User Guide

AWS Toolkit for VS Code

AWS Toolkit for VS Code: User Guide

Table of Contents

AWS Toolkit for Visual Studio Code

This is the user guide for the **AWS Toolkit for VS Code**. If you are looking for the **AWS Toolkit for Visual Studio**, see the *User Guide for the AWS Toolkit for Visual Studio*.

What is the AWS Toolkit for Visual Studio Code

The Toolkit for VS Code is an open-source extension for the Visual Studio Code (VS Code) editor. This extension makes it easier for developers to develop, debug locally, and deploy serverless applications that use Amazon Web Services (AWS).

Topics

Related information

Use the following resources to access the source code for the toolkit or view currently open issues.

- Source Code
- Issue Tracker

To learn more about the Visual Studio Code editor, visit https://code.visualstudio.com/.

Amazon Q Developer and Amazon CodeWhisperer

As of April 30th 2024, Amazon CodeWhisperer is now part of Amazon Q Developer, this includes inline code suggestions and Amazon Q Developer security scans. Download the Amazon Q Developer IDE extension from the VS Code Marketplace to get started.

For details about the Amazon Q Developer service, see the Amazon Q Developer User Guide. For detailed information about plans and pricing for Amazon Q, see the Amazon Q pricing guide.

Downloading the Toolkit for VS Code

You can download, install, and set up the AWS Toolkit for Visual Studio Code through the VS Code Marketplace in your IDE. For detailed instructions, see the Download and install section in the *Getting started* topic of this User Guide.

Downloading the Toolkit from the VS Code Marketplace

Alternatively, you can download the AWS Toolkit for Visual Studio Code installation files by navigating to the **VS Code Marketplace** from your web browser.

Additional IDE Toolkits from AWS

In addition to the AWS Toolkit for Visual Studio Code, AWS also offers IDE Toolkits for JetBrains and Visual Studio.

AWS Toolkit for JetBrains links

- Follow this link to Download the AWS Toolkit for JetBrains from the JetBrains Marketplace.
- To learn more about the AWS Toolkit for JetBrains, see the AWS Toolkit for JetBrains User Guide.

Toolkit for Visual Studio links

- Follow this link to Download the Toolkit for Visual Studio from the Visual Studio Marketplace.
- To learn more about the Toolkit for Visual Studio, see the Toolkit for Visual Studio User Guide.

Getting Started with the AWS Toolkit for Visual Studio Code

The AWS Toolkit for Visual Studio Code makes your AWS services and resources available, directly from your VS Code integrated development environment (IDE).

To get you started, the following topics describe how to set up, install, and configure the AWS Toolkit for Visual Studio Code.

Topics

Installing the AWS Toolkit for Visual Studio Code

Prerequisites

To get started working with AWS Toolkit for Visual Studio Code from VS Code, the following perquisites must be met. To learn more about accessing all of the AWS services and resources available from the AWS Toolkit for Visual Studio Code, see the the section called “Optional prerequisites” section of this guide.

- VS Code requires a Windows, macOS, or Linux operating system.
- The AWS Toolkit for Visual Studio Code requires you to work from VS Code version 1.73.0 or a later version.

For additional information about VS Code or to download the latest version of VS Code, see the VS Code downloads website.

Downloading and installing the AWS Toolkit for Visual Studio Code

You can download, install, and set up the AWS Toolkit for Visual Studio Code through the VS Code Marketplace in your IDE. Alternatively, you can download the AWS Toolkit for Visual Studio Code installation files by navigating to the VS Code Marketplace from your web browser.

Installing the AWS Toolkit for Visual Studio Code from the VS Code IDE Marketplace

1. Open the AWS Toolkit for Visual Studio Code extension in your VS Code IDE with the following link: Open the VS Code Marketplace.

   > **Note**
   >
   > If VS Code is not already running on your machine, this operation may take a few moments while VS Code is loading.

2. From the AWS Toolkit for Visual Studio Code extension in the VS Code Marketplace, choose **Install** to begin the installation process.
3. When prompted, choose to restart VS Code to complete the installation process.

Optional prerequisites

Before you can use certain features of the AWS Toolkit for Visual Studio Code, you must have the following:

- **Amazon Web Services (AWS) account**: An AWS account isn't a requirement to use the AWS Toolkit for Visual Studio Code, but functionality is significantly limited without it. To obtain an AWS account, go to the AWS home page. Choose **Create an AWS Account**, or **Complete Sign Up** (if you've visited the site before).
- **Code Development** – The relevant SDK for the language that you want to use. You can download from the following links, or use your favorite package manager:
  - .NET SDK: https://dotnet.microsoft.com/download
  - Node.js SDK: https://nodejs.org/en/download
  - Python SDK: https://www.python.org/downloads
  - Java SDK: https://aws.amazon.com/corretto/
  - Go SDK: https://golang.org/doc/install

- **AWS SAM CLI** – This is an AWS CLI tool that helps you develop, test, and analyze your serverless applications locally. This isn't required for installing the toolkit. However, we recommend that you install it (and Docker, described next) because it's required for any AWS Serverless Application Model (AWS SAM) functionality, such as Creating a new serverless application (local).

  For more information, see Installing the AWS SAM CLI in the *AWS Serverless Application Model Developer Guide*.
- **Docker** – The AWS SAM CLI requires this open-source software container platform. For more information and download instructions, see Docker.
- **Package Manager** – A package manager so you can download and share application code.
  - .NET: NuGet
  - Node.js: npm
  - Python: pip
  - Java: Gradle or Maven

Connecting to AWS

Most Amazon Web Services (AWS) resources are managed through an AWS account. An AWS account isn't required to use the AWS Toolkit for Visual Studio Code, however Toolkit functions are limited without a connection.

If you've previously set up an AWS account and authentication through another AWS service (such as the AWS Command Line Interface), then the AWS Toolkit for Visual Studio Code automatically detects your credentials.

Sign up for an AWS account

To get started with AWS, you need an AWS account. For information about creating an AWS account, see Getting started with an AWS account in the *AWS Account Management Reference Guide*.

Prerequisites

If you're new to AWS or haven't created an account, then there are 2 main steps to connect the AWS Toolkit for Visual Studio Code with your AWS account:

1. **Setting up authentication**: There are 3 primary methods to authenticate with your AWS account from the AWS Toolkit for Visual Studio Code. To learn more about each of these methods, see the Authentication and Access topic in this User Guide.
2. **Authenticating with AWS from the Toolkit**: You can connect with your AWS account from the Toolkit by completing the procedures in the following sections of this User Guide.

Opening the Sign In panel

Complete one of the following procedures to open the **AWS Toolkit Sign In** panel.

To open the AWS Toolkit Sign In panel from the AWS Explorer:

1. From the AWS Toolkit for Visual Studio Code, expand **EXPLORER**.
2. Expand the **More Actions...** menu by selecting the **...** icon.
3. From the **More Actions...** menu, choose **Connect to AWS** to open the **AWS Toolkit Sign In** panel.

To open the AWS Toolkit Sign In panel using the VS Code command pallet:

1. Open the command pallet by pressing **`Shift+Command+P`** (**`Ctrl+Shift+P`** Windows).
2. Enter **`AWS: Add a New Connection`** into the search field.
3. Select **`AWS: Add a New Connection`** to open the **AWS Toolkit Sign In** panel.

Connecting to AWS from the Toolkit

Authenticate and connect with SSO

To authenticate and connect with AWS using AWS IAM Identity Center, complete the following procedure.

> **Note**
>
> Authentication with AWS Builder ID or IAM Identity Center launches the AWS authorization portal in your default web browser. Each time your credentials expire this process must be repeated to renew the connection between your AWS account and the AWS Toolkit for Visual Studio Code.

Authenticate and connect with AWS IAM Identity Center

1. From the **AWS Toolkit Sign In** panel, choose the **Workforce** tab, then select the **Continue** button to proceed.
2. From the **Sign in with IAM Identity Center** panel, enter the **Start URL** for your organization. This URL is provided to you by an admin or help desk at your company.
3. Select your AWS **Region** from the drop-down menu. This is the AWS region that hosts your identity directory.
4. Choose the **Continue** button and confirm that you want to open the **AWS Authorization request** website in your default web browser.
5. Follow the prompts in your default web browser, you're notified when the authorization process is complete, it's safe to close your browser, and return to VS Code.

Authenticate and connect with IAM Credentials

To authenticate and connect with AWS using IAM Credentials, complete the following procedure.

Authenticate and connect with IAM Credentials

1. From the **AWS Toolkit Sign In** panel, choose **IAM Credential**, then select the **Continue** button to proceed.
2. Enter the **`Profile Name`**, **`Access Key`**, and **`Secret Key`** of your AWS account in the provided fields, then choose the **Continue** button to add the profile to your config file and connect the Toolkit with your AWS account.
3. The Toolkit **AWS Explorer** updates to display your AWS services and resources when authentication is complete and a connection has been established.

Authentication for Amazon CodeCatalyst

To get started working with CodeCatalyst from the Toolkit, authenticate and connect with either your AWS Builder ID or IAM Identity Center credentials.

The following procedures describe how to authenticate and connect the Toolkit with your AWS account.

Authenticate and connect with an AWS Builder ID

1. From the **AWS Toolkit Sign In** panel, choose the **Workforce** tab, then select the **Continue** button to proceed.
2. At the top of the **Sign in with SSO** panel, choose the **Skip to sign-in** link.
3. Follow the prompts in your default web browser, you're notified when the authorization process is complete, it's safe to close your browser, and return to VS Code.

Authenticate and connect with IAM Identity Center

1. From the **AWS Toolkit Sign In** panel, choose the **Workforce** tab, then select the **Continue** button to proceed.
2. From the **Sign in with IAM Identity Center** panel, enter the **Start URL** for your organization. This URL is provided to you by an admin or help desk at your company.
3. Select your AWS **Region** from the drop-down menu. This is the AWS region that hosts your identity directory.
4. Choose the **Continue** button and confirm that you want to open the **AWS Authorization request** website in your default web browser.
5. Follow the prompts in your default web browser, you're notified when the authorization process is complete, it's safe to close your browser, and return to VS Code.

Changing AWS Regions

An AWS Region specifies where your AWS resources are managed. Your default AWS Region is detected when you connect to your AWS account from the AWS Toolkit for Visual Studio Code, automatically displaying in the **AWS Explorer**.

The following sections describe how to add or hide a Region from the **AWS Explorer**.

Adding a Region to the AWS Explorer

Complete the following procedure to add a Region to the AWS Explorer.

1. From VS Code, open the **Command Palette** by expanding **View** on the main menu and choosing **Command Palette**. Or use the following shortcut keys:

   - Windows and Linux – Press **`Ctrl+Shift+P`**.

- macOS – Press **Shift+Command+P**.

2. From the **Command Palette**, search for **AWS: Show or Hide Regions** and choose **AWS: Show or Hide Regions** to display a list of available Regions.
3. From the list, select the AWS Regions that you want to add to the **AWS Explorer**.
4. Choose the **OK** button to confirm your choices and update the **AWS Explorer**.

Hide a Region from the AWS Explorer

To hide a Region from the AWS Explorer view, complete the following procedure.

1. From the **AWS Explorer**, locate the AWS Region that you want to hide.
2. Open the context menu for (right-click) the Region you want to hide.
3. Choose **Show or Hide Regions** to open the **AWS: Show or Hide Regions** options in VS Code.
4. Deselect the Regions that you want to hide in the AWS Explorer view.

Configuring your toolchain

The AWS Toolkit for Visual Studio Code supports multiple languages across all the AWS services. The following sections describe how to configure your toolchain for different languages.

Configure a toolchain for .NET Core

1. Ensure that you have the AWS Toolkit for VS Code installed.
2. Install the C# extension. This extension enables VS Code to debug .NET Core applications.
3. Open an AWS Serverless Application Model (AWS SAM) application, or create one.
4. Open the folder that contains `template.yaml`.

Configure a toolchain for Node.js

1. Ensure that you have the AWS Toolkit for VS Code installed.
2. Open an AWS SAM application, or create one.
3. Open the folder that contains `template.yaml`.

> **Note**
>
> When debugging a TypeScript Lambda function directly from the source code (launch configuration has `"target": "code"`), the TypeScript compiler must be installed either globally or in your project's `package.json`.

Configure a toolchain for Python

1. Ensure that you have the AWS Toolkit for VS Code installed.
2. Install the Python extension for Visual Studio Code. This extension enables VS Code to debug Python applications.
3. Open an AWS SAM application, or create one.
4. Open the folder that contains `template.yaml`.
5. Open a terminal at the root of your application, and configure `virtualenv` by running `python -m venv ./.venv`.

> **Note**
>
> You only need to configure `virtualenv` once per system.

6. Activate `virtualenv` by running one of the following:

   - Linux or macOS (Bash): `source ./.venv/bin/activate`
   - Windows (PowerShell): `./.venv/Scripts/Activate.ps1`
   - Windows (Command Prompt): `.venv\Scripts\activate.bat`

Configure a toolchain for Java

1. Ensure that you have the AWS Toolkit for VS Code installed.
2. Install the Java extension and Java 11. This extension enables VS Code to recognize Java functions.
3. Install the Java debugger extension. This extension enables VS Code to debug Java applications.

4. Open an AWS SAM application, or create one.
5. Open the folder that contains `template.yaml`.

Configure a toolchain for Go

1. Ensure that you have the AWS Toolkit for VS Code installed.
2. Go 1.14 or higher is required for debugging Go Lambda functions.
3. Install the Go extension.

> **Note**
>
> Version 0.25.0 or higher is required for debugging Go1.15+ runtimes.

4. Install Go tools using the command palette:
   a. From the command palette, choose `Go: Install/Update Tools`.
   b. From the set of check boxes, select `dlv` and `gopls`.
5. Open an AWS SAM application, or create one.
6. Open the folder that contains `template.yaml`.

Using Your toolchain

Once you have your toolchain set up, you can use it to run or debug the AWS SAM application.

Authentication and access for the AWS Toolkit for Visual Studio Code

You don't need to authenticate with AWS to start working with the AWS Toolkit for Visual Studio Code. However, most AWS resources are managed through an AWS account. To access all of the AWS Toolkit for Visual Studio Code services and features, you'll need to authenticate with AWS IAM Identity Center, AWS Builder ID or IAM credentials.

The following topics contain additional details about each credential type.

For details about how to connect to AWS in the AWS Toolkit for Visual Studio Code with your existing credentials, see the Connecting to AWS topic in this User Guide.

Topics

AWS IAM Identity Center

AWS IAM Identity Center is the recommended best practice for managing your AWS account authentication.

For detailed instructions on how to set up IAM Identity Center for Software Development Kits (SDKs), see the IAM Identity Center authentication section of the *AWS SDKs and Tools Reference Guide*.

For details on how to authenticate and connect the AWS toolkit with your existing IAM Identity Center credentials, see the Connect to AWS topic in this User Guide.

AWS IAM credentials

AWS IAM credentials authentication with your AWS account through locally stored access keys.

For details about how to authenticate and connect the AWS toolkit with your existing AWS IAM credentials, see the Connect to AWS topic in this User Guide.

The following sections describe how to set up IAM credentials to authenticate with your AWS account from the AWS Toolkit for Visual Studio Code.

Important

Before setting up IAM credentials to authenticate with your AWS account, note that:

- If you've already set IAM credentials through another AWS service (such as the AWS CLI), then the AWS Toolkit for Visual Studio Code automatically detects those credentials and makes them available in VS Code.
- AWS recommends using IAM Identity Center authentication. For additional information about AWS IAM best practices, see the Security best practice in IAM section of the AWS *Identity and Access Management* User Guide.
- To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

Creating an IAM user

Before you can set up the AWS Toolkit for Visual Studio Code to authenticate with your AWS account, you need to complete **Step 1: Create your IAM user** and **Step 2: Get your access keys** in the Authenticate using long-term credentials topic in the *AWS SDKs and Tools Reference Guide*.

Note

Step 3: Update the shared credentials file in the *AWS SDKs and Tools Reference Guide* is optional.
If you complete Step 3, the AWS Toolkit for Visual Studio Code automatically detects your credentials during the the section called “Creating a shared credentials file from the AWS Toolkit for Visual Studio Code” located below.
If you haven't completed Step 3, the AWS Toolkit for Visual Studio Code walks you through the process of creating a `credentials file` as described in the the section called

"Creating a shared credentials file from the AWS Toolkit for Visual Studio Code" located below.

Creating a shared credentials file from the AWS Toolkit for Visual Studio Code

Your *shared config file* and *shared credentials file* store configuration and credential information for your AWS accounts. For more information about shared configuration and credentials, see the Where are configuration settings stored? section in the *AWS Command Line Interface User Guide*.

Creating a shared credentials file through the AWS Toolkit for Visual Studio Code

1. Open the command pallet by pressing **Shift+Command+P** (**Ctrl+Shift+P** Windows).
2. Enter **AWS: Add a New Connection** into the search field.
3. Select **AWS: Add a New Connection** to open the **AWS Toolkit Sign In** panel.
4. From the **AWS Toolkit Sign In** panel, choose **IAM Credential**, then select the **Continue** button to proceed.
5. Enter the **Profile Name**, **Access Key**, and **Secret Key** of your AWS account in the provided fields, then choose the **Continue** button to add the profile to your config file and connect the Toolkit with your AWS account.
6. The Toolkit **AWS Explorer** updates to display your AWS services and resources when authentication is complete and a connection has been established.

Note

In this example, assume that *[Profile_Name]* contains syntax errors and causes authentication to fail.

```
[Profile_Name]
xaws_access_key_id= AKIAI44QH8DHBEXAMPLE
xaws_secret_access_key= wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

The following is an example of a log message that's generated in response to a failed authentication attempt.

```
2022-11-02 22:01:54 [ERROR]: Profile [Profile_Name] is not a valid Credential
 Profile: not supported by the Toolkit
2022-11-02 22:01:54 [WARN]: Shared Credentials Profile [Profile_Name] is not
 valid. It will not be used by the toolkit.
```

Add additional credential profiles

You can add multiple credentials to your configuration files. To do so, open the **Command Palette** and choose **AWS Toolkit Create Credentials Profile**. This will open the credentials file. On this page, you can add a new profile below your first profile, as shown in the following example:

```
# Amazon Web Services Credentials File used by AWS CLI, SDKs, and tools
# This file was created by the AWS Toolkit for Visual Studio Code extension.
#
# Your AWS credentials are represented by access keys associated with IAM users.
# For information about how to create and manage AWS access keys for a user, see:
# https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html
#
# This credential file can store multiple access keys by placing each one in a
# named "profile". For information about how to change the access keys in a
# profile or to add a new profile with a different access key, see:
# https://docs.aws.amazon.com/cli/latest/userguide/cli-config-files.html
#
[Profile1_Name]
# The access key and secret key pair identify your account and grant access to AWS.
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
# Treat your secret key like a password. Never share your secret key with anyone. Do
# not post it in online forums, or store it in a source control system. If your secret
# key is ever disclosed, immediately use IAM to delete the access key and secret key
# and create a new key pair. Then, update this file with the replacement key details.
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
[Profile2_Name]
aws_access_key_id = AKIAI44QH8DHBEXAMPLE
aws_secret_access_key = je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
```

AWS Builder ID for developers

An AWS Builder ID is an additional AWS account that is optional or required for certain AWS services. For detailed information about the AWS Builder ID authentication method, see the Sign in with AWS Builder ID topic in the *AWS Sign-in* User Guide.

For details about how to authenticate and connect the AWS toolkit with your existing AWS Builder ID, see the Connect to AWS topic in this User Guide.

Using an external credential process

You can configure the AWS Toolkit for Visual Studio Code for credential processes that aren't directly supported by AWS, by modifying your `shared config file`.

Modifying your `shared config file` for credential processes is the same for both the AWS Toolkit for Visual Studio Code and the AWS Command Line Interface. For detailed information about how to set up external credentials, see the Sourcing credentials with an external process topic in the *AWS Command Line Interface User Guide*.

Updating firewalls and gateways to allow access

If you filter access to specific AWS domains or URL endpoints by using a web-content filtering solution, the following endpoints must be allow listed in order to access all of the services and features available through the AWS Toolkit for Visual Studio Code and Amazon Q.

AWS Toolkit for Visual Studio Code Endpoints

The following are lists of AWS Toolkit for Visual Studio Code specific endpoints and references that need to be allow listed.

Endpoint

```
https://idetoolkits.amazonwebservices.com/endpoints.json
```

Hosted files

```
https://idetoolkits-hostedfiles.amazonaws.com/Notifications/VSCode/startup/1.x.json
https://idetoolkits-hostedfiles.amazonaws.com/Notifications/VSCode/emergency/1.x.json
```

Schema support

```
https://raw.githubusercontent.com/aws/serverless-application-model/main/samtranslator/
schema/schema.json
https://api.github.com/repos/devfile/api/releases/latest
https://raw.githubusercontent.com/devfile/api/${devfileSchemaVersion}/schemas/latest/
devfile.json
```

cSharpSamDebug install script

```
https://aka.ms/getvsdbgps1
https://aka.ms/getvsdbgsh
```

Amazon Q plugin endpoints

The following is a list of Amazon Q plugin specific endpoints and references that need to be allow listed.

```
https://idetoolkits-hostedfiles.amazonaws.com/*    (Plugin for configs)
https://idetoolkits.amazonwebservices.com/*   (Plugin for endpoints)
https://aws-toolkit-language-servers.amazonaws.com/*  (Language Server Process)
https://client-telemetry.us-east-1.amazonaws.com/ (Telemetry)
https://cognito-identity.us-east-1.amazonaws.com    (Telemetry)
https://aws-language-servers.us-east-1.amazonaws.com (Language Server Process)
```

Amazon Q Developer endpoints

The following is a list of Amazon Q Developer specific endpoints and references that need to be allow listed.

```
https://codewhisperer.us-east-1.amazonaws.com (Inline,Chat, QSDA,...)
https://q.us-east-1.amazonaws.com (Inline,Chat, QSDA....)
https://desktop-release.codewhisperer.us-east-1.amazonaws.com/ (Download url for CLI.)
https://specs.q.us-east-1.amazonaws.com (Url for autocomplete specs used by CLI)
* aws-language-servers.us-east-1.amazonaws.com (Local Workspace context)
```

Amazon Q Code Transform Endpoints

The following is a list of Amazon Q Code Transform specific endpoints and references that need to be allow listed.

```
https://docs.aws.amazon.com/amazonq/latest/qdeveloper-ug/security_iam_manage-access-with-policies.html
```

Authentication endpoints

The following is a list of authentication endpoints and references that need to be allow listed.

```
[Directory ID or alias].awsapps.com
* oidc.[Region].amazonaws.com
*.sso.[Region].amazonaws.com
*.sso-portal.[Region].amazonaws.com
*.aws.dev
*.awsstatic.com
*.console.aws.a2z.com
*.sso.amazonaws.com
```

Identity Endpoints

The following lists contain endpoints that are specific to identity, such as AWS IAM Identity Center and AWS Builder ID.

AWS IAM Identity Center

For details on required endpoints for IAM Identity Center, see the Enable IAM Identity Center topic in the *AWS IAM Identity Center* User Guide.

Enterprise IAM Identity Center

```
https://[Center director id].awsapps.com/start (should be permitted to initiate auth)
https://us-east-1.signin.aws (for facilitating authentication, assuming IAM Identity
 Center is in IAD)
https://oidc.(us-east-1).amazonaws.com
https://log.sso-portal.eu-west-1.amazonaws.com.
https://portal.sso.eu-west-1.amazonaws.com
```

AWS Builder ID

```
https://view.awsapps.com/start (must be blocked to disable individual tier)
https://codewhisperer.us-east-1.amazonaws.com and q.us-east-1.amazonaws.com (should be
 permitted)
```

Telemetry

The following is a Telemetry specific endpoints that needs to be allow listed.

```
https://telemetry.aws-language-servers.us-east-1.amazonaws.com/
https://client-telemetry.us-east-1.amazonaws.com
```

References

The following is a list of endpoint references.

```
idetoolkits-hostedfiles.amazonaws.com.
cognito-identity.us-east-1.amazonaws.com.
amazonwebservices.gallery.vsassets.io.
eu-west-1.prod.pr.analytics.console.aws.a2z.com.
prod.pa.cdn.uis.awsstatic.com.
portal.sso.eu-west-1.amazonaws.com.
log.sso-portal.eu-west-1.amazonaws.com.
```

