AWS Toolkit for VS Code

User Guide

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

- Setting Up the AWS Toolkit for Visual Studio Code (p. 2)
- Navigating the AWS Toolkit for Visual Studio Code (p. 18)
- Working with AWS Services (p. 20)

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/.

Setting Up the AWS Toolkit for Visual Studio Code

This section helps you set up the AWS Toolkit for Visual Studio Code. It provides information about how to install and configure the toolkit, set up your credentials, and connect to AWS.

Topics

Installing the AWS Toolkit for Visual Studio Code

This section describes how to install the AWS Toolkit for Visual Studio Code.

Prerequisites

Required

Before you can install the Toolkit for VS Code, you must have the following:

- **An Amazon Web Services account** – 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). Signing up enables you to use all of the services that AWS offers.
- **A supported operating system** – The Toolkit for VS Code is supported on Windows, Linux, and macOS.
- **VS Code version 1.42.0 or later** – We try to keep the Toolkit for VS Code current with the default version that's available on the VS Code download page.

Optional

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

- **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) (p. 68).

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

Install the Toolkit for VS Code

1. Start the VS Code editor.

2. In the **Activity Bar** on the side of the VS Code editor, choose the **Extensions** icon. This opens the **Extensions** view, which allows you to access the VS Code Marketplace.
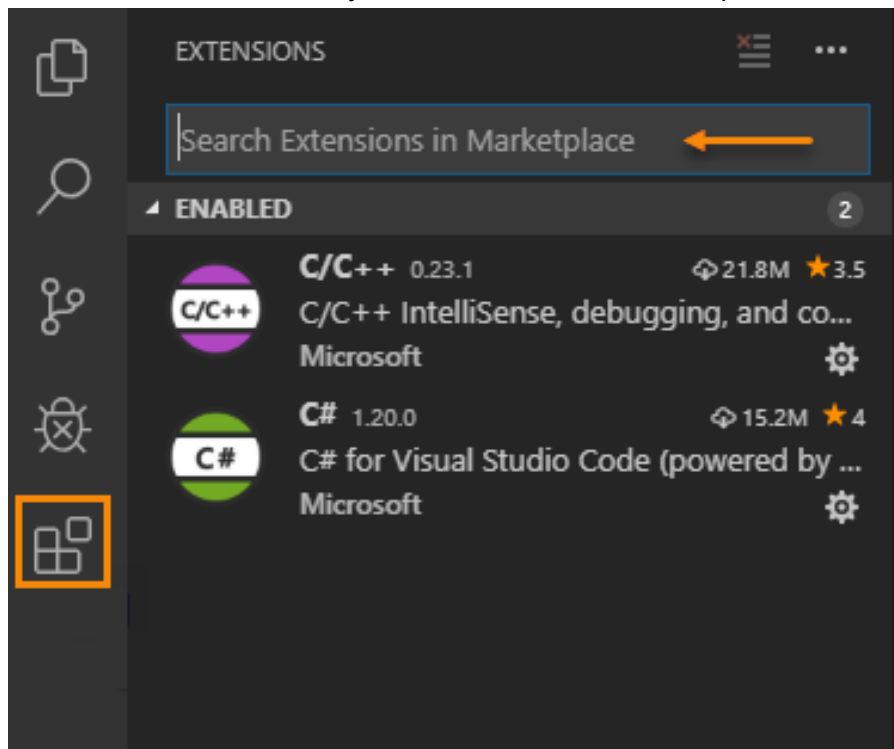
3. In the search box for **Extensions**, search for **AWS Toolkit**. Choose the entry to see its details in the right pane.

4. In the right pane, choose **Install**.

5. Once installed, if you're prompted to restart the editor, choose **Reload Required** to finish installation.

After you install the Toolkit for VS Code, you should configure your credentials (p. 4) to enable you to access your AWS resources from within VS Code.

Establishing credentials for the AWS Toolkit for Visual Studio Code

This section shows you the types of credentials that you can use with the AWS Toolkit for VS Code. It provides information about how to get and configure those credentials.

You can obtain credentials through AWS and provide them to the toolkit by using configuration files. You can also obtain credentials through an external credential process that isn't directly supported by AWS.

Topics

Using AWS-supported credentials

AWS credentials can be provided to the AWS Toolkit for VS Code by using your *shared AWS config file* or your *shared AWS credentials file*. The methods for using these files are the same as those for the AWS CLI. For general information about how to use these files, see Configuration and Credential Files in the *AWS Command Line Interface User Guide*.

Topics

Obtaining AWS access keys

To access Amazon Web Services (AWS) with the AWS Toolkit for Visual Studio Code, you must configure the toolkit with AWS account credentials. To do this with AWS-supported credentials, you must first obtain appropriate AWS access keys.

For more information about users and credentials that is out of scope for this guide, see the following resources:

- AWS Security Credentials in the *Amazon Web Services General Reference*
- Overview of Identity Management: Users in the *IAM User Guide*

What are AWS access keys

Access keys are the credentials that identify you to AWS and enable you to programmatically access AWS services and resources. Access keys can be associated with your AWS account (the account's "root user") or with users that you create with AWS Identity and Access Management (IAM).

> **Warning**
> Because the root user is essentially an administrator with full access to services and resources, we recommend that you instead create an IAM user with only those permissions needed to perform the required tasks. Then, for your credentials, you can use an access key that is associated with that user. For details, see Creating an IAM User in Your AWS Account in the *IAM User Guide*.

An access key consists of an *access key ID*, which is similar to a user name, and a *secret access key*, which is similar to a password. This access key is used to sign programmatic requests that you make to AWS. If

you don't have access keys, you can create them by using the AWS Management Console. We recommend that you use access keys for an IAM user instead of the keys for your account's root user.

> **Note**
> To create access keys, you must have permissions to perform the required IAM actions. For more information, see Granting IAM User Permissions to Manage Password Policy and Credentials in the *IAM User Guide*.

Get your AWS access keys

1. Open the IAM console at https://console.aws.amazon.com/iam/.

2. On the navigation menu, choose **Users**.

3. Choose your IAM user name (not the check box) to view its details.

4. On the **Security Credentials** tab, choose **Create access key**.

5. To see the new access key, choose **Show**. The credentials resemble the following:

   - Access key ID: AKIAIOSFODNN7EXAMPLE

   - Secret access key: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

6. To download the key pair, choose **Download .csv file**. Store the keys in a secure location.

   > **Important**
   >
   > - Keep the keys confidential to protect your AWS account, and never email them. Do not share them outside of your organization, even if an inquiry appears to come from AWS or Amazon.com. *No one who legitimately represents Amazon will ever ask you for your secret key*.
   >
   > - You can't recover the secret key if you lose it. Instead, for security reasons, you must create a new key pair and delete the old pair.

After you have obtained your AWS access keys, you can use the AWS Toolkit for VS Code to store them in your shared AWS config file or your shared AWS credentials file. See Add your AWS access keys to your environment (p. 6) to learn how.

Setting up your AWS credentials

To access Amazon Web Services (AWS) with the AWS Toolkit for Visual Studio Code, you must make your AWS account credentials available to the toolkit. To use AWS-supported credentials, continue reading in this topic. To use an external credential process, see Using an external credential process (p. 9).

> **Note**
> Some features of the Toolkit for VS Code, such as creating a serverless application, don't require AWS credentials.

Get your AWS access keys

If you don't already have appropriate AWS access keys to store in your shared AWS config file or your shared AWS credentials file, you must get them now.

To do so, see Obtaining AWS access keys (p. 4).

About shared AWS files

Your *shared AWS config file* and your *shared AWS credentials file* are files that you can use to store configuration and credential information for AWS. By default, these files are located in the `.aws` directory within your home directory and are named `config` and `credentials`, respectively. For more

information, see Where Are Configuration Settings Stored? in the *AWS Command Line Interface User Guide*.

The Toolkit for VS Code locates and uses AWS access keys through your shared AWS config file and your shared AWS credentials file. This is the method that is used by the AWS CLI and the AWS SDKs. Access keys that you enter in the Toolkit for VS Code are saved to one of these files.

These shared files can contain the credentials for more than one AWS account, stored as *profiles*. Multiple accounts can be useful, for example, to provide developers and administrators with separate resources for development and for release or publication.

Add your AWS access keys to your environment

If you have already set your AWS credentials (for example, by using the AWS CLI), the Toolkit for VS Code will automatically detect those credentials and make them available to the toolkit. If you haven't already set your AWS credentials, or if you want to include additional AWS credentials in your environment or update an existing credentials profile, you can do so through the Toolkit for VS Code, as shown here.

> **Note**
> As an alternative to these procedures, you can use the `AWS configure` AWS CLI command to add AWS credentials to your environment. You can also use `AWS configure` to set the default AWS Region, which is needed for certain operations such as creating a serverless application.

Create the shared AWS credentials file

- If you already have a shared AWS credentials file, you can skip to the next procedure (p. 7).
- If you already have a shared AWS *config* file and want to use it, you can skip to the next procedure (p. 7).
- If you have only a shared AWS *config* file but do NOT want to use it, you must first create a shared AWS credentials file by using techniques that are normal for your operating-system. After that, you can skip to the next procedure (p. 7).

Follow these steps to create the shared AWS credentials file.

1. Open VS Code.
2. To open the **Command Palette**, on the menu bar, choose **View**, **Command Palette**. Or use the following shortcut keys:

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

3. Search for **AWS** and choose **AWS Toolkit Create Credentials Profile**.
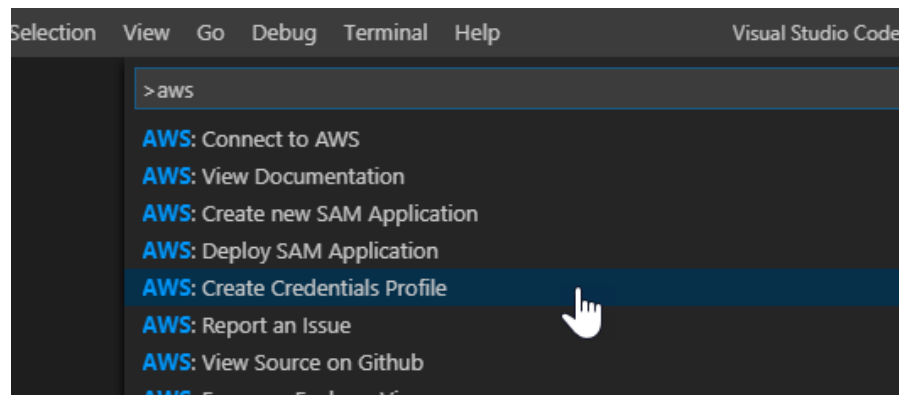
4. Enter a name for the initial profile.

Create a new AWS credential profile (1/3)

default

Choose a unique name for the new profile (Press 'Enter' to confirm or 'Escape' to cancel)
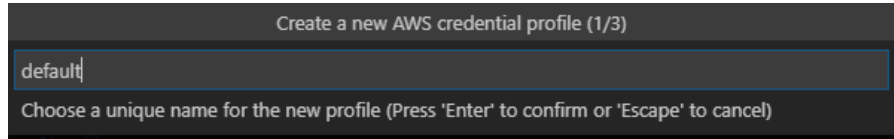
5. Enter the *access key ID* for the credentials. If you don't have an access key ID, see Obtaining AWS access keys (p. 4).

Create a new AWS credential profile (2/3)

YOUR_ACCESS_KEY_ID

Input the AWS Access Key (Press 'Enter' to confirm or 'Escape' to cancel)

6. Enter the *secret access key* for the credentials. If you don't have a secret access key, see Obtaining AWS access keys (p. 4).

Create a new AWS credential profile (3/3)

YOUR_SECRET_KEY

Input the AWS Secret Key (Press 'Enter' to confirm or 'Escape' to cancel)

After you complete this procedure, you can verify the shared AWS credentials file by opening it directly or by performing the first three steps of the next procedure (without making any updates).

Update one of your shared files

Follow these steps to add a new profile to your existing shared AWS config file or shared AWS credentials file. You can also update an existing profile.

1. Open VS Code.

2. To open the **Command Palette**, on the menu bar, choose **View**, **Command Palette**. Or use the following shortcut keys:

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

   - macOS – Press **Shift+Command+P**.

3. Search for **AWS** and choose **AWS Toolkit Create Credentials Profile**.

Selection   View   Go   Debug   Terminal   Help   Visual Studio Code

> aws

AWS: Connect to AWS
AWS: View Documentation
AWS: Create new SAM Application
AWS: Deploy SAM Application
AWS: Create Credentials Profile
AWS: Report an Issue
AWS: View Source on Github
AWS: Focus on Explorer View
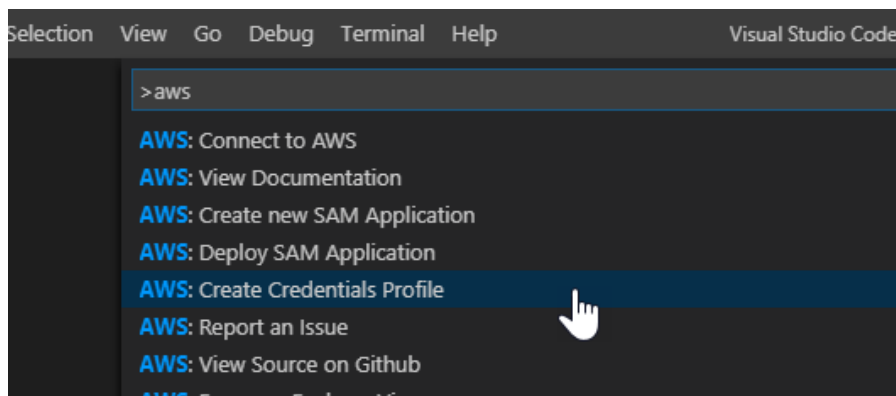
4. When one or both of the shared files opens in the VS Code editor, add or update a profile.

5. When you're finished updating the file, save it.

Add additional credential profiles

You can add additional profiles and credentials. 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 in the example below:

```
# 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
```

Using IAM Identity Center credentials

To connect with AWS IAM Identity Center (successor to AWS Single Sign-On), you must complete the following prerequisites:

1. **Enable IAM Identity Center** – This includes choosing your identity source and setting up IAM Identity Center access to your AWS accounts. For more information, see Getting started in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.
2. **Add an IAM Identity Center profile** – With IAM Identity Center, you define a named profile in the credentials file or config that you use to retrieve temporary credentials for your AWS account. The profile definition specifies the AWS access portal as well as the AWS account and IAM role associated with the user requesting access.

To add an IAM Identity Center profile

The following procedure outlines how to add an IAM Identity Center profile to your `credentials` or `config` file.

