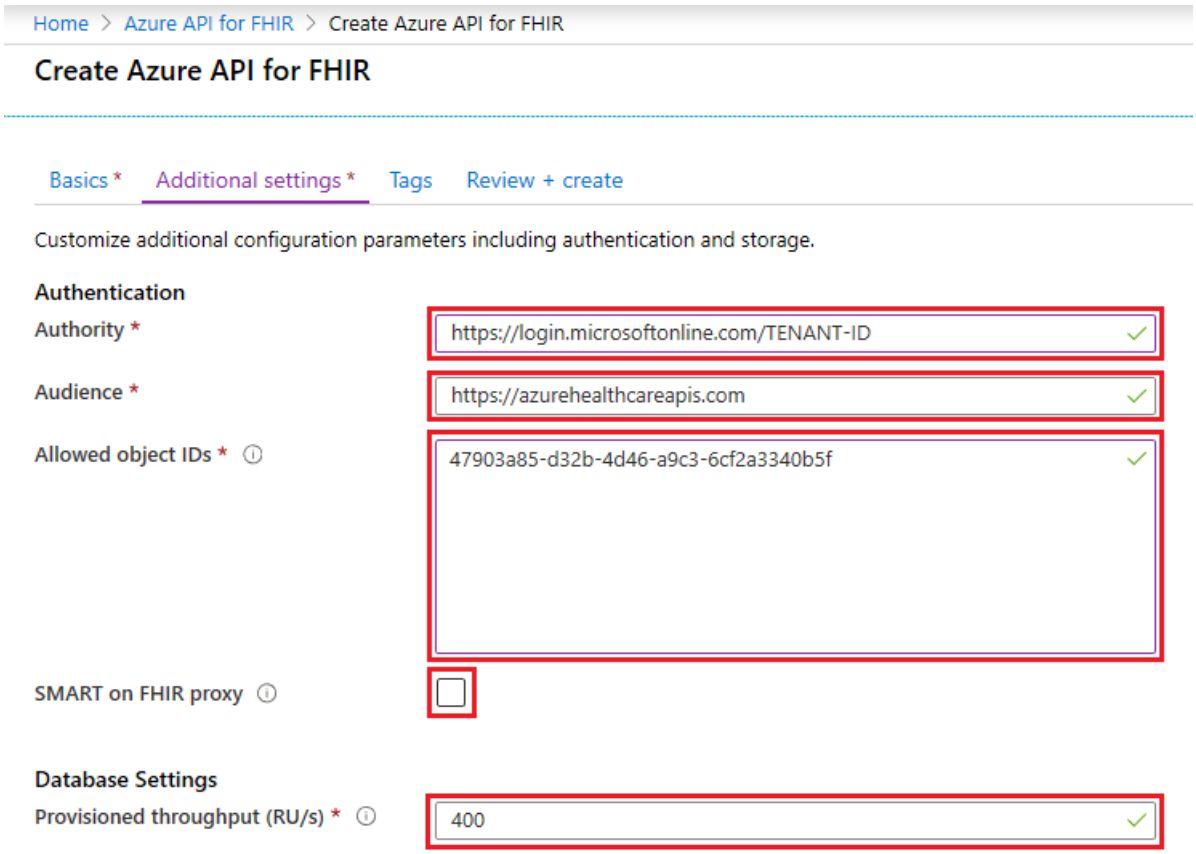


Confirm creation and await FHIR API deployment.

Additional settings

Click **Next: Additional settings** to configure the authority, audience, identity object IDs that should be allowed to access this Azure API for FHIR, enable SMART on FHIR if needed, and configure database throughput:

- **Authority:** You can specify different Azure AD tenant from the one that you are logged into as authentication authority for the service.
- **Audience:** Best practice, and the default setting, is that the audience is set to the URL of the FHIR server. You can change that here. The audience identifies the recipient that the token is intended for. In this context, it should be set to something representing the FHIR API itself.
- **Allowed object IDs:** You can specify identity object IDs that should be allowed to access this Azure API for FHIR. You can learn more on finding the object id for users and service principals in the Find identity object IDs how-to guide.
- **Smart On FHIR proxy:** You can enable SMART on FHIR proxy. For details on how to configure SMART on FHIR proxy see tutorial Azure API for FHIR SMART on FHIR proxy
- **Provisioned throughput (RU/s):** Here you can specify throughput settings for the underlying database for your Azure API for FHIR. You can change this setting later in the Database blade. For more details, please see the configure database settings page.



Fetch FHIR API capability statement

To validate that the new FHIR API account is provisioned, fetch a capability statement by pointing a browser to `https://<ACCOUNT-NAME>.azurehealthcareapis.com/metadata`.

Clean up resources

When no longer needed, you can delete the resource group, Azure API for FHIR, and all related resources. To do so, select the resource group containing the Azure API for FHIR account, select **Delete resource group**, then confirm the name of the resource group to delete.

Next steps

In this quickstart guide, you've deployed the Azure API for FHIR into your subscription. To set additional settings in your Azure API for FHIR, proceed to the additional settings how-to guide.

Additional settings in Azure API for FHIR

Quickstart: Deploy Azure API for FHIR using PowerShell

2 minutes to read • Edit Online

In this quickstart, you'll learn how to deploy Azure API for FHIR using PowerShell.

If you don't have an Azure subscription, create a free account before you begin.

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select **Try It** in the upper-right corner of a code block. Selecting **Try It** doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl**+**Shift**+**V** on Windows and Linux or by selecting **Cmd**+**Shift**+**V** on macOS.
4. Select **Enter** to run the code.

> **NOTE**
> This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

Register the Azure API for FHIR resource provider

If the `Microsoft.HealthcareApis` resource provider is not already registered for your subscription, you can register it with:

```
Register-AzResourceProvider -ProviderNamespace Microsoft.HealthcareApis
```

Locate your identity object ID

Object ID values are guids that correspond to the object IDs of specific Azure Active Directory users or service principals in the directory associated with the subscription. If you would like to know the object ID of a specific user, you can find it with a command like:

```
$(Get-AzureADUser -Filter "UserPrincipalName eq 'myuser@consoso.com'").ObjectId
```

Read the how-to guide on finding identity object IDs for more details.

Create Azure resource group

```
New-AzResourceGroup -Name "myResourceGroupName" -Location westus2
```

Deploy Azure API for FHIR

```
New-AzHealthcareApisService -Name nameoffhirservice -ResourceGroupName myResourceGroupName -Location westus2 -
Kind fhir-R4 -AccessPolicyObjectId "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
```

where `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx` is the identity object ID for a user or service principal that you would like to have access to the FHIR API.

Fetch capability statement

You'll be able to validate that the Azure API for FHIR account is running by fetching a FHIR capability statement:

```
$metadata = Invoke-WebRequest -Uri "https://nameoffhirservice.azurehealthcareapis.com/metadata"
$metadata.RawContent
```

Clean up resources

If you're not going to continue to use this application, delete the resource group with the following steps:

```
Remove-AzResourceGroup -Name myResourceGroupName
```

Next steps

In this quickstart guide, you've deployed the Azure API for FHIR into your subscription. To set additional settings in your Azure API for FHIR, proceed to the additional settings how-to guide.

Additional settings in Azure API for FHIR

Quickstart: Deploy Azure API for FHIR using Azure CLI

2 minutes to read • Edit Online

In this quickstart, you'll learn how to deploy Azure API for FHIR in Azure using the Azure CLI.

If you don't have an Azure subscription, create a free account before you begin.

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select **Try It** in the upper-right corner of a code block. Selecting **Try It** doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl**+**Shift**+**V** on Windows and Linux or by selecting **Cmd**+**Shift**+**V** on macOS.
4. Select **Enter** to run the code.

Add HealthcareAPIs extension

```
az extension add --name healthcareapis
```

Get a list of commands for HealthcareAPIs:

```
az healthcareapis --help
```

Locate your identity object ID

Object ID values are guids that correspond to the object IDs of specific Azure Active Directory users or service

principals in the directory associated with the subscription. If you would like to know the object ID of a specific user, you can find it with a command like:

```
az ad user show --id myuser@consoso.com | jq -r .objectId
```

Read the how-to guide on finding identity object IDs for more details.

Create Azure Resource Group

Pick a name for the resource group that will contain the Azure API for FHIR and create it:

```
az group create --name "myResourceGroup" --location westus2
```

Deploy the Azure API for FHIR

```
az healthcareapis create --resource-group myResourceGroup --name nameoffhiraccount --kind fhir-r4 --location
westus2 --access-policies-object-id "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
```

where `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx` is the identity object ID for a user or service principal that you would like to have access to the FHIR API.

Fetch FHIR API capability statement

Obtain a capability statement from the FHIR API with:

```
curl --url "https://nameoffhiraccount.azurehealthcareapis.com/metadata"
```

Clean up resources

If you're not going to continue to use this application, delete the resource group with the following steps:

```
az group delete --name "myResourceGroup"
```

Next steps

In this quickstart guide, you've deployed the Azure API for FHIR into your subscription. To set additional settings in your Azure API for FHIR, proceed to the additional settings how-to guide.

Additional settings in Azure API for FHIR

Quickstart: Deploy Open Source FHIR server using Azure portal

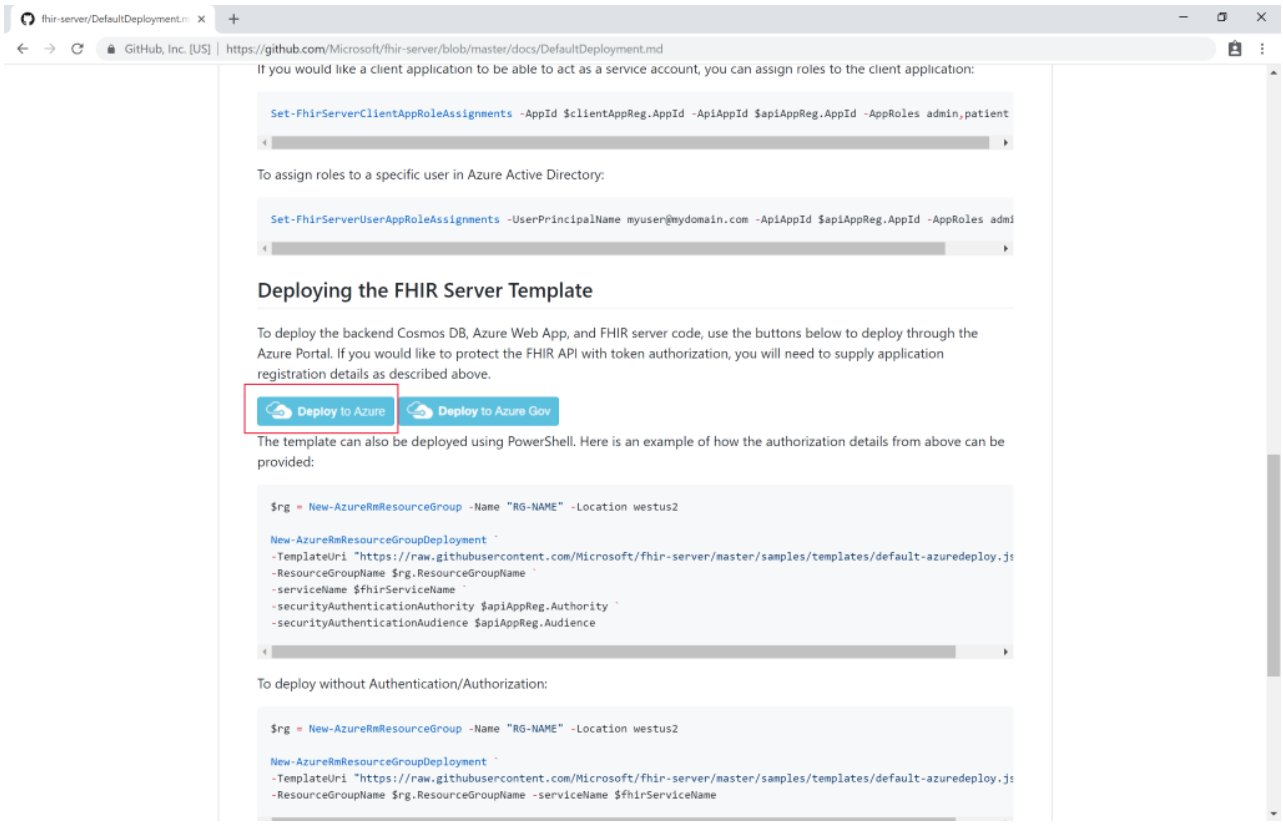
2 minutes to read • Edit Online

In this quickstart, you'll learn how to deploy an Open Source FHIR Server in Azure using the Azure portal. We will use easy deployment links in the Open Source repository

If you don't have an Azure subscription, create a free account before you begin.

GitHub Open Source repository

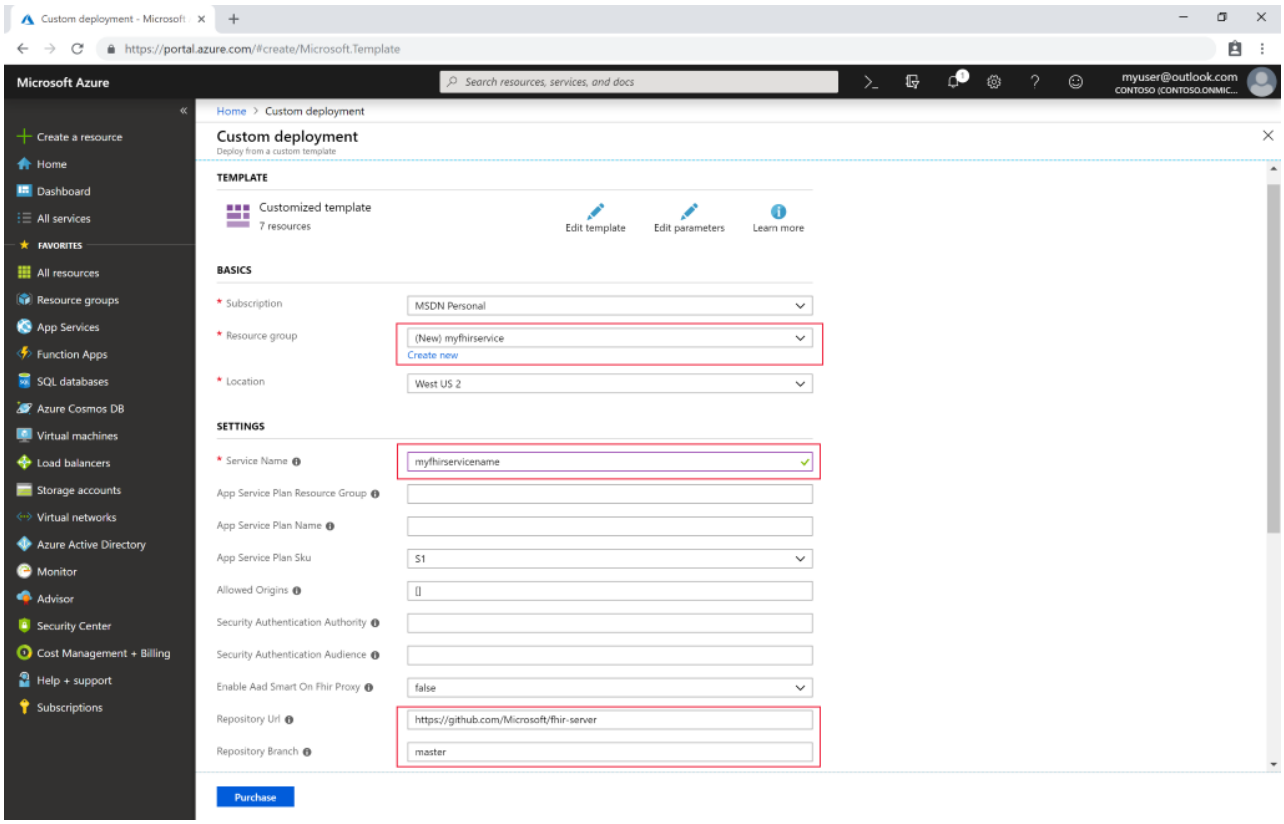
Navigate to the GitHub deployment page and locate the "Deploy to Azure" buttons:



Click the deployment button and the Azure portal opens.

Fill in deployment parameters

Choose to create a new resource group and give it a name. Only other required parameter is a name for the service.



Notice that the deployment will pull the source code directly from the open-source repository on GitHub. If you have forked the repository, you can point to your own for and a specific branch.

After filling in the details, you can start the deployment.

Validate FHIR Server is running

Once the deployment is complete, you can point your browser to `https://SERVICENAME.azurewebsites.net/metadata` to obtain a capability statement. It will take a minute or so for the server to respond the first time.

Clean up resources

When no longer needed, you can delete the resource group and all related resources. To do so, select the resource group containing the provisioned resources, select **Delete resource group**, then confirm the name of the resource group to delete.

Next steps

In this tutorial, you've deployed the Microsoft Open Source FHIR Server for Azure into your subscription. To learn how to access the FHIR API using Postman, proceed to the Postman tutorial.

Access FHIR API using Postman

Quickstart: Deploy Open Source FHIR server using PowerShell

2 minutes to read • Edit Online

In this quickstart, learn how to deploy the Open Source Microsoft FHIR server for Azure Using PowerShell.

If you don't have an Azure subscription, create a free account before you begin.

> **NOTE**
> This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select **Try It** in the upper-right corner of a code block. Selecting **Try It** doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl**+**Shift**+**V** on Windows and Linux or by selecting **Cmd**+**Shift**+**V** on macOS.
4. Select **Enter** to run the code.

Create a resource group

Pick a name for the resource group that will contain the provisioned resources and create it:

```
$fhirServiceName = "MyFhirService"
$rg = New-AzResourceGroup -Name $fhirServiceName -Location westus2
```

Deploy the FHIR server template

The Microsoft FHIR Server for Azure GitHub Repository contains a template that will deploy all necessary resources. Deploy it with:

```
New-AzResourceGroupDeployment -TemplateUri https://raw.githubusercontent.com/Microsoft/fhir-server/master/samples/templates/default-azuredeploy.json -ResourceGroupName $rg.ResourceGroupName -serviceName $fhirServiceName
```

Verify FHIR server is running

```
$metadataUrl = "https://" + $fhirServiceName + ".azurewebsites.net/metadata"
$metadata = Invoke-WebRequest -Uri $metadataUrl
$metadata.RawContent
```

It will take a minute or so for the server to respond the first time.

Clean up resources

If you're not going to continue to use this application, delete the resource group with the following steps:

```
Remove-AzResourceGroup -Name $rg.ResourceGroupName
```

Next steps

In this tutorial, you've deployed the Microsoft Open Source FHIR Server for Azure into your subscription. To learn how to access the FHIR API using Postman, proceed to the Postman tutorial.

Access FHIR API using Postman

Quickstart: Deploy Open Source FHIR server using Azure CLI

2 minutes to read • Edit Online

In this quickstart, you'll learn how to deploy an Open Source FHIR® server in Azure using the Azure CLI.

If you don't have an Azure subscription, create a free account before you begin.

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select **Try It** in the upper-right corner of a code block. Selecting **Try It** doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl**+**Shift**+**V** on Windows and Linux or by selecting **Cmd**+**Shift**+**V** on macOS.
4. Select **Enter** to run the code.

Create resource group

Pick a name for the resource group that will contain the provisioned resources and create it:

```
servicename="myfhirservice"
az group create --name $servicename --location westus2
```

Deploy template

The Microsoft FHIR Server for Azure GitHub Repository contains a template that will deploy all necessary resources. Deploy it with:

```
az group deployment create -g $servicename --template-uri https://raw.githubusercontent.com/Microsoft/fhir-server/master/samples/templates/default-azuredeploy.json --parameters serviceName=$servicename
```

Verify FHIR server is running

Obtain a capability statement from the FHIR server with:

```
metadataurl="https://${servicename}.azurewebsites.net/metadata"
curl --url $metadataurl
```

It will take a minute or so for the server to respond the first time.

Clean up resources

If you're not going to continue to use this application, delete the resource group with the following steps:

```
az group delete --name $servicename
```

Next steps

In this tutorial, you've deployed the Microsoft Open Source FHIR Server for Azure into your subscription. To learn how to access the FHIR API using Postman, proceed to the Postman tutorial.

Access FHIR API using Postman

Deploy JavaScript app to read data from FHIR service

2 minutes to read • Edit Online

In this tutorial, you will deploy a small JavaScript app, which reads data from a FHIR service. The steps in this tutorial are:

1. Deploy a FHIR server
2. Register a public client application
3. Test access to the application
4. Create a web application that reads this FHIR data

Prerequisites

Before starting this set of tutorials, you will need the following items:

1. An Azure subscription
2. An Azure Active Directory tenant
3. Postman installed

> **NOTE**
> For this tutorial, the FHIR service, Azure AD application, and Azure AD users are all in the same Azure AD tenant. If this is not the case, you can still follow along with this tutorial, but may need to dive into some of the referenced documents to do additional steps.

Deploy Azure API for FHIR

The first step in the tutorial is to get your Azure API for FHIR setup correctly.

1. Deploy the Azure API for FHIR
2. Once you have your Azure API for FHIR deployed, configure the CORS settings by going to your Azure API for FHIR and selecting CORS.
   a. Set **Origins** to *
   b. Set **Headers** to *
   c. Under **Methods**, choose **Select all**
   d. Set the **Max age** to **600**

Next Steps

Now that you have your Azure API for FHIR deployed, you are ready to register a public client application.

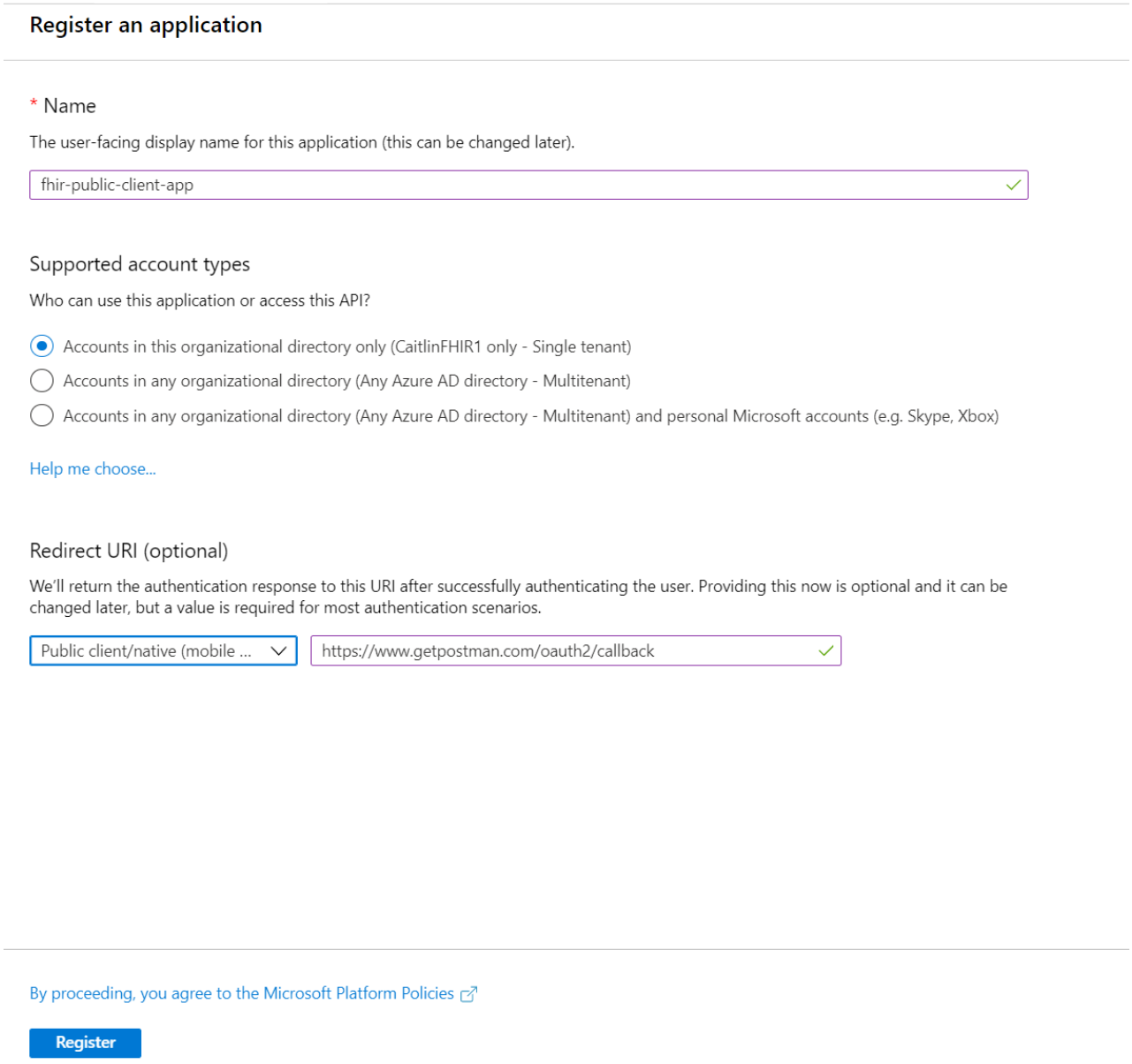
Register public client application

Client application registration

2 minutes to read • Edit Online

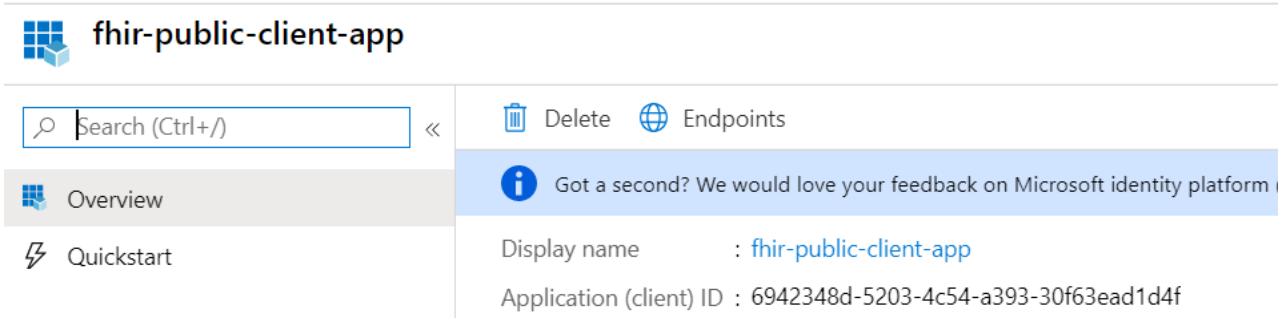
In the previous tutorial, you deployed and set up your Azure API for FHIR. Now that you have your Azure API for FHIR setup, we will register a public client application. You can read through the full register a public client app how-to guide for more details or troubleshooting, but we have called out the major steps for this tutorial below.

1. Navigate to Azure Active Directory
2. Select **App Registration** --> **New Registration**
3. Name your application and set up the redirect URI to https://www.getpostman.com/oauth2/callback

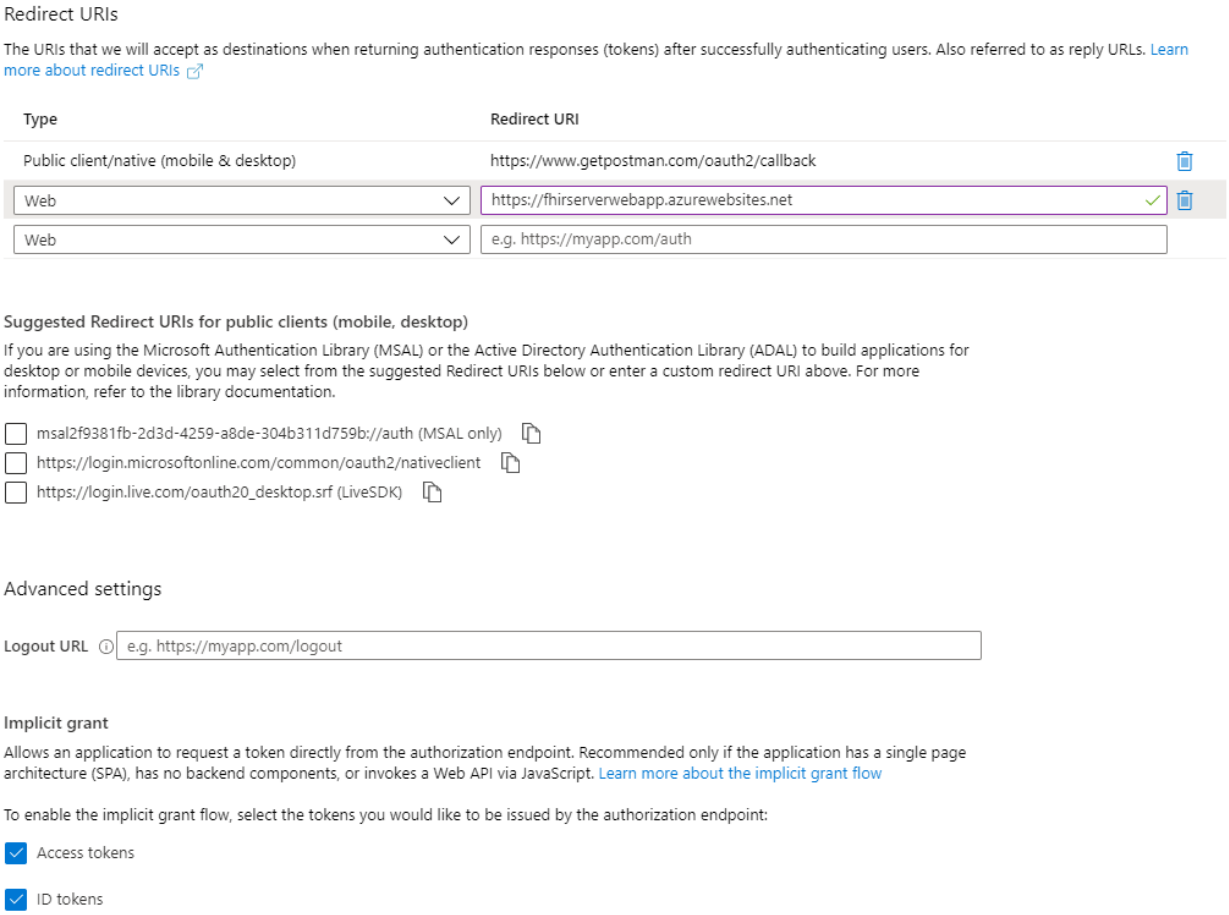


Client application settings

Once your client application is registered, copy the Application (client) ID from the Overview Page. You will need this value later when accessing the client.