```
prod.assets.shortbread.aws.dev.
prod.tools.shortbread.aws.dev.
prod.log.shortbread.aws.dev.
a.b.cdn.console.awsstatic.com.
assets.sso-portal.eu-west-1.amazonaws.com.
oidc.eu-west-1.amazonaws.com.
aws-toolkit-language-servers.amazonaws.com.
aws-language-servers.us-east-1.amazonaws.com.
idetoolkits.amazonwebservices.com.
```

Working with AWS services and tools

The AWS Toolkit for Visual Studio Code makes AWS services, tools, and resources available to you, directly in VS Code. The following is a list of guide topics covering each Toolkit for VS Code service and their features. Choose a service or tool for more information on what it does, how to set it up, and working with basic features.

Topics

- Working with serverless applications
- Working with Systems Manager Automation documents
- AWS Step Functions
- Working with Threat Composer
- Working with resources

Working with experimental features

Experimental features offer early access to features in the AWS Toolkit for Visual Studio Code before they're officially released.

> **Warning**
>
> Because experimental features continue to be tested and updated, they may have usability issues. And experimental features may be removed from the AWS Toolkit for Visual Studio Code without notice.

You can enable experimental features for specific AWS services in the **AWS Toolkit** section of the **Settings** pane in your VS Code IDE.

1. To edit AWS settings in VS Code, choose **File**, **Preferences**, **Settings**.
2. In the **Settings** pane, expand **Extensions** and choose **AWS Toolkit**.
3. Under **AWS: Experiments**, select the checkboxes for the experimental features you want to access prior to release. If you want to switch off an experimental feature, clear the relevant checkbox.

Working with AWS Services in the AWS Explorer

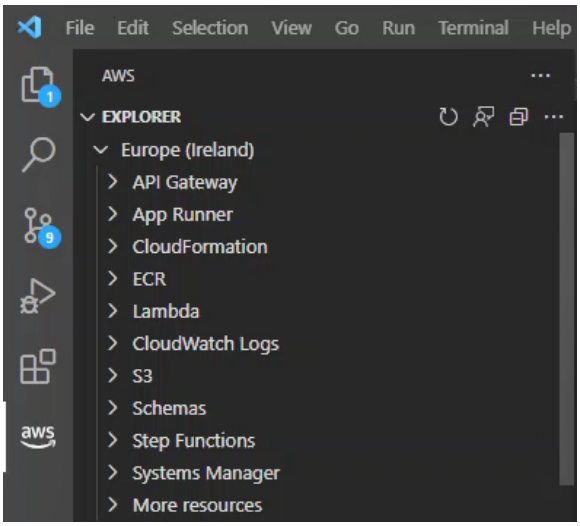
The **AWS Explorer** gives you a view of some of the AWS services that you can work with when using the AWS Toolkit for Visual Studio Code.

This section provides information about how to access and use the **AWS Explorer** in VS Code. It assumes that you've already installed and configured the Toolkit for VS Code on your system.

Some important points:

- If the toolkit is installed and configured correctly, you should see items in the **AWS Explorer**. To see the **AWS Explorer**, choose the **AWS** icon in the **Activity bar**.

  For example:



- Certain features require certain AWS permissions. For example, to see the AWS Lambda functions in your AWS account, the credentials you configured in Authentication and access must include at least read-only Lambda permissions. See the following topics for more information about the permissions that each feature needs.
- If you want to interact with AWS services that aren't immediately visible in the **AWS Explorer**, you can go to **More resources** and choose from hundreds of resources that can added to the interface.

  For example, you can choose **AWS Toolkit:CodeArtifact::Repository** from the selection of available resource types. After this resource type is added to **More resources**, you can expand the entry to view a list of resources that create different CodeArtifact repositories with their own properties and attributes. Moreover, you can describe the properties and attributes of resources in JSON-formatted templates, which can be saved to create new resources in the AWS Cloud.

AWS Documents

The AWS Toolkit for Visual Studio Code supports the AWS Serverless Application Model `JSON Schema` for `AWS SAM templates`, enhancing the template authoring experience by enabling definitions, autocompletion, and validation directly in VS Code. AWS Documents supports all AWS SAM and CloudFormation resources. For additional details see the following resources:

- For specific information about JSON Schema, see the JSON Schema JSON-Schema.org website.
- For additional information about AWS SAM templates, see the AWS SAM template anatomy topic in the *AWS Serverless Application Model* Developer Guide.
- For additional information about AWS resources and property types, see the AWS resource and property types reference topic in the *CloudFormation* User Guide.
- For detailed information about the AWS SAM schema utilized by the AWS Toolkit, see the AWS Serverless Application Model schema in the AWS GitHub repository.

Getting Started with AWS Documents

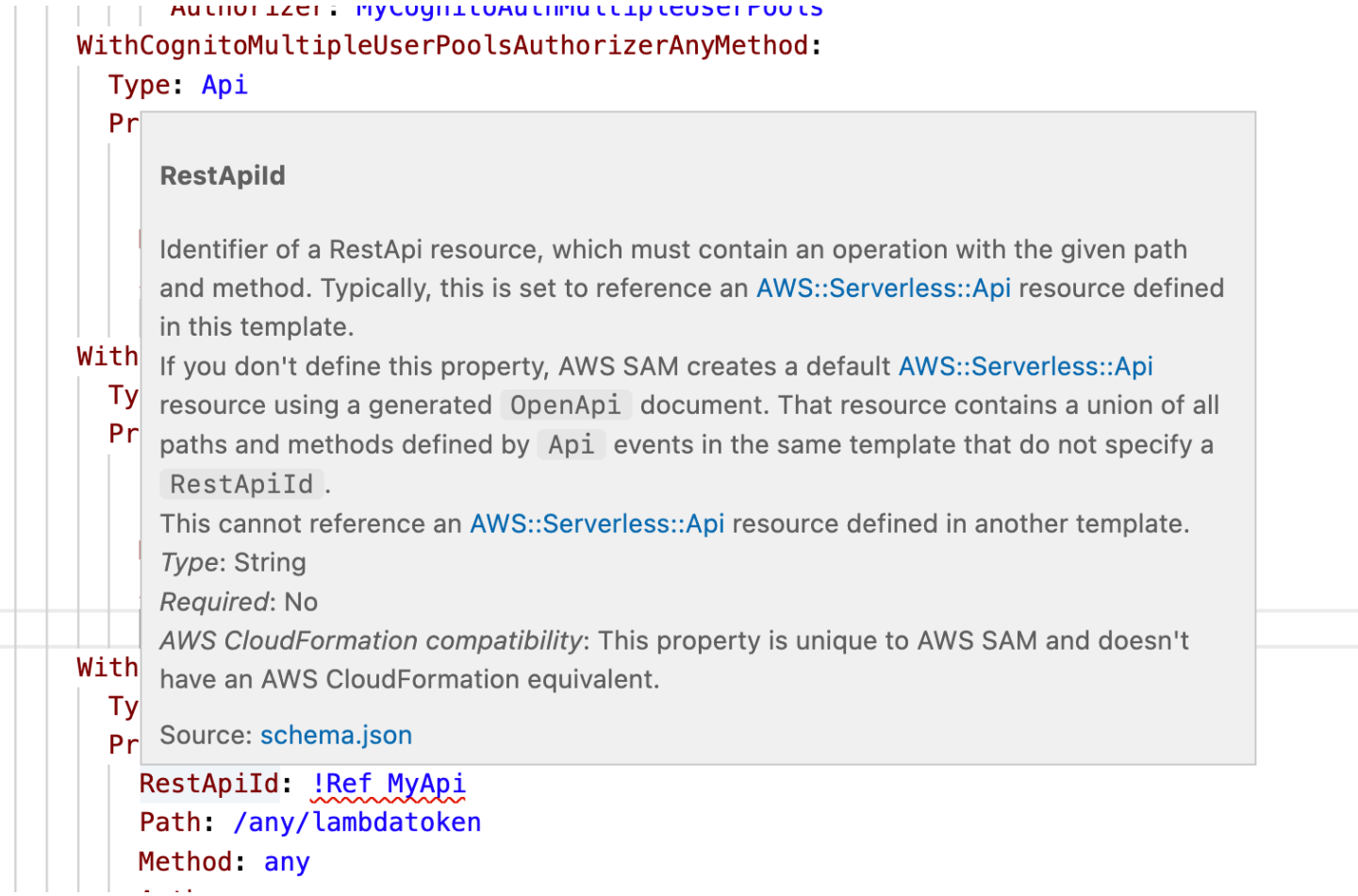
To get started working with AWS Documents in VS Code, install the AWS Toolkit for Visual Studio Code extension from your IDE or the VS Code Marketplace, and open any AWS SAM template.

Viewing documentation, autocompletion, and validation in VS Code

Viewing documentation, autocompletion, and validation are features included with the AWS Toolkit. See the image below for an example of what these features look like in VS Code.

- To view documentation from your open AWS SAM template, hover your pointer over a line-entry in the document.
- For autocompletion, start typing in your AWS SAM template to activate a pop-up with suggestions based on your input.
- Your AWS SAM template is automatically scanned for validation and errors are highlighted by a light bulb icon that you can select for additional suggestions.

See the image below for an example of what these features look like in VS Code.



Amazon CodeCatalyst for VS Code

What is Amazon CodeCatalyst?

Amazon CodeCatalyst is a cloud-based collaboration space for software development teams. Through the AWS Toolkit for Visual Studio Code, you can view and manage your CodeCatalyst resources directly from VS Code. You can also work directly in the cloud by using the AWS Toolkit to launch, Dev Environments virtual computing environments running VS Code. For more information about the CodeCatalyst service, see the Amazon CodeCatalyst User Guide.

The following topics describe how to connect VS Code with CodeCatalyst, and how to work with CodeCatalyst from the Toolkit for VS Code.

Topics

- Getting started with CodeCatalyst and the Toolkit for VS Code
- Working with Amazon CodeCatalyst resources in VS Code

- Working with the Toolkit in a Dev Environments
- Troubleshooting Amazon CodeCatalyst and VS Code

Getting started with CodeCatalyst and the Toolkit for VS Code

To get started working with CodeCatalyst in VS Code, follow these procedures.

Topics

- Creating a CodeCatalyst account
- Connecting the AWS Toolkit with CodeCatalyst

Creating a CodeCatalyst account

You must have active AWS Builder ID or AWS IAM Identity Center credentials to connect to CodeCatalyst from the Toolkit for VS Code. To learn more about AWS Builder ID, IAM Identity Center, and CodeCatalyst credentials, see the Setting up with CodeCatalyst section in the *CodeCatalyst* User Guide.

Connecting the AWS Toolkit with CodeCatalyst

To connect the AWS Toolkit with your CodeCatalyst account, see the Authentication for Amazon CodeCatalyst section in the *Connecting to AWS* topic of this User Guide.

Working with Amazon CodeCatalyst resources in VS Code

The following sections provide an overview of the Amazon CodeCatalyst resource management features that are available from the Toolkit for VS Code.

For more information about Dev Environments and how you can access them from CodeCatalyst, see the Dev Environments section in the *Amazon CodeCatalyst* User Guide.

The following sections describe how to create, open, and work with Dev Environments from VS Code.

Topics

- Clone a repository
- Opening a Dev Environment
- Creating a CodeCatalyst Dev Environment

- Creating a Dev Environment from a third-party repository
- CodeCatalyst commands in VS Code

Clone a repository

CodeCatalyst is a cloud-based service that requires you to be connected to the cloud to work on CodeCatalyst projects. If you prefer to work on a project locally, you can clone your CodeCatalyst repositories to your local machine and sync it with your CodeCatalyst project online, the next time that you're connected to the cloud.

To clone a repository from your CodeCatalyst account to VS Code with the AWS Toolkit, complete the following steps:

> **Note**
>
> If you are cloning a repository from a 3rd party service, you may be prompted to authenticate with that service's credentials.
> While the repository is being cloned, VS Code displays the progress in the **Cloning Repository** status window. After the repository is cloned, the **Would you like to open the cloned repository?** message appears.

1. From the Toolkit for VS Code, expand the **DEVELOPER TOOLS** explorer.
2. Expand **CodeCatalyst**, choose **Clone Repository**.
3. From the **Select a CodeCatalyst Repository** dialog, search for the repository that you want to clone, then select it to open the **Choose a folder to clone** dialog.
4. Choose **Select Repository Location** to close the prompt and begin cloning the repository.
5. From the dialog window, choose one of the following to complete the cloning process:
   - To open your repository in your current VS Code window, choose **Open**.
   - To open your repository in a new VS Code window, choose **Open in new window**.
   - To complete the cloning process without opening your repository, close the dialog window.

Opening a Dev Environment

To open an existing Dev Environment in VS Code, complete the following steps.

Note

Selecting the Dev Environment starts the process to connect VS Code with CodeCatalyst by opening your Dev Environment. During this process, VS Code displays progress updates in a CodeCatalyst status window. The status window updates when the process is complete.

- If the Dev Environment fails to open, the status updates with information about why the process failed and a link to open the process logs.
- If the process is successful, your Dev Environment opens in a new window, from VS Code.

1. From the Toolkit for VS Code, expand the **DEVELOPER TOOLS** explorer.
2. Expand **CodeCatalyst** and choose **Open Dev Environment** to open the **Select a CodeCatalyst Dev Environment** dialog in VS Code.
3. From the **Select a CodeCatalyst Dev Environment** dialog, choose the **Dev Environment** that you want to open.

Creating a CodeCatalyst Dev Environment

To create a new Dev Environment, complete the following steps:

Note

When creating a new Dev Environment, observe the following:

- AWS recommends that you specify an alias because it simplifies organization and improves search capabilities for Dev Environments.
- Dev Environments saves your work persistently. This means that your Dev Environment can be stopped without losing your work. Stopping your Dev Environment reduces the costs that are required to keep you Dev Environment up and running.
- **Storage** is the only setting that can't be changed after your Dev Environment has been created.
- VS Code displays the progress of your Dev Environment being created in a status window. After the Dev Environment is created, VS Code opens the Dev Environment in a new window and the **Do you trust the authors of the files in this folder?** prompt

also appears. Agree to the terms and conditions to continue working in your Dev Environment.

1. From the Toolkit for VS Code, expand the **DEVELOPER TOOLS** explorer.
2. Expand **CodeCatalyst**, and choose the **Create Dev Environment** option to open the **Create a CodeCatalyst Dev Environment** menu in VS Code.
3. From the **Source Code** section, choose one of the following options:
   - **Use an existing CodeCatalyst Repository**: Creates a Dev Environment from an existing CodeCatalyst repository. You must select the CodeCatalyst **Project** and **Branch**.
   - **Create an empty Dev Environment**: Creates an empty Dev Environment.
4. (Optional) From the **Alias** section, enter an alternate name for your Dev Environment.
5. (Optional) From the **Dev Environments Configuration** section, change the following settings to meet your specific needs.
   - **Compute**: Choose **Edit Compute** to change the amount of processing power and RAM that's assigned to your system.
   - **Timeout**: Choose **Edit Timeout** to change the amount of system idle time allowed before your Dev Environment is stopped.
   - **Storage**: Choose **Edit Storage Size** to change the amount of storage space that's assigned to your system.
6. Choose **Create Dev Environment** to create your new cloud development environment.

Creating a Dev Environment from a third-party repository

You can create Dev Environments from a third-party repository by linking to the repository as a source.

Linking to a third-party repository as a source is handled at the project level in CodeCatalyst. For instructions and additional details on how to connect a third-party repository to your Dev Environment, see the Linking a source repository topic in the *Amazon CodeCatalyst* User Guide.

CodeCatalyst commands in VS Code

There are additional VS Code commands that are assigned to CodeCatalyst-related features that aren't displayed directly in the AWS Toolkit.

To view a list of commands that are assigned to CodeCatalyst from the command palette, complete the following steps:

1. From the Toolkit for VS Code, expand the **DEVELOPER TOOLS** explorer.
2. Choose **Show CodeCatalyst Commands** to open the **Command Palette** with a pre-populated search for `CodeCatalyst`.
3. Select a CodeCatalyst command from the list to activate it.

Working with the Toolkit in a Dev Environments

Dev Environments are virtual computing environments for Amazon CodeCatalyst. The following sections describe how to create, launch, and work from Dev Environments using the AWS Toolkit for Visual Studio Code.

For detailed information about Dev Environments, see the Dev Environments topic in the *Amazon CodeCatalyst* User Guide.

Configuring your Dev Environment with devfiles

The `devfile` specification is an open-standard format for YAML that can be used to define configurations for Dev Environments. Every Dev Environment has a devfile. If you create a Dev Environment without a repository or from a repository that doesn't contain a devfile, a default is applied to the source automatically. Devfiles can be updated from CodeCatalyst or your IDE. The processes to update a devfile in a local or remote instance of VS Code are identical, but if you update a devfile locally, you must push the updates to your source repository before the updates take effect.

For detailed information about configuring Dev Environments with devfiles, see the Configuring your Dev Environment topic in the *Amazon CodeCatalyst* User Guide.

The following procedure describes how to edit your devfile from a remote instance of the Toolkit while it's running in a Dev Environment.

> **Important**
>
> If you edit the `Devfile` from VS Code, be aware of the following:
>
> - Changing the name of the devfile or the devfile component name replaces the contents of your root directory. All previous content is lost and unrecoverable.

- If you create a Dev Environment without a devfile in the root folder or a Dev Environment that's not associated with a source repository, a devfile with default configuration settings is generated for your Dev Environment when you create it.
- For instructions on how to define and configure your `Devfile`, see the Adding Commands documentation on the devfile.io website.

1. From the Toolkit for VS Code, expand the **DEVELOPER TOOLS** explorer.
2. Expand **CodeCatalyst** and choose **Open Devfile** to open `devfile.yaml` in a new editor window, within your current Dev Environment.
3. From the VS Code editor, update your devfile, then save your changes.
4. The next time you launch your Dev Environment, the configuration is updated to match the specifications that are defined in your `Devfile`.

Authenticating and connecting to AWS from your Dev Environment

To access all of your AWS resources from your Dev Environment, you must authenticate and connect your remote instance of the Toolkit with your AWS account. The remote instance of the Toolkit automatically authenticates with the credentials inherited from your local instance of the Toolkit when your Dev Environment is launched.

The procedures to update your credentials for a remote instance of the Toolkit are identical to the authentication experience in your local instance of the Toolkit. For detailed instructions on how to update credentials, authenticate, and connect to AWS from the Toolkit, see the Connecting to AWS section in the *Getting started* topic of this User Guide.

For additional information about each of the AWS authentication methods compatible with the AWS Toolkit for Visual Studio Code, see the Authentication and access topic in this User Guide.

Working with the Toolkit for VS Code in Dev Environments

After you open or create a Dev Environment in VS Code, you can work from the Toolkit for VS Code, similar to how you can from a local instance of VS Code. Dev Environments running VS Code are configured to automatically install the AWS Toolkit and connect with your AWS Builder ID.

Stopping a Dev Environment

To stop your current Dev Environment:

1. From the Toolkit for VS Code, expand the **DEVELOPER TOOLS** explorer.
2. Expand **CodeCatalyst** and choose **Stop Dev Environment**.
3. When prompted by VS Code, confirm that you want to stop your Dev Environment.
4. Your Dev Environment has successfully stopped when VS Code closes the remote connection and returns to a local development instance.

Opening Dev Environment settings

To open the settings for your current Dev Environment, complete the following steps:

> **Note**
>
> You can't change the amount of storage space assigned to your Dev Environment after it has been created.

1. From the Toolkit for VS Code, expand the **DEVELOPER TOOLS** explorer.
2. Expand **CodeCatalyst** and choose **Open Settings** to open the **Dev Environment Settings** view, for your current Dev Environment.
3. From the **Dev Environment Settings** view, the following sections contain options for your Dev Environment:
   - **Alias**: View and change the **Alias** assigned to your Dev Environment.
   - **Status**: View your current Dev Environment status, the project it's assigned to, and stop your environment.
   - **Devfile**: View the name and location of the `Devfile` for your Dev Environment. Open your `Devfile` by choosing the **Open in Editor** button.
   - **Compute Settings**: Change the size and default **Timeout Length** for your Dev Environment.

Troubleshooting Amazon CodeCatalyst and VS Code

The following topics address potential technical issues when working with Amazon CodeCatalyst and VS Code.

Topics

VS Code version

Your version of VS Code is expected to set up a handler for `vscode://` URIs on your system. Without this handler, you can't access all CodeCatalyst features from the AWS Toolkit. For example, you encounter an error when launching a Dev Environment from VS Code Insiders. This is because VS Code Insiders handles `vscode-insiders://` URIs and doesn't handle `vscode://` URIs.

Permissions for Amazon CodeCatalyst

The following are file permission requirements for working with CodeCatalyst from the AWS Toolkit for Visual Studio Code:

- Set your own access permissions for your `~/.ssh/config` file to `read` and `write`. Restrict `write` permissions for all other users.
- Set your access permissions for the `~/.ssh/id_dsa` and `~/.ssh/id_rsa` files to `read` only. Restrict `read`, `write` and `execute` permissions for all other users.
- Your `globals.context.globalStorageUri.fsPath` file must be in a writable location.

Connecting to a Dev Environment from the Toolkit for VS Code

If you receive the following error when attempting to connect to a Dev Environment from the AWS Toolkit for Visual Studio Code:

Your `~/.ssh/config` has an `aws-devenv-*` section that might be out of date.

- Choose the **Open config. . .** button to open your `~/.ssh/config` file in the VS Code **Editor**.
- From the **Editor**, select and delete the contents of the `Host aws-devenv-*` section.
- Save the changes you made to the `Host aws-devenv-*` of `~/.ssh/config`. Then, close the file.
- Reattempt to connect to a Dev Environment from the Toolkit for VS Code.

Working with Amazon API Gateway

You can browse and run remote API Gateway resources in your connected AWS account using the AWS Toolkit for Visual Studio Code.

> **Note**
>
> This feature does not support debugging.

To browse and run remote API Gateway resources

1. In the **AWS Explorer**, choose **API Gateway** to expand the menu. The remote API Gateway resources are listed.
2. Locate the API Gateway resource you want to invoke, open its context (right-click) menu, and then choose **Invoke on AWS**.
3. In the parameters form, specify the invoke parameters.
4. To run the remote API Gateway resource, choose **Invoke**. The results are deplayed in the **VS Code Output** view.

Using AWS App Runner with AWS Toolkit for Visual Studio Code

AWS App Runner provides a fast, simple, and cost-effective way to deploy from source code or a container image directly to a scalable and secure web application in the AWS Cloud. Using it, you don't need to learn new technologies, decide which compute service to use, or know how to provision and configure AWS resources.

You can use AWS App Runner to create and manage services based on a *source image* or *source code*. If you use a source image, you can choose a public or private container image that's stored in an image repository. App Runner supports the following image repository providers:

- Amazon Elastic Container Registry (Amazon ECR): Stores private images in your AWS account.
- Amazon Elastic Container Registry Public (Amazon ECR Public): Stores publicly readable images.

If you choose the source code option, you can deploy from a source code repository that's maintained by a supported repository provider. Currently, App Runner supports GitHub as a source code repository provider.

Prerequisites

To interact with App Runner using the AWS Toolkit for Visual Studio Code requires the following:

- An AWS account
- A version of AWS Toolkit for Visual Studio Code that features AWS App Runner

In addition to those core requirements, make sure that all relevant IAM users have permissions to interact with the App Runner service. Also you need to obtain specific information about your service source such as the container image URI or the connection to the GitHub repository. You need this information when creating your App Runner service.

Configuring IAM permissions for App Runner

The easiest way to grant the permissions that are required for App Runner is to attach an existing AWS managed policy to the relevant AWS Identity and Access Management (IAM) entity, specifically a user or group. App Runner provides two managed policies that you can attach to your IAM users:

- `AWSAppRunnerFullAccess`: Allows users to perform all App Runner actions.
- `AWSAppRunnerReadOnlyAccess`: Allow users to list and view details about App Runner resources.

In addition, if you choose a private repository from the Amazon Elastic Container Registry (Amazon ECR) as the service source, you must create the following access role for your App Runner service:

- `AWSAppRunnerServicePolicyForECRAccess`: Allows App Runner to access Amazon Elastic Container Registry (Amazon ECR) images in your account.

You can create this role automatically when configuring your service instance with VS Code's **Command Palette**.

Note

The **AWSServiceRoleForAppRunner** service-linked role allows AWS App Runner to complete the following tasks:

- Push logs to Amazon CloudWatch Logs log groups.
- Create Amazon CloudWatch Events rules to subscribe to Amazon Elastic Container Registry (Amazon ECR) image push.

You don't need to manually create the service-linked role. When you create an AWS App Runner in the AWS Management Console or by using API operations that are called by AWS Toolkit for Visual Studio Code, AWS App Runner creates this service-linked role for you.

For more information, see Identity and access management for App Runner in the *AWS App Runner Developer Guide*.

Obtaining service sources for App Runner

You can use AWS App Runner to deploy services from a source image or source code.

Source image

If you're deploying from a source image, you can obtain a link to the repository for that image from a private or public AWS image registry.

- Amazon ECR private registry: Copy the URI for a private repository that uses the Amazon ECR console at https://console.aws.amazon.com/ecr/repositories.
- Amazon ECR public registry: Copy the URI for a public repository that uses the Amazon ECR Public Gallery at https://gallery.ecr.aws/.

Note

You can also obtain the URI for a private Amazon ECR repository directly from **AWS Explorer** in Toolkit for VS Code:

- Open **AWS Explorer** and expand the **ECR** node to view the list of repositories for that AWS Region.

- Right-click a repository and choose **Copy Repository URI** to copy the link to your clipboard.

You specify the URI for the image repository when configuring your service instance with VS Code's **Command Palette**

For more information, see App Runner service based on a source image in the *AWS App Runner Developer Guide*.

Source code

For your source code to be deployed to an AWS App Runner service, that code must be stored in a Git repository that's maintained by a supported repository provider. App Runner supports one source code repository provider: GitHub.

For information about setting up a GitHub repository, see the Getting started documentation on GitHub.

To deploy your source code to an App Runner service from a GitHub repository, App Runner establishes a connection to GitHub. If your repository is private (that is, it isn't publicly accessible on GitHub), you must provide App Runner with connection details.

Important

To create GitHub connections, you must use the App Runner console (https://console.aws.amazon.com/apprunner) to create a connection that links GitHub to AWS. You can select the connections that are available on the **GitHub connections** page when configuring your service instance with VS Code's **Command Palette**.
For more information, see Managing App Runner connections in the *AWS App Runner Developer Guide*.

The App Runner service instance provides a managed runtime that allows your code to build and run. AWS App Runner currently supports the following runtimes:

- Python managed runtime
- Node.js managed runtime

As part of your service configuration, you provide information about how the App Runner service builds and starts your service. You can enter this information using the **Command Palette** or specify a YAML-formatted App Runner configuration file. Values in this file instruct App Runner how to build and start your service, and provide runtime context. This includes relevant network settings and environment variables. The configuration file is named `apprunner.yaml`. It's automatically added to root directory of your application's repository.

Pricing

You're charged for the compute and memory resources that your application uses. In addition, if you automate your deployments, you also pay a set monthly fee for each application that covers all automated deployments for that month. If you opt to deploy from source code, you additionally pay a build fee for the amount of time that it takes App Runner to build a container from your source code.

For more information, see AWS App Runner Pricing.

Topics

- Creating App Runner services
- Managing App Runner services

Creating App Runner services

You can create an App Runner service in Toolkit for VS Code by using the **AWS Explorer** and VS Code's **Command Palette**. After you choose to create a service in a specific AWS Region, numbered steps provided by the **Command Palette** guide you through the process of configuring the service instance where your application runs.

Before creating an App Runner service, make sure that you've completed the prerequisites. This includes providing the relevant IAM permissions and confirming the specific source repository that you want to deploy.

To create an App Runner service

1. Open AWS Explorer, if it isn't already open.
2. Right-click the **App Runner** node and choose **Create Service**.

The **Command Palette** displays.

3. For **Select a source code location type**, choose **ECR** or **Repository**.

   If you choose **ECR**, you specify a container image in a repository maintained by Amazon Elastic Container Registry. If you choose **Repository**, you specify a source code repository that's maintained by a supported repository provider. Currently, App Runner supports GitHub as a source code repository provider.

Deploying from ECR

1. For **Select or enter an image repository**, choose or enter the URL of the image repository that's maintained by your Amazon ECR private registry or the Amazon ECR Public Gallery.

   > **Note**
   >
   > If you specify a repository from the Amazon ECR Public Gallery, make sure that automatic deployments are turned off because App Runner doesn't support automatic deployments for an image in an ECR Public repository.
   > Automatic deployments are switched off by default, and this is indicated when the icon on the **Command Palette** header features a diagonal line through it. If you chose to switch on automatic deployments, a message informs you that this option can incur additional costs.

2. If the **Command Palette** step reports that **No tags found**, you need to go back a step to select a repository that contains a tagged container image.
3. If you're using an Amazon ECR private registry, you require the ECR access role, **AppRunnerECRAccessRole**, that allows App Runner to access Amazon Elastic Container Registry (Amazon ECR) images in your account. Choose the "+" icon on the **Command Palette** header to automatically create this role. (An access role isn't required if your image is stored in Amazon ECR Public, where images are publicly available.)
4. For **Port**, enter the IP port that's used by the service (Port `8000`, for example).
5. For **Configure environment variables**, you can specify a file that contains environment variables that are used to customize behavior in your service instance. Or you can skip this step.
6. For **Name your service**, enter a unique name without spaces and press **Enter**.

7. For **Select instance configuration**, choose a combination of CPU units and memory in GB for your service instance.

   When your service is being created, its status changes from **Creating** to **Running**.
8. After your service starts running, right-click it and choose **Copy Service URL**.
9. To access your deployed application, paste the copied URL into the address bar of your web browser.

Deploying from a remote repository

1. For **Select a connection**, choose a connection that links GitHub to AWS. The connections that are available for selection are listed on the **GitHub connections** page on the App Runner console.
2. For **Select a remote GitHub repository**, choose or enter a URL for the remote repository.

   Remote repositories that are already configured with Visual Studio Code's source control management (SCM) are available for selection. You can also paste a link to the repository if it's not listed.
3. For **Select a branch**, choose which Git branch of your source code that you want to deploy.
4. For **Choose configuration source**, specify how you want to define your runtime configuration.

   If you choose **Use configuration file**, your service instance is configured by settings that are defined by the `apprunner.yaml` configuration file. This file is in the root directory of your application's repository.

   If you choose **Configure all settings here**, use the **Command palette** to specify the following:

   - **Runtime**: Choose **Python 3** or **Nodejs 12**.
   - **Build command**: Enter the command to build your application in the runtime environment of your service instance.
   - **Start command**: Enter the command to start your application in the runtime environment of your service instance.
5. For **Port**, enter the IP port that's used by the service (Port `8000`, for example).
6. For **Configure environment variables**, you can specify a file that contains environment variables that are used to customize behavior in your service instance. Or you can skip this step.

7. For **Name your service**, enter a unique name without spaces and press **Enter**.
8. For **Select instance configuration**, choose a combination of CPU units and memory in GB for your service instance.

   When your service is being created, its status changes from **Creating** to **Running**.
9. After your service starts running, right-click it and choose **Copy Service URL**.
10. To access your deployed application, paste the copied URL into the address bar of your web browser.

> **Note**
>
> If your attempt to create an App Runner service fails, the service shows a status of **Create failed** in **AWS Explorer**. For troubleshooting tips, see When service creation fails in the *App Runner Developer Guide*.

Managing App Runner services

After creating an App Runner service, you can manage it by using the AWS Explorer pane to carry out the following activities:

- Pausing and resuming App Runner services
- Deploying App Runner services
- Viewing logs streams for App Runner
- Deleting App Runner services

Pausing and resuming App Runner services

If you need to disable your web application temporarily and stop the code from running, you can pause your AWS App Runner service. App Runner reduces the compute capacity for the service to zero. When you're ready to run your application again, resume your App Runner service. App Runner provisions new compute capacity, deploys your application to it, and runs the application.

> **Important**
>
> You're billed for App Runner only when it's running. Therefore, you can pause and resume your application as needed to manage costs. This is particularly helpful in development and testing scenarios.

To pause your App Runner service

1. Open AWS Explorer, if it isn't already open.
2. Expand **App Runner** to view the list of services.
3. Right-click your service and choose **Pause**.
4. In the dialog box that displays, choose **Confirm**.

   While the service is pausing, the service status changes from **Running** to **Pausing** and then to **Paused**.

To resume your App Runner service

1. Open AWS Explorer, if it isn't already open.
2. Expand **App Runner** to view the list of services.
3. Right-click your service and choose **Resume**.

   While the service is resuming, the service status changes from **Resuming** to **Running**.

Deploying App Runner services

If you choose the manual deployment option for your service, you need to explicitly initiate each deployment to your service.

1. Open AWS Explorer, if it isn't already open.
2. Expand **App Runner** to view the list of services.
3. Right-click your service and choose **Start Deployment**.
4. While your application is being deployed, the service status changes from **Deploying** to **Running**.

5. To confirm that your application is successfully deployed, right-click the same service and choose **Copy Service URL**.
6. To access your deployed web application, paste the copied URL into the address bar of your web browser.

Viewing logs streams for App Runner

Use CloudWatch Logs to monitor, store, and access your log streams for services such as App Runner. A log stream is a sequence of log events that share the same source.

1. Expand **App Runner** to view the list of service instances.
2. Expand a specific service instance to view the list of log groups. (A log group is a group of log streams that share the same retention, monitoring, and access control settings.)
3. Right-click a log group and choose **View Log Streams**.
4. From the **Command Palette**, choose a log stream from the group.

   The VS Code editor displays the list of log events that make up the stream. You can choose to load older or newer events into the editor.

Deleting App Runner services

> **Important**
>
> If you delete your App Runner service, it's permanently removed and your stored data is deleted. If you need to recreate the service, App Runner needs to fetch your source again and build it if it's a code repository. Your web application gets a new App Runner domain.

1. Open AWS Explorer, if it isn't already open.
2. Expand **App Runner** to view the list of services.
3. Right-click a service and choose **Delete Service**.
4. In the **Command Palette**, enter *delete* and then press **Enter** to confirm.

   The deleted service displays the **Deleting** status, and then the service disappears from the list.

AWS Application Builder

AWS Application Builder for the AWS Toolkit for Visual Studio Code is your guide to building projects visually, iterating on them locally, and deploying your applications to AWS.

The following topics describe how to work with AWS Application Builder from the AWS Toolkit for Visual Studio Code.

Topics

Working with AWS Application Builder

The following sections describe how to access AWS Application Builder in the AWS Toolkit for Visual Studio Code. With Application Builder, you can build projects visually, iterate on them locally, and deploy them to AWS. For an overview of features and potential use cases for Application Builder and your local AWS Lambda experience, see the AWS Developer YouTube video *New* AWS Lambda Local IDE Experience!.

Working with the AWS Application Builder explorer

To access Application Builder in the AWS Toolkit, open the AWS Toolkit in VS Code, then expand the **AWS Application Builder** explorer. The AWS Application Builder explorer contains a link to open the **Walkthrough of Application Builder** in a VS Code editor tab, and displays folders within your current VS Code workspace that contain AWS Application Builder related resources.

From the Application Builder explorer in the AWS Toolkit, there are 4 project-folder-level actions that are accessible from the button icons located next to your project folder or by opening the context menu for (right-clicking) the project folder:

- **Open Template File**: Opens your template file in the VS Code explorer.
- **Open with Infrastructure Composer**: Opens your template file with AWS Infrastructure Composer in the VS Code editor. For detailed information about working with AWS Infrastructure Composer, see the What is AWS Infrastructure Composer topic in the *AWS Infrastructure Composer* Developer Guide.
- **Build SAM Template**: Opens the **Specify parameters for build** dialog in the AWS Toolkit. You can choose to **Specify build flags** for the build or **Use default values from samconfig**. For

detailed information about AWS SAM templates, see the Template anatomy topic in the *AWS Serverless Application Model* Developer Guide.

- **Deploy SAM Application**: Opens the **Select deployment command** dialog in VS Code where your can choose to **Deploy** your application or **Sync**, to update an application you've already deployed. For detailed information on deploying AWS SAM applications see the Deploy your application and resources topic in the *AWS Serverless Application Model* Developer Guide.

There are 2 actions that are accessible from the button icons located next to the AWS Lambda function in your project folder or by right-clicking the AWS Lambda function:

- **Local Invoke and Debug Configuration**: Opens the **Local Invoke and Debug Configuration** form in your VS Code editor. With this form you can create, edit, and run launch-configs of type:`aws-sam`. For additional information about SAM Debug configurations, see the Configuration options for debugging serverless applications topic in this User Guide.

  > **Note**
  >
  > At present, debugging a .NET Core application on an ARM64 architecture is not supported by VS Code. If you attempt to debug a .NET Core application the following error is displayed:
  > `The vsdbg debugger does not currently support the arm64 architecture. Function will run locally without debug.`
  > For additional details about this issue, see this VSCode-csharp issue in the DotNet GitHub repository.

- **Open Function Handler**: Opens your project file that contains the function handler.

There are 2 additional actions available for deployed AWS Lambda functions.

- **Remote invoke**: opens the **Remote invoke configuration** menu in the VS Code editor.
- **Search logs**: Opens the **Search logs** dialog in VS Code.

Walkthrough of Application Builder

The **Walkthrough of Application Builder** is a step-by-step interactive guide that takes you through the process of building a new application with AWS Application Builder. You can access

the **Walkthrough of Application Builder** from two places: the Application Builder explorer in the AWS Toolkit for Visual Studio Code and the VS Code **Welcome** tab. When you select **Walkthrough of Application Builder** from the Application Builder explorer in the AWS Toolkit, it opens the **Walkthrough of Application Builder** in the VS Code **Welcome** tab in the VS Code **Editor** window.

The **Walkthrough of Application Builder** is comprised of 5 main sections:

1. Installation

The Installation section checks to see if you have installed the AWS CLI tools required by Application Builder and other optional tools. If you don't have the required tools or your tools are out of date, you're guided through the process of installing the correct versions.

To see if you have the correct AWS CLI and optional tools installed, select the button for the AWS CLI or another tool that you want to test. After selecting a button, your **AWS Toolkit Logs** update and VS Code displays an alert message with the status of your tools. If you need to install or update your tools, the **Walkthrough of Application Builder** updates with the instructions and resources your need to proceed.

For detailed information on installing the AWS CLI, see the Install or update to the latest version of the AWS CLI topic in the *AWS CLI* Developer Guide. For detailed information on installing the AWS SAM CLI, see the Install AWS SAM CLI topic in the *AWS SAM CLI* Developer Guide.

2. Choose your application template

The Choose your application template section guides you through the process of building a new application from a template.

To choose a template and initialize your application, complete the following steps.

1. From the **Walkthrough of Application Builder**, select the **Choose your application template** section to display a list of template options on your screen.
2. Choose a template from the list, then choose the **Initialize your project** button to open a VS Code dialog.
3. Complete the steps in the VS Code dialog to initialize your new application.
4. The AWS Toolkit logs update with the status of your application during the initialization process.
5. To view your application in the Application Builder explorer, choose the **Refresh Application Builder Explorer** icon to update the explorer with your changes.

3. Iterate locally

The Iterate locally section contains example images that demonstrate how you can iterate with the Application Builder features available in the VS Code and AWS Toolkit explorers.

For additional information about all of the Application Builder features available in the VS Code and AWS Toolkit explorers, see the *Working with the Application Builder explorer* section, located in this User Guide topic.

4. Deploy to AWS

The Deploy to AWS section contains information on how to configure your credentials to connect with AWS for the purposes of deploying your application and examples of how to deploy your application with Application Builder.

To connect to AWS with your existing credentials from the **Walkthrough of Application Builder**, complete one of the following procedures.

Workforce: Sign in to AWS with single sign-on.

1. From the **Deploy to AWS** section in the **Walkthrough of Application Builder**, choose the **Configure credentials** button to open the **AWS: LOGIN** menu in the AWS Toolkit explorer.
2. From the **AWS: LOGIN** menu, choose **Workforce**, then choose the **Continue** button to proceed.
3. Enter your **Start URL** into the provided field, choose your **AWS Region** from the drop-down menu, then choose the **Continue** button to proceed.
4. From the VS Code pop-up window, confirm that you want to open the AWS Authentication site in your default browser.
5. From your default browser, complete the authentication steps, you're notified when authentication is complete and it's safe to close your browser window.

IAM Credentials: Store keys for use with AWS CLI tools.

1. From the **Deploy to AWS** section in the **Walkthrough of Application Builder**, choose the **Configure credentials** button to open the **AWS: LOGIN** menu in the AWS Toolkit explorer.
2. From the **AWS: LOGIN** menu, choose **IAM Credentials**, then choose the **Continue** button to proceed.
3. Enter a **Profile Name** into the provided field, then input your **`Access Key`** and **`Secret Key`**, then choose the **Continue** button to proceed.

4. VS Code displays the status of your authentication, notifying you if authentication is complete or your credentials are invalid.

For detailed information on configuring your credentials for deployment with the AWS CLI, see the Configure the AWS CLI topic in the *AWS CLI* Developer Guide. For additional information about connecting to AWS from the AWS Toolkit using your existing credentials, see the Connecting to AWS topic in this User Guide.

AWS Infrastructure Composer

You can use the AWS Toolkit for Visual Studio Code to work with the AWS Infrastructure Composer. AWS Infrastructure Composer is a visual builder for AWS applications that assists with designing your application architecture and visualizing your CloudFormation infrastructure.

For detailed information about AWS Infrastructure Composer, see the AWS Infrastructure Composer User Guide.

The following topics describe how to work with AWS Infrastructure Composer from the AWS Toolkit for Visual Studio Code.

Topics

- Working with AWS Infrastructure Composer in the Toolkit

Working with AWS Infrastructure Composer in the Toolkit

AWS Infrastructure Composer for the AWS Toolkit for Visual Studio Code allows you to visually design applications through an interactive canvas. You can also use Infrastructure Composer to visualize and modify CloudFormation and AWS Serverless Application Model (AWS SAM) templates. While working with Infrastructure Composer, your changes are stored persistently enabling you to switch seamlessly between editing files directly in the VS Code editor or using the interactive canvas.

For detailed information about AWS Infrastructure Composer, getting started information, and tutorials, see the AWS Infrastructure Composer User Guide.

The following sections describe how to access AWS Infrastructure Composer from the AWS Toolkit for Visual Studio Code.

Accessing AWS Infrastructure Composer from the Toolkit

There are 3 main ways that you can access AWS Infrastructure Composer from the Toolkit.

Accessing AWS Infrastructure Composer from an existing template

1. From VS Code, open an existing template file in the VS Code editor.
2. From the **editor window**, click the AWS Infrastructure Composer button located in the upper right-hand corner of the editor window.
3. AWS Infrastructure Composer opens and visualizes your template file in the VS Code editor window.

Accessing AWS Infrastructure Composer from the context menu (right-click)

1. From VS Code right-click the template file you want to open with AWS Infrastructure Composer.
2. In the context menu, choose the **Open with App Composer** option.
3. AWS Infrastructure Composer opens and visualizes your template file in a new VS Code editor window.

Accessing AWS Infrastructure Composer from the Command Palette

1. From VS Code open the Command Palette by pressing **`Cmd + Shift + P`** or **`Ctrl + Shift + P`** (Windows)
2. In the search field, enter **`AWS Infrastructure Composer`** and choose **AWS Infrastructure Composer** when it populates in the results.
3. Choose the template file you want to open, AWS Infrastructure Composer opens and visualizes your template file in a new VS Code editor window.

AWS CDK for VS Code

This is prerelease documentation for a feature in preview release. It is subject to change.

The **AWS CDK service** enables you to work with AWS Cloud Development Kit (AWS CDK) applications, or *apps*. You can find detailed information about the AWS CDK in the AWS Cloud Development Kit (AWS CDK) Developer Guide.

AWS CDK apps are composed of building blocks known as *constructs*, which include definitions for your CloudFormation stacks and the AWS resources within them. Using the **AWS CDK Explorer**, you can visualize the stacks and resources that are defined in AWS CDK constructs. This visualization is provided in a *tree view* in the Developer Tools pane within the Visual Studio Code (VS Code) editor.

This section provides information about how to access and use **AWS CDK** in the VS Code editor. It assumes that you've already installed and configured the Toolkit for VS Code for your local IDE.

Topics

Working with AWS CDK applications

This is prerelease documentation for a feature in preview release. It is subject to change.

Use the **AWS CDK Explorer** in the AWS Toolkit for VS Code to visualize and work with AWS CDK applications.

Prerequisites

- Be sure your system meets the the prerequisites specified in Installing the Toolkit for VS Code.
- Install the AWS CDK command line interface, as described in the first few sections of Getting Started with the AWS CDK in the *AWS Cloud Development Kit (AWS CDK) Developer Guide*.

> **Important**
>
> The AWS CDK version must be 1.17.0 or later. Use **`cdk --version`** on the command line to see what version you're running.

Visualize an AWS CDK application

Using the AWS Toolkit for VS Code AWS CDK Explorer, you can manage the stacks and resources that are stored in the CDK constructs of your apps. The AWS CDK Explorer displays your resources in a tree view using the information defined in the `tree.json` file, which is created when you run the **`cdk synth`** command. The `tree.json` file is located in an app's `cdk.out` directory, by default.

To get started using the Toolkit AWS CDK Explorer, you'll need to create a CDK application.

1. Complete the first several steps of the Hello World Tutorial located in the AWS CDK Developer Guide.

   > **Important**
   >
   > When you arrive at the tutorial step **Deploying the Stack**, stop and return to this guide.

   > **Note**
   >
   > You can run the commands provided in the tutorial, for example, **`mkdir`** and **`cdk init`**, on an operating system command line or in a **Terminal** window inside the VS Code editor.

2. After you complete the required steps of the CDK tutorial, open the CDK content that you created in the VS Code editor.
3. From the AWS navigation pane, expand the **CDK (Preview)** heading. Your CDK applications and their associated resources are now displayed in the CDK Explorer tree view.

Important notes

- When you load CDK apps into the VS Code editor, you can load multiple folders at one time. Each folder can contain multiple CDK apps, as shown in the preceding image. The AWS CDK Explorer finds apps in the project root directory and its direct subdirectories.

- When you perform the first several steps of the tutorial, you might notice that the last command you execute is **`cdk synth`**, which generates the `tree.json` file. If you change aspects of a CDK app, for example, add more resources, you need to execute that command again to see the changes reflected in the tree view.

Perform other operations on an AWS CDK app

You can use the VS Code editor to perform other operations on a CDK app, just as you would by using the command line of your operating system or other tools. For example, you can update the code files in the editor and deploy the app by using a VS Code **Terminal** window.

To try out these types of actions, use the VS Code editor to continue the Hello World Tutorial in the *AWS CDK Developer Guide*. Be sure to perform the last step, **Destroying the App's Resources**, so that you don't incur unexpected costs to your AWS account.

Working with AWS CloudFormation stacks

The AWS Toolkit for Visual Studio Code provides support for AWS CloudFormation stacks. Using the Toolkit for VS Code, you can perform certain tasks with AWS CloudFormation stacks, such as deleting them.

Topics

- Deleting an CloudFormation stack
- Create a AWS CloudFormation template using the AWS Toolkit for Visual Studio Code

Deleting an CloudFormation stack

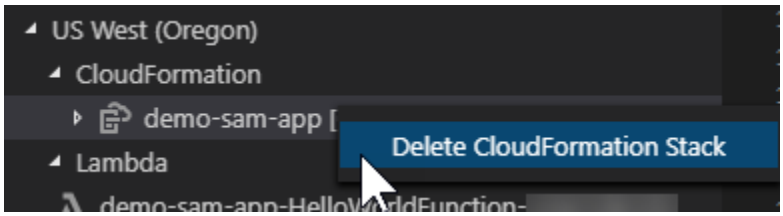
You can use the AWS Toolkit for Visual Studio Code to delete CloudFormation stacks.

Prerequisites

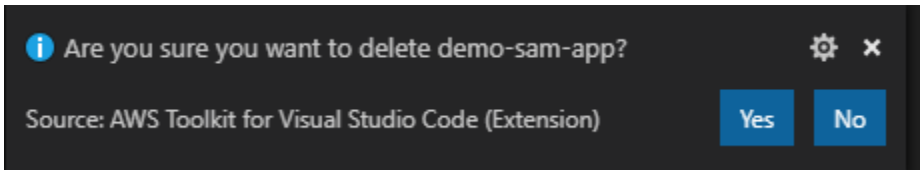
- Be sure your system meets the the prerequisites specified in Installing the Toolkit for VS Code.
- Ensure that the credentials you configured in Authentication and access include appropriate read/write access to the CloudFormation service. If in the **AWS Explorer**, under **CloudFormation**, you see a message similar to "Error loading CloudFormation resources", check the permissions attached to those credentials. Changes that you make to permissions will take a few minutes to affect the **AWS Explorer** in VS Code.

Delete a CloudFormation stack

1. In the **AWS Explorer**, open the context menu of the CloudFormation stack you want to delete.



2. Choose **Delete CloudFormation Stack**.
3. In the message that appears, choose **Yes** to confirm the delete.



After the stack is deleted, it's no longer listed in the **AWS Explorer**.

Create a AWS CloudFormation template using the AWS Toolkit for Visual Studio Code

The AWS Toolkit for Visual Studio Code can assist you in writing AWS CloudFormation and SAM templates.

Prerequisites

Toolkit for VS Code and credential prerequisites

- Before you can access the CloudFormation service from the Toolkit for VS Code, you need to meet the requirements outlined in the the userguide Installing the Toolkit for VS Code.
- The credentials you created in Authentication and access must include appropriate read/write access to the AWS CloudFormation service.

> **Note**
>
> If the **CloudFormation** service displays an **Error loading CloudFormation resources** message, check the permissions you've attached to those credentials. Also note that Changes made to permissions may take a few minutes to update in the **AWS Explorer**.

CloudFormation template prerequisites

- Install and enable the Redhat Developer YAML VS Code extension.
- You need to be connected to the internet when using the Redhat Developer YAML VS Code extension because it's used to download and cache JSON schemas on your machine.

Writing a CloudFormation template with YAML Schema Support

The toolkit uses YAML language support and JSON schemas to streamline the process of writing CloudFormation and SAM templates. Features like syntax validation and autocompletion not only make the process faster, but also help improve the quality of your template. When selecting a schema for your template, the following are recommended best practices.

CloudFormation template

- File has a .yaml or .yml extension.
- The file has a top-level `AWSTemplateFormatVersion` or **Resources** node.

SAM Template

- All of the criteria already described for CloudFormation
- The file has a top-level **Transform** node, containing a value that begins with `AWS::Serverless`.

The schema will be applied upon file modification. For example, a SAM Template schema will be applied after adding a serverless transform to a CloudFormation template and saving the file.

Syntax Validation

The YAML extension will automatically apply type validation to your template. This highlights entries with invalid types for a given property. If you hover over a highlighted entry, the extensions displays corrective actions.

Autocompletion

When adding new fields, enumerated values, or other resource types, you can initiate the YAML extension's autocompletion feature by typing **Ctrl + space**.

Working with CloudWatch Logs by using the AWS Toolkit for Visual Studio Code

Amazon CloudWatch Logs enables you to centralize the logs from all of your systems, applications, and AWS services that you use, in a single, highly scalable service. You can then easily view them, search them for specific error codes or patterns, filter them based on specific fields, or archive them securely for future analysis. For more information, see What Is Amazon CloudWatch Logs? in the *Amazon CloudWatch User Guide*.

The following topics describe how to use the AWS Toolkit for Visual Studio Code to work with CloudWatch Logs in an AWS account.

Topics

Viewing CloudWatch log groups and log streams by using the AWS Toolkit for Visual Studio Code

A *log stream* is a sequence of log events that share the same source. Each separate source of logs into CloudWatch Logs makes up a separate log stream.

A *log group* is a group of log streams that share the same retention, monitoring, and access control settings. You can define log groups and specify which streams to put into each group. There is no limit on the number of log streams that can belong to one log group.

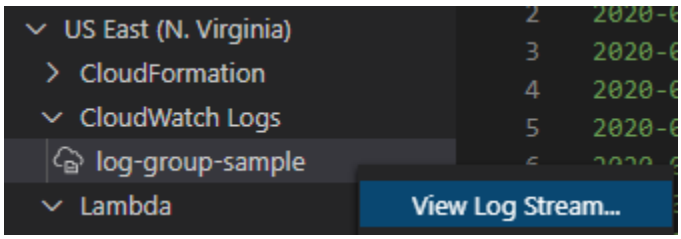
For more information, see Working with Log Groups and Log Streams in the *Amazon CloudWatch User Guide*.

Topics

Viewing log groups and log streams with the CloudWatch Logs node

1. In VS Code, choose **View**, **Explorer** to open AWS Explorer.
2. Click the **CloudWatch Logs** node to expand the list of log groups.

   The log groups for the current AWS Region are displayed under the **CloudWatch Logs** node.
3. To view the log streams in a log group, right-click the name of the log group, and then choose **View Log Stream**.



4. From the **Command Palette**, select a log stream from the group to view.

   > **Note**
   >
   > The **Command Palette** displays a timestamp for the last event in each stream.

   The **Log Stream** editor launches to display the stream's log events.

Working with CloudWatch log events in log streams by using the AWS Toolkit for Visual Studio Code

After you've opened the **Log Steam** editor, you can access the log events in each stream. Log events are records of activity recorded by the application or resource being monitored.

Topics

Viewing and copying log stream information

When you open a log stream, the **Log Stream** editor displays that stream's sequence of log events.

1. To find a log stream to view, open the **Log Stream** editor (see Viewing CloudWatch log groups and log streams).

   Each line listing an event is timestamped to show when it was logged.
2. You can view and copy information about the stream's events using the following options:

   - View events by time: Display the latest and older log events by choosing **Load newer events** or **Load older events**.

     > **Note**
     >
     > The **Log Stream** editor initially loads a batch of the most recent 10,000 lines of log events or 1 MB of log data (whichever is smaller). If you choose **Load newer events**, the editor displays events that were logged after the last batch was loaded. If you choose **Load older events**, the editor displays a batch of events that occurred before those currently displayed.

   - Copy log events: Select the events to copy, then right-click and select **Copy** from the menu.
   - Copy the log stream's name: Right-click the tab of **Log Stream** editor and choose **Copy Log Stream Name**.

> **Note**
>
> You can also use the **Command Palette** to run **AWS Toolkit Copy Log Stream Name**.

Save the contents of the log stream editor to a local file

You can download the contents of the CloudWatch log stream editor to a `log` file on your local machine.

> **Note**
>
> With this option, you save to file only the log events that are currently displayed in the log stream editor. For example, if the total size of a log stream is 5MB and only 2MB is loaded in the editor, your saved file will also contain only 2MB of log data. To display more data to be saved, choose **Load newer events** or **Load older events** in the editor.

1. To find a log stream to copy, open the **Log Streams** editor (see Viewing CloudWatch log groups and log streams).
2. Choose the **Save** icon beside the tab displaying the log stream's name.

   > **Note**
   >
   > You can also use the **Command Palette** to run **AWS Toolkit Save Current Log Stream Content**.

3. Use the dialog box to select or create a download folder for the log file, and click **Save**.

Searching CloudWatch log groups

You can use Search Log Group to search all log streams in a log group.

For detailed information about the Amazon CloudWatch Logs service, see the Working with Log Groups and Log Streams topic in the *Amazon CloudWatch User Guide*.

Searching log groups from the VS Code Command Palette

To search log groups from the VS Code Command Palette, complete the following steps.

For detailed information about Amazon CloudWatch Logs filters and patterns, see the Filter and pattern syntax section in the *Amazon CloudWatch User Guide*.

1. From VS Code, open the **Command Palette** by pressing **`cmd+shift+p`** (windows: **`ctrl+shift+p`**).
2. From the Command Palette, enter the command **`AWS: Search Log Group`**, then select it to open the search log group dialog in VS Code and follow the prompts to continue.

   > **Note**
   >
   > From the first prompt, you have the option to switch your AWS region before proceeding to the next steps.

3. From the **Select Log Group (1/3)** prompt, choose the log group you want to search.
4. From the **Select Time Filter (2/3)** prompt, choose the time filter to apply to your search.
5. From the **Search Log Group... (3/3)** prompt, enter your search pattern in the provided field, then press the **`Enter`** key to continue or the **ESC** key to cancel the search.
6. Your search results open in the VS Code editor, when the search is complete.

Searching log groups from the AWS Explorer

To search log groups from the AWS Toolkit for Visual Studio Code Explorer, complete the following steps.

1. From the AWS Toolkit for Visual Studio Code Explorer expand **CloudWatch**.
2. Open the context menu for (right-click) the Search Log Group you want to search, then choose **Search Log Group** to open the search prompt.
3. Follow the prompts by selecting a time frame to continue.
4. When prompted, enter your search pattern in the provided field, then press the **`Enter`** key to continue or the **ESC** key to cancel the search.
5. Your search results open in the VS Code editor when the search is complete.

Working with search log results

After completing a successful CloudWatch log group search, your search results open in the VS Code editor. The following procedures describe how to work with search log results.

> **Note**
>
> When viewing a single log stream, the following features are limited to the results in your currently-active log stream.

Saving your search log group results

To save your search log group results locally, complete the following steps.

1. From your search log group results, choose the **Save Log to File** icon button, located in the upper right-hand corner of the VS Code editor.
2. From the **Save As** prompt, specify the name and location you would like to save the file to.
3. Choosing **OK** saves the file to your local machine.

Changing the time range the time-range

To change the time range that is active in your search log group results, complete the following steps.

1. From your search log group results, choose the **Search by date...** icon button, located in the upper right-hand corner of the VS Code editor.
2. From the **Select Time Filter** prompt, choose a new time range for your search log results.
3. Your results are updated when the **Select Time Filter** prompt is closed.

Changing the search pattern

To change the search pattern that is active in your search log group results, complete the following steps.

1. From your search log group results, choose the **Search by Pattern...** icon button, located in the upper right-hand corner of VS Code editor.

2. From the **Search Log Group** prompt, enter the new search pattern in the provided field.
3. Press the **Enter** key to close the prompt and update your results with the new search pattern.

Amazon CloudWatch Logs Live Tail

Live stream your CloudWatch log events as they are ingested into a particular Log Group with Amazon CloudWatch Logs Live Tail.

For detailed information about the Live Tail feature, see the Troubleshoot with CloudWatch Logs Live Tail topic in the *Amazon CloudWatch Logs* User Guide.

Live Tail sessions incur costs by session usage time, per minute. For information about pricing, view the *Logs* tab in the **Paid Tier** section of the Amazon CloudWatch Pricing guide.

Starting a Live Tail session from the VS Code Command Palette

To start a Live Tail session from the VS Code Command Palette, complete the following steps.

For detailed information about Amazon CloudWatch Logs filters and patterns, see the Filter and pattern syntax section in the *Amazon CloudWatch* User Guide.

Starting a tailing session from the Command Palette

1. From VS Code, open the **Command Palette** by pressing **cmd+shift+p** (Windows: **ctrl +shift+p**).
2. From the **Command Palette**, enter the command `AWS: Tail Log Group`, then select it to open the **Tail log group** dialog in VS Code and follow the prompts to continue.

   > **Note**
   >
   > At the first prompt you have the option to switch your AWS Region before proceeding to the next steps.

3. From the **Tail Log Group (1/3)** prompt, choose the log group you want to tail.
4. From the **Include log events from... (2/3)** prompt, choose the log stream filter to apply to your tailing session.
5. From the **Provide log event filter pattern... (3/3)** prompt, enter your filter pattern in the provided field, then press the **Enter** key to continue or the **ESC** key to cancel the search.

6. Upon completion, your results stream into the VS Code editor

> **Note**
>
> If a Live Tail session running in the VS Code window matches the configuration of a newly submitted Tail Log Group command, a new session doesn't start. Instead, your existing session becomes the active text editor.

Starting a Live Tail session from the AWS Explorer

To start a Live Tail session from the AWS Toolkit Explorer, complete the following steps.

Starting a tailing session from the AWS Explorer

1. From the AWS Toolkit Explorer, expand **CloudWatch**.
2. Open the context menu for (right-click) the Log Group you want to tail, then choose **Tail Log Group** to open the tailing prompt.
3. Follow the prompts to continue.
4. Your results will stream into the VS Code editor.

Stopping a Live Tail session

There are 2 ways to stop a running Tailing session.

Stopping a tailing session

1. Click the **Stop tailing** CodeLens at the bottom of the tailing-session text document.
2. Close all editors containing the tailing-session text document.

Amazon DocumentDB

You can manage your Amazon DocumentDB clusters and instances directly in VS Code with the AWS Toolkit for Visual Studio Code. Amazon DocumentDB (with MongoDB compatibility) is a fast, reliable, and fully-managed database service that simplifies the set up, operation, and scaling of MongoDB-compatible databases in the cloud. For detailed information about the Amazon DocumentDB service, see the Amazon DocumentDB Developer Guide.

The following topics describe how to work with Amazon DocumentDB with the AWS Toolkit for Visual Studio Code.

Topics

Working with Amazon DocumentDB in the Toolkit

Amazon DocumentDB (with MongoDB compatibility) is a fast, reliable, and fully-managed database service that simplifies the set up, operation, and scaling of MongoDB-compatible databases in the cloud.

For detailed information about Amazon DocumentDB, getting started information, and tutorials, see the Amazon DocumentDB Developer Guide.

The following sections describe how to work with Amazon DocumentDB with the AWS Toolkit for Visual Studio Code.

Accessing Amazon DocumentDB from the AWS Toolkit

To access Amazon DocumentDB with the AWS Toolkit, complete the following procedure.

Accessing Amazon DocumentDB in the AWS Toolkit

1. From VS Code, open AWS Toolkit for Visual Studio Code.
2. From the AWS Toolkit, expand the **Explorer**.
3. From the **Explorer**, expand Amazon DocumentDB to display your existing Amazon DocumentDB resources.

Creating an instance-based cluster.

To get started working with Amazon DocumentDB, create a cluster by completing the following procedure.

Creating an instance-based cluster

1. From the AWS Toolkit for Visual Studio Code, open the context menu for (right-click) Amazon DocumentDB, then select **Create Cluster** to open the **Create Amazon DocumentDB Cluster** dialog in VS Code.

2. From the **Cluster type** screen, choose **Instance Based Cluster**.
3. From the **Cluster name** screen, specify a name for your new cluster.
4. From the **Select engine version** screen, choose your preferred Amazon DocumentDB engine version.
5. From the **Admin username and password** screens, specify an admin username and password to protect your cluster.
6. From the **Specify Storage encryption** screen, choose whether or not to encrypt your cluster.
7. From the **Number of instances** screen, configure your preferred number of instances.
8. From the **Select instance class** screen, choose your preferred instance class then proceed to create your new cluster.

> **Note**
>
> It could take several minutes to spin-up your cluster.

Copying a cluster endpoint

To copy your Amazon DocumentDB cluster endpoint, complete the following procedure.

Copying a cluster endpoint

1. From the AWS Toolkit for Visual Studio Code, expand **Amazon DocumentDB** to display your Amazon DocumentDB clusters.
2. Right-click the cluster you want to copy the connection details from, then choose **Copy Endpoint** to copy the cluster endpoint information to your clipboard.
3. Your cluster endpoint can now be pasted into your documents.

Open in browser

Open your Amazon DocumentDB clusters in the AWS Console for a more cluster management features. To open the AWS Console to your Amazon DocumentDB cluster in your default web browser, complete the following procedure.

Opening your cluster in the AWS Console

1. From the AWS Toolkit for Visual Studio Code, expand **Amazon DocumentDB** to display your Amazon DocumentDB clusters.
2. Right-click the cluster you want to view in the AWS Console, then choose **Open in Browser**.
3. The AWS Console opens to the Amazon DocumentDB cluster in your default web browser.

Expanding an existing cluster

To scale your Amazon DocumentDB clusters by adding instances, complete the following procedure.

Adding an instance to expand your cluster

1. From the AWS Toolkit for Visual Studio Code, expand **Amazon DocumentDB** to display your Amazon DocumentDB clusters.
2. Right-click the cluster you want to expand and choose **Add an Instance** to open the **Add an Instance** dialog in VS Code.
3. When prompted, input a name for your new instance into the text field, then press the **`Enter`** key to continue.
4. When prompted, select an instance class from the list to continue.
5. The **AWS Explorer** displays the creation status and updates when the new instance is ready.

Stopping a cluster

Stop your Amazon DocumentDB cluster by completing the following procedure.

> **Note**
>
> While your cluster is stopped, most cluster management features will be unavailable.

Stopping your Amazon DocumentDB cluster

1. From the AWS Toolkit for Visual Studio Code, expand **Amazon DocumentDB** to display your Amazon DocumentDB clusters.

2. Choose the **Stop Cluster** button located next to the cluster your want to stop or right-click the cluster and choose **Stop Cluster**.
3. When prompted, choose **Yes** to stop your cluster or **Cancel** to cancel the stop process and leave your cluster running.
4. The **AWS Explorer** displays the status of your cluster and updates when the cluster has stopped.

Rebooting an instance

Rebooting an instance is useful for troubleshooting and making minor changes without impacting your entire cluster. To reboot an Amazon DocumentDB instance, complete the following procedure.

Rebooting a cluster instance

1. From the AWS Toolkit for Visual Studio Code, expand **Amazon DocumentDB** to display your Amazon DocumentDB clusters.
2. Right-click the cluster instance that you want to reboot, then choose **Reboot Instance**.
3. When prompted, choose **Yes** to reboot your instance or **Cancel** to cancel the reboot process and leave your instance stopped.
4. The **AWS Explorer** displays the status of your cluster and updates when your instance has rebooted.

Deleting an instance

To delete an Amazon DocumentDB cluster instance, complete the following procedure.

> **Note**
>
> Deleting an instance doesn't impact the data in your cluster. If you delete the primary instance, one of the replica instances takes over as the writable instance.

Deleting a cluster instance

1. From the AWS Toolkit for Visual Studio Code, expand **Amazon DocumentDB** to display your Amazon DocumentDB clusters.

2. Right-click the cluster instance you want to delete, then choose **Delete** to open the delete-cluster-instance confirmation dialog in VS Code.
3. Follow the confirmation prompt, then press the **`Enter`** key to delete your cluster instance.
4. The **AWS Explorer** displays the status of your cluster instance and updates when your instance has been deleted.

Viewing, adding, and removing tags

Tags are used to organize and track resources within your environment. To view or edit the tags associated with your Amazon DocumentDB cluster, complete one of the following procedures.

Viewing cluster tags

1. From the AWS Toolkit for Visual Studio Code, expand **Amazon DocumentDB** to display your Amazon DocumentDB clusters.
2. Right-click the cluster you want to view tags for, then choose **Tags...** to open the **Tags for `your cluster name`** dialog.
3. Your tags are displayed in the dialog window, if no tags are associated with your cluster then the message **[No tags assigned]** is displayed.

Adding tags to your cluster

1. From the AWS Toolkit for Visual Studio Code, expand **Amazon DocumentDB** to display your Amazon DocumentDB clusters.
2. Right-click the cluster you want to add tags for, then choose **Tags...** to open the **Tags for `your cluster name`** dialog.
3. Choose the **Add tag...** button to open the **Add Tag** dialog in VS Code.
4. Enter a new tag into the text field, then press the **Enter** key to continue.
5. Enter a value into the text field, then press the **Enter** to add the key/value pair to your cluster.

Removing tags from your cluster

1. From the AWS Toolkit for Visual Studio Code, expand **Amazon DocumentDB** to display your Amazon DocumentDB clusters.

2. Right-click the cluster you want to remove tags from, then choose **Tags...** to open the **Tags for `your cluster name`** dialog.
3. Choose the **Remove tag...** button to open the **Remove a tag from `your cluster name`** dialog in VS Code.
4. Choose the tag your want to remove from the provided list to remove the tag from your cluster.

Modifying an instance class

To modify the class of an Amazon DocumentDB cluster instance, complete the following procedure.

Modifying an instance class

1. From the AWS Toolkit for Visual Studio Code, expand **Amazon DocumentDB** to display your Amazon DocumentDB clusters.
2. Right-click the cluster instance you want to modify, then choose **Modify Class...** to open the **Select instance class** dialog in VS Code.
3. Choose a new class for your instance from the list to update the class.
4. The **AWS Explorer** displays the status of your cluster instance and updates when the class of your instance has been updated.

Amazon Elastic Compute Cloud

Amazon Elastic Compute Cloud for the AWS Toolkit for Visual Studio Code allows you to launch and connect to your Amazon EC2 instances from VS Code. For detailed information about Amazon EC2, see the What is Amazon EC2? topic in the *Amazon Elastic Compute Cloud* User Guide.

The following topics describe how to work with AWS Application Builder from the AWS Toolkit for Visual Studio Code.

Topics

- Working with Amazon Elastic Compute Cloud
- Troubleshooting Amazon Elastic Compute Cloud

Working with Amazon Elastic Compute Cloud

The following sections describe how to work with Amazon Elastic Compute Cloud in the AWS Toolkit for Visual Studio Code.

Prerequisites

The features described in this user guide topic have been tested on Amazon EC2 instances with the following operating systems:

- Windows 2016+

> **Note**
>
> This OS only works when connecting a VS Code terminal. It doesn't work when connecting a full VS Code remote instance. For additional information about VS Code terminals and remote instances, see the Getting started with the terminal and VS Code Remote Development topics in the VS Code documentation.

- Amazon Linux 2023
- Ubuntu, 22.04

A locally installed **SSH** is required to open a remote connection to an Amazon EC2 instance, but is not required to open a terminal to an Amazon EC2 instance.

Your Amazon EC2 instance profile must include the following AWS Identity and Access Management (IAM) permissions.

```
"ssmmessages:CreateControlChannel",
"ssmmessages:CreateDataChannel",
"ssmmessages:OpenControlChannel",
"ssmmessages:OpenDataChannel",
"ssm:DescribeAssociation",
"ssm:ListAssociations",
"ssm:UpdateInstanceInformation
```

Note

The required permissions are included in the following AWS managed policy.

- `AmazonSSMManagedInstanceCore`
- `AmazonSSMManagedEC2InstanceDefaultPolicy`

Viewing existing Amazon EC2 instances

To view your existing Amazon EC2 instances from the AWS Toolkit, complete the following steps.

1. From the AWS Toolkit, expand the AWS Toolkit Explorer.
2. Expand the region that contains the Amazon EC2 instances that you want to view.
3. Expand the **EC2** heading to display your existing Amazon EC2 instances.

Launching a new Amazon EC2 instance

There are 3 ways to create a new Amazon EC2 instance with the AWS Toolkit.

Each work flow opens the **Launch an instance** wizard in the AWS console. For detailed information about launching a new Amazon EC2 instance from the **Launch an instance** wizard, see the Launch an EC2 instance using the launch instance wizard in the console topic in the *Amazon Elastic Compute Cloud* User Guide. To launch a new Amazon EC2 instance, complete one of the following procedures.

Launching a new Amazon EC2 instance from the VS Code Command Palette

1. From VS Code, open the VS Code Command Palette by pressing **`command + shift + P (Windows: ctrl + shift + P)`**
2. From the VS Code Command Palette, search for the **`AWS: Launch EC2`** command and select it when it populates in the list to open the Launch EC2 instance **Select Region** prompt in VS Code.
3. From the Launch EC2 instance **Select Region** prompt, choose the region you want to launch your new instance in, then confirm you want to open the AWS Console in your default web browser.
4. From the AWS Console in your default web browser, complete the authentication process to proceed to the **Launch an instance** wizard.

5. From the **Launch an instance** wizard, complete the required sections, then choose the **Launch instance** button to launch your new Amazon EC2 instance.
6. The AWS Explorer updates to show your new Amazon EC2 instance.

Launching a new Amazon EC2 instance from the AWS Explorer

1. Expand the AWS Toolkit Explorer, then expand the region you want to create the new Amazon EC2 instance in.
2. Expand or hover over the **EC2** heading, then choose the **+ (Launch EC2 instance)** icon.
3. When prompted, confirm that you want to open the AWS Console in your default web browser.
4. From the AWS Console in your web browser, complete the authentication process to proceed to the **Launch an instance** wizard.
5. From the **Launch an instance** wizard, complete the required sections, then choose the **Launch instance** button to launch your new Amazon EC2 instance.
6. The AWS Explorer updates to show your new Amazon EC2 instance.

Launching a new Amazon EC2 instance from the context (right-click) menu

1. Expand the AWS Toolkit Explorer, then expand the region you want to create the new Amazon EC2 instance in.
2. Right-click the **EC2** heading, then choose **Launch EC2 instance**.
3. When prompted, confirm that you want to open the AWS Console in your default web browser.
4. From the AWS Console in your web browser, complete the authentication process to proceed to the **Launch an instance** wizard.
5. From the **Launch an instance** wizard, complete the required sections, then choose the **Launch instance** button to launch your new Amazon EC2 instance.
6. The AWS Explorer updates to show your new Amazon EC2 instance.

Connecting VS Code to an Amazon EC2 instance

There are 3 ways to connect to an Amazon EC2 instance from VS Code. To connect VS Code to your EC2 instance, complete one of the following procedures.

Connecting VS Code to an Amazon EC2 instance from the Command Palette

1. From VS Code, open the VS Code Command Palette by pressing **`command + shift + P (Windows: ctrl + shift + P)`**
2. From the VS Code Command Palette, search for the **`AWS: Connect VS Code to EC2 instance...`** command and select it when it populates in the list to open the **Select EC2 Instance** prompt in VS Code.
3. From the **Select EC2 Instance** prompt, choose the region that contains the instance you want to connect to, then choose the instance you want to connect to.
4. VS Code displays the status while the connection is being established.
5. A new window opens to display your Amazon EC2 instance when the connection is complete.

Connecting VS Code to an Amazon EC2 instance from the AWS Explorer.

1. Expand the AWS Toolkit Explorer, then expand the region that contains the Amazon EC2 instance you want to connect to.
2. Hover over the Amazon EC2 instance, then choose the **(Connect VS Code to EC2 instance)** icon.

> **Note**
>
> You can also choose the **(Connect VS Code to EC2 instance)** icon from the **EC2** service heading in the AWS Explorer.

3. VS Code displays the status while the connection is being established.
4. A new window opens to display your Amazon EC2 instance when the connection is complete.

Connecting VS Code to an Amazon EC2 instance from the right-click menu

1. Expand the AWS Toolkit Explorer, then expand the region that contains the Amazon EC2 instance you want to connect to.
2. Right-click the Amazon EC2 instance you want to connect to, then choose **Connect VS Code to EC2 instance**.

> **Note**
>
> You can also right-click the **EC2** service heading in the AWS Explorer and choose the **Connect VS Code to EC2 instance**.

3. VS Code displays the status while the connection is being established.
4. A new window opens to display your Amazon EC2 instance when the connection is complete.

Opening a terminal to an Amazon EC2 instance.

There are 3 ways to connect to an Amazon EC2 instance from the VS Code terminal.

Connecting VS Code to an Amazon EC2 instance from the Command Palette

1. From VS Code, open the VS Code Command Palette by pressing **`command + shift + P (Windows: ctrl + shift + P)`**
2. From the VS Code Command Palette, search for the **`AWS:Open terminal to EC2 instance...`** command and select it when it populates in the list to open the **Select EC2 Instance** prompt in VS Code.
3. From the **Select EC2 Instance** prompt, choose the region containing the instance you want to open in the terminal, then choose the instance.
4. VS Code displays the status while the connection is being established.
5. The VS Code Terminal opens to display your new session when the connection is complete.

Opening an Amazon EC2 instance in the VS Code terminal from the AWS Explorer.

1. Expand the AWS Toolkit Explorer, then expand the region that contains the Amazon EC2 instance you want to connect to.
2. Hover over the Amazon EC2 instance, then choose the **(Open terminal to EC2 instance...)** icon.

> **Note**
>
> You can also choose the **(Open terminal to EC2 instance...)** icon from the **EC2** service heading in the AWS Explorer.

3. VS Code displays the status while the connection is being established.

4. The VS Code Terminal opens to display your new session when the connection is complete.

Opening an Amazon EC2 instance in the VS Code terminal from the right-click menu

1. Expand the AWS Toolkit Explorer, then expand the region that contains the Amazon EC2 instance you want to open in the VS Code terminal.
2. Right-click the Amazon EC2 instance you want to open in the terminal, then choose **Open terminal to EC2 instance...**.

> **Note**
>
> You can also right-click the **EC2** service heading in the AWS Explorer and choose the **Open terminal to EC2 instance...**.

3. VS Code displays the status while the connection is being established.
4. The VS Code Terminal opens to display your new session when the connection is complete.

Starting or rebooting an Amazon EC2 instance

There are 3 ways to start or reboot an Amazon EC2 instance.

Rebooting an Amazon EC2 instance from the Command Palette

1. From VS Code, open the VS Code Command Palette by pressing **`command + shift + P (Windows: ctrl + shift + P)`**
2. From the VS Code Command Palette, search for the **`AWS: Reboot EC2 instance`** command and select it when it populates in the list to open the **Select EC2 Instance** prompt in VS Code.

> **Note**
>
> To start an instance that isn't running, you must choose the **`AWS: Start EC2 instance`** command. The **`AWS: Reboot EC2 instance`** command only reboots instances that are currently running.

3. From the **Select EC2 Instance** prompt, choose the region that contains the instance you want to start or reboot.
4. VS Code displays the status while the instance is rebooting.

5. The AWS Explorer updates to show that your instance is running when it has finished rebooting.

Starting or rebooting an Amazon EC2 instance from the AWS Explorer

1. Expand the AWS Toolkit Explorer, then expand the region that contains the Amazon EC2 instance you want to start or reboot.
2. Hover over the Amazon EC2 instance, then choose the **(Reboot EC2 instance)** icon.

> **Note**
>
> If the instance is stopped, then the only options is the **(Start EC2 instance)** icon

3. VS Code displays the status while the instance is rebooting.
4. The AWS Explorer updates to show that your instance is running when it has finished rebooting.

Starting or rebooting an Amazon EC2 instance from the right-click menu

1. Expand the AWS Toolkit Explorer, then expand the region that contains the Amazon EC2 instance you want to start or reboot.
2. Right-click the Amazon EC2 instance you want to connect to, then choose **Reboot EC2 instance**.

> **Note**
>
> If the instance is stopped, then the only options is the **Start EC2 instance**.

3. VS Code displays the status while the instance is rebooting.
4. The AWS Explorer updates to show that your instance is running when it has finished rebooting.

Stopping an Amazon EC2 instance

There are 3 ways to stop an Amazon EC2 instance.

Stopping an Amazon EC2 instance from the Command Palette

1. From VS Code, open the VS Code Command Palette by pressing **`command + shift + P (Windows: ctrl + shift + P)`**
2. From the VS Code Command Palette, search for the **`AWS: Stop EC2 instance`** command and select it when it populates in the list to open the **Select EC2 Instance** prompt in VS Code.
3. From the **Select EC2 Instance** prompt, choose the region that contains the instance you want to stop.
4. VS Code displays the status while the instance is stopping.
5. The AWS Explorer updates to show that your instance is stopped.

Stopping an Amazon EC2 instance from the AWS Explorer

1. Expand the AWS Toolkit Explorer, then expand the region that contains the Amazon EC2 instance you want to stop.
2. Hover over the Amazon EC2 instance, then choose the **(Stop EC2 instance)** icon.
3. VS Code displays the status while the instance is stopping.
4. The AWS Explorer updates to show that your instance has stopped.

Stopping an Amazon EC2 instance from the right-click menu

1. Expand the AWS Toolkit Explorer, then expand the region that contains the Amazon EC2 instance you want to stop.
2. Right-click the Amazon EC2 instance you want to connect to, then choose **Reboot EC2 instance**.
3. VS Code displays the status while the instance is stopping.
4. The AWS Explorer updates to show that your instance has stopped.

Copy Instance ID

To copy an instance ID, complete the following steps.

1. Right-click the instance your want to copy the ID from.
2. Choose **Copy Instance ID**.

3. The instance ID is copied to your local clipboard.

Copy Name

To copy an instance name, complete the following steps.

1. Right-click the instance your want to copy the name from.
2. Choose **Copy Instance Name**.
3. The instance name is copied to your local clipboard.

Copy ARN

To copy an instance ARN, complete the following steps.

1. Right-click the instance your want to copy the ARN from.
2. Choose **Copy Instance ARN**.
3. The instance ARN is copied to your local clipboard.

Troubleshooting Amazon Elastic Compute Cloud

The following sections describe how to troubleshoot known issues that can occur when working with Amazon Elastic Compute Cloud in the AWS Toolkit for Visual Studio Code. For detailed information about troubleshooting issues specific to the Amazon EC2 service, see the Troubleshoot issues with Amazon EC2 instances topic in the *Amazon Elastic Compute Cloud* User Guide.

General Debugging

If you encounter a remote connection issue for any reason, start by checking to see if an AWS Systems Manager connection can be established from the AWS Console.

To connect to an Amazon EC2 instance through Systems Manager from the AWS Console, complete the following steps.

1. From your web browser, navigate to the AWS Console.
2. Complete authentication to proceed to the AWS Console EC2 landing.
3. From the Amazon EC2 navigation pane, choose **Instances**.

4. Select the box located next to the instance that you want to connect to.
5. Choose the **Connect** button to open the **Connect to instance** screen in a new browser tab.

> **Note**
>
> You can only connect to an instance if it's running. If you're not able to select the **Connect** button, check to make sure that your instance is running.

6. From the **Connect to instance** screen, choose the **Session Manager** tab, then choose the **Connect** button to open the Systems Manager connection in your current browser tab.

> **Note**
>
> If you recently started your instance and the option isn't available for you to connect the Systems Manager, you may need to wait a few additional minutes before the option becomes available.

Target instance is not running

To connect to an Amazon EC2 instance from the terminal or a remote connection, the instance must be running. Before you attempt to connect to your instance from the AWS Toolkit, start it from the AWS Explorer, AWS Management Console, or AWS Command Line Interface.

Target instance doesn't have an IAM role or has an IAM role with improper permissions

To connect to your Amazon EC2 instance, it must have an IAM role with the correct permissions attached. If you attempt to connect to an instance that doesn't have an IAM role attached, you're notified by VS Code.

If you attempt to connect to an instance that has an IAM role but lacks necessary permissions, you're prompted to add the minimum necessary actions as an inline policy to the existing IAM role. After updating the inline policy, you're connected to your instance. For detailed information about IAM roles, permissions, and attaching a role to an instance, see the IAM roles for Amazon EC2 topic in the *Amazon Elastic Compute Cloud* User Guide and the Step 2: Verify or add instance permissions for Session Manager topic in the *AWS Systems Manager* User Guide.

The following example contains the minimum-necessary actions.

```
"ssmmessages:CreateControlChannel",
"ssmmessages:CreateDataChannel",
"ssmmessages:OpenControlChannel",
"ssmmessages:OpenDataChannel",
"ssm:DescribeAssociation",
"ssm:ListAssociations",
"ssm:UpdateInstanceInformation
```

Note

The required permissions are included in the following AWS managed policy.

- `AmazonSSMManagedEC2InstanceDefaultPolicy`
- `AmazonSSMManagedInstanceCore`

Target instance doesn't have a Systems Manager agent running

You may encounter this issue for a number of different reasons. To fix the issue, start by rebooting the instance and making another connection attempt. Or, manually start an initial connection through a non-ssm connection method. For more detailed information about Systems Manager, see the Working with Systems Manager Agent topic in the *AWS Systems Manager*.

On start-up, Amazon EC2 status indicates it's running, but connections aren't going through

If you recently started or created a new IAM role for an instance and are unable to establish a connection, wait a few additional minutes before making another attempt to establish a connection.

Working with Amazon Elastic Container Registry

Amazon Elastic Container Registry (Amazon ECR) is an AWS managed container-registry service that's secure, and scalable. Several Amazon ECR service functions are accessible from the Toolkit for VS Code Explorer.

- Creating a repository.

- Creating an AWS App Runner service for your repository or tagged image.
- Accessing image tag and repository URIs or ARNs.
- Deleting image tags and repositories.

You can also access the full-range of Amazon ECR functions through the VS Code console by integrating the AWS CLI and other platforms, with VS Code.

For more information about Amazon ECR, see What is Amazon ECR? in the Amazon Elastic Container Registry User Guide.

Topics

Working with Amazon Elastic Container Registry

You can access the Amazon Elastic Container Registry (Amazon ECR) service directly from the AWS Explorer in VS Code and use it to push a program image to an Amazon ECR repository. To get started, you need to do these steps:

1. Create a Dockerfile that contains the information necessary to build an image.
2. Build an image from that Dockerfile and tag the image for processing.
3. Create a repository inside your Amazon ECR instance.
4. Push the tagged image to your repository.

Prerequisites

You need to complete these steps in order to access the Amazon ECR service from the VS Code Explorer.

Create an IAM user

Before you can access an AWS service, such as Amazon ECR, you must provide credentials. This is so that the service can determine if you have permission to access its resources. We don't recommend that you access AWS directly through the credentials for your root AWS account. Instead, use AWS Identity and Access Management (IAM) to create an IAM user and then add that user to an

IAM group with administrative permissions. You can then access AWS using a special URL and the credentials for the IAM user.

If you signed up for AWS but didn't create an IAM user for yourself, you can create one by using the IAM console.

To create an administrator user, choose one of the following options.

Choose one way to manage your administrator	To	By	You can also
In IAM Identity Center (Recommended)	Use short-term credentials to access AWS. This aligns with the security best practices. For information about best practices, see Security best practices in IAM in the *IAM User Guide*.	Following the instructions in Getting started in the *AWS IAM Identity Center User Guide*.	Configure programmatic access by Configuring the AWS CLI to use AWS IAM Identity Center in the *AWS Command Line Interface User Guide*.
In IAM (Not recommended)	Use long-term credentials to access AWS.	Following the instructions in Create an IAM user for emergency access in the *IAM User Guide*.	Configure programmatic access by Manage access keys for IAM users in the *IAM User Guide*.

To sign in as this new IAM user, sign out of the AWS console, then use the following URL.
In the following URL, where *your_aws_account_id* is your AWS account number without the hyphens (for example, if your AWS account number is `1234-5678-9012`, your AWS account ID is `123456789012`):

```
https://your_aws_account_id.signin.aws.amazon.com/console/
```

Enter the IAM user name and password that you just created. When you're signed in, the navigation bar displays "*your_user_name @ your_aws_account_id*".

If you don't want the URL for your sign-in page to contain your AWS account ID, you can create an account alias. From the IAM dashboard, choose **Customize** and enter an **Account Alias**. This can be your company name. For more information, see Your AWS account ID and its alias in the IAM User Guide.

To sign in after you create an account alias, use the following URL:

```
https://your_account_alias.signin.aws.amazon.com/console/
```

To verify the sign-in link for IAM users for your account, open the IAM console and check under **IAM users sign-in link** on the dashboard.

For more information about IAM, see the AWS Identity and Access Management User Guide.

Install and configure Docker

You can install and configure Docker by selecting your preferred operating system from the Install Docker Engine user guide and following the instructions.

Install and configure AWS CLI version 2

Install and configure AWS CLI version 2 by selecting your preferred operating system from the Installing, updating, and uninstalling the AWS CLI version 2 user guide.

1. Creating a Dockerfile

Docker uses a file called a Dockerfile to define an image that can be pushed and stored on a remote repository. Before you can upload an image to an ECR repository, you must create a Dockerfile and then build an image from that Dockerfile.

Creating a Dockerfile

1. Use the Toolkit for VS Code explorer to navigate to the directory where you want to store your Dockerfile.

2. Create a new file that's called **Dockerfile**.

> **Note**
>
> VS Code could prompt you to select a file type or file extension. If this occurs, select **plaintext**. Vs Code has a "dockerfile" extension. However, we don't recommend you use it. This is because the extension might cause conflicts with certain versions of Docker or other associated applications.

Edit your Dockerfile using VS Code

If your Dockerfile has a file extension, open the context (right-click) menu for the file and remove the file extension.

After the file extension is removed from your Dockerfile:

1. Open the empty Dockerfile directly in VS Code.
2. Copy the contents of the following example into your Dockerfile:

Example Dockerfile image template

```
FROM ubuntu:18.04

# Install dependencies
RUN apt-get update && \
 apt-get -y install apache2

# Install apache and write hello world message
RUN echo 'Hello World!' > /var/www/html/index.html

# Configure apache
RUN echo '. /etc/apache2/envvars' > /root/run_apache.sh && \
 echo 'mkdir -p /var/run/apache2' >> /root/run_apache.sh && \
 echo 'mkdir -p /var/lock/apache2' >> /root/run_apache.sh && \
 echo '/usr/sbin/apache2 -D FOREGROUND' >> /root/run_apache.sh && \
 chmod 755 /root/run_apache.sh

EXPOSE 80

CMD /root/run_apache.sh
```

```
```

This is a Dockerfile that uses an Ubuntu 18.04 image. The **RUN** instructions update the package caches. Install software packages for the web server, and then write the "Hello World!" content to the document root of the web server. The **EXPOSE** instruction exposes port 80 on the container, and the **CMD** instruction starts the web server.

3. Save your Dockerfile.

> **Important**
>
> Make sure that your Dockerfile doesn't have an extension attached to the name. A Dockerfile with extensions might cause conflicts with certain versions of Docker or other associated applications.

2 . Build your image from your Dockerfile

The Dockerfile that you created contains the information necessary to build an image for a program. Before you can push that image to your Amazon ECR instance, you must first build the image.

Build an image from your Dockerfile

1. Use the Docker CLI or a CLI that's integrated with your instance of Docker to navigate into the directory that contains your Dockerfile.
2. Run the **Docker build** command to build the image that's defined in your Dockerfile.

```
        docker build -t hello-world .
```

3. Run the **Docker images** command to verify that the image was created correctly.

```
docker images --filter reference=hello-world
```

Example output:

```
```

```
REPOSITORY          TAG                 IMAGE ID            CREATED
 SIZE
hello-world         latest              e9ffedc8c286        4 minutes ago
 241MB
```

4. **Note**

   This step isn't necessary to create or push your image, but you can see how the program image works when it's run.

   To run the newly built image use the **Docker run** command.

   ```
   docker run -t -i -p 80:80 hello-world
   ```

   The **-p** option that's specified in the preceding example maps the exposed **port 80** on the container to **port 80** of the host system. If you're running Docker locally, navigate to http://localhost:80 using your web browser. If the program ran correctly, a "Hello World!" statement is displayed.

   For more information about the **Docker run** command, see Docker run reference on the Docker website.

3. Create a new repository

To upload your image into your Amazon ECR instance, create a new repository where it can be stored in.

Create a new Amazon ECR repository

1. From the VS Code **Activity Bar**, choose the **AWS Toolkit icon**.
2. Expand the **AWS Explorer** menu.
3. Locate the default AWS Region that's associated with your AWS account. Then, select it to see a list of the services that are through the Toolkit for VS Code.
4. Choose the **ECR +** option to begin the **Create new repository** process.
5. Follow the prompts to complete the process.

6. After it's complete, you can access your new repository from the **ECR** section of the AWS Explorer menu.

4. Push, pull, and delete images

After you built an image from your Dockerfile and created a repository, you can push your image into your Amazon ECR repository. Additionally, using the AWS Explorer with Docker and the AWS CLI, you can do the following:

- Pull an image from your repository.
- Delete an image that's stored in your repository.
- Delete your repository.

Authenticate Docker with your default registry

Authentication is required to exchange data between Amazon ECR and Docker instances. To authenticate Docker with your registry:

1. Open a command line operating system that's connected to your instance of AWS CLI.
2. Use the **get-login-password** method to authenticate to your private ECR registry.

```
aws ecr get-login-password --region region | docker login --username AWS --
password-stdin AWS_account_id.dkr.ecr.region.amazonaws.com
```

> **Important**
>
> In the preceding command, you must update both the **`region`** and the **`AWS_account_id`** to the information that's specific to your AWS account.

Tag and push an image to your repository

After you authenticated Docker with your instance of AWS, push an image to your repository.

1. Use the **Docker images** command to view the images that you stored locally and identify the one you would like to tag.

```
docker images
```

Example output:

```
REPOSITORY          TAG                 IMAGE ID            CREATED
 SIZE
hello-world         latest              e9ffedc8c286        4 minutes ago
 241MB
```

2. Tag your image with the **Docker tag** command.

```
docker tag hello-world:latest AWS_account_id.dkr.ecr.region.amazonaws.com/hello-
world:latest
```

3. Push the tagged image to your repository with the **Docker tag** command.

```
docker push AWS_account_id.dkr.ecr.region.amazonaws.com/hello-world:latest
```

Example output:

```
The push refers to a repository [AWS_account_id.dkr.ecr.region.amazonaws.com/hello-
world] (len: 1)
e9ae3c220b23: Pushed
a6785352b25c: Pushed
0998bf8fb9e9: Pushed
0a85502c06c9: Pushed
latest: digest:
 sha256:215d7e4121b30157d8839e81c4e0912606fca105775bb0636b95aed25f52c89b size: 6774
```

After your tagged image has been successfully uploaded to your repository, it's visible in the AWS Explorer menu.

Pull an image from Amazon ECR

- You can pull an image to your local instance of **Docker tag** command.

```
docker pull AWS_account_id.dkr.ecr.region.amazonaws.com/hello-world:latest
```

Example output:

```
The push refers to a repository [AWS_account_id.dkr.ecr.region.amazonaws.com/hello-
world] (len: 1)
e9ae3c220b23: Pushed
a6785352b25c: Pushed
0998bf8fb9e9: Pushed
0a85502c06c9: Pushed
latest: digest:
 sha256:215d7e4121b30157d8839e81c4e0912606fca105775bb0636b95aed25f52c89b size: 6774
```

Delete an image from your Amazon ECR repository

There are two methods for deleting an image from VS Code. The first method is to use the AWS Explorer.

1. From the AWS Explorer, expand the **ECR**menu
2. Expand the repository that you want to delete an image from
3. Choose the image tag associated with the image that you wish to delete, by opening the context menu (right-click)
4. Choose the **Delete Tag...** option to delete all stored images associated with that tag

Delete an image using the AWS CLI

- You can also delete an image from your repository with the **AWS ecr batch-delete-image** command.

```
AWS ecr batch-delete-image \
      --repository-name hello-world \
      --image-ids imageTag=latest
```

Example output:

```
{
    "failures": [],
    "imageIds": [
        {
            "imageTag": "latest",
            "imageDigest":
 "sha256:215d7e4121b30157d8839e81c4e0912606fca105775bb0636b95aed25f52c89b"
        }
    ]
}
```

Delete a repository from your Amazon ECR instance

There are two methods for deleting a repository from VS Code. The first method is to use the AWS Explorer.

1. From the AWS Explorer, expand the **ECR** menu
2. Choose the repository that you want to delete by opening the context (right-click) menu
3. Choose the **Delete Repository...** option to the chosen repository

Delete an Amazon ECR repository from the AWS CLI

- You can delete a repository with the **AWS ecr delete-repository** command.

> **Note**
>
> By default, you can't delete a repository that contains images. However, the **--force** flag allows this.

```
    AWS ecr delete-repository \
  --repository-name hello-world \
  --force
```

Example output:

```
{
    "failures": [],
    "imageIds": [
        {
            "imageTag": "latest",
            "imageDigest":
 "sha256:215d7e4121b30157d8839e81c4e0912606fca105775bb0636b95aed25f52c89b"
        }
    ]
}
```

Creating an App Runner service through Amazon ECR

The following topic describes how to create and launch an AWS App Runner service from the Amazon Elastic Container Registry (Amazon ECR) node, in the AWS Toolkit for Visual Studio Code. For detailed information about the AWS App Runner and Amazon ECR services, see the AWS App Runner and Amazon ECR User Guides.

Prerequisites

Before you can create and launch an AWS App Runner from Amazon ECR in the AWS Toolkit, you must complete the following. For a detailed guide on how to complete these procedures, see the Working with Amazon Elastic Container Registry topic in this User Guide.

1. Create a `dockerfile`.
2. Build an image from your `dockerfile`.
3. Create a new repository.
4. Tag and push an image to your repository.

Creating an AWS App Runner service from an existing Amazon ECR repository

The following procedure describes how to create an AWS App Runner service from an existing Amazon ECR repository, in the AWS Toolkit.

1. From the AWS Explorer, expand the region that contains the Amazon ECR repository you want to create an AWS App Runner service from.
2. Expand the Amazon ECR service node to view your Amazon ECR repositories.
3. Open the context menu for (right-click) the Amazon ECR repository or repository image you want to create an AWS App Runner service from.
4. From the context menu, choose **Create App Runner Service** to open the AWS App Runner creation wizard in VS Code
5. From **Enter a port for the new service (1/5)**, enter the port number you want to use, then press **`Enter`** to continue.
6. From **Configure environment variables (2/5)**, choose **Use file...** to browse to select browse your local files or choose **Skip** to skip this step.
7. From **Select a role to pull from ECR (3/5)**, choose an existing IAM role from the list.

   > **Note**
   >
   > The **AppRunnerECRAccessRole** access role is required to create an AWS App Runner service from an Amazon ECR private registry. If a valid role isn't available from the list, choose the **+ (Create Role...)** icon to automatically create and assign **AppRunnerECRAccessRole** to your registry.

8. From **Name your service (4/5)**, enter a name for your new service, then press **`Enter`** to continue.
9. From **Select instance configuration (5/5)** choose the **`vCPU`** and **`Memory`** configuration from the list to create your new service.
10. From the AWS Explorer, expand the **App Runner** service node to view your AWS App Runner resources. When your new service has been created successfully, the status automatically updates to **Running**.

Working with Amazon Elastic Container Service

The AWS Toolkit for Visual Studio Code provides some support for Amazon Elastic Container Service (Amazon ECS). The Toolkit for VS Code assists you in certain Amazon ECS-related work, such as creating task definitions.

Topics

Using IntelliSense for Amazon ECS task-definition files

One of the things that you might do when working with Amazon Elastic Container Service (Amazon ECS) is to create task definitions, as described in Creating a Task Definition from the *Amazon Elastic Container Service Developer Guide*. When you install the AWS Toolkit for Visual Studio Code, the installation includes IntelliSense functionality for Amazon ECS task-definition files.

Prerequisites

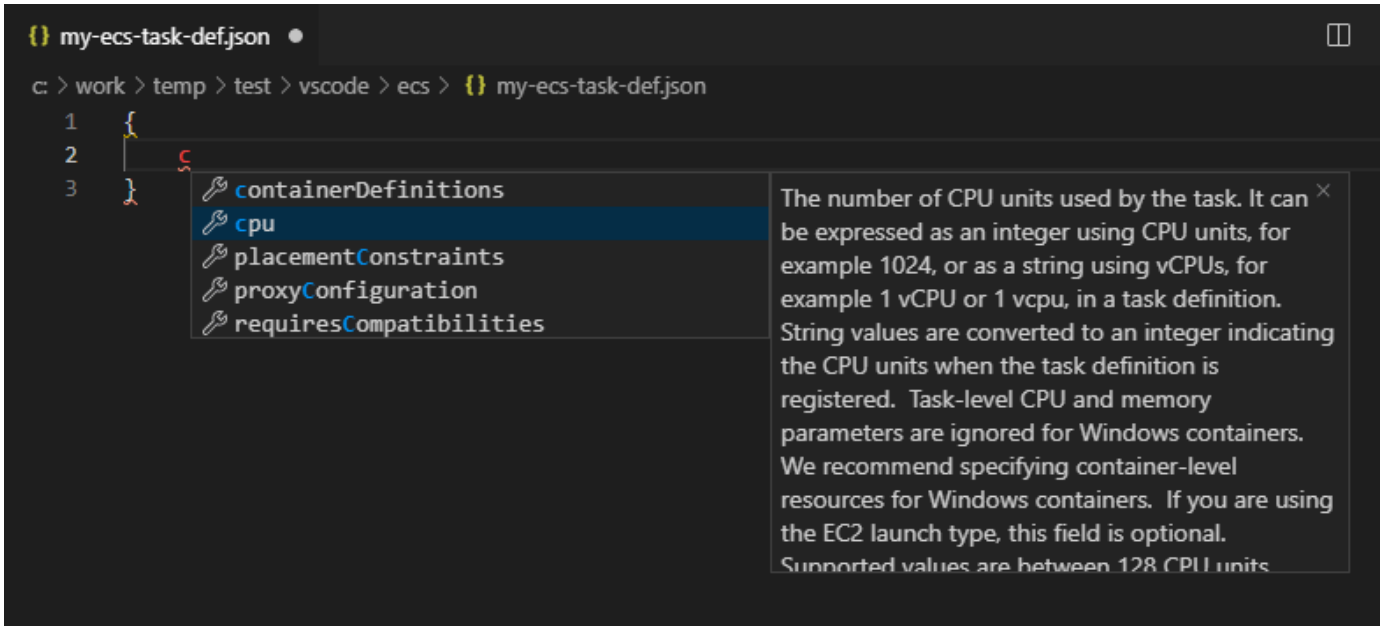
- Be sure your system meets the prerequisites specified in Installing the Toolkit for VS Code.

Use IntelliSense in Amazon ECS task-definition files

The following example shows you how you can take advantage of IntelliSense in Amazon ECS task-definition files.

1. Create a JSON file for your Amazon ECS task definition. The file's name must have `ecs-task-def.json` at the end, but can have additional characters at the beginning.

   For this example, create a file named `my-ecs-task-def.json`
2. Open the file in a VS Code editor and enter the initial curly braces.
3. Enter the letter "c" as if you wanted to add `cpu` to the definition. Observe the IntelliSense dialog that opens, which is similar to the following.



Amazon Elastic Container Service Exec in AWS Toolkit for Visual Studio Code

You can issue single commands in an Amazon Elastic Container Service (Amazon ECS) container with the AWS Toolkit for Visual Studio Code, using the Amazon ECS Exec feature.

> **Important**
>
> Enabling and Disabling Amazon ECS Exec changes the state of resources in your AWS account. This includes stopping and restarting the service. Altering the state of resources while the Amazon ECS Exec is enabled can lead to unpredictable results. For more information about Amazon ECS, see the developer guide Using Amazon ECS Exec for Debugging.

Amazon ECS Exec prerequisites

Before you can use the Amazon ECS Exec feature, there are some prerequisite conditions that need to be met.

Amazon ECS requirements

Depending on whether your tasks are hosted on Amazon EC2 or AWS Fargate, Amazon ECS Exec has different version requirements.

- If you're using Amazon EC2, you must use an Amazon ECS optimized AMI that was released after January 20th, 2021, with an agent version of 1.50.2 or greater. Additional information is available for you in the developer guide Amazon ECS optimized AMIs.
- If you're using AWS Fargate, you must use platform version 1.4.0 or higher. Additional information about Fargate requirements is available to you in the developer guide AWS Fargate platform versions.

AWS account configuration and IAM permissions

To use the Amazon ECS Exec feature, you need to have an existing Amazon ECS cluster associated with your AWS account. Amazon ECS Exec uses Systems Manager to establish a connection with the containers on your cluster and requires specific Task IAM Role Permissions to communicate with the SSM service.

You can find IAM role and policy information, specific to Amazon ECS Exec, in the IAM permissions required for ECS Exec developer guide.

Working with the Amazon ECS Exec

You can enable or disable the Amazon ECS Exec directly from the AWS Explorer in the Toolkit for VS Code. When you have enabled Amazon ECS Exec, you can choose containers from the Amazon ECS menu and then run commands against them.

Enabling Amazon ECS Exec

1. From the AWS Explorer, locate and expand the Amazon ECS menu.
2. Expand the cluster with the service that you want to modify.
3. Open the context menu for (right-click) the service and choose **Enable Command Execution**.

> **Important**
>
> This will start a new deployment of your Service and may take a few minutes. For more information, see the note at the beginning of this section.

Disabling Amazon ECS Exec

1. From the AWS Explorer, locate and expand the Amazon ECS menu.
2. Expand the cluster that houses the service you want.
3. Open the context menu for (right-click) the service and choose **Disable Command Execution**.

> **Important**
>
> This will start a new deployment of your Service and may take a few minutes. For more information, see the note at the beginning of this section.)

Running commands against a Container

To run commands against a container using the AWS Explorer, Amazon ECS Exec must be enabled. If it's not enabled, see the **Enabling ECS Exec** procedure in this section.

1. From the AWS Explorer, locate and expand the Amazon ECS menu.
2. Expand the cluster that houses the service you want.
3. Expand the service to list the associated containers.
4. Open the context menu for (right-click) the container and choose **Run Command in Container**.
5. A **prompt** will open with a list of running Tasks, choose the **Task ARN** that you want.

   > **Note**
   >
   > If only one Task is running for that Service, it will be auto-selected and this step will be skipped.

6. When prompted, type the command that you want to run and press **Enter** to process.

Working with Amazon EventBridge

The AWS Toolkit for Visual Studio Code (VS Code) provides support for Amazon EventBridge. Using the Toolkit for VS Code, you can work with certain aspects of EventBridge, such as schemas.

Topics

- Working with Amazon EventBridge Schemas

Working with Amazon EventBridge Schemas

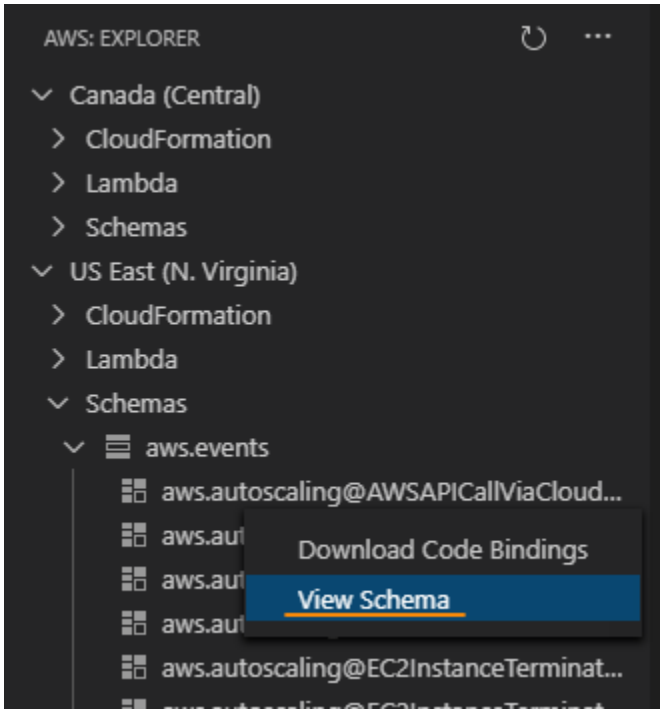
You can use the AWS Toolkit for Visual Studio Code (VS Code) to perform various operations on Amazon EventBridge schemas.

Prerequisites

- Be sure your system meets the prerequisites specified in Installing the Toolkit for VS Code.
- The EventBridge schema you want to work with must be available in your AWS account. If it isn't, create or upload it. See Amazon EventBridge Schemas in the Amazon EventBridge User Guide.

View an Available Schema

1. In the **AWS Explorer**, expand **Schemas**.
2. Expand the name of the registry that contains the schema you want to view. For example, many of the schemas that AWS supplies are in the **aws.events** registry.
3. To view a schema in the editor, open the context menu of the schema, and then choose **View Schema**.



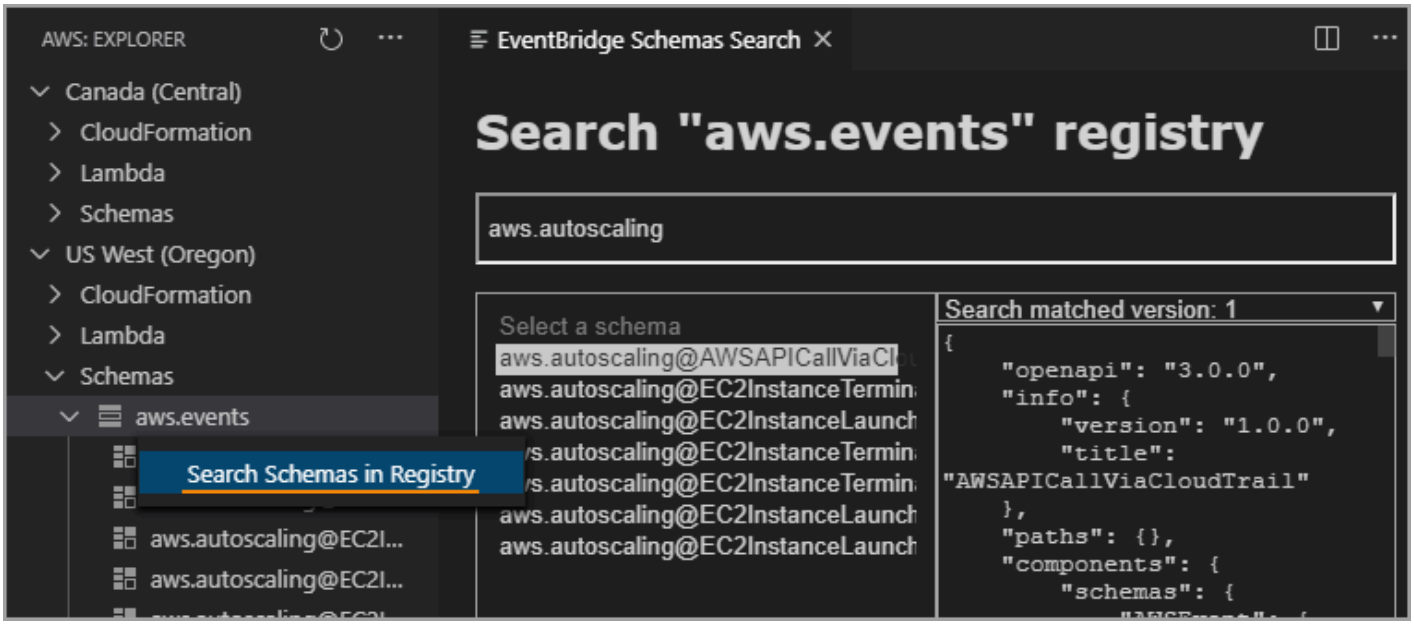
Find an Available Schema

In the **AWS Explorer**, do one or more of the following:

- Begin typing the title of the schema you want to find. The **AWS Explorer** highlights the schema titles that contain a match. (A registry must be expanded for you to see the highlighted titles.)

- Open the context menu for **Schemas**, and then choose **Search Schemas**. Or expand **Schemas**, open the context menu for the registry that contains the schema you want to find, and then choose **Search Schemas in Registry**. In the **EventBridge Schemas Search** dialog box, begin typing the title of the schema you want to find. The dialog box displays the schema titles that contain a match.

  To display the schema in the dialog box, select the title of the schema.



Generate Code for an Available Schema

1. In the **AWS Explorer**, expand **Schemas**.
2. Expand the name of the registry that contains the schema you want to generate code for.
3. Right-click the title of the schema, and then choose **Download code bindings**.
4. In the resulting wizard pages, choose the following:

   - The **Version** of the schema

- The code binding language
- The workspace folder where you want to store the generated code on your local development machine

AWS IAM Access Analyzer

You can run AWS Identity and Access Management (IAM) Access Analyzer policy checks on your IAM policies authored in CloudFormation templates, Terraform plans, and JSON policy documents, using the IAM Access Analyzer in the AWS Toolkit for Visual Studio Code.

IAM Access Analyzer policy checks include policy validation and custom policy checks. Policy validation helps validate your IAM policies according to the standards detailed in the Grammar of the IAM JSON policy language and AWS Security best practices in IAM topics, located in the *AWS Identity and Access Management* User Guide. Your policy validation findings include security warnings, errors, general warnings, and policy suggestions.

You can also run custom policy checks for new access, based on your security standards. A charge is associated with each custom policy check for new access. For detailed information about pricing, see the AWS IAM Access Analyzer pricing site. For details about IAM Access Analyzer policy checks, see the Checks for validating policies topic in the *AWS Identity and Access Management* User Guide.

The following topics describe how to work with IAM Access Analyzer policy checks in the AWS Toolkit for Visual Studio Code.

Topics

- Working with AWS IAM Access Analyzer

Working with AWS IAM Access Analyzer

The following sections describe how to perform IAM policy validation and custom policy checks in the AWS Toolkit for Visual Studio Code. For additional details, see the following topics in the AWS Identity and Access Management User Guide: IAM Access Analyzer policy validation and IAM Access Analyzer custom policy checks.

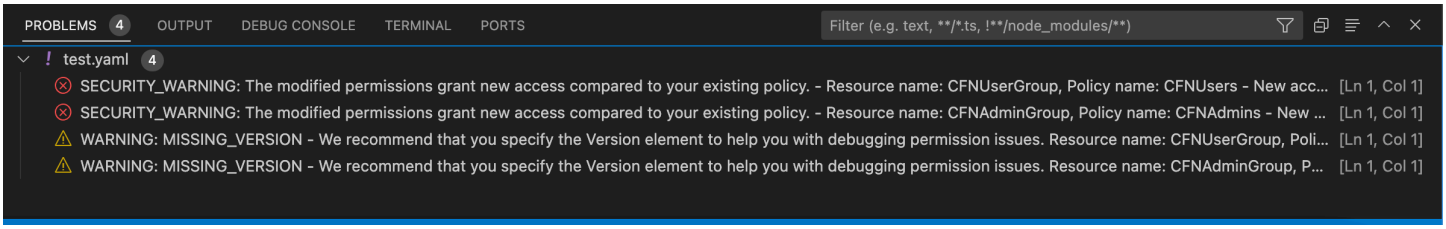
Prerequisites

The following prerequisites must be met before you can work with IAM Access Analyzer policy checks from the Toolkit.

- Install Python version 3.6 or later.
- Install either the IAM Policy Validator for CloudFormation or the IAM Policy Validator for Terraform that is required by Python CLI tools and specified in the IAM Policy Checks window.
- Configure your AWS Role credentials.

IAM Access Analyzer policy checks

You can perform policy checks for CloudFormation templates, Terraform plans, and JSON Policy documents, using the AWS Toolkit for Visual Studio Code. Your check findings are viewable in the VS Code **Problems Panel**. The following image shows the VS Code **Problems Panel**.



IAM Access Analyzer provides 4 types of checks:

- Validate Policy
- CheckAccessNotGranted
- CheckNoNewAccess
- CheckNoPublicAccess

The following sections describe how to run each type of check.

> **Note**
>
> Configure your AWS Role credentials prior to running any type of check. Supported files include the following document types: CloudFormation templates, Terraform plans, and JSON Policy documents
> File path references are typically provided by your administrator or security team, and can be a system file path or an Amazon S3 bucket URI. To use an Amazon S3 bucket URI, your current role must have access to the Amazon S3 bucket.
> A charge is associated with each custom policy check. For details about custom policy check pricing, see the AWS IAM Access Analyzer pricing guide.

Running Validate Policy

The Validate Policy check, also known as policy validation, validates your policy against IAM policy grammar and AWS best practices. For additional information, see the Grammar of the IAM JSON policy language and AWS Security best practices in IAM topics, located in the *AWS Identity and Access Management* User Guide.

1. From VS Code, open a supported file that contains AWS IAM Policies, in the VS Code editor.
2. To open IAM Access Analyzer policy checks, open the VS Code Command Pallete by pressing **`CRTL+Shift+P`**, search for **`IAM Policy Checks`**, then click to open the **IAM Policy Checks** pane in the VS Code editor.
3. From the **IAM Policy Checks** pane, select your document type from the drop-down menu.
4. From the **Validate Policies** section, choose the **Run Policy Validation** button to run the Validate Policy check.
5. From the **Problems Panel** in VS Code, review your policy check findings.
6. Update your policy and repeat this procedure, re-running the Validate Policy check until your policy check findings no longer display security warnings or errors.

Running CheckAccessNotGranted

CheckAccessNotGranted is a custom policy check to verify that specific IAM actions are not allowed by your policy.

> **Note**
>
> File path references are typically provided by your administrator or security team, and can be a system file path or an Amazon S3 bucket URI. To use an Amazon S3 bucket URI, your current role must have access to the Amazon S3 bucket. At least one action or resource must be specified and the file should be structured after the following example:
>
> ```
> {"actions": ["action1", "action2", "action3"], "resources":
> ["resource1", "resource2", "resource3"]}
> ```

1. From VS Code, open a supported file that contains AWS IAM Policies, in the VS Code editor.

2. To open IAM Access Analyzer policy checks, open the VS Code Command Pallete by pressing **CRTL+Shift+P**, search for **IAM Policy Checks**, then click to open the **IAM Policy Checks** pane in the VS Code editor.
3. From the **IAM Policy Checks** pane, select your document type from the drop-down menu.
4. From the **Custom Policy Checks** section, select **CheckAccessNotGranted**.
5. In the text-input field, you can enter a comma-separated list that contains actions and resource ARNs. At least one action or resource must be provided.
6. Choose the **Run Custom Policy Check** button.
7. From the **Problems Panel** in VS Code, review your policy check findings. Custom policy checks return a PASS or FAIL result.
8. Update your policy and repeat this procedure, re-running the CheckAccessNotGranted check until it returns PASS.

Running CheckNoNewAccess

CheckNoNewAccess is a custom policy check to verify whether your policy grants new access compared to a reference policy.

1. From VS Code, open a supported file that contains AWS IAM Policies, in the VS Code editor.
2. To open IAM Access Analyzer policy checks, open the VS Code Command Pallete by pressing **CRTL+Shift+P**, search for **IAM Policy Checks**, then click to open the **IAM Policy Checks** pane in the VS Code editor.
3. From the **IAM Policy Checks** pane, select your document type from the drop-down menu.
4. From the **Custom Policy Checks** section, select **CheckNoNewAccess**.
5. Input a reference JSON policy document. Alternatively, you can provide a file path that references a JSON policy document.
6. Select the **Reference Policy Type** that matches the type of your reference document.
7. Choose the **Run Custom Policy Check** button.
8. From the **Problems Panel** in VS Code, review your policy check findings. Custom policy checks return a PASS or FAIL result.
9. Update your policy and repeat this procedure, re-running the CheckNoNewAccess check until it returns PASS.

Running CheckNoPublicAccess

CheckNoPublicAccess is a custom policy check to verify whether your policy grants public access to supported resource types within your template.

For specific information about supported resource types, see the cloudformation-iam-policy-validator and terraform-iam-policy-validator GitHub repositories.

1. From VS Code, open a supported file that contains AWS IAM Policies, in the VS Code editor.
2. To open IAM Access Analyzer policy checks, open the VS Code Command Pallete by pressing **`CRTL+Shift+P`**, search for **`IAM Policy Checks`**, then click to open the **IAM Policy Checks** pane in the VS Code editor.
3. From the **IAM Policy Checks** pane, select your document type from the drop-down menu.
4. From the **Custom Policy Checks** section, select **CheckNoPublicAccess**.
5. Choose the **Run Custom Policy Check** button.
6. From the **Problems Panel** in VS Code, review your policy check findings. Custom policy checks return a PASS or `FAIL` result.
7. Update your policy and repeat this procedure, re-running the CheckNoNewAccess check until it returns PASS.

Working with AWS IoT in AWS Toolkit for Visual Studio Code

AWS IoT in AWS Toolkit for Visual Studio Code allows you to interact with the AWS IoT service, while minimizing interruptions to your work flow in VS Code. This user guide is intended to help you get started using the AWS IoT service features that are available in the AWS Toolkit for Visual Studio Code. For additional information about the AWS IoT service, see the developer guide What is AWS IoT?

AWS IoT prerequisites

To get started using AWS IoT from Toolkit for VS Code, make sure your AWS account and VS Code meet the requirements in these guides:

- For AWS account requirements and AWS user permissions specific to the AWS IoT service, see the Getting Started with AWS IoT Core developer guide.
- For Toolkit for VS Code specific requirements, see the Setting up the Toolkit for VS Code user guide.

AWS IoT Things

AWS IoT connects devices to AWS cloud services and resources. You can connect your devices to AWS IoT by using objects called **things**. A thing is a representation of a specific device or logical entity. It can be a physical device or sensor (for example, a light bulb or a switch on a wall). For additional information about AWS IoT things, see the developer guide Managing devices with AWS IoT.

Managing AWS IoT things

The Toolkit for VS Code has several features that make your AWS IoT thing management more efficient. These are ways that you can use the VS Code toolkit to manage your AWS IoT things:

- Create a thing
- Attach a certificate to a thing
- Detach a certificate from a thing
- Delete a thing

To create a thing

1. From the AWS Explorer, expand the **IoT** service heading, and context-select (right-click) **Things**.
2. Choose **Create Thing** from the context-menu to open a dialog box.
3. Follow the prompt by entering a name for your IoT thing into the **Thing Name** field.
4. When this is complete, a **thing icon** followed by the name you specified will be visible in the **Thing** section.

To attach a certificate to a thing

1. From the AWS Explorer, expand the **IoT** service section.
2. Under the **Things** subsection, locate the **thing** where you are attaching the certificate.
3. Context-select (right-click) the **thing** and choose **Attach Certificate** from the context-menu, to open an input selector with a list of your certificates.
4. From the list, choose the **certificate ID** that corresponds to the certificate you want to attach to your thing.

5. When this is complete, your certificate is accessible in the AWS explorer, as an item of the thing that you attached it to.

To detach a certificate from a thing

1. From the AWS Explorer, expand the **IoT** service section
2. In the **Things** subsection, locate the **thing** that you want to detach a certificate from.
3. Context-select (right-click) the **thing** and choose **Detach Certificate** from the context-menu.
4. When this is complete, the detached certificate will no longer display under that thing in the AWS Explorer, but it will still be accessible from the **Certificates** subsection.

To delete a thing

1. From the AWS Explorer, expand the **IoT** service section.
2. In the **Things** subsection, locate the **thing** you want to delete.
3. Context-select (right-click) the thing and choose **Delete Thing** from the context-menu to delete it.
4. When this is complete, the deleted thing will no longer be available from the **Things** subsection.

> **Note**
>
> Note: You can only delete a thing that doesn't have a certificate attached to it.

AWS IoT certificates

Certificates are a common way to create a secure connection between your AWS IoT services and devices. X.509 certificates are digital certificates that use the X.509 public key infrastructure standard to associate a public key with an identity contained in a certificate. For additional information about AWS IoT certificates, see the developer guide Authentication (IoT).

Managing certificates

The VS Code toolkit offers a variety of ways for you to manage your AWS IoT certificates, directly from the AWS Explorer.

- Create a certificate
- Change a certificate status
- Attach a policy to a certificate
- Delete a certificate

To create an AWS IoT certificate

An X.509 certificate can be used to connect with your instance of AWS IoT.

1. From the AWS Explorer, expand the **IoT** service section, and context-select (right-click) **Certificates**.
2. Choose **Create Certificate** from the context-menu to open a dialog box.
3. Select a directory in your local file system to save your RSA key pair and X.509 certificate.

> **Note**
>
> - The default file names contain the certificate ID as a prefix.
> - Only the X.509 certificate is stored with your AWS account, through the AWS IoT service.
> - Your RSA key pair can only be issued once, save them to a secure location in your file system when you're prompted.
> - If either the certificate or the key pair can't be saved to your file system at this time, then the AWS Toolkit deletes the certificate from your AWS account.

To modify a certificate status

The status of an individual certificate is displayed next to its ID in the AWS Explorer and can be set to: active, inactive, or revoked.

Note

- Your certificate needs an **active** status before you can use it to connect your device to your AWS IoT service.
- An **inactive** certificate can be activated, whether it has been deactivated previously or is inactive by default.
- A certificate that has been **revoked** can't be reactivated.

1. From the AWS Explorer, expand the **IoT** service section.
2. In the **Certificates** subsection, locate the certificate you want to modify.
3. Context-select (right-click) the certificate to open a context menu that displays the status change options available for that certificate.

- If a certificate has the status **inactive**, choose **activate** to change the status to **active**.
- If a certificate has the status **active**, choose **deactivate** to change the status to **inactive**.
- If a certificate has either an **active** or **inactive** status, choose **revoke** to change the status to **revoked**.

Note

Each of these status-changing actions are also available if you select a certificate that is attached to a thing while it's displayed in the **Things** subsection.

To attach an IoT policy to a certificate

1. From the AWS Explorer, expand the **IoT** service section.
2. In the **Certificates** subsection, locate the certificate you want to modify.
3. Context-select (right-click) the certificate and choose **Attach Policy** from the context menu, to open an input selector with a list of your available policies.
4. Choose the policy you want to attach to the certificate.

5. When this is complete, the policy you selected will be added to the certificate as a sub-menu item.

To detach an IoT policy from a certificate

1. From the AWS Explorer, expand the **IoT** service section.
2. In the **Certificates** subsection, locate the certificate you want to modify.
3. Expand the certificate and locate the policy you want to detach.
4. Context-select (right-click) the policy and choose **Detach** from the context menu.
5. When this is complete, the policy will no longer be an item that is accessible from your certificate, but it will be available from the **Policy** subsection.

To delete a certificate

1. From the AWS Explorer, expand the **IoT** service heading.
2. In the **Certificates** subsection, locate the certificate you want to delete.
3. Context-select (right-click) the certificate and choose **Delete Certificate** from the context menu.

> **Note**
>
> You can't delete a certificate if it's attached to a thing or has an active status. You can delete a certificate that has attached policies.

AWS IoT policies

AWS IoT Core policies are defined through JSON documents, each containing one or more policy statements. Policies define how AWS IoT, AWS, and your device can interact with each other. For more information about how to create a policy document, see the developer guide IoT Polices.

> **Note**
>
> Named policies are versioned so you can roll them back. In The AWS Explorer, your IoT polices are listed under the **Policies** subsection, in the IoT service. You can view policy versions by expanding a policy. The default version is denoted by an asterisk.

Managing policies

The Toolkit for VS Code offers several ways for you to manage your AWS IoT service policies. These are ways that you can manage or modify your policies directly from the AWS Explorer in VS Code:

- Create a policy
- Upload a new policy version
- Edit a policy version
- Change the policy version defualt
- Change the policy version defualt

To create an AWS IoT policy

> **Note**
>
> You can create a new policy from the AWS Explorer, but the JSON document that defines the policy must already exist in your file system.

1. From the AWS Explorer, expand the **IoT** service section.
2. Context-select (right-click) the **Policies** subsection and choose **Create Policy from Document**, to open the **Policy Name** input field.
3. Enter a name and follow the prompts to open a dialog asking you to select a JSON document from your file system.
4. Choose the JSON file that contains your policy definitions, the policy will be available in the AWS explorer when this is complete.

To upload a new AWS IoT policy version

A new version of a policy can be created by uploading a JSON document to the policy.

> **Note**
>
> The new JSON document must be present on your file system to create a new version using the AWS Explorer.

1. From the AWS Explorer, expand the **IoT** service section.
2. Expand the **Policies** subsection to view your AWS IoT policies
3. Context-select (right-click) the policy that you want to update and choose **Create new version from Document**.
4. When the dialog opens, choose the JSON file that contains the updates to your policy definitions.
5. The new version will be accessible from your policy in the AWS Explorer.

To edit an AWS IoT policy version

A policy document can be opened and edited using VS Code. When you are finished editing the document, you can save it to your file system. Then, you can upload it to your AWS IoT service from the AWS Explorer.

1. From the AWS Explorer, expand the **IoT** service section.
2. Expand the **Policies** subsection and locate the policy you want to update.**Create Policy from Document** to open the **Policy Name** input field.
3. Expand the policy that you want to update and then Context-select (right-click) the policy version that you want to edit.
4. Choose **View** from the context-menu to open the policy version in VS Code
5. When the policy document is opened, make and save the changes you want.

> **Note**
>
> At this point, the changes you made to the policy are only saved to your local file system. To update the version and track it with the AWS Explorer, repeat the steps described in the Upload a new policy version procedure.

To select a new policy version default

1. From the AWS Explorer, expand the **IoT** service section.
2. Expand the **Policies** subsection and locate the policy you want to update.
3. Expand the policy that you want to update and then Context-select (right-click) the policy version that you want to set and choose **Set as Default**.
4. When this is complete, the new default version you selected will have a star located next it.

To delete policies

> **Note**
>
> Before you can delete a policy or a policy version, there are conditions that need to be met.
>
> - You can't delete a policy if it's attached to a certificate.
> - You can't delete a policy if it has any non-default versions.
> - You can't delete the default version of a policy unless a new default version is selected, or the entire policy is deleted.
> - Before you can delete an entire policy, all of the non-default version of that policy must be deleted first.

1. From the AWS Explorer, expand the **IoT** service section.
2. Expand the **Policies** subsection and locate the policy you want to update.
3. Expand the policy that you want to update and then Context-select (right-click) the policy version that you want delete and choose **Delete**.
4. When a version is deleted, it will no longer be visible from the Explorer.

5. When the only version left for a policy is the default, then you can context-select (right-click) the parent policy and choose **Delete** to delete it.

AWS Lambda Functions

The AWS Toolkit for Visual Studio Code provides comprehensive support for AWS Lambda functions, enabling you to build, test, and deploy directly from VS Code.

Lambda is a fully managed, event-driven compute service that automatically runs your code in response to events from over 200 AWS services and software-as-a-service (SaaS) applications. For detailed information about the AWS Lambda service, see the AWS Lambda Developer Guide.

The following topics decribe how to work with AWS Lambda in the AWS Toolkit for Visual Studio Code.

Topics

Working with AWS Lambda Functions

The AWS Toolkit for Visual Studio Code allows you to work with your AWS Lambda functions in your local VS Code environment. With the AWS Toolkit, you can create, edit, test, debug, and deploy your Lambda functions, without having to leave the IDE. For detailed information about the AWS Lambda service, see the AWS Lambda Developer Guide.

The following sections describe how to get started working with Lambda functions in the AWS Toolkit for Visual Studio Code.

> **Note**
>
> If you have already created Lambda functions by using the AWS Management Console, then you can invoke them from the Toolkit. Additionally, you can open your Lambda functions into VS Code from the AWS Lambda console, for additional information, see the AWS

Lambda console to IDE topic in this user guide. To create a new Lambda function in VS Code, follow the steps outlined in the Creating a new serverless application (local) topic in this user guide.

Prerequisites

The following conditions must be met to work with the AWS Lambda service in the AWS Toolkit.

- The latest version of the AWS Toolkit for Visual Studio Code is installed and set up with your AWS credentials.
- Your AWS Identity and Access Management (IAM) managed permissions and policies are configured to work with the AWS Lambda service. For detailed information on how to configure your permissions and create a compatible AWS managed policy, see the AWS Identity and Access Management for AWS Lambda topic in the *AWS Lambda Developer Guide*.
- You have existing AWS Lambda functions or are familiar with how to create one. For instructions on how to create a Lambda function, see the Create your first Lambda function topic in the *AWS Lambda Developer Guide*.

Invoking a Lambda Function

To invoke a Lambda function from your AWS account into VS Code, complete the following steps.

1. From the AWS Toolkit for Visual Studio Code, expand the AWS explorer.
2. From the AWS explorer, expand **Lambda** to view your Lambda resources.
3. Open the context menu for (right-click) the Lambda function your want to invoke, then choose **Invoke in the cloud** or choose the **Invoke in the cloud** icon to open the **Remote invoke configuration** menu in VS Code.
4. From the **Remote invoke configuration** menu, specify your **Payload** settings and add any additional information that is required for the event.

> **Note**
>
> The first invoke process may start running as soon as you choose **Invoke in the cloud** in the AWS explorer. The output is displayed in the **OUTPUT** tab of the VS Code terminal.

5. Choose the **Remote Invoke** button to invoke your function, The output is displayed in the **OUTPUT** tab of the VS Code terminal.

Deleting a Lambda function

To delete a Lambda function, complete the following procedure.

> **Warning**
>
> Do not use this procedure to delete Lambda functions that are associated with CloudFormation. These functions must be deleted through your CloudFormation stack.

1. From the AWS Toolkit for Visual Studio Code, expand the AWS explorer.
2. From the AWS explorer, expand **Lambda** to view your Lambda resources.
3. Right-click the Lambda function your want to delete, then choose **Delete**.
4. When prompted, confirm that you want to delete your function.

After the function is deleted, it's no longer listed in the AWS explorer.

Downloading a Lambda function

You can download code from a remote Lambda function into your VS Code workspace for editing and debugging.

> **Note**
>
> To download your Lambda function, you must be working in a VS Code workspace with an accessible folder and the AWS Toolkit only supports this feature with Lambda functions using Node.js and Python runtimes.

1. From the AWS Toolkit for Visual Studio Code, expand the AWS explorer.
2. From the AWS explorer, expand **Lambda** to view your Lambda resources.
3. Right-click the Lambda function your want to download, then choose **Download**.
4. Your Lambda function opens in the VS Code editor and displays in the AWS explorer when the download is complete. The AWS Toolkit also creates a *launch configuration* in the VS Code

run panel allowing you to run and debug the Lambda function locally with AWS Serverless Application Model. For more information about using AWS SAM, see the section called "Running and debugging a serverless application from template (local)".

Deploying updates for new Lambda functions

You can deploy updates to new Lambda functions from an unspecified, temporary location on your local machine.

> **Note**
>
> When there are un-deployed changes to your lambda files, you're notified by the **M** icon located next to the modified files in the VS Code editor and in the AWS explorer.

Deploying from the VS Code editor

1. Open a file from your Lambda function in the VS Code editor, then make a change to the file.
2. Manually save from the VS Code main menu or pressing **`option+s`** (Mac) **`ctrl+s`** (Windows).
3. VS Code automatically prompts you about deploying your changes to the cloud, choose the **Deploy** button to confirm the deployment.
4. VS Code updates you on the status of your deployment and notifies you when the process is complete.

Deploying from the AWS Explorer

1. Open a file from your Lambda function in the VS Code editor, then make a change to the file.
2. From the AWS Toolkit, expand the AWS explorer.
3. From the AWS explorer, expand the AWS region with the Lambda function that you want to deploy changes for.
4. From the AWS region, expand Lambda and navigate the function that you want to deploy changes for.
5. From the quick menu next to your function, choose the **Save and deploy your code** icon.
6. VS Code updates you on the status of your deployment and notifies you when the process is complete.

Uploading updates for existing Lambda functions

The following procedures describe how to upload local changes made to your existing Lambda functions. This feature supports uploads with any Lambda supported runtime.

> **Warning**
>
> Before uploading your lambda function, be aware of the following:
>
> - Updating code in this way doesn't use the AWS SAM CLI for deployment or create an CloudFormation stack
> - The AWS Toolkit doesn't validate code. Validate your code and test your function(s) before uploading any changes to the cloud.

Uploading a Zip Archive

1. From the AWS Toolkit for Visual Studio Code, expand the AWS explorer.
2. From the AWS explorer, expand **Lambda** to view your Lambda resources.
3. Right-click the Lambda function your want to upload your changes to, then choose **Upload Lambda...** to open the **Select Upload Type** menu.
4. Choose **ZIP Archive** to locate the `ZIP Archive` in your local directory.
5. When prompted, confirm the upload to start the upload of the selected `ZIP Archive`.
6. The status of your upload is displayed in VS Code and you're notified when the upload process is complete.

Uploading a directory without building

1. From the AWS Toolkit for Visual Studio Code, expand the AWS explorer.
2. From the AWS explorer, expand **Lambda** to view your Lambda resources.
3. Right-click the Lambda function your want to upload your changes to, then choose **Upload Lambda...** to open the **Select Upload Type** menu.
4. Choose **Directory** to proceed to the **Build directory** screen.
5. From the **Build directory** screen, choose **No** to choose a local directory for upload.
6. When prompted, confirm the upload to upload the selected directory.

7. The status of your upload is displayed in VS Code and you're notified when the upload process is complete.

Uploading a directory with a build

> **Note**
>
> Be aware of the following:
>
> - This procedure requires the AWS Serverless Application Model CLI.
> - The AWS Toolkit notifies you a matching handler can't be detected prior to upload.
> - To change the handler attached to your Lambda function, use the AWS Lambda console or the AWS Command Line Interface.

1. From the AWS Toolkit for Visual Studio Code, expand the AWS explorer.
2. From the AWS explorer, expand **Lambda** to view your Lambda resources.
3. Right-click the Lambda function your want to upload your changes to, then choose **Upload Lambda...** to open the **Select Upload Type** menu.
4. Choose **Directory** to proceed to the **Build directory** screen.
5. From the **Build directory** screen, choose **Yes**, then select a local directory for upload.
6. When prompted, confirm the upload to start building and uploading the selected directory.
7. The status of your upload is displayed in VS Code and you're notified when the upload process is complete.

Converting your Lambda function to an AWS SAM project

To convert your Lambda function into an AWS SAM stack, complete the following steps.

> **Warning**
>
> Currently, only a subset of resources are supported when converting a Lambda function to an AWS SAM project. To locate any missing resources after a conversion, check the Lambda console and add them manually to your AWS SAM template. For additional details about

supported and unsupported resources, see the Resource type support topic in the *AWS CloudFormation Developer Guide*.

1. From the AWS Toolkit, expand the AWS explorer.
2. From the AWS explorer, expand the AWS region with the Lambda function that you want to convert into an AWS SAM project.
3. From the AWS region, expand Lambda and navigate the function that you want to convert into an AWS SAM stack.
4. From the quick menu next to your Lambda function, choose the **Convert to SAM Application** icon to browse your local file system and specify a location for your new AWS SAM project.
5. After specifying a location the AWS Toolkit begins converting your Lambda function into an AWS SAM project, VS Code provides updates on the status of the process.

   > **Note**
   >
   > This process may take a few minutes.

6. When prompted by VS Code, enter a stack name, then press the **`Enter`** key to continue.
7. VS Code continues to update you with the status of your project, then notifies your when the process is complete and opens your new AWS SAM project as a VS Code workspace.

AWS Lambda console to IDE

The AWS Lambda console to IDE feature allows you to download your AWS Lambda functions from the AWS Lambda console into VS Code. Working with your Lambda functions in VS Code gives you access to other local-development options such as AWS Serverless Application Model (AWS SAM) and the AWS Cloud Development Kit (AWS CDK).

For more information about AWS Lambda, see the AWS Lambda Developer Guide. To get started working with your Lambda function in the AWS Toolkit, see the Working with AWS Lambda Functions topic in this user guide. The following sections describe how to move your work flow from the Lambda console to VS Code. For detailed information about moving your Lambda functions from the Lambda console to VS Code, including how to get started working with the Lambda console, see the Developing Lambda functions locally with VS Code topic in the *AWS Lambda Developer Guide*.

Moving from console to local development

To open a Lambda function from the Lambda console in VS Code, complete the following steps:

1. From your web browser, open the Lambda console.
2. From Lambda console, choose the function you want to open in VS Code.
3. From the function view, navigate to the **Code source** tab.
4. From the **Code source** tab, choose **Open in VS Code**.

Working with your Lambda function in VS Code

When your Lambda function opens in VS Code via the Lambda console:

- VS Code automatically launches on your local machine.
- Your Lambda function opens as a VS Code workspace.
- Your Lambda `handler file` opens in the VS Code editor.

> **Note**
>
> If there is not a properly configured `handler file` in the workspace, no file opens in the VS Code editor.

Opening your Lambda function in VS Code via the Lambda console allows you to access all of the existing AWS Toolkit Lambda features, including the ability to edit function code with full language support, local testing, remote debugging, deployment support, and dependency management. For more information about Lambda features supported in the AWS Toolkit, see the AWS Lambda service table of contents in this user guide.

AWS Lambda with LocalStack support

Build, test, and debug your serverless applications with LocalStack support in the AWS Toolkit for Visual Studio Code. LocalStack is an AWS Cloud emulator that allows for local testing of serverless applications.

For additional information about AWS Lambda, see the AWS Lambda *Developer Guide*. To learn more about LocalStack, visit their website LocalStack.

Prerequisites

The following are prerequisites to working with LocalStack in VS Code.

> **Note**
>
> The LocalStack CLI is installed during the setup process, but if you prefer a different version of the LocalStack CLI, the minium required version is *4.8.0*.

- A LocalStack Web Application account is required for access to all features available for the free and paid LocalStack tiers. LocalStack community edition is available without an account.
- Docker is required to work with LocalStack in VS Code. For more information about LocalStack requirements for Docker, see the LocalStack Docker Images topic in the LocalStack documentation.
- **Recommended:** The AWS Command Line Interface (AWS CLI) assists you in working with services in your simulated cloud environment.

Installing LocalStack

To install LocalStack free and paid tiered versions, complete the following steps.

> **Note**
>
> For instructions on how to set up LocalStack Community edition, see the *LocalStack Community* content in the *Setting up LocalStack* section of this topic.

1. From the AWS Toolkit, expand the **APPLICATION BUILDER** explorer.
2. Choose the **Open Walkthrough** button to open the **Get started building your application** walkthrough tab in the VS Code editor.
3. From the walkthrough, choose the **Install LocalStack** to start the LocalStack installation process in VS Code.

Setting up LocalStack

After you install the LocalStack extension for VS Code, you may see one of the following indicators when setup is needed:

- In the VS Code Status Bar, located in the lower-left corner of the IDE by default, the LocalStack status is red.
- VS Code prompts you to set up LocalStack.

There are two types of setup and configurations for LocalStack, depending on which version of LocalStack you're using. The following tabbed sections describe each LocalStack setup process.

> **Note**
>
> LocalStack auth tokens are required for the free and paid tier versions of LocalStack. For specific information about LocalStack pricing, see their Choose your plan pricing guide.

LocalStack free and paid tiers

There are 2 ways to set up LocalStack.

- From the VS Code **Setup LocalStack to get started** prompt, choose the **Setup** button.
- From the VS Code status bar, choose the LocalStack status icon to open the **Setup LocalStack to get started** prompt, then choose the **Setup** button.

During setup, the system goes through the following steps:

1. Installs the LocalStack CLI.
2. Checks to see if you have a LocalStack account.
3. If you have a LocalStack account, the system guides you through the authentication process in your default web browser. Similarly, if you do not have a LocalStack account, the system guides you through account setup before the authentication process.

After LocalStack is set up, the LocalStack status updates in the VS Code status bar.

> **Note**
>
> If you haven't created an AWS profile for LocalStack, then a new one is automatically created for you as part of the LocalStack setup process.

LocalStack Community

The Community edition of LocalStack is free to use and doesn't require you to sign up for an account, it runs from a Docker image that doesn't require a license. For additional details about LocalStack Community Edition, see the LocalStack Community image documentation. The following sections describe prerequisites and the basic setup that is required to work with LocalStack community edition in VS Code.

Launching a new instance

To launch a new instance of LocalStack Community, complete the following procedure.

> **Note**
>
> The following example starts a container instance of LocalStack on port 4566. If you specify different port values, you must update the port value specified in the procedure located in the *Configuring the AWS CLI and AWS Toolkit* section.

1. From VS Code, open the VS Code terminal by pressing **`ctrl + `(backtick)`**.
2. Enter the following into the terminal.

   Mac:

   ```
   docker run -d --name localstack_main \
   >> -p 4566:4566 \
   >> -v /var/run/docker.sock:/var/run/docker.sock \
   >> localstack/localstack
   ```

   Windows:

   ```
   docker run -d --name localstack_main `
   >> -p 4566:4566 `
   ```

```
>> -v /var/run/docker.sock:/var/run/docker.sock `
>> localstack/localstack
```

3. The terminal updates with the status of your Docker instance when the process is complete.

This containerized instance of LocalStack gives you access to the AWS services that you specified during the download process.

Configuring the CLI for LocalStack and Docker.

To configure the AWS CLI and AWS Toolkit to work with LocalStack in Docker, set up a new profile by completing the following steps:

1. From VS Code, open the VS Code terminal by pressing **`ctrl + `(backtick)`**.
2. Enter the following into the terminal.

```
~/.aws/credentials
[localstack]
aws_access_key_id = test
aws_secret_access_key = test
~/.aws/config
[profile localstack]
region = us-east-1
output = json
endpoint_url = http://localhost:4566 [default localstack endpoint]
```

3. The AWS Toolkit detects your LocalStack profile and updates the connection status menu.

After setup, choosing your LocalStack profile from the AWS profile section of the status bar makes your LocalStack resources visible in the AWS explorer. Additionally, you can view your LocalStack logs in the **Output** tab of the VS Code terminal.

Starting LocalStack in VS Code

You can start LocalStack using any of the following methods:

Starting LocalStack from the VS Code Status Bar

1. From VS Code, navigate to the status bar, then choose the **Start LocalStack** button to launch LocalStack.

2. The VS Code Status Bar updates when LocalStack has launched successfully.

Starting LocalStack from the VS Code Command Palette

1. From VS Code, open the **Command Palette** by pressing **`Cmd + Shift + P`** (Mac) or **`Control + Shift + P`** (Windows).
2. From the **Command Palette**, enter **`Start LocalStack`** in the search bar and choose it from the list when it populates in the results.
3. The VS Code Status Bar updates when LocalStack has launched successfully.

Starting LocalStack from the VS Code terminal

1. From VS Code, open the VS Code terminal by pressing **`ctrl + `(backtick)`**.
2. From the VS Code terminal, enter **`localstack start`** CLI command.
3. The VS Code Status Bar updates when LocalStack has launched successfully.

Building a sample serverless application

To start working with LocalStack in VS Code, you need a sample serverless application. If you already have an existing application in your AWS account you can deploy it locally using LocalStack or you can create a new application with AWS Serverless Land.

For additional information about creating an application with Serverless Land in the AWS Toolkit, see the Working with AWS Serverless Land topic in this User Guide. For detailed information about Serverless Land, see the Serverless Land web-application main landing page.

Testing and debugging Lambda functions with LocalStack

Testing and debugging your Lambda functions in the LocalStack VS Code extension is similar to working with your functions deployed to the AWS cloud. The main difference is that your AWS Toolkit instance must be authenticated with your LocalStack account to deploy and debug your functions with LocalStack.

> **Note**
>
> The testing and debugging features described in this section are not available for LocalStack Community edition.

To work with LocalStack in VS Code, connect to your LocalStack profile in the AWS Toolkit. When your LocalStack profile is active, the VS Code status bar shows **AWS: profile:localstack (custom endpoint)** with a check mark.

For detailed information about working with your Lambda functions in the AWS Toolkit, see the Working with AWS Lambda functions topic in this user guide.

AWS Lambda remote debugging

The AWS Toolkit for Visual Studio Code enables you to debug your AWS Lambda functions that are running in the cloud, directly in VS Code. With AWS Lambda remote debugging you can inspect running functions, set breakpoints, examine variables, and step-through debugging without modifying their existing development workflow.

The following sections describe how to work with Lambda remote debugging in the AWS Toolkit for Visual Studio Code.

How Lambda remote debugging works

The AWS Toolkit enables remote debugging by temporarily modifying your Lambda functions with an additional Lambda debugging layer and extending the Lambda invoke timeout limit to 900 seconds. A secure connection is established between your local debugger and the Lambda runtime environment using AWS IoT Secure Tunneling. This connection allows you to use your local-code breakpoints to step through the function as it executes remotely. After your debugging session is complete, all of the temporary modifications are automatically reverted to their original settings.

Getting Started

Supported runtimes

The following runtimes are supported by Lambda remote debugging.

- Python (Amazon Linux 2023)
- Java
- Typescript/JavaScript/Node.js (Amazon Linux 2023)

> **Note**
>
> Lambda managed instances and OCI image function types are not supported by Lambda remote debugging.

Prerequisites

Before you begin, the following prerequisites must be met.

- You must have valid AWS credentials configured in the AWS Toolkit. For additional details about installing the AWS Toolkit and configuring your credentials, see the Getting started topic in this user guide.
- A Lambda function has been deployed to your AWS account. For details on deploying a Lambda function, see the Create your first Lambda function topic in the *AWS Lambda* Developer Guide.
- You must have appropriate AWS Identity and Access Management (IAM) policy and permissions to debug your function. For additional details on Lambda permissions, see the AWS managed policies for AWS Lambda topic in the *AWS Lambda* Developer Guide. The following is an example of a policy that contains the minimum required permissions for working with Lambda remote debugging in the AWS Toolkit.

  > **Note**
  >
  > Remote debugging is enabled through AWS AWS IoT Secure Tunneling. This allows your local debugger to establish a secure connection to the Lambda runtime environment.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:ListFunctions",
        "lambda:GetFunction",
        "lambda:GetFunctionConfiguration",
        "lambda:GetLayerVersion",
        "lambda:UpdateFunctionConfiguration",
```

```
        "lambda:InvokeFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "iot:OpenTunnel",
        "iot:RotateTunnelAccessToken",
        "iot:ListTunnels"
      ],
      "Resource": "*"
    }
  ]
}
```

Accessing Lambda remote debugging

There are two main paths to access Lambda remote debugging in the AWS Toolkit: the AWS explorer or the Application Builder explorer. From the AWS explorer, you can access Lambda remote debugging through your AWS Lambda nodes. From the Application Builder explorer, you can access Lambda remote debugging through your local AWS SAM projects.

Accessing Lambda remote debugging from the AWS explorer

1. From VS Code, open the AWS Toolkit extension.
2. From the AWS Toolkit, expand the AWS explorer.
3. From the explorer, expand the **Lambda** node.
4. Navigate to the function you want to debug, then choose the **Invoke remotely** icon from the context menu to open the **Remote invoke configuration** screen.

Accessing Lambda remote debugging from the Application Builder explorer.

1. From VS Code, open the AWS Toolkit extension.
2. From the AWS Toolkit, expand the application builder explorer.
3. From the explorer expand the AWS SAM project that contains the Lambda project you want to debug.
4. Expand the deployed Lambda function that you want to debug.
5. Navigate to the function remote, then choose the **Invoke remotely** icon from the context menu to open the **Remote invoke configuration** screen.

Working with Lambda remote debugging

The following sections describe how to work with Lambda remote debugging in the AWS Toolkit for Visual Studio Code.

> **Note**
>
> Lambda functions have a 5-layer limit and a 250MB combined limit for function code and all attached layers. Lambda remote debugging requires at least 1 free layer to run.

Setting up a debugging session

Before you begin, configure your debugging session by completing the following procedure.

1. Open the **Remote invoke configuration** menu by completing the *Accessing Lambda remote debugging from the AWS explorer* or the *Accessing Lambda remote debugging from the Application Builder explorer* procedure, located in the previous section.
2. From the **Remote invoke configuration** menu, select the **Remote Debugging** check box to display the remote debugging properties.
3. Specify the **Local Root Path** to your local handler file.

   > **Note**
   >
   > The local root path is the location of your source code that matches the deployed Lambda function. If you're working from a deployed function in the Application Builder explorer, your local root path is automatically detected.
   > If you don't have the source code stored locally, choose the **Download remote code** button to retrieve your Lambda function source code. This will open your `handler file` in the VS Code editor.

4. From the **Payload** section, specify where your test-event data is obtained.

Setting breakpoints and debugging

Set breakpoints and begin debugging by completing the following procedure.

1. From your `handler file` in the VS Code editor, click in the gutter-margin to set breakpoints at the line numbers where you want to pause debugging.

2. When you're satisfied with the breakpoints, return to the **Remote invoke configuration** menu to verify that your settings are configured correctly, then choose the **Remote invoke** button to start debugging.
3. The AWS Toolkit updates your Lambda function with debugging capabilities, establishes a secure tunnel for the debugging session, invokes your function with the specified payload, then pauses the process when it reaches a breakpoint.
4. At a breakpoint pause, use the **RUN AND DEBUG** pane to view your **VARIABLES**, **CALL STACK**, and **BREAKPOINTS**.

Updating and testing your function

To modify your code and test changes with a quick deployment, complete the following procedure.

1. With your debugging session active, make changes to your `handler file` in the VS Code editor.
2. Save your changes (**`Command+S on macOS`**,**`Ctrl+S on Windows`**)
3. When prompted, confirm that you wish to proceed to deploy your changes. The AWS Toolkit will update your Lambda function with the modified code.
4. Continue debugging and testing your changes by setting new breakpoints and selecting the **Remote invoke** button again.

> **Note**
>
> Alternatively, you can deselect the **Attach debugger** option in the VS Code debugging controls and choose the **Remote invoke** button to run your function without debugging.

Ending a debugging session

Each of the following options ends your remote debugging session and removes the debug layer from your project.

- Choosing the **Remove Debug Setup** option from the **Remote invoke configuration** screen.
- Choosing the **disconnect** icon from the VS Code debugging controls.
- Closing the `handler file` in the VS Code editor.

Note

Take note of the following:

- The Lambda debug layer is automatically removed after 60 seconds of inactivity. The count begins when your last invoke is complete.
- If you made code changes to your infrastructure-as-code (IaC) managed (AWS SAM, AWS CDK, Terraform) functions during the debugging process, save them to your local project and consider updating your source-control repository. Unsaved changes are overwritten when your IaC function redeploys.
- If you made temporary changes for debugging purposes only, you may want to redeploy your function from your source control to ensure it matches your production code.

Debugging TypeScript Lambda functions with source maps

The following sections describe how to debug your TypeScript Lambda functions with source maps.

Prerequisites

To debug your TypeScript Lambda functions, the following prerequisites must be met.

- Your TypeScript must be compiled with the source map option enabled. For additional information, see the JavaScript source map support topic in the VS Code documentation.
- In-line source maps are not supported. You must use a separate `.js.map` file to store the source map.

Configuration

To configure Lambda remote debugging for TypeScript Lambda functions in the AWS Toolkit, complete the following steps.

1. From the AWS Toolkit, expand the AWS explorer.
2. From the explorer, expand the **Lambda** node.
3. Navigate to the function you want to configure for TypeScript, then choose the **Invoke remotely** icon from the context menu to open the **Remote invoke configuration** screen.
4. Enable remote debugging by select the **Remote debugging** check box.

5. Configure your **Local Root Path** by pointing to the directory containing your `TypeScript handler file`.

   > **Note**
   >
   > The `TypeScript handler file` is where you set your debugging breakpoints.

6. Expand **Remote debug additional configuration** settings.
7. Enable source mapping by selecting the **Source map** check box.
8. Set the **Out files** field to the local directory of your Lambda function copy.

   Example

   If `app.js` and `app.map` are in `.aws-sam/build/HelloWorldFunction`, then make the **Out files** location `/Users/`**`user/project`**`/aws-sam/build/`**`HelloWorldFunction`**`/*`.

   > **Note**
   >
   > The **Out file** path should be an absolute path.
   > For AWS SAM and AWS CDK projects, the AWS Toolkit supports automatic source map detection. If the **Out files** field is left empty for these projects, the toolkit will automatically attempt to detect the source map location.

9. When you're satisfied with the settings, choose the **Remote invoke** button to begin debugging your TypeScript function.

Troubleshooting and advanced use cases

If your debug session fails, start the troubleshooting process by completing these steps.

1. Update the AWS Toolkit to the latest version.
2. Refresh the web view by closing the **Remote invoke configuration** web view and reopening it.
3. Restart VS Code by closing it completely and reopening it.
4. Open the VS Code Command Palette and enter the command **`AWS: Reset Lambda Remote Debugging Snapshot`**, select it when it populates in the results to reset your Lambda remote debugging snapshot.

5. If you're not able to troubleshoot the problem, submit an issue to AWS Toolkit for Visual Studio Code GitHub Issues.

Advanced use case: code-signing configuration

Remote debugging requires attaching a debug layer to your Lambda function. If your function has code-signing configuration enabled and enforced, the AWS Toolkit can't automatically attach the debug layer to your function.

There are two options to resolve the code-signing configuration issue.

- Temporarily remove code signing.
- Use a signed debug layer.

Temporarily removing code signing

Update the code-signing configuration by setting `UntrustedArtifactOnDeployment : Warn`, then re-enable it back to `Enforced` after the debugging process is complete.

For more information, see the UpdateCodeSigningConfig reference in the *AWS Lambda API Reference*.

Using a signed debug layer

1. From Lambda remote debugging in the AWS Toolkit, expand the **Remote debug additional configuration** section.
2. From the **Remote debug additional configuration** section, copy your Region layer ARN from the **Layer override** field.
3. From the AWS CLI, use the following command to download the layer version `aws lambda get-layer-version-by-arn --arn` **`layer-arn`**, replacing *layer-arn* with your layer ARN. For detailed instructions on how to download the signed debug layer, see the get-layer-version-by-arn reference in the *AWS CLI Command Reference*.
4. Sign the layer with your code-signing configuration and publish it to your account. For signing and publishing guidance, see the Set up code signing for your AWS SAM application topic in the *AWS Serverless Application Model Developer Guide*.
5. After the layer has been signed and published to your account, return to the **Remote debug additional configuration** section of Lambda remote debugging, then enter the new layer ARN

into the **Layer override** field. When the process is complete, Lambda remote debugging uses your signed layer instead of the default layer.

Advanced use case: debugging functions with SnapStart or provisioned concurrency

For Lambda functions configured with SnapStart or provisioned concurrency, publishing a new version takes significantly more time. To speed up your debugging workflow, you can configure Lambda remote debugging to update only the $LATEST version of your function instead of publishing a new version.

1. From the **Remote invoke configuration** screen, expand the **Remote debug additional configuration** settings.
2. Deselect the **Publish version** option.
3. The AWS Toolkit will now only update your function's $LATEST version and debug using it.

> **Note**
>
> As a side effect of debugging with the $LATEST version, you should avoid other traffic that may invoke your $LATEST version to ensure an undisturbed debugging environment.

Supported regions

The following error occurs when a region doesn't support remote debugging.

```
Region ${region} doesn't support remote debugging yet
```

The following is a list of supported regions.

- ap-east-1
- ap-northeast-1
- ap-northeast-2
- ap-south-1
- ap-southeast-1
- ap-southeast-2

- ca-central-1
- eu-central-1
- eu-north-1
- eu-west-1
- eu-west-2
- eu-west-3
- me-central-1
- me-south-1
- sa-east-1
- us-east-1
- us-east-2
- us-west-1
- us-west-2

Lambda RequestEntityTooLargeException

Lambda functions have a 5-layer limit and a 250MB combined limit for function code and all attached layers. The remote debugging layer is approximately 40MB, which may cause your function to exceed this limit if you have a large function package or multiple layers. For additional details, see the Lambda: InvalidParameterValueException or RequestEntityTooLargeException topic section in the *AWS Lambda Developer Guide*.

The following list describes ways to troubleshoot and correct this error.

- **Reduce function size**: Optimize your function code and remove unnecessary dependencies.
- **Remove unused layers**: Temporarily remove non-essential layers during debugging.
- **Use external dependencies**: Move large dependencies to external storage, such as Amazon S3, and load them at runtime.

Troubleshooting Java debugging

To debug a Java Lambda function, you must have the same Java version installed locally that matches your Lambda function's runtime version.

For example, when debugging a Java 25 function, you must have Java 25 installed in your local environment where the AWS Toolkit is running. If you attempt to debug a Java 25 function with Java 21 or an earlier version installed locally, remote debugging will not be able to stop at the breakpoints you set.

Ensure your local Java version matches your Lambda function's runtime version before starting a debugging session.

IoT secure tunneling quota exceeded

The following is an example of the *tunnel quota exceeded error* that occurs when you've reached the daily limit for AWS IoT secure tunneling connections in Lambda remote debugging.

```
Error creating/reusing tunnel: LimitExceededException: Exceeded quota of Lambda
 debugging tunnels
```

AWS IoT Secure Tunneling connection have the following quotas:

- Free-tier IoT secure tunneling is allotted 10 connections per day.
- Each tunnel supports one VS Code instance for up to 12 hours.
- The quota applies per AWS account, per day.

If you encounter the AWS IoT secure tunneling error, wait for the daily quota reset or contact AWS support to request a quota-limit increase. For AWS support contact info, see the AWS support contact portal. For detailed information about AWS IoT secure tunneling, see the AWS IoT secure tunneling topic in the *AWS IoT Developer Guide*.

Amazon Redshift in the Toolkit for VS Code

Amazon Redshift is a fully managed, petabyte-scale data warehouse service in the cloud. For detailed information about the Amazon Redshift service, see the Amazon Redshift User Guides table of contents.

The following topics desribe how to work with Amazon Redshift from the AWS Toolkit for Visual Studio Code.

Topics

- Working with Amazon Redshift from the Toolkit for VS Code

Working with Amazon Redshift from the Toolkit for VS Code

The following sections describe how to get started working with Amazon Redshift from the AWS Toolkit for Visual Studio Code.

For detailed information about the Amazon Redshift service, see the Amazon Redshift User Guide topics.

Getting started

Before you start working with Amazon Redshift from the AWS Toolkit for Visual Studio Code, the following requirements must be met.

1. You're connected to your AWS account(s) from the Toolkit. For additional information about connecting to your AWS account from the Toolkit, see the Connecting to AWS topic in this User Guide.
2. You've created a provisioned or serverless data warehouse.

If you've not yet created an Amazon Redshift Serverless or an Amazon Redshift provisioned cluster, the following procedures describe how to create a data warehouse with a sample dataset, from the AWS Console.

Creating a provisioned data warehouse

For additional details on creating an Amazon Redshift provisioned cluster data warehouse, see the Create a sample Amazon Redshift cluster topic in the *Amazon Redshift getting started* User Guide.

1. From your preferred internet browser, sign into the AWS Management Console and open the Amazon Redshift console at https://console.aws.amazon.com/redshift/.
2. From the Amazon Redshift console, choose **Provisioned Clusters dashboard**.
3. From the **Provisioned Clusters dashboard**, choose the **Create cluster** button to open the **Create cluster** pane.
4. Complete the required fields in the **Cluster configuration** section.
5. In the **Sample data** section, select the **Load sample data** box to load the sample dataset **`Tickit`** into the default database **`Dev`** with the **`public`** schema.
6. In the **Database configurations** section, input values for the **Admin user name** and **Admin user password** fields.
7. Choose **Create cluster** to create your provisioned data warehouse.

Creating a serverless data warehouse

For additional details on creating an Amazon Redshift Serverless data warehouse, see the Creating a data warehouse with Amazon Redshift Serverless section in the *Amazon Redshift getting started* User Guide.

1. From your preferred internet browser, sign into the AWS Management Console and open the Amazon Redshift console at https://console.aws.amazon.com/redshift/.
2. From the Amazon Redshift console, choose the **Try Amazon Redshift Serverless** button to open the **Get started with Amazon Redshift Serverless** pane.
3. In the **Configurations** section, choose the **Use default settings** radial.
4. At the bottom of the **Get started with Amazon Redshift Serverless** pane, choose **Save configuration** to create a serverless data warehouse with default workgroup, namespace, credential, and encryption settings.

Connecting to a data warehouse from the Toolkit

There are 3 methods to connect to a database from the Toolkit:

- **Database user name and password**
- **AWS Secrets Manager**
- **Temporary credentials**

To connect to a database located on an existing provisioned cluster or serverless data warehouse from the Toolkit, complete the following steps.

> **Important**
>
> If you've completed the steps in the *Prerequisites* section of this User Guide topic and your data warehouse is not visible in the Toolkit explorer, make sure that you're working from the correct AWS region in the explorer.

Connecting to your data warehouse with the Database user name and password method

1. From the Toolkit explorer, expand the AWS Region where your data warehouse exists.

2. Expand **Redshift** and choose your data warehouse to open the **Select a Connection Type** dialog in VS Code.
3. From the **Select a Connection Type** dialog, choose **Database user name and password** and provide the information required by each of the prompts.
4. Your available databases, tables, and schemas are visible in the Toolkit explorer when the Toolkit connects to your data warehouse and the procedure is complete.

Connecting to your data warehouse with AWS Secrets Manager

> **Note**
>
> This procedure requires an AWS secrets manager database secret to complete. For instructions on how to set up a database secret, see the Create an AWS Secrets Manager database secret in the *AWS Secrets Manager* User Guide.

1. From the Toolkit explorer, expand the AWS Region where your data warehouse exists.
2. Expand **Redshift** and choose your data warehouse to open the **Select a Connection Type** dialog in VS Code.
3. From the **Select a Connection Type** dialog, choose **Secrets Manager** and provide the information required by each of the prompts.
4. Your available databases, tables, and schemas are visible in the Toolkit explorer when the Toolkit connects to your data warehouse and the procedure is complete.

Connecting to your data warehouse with Temporary credentials

1. From the Toolkit explorer, expand the AWS region where your data warehouse exists.
2. Expand **Redshift** and choose your data warehouse to open the **Select a Connection Type** dialog in VS Code.
3. From the **Select a Connection Type** dialog, choose **Temporary credentials** and provide the information required by each of the prompts.
4. Your available databases, tables, and schemas are visible in the Toolkit explorer when the Toolkit connects to your data warehouse and the procedure is complete.

Editing the connection to your data warehouse

You can edit the connection to your data warehouse to change which database to connect to.

1. From the Toolkit explorer, expand the AWS Region where your data warehouse exists.
2. Expand **Redshift**, right-click the data warehouse you are connected to, choose **Edit connection**, and provide the name of the database you want to connect to.
3. Your available databases, tables, and schemas are visible in the Toolkit explorer when the Toolkit connects to your data warehouse and the procedure is complete.

Deleting the connection to your data warehouse

1. From the Toolkit explorer, expand the AWS Region where your data warehouse exists.
2. Expand **Redshift**, right-click the data warehouse with the connection you want to delete, and choose **Delete connection**. Doing so removes the available databases, tables, and schemas from the Toolkit explorer.
3. To reconnect to your data warehouse, choose **Click to connect** and provide the information required by each of the prompts. By default, reconnecting uses the previous method of authentication to connect to the data warehouse. To use a different method, choose the back arrow in the dialog until you reach the authentication prompt.

Running SQL Statements

The following procedures describe how to create and run SQL statements in your database from the AWS Toolkit for Visual Studio Code.

> **Note**
>
> To complete the steps in each of the following procedures, you must first complete the section *Connecting to a data warehouse from the Toolkit*, located in this User Guide topic.

1. From the Toolkit explorer, expand **Redshift**, then expand that data warehouse that contains the database you want to query.
2. Choose **Create-Notebook** to specify a file name and location to store your notebook locally, then choose **OK** to open the notebook in your VS Code editor.

3. From the VS Code editor, input the SQL statements you want stored in this notebook.
4. Choose the **Run All** button to run the SQL statements you entered.
5. The output for your SQL statements is displayed below the statements that you entered.

Adding Markdown to a notebook

1. From your notebook in the VS Code editor, choose the **Markdown** button to add a Markdown cell to your notebook.
2. Input your Markdown into the provided cell.
3. The Markdown cell can be edited using the editor tools located in the upper-right corner of the Markdown cell.

Adding code to a notebook

1. From your notebook in the VS Code editor, choose the **Code** button to add a Code cell to your notebook.
2. Input your code into the provided cell.
3. You can choose to run your code above or below the Code cell by selecting the appropriate button from the cell editor tools, located in the upper-right corner of the Code cell.

Working with Amazon S3

Amazon Simple Storage Service (Amazon S3) is a scalable data-storage service. The AWS Toolkit for Visual Studio Code allows you to manage your Amazon S3 objects and resources directly from VS Code.

For detailed information about the Amazon S3 service, see the Amazon S3 User Guide.

The following topics describe how to work with Amazon S3 objects and resources from the AWS Toolkit for Visual Studio Code.

Topics

Working with Amazon S3 resources

You can use Amazon S3 from the AWS Toolkit for Visual Studio Code to view, manage, and edit your Amazon S3 buckets and other resources.

The following sections desribe how to work with Amazon S3 resources from the AWS Toolkit for Visual Studio Code. For information about working with Amazon S3 objects, such as folders and files, from the AWS Toolkit for Visual Studio Code, see the Working with S3 objects topic in this User Guide.

Creating an Amazon S3 bucket

1. From the Toolkit explorer, open the context (right-click) menu for the **S3** service, and choose **Create Bucket...**. Alternatively, choose the **Create Bucket** icon to open the **Create Bucket** dialog box.
2. In the **Bucket Name** field, enter a valid name for the bucket.

   Press **Enter** to create the bucket and close the dialog box. Your new bucket is then displayed under the S3 service in the toolkit.

   > **Note**
   >
   > Since Amazon S3 allows your bucket to be used as a URL that can be accessed publicly, the bucket name that you choose must be globally unique. If another account already created a bucket with the name that you want to use, you must use a different name. If you can't create a new bucket, check the **AWS Toolkit Logs** in the **Output** tab. If you attempt to use an invalid bucket name, a `BucketAlreadyExists` error occurs. For more information, see Bucket restrictions and limitations in the **Amazon Simple Storage Service User Guide**.

Adding a folder to an Amazon S3 bucket

You can organize the contents of an S3 bucket by grouping your objects into folders. You can also create folders within folders.

1. From the Toolkit explorer, expand the **S3** service to view a list of your S3 resources.
2. Choose the **Create Folder icon** to open the **Create Folder** dialog box. Or, open the context (right-click) menu for a bucket or folder, and then choose **Create Folder**.

3. Enter a value into the **Folder Name** field and press **Enter** to create the folder and close the dialog box. Your new folder is displayed under the corresponding S3 resource in the toolkit menu.

Deleting an Amazon S3 bucket

When you delete an S3 bucket, you also delete the folders and objects that it contains. So, when you attempt to delete a bucket, you're asked to confirm that you want to delete it.

1. From the toolkit main menu, expand the **Amazon S3** service to view a list of your S3 resources.
2. Open the context (right-click) menu for a bucket or folder, then choose **Delete S3 Bucket**.
3. When you're prompted, enter the bucket's name into the text field, and then press **Enter** to delete the bucket and close the confirmation prompt.

> **Note**
>
> If your bucket contains objects, it's emptied before it's deleted. If you attempt to delete a large number of resources or objects at one time, it can take some time for them to be deleted. After they're deleted, you receive a notification that says that they're successfully deleted.

Working with Amazon S3 objects

Your files, folders, and any other data that's stored in an S3 resource bucket are known as S3 objects.

The following sections describe how to work with Amazon S3 objects from the AWS Toolkit for Visual Studio Code. For details on working with Amazon S3 resources, such as S3 buckets, from the AWS Toolkit for Visual Studio Code, see the Working with S3 resources topic in this User Guide.

Object pagination

If you're working with a large number of Amazon S3 objects and folders, pagination allows you to specify the number of items that you want to display on a page.

1. Navigate to the VS Code **Activity Bar** and choose **Extensions**.

2. From the AWS Toolkit extension, choose the settings icon, and then choose **Extension Settings**.
3. On the **Settings** page, scroll down to the **AWS > S3: Max Items Per Page** setting.
4. Change the default value to the number of S3 items that you want to be displayed before "load more" is displayed.

> **Note**
>
> Valid values include any number between 3 and 1000. This setting applies only to the number of objects or folders displayed at one time. All the buckets you created are displayed at once. By default, you can create up to 100 buckets in each of your AWS accounts.

5. Close the **Settings** page to confirm your changes.

You can also update the settings in a JSON-formatted file by choosing the **Open Settings (JSON)** icon in the upper right of the **Settings** page.

Uploading and downloading Amazon S3 objects

You can upload locally-stored files to your Amazon S3 buckets or download remote Amazon S3 objects to your local system, from the AWS Toolkit for Visual Studio Code.

Upload a file using the Toolkit

1. From the Toolkit explorer, expand the **Amazon S3** service to view a list of your S3 resources.
2. Choose the **Upload File icon** that's located next to a bucket or folder to open the **Upload File dialog**. Or you can open the context (right-click) menu and choose **Upload File**.

> **Note**
>
> To upload a file to the object's parent folder or resource, open the context (right-click) menu for any S3 object and choose **Upload to Parent**.

3. Use your system's file manager to select a file, then choose **Upload File** to close the dialog and upload the file.

Upload a file using the Command Palette

You can use the Toolkit interface or the **Command Palette** to upload a file to a bucket.

1. To select a file for upload, choose that file's tab in VS Code.
2. Press **Ctrl+Shift+P** to display the **Command Palette**.
3. In the **Command Palette**, enter the phrase `upload file` to display a list of recommended commands.
4. Choose the **AWS: Upload File** command to open the **AWS: Upload File** dialog.
5. When prompted, choose the file you want to upload, then choose the bucket you want to upload that file to.
6. Confirm your upload to close the dialog and begin the upload process. When the upload is complete, the object displays in the toolkit menu with metadata that includes the object size, last modification date, and path.

Downloading an Amazon S3 object

1. From the Toolkit explorer, expand the **S3** service.
2. From a bucket or folder, open the context (right-click) menu for an object that you want to download. Then, choose **Download As** to open the Download As dialog box. Or, alternatively, choose the **Download As** icon near the object.
3. Using your system's file manager, choose a destination folder, enter a file name, and then choose **Download** to close the dialog and start the download.

Editing remote objects

You can use the AWS Toolkit for Visual Studio Code to edit your Amazon S3 objects that are stored in your remote Amazon S3 resources.

1. From the Toolkit explorer, expand the **S3** service.
2. Expand the S3 resource that contains the file that you want to edit.
3. To edit the file, choose the **pencil icon (Edit File)**.
4. To edit a file that's open in read-only mode, view the file in the VS Code editor, then choose the **pencil icon** located on the upper-right hand corner of the UI.

Note

- If you restart or exit VS Code, your IDE disconnects from your S3 resources. If any remote S3 files are being edited when you disconnect, the edit stops. You must restart VS Code and reopen the edit tab to resume the edit.
- The **Edit File** button is in the upper-right hand corner of the UI. It's only visible when you're actively viewing a read-only file in the VS Code editor.
- Non-text files can't be opened in a read-only mode. They always open in edit-mode.
- You can't toggle back to read-only mode from edit-only mode, only the other way around.

Copying the path of an Amazon S3 object

The following procedure describes how to copy the path of an Amazon S3 object from the AWS Toolkit for Visual Studio Code.

1. From the Toolkit explorer, expand the **S3** service.
2. Expand the resource bucket that contains the object you want to copy the path for.
3. Open the context (right-click) menu for the object that you want to copy the path for, then choose **Copy Path** to copy the object path to your local clipboard.

Generating a presigned URL for an Amazon S3 object

You can share private Amazon S3 objects with others by granting time-limited permissions for downloads through the presigned URL feature. For more information, see Sharing an object with a presigned URL.

1. From the Toolkit explorer, expand the **S3** service.
2. From a bucket or folder, open the context (right-click) menu for an object that you want to share. Then, choose **Generate Presigned URL** to open the **Command palette**.
3. From the **Command Palette**, enter the number of minutes that the URL can be used to access your object. Then, choose **Enter** to confirm and close the dialog.
4. After the presigned URL is generated, the VS Code **Status Bar** displays the presigned URL for the object that has been copied to your local **clipboard**.

Deleting an Amazon S3 object

If an object is in a non-versioned bucket, you can permanently delete it. For buckets that have versioning enabled, a delete request doesn't permanently delete that object. Instead, Amazon S3 inserts a delete marker in the bucket. For more information, see Deleting object versions.

1. From the Toolkit explorer, expand the **S3** service to view a list of your S3 resources.
2. Open the context (right-click) menu for an object you want to delete, then choose **Delete** to open the confirmation dialog.
3. Choose **Delete. . .** to confirm that you want to delete the S3 object. Then, close the dialog.

Amazon SageMaker Unified Studio for VS Code

As a part of the next generation of Amazon SageMaker, the Amazon SageMaker Unified Studio is a unified development experience that brings together AWS data, analytics, artificial intelligence (AI), and machine learning (ML) services. It provides a place to build, deploy, execute, and monitor workflows from a single interface. For more information about setting up the Amazon SageMaker Unifed Studio integration with the VS Code IDE, see Setting up the Amazon SageMaker Unified Studio integration in VS Code in the Amazon SageMaker Unified Studio User Guide.

Working with serverless applications

The AWS Toolkit for Visual Studio Code provides support for AWS Serverless Application. The following topics describe how to get started creating and working with AWS Serverless Application Model (AWS SAM) applications, from the AWS Toolkit for Visual Studio Code.

Topics

- Getting Started with serverless applications
- Working with AWS Serverless Land
- Running and debugging Lambda functions directly from code
- Running and debugging local Amazon API Gateway resources
- Configuration options for debugging serverless applications
- Troubleshooting serverless applications

Getting Started with serverless applications

The following sections describe how to get started creating an AWS Serverless Application from the AWS Toolkit for Visual Studio Code, using AWS Serverless Application Model (AWS SAM) and CloudFormation stacks.

Prerequisites

Before you can create or work with an AWS Serverless Application, the following prerequisites must be completed.

> **Note**
>
> The following operations may require you to exit or restart VS Code before the changes are complete.

- Install the AWS SAM command line interface (CLI). For additional information and instructions on how to install the AWS SAM CLI, see the Installing the AWS SAM CLI topic in this *AWS Serverless Application Model User Guide*.
- From your AWS config file, identify your default AWS Region. For more information on your config file, see the Configuration and credential file settings topic in the *AWS Command Line Interface User Guide*.
- Install your language SDK and configure your toolchain. For additional information on how to configure your toolchain from the AWS Toolkit for Visual Studio Code see the configure your toolchain topic in this User Guide.
- Install the YAML language support extension from the VS Code marketplace. This is required for the CodeLens feature of AWS SAM template files are accessible. For additional information about CodeLens, see the CodeLens section in the VS Code documentation

IAM permissions for serverless applications

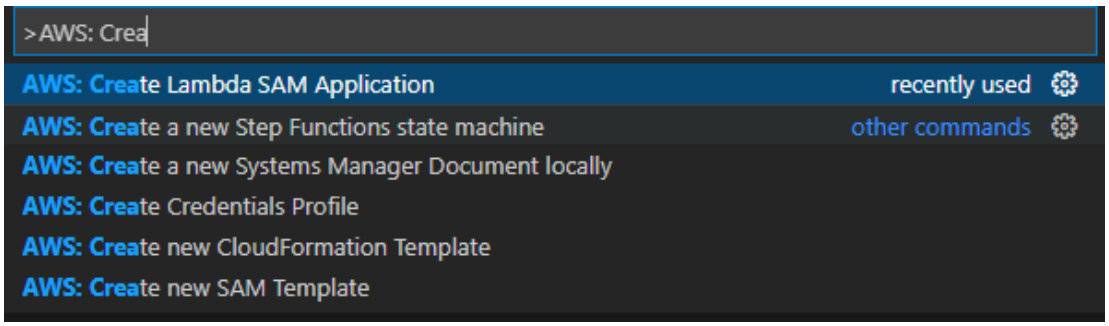
In the Toolkit for VS Code you must have a credentials profile that contains the AWS Identity and Access Management (IAM) permissions necessary to deploy and run serverless applications. You must have appropriate read/write access to the following services: CloudFormation, IAM, Lambda, Amazon API Gateway, Amazon Simple Storage Service (Amazon S3), and Amazon Elastic Container Registry (Amazon ECR).

For additional information about setting up authentication required to deploy and run serverless applications, see the Managing resource access and permissions in the *AWS Serverless Application Model Developer Guide*. For information on how to set up your credentials, see the AWS IAM credentials in this User Guide.

Creating a new serverless application (local)

This procedure shows how to create a serverless application with the Toolkit for VS Code by using AWS SAM. The output of this procedure is a local directory on your development host containing a sample serverless application, which you can build, locally test, modify, and deploy to the AWS Cloud.

1. To open the **Command Palette**, choose **View**, **Command Palette**, and then enter **AWS**.
2. Choose **AWS Toolkit Create Lambda SAM Application**.



> **Note**
>
> If the AWS SAM CLI isn't installed, you get an error in the lower-right corner of the VS Code editor. If this happens, verify that you've met all the assumptions and prerequisites.

3. Choose the runtime for your AWS SAM application.

> **Note**
>
> If you select one of the runtimes with "(Image)", your application is package type `Image`. If you select one of the runtimes without "(Image)", your application is type `Zip`. For more information about the difference between `Image` and `Zip` package types, see Lambda deployment packages in the *AWS Lambda Developer Guide*.

4. Depending on the runtime you select, you may be asked to select a dependency manager and a runtime architecture for your SAM application.

   Dependency Manager

   Choose between **Gradle** or **Maven**.

   > **Note**
   >
   > This choice of build automation tools is available only for Java runtimes.

   Architecture

   Choose between **x86_64** or **arm64**.

   The option to run your serverless application in an ARM64-based emulated environment instead of the default x86_64-based environment is available for the following runtimes:

   - nodejs12.x (ZIP and image)
   - nodejs14.x (ZIP and image)
   - python3.8 (ZIP and image)
   - python3.9 (ZIP and image)
   - python3.10 (ZIP and image)
   - python3.11 (ZIP and image)
   - python3.12 (ZIP and image)
   - java8.al2 with Gradle (ZIP and image)
   - java8.al2 with Maven (ZIP only)
   - java11 with Gradle (ZIP and image)
   - java11 with Maven (ZIP only)

   > **Important**
   >
   > You must install AWS CLI version 1.33.0 or later to allow applications to run in ARM64-based environments. For more information, see Prerequisites.

5. Choose a location for your new project. You can use an existing workspace folder if one is open, **Select a different folder** that already exists, or create a new folder and select it. For this example, choose **There are no workspace folders open** to create a folder named `MY-SAM-APP`.
6. Enter a name for your new project. For this example, use `my-sam-app-nodejs`. After you press **Enter**, the Toolkit for VS Code takes a few moments to create the project.

When the project is created, your application is added to your current workspace. You should see it listed in the **Explorer** window.

Opening a serverless application (local)

To open a serverless application on your local development host, open the folder that contains the application's template file.

1. From the **File**, choose **Open Folder...**.
2. In the **Open Folder** dialog box, navigate to the serverless application folder that you want to open.
3. Choose the **Select Folder** button.

When you open an application's folder, it is added to the **Explorer** window.

Running and debugging a serverless application from template (local)

You can use the Toolkit for VS Code to configure how to debug serverless applications and run them locally in your development environment.

You start to configure debug behavior by using the VS Code CodeLens feature to identify an eligible Lambda function. CodeLens enables content-aware interactions with your source code. For information about ensuring that you can access the CodeLens feature, review the Prerequisites section from earlier in this topic.

> **Note**
>
> In this example, you debug an application that uses JavaScript. However, you can use Toolkit for VS Code debugging features with the following languages and runtimes:
>
> - C# – .NET Core 2.1, 3.1; .NET 5.0

- JavaScript/TypeScript – Node.js 12.*x*, 14.*x*
- Python – 3.6, 3.7, 3.8, 3.9, 3.10, 3.11, 3.12
- Java – 8, 8.al2, 11
- Go – 1.x

Your language choice also affects how CodeLens detects eligible Lambda handlers. For more information, see Running and debugging Lambda functions directly from code.

In this procedure, you use the example application created in the Creating a new serverless application (local) section earlier in this topic.

1. To view your application files in VS Code's File Explorer, choose **View**, **Explorer**.
2. From the application folder (for example, *my-sample-app*), open the `template.yaml` file.

   > **Note**
   >
   > If you use a template with a name that's different from `template.yaml`, the CodeLens indicator isn't automatically available in the YAML file. This means that you must manually add a debug configuration.

3. In the editor for `template.yaml`, go to the `Resources` section of the template that defines serverless resources. In this case, this is the `HelloWorldFunction` resource of type `AWS::Serverless::Function`.

   In the CodeLens indicator for this resource, choose **Add Debug Configuration**.
   Using the CodeLens indicator in the `template.yaml` file to add a debug configuration.
4. In the **Command Palette**, select the runtime in which your AWS SAM application will run.
5. In the editor for the `launch.json` file, edit or confirm values for the following configuration properties:

   - `"name"` – Enter a reader-friendly name to appear in the **Configuration** drop-down field in the **Run** view.
   - `"target"` – Ensure that the value is `"template"` so that the AWS SAM template is the entry point for the debug session.
   - `"templatePath"` – Enter a relative or absolute path for the `template.yaml` file.

- `"logicalId"` – Ensure that the name matches the one specified in the **Resources** section of the AWS SAM template. In this case, it's the `HelloWorldFunction` of type `AWS::Serverless::Function`.

Configuring the `launch.json` file for template-based debugging.

For more information about these and other entries in the `launch.json` file, see Configuration options for debugging serverless applications.

6. If you're satisfied with your debug configuration, save `launch.json`. Then, to start debugging, choose the green "play" button in the **RUN** view.

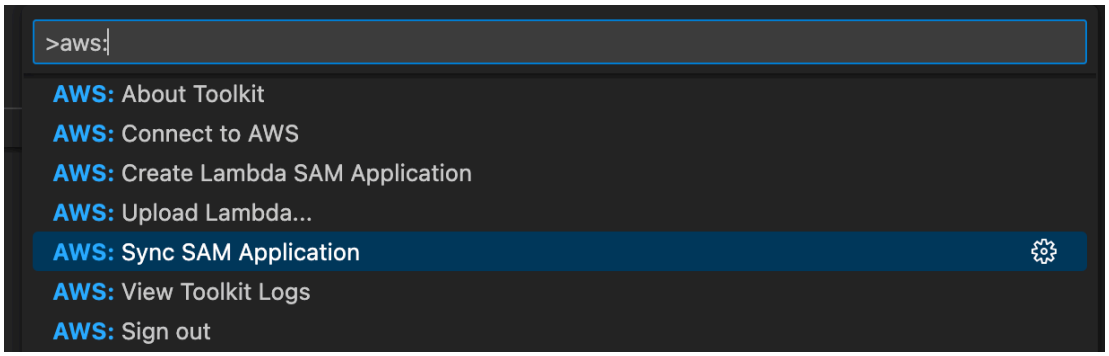
   When the debugging sessions starts, the **DEBUG CONSOLE** panel shows debugging output and displays any values returned by the Lambda function. (When debugging AWS SAM applications, the **AWS Toolkit** is selected as the **Output** channel in the **Output** panel.)

Syncing AWS SAM applications

The AWS Toolkit for Visual Studio Code runs the AWS SAM CLI command `sam sync` to deploy your serverless applications to the AWS Cloud. For additional information about AWS SAM sync, see the AWS SAM CLI command reference topic in the *AWS Serverless Application Model Developer Guide*

The following procedure describes how to deploy your serverless applications to the AWS Cloud with `sam sync` from the Toolkit for VS Code.

1. From the main menu in VS Code, open the **Command Palette** by expanding **View** and choosing **Command Palette**.
2. From the **Command Palette** search for **AWS** and choose **Sync SAM Application** to start setting up your sync.



3. Choose the AWS Region to sync your serverless application to.

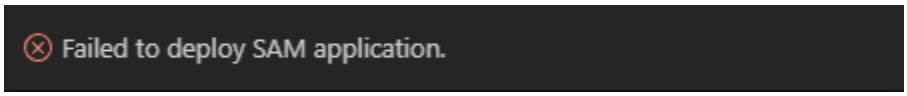
4. Choose the `template.yaml` file to use for the deployment.
5. Select an existing Amazon S3 bucket or enter a new Amazon S3 bucket name to deploy your application to.

> **Important**
>
> Your Amazon S3 bucket must meet the following requirements:
>
> - The bucket must be in the Region that you're syncing to.
> - The Amazon S3 bucket name must be globally unique across all existing bucket names in Amazon S3.

6. If your serverless application includes a function with package type `Image`, enter the name of an Amazon ECR repository that this deployment can use. The repository must be in the Region that you're deploying to.
7. Select a deployment stack from the list of your previous deployments, or create a new deployment stack be entering a new stack name. Then, proceed to begin the sync process.

> **Note**
>
> Stacks used in previous deployments are recalled per workspace and region.

8. During the syncing process, the status of your deployment is captured in the **Terminal** tab of VS Code. Verify that your sync was successful from the terminal tab, if an error occurs you receive a notification.



> **Note**
>
> For additional details about your sync, the AWS Toolkit for Visual Studio Code logs are accessible from the **Command Palette**.

To access the your AWS Toolkit for Visual Studio Code logs from the Command Palette, expand **View**, choose **Command Palette**, then search for **`AWS: View AWS Toolkits Logs`**, and select it when it populates in the list.

When the deployment is complete, you see your application listed in the **AWS Explorer**. For more information about how to invoke the Lambda function created as part of the application, see the Working with AWS Lambda Functions topic in this User Guide.

Deleting a serverless application from the AWS Cloud

Deleting a serverless application involves deleting the CloudFormation stack that you previously deployed to the AWS Cloud. Note that this procedure does not delete your application directory from your local host.

1. Open the AWS Explorer.
2. In the **AWS Toolkit Explorer** window, expand the Region containing the deployed application that you want to delete, and then expand **CloudFormation**.
3. Open the context (right-click) menu for the name of the CloudFormation stack that corresponds to the serverless application that you want to delete, and then choose **Delete CloudFormation Stack**.
4. To confirm that you want to delete the selected stack, choose **Yes**.

If the stack deletion succeeds, the Toolkit for VS Code removes the stack name from the CloudFormation list in **AWS Explorer**.

Working with AWS Serverless Land

AWS Serverless Land in the AWS Toolkit for Visual Studio Code is a collection of features that assists you with building event-driven architectures. The following topic sections describe how to work with Serverless Land in the AWS Toolkit. For detailed information about Serverless Land, see the Serverless Land web application.

Accessing Serverless Land

There are 3 main entry points to access Serverless Land in the AWS Toolkit:

- The VS Code Command Palette
- The AWS Toolkit Explorer
- The AWS Toolkit **Application Builder** explorer

Opening Serverless Land from the VS Code Command Palette

To open Serverless Land from the VS Code Command Palette, complete the following steps.

1. From VS Code, open the Command Palette by pressing **`option+shift+p`** (Mac) or **`control+shift+p`** (Windows).
2. From the VS Code Command Palette, enter **`AWS Create application with Serverless template`** into the search bar.
3. Choose **AWS: Create application with Serverless template** when it populates in the list.
4. The Serverless Land wizard opens to the **Select a Pattern for you application (1/5)** screen in VS Code when the process is complete.

Opening Serverless Land from the AWS Toolkit Explorer.

To open Serverless Land from the AWS Toolkit Explorer, complete the following steps.

1. From the AWS Toolkit Explorer, expand the region that you want to open Serverless Land in.
2. Open the context menu for (right-click) the Lambda node.
3. Choose **Create application with Serverless template** from the context menu.
4. The Serverless Land wizard opens to the **Select a Pattern for you application (1/5)** screen in VS Code when the process is complete.

Opening Serverless Land from the Application Builder explorer

To open Serverless Land from the AWS Toolkit Application Builder explorer, complete the following steps.

1. From the AWS Toolkit Explorer, navigate to the Application Builder explorer.
2. Right-click the Application Builder explorer and choose **Create application with Serverless template** from the context menu.
3. The Serverless Land wizard opens to the **Select a Pattern for you application (1/5)** screen in VS Code when the process is complete.

Creating an application with Serverless template

To create an application with Serverless template, complete the following steps.

1. From the Serverless Land wizard **Select a Pattern for you application (1/5)** screen, choose a Pattern for the base of your application.

   > **Note**
   >
   > To view a preview and more details about a particular Pattern, choose the **Open in Serverless Land** icon located next to the Pattern you want to view. The Serverless Land Pattern opens in your default web browser.

2. From the **Select Runtime (2/5)** screen, choose a runtime for your project.
3. From the **Select IaC (3/5)** screen, choose an IaC option for your project.
4. From the **Select a project location (4/5)** screen, choose a location to store your project.
5. From the **Enter Project Name (5/5)** screen, enter a name for your new application.
6. Your new application displays in the VS Code explorer and your project `readme.md` opens in the VS Code editor, when the procedure is complete.

   > **Note**
   >
   > After your new application is created, additional actions that are specific to your application type can be found in the `readme.md` file. Additionally, your AWS Serverless Application Model (AWS SAM) applications can be opened with AWS Application Builder for local testing, debugging, and more.
   > For details about working with Application Builder in the AWS Toolkit, see the Working with the AWS Application Builder explorer topic in this User Guide.

Running and debugging Lambda functions directly from code

When testing the AWS SAM application, you can choose to run and debug just the Lambda function and exclude other resources that the AWS SAM template defines. This approach involves using the CodeLens feature to identify Lambda function handlers in the source code that you can directly invoke.

The Lambda handlers that are detected by CodeLens depend on the language and runtime that you're using for your application.

Language/runtime	Criteria for Lambda functions to be identified by CodeLens indicators
C# (dotnetcore2.1, 3.1; .NET 5.0)	The function has the following features: • It's a public function of a public class. • It has one or two parameters. With two parameters, the second parameter must implement the `ILambdaContext` interface. • It has a `*.csproj` file in its parent folder within the VS Code workspace folder. The ms-dotnettools.csharp extension (or any extension that provides language symbols for C#) is installed and enabled.
JavaScript/TypeScript (Node.js 12.x, 14.x)	The function has the following features: • It's an exported function with up to three parameters. • It has a `package.json` file in its parent folder within the VS Code workspace folder.
Python (3.7, 3.8, 3.9, 3.10, 3.11, 3.12)	The function has the following features: • It's a top-level function. • It has a `requirements.txt` file in its parent folder within the VS Code workspace folder. The ms-python.python extension (or any extension that provides language symbols for Python) is installed and enabled.
Java (8, 8.al2, 11)	The function has the following features:

Language/runtime	Criteria for Lambda functions to be identified by CodeLens indicators

	• It's a public function of a public, non-abstract class.
	• It has one, two, or three parameters:
	  • One parameter: Parameter can be anything.
	  • Two parameters: Parameters must be a `java.io.InputStream` and a `java.io.OutputStream` OR the last parameter must be a `com.amazonaws.services.lambda.runtime.Context` .
	  • Three parameters: Parameters must be a `java.io.InputStream` and a `java.io.OutputStream` AND the last parameter must be a `com.amazonaws.services.lambda.runtime.Context` .
	• It has a `build.gradle` (Gradle) or `pom.xml` (Maven) file in its parent folder within the VS Code workspace folder.
	
	The redhat.java extension (or any extension that provides language symbols for Java) is installed and enabled. This extension requires Java 11, no matter which Java runtime you're using.
	
	The vscjava.vscode-java-debug extension (or any extension that provides a Java debugger) is installed and enabled.

Language/runtime	Criteria for Lambda functions to be identified by CodeLens indicators
Go (1.x)	The function has the following features: • It's a top-level function. • It takes between 0 and 2 arguments. If there are two arguments, the first argument must implement `context.Context` . • It returns between 0 and 2 arguments. If there are more than 0 arguments, the last argument must implement `error`. • It has a `go.mod` file within the VS Code workspace folder. The golang.go extension is installed, configured, and enabled.

To run and debug a serverless application directly from the application code

1. To view your application files in the VS Code File Explorer, choose **View**, **Explorer**.
2. From the application folder (for example, *my-sample-app*), expand the function folder (in this case, *hello-world*) and open the `app.js` file.
3. In the CodeLens indicator that identifies an eligible Lambda function handler, choose `Add Debug Configuration`.
   Access the Add Debug Configuration option in the CodeLens indicator for a Lambda function handler.
4. In the **Command Palette**, select the runtime in which your AWS SAM application will run.
5. In the editor for the `launch.json` file, edit or confirm values for the following configuration properties:
   - `"name"` – Enter a reader-friendly name to appear in the **Configuration** dropdown field in the **Run** view.

- `"target"` – Ensure that the value is `"code"` so that a Lambda function handler is directly invoked.
- `"lambdaHandler"` – Enter the name of the method within your code that Lambda calls to invoke your function. For example, for applications in JavaScript, the default is `app.lambdaHandler`.
- `"projectRoot"` – Enter the path to the application file that contains the Lambda function.
- `"runtime"` – Enter or confirm a valid runtime for the Lambda execution environment, for example, `"nodejs12.x"`.
- `"payload"` – Choose one of the following options to define the event payload that you want to provide to your Lambda function as input:
  - `"json"`: JSON-formatted key-value pairs that define the event payload.
  - `"path"`: A path to the file that's used as the event payload.

  In the example below, the `"json"` option defines the payload.

Configuring the `launch.json` file for directly invoking Lambda functions.

For more information about these and other entries in the `launch.json` file, see Configuration options for debugging serverless applications.

6. If you're satisfied with the debug configuration, to start debugging, choose the green play arrow next to **RUN**.

   When the debugging sessions starts, the **DEBUG CONSOLE** panel shows debugging output and displays any values that the Lambda function returns. (When debugging AWS SAM applications, **AWS Toolkit** is selected as the **Output** channel in the **Output** panel.)

Running and debugging local Amazon API Gateway resources

You can run or debug AWS SAM API Gateway local resources, specified in `template.yaml`, by running a VS Code launch config of `type=aws-sam` with the `invokeTarget.target=api`.

> **Note**
>
> API Gateway supports two types of APIs, REST and HTTP. However, the API Gateway feature with the AWS Toolkit for Visual Studio Code only supports REST APIs. Sometimes HTTP APIs are called "API Gateway V2 APIs."

To run and debug local API Gateway resources

1. Choose one of the following approaches to create a launch config for an AWS SAM API Gateway resource:

   - **Option 1:** Visit the handler source code (.js, .cs, or .py file) in your AWS SAM project, hover over the Lambda handler, and choose the **Add Debug Configuration** CodeLens. Then, in the menu, choose the item marked **API Event**.
   - **Option 2:** Edit `launch.json` and create a new launch configuration using the following syntax.

   ```
   {
     "type": "aws-sam",
     "request": "direct-invoke",
     "name": "myConfig",
     "invokeTarget": {
       "target": "api",
       "templatePath": "n12/template.yaml",
       "logicalId": "HelloWorldFunction"
     },
     "api": {
       "path": "/hello",
       "httpMethod": "post",
       "payload": {
         "json": {}
       }
     },
     "sam": {},
     "aws": {}
   }
   ```

2. In the VS Code **Run** panel, choose the launch config (named `myConfig` in the above example).

3. (Optional) Add breakpoints to your Lambda project code.
4. Type **F5** or choose **Play** in the **Run** panel.
5. In the output pane, view the results.

Configuration

When you use the `invokeTarget.target` property value `api`, the Toolkit changes the launch configuration validation and behavior to support an `api` field.

```
{
  "type": "aws-sam",
  "request": "direct-invoke",
  "name": "myConfig",
  "invokeTarget": {
    "target": "api",
    "templatePath": "n12/template.yaml",
    "logicalId": "HelloWorldFunction"
  },
  "api": {
    "path": "/hello",
    "httpMethod": "post",
    "payload": {
      "json": {}
    },
    "querystring": "abc=def&qrs=tuv",
    "headers": {
        "cookie": "name=value; name2=value2; name3=value3"
    }
  },
  "sam": {},
  "aws": {}
}
```

Replace the values in the example as follows:

invokeTarget.logicalId

An API resource.

path

The API path that the launch config requests, for example, `"path": "/hello"`.

Must be a valid API path resolved from the `template.yaml` specified by `invokeTarget.templatePath`.

httpMethod

One of the following verbs: "delete", "get", "head", "options", "patch", "post", "put".

payload

The JSON payload (HTTP body) to send in the request , with the same structure and rules as the lambda.payload field.

`payload.path` points to a file containing the JSON payload.

`payload.json` specifies a JSON payload inline.

headers

Optional map of name-value pairs, which you use to specify HTTP headers to include in the request, as shown in the following example.

```
"headers": {
     "accept-encoding": "deflate, gzip;q=1.0, *;q=0.5",
     "accept-language": "fr-CH, fr;q=0.9, en;q=0.8, de;q=0.7, *;q=0.5",
     "cookie": "name=value; name2=value2; name3=value3",
     "user-agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6)
 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.198 Safari/537.36",
}
```

querystring

Optional string which sets the `querystring` of the request, for example, `"querystring": "abc=def&ghi=jkl"`.

AWS

How AWS connection information is provided. For more information, see the **AWS connection ("aws") properties** table in the Configuration options for debugging serverless applications section.

sam

How the AWS SAM CLI builds the application. For more information, see the **AWS SAM CLI ("sam") properties** table in the Configuration options for debugging serverless applications section.

Configuration options for debugging serverless applications

When you open the `launch.json` file to edit debug configurations, you can use the VS Code IntelliSense feature to view and automatically complete valid properties. To trigger IntelliSense in the editor, press **Ctrl**+**Spacebar**.

IntelliSense enables you to find and define properties for invoking Lambda functions directly or with the AWS SAM template. You can also define properties for `"lambda"` (how the function runs), `"sam"` (how the AWS SAM CLI builds the application), and `"aws"` (how AWS connection information is provided).

AWS SAM: Direct Lambda handler invoke / Template-based Lambda invoke

Property	Description
`type`	Specifies which extension manages the launch configuration. Always set to `aws-sam` to use the AWS SAM CLI to build and debug locally.
`name`	Specifies a reader-friendly name to appear in the **Debug launch configuration** list.
`request`	Specifies the type of configuration to be performed by the designated extension (`aws-sam`). Always set to `direct-invoke` to start the Lambda function.
`invokeTarget`	Specifies the entry point for invoking the resource.

Property	Description
	For invoking the Lambda function directly, set values for the following `invokeTarget` fields: • `target` – Set to `code`. • `lambdaHandler` – The name of the Lambda function handler to invoke. • `projectRoot` – The path for the application file containing the Lambda function handler. • `architecture` – Processor architecture of the emulated environment in which your local SAM Lambda application runs. For certain runtimes, you can choose `arm64` instead of the default `x86_64` architecture. For more information, see Creating a new serverless application (local). For invoking the Lambda resources with the AWS SAM template, set values for the following `invokeTarget` fields: • `target` – Set to `template`. • `templatePath` – The path to the AWS SAM template file. • `logicalId` – The resource name of the `AWS::Lambda::Function` or `AWS::Serverless::Function` to invoke. You can find the resource name in the YAML-formatted AWS SAM template. Note that the AWS Toolkit implicitly recognizes functions defined with `PackageType: Image` in the AWS SAM template as Image-based Lambda functions. For more information, see Lambda deployment packages in the *AWS Lambda Developer Guide*.

Lambda (`"lambda"`) properties

Property	Description
`environmentVariables`	Passes operational parameters to your Lambda function. For example, if you're writing to an Amazon S3 bucket, instead of

Property	Description

	hard-coding the bucket name that you're writing to, configure the bucket name as an environment variable.
	
	Note
	When specifying environment variables for a serverless application, you must add configurations to both the AWS SAM template (`template.yaml`) and the `launch.json` file.
	Example of formatting for an environment variable in the AWS SAM template:
	`Resources:`
	`HelloWorldFunction:`
	`Type: AWS::Serverless::Function`
	`Properties:`
	`CodeUri: hello-world/`
	`Handler: app.lambdaHandlerN10`
	`Runtime: nodejs10.x`
	`Environment:`
	`Variables:`
	`SAMPLE1: Default Sample 1 Value`
	Example of formatting for an environment variable in the `launch.json` file:
	`"environmentVariables": {`
	`"SAMPLE1": "My sample 1 value"`
	`}`

`payload`	Provides two options for the event payload that you provide to your Lambda function as input. • `"json"`: JSON-formatted key-value pairs that define the event payload. • `"path"`: A path to the file that's used as the event payload.

Property	Description
`memoryMB`	Specifies megabytes (MB) of memory provided for running an invoked Lambda function.
`runtime`	Specifies the runtime that the Lambda function uses. For more information, see AWS Lambda runtimes.
`timeoutSec`	Sets the time allowed, in seconds, before the debug session times out.

Property	Description
`pathMappings`	Specifies where local code is in relation to where it runs in the container. By default, the Toolkit for VS Code sets `localRoot` to the Lambda function's code root in the local workspace, and `remoteRoot` to `/var/task`, which is the default working directory for code running in Lambda. If the working directory is changed in the Dockerfile or with the `WorkingDirectory` parameter in the CloudFormation template file, at least one `pathMapping` entry must be specified so that the debugger can successfully map locally set breakpoints to the code running in the Lambda container. Example of formatting for `pathMappings` in the `launch.json` file: `"pathMappings": [ { "localRoot": " ${workspaceFolder}/sam-app/HelloWorldFunction ", "remoteRoot": " /var/task " } ]` Caveats: • For .NET image-based Lambda functions, the `remoteRoot` entry must be the build directory. • For Node.js-based Lambda functions, you can specify only a single path mapping entry.

The Toolkit for VS Code uses the AWS SAM CLI to build and debug serverless applications locally. You can configure the behavior of AWS SAM CLI commands using properties of the `"sam"` configuration in the `launch.json` file.

AWS SAM CLI ("sam") properties

Property	Description	Default value
buildArguments	Configures how the sam build command builds your Lambda source code. To view build options, see sam build in the *AWS Serverless Application Model Developer Guide*.	Empty string
containerBuild	Indicates whether to build your function inside a Lambda-like Docker container.	false
dockerNetwork	Specifies the name or ID of an existing Docker network that the Lambda Docker containers should connect to, along with the default bridge network. If not specified, the Lambda containers connect only to the default bridge Docker network.	Empty string
localArguments	Specifies additional local invoke arguments.	Empty string
skipNewImageCheck	Specifies whether the command should skip pulling down the latest Docker image for Lambda runtime.	false
template	Customizes your AWS SAM template using parameters to input customer values. For more information, see	"parameters":{}

Property	Description	Default value
	Parameters in the *AWS CloudFormation User Guide*.	

AWS connection ("aws") properties

Property	Description	Default value
`credentials`	Selects a specific profile (for example, `profile:default`) from your credential file to get AWS credentials.	The AWS credentials that your existing shared AWS config file or shared AWS credentials file provide to the Toolkit for VS Code.
`region`	Sets the AWS Region of the service (for example, us-east-1).	The default AWS Region associated with the active credentials profile.

Example: Template launch configuration

Here is an example launch configuration file for an AWS SAM template target:

```
{
    "configurations": [
        {
            "type": "aws-sam",
            "request": "direct-invoke",
            "name": "my-example:HelloWorldFunction",
            "invokeTarget": {
                "target": "template",
                "templatePath": "template.yaml",
                "logicalId": "HelloWorldFunction"
            },
            "lambda": {
                "payload": {},
                "environmentVariables": {}
            }
        }
    ]
```

```
}
```

Example: Code launch configuration

Here is an example launch configuration file for a Lambda function target:

```
{
    "configurations": [
        {
            "type": "aws-sam",
            "request": "direct-invoke",
            "name": "my-example:app.lambda_handler (python3.7)",
            "invokeTarget": {
                "target": "code",
                "projectRoot": "hello_world",
                "lambdaHandler": "app.lambda_handler"
            },
            "lambda": {
                "runtime": "python3.7",
                "payload": {},
                "environmentVariables": {}
            }
        }
    ]
}
```

Troubleshooting serverless applications

This topic details common errors that you might encounter when creating serverless applications with the Toolkit for VS Code and how to resolve them.

Topics

How can I use a samconfig.toml with a SAM launch configuration?

Specify the location of your SAM CLI samconfig.toml by configuring the `--config-file` argument in the `sam.localArguments` property of your launch configuration. For example, if the samconfig.toml file is located at the top level of your workspace:

```
"sam": {
    "localArguments": ["--config-file", "${workspaceFolder}/samconfig.toml"],
}
```

Error: "RuntimeError: Container does not exist"

The `sam build` command can show this error if your system does not have enough disk space for the Docker container. If your system storage has only 1-2 GB of space available, `sam build` might fail during processing, even if system storage is not completely full before the build starts. For more information, see this GitHub issue.

Error: "docker.errors.APIError: 500 Server Error ... You have reached your pull rate limit."

Docker Hub limits requests that anonymous users can make. If your system reaches the limit, Docker fails and this error appears in the OUTPUT view of VS Code:

```
docker.errors.APIError: 500 Server Error: Internal Server Error ("toomanyrequests: You
 have
reached your pull rate limit. You may increase the limit by authenticating and
 upgrading:
https://www.docker.com/increase-rate-limit")
```

Ensure that your system Docker service has authenticated with your Docker Hub credentials.

Error: "500 Server Error: Mounting C:\Users\..."

Windows users might see this Docker mounting error when debugging AWS SAM applications:

```
Fetching lambci/lambda:nodejs10.x Docker container image......
2019-07-12 13:36:58 Mounting C:\Users\<username>\AppData\Local\Temp\ ... as /var/
task:ro,delegated inside runtime container
Traceback (most recent call last):
```

```
...
requests.exceptions.HTTPError: 500 Server Error: Internal Server Error ...
```

Try refreshing the credentials for your shared drives (in the Docker settings).

Using WSL, webviews (for example, the "Invoke on AWS" form) are broken

This is a known VS Code issue for users of Cisco VPN. For more information, see this GitHub issue.

A workaround is suggested in this WSL tracking issue.

Debugging a TypeScript application, but breakpoints are not working

This will happen if there isn't a source map to link the compiled JavaScript file to the source TypeScript file. To correct this, open your `tsconfig.json` file and ensure the following option and value are set: `"inlineSourceMap": true`.

Working with Systems Manager Automation documents

AWS Systems Manager gives you visibility and control of your infrastructure on AWS. Systems Manager provides a unified user interface so you can view operational data from multiple AWS services and automate operational tasks across your AWS resources.

A Systems Manager document defines the actions that Systems Manager performs on your managed instances. An Automation document is a type of Systems Manager document that you use to perform common maintenance and deployment tasks such as creating or updating an Amazon Machine Image (AMI). This topic outlines how to create, edit, publish, and delete Automation documents with AWS Toolkit for Visual Studio Code.

Topics

- Troubleshooting Systems Manager Automation documents in Toolkit for VS Code

Assumptions and prerequisites

Before you begin, make sure:

- You have installed Visual Studio Code and the latest version of the AWS Toolkit for Visual Studio Code. For more information, see Installing the AWS Toolkit for Visual Studio Code.
- You're familiar with Systems Manager. For more information, see the *AWS Systems Manager User Guide*.
- You're familiar with Systems Manager Automation use cases. For more information, see AWS Systems Manager Automation in the *AWS Systems Manager User Guide*.

IAM permissions for Systems Manager Automation documents

In the Toolkit for VS Code you must have a credentials profile that contains the AWS Identity and Access Management (IAM) permissions necessary to create, edit, publish, and delete Systems Manager Automation documents. The following policy document defines the necessary IAM permissions that can be used in a principal policy:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:ListDocuments",
                "ssm:ListDocumentVersions",
                "ssm:DescribeDocument",
                "ssm:GetDocument",
                "ssm:CreateDocument",
                "ssm:UpdateDocument",
                "ssm:UpdateDocumentDefaultVersion",
                "ssm:DeleteDocument"
            ],
            "Resource": "*"
        }
```

```
    ]
}
```

For information on how to update an IAM policy, see Creating IAM policies in the *IAM User Guide*. For information on how to set up your credentials profile, see AWS IAM credentials.

Creating a new Systems Manager Automation document

You can create a new Automation document in `JSON` or `YAML` using Visual Studio Code. When you create a new Automation document, it will be presented in an untitled file. You can name your file and save it in VS Code, however the name of the file isn't visible to AWS.

To create a new Automation document

1. Open VS Code.
2. On the **View** menu, choose **Command Palette** to open the Command Palette.
3. In the Command Palette, enter **AWS Toolkit Create a new Systems Manager Document Locally**.
4. Choose one of the starter templates for a Hello World example.
5. Choose either `JSON` or `YAML`.

   A new Automation document is created.

> **Note**
>
> Your new Automation document in VS Code doesn't automatically appear in AWS. You must publish it to AWS before you can run it.

Opening an existing Systems Manager Automation document

You use the AWS Explorer to find existing Systems Manager Automation documents. When you open an existing Automation document, it appears as an untitled file in VS Code.

To open your Automation document

1. Open VS Code.

2. From the left-hand navigation, choose **AWS** to open the AWS Explorer.
3. In the AWS Explorer, for **Systems Manager**, choose the download icon on the document that you want to open and then choose the document version. The file will open in the format for that version. Otherwise choose either **Download as JSON** or **Download as YAML**.

> **Note**
>
> Locally saving an Automation document as a file in VS Code doesn't make it appear in AWS. It needs to be published to AWS before executing.

Editing a Systems Manager Automation document

If you own any Automation documents, they appear in the **Owned by Me** category of Systems Manager documents in the AWS Explorer. You can own Automation documents that already exist in AWS, and you can own new or updated documents that you previously published to AWS from VS Code.

When you open an Automation document for editing in VS Code, you can do more with it than you can in the AWS Management Console. For example:

- There is schema validation on both `JSON` and `YAML` formats.
- There are snippets available in the document editor for you to create any of the automation step types.
- There is auto-complete support on various options in `JSON` and `YAML`.

Working with versions

Systems Manager Automation documents use versions for change management. You can choose the default version for an Automation document in VS Code.

To set a default version

- In the AWS Explorer, navigate to the document that you want to set the default version on, open the context (right-click) menu for the document, and choose **Set default version**.

> **Note**
>
> If the chosen document only has one version, you won't be able to change the default.

Publishing a Systems Manager Automation document

After you edit your Automation document in VS Code, you can publish it to AWS.

To publish your Automation document

1. Open the Automation document that you want to publish using the procedure outlined in Opening an existing Systems Manager Automation document.
2. Make the changes that you want to be published. For more information, see Editing a Systems Manager Automation document.
3. In the upper right of the open file, choose the upload icon.
4. In the publishing workflow dialog box, choose the AWS Region that you want to publish the Automation document to.
5. If you're publishing a new document, choose **Quick Create**. Otherwise, choose **Quick Update** to update an existing Automation document in that AWS Region.
6. Enter the name for this Automation document.

When you publish an update to an existing Automation document to AWS, a new version is added to the document.

Deleting a Systems Manager Automation document

You can delete Automation documents in VS Code. Deleting an Automation document deletes the document and all versions of the document.

> **Important**
>
> - Deleting is a destructive action that can't be undone.
> - Deleting an Automation document that has already been run doesn't delete the AWS resources that were created or modified when it was started.

To delete your Automation document

1. Open VS Code.
2. From the left-hand navigation, choose **AWS** to open the AWS Explorer.
3. In the AWS Explorer, for **Systems Manager**, open the context (right-click) menu for the document you want to delete, and choose **Delete document**.

Executing a Systems Manager Automation document

Once your Automation document is published to AWS, you can run it to perform tasks on your behalf in your AWS account. To run your Automation document, you use the AWS Management Console, the Systems Manager APIs, the AWS CLI, or the AWS Tools for PowerShell. For instructions on how to run an Automation document, see Running a simple automation in the *AWS Systems Manager User Guide*.

Alternatively, if you want to use one of the AWS SDKs with the Systems Manager APIs to run your Automation document, see the AWS SDK references.

> **Note**
>
> Executing an Automation document can create new resources in AWS and can incur billing costs. We strongly recommend that you understand what your Automation document will create in your account before you started it.

Troubleshooting Systems Manager Automation documents in Toolkit for VS Code

I saved my Automation document in VS Code, but I don't see it in the AWS Management Console.

Saving an Automation document in VS Code does not publish the Automation document to AWS. For more information on publishing your Automation document, see Publishing a Systems Manager Automation document.

Publishing my Automation document failed with a permissions error.

Make sure your AWS credentials profile has the necessary permissions to publish Automation documents. For an example permissions policy, see IAM permissions for Systems Manager Automation documents.

I published my Automation document to AWS, but I don't see it in the AWS Management Console.

Make sure that you've published the document to the same AWS Region you're browsing in the AWS Management Console.

I've deleted my Automation document, but I'm still being billed for the resources it created.

Deleting an Automation document doesn't delete the resources it created or modified. You can identify the AWS resources that you've created from the AWS Billing Management Console, explore your charges, and choose what resources to delete from there.

AWS Step Functions

With AWS Step Functions, you can create workflows (also called state machines) to build distributed applications, automate processes, orchestrate microservices, and create data and machine learning pipelines. The following topics describe how to work with AWS Step Functions in the AWS Toolkit for Visual Studio Code. For detailed information about the AWS Step Functions service, see the AWS Step Functions Developer Guide.

Topics

- Working with AWS Step Functions
- Working with AWS Step Functions Workflow Studio

Working with AWS Step Functions

The following sections describe how to work with AWS Step Functions `Amazon State Language` (`ASL`) files containing state machine definitions in the AWS Toolkit. For detailed information about AWS Step Functions state machines, see the Learn about state machines in Step Functions topic in the *AWS Step Functions* Developer Guide.

Viewing Step Functions state machines

To view your existing `ASL` files containing state machine definitions in the AWS Toolkit Explorer, complete the following steps.

1. From the AWS Toolkit Explorer, expand the region that contains the ASL file that you want to view.
2. Expand the **Step Functions** heading.
3. Your ASL files are displayed in the AWS Explorer.

Creating a Step Functions state machine

In the AWS Toolkit, you can create a new Step Functions state machine from a file or you can use a template. The following procedure describes how to create a Step Functions state machine from a file. For details about creating a SFN; state machine from a template, see the *State machine templates* section located below, in this User Guide topic.

> **Note**
>
> To work with Step Functions in VS Code, the extension of your `Amazon State Language`(ASL) file that contains your state machine definition must end with `asl.json`, `asl.yml`, or `.asl.yaml`.
> By default, relevant Step Functions files open in Workflow Studio. For detailed information about working in Workflow Studio through the AWS Toolkit, see the Working with Workflow Studio topic in this User Guide.

1. From your workspace in VS Code, create a new file.
2. Name your file and specify the file extension as `asl.json`, `asl.yml`, or `.asl.yaml`.
3. Upon creation, the AWS Toolkit opens the new file in AWS Step Functions Workflow Studio.
4. From **Workflow Studio** choose the **Save** button from the utility menu to save your new ASL file.

Creating a Step Functions state machine from a template

In the AWS Toolkit, you can create a Step Functions state machine from a template. The template process creates a ASL file that contains a state machine definition, providing a starting point for your project. The following procedure describes how to create a Step Functions state machine from a template in the AWS Toolkit.

1. From the AWS Toolkit Explorer, expand the region that you want to create a Step Functions state machine in.
2. Open the context menu for (right-click) **Step Functions** and choose **Create a new Step Functions state machine** to open the **Select a starter template(1/2)** wizard in VS Code.
3. From the **Select a starter template(1/2)** wizard, choose the template type for your Step Functions state machine to proceed.
4. From the **Select template format(2/2)** screen, choose either **YAML** or **JSON** for your template format.
5. A new ASL file containing your state machine definition is opened in the VS Code editor.

Downloading a Step Functions state machine

To download a remotely stored Step Functions state machine to your local instance of VS Code, complete the following steps.

1. From the AWS Toolkit Explorer, expand the region that contains the Step Functions state machine that your want to download.
2. Expand **Step Functions**, then right-click the Step Functions state machine you want to download and choose **Download Definition...**.
3. Specify a location to store your Step Functions state machine locally to proceed.
4. The Step Functions state machine opens in Workflow Studio when the procedure is complete.

Saving changes to a Step Functions state machine

The following procedure describes how to save changes made to your Step Functions state machine.

> **Note**
>
> Edits made in Workflow Studio sync to your local file, but remain unsaved until your work is saved in the VS Code editor or Workflow Studio. If you local file is modified and saved while Workflow Studio is open and there are no errors detected in your ASL file, then you receive a **Success** notification in Workflow Studio, when saving is complete. However, if your local file contains invalid JSON or YAML and you attempt to save, then your local file fails to sync and you receive a **Warning** notification in Workflow Studio.

1. From an open ASL file containing a state machine definition in Workflow Studio, navigate to the **Utility buttons**.
2. Choose the **Save** button.
3. VS Code notifies you when the file has been saved.

Running a Step Functions state machine

The following procedure describes how to run a Step Functions state machine in the AWS Toolkit.

1. From the AWS Toolkit Explorer, expand the region containing the Step Functions state machine that you want to run.
2. Expand **Step Functions**, then right-click the Step Functions state machine that you want to run.
3. From the context menu, choose **Start Execution** to initiate the launch process.
4. The status of the launch is displayed in the **AWS Toolkit Output** window in VS Code.

Working with code snippets

Code snippets are automated suggestions that generate based on the code that you're working on. To work with code snippets with Step Functions in the toolkit, complete the following steps.

> **Note**
>
> To work with Step Functions code snippets in VS Code, the extension of your ASL file that contains your state machine definition must end with `.asl.json`, `.asl.yml`, or `.asl.yaml`.
> By default, your relevant Step Functions files open in Workflow Studio.

1. From VS Code, open an ASL file containing the state machine definition that you want to modify or create a new ASL file.
2. From Workflow Studio, switch to **Code** mode if you're in **Design** mode.
3. From the Workflow Studio code editor, place your cursor in the `"States"` property.
4. Press **`control + space`** to open the code snippets menu, additional properties can be accessed by pressing **`control + space`** and are based on the `"State" "Type"`.

5. Choose the code snippet that you want from the list.

Code validation

As you work on Step Functions in Workflow Studio, code validation actively identifies errors and makes suggestions for the following:

- Missing properties
- Incorrect values
- Non terminal state
- Nonexistent states that are pointed to

Working with AWS Step Functions Workflow Studio

The following sections describe how to work with AWS Step Functions Workflow Studio in the AWS Toolkit for Visual Studio Code. For detailed information about AWS Step Functions Workflow Studio, see the Developing workflows topic in the *AWS Step Functions* Developer Guide

Opening Workflow Studio

The following list describes the different paths available for you to open Workflow Studio in VS Code.

> **Note**
>
> To work with Workflow Studio in VS Code, the extension of your `Amazon State Language`(ASL) file that contains your state machine definition, must end with `asl.json`, `asl.yml` or `asl.yaml`. For details about downloading or creating a new state machine definition in the AWS Toolkit, see the *Downloading state machines* and *Creating a state machine* sections in the Working with AWS Step Functions topic of this User Guide.

- From the AWS Explorer, open the context menu for (right-click) an ASL file containing a state machine definition, then choose **Open in Workflow Studio**.
- From an open ASL file containing a state machine definition, choose the **Open with Workflow Studio** icon located next to the tabs in the VS Code editor window.

- From an open ASL file containing a state machine definition, choose the CodeLens command **Open with Workflow Studio**, located at the top of the file.
- Closing and reopening an ASL file containing a state machine definition automatically reopens the file in Workflow Studio, unless the default Workflow Studio is disabled manually.

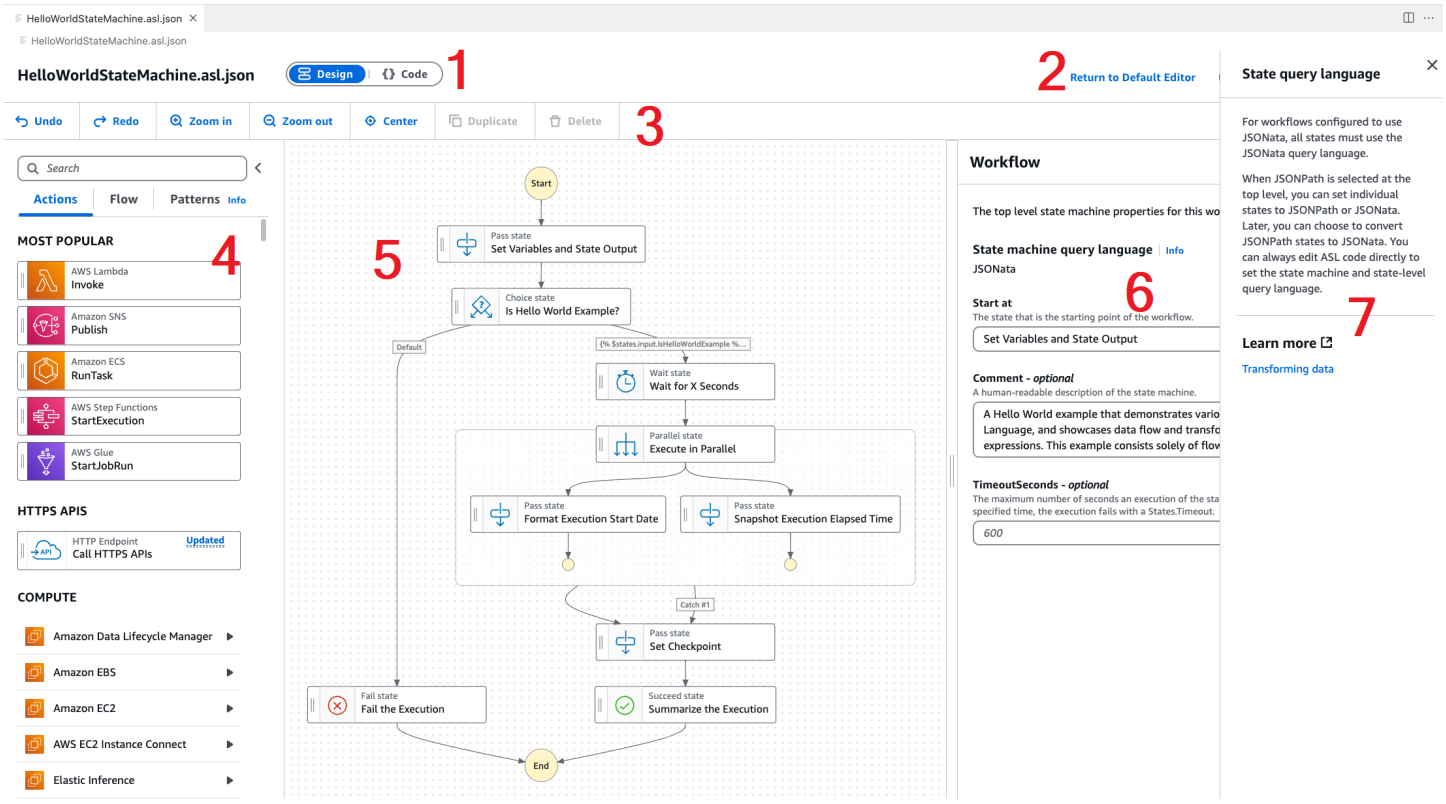
Design mode and Code mode

Workflow Studio has two modes for working with your ASL files containing a state machine definition: **Design** mode and **Code** mode. **Design** mode provides a graphic interface to visualize your workflows as you build prototypes. **Code** mode has an integrated code editor where you can view, write, and edit the ASL definitions in your workflows.

> **Note**
>
> For detailed information about each of the UI sections in both Design and Code modes, see the Using Workflow Studio topic in the *AWS Step Functions* Developer Guide. Not all Workflow Studio features are available in the AWS Toolkit, such as **Config mode**, for example.

The **Design** mode UI has 7 main sections, as labeled and described in the following image.

1. Mode Buttons: Buttons for switching between **Design** and **Code** modes.
2. Utility buttons: A set of buttons for performing tasks, such as exiting Workflow Studio, saving your workflows, or exporting ASL definitions in a JSON or YAML file.
3. Design toolbar: Toolbar containing a set of buttons that perform common actions, such as undo, delete, and zoom control.
4. States Browser: Browser containing drag-and-drop states for your workflow canvas. States are organized into tabs and defined as **Actions**, **Flow**, and **Patterns**.
5. The Canvas and workflow graph: A visual rendering of your workflow where you can drop, reorganize, and select states for configuration.
6. Inspector Panel: View and edit the properties of any state selected on the canvas. Depending on the state selected in the canvas workflow graph, tabs populate with state-specific options for **Configuration**, **Input/Output**, **Variables**, and **Error handling**.
7. Info links: Opens a panel with contextual information when you need help. These panels also include links to related topics in the *AWS Step Functions* Developer Guide.



Using single-state tests during design

From the Workflow Studio test-state UI, you can test the individual states of your state machine. This includes the ability to provide state inputs, set variables, and make both AWS SAM and CloudFormation definition substitutions.

To learn more about infrastructure as code (IaC), resource definitions, and transforming data, see the Using AWS SAM to build Step Functions workflows and Transforming data with JSONata in Step Functions topics in the *AWS Step Functions* Developer Guide.

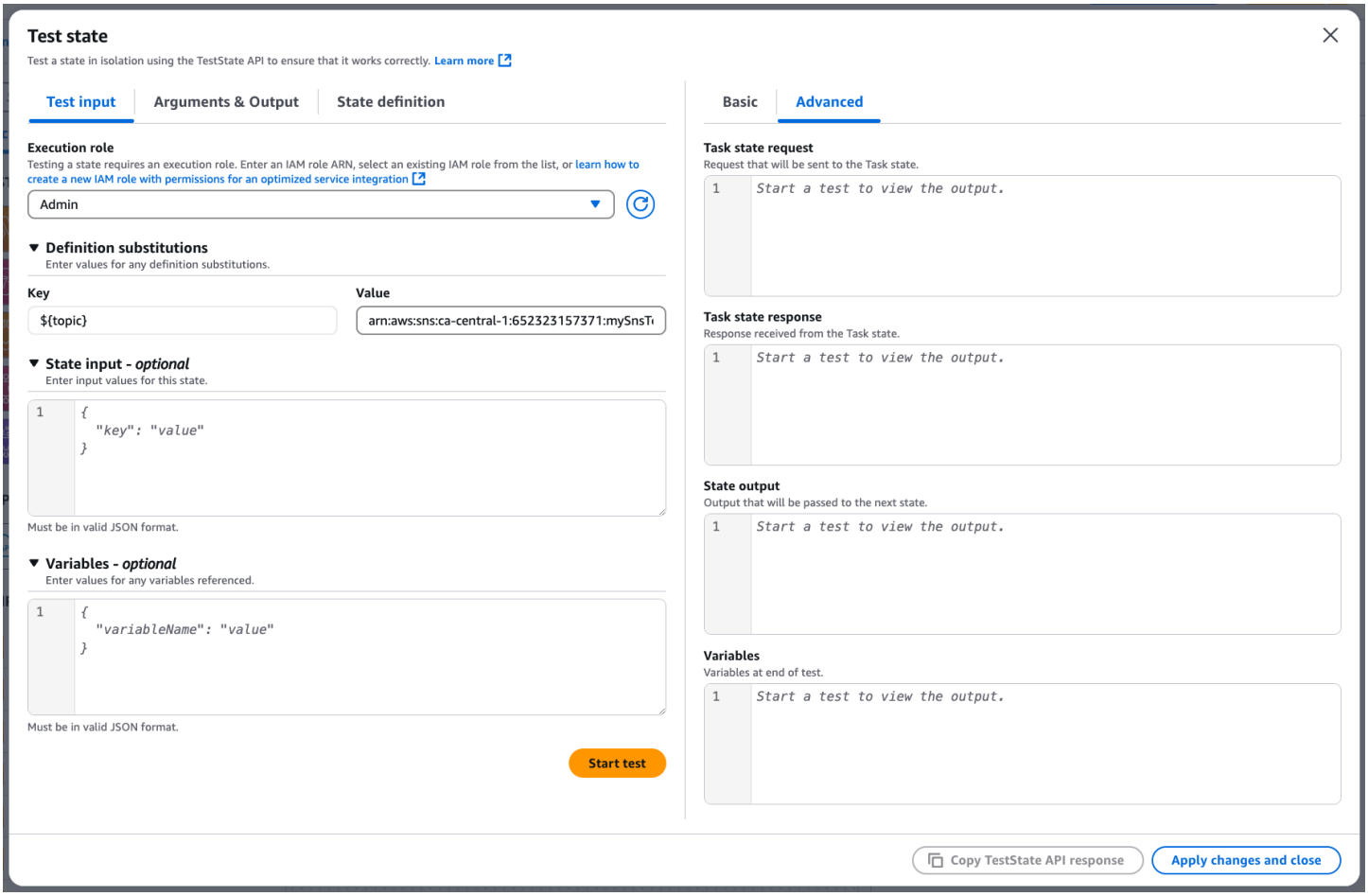
The following procedure describes how to open the test-state UI in Workflow Studio.

Opening the test state UI

1. From the **Design** mode tab in Workflow Studio, navigate to the canvas and choose a state to open it in the **Inspector** panel.
2. From the **Inspector** panel, choose the **Test state** button.
3. The **Test state** UI opens in VS Code.

The test-state UI has 3 main tabs, **Test input**, **Arguments & Output**, **State definition**. The **Test input** tab has 3 additional fields that allow you to provide **State input**, set **variables**, and specify **Definition substitutions** from your AWS SAM or CloudFormation templates. In the **State definition** tab, you can adjust the workflow and re-test. When you're finished running tests, you can apply and save changes to your state machine definition.

The following screenshot shows the test-state UI, which includes a topic-resources definition.



Disabling Workflow Studio by default

By default, Workflow Studio is the default editor for ASL files containing a state machine definition. You can disable the default setting by modifying your `settings.json` file in your local `.vscode` directory. If you disable Workflow Studio by default, it is still accessible through the methods listed in the *Opening Workflow Studio* section, located in this topic.

To edit your `settings.json` file from VS Code, complete the following steps.

1. From VS Code, open the **Command Palette** by pressing **`option+shift+p`** (Mac) or **`ctrl+shift+p`** (Windows).
2. From the VS Code **Command Palette**, enter **`Open User Settings (JSON)`** into the search field and choose the option when it populates in the list.
3. From `settings.json` in your editor, add the following modification to your file.

```
                {
                    "workbench.editorAssociations": {
                    // Use all the following overrides or a specific one for a
certain file type
                    "*.asl.json": "default",
                    "*.asl.yaml": "default",
                    "*.asl.yml": "default"
                    }
                }
```

4. Save your changes to `settings.json` and refresh or restart VS Code.

Working with Threat Composer

You can use the AWS Toolkit for Visual Studio Code to work with the Threat Composer tool. Threat Composer is a threat-modeling tool that can simplify your threat modeling process.

For detailed information about the Threat Composer tool, see the Threat Composer GitHub repository.

The following topics describe how to work with Threat Composer in the AWS Toolkit for Visual Studio Code.

Topics

- Working with Threat Composer from the Toolkit

Working with Threat Composer from the Toolkit

With Threat Composer you can create, view, and edit Threat Composer threat models directly in VS Code. For detailed information about the Threat Composer tool, see the Threat Composer GitHub repository.

The following sections describe how to access Threat Composer tools in the AWS Toolkit for Visual Studio Code.

Accessing Threat Composer from the Toolkit

There are 3 main ways that you can access Threat Composer from the Toolkit.

Accessing Threat Composer through an existing threat model

To open Threat Composer, open an existing threat-model file (extension `.tc.json`) in VS Code. Threat Composer automatically opens and renders a visualization of your threat-model file in the VS Code editor window.

Creating a new Threat Composer threat model

1. From the VS Code main menu, expand **File**, then choose **New File**.
2. From the **New File** dialog, choose **Threat Composer File...**.
3. When prompted, enter a `file name`, then press the **`enter`** key to open Threat Composer and create a visualization of your empty threat-model file in a new VS Code editor window.

Creating a new Threat Composer threat model from the Command Palette

1. From VS Code, open the Command Palette by pressing **`Cmd + Shift + P`** or **`Ctrl + Shift + P`** (Windows).
2. In the search field, enter **`Threat Composer`** and choose **Create New Threat Composer File** when it populates in the results.
3. When prompted, enter a `file name`, then press the **`enter`** key to open Threat Composer and create a visualization of your empty threat-model file in a new VS Code editor window.

Working with resources

In addition to accessing AWS services that are listed by default in the AWS Explorer, you can also go to **Resources** and choose from hundreds of resources to add to the interface. In AWS, a **resource** is an entity you can work with. Some of the resources that can be added include Amazon AppFlow, Amazon Kinesis Data Streams, AWS IAM roles, Amazon VPC, and Amazon CloudFront distributions.

After making your selection, you can go to **Resources** and expand the resource type to list the available resources for that type. For example, if you select the `AWS`

`Toolkit:Lambda::Function` resource type, you can access the resources that define different functions, their properties, and their attributes.

After adding a resource type to **Resources**, you can interact with it and its resources in the following ways:

- View a list of existing resources that are available in the current AWS Region for this resource type.
- View a read-only version of the JSON file that describes a resource.
- Copy the resource identifier for the resource.
- View the AWS documentation that explains the purpose of the resource type and the schema (in JSON and YAML formats) for modelling a resource.
- Create a new resource by editing and saving a JSON-formatted template that conforms to a schema.*
- Update or delete an existing resource.*

> **Important**
>
> *In the current release of the AWS Toolkit for Visual Studio Code the option to create, edit, and delete resources is an *experimental feature*. Because experimental features continue to be tested and updated, they may have usability issues. And experimental features may be removed from the AWS Toolkit for Visual Studio Code without notice.
> To allow the use of experimental features for resources, open the **Settings** pane in your VS Code IDE, and expand **Extensions** and choose **AWS Toolkit**.
> Under **AWS Toolkit Experiments**, select **jsonResourceModification** to allow you to create, update, and delete resources.
> For more information, see Working with experimental features.

IAM permissions for accessing resources

You require specific AWS Identity and Access Management permissions to access the resources associated with AWS services. For example, an IAM entity, such as a user or a role, requires Lambda permissions to access AWS `Toolkit:Lambda::Function` resources.

In addition to permissions for service resources, an IAM entity requires permissions to permit the Toolkit for VS Code to call AWS Cloud Control API operations on its behalf. Cloud Control API operations allow the IAM user or role to access and update the remote resources.

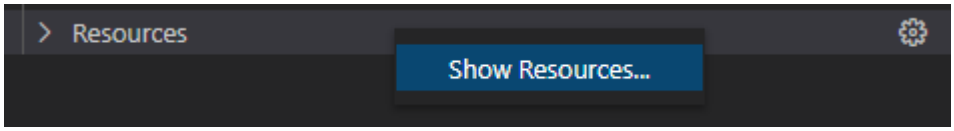
The easiest way to grant permissions is to attach the AWS managed policy, **PowerUserAccess**, to the IAM entity that's calling these API operations using the Toolkit interface. This managed policy grants a range of permissions for performing application development tasks, including calling API operations.

For specific permissions that define allowable API operations on remote resources, see the AWS Cloud Control API User Guide.

Adding and interacting with existing resources

1. In the **AWS Explorer**, right-click **Resources** and choose **Show Resources**.

   A pane displays a list of resource types that are available for selection.



2. In the selection pane, select the resource types to add to the **AWS Explorer** and press **Return** or choose **OK** to confirm.

   The resource types that you selected are listed under **Resources**.

   > **Note**
   >
   > If you've already added a resource type to the **AWS Explorer** and then clear the checkbox for that type, it's no longer listed under **Resources** after you choose **OK**. Only those resource types that are currently selected are visible in the **AWS Explorer**.

3. To view the resources that already exist for a resource type, expand the entry for that type.

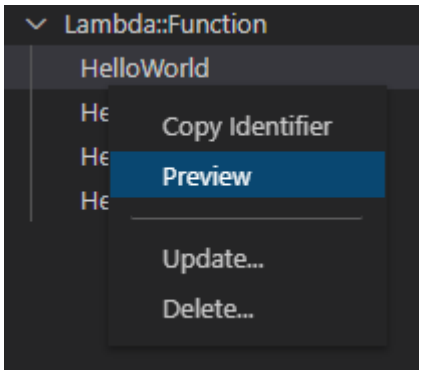
   A list of available resources is displayed under their resource type.

4. To interact with a specific resource, right-click its name and choose one of the following options:

- **Copy Resource Identifier**: Copy the identifier for the specific resource to the clipboard. (For example, the `AWS Toolkit:DynamoDB::Table` resource can be identified using the `TableName` property.)
- **Preview**: View a read-only version of the JSON-formatted template that describes the resource.

  After the resource template displays, you can modify it by choosing the **Update** icon at the right of editor tab. To update a resource, you must have the required ??? enabled.
- **Update**: Edit the JSON-formatted template for the resource in a VS Code editor. For more information, see Creating and editing resources.
- **Delete**: Delete the resource by confirming the deletion in a dialog box that is displayed. (Deleting resources is currently an ??? in this version of AWS Toolkit for Visual Studio Code.)

> **Warning**
>
> If you delete a resource, any AWS CloudFormation stack that uses that resource will fail to update. To fix this update failure, you need to either recreate the resource or remove the reference to it in the stack's CloudFormation template. For more information, see this Knowledge Center article.



Creating and editing resources

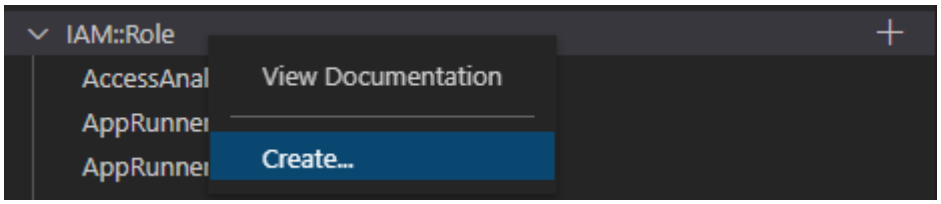
> **Important**
>
> The creation and updating of resources is currently an ??? in this version of the AWS Toolkit for Visual Studio Code.

Creating a new resource involves adding a resource type to the **Resources** list and then editing a JSON-formatted template that defines the resource, its properties, and its attributes.

For example, a resource that belongs to the `AWS Toolkit:SageMaker::UserProfile` resource type is defined with a template that creates a user profile for Amazon SageMaker AI Studio. The template that defines this user profile resource must conform to the resource type schema for `AWS Toolkit:SageMaker::UserProfile`. If the template doesn't comply with the schema because of missing or incorrect properties, for example, the resource can't be created or updated.

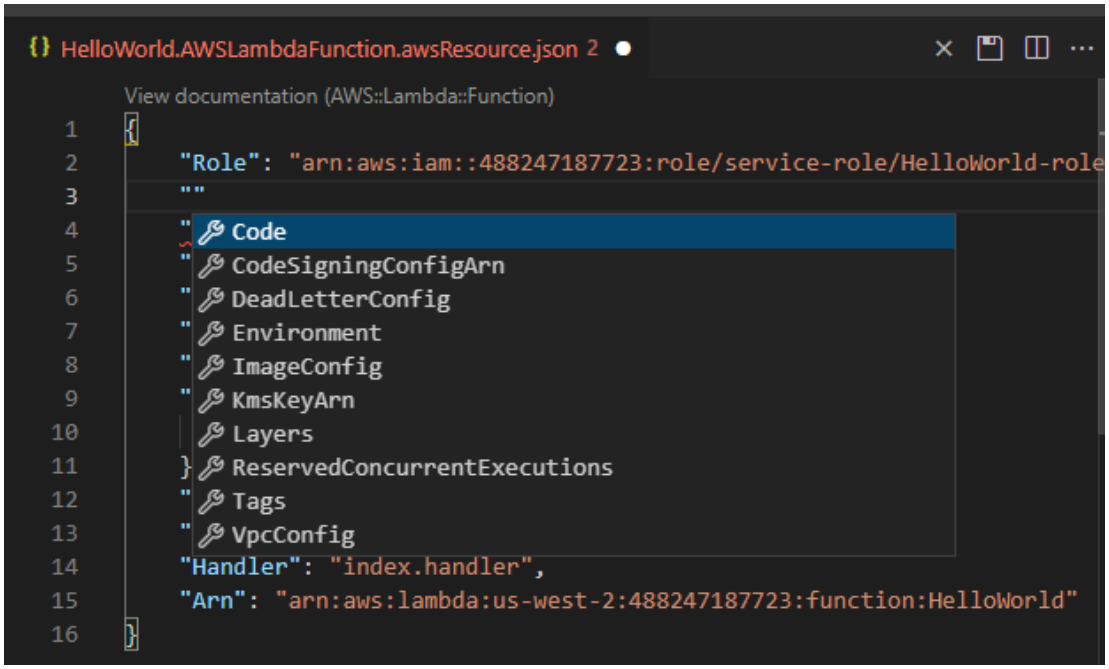
1. Add the resource type for the resource you want to create by right-clicking **Resources** and choosing **Show Resources**.
2. After the resource type is added under **Resources**, choose the plus ("+") icon to open the template file in a new editor.

   Alternatively, you can right-click the resource type's name and choose **Create**. You can also access information about how to model the resource by choosing **View Documentation**.

3. In the editor, start to define properties that make up the resource template. The autocomplete feature suggests property names that conform with your template's schema. When you hover over a property type, a pane displays a description of what it's used for. For detailed information about the schema, choose **View Documentation**.

   Any text that doesn't conform to the resource schema is indicated by a wavy red underline.



4. After you finish declaring your resource, choose the **Save** icon to validate your template and save the resource to the remote AWS Cloud.

   If your template defines the resource in accordance with its schema, a message displays to confirm that the resource was created. (If the resource already exists, the message confirms that the resource was updated.)

   After the resource is created, it's added to the list under the resource type heading.

5. If your file contains errors, a message displays to explain that the resource couldn't be created or updated. Choose **View Logs** to identify the template elements that you need to fix.

Troubleshooting the AWS Toolkit for Visual Studio Code

The following sections contain general troubleshooting information about the AWS Toolkit for Visual Studio Code and working with AWS services from the toolkit. For issues specifically related to troubleshooting SAM issues in the AWS Toolkit, see the Troubleshooting serverless applications topic in this User Guide.

Topics

Troubleshooting best practices

The following are recommended best practices when troubleshooting AWS Toolkit for Visual Studio Code issues. For detailed information about how you can contribute to the AWS Toolkit for Visual Studio Code, see the Contributing to AWS Toolkit for Visual Studio Code topic in the AWS Toolkit for Visual Studio Code GitHub repository.

- Attempt to recreate your issue or error prior to sending a report.
- Take detailed notes of each step, setting, and error message during the recreation process.
- Collect your AWS Toolkit Debug Logs. For a detailed description of how to locate your AWS Toolkit Debug logs, see the *How to locate your AWS logs* procedure, located in this user guide topic.
- Check for open requests, known solutions, or report your unresolved issue in the AWS Toolkit for Visual Studio Code Issues section of the AWS Toolkit for Visual Studio Code GitHub repository.

> **Note**
>
> The following procedure describes how to view your AWS Toolkit Debug logs. The process to view your Amazon Q Debug logs is identical except you choose **Amazon Q: View Logs** from the VS Code Command Palette.

How to locate your AWS Toolkit for Visual Studio Code Debug logs

1. From the VS Code open the Command Palette by pressing **`Cmd + Shift + P`** or **`Ctrl + Shift + P`** (Windows) and enter **`AWS View Logs`** into the search field.
2. Choose **AWS View Logs** to open your AWS Toolkit logs in the **VS Code terminal output** window.
3. From the **VS Code terminal output** window, expand the **Gear** icon menu and choose **Debug**.
4. Expand the **Gear** icon menu again and choose **Set As Default**.
5. Re-open the Command Palette by pressing **`Cmd + Shift + P`** or **`Ctrl + Shift + P`** (Windows) and search for **`Reload Window`**, then choose **Developer: Reload Window**.
6. VS Code reloads and the **VS Code terminal output** window displays your updated AWS Toolkit Debug logs.

Profile ... could not be found in the config file

Issue

> **Note**
>
> This issue only applies to the `~/.aws/config` file and not the `~/.aws/credentials` file. For detailed information about AWS config and AWS credentials files, see the Shared config and credentials files topic in the *AWS SDK and Tools* reference guide.

When choosing credentials AWS Toolkit logs display a message with this structure: `Profile` *`name`* `could not be found in shared credentials file`.

The following is an example of what this error looks like in your AWS Toolkit logs:

```
        2023-08-08 18:20:45 [ERROR]: _aws.auth.reauthenticate: Error: Unable to
authenticate connection
        -> CredentialsProviderError: Profile vscode-prod-readonly could not be found
in shared credentials file.
```

Solution

If your profile already exists in `~/.aws/config`, check that it starts with `[profile .` The following is an example of a user profile that is structured **correctly**:

```
[profile example]
region=us-west-2
credential_process=...
```

The following is an example of a user profile that is structured **incorrectly**:

```
[example]
region=us-west-2
credential_process=...
```

SAM json schema: cannot change schema in template.yaml file

Issue

You are unable to manually select a different json schema in SAM template.yaml

Solution

After updating to vscode-yaml version 1.11+, you can add a **`yaml-language-server`** modeline to the top of a YAML file to force the use of a schema by URI. For additional information about Using inlined schema section in the *yaml language server* topic of the *Redhat developer* GitHub repository. The following is an example of a **`yaml-language-server`** modeline.

```
# yaml-language-server: $schema=https://raw.githubusercontent.com/aws/
serverless-application-model/main/samtranslator/schema/schema.json
```

Security in AWS Toolkit for VS Code

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 VS Code, 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 VS Code. The following topics show you how to configure Toolkit for VS Code 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 VS Code resources.

Topics

Data protection in AWS Toolkit for VS Code

The AWS shared responsibility model applies to data protection in AWS Toolkit for VS Code. 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 Data Privacy FAQ. For information about data protection in Europe, see the General Data Protection Regulation (GDPR) Center.

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 VS Code 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.

Document history for the AWS Toolkit for Visual Studio Code User Guide

The following table describes important changes in each release of the AWS Toolkit for Visual Studio Code. For notification about updates to this documentation, you can subscribe to an RSS feed.

Change	Description	Date
Amazon SageMaker Unified Studio	Integration with the Amazon SageMaker Unified Studio service.	September 18, 2025
LocalStack	New User Guide topic added to support the launch of LocalStack.	September 11, 2025
Working with AWS Lambda functions	Updating user guide topic to include updated Lambda features in the toolkit.	July 17, 2025
AWS Lambda remote debugging	Added new AWS Lambda remote debugging topic to the AWS Toolkit for Visual Studio Code User Guide.	July 17, 2025
AWS Lambda Console to IDE	Added new AWS Lambda console to IDE topic to the AWS Toolkit for Visual Studio Code User Guide.	July 17, 2025
Updates to AWS Step Functions content and added support for Workflow Studio	Added updates to existing content for AWS Step Functions and User Guide topic for AWS Step Functions Workflow Studio, in support of feature launch.	March 6, 2025

AWS Serverless Land	Added new AWS Serverles s Land topic to the AWS Application Builder TOC.	March 6, 2025
Updating firewalls and gateways to allow access	Lists of endpoints and resources that must be allow listed to access all service and features in the AWS Toolkit for Visual Studio Code and Amazon Q for VS Code extensions.	February 28, 2025
Support for Amazon ECR App Runner	Added documentation support for launching an AWS App Runner service from the Amazon Elastic Container Registry node, in the AWS Toolkit.	February 6, 2025
Amazon DocumentDB	Added new Amazon DocumentDB topic to the AWS Toolkit for Visual Studio Code User Guide.	February 6, 2025
EC2 support	Support for the Amazon Elastic Compute Cloud service has been added to the toolkit.	January 31, 2025
AWS Documents	Added new user guide topic for AWS Documents.	January 20, 2025
Amazon CloudWatch Logs Live Tail	Added new subtopic to support the Amazon CloudWatch Logs Live Tail feature in the AWS Toolkit for Visual Studio Code.	December 15, 2024

AWS Application Builder	Added new AWS Applicati on Builder topic to the AWS Toolkit for Visual Studio Code User Guide.	October 30, 2024
Infrastructure Composer	AWS Application Composer is now AWS Infrastructure Composer.	October 3, 2024
AWS Identity and Access Management (IAM) Access Analyzer updates	Updated IAM Access Analyzer content to include new API references.	July 10, 2024
AWS Identity and Access Management (IAM) Access Analyzer	Added new user guide topic for IAM Access Analyzer.	May 23, 2024
Connect to AWS authorization flow updated	Authorization flow was updated to reflect changes to the auth process and the separation of Amazon Q from the AWS Toolkit for Visual Studio Code.	April 30, 2024
Amazon Q Extension for VS Code	As of April 30th 2024, CodeWhisperer is now part of Amazon Q and Amazon Q is available as an extension for VS Code.	April 30, 2024
Support for Virtual Private Cloud in Dev Environments	Updated content covering UI changes to support VPC in Dev Environments.	January 21, 2024
Infrastructure Composer	Added new Infrastructure Composer topic to the AWS Toolkit for Visual Studio Code User Guide.	November 28, 2023

SSO support for CodeCatalyst	Updated content to cover IAM Identity Center support for CodeCatalyst and Dev Environments.	November 17, 2023
Added VS Code and Toolkit download links	Updated content with download links for VS Code and the AWS Toolkit for Visual Studio Code.	November 1, 2023
Amazon Redshift topic	Added new Amazon Redshift topic to the AWS Toolkit for Visual Studio Code User Guide.	October 17, 2023
Connect to AWS authorization flow updated	Authorization flow updated to focus on service-specific authentication methods.	September 29, 2023
Created userguide: Create a CloudFormation template	Created a new userguide describing how to Create a CloudFormation template using the Toolkit for VS Code	December 17, 2021
Minor UI Update	Updated existing text for "Preview Machine State" to "Render graph" in order to better match the UI.	December 14, 2021
Created user guide for Amazon Elastic Container Service Exec	This is an overview of the Amazon ECS Exec.	December 13, 2021
Created user guide for the AWS IoT Toolkit for VS Code service	This user guide is intended to help you get started using the AWS IoT service for Toolkit for VS Code.	November 22, 2021

Support for experimental features	Added support for turning on experimental features for AWS services.	October 14, 2021
Support for AWS resources	Added support for accessing resource types along with interface options to create, edit, and delete resources.	October 14, 2021
Overview of the Amazon ECR service for AWS Toolkit for Visual Studio Code	Added an overview and walkthrough for the features and functions of the Amazon ECR service that are accessible in VS Code	October 14, 2021
Support for ARM64 environments	You can now run serverless applications in ARM64-based emulated environments as well as in x86_64-based environments.	October 1, 2021
AWS Serverless Application	Added support for running AWS SAM applications on ARM64 platform	September 30, 2021
Format update Node.js section	Per customer feedback, updated formatting for Node.js/ TypeScript.	August 12, 2021
App Runner support	Added support for AWS App Runner to AWS Toolkit for Visual Studio Code.	August 11, 2021
Debugging Go functions	Added support for debugging local Go functions.	May 10, 2021
Debugging Java functions	Added support for debugging local Java functions.	April 22, 2021

YAML support for AWS Step Functions	Added YAML support for AWS Step Functions.	March 4, 2021
Debugging Amazon API Gateway resources	Added support for debugging local Amazon API Gateway resources.	December 1, 2020
Amazon API Gateway	Added support for Amazon API Gateway.	December 1, 2020
AWS Serverless Application	Added support for Lambda container images with serverless applications.	December 1, 2020
AWS Systems Manager support	Added support for Systems Manager Automation documents.	September 30, 2020
CloudWatch Logs	Added support for CloudWatch Logs.	August 24, 2020
Amazon S3	Added support for Amazon S3.	July 30, 2020
AWS Step Functions support	Added support for AWS Step Functions.	March 31, 2020
Working with Amazon EventBridge Schemas	Added support for Amazon EventBridge Schemas	December 1, 2019
AWS CDK	Preview release of the AWS CDK service.	November 25, 2019
Using an external credential process	Added information about using an external credential process to obtain AWS credentials.	September 25, 2019

Using IntelliSense for task-definition files	IntelliSense support was added for working with Amazon ECS task Definition files.	September 24, 2019
User Guide for the AWS Toolkit for Visual Studio Code	Release for general availabil ity.	July 11, 2019
User Guide for the AWS Toolkit for Visual Studio Code	Updated the document structure for clarity and ease of use.	July 3, 2019
Installing the AWS Toolkit for Visual Studio Code	Added information about installing language SDKs to support various toolchains.	June 12, 2019
Configure your toolchain	Added information about configuring various toolchain s.	June 12, 2019
Initial Release	Initial release of the user guide for AWS Toolkit for Visual Studio Code.	March 28, 2019