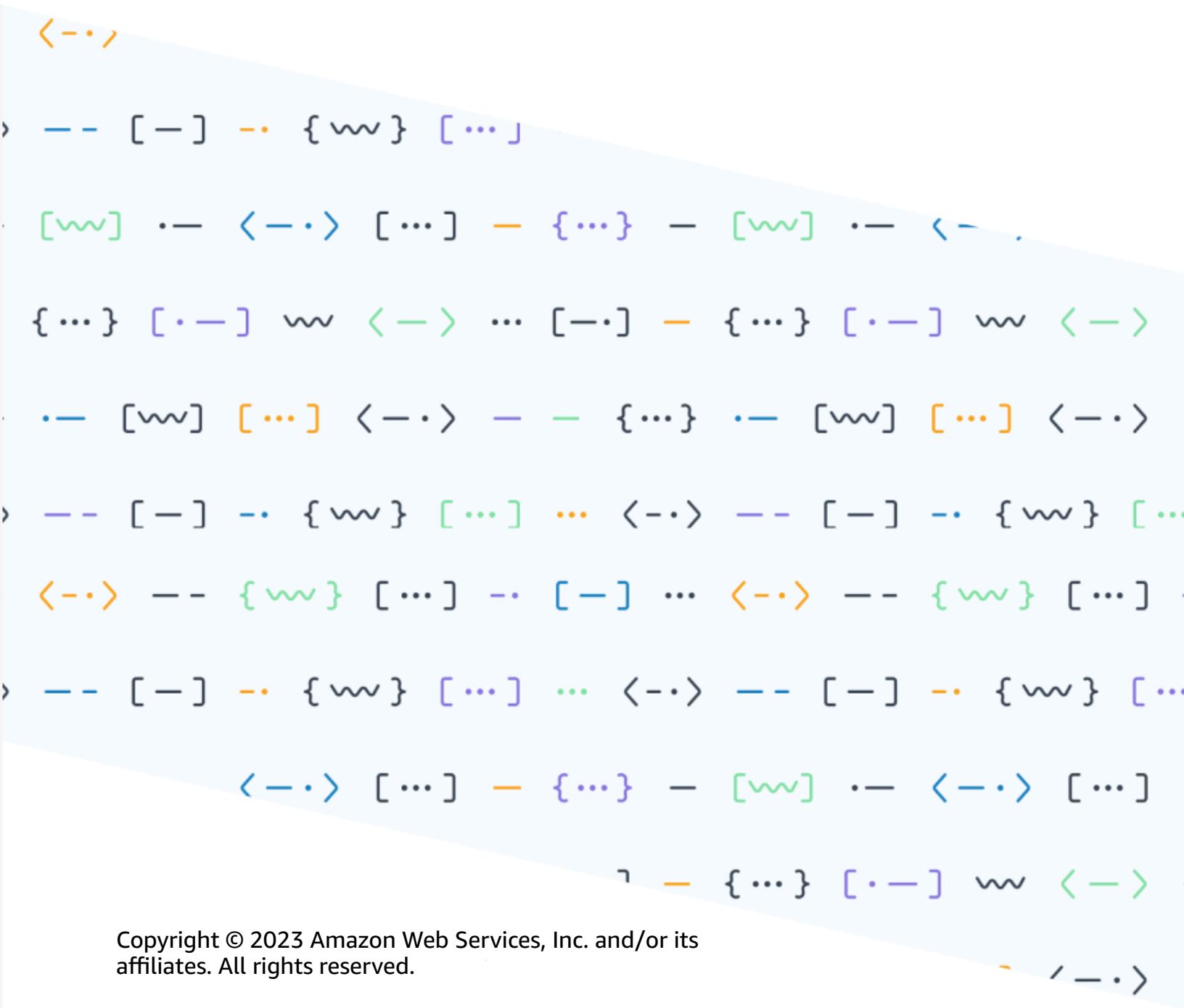
aws

Developer Guide

AWS Application Composer



Copyright © 2023 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Application Composer: Developer Guide

Copyright © 2023 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

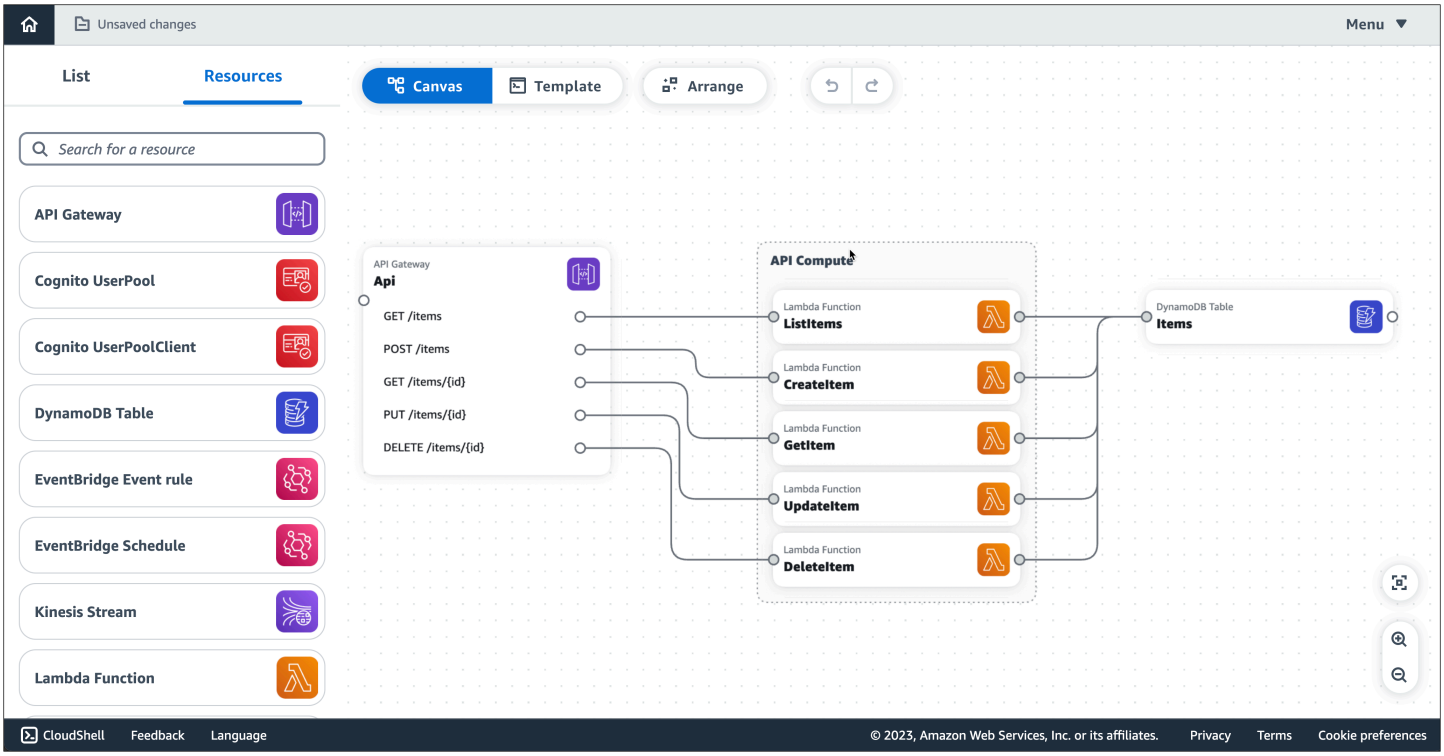
What is Application Composer? 1
Design your architecture 2
Define your templates 4
Integrate with your workflows 5
Access Application Composer 6
Benefits of Application Composer 8
Learn more 8
Next steps 8
Setting up 9
Sign up for an AWS account 9
Create an administrative user 10
Accessing Application Composer 11
Accessing Application Composer from the AWS Management Console 11
Accessing Application Composer from the AWS Toolkit for Visual Studio Code 11
Next steps 12
Getting started 13
Take a tour 13
Tutorial 1: Load and modify the demo 14
Step 1: Open the demo 14
Step 2: Explore the visual canvas 15
Step 3: Expand your architecture 19
Step 4: Save your application 20
Next steps 21
Tutorial 2: Build your first application 21
Resource properties 22
Step 1: Create your project 22
Add cards 25
Step 3: Configure your REST API 26
Step 4: Configure your functions 27
Step 5: Connect your cards 28
Step 6: Organize the canvas 29
Add a DynamoDB table 30
Step 8: Review your template 31
Step 9: Integrate into your workflows 32

Next steps ... 32
Using Application Composer ... 33
Application Composer basics ... 33
Gestures and shortcuts ... 33
Designing with cards ... 38
Connect cards together ... 48
View code updates with the Change Inspector ... 54
Referencing external files ... 58
Featured AWS services ... 67
Amazon RDS ... 67
Step Functions ... 73
Amazon VPC ... 82
Using Application Composer from the AWS Management Console ... 95
Prerequisites ... 95
Important links ... 96
Visual overview ... 96
Manage your project ... 104
Using Application Composer with your local IDE ... 109
Using Application Composer from the Toolkit for VS Code ... 112
Prerequisites ... 112
Important links ... 113
Visual overview ... 113
Manage your project ... 115
Deploy your application ... 115
Using Application Composer with CodeWhisperer ... 117
Working with other services ... 120
AWS CloudFormation and AWS SAM ... 120
What is AWS SAM? ... 120
AWS SAM prerequisites ... 122
Using Application Composer with the AWS SAM CLI ... 122
Examples ... 124
Lambda console ... 132
Reference ... 134
File System Access API ... 134
What is the File System Access API? ... 134
What is the local sync mode? ... 135

What web browsers are supported? ... 135
What does Application Composer gain access to? ... 135
Card reference ... 135
Enhanced component cards ... 136
Future enhanced component card support ... 137
Troubleshooting ... 137
Error messages ... 137
Submit feedback ... 139
Serverless concepts ... 140
Serverless concepts ... 140
Security ... 142
Data protection ... 142
Data encryption ... 144
Encryption in transit ... 144
Key management ... 144
Inter-network traffic privacy ... 144
Identity and access management ... 144
Audience ... 145
Authenticating with identities ... 145
Managing access using policies ... 148
How AWS Application Composer works with IAM ... 151
Compliance validation ... 157
Resilience ... 159
Document history ... 160

What is AWS Application Composer?

AWS Application Composer is a visual builder for modern applications on AWS. Use Application Composer to design your application architecture and visualize your AWS CloudFormation infrastructure. As you design, through a delightful drag-and-drop interface, Application Composer creates your infrastructure as code (IaC) templates, following AWS best practices. With Application Composer, you can start with an initial sketch, create deployable code, and integrate into your current workflows to improve your development experience.



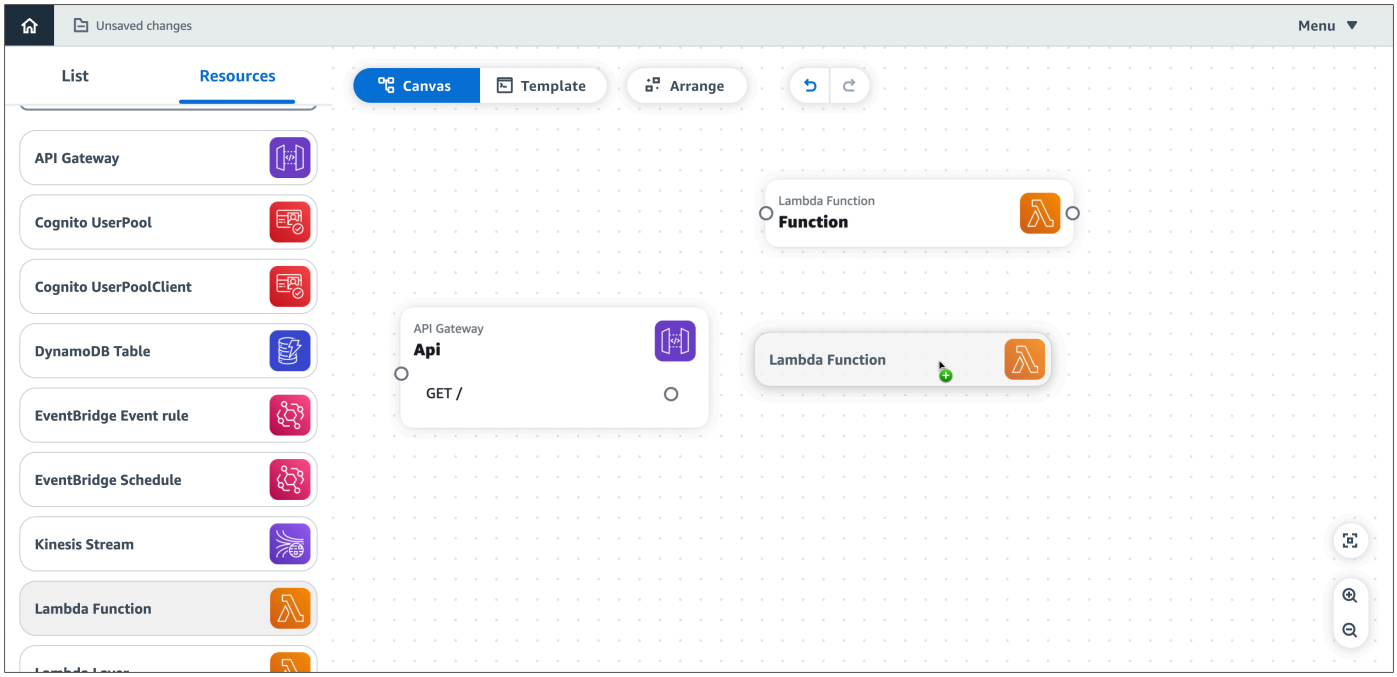
Topics

- Design your application architecture
- Define your infrastructure as code (IaC) templates
- Integrate with your existing workflows
- Ways to access Application Composer
- Benefits of Application Composer
- Learn more
- Next steps

Design your application architecture

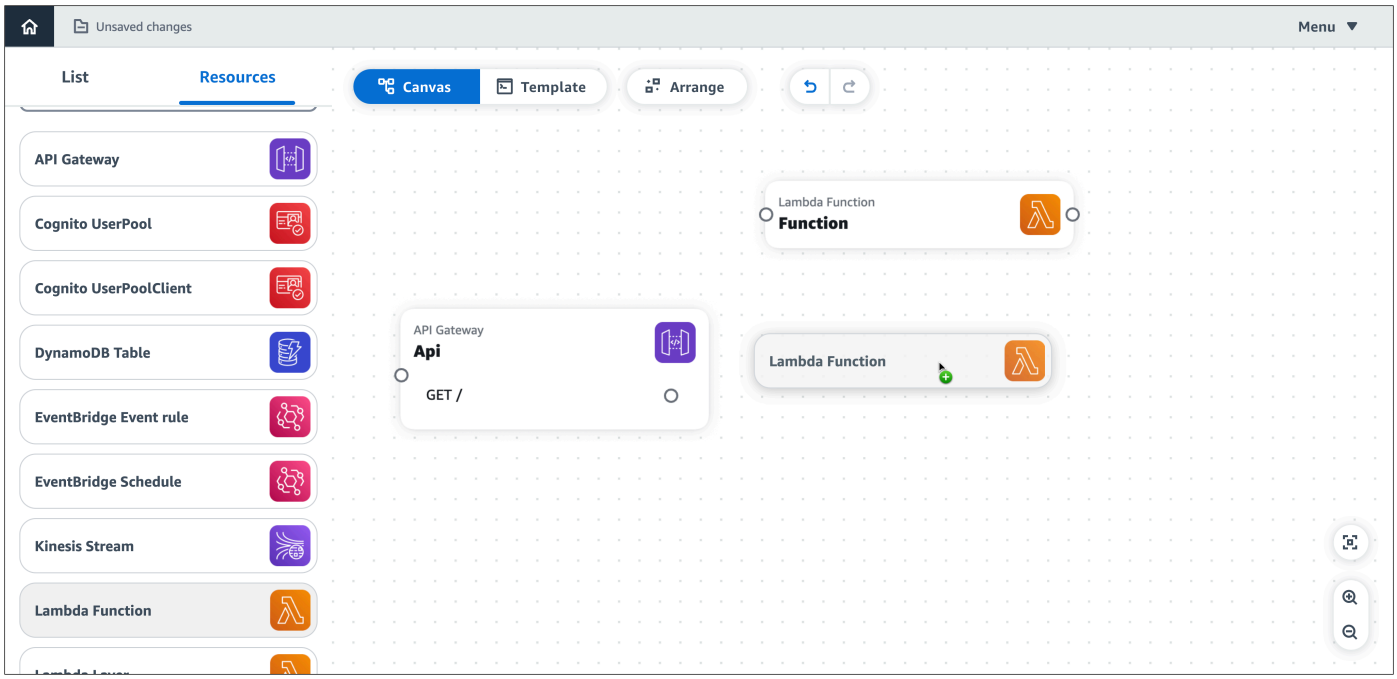
Design with cards

Place cards on the Application Composer canvas to visualize and design your application architecture.



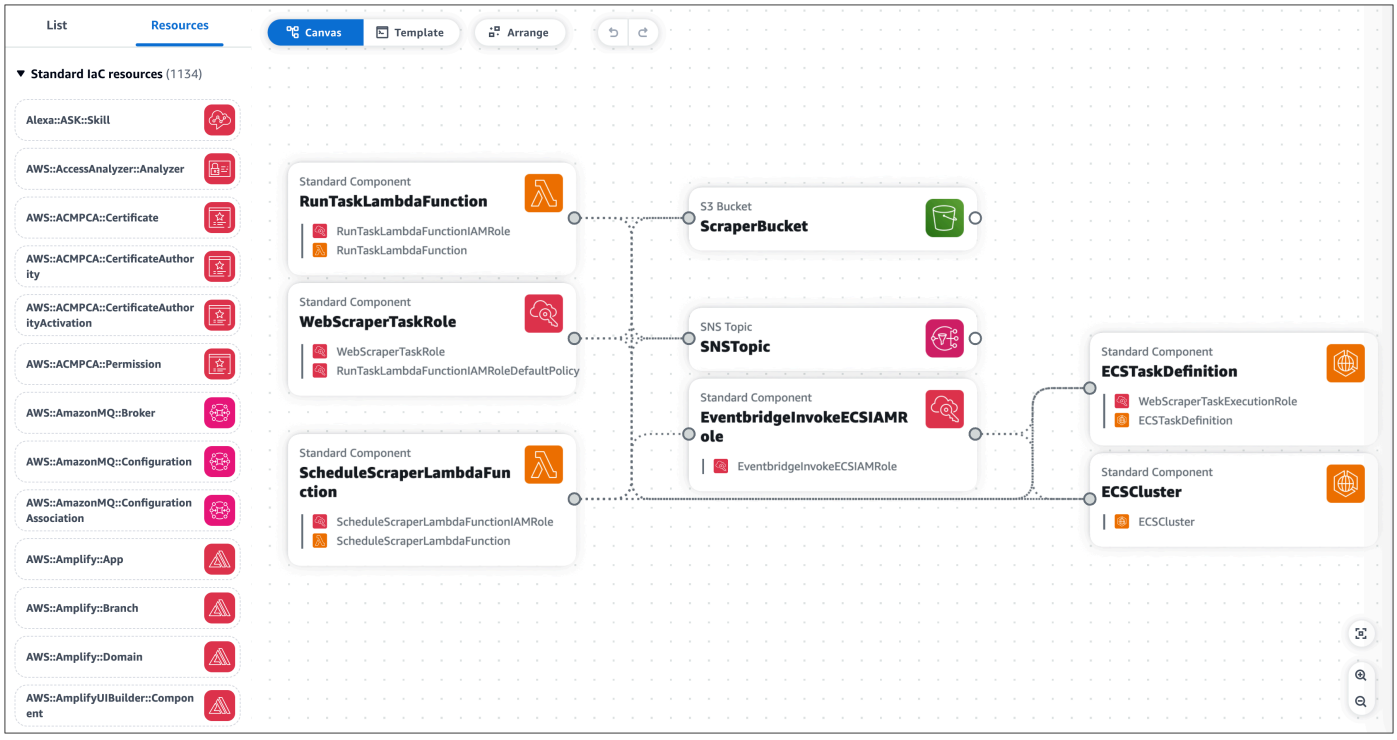
Connect cards together

Configure how your resources interact with each other by visually connecting them together. Specify their properties further through a curated properties panel.



Work with any AWS CloudFormation resource

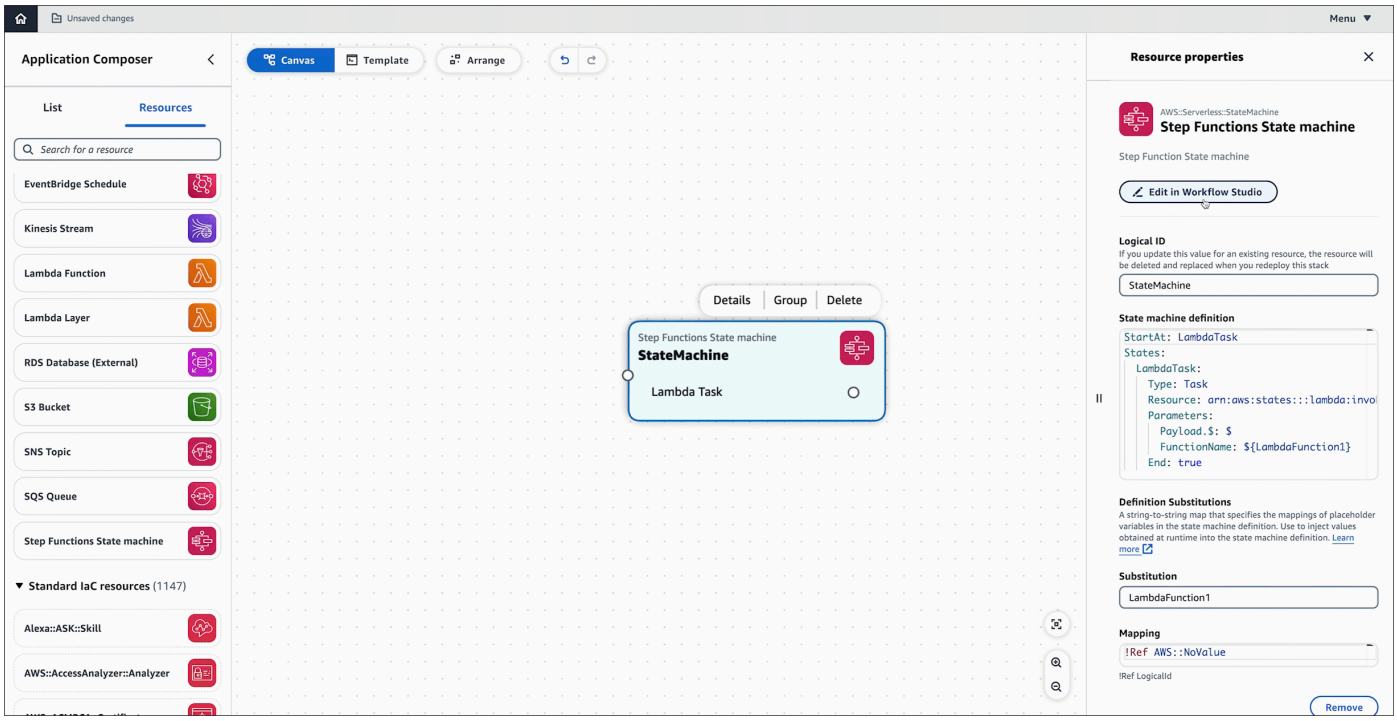
Drag any AWS CloudFormation resource onto the canvas to design your application architecture. Application Composer provides a starting IaC template that you can use to specify the properties of your resource. To learn more, see Designing with cards in Application Composer.



Access additional capabilities with featured AWS services

Application Composer features AWS services that are commonly used or configured together when building applications. To learn more, see Featured AWS services.

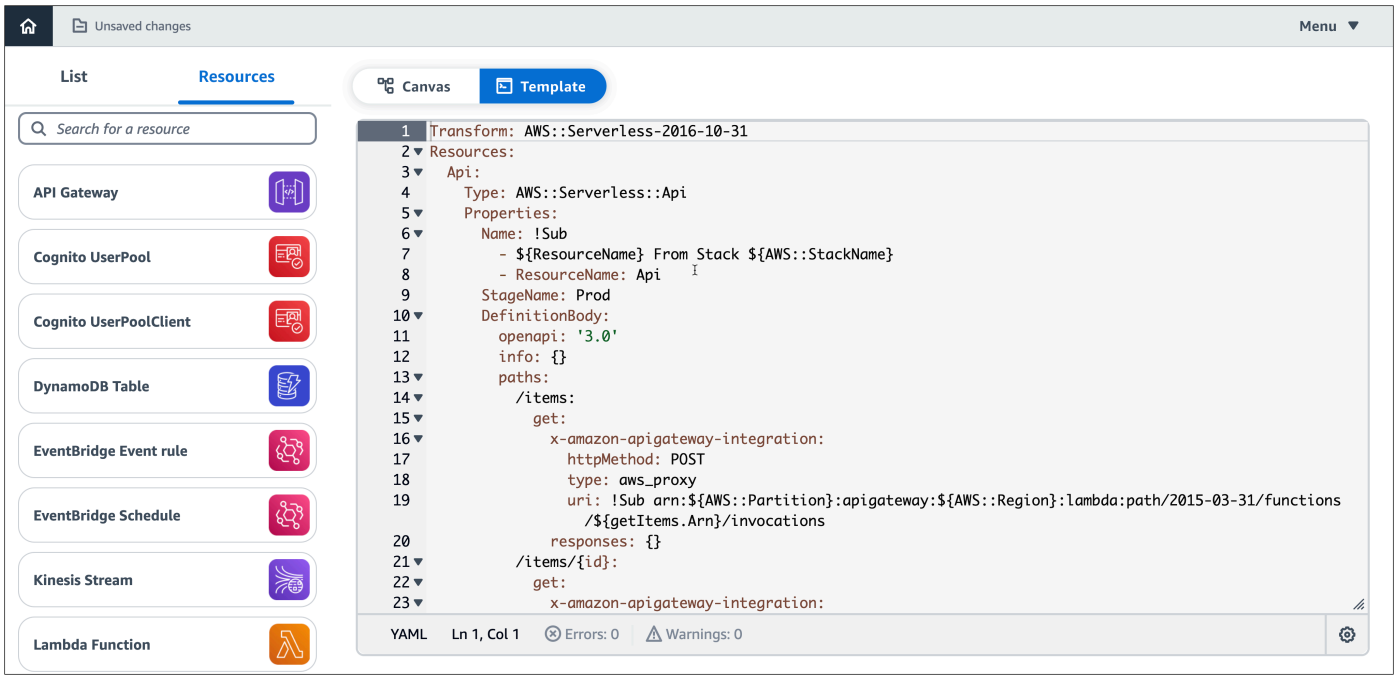
The following is an example of the AWS Step Functions feature, which provides an integration to launch Step Functions Workflow Studio directly within the Application Composer canvas.



Define your infrastructure as code (IaC) templates

Application Composer creates your infrastructure code

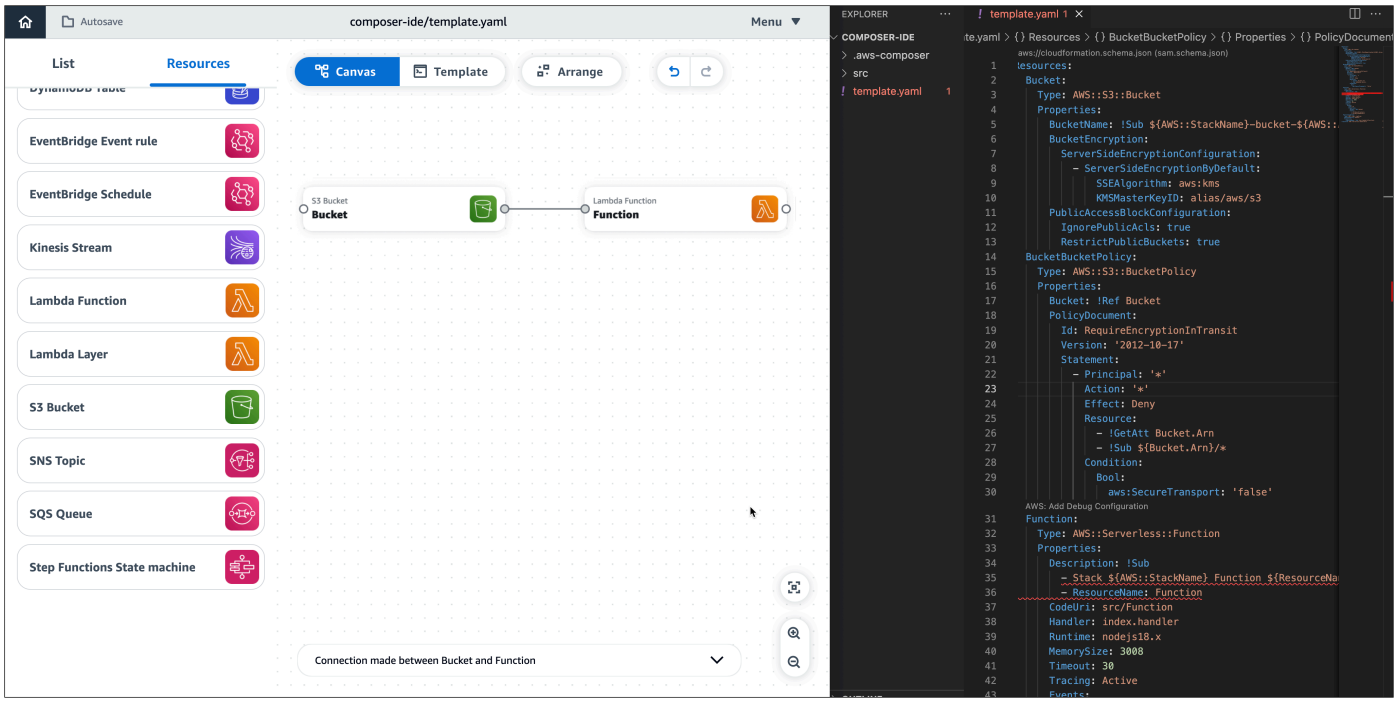
As you design, Application Composer automatically creates your AWS CloudFormation and AWS Serverless Application Model (AWS SAM) templates, following AWS best practices. You can view and modify your templates directly from within Application Composer. Application Composer automatically syncs changes between the visual canvas and your template code.



