FHIR service documentation

FHIR service is a managed, standards-based, compliant API for clinical health data that enables solutions for actionable analytics and machine learning.

Get started

OVERVIEW

What is FHIR service?

QUICKSTART

Deploy FHIR service using portal

Deploy FHIR service using ARM template

Access the FHIR service

CONCEPT

Azure AD for FHIR service

Access token validation

TUTORIAL

Access using Postman

FHIR search capabilities

OVERVIEW

Overview of FHIR search

HOW-TO GUIDE

Defining custom search parameters

How to run a reindex job

Search examples

Interoperability and patient access

OVERVIEW

CMS interoperability and patient access rule

HOW-TO GUIDE

CARIN Implementation Guide for Blue Button

Da Vinci Drug Formulary

FHIR Operations

HOW-TO GUIDE

$member-match operation

Patient/$everything

Profile validation

$purge-history operation

Export and Convert Data

HOW-TO GUIDE

Configure settings for export

Configure settings for bulk-import

Convert data

Copy data to Synapse

Export data

What is the FHIR service?

Article • 09/01/2023

The FHIR service in Azure Health Data Services enables rapid exchange of health data using the Fast Healthcare Interoperability Resources (FHIR®) data standard. As part of a managed Platform-as-a-Service (PaaS), the FHIR service makes it easy for anyone working with health data to securely store and exchange Protected Health Information (PHI) in the cloud.

The FHIR service offers the following:

- Managed FHIR-compliant server, provisioned in the cloud in minutes
- Enterprise-grade FHIR API endpoint for FHIR data access and storage
- High performance, low latency
- Secure management of Protected Health Information (PHI) in a compliant cloud environment
- SMART on FHIR for mobile and web clients
- Controlled access to FHIR data at scale with Azure Active Directory Role-Based Access Control (RBAC)
- Audit log tracking for access, creation, and modification events within the FHIR service data store

The FHIR service allows you to quickly create and deploy a FHIR server to leverage the elastic scale of the cloud for ingesting, persisting, and querying FHIR data. The Azure services that power the FHIR service are designed for high performance no matter how much data you're working with.

The FHIR API provisioned in the FHIR service enables any FHIR-compliant system to securely connect and interact with FHIR data. As a PaaS offering, Microsoft takes on the operations, maintenance, update, and compliance requirements for the FHIR service so you can free up your own operational and development resources.

Leveraging the power of your data with FHIR

The healthcare industry is rapidly adopting FHIR® as the industry-wide standard for health data storage, querying, and exchange. FHIR provides a robust, extensible data model with standardized semantics that all FHIR-compliant systems can use interchangeably. With FHIR, organizations can unify disparate electronic health record systems (EHRs) and other health data repositories – allowing all data to be persisted and exchanged in a single, universal format. With the addition of SMART on FHIR, user-facing mobile and web-based applications can securely interact with FHIR data –

opening a new range of possibilities for patient and provider access to PHI. Most of all, FHIR simplifies the process of assembling large health datasets for research – enabling researchers and clinicians to apply machine learning and analytics at scale for gaining new health insights.

Securely manage health data in the cloud

The FHIR service in Azure Health Data Services makes FHIR data available to clients through a RESTful API. This API is an implementation of the HL7 FHIR API specification. As a managed PaaS offering in Azure, the FHIR service gives organizations a scalable and secure environment for the storage and exchange of Protected Health Information (PHI) in the native FHIR format.

Free up your resources to innovate

You could invest resources building and maintaining your own FHIR server, but with the FHIR service in Azure Health Data Services, Microsoft handles setting up the server's components, ensuring all compliance requirements are met so you can focus on building innovative solutions.

Enable interoperability with FHIR

The FHIR service enables connection with any health data system or application capable of sending FHIR API requests. Coupled with other parts of the Azure ecosystem, the FHIR service forms a link between electronic health records systems (EHRs) and Azure's powerful suite of data analytics and machine learning tools – enabling organizations to build patient and provider-facing applications that harness the full power of the Microsoft cloud.

Control Data Access at Scale

With the FHIR service, you control your data – at scale. The FHIR service's Role-Based Access Control (RBAC) is rooted in Azure AD identities management, which means you can grant or deny access to health data based on the roles given to individuals in your organization. These RBAC settings for the FHIR service are configurable in Azure Health Data Services at the workspace level. This simplifies system management and guarantees your organization's PHI is safe within a HIPAA and HITRUST-compliant environment.

Secure your data

As part of the Azure family of services, the FHIR service protects your organization's PHI with an unparalleled level of security. In Azure Health Data Services, your FHIR data is isolated to a unique database per FHIR service instance and protected with multi-region failover. On top of this, FHIR service implements a layered, in-depth defense and advanced threat detection for your data – giving you peace of mind that your organization's PHI is guarded by Azure's industry-leading security.

Applications for the FHIR service

FHIR servers are essential for interoperability of health data. The FHIR service is designed as a managed FHIR server with a RESTful API for connecting to a broad range of client systems and applications. Some of the key use cases for the FHIR service are listed below:

- **Startup App Development:** Customers developing a patient- or provider-centric app (mobile or web) can leverage FHIR service as a fully managed backend for health data transactions. The FHIR service enables secure transfer of PHI, and with SMART on FHIR, app developers can take advantage of the robust identities management in Azure AD for authorization of FHIR RESTful API actions.

- **Healthcare Ecosystems:** While EHRs exist as the primary 'source of truth' in many clinical settings, it isn't uncommon for providers to have multiple databases that aren't connected to one another (often because the data is stored in different formats). Utilizing the FHIR service as a conversion layer between these systems allows organizations to standardize data in the FHIR format. Ingesting and persisting in FHIR enables health data querying and exchange across multiple disparate systems.

- **Research:** Health researchers have embraced the FHIR standard as it gives the community a shared data model and removes barriers to assembling large datasets for machine learning and analytics. With the FHIR service's data conversion and PHI de-identification capabilities, researchers can prepare HIPAA-compliant data for secondary use before sending the data to Azure Machine Learning and analytics pipelines. The FHIR service's audit logging and alert mechanisms also play an important role in research workflows.

FHIR platforms from Microsoft

FHIR capabilities from Microsoft are available in three configurations:

- The **FHIR service** is a managed platform as a service (PaaS) that operates as part of Azure Health Data Services. In addition to the FHIR service, Azure Health Data Services includes managed services for other types of health data such as the DICOM service for medical imaging data and the MedTech service for medical IoT data. All services (FHIR service, DICOM service, and MedTech service) can be connected and administered within an Azure Health Data Services workspace.
- **Azure API for FHIR** is a managed FHIR server offered as a PaaS in Azure – easily provisioned in the Azure portal. Azure API for FHIR is not part of Azure Health Data Services and lacks some of the features of the FHIR service.
- **FHIR Server for Azure**, an open-source FHIR server that can be deployed into your Azure subscription, is available on GitHub at https://github.com/Microsoft/fhir-server .

For use cases that require customizing a FHIR server with admin access to the underlying services (e.g., access to the database without going through the FHIR API), developers should choose the open-source FHIR Server for Azure. For implementation of a turnkey, production-ready FHIR API with a provisioned database backend (i.e., data can only be accessed through the FHIR API - not the database directly), developers should choose the FHIR service.

Next Steps

To start working with the FHIR service, follow the 5-minute quickstart instructions for FHIR service deployment.

Deploy FHIR service

Autoscaling

Article • 03/02/2023

Azure Health Data Services provides a managed service for persisting FHIR-compliant healthcare data and interacting with it securely through the API service endpoint.

Autoscaling is a capability to dynamically scale FHIR service based on the load reported. The FHIR service in Azure Health Data Services provides the built-in autoscaling capability and the process is automated. This capability provides elasticity and enables provisioning of more instances for FHIR service customers on demand.

The autoscaling feature for FHIR service is available in all regions where the FHIR service is supported.

> ⓘ **Note**
>
> Autoscaling feature is subject to the resources availability in Azure regions.

The autoscaling feature adjusts computing resources automatically to optimize service scalability. There's no action required from customers.

Autoscale at Compute level

- Scaling Trigger

Scaling Trigger describes when scaling of the service is performed. Conditions that are defined in the trigger are checked periodically to determine if a service should be scaled or not. All triggers that are currently supported are Average CPU, Max Worker Thread, Average LogWrite, Average data IO.

- Scaling mechanism

The scaling mechanism is applied if the trigger check determines that scaling is necessary. Additionally, the scaling trigger won't be evaluated again until the scaling interval has expired, which is set to one minute for FHIR service.

To ensure the best possible outcome, we recommend customers to gradually increase their request rate to match the expected push rate, rather than pushing all requests at once.

FAQ

What is the cost to enable autoscaling for FHIR service?

The autoscaling feature incurs no extra costs to customers.

What should customers do if there is high volume of HTTP 429 errors?

We recommend customers to gradually increase the request rate to see if brings reduction in HTTP 429 errors. For consistent 429 errors, we ask customer(s) to create a support ticket through Azure portal on issue observed. Support team will engage to understand scaling trigger needs.

Next steps

In this article, you've learned about the FHIR service autoscaling feature in Azure Health Data Services. For more information about the FHIR service supported features, see

Supported FHIR Features

FHIR REST API capabilities for Azure Health Data Services FHIR service

Article • 07/27/2023

In this article, we'll cover some of the nuances of the RESTful interactions of Azure Health Data Services FHIR service (hereby called FHIR service).

Conditional create/update

The FHIR service supports create, conditional create, update, and conditional update as defined by the FHIR specification. One useful header in these scenarios is the If-Match header. The `If-Match` header is used and will validate the version being updated before making the update. If the `ETag` doesn't match the expected `ETag`, it will produce the error message *412 Precondition Failed*.

Delete and Conditional Delete

FHIR service offers two delete types. There's Delete, which is also know as Hard + Soft Delete, and Conditional Delete.

Delete (Hard + Soft Delete)

Delete defined by the FHIR specification requires that after deleting a resource, subsequent non-version specific reads of a resource returns a 410 HTTP status code. Therefore, the resource is no longer found through searching. Additionally, the FHIR service 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 {{FHIR_URL}}/{resource}/{id}?hardDelete=true)`. If you don't pass this parameter or set `hardDelete` to false, the historic versions of the resource will still be available.

> **Note**
>
> If you only want to delete the history, FHIR service supports a custom operation called `$purge-history`. This operation allows you to delete the history off of a resource.

Conditional Delete

Conditional Delete allows you to pass search criteria to delete a resource. By default, the Conditional Delete allows you to delete one item at a time. You can also specify the `_count` parameter to delete up to 100 items at a time. Below are some examples of using Conditional Delete.

To delete a single item using Conditional Delete, you must specify search criteria that returns a single item.

```
DELETE https://{{FHIR_URL}}/Patient?identifier=1032704
```

You can do the same search but include `hardDelete=true` to also delete all history.

```
DELETE https://{{FHIR_URL}}/Patient?identifier=1032704&hardDelete=true
```

To delete multiple resources, include `_count=100` parameter. This parameter will delete up to 100 resources that match the search criteria.

```
DELETE https://{{FHIR_URL}}/Patient?identifier=1032704&_count=100
```

Recovery of deleted files

If you don't use the hard delete parameter, then the record(s) in FHIR service should still exist. The record(s) can be found by doing a history search on the resource and looking for the last version with data.

If the ID of the resource that was deleted is known, use the following URL pattern:

```
<FHIR_URL>/<resource-type>/<resource-id>/_history
```

For example: `https://myworkspace-myfhirserver.fhir.azurehealthcareapis.com/Patient/123456789/_history`

If the ID of the resource isn't known, do a history search on the entire resource type:

```
<FHIR_URL>/<resource-type>/_history
```

For example: `https://myworkspace-myfhirserver.fhir.azurehealthcareapis.com/Patient/_history`

After you've found the record you want to restore, use the `PUT` operation to recreate the resource with the same ID, or use the `POST` operation to make a new resource with the same information.

> ⓘ **Note**

There is no time-based expiration for history/soft delete data. The only way to remove history/soft deleted data is with a hard delete or the purge history operation.

Batch and Transaction Bundles

In FHIR, bundles can be considered as a container that holds multiple resources. Batch and transaction bundles enable users to submit a set of actions to be performed on a server in single HTTP request/response. The actions may be performed independently as a "batch", or as a single atomic "transaction" where the entire set of changes succeed or fail as a single entity. Actions on multiple resources of the same or different types may be submitted, such as create, update, or delete. For more information, visit FHIR bundles.

A batch or transaction bundle interaction with FHIR service is performed with HTTP POST command at base URL.

HTTP

```
POST {{fhir_url}}
{
  "resourceType": "Bundle",
  "type": "batch",
  "entry": [
    {
      "resource": {
        "resourceType": "Patient",
        "id": "patient-1",
        "name": [
          {
            "given": ["Alice"],
            "family": "Smith"
          }
        ],
        "gender": "female",
        "birthDate": "1990-05-15"
      },
      "request": {
        "method": "POST",
        "url": "Patient"
      }
    },
    {
      "resource": {
        "resourceType": "Patient",
        "id": "patient-2",
        "name": [
          {
```

```
            "given": ["Bob"],
            "family": "Johnson"
          }
        ],
        "gender": "male",
        "birthDate": "1985-08-23"
      },
      "request": {
        "method": "POST",
        "url": "Patient"
      }
    }
   }
  ]
}
```

In the case of a batch, each entry is treated as an individual interaction or operation. In the case of a transaction bundle, all interactions or operations either succeed or fail together. For a batch, or a successful transaction bundle, the response contains one entry for each entry in the request. For failed transaction bundle, FHIR service returns single OperationOutcome.

> **Note**
>
> For batch bundles there should be no interdependencies between different entries in FHIR bundle. The success or failure of one entry should not impact the success or failure of another entry.

Batch bundle parallel processing in public preview

Currently Batch and transaction bundles are executed serially in FHIR service. To improve performance and throughput, we're enabling parallel processing of batch bundles in public preview.
To use the capability of parallel batch bundle processing-

- Set header “x-bundle-processing-logic” value to “parallel”.
- Ensure there's no overlapping resource ID that is executing on DELETE, POST, PUT or PATCH operations in the same bundle.

> **Important**
>
> Bundle parallel processing is currently in public preview. Preview APIs and SDKs are provided without a service-level agreement. We recommend that you don't use them for production workloads. Some features might not be supported, or they

might have constrained capabilities. For more information, review Supplemental Terms of Use for Microsoft Azure Previews

Patch and Conditional Patch

Patch is a valuable RESTful operation when you need to update only a portion of the FHIR resource. Using patch allows you to specify the element(s) that you want to update in the resource without having to update the entire record. FHIR defines three ways to patch resources: JSON Patch, XML Patch, and FHIRPath Patch. The FHIR Service support both JSON Patch and FHIRPath Patch along with Conditional JSON Patch and Conditional FHIRPath Patch (which allows you to patch a resource based on a search criteria instead of a resource ID). To walk through some examples, refer to the sample FHIRPath Patch REST file and the JSON Patch REST file for each approach. For additional details, read the HL7 documentation for patch operations with FHIR.

> **Note**
>
> When using `PATCH` against STU3, and if you are requesting a History bundle, the patched resource's `Bundle.entry.request.method` is mapped to `PUT`. This is because STU3 doesn't contain a definition for the `PATCH` verb in the **HTTPVerb value set**.

Patch with FHIRPath Patch

This method of patch is the most powerful as it leverages FHIRPath for selecting which element to target. One common scenario is using FHIRPath Patch to update an element in a list without knowing the order of the list. For example, if you want to delete a patient's home telecom information without knowing the index, you can use the example below.

PATCH `http://{FHIR-SERVICE-HOST-NAME}/Patient/{PatientID}`
Content-type: `application/fhir+json`

```
{
    "resourceType": "Parameters",
    "parameter": [
        {
            "name": "operation",
            "part": [
                {
```

```
                    "name": "type",
                    "valueCode": "delete"
                },
                {
                    "name": "path",
                    "valueString": "Patient.telecom.where(use = 'home')"
                }
            ]
        }
    ]
}
```

Any FHIRPath Patch operations must have the `application/fhir+json` Content-Type header set. FHIRPatch Patch supports add, insert, delete, remove, and move operations. FHIRPatch Patch operations also can be easily integrated into Bundles. For more examples, look at the sample FHIRPath Patch REST file.

Patch with JSON Patch

JSON Patch in the FHIR Service conforms to the well-used specification defined by the Internet Engineering Task Force. The payload format does not use FHIR resources and instead uses a JSON document leveraging JSON-Pointers for element selection. JSON Patch is more compact and has a test operation that allows you to validate that a condition is true before doing the patch. For example, if you want to set a patient as deceased only if they're not already marked as deceased, you can use the example below.

PATCH `http://{FHIR-SERVICE-HOST-NAME}/Patient/{PatientID}`
Content-type: `application/json-patch+json`

```
[
    {
        "op": "test",
        "path": "/deceasedBoolean",
        "value": false
    },
    {
        "op": "replace",
        "path": "/deceasedBoolean",
        "value": true
    }
]
```

Any JSON Patch operations must have the `application/json-patch+json` Content-Type header set. JSON Patch supports add, remove, replace, copy, move, and test operations. For more examples, look at the sample JSON Patch REST file.

JSON Patch in Bundles

By default, JSON Patch isn't supported in Bundle resources. This is because a Bundle only supports with FHIR resources and the JSON Patch payload isn't a FHIR resource. To work around this, we'll use Binary resources with a Content-Type of `"application/json-patch+json"` and the base64 encoding of the JSON payload inside of a Bundle. For information about this workaround, view this topic on the FHIR Chat Zulip.

In the example below, we want to change the gender on the patient to female. We've taken the JSON patch `[{"op":"replace","path":"/gender","value":"female"}]` and encoded it to base64.

POST `https://{FHIR-SERVICE-HOST-NAME}/`
Content-Type: `application/json`

```
{
    "resourceType": "Bundle",
    "id": "bundle-batch",
    "type": "batch",
    "entry": [
        {
            "fullUrl": "Patient/{PatientID}",
            "resource": {
                "resourceType": "Binary",
                "contentType": "application/json-patch+json",
                "data":
"W3sib3AiOiJyZXBsYWNlIiwicGF0aCI6Ii9nZW5kZXIiLCJ2YWx1ZSI6ImZlbWFsZSJ9XQ=="
            },
            "request": {
                "method": "PATCH",
                "url": "Patient/{PatientID}"
            }
        }
    ]
}
```

Next steps

In this article, you learned about some of the REST capabilities of the FHIR service. Next, you can learn more about the key aspects to searching resources in the FHIR service.

Overview of FHIR search

Versioning policy and history management

Article • 09/26/2022

The versioning policy in the Azure Health Data Services FHIR service is a configuration, which determines how history is stored for every resource type with the option for resource specific configuration. This policy is directly related to the concept of managing history for FHIR resources.

History in FHIR

History in FHIR gives you the ability to see all previous versions of a resource. History in FHIR can be queried at the resource level, type level, or system level. The HL7 FHIR documentation has more information about the history interaction. History is useful in scenarios where you want to see the evolution of a resource in FHIR or if you want to see the information of a resource at a specific point in time.

All past versions of a resource are considered obsolete and the current version of a resource should be used for normal business workflow operations. However, it can be useful to see the state of a resource as a point in time when a past decision was made.

Versioning policy

Versioning policy in the FHIR service lets you decide how history is stored either at a FHIR service level or at a specific resource level.

There are three different levels for versioning policy:

- `versioned`: History is stored for operation on resources. Resource version is incremented. This is the default.
- `version-update`: History is stored for operation on resources. Resource version is incremented. Updates require a valid `If-Match` header. For more information, see VersionedUpdateExample.http.
- `no-version`: History isn't created for resources. Resource version is incremented.

Versioning policy available to configure at as a system-wide setting and also to override at a resource level. The system-wide setting is used for all resources in your FHIR service, unless a specific resource level versioning policy has been added.

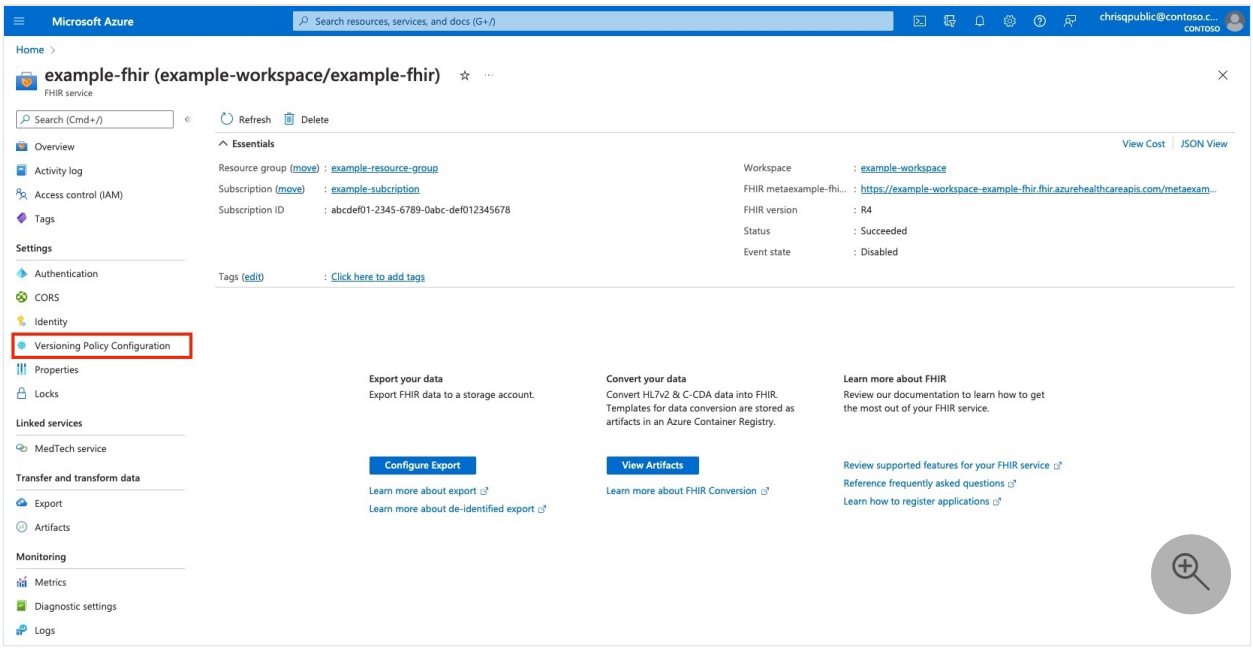
Versioning policy comparison

Policy Value	History Behavior	`meta.versionId` Update Behavior	Default
`versioned`	History is stored	If-Match not required	Yes
`version-update`	History is stored	If-Match required	No
`no-version`	History isn't stored	If-Match not required	No

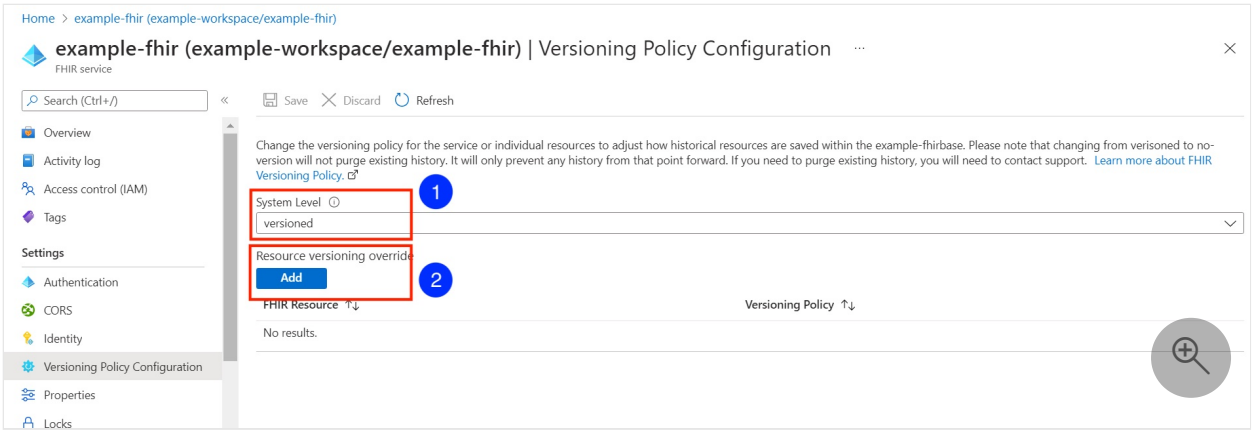
> **Note**
>
> Changing the versioning policy to `no-version` has no effect on existing resource history. If history needs to be removed for resources, use the **$purge-history** operation.

Configuring versioning policy

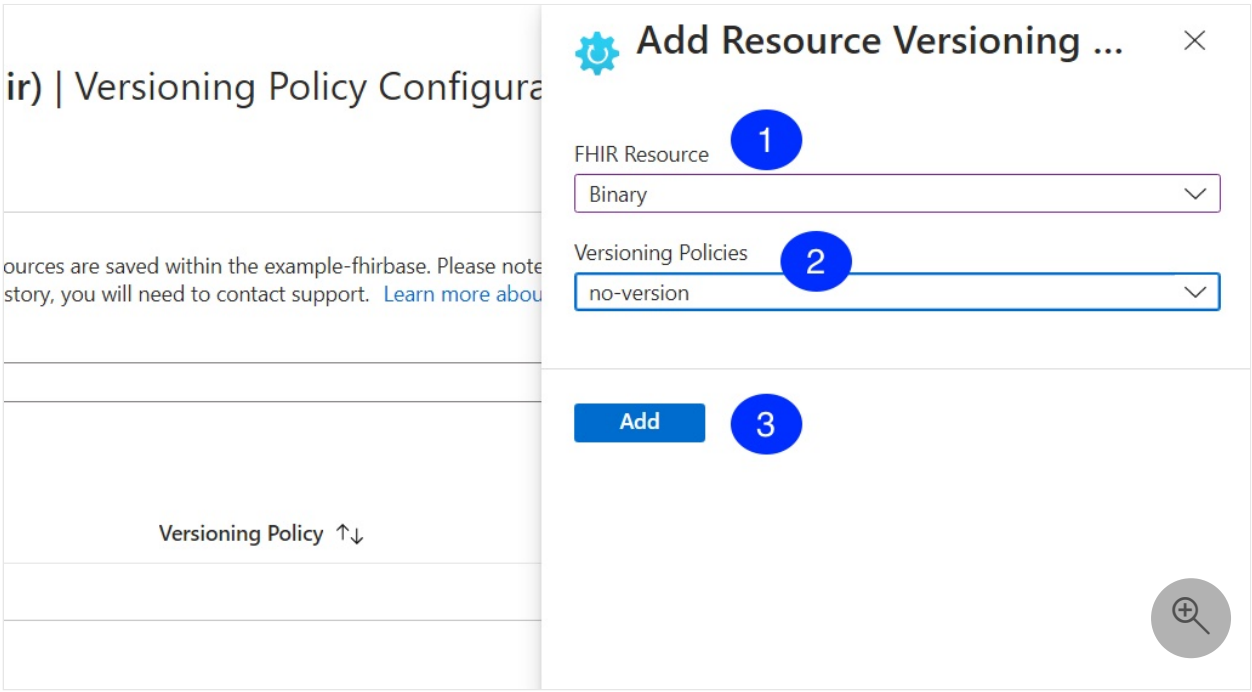
To configure versioning policy, select the **Versioning Policy Configuration** blade inside your FHIR service.



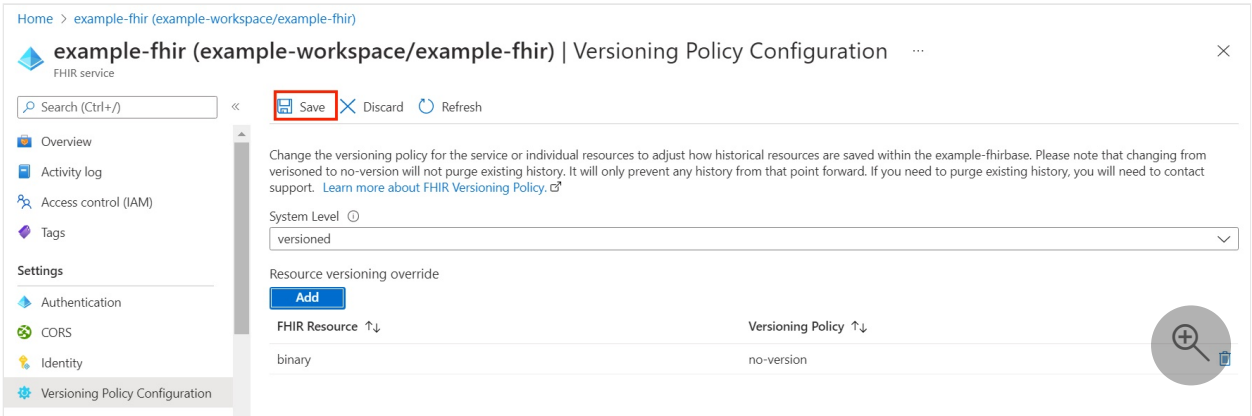
After you've browsed to Versioning Policy Configuration, you'll be able to configure the setting at both system level and the resource level (as an override of the system level). The system level configuration (annotated as 1) will apply to every resource in your FHIR service unless a resource specific override (annotated at 2) has been configured.



When configuring resource level configuration, you'll be able to select the FHIR resource type (annotated as 1) and the specific versioning policy for this specific resource (annotated as 2). Make sure to select the **Add** button (annotated as 3) to queue up this setting for saving.



Make sure to select **Save** after you've completed your versioning policy configuration.



History management

History in FHIR is important for end users to see how a resource has changed over time. It's also useful in coordination with audit logs to see the state of a resource before and after a user modified it. In general, it's recommended to keep history for a resource unless you know that the history isn't needed. Frequent updates of resources can result in a large amount of data storage, which can be undesired in FHIR services with a large amount of data.

Changing the versioning policy either at a system level or resource level won't remove the existing history for any resources in your FHIR service. If you're looking to reduce the history data size in your FHIR service, you must use the $purge-history operation.

Next steps

In this article, you learned how to purge the history for resources in the FHIR service. For more information about how to disable history and some concepts about history management, see

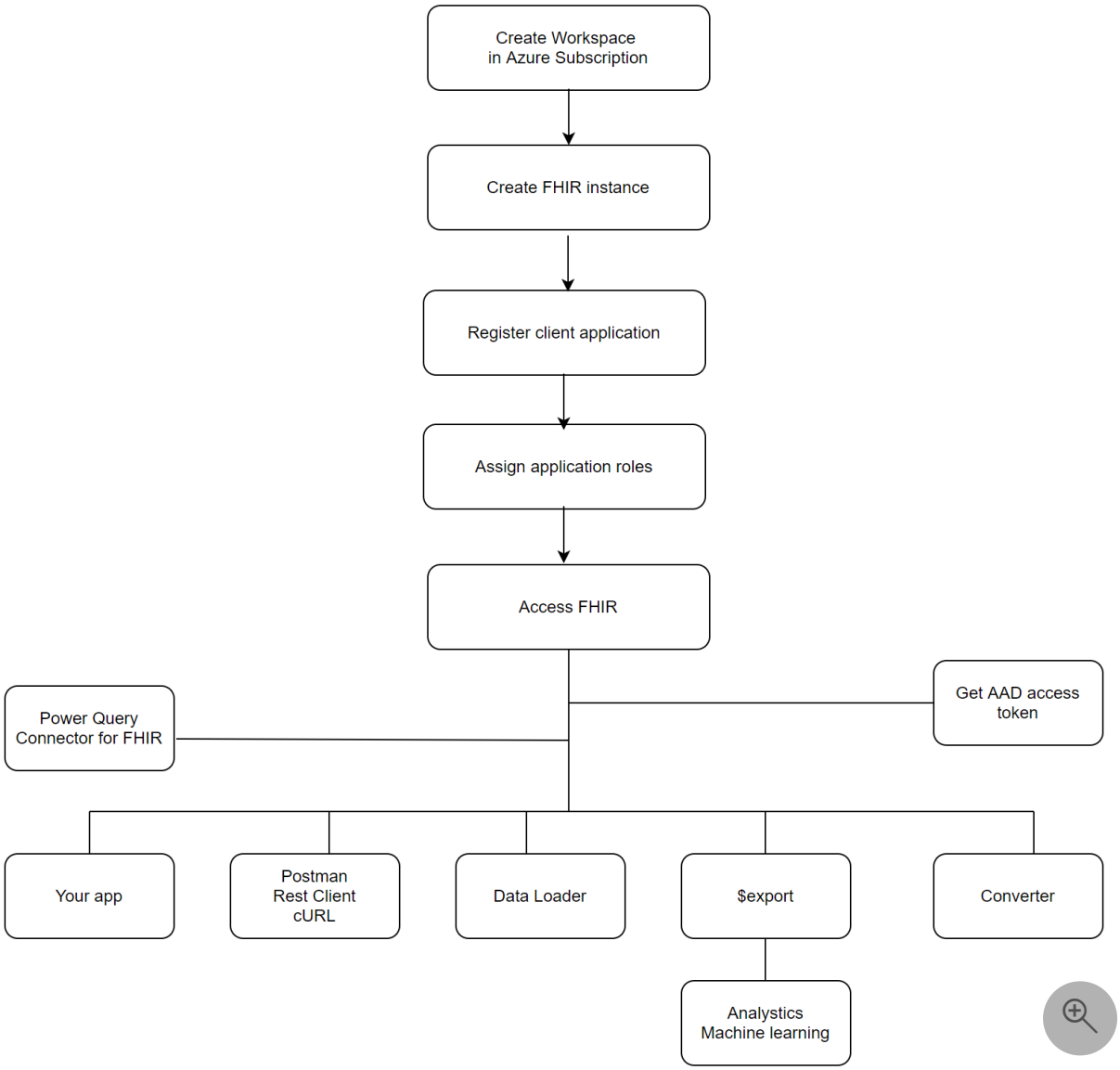
Purge history operation

Get started with FHIR service

Article • 09/26/2022

This article outlines the basic steps to get started with the FHIR service in Azure Health Data Services.

As a prerequisite, you'll need an Azure subscription and have been granted proper permissions to create Azure resource groups and deploy Azure resources. You can follow all the steps, or skip some if you have an existing environment. Also, you can combine all the steps and complete them in PowerShell, Azure CLI, and REST API scripts.



Create a workspace in your Azure subscription

You can create a workspace from the Azure portal, or using PowerShell, Azure CLI, and REST API. You can find scripts from the Azure Health Data Services samples.

> **Note**
>
> There are limits to the number of workspaces and the number of FHIR service instances you can create in each Azure subscription.

Create a FHIR service in the workspace

You can create a FHIR service instance from the Azure portal, or using PowerShell, Azure CLI, and REST API. You can find scripts from the Azure Health Data Services samples.

Optionally, you can create a DICOM service and MedTech service in the workspace.

Access the FHIR service

The FHIR service is secured by Azure Active Directory (Azure AD) that can't be disabled. To access the service API, you must create a client application that's also referred to as a service principal in Azure AD and grant it with the right permissions.

Register a client application

You can create or register a client application from the Azure portal, or using PowerShell and Azure CLI scripts. This client application can be used for one or more FHIR service instances. It can also be used for other services in Azure Health Data Services.

If the client application is created with a certificate or client secret, ensure that you renew the certificate or client secret before expiration and replace the client credentials in your applications.

You can delete a client application. Before you delete a client application, ensure that it's not used in production, dev, test, or quality assurance (QA) environments.

Grant access permissions

You can grant access permissions or assign roles from the Azure portal, or using PowerShell and Azure CLI scripts.

Perform create, read, update, and delete (CRUD) transactions

You can perform create, read (search), update, and delete (CRUD) transactions against the FHIR service in your applications or by using tools such as Postman, REST Client, and cURL. Because the FHIR service is secured by default, you must obtain an access token and include it in your transaction request.

Get an access token

You can obtain an Azure AD access token using PowerShell, Azure CLI, REST CCI, or .NET SDK. For more information, see Get access token.

Access using existing tools

- Postman
- REST Client
- cURL

Load data

You can load data directly using the POST or PUT method against the FHIR service. To bulk load data, you can use one of the Open Source tools listed below.

- FHIR Loader This is a .NET console app and loads data stored in Azure storage to the FHIR service. It's a single thread app, but you can run multiple copies locally or in a Docker container.
- FHIR Bulk Loader This tool is an Azure function app (microservice) and runs in parallel threads.
- Bulk import This tool works with the Open Source FHIR server only. However, it will be available for Azure Health Data Services in the future.

CMS, search, profile validation, and reindex

You can find more details on interoperability and patient access, search, profile validation, and reindex in the FHIR service documentation.

Export data

Optionally, you can export ($export) data to Azure Storage and use it in your analytics or machine-learning projects. You can export the data "as-is" or de-id in `ndjson` format.

You can also export data to Synapse using the Open Source project. In the future, this feature will be integrated to the managed service.

Converting data

Optionally, you can convert HL7 v2 and other format data to FHIR.

Using FHIR data in Power BI Dashboard

Optionally, you can create Power BI dashboard reports with FHIR data.

- Power Query connector for FHIR
- MedTech service and Microsoft Power BI

Next steps

This article described the basic steps to get started using the FHIR service. For information about deploying FHIR service in the Azure Health Data Services workspace, see

Deploy a FHIR service within Azure Health Data Services

Deploy a FHIR service within Azure Health Data Services - using portal

Article • 09/26/2022

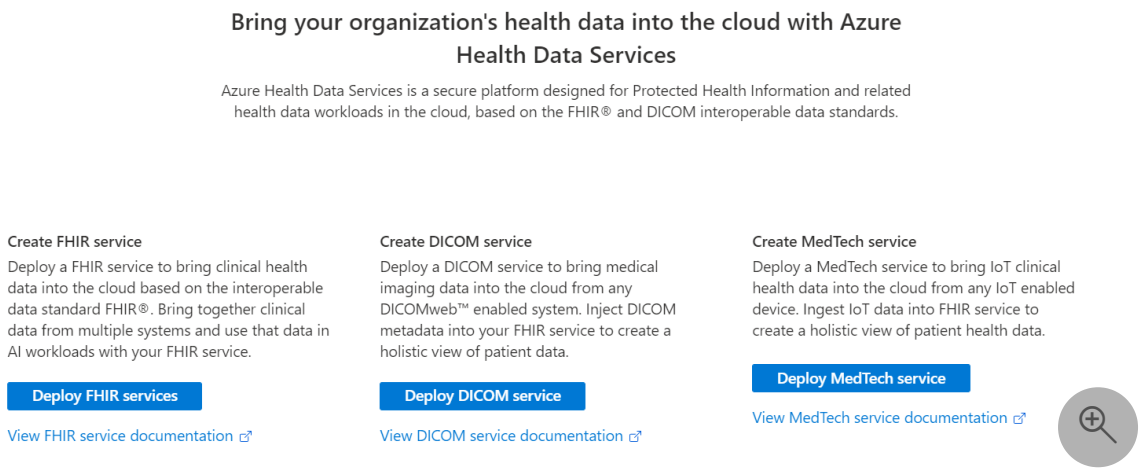
In this article, you'll learn how to deploy FHIR service within Azure Health Data Services (hereby called the FHIR service) using the Azure portal.

Prerequisite

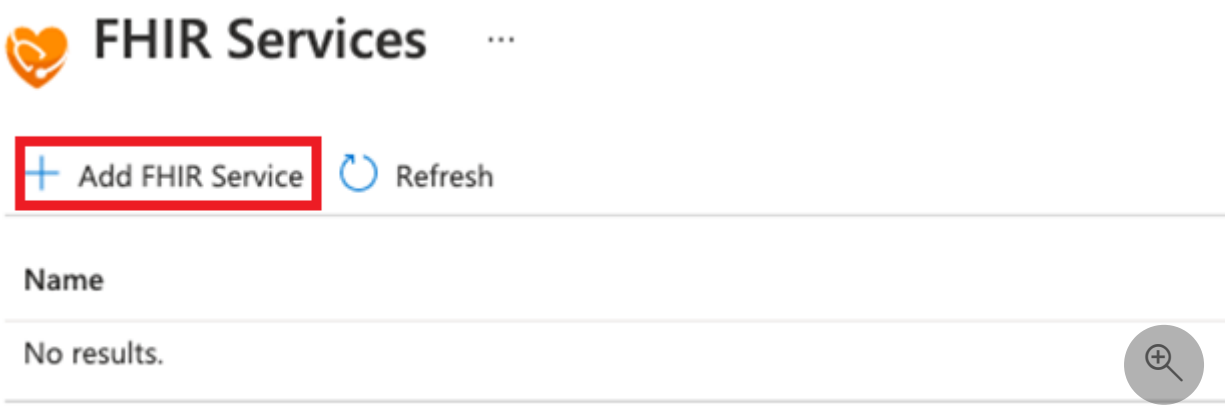
Before getting started, you should have already deployed Azure Health Data Services. For more information about deploying Azure Health Data Services, see Deploy workspace in the Azure portal.

Create a new FHIR service

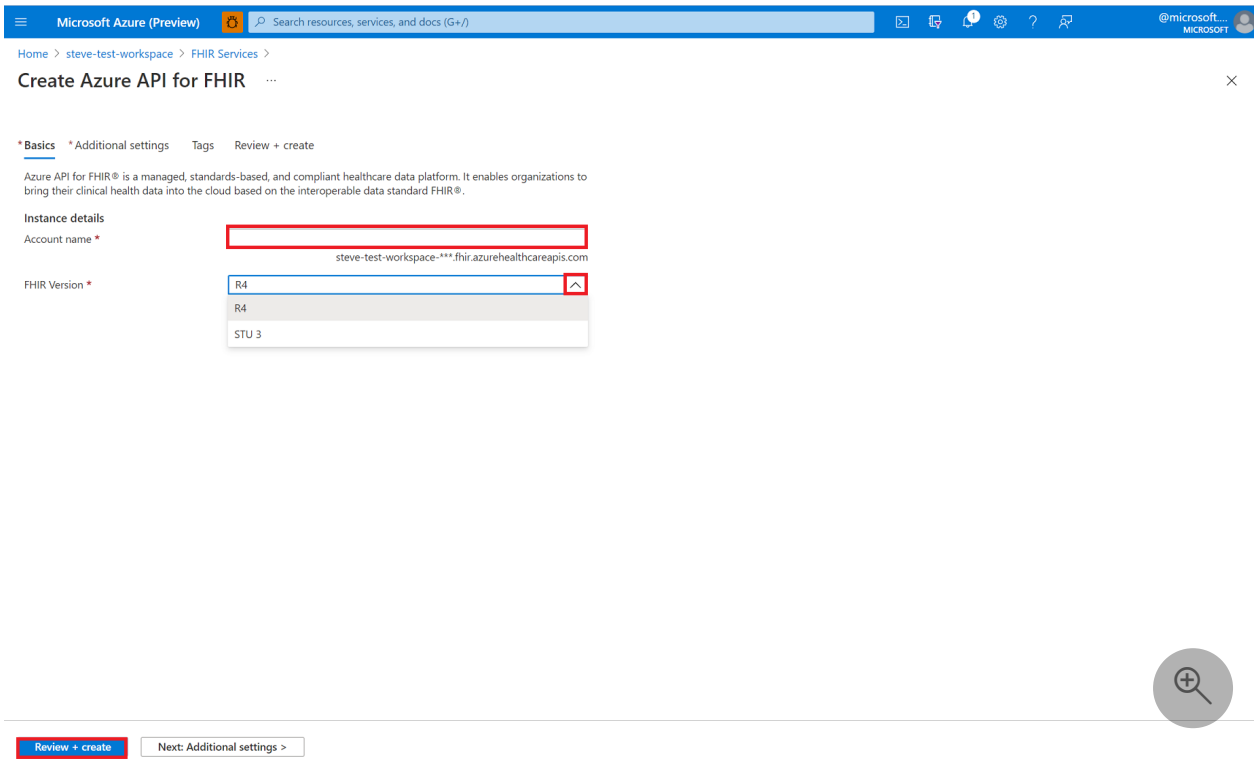
From the workspace, select **Deploy FHIR service**.



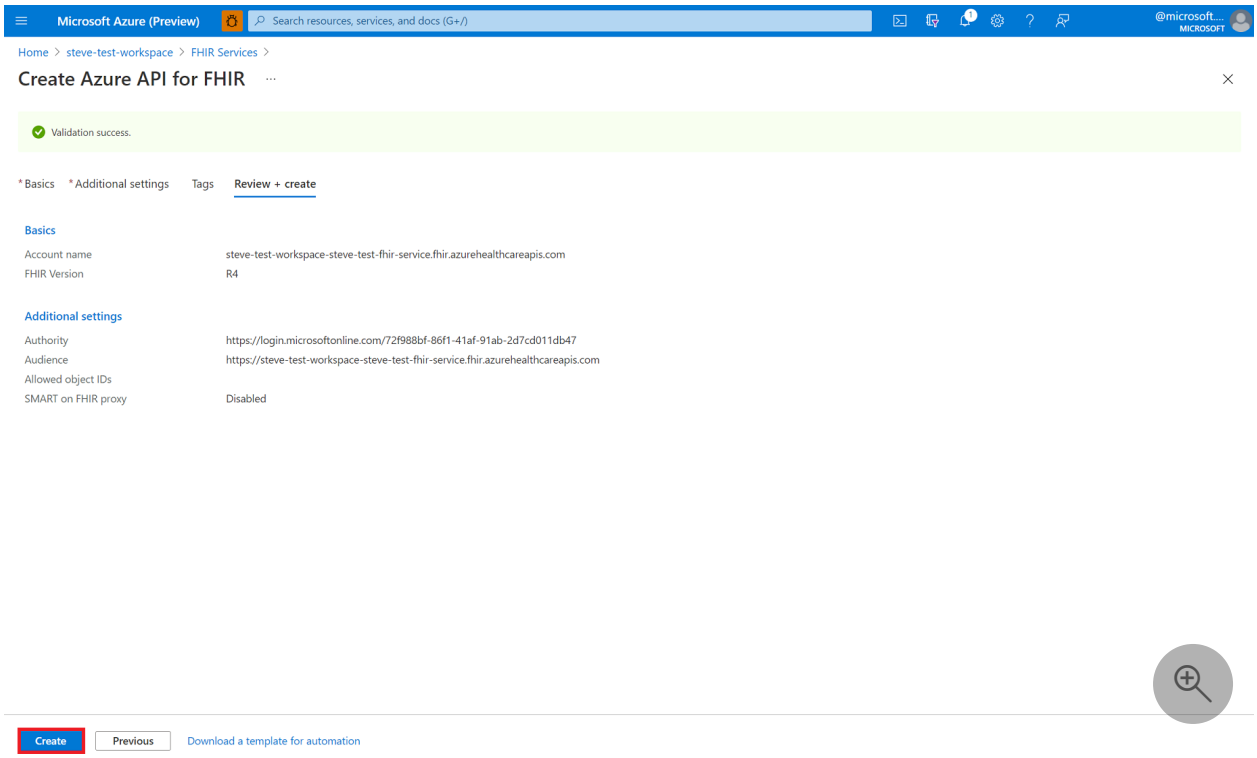
Select + **Add FHIR Service**.



Enter an **Account name** for your FHIR service. Select the **FHIR version** (**STU3** or **R4**), and then select **Review + create**.

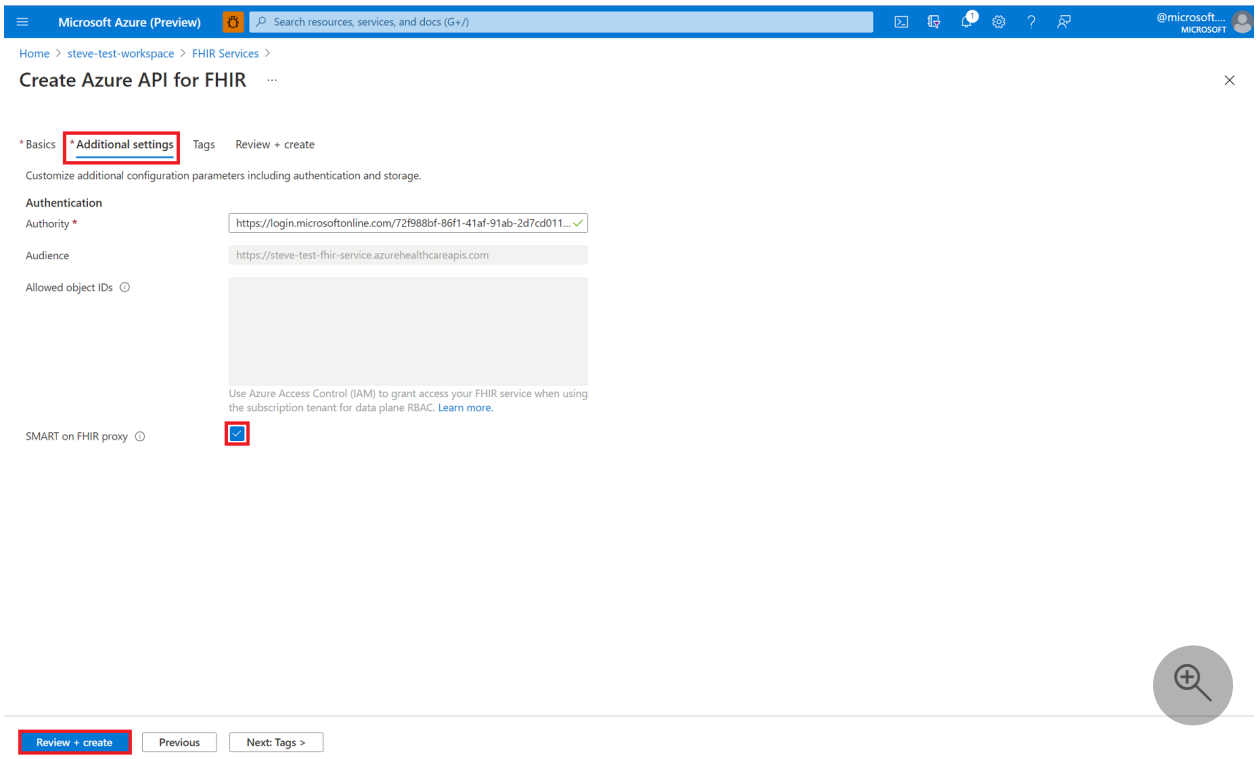


Before you select **Create**, review the properties of the **Basics** and **Additional settings** of your FHIR service. If you need to go, back and make changes, select **Previous**. Confirm that the **Validation success** message is displayed.