Next, set the correct authentication options. Select **Authentication** from the left-hand side. Check the **Access Token** and **ID token** boxes. You can also setup the redirect URI in preparation for when you create your web application in the fourth part of this tutorial. To do this, add https://<WEB-APP-NAME>.azurewebsites.net to the redirect URI list. If you choose a different name during the step where you write your web app, you will need to come back and update this.



Now that you have setup the correct authentication, set the API permissions.

1. Select **API permissions** and click **Add a permission**
2. Under **APIs my organization uses**, search for Azure Healthcare APIs
3. Select **user_impersonation** and click **add permissions**

Next Steps

You now have a public client application. In the next tutorial, we will walk through testing and gaining access to this application through Postman.

Test client application in Postman

Testing the FHIR API

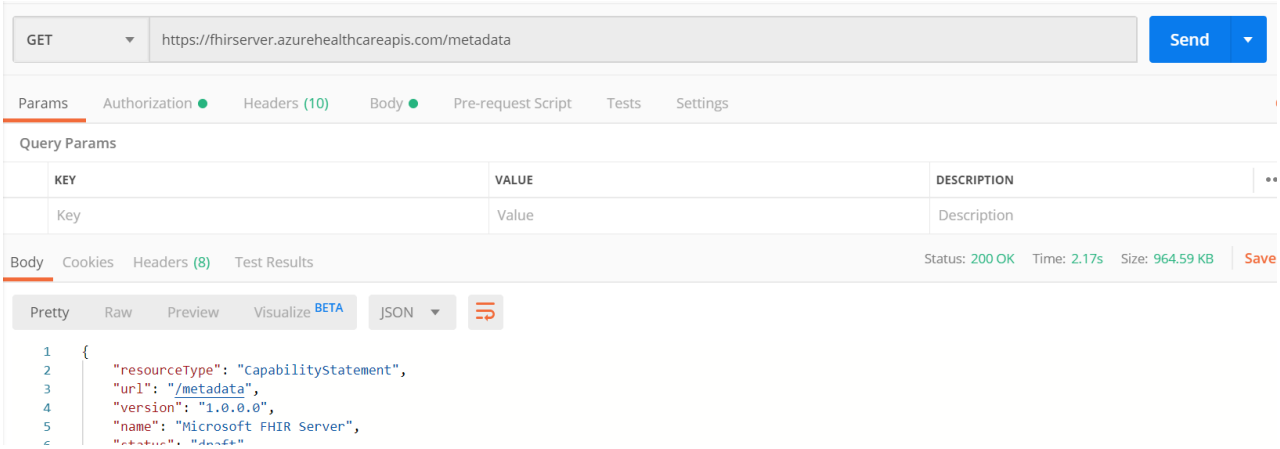
2 minutes to read • Edit Online

In the previous two steps, you deployed the Azure API for FHIR and registered your client application. You are now ready to test that your Azure API for FHIR is set up with your client application.

Retrieve capability statement

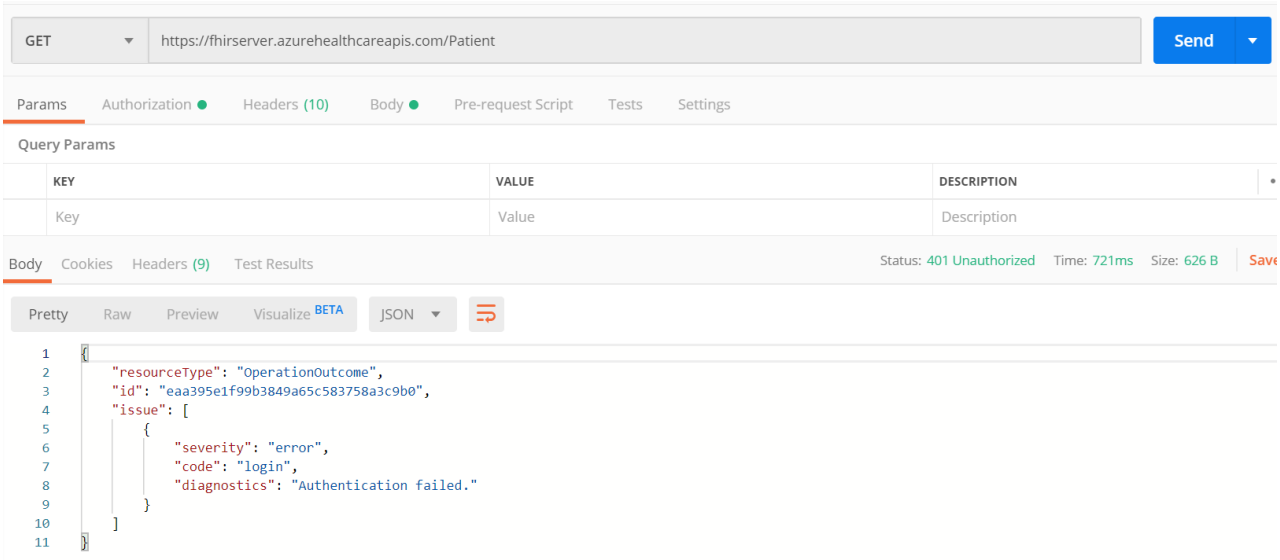
First we will get the capability statement for your Azure API for FHIR.

1. Open Postman
2. Retrieve the capability statement by doing GET https://<FHIR-SERVER-NAME>.azurehealthcareapis.com/metadata. In the image below the FHIR server name is **fhirserver**.



Next we will attempt to retrieve a patient. To retrieve a patient, enter GET https://<FHIR-SERVER-NAME>.azurehealthcareapis.com/Patient. You will receive a 401 Unauthorized error. This error is because you haven't proven that you should have access to patient data.

Get patient from FHIR server



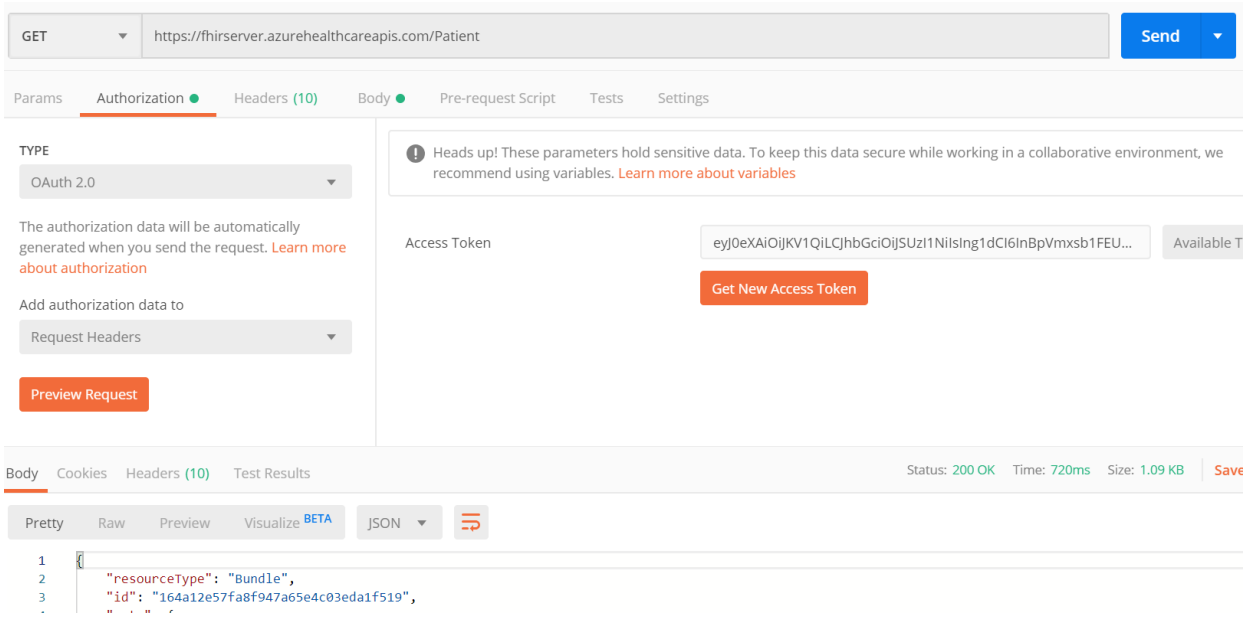
In order to gain access, you need an access token.

1. In Postman, select **Authorization** and set the Type to **OAuth2.0**

2. Select **Get New Access Token**
3. Fill in the fields and select **Request Token**. Below you can see the values for each field for this tutorial.

FIELD	VALUE
Token Name	A name for your token
Grant Type	Authorization Code
Callback URL	https://www.getpostman.com/oauth2/callback
Auth URL	https://login.microsoftonline.com/<AZURE-AD-TENANT-ID>/oauth2/?resource=https://<FHIR-SERVER-NAME>.azurehealthcareapis.com
Access Token URL	https://login.microsoftonline.com/<AZURE-AD-TENANT-ID>/oauth2/token
Client ID	The client ID that you copied during the previous steps
Client Secret	<BLANK>
Scope	<BLANK>
State	1234
Client Authentication	Send client credentials in body

4. Sign in with your credentials and select **Accept**
5. Scroll down on the result and select **Use Token**
6. Select **Send** again at the top and this time you should get a result

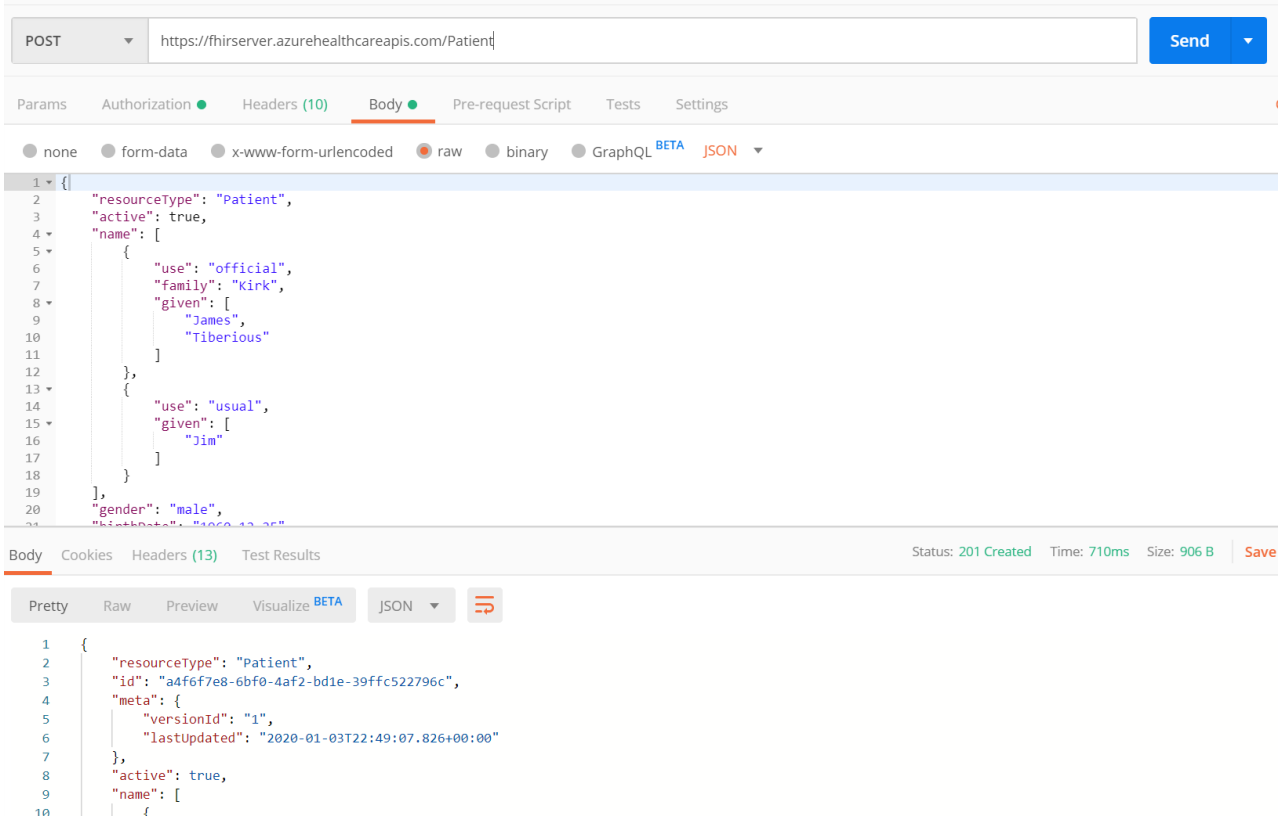


Post patient into FHIR server

Now you have access, you can create a new patient. Here is a sample of a simple patient you can add into your FHIR server. Enter the code below into the **Body** section of Postman.

```
{
 "resourceType": "Patient",
 "active": true,
 "name": [
  {
   "use": "official",
   "family": "Kirk",
   "given": [
    "James",
    "Tiberious"
   ]
  },
  {
   "use": "usual",
   "given": [
    "Jim"
   ]
  }
 ],
 "gender": "male",
 "birthDate": "1960-12-25"
}
```

This POST will create a new patient in your FHIR server with the name James Tiberious Kirk.



If you do the GET step above to retrieve a patient again, you will see James Tiberious Kirk listed in the output.

Troubleshooting access issues

If you ran into issues during any of these steps, review the documents we have put together on Azure Active Directory and the Azure API for FHIR.

- Azure AD and Azure API for FHIR - This document outlines some of the basic principles of Azure Active Directory and how it interacts with the Azure API for FHIR.
- Access token validation - This how-to guide gives more specific details on access token validation and steps to take to resolve access issues.

Next Steps

Now that you can successfully connect to your client application, you are ready to write your web application.

Write a web application

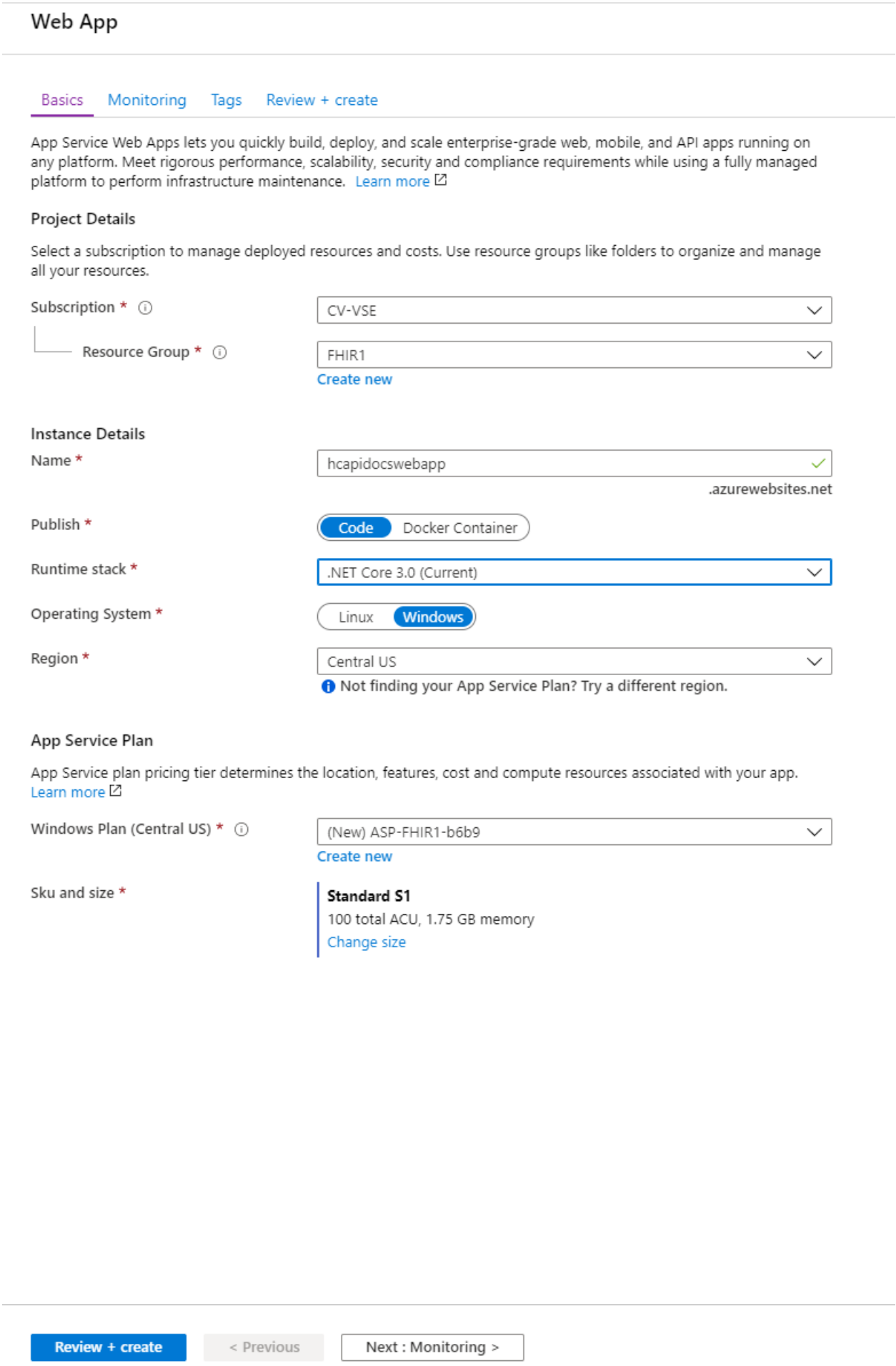
Write Azure web application to read FHIR data

2 minutes to read • Edit Online

Now that you are able to connect to your FHIR server and POST data, you are ready to write a web application that will read FHIR data. In this final step of the tutorial, we will walk through writing and accessing the web application.

Create web application

In Azure, select **Create a resource** and select **Web App**. Make sure to name your web application whatever you specified in the redirect URI for your client application or go back and update the redirect URI with the new name.



Once the web application is available, **Go to resource**. Select **App Service Editor (Preview)** under Development Tools on the right and then select **Go**. Selecting Go will open up the App Service Editor. Right click in the grey space under *Explore* and create a new file called **index.html**.

Below is the code that you can input into **index.html**. You will need to update the following items:

- **clientId** - Update with your client application ID. This ID will be the same ID you pulled when retrieving your token
- **authority** - Update with your Azure AD tenant ID
- **FHIRendpoint** - Update the FHIRendpoint to have your FHIR service name
- **scopes** - Update to reflect the full URL for your audience

```
<!DOCTYPE html>
<html>

<head>
    <title>FHIR Patient browser sample app</title>
    <script src="https://secure.aadcdn.microsoftonline-p.com/lib/1.0.0/js/msal.js"></script>
</head>

<body>
    <div class="leftContainer">
        <p id="WelcomeMessage">Welcome to the FHIR Patient browsing sample Application</p>
        <button id="SignIn" onclick="signIn()">Sign In</button>
    </div>

    <div id="patientTable">
    </div>

    <script>
        var msalConfig = {
            auth: {
                clientId: '<CLIENT-ID>',
                authority: "https://login.microsoftonline.com/<AZURE-AD-TENANT-ID>"
            },
            cache: {
                cacheLocation: "localStorage",
                storeAuthStateInCookie: true
            }
        }

        var FHIRConfig = {
            FHIRendpoint: "https://<FHIR-SERVER-NAME>.azurehealthcareapis.com"
        }
        var requestObj = {
            scopes: ["https://<FHIR-SERVER-NAME>.azurehealthcareapis.com/user_impersonation"]
        }

        function authRedirectCallBack(error, response) {
            if (error) {
                console.log(error);
            } else {
                if (response.tokenType === "access_token") {
                    callFHIRServer(FHIRConfig.FHIRendpoint + '/Patient', 'GET', null, response.accessToken,
FHIRCallback);
                }
            }
        }

        var myMSALObj = new Msal.UserAgentApplication(msalConfig);
        myMSALObj.handleRedirectCallback(authRedirectCallBack);

        function signIn() {
            myMSALObj.loginPopup(requestObj).then(function (loginResponse) {
                showWelcomeMessage();
                acquireTokenPopupAndCallFHIRServer();
            }).catch(function (error) {
                console.log(error);
            })
```

```
        }

        function showWelcomeMessage() {
            var divWelcome = document.getElementById('WelcomeMessage');
            divWelcome.innerHTML = "Welcome " + myMSALObj.getAccount().userName + " to FHIR Patient Browsing
App";
            var loginbutton = document.getElementById('SignIn');
            loginbutton.innerHTML = 'Sign Out';
            loginbutton.setAttribute('onclick', 'signOut()')
        }

        function signOut() {
            myMSALObj.logout();
        }

        function acquireTokenPopupAndCallFHIRServer() {
            myMSALObj.acquireTokenSilent(requestObj).then(function (tokenResponse) {
                callFHIRServer(FHIRConfig.FHIRendpoint + '/Patient', 'GET', null, tokenResponse.accessToken,
FHIRCallback);
            }).catch(function (error) {
                console.log(error);
                if (requiresInteraction(error.errorCode)) {
                    myMSALObj.acquireTokenPopup(requestObj).then(function (tokenResponse) {
                        callFHIRServer(FHIRConfig.FHIRendpoint + '/Patient', 'GET', null,
tokenResponse.accessToken, FHIRCallback);
                    }).catch(function (error) {
                        console.log(error);
                    })
                }
            });
        }

        function callFHIRServer(theUrl, method, message, accessToken, callBack) {
            var xmlHttp = new XMLHttpRequest();
            xmlHttp.onreadystatechange = function () {
                if (this.readyState == 4 && this.status == 200)
                    callBack(JSON.parse(this.responseText));
            }
            xmlHttp.open(method, theUrl, true);
            xmlHttp.setRequestHeader("Content-Type", "application/json;charset=UTF-8");
            xmlHttp.setRequestHeader('Authorization', 'Bearer ' + accessToken);
            xmlHttp.send(message);
        }

        function FHIRCallback(data) {
            patientListHtml = '<ol>';
            data.entry.forEach(function(e) {
                patientListHtml += '<li>' + e.resource.name[0].family + ', ' + e.resource.name[0].given + ' ('
+ e.resource.id + ')';
            });
            patientListHtml += '</ol>';
            document.getElementById("patientTable").innerHTML = patientListHtml;
        }
    </script>
</body>

</html>
```

From here, you can go back to your web application resource and open the URL found on the Overview page. Log in to see the patient James Tiberious Kirk that you previously created.

Next Steps

You have successfully deployed the Azure API for FHIR, registered a public client application, tested access, and created a small web application. Check out the Azure API for FHIR supported features as a next step.

Supported Features

Access Azure API for FHIR with Postman

3 minutes to read • Edit Online

A client application would access an FHIR API through a REST API. You may also want to interact directly with the FHIR server as you build applications, for example, for debugging purposes. In this tutorial, we will walk through the steps needed to use Postman to access an FHIR server. Postman is a tool often used for debugging when building applications that access APIs.

Prerequisites

- A FHIR endpoint in Azure. You can set that up using the managed Azure API for FHIR or the Open Source FHIR server for Azure. Set up the managed Azure API for FHIR using Azure portal, PowerShell, or Azure CLI.
- Postman installed. You can get it from https://www.getpostman.com

FHIR server and authentication details

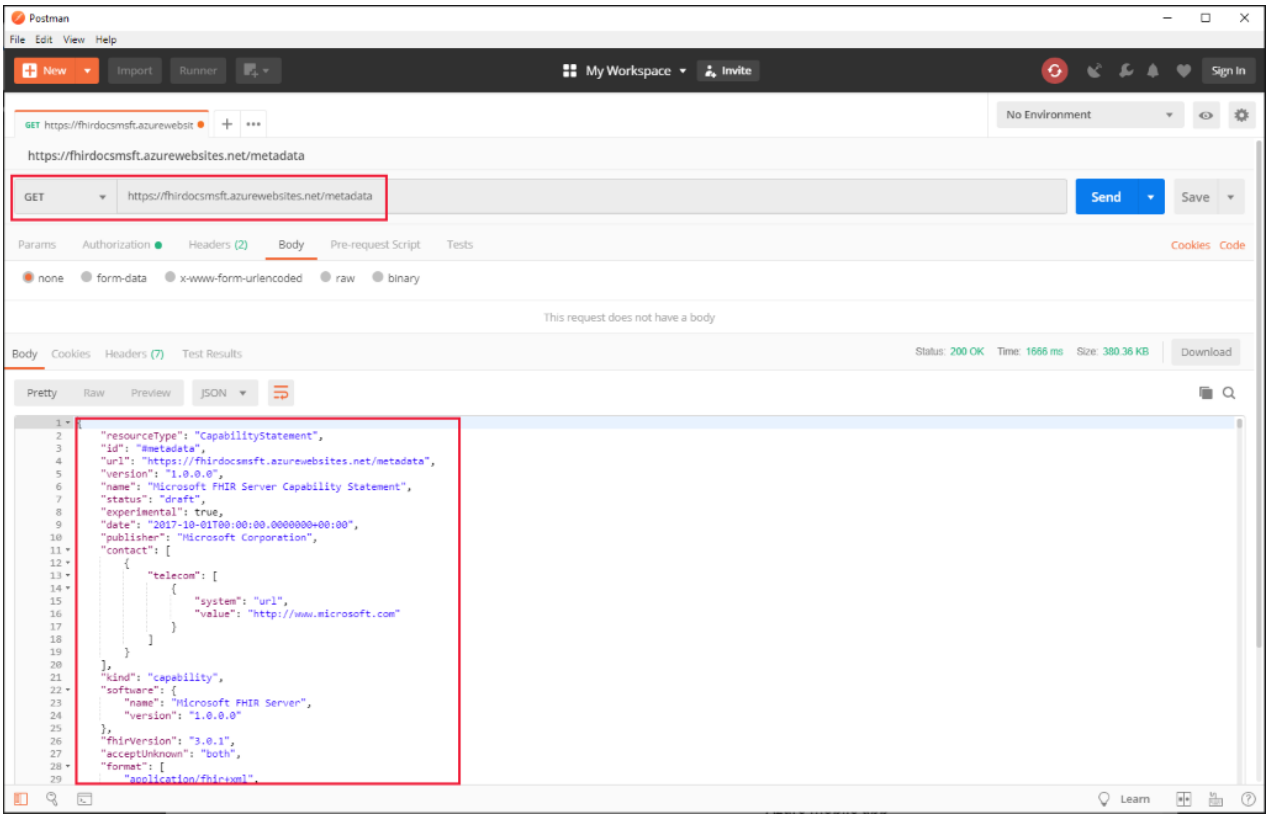
In order to use Postman, the following details are needed:

- Your FHIR server URL, for example `https://MYACCOUNT.azurehealthcareapis.com`
- The identity provider `Authority` for your FHIR server, for example, `https://login.microsoftonline.com/{TENANT-ID}`
- The configured `audience`. This is is usually the URL of the FHIR server, e.g. `https://MYACCOUNT.azurehealthcareapis.com` or just `https://azurehealthcareapis.com`.
- The `client_id` (or application ID) of the client application you will be using to access the FHIR service.
- The `client_secret` (or application secret) of the client application.

Finally, you should check that `https://www.getpostman.com/oauth2/callback` is a registered reply URL for your client application.

