

Contents

Azure API for FHIR

Overview

[About Azure API for FHIR](#)

[About Azure IoT Connector for FHIR \(preview\)](#)

Quickstarts

[Deploy Azure API for FHIR](#)

[Using Portal](#)

[Using PowerShell](#)

[Using CLI](#)

[Using ARM template](#)

[Deploy Azure IoT Connector for FHIR \(preview\)](#)

[Using Portal](#)

[Using ARM template](#)

Tutorials

[Deploy JavaScript application](#)

[1. Initial setup and FHIR deployment](#)

[2. Register public client application](#)

[3. Test setup with Postman](#)

[4. Write web application](#)

[Access FHIR API with Postman](#)

[Use SMART on FHIR proxy](#)

[Ingest data from IoT devices](#)

[Receive device data through Azure IoT Hub](#)

[Interoperability and Patient Access](#)

[CMS Interoperability and Patient Access rule introduction](#)

[CARIN Implementation Guide for Blue Button](#)

[Da Vinci Drug Formulary](#)

[Da Vinci PDex](#)

How-to guides

Registering applications

- Register applications for Azure API for FHIR overview

- Resource application

- Confidential client application

- Public client application

- Service client application

Configure settings

- Configure another Azure API for FHIR settings

- Configure Azure RBAC

- Configure Local RBAC

- Configure database settings

- Configure customer-managed keys

- Configure CORS

- Configure Export

- Configure Private Link

Search

- Overview of FHIR search

- Defining custom search parameters

- How to run a reindex job

- Search examples for Azure API for FHIR

Operations

- Profile validation

- Export data

 - Export data

 - De-identified export

 - Move data to Synapse

- Convert data

 - \$convert-data and FHIR Converter Extension Templates

- Patient-everything in FHIR

- \$member-match operation

- Find identity object IDs

- Diagnostic logging and metrics

[Enable Diagnostics Logging in Azure API for FHIR](#)

[Display and configure Azure IoT Connector for FHIR \(preview\) metrics](#)

[Get a token for Azure API for FHIR - CLI](#)

[Troubleshoot failures in Azure IoT Connector for FHIR \(preview\)](#)

Concepts

[Azure AD and Azure API for FHIR Overview](#)

[Access token validation](#)

[Use Custom HTTP headers to add data to Audit Logs](#)

[Azure IoT Connector for FHIR \(preview\) workings](#)

[Azure IoT Connector for FHIR \(preview\) data flow](#)

[Azure IoT Connector for FHIR \(preview\) mapping templates](#)

Security

[Security controls by Azure Policy](#)

Resources

[FAQ](#)

[Supported features](#)

[GitHub Projects](#)

[Partner ecosystem](#)

Reference

[Azure CLI](#)

[Azure Policy built-ins](#)

What is Azure API for FHIR®?

3/11/2021 • 8 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 Information (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 you to 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

Azure IoT Connector for FHIR (preview)

Azure IoT Connector for Fast Healthcare Interoperability Resources (FHIR®)* is an optional feature of Azure API for FHIR that provides the capability to ingest data from Internet of Medical Things (IoMT) devices. Internet of Medical Things is a category of IoT devices that capture and exchange health & wellness data with other healthcare IT systems over network. Some examples of IoMT devices include fitness and clinical wearables, monitoring sensors, activity trackers, point of care kiosks, or even a smart pill. The Azure IoT Connector for FHIR feature enables you to quickly set up a service to ingest IoMT data into Azure API for FHIR in a scalable, secure, and compliant manner.

Azure IoT Connector for FHIR can accept any JSON-based messages sent out by an IoMT device. This data is first transformed into appropriate FHIR-based [Observation](#) resources and then persisted into Azure API for FHIR. The data transformation logic is defined through a pair of mapping templates that you configure based on your message schema and FHIR requirements. Device data can be pushed directly to Azure IoT Connector for FHIR or seamlessly used in concert with other Azure IoT solutions ([Azure IoT Hub](#) and [Azure IoT Central](#)). Azure IoT Connector for FHIR provides a secure data pipeline while allowing the Azure IoT solutions manage provisioning and maintenance of the physical devices.

Applications of Azure IoT Connector for FHIR (preview)

Use of IoMT devices is rapidly expanding in healthcare and Azure IoT Connector for FHIR is designed to bridge the gap of bringing multiple devices data with security and compliance into Azure API for FHIR. Bringing IoMT data into a FHIR server enables holistic data insights and innovative clinical workflows. Some common scenarios for Azure IoT Connector for FHIR are:

- **Remote Patient Monitoring/Telehealth:** Remote patient monitoring provides the ability to gather patient health data outside of traditional healthcare settings. Healthcare institutions can use Azure IoT Connector for FHIR to bring health data generated by remote devices into Azure API for FHIR. This data could be used to closely track patients health status, monitor patients adherence to the treatment plan and provide personalized care.
- **Research and Life Sciences:** Clinical trials are rapidly adopting IoMT devices like bio sensors, wearables, mobile apps to capture trial data. These trials can harness Azure IoT Connector for FHIR to transmit device data to Azure API for FHIR in a secure, efficient, and effective manner. Once in Azure API for FHIR, trial data could be used to run real-time analysis of trial data.
- **Advanced Analytics:** IoMT devices can provide large volume and variety of data at a high velocity, which makes them a great fit for serving training and testing data for your machine learning models. Azure IoT Connector for FHIR is inherently built to work with wide range of data frequency, flexible data schema, and cloud scaling with low latency. These attributes make Azure IoT Connector for FHIR an excellent choice for capturing device data for your advanced analytics needs.
- **Smart Hospitals/Clinics:** Today smart hospitals and clinics are setting up an infrastructure of interconnected digital assets. Azure IoT Connector for FHIR can be used to capture and integrate data from these connected components. Actionable insights from such data set enable better patient care and operational efficiency.

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](#)

To try out the Azure IoT Connector for FHIR feature, check out the quickstart to deploy Azure IoT Connector for FHIR using Azure portal.

[Deploy Azure IoT Connector for FHIR](#)

*In the Azure portal, Azure IoT Connector for FHIR is referred to as IoT Connector (preview). FHIR is a registered trademark of HL7 and is used with the permission of HL7.

What is Azure API for FHIR®?

3/11/2021 • 8 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 Information (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 you to 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

Azure IoT Connector for FHIR (preview)

Azure IoT Connector for Fast Healthcare Interoperability Resources (FHIR®)* is an optional feature of Azure API for FHIR that provides the capability to ingest data from Internet of Medical Things (IoMT) devices. Internet of Medical Things is a category of IoT devices that capture and exchange health & wellness data with other healthcare IT systems over network. Some examples of IoMT devices include fitness and clinical wearables, monitoring sensors, activity trackers, point of care kiosks, or even a smart pill. The Azure IoT Connector for FHIR feature enables you to quickly set up a service to ingest IoMT data into Azure API for FHIR in a scalable, secure, and compliant manner.

Azure IoT Connector for FHIR can accept any JSON-based messages sent out by an IoMT device. This data is first transformed into appropriate FHIR-based [Observation](#) resources and then persisted into Azure API for FHIR. The data transformation logic is defined through a pair of mapping templates that you configure based on your message schema and FHIR requirements. Device data can be pushed directly to Azure IoT Connector for FHIR or seamlessly used in concert with other Azure IoT solutions ([Azure IoT Hub](#) and [Azure IoT Central](#)). Azure IoT Connector for FHIR provides a secure data pipeline while allowing the Azure IoT solutions manage provisioning and maintenance of the physical devices.

Applications of Azure IoT Connector for FHIR (preview)

Use of IoMT devices is rapidly expanding in healthcare and Azure IoT Connector for FHIR is designed to bridge the gap of bringing multiple devices data with security and compliance into Azure API for FHIR. Bringing IoMT data into a FHIR server enables holistic data insights and innovative clinical workflows. Some common scenarios for Azure IoT Connector for FHIR are:

- **Remote Patient Monitoring/Telehealth:** Remote patient monitoring provides the ability to gather patient health data outside of traditional healthcare settings. Healthcare institutions can use Azure IoT Connector for FHIR to bring health data generated by remote devices into Azure API for FHIR. This data could be used to closely track patients health status, monitor patients adherence to the treatment plan and provide personalized care.
- **Research and Life Sciences:** Clinical trials are rapidly adopting IoMT devices like bio sensors, wearables, mobile apps to capture trial data. These trials can harness Azure IoT Connector for FHIR to transmit device data to Azure API for FHIR in a secure, efficient, and effective manner. Once in Azure API for FHIR, trial data could be used to run real-time analysis of trial data.
- **Advanced Analytics:** IoMT devices can provide large volume and variety of data at a high velocity, which makes them a great fit for serving training and testing data for your machine learning models. Azure IoT Connector for FHIR is inherently built to work with wide range of data frequency, flexible data schema, and cloud scaling with low latency. These attributes make Azure IoT Connector for FHIR an excellent choice for capturing device data for your advanced analytics needs.
- **Smart Hospitals/Clinics:** Today smart hospitals and clinics are setting up an infrastructure of interconnected digital assets. Azure IoT Connector for FHIR can be used to capture and integrate data from these connected components. Actionable insights from such data set enable better patient care and operational efficiency.

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](#)

To try out the Azure IoT Connector for FHIR feature, check out the quickstart to deploy Azure IoT Connector for FHIR using Azure portal.

[Deploy Azure IoT Connector for FHIR](#)

*In the Azure portal, Azure IoT Connector for FHIR is referred to as IoT Connector (preview). FHIR is a registered trademark of HL7 and is used with the permission of HL7.

Quickstart: Deploy Azure API for FHIR using Azure portal

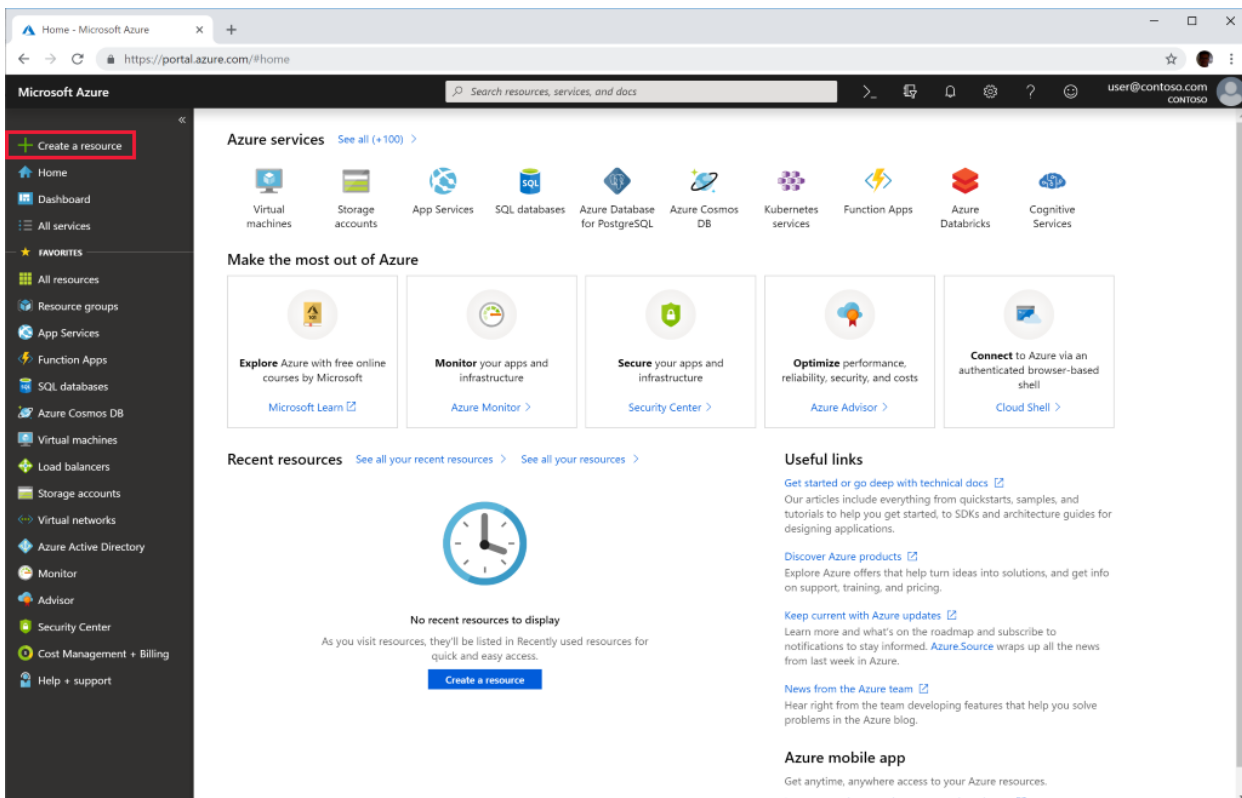
3/11/2021 • 2 minutes to read • [Edit Online](#)

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:

New

Azure API for FHIR

Get started


Recently created


Recently created


AI + Machine Learning

Analytics

Blockchain

 Windows Server 2016 Datacenter
[Quickstart tutorial](#)

 Ubuntu Server 18.04 LTS
[Learn more](#)


 Web App
[Quickstart tutorial](#)

Create Azure API for FHIR account

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

Home > New > Azure API for FHIR

Azure API for FHIR
Microsoft



Azure API for FHIR [Save for later](#)
Microsoft
[Create](#)

The Azure API for FHIR® is a managed, standards-based, and healthcare data platform. It enables organizations to bring their clinical health data into the cloud based on the interoperable data standard FHIR®.

FHIR helps unlock the value of data and respond to changing business dynamics more easily.

Organizations are able to bring together clinical data from multiple systems of records, normalize the data using common models and specifications, and use that data in AI workloads to derive insight and power new systems of engagement, including clinician and patient dashboards, diagnostic assistants, population health insights, and connected healthcare scenarios, such as Remote Patient Monitoring.

The Azure API for FHIR is capable of powering Internet of Medical Things (IoMT) scenarios, population health research projects, AI-powered diagnostic solutions and much more.

Security and privacy features are embedded into the service. Customers own and control patient data, knowing how it is stored and accessed.

Useful Links
[HL7 FHIR Specification](#)
[FHIR Server for Azure](#)
[Documentation](#)

Enter account details

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

Create Azure API for FHIR

[Basics *](#) [Additional settings *](#) [Tags](#) [Review + create](#)

The Azure API for FHIR® is a managed, standards-based, and compliant healthcare data platform. It enables organizations to bring their clinical health data into the cloud based on the interoperable data standard FHIR®.

Project details

Subscription *	Health Demo
Resource group *	demo

[Create new](#)

Instance details

Account name *	mydemofhir
Location *	North Europe
FHIR Version *	R4

.azurehealthcareapis.com

[Review + create](#)

[Next: Additional settings >](#)

Confirm creation and await FHIR API deployment.

Additional settings (optional)

You can also click **Next: Additional settings** to view the authentication settings. The default configuration for the Azure API for FHIR is to [use Azure RBAC for assigning data plane roles](#). When configured in this mode, the "Authority" for the FHIR service will be set to the Azure Active Directory tenant of the subscription:

Create Azure API for FHIR

Basics * Additional settings * Tags Review + create

Customize additional configuration parameters including authentication and storage.

Authentication

Authority *

Audience * ✓

Allowed object IDs ⓘ

Use Azure Access Control (IAM) to grant access your FHIR service when using the subscription tenant for data plane RBAC. [Learn more.](#)

SMART on FHIR proxy ⓘ ☐

Database Settings

Provisioned throughput (RU/s) * ⓘ

[Review + create](#)

[Previous](#)

[Next: Tags >](#)

Notice that the box for entering allowed object IDs is grayed out, since we use Azure RBAC for configuring role assignments in this case.

If you wish to configure the FHIR service to use an external or secondary Azure Active Directory tenant, you can change the Authority and enter object IDs for user and groups that should be allowed access to the server. For more information, see the [local RBAC configuration](#) guide.

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. If you are ready to start using the Azure API for FHIR, read more on how to register applications.

[Additional settings in Azure API for FHIR](#)

[Register Applications Overview](#)

Quickstart: Deploy Azure API for FHIR using PowerShell

5/28/2021 • 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 Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

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
```

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
```

NOTE

Depending on the version of the `Az` PowerShell module you have installed, the provisioned FHIR server may be configured to use [local RBAC](#) and have the currently signed in PowerShell user set in the list of allowed identity object IDs for the deployed FHIR service. Going forward, we recommend that you [use Azure RBAC](#) for assigning data plane roles and you may need to delete this users object ID after deployment to enable Azure RBAC mode.

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. If you are ready to start using the Azure API for FHIR, read more on how to register applications.

[Additional settings in Azure API for FHIR](#)

[Register Applications Overview](#)

Quickstart: Deploy Azure API for FHIR using Azure CLI

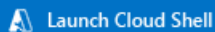
3/11/2021 • 2 minutes to read • [Edit Online](#)

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.

Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#).



- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
 - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
 - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
 - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).

Add HealthcareAPIs extension

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

Get a list of commands for HealthcareAPIs:

```
az healthcareapis --help
```

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
```

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. If you are ready to start using the Azure API for FHIR, read more on how to register applications.

[Additional settings in Azure API for FHIR](#)

[Register Applications Overview](#)

Quickstart: Use an ARM template to deploy Azure API for FHIR

5/28/2021 • 6 minutes to read • [Edit Online](#)

In this quickstart, you'll learn how to use an Azure Resource Manager template (ARM template) to deploy Azure API for Fast Healthcare Interoperability Resources (FHIR®). You can deploy Azure API for FHIR through the Azure portal, PowerShell, or CLI.

An [ARM template](#) is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax. In declarative syntax, you describe your intended deployment without writing the sequence of programming commands to create the deployment.

If your environment meets the prerequisites and you're familiar with using ARM templates, select the **Deploy to Azure** button. The template will open in the Azure portal once you sign in.



Deploy to Azure

Prerequisites

- [Portal](#)
- [PowerShell](#)
- [CLI](#)

An Azure account with an active subscription. [Create one for free.](#)

Review the template

The template used in this quickstart is from [Azure Quickstart Templates](#).

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "serviceName": {
      "type": "string",
      "metadata": {
        "description": "The name of the service."
      }
    },
  },
  "location": {
    "type": "string",
    "allowedValues": [
      "australiaeast",
      "eastus",
      "eastus2",
      "japaneast",
      "northcentralus",
      "northeurope",
      "southcentralus",
      "southeastasia",
      "uksouth",
      "ukwest",
      "westcentralus",
      "westeurope",
      "westus2"
    ],
    "metadata": {
      "description": "Location of Azure API for FHIR"
    }
  },
  "resources": [
    {
      "type": "Microsoft.HealthcareApis/services",
      "apiVersion": "2020-03-15",
      "name": "[parameters('serviceName')]",
      "location": "[parameters('location')]",
      "kind": "fhir-R4",
      "properties": {
        "authenticationConfiguration": {
          "audience": "[concat('https://', parameters('serviceName'), '.azurehealthcareapis.com')]",
          "authority": "[uri(environment().authentication.loginEndpoint, subscription().tenantId)]"
        }
      }
    }
  ]
}
```

The template defines one Azure resource:

- **Microsoft.HealthcareApis/services**

Deploy the template

- [Portal](#)
- [PowerShell](#)
- [CLI](#)

Select the following link to deploy the Azure API for FHIR using the ARM template in the Azure portal:



On the **Deploy Azure API for FHIR** page:

1. If you want, change the **Subscription** from the default to a different subscription.
2. For **Resource group**, select **Create new**, enter a name for the new resource group, and select **OK**.
3. If you created a new resource group, select a **Region** for the resource group.
4. Enter a new **Service Name** and choose the **Location** of the Azure API for FHIR. The location can be the same as or different from the region of the resource group.

The screenshot shows the 'Deploy Azure API for FHIR' page in the Azure portal. The page has a blue header with the Microsoft Azure logo and a search bar. The main content area is titled 'Deploy Azure API for FHIR' and includes a 'Basics' tab. Under the 'Basics' tab, there are sections for 'Template' (101-azure-api-for-fhir, 1 resource), 'Deployment scope' (Subscription, Resource group), and 'Parameters' (Region, Service Name, Location). The 'Resource group' field is highlighted with a red box, and the 'Region' field is also highlighted. The 'Service Name' and 'Location' fields are highlighted together. At the bottom, there are three buttons: 'Review + create' (highlighted), '< Previous', and 'Next : Review + create >'.

5. Select **Review + create**.
6. Read the terms and conditions, and then select **Create**.

NOTE

The deployment takes a few minutes to complete. Note the names for the Azure API for FHIR service and the resource group, which you use to review the deployed resources later.

Review deployed resources

- [Portal](#)
- [PowerShell](#)
- [CLI](#)

Follow these steps to see an overview of your new Azure API for FHIR service:

1. In the [Azure portal](#), search for and select **Azure API for FHIR**.
2. In the FHIR list, select your new service. The **Overview** page for the new Azure API for FHIR service appears.
3. To validate that the new FHIR API account is provisioned, select the link next to **FHIR metadata endpoint** to fetch the FHIR API capability statement. The link has a format of `https://<service-name>.azurehealthcareapis.com/metadata`. If the account is provisioned, a large JSON file is displayed.

Clean up resources

When it's no longer needed, delete the resource group, which deletes the resources in the resource group.

- [Portal](#)
- [PowerShell](#)
- [CLI](#)

1. In the [Azure portal](#), search for and select **Resource groups**.
2. In the resource group list, choose the name of your resource group.
3. In the **Overview** page of your resource group, select **Delete resource group**.
4. In the confirmation dialog box, type the name of your resource group, and then select **Delete**.

For a step-by-step tutorial that guides you through the process of creating an ARM template, see the [tutorial to create and deploy your first ARM template](#)

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. If you are ready to start using the Azure API for FHIR, read more on how to register applications.

[Additional settings in Azure API for FHIR](#)

[Register Applications Overview](#)

Quickstart: Deploy Azure IoT Connector for FHIR (preview) using Azure portal

4/5/2021 • 6 minutes to read • [Edit Online](#)

Azure IoT Connector for Fast Healthcare Interoperability Resources (FHIR®)* is an optional feature of Azure API for FHIR that provides the capability to ingest data from Internet of Medical Things (IoMT) devices. During the preview phase, Azure IoT Connector for FHIR feature is being available for free. In this quickstart, you'll learn how to:

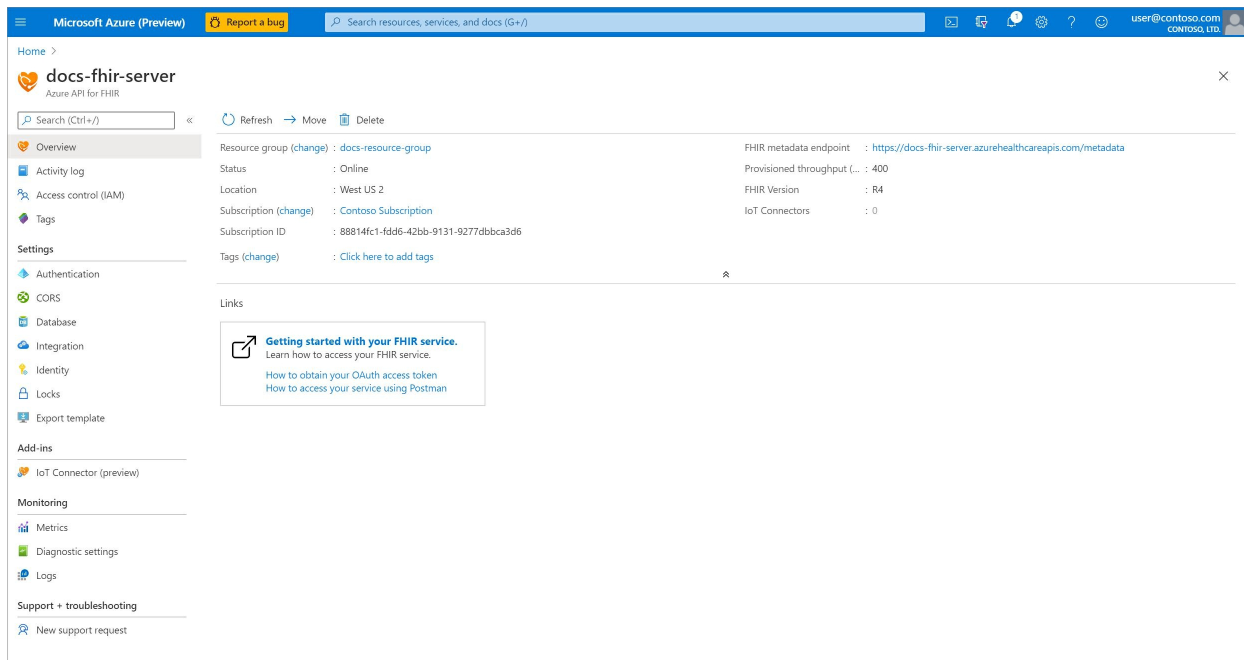
- Deploy and configure Azure IoT Connector for FHIR using the Azure portal
- Use a simulated device to send data to Azure IoT Connector for FHIR
- View resources created by Azure IoT Connector for FHIR on Azure API for FHIR

Prerequisites

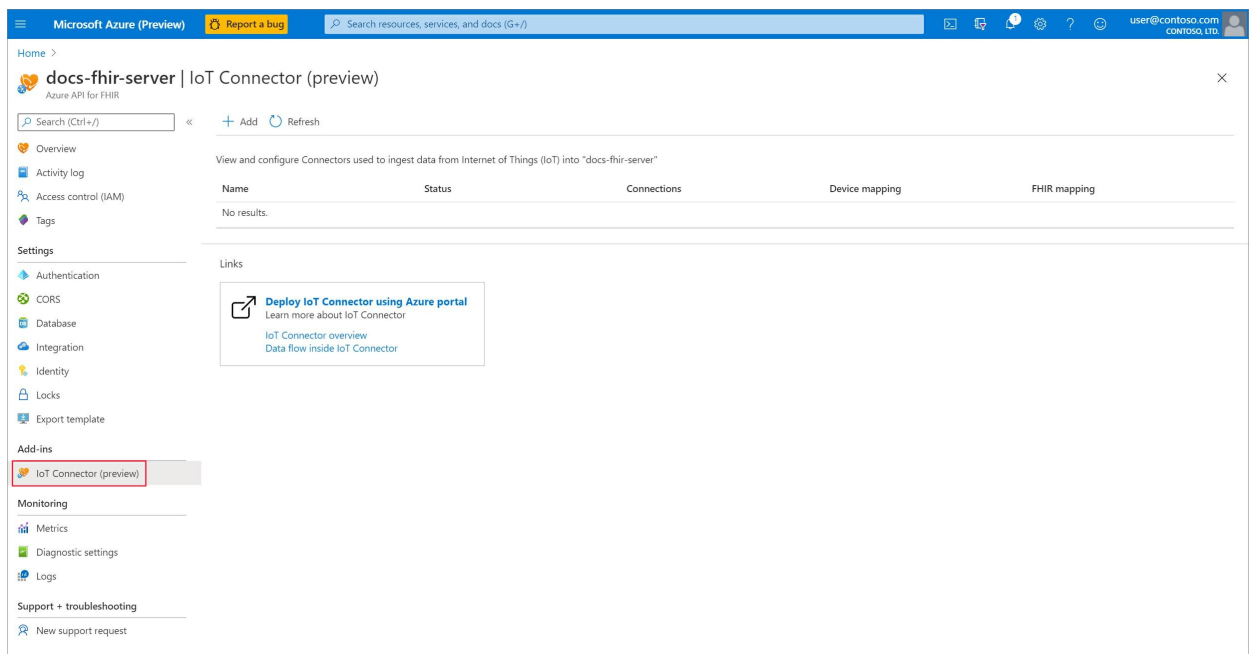
- An active Azure subscription - [Create one for free](#)
- Azure API for FHIR resource - [Deploy Azure API for FHIR using Azure portal](#)

Go to Azure API for FHIR resource

Open the [Azure portal](#) and go to the **Azure API for FHIR** resource for which you'd like to create the Azure IoT Connector for FHIR feature.

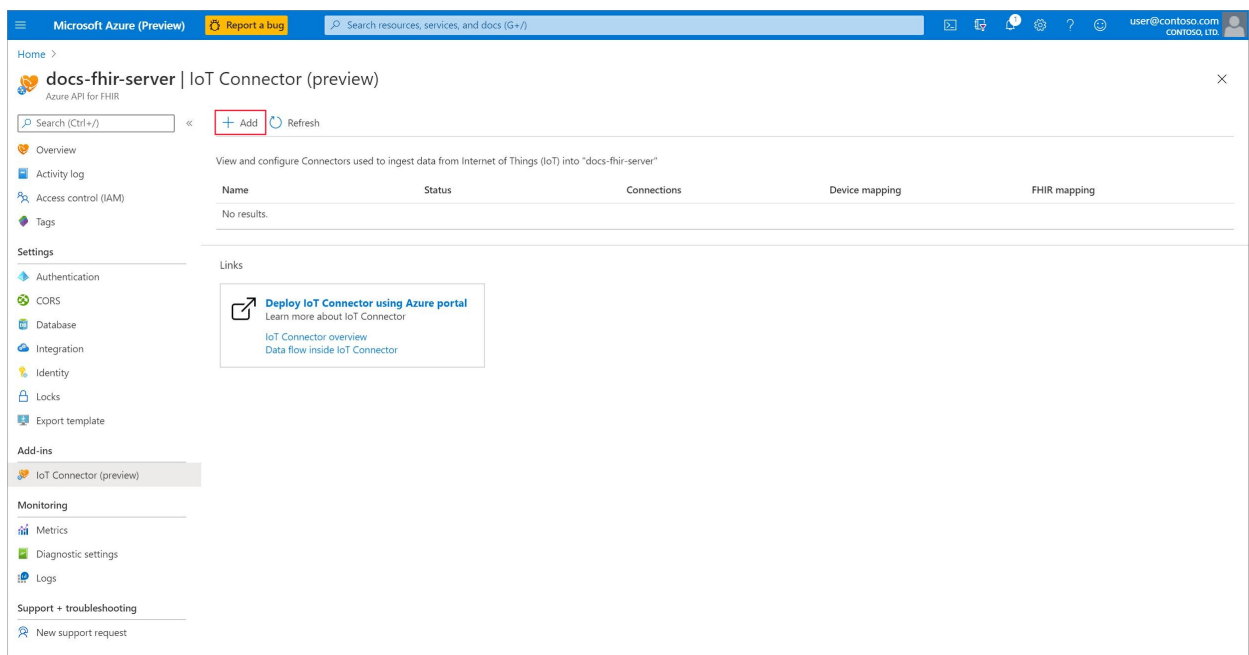


On the left-hand navigation menu, click on **IoT Connector (preview)** under the **Add-ins** section to open the **IoT Connectors** page.



Create new Azure IoT Connector for FHIR (preview)

Click on the **Add** button to open the **Create IoT Connector** page.



Enter settings for the new Azure IoT Connector for FHIR. Click on **Create** button and await Azure IoT Connector for FHIR deployment.

NOTE

Must select **Create** as the value for the **Resolution type** drop down for this installation.

Microsoft Azure (Preview)
Report a bug
Search resources, services, and docs (G+/I)
user@contoso.com
CONTOSO, LTD.

Home > docs-fhir-server | IoT Connector (preview) >

Create IoT Connector

IoT Connector (preview)

Create a new Connector to ingest data from Internet of Things (IoT) into "docs-fhir-server"

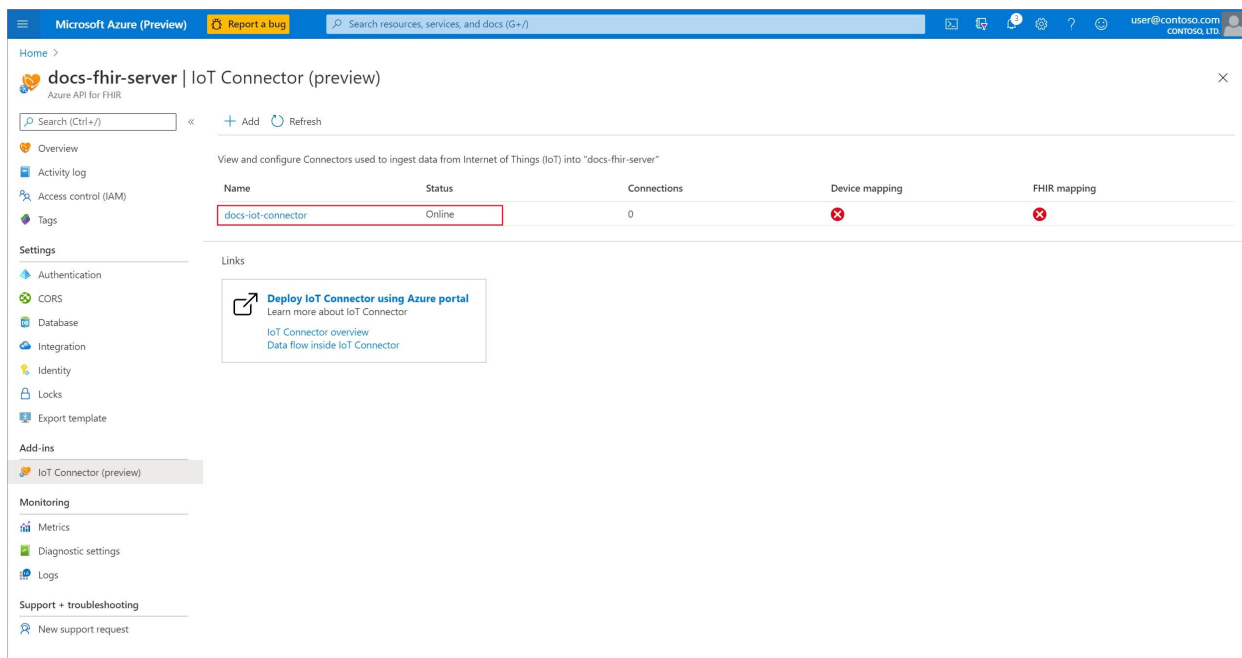
Connector name *
docs-iot-connector

Resolution type *
Create

Create

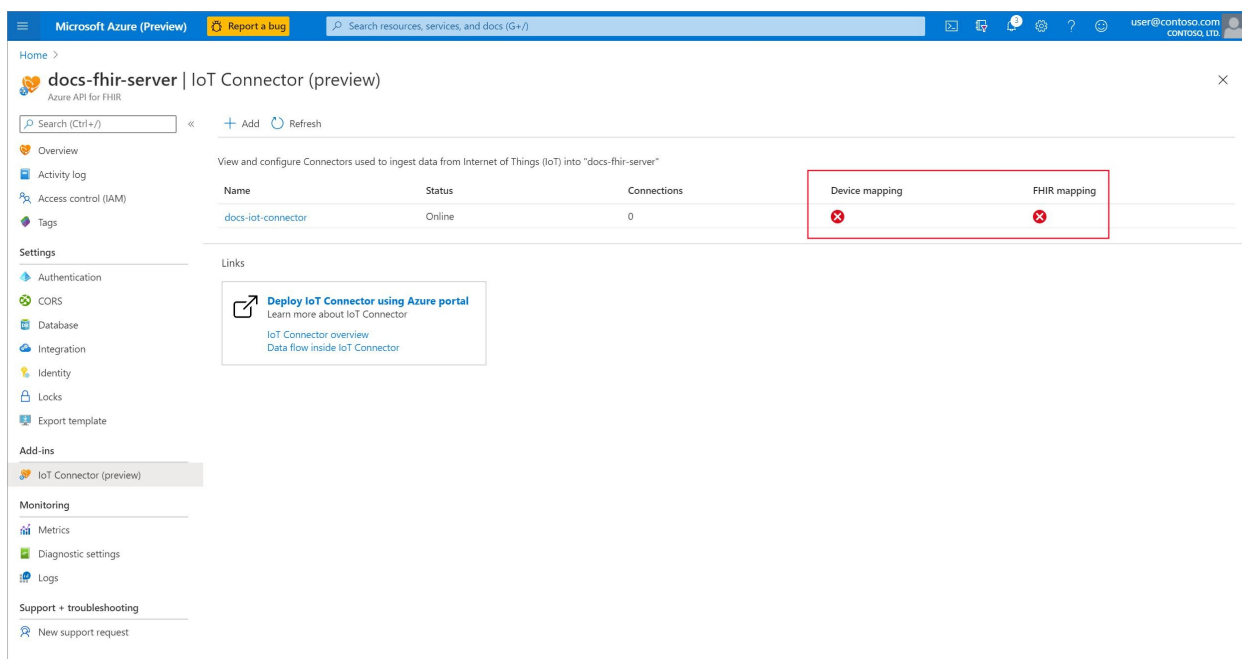
SETTING	VALUE	DESCRIPTION
Connector name	A unique name	Enter a name to identify your Azure IoT Connector for FHIR This name should be unique within an Azure API for FHIR resource. The name can only contain lowercase letters, numbers, and the hyphen (-) character. It must start and end with a letter or a number, and must be between 3-24 characters in length.
Resolution type	Lookup or Create	Select Lookup if you have an out-of-band process to create Device and Patient FHIR resources in your Azure API for FHIR. Azure IoT Connector for FHIR will use reference to these resources when creating an Observation FHIR resource to represent the device data. Select Create when you want Azure IoT Connector for FHIR to create bare-bones Device and Patient resources in your Azure API for FHIR using respective identifier values present in the device data.

Once installation is complete, the newly created Azure IoT Connector for FHIR will show up on the **IoT Connectors** page.

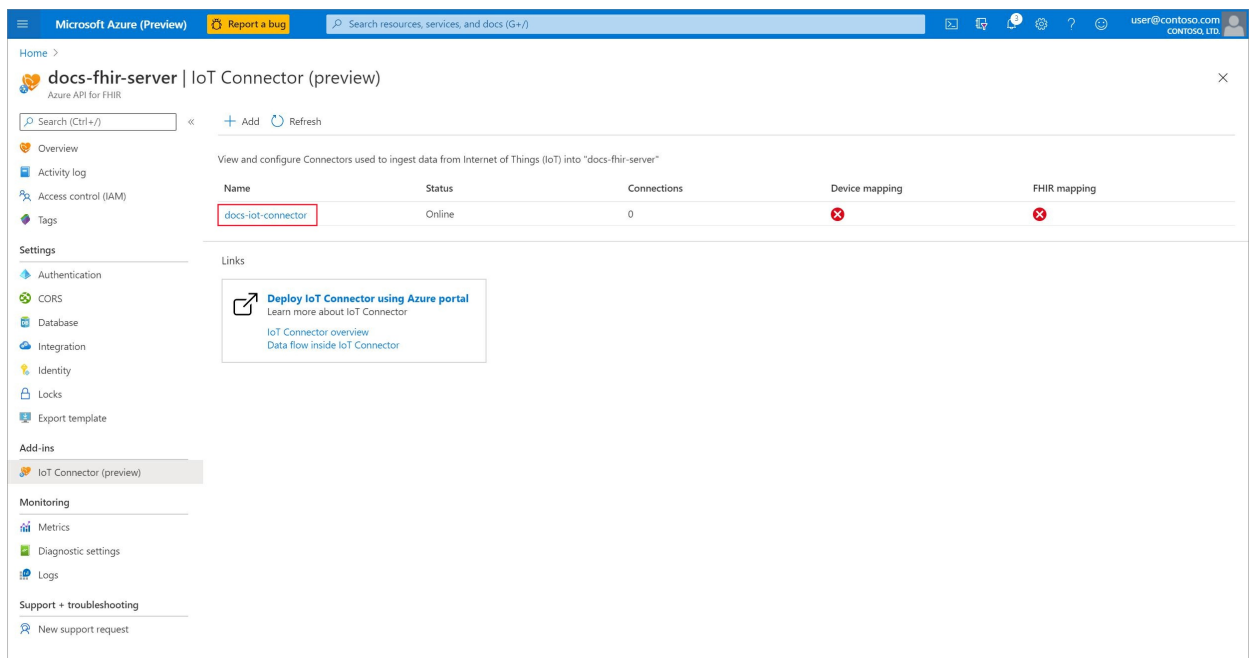


Configure Azure IoT Connector for FHIR (preview)

Azure IoT Connector for FHIR needs two mapping templates to transform device messages into FHIR-based Observation resource(s): **device mapping** and **FHIR mapping**. Your Azure IoT Connector for FHIR isn't fully operational until these mappings are uploaded.

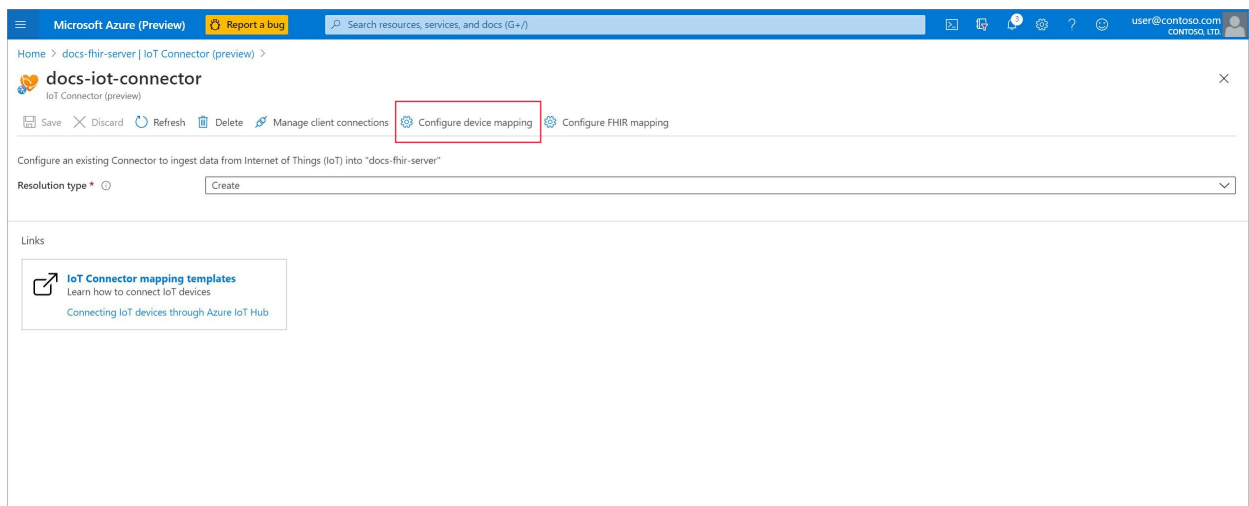


To upload mapping templates, click on the newly deployed Azure IoT Connector for FHIR to go to the **IoT Connector** page.



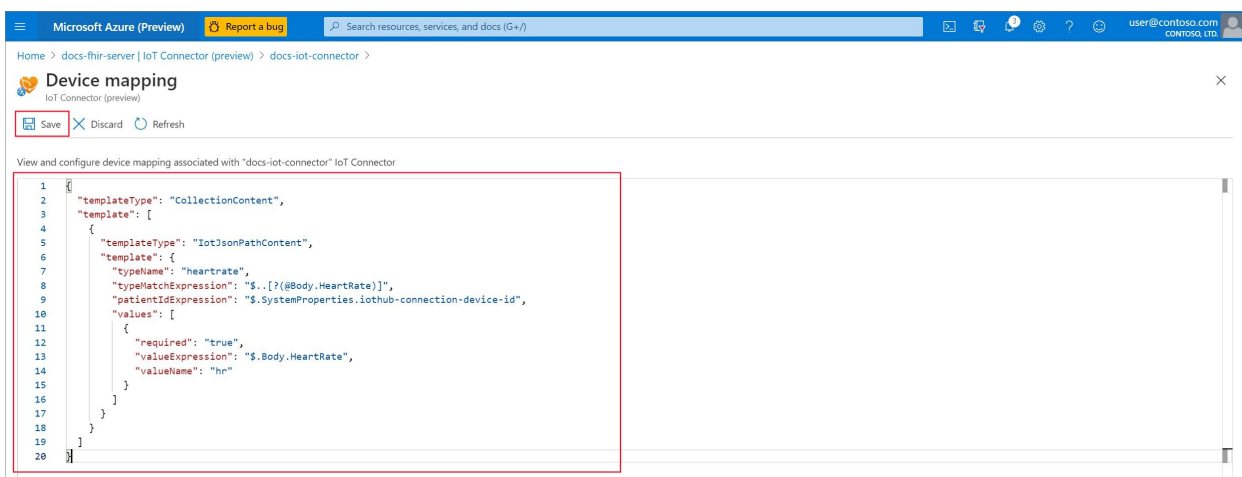
Device mapping

Device mapping template transforms device data into a normalized schema. On the **IoT Connector** page, click on **Configure device mapping** button to go to the **Device mapping** page.



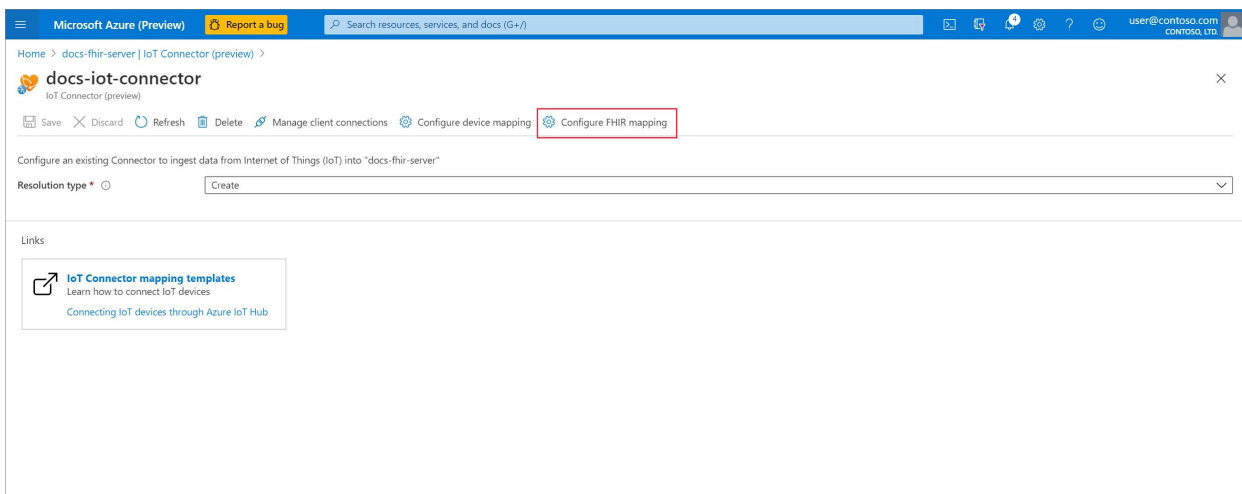
On the **Device mapping** page, add the following script to the JSON editor and click **Save**.

```
{
  "templateType": "CollectionContent",
  "template": [
    {
      "templateType": "IotJsonPathContent",
      "template": {
        "typeName": "heartrate",
        "typeMatchExpression": "$..[?(@Body.telemetry.HeartRate)]",
        "patientIdExpression": "$.Properties.iotcentral-device-id",
        "values": [
          {
            "required": "true",
            "valueExpression": "$.Body.telemetry.HeartRate",
            "valueName": "hr"
          }
        ]
      }
    ]
  ]
}
```



FHIR mapping

FHIR mapping template transforms a normalized message to a FHIR-based Observation resource. On the IoT Connector page, click on **Configure FHIR mapping** button to go to the FHIR mapping page.