Integrate with your existing workflows

Import existing templates and projects

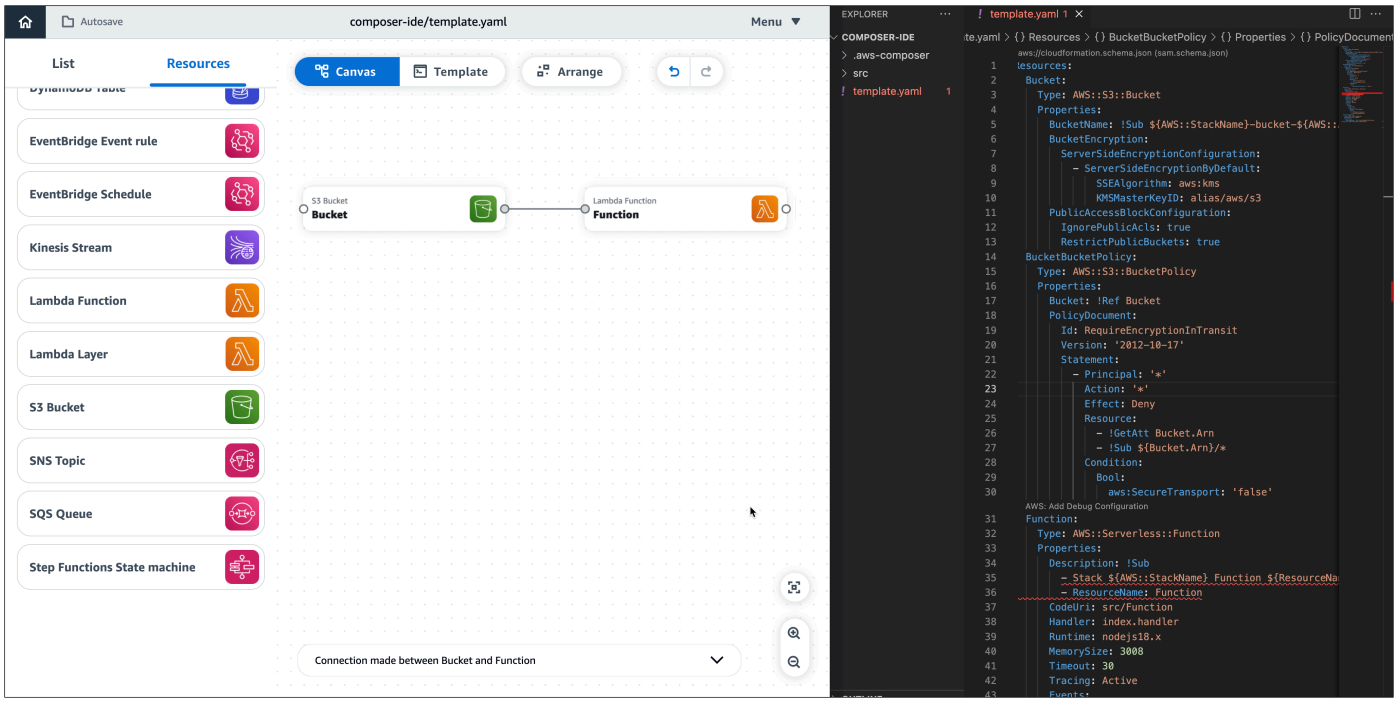
Import existing AWS CloudFormation and AWS SAM templates to visualize them for better understanding and modify their design. Export the templates that you create within Application Composer and integrate them into your existing workflows towards deployment.



Ways to access Application Composer

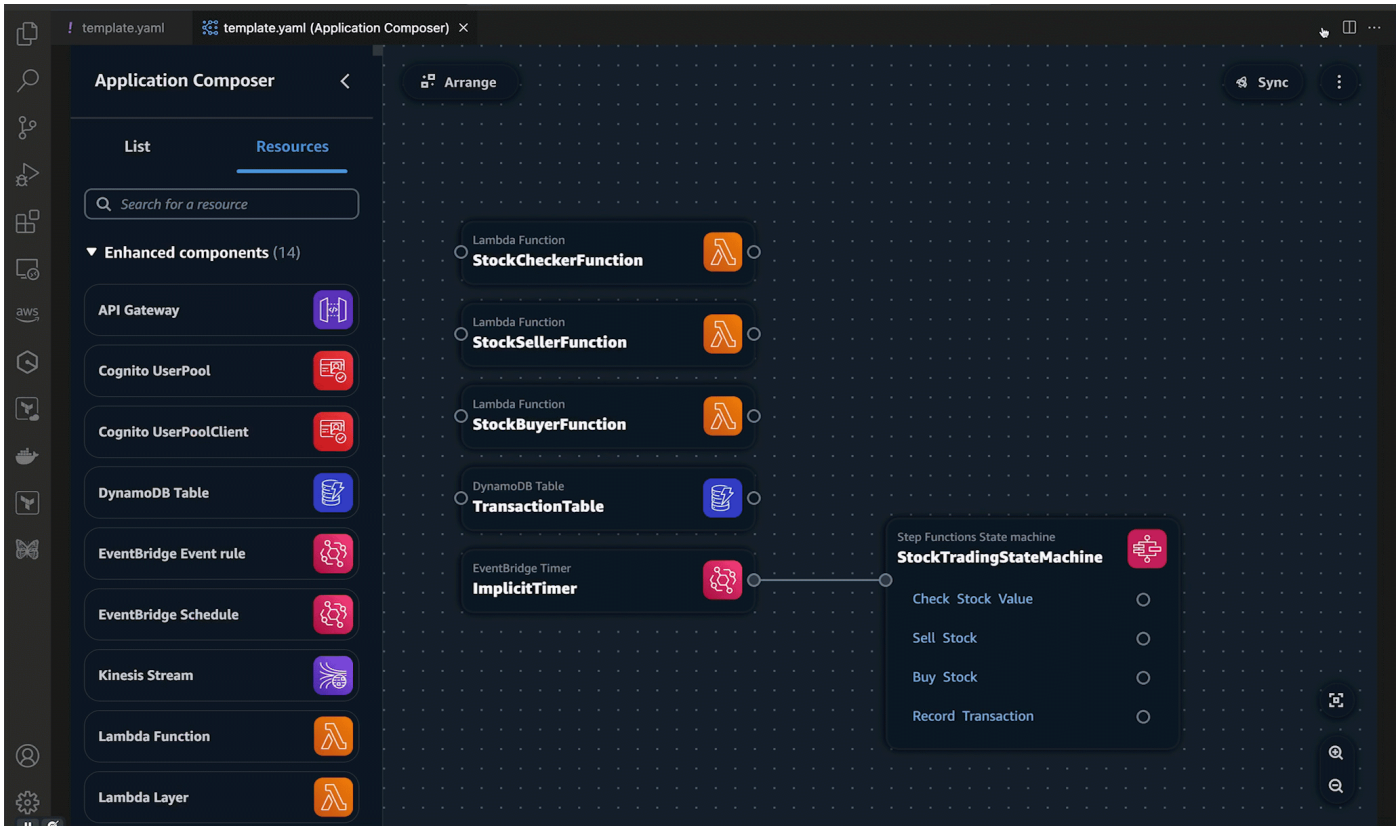
AWS Management Console

Access Application Composer through the AWS Management Console to get started quickly. Use **local sync** mode to automatically sync and save Application Composer with your local machine.



AWS Toolkit for Visual Studio Code

Access Application Composer through the Toolkit for VS Code extension to bring Application Composer into your local development environment.



Benefits of Application Composer

With Application Composer, you can focus on designing *what* you want to accomplish and let Application Composer build *how* it is accomplished. Use Application Composer to create deployable code without needing to be an expert in AWS CloudFormation.

Learn more

To continue learning about Application Composer, see the following:

- Getting started with Application Composer.
- Visually design and build serverless applications | Serverless Office Hours – Overview and demo of Application Composer.

Next steps

To set up Application Composer, see Setting up Application Composer.

Setting up Application Composer

Before using AWS Application Composer for the first time, complete the tasks in this section to get set up.

> **Note**
>
> Access to Application Composer from the AWS Management Console requires, at minimum, read-only access to the AWS Management Console. If you're an existing AWS user and meet those requirements, see Accessing Application Composer. If you don't have an AWS account, then we recommend that you complete the following steps to access Application Composer.

Topics

- Sign up for an AWS account
- Create an administrative user
- Accessing Application Composer
- Next steps

Sign up for an AWS account

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

To sign up for an AWS account

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

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

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

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

Create an administrative user

After you sign up for an AWS account, create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

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

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

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

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

Create an administrative user

- For your daily administrative tasks, grant administrative access to an administrative user in AWS IAM Identity Center.

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

Sign in as the administrative user

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

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

Accessing Application Composer

You can access Application Composer through the AWS Management Console or the AWS Toolkit for Visual Studio Code.

Accessing Application Composer from the AWS Management Console

Use any modern web browser. For the best experience, we recommend using Google Chrome or Microsoft Edge, which both support the Application Composer **local sync** mode. This mode requires a browser that supports the File System Access API, which allows web applications to read, write, and save files in your local file system. For more information about the File System Access API, see File System Access API.

To access Application Composer through the AWS Management Console

1. Sign in to the AWS Management Console with an AWS account.
2. In the navigation bar, choose an AWS Region.
3. In the navigation bar, search for and choose **Application Composer**.

Accessing Application Composer from the AWS Toolkit for Visual Studio Code

To install Application Composer from the Toolkit for VS Code

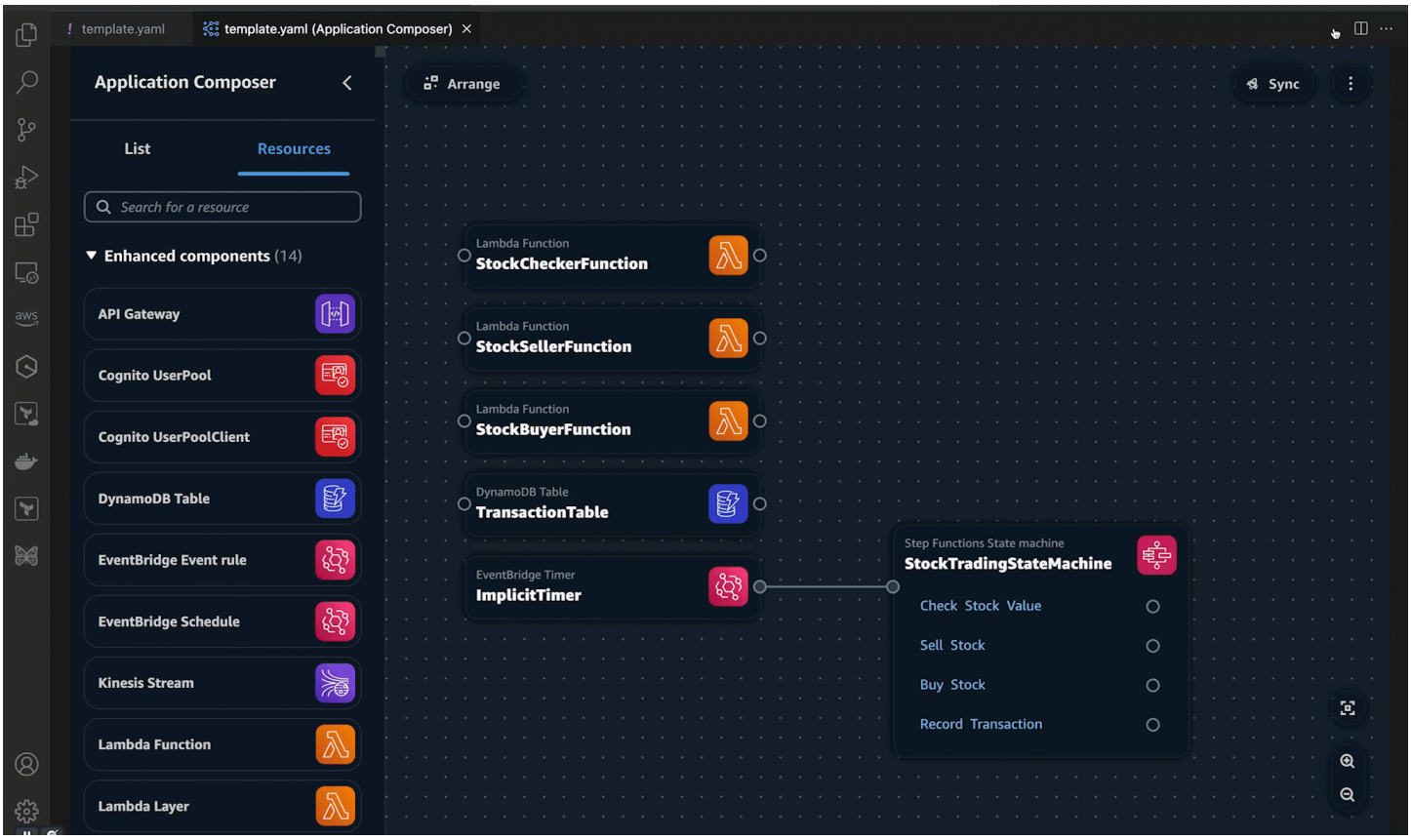
- Download and install the Toolkit for VS Code. For instructions, see Downloading the Toolkit for VS Code.

To access Application Composer from the Toolkit for VS Code

You can access Application Composer in any of the following ways:

1. By selecting the Application Composer button from any AWS CloudFormation or AWS SAM template.
2. Through the context menu by right-clicking on your AWS CloudFormation or AWS SAM template.
3. From the VS Code Command Palette.

The following is an example of accessing Application Composer from the Application Composer button:



For more information on accessing Application Composer, see Accessing AWS Application Composer from the Toolkit.

Next steps

To get started with using Application Composer, see Getting started with Application Composer.

Getting started with Application Composer

Get started with AWS Application Composer by learning how to use its visual canvas to design an application.

Topics

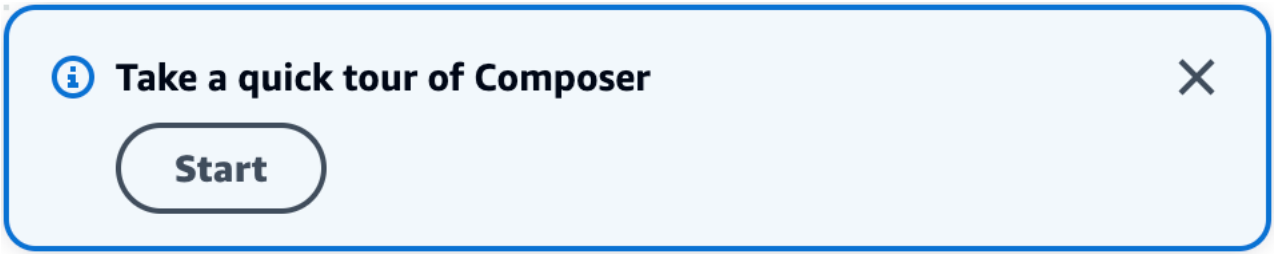
- Take a tour of Application Composer
- Tutorial 1: Load and modify the Application Composer demo project
- Tutorial 2: Build your first application with Application Composer

Take a tour of Application Composer

Go through an embedded tour of AWS Application Composer to learn what you can do with the service.

To take a tour of Application Composer

1. Sign in to the Application Composer console.
2. On the **Home** page, choose **Open demo**.
3. In the upper-right corner, in the **Take a quick tour of Composer** window, choose **Start**.



4. In the **Composer tour** window, do the following:
   - To move to the next step, choose **Next**.
   - To return to the previous step, choose **Previous**.
   - On the final step, to finish the tour, choose **End**.

Tutorial 1: Load and modify the Application Composer demo project

This tutorial guides you through creating a demo application to learn the user interface of AWS Application Composer.

For this tutorial, we use Application Composer in the AWS Management Console.

Topics

- Step 1: Open the demo
- Step 2: Explore the visual canvas of Application Composer
- Step 3: Expand your application architecture
- Step 4: Save your application
- Next steps

Step 1: Open the demo

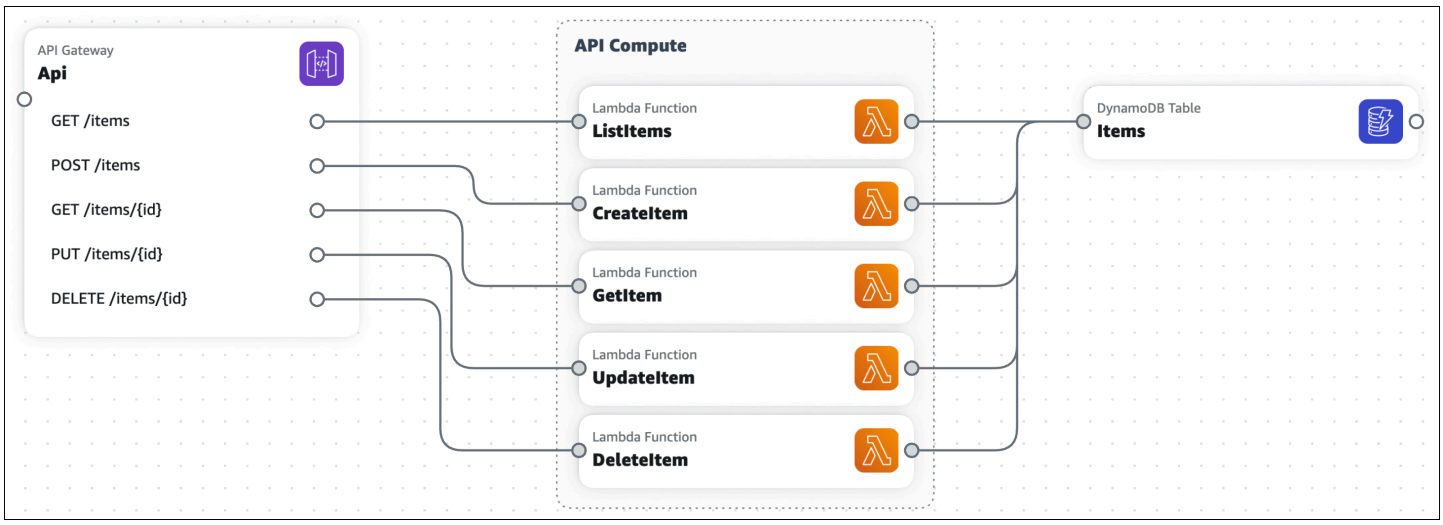
Start using Application Composer by creating a demo project.

To create a demo project

1. Sign in to the Application Composer console.
2. On the **Home** page, choose **Open demo**.

The demo application is a basic CRUD serverless application that includes:

- An Amazon API Gateway resource with five routes.
- Five AWS Lambda functions.
- An Amazon DynamoDB table.

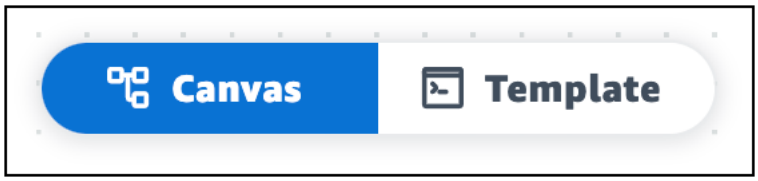


Step 2: Explore the visual canvas of Application Composer

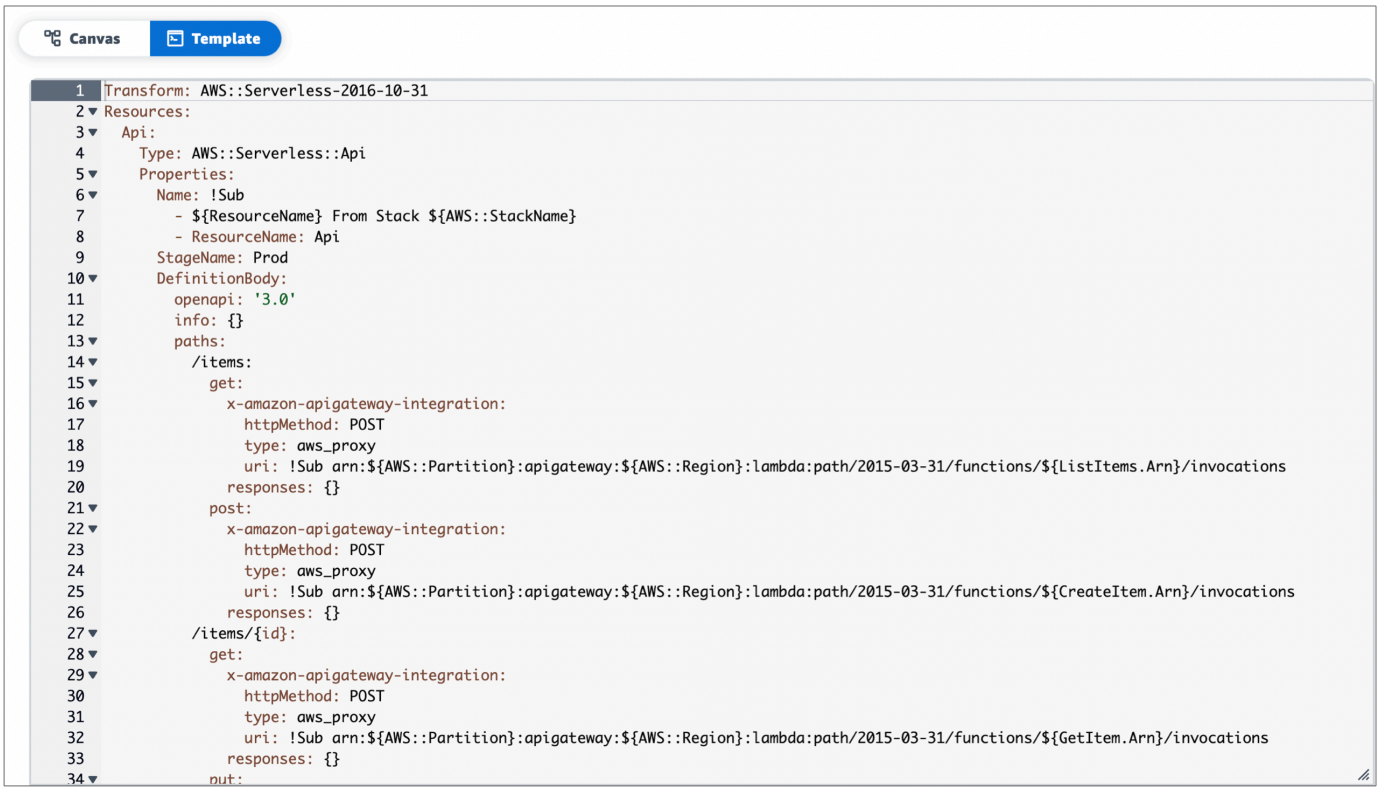
Learn the features of the visual canvas to build out your Application Composer demo project. For an overview of the visual canvas layout, see Visual overview.

To explore the features of the visual canvas

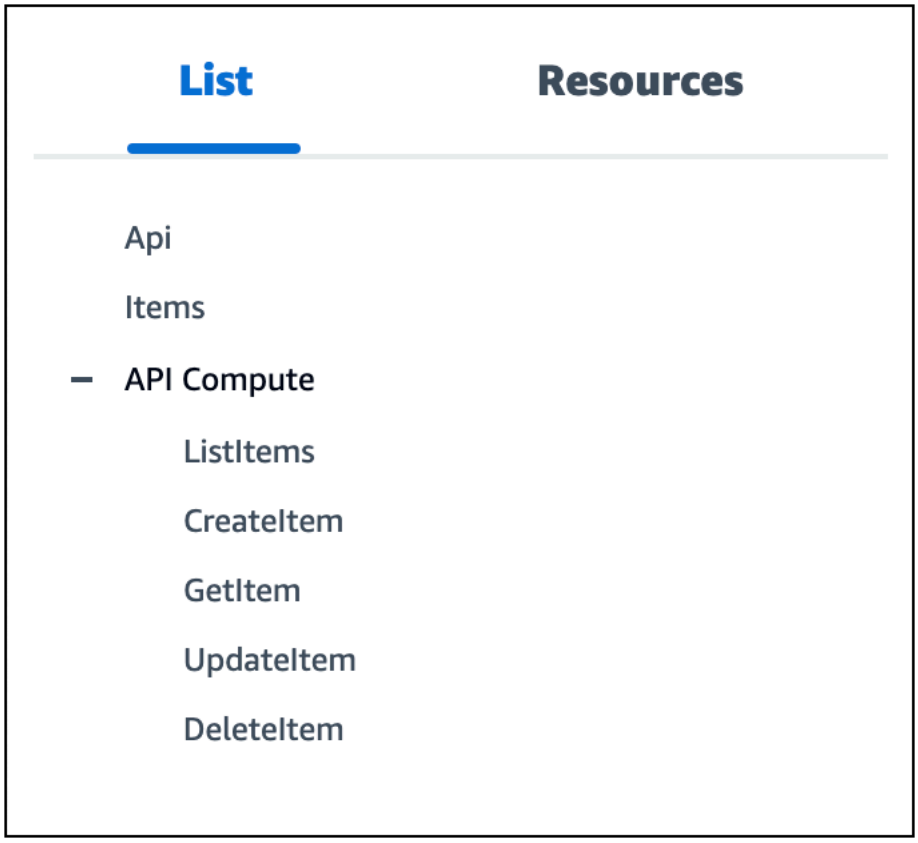
1. When you open a new or existing application project, Application Composer loads the canvas view, as indicated above the main view area.

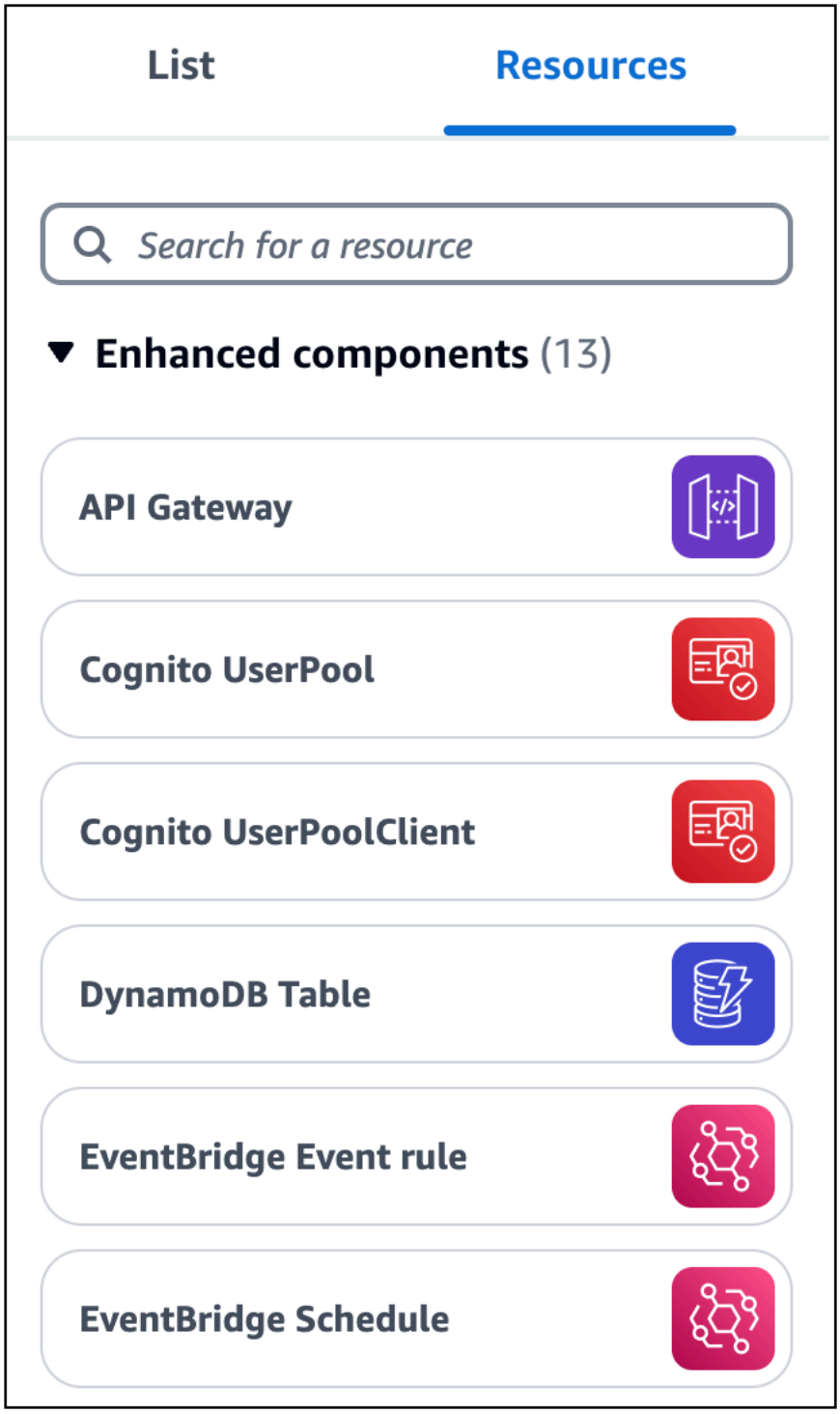
   To show your application's infrastructure code in the main view area, choose **Template**. For example, here is the AWS Serverless Application Model (AWS SAM) template view of the Application Composer demo project.



2. To show the canvas view of your application again, choose **Canvas**.
3. To show your application's resources organized in a tree view, choose **List**.



4. To show the resource palette, choose **Resources**. This palette features cards that you can use to expand your application architecture. You can search for cards or scroll through the list.