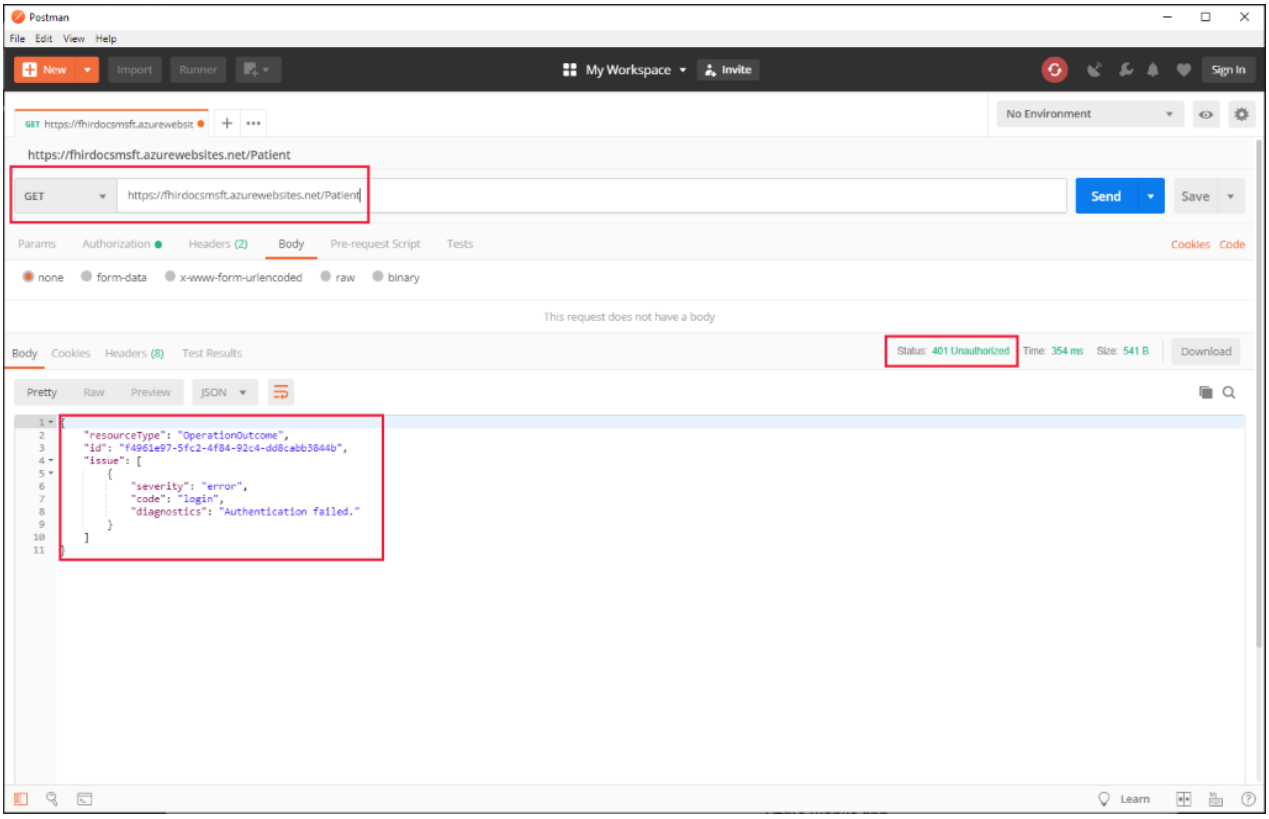
Connect to FHIR server

Using Postman, do a `GET` request to `https://fhir-server-url/metadata`:



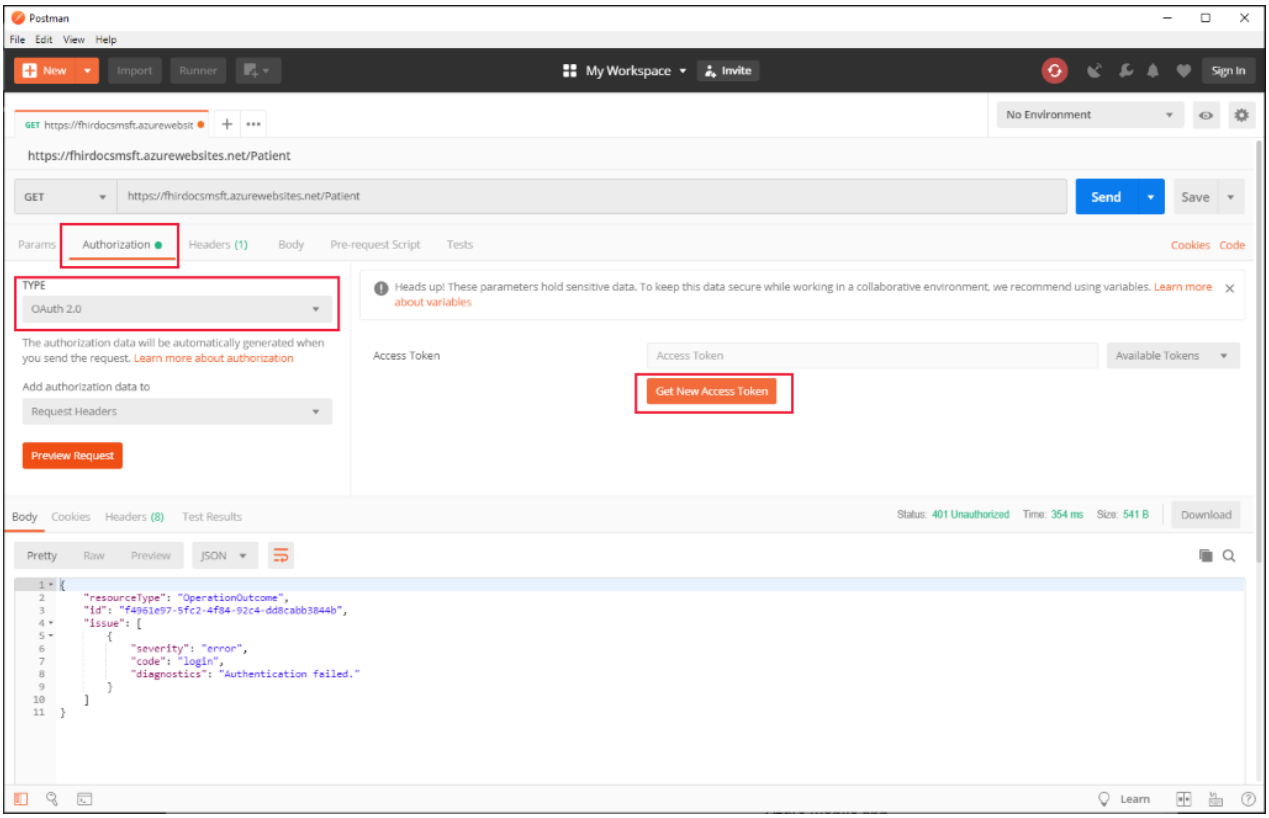
The metadata URL for Azure API for FHIR is `https://MYACCOUNT.azurehealthcareapis.com/metadata`. In this example, the FHIR server URL is `https://fhirdocsmsft.azurewebsites.net` and the capability statement of the server is available at `https://fhirdocsmsft.azurewebsites.net/metadata`. That endpoint should be accessible without authentication.

If you attempt to access restricted resources, you should get an "Authentication failed" response:

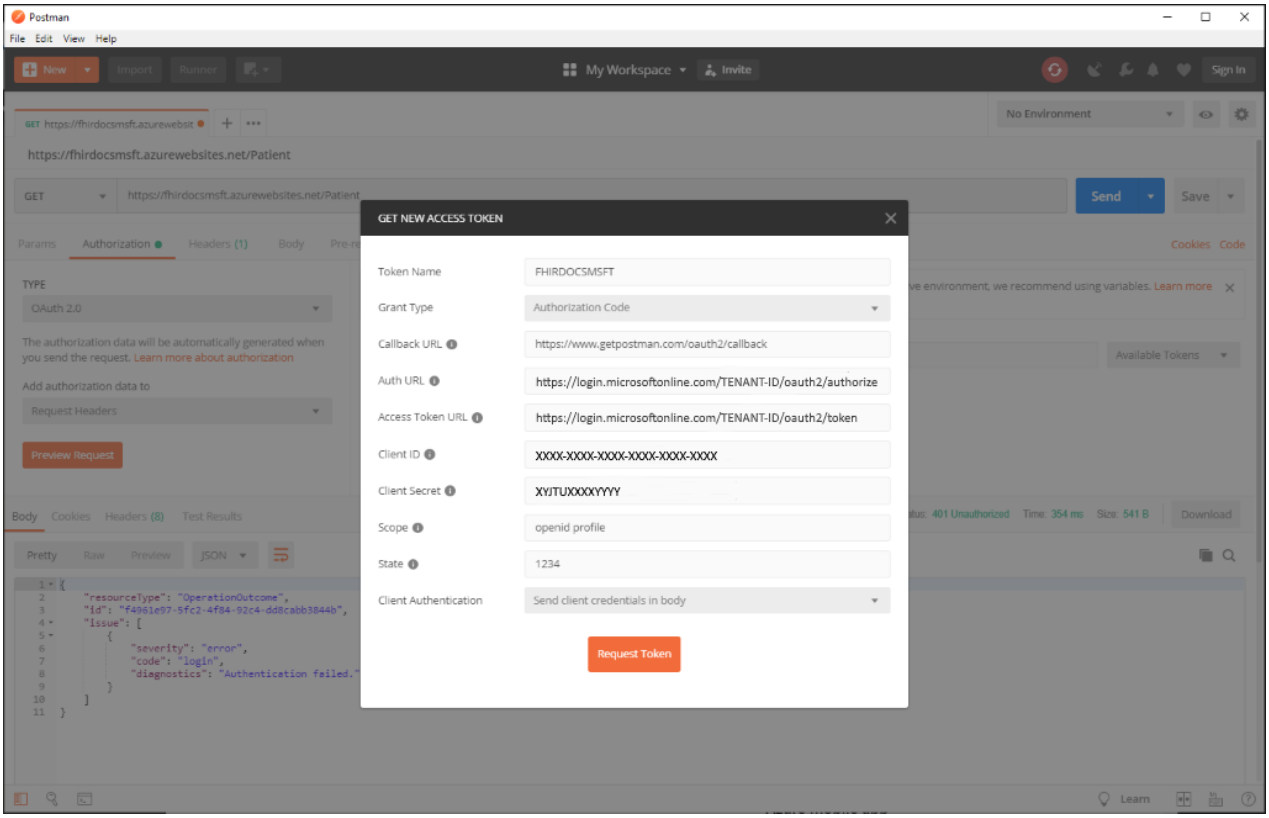


Obtaining an access token

To obtain a valid access token, select "Authorization" and pick TYPE "OAuth 2.0":



Hit "Get New Access Token" and a dialog appears:



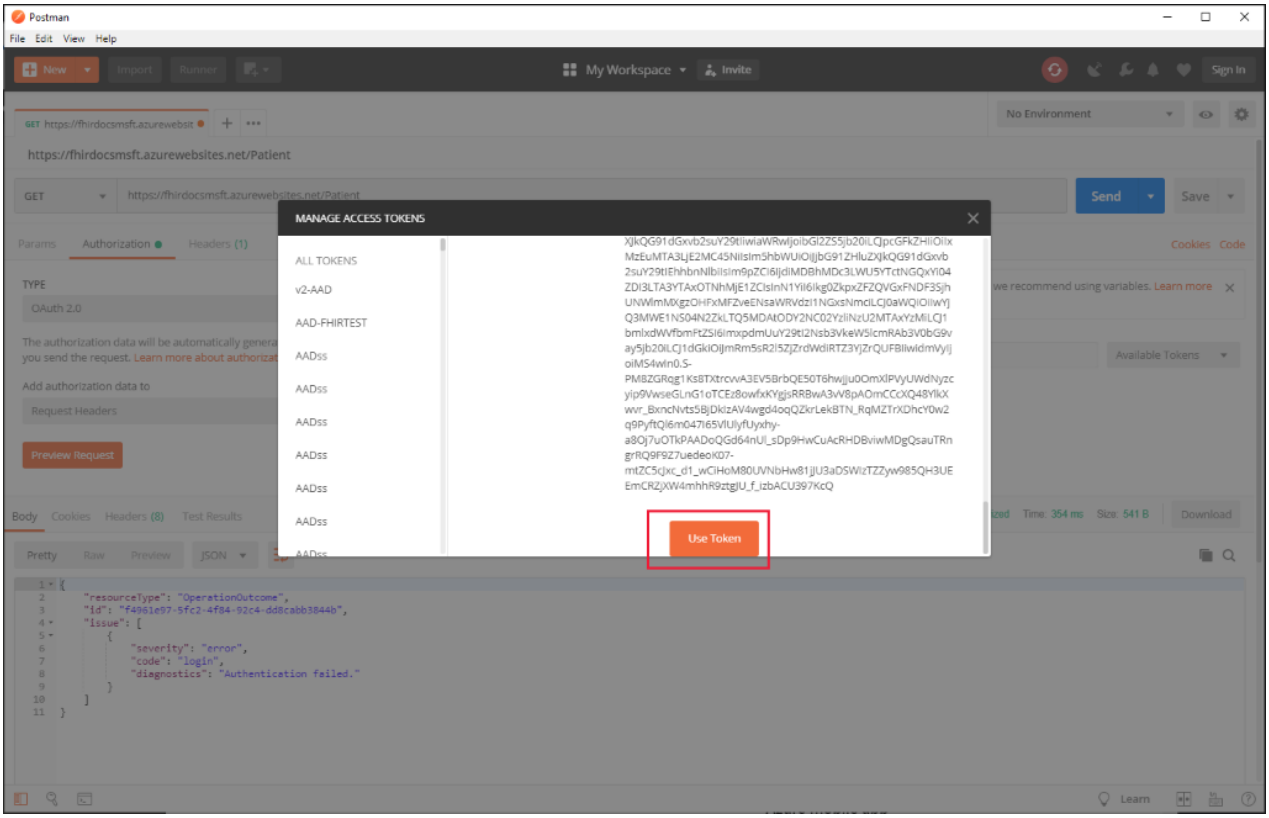
You will need to some details:

FIELD	EXAMPLE VALUE	COMMENT
Token Name	MYTOKEN	A name you choose
Grant Type	Authorization Code	

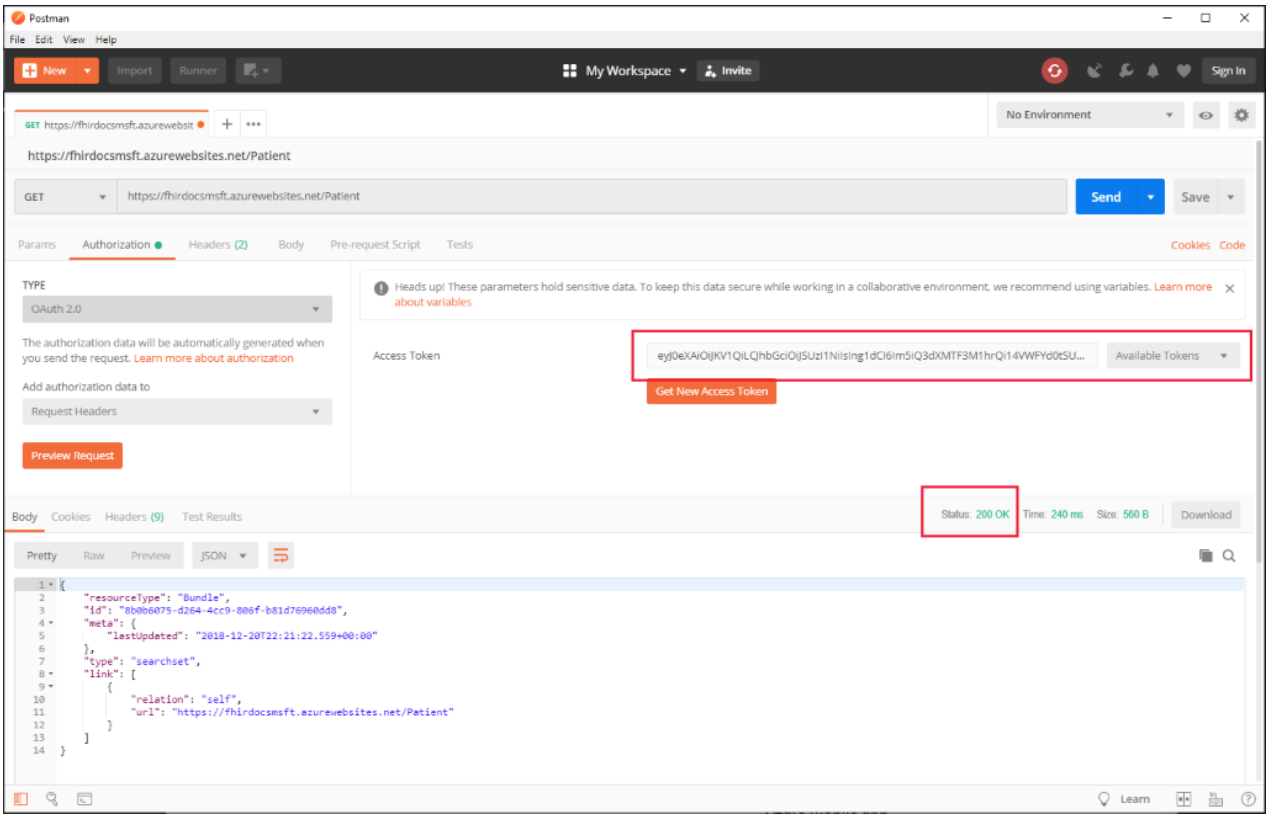
FIELD	EXAMPLE VALUE	COMMENT
Callback URL	`https://www.getpostman.com/oauth2/callback`	
Auth URL	`https://login.microsoftonline.com/{TEN` `audience` is `ID}/oauth2/authorize?resource=<audienc` `https://MYACCOUNT.azurehealthcareapis.com` for Azure API for FHIR	
Access Token URL	`https://login.microsoftonline.com/{TENANT ID}/oauth2/token`	
Client ID	`XXXXXXXX-XXX-XXXX-XXXX-XXXXXXXXXXXX`	Application ID
Client Secret	`XXXXXXXX`	Secret client key
Scope	`<Leave Blank>`	
State	`1234`	
Client Authentication	Send client credentials in body	

Hit "Request Token" and you will be guided through the Azure Active Directory Authentication flow and a token will be returned to Postman. If you run into problems open the Postman Console (from the "View->Show Postman Console" menu item).

Scroll down on the returned token screen and hit "Use Token":



The token should now be populated in the "Access Token" field and you can select tokens from "Available Tokens". If you "Send" again to repeat the `Patient` resource search, you should get a Status `200 OK`:



In this case, there are no patients in the database and the search is empty.

If you inspect the access token with a tool like https://jwt.ms, you should see content like:

```
{
  "aud": "https://MYACCOUNT.azurehealthcareapis.com",
  "iss": "https://sts.windows.net/{TENANT-ID}/",
  "iat": 1545343803,
  "nbf": 1545343803,
  "exp": 1545347703,
  "acr": "1",
  "aio": "AUQAu/8JXXXXXXXXXdQxcxn1eis459j70Kf9DwcUjlKY3I2G/9aOnSbw==",
  "amr": [
    "pwd"
  ],
  "appid": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "oid": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "appidacr": "1",

  ...// Truncated
}
```

In troubleshooting situations, validating that you have the correct audience (`aud` claim) is a good place to start. The managed Azure API for FHIR uses identity object IDs to restrict access to the service. Make sure that `oid` claim of the token contains an object ID from the list of allowed object IDs.

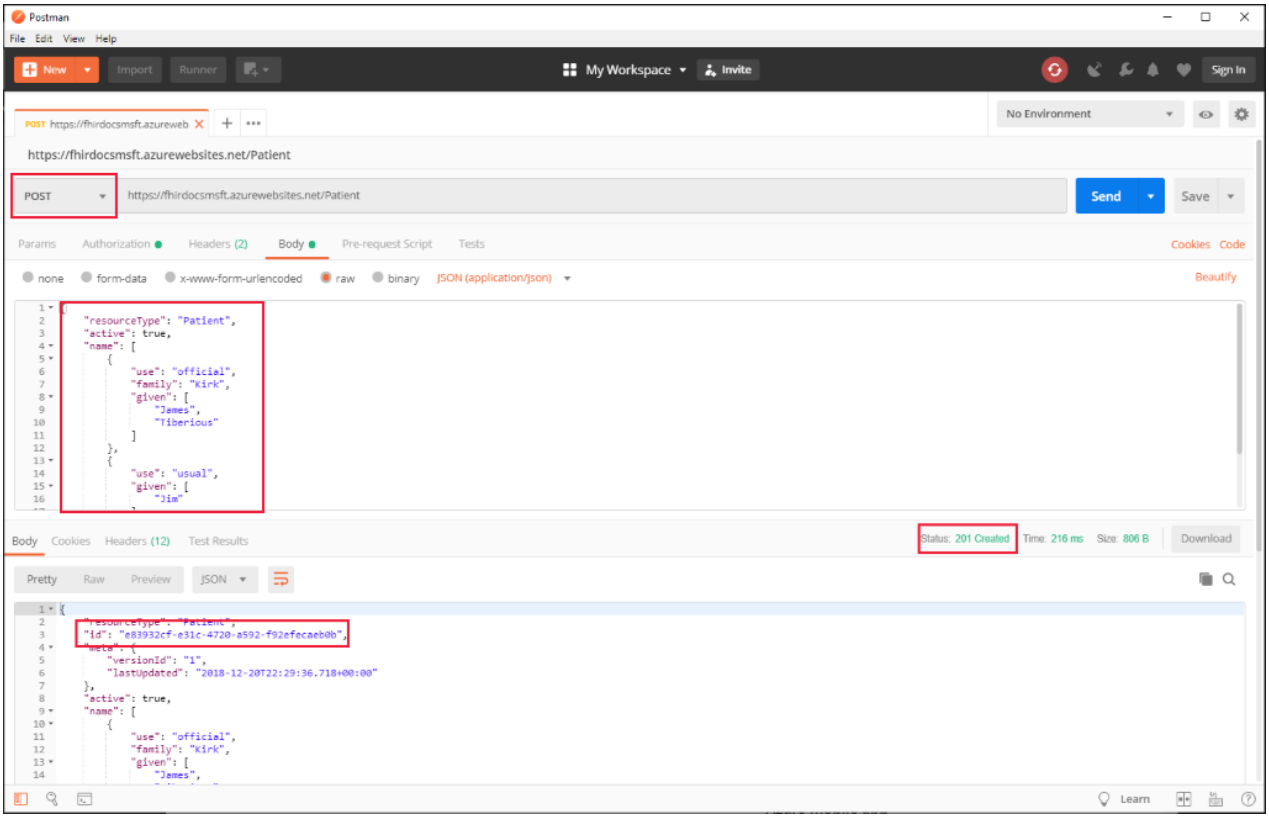
It is also possible to get a token for the Azure API for FHIR using the Azure CLI.

Inserting a patient

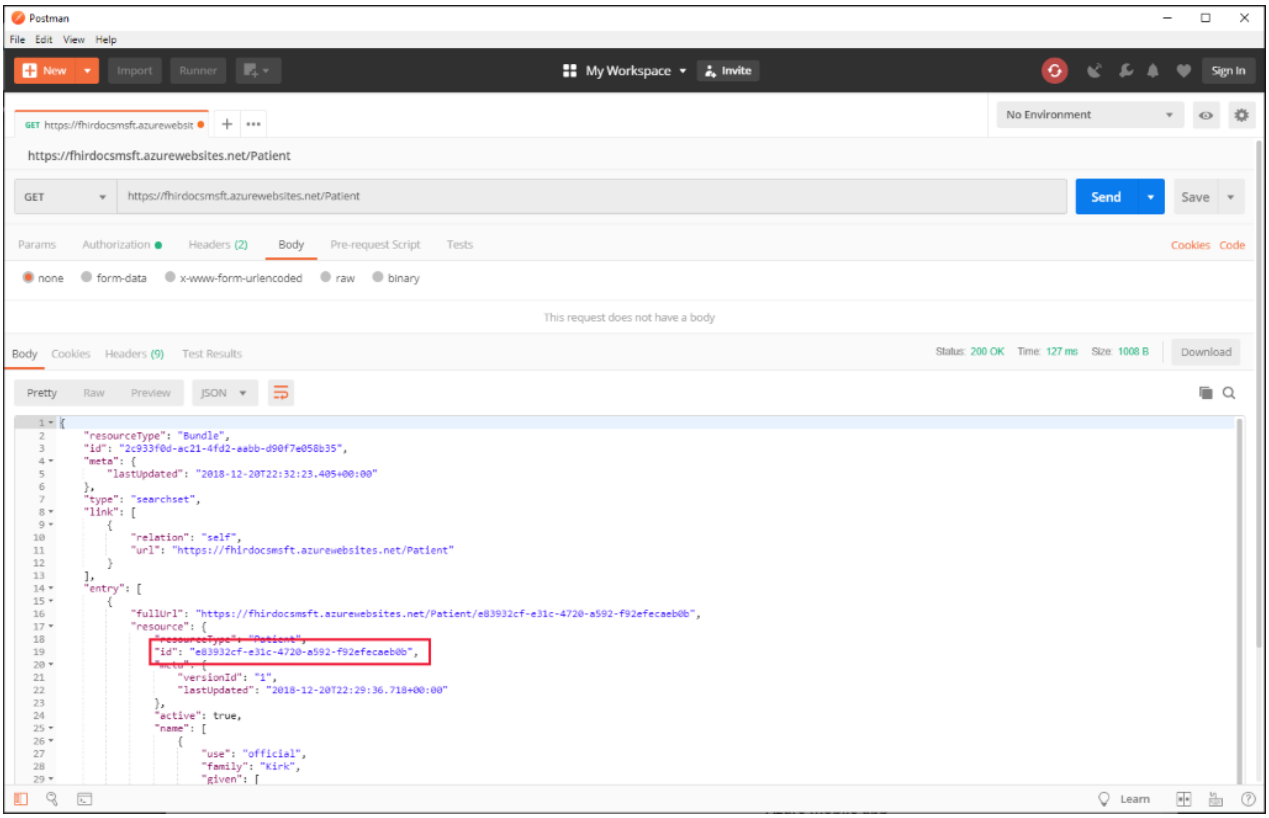
Now that you have a valid access token. You can insert a new patient. Switch to method "POST" and add the following JSON document in the body of the request:

```
{
    "resourceType": "Patient",
    "active": true,
    "name": [
        {
            "use": "official",
            "family": "Kirk",
            "given": [
                "James",
                "Tiberious"
            ]
        },
        {
            "use": "usual",
            "given": [
                "Jim"
            ]
        }
    ],
    "gender": "male",
    "birthDate": "1960-12-25"
}
```

Hit "Send" and you should see that the patient is successfully created:



If you repeat the patient search, you should now see the patient record:



Next steps

In this tutorial, you've accessed an FHIR API using postman. Read about the supported API features in our supported features section.

Supported features

Tutorial: Azure Active Directory SMART on FHIR proxy

5 minutes to read • Edit Online

SMART on FHIR is a set of open specifications to integrate partner applications with FHIR servers and electronic medical records systems that have FHIR interfaces. One of the main purposes of the specifications is to describe how an application should discover authentication endpoints for an FHIR server and start an authentication sequence.

Authentication is based on OAuth2. But because SMART on FHIR uses parameter naming conventions that are not immediately compatible with Azure Active Directory (Azure AD), the Azure API for FHIR has a built-in Azure AD SMART on FHIR proxy that enables a subset of the SMART on FHIR launch sequences. Specifically, the proxy enables the EHR launch sequence.

This tutorial describes how to use the proxy to enable SMART on FHIR applications with the Azure API for FHIR.

Prerequisites

- An instance of the Azure API for FHIR
- .NET Core 2.2

Configure Azure AD registrations

SMART on FHIR requires that `Audience` has an identifier URI equal to the URI of the FHIR service. The standard configuration of the Azure API for FHIR uses an `Audience` value of `https://azurehealthcareapis.com`. However, you can also set a value matching the specific URL of your FHIR service (for example `https://MYFHIRAPI.azurehealthcareapis.com`). This is required when working with the SMART on FHIR proxy.

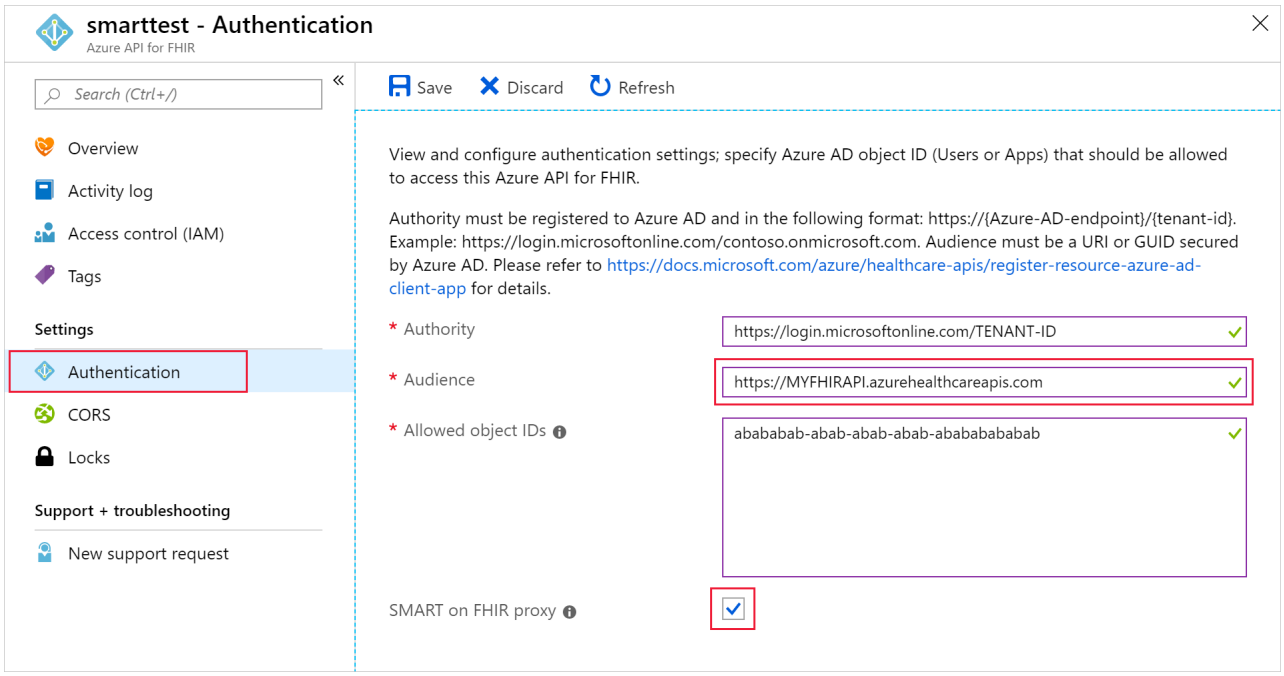
You will also need a client application registration. Most SMART on FHIR applications are single-page JavaScript applications. So you should follow the instructions for configuring a public Azure AD client application.