Additional settings (optional)

You can also select the **Additional settings** tab to view the authentication settings. The default configuration for Azure API for FHIR is to **use Azure RBAC for assigning data plane roles**. When it's configured in this mode, the "Authority" for FHIR service will be set to the Azure Active Directory tenant of the subscription.



Notice that the box for entering **Allowed object IDs** is grayed out. This is because 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.

Fetch FHIR API capability statement

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

Next steps

In this article, you learned how to deploy FHIR service within Azure Health Data Services using the Azure portal. For more information about accessing FHIR service using Postman, see

Access FHIR service using Postman

Deploy a FHIR service within Azure Health Data Services using Bicep

Article • 04/13/2023

In this article, you'll learn how to deploy FHIR service within the Azure Health Data Services using Bicep.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

PowerShell

- An Azure account with an active subscription. Create one for free.
- If you want to run the code locally:
  - Azure PowerShell.

Review the Bicep file

The Bicep file used in this article is from Azure Quickstart Templates.

Bicep

```
@description('The name of the service.')
param serviceName string

@description('Location of Azure API for FHIR')
@allowed([
  'australiaeast'
  'eastus'
  'eastus2'
  'japaneast'
  'northcentralus'
  'northeurope'
  'southcentralus'
  'southeastasia'
  'uksouth'
  'ukwest'
  'westcentralus'
```

```
  'westeurope'
  'westus2'
])
param location string

resource service 'Microsoft.HealthcareApis/services@2021-11-01' = {
  name: serviceName
  location: location
  kind: 'fhir-R4'
  properties: {
    authenticationConfiguration: {
      audience: 'https://${serviceName}.azurehealthcareapis.com'
      authority: uri(environment().authentication.loginEndpoint,
subscription().tenantId)
    }
  }
}
```

The Bicep file defines three Azure resources:

- Microsoft.HealthcareApis/workspaces: create a Microsoft.HealthcareApis/workspaces resource.
- Microsoft.HealthcareApis/workspaces/fhirservices: create a Microsoft.HealthcareApis/workspaces/fhirservices resource.
- Microsoft.Storage/storageAccounts: create a Microsoft.Storage/storageAccounts resource.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

   PowerShell

   Azure PowerShell

   ```
   New-AzResourceGroup -Name exampleRG -Location eastus
   New-AzResourceGroupDeployment -ResourceGroupName exampleRG -
   TemplateFile ./main.bicep -serviceName "<service-name>" -location "
   <location>"
   ```

   Replace <**service-name**> with the name of the service. Replace <**location**> with the location of the Azure API for FHIR. Location options include:

- australiaeast
- eastus
- eastus2
- japaneast
- northcentralus
- northeurope
- southcentralus
- southeastasia
- uksouth
- ukwest
- westcentralus
- westeurope
- westus2

> **Note**
>
> When the deployment finishes, you should see a message indicating the deployment succeeded.

Review the deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

PowerShell

Azure PowerShell

```
Get-AzResource -ResourceGroupName exampleRG
```

> **Note**
>
> You can also verify that the FHIR service is up and running by opening a browser and navigating to `https://<yourfhirservice>.azurehealthcareapis.com/metadata`. If the capability statement is automatically displayed or downloaded, your deployment was successful. Make sure to replace **<yourfhirservice>** with the **<service-name>** you used in the deployment step of this quickstart.

Clean up the resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

PowerShell

Azure PowerShell

```
Remove-AzResourceGroup -Name exampleRG
```

Next steps

In this quickstart guide, you've deployed the FHIR service within Azure Health Data Services using Bicep. For more information about FHIR service supported features, proceed to the following article:

Supported FHIR Features

Deploy a FHIR service within Azure Health Data Services - using ARM template

Article • 06/06/2023

In this article, you'll learn how to deploy FHIR service within the Azure Health Data Services (hereby called FHIR service) using the Azure Resource Manager template (ARM template). We provide you two options using PowerShell or using CLI.

An ARM template is a 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.

Prerequisites

PowerShell

- An Azure account with an active subscription. Create one for free.
- If you want to run the code locally:
  - Azure PowerShell.

Review the ARM template

The template used in this article is from Azure Quickstart Templates.

The template defines three Azure resources:

- Microsoft.HealthcareApis/workspaces
- Microsoft.HealthcareApis/workspaces/fhirservices
- Microsoft.Storage/storageAccounts

> **Note**
>
> Local RBAC will be deprecated on September 9th, 2023. Access Policies configuration associated with Local RBAC in ARM template will be deprecated. For questions, please **contact us**.

You can deploy the FHIR service resource by **removing** the workspaces resource, the storage resource, and the `dependsOn` property in the "Microsoft.HealthcareApis/workspaces/fhirservices" resource.

JSON

```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "authorityurl": {
            "type": "string",
            "defaultValue": "https://login.microsoftonline.com"
        },
        "tagName": {
            "type": "string",
            "defaultValue": "My Deployment"
        },
        "region": {
            "type": "string",
                  "allowedValues": [
                "australiaeast",
                "canadacentral",
                "eastus",
                "eastus2",
                "germanywestcentral",
                "japaneast",
                "northcentralus",
                "northeurope",
                "southafricanorth",
                "southcentralus",
                "southeastasia",
                "switzerlandnorth",
                "uksouth",
                "ukwest",
                "westcentralus",
                "westeurope",
                "westus2"
            ]
        },
        "workspaceName": {
            "type": "string"
        },
        "fhirServiceName": {
            "type": "string"
        },
        "tenantid": {
            "type": "string"
        },
        "storageAccountName": {
            "type": "string"
```

```
        },
        "storageAccountConfirm": {
            "type": "bool",
            "defaultValue": true
        },
        "AccessPolicies": {
            "type": "array",
            "defaultValue": []
        },
        "smartProxyEnabled": {
            "type": "bool",
            "defaultValue": false
        }
    },
    "variables": {
        "authority": "[Concat(parameters('authorityurl'), '/',
parameters('tenantid'))]",
        "createManagedIdentity": true,
        "managedIdentityType": {
            "type": "SystemAssigned"
        },
        "storageBlobDataContributerRoleId": "[concat('/subscriptions/',
subscription().subscriptionId,
'/providers/Microsoft.Authorization/roleDefinitions/', 'ba92f5b4-2d11-453d-
a403-e96b0029c9fe')]"
    },
    "resources": [
        {
            "type": "Microsoft.HealthcareApis/workspaces",
            "name": "[parameters('workspaceName')]",
            "apiVersion": "2020-11-01-preview",
            "location": "[parameters('region')]",
            "properties": {}
        },
       {
            "type": "Microsoft.HealthcareApis/workspaces/fhirservices",
            "kind": "fhir-R4",
            "name": "[concat(parameters('workspaceName'), '/',
parameters('fhirServiceName'))]",
            "apiVersion": "2020-11-01-preview",
            "location": "[parameters('region')]",
            "dependsOn": [
                "[resourceId('Microsoft.HealthcareApis/workspaces',
parameters('workspaceName'))]"
            ],
            "tags": {
                "environmentName": "[parameters('tagName')]"
            },
            "properties": {
                "accessPolicies": "[parameters('AccessPolicies')]",
                "authenticationConfiguration": {
                    "authority": "[variables('Authority')]",
                    "audience": "[concat('https//',
parameters('workspaceName'), '-', parameters('fhirServiceName'),
'.fhir.azurehealthcareapis.com')]",
```

```
                    "smartProxyEnabled": "[parameters('smartProxyEnabled')]"
                },
                "corsConfiguration": {
                    "allowCredentials": false,
                    "headers": ["*"],
                    "maxAge": 1440,
                    "methods": ["DELETE", "GET", "OPTIONS", "PATCH", "POST",
"PUT"],
                    "origins": ["https://localhost:6001"]
                },
                "exportConfiguration": {
                    "storageAccountName": "
[parameters('storageAccountName')]"
                }
            },
            "identity": "[if(variables('createManagedIdentity'),
variables('managedIdentityType'), json('null'))]"
        },
        {
            "name": "[parameters('storageAccountName')]",
            "type": "Microsoft.Storage/storageAccounts",
            "apiVersion": "2019-06-01",
            "location": "[resourceGroup().location]",
            "properties": {
                "supportsHttpsTrafficOnly": "true"
            },
            "condition": "[parameters('storageAccountConfirm')]",
            "dependsOn": [
                "[parameters('fhirServiceName')]"
            ],
            "sku": {
                "name": "Standard_LRS"
            },
            "kind": "Storage",
            "tags": {
                "environmentName": "[parameters('tagName')]",
                "test-account-rg": "true"
            }
        }
    ],
    "outputs": {
    }
}
```

Deploy ARM template

You can deploy the ARM template using two options: PowerShell or CLI.

The sample code provided below uses the template in the “templates” subfolder of the subfolder “src”. You may want to change the location path to reference the template file properly.

The deployment process takes a few minutes to complete. Take a note of the names for the FHIR service and the resource group, which you'll use later.

PowerShell

Deploy the template: using PowerShell

Run the code in PowerShell locally, in Visual Studio Code, or in Azure Cloud Shell, to deploy the FHIR service.

If you haven't logged in to Azure, use "Connect-AzAccount" to log in. Once you've logged in, use "Get-AzContext" to verify the subscription and tenant you want to use. You can change the subscription and tenant if needed.

You can create a new resource group, or use an existing one by skipping the step or commenting out the line starting with “New-AzResourceGroup”.

PowerShell

```
### variables
$resourcegroupname="your resource group"
$location="South Central US"
$workspacename="your workspace name"
$servicename="your fhir service name"
$tenantid="xxx"
$subscriptionid="xxx"
$storageaccountname="storage account name"
$storageaccountconfirm=1

### login to azure
Connect-AzAccount
#Connect-AzAccount SubscriptionId $subscriptionid
Set-AzContext -Subscription $subscriptionid
Connect-AzAccount -Tenant $tenantid -SubscriptionId $subscriptionid
#Get-AzContext

### create resource group
New-AzResourceGroup -Name $resourcegroupname -Location $location

### deploy the resource
New-AzResourceGroupDeployment -ResourceGroupName $resourcegroupname -
TemplateFile "src/templates/fhirtemplate.json" -region $location -
workspaceName $workspacename -fhirServiceName $fhirservicename -tenantid
$tenantid -storageAccountName $storageaccountname -storageAccountConfirm
$storageaccountconfirm
```

Review the deployed resources

You can verify that the FHIR service is up and running by opening the browser and navigating to `https://<yourfhir servic>.azurehealthcareapis.com/metadata`. If the capability statement is displayed or downloaded automatically, your deployment is successful.

Clean up the resources

When the resource is no longer needed, run the code below to delete the resource group.

PowerShell

PowerShell

```
$resourceGroupName = “your resource group name”
Remove-AzResourceGroup -Name $resourceGroupName
```

Next steps

In this quickstart guide, you've deployed the FHIR service within Azure Health Data Services using an ARM template. For more information about FHIR service supported features, see.

Supported FHIR Features

SMART on FHIR

Article • 08/10/2023

Substitutable Medical Applications and Reusable Technologies(SMART on FHIR) is a healthcare standard through which applications can access clinical information through a data store. It adds a security layer based on open standards including OAuth2 and OpenID Connect, to FHIR interfaces to enable integration with EHR systems. Using SMART on FHIR provides at least three important benefits:

- Applications have a known method for obtaining authentication/authorization to a FHIR repository.
- Users accessing a FHIR repository with SMART on FHIR are restricted to resources associated with the user, rather than having access to all data in the repository.
- Users have the ability to grant applications access to a limited set of their data by using SMART clinical scopes.

Below tutorials provide steps to enable SMART on FHIR applications with FHIR Service.

Prerequisites

- An instance of the FHIR Service
- .NET SDK 6.0
- Enable cross-origin resource sharing (CORS)
- Register public client application in Azure AD
  - After registering the application, make note of the applicationId for client application.
- Ensure you have access to Azure Subscription of FHIR service, to create resources and add role assignments.

SMART on FHIR Enhanced using Azure Health Data Services Samples

Step 1 : Set up FHIR SMART user role

Follow the steps listed under section Manage Users: Assign Users to Role. Any user added to this role will be able to access the FHIR Service if their requests comply with the SMART on FHIR implementation Guide, such as request having access token which includes a fhirUser claim and a clinical scopes claim. The access granted to the users in

this role will then be limited by the resources associated to their fhirUser compartment and the restrictions in the clinical scopes.

Step 2 : FHIR server integration with samples

For integration with Azure Health Data Services samples, you would need to follow the steps in samples open source solution.

Click on the link to navigate to Azure Health Data Service Samples OSS. This steps listed in the document will enable integration of FHIR server with other Azure Services (such as APIM, Azure functions and more).

> ⓘ Note
>
> Samples are open-source code, and you should review the information and licensing terms on GitHub before using it. They are not part of the Azure Health Data Service and are not supported by Microsoft Support. These samples can be used to demonstrate how Azure Health Data Services and other open-source tools can be used together to demonstrate **§170.315(g)(10) Standardized API for patient and population services criterion** compliance, using Azure Active Directory as the identity provider workflow.

SMART on FHIR Proxy

▼ Click to expand!

> ⓘ Note
>
> This is another option to using "SMART on FHIR Enhanced using AHDS Samples" mentioned above. We suggest you to adopt SMART on FHIR enhanced. SMART on FHIR Proxy option is legacy option. SMART on FHIR enhanced version provides added capabilities than SMART on FHIR proxy. SMART on FHIR enhanced capability can be considered to meet requirements with **SMART on FHIR Implementation Guide (v 1.0.0)** and **§170.315(g)(10) Standardized API for patient and population services criterion**.

Step 1 : Set admin consent for your client application

To use SMART on FHIR, you must first authenticate and authorize the app. The first time you use SMART on FHIR, you also must get administrative consent to let the app access your FHIR resources.

If you don't have an ownership role in the app, contact the app owner and ask them to grant admin consent for you in the app.

If you do have administrative privileges, complete the following steps to grant admin consent to yourself directly. (You also can grant admin consent to yourself later when you're prompted in the app.) You can complete the same steps to add other users as owners, so they can view and edit this app registration.

To add yourself or another user as owner of an app:

1. In the Azure portal, go to Azure Active Directory.
2. In the left menu, select **App Registration**.
3. Search for the app registration you created, and then select it.
4. In the left menu, under **Manage**, select **Owners**.
5. Select **Add owners**, and then add yourself or the user you want to have admin consent.
6. Select **Save**

Step 2: Enable the SMART on FHIR proxy

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

To enable the SMART on FHIR proxy in the **Authentication** settings for your FHIR instance, select the **SMART on FHIR proxy** check box.

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:

HTTP

```
https://MYFHIRAPI.azurehealthcareapis.com/AadSmartOnFhirProxy/callback/aHR0c
HM6Ly9sb2NhbGhvc3Q6NTAwMS9zYW1wbGVhcHAvaW5kZXguaHRtbA
```

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

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

PowerShell

```
$replyUrl = "https://localhost:5001/sampleapp/index.html"
$fhirServerUrl = "https://MYFHIRAPI.fhir.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

Step 3 : Get a test patient

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

Step 4 : 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:

PowerShell

```
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`:

JSON

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

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

PowerShell

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

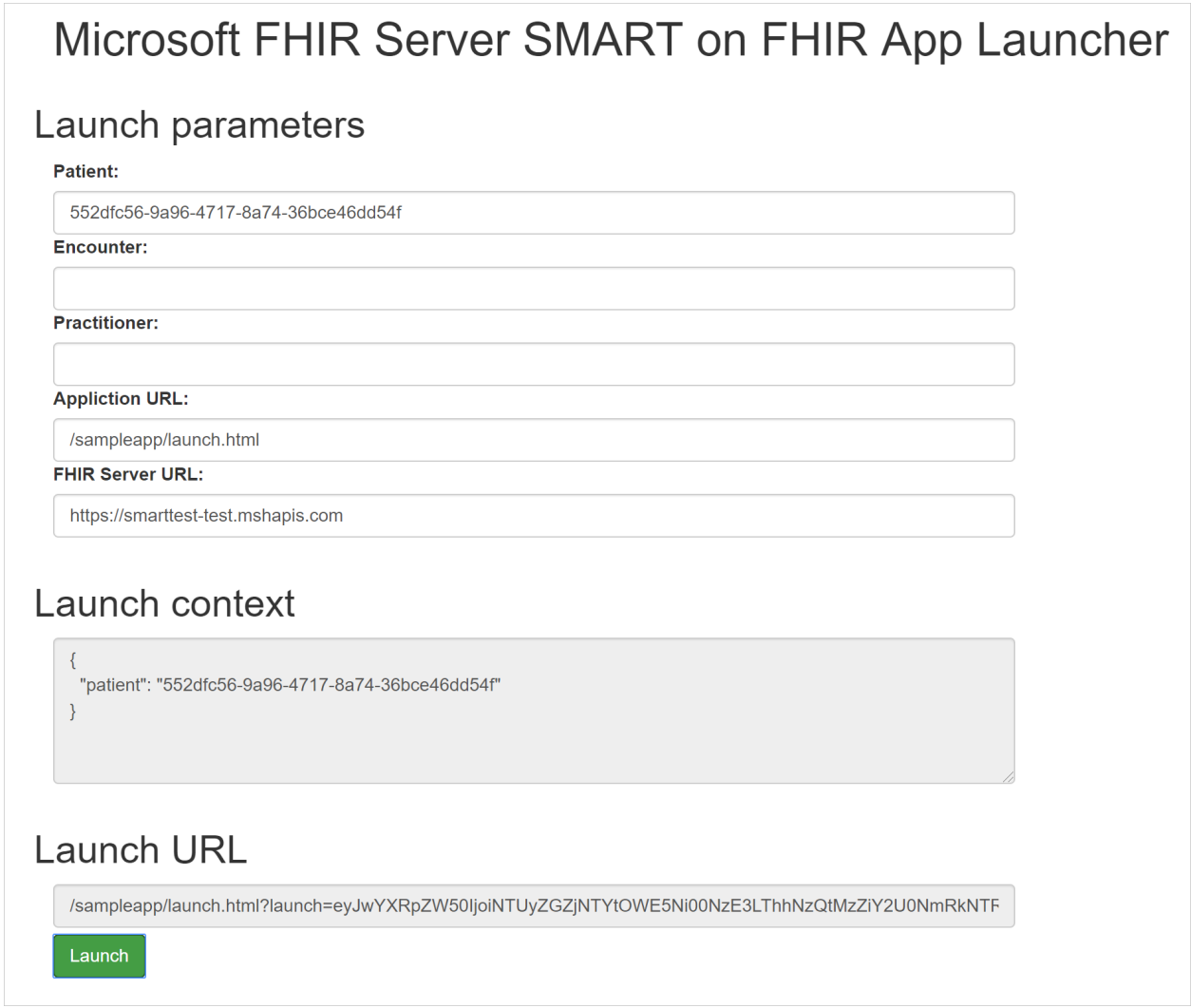
Use this command to run the application:

PowerShell

```
dotnet run
```

Step 5 : Test the SMART on FHIR proxy

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



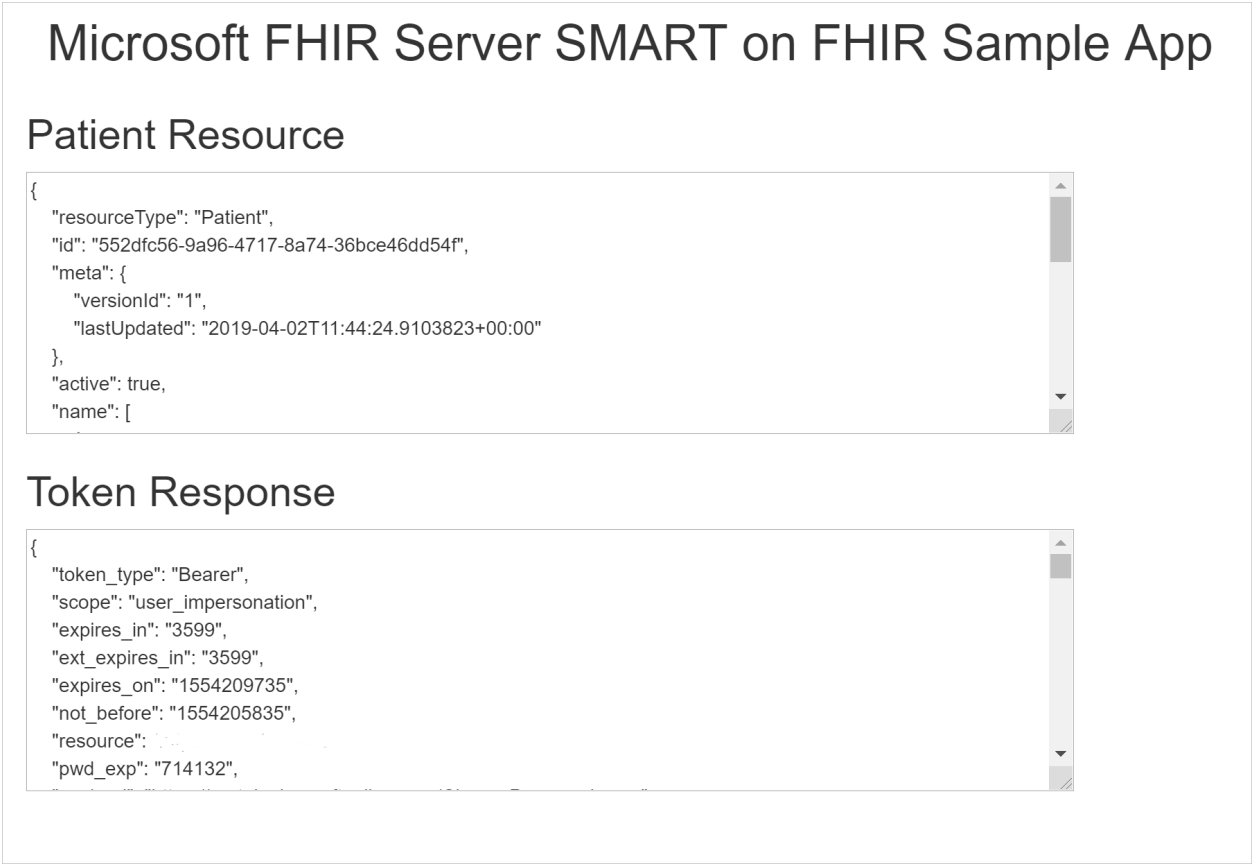
When you enter **Patient**, **Encounter**, or **Practitioner** information, you'll notice that the **Launch context** is updated. When you're using the FHIR service, the launch context is simply a JSON document that contains information about patient, practitioner, and more. This launch context is base64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

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

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

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

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



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

Next steps

Now that you've learned about enabling SMART on FHIR functionality, see the search samples page for details about how to search using search parameters, modifiers, and other FHIR search methods.

FHIR search examples

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

Article • 09/26/2022

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 FHIR service in Azure Health Data Services (hereby called FHIR service) 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 (Originally required Jan 1, 2022 - Currently Delayed)** – 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 FHIR service 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're 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 FHIR service 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 FHIR service for the CARIN IG for Blue Button.

CARIN Implementation Guide for Blue Button

CARIN Implementation Guide for Blue Button®

Article • 09/26/2022

In this tutorial, we'll walk through setting up the FHIR service in Azure Health Data Services (hereby called the FHIR service) 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 FHIR service against the C4BB IG capability statement. If you run this test against the FHIR service 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 FHIR service without any additional updates.

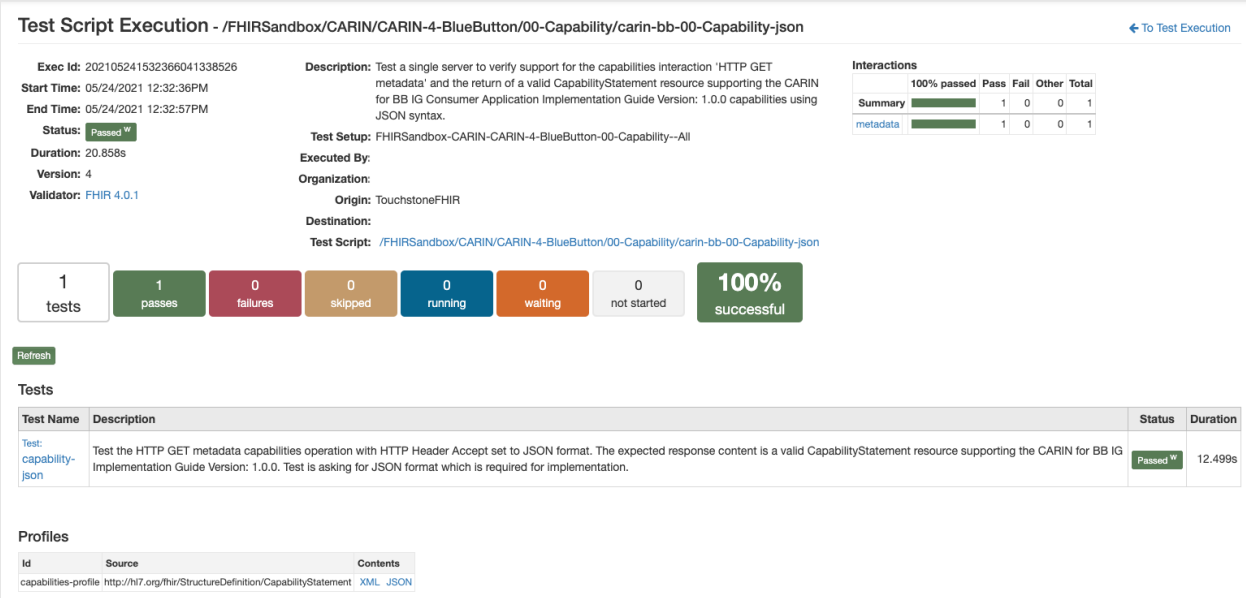
Store profiles

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

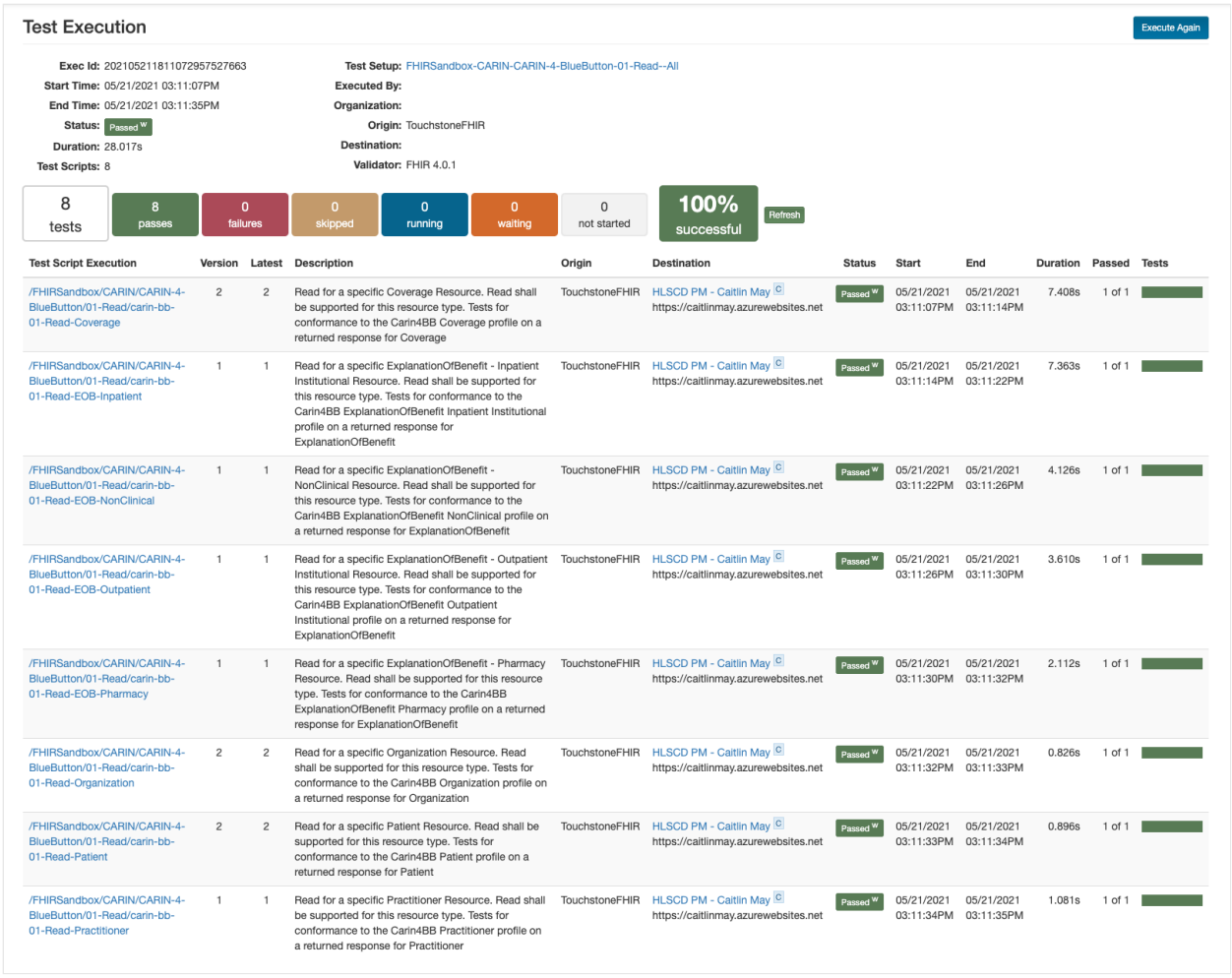
Sample rest file

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.



Touchstone read test

After testing the capabilities statement, we'll test the read capabilities of the FHIR service against the C4BB IG. This test is testing conformance against the eight profiles you loaded in the first test. You'll 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: 20210521181107295752766 3
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

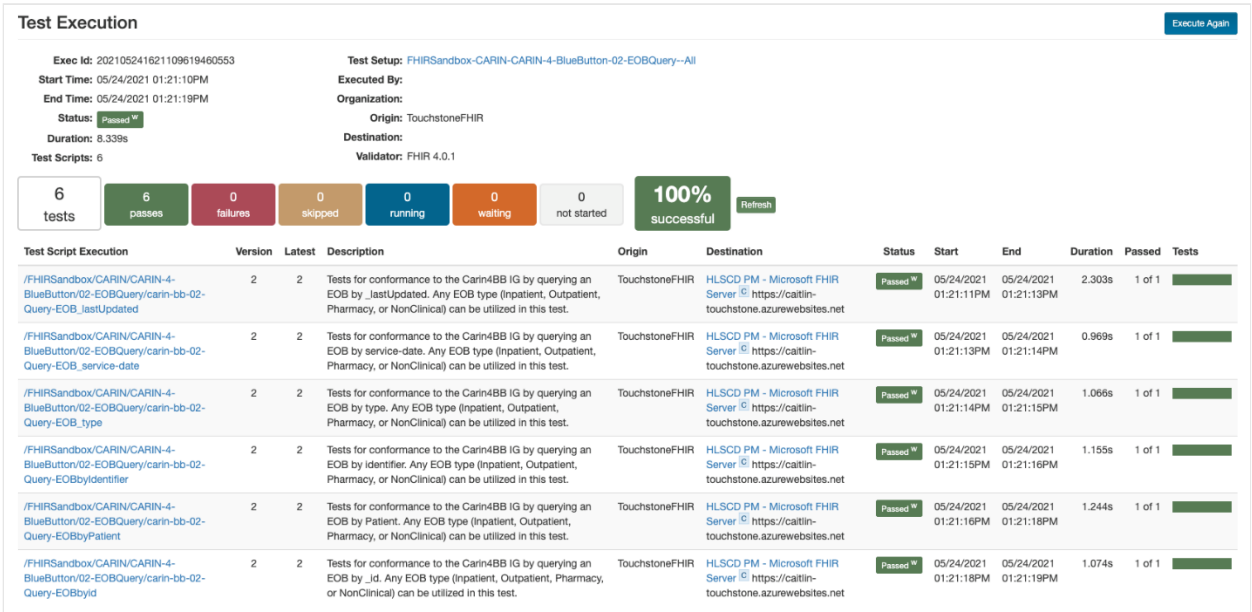
Test Setup: FHIRSandbox-CARIN-CARIN-4-BlueButton-01-Read--All
Executed By:
Organization:
Origin: TouchstoneFHIR
Destination:
Validator: FHIR 4.0.1

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	
/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	
/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	
/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	
/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	
/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	
/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	
/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	

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: 20210524162109619460553
Start Time: 05/24/2021 01:21:10PM
End Time: 05/24/2021 01:21:19PM
Status: Passed W
Duration: 8.339s
Test Scripts: 6

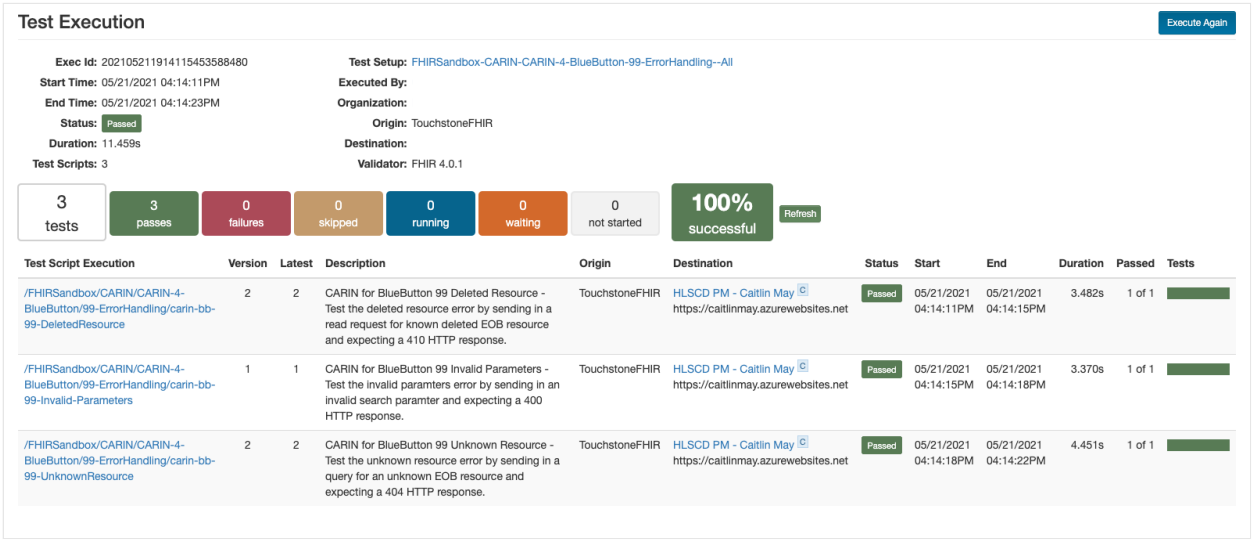
Test Setup: FHIRSandbox-CARIN-CARIN-4-BlueButton-02-EOBQuery--All
Executed By:
Organization:
Origin: TouchstoneFHIR
Destination:
Validator: FHIR 4.0.1

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 C https://caitlin-touchstone.azurewebsites.net	Passed W	05/24/2021 01:21:11PM	05/24/2021 01:21:13PM	2.303s	1 of 1	
/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 C https://caitlin-touchstone.azurewebsites.net	Passed W	05/24/2021 01:21:13PM	05/24/2021 01:21:14PM	0.969s	1 of 1	
/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 C https://caitlin-touchstone.azurewebsites.net	Passed W	05/24/2021 01:21:14PM	05/24/2021 01:21:15PM	1.066s	1 of 1	
/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 C https://caitlin-touchstone.azurewebsites.net	Passed W	05/24/2021 01:21:15PM	05/24/2021 01:21:16PM	1.155s	1 of 1	
/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 C https://caitlin-touchstone.azurewebsites.net	Passed W	05/24/2021 01:21:16PM	05/24/2021 01:21:18PM	1.244s	1 of 1	
/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 C https://caitlin-touchstone.azurewebsites.net	Passed W	05/24/2021 01:21:18PM	05/24/2021 01:21:19PM	1.074s	1 of 1	

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.



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

Tutorial for Da Vinci Drug Formulary

Article • 09/26/2022

In this tutorial, we'll walk through setting up the FHIR service in Azure Health Data Services (hereby called FHIR service) 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 FHIR service 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 FHIR service without any more updates.

Store profiles

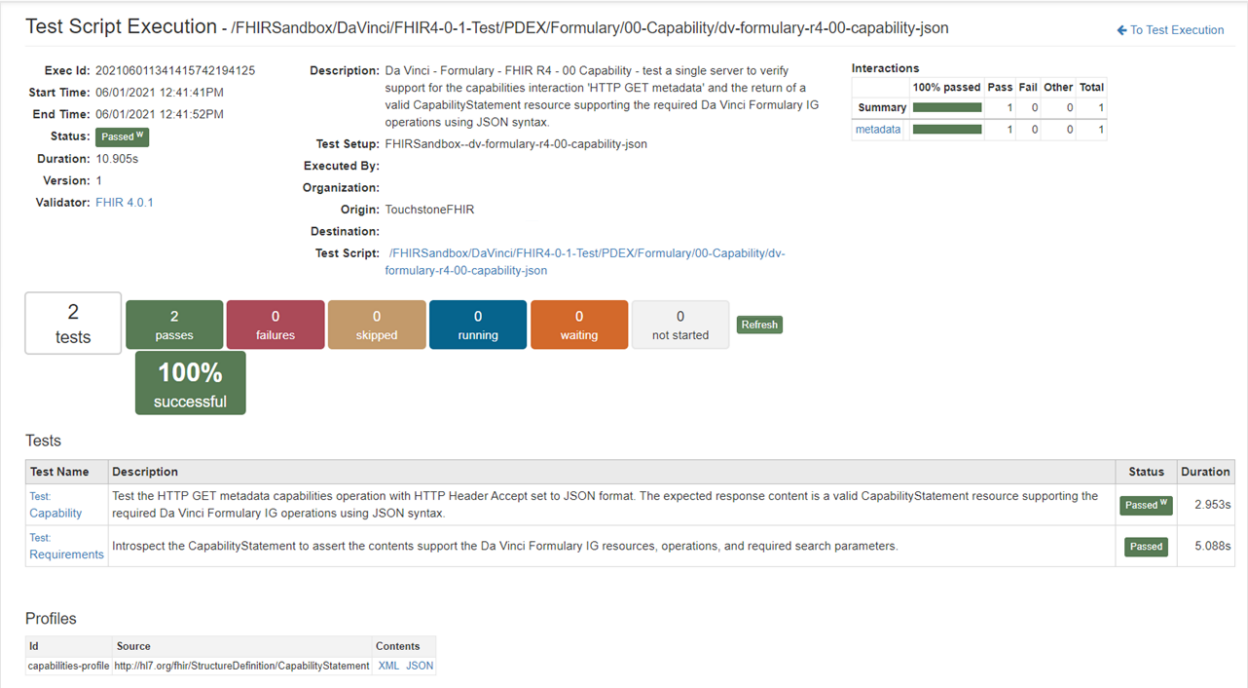
Outside of defining search parameters, the only other update you need to make to pass this test is to load the required profiles. There are two profiles used as part of the Da Vinci Drug Formulary IG.

- Formulary Drug
- Formulary Coverage Plan

Sample rest file

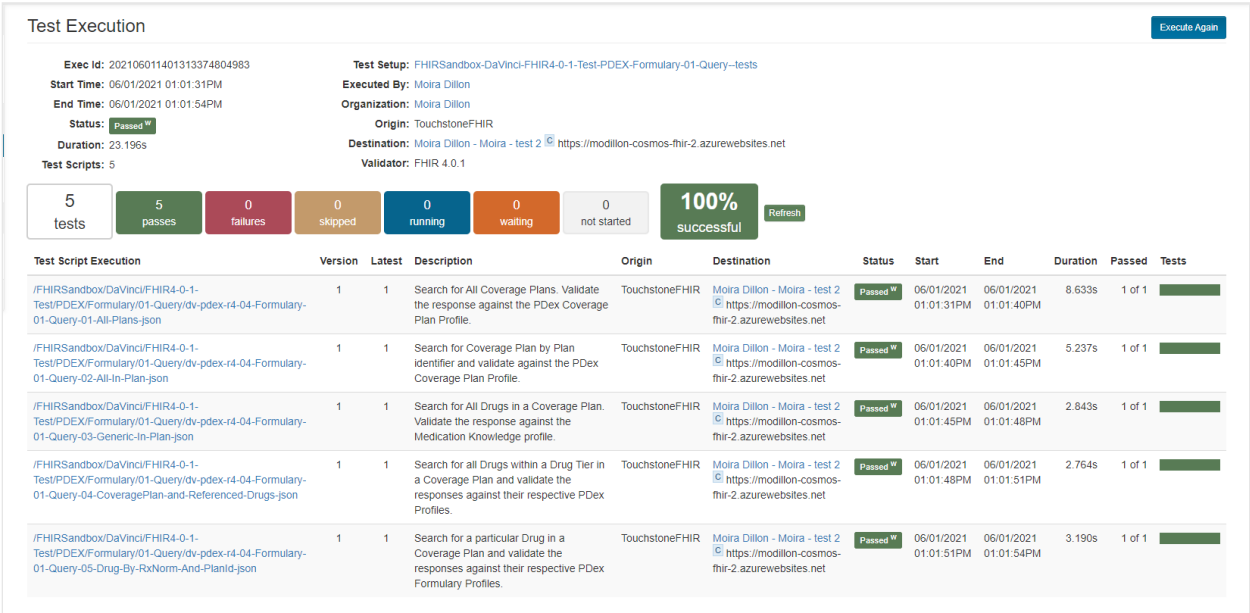
To assist with creation of these search parameters and profiles, we have the Da Vinci Formulary sample HTTP file on the open-source site that includes all the steps

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:



Touchstone query test

The second test is the query capabilities. This test validates that you can search for specific Coverage Plan and Drug resources using various parameters. The best path would be to test against resources that you already have in your database, but we also have the Da VinciFormulary_Sample_Resources HTTP file available with sample resources pulled from the examples in the IG that you can use to create the resources and test against.



Next steps

In this tutorial, we walked through how to pass the Da Vinci Payer Data Exchange US Drug Formulary in Touchstone. Next, you can learn how to test the Da Vinci PDex Implementation Guide in Touchstone.

Da Vinci PDex

Da Vinci PDex

Article • 09/26/2022

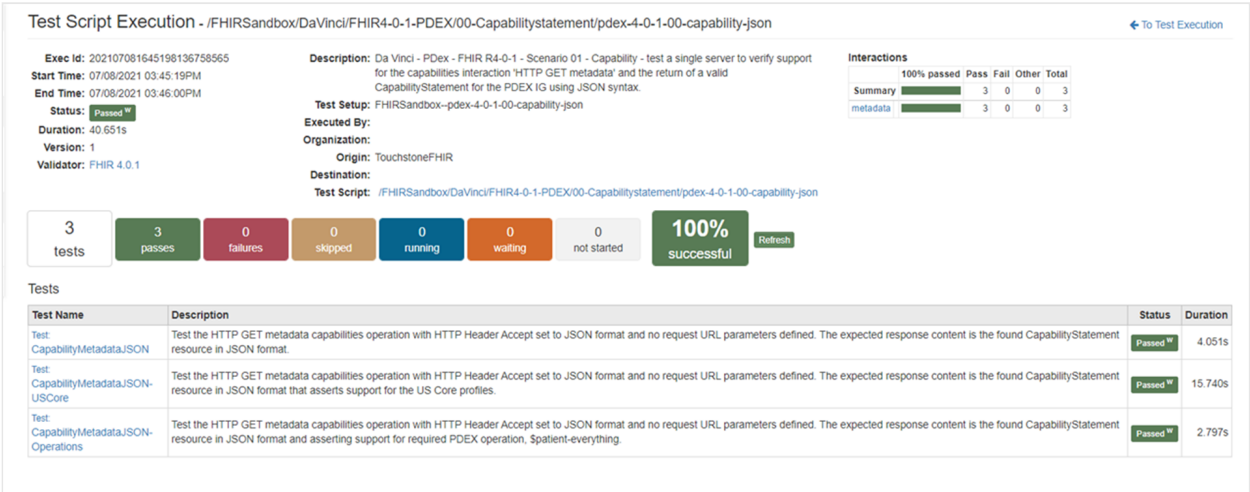
In this tutorial, we'll walk through setting up the FHIR service in Azure Health Data Services (hereby called FHIR service) to pass the Touchstone tests for the Da Vinci Payer Data Exchange Implementation Guide (PDex IG).

> **Note**
>
> FHIR service only supports JSON. The Microsoft open-source FHIR service supports both JSON and XML, and in open-source you can use the _format parameter to view the XML capability statement: `GET {fhirurl}/metadata?_format=xml`

Touchstone capability statement

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

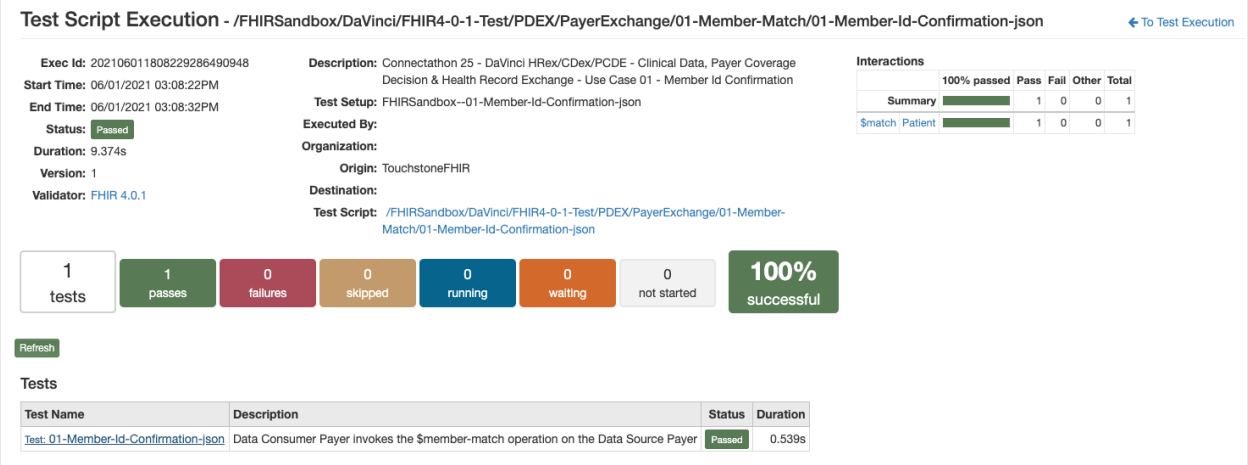
- The first test validates the basic capability statement against the IG requirements and will pass without any updates.
- The second test validates all the profiles have been added for US Core. This test will pass without updates but will include a bunch of warnings. To have these warnings removed, 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.



Touchstone $member-match test

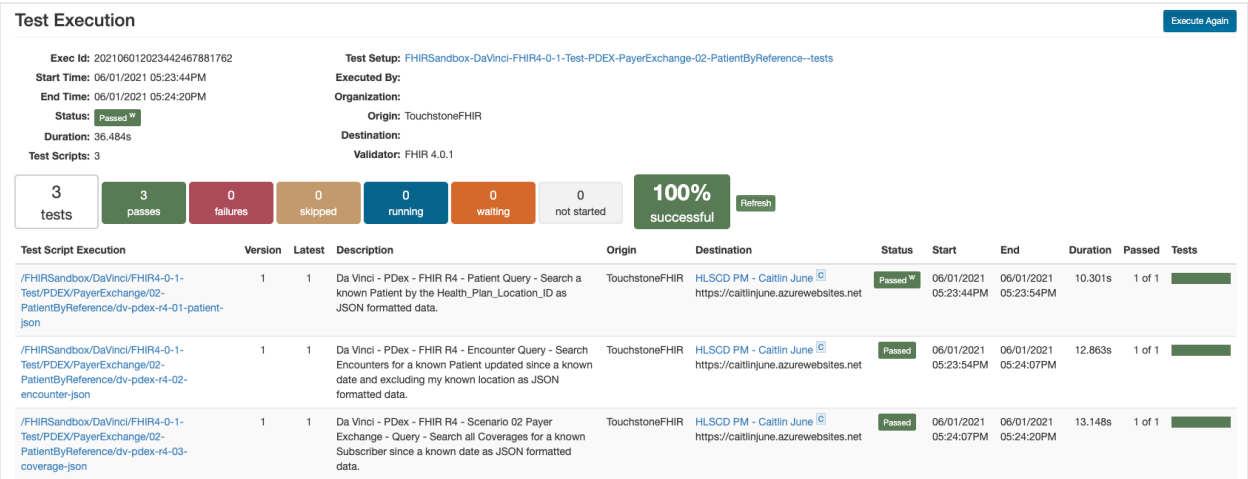
The second test in the Payer Data Exchange section tests the existence of the $member-match operation. You can read more about the $member-match operation in our $member-match operation overview.