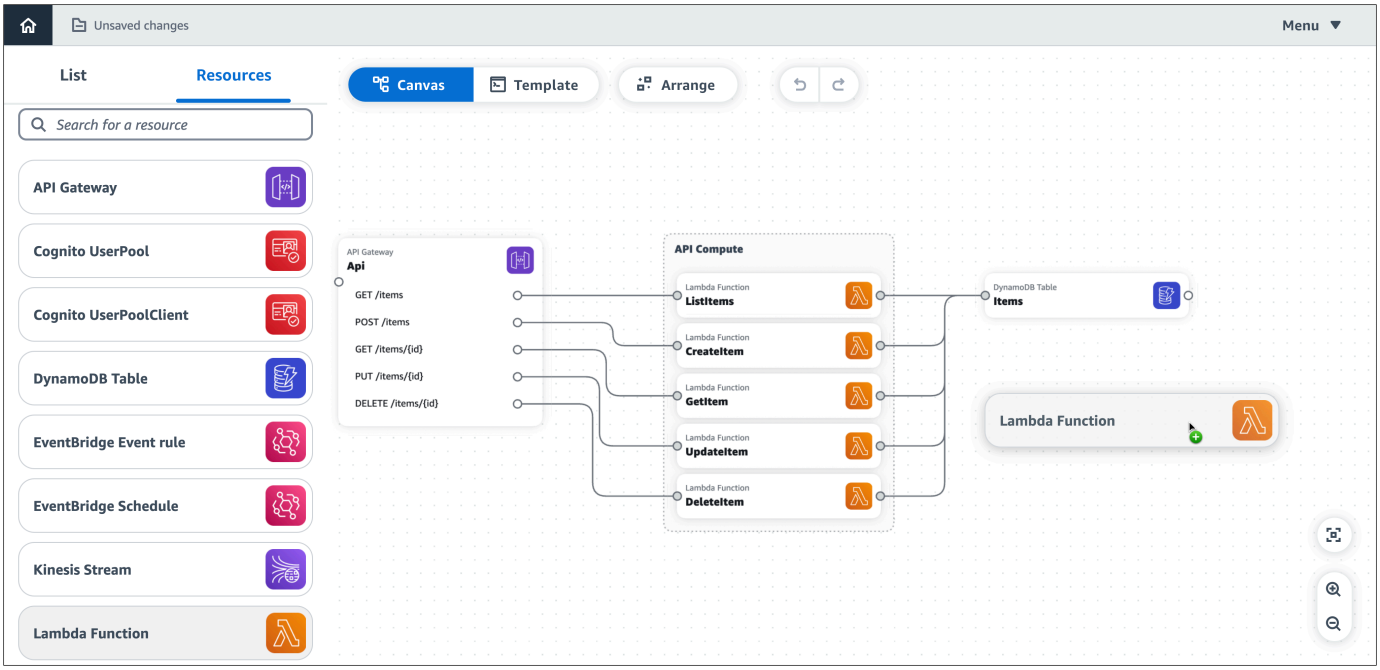
5. To move around the visual canvas, use basic gestures. For more information, see Gestures and shortcuts.

Step 3: Expand your application architecture

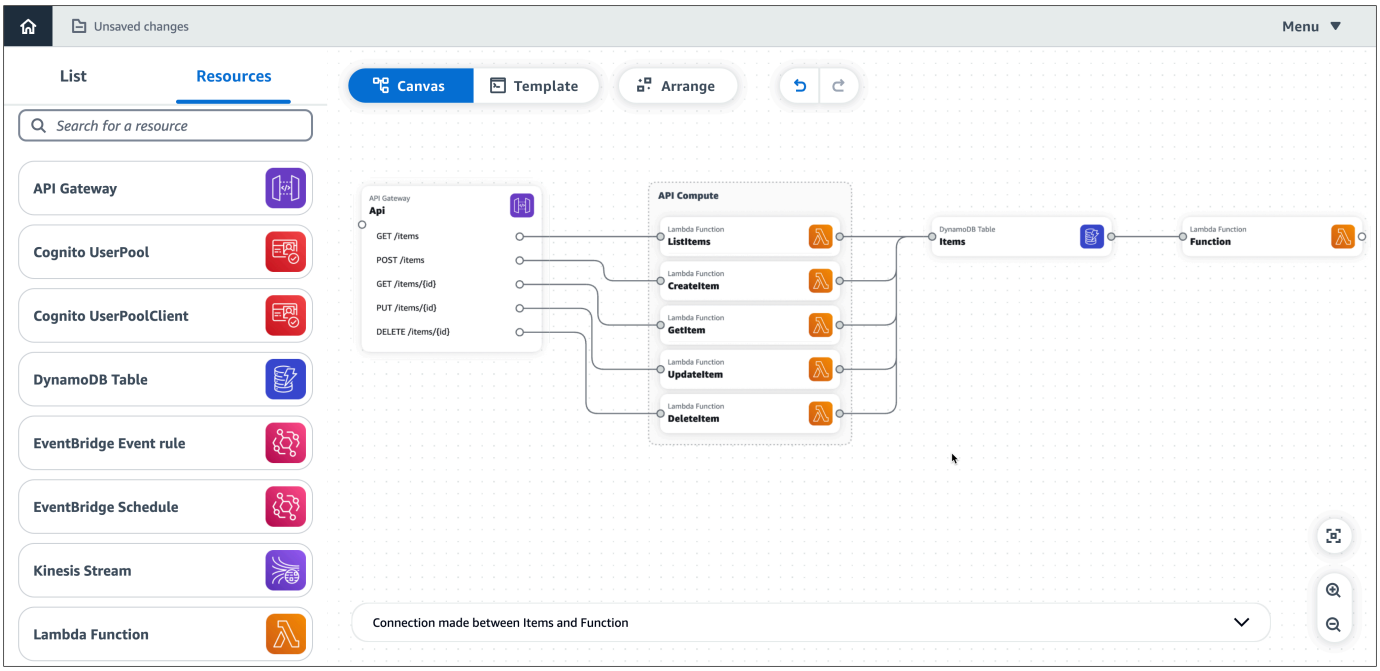
In this step, you will expand your application architecture by adding a Lambda function to your DynamoDB table.

To add a Lambda function to your DynamoDB table

1. From the resource palette (**Resources**), drag the **Lambda Function** enhanced component card onto the canvas, to the right of the **DynamoDB Table** card.



2. Connect the DynamoDB table to the Lambda function. To connect them, click the right port of the **DynamoDB Table** card and drag it onto the left port of the **Lambda Function** card.
3. Choose **Arrange** to organize the cards in the canvas view.



4. Configure your Lambda function. To configure it, do either of the following:

   - In the canvas view, modify the function's properties on the **Resource properties** panel. To open the panel, double-click the **Lambda Function** card. Or, select the card, and then choose **Details**. For more information about the configurable Lambda function properties listed in the **Resource properties** panel, see the AWS Lambda Developer Guide.
   - In the template view, modify the code for your function (`AWS::Serverless::Function`). Application Composer automatically syncs your changes to the canvas. For more information about the function resource in an AWS SAM template, see AWS::Serverless::Function in the *AWS SAM resource and property reference*.

Step 4: Save your application

Save your application by manually saving your application template to your local machine or by activating **local sync**.

To manually save your application template

1. From the **menu**, select **Save** > **Save template file**.
2. Provide a name for your template and choose a location on your local machine to save your template. Press **Save**.

For instructions on activating **local sync**, see Automatically sync and save your project.

Next steps

To get started with building your first application, see Tutorial 2: Build your first application.

Tutorial 2: Build your first application with Application Composer

In this tutorial, you use AWS Application Composer to create a CRUD serverless application that manages users in a database.

For this tutorial, we use Application Composer in the AWS Management Console. We recommend that you use Google Chrome or Microsoft Edge, and a full-screen browser window.

Are you new to serverless?

We recommend a basic understanding of the following topics:

- Event-driven architecture
- Infrastructure as Code (IaC)
- Serverless technologies

To learn more, see Serverless concepts.

Topics

- Resource properties reference
- Step 1: Create your project
- Step 2: Add cards to the canvas
- Step 3: Configure your API Gateway REST API
- Step 4: Configure your Lambda functions
- Step 5: Connect your cards
- Step 6: Organize the canvas

- Step 7: Add and connect a DynamoDB table
- Step 8: Review your AWS CloudFormation template
- Step 9: Integrate into your development workflows
- Next steps

Resource properties reference

While building your application, use this table for reference to configure the properties of your Amazon API Gateway and AWS Lambda resources.

Method	Path	Function name
GET	/items	getItems
GET	/items/{id}	getItem
PUT	/items/{id}	updateItem
POST	/item	addItem
DELETE	/items/{id}	deleteItem

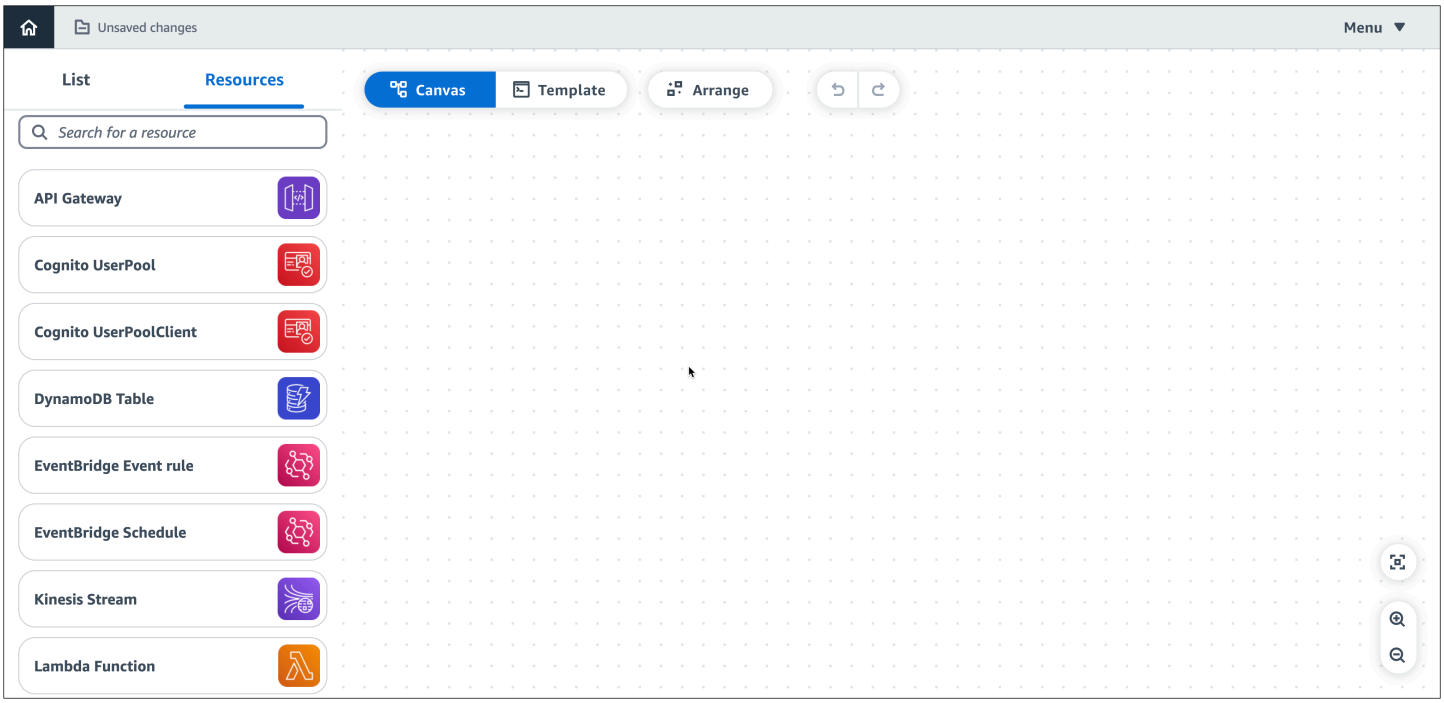
Step 1: Create your project

To get started with your CRUD serverless application, create a new project in Application Composer and activate **local sync**.

To create a new blank project

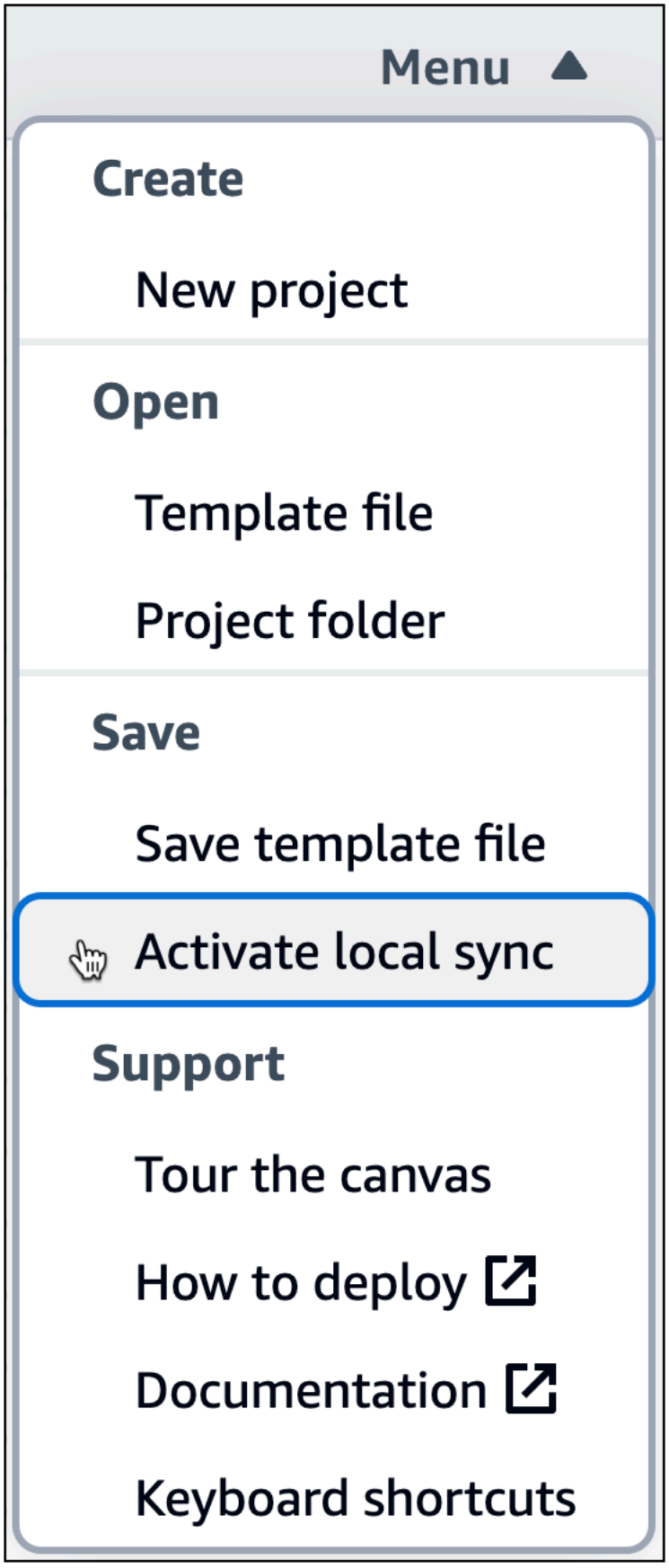
1. Sign in to the Application Composer console.
2. On the **Home** page, choose **Create project**.

Application Composer loads a starting application template and opens the visual canvas.



To activate local sync

1. From the Application Composer **menu**, select **Save** > **Activate local sync**.


Menu
Create
New project
Open
Template file
Project folder
Save
Save template file
Activate local sync
Support
Tour the canvas
How to deploy
Documentation
Keyboard shortcuts

2. For **Project location**, press **Select folder** and choose a directory. This is where Application Composer will save and sync your template files and folders as you design.

   The project location must not contain an existing application template.

   > **Note**
   >
   > **Local sync** requires a browser that supports the File System Access API. For more information, see What is the File System Access API?.

3. When prompted to allow access, select **View files**.
4. Press **Activate** to turn on **local sync**. When prompted to save changes, select **Save changes**.

   When activated, the **Autosave** indicator will be displayed in the upper-left area of your canvas.

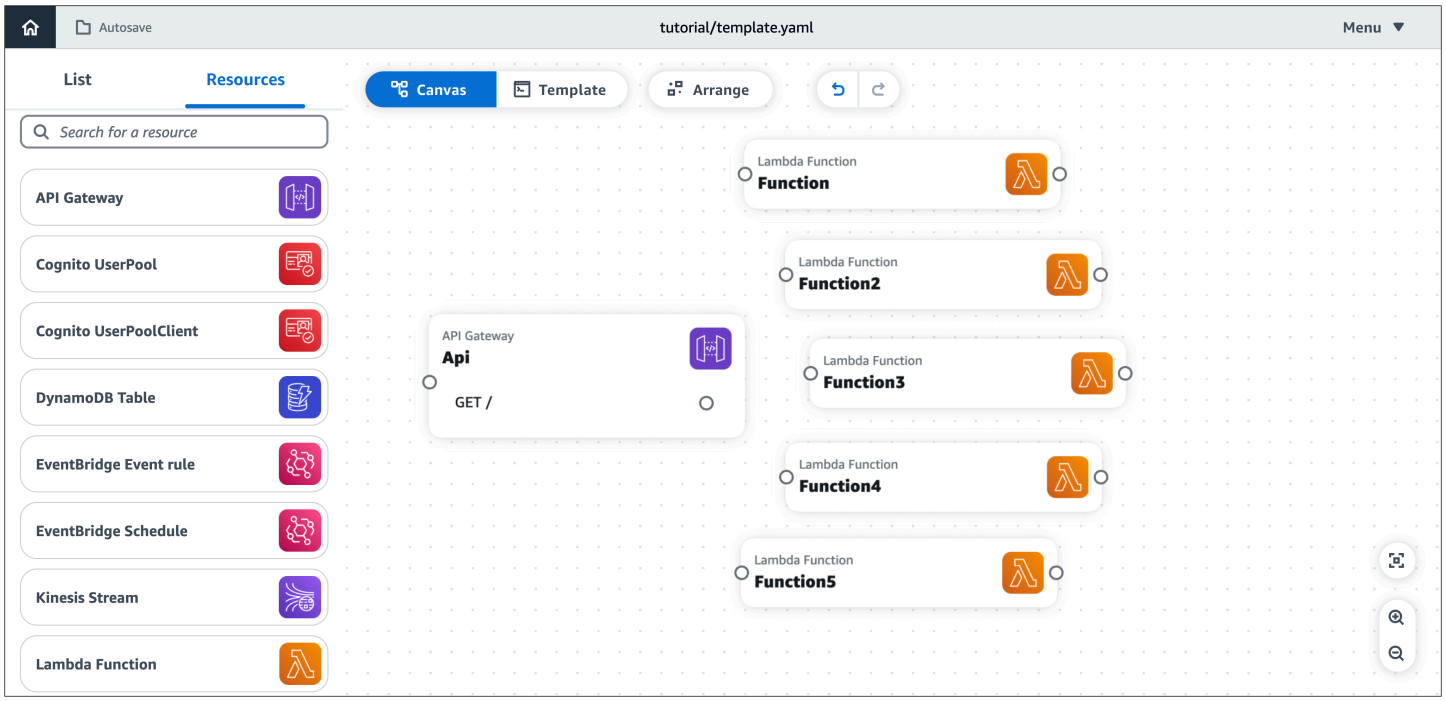
Step 2: Add cards to the canvas

Start to design your application architecture using enhanced component cards, beginning with an API Gateway REST API and five Lambda functions.

To add API Gateway and Lambda cards to the canvas

From the **Resources** palette, under the **Enhanced components** section, do the following:

1. Drag an **API Gateway** card onto the canvas.
2. Drag a **Lambda Function** card onto the canvas. Repeat until you've added five **Lambda Function** cards to the canvas.



Step 3: Configure your API Gateway REST API

Next, add five routes within your API Gateway card.

To add routes to the API Gateway card

1. Open the **Resource properties** panel for the **API Gateway** card. To open the panel, double-click the card. Or, select the card, and then choose **Details**.
2. In the **Resource properties** panel, under **Routes**, do the following:

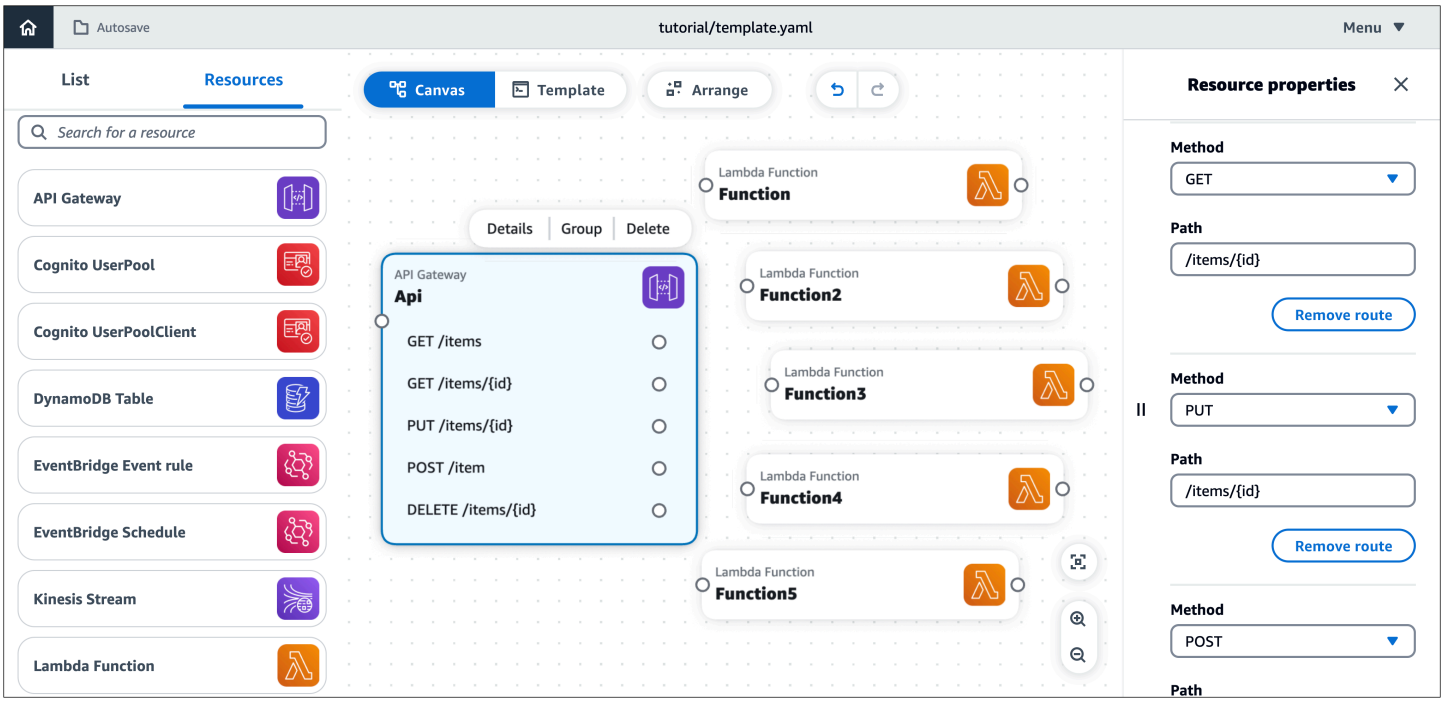
   > **Note**
   >
   > For each of the following routes, use the HTTP method and path values specified in the resource properties reference table.

   a. For **Method**, choose the specified HTTP method. For example, **GET**.

   b. For **Path**, enter the specified path. For example, **`/items`**.

   c. Choose **Add route**.

   d. Repeat the previous steps until you've added all five specified routes.
3. Choose **Save**.

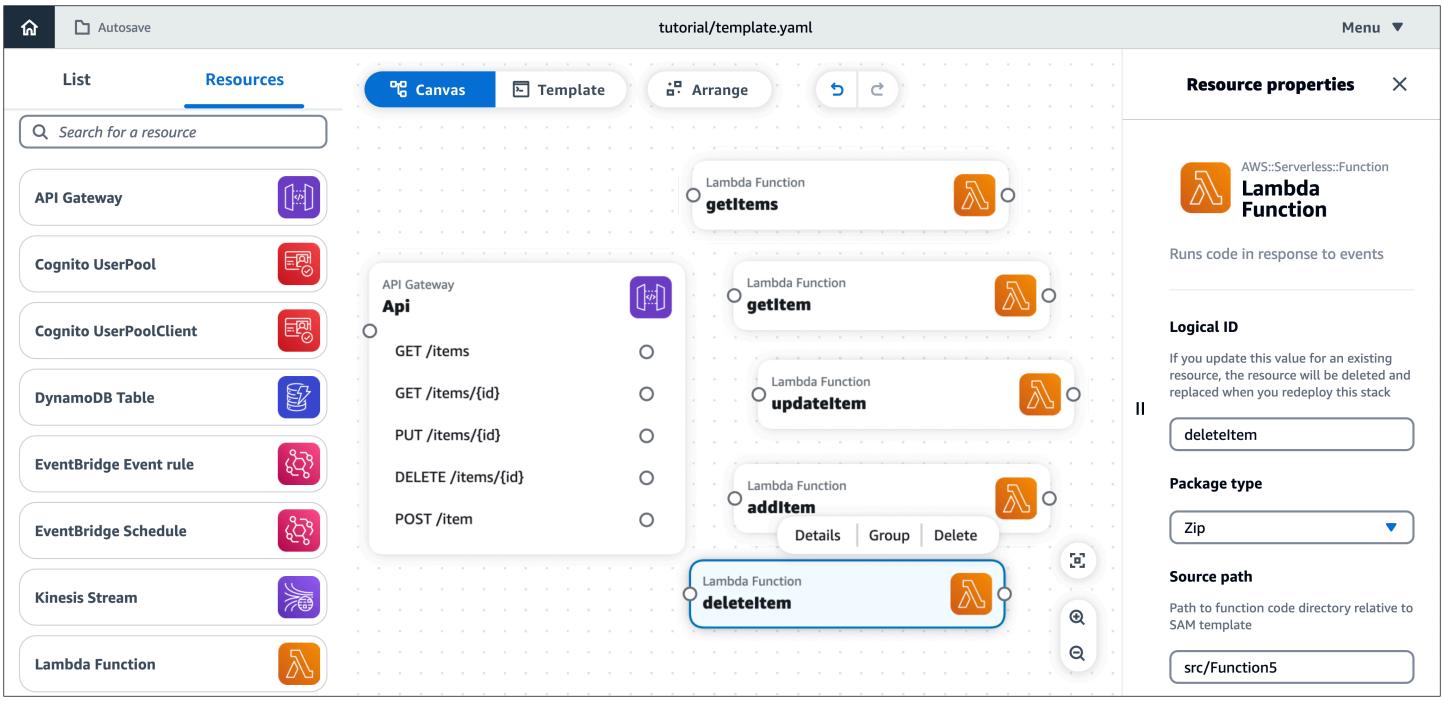


Step 4: Configure your Lambda functions

Name each of the five Lambda functions as specified in the resource properties reference table.

To name the Lambda functions

1. Open the **Resource properties** panel of a **Lambda Function** card. To open the panel, double-click the card. Or, select the card, and then choose **Details**.
2. In the **Resource properties** panel, for **Logical ID**, enter a specified function name. For example, **`getItems`**.
3. Choose **Save**.
4. Repeat the previous steps until you've named all five functions.

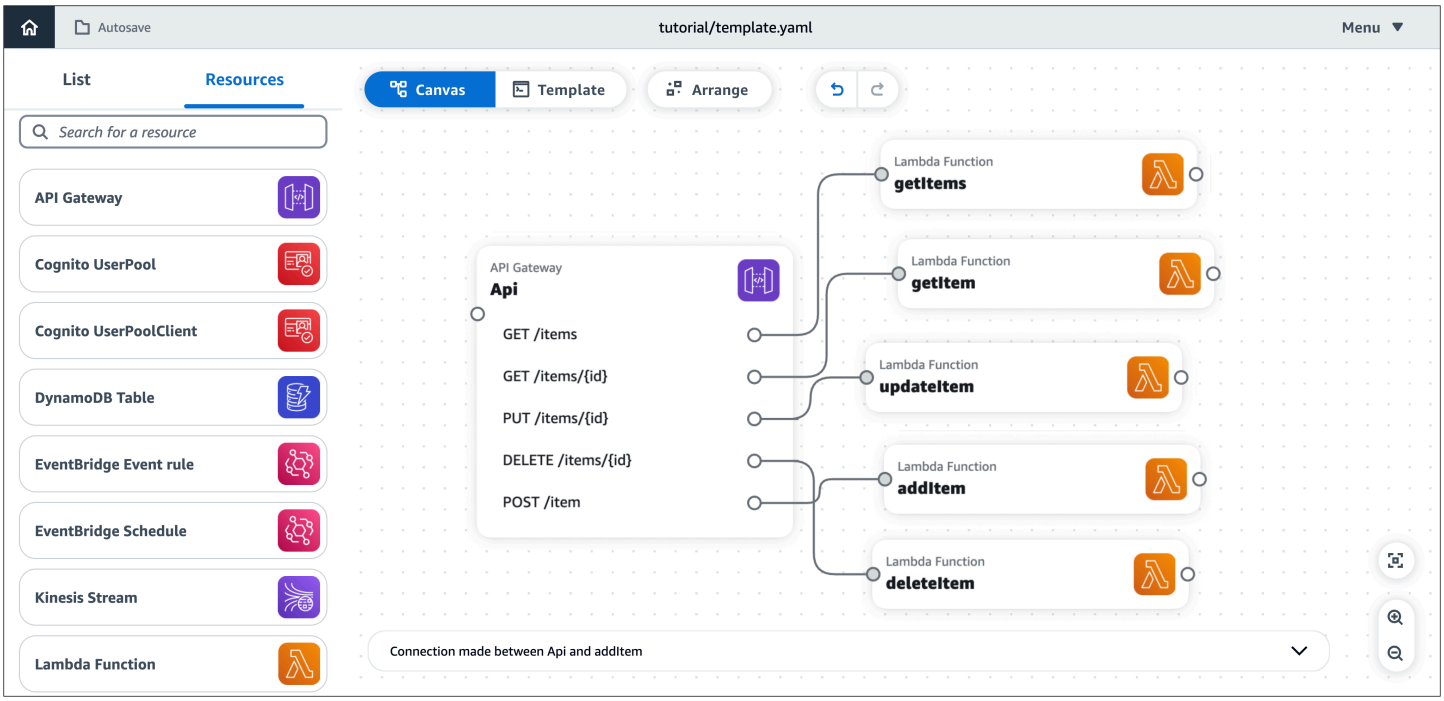


Step 5: Connect your cards

Connect each route on your **API Gateway** card to its related **Lambda Function** card, as specified in the resource properties reference table.

To connect your cards

1. Click a right port on the **API Gateway** card and drag it to the left port of the specified **Lambda Function** card. For example, click the **GET /items** port and drag it to the left port of **getItems**.
2. Repeat the previous step until you've connected all five routes on the **API Gateway** card to corresponding **Lambda Function** cards.



Step 6: Organize the canvas

Organize the visual canvas by grouping together your Lambda functions and arranging all the cards.

