# Program Workspaces and Capacities

Whether you’re deploying a one-off solution in Fabric or you’re developing a large-scale multitenant application, you need the skills to automate the creation and management of workspaces. This chapter explains how to create and configure workspaces using the Fabric REST APIs. Just as a journey of a thousand miles begins with a single step, the deployment of a Fabric solution begins with the creation of a single workspace.

The chapter begins by reviewing the fundamental role that capacities play in the Fabric environment. The chapter also explains the **why** and **how** of assigning workspaces to capacities during the workspace provisioning process. You’ll learn how to configure authorized access to a workspace by adding workspaces roles for users, groups and service principals. The chapter concludes by examining how to automate the provisioning of workspace identity.

## Assign Workspaces to Fabric Capacities

As a Fabric solution developer, it’s essential you understand the role that capacities play in a Fabric environment. Capacities are used by Fabric as a mechanism to allocate a distinct pool of resources for computation. The size of the capacity determines the amount of computation power available. The size of the capacity can also determine which Fabric features are supported as well as what additional user licensing might be required.

In a Fabric environment, all top-level items such as workspaces and capacities are created inside the scope of an Entra Id tenant. You configure resource allocation in a Fabric environment by assigning workspaces to capacities as part of the workspace provisioning process as shown in the following diagram. Keep in mind you cannot assign a workspace from one Entra Id tenant to a capacity in a different Entra Id tenant.

A diagram of a work space

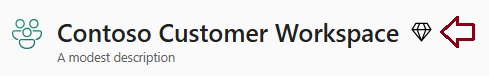
Description automatically generated

Microsoft recommends creating new capacities using Fabric F SKUs. However, it’s also possible to create capacities using Power BI P SKUs. Capacities created using F SKU licensing such as F4, F16 or F64 are known as **Fabric capacities**. Capacities created through a Power BI P SKU licensing such as P1 or P2 are referred to as **premium capacities**. Either type of capacity can be used to develop and deploy Fabric solutions.

There is a one-to-many relationship between capacities and workspaces. A workspace can only be assigned to a single capacity. However, it’s possible to assign multiple workspaces to one capacity. At one extreme, you can assign all your workspaces to one large capacity. At the other extreme, you can assign each workspace to its own dedicated capacity. Most organizations end up somewhere in the middle. With a Fabric environment, you can decide on whatever set of capacities best fits your need to provide the necessary resources to a set of workspaces.

Consider the lonely plight of **Workspace 8** in the prior diagram. This workspace hasn’t been explicitly assigned to either a Fabric capacity or a premium capacity. As it turns out, this type of workspace is automatically assigned to a hidden capacity which is known as the **shared capacity**.

Workspaces which run out of the shared capacity are very limited in terms of functionality and scale. This makes it impractical to develop and test Fabric solutions in your Fabric environment if you don’t have access to a Fabric capacity or premium capacity. In some cases, you can start a Fabric trial which provides a free Fabric capacity for a time period of 60 days. In all case, you want to ensure the workspaces you create are assigned to a Fabric capacity or premium capacities as indicated by the diamond icon to the right of the workspace display name.



Visit the following link to get additional information on Fabric SKUs and Fabric capacity licensing.

* <https://learn.microsoft.com/en-us/fabric/enterprise/licenses>

## Platform Item Types versus Workspace Item Types

The Fabric REST APIs provide support for programming many different types of items. You can separate these item types into two categories which are platform items and workspace items. **Platform items** are created inside a scope defined by the hosting Entra Id tenant. **Workspace items** are created inside the scope of a workspace.

A screenshot of a computer

Description automatically generated

Platform items exist as top-level objects within a Fabric environment. Each platform item has its own type of access control list (ACL) which makes it possible to configure access for users, groups and service principals. For example, the ACL for a workspace is a list of **workspace roles**. You configure access to a workspace by adding workspace roles for users, groups and service principals.

Configuring access to workspace items is different than it is for platform items. In the majority of cases, you just need to configure access at the workspace level by adding a workspace role. Once a user or service principal has been assigned to a workspace role, that security principal has access to all the workspace items inside.

