aws

User Guide

AWS Toolkit for .NET Refactoring

AWS Toolkit for .NET Refactoring: User Guide

Table of Contents

AWS .NET Modernization Tools Porting Assistant (PA) for .NET, AWS App2Container (A2C), AWS Toolkit for .NET Refactoring (TR), and AWS Microservice Extractor (ME) for .NET is no longer open to new customers. If you would like to use the service, sign up prior to November 7, 2025. Alternatively use AWS Transform, which is an agentic AI service developed to accelerate enterprise modernization of .NET.

What is AWS Toolkit for .NET Refactoring?

AWS Toolkit for .NET Refactoring is an extension for Microsoft Visual Studio that reduces the time and effort that is required to refactor legacy .NET applications to alternatives on AWS Cloud.

Toolkit for .NET Refactoring assesses the application source code and recommends modernization pathways, such as porting to .NET Core. It also identifies Microsoft Windows dependencies on Microsoft Internet Information Services (IIS) and Microsoft Active Directory (AD), performs code modifications where possible, and assists in validating the refactored application on AWS services. Using the Toolkit for .NET Refactoring Visual Studio extension, you can perform all of these tasks within the Visual Studio integrated development environment (IDE).

Topics

Features of Toolkit for .NET Refactoring

The Toolkit for .NET Refactoring extension provides compatibility assessments, porting assistance, and testing on AWS.

Compatibility assessment

Before you update your source code, Toolkit for .NET Refactoring performs an assessment of your .NET application source code and packages, such as NuGet and Microsoft Core. The extension determines compatibility based on whether a project can move to .NET Core runtime without code changes or package upgrades. The extension assesses .NET Core compatibility for solutions, projects, NuGet packages, and source code.

The compatibility assessment identifies the following:

- Microsoft Windows dependencies on Microsoft Internet Information Services (IIS) and Microsoft Active Directory (AD).
- API and package incompatibilities with newer cross-platform .NET versions, such as .NET Core 3.1, .NET 5, and .NET 6.

Toolkit for .NET Refactoring scans third-party and internal packages to classify them as compatible or incompatible. For each incompatible package, Toolkit for .NET Refactoring provides replacement options, if they are available.

Porting assistance

When the compatibility assessment is complete, the Toolkit for .NET Refactoring extension provides porting assistance by suggesting code changes to remove incompatibilities that it found in the assessment. If the latest version of a package is compatible with .NET Core, the extension upgrades the package to its latest compatible version and updates the relevant project reference files and `web.config` files to a format that is compatible with .NET Core. Although the extension doesn't eliminate the need for manual source code changes, it reduces the initial effort that is required to refactor the source code.

Testing on AWS

As you refactor your source code, you can use the Toolkit for .NET Refactoring extension to test and validate the code. To validate the code, you can deploy directly from Visual Studio to Amazon Elastic Container Service (Amazon ECS), hosted on AWS Fargate.

If the solution is a web application that has been ported to .NET Core, the Toolkit for .NET Refactoring extension provides the ability to test the application by running the application in the AWS Cloud. To do this, you might need to set up the necessary resources in the AWS account and build the artifact with the .NET `publish` command or rebuild the solution. You can use the Toolkit for .NET Refactoring extension to upload the artifact into your AWS account and run it inside an Amazon ECS Linux container instance to verify that the solution is fully compatible in a Linux environment.

Concepts

An understanding of the terminology and concepts below is necessary to make full use of the Toolkit for .NET Refactoring extension.

Amazon ECS

Amazon Elastic Container Service (Amazon ECS) is a highly scalable and fast container management service. It is an abstraction of a container runtime environment that is analogous to a Kubernetes Pod, but with Amazon ECS, you don't need to worry about the complexity

of Kubernetes. Within Amazon ECS, containers are grouped in a task. Within a task, multiple containers can run and share resources such as disk volumes or networks. Tasks are grouped in a service, which is an abstraction of a scalable multi-worker application. For more information, see What is Amazon Elastic Container Service?

IAM roles, permissions, and delegations

To access an AWS resource, a user must have permissions to access the resource. Permissions are grouped into IAM roles. IAM roles have restrictions regarding which users can use the roles. These restrictions can be included in the trust relationship of the role or as a permission assigned to a specific user.

AWS services also require permissions to perform operations.

- The service can act on the behalf of the user that is accessing the resource. In this case, the service uses the permissions of the user.
- You can specify an IAM role for the service and the service will assume the role to perform the operations.

For more information about roles, see Using IAM roles in the *IAM User Guide*.

AWS Managed Microsoft AD

AWS Directory Service for Microsoft Active Directory (AWS Managed Microsoft AD) runs on a Windows host, where the host and Active Directory are managed by AWS. An Active Directory administrator can use Active Directory management tools from a Windows instance that is joined to the Active Directory.

The Active Directory acts as a Domain Name System (DNS) for the nodes that are joined to the domain. Usually, a Dynamic Host Configuration Protocol (DHCP) server within the network is configured to return the IP of the Active Directory as a DNS server for the network.

Sidecar container

A sidecar container is a container that runs on the same node, which is the same Amazon ECS task, as the main application container. The sidecar container performs auxiliary tasks such as log collection, authentication, authorization, and network routing. Using a sidecar container is preferable to running an agent process in the same container in the application. A sidecar container

allows the auxiliary processes to run in isolation and have an independent compute environment, including installed libraries, packages, and operating system.

The Amazon ECS on Fargate task will run the following two containers:

- The application container.
- The sidecar container that is used for Toolkit for .NET Refactoring technical tasks, such as file sync and Active Directory authentication.

Microsoft Active Directory authentication in Linux containers

Microsoft Active Directory (AD) can act as Kerberos server. Linux Kerberos utilities are used to obtain an authentication token for the application that is ported to .NET Core. The application uses this token to authenticate itself to the dependent services.

AWS .NET Modernization Tools availability change

Porting Assistant (PA) for .NET, AWS App2Container (A2C), AWS Toolkit for .NET Refactoring (TR), and AWS Microservice Extractor (ME) for .NET will stop accepting new customers starting November 7, 2025. AWS Transform, launched in 2025, is our next-generation service that provides enhanced capabilities for modernization with AI-driven automation. Existing Porting Assistant (PA) for .NET, AWS App2Container (A2C), AWS Toolkit for .NET Refactoring (TR), and AWS Microservice Extractor (ME) for .NET customers can continue using the service to complete their ongoing modernization projects. All current .NET Modernization Tools features, including Porting Assistant for .NET, AWS App2Container, AWS Toolkit for .NET Refactoring, and AWS Microservice Extractor for .NET, are available in AWS Transform with improved functionality through secure Managed Development Environments and LLM integration.

While we will not be adding new features to the service, we remain committed to providing security updates and maintaining service availability to ensure your ongoing migration projects continue to run smoothly. Our focus is on ensuring a stable environment for existing customers to complete their in-flight projects while preparing for transitioning to the enhanced capabilities available in AWS Transform.