On the **FHIR mapping** page, add the following script to the JSON editor and click **Save**.


```

{
  "templateType": "CollectionFhir",
  "template": [
    {
      "templateType": "CodeValueFhir",
      "template": {
        "codes": [
          {
            "code": "8867-4",
            "system": "http://loinc.org",
            "display": "Heart rate"
          }
        ],
        "periodInterval": 0,
        "typeName": "heartrate",
        "value": {
          "unit": "count/min",
          "valueName": "hr",
          "valueType": "Quantity"
        }
      }
    }
  ]
}

```

Microsoft Azure (Preview) | IoT Connector (preview) > docs-fhir-server > docs-fhir-server > docs-fhir-server

FHIR mapping

IoT Connector (preview)

Save Discard Refresh

View and configure FHIR mapping associated with "docs-fhir-server" IoT Connector

```

1 {
2   "templateType": "CollectionFhir",
3   "template": [
4     {
5       "templateType": "CodeValueFhir",
6       "template": {
7         "codes": [
8           {
9             "code": "8867-4",
10            "system": "http://loinc.org",
11            "display": "Heart rate"
12          }
13        ],
14        "periodInterval": 0,
15        "typeName": "heartrate",
16        "value": {
17          "unit": "count/min",
18          "valueName": "hr",
19          "valueType": "Quantity"
20        }
21      }
22    }
23  ]
24 }

```

Generate a connection string

IoMT device needs a connection string to connect and send messages to Azure IoT Connector for FHIR. On the **IoT Connector** page for the newly deployed Azure IoT Connector for FHIR, select **Manage client connections** button.

Microsoft Azure (Preview) | IoT Connector (preview) > docs-fhir-server > docs-fhir-server > docs-fhir-server

docs-iot-connector

IoT Connector (preview)

Save Discard Refresh Delete Manage client connections Configure device mapping Configure FHIR mapping

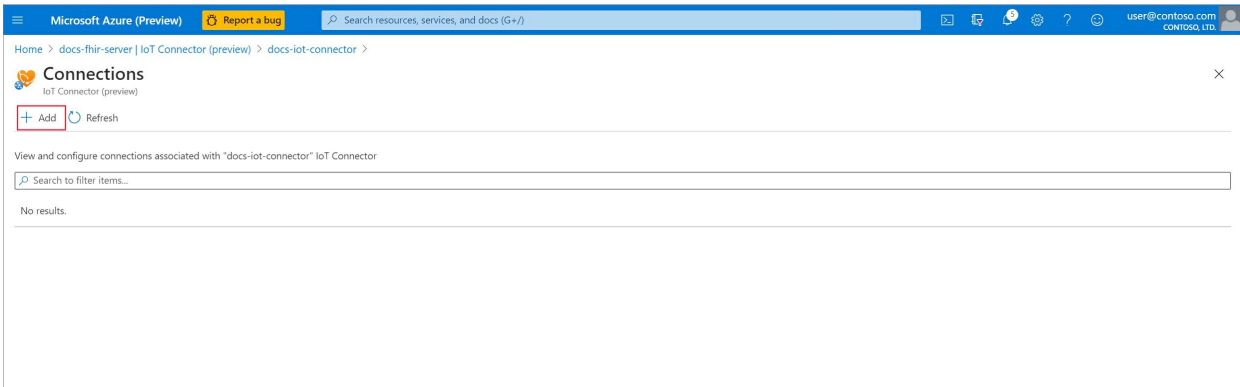
Configure an existing Connector to ingest data from Internet of Things (IoT) into "docs-fhir-server"

Resolution type* ☐ Create

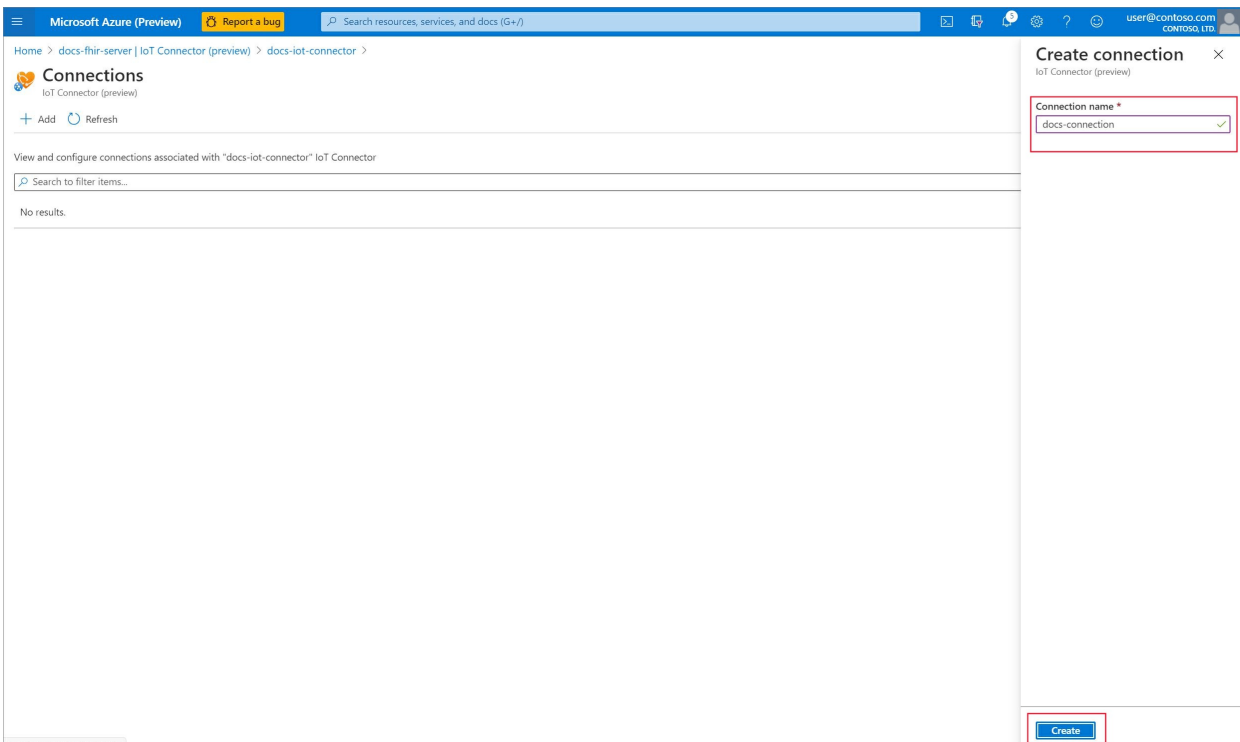
Links

[IoT Connector mapping templates](#)
Learn how to connect IoT devices
Connecting IoT devices through Azure IoT Hub

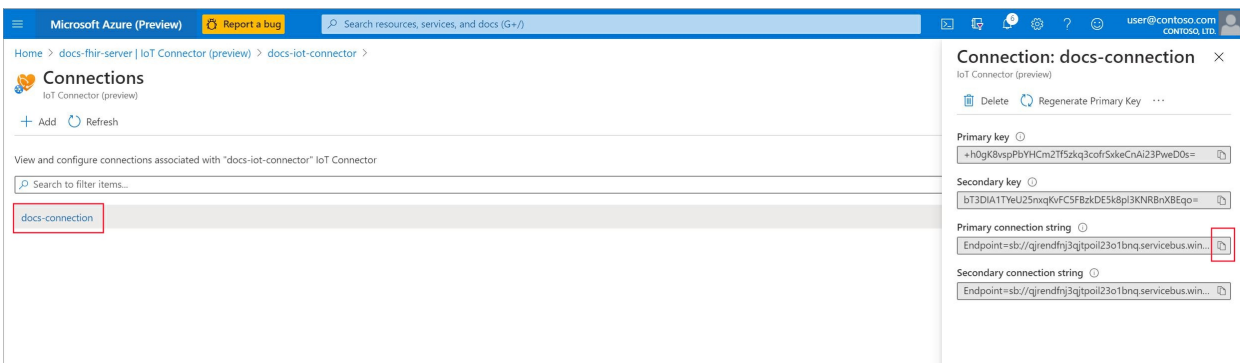
Once on **Connections** page, click on **Add** button to create a new connection.



Provide a friendly name for this connection on the overlay window and select the **Create** button.



Select the newly created connection from the **Connections** page and copy the value of **Primary connection string** field from the overlay window on the right.



Preserve this connection string to be used at a later step.

Connect your devices to IoT

Azure offers an extensive suite of IoT products to connect and manage your IoT devices. You can build your own solution based on PaaS using Azure IoT Hub, or start with a manage IoT apps platform with Azure IoT Central. For this tutorial, we'll leverage Azure IoT Central, which has industry-focused solution templates to help you get

started.

Deploy the [Continuous patient monitoring application template](#). This template includes two simulated devices producing real-time data to help you get started: **Smart Vitals Patch** and **Smart Knee Brace**.

NOTE

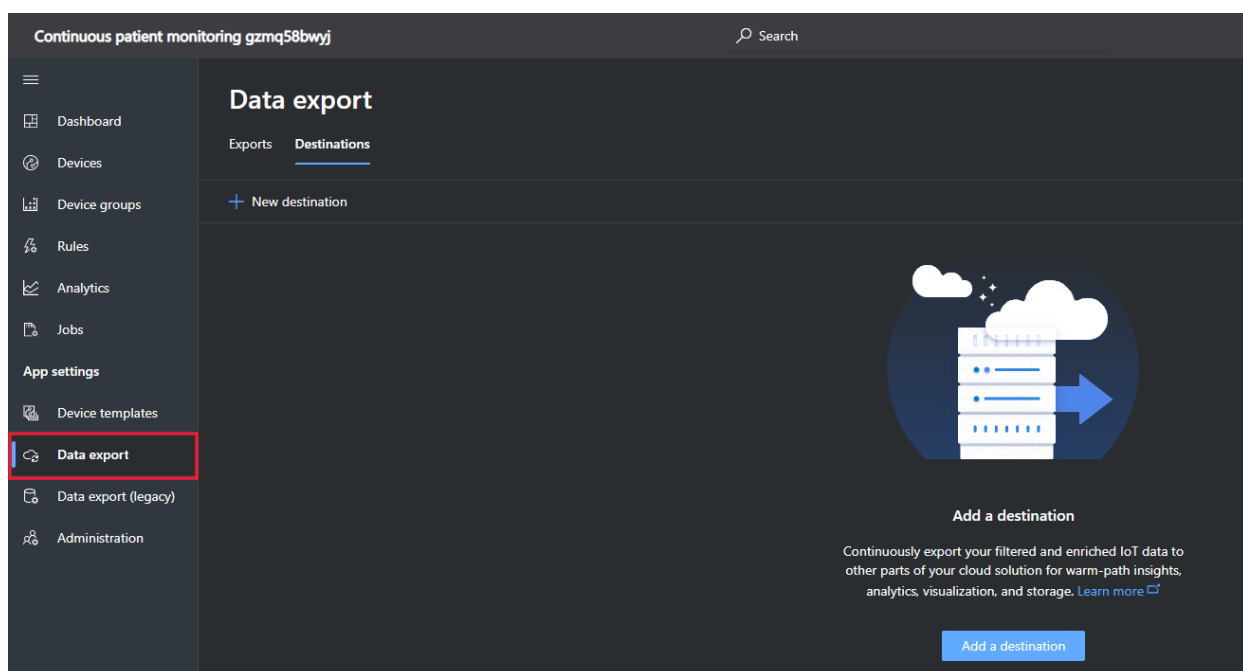
Whenever your real devices are ready, you can use same IoT Central application to [onboard your devices](#) and replace device simulators. Your device data will automatically start flowing to FHIR as well.

Connect your IoT data with the Azure IoT Connector for FHIR (preview)

Once you've deployed your IoT Central application, your two out-of-the-box simulated devices will start generating telemetry. For this tutorial, we'll ingest the telemetry from *Smart Vitals Patch* simulator into FHIR via the Azure IoT Connector for FHIR. To export your IoT data to the Azure IoT Connector for FHIR, we'll want to [set up a continuous data export within IoT Central](#). We'll first need to create a connection to the destination, and then we'll create a data export job to continuously run:

NOTE

You will want to select **Data export** vs. **Data export (legacy)** within the IoT Central App settings for this section.



Create a new destination:

- Go to the **Destinations** tab and create a new destination.
- Start by giving your destination a unique name.
- Pick *Azure Event Hubs* as the destination type.
- Provide Azure IoT Connector for FHIR's connection string obtained in a previous step for the **Connection string** field.

Create a new data export:

- Once you've created your destination, go over to the **Exports** tab and create a new data export.
- Start by giving it the data export a unique name.

- Under **Data** select *Telemetry* as the *Type of data to export*.
- Under **Destination** select the destination name you created in the previous name.

View device data in Azure API for FHIR

You can view the FHIR-based Observation resource(s) created by Azure IoT Connector for FHIR on Azure API for FHIR using Postman. Set up your [Postman to access Azure API for FHIR](#) and make a `GET` request to `https://your-fhir-server-url/Observation?code=http://loinc.org|8867-4` to view Observation FHIR resources with heart rate value.

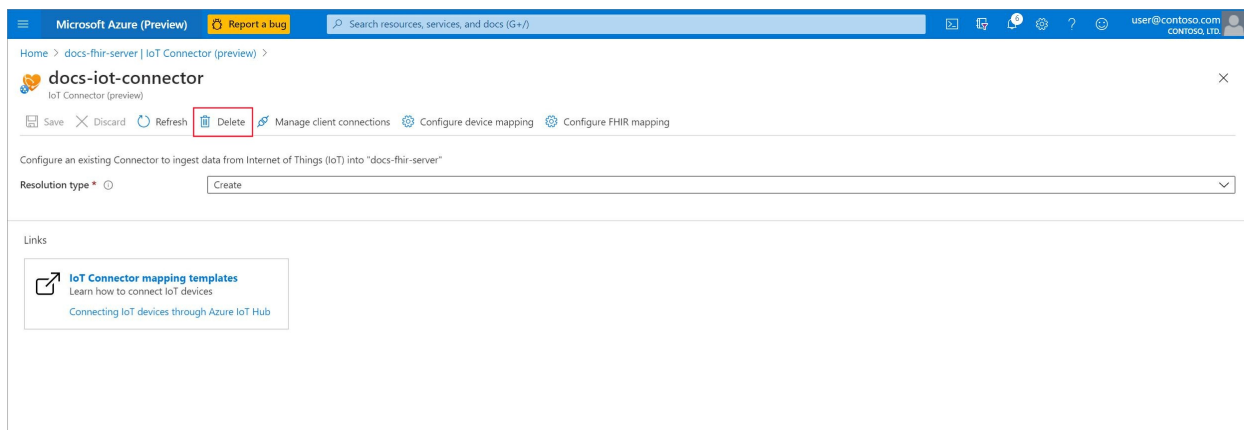
TIP

Ensure that your user has appropriate access to Azure API for FHIR data plane. Use [Azure role-based access control \(Azure RBAC\)](#) to assign required data plane roles.

Clean up resources

When no longer needed, you can delete an instance of Azure IoT Connector for FHIR by removing the associated resource group, or the associated Azure API for FHIR service, or the Azure IoT Connector for FHIR instance itself.

To directly remove an Azure IoT Connector for FHIR instance, select the instance from **IoT Connectors** page to go to **IoT Connector** page and click on **Delete** button. Select **Yes** when asked for confirmation.



Next steps

In this quickstart guide, you've deployed Azure IoT Connector for FHIR feature in your Azure API for FHIR resource. Select from below next steps to learn more about Azure IoT Connector for FHIR:

Understand different stages of data flow within Azure IoT Connector for FHIR.

[Azure IoT Connector for FHIR data flow](#)

Learn how to configure IoT Connector using device and FHIR mapping templates.

[Azure IoT Connector for FHIR mapping templates](#)

*In the Azure portal, Azure IoT Connector for FHIR is referred to as IoT Connector (preview). FHIR is a registered trademark of HL7 and is used with the permission of HL7.

Quickstart: Use an Azure Resource Manager (ARM) template to deploy Azure IoT Connector for FHIR (preview)

5/28/2021 • 11 minutes to read • [Edit Online](#)

In this quickstart, you'll learn how to use an Azure Resource Manager template (ARM template) to deploy Azure IoT Connector for Fast Healthcare Interoperability Resources (FHIR®)*, a feature of Azure API for FHIR. To deploy a working instance of Azure IoT Connector for FHIR, this template also deploys a parent Azure API for FHIR service and an Azure IoT Central application that exports telemetry from a device simulator to Azure IoT Connector for FHIR. You can execute ARM template to deploy Azure IoT Connector for FHIR through the Azure portal, PowerShell, or CLI.

An [ARM template](#) is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax. In declarative syntax, you describe your intended deployment without writing the sequence of programming commands to create the deployment.

If your environment meets the prerequisites and you're familiar with using ARM templates, select the **Deploy to Azure** button. The template will open in the Azure portal once you sign in.



Prerequisites

- [Portal](#)
- [PowerShell](#)
- [CLI](#)

An Azure account with an active subscription. [Create one for free.](#)

Review the template

The [template](#) defines following Azure resources:

- Microsoft.HealthcareApis/services
- Microsoft.HealthcareApis/services/iomtconnectors
- Microsoft.IoTCentral/IoTApps

Deploy the template

- [Portal](#)
- [PowerShell](#)
- [CLI](#)

Select the following link to deploy the Azure IoT Connector for FHIR using the ARM template in the Azure portal:



On the **Deploy Azure API for FHIR** page:

1. If you want, change the **Subscription** from the default to a different subscription.
2. For **Resource group**, select **Create new**, enter a name for the new resource group, and select **OK**.
3. If you created a new resource group, select a **Region** for the resource group.
4. Enter a name for your new Azure API for FHIR instance in **FHIR Service Name**.
5. Choose the **Location** for your Azure API for FHIR. The location can be the same as or different from the region of the resource group.
6. Provide a name for your Azure IoT Connector for FHIR instance in **IoT Connector Name**.
7. Provide a name for a connection created within Azure IoT Connector for FHIR in **Connection Name**. This connection is used by Azure IoT Central application to push simulated device telemetry into Azure IoT Connector for FHIR.
8. Enter a name for your new Azure IoT Central application in **IoT Central Name**. This application will use *Continuous patient monitoring* template to simulate a device.
9. Choose the location of your IoT Central application from **IoT Central Location** drop-down.
10. Select **Review + create**.
11. Read the terms and conditions, and then select **Create**.

NOTE

The deployment takes a few minutes to complete. Note the names for the Azure API for FHIR service, Azure IoT Central application, and the resource group, which you use to review the deployed resources later.

Review deployed resources

- [Portal](#)
- [PowerShell](#)
- [CLI](#)

Follow these steps to see an overview of your new Azure API for FHIR service:

1. In the [Azure portal](#), search for and select **Azure API for FHIR**.
2. In the FHIR list, select your new service. The **Overview** page for the new Azure API for FHIR service appears.
3. To validate that the new FHIR API account is provisioned, select the link next to **FHIR metadata endpoint** to fetch the FHIR API capability statement. The link has a format of `https://<service-name>.azurehealthcareapis.com/metadata`. If the account is provisioned, a large JSON file is displayed.
4. To validate that the new Azure IoT Connector for FHIR is provisioned, select the **IoT Connector (preview)** from left navigation menu to open the **IoT Connectors** page. The page must show the provisioned Azure IoT Connector for FHIR with *Status* value as *Online*, *Connections* value as *1*, and both *Device mapping* and *FHIR mapping* show *Success* icon.
5. In the [Azure portal](#), search for and select **IoT Central Applications**.
6. In the list of IoT Central Applications, select your new service. The **Overview** page for the new IoT Central application appears.

Connect your IoT data with the Azure IoT Connector for FHIR (preview)

IMPORTANT

The Device mapping template provided in this guide is designed to work with Data Export (legacy) within IoT Central.

IoT Central application currently doesn't provide ARM template or PowerShell and CLI commands to set data export. So, follow the instructions below using Azure portal.

Once you've deployed your IoT Central application, your two out-of-the-box simulated devices will start generating telemetry. For this tutorial, we'll ingest the telemetry from *Smart Vitals Patch* simulator into FHIR via the Azure IoT Connector for FHIR. To export your IoT data to the Azure IoT Connector for FHIR, we'll want to [set up a Data export \(legacy\) within IoT Central](#). On the Data export (legacy) page:

- Pick *Azure Event Hubs* as the export destination.
- Select *Use a connection string* value for **Event Hubs namespace** field.
- Provide Azure IoT Connector for FHIR's connection string obtained in a previous step for the **Connection String** field.
- Keep **Telemetry** option *On* for **Data to Export** field.

View device data in Azure API for FHIR

You can view the FHIR-based Observation resource(s) created by Azure IoT Connector for FHIR on your FHIR server using Postman. Set up your [Postman to access Azure API for FHIR](#) and make a `GET` request to `https://your-fhir-server-url/Observation?code=http://loinc.org|8867-4` to view Observation FHIR resources with heart rate value.

TIP

Ensure that your user has appropriate access to Azure API for FHIR data plane. Use [Azure role-based access control \(Azure RBAC\)](#) to assign required data plane roles.

Clean up resources

When it's no longer needed, delete the resource group, which deletes the resources in the resource group.

- [Portal](#)
- [PowerShell](#)
- [CLI](#)

1. In the [Azure portal](#), search for and select **Resource groups**.
2. In the resource group list, choose the name of your resource group.
3. In the **Overview** page of your resource group, select **Delete resource group**.
4. In the confirmation dialog box, type the name of your resource group, and then select **Delete**.

For a step-by-step tutorial that guides you through the process of creating an ARM template, see the [tutorial to create and deploy your first ARM template](#)

Next steps

In this quickstart guide, you've deployed Azure IoT Connector for FHIR feature in your Azure API for FHIR resource. Select from below next steps to learn more about Azure IoT Connector for FHIR:

Understand different stages of data flow within Azure IoT Connector for FHIR.

[Azure IoT Connector for FHIR data flow](#)

Learn how to configure IoT Connector using device and FHIR mapping templates.

[Azure IoT Connector for FHIR mapping templates](#)

*In the Azure portal, Azure IoT Connector for FHIR is referred to as IoT Connector (preview). FHIR is a registered trademark of HL7 and is used with the permission of HL7.

Deploy JavaScript app to read data from FHIR service

3/11/2021 • 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. If you haven't already, 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

3/11/2021 • 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
4. Select **Public client/native (mobile & desktop)** and set the redirect URI to

`https://www.getpostman.com/oauth2/callback` .

Register an application

* Name

The user-facing display name for this application (this can be changed later).

fhir-public-client-app ✓

Supported account types

Who can use this application or access this API?

- ☒ Accounts in this organizational directory only (CaitlinFHIR1 only - Single tenant)
- ☐ Accounts in any organizational directory (Any Azure AD directory - Multitenant)
- ☐ Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)

[Help me choose...](#)

Redirect URI (optional)

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Public client/native (mobile ... ✓ `https://www.getpostman.com/oauth2/callback` ✓

By proceeding, you agree to the [Microsoft Platform Policies](#) 

Register

Client application settings

Once your client application is registered, copy the Application (client) ID and the Tenant ID from the Overview Page. You will need these two values later when accessing the client.

« Delete Endpoints

Got a second? We would love your feedback on Microsoft identity platform (previously Azure AD for developer). →

view) Display name : LunarHealthcareClient Supported account types

Application (client) ID : Example application ID Redirect URIs

Directory (tenant) ID : Example directory ID Application ID URI

Object ID : Example object ID Managed applications

⌵

Connect with web app

If you have [written your web app](#) to connect with the Azure API for FHIR, you also need to set the correct authentication options.

1. In the left menu, under **Manage**, select **Authentication**.
2. To add a new platform configuration, select **Web**.
3. Set up 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.
4. Select the **Access Token** and **ID token** check boxes.

App registrations > LunarHealthcareClient | Authentication

Platform configurations

Depending on the platform or device this application is targeting, additional configuration may be required such as redirect URIs, specific authentication settings, or fields specific to the platform.

+ Add a platform

Mobile and desktop applications

Redirect URIs: 1

Supported account types

Who can use this application or access this API?

☒ Accounts in this organizational directory only (Chun Lin Goh only - Single tenant)

☐ Accounts in any organizational directory (Any Azure AD directory - Multitenant)

Help me decide...

Due to temporary differences in supported functionality, we don't recommend enabling personal Microsoft accounts for an existing registration. If you need to enable personal accounts, you can do so using the manifest editor. [Learn more about these restrictions.](#)

Advanced settings

Default client type

Treat application as a public client. Required for the use of the following flows where a redirect URI is not used: Yes No

Resource owner password credential (ROPC) [Learn more...](#)

Configure Web

Redirect URIs

The URIs that we will accept as destinations when returning authentication responses (tokens) after successfully authenticating users. Also referred to as reply URLs. [Learn more about Redirect URIs and their restrictions](#)

https://hcapidocswebapp.azurewebsites.net ✓

Logout URL

This is where we send a request to have the application clear the user's session data. This is required for single sign-out to work correctly.

e.g. https://myapp.com/logout

Implicit grant

Allows an application to request a token directly from the authorization endpoint. Checking Access tokens and ID tokens is recommended only if the application has a single-page architecture (SPA), has no back-end components, does not use the latest version of MSAL.js with auth code flow, or it invokes a web API via JavaScript. ID Token is needed for ASP.NET Core Web Apps. [Learn more about the implicit grant flow](#)

To enable the implicit grant flow, select the tokens you would like to be issued by the authorization endpoint:

☒ Access tokens

☒ ID tokens

Configure Cancel

Add API permissions

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**.

Home > User | App registrations > LunarHealthcareClient | API permissions

Search (Cmd+/) Refresh

Overview
Quickstart
Integration assistant (preview)

Manage

Branding
Authentication
Certificates & secrets
Token configuration
API permissions
Expose an API
Owners
Roles and administrators (Preview)
Manifest

Support + Troubleshooting
Troubleshooting
New support request

Configured permissions

Applications are authorized to call APIs when they are granted permissions by us. All the permissions the application needs. [Learn more about permissions and configurations](#)

[+ Add a permission](#) [Grant admin consent for Chun Lin Goh](#)

API / Permissions name	Type	Description
▼ Azure Healthcare APIs (1)		
user_impersonation	Delegated	Access Azure Healthcare APIs
▼ Microsoft Graph (1)		
User.Read	Delegated	Sign in and read user profile

Request API permissions

← All APIs

Azure Healthcare APIs
https://azurehealthcareapis.us

What type of permissions does your application require?

Delegated permissions
Your application needs to access the API as the signed-in user.

Application permissions
Your application runs as a background service or daemon without signed-in user.

Select permissions

Type to search

Permission	Admin consent required
<input checked="" type="checkbox"/> user_impersonation Access Azure Healthcare APIs	-

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

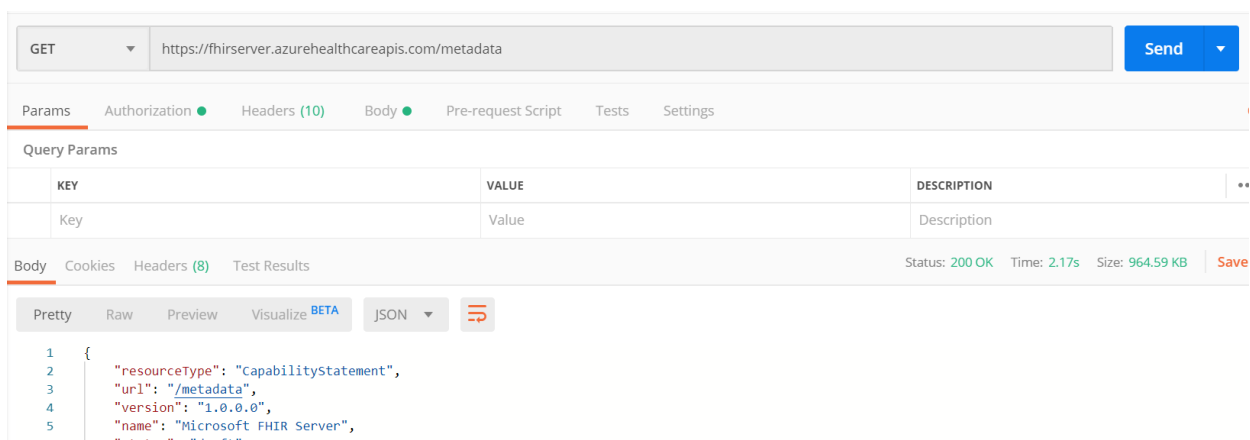
3/11/2021 • 2 minutes to read • [Edit Online](#)

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

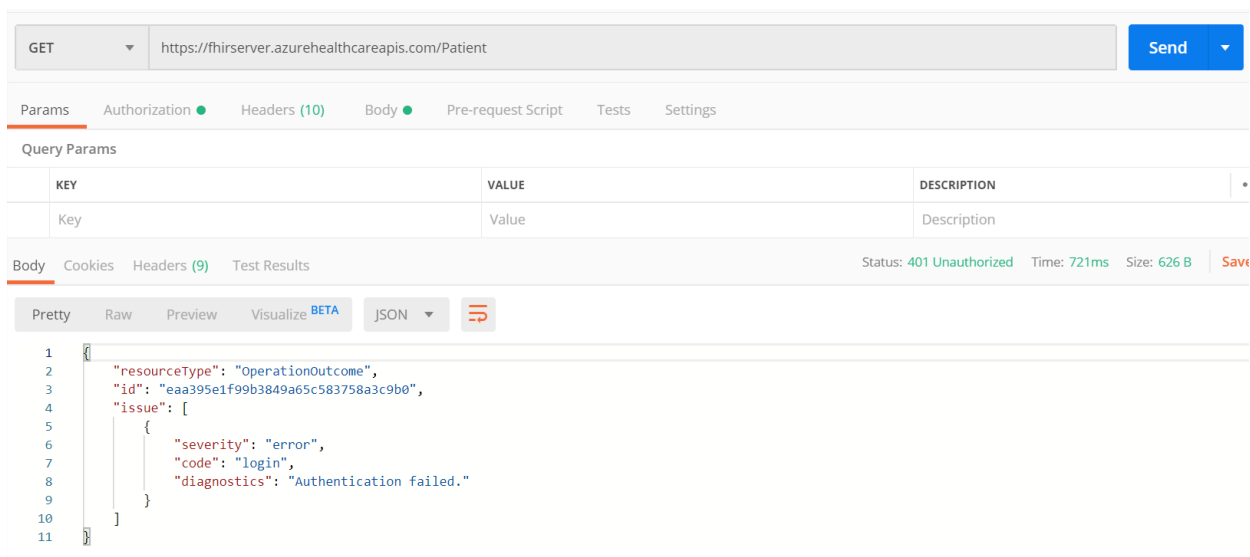
First we will get the capability statement for your Azure API for FHIR.

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

The screenshot shows the Postman interface for an OAuth 2.0 authorization request. The URL is `https://fhirserver.azurehealthcareapis.com/Patient` and the method is `GET`. The `Authorization` tab is selected, showing the `OAuth 2.0` type. A warning message states: "Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)". The `Access Token` field contains a long alphanumeric string, and a `Get New Access Token` button is visible. The `Body` tab is selected at the bottom, showing the response in `JSON` format. The response status is `200 OK` with a time of `720ms` and a size of `1.09 KB`. The response body is a JSON object:

```
1 {
2   "resourceType": "Bundle",
3   "id": "164a12e57fa8f947a65e4c03eda1f519",
4   "type": "search",
5   "total": 1,
6   "link": [
7     {
8       "rel": "self",
9       "url": "https://fhirserver.azurehealthcareapis.com/Patient/164a12e57fa8f947a65e4c03eda1f519"
10    },
11    {
12      "rel": "next",
13      "url": "https://fhirserver.azurehealthcareapis.com/Patient/164a12e57fa8f947a65e4c03eda1f519"
14    }
15  ],
16   "entry": [
17     {
18       "resource": {
19         "resourceType": "Patient",
20         "id": "164a12e57fa8f947a65e4c03eda1f519",
21         "name": {
22           "family": "Doe",
23           "given": "John"
24         },
25         "gender": "male",
26         "birthDate": "1970-01-01",
27         "address": [
28           {
29             "line": [
30               "123 Main St",
31               "Apt 456"
32             ],
33             "city": "New York",
34             "state": "NY",
35             "postalCode": "10001",
36             "country": "US"
37           }
38         ],
39         "email": [
40           {
41             "address": "john.doe@example.com"
42           }
43         ],
44         "phone": [
45           {
46             "number": "212-555-1234",
47             "use": "home"
48           }
49         ],
50         "photo": [
51           {
52             "url": "https://fhirserver.azurehealthcareapis.com/Patient/164a12e57fa8f947a65e4c03eda1f519/photo"
53           }
54         ],
55         "active": true
56       }
57     }
58   ]
59 }
```

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.

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** `https://fhirserver.azurehealthcareapis.com/Patient`
- Body:** The JSON payload from the previous block is shown in the "Body" tab.
- Response:** The "Body" tab shows the response JSON:


```

1 {
2   "resourceType": "Patient",
3   "id": "a4f6f7e8-6bf0-4af2-bd1e-39ffc522796c",
4   "meta": {
5     "versionId": "1",
6     "lastUpdated": "2020-01-03T22:49:07.826+00:00"
7   },
8   "active": true,
9   "name": [
10    {

```
- Status:** 201 Created
- Time:** 710ms
- Size:** 906 B
- Buttons:** Pretty, Raw, Preview, Visualize, JSON, Save

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

3/11/2021 • 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.

Web App

[Basics](#) [Monitoring](#) [Tags](#) [Review + create](#)

App Service Web Apps lets you quickly build, deploy, and scale enterprise-grade web, mobile, and API apps running on any platform. Meet rigorous performance, scalability, security and compliance requirements while using a fully managed platform to perform infrastructure maintenance. [Learn more](#)

Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *	CV-VSE
Resource Group *	FHIR1

[Create new](#)

Instance Details

Name *	hcapidocswebapp
Publish *	Code Docker Container
Runtime stack *	.NET Core 3.0 (Current)
Operating System *	Linux Windows
Region *	Central US

[Not finding your App Service Plan? Try a different region.](#)

App Service Plan

App Service plan pricing tier determines the location, features, cost and compute resources associated with your app. [Learn more](#)

Windows Plan (Central US) *	(New) ASP-FHIR1-b6b9
Sku and size *	Standard S1 100 total ACU, 1.75 GB memory

[Create new](#) [Change size](#)

[Review + create](#)

[< Previous](#)

[Next : Monitoring >](#)

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);
      })
    }
  </script>
</body>
</html>
```

```

    }

    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.

Access Azure API for FHIR with Postman

3/29/2021 • 4 minutes to read • [Edit Online](#)

A client application can access the Azure API for FHIR through a [REST API](#). To send requests, view responses, and debug your application as it is being built, use an API testing tool of your choice. In this tutorial, we'll walk you through the steps of accessing the FHIR server using [Postman](#).

Prerequisites

- A FHIR endpoint in Azure.

To deploy the Azure API for FHIR (a managed service), you can use the [Azure portal](#), [PowerShell](#), or [Azure CLI](#).

- A registered [confidential client application](#) to access the FHIR service.
- You have granted permissions to the confidential client application, for example, "FHIR Data Contributor", to access the FHIR service. For more information, see [Configure Azure RBAC for FHIR](#).
- Postman installed.

For more information about Postman, see [Get Started with Postman](#).

FHIR server and authentication details

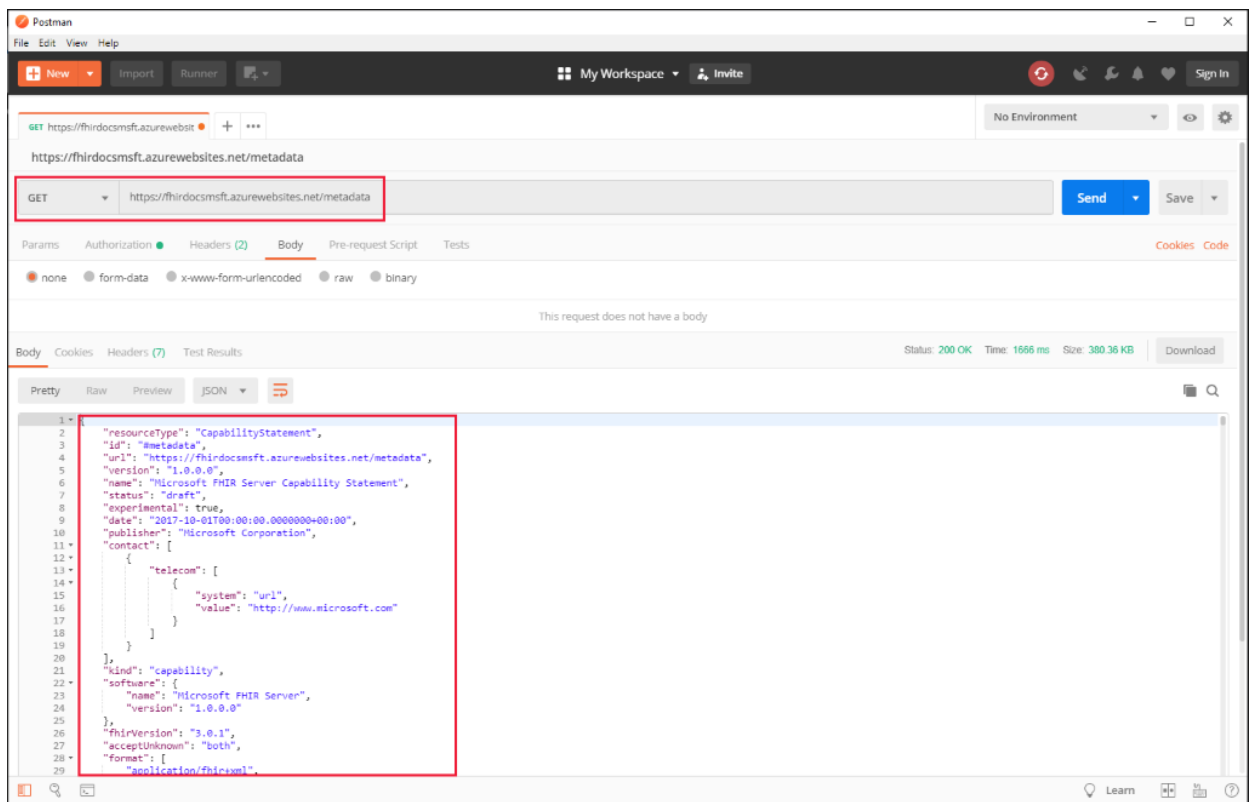
To use Postman, the following authentication parameters are required:

- 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` that is usually the URL of the FHIR server, for example, `https://<FHIR-SERVER-NAME>.azurehealthcareapis.com` OR `https://azurehealthcareapis.com`.
- The `client_id` or application ID of the [confidential client application](#) used for accessing the FHIR service.
- The `client_secret` or application secret of the confidential 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

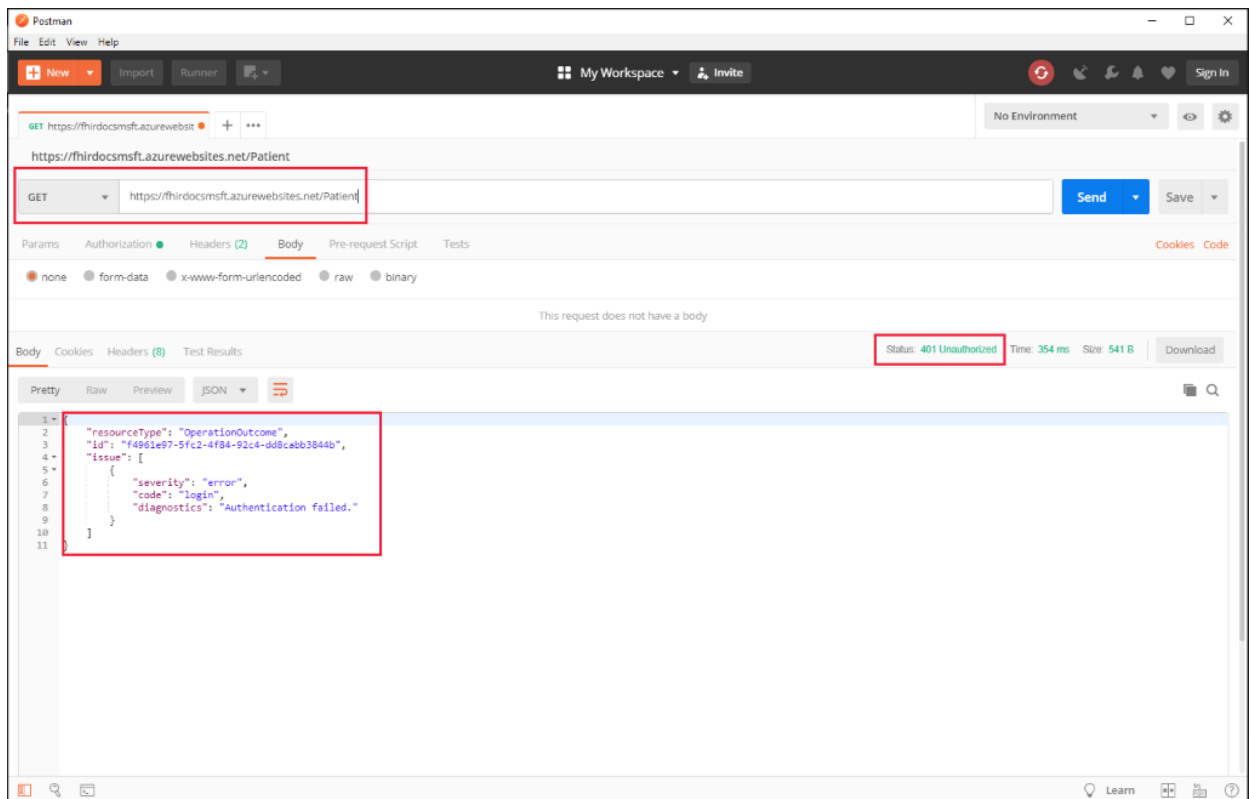
Open Postman, and then select GET to make a request to `https://fhir-server-url/metadata`.



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`. This endpoint is accessible without authentication.

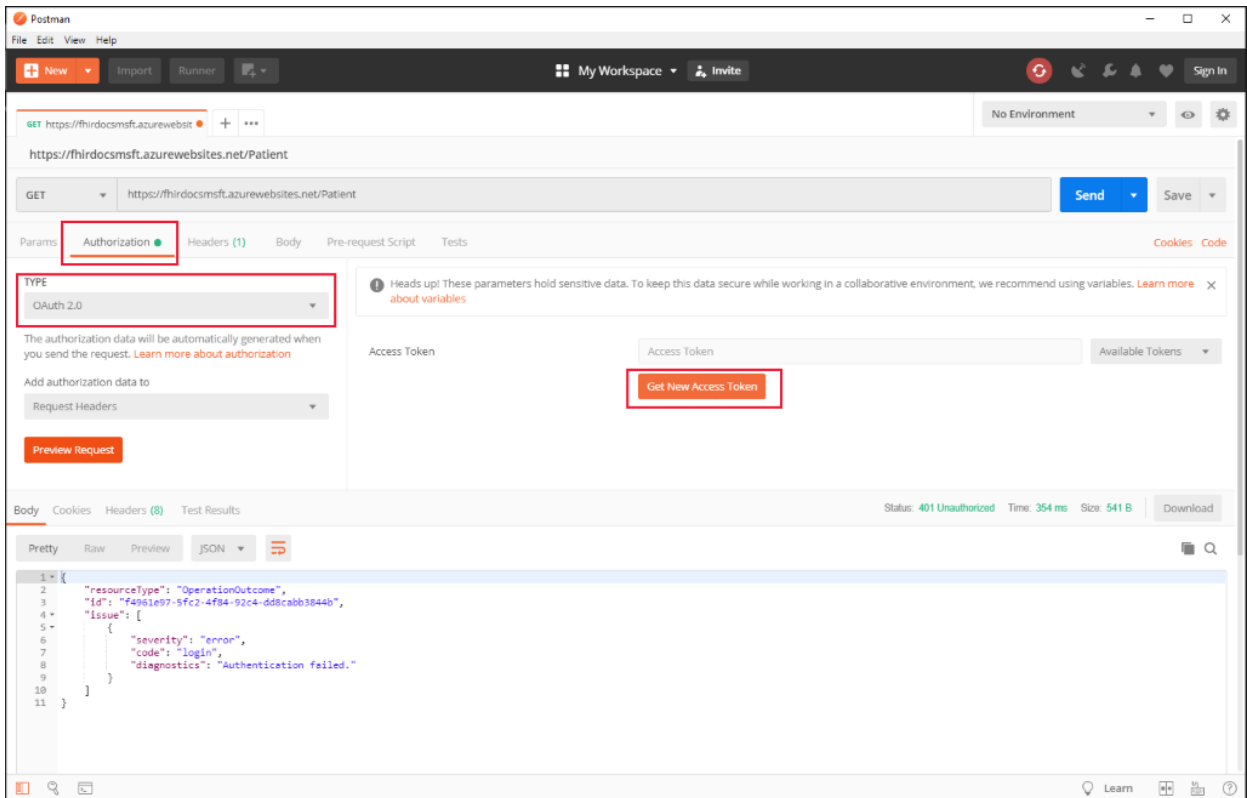
If you attempt to access restricted resources, an "Authentication failed" response occurs.



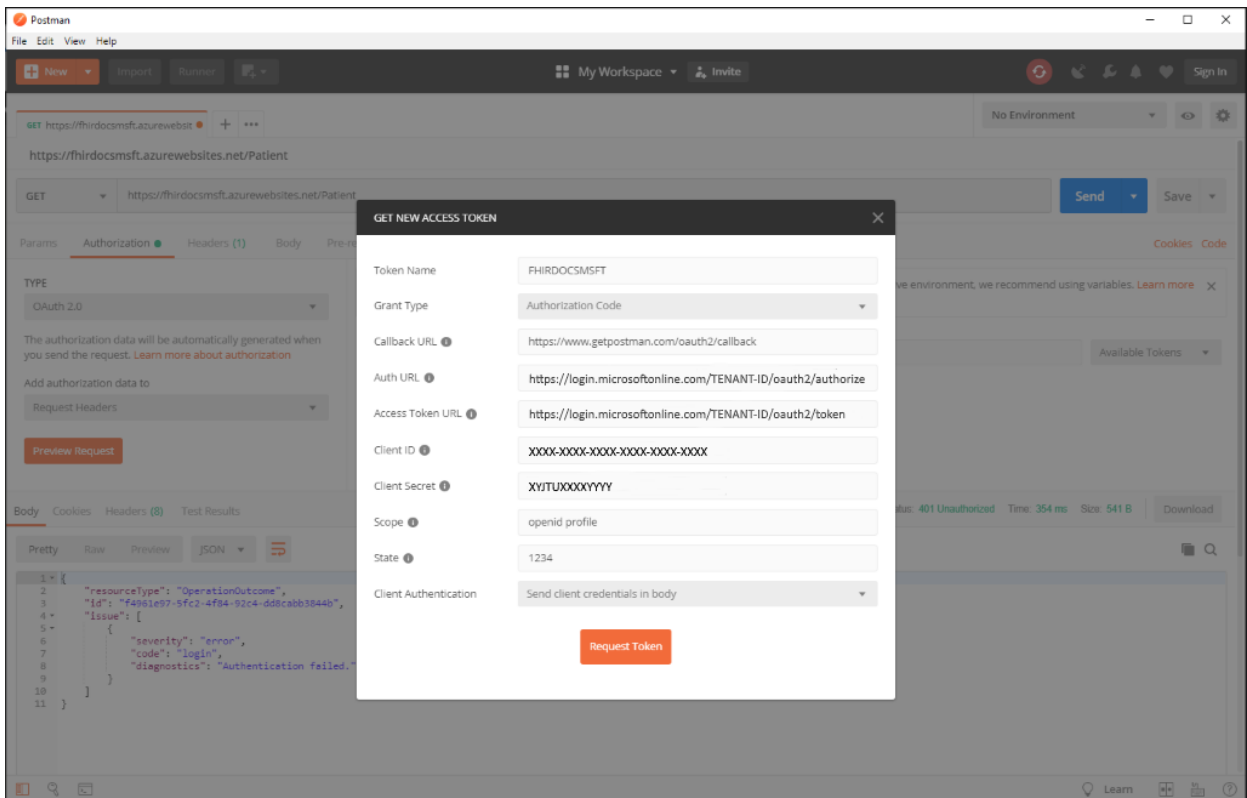
Obtaining an access token

Select **Get New Access Token**.