Adding an IAM Identity Center profile to your credentials file in VS Code
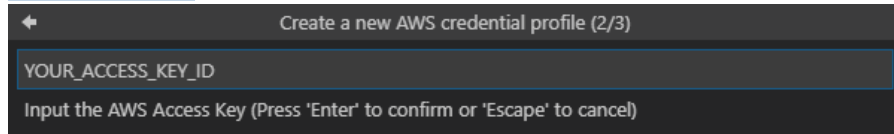
1. Open VS Code.
2. To open the **Command Palette**, on the menu bar, choose **View**, **Command Palette**. Or use the following shortcut keys:

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

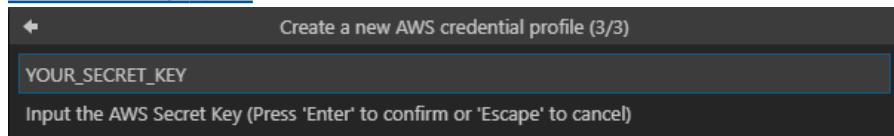3. Search for **AWS** and choose **AWS Toolkit Create Credentials Profile**. This will open the credentials file.

4. In the either the `credentials` or `config` file, under `[default]`, add a template for a named IAM Identity Center profile. An example profile follows:

```
... Named profile in credentials file ...

[profile sso-user-1]
sso_start_url = https://example.com/start
sso_region = us-east-2
sso_account_id = 123456789011
sso_role_name = readOnly
region = us-west-2
```

> **Important**
> Do not use the word *profile* when creating an entry in the credentials file. This is because the credentials file uses a different naming format than the config file. Include the prefix word `profile` only when configuring a named profile in the config file.

When assigning values for your profile, keep the following in mind:

- `sso_start_url` – The URL that points to your organization's AWS access portal.
- `sso_region` – The AWS Region that contains your AWS access portal host. This can be different from the AWS Region specified later in the default `region` parameter.
- `sso_account_id` – The AWS account ID that contains the IAM role with the permission that you want to grant to this IAM Identity Center user.
- `sso_role_name` – The name of the IAM role that defines the user's permissions when using this profile to get credentials through IAM Identity Center.
- `region` – The default AWS Region that this IAM Identity Center user will sign into.

Signing in with IAM Identity Center

When signing in with an IAM Identity Center profile, the default browser is launched to the specified portal. You must verify your IAM Identity Center login before you can access your AWS resources in VS Code. Note that if your credentials expire, you'll have to repeat the connection process to obtain new temporary credentials.

Using an external credential process

If you have a process to generate or lookup credentials that isn't directly supported by AWS, you can configure the AWS Toolkit for VS Code to use that process instead of any stored AWS credentials (p. 4).

The method for specifying such an external credential process is the same as for the AWS CLI, and consists of adding a `credential_process` definition to your *shared AWS config file*. For detailed information about how to do this, see Sourcing Credentials with an External Process in the *AWS Command Line Interface User Guide*.

Connecting to AWS through the AWS Toolkit for Visual Studio Code

To interact with Amazon Web Services (AWS) through the AWS Toolkit for Visual Studio Code, you must establish a connection to AWS.

Connect to AWS through the Toolkit for VS Code

1. Open VS Code.

2. To open the **Command Palette**, on the menu bar, choose **View**, **Command Palette**. Or use the following shortcut keys:

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

   - macOS – Press **Shift+Command+P**.

3. Search for **AWS** and choose **AWS Toolkit Connect to AWS**.
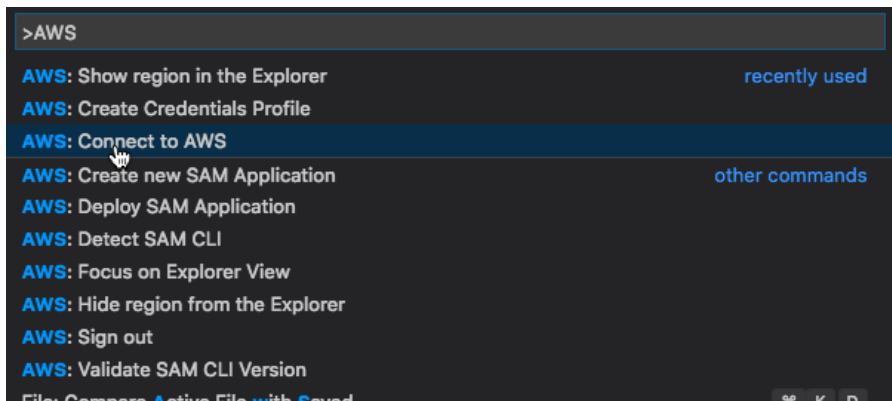
4. Choose a profile from the list.
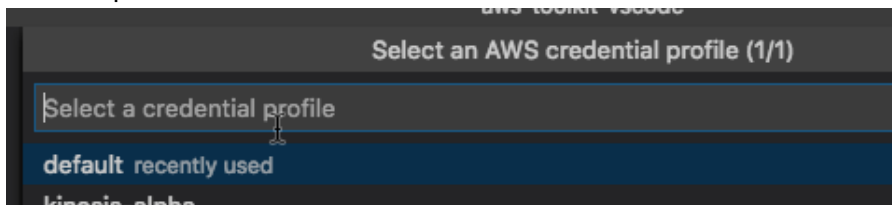
   If you don't have a credentials profile set up, you are prompted to set one up. Look for a pop-up in the lower-right corner of the editor. Choose **Yes**, and then follow the setup wizard to enter a profile name, your access key ID, and your secret access key. For details, see Setting up your AWS credentials (p. 5).

   > **Note**
   > If you want to provide an external credential process instead of using AWS-supported credentials, choose **No** and see Using an external credential process (p. 9) instead.
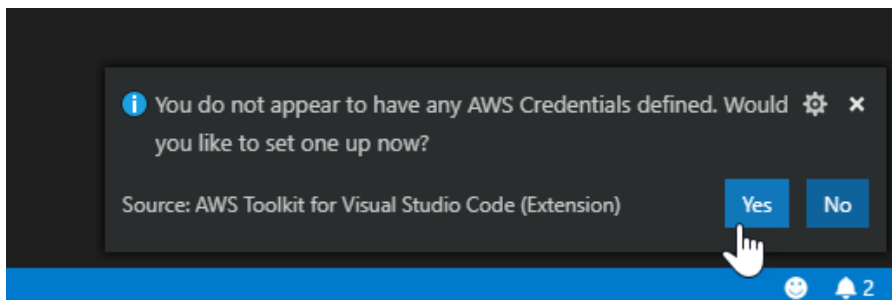
5. Open the **AWS Toolkit Explorer** Side Bar, which we call the *AWS Explorer*, to verify the connection. You will see either a list of AWS Regions (if you have made any Regions visible (p. 13) in the **AWS Explorer**) or a message to add Regions to the **AWS Explorer**.

   Before adding Regions (p. 12) to the **AWS Explorer**, you see the following.
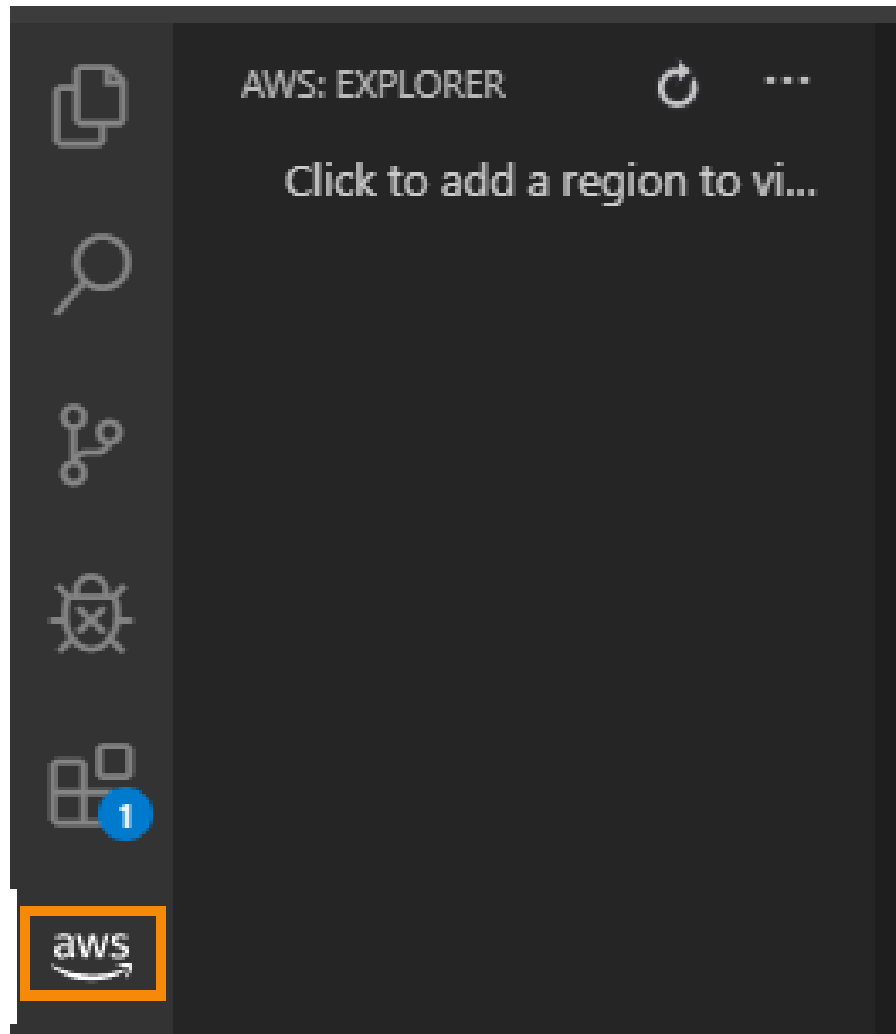
After adding Regions to the **AWS Explorer**, you see something like the following.
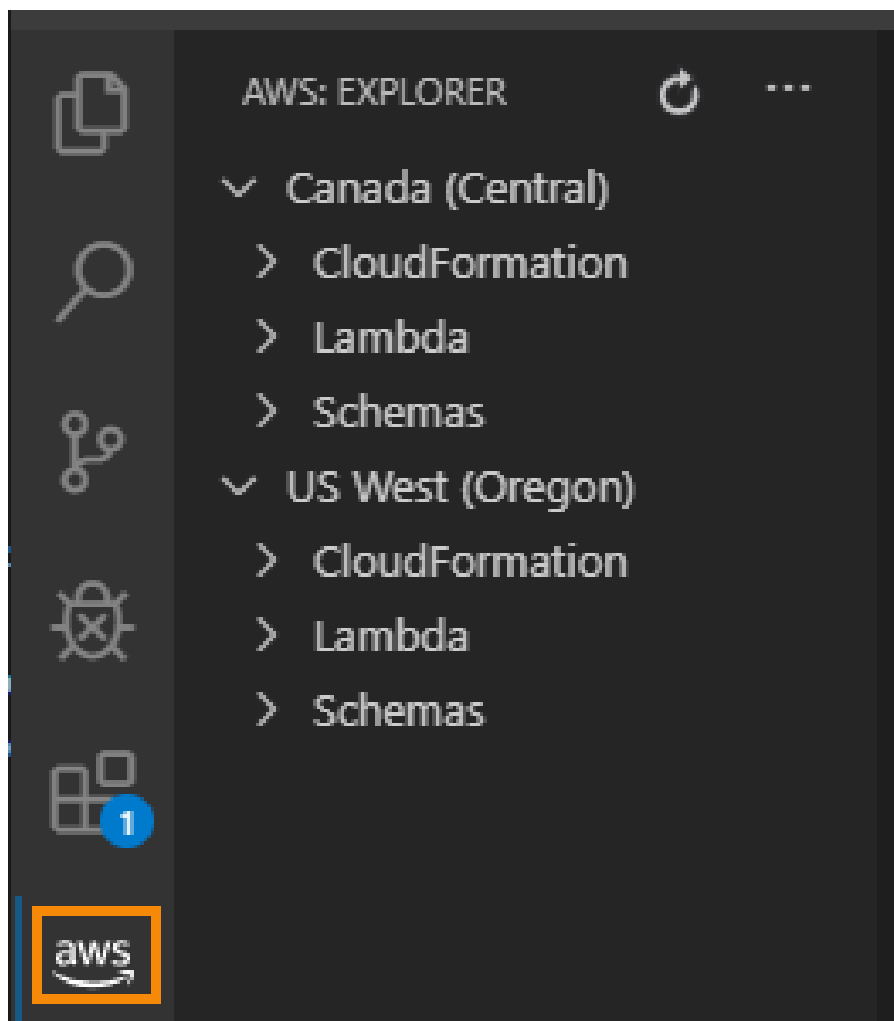
Use multiple AWS accounts with the Toolkit

You might have multiple AWS accounts that you want to access from the Toolkit for VS Code. Multiple accounts can be useful, for example, to provide developers and administrators with separate resources for development and for release or publication.

Separate sets of AWS credentials are stored as *profiles* within the shared AWS config file or the shared AWS credentials file. To choose a different set of credentials, follow the steps in the previous procedure, and choose a different profile.

Changing AWS Regions

When you set up your credentials, the AWS Toolkit for Visual Studio Code automatically selects and shows the default AWS Region for those credentials in the **AWS Explorer**. This topic describes how to change the list of Regions that is shown in the **AWS Explorer**.

Add a Region to the AWS Explorer

1. To open the **Command Palette**, on the menu bar, choose **View**, **Command Palette**. Or use the following shortcut keys:

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

   - macOS – Press **Shift+Command+P**.

2. Search for **AWS** and choose **AWS Toolkit Show Region in the Explorer**.
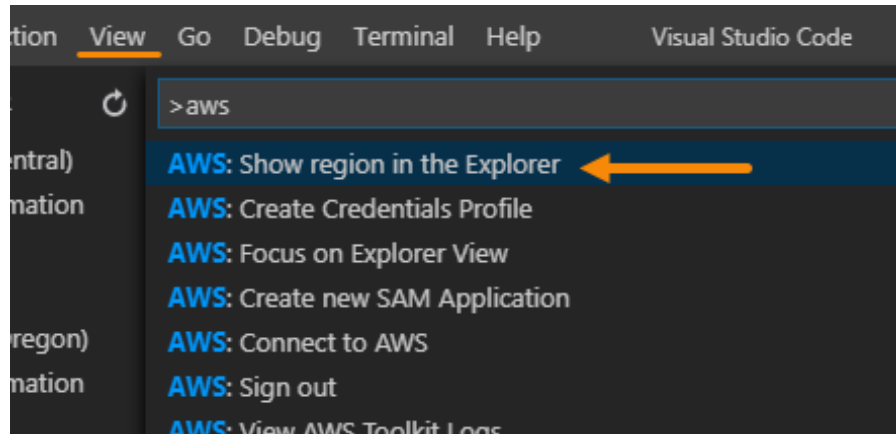
3. Choose the Region that you want to add to the **AWS Explorer**.
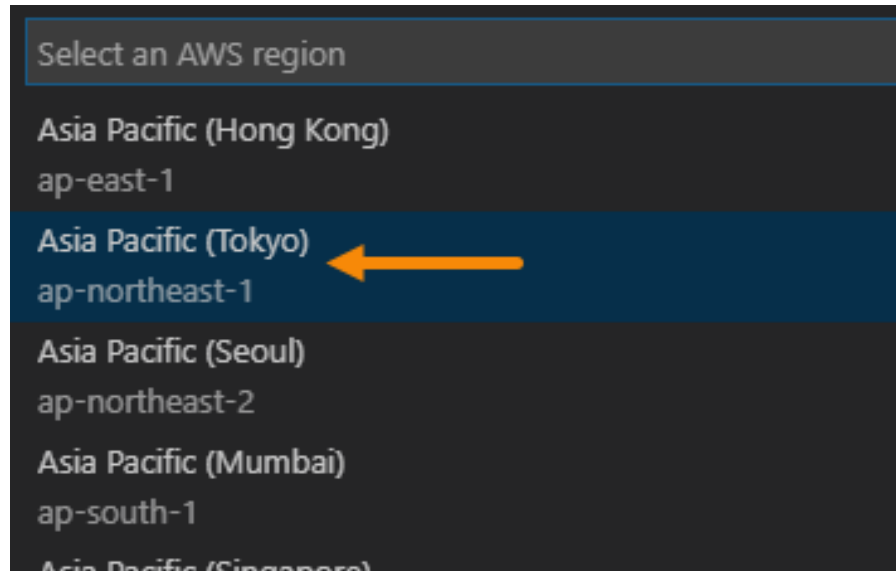
   > **Note**
   > The list contains only those Regions that are available to add to the **AWS Explorer**. Regions you've already added don't appear in the list.

4. Verify that the Region now appears in the **AWS Explorer**.
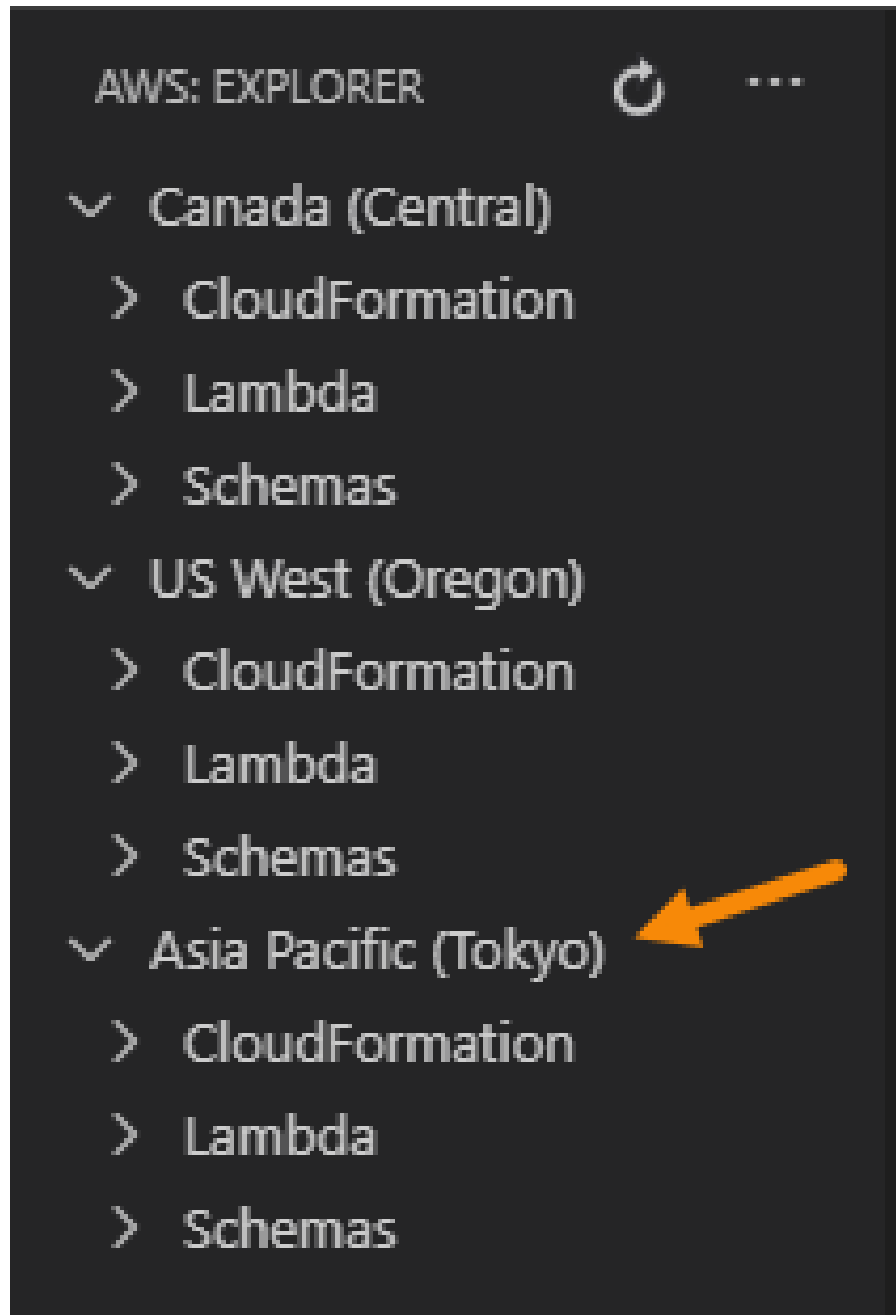
Hide a Region from the AWS Explorer

1. Choose the AWS icon in the **Activity bar** to open the **AWS Explorer**.
2. Choose one of the Regions in the list, and open its context menu.
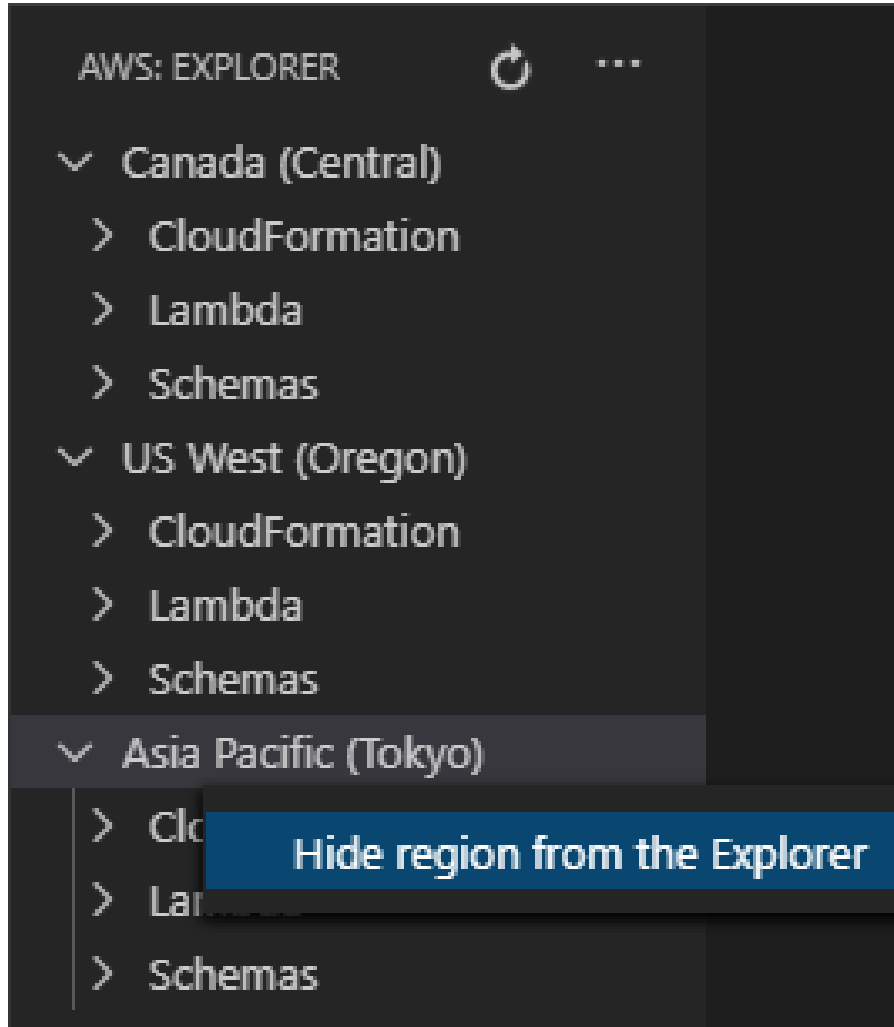
3. Choose **Hide Region from the Explorer**.

Configuring your toolchain

The AWS Toolkit for Visual Studio Code supports multiple languages that you can use to interact with AWS. This topic describes how to set up the toolchain for each of these languages.

Configure a toolchain for .NET Core

1. Ensure that you have the AWS Toolkit for VS Code installed (p. 3).
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 (p. 68).
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 (p. 3).

2. Open an AWS SAM application, or create one (p. 68).

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 (p. 3).

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 (p. 68).

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:

    - Bash shell: `./.venv/Scripts/activate`
    - PowerShell: `./.venv/Scripts/Activate.ps1`

Configure a toolchain for Java

1. Ensure that you have the AWS Toolkit for VS Code installed (p. 3).

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 (p. 68).

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 (p. 3).

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 pallete, choose `Go: Install/Update Tools`.

    b. From the set of checkboxes, select `dlv` and `gopls`.

5. Open an AWS SAM application, or create one (p. 68).

6. Open the folder that contains `template.yaml`.

Using Your toolchain

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

Navigating the AWS Toolkit for Visual Studio Code

This topic describes how to navigate in the AWS Toolkit for Visual Studio Code. Be sure to first install the toolkit (p. 2) before reading this topic.

For general information about the Visual Studio Code user interface (UI), see the UI documentation for VS Code.

Fundamental UI Components

The Toolkit for VS Code is available from your VS Code **Activity Bar**. To open the Toolkit for VS Code AWS navigation pane in your **VS Code Explorer**, choose the **AWS icon** from the VS Code **Activity Bar**. The AWS navigation pane contains two main section headings:

- The **EXPLORER** menu heading, which expands the **AWS Explorer**.
- The **CDK (PREVIEW)** menu heading, which expands the **AWS Cloud Development Kit (CDK) Explorer**.
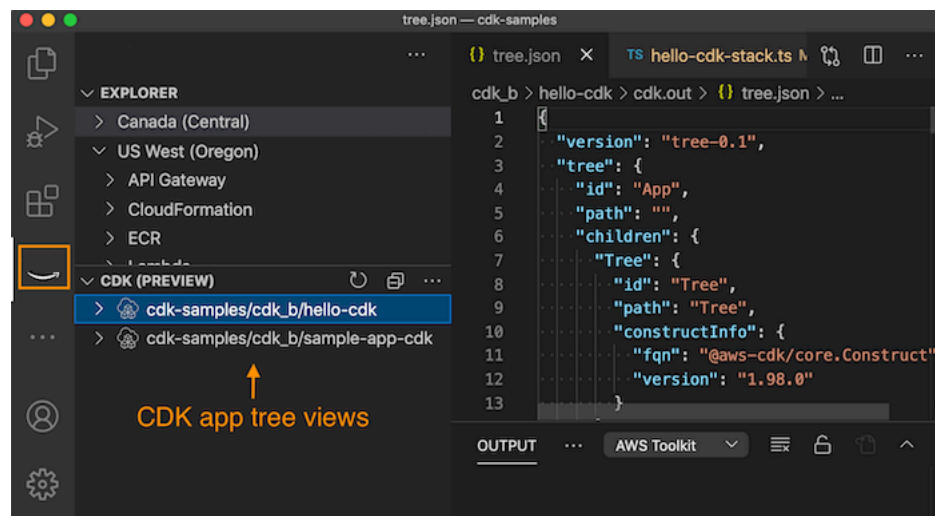
The AWS Explorer

The **AWS Explorer** is the main navigation pane for the Toolkit.

Your AWS services and resources are available through the AWS Explorer and are organized by region.

1. From your **AWS Explorer**, choose the **EXPLORER** menu heading to expand the Region selector navigation pane.
2. Choose a **region** to expand a list of your AWS services associated with that region.
3. Choose an **AWS service** to view its associated features and resources.

For additional information about the AWS Explorer, the **Region selector**, and the AWS services available from the Toolkit, see the user guide Working with AWS Services in the AWS Explorer (p. 21).

The AWS CDK Explorer

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

The **AWS CDK Explorer** is a navigation pane which displays your AWS CDK applications. For detailed information about the AWS CDK Explorer and features, see the user guide AWS CDK for VS Code (p. 35).

Toolkit for VS Code commands

The Toolkit for VS Code has its own set of VS Code commands. You can view a list of these commands from the VS Code Command Pallette.

Visual Studio Code Command Palette

The VS Code Command Palette is a reference tool that contains all of the VS Code commands currently associated with your instance of VS Code. The Command Palette includes all of the default commands that were installed with the current version of your VS Code IDE, as well as any additional commands that are associated with any VS Code extensions that you've installed.

TO view a list of the commands that are exclusive to your instance of the Toolkit for VS Code, you can use a search filter from the VS Code Command Palette. **View** menu. Or use the following shortcuts:

1. From the VS Code **Main Menu** choose the **View** heading.
2. Choose a **Command Palattte** to open the Command Palattte menu.
3. From the input field enter the text: **AWS**. As you type the text, the Command Palattte starts filtering and displays all of the commands associated with your search.

You can also open the Command Palattte by keyboard shortcut.

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

AWS Explorer Menu

The **AWS Explorer** has a **More Actions** menu that contains the most common commands for working with AWS services, as well as information about the current version of your Toolkit.

To open the **More Actions** menu, start from the AWS navigation pane.

1. Choose the **EXPLORER** heading, to expand the AWS Explorer.
2. Choose the **. . .** icon, located next to the expanded **EXPLORER** heading, to expand the **More Actions** menu.

Working with AWS Services

The AWS Toolkit for Visual Studio Code makes AWS services and resources available to you, directly in the VS Code UI. The following is a list of guide topics for each Toolkit for VS Code service and their features. Choose a service for more information on what that service does, how to set it up, and ways work with its basic features.

Topics

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. See a high-level view of the **AWS Explorer** in the topic.

This section provides information about how to access and use the **AWS Explorer** in VS Code. It assumes that you've already 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:
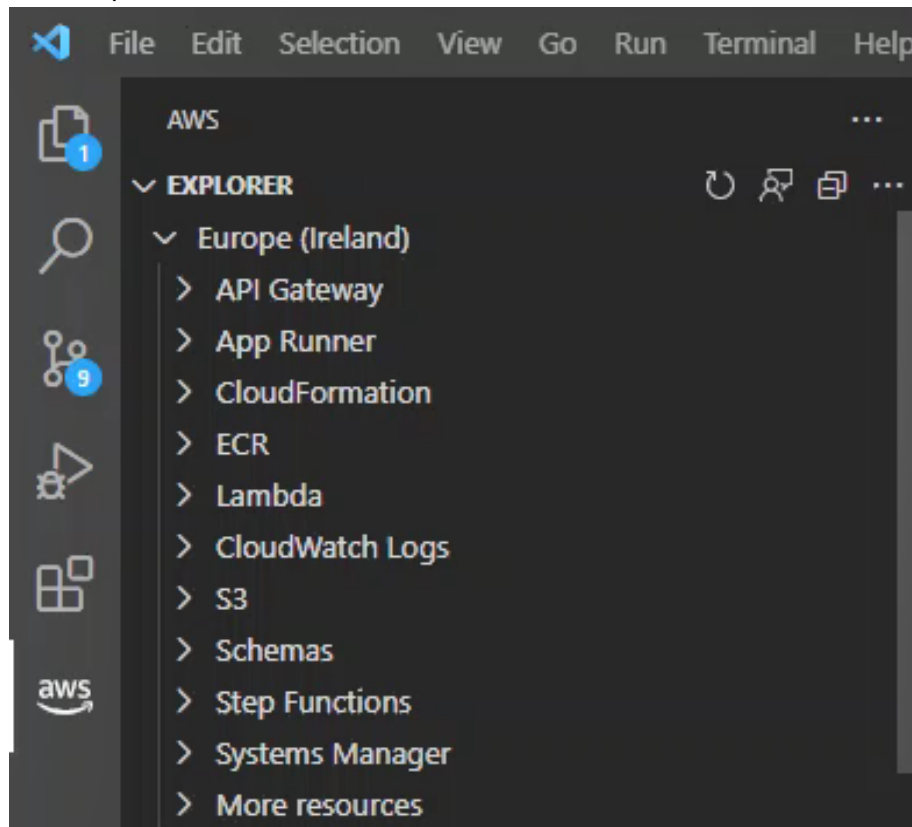
- Certain features require certain AWS permissions. For example, to see the AWS Lambda functions in your AWS account, the credentials you configured in must include at least read-only Lambda permissions. See the following topics for more information about the permissions that each feature needs.
- If 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.

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

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

Topics

Installing VS Code

Before you begin, make sure that you're using the latest version of VS Code. To download the latest version of VS Code, see Download VS Code on the VS Code website.

Installing the Toolkit for VS Code

Install the latest version of the Toolkit for VS Code extension to connect the Toolkit for VS Code with your CodeCatalyst account. You can find the latest version and complete the AWS Toolkit installation from the VS Code **Extensions: Marketplace**.

To open the VS Code **Extensions: Marketplace**, and install the Toolkit for VS Code, complete the following steps:

1. From the VS Code **Activity Bar** located at the left edge of the main screen, select the **Extensions** icon to open the **EXTENSIONS: MARKETPLACE** view.

Note
You can also use the VS Code **Command Palette** to open the Extensions: Marketplace view. To do this, open the command palette by pressing `Command+Shift+P` (macOS) or `Ctrl +Shift+P` (Windows). Then, enter `Extensions: Focus on Marketplace View` into the provided search field and choose **Extensions: Focus on Marketplace View** to open the marketplace view.

2. From the **EXTENSIONS: MARKET PLACE** search bar, enter `AWS Toolkit`.

3. The search results filter as you type, when **AWS Toolkit extensions** is available, choose **Install** to begin installing the latest version of the AWS Toolkit.

Creating a CodeCatalyst account

In addition to installing the latest version of the AWS Toolkit, you must have an active AWS Builder ID to connect CodeCatalyst with the Toolkit for VS Code. If you don't have an active AWS Builder ID or CodeCatalyst account, see the [Setting up with CodeCatalyst](#) section in the *CodeCatalyst* User Guide.

Note
An AWS Builder ID is different from your IAM Credentials. IAM Credentials are required for most AWS services that are accessible from the AWS Toolkit. In contrast, an AWS Builder ID is required to create a new CodeCatalyst account and connect it with the AWS Toolkit.

Connecting the AWS Toolkit with CodeCatalyst

To connect the AWS Toolkit with your CodeCatalyst account, complete the following steps:

1. From the Toolkit for VS Code, expand the **DEVELOPER TOOLS** explorer.

2. From the **DEVELOPER TOOLS** explorer, expand **CodeCatalyst** and choose **Start**.

3. When prompted by VS Code, open the AWS Builder ID portal in your preferred web browser to continue.

   Note
   If you do not have an AWS Builder ID, use the provided link to create one. Then, complete the following steps to authenticate and connect your AWS Builder ID credentials to the Toolkit for VS Code.

4. From the AWS Builder ID portal, follow the steps to authenticate your AWS Builder ID. When prompted, choose to **Allow** the Toolkit for VS Code to access your data.

5. Your browser updates automatically when your AWS Builder ID credentials have successfully connected with the Toolkit for VS Code, indicating that it's safe to close your browser and return to the Toolkit for VS Code.

   Note
   If your Toolkit for VS Code is already authenticated with your IAM Credentials, VS Code opens the following prompt:
   Some tools you've been using don't work with AWS Builder ID. Keep using AWS Builder ID in the background while using AWS Credentials?

   - To proceed using your AWS Builder ID to authenticate the CodeCatalyst service and your IAM Credentials to authenticate your other Toolkit for VS Code services, choose the **Yes, keep using AWS Builder ID with CodeCatalyst while using IAM Credentials with other services.**

   - To proceed using only your AWS Builder ID to authenticate all Toolkit for VS Code services choose the **No, switch everything to authenticate with AWS Builder ID**

6. When your AWS Builder ID has successfully connected with the Toolkit for VS Code, the **DEVELOPER TOOLS** explorer updates the features for CodeCatalyst and indicates a **AWS Builder ID Connected** status.

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

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:

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.

   > **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.

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

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.

   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.

Creating a CodeCatalyst Dev Environment

To create a new 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.

   Note
   Though not required, we recommend that you specify an alias because it simplifies the organization and search of your Dev Environments.
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.

     Note
     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**: Choose **Edit Storage Size** to change the amount of storage space that's assigned to your system.

     Note
     Storage is the only setting that can't be changed after your Dev Environment has been created.
6. Choose **Create Dev Environment** to create your new cloud development environment.

   Note
   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.

CodeCatalyst commands in VS Code

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

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

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 in Dev Environments and VS Code

You can launch a VS Code virtual computing environment, known as a Dev Environment, from the Toolkit for VS Code. Dev Environments are customizable cloud-development environments that you can copy and share among different team members in your Space.

The following sections describe how to configure your Dev Environments locally and remotely, as well as basic features that are accessible from the Toolkit for VS Code when it's launched in a Dev Environment. For more information about Dev Environments, see Dev Environments in the *CodeCatalyst* guide.

Topics
- Configuring your Dev Environment with devfiles (p. 26)
- Connecting AWS IAM Credentials to your Dev Environment (p. 27)
- Working with the Toolkit for VS Code in Dev Environments (p. 27)

Configuring your Dev Environment with devfiles

You can define the project-specific tools and application libraries that are installed to your Dev Environment in the `devfile` of your Dev Environment. The `devfile` specification is an open standard that your write into a YAML document.

You can edit your `devfile` locally and remotely in VS Code. The process is similar to creating or editing any VS Code compatible file. You can also edit the `Devfile` directly in CodeCatalyst, from the source repository of your project. For instructions on how to edit your devfile from CodeCatalyst, see the Configuring your Dev Environment section of the *CodeCatalyst* guide.

> **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.

To edit your `Devfile` in a local instance of VS Code:

1. From **VS Code Explorer**, locate the `devfile.yaml` file, and open it in the VS Code editor.
2. Make your changes to the document, then save it.

3. Commit your changes, then push the changes to your source repository.

4. The next time you launch your Dev Environment, the configuration is updated to match the specifications that are defined in your `Devfile`.

Connecting AWS IAM Credentials to your Dev Environment

In order to access all Toolkit for VS Code services in a Dev Environment, you need to connect AWS IAM Credentials to your Dev Environment. To connect IAM credentials to your Dev Environment, complete the following:

1. From the Toolkit for VS Code **EXPLORER** in your Dev Environment, choose **Select IAM Credentials to View Resources** to open the **Switch Connection** dialog.

2. Choose the IAM credentials you want to use from the list or choose **Add New Connection** to add new credentials.

3. Choose from these options and follow the prompts to create a new connection:

   - **Use a personal email to sign up and sign in with AWS Builder ID**: Creates a new AWS Builder ID for personal use. This option only supports the use of AWS Builder ID exclusive services such as CodeCatalyst.

   - **Connect using AWS IAM Identity Center**: Allows you to sign in to your company's IAM access portal. This option is only available through a company that has IAM Identity Center access. The AWS resources and services that are accessible from this account are managed by the company that owns the account.

   - **Use IAM Credentials**: Allows you to work with your AWS services and resources from the Toolkit for VS Code. Using this option alone doesn't support the use of AWS Builder ID exclusive services, such as CodeCatalyst.

4. After your IAM Credentials are successfully connected with the Toolkit for VS Code in your Dev Environment, the **AWS EXPLORER** updates with your AWS services and resources.

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. When your Dev Environment has successfully been stopped, 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:

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.

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

Configuring your Dev Environment

To open your Dev Environment `devfile` and configure your current Dev Environment, complete the following steps. For more detailed information about editing devfiles, see the resources listed in the Configuring your Dev Environment with devfiles section of this guide.

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.

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 (p. 32)
- Managing App Runner services (p. 33)

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 (p. 30). 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

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 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 AWS 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 (p. 2) 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 (p. 2).

- 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 turotial 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 AWS CloudFormation stack (p. 37)
- Create a AWS CloudFormation template using the AWS Toolkit for Visual Studio Code (p. 38)

Deleting an AWS CloudFormation stack

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

Prerequisites

- Be sure your system meets the the prerequisites specified in Installing the Toolkit for VS Code (p. 2).
- Ensure that the credentials you configured in Establishing credentials (p. 4) include appropriate read/write access to the AWS 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 AWS CloudFormation stack you want to delete.
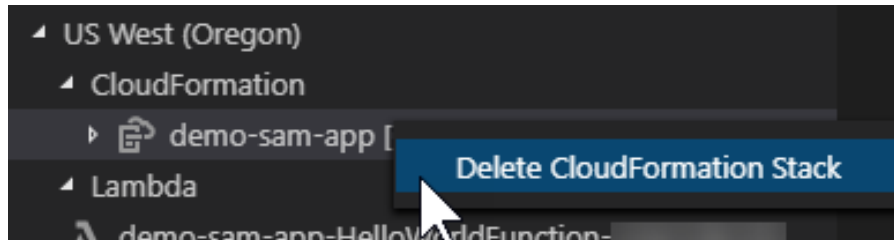
2. Choose **Delete CloudFormation Stack**.
3. In the message that appears, choose **Yes** to confirm the delete.
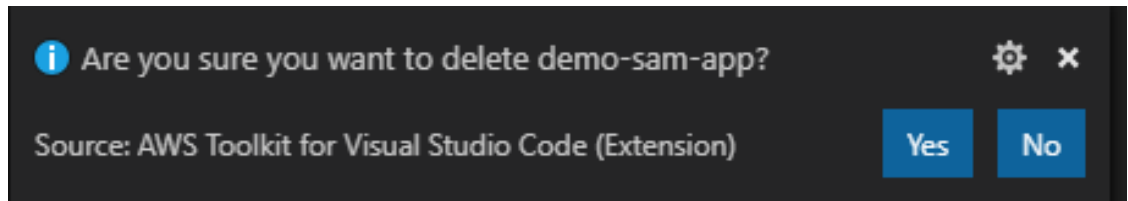
After the stack is deleted, it's no longer listed in the **AWS Explorer**.

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 (p. 2).
- The credentials you created in Establishing credentials (p. 4) 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 cash 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 (p. 39)
- Working with CloudWatch log events in log streams by using the AWS Toolkit for Visual Studio Code (p. 40)

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 (p. 39)

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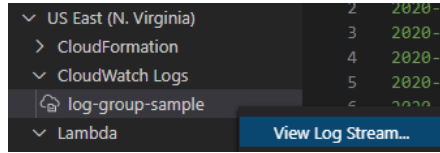
4. From the **Command Palette**, select a log stream from the group to view.

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

The **Log Stream** editor (p. 40) 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 (p. 39)).

   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).
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**.

Working with Amazon CodeWhisperer

What is CodeWhisperer?

Amazon CodeWhisperer is a general purpose, machine learning-powered code generator that provides you with code recommendations, in real time. As you write code, CodeWhisperer automatically generates suggestions based on your existing code and comments. Your personalized recommendations can vary in size and scope, ranging from a single line comment to fully formed functions.

CodeWhisperer can also scan your code to highlight and define security issues.

For more information, see the Amazon CodeWhisperer User Guide.

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 acces 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.

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 administra	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 (successor to AWS Single Sign-On) User Guide*.	Configure programmatic access by Configuring the AWS CLI to use AWS IAM Identity Center (successor to AWS Single Sign-On) in the *AWS Command Line Interface User Guide*.
In IAM (Not recommended)	Use long-term credentials to access AWS.	Following the instructions in Creating your first IAM admin user and user group in the *IAM User Guide*.	Configure programmatic access by Managing 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.

Topics

Working with the Amazon Elastic Container Registry service in VS Code

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.

Sections

Prerequisites

Before you can use the Amazon ECR service feature of the Toolkit for VS Code, you must meet these prerequisites (p. 41).

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 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 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 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 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 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 example output:

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

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 (p. 48)
- Amazon Elastic Container Service Exec in AWS Toolkit for Visual Studio Code (p. 49)

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 .

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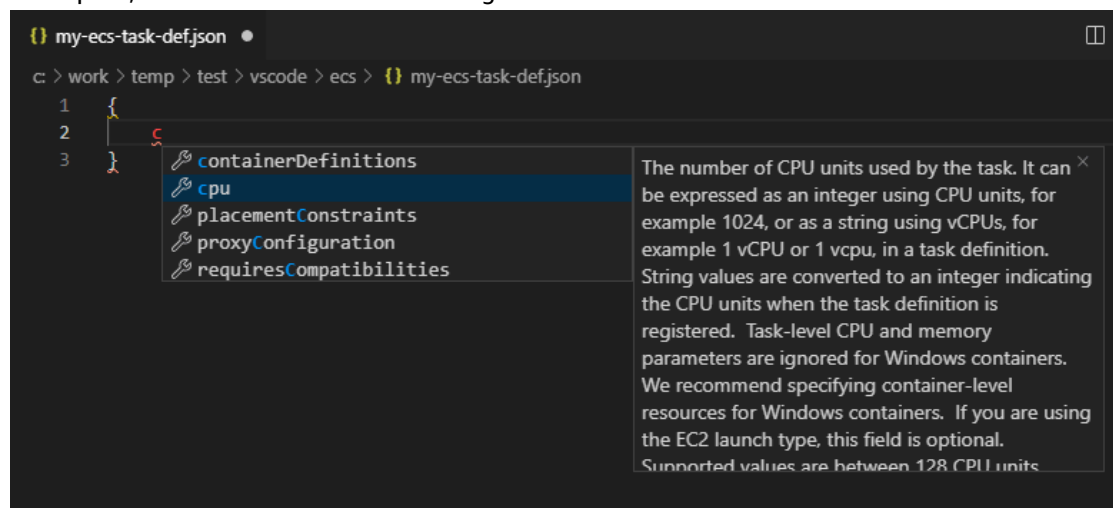
Amazon Elastic Container Service Exec in AWS Toolkit for Visual Studio Code

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 (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 (p. 51)

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 (p. 2).

*   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.

*   Open the **AWS Explorer** (p. 18) side bar.

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**.
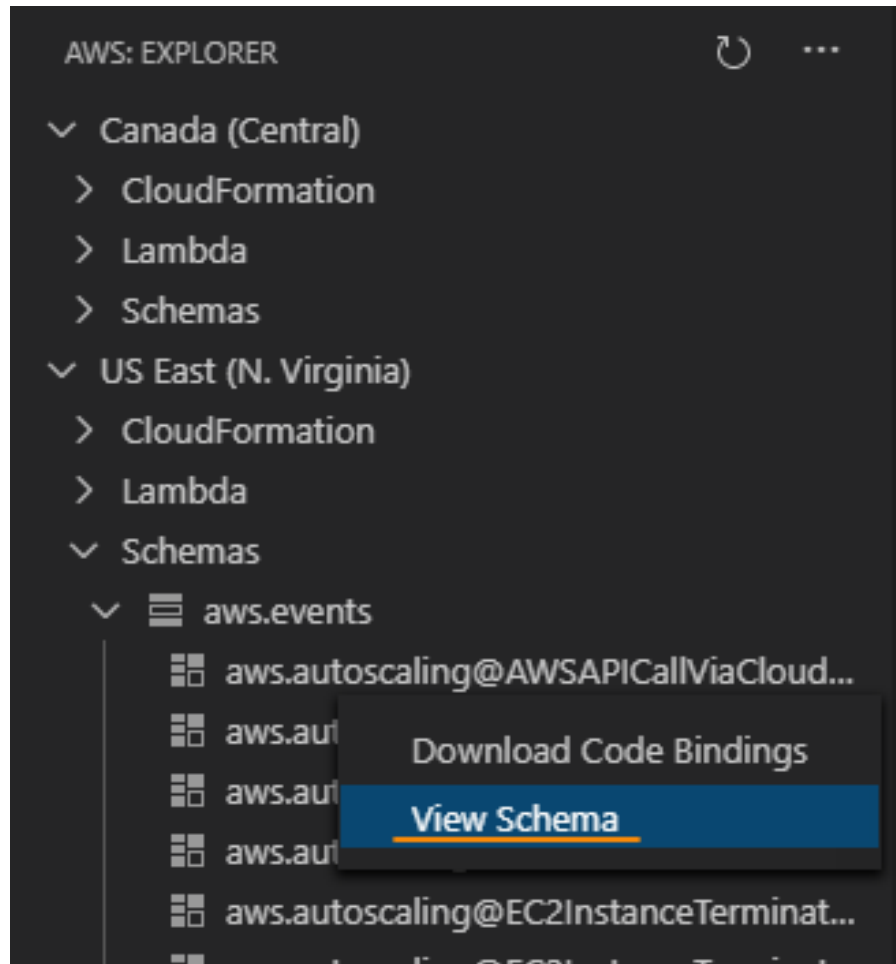
Find an Available Schema

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.
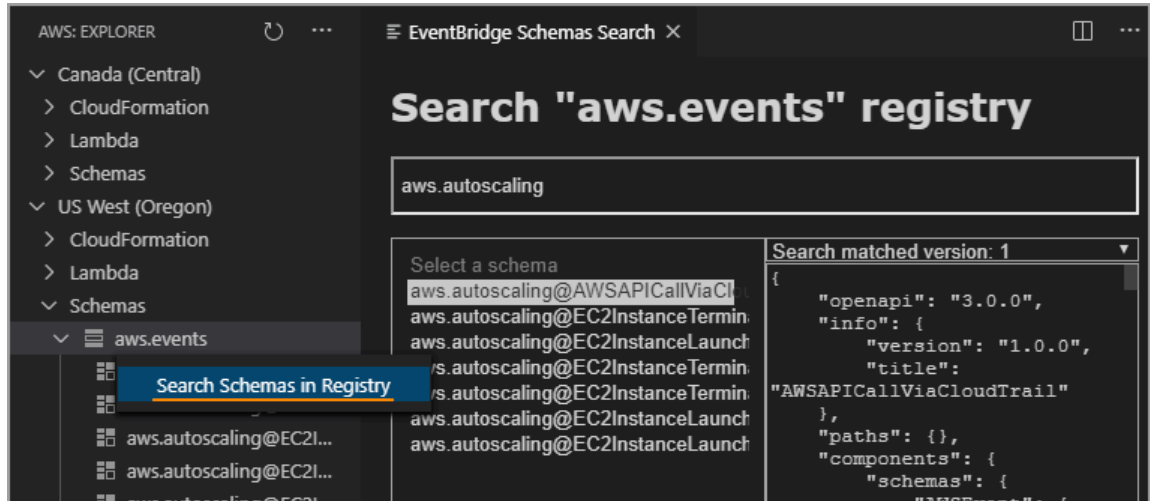
Generate Code for an Available Schema

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

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.

Working with AWS Lambda Functions

The AWS Toolkit for Visual Studio Code provides support for [AWS Lambda](#) functions. Using the Toolkit for VS Code, you can author code for Lambda functions that are part of [serverless applications](#). In addition, you can invoke Lambda functions either locally or on AWS.

Lambda is a fully managed compute service that runs your code in response to events generated by custom code or from various AWS services, such as Amazon Simple Storage Service (Amazon S3), Amazon DynamoDB, Amazon Kinesis, Amazon Simple Notification Service (Amazon SNS), and Amazon Cognito.

Topics

- Interacting with Remote Lambda Functions (p. 59)

Interacting with Remote Lambda Functions

Using the Toolkit for VS Code, you can interact with AWS Lambda functions in various ways, as described later in this topic.

For more information about Lambda, see the AWS Lambda Developer Guide.

> **Note**
> If you have already created Lambda functions by using the AWS Management Console or in some other way, you can invoke them from the Toolkit. To create a new function (using VS Code) that you can deploy to AWS Lambda, you must first create a serverless application (p. 68).

Topics

- Prerequisites (p. 59)
- Invoke a Lambda Function (p. 59)
- Delete a Lambda Function (p. 61)
- Import a Lambda Function (p. 62)
- Upload a Lambda Function (p. 63)

Prerequisites

- Be sure your system meets the the prerequisites specified in Installing the Toolkit for VS Code (p. 2).

- Ensure that the credentials you configured in Establishing credentials (p. 4) include appropriate read/write access to the AWS Lambda service. If in the **AWS Explorer**, under **Lambda**, you see a message similar to "Error loading Lambda 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.
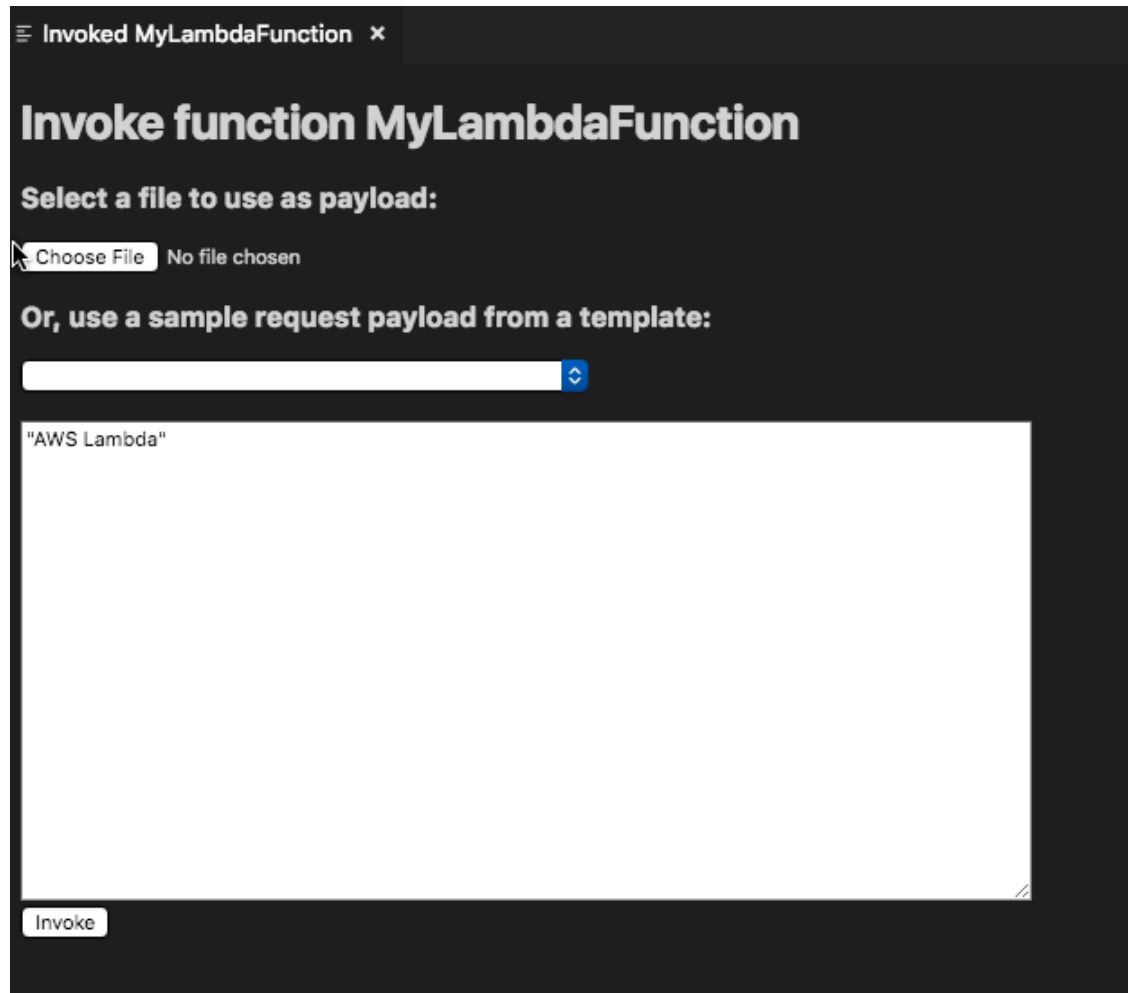
Invoke a Lambda Function

You can invoke a Lambda function on AWS from the Toolkit for VS Code.

1. In the **AWS Explorer**, choose the name of the Lambda function you want to invoke, and then open its context menu.
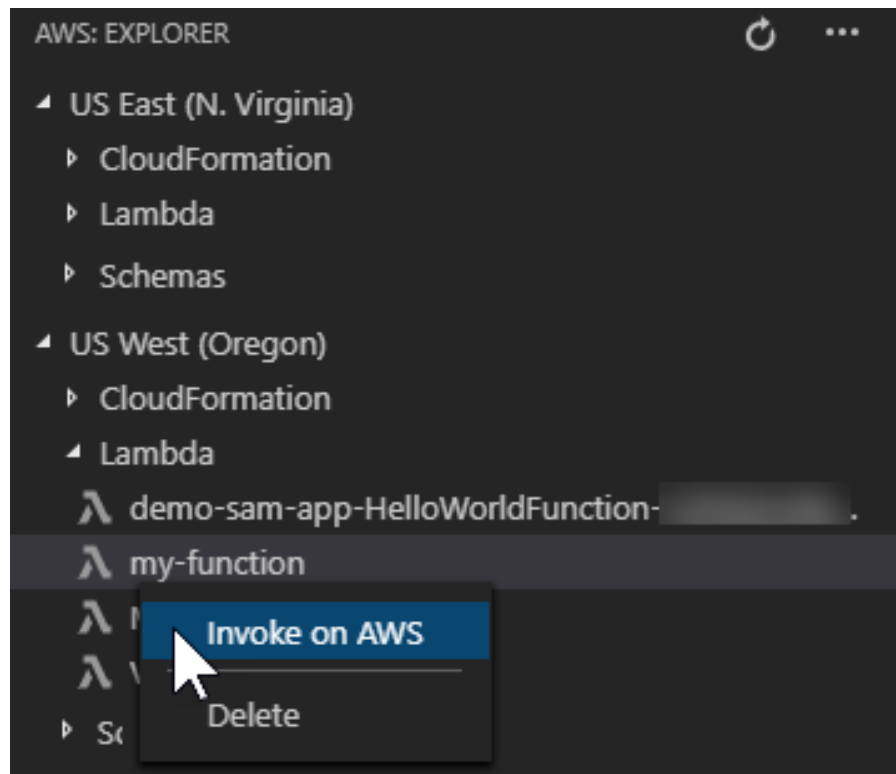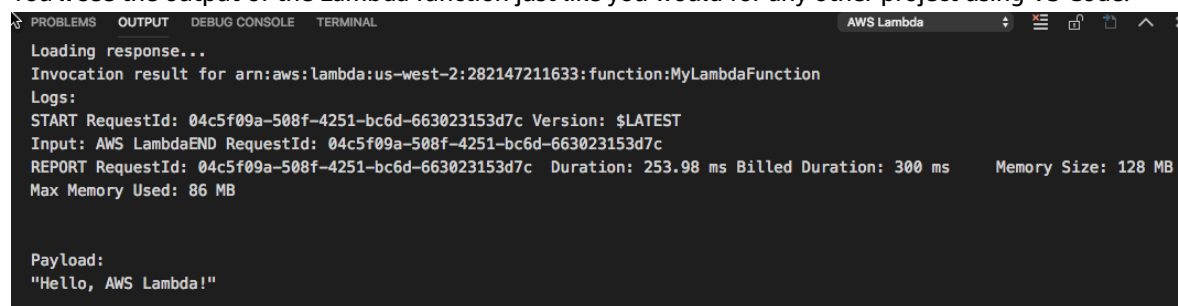
2. Choose **Invoke on AWS**.

3. In the invoke window that opens, enter the input that your Lambda function needs. The Lambda function might, for example, require a string as an input, as shown in the text box.

You'll see the output of the Lambda function just like you would for any other project using VS Code.

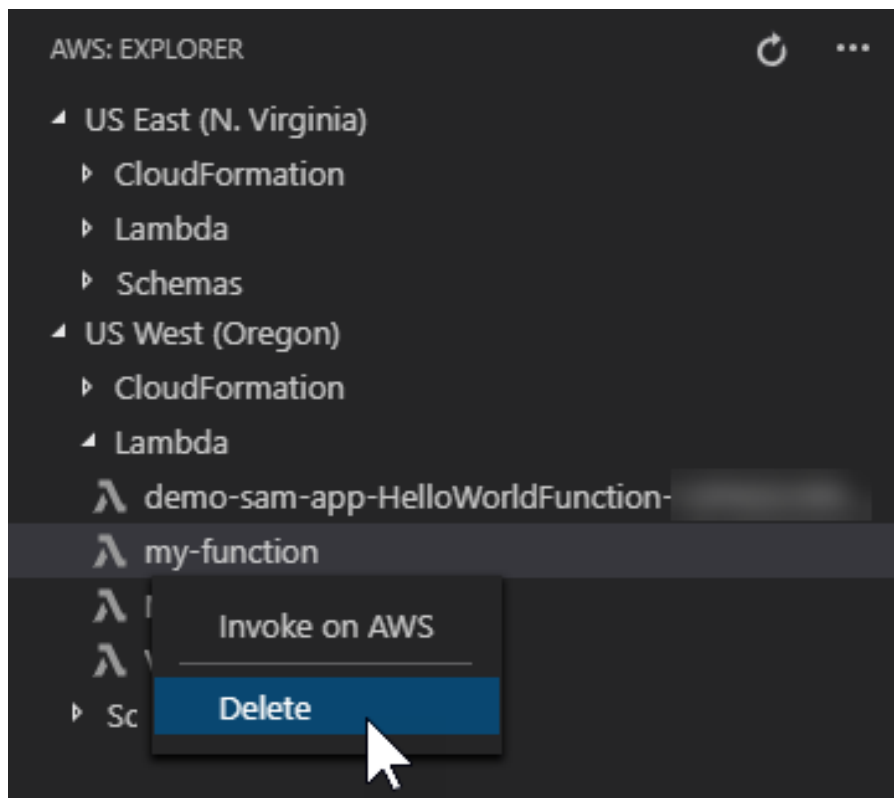# Delete a Lambda Function

You can also delete a Lambda function using the same context menu.

Warning

Do not use this procedure to delete Lambda functions that are associated with AWS CloudFormation (for example, the Lambda function that was created when creating a serverless application (p. 68) earlier in this guide). These functions must be deleted through the AWS CloudFormation stack.

1. In the **AWS Explorer**, choose the name of the Lambda function you want to delete, and then open its context menu.

2. Choose **Delete**.

3. In the message that appears, choose **Yes** to confirm the delete.
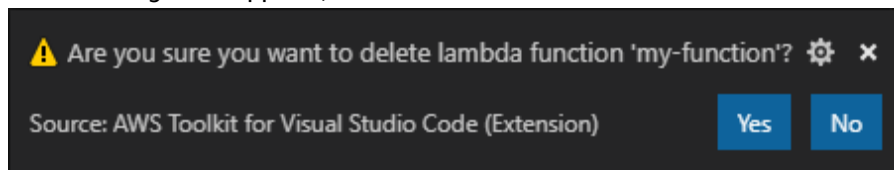
After the function is deleted, it's no longer listed in the **AWS Explorer**.

Import a Lambda Function

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

Note
The toolkit only supports importing Lambda functions using supported Node.js and Python runtimes.

1. In the **AWS Explorer**, choose the name of the Lambda function you want to import, then open its context menu.

2. Choose **Import...**

3. Choose a folder to import the Lambda code to. Folders outside the current workspace will be added to your current workspace.

4. After download, the Toolkit adds the code to your workspace and opens the file containing the Lambda handler code. The Toolkit also creates a *launch configuration*, which appears in the VS Code run panel so you can locally run and debug the Lambda function using AWS Serverless Application Model. For more information about using AWS SAM, see the section called "Running and debugging a serverless application from template (local)" (p. 70).

Upload a Lambda Function

You can update existing Lambda functions with local code. Updating code in this way does not use the AWS SAM CLI for deployment and does not create an AWS CloudFormation stack. This functionality can upload a Lambda function with any runtime supported by Lambda.

> **Warning**
> The toolkit can't check whether your code works. Make sure the code works before updating production Lambda functions.

1. In the **AWS Explorer**, choose the name of the Lambda function you want to import, then open its context menu.

2. Choose **Upload Lambda...**

3. Choose from the three options for uploading your Lambda function. The options include:

   Upload a premade .zip archive

   - Choose **Zip Archive** from the Quick Pick menu.
   - Choose a .zip file from your file system and confirm the upload with the modal dialog. This uploads the .zip file as is and immediately updates the Lambda following deployment.

   Upload a directory as is

   - Choose **Directory** from the Quick Pick menu.
   - Choose a directory from your file system.
   - Choose **No** when prompted to build the directory, then confirm the upload with the modal dialog. This uploads the directory as is and immediately updates the Lambda following deployment.

   Build and upload a directory

   > **Note**
   > This requires the AWS SAM CLI.

   - Choose **Directory** from the Quick Pick menu.
   - Choose a directory from your file system.
   - Choose **Yes** when prompted to build the directory, then confirm the upload with the modal dialog. This builds the code in the directory using the AWS SAM CLI `sam build` command and immediately updates the Lambda following deployment.

     > **Note**
     > The toolkit will warn you if it can't detect a matching handler prior to upload. If you wish to change the handler tied to the Lambda function, you can do so through the AWS Management Console or the AWS CLI.

Working with Amazon S3

The following topics describe how to use the AWS Toolkit for Visual Studio Code to work with [Amazon S3](#) buckets and objects in an AWS account.

Topics

Amazon S3 service overview

You can use Amazon S3 from the AWS Toolkit for Visual Studio Code to view, manage, and edit your Amazon S3 resources, including buckets and objects.

Topics

Managing your Amazon S3 resources

Topics

Creating an Amazon S3 bucket

Your files (known as *objects* in S3) are stored in S3 resources called *buckets*.

1. From the toolkit main menu, open the context (right-click) menu for the **S3** service, and choose **Create Bucket. . .**. Or, 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**
   > Because 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 main menu, 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.

Managing your Amazon S3 objects

Your files, folders, and any other data that's stored in an S3 bucket are referred to as S3 objects. You can use Amazon S3 to upload, download, and manage your S3 objects.

Topics
- Configuring Amazon S3 object pagination (p. 65)
- Uploading and downloading Amazon S3 objects (p. 66)
- Edit objects stored in your remote Amazon S3 resources (p. 66)
- Deleting an Amazon S3 object (p. 67)
- Generating a presigned URL for an Amazon S3 object (p. 67)

Configuring Amazon S3 object pagination

If you're working with a large number of S3 objects and folders, you can specify the number of items that you want to be displayed 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 use the Toolkit interface or the **Command Palette** to upload a file to a bucket.

Upload a file to an Amazon S3 bucket using the toolkit

1. From the toolkit main menu, 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 to an Amazon S3 bucket 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

You can download objects in an Amazon S3 bucket from the AWS Cloud to your system.

1. From the toolkit main menu, 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.

Edit objects stored in your remote Amazon S3 resources

You can use the S3 service to edit files that are stored remotely, in your remote S3 resources.

1. From the toolkit main menu, 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.

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 main menu, 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.

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 main menu, 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**.

Working with serverless applications

The AWS Toolkit for Visual Studio Code provides support for serverless applications. Using the Toolkit for VS Code, you can create serverless applications that contain AWS Lambda functions, and then deploy the applications to an AWS CloudFormation stack.

Topics

- Running and debugging local Amazon API Gateway resources (p. 75)
- Configuration options for debugging serverless applications (p. 77)
- Troubleshooting serverless applications (p. 82)

Assumptions and prerequisites

- Be sure that your system meets the required prerequisites specified in Installing the AWS Toolkit for Visual Studio Code (p. 2).
- Install the AWS Serverless Application Model (AWS SAM) command line interface (CLI) and its prerequisites. See **AWS SAM CLI** in the setup prerequisites (p. 2). If Visual Studio Code is open when you perform these installations, you might need to close and reopen the editor.
- Identify your default AWS Region in your AWS config file. For more information, see Configuration and credential file settings in the *AWS Command Line Interface User Guide*. If Visual Studio Code is open when you update your config file, you might need to close and reopen the editor.
- After installing your language SDK, be sure to configure your toolchain (p. 15).
- To ensure that you can access the CodeLens feature in AWS SAM template files, install the YAML language support VS Code extension.

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: AWS CloudFormation, IAM, Lambda, Amazon API Gateway, Amazon Simple Storage Service (Amazon S3), and Amazon Elastic Container Registry (Amazon ECR).

For information about ensuring that you have the necessary permissions to deploy and run serverless applications, see Permissions in the *AWS Serverless Application Model Developer Guide*. For information on how to set up your credentials profile, see Setting up your AWS credentials (p. 5).

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**.
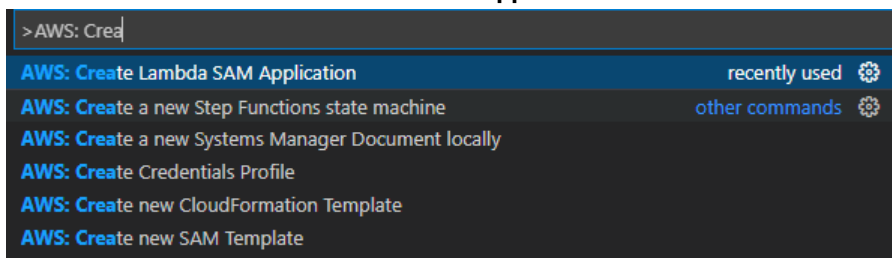
   > **Note**
   > If the AWS SAM CLI isn't installed, you get an error in the lower-right corner of the VS Code editor. If this happens, verify that you've met all the assumptions and prerequisites (p. 68).
3. Choose the runtime for your AWS SAM application.

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)
   - 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 (p. 2).

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.

AWS Toolkit for VS Code User Guide
Running and debugging a serverless
application from template (local)

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 Assumptions and prerequisites (p. 68) 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 – 2.7, 3.6, 3.7, 3.8, 3.9
> - 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 (p. 72).

In this procedure, you use the example application created in the Creating a new serverless application (local) (p. 68) 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**.

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`.

For more information about these and other entries in the `launch.json` file, see Configuration options for debugging serverless applications (p. 77).

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.)

Deploying a serverless application to the AWS Cloud

This example shows how to deploy the serverless application that created in the previous section (Creating a new serverless application (local) (p. 68)) to the AWS Cloud using the Toolkit for VS Code.

Before you perform a deployment, make sure that the following items are covered:

- For applications with deployment type `Zip`, be sure to have a globally unique Amazon S3 bucket name to use for the deployment.
- For applications with deployment type `Image`, be sure to have both a globally unique Amazon S3 bucket name and an Amazon ECR repository URI to use for the deployment.

For more information about Lambda package types, see Lambda deployment packages in the *AWS Lambda Developer Guide*.

To deploy a serverless application, follow these steps:

1. To open the **Command Palette**, choose **View**, **Command Palette**, and then enter **AWS**.
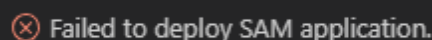2. Choose **AWS Toolkit Deploy SAM Application**.
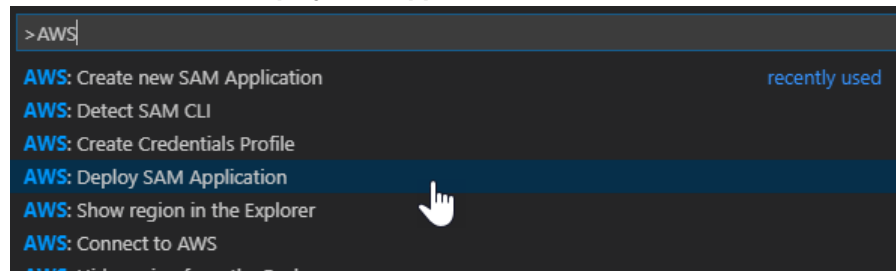
3. Choose the `template.yaml` file to use for the deployment.
4. Choose the AWS Region to deploy to.
5. Enter the name of an Amazon S3 bucket that this deployment can use. The bucket must be in the Region that you're deploying to.

   > **Warning**
   > The Amazon S3 bucket name must be globally unique across all existing bucket names in Amazon S3. Therefore, you should add a unique identifier to the name given in the following example, or choose a different name.

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. Enter a name for the deployed stack, either a new stack name or an existing stack name.
8. Verify the success of the deployment on the **OUTPUT** tab of VS Code.

   If an error occurs, a message pops up in the lower-right that is similar to the following:

If this happens, check the text in the **OUTPUT** tab for details. The following is an example of error details:

```
Error with child process: Unable to upload artifact HelloWorldFunction referenced by
 CodeUri parameter of HelloWorldFunction resource.
S3 Bucket does not exist. Execute the command to create a new bucket
AWS s3 mb s3://pbart-my-sam-app-bucket
An error occurred while deploying a SAM Application. Check the logs for more
 information by running the "View AWS Toolkit Logs" command from the Command Palette.
```

In this example, the error occurred because the Amazon S3 bucket did not exist.

If an error occurs, you can also view the AWS Toolkit logs using the Command Palette. To do this, go to the **View** menu, choose **Command Palette**, enter **AWS**, and choose **AWS Toolkit View AWS Toolkit Logs**.

When the deployment is complete, you see your application listed in the **AWS Explorer**. To learn how to invoke the Lambda function that was created as part of the application, see Interacting with Remote Lambda Functions (p. 59).

Deleting a serverless application from the AWS Cloud

Deleting a serverless application involves deleting the AWS 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 (p. 21).

2. In the **AWS Toolkit Explorer** window, expand the Region containing the deployed application that you want to delete, and then expand **AWS CloudFormation**.

3. Open the context (right-click) menu for the name of the AWS CloudFormation stack that corresponds to the serverless application that you want to delete, and then choose **Delete AWS 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 AWS CloudFormation list in **AWS Explorer**.

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.

AWS Toolkit for VS Code User Guide
Running and debugging Lambda
functions directly from code

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 (2.7, 3.6, 3.7, 3.8, 3.9)	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:
	
	• 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.

AWS Toolkit for VS Code User Guide
Running and debugging Lambda
functions directly from code

Language/runtime	Criteria for Lambda functions to be identified by CodeLens indicators
	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.
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.

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, `"nodejs.12x"`.

   - `"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.

For more information about these and other entries in the `launch.json` file, see Configuration options for debugging serverless applications (p. 77).

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.

AWS Toolkit for VS Code User Guide
Configuration options for
debugging serverless applications

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 section.

sam

How the AWS SAM CLI builds the application. For more information, see the **AWS SAM CLI ("sam") properties** table in the 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 Toolkit for VS Code User Guide
Configuration options for
debugging serverless applications

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. 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) (p. 68). 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 hard-coding the bucket name that you're writing to, configure the bucket name as an environment variable.

AWS Toolkit for VS Code User Guide
Configuration options for
debugging serverless applications

Property	Description

	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.
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.

AWS Toolkit for VS Code User Guide
Configuration options for
debugging serverless applications

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 AWS 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:
	
	<pre>"pathMappings": [
	    {
	        "localRoot": "${workspaceFolder}/sam-app/
	HelloWorldFunction",
	        "remoteRoot": "/var/task"
	    }
	]</pre>
	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	Empty string

AWS Toolkit for VS Code User Guide
Configuration options for
debugging serverless applications

Property	Description	Default value
	containers connect only to the default bridge Docker network.	
`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 in the *AWS CloudFormation User Guide*.	`"parameters":{}`

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 (p. 5) 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

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 (p. 2).
- 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:

```
{
    "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 Setting up your AWS credentials (p. 5).

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 (p. 85).
2. Make the changes that you want to be published. For more information, see Editing a Systems Manager Automation document (p. 85).
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 (p. 86).

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 (p. 84).

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.

Working with AWS Step Functions

The AWS Toolkit for Visual Studio Code (VS Code) provides support for AWS Step Functions. Using the Toolkit for VS Code, you can create, update and execute Step Functions state machines.

Topics
- Working with AWS Step Functions (p. 87)

Working with AWS Step Functions

You can use the AWS Toolkit for Visual Studio Code (VS Code) to perform various operations with state machines.

Topics

Prerequisites

- Be sure your system meets the prerequisites specified in Installing the Toolkit for VS Code (p. 2), then install the toolkit.
- Ensure that you have configured your credentials before opening the **AWS Explorer**.

Work with state machines in VS Code

You can use VS Code to interact with remote state machines, and develop state machines locally in JSON or YAML format. You can create or update state machines, list existing state machines, run them, and download them. VS Code also lets you create new state machines from templates, see a visualization of your state machine, and provides code snippets, code completion, and code validation.

List existing state machines

If you've already created state machines, you can view a list of them:

1. Open the **AWS Explorer**.
2. Select Step Functions
3. Verify that it lists all the state machines in your account.
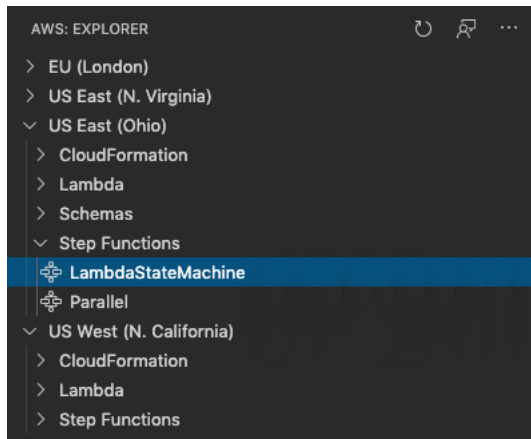
Download a state machine

To download a state machine:

1. In the **AWS Explorer**, right click the state machine that you want to download.
2. Select Download, then select the location where you want to download the state machine.
3. Verify that it downloaded correctly.
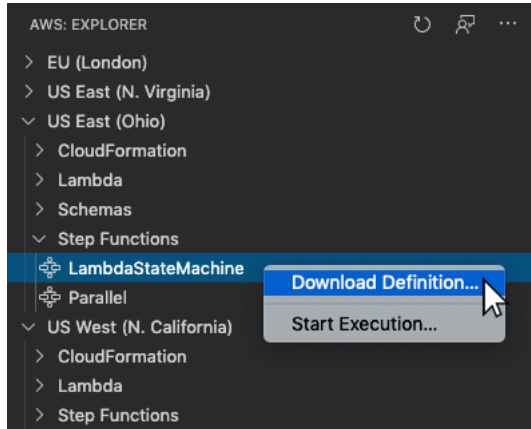
Create a state machine

You can create a new state machine yourself, or you can use a template. For more information on creating a state machine from a template, see the **State Machine Templates** section. To create a new state machine:

1. Create a new Amazon States Language (ASL) file with your state machine definition. Use the menu at the bottom right to set it as Amazon States Language.

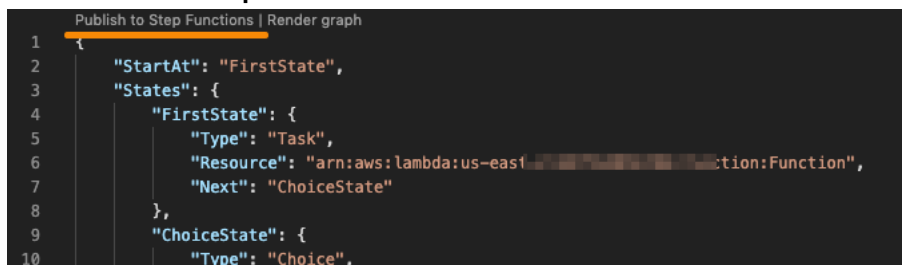2. Select **Publish to Step Functions**.
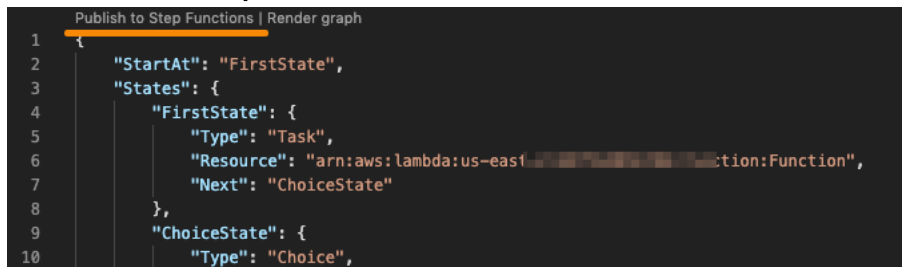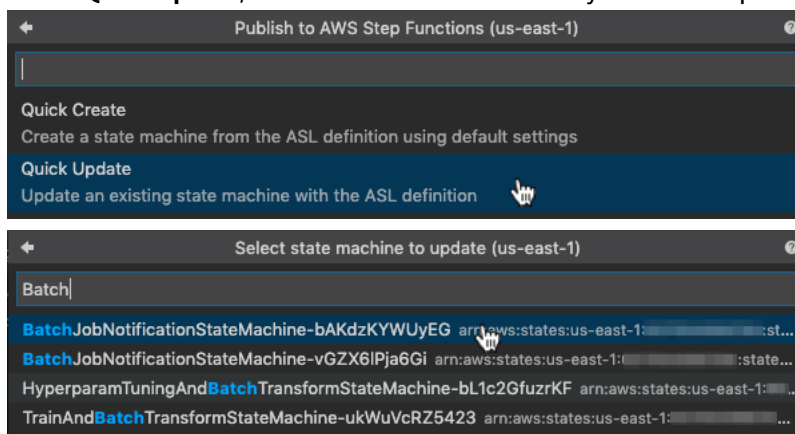

3. Select **Quick Create**, choose a role, and name your state machine.
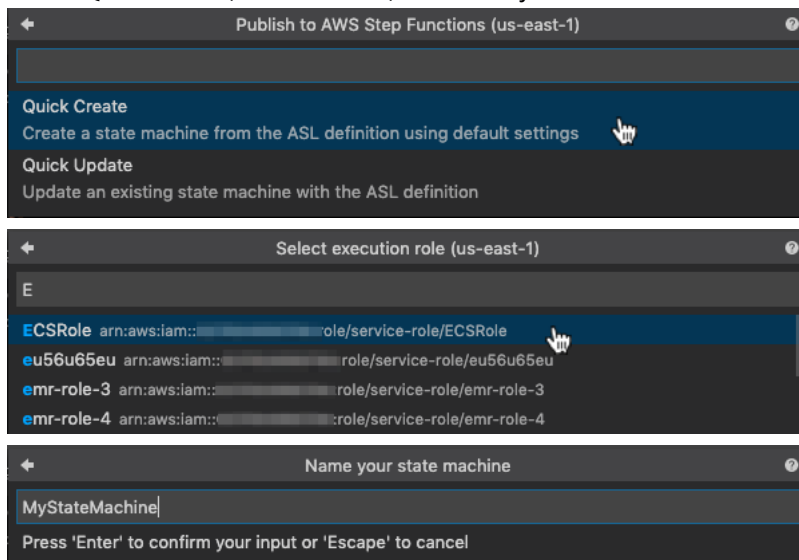
Update a state machine

To update a state machine:

1. Edit the ASL file with your state machine definition.
2. Select **Publish to Step Functions**.

3. Select **Quick Update**, then select the state machine you want to update.
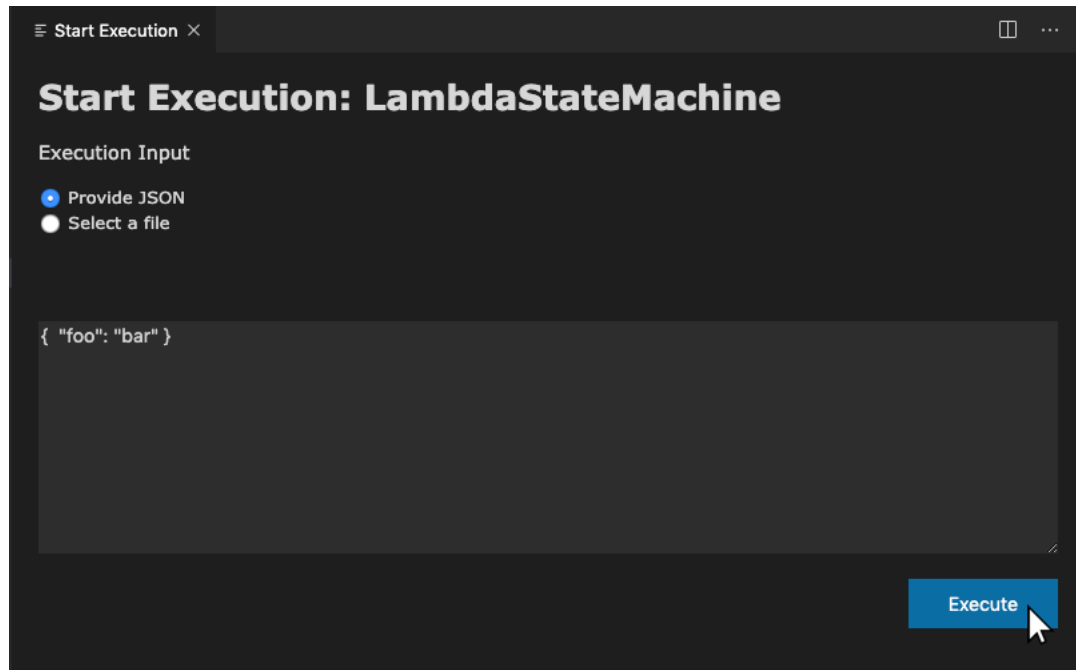

Run a state machine

To run a state machine:

1. In the **AWS Explorer**, right click the state machine that you want to run.
2. Provide input for the state machine. You can try both input from a file, and input in a text box.
3. Start the state machine and verify that it runs successfully.
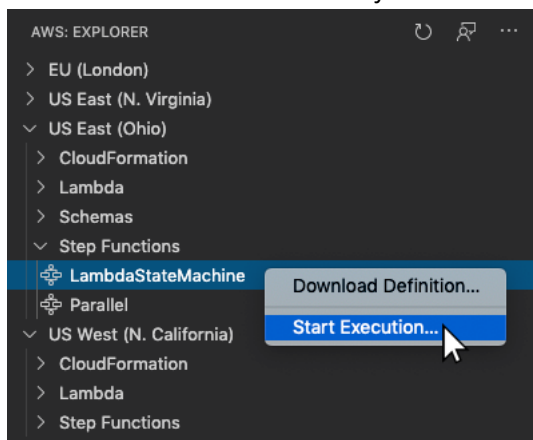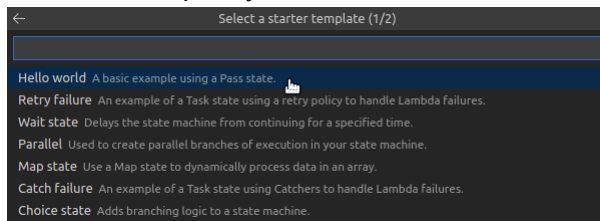
State machine templates

When you create a state machine, you have the option to create it from a template. The template contains a sample state machine definition with several commonly used states, and provides you with a starting point. To use state machine templates:

1. Open the **Command Palette** in VS Code.

2. Select **AWS Toolkit Create a new Step Functions state machine**.

3. Choose the template you want to use.

4. Choose whether you want to use the JSON or the YAML template format.
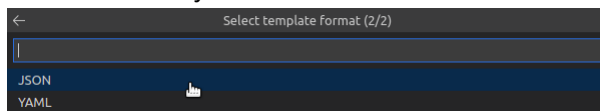

State machine graph visualization

Graph visualizations let you see what your state machine looks like in graphical format. When you create a graph visualization, another tab will open and display a visualizion of the state machine JSON or YAML. You can then compare the state machine definition you are writing concurrently with its visualization. As you change your state machine definition, the visualization will be updated.

Note
To create a visualizion of a state machine definition, the definition must be open in the active editor. If you close or rename the definition file, the visualization will close.

To create a state machine graph visualization:

1. Define your state machine.
2. Open the **Command Palette** in VS Code.
3. To create a visualization, use the visualization button in the upper right corner, or choose **AWS Render graph**.
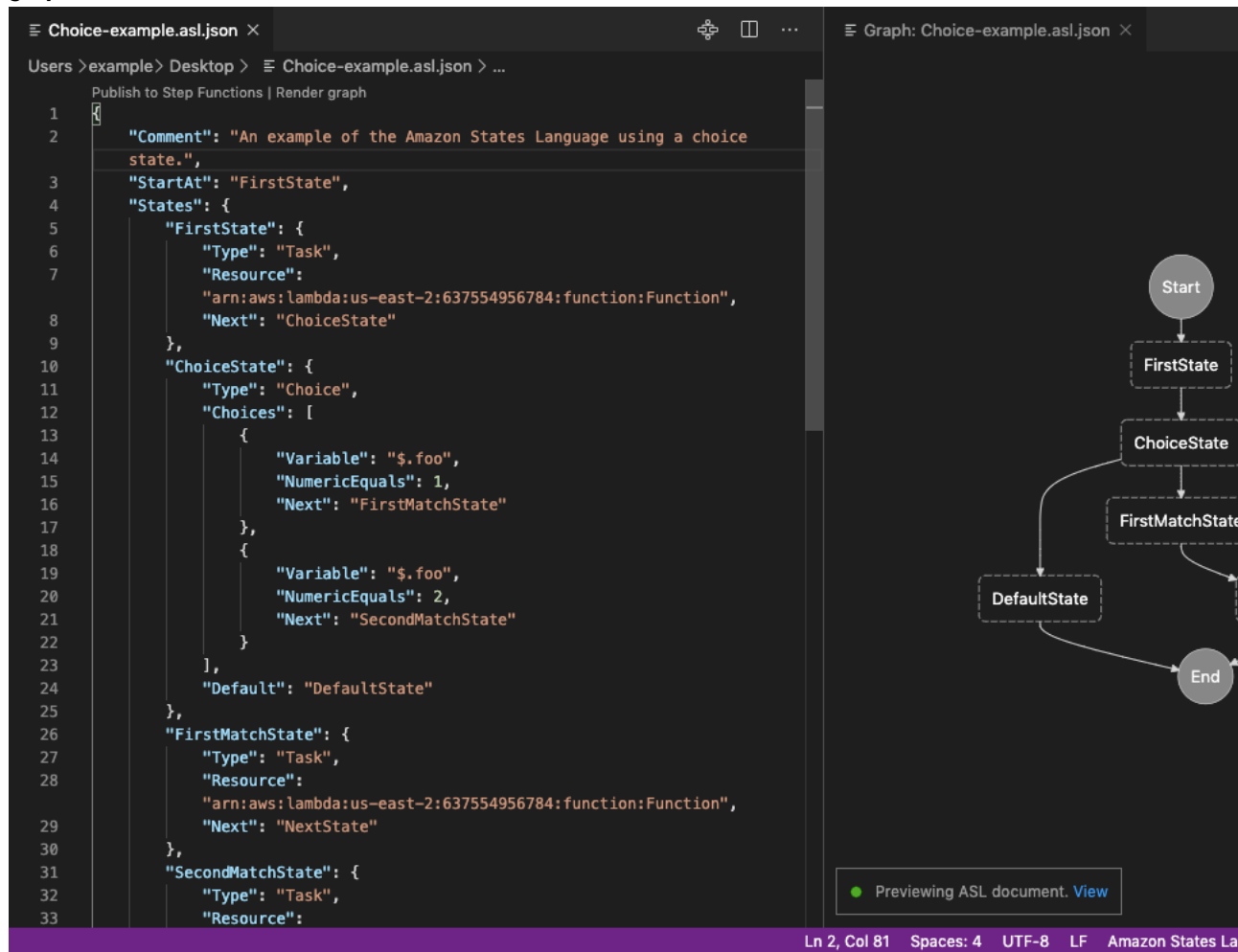

Code snippets

Code snippets let you insert short sections of code. To use code snippets:

1. Open a file and save it with the extension `.asl.json` for JSON format, or `.asl.yaml` for YAML format.
2. Create a new state machine with the **States** property.
3. Place the cursor within **States**.
4. Use the key combination `Control + Space`, and select your preferred code snippet.
5. Use Tab to traverse the variable and parameters in the code snippet.

6. Test **Retry** and **Catch** snippets by placing the cursor within the related state.
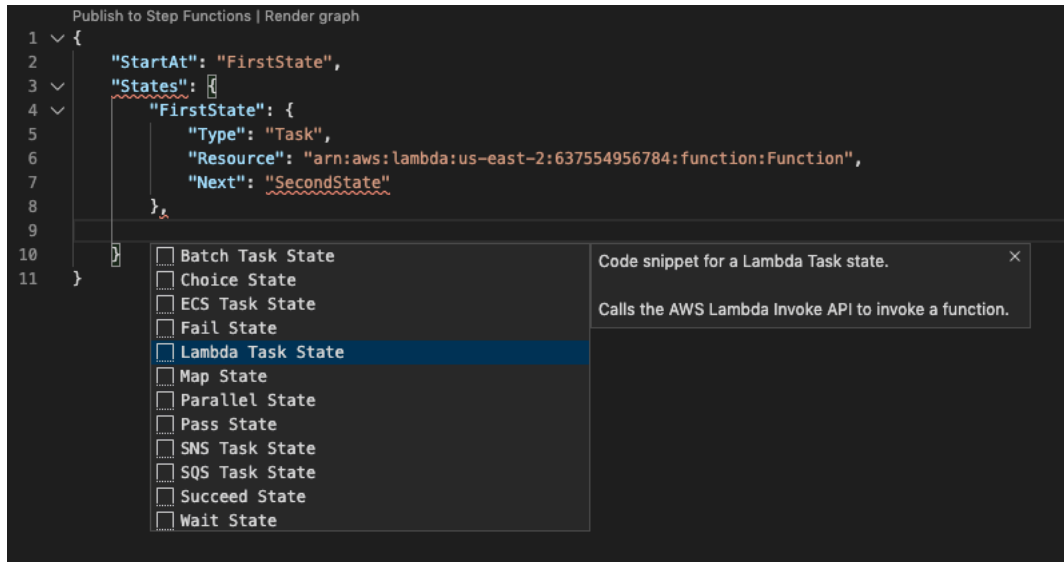

Code completion and validation

To see how code completion works:

1. Create several states.

2. Place the cursor after a **Next**, **StartAt**, or **Default** property.

3. Use the key combination `Control + Space` to list available completions. Additional properties can be accessed using `Control + Space` again, and will be based on the `Type` of the `State`.

4. As you work, code validation will happen for:

- Missing properties

- Incorrect values

- No terminal state

- Nonexistent states that are pointed to
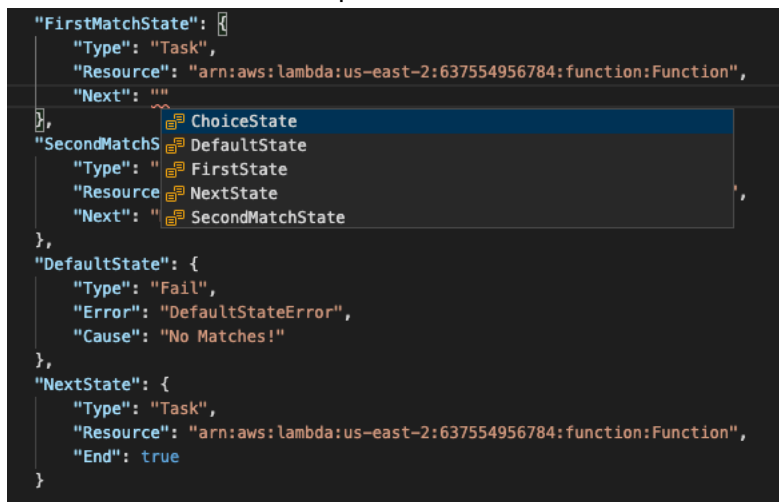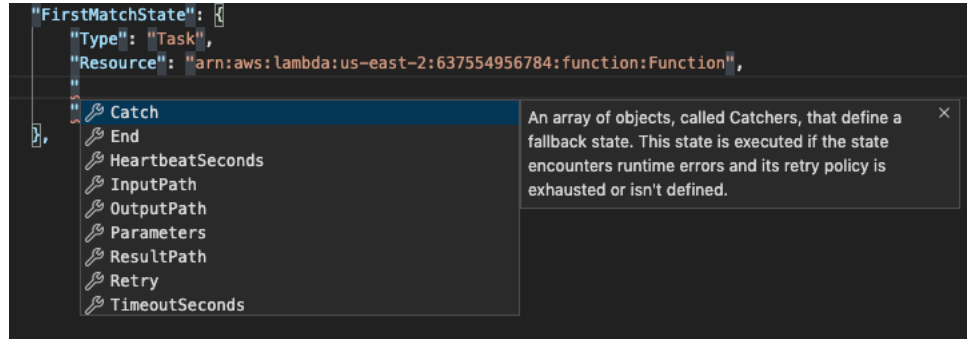
Working with resources

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 (p. 20).

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.
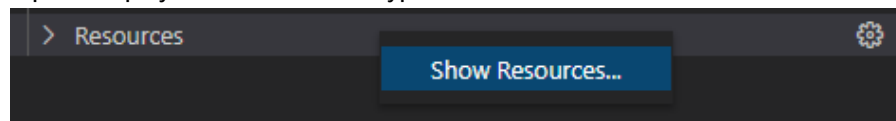
2. In the selection pane, select the resource types to add to the **AWS Explorer** and press **Return** or choose **OK** to confirm.

   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 ??? (p. 94) enabled.

   - **Update**: Edit the JSON-formatted template for the resource in a VS Code editor. For more information, see Creating and editing resources (p. 96).

   - **Delete**: Delete the resource by confirming the deletion in a dialog box that is displayed. (Deleting resources is currently an ??? (p. 94) 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 AWS CloudFormation template. For more information, see this Knowledge Center article.
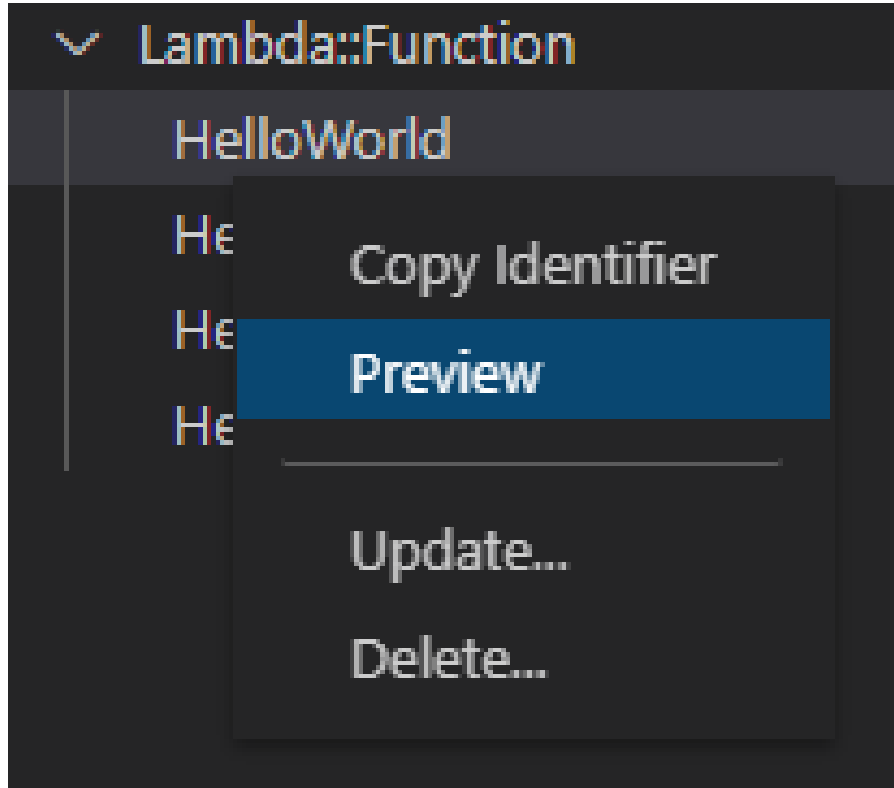
Creating and editing resources

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 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**.
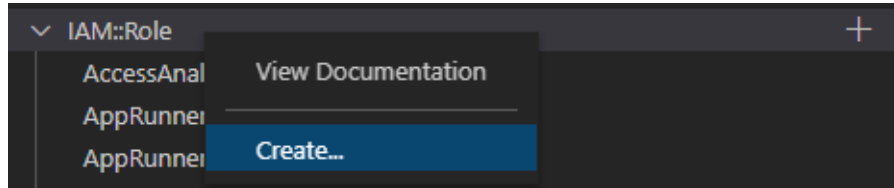
3. In the editor, start to define properties that make up the resource template. The autocomplete feature suggests property names that conform with your template's schema. When you hover over a property type, a pane displays a description of what it's used for. For detailed information about the schema, choose **View Documentation**.

   Any text that doesn't conform to the resource schema is indicated by a wavy red underline.
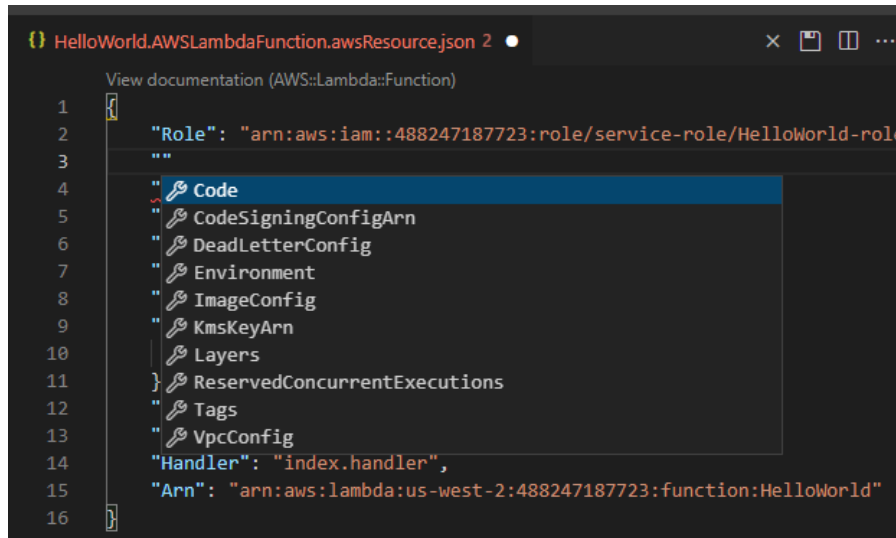
4. After you finish declaring your resource, choose the **Save** icon to validate your template and save the resource to the remote AWS Cloud.

   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.

Security for AWS Toolkit for Visual Studio Code

Topics
- Data protection in AWS Toolkit for Visual Studio Code (p. 98)

Data protection in AWS Toolkit for Visual Studio Code

The AWS shared responsibility model applies to data protection in AWS Toolkit for JetBrains. 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. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center (successor to AWS Single Sign-On) 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 recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- 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 140-2 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) 140-2.

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 AWS Toolkit for JetBrains 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

Latest major documentation update: October 14, 2021

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 S3 Service guides consolidated. (p. 64)	All previous S3 service and feature guides have been consolidated into the guide S3 service overview.	March 23, 2022
Created userguide: Create a CloudFormation template (p. 38)	Created a new userguide describing how to Create a CloudFormation template using the Toolkit for VS Code	December 17, 2021
Minor UI Update (p. 87)	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 (p. 49)	This is an overview of the Amazon ECS Exec.	December 13, 2021
Created user guide for the AWS IoT Toolkit for VS Code service (p. 53)	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 (p. 20)	Added support for turning on experimental features for AWS services.	October 14, 2021
Support for AWS resources (p. 94)	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 (p. 41)	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 (p. 67)	You can now run serverless applications in ARM64-based	October 1, 2021

	emulated environments as well as in x86_64-based environments.	
AWS Serverless Application (p. 67)	Added support for running AWS SAM applications on ARM64 platform	September 30, 2021
Format update Node.js section (p. 15)	Per customer feedback, updated formatting for Node.js/ TypeScript.	August 12, 2021
App Runner support (p. 29)	Added support for AWS App Runner to AWS Toolkit for Visual Studio Code.	August 11, 2021
Debugging Go functions (p. 70)	Added support for debugging local Go functions.	May 10, 2021
Debugging Java functions (p. 70)	Added support for debugging local Java functions.	April 22, 2021
YAML support for AWS Step Functions (p. 87)	Added YAML support for AWS Step Functions.	March 4, 2021
Debugging Amazon API Gateway resources (p. 67)	Added support for debugging local Amazon API Gateway resources.	December 1, 2020
Amazon API Gateway (p. 29)	Added support for Amazon API Gateway.	December 1, 2020
AWS Serverless Application (p. 67)	Added support for Lambda container images with serverless applications.	December 1, 2020
AWS Systems Manager support (p. 83)	Added support for Systems Manager Automation documents.	September 30, 2020
CloudWatch Logs (p. 39)	Added support for CloudWatch Logs.	August 24, 2020
Amazon S3 (p. 64)	Added support for Amazon S3.	July 30, 2020
AWS Step Functions support (p. 87)	Added support for AWS Step Functions.	March 31, 2020
Security Content (p. 98)	Added security content.	February 6, 2020
Working with Amazon EventBridge Schemas (p. 51)	Added support for Amazon EventBridge Schemas	December 1, 2019
AWS CDK (p. 35)	Preview release of the AWS CDK service.	November 25, 2019
Using an external credential process (p. 9)	Added information about using an external credential process to obtain AWS credentials.	September 25, 2019

Using IntelliSense for task-definition files (p. 48)	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 (p. 1)	Release for general availability.	July 11, 2019
User Guide for the AWS Toolkit for Visual Studio Code (p. 1)	Updated the document structure for clarity and ease of use.	July 3, 2019
Installing the AWS Toolkit for VS Code (p. 2)	Added information about installing language SDKs to support various toolchains.	June 12, 2019
Configure your toolchain (p. 15)	Added information about configuring various toolchains.	June 12, 2019
Initial Release (p. 99)	Initial release of the user guide for AWS Toolkit for Visual Studio Code.	March 28, 2019