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'll need for the test. Once this data is loaded, you'll be able to successfully pass this test. If the data isn't loaded, you'll receive a 422 response due to not finding an exact match.



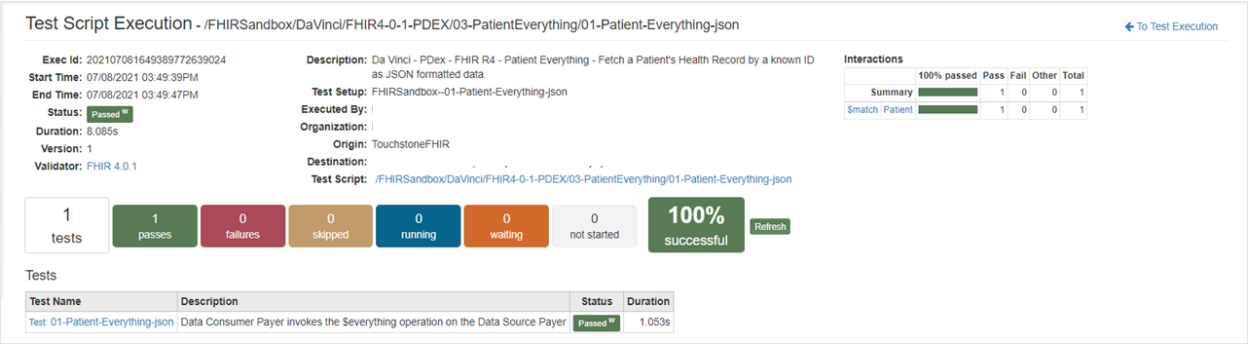
Touchstone patient by reference

The next tests we'll review is the patient by reference tests. This set of tests validates 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've uploaded a sample resource file that you can load to use as well.



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 how to test the Da Vinci PDEX Payer Network (Plan-Net) Implementation Guide.

Da Vinci Plan Net

Da Vinci Plan Net

Article • 09/26/2022

In this tutorial, we'll walk through setting up the FHIR service in Azure Health Data Services (hereby called FHIR service) to pass the Touchstone tests for the Da Vinci PDEX Payer Network (Plan-Net) Implementation Guide.

Touchstone capability statement

The first test that we'll focus on is testing the FHIR service against the Da Vinci Plan-Net 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 Plan-Net IG, you'll need to define six new search parameters for the Healthcare Service, Insurance Plan, Practitioner Role, Organization, and Organization Affiliation resources. All six of these are tested in the capability statement:

- Healthcare Service Coverage Area
- Insurance Plan Coverage Area
- Insurance Plan Plan Type
- Organization Coverage Area
- Organization Affiliation Network
- Practitioner Role Network

> **Note**
>
> In the raw JSON for these search parameters, the name is set to `Plannet_sp_<Resource Name>_<SearchParameter Name>`. The Touchstone test is expecting that the name for these will be only the `SearchParameter Name` (coverage-area, plan-type, or network).

The rest of the search parameters needed for the Da Vinci Plan-Net IG are defined by the base specification and are already available in the FHIR service without any additional updates.

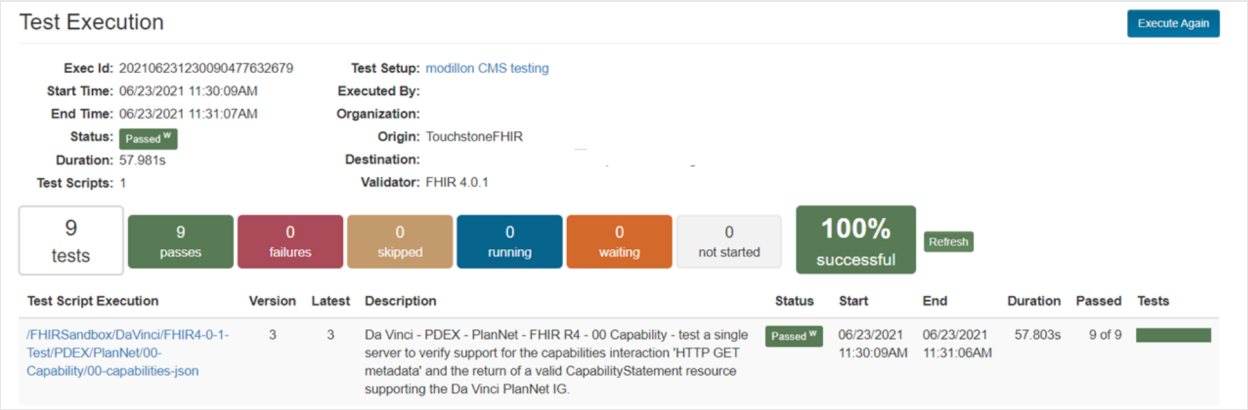
Store profiles

Outside of defining search parameters, you need to load the required profiles and extensions to pass this test. There are nine profiles used as part of the Da Vinci Plan-Net IG:

- Plan-Net Endpoint
- Plan-Net Healthcare Service
- Plan-Net InsurancePlan
- Plan-Net Location
- Plan-Net Network
- Plan-Net Organization
- Plan-Net OrganizationAffiliation
- Plan-Net Practitioner
- Plan-Net PractitionerRole

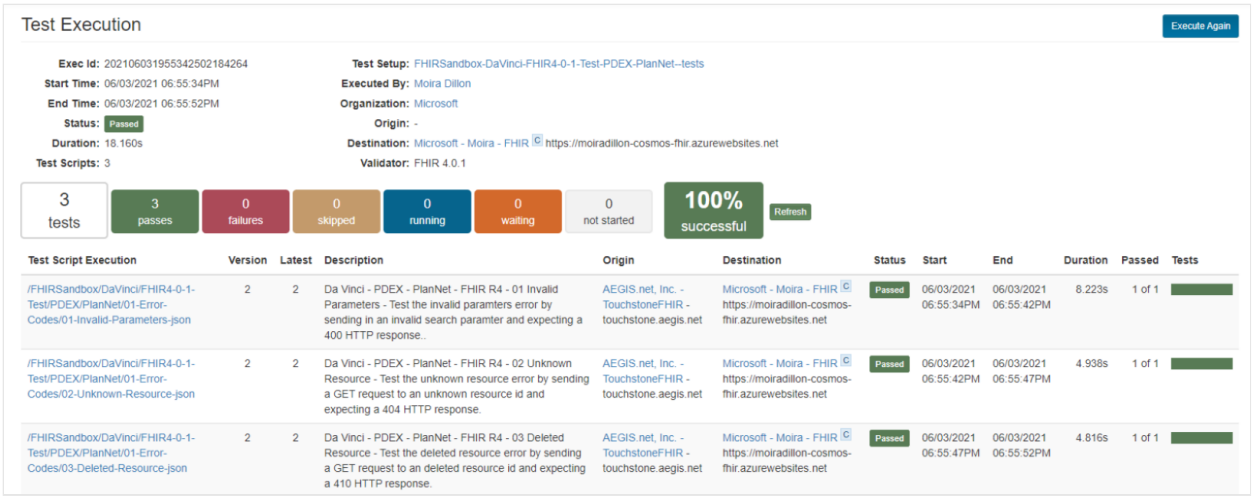
Sample rest file

To assist with creation of these search parameters and profiles, we have a 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.



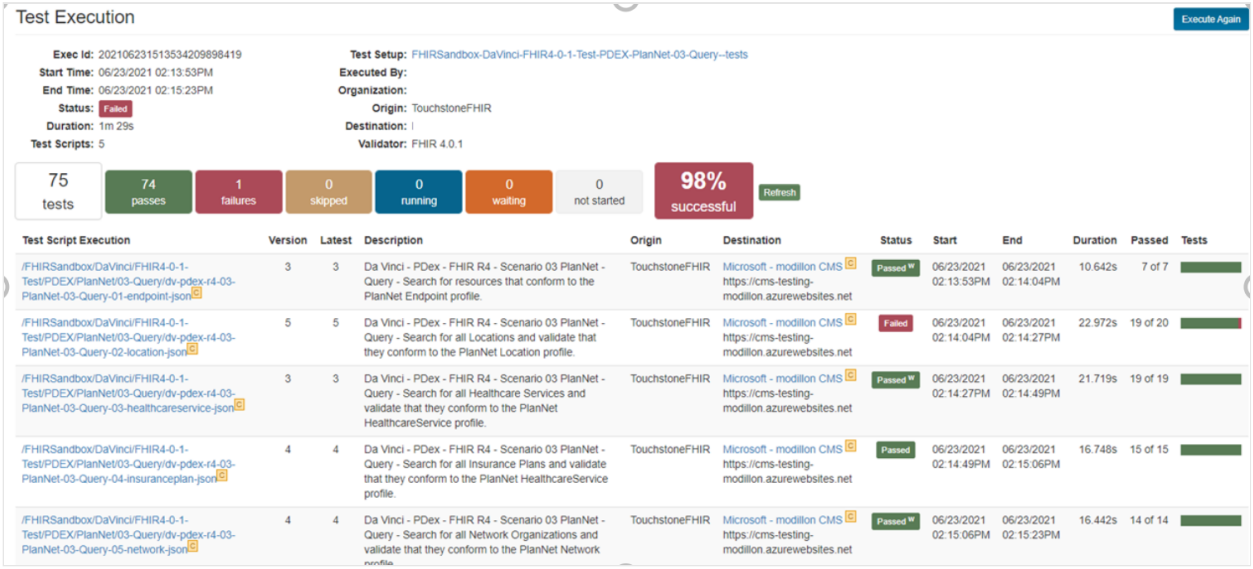
Touchstone error handling test

The second test we'll walk through is testing error handling. The only step you must do is delete a HealthcareService resource from your database and use the ID of the deleted HealthcareService resource in the test. The sample DaVinci_PlanNet.http file in the open-source site provides an example HealthcareService to post and delete for this step.



Touchstone query test

The next test we'll walk through is the query capabilities test. This test is testing conformance against the profiles you loaded in the first test. You'll 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 the DaVinci_PlanNet_Sample_Resources.http file with sample resources pulled from the examples in the IG that you can use to create the resources and test against.



> **Note**
>
> With the sample resources provided, you should expect a 98% success rate of the query tests. There's an open GitHub issue against the FHIR Server that's causing one of these tests to fail. Resource returned multiple times if it meets both base criteria and _include criteria. #2037

Next steps

In this tutorial, we walked through setting up the Azure API for FHIR to pass the Touchstone tests for the Da Vinci PDEX Payer Network (Plan-Net) Implementation Guide. Next, you can learn about all the FHIR service features.

Supported features

Access using Postman

Article • 03/27/2023

In this article, we'll walk through the steps of accessing the Azure Health Data Services (hereafter called FHIR service) with Postman.

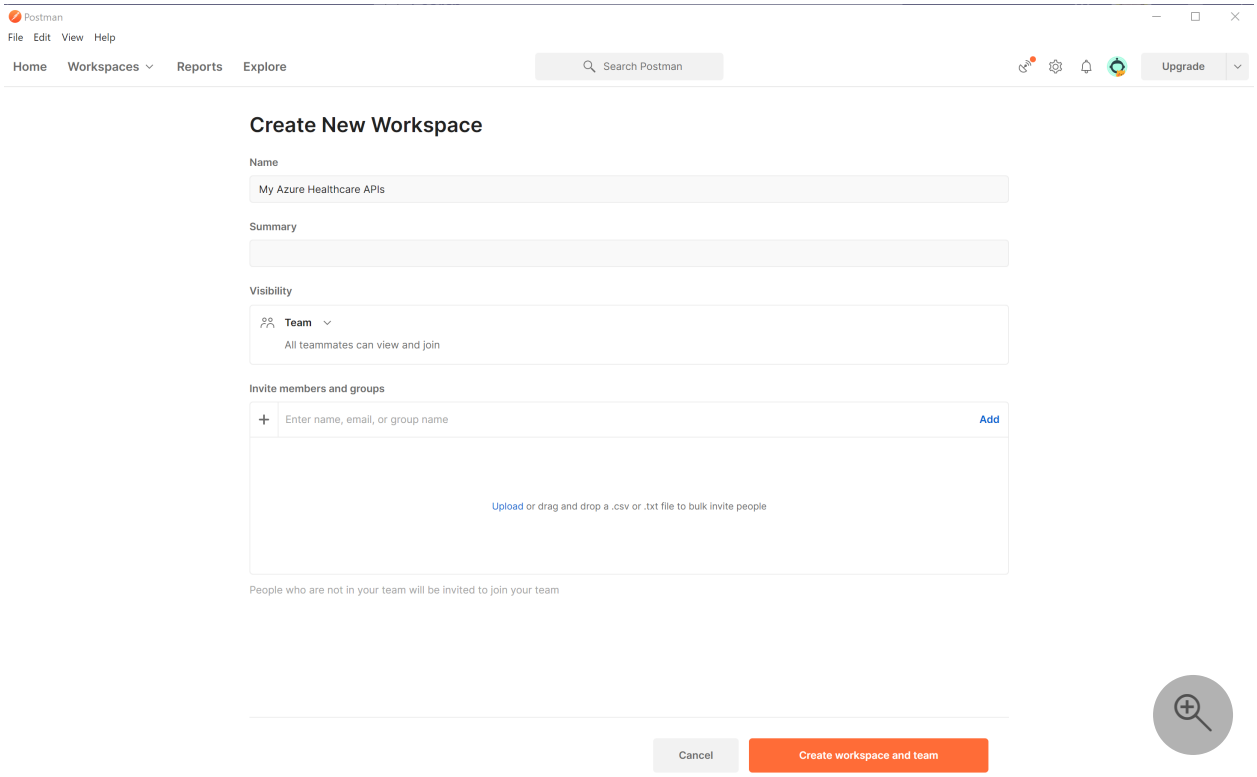
Prerequisites

- FHIR service deployed in Azure. For information about how to deploy the FHIR service, see Deploy a FHIR service.
- A registered client application to access the FHIR service. For information about how to register a client application, see Register a service client application in Azure Active Directory.
- Permissions granted to the client application and your user account, for example, "FHIR Data Contributor", to access the FHIR service.
- Postman installed locally. For more information about Postman, see Get Started with Postman.

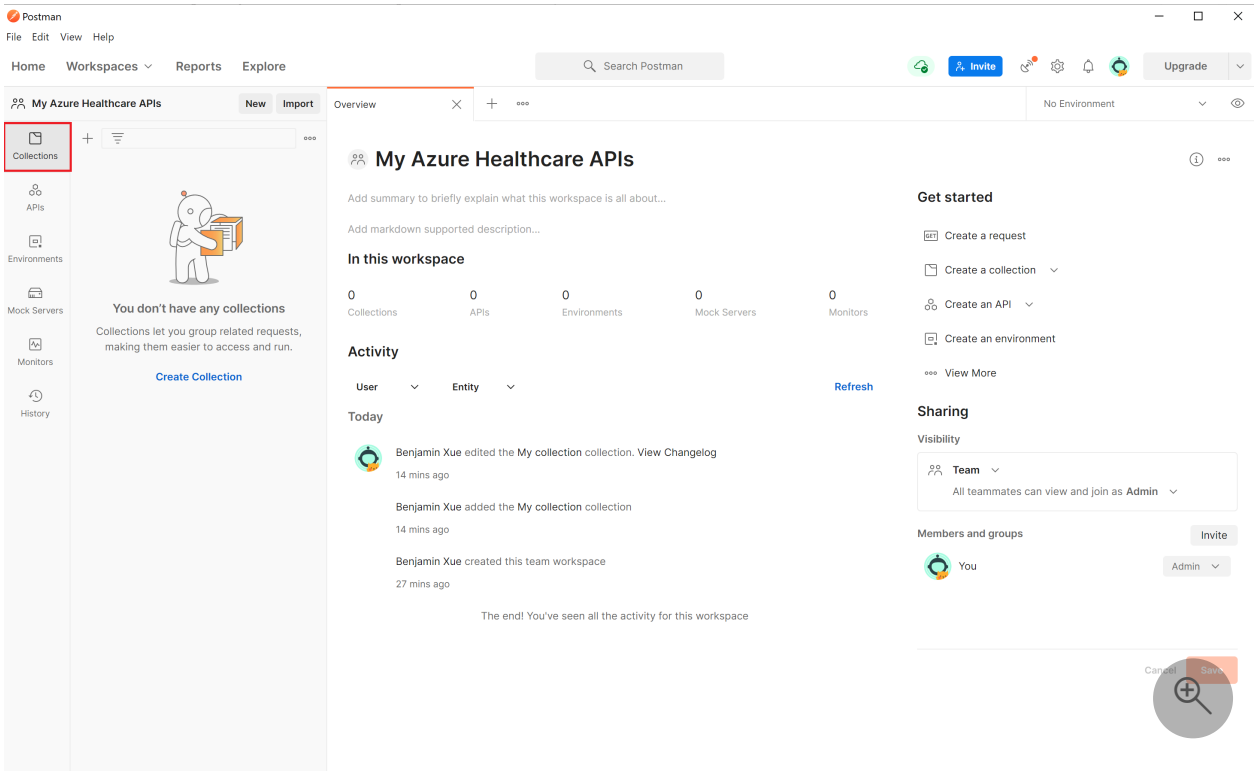
Using Postman: create workspace, collection, and environment

If you're new to Postman, follow the steps below. Otherwise, you can skip this step.

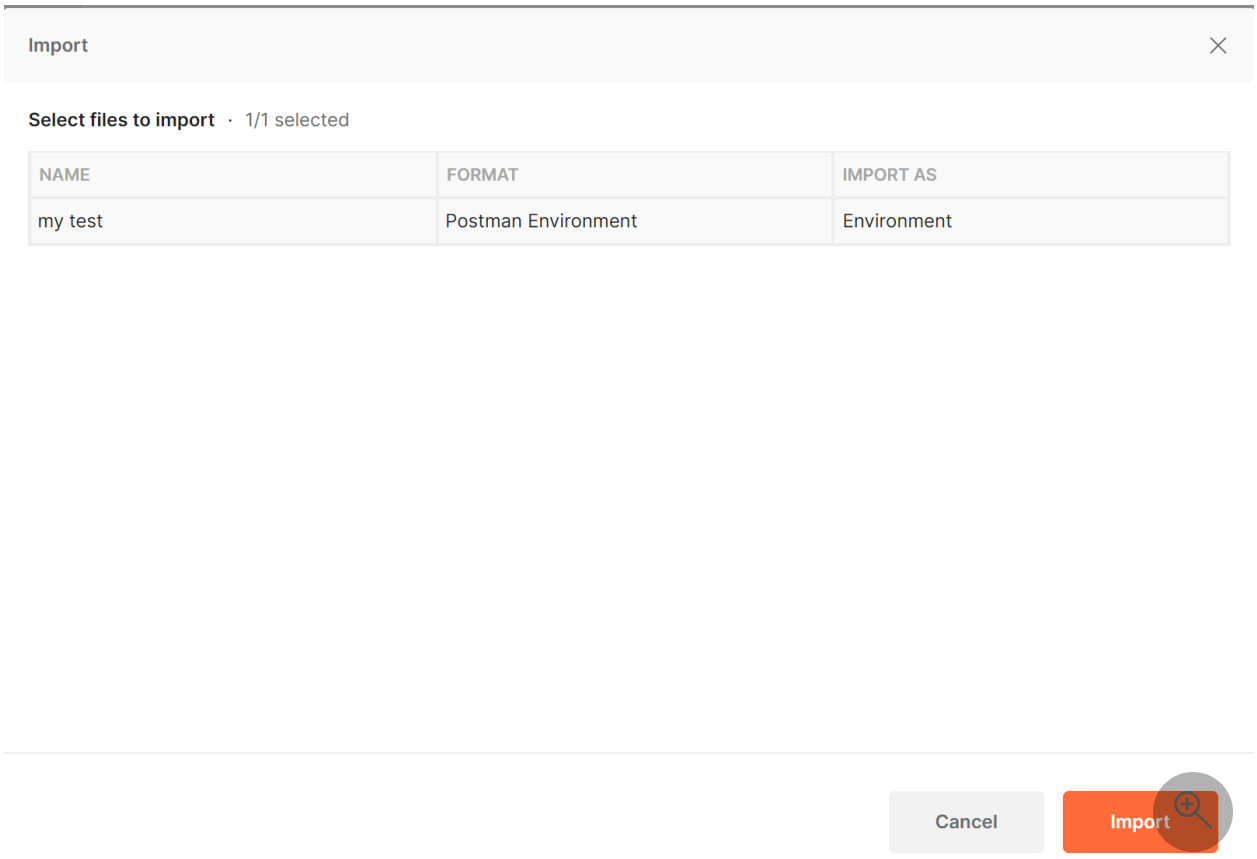
Postman introduces the workspace concept to enable you and your team to share APIs, collections, environments, and other components. You can use the default “My workspace” or “Team workspace” or create a new workspace for you or your team.



Next, create a new collection where you can group all related REST API requests. In the workspace, select **Create Collections**. You can keep the default name **New collection** or rename it. The change is saved automatically.



You can also import and export Postman collections. For more information, see the Postman documentation.



Create or update environment variables

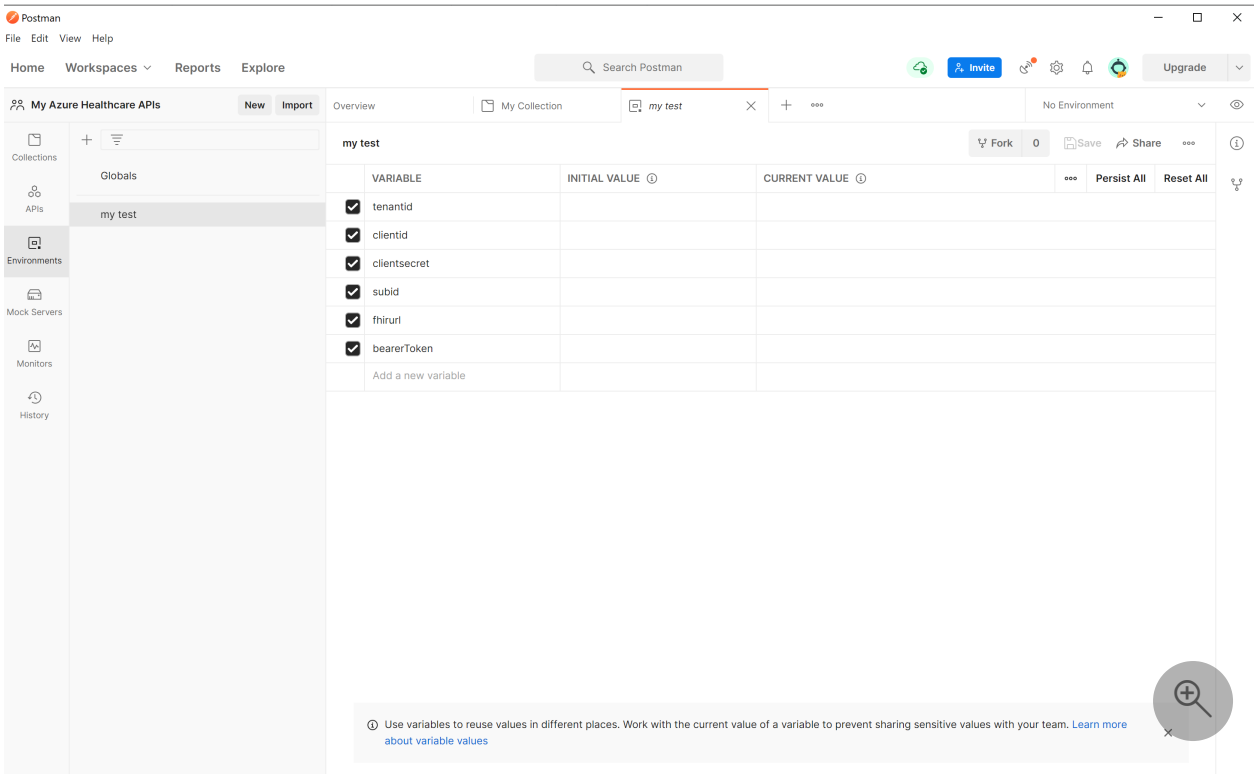
While you can use the full URL in the request, it's recommended that you store the URL and other data in variables and use them.

To access the FHIR service, we'll need to create or update the following variables.

- **tenantid** – Azure tenant where the FHIR service is deployed in. It's located from the **Application registration overview** menu option.
- **subid** – Azure subscription where the FHIR service is deployed in. It's located from the **FHIR service overview** menu option.
- **clientid** – Application client registration ID.
- **clientsecret** – Application client registration secret.
- **fhirurl** – The FHIR service full URL. For example, `https://xxx.azurehealthcareapis.com`. It's located from the **FHIR service overview** menu option.
- **bearerToken** – The variable to store the Azure Active Directory (Azure AD) access token in the script. Leave it blank.

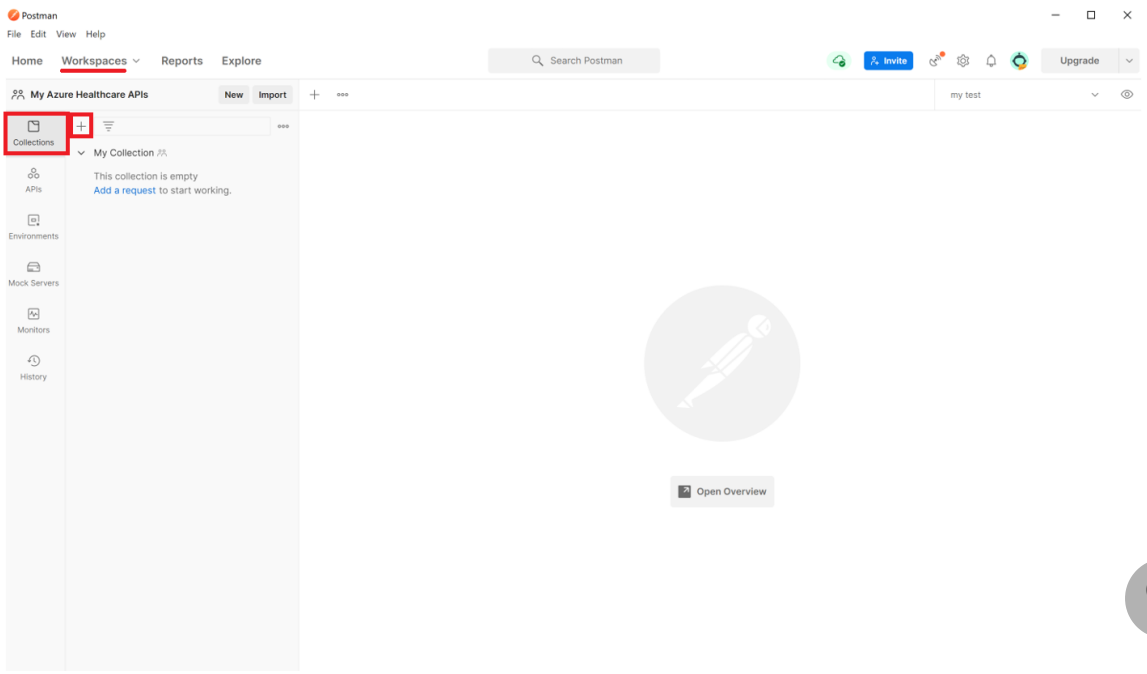
> **Note**

Ensure that you've configured the redirect URL, `https://www.getpostman.com/oauth2/callback`, in the client application registration.



Connect to the FHIR server

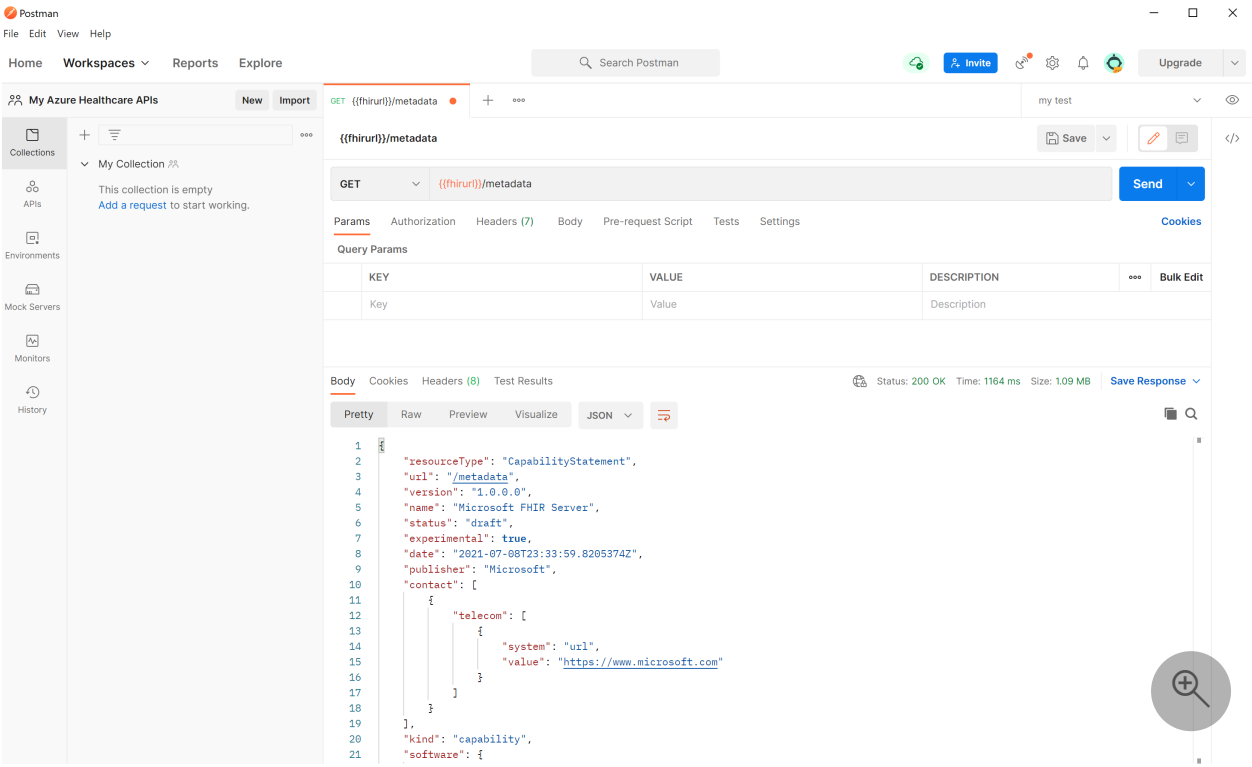
Open Postman, select the **workspace**, **collection**, and **environment** you want to use. Select the `+` icon to create a new request.

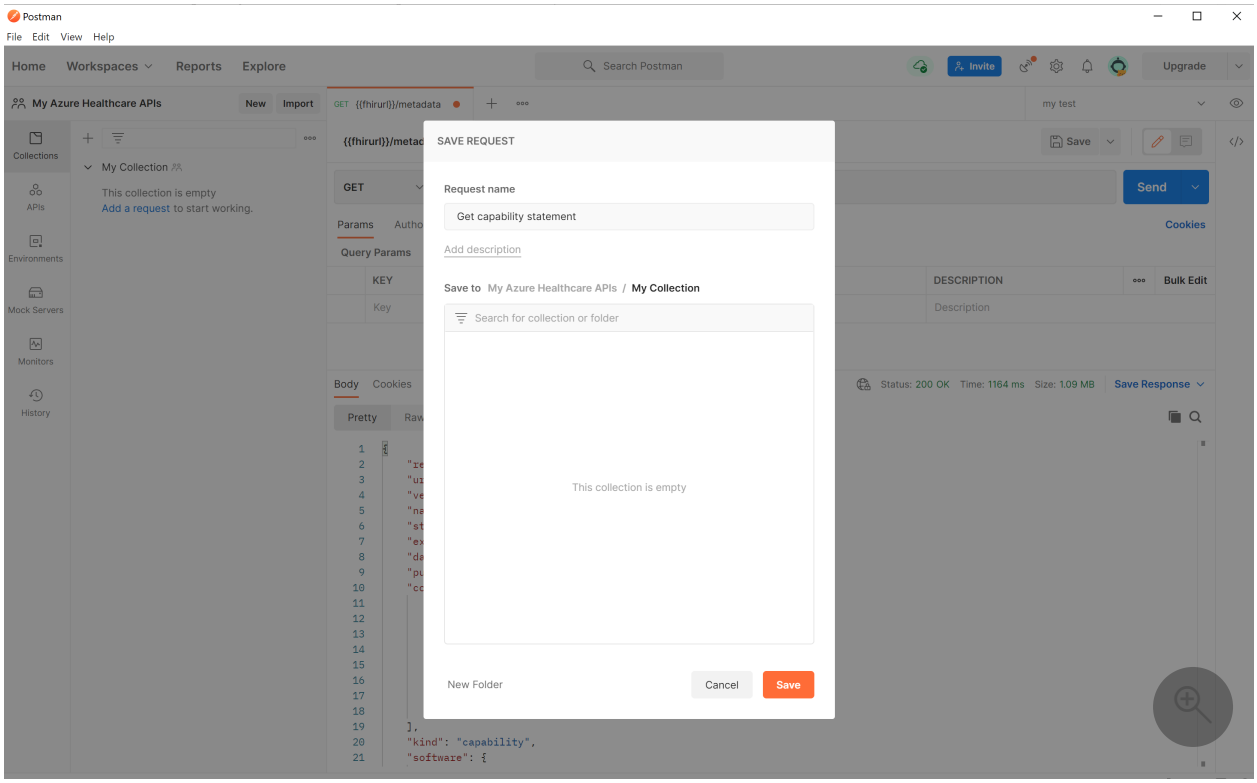


To perform health check on FHIR service, enter `{{fhirurl}}/health/check` in the GET request, and select 'Send'. You should be able to see Status of FHIR service - HTTP Status code response with 200 and OverallStatus as "Healthy" in response, means your health check is succesful.

Get capability statement

Enter `{{fhirurl}}/metadata` in the `GET` request, and select `Send`. You should see the capability statement of the FHIR service.





Get Azure AD access token

The FHIR service is secured by Azure AD. The default authentication can't be disabled. To access the FHIR service, you must get an Azure AD access token first. For more information, see Microsoft identity platform access tokens.

Create a new `POST` request:

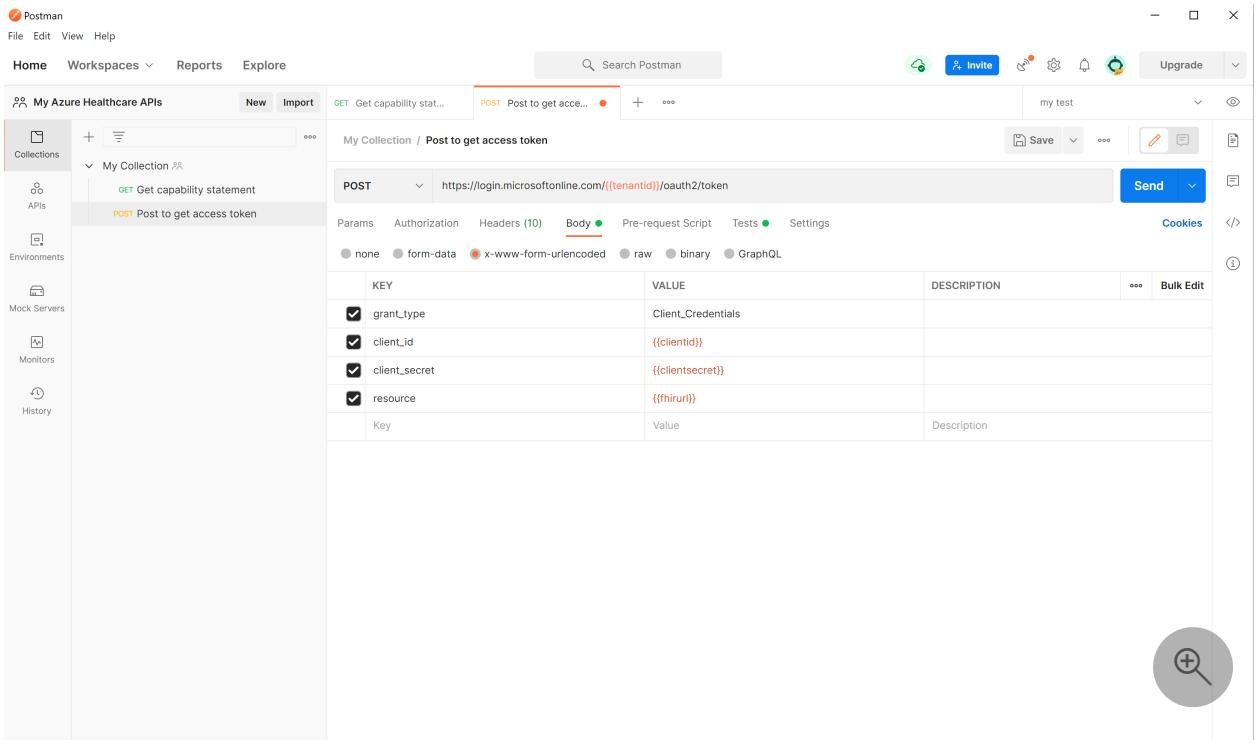
1. Enter in the request header:
   `https://login.microsoftonline.com/{{tenantid}}/oauth2/token`

2. Select the **Body** tab and select **x-www-form-urlencoded**. Enter the following values in the key and value section:

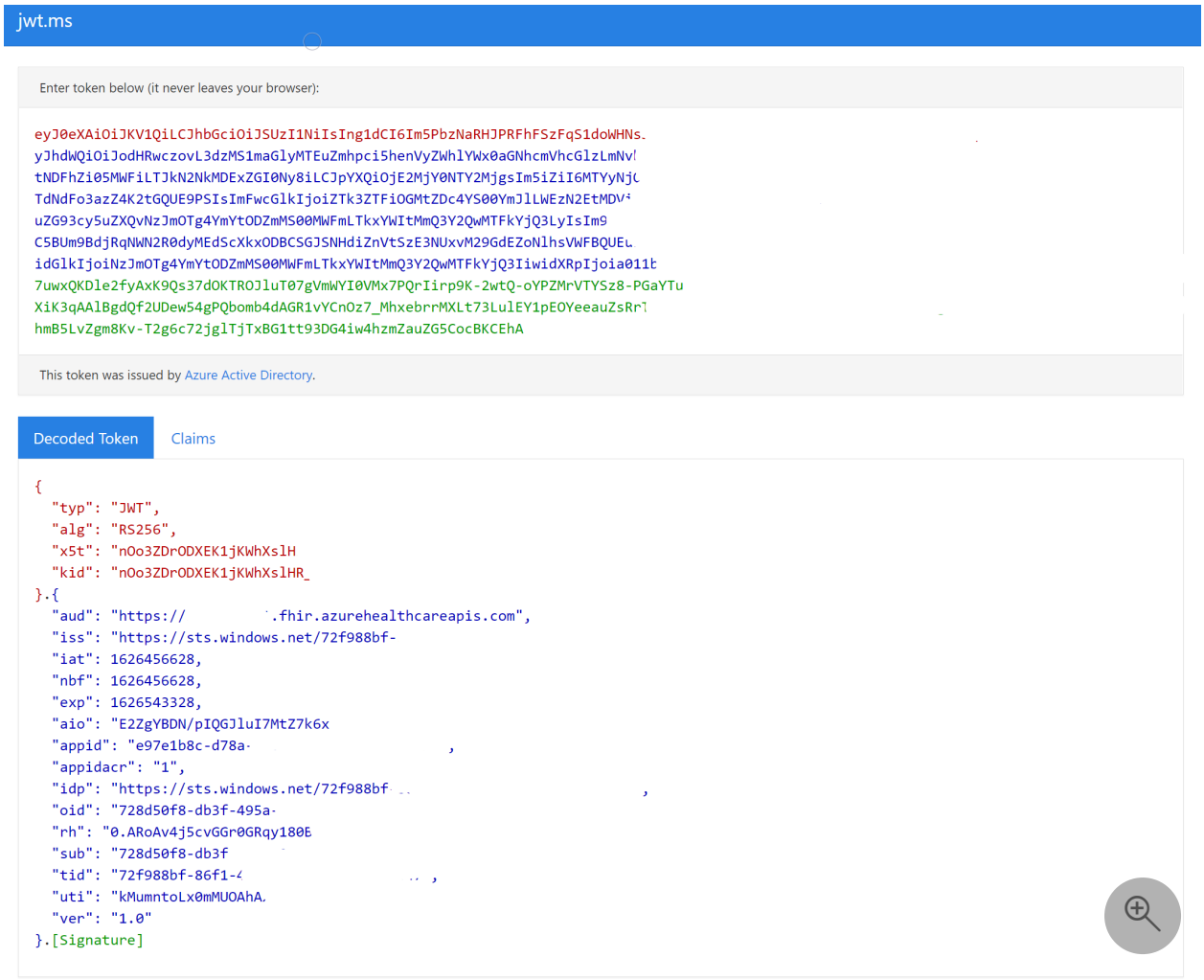
   - **grant_type**: `Client_Credentials`
   - **client_id**: `{{clientid}}`
   - **client_secret**: `{{clientsecret}}`
   - **resource**: `{{fhirurl}}`

> ⓘ Note
>
> In the scenarios where the FHIR service audience parameter is not mapped to the FHIR service endpoint url. The resource parameter value should be mapped to Audience value under FHIR Service Authentication blade.

3. Select the **Test** tab and enter in the text section: `pm.environment.set("bearerToken", pm.response.json().access_token);` To make the value available to the collection, use the pm.collectionVariables.set method. For more information on the set method and its scope level, see Using variables in scripts.
4. Select **Save** to save the settings.
5. Select **Send**. You should see a response with the Azure AD access token, which is saved to the variable `bearerToken` automatically. You can then use it in all FHIR service API requests.



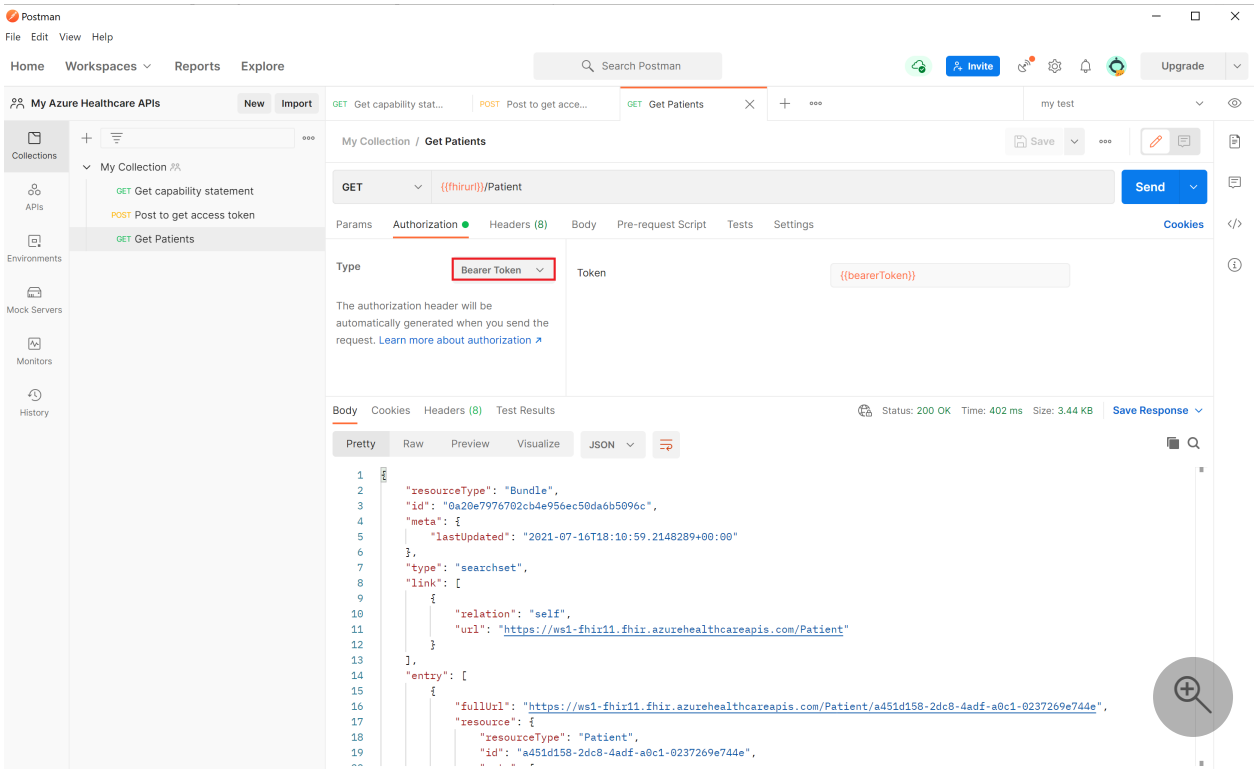
You can examine the access token using online tools such as https://jwt.ms. Select the **Claims** tab to see detailed descriptions for each claim in the token.



Get FHIR resource

After you've obtained an Azure AD access token, you can access the FHIR data. In a new `GET` request, enter `{{fhirurl}}/Patient`.

Select **Bearer Token** as authorization type. Enter `{{bearerToken}}` in the **Token** section. Select **Send**. As a response, you should see a list of patients in your FHIR resource.



Create or update your FHIR resource

After you've obtained an Azure AD access token, you can create or update the FHIR data. For example, you can create a new patient or update an existing patient.

Create a new request, change the method to “Post”, and enter the value in the request section.

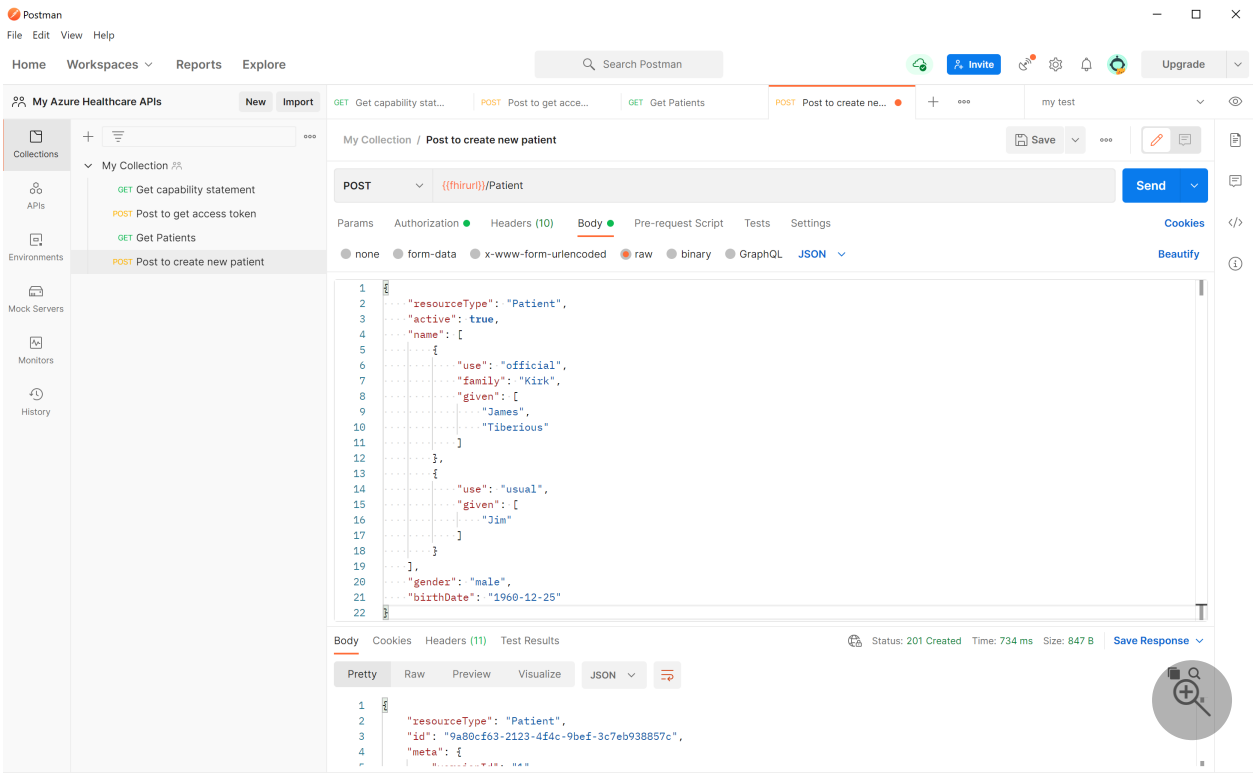
`{{fhirurl}}/Patient`

Select **Bearer Token** as the authorization type. Enter `{{bearerToken}}` in the **Token** section. Select the **Body** tab. Select the **raw** option and **JSON** as body text format. Copy and paste the text to the body section.

```
{
    "resourceType": "Patient",
    "active": true,
    "name": [
        {
            "use": "official",
            "family": "Kirk",
            "given": [
                "James",
                "Tiberious"
            ]
        },
        {
```

```
            "use": "usual",
            "given": [
                "Jim"
            ]
        }
    ],
    "gender": "male",
    "birthDate": "1960-12-25"
}
```

Select **Send**. You should see a new patient in the JSON response.