To obtain a valid access token, select **Authorization** and select **OAuth 2.0** from the **TYPE** drop-down menu.



Select **Get New Access Token**.



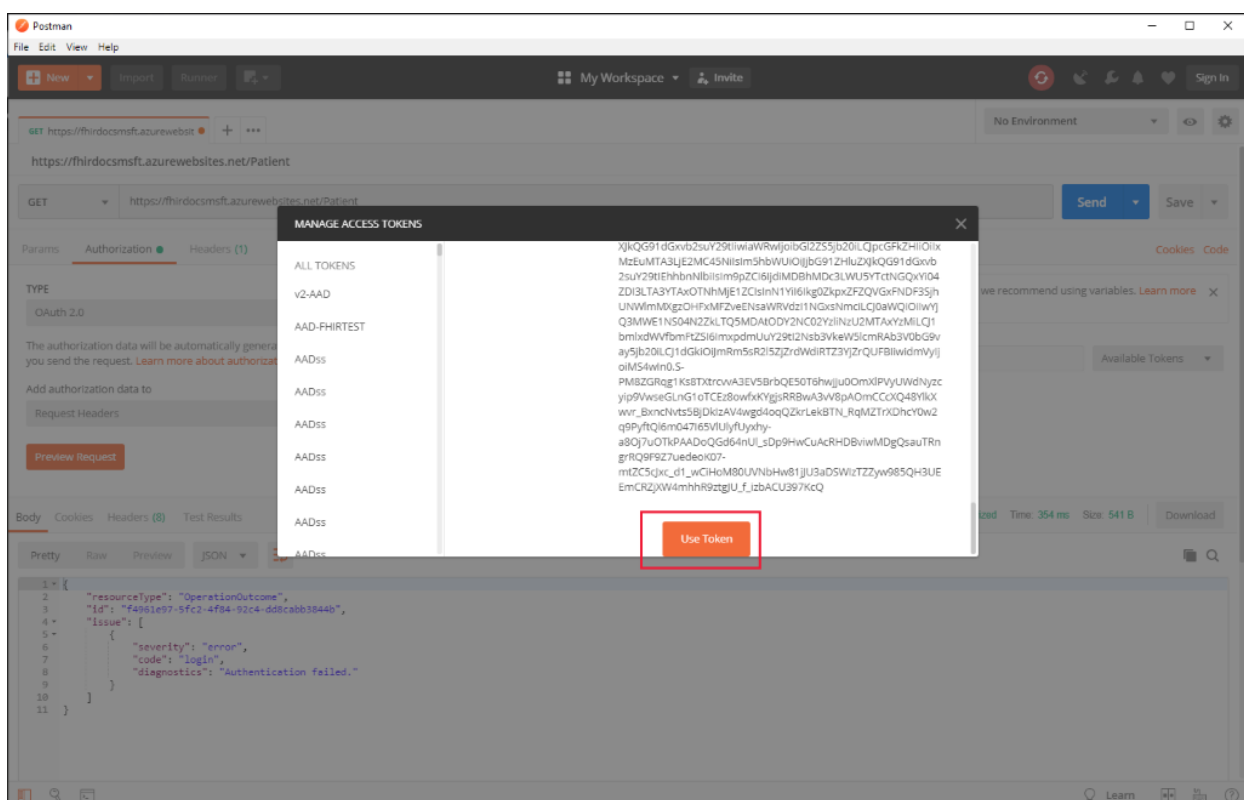
In the **Get New Access Token** dialog box, enter the following details:

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

FIELD	EXAMPLE VALUE	COMMENT
Callback URL	<code>https://www.getpostman.com/oauth2/callback</code>	
Auth URL	<code>https://login.microsoftonline.com/{TENANT ID}/oauth2/authorize?resource=<audience></code>	audience is <code>https://MYACCOUNT.azurehealthcareapis.com</code> for Azure API for FHIR
Access Token URL	<code>https://login.microsoftonline.com/{TENANT ID}/oauth2/token</code>	
Client ID	<code>XXXXXXXX-XXX-XXXX-XXXX-XXXXXXXXXXXX</code>	Application ID
Client Secret	<code>XXXXXXXX</code>	Secret client key
Scope	<code><Leave Blank></code>	Scope is not used; therefore, it can be left blank.
State	<code>1234</code>	State is an opaque value to prevent cross-site request forgery. It is optional and can take an arbitrary value such as '1234'.
Client Authentication	Send client credentials in body	

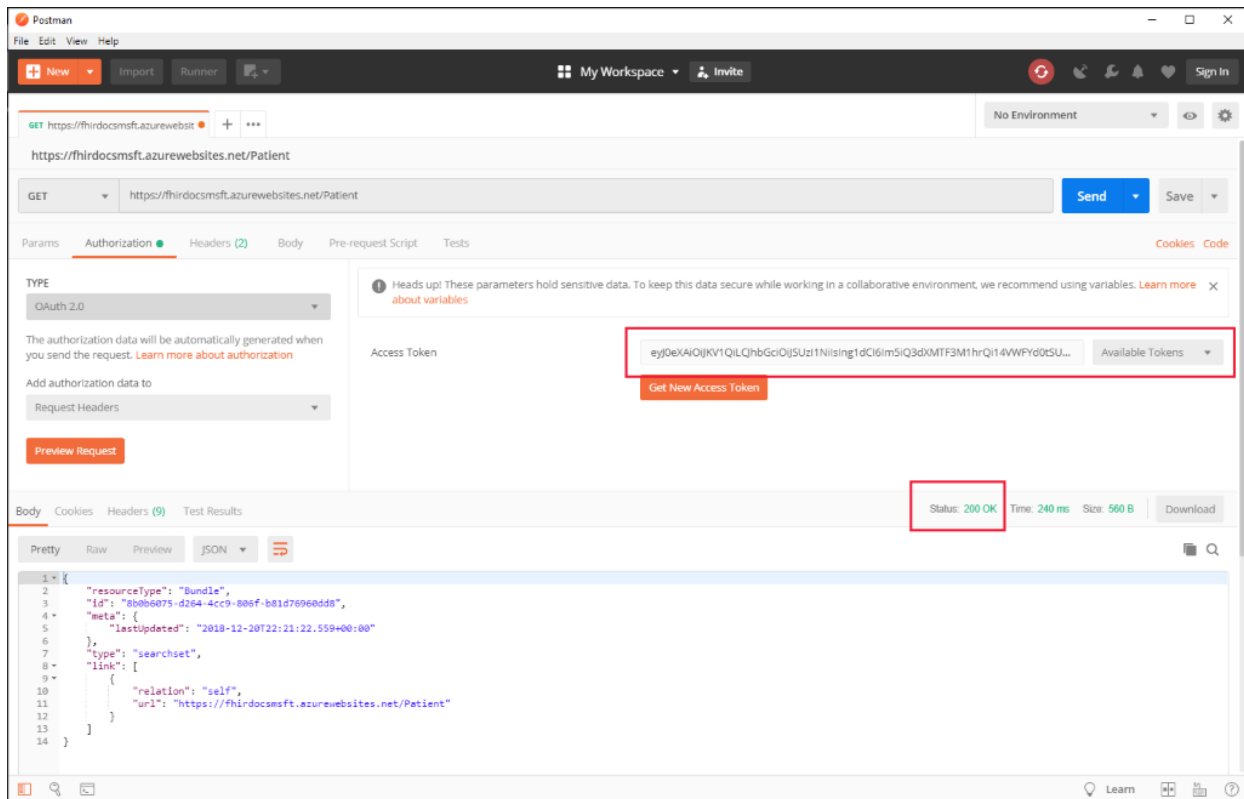
Select **Request Token** to be guided through the Azure Active Directory Authentication flow, and a token will be returned to Postman. If an authentication failure occurs, refer to the Postman Console for more details. **Note:** On the ribbon, select **View**, and then select **Show Postman Console**. The keyboard shortcut to the Postman Console is **Alt-Ctrl+C**.

Scroll down to view the returned token screen, and then select **Use Token**.



Refer to the **Access Token** field to view the newly populated token. If you select **Send** to repeat the **Patient**

resource search, a **Status** `200 OK` gets returned. A returned status `200 OK` indicates a successful HTTP request.



In the *Patient search* example, there are no patients in the database such that the search result is empty.

You can inspect the access token using a tool like jwt.ms. An example of the content is shown below.

```
{
  "aud": "https://MYACCOUNT.azurehealthcareapis.com",
  "iss": "https://sts.windows.net/{TENANT-ID}/",
  "iat": 1545343803,
  "nbf": 1545343803,
  "exp": 1545347703,
  "acr": "1",
  "aio": "AUQAu/8JXXXXXXXdQxcxn1eis459j70Kf9DwcUj1KY3I2G/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. If your token is from the correct issuer (`iss` claim) and has the correct audience (`aud` claim), but you are still unable to access the FHIR API, it is likely that the user or service principal (`oid` claim) doesn't have access to the FHIR data plane. We recommend you use [Azure role-based access control \(Azure RBAC\)](#) to assign data plane roles to users. If you're using an external, secondary Azure Active directory tenant for your data plane, you'll need to [Configure local RBAC for FHIR](#) assignments.

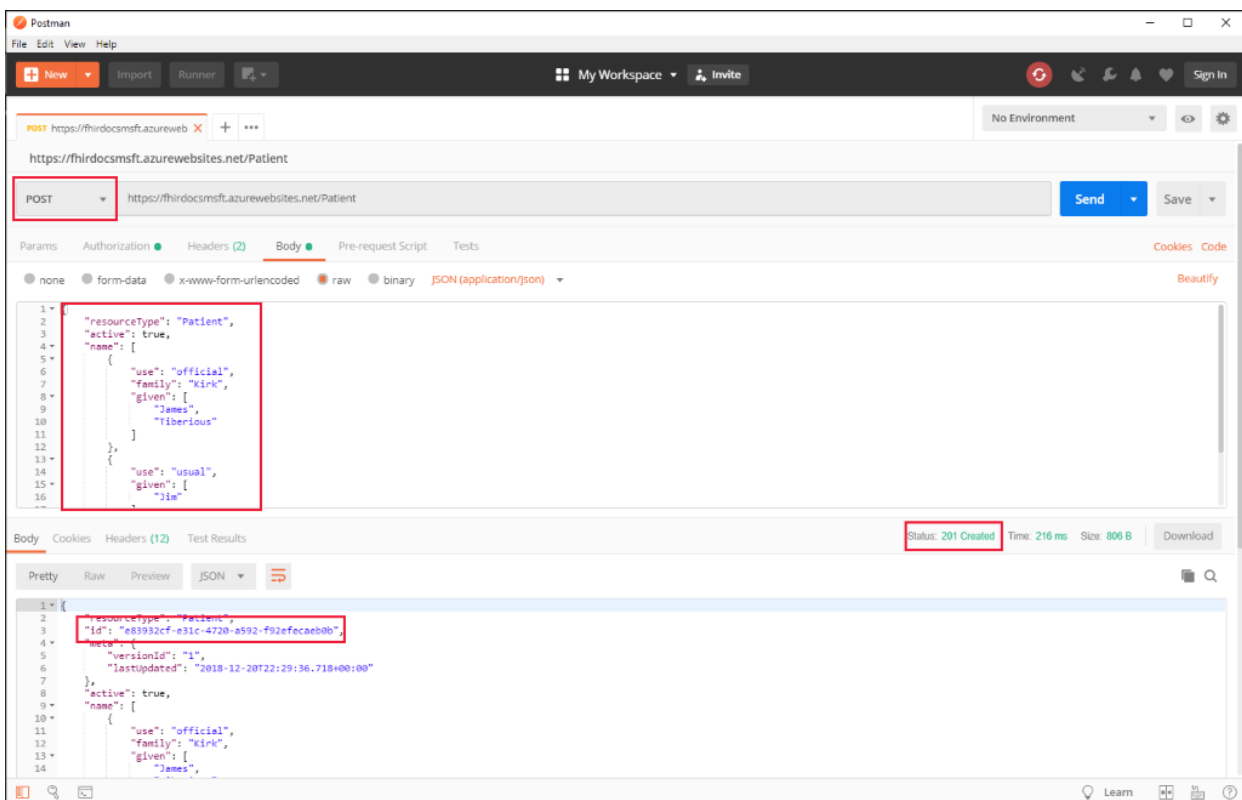
It's also possible to get a token for the [Azure API for FHIR using the Azure CLI](#). If you're using a token obtained with the Azure CLI, you should use Authorization type *Bearer Token*. Paste the token in directly.

Inserting a patient

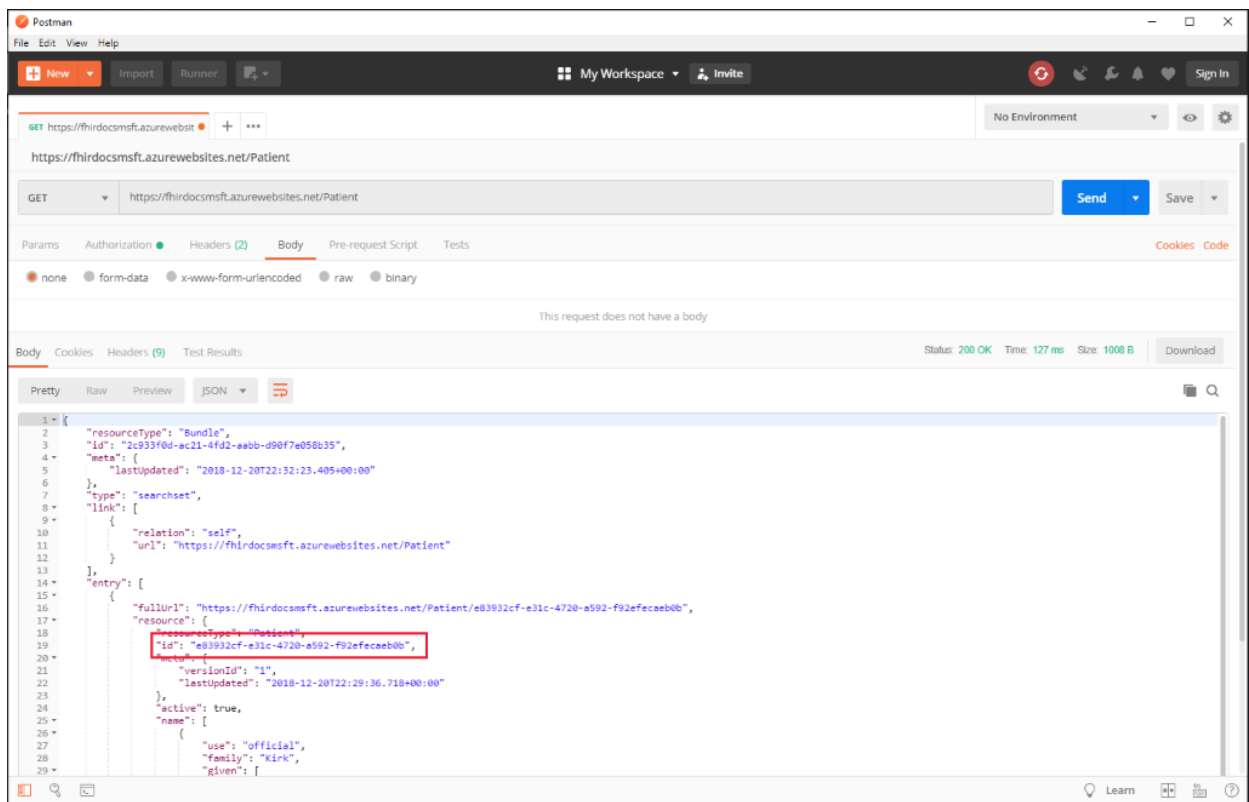
With a valid access token, you can now insert a new patient. In Postman, change the method by selecting **Post**, and then add the following JSON document in the body of the request.

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

Select **Send** to determine 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 the Azure API for FHIR using Postman. For more information about the Azure API for FHIR features, see

[Supported features](#)

Tutorial: Azure Active Directory SMART on FHIR proxy

3/11/2021 • 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:

Search (Ctrl+ /)

Save

Discard

Refresh

Overview

Activity log

Access control (IAM)

Tags

Settings

Authentication

CORS

Locks

Support + troubleshooting

New support request

View and configure authentication settings; specify Azure AD object ID (Users or Apps) that should be allowed to access this Azure API for FHIR.

Authority must be registered to Azure AD and in the following format: `https://(Azure-AD-endpoint)/(tenant-id)`. Example: `https://login.microsoftonline.com/contoso.onmicrosoft.com`. Audience must be a URI or GUID secured by Azure AD. Please refer to <https://docs.microsoft.com/azure/healthcare-apis/register-resource-azure-ad-client-app> for details.

* Authority

https://login.microsoftonline.com/TENANT-ID

* Audience

https://MYFHIRAPI.azurehealthcareapis.com

* Allowed object IDs ⓘ

abababab-abab-abab-abab-abababababab

SMART on FHIR proxy ⓘ

☒

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:

Search (Ctrl+ /)

Save

Discard

Refresh

Overview

Activity log

Access control (IAM)

Tags

Settings

Authentication

CORS

Locks

Support + troubleshooting

New support request

Cross-origin resource sharing (CORS) is a security mechanism that allows a web page from one domain or origin to access a resource with a different domain (a cross-domain request). Without features like CORS, websites are restricted to accessing resources from the same origin through what is known as same-origin policy. Please refer to <https://docs.microsoft.com/azure/healthcare-apis/configure-cross-origin-resource-sharing> for details on how to configure CORS.

Origins ⓘ

*

Headers ⓘ

*

Methods ⓘ

6 selected

Max age ⓘ

600

Allow credentials ⓘ

☐

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/aHR0cHM6Ly9sb2NhbgVhc3Q6NTAwMS9zYW1wbGVhcHAvaW5kZXguaHRtbA
```

In that reply, `aHR0cHM6Ly9sb2NhbgVhc3Q6NTAwMS9zYW1wbGVhcHAvaW5kZXguaHRtbA` 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:

Save Discard Got feedback?

Redirect URIs

The URIs that we will accept as destinations when returning authentication responses (tokens) after successfully authenticating users. Also referred to as reply URLs. [Learn more about adding support for web, mobile and desktop clients](#)

TYPE	REDIRECT URI
Public client (mobile & desktop)	https://www.getpostman.com/oauth2/callback
Public client (mobile & desktop)	https://MYFHIRAPI.azurehealthcareapis.com/AadSmartOnFhirProxy/callback/aHR0cHM6Ly9sb2NhbgVhc3Q6NTAwMS9zYW1wbGVhcHAvaW5kZXguaHRtbA
Web	e.g. https://myapp.com/auth

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:

Microsoft FHIR Server SMART on FHIR App Launcher

Launch parameters

Patient:

Encounter:

Practitioner:

Application URL:

FHIR Server URL:

Launch context

```
{
  "patient": "552dfc56-9a96-4717-8a74-36bce46dd54f"
}
```

Launch URL

Launch

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:

Microsoft FHIR Server SMART on FHIR Sample App

Patient Resource

```
{
  "resourceType": "Patient",
  "id": "552dfc56-9a96-4717-8a74-36bce46dd54f",
  "meta": {
    "versionId": "1",
    "lastUpdated": "2019-04-02T11:44:24.9103823+00:00"
  },
  "active": true,
  "name": [
```

Token Response

```
{
  "token_type": "Bearer",
  "scope": "user_impersonation",
  "expires_in": "3599",
  "ext_expires_in": "3599",
  "expires_on": "1554209735",
  "not_before": "1554205835",
  "resource": "...",
  "pwd_exp": "714132",
```

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](#)

Tutorial: Receive device data through Azure IoT Hub

4/21/2021 • 5 minutes to read • [Edit Online](#)

Azure IoT Connector for Fast Healthcare Interoperability Resources (FHIR®)* provides you the capability to ingest data from Internet of Medical Things (IoMT) devices into Azure API for FHIR. The [Deploy Azure IoT Connector for FHIR \(preview\) using Azure portal](#) quickstart showed an example of device managed by Azure IoT Central [sending telemetry](#) to Azure IoT Connector for FHIR. Azure IoT Connector for FHIR can also work with devices provisioned and managed through Azure IoT Hub. This tutorial provides the procedure to connect and route device data from Azure IoT Hub to Azure IoT Connector for FHIR.

Prerequisites

- An active Azure subscription - [Create one for free](#)
- Azure API for FHIR resource with at least one Azure IoT Connector for FHIR - [Deploy Azure IoT Connector for FHIR \(preview\) using Azure portal](#)
- Azure IoT Hub resource connected with real or simulated device(s) - [Create an IoT hub using the Azure portal](#)

TIP

If you are using an Azure IoT Hub simulated device application, feel free to pick the application of your choice amongst different supported languages and systems.

Get connection string for Azure IoT Connector for FHIR (preview)

Azure IoT Hub requires a connection string to securely connect with your Azure IoT Connector for FHIR. Create a new connection string for your Azure IoT Connector for FHIR as described in [Generate a connection string](#). Preserve this connection string to be used in the next step.

Azure IoT Connector for FHIR uses an Azure Event Hub instance under the hood to receive device messages. The connection string created above is basically the connection string to this underlying Event Hub.

Connect Azure IoT Hub with the Azure IoT Connector for FHIR (preview)

Azure IoT Hub supports a feature called [message routing](#) that provides capability to send device data to various Azure services like Event Hub, Storage Account, and Service Bus. Azure IoT Connector for FHIR leverages this feature to connect and send device data from Azure IoT Hub to its Event Hub endpoint.

NOTE

At this time you can only use PowerShell or CLI command to [create message routing](#) because Azure IoT Connector for FHIR's Event Hub is not hosted on the customer subscription, hence it won't be visible to you through the Azure portal. Though, once the message route objects are added using PowerShell or CLI, they are visible on the Azure portal and can be managed from there.

Setting up a message routing consists of two steps.

Add an endpoint

This step defines an endpoint to which the IoT Hub would route the data. Create this endpoint using either [Add-AzIoTHubRoutingEndpoint](#) PowerShell command or [az iot hub routing-endpoint create](#) CLI command, based on your preference.

Here is the list of parameters to use with the command to create an endpoint:

POWERSHELL PARAMETER	CLI PARAMETER	DESCRIPTION
ResourceGroupName	resource-group	Resource group name of your IoT Hub resource.
Name	hub-name	Name of your IoT Hub resource.
EndpointName	endpoint-name	Use a name that you would like to assign to the endpoint being created.
EndpointType	endpoint-type	Type of endpoint that IoT Hub needs to connect with. Use literal value of "EventHub" for PowerShell and "eventhub" for CLI.
EndpointResourceGroup	endpoint-resource-group	Resource group name for your Azure IoT Connector for FHIR's Azure API for FHIR resource. You can get this value from the Overview page of Azure API for FHIR.
EndpointSubscriptionId	endpoint-subscription-id	Subscription Id for your Azure IoT Connector for FHIR's Azure API for FHIR resource. You can get this value from the Overview page of Azure API for FHIR.
ConnectionString	connection-string	Connection string to your Azure IoT Connector for FHIR. Use the value you obtained in the previous step.

Add a message route

This step defines a message route using the endpoint created above. Create a route using either [Add-AzIoTHubRoute](#) PowerShell command or [az iot hub route create](#) CLI command, based on your preference.

Here is the list of parameters to use with the command to add a message route:

POWERSHELL PARAMETER	CLI PARAMETER	DESCRIPTION
ResourceGroupName	g	Resource group name of your IoT Hub resource.
Name	hub-name	Name of your IoT Hub resource.
EndpointName	endpoint-name	Name of the endpoint you have created above.
RouteName	route-name	A name you want to assign to message route being created.

POWERSHELL PARAMETER	CLI PARAMETER	DESCRIPTION
Source	source-type	Type of data to send to the endpoint. Use literal value of "DeviceMessages" for PowerShell and "devicemessages" for CLI.

Send device message to IoT Hub

Use your device (real or simulated) to send the sample heart rate message shown below to Azure IoT Hub. This message will get routed to Azure IoT Connector for FHIR, where the message will be transformed into a FHIR Observation resource and stored into the Azure API for FHIR.

```
{
  "HeartRate": 80,
  "RespiratoryRate": 12,
  "HeartRateVariability": 64,
  "BodyTemperature": 99.08839032397609,
  "BloodPressure": {
    "Systolic": 23,
    "Diastolic": 34
  },
  "Activity": "walking"
}
```

IMPORTANT

Make sure to send the device message that conforms to the [mapping templates](#) configured with your Azure IoT Connector for FHIR.

View device data in Azure API for FHIR

You can view the FHIR Observation resource(s) created by Azure IoT Connector for FHIR on Azure API for FHIR using Postman. Set up your [Postman to access Azure API for FHIR](#) and make a `GET` request to `https://your-fhir-server-url/Observation?code=http://loinc.org|8867-4` to view Observation FHIR resources with heart rate value submitted in the above sample message.

TIP

Ensure that your user has appropriate access to Azure API for FHIR data plane. Use [Azure role-based access control \(Azure RBAC\)](#) to assign required data plane roles.

Next steps

In this quickstart guide, you set up Azure IoT Hub to route device data to Azure IoT Connector for FHIR. Select from below next steps to learn more about Azure IoT Connector for FHIR:

Understand different stages of data flow within Azure IoT Connector for FHIR.

[Azure IoT Connector for FHIR data flow](#)

Learn how to configure IoT Connector using device and FHIR mapping templates.

[Azure IoT Connector for FHIR mapping templates](#)

*In the Azure portal, Azure IoT Connector for FHIR is referred to as IoT Connector (preview). FHIR is a registered trademark of HL7 and is used with the permission of HL7.

Centers for Medicare and Medicaid Services (CMS) Interoperability and Patient Access rule introduction

6/8/2021 • 4 minutes to read • [Edit Online](#)

In this series of tutorials, we'll cover a high-level summary of the Center for Medicare and Medicaid Services (CMS) Interoperability and Patient Access rule, and the technical requirements outlined in this rule. We'll walk through the various implementation guides referenced for this rule. We'll also provide details on how to configure the Azure API for FHIR to support these implementation guides.

Rule overview

The CMS released the [Interoperability and Patient Access rule](#) on May 1, 2020. This rule requires free and secure data flow between all parties involved in patient care (patients, providers, and payers) to allow patients to access their health information when they need it. Interoperability has plagued the healthcare industry for decades, resulting in siloed data that causes negative health outcomes with higher and unpredictable costs for care. CMS is using their authority to regulate Medicare Advantage (MA), Medicaid, Children's Health Insurance Program (CHIP), and Qualified Health Plan (QHP) issuers on the Federally Facilitated Exchanges (FfEs) to enforce this rule.

In August 2020, CMS detailed how organizations can meet the mandate. To ensure that data can be exchanged securely and in a standardized manner, CMS identified FHIR version release 4 (R4) as the foundational standard required for the data exchange.

There are three main pieces to the Interoperability and Patient Access ruling:

- **Patient Access API (Required July 1, 2021)** – CMS-regulated payers (as defined above) are required to implement and maintain a secure, standards-based API that allows patients to easily access their claims and encounter information, including cost, as well as a defined subset of their clinical information through third-party applications of their choice.
- **Provider Directory API (Required July 1, 2021)** – CMS-regulated payers are required by this portion of the rule to make provider directory information publicly available via a standards-based API. Through making this information available, third-party application developers will be able to create services that help patients find providers for specific care needs and clinicians find other providers for care coordination.
- **Payer-to-Payer Data Exchange (Required January 1, 2022)** – CMS-regulated payers are required to exchange certain patient clinical data at the patient's request with other payers. While there's no requirement to follow any kind of standard, applying FHIR to exchange this data is encouraged.

Key FHIR concepts

As mentioned above, FHIR R4 is required to meet this mandate. In addition, there have been several implementation guides developed that provide guidance for the rule. [Implementation guides](#) provide extra context on top of the base FHIR specification. This includes defining additional search parameters, profiles, extensions, operations, value sets, and code systems.

The Azure API for FHIR has the following capabilities to help you configure your database for the various implementation guides:

- [Support for RESTful interactions](#)
- [Storing and validating profiles](#)

- [Defining and indexing custom search parameters](#)
- [Converting data](#)

Patient Access API Implementation Guides

The Patient Access API describes adherence to four FHIR implementation guides:

- [CARIN IG for Blue Button®](#): Payers are required to make patients' claims and encounters data available according to the CARIN IG for Blue Button Implementation Guide (C4BB IG). The C4BB IG provides a set of resources that payers can display to consumers via a FHIR API and includes the details required for claims data in the Interoperability and Patient Access API. This implementation guide uses the ExplanationOfBenefit (EOB) Resource as the main resource, pulling in other resources as they are referenced.
- [HL7 FHIR Da Vinci PDex IG](#): The Payer Data Exchange Implementation Guide (PDex IG) is focused on ensuring that payers provide all relevant patient clinical data to meet the requirements for the Patient Access API. This uses the US Core profiles on R4 Resources and includes (at a minimum) encounters, providers, organizations, locations, dates of service, diagnoses, procedures, and observations. While this data may be available in FHIR format, it may also come from other systems in the format of claims data, HL7 V2 messages, and C-CDA documents.
- [HL7 US Core IG](#): The HL7 US Core Implementation Guide (US Core IG) is the backbone for the PDex IG described above. While the PDex IG limits some resources even further than the US Core IG, many resources just follow the standards in the US Core IG.
- [HL7 FHIR Da Vinci - PDex US Drug Formulary IG](#): Part D Medicare Advantage plans have to make formulary information available via the Patient API. They do this using the PDex US Drug Formulary Implementation Guide (USDF IG). The USDF IG defines a FHIR interface to a health insurer's drug formulary information, which is a list of brand-name and generic prescription drugs that a health insurer agrees to pay for. The main use case of this is so that patients can understand if there are alternative drug available to one that has been prescribed to them and to compare drug costs.

Provider Directory API Implementation Guide

The Provider Directory API describes adherence to one implementation guide:

- [HL7 Da Vinci PDex Plan Network IG](#): This implementation guide defines a FHIR interface to a health insurer's insurance plans, their associated networks, and the organizations and providers that participate in these networks.

Touchstone

To test adherence to the various implementation guides, [Touchstone](#) is a great resource. Throughout the upcoming tutorials, we'll focus on ensuring that the Azure API for FHIR is configured to successfully pass various Touchstone tests. The Touchstone site has a lot of great documentation to help you get up and running.

Next steps

Now that you have a basic understanding of the Interoperability and Patient Access rule, implementation guides, and available testing tool (Touchstone), we'll walk through setting up the Azure API for FHIR for the CARIN IG for Blue Button.

[CARIN Implementation Guide for Blue Button](#)

CARIN Implementation Guide for Blue Button®

6/8/2021 • 2 minutes to read • [Edit Online](#)

In this tutorial, we'll walk through setting up the Azure API for FHIR to pass the [Touchstone](#) tests for the [CARIN Implementation Guide for Blue Button](#) (C4BB IG).

Touchstone capability statement

The first test that we'll focus on is testing the Azure API for FHIR against the [C4BB IG capability statement](#). If you run this test against the Azure API for FHIR without any updates, the test will fail due to missing search parameters and missing profiles.

Define search parameters

As part of the C4BB IG, you'll need to define three [new search parameters](#) for the `ExplanationOfBenefit` resource. Two of these are tested in the capability statement (type and service-date), and one is needed for `_include` searches (insurer).

- [type](#)
- [service-date](#)
- [insurer](#)

NOTE

In the raw JSON for these search parameters, the name is set to `ExplanationOfBenefit_<SearchParameter Name>`. The Touchstone test is expecting that the name for these will be **type**, **service-date**, and **insurer**.

The rest of the search parameters needed for the C4BB IG are defined by the base specification and are already available in the Azure API for FHIR without any additional updates.

Store profiles

Outside of defining search parameters, the other update you need to make to pass this test is to load the [required profiles](#). There are eight profiles defined within the C4BB IG.

- [C4BB Coverage](#)
- [C4BB ExplanationOfBenefit Inpatient Institutional](#)
- [C4BB ExplanationOfBenefit Outpatient Institutional](#)
- [C4BB ExplanationOfBenefit Pharmacy](#)
- [C4BB ExplanationOfBenefit Professional NonClinician](#)
- [C4BB Organization](#)
- [C4BB Patient](#)
- [C4BB Practitioner](#)

Sample rest file

To assist with creation of these search parameters and profiles, we have a [sample http file](#) that includes all the steps outlined above in a single file. Once you've uploaded all the necessary profiles and search parameters, you can run the capability statement test in Touchstone.

Test Script Execution - /FHIRsandbox/CARIN/CARIN-4-BlueButton/00-Capability/carin-bb-00-Capability-json

[← To Test Execution](#)

Exec Id: 202105241532366041338526
Start Time: 05/24/2021 12:32:36PM
End Time: 05/24/2021 12:32:57PM
Status: Passed W
Duration: 20.858s
Version: 4
Validator: FHIR 4.0.1

Description: Test a single server to verify support for the capabilities interaction 'HTTP GET metadata' and the return of a valid CapabilityStatement resource supporting the CARIN for BB IG Consumer Application Implementation Guide Version: 1.0.0 capabilities using JSON syntax.
Test Setup: FHIRsandbox-CARIN-CARIN-4-BlueButton-00-Capability--All
Executed By:
Organization:
Origin: TouchstoneFHIR
Destination:
Test Script: /FHIRsandbox/CARIN/CARIN-4-BlueButton/00-Capability/carin-bb-00-Capability-json

Interactions					
	100% passed	Pass	Fail	Other	Total
Summary	<div></div>	1	0	0	1
metadata	<div></div>	1	0	0	1

1 tests	1 passes	0 failures	0 skipped	0 running	0 waiting	0 not started	100% successful
---------	----------	------------	-----------	-----------	-----------	---------------	-----------------

[Refresh](#)

Tests

Test Name	Description	Status	Duration
Test: capability-json	Test the HTTP GET metadata capabilities operation with HTTP Header Accept set to JSON format. The expected response content is a valid CapabilityStatement resource supporting the CARIN for BB IG Implementation Guide Version: 1.0.0. Test is asking for JSON format which is required for implementation.	Passed W	12.499s

Profiles

Id	Source	Contents
capabilities-profile	http://hl7.org/fhir/StructureDefinition/CapabilityStatement	XML JSON

Touchstone read test

After testing the capabilities statement, we will test the [read capabilities](#) of the Azure API for FHIR against the C4BB IG. This test is testing conformance against the eight profiles you loaded in the first test. You will need to have resources loaded that conform to the profiles. The best path would be to test against resources that you already have in your database, but we also have an [http file](#) available with sample resources pulled from the examples in the IG that you can use to create the resources and test against.

Test Execution

[Execute Again](#)

Exec Id: 202105211811072957527663
Start Time: 05/21/2021 03:11:07PM
End Time: 05/21/2021 03:11:35PM
Status: Passed W
Duration: 28.017s
Test Scripts: 8

Test Setup: FHIRsandbox-CARIN-CARIN-4-BlueButton-01-Read--All
Executed By:
Organization:
Origin: TouchstoneFHIR
Destination:
Validator: FHIR 4.0.1

8 tests	8 passes	0 failures	0 skipped	0 running	0 waiting	0 not started	100% successful	Refresh
---------	----------	------------	-----------	-----------	-----------	---------------	-----------------	-------------------------

Test Script Execution	Version	Latest	Description	Origin	Destination	Status	Start	End	Duration	Passed	Tests
/FHIRsandbox/CARIN/CARIN-4-BlueButton/01-Read/carin-bb-01-Read-Coverage	2	2	Read for a specific Coverage Resource. Read shall be supported for this resource type. Tests for conformance to the Carin4BB Coverage profile on a returned response for Coverage	TouchstoneFHIR	HLSCD PM - Caitlin May C https://caitlinmay.azurewebsites.net	Passed W	05/21/2021 03:11:07PM	05/21/2021 03:11:14PM	7.408s	1 of 1	<div></div>
/FHIRsandbox/CARIN/CARIN-4-BlueButton/01-Read/carin-bb-01-Read-EOB-Inpatient	1	1	Read for a specific ExplanationOfBenefit - Inpatient Institutional Resource. Read shall be supported for this resource type. Tests for conformance to the Carin4BB ExplanationOfBenefit Inpatient Institutional profile on a returned response for ExplanationOfBenefit	TouchstoneFHIR	HLSCD PM - Caitlin May C https://caitlinmay.azurewebsites.net	Passed W	05/21/2021 03:11:14PM	05/21/2021 03:11:22PM	7.363s	1 of 1	<div></div>
/FHIRsandbox/CARIN/CARIN-4-BlueButton/01-Read/carin-bb-01-Read-EOB-NonClinical	1	1	Read for a specific ExplanationOfBenefit - NonClinical Resource. Read shall be supported for this resource type. Tests for conformance to the Carin4BB ExplanationOfBenefit NonClinical profile on a returned response for ExplanationOfBenefit	TouchstoneFHIR	HLSCD PM - Caitlin May C https://caitlinmay.azurewebsites.net	Passed W	05/21/2021 03:11:22PM	05/21/2021 03:11:26PM	4.126s	1 of 1	<div></div>
/FHIRsandbox/CARIN/CARIN-4-BlueButton/01-Read/carin-bb-01-Read-EOB-Outpatient	1	1	Read for a specific ExplanationOfBenefit - Outpatient Institutional Resource. Read shall be supported for this resource type. Tests for conformance to the Carin4BB ExplanationOfBenefit Outpatient Institutional profile on a returned response for ExplanationOfBenefit	TouchstoneFHIR	HLSCD PM - Caitlin May C https://caitlinmay.azurewebsites.net	Passed W	05/21/2021 03:11:26PM	05/21/2021 03:11:30PM	3.610s	1 of 1	<div></div>
/FHIRsandbox/CARIN/CARIN-4-BlueButton/01-Read/carin-bb-01-Read-EOB-Pharmacy	1	1	Read for a specific ExplanationOfBenefit - Pharmacy Resource. Read shall be supported for this resource type. Tests for conformance to the Carin4BB ExplanationOfBenefit Pharmacy profile on a returned response for ExplanationOfBenefit	TouchstoneFHIR	HLSCD PM - Caitlin May C https://caitlinmay.azurewebsites.net	Passed W	05/21/2021 03:11:30PM	05/21/2021 03:11:32PM	2.112s	1 of 1	<div></div>
/FHIRsandbox/CARIN/CARIN-4-BlueButton/01-Read/carin-bb-01-Read-Organization	2	2	Read for a specific Organization Resource. Read shall be supported for this resource type. Tests for conformance to the Carin4BB Organization profile on a returned response for Organization	TouchstoneFHIR	HLSCD PM - Caitlin May C https://caitlinmay.azurewebsites.net	Passed W	05/21/2021 03:11:32PM	05/21/2021 03:11:33PM	0.826s	1 of 1	<div></div>
/FHIRsandbox/CARIN/CARIN-4-BlueButton/01-Read/carin-bb-01-Read-Patient	2	2	Read for a specific Patient Resource. Read shall be supported for this resource type. Tests for conformance to the Carin4BB Patient profile on a returned response for Patient	TouchstoneFHIR	HLSCD PM - Caitlin May C https://caitlinmay.azurewebsites.net	Passed W	05/21/2021 03:11:33PM	05/21/2021 03:11:34PM	0.896s	1 of 1	<div></div>
/FHIRsandbox/CARIN/CARIN-4-BlueButton/01-Read/carin-bb-01-Read-Practitioner	1	1	Read for a specific Practitioner Resource. Read shall be supported for this resource type. Tests for conformance to the Carin4BB Practitioner profile on a returned response for Practitioner	TouchstoneFHIR	HLSCD PM - Caitlin May C https://caitlinmay.azurewebsites.net	Passed W	05/21/2021 03:11:34PM	05/21/2021 03:11:35PM	1.081s	1 of 1	<div></div>

Touchstone EOB query test

The next test we'll review is the [EOB query test](#). If you've already completed the read test, you have all the data

loaded that you'll need. This test validates that you can search for specific `Patient` and `ExplanationOfBenefit` resources using various parameters.

Test Execution

Execute Again

Exec Id: 202105241621109619460553

Start Time: 05/24/2021 01:21:10PM

End Time: 05/24/2021 01:21:19PM

Status: Passed

Duration: 8.339s

Test Scripts: 6

Test Setup: FHIRSandbox-CARIN-CARIN-4-BlueButton-02-EOBQuery--All

Executed By:

Organization:

Origin: TouchstoneFHIR

Destination:

Validator: FHIR 4.0.1

6 tests

6 passes

0 failures

0 skipped

0 running

0 waiting

0 not started

100% successful

Refresh

Test Script Execution	Version	Latest	Description	Origin	Destination	Status	Start	End	Duration	Passed	Tests
/FHIRSandbox/CARIN/CARIN-4-BlueButton/02-EOBQuery/carin-bb-02-Query-EOB_LastUpdated	2	2	Tests for conformance to the Carin4BB IG by querying an EOB by _lastUpdated. Any EOB type (Inpatient, Outpatient, Pharmacy, or NonClinical) can be utilized in this test.	TouchstoneFHIR	HLSCD PM - Microsoft FHIR Server https://cattlin-touchstone.azurewebsites.net	Passed	05/24/2021 01:21:11PM	05/24/2021 01:21:13PM	2.303s	1 of 1	<div></div>
/FHIRSandbox/CARIN/CARIN-4-BlueButton/02-EOBQuery/carin-bb-02-Query-EOB_service-date	2	2	Tests for conformance to the Carin4BB IG by querying an EOB by service-date. Any EOB type (Inpatient, Outpatient, Pharmacy, or NonClinical) can be utilized in this test.	TouchstoneFHIR	HLSCD PM - Microsoft FHIR Server https://cattlin-touchstone.azurewebsites.net	Passed	05/24/2021 01:21:13PM	05/24/2021 01:21:14PM	0.969s	1 of 1	<div></div>
/FHIRSandbox/CARIN/CARIN-4-BlueButton/02-EOBQuery/carin-bb-02-Query-EOB_type	2	2	Tests for conformance to the Carin4BB IG by querying an EOB by type. Any EOB type (Inpatient, Outpatient, Pharmacy, or NonClinical) can be utilized in this test.	TouchstoneFHIR	HLSCD PM - Microsoft FHIR Server https://cattlin-touchstone.azurewebsites.net	Passed	05/24/2021 01:21:14PM	05/24/2021 01:21:15PM	1.066s	1 of 1	<div></div>
/FHIRSandbox/CARIN/CARIN-4-BlueButton/02-EOBQuery/carin-bb-02-Query-EOBbyIdentifier	2	2	Tests for conformance to the Carin4BB IG by querying an EOB by identifier. Any EOB type (Inpatient, Outpatient, Pharmacy, or NonClinical) can be utilized in this test.	TouchstoneFHIR	HLSCD PM - Microsoft FHIR Server https://cattlin-touchstone.azurewebsites.net	Passed	05/24/2021 01:21:15PM	05/24/2021 01:21:16PM	1.155s	1 of 1	<div></div>
/FHIRSandbox/CARIN/CARIN-4-BlueButton/02-EOBQuery/carin-bb-02-Query-EOBbyPatient	2	2	Tests for conformance to the Carin4BB IG by querying an EOB by Patient. Any EOB type (Inpatient, Outpatient, Pharmacy, or NonClinical) can be utilized in this test.	TouchstoneFHIR	HLSCD PM - Microsoft FHIR Server https://cattlin-touchstone.azurewebsites.net	Passed	05/24/2021 01:21:16PM	05/24/2021 01:21:18PM	1.244s	1 of 1	<div></div>
/FHIRSandbox/CARIN/CARIN-4-BlueButton/02-EOBQuery/carin-bb-02-Query-EOBbyId	2	2	Tests for conformance to the Carin4BB IG by querying an EOB by _id. Any EOB type (Inpatient, Outpatient, Pharmacy, or NonClinical) can be utilized in this test.	TouchstoneFHIR	HLSCD PM - Microsoft FHIR Server https://cattlin-touchstone.azurewebsites.net	Passed	05/24/2021 01:21:18PM	05/24/2021 01:21:19PM	1.074s	1 of 1	<div></div>

Touchstone error handling test

The final test we'll walk through is testing `error handling`. The only step you need to do is delete an `ExplanationOfBenefit` resource from your database and use the ID of the deleted `ExplanationOfBenefit` resource in the test.

Test Execution

Execute Again

Exec Id: 202105211914115453588480

Start Time: 05/21/2021 04:14:11PM

End Time: 05/21/2021 04:14:23PM

Status: Passed

Duration: 11.459s

Test Scripts: 3

Test Setup: FHIRSandbox-CARIN-CARIN-4-BlueButton-99-ErrorHandling--All

Executed By:

Organization:

Origin: TouchstoneFHIR

Destination:

Validator: FHIR 4.0.1

3 tests

3 passes

0 failures

0 skipped

0 running

0 waiting

0 not started

100% successful

Refresh

Test Script Execution	Version	Latest	Description	Origin	Destination	Status	Start	End	Duration	Passed	Tests
/FHIRSandbox/CARIN/CARIN-4-BlueButton/99-ErrorHandling/carin-bb-99-DeletedResource	2	2	CARIN for BlueButton 99 Deleted Resource - Test the deleted resource error by sending in a read request for known deleted EOB resource and expecting a 410 HTTP response.	TouchstoneFHIR	HLSCD PM - Cattlin May https://cattlinmay.azurewebsites.net	Passed	05/21/2021 04:14:11PM	05/21/2021 04:14:15PM	3.482s	1 of 1	<div></div>
/FHIRSandbox/CARIN/CARIN-4-BlueButton/99-ErrorHandling/carin-bb-99-Invalid-Parameters	1	1	CARIN for BlueButton 99 Invalid Parameters - Test the invalid paramters error by sending in an invalid search parameter and expecting a 400 HTTP response.	TouchstoneFHIR	HLSCD PM - Cattlin May https://cattlinmay.azurewebsites.net	Passed	05/21/2021 04:14:15PM	05/21/2021 04:14:18PM	3.370s	1 of 1	<div></div>
/FHIRSandbox/CARIN/CARIN-4-BlueButton/99-ErrorHandling/carin-bb-99-UnknownResource	2	2	CARIN for BlueButton 99 Unknown Resource - Test the unknown resource error by sending in a query for an unknown EOB resource and expecting a 404 HTTP response.	TouchstoneFHIR	HLSCD PM - Cattlin May https://cattlinmay.azurewebsites.net	Passed	05/21/2021 04:14:18PM	05/21/2021 04:14:22PM	4.451s	1 of 1	<div></div>

Next steps

In this tutorial, we walked through how to pass the CARIN IG for Blue Button tests in Touchstone. Next, you can review how to test the Da Vinci formulary tests.

[DaVinci Drug Formulary](#)

Da Vinci Drug Formulary

6/8/2021 • 2 minutes to read • [Edit Online](#)

In this tutorial, we'll walk through setting up the Azure API for FHIR to pass the [Touchstone](#) tests for the [Da Vinci Payer Data Exchange US Drug Formulary Implementation Guide](#).

Touchstone capability statement

The first test that we'll focus on is testing the Azure API for FHIR against the [Da Vinci Drug Formulary capability statement](#). If you run this test without any updates, the test will fail due to missing search parameters and missing profiles.

Define search parameters

As part of the Da Vinci Drug Formulary IG, you'll need to define three [new search parameters](#) for the FormularyDrug resource. All three of these are tested in the capability statement.

- [DrugTier](#)
- [DrugPlan](#)
- [DrugName](#)

The rest of the search parameters needed for the Da Vinci Drug Formulary IG are defined by the base specification and are already available in the Azure API for FHIR without any more updates.

Store profiles

Outside of defining search parameters, the only other update you need to make to pass this test is to load the [required profiles](#). There are two profiles used as part of the Da Vinci Drug Formulary IG.