The level of access within a workspace is determined by the type of workspace role. Fabric defines four different types of workspace roles include **Admin**, **Member**, **Contributor** and **Visitor**. You can visit the following link to get additional information and see a table which lists the permissions included with each type of workspace role.

* <https://learn.microsoft.com/en-us/fabric/get-started/roles-workspaces#-workspace-roles>

## Program Capacities

When it comes to programming capacities, there are really only two things you can do with the Fabric REST APIs. First, you can call the **List Capacities** API to retrieve the list of the capacities to which the caller has access. Second, the Fabric REST APIs support assigning a workspace to a capacity. However, the Fabric REST APIs do not support creating new capacities. There is a different API you can use for that which is discussed later this chapter.

### List Capacities

You call the **List Capacities** API by executing a GET request to a URI with **/capacities** appended to the base URI.

GET https://api.fabric.microsoft.com/v1/capacities

In Microsoft’s public cloud, the Fabric REST APIs have a base URI of **https://api.fabric.microsoft.com/v1**. This is the base URL that will be used for code samples in this guidance document. However, you should understand that the base URI for the Fabric REST APIs will be slightly different in sovereign clouds and government clouds.

A successful call to the **List Capacities** API returns an HTTP status code of **200 OK** and a JSON result which contains the list of capacities in the current Entra Id tenant accessible to the caller as seen in the following listing.

{

"value": [

{

"id": "11111111-1111-1111-1111-111111111111",

"displayName": "My Little Capacity",

"sku": "F4",

"region": "West Central US",

"state": "Active"

},

{

"id": "22222222-2222-2222-2222-222222222222",

"displayName": "My Big Capacity",

"sku": "F64",

"region": "West Central US",

"state": " Active"

}

]

}

Now let’s look at a simple example of calling the **List Capacities** API using the .NET SDK. The following code listing demonstrates how to chain a call to **ToList** after the call to **ListCapacities**. This makes it possible to retrieve the capacities list as a standard .NET collection which you can enumerate through using a **foreach** loop.

// create .NET SDK Client object

string accessToken = EntraIdTokenManager.GetFabricAccessToken();

FabricClient fabricApiClient = new FabricClient(accessToken);

// Call List Capacities API

var capacities = fabricApiClient.Core.Capacities.ListCapacities().ToList();

// enumerate through capacities list

foreach (var capacity in capacities) {

Console.WriteLine($"[{capacity.Sku}] {capacity.DisplayName} (ID={capacity.Id})");

}

Remember Fabric REST APIs such as **List Capacities** can return paginated results if the requested list is too large. This isn’t an issue when you’re using the .NET SDK because it automatically handles paginated results for you. If you are calling the **List Capacities** API without using the .NET SDK, you should add extra logic to inspect results for the presence of continuation tokens and to implement the pattern for paginated results as discussed earlier in the **Fabric REST API Fundamentals** chapter.

It’s possible that a call to **List Capacities** will succeed but still return an empty list of capacities. That means your code is running under an identity of a user or service principal which has no access to any premium capacities. This is a problem you will need to resolve. Your code needs access to a premium capacity to which it can assign workspaces during solution deployment.

### Create Fabric Capacities using Code

The Fabric REST API support for programming capacities is limited to the **List Capacities** API and a few other APIs used to assign workspaces to capacities. Assigning workspaces to capacities using the Fabric REST APIs will be covered later in this chapter. However, you should understand that the Fabric REST APIs do not provide support for creating capacities.

You can use the **Azure Microsoft Fabric REST API** to automate the creation, configuration and management of Fabric capacities. This is the same REST API used to createother types of Azure resources such as Managed Identities, ADLS Gen2 Storage containers, Key Vaults and Azure SQL Databases.

The Azure Microsoft Fabric RESTAPIs provide endpoints to list, create and manage Fabric capacities. The following table lists the Azure Microsoft Fabric RESTAPI operations for programing Fabric capacities.

|  |  |  |
| --- | --- | --- |
| Check Name Availability | List By Resource Group | Resume |
| Create Or Update | List By Subscription | Suspend |
| Delete | List Skus | Update |
| Get | List Skus For Capacity |  |

This guidance document will not examine creating and managing capacities using Azure Microsoft Fabric REST API. Reader interested in automating the creation of Fabric capacities can visit the following URL for more information.