To group together your functions

1. Press and hold **Shift**, then select each **Lambda Function** card on the canvas.
2. Choose **Group**.

To name your group

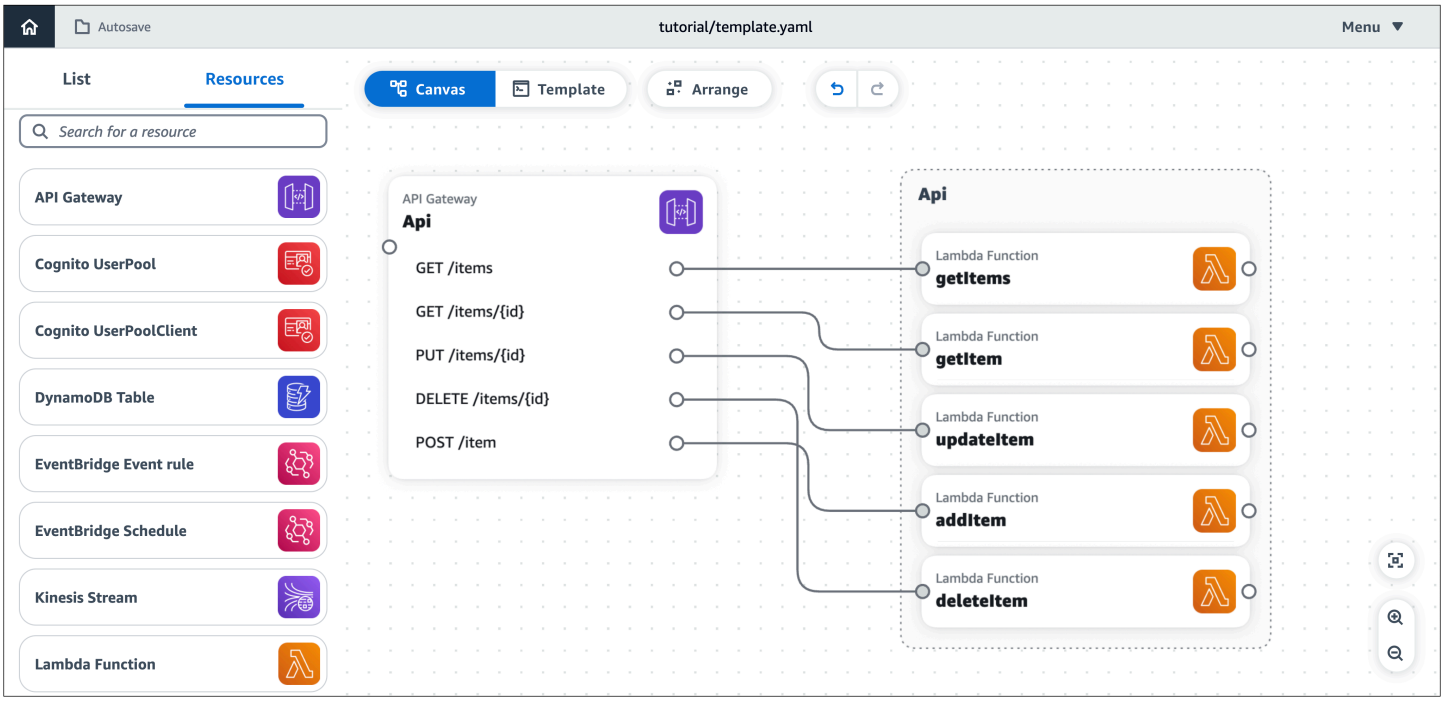
1. Double-click the top of the group, near the group name (**Group**).

   The **Group properties** panel opens.
2. On the **Group properties** panel, for **Group name**, enter **`API`**.
3. Choose **Save**.

To arrange your cards

On the canvas, above the main view area, choose **Arrange**.

Application Composer arranges and aligns all cards on the visual canvas, including your new group (**API**), as shown here:

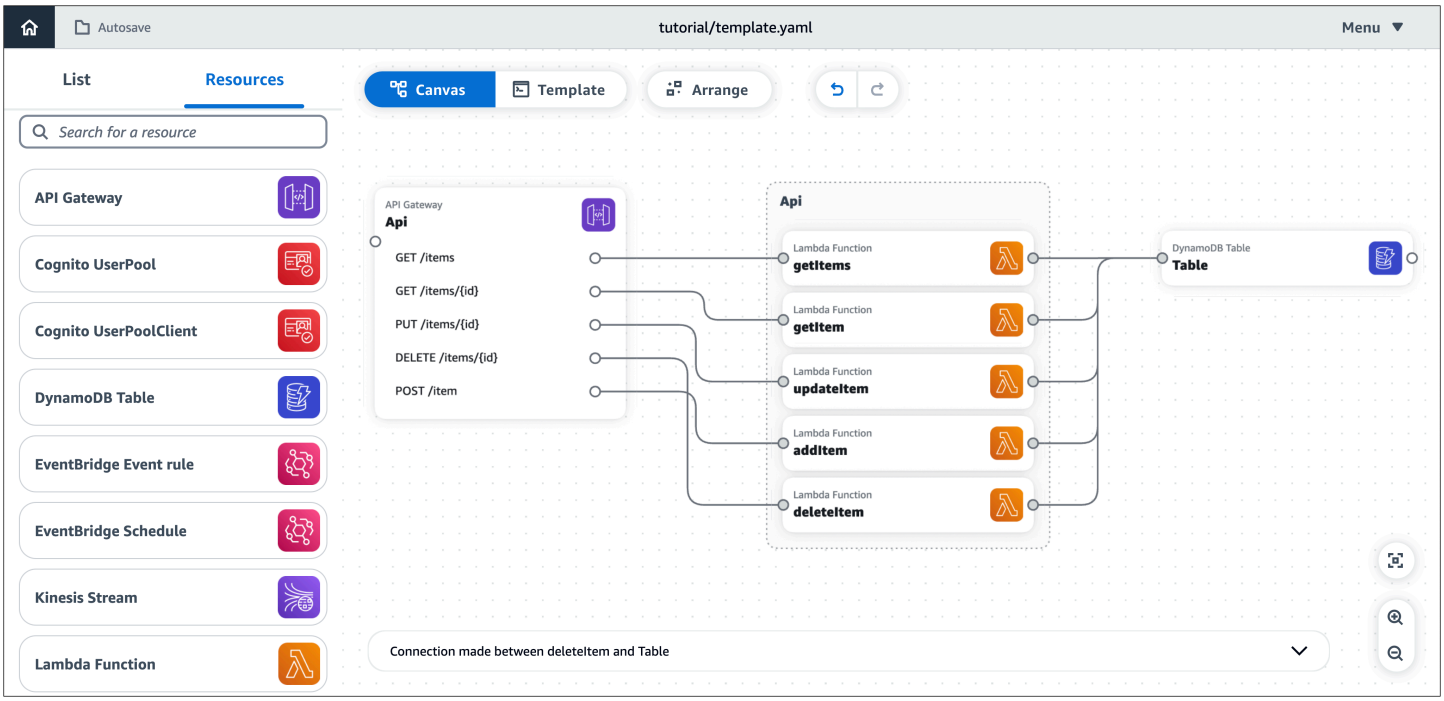


Step 7: Add and connect a DynamoDB table

Now, add a DynamoDB table to your application architecture and connect it to your Lambda functions.

To add and connect a DynamoDB table

1. From the resource palette (**Resources**), under the **Enhanced components** section, drag a **DynamoDB Table** card onto the canvas.
2. Click the right port on a **Lambda Function** card and drag it to the left port of the **DynamoDB Table** card.
3. Repeat the previous step until you've connected all five **Lambda Function** cards to the **DynamoDB Table** card.
4. (Optional) To reorganize and realign the cards on the canvas, choose **Arrange**.

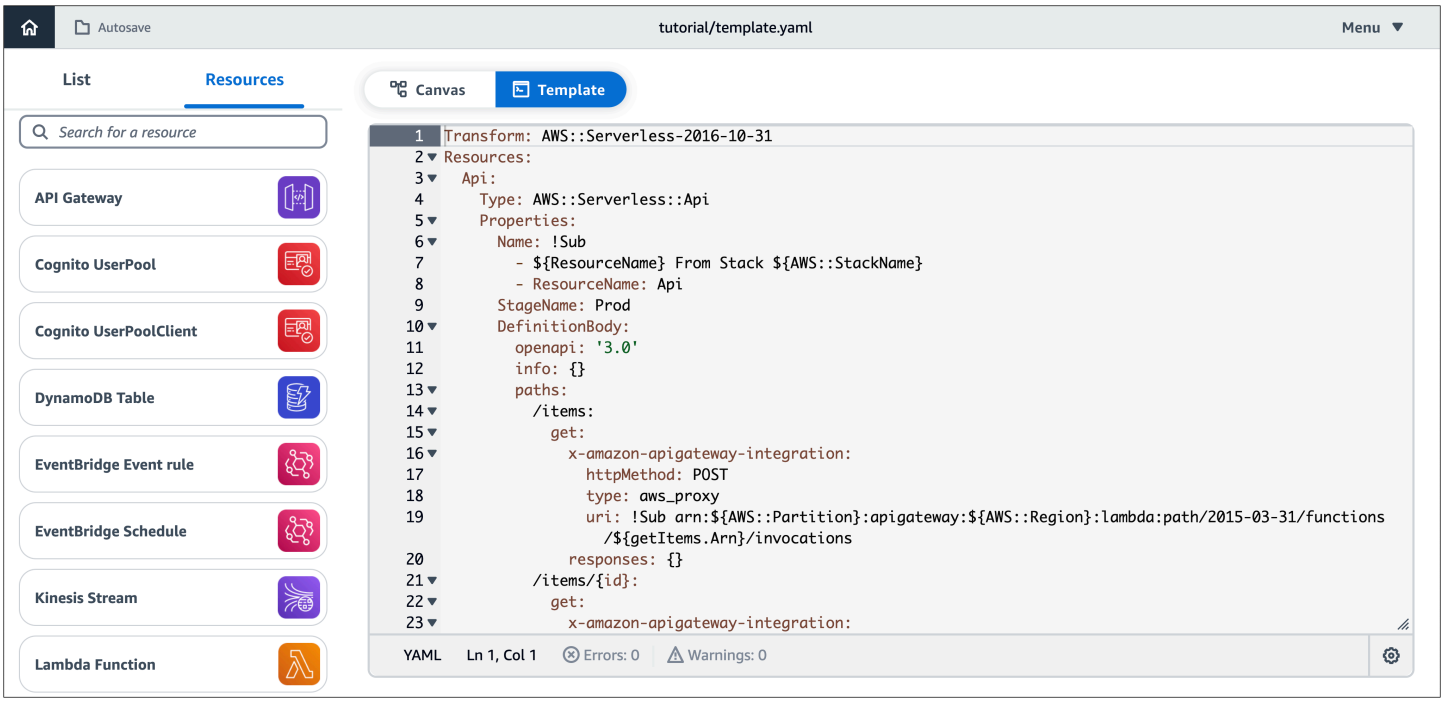


Step 8: Review your AWS CloudFormation template

Congratulations! You've successfully designed a serverless application that's ready for deployment. Finally, choose **Template** to review the AWS CloudFormation template that Application Composer has automatically generated for you.

In the template, Application Composer has defined the following:

- The `Transform` declaration, which specifies the template as an AWS Serverless Application Model (AWS SAM) template. For more information, see AWS SAM template anatomy in the *AWS Serverless Application Model Developer Guide*.
- An `AWS::Serverless::Api` resource, which specifies your API Gateway REST API with its five routes.
- Five `AWS::Serverless::Function` resources, which specify your Lambda functions' configurations, including their environment variables and permissions policies.
- An `AWS::DynamoDB::Table` resource, which specifies your DynamoDB table and its properties.
- The `Metadata` section, which contains information about your resource group (**API**). For more information about this section, see Metadata in the *AWS CloudFormation User Guide*.



Step 9: Integrate into your development workflows

Use the template file and project directories that Application Composer created for further testing and deployment.

- With **local sync**, you can connect Application Composer to the IDE on your local machine to speed up development. To learn more, see Using Application Composer with your local IDE.
- With **local sync**, you can use the AWS Serverless Application Model Command Line Interface (AWS SAM CLI) on your local machine to test and deploy your application. To learn more, see Using Application Composer with AWS CloudFormation and AWS SAM.

Next steps

You're now ready to build your own applications with Application Composer.

Using Application Composer to design and build your applications

Use AWS Application Composer to visually design your application architecture through its interactive canvas.

Topics

- Using AWS Application Composer basics
- Using Application Composer with featured AWS services
- Using AWS Application Composer from the AWS Management Console
- Using AWS Application Composer from the AWS Toolkit for Visual Studio Code

Using AWS Application Composer basics

This section covers the basics of using Application Composer from the AWS Management Console or the AWS Toolkit for Visual Studio Code.

Topics

- Using gestures and keyboard shortcuts in Application Composer
- Designing with cards in Application Composer
- Connect cards together using the Application Composer visual canvas
- View code updates with the Change Inspector
- Working with templates that reference external files

Using gestures and keyboard shortcuts in Application Composer

Use gestures and keyboard shortcuts in AWS Application Composer.

Gestures and keyboard shortcuts are supported in both the console and toolkit versions of Application Composer.

Topics

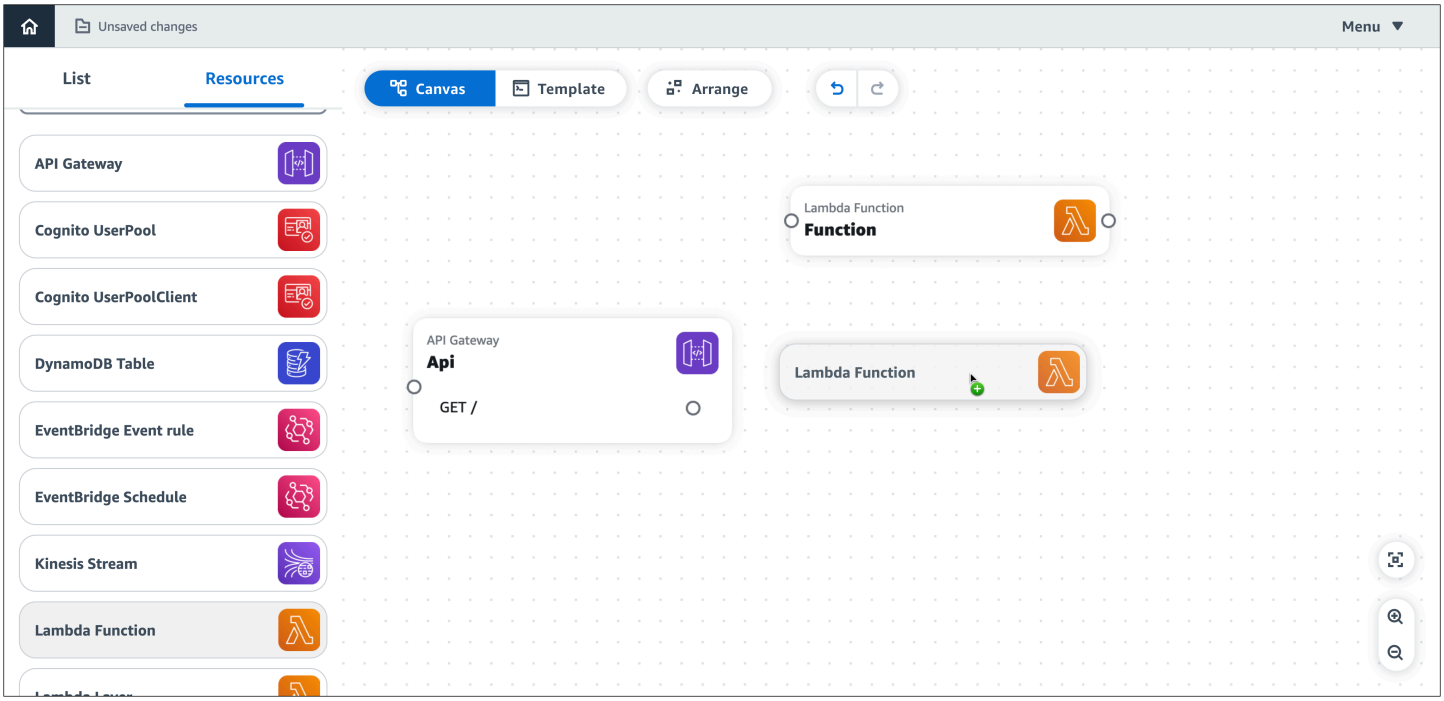
- Gestures

- Keyboard shortcuts

Gestures

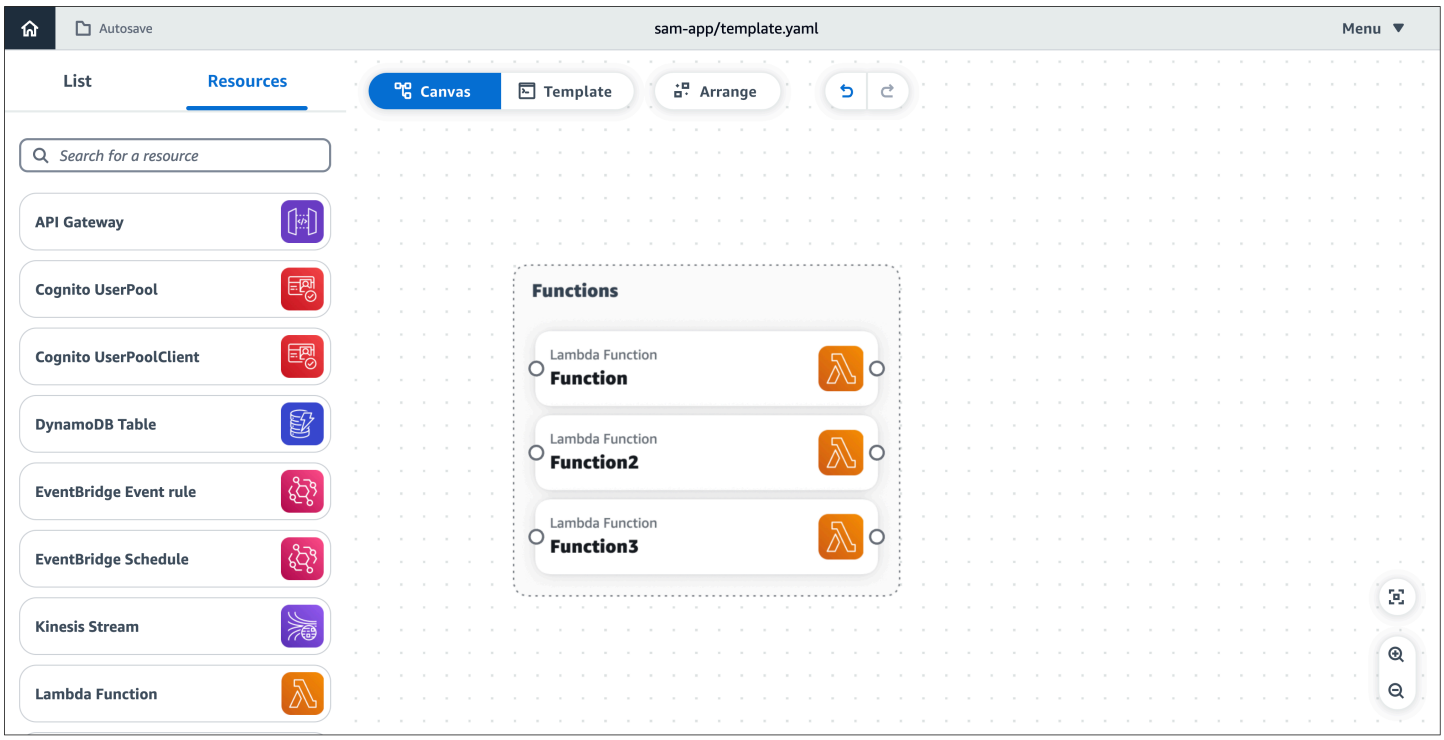
Select a card to design with

To add a card to your application, select it from the resource palette and drag it onto the canvas.



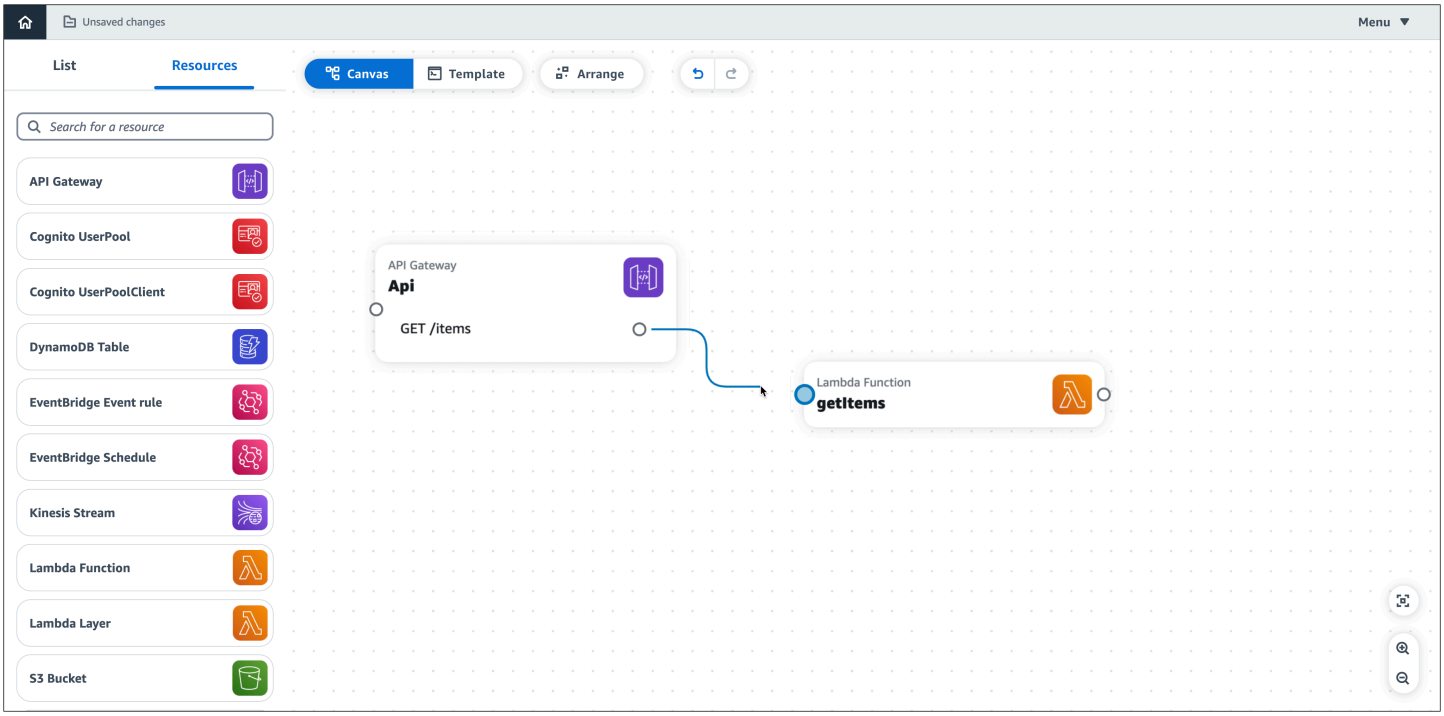
Group cards together

While pressing **Shift**, select cards to group. Then, choose **Group** from the resource actions menu.



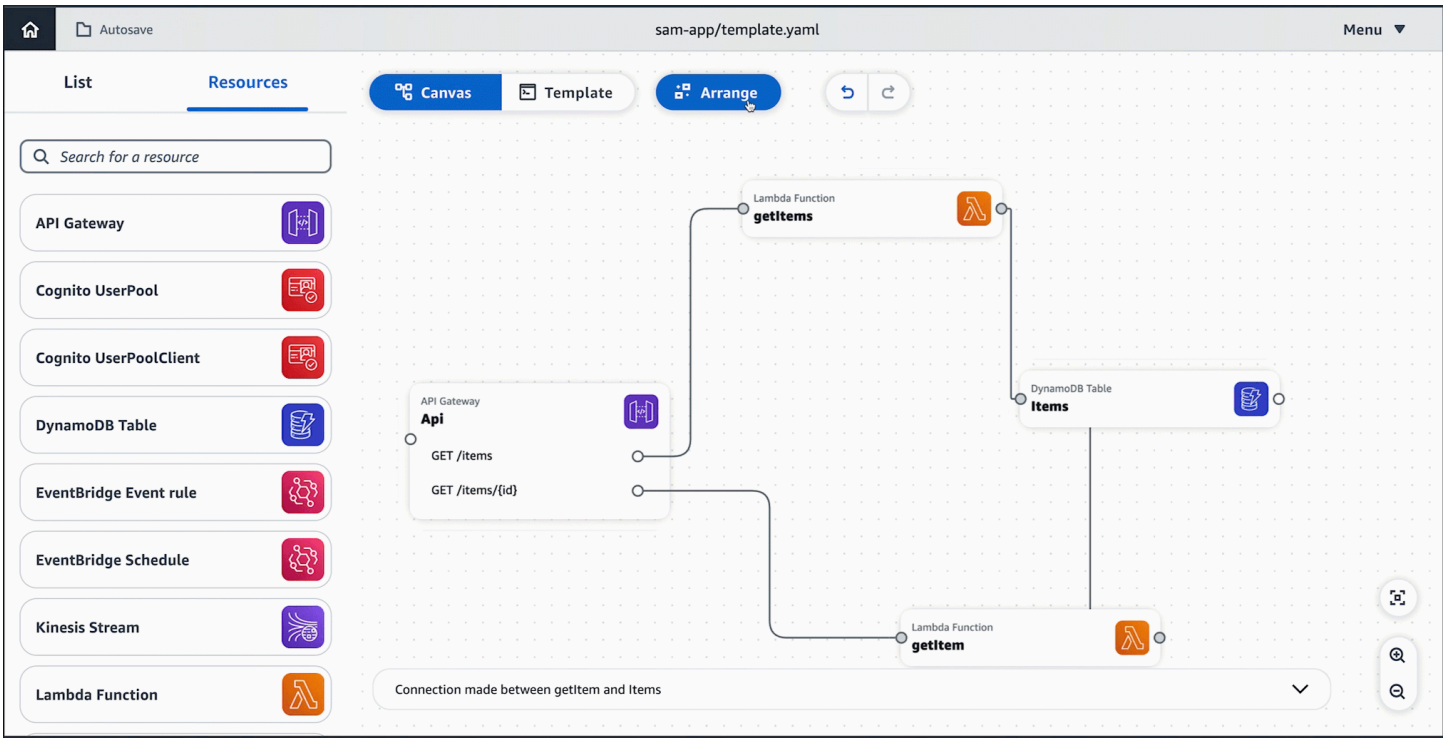
Connect cards together

Click on a port from one card and drag it onto a port on another card.



Arrange cards on your canvas

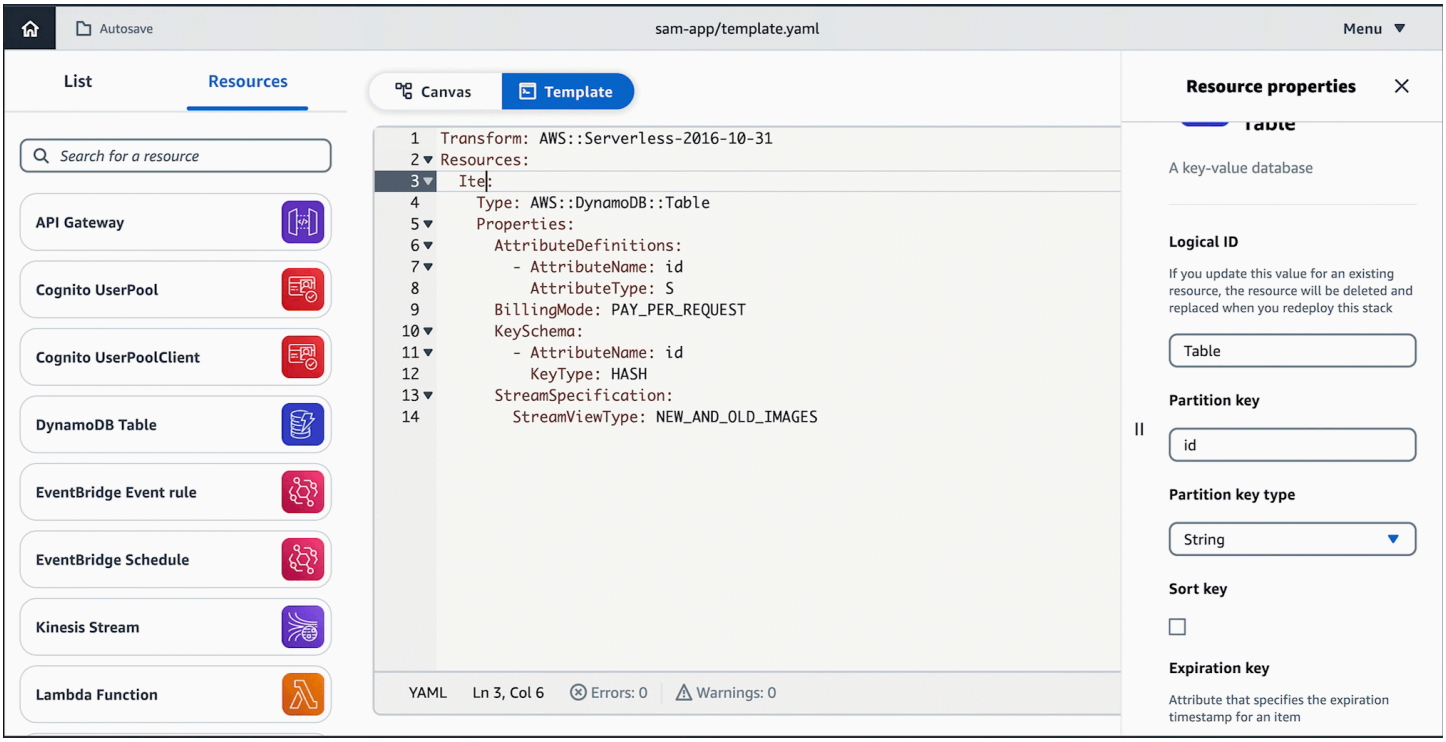
Select **Arrange** to visually arrange and organize the canvas.