- [Formulary Drug](#)
- [Formulary Coverage Plan](#)

Sample rest file

To assist with creation of these search parameters and profiles, we have the [Da Vinci Formulary](#) sample HTTP file on the open-source site that includes all the steps outlined above in a single file. Once you've uploaded all the necessary profiles and search parameters, you can run the capability statement test in Touchstone. You should get a successful run:

Da Vinci PDex

Da Vinci PDex

6/8/2021 • 2 minutes to read • [Edit Online](#)

In this tutorial, we'll walk through setting up the Azure API for FHIR to pass the [Touchstone](#) tests for the [Da Vinci Payer Data Exchange Implementation Guide](#) (PDex IG).

NOTE

For all these tests, we'll run them against the JSON tests. The Azure API for FHIR supports both JSON and XML, but it doesn't have separate endpoints to access JSON or XML. Because of this, all the XML tests will fail. If you want to view the capability statement in XML you simply pass the `_format` parameter: ``GET {fhirurl}/metadata?_format=xml``

Touchstone capability statement

The first set of tests that we'll focus on is testing the Azure API for FHIR against the PDex IG capability statement. This includes three tests:

- The first test validates the basic capability statement against the IG requirements and will pass without any updates.
- The second test validates all the profiles have been added for US Core. This test will pass without updates but will include a bunch of warnings. To have these warnings removed, you need to [load the US Core profiles](#). We've created a [sample HTTP file](#) that walks through creating all the profiles. You can also get the [profiles](#) from the HL7 site directly, which will have the most current versions.
- The third test validates that the [\\$patient-everything operation](#) is supported. Right now, this test will fail. The operation will be available in mid-June 2021 in the Azure API for FHIR and is available now in the open-source FHIR server on Cosmos DB. However, it is missing from the capability statement, so this test will fail until we release a fix to bug [1989](#).

Test Script Execution - /FHIRSandbox/DaVinci/FHIR4-0-1-Test/PDEX/PayerExchange/00-Capabilitystatement/pdex-4-0-1-00-capability-json

[← To Test Execution](#)

Exec Id: 202106011740124518353949

Start Time: 06/01/2021 02:40:12PM

End Time: 06/01/2021 02:40:39PM

Status: Failed

Duration: 26.627s

Version: 1

Validator: FHIR 4.0.1

Description: Da Vinci - PDex - FHIR R4-0-1 - Scenario 01 - Capability - test a single server to verify support for the capabilities interaction 'HTTP GET metadata' and the return of a valid CapabilityStatement for the PDEX IG using JSON syntax.

Test Setup: FHIRSandbox--pdex-4-0-1-00-capability-json

Executed By:

Organization:

Origin: TouchstoneFHIR

Destination:

Test Script: /FHIRSandbox/DaVinci/FHIR4-0-1-Test/PDEX/PayerExchange/00-Capabilitystatement/pdex-4-0-1-00-capability-json

Interactions

	66% passed	Pass	Fail	Other	Total
Summary		2	1	0	3
metadata		2	1	0	3



[Refresh](#)

Tests

Test Name	Description	Status	Duration
Test: CapabilityMetadata.JSON	Test the HTTP GET metadata capabilities operation with HTTP Header Accept set to JSON format and no request URL parameters defined. The expected response content is the found CapabilityStatement resource in JSON format.	Passed	5.641s
Test: CapabilityMetadata.JSON-USCore	Test the HTTP GET metadata capabilities operation with HTTP Header Accept set to JSON format and no request URL parameters defined. The expected response content is the found CapabilityStatement resource in JSON format that asserts support for the US Core profiles.	Passed	14.432s
Test: CapabilityMetadata.JSON-Operations	Test the HTTP GET metadata capabilities operation with HTTP Header Accept set to JSON format and no request URL parameters defined. The expected response content is the found CapabilityStatement resource in JSON format and asserting support for required PDEX operation, \$patient-everything.	Failed	3.257s

Touchstone \$member-match test

The [second test](#) in the Payer Data Exchange section tests the existence of the [\\$member-match operation](#). You can read more about the \$member-match operation in our [\\$member-match operation overview](#).

In this test, you'll need to load some sample data for the test to pass. We have a rest file [here](#) with the patient and coverage linked that you will need for the test. Once this data is loaded, you'll be able to successfully pass this test. If the data is not loaded, you'll receive a 422 response due to not finding an exact match.

Test Script Execution - /FHIR Sandbox/DaVinci/FHIR4-0-1-Test/PDEX/PayerExchange/01-Member-Match/01-Member-Id-Confirmation-json [← To Test Execution](#)

Exec ID: 202106011808229286490948
Start Time: 06/01/2021 03:08:22PM
End Time: 06/01/2021 03:08:32PM
Status: Passed
Duration: 9.374s
Version: 1
Validator: FHIR 4.0.1

Description: Connectathon 25 - DaVinci HREx/CDex/PCDE - Clinical Data, Payer Coverage Decision & Health Record Exchange - Use Case 01 - Member Id Confirmation
Test Setup: FHIR Sandbox-01-Member-Id-Confirmation-json
Executed By:
Organization:
Origin: TouchstoneFHIR
Destination:
Test Script: /FHIR Sandbox/DaVinci/FHIR4-0-1-Test/PDEX/PayerExchange/01-Member-Match/01-Member-Id-Confirmation-json

Interactions

	100% passed	Pass	Fail	Other	Total
Summary	<div></div>	1	0	0	1
\$match Patient	<div></div>	1	0	0	1

1 tests

1 passes

0 failures

0 skipped

0 running

0 waiting

0 not started

100% successful

[Refresh](#)

Tests

Test Name	Description	Status	Duration
Test: 01-Member-Id-Confirmation-json	Data Consumer Payer invokes the \$member-match operation on the Data Source Payer	Passed	0.539s

Touchstone patient by reference

The next tests we'll review is the [patient by reference](#) tests. This set of tests validate that you can find a patient based on various search criteria. The best way to test the patient by reference will be to test against your own data, but we have uploaded a [sample resource file](#) that you can load to use as well.

Test Execution [Execute Again](#)

Exec ID: 202106012023442467881762
Start Time: 06/01/2021 05:23:44PM
End Time: 06/01/2021 05:24:20PM
Status: Passed
Duration: 36.484s
Test Scripts: 3

Test Setup: FHIR Sandbox-DaVinci-FHIR4-0-1-Test-PDEX-PayerExchange-02-PatientByReference-tests
Executed By:
Organization:
Origin: TouchstoneFHIR
Destination:
Validator: FHIR 4.0.1

3 tests

3 passes

0 failures

0 skipped

0 running

0 waiting

0 not started

100% successful

[Refresh](#)

Test Script Execution	Version	Latest	Description	Origin	Destination	Status	Start	End	Duration	Passed	Tests
/FHIR Sandbox/DaVinci/FHIR4-0-1-Test/PDEX/PayerExchange/02-PatientByReference/dv-pdex-r4-01-patient-json	1	1	Da Vinci - PDEX - FHIR R4 - Patient Query - Search a known Patient by the Health_Plan_Location_ID as JSON formatted data.	TouchstoneFHIR	HLSCD PM - Caitlin June 🔗 https://caitlinjune.azurewebsites.net	Passed	06/01/2021 05:23:44PM	06/01/2021 05:23:54PM	10.301s	1 of 1	<div></div>
/FHIR Sandbox/DaVinci/FHIR4-0-1-Test/PDEX/PayerExchange/02-PatientByReference/dv-pdex-r4-02-encounter-json	1	1	Da Vinci - PDEX - FHIR R4 - Encounter Query - Search Encounters for a known Patient updated since a known date and excluding my known location as JSON formatted data.	TouchstoneFHIR	HLSCD PM - Caitlin June 🔗 https://caitlinjune.azurewebsites.net	Passed	06/01/2021 05:23:54PM	06/01/2021 05:24:07PM	12.863s	1 of 1	<div></div>
/FHIR Sandbox/DaVinci/FHIR4-0-1-Test/PDEX/PayerExchange/02-PatientByReference/dv-pdex-r4-03-coverage-json	1	1	Da Vinci - PDEX - FHIR R4 - Scenario 02 Payer Exchange - Query - Search all Coverages for a known Subscriber since a known date as JSON formatted data.	TouchstoneFHIR	HLSCD PM - Caitlin June 🔗 https://caitlinjune.azurewebsites.net	Passed	06/01/2021 05:24:07PM	06/01/2021 05:24:20PM	13.148s	1 of 1	<div></div>

Touchstone patient/\$everything test

The final test we'll walk through is testing patient-everything. For this test, you'll need to load a patient, and then you'll use that patient's ID to test that you can use the \$everything operation to pull all data related to the patient.

Next steps

In this tutorial, we walked through how to pass the Payer Exchange tests in Touchstone. Next, you can learn about all the Azure API for FHIR features.

[Supported features](#)

Register the Azure Active Directory apps for Azure API for FHIR

3/11/2021 • 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

3/11/2021 • 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 on the [GitHub repo](#) to register a resource application.

Next steps

In this article, you've learned how to register a resource application in Azure Active Directory. Next, register your confidential client application.

[Register Confidential Client Application](#)

Register a confidential client application in Azure Active Directory

4/9/2021 • 2 minutes to read • [Edit Online](#)

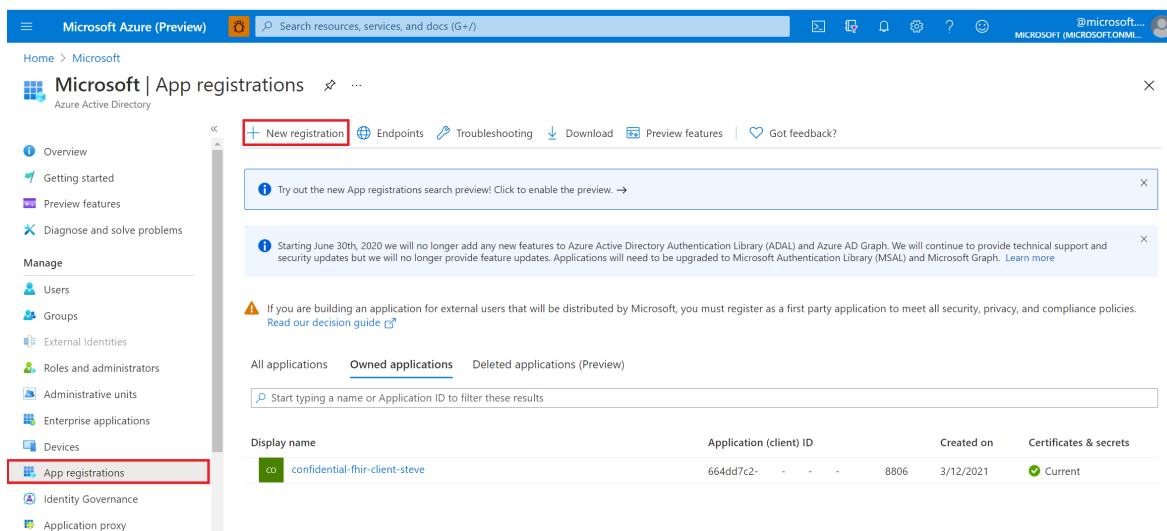
In this tutorial, you'll learn how to register a confidential client application in Azure Active Directory (Azure AD).

A client application registration is an Azure AD 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 client application, refer to the steps below.

Register a new application

1. In the [Azure portal](#), select **Azure Active Directory**.
2. Select **App registrations**.



3. Select **New registration**.
4. Give the application a user-facing display name.
5. For **Supported account types**, select who can use the application or access the API.
6. (Optional) Provide a **Redirect URI**. These details can be changed later, but if you know the reply URL of your application, enter it now.

Home > Microsoft > Register an application

Register an application

Name
The user-facing display name for this application (this can be changed later).
confidential-fhir-client

Supported account types
Who can use this application or access this API?
☒ Accounts in this organizational directory only (Microsoft only - Single tenant)
☐ Accounts in any organizational directory (Any Azure AD directory - Multitenant)
☐ Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)
☐ Personal Microsoft accounts only

Redirect URI (optional)
We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.
Web | https://my-confidential-client/auth/

By proceeding, you agree to the [Microsoft Platform Policies](#)

Register

7. Select Register.

API permissions

Now that you've registered your application, you must select which API permissions this application should request on behalf of users.

1. Select API permissions.

Home > Microsoft > confidential-fhir-client

confidential-fhir-client | API permissions

Search (Ctrl+J) | Refresh | Got feedback?

Manage
 Branding
 Authentication
 Certificates & secrets
 Token configuration
API permissions
 Expose an API
 App roles
 Owners
 Roles and administrators | Preview
 Manifest

Support + Troubleshooting
 Troubleshooting
 New support request

Configured permissions
 Applications are authorized to call APIs when they are granted permissions by users/admins as part of the consent process. The list of configured permissions should include all the permissions the application needs. [Learn more about permissions and consent](#)

Add a permission | Grant admin consent for Microsoft

API / Permissions name	Type	Description	Admin consent req...	Status
Microsoft Graph (1)				...
User.Read	Delegated	Sign in and read user profile	No	...

To view and manage permissions and user consent, try [Enterprise applications](#).

2. Select Add a permission.

If you're using the Azure API for FHIR, you'll add a permission to the Azure Healthcare APIs by searching for **Azure Healthcare API** under **APIs my organization uses**. The search result for Azure Healthcare API will only return if you've already [deployed the Azure API for FHIR](#).

If you're referencing a different resource application, select your [FHIR API Resource Application Registration](#) that you created previously under **My APIs**.

Request API permissions



Select an API

Microsoft APIs

APIs my organization uses

My APIs

Apps in your directory that expose APIs are shown below

Name	Application (client) ID
Azure Healthcare APIs	4f6778d8- - b9495

3. Select scopes (permissions) that the confidential client application will ask for on behalf of a user. Select **user_impersonation**, and then select **Add permissions**.

[< All APIs](#)

AH Azure Healthcare APIs
https://*.azurehealthcareapis.com

What type of permissions does your application require?

Delegated permissions

Your application needs to access the API as the signed-in user.

Application permissions

Your application runs as a background service or daemon without a signed-in user.

Select permissions

[expand all](#)

i The "Admin consent required" column shows the default value for an organization. However, user consent can be customized per permission, user, or app. This column may not reflect the value in your organization, or in organizations where this app will be used. [Learn more](#)

Permission	Admin consent required
Permissions (1)	
<input checked="" type="checkbox"/> user_impersonation ⓘ Access Azure Healthcare APIs	No

Add permissions

Discard

Application secret

1. Select **Certificates & secrets**, and then select **New client secret**.

Microsoft Azure (Preview) Search resources, services, and docs (G+)

Home > Microsoft > confidential-fhir-client

confidential-fhir-client | Certificates & secrets

Search (Ctrl+/) Got feedback?

- Overview
- Quickstart
- Integration assistant
- Manage
 - Branding
 - Authentication
 - Certificates & secrets**
 - Token configuration
 - API permissions
 - Expose an API
 - App roles
 - Owners
 - Roles and administrators | Preview
 - Manifest
- Support + Troubleshooting
 - Troubleshooting
 - New support request

Certificates

Credentials enable confidential applications to identify themselves to the authentication service when receiving tokens at a web addressable location (using an HTTPS scheme). For a higher level of assurance, we recommend using a certificate (instead of a client secret) as a credential.

Certificates can be used as secrets to prove the application's identity when requesting a token. Also can be referred to as public keys.

Upload certificate

Thumbprint	Start date	Expires	ID
No certificates have been added for this application.			

Client secrets

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

+ New client secret

Description	Expires	Value	ID
No client secrets have been created for this application.			

2. Enter a **Description** for the client secret. Select the **Expires** drop-down menu to choose an expiration time frame, and then click **Add**.

Add a client secret



Description

Enter a description for this client secret

Expires

Recommended: 6 months



Recommended: 6 months

3 months

12 months

18 months

24 months

Custom

Add

Cancel

- After the client secret string is created, copy its **Value** and **ID**, and store them in a secure location of your choice.

Client secrets

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

+ New client secret

Description	Expires	Value	ID
my-fhir-client	3/16/2022	Fw_*****	07aa2806-0090-4973-a06e-cf0d431e34af  

NOTE

The client secret string is visible only once in the Azure portal. When you navigate away from the Certificates & secrets web page and then return back to it, the Value string becomes masked. It's important to make a copy your client secret string immediately after it is generated. If you don't have a backup copy of your client secret, you must repeat the above steps to regenerate it.

Next steps

In this article, you were guided through the steps of how to register a confidential client application in the Azure AD. You were also guided through the steps of how to add API permissions to the Azure Healthcare API. Lastly, you were shown how to create an application secret. Furthermore, you can learn how to access your FHIR server using Postman.

[Access Azure API for FHIR with Postman](#)

Register a public client application in Azure Active Directory

3/11/2021 • 2 minutes to read • [Edit Online](#)

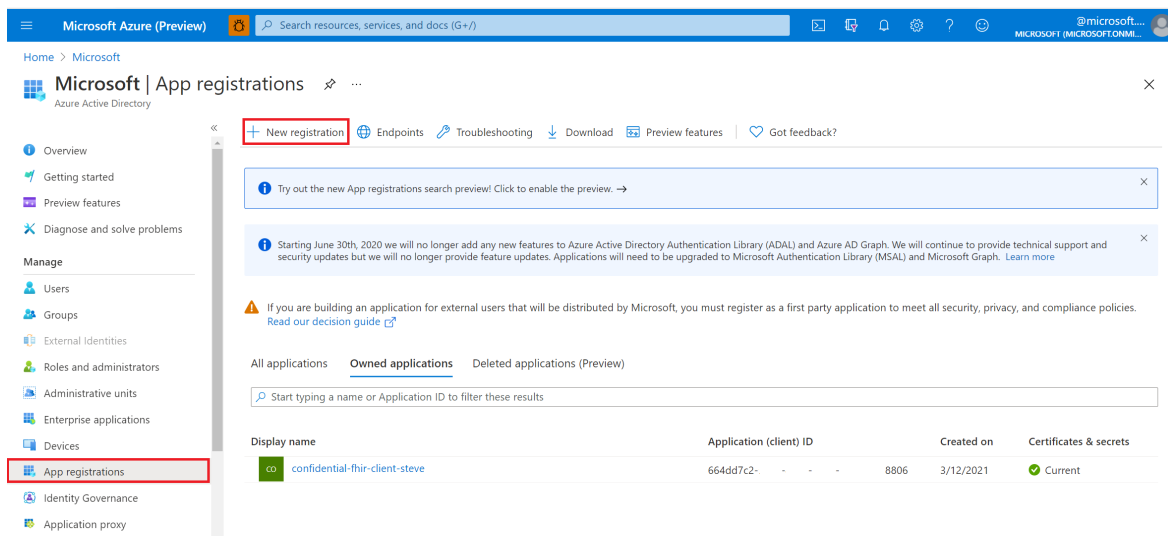
In this article, you'll learn how to register a public application in Azure Active Directory.

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.

The quickstart provides general information about how to [register an application with the Microsoft identity platform](#).

App registrations in Azure portal

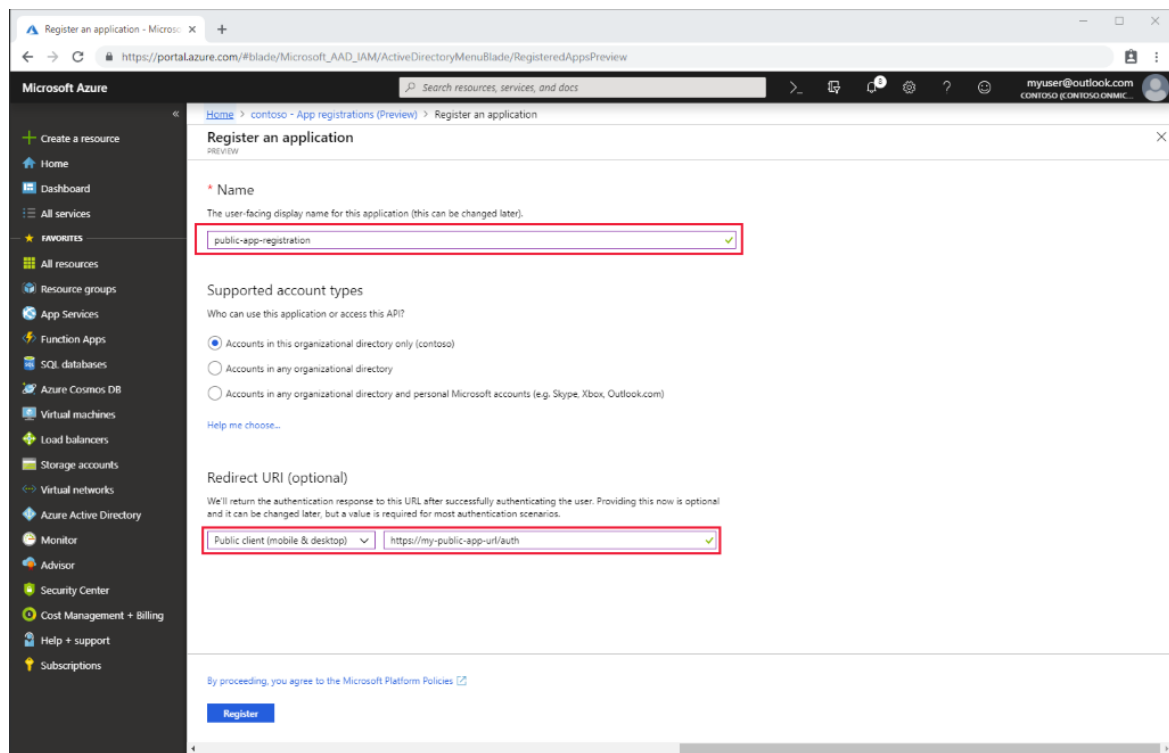
1. In the [Azure portal](#), on the left navigation panel, click **Azure Active Directory**.
2. In the **Azure Active Directory** blade, click **App registrations**:



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.



To configure your [desktop](#), [mobile](#) or [single-page](#) application as public application:

1. In the [Azure portal](#), in **App registrations**, select your app, and then select **Authentication**.
2. Select **Advanced settings** > **Default client type**. For **Treat application as a public client**, select **Yes**.
3. For a single-page application, select **Access tokens** and **ID tokens** to enable implicit flow.
 - If your application signs in users, select **ID tokens**.
 - If your application also needs to call a protected web API, select **Access tokens**.

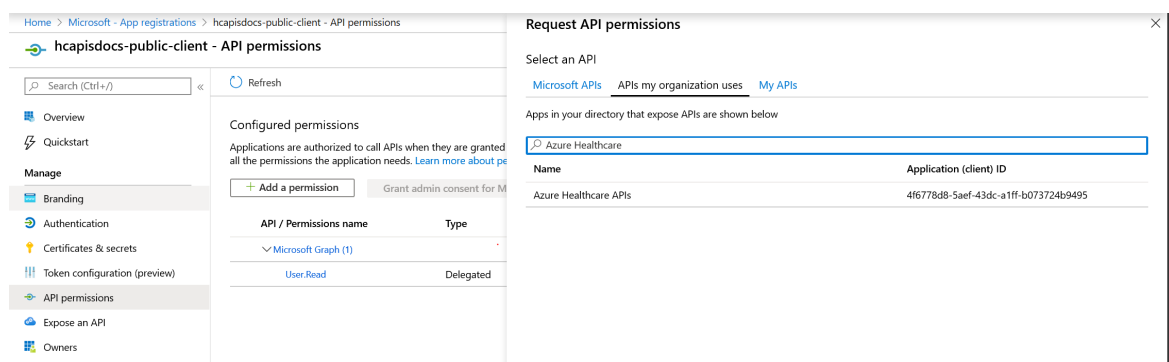
API permissions

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**. You will only be able to find this if you have already [deployed the Azure API for FHIR](#).

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:

Request API permissions

[← All APIs](#)

AH Azure Healthcare APIs
https://*.azurehealthcareapis.com

What type of permissions does your application require?

Delegated permissions
Your application needs to access the API as the signed-in user.

Application permissions
Your application runs as a background service or daemon without a signed-in user.

Select permissions [expand all](#)

Permission	Admin Consent Required
<input checked="" type="checkbox"/> user_impersonation Access Azure Healthcare APIs ⓘ	-

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

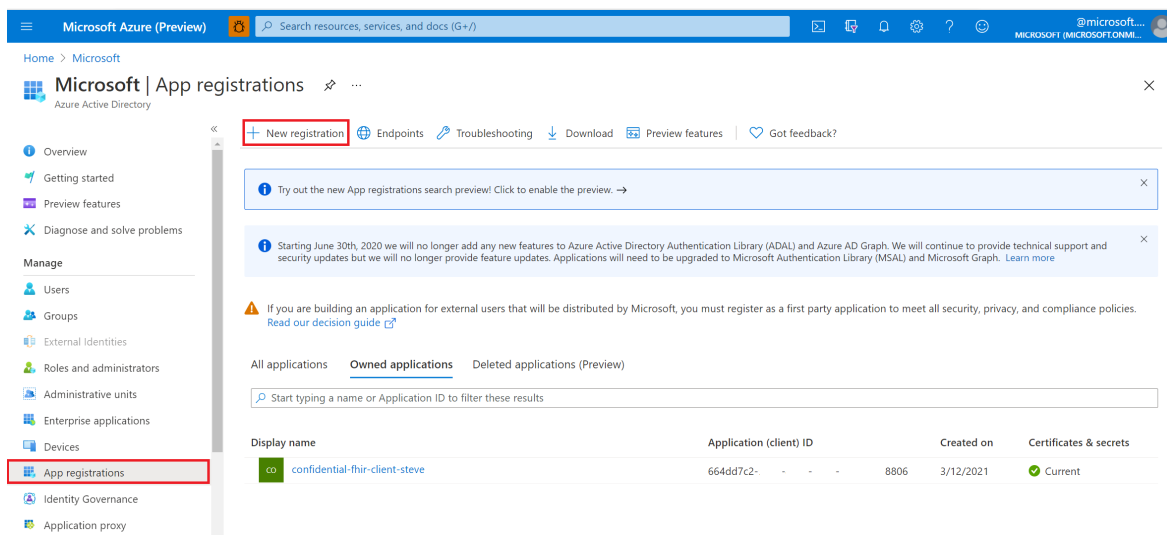
3/11/2021 • 2 minutes to read • [Edit Online](#)

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 these steps to create a new service client.

App registrations in Azure portal

1. In the [Azure portal](#), navigate to **Azure Active Directory**.
2. Select **App registrations**.



The screenshot shows the Azure portal interface for App registrations. The left sidebar contains a navigation menu with 'App registrations' highlighted. The main content area shows the 'New registration' button highlighted with a red box. Below this, there are several informational messages and a table of owned applications.

Display name	Application (client) ID	Created on	Certificates & secrets
confidential-fhir-client-steve	664dd7c2- - - - 8806	3/12/2021	Current

3. Select **New registration**.
4. Give the service client a display name. Service client applications typically do not use a reply URL.

Register an application

* Name

The user-facing display name for this application (this can be changed later).

FHIR-Service-Client

Supported account types

Who can use this application or access this API?

- ☒ Accounts in this organizational directory only (Default Directory only - Single tenant)
- ☐ Accounts in any organizational directory (Any Azure AD directory - Multitenant)
- ☐ Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)
- ☐ Personal Microsoft accounts only

[Help me choose...](#)

Redirect URI (optional)

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Web

e.g. https://myapp.com/auth

5. Select Register.

API permissions

Now that you have registered your application, you'll need to select which API permissions this application should be able to request on behalf of users:

1. Select **API permissions**.
2. Select **Add a permission**.

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**.

If you are referencing a different Resource Application, select your [FHIR API Resource Application Registration](#) that you created previously under **My APIs**.

Request API permissions

×

Select an API

Microsoft APIs

APIs my organization uses

My APIs

Apps in your directory that expose APIs are shown below

🔍 Azure Healthcare AP

Name

Application (client) ID

Azure Healthcare APIs

4f6778d8-5aef-43dc-a1ff-b073724b9495

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

< All APIs

AH

Azure Healthcare APIs
https://*.azurehealthcareapis.com

What type of permissions does your application require?

Delegated permissions
Your application needs to access the API as the signed-in user.

Application permissions
Your application runs as a background service or daemon without a signed-in user.

Select permissions

expand all

🔍 Start typing a reply url to filter these results	
Permission	Admin consent required
▼ Permissions (1)	
<input checked="" type="checkbox"/> user_impersonation ⓘ Access Azure Healthcare APIs	-

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

Search (Ctrl+/) Refresh Got feedback?

Overview

Quickstart

Integration assistant | Preview

Manage

Branding

Authentication

Certificates & secrets

Token configuration

API permissions

Expose an API

Owners

Roles and administrators | Preview

You are editing permission(s) to your application, users will have to consent even if they've already done so previously.

Configured permissions

Applications are authorized to call APIs when they are granted permissions by users/admins as part of the consent process. The list of configured permissions should include all the permissions the application needs. [Learn more about permissions and consent](#)

+ Add a permission

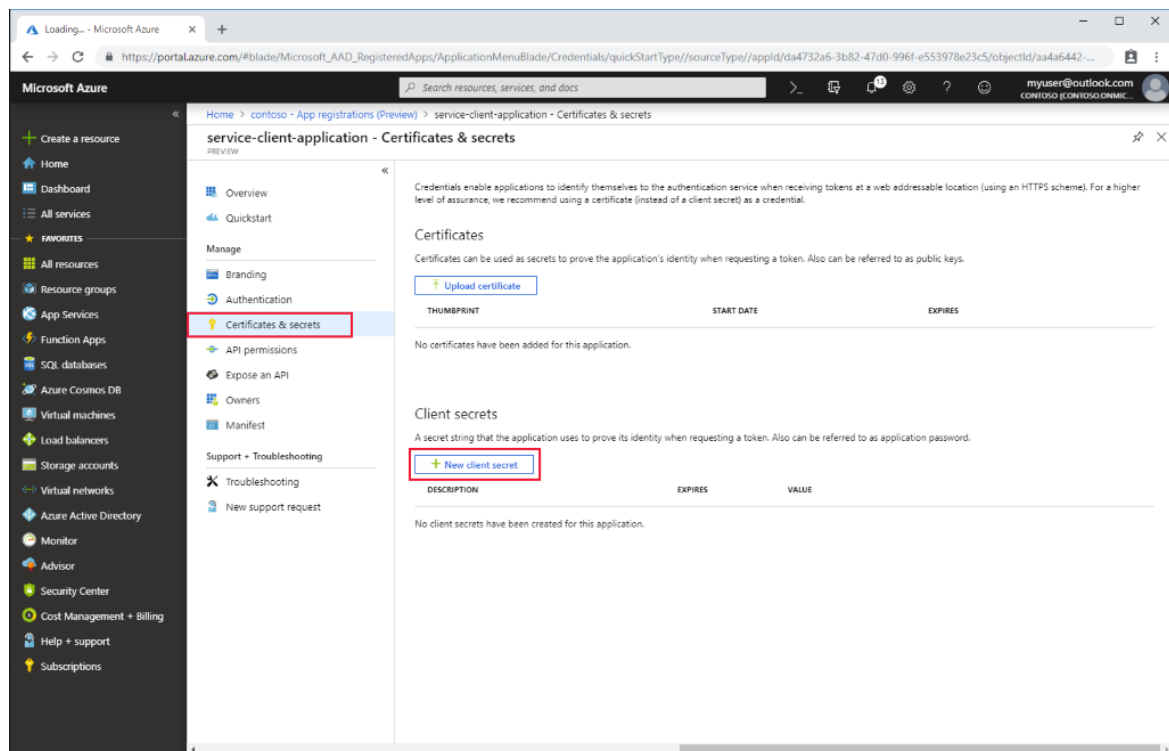
Grant admin consent for Default Directory

API / Permissions name	Type	Description	Admin consent req...	Status
Azure Healthcare APIs (1)				
user_impersonation	Delegated	Access Azure Healthcare APIs	-	...
Microsoft Graph (1)				
User.Read	Delegated	Sign in and read user profile	-	...

Application secret

The service client needs a secret (password) to obtain a token.

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



3. Provide a description and duration of the secret (either 1 year, 2 years or never).
4. Once the secret has been generated, it will only be displayed once in the portal. Make a note of it and store it securely.

Next steps

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

[Access Azure API for FHIR with Postman](#)

Additional settings for Azure API for FHIR

3/11/2021 • 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](#).

Access control

The Azure API for FHIR will only allow authorized users to access the FHIR API. You can configure authorized users through two different mechanisms. The primary and recommended way to configure access control is using [Azure role-based access control \(Azure RBAC\)](#), which is accessible through the **Access control (IAM)** blade. Azure RBAC only works if you want to secure data plane access using the Azure Active Directory tenant associated with your subscription. If you wish to use a different tenant, the Azure API for FHIR offers a local FHIR data plane access control mechanism. The configuration options are not as rich when using the local RBAC mechanism. For details, choose one of the following options:

- [Azure RBAC for FHIR data plane](#). This is the preferred option when you are using the Azure Active Directory tenant associated with your subscription.
- [Local FHIR data plane access control](#). Use this option only when you need to use an external Azure Active Directory tenant for data plane access control.

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 Azure RBAC for FHIR

3/11/2021 • 2 minutes to read • [Edit Online](#)

In this article, you will learn how to use [Azure role-based access control \(Azure RBAC\)](#) to assign access to the Azure API for FHIR data plane. Azure RBAC is the preferred methods for assigning data plane access when data plane users are managed in the Azure Active Directory tenant associated with your Azure subscription. If you are using an external Azure Active Directory tenant, refer to the [local RBAC assignment reference](#).

Confirm Azure RBAC mode

To use Azure RBAC, your Azure API for FHIR must be configured to use your Azure subscription tenant for data plane and there should be no assigned identity object IDs. You can verify your settings by inspecting the **Authentication** blade of your Azure API for FHIR:

The screenshot shows the 'Authentication' blade in the Azure portal. The left sidebar contains navigation links: Overview, Activity log, Access control (IAM), Tags, Settings (with 'Authentication' highlighted), CORS, Database, Integration, Identity, Locks, Export template, Monitoring (Metrics, Diagnostic settings, Logs), and Support + troubleshooting (New support request). The main content area has a title bar with 'Save', 'Discard', and 'Refresh' buttons. Below the title bar, instructions state: 'View and configure authentication settings; specify Azure AD object ID (Users or Apps) that should be allowed to access this Azure API for FHIR. Authority must be registered to Azure AD and in the following format: https://(Azure-AD-endpoint)/(tenant-id). Examples: https://login.microsoftonline.com/contoso.onmicrosoft.com, https://login.microsoftonline.com/abfde7b2-df0f-47e6-aabf-2462b07508dc. Audience must be a URI or GUID secured by Azure AD. Please refer to https://docs.microsoft.com/azure/healthcare-apis/register-resource-azure-ad-client-app for details.'

The configuration fields are as follows:

- Authority ***: A text box containing 'https://login.microsoftonline.com/ <tenant id>' with a green checkmark icon on the right.
- Audience ***: A text box containing 'https://documentdemo.azurehealthcareapis.com' with a green checkmark icon on the right.
- Allowed object IDs**: A section with a header 'Allowed object IDs ⓘ' and a large, disabled text box. Below the box, text reads: 'Use Azure Access Control (IAM) to grant access your FHIR service when using the subscription tenant for data plane RBAC. [Learn more.](#)'
- SMART on FHIR proxy ⓘ**: A checkbox that is currently unchecked.

The **Authority** should be set to the Azure Active directory tenant associated with your subscription and there should be no GUIDs in the box labeled **Allowed object IDs**. You will also notice that the box is disabled and a label indicates that Azure RBAC should be used to assign data plane roles.

Assign roles

To grant users, service principals or groups access to the FHIR data plane, click **Access control (IAM)**, then click **Role assignments** and click **+ Add**:

Search (Ctrl+/)

Overview

Activity log

Access control (IAM)

Tags

Settings

Authentication

CORS

Database

Integration

Identity

Locks

Export template

Monitoring

Metrics

Diagnostic settings

Logs

Support + troubleshooting

New support request

+ Add

Edit columns

Refresh

Remove

Got feedback?

Check access

Role assignments

Deny assignments

Classic administrators

Roles

Manage access to Azure resources for users, groups, service principals and managed identities at this scope by creating role assignments. [Learn more](#)

Number of role assignments for this subscription

5202000

Name

Type

Role

Scope

Group by

Role

0 items

Name	Type	Role	Scope
No user assignments exist			

In the **Role** selection, search for one of the built-in roles for the FHIR data plane:

Add role assignment

Role ⓘ

FHIR Data Reader ⓘ

Assign access to ⓘ

Azure AD user, group, or service principal

Select ⓘ

fhir-clie



fhir-client

Selected members:



fhir-client

Remove

Save

Discard

You can choose between:

- FHIR Data Reader: Can read (and search) FHIR data.
- FHIR Data Writer: Can read, write, and soft delete FHIR data.
- FHIR Data Exporter: Can read and export (`$export` operator) data.
- FHIR Data Contributor: Can perform all data plane operations.

If these roles are not sufficient for your need, you can also [create custom roles](#).

In the **Select** box, search for a user, service principal, or group that you wish to assign the role to.

Caching behavior

The Azure API for FHIR will cache decisions for up to 5 minutes. If you grant a user access to the FHIR server by adding them to the list of allowed object IDs, or you remove them from the list, you should expect it to take up to five minutes for changes in permissions to propagate.

Next steps

In this article, you learned how to assign Azure roles for the FHIR data plane. To learn about additional settings for the Azure API for FHIR:

Configure local RBAC for FHIR

3/11/2021 • 2 minutes to read • [Edit Online](#)

This article explains how to configure the Azure API for FHIR to use an external, secondary Azure Active Directory tenant for managing data plane access. Use this mode only if it is not possible for you to use the Azure Active Directory tenant associated with your subscription.

NOTE

If your FHIR service data plane is configured to use your primary Azure Active Directory tenant associated with your subscription, [use Azure RBAC to assign data plane roles](#).

Add service principal

Local RBAC allows you to use an external Azure Active Directory tenant with your FHIR server. In order to allow the local RBAC system to check group memberships in this tenant, the Azure API for FHIR must have a service principal in the tenant. This service principal will get created automatically in tenants tied to subscriptions that have deployed the Azure API for FHIR, but in case your tenant has no subscription tied to it, a tenant administrator will need to create this service principal with one of the following commands:

Using the `Az` PowerShell module:

```
New-AzADServicePrincipal -ApplicationId 3274406e-4e0a-4852-ba4f-d7226630abb7
```

or you can use the `AzureAd` PowerShell module:

```
New-AzureADServicePrincipal -AppId 3274406e-4e0a-4852-ba4f-d7226630abb7
```

or you can use Azure CLI:

```
az ad sp create --id 3274406e-4e0a-4852-ba4f-d7226630abb7
```

Configure local RBAC

You can configure the Azure API for FHIR to use an external or secondary Azure Active Directory tenant in the **Authentication** blade:

In the authority box, enter a valid Azure Active Directory tenant. Once the tenant has been validated, the **Allowed object IDs** box should be activated and you can enter a list of identity object IDs. These IDs can be the identity object IDs of:

- An Azure Active Directory user.
- An Azure Active Directory service principal.
- An Azure Active directory security group.

You can read the article on how to [find identity object IDs](#) for more details.

After entering the required object IDs, click **Save** and wait for changes to be saved before trying to access the data plane using the assigned users, service principals, or groups.

Caching behavior

The Azure API for FHIR will cache decisions for up to 5 minutes. If you grant a user access to the FHIR server by adding them to the list of allowed object IDs, or you remove them from the list, you should expect it to take up to five minutes for changes in permissions to propagate.

Next steps

In this article, you learned how to assign FHIR data plane access using an external (secondary) Azure Active Directory tenant. Next learn about additional settings for the Azure API for FHIR:

[Additional settings Azure API for FHIR](#)

Configure database settings

3/11/2021 • 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.

If the database throughput is greater than 10,000 RU/s or if the data stored in the database is more than 50 GB, your client application must be capable of handling continuation tokens. A new partition is created in the database for every throughput increase of 10,000 RU/s or if the amount of data stored is more than 50 GB. Multiple partitions creates a multi-page response in which pagination is implemented by using continuation tokens.

NOTE

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

Home > Azure API for FHIR > fhirdemoaccount - Database

fhirdemoaccount - Database

Azure API for FHIR

Search (Ctrl+/) Save Discard Refresh

- Overview
- Activity log
- Access control (IAM)
- Tags
- Settings
 - Authentication
 - CORS
 - Database**
 - Locks
 - Export template
- Monitoring
 - Metrics
 - Diagnostic settings
 - Logs
- Support + troubleshooting
 - New support request

Specify the desired throughput (RU/s) for the database account used by your Azure API for FHIR. Please refer to <https://docs.microsoft.com/en-us/azure/healthcare-apis/configure-database> for more details on Request Units. You can set a maximum of 10,000 RU/s. If you need more than 10,000 RU/s, please contact Azure support.

Provisioned throughput (RU/s) * ⓘ

Next steps

In this article, you learned how to update your RUs for Azure API for FHIR. To learn about configuring customer-managed keys as a database setting:

[Configure customer-managed keys](#)

Or you can deploy a fully managed Azure API for FHIR:

[Deploy Azure API for FHIR](#)

Configure customer-managed keys at rest

5/28/2021 • 3 minutes to read • [Edit Online](#)

When you create a new Azure API for FHIR account, your data is encrypted using Microsoft-managed keys by default. Now, you can add a second layer of encryption for the data using your own key that you choose and manage yourself.

In Azure, this is typically accomplished using an encryption key in the customer's Azure Key Vault. Azure SQL, Azure Storage, and Cosmos DB are some examples that provide this capability today. Azure API for FHIR leverages this support from Cosmos DB. When you create an account, you will have the option to specify an Azure Key Vault key URI. This key will be passed on to Cosmos DB when the DB account is provisioned. When a FHIR request is made, Cosmos DB fetches your key and uses it to encrypt/decrypt the data.

To get started, refer to the following links:

- [Register the Azure Cosmos DB resource provider for your Azure subscription](#)
- [Configure your Azure Key Vault instance](#)
- [Add an access policy to your Azure Key Vault instance](#)
- [Generate a key in Azure Key Vault](#)

Using Azure portal

When creating your Azure API for FHIR account on Azure portal, you'll notice **Data Encryption** configuration option under the **Database Settings** on the **Additional Settings** tab. By default, the service-managed key option will be selected.

IMPORTANT

The data encryption option is only available when the Azure API for FHIR is created and cannot be changed afterwards. However, you can view and update the encryption key if the **Customer-managed key** option is selected.

You can choose your key from the KeyPicker:

[Home](#) > [Azure API for FHIR](#) > [Create Azure API for FHIR](#) >

Select key from Azure Key Vault

Subscription *

Key vault *
[Create new](#)

Key *

- byokkey
- byokkey2
- disabled-key
- key-size-2048
- key-size-4096
- key-type-EC

You can also specify your Azure Key Vault key here by selecting **Customer-managed key** option.

You can also enter the key URI here:

Create Azure API for FHIR

* Basics * **Additional settings** Tags Review + create

Customize additional configuration parameters including authentication and storage.

Authentication

Authority * ✓

Audience * ✓

Allowed object IDs ⓘ

Use Azure Access Control (IAM) to grant access your FHIR service when using the subscription tenant for data plane RBAC. [Learn more.](#)

SMART on FHIR proxy ⓘ

☐

Database Settings

Provisioned throughput (RU/s) * ⓘ ✓

Data Encryption ⓘ

- ☐ Service-managed key
☒ Customer-managed key

Select a key

Key URI *

Example: `https://<my-vault>.vault.azure.net/keys/<my-key>`

IMPORTANT

Ensure all permissions for Azure Key Vault are set appropriately. For more information, see [Add an access policy to your Azure Key Vault instance](#). Additionally, ensure that the soft delete is enabled in the properties of the Key Vault. Not completing these steps will result in a deployment error. For more information, see [Verify if soft delete is enabled on a key vault and enable soft delete](#).

For existing FHIR accounts, you can view the key encryption choice (**Service-managed key** or **Customer-managed key**) in the **Database** blade as shown below. The configuration option can't be modified once it's selected. However, you can modify and update your key.

«

Save
Discard
Refresh

Specify the desired throughput (RU/s) for the database account used by your Azure API for FHIR. Please refer to <https://docs.microsoft.com/en-us/azure/> please contact Azure support.

Provisioned throughput (RU/s) * ⓘ

Specify the key used to encrypt the database at rest. The selection for service-managed or customer-managed keys can only be done at creation time.

Data Encryption ☐ Service-managed key ☒ Customer-managed key

[Update key](#)

Key URI *

In addition, you can create a new version of the specified key, after which your data will get encrypted with the new version without any service interruption. You can also remove access to the key to remove access to the data. When the key is disabled, queries will result in an error. If the key is re-enabled, queries will succeed again.