Export FHIR data

After you've obtained an Azure AD access token, you can export FHIR data to an Azure storage account.

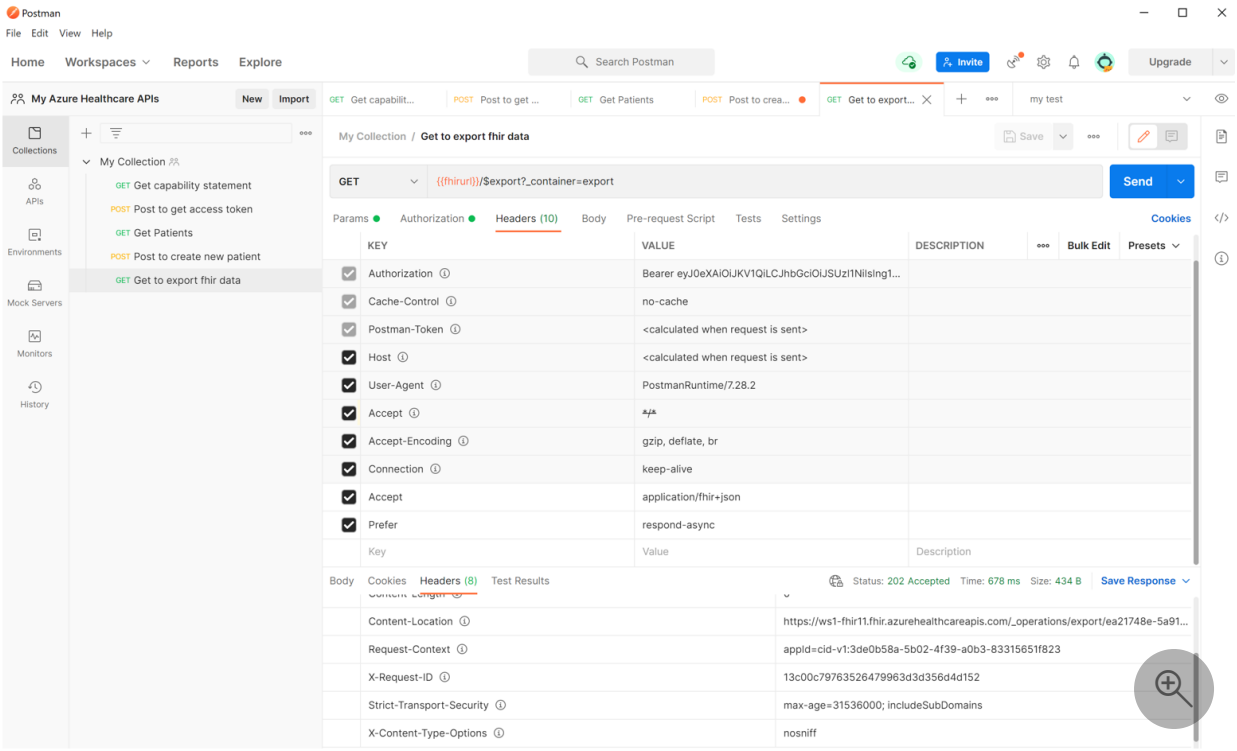
Create a new `GET` request: `{{fhirurl}}/$export?_container=export`

Select **Bearer Token** as authorization type. Enter `{{bearerToken}}` in the **Token** section. Select **Headers** to add two new headers:

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

Select **Send**. You should notice a `202 Accepted` response. Select the **Headers** tab of the response and make a note of the value in the **Content-Location**. You can use the value

to query the export job status.



Next steps

In this article, you learned how to access the FHIR service in Azure Health Data Services with Postman. For information about FHIR service in Azure Health Data Services, see

What is FHIR service?

For a starter collection of sample Postman queries, please see our samples repo on Github.

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

Access the Azure Health Data Services with cURL

Article • 05/15/2023

In this article, you'll learn how to access Azure Health Data Services with cURL.

Prerequisites

PowerShell

- An Azure account with an active subscription. Create one for free.
- If you want to run the code locally, install PowerShell and Azure Az PowerShell.
- Optionally, you can run the scripts in Visual Studio Code with the REST Client extension. For more information, see Make a link to the REST Client doc.
- Download and install cURL.

CLI

- An Azure account with an active subscription. Create one for free.
- If you want to run the code locally, install Azure CLI.
- Optionally, install a Bash shell, such as Git Bash, which it's included in Git for Windows.
- Optionally, run the scripts in Visual Studio Code with the REST Client extension. For more information, see Make a link to the REST Client doc.
- Download and install cURL.

Obtain Azure Access Token

Before accessing the Azure Health Data Services, you must grant the user or client app with proper permissions. For more information on how to grant permissions, see Azure Health Data Services authorization.

There are several different ways to obtain an Azure access token for the Azure Health Data Services.

> **Note**

Make sure that you have logged into Azure and that you are in the Azure subscription and tenant where you have deployed the Azure Health Data Services instance.

PowerShell

```
### check Azure environment and PowerShell versions
Get-AzContext
Set-AzContext -Subscription <subscriptionid>
$PSVersionTable.PSVersion
Get-InstalledModule -Name Az -AllVersions
curl --version

### get access token for the FHIR service
$fhirservice="https://<fhirservice>.fhir.azurehealthcareapis.com"
$token=(Get-AzAccessToken -ResourceUrl $fhirservice).Token

### Get access token for the DICOM service
$dicomtokenurl= "https://dicom.healthcareapis.azure.com/"
$token=$( Get-AzAccessToken -ResourceUrl $dicomtokenurl).Token
```

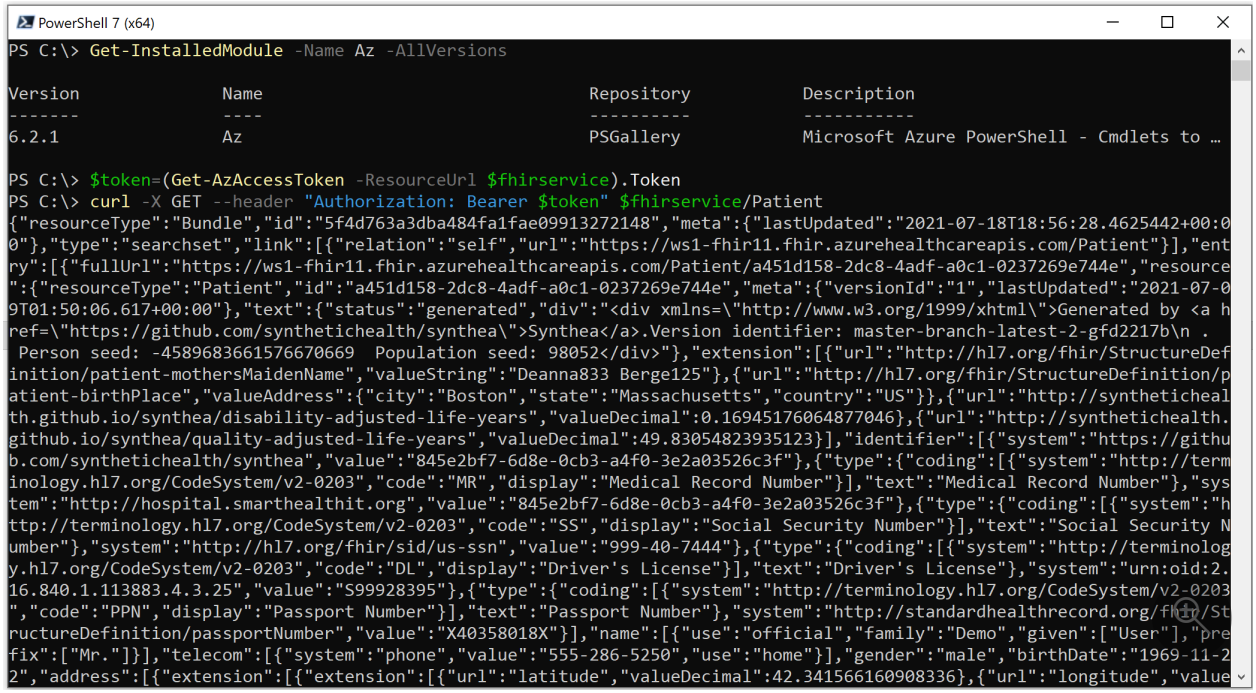
> ⓘ **Note**
>
> In the scenarios where the FHIR service audience parameter is not mapped to the FHIR service endpoint url. The resource parameter value should be mapped to Audience value under FHIR Service Authentication blade.

Access data in the FHIR service

PowerShell

```
$fhirservice="https://<fhirservice>.fhir.azurehealthcareapis.com"
```

`curl -X GET --header "Authorization: Bearer $token" $fhirservice/Patient`

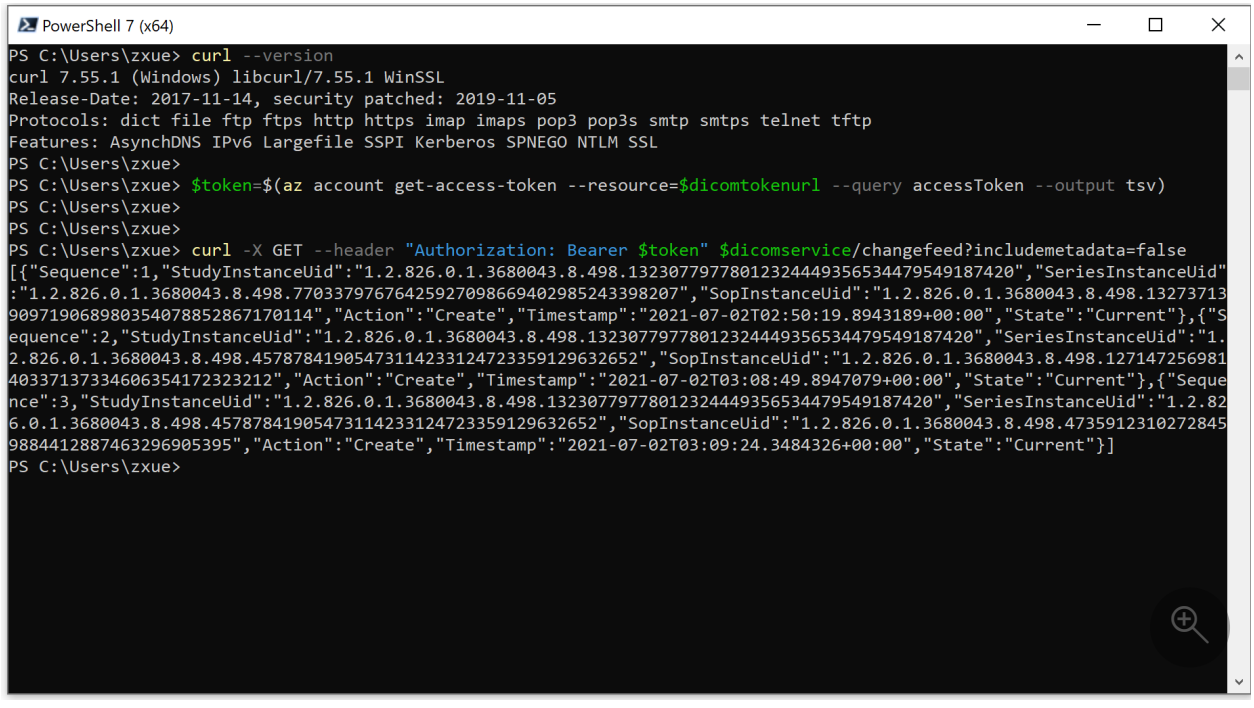


Access data in the DICOM service

PowerShell

```
$dicomservice="https://<dicomservice>.dicom.azurehealthcareapis.com"
```

`curl -X GET --header "Authorization: Bearer $token" $dicomservice/changefeed?includemetadata=false`



Next steps

In this article, you learned how to access Azure Health Data Services data using cURL.

To learn about how to access Azure Health Data Services data using REST Client extension in Visual Studio Code, see

Access Azure Health Data Services using REST Client

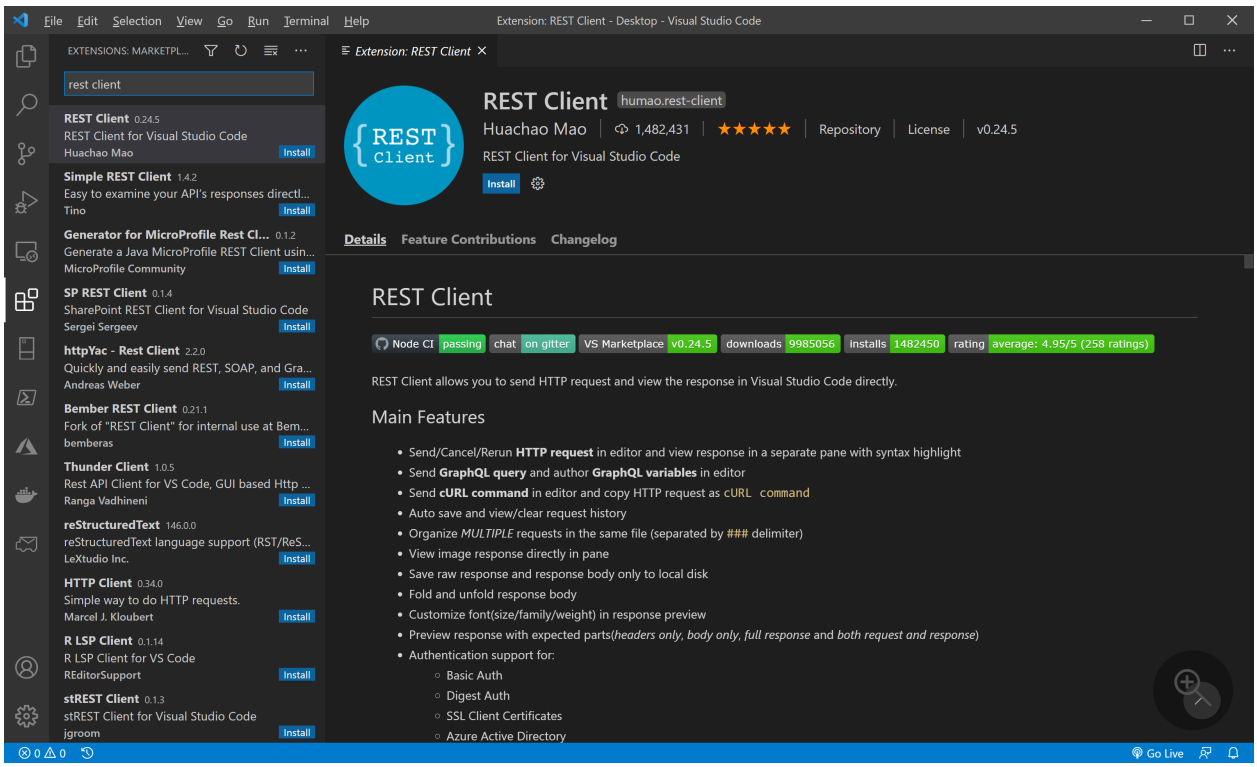
Accessing Azure Health Data Services using the REST Client Extension in Visual Studio Code

Article • 05/15/2023

In this article, you'll learn how to access Azure Health Data Services using REST Client extension in Visual Studio Code.

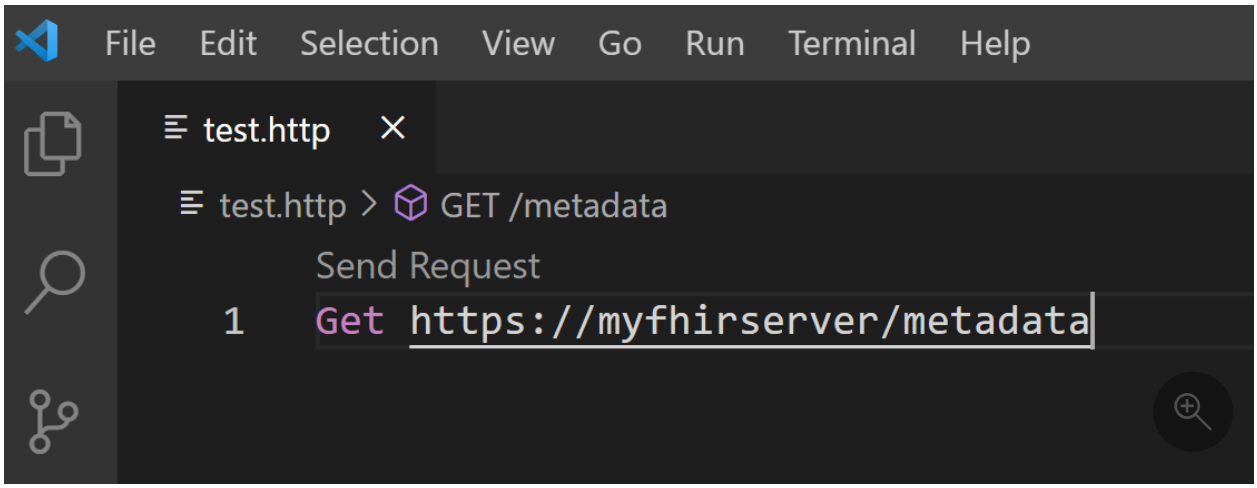
Install REST Client extension

Select the Extensions icon on the left side panel of your Visual Studio Code, and search for "REST Client". Find the REST Client extension and install.



Create a `.http` file and define variables

Create a new file in Visual Studio Code. Enter a `GET` request command line in the file, and save it as `test.http`. The file suffix `.http` automatically activates the REST Client environment. Select `Send Request` to get the metadata.



Get client application values

> **Important**
>
> Before calling the FHIR server REST API (other than getting the metadata), you must complete application registration. Make a note of your Azure **tenant ID**, **client ID**, **client secret** and the **service URL**.

While you can use values such as the client ID directly in calls to the REST API, it's a good practice that you define a few variables for these values and use the variables instead.

In your `test.http` file, include the following information obtained from registering your application:

```
### REST Client
@fhirurl =https://xxx.azurehealthcareapis.com
@clientid =xxx....
@clientsecret =xxx....
@tenantid =xxx....
```

Get Azure AD Access Token

After including the information below in your `test.http` file, hit `Send Request`. You'll see an HTTP response that contains your access token.

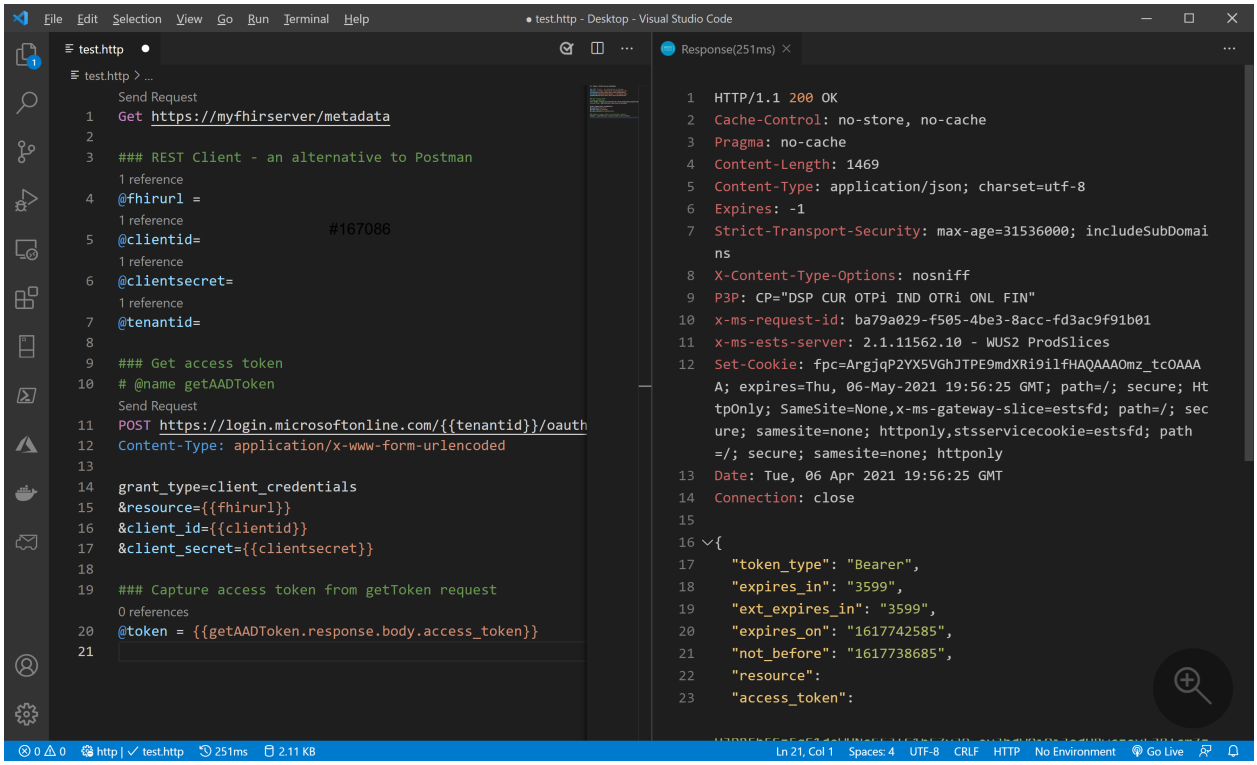
The line starting with `@name` contains a variable that captures the HTTP response containing the access token. The variable, `@token`, is used to store the access token.

> **Note**
>
> The `grant_type` of `client_credentials` is used to obtain an access token.

```
### Get access token
# @name getAADToken
POST https://login.microsoftonline.com/{{tenantid}}/oauth2/token
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials
&resource={{fhirurl}}
&client_id={{clientid}}
&client_secret={{clientsecret}}

### Extract access token from getAADToken request
@token = {{getAADToken.response.body.access_token}}
```

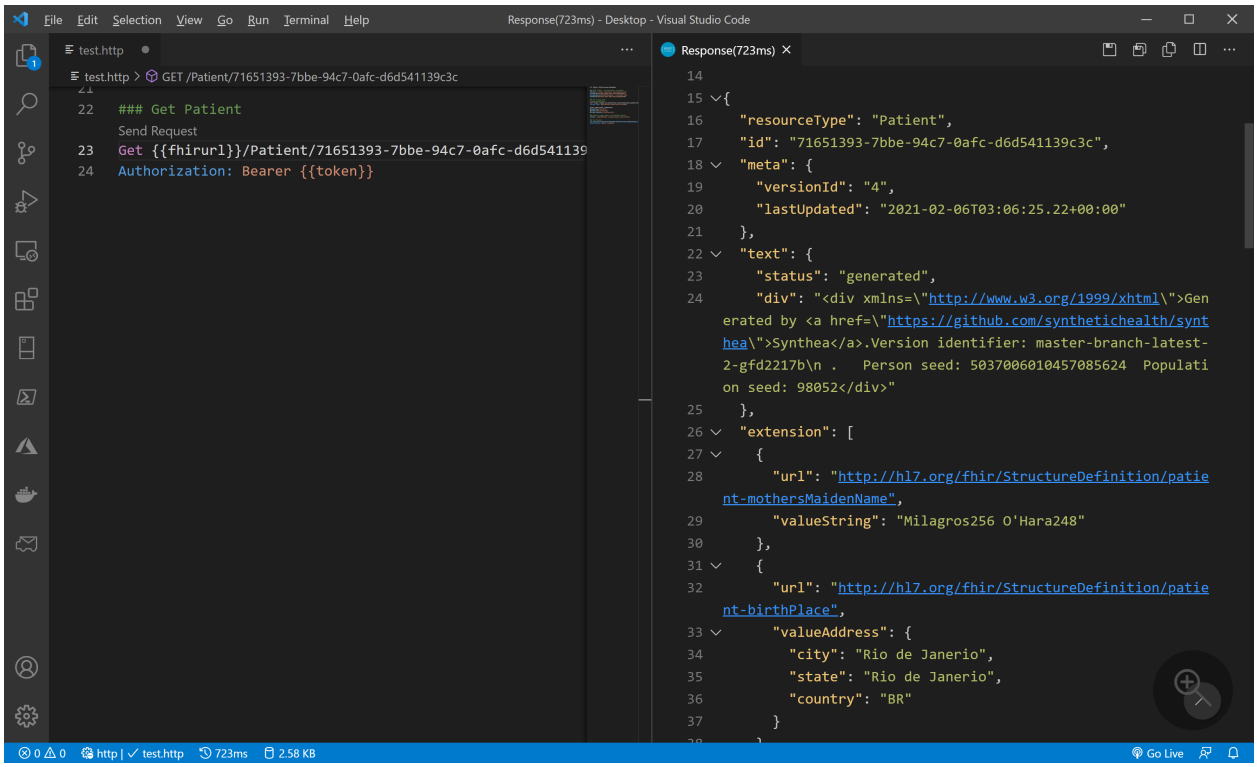


> **Note**
>
> In the scenarios where the FHIR service audience parameter is not mapped to the FHIR service endpoint url. The resource parameter value should be mapped to Audience value under FHIR Service Authentication blade.

`GET` FHIR Patient data

You can now get a list of patients or a specific patient with the `GET` request. The line with `Authorization` is the header info for the `GET` request. You can also send `PUT` or `POST` requests to create/update FHIR resources.

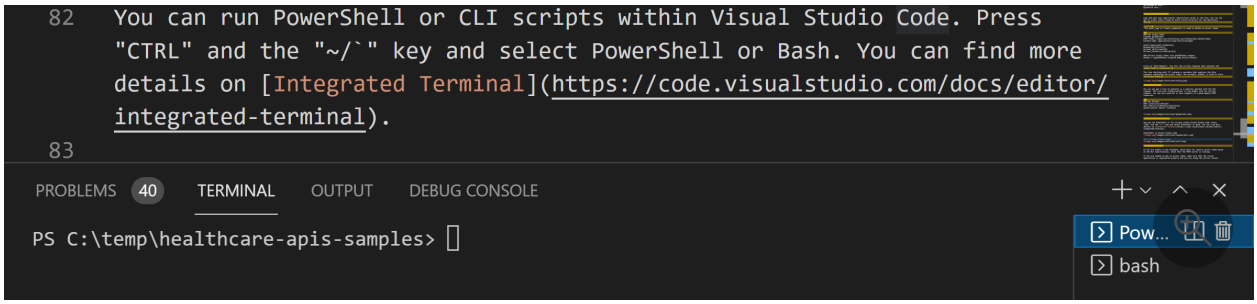
```
### GET Patient
GET {{fhirurl}}/Patient/<patientid>
Authorization: Bearer {{token}}
```



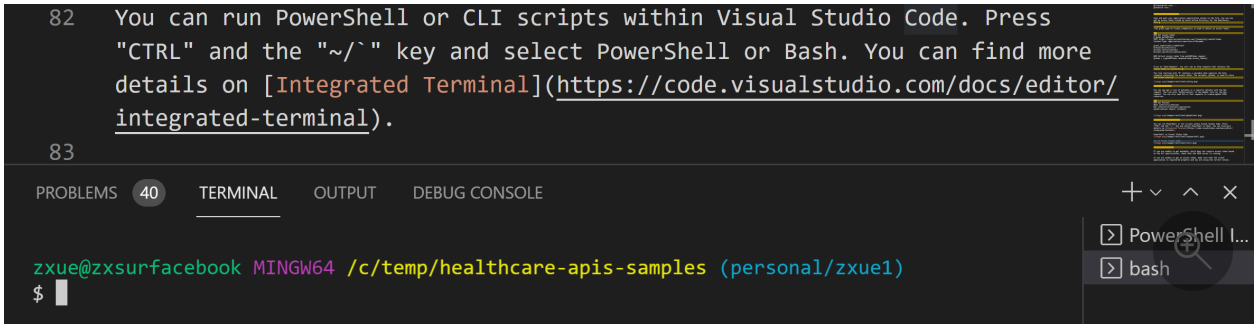
Run PowerShell or CLI

You can run PowerShell or CLI scripts within Visual Studio Code. Press `CTRL` and the `~` key and select PowerShell or Bash. You can find more details on Integrated Terminal.

PowerShell in Visual Studio Code



CLI in Visual Studio Code



Troubleshooting

If you're unable to get the metadata, which doesn't require access token based on the HL7 specification, check that your FHIR server is running properly.

If you're unable to get an access token, make sure that the client application is registered properly and you're using the correct values from the application registration step.

If you're unable to get data from the FHIR server, make sure that the client application (or the service principal) has been granted access permissions such as "FHIR Data Contributor" to the FHIR server.

Next steps

In this article, you learned how to access Azure Health Data Services data using the using the REST Client extension in Visual Studio Code.

To learn about how to validate FHIR resources against profiles in Azure Health Data Services, see

Validate FHIR resources against profiles in Azure Health Data Services

Configure cross-origin resource sharing in FHIR service

Article • 09/26/2022

What is cross-origin resource sharing in FHIR service?

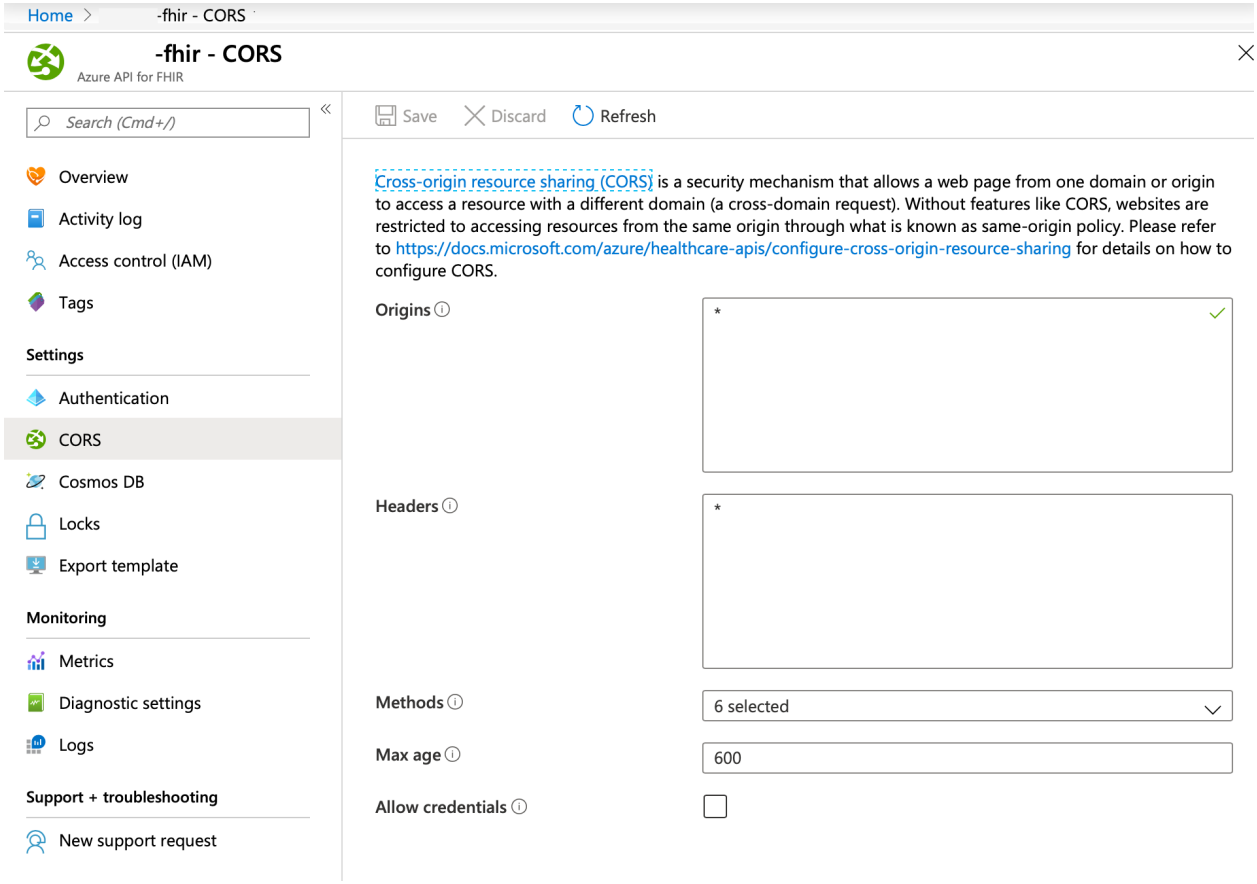
FHIR service in Azure Health Data Services (hereby called FHIR service) supports cross-origin resource sharing (CORS). CORS allows you to configure settings so that applications from one domain (origin) can access resources from a different domain, known as a cross-domain request.

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

Cross-origin resource sharing configuration settings

To configure a CORS setting in the FHIR service, specify the following settings:

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



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

Next steps

In this tutorial, we walked through how to configure a CORS setting in the FHIR service. Next, you can review how to pass the CARIN IG for Blue Button tests in Touchstone.

CARIN Implementation Guide for Blue Button®

View and enable diagnostic settings in the FHIR service

Article • 09/06/2023

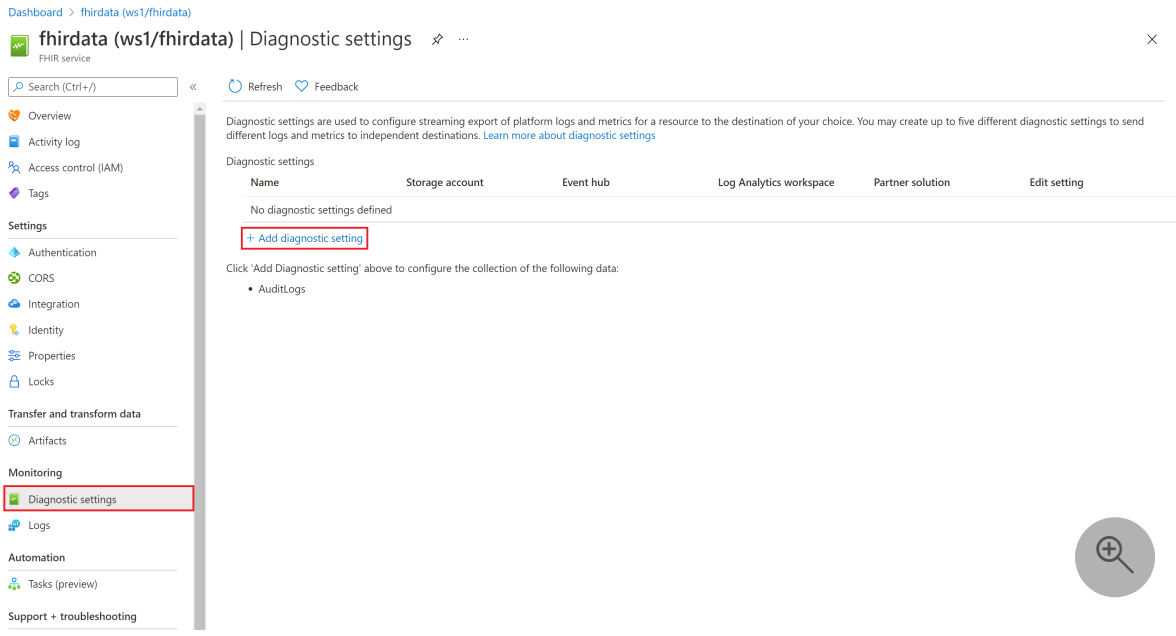
Access to diagnostic logs is essential for any healthcare service. Compliance with regulatory requirements like Health Insurance Portability and Accountability Act (HIPAA) is a must. In this article, you'll learn how to choose settings for diagnostic logs in the FHIR service within Azure Health Data Services. You'll also review some sample queries for these logs.

Steps to enable diagnostic logs

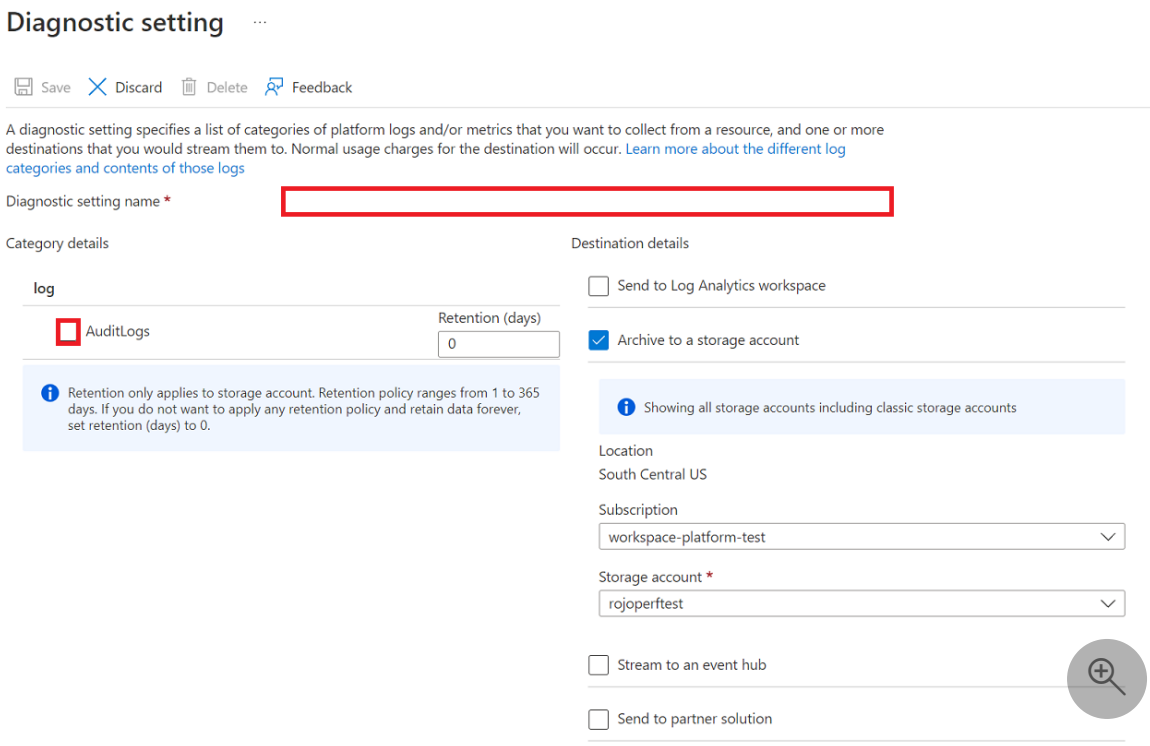
1. Select your FHIR service in the Azure portal.

2. Under **Monitoring**, select **Diagnostic settings**.

3. Select **+ Add diagnostic settings**.



4. Enter the **Diagnostic setting name**.



5. Select the method that you want to use to access your logs:

   - **Send to Log Analytics workspace** is for sending logs and metrics to a Log Analytics workspace in Azure Monitor. You need to create your Log Analytics workspace before you can select this option.

   - **Archive to a storage account** is for auditing or manual inspection. The storage account that you want to use needs to be already created. The retention option only applies to a storage account. Retention policy ranges from 1 to 365 days. If you don't want to apply any retention policy and retain data forever, set the retention (days) to 0.

   - **Stream to an event hub** is for ingestion by a third-party service or custom analytic solution. You need to create an event hub namespace and event hub policy before you can configure this option.

   - **Send to partner solution** should be selected if you've enabled a partner solution that Azure supports. For more information, see Extend Azure with solutions from partners.

6. Select **AuditLogs**.

7. Select **Save**.

> **Note**

It might take up to 15 minutes for the first logs to appear in the Log Analytics workspace, if you selected that method. If the FHIR service is moved from one resource group or subscription to another, update the settings after the move is complete.

Diagnostic log details

At this time, the FHIR service returns the following fields in a diagnostic log:

Field name	Type	Notes
`CallerIdentity`	Dynamic	A generic property bag that contains identity information.
`CallerIdentityIssuer`	String	The issuer.
`CallerIdentityObjectId`	String	The object ID.
`CallerIPAddress`	String	The caller's IP address.
`CorrelationId`	String	The correlation ID.
`FhirResourceType`	String	The resource type for which the operation was executed.
`LogCategory`	String	The log category. (In this article, we're returning `AuditLogs`.)
`Location`	String	The location of the server that processed the request. For example: `South Central US`.
`OperationDuration`	Int	The time it took to complete this request, in seconds.
`OperationName`	String	The type of operation. For example: `update` or `search-type`.
`RequestUri`	String	The request URI.
`ResultType`	String	The status of the log. Available values are `Started`, `Succeeded`, or `Failed`.
`StatusCode`	Int	The HTTP status code. For example: `200`.
`TimeGenerated`	DateTime	The date and time of the event.
`Properties`	String	The properties of `FhirResourceType`.
`SourceSystem`	String	The source system, which is always `Azure` in this case.
`TenantId`	String	The tenant ID.

Field name	Type	Notes
`Type`	String	The type of log, which is always `MicrosoftHealthcareApisAuditLog` in this case.
`_ResourceId`	String	Details about the resource.

Sample queries

You can use these basic Log Analytics queries to explore your log data:

- Run the following query to view the *100 most recent* logs:

  ```
  MicrosoftHealthcareApisAuditLogs | limit 100
  ```

- Run the following query to group operations by *FHIR resource type*:

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

- Run the following 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. In this article, you learned how to enable these logs for the FHIR service.

> **Note**
>
> Metrics will be added when the Azure Health Data Services service is generally available.

Next steps

To learn about setting custom headers on diagnostic logs ,visit

Setting custom headers for logs

Add data to audit logs by using custom HTTP headers in FHIR service

Article • 09/06/2023

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 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:

JSON

```
{ "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:

C#

```
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 FHIR service. For information about FHIR service, see

FHIR Overview

Overview of FHIR search

Article • 10/12/2022

The Fast Healthcare Interoperability Resources (FHIR®) specification defines an API for querying resources in a FHIR server database. This article will guide you through some key aspects of querying data in FHIR. For complete details about the FHIR search API, refer to the HL7 FHIR Search documentation.

Throughout this article, we'll demonstrate FHIR search syntax in example API calls with the `{{FHIR_URL}}` placeholder to represent the FHIR server URL. In the case of the FHIR service in Azure Health Data Services, this URL would be `https://<WORKSPACE-NAME>-<FHIR-SERVICE-NAME>.fhir.azurehealthcareapis.com`.

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

```
HTTP

GET {{FHIR_URL}}/Patient
```

You can also search using `POST`. To search using `POST`, the search parameters are delivered in the body of the request. This makes it easier to send queries with longer, more complex series of parameters.

With either `POST` or `GET`, if the search request is successful, you'll receive a FHIR `searchset` bundle containing the resource instance(s) returned from the search. If the search fails, you'll find the error details in an `OperationOutcome` response.

In the following sections, we'll cover the various aspects of querying resources in FHIR. Once you've reviewed these topics, refer to the FHIR search samples page, which features examples of different FHIR search methods.

Search parameters

When you do a search in FHIR, you are searching the database for resources that match certain search criteria. The FHIR API specifies a rich set of search parameters for fine-tuning search criteria. Each resource in FHIR carries information as a set of elements, and search parameters work to query the information in these elements. In a FHIR search API

call, if a positive match is found between the request's search parameters and the corresponding element values stored in a resource instance, then the FHIR server returns a bundle containing the resource instance(s) whose elements satisfied the search criteria.

For each search parameter, the FHIR specification defines the data type(s) that can be used. Support in the FHIR service for the various data types is outlined below.

Search parameter type	FHIR service in Azure Health Data Services	Azure API for FHIR	Comment
number	Yes	Yes	
date	Yes	Yes	
string	Yes	Yes	
token	Yes	Yes	
reference	Yes	Yes	
composite	Partial	Partial	The list of supported composite types is given later in this article.
quantity	Yes	Yes	
uri	Yes	Yes	
special	No	No	

Common search parameters

There are common search parameters that apply to all resources in FHIR. These are listed below, along with their support in the FHIR service:

Common search parameter	FHIR service in Azure Health Data Services	Azure API for FHIR	Comment
`_id`	Yes	Yes	
`_lastUpdated`	Yes	Yes	
`_tag`	Yes	Yes	
`_type`	Yes	Yes	
`_security`	Yes	Yes	

Common search parameter	FHIR service in Azure Health Data Services	Azure API for FHIR	Comment
`_profile`	Yes	Yes	
`_has`	Yes	Yes	
`_query`	No	No	
`_filter`	No	No	
`_list`	No	No	
`_text`	No	No	
`_content`	No	No	

Resource-specific parameters

The FHIR service in Azure Health Data Services supports almost all resource-specific search parameters defined in the FHIR specification. Search parameters that are not supported are listed 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:

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

To view the supported search parameters in the capability statement, navigate to `CapabilityStatement.rest.resource.searchParam` for the resource-specific search parameters and `CapabilityStatement.rest.searchParam` for search parameters that apply to all resources.

> **Note**
>
> The FHIR service in Azure Health Data Services does not automatically index search parameters that are not defined in the base FHIR specification. However, the FHIR service does support **custom search parameters**.

Composite search parameters

Composite searches in FHIR allow you to search against element pairs as logically connected units. For example, if you were searching for observations where the height of the patient was over 60 inches, you would want to make sure that a single property of the observation contained the height code *and* a value greater than 60 inches (the value should only pertain to height). You wouldn't want to return a positive match on an observation with the height code *and* an arm to arm length over 60 inches, for example. Composite search parameters prevent this problem by searching against pre-specified pairs of elements whose values must both meet the search criteria for a positive match to occur.

The FHIR service in Azure Health Data Services supports the following search parameter type pairings for composite searches:

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

For more information, see the HL7 Composite Search Parameters documentation.

> **Note**
>
> Composite search parameters do not support modifiers, as per the FHIR specification.

Modifiers & prefixes

Modifiers allow you to qualify search parameters with additional conditions. Below is a list of FHIR modifiers and their support in the FHIR service:

Modifiers	FHIR service in Azure Health Data Services	Azure API for FHIR	Comment
`:missing`	Yes	Yes	
`:exact`	Yes	Yes	
`:contains`	Yes	Yes	
`:text`	Yes	Yes	

Modifiers	FHIR service in Azure Health Data Services	Azure API for FHIR	Comment
`:type` (reference)	Yes	Yes	
`:not`	Yes	Yes	
`:below` (uri)	Yes	Yes	
`:above` (uri)	Yes	Yes	
`:in` (token)	No	No	
`:below` (token)	No	No	
`:above` (token)	No	No	
`:not-in` (token)	No	No	

For search parameters that have a specific order (numbers, dates, and quantities), you can use a prefix before the parameter value to refine the search criteria (e.g. `Patient?_lastUpdated=gt2022-08-01` where the prefix `gt` means "greater than"). The FHIR service in Azure Health Data Services supports all prefixes defined in the FHIR standard.

Search result parameters

FHIR specifies a set of search result parameters to help manage the information returned from a search. For detailed information on how to use search result parameters in FHIR, refer to the HL7 website. Below is a list of FHIR search result parameters and their support in the FHIR service.

Search result parameters	FHIR service in Azure Health Data Services	Azure API for FHIR	Comment
`_elements`	Yes	Yes	
`_count`	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 included in the bundle.

Search result parameters	FHIR service in Azure Health Data Services	Azure API for FHIR	Comment
`_include`	Yes	Yes	Items retrieved with `_include` are limited to 100. `_include` on PaaS and OSS on Azure Cosmos DB doesn't support `:iterate` (#2137).
`_revinclude`	Yes	Yes	Items retrieved with `_revinclude` are limited to 100. `_revinclude` on PaaS and OSS on Azure Cosmos DB doesn't support `:iterate` (#2137). There's also an incorrect status code for a bad request #1319.
`_summary`	Yes	Yes	
`_total`	Partial	Partial	`_total=none` and `_total=accurate`
`_sort`	Partial	Partial	`sort=_lastUpdated` is supported on the FHIR service. For the FHIR service and the OSS SQL DB FHIR servers, sorting by strings and dateTime fields are supported. For Azure API for FHIR and OSS Azure Cosmos DB databases created after April 20, 2021, sort is supported on first name, last name, birthdate, and clinical date.
`_contained`	No	No	
`_containedType`	No	No	
`_score`	No	No	

> **Note**
>
> By default, `_sort` arranges records in ascending order. You can also use the prefix `-` to sort in descending order. The FHIR service only allows you to sort on a single field at a time.

By default, the FHIR service in Azure Health Data Services 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 include the `Prefer` header and set `handling=strict`.