* [Fabric Capacities - REST API (Azure Fabric) | Microsoft Learn](https://learn.microsoft.com/en-us/rest/api/microsoftfabric/fabric-capacities?view=rest-microsoftfabric-2023-11-01)

## Program Workspaces

Now we’ll begin with an examination of programming workspaces using the Fabric REST APIs. The chapter will begin with a review of the List Workspaces API and then move onto to the APIs which support CRUD operations. In addition to creating, updating and deleting workspaces, you will learn which APIs to use to assign a workspace to a Fabric capacity, to add workspace roles and to provision workspace identity.

### List Workspaces

Let’s revisit the code listing from the previous chapter which demonstrates calling the List Workspaces API.

// create .NET SDK Client object

string accessToken = EntraIdTokenManager.GetFabricAccessToken();

FabricClient fabricApiClient = new FabricClient(accessToken);

// call the List Workspaces API

List<Workspace> workspaces = fabricApiClient.Core.Workspaces.ListWorkspaces().ToList();

// enumerate through list of workspaces

foreach (var workspace in workspaces) {

Console.WriteLine(workspace.DisplayName);

}

Ssss

GET https://api.fabric.microsoft.com/v1/workspaces

Xxx

{

"value": [

{

"id": "11111111-1111-1111-1111-111111111111",

"displayName": "Workspace 1",

"description": "",

"type": "Workspace",

"capacityId": "99999999-9999-9999-9999-999999999999"

}

]

}

Calling this is a good test to see if a service principal has been given access to API.

Remember that Fabric REST APIs that return list-based results might return paginated results.

### Create a Workspace

You can call the **Create Workspace** API by executing a POST request against the **/workspaces** endpoint.

POST https://api.fabric.microsoft.com/v1/workspaces

You must include JSON in the request body with the **displayName** property. In addition to the **displayName** property which is required, you can optionally add the **description** property and the **capacityId** property.

{

"displayName": "Contoso Customer Workspace",

"description": "A modest description",

"capacityId": "99999999-9999-9999-9999-999999999999"

}

A successful call to the **Create Workspace** API runs synchronously and returns a **201 Created** HTTP status code. A successful response will also include JSON in the response body with several properties including the **id** property.

{

"id": "11111111-1111-1111-1111-111111111111",

"displayName": "Contoso Customer Workspace",

"description": "A modest description",

"type": "Workspace",

"capacityId": "99999999-9999-9999-9999-999999999999"

}

Now let’s look at an example of calling the **Create Workspace** API using the .NET SDK. First, you must create a **CreateWorkspaceRequest** object and initialize it with property values to populate the POST body. Next, you call the **CreateWorkspace** method exposed by the **Workspaces** client passing the **CreateWorkspaceRequest** object.

// acquire data for create workspace request

string workspaceName = "Contoso Customer Workspace";

string workspaceDescription = "Contoso Customer Workspace";

Guid capacityId = new Guid("99999999-9999-9999-9999-999999999999");

// prepare create workspace request

var createRequest = new CreateWorkspaceRequest(workspaceName);

createRequest.Description = workspaceDescription;

createRequest.CapacityId = capacityId;

// call Create Workspace API

Workspace workspace = fabricApiClient.Core.Workspaces.CreateWorkspace(createRequest).Value;

// capture Id of new workspace

Guid workspaceId = workspace.Id;

Take note that there is a chained call to the **Value** property after the call to **CreateWorkspace**. This returns a strongly-typed .NET object with properties for the new workspace. This makes it possible to capture the **Id** property for the new workspace which is important as you begin to create workspace items inside a new workspace.

If the name of a workspace you are creating conflicts with another workspace, the Create Workspace API will return a **409 Conflict** error with a message.

If the capacity Id does not work, you get a **404 Not Found** error.