Using Azure PowerShell

With your Azure Key Vault key URI, you can configure CMK using PowerShell by running the PowerShell command below:

```
New-AzHealthcareApisService
  -Name "myService"
  -Kind "fhir-R4"
  -ResourceGroupName "myResourceGroup"
  -Location "westus2"
  -CosmosKeyVaultKeyUri "https://<my-vault>.vault.azure.net/keys/<my-key>"
```

Using Azure CLI

As with PowerShell method, you can configure CMK by passing your Azure Key Vault key URI under the `key-vault-key-uri` parameter and running the CLI command below:

```
az healthcareapis service create
  --resource-group "myResourceGroup"
  --resource-name "myResourceName"
  --kind "fhir-R4"
  --location "westus2"
  --cosmos-db-configuration key-vault-key-uri="https://<my-vault>.vault.azure.net/keys/<my-key>"
```

Using Azure Resource Manager Template

With your Azure Key Vault key URI, you can configure CMK by passing it under the `keyVaultKeyUri` property in the `properties` object.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "services_myService_name": {
      "defaultValue": "myService",
      "type": "String"
    }
  },
  "variables": {},
  "resources": [
    {
      "type": "Microsoft.HealthcareApis/services",
      "apiVersion": "2020-03-30",
      "name": "[parameters('services_myService_name')]",
      "location": "westus2",
      "kind": "fhir-R4",
      "properties": {
        "accessPolicies": [],
        "cosmosDbConfiguration": {
          "offerThroughput": 400,
          "keyVaultKeyUri": "https://<my-vault>.vault.azure.net/keys/<my-key>"
        },
        "authenticationConfiguration": {
          "authority": "https://login.microsoftonline.com/72f988bf-86f1-41af-91ab-2d7cd011db47",
          "audience": "[concat('https://', parameters('services_myService_name'),
'.azurehealthcareapis.com')]",
          "smartProxyEnabled": false
        },
        "corsConfiguration": {
          "origins": [],
          "headers": [],
          "methods": [],
          "maxAge": 0,
          "allowCredentials": false
        }
      }
    }
  ]
}
```

And you can deploy the template with the following PowerShell script:

```
$resourceGroupName = "myResourceGroup"
$accountName = "mycosmosaccount"
$accountLocation = "West US 2"
$keyVaultKeyUri = "https://<my-vault>.vault.azure.net/keys/<my-key>"

New-AzResourceGroupDeployment `
  -ResourceGroupName $resourceGroupName `
  -TemplateFile "deploy.json" `
  -accountName $accountName `
  -location $accountLocation `
  -keyVaultKeyUri $keyVaultKeyUri
```

Next steps

In this article, you learned how to configure customer-managed keys at rest using the Azure portal, PowerShell, CLI, and Resource Manager Template. You can refer to the Azure Cosmos DB FAQ section for more information.

[Cosmos DB: how to setup CMK](#)

Configure cross-origin resource sharing in Azure API for FHIR

3/11/2021 • 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 (*).

[Home](#) > [matjazl-fhir - CORS](#)



matjazl-fhir - CORS
Azure API for FHIR

Search (Cmd+[/](#))

Save Discard Refresh

Overview

Activity log

Access control (IAM)

Tags

Settings

Authentication

CORS

Cosmos DB

Locks

Export template

Monitoring

Metrics

Diagnostic settings

Logs

Support + troubleshooting

New support request

[Cross-origin resource sharing \(CORS\)](#) is a security mechanism that allows a web page from one domain or origin to access a resource with a different domain (a cross-domain request). Without features like CORS, websites are restricted to accessing resources from the same origin through what is known as same-origin policy. Please refer to <https://docs.microsoft.com/azure/healthcare-apis/configure-cross-origin-resource-sharing> for details on how to configure CORS.

Origins ⓘ

*

✓

Headers ⓘ

*

Methods ⓘ

6 selected

▼

Max age ⓘ

600

Allow credentials ⓘ

☐

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 set up the storage account

5/17/2021 • 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.

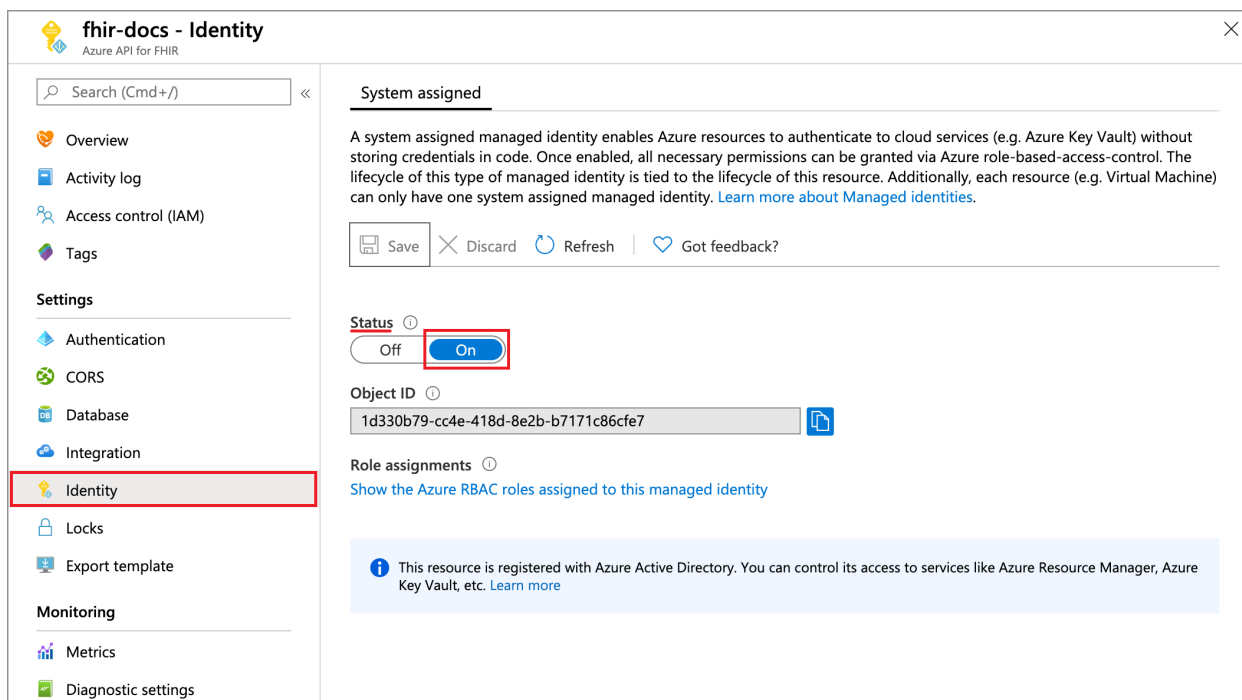
There are three steps involved in configuring export in Azure API for FHIR:

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.

Enabling Managed Identity on Azure API for FHIR

The first step in configuring Azure API for FHIR for export is to enable system wide managed identity on the service. For more information about managed identities in Azure, see [About managed identities for Azure resources](#).

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



Now, you can move to the next step by creating a storage account and assign permission to our service.

Adding permission to storage account

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

After you've created a storage account, go to the **Access Control (IAM)** in the storage account, and then select **Add role assignment**.

For more information about assigning roles in the Azure portal, see [Azure built-in roles](#).

It is here that you'll add the role [Storage Blob Data Contributor](#) to our service name, and then select **Save**.

Add role assignment ✕

Role ⓘ
Select a role ▼

Assign access to ⓘ
User, group, or service principal ▼

Select ⓘ
Search by name or email address

- AD Admin
- AL Alain
- AT Alain Team

Selected members:
No members selected. Search for and add one or more members you want to assign to the role for this resource.

[Learn more about RBAC](#)

Save Discard

Now you are ready to select the storage account in Azure API for FHIR as a default storage account for \$export.

Selecting the storage account for \$export

The final step is to assign the Azure storage account that Azure API for FHIR will use to export the data to. To do this, go to **Integration** in Azure API for FHIR service and select the storage account.

fhir-docs | Integration

Azure API for FHIR

Search (Cmd+ /)

<<

Overview

Activity log

Access control (IAM)

Tags

Settings

Authentication

CORS

Database

Integration

Identity

Locks

Export template

Save

Discard

Refresh

Views and configures settings for integrating the Azure API for FHIR server with other Azure services.

Export

When attaching a storage account for export make sure that the FHIR server has permission to access it by enabling System Assigned Managed Identity and giving Azure API for FHIR permission to the storage account.

Export Storage Account

fhirexporter

Name

Resource group: matjazl-fhir-sample

Region: westus2

Name ↑↓

Resource group ↑↓

Region ↑↓

fhirexporter	matjazl-fhir-sample	westus2
--------------	---------------------	---------

After you've completed this final step, you are now ready to export the data using `$export` command.

NOTE

Only storage accounts in the same subscription as that for Azure API for FHIR are allowed to be registered as the destination for `$export` operations.

For more information about configuring database settings, access control, enabling diagnostic logging, and using custom headers to add data to audit logs, see:

[Additional Settings](#)

Configure private link

5/28/2021 • 3 minutes to read • [Edit Online](#)

Private link enables you to access Azure API for FHIR over a private endpoint, which is a network interface that connects you privately and securely using a private IP address from your virtual network. With private link, you can access our services securely from your VNet as a first party service without having to go through a public Domain Name System (DNS). This article describes how to create, test, and manage your private endpoint for Azure API for FHIR.

NOTE

Neither Private Link nor Azure API for FHIR can be moved from one resource group or subscription to another once Private Link is enabled. To make a move, delete the Private Link first, then move Azure API for FHIR. Create a new Private Link once the move is complete. Assess potential security ramifications before deleting Private Link.

If exporting audit logs and metrics is enabled for Azure API for FHIR, update the export setting through **Diagnostic Settings** from the portal.

Prerequisites

Before creating a private endpoint, there are some Azure resources that you'll need to create first:

- Resource Group – The Azure resource group that will contain the virtual network and private endpoint.
- Azure API for FHIR – The FHIR resource you would like to put behind a private endpoint.
- Virtual Network – The VNet to which your client services and Private Endpoint will be connected.

For more information, see [Private Link Documentation](#).

Create private endpoint

To create a private endpoint, a developer with Role-based access control (RBAC) permissions on the FHIR resource can use the Azure portal, [Azure PowerShell](#), or [Azure CLI](#). This article will guide you through the steps on using Azure portal. Using the Azure portal is recommended as it automates the creation and configuration of the Private DNS Zone. For more information, see [Private Link Quick Start Guides](#).

There are two ways to create a private endpoint. Auto Approval flow allows a user that has RBAC permissions on the FHIR resource to create a private endpoint without a need for approval. Manual Approval flow allows a user without permissions on the FHIR resource to request a private endpoint to be approved by owners of the FHIR resource.


NOTE

When an approved private endpoint is created for Azure API for FHIR, public traffic to it is automatically disabled.

Auto approval

Ensure the region for the new private endpoint is the same as the region for your virtual network. The region for your FHIR resource can be different.

Create a private endpoint

 Changes you make on this tab may affect any configuration you've done on other tabs. Review all options prior to creating the private endpoint.

✓ Basics 2 Resource 3 Configuration 4 Tags 5 Review + create

Use private endpoints to privately connect to a service or resource. Your private endpoint must be in the same region as your virtual network, but can be in a different region from the private link resource that you are connecting to. [Learn more](#)

Project details

Subscription * ⓘ ✓
Resource group * ⓘ ✓
[Create new](#)

Instance details

Name * ✓
Region * ✓

For the resource type, search and select **Microsoft.HealthcareApis/services**. For the resource, select the FHIR resource. For target sub-resource, select **FHIR**.

Create a private endpoint

✓ Basics 2 Resource 3 Configuration 4 Tags 5 Review + create

Private Link offers options to create private endpoints for different Azure resources, like your private link service, a SQL server, or an Azure storage account. Select which resource you would like to connect to using this private endpoint. [Learn more](#)

Connection method ⓘ
☒ Connect to an Azure resource in my directory.
☐ Connect to an Azure resource by resource ID or alias.

Subscription * ⓘ ✓
Resource type * ⓘ ✓
Resource * ⓘ ✓

If you do not have an existing Private DNS Zone set up, select **(New)privatelink.azurehealthcareapis.com**. If you already have your Private DNS Zone configured, you can select it from the list. It must be in the format of **privatelink.azurehealthcareapis.com**.

Create a private endpoint

- ✓ Basics
- ✓ Resource
- 3 Configuration**
- 4 Tags
- 5 Review + create

Networking

To deploy the private endpoint, select a virtual network subnet. [Learn more](#)

Virtual network * ⓘ

plexample

Subnet * ⓘ

default (10.8.0.0/24)

i If you have a network security group (NSG) enabled for the subnet above, it will be disabled for private endpoints on this subnet only. Other resources on the subnet will still have NSG enforcement.

Private DNS integration

To connect privately with your private endpoint, you need a DNS record. We recommend that you integrate your private endpoint with a private DNS zone. You can also utilize your own DNS servers or create DNS records using the host files on your virtual machines. [Learn more](#)

Integrate with private DNS zone

☒ Yes

☐ No

Configuration name	Subscription	Private DNS zones
privatelink-azurehealt...	<div>your subscription here</div>	<div>(New) privatelink.azurehealthcareapis.com</div>

After the deployment is complete, you can go back to **Private endpoint connections** tab of which you'll notice **Approved** as the connection state.

Manual Approval

For manual approval, select the second option under Resource, "Connect to an Azure resource by resource ID or alias". For Target sub-resource, enter "fhir" as in Auto Approval.

Create a private endpoint

- ✓ Basics
- 2 Resource**
- 3 Configuration
- 4 Tags
- 5 Review + create

Private Link offers options to create private endpoints for different Azure resources, like your private link service, a SQL server, or an Azure storage account. Select which resource you would like to connect to using this private endpoint. [Learn more](#)

Connection method ⓘ

☐ Connect to an Azure resource in my directory.

☒ Connect to an Azure resource by resource ID or alias.

After the deployment is complete, you can go back to "Private endpoint connections" tab, on which you can Approve, Reject, or Remove your connection.

Public access

Private endpoint connections

Private endpoint

Approve

Reject

Remove

Refresh

Filter by name...

All connection states

<input type="checkbox"/> CONNECTION NAME	CONNECTION STATE	PRIVATE ENDPOINT	DESCRIPTION
--	------------------	------------------	-------------

Test private endpoint

To ensure that your FHIR server is not receiving public traffic after disabling public network access, select the

/metadata endpoint for your server from your computer. You should receive a 403 Forbidden.

NOTE

It can take up to 5 minutes after updating the public network access flag before public traffic is blocked.

To ensure your private endpoint can send traffic to your server:

1. Create a virtual machine (VM) that is connected to the virtual network and subnet your private endpoint is configured on. To ensure your traffic from the VM is only using the private network, disable the outbound internet traffic using the network security group (NSG) rule.
2. RDP into the VM.
3. Access your FHIR server's /metadata endpoint from the VM. You should receive the capability statement as a response.

Manage private endpoint

View

Private endpoints and the associated network interface controller (NIC) are visible in Azure portal from the resource group they were created in.

The screenshot shows the Azure portal interface for a resource group named 'pl-example'. The left sidebar contains navigation options: Overview, Activity log, Access control (IAM), Tags, Events, Settings, Quickstart, Resource costs, Deployments, Policies, Properties, and Locks. The main area displays the 'Essentials' section with subscription details and a list of private endpoints. The list is filtered by name and shows 5 records. The 'plexample.nic.eb' entry is highlighted with a red box, indicating it is the selected private endpoint.

Name	Type	Location
plexample	Private endpoint	West US
plexample	Private endpoint	West US
plexample	Private endpoint	West US
plexample.nic.eb	Private endpoint	West US
privatelink.azurehealthcareapis.com	Private endpoint	West US

Delete

Private endpoints can only be deleted from the Azure portal from the **Overview** blade or by selecting the **Remove** option under the **Networking Private endpoint connections** tab. Selecting **Remove** will delete the private endpoint and the associated NIC. If you delete all private endpoints to the FHIR resource and the public network, access is disabled and no request will make it to your FHIR server.

Showing 1 to 26 of 26 records.

<div><input checked="" type="checkbox"/></div> Name ↑↓	Resource ↑↓	Target sub-resource ↑↓	Subnet ↑↓	Connection state ↑↓
<input type="checkbox"/> pe1	 anr	fhir	an/default	Approved
<input type="checkbox"/> peportal	 anr	fhir	y/default	Approved
<input type="checkbox"/> test1	 createflowone	fhir	an/default	Approved
<input type="checkbox"/> pe-portal	 erti	fhir	lcan/default	Approved
<input type="checkbox"/> pe-pshell	 e	fhir	lcan/default	Approved
<input type="checkbox"/> fhir-api-private-endpoint	 fhir-api-private-link	fhir	fhir-api-virtual-network/default	Approved
<input checked="" type="checkbox"/> plexample	 plexample	fhir	plexample/default	Approved

Overview of FHIR search

5/25/2021 • 7 minutes to read • [Edit Online](#)

The FHIR specification defines the fundamentals of search for FHIR resources. This article will guide you through some key aspects to searching resources in FHIR. For complete details about searching FHIR resources, refer to [Search](#) in the HL7 FHIR Specification. Throughout this article, we will give examples of search syntax. Each search will be against your FHIR server, which typically has a URL of

`https://<FHIRSERVERNAME>.azurewebsites.net`. In the examples, we will use the placeholder `{{FHIR_URL}}` for this URL.

FHIR searches can be against a specific resource type, a specified [compartment](#), or all resources. The simplest way to execute a search in FHIR is to use a `GET` request. For example, if you want to pull all patients in the database, you could use the following request:

```
GET {{FHIR_URL}}/Patient
```

You can also search using `POST`, which is useful if the query string is too long. To search using `POST`, the search parameters can be submitted as a form body. This allows for longer, more complex series of query parameters that might be difficult to see and understand in a query string.

If the search request is successful, you'll receive a FHIR bundle response with the type `searchset`. If the search fails, you'll find the error details in the `OperationOutcome` to help you understand why the search failed.

In the following sections, we'll cover the various aspects involved in searching. Once you've reviewed these details, refer to our [samples page](#) that has examples of searches that you can make in the Azure API for FHIR.

Search parameters

When you do a search, you'll search based on various attributes of the resource. These attributes are called search parameters. Each resource has a set of defined search parameters. The search parameter must be defined and indexed in the database for you to successfully search against it.

Each search parameter has a defined [data types](#). The support for the various data types is outlined below:

SEARCH PARAMETER TYPE	SUPPORTED - PAAS	SUPPORTED - OSS (SQL)	SUPPORTED - OSS (COSMOS DB)	COMMENT
number	Yes	Yes	Yes	
date	Yes	Yes	Yes	
string	Yes	Yes	Yes	
token	Yes	Yes	Yes	
reference	Yes	Yes	Yes	
composite	Partial	Partial	Partial	The list of supported composite types is described later in this article

SEARCH PARAMETER TYPE	SUPPORTED - PAAS	SUPPORTED - OSS (SQL)	SUPPORTED - OSS (COSMOS DB)	COMMENT
quantity	Yes	Yes	Yes	
uri	Yes	Yes	Yes	
special	No	No	No	

Common search parameters

There are [common search parameters](#) that apply to all resources. These are listed below, along with their support within the Azure API for FHIR:

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	
_type	Yes	Yes	Yes	
_security	Yes	Yes	Yes	
_profile	Yes	Yes	Yes	If you created your R4 database before February 20, 2021, you'll need to run a reindex job to enable _profile .
_has	Partial	Yes	Partial	Support for _has is in MVP in the Azure API for FHIR and the OSS version backed by Cosmos DB. More details are included under the chaining section below.
_query	No	No	No	
_filter	No	No	No	
_list	No	No	No	
_text	No	No	No	
_content	No	No	No	

Resource-specific parameters

With the Azure API for FHIR, we support almost all [resource-specific search parameters](#) defined by the FHIR specification. The only search parameters we don't support are available in the links below:

- [STU3 Unsupported Search Parameters](#)
- [R4 Unsupported Search Parameters](#)

You can also see the current support for search parameters in the [FHIR Capability Statement](#) with the following request:

```
GET {{FHIR_URL}}/metadata
```

To see the search parameters in the capability statement, navigate to

`CapabilityStatement.rest.resource.searchParam` to see the search parameters for each resource and

`CapabilityStatement.rest.searchParam` to find the search parameters for all resources.

NOTE

The Azure API for FHIR does not automatically create or index any search parameters that are not defined by the FHIR specification. However, we do provide support for you to define your own [search parameters](#).

Composite search parameters

Composite search allows you to search against value pairs. For example, if you were searching for a height observation where the person was 60 inches, you would want to make sure that a single component of the observation contained the code of height **and** the value of 60. You wouldn't want to get an observation where a weight of 60 and height of 48 was stored, even though the observation would have entries that qualified for value of 60 and code of height, just in different component sections.

With the Azure API for FHIR, we support the following search parameter type pairings:

- Reference, Token
- Token, Date
- Token, Number, Number
- Token, Quantity
- Token, String
- Token, Token

For more information, see the HL7 [Composite Search Parameters](#).

NOTE

Composite search parameters do not support modifiers per the FHIR specification.

Modifiers & prefixes

[Modifiers](#) allow you to modify the search parameter. Below is an overview of all the FHIR modifiers and the support in the Azure API for FHIR.

MODIFIERS	SUPPORTED - PAAS	SUPPORTED - OSS (SQL)	SUPPORTED - OSS (COSMOS DB)
:missing	Yes	Yes	Yes
:exact	Yes	Yes	Yes
:contains	Yes	Yes	Yes

MODIFIERS	SUPPORTED - PAAS	SUPPORTED - OSS (SQL)	SUPPORTED - OSS (COSMOS DB)
:text	Yes	Yes	Yes
:type (reference)	Yes	Yes	Yes
:not	Yes	Yes	Yes
:below (uri)	Yes	Yes	Yes
:above (uri)	Yes	Yes	Yes
:in (token)	No	No	No
:below (token)	No	No	No
:above (token)	No	No	No
:not-in (token)	No	No	No

For search parameters that have a specific order (numbers, dates, and quantities), you can use a [prefix](#) on the parameter to help with finding matches. The Azure API for FHIR supports all prefixes.

Search result parameters

To help manage the returned resources, there are search result parameters that you can use in your search. For details on how to use each of the search result parameters, refer to the [HL7](#) website.

SEARCH RESULT PARAMETERS	SUPPORTED - PAAS	SUPPORTED - OSS (SQL)	SUPPORTED - OSS (COSMOS DB)	COMMENTS
_elements	Yes	Yes	Yes	
_count	Yes	Yes	Yes	_count is limited to 1000 resources. If it's set higher than 1000, only 1000 will be returned and a warning will be returned in the bundle.
_include	Yes	Yes	Yes	Included items are limited to 100. _include on PaaS and OSS on Cosmos DB do not include :iterate support (#1313).
_revinclude	Yes	Yes	Yes	Included items are limited to 100. _revinclude on PaaS and OSS on Cosmos DB do not include :iterate support (#1313). Issue #1319

SEARCH RESULT PARAMETERS	SUPPORTED - PAAS	SUPPORTED - OSS (SQL)	SUPPORTED - OSS (COSMOS DB)	COMMENTS
_summary	Yes	Yes	Yes	
_total	Partial	Partial	Partial	_total=none and _total=accurate
_sort	Partial	Partial	Partial	sort=_lastUpdated is supported. For Azure API for FHIR and OSS Cosmos DB databases created after April 20, 2021 sort is also supported on first name, last name, and clinical date.
_contained	No	No	No	
_containedType	No	No	No	
_score	No	No	No	

By default, the Azure API for FHIR is set to lenient handling. This means that the server will ignore any unknown or unsupported parameters. If you want to use strict handling, you can use the **Prefer** header and set `handling=strict`.

Chained & reverse chained searching

A [chained search](#) allows you to search using a search parameter on a resource referenced by another resource. For example, if you want to find encounters where the patient's name is Jane, use:

```
GET {{FHIR_URL}}/Encounter?subject:Patient.name=Jane
```

Similarly, you can do a reverse chained search. This allows you to get resources where you specify criteria on other resources that refer to them. For more examples of chained and reverse chained search, refer to the [FHIR search examples](#) page.

NOTE

In the Azure API for FHIR and the open source backed by Cosmos DB, there's a limitation where each subquery required for the chained and reverse chained searches will only return 100 items. If there are more than 100 items found, you'll receive the following error message: "Subqueries in a chained expression can't return more than 100 results, please use a more selective criteria." To get a successful query, you'll need to be more specific in what you are looking for.

Pagination

As mentioned above, the results from a search will be a paged bundle. By default, the search will return 10 results per page, but this can be increased (or decreased) by specifying `_count`. Within the bundle, there will be a self link that contains the current result of the search. If there are additional matches, the bundle will contain a next link. You can continue to use the next link to get the subsequent pages of results. `_count` is limited to 1000 items or less.

Currently, the Azure API for FHIR only supports the next link in bundles, and it doesn't support first, last, or

previous links.

Next steps

Now that you've learned about the basics of search, see the search samples page for details about how to search using different search parameters, modifiers, and other FHIR search scenarios.

[FHIR search examples](#)

Defining custom search parameters

5/25/2021 • 5 minutes to read • [Edit Online](#)

The FHIR specification defines a set of search parameters for all resources and search parameters that are specific to a resource(s). However, there are scenarios where you might want to search against an element in a resource that isn't defined by the FHIR specification as a standard search parameter. This article describes how you can define your own [search parameters](#) to be used in the Azure API for FHIR.

NOTE

Each time you create, update, or delete a search parameter you'll need to run a [reindex job](#) to enable the search parameter to be used in production. Below we will outline how you can test search parameters before reindexing the entire FHIR server.

Create new search parameter

To create a new search parameter, you `POST` the `SearchParameter` resource to the database. The code example below shows how to add the [US Core Race SearchParameter](#) to the `Patient` resource.

POST {{FHIR_URL}}/SearchParameter

```
{
  "resourceType" : "SearchParameter",
  "id" : "us-core-race",
  "url" : "http://hl7.org/fhir/us/core/SearchParameter/us-core-race",
  "version" : "3.1.1",
  "name" : "USCoreRace",
  "status" : "active",
  "date" : "2019-05-21",
  "publisher" : "US Realm Steering Committee",
  "contact" : [
    {
      "telecom" : [
        {
          "system" : "other",
          "value" : "http://www.healthit.gov/"
        }
      ]
    }
  ],
  "description" : "Returns patients with a race extension matching the specified code.",
  "jurisdiction" : [
    {
      "coding" : [
        {
          "system" : "urn:iso:std:iso:3166",
          "code" : "US",
          "display" : "United States of America"
        }
      ]
    }
  ],
  "code" : "race",
  "base" : [
    "Patient"
  ],
  "type" : "token",
  "expression" : "Patient.extension.where(url = 'http://hl7.org/fhir/us/core/StructureDefinition/us-core-race').extension.value.code"
}
```

NOTE

The new search parameter will appear in the capability statement of the FHIR server after you POST the search parameter to the database **and** reindex your database. Viewing the `SearchParameter` in the capability statement is the only way to tell if a search parameter is supported in your FHIR server. If you can find the search parameter by searching for the search parameter but cannot see it in the capability statement, you still need to index the search parameter. You can POST multiple search parameters before triggering a reindex operation.

Important elements of a `SearchParameter` :

- **url**: A unique key to describe the search parameter. Many organizations, such as HL7, use a standard URL format for the search parameters that they define, as shown above in the US Core race search parameter.
- **code**: The value stored in **code** is what you'll use when searching. For the example above, you would search with `GET {FHIR_URL}/Patient?race=<code>` to get all patients of a specific race. The code must be unique for the resource(s) the search parameter applies to.
- **base**: Describes which resource(s) the search parameter applies to. If the search parameter applies to all resources, you can use `Resource`; otherwise, you can list all the relevant resources.

- **type**: Describes the data type for the search parameter. Type is limited by the support for the Azure API for FHIR. This means that you cannot define a search parameter of type Special or define a [composite search parameter](#) unless it is a supported combination.
- **expression**: Describes how to calculate the value for the search. When describing a search parameter, you must include the expression, even though it is not required by the specification. This is because you need either the expression or the xpath syntax and the Azure API for FHIR ignores the xpath syntax.

Test search parameters

While you cannot use the search parameters in production until you run a reindex job, there are a few ways to test your search parameters before reindexing the entire database.

First, you can test your new search parameter to see what values will be returned. By running the command below against a specific resource instance (by inputting their ID), you'll get back a list of value pairs with the search parameter name and the value stored for the specific patient. This will include all of the search parameters for the resource and you can scroll through to find the search parameter you created. Running this command will not change any behavior in your FHIR server.

```
GET https://{FHIR_URL}/{RESOURCE}/{RESOURCE_ID}/$reindex
```

For example, to find all search parameters for a patient:

```
GET https://{FHIR_URL}/Patient/{PATIENT_ID}/$reindex
```

The result will look like this:

```
{
  "resourceType": "Parameters",
  "id": "8be24e78-b333-49da-a861-523491c3437a",
  "meta": {
    "versionId": "1"
  },
  "parameter": [
    {
      "name": "deceased",
      "valueString": "http://hl7.org/fhir/special-values|false"
    },
    {
      "name": "language",
      "valueString": "urn:ietf:bcp:47|en-US"
    },
    {
      "name": "race",
      "valueString": "2028-9"
    },
    ...
  ]
}
```

Once you see that your search parameter is displaying as expected, you can reindex a single resource to test searching with the element. First you will reindex a single resource:

```
POST https://{FHIR_URL}/{RESOURCE}/{RESOURCE_ID}/$reindex
```

Running this, sets the indices for any search parameters for the specific resource that you defined for that

resource type. This does make an update to the FHIR server. Now you can search and set the use partial indices header to true, which means that it will return results where any of the resources has the search parameter indexed, even if not all resources of that type have it indexed.

Continuing with our example above, you could index one patient to enable the US Core Race `SearchParameter` :

```
POST https://{{FHIR_URL}}/Patient/{{PATIENT_ID}}/$reindex
```

And then search for patients that have a specific race:

```
GET https://{{FHIR_URL}}/Patient?race=2028-9
x-ms-use-partial-indices: true
```

After you have tested and are satisfied that your search parameter is working as expected, run or schedule your reindex job so the search parameters can be used in the FHIR server for production use cases.

Update a search parameter

To update a search parameter, use `PUT` to create a new version of the search parameter. You must include the `SearchParameter ID` in the `id` element of the body of the `PUT` request and in the `PUT` call.

NOTE

If you don't know the ID for your search parameter, you can search for it. Using `GET {{FHIR_URL}}/SearchParameter` will return all custom search parameters, and you can scroll through the search parameter to find the search parameter you need. You could also limit the search by name. With the example below, you could search for name using

```
USCoreRace: GET {{FHIR_URL}}/SearchParameter?name=USCoreRace .
```



```

PUT {{FHIR_URL}}/SearchParameter/{SearchParameter ID}

{
  "resourceType" : "SearchParameter",
  "id" : "SearchParameter ID",
  "url" : "http://hl7.org/fhir/us/core/SearchParameter/us-core-race",
  "version" : "3.1.1",
  "name" : "USCoreRace",
  "status" : "active",
  "date" : "2019-05-21",
  "publisher" : "US Realm Steering Committee",
  "contact" : [
    {
      "telecom" : [
        {
          "system" : "other",
          "value" : "http://www.healthit.gov/"
        }
      ]
    }
  ],
  "description" : "New Description!",
  "jurisdiction" : [
    {
      "coding" : [
        {
          "system" : "urn:iso:std:iso:3166",
          "code" : "US",
          "display" : "United States of America"
        }
      ]
    }
  ],
  "code" : "race",
  "base" : [
    "Patient"
  ],
  "type" : "token",
  "expression" : "Patient.extension.where(url = 'http://hl7.org/fhir/us/core/StructureDefinition/us-core-race').extension.value.code"
}

```

The result will be an updated `SearchParameter` and the version will increment.

WARNING

Be careful when updating `SearchParameters` that have already been indexed in your database. Changing an existing `SearchParameter`'s behavior could have impacts on the expected behavior. We recommend running a reindex job immediately.

Delete a search parameter

If you need to delete a search parameter, use the following:

```

Delete {{FHIR_URL}}/SearchParameter/{SearchParameter ID}

```

WARNING

Be careful when deleting SearchParameters that have already been indexed in your database. Changing an existing SearchParameter's behavior could have impacts on the expected behavior. We recommend running a reindex job immediately.

Next steps

In this article, you've learned how to create a search parameter. Next you can learn how to reindex your FHIR server.

[How to run a reindex job](#)

Running a reindex job

5/25/2021 • 4 minutes to read • [Edit Online](#)

There are scenarios where you may have search or sort parameters in the Azure API for FHIR that haven't yet been indexed. This is particularly relevant when you define your own search parameters. Until the search parameter is indexed, it can't be used in search. This article covers an overview of how to run a reindex job to index any search or sort parameters that have not yet been indexed in your database.

WARNING

It's important that you read this entire article before getting started. A reindex job can be very performance intensive. This article includes options for how to throttle and control the reindex job.

How to run a reindex job

To start a reindex job, use the following code example:

```
POST {{FHIR_URL}}/$reindex

{
  "resourceType": "Parameters",
  "parameter": []
}
```

If the request is successful, a status of **201 Created** gets returned. The result of this message will look like:

HTTP/1.1 201 Created

Content-Location: https://{FHIR URL}}/_operations/reindex/560c7c61-2c70-4c54-b86d-c53a9d29495e

```
{
  "resourceType": "Parameters",
  "id": "560c7c61-2c70-4c54-b86d-c53a9d29495e",
  "meta": {
    "versionId": "\"4c0049cd-0000-0100-0000-607dc5a90000\""
  },
  "parameter": [
    {
      "name": "id",
      "valueString": "560c7c61-2c70-4c54-b86d-c53a9d29495e"
    },
    {
      "name": "queuedTime",
      "valueDateTime": "2021-04-19T18:02:17.0118558+00:00"
    },
    {
      "name": "totalResourcesToReindex",
      "valueDecimal": 0.0
    },
    {
      "name": "resourcesSuccessfullyReindexed",
      "valueDecimal": 0.0
    },
    {
      "name": "progress",
      "valueDecimal": 0.0
    },
    {
      "name": "status",
      "valueString": "Queued"
    },
    {
      "name": "maximumConcurrency",
      "valueDecimal": 1.0
    },
    {
      "name": "resources",
      "valueString": ""
    },
    {
      "name": "searchParams",
      "valueString": ""
    }
  ]
}
```

NOTE

To check the status of or to cancel a reindex job, you'll need the reindex ID. This is the ID of the resulting Parameters resource. In the example above, the ID for the reindex job would be `560c7c61-2c70-4c54-b86d-c53a9d29495e`.

How to check the status of a reindex job

Once you've started a reindex job, you can check the status of the job using the following:

```
GET {FHIR URL}}/_operations/reindex/{reindexJobId}
```

The status of the reindex job result is shown below:

```

{
  "resourceType": "Parameters",
  "id": "b65fd841-1c62-47c6-898f-c9016ced8f77",
  "meta": {
    "versionId": "\"1800f05f-0000-0100-0000-607a1a7c0000\""
  },
  "parameter": [
    {
      "name": "id",
      "valueString": "b65fd841-1c62-47c6-898f-c9016ced8f77"
    },
    {
      "name": "startTime",
      "valueDateTime": "2021-04-16T23:11:35.4223217+00:00"
    },
    {
      "name": "queuedTime",
      "valueDateTime": "2021-04-16T23:11:29.0288163+00:00"
    },
    {
      "name": "totalResourcesToReindex",
      "valueDecimal": 262544.0
    },
    {
      "name": "resourcesSuccessfullyReindexed",
      "valueDecimal": 5754.0
    },
    {
      "name": "progress",
      "valueDecimal": 2.0
    },
    {
      "name": "status",
      "valueString": "Running"
    },
    {
      "name": "maximumConcurrency",
      "valueDecimal": 1.0
    },
    {
      "name": "resources",
      "valueString":
        "{LIST OF IMPACTED RESOURCES}"
    }
  ]
}

```

The following information is shown in the reindex job result:

- **totalResourcesToReindex:** Includes the total number of resources that are being reindexed as part of the job.
- **resourcesSuccessfullyReindexed:** The total that have already been successfully reindexed.
- **progress:** Reindex job percent complete. Equals

$\text{resourcesSuccessfullyReindexed} / \text{totalResourcesToReindex} \times 100$.

- **status**: This will state if the reindex job is queued, running, complete, failed, or canceled.
- **resources**: This lists all the resource types impacted by the reindex job.

Delete a reindex job

If you need to cancel a reindex job, use a delete call and specify the reindex job ID:

```
DELETE {{FHIR_URL}}/_operations/reindex/{{reindexJobId}}
```

Performance considerations

A reindex job can be quite performance intensive. We've implemented some throttling controls to help you manage how a reindex job will run on your database.

NOTE

It is not uncommon on large datasets for a reindex job to run for days. For a database with 30,000,000 million resources, we noticed that it took 4-5 days at 100K RUs to reindex the entire database.

Below is a table outlining the available parameters, defaults, and recommended ranges. You can use these parameters to either speed up the process (use more compute) or slow down the process (use less compute). For example, you could run the reindex job on a low traffic time and increase your compute to get it done quicker. Instead, you could use the settings to ensure a very low usage of compute and have it run for days in the background.

PARAMETER	DESCRIPTION	DEFAULT	RECOMMENDED RANGE
QueryDelayIntervalInMillisecons	This is the delay between each batch of resources being kicked off during the reindex job.	500 MS (.5 seconds)	50 to 5000: 50 will speed up the reindex job and 5000 will slow it down from the default.
MaximumResourcesPerQuery	This is the maximum number of resources included in the batch of resources to be reindexed.	100	1-500
MaximumConcurrency	This is the number of batches done at a time.	1	1-5
targetDataStoreUsagePercentage	This allows you to specify what percent of your data store to use for the reindex job. For example, you could specify 50% and that would ensure that at most the reindex job would use 50% of available RUs on Cosmos DB.	No present, which means that up to 100% can be used.	1-100

If you want to use any of the parameters above, you can pass them into the Parameters resource when you start the reindex job.

```
{
  "resourceType": "Parameters",
  "parameter": [
    {
      "name": "maximumConcurrency",
      "valueInteger": "3"
    },
    {
      "name": "targetDataStoreUsagePercentage",
      "valueInteger": "20"
    },
    {
      "name": "queryDelayIntervalInMilliseconds",
      "valueInteger": "1000"
    },
    {
      "name": "maximumNumberOfResourcesPerQuery",
      "valueInteger": "1"
    }
  ]
}
```

Next steps

In this article, you've learned how to start a reindex job. To learn how to define new search parameters that require the reindex job, see

[Defining custom search parameters](#)

FHIR search examples

5/25/2021 • 7 minutes to read • [Edit Online](#)

Below are some examples of using FHIR search operations, including search parameters and modifiers, chain and reverse chain search, composite search, viewing the next entry set for search results, and searching with a `POST` request. For more information about search, see [Overview of FHIR Search](#).

Search result parameters

_include

`_include` searches across resources for the ones that include the specified parameter of the resource. For example, you can search across `MedicationRequest` resources to find only the ones that include information about the prescriptions for a specific patient, which is the `reference` parameter `patient`. In the example below, this will pull all the `MedicationRequests` and all patients that are referenced from the `MedicationRequests`:

```
GET [your-fhir-server]/MedicationRequest?_include=MedicationRequest:patient
```

NOTE

`_include` and `_revinclude` is limited to 100 items.

_revinclude

`_revinclude` allows you to search the opposite direction as `_include`. For example, you can search for patients and then reverse include all encounters that reference the patients:

```
GET [your-fhir-server]/Patient?_revinclude=Encounter:subject
```

_elements

`_elements` narrows down the search result to a subset of fields to reduce the response size by omitting unnecessary data. The parameter accepts a comma-separated list of base elements:

```
GET [your-fhir-server]/Patient?_elements=identifier,active
```

In this request, you'll get back a bundle of patients, but each resource will only include the identifier(s) and the patient's active status. Resources in this returned response will contain a `meta.tag` value of `SUBSETTED` to indicate that they're an incomplete set of results.

Search modifiers

:not

`:not` allows you to find resources where an attribute is not true. For example, you could search for patients where the gender is not female:


```
GET [your-fhir-server]/Patient?gender:not=female
```

As a return value, you would get all patient entries where the gender is not female, including empty values (entries specified without gender). This is different than searching for Patients where gender is male, since that would not include the entries without a specific gender.

:missing

`:missing` returns all resources that don't have a value for the specified element when the value is `true`, and returns all the resources that contain the specified element when the value is `false`. For simple data type elements, `:missing=true` will match on all resources where the element is present with extensions but has an empty value. For example, if you want to find all `Patient` resources that are missing information on birth date, you can do:

```
GET [your-fhir-server]/Patient?birthdate:missing=true
```

:exact

`:exact` is used for `string` parameters, and returns results that match the parameter precisely, such as in casing and character concatenating.

```
GET [your-fhir-server]/Patient?name:exact=Jon
```

This request returns `Patient` resources that have the name exactly the same as `Jon`. If the resource had Patients with names such as `Jonathan` or `joN`, the search would ignore and skip the resource as it does not exactly match the specified value.

:contains

`:contains` is used for `string` parameters and searches for resources with partial matches of the specified value anywhere in the string within the field being searched. `contains` is case insensitive and allows character concatenating. For example:

```
GET [your-fhir-server]/Patient?address:contains=Meadow
```

This request would return you all `Patient` resources with `address` fields that have values that contain the string "Meadow". This means you could have addresses that include values such as "Meadowers" or "59 Meadow ST" returned as search results.

Chained search

To perform a series of search operations that cover multiple reference parameters, you can "chain" the series of reference parameters by appending them to the server request one by one using a period `.`. For example, if you want to view all `DiagnosticReport` resources with a `subject` reference to a `Patient` resource that includes a particular `name`:

```
GET [your-fhir-server]/DiagnosticReport?subject:Patient.name=Sarah
```

This request would return all the resources with the patient subject named "Sarah". The period `.` after the field `Patient` performs the chained search on the reference parameter of the `subject` parameter.

Another common use of a regular search (not a chained search) is finding all encounters for a specific patient. `Patient`s will often have one or more `Encounter`s with a subject. To search for all `Encounter` resources for a `Patient` with the provided `id`:

```
GET [your-fhir-server]/Encounter?subject=Patient/78a14cbe-8968-49fd-a231-d43e6619399f
```

Using chained search, you can find all the `Encounter` resources that matches a particular piece of `Patient` information, such as the `birthdate`:

```
GET [your-fhir-server]/Encounter?subject:Patient.birthdate=1987-02-20
```

This would allow not just searching `Encounter` resources for a single patient, but across all patients that have the specified birth date value.

In addition, chained search can be done more than once in one request by using the symbol `&`, which allows you to search for multiple conditions in one request. In such cases, chained search "independently" searches for each parameter, instead of searching for conditions that only satisfy all the conditions at once. It's an OR operation, not an AND operation. For instance, if you want to get all patients who had a practitioner with a certain name or from a particular state:

```
GET [your-fhir-server]/Patient?general-practitioner.name=Sarah&general-practitioner.address-state=WA
```

This would return all `Patient` resources that have "Sarah" as the `generalPractitioner`, and all `Patient` resources that have `generalPractitioner` that have the address with the state WA. In other words, you can have Sarah from the state NY and Bill from the state WA both as the returned results. Chained search doesn't require meeting all conditions and is evaluated individually per the parameter.

For scenarios in which the search has to be an AND operation that covers all conditions as a group, refer to the **composite search** example below.

Reverse chain search

Chain search lets you search for resources based on the properties of resources they refer to. Using reverse chain search, allows you do it the other way around. You can search for resources based on the properties of resources that refer to them, using `_has` parameter. For example, `Observation` resource has a search parameter `patient` referring to a Patient resource. To find all Patient resources that are referenced by `Observation` with a specific `code`:

```
GET [base]/Patient?_has:Observation:patient:code=527
```

This request returns Patient resources that are referred by `Observation` with the code `527`.

In addition, reverse chain search can have a recursive structure. For example, if you want to search for all patients that have `Observation` where the observation has an audit event from a specific user `janedoe`, you could do:

```
GET [base]/Patient?_has:Observation:patient:_has:AuditEvent:entity:user=janedoe
```

NOTE

In the Azure API for FHIR and the open-source FHIR server backed by Cosmos, the chained search and reverse chained search is an MVP implementation. To accomplish chained search on Cosmos DB, the implementation walks down the search expression and issues sub-queries to resolve the matched resources. This is done for each level of the expression. If any query returns more than 100 results, an error will be thrown. By default, chained search is behind a feature flag. To use the chained searching on Cosmos DB, use the header `x-ms-enable-chained-search: true`.

Composite search

To search for resources that meet multiple conditions at once, use composite search that joins a sequence of single parameter values with a symbol `$`. The returned result would be the intersection of the resources that match all of the conditions specified by the joined search parameters. Such search parameters are called composite search parameters, and they define a new parameter that combines the multiple parameters in a nested structure. For example, if you want to find all `DiagnosticReport` resources that contain `Observation` with a potassium value less than or equal to 9.2:

```
GET [your-fhir-server]/DiagnosticReport?result.code-value-quantity=2823-3$1t9.2
```

This request specifies the component containing a code of `2823-3`, which in this case would be potassium. Following the `$` symbol, it specifies the range of the value for the component using `1t` for "less than or equal to" and `9.2` for the potassium value range.

Search the next entry set

The maximum number of entries that can be returned per a single search query is 1000. However, you might have more than 1000 entries that match the search query, and you might want to see the next set of entries after the first 1000 entries that were returned. In such case, you would use the continuation token `url` value in `searchset` as in the `Bundle` result below:

```
{
  "resourceType": "Bundle",
  "id": "98731cb7-3a39-46f3-8a72-afe945741bd9",
  "meta": {
    "lastUpdated": "2021-04-22T09:58:16.7823171+00:00"
  },
  "type": "searchset",
  "link": [
    {
      "relation": "next",
      "url": "[your-fhir-server]/Patient?_sort=_lastUpdated&ct=WzUxMDAxNzc1NzgZ0Dc5MjAwODBd"
    },
    {
      "relation": "self",
      "url": "[your-fhir-server]/Patient?_sort=_lastUpdated"
    }
  ],
}
```

And you would do a GET request for the provided URL under the field `relation: next`:

```
GET [your-fhir-server]/Patient?_sort=_lastUpdated&ct=WzUxMDAxNzc1NzgZODc5MjAwODBd
```

This will return the next set of entries for your search result. The `searchset` is the complete set of search result entries, and the continuation token `ur1` is the link provided by the server for you to retrieve the entries that don't show up on the first set because the restriction on the maximum number of entries returned for a search query.

Search using POST

All of the search examples mentioned above have used `GET` requests. You can also do search operations using `POST` requests using `_search`:

```
POST [your-fhir-server]/Patient/_search?_id=45
```

This request would return all `Patient` resources with the `id` value of 45. Just as in `GET` requests, the server determines which of the set of resources meets the condition(s), and returns a bundle resource in the HTTP response.

Another example of searching using `POST` where the query parameters are submitted as a form body is:

```
POST [your-fhir-server]/Patient/_search
content-type: application/x-www-form-urlencoded

name=John
```

Next steps

[Overview of FHIR Search](#)

How to validate FHIR resources against profiles

5/25/2021 • 10 minutes to read • [Edit Online](#)

HL7 FHIR defines a standard and interoperable way to store and exchange healthcare data. Even within the base FHIR specification, it can be helpful to define additional rules or extensions based on the context that FHIR is being used. For such context-specific uses of FHIR, **FHIR profiles** are used for the extra layer of specifications.

FHIR profile describes additional context, such as constraints or extensions, on a resource represented as a `StructureDefinition`. The HL7 FHIR standard defines a set of base resources, and these standard base resources have generic definitions. FHIR profile allows you to narrow down and customize resource definitions using constraints and extensions.

Azure API for FHIR allows validating resources against profiles to see if the resources conform to the profiles. This article walks through the basics of FHIR profile, and how to use `$validate` for validating resources against the profiles when creating and updating resources.

FHIR profile: the basics

A profile sets additional context on the resource, usually represented as a `StructureDefinition` resource. `StructureDefinition` defines a set of rules on the content of a resource or a data type, such as what fields a resource has and what values these fields can take. For example, profiles can restrict cardinality (e.g. setting the maximum cardinality to 0 to rule out the element), restrict the contents of an element to a single fixed value, or define required extensions for the resource. It can also specify additional constraints on an existing profile. A `StructureDefinition` is identified by its canonical URL:

```
http://hl7.org/fhir/StructureDefinition/{profile}
```

Where in the `{profile}` field, you specify the name of the profile.

For example:

- `http://hl7.org/fhir/StructureDefinition/patient-birthPlace` is a base profile that requires information on the registered address of birth of the patient.
- `http://hl7.org/fhir/StructureDefinition/bmi` is another base profile that defines how to represent Body Mass Index (BMI) observations.
- `http://hl7.org/fhir/us/core/StructureDefinition/us-core-allergyintolerance` is a US Core profile that sets minimum expectations for `AllergyIntolerance` resource associated with a patient, and identifies mandatory fields such as extensions and value sets.

Base profile and custom profile

There are two types of profiles: base profile and custom profile. A base profile is a base `StructureDefinition` to which a resource needs to conform to, and has been defined by base resources such as `Patient` or `Observation`. For example, a Body Mass Index (BMI) `Observation` profile would start like this:

```
{
  "resourceType" : "StructureDefinition",
  "id" : "bmi",
  ...
}
```

A custom profile is a set of additional constraints on top of a base profile, restricting or adding resource parameters that are not part of the base specification. Custom profile is useful because you can customize your own resource definitions by specifying the constraints and extensions on the existing base resource. For example, you might want to build a profile that shows `AllergyIntolerance` resource instances based on `Patient` genders, in which case you would create a custom profile on top of an existing `Patient` profile with `AllergyIntolerance` profile.

NOTE

Custom profiles must build on top of the base resource and cannot conflict with the base resource. For example, if an element has a cardinality of 1..1, the custom profile cannot make it optional.

Custom profiles also specified by various Implementation Guides. Some common Implementation Guides are:

NAME	URL
Us Core	https://www.hl7.org/fhir/us/core/
CARIN Blue Button	http://hl7.org/fhir/us/caribb/
Da Vinci Payer Data Exchange	http://hl7.org/fhir/us/davinci-pdex/
Argonaut	http://www.fhir.org/guides/argonaut/pd/

Accessing profiles and storing profiles

Storing profiles

For storing profiles to the server, you can do a `POST` request:

```
POST http://<your FHIR service base URL>/{Resource}
```

In which the field `{Resource}` will be replaced by `StructureDefinition`, and you would have your `StructureDefinition` resource `POST` ed to the server in `JSON` or `XML` format. For example, if you would like to store `us-core-allergyintolerance` profile, you would do:

```
POST http://my-fhir-server.azurewebsites.net/StructureDefinition?url=http://hl7.org/fhir/us/core/StructureDefinition/us-core-allergyintolerance
```

Where the US Core Allergy Intolerance profile would be stored and retrieved:

```
{
  "resourceType" : "StructureDefinition",
  "id" : "us-core-allergyintolerance",
  "text" : {
    "status" : "extensions"
  },
  "url" : "http://hl7.org/fhir/us/core/StructureDefinition/us-core-allergyintolerance",
  "version" : "3.1.1",
  "name" : "USCoreAllergyIntolerance",
  "title" : "US Core AllergyIntolerance Profile",
  "status" : "active",
  "experimental" : false,
  "date" : "2020-06-29",
  "publisher" : "HL7 US Realm Steering Committee",
  "contact" : [
    {
      "telecom" : [
        {
          "system" : "url",
          "value" : "http://www.healthit.gov"
        }
      ]
    }
  ],
  "description" : "Defines constraints and extensions on the AllergyIntolerance resource for the minimal set of data to query and retrieve allergy information.",
  ...
}
```

Most profiles have the resource type `StructureDefinition`, but they can also be of the types `ValueSet` and `CodeSystem`, which are [terminology](#) resources. For example, if you `POST` a `ValueSet` profile in a JSON form, the server will return the stored profile with the assigned `id` for the profile, just as it would with `StructureDefinition`. Below is an example you would get when you upload a [Condition Severity](#) profile, which specifies the criteria for a condition/diagnosis severity grading:

```
{
  "resourceType": "ValueSet",
  "id": "35ab90e5-c75d-45ca-aa10-748fefaca7ee",
  "meta": {
    "versionId": "1",
    "lastUpdated": "2021-05-07T21:34:28.781+00:00",
    "profile": [
      "http://hl7.org/fhir/StructureDefinition/shareablevalueset"
    ]
  },
  "text": {
    "status": "generated"
  },
  "extension": [
    {
      "url": "http://hl7.org/fhir/StructureDefinition/structuredefinition-wg",
      "valueCode": "pc"
    }
  ],
  "url": "http://hl7.org/fhir/ValueSet/condition-severity",
  "identifier": [
    {
      "system": "urn:ietf:rfc:3986",
      "value": "urn:oid:2.16.840.1.113883.4.642.3.168"
    }
  ],
  "version": "4.0.1",
  "name": "Condition/DiagnosisSeverity",
  "title": "Condition/Diagnosis Severity",
  "status": "draft",
  "experimental": false,
  "date": "2019-11-01T09:29:23+11:00",
  "publisher": "FHIR Project team",
  ...
}
```

You can see that the `resourceType` is a `ValueSet`, and the `url` for the profile also specifies that this is a type `ValueSet`: `"http://hl7.org/fhir/ValueSet/condition-severity"`.

Viewing profiles

You can access your existing custom profiles in the server using a `GET` request. All valid profiles, such as the profiles with valid canonical URLs in Implementation Guides, should be accessible by querying:

```
GET http://<your FHIR service base URL>/StructureDefinition?url={canonicalUrl}
```

Where the field `{canonicalUrl}` would be replaced with the canonical URL of your profile.

For example, if you want to view US Core `Goal` resource profile:

```
GET http://my-fhir-server.azurewebsites.net/StructureDefinition?url=http://hl7.org/fhir/us/core/StructureDefinition/us-core-goal
```

This will return the `StructureDefinition` resource for US Core Goal profile, that will start like this:


```
{
  "resourceType" : "StructureDefinition",
  "id" : "us-core-goal",
  "url" : "http://hl7.org/fhir/us/core/StructureDefinition/us-core-goal",
  "version" : "3.1.1",
  "name" : "USCoreGoalProfile",
  "title" : "US Core Goal Profile",
  "status" : "active",
  "experimental" : false,
  "date" : "2020-07-21",
  "publisher" : "HL7 US Realm Steering Committee",
  "contact" : [
    {
      "telecom" : [
        {
          "system" : "url",
          "value" : "http://www.healthit.gov"
        }
      ]
    }
  ],
  "description" : "Defines constraints and extensions on the Goal resource for the minimal set of data to query and retrieve a patient's goal(s).",
  ...
}
```

Our FHIR server does not return `StructureDefinition` instances for the base profiles, but they can be found easily on the HL7 website, such as:

- <http://hl7.org/fhir/Observation.profile.json.html>
- <http://hl7.org/fhir/Patient.profile.json.html>

Profiles in the capability statement

The `Capability Statement` lists all possible behaviors of your FHIR server to be used as a statement of the server functionality, such as Structure Definitions and Value Sets. Azure API for FHIR updates the capability statement with information on the uploaded and stored profiles in the forms of:

- `CapabilityStatement.rest.resource.profile`
- `CapabilityStatement.rest.resource.supportedProfile`

These will show all of the specification for the profile that describes the overall support for the resource, including any constraints on cardinality, bindings, extensions, or other restrictions. Therefore, when you `POST` a profile in the form of a `StructureDefinition`, and `GET` the resource metadata to see the full capability statement, you will see next to the `supportedProfiles` parameter all the details on the profile you uploaded.

For example, if you `POST` a US Core Patient profile, which starts like this:

```
{
  "resourceType": "StructureDefinition",
  "id": "us-core-patient",
  "url": "http://hl7.org/fhir/us/core/StructureDefinition/us-core-patient",
  "version": "3.1.1",
  "name": "USCorePatientProfile",
  "title": "US Core Patient Profile",
  "status": "active",
  "experimental": false,
  "date": "2020-06-27",
  "publisher": "HL7 US Realm Steering Committee",
  ...
}
```

And send a `GET` request for your `metadata` :

```
GET http://<your FHIR service base URL>/metadata
```

You will be returned with a `CapabilityStatement` that includes the following information on the US Core Patient profile you uploaded to your FHIR server:

```
...
{
  "type": "Patient",
  "profile": "http://hl7.org/fhir/StructureDefinition/Patient",
  "supportedProfile": [
    "http://hl7.org/fhir/us/core/StructureDefinition/us-core-patient"
  ],
  ...
}
```

Validating resources against the profiles

FHIR resources, such as `Patient` or `Observation`, can express their conformance to specific profiles. This allows our FHIR server to **validate** given resources against the associated profiles or the specified profiles. Validating a resource against profiles means checking if your resource conforms to the profiles, including the specifications listed in `Resource.meta.profile` or in an Implementation Guide.

There are two ways for you to validate your resource. First, you can use `$validate` operation against a resource that is already in the FHIR server. Second, you can `POST` it to the server as part of a resource `Update` or `Create` operation. In both cases, you can decide via your FHIR server configuration what to do when the resource does not conform to your desired profile.

Using \$validate

The `$validate` operation checks whether the provided profile is valid, and whether the resource conforms to the specified profile. As mentioned in the [HL7 FHIR specifications](#), you can also specify the `mode` for `$validate`, such as `create` and `update`:

- `create`: The server checks that the profile content is unique from the existing resources and that it is acceptable to be created as a new resource
- `update`: checks that the profile is an update against the nominated existing resource (e.g. that no changes are made to the immutable fields)

The server will always return an `OperationOutcome` as the validation results.

Validating an existing resource

To validate an existing resource, use `$validate` in a `GET` request:

```
GET http://<your FHIR service base URL>/{resource}/{resource ID}/$validate
```

For example:

```
GET http://my-fhir-server.azurewebsites.net/Patient/a6e11662-def8-4dde-9ebc-4429e68d130e/$validate
```

In the example above, you would be validating the existing `Patient` resource `a6e11662-def8-4dde-9ebc-4429e68d130e`. If it is valid, you will get an `OperationOutcome` such as the following:

```
{
  "resourceType": "OperationOutcome",
  "issue": [
    {
      "severity": "information",
      "code": "informational",
      "diagnostics": "All OK"
    }
  ]
}
```

If the resource is not valid, you will get an error code and an error message with details on why the resource is invalid. A `4xx` or `5xx` error means that the validation itself could not be performed, and it is unknown whether the resource is valid or not. An example `OperationOutcome` returned with error messages could look like the following:

```
{
  "resourceType": "OperationOutcome",
  "issue": [
    {
      "severity": "error",
      "code": "invalid",
      "details": {
        "coding": [
          {
            "system": "http://hl7.org/fhir/dotnet-api-operation-outcome",
            "code": "1028"
          }
        ],
        "text": "Instance count for 'Patient.identifier.value' is 0, which is not within the specified cardinality of 1..1"
      },
      "location": [
        "Patient.identifier[1]"
      ]
    },
    {
      "severity": "error",
      "code": "invalid",
      "details": {
        "coding": [
          {
            "system": "http://hl7.org/fhir/dotnet-api-operation-outcome",
            "code": "1028"
          }
        ],
        "text": "Instance count for 'Patient.gender' is 0, which is not within the specified cardinality of 1..1"
      },
      "location": [
        "Patient"
      ]
    }
  ]
}
```

In this example above, the resource did not conform to the provided `Patient` profile which required a patient identifier value and gender.

If you would like to specify a profile as a parameter, you can specify the canonical URL for the profile to validate against, such as the following example with US Core `Patient` profile and a base profile for `heartrate`:

```
GET http://<your FHIR service base URL>/<Resource>/<Resource ID>/$validate?profile={canonicalUrl}
```

For example:

```
GET http://my-fhir-server.azurewebsites.net/Patient/a6e11662-def8-4dde-9ebc-4429e68d130e/$validate?
profile=http://hl7.org/fhir/us/core/StructureDefinition/us-core-patient
GET http://my-fhir-server.azurewebsites.net/Observation/12345678/$validate?
profile=http://hl7.org/fhir/StructureDefinition/hearttrate
```

Validating a new resource

If you would like to validate a new resource that you are uploading to the server, you can do a `POST` request:

```
POST http://<your FHIR service base URL>/<Resource>/$validate
```

For example:

```
POST http://my-fhir-server.azurewebsites.net/Patient/$validate
```

This request will create the new resource you are specifying in the request payload, whether it is in a JSON or XML format, and validate the uploaded resource. Then, it will return an `OperationOutcome` as a result of the validation on the new resource.

Validate on resource CREATE or resource UPDATE

You can choose when you would like to validate your resource, such as on resource CREATE or UPDATE. You can specify this in the server configuration setting, under the `CoreFeatures`:

```
{
  "FhirServer": {
    "CoreFeatures": {
      "ProfileValidationOnCreate": true,
      "ProfileValidationOnUpdate": false
    }
  }
}
```

If the resource conforms to the provided `Resource.meta.profile` and the profile is present in the system, the server will act accordingly to the configuration setting above. If the provided profile is not present in the server, the validation request will be ignored and left in `Resource.meta.profile`. Validation is usually an expensive operation, so it is usually run only on test servers or on a small subset of resources - which is why it is important to have these ways to turn the validation operation on or off validation on the server side. If the server configuration specifies to opt out of validation on resource Create/Update, user can override the behavior by specifying it in the `header` of the Create/Update request:

```
x-ms-profile-validation: true
```

Next steps

In this article, you have learned about FHIR profiles, and how to validate resources against profiles using `$validate`. To learn about Azure API for FHIR's other supported features, check out:

[FHIR supported features](#)

How to export FHIR data

5/25/2021 • 6 minutes to read • [Edit Online](#)

The Bulk Export feature allows data to be exported from the FHIR Server per the [FHIR specification](#).

Before using \$export, you'll want to make sure that the Azure API for FHIR is configured to use it. For configuring export settings and creating Azure storage account, refer to [the configure export data page](#).

Using \$export command

After configuring the Azure API for FHIR for export, you can use the \$export command to export the data out of the service. The data will be stored into the storage account you specified while configuring export. To learn how to invoke \$export command in FHIR server, read documentation on the [HL7 FHIR \\$export specification](#).

Jobs stuck in a bad state

In some situations, there is a potential for a job to be stuck in a bad state. This can occur especially if the storage account permissions have not been setup properly. One way to validate if your export is successful is to check your storage account to see if the corresponding container (that is, ndjson) files are present. If they are not present, and there are no other export jobs running, then there is a possibility the current job is stuck in a bad state. You should cancel the export job by sending a cancellation request and try re-queuing the job again. Our default run time for an export in bad state is 10 minutes before it will stop and move to a new job or retry the export.

The Azure API For FHIR supports \$export at the following levels:

- **System:** `GET https://<<FHIR service base URL>>/$export>>`
- **Patient:** `GET https://<<FHIR service base URL>>/Patient/$export>>`
- **Group of patients*** - Azure API for FHIR exports all related resources but doesn't export the characteristics of the group: `GET https://<<FHIR service base URL>>/Group/[ID]/$export>>`

When data is exported, a separate file is created for each resource type. To ensure that the exported files don't become too large. We create a new file after the size of a single exported file becomes larger than 64 MB. The result is that you may get multiple files for each resource type, which will be enumerated (that is, Patient-1.ndjson, Patient-2.ndjson).

NOTE

`Patient/$export` and `Group/[ID]/$export` may export duplicate resources if the resource is in a compartment of more than one resource, or is in multiple groups.

In addition, checking the export status through the URL returned by the location header during the queuing is supported along with canceling the actual export job.

Exporting FHIR data to ADLS Gen2

Currently we support \$export for ADLS Gen2 enabled storage accounts, with the following limitation:

- User cannot take advantage of [hierarchical namespaces](#), yet there isn't a way to target export to a specific subdirectory within the container. We only provide the ability to target a specific container (where we create a new folder for each export).
- Once an export is complete, we never export anything to that folder again, since subsequent exports to the

same container will be inside a newly created folder.

Settings and parameters

Headers

There are two required header parameters that must be set for \$export jobs. The values are defined by the current [\\$export specification](#).

- **Accept** - application/fhir+json
- **Prefer** - respond-async

Query parameters

The Azure API for FHIR supports the following query parameters. All of these parameters are optional:

QUERY PARAMETER	DEFINED BY THE FHIR SPEC?	DESCRIPTION
_outputFormat	Yes	Currently supports three values to align to the FHIR Spec: application/fhir+ndjson, application/ndjson, or just ndjson. All export jobs will return <code>ndjson</code> and the passed value has no effect on code behavior.
_since	Yes	Allows you to only export resources that have been modified since the time provided
_type	Yes	Allows you to specify which types of resources will be included. For example, _type=Patient would return only patient resources
_typefilter	Yes	To request finer-grained filtering, you can use _typefilter along with the _type parameter. The value of the _typeFilter parameter is a comma-separated list of FHIR queries that further restrict the results
_container	No	Specifies the container within the configured storage account where the data should be exported. If a container is specified, the data will be exported into a folder into that container. If the container is not specified, the data will be exported to a new container.

NOTE

Only storage accounts in the same subscription as that for Azure API for FHIR are allowed to be registered as the destination for \$export operations.

Secure Export to Azure Storage

Azure API for FHIR supports a secure export operation. Choose one of the two options below:

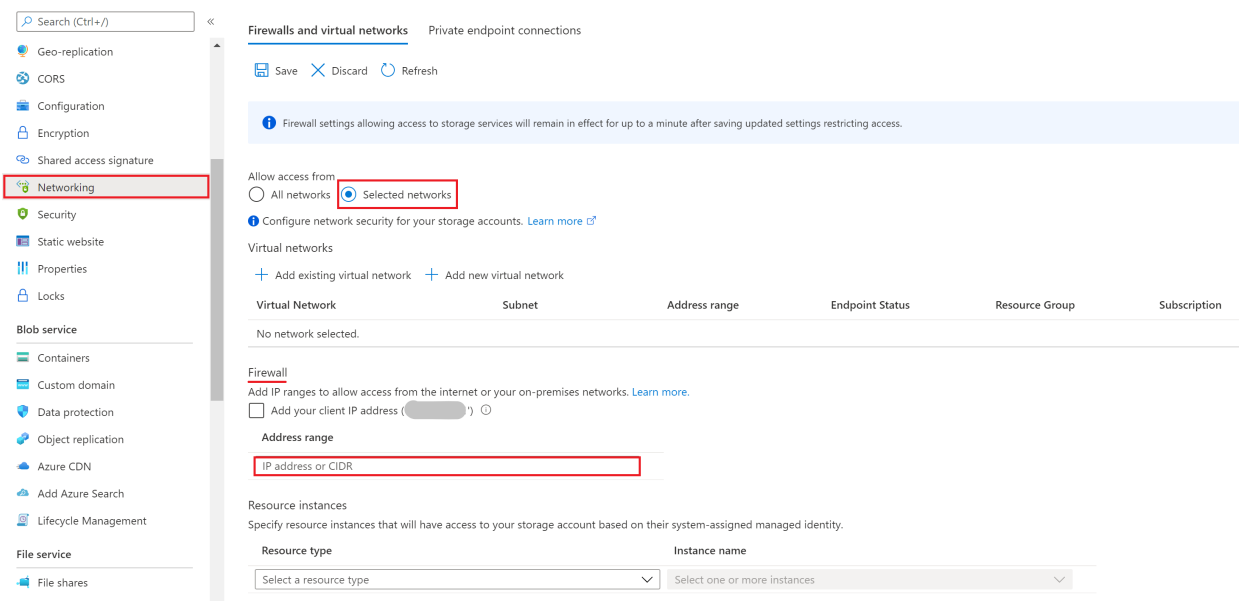
- Allowing Azure API for FHIR as a Microsoft Trusted Service to access the Azure storage account.
- Allowing specific IP addresses associated with Azure API for FHIR to access the Azure storage account. This option provides two different configurations depending on whether the storage account is in the same location as, or is in a different location from that of the Azure API for FHIR.

Allowing Azure API for FHIR as a Microsoft Trusted Service

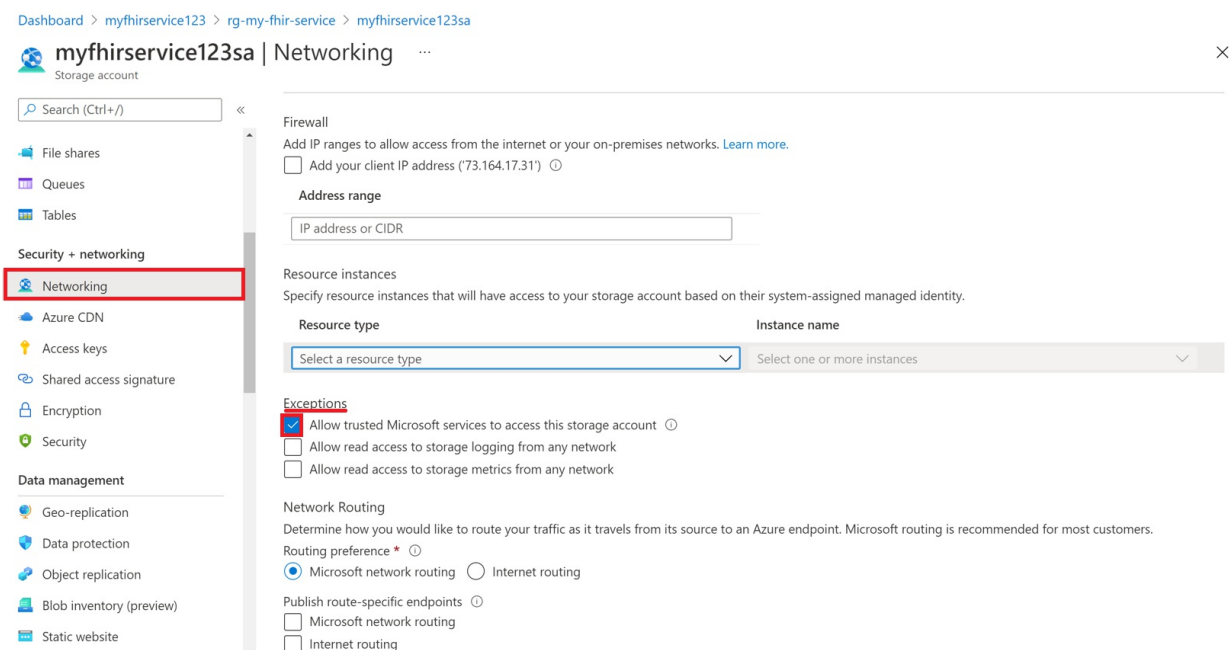
Select a storage account from the Azure portal, and then select the **Networking** blade. Select **Selected networks** under the **Firewalls and virtual networks** tab.

IMPORTANT

Ensure that you've granted access permission to the storage account for Azure API for FHIR using its managed identity. For more details, see [Configure export setting and set up the storage account](#).



Under the **Exceptions** section, select the box **Allow trusted Microsoft services to access this storage account** and save the setting.



You're now ready to export FHIR data to the storage account securely. Note that the storage account is on selected networks and is not publicly accessible. To access the files, you can either enable and use private

endpoints for the storage account, or enable all networks for the storage account for a short period of time.

IMPORTANT

The user interface will be updated later to allow you to select the Resource type for Azure API for FHIR and a specific service instance.

Allowing specific IP addresses for the Azure storage account in a different region

Select **Networking** of the Azure storage account from the portal.

Select **Selected networks**. Under the Firewall section, specify the IP address in the **Address range** box. Add IP ranges to allow access from the internet or your on-premises networks. You can find the IP address in the table below for the Azure region where the Azure API for FHIR service is provisioned.

AZURE REGION	PUBLIC IP ADDRESS
Australia East	20.53.44.80
Canada Central	20.48.192.84
Central US	52.182.208.31
East US	20.62.128.148
East US 2	20.49.102.228
East US 2 EUAP	20.39.26.254
Germany North	51.116.51.33
Germany West Central	51.116.146.216
Japan East	20.191.160.26
Korea Central	20.41.69.51
North Central US	20.49.114.188
North Europe	52.146.131.52
South Africa North	102.133.220.197
South Central US	13.73.254.220
Southeast Asia	23.98.108.42
Switzerland North	51.107.60.95
UK South	51.104.30.170
UK West	51.137.164.94
West Central US	52.150.156.44

AZURE REGION	PUBLIC IP ADDRESS
West Europe	20.61.98.66
West US 2	40.64.135.77

NOTE

The above steps are similar to the configuration steps described in the document [How to convert data to FHIR \(Preview\)](#). For more information, see [Host and use templates](#)

Allowing specific IP addresses for the Azure storage account in the same region

The configuration process is the same as above except a specific IP address range in CIDR format is used instead, 100.64.0.0/10. The reason why the IP address range, which includes 100.64.0.0 – 100.127.255.255, must be specified is because the actual IP address used by the service varies, but will be within the range, for each \$export request.

NOTE

It is possible that a private IP address within the range of 10.0.2.0/24 may be used instead. In that case, the \$export operation will not succeed. You can retry the \$export request, but there is no guarantee that an IP address within the range of 100.64.0.0/10 will be used next time. That's the known networking behavior by design. The alternative is to configure the storage account in a different region.

Next steps

In this article, you've learned how to export FHIR resources using \$export command. Next, to learn how to export de-identified data, see:

[Export de-identified data](#)

Exporting de-identified data (preview)

6/8/2021 • 2 minutes to read • [Edit Online](#)

NOTE

Results when using the de-identified export will vary based on factors such as data inputted, and functions selected by the customer. Microsoft is unable to evaluate the de-identified export outputs or determine the acceptability for customer's use cases and compliance needs. The de-identified export is not guaranteed to meet any specific legal, regulatory, or compliance requirements.

The \$export command can also be used to export de-identified data from the FHIR server. It uses the anonymization engine from [FHIR tools for anonymization](#), and takes anonymization config details in query parameters. You can create your own anonymization config file or use the [sample config file](#) for HIPAA Safe Harbor method as a starting point.

```
https://<<FHIR service base URL>>/$export?_container=<<container_name>>&_anonymizationConfig=<<config file name>>&_anonymizationConfigEtag=<<ETag on storage>>
```

NOTE

Right now the Azure API for FHIR only supports de-identified export at the system level (\$export).

QUERY PARAMETER	EXAMPLE	OPTIONALITY	DESCRIPTION
<i>_anonymizationConfig</i>	DemoConfig.json	Required for de-identified export	Name of the configuration file. See the configuration file format here . This file should be kept inside a container named anonymization within the same Azure storage account that is configured as the export location.
<i>_anonymizationConfigEtag</i>	"0x8D8494A069489EC"	Optional for de-identified export	This is the Etag of the configuration file. You can get the Etag using Azure Storage Explorer from the blob property

IMPORTANT

Both raw export as well as de-identified export writes to the same Azure storage account specified as part of export configuration. It is recommended that you use different containers corresponding to different de-identified config and manage user access at the container level.

Moving data from Azure API for FHIR to Azure Synapse Analytics

5/25/2021 • 9 minutes to read • [Edit Online](#)

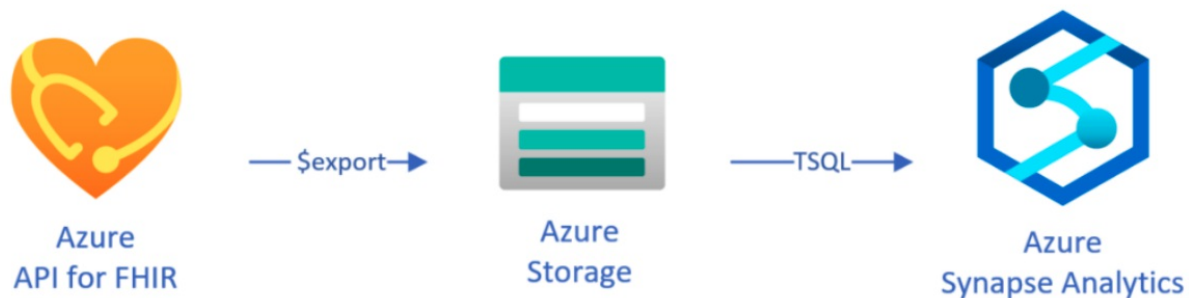
In this article you will learn a couple of ways to move data from Azure API for FHIR to [Azure Synapse Analytics](#), which is a limitless analytics service that brings together data integration, enterprise data warehousing, and big data analytics.

Moving data from the FHIR server to Synapse involves exporting the data using the FHIR `$export` operation followed by a series of steps to transform and load the data to Synapse. This article will walk you through two of the several approaches, both of which will show how to convert FHIR resources into tabular formats while moving them into Synapse.

- **Load exported data to Synapse using T-SQL:** Use `$export` operation to move FHIR resources into a **Azure Data Lake Gen 2 (ADL Gen 2) blob storage** in `NDJSON` format. Load the data from the storage into **serverless or dedicated SQL pools** in Synapse using T-SQL. Convert these steps into a robust data movement pipeline using [Synapse pipelines](#).
- **Use the tools from the FHIR Analytics Pipelines OSS repo:** The [FHIR Analytics Pipeline](#) repo contains tools that can create an **Azure Data Factory (ADF) pipeline** to move FHIR data into a **Common Data Model (CDM)** folder, and from the CDM folder to Synapse.

Load exported data to Synapse using T-SQL

`$export` **for moving FHIR data into Azure Data Lake Gen 2 storage**



Configure your FHIR server to support `$export`

Azure API for FHIR implements the `$export` operation defined by the FHIR specification to export all or a filtered subset of FHIR data in `NDJSON` format. In addition, it supports [de-identified export](#) to anonymize FHIR data during the export. If you use `$export`, you get de-identification feature by default its capability is already integrated in `$export`.

To export FHIR data to Azure blob storage, you first need to configure your FHIR server to export data to the storage account. You will need to (1) enable Managed Identity, (2) go to Access Control in the storage account and add role assignment, (3) select your storage account for `$export`. More step-by-step can be found [here](#).

You can configure the server to export the data to any kind of Azure storage account, but we recommend exporting to ADL Gen 2 for best alignment with Synapse.

Using `$export` command

After configuring your FHIR server, you can follow the [documentation](#) to export your FHIR resources at System, Patient, or Group level. For example, you can export all of your FHIR data related to the patients in a `Group` with

the following `$export` command, in which you specify your ADL Gen 2 blob storage name in the field `{{BlobContainer}}`:

```
https://{{FHIR service base URL}}/Group/{{GroupId}}/$export?_container={{BlobContainer}}
```

You can also use `_type` parameter in the `$export` call above to restrict the resources we you want to export. For example, the following call will export only `Patient`, `MedicationRequest`, and `Observation` resources:

```
https://{{FHIR service base URL}}/Group/{{GroupId}}/$export?_container={{BlobContainer}}&_type=Patient,MedicationRequest,Condition
```

For more information on the different parameters supported, check out our `$export` page section on the [query parameters](#).

Create a Synapse workspace

Before using Synapse, you will need a Synapse workspace. You will create a Azure Synapse Analytics service on Azure portal. More step-by-step guide can be found [here](#). You need an `ADLSGEN2` account to create a workspace. Your Azure Synapse workspace will use this storage account to store your Synapse workspace data.

After creating a workspace, you can view your workspace on Synapse Studio by signing into your workspace on <https://web.azuresynapse.net>, or launching Synapse Studio in the Azure portal.

Creating a linked service between Azure storage and Synapse

To move your data to Synapse, you need to create a linked service that connects your Azure Storage account with Synapse. More step-by-step can be found [here](#).

1. On Synapse Studio, navigate to the **Manage** tab, and under **External connections**, select **Linked services**.
2. Select **New** to add a new linked service.
3. Select **Azure Data Lake Storage Gen2** from the list and select **Continue**.
4. Enter your authentication credentials. Select **Create** when finished.

Now that you have a linked service between your ADL Gen 2 storage and Synapse, you are ready to use Synapse SQL pools to load and analyze your FHIR data.

Decide between serverless and dedicated SQL pool

Azure Synapse Analytics offers two different SQL pools, serverless SQL pool and dedicated SQL pool. Serverless SQL pool gives the flexibility of querying data directly in the blob storage using the serverless SQL endpoint without any resource provisioning. Dedicated SQL pool has the processing power for high performance and concurrency, and is recommended for enterprise-scale data warehousing capabilities. For more details on the two SQL pools, check out the [Synapse documentation page](#) on SQL architecture.

Using serverless SQL pool

Since it is serverless, there's no infrastructure to setup or clusters to maintain. You can start querying data from Synapse Studio as soon as the workspace is created.

For example, the following query can be used to transform selected fields from `Patient.ndjson` into a tabular structure:

```

SELECT * FROM
OPENROWSET(bulk 'https://{youraccount}.blob.core.windows.net/{yourcontainer}/Patient.ndjson',
FORMAT = 'csv',
FIELDTERMINATOR = '0x0b',
FIELDQUOTE = '0x0b')
WITH (doc NVARCHAR(MAX)) AS rows
CROSS APPLY OPENJSON(doc)
WITH (
    ResourceId VARCHAR(64) '$.id',
    Active VARCHAR(10) '$.active',
    FullName VARCHAR(100) '$.name[0].text',
    Gender VARCHAR(20) '$.gender',
    ...
)

```

In the query above, the `OPENROWSET` function accesses files in Azure Storage, and `OPENJSON` parses JSON text and returns the JSON input properties as rows and columns. Every time this query is executed, the serverless SQL pool reads the file from the blob storage, parses the JSON, and extracts the fields.

You can also materialize the results in Parquet format in an [External Table](#) to get better query performance, as shown below:

```

-- Create External data source where the parquet file will be written
CREATE EXTERNAL DATA SOURCE [MyDataSource] WITH (
    LOCATION = 'https://{youraccount}.blob.core.windows.net/{exttblcontainer}'
);
GO

-- Create External File Format
CREATE EXTERNAL FILE FORMAT [ParquetFF] WITH (
    FORMAT_TYPE = PARQUET,
    DATA_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'
);
GO

CREATE EXTERNAL TABLE [dbo].[Patient] WITH (
    LOCATION = 'PatientParquet/',
    DATA_SOURCE = [MyDataSource],
    FILE_FORMAT = [ParquetFF]
) AS
SELECT * FROM
OPENROWSET(bulk 'https://{youraccount}.blob.core.windows.net/{yourcontainer}/Patient.ndjson'
-- Use rest of the SQL statement from the previous example --

```

Using dedicated SQL pool

Dedicated SQL pool supports managed tables and a hierarchical cache for in-memory performance. You can import big data with simple T-SQL queries, and then use the power of the distributed query engine to run high-performance analytics.

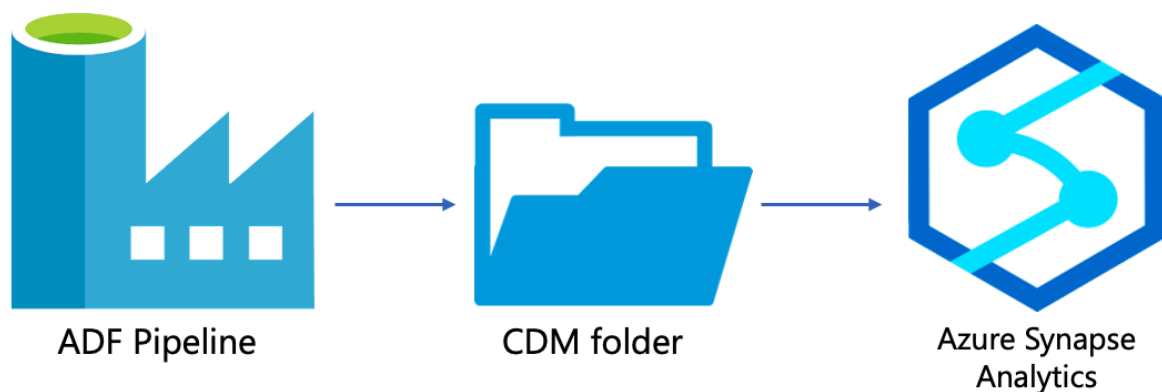
The simplest and fastest way to load data from your storage to a dedicated SQL pool is to use the `COPY` command in T-SQL, which can read CSV, Parquet, and ORC files. As in the example query below, use the `COPY` command to load the `NDJSON` rows into a tabular structure.

```
-- Create table with HEAP, which is not indexed and does not have a column width limitation of
NVARCHAR(4000)
CREATE TABLE StagingPatient (
Resource NVARCHAR(MAX)
) WITH (HEAP)
COPY INTO StagingPatient
FROM 'https://{{yourblobaccount}}.blob.core.windows.net/{{yourcontainer}}/Patient.ndjson'
WITH (
FILE_TYPE = 'CSV',
ROWTERMINATOR='0x0a',
FIELDQUOTE = '',
FIELDTERMINATOR = '0x00'
)
GO
```

Once you have the JSON rows in the `StagingPatient` table above, you can create different tabular formats of the data using the `OPENJSON` function and storing the results into tables. Here is a sample SQL query to create a `Patient` table by extracting a few fields from the `Patient` resource:

```
SELECT RES.*
INTO Patient
FROM StagingPatient
CROSS APPLY OPENJSON(Resource)
WITH (
ResourceId VARCHAR(64) '$.id',
FullName VARCHAR(100) '$.name[0].text',
FamilyName VARCHAR(50) '$.name[0].family',
GivenName VARCHAR(50) '$.name[0].given[0]',
Gender VARCHAR(20) '$.gender',
DOB DATETIME2 '$.birthDate',
MaritalStatus VARCHAR(20) '$.maritalStatus.coding[0].display',
LanguageOfCommunication VARCHAR(20) '$.communication[0].language.text'
) AS RES
GO
```

Use FHIR Analytics Pipelines OSS tools



NOTE

[FHIR Analytics pipeline](#) is an open source tool released under MIT license, and is not covered by the Microsoft SLA for Azure services.

ADF pipeline for moving FHIR data into CDM folder

Common Data Model (CDM) folder is a folder in a data lake that conforms to well-defined and standardized metadata structures and self-describing data. These folders facilitate metadata interoperability between data producers and data consumers. Before you move FHIR data into CDM folder, you can transform your data into a table configuration.

Generating table configuration

Clone the repo to get all the scripts and source code. Use `npm install` to install the dependencies. Run the following command from the `Configuration-Generator` folder to generate a table configuration folder using YAML format instructions:

```
Configuration-Generator> node .\generate_from_yaml.js -r {resource configuration file} -p {properties group file} -o {output folder}
```

You may use the sample `YAML` files, `resourcesConfig.yml` and `propertiesGroupConfig.yml` provided in the repo.

Generating ADF pipeline

Now you can use the content of the generated table configuration and a few other configurations to generate an ADF pipeline. This ADF pipeline, when triggered, exports the data from the FHIR server using `$export` API and writes to a CDM folder along with associated CDM metadata.

1. Create an Azure Active Directory (AD) application and service principal. The ADF pipeline uses an Azure batch service to do the transformation, and needs an Azure AD application for the batch service. Follow [Azure AD documentation](#).
2. Grant access for export storage location to the service principal. In the `Access Control` of the export storage, grant `Storage Blob Data Contributor` role to the Azure AD application.
3. Deploy the egress pipeline. Use the template `fhirServiceToCdm.json` for a custom deployment on Azure. This step will create the following Azure resources:
 - An ADF pipeline with the name `{pipelinename}-df`.
 - A key vault with the name `{pipelinename}-kv` to store the client secret.
 - A batch account with the name `{pipelinename}batch` to run the transformation.
 - A storage account with the name `{pipelinename}storage`.
4. Grant access to the Azure Data Factory. In the access control panel of your FHIR service, grant `FHIR data exporter` and `FHIR data reader` roles to the data factory, `{pipelinename}-df`.
5. Upload the content of the table configuration folder to the configuration container.
6. Go to `{pipelinename}-df`, and trigger the pipeline. You should see the exported data in the CDM folder on the storage account `{pipelinename}storage`. You should see one folder for each table having a CSV file.

From CDM folder to Synapse

Once you have the data exported in a CDM format and stored in your ADL Gen 2 storage, you can now move your data in the CDM folder to Synapse.

You can create CDM to Synapse pipeline using a configuration file, which would look something like this:

```
{
  "ResourceGroup": "",
  "TemplateFilePath": "../Templates/cdmToSynapse.json",
  "TemplateParameters": {
    "DataFactoryName": "",
    "SynapseWorkspace": "",
    "DedicatedSqlPool": "",
    "AdlsAccountForCdm": "",
    "CdmRootLocation": "cdm",
    "StagingContainer": "adfstaging",
    "Entities": ["LocalPatient", "LocalPatientAddress"]
  }
}
```

Run this script with the configuration file above:

```
.\DeployCdmToSynapsePipeline.ps1 -Config: config.json
```

Add ADF Managed Identity as a SQL user into SQL database. Here is a sample SQL script to create a user and an assign role:

```
CREATE USER [datafactory-name] FROM EXTERNAL PROVIDER
GO
EXEC sp_addrolemember db_owner, [datafactory-name]
GO
```

Next steps

In this article, you learned two different ways to move your FHIR data into Synapse: (1) using `$export` to move data into ADL Gen 2 blob storage then loading the data into Synapse SQL pools, and (2) using ADF pipeline for moving FHIR data into CDM folder then into Synapse.

Next, you can learn about anonymization of your FHIR data while moving data to Synapse to ensure your healthcare information is protected:

[Exporting de-identified data](#)

How to convert data to FHIR (Preview)

5/17/2021 • 4 minutes to read • [Edit Online](#)

IMPORTANT

This capability is in public preview, and it's provided without a service level agreement. It's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

The \$convert-data custom endpoint in the Azure API for FHIR is meant for data conversion from different formats to FHIR. It uses the Liquid template engine and the templates from the [FHIR Converter](#) project as the default templates. You can customize these conversion templates as needed. Currently it supports HL7v2 to FHIR conversion.

Use the \$convert-data endpoint

```
https://<<FHIR service base URL>>/$convert-data
```

\$convert-data takes a [Parameter](#) resource in the request body as described below:

Parameter Resource:

PARAMETER NAME	DESCRIPTION	ACCEPTED VALUES
inputData	Data to be converted.	A valid value of JSON String datatype
inputDataType	Data type of input.	<code>HL7v2</code>
templateCollectionReference	Reference to a template collection. It can be a reference either to the Default templates , or a custom template image that's registered with Azure API for FHIR. See below to learn about customizing the templates, hosting those on ACR, and registering to the Azure API for FHIR.	<code>microsofthealth/fhirconverter:default</code> , <RegistryServer>/<imageName>@<imageDigest>
rootTemplate	The root template to use while transforming the data.	<code>ADT_A01</code> , <code>OML_021</code> , <code>ORU_R01</code> , <code>VXU_V04</code>

WARNING

Default templates help you get started quickly. However, these may get updated when we upgrade the Azure API for FHIR. In order to have consistent data conversion behavior across different versions of Azure API for FHIR, you must host your own copy of templates on an Azure Container Registry, register those to the Azure API for FHIR, and use in your API calls as described later.

Sample request:

```
{
  "resourceType": "Parameters",
  "parameter": [
    {
      "name": "inputData",
      "valueString":
"MSH|^~\\&|SIMHOSP|SFAC|RAPP|RFAC|20200508131015||ADT^A01|517|T|2.3|||AL||44|ASCII\\nEVN|A01|20200508131015||
|C005^Whittingham^Sylvia^^^Dr^^^DRNBR^PRSNL^^^ORGDR|\\nPID|1|3735064194^^^SIMULATOR
MRN^MRN|3735064194^^^SIMULATOR
MRN^MRN~2021051528^^^NHSNBR^NHSNMBR||Kinmonth^Joanna^Chelsea^Ms^^CURRENT||19870624000000|F|||89 Transaction
House^Handmaiden Street^Wembley^FV75 4GJ^GBR^HOME||020 3614 5541^HOME|||||C^White -
Other^^^|||||\\nPD1|||FAMILY PRACTICE^^12345|\\nPV1|1|I|OtherWard^MainRoom^Bed 183^Simulated
Hospital^^BED^Main
Building^4|28b|||C005^Whittingham^Sylvia^^^Dr^^^DRNBR^PRSNL^^^ORGDR|||CAR|||||||16094728916771313876^^^vi
sitid|||||||||||||ARRIVED||20200508131015||"
    },
    {
      "name": "inputDataType",
      "valueString": "H17v2"
    },
    {
      "name": "templateCollectionReference",
      "valueString": "microsofthealth/fhirconverter:default"
    },
    {
      "name": "rootTemplate",
      "valueString": "ADT_A01"
    }
  ]
}
```

Sample response:

```
{
  "resourceType": "Bundle",
  "type": "transaction",
  "entry": [
    {
      "fullUrl": "urn:uuid:9d697ec3-48c3-3e17-db6a-29a1765e22c6",
      "resource": {
        "resourceType": "Patient",
        "id": "9d697ec3-48c3-3e17-db6a-29a1765e22c6",
        ...
      },
      "request": {
        "method": "PUT",
        "url": "Location/50becdb5-ff56-56c6-40a1-6d554dca80f0"
      }
    }
  ]
}
```

Customize templates

You can use the [FHIR Converter extension](#) for Visual Studio Code to customize the templates as per your needs. The extension provides an interactive editing experience, and makes it easy to download Microsoft-published templates and sample data. Refer to the documentation in the extension for more details.

Host and use templates

It's strongly recommended that you host your own copy of templates on ACR. There're four steps involved in hosting your own copy of templates and using those in the \$convert-data operation:

1. Push the templates to your Azure Container Registry.
2. Enable Managed Identity on your Azure API for FHIR instance.
3. Provide access of the ACR to the Azure API for FHIR Managed Identity.
4. Register the ACR servers in the Azure API for FHIR.
5. Optionally configure ACR firewall for secure access.

Push templates to Azure Container Registry

After creating an ACR instance, you can use the *FHIR Converter: Push Templates* command in the [FHIR Converter extension](#) to push the customized templates to the ACR. Alternatively, you can use the [Template Management CLI tool](#) for this purpose.

Enable Managed Identity on Azure API for FHIR

Browse to your instance of Azure API for FHIR service in the Azure portal, and then select the **Identity** blade. Change the status to **On** to enable managed identity in Azure API for FHIR.

fhir-docs - Identity
Azure API for FHIR

Search (Cmd+/) <<

Overview
Activity log
Access control (IAM)
Tags

Settings

Authentication
CORS
Database
Integration
Identity
Locks
Export template

Monitoring

Metrics
Diagnostic settings

System assigned

A system assigned managed identity enables Azure resources to authenticate to cloud services (e.g. Azure Key Vault) without storing credentials in code. Once enabled, all necessary permissions can be granted via Azure role-based-access-control. The lifecycle of this type of managed identity is tied to the lifecycle of this resource. Additionally, each resource (e.g. Virtual Machine) can only have one system assigned managed identity. [Learn more about Managed identities.](#)

Save Discard Refresh Got feedback?

Status ⓘ
Off On

Object ID ⓘ
1d330b79-cc4e-418d-8e2b-b7171c86cfe7

Role assignments ⓘ
[Show the Azure RBAC roles assigned to this managed identity](#)

i This resource is registered with Azure Active Directory. You can control its access to services like Azure Resource Manager, Azure Key Vault, etc. [Learn more](#)

Provide access of the ACR to Azure API for FHIR

1. Browse to the **Access control (IAM)** blade.
2. Select **Add**, and then select **Add role assignment** to open the Add role assignment page.
3. Assign the [AcrPull](#) role.

For more information about assigning roles in the Azure portal, see [Azure built-in roles](#).

Register the ACR servers in Azure API for FHIR

You can register the ACR server using the Azure portal, or using CLI.

Registering the ACR server using Azure portal

Browse to the **Artifacts** blade under **Data transformation** in your Azure API for FHIR instance. You will see the list of currently registered ACR servers. Select **Add**, and then select your registry server from the drop-down menu. You'll need to select **Save** for the registration to take effect. It may take a few minutes to apply the change and restart your instance.

Registering the ACR server using CLI

You can register up to 20 ACR servers in the Azure API for FHIR.

Install the Healthcare APIs CLI from Azure PowerShell if needed:

```
az extension add -n healthcareapis
```

Register the acr servers to Azure API for FHIR following the examples below:

Register a single ACR server

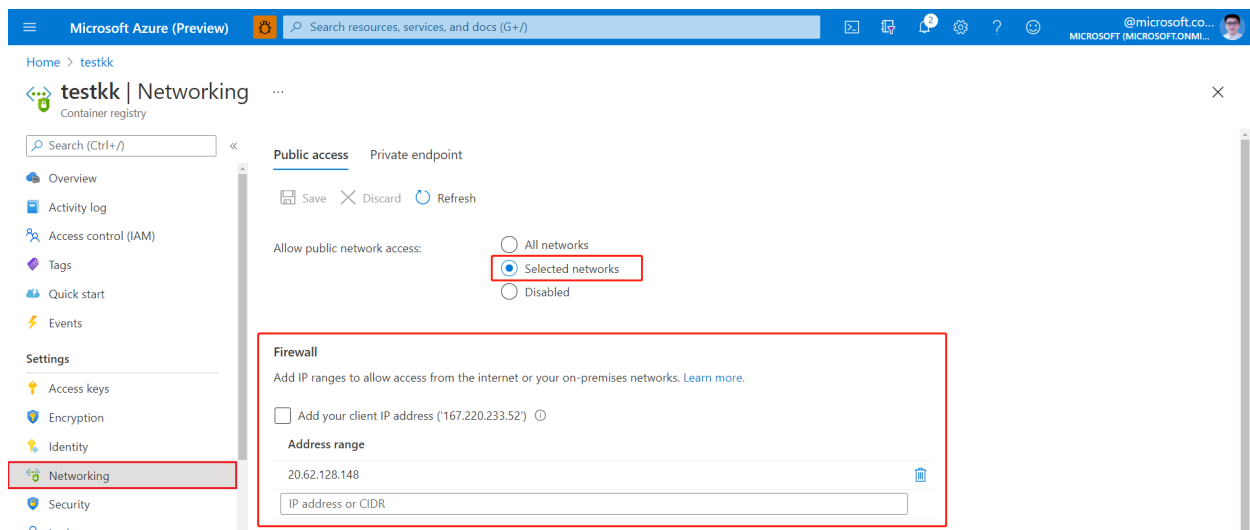
```
az healthcareapis acr add --login-servers "fhiracr2021.azurecr.io" --resource-group fhir-test --resource-name fhirtest2021
```

Register multiple ACR servers

```
az healthcareapis acr add --login-servers "fhiracr2021.azurecr.io fhiracr2020.azurecr.io" --resource-group fhir-test --resource-name fhirtest2021
```

Configure ACR firewall

Select **Networking** of the Azure storage account from the portal.



Select **Selected networks**.

Under the **Firewall** section, specify the IP address in the **Address range** box. Add IP ranges to allow access from the internet or your on-premises networks.

In the table below, you'll find the IP address for the Azure region where the Azure API for FHIR service is provisioned.

AZURE REGION	PUBLIC IP ADDRESS
Australia East	20.53.44.80
Canada Central	20.48.192.84
Central US	52.182.208.31
East US	20.62.128.148
East US 2	20.49.102.228
East US 2 EUAP	20.39.26.254
Germany North	51.116.51.33
Germany West Central	51.116.146.216
Japan East	20.191.160.26
Korea Central	20.41.69.51

AZURE REGION	PUBLIC IP ADDRESS
North Central US	20.49.114.188
North Europe	52.146.131.52
South Africa North	102.133.220.197
South Central US	13.73.254.220
Southeast Asia	23.98.108.42
Switzerland North	51.107.60.95
UK South	51.104.30.170
UK West	51.137.164.94
West Central US	52.150.156.44
West Europe	20.61.98.66
West US 2	40.64.135.77

NOTE

The above steps are similar to the configuration steps described in the document [How to export FHIR data](#). For more information, see [Secure Export to Azure Storage](#)

Verify

Make a call to the \$convert-data API specifying your template reference in the templateCollectionReference parameter.