Configure card properties

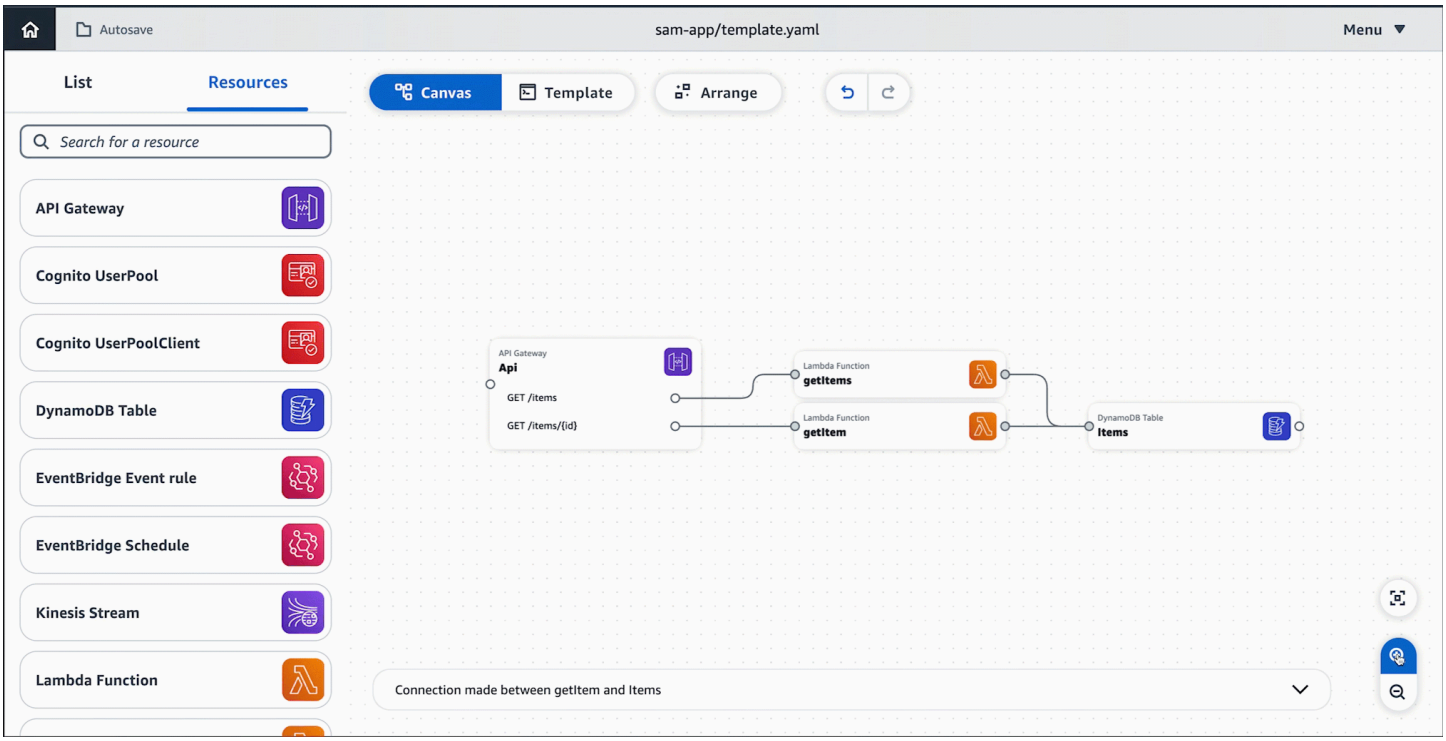
Do any of the following to configure card properties:

- Double-click on a card to bring up the resource properties panel.
- Click on a card and select **Details** to bring up the resource properties panel.
- For Application Composer from the AWS Management Console, select **Template** to show your application code. Configure directly from here.



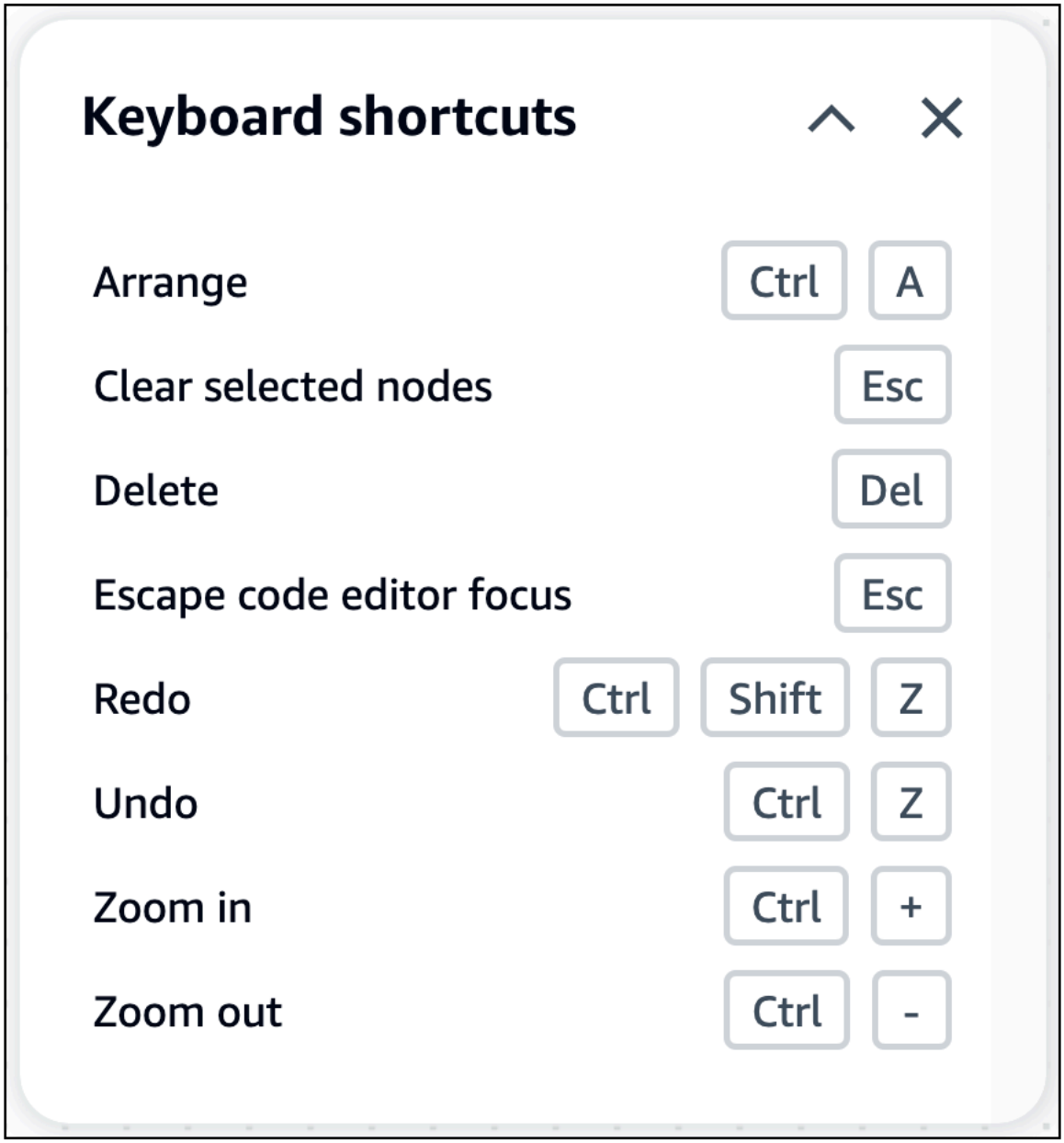
Zoom in and out

Use the zoom controls to zoom in and out of your canvas. You can also use a multi-touch trackpad, with the general pinch-to-zoom gestures.



Keyboard shortcuts

For a list of keyboard shorcuts, select **Keyboard shortcuts** from the Application Composer menu.



Designing with cards in Application Composer

Use cards in AWS Application Composer to design your application architecture.

This topic covers using Application Composer from the AWS Management Console or the AWS Toolkit for Visual Studio Code extension.

For an introduction to Application Composer, see What is AWS Application Composer?.

Topics

- What are cards?
- What are resource cards?
- What are component cards?
- Using standard IaC resource cards
- Using standard component cards
- Deleting standard resource and component cards
- Using enhanced component cards
- Learn more

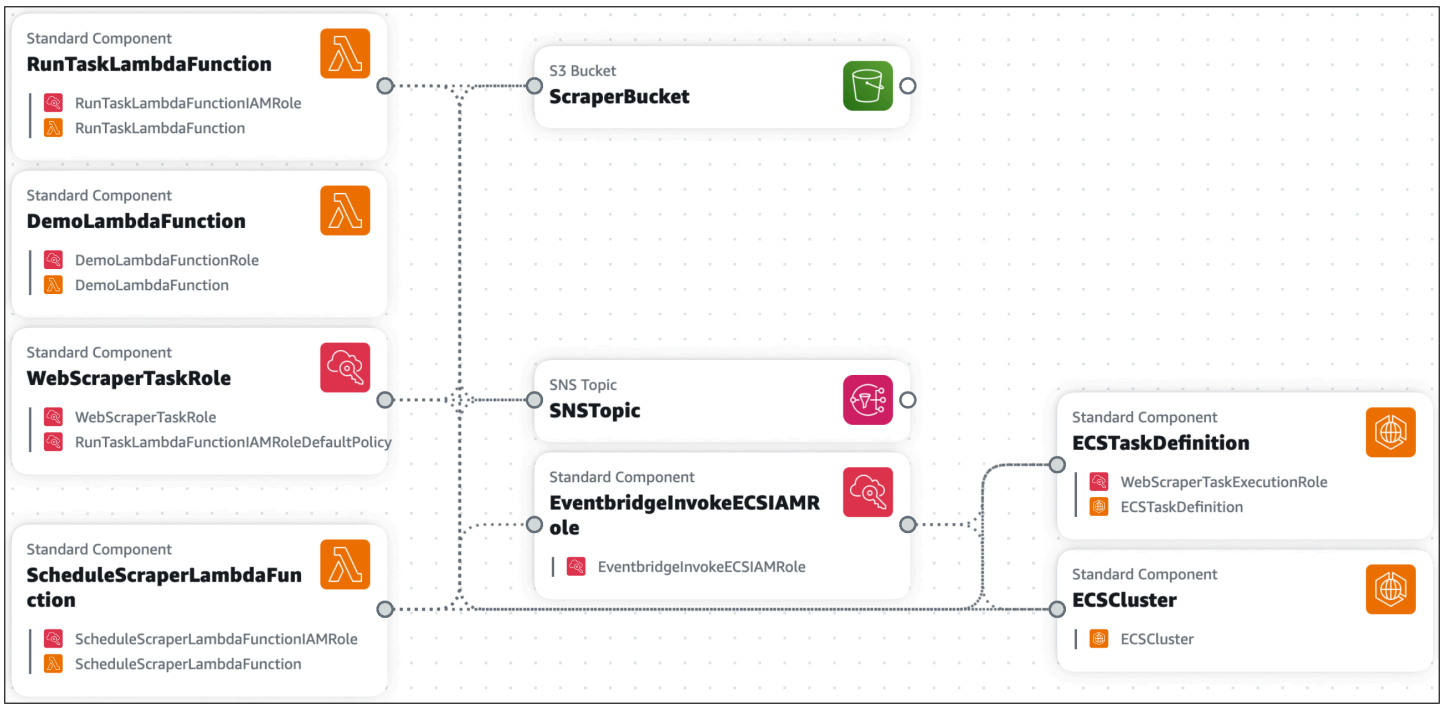
What are cards?

In Application Composer, a card can represent the following:

- **Resource** – A single AWS CloudFormation resource.
- **Component** – A collection of one or more AWS CloudFormation resources.

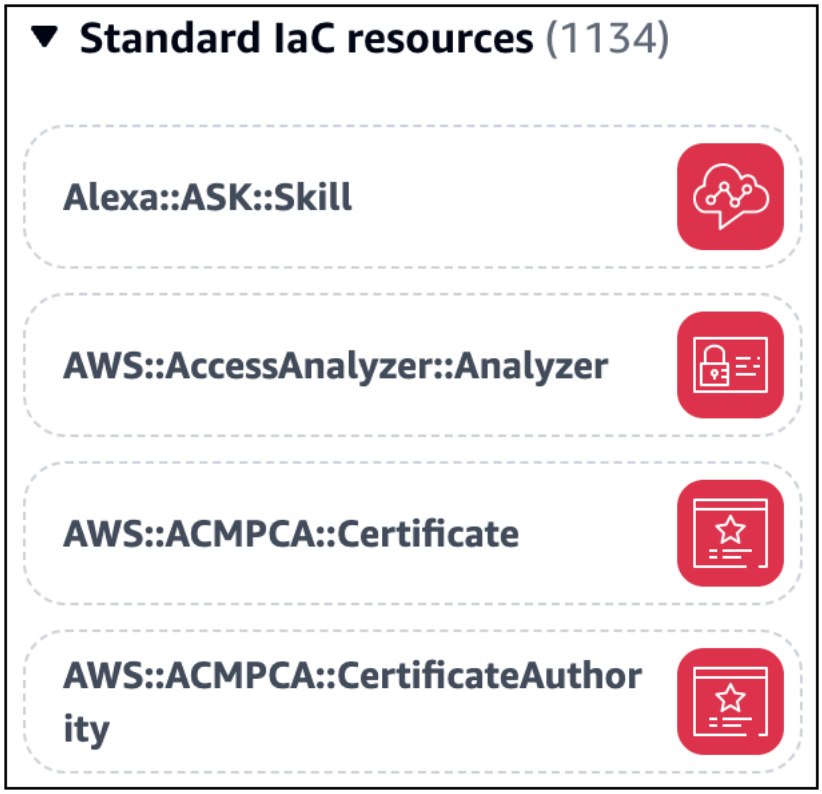
Some types of cards are available from the **Resources** palette for you to drag onto the canvas to use. Cards will also appear on the canvas when you import an existing AWS CloudFormation or AWS Serverless Application Model (AWS SAM) template into Application Composer.

The following is an example of an imported application that contains various card types:



What are resource cards?

A resource card represents a single AWS CloudFormation resource. Resource cards are called **standard infrastructure as code (IaC) resources** in Application Composer. They are available from the **Resources** palette under the **Standard IaC resources** section.



Each standard IaC resource card can be identified by its AWS CloudFormation resource type. The following is an example of a standard IaC resource card that represents an `AWS::ECS::Cluster` AWS CloudFormation resource type:

What are component cards?

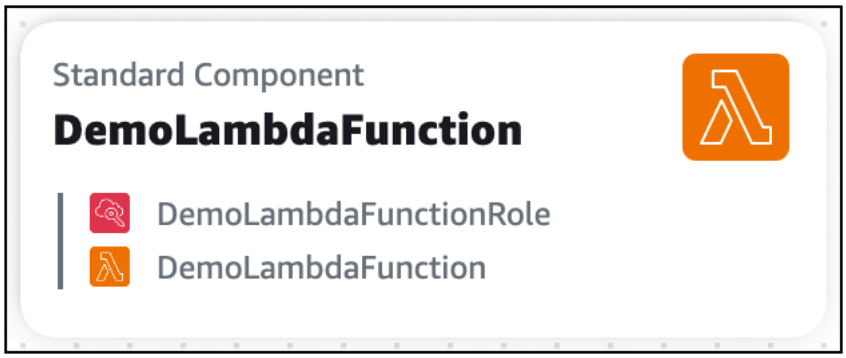
A component card represents a collection of one or more AWS CloudFormation resources. There are two categories of component cards:

1. Standard component card
2. Enhanced component card

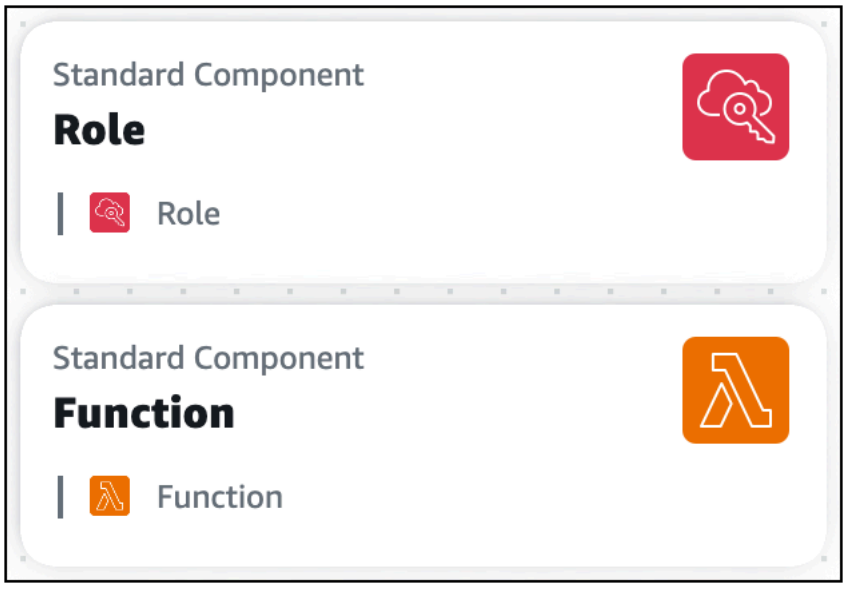
Standard component cards

A standard component card represents a collection of one or more standard IaC resource cards. Standard component cards are not available from the **Resources** palette. When you drag a standard IaC resource card onto the canvas, Application Composer may convert it to a standard component card. Also, when you import an existing application template, Application Composer may visualize AWS CloudFormation resources as standard component cards on the canvas.

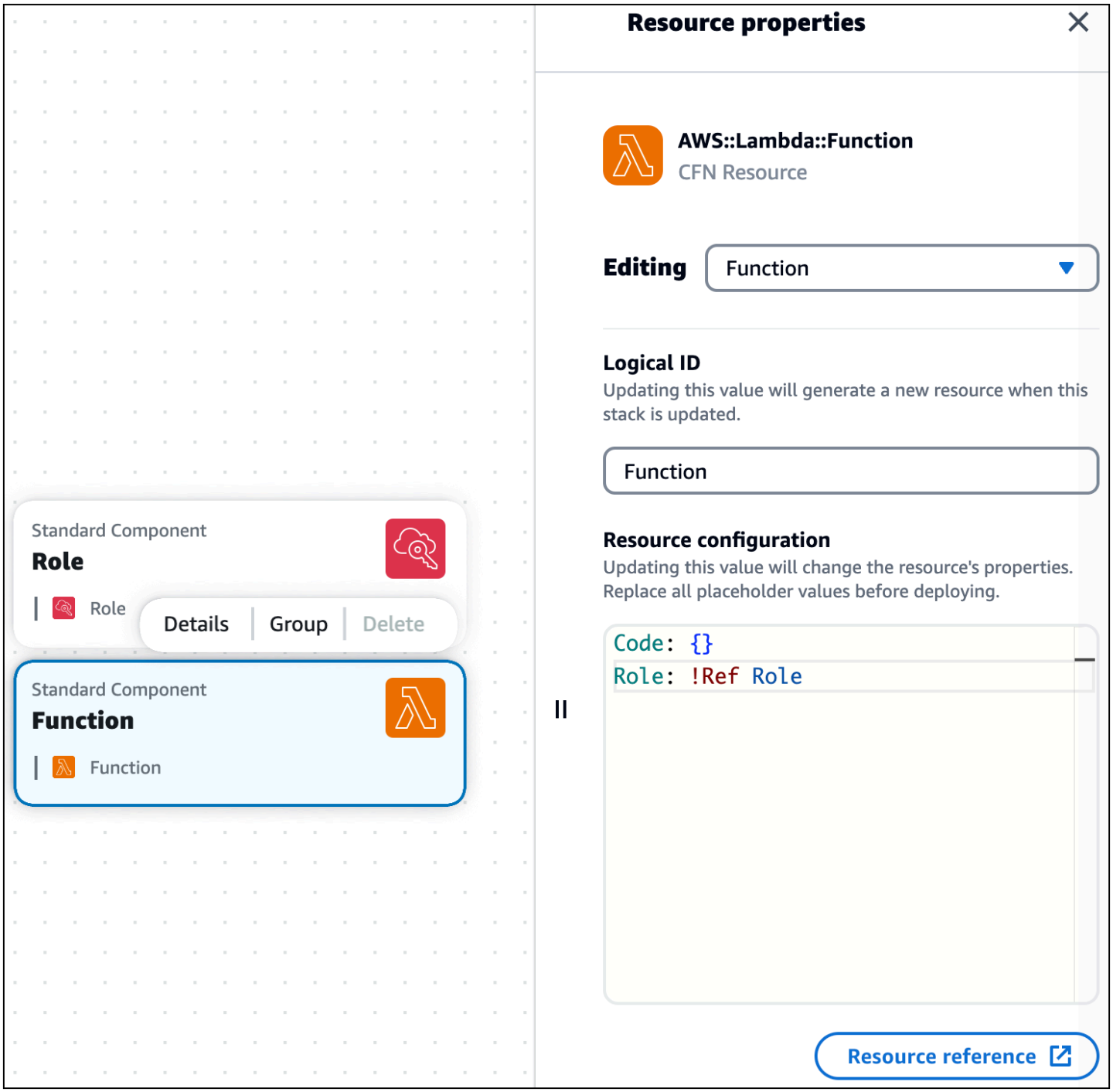
Standard component cards are labeled on the canvas. Each standard component card visualizes the AWS CloudFormation resources that it contains. The following is an example of a standard component card that includes two standard IaC resources:



As you configure the properties of your standard component cards, Application Composer may combine related cards together. For example, here are two standard component cards:



In the **Resource properties** panel of the standard component card representing an `AWS::Lambda::Function` resource, we reference the AWS Identity and Access Management (IAM) role by its logical ID:



After saving our template, the two standard component cards combine into a single standard component card.

Enhanced component cards

An enhanced component card contains a collection of AWS CloudFormation resources that are commonly used together. They are available from the **Resources** palette, under the **Enhanced components** section.

The following is an example of an **S3 Bucket** enhanced component:

When you drag an **S3 Bucket** component card onto the canvas and view your template, you will see the following two AWS CloudFormation resources added to your template:

- `AWS::S3::Bucket`
- `AWS::S3::BucketPolicy`

The **S3 Bucket** enhanced component card represents two AWS CloudFormation resources that are both required for an Amazon Simple Storage Service (Amazon S3) bucket to interact with other services in your application.

Enhanced component cards are created and managed by Application Composer. Each card contains AWS CloudFormation resources that are commonly used together when building applications on AWS. Their infrastructure code is created by Application Composer following AWS best practices. Enhanced components are a great place to start designing your applications with.

Using standard IaC resource cards

All AWS CloudFormation resources are available to use as standard IaC resource cards from the **Resources** palette. When you drag a standard IaC resource onto the canvas, Application Composer creates a starting template for your resource in your application. The following is an example starting template of an `Alexa::ASK::Skill` standard IaC resource:

```
Resources:
  Skill:
    Type: Alexa::ASK::Skill
    Properties:
      AuthenticationConfiguration:
        RefreshToken: <String>
        ClientSecret: <String>
        ClientId: <String>
      VendorId: <String>
      SkillPackage:
        S3Bucket: <String>
        S3Key: <String>
```

A standard IaC resource card starting template consists of the following:

- The AWS CloudFormation resource type.
- Required or commonly used properties.
- The required type of the value to provide for each property.

To modify the properties of a standard IaC resource, use the following options:

1. Directly modify the resource's properties in the **Resource properties** panel of the resource. To do this, select the card on the canvas and choose **Details** to bring up the **Resource properties** panel.
2. When your template is connected with Application Composer, you can directly modify the resource's properties from your template. Application Composer will automatically update the canvas to reflect your changes.

   You can connect a template with Application Composer in the following ways:

   - For Application Composer from the AWS Management Console, activate **local sync** to work with a template from your local machine.

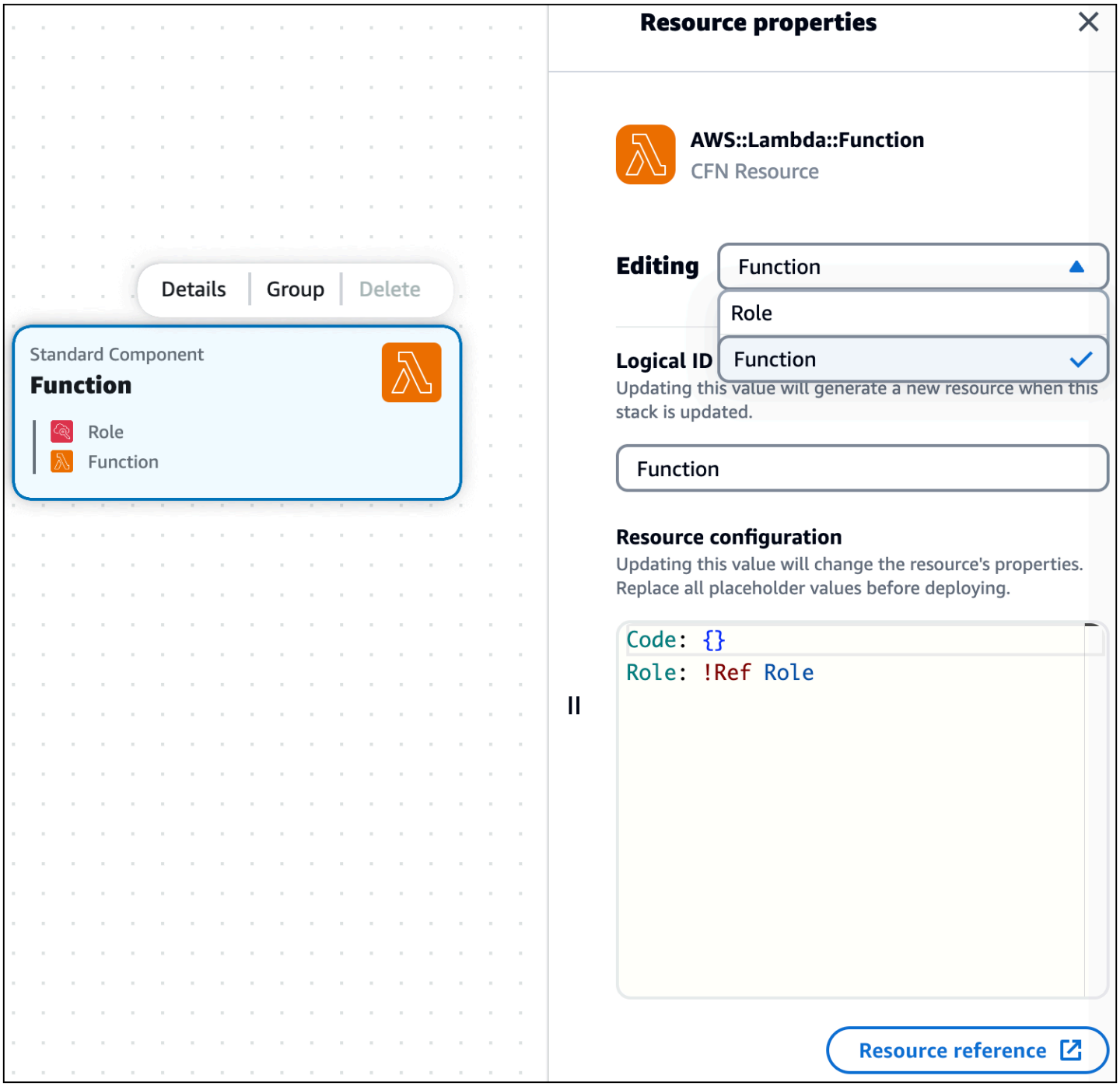
- For Application Composer from the AWS Management Console, select the **Template** view to directly modify your template.
- For Application Composer from the AWS Toolkit for Visual Studio Code, you can work with any template by opening Application Composer from the template.

Using Amazon CodeWhisperer to generate infrastructure code

You can use CodeWhisperer to generate infrastructure code suggestions for standard resource cards. To learn more, see Using AWS Application Composer with Amazon CodeWhisperer.

Using standard component cards

You can modify the infrastructure code for each resource in a standard component card through the **Resource properties** panel.



To modify a standard component card

1. Open the **Resource properties** panel of the standard IaC component card.
2. In the **Editing** field, select the standard IaC resource to edit from the dropdown list.
3. Modify your infrastructure code and **Save**.

You can also modify your infrastructure code through the **Template** view.

Using Amazon CodeWhisperer to generate infrastructure code

You can use CodeWhisperer to generate infrastructure code suggestions for standard component cards. To learn more, see Using AWS Application Composer with Amazon CodeWhisperer.

Deleting standard resource and component cards

To delete standard resource and component cards, you must manually remove the infrastructure code for each AWS CloudFormation resource from your template. The following is a simple way to accomplish this:

1. Take note of the logical ID for the resource to delete.
2. On your template, locate the resource by its logical ID from the `Resources` or `Outputs` section.
3. Delete the resource from your template. This includes the resource logical ID and its nested values, such as `Type` and `Properties`.
4. Check the **Canvas** view to verify that the resource has been removed from your canvas.

Using enhanced component cards

To configure enhanced components, Application Composer provides a form in the **Resource properties** panel. Forms in Application Composer are curated uniquely to guide you through configuring each enhanced component. As you fill out the form, Application Composer modifies your infrastructure code.

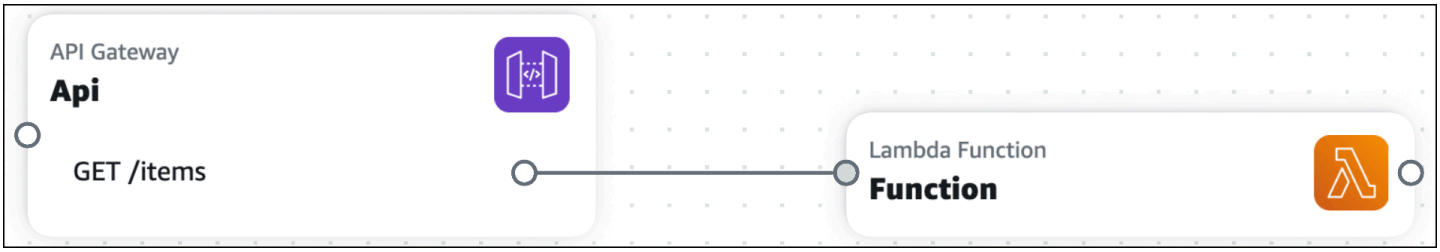
Learn more

For reference information on Application Composer cards, see Application Composer card reference.

Connect cards together using the Application Composer visual canvas

In AWS Application Composer, a connection between two cards is visually displayed by a line. These lines represent event-driven relationships within your application.