After you complete these steps, you should have:

- A FHIR server with rge audience set to `https://MYFHIRAPI.azurehealthcareapis.com`, where `MYFHIRAPI` is the name of your Azure API for FHIR instance.
- A public client application registration. Make a note of the application ID for this client application.

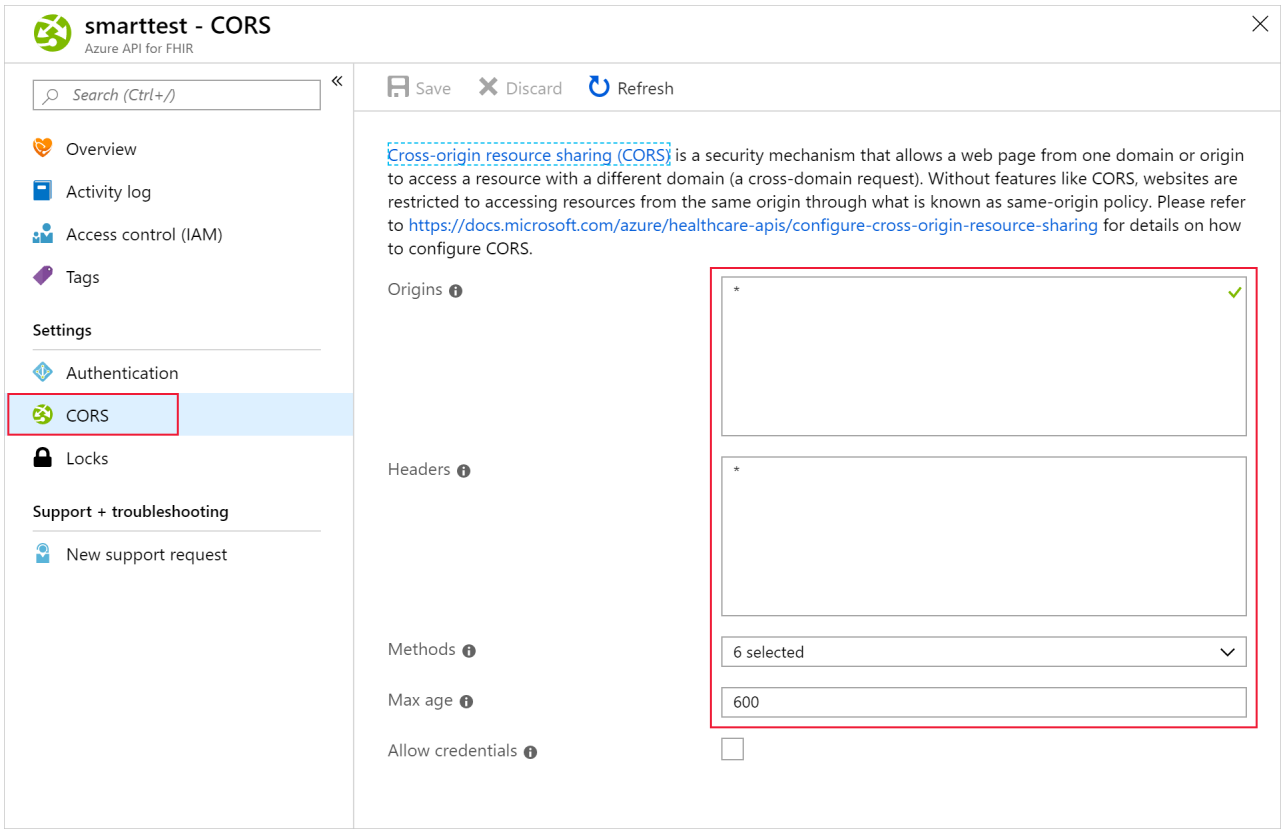
Enable the SMART on FHIR proxy

Enable the SMART on FHIR proxy in the **Authentication** settings for your Azure API for FHIR instance by selecting the **SMART on FHIR proxy** check box:



Enable CORS

Because most SMART on FHIR applications are single-page JavaScript apps, you need to enable cross-origin resource sharing (CORS) for the Azure API for FHIR:



Configure the reply URL

The SMART on FHIR proxy acts as an intermediary between the SMART on FHIR app and Azure AD. The authentication reply (the authentication code) must go to the SMART on FHIR proxy instead of the app itself. The proxy then forwards the reply to the app.

Because of this two-step relay of the authentication code, you need to set the reply URL (callback) for your Azure AD client application to a URL that is a combination of the reply URL for the SMART on FHIR proxy and the reply URL for the SMART on FHIR app. The combined reply URL takes this form:

```
https://MYFHIRAPI.azurehealthcareapis.com/AadSmartOnFhirProxy/callback/aHR0cHM6Ly9sb2NhbGhvc3Q6NTAwMS9zYW1wbGVhcHAvaW5kZXguaHRtbA
```

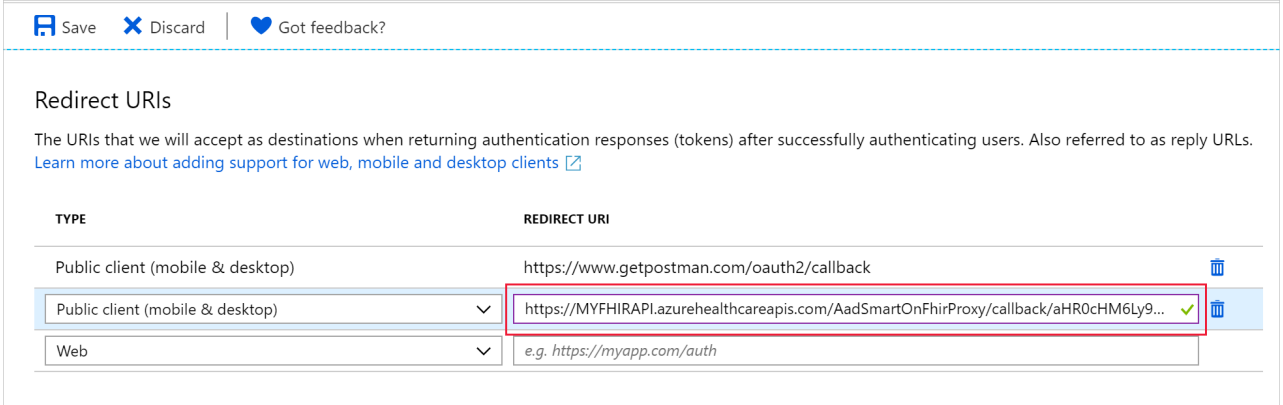
In that reply, `aHR0cHM6Ly9sb2NhbGhvc3Q6NTAwMS9zYW1wbGVhcHAvaW5kZXguaHRtbA` is a URL-safe, base64-encoded version of the reply URL for the SMART on FHIR app. For the SMART on FHIR app launcher, when the app is running locally, the reply URL is `https://localhost:5001/sampleapp/index.html`.

You can generate the combined reply URL by using a script like this:

```
$replyUrl = "https://localhost:5001/sampleapp/index.html"
$fhirServerUrl = "https://MYFHIRAPI.azurewebsites.net"
$bytes = [System.Text.Encoding]::UTF8.GetBytes($ReplyUrl)
$encodedText = [Convert]::ToBase64String($bytes)
$encodedText = $encodedText.TrimEnd('=');
$encodedText = $encodedText.Replace('/','_');
$encodedText = $encodedText.Replace('+','-');

$newReplyUrl = $FhirServerUrl.TrimEnd('/') + "/AadSmartOnFhirProxy/callback/" + $encodedText
```

Add the reply URL to the public client application that you created earlier for Azure AD:



Get a test patient

To test the Azure API for FHIR and the SMART on FHIR proxy, you'll need to have at least one patient in the database. If you have not interacted with the API yet and you don't have data in the database, follow the FHIR API Postman tutorial to load a patient. Make a note of the ID of a specific patient.

Download the SMART on FHIR app launcher

The open-source FHIR Server for Azure repository includes a simple SMART on FHIR app launcher and a sample SMART on FHIR app. In this tutorial, use this SMART on FHIR launcher locally to test the setup.

You can clone the GitHub repository and go to the application by using these commands:

```
git clone https://github.com/Microsoft/fhir-server
cd fhir-server/samples/apps/SmartLauncher
```

The application needs a few configuration settings, which you can set in `appsettings.json`:

```
{
    "FhirServerUrl": "https://MYFHIRAPI.azurehealthcareapis.com",
    "ClientId": "APP-ID",
    "DefaultSmartAppUrl": "/sampleapp/launch.html"
}
```

We recommend that you use the `dotnet user-secrets` feature:

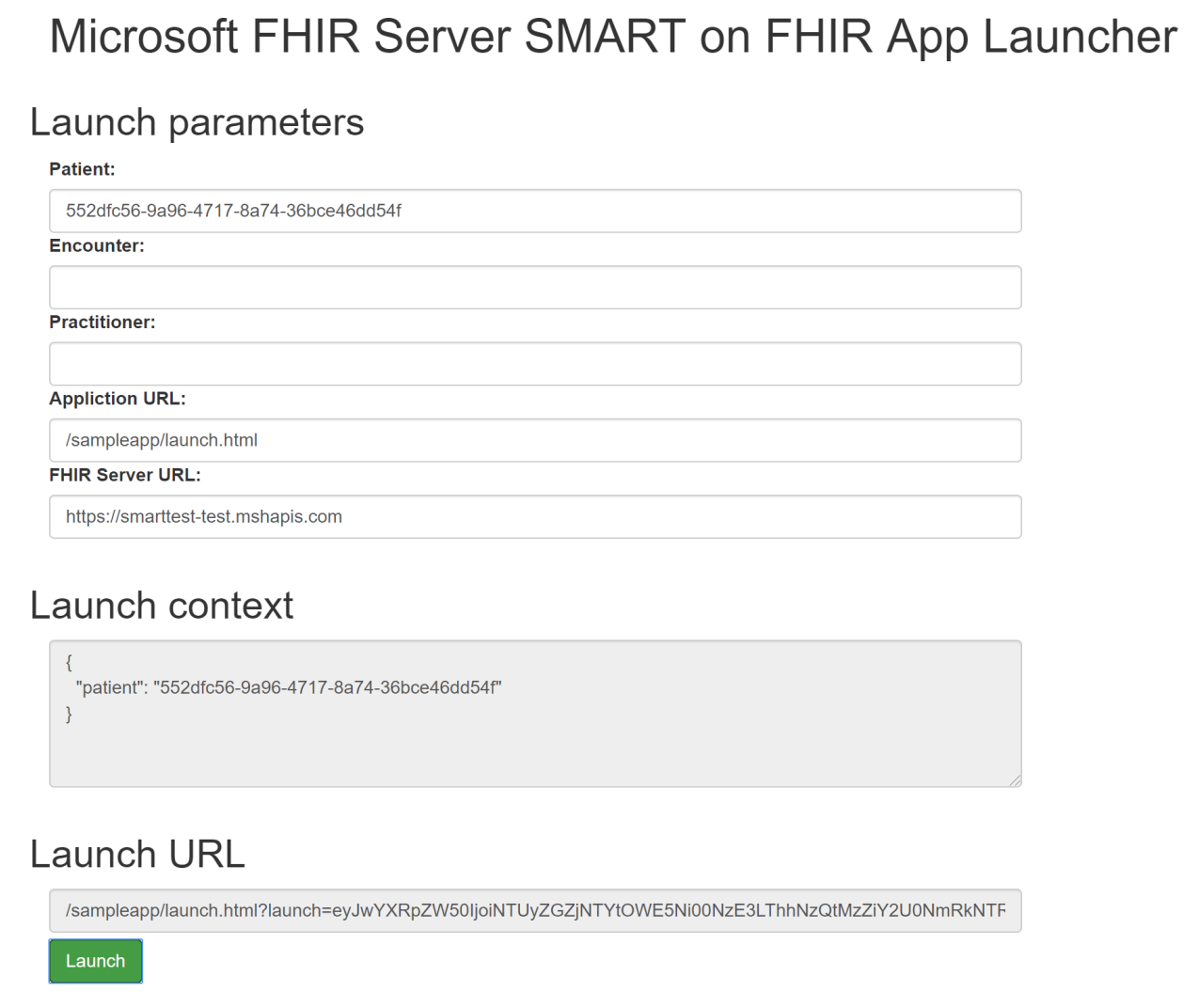
```
dotnet user-secrets set FhirServerUrl https://MYFHIRAPI.azurehealthcareapis.com
dotnet user-secrets set ClientId <APP-ID>
```

Use this command to run the application:

```
dotnet run
```

Test the SMART on FHIR proxy

After you start the SMART on FHIR app launcher, you can point your browser to `https://localhost:5001`, where you should see the following screen:



When you enter **Patient**, **Encounter**, or **Practitioner** information, you'll notice that the **Launch context** is updated. When you're using the Azure API for FHIR, the launch context is simply a JSON document that contains information about patient, practitioner, and more. This launch context is base64 encoded and passed to the SMART on FHIR app as the `launch` query parameter. According to the SMART on FHIR specification, this variable is

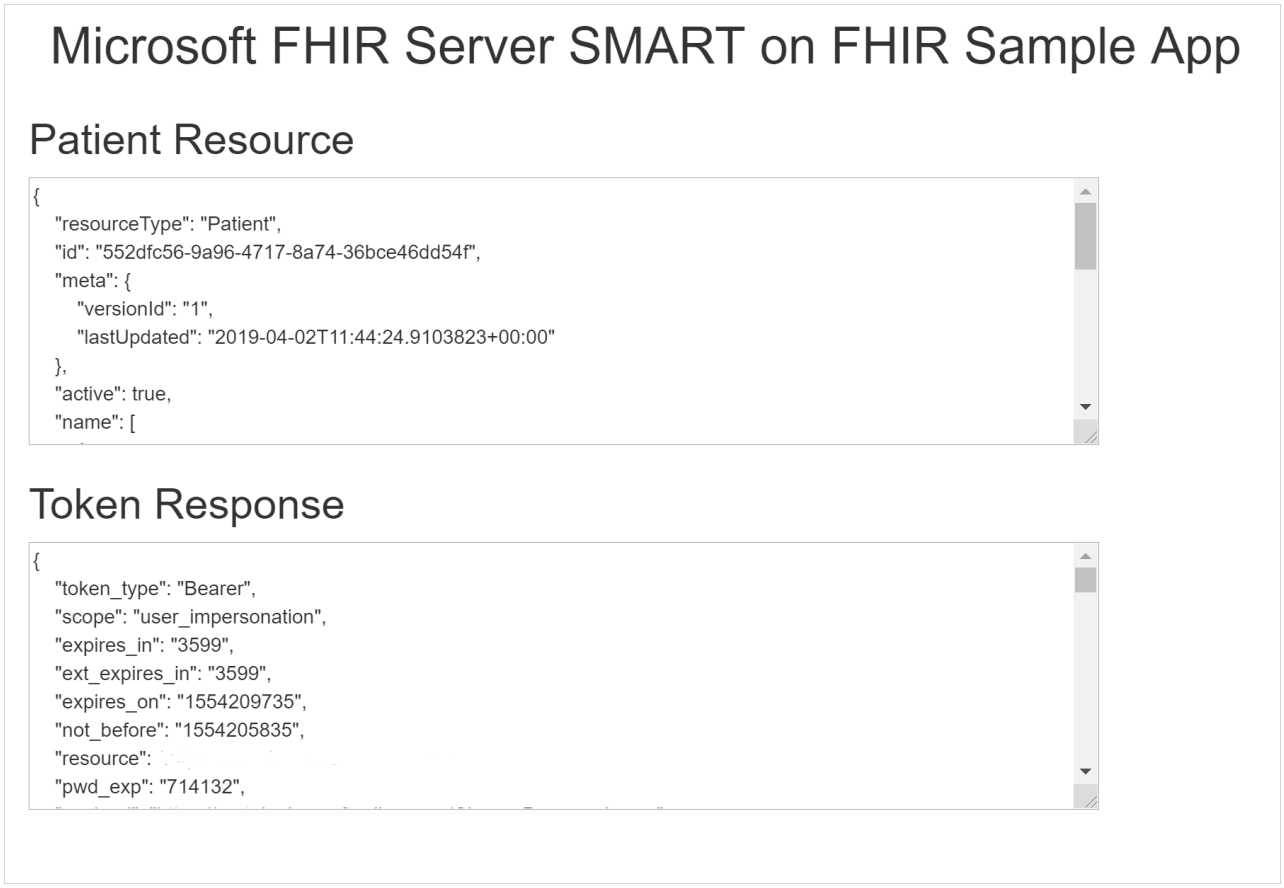
opaque to the SMART on FHIR app and passed on to the identity provider.

The SMART on FHIR proxy uses this information to populate fields in the token response. The SMART on FHIR app *can* use these fields to control which patient it requests data for and how it renders the application on the screen. The SMART on FHIR proxy supports the following fields:

- `patient`
- `encounter`
- `practitioner`
- `need_patient_banner`
- `smart_style_url`

These fields are meant to provide guidance to the app, but they don't convey any security information. A SMART on FHIR application can ignore them.

Notice that the SMART on FHIR app launcher updates the **Launch URL** information at the bottom of the page. Select **Launch** to start the sample app, and you should see something like this sample:



Inspect the token response to see how the launch context fields are passed on to the app.

Next steps

In this tutorial, you've configured the Azure Active Directory SMART on FHIR proxy. To explore the use of SMART on FHIR applications with the Azure API for FHIR and the open-source FHIR Server for Azure, go to the repository of FHIR server samples on GitHub:

FHIR server samples

Register the Azure Active Directory apps for Azure API for FHIR

2 minutes to read • Edit Online

You have several configuration options to choose from when you're setting up the Azure API for FHIR or the FHIR Server for Azure (OSS). For open source, you'll need to create your own resource application registration. For Azure API for FHIR, this resource application is created automatically.

Application registrations

In order for an application to interact with Azure AD, it needs to be registered. In the context of the FHIR server, there are two kinds of application registrations to discuss:

1. Resource application registrations.
2. Client application registrations.

Resource applications are representations in Azure AD of an API or resource that is secured with Azure AD, specifically it would be the Azure API for FHIR. A resource application for Azure API for FHIR will be created automatically when you provision the service, but if you're using the open-source server, you'll need to register a resource application in Azure AD. This resource application will have an identifier URI. It's recommended that this URI be the same as the URI of the FHIR server. This URI should be used as the `Audience` for the FHIR server. A client application can request access to this FHIR server when it requests a token.

Client applications are registrations of the clients that will be requesting tokens. Often in OAuth 2.0, we distinguish between at least three different types of applications:

1. **Confidential clients**, also known as web apps in Azure AD. Confidential clients are applications that use authorization code flow to obtain a token on behalf of a signed in user presenting valid credentials. They are called confidential clients because they are able to hold a secret and will present this secret to Azure AD when exchanging the authentication code for a token. Since confidential clients are able to authenticate themselves using the client secret, they are trusted more than public clients and can have longer lived tokens and be granted a refresh token. Read the details on how to register a confidential client. Note that is important to register the reply url at which the client will be receiving the authorization code.
2. **Public clients**. These are clients that cannot keep a secret. Typically this would be a mobile device application or a single page JavaScript application, where a secret in the client could be discovered by a user. Public clients also use authorization code flow, but they are not allowed to present a secret when obtaining a token and they may have shorter lived tokens and no refresh token. Read the details on how to register a public client.
3. Service clients. These clients obtain tokens on behalf of themselves (not on behalf of a user) using the client credentials flow. They typically represent applications that access the FHIR server in a non-interactive way. An example would be an ingestion process. When using a service client, it is not necessary to start the process of getting a token with a call to the `/authorize` endpoint. A service client can go straight to the `/token` endpoint and present client ID and client secret to obtain a token. Read the details on how to register a service client

Next steps

In this overview, you've gone through the types of application registrations you may need in order to work with a FHIR API.

Based on your setup, please see the how-to-guides to register your applications

- Register a resource application
- Register a confidential client application
- Register a public client application
- Register a service application

Once you have registered your applications, you can deploy the Azure API for FHIR.

Deploy Azure API for FHIR

Register a resource application in Azure Active Directory

2 minutes to read • Edit Online

In this article, you'll learn how to register a resource (or API) application in Azure Active Directory. A resource application is an Azure Active Directory representation of the FHIR server API itself and client applications can request access to the resource when authenticating. The resource application is also known as the *audience* in OAuth parlance.

Azure API for FHIR

If you are using the Azure API for FHIR, a resource application is automatically created when you deploy the service. As long as you are using the Azure API for FHIR in the same Azure Active Directory tenant as you are deploying your application, you can skip this how-to-guide and instead deploy your Azure API for FHIR to get started.

If you are using a different Azure Active Directory tenant (not associated with your subscription), you can import the Azure API for FHIR resource application into your tenant with PowerShell:

```
New-AzADServicePrincipal -ApplicationId 4f6778d8-5aef-43dc-a1ff-b073724b9495
```

or you can use Azure CLI:

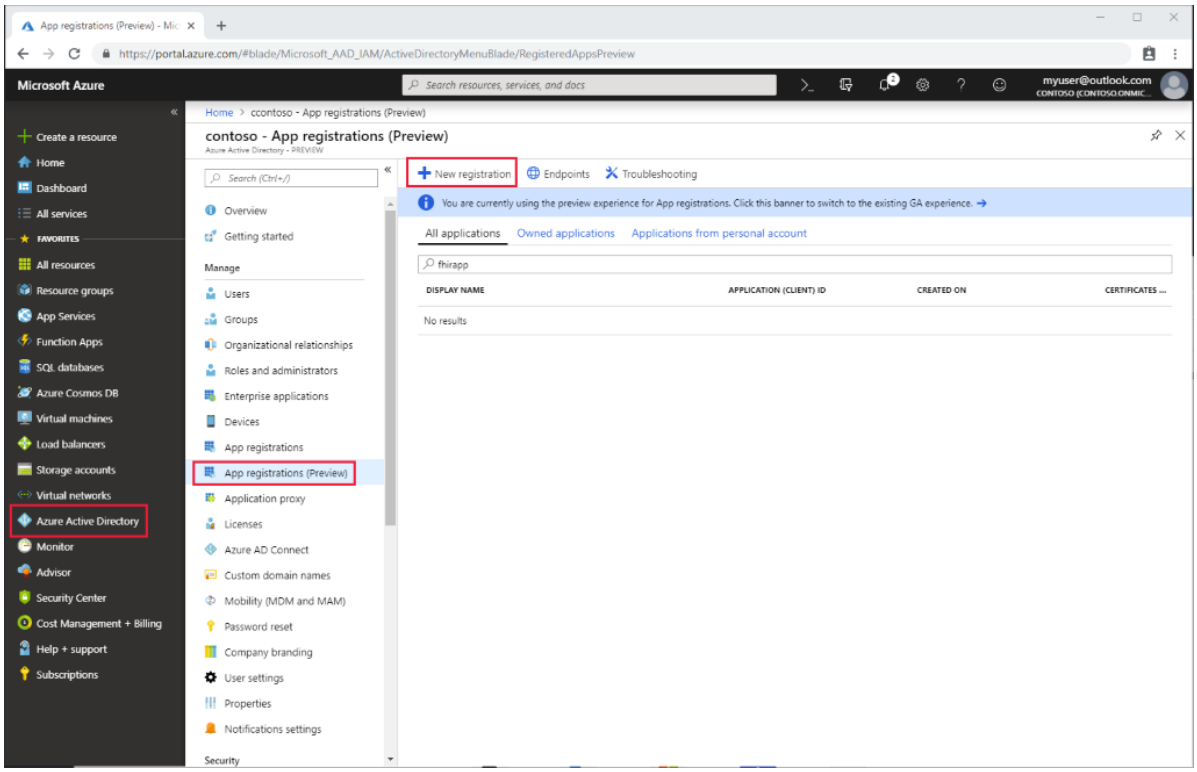
```
az ad sp create --id 4f6778d8-5aef-43dc-a1ff-b073724b9495
```

FHIR Server for Azure

If you are using the open source FHIR Server for Azure, follow the steps below to register a resource application.

App registrations in Azure portal

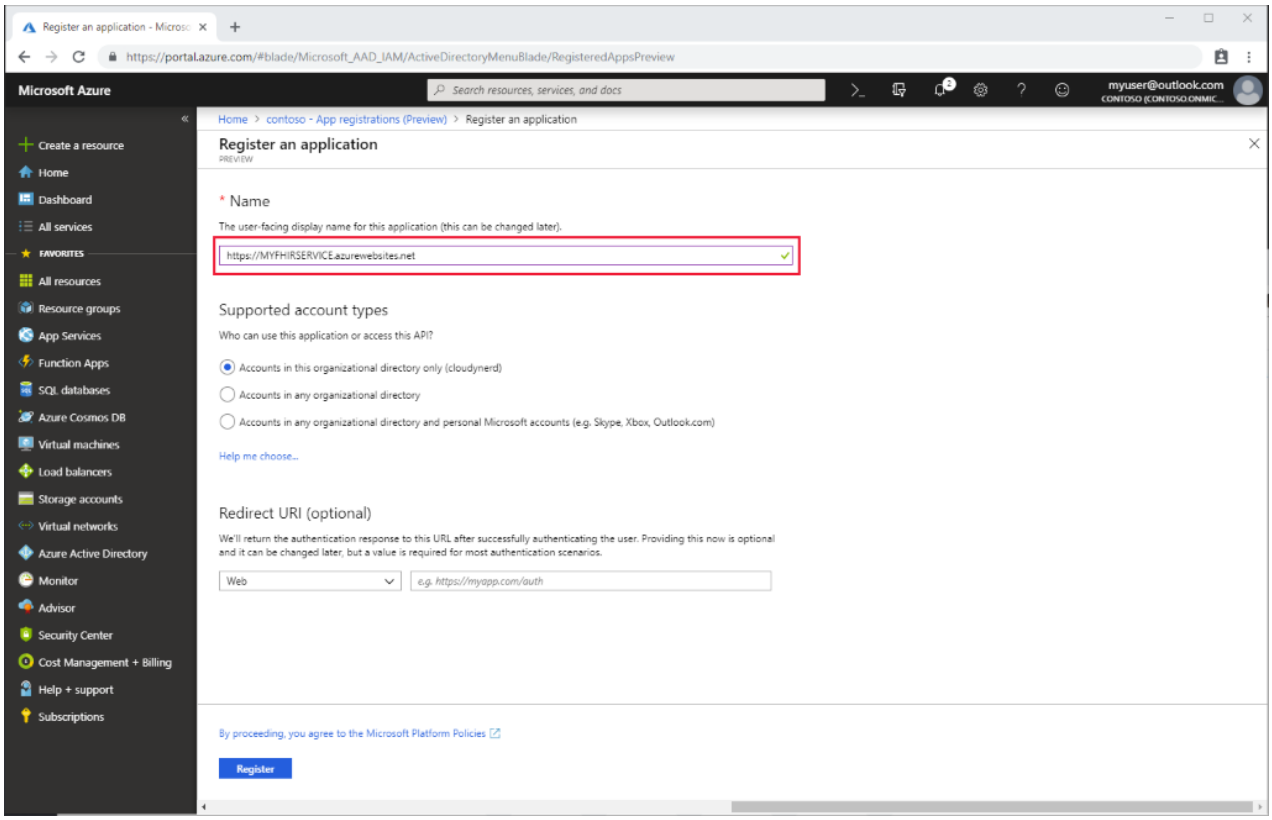
1. In the Azure portal, on the left navigation panel, click **Azure Active Directory**.
2. In the **Azure Active Directory** blade click **App registrations**:



3. Click the **New registration**.

Add a new application registration

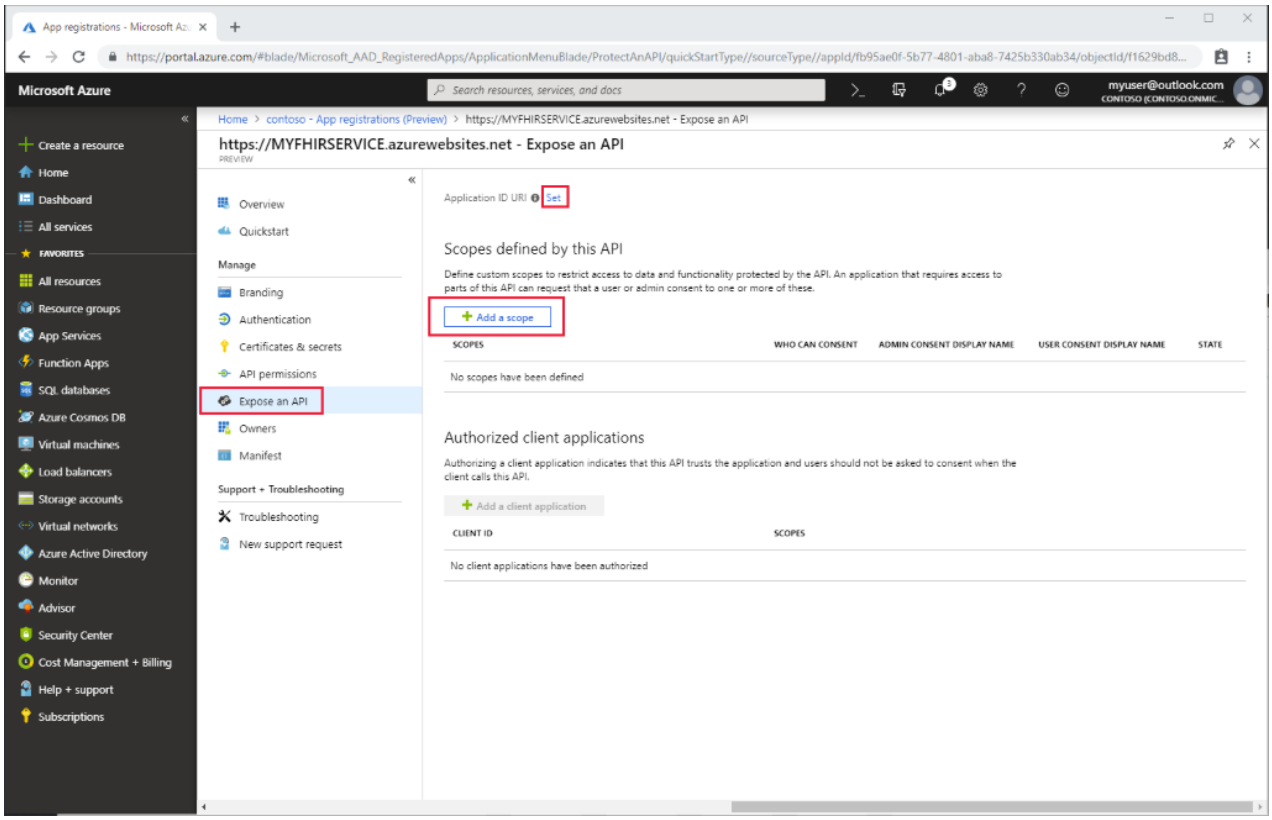
Fill in the details for the new application. There are no specific requirements for the display name, but setting it to the URI of the FHIR server makes it easy to find:



Set identifier URI and define scopes

A resource application has an identifier URI (Application ID URI), which clients can use when requesting access to the resource. This value will populate the `aud` claim of the access token. It is recommended that you set this URI to be the URI of your FHIR server. For SMART on FHIR apps, it is assumed that the *audience* is the URI of the FHIR server.