```
<RegistryServer>/<imageName>@<imageDigest>
```

Patient-everything in FHIR

6/8/2021 • 2 minutes to read • [Edit Online](#)

The [\\$patient-everything](#) operation was created to provide a patient with access to their entire record or for a provider or other user to perform a bulk data download. This operation is used to return all the information related to one or more patients described in the resource or context on which this operation is invoked.

Use patient-everything

To call patient-everything, use the following command:

```
GET {FHIRURL}/Patient/{ID}/$everything
```

The Azure API for FHIR validates that it can find the patient matching the provided patient ID. If a result is found, the response will be a bundle of type "searchset" with the following information:

- [Patient resource](#)
- Resources that are directly referenced by the Patient resource (except link)
- Resources in the Patient's [compartment](#)
- [Device resources](#) that reference the Patient resource. Note that this is limited to 100 devices. If the patient has more than 100 devices linked to them, only 100 will be returned.

NOTE

\$patient-everything is available in the Open Source FHIR Server backed by Cosmos DB now and will be available in Azure API for FHIR before July 1st. The capability statement for the FHIR Server is missing support for \$patient-everything, which is tracked here: Issue [1989](#).

Patient-everything parameters

The Azure API for FHIR supports the following query parameters. All of these parameters are optional:

QUERY PARAMETER	DESCRIPTION
<code>_type</code>	Allows you to specify which types of resources will be included in the response. For example, <code>_type=Encounter</code> would return only <code>Encounter</code> resources associated with the patient.
<code>_since</code>	Will return only resources that have been modified since the time provided.
<code>start</code>	Specifying the start date will pull in resources where there clinical date is after the specified start date. If no start date is provided, all records prior to the end date are in scope.
<code>end</code>	Specifying the end date will pull in resources where there clinical date is before the specified end date. If no end date is provided, all records after the start date are in scope.

NOTE

You must specify an ID for a specific patient. If you need all data for all patients, see [\\$export](#).

Examples of \$patient-everything

Below are some additional examples of using the \$patient-everything operation.

To use \$patient-everything to query a patient's "everything" between 2010 and 2020, use the following call:

```
GET {FHIRURL}/Patient/{ID}/$everything?start=2010&end=2020
```

To use \$patient-everything to query a patient's Observation and Encounter, use the following call:

```
GET {FHIRURL}/Patient/{ID}/$everything_type=Observation,Encounter
```

To use \$patient-everything to query a patient's "everything" since 2021-05-27T05:00:00Z, use the following call:

```
GET {FHIRURL}/Patient/{ID}/$everything?_since=2021-05-27T05:00:00Z
```

If a Patient is found for each of these calls, you'll get back a 200 response with a Bundle of the corresponding resources.

Next step

Now that you know how to use the patient-everything operation, you can learn about more search options on the overview of search guide.

[Overview of FHIR search](#)

\$member-match operation

6/8/2021 • 2 minutes to read • [Edit Online](#)

[\\$member-match](#) is an operation that is defined as part of the Da Vinci Health Record Exchange (HREx). In this guide, we'll walk through what \$member-match is and how to use it.

Overview of \$member-match

The \$member-match operation was created to help with the payer-to-payer data exchange, by allowing a new payer to get a unique identifier for a patient from the patient's previous payer. The \$member-match operation requires three pieces of information to be passed in the body of the request:

- Patient demographics
- The old coverage information
- The new coverage information (not required based on our implementation)

After the data is passed in, the Azure API for FHIR validates that it can find a patient that exactly matches the demographics passed in with the old coverage information passed in. If a result is found, the response will be a bundle with the original patient data plus a new identifier added in from the old payer, and the old coverage information.

NOTE

The specification describes passing in and back the new coverage information. We've decided to omit that data to keep the results smaller.

Example of \$member-match

To use \$member-match, use the following call:

```
POST {{fhirurl}}/Patient/$member-match
```

You'll need to include a parameters resource in the body that includes the patient, the old coverage, and the new coverage. To see a JSON representation, see [\\$member-match example request](#).

If a single match is found, you'll receive a 200 response with another identifier added:

```

"parameter": [
  {
    "name": "MemberPatient",
    "resource": {
      "resourceType": "Patient",
      "id": "1",
      "identifier": [
        {
          "type": {
            "coding": [
              {
                "system": "http://hl7.davinci.org",
                "code": "MB"
              }
            ]
          },
          "system": "http://oldhealthplan.example.com",
          "value": "55678",
          "assigner": {
            "reference": "Organization/2"
          }
        }
      ],
      "type": {
        "coding": [
          {
            "system": "http://terminology.hl7.org/CodeSystem/v2-0203",
            "code": "UMB"
          }
        ]
      },
      "text": "Member Match"
    },
    "system": "http://oldhealthplan.example.com",
    "value": "55678"
  }
]

```

type3

If the \$member-match can't find a unique match, you'll receive a 422 response with an error code.

Next steps

In this guide, you've learned about the \$member-match operation. Next, you can learn about testing the Da Vinci Payer Data Exchange IG in Touchstone, which requires the \$member-match operation.

[DaVinci PDex](#)

Find identity object IDs for authentication configuration

3/11/2021 • 2 minutes to read • [Edit Online](#)

In this article, you'll learn how to find identity object IDs needed when configuring the Azure API for FHIR to [use an external or secondary Active Directory tenant](#) for data plane.

Find user object ID

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

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

or you can use the Azure CLI:

```
az ad user show --id myuser@contoso.com --query objectId --out tsv
```

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-XXXXXXXXXXXXX').ObjectId
```

where `XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXXX` 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-XXXXXXXXXXXXX --query objectId --out tsv
```

Find a security group object ID

If you would like to locate the object ID of a security group, you can use the following PowerShell command:

```
$(Get-AzureADGroup -Filter "DisplayName eq 'mygroup').ObjectId
```

Where `mygroup` is the name of the group you are interested in.

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

```
az ad group show --group "mygroup" --query objectId --out tsv
```

Next steps

In this article, you've learned how to find identity object IDs needed to configure the Azure API for FHIR to use an external or secondary Azure Active Directory tenant. Next read about how to use the object IDs to configure local RBAC settings:

[Configure local RBAC settings](#)

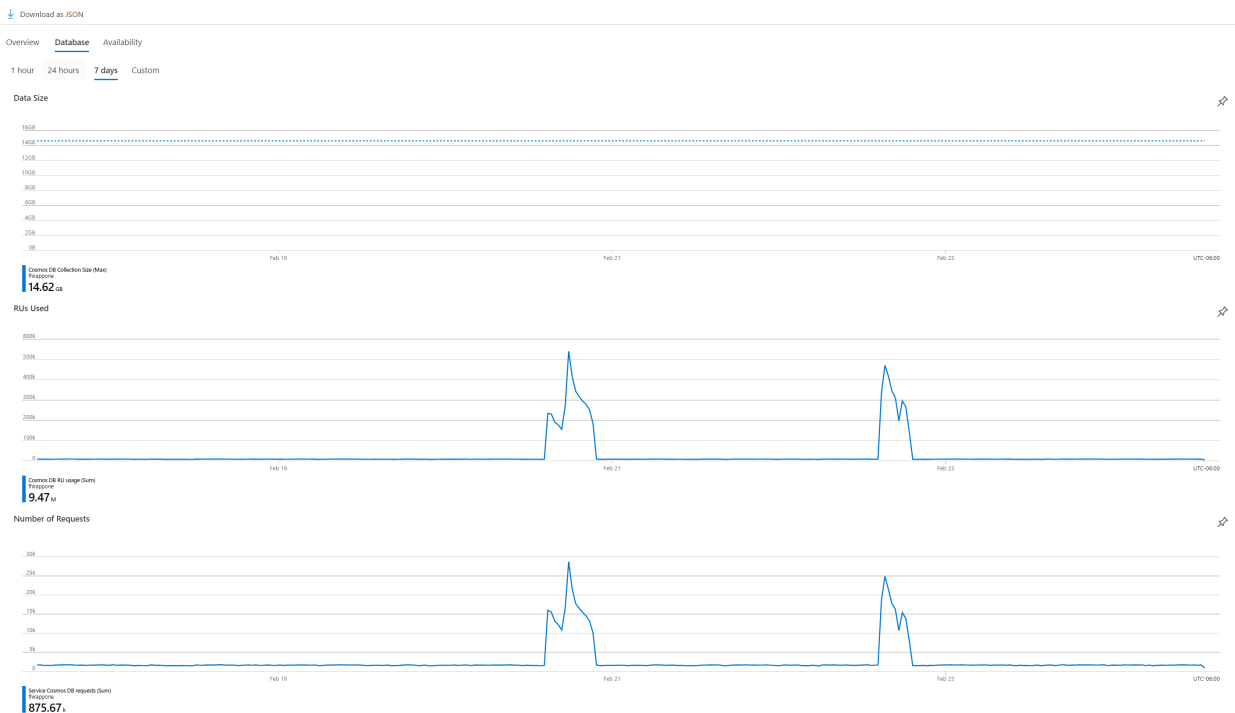
Enable Diagnostic Logging in Azure API for FHIR

3/26/2021 • 3 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.

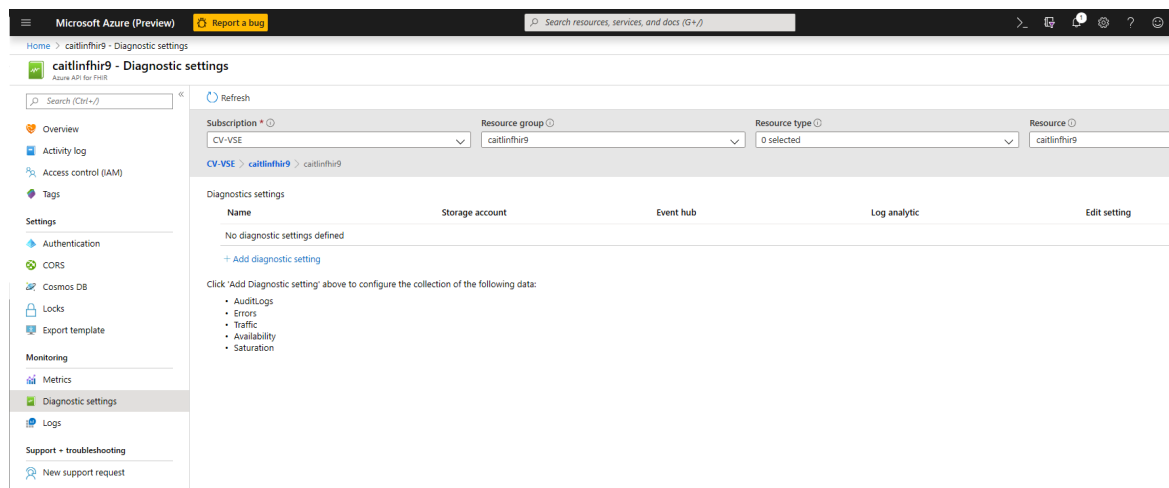
View and Download FHIR Metrics Data

You can view the metrics under Monitoring | Metrics from the portal. The metrics include Number of Requests, Average Latency, Number of Errors, Data Size, RUs Used, Number of requests that exceeded capacity, and Availability (in %). The screenshot below shows RUs used for a sample environment with very few activities in the last 7 days. You can download the data in Json format.



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:

- Archive to a storage account** for auditing or manual inspection. The storage account you want to use needs to be already created.
- 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.
- 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/or **AllMetrics**. The metrics include service name, availability, data size, total latency, total requests, total errors and timestamp. You can find more detail on [supported metrics](#).

Diagnostic setting ...

Save Discard Delete Feedback

A diagnostic setting specifies a list of categories of platform logs and/or metrics that you want to collect from a resource, and one or more destinations that you would stream them to. Normal usage charges for the destination will occur. [Learn more about the different log categories and contents of those logs](#)

Diagnostic setting name *

Category details

log

☐ AuditLogs

metric

☐ AllMetrics

Destination details

☐ Send to Log Analytics workspace

☐ Archive to a storage account

☐ Stream to an event hub

7. Select **Save**

NOTE

It might take up to 15 minutes for the first Logs to show in Log Analytics. Also, if Azure API for FHIR is moved from one resource group or subscription to another, update the setting once the move is complete.

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](#)

Display and configure Azure IoT Connector for FHIR (preview) metrics

3/11/2021 • 2 minutes to read • [Edit Online](#)

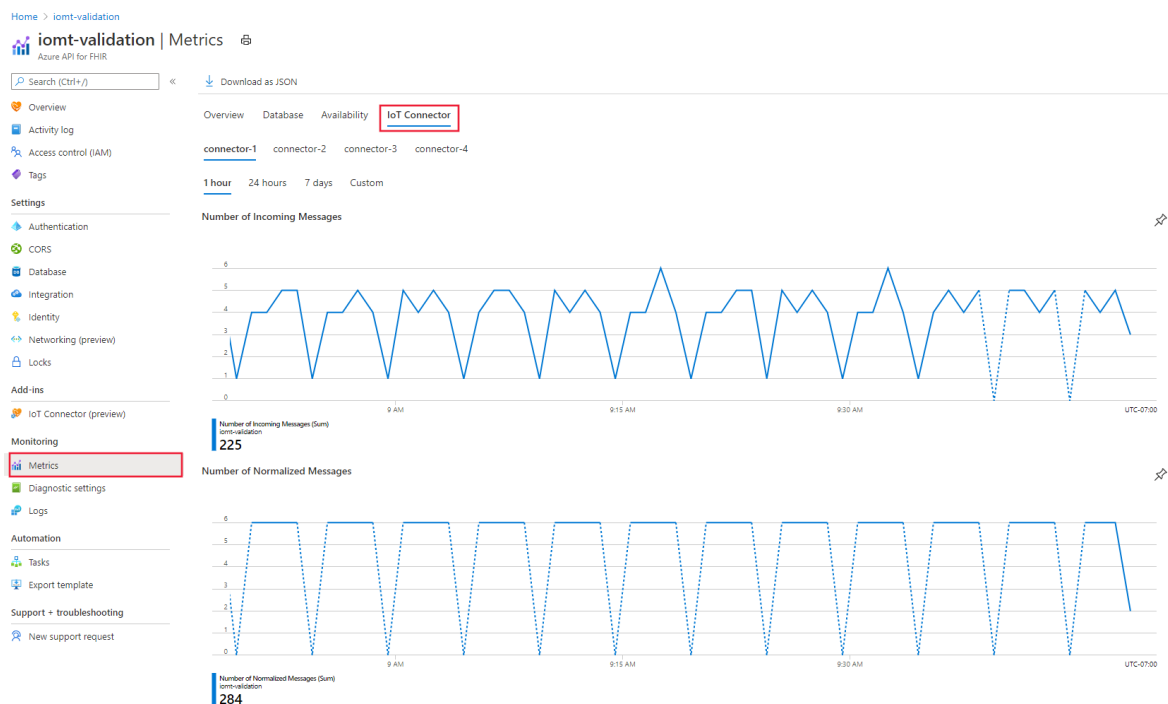
In this article, you'll learn how to display and configure Azure IoT Connector for Fast Healthcare Interoperability Resources (FHIR®)* metrics.

TIP

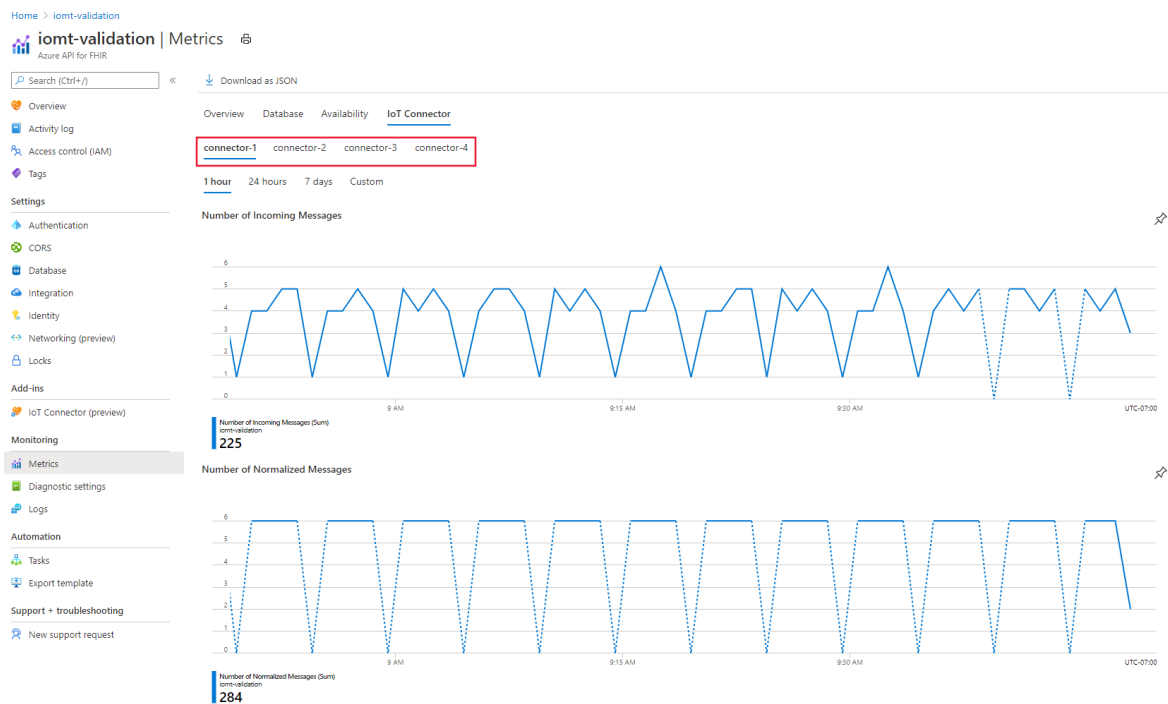
To learn how to set up the export of metrics data, follow the guidance in [Export Azure IoT Connector for FHIR \(preview\) metrics through diagnostics settings](#).

Display metrics for Azure IoT Connector for FHIR (preview)

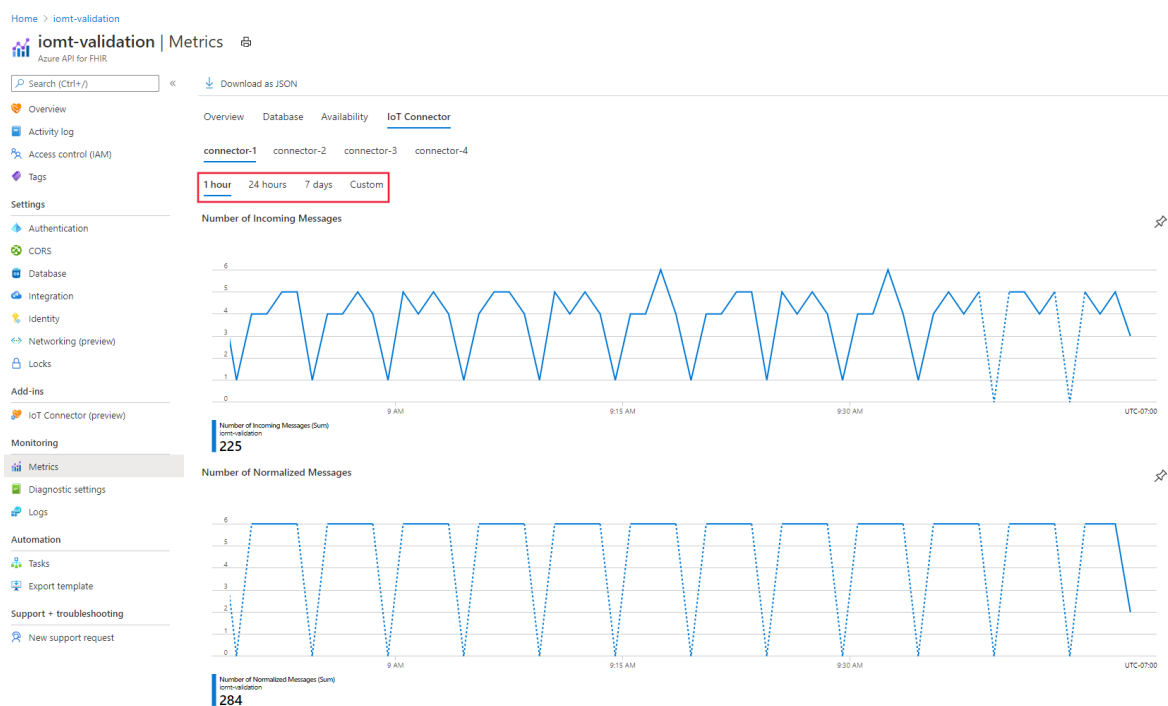
1. Sign in to the Azure portal, and then select your Azure API for FHIR service.
2. On the left pane, select **Metrics**.
3. Select the **IoT Connector** tab.



4. Select an IoT Connector to view its metrics. For example, there are four IoT Connectors (*connector 1*, *connector 2*, and so on) associated with this Azure API for FHIR service.



5. Select the time period (for example, **1 hour**, **24 hours**, **7 days**, or **Custom**) of the IoT Connector metrics you want to display. By selecting the **Custom** tab, you can create specific time/date combinations for displaying IoT Connector metrics.



Metric types for Azure IoT Connector for FHIR (preview)

TIP

To learn about data flow in Azure IoT Connector for FHIR, view [Azure IoT Connector for FHIR \(preview\) data flow](#) and [Azure IoT Connector for FHIR \(preview\) troubleshooting guide](#) to learn more about error messages and fixes.

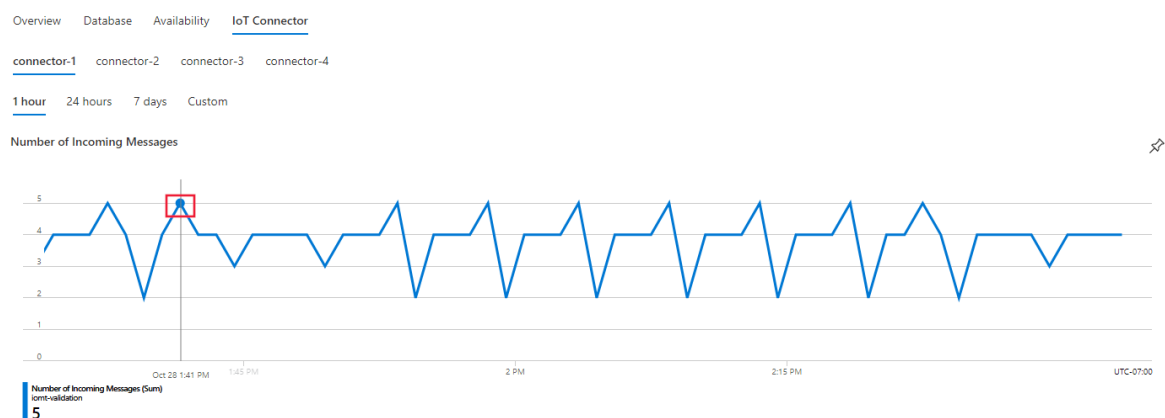
The IoT Connector metrics you can display are listed in the following table:

METRIC TYPE	METRIC PURPOSE
Number of Incoming Messages	Displays the number of received raw incoming messages (for example, the device events).
Number of Normalized Messages	Displays the number of normalized messages.
Number of Message Groups	Displays the number of groups that have messages aggregated in the designated time window.
Average Normalized Stage Latency	Displays the average latency of the normalized stage. The normalized stage performs normalization on raw incoming messages.
Average Group Stage Latency	Displays the average latency of the group stage. The group stage performs buffering, aggregating, and grouping on normalized messages.
Total Error Count	Displays the total number of errors.

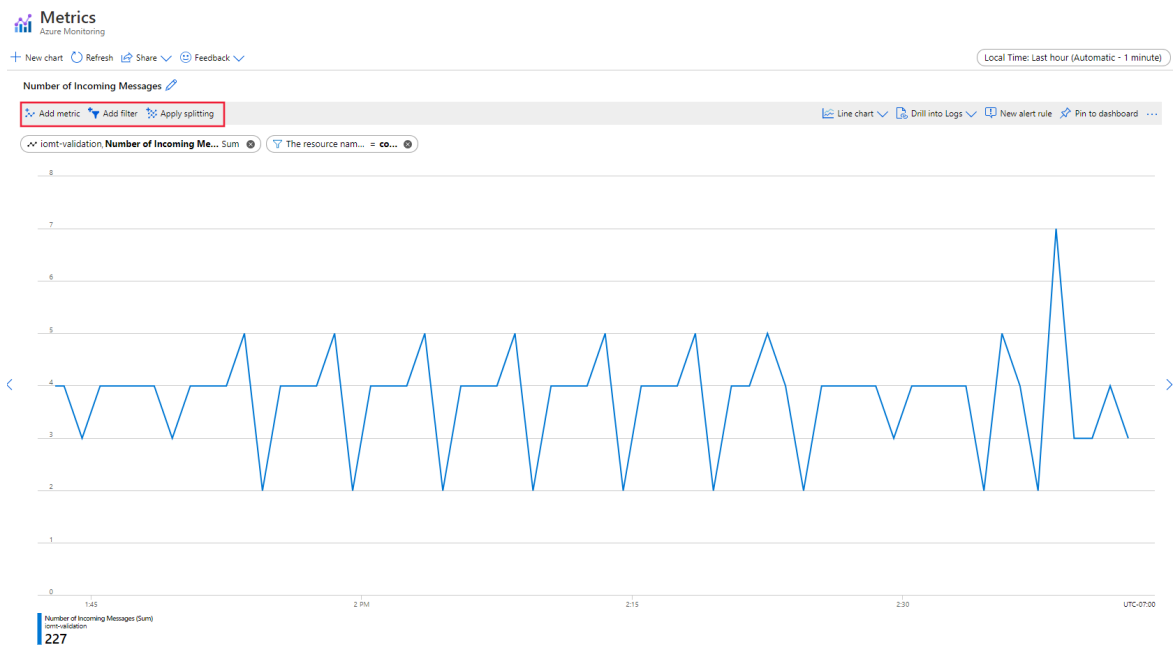
Focus on and configure Azure IoT Connector for FHIR (preview) metrics

In this example, let's focus on the **Number of Incoming Messages** metric.

1. Select a point-in-time that you want to focus on.



2. On the **Number of Incoming Messages** pane, you can further customize the metric by selecting **Add metric**, **Add filter**, or **Apply splitting**.



Conclusion

Having access to data plane metrics is essential for monitoring and troubleshooting. Azure IoT Connector for FHIR assists you with these actions through metrics.

Next steps

Get answers to frequently asked questions about Azure IoT Connector for FHIR.

[Azure IoT Connector for FHIR FAQ](#)

*In the Azure portal, Azure IoT Connector for FHIR is referred to as IoT Connector (preview). FHIR is a registered trademark of HL7 and is used with the permission of HL7.


Get access token for Azure API for FHIR using Azure CLI

3/11/2021 • 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.

Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#).

 [Launch Cloud Shell](#)

- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
 - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
 - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
 - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).

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 --query accessToken --output tsv)
```

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 IoT Connector for FHIR (preview)

troubleshooting guide

3/11/2021 • 9 minutes to read • [Edit Online](#)

This article provides steps for troubleshooting common Azure IoT Connector for Fast Healthcare Interoperability Resources (FHIR®)* error messages and conditions.

You'll also learn how to create copies of the Azure IoT Connector for FHIR conversion mappings JSON (for example: Device and FHIR).

You can use the conversion mapping JSON copies for editing and archiving outside of the Azure portal.

TIP

If you'll be opening a [Azure Technical Support](#) ticket for the Azure IoT Connector for FHIR, make sure to include copies of your conversion mapping JSON to help with the troubleshooting process.

Device and FHIR Conversion Mapping JSON Template Validations for Azure IoT Connector for FHIR (preview)

In this section, you'll learn about the validation process that Azure IoT Connector for FHIR performs to validate the Device and FHIR conversion mapping JSON templates before allowing them to be saved for use. These elements are required in the Device and FHIR Conversion Mapping JSON.

Device Mapping

ELEMENT	REQUIRED
TypeName	True
TypeMatchExpression	True
DeviceIdExpression	True
TimestampExpression	True
Values[].ValueName	True
Values[].ValueExpression	True

NOTE

Values[].ValueName and Values[].ValueExpression

These elements are only required if you have a value entry in the array - it is valid to have no values mapped. This is used when the telemetry being sent is an event. For example: When a wearable IoMT device is put on or removed. The element(s) do not have any values except for a name that Azure IoT Connector for FHIR matches and emits. On the FHIR conversion, Azure IoT Connector for FHIR maps it to a code-able concept based on the semantic type - no actual values are populated.

FHIR Mapping

ELEMENT	REQUIRED
TypeName	True

NOTE

This is the only required FHIR Mapping element validated at this time.

Error messages and fixes for Azure IoT Connector for FHIR (preview)

MESSAGE	DISPLAYED	CONDITION	FIX
Invalid mapping name, mapping name should be device or FHIR.	API	Mapping type supplied isn't device or FHIR.	Use one of the two supported mapping types (for example: Device or FHIR).
Validation failed. Required information is missing or not valid.	API and Azure portal	Attempting to save a conversion mapping missing needed information or element.	Add missing conversion mapping information or element and attempt to save the conversion mapping again.
Regenerate key parameters not defined.	API	Regenerate key request.	Include the parameters in the regeneration key request.
Reached the maximum number of IoT Connector instances that can be provisioned in this subscription.	API and Azure portal	Azure IoT Connector for FHIR subscription quota reached (Default is (2) per subscription).	Delete one of the existing instances of Azure IoT Connector for FHIR. Use a different subscription that hasn't reached the subscription quota. Request a subscription quota increase.
Move resource is not supported for IoT Connector enabled Azure API for FHIR resource.	API and Azure portal	Attempting to do a move operation on an Azure API for FHIR resource that has one or more instances of the Azure IoT Connector for FHIR.	Delete existing instance(s) of Azure IoT Connector for FHIR to do the move operation.
IoT Connector not provisioned.	API	Attempting to use child services (connections & mappings) when parent (Azure IoT Connector for FHIR) hasn't been provisioned.	Provision an Azure IoT Connector for FHIR.
The request is not supported.	API	Specific API request isn't supported.	Use the correct API request.

MESSAGE	DISPLAYED	CONDITION	FIX
Account does not exist.	API	Attempting to add an Azure IoT Connector for FHIR and the Azure API for FHIR resource doesn't exist.	Create the Azure API for FHIR resource and then reattempt the operation.
Azure API for FHIR resource FHIR version is not supported for IoT Connector.	API	Attempting to use an Azure IoT Connector for FHIR with an incompatible version of the Azure API for FHIR resource.	Create a new Azure API for FHIR resource (version R4) or use an existing Azure API for FHIR resource (version R4).

Why is my Azure IoT Connector for FHIR (preview) data not showing up in Azure API for FHIR?

POTENTIAL ISSUES	FIXES
Data is still being processed.	Data is egressed to the Azure API for FHIR in batches (every ~15 minutes). It's possible the data is still being processed and additional time is needed for the data to be persisted in the Azure API for FHIR.
Device conversion mapping JSON hasn't been configured.	Configure and save conforming device conversion mapping JSON.
FHIR conversion mapping JSON has not been configured.	Configure and save conforming FHIR conversion mapping JSON.
The device message doesn't contain an expected expression defined in the device mapping.	Verify JsonPath expressions defined in the device mapping match tokens defined in the device message.
A Device Resource hasn't been created in the Azure API for FHIR (Resolution Type: Lookup only)*.	Create a valid Device Resource in the Azure API for FHIR. Be sure the Device Resource contains an Identifier that matches the device identifier provided in the incoming message.
A Patient Resource has not been created in the Azure API for FHIR (Resolution Type: Lookup only)*.	Create a valid Patient Resource in the Azure API for FHIR.
The Device.patient reference isn't set, or the reference is invalid (Resolution Type: Lookup only)*.	Make sure the Device Resource contains a valid Reference to a Patient Resource.

*Reference [Quickstart: Deploy Azure IoT Connector \(preview\) using Azure portal](#) for a functional description of the Azure IoT Connector for FHIR resolution types (For example: Lookup or Create).

Use Metrics to troubleshoot issues in Azure IoT Connector for FHIR (preview)

Azure IoT Connector for FHIR generates multiple metrics to provide insights into the data flow process. One of the supported metrics is called *Total Errors*, which provides the count for all errors that occur within an instance of Azure IoT Connector for FHIR.

Each error gets logged with a number of associated properties. Every property provides a different aspect about the error, which could help you to identify and troubleshoot issues. This section lists different properties captured for each error in the *Total Errors* metric, and possible values for these properties.

NOTE

You can navigate to the *Total Errors* metric for an instance of Azure IoT Connector for FHIR (preview) as described on the [Azure IoT Connector for FHIR \(preview\) Metrics page](#).

Click on the *Total Errors* graph and then click on *Add filter* button to slice and dice the error metric using any of the properties mentioned below.

The operation performed by the Azure IoT Connector for FHIR (preview)

This property represents the operation being performed by IoT Connector when the error has occurred. An operation generally represents the data flow stage while processing a device message. Here is the list of possible values for this property.

NOTE

You can read more about different stages of data flow in Azure IoT Connector for FHIR (preview) [here](#).

DATA FLOW STAGE	DESCRIPTION
Setup	Operation specific to setting up your instance of IoT Connector
Normalization	Data flow stage where device data gets normalized
Grouping	Data flow stage where normalized data gets grouped
FHIRConversion	Data flow stage where grouped-normalized data is transformed into a FHIR resource
Unknown	The operation type is unknown when error occurred

The severity of the error

This property represents the severity of the occurred error. Here is the list of possible values for this property.

SEVERITY	DESCRIPTION
Warning	Some minor issue exists in the data flow process, but processing of the device message doesn't stop
Error	Processing of a specific device message has run into an error and other messages may continue to execute as expected
Critical	Some system level issue exists with the IoT Connector and no messages are expected to process

The type of the error

This property signifies a category for a given error, which basically represents a logical grouping for similar type of errors. Here is the list of possible value for this property.

ERROR TYPE	DESCRIPTION
DeviceTemplateError	Errors related to device mapping templates

ERROR TYPE	DESCRIPTION
DeviceMessageError	Errors occurred when processing a specific device message
FHIRTemplateError	Errors related to FHIR mapping templates
FHIRConversionError	Errors occurred when transforming a message into a FHIR resource
FHIRResourceError	Errors related to existing resources in the FHIR server that are referenced by IoT Connector
FHIRServerError	Errors that occur when communicating with FHIR server
GeneralError	All other types of errors

The name of the error

This property provides the name for a specific error. Here is the list of all error names with their description and associated error type(s), severity, and data flow stage(s).

ERROR NAME	DESCRIPTION	ERROR TYPE(S)	ERROR SEVERITY	DATA FLOW STAGE(S)
MultipleResourceFoundException	Error occurred when multiple patient or device resources are found in the FHIR server for respective identifiers present in the device message	FHIRResourceError	Error	FHIRConversion
TemplateNotFoundException	A device or FHIR mapping template isn't configured with the instance of IoT Connector	DeviceTemplateError, FHIRTemplateError	Critical	Normalization, FHIRConversion
CorrelationIdNotDefinedException	Correlation ID isn't specified in the device mapping template. CorrelationIdNotDefinedException is a conditional error that would occur only when FHIR Observation must group device measurements using a correlation ID but it's not configured correctly	DeviceMessageError	Error	Normalization

ERROR NAME	DESCRIPTION	ERROR TYPE(S)	ERROR SEVERITY	DATA FLOW STAGE(S)
PatientDeviceMismatchException	This error occurs when the device resource on the FHIR server has a reference to a patient resource, which doesn't match with the patient identifier present in the message	FHIRResourceError	Error	FHIRConversionError
PatientNotFoundException	No Patient FHIR resource is referenced by the Device FHIR resource associated with the device identifier present in the device message. Note this error will only occur when IoT Connector instance is configured with <i>Lookup</i> resolution type	FHIRConversionError	Error	FHIRConversion
DeviceNotFoundException	No device resource exists on the FHIR Server associated with the device identifier present in the device message	DeviceMessageError	Error	Normalization
PatientIdentityNotDefinedException	This error occurs when expression to parse patient identifier from the device message isn't configured on the device mapping template or patient identifier isn't present in the device message. Note this error occurs only when IoT Connector's resolution type is set to <i>Create</i>	DeviceTemplateError	Critical	Normalization
DeviceIdentityNotDefinedException	This error occurs when expression to parse device identifier from the device message isn't configured on the device mapping template or device identifier isn't present in the device message	DeviceTemplateError	Critical	Normalization

ERROR NAME	DESCRIPTION	ERROR TYPE(S)	ERROR SEVERITY	DATA FLOW STAGE(S)
NotSupportedException	Error occurred when device message with unsupported format is received	DeviceMessageError	Error	Normalization

Creating copies of the Azure IoT Connector for FHIR (preview) conversion mapping JSON

The copying of Azure IoT Connector for FHIR mapping files can be useful for editing and archiving outside of the Azure portal website.

The mapping file copies should be provided to Azure Technical Support when opening a support ticket to assist in troubleshooting.

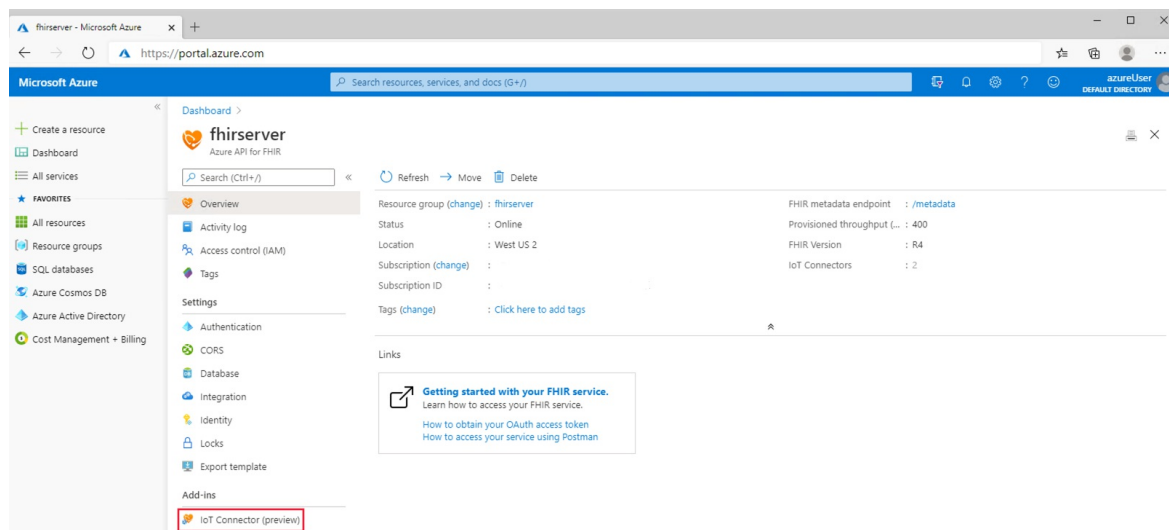
NOTE

JSON is the only supported format for Device and FHIR mapping files at this time.

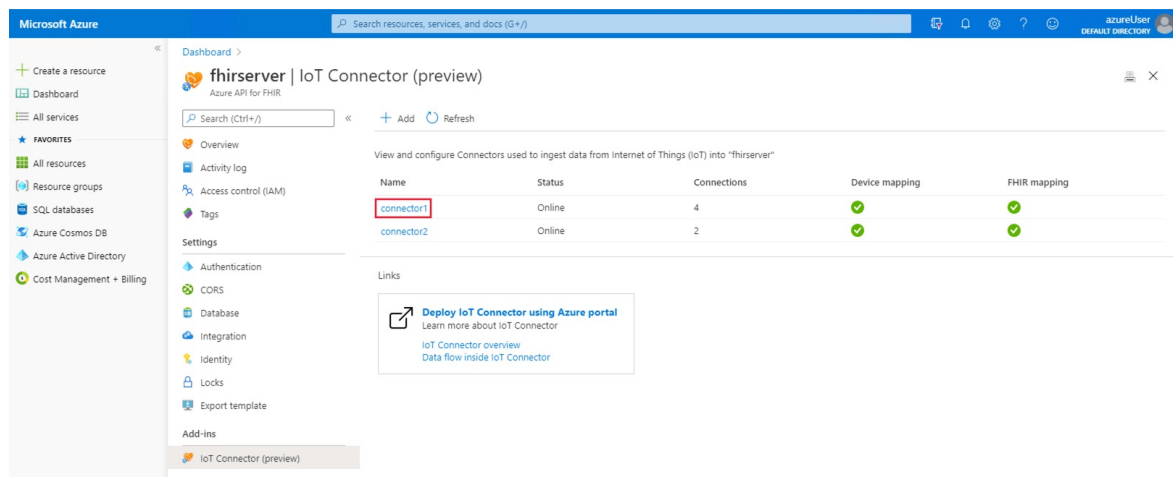
TIP

Learn more about the Azure IoT Connector for FHIR [Device](#) and [FHIR conversion mapping JSON](#)

1. Select **"IoT Connector (preview)"** on the lower left side of the Azure API for FHIR resource dashboard in the **"Add-ins"** section.



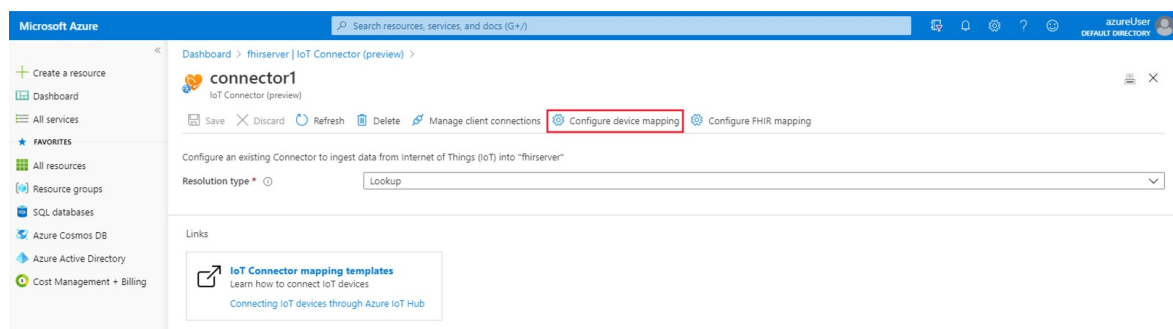
2. Select the **"Connector"** that you'll be copying the conversion mapping JSON from.



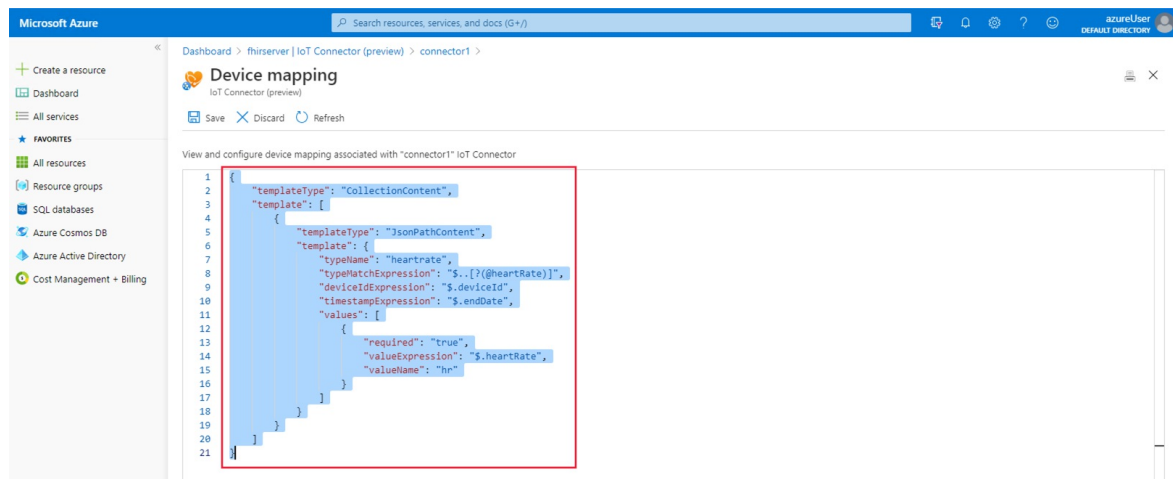
NOTE

This process may also be used for copying and saving the contents of the "Configure FHIR mapping" JSON.

3. Select "Configure device mapping".



4. Select the contents of the JSON and do a copy operation (for example: Select Ctrl + c).



5. Do a paste operation (for example: Select Ctrl + v) into a new file within an editor (for example: Visual Studio Code, Notepad) and save the file with an *.json extension.

TIP

If you'll be opening a [Azure Technical Support](#) ticket for the Azure IoT Connector for FHIR, make sure to include copies of your conversion mapping JSON to help with the troubleshooting process.

Next steps

Check out frequently asked questions about the Azure IoT Connector for FHIR.

[Azure IoT Connector for FHIR FAQs](#)

*In the Azure portal, Azure IoT Connector for FHIR is referred to as IoT Connector (preview). FHIR is a registered trademark of HL7 and is used with the permission of HL7.

Azure Active Directory identity configuration for Azure API for FHIR

3/11/2021 • 4 minutes to read • [Edit Online](#)

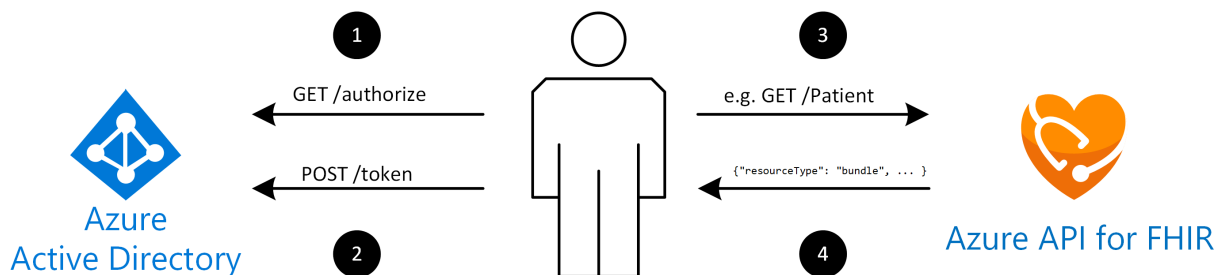
An important piece when working with healthcare data is to ensure that the data is secure and cannot be accessed by unauthorized users or applications. FHIR servers use [OAuth 2.0](#) to ensure this data security. The [Azure API for FHIR](#) is secured using [Azure Active Directory](#), which is an example 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 (`a1g`) 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.eyJvawQioiIxMjMiLCAlaXNzIjoiaHR0cHM6Ly9pc3N1ZXJ1cmwiLCJpYXQiOjE0MjI3Nzk2MzgsInJvbGVZInpjbImFkbWluIl19.gzSraSYS8EXBxLN oWnFSRGzczmJmMjiLuyy5CSpyHI

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

3/11/2021 • 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 user or principal that the token was issued for is allowed to access the FHIR server data plane. The `oid` claim of the token contains an identity object ID, which uniquely identifies the user or principal.