Chained & reverse chained searching

A chained search allows you to perform fine-targeted queries for resources that have a reference to 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
```

The `.` in the above request steers the path of the chained search to the target parameter (`name` in this case).

Similarly, you can do a reverse chained search with the `_has` parameter. This allows you to retrieve resource instances by specifying criteria on other resources that reference the resources of interest. For examples of chained and reverse chained search, refer to the FHIR search examples page.

Pagination

As mentioned above, the results from a FHIR search will be available in paginated form at a link provided in the `searchset` bundle. By default, the FHIR service will display 10 search results per page, but this can be increased (or decreased) by setting the `_count` parameter. If there are more matches than fit on one page, the bundle will include a `next` link. Repeatedly fetching from the `next` link will yield the subsequent pages of results. Note that the `_count` parameter value cannot exceed 1000.

Currently, the FHIR service in Azure Health Data Services only supports the `next` link and doesn't support `first`, `last`, or `previous` links in bundles returned from a search.

Next steps

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

FHIR search examples

Defining custom search parameters

Article • 03/17/2023

The FHIR specification defines a set of search parameters that apply to all resources. Additionally, FHIR defines many search parameters that are specific to certain resources. There are scenarios, however, 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 custom search parameters for use in the FHIR service in Azure Health Data Services.

> **Note**
>
> Each time you create, update, or delete a search parameter, you'll need to run a **reindex job** to enable the search parameter for live production. Below we will outline how you can test search parameters before reindexing the entire FHIR service database.

Create new search parameter

To create a new search parameter, you need to `POST` a `SearchParameter` resource to the FHIR service database.

```HTTP
POST {{FHIR_URL}}/SearchParameter
```

The examples below demonstrate creating new custom search parameter

Create new search parameter per definition in Implementation Guide

The code example below shows how to add the US Core Race search parameter to the `Patient` resource type in your FHIR service database.

```HTTP
{
  "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"
}
```

Create new search parameter for resource attributes with reference type

The code example shows how to create a custom search parameter to search MedicationDispense resources based on the location where they were dispensed. This is an example of adding custom search parameter for a Reference type.

HTTP

```
{
 "resourceType": "SearchParameter",
  "id": "a3c28d46-fd06-49ca-aea7-5f9314ef0497",
```

```
  "url": "{{An absolute URI that is used to identify this search
parameter}}",
  "version": "1.0",
  "name": "MedicationDispenseLocationSearchParameter",
  "status": "active",
  "description": "Search parameter for MedicationDispense by location",
  "code": "location",
  "base": ["MedicationDispense"],
  "target": ["Location"],
  "type": "reference",
  "expression": "MedicationDispense.location"
}
```

> ⓘ **Note**
>
> The new search parameter will appear in the capability statement of the FHIR service 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 service. If you cannot find the `SearchParameter` in the capability statement, then you still need to reindex your database to activate the search parameter. You can `POST` multiple search parameters before triggering a reindex operation.

Important elements of a `SearchParameter` resource:

- `url`: A unique key to describe the search parameter. 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 the **code** element is the name used for the search parameter when it's included in an API call. For the example above with the "US Core Race" extension, you would search with `GET {{FHIR_URL}}/Patient?race=<code>` where `<code>` is in the value set from the specified coding system. This call would retrieve all patients of a certain race.
- `base`: Describes which resource type(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 resource types.
- `target`: Describes which resource type(s) the search parameter matches to.
- `type`: Describes the data type for the search parameter. Type is limited by the support for data types in the FHIR service. This means that you can't define a

search parameter of type Special or define a composite search parameter unless it's 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 isn't required by the specification. This is because you need either the expression or the xpath syntax and the FHIR service ignores the xpath syntax.

Test new search parameters

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

First, you can test a new search parameter to see what values will be returned. By running the command below against a specific resource instance (by supplying the resource ID), you get back a list of value pairs with the search parameter name and the value stored in the corresponding element. This list includes all of the search parameters for the resource. You can scroll through to find the search parameter you created. Running this command won't change any behavior in your FHIR service.

HTTP

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

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

HTTP

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

The result looks like this:

JSON

```
{
  "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 your new search parameter. To reindex a single resource:

```
HTTP

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

Running this `POST` call sets the indices for any search parameters defined for the resource instance specified in the request. This call does make a change to the FHIR service database. Now you can search and set the `x-ms-use-partial-indices` header to `true`, which causes the FHIR service to return results for any of the resources that have the search parameter indexed, even if not all resource instances of that type have it indexed.

Continuing with our example, you could index one patient to enable `SearchParameter`:

```
HTTP

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

And then do a test search

1. For the patient by race:

```
HTTP

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

1. For Location (reference type):

HTTP

```
{{fhirurl}}/MedicationDispense?location=<locationid referenced in
MedicationDispense Resource>
x-ms-use-partial-indices: true
```

After you've tested your new search parameter and confirmed that it's working as expected, run or schedule your reindex job so the new search parameter(s) can be used in live production.

See Running a reindex job for information on how to reindex your FHIR service database.

Update a search parameter

To update a search parameter, use `PUT` to create a new version of the search parameter. You must include the search parameter ID in the `id` field in the body of the `PUT` request as well as the `PUT` request string.

> ⓘ **Note**
>
> If you don't know the ID for your search parameter, you can search for it using `GET {{FHIR_URL}}/SearchParameter`. This will return all custom as well as standard search parameters, and you can scroll through the list to find the search parameter you need. You could also limit the search by name. As shown in the example request below, the name of the custom `SearchParameter` resource instance is `USCoreRace`. You could search for this `SearchParameter` resource by name using `GET {{FHIR_URL}}/SearchParameter?name=USCoreRace`.

HTTP

```
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 of the above request will be an updated `SearchParameter` resource.

> ⚠ **Warning**
>
> Be careful when updating search parameters. Changing an existing search parameter 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:

```
HTTP

DELETE {{FHIR_URL}}/SearchParameter/{{SearchParameter_ID}}
```

> ⚠ **Warning**
>
> Be careful when deleting search parameters. Changing an existing search parameter 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 custom search parameter. Next you can learn how to reindex your FHIR service database. For more information, see

How to run a reindex job

Running a reindex job

Article • 06/08/2023

There are scenarios where you may have search parameters in the FHIR service in Azure Health Data Services that haven't yet been indexed. This scenario is relevant when you define your own custom search parameters. Until the search parameter is indexed, it can't be used in live production. This article covers how to run a reindex job to index any custom search parameters that haven't yet been indexed in your FHIR service database.

> ⚠ **Warning**
>
> It's important that you read this entire article before getting started. A reindex job can be very performance intensive. This article discusses options for how to throttle and control a reindex job.

How to run a reindex job

Reindex job can be executed against entire FHIR service database and against specific custom search parameter.

Run reindex job on entire FHIR service database

To run reindex job, use the following `POST` call with the JSON formatted `Parameters` resource in the request body:

JSON

```
POST {{FHIR_URL}}/$reindex
content-type: application/fhir+json
{

"resourceType": "Parameters",

"parameter": []

}
```

Leave the `"parameter": []` field blank (as shown) if you don't need to tweak the resources allocated to the reindex job.

If the request is successful, you receive a **201 Created** status code in addition to a `Parameters` resource in the response.

JSON

```
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": "138035"
    },
    "parameter": [
        {
            "name": "id",
            "valueString": "560c7c61-2c70-4c54-b86d-c53a9d29495e"
        },
        {
            "name": "lastModified",
            "valueDateTime": "2023-06-08T04:52:44.0974408+00:00"
        },
        {
            "name": "queuedTime",
            "valueDateTime": "2023-06-08T04:52:44.0974406+00:00"
        },
        {
            "name": "totalResourcesToReindex",
            "valueDecimal": 0.0
        },
        {
            "name": "resourcesSuccessfullyReindexed",
            "valueDecimal": 0.0
        },
        {
            "name": "progress",
            "valueDecimal": 0.0
        },
        {
            "name": "status",
            "valueString": "Queued"
        },
        {
            "name": "maximumConcurrency",
            "valueDecimal": 3.0
        },
        {
            "name": "queryDelayIntervalInMilliseconds",
            "valueDecimal": 500.0
        },
        {
            "name": "maximumNumberOfResourcesPerQuery",
```

```
            "valueDecimal": 100.0
        }
    ]
}
```

Run reindex job against specific custom search parameter

To run reindex job against specific custom search parameter, use the following `POST` call with the JSON formatted `Parameters` resource in the request body:

JSON

```
POST {{FHIR_URL}}/$reindex
content-type: application/fhir+json
{

"resourceType": "Parameters",

"parameter": [
      "name": "targetSearchParameterTypes",
      "valueString": "{url of custom search parameter. In case of multiple
custom search parameters, url list can be comma seperated.}"

]

}
```

> **Note**
>
> To check the status of a reindex job or to cancel the job, you'll need the reindex ID. This is the `"id"` carried in the `"parameter"` value returned in the response. 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 call:

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

An example response:

JSON

```
{
    "resourceType": "Parameters",
    "id": "560c7c61-2c70-4c54-b86d-c53a9d29495e",
    "meta": {
        "versionId": "138087"
    },
    "parameter": [
        {
            "name": "id",
            "valueString": "560c7c61-2c70-4c54-b86d-c53a9d29495e"
        },
        {
            "name": "startTime",
            "valueDateTime": "2023-06-08T04:54:53.2943069+00:00"
        },
        {
            "name": "endTime",
            "valueDateTime": "2023-06-08T04:54:54.4052272+00:00"
        },
        {
            "name": "lastModified",
            "valueDateTime": "2023-06-08T04:54:54.4053002+00:00"
        },
        {
            "name": "queuedTime",
            "valueDateTime": "2023-06-08T04:52:44.0974406+00:00"
        },
        {
            "name": "totalResourcesToReindex",
            "valueDecimal": 2.0
        },
        {
            "name": "resourcesSuccessfullyReindexed",
            "valueDecimal": 2.0
        },
        {
            "name": "progress",
            "valueDecimal": 100.0
        },
        {
            "name": "status",
            "valueString": "Completed"
        },
        {
            "name": "maximumConcurrency",
            "valueDecimal": 3.0
        },
        {
            "name": "resources",
            "valueString": "{{LIST_OF_IMPACTED_RESOURCES}}"
        },
        {
            "name": "resourceReindexProgressByResource (CountReindexed of
Count)",
```

```
            "valueString": "{{RESOURCE_TYPE:REINDEXED_COUNT OF
TOTAL_COUNT}}"
        },
        {
            "name": "searchParams",
            "valueString": "{{LIST_OF_SEARCHPARAM_URLS}}"
        },
        {
            "name": "queryDelayIntervalInMilliseconds",
            "valueDecimal": 500.0
        },
        {
            "name": "maximumNumberOfResourcesPerQuery",
            "valueDecimal": 100.0
        }
    ]
}
```

The following information is shown in the above response:

- `totalResourcesToReindex`: Includes the total number of resources that are being reindexed in this job.
- `resourcesSuccessfullyReindexed`: The total number of resources that have already been reindexed in this job.
- `progress`: Reindex job percent complete. Equals `resourcesSuccessfullyReindexed`/`totalResourcesToReindex` x 100.
- `status`: States if the reindex job is queued, running, complete, failed, or canceled.
- `resources`: Lists all the resource types impacted by the reindex job.
- 'resourceReindexProgressByResource (CountReindexed of Count)': Provides reindexed count of the total count, per resource type. In cases where reindexing for a specific resource type is queued, only Count is provided.
- 'searchParams': Lists url of the search parameters 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. The FHIR service offers some throttling controls to help you manage how a reindex job run on your database.

> **Note**
>
> It is not uncommon on large datasets for a reindex job to run for days.

Below is a table outlining the available parameters, defaults, and recommended ranges for controlling reindex job compute resources. You can use these parameters to either speed up the process (use more compute) or slow down the process (use less compute).

Parameter	Description	Default	Available Range
`QueryDelayIntervalInMilliseconds`	The delay between each batch of resources being kicked off during the reindex job. A smaller number speeds up the job while a larger number slows it down.	500 MS (.5 seconds)	50 to 500000
`MaximumResourcesPerQuery`	The maximum number of resources included in the batch of resources to be reindexed.	100	1-5000
`MaximumConcurrency`	The number of batches done at a time.	1	1-10

If you want to use any of the parameters above, you can pass them into the `Parameters` resource when you send the initial `POST` request to start a reindex job.

JSON

```
POST {{FHIR_URL}}/$reindex
content-type: application/fhir+json
{
  "resourceType": "Parameters",
  "parameter": [
    {
      "name": "maximumConcurrency",
      "valueInteger": "3"
    },
    {
      "name": "queryDelayIntervalInMilliseconds",
      "valueInteger": "1000"
    },
    {
      "name": "maximumNumberOfResourcesPerQuery",
```

```
      "valueInteger": "1"
    }
  ]
}
```

Next steps

In this article, you've learned how to perform a reindex job in your FHIR service. To learn how to define custom search parameters, see

Defining custom search parameters

FHIR search examples

Article • 09/26/2022

Below are some examples of Fast Healthcare Interoperability Resources (FHIR®) search API calls featuring various search parameters, modifiers, chained and reverse chained searches, composite searches, `POST` search requests, and more. For a general introduction to FHIR search concepts, see Overview of FHIR Search.

Search result parameters

`_include`

`_include` lets you search for resource instances and include in the results other resources referenced by the target resource instances. For example, you can use `_include` to query for `MedicationRequest` resources and limit the search to prescriptions for a specific patient. The FHIR service would then return the `MedicationRequest` resources as well as the referenced `Patient` resource. In the example below, the request will pull all `MedicationRequest` resource instances in the database and all patients that are referenced by the `MedicationRequest` instances:

```
HTTP

GET {{FHIR_URL}}/MedicationRequest?_include=MedicationRequest:patient
```

> **Note**
>
> The FHIR service in Azure Health Data Services limits searches with `_include` and `_revinclude` to return a maximum of 100 items.

`_revinclude`

`_revinclude` allows you to search for resource instances and include in the results other resources that reference the target resource instances. For example, you can search for patients and then reverse include all encounters that reference the patients:

```
HTTP
```

```
GET {{FHIR_URL}}/Patient?_revinclude=Encounter:subject
```

`_elements`

`_elements` narrows the information in the search results to a subset of the elements defined for a resource type. The `_elements` parameter accepts a comma-separated list of base elements:

```
HTTP

GET {{FHIR_URL}}/Patient?_elements=identifier,active
```

In the above request, you'll receive a bundle of patients, but each entry will only include the identifier(s) and the patient's active status. The entries in the response will contain a `meta.tag` value of `SUBSETTED` to indicate that not all elements defined for the resource are included.

Search modifiers

`:not`

`:not` allows you to find resources with an element that does not have a given value. For example, you could search for patients who are not female:

```
HTTP

GET {{FHIR_URL}}/Patient?gender:not=female
```

In return, you would get all `Patient` resources whose `gender` element value is not `female`, including any patients with no gender value specified. This is different from searching for `Patient` resources with the `male` gender value since that would ignore patients with no specified gender.

`:missing`

`:missing` returns all resources that don't have a value for the specified element when `:missing=true`. Additionally, `:missing` returns all resources that contain the specified

element when `:missing=false`. For simple data type elements, `:missing=true` will match on all resources where an element is present but has an empty value. For example, if you want to find all `Patient` resources that are missing information on `birthdate`, you can call:

```
HTTP

GET {{FHIR_URL}}/Patient?birthdate:missing=true
```

`:exact`

`:exact` is used to search for elements with `string` data types and returns positive if the parameter value precisely matches the case and full character sequence of the element value.

```
HTTP

GET {{FHIR_URL}}/Patient?name:exact=Jon
```

This request returns `Patient` resources that have the `given` or `family` name of `Jon`. If there were patients with names such as `Jonathan` or `JON`, the search would ignore those resources as their names do not match the specified value exactly.

`:contains`

`:contains` is used to query for `string` type elements and allows for matches with the specified value anywhere within the field. `contains` is case insensitive and recognizes matching strings concatenated with other characters. For example:

```
HTTP

GET {{FHIR_URL}}/Patient?address:contains=Meadow
```

This request would return all `Patient` resources with `address` element fields that contain the string "Meadow" (case insensitive). This means you could have addresses with values such as "Meadows Lane", "Pinemeadow Place", or "Meadowlark St" that return positive matches.

Chained search

To perform search operations that cover elements contained within a referenced resource, you can "chain" a series of parameters together with `.`. For example, if you want to view all `DiagnosticReport` resources with a `subject` reference to a patient specified by `name`:

```
HTTP

GET {{FHIR_URL}}/DiagnosticReport?subject:Patient.name=Sarah
```

This request would return all `DiagnosticReport` resources with a patient subject named "Sarah". The `.` points the chained search to the `name` element within the referenced `Patient` resource.

Another common use of FHIR search is finding all encounters for a specific patient. To do a regular (non-chained) search for `Encounter` resources that reference a `Patient` with a given `id`:

```
HTTP

GET {{FHIR_URL}}/Encounter?subject=Patient/78a14cbe-8968-49fd-a231-
d43e6619399f
```

Using chained search, you can find all `Encounter` resources that reference patients whose details match a search parameter. The example below demonstrates how to search for encounters referencing patients narrowed by `birthdate`:

```
HTTP

GET {{FHIR_URL}}/Encounter?subject:Patient.birthdate=1987-02-20
```

This would return all `Encounter` instances that reference patients with the specified `birthdate` value.

In addition, you can initiate multiple chained searches by using the `&` operator, which allows searching against multiple references in one request. In such cases with `&`, chained search "independently" scans for each element value:

```
HTTP
```

```
GET {{FHIR_URL}}/Patient?general-
practitioner:Practitioner.name=Sarah&general-
practitioner:Practitioner.address-state=WA
```

This would return all `Patient` resources that have a reference to "Sarah" as a `generalPractitioner` plus a reference to a `generalPractitioner` that has an address in the state of Washington. In other words, if a patient had a `generalPractitioner` named Sarah from New York state and another `generalPractitioner` named Bill from Washington state, this would meet the conditions for a positive match when doing this search.

For scenarios in which the search requires a logical AND condition that strictly checks for paired element values, refer to the **composite search** examples below.

Reverse chained search

Using reverse chained search in FHIR allows you to search for target resource instances referenced by other resources. In other words, you can search for resources based on the properties of resources that refer to them. This is accomplished with the `_has` parameter. For example, the `Observation` resource has a search parameter `patient` that checks for a reference to a `Patient` resource. To find all `Patient` resources that are referenced by an `Observation` with a specific `code`:

HTTP

```
GET {{FHIR_URL}}/Patient?_has:Observation:patient:code=527
```

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

In addition, reverse chained search can have a recursive structure. For example, if you want to search for all patients referenced by an `Observation` where the observation is referenced by an `AuditEvent` from a specific practitioner named `janedoe`:

HTTP

```
GET {{FHIR_URL}}/Patient?
_has:Observation:patient:_has:AuditEvent:entity:agent:Practitioner.name=jane
doe
```

Composite search

To search for resources that contain elements grouped together as logically connected pairs, FHIR defines composite search, which joins single parameter values together with the `$` operator – forming a connected pair of parameters. In a composite search, a positive match occurs when the intersection of element values satisfies all conditions set in the paired search parameters. For example, if you want to find all `DiagnosticReport` resources that contain a potassium value less than `9.2`:

HTTP

```
GET {{FHIR_URL}}/DiagnosticReport?result.code-value-quantity=2823-3$lt9.2
```

The paired elements in this case would be the `code` element (from an `Observation` resource referenced as the `result`) and the `value` element connected with the `code`. Following the code with the `$` operator sets the `value` condition as `lt` (for "less than") `9.2` (for the potassium mmol/L value).

Composite search parameters can also be used to filter multiple component code value quantities with a logical OR. For example, to query for observations with diastolic blood pressure greater than 90 OR systolic blood pressure greater than 140:

HTTP

```
GET {{FHIR_URL}}/Observation?component-code-value-
quantity=http://loinc.org|8462-4$gt90,http://loinc.org|8480-6$gt140
```

Note how `,` functions as the logical OR operator between the two conditions.

View the next entry set

The maximum number of resources that can be returned at once from a search query is 1000. However, you might have more than 1000 resource instances that match the search query and you want to retrieve the next set of results after the first 1000 entries. In such a case, you would use the continuation (i.e. `"next"`) token `url` value in the `searchset` bundle returned from the search:

JSON

```
    "resourceType": "Bundle",
    "id": "98731cb7-3a39-46f3-8a72-afe945741bd9",
    "meta": {
        "lastUpdated": "2021-04-22T09:58:16.7823171+00:00"
    },
    "type": "searchset",
    "link": [
        {
            "relation": "next",
            "url": "{{FHIR_URL}}/Patient?
_sort=_lastUpdated&ct=WzUxMDAxNzc1NzgzODc5MjAwODBd"
        },
        {
            "relation": "self",
            "url": "{{FHIR_URL}}/Patient?_sort=_lastUpdated"
        }
    ],
```


You would make a `GET` request for the provided URL:

HTTP

```
GET {{FHIR_URL}}/Patient?_sort=_lastUpdated&ct=WzUxMDAxNzc1NzgzODc5MjAwODBd
```

This would return the next set of entries for your search results. The `searchset` bundle is the complete set of search result entries, and the continuation token `url` is the link provided by the FHIR service to retrieve the entries that don't fit in the first subset (due to the restriction on the maximum number of entries returned for one page).

Search using `POST`

All of the search examples mentioned above use `GET` requests. However, you can also make FHIR search API calls using `POST` with the `_search` parameter:

HTTP

```
POST {{FHIR_URL}}/Patient/_search?_id=45
```

This request would return the `Patient` resource instance with the given `id` value. Just as with `GET` requests, the server determines which resource instances satisfy the condition(s) and returns a bundle in the HTTP response.

Another feature of searching with `POST` is that it lets you submit the query parameters as a form body:

```HTTP
POST {{FHIR_URL}}/Patient/_search
content-type: application/x-www-form-urlencoded

name=John
```

Next steps

In this article, you learned about searching in FHIR using search parameters, modifiers, and other methods. For more information about FHIR search, see

Overview of FHIR Search

Overview of $convert-data

Article • 08/03/2023

> **Note**
>
> **Fast Healthcare Interoperability Resources (FHIR®)** is an open healthcare specification.

By using the `$convert-data` operation in the FHIR service, you can convert health data from various formats to FHIR R4 data. The `$convert-data` operation uses Liquid templates from the FHIR Converter project for FHIR data conversion. You can customize these conversion templates as needed. Currently, the `$convert-data` operation supports four types of data conversion:

- HL7v2 to FHIR R4
- C-CDA to FHIR R4
- JSON to FHIR R4 (intended for custom conversion mappings)
- FHIR STU3 to FHIR R4

> **Note**
>
> You can use the `$convert-data` endpoint as a component within an ETL (extract, transform, and load) pipeline for the conversion of health data from various formats (for example: HL7v2, CCDA, JSON, and FHIR STU3) into the **FHIR format**. You can create an ETL pipeline for a complete workflow as you convert your health data. We recommend that you use an ETL engine that's based on **Azure Logic Apps** or **Azure Data Factory**. As an example, a workflow might include: data ingestion, performing `$convert-data` operations, validation, data pre/post processing, data enrichment, data deduplication, and loading the data for persistence in the **FHIR service**.

Use the $convert-data endpoint

The `$convert-data` operation is integrated into the FHIR service as a REST API action. You can call the `$convert-data` endpoint as follows:

```
POST {{fhirurl}}/$convert-data
```

The health data for conversion is delivered to the FHIR service in the body of the `$convert-data` request. If the request is successful, the FHIR service returns a FHIR Bundle response with the data converted to FHIR R4.

Parameters

A `$convert-data` operation call packages the health data for conversion inside a JSON-formatted parameters in the body of the request. The parameters are described in the following table:

Parameter name	Description	Accepted values
inputData	Data payload to be converted to FHIR.	For `Hl7v2`: string For `Ccda`: XML For `Json`: JSON For `FHIR STU3`: JSON
inputDataType	Type of data input.	`Hl7v2`, `Ccda`, `Json`, `Fhir`

templateCollectionReference	Reference to an OCI image template collection in Azure Container Registry. The reference is to an image that contains Liquid templates to use for conversion. It can refer either to default templates or to a custom template image that's registered within the FHIR service. The following sections cover customizing the templates, hosting them on Azure Container Registry, and registering to the FHIR service.	For **default/sample** templates:
		HL7v2 templates:
		`microsofthealth/fhirconverter:default`
		`microsofthealth/hl7v2templates:default`
		C-CDA templates:
		`microsofthealth/ccdatemplates:default`
		JSON templates:
		`microsofthealth/jsontemplates:default`
		FHIR STU3 templates:
		`microsofthealth/stu3tor4templates:default`
		
		For **custom** templates:
		`<RegistryServer>/<imageName>@<imageDigest>`,
		`<RegistryServer>/<imageName>:<imageTag>`

Parameter name	Description	Accepted values
rootTemplate	The root template to use while transforming the data.	For **HL7v2**: ADT_A01, ADT_A02, ADT_A03, ADT_A04, ADT_A05, ADT_A08, ADT_A11, ADT_A13, ADT_A14, ADT_A15, ADT_A16, ADT_A25, ADT_A26, ADT_A27, ADT_A28, ADT_A29, ADT_A31, ADT_A47, ADT_A60, OML_O21, ORU_R01, ORM_O01, VXU_V04, SIU_S12, SIU_S13, SIU_S14, SIU_S15, SIU_S16, SIU_S17, SIU_S26, MDM_T01, MDM_T02 For **C-CDA**: CCD, ConsultationNote, DischargeSummary, HistoryandPhysical, OperativeNote, ProcedureNote, ProgressNote, ReferralNote, TransferSummary For **JSON**: ExamplePatient, Stu3ChargeItem For **FHIR STU3**: FHIR STU3 resource name (for example: Patient, Observation, Organization)

Note

FHIR STU3 to FHIR R4 templates are Liquid templates that provide mappings of field differences only between a FHIR STU3 resource and its equivalent resource in the FHIR R4 specification. Some of the FHIR STU3 resources are renamed or removed from FHIR R4. For more information about the resource differences and constraints for FHIR STU3 to FHIR R4 conversion, see **Resource differences and constraints for FHIR STU3 to FHIR R4 conversion**.

Note

JSON templates are sample templates for use in building your own conversion mappings. They are *not* default templates that adhere to any pre-defined health data message types. JSON itself is not specified as a health data format, unlike HL7v2 or C-CDA. Therefore, instead of providing default JSON templates, we provide some sample JSON templates that you can use as a starting point for your own customized mappings.

> ⚠ **Warning**
>
> Default templates are released under the MIT License and are *not* supported by Microsoft Support.
>
> The default templates are provided only to help you get started with your data conversion workflow. These default templates are not intended for production and might change when Microsoft releases updates for the FHIR service. To have consistent data conversion behavior across different versions of the FHIR service, you must do the following:
>
> 1. Host your own copy of the templates in an Azure Container Registry instance.
> 2. Register the templates to the FHIR service.
> 3. Use your registered templates in your API calls.
> 4. Verify that the conversion behavior meets your requirements.
>
> For more information on hosting your own templates, see **Host your own templates**

Sample request

JSON


```
{
    "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^D^^^OR
GDR|\nPID|1|3735064194^^^SIMULATOR MRN^MRN|3735064194^^^SIMULATOR
MRN^MRN~2021051528^^^NHSNBR^NHSNMBR||Kinmonth^Joanna^Chelsea^^Ms^^D||1987062
4000000|F|||89 Transaction House^Handmaiden Street^Wembley^^FV75
4GJ^GBR^HOME||020 3614 5541^PRN||||||||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^D^^^ORGDR|||CAR|||||||
||160947289167713138 76^^^^visitid|||||||||||||||||||||ARRIVED|||20200508131
015||"
        },
        {
            "name": "inputDataType",
            "valueString": "Hl7v2"
        },
```

```
        {
            "name": "templateCollectionReference",
            "valueString": "microsofthealth/fhirconverter:default"
        },
        {
            "name": "rootTemplate",
            "valueString": "ADT_A01"
        }
    ]
}
```

Sample response

JSON

```
{
    "resourceType": "Bundle",
    "type": "batch",
    "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"
            }
        }
    ]
}
```

The outcome of FHIR conversion is a FHIR Bundle as a batch.

- The FHIR Bundle should align with the expectations of the FHIR R4 specification - Bundle - FHIR v4.0.1.
- If you're trying to validate against a specific profile, you need to do some post processing by utilizing the FHIR $validate operation.

Next steps

In this article, you learned about the `$convert-data` operation and how to use the endpoint for converting health data to FHIR R4 by using the FHIR service in the Azure Health Data Service.

To learn how to configure settings for `$convert-data` using the Azure portal, see

Configure settings for $convert-data using the Azure portal

To learn how to troubleshoot `$convert-data`, see

Troubleshoot $convert-data

To learn about the frequently asked questions (FAQs) for `$convert-data`, see

Frequently asked questions about $convert-data

Configure settings for $convert-data using the Azure portal

Article • 08/28/2023

> **Note**
>
> **Fast Healthcare Interoperability Resources (FHIR®)** is an open healthcare specification.

In this article, learn how to configure settings for `$convert-data` using the Azure portal to convert your existing health data into FHIR R4.

Default templates

Microsoft publishes a set of predefined sample Liquid templates from the FHIR Converter project to support FHIR data conversion. These templates are only provided to help get you started with your data conversion workflow. It's recommended that you customize and host your own templates that support your own data conversion requirements. For information on customized templates, see Customize templates.

The default templates are hosted in a public container registry and require no further configurations or settings for your FHIR service. To access and use the default templates for your conversion requests, ensure that when invoking the `$convert-data` operation, the `templateCollectionReference` request parameter has the appropriate value based on the type of data input.

- HL7v2 templates
- C-CDA templates
- JSON templates
- FHIR STU3 templates

> **⚠ Warning**
>
> Default templates are released under the MIT License and are *not* supported by Microsoft Support.
>
> The default templates are provided only to help you get started with your data conversion workflow. These default templates are not intended for production and might change when Microsoft releases updates for the FHIR service. To have

consistent data conversion behavior across different versions of the FHIR service, you must do the following:

1. Host your own copy of the templates in an **Azure Container Registry** (ACR) instance.
2. Register the templates to the FHIR service.
3. Use your registered templates in your API calls.
4. Verify that the conversion behavior meets your requirements.

For more information on hosting your own templates, see **Host your own templates**

Customize templates

You can use the FHIR Converter Visual Studio Code extension to customize templates according to your specific requirements. The extension provides an interactive editing experience and makes it easy to download Microsoft-published templates and sample data.

> **Note**
>
> The FHIR Converter extension for Visual Studio Code is available for HL7v2, C-CDA, and JSON Liquid templates. FHIR STU3 to FHIR R4 Liquid templates are currently not supported.

The provided default templates can be used as a base starting point if needed, on top of which your customizations can be added. When making updates to the templates, consider following these guidelines to avoid unintended conversion results. The template should be authored in a way such that it yields a valid structure for a FHIR bundle resource.

For instance, the Liquid templates should have a format such as the following code:

JSON

```
<liquid assignment line 1 >
<liquid assignment line 2 >
.
.
<liquid assignment line n >
{
    "resourceType": "Bundle",
```

```
    "type": "xxx",
    <...liquid code...>
    "identifier":
    {
        "value":"xxxxx",
    },
    "id":"xxxx",
    "entry": [
    <...liquid code...>
   ]
}
```

The overall template follows the structure and expectations for a FHIR bundle resource, with the FHIR bundle JSON being at the root of the file. If you choose to add custom fields to the template that aren't part of the FHIR specification for a bundle resource, the conversion request could still succeed. However, the converted result could potentially have unexpected output and wouldn't yield a valid FHIR bundle resource that can be persisted in the FHIR service as is.

For example, consider the following code:

JSON

```
<liquid assignment line 1 >
<liquid assignment line 2 >
.
.
<liquid assignment line n >
{
   “customfield_message”: “I will have a message here”,
    “customfield_data”: {
      "resourceType": "Bundle",
      "type": "xxx",
      <...liquid code...>
      "identifier":
      {
        "value":"xxxxx",
      },
       "id":"xxxx",
       "entry": [
      <...liquid code...>
    ]
  }
}
```

In the example code, two example custom fields `customfield_message` and `customfield_data` that aren't FHIR properties per the specification and the FHIR bundle resource seem to be nested under `customfield_data` (that is, the FHIR bundle JSON isn't at the root of the file). This template doesn't align with the expected structure around a

FHIR bundle resource. As a result, the conversion request might succeed using the provided template. However, the returned converted result could potentially have unexpected output (due to certain post conversion processing steps being skipped). It wouldn't be considered a valid FHIR bundle (since it's nested and has non FHIR specification properties) and attempting to persist the result in your FHIR service fails.

Host your own templates

It's recommended that you host your own copy of templates in an Azure Container Registry (ACR) instance. ACR can be used to host your custom templates and support with versioning.

Hosting your own templates and using them for `$convert-data` operations involves the following seven steps:

1. Create an Azure Container Registry instance
2. Push the templates to your Azure Container Registry instance
3. Enable Azure Managed identity in your FHIR service instance
4. Provide Azure Container Registry access to the FHIR service managed identity
5. Register the Azure Container Registry server in the FHIR service
6. Configure the Azure Container Registry firewall for secure access
7. Verify the $convert-data operation

Step 1: Create an Azure Container Registry instance

Read the Introduction to container registries in Azure and follow the instructions for creating your own ACR instance. We recommend that you place your ACR instance in the same resource group as your FHIR service.

Step 2: Push the templates to your Azure Container Registry instance

After you create an ACR instance, you can use the **FHIR Converter: Push Templates** command in the FHIR Converter extension to push your custom templates to your ACR instance. Alternatively, you can use the Template Management CLI tool for this purpose.

To maintain different versions of custom templates in your Azure Container Registry, you may push the image containing your custom templates into your ACR instance with different image tags.

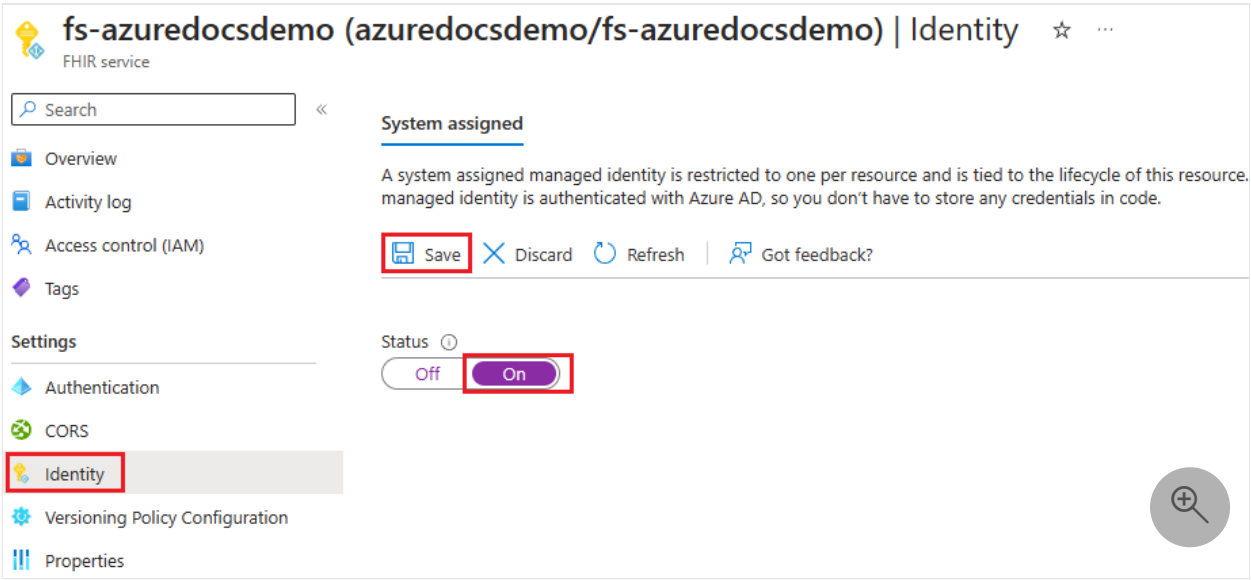
- For more information about ACR registries, repositories, and artifacts, see About registries, repositories, and artifacts.
- For more information about image tag best practices, see Recommendations for tagging and versioning container images.

To reference specific template versions in the API, be sure to use the exact image name and tag that contains the versioned template to be used. For the API parameter `templateCollectionReference`, use the appropriate **image name + tag** (for example: `<RegistryServer>/<imageName>:<imageTag>`).

Step 3: Enable Azure Managed identity in your FHIR service instance

1. Go to your instance of the FHIR service in the Azure portal, and then select the **Identity** option.

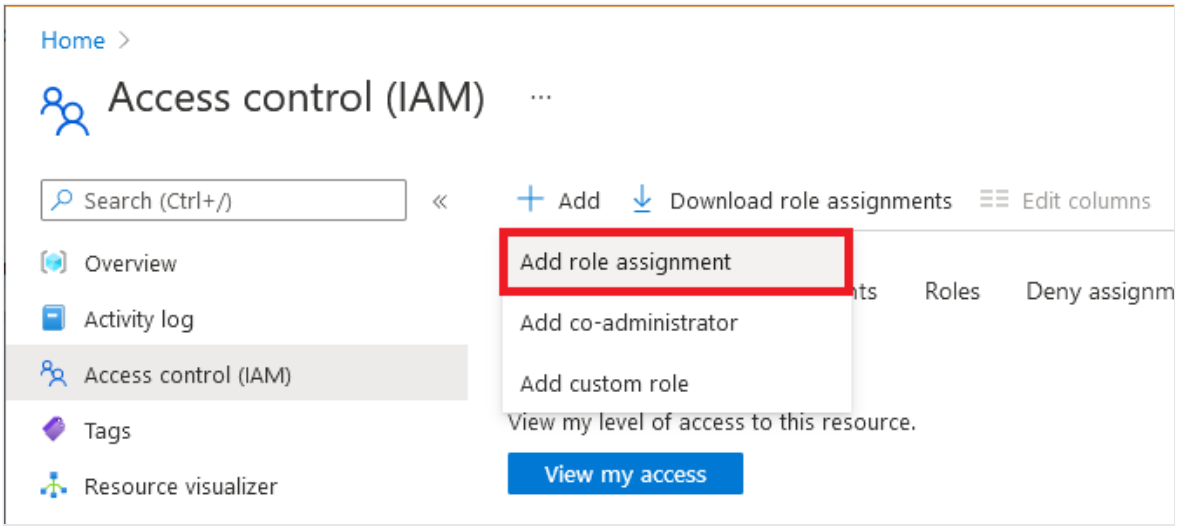
2. Change the **Status** to **On** and select **Save** to enable the system-managed identity in the FHIR service.



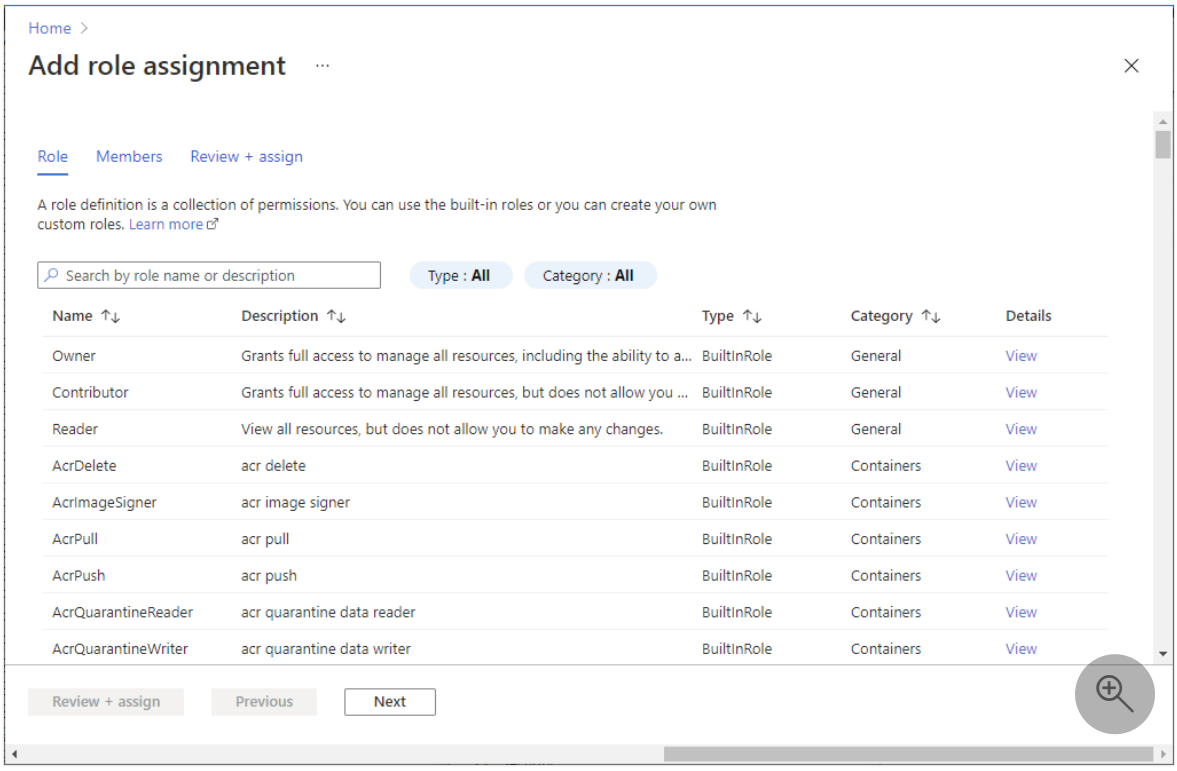
Step 4: Provide Azure Container Registry access to the FHIR service managed identity

1. In your resource group, go to your **Container Registry** instance, and then select the **Access control (IAM)** tab.

2. Select **Add** > **Add role assignment**. If the **Add role assignment** option is unavailable, ask your Azure administrator to grant you the permissions for performing this task.



3. On the **Role** pane, select the AcrPull role.



4. On the **Members** tab, select **Managed identity**, and then select **Select members**.

5. Select your Azure subscription.

6. Select **System-assigned managed identity**, and then select the FHIR service you're working with.

7. On the **Review + assign** tab, select **Review + assign** to assign the role.

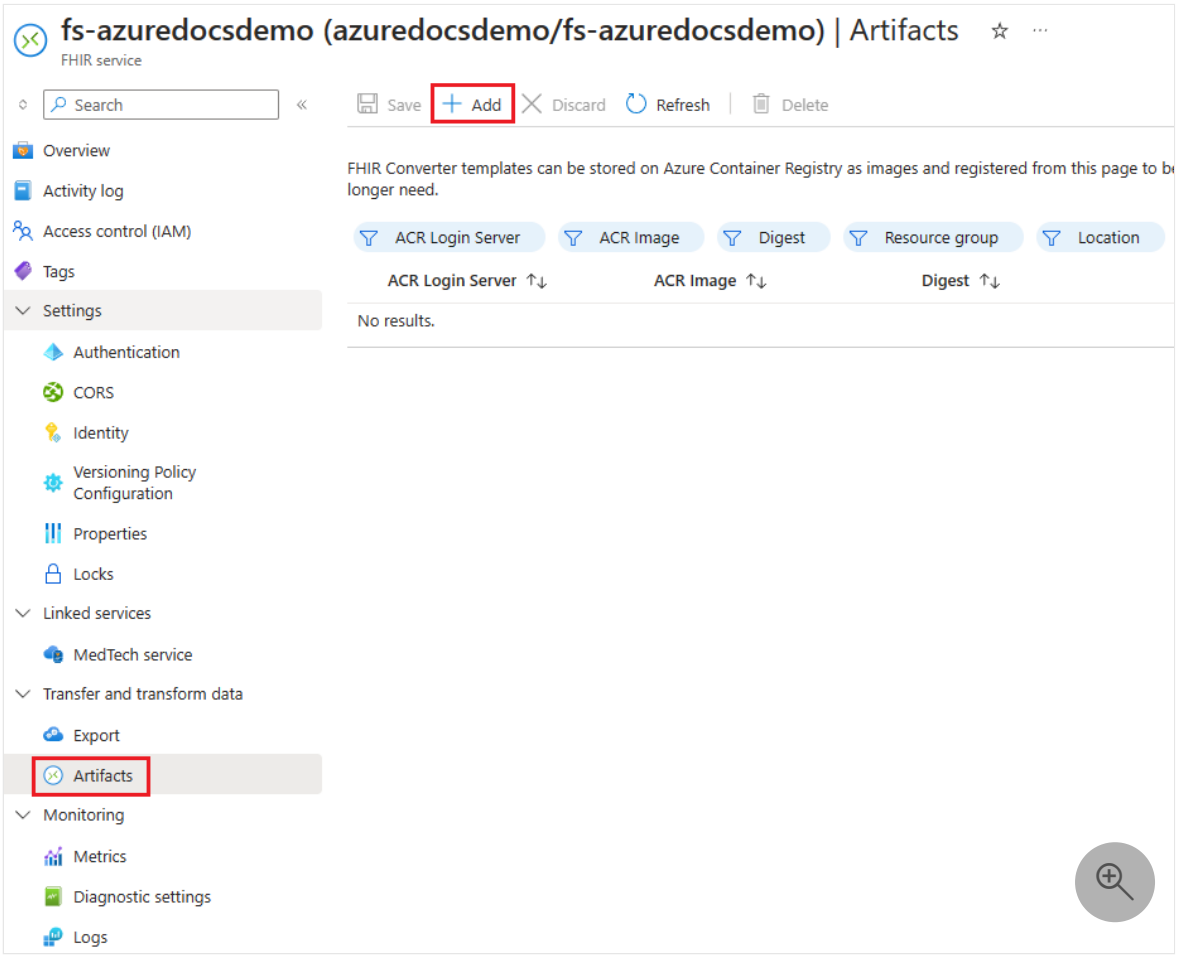
For more information about assigning roles in the Azure portal, see Azure built-in roles.

Step 5: Register the Azure Container Registry server in the FHIR service

You can register the ACR server by using the Azure portal.

To use the Azure portal:

1. In your FHIR service instance, under **Transfer and transform data**, select **Artifacts**. A list of currently registered Azure Container Registry servers is displayed.
2. Select **Add** and then, in the dropdown list, select your registry server.
3. Select **Save**.



You can register up to 20 ACR servers in the FHIR service.

> ⓘ **Note**
>
> It might take a few minutes for the registration to take effect.

Step 6: Configure the Azure Container Registry firewall for secure access

There are many methods for securing ACR using the built-in firewall depending on your particular use case.

- Connect privately to an Azure container registry using Azure Private Link
- Configure public IP network rules
- Azure Container Registry mitigating data exfiltration with dedicated data endpoints
- Restrict access to a container registry using a service endpoint in an Azure virtual network
- Allow trusted services to securely access a network-restricted container registry
- Configure rules to access an Azure container registry behind a firewall
- Azure IP Ranges and Service Tags – Public Cloud

> **Note**
>
> The FHIR service has been registered as a trusted Microsoft service with Azure Container Registry.

Step 7: Verify the $convert-data operation

Make a call to the `$convert-data` operation by specifying your template reference in the `templateCollectionReference` parameter:

`<RegistryServer>/<imageName>@<imageDigest>`

You should receive a `bundle` response that contains the health data converted into the FHIR format.

Next steps

In this article, you've learned how to configure the settings for `$convert-data` to begin converting various health data formats into the FHIR format.

For an overview of `$convert-data`, see

Overview of $convert-data

To learn how to troubleshoot `$convert-data`, see

Troubleshoot $convert-data

To learn about the frequently asked questions (FAQs) for `$convert-data`, see

Frequently asked questions about $convert-data

Transform HL7v2 data to FHIR R4 with the $convert-data operation and Azure Data Factory

Article • 09/05/2023

> **Note**
>
> **Fast Healthcare Interoperability Resources (FHIR®)** is an open healthcare specification.

In this article, we detail how to use Azure Data Factory (ADF) with the `$convert-data` operation to transform HL7v2 data to FHIR R4. The transformed results are then persisted within an Azure storage account with Azure Data Lake Storage (ADLS) Gen2 capabilities.

Prerequisites

Before getting started, ensure you have taken the following steps:

1. Deploy an instance of the FHIR service. The FHIR service is used to invoke the $convert-data operation.

2. By default, the ADF pipeline in this scenario uses the predefined templates provided by Microsoft for conversion. If your use case requires customized templates, set up your Azure Container Registry instance to host your own templates to be used for the conversion operation.

3. Create storage account(s) with Azure Data Lake Storage Gen2 (ADLS Gen2) capabilities by enabling a hierarchical namespace and container(s) to store the data to read from and write to.

   > **Note**
   >
   > You can create and use either one or separate ADLS Gen2 accounts and containers to:
   >
   > - Store the HL7v2 data to be transformed (for example: the source account and container the pipeline will read the data to be transformed

from).

- Store the transformed FHIR R4 bundles (for example: the destination account and container the pipeline will write the transformed result to).
- Store the errors encountered during the transformation (for example: the destination account and container the pipeline will write execution errors to).

4. Create an instance of ADF, which serves as a business logic orchestrator. Ensure that a system-assigned managed identity has been enabled.