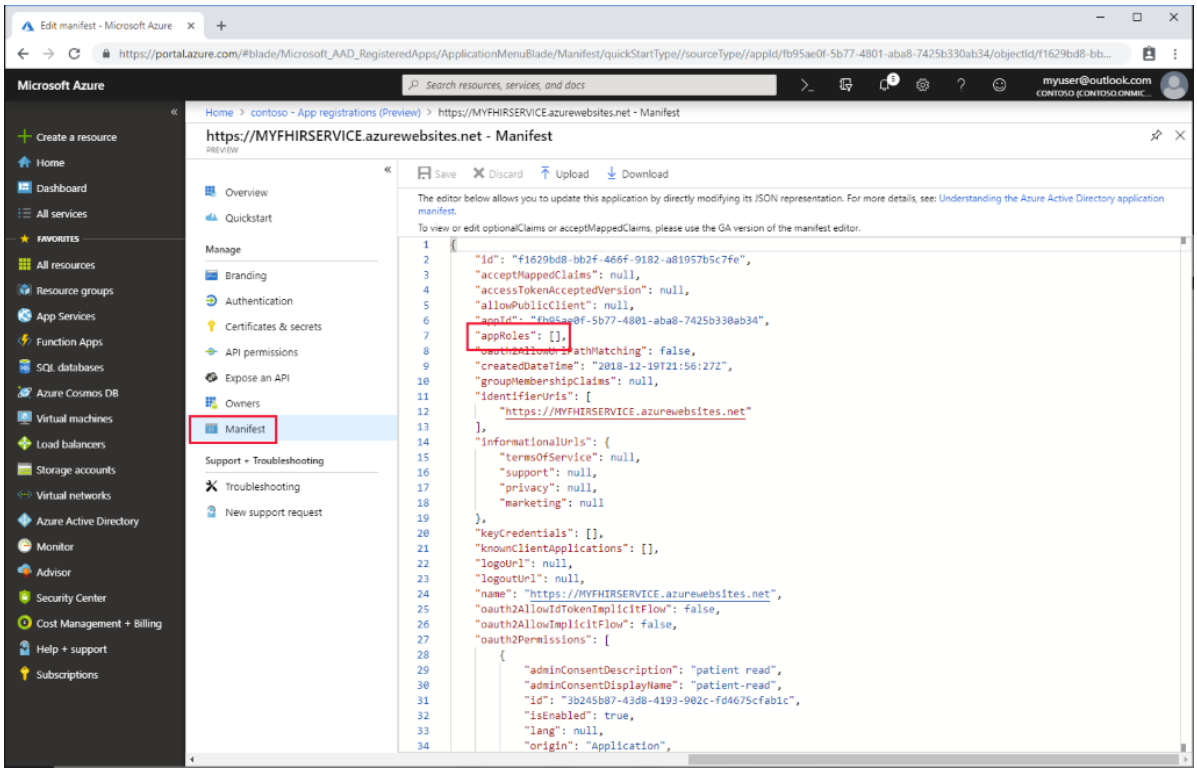
1. Click **Expose an API**
2. Click **Set** next to *Application ID URI*.
3. Enter the identifier URI and click **Save**. A good identifier URI would be the URI of your FHIR server.
4. Click **Add a scope** and add any scopes that you would like to define for your API. You are required to add at least one scope in order to grant permissions to your resource application in the future. If you don't have any specific scopes you want to add, you can add user_impersonation as a scope.



Define application roles

The Azure API for FHIR and the OSS FHIR Server for Azure use Azure Active Directory application roles for role-based access control. To define which roles should be available for your FHIR Server API, open the resource application's manifest:

1. Click **Manifest**:



2. In the `appRoles` property, add the roles you would like users or applications to have:

```
"appRoles": [
  {
    "allowedMemberTypes": [
      "User",
      "Application"
    ],
    "description": "FHIR Server Administrators",
    "displayName": "admin",
    "id": "1b4f816e-5eaf-48b9-8613-7923830595ad",
    "isEnabled": true,
    "value": "admin"
  },
  {
    "allowedMemberTypes": [
      "User"
    ],
    "description": "Users who can read",
    "displayName": "reader",
    "id": "c20e145e-5459-4a6c-a074-b942bbd4cfe1",
    "isEnabled": true,
    "value": "reader"
  }
],
```

Next steps

In this article, you've learned how to register a resource application in Azure Active Directory. Next, deploy the Azure API for FHIR.

Deploy Azure API for FHIR

Register a confidential client application in Azure Active Directory

2 minutes to read • Edit Online

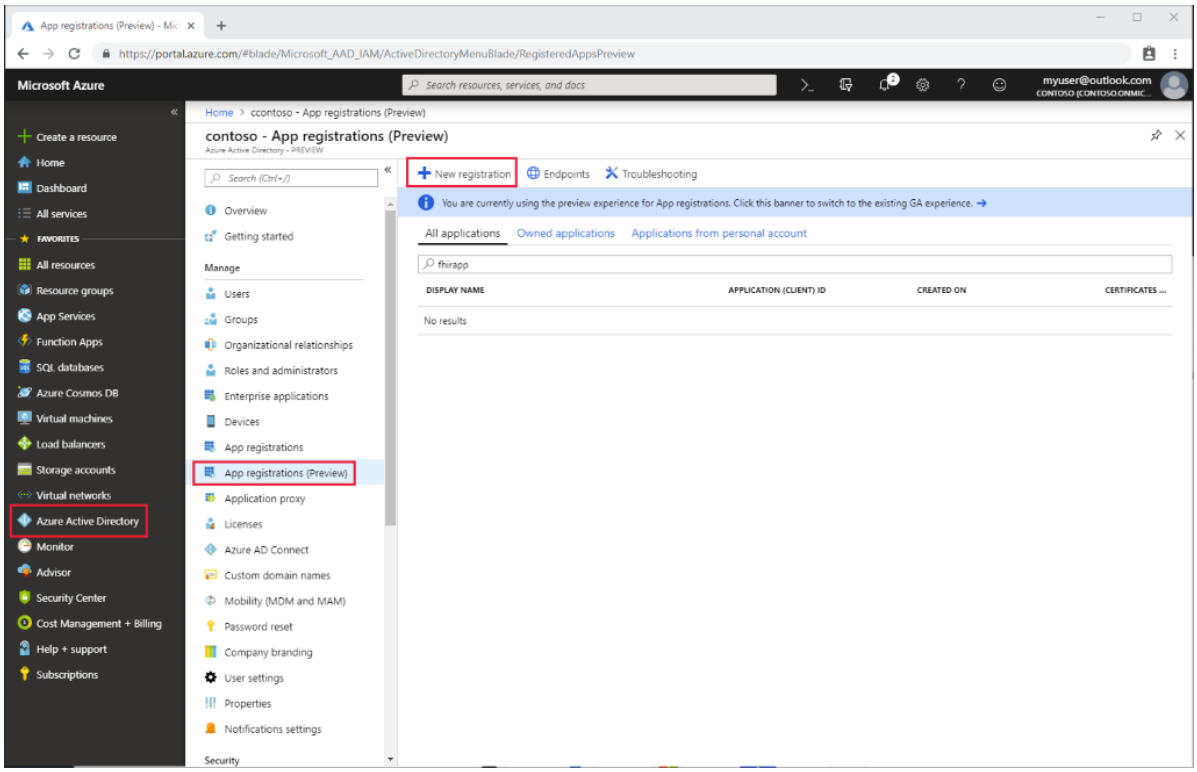
In this tutorial, you'll learn how to register a confidential client application in Azure Active Directory.

A client application registration is an Azure Active Directory representation of an application that can be used to authenticate on behalf of a user and request access to resource applications. A confidential client application is an application that can be trusted to hold a secret and present that secret when requesting access tokens. Examples of confidential applications are server-side applications.

To register a new confidential application in the portal, follow the steps below.

App registrations in Azure portal

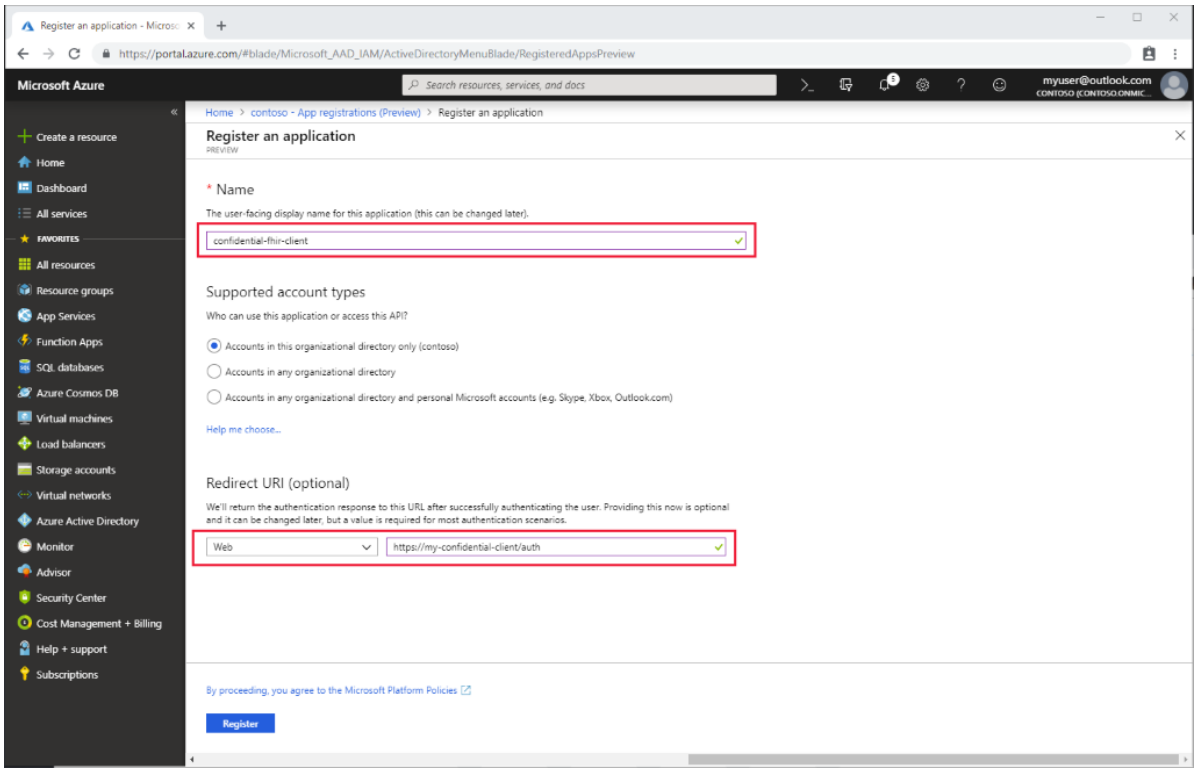
1. In the Azure portal, on the left navigation panel, click **Azure Active Directory**.
2. In the **Azure Active Directory** blade click **App registrations**:



3. Click the **New registration**.

Register a new application

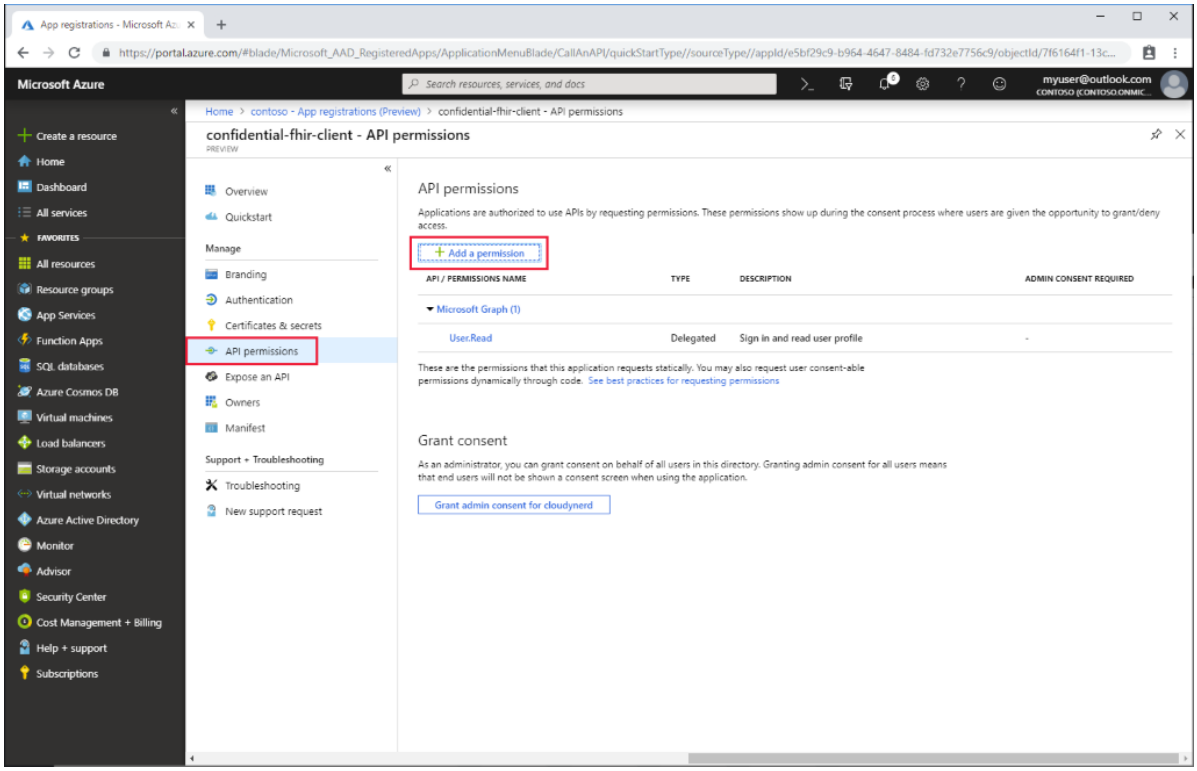
1. Give the application a display name.
2. Provide a reply URL. These details can be changed later, but if you know the reply URL of your application, enter it now.



API permissions

Next add API permissions:

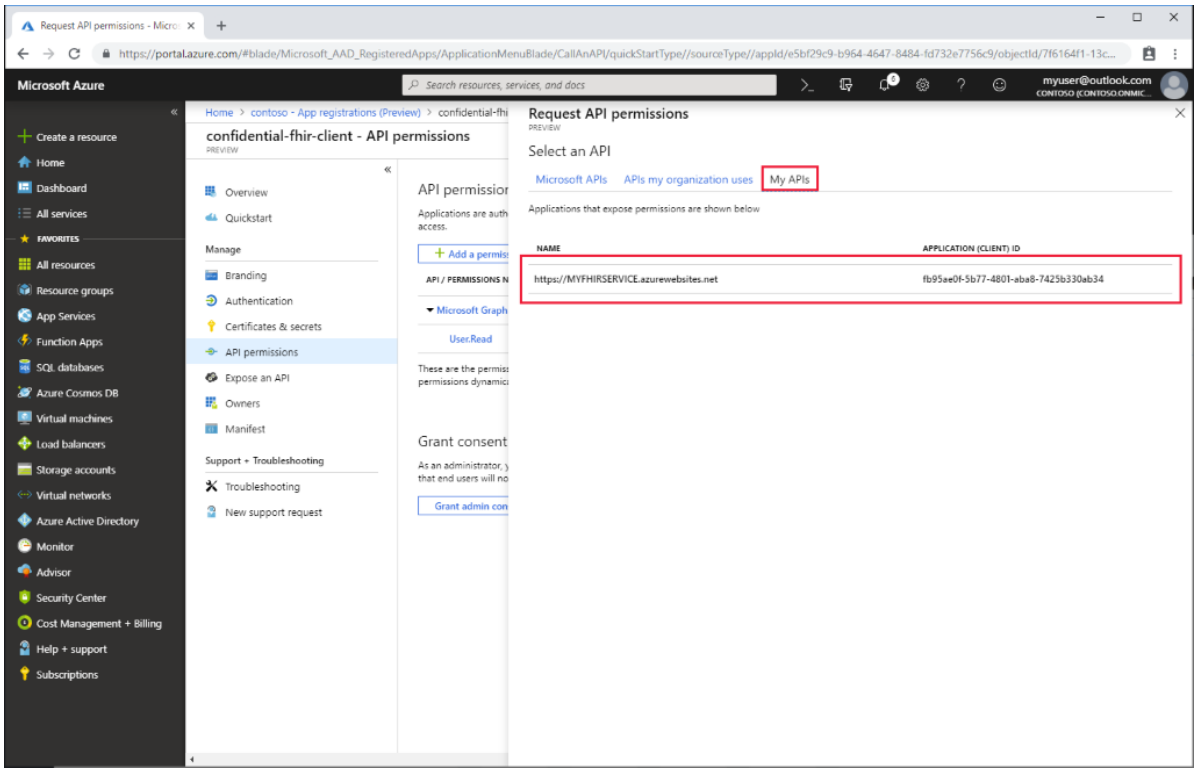
1. Open the **API permissions**:



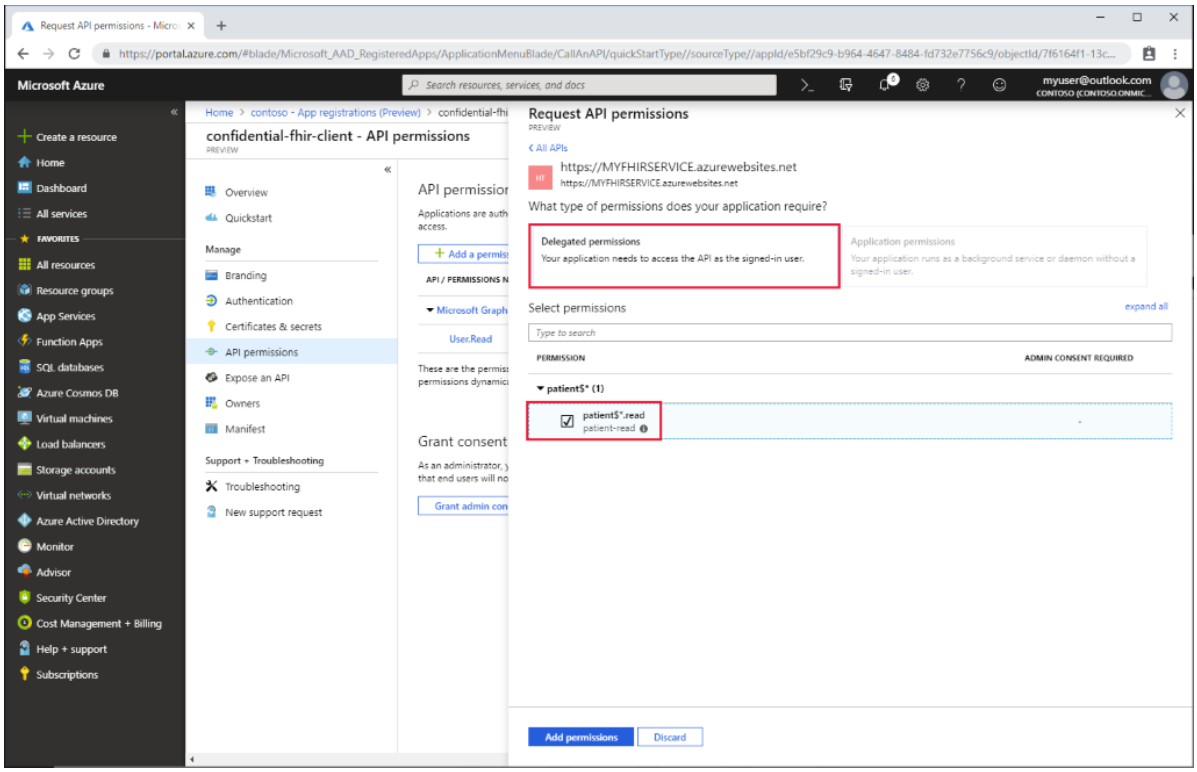
2. Click **Add a permission**

3. Select appropriate resource API:

   For the Azure API for FHIR (managed service), click **APIs my organization uses** and search for "Azure Healthcare APIs". For the Open Source FHIR server for Azure, select your FHIR API Resource Application Registration:

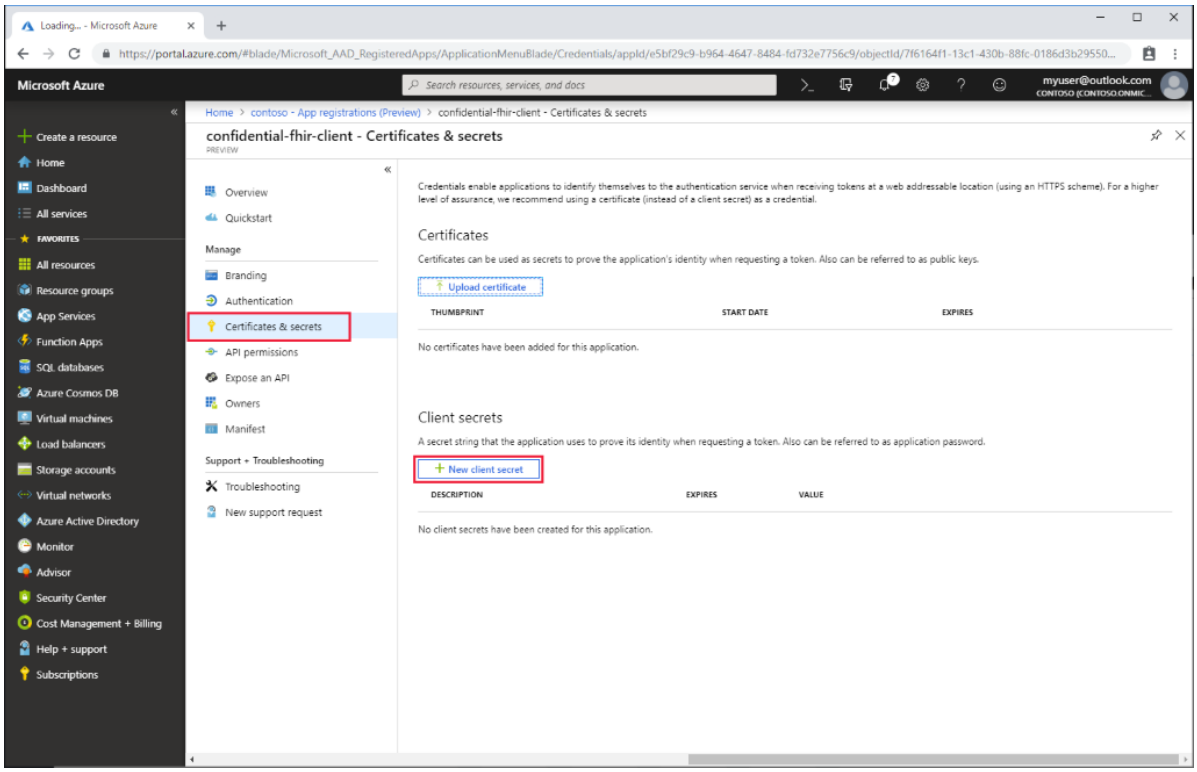


4. Select scopes (permissions) that the confidential application should be able to ask for on behalf of a user:



Application secret

1. Create an application secret (client secret):



2. Provide a description and duration of the secret.

3. Once generated, it will be displayed in the portal only once. Make a note of it and store it securely.

Next steps

In this article, you've learned how to register a confidential client application in Azure Active Directory. You are now ready to deploy the Azure API for FHIR.

Once you have deployed the Azure API for FHIR, you can review additional available settings.

Deploy Azure API for FHIR

Register a public client application in Azure Active Directory

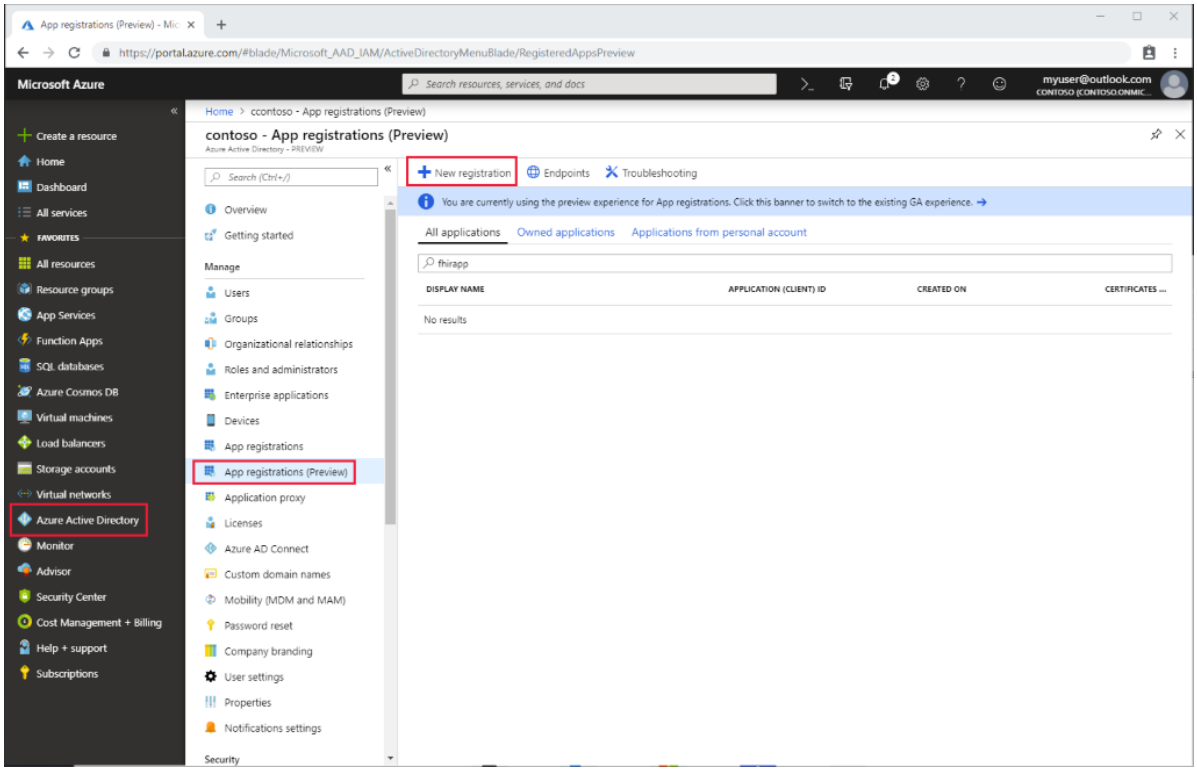
2 minutes to read • Edit Online

In this article, you'll learn how to register a public application in Azure Active Directory.

Client application registrations are Azure Active Directory representations of applications that can authenticate and ask for API permissions on behalf of a user. Public clients are applications such as mobile applications and single page JavaScript applications that can't keep secrets confidential. The procedure is similar to registering a confidential client, but since public clients can't be trusted to hold an application secret, there's no need to add one.