AWS Transform for .NET accelerates large-scale modernization from .NET Framework to cross-platform .NET by up to 4x. With the .NET modernization agent, modernization teams can collaboratively execute larger and more complex projects with consistency, remove Windows license dependencies to reduce operating costs by up to 40%, and enhance code quality, performance, and security.

Existing users can complete their current projects using existing tools, while all new modernization projects can start in AWS Transform. To begin using AWS Transform, see the Getting Started Guide. No data migration or special transition tools are needed, as users can organically switch to AWS Transform for any new work.

If you have additional questions, contact AWS Support or read our FAQs:

- **What does this mean for the service (will it be shut down)?**

  Porting Assistant (PA) for .NET, AWS App2Container (A2C), AWS Toolkit for .NET Refactoring (TR), and AWS Microservice Extractor (ME) for .NET will stop accepting new customers starting November 7, 2025. The service will continue to operate for existing customers to complete their ongoing migration projects.

- **How will existing customers be impacted?**

  Existing customers can continue using the service until their in-flight modernization projects are completed. All ongoing projects will remain accessible. No disruption to existing workloads or in-flight modernizations.
- **On November 7, 2025, how can I get help if I have issues?**

  Customers experiencing issues can contact AWS Support.
- **What alternatives can I explore?**

  AWS Transform is the recommended alternative. With the .NET modernization agent, modernization teams can collaboratively execute larger and more complex projects with consistency, remove Windows license dependencies to reduce operating costs by up to 40%, and enhance code quality, performance, and security.
- **How do I migrate off of .NET Modernization Tools?**

  No formal migration process is required. Existing projects can continue in Porting Assistant (PA) for .NET, AWS App2Container (A2C), AWS Toolkit for .NET Refactoring (TR), or AWS Microservice Extractor (ME) for .NET until completion. For new projects, you can start directly in AWS Transform, which provides all the familiar capabilities of .NET Modernization Tools with enhanced features. No data migration is needed, and AWS Support is available to assist with the transition.

Get started with Toolkit for .NET Refactoring

The sections below contain information to help you set up your Toolkit for .NET Refactoring environment and to start assessing, converting, and porting your .NET Framework application to .NET Core.

Download the Toolkit for .NET Refactoring extension from the Microsoft Visual Studio Extensions Marketplace. With the extension, you can run a compatibility assessment on any solution in your portfolio.

Topics

Prerequisites

The sections below describe the prerequisites that you must verify before you run an assessment with Toolkit for .NET Refactoring. Additional prerequisites are required for testing your application on AWS. Those prerequisites are detailed in the Setting up for testing on AWS section of this guide.

Topics

Supported versions

Toolkit for .NET Refactoring supports the following .NET versions.

- Source versions:
  - .NET Framework 3.5 and later

- Target versions:
  - .NET Core 3.1
  - .NET 5.0
  - .NET 6.0 on Microsoft Visual Studio 2022
- The AWS test environment supports the following runtime versions:
  - .NET Core 3.1
  - .NET 5.0
  - .NET 6.0

For compatibility assessments and porting, Windows services and ASP.NET applications are supported.

The following Microsoft Visual Studio versions are supported:

- Visual Studio 2019
- Visual Studio 2022

General prerequisites

Before you use the Toolkit for .NET Refactoring extension, verify the following prerequisites:

- You must have either a local machine or an Amazon EC2 instance where you will install AWS CLI, Microsoft Visual Studio, the .NET version that you want to use, and Toolkit for .NET Refactoring. For information about using a local machine versus an Amazon EC2 instance, see AWS Identity and Access Management (IAM) overview in this guide.
- One of the following .NET versions must be installed on the local machine or Amazon EC2 instance:
  - .NET 6.0
  - .NET 5.0
  - .NET Core 3.1

  To download and install, see Download .NET on the Microsoft website.
- AWS CLI must be installed on the local machine or Amazon EC2 instance. Toolkit for .NET Refactoring uses information from your AWS CLI profile to determine compatibility between:
  - Public NuGet packages

- The APIs within the NuGet packages that your application uses
- The AWS test environment

  The extension requires a valid AWS Command Line Interface (AWS CLI) profile to collect compatibility information on the public NuGet packages and the APIs within the packages that your application uses. For more information about the application data that Toolkit for .NET Refactoring collects, see Data collected in this guide.

  For information about installing the AWS CLI, see Installing or updating the latest version of the AWS CLI in the *AWS CLI User Guide*. For information about how to configure an AWS CLI profile, see Configuring the AWS CLI in the *AWS CLI User Guide*.
- Microsoft Visual Studio 2019 version 16.9 or later, or Visual Studio 2022 must be installed.

Memory requirements

Use the table below to determine how much memory is required for the Toolkit for .NET Refactoring extension to run based on the solution size.

Solution size	Minimum memory
Small solutions (1,000 to 50,000 lines of code)	4 GB
Medium solutions (50,000 to 400,000 lines of code)	8 GB
Large solutions (400,000 or more lines of code)	16 GB or more, depending on the size of the source code

Automatic setup

You can automatically perform the prerequisite steps to use Toolkit for .NET Refactoring. You can perform the setup entirely from within the tool or by using a script. Administrative or similar privileges are required for automatic setup.

Topics

- Start the automatic setup process

- In-tool setup
- Script setup

Start the automatic setup process

To install Toolkit for .NET Refactoring to perform the setup, see Install Toolkit for .NET Refactoring. From the **Get Started** page, following the credential selection, select **Prerequisite Setup**. You can perform the setup for your currently selected profile and the AWS account to which it belongs. Your selected credentials can be updated.

In-tool setup

You can perform the setup from entirely within the tool. To do so, choose **Run Setup**. The tool will make all of the required calls using your selected credentials. The response to these calls will be displayed in the output window on the right. If your selected credentials are invalid, we recommend that you try the script setup.

Script setup

Alternatively, you can perform the setup using a script. Choose **Download** to generate a .ps1 script. Run the script to perform the setup. If the user to set up has insufficient permissions, an administrator can run the generated script. Verify the contents of the script before you run it to ensure that it has not been altered.

Setting up for testing on AWS

The sections below contain detailed descriptions of the configuration, roles, and permissions that are required to run a test deployment on AWS with Toolkit for .NET Refactoring.

Topics

- Prerequisites for testing on AWS
- AWS roles and managed policies for Toolkit for .NET Refactoring test deployment
- Active Directory setup

Prerequisites for testing on AWS

Verify the prerequisites below before you test your application on AWS.

Topics

S3 bucket

You must have an S3 bucket in the AWS Region where you want to run the test deployment. You can create a S3 bucket using Amazon Simple Storage Service (Amazon S3). Toolkit for .NET Refactoring supports the following Regions:

- US East (Ohio) – `us-east-2`
- Europe (London) – `eu-west-2`

Toolkit for .NET Refactoring uses the S3 bucket to do the following:

- Prepare the data for the container image build.
- Transfer the application files to the Amazon ECS task.

VPC requirements

When you create a test deployment, the default setting is to create a new virtual private cloud (VPC) in your AWS account. For more information, see Virtual private clouds (VPC) in the *Amazon Virtual Private Cloud User Guide*.

Alternatively, you can choose to use an existing VPC. If you use an existing VPC, it must satisfy the following requirements:

- It must have two public subnets in different Availability Zones. For information about Availability Zones, see Regions and Zones in the *Amazon Elastic Compute Cloud User Guide.*
- It must have an internet gateway. For information about internet gateways, see Connect to the internet using an internet gateway in the *Amazon VPC User Guide*.

It must have a routing table that connects the internet gateway to the subnets. This means that the subnet is public. For information about subnets, see VPC with public and private subnets (NAT) in the *Amazon VPC User Guide*.

Application listener ports

The application that you want to test on AWS must listen on the following port:

```
0.0.0.0:port
```

Note that the application should not listen on `localhost:port` or `127.0.0.1:port`. This is because when your application is deployed to the cloud, you will connect to the application using a public IP address.

Kestrel server

When your application is ported from .NET to .NET Core, the Kestrel server, which is part of the .NET Core framework, is used as a default web server. The default endpoints (`http://localhost:5000` and `https://localhost:5001`) will not work for the AWS connection. You must explicitly set the Kestrel endpoints. For information about how to set the Kestrel endpoints, see the Microsoft Configure endpoints for the ASP.NET Core Kestrel web server documentation. Note that the documentation uses `http://localhost:5000` as an example but you must use `http://0.0.0.0:5000` or `https://0.0.0.0:5001`.

HTTPS requirements

If the application supports HTTPS, the certificate and private key are required and must be included in the application artifacts.

Linux compatibility requirements

The application that you port to .NET Core will run on a Linux OS, which can cause errors if you are porting your application from a Windows OS and you are unaware of the differences between the operating systems.

In particular, the Linux file system is case-sensitive and the Windows file system is not. Therefore, the paths and names of the files that your application uses must be consistent. For example, if you have a file named `Dinosaur.cfg` and you refer to it as `dinosaur.cfg` in your code, you will receive an error. You must refer to the file as `Dinosaur.cfg` in your code.

AWS Identity and Access Management (IAM) overview

If you are using a license included Visual Studio Amazon Machine Images (AMIs) on Amazon EC2, you can use the caller role (`refactoringtoolkit-RefactoringToolkitCallerRole`) without providing user credentials or editing configuration files. In this case, you do not need to follow the steps in the AWS Identity and Access Management section of this guide. For more information, see Install Toolkit for .NET Refactoring in this guide.

If you are not using a license included Visual Studio AMI on Amazon EC2 or you prefer to create a user and assign roles to the user, see AWS Identity and Access Management in the *Security* section of this guide to view the IAM prerequisites. Follow the steps to create the user, create the access keys, and configure your AWS profile.

AWS roles and managed policies for Toolkit for .NET Refactoring test deployment

To test your application on AWS using Toolkit for .NET Refactoring, you must have the required permissions. Permissions are required for the following tasks:

- Containerization of your .NET Core application.
- Deployment of the ported application to AWS Fargate.

The AWS managed policies page contains detailed information about the managed policies that you can use with Toolkit for .NET Refactoring.

Topics

- Manually create roles and policies
- Create roles and policies with CloudFormation
- AWS KMS key policy

Manually create roles and policies

You can manually create the roles and policies required for Toolkit for .NET Refactoring.

Roles and policies for Toolkit for .NET Refactoring

Use the AWS Management Console to create roles and policies to use Toolkit for .NET Refactoring.

1. Navigate to the AWS Management Console and search for **IAM**.
2. From the IAM dashboard, create a policy named `refactoringtoolkit-EnablePassRoleAccess` and include the following JSON statement:

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam::048661490019:role/refactoringtoolkit-
CodeBuildServiceRole",
        "arn:aws:iam::048661490019:role/refactoringtoolkit-
ECSTaskExecutionRole",
        "arn:aws:iam::048661490019:role/refactoringtoolkit-ECSTaskRole"
      ],
      "Effect": "Allow"
    }
  ]
}
```

3. Create a policy named `refactoringtoolkit-EnableTelemetryAccess` and include the following JSON statement:

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "execute-api:invoke"
      ],
      "Resource": [
        "arn:aws:execute-api:us-east-1:492443789615:3dmmp07yx6/*",
        "arn:aws:execute-api:us-east-1:547614552430:8q2itpfg51/*",
        "arn:aws:execute-api:us-east-1:226975336241:m43z210z43/*",
```

```
        "arn:aws:execute-api:us-east-1:651331843990:lqi4wznpac/*",
        "arn:aws:execute-api:us-west-2:930729463547:ml5fgmwiy3/*"
      ],
      "Effect": "Allow",
      "Sid": "EnCorePermission"
    }
  ]
}
```

4. Create a role named `refactoringtoolkit-CodeBuildServiceRole` and add an inline policy called `CodeBuildServiceRolePolicy` that includes the following JSON statement:

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*",
      "Effect": "Allow",
      "Sid": "CloudWatchLogsPolicy"
    },
    {
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:ListBucket"
      ],
      "Resource": "*",
      "Effect": "Allow",
      "Sid": "S3GetObjectPolicy"
    },
    {
      "Action": [
        "s3:GetBucketAcl",
        "s3:GetBucketLocation"
      ],
      "Resource": "*",
```

```
      "Effect": "Allow",
      "Sid": "S3BucketIdentity"
    },
    {
      "Action": [
        "ecr:InitiateLayerUpload",
        "ecr:PutImage",
        "ecr:UploadLayerPart",
        "ecr:CompleteLayerUpload",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Resource": "*",
      "Effect": "Allow",
      "Sid": "ECRPushPolicy"
    },
    {
      "Action": [
        "ecr:GetAuthorizationToken"
      ],
      "Resource": "*",
      "Effect": "Allow",
      "Sid": "ECRAuthPolicy"
    }
  ]
}
```

5. Create a role named `refactoringtoolkit-ECSTaskRole` and add the `AWSRefactoringToolkitSidecarPolicy` policy.
6. Create a role named `refactoringtoolkit-ECSTaskExecutionRole` and add the `AmazonECSTaskExecutionRolePolicy` policy.
7. Create a role named `refactoringtoolkit-RefactoringToolkitCallerRole` and add the `refactoringtoolkit-EnablePassRoleAccess`, `refactoringtoolkit-EnableTelemetryAccess`, and `AWSRefactoringToolkitFullAccess` policies.

Create roles and policies with CloudFormation

Use the CloudFormation template to create roles and policies that you will use for Toolkit for .NET Refactoring.

1. Use the following command to download the CloudFormation template.

```
aws s3 cp s3://aws.portingassistant.dotnet.download/ide.extension.role.creation/
aws-refactoringtoolkit-iam-roles.yaml .
```

2. Create the required IAM roles in your account. This step uses the `aws-refactoringtoolkit-iam-roles.yaml` file that you downloaded in the previous step. The user that executes the command to create the roles must have the following permissions:

   - `iam:CreateRole`
   - `iam:CreatePolicy`
   - `iam:AttachRolePolicy`

   To create the roles, run the following AWS CLI command:

```
aws cloudformation deploy --stack-name refactoringtoolkit --template-file aws-
refactoringtoolkit-iam-roles.yaml --capabilities CAPABILITY_NAMED_IAM
```

The following sections provide detailed information about the roles that are created by the CloudFormation template.

Topics

Role and policy for the user calling the API

The CloudFormation template creates the following policy and role:

- Policy – `refactoringtoolkit-EnablePassRoleAccess`
- Policy – `refactoringtoolkit-EnableTelemetryAccess`

- Role – `refactoringtoolkit-RefactoringToolkitCallerRole`

Assign a user to the role created by the CloudFormation template

The role that was created by the CloudFormation template contains the policies required to use Toolkit for .NET Refactoring. You can assign a user to this role.

To enable a user to assume the role, you need the Amazon Resource Name (ARN) of the user. For more information, see Amazon Resource Names (ARNs) in the *AWS General Reference*.

To allow the user to be assigned a role, you must edit the trust relationship for the `refactoringtoolkit-RefactoringToolkitCallerRole` role. For more information, see Granting a user permissions to switch roles in the *AWS Identity and Access Management User Guide*.

Edit the trust relationship to add the principal with the ARN of the user:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::123456789012:user/John",
      "Service": "ec2.amazonaws.com"
    },
    "Action": "sts:AssumeRole"
  }
}
```

Note

By default, this role has a trust relationship that allows the Amazon EC2 instance to use the role. If you delete the `"Service": "ec2.amazonaws.com"` line, the instance profile will no longer work.

Create a new named profile that uses a role

> **Important**
>
> The `~/.aws/config` file contains a section for your user profile. In this step, do not edit the section for your user profile. Instead, you will create a new role profile section.

Add the text below to the `~/.aws/config` file to create the `refactoringtoolkit` profile. You can substitute any string for *<user_role_profile>*, such as `refactoringtoolkit_profile`.

```
[<user_role_profile>]
role_arn = arn:aws:iam::<AWS_account_ID>:role/refactoringtoolkit-
RefactoringToolkitCallerRole
source_profile = <user_profile>
```

For information about how to add a role to a profile, see Using an IAM role in the AWS CLI in the *AWS CLI User Guide for Version 2*. For information about credentials, see Shared config and credentials files in the *AWS SDKs and Tools Reference Guide*.

Attach the policies to the user

Alternatively, you can attach the policies to a user rather than a role. If you want to do this, you can attach the following policies directly to the user:

- `AWSRefactoringToolkitFullAccess`
- `refactoringtoolkit-EnableTelemetryAccess`
- `refactoringtoolkit-EnablePassRoleAccess`

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

  Create a permission set. Follow the instructions in Create a permission set in the *AWS IAM Identity Center User Guide*.
- Users managed in IAM through an identity provider:

  Create a role for identity federation. Follow the instructions in Create a role for a third-party identity provider (federation) in the *IAM User Guide*.

- IAM users:
  - Create a role that your user can assume. Follow the instructions in Create a role for an IAM user in the *IAM User Guide*.
  - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in Adding permissions to a user (console) in the *IAM User Guide*.

Toolkit for .Net Refactoring caller role

The CloudFormation template creates the following caller role:

```
refactoringtoolkit-RefactoringToolkitCallerRole
```

This role grants Toolkit for .NET Refactoring permissions to do the following:

- Call the required APIs in AWS.
- Upload application artifacts and download the resulting artifacts from S3.
- Build the application into a container image using AWS CodeBuild and store and retrieve the images in Amazon Elastic Container Registry (Amazon ECR).
- Deploy the application to container services on AWS, such as Amazon Elastic Container Service (Amazon ECS).
- Optionally, create Amazon VPC resources.
- Optionally, connect to existing infrastructure, such as Directory Service.

The following policies are attached to the role:

- `refactoringtoolkit-EnablePassRoleAccess`
- `refactoringtoolkit-EnableTelemetryAccess`
- `AWSRefactoringToolkitFullAccess`

To use this role, you must do one of the following:

- Add it to a named profile in the `~/.aws/config` file. For more information, see Assign a user to the role created by the CloudFormation template in this guide.
- Use it with an Amazon EC2 instance profile. For more information, see Install Toolkit for .NET Refactoring in this guide.

AWS CodeBuild role

The CloudFormation template creates the following CodeBuild role:

```
refactoringtoolkit-CodeBuildServiceRole
```

AWS CodeBuild uses this role to download application artifacts from S3, build the application container, push to Amazon Elastic Container Registry (Amazon ECR), and create logs in Amazon CloudWatch.

The following policy is attached to the role:

```
CodeBuildServiceRolePolicy
```

Toolkit for .NET Refactoring will automatically use a role with this name.

Amazon ECS task execution role

The CloudFormation template creates the following Amazon ECS task execution role:

```
refactoringtoolkit-ECSTaskExecutionRole
```

This role allows Amazon ECS to pull the application Docker image from Amazon ECR and upload logs.

The following policies are attached to the role:

- `AmazonECSTaskExecutionRolePolicy`
- `ECSSecretsPolicy`

Toolkit for .NET Refactoring will automatically use a role with this name.

For more information, see Amazon ECS task execution IAM role in the *Amazon ECS User Guide*.

Amazon ECS task role

The CloudFormation template creates the following Amazon ECS task role:

```
refactoringtoolkit-ECSTaskRole
```

This role allows your application to authenticate with other AWS services. It is similar to the IAM roles that are used as instance profiles for Amazon EC2 instances. This role is required for test deployment to allow the sidecar to sync files from S3 and for Active Directory to retrieve credentials from AWS Secrets Manager.

The following policy is attached to the role:

```
AWSRefactoringToolkitSidecarPolicy
```

Toolkit for .NET Refactoring will automatically use a role with this name.

For more information, see IAM roles for tasks in the *Amazon ECS User Guide*.

AWS KMS key policy

To use Toolkit for .NET Refactoring with an AWS KMS key to encrypt AWS resources, the KMS key policy must include the following statement:

```
{
    "Effect": "Allow",
    "Principal": {
        "Service": "application-transformation.amazonaws.com"
    },
    "Action": [
        "kms:Decrypt",
        "kms:DescribeKey",
        "kms:GenerateDataKey"
    ],
    "Resource": "kmsKeyArn"
}
```

You can create a new KMS key with the previously mentioned policy, or add it to an existing KMS key. For more information, see Creating a key policy and Changing a key policy in the *AWS Key Management Service User Guide*.

Active Directory setup

The application that you use for the test deployment can use Microsoft Active Directory authentication against its dependencies, such as a Microsoft SQL Server database that is joined into an Active Directory domain.

Note that this allows authentication of the application in the test deployment. It does not provide single sign-on for the incoming on-premises user connections with Windows Authentication tokens in HTTP headers.

For more information, see Join an Amazon EC2 instance to your AWS Managed Microsoft AD directory in the *AWS Directory Service Administration Guide*.

Topics

Create a directory

Create a directory in the Directory Service using the same VPC that you will use for test deployment. If you use Active Directory, you must use the **Select an Amazon VPC** option when you create the test deployment. For more information, see Create your AWS Managed Microsoft AD in the *AWS Directory Service Administration Guide*.