This topic covers using Application Composer from the AWS Management Console or the AWS Toolkit for Visual Studio Code extension.



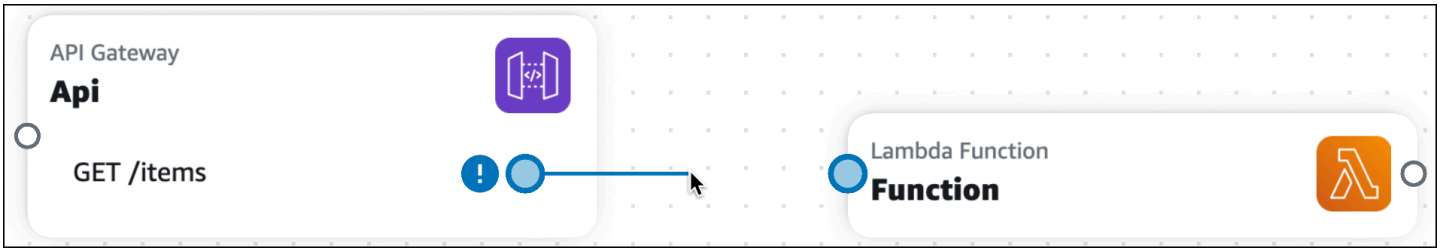
How you connect cards together will vary, depending on the card type. To learn more about card types, see Designing with cards.

Connecting enhanced component cards

On enhanced component cards, ports visually identify where connections can be made.

- A port on the right side of a card indicates an opportunity for the card to invoke another card.
- A port on the left side of a card indicates an opportunity for the card to be invoked by another card.

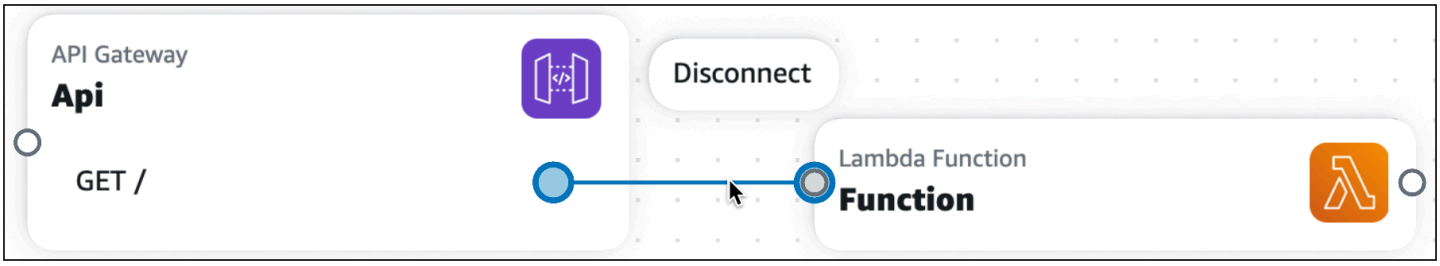
Connect cards together by clicking on a the right port of one card and dragging it onto a left port on another card.



When you connect enhanced component cards together, Application Composer automatically creates the infrastructure code in your template to provision the event-driven relationship between your resources.

Disconnecting cards

To disconnect enhanced component cards, select the line and choose **Disconnect**.

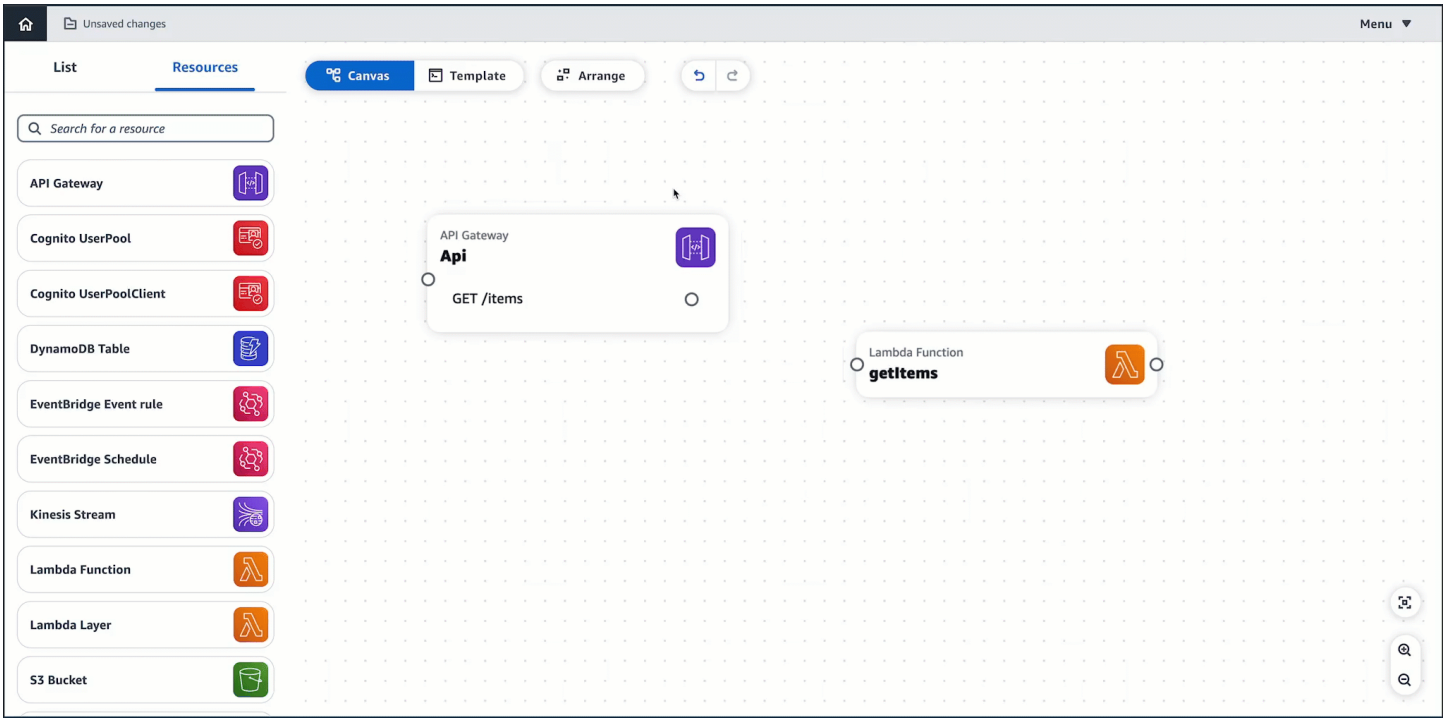


Application Composer will automatically modify your template to remove the event-driven relationship from your application.

Supported connections

When you create a connection, a message will display, letting you know if the connection was successfully made.

- When successful, click on the message to view the **Change inspector**. Here, you can see what Application Composer modified to provision your connection.
- When unsuccessful, a message will display. You can select the **Template view** and manually update your infrastructure code to provision the connection.



What Application Composer provisions

Connections between two cards, visually indicated by a line, provision the following when necessary:

1. AWS Identity and Access Management (IAM) policies
2. Environment variables
3. Events

IAM policies

When a resource needs permission to invoke another resource, Application Composer provisions resource-based policies using AWS Serverless Application Model (AWS SAM) policy templates.

- To learn more about IAM permissions and policies, see Overview of access management: Permissions and policies in the *IAM User Guide*.
- To learn more about AWS SAM policy templates, see AWS SAM policy templates in the *AWS Serverless Application Model Developer Guide*.

Environment variables

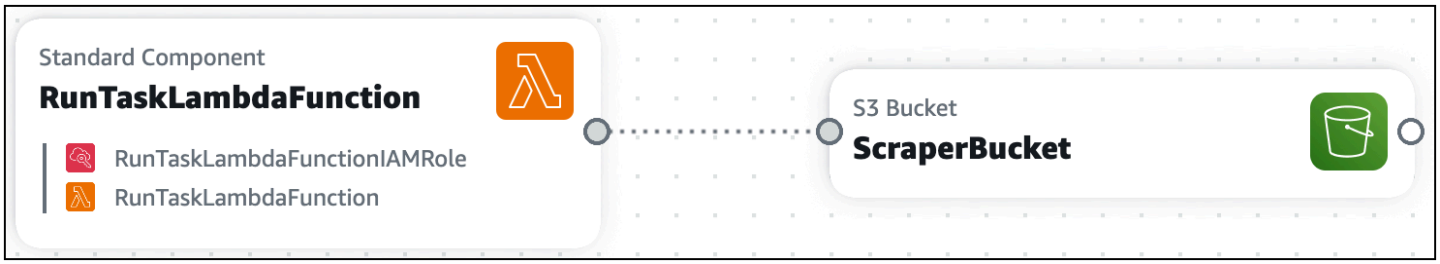
Environment variables are temporary values that can be changed to affect the behavior of your resources. When necessary, Application Composer defines the infrastructure code to utilize environment variables between resources.

Events

Resources can invoke another resource through different types of events. When necessary, Application Composer defines the infrastructure code necessary for resources to interact through event types.

Connecting standard component cards and standard resource cards

Standard component cards and standard resource cards do not include ports to create connections with other resources. When you specify event-driven relationships in the template of your application, Application Composer will automatically detect these connections and visualize them with a dotted line between your cards. The following is an example of a connection between a standard component card and an enhanced component card:



Examples

Invoke an AWS Lambda function when an item is placed in an Amazon Simple Storage Service (Amazon S3) bucket

In this example, an **Amazon S3 bucket** card is connected to a **Lambda function** card. When an item is placed in the Amazon S3 bucket, the function is invoked. The function can then be used to process the item or trigger other events in your application.

This interaction requires that an event be defined for the function. Here is what Application Composer provisions:

```
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyBucket:
    Type: AWS::S3::Bucket
    ...
  MyBucketBucketPolicy:
    Type: AWS::S3::BucketPolicy
    ...
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      ...
      Events:
        MyBucket:
          Type: S3
          Properties:
            Bucket: !Ref MyBucket
```

```
        Events:
          - s3:ObjectCreated:* # Event that triggers invocation of function
          - s3:ObjectRemoved:* # Event that triggers invocation of function
```

Invoke an Amazon S3 bucket from a Lambda function

In this example, a **Lambda function** card invokes an **Amazon S3 bucket** card. The Lambda function can be used to perform CRUD operations on items in the Amazon S3 bucket.

This interaction requires the following, which is provisioned by Application Composer:

- IAM policies that allow the Lambda function to interact with the Amazon S3 bucket.
- Environment variables that influence the behavior of the Lambda function.

```
Transform: AWS::Serverless-2016-10-31
...
Resources:
  MyBucket:
    Type: AWS::S3::Bucket
    ...
  MyBucketBucketPolicy:
    Type: AWS::S3::BucketPolicy
    ...
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      ...
      Environment:
        Variables:
          BUCKET_NAME: !Ref MyBucket
          BUCKET_ARN: !GetAtt MyBucket.Arn
      Policies:
        - S3CrudPolicy:
          BucketName: !Ref MyBucket
```

View code updates with the Change Inspector

As you design in AWS Application Composer, your infrastructure code is automatically created. Use the **Change Inspector** to view your template code updates and learn what Application Composer is creating for you.

This topic covers using Application Composer from the AWS Management Console or the AWS Toolkit for Visual Studio Code extension.

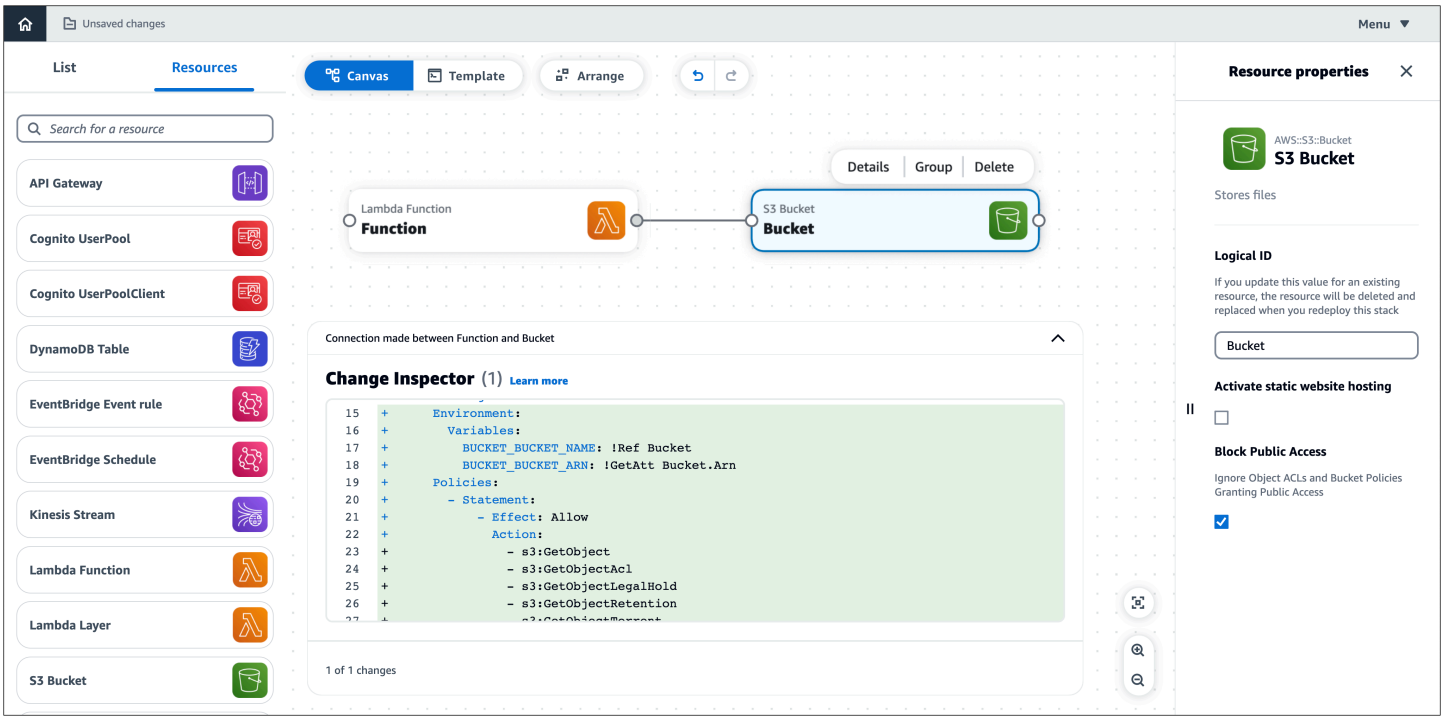
Topics

- What is the Change Inspector?
- Using the Change Inspector
- Benefits of the Change Inspector
- Learn more

What is the Change Inspector?

The **Change Inspector** is a visual tool within Application Composer that shows you recent code updates.

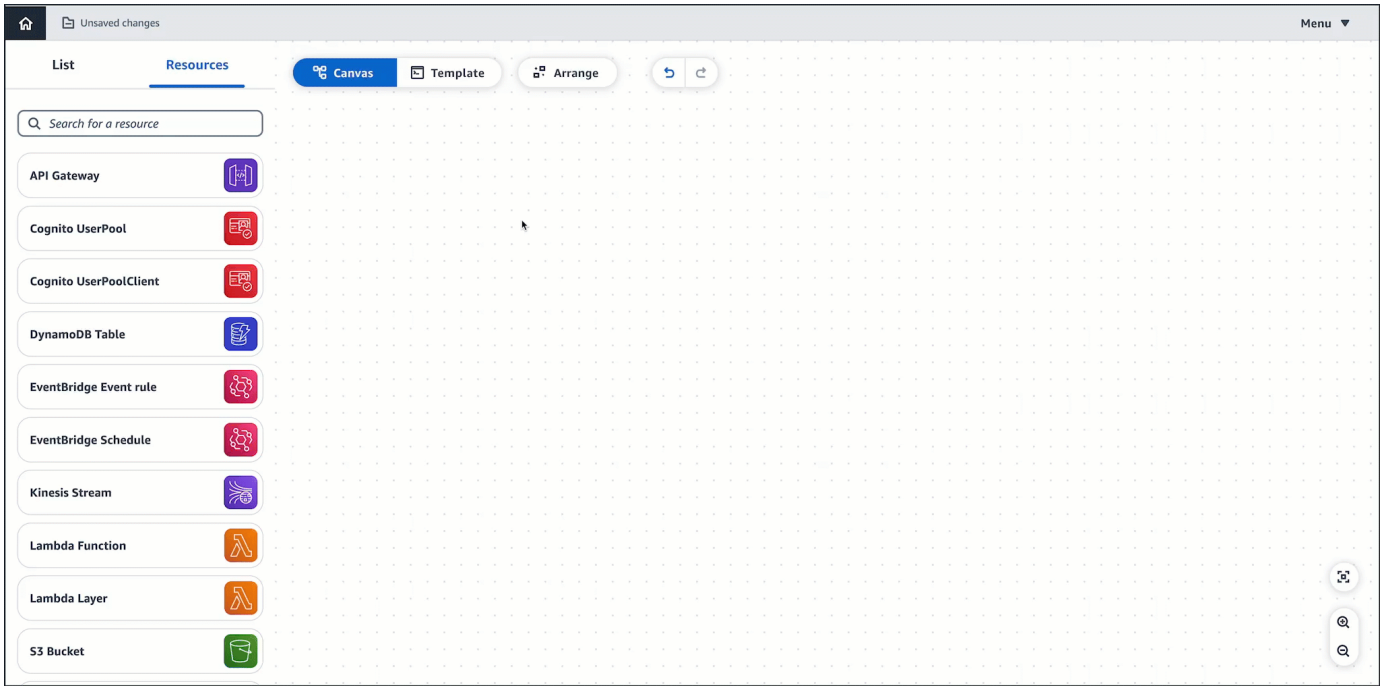
- As you design your application, messages display at the bottom of the visual canvas. These messages provide commentary on the actions you are performing.
- When supported, you can expand a message to view the **Change Inspector**.
- The **Change Inspector** displays code changes from your most recent interaction.



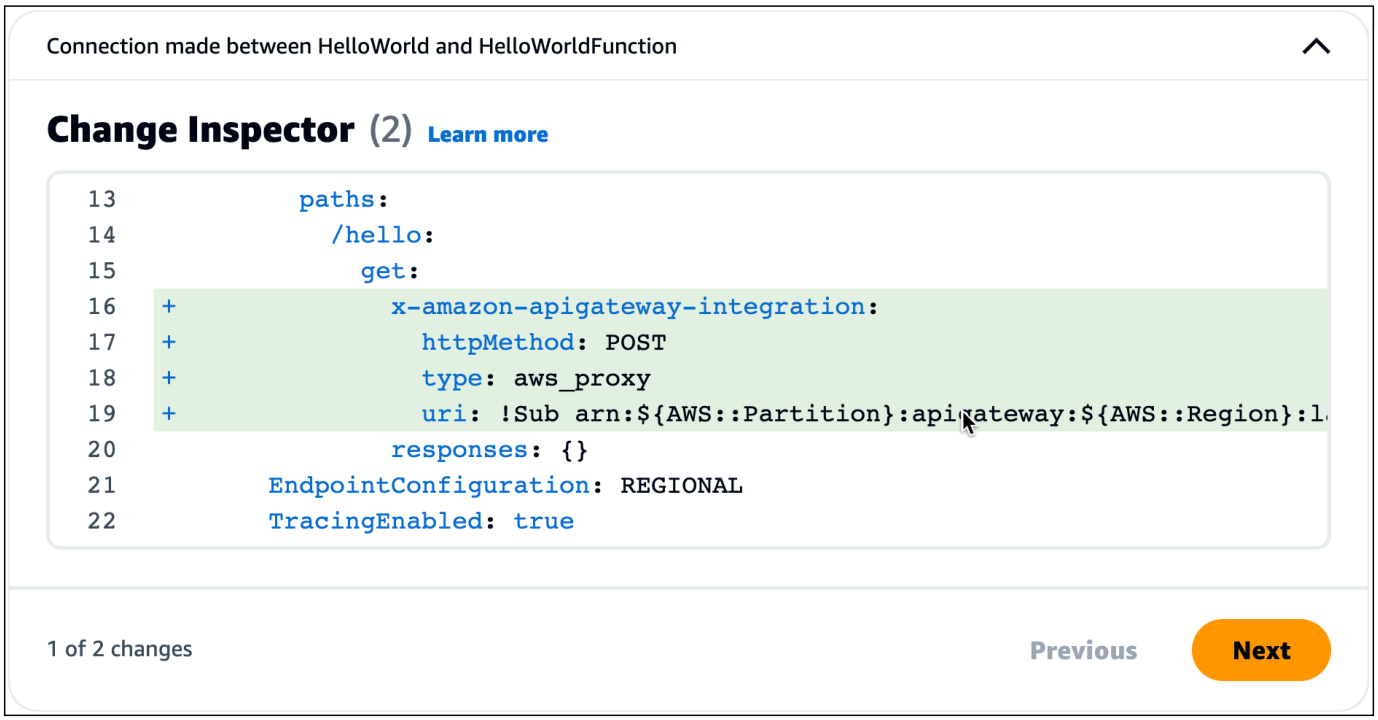
Using the Change Inspector

To use the Change Inspector

1. Expand a message to bring up the **Change Inspector**.



2. View the code that has been automatically composed for you.

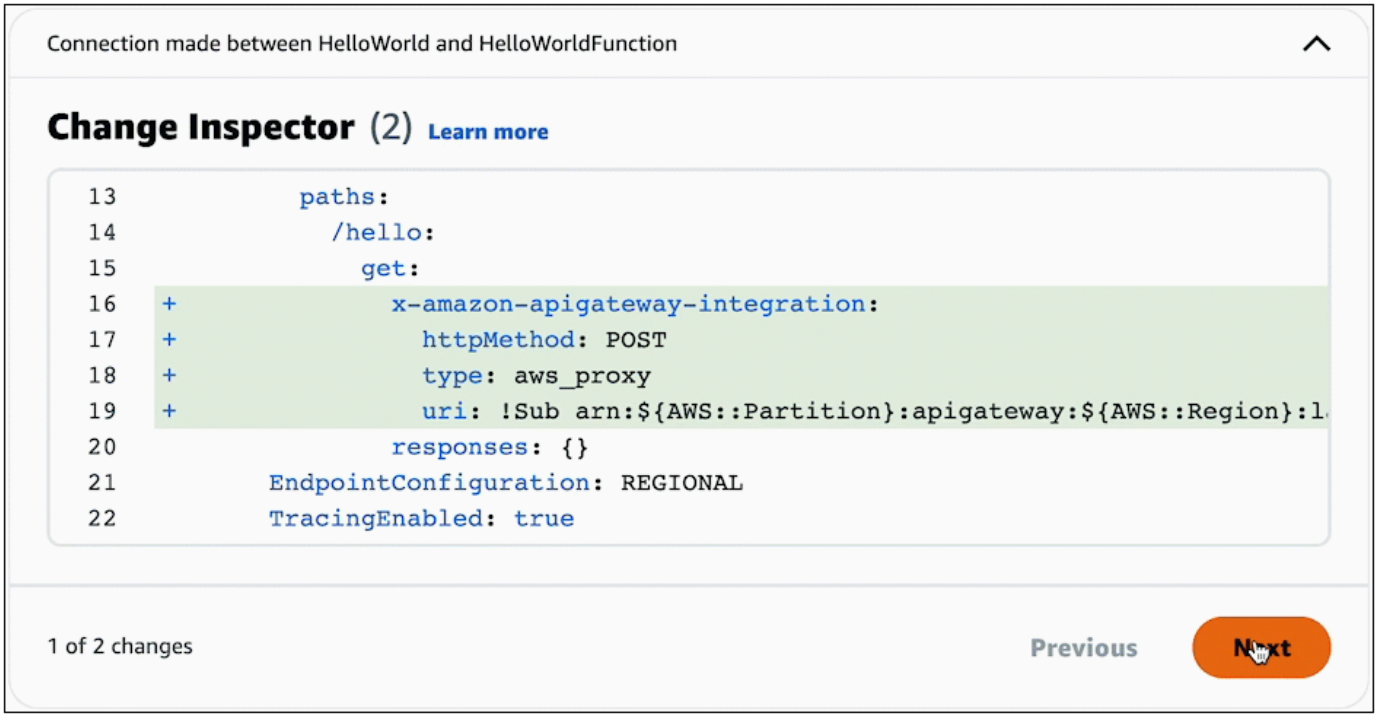


a. Code highlighted **green** indicate newly added code.

b. Code highlighted **red** indicate newly removed code.

c. **Line numbers** indicate the location within your template.

3. When multiple sections of your template have been updated, the **Change Inspector** organizes them. Select the **Previous** and **Next** buttons to view all changes.



Note

For Application Composer from the console, you can view code changes in the context of your entire template, by using the **Template View**. You can also sync Application Composer with a local IDE and view your entire template on your local machine. To learn more, see Using Application Composer with your local IDE.

Benefits of the Change Inspector

The **Change Inspector** is a great way to view the template code that Application Composer creates for you. It is also a great way to learn how to write infrastructure code. As you design applications in Application Composer, view code updates in the **Change Inspector** to learn about the code needed to provision your design.

Learn more

For more information about the code that Application Composer creates, see the following:

- Connect cards together using the Application Composer visual canvas.

Working with templates that reference external files

You can use external files with your AWS Serverless Application Model (AWS SAM) templates to reuse repeated code and organize your projects. For example, you may have multiple Amazon API Gateway REST API resources that are described by an OpenAPI specification. Instead of replicating the OpenAPI specification code in your template, you can create one external file and reference it for each of your resources.

AWS Application Composer supports the following external file use cases:

- API Gateway REST API resources defined by external OpenAPI specification files.
- AWS Step Functions state machine resources defined by external state machine definition files.

To learn more about configuring external files for supported resources, see the following:

- `DefinitionBody` for `AWS::Serverless::Api`.
- `DefinitionUri` for `AWS::Serverless::StateMachine`.

> **Note**
>
> To reference external files with Application Composer from the AWS Management Console, you must use Application Composer in **local sync** mode. For more information, see Local sync mode.

Topics

- Create an external file reference
- Load a project that contains an external file reference
- Best practices
- Examples

Create an external file reference

You can create an external file reference from the **resource properties** panel of supported resources.

To create an external file reference

1. From an **API Gateway** or **Step Functions** enhanced component card, select **Details** to bring up the **resource properties** panel.
2. Locate and select the **Use external file** option.
3. Specify the relative path to the external file. This is the path from your `template.yaml` file to the external file.

   For example, to reference the `api-spec.yaml` external file from the following project's structure, specify `./api-spec.yaml` as your relative path.

   ```
   demo
   ### api-spec.yaml
   ### src
   # ### Function
   # ### index.js
   # ### package.json
   ### template.yaml
   ```

   > **Note**
   >
   > If the external file and its specified path does not exist, Application Composer will create it.

4. **Save** your changes.

Load a project that contains an external file reference

Application Composer from the AWS Management Console

When you load a project, the following occurs:

- If your browser supports the File System Access API, Application Composer will prompt you to connect to the root folder of your project. Application Composer will open your project in **local sync** mode to support your external file.
- If the referenced external file is not supported, you will receive an error message. For more information about error messages, see Troubleshooting.

Application Composer from the Toolkit for VS Code

When you access Application Composer from a template, Application Composer will automatically detect your external file. If the referenced external file is not supported, you will receive an error message. For more information about error messages, see Troubleshooting.

Best practices

For Application Composer from the AWS Management Console, use Application Composer with a local IDE

When you use Application Composer with a local IDE in **local sync** mode, you can use your local IDE to view and modify external files. Content from supported external files that are referenced on your template will automatically update in the Application Composer canvas. To learn more, see Using Application Composer with your local IDE.

Keep external files within your project's parent directory

You can create subdirectories within your project's parent directory to organize your external files. Application Composer can't access external files that are stored in a directory outside of your project's parent directory.

Deploy your application using the AWS SAM CLI

When deploying your application to the AWS Cloud, local external files need to first be uploaded to an accessible location, such as Amazon Simple Storage Service (Amazon S3). You can use the AWS SAM CLI to automatically facilitate this process. To learn more, see Upload local files at deployment in the *AWS Serverless Application Model Developer Guide*.