App registrations in Azure portal

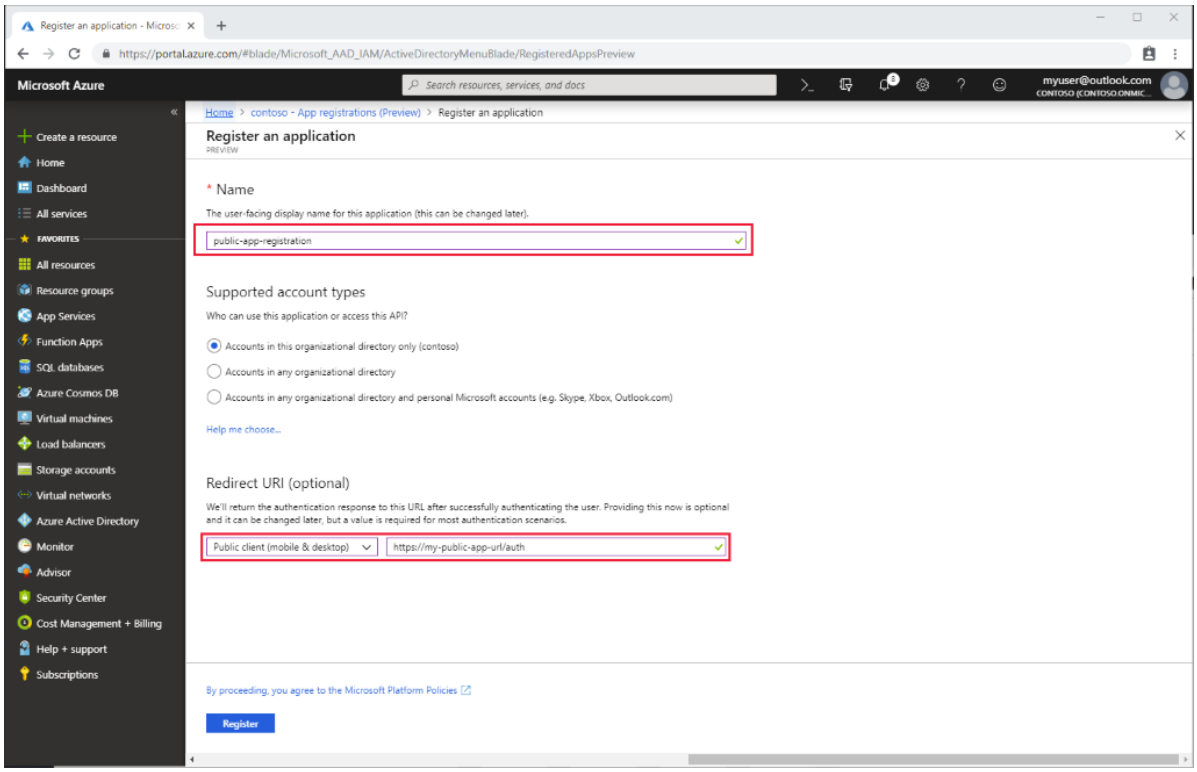
1. In the Azure portal, on the left navigation panel, click **Azure Active Directory**.
2. In the **Azure Active Directory** blade, click **App registrations**:



3. Click the **New registration**.

Application registration overview

1. Give the application a display name.
2. Provide a reply URL. The reply URL is where authentication codes will be returned to the client application. You can add more reply URLs and edit existing ones later.



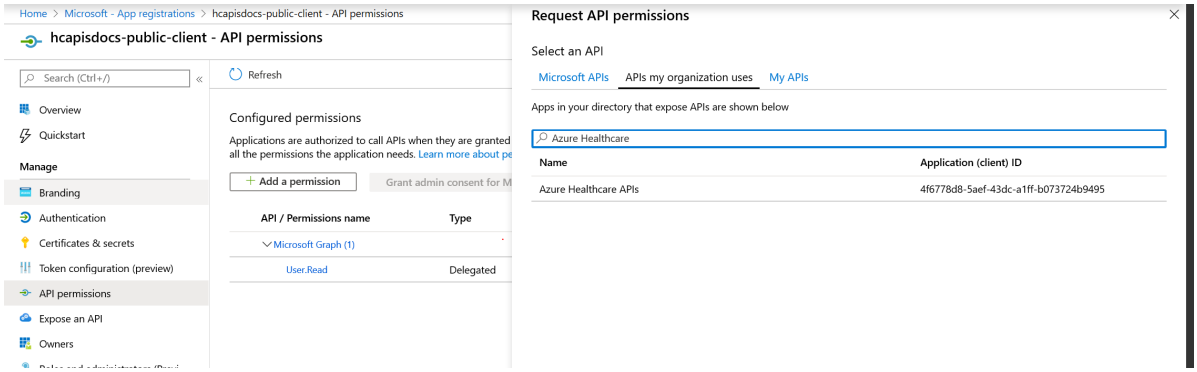
API permissions

Similarly to the confidential client application, you'll need to select which API permissions this application should be able to request on behalf of users:

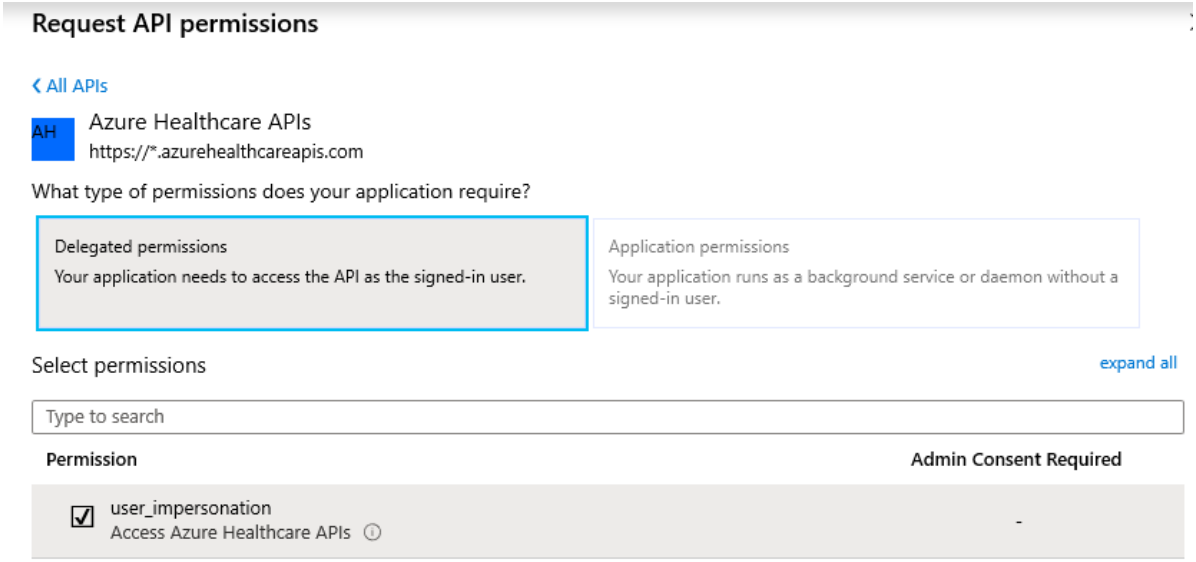
1. Open the **API permissions**.

   If you are using the Azure API for FHIR, you will add a permission to the Azure Healthcare APIs by searching for Azure Healthcare APIs under **APIs my organization uses** (image below).

   If you are referencing a different Resource Application, select your FHIR API Resource Application Registration that you created previously under **My APIs**:



2. Select the permissions that you would like the application to be able to request:



Validate FHIR server authority

If the application you registered in this article and your FHIR server are in the same Azure AD tenant, you are good to proceed to the next steps.

If you configure your client application in a different Azure AD tenant from your FHIR server, you will need to update the **Authority**. In Azure API for FHIR, you do set the Authority under Settings --> Authentication. Set your Authority to **https://login.microsoftonline.com/**.

Next steps

In this article, you've learned how to register a public client application in Azure Active Directory. Next, test access to your FHIR server using Postman.

Access Azure API for FHIR with Postman

Register a service client application in Azure Active Directory

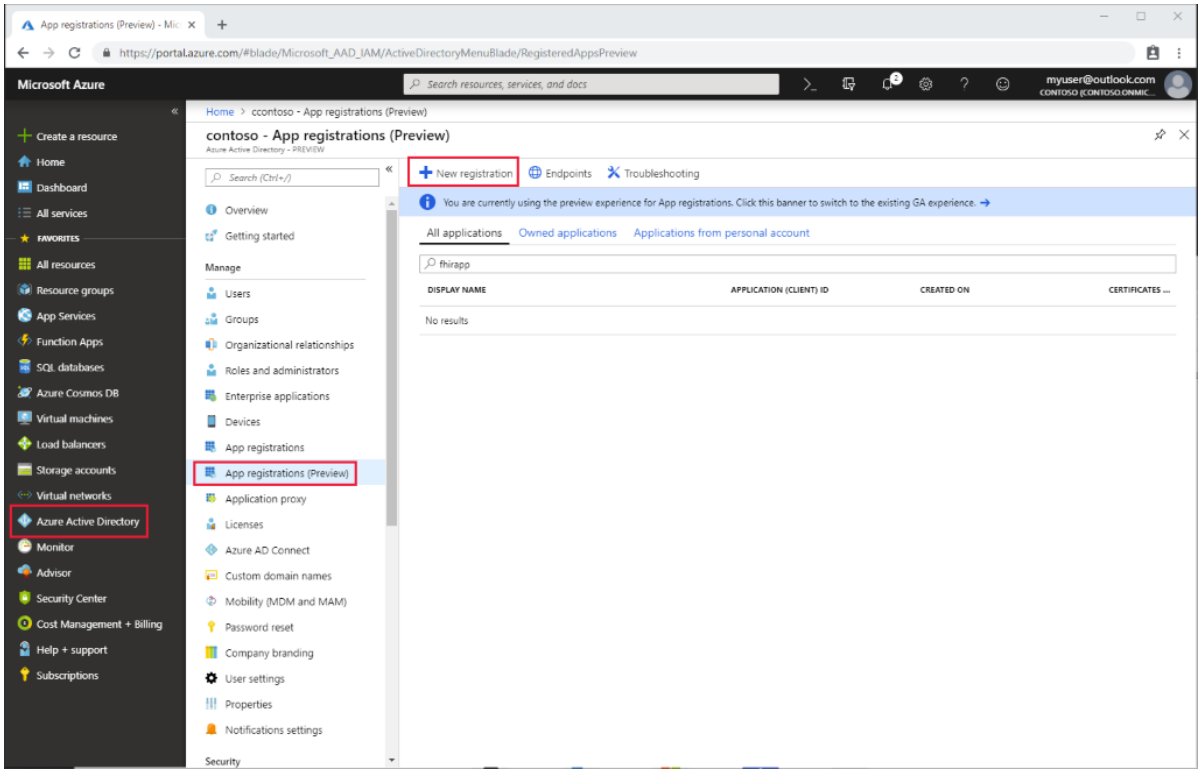
2 minutes to read • Edit Online

In this article, you'll learn how to register a service client application in Azure Active Directory. Client application registrations are Azure Active Directory representations of applications that can be used to authenticate and obtain tokens. A service client is intended to be used by an application to obtain an access token without interactive authentication of a user. It will have certain application permissions and use an application secret (password) when obtaining access tokens.

Follow the steps below to create a new service client.

App registrations in Azure portal

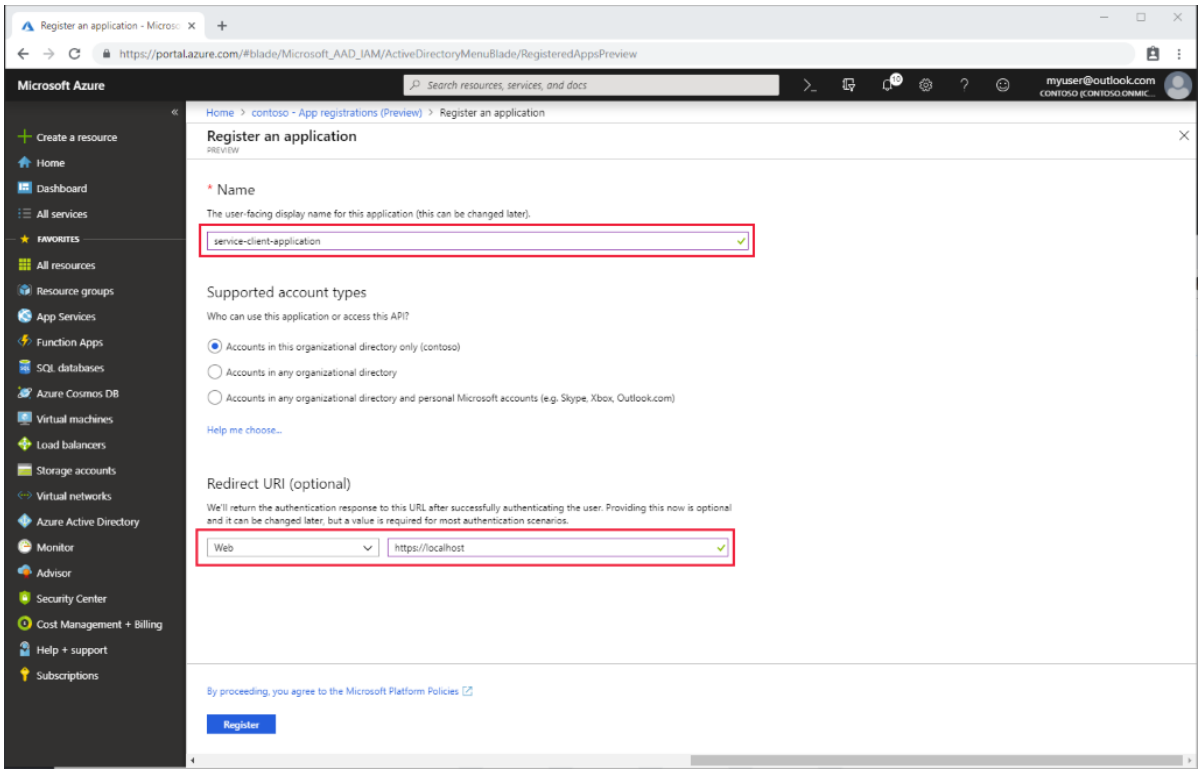
1. In the Azure portal, on the left navigation panel, click **Azure Active Directory**.
2. In the **Azure Active Directory** blade click **App registrations**:



3. Click **New registration**.

Service client application details

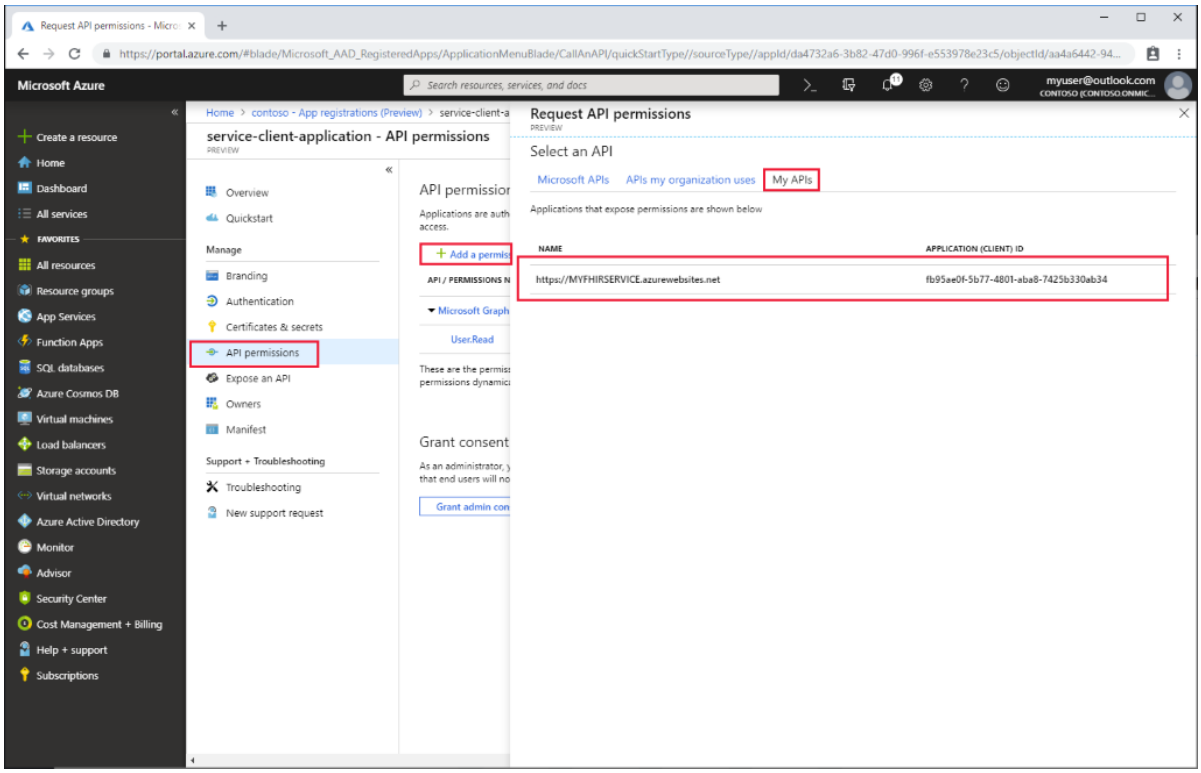
- The service client needs a display name and you can also provide a reply URL but it will typically not be used.



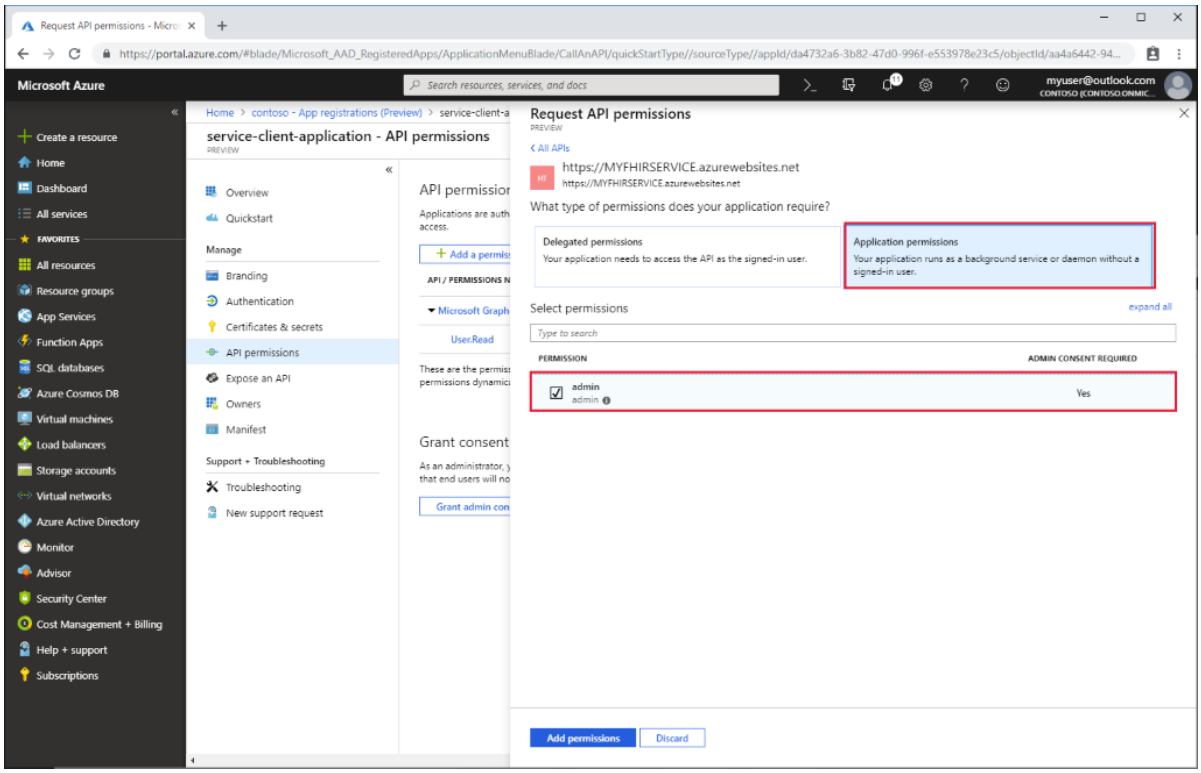
API permissions

You will need to grant the service client application roles.

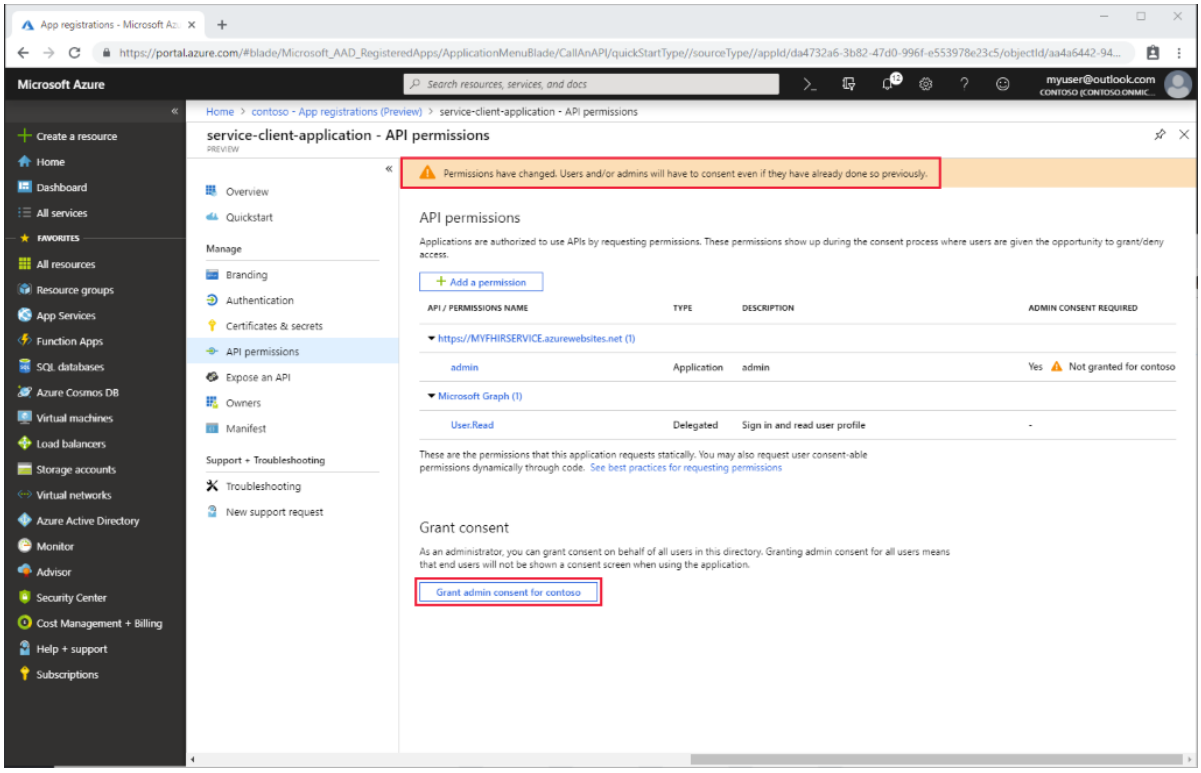
1. Open the **API permissions** and select your FHIR API Resource Application Registration. If you are using the Azure API for FHIR, you will add a permission to the Azure Healthcare APIs by searching for Azure Healthcare APIs under **APIs my organization uses**.



2. Select the application roles you from the ones that are defined on the resource application:



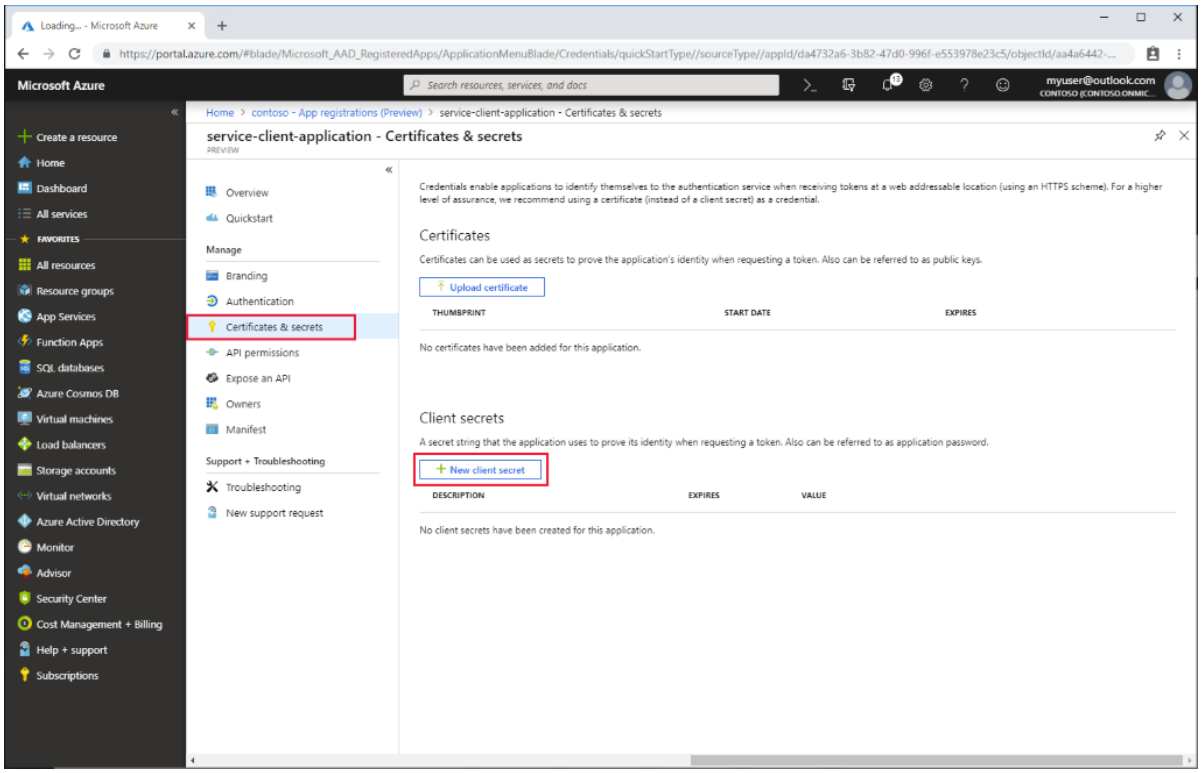
3. Grant consent to the application. If you don't have the permissions required, check with your Azure Active Directory administrator:



Application secret

The service client needs a secret (password), which you will used when obtaining tokens.

1. Click **Certificates & secrets**
2. Click **New client secret**



3. Provide a duration of the secret.

4. Once it has been generated, it will only be displayed once in the portal. Make a note of it and store in a securely.

Next steps

In this article, you've learned how to register a service client application in Azure Active Directory. Next, deploy a FHIR API in Azure.

Deploy Open Source FHIR server

Additional settings for Azure API for FHIR

2 minutes to read • Edit Online

In this how-to guide, we will review the additional settings you may want to set in your Azure API for FHIR. There are additional pages that drill into even more details.

Configure Database settings

Azure API for FHIR uses database to store its data. Performance of the underlying database depends on the number of Request Units (RU) selected during service provisioning or in database settings after the service has been provisioned.

Throughput must be provisioned to ensure that sufficient system resources are available for your database at all times. How many RUs you need for your application depends on operations you perform. Operations can range from simple read and writes to more complex queries.

For more information on how to change the default settings, see configure database settings.

Find identity object IDs

The fully managed Azure API for FHIR service is configured to allow access for only a pre-defined list of identity object IDs. When an application or user is trying to access the FHIR API, a bearer token must be presented. This bearer token will have certain claims (fields). In order to grant access to the FHIR API, the token must contain the right issuer (`iss`), audience (`aud`), and an object ID (`oid`) from a list of allowed object IDs. An identity object ID is either the object ID of a user or a service principal in Azure Active Directory.

When you create a new Azure API for FHIR instance, you can configure a list of allowed object IDs. To configure this list, see our how-to-guide to find identity object IDs.

Enable diagnostic logging

You may want to enable diagnostic logging as part of your setup to be able to monitor your service and have accurate reporting for compliance purposes. For details on how to set up diagnostic logging, see our how-to-guide on how to set up diagnostic logging, along with some sample queries.

Use custom headers to add data to audit logs

In the Azure API for FHIR, you may want to include additional information in the logs, which comes from the calling system. To do including this information, you can use custom headers.

You can use custom headers to capture several types of information. For example:

- Identity or authorization information
- Origin of the caller
- Originating organization
- Client system details (electronic health record, patient portal)

To add this data to your audit logs, see the Use Custom HTTP headers to add data to Audit Logs how-to-guide.

Next steps

In this how-to guide, you set up additional settings for the Azure API for FHIR.

Next check out the series of tutorials to create a web application that reads FHIR data.

Deploy JavaScript application

Configure Database settings

2 minutes to read • Edit Online

Azure API for FHIR uses database to store its data. Performance of the underlying database depends on the number of Request Units (RU) selected during service provisioning or in database settings after the service has been provisioned.

Azure API for FHIR borrows the concept of RUs from Cosmos DB (see Request Units in Azure Cosmos DB) when setting the performance of underlying database.

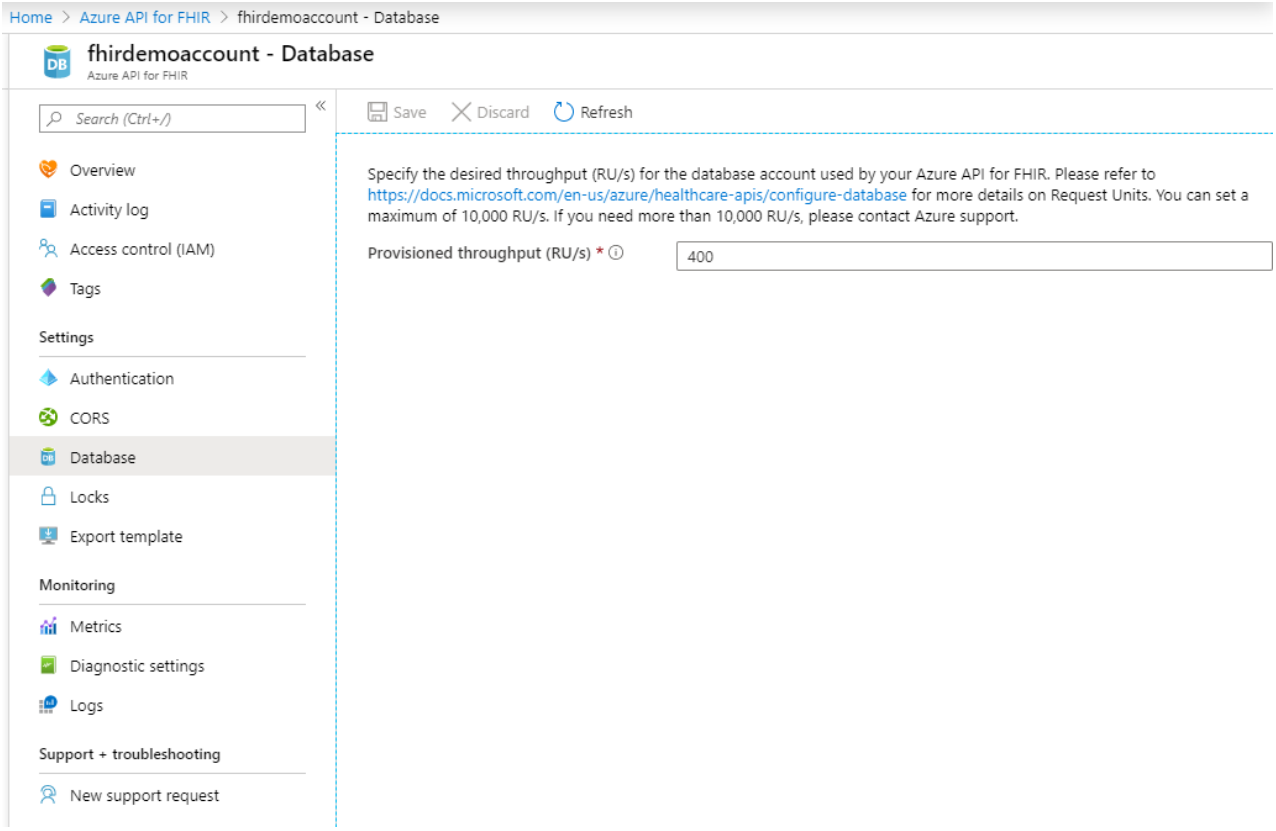
Throughput must be provisioned to ensure that sufficient system resources are available for your database at all times. How many RUs you need for your application depends on operations you perform. Operations can range from simple read and writes to more complex queries.

> **NOTE**
> As different operations consume different number of RU, we return the actual number of RUs consumed in every API call in response header. This way you can profile the number of RUs consumed by your application.

Update throughput

To change this setting in the Azure portal, navigate to your Azure API for FHIR and open the Database blade. Next, change the Provisioned throughput to the desired value depending on your performance needs. You can change the value up to a maximum of 10,000 RU/s. If you need a higher value, contact Azure support.

> **NOTE**
> Higher value means higher Azure API for FHIR throughput and higher cost of the service.



Next steps

In this article, you learned how to update your RUs for Azure API for FHIR. Next deploy a fully managed Azure API for FHIR:

Deploy Azure API for FHIR

Configure cross-origin resource sharing in Azure API for FHIR

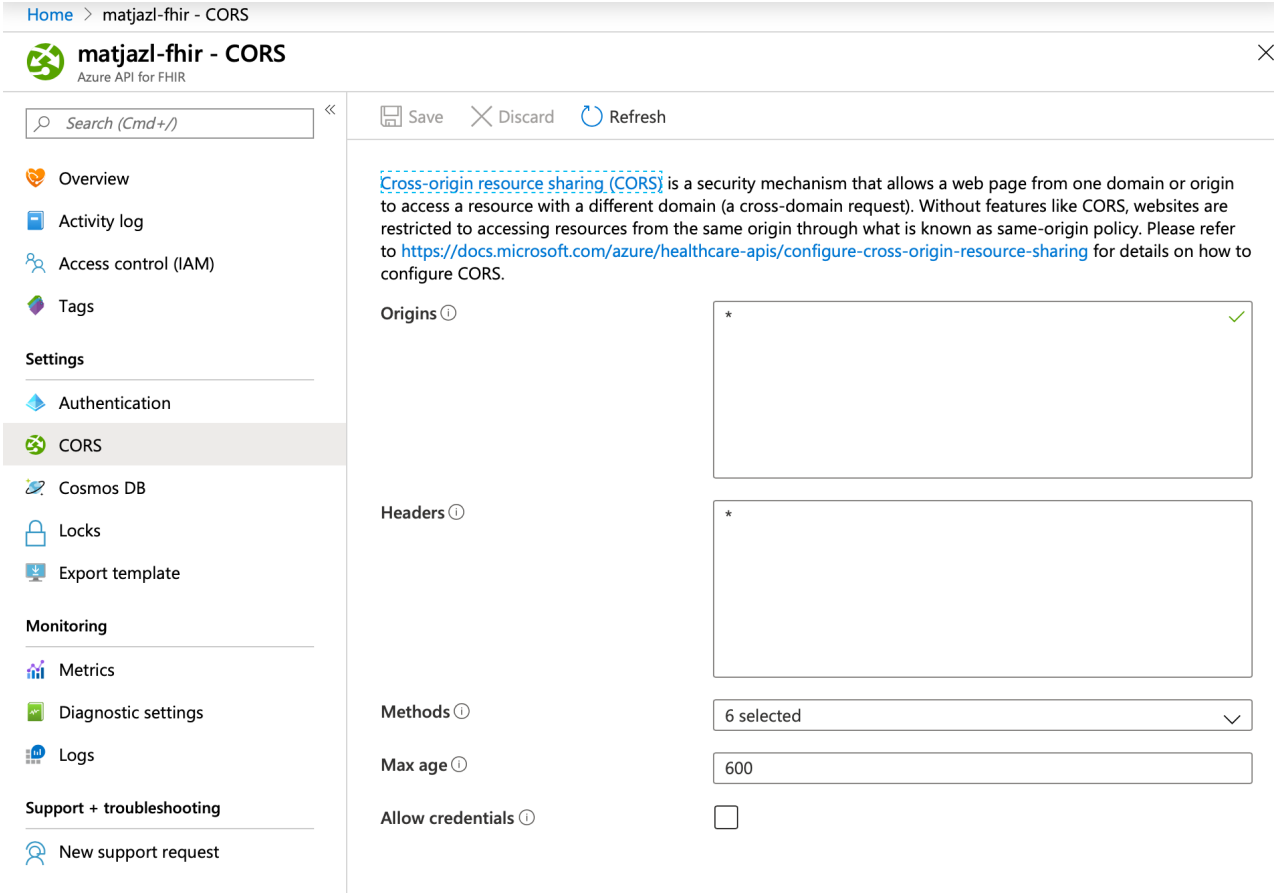
2 minutes to read • Edit Online

Azure API for Fast Healthcare Interoperability Resources (FHIR) supports cross-origin resource sharing (CORS). CORS allows you to configure settings so that applications from one domain (origin) can access resources from a different domain, known as a cross-domain request.

CORS is often used in a single-page app that must call a RESTful API to a different domain.

To configure a CORS setting in the Azure API for FHIR, specify the following settings:

- **Origins (Access-Control-Allow-Origin)**. A list of domains allowed to make cross-origin requests to the Azure API for FHIR. Each domain (origin) must be entered in a separate line. You can enter an asterisk (*) to allow calls from any domain, but we don't recommend it because it's a security risk.
- **Headers (Access-Control-Allow-Headers)**. A list of headers that the origin request will contain. To allow all headers, enter an asterisk (*).
- **Methods (Access-Control-Allow-Methods)**. The allowed methods (PUT, GET, POST, and so on) in an API call. Choose **Select all** for all methods.
- **Max age (Access-Control-Max-Age)**. The value in seconds to cache preflight request results for Access-Control-Allow-Headers and Access-Control-Allow-Methods.
- **Allow credentials (Access-Control-Allow-Credentials)**. CORS requests normally don't include cookies to prevent cross-site request forgery (CSRF) attacks. If you select this setting, the request can be made to include credentials, such as cookies. You can't configure this setting if you already set Origins with an asterisk (*).



NOTE

You can't specify different settings for different domain origins. All settings (**Headers**, **Methods**, **Max age**, and **Allow credentials**) apply to all origins specified in the Origins setting.

Next steps

In this article, you learned how to configure cross-origin sharing in Azure API for FHIR. Next deploy a fully managed Azure API for FHIR:

Deploy Azure API for FHIR

Configure export setting and export the data to a storage account

2 minutes to read • Edit Online

Azure API for FHIR supports $export command that allows you to export the data out of Azure API for FHIR account to a storage account.

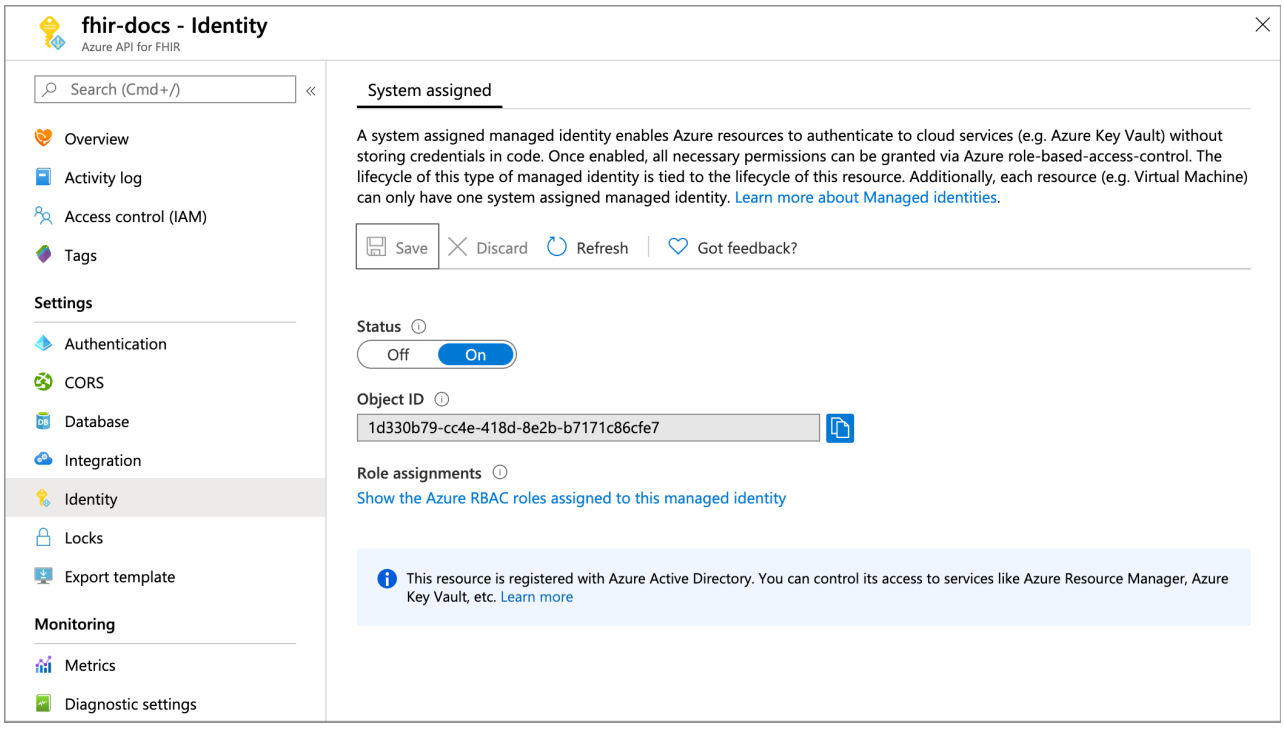
There are four steps involved in performing export in Azure API for FHIR:

1. Enable Managed Identity on Azure API for FHIR Service
2. Creating a Azure storage account (if not done before) and assigning permission to Azure API for FHIR to the storage account
3. Selecting the storage account in Azure API for FHIR as export storage account
4. Executing the export by invoking $export command on Azure API for FHIR

Enabling Managed Identity on Azure API for FHIR

First step in configuring Azure API for FHIR for export is to enable system wide managed identity on the service. You can read all about Managed Identities in Azure here.

To do so, navigate to Azure API for FHIR service and select Identity blade. Changing the status to On will enable managed identity in Azure API for FHIR Service.

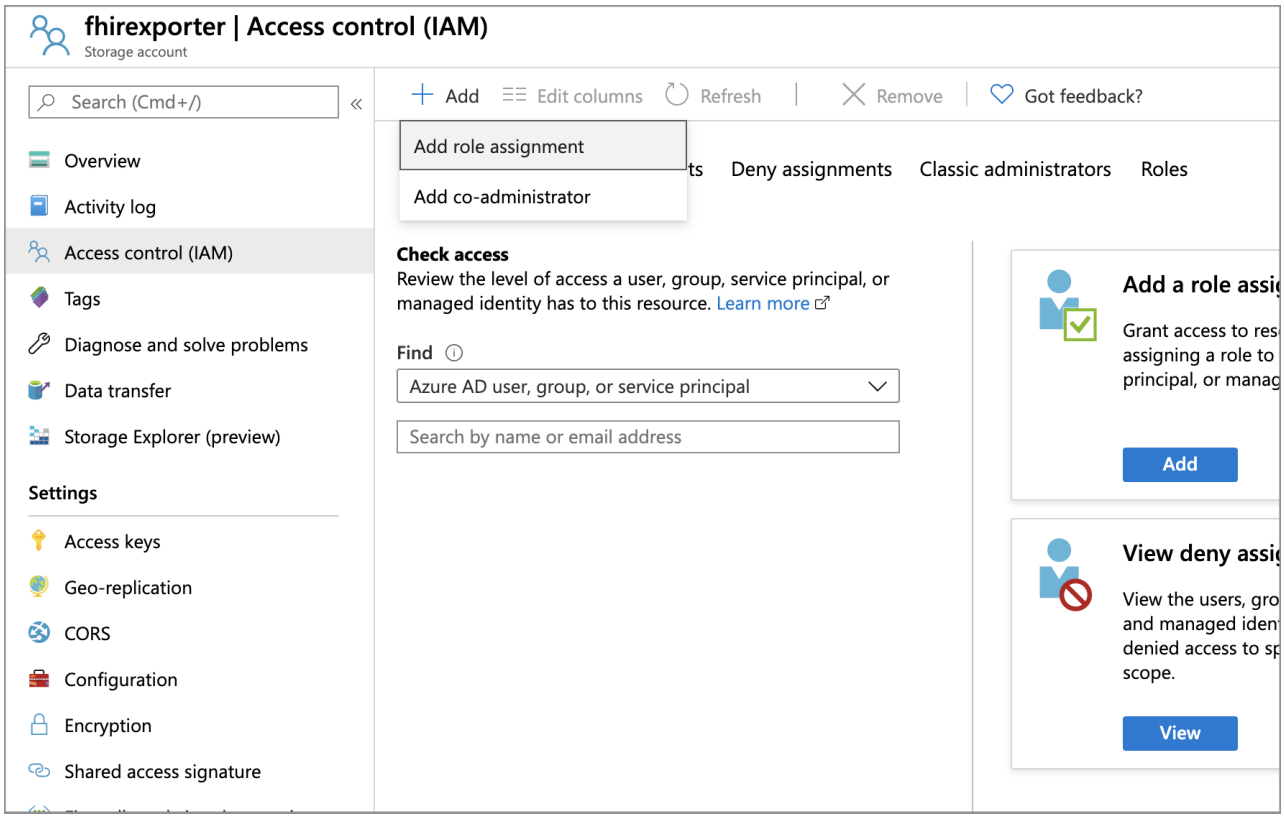


Now we can move to next step and create a storage account and assign permission to our service.

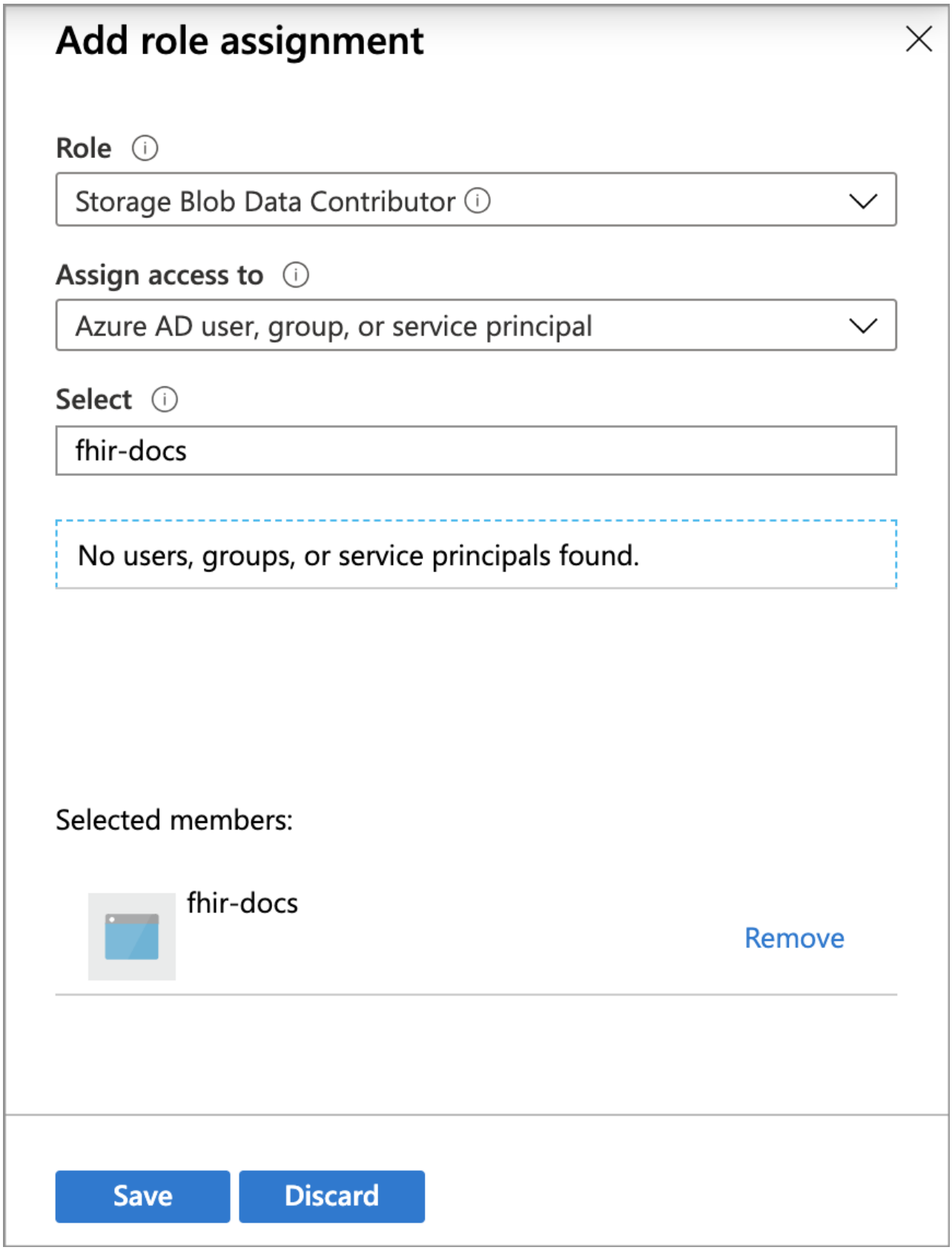
Adding permission to storage account

Next step in export is to assign permission for Azure API for FHIR service to write to the storage account.

After we have created a storage account, navigate to Access Control (IAM) blade in Storage Account and select Add Role Assignments



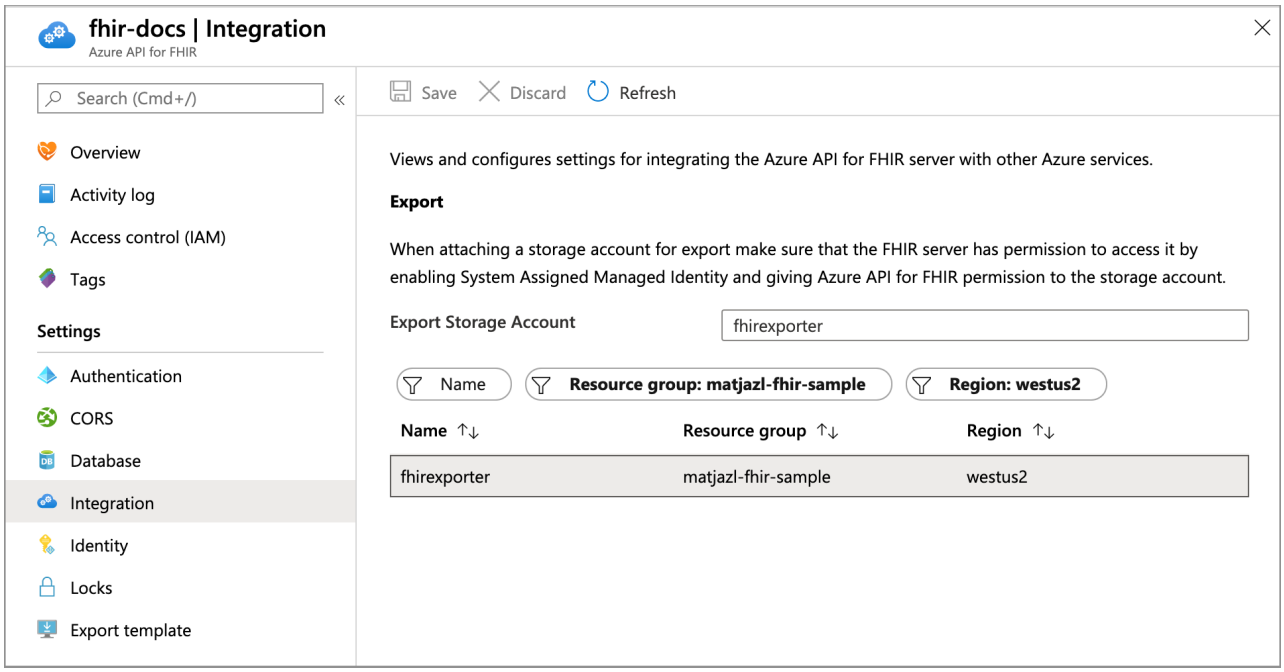
Here we then add role Storage Blob Data Contributor to our service name.



Now we are ready for next step where we can select the storage account in Azure API for FHIR as a default storage account for $export.

Selecting the storage account for $export

Final step before invoking $export command is to assign the Azure storage account that Azure API for FHIR will use to export the data to. To do this, navigate to Integration blade in Azure API for FHIR service in Azure portal and select the storage account



After that we are ready to export the data using $export command.

Exporting the data using $export command

After we have configured Azure API for FHIR for export, we can now go and use the $export command to export the data out of the service into the storage account we specified. To learn how to invoke $export command in FHIR server, please read documentation on $export specification at https://hl7.org/Fhir/uv/bulkdata/export/index.html

> **IMPORTANT**
> Note that currently Azure API for FHIR only supports System Level Export as defined in Export specification at https://hl7.org/Fhir/uv/bulkdata/export/index.html. We also currently do not support query parameters with the $export.

Additional Settings

Find identity object IDs for authentication configuration

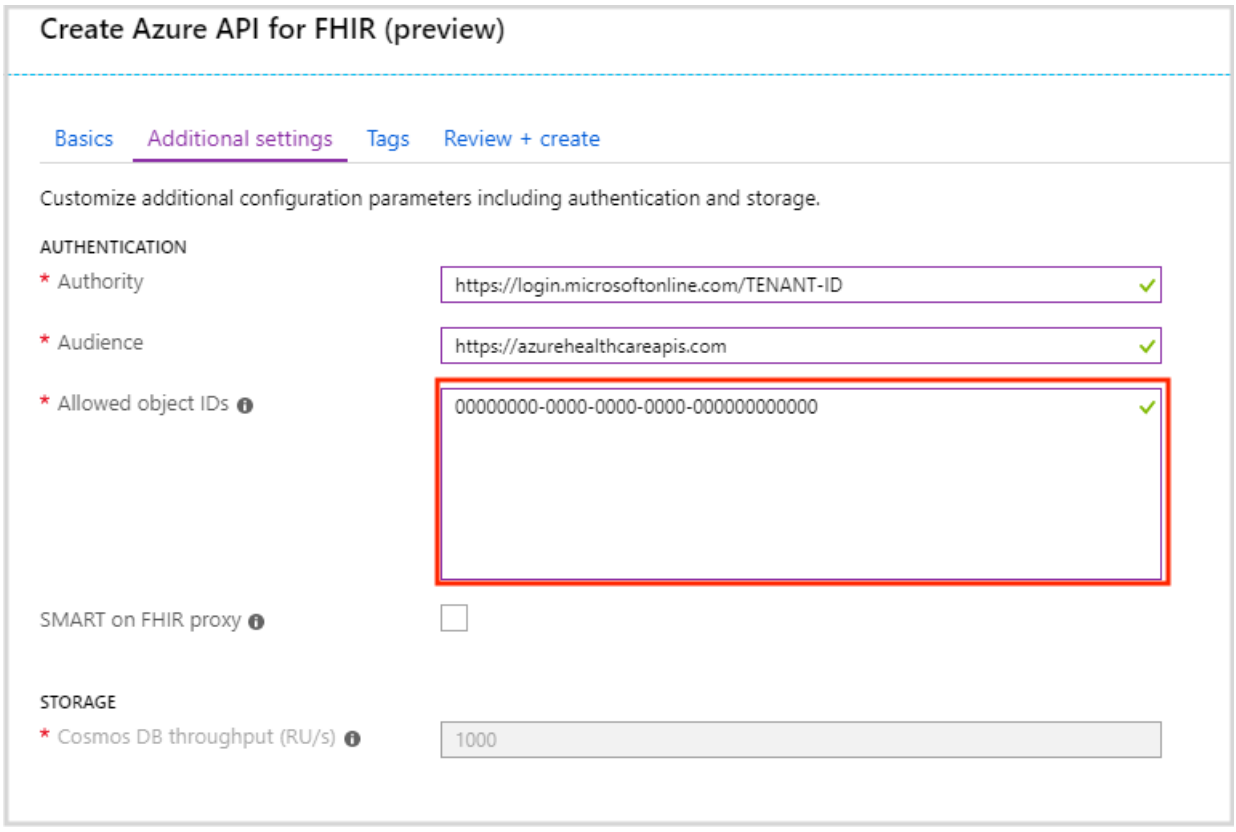
2 minutes to read • Edit Online

In this article, you'll learn how to find identity object IDs needed to configure the list of allowed identity object IDs for the Azure API for FHIR.

The fully managed Azure API for FHIR® service is configured to allow access for only a pre-defined list of identity object IDs. When an application or user is trying to access the FHIR API, a bearer token must be presented. This bearer token will have certain claims (fields). In order to grant access to the FHIR API, the token must contain the right issuer (`iss`), audience (`aud`), and an object ID (`oid`) from a list of allowed object IDs. An identity object ID is either the object ID of a user or a service principal in Azure Active Directory.

Configure list of allowed object IDs

When you create a new Azure API for FHIR instance, you can configure a list of allowed object IDs:



These object IDs can either be IDs for specific users or service principals in your Azure Active Directory.

Find user object ID

If you have a user with user name `myuser@consoso.com`, you can locate the users `ObjectId` using the following PowerShell command:

```
$(Get-AzureADUser -Filter "UserPrincipalName eq 'myuser@consoso.com'").ObjectId
```

or you can use the Azure CLI:

```
az ad user show --upn-or-object-id myuser@consoso.com | jq -r .objectId
```

Find service principal object ID

Suppose you have registered a service client app and you would like to allow this service client to access the Azure API for FHIR, you can find the object ID for the client service principal with the following PowerShell command:

```
$(Get-AzureADServicePrincipal -Filter "AppId eq 'XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX'").ObjectId
```

where `XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX` is the service client application ID. Alternatively, you can use the `DisplayName` of the service client:

```
$(Get-AzureADServicePrincipal -Filter "DisplayName eq 'testapp'").ObjectId
```

If you are using the Azure CLI, you can use:

```
az ad sp show --id XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX | jq -r .objectId
```

Next steps

In this article, you've learned how to find identity object IDs needed to configure the identities that are allowed to access an Azure API for FHIR instance. Next deploy a fully managed Azure API for FHIR:

Deploy Azure API for FHIR

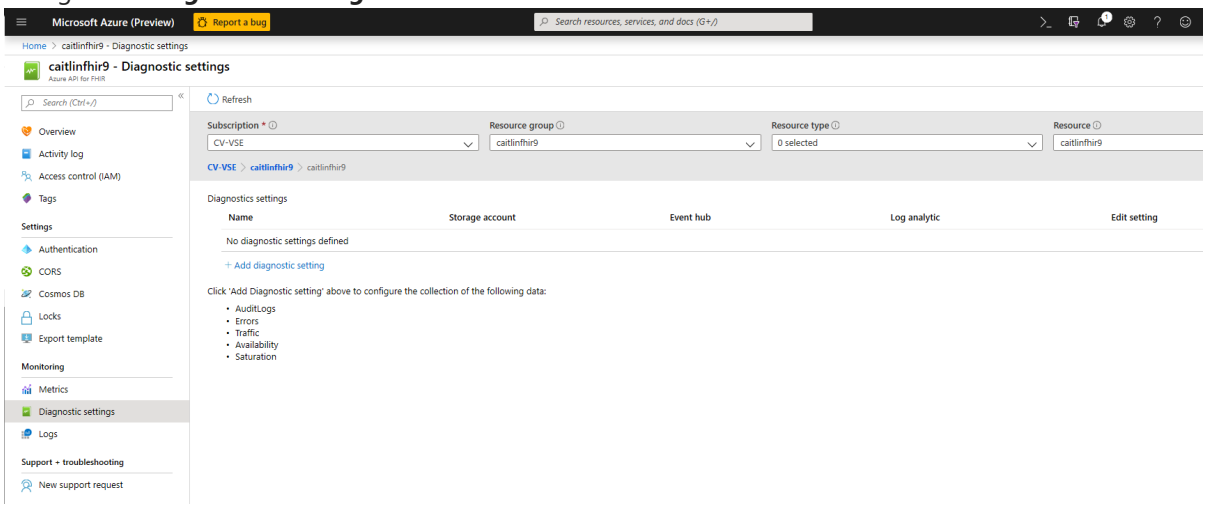
Enable Diagnostic Logging in Azure API for FHIR®

2 minutes to read • Edit Online

In this article, you will learn how to enable diagnostic logging in Azure API for FHIR® and be able to review some sample queries for these logs. Access to diagnostic logs is essential for any healthcare service where compliance with regulatory requirements (such as HIPAA) is a must. The feature in Azure API for FHIR® that enables diagnostic logs is the **Diagnostic settings** in the Azure portal.

Enable audit logs

1. To enable diagnostic logging in Azure API for FHIR®, select your Azure API for FHIR® service in the Azure portal
2. Navigate to **Diagnostic settings**



3. Select **+ Add diagnostic setting**
4. Enter a name for the setting
5. Select the method you want to use to access your diagnostic logs:
   a. **Archive to a storage account** for auditing or manual inspection. The storage account you want to use needs to be already created.
   b. **Stream to event hub** for ingestion by a third-party service or custom analytic solution. You will need to create an event hub namespace and event hub policy before you can configure this step.
   c. **Stream to the Log Analytics** workspace in Azure Monitor. You will need to create your Logs Analytics Workspace before you can select this option.
6. Select **AuditLogs** and any metrics you want to capture
7. Click Save

> **NOTE**
> It might take up to 15 minutes for the first Logs to show in Log Analytics.

For more information on how to work with diagnostic logs, please refer to the Azure Resource Log documentation

Audit log details

At this time, the Azure API for FHIR ® service returns the following fields in the audit log:

FIELD NAME	TYPE	NOTES
CallerIdentity	Dynamic	A generic property bag containing identity information
CallerIdentityIssuer	String	Issuer
CallerIdentityObjectId	String	Object_Id
CallerIPAddress	String	The caller's IP address
CorrelationId	String	Correlation ID
FhirResourceType	String	The resource type for which the operation was executed
LogCategory	String	The log category (we are currently returning 'AuditLogs' LogCategory)
Location	String	The location of the server that processed the request (e.g., South Central US)
OperationDuration	Int	The time it took to complete this request in seconds
OperationName	String	Describes the type of operation (e.g. update, search-type)
RequestUri	String	The request URI
ResultType	String	The available values currently are **Started**, **Succeeded**, or **Failed**
StatusCode	Int	The HTTP status code. (e.g., 200)
TimeGenerated	DateTime	Date and time of the event
Properties	String	Describes the properties of the fhirResourceType
SourceSystem	String	Source System (always Azure in this case)
TenantId	String	Tenant ID
Type	String	Type of log (always MicrosoftHealthcareApisAuditLog in this case)
_ResourceId	String	Details about the resource

Sample queries

Here are a few basic Application Insights queries you can use to explore your log data.

Run this query to see the **100 most recent** logs:

```
MicrosoftHealthcareApisAuditLogs
| limit 100
```

Run this query to group operations by **FHIR Resource Type**:

```
MicrosoftHealthcareApisAuditLogs
| summarize count() by FhirResourceType
```

Run this query to get all the **failed results**

```
MicrosoftHealthcareApisAuditLogs
| where ResultType == "Failed"
```

Conclusion

Having access to diagnostic logs is essential for monitoring a service and providing compliance reports. Azure API for FHIR® allows you to do these actions through diagnostic logs.

FHIR® is the registered trademark of HL7 and is used with the permission of HL7.

Next steps

In this article, you learned how to enable Audit Logs for Azure API for FHIR®. Next, learn about other additional settings you can configure in the Azure API for FHIR

Additional Settings

Get access token for Azure API for FHIR using Azure CLI

2 minutes to read • Edit Online

In this article, you'll learn how to obtain an access token for the Azure API for FHIR using the Azure CLI. When you provision the Azure API for FHIR, you configure a set of users or service principals that have access to the service. If your user object ID is in the list of allowed object IDs, you can access the service using a token obtained using the Azure CLI.

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select **Try It** in the upper-right corner of a code block. Selecting **Try It** doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl**+**Shift**+**V** on Windows and Linux or by selecting **Cmd**+**Shift**+**V** on macOS.
4. Select **Enter** to run the code.

Sign in with Azure CLI

Before you can obtain a token, you need to sign in with the user that you want to obtain a token for:

```
az login
```

Obtain a token

The Azure API for FHIR uses a `resource` or `Audience` with URI equal to the URI of the FHIR server `https://<FHIR ACCOUNT NAME>.azurehealthcareapis.com`. You can obtain a token and store it in a variable (named

`$token`) with the following command:

```
token=$(az account get-access-token --resource=https://<FHIR ACCOUNT NAME>.azurehealthcareapis.com | jq -r
.accessToken)
```

Use with Azure API for FHIR

```
curl -X GET --header "Authorization: Bearer $token" https://<FHIR ACCOUNT
NAME>.azurehealthcareapis.com/Patient
```

Next steps

In this article, you've learned how to obtain an access token for the Azure API for FHIR using the Azure CLI. To learn how to access the FHIR API using Postman, proceed to the Postman tutorial.

Access FHIR API using Postman

Azure Active Directory identity configuration for Azure API for FHIR

4 minutes to read • Edit Online

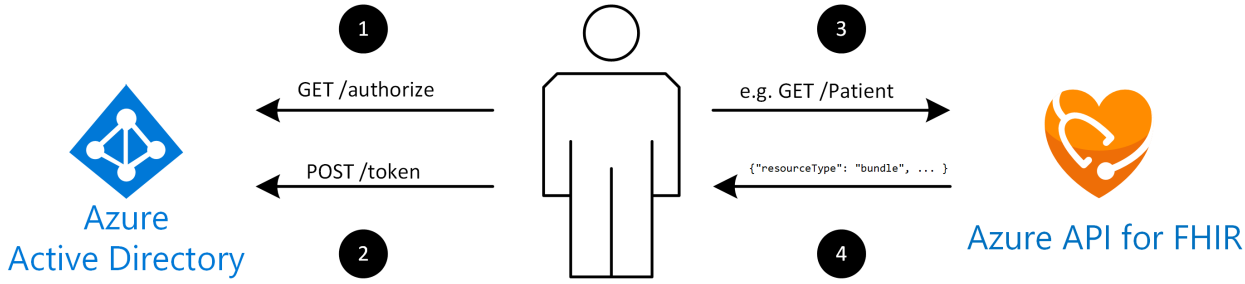
An important piece when working with healthcare data is to ensure that the data is secure and cannot be accessed by unauthorized users or applications. FHIR servers use OAuth 2.0 to ensure this data security. The Azure API for FHIR is secured using Azure Active Directory, which is an example of an OAuth 2.0 identity provider. This article provides an overview of FHIR server authorization and the steps needed to obtain a token to access a FHIR server. While these steps will apply to any FHIR server and any identity provider, we will walk through Azure API for FHIR as the FHIR server and Azure AD as our identity provider in this article.

Access control overview

In order for a client application to access Azure API for FHIR, it must present an access token. The access token is a signed, Base64 encoded collection of properties (claims) that convey information about the client's identity and roles and privileges granted to the client.

There are a number of ways to obtain a token, but the Azure API for FHIR doesn't care how the token is obtained as long as it's an appropriately signed token with the correct claims.

Using authorization code flow as an example, accessing a FHIR server goes through the four steps below:



1. The client sends a request to the `/authorize` endpoint of Azure AD. Azure AD will redirect the client to a sign-in page where the user will authenticate using appropriate credentials (for example username and password or two-factor authentication). See details on obtaining an authorization code. Upon successful authentication, an *authorization code* is returned to the client. Azure AD will only allow this authorization code to be returned to a registered reply URL configured in the client application registration (see below).
2. The client application exchanges the authorization code for an *access token* at the `/token` endpoint of Azure AD. When requesting a token, the client application may have to provide a client secret (the applications password). See details on obtaining an access token.
3. The client makes a request to the Azure API for FHIR, for example `GET /Patient` to search all patients. When making the request, it includes the access token in an HTTP request header, for example `Authorization: Bearer eyJ0e...`, where `eyJ0e...` represents the Base64 encoded access token.
4. The Azure API for FHIR validates that the token contains appropriate claims (properties in the token). If everything checks out, it will complete the request and return a FHIR bundle with results to the client.

It is important to note that the Azure API for FHIR isn't involved in validating user credentials and it doesn't issue the token. The authentication and token creation is done by Azure AD. The Azure API for FHIR simply validates that the token is signed correctly (it is authentic) and that it has appropriate claims.

Structure of an access token

Development of FHIR applications often involves debugging access issues. If a client is denied access to the Azure API for FHIR, it's useful to understand the structure of the access token and how it can be decoded to inspect the contents (the claims) of the token.

FHIR servers typically expect a JSON Web Token (JWT, sometimes pronounced "jot"). It consists of three parts:

1. A header, which could look like:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

2. The payload (the claims), for example:

```
{
 "oid": "123",
 "iss": "https://issuerurl",
 "iat": 1422779638,
 "roles": [
    "admin"
  ]
}
```

3. A signature, which is calculated by concatenating the Base64 encoded contents of the header and the payload and calculating a cryptographic hash of them based on the algorithm (`alg`) specified in the header. A server will be able to obtain public keys from the identity provider and validate that this token was issued by a specific identity provider and it hasn't been tampered with.

The full token consists of the Base64 encoded (actually Base64 url encoded) versions of those three segments. The three segments are concatenated and separated with a `.` (dot).

An example token is seen below:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJvaWQiOiIxMjMiLCAiaXNzIjoiaHR0cHM6Ly9pc3N1ZXJ1cmwiLCJpYXQiOjE0MjI3Nzk2M
zgsInJvbGVzIjpbImFkbWluIl19.gzSraSYS8EXBxLN_oWnFSRgCzcmJmMjLiuyu5CSpyHI
```

The token can be decoded and inspected with tools such as https://jwt.ms. The result of decoding the token is:

```
{
  "alg": "HS256",
  "typ": "JWT"
}.{
  "oid": "123",
  "iss": "https://issuerurl",
  "iat": 1422779638,
  "roles": [
    "admin"
  ]
}.[Signature]
```

Obtaining an access token

As mentioned above, there are several ways to obtain a token from Azure AD. They are described in detail in the Azure AD developer documentation.

Azure AD has two different versions of the OAuth 2.0 endpoints, which are referred to as `v1.0` and `v2.0`. Both of these versions are OAuth 2.0 endpoints and the `v1.0` and `v2.0` designations refer to differences in how Azure AD implements that standard.

When using a FHIR server, you can use either the `v1.0` or the `v2.0` endpoints. The choice may depend on the authentication libraries you are using in your client application.

The pertinent sections of the Azure AD documentation are:

- `v1.0` endpoint:
  - Authorization code flow.
  - Client credentials flow.
- `v2.0` endpoint:
  - Authorization code flow.
  - Client credentials flow.

There are other variations (for example on behalf of flow) for obtaining a token. Check the Azure AD documentation for details. When using the Azure API for FHIR, there are also some shortcuts for obtaining an access token (for debugging purposes) using the Azure CLI.

Next steps

In this document, you learned some of the basic concepts involved in securing access to the Azure API for FHIR using Azure AD. To learn how to deploy an instance of the Azure API for FHIR, continue to the deployment quickstart.

Deploy Azure API for FHIR

Azure API for FHIR access token validation

2 minutes to read • Edit Online

How Azure API for FHIR validates the access token will depend on implementation and configuration. In this article, we will walk through the validation steps, which can be helpful when troubleshooting access issues.

Validate token has no issues with identity provider

The first step in the token validation is to verify that the token was issued by the correct identity provider and that it hasn't been modified. The FHIR server will be configured to use a specific identity provider known as the authority `Authority`. The FHIR server will retrieve information about the identity provider from the `/.well-known/openid-configuration` endpoint. When using Azure AD, the full URL would be:

```
GET https://login.microsoftonline.com/<TENANT-ID>/.well-known/openid-configuration
```

where `<TENANT-ID>` is the specific Azure AD tenant (either a tenant ID or a domain name).

Azure AD will return a document like the one below to the FHIR server.

```
{
    "authorization_endpoint": "https://login.microsoftonline.com/<TENANT-ID>/oauth2/authorize",
    "token_endpoint": "https://login.microsoftonline.com/<TENANT-ID>/oauth2/token",
    "token_endpoint_auth_methods_supported": [
        "client_secret_post",
        "private_key_jwt",
        "client_secret_basic"
    ],
    "jwks_uri": "https://login.microsoftonline.com/common/discovery/keys",
    "response_modes_supported": [
        "query",
        "fragment",
        "form_post"
    ],
    "subject_types_supported": [
        "pairwise"
    ],
    "id_token_signing_alg_values_supported": [
        "RS256"
    ],
    "http_logout_supported": true,
    "frontchannel_logout_supported": true,
    "end_session_endpoint": "https://login.microsoftonline.com/<TENANT-ID>/oauth2/logout",
    "response_types_supported": [
        "code",
        "id_token",
        "code id_token",
        "token id_token",
        "token"
    ],
    "scopes_supported": [
        "openid"
    ],
    "issuer": "https://sts.windows.net/<TENANT-ID>/",
    "claims_supported": [
        "sub",
        "iss",
        "cloud_instance_name",
        "cloud_instance_host_name",
        "cloud_graph_host_name",
        "msgraph_host",
        "aud",
        "exp",
        "iat",
        "auth_time",
        "acr",
        "amr",
        "nonce",
        "email",
        "given_name",
        "family_name",
        "nickname"
    ],
    "microsoft_multi_refresh_token": true,
    "check_session_iframe": "https://login.microsoftonline.com/<TENANT-ID>/oauth2/checksession",
    "userinfo_endpoint": "https://login.microsoftonline.com/<TENANT-ID>/openid/userinfo",
    "tenant_region_scope": "WW",
    "cloud_instance_name": "microsoftonline.com",
    "cloud_graph_host_name": "graph.windows.net",
    "msgraph_host": "graph.microsoft.com",
    "rbac_url": "https://pas.windows.net"
}
```


The important properties for the FHIR server are `jwks_uri`, which tells the server where to fetch the encryption keys needed to validate the token signature and `issuer`, which tells the server what will be in the issuer claim (`iss`) of tokens issued by this server. The FHIR server can use this to validate that it is receiving an authentic token.

Validate claims of the token

Once the server has verified the authenticity of the token, the FHIR server will then proceed to validate that the client has the required claims to access the token.

When using the Azure API for FHIR, the server will validate:

1. The token has the right `Audience` (`aud` claim).
2. The `oid` claim contains an identity object ID, which is in the list of allowed object IDs.

See details on finding identity object IDs.

When using the OSS Microsoft FHIR server for Azure, the server will validate:

1. The token has the right `Audience` (`aud` claim).
2. The token has a role in the `roles` claim, which is allowed access to the FHIR server.

Consult details on how to define roles on the FHIR server.

A FHIR server may also validate that an access token has the scopes (in token claim `scp`) to access the part of the FHIR API that a client is trying to access. Currently, the Azure API for FHIR and the FHIR server for Azure do not validate token scopes.

Next steps

Now that you know how to walk through token validation, you can complete the tutorial to create a JavaScript application and read FHIR data.

Web application tutorial

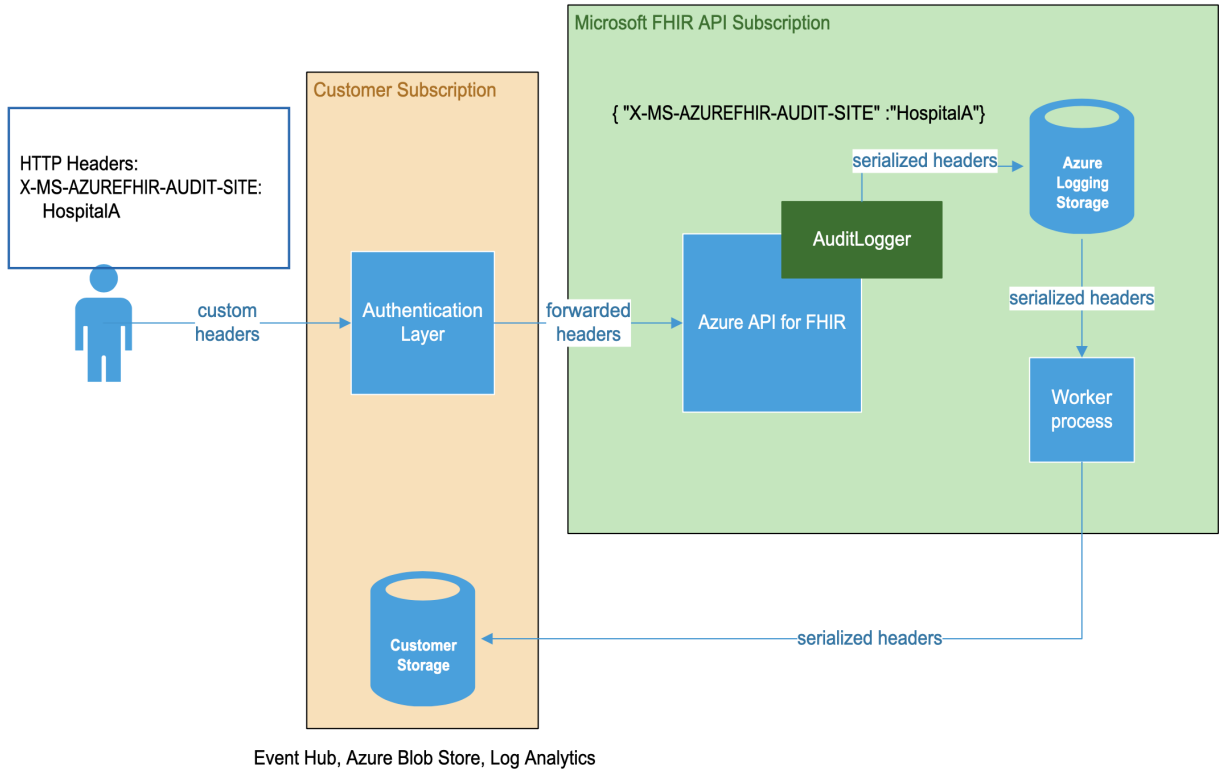
Add data to audit logs by using custom HTTP headers

2 minutes to read • Edit Online

In the Azure Fast Healthcare Interoperability Resources (FHIR) API, a user might want to include additional information in the logs, which comes from the calling system.

For example, when the user of the API is authenticated by an external system, that system forwards the call to the FHIR API. At the FHIR API layer, the information about the original user has been lost, because the call was forwarded. It might be necessary to log and retain this user information for auditing or management purposes. The calling system can provide user identity, caller location, or other necessary information in the HTTP headers, which will be carried along as the call is forwarded.

You can see this data flow in the following diagram:



You can use custom headers to capture several types of information. For example:

- Identity or authorization information
- Origin of the caller
- Originating organization
- Client system details (electronic health record, patient portal)

IMPORTANT

Be aware that the information sent in custom headers is stored in a Microsoft internal logging system for 30 days after being available in Azure Log Monitoring. We recommend encrypting any information before adding it to custom headers. You should not pass any PHI information through customer headers.

You must use the following naming convention for your HTTP headers: X-MS-AZUREFHIR-AUDIT-AUDIT-<name>.

These HTTP headers are included in a property bag that is added to the log. For example:

- X-MS-AZUREFHIR-AUDIT-USERID: 1234
- X-MS-AZUREFHIR-AUDIT-USERLOCATION: XXXX
- X-MS-AZUREFHIR-AUDIT-XYZ: 1234

This information is then serialized to JSON when it's added to the properties column in the log. For example:

```
{ "X-MS-AZUREFHIR-AUDIT-USERID" : "1234",
"X-MS-AZUREFHIR-AUDIT-USERLOCATION" : "XXXX",
"X-MS-AZUREFHIR-AUDIT-XYZ" : "1234" }
```

As with any HTTP header, the same header name can be repeated with different values. For example:

- X-MS-AZUREFHIR-AUDIT-USERLOCATION: HospitalA
- X-MS-AZUREFHIR-AUDIT-USERLOCATION: Emergency

When added to the log, the values are combined with a comma delimited list. For example:

{ "X-MS-AZUREFHIR-AUDIT-USERLOCATION" : "HospitalA, Emergency" }

You can add a maximum of 10 unique headers (repetitions of the same header with different values are only counted as one). The total maximum length of the value for any one header is 2048 characters.

If you're using the Firely C# client API library, the code looks something like this:

```
FhirClient client;
client = new FhirClient(serverUrl);
client.OnBeforeRequest += (object sender, BeforeRequestEventArgs e) =>
{
    // Add custom headers to be added to the logs
    e.RawRequest.Headers.Add("X-MS-AZUREFHIR-AUDIT-UserLocation", "HospitalA");
};
client.Get("Patient");
```

Next steps

In this article, you learned how to add data to audit logs by using custom headers in the Azure API for FHIR. Next, learn about other additional settings you can configure in the Azure API for FHIR.

Additional Settings

Frequently asked questions about the Azure API for FHIR

2 minutes to read • Edit Online

What is FHIR?

The Fast Healthcare Interoperability Resources (FHIR - Pronounced "fire") is an interoperability standard intended to enable the exchange of healthcare data between different health systems. This standard was developed by the HL7 organization and is being adopted by healthcare organizations around the world. The most current version of FHIR available is R4 (Release 4). The Azure API for FHIR supports R4 and also supports the previous version STU3 (Standard for Trial Use 3). For more information on FHIR, visit HL7.org.

Is the data behind the FHIR APIs stored in Azure?

Yes, the data is stored in managed databases in Azure. The Azure API for FHIR does not provide direct access to the underlying data store.

What identity provider do you support?

We currently support Microsoft Azure Active Directory as the identity provider.

What FHIR version do you support?

We support versions 4.0.0 and 3.0.1 on both the Azure API for FHIR (PaaS) and FHIR Server for Azure (open source).

For details, see Supported features. Read about what has changed between versions in the version history for HL7 FHIR.

What's the difference between the open-source Microsoft FHIR Server for Azure and the Azure API for FHIR?

The Azure API for FHIR is a hosted and managed version of the open-source Microsoft FHIR Server for Azure. In the managed service, Microsoft provides all maintenance and updates.

When you're running FHIR Server for Azure, you have direct access to the underlying services. But you're also responsible for maintaining and updating the server and all required compliance work if you're storing PHI data.

From a development standpoint, every feature is deployed to the open-source Microsoft FHIR Server for Azure first. Once it has been validated in open-source, it will be released to the PaaS Azure API for FHIR solution. The time between the release in open-source and PaaS depends on the complexity of the feature and other roadmap priorities.

What is SMART on FHIR?

SMART (Substitutable Medical Applications and Reusable Technology) on FHIR is a set of open specifications to integrate partner applications with FHIR Servers and other Health IT systems, such as Electronic Health Records and Health Information Exchanges. By creating a SMART on FHIR application, you can ensure that your application can be accessed and leveraged by a plethora of different systems. Authentication and Azure API for FHIR. To learn more about SMART, visit SMART Health IT.

Next steps

In this article, you've read some of the frequently asked questions about the Azure API for FHIR. Read about the supported features in FHIR Server for Azure:

Supported FHIR features

Features

3 minutes to read • Edit Online

Azure API for FHIR provides a fully managed deployment of the Microsoft FHIR Server for Azure. The server is an implementation of the FHIR standard. This document lists the main features of the FHIR Server.

FHIR version

Latest version supported: `4.0.0`

Previous versions also currently supported include: `3.0.1`

REST API

API	SUPPORTED - PAAS	SUPPORTED - OSS (SQL)	SUPPORTED - OSS (COSMOS DB)	COMMENT
read	Yes	Yes	Yes	
vread	Yes	Yes	Yes	
update	Yes	Yes	Yes	
update with optimistic locking	Yes	Yes	Yes	
update (conditional)	Yes	Yes	Yes	
patch	No	No	No	
delete	Yes	Yes	Yes	
delete (conditional)	No	No	No	
create	Yes	Yes	Yes	Support both POST/PUT
create (conditional)	Yes	Yes	Yes	
search	Partial	Partial	Partial	See below
capabilities	Yes	Yes	Yes	
batch	Yes	Yes	Yes	
transaction	No	Yes	No	
history	Yes	Yes	Yes	

API	SUPPORTED - PAAS	SUPPORTED - OSS (SQL)	SUPPORTED - OSS (COSMOS DB)	COMMENT
paging	Partial	Partial	Partial	`self` and `next` are supported
intermediaries	No	No	No	

Search

All search parameter types are supported. Chained parameters and reverse chaining are *not* supported.

SEARCH PARAMETER TYPE	SUPPORTED - PAAS	SUPPORTED - OSS (SQL)	SUPPORTED - OSS (COSMOS DB)	COMMENT
Number	Yes	Yes	Yes	
Date/DateTime	Yes	Yes	Yes	
String	Yes	Yes	Yes	
Token	Yes	Yes	Yes	
Reference	Yes	Yes	Yes	
Composite	Yes	Yes	Yes	
Quantity	Yes	Yes	Yes	
URI	Yes	Yes	Yes	
Special	No	No	No	

MODIFIERS	SUPPORTED - PAAS	SUPPORTED - OSS (SQL)	SUPPORTED - OSS (COSMOS DB)	COMMENT
`:missing`	Yes	Yes	Yes	
`:exact`	Yes	Yes	Yes	
`:contains`	Yes	Yes	Yes	
`:text`	Yes	Yes	Yes	
`:in` (token)	No	No	No	
`:below` (token)	No	No	No	
`:above` (token)	No	No	No	
`:not-in` (token)	No	No	No	

MODIFIERS	SUPPORTED - PAAS	SUPPORTED - OSS (SQL)	SUPPORTED - OSS (COSMOS DB)	COMMENT
`:[type]` (reference)	No	No	No	
`:below` (uri)	Yes	Yes	Yes	
`:not`	No	No	No	
`:above` (uri)	No	No	No	Issue #158

COMMON SEARCH PARAMETER	SUPPORTED - PAAS	SUPPORTED - OSS (SQL)	SUPPORTED - OSS (COSMOS DB)	COMMENT
`_id`	Yes	Yes	Yes	
`_lastUpdated`	Yes	Yes	Yes	
`_tag`	Yes	Yes	Yes	
`_profile`	Yes	Yes	Yes	
`_security`	Yes	Yes	Yes	
`_text`	No	No	No	
`_content`	No	No	No	
`_list`	No	No	No	
`_has`	No	No	No	
`_type`	Yes	Yes	Yes	
`_query`	No	No	No	

SEARCH OPERATIONS	SUPPORTED - PAAS	SUPPORTED - OSS (SQL)	SUPPORTED - OSS (COSMOS DB)	COMMENT
`_filter`	No	No	No	
`_sort`	No	No	No	
`_score`	No	No	No	
`_count`	Yes	Yes	Yes	
`_summary`	Partial	Partial	Partial	`_summary=count` is supported
`_include`	No	Yes	No	

SEARCH OPERATIONS	SUPPORTED - PAAS	SUPPORTED - OSS (SQL)	SUPPORTED - OSS (COSMOS DB)	COMMENT
`_revinclude`	No	No	No	
`_contained`	No	No	No	
`_elements`	No	No	No	

Persistence

The Microsoft FHIR Server has a pluggable persistence module (see `Microsoft.Health.Fhir.Core.Features.Persistence`).

Currently the FHIR Server open-source code includes an implementation for Azure Cosmos DB and SQL Database.

Cosmos DB is a globally distributed multi-model (SQL API, MongoDB API, etc.) database. It supports different consistency levels. The default deployment template configures the FHIR Server with `Strong` consistency, but the consistency policy can be modified (generally relaxed) on a request by request basis using the `x-ms-consistency-level` request header.

Role-based access control

The FHIR Server uses Azure Active Directory for access control. Specifically, Role-Based Access Control (RBAC) is enforced, if the `FhirServer:Security:Enabled` configuration parameter is set to `true`, and all requests (except `/metadata`) to the FHIR Server must have `Authorization` request header set to `Bearer <TOKEN>`. The token must contain one or more roles as defined in the `roles` claim. A request will be allowed if the token contains a role that allows the specified action on the specified resource.

Currently, the allowed actions for a given role are applied *globally* on the API.

Next steps

In this article, you've read about the supported FHIR features in Azure API for FHIR. Next deploy the Azure API for FHIR.

Deploy Azure API for FHIR

Partner ecosystem for Azure API for FHIR

2 minutes to read • Edit Online

We are excited that Azure API for FHIR has been released in generally availability to all Azure Customers. We are even more excited about the solutions that you will build with our service.

When creating an end-to-end solution built around Azure API for FHIR, you may require the help of a partner for their unique IP or for help stitching everything together. We are hard at work growing this ecosystem of diverse partners and I'd like to introduce you to a few of them.

PARTNER	CAPABILITIES	SUPPORTED COUNTRIES	CONTACT
Medal	De-identification, Legacy-FHIR conversion	USA	Contact
Rhapsody	Legacy-FHIR conversion	USA, Australia, New Zealand	Contact
iNTERFACEWARE	Legacy-FHIR conversion	USA, Canada	Contact
Darena Solutions	Application Development, System Integrator	USA	Contact
NewWave	Application Development, System Integrator	USA	Contact
Dapasoft	Application Development, System Integrator	USA, Canada	Contact
CitiusTech	Application Development, System Integrator	USA, UAE, UK	Contact
Firely	Application Development, System Integrator	USA, EU	Contact
Perspecta	Application Development, System Integrator	USA	Contact
Aridhia	Analytics	USA, EU	Contact