### Assign a Workspace to a Capacity

Test

var assignmentRequest = new AssignWorkspaceToCapacityRequest(premiumCapcityId);

fabricApiClient.Core.Workspaces.AssignToCapacity(workspaceId, assignmentRequest);

ssss

POST https://api.fabric.microsoft.com/v1/workspaces/{workspaceId}/assignToCapacity

X

{

"capacityId": "88888888-8888-8888-8888-888888888888"

}

s

### Unassign a Workspace from a Capacity

Sss

fabricApiClient.Core.Workspaces.UnassignFromCapacity(workspaceId);

Ssss

POST https://api.fabric.microsoft.com/v1/workspaces/{workspaceId}/unassignFromCapacity

Sssss

ssss

### Get Workspace Information

Sss

WorkspaceInfo workspaceInfo = fabricApiClient.Core.Workspaces.GetWorkspace(workspaceId);

string capacityRegion = workspaceInfo.CapacityRegion.Value.ToString();

sss

GET https://api.fabric.microsoft.com/v1/workspaces/{workspaceId}

Ddd

{

"id": "12341234-1234-1234-1234-123412341234”,

"displayName": "New workspace",

"description": "New workspace description",

"type": "Workspace",

"capacityId": "99999999-9999-9999-9999-999999999999",

"capacityAssignmentProgress": "Completed",

"capacityRegion": "East US",

"workspaceIdentity": {

"applicationId": " aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa",

"servicePrincipalId": "bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb"

},

"oneLakeEndpoints": {

"blobEndpoint": "https://eastus-onelake.blob.fabric.microsoft.com",

"dfsEndpoint": "https://eastus-onelake.dfs.fabric.microsoft.com"

}

}

xxx

### Update a Workspace

Dsdd

var updateReqiest = new UpdateWorkspaceRequest {

DisplayName = "Workspace display name V2",

Description = "A much better description"

};

fabricApiClient.Core.Workspaces.UpdateWorkspace(workspaceId, updateReqiest);

Xxx

PATCH https://api.fabric.microsoft.com/v1/workspaces/{workspaceId}

X

{

"displayName": "Workspace Display Name V2",

"description": "A much better description "

}

xx

### Delete a Workspace

Sssss

fabricApiClient.Core.Workspaces.DeleteWorkspace(workspaceId);

ssss

DELETE https://api.fabric.microsoft.com/v1/workspaces/{workspaceId}

If the call succeeds, it will return a HTTP status code of **OK 200**.

x

### Build a Workflow for Testing Solutions

Ddddd

string workspaceName = "Contoso Customer Workspace";

string workspaceDescription = "Contoso Customer Workspace";

Guid premiumCapcityId = new Guid("99999999-9999-9999-9999-999999999999");

var createRequest = new CreateWorkspaceRequest(workspaceName);

createRequest.Description = workspaceDescription;

createRequest.CapacityId = premiumCapcityId;

// check to see if another workspace already exists wth same display name

var workspaces = fabricApiClient.Core.Workspaces.ListWorkspaces().ToList();

foreach (var workspace in workspaces) {

if (workspace.DisplayName.Equals(workspaceDisplayName)) {

// if workspace exists, delete it and break out of foreach loop

fabricApiClient.Core.Workspaces.DeleteWorkspace(workspace.Id);

break;

}

}

// create workspace for testing

var createRequest = new CreateWorkspaceRequest(workspaceDisplayName);

Workspace newWorkspace = fabricApiClient.Core.Workspaces.CreateWorkspace(createRequest);

## Add Workspace Roles

Adding users

### Add a Workspace Role for a User

public static void AddUserAsWorkspaceMemeber(Guid WorkspaceId, Guid UserId, WorkspaceRole RoleAssignment) {

var user = new Principal(UserId, PrincipalType.User);

var roleAssignment = new AddWorkspaceRoleAssignmentRequest(user, RoleAssignment);

fabricApiClient.Core.Workspaces.AddWorkspaceRoleAssignment(WorkspaceId, roleAssignment);