We recommend that the FHIR service be [configured to use Azure RBAC](#) to manage data plane role assignments. But you can also [configure local RBAC](#) if your FHIR service uses an external or secondary Azure Active Directory tenant.

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](#)

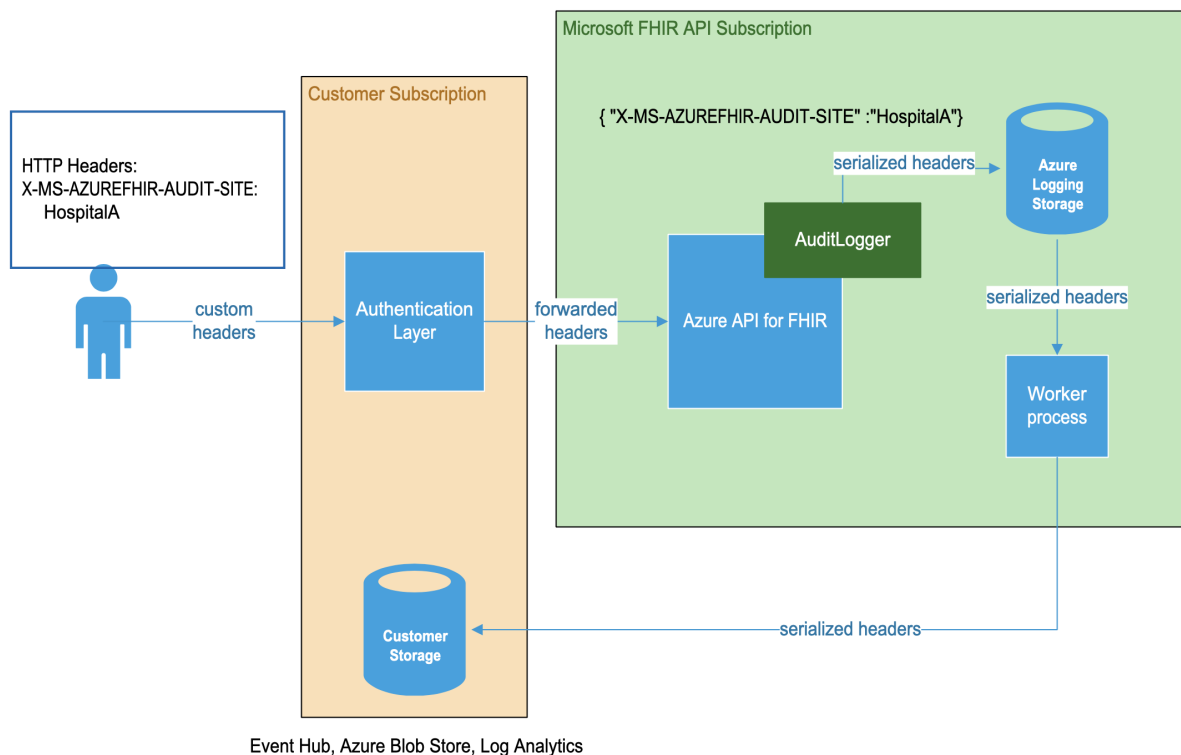
Add data to audit logs by using custom HTTP headers

3/11/2021 • 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-<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 Firefly 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](#)

Azure IoT Connector for FHIR (preview) data flow

3/11/2021 • 3 minutes to read • [Edit Online](#)

This article provides an overview of data flow in Azure IoT Connector for Fast Healthcare Interoperability Resources (FHIR®)*. You'll learn about different data processing stages within Azure IoT Connector for FHIR that transform device data into FHIR-based [Observation](#) resources.

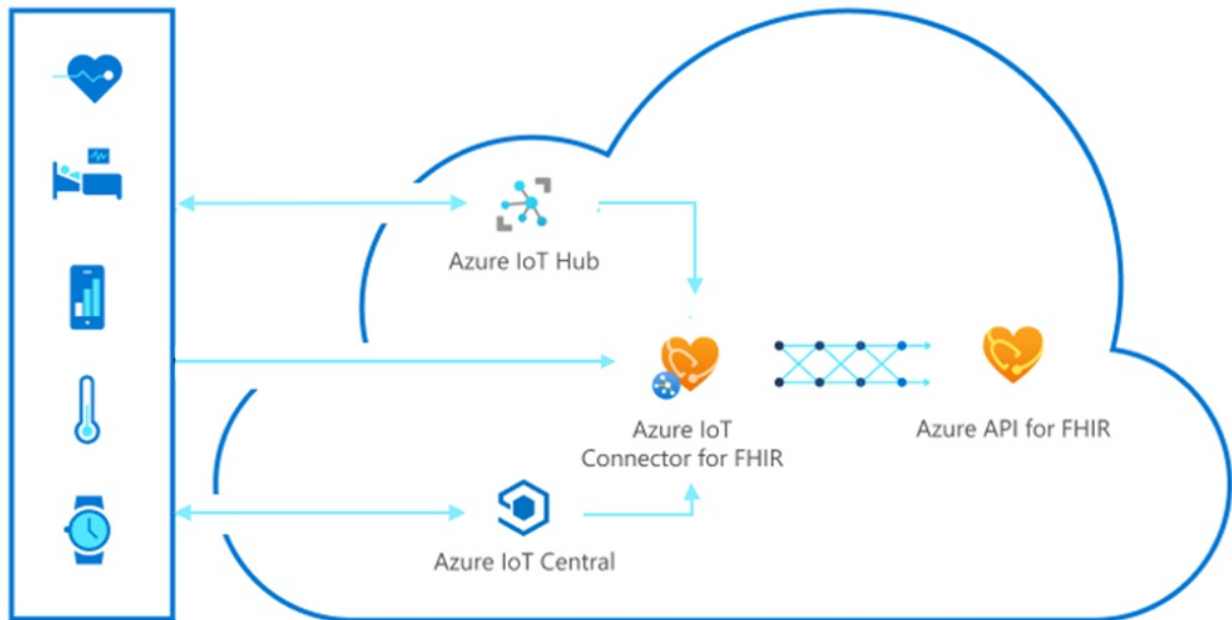


Diagram above shows common data flows using Azure IoT Connector for FHIR.

Below are different stages that data goes through once received by Azure IoT Connector for FHIR.

Ingest

Ingest is the first stage where device data is received into Azure IoT Connector for FHIR. The ingestion endpoint for device data is hosted on an [Azure Event Hub](#). Azure Event Hub platform supports high scale and throughput with ability to receive and process millions of messages per second. It also enables Azure IoT Connector for FHIR to consume messages asynchronously, removing the need for devices to wait while device data gets processed.

NOTE

JSON is the only supported format at this time for device data.

Normalize

Normalize is the next stage where device data is retrieved from the above Azure Event Hub and processed using device mapping templates. This mapping process results in transforming device data into a normalized schema.

The normalization process not only simplifies data processing at later stages but also provides the ability to project one input message into multiple normalized messages. For instance, a device could send multiple vital signs for body temperature, pulse rate, blood pressure, and respiration rate in a single message. This input message would create four separate FHIR resources. Each resource would represent different vital sign, with the input message projected into four different normalized messages.

Group

Group is the next stage where the normalized messages available from the previous stage are grouped using three different parameters: device identity, measurement type, and time period.

Device identity and measurement type grouping enable use of [SampledData](#) measurement type. This type provides a concise way to represent a time-based series of measurements from a device in FHIR. And time period controls the latency at which Observation resources generated by Azure IoT Connector for FHIR are written to Azure API for FHIR.

NOTE

The time period value is defaulted to 15 minutes and cannot be configured for preview.

Transform

In the Transform stage, grouped-normalized messages are processed through FHIR mapping templates. Messages matching a template type get transformed into FHIR-based Observation resources as specified through the mapping.

At this point, [Device](#) resource, along with its associated [Patient](#) resource, is also retrieved from the FHIR server using the device identifier present in the message. These resources are added as a reference to the Observation resource being created.

NOTE

All identity look ups are cached once resolved to decrease load on the FHIR server. If you plan on reusing devices with multiple patients it is advised you create a virtual device resource that is specific to the patient and send virtual device identifier in the message payload. The virtual device can be linked to the actual device resource as a parent.

If no Device resource for a given device identifier exists in the FHIR server, the outcome depends upon the value of `Resolution Type` set at the time of creation. When set to `Lookup`, the specific message is ignored, and the pipeline will continue to process other incoming messages. If set to `Create`, Azure IoT Connector for FHIR will create a bare-bones Device and Patient resources on the FHIR server.

Persist

Once the Observation FHIR resource is generated in the Transform stage, resource is saved into Azure API for FHIR. If the FHIR resource is new, it will be created on the server. If the FHIR resource already existed, it will get updated.

Next steps

Click below next step to learn how to create device and FHIR mapping templates.

[Azure IoT Connector for FHIR mapping templates](#)

*In the Azure portal, Azure IoT Connector for FHIR is referred to as IoT Connector (preview). FHIR is a registered trademark of HL7 and is used with the permission of HL7.

Azure IoT Connector for FHIR (preview) mapping templates

4/5/2021 • 11 minutes to read • [Edit Online](#)

This article details how to configure Azure IoT Connector for Fast Healthcare Interoperability Resources (FHIR®)* using mapping templates.

The Azure IoT Connector for FHIR requires two types of JSON-based mapping templates. The first type, **Device mapping**, is responsible for mapping the device payloads sent to the `devicedata` Azure Event Hub end point. It extracts types, device identifiers, measurement date time, and the measurement value(s). The second type, **FHIR mapping**, controls the mapping for FHIR resource. It allows configuration of the length of the observation period, FHIR data type used to store the values, and terminology code(s).

The mapping templates are composed into a JSON document based on their type. These JSON documents are then added to your Azure IoT Connector for FHIR through the Azure portal. The Device mapping document is added through the **Configure Device mapping** page and the FHIR mapping document through the **Configure FHIR mapping** page.

NOTE

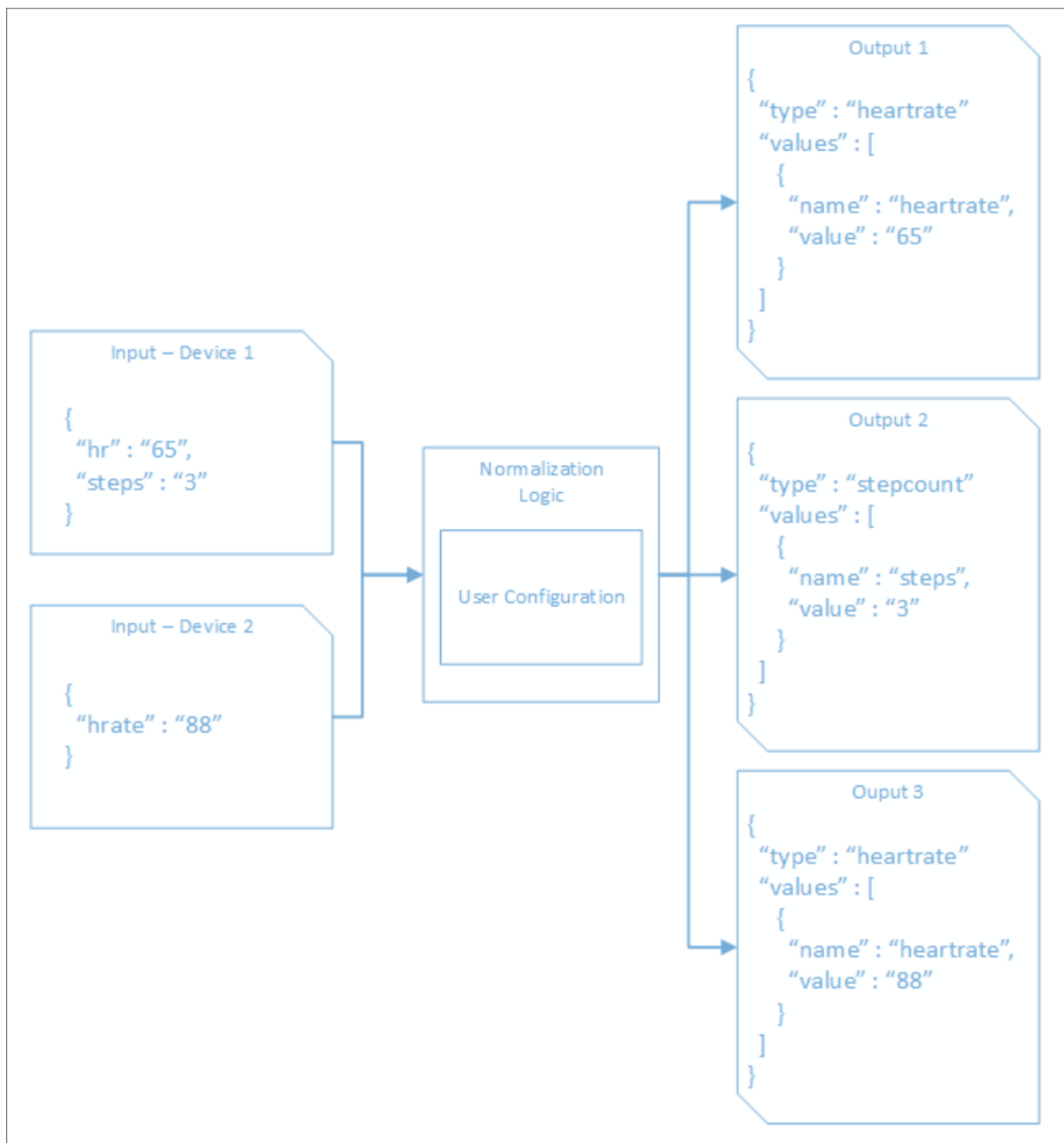
Mapping templates are stored in an underlying blob storage and loaded from blob per compute execution. Once updated they should take effect immediately.

Device mapping

Device mapping provides mapping functionality to extract device content into a common format for further evaluation. Each message received is evaluated against all templates. This approach allows a single inbound message to be projected to multiple outbound messages, which are later mapped to different observations in FHIR. The result is a normalized data object representing the value or values parsed by the templates. The normalized data model has a few required properties that must be found and extracted:

PROPERTY	DESCRIPTION
Type	The name/type to classify the measurement. This value is used to bind to the required FHIR mapping template. Multiple templates can output to the same type allowing you to map different representations across multiple devices to a single common output.
OccurrenceTimeUtc	The time the measurement occurred.
DeviceId	The identifier for the device. This value should match an identifier on the device resource that exists on the destination FHIR server.
Properties	Extract at least one property so the value can be saved in the Observation resource created. Properties are a collection of key value pairs extracted during normalization.

Below is a conceptual example of what happens during normalization.



The content payload itself is an Azure Event Hub message, which is composed of three parts: Body, Properties, and SystemProperties. The `Body` is a byte array representing an UTF-8 encoded string. During template evaluation, the byte array is automatically converted into the string value. `Properties` is a key value collection for use by the message creator. `SystemProperties` is also a key value collection reserved by the Azure Event Hub framework with entries automatically populated by it.

```
{
  "Body": {
    "content": "value"
  },
  "Properties": {
    "key1": "value1",
    "key2": "value2"
  },
  "SystemProperties": {
    "x-opt-sequence-number": 1,
    "x-opt-enqueued-time": "2019-02-01T22:46:01.875000Z",
    "x-opt-offset": 1,
    "x-opt-partition-key": "1"
  }
}
```

Mapping with JSON path

The three device content template types supported today rely on JSON Path to both match the required template and extracted values. More information on JSON Path can be found [here](#). All three template types use the [JSON .NET implementation](#) for resolving JSON Path expressions.

JsonPathContentTemplate

The JsonPathContentTemplate allows matching on and extracting values from an Event Hub message using JSON Path.

PROPERTY	DESCRIPTION	EXAMPLE
TypeName	The type to associate with measurements that match the template.	<code>heartrate</code>
TypeMatchExpression	The JSON Path expression that is evaluated against the Event Hub payload. If a matching JToken is found, the template is considered a match. All subsequent expressions are evaluated against the extracted JToken matched here.	<code>\$.?(@heartRate)]</code>
TimestampExpression	The JSON Path expression to extract the timestamp value for the measurement's OccurrenceTimeUtc.	<code>\$.endDate</code>
DeviceIdExpression	The JSON Path expression to extract the device identifier.	<code>\$.deviceId</code>
PatientIdExpression	<i>Optional.</i> The JSON Path expression to extract the patient identifier.	<code>\$.patientId</code>
EncounterIdExpression	<i>Optional.</i> The JSON Path expression to extract the encounter identifier.	<code>\$.encounterId</code>
Values[].ValueName	The name to associate with the value extracted by the subsequent expression. Used to bind the required value/component in the FHIR mapping template.	<code>hr</code>

PROPERTY	DESCRIPTION	EXAMPLE
Values[].ValueExpression	The JSON Path expression to extract the required value.	<code>\$.heartRate</code>
Values[].Required	Will require the value to be present in the payload. If not found, a measurement will not be generated and an InvalidOperationException will be thrown.	<code>true</code>

Examples

Heart rate

Message

```
{
  "Body": {
    "heartRate": "78",
    "endDate": "2019-02-01T22:46:01.875000Z",
    "deviceId": "device123"
  },
  "Properties": {},
  "SystemProperties": {}
}
```

Template

```
{
  "templateType": "JsonPathContent",
  "template": {
    "typeName": "heartrate",
    "typeMatchExpression": "$..[?(@heartRate)]",
    "deviceIdExpression": "$.deviceId",
    "timestampExpression": "$.endDate",
    "values": [
      {
        "required": "true",
        "valueExpression": "$.heartRate",
        "valueName": "hr"
      }
    ]
  }
}
```

Blood pressure

Message

```
{
  "Body": {
    "systolic": "123",
    "diastolic": "87",
    "endDate": "2019-02-01T22:46:01.875000Z",
    "deviceId": "device123"
  },
  "Properties": {},
  "SystemProperties": {}
}
```

Template

```
{
  "typeName": "bloodpressure",
  "typeMatchExpression": "$..[?(@systolic && @diastolic)]",
  "deviceIdExpression": "$.deviceId",
  "timestampExpression": "$.endDate",
  "values": [
    {
      "required": "true",
      "valueExpression": "$.systolic",
      "valueName": "systolic"
    },
    {
      "required": "true",
      "valueExpression": "$.diastolic",
      "valueName": "diastolic"
    }
  ]
}
```

Project multiple measurements from single message

Message

```
{
  "Body": {
    "heartRate": "78",
    "steps": "2",
    "endDate": "2019-02-01T22:46:01.875000Z",
    "deviceId": "device123"
  },
  "Properties": {},
  "SystemProperties": {}
}
```

Template 1

```
{
  "templateType": "JsonPathContent",
  "template": {
    "typeName": "heartrate",
    "typeMatchExpression": "$..[?(@heartRate)]",
    "deviceIdExpression": "$.deviceId",
    "timestampExpression": "$.endDate",
    "values": [
      {
        "required": "true",
        "valueExpression": "$.heartRate",
        "valueName": "hr"
      }
    ]
  }
}
```

Template 2

```

{
  "templateType": "JsonPathContent",
  "template": {
    "typeName": "stepcount",
    "typeMatchExpression": "$..[?(@steps)]",
    "deviceIdExpression": "$.deviceId",
    "timestampExpression": "$.endDate",
    "values": [
      {
        "required": "true",
        "valueExpression": "$.steps",
        "valueName": "steps"
      }
    ]
  }
}

```

Project multiple measurements from array in message

Message

```

{
  "Body": [
    {
      "heartRate": "78",
      "endDate": "2019-02-01T22:46:01.8750000Z",
      "deviceId": "device123"
    },
    {
      "heartRate": "81",
      "endDate": "2019-02-01T23:46:01.8750000Z",
      "deviceId": "device123"
    },
    {
      "heartRate": "72",
      "endDate": "2019-02-01T24:46:01.8750000Z",
      "deviceId": "device123"
    }
  ],
  "Properties": {},
  "SystemProperties": {}
}

```

Template

```

{
  "templateType": "JsonPathContent",
  "template": {
    "typeName": "heartrate",
    "typeMatchExpression": "$..[?(@heartRate)]",
    "deviceIdExpression": "$.deviceId",
    "timestampExpression": "$.endDate",
    "values": [
      {
        "required": "true",
        "valueExpression": "$.heartRate",
        "valueName": "hr"
      }
    ]
  }
}

```

The `lotJsonPathContentTemplate` is similar to the `JsonPathContentTemplate` except the `DeviceIdExpression` and `TimestampExpression` aren't required.

The assumption when using this template is the messages being evaluated were sent using the [Azure IoT Hub Device SDKs](#) or [Export Data \(legacy\)](#) feature of [Azure IoT Central](#). When using these SDKs, the device identity (assuming the device identifier from Azure IoT Hub/Central is registered as an identifier for a device resource on the destination FHIR server) and the timestamp of the message are known. If you're using Azure IoT Hub Device SDKs but are using custom properties in the message body for the device identity or measurement timestamp, you can still use the `JsonPathContentTemplate`.

Note: When using the `lotJsonPathContentTemplate`, the `TypeMatchExpression` should resolve to the entire message as a `JToken`. See the examples below.

Examples

Heart rate

Message

```
{
  "Body": {
    "heartRate": "78"
  },
  "Properties": {
    "iothub-creation-time-utc" : "2019-02-01T22:46:01.8750000Z"
  },
  "SystemProperties": {
    "iothub-connection-device-id" : "device123"
  }
}
```

Template

```
{
  "templateType": "JsonPathContent",
  "template": {
    "typeName": "heartrate",
    "typeMatchExpression": "$..[?(@Body.heartRate)]",
    "deviceIdExpression": "$.deviceId",
    "timestampExpression": "$.endDate",
    "values": [
      {
        "required": "true",
        "valueExpression": "$.Body.heartRate",
        "valueName": "hr"
      }
    ]
  }
}
```

Blood pressure

Message

```
{
  "Body": {
    "systolic": "123",
    "diastolic" : "87"
  },
  "Properties": {
    "iothub-creation-time-utc" : "2019-02-01T22:46:01.8750000Z"
  },
  "SystemProperties": {
    "iothub-connection-device-id" : "device123"
  }
}
```

Template

```
{
  "typeName": "bloodpressure",
  "typeMatchExpression": "$..[?(@Body.systolic && @Body.diastolic)]",
  "values": [
    {
      "required": "true",
      "valueExpression": "$.Body.systolic",
      "valueName": "systolic"
    },
    {
      "required": "true",
      "valueExpression": "$.Body.diastolic",
      "valueName": "diastolic"
    }
  ]
}
```

lotCentralJsonPathContentTemplate

The `lotCentralJsonPathContentTemplate` also doesn't require `DeviceIdExpression` and `TimestampExpression`, and used when messages being evaluated are sent through the [Export Data](#) feature of [Azure IoT Central](#). When using this feature, the device identity (assuming the device identifier from Azure IoT Central is registered as an identifier for a device resource on the destination FHIR server) and the timestamp of the message are known. If you're using Azure IoT Central's Data Export feature but are using custom properties in the message body for the device identity or measurement timestamp, you can still use the `JsonPathContentTemplate`.

Note: When using the `lotCentralJsonPathContentTemplate`, the `TypeMatchExpression` should resolve to the entire message as a `JToken`. See the examples below.

Examples

Heart rate

Message


```
{
  "applicationId": "1dffa667-9bee-4f16-b243-25ad4151475e",
  "messageSource": "telemetry",
  "deviceId": "1vzb5ghlsg1",
  "schema": "default@v1",
  "templateId": "urn:qugj6vbw5:___qbj_27r",
  "enqueuedTime": "2020-08-05T22:26:55.455Z",
  "telemetry": {
    "HeartRate": "88",
  },
  "enrichments": {
    "userSpecifiedKey": "sampleValue"
  },
  "messageProperties": {
    "messageProp": "value"
  }
}
```

Template

```
{
  "templateType": "IotCentralJsonPathContent",
  "template": {
    "typeName": "heartrate",
    "typeMatchExpression": "$..[?(@telemetry.HeartRate)]",
    "values": [
      {
        "required": "true",
        "valueExpression": "$.telemetry.HeartRate",
        "valueName": "hr"
      }
    ]
  }
}
```

Blood pressure

Message

```
{
  "applicationId": "1dffa667-9bee-4f16-b243-25ad4151475e",
  "messageSource": "telemetry",
  "deviceId": "1vzb5ghlsg1",
  "schema": "default@v1",
  "templateId": "urn:qugj6vbw5:___qbj_27r",
  "enqueuedTime": "2020-08-05T22:26:55.455Z",
  "telemetry": {
    "BloodPressure": {
      "Diastolic": "87",
      "Systolic": "123"
    }
  },
  "enrichments": {
    "userSpecifiedKey": "sampleValue"
  },
  "messageProperties": {
    "messageProp": "value"
  }
}
```

Template

```

{
  "templateType": "IotCentralJsonPathContent",
  "template": {
    "typeName": "bloodPressure",
    "typeMatchExpression": "$..[?(@telemetry.BloodPressure.Diastolic &&
@telemetry.BloodPressure.Systolic)]",
    "values": [
      {
        "required": "true",
        "valueExpression": "$.telemetry.BloodPressure.Diastolic",
        "valueName": "bp_diastolic"
      },
      {
        "required": "true",
        "valueExpression": "$.telemetry.BloodPressure.Systolic",
        "valueName": "bp_systolic"
      }
    ]
  }
}

```

FHIR mapping

Once the device content is extracted into a normalized model, the data is collected and grouped according to device identifier, measurement type, and time period. The output of this grouping is sent for conversion into a FHIR resource ([Observation](#) currently). Here the FHIR mapping template controls how the data is mapped into a FHIR Observation. Should an observation be created for a point in time or over a period of an hour? What codes should be added to the observation? Should value be represented as [SampledData](#) or a [Quantity](#)? These data types are all options the FHIR mapping configuration controls.

CodeValueFhirTemplate

The CodeValueFhirTemplate is currently the only template supported in FHIR mapping at this time. It allows you to define codes, the effective period, and the value of the observation. Multiple value types are supported: [SampledData](#), [CodeableConcept](#), and [Quantity](#). Along with these configurable values, the identifier for the Observation resource and linking to the proper Device and Patient resources are handled automatically.

PROPERTY	DESCRIPTION
TypeName	The type of measurement this template should bind to. There should be at least one Device mapping template that outputs this type.
PeriodInterval	The period of time the observation created should represent. Supported values are 0 (an instance), 60 (an hour), 1440 (a day).
Category	Any number of CodeableConcepts to classify the type of observation created.
Codes	One or more Codings to apply to the observation created.
Codes[].Code	The code for the Coding .
Codes[].System	The system for the Coding .
Codes[].Display	The display for the Coding .

PROPERTY	DESCRIPTION
Value	The value to extract and represent in the observation. For more information, see Value Type Templates .
Components	<i>Optional:</i> One or more components to create on the observation.
Components[].Codes	One or more Codings to apply to the component.
Components[].Value	The value to extract and represent in the component. For more information, see Value Type Templates .

Value type templates

Below are the currently supported value type templates. In the future, further templates may be added.

SampledData

Represents the [SampledData](#) FHIR data type. Observation measurements are written to a value stream starting at a point in time and incrementing forward using the period defined. If no value is present, an `E` will be written into the data stream. If the period is such that two more values occupy the same position in the data stream, the latest value is used. The same logic is applied when an observation using the SampledData is updated.

PROPERTY	DESCRIPTION
DefaultPeriod	The default period in milliseconds to use.
Unit	The unit to set on the origin of the SampledData.

Quantity

Represents the [Quantity](#) FHIR data type. If more than one value is present in the grouping, only the first value is used. When new value arrives that maps to the same observation it will overwrite the old value.

PROPERTY	DESCRIPTION
Unit	Unit representation.
Code	Coded form of the unit.
System	System that defines the coded unit form.

CodeableConcept

Represents the [CodeableConcept](#) FHIR data type. The actual value isn't used.

PROPERTY	DESCRIPTION
Text	Plain text representation.
Codes	One or more Codings to apply to the observation created.
Codes[].Code	The code for the Coding .
Codes[].System	The system for the Coding .

PROPERTY	DESCRIPTION
<code>Codes[].Display</code>	The display for the Coding .

Examples

Heart rate - SampledData

```
{
  "templateType": "CodeValueFhir",
  "template": {
    "codes": [
      {
        "code": "8867-4",
        "system": "http://loinc.org",
        "display": "Heart rate"
      }
    ],
    "periodInterval": 60,
    "typeName": "heartrate",
    "value": {
      "defaultPeriod": 5000,
      "unit": "count/min",
      "valueName": "hr",
      "valueType": "SampledData"
    }
  }
}
```

Steps - SampledData

```
{
  "templateType": "CodeValueFhir",
  "template": {
    "codes": [
      {
        "code": "55423-8",
        "system": "http://loinc.org",
        "display": "Number of steps"
      }
    ],
    "periodInterval": 60,
    "typeName": "stepsCount",
    "value": {
      "defaultPeriod": 5000,
      "unit": "",
      "valueName": "steps",
      "valueType": "SampledData"
    }
  }
}
```

Blood pressure - SampledData

```

{
  "templateType": "CodeValueFhir",
  "template": {
    "codes": [
      {
        "code": "85354-9",
        "display": "Blood pressure panel with all children optional",
        "system": "http://loinc.org"
      }
    ],
    "periodInterval": 60,
    "typeName": "bloodpressure",
    "components": [
      {
        "codes": [
          {
            "code": "8867-4",
            "display": "Diastolic blood pressure",
            "system": "http://loinc.org"
          }
        ],
        "value": {
          "defaultPeriod": 5000,
          "unit": "mmHg",
          "valueName": "diastolic",
          "valueType": "SampledData"
        }
      },
      {
        "codes": [
          {
            "code": "8480-6",
            "display": "Systolic blood pressure",
            "system": "http://loinc.org"
          }
        ],
        "value": {
          "defaultPeriod": 5000,
          "unit": "mmHg",
          "valueName": "systolic",
          "valueType": "SampledData"
        }
      }
    ]
  }
}

```

Blood pressure - Quantity

```

{
  "templateType": "CodeValueFhir",
  "template": {
    "codes": [
      {
        "code": "85354-9",
        "display": "Blood pressure panel with all children optional",
        "system": "http://loinc.org"
      }
    ],
    "periodInterval": 0,
    "typeName": "bloodpressure",
    "components": [
      {
        "codes": [
          {
            "code": "8867-4",
            "display": "Diastolic blood pressure",
            "system": "http://loinc.org"
          }
        ],
        "value": {
          "unit": "mmHg",
          "valueName": "diastolic",
          "valueType": "Quantity"
        }
      },
      {
        "codes": [
          {
            "code": "8480-6",
            "display": "Systolic blood pressure",
            "system": "http://loinc.org"
          }
        ],
        "value": {
          "unit": "mmHg",
          "valueName": "systolic",
          "valueType": "Quantity"
        }
      }
    ]
  }
}

```

Device removed - CodeableConcept

```
{
  "templateType": "CodeValueFhir",
  "template": {
    "codes": [
      {
        "code": "deviceEvent",
        "system": "https://www.mydevice.com/v1",
        "display": "Device Event"
      }
    ],
    "periodInterval": 0,
    "typeName": "deviceRemoved",
    "value": {
      "text": "Device Removed",
      "codes": [
        {
          "code": "deviceRemoved",
          "system": "https://www.mydevice.com/v1",
          "display": "Device Removed"
        }
      ],
      "valueName": "deviceRemoved",
      "valueType": "CodeableConcept"
    }
  }
}
```

Next steps

Check out frequently asked questions on Azure IoT Connector for FHIR (preview).

[Azure IoT Connector for FHIR FAQs](#)

*In the Azure portal, Azure IoT Connector for FHIR is referred to as IoT Connector (preview). FHIR is a registered trademark of HL7 and is used with the permission of HL7.

Azure Policy Regulatory Compliance controls for Azure API for FHIR

5/14/2021 • 2 minutes to read • [Edit Online](#)

[Regulatory Compliance in Azure Policy](#) provides Microsoft created and managed initiative definitions, known as *built-ins*, for the **compliance domains** and **security controls** related to different compliance standards. This page lists the **compliance domains** and **security controls** for Azure API for FHIR. You can assign the built-ins for a **security control** individually to help make your Azure resources compliant with the specific standard.

The title of each built-in policy definition links to the policy definition in the Azure portal. Use the link in the **Policy Version** column to view the source on the [Azure Policy GitHub repo](#).

IMPORTANT

Each control below is associated with one or more [Azure Policy](#) definitions. These policies may help you [assess compliance](#) with the control; however, there often is not a one-to-one or complete match between a control and one or more policies. As such, **Compliant** in Azure Policy refers only to the policies themselves; this doesn't ensure you're fully compliant with all requirements of a control. In addition, the compliance standard includes controls that aren't addressed by any Azure Policy definitions at this time. Therefore, compliance in Azure Policy is only a partial view of your overall compliance status. The associations between controls and Azure Policy Regulatory Compliance definitions for these compliance standards may change over time.

CMMC Level 3

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - CMMC Level 3](#). For more information about this compliance standard, see [Cybersecurity Maturity Model Certification \(CMMC\)](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Access Control	AC.1.001	Limit information system access to authorized users, processes acting on behalf of authorized users, and devices (including other information systems).	CORS should not allow every domain to access your API for FHIR	1.0.0
Access Control	AC.1.002	Limit information system access to the types of transactions and functions that authorized users are permitted to execute.	CORS should not allow every domain to access your API for FHIR	1.0.0
Access Control	AC.2.016	Control the flow of CUI in accordance with approved authorizations.	CORS should not allow every domain to access your API for FHIR	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Configuration Management	CM.3.068	Restrict, disable, or prevent the use of nonessential programs, functions, ports, protocols, and services.	CORS should not allow every domain to access your API for FHIR	1.0.0
System and Communications Protection	SC.3.177	Employ FIPS-validated cryptography when used to protect the confidentiality of CUI.	Azure API for FHIR should use a customer-managed key to encrypt data at rest	1.0.1
System and Communications Protection	SC.3.183	Deny network communications traffic by default and allow network communications traffic by exception (i.e., deny all, permit by exception).	CORS should not allow every domain to access your API for FHIR	1.0.0

Next steps

- Learn more about [Azure Policy Regulatory Compliance](#).
- See the built-ins on the [Azure Policy GitHub repo](#).

Frequently asked questions about the Azure API for FHIR

6/8/2021 • 7 minutes to read • [Edit Online](#)

Azure API for FHIR: The Basics

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](https://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 is the Recovery Point Objective (RPO) for the Azure API for FHIR?

The Azure API for FHIR is backed by Cosmos DB as our persistence provider. Because of this, the RPO for the service equals [Cosmos DB \(single region\)](#) and is < 240 minutes.

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 FHIR versions (i.e. STU3 to R4) in the [version history for HL7 FHIR](#).

Azure IoT Connector for FHIR (preview) currently supports only FHIR version R4, and is visible only on R4 instances of Azure API for FHIR.

What's the difference between '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 run the FHIR Server for Azure, you have direct access to the underlying services, but are responsible for maintaining and updating the server and all required compliance work if you're storing PHI data.

For a development standpoint, every feature that doesn't apply only to the managed service is first deployed to the open-source Microsoft FHIR Server for Azure. 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. This is the same process for all of our services, such as Azure IoT Connector for FHIR (preview).

In which regions is Azure API for FHIR Available?

Currently, we have general availability for both public and government in [multiple geo-regions](#). For information about government cloud services at Microsoft, check out [Azure services by FedRAMP](#).

Where can I see what is releasing into the Azure API for FHIR?

To see some of what is releasing into the Azure API for FHIR, please refer to the [release](#) of the open-source FHIR Server. We have worked to tag items with Azure-API-for-FHIR if they will release to the managed service and are usually available two weeks after they are on the release page in open-source. We have also included instructions on how to test the build [here](#) if you would like to test in your own environment. We are evaluating how to best share additional managed service updates.

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](#).

Where can I find what version of FHIR is running on my database.

You can find the exact FHIR version exposed in the capability statement under the "fhirVersion" property.

FHIR Implementations and Specifications

Can I create a custom FHIR resource?

We do not allow custom FHIR resources. If you need a custom FHIR resource, you can build a custom resource on top of the [Basic resource](#) with extensions.

Are [extensions](#) supported on Azure API for FHIR?

We allow you to load any valid FHIR JSON data into the server. If you want to store the structure definition that defines the extension, you could save this as a structure definition resource. To search on extensions, you'll need to [define your own search parameters](#).

What is the limit on `_count`?

The current limit on `_count` is 1000. If you set `_count` to more than 1000, you'll receive a warning in the bundle that only 1000 records will be shown.

Are there any limitations on the Group Export functionality?

For Group Export we only export the included references from the group, not all the characteristics of the [group resource](#).

Can I post a bundle to the Azure API for FHIR?

We currently support posting [batch bundles](#) but do not support posting transaction bundles in the Azure API for FHIR. You can use the open-source FHIR Server backed by SQL to post transaction bundles.

How can I get all resources for a single patient in the Azure API for FHIR?

We support [compartment search](#) in the Azure API for FHIR. This allows you to get all the resources related to a specific patient. Note that right now compartment includes all the resources related to the patient but not the patient itself so you will need to also search to get the patient if you need the patient resource in your results.

Some examples of this are below:

- GET Patient/*
- GET Patient//Observation
- GET Patient//Observation?code=8302-2

What is the default sort when searching for resources in Azure API for FHIR?

We support sorting by the date last updated: `_sort=_lastUpdated`. For more information about other supported search parameters, see [Overview of FHIR Search](#).

Does the Azure API for FHIR support \$everything?

No. At this time we do not support \$everything. However it can be achieved with two API calls. For example to get Patient\$everything, you can first grab the patient record using /Patient/[ID] and then a second call to retrieve all the patient data using /Patient/[ID]/*.

You can see more details at this [community post](#).

How does \$export work?

\$export is part of the FHIR specification: <https://hl7.org/fhir/uv/bulkdata/export/index.html>. If the FHIR service is configured with a managed identity and a storage account, and if the managed identity has access to that storage account - you can simply call \$export on the FHIR API and all the FHIR resources will be exported to the storage account. For more information, check out our [article on \\$export](#).

Is de-identified export available at Patient and Group level as well?

Anonymized export is currently supported only on a full system export (/ \$export), and not for Patient export (/Patient/ \$export). We are working on making it available at the Patient level as well.

Using Azure API for FHIR

How do I enable log analytics for Azure API for FHIR?

We enable diagnostic logging and allow reviewing sample queries for these logs. For details on enabling audit logs and sample queries, check out [this section](#). If you want to include additional information in the logs, check out [using custom HTTP headers](#).

Where can I see some examples of using the Azure API for FHIR within a workflow?

We have a collection of reference architectures available on the [Health Architecture GitHub page](#).

Where can I see an example of connecting a web application to Azure API for FHIR?

We have a [Health Architecture GitHub page](#) that contains example applications and scenarios. It illustrates how to connect a web application to Azure API for FHIR.

Azure API for FHIR Features and Services

Is there a way to encrypt my data using my personal key not a default key?

Yes, Azure API for FHIR allows configuring customer-managed keys, leveraging support from Cosmos DB. For more information about encrypting your data with a personal key, check out [this section](#).

Azure API for FHIR: Preview Features

Can I configure scaling capacity for Azure IoT Connector for FHIR (preview)?

Since Azure IoT Connector for FHIR is free of charge during public preview, its scaling capacity is fixed and limited. Azure IoT Connector for FHIR configuration available in public preview is expected to provide a throughput of about 200 messages per second. Some form of resource capacity configuration will be made available in General Availability (GA).

Why can't I install Azure IoT Connector for FHIR (preview) when Private Link is enabled on Azure API for FHIR?

Azure IoT Connector for FHIR doesn't support Private Link capability at this time. Hence, if you have Private Link enabled on Azure API for FHIR, you can't install Azure IoT Connector for FHIR and vice-versa. This limitation is expected to go away when Azure IoT Connector for FHIR is available for General Availability (GA).

Features

6/8/2021 • 4 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: `4.0.1`

Previous versions also currently supported include: `3.0.2`

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	See Note below.
delete (conditional)	Yes	Yes	Yes	
history	Yes	Yes	Yes	
create	Yes	Yes	Yes	Support both POST/PUT
create (conditional)	Yes	Yes	Yes	Issue #1382
search	Partial	Partial	Partial	See Overview of FHIR Search .
chained search	Partial	Yes	Partial	See Note 2 below.
reverse chained search	Partial	Yes	Partial	See Note 2 below.
capabilities	Yes	Yes	Yes	

API	SUPPORTED - PAAS	SUPPORTED - OSS (SQL)	SUPPORTED - OSS (COSMOS DB)	COMMENT
batch	Yes	Yes	Yes	
transaction	No	Yes	No	
paging	Partial	Partial	Partial	<code>self</code> and <code>next</code> are supported
intermediaries	No	No	No	

NOTE

Delete defined by the FHIR spec requires that after deleting, subsequent non-version specific reads of a resource returns a 410 HTTP status code and the resource is no longer found through searching. The Azure API for FHIR also enables you to fully delete (including all history) the resource. To fully delete the resource, you can pass a parameter settings

`hardDelete` to true (`DELETE {server}/{resource}/{id}?hardDelete=true`). If you do not pass this parameter or set `hardDelete` to false, the historic versions of the resource will still be available.

Note 2

- Adds MVP support for Chained and Reverse Chained FHIR Search in CosmosDB.

In the Azure API for FHIR and the open-source FHIR server backed by Cosmos, the chained search and reverse chained search is an MVP implementation. To accomplish chained search on Cosmos DB, the implementation walks down the search expression and issues sub-queries to resolve the matched resources. This is done for each level of the expression. If any query returns more than 100 results, an error will be thrown. By default, chained search is behind a feature flag. To use the chained searching on Cosmos DB, use the header `x-ms-enable-chained-search: true`. For more details, see [PR 1695](#).

Extended Operations

All the operations that are supported that extend the RESTful API.

SEARCH PARAMETER TYPE	SUPPORTED - PAAS	SUPPORTED - OSS (SQL)	SUPPORTED - OSS (COSMOS DB)	COMMENT
\$export (whole system)	Yes	Yes	Yes	
Patient/\$export	Yes	Yes	Yes	
Group/\$export	Yes	Yes	Yes	
\$convert-data	Yes	Yes	Yes	
\$validate	Yes	Yes	Yes	
\$member-match	Yes	Yes	Yes	
\$patient-everything	No	No	Yes	

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.

Service limits

- **Request Units (RUs)** - You can configure up to 10,000 RUs in the portal for Azure API for FHIR. You will need a minimum of 400 RUs or 40 RUs/GB, whichever is larger. If you need more than 10,000 RUs, you can put in a support ticket to have this increased. The maximum available is 1,000,000.
- **Bundle size** - Each bundle is limited to 500 items.
- **Data size** - Data/Documents must each be slightly less than 2 MB.
- **Subscription Limit** - By default, each subscription is limited to a maximum of 10 FHIR Server Instances. If you need more instances per subscription, open a support ticket and provide details about your needs.
- **Concurrent connections and Instances** - By default, you have 15 concurrent connections on two instances in the cluster (for a total of 30 concurrent requests). If you need more concurrent requests, open a support ticket and provide details about your needs.

Performance expectations

The performance of the system is dependent on the number of RUs, concurrent connections, and the type of operations you're performing (Put, Post, etc.). Below are some general ranges of what you can expect based on configured RUs. In general, performance scales linearly with an increase in RUs:

# OF RUS	RESOURCES/SEC	MAX STORAGE (GB)*
400	5-10	10
1,000	100-150	25
10,000	225-400	250
100,000	2,500-4,000	2,500

Note: Per Cosmos DB requirement, there is a requirement of a minimum throughput of 40 RU/s per GB of storage.

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](#)

Related GitHub Projects

3/24/2021 • 2 minutes to read • [Edit Online](#)

We have many open-source projects on GitHub that provide you the source code and instructions to deploy services for various uses. You are always welcome to visit our GitHub repositories to learn and experiment with our features and products.

FHIR Server

- [microsoft/fhir-server](#): open-source FHIR Server, which is the basis for Azure API for FHIR
- To see the latest releases, please refer to [Release Notes](#)
- [microsoft/fhir-server-samples](#): a sample environment

Data Conversion & Anonymization

FHIR Converter

- [microsoft/FHIR-Converter](#): a conversion utility to translate legacy data formats into FHIR
- Integrated with the Azure API for FHIR as well as FHIR server for Azure in the form of \$convert-data operation
- Ongoing improvements in OSS, and continual integration to the FHIR servers

FHIR Converter - VS Code Extension

- [microsoft/FHIR-Tools-for-Anonymization](#): a set of tools for helping with data (in FHIR format) anonymization
- Integrated with the Azure API for FHIR as well as FHIR server for Azure in the form of 'de-identified export'

FHIR Tools for Anonymization

- [microsoft/vscode-azurehealthcareapis-tools](#): a VS Code extension that contains a collection of tools to work with Azure Healthcare APIs
- Released to Visual Studio Marketplace
- Used for authoring Liquid templates to be used in the FHIR Converter

IoT Connector

Integration with IoT Hub and IoT Central

- [microsoft/iomt-fhir](#): integration with IoT Hub or IoT Central to FHIR with data normalization and FHIR conversion of the normalized data
- Normalization: device data information is extracted into a common format for further processing
- FHIR Conversion: normalized and grouped data is mapped to FHIR. Observations are created or updated according to configured templates and linked to the device and patient.
- [Tools to help build the conversation map](#): visualize the mapping configuration for normalizing the device input data and transform it to the FHIR resources. Developers can use this tool to edit and test the mappings, device mapping and FHIR mapping, and export them for uploading to the IoT Connector in the Azure portal.

HealthKit and FHIR Integration

- [microsoft/healthkit-on-fhir](#): a Swift library that automates the export of Apple HealthKit Data to a FHIR Server

Partner ecosystem for Azure API for FHIR

3/11/2021 • 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/REGIONS	CONTACT
Medal	De-identification, Legacy-FHIR conversion	USA	Contact
Rhapsody	Legacy-FHIR conversion	USA, Australia, New Zealand	Contact
iINTERFACEWARE	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

Azure Policy built-in definitions for Azure API for FHIR

5/14/2021 • 2 minutes to read • [Edit Online](#)

This page is an index of [Azure Policy](#) built-in policy definitions for Azure API for FHIR. For additional Azure Policy built-ins for other services, see [Azure Policy built-in definitions](#).

The name of each built-in policy definition links to the policy definition in the Azure portal. Use the link in the **Version** column to view the source on the [Azure Policy GitHub repo](#).

Azure API for FHIR

NAME (AZURE PORTAL)	DESCRIPTION	EFFECT(S)	VERSION (GITHUB)
Azure API for FHIR should use a customer-managed key to encrypt data at rest	Use a customer-managed key to control the encryption at rest of the data stored in Azure API for FHIR when this is a regulatory or compliance requirement. Customer-managed keys also deliver double encryption by adding a second layer of encryption on top of the default one done with service-managed keys.	audit, disabled	1.0.1
Azure API for FHIR should use private link	Azure API for FHIR should have at least one approved private endpoint connection. Clients in a virtual network can securely access resources that have private endpoint connections through private links. For more information, visit: https://aka.ms/fhir-privatelink .	Audit, Disabled	1.0.0
CORS should not allow every domain to access your API for FHIR	Cross-Origin Resource Sharing (CORS) should not allow all domains to access your API for FHIR. To protect your API for FHIR, remove access for all domains and explicitly define the domains allowed to connect.	audit, disabled	1.0.0

Next steps

- See the built-ins on the [Azure Policy GitHub repo](#).
- Review the [Azure Policy definition structure](#).
- Review [Understanding policy effects](#).