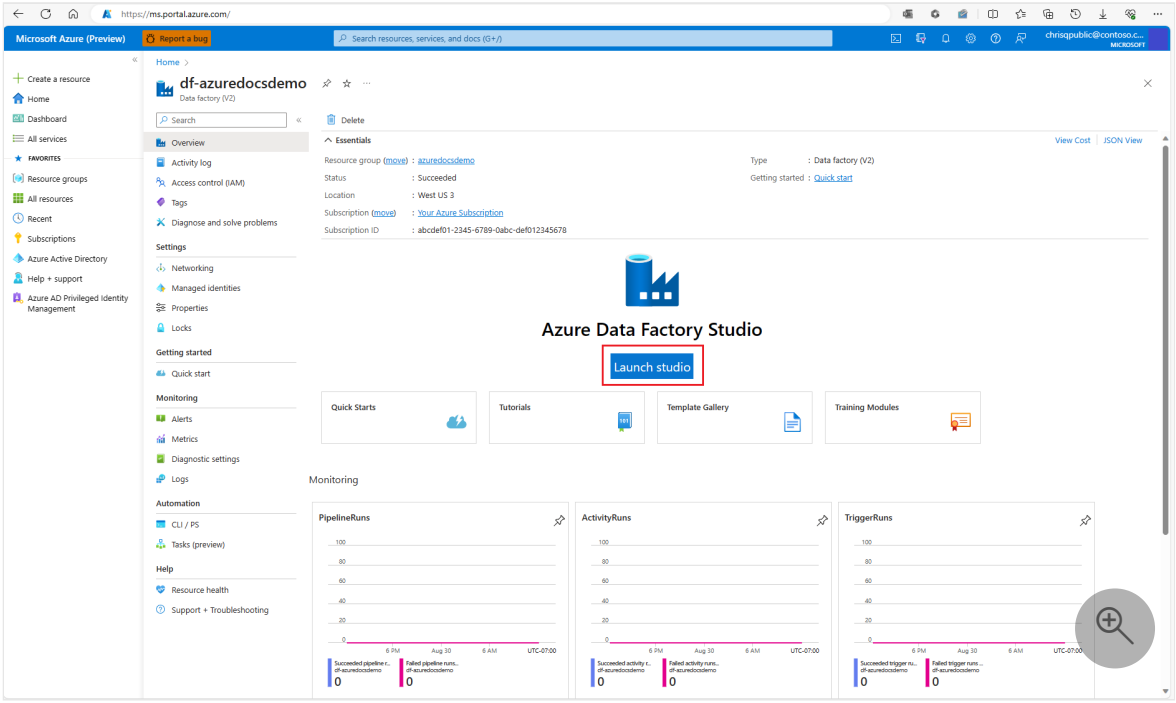
5. Add the following Azure role-based access control (Azure RBAC) assignments to the ADF system-assigned managed identity:

    - **FHIR Data Converter** role to grant permission to the FHIR service.
    - **Storage Blob Data Contributor** role to grant permission to the ADLS Gen2 account.

Configure an Azure Data Factory pipeline

In this example, an ADF pipeline is used to transform HL7v2 data and persist transformed FHIR R4 bundle in a JSON file within the configured destination ADLS Gen2 account and container.

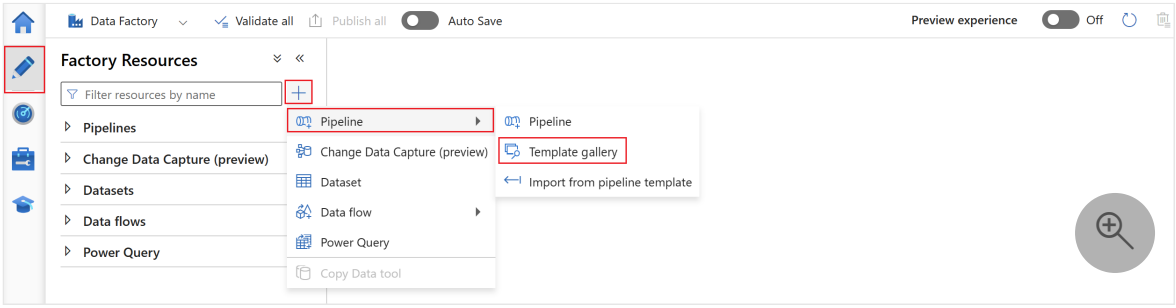
1. From the Azure portal, open your Azure Data Factory instance and select **Launch Studio** to begin.



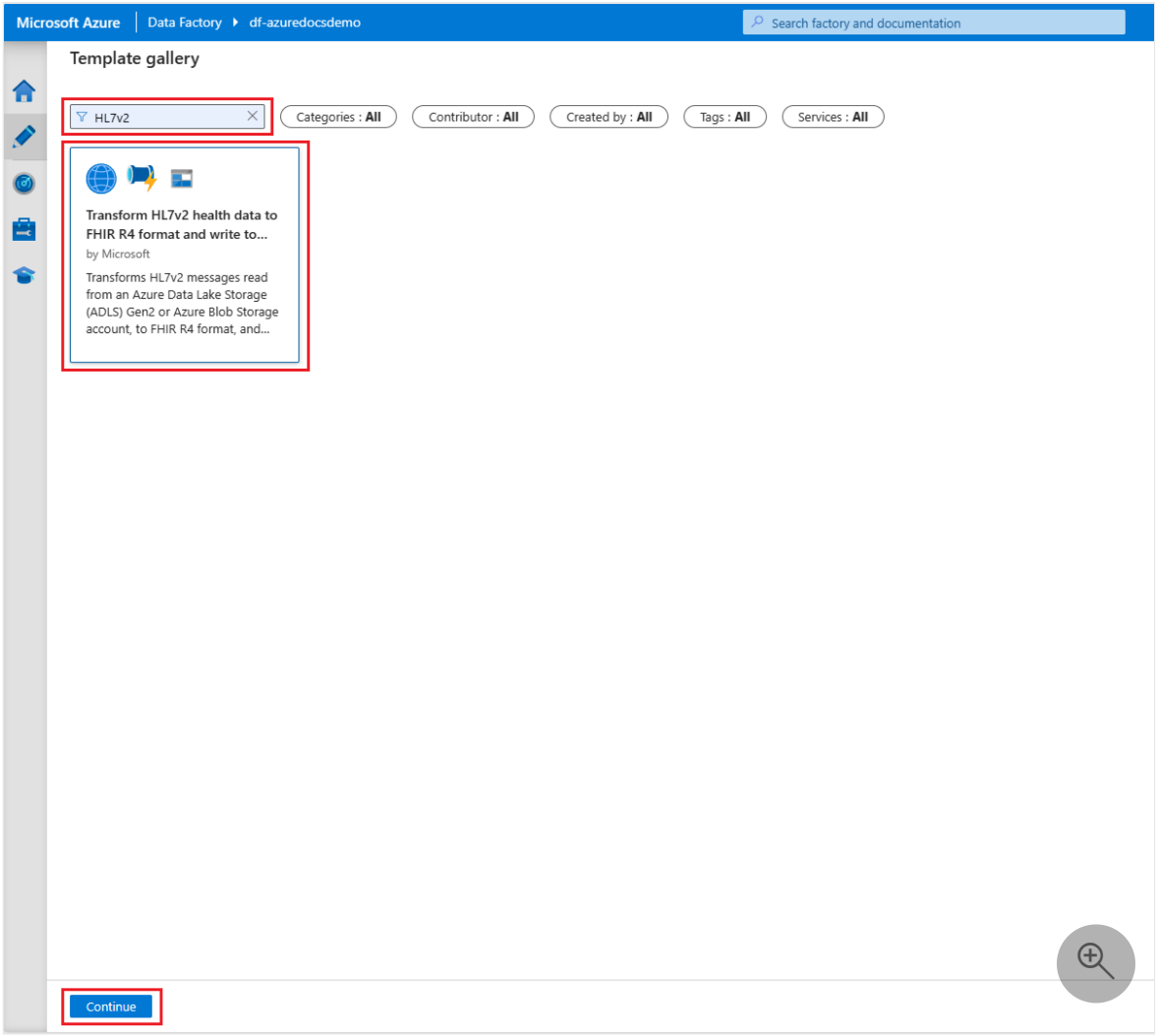
Create a pipeline

Azure Data Factory pipelines are a collection of activities that perform a task. This section details the creation of a pipeline that performs the task of transforming HL7v2 data to FHIR R4 bundles. Pipeline execution can be in an improvised fashion or regularly based on defined triggers.

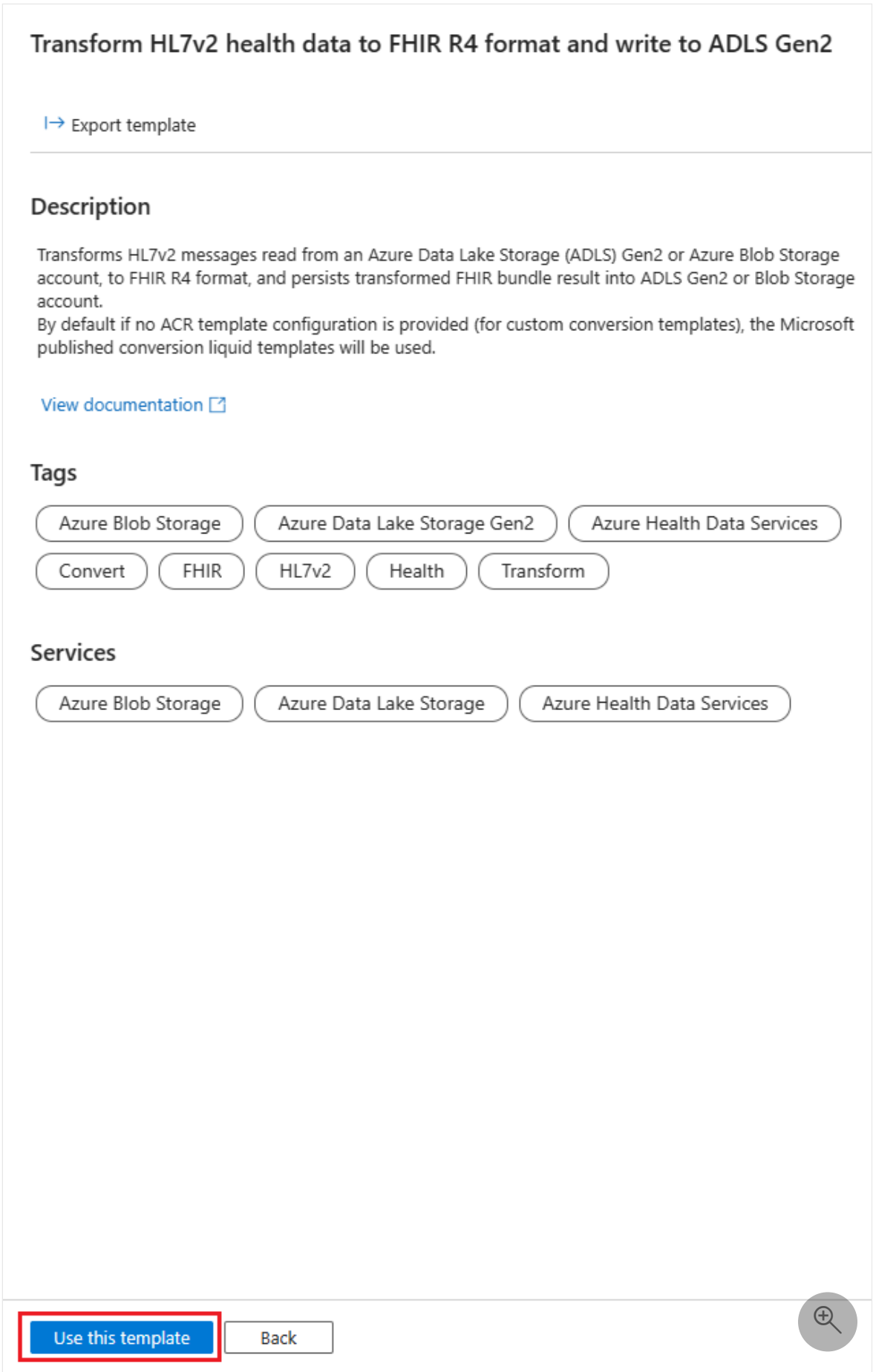
1. Select **Author** from the navigation menu. In the **Factory Resources** pane, select the + to add a new resource. Select **Pipeline** and then **Template gallery** from the menu.



2. In the Template gallery, search for **HL7v2**. Select the **Transform HL7v2 health data to FHIR R4 format and write to ADLS Gen2** tile and then select **Continue**.

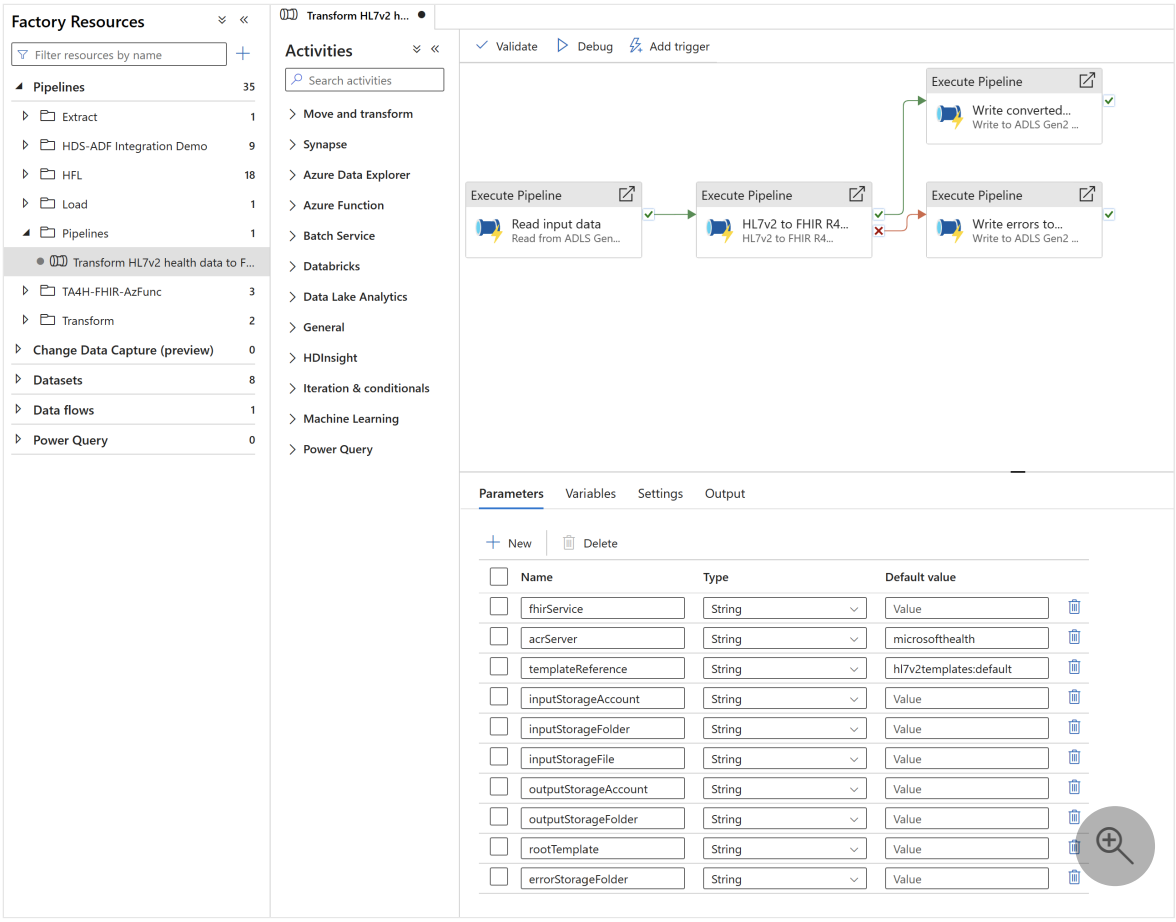


3. Select **Use this template** to create the new pipeline.



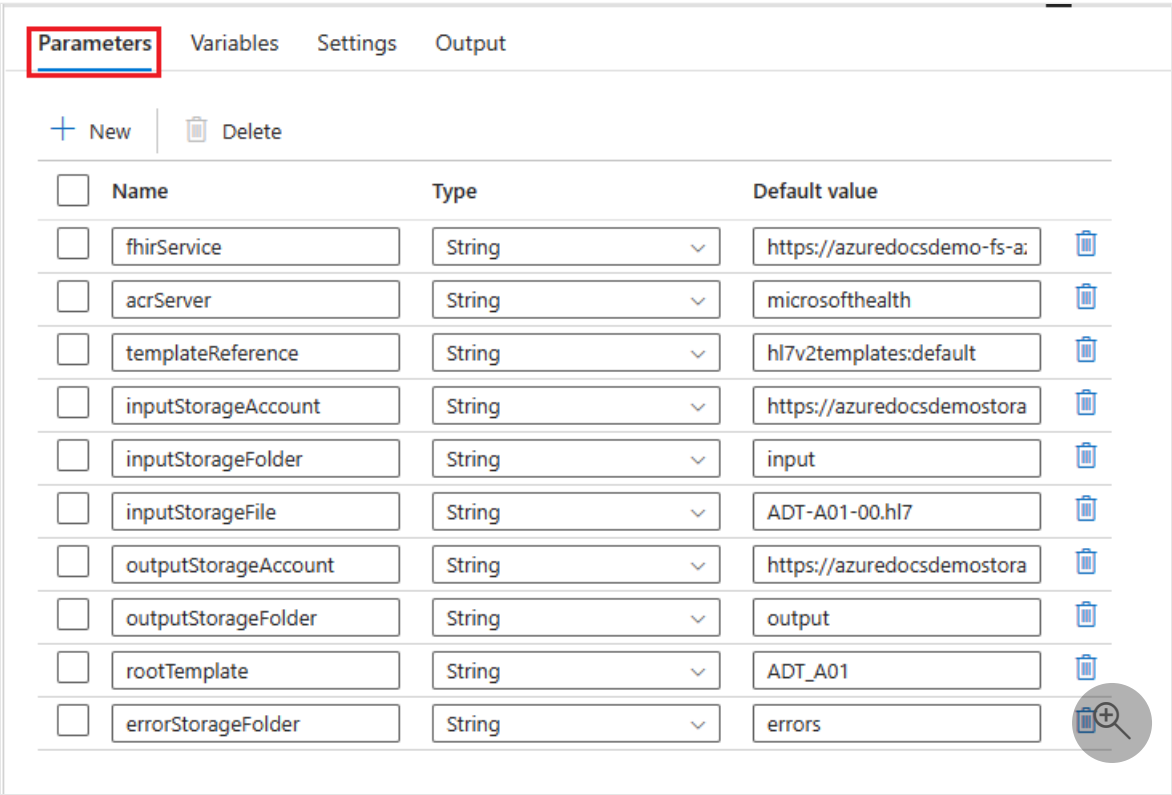
ADF imports the template, which is composed of an end-to-end main pipeline and a set of individual pipelines/activities. The main end-to-end pipeline for this

scenario is named **Transform HL7v2 health data to FHIR R4 format and write to ADLS Gen2** and can be accessed by selecting **Pipelines**. The main pipeline invokes the other individual pipelines/activities under the subcategories of **Extract**, **Load**, and **Transform**.



If needed, you can make any modifications to the pipelines/activities to fit your scenario (for example: if you don't intend to persist the results in a destination ADLS Gen2 storage account, you can modify the pipeline to remove the **Write converted result to ADLS Gen2** pipeline altogether).

4. Select the **Parameters** tab and provide values based your configuration/setup. Some of the values are based on the resources setup as part of the prerequisites.



- **fhirService** – Provide the URL of the FHIR service to target for the `$convert-data` operation. For example: `https://**myservice-fhir**.fhir.azurehealthcareapis.com/`
- **acrServer** – Provide the name of the ACR server to pull the Liquid templates to use for conversion. By default, option is set to `microsofthealth`, which contains the predefined template collection published by Microsoft. To use your own template collection, replace this value with your ACR instance that hosts your templates and is registered to your FHIR service.
- **templateReference** – Provide the reference to the image within the ACR that contains the Liquid templates to use for conversion. By default, this option is set to `hl7v2templates:default` to pull the latest published Liquid templates for HL7v2 conversion by Microsoft. To use your own template collection, replace this value with the reference to the image within your ACR that hosts your templates and is registered to your FHIR service.
- **inputStorageAccount** – The primary endpoint of the ADLS Gen2 storage account containing the input HL7v2 data to transform. For example: `https://**mystorage**.blob.core.windows.net`.
- **inputStorageFolder** – The container and folder path within the configured. For example: `**mycontainer**/**myHL7v2folder**`.

> **Note**
>
> This can be a static folder path or can be left blank here and dynamically configured when setting up storage account triggers for this pipeline

execution (refer to the section titled **Executing a pipeline**).

- **inputStorageFile** – The name of the file within the configured container.
- **inputStorageAccount** and **inputStorageFolder** that contains the HL7v2 data to transform. For example: `**myHL7v2file**.hl7`.

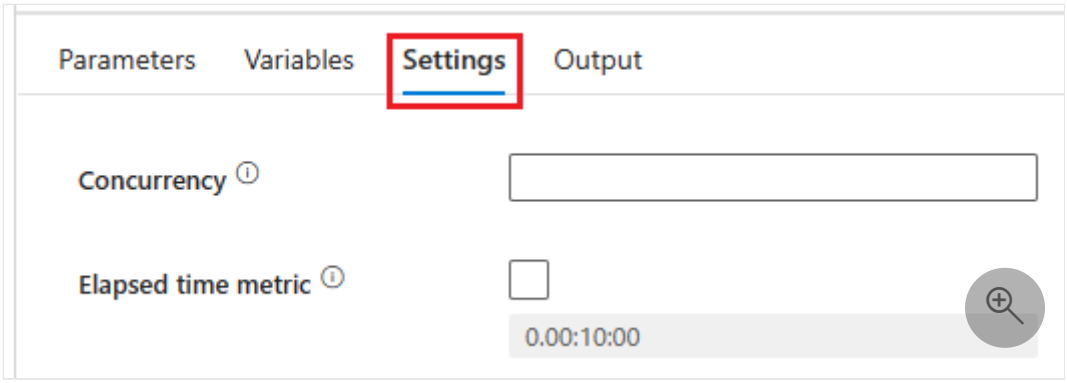
> ⓘ **Note**
>
> This can be a static folder path or can be left blank here and dynamically configured when setting up storage account triggers for this pipeline execution (refer to the section titled **Executing a pipeline**).

- **outputStorageAccount** – The primary endpoint of the ADLS Gen2 storage account to store the transformed FHIR bundle. For example: `https://**mystorage**.blob.core.windows.net`.
- **outputStorageFolder** – The container and folder path within the configured **outputStorageAccount** to which the transformed FHIR bundle JSON files are written to.
- **rootTemplate** – The root template to use while transforming the provided HL7v2 data. For example: ADT_A01, ADT_A02, ADT_A03, ADT_A04, ADT_A05, ADT_A08, ADT_A11, ADT_A13, ADT_A14, ADT_A15, ADT_A16, ADT_A25, ADT_A26, ADT_A27, ADT_A28, ADT_A29, ADT_A31, ADT_A47, ADT_A60, OML_O21, ORU_R01, ORM_O01, VXU_V04, SIU_S12, SIU_S13, SIU_S14, SIU_S15, SIU_S16, SIU_S17, SIU_S26, MDM_T01, MDM_T02.

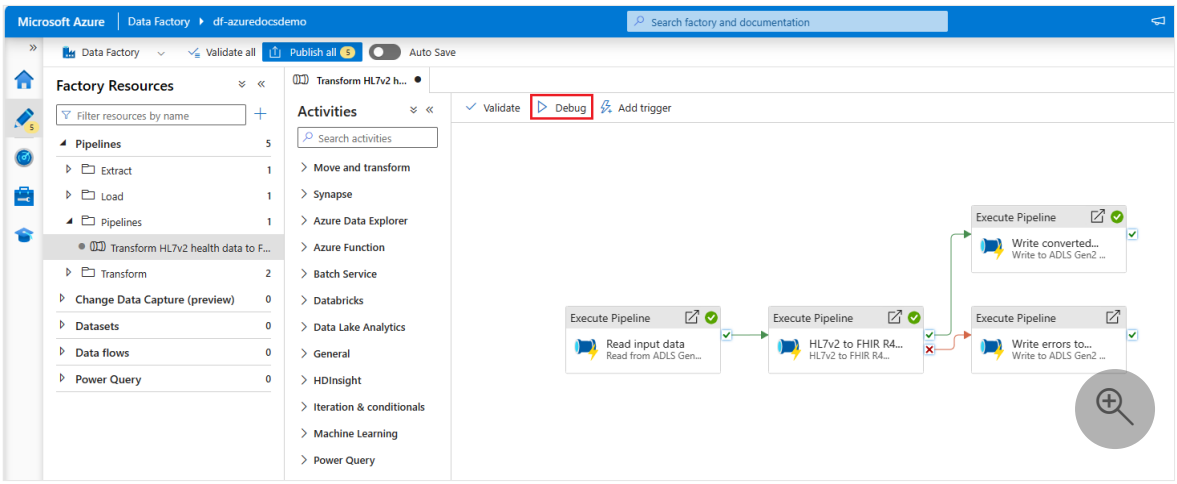
> ⓘ **Note**
>
> This can be a static folder path or can be left blank here and dynamically configured when setting up storage account triggers for this pipeline execution (refer to the section titled **Executing a pipeline**).

- **errorStorageFolder** - The container and folder path within the configured **outputStorageAccount** to which the errors encountered during execution are written to. For example: `**mycontainer**/**myerrorfolder**`.

5. You can configure more pipeline settings under the **Settings** tab based on your requirements.



6. You can also optionally debug your pipeline to verify the setup. Select **Debug**.



7. Verify your pipeline run parameters and select **OK**.

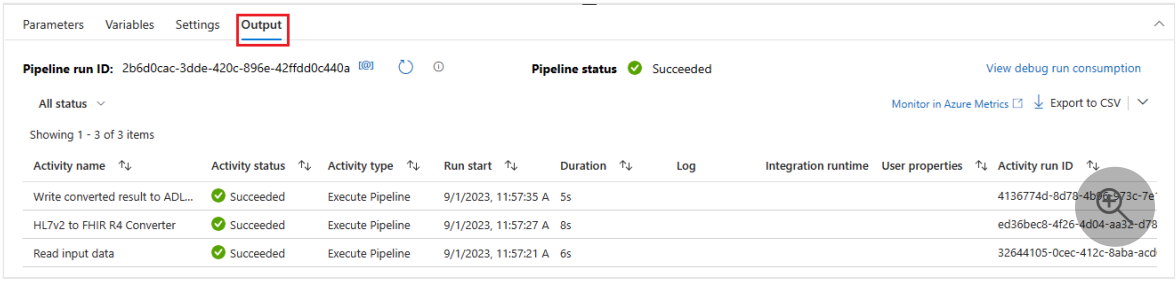
Pipeline run

Parameters

Name	Type	Value
fhirService	string	https://azuredocsdemo-fs-az...
acrServer	string	microsofthealth
templateReference	string	hl7v2templates:default
inputStorageAccount	string	https://azuredocsdemostora...
inputStorageFolder	string	input
inputStorageFile	string	ADT-A01-00.hl7
outputStorageAccount	string	https://azuredocsdemostora...
outputStorageFolder	string	output
rootTemplate	string	ADT_A01
errorStorageFolder	string	errors

OK Cancel

8. You can monitor the debug execution of the pipeline under the **Output** tab.



9. Once you're satisfied with your pipeline setup, select **Publish all**.

10. Select **Publish** to save your pipeline within your own ADF instance.

Publish all

You are about to publish all pending changes to the live environment. Learn more

Pending changes (5)

NAME	CHANGE	EXISTING
Pipelines		
Transform HL7v2 health d...	(New)	-
Read from ADLS Gen2 or ...	(New)	-
HL7v2 to FHIR R4 Converter	(New)	-
Write to ADLS Gen2 or Blo...	(New)	-
Generic FHIR R4 Converter	(New)	-

Publish Cancel

Executing a pipeline

You can execute (or run) a pipeline either manually or by using a trigger. There are different types of triggers that can be created to help automate your pipeline execution. For example:

- **Manual trigger**
- **Schedule trigger**
- **Tumbling window trigger**
- **Event-based trigger**

For more information on the different trigger types and how to configure them, see Pipeline execution and triggers in Azure Data Factory or Azure Synapse Analytics.

By setting triggers, you can simulate batch transformation of HL7v2 data. The pipeline executes automatically based on the configured trigger parameters without requiring individual invocation of the `$convert-data` operation for each input message.

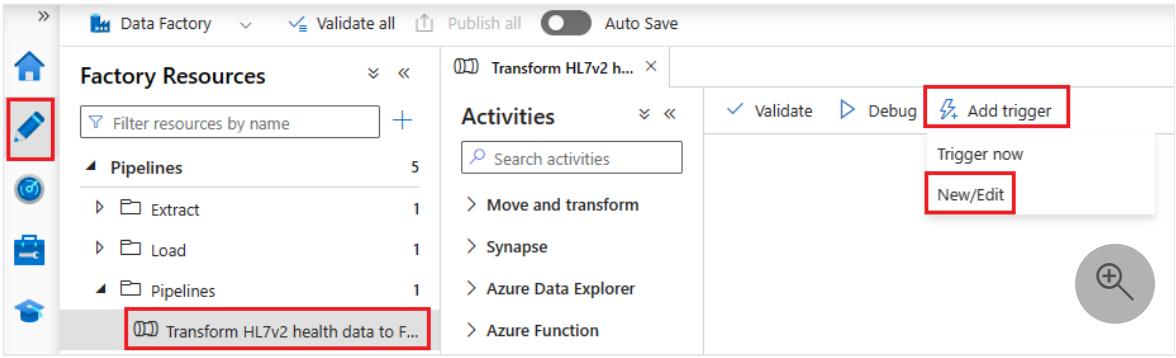
> ⓘ **Important**
>
> In a scenario with batch processing of HL7v2 messages, this template does not take sequencing into account, so post processing will be needed if sequencing is a requirement.

Create a new storage event trigger

In the following example, a storage event trigger is used. The storage event trigger automatically triggers the pipeline whenever a new HL7v2 data blob file to be processed is uploaded to the ADLS Gen2 storage account.

To configure the pipeline to automatically run whenever a new HL7v2 blob file in the source ADLS Gen2 storage account is available to transform, follow these steps:

1. Select **Author** from the navigation menu. Select the pipeline configured in the previous section and select **Add trigger** and **New/Edit** from the menu bar.



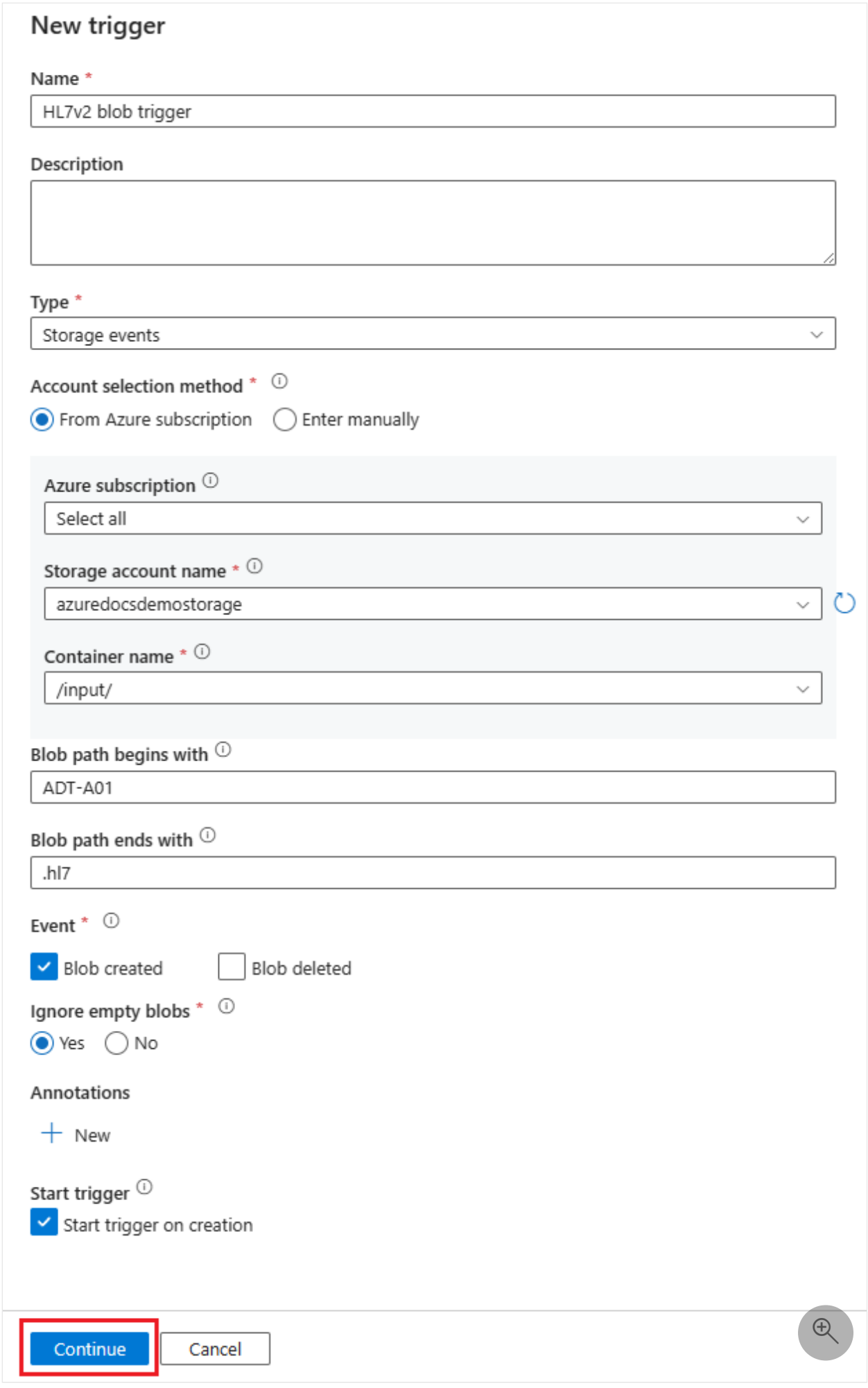
2. In the **Add triggers** panel, select the **Choose trigger** dropdown and then select **New**.

3. Enter a **Name** and **Description** for the trigger.

4. Select **Storage events** as the **Type**.

5. Configure the storage account details containing the source HL7v2 data to transform (for example: ADLS Gen2 storage account name, container name, blob path, etc.) to reference for the trigger.

6. Select **Blob created** as the **Event**.



7. Select **Continue** to see the **Data preview** for the configured settings.

8. Select **Continue** again at **Data preview** to continue configuring the trigger run parameters.

Configure trigger run parameters

Triggers not only define when to run a pipeline, they also include parameters that are passed to the pipeline execution. You can configure pipeline parameters dynamically using the trigger run parameters.

The storage event trigger captures the folder path and file name of the blob into the properties `@triggerBody().folderPath` and `@triggerBody().fileName`. To use the values of these properties in a pipeline, you must map the properties to pipeline parameters. After mapping the properties to parameters, you can access the values captured by the trigger through the `@pipeline().parameters.parameterName` expression throughout the pipeline. For more information, see Reference trigger metadata in pipeline runs.

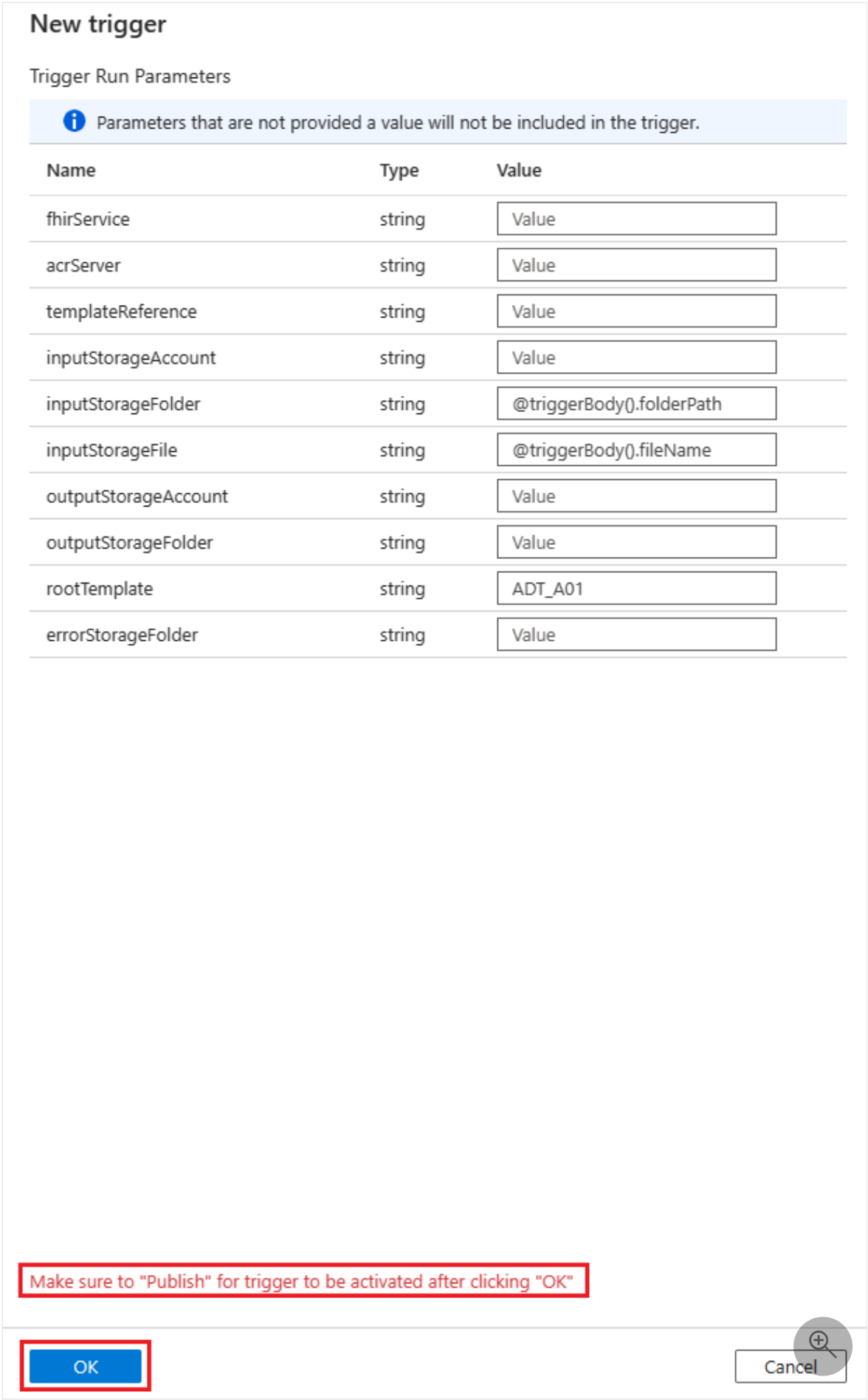
For the **Transform HL7v2 health data to FHIR R4 format and write to ADLS Gen2 template**, the storage event trigger properties can be used to configure certain pipeline parameters.

> ⓘ **Note**
>
> If no value is supplied during configuration, then the previously configured default value will be used for each parameter.

1. In the **New trigger** pane, within the **Trigger Run Parameters** options, use the following values:

    - For **inputStorageFolder** use `@triggerBody().folderPath`. This parameter provides the runtime value for this parameter based on the folder path associated with the event triggered (for example: folder path of the new HL7v2 blob created/updated in the storage account configured in the trigger).
    - For **inputStorageFile** use `@triggerBody().fileName`. This parameter provides the runtime value for this parameter based on the file associated with the event triggered (for example: file name of the new HL7v2 blob created/updated in the storage account configured in the trigger).
    - For **rootTemplate** specify the name of the template to be used for the pipeline executions associated with this trigger (for example: `ADT_A01`).

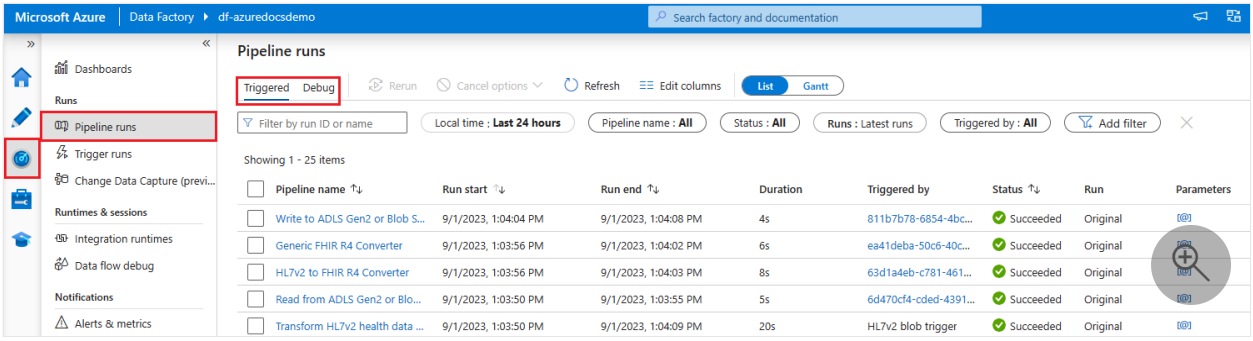
2. Select **OK** to create the new trigger. Be sure to select **Publish** on the menu bar to begin your trigger running on the defined schedule.



After the trigger is published, it can be triggered manually using the **Trigger now** option. If the start time was set for a value in the past, the pipeline starts immediately.

Monitoring pipeline runs

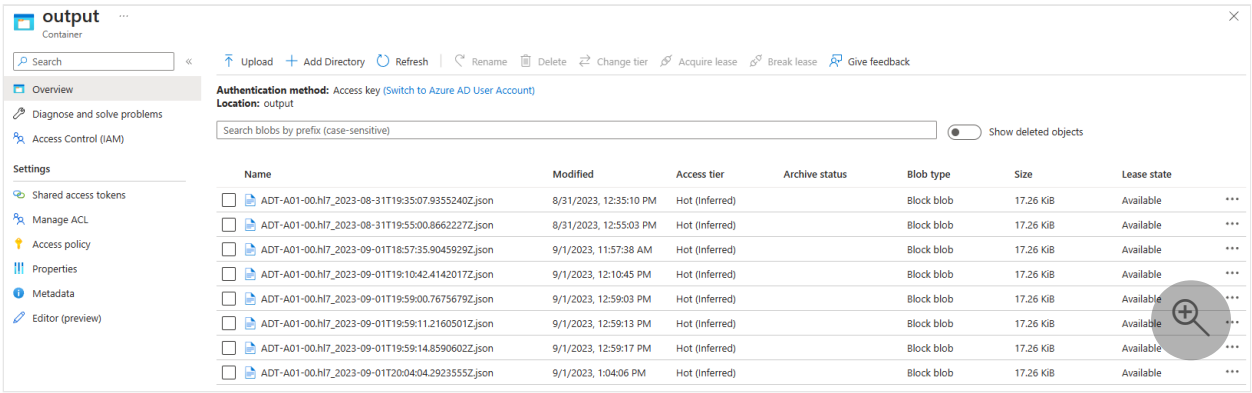
Trigger runs and their associated pipeline runs can be viewed in the **Monitor** tab. Here, users can browse when each pipeline ran, how long it took to execute, and potentially debug any problems that arose.



Pipeline execution results

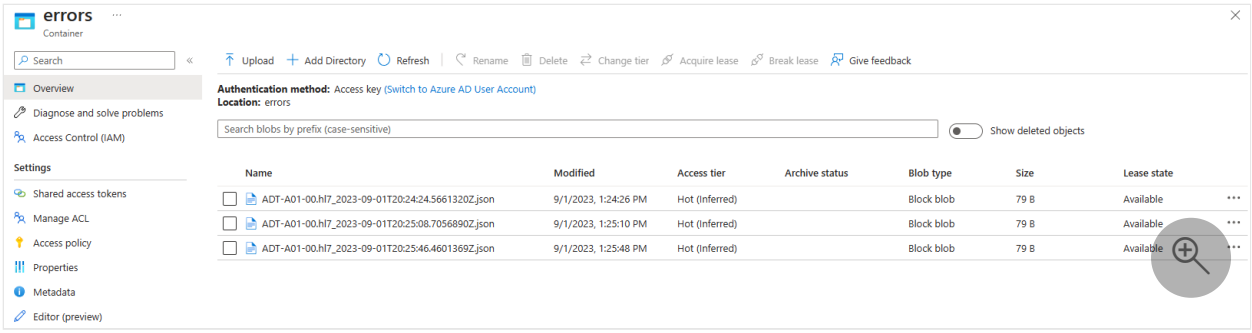
Transformed FHIR R4 results

Successful pipeline executions result in the transformed FHIR R4 bundles as JSON files in the configured destination ADLS Gen2 storage account and container.



Errors

Errors encountered during conversion, as part of the pipeline execution, result in error details captured as JSON file in the configured error destination ADLS Gen2 storage account and container. For information on how to troubleshoot `$convert-data`, see Troubleshoot $convert-data.



Next steps

In this article, you learned how to use Azure Data Factory templates to create a pipeline to transform HL7v2 data to FHIR R4 persisting the results within an Azure Data Lake Storage Gen2 account. You also learned how to configure a trigger to automate the pipeline execution based on incoming HL7v2 data to be transformed.

For an overview of `$convert-data`, see

Overview of $convert-data

To learn how to configure settings for `$convert-data` using the Azure portal, see

Configure settings for $convert-data using the Azure portal

To learn how to troubleshoot `$convert-data`, see

Troubleshoot $convert-data

Troubleshoot $convert-data

Article • 08/28/2023

In this article, learn how to troubleshoot `$convert-data`.

Performance

Two main factors come into play that determine how long a `$convert-data` operation call can take:

- The size of the message.
- The complexity of the template.

Any loops or iterations in the templates can have large impacts on performance. The `$convert-data` operation has a post processing step that is run after the template is applied. In particular, the deduping step can mask template issues that cause performance problems. Updating the template so duplicates aren't generated can greatly increase performance. For more information and details about the post processing step, see Post processing.

Post processing

The `$convert-data` operation applies post processing logic after the template is applied to the input. This post processing logic can result in the output looking different or unexpected errors compared to if you ran the default Liquid template directly. Post processing ensures the output is valid JSON and removes any duplicates based on the ID properties generated for resources in the template. To see the post processing logic in more detail, see the FHIR-Converter GitHub repository.

Message size

There currently isn't a hard limit on the size of the messages allowed for the `$convert-data` operation, however, for content with a request size greater than 10 MB, server 500 errors are possible. If you're receiving 500 server errors, ensure your requests are under 10 MB.

Default templates and customizations

Default template implementations for many common scenarios can be found on the FHIR-Converter GitHub repository. The default templates can be used as a guide and reference for customizing and creating your own templates. In addition to the default templates, the `$convert-data` operation supports several customer Liquid filters and tags that help simplify common scenarios.

Debugging and testing

In addition to testing templates on an instance of the service, a Visual Studio Code extension is available. The extension can be used to modify templates and test them with sample data payloads. There are also several existing test scenarios in the FHIR Converter GitHub repository that can be used as a reference.

Next steps

In this article, you learned how to troubleshoot `$convert-data`.

For an overview of `$convert-data`, see

Overview of $convert-data

To learn how to configure settings for `$convert-data` using the Azure portal, see

Configure settings for $convert-data using the Azure portal

To learn about the frequently asked questions (FAQs) for `$convert-data`, see

Frequently asked questions about $convert-data

Frequently asked questions about $convert-data

Article • 08/28/2023

> **Note**
>
> **Fast Healthcare Interoperability Resources (FHIR®)** is an open healthcare specification.

$convert-data: The basics

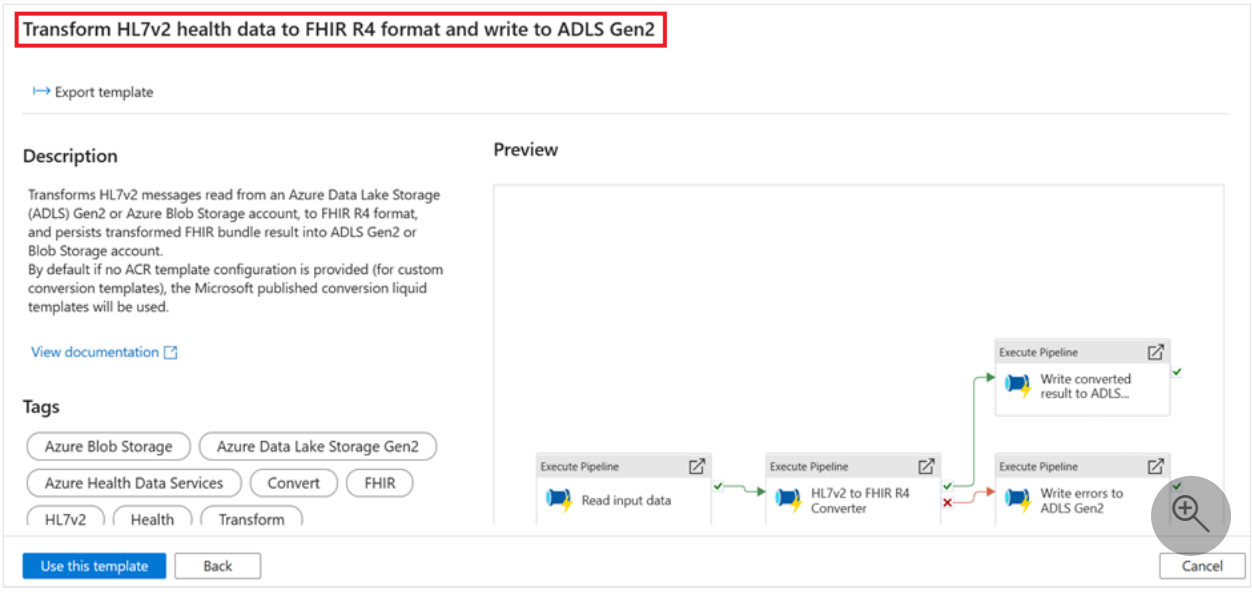
Does your service create/manage the entire ETL pipeline for me?

You can use the `$convert-data` endpoint as a component within an ETL (extract, transform, and load) pipeline for the conversion of health data from various formats (for example: HL7v2, CCDA, JSON, and FHIR STU3) into the FHIR format. You can create an ETL pipeline for a complete workflow as you convert your health data. We recommend that you use an ETL engine that's based on Azure Logic Apps or Azure Data Factory. As an example, a workflow might include: data ingestion, performing `$convert-data` operations, validation, data pre/post processing, data enrichment, data deduplication, and loading the data for persistence in the FHIR service.

However, the `$convert-data` operation itself isn't an ETL pipeline.

Where can I find an example of an ETL pipeline that I can reference?

There's an example published in the Azure Data Factory Template Gallery named **Transform HL7v2 health data to FHIR R4 format and write to ADLS Gen2**. This template transforms HL7v2 messages read from an Azure Data Lake Storage (ADLS) Gen2 or an Azure Blob Storage account into the FHIR R4 format. It then persists the transformed FHIR bundle JSON file into an ADLS Gen2 or a Blob Storage account. Once you're in the Azure Data Factory Template Gallery, you can search for the template.



> **ⓘ Important**
>
> The purpose of this template is to help you get started with an ETL pipeline. Any steps in this pipeline can be removed, added, edited, or customized to fit your needs.
>
> In a scenario with batch processing of HL7v2 messages, this template does not take sequencing into account. Post processing will be needed if sequencing is a requirement.

How can I persist the converted data into the FHIR service using Postman?

You can use the FHIR service's APIs to persist the converted data into the FHIR service by using `POST {{fhirUrl}}/{{FHIR resource type}}` with the request body containing the FHIR resource to be persisted in JSON format.

For more information about using Postman with the FHIR service, see Access the Azure Health Data Services FHIR service using Postman.

Is there a difference in the experience of the $convert-data endpoint in Azure API for FHIR versus in the Azure Health Data Services?

The experience and core `$convert-data` operation functionality is similar for both Azure API for FHIR and the Azure Health Data Services FHIR service. The only difference exists

in the setup for the Azure API for FHIR version of the `$convert-data` operation, which requires assigning permissions to the right resources. For more information about `$convert-data` operation versions, see:

- Azure API for FHIR: Data conversion for Azure API for FHIR
- Azure Health Data Services: Overview of $convert-data

I'm not familiar with Liquid templates. Where do I start?

Liquid is a template language/engine that allows displaying data in a template. Liquid has constructs such as output, logic, loops and deals with variables. Liquid files are a mixture of HTML and Liquid code, and have the `.liquid` file extension. The open source FHIR Converter comes with a few ready to use Liquid templates and custom filters for the supported conversion formats to help you get started.

The conversion succeeded, does this mean I have a valid FHIR bundle?

The outcome of FHIR conversion is a FHIR bundle as a batch.

- The FHIR bundle should align with the expectations of the FHIR R4 specification - Bundle - FHIR v4.0.1.
- If you're trying to validate against a specific profile, you need to do some post processing by utilizing the FHIR $validate operation.

Can I customize a default Liquid template?

Yes. You can use the FHIR Converter Visual Studio Code extension to customize templates according to your specific requirements. The extension provides an interactive editing experience and makes it easy to download Microsoft-published templates and sample data. The FHIR Converter extension for Visual Studio Code is available for HL7v2, C-CDA, and JSON Liquid templates. FHIR STU3 to FHIR R4 Liquid templates are currently not supported. For more information about setting up custom templates, see Configure settings for $convert-data using the Azure portal.

Once I customize a template, is it possible to reference and store various versions of the template?

Yes. It's possible to store and reference custom templates. See Configure settings for $convert-data using the Azure portal for instructions to reference and store various versions of custom templates.

If I need support troubleshooting issues, where can I go?

Depending on the version of `$convert-data` you're using, you can:

- Use the troubleshooting guide for the Azure Health Data Service FHIR service version of the `$convert-data` operation.
- Open a support request for the Azure Health Data Service FHIR service version of the `$convert-data` operation.
- Leave a comment on the GitHub repository for the open source version of the FHIR Converter.

Next steps

In this article, you learned about the frequently asked questions (FAQs) about the `$convert-data` operation and endpoint for converting health data into FHIR by using the FHIR service in Azure Health Data Services.

For an overview of `$convert-data`, see

Overview of $convert-data

To learn how to configure settings for `$convert-data` using the Azure portal, see

Configure settings for $convert-data using the Azure portal

To learn how to troubleshoot `$convert-data`, see

Troubleshoot $convert-data

Configure export settings and set up a storage account

Article • 07/28/2023

The FHIR service supports the `$export` operation specified by HL7 for exporting FHIR data from a FHIR server. In the FHIR service implementation, calling the `$export` endpoint causes the FHIR service to export data into a pre-configured Azure storage account.

Ensure you are granted with application role - 'FHIR Data exporter role' prior to configuring export. To understand more on application roles, see Authentication and Authorization for FHIR service.

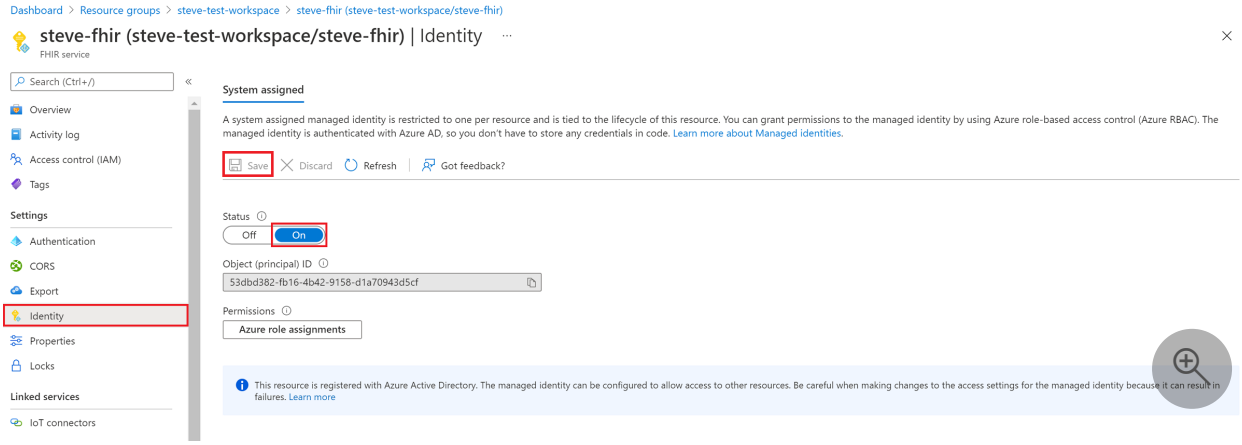
Three steps in setting up the `$export` operation for the FHIR service-

- Enable a managed identity for the FHIR service.
- Configure a new or existing Azure Data Lake Storage Gen2 (ADLS Gen2) account and give permission for the FHIR service to access the account.
- Set the ADLS Gen2 account as the export destination for the FHIR service.

Enable managed identity for the FHIR service

The first step in configuring your environment for FHIR data export is to enable a system-wide managed identity for the FHIR service. This managed identity is used to authenticate the FHIR service to allow access to the ADLS Gen2 account during an `$export` operation. For more information about managed identities in Azure, see About managed identities for Azure resources.

In this step, browse to your FHIR service in the Azure portal and select the **Identity** blade. Set the **Status** option to **On**, and then click **Save**. When the **Yes** and **No** buttons display, select **Yes** to enable the managed identity for the FHIR service. Once the system identity has been enabled, you'll see an **Object (principal) ID** value for your FHIR service.

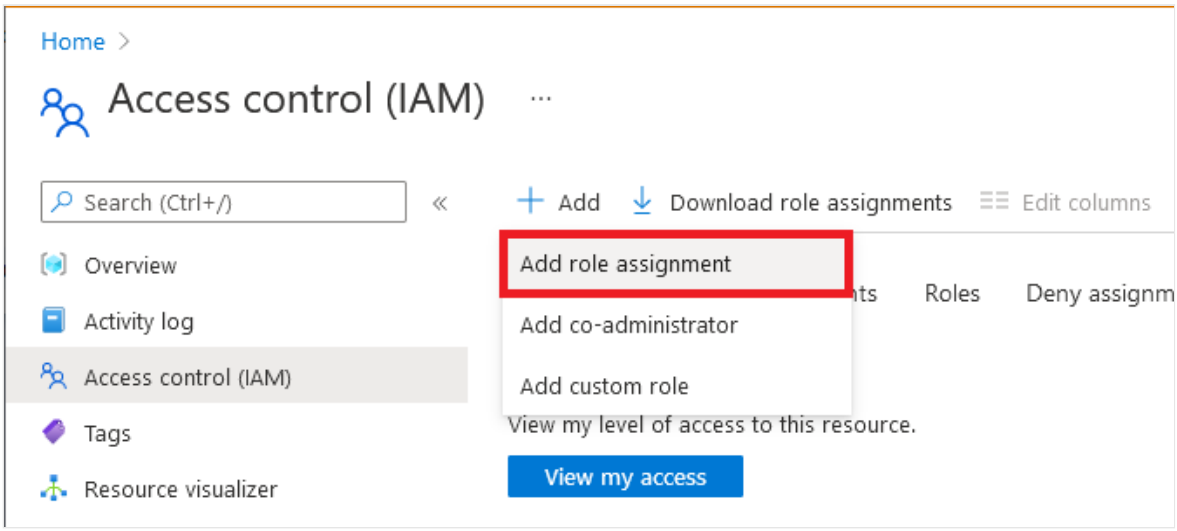


Give permission in the storage account for FHIR service access

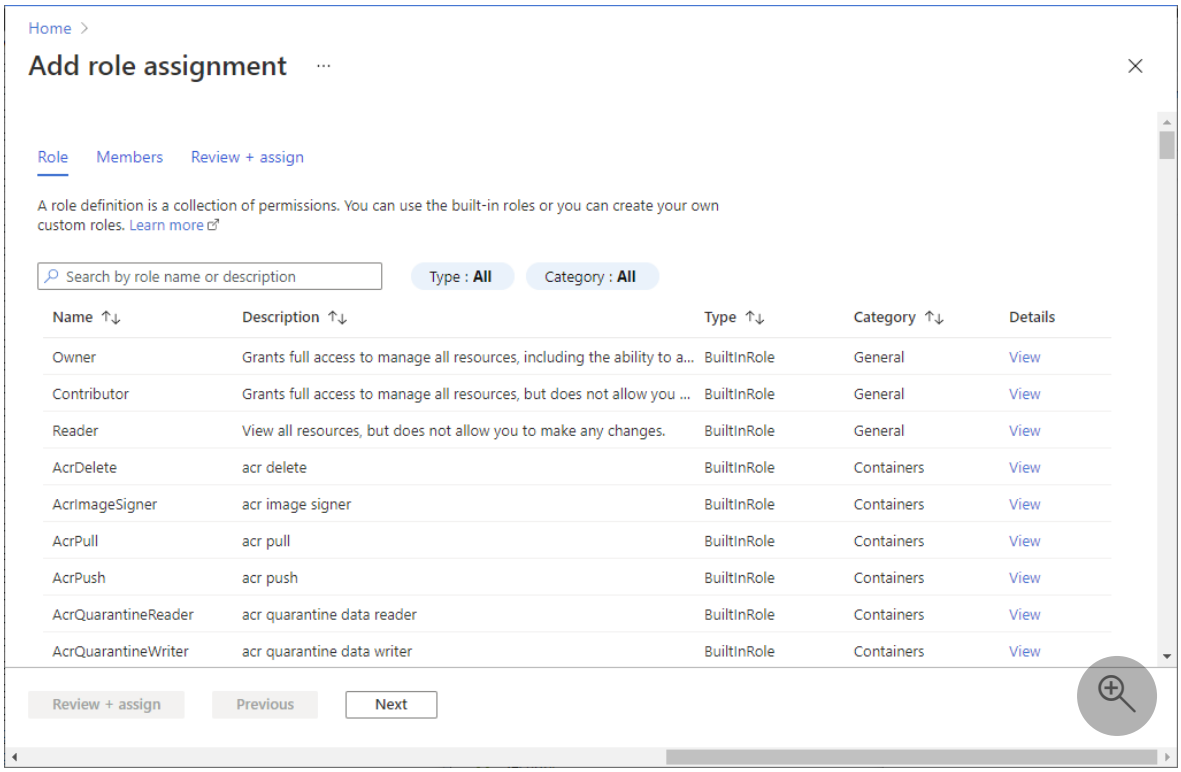
1. Go to your ADLS Gen2 account in the Azure portal. If you don't already have an ADSL Gen2 account deployed, follow these instructions for creating an Azure storage account and upgrading to ADLS Gen2. Make sure to enable the hierarchical namespace option in the **Advanced** tab to create an ADLS Gen2 account.

2. In your ADLS Gen2 account, select **Access control (IAM)**.

3. Select **Add** > **Add role assignment**. If the **Add role assignment** option is grayed out, ask your Azure administrator for help with this step.



4. On the **Role** tab, select the Storage Blob Data Contributor role.



5. On the **Members** tab, select **Managed identity**, and then click **Select members**.

6. Select your Azure subscription.

7. Select **System-assigned managed identity**, and then select the managed identity that you enabled earlier for your FHIR service.

8. On the **Review + assign** tab, click **Review + assign** to assign the **Storage Blob Data Contributor** role to your FHIR service.

For more information about assigning roles in the Azure portal, see Azure built-in roles.

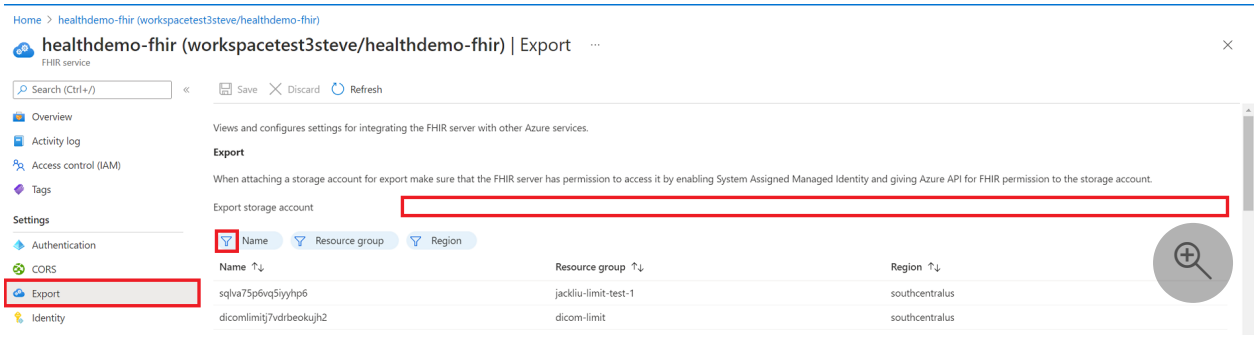
Now you're ready to configure the FHIR service by setting the ADLS Gen2 account as the default storage account for export.

Specify the storage account for FHIR service export

The final step is to specify the ADLS Gen2 account that the FHIR service uses when exporting data.

> **Note**
>
> In the storage account, if you haven't assigned the **Storage Blob Data Contributor** role to the FHIR service, the `$export` operation will fail.

1. Go to your FHIR service settings.
2. Select the **Export** blade.
3. Select the name of the storage account from the list. If you need to search for your storage account, use the **Name**, **Resource group**, or **Region** filters.



After you've completed this final configuration step, you're ready to export data from the FHIR service. See How to export FHIR data for details on performing `$export` operations with the FHIR service.

> **Note**
>
> Only storage accounts in the same subscription as the FHIR service are allowed to be registered as the destination for `$export` operations.

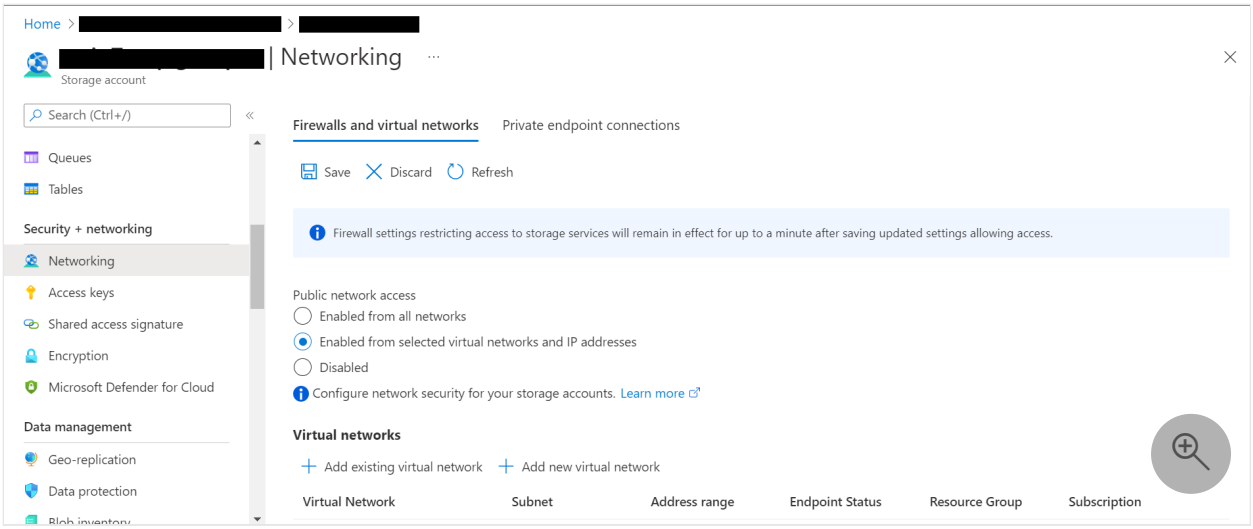
Securing the FHIR service `$export` operation

For securely exporting from the FHIR service to an ADLS Gen2 account, there are two main options:

- Allowing the FHIR service to access the storage account as a Microsoft Trusted Service.
- Allowing specific IP addresses associated with the FHIR service to access the storage account. This option permits two different configurations depending on whether or not the storage account is in the same Azure region as the FHIR service.

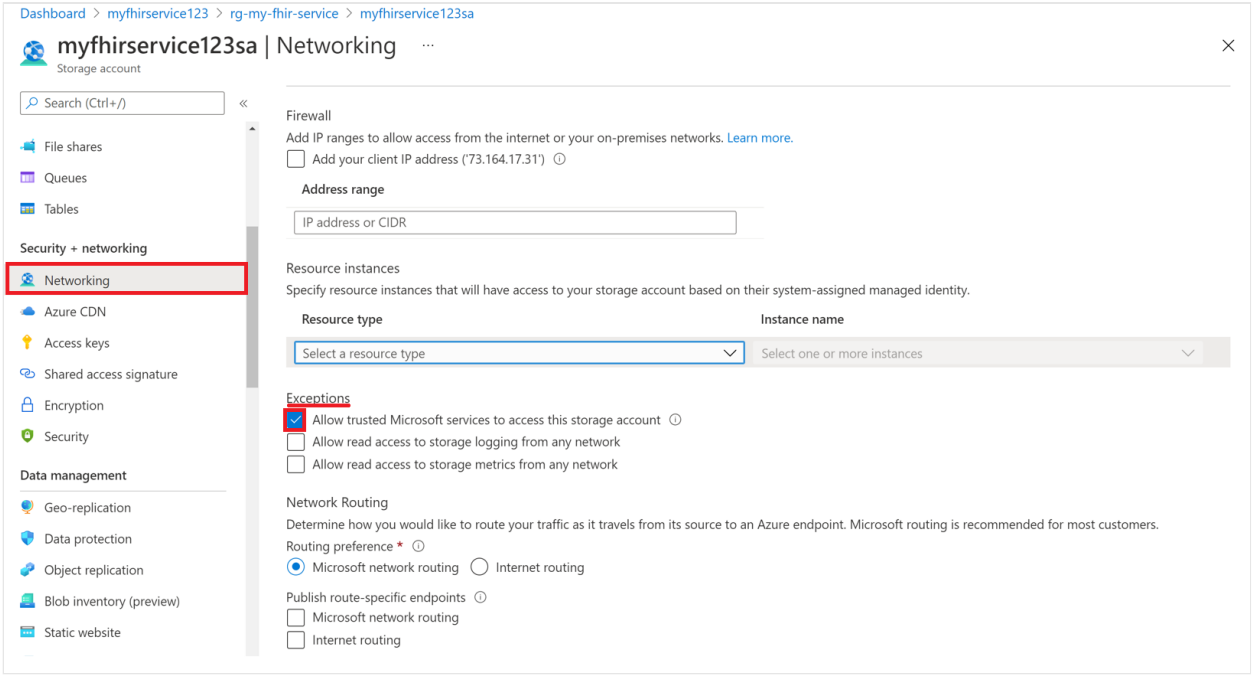
Allowing FHIR service as a Microsoft Trusted Service

Go to your ADLS Gen2 account in the Azure portal and select the **Networking** blade. Select **Enabled from selected virtual networks and IP addresses** under the **Firewalls and virtual networks** tab.



Select **Microsoft.HealthcareApis/workspaces** from the **Resource type** dropdown list and then select your workspace from the **Instance name** dropdown list.

Under the **Exceptions** section, select the box **Allow Azure services on the trusted services list to access this storage account**. Make sure to click **Save** to retain the settings.



Next, run the following PowerShell command to install the `Az.Storage` PowerShell module in your local environment. This allows you to configure your Azure storage account(s) using PowerShell.

PowerShell

```powershell
Install-Module Az.Storage -Repository PsGallery -AllowClobber -Force
```

Now, use the PowerShell command below to set the selected FHIR service instance as a trusted resource for the storage account. Make sure that all listed parameters are defined in your PowerShell environment.

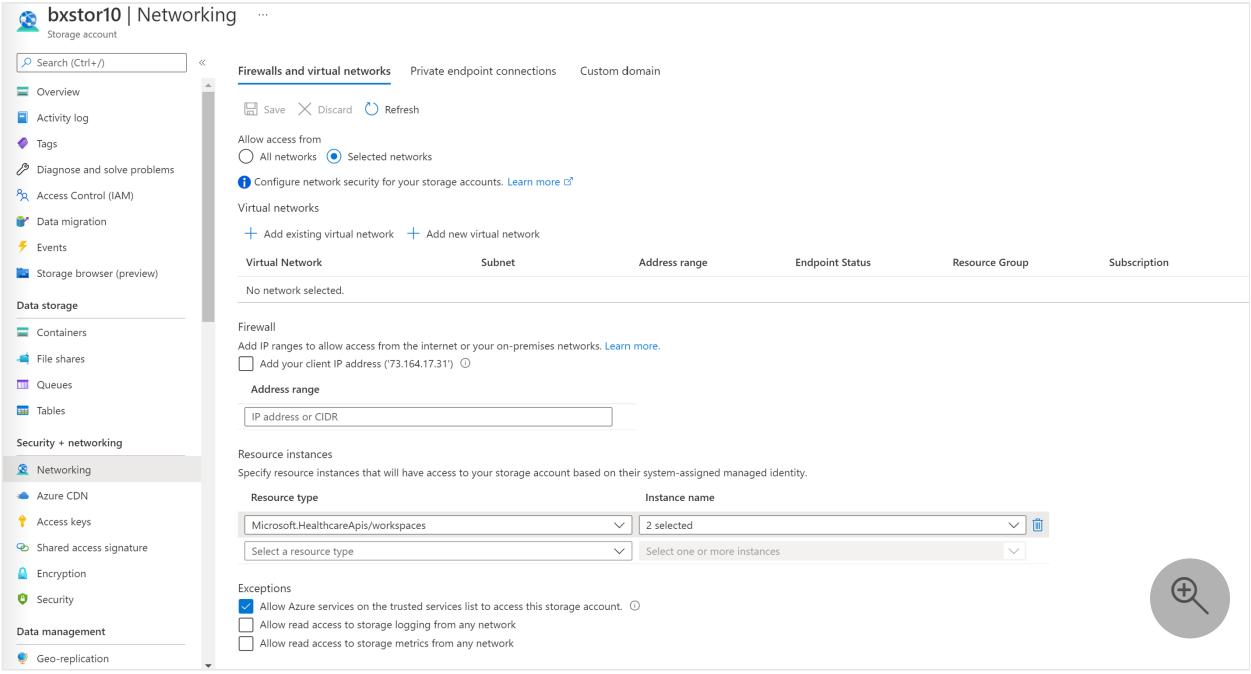
You'll need to run the `Add-AzStorageAccountNetworkRule` command as an administrator in your local environment. For more information, see Configure Azure Storage firewalls and virtual networks.

PowerShell

```
$subscription="xxx"
$tenantId = "xxx"
$resourceGroupName = "xxx"
$storageaccountName = "xxx"
$workspacename="xxx"
$fhirname="xxx"
$resourceId =
"/subscriptions/$subscription/resourceGroups/$resourceGroupName/providers/Microsoft.HealthcareApis/workspaces/$workspacename/fhirservices/$fhirname"

Add-AzStorageAccountNetworkRule -ResourceGroupName $resourceGroupName -Name
$storageaccountName -TenantId $tenantId -ResourceId $resourceId
```

After running this command, in the **Firewall** section under **Resource instances** you will see **2 selected** in the **Instance name** dropdown list. These are the names of the workspace instance and FHIR service instance that you registered as Microsoft Trusted Resources.



You're now ready to securely export FHIR data to the storage account.

The storage account is on selected networks and isn't publicly accessible. To securely access the files, you can enable private endpoints for the storage account.

Allowing specific IP addresses from other Azure regions to access the Azure storage account

In the Azure portal, go to the ADLS Gen2 account and select the **Networking** blade.

Select **Enabled from selected virtual networks and IP addresses**. 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 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

Azure Region	Public IP Address
West Central US	52.150.156.44
West Europe	20.61.98.66
West US 2	40.64.135.77

Allowing specific IP addresses to access the Azure storage account in the same region

The configuration process for IP addresses in the same region is just like above except a specific IP address range in Classless Inter-Domain Routing (CIDR) format is used instead (i.e., 100.64.0.0/10). The reason why the IP address range (100.64.0.0 – 100.127.255.255) must be specified is because an IP address for the FHIR service will be allocated each time an `$export` request is made.

> **Note**
>
> It is possible that a private IP address within the range of 10.0.2.0/24 may be used, but there is no guarantee that the `$export` operation will succeed in such a case. You can retry if the `$export` request fails, but until an IP address within the range of 100.64.0.0/10 is used, the request will not succeed. This network behavior for IP address ranges is by design. The alternative is to configure the storage account in a different region.

Next steps

In this article, you learned about the three steps in configuring your environment to allow export of data from your FHIR service to an Azure storage account. For more information about Bulk Export capabilities in the FHIR service, see

How to export FHIR data

Export your FHIR data

Article • 12/13/2022

By using the bulk `$export` operation in the FHIR service, you can export data as described in the HL7 FHIR Bulk Data Access specification.

Before you attempt to use `$export`, make sure that your FHIR service is configured to connect with an Azure Data Lake Storage Gen2 account. To configure export settings and create a Data Lake Storage Gen2 account, refer to Configure settings for export.

Call the `$export` endpoint

After you set up the FHIR service to connect with a Data Lake Storage Gen2 account, you can call the `$export` endpoint, and the FHIR service will export data into an Azure Blob Storage container inside the storage account. The following example request exports all resources into a container, which is specified by name (`{{containerName}}`). Note that you must create the container in the Data Lake Storage Gen2 account beforehand if you want to specify the `{{containerName}}` in the request.

```
GET {{fhirurl}}/$export?_container={{containerName}}
```

If you don't specify a container name in the request (for example, by calling `GET {{fhirurl}}/$export`), a new container with an autogenerated name will be created for the exported data.

For general information about the FHIR `$export` API spec, see the HL7 FHIR Export Request Flow documentation.

The FHIR service supports `$export` at the following levels:

- System: `GET {{fhirurl}}/$export`
- Patient: `GET {{fhirurl}}/Patient/$export`
- Group of patients*: `GET {{fhirurl}}/Group/[ID]/$export`
  *The FHIR service exports all referenced resources but doesn't export the characteristics of the group resource itself.

Data is exported in multiple files. Each file contains resources of only one type. The number of resources in an individual file will be limited. The maximum number of

resources is based on system performance. It is currently set to 50,000, but can change. The result is that you might get multiple files for a resource type. The file names will follow the format `<resourceName>-<number>-<number>.ndjson`. The order of the files is not guarantied to correspond to any ordering of the resources in the database.

> **Note**
>
> `Patient/$export` and `Group/[ID]/$export` can export duplicate resources if a resource is in multiple groups or in a compartment of more than one resource.

In addition to checking the presence of exported files in your storage account, you can check your `$export` operation status through the URL in the `Content-Location` header that's returned in the FHIR service response. For more information, see the Bulk Data Status Request documentation from HL7.

Export your FHIR data to Data Lake Storage Gen2

Currently, the FHIR service supports `$export` to Data Lake Storage Gen2 accounts, with the following limitations:

- Data Lake Storage Gen2 provides hierarchical namespaces, yet there isn't a way to target `$export` operations to a specific subdirectory within a container. The FHIR service can specify only the destination container for the export, where a new folder for each `$export` operation is created.
- After an `$export` operation is complete and all data has been written inside a folder, the FHIR service doesn't export anything to that folder again, because subsequent exports to the same container will be inside a newly created folder.

To export data to a storage account behind a firewall, see Configure settings for export.

Settings and parameters

Headers

Two required header parameters must be set for `$export` jobs. The values are set according to the current HL7 $export specification.

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

Query parameters

The FHIR service supports the following query parameters for filtering exported data. All these parameters are optional.

Query parameter	Defined by the FHIR specification?	Description
`_outputFormat`	Yes	Currently supports three values to align to the FHIR specification: `application/fhir+ndjson`, `application/ndjson`, or just `ndjson`. All export jobs will return `.ndjson` files and the passed value has no effect on code behavior.
`_since`	Yes	Allows you to export only resources that have been modified since the specified time.
`_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 limit the results.
`_container`	No	Specifies the name of the container in the configured storage account where the data should be exported. If a container is specified, the data will be exported into a folder in that container. If the container isn't specified, the data will be exported to a new container with an autogenerated name.
`_till`	No	Allows you to export resources that have been modified till the specified time. This parameter is applicable only with System-Level export. In this case, if historical versions have not been disabled or purged, export guarantees true snapshot view, or, in other words, enables time travel.

> ⓘ **Note**
>
> Only storage accounts in the same subscription as the FHIR service are allowed to be registered as the destination for `$export` operations.

Troubleshoot

The following information can help you resolve problems with exporting FHIR data.

Jobs stuck in a bad state

In some situations, there's a potential for a job to be stuck in a bad state while the FHIR service is attempting to export data. This can occur especially if the Data Lake Storage Gen2 account permissions haven't been set up correctly.

One way to check the status of your `$export` operation is to go to your storage account's *storage browser* and see whether any `.ndjson` files are present in the export container. If the files aren't present and no other `$export` jobs are running, it's possible that the current job is stuck in a bad state. In this case, you can cancel the `$export` job by calling the FHIR service API with a `DELETE` request. Later, you can requeue the `$export` job and try again.

For more information about canceling an `$export` operation, see the Bulk Data Delete Request documentation from HL7.

> **Note**
>
> In the FHIR service, the default time for an `$export` operation to idle in a bad state is 10 minutes before the service stops the operation and moves to a new job.

Next steps

In this article, you've learned about exporting FHIR resources by using the `$export` operation. For information about how to set up and use additional options for export, see:

Export de-identified data

Copy data from the FHIR service to Azure Synapse Analytics

Exporting de-identified data

Article • 09/26/2022

> **Note**
>
> Results when using the FHIR service's de-identified export will vary based on the nature of the data being exported and what de-id functions are in use. Microsoft is unable to evaluate de-identified export outputs or determine the acceptability for customers' use cases and compliance needs. The FHIR service's de-identified export is not guaranteed to meet any specific legal, regulatory, or compliance requirements.

The FHIR service is able to de-identify data on export when running an `$export` operation. For de-identified export, the FHIR service uses the anonymization engine from the FHIR tools for anonymization (OSS) project on GitHub. There is a sample config file to help you get started redacting/transforming FHIR data fields that contain personally identifying information.

Configuration file

The anonymization engine comes with a sample configuration file to help you get started with HIPAA Safe Harbor Method de-id requirements. The configuration file is a JSON file with four properties: `fhirVersion`, `processingErrors`, `fhirPathRules`, `parameters`.

- `fhirVersion` specifies the FHIR version for the anonymization engine.
- `processingErrors` specifies what action to take for any processing errors that may arise during the anonymization. You can *raise* or *keep* the exceptions based on your needs.
- `fhirPathRules` specifies which anonymization method to use. The rules are executed in the order they appear in the configuration file.
- `parameters` sets additional controls for the anonymization behavior specified in *fhirPathRules*.

Here's a sample configuration file for FHIR R4:

JSON

```
{
  "fhirVersion": "R4",
  "processingError":"raise",
  "fhirPathRules": [
    {"path": "nodesByType('Extension')", "method": "redact"},
    {"path": "Organization.identifier", "method": "keep"},
    {"path": "nodesByType('Address').country", "method": "keep"},
    {"path": "Resource.id", "method": "cryptoHash"},
    {"path": "nodesByType('Reference').reference", "method": "cryptoHash"},
    {"path": "Group.name", "method": "redact"}
  ],
  "parameters": {
    "dateShiftKey": "",
    "cryptoHashKey": "",
    "encryptKey": "",
    "enablePartialAgesForRedact": true
  }
}
```

For detailed information on the settings within the configuration file, visit here.

Manage Configuration File in storage account

You will need to create a container for the de-identified export in your ADLS Gen2 account and specify the `<<container_name>>` in the API request as shown above. Additionally, you will need to place the JSON config file with the

anonymization rules inside the container and specify the `<<config file name>>` in the API request (see below).

> ⓘ Note
>
> It is common practice to name the container `anonymization`. The JSON file within the container is often named `anonymizationConfig.json`.

Manage Configuration File in ACR

It's recommended that you host the export configuration files on Azure Container Registry(ACR). It takes the following steps similar as hosting templates in ACR for $convert-data.

1. Push the configuration files to your Azure Container Registry.
2. Enable Managed Identity on your FHIR service instance.
3. Provide access of the ACR to the FHIR service Managed Identity.
4. Register the ACR servers in the FHIR service. You can use the portal to open "Artifacts" blade under "Transform and transfer data" section to add the ACR server.
5. Optionally configure ACR firewall for secure access.

Using the `$export` endpoint for de-identifying data

```
https://<<FHIR service base URL>>/$export?_container=<<container_name>>&_anonymizationConfigCollectionReference=<<ACR image reference>>&_anonymizationConfig=<<config file name>>&_anonymizationConfigEtag=<<ETag on storage>>
```

> ⓘ Note
>
> Right now the FHIR service only supports de-identified export at the system level (`$export`).

Query parameter	Example	Optionality	Description
_container	exportContainer	Required	Name of container within the configured storage account where the data will be exported.
_anonymizationConfigCollectionReference	"myacr.azurecr.io/deidconfigs:default"	Optional	Reference to an OCI image on ACR containing de-id configuration files for de-id export (such as stu3-config.json, r4-config.json). The ACR server of the image should be registered within the FHIR service. (Format: `<RegistryServer>/<imageName>@<imageDigest>`, `<RegistryServer>/<imageName>:<imageTag>`)
_anonymizationConfig	`anonymizationConfig.json`	Required	Name of the configuration file. See the configuration file format here. If *_anonymizationConfigCollectionReference* is provided, we will search and use this file from the specified image. Otherwise, we will search and use this file inside a container named **anonymization** within the configured ADLS Gen2 account.

Query parameter	Example	Optionality	Description
_anonymizationConfigEtag	"0x8D8494A069489EC"	Optional	Etag of the configuration file which can be obtained from the blob property in Azure Storage Explorer. Specify this parameter only if the configuration file is stored in Azure storage account. If you use ACR to host the configuration file, you should not include this parameter.

Important

Both the raw export and de-identified export operations write to the same Azure storage account specified in the export configuration for the FHIR service. If you have need for multiple de-identification configurations, it is recommended that you create a different container for each configuration and manage user access at the container level.

Next steps

In this article, you've learned how to set up and use the de-identified export feature in the FHIR service. For more information about how to export FHIR data, see

Export data

Configure bulk-import settings

Article • 06/04/2023

The FHIR service supports $import operation that allows you to import data into FHIR service from a storage account. Import splits input files in several data streams for optimal performance and doesn't guarantee order in which resources are processed. There are two modes of $import supported today-

- Initial mode is intended to load FHIR resources into an empty FHIR server. Initial mode only supports CREATE operations and, when enabled, blocks API writes to the FHIR server.
- Incremental mode is optimized to load data into FHIR server periodically and doesn't block writes via API. It also allows you to load lastUpdated and versionId from resource Meta (if present in resource JSON).

> **Important**
>
> Incremental mode capability is currently in preview. Preview APIs and SDKs are provided without a service-level agreement. We recommend that you don't use them for production workloads. Some features might not be supported, or they might have constrained capabilities.
>
> For more information, review **Supplemental Terms of Use for Microsoft Azure Previews**.

In this document we go over the three steps used in configuring import settings on the FHIR service:

1. Enable managed identity on the FHIR service.
2. Create an Azure storage account or use an existing storage account, and then grant permissions to the FHIR service to access it.
3. Set the import configuration in the FHIR service.

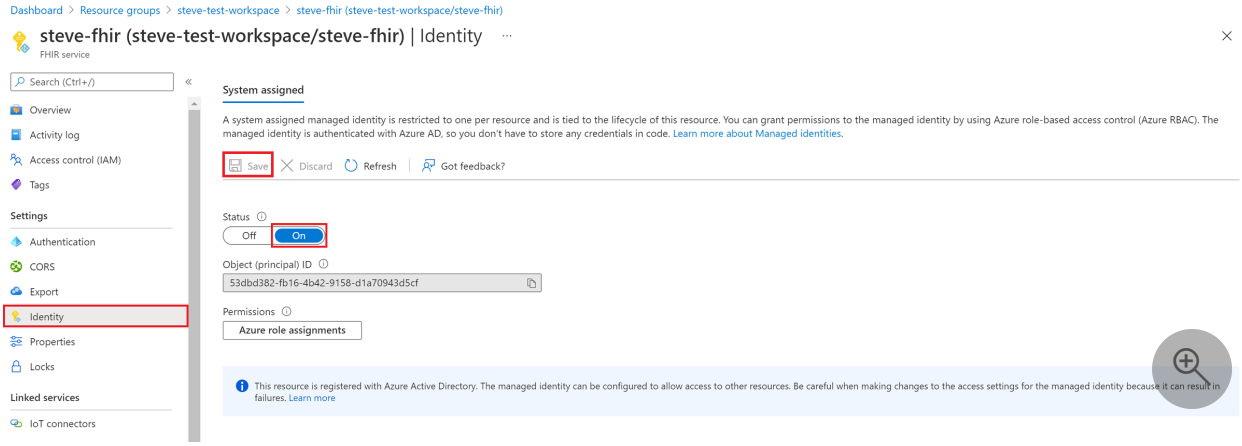
Step 1: Enable managed identity on the FHIR service

The first step is to enable system wide managed identity on the service. This will be used to grant FHIR service an access to the storage account. For more information about managed identities in Azure, see About managed identities for Azure resources.

Follow the steps to enable managed identity on FHIR service

1. Browse to your FHIR service in the Azure portal.
2. Select the **Identity** blade.
3. Select the **Status** option to **On** , and then select **Save**.
4. Select **Yes** to enable the managed identity for FHIR service.

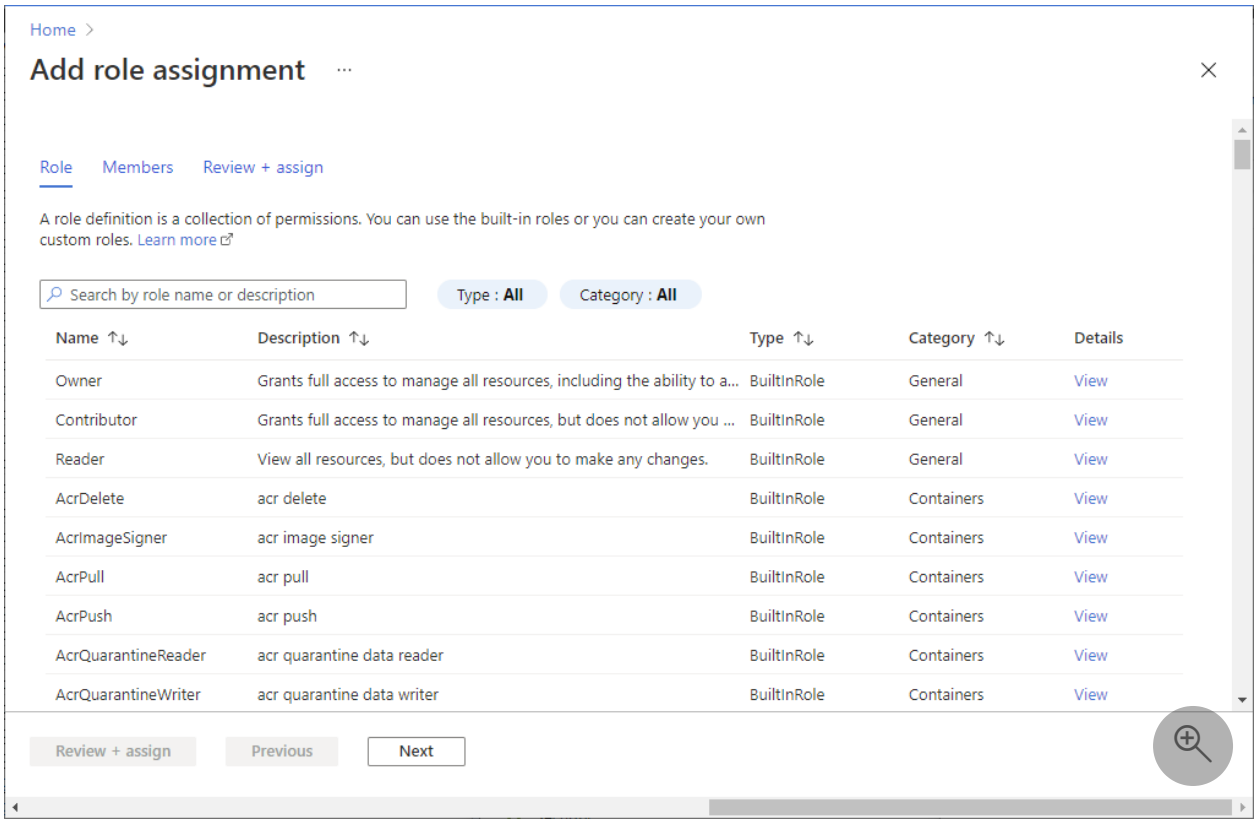
After the system identity has been enabled, you'll see a system assigned GUID value.



Step 2: Assign permissions to the FHIR service to access the storage account

Follow the steps below to assign permissions to access the storage account

1. Browse to the **Access Control (IAM)** in the storage account.
2. Select **Add role assignment**. During this step, if the add role assignment option is grayed out, you need to ask your Azure Administrator to assign you permission to perform this step. For more information about assigning roles in the Azure portal, see Azure built-in roles.
3. Add the role Storage Blob Data Contributor to the FHIR service.
4. Select **Save**.



Now you're ready to select the storage account for import.

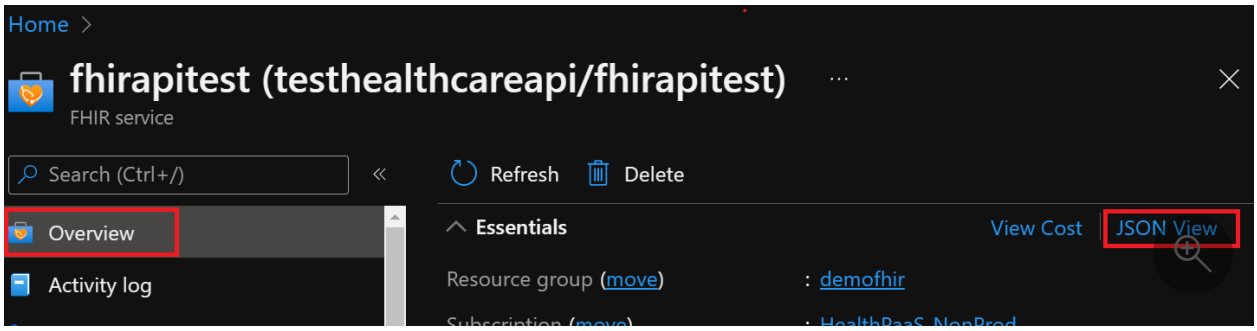
Step 3: Set import configuration of the FHIR service

> **Note**
>
> If you haven't assigned storage access permissions to the FHIR service, the import operations ($import) will fail.

For this step you need to get request URL and JSON body. Follow the directions below

1. Browse to the Azure portal of your FHIR service.
2. Select **Overview**.
3. Select **JSON View**.
4. Select the API version to **2022-06-01** or later version.

To specify the Azure Storage account in JSON view, you need to use REST API to update the FHIR service.

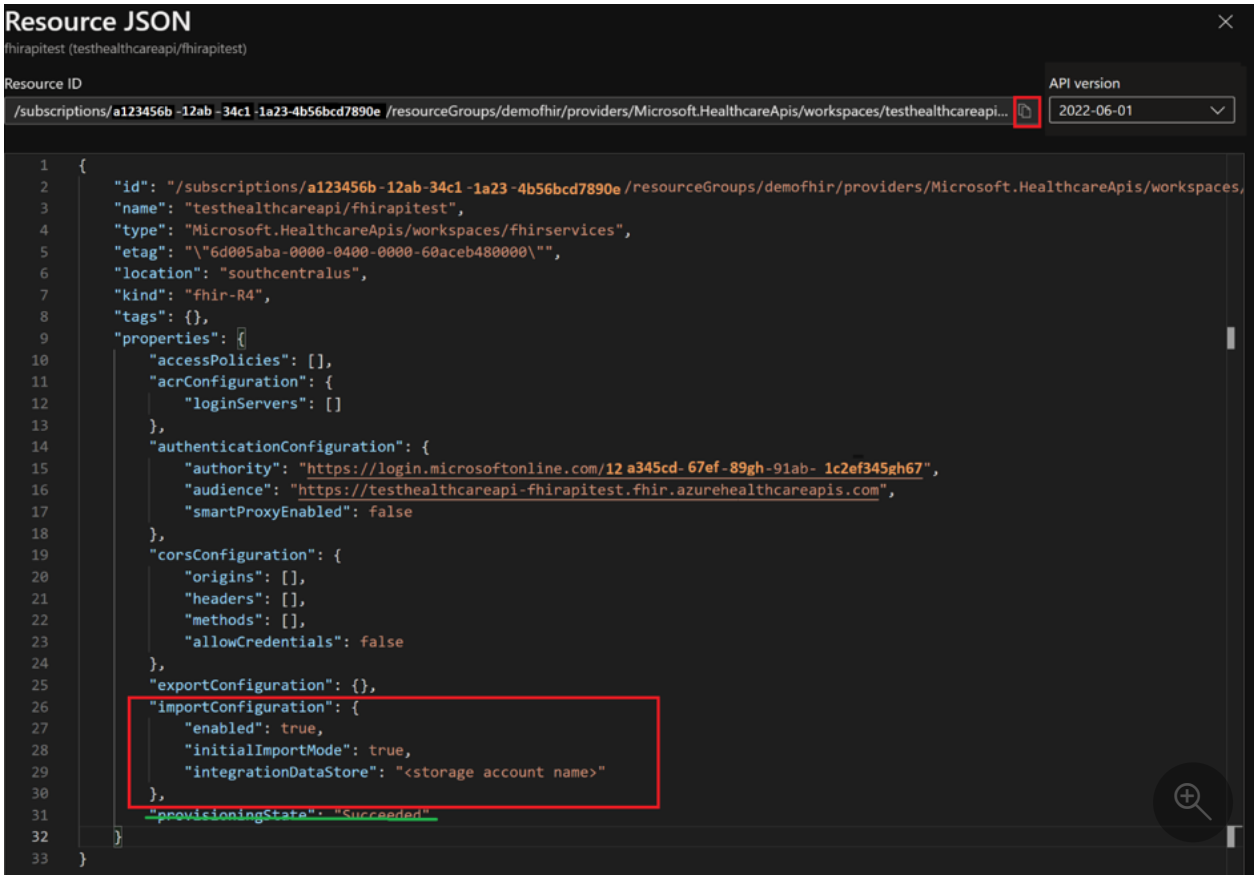


Below steps walk through setting configurations for initial and incremental import mode. Choose the right import mode for your use case.

Step 3.1: Set import configuration for Initial import mode.

Do following changes to JSON:

1. Set enabled in importConfiguration to **true**.
2. Update the integrationDataStore with target storage account name.
3. Set initialImportMode in importConfiguration to **true**.
4. Drop off provisioningState.



After you've completed this final step, you're ready to perform **Initial mode** import using $import.

Step 3.2: Set import configuration for Incremental import mode.

Do following changes to JSON:

1. Set enabled in importConfiguration to **true**.
2. Update the integrationDataStore with target storage account name.
3. Set initialImportMode in importConfiguration to **false**.
4. Drop off provisioningState.

After you've completed this final step, you're ready to perform **Incremental mode** import using $import.

Note : You can also use the **Deploy to Azure** button to open custom Resource Manager template that updates the configuration for $import.

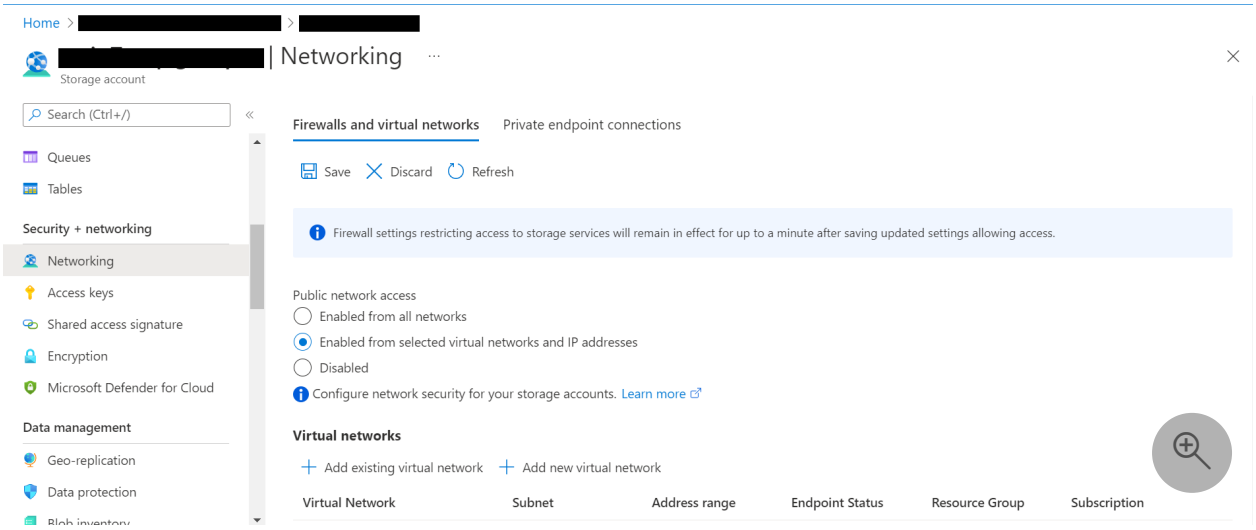
Securing the FHIR service $import operation

For you to securely import FHIR data into the FHIR service from an ADLS Gen2 account, there are two options:

- Option 1: Enabling FHIR service as a Microsoft Trusted Service.
- Option 2: Allowing specific IP addresses associated with the FHIR service to access the storage account. This option permits two different configurations depending on whether or not the storage account is in the same Azure region as the FHIR service.

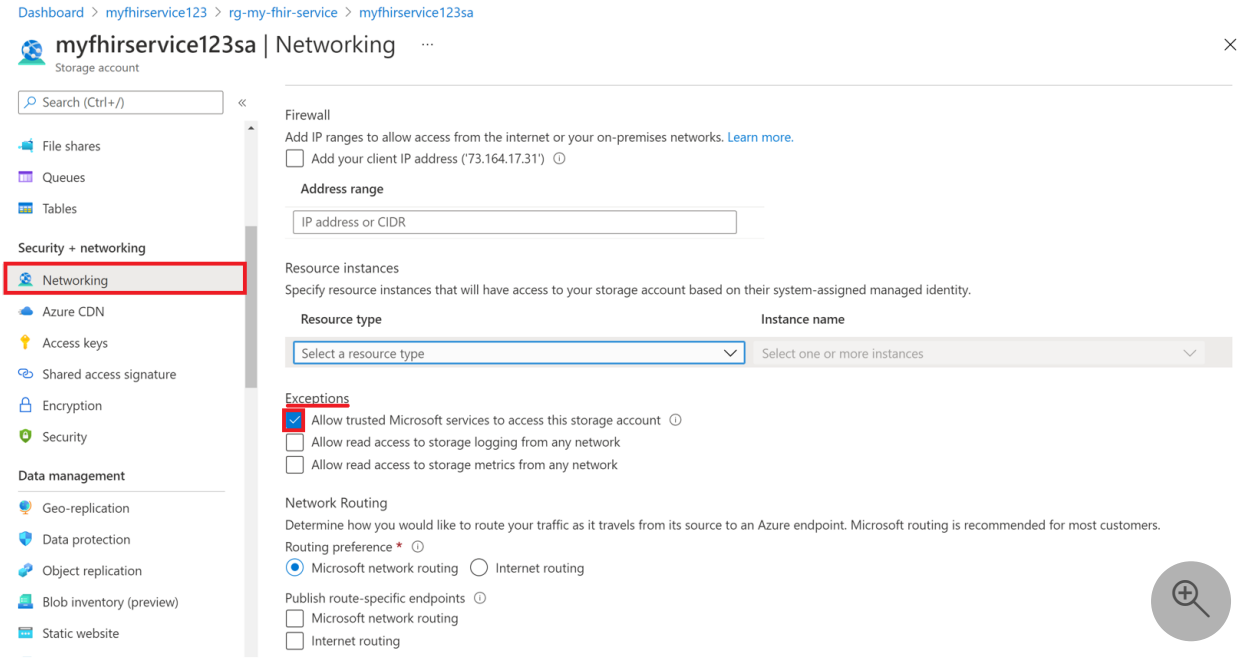
Option 1: Enabling FHIR service as a Microsoft Trusted Service.

Go to your ADLS Gen2 account in the Azure portal and select the **Networking** blade. Select **Enabled from selected virtual networks and IP addresses** under the **Firewalls and virtual networks** tab.



Select **Microsoft.HealthcareApis/workspaces** from the **Resource type** dropdown list and then select your workspace from the **Instance name** dropdown list.

Under the **Exceptions** section, select the box **Allow Azure services on the trusted services list to access this storage account**. Make sure to click **Save** to retain the settings.



Next, run the following PowerShell command to install the `Az.Storage` PowerShell module in your local environment. This will allow you to configure your Azure storage account(s) using PowerShell.

PowerShell

```powershell
Install-Module Az.Storage -Repository PsGallery -AllowClobber -Force
```

Now, use the PowerShell command below to set the selected FHIR service instance as a trusted resource for the storage account. Make sure that all listed parameters are

defined in your PowerShell environment.

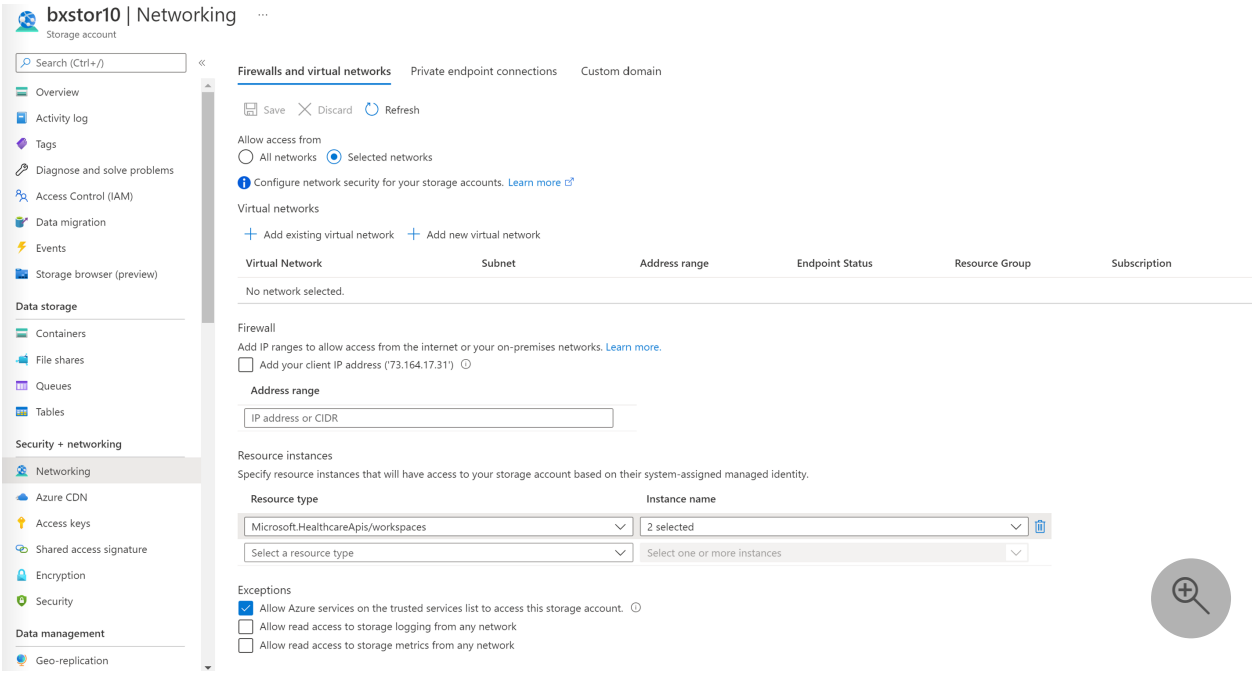
Note that you need to run the `Add-AzStorageAccountNetworkRule` command as an administrator in your local environment. For more information, see Configure Azure Storage firewalls and virtual networks.

PowerShell

```
$subscription="xxx"
$tenantId = "xxx"
$resourceGroupName = "xxx"
$storageaccountName = "xxx"
$workspacename="xxx"
$fhirname="xxx"
$resourceId =
"/subscriptions/$subscription/resourceGroups/$resourceGroupName/providers/Microsoft.HealthcareApis/workspaces/$workspacename/fhirservices/$fhirname"

Add-AzStorageAccountNetworkRule -ResourceGroupName $resourceGroupName -Name
$storageaccountName -TenantId $tenantId -ResourceId $resourceId
```

After you've executed above command, in the **Firewall** section under **Resource instances** you'll see **2 selected** in the **Instance name** dropdown list. These are the names of the workspace instance and FHIR service instance that you registered as Microsoft Trusted Resources.



You're now ready to securely import FHIR data from the storage account. The storage account is on selected networks and isn't publicly accessible. To securely access the files, you can enable private endpoints for the storage account.

Option 2: Allowing specific IP addresses to access the Azure storage account

Option 2.1: Access storage account provisioned in different Azure region than FHIR service

In the Azure portal, go to the ADLS Gen2 account and select the **Networking** blade.

Select **Enabled from selected virtual networks and IP addresses**. 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 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

Azure Region	Public IP Address
West Central US	52.150.156.44
West Europe	20.61.98.66
West US 2	40.64.135.77

Option 2.2: Access storage account provisioned in same Azure region as FHIR service

The configuration process for IP addresses in the same region is just like above except a specific IP address range in Classless Inter-Domain Routing (CIDR) format is used instead (that is, 100.64.0.0/10). The reason why the IP address range (100.64.0.0 – 100.127.255.255) must be specified is because an IP address for the FHIR service will be allocated each time an `$import` request is made.

> **Note**
>
> It is possible that a private IP address within the range of 10.0.2.0/24 may be used, but there is no guarantee that the `$import` operation will succeed in such a case. You can retry if the `$import` request fails, but until an IP address within the range of 100.64.0.0/10 is used, the request will not succeed. This network behavior for IP address ranges is by design. The alternative is to configure the storage account in a different region.

Next steps

In this article, you've learned how the FHIR service supports $import operation and it allows you to import data into FHIR service from a storage account. You also learned about the three steps used in configuring import settings in the FHIR service. For more information about converting data to FHIR, exporting settings to set up a storage account, and moving data to Azure Synapse, see

Use $import

Import Operation

Article • 09/05/2023

Import operation enables loading Fast Healthcare Interoperability Resources (FHIR®) data to the FHIR server at high throughput using the $import operation. Import supports both initial and incremental data load into FHIR server.

Using $import operation

> ⓘ **Note**
>
> You must have the **FHIR Data Importer** role on the FHIR server to use $import.

To use $import, you need to configure the FHIR server using the instructions in the Configure import settings article. The FHIR data to be imported must be stored in resource specific files in FHIR NDJSON format on the Azure blob store.

For import operation, ensure

- All the resources in a file must be of the same type. You may have multiple files per resource type.
- The data to be imported must be in the same Tenant as of the FHIR service.
- Maximum number of files to be imported per operation is 10,000.

Note:

- Import operation does not support conditional references in resources.
- During import operation, If multiple resources share the same resource ID, then only one of those resources is imported at random. There is an error logged for the resources sharing the same resource ID.

Calling $import

Make a `POST` call to `<<FHIR service base URL>>/$import` with the request header and body shown:

Request Header

```
HTTP
```

```
Prefer:respond-async
Content-Type:application/fhir+json
```

Body

Parameter Name	Description	Card.	Accepted values
inputFormat	String representing the name of the data source format. Currently only FHIR NDJSON files are supported.	1..1	`application/fhir+ndjson`
mode	Import mode value	1..1	For initial import use `InitialLoad` mode value. For incremental import mode use `IncrementalLoad` mode value. If no mode value is provided, IncrementalLoad mode value is considered by default.
input	Details of the input files.	1..*	A JSON array with three parts described in the table below.

Input part name	Description	Card.	Accepted values
type	Resource type of input file	1..1	A valid FHIR resource type that matches the input file.
URL	Azure storage url of input file	1..1	URL value of the input file that can't be modified.
etag	Etag of the input file on Azure storage used to verify the file content hasn't changed.	0..1	Etag value of the input file that can't be modified.

Sample body for Initial load import:

JSON

```
{
    "resourceType": "Parameters",
    "parameter": [
        {
            "name": "inputFormat",
            "valueString": "application/fhir+ndjson"
        },
        {
            "name": "mode",
```

```
            "valueString": "InitialLoad"
        },
        {
            "name": "input",
            "part": [
                {
                    "name": "type",
                    "valueString": "Patient"
                },
                {
                    "name": "url",
                    "valueUri":
"https://example.blob.core.windows.net/resources/Patient.ndjson"
                },
                {
                    "name": "etag",
                    "valueUri": "0x8D92A7342657F4F"
                }
            ]
        },
        {
            "name": "input",
            "part": [
                {
                    "name": "type",
                    "valueString": "CarePlan"
                },
                {
                    "name": "url",
                    "valueUri":
"https://example.blob.core.windows.net/resources/CarePlan.ndjson"
                }
            ]
        }
    ]
}
```

Checking import status

Once $import is initiated, an empty response body with a **callback** link is returned in the `Content-location` header of the response together with `202-Accepted` status code. Store this callback link to check the import status.

To check import status, make the REST call with the `GET` method to the **callback** link returned in the previous step. You can interpret the response using the following table:

Response code	Response body	Description
202 Accepted		The operation is still running.
200 OK	The response body doesn't contain any error.url entry	All resources were imported successfully.
200 OK	The response body contains some error.url entry	Error occurred while importing some of the resources. See the files located at error.url for the details. Rest of the resources were imported successfully.
Other		A fatal error occurred and the operation has stopped. Successfully imported resources haven't been rolled back.

Table below provides some of the important fields in the response body:

Field	Description
transactionTime	Start time of the bulk-import operation.
output.count	Count of resources that were successfully imported
error.count	Count of resources that weren't imported due to some error
error.url	URL of the file containing details of the error. Each error.url is unique to an input URL.

Sample response:

JSON


```
{
    "transactionTime": "2021-07-16T06:46:52.3873388+00:00",
    "request": "https://importperf.azurewebsites.net/$Import",
    "output": [
        {
            "type": "Patient",
            "count": 10000,
            "inputUrl":
"https://example.blob.core.windows.net/resources/Patient.ndjson"
        },
        {
            "type": "CarePlan",
            "count": 199949,
            "inputUrl":
"https://example.blob.core.windows.net/resources/CarePlan.ndjson"
        }
```

```
    ],
    "error": [
        {
        "type": "OperationOutcome",
        "count": 51,
        "inputUrl":
"https://example.blob.core.windows.net/resources/CarePlan.ndjson",
        "url":
"https://example.blob.core.windows.net/fhirlogs/CarePlan06b88c6933a34c7c83cb
18b7dd6ae3d8.ndjson"
        }
    ]
}
```

Troubleshooting

Lets walk-through solutions to some error codes you may encounter during the import operation.

200 OK, but there's an error with the URL in the response

Behavior: Import operation succeeds and returns `200 OK`. However, `error.url` are present in the response body. Files present at the `error.url` location contain JSON fragments similar to below example:

JSON

```
{
    "resourceType": "OperationOutcome",
    "issue": [
        {
            "severity": "error",
            "details": {
                "text": "Given conditional reference '{0}' does'nt resolve
to a resource."
            },
            "diagnostics": "Failed to process resource at line: {0} with
stream start offset: {1}"
        }
    ]
}
```

Cause: NDJSON files contain resources with conditional references, which are currently not supported by $import.

Solution: Replace the conditional references to normal references in the NDJSON files.

400 Bad Request

Behavior: Import operation failed and `400 Bad Request` is returned. Response body has the following content:

JSON

```
{
    "resourceType": "OperationOutcome",
    "id": "13876ec9-3170-4525-87ec-9e165052d70d",
    "issue": [
        {
            "severity": "error",
            "code": "processing",
            "diagnostics": "import operation failed for reason: No such host
is known. (example.blob.core.windows.net:443)"
        }
    ]
}
```

Solution: Verify the link to the Azure storage is correct. Check the network and firewall settings to make sure that the FHIR server is able to access the storage. If your service is in a VNet, ensure that the storage is in the same VNet or in a VNet that has peering with the FHIR service VNet.

403 Forbidden

Behavior: Import operation failed and `403 Forbidden` is returned. The response body has the following content:

JSON

```
{
    "resourceType": "OperationOutcome",
    "id": "bd545acc-af5d-42d5-82c3-280459125033",
    "issue": [
        {
            "severity": "error",
            "code": "processing",
            "diagnostics": "import operation failed for reason: Server
failed to authenticate the request. Make sure the value of Authorization
header is formed correctly including the signature."
        }
    ]
}
```

Cause: We use managed identity for source storage auth. This error may be caused by a missing or wrong role assignment.

Solution: Assign *Storage Blob Data Contributor* role to the FHIR server following the RBAC guide.

500 Internal Server Error

Behavior: Import operation failed and `500 Internal Server Error` is returned. Response body has the following content:

JSON

```
{
    "resourceType": "OperationOutcome",
    "id": "0d0f007d-9e8e-444e-89ed-7458377d7889",
    "issue": [
        {
            "severity": "error",
            "code": "processing",
            "diagnostics": "import operation failed for reason: The database
'****' has reached its size quota. Partition or delete data, drop indexes,
or consult the documentation for possible resolutions."
        }
    ]
}
```

Cause: You've reached the storage limit of the FHIR service.

Solution: Reduce the size of your data or consider Azure API for FHIR, which has a higher storage limit.

Next steps

In this article, you've learned about how the import operation enables importing FHIR data to the FHIR server at high throughput. For more information about converting data to FHIR, exporting settings to set up a storage account, and moving data to Azure Synapse, see

Converting your data to FHIR

Configure export settings and set up a storage account

Copy data from Azure API for FHIR to Azure Synapse Analytics

$member-match operation in FHIR service

Article • 09/26/2022

$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 FHIR service in Azure Health Data Services (hereby called FHIR service) 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:

```type3
"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"
        }
```

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

Using Patient-everything in FHIR service

Article • 09/26/2022

The Patient-everything operation is used to provide a view of all resources related to a patient. This operation can be useful to give patients' access to their entire record or for a provider or other user to perform a bulk data download related to a patient. According to the Fast Healthcare Interoperability Resources (FHIR®) specification, Patient-everything returns all the information related to one or more patients described in the resource or context on which this operation is invoked. In the FHIR service in Azure Health Data Services(hereby called FHIR service), Patient-everything is available to pull data related to a specific patient.

Use Patient-everything

To call Patient-everything, use the following command:

```
JSON

GET {FHIRURL}/Patient/{ID}/$everything
```

> **Note**
>
> You must specify an ID for a specific patient. If you need all data for all patients, see **$export**.

FHIR service 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 references that aren't of see also or if the `seealso` link references a `RelatedPerson`.
- If there are `seealso` link reference(s) to other patient(s), the results will include Patient-everything operation against the `seealso` patient(s) listed.
- Resources in the Patient Compartment.
- Device resources that reference the patient resource.

> **Note**

If the patient has more than 100 devices linked to them, only 100 will be returned.

Patient-everything parameters

FHIR service supports the following query parameters. All of these parameters are optional:

Query parameter	Description
_type	Allows you to specify which types of resources will be included in the response. For example, _type=Encounter would return only `Encounter` resources associated with the patient.
_since	Will return only resources that have been modified since the time provided.
start	Specifying the start date will pull in resources where their clinical date is after the specified start date. If no start date is provided, all records before the end date are in scope.
end	Specifying the end date will pull in resources where their clinical date is before the specified end date. If no end date is provided, all records after the start date are in scope.

> ⓘ Note
>
> This implementation of Patient-everything does not support the _count parameter.

Processing patient links

On a patient resource, there's an element called link, which links a patient to other patients or related persons. These linked patients help give a holistic view of the original patient. The link reference can be used when a patient is replacing another patient or when two patient resources have complementary information. One use case for links is when an ADT 38 or 39 HL7v2 message comes. It describes an update to a patient. This update can be stored as a reference between two patients in the link element.

The FHIR specification has a detailed overview of the different types of patient links, but we've include a high-level summary:

- replaces - The Patient resource replaces a different Patient.

- refer - Patient is valid, but it's not considered the main source of information. Points to another patient to retrieve additional information.
- seealso - Patient contains a link to another patient that's equally valid.
- replaced-by - The Patient resource replaces a different Patient.

Patient-everything patient links details

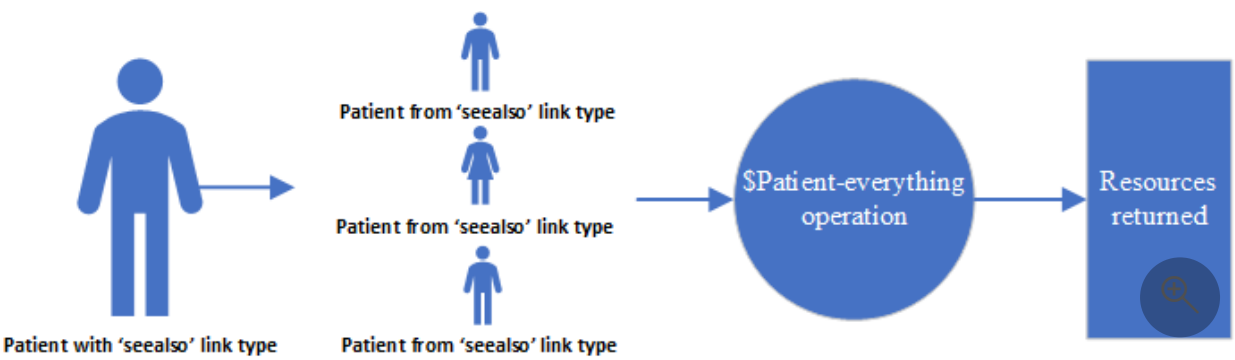
The Patient-everything operation in the FHIR service processes patient links in different ways to give you the most holistic view of the patient.

> ⓘ Note
>
> A link can also reference a `RelatedPerson`. Right now, `RelatedPerson` resources are not processed in Patient-everything and are not returned in the bundle.

Right now, replaces and refer links are ignored by the Patient-everything operation, and the linked patient isn't returned in the bundle.

As described, seealso links reference another patient that's considered equally valid to the original. After the Patient-everything operation is run, if the patient has `seealso` links to other patients, the operation runs Patient-everything on each `seealso` link. This means if a patient links to five other patients with a type `seealso` link, we'll run Patient-everything on each of those five patients.

> ⓘ Note
>
> This is set up to only follow `seealso` links one layer deep. It doesn't process a `seealso` link's `seealso` links.



The final link type is replaced-by. In this case, the original patient resource is no longer being used and the `replaced-by` link points to the patient that should be used. This implementation of `Patient-everything` will include by default an operation

outcome at the start of the bundle with a warning that the patient is no longer valid. This will also be the behavior when the `Prefer` header is set to `handling=lenient`.

In addition, you can set the `Prefer` header to `handling=strict` to throw an error instead. In this case, a return of error code 301 `MovedPermanently` indicates that the current patient is out of date and returns the ID for the correct patient that's included in the link. The `ContentLocation` header of the returned error will point to the correct and up-to-date request.

> ⓘ **Note**
>
> If a `replaced-by` link is present, `Prefer: handling=lenient` and results are returned asynchronously in multiple bundles, only an operation outcome is returned in one bundle.

Patient-everything response order

The Patient-everything operation returns results in phases:

1. Phase 1 returns the `Patient` resource itself in addition to any `generalPractitioner` and `managingOrganization` resources ir references.
2. Phase 2 and 3 both return resources in the patient compartment. If the `start` or `end` query parameters are specified, Phase 2 returns resources from the compartment that can be filtered by their clinical date, and Phase 3 returns resources from the compartment that can't be filtered by their clinical date. If neither of these parameters are specified, Phase 2 is skipped and Phase 3 returns all patient-compartment resources.
3. Phase 4 will return any devices that reference the patient.

Each phase will return results in a bundle. If the results span multiple pages, the next link in the bundle will point to the next page of results for that phase. After all results from a phase are returned, the next link in the bundle will point to the call to initiate the next phase.

If the original patient has any `seealso` links, phases 1 through 4 will be repeated for each of those patients.

Examples of Patient-everything

Here are some examples of using the Patient-everything operation. In addition to these examples, we have a sample REST file that illustrates how the `seealso` and `replaced-by` behavior works.

To use Patient-everything to query a patient's "everything" between 2010 and 2020, use the following call:

```
JSON

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:

```
JSON

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:

```
JSON

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 steps

Now that you know how to use the Patient-everything operation, you can learn about the search options. For more information, see

Overview of FHIR search

Purge history operation

Article • 09/26/2022

`$purge-history` is an operation that allows you to delete the history of a single Fast Healthcare Interoperability Resources (FHIR®) resource. This operation isn't defined in the FHIR specification, but it's useful for history management in large FHIR service instances.

Overview of purge history

The `$purge-history` operation was created to help with the management of resource history in FHIR service. It's uncommon to need to purge resource history. However, it's needed in cases when the system level or resource level versioning policy changes, and you want to clean up existing resource history.

Since `$purge-history` is a resource level operation versus a type level or system level operation, you'll need to run the operation for every resource that you want remove the history from.

Examples of purge history

To use `$purge-history`, you must add `/$purge-history` to the end of a standard delete request. The template of the request is:

```HTTP
DELETE <FHIR-Service-Url>/<Resource-Type>/<Resource-Id>/$purge-history
```

For example:

```HTTP
DELETE https://workspace-
fhir.fhir.azurehealthcareapis.com/Observation/123/$purge-history
```

Next steps

In this article, you learned how to purge the history for resources in the FHIR service. For more information about how to disable history and some concepts about history

management, see

Versioning policy and history management

Supported FHIR features

FHIR REST API capabilities for Azure Health Data Services FHIR service

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

Store profiles in FHIR service

Article • 02/22/2023

HL7 Fast Healthcare Interoperability Resources (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 other 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 allows you to narrow down and customize resource definitions using constraints and extensions.

The FHIR service in Azure Health Data Services (hereby called FHIR service) allows validating resources against profiles to see if the resources conform to the profiles. This article guides you through the basics of FHIR profiles and how to store them. For more information about FHIR profiles outside of this article, visit HL7.org.

FHIR profile: the basics

A profile sets additional context on the resource that's represented as a `StructureDefinition` resource. A `StructureDefinition` defines a set of rules on the content of a resource or a data type, such as what elements a resource has and what values these elements can take.

Below are some examples of how profiles can modify the base resource:

- Restrict cardinality: For example, you can set the maximum cardinality on an element to 0, which means that the element is ruled out in the specific context.
- Restrict the contents of an element to a single fixed value.
- Define required extensions for the resource.

A `StructureDefinition` is identified by its canonical URL:
`http://hl7.org/fhir/StructureDefinition/{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 it identifies mandatory fields such as extensions and value sets.

When a resource conforms to a profile, the profile is specified inside the `profile` element of the resource. Below you can see an example of the beginning of a 'Patient' resource, which has http://hl7.org/fhir/us/carin-bb/StructureDefinition/C4BB-Patient profile.

JSON

```
{
  "resourceType" : "Patient",
  "id" : "ExamplePatient1",
  "meta" : {
    "lastUpdated" : "2020-10-30T09:48:01.8512764-04:00",
    "source" : "Organization/PayerOrganizationExample1",
    "profile" : [
      "http://hl7.org/fhir/us/carin-bb/StructureDefinition/C4BB-Patient"
    ]
  },
```

> **Note**
>
> 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 profile cannot make it optional.

Profiles are also specified by various Implementation Guides (IGs). Some common IGs are listed below. For more information, visit the specific IG site to learn more about the IG and the profiles defined within it:

- US Core
- CARIN Blue Button
- Da Vinci Payer Data Exchange
- Argonaut

> **Note**
>
> The FHIR service does not store any profiles from implementation guides by default. You will need to load them into the FHIR service.

Accessing profiles and storing profiles

Storing profiles

To store profiles in FHIR service, you can `PUT` the `StructureDefinition` with the profile content in the body of the request. A standard `PUT` or a conditional update are both good methods to store profiles on the FHIR service. Use the conditional update if you are unsure which to use.

Standard `PUT`: `PUT http://<your FHIR service base URL>/StructureDefinition/profile-id`

or

Conditional update: `PUT http://<your FHIR service base URL>/StructureDefinition?url=http://sample-profile-url`

```
{
"resourceType" : "StructureDefinition",
"id" : "profile-id",
"url": "http://sample-profile-url"
    …
}
```

For example, if you'd like to store the `us-core-allergyintolerance` profile, you'd use the following rest command with the US Core allergy intolerance profile in the body. We've included a snippet of this profile for the example.

HTTP

```
PUT https://<your FHIR service base URL>/StructureDefinition?
url=http://hl7.org/fhir/us/core/StructureDefinition/us-core-
allergyintolerance
```

JSON

```
{
    "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.",
```

For more examples, see the US Core sample REST file on the open-source site that walks through storing US Core profiles. To get the most up to date profiles, you should get the profiles directly from HL7 and the implementation guide that defines them.

Viewing profiles

You can access your existing custom profiles using a `GET` request, `GET http://<your FHIR service base URL>/StructureDefinition?url={canonicalUrl}`, where `{canonicalUrl}` is the canonical URL of your profile.

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

`GET https://<your FHIR service base URL>/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:

JSON

```
{
  "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).",

}
```

> ⓘ **Note**
>
> You'll only see the profiles that you've loaded into the FHIR service.

FHIR service doesn't 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 service. The FHIR service updates the capability statement with details of the stored profiles in the forms of:

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

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

JSON

```
{
  "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'll be returned with a `CapabilityStatement` that includes the following information on the US Core Patient profile you uploaded to your FHIR server:

JSON

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

Bindings in Profiles

A terminology service is a set of functions that can perform operations on medical "terminologies," such as validating codes, translating codes, expanding value sets, etc. The FHIR service doesn't support terminology service. Information for supported operations ($), resource types and interactions can be found in the service's CapabilityStatement. Resource types ValueSet, StructureDefinition and CodeSystem are supported with basic CRUD operations and Search (as defined in the CapabilityStatement) as well as being leveraged by the system for use in $validate.

ValueSets can contain a complex set of rules and external references. Today, the service will only consider the pre-expanded inline codes. Customers need to upload supported ValueSets to the FHIR server prior to utilizing the $validate operation. The ValueSet resources must be uploaded to the FHIR server, using PUT or conditional update as mentioned under Storing Profiles section above.

Next steps

In this article, you've learned about FHIR profiles. Next, you'll learn how you can use $validate to ensure that resources conform to these profiles.

Validate FHIR resources against profiles

Validate FHIR resources against profiles in Azure Health Data Services

Article • 09/06/2023

In the store profiles in the FHIR service article, you walked through the basics of FHIR profiles and storing them. The FHIR service in Azure Health Data Services (hereby called the FHIR service) allows validating resources against profiles to see if the resources conform to the profiles. This article will guide you through how to use `$validate` for validating resources against profiles.

`$validate` is an operation in Fast Healthcare Interoperability Resources (FHIR®) that allows you to ensure that a FHIR resource conforms to the base resource requirements or a specified profile. This operation ensures that the data in FHIR service has the expected attributes and values. For information on validate operation, visit HL7 FHIR Specification. Per specification, Mode can be specified with `$validate`, such as create and update:

- `create`: Azure API for FHIR checks that the profile content is unique from the existing resources and that it's acceptable to be created as a new resource.
- `update`: Checks that the profile is an update against the nominated existing resource (that is no changes are made to the immutable fields).

There are different ways provided for you to validate resource:

- Option 1: Validate an existing resource with validate operation.
- Option 2: Validate a new resource with validate operation.
- Option 3: Validate on resource CREATE/ UPDATE using header.

On the successful validation of an existing/ new resource with the validate operation, resource is not persisted into the FHIR service. Use Option 3: Validate on resource CREATE/ UPDATE using header, to persist successfully validated resource to the FHIR service.

FHIR Service will always return an `OperationOutcome` as the validation results for $validate operation. FHIR service does two step validation, once a resource is passed into $validate endpoint - the first step is a basic validation to ensure resource can be parsed. During resource parsing, individual errors need to be fixed before proceeding further to next step. Once resource is successfully parsed, full validation is conducted as second step.

> **Note**
>
> Any valuesets that are to be used for validation must be uploaded to the FHIR server. This includes any Valuesets which are part of the FHIR specification, as well as any ValueSets defined in Implementation Guides. Only fully expanded Valuesets which contain a full list of all codes are supported. Any ValueSet definitions which reference external sources are not supported.

Option 1: 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 https://myworkspace-myfhirserver.fhir.azurehealthcareapis.com/Patient/a6e11662-def8-4dde-9ebc-4429e68d130e/$validate
```

In this example, you're validating the existing Patient resource `a6e11662-def8-4dde-9ebc-4429e68d130e` against the base Patient resource. If it's valid, you'll get an `OperationOutcome` such as the following code example:

JSON

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

If the resource isn't valid, you'll get an error code and an error message with details on why the resource is invalid. An example `OperationOutcome` gets returned with error messages and could look like the following code example:

JSON

```
{
    "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, the resource didn't conform to the provided Patient profile, which required a patient identifier value and gender.

If you'd 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 for the HL7 base profile for `heartrate`:

`GET https://myworkspace-myfhirserver.fhir.azurehealthcareapis.com/Observation/12345678/$validate?profile=http://hl7.org/fhir/StructureDefinition/heartrate`

Option 2: Validating a new resource

If you'd like to validate a new resource that you're uploading to the server, you can do a `POST` request:

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

For example:

```
POST https://myworkspace-myfhirserver.fhir.azurehealthcareapis.com/Patient/$validate
```

This request will first validate the resource. New resource you're specifying in the request will be created after validation. The server will always return an `OperationOutcome` as the result.

Option 3: Validate on resource CREATE/UPDATE using header

You can choose when you'd like to validate your resource, such as on resource `CREATE` or `UPDATE`. By default, the FHIR service is configured to opt out of validation on resource `Create/Update`. This capability allows to validate on `Create/Update`, using the `x-ms-profile-validation` header. Set `x-ms-profile-validation' to true for validation.

> ⓘ **Note**
>
> In the open-source FHIR service, you can change the server configuration setting, under the CoreFeatures.

JSON

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

To enable strict validation, use 'Prefer: handling' header with value strict. By setting this header, validation warning will be reported as an error.

Next steps

In this article, you learned how to validate resources against profiles using `$validate`. To learn about the other FHIR service supported features, see

Supported FHIR features

Copy data from FHIR service to Azure Synapse Analytics

Article • 01/23/2023

In this article, you'll learn three ways to copy data from the FHIR service in Azure Health Data Services to Azure Synapse Analytics, which is a limitless analytics service that brings together data integration, enterprise data warehousing, and big data analytics.

- Use the FHIR to Synapse Sync Agent OSS tool
- Use the FHIR to CDM pipeline generator OSS tool
- Use $export and load data to Synapse using T-SQL

Using the FHIR to Synapse Sync Agent OSS tool

> **Note**
>
> **FHIR to Synapse Sync Agent** is an open source tool released under MIT license, and is not covered by the Microsoft SLA for Azure services.

The **FHIR to Synapse Sync Agent** is a Microsoft OSS project released under MIT License. It's an Azure function that extracts data from a FHIR server using FHIR Resource APIs, converts it to hierarchical Parquet files, and writes it to Azure Data Lake in near real time. This also contains a script to create external tables and views in Synapse Serverless SQL pool pointing to the Parquet files.

This solution enables you to query against the entire FHIR data with tools such as Synapse Studio, SSMS, and Power BI. You can also access the Parquet files directly from a Synapse Spark pool. You should consider this solution if you want to access all of your FHIR data in near real time, and want to defer custom transformation to downstream systems.

Follow the OSS documentation for installation and usage instructions.

Using the FHIR to CDM pipeline generator OSS tool

> **Note**

FHIR to CDM pipeline generator is an open source tool released under MIT license, and is not covered by the Microsoft SLA for Azure services.

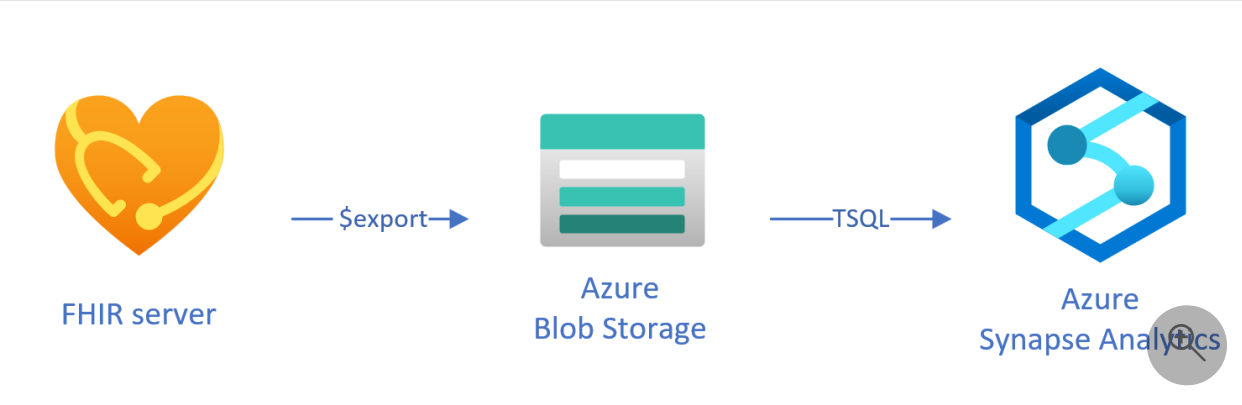
The **FHIR to CDM pipeline generator** is a Microsoft OSS project released under MIT License. It's a tool to generate an ADF pipeline for copying a snapshot of data from a FHIR server using $export API, transforming it to csv format, and writing to a CDM folder in Azure Data Lake Storage Gen 2. The tool requires a user-created configuration file containing instructions to project and flatten FHIR Resources and fields into tables. You can also follow the instructions for creating a downstream pipeline in Synapse workspace to move data from CDM folder to Synapse dedicated SQL pool.

This solution enables you to transform the data into tabular format as it gets written to CDM folder. You should consider this solution if you want to transform FHIR data into a custom schema after it's extracted from the FHIR server.

Follow the OSS documentation for installation and usage instructions.

Loading exported data to Synapse using T-SQL

In this approach, you use the FHIR `$export` operation to copy FHIR resources into a **Azure Data Lake Gen 2 (ADL Gen 2) blob storage** in `NDJSON` format. Subsequently, you load the data from the storage into **serverless or dedicated SQL pools** in Synapse using T-SQL. You can convert these steps into a robust data movement pipeline using Synapse pipelines.



Using `$export` to copy data

Configuring `$export` in the FHIR server

The FHIR server in Azure Health Data Services 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.

To export FHIR data to Azure blob storage, you first need to configure your FHIR server to export data to the storage account. You'll 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}}`:

```
HTTP

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 that you want to export. For example, the following call will export only `Patient`, `MedicationRequest`, and `Observation` resources:

```
HTTP

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.

Using Synapse for Analytics

Creating a Synapse workspace

Before using Synapse, you'll need a Synapse workspace. You'll create an 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 in 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 copy your data to Synapse, you need to create a linked service that connects your Azure Storage account, where you've exported your data, with Synapse. More step-by-step instructions can be found here.

1. In Synapse Studio, browse 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're 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's 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:

```
SQL
```

```
SELECT * FROM
OPENROWSET(bulk
'https://{{youraccount}}.blob.core.windows.net/{{yourcontainer}}/Patient.ndj
son',
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:

SQL

```
-- 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.ndj
```

```
son'
-- 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.

SQL

```
-- 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's a sample SQL query to create a `Patient` table by extracting a few fields from the `Patient` resource:

SQL

```
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
```

Next steps

In this article, you learned three different ways to copy your FHIR data into Synapse.

Next, you can learn about how you can de-identify your FHIR data while exporting it to Synapse in order to protect PHI.

Exporting de-identified data

Frequently asked questions about FHIR service

Article • 08/16/2023

This section covers some of the frequently asked questions about the Azure Health Data Services FHIR service (hereby called FHIR service).

FHIR service: 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 FHIR service supports R4 and the previous version STU3 (Standard for Trial Use 3). For more information on FHIR, visit HL7.org.

Is the data behind the FHIR APIs stored in Azure?

Yes, the data is stored in managed databases in Azure. The FHIR service in Azure Health Data Services doesn't provide direct access to the underlying data store.

How can I get access to the underlying data?

In the managed service, you can't access the underlying data. This is to ensure that the FHIR service offers the privacy and compliance certifications needed for healthcare data. If you need access to the underlying data, you can use the open-source FHIR server.

What identity provider do you support?

We support Microsoft Azure Active Directory as the identity provider.

Can I use Azure AD B2C with the FHIR service?

No, we don't support B2C in the FHIR service. If you need more granular access controls, we recommend looking at the open-source FHIR proxy.

What FHIR version do you support?

We support versions 4.0.0 and 3.0.1.

For more information, see Supported FHIR features. You can also read about what has changed between FHIR versions (STU3 to R4) in the version history for HL7 FHIR.

What is the difference between Azure API for FHIR and the FHIR service in the Azure Health Data Services?

FHIR service is our implementation of the FHIR specification that sits in the Azure Health Data Services, which allows you to have a FHIR service and a DICOM service within a single workspace. Azure API for FHIR was our initial GA product and is still available as a stand-alone product. The main feature differences are:

- FHIR service has a limit of 4 TB, and Azure API for FHIR supports more than 4 TB.
- FHIR service support additional capabilities as ** Transaction bundles. ** Incremental Import. ** Autoscaling is enabled by default.
- Azure API for FHIR has more platform features (such as customer managed keys, and cross region DR) that aren't yet available in FHIR service in Azure Health Data Services.

What's the difference between the FHIR service in Azure Health Data Services and the open-source FHIR server?

FHIR service in Azure Health Data Services 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 we're responsible for maintaining and updating the server and all required compliance work if you're storing PHI data.

In which regions is the FHIR service available?

FHIR service is available in all regions that Azure Health Data Services is available. You can see that on the Products by Region page.

Where can I see what is releasing into the FHIR service?

The release notes page provides an overview of everything that has shipped to the managed service in the previous month.

To see what will be releasing to the managed service, you can review the releases page of the open-source FHIR Server. We've worked to tag items with Azure Health Data Services if they'll release to the managed service and are available two weeks after they are on the release page in open-source. We have also included instructions on how to test the build if you'd like to test in your own environment. We're evaluating how to best share additional managed service updates.

To see what release package is currently in the managed service, you can view the capability statement for the FHIR service and under the `software.version` property. You'll see which package is deployed.

Where can I find what version of FHIR (R4/STU3) is running on my database?

You can find the exact FHIR version exposed in the capability statement under the `fhirVersion` property (FHIR URL/metadata).

Can I switch my FHIR service from STU3 to R4?

No. We don't have a way to change the version of an existing database. You'll need to create a new FHIR service and reload the data. You can leverage the JSON to FHIR converter as a place to start with converting STU3 data into R4.

Can I customize the URL for my FHIR service?

No. You can't change the URL for the FHIR service.

FHIR Implementations and Specifications

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. For more information about SMART, see SMART Health IT.

Does the FHIR service support SMART on FHIR?

Yes, SMART on FHIR capability is supported using AHDS samples. This is referred to as SMART on FHIR(Enhanced). SMART on FHIR(Enhanced) can be considered to meet requirements with SMART on FHIR Implementation Guide (v 1.0.0) and §170.315(g)(10) Standardized API for patient and population services criterion. For more information, visit SMART on FHIR(Enhanced) Documentation.

Can I create a custom FHIR resource?

We don't 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 the FHIR service?

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 can save this as a structure definition resource. To search on extensions, you'll need to define your own search parameters.

How do I see the FHIR service in XML?

In the managed service, we only support JSON. The open-source FHIR server supports JSON and XML. To view the XML version in open-source, use `_format=application/fhir+xml`.

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.

Can I post a bundle to the FHIR service?

We currently support posting batch bundles and posting transaction bundles in the FHIR service.

How can I get all resources for a single patient in the FHIR service?

We support the $patient-everything operation which will get you all data related to a single patient.

What is the default sort when searching for resources in the FHIR service?

We support sorting by string and dateTime fields in the FHIR service. For more information about other supported search parameters, see Overview of FHIR search.

Does the FHIR service support any terminology operations?

No, the FHIR service doesn't support terminology operations today.

What are the differences between delete types in the FHIR service?

There are two basic Delete types supported within the FHIR service. These are Delete and Conditional Delete.

- With Delete, you can choose to do a soft delete (most common type) and still be able to recover historic versions of your record.
- With Conditional Delete, you can pass search criteria to delete a resource one item at a time or several at a time.
- If you passed the `hardDelete` parameter with either Delete or Conditional Delete, all the records and history are deleted and unrecoverable.

Using the FHIR service

Can I perform health checks on FHIR service?

To perform health check on FHIR service , enter `{{fhirurl}}/health/check` in the GET request. You should be able to see Status of FHIR service. HTTP Status code response with 200 and OverallStatus as "Healthy" in response, means your health check is successful. In case of errors, you will receive error response with HTTP status code 404 (Not Found) or status code 500 (Internal Server Error), and detailed information in response body in some scenarios.

Next steps

In this article, you've learned the answers to frequently asked questions about FHIR service. To see the frequently asked questions about FHIR service in Azure API for FHIR,

see

FAQs about Azure API for FHIR

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

Supported FHIR Features

Article • 09/26/2022

FHIR® service in Azure Health Data Services (hereby called FHIR service) provides a fully managed deployment of the open-source FHIR Server and is an implementation of the FHIR standard. This document lists the main features of the FHIR service.

FHIR version

Latest version supported: `4.0.1`

Previous versions also currently supported include: `3.0.2`

REST API

Below is a summary of the supported RESTful capabilities. For more information on the implementation of these capabilities, see FHIR REST API capabilities.

API	Azure API for FHIR	FHIR service in Azure Health Data Services	Comment
read	Yes	Yes	
vread	Yes	Yes	
update	Yes	Yes	
update with optimistic locking	Yes	Yes	
update (conditional)	Yes	Yes	
patch	Yes	Yes	Support for JSON Patch and FHIRPath Patch only.
patch (conditional)	Yes	Yes	Support for JSON Patch and FHIRPath Patch only.
history	Yes	Yes	
create	Yes	Yes	Support both POST/PUT
create (conditional)	Yes	Yes	Issue #1382

API	Azure API for FHIR	FHIR service in Azure Health Data Services	Comment
search	Partial	Partial	See Overview of FHIR Search.
chained search	Yes	Yes	
reverse chained search	Yes	Yes	
batch	Yes	Yes	
transaction	No	Yes	
paging	Partial	Partial	`self` and `next` are supported
intermediaries	No	No	

Extended Operations

All the operations that are supported that extend the REST API.

Search parameter type	Azure API for FHIR	FHIR service in Azure Health Data Services	Comment
$export (whole system)	Yes	Yes	Supports system, group, and patient.
$convert-data	Yes	Yes	
$validate	Yes	Yes	
$member-match	Yes	Yes	
$patient-everything	Yes	Yes	
$purge-history	Yes	Yes	

Role-based access control

FHIR service uses Azure Active Directory for access control.

Service limits

- **Bundle size** - Each bundle is limited to 500 items.

- **Subscription Limit** - By default, each subscription is limited to a maximum of 10 FHIR services. The limit can be used in one or many workspaces.

Next steps

In this article, you've read about the supported FHIR features in the FHIR service. For information about deploying FHIR service, see

Deploy FHIR service