Verify that the directory meets the following requirements:

- The inbound rules of the security group used by your directory must allow incoming connections from the same VPC. For more information, see Understand your directory's AWS security group configuration in the *AWS Directory Service Administration Guide*.
- The VPC must have a DHCP options set that lists both of the IP addresses of the directory as DNS servers. For more information, see Create a DHCP options set in the *AWS Directory Service Administration Guide*.

Create a user

Create a user in the directory. Remember the sign-in credentials.

Create a secret

Create a secret to pass the username and password to your application. Create the secret with the user credentials in the AWS Secrets Manager before you run the test deployment on AWS. The user credentials must contain the following fields:

- `Username` – The domain in the `Username` value must be uppercase: *`user`*`@`*`AD_DOMAIN`*
- `Password`

You can also use the AWS Secrets Manager in the AWS Management Console to create the secret. Create a secret and add values in the following way:

```
Key: Username, value: user@AD_DOMAIN,
Key: Password, value: password
```

You can use the JSON format option in the secret as:

```
{
    "Username": "user@AD_DOMAIN",
    "Password": "password"
}
```

You can also use the AWS CLI to create the secret. For more information, see create-secret in the *AWS CLI Command Reference*.

If you use AWS CLI, use the `--secret-string` parameter as follows:

```
'{"Username":"user@AD_DOMAIN.COM","Password":"password"}'
```

The single quotes that enclose the JSON value allow you to pass the double quotes unchanged.

Remember the ARN of the secret for later use.

Allow the task role to read the secret

In the AWS Secrets Manager in the AWS Management Console, open the secret that you created and select the **Resource permissions** tab in the information page of the secret.

Add the policy below to the secret. The policy contains the following ARNs:

- The ARN of the Amazon ECS task role that you created with the CloudFormation template.
- The ARN of the secret that you just created.

JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [
    {
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::123456789012:role/my-ecs-task-role"
        },
        "Action": "secretsmanager:GetSecretValue",
        "Resource": "arn:aws:secretsmanager:us-east-1:123456789012:secret:my-secret-XyZ9Qw"
    }
  ]
}
```

Install Toolkit for .NET Refactoring

You can install Toolkit for .NET Refactoring within Microsoft Visual Studio, or the extension is available as part of license included Visual Studio Amazon Machine Images (AMIs) on Amazon EC2.

Install in Visual Studio

To install the Toolkit for .NET Refactoring extension, open Microsoft Visual Studio and complete the following steps:

1. In Visual Studio, open the **Extensions** menu and select **Manage Extensions**.
2. In the **Manage Extensions** window, search for Toolkit for .NET Refactoring. Select the extension and click **Download**.
3. When the download is complete, Visual Studio prompts you to restart the application. Restart Visual Studio to install the Toolkit for .NET Refactoring extension.

Install with an AMI on Amazon EC2

If you are using a license included Visual Studio Amazon Machine Image (AMI) on Amazon EC2, you can use the caller role (`refactoringtoolkit-RefactoringToolkitCallerRole`) directly,

without providing user credentials or editing configuration files. For more information about AMIs, see User-based subscriptions in AWS License Manager in the *AWS License Manager User Guide*.

The CloudFormation template creates an instance profile named `refactoringtoolkit-Ec2InstanceProfile`. For more information about the CloudFormation template, see Create roles and policies with CloudFormation in this guide.

You can launch an Amazon EC2 instance with an IAM role through the AWS Management Console or AWS CLI.

Launch an Amazon EC2 instance in the AWS Management Console

1. Launch an instance. For more information, see Quickly launch an instance in the *Amazon Elastic Compute Cloud User Guide for Linux Instances*.
2. Expand **Advanced details**, and in the **IAM instance profile** field, select `refactoringtoolkit-Ec2InstanceProfile`.

Launch an EC2 instance with AWS CLI

Use AWS CLI to launch an instance that uses the instance profile. For more information, see run-instances in the *AWS CLI Command Reference.*

The code below is an example command that illustrates how to launch an EC2 instance with the instance profile:

```
aws ec2 run-instances \
    --image-id ami-11aa22bb \
    --iam-instance-profile Name="s3access-profile" \
    --key-name my-key-pair \
    --security-groups my-security-group \
    --subnet-id subnet-1a2b3c4d
```

For more information about instance profiles, see Using instance profiles in the *AWS Identity and Access Management User Guide*. For more information about using IAM roles with Amazon EC2 instances, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *AWS Identity and Access Management User Guide*. For information about retrieving credentials from instance metadata, see Retrieve security credentials from instance metadata in the *Amazon Elastic Compute Cloud User Guide for Linux Instances*.

Pricing for Toolkit for .NET Refactoring

The Toolkit for .NET Refactoring extension for Microsoft Visual Studio is available for use at no cost. However, testing an application on AWS requires the use of Amazon Virtual Private Cloud, Amazon Simple Storage Service, and AWS Fargate:

- Amazon VPC pricing
- Amazon S3 pricing
- AWS Fargate pricing

Toolkit for .NET Refactoring extension

With the Toolkit for .NET Refactoring extension, you can use porting, containerization, and deployment features seamlessly from within Microsoft Visual Studio. The Toolkit for .NET Refactoring extension provides prescriptive guidance to help you assess and port your Microsoft Windows .NET Framework applications to .NET Core on Linux. After you port a web application, you can test the application on AWS. The extension facilitates collaboration with other developers that are analyzing, debugging, testing, and refactoring the same application code.

The sections below show you how to assess your solution and analyze the results.

Topics

Run a compatibility assessment

After the extension is installed, you can run a compatibility assessment with Toolkit for .NET Refactoring to find Microsoft Windows dependencies and incompatibilities between your application and newer .NET Core versions. Perform the following steps to run the assessment:

1. In Microsoft Visual Studio, open a solution file and open a `.cs` file within the solution that you want to run the assessment on.
2. From the Visual Studio menu, open **Extensions** and select **AWS Toolkit for .NET Refactoring** from the drop-down menu. Select **Get Started**.
3. The first time that you run an assessment, the **Start modernization journey** tab opens. In the **Start modernization journey** tab, configure the following options:

    - **AWS profile** – Select a named profile from the drop-down menu.
    - **Use existing AWS CLI / SDK credentials** – Select this option if you have temporary credentials. For more information, see Temporary security credentials in IAM in the *IAM User Guide*.

- **Share my usage data** – Select this option to allow Toolkit for .NET Refactoring to collect usage data. For more information, see Data collected in this guide.

4. From the **Getting Started** screen, you can choose the **Target Framework** version that you want to port your application to.
5. Click **Save** to save your settings.
6. Click **Next** to open the Toolkit for .NET Refactoring dashboard.
7. In the dashboard, click **Start assessment** to begin the assessment. Toolkit for .NET Refactoring runs a one-time, full assessment for the compatibility solution that is loaded in Visual Studio.

There is an **Error list** pane at the bottom of the window that displays the incompatibilities that Toolkit for .NET Refactoring discovered during the assessment. Select an entry in the list to view the incompatibility in the source code. The incompatible code is highlighted.

You can change the settings you entered into the **Getting Started** screen by selecting the **Tools** tab and choosing **Options** from the drop-down menu. Under **AWS Toolkit for .NET Refactoring VS Extension**, select **Data usage sharing** to select a different **AWS Named Profile**, **Add a named profile**, or to change your usage data selection. Choose **General** under **AWS Toolkit for .NET Refactoring VS Extension** to update the **Target Framework**.

Analyze the results

When the assessment is complete, you can view the results in the dashboard. The dashboard contains the following information:

- **Status** – The status of the assessment. **Assessment complete** indicates that you can review the assessment in the **Assessment Overview** pane.
- **Solution file** – The file that you selected to run the assessment on.
- **Version** – The .NET Core version that your solution file was compared to during the assessment.
- **Incompatible NuGet packages** – The number of .NET Framework NuGet packages that are incompatible with the .NET Core version that you selected for the assessment.
- **Portable NuGet packages** – The number of .NET Framework NuGet packages that are compatible with the .NET Core version that you selected for the assessment.
- **Ported projects** – The number of projects that have been ported to the .NET Core version that you selected for the assessment.

- **Incompatible APIs** – The number of .NET Framework APIs that are incompatible with the .NET Core version that you selected for the assessment.
- **Portable APIs** – The number of .NET Framework APIs that are compatible with the .NET Core version that you selected for the assessment.

You can click on the incompatible NuGet packages or incompatible APIs to view details about individual incompatibilities, view the project dependency graph, or export the results to a `.csv` file.

Port the solution

Toolkit for .NET Refactoring provides the following features to help you port the solution.

Guided assistance

You can apply all the recommended code changes at once or you can view each incompatibility individually. When you port a project or solution, Toolkit for .NET Refactoring prompts you to choose whether you want to apply all the recommended code changes. If you do not select that option, Toolkit for .NET Refactoring prompts you to view each incompatibility. After you make a code change, save your updates to view the updated recommendations for your code after the modifications have been applied.

When you port your solution to .NET Core, the Toolkit for .NET Refactoring extension provides guided assistance in the source editor.

When you are ready to port a project, select one of the following options from the **Extensions** tab:

- **Port solution to .NET Core** – Use this option to port all projects to .NET Core.
- **Port project to .NET Core** – Use this option to port a single project to .NET Core.

When the source file is open with a source editor, porting options are provided to assist you, and to automate the porting of each source file. Automated porting involves running the source code through code translation assistant rules for porting. This can be performed at the solution level when you select **Port solution to .NET Core** and choose to apply recommended source code changes.

View incompatibilities

View the list of incompatibilities and suggestions from the **Error list** pane at the bottom of the window. Select an error in the list to view the code in the source editor. The source file is annotated with the error message, details about the compatibility issue, and a recommended solution.

If you used automated porting to apply all the recommended code changes, a light bulb icon appears next to each line of code that was modified, with a comment describing the change.

In the source file, each line of source code with compatibility issues is highlighted. This helps you to visualize the issues in the file as you address them. The highlighting is disappears when the compatibility issues are addressed.

Suggestions

The source editor provides a replacement suggestion for each incompatibility in the source file. If a direct replacement exists, you can choose to select and replace it. If there is no direct replacement, references or contextual help on how to proceed are provided.

Test the application on AWS

After you run the assessment and made code changes, you can run a test deployment to verify the changes to your application on AWS.

Topics

What is a test deployment?

A test deployment is a simplified application deployment into AWS that allows you to deploy the application and validate code changes quickly.

Test deployment is intended for use during the development cycle of the application. To reduce turnaround time and costs, a test deployment is simplified in the following ways:

- Scalability is limited to one instance of the application.
- Because there is only one instance of the application, a load balancer is not created.
- Amazon API Gateway is not created.
- Route 53 Private DNS for VPCs is not created.
- A health check is not performed on the application. You are expected to perform testing to make sure the application works as expected. For debugging, you can use logs as well as the Amazon Elastic Container Service (Amazon ECS) `execute` command to execute commands inside the application container. If the application crashes, the container and Amazon ECS task is not deleted so you can review the container contents to determine why the application crashed. Use the `deploy` command to restart the application.

Test deployment cycle

The test development cycle consists of the following steps:

1. The developer builds the project and produces the application files, such as executable files, data files, and configuration files.
2. Toolkit for .NET Refactoring creates a container image for the application that is added to the Amazon Elastic Container Registry (Amazon ECR) of the account.
3. If a custom Docker file is specified, the container is created according to the file. Note that the file must adhere to the following guidelines, which are required for the application redeployment and restart:
   - The entire application must be located in the `/app` directory.
   - The start script for the application must be named `/app/entryfile`.

   If a custom Docker file is not specified, Toolkit for .NET Refactoring does the following:
   - The application files are placed into the `/app` directory in the container. The subdirectory structure is preserved.
   - The application start script is created, which executes the following command: `dotnet <application_name>`
4. Toolkit for .NET Refactoring creates a deployment in AWS that includes the AWS Fargate cluster, the Amazon ECS service, and the AWS Fargate task.
5. Toolkit for .NET Refactoring creates security groups for the application to provide network security. Only traffic to the specified application ports is permitted.

6. Toolkit for .NET Refactoring displays:
   - The IP address of the deployed application. You can use the IP address to connect the clients and browsers to the application ports.
   - The names of Amazon CloudWatch log groups and log streams. You can use these to view the logs of the application. Note that only console logs are supported.
7. If you change the code, data, or configuration files, rebuild the project and deploy it again.
   - If your changes include significant modifications to the application compute environment, such as open ports, the base image, or the Docker file, Toolkit for .NET Refactoring will rebuild a container image and deploy a new Amazon ECS service.
   - If the changes are limited to the application files, Toolkit for .NET Refactoring will deliver the new files to the Amazon ECS task and restart the application.

Create a deployment

Create a deployment to test your application on AWS. Before you begin, make sure you have verified the requirements in Setting up for testing on AWS.

Complete the following steps to run a test deployment on AWS with Toolkit for .NET Refactoring.

1. Compile and build the test application in Visual Studio. See the Microsoft Compile and build in Visual Studio documentation for details.
2. In Microsoft Visual Studio, open the **Extensions** menu and select **AWS Toolkit for .NET Refactoring**, **Test on AWS**.
3. The **Toolkit for .NET Refactoring – Test on AWS** pop-up appears. In the pop-up, configure the following options:
   - **AWS profile to run test deployment** – The profile that you want to use to run the deployment.
   - **AWS region for test deployment** – The region where you want to run the test deployment. You must have an Amazon S3 bucket in the region that you select. Toolkit for .NET Refactoring supports the following regions:
     - US East - `us-east-2`
     - Europe (London) - `eu-west-2`
   - **Published artifacts** – Select the folder that contains the build artifacts that were generated by Visual Studio when you published to a folder. For information about publishing to a

folder in Visual Studio, see the Microsoft Build and clean projects and solutions in Visual Studio documentation.

Note that if the application supports HTTPS, you must provide the certificate and a private key as part of the application artifacts in the build folder that will be added to the `/app` directory in the container. In other words, the certificate paths in your configuration files should be `/app/<certificate-file>`.

- **Test deployment project name** – The project that you want to deploy. This is a project that has been assessed by Toolkit for .NET Refactoring.
- **Deployment name** – The name that you will use to identify the deployment. The deployment name should be unique for each project deployment in the solution.
- **Amazon S3 bucket name** – The name of the Amazon S3 bucket where you want to upload the application artifact.
- **Amazon Virtual Private Cloud for test deployment**
  - **Create a new Amazon VPC** – A new Amazon VPC will be created as part of the deployment.
  - **Select an Amazon VPC** – If you select an existing Amazon VPC, check the VPC requirements to verify that the VPC satisfies the requirements.
- **Deploy with Microsoft Active Directory** – Note that if you select this option, you must also select the **Select an Amazon VPC** option. You cannot use Active Directory with the **Create a new Amazon VPC** option. See Active Directory setup for more information about using Active Directory in a test deployment.
- **Advanced settings**
  - **Custom Docker file** – The Docker file that you want to use to build the container. Note that the file must adhere to the following guidelines, which are required for application redeployment and restart:
    - The entire application must be located in the `/app` directory.
    - The start script for the application must be named `/app/entryfile`.
  - **Number of CPUs** – The default CPU value is one vCPU. Adjust the number of CPUs based on your application.
  - **Memory size** – Default is 1 GB.

4. Click **Test on AWS** to start the test deployment.

To view the status of the deployment, open the **Extensions** menu in Visual Studio and select **AWS Toolkit for .NET Refactoring**, **View Deployments Running on AWS**. When the deployment is complete there will be a status update at the top of the Visual Studio window.

Delete a deployment

Toolkit for .NET Refactoring does not delete deployments automatically. To delete a deployment manually, click the **Delete** button within the deployment. This will open CloudFormation in the AWS Management Console. In the AWS Management Console, you can delete the CloudFormation stack.

Debugging

This section contains debugging information for Toolkit for .NET Refactoring.

Debugging the deployment using Amazon ECS execute-command

You can use Amazon ECS to run commands, such as bash and sh commands, in a task that is running.

1. To use Amazon ECS to run a command, open Amazon ECS in the AWS Management Console and navigate to the cluster and the task that contains the application. The cluster will be named *<application_id>*-cluster.
2. Locate a single service in the cluster and locate an active task in the cluster. Use the cluster name and task ID in an AWS CLI command:

```
aws ecs execute-command --cluster <cluster_name> \
    --task <task_id> \
    --container app \
    --interactive \
    --command "bash"
```

For more information, see the **Running commands using Amazon ECS Exec** section of Using Amazon ECS Exec for debugging in the *Amazon ECS User Guide*.

Security in AWS Toolkit for .NET Refactoring

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to AWS Toolkit for .NET Refactoring, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Toolkit for .NET Refactoring. The following topics show you how to configure Toolkit for .NET Refactoring to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Toolkit for .NET Refactoring resources.

Topics

AWS Identity and Access Management (IAM)

If you use Toolkit for .NET Refactoring with a license included Visual Studio Amazon Machine Image (AMI) on Amazon EC2, you can use the `refactoringtoolkit-RefactoringToolkitCallerRole` without providing credentials or modifying configuration files. For more information, see Install Toolkit for .NET Refactoring in this guide.

If you are not using Toolkit for .NET Refactoring with Amazon EC2 or if you prefer to create a user and assign roles to the user, follow the steps in this section to create the user, create access keys, and configure your AWS profile.

Topics

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Create access keys for your user

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	To	By
IAM	(Recommended) Use console credentials as temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Login for AWS local development in the *AWS Command Line Interface User Guide*. • For AWS SDKs, see Login for AWS local development in the *AWS SDKs and Tools Reference Guide*.
Workforce identity (Users managed in IAM Identity Center)	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the *AWS Command Line Interface User Guide*.

Which user needs programmatic access?	To	By
		• For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide*.
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions in Using temporary credentials with AWS resources in the *IAM User Guide*.
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Authenticating using IAM user credentials in the *AWS Command Line Interface User Guide*. • For AWS SDKs and tools, see Authenticate using long-term credentials in the *AWS SDKs and Tools Reference Guide*. • For AWS APIs, see Managing access keys for IAM users in the *IAM User Guide*.

Configure your AWS profile

After you have created a user, you can configure your AWS named profile to apply settings and credentials to be applied when you run commands.

Configure AWS profile in the assessment tool

1. In the Toolkit for .NET Refactoring assessment tool, navigate to **Set up Toolkit for .NET Refactoring**.
2. Choose **Add a profile** under **AWS named profile**.
3. Enter your new **Profile name**, **AWS access key ID**, and **AWS secret access key**.
4. Choose **Add**.

Configure AWS profile using the AWS CLI

1. Run the following AWS CLI command to create a profile for Toolkit for .NET Refactoring. The profile is named `default` in the credentials file.