}

xxxxx

Guid TestUser1Id = new Guid(AppSettings.TestUser1Id);

Guid TestUser2Id = new Guid(AppSettings.TestUser2Id);

AddUserAsWorkspaceMemeber(workspaceId, TestUser1Id, WorkspaceRole.Admin);

AddUserAsWorkspaceMemeber(workspaceId, TestUser2Id, WorkspaceRole.Viewer);

Xxxx

POST https://api.fabric.microsoft.com/v1/workspaces/{workspaceId}/roleAssignments

Cccc

{

"principal": {

"id": "8eedb1b0-3af8-4b17-8e7e-663e61e12211",

"type": "User"

},

"role": "Admin"

}

Ccccc

{

"principal": {

"id": "1a5ba110-02e2-4308-945f-8d95b16b7aff",

"type": "User"

},

"role": "Viewer"

}

xxx

### Add a Workspace Role for a Group

public static void AddGroupAsWorkspaceMemeber(Guid WorkspaceId, Guid GroupId, WorkspaceRole RoleAssignment) {

var group = new Principal(GroupId, PrincipalType.Group);

var roleAssignment = new AddWorkspaceRoleAssignmentRequest(group, RoleAssignment);

fabricApiClient.Core.Workspaces.AddWorkspaceRoleAssignment(WorkspaceId, roleAssignment);

}

sss

Guid TestADGroup1 = new Guid(AppSettings.TestADGroup1);

AddGroupAsWorkspaceMemeber(workspace.Id, TestADGroup1, WorkspaceRole.Member);

Xxxx

{

"principal": {

"id": "9cb0f004-d036-4797-9039-52917b048a57",

"type": "Group"

},

"role": "Member"

}

xxx

### Add a Workspace Role for a Service Principal

xxxx

public static void AddServicePrincipalAsWorkspaceMemeber(Guid WorkspaceId,

Guid ServicePrincipalObjectId,

WorkspaceRole RoleAssignment) {

var user = new Principal(ServicePrincipalObjectId, PrincipalType.ServicePrincipal);

var roleAssignment = new AddWorkspaceRoleAssignmentRequest(user, RoleAssignment);

fabricApiClient.Core.Workspaces.AddWorkspaceRoleAssignment(WorkspaceId, roleAssignment);

}

Xxx

Guid TestServicePrincipal = new Guid(AppSettings.ServicePrincipalObjectId);

FabricRestApi.AddServicePrincipalAsWorkspaceMemeber(workspace.Id, TestServicePrincipal, WorkspaceRole.Admin);

Xxxx

{

"principal": {

"id": "7fafd5a4-9051-4af0-bc8d-29f66b957c25",

"type": "ServicePrincipal"

},

"role": "Admin"

}

## Program Workspace Identity

Xxxxxx

### Provision Workspace Identity

Workspace workspace = fabricApiClient.Core.Workspaces.CreateWorkspace(createRequest).Value;

fabricApiClient.Core.Workspaces.ProvisionIdentity(workspace.Id);

xxxx

POST https://api.fabric.microsoft.com/v1/workspaces/{workspaceId}/provisionIdentity

Successful call to the Provision Workspace Identity will return an HTTP status code of 202 Accepted and begin processing the request as a long running operation. The .NET SDK makes it easy to deal with long running operations. However, those not using the .NET SDK need to monitor the progress of the provisioning process is discussed in the **Fabric REST API Fundamentals** chapter.

Afer

fabricApiClient.Core.Workspaces.ProvisionIdentity(workspace.Id);

WorkspaceInfo workspaceInfo = fabricApiClient.Core.Workspaces.GetWorkspace(workspace.Id);

string servicePrincipalIdForWorkspace = workspaceInfo.WorkspaceIdentity.ServicePrincipalId;

### Deprovision Workspace Identity

Ccccc

fabricApiClient.Core.Workspaces.DeprovisionIdentity(workspace.Id);

c

POST https://api.fabric.microsoft.com/v1/workspaces/{workspaceId}/deprovisionIdentity

When you call the API to deprovision workspace identity, the request is processed as a long running operation.