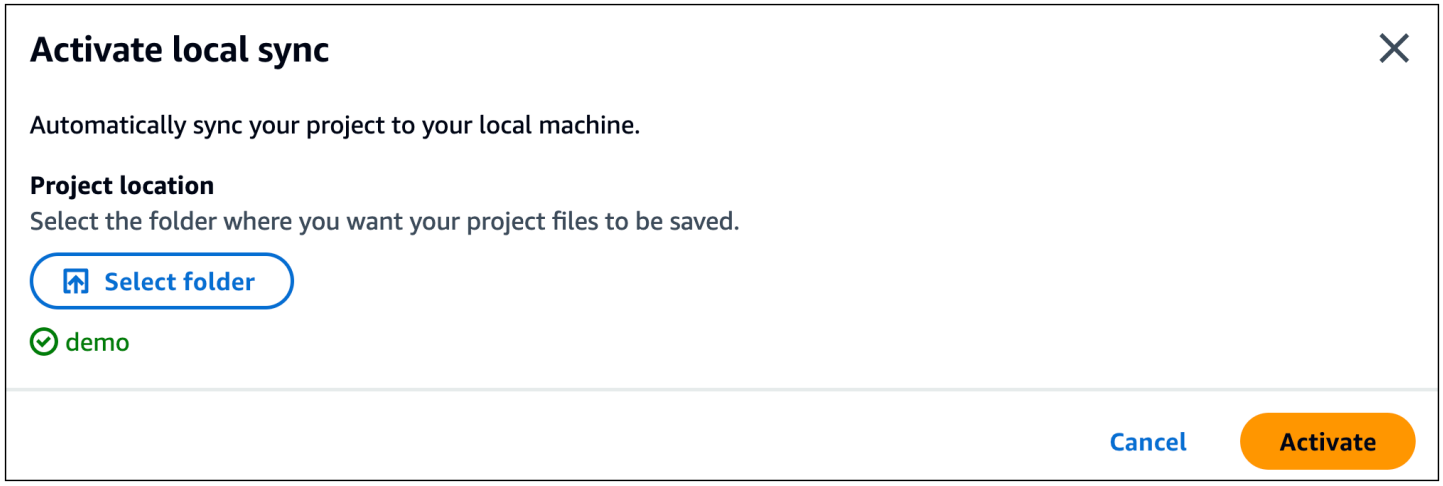
Examples

Reference an OpenAPI specification external file

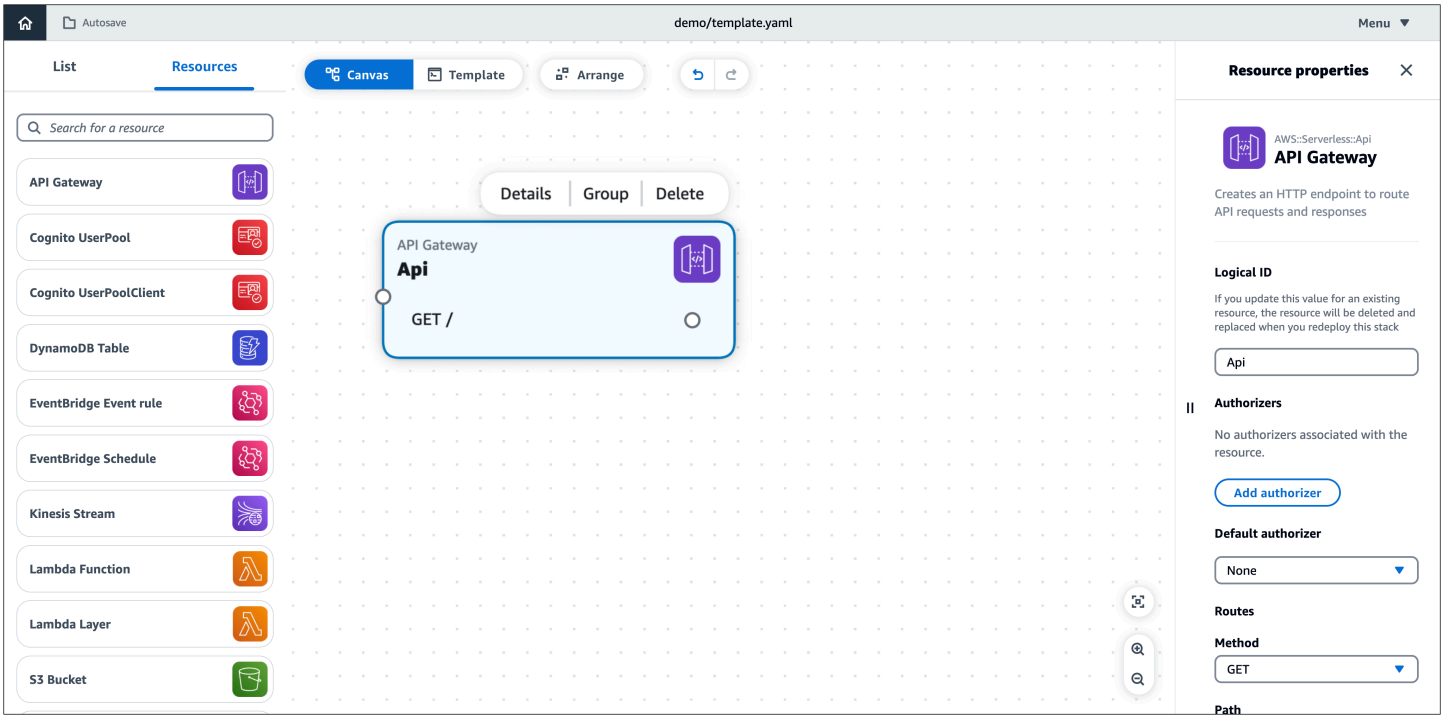
In this example, we use Application Composer from the console to reference an external OpenAPI specification file that defines our API Gateway REST API.

First, we create a new project from the Application Composer **home** page.

Next, we activate **local sync** by selecting **Activate local sync** from the **Menu**. We create a new folder named `demo`, allow the prompt to view files, and select **Activate**. When prompted, we select **Save changes**.

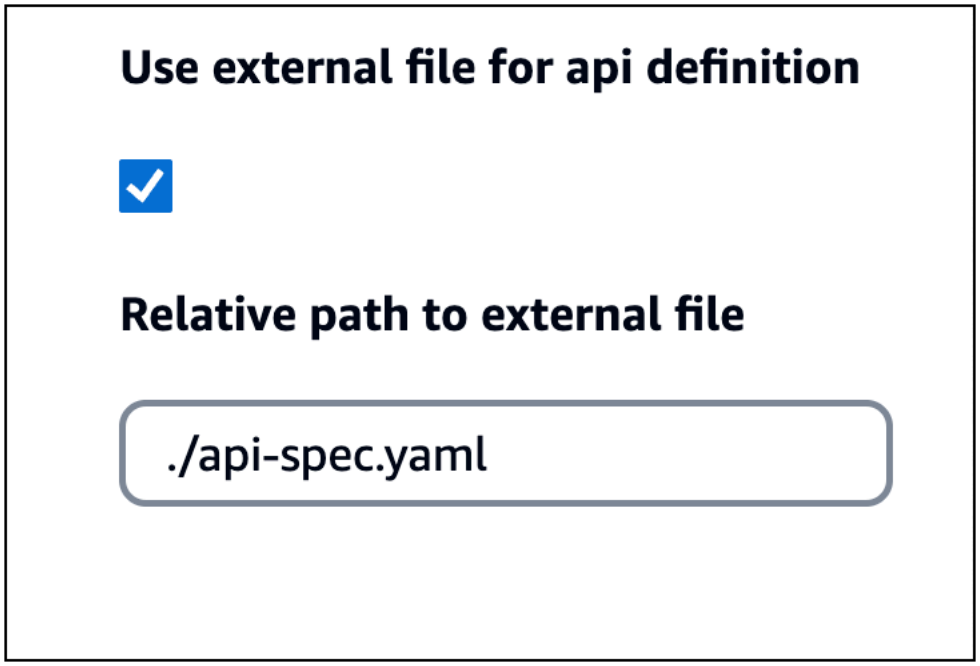


Next, we drag an Amazon API Gateway card onto the canvas. We select **Details** to bring up the **Resource properties** panel.



From the **Resource properties** panel, we configure the following and **save**.

- Select the **Use external file for api definition** option.
- Input `./api-spec.yaml` as the **relative path to external file**



This creates the following directory on our local machine:

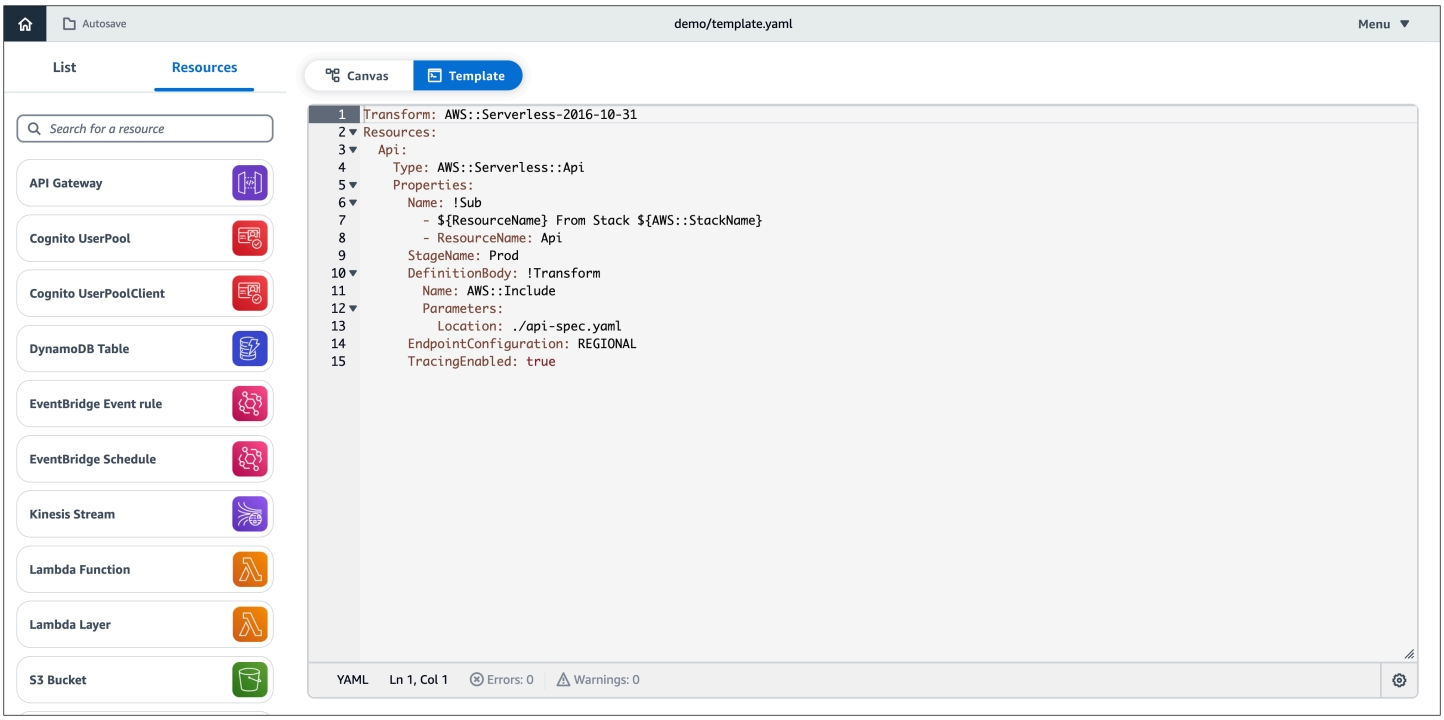
```
demo
### api-spec.yaml
```

Now, we can configure the external file on our local machine. Using our IDE, we open the `api-spec.yaml` located in our project folder. We replace its contents with the following:

```
openapi: '3.0'
info: {}
paths:
  /:
    get:
      responses: {}
    post:
      x-amazon-apigateway-integration:
        credentials:
          Fn::GetAtt:
            - ApiQueuesendmessageRole
            - Arn
        httpMethod: POST
        type: aws
        uri:
          Fn::Sub: arn:${AWS::Partition}:apigateway:${AWS::Region}:sqs:path/
${AWS::AccountId}/${Queue.QueueName}
```

```
        requestParameters:
          integration.request.header.Content-Type: '''application/x-www-form-
urlencoded'''
        requestTemplates:
          application/json: Action=SendMessage&MessageBody={"data":$input.body}
        responses:
          default:
            statusCode: 200
      responses:
        '200':
          description: 200 response
```

In the Application Composer **Template** view, we can see that Application Composer has automatically updated our template to reference the external file.



Create an application using the AWS SAM CLI and load it in Application Composer

In this example, we use the AWS SAM CLI to create an application that references an external file for its state machine definition. We then load our project in Application Composer with our external file properly referenced.

First, we use the AWS SAM CLI **sam init** command to initialize a new application named demo. During the interactive flow, we select the **Multi-step workflow** quick start template.

```
$ sam init
```

```
...

Which template source would you like to use?
        1 - AWS Quick Start Templates
        2 - Custom Template Location
Choice: 1

Choose an AWS Quick Start application template
        1 - Hello World Example
        2 - Multi-step workflow
        3 - Serverless API
        4 - Scheduled task
        ...
Template: 2

Which runtime would you like to use?
        1 - dotnet6
        2 - dotnetcore3.1
        ...
        15 - python3.7
        16 - python3.10
        17 - ruby2.7
Runtime: 16

Based on your selections, the only Package type available is Zip.
We will proceed to selecting the Package type as Zip.

Based on your selections, the only dependency manager available is pip.
We will proceed copying the template using pip.

Would you like to enable X-Ray tracing on the function(s) in your application?  [y/
N]: ENTER

Would you like to enable monitoring using CloudWatch Application Insights?
For more info, please view https://docs.aws.amazon.com/AmazonCloudWatch/latest/
monitoring/cloudwatch-application-insights.html [y/N]: ENTER

Project name [sam-app]: demo

    -----------------------
    Generating application:
    -----------------------
    Name: demo
```

```
    Runtime: python3.10
    Architectures: x86_64
    Dependency Manager: pip
    Application Template: step-functions-sample-app
    Output Directory: .
    Configuration file: demo/samconfig.toml

    Next steps can be found in the README file at demo/README.md

...
```

This application references an external file for the state machine definition.

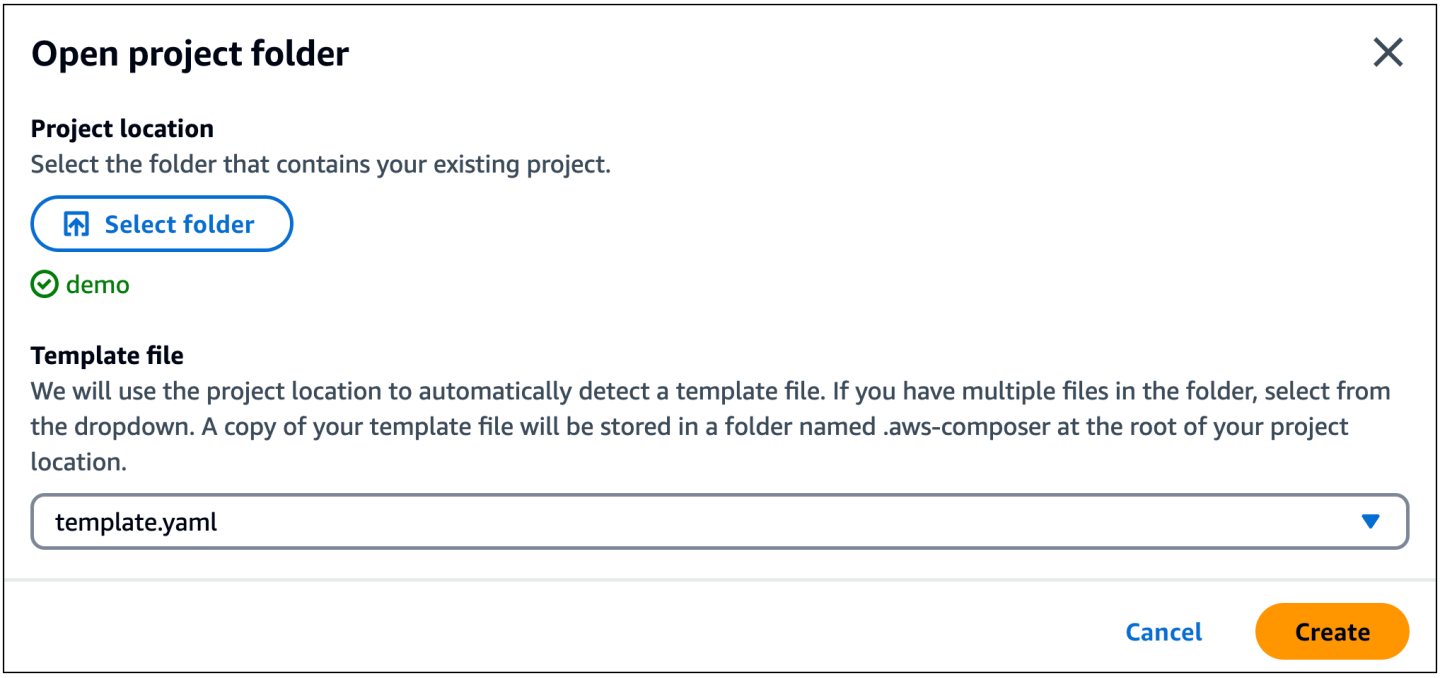
```
...
Resources:
  StockTradingStateMachine:
    Type: AWS::Serverless::StateMachine
    Properties:
      DefinitionUri: statemachine/stock_trader.asl.json
...
```

The external file is located in the `statemachine` subdirectory of our application.

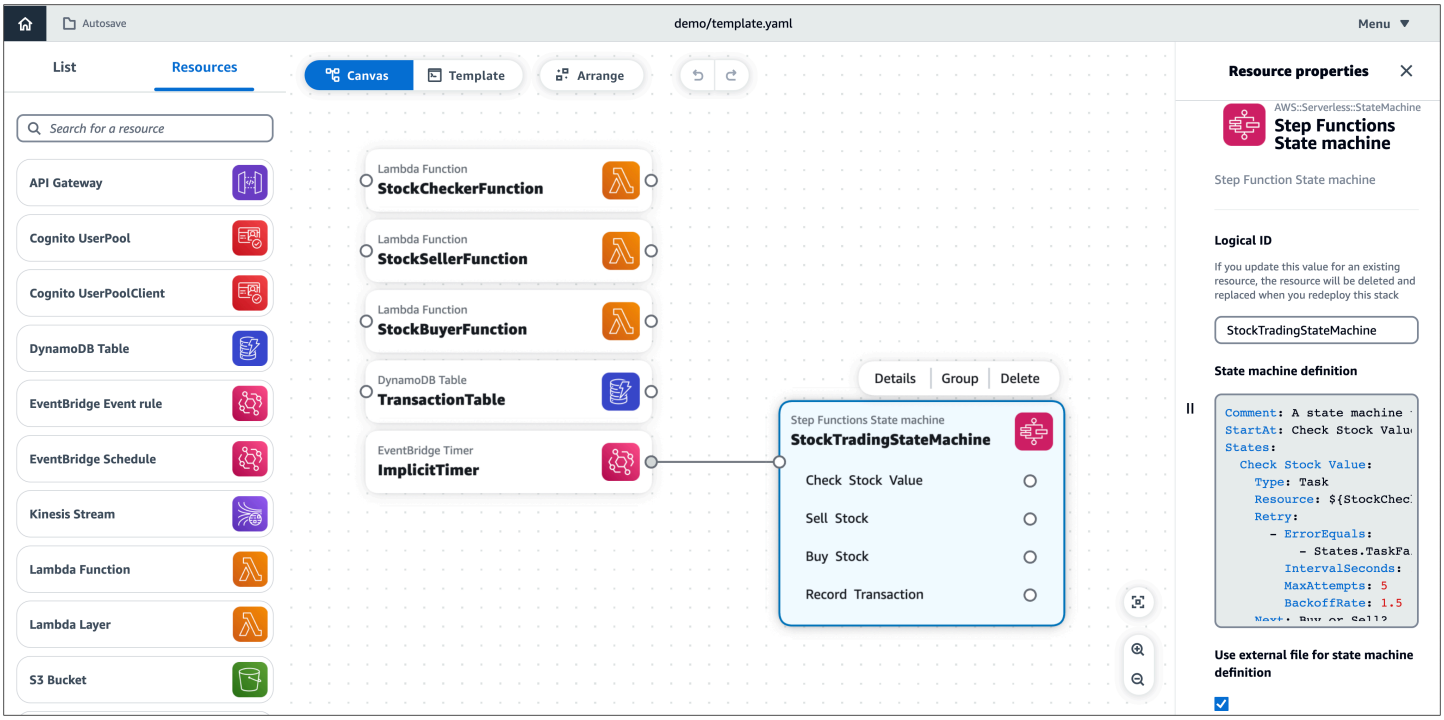
```
demo
### README.md
### __init__.py
### functions
#   ### __init__.py
#   ### stock_buyer
#   ### stock_checker
#   ### stock_seller
### samconfig.toml
### statemachine
#   ### stock_trader.asl.json
### template.yaml
### tests
```

Next, we load our application in Application Composer from the console. From the Application Composer **home** page, we select **Load a CloudFormation template**.

We select our demo project folder and allow the prompt to view files. We select our `template.yaml` file and select **Create**. When prompted, we select **Save changes**.



Application Composer automatically detects the external state machine definition file and loads it. We select our **StockTradingStateMachine** resource and choose **Details** to show the **Resource properties** panel. Here, we can see that Application Composer has automatically connected to our external state machine definition file.



Any changes made to the state machine definition file will be automatically reflected in Application Composer.

Using Application Composer with featured AWS services

You can use AWS Application Composer to design and build modern applications from all AWS services that are supported by AWS CloudFormation through standard IaC resource cards.

Application Composer also features some AWS services by providing integrations that improve the experience of designing with those services. This section provides documentation on using featured services within Application Composer.

This section covers using Application Composer from the AWS Management Console or the AWS Toolkit for Visual Studio Code extension.

Topics

- Using Application Composer with Amazon Relational Database Service (Amazon RDS)
- Using AWS Application Composer with AWS Step Functions
- Using Application Composer with Amazon Virtual Private Cloud (Amazon VPC)

Using Application Composer with Amazon Relational Database Service (Amazon RDS)

AWS Application Composer features an integration with the Amazon Relational Database Service (Amazon RDS) service. Using the **RDS Database (External)** enhanced component card in Application Composer, you can connect your application to an Amazon RDS DB cluster or instance that is defined on another template.

Topics

- What is the RDS Database (External) enhanced component card?
- How do I connect my application to an external Amazon RDS DB cluster or instance?
- Requirements
- Connecting to an external Amazon RDS DB instance or cluster
- How Application Composer creates your connection

What is the RDS Database (External) enhanced component card?

The **RDS Database (External)** enhanced component card is available from the **Resources** palette for you to drag onto the Application Composer canvas. This card represents an Amazon RDS DB cluster or instance that is defined on another template.

You can connect your application to the external Amazon RDS DB cluster or instance through an AWS Lambda function.

How do I connect my application to an external Amazon RDS DB cluster or instance?

The following is an overview of how to connect your application to an external Amazon RDS DB cluster or instance:

1. Create a new project in Application Composer.
2. Drag an **RDS Database (external)** enhanced component card onto the canvas and specify its properties.
3. Drag a **Lambda Function** enhanced component card onto the canvas.
4. Connect the right port of the **Lambda Function** card to the left port of the **RDS Database (external)** card.

You can design your application in Application Composer and connect to your Lambda function. Then, you can connect to your external Amazon RDS DB instance or cluster through the connection from your Lambda function.

Requirements

To use this feature, you must meet the following requirements:

1. Your external Amazon RDS DB cluster or instance must be using AWS Secrets Manager to manage the user password. To learn more, see Password management with Amazon RDS and AWS Secrets Manager in the *Amazon RDS User Guide*.

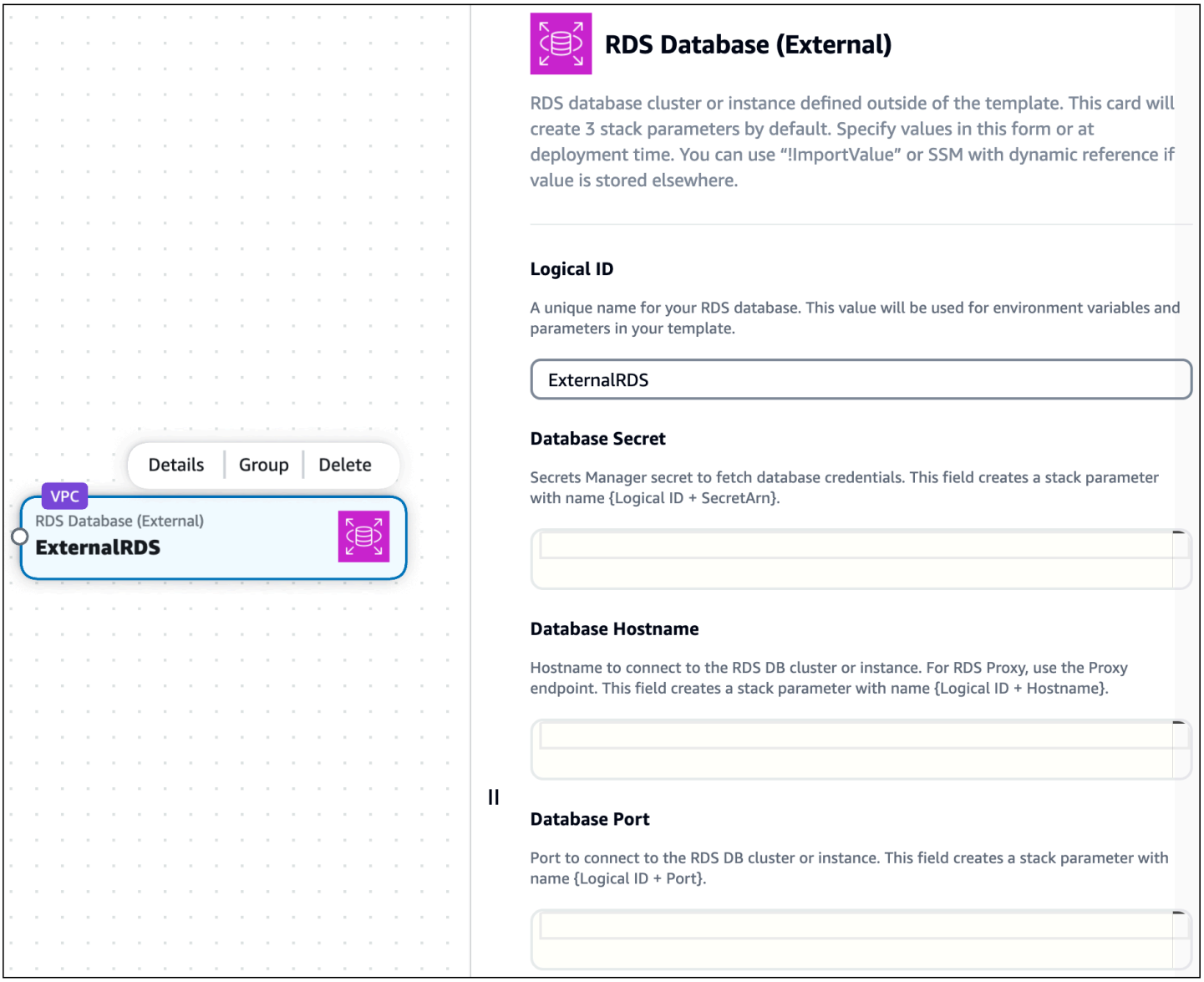
2. Your application in Application Composer must be a new project or must have been originally created in Application Composer.

Connecting to an external Amazon RDS DB instance or cluster

Step 1: Configure the external RDS Database card

From the **Resources** palette, drag an **RDS Database (external)** enhanced component card onto the canvas. This card represents the external Amazon RDS DB cluster or instance.

Select the card and choose **Details** or double-click on the card to bring up the **Resource properties** panel.



You can configure the following here:

- **Logical ID** – A unique name for your external Amazon RDS DB cluster or instance. This ID does not have to match the logical ID value of your external Amazon RDS DB cluster or instance.
- **Database secret** – An identifier for the AWS Secrets Manager secret that is associated with your Amazon RDS DB cluster or instance. This field accepts the following values:
  - **Static value** – A unique identifier of the database secret, such as the secret ARN. The following is an example: `arn:aws:secretsmanager:us-west-2:123456789012:secret:my-path/my-secret-name-1a2b3c`. For more information, see AWS Secrets Manager concepts in the *AWS Secrets Manager User Guide*.
  - **Output value** – When a Secrets Manager secret is deployed to AWS CloudFormation, an output value is created. You can specify the output value here using the `Fn::ImportValue` intrinsic function. For example, `!ImportValue MySecret`.
  - **Value from the SSM Parameter Store** – You can store your secret in the SSM Parameter Store and specify its value using a dynamic reference. For example, `{{resolve:ssm:MySecret}}`. For more information, see SSM parameters in the *AWS CloudFormation User Guide*.
- **Database hostname** – The hostname that can be used to connect to your Amazon RDS DB cluster or instance. This value is specified in the external template that defines your Amazon RDS database. The following values are accepted:
  - **Static value** – A unique identifier of the database hostname, such as the endpoint address. The following is an example: `mystack-mydb-1apw1j4phylrk.cg034hpkmmjt.us-east-2.rds.amazonaws.com`.
  - **Output value** – The output value of a deployed Amazon RDS DB cluster or instance. You can specify the output value using the `Fn::ImportValue` intrinsic function. For example, `!ImportValue myStack-myDatabase-abcd1234`.
  - **Value from the SSM Parameter Store** – You can store the database hostname in the SSM Parameter Store and specify its value using a dynamic reference. For example, `{{resolve:ssm:MyDatabase}}`.
- **Database port** – The port number that can be used to connect to your Amazon RDS DB cluster or instance. This value is specified in the external template that defines your Amazon RDS database. The following values are accepted:
  - **Static value** – The database port. For example, `3306`.
  - **Output value** – The output value of a deployed Amazon RDS DB cluster or instance. For example, `!ImportValue myStack-MyRDSInstancePort`.

- **Value from SSM Parameter Store** – You can store the database hostname in the SSM Parameter Store and specify its value using a dynamic reference. For example, `{{resolve:ssm:MyRDSInstancePort}}`.

> **Note**
>
> Only the logical ID value must be configured here. You can configure the other properties at deployment time if you prefer.

Step 2: Connect a Lambda Function card

From the **Resources** palette, drag a **Lambda Function** enhanced component card onto the canvas.

Connect the left port of the **Lambda Function** card to the right port of the **RDS Database (external)** card. Application Composer will provision your template to facilitate this connection.

How Application Composer creates your connection

Specifying the external Amazon RDS DB cluster or instance

When you drag an **RDS Database (external)** card onto the canvas, Application Composer updates the `Metadata` and `Parameters` sections of your template as needed. The following is an example:

```
Metadata:
  AWS::Composer::ExternalResources:
    ExternalRDS:
      Type: externalRDS
      Settings:
        Port: !Ref ExternalRDSPort
        Hostname: !Ref ExternalRDSHostname
        SecretArn: !Ref ExternalRDSSecretArn
Parameters:
  ExternalRDSPort:
    Type: Number
```

```
  ExternalRDSHostname:
    Type: String
  ExternalRDSSecretArn:
    Type: String
```

Metadata is an AWS CloudFormation template section that is used to store details about your template. Metadata that is specific to Application Composer is stored under the `AWS::Composer::ExternalResources` metadata key. Here, Application Composer stores the values that you specify for your Amazon RDS DB cluster or instance.

The Parameters section of an AWS CloudFormation template is used to store custom values that can be inserted throughout your template at deployment. Depending on the type of values that you provide, Application Composer may store values here for your Amazon RDS DB cluster or instance and specify them throughout your template.

String values in the `Metadata` and `Parameters` section use the logical ID value that you specify on your **RDS Database (external)** card. If you update the logical ID, the string values will change.