```
aws configure
```

2. For each prompt, enter the corresponding information.
   - `AWS Access Key ID`
   - `AWS Secret Access Key`
   - `Default region name` (For example, `us-west-2`)
   - `Default output format`
3. After you configure the profile using the AWS CLI, Toolkit for .NET Refactoring will display the `default` profile under **AWS named profile** on the **Set up Toolkit for .NET Refactoring** page of the assessment tool.

For more information about configuring the AWS CLI, see Configuring the AWS CLI in the *AWS CLI User Guide*.

EULA

AWS Toolkit for .NET Refactoring is licensed as *AWS Content* under the terms and conditions of the AWS Customer Agreement. For more information, see AWS Customer Agreement and AWS Service Terms. By installing, using, or accessing AWS Toolkit for .NET Refactoring, you agree to such terms and conditions. The term *AWS Content* does not include software and assets distributed under separate license terms (such as code licensed under an open source license).

Data protection in AWS Toolkit for .NET Refactoring

The AWS shared responsibility model applies to data protection in AWS Toolkit for .NET Refactoring. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with Toolkit for .NET Refactoring or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Data collected by Toolkit for .NET Refactoring

If you accept the data collection option in the **Settings** menu of the Toolkit for .NET Refactoring extension, the following application data is collected:

1. Application errors generated when running assessments, porting, or when performing other functions provided by the Toolkit for .NET Refactoring extension.
2. Names and versions of public NuGet packages assessed by the Toolkit for .NET Refactoring extension.
3. Metrics for assessments run by the Toolkit for .NET Refactoring extension on public NuGet packages, such as the number of packages and solutions, and the amount of time taken to create a solution.

You can change your data collection settings at any time in the **Settings** menu of the Toolkit for .NET Refactoring extension.

Toolkit for .NET Refactoring will perform surface level analysis of your .NET Framework solution and generate a list of the following that are in use:

- NuGet packages
- NuGet APIs
- .NET SDK APIs

Only the generated list is stored in the S3 bucket you provide, and your source code never leaves your local system. Toolkit for .NET Refactoring has a scalable backend that processes source code metadata ephemerally. The backend returns compatibility and recommendation information back to you. For more information about data privacy, see Data Privacy FAQ.

You can choose whether to encrypt your source code metadata that is uploaded to Amazon S3. Toolkit for .NET Refactoring supports using your own customer managed keys in AWS Key Management Service (AWS KMS) for encryption and decryption. For more information, see Protecting data with encryption in the *Amazon Simple Storage Service User Guide* and AWS KMS concepts in the *AWS Key Management Service Developer Guide*.

Encryption at rest

All data within Toolkit for .NET Refactoring is encrypted at rest in accordance with industry standards.

Encryption in transit

All requests to Toolkit for .NET Refactoring must be made over the Transport Layer Security protocol (TLS). We recommend TLS 1.2 or later.

AWS managed policies for AWS Toolkit for .NET Refactoring

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the *IAM User Guide*.

AWS managed policy: AWSRefactoringToolkitFullAccess

Use this policy to test your application on AWS when you modernize your application with the AWS Toolkit for .NET Refactoring extension for Microsoft Visual Studio. Attach the policy to your local AWS profile. The policy grants permissions to upload application artifacts and download the resulting artifacts from S3. It grants permissions to build the application into a container image using AWS CodeBuild and store and retrieve the images in Amazon Elastic Container Registry (Amazon ECR). In addition, it allows for the deployment of the application to container services on AWS, such as Amazon Elastic Container Service (Amazon ECS), the optional creation of Amazon VPC resources, and the optional connection to existing infrastructure, such as AWS Directory Service.

To view the permissions for this policy, see AWSRefactoringToolkitFullAccess in the *AWS Managed Policy Reference*.

Permission details

This policy includes permissions for the following services:

- Application Transformation – Allows Toolkit for .NET Refactoring to perform compatibility assessments along with containerization and deployment operations.
- Amazon CloudWatch – Allows Toolkit for .NET Refactoring to create log groups and store log output from Toolkit for .NET Refactoring operations and your applications.
- Amazon Elastic Compute Cloud (Amazon EC2) – Allows Toolkit for .NET Refactoring to create and modify security groups, internet gateways, and Amazon VPC.
- Amazon Elastic Container Registry (Amazon ECR) – Allows Toolkit for .NET Refactoring to create and read your Amazon ECR repository and to tag resources.
- Amazon Elastic Container Service (Amazon ECS) – Allows Toolkit for .NET Refactoring to run your container images as tasks in Amazon ECS and to facilitate debugging.
- Amazon Simple Storage Service (Amazon S3) – Allows Toolkit for .NET Refactoring to list and manage objects from Amazon S3 buckets that are used by Toolkit for .NET Refactoring.
- CloudFormation – Allows Toolkit for .NET Refactoring to deploy the infrastructure components of CodeBuild and application deployment in the form of CloudFormation stacks.
- AWS CodeBuild – Allows Toolkit for .NET Refactoring to allocate resources for CodeBuild projects and start builds.
- AWS Identity and Access Management (IAM) – Allows Toolkit for .NET Refactoring to verify which roles are passed to other AWS services.
- AWS Key Management Service (AWS KMS) – Allows Toolkit for .NET Refactoring to utilize user-provided KMS keys across AWS services for encryption.
- AWS Systems Manager (Systems Manager) – Allows Toolkit for .NET Refactoring to manage Systems Manager parameters and communicate with Amazon ECS tasks.

AWS managed policy: AWSRefactoringToolkitSidecarPolicy

This policy is used by the Amazon ECS tasks that are created to run a test application on AWS with the AWS Toolkit for .NET Refactoring extension for Microsoft Visual Studio. The policy grants

permissions to download application artifacts from S3 and to communicate the status of the task using AWS Systems Manager.

To view the permissions for this policy, see AWSRefactoringToolkitSidecarPolicy in the *AWS Managed Policy Reference*.

Toolkit for .NET Refactoring updates to AWS managed policies

View details about updates to AWS managed policies for Toolkit for .NET Refactoring since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Toolkit for .NET Refactoring Document history page.

Change	Description	Date
AWSRefactoringToolkitFullAccess – Updated policy	Modified the permissions for `CreateTags` by adding conditions that use the `aws:CallVia` condition key.	April 9, 2025
AWSRefactoringToolkitFullAccess – Updated policy	Added permissions for the `cloudformation:TagResource` and `cloudformation:UntagResource` actions to manage resource tags on stacks created for AWS App2Container.	March 25, 2024

Change	Description	Date
AWSRefactoringToolkitFullAccess – Updated policy	Added permissions for `application-transformation` actions and KMS permissions for resources matching `ForAnyValue:StringLike` with `"kms:ResourceAliases": "alias/application-transformation*"` . Changed permissions for EC2, ECR, ECS, and CloudWatch Logs to scope them to the `application-transformation` request and resource tag.	November 18, 2023
AWSRefactoringToolkitFullAccess – Updated policy	Added `ListStacks` permissions.	March 22, 2023

Change	Description	Date
AWSRefactoringToolkitFullAccess – Updated policy	Added tagging permissions that allow new accounts to perform the `CreateLogGroup` action.	December 15, 2022
AWSRefactoringToolkitSidecarPolicy – Updated policy	Added permissions to open a data channel to transfer files to the customer's container.	October 29, 2022
AWSRefactoringToolkitFullAccess – New policy	Added the `AWSRefactoringToolkitFullAccess` policy.	October 25, 2022
AWSRefactoringToolkitSidecarPolicy – New policy	Added the `AWSRefactoringToolkitSidecarPolicy` policy.	October 25, 2022

Troubleshooting

This section contains troubleshooting information for Toolkit for .NET Refactoring.

Sidecar logs

If you are using Microsoft Active Directory (AD) with Toolkit for .NET Refactoring, the sidecar container performs authentication with Active Directory using the credentials from the specified secret. If the authentication fails, the deployment job will fail.

The logs of the sidecar are returned in the details of the deployment. Check the logs for text that says something similar to `invalid password`.

Document history for the Toolkit for .NET Refactoring User Guide

The table below describes the documentation releases for Toolkit for .NET Refactoring.

Change	Description	Date
AWS managed policy updates	Updated the `AWSRefactoringToolkitFullAccess` policy.	April 9, 2025
AWS managed policy updates	Updated the `AWSRefactoringToolkitFullAccess` policy.	March 25, 2024
AWS managed policy updates	Updated the `AWSRefactoringToolkitFullAccess` policy.	November 18, 2023
AWS managed policy updates	Updated the `AWSRefactoringToolkitFullAccess` policy.	March 22, 2023
IAM best practices updates	Updated the guide to align with the IAM best practices . For more information, see Security best practices in IAM.	February 22, 2023
AWS managed policy updates	Updated the `AWSRefactoringToolkitFullAccess` policy.	December 15, 2022
AWS managed policy updates	Updated the `AWSRefactoringToolkitSidecarPolicy` policy.	October 29, 2022

Initial release	Initial release of AWS Toolkit for .NET Refactoring: an extension for Microsoft Visual Studio that reduces the time and effort that is required to refactor legacy .NET applications to cloud-native alternatives on AWS.	October 25, 2022
AWS managed policy updates	Added the `AWSRefactoringToolkitFullAccess` and `AWSRefactoringToolkitSidecarPolicy` policies.	October 25, 2022