Azure App Configuration

Welcome to the bonus track! If you made it this far, that's great! You are now ready to learn about Azure App Configuration. This is a service that allows you to store your application settings and feature flags in a central place. It is a great way to decouple your application from its configuration data. It also provides a way to manage the configurations without having to redeploy your application. Another great use for the App Configuration is for sharing configurations between multiple applications.

In this section, you will learn how to create an App Configuration instance and how to use it in your application. You will then reset the configuration values from the App Service to leverage the App Configuration instead of the Key Vault directly (don't worry, the Key Vault is still used to store the secrets). You will also need to make a small code change so that the application can leverage the App Configuration.

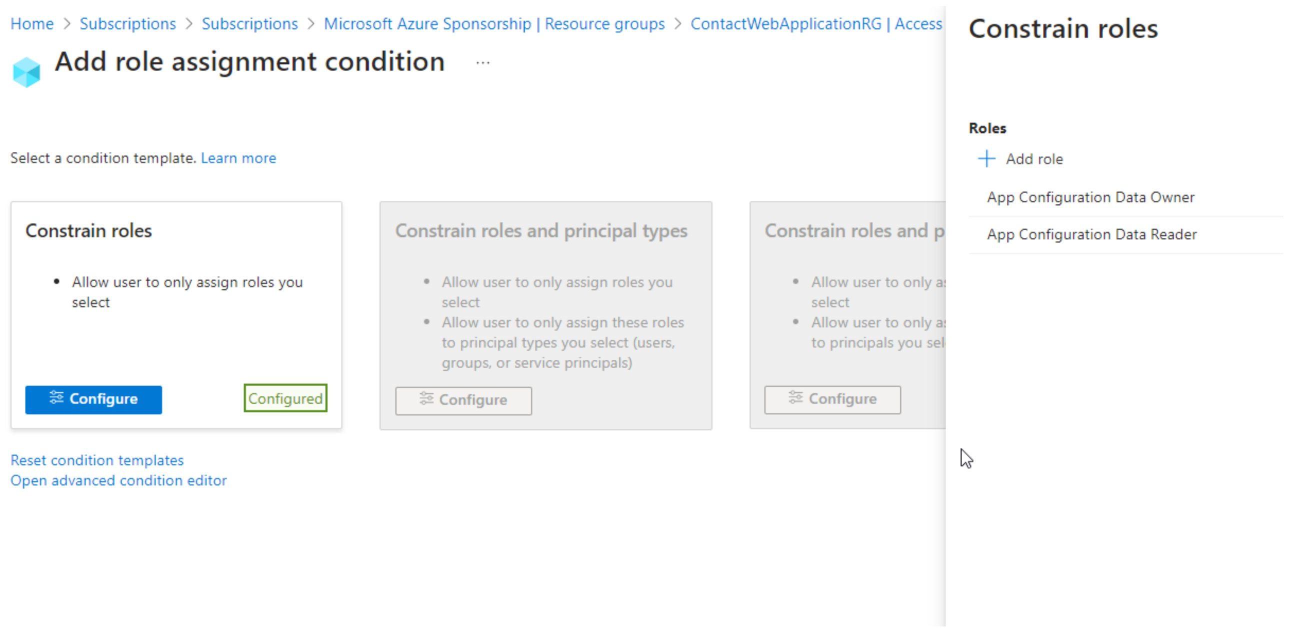
Before you start

It is expected you have completed the first three parts. If not, this part won't make sense. Also, in the earlier part of the work, you created a service principal to execute deployments. That was created with `Contributor` access on the subscription.

1. Elevate the principal to owner

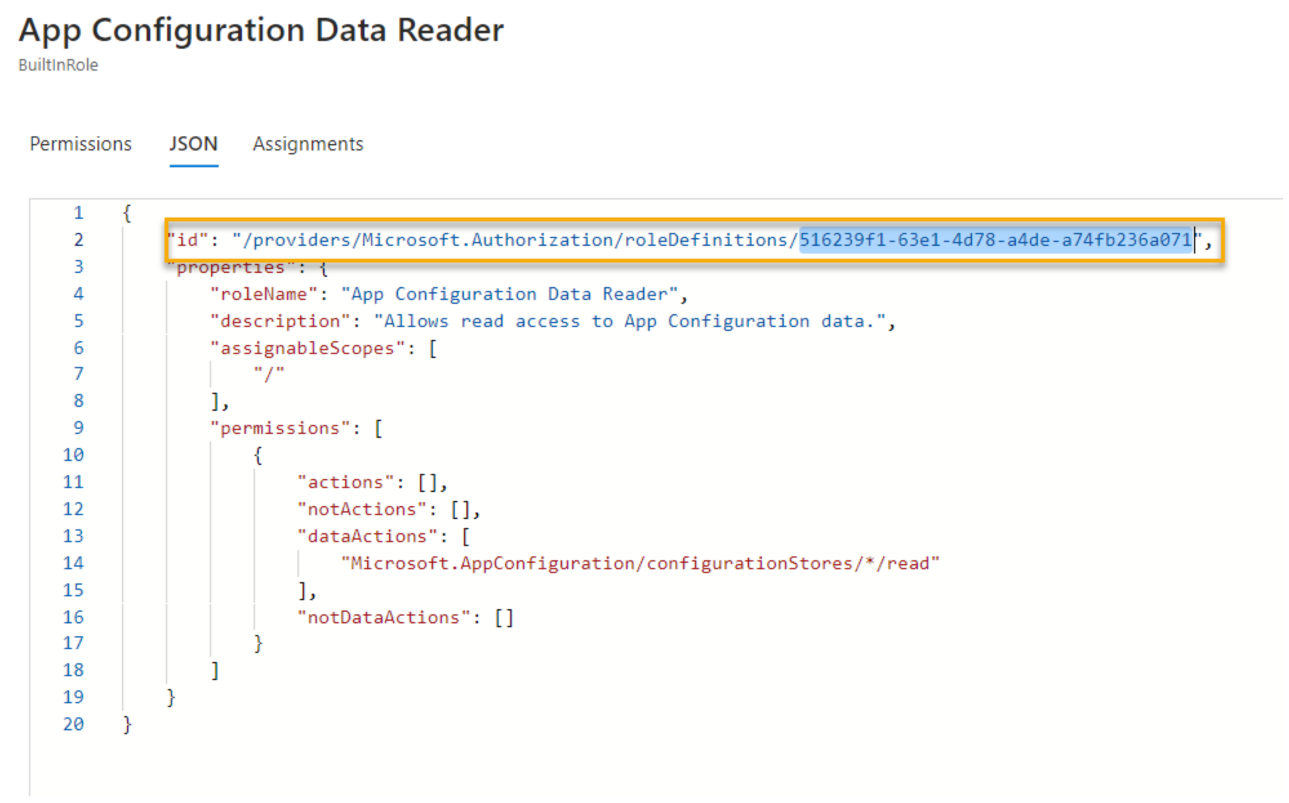
   To complete the role assignment required in this part for `App Configuration Data Reader` for the web app identity, you must elevate the principal to `owner` access. If your resource group is already created, you could only elevate the app registration on the resource group as owner. If not, or if you want to deploy from scratch, you can set the ownership on the subscription and limit the role to be able to assign only for the `App Configuration Data Reader`. Without owner role on at least the resource group, the role assignment will fail during the final part of the app configuration deployment.

   Setting the role to restricted privileges for user access:


Home > Subscriptions > Subscriptions > Microsoft Azure Sponsorship | Resource groups > ContactWebApplicationRG | Access
Add role assignment condition
Select a condition template. Learn more
Constrain roles
Allow user to only assign roles you select
Configure
Configured
Constrain roles and principal types
Allow user to only assign roles you select
Allow user to only assign these roles to principal types you select (users, groups, or service principals)
Configure
Constrain roles and p
Allow user to only a select
Allow user to only a to principals you sel
Configure
Reset condition templates
Open advanced condition editor
Constrain roles
Roles
Add role
App Configuration Data Owner
App Configuration Data Reader

2. Why the App Configuration Data Reader role is hard-coded:

Another note is that it has proven very difficult to get the correct id for the role definition within the bicep. For that reason I've chosen to hard-code this value since it is universal across all subscriptions.

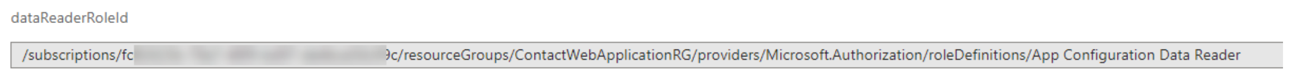


```
{
    "id": "/providers/Microsoft.Authorization/roleDefinitions/516239f1-63e1-4d78-a4de-a74fb236a071",
    "properties": {
        "roleName": "App Configuration Data Reader",
        "description": "Allows read access to App Configuration data.",
        "assignableScopes": [
            "/"
        ],
        "permissions": [
            {
                "actions": [],
                "notActions": [],
                "dataActions": [
                    "Microsoft.AppConfiguration/configurationStores/*/read"
                ],
                "notDataActions": []
            }
        ]
    }
}
```

The value is:

```
/providers/Microsoft.Authorization/roleDefinitions/516239f1-63e1-4d78-a4de-a74fb236a071
```

The output for the retrieved value is the string name, not the id. This is why I've chosen to hard-code the value.

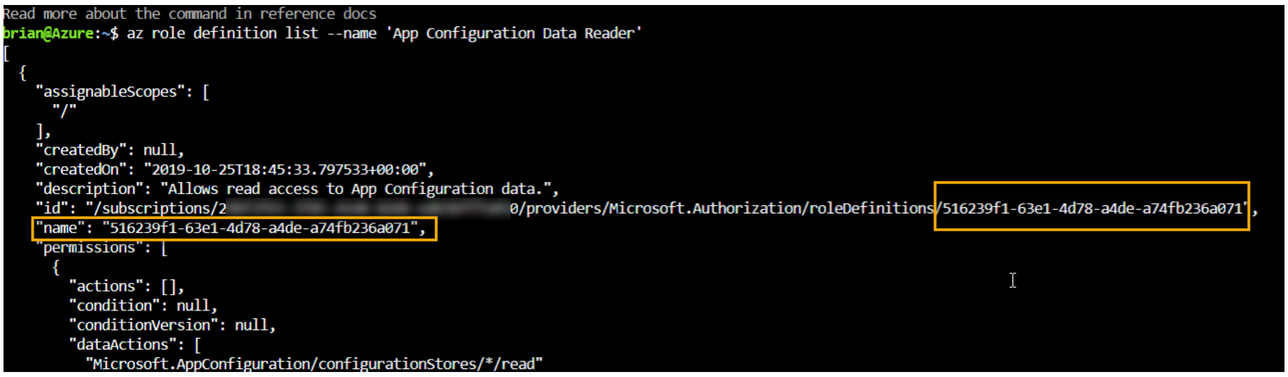


```
/subscriptions/fc[redacted]c/resourceGroups/ContactWebApplicationRG/providers/Microsoft.Authorization/roleDefinitions/App Configuration Data Reader
```

If you want to get the value, you can use the following CLI command:

```
az role definition list --name 'App Configuration Data Reader'
```

You'll see that the `name` field contains the same `516239f1-63e1-4d78-a4de-a74fb236a071` value.



```
Read more about the command in reference docs
brian@Azure:~$ az role definition list --name 'App Configuration Data Reader'
[
  {
    "assignableScopes": [
      "/"
    ],
    "createdBy": null,
    "createdOn": "2019-10-25T18:45:33.797533+00:00",
    "description": "Allows read access to App Configuration data.",
    "id": "/subscriptions/2[redacted]0/providers/Microsoft.Authorization/roleDefinitions/516239f1-63e1-4d78-a4de-a74fb236a071",
    "name": "516239f1-63e1-4d78-a4de-a74fb236a071",
    "permissions": [
      {
        "actions": [],
        "condition": null,
        "conditionVersion": null,
        "dataActions": [
          "Microsoft.AppConfiguration/configurationStores/*/read"
```

3. One last thing - the source files

   For this final part, the source files are located in the `Part4-bicep` folder and the `raw_final` for the `c#` code. The only files that are contained in these folders are files that were modified. You would still need the rest of the bicep files for the deployment, and of course you need the full application for the website to work.

Let's get started!

Part 1 - Update the Bicep for the Key Vault

In hindsight, it would have been better to have a user-managed identity that could be used by both the App Service and the App Configuration from the start of this activity.

However, we are going to keep the system-managed identity for the App Service, and we'll add a new user-managed identity for the App Configuration.

This approach will allow us to use the App Configuration to get the secrets from the Key Vault. Any additional resources that would need to access the Key Vault would be able to use the managed identity going forward.

> **Note:** User managed identities can be shared by resources to allow for configurations and permissions like this for multiple services. However, they pose a small security risk because they are not automatically cleaned up when resources are deleted.

You will also want to add yourself to the permissions. For this I recommend adding a group and putting your user in the group, then getting the object id for the group. This will allow you to add other users to the group in the future without having to update the permissions in the Key Vault, as well as remove those who leave the organization, etc.

Task 1 - Update the Existing Key Vault Bicep

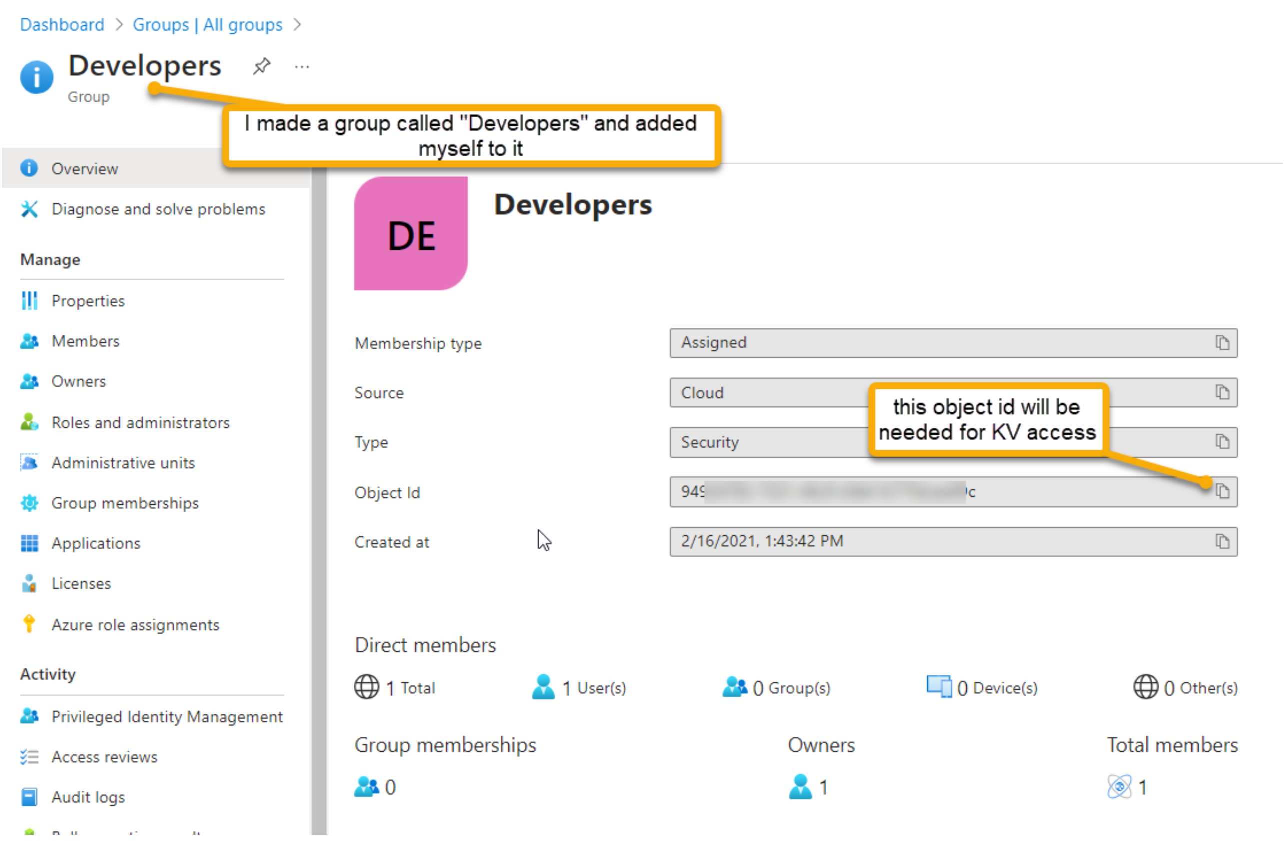
In this task, you'll update the bicep to let your group and a user managed identity have appropriate permissions to the Key Vault secrets, keys, and certificates.

Step 1 - Create your user group

In this step you'll set up your user for permission access via a group membership.

1. In the Azure Portal, navigate to Groups

   Add a new group and add yourself to the group.


Dashboard > Groups | All groups >

Developers
Group

I made a group called "Developers" and added myself to it

DE Developers

Membership type: Assigned
Source: Cloud
Type: Security
Object Id: 94[redacted]c
Created at: 2/16/2021, 1:43:42 PM

this object id will be needed for KV access

Direct members
1 Total | 1 User(s) | 0 Group(s) | 0 Device(s) | 0 Other(s)

Group memberships: 0
Owners: 1
Total members: 1

2. Get the object ID for the group

   You will use this as a parameter to your IaC. In reality you could also have done all of this with the CLI or via other bicep files. However, let's draw the line since this is pretty much a one-time thing per tenant/subscription.

Step 2 - Modify the Key Vault Bicep

In this step you'll modify the key vault bicep to create the user managed identity, and add the group and user managed identity to the permissions.

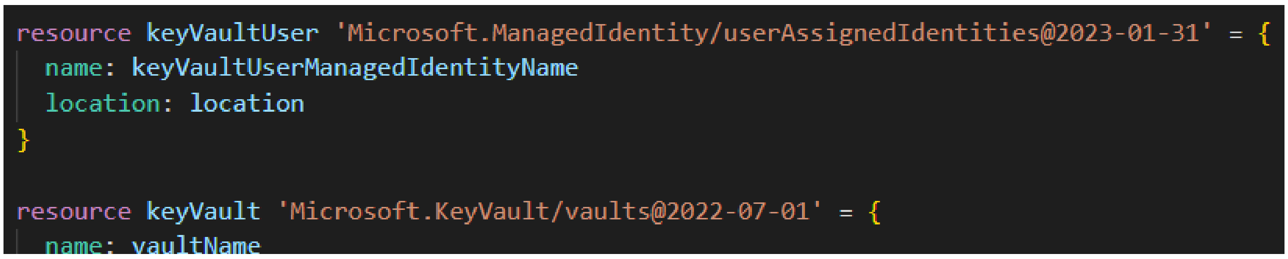
1. Two things are changing. The first is the creation of the user managed identity. The second is the permissions. You'll add the group and the user managed identity to the permissions.

   Update the bicep to include the following:

```
param location string
@description('Provide a unique datetime and initials string to make your instances unique. Use only lower case letters and numbers')
@minLength(11)
@maxLength(11)
param uniqueIdentifier string

@minLength(10)
@maxLength(13)
param keyVaultName string

param webAppFullName string
param databaseServerName string
param sqlDatabaseName string
@secure()
param sqlServerAdminPassword string

param developersGroupObjectId string
param keyVaultUserManagedIdentityName string

var vaultName = '${keyVaultName}${uniqueIdentifier}'
var skuName = 'standard'
var softDeleteRetentionInDays = 7

resource webApp 'Microsoft.Web/sites@2023-01-01' existing = {
  name: webAppFullName
}

resource databaseServer 'Microsoft.Sql/servers@2023-05-01-preview' existing = {
  name: databaseServerName
}

resource keyVaultUser 'Microsoft.ManagedIdentity/userAssignedIdentities@2023-01-
31' = {
  name: keyVaultUserManagedIdentityName
  location: location
}

resource keyVault 'Microsoft.KeyVault/vaults@2022-07-01' = {
  name: vaultName
  location: location
  properties: {
    enabledForDeployment: true
    enabledForDiskEncryption: false
    enabledForTemplateDeployment: true
    tenantId: subscription().tenantId
    enableSoftDelete: true
    softDeleteRetentionInDays: softDeleteRetentionInDays
    sku: {
      name: skuName
      family: 'A'
    }
    accessPolicies: [
      {
        tenantId: subscription().tenantId
        objectId: webApp.identity.principalId
        permissions: {
          keys: []
          secrets: ['Get']
          certificates: []
        }
      }
      {
        tenantId: subscription().tenantId
        objectId: keyVaultUser.properties.principalId
        permissions: {
          keys: []
          secrets: ['Get']
          certificates: []
        }
      }
      {
        tenantId: subscription().tenantId
        objectId: developersGroupObjectId
        permissions: {
          keys: ['all']
          secrets: ['all']
          certificates: ['all']
        }
      }
    ]
    networkAcls: {
      defaultAction: 'Allow'
      bypass: 'AzureServices'
    }
  }
}

resource identityDBConnectionSecret 'Microsoft.KeyVault/vaults/secrets@2022-11-01'
= {
  name: 'IdentityDbConnectionSecret'
  parent: keyVault
  properties: {
    value:
'Server=tcp:${databaseServer.name}${environment().suffixes.sqlServerHostname},1433
;Initial Catalog=${sqlDatabaseName};Persist Security Info=False;User
ID=${databaseServer.properties.administratorLogin};Password=${sqlServerAdminPasswo
rd};MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Conne
ction Timeout=30;'
  }
}

resource contactManagerDBConnectionSecret 'Microsoft.KeyVault/vaults/secrets@2022-
11-01' = {
  name: 'ContactManagerDbConnectionSecret'
  parent: keyVault
  properties: {
    value:
'Server=tcp:${databaseServer.name}${environment().suffixes.sqlServerHostname},1433
;Initial Catalog=${sqlDatabaseName};Persist Security Info=False;User
ID=${databaseServer.properties.administratorLogin};Password=${sqlServerAdminPasswo
rd};MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Conne
ction Timeout=30;'
  }
}

output keyVaultName string = keyVault.name
output identityDBConnectionSecretURI string =
identityDBConnectionSecret.properties.secretUri
output managerDBConnectionSecretURI string =
contactManagerDBConnectionSecret.properties.secretUri
output keyVaultUserManagedIdentityName string = keyVaultUser.name
```

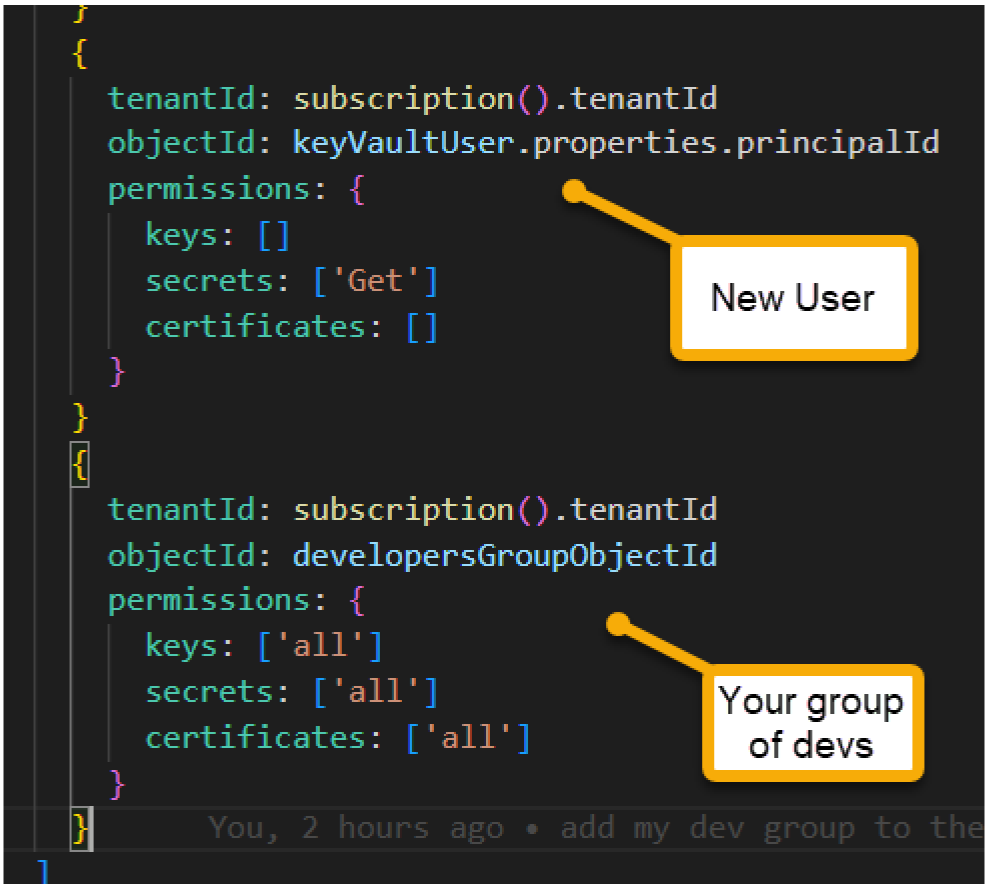
1. Review the code updates

   Notice the new `keyVaultUser` identity that is created using the vault name and the `GetSecretsOnly` as part of the name to help discern what hte UMI is for. Also notice that the `keyVaultUser` is added to the permissions with `Get` permissions for secrets. This is the only permission that is needed for the App Configuration to get the secrets from the Key Vault.

   ```
   resource keyVaultUser 'Microsoft.ManagedIdentity/userAssignedIdentities@2023-01-31' = {
     name: keyVaultUserManagedIdentityName
     location: location
   }

   resource keyVault 'Microsoft.KeyVault/vaults@2022-07-01' = {
     name: vaultName
   ```

   Note that the group id is also added to the vault permissions with `all` permissions for keys, secrets, and certificates. This is so that you can add other users to the group and they will have the same permissions, and you can interact with the vault and secrets from your local code as well as through the portal and app configuration (even if hte App Configuration has rights if you don't you can't do anything with the secrets).

   ```
       }
       {
         tenantId: subscription().tenantId
         objectId: keyVaultUser.properties.principalId
         permissions: {
           keys: []
           secrets: ['Get']
           certificates: []
         }
       }
       {
         tenantId: subscription().tenantId
         objectId: developersGroupObjectId
         permissions: {
           keys: ['all']
           secrets: ['all']
           certificates: ['all']
         }
       }        You, 2 hours ago • add my dev group to the
     ]
   ```

2. Modify the params to include the new parameters

   Add the following to the params for the main deployment:

```
param developersGroupObjectId string
param keyVaultUserManagedIdentityName string
```

```
These values will be passed in and the name is output because it will be
configured and to avoid any possible mistakes it's easier to just leverage the
name once it's created.
```

1. Modify the parameters file to include the new parameters

   Add the following to the parameters file (don't forget to add the comma to the previous line):

```
        "developersGroupObjectId": {
            "value": "some-object-id-passed-in"
        },
        "keyVaultUserManagedIdentityName": {
            "value": "some umi name passed in"
        }
```

1. Modify the main deployment

   Update the `deployContactWebArchitecture.bicep` file to include the new parameters and pass them in to the Key Vault bicep.

```
param keyVaultUserManagedIdentityName string
param developersGroupObjectId string
var keyVaultUMIFullName = '${keyVaultName}-${keyVaultUserManagedIdentityName}'
```

```
Note that the parameter used for the keyvault name will be decorated with the name
of the vault at this top level.

Update the call to the Key Vault bicep to include the new parameters:
```

```
module contactWebVault 'keyVault.bicep' = {
  name: '${keyVaultName}-deployment'
  scope: contactWebResourceGroup
  params: {
    location: contactWebResourceGroup.location
    uniqueIdentifier: uniqueIdentifier
    webAppFullName: contactWebApplicationPlanAndSite.outputs.webAppFullName
    databaseServerName: contactWebDatabase.outputs.sqlServerName
    keyVaultName: keyVaultName
```

```
      sqlDatabaseName: sqlDatabaseName
      sqlServerAdminPassword: sqlServerAdminPassword
      developersGroupObjectId: developersGroupObjectId
      keyVaultUserManagedIdentityName: keyVaultUMIFullName
    }
  }
```

1. Modify the deployContactWebArchitecture.parameters.json file

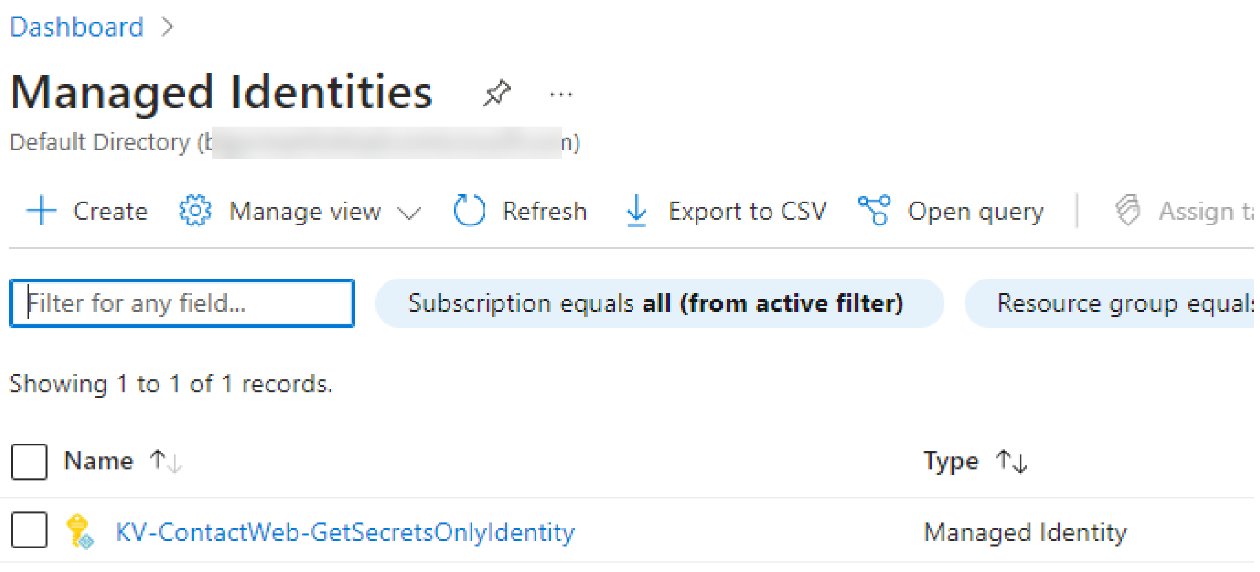
   Update the parameters file to include the new parameters:

```
        "developersGroupObjectId": {
            "value": "your-object-id-here"
        },
        "keyVaultUserManagedIdentityName": {
            "value": "GetSecretsOnlyIdentity"
        },
```

```
Make sure that you put your actual object id in the developersGroupObjectId
parameter.
```

1. Deploy the changes & validate

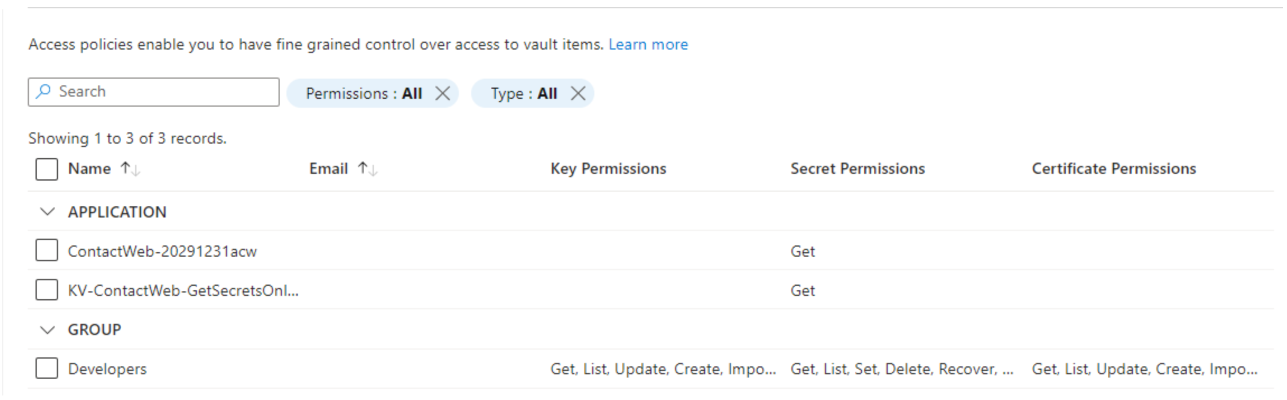
   Push the changes and validate that you have the new user managed identity and the permissions are correct.



Showing 1 to 1 of 1 records.

Name	Type
KV-ContactWeb-GetSecretsOnlyIdentity	Managed Identity

2. Validate Key Vault access permissions include the UMI, the web app, and your user group as expected.


Access policies enable you to have fine grained control over access to vault items. Learn more

Search | Permissions : All | Type : All

Showing 1 to 3 of 3 records.

Name	Email	Key Permissions	Secret Permissions	Certificate Permissions
APPLICATION				
ContactWeb-20291231acw			Get	
KV-ContactWeb-GetSecretsOnl...			Get	
GROUP				
Developers		Get, List, Update, Create, Impo...	Get, List, Set, Delete, Recover, ...	Get, List, Update, Create, Impo...

Task 2 - Create an App Configuration instance

In this task, you'll create an app configuration with references to the Key Vault secrets for the two database connections. You could do additional shared settings here as well that wouldn't have to be key vault secrets, but could just be setting key-value pairs. For this training, we're just doing the two database connections.

Step 1 - Create the bicep

In this step, you will create the bicep to create the App Configuration instance. The instance will need to make sure to have the following:

- App configuration with Identity
- Two KeyVault references to the Key Vault secrets
- Web App needs to have permission as `App Data Reader Role' to read the values from the App Configuration

1. Create the bicep file.

   Create a new file called `appConfigStore.bicep` and add the following:

```
param location string
@minLength(11)
@maxLength(11)
param uniqueIdentifier string
@minLength(5)
@maxLength(12)
param appConfigStoreName string
param identityDBConnectionStringKey string
param managerDBConnectionStringKey string
param identityDbSecretURI string
param managerDbSecretURI string
param keyVaultUserManagedIdentityName string
param webAppName string
param roleDefinitionName string
param appDataReaderRoleId string

var configName = '${appConfigStoreName}-${uniqueIdentifier}'
var roleAssignmentId = guid('${webApp.name}',
'${appDataReaderRole.name}','${appConfig.name}')
```

```
resource appDataReaderRole 'Microsoft.Authorization/roleDefinitions@2022-05-01-
preview' existing = {
  name: roleDefinitionName
}

resource webApp 'Microsoft.Web/sites@2023-01-01' existing = {
  name: webAppName
}

resource keyVaultUser 'Microsoft.ManagedIdentity/userAssignedIdentities@2023-01-
31' existing = {
  name: keyVaultUserManagedIdentityName
}

resource appConfig 'Microsoft.AppConfiguration/configurationStores@2023-03-01' = {
  name: configName
  location: location
  sku: {
    name: 'free'
  }
  identity: {
    type: 'UserAssigned'
    userAssignedIdentities: {
      '${keyVaultUser.id}': {}
    }
  }
  properties: {
    encryption: {}
    disableLocalAuth: false
    softDeleteRetentionInDays: 0
    enablePurgeProtection: false
  }
}

resource identityDBConnectionKeyValuePair
'Microsoft.AppConfiguration/configurationStores/keyValues@2023-03-01' = {
  name: identityDBConnectionStringKey
  parent: appConfig
  properties: {
    contentType:
'application/vnd.microsoft.appconfig.keyvaultref+json;charset=utf-8'
    value: '{"uri": "${identityDbSecretURI}"}'
  }
}

resource managerDBConnectionKeyValuePair
'Microsoft.AppConfiguration/configurationStores/keyValues@2023-03-01' = {
  name: managerDBConnectionStringKey
  parent: appConfig
  properties: {
    contentType:
'application/vnd.microsoft.appconfig.keyvaultref+json;charset=utf-8'
    value: '{"uri": "${managerDbSecretURI}"}'
  }
}

resource roleAssignment 'Microsoft.Authorization/roleAssignments@2022-04-01' = {
  name: roleAssignmentId
  scope: appConfig
  properties: {
    principalId: webApp.identity.principalId
    roleDefinitionId: appDataReaderRoleId
    principalType: 'ServicePrincipal'
  }
}

output appConfigStoreName string = appConfig.name
output appConfigStoreEndpoint string = appConfig.properties.endpoint
output dataReaderRoleName string = appDataReaderRole.name
output dataReaderRoleId string = appDataReaderRole.id
```

This allows you to create the App Configuration instance with the Key Vault references to the secrets. The App Configuration instance will have a user managed identity that will be used to access the Key Vault secrets. You will get the user managed identity from the name of the user managed identity that was created in the Key Vault bicep.

Step 2 - Create the parameters file

Add a new parameters file called `appConfigStore.parameters.json` and add the following:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "location": {
            "value": "eastus"
        },
        "uniqueIdentifier": {
            "value": "20291231acw"
        },
        "appConfigStoreName": {
            "value": "orgAppConfig"
        },
        "identityDBConnectionStringKey": {
            "value": "passedinfrommain"
        },
        "managerDBConnectionStringKey": {
            "value": "passedinfrommain"
        },
        "identityDbSecretURI": {
            "value": "passedinfrommain"
        },
        "managerDbSecretURI" : {
            "value": "passedinfrommain"
        },
        "webAppName" : {
            "value": "passedinfrommain"
        },
        "roleDefinitionName" : {
            "value": "App Configuration Data Reader"
        },
        "appDataReaderRoleId": {
            "value": "/providers/Microsoft.Authorization/roleDefinitions/516239f1-
63e1-4d78-a4de-a74fb236a071"
        }
    }
}
```

```
Note that most of the values are passed in for this to work correctly, so if you
wanted to run this separately you would need to find and configure the correct
values for those settings.
```

Step 3 - Modify the main deployment

In this step, you will modify the main deployment to include the App Configuration bicep.

1. Modify the main bicep to include a call to the App Configuration bicep

   Add the following to the params for the main deployment:

```
module orgAppConfiguration 'appConfigStore.bicep' = {
  name: '${appConfigStoreName}-deployment'
  scope: contactWebResourceGroup
  params: {
    location: contactWebResourceGroup.location
    uniqueIdentifier: uniqueIdentifier
    appConfigStoreName: appConfigStoreName
    identityDBConnectionStringKey: identityDBConnectionStringKey
    managerDBConnectionStringKey: managerDBConnectionStringKey
    identityDbSecretURI: contactWebVault.outputs.identityDBConnectionSecretURI
    managerDbSecretURI: contactWebVault.outputs.managerDBConnectionSecretURI
    keyVaultUserManagedIdentityName:
contactWebVault.outputs.keyVaultUserManagedIdentityName
    webAppName: contactWebApplicationPlanAndSite.outputs.webAppFullName
    roleDefinitionName: appDataReaderRoleDefinitionName
    appDataReaderRoleId: appDataReaderRoleId
  }
}
```

```
Notice a number of the parameters are passed in from the Key Vault bicep.
Additionally, the configuration setting keys are necessary for the app
```

```
configuration to be able to get to the key vault secrets.  The only real new
variables here are the name of the app configuration and the name of the `App
Configuration Data Reader` role.

Don't forget to add the new parameters to the main deployment parameters file.
```

```
@minLength(5)
@maxLength(12)
param appConfigStoreName string
param appDataReaderRoleDefinitionName string
param appDataReaderRoleId string
```

```
Add the following to the parameters file:
```

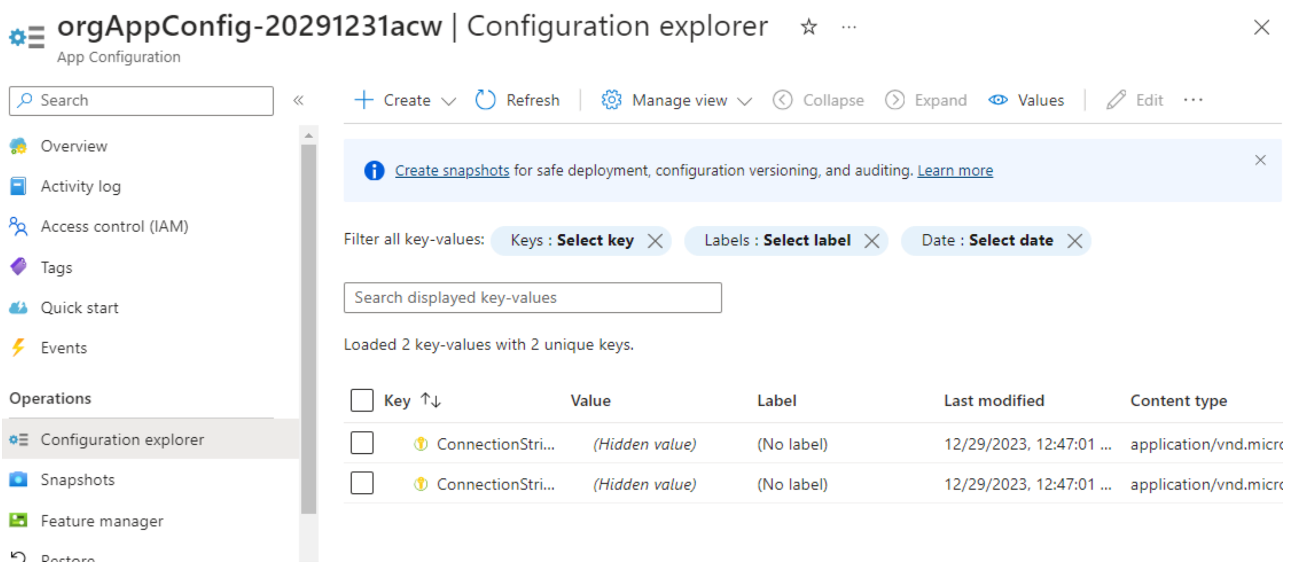
```
        "appConfigStoreName": {
            "value": "orgAppConfig"
        },
        "appDataReaderRoleDefinitionName": {
            "value": "App Configuration Data Reader"
        },
        "appDataReaderRoleId": {
            "value": "/providers/Microsoft.Authorization/roleDefinitions/516239f1-63e1-4d78-a4de-a74fb236a071"
        }
```

```
This will be the name of the App Configuration instance.
```

Step 4 - Deploy and Validate

Push your changes and validate the deployment.

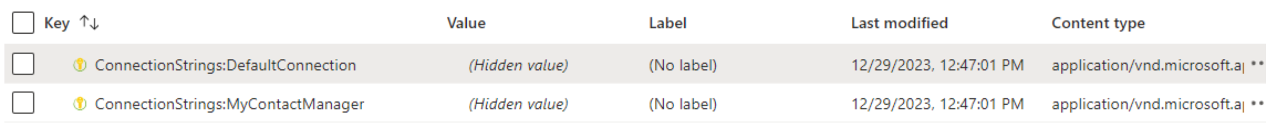
1. Validate the app configuration is created



Loaded 2 key-values with 2 unique keys.

Key	Value	Label	Last modified	Content type
ConnectionStri...	*(Hidden value)*	(No label)	12/29/2023, 12:47:01 ...	application/vnd.micr
ConnectionStri...	*(Hidden value)*	(No label)	12/29/2023, 12:47:01 ...	application/vnd.micr

2. Validate the two secrets are in place on the Configuration Explorer



Key	Value	Label	Last modified	Content type
ConnectionStrings:DefaultConnection	*(Hidden value)*	(No label)	12/29/2023, 12:47:01 PM	application/vnd.microsoft.a
ConnectionStrings:MyContactManager	*(Hidden value)*	(No label)	12/29/2023, 12:47:01 PM	application/vnd.microsoft.a

Task 3 - Reset the App Service to use the App Configuration and no longer directly reference the Key Vault

There are a number of things that you could do at this point. One of them is that you could go back and remove the update/merge from previous work and you could just add the app config settings from the start of the creation of the web app. As this is an organic feel of moving through iterations, and for simplicity, I've chosen to just create another settings update. This time, you will reset the App configuration settings to only have the Application Insights connection string and the new endpoint URL from the previous step for the App Configuration.

Step 1 - Create the reset bicep

The settings will not let you pass an empty object, so for this you will create a new file for reset and you will make the current settings with only the application insights connection string, and then you'll merge that with the new setting for the app configuration endpoint.

1. Create a new file called `contactWebAppServiceSettingsResetForAppConfiguration.bicep` and add the following:

```
param webAppName string
param appConfigurationEndpointKey string
param appConfigurationEndpointValue string
param applicationInsightsConnectionStringKey string
param applicationInsightsName string

resource webApp 'Microsoft.Web/sites@2023-01-01' existing = {
  name: webAppName
}
```

```
resource appInsights 'Microsoft.Insights/components@2020-02-02-preview' existing =
{
  name: applicationInsightsName
}

//reset the current app settings with the new app settings WARNING: Wipes out
everything that isn't explicitly set in the appSettings object
module resetAppConfigurationSettings 'contactWebAppServiceSettingsMerge.bicep' = {
  name: 'webAppSettings-${webAppName}'
  params: {
    currentAppSettings: { '${applicationInsightsConnectionStringKey}':
appInsights.properties.ConnectionString }
    appSettings: {
      '${appConfigurationEndpointKey}' : appConfigurationEndpointValue
    }
    webAppName: webApp.name
  }
}
```

The file needs the web app and the app insights to get the objects, then merges values with the keys as appropriate, additionally, the new app configuration endpoint value must be passed in.

> **Note**: For this step I've chosen not to include a parameters file. You could do that, but it's not necessary for this step.

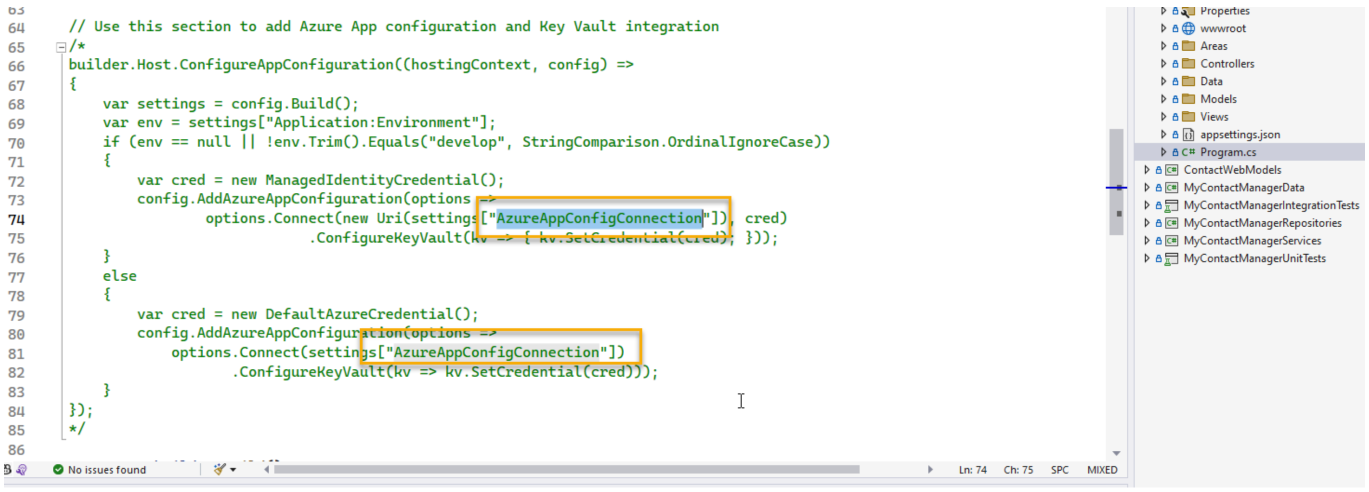
Step 2 - Add to the main deployment

Add the reset task to the main deployment.

1. Add the following to the main deployment:

```
module resetContactWebAppSettingsForAppConfiguration
'contactWebAppServiceSettingsResetForAppConfiguration.bicep' = {
  name: '${webAppName}-resettingAppSettingsForAppConfiguration'
  scope: contactWebResourceGroup
  params: {
    webAppName: contactWebApplicationPlanAndSite.outputs.webAppFullName
    appConfigurationEndpointKey: appConfigurationEndpointKey
    appConfigurationEndpointValue:
orgAppConfiguration.outputs.appConfigStoreEndpoint
    applicationInsightsConnectionStringKey: appInsightsConnectionStringKey
    applicationInsightsName:
contactWebApplicationInsights.outputs.applicationInsightsName
  }
}
```

Note the new parameter for the application insights connection string key. This will map directly to the value the code is expecting to find `AzureAppConfigConnection`:



```
// Use this section to add Azure App configuration and Key Vault integration
/*
builder.Host.ConfigureAppConfiguration((hostingContext, config) =>
{
    var settings = config.Build();
    var env = settings["Application:Environment"];
    if (env == null || !env.Trim().Equals("develop", StringComparison.OrdinalIgnoreCase))
    {
        var cred = new ManagedIdentityCredential();
        config.AddAzureAppConfiguration(options =>
                options.Connect(new Uri(settings["AzureAppConfigConnection"]), cred)
                        .ConfigureKeyVault(kv => { kv.SetCredential(cred); }));
    }
    else
    {
        var cred = new DefaultAzureCredential();
        config.AddAzureAppConfiguration(options =>
            options.Connect(settings["AzureAppConfigConnection"])
                    .ConfigureKeyVault(kv => kv.SetCredential(cred)));
    }
});
*/
```

> **Note:** Running the code locally requires the app config connection string and a modification to the app settings. The App Service, however, can run from Azure with these settings in place.

1. Add the new parameter in the main bicep

```
@minLength(5)
@maxLength(12)
param appConfigStoreName string
param appConfigurationEndpointKey string
```

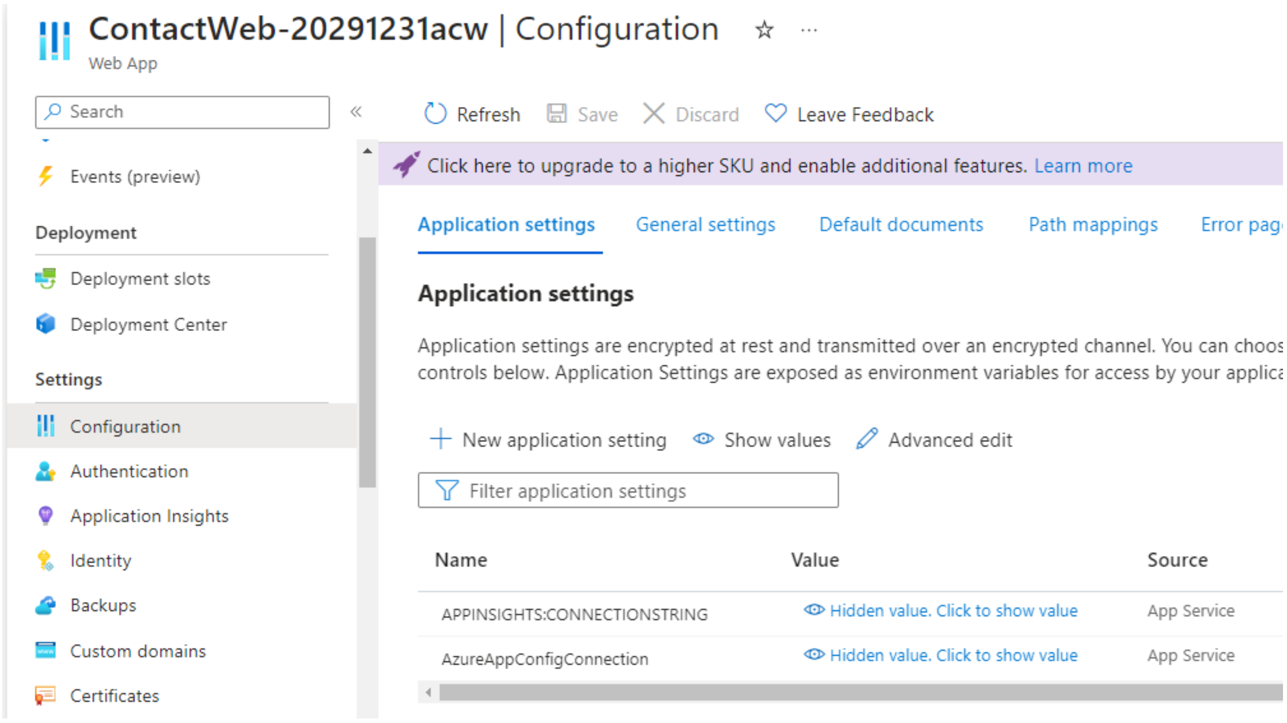
1. Add the parameters to the main deployment parameters file

```
        "appConfigurationEndpointKey": {
            "value": "AzureAppConfigConnection"
        }
```

Step 3 - Deploy and Validate

Push your changes and validate the deployment.

1. The App Service Settings should now only have the Application Insights connection string and the App Configuration endpoint.



ContactWeb-20291231acw | Configuration

Web App

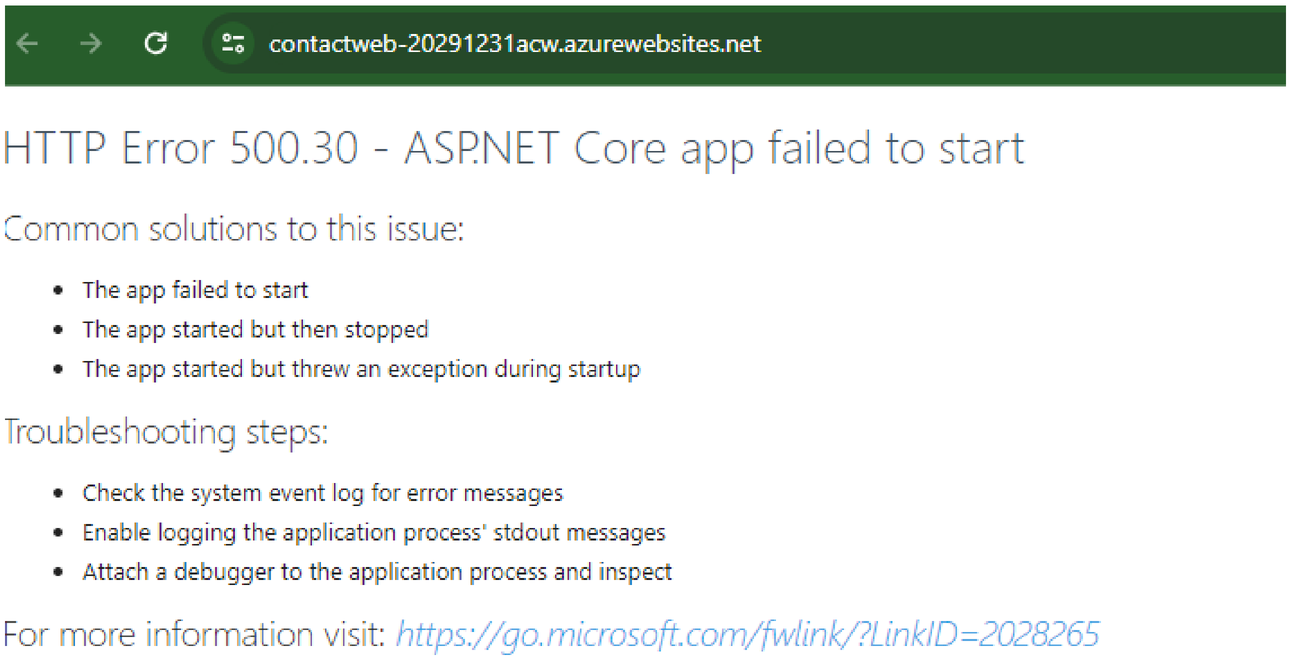
Click here to upgrade to a higher SKU and enable additional features. Learn more

Application settings

Application settings are encrypted at rest and transmitted over an encrypted channel. You can choos
controls below. Application Settings are exposed as environment variables for access by your applica

Name	Value	Source
APPINSIGHTS:CONNECTIONSTRING	Hidden value. Click to show value	App Service
AzureAppConfigConnection	Hidden value. Click to show value	App Service

> **Note**: the Application gets a 500.3 error. This is because the code needs to be modified to leverage app Configuration. We'll do that in the next Task.



contactweb-20291231acw.azurewebsites.net

HTTP Error 500.30 - ASP.NET Core app failed to start

Common solutions to this issue:

- The app failed to start
- The app started but then stopped
- The app started but threw an exception during startup

Troubleshooting steps:

- Check the system event log for error messages
- Enable logging the application process' stdout messages
- Attach a debugger to the application process and inspect

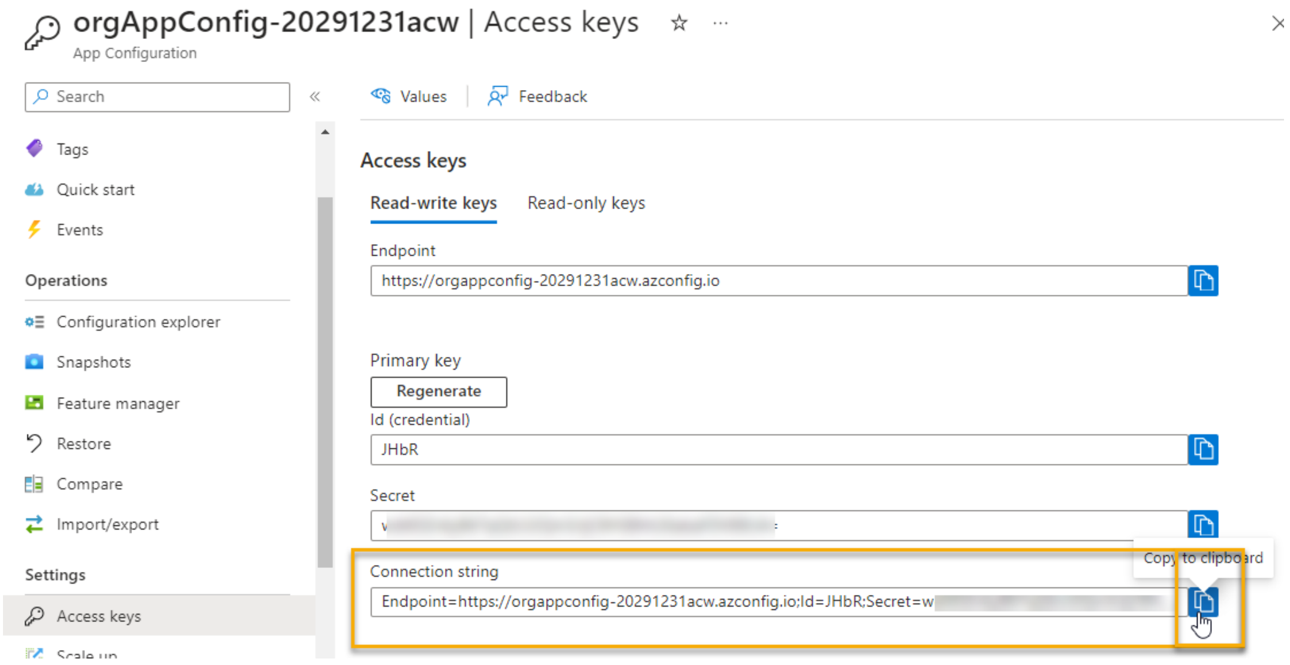
For more information visit: https://go.microsoft.com/fwlink/?LinkID=2028265

Task 4 - Modify the code to leverage the App Configuration

In this task, you'll modify the solution code to work against the App Configuration instead of the Key Vault directly.

1. Get the app working locally (optional)

   You can get the app working locally by adding the app configuration connection string to your local settings. You can get the connection string from the App Configuration instance in the portal.



orgAppConfig-20291231acw | Access keys

App Configuration

Access keys

Read-write keys | Read-only keys

Endpoint: https://orgappconfig-20291231acw.azconfig.io

Primary key

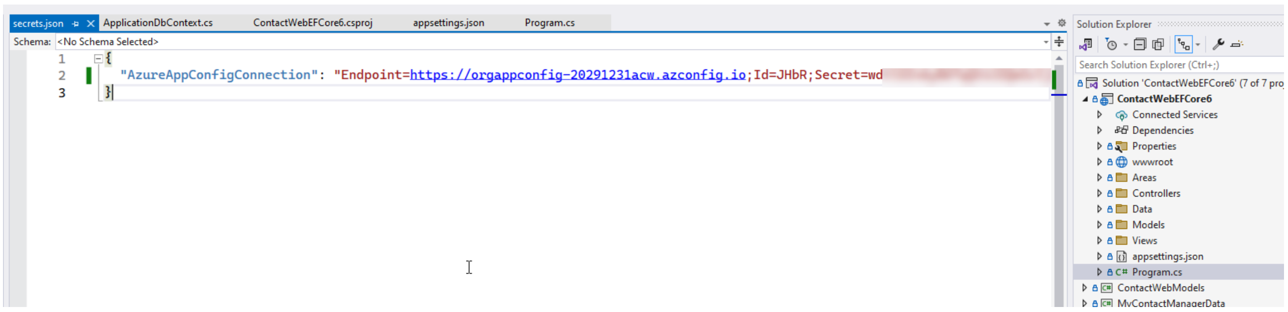
Regenerate

Id (credential): JHbR

Secret: v[redacted]

Connection string: Endpoint=https://orgappconfig-20291231acw.azconfig.io;Id=JHbR;Secret=w[redacted]

Add the connection string to your local settings and run the app locally. You should be able to see the app working locally.

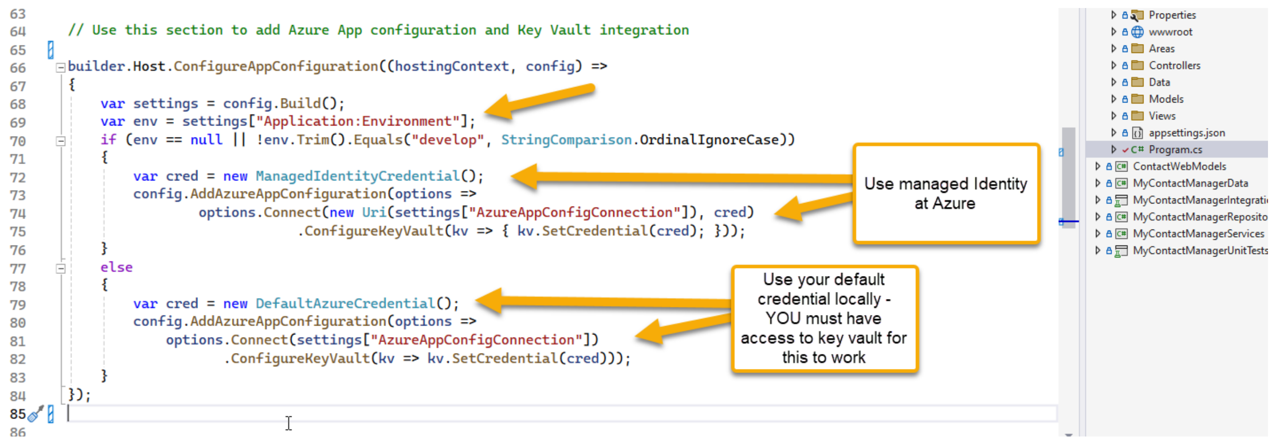


```
{
  "AzureAppConfigConnection": "Endpoint=https://orgappconfig-20291231acw.azconfig.io;Id=JHbR;Secret=wd[redacted]
}
```

Step 1 - Modify the code

1. Modify the code to leverage the App Configuration

   Remove the comments to allow the code in lines 66-84 to execute.



```
63
64    // Use this section to add Azure App configuration and Key Vault integration
65
66  builder.Host.ConfigureAppConfiguration((hostingContext, config) =>
67  {
68      var settings = config.Build();
69      var env = settings["Application:Environment"];
70      if (env == null || !env.Trim().Equals("develop", StringComparison.OrdinalIgnoreCase))
71      {
72          var cred = new ManagedIdentityCredential();
73          config.AddAzureAppConfiguration(options =>
74                  options.Connect(new Uri(settings["AzureAppConfigConnection"]), cred)
75                              .ConfigureKeyVault(kv => { kv.SetCredential(cred); }));
76      }
77      else
78      {
79          var cred = new DefaultAzureCredential();
80          config.AddAzureAppConfiguration(options =>
81              options.Connect(settings["AzureAppConfigConnection"])
82                      .ConfigureKeyVault(kv => kv.SetCredential(cred)));
83      }
84  });
85
86
```

Use managed Identity at Azure
Use your default credential locally - YOU must have access to key vault for this to work
Properties
wwwroot
Areas
Controllers
Data
Models
Views
appsettings.json
Program.cs
ContactWebModels
MyContactManagerData
MyContactManagerIntegrati
MyContactManagerReposito
MyContactManagerServices
MyContactManagerUnitTests

> **Note**: The code is now using the App Configuration to get the connection string for the database. In order for this to work locally, you will need to have access to key vault (even if the app config does, you don't from your local machine).

Notice at Azure, the `ManagedIdentityCredential` is used to allow authorization to resources like Key Vault.

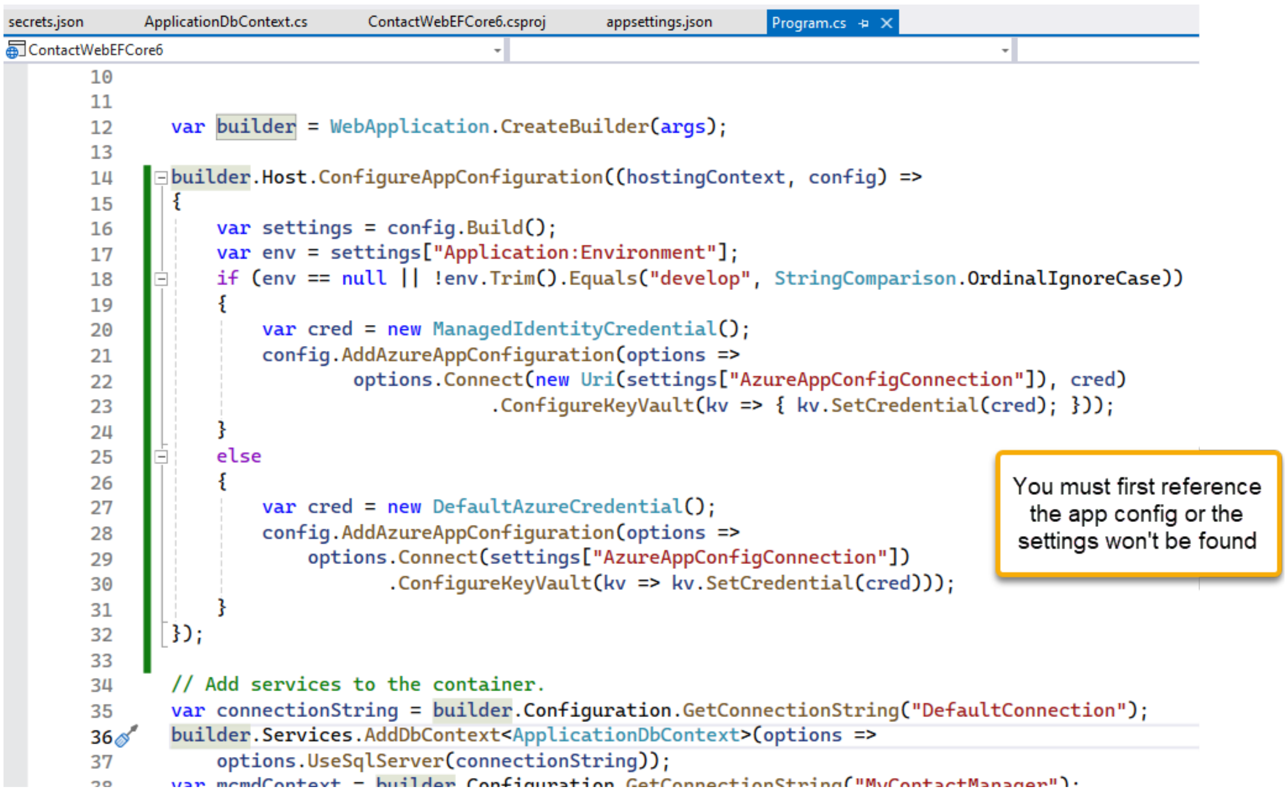
To toggle the `DefaultAzureCredential` for your local machine (you don't have a managed identity), add a setting to your local settings called `Application:Environment` and set it to `develop`. This will allow you to use the `DefaultAzureCredential` to get the connection string from the Key Vault.

2. Move the code to be created prior to trying to leverage the secrets

   Cut the code and move it to the top of the file, right after line 12's `var builder = WebApplication.CreateBuilder(args);` and before the line:

   `var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");`

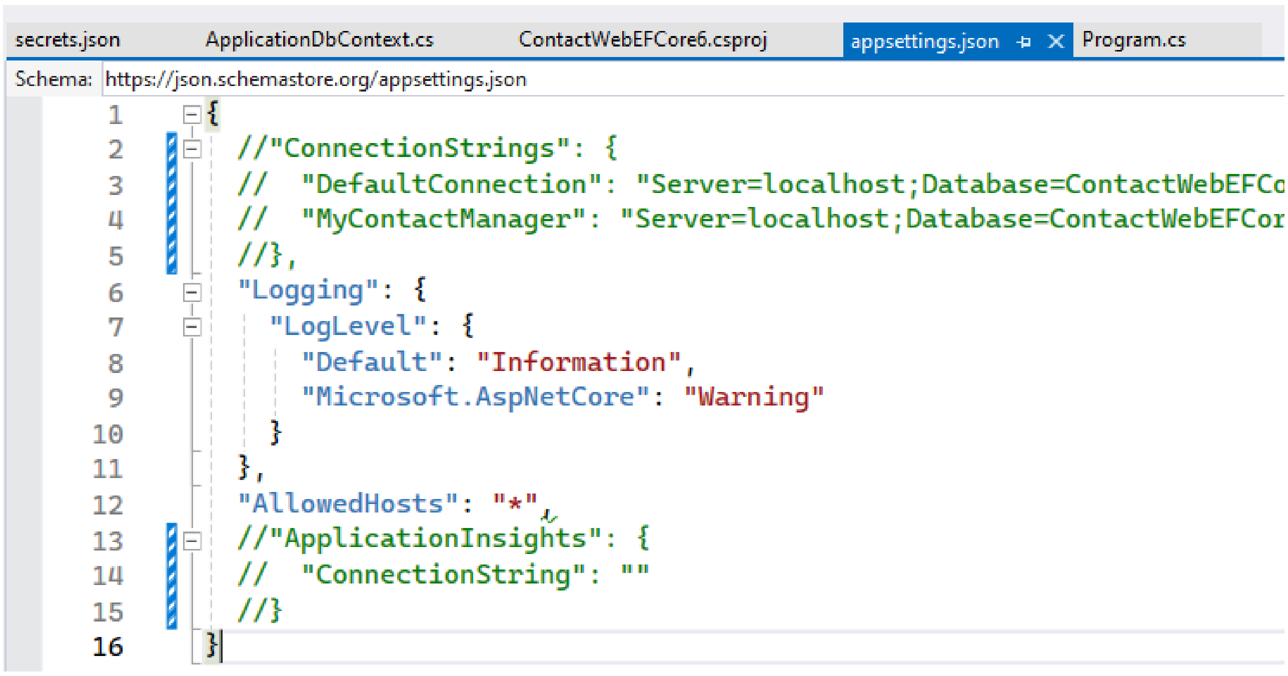
   If you don't do this, you will get an error because the configuration is not yet set up and the connection string will be null.



```
10
11
12  var builder = WebApplication.CreateBuilder(args);
13
14  builder.Host.ConfigureAppConfiguration((hostingContext, config) =>
15  {
16      var settings = config.Build();
17      var env = settings["Application:Environment"];
18      if (env == null || !env.Trim().Equals("develop", StringComparison.OrdinalIgnoreCase))
19      {
20          var cred = new ManagedIdentityCredential();
21          config.AddAzureAppConfiguration(options =>
22                  options.Connect(new Uri(settings["AzureAppConfigConnection"]), cred)
23                          .ConfigureKeyVault(kv => { kv.SetCredential(cred); }));
24      }
25      else
26      {
27          var cred = new DefaultAzureCredential();
28          config.AddAzureAppConfiguration(options =>
29              options.Connect(settings["AzureAppConfigConnection"])
30                      .ConfigureKeyVault(kv => kv.SetCredential(cred)));
31      }
32  });
33
34  // Add services to the container.
35  var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
36  builder.Services.AddDbContext<ApplicationDbContext>(options =>
37      options.UseSqlServer(connectionString));
38  var mcmdContext = builder.Configuration.GetConnectionString("MyContactManager");
```

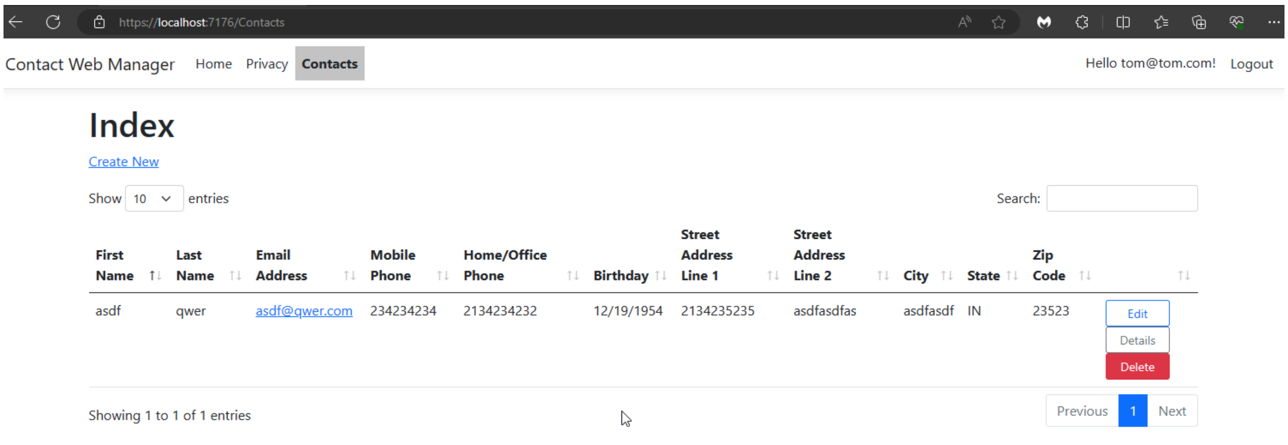
You must first reference the app config or the settings won't be found

3. Comment out any references to connection strings in your `appsettings.json`.



```
{
  //"ConnectionStrings": {
  //  "DefaultConnection": "Server=localhost;Database=ContactWebEFCo
  //  "MyContactManager": "Server=localhost;Database=ContactWebEFCor
  //},
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  //"ApplicationInsights": {
  //  "ConnectionString": ""
  //}
}
```

4. Run the application locally (it will be hitting your azure db so if you don't have ip allowed on the db you may also get an error for the database).



Index

Create New

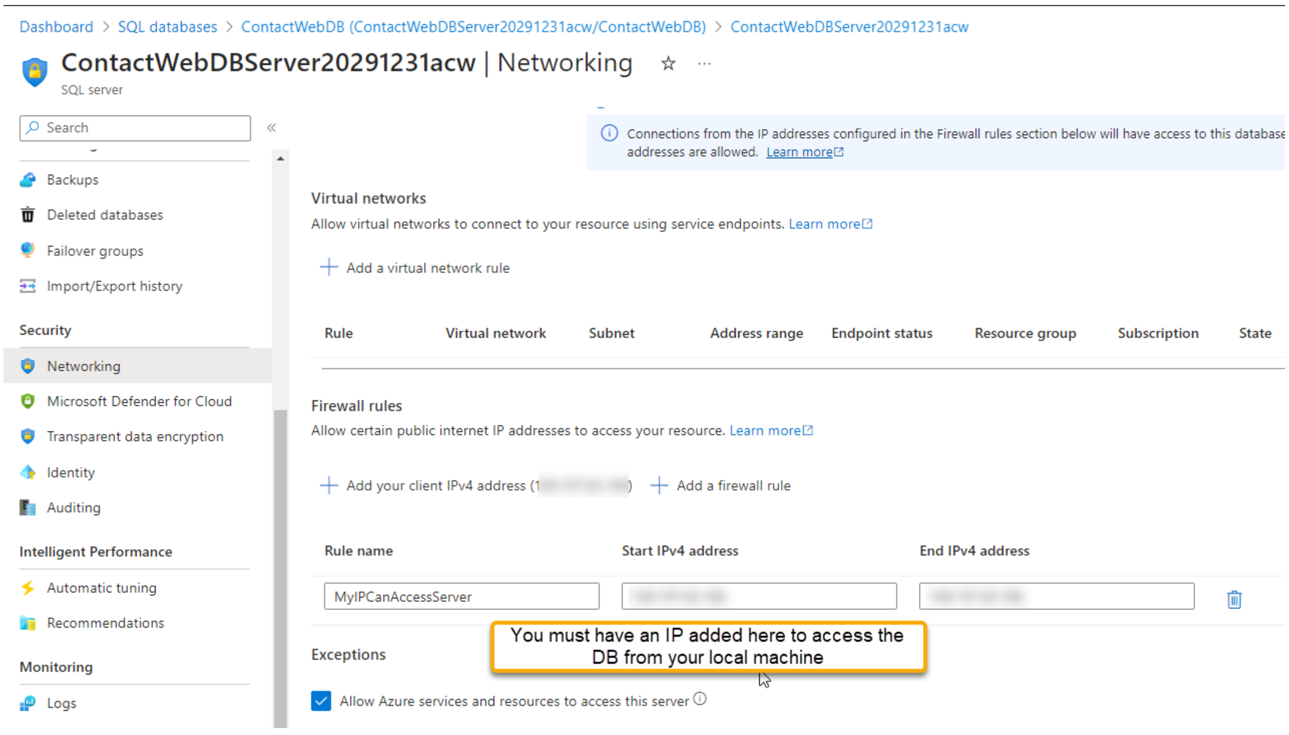
Show 10 entries Search:

First Name	Last Name	Email Address	Mobile Phone	Home/Office Phone	Birthday	Street Address Line 1	Street Address Line 2	City	State	Zip Code	
asdf	qwer	asdf@qwer.com	234234234	2134234232	12/19/1954	2134235235	asdfasdfas	asdfasdf	IN	23523	Edit Details Delete

Showing 1 to 1 of 1 entries

Previous 1 Next

Run the application locally and validate that it works. If you get a 500.3 error, make sure you check the database firewall settings and add your local IP.


Dashboard > SQL databases > ContactWebDB (ContactWebDBServer20291231acw/ContactWebDB) > ContactWebDBServer20291231acw

ContactWebDBServer20291231acw | Networking

SQL server

Connections from the IP addresses configured in the Firewall rules section below will have access to this database ... addresses are allowed. Learn more

Virtual networks

Allow virtual networks to connect to your resource using service endpoints. Learn more

+ Add a virtual network rule

Rule	Virtual network	Subnet	Address range	Endpoint status	Resource group	Subscription	State

Firewall rules

Allow certain public internet IP addresses to access your resource. Learn more

+ Add your client IPv4 address ([redacted]) + Add a firewall rule

Rule name	Start IPv4 address	End IPv4 address
MyIPCanAccessServer	[redacted]	[redacted]

You must have an IP added here to access the DB from your local machine

Exceptions

☑ Allow Azure services and resources to access this server

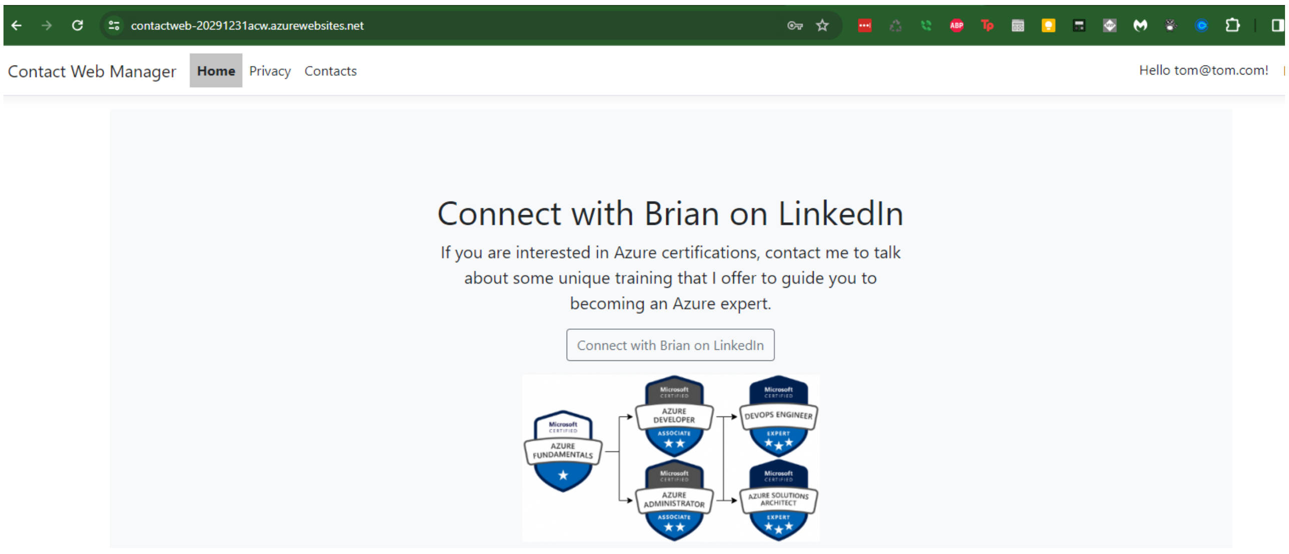
Step 2 - Deploy and Validate

Push your changes and validate the deployment.

1. Make sure you have updated the application code as expected and push your changes to the repo.

   This will run the final deployment where all the architecture and your application are in place.

2. Validate the application is working as expected.


contactweb-20291231acw.azurewebsites.net

Contact Web Manager
Home
Privacy
Contacts
Hello tom@tom.com!

Connect with Brian on LinkedIn

If you are interested in Azure certifications, contact me to talk about some unique training that I offer to guide you to becoming an Azure expert.

Connect with Brian on LinkedIn
AZURE FUNDAMENTALS
AZURE DEVELOPER ASSOCIATE
AZURE ADMINISTRATOR ASSOCIATE
DEVOPS ENGINEER EXPERT
AZURE SOLUTIONS ARCHITECT EXPERT

Completion

You have now completed the activities for this cloud workshop. We hope you enjoyed it and learned a lot during our training. Please feel free to reach out to us with any questions or feedback. If you found any problems, please open issues on the repository.

We'd love to hear from you!