Connecting the Lambda function to your database

When you connect a **Lambda Function** card to the **RDS Database (external)** card, Application Composer provisions environment variables and AWS Identity and Access Management (IAM) policies. The following is an example:

```
Resources:
  Function:
    Type: AWS::Serverless::Function
    Properties:
      ...
      Environment:
        Variables:
          EXTERNALRDS_PORT: !Ref ExternalRDSPort
          EXTERNALRDS_HOSTNAME: !Ref ExternalRDSHostname
          EXTERNALRDS_SECRETARN: !Ref ExternalRDSSecretArn
      Policies:
        - AWSSecretsManagerGetSecretValuePolicy:
            SecretArn: !Ref ExternalRDSSecretArn
```

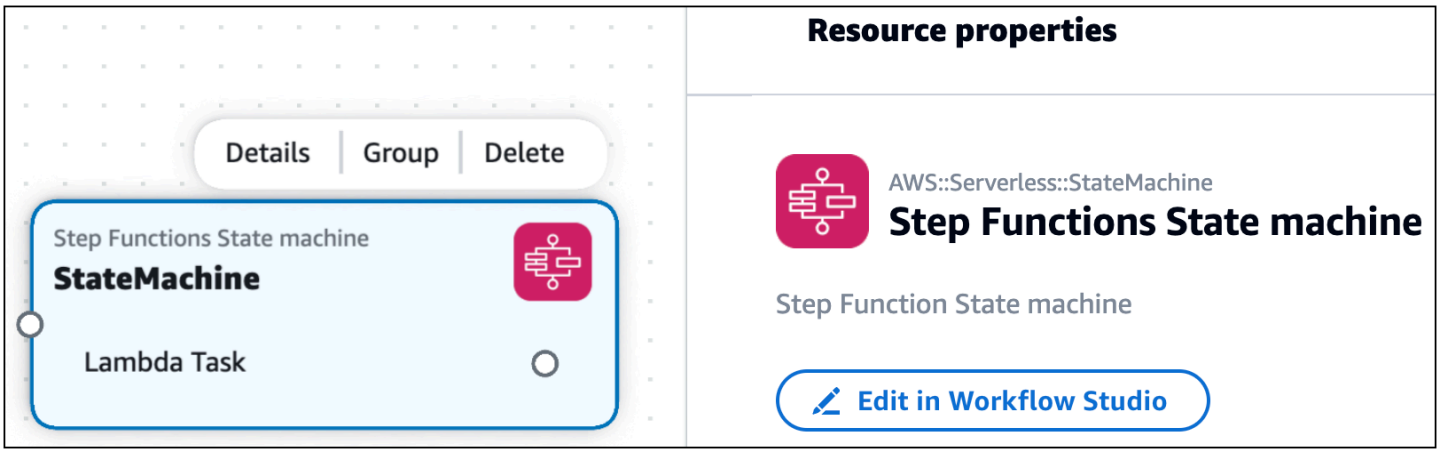
Environment variables are variables that can be used by your function at runtime. To learn more, see Using Lambda environment variables in the *AWS Lambda Developer Guide*.

Policies provision permissions for your function. Here, Application Composer creates a policy to allow read access from your function to Secrets Manager to obtain your password for access to the Amazon RDS DB cluster or instance.

Using AWS Application Composer with AWS Step Functions

AWS Application Composer features an integration with AWS Step Functions Workflow Studio. Use Application Composer to do the following:

- Launch Step Functions Workflow Studio directly within Application Composer.
- Create and manage new workflows or import existing workflows into Application Composer.
- Integrate your workflows with other AWS resources using the Application Composer canvas.



With Step Functions Workflow Studio in Application Composer, you can use the benefits of two powerful visual designers in a single place. As you design your workflow and application, Application Composer creates your infrastructure as code (IaC) to guide you towards deployment.

Topics

- Getting started with Step Functions Workflow Studio in Application Composer
- Using Step Functions Workflow Studio in Application Composer
- Learn more

Getting started with Step Functions Workflow Studio in Application Composer

To get started, you can create new workflows or import existing workflows.

Create a new workflow

To create a new workflow

1. From the **Resources** palette, drag a **Step Functions State machine** enhanced component card onto the canvas.

   When you drag a **Step Functions State machine** card onto the canvas, Application Composer creates the following:

   - An `AWS::Serverless::StateMachine` resource that defines your state machine. By default, Application Composer creates a standard workflow. To create an express workflow, change the `Type` value in your template from `STANDARD` to `EXPRESS`.
   - An `AWS::Logs::LogGroup` resource that defines an Amazon CloudWatch log group for your state machine.

2. Open the card's **Resource properties** panel and select **Edit in Workflow Studio** to open Workflow Studio within Application Composer.

   Step Functions Workflow Studio opens in **Design** mode. To learn more, see Design mode in the *AWS Step Functions Developer Guide*.

   > **Note**
   >
   > You can modify Application Composer to save your state machine definition in an external file. To learn more, see Working with external files.

3. Create your workflow and choose **Save**. To exit Workflow Studio, choose **Return to Application Composer**.

   Application Composer defines your workflow using the `Defintion` property of the `AWS::Serverless::StateMachine` resource.

4. You can modify your workflow by doing any of the following:

   - Open Workflow Studio again and modify your workflow.

- For Application Composer from the console, you can open the **Template** view of your application and modify your template. If using **local sync**, you can modify your workflow in your local IDE. Application Composer will detect your changes and update your workflow in Application Composer.
- For Application Composer from the Toolkit for VS Code, you can directly modify your template. Application Composer will detect your changes and update your workflow in Application Composer.

Import existing workflows

You can import workflows from applications that are defined using AWS Serverless Application Model (AWS SAM) templates. Any state machine defined using the `AWS::Serverless::StateMachine` resource type will visualize as a **Step Functions State machine** enhanced component card that you can use to launch Workflow Studio.

The `AWS::Serverless::StateMachine` resource can define workflows using either of the following properties:

- `Definition` – The workflow is defined within the AWS SAM template as an object.
- `DefinitionUri` – The workflow is defined on an external file using the Amazon States Language. The file's local path is then specified with this property.

Definition property

Application Composer from the console

For workflows defined using the `Definition` property, you can import a single template or the entire project.

- **Template** – For instructions on importing a template, see Import an existing project template. To save changes that you make within Application Composer, you must export your template.
- **Project** – When you import a project, you must activate **local sync**. Changes that you make are automatically saved to your local machine. For instructions on importing a project, see Import an existing project folder.

Application Composer from the Toolkit for VS Code

For workflows defined using the `Definition` property, you can open Application Composer from your template. For instructions, see Accessing Application Composer from the AWS Toolkit for Visual Studio Code.

DefinitionUri property

Application Composer from the console

For workflows defined using the `DefinitionUri` property, you must import the project and activate **local sync**. For instructions on importing a project, see Import an existing project folder.

Application Composer from the Toolkit for VS Code

For workflows defined using the `DefinitionUri` property, you can open Application Composer from your template. For instructions, see Accessing Application Composer from the AWS Toolkit for Visual Studio Code.

Using Step Functions Workflow Studio in Application Composer

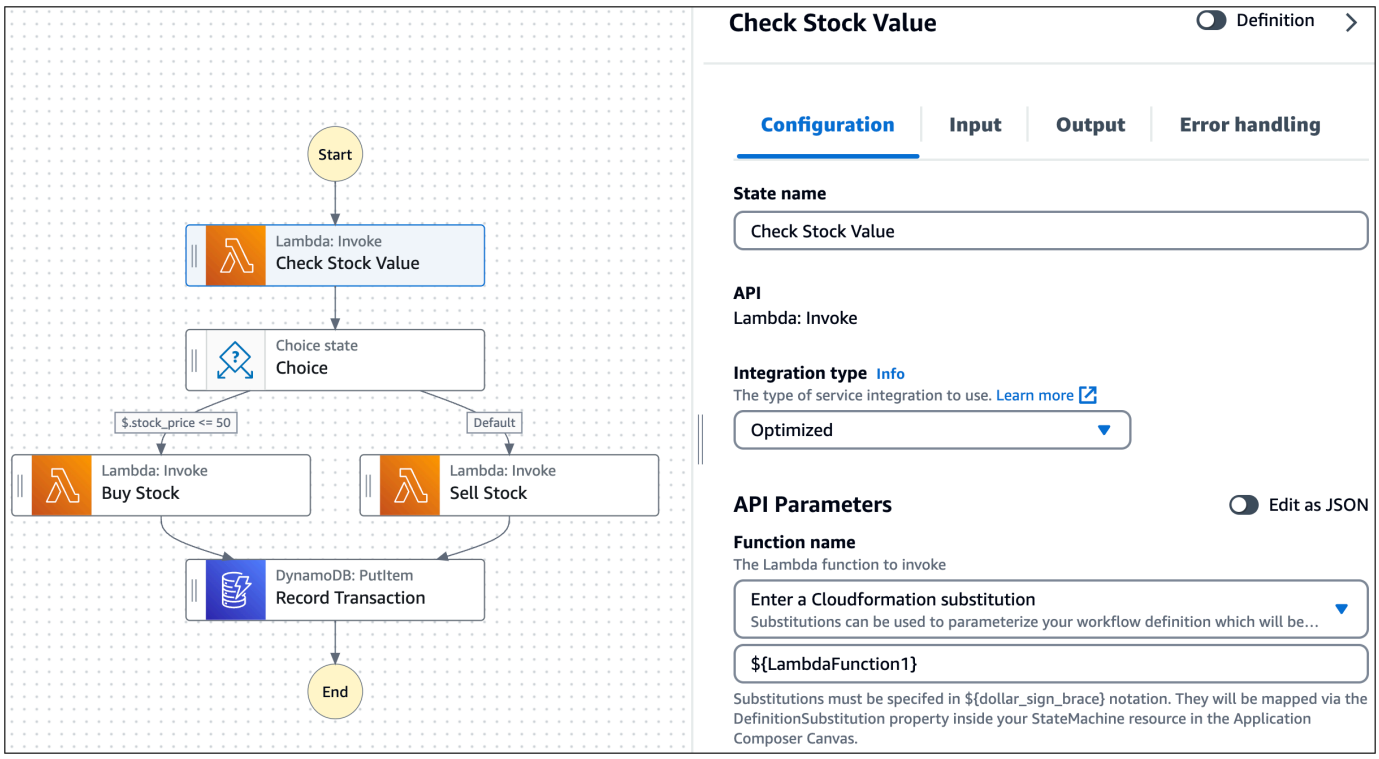
Build workflows

Application Composer uses definition substitutions to map workflow tasks to resources in your application. To learn more about definition substitutions, see `DefinitionSubstitutions` in the *AWS Serverless Application Model Developer Guide*.

When you create tasks in Workflow Studio, specify a definition substitution for each task. You can then connect tasks to resources on the Application Composer canvas.

To specify a definition substitution in Workflow Studio

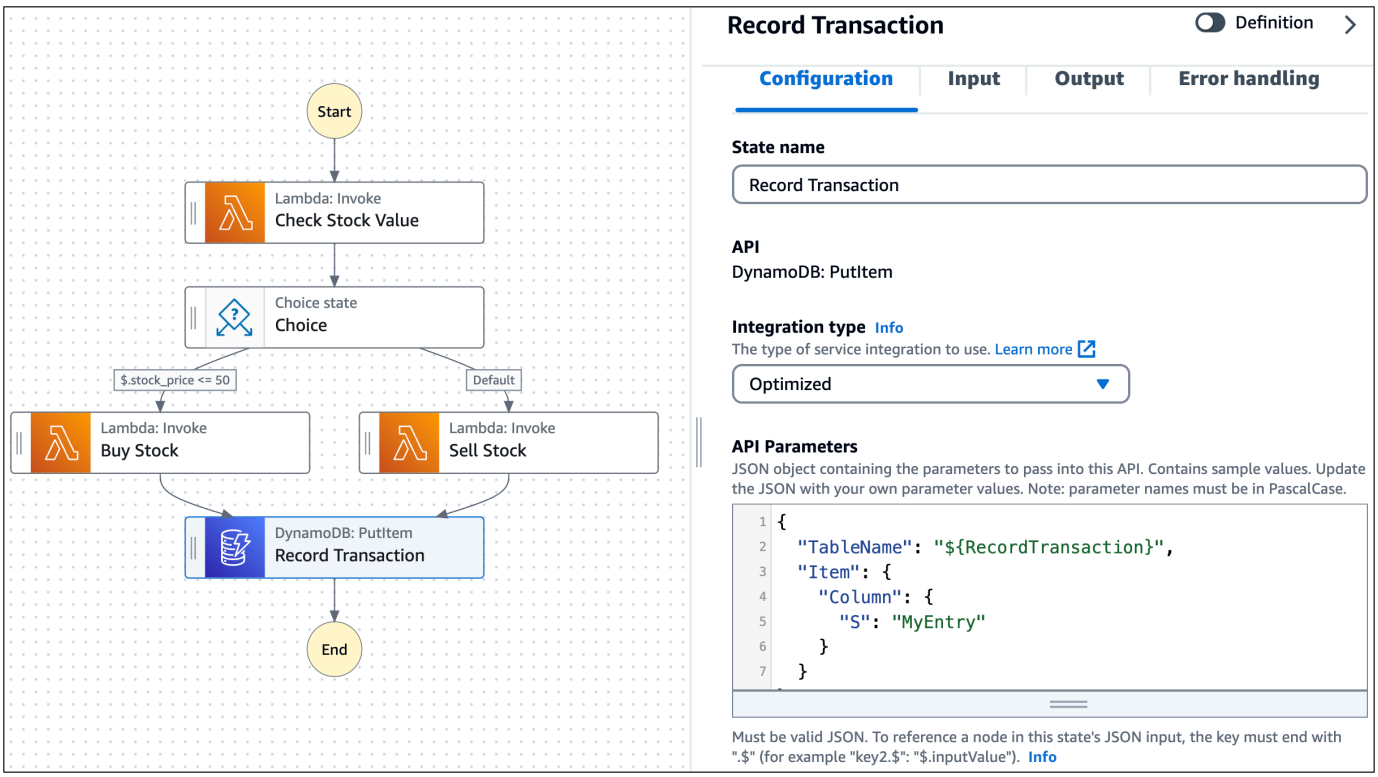
1. Open the **Configuration** tab of the task and locate the **API Parameters** field.



2. If the **API Parameters** field has a drop down option, choose **Enter a AWS CloudFormation substitution**. Then, provide a unique name.

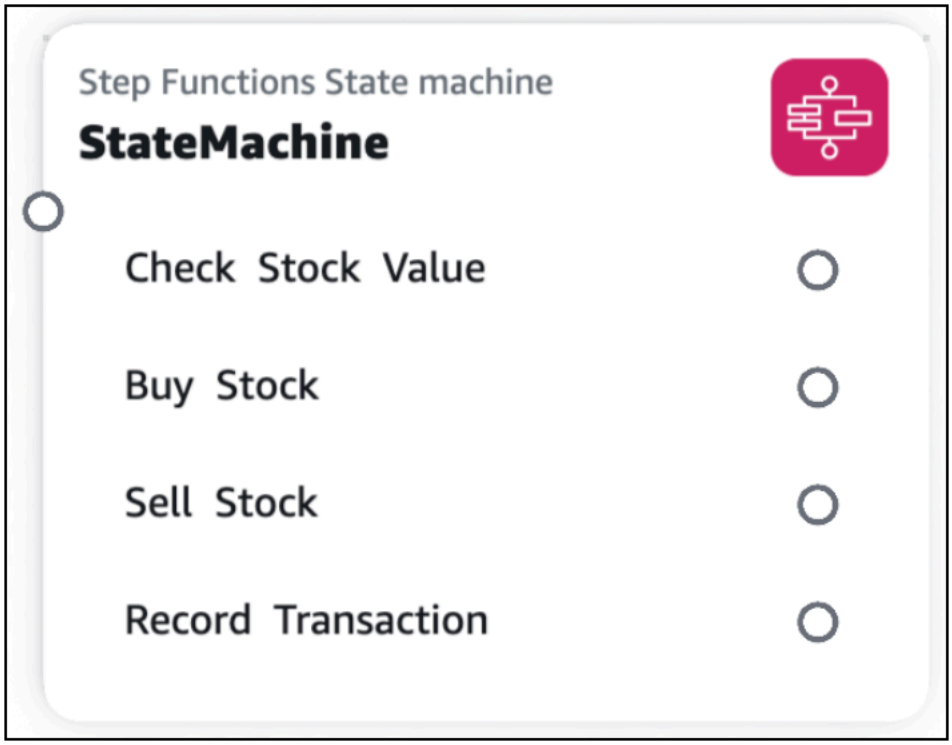
   For tasks that connect to the same resource, specify the same definition substitution for each task. To use an existing definition substitution, choose **Select a AWS CloudFormation substitution** and select the substitution to use.

3. If the **API Parameters** field contains a JSON object, modify the entry that specifies the resource name to use a definition substitution. In the following example, we change `"MyDynamoDBTable"` to `"${RecordTransaction}"`.



4. Select **Save** and **Return to Application Composer**.

The tasks from your workflow will visualize on the **Step Functions State machine** card.



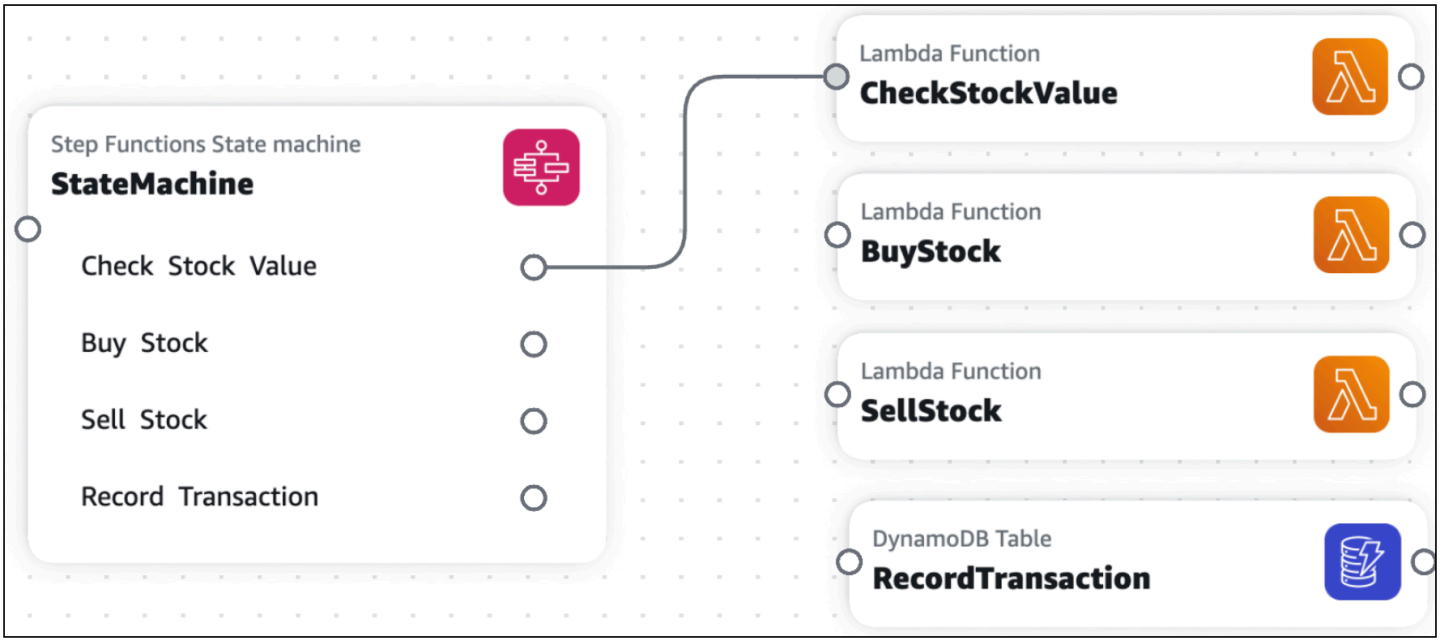
Connect resources to workflow tasks

You can create connections in Application Composer between supported workflow tasks and supported Application Composer cards.

- **Supported workflow tasks** – Tasks for AWS services that are optimized for Step Functions. To learn more, see Optimized integrations for Step Functions in the *AWS Step Functions Developer Guide*.
- **Supported Application Composer cards** – Enhanced component cards are supported. To learn more about cards in Application Composer, see Designing with cards in Application Composer.

When creating a connection, the AWS service of the task and card must match. For example, you can connect a workflow task that invokes a Lambda function to a **Lambda Function** enhanced component card.

To create a connection, click and drag the port of a task to the left port of an enhanced component card.



Application Composer will automatically update your `DefinitionSubstitution` value to define your connection. The following is an example:

```
Transform: AWS::Serverless-2016-10-31
Resources:
  StateMachine:
```

```
    Type: AWS::Serverless::StateMachine
    Properties:
      Definition:
        StartAt: Check Stock Value
        States:
          Check Stock Value:
            Type: Task
            Resource: arn:aws:states:::lambda:invoke
            Parameters:
              Payload.$: $
              FunctionName: ${CheckStockValue}
            Next: Choice
          ...
      DefinitionSubstitutions:
        CheckStockValue: !GetAtt CheckStockValue.Arn
        ...
  CheckStockValue:
    Type: AWS::Serverless::Function
    Properties:
      ...
```

IAM policies

When you connect tasks from your workflow to resources, Application Composer automatically creates the AWS Identity and Access Management (IAM) policies required to authorize the interaction between your resources. The following is an example:

```
Transform: AWS::Serverless-2016-10-31
Resources:
  StockTradingStateMachine:
    Type: AWS::Serverless::StateMachine
    Properties:
      ...
      Policies:
        - LambdaInvokePolicy:
            FunctionName: !Ref CheckStockValue
      ...
  CheckStockValue:
    Type: AWS::Serverless::Function
    ...
```

If necessary, you can add more IAM policies to your template.

Working with external files

When you create a workflow from the **Step Functions State machine** card, Application Composer saves your state machine definition within your template using the `Definition` property. You can configure Application Composer to save your state machine definition on an external file.

> **Note**
>
> To use this feature with Application Composer from the AWS Management Console, you must have **local sync** activated. For more information, see Automatically sync and save your project.

To save your state machine definition on an external file

1. Open the **Resource properties** panel of your **Step Functions State machine** card.
2. Select the **Use external file for state machine definition** option.
3. Provide a relative path and name for your state machine definition file.
4. Choose **Save**.

Application Composer will do the following:

1. Move your state machine definition from the `Definition` field to your external file.
2. Save your state machine definition in an external file using the Amazon States Language.
3. Modify your template to reference the external file using the `DefinitionUri` field.

Learn more

To learn more about Step Functions in Application Composer, see the following:

- Using Workflow Studio in Application Composer in the *AWS Step Functions Developer Guide*.
- DefinitionSubstitutions in AWS SAM templates in the *AWS Step Functions Developer Guide*.

Using Application Composer with Amazon Virtual Private Cloud (Amazon VPC)

AWS Application Composer features an integration with the Amazon Virtual Private Cloud (Amazon VPC) service. Using Application Composer, you can do the following:

- Identify the resources on your canvas that are in a VPC through a visual **VPC** tag.
- Configure AWS Lambda functions with VPCs from an external template.

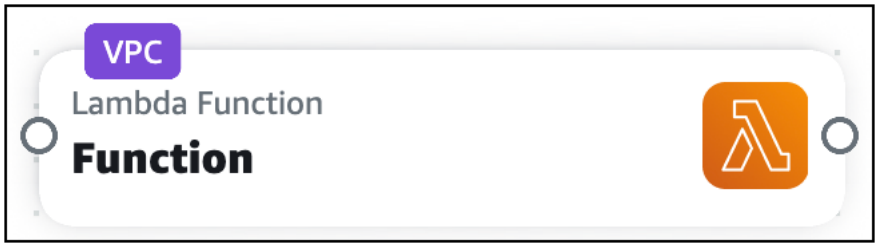
To learn more about Amazon VPC, see What is Amazon VPC? in the *Amazon VPC User Guide*.

Topics

- Identify resources in a VPC with the VPC tag
- Configure Lambda functions with external VPCs
- Importing a template with parameters
- Examples

Identify resources in a VPC with the VPC tag

Application Composer visualizes resources in a VPC using a **VPC** tag. This tag is applied to cards on the canvas. The following is an example of a Lambda function with a VPC tag:



VPC tags are applied to cards on the canvas when you do the following:

- Configure a Lambda function with a VPC in Application Composer.

- Import a template that contains resources configured with a VPC.

Configure Lambda functions with external VPCs

To start configuring a Lambda function with a VPC that is defined on another template, use the **Lambda Function** enhanced component card. This card represents a Lambda function using the AWS Serverless Application Model (AWS SAM) `AWS::Serverless::Function` resource type.

To configure a Lambda function with a VPC from an external template

1. From the **Lambda Function** resource properties panel, expand the **VPC settings (advanced)** dropdown section.
2. Select **Assign to external VPC**.
3. Provide values for the security groups and subnets to configure for the Lambda function.
4. **Save** your changes.

Security group and subnet identifiers

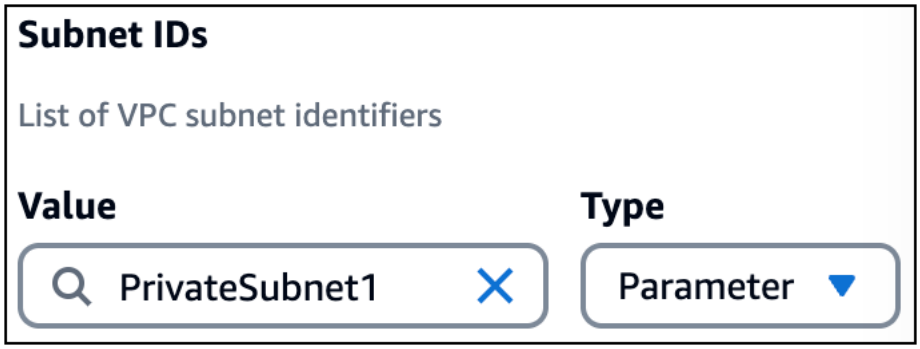
A Lambda function can be configured with multiple security groups and subnets. To configure a security group or subnet for a Lambda function, provide a value and type.

- **Value** – An identifier for the security group or subnet. Accepted values will vary based on the **type**.
- **Type** – The following types of values are allowed:
  - Parameter name
  - AWS Systems Manager (SSM) Parameter Store
  - Static value

Parameter type

The `Parameters` section of an AWS CloudFormation template can be used to store resource information across multiple templates. For more information on parameters, see Parameters in the *AWS CloudFormation User Guide*.

For the **Parameter** type, you can provide a parameter name. In the following example, we provide a `PrivateSubnet1` parameter name value:



When you provide a parameter name, Application Composer defines it in the `Parameters` section of your template. Then, Application Composer references the parameter in your Lambda function resource. The following is an example:

```
...
Resources:
  Function:
    Type: AWS::Serverless::Function
    Properties:
      ...
      VpcConfig:
        SubnetIds:
          - !Ref PrivateSubnet1
Parameters:
  PrivateSubnet1:
    Type: AWS::EC2::Subnet::Id
    Description: Parameter is generated by Application Composer
```

SSM type

The SSM Parameter Store provides a secure, hierarchical storage for configuration data management and secrets management. For more information, see AWS Systems Manager Parameter Store in the *AWS Systems Manager User Guide*.

For the **SSM** type, you can provide the following values:

- Dynamic reference to a value from the SSM Parameter Store.
- Logical ID of an `AWS::SSM::Parameter` resource defined in your template.

Dynamic reference

You can reference a value from the SSM Parameter Store using a dynamic reference in the following format: `{{resolve:ssm:reference-key}}`. For more information, see SSM parameters in the *AWS CloudFormation User Guide*.

Application Composer creates the infrastructure code to configure your Lambda function with the value from the SSM Parameter Store. The following is an example:

```
...
Resources:
  Function:
    Type: AWS::Serverless::Function
    Properties:
      ...
      VpcConfig:
        SecurityGroupIds:
          - '{{resolve:ssm:demo-app/sg-0b61d5c742dc2c773}}'
  ...
```

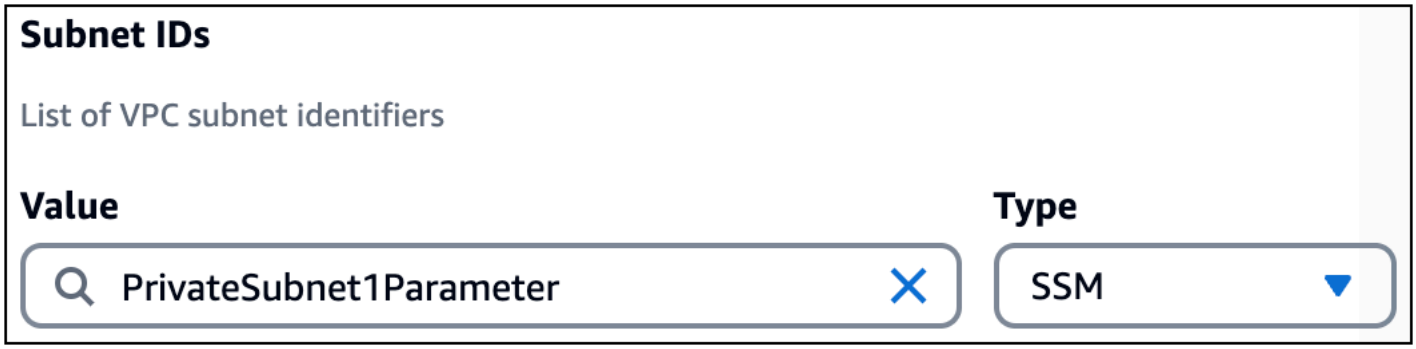
Logical ID

You can reference an `AWS::SSM::Parameter` resource in the same template by logical ID.

The following is an example of an `AWS::SSM::Parameter` resource named `PrivateSubnet1Parameter` that stores the subnet ID for `PrivateSubnet1`:

```
...
Resources:
  PrivateSubnet1Parameter:
    Type: AWS::SSM::Parameter
    Properties:
      Name: /MyApp/VPC/SubnetIds
      Description: Subnet ID for PrivateSubnet1
      Type: String
      Value: subnet-04df123445678a036
```

The following is an example of this resource value being provided by logical ID for the Lambda function:



Application Composer creates the infrastructure code to configure your Lambda function with the SSM parameter:

```
...
Resources:
  Function:
    Type: AWS::Serverless::Function
    Properties:
      ...
      VpcConfig:
        SubnetIds:
          - !Ref PrivateSubnet1Parameter
  ...
  PrivateSubnet1Parameter:
    Type: AWS::SSM::Parameter
    Properties:
      ...
```

Static value type

When a security group or subnet is deployed to AWS CloudFormation, an ID value is created. You can provide this ID as a static value.

For the **static value** type, the following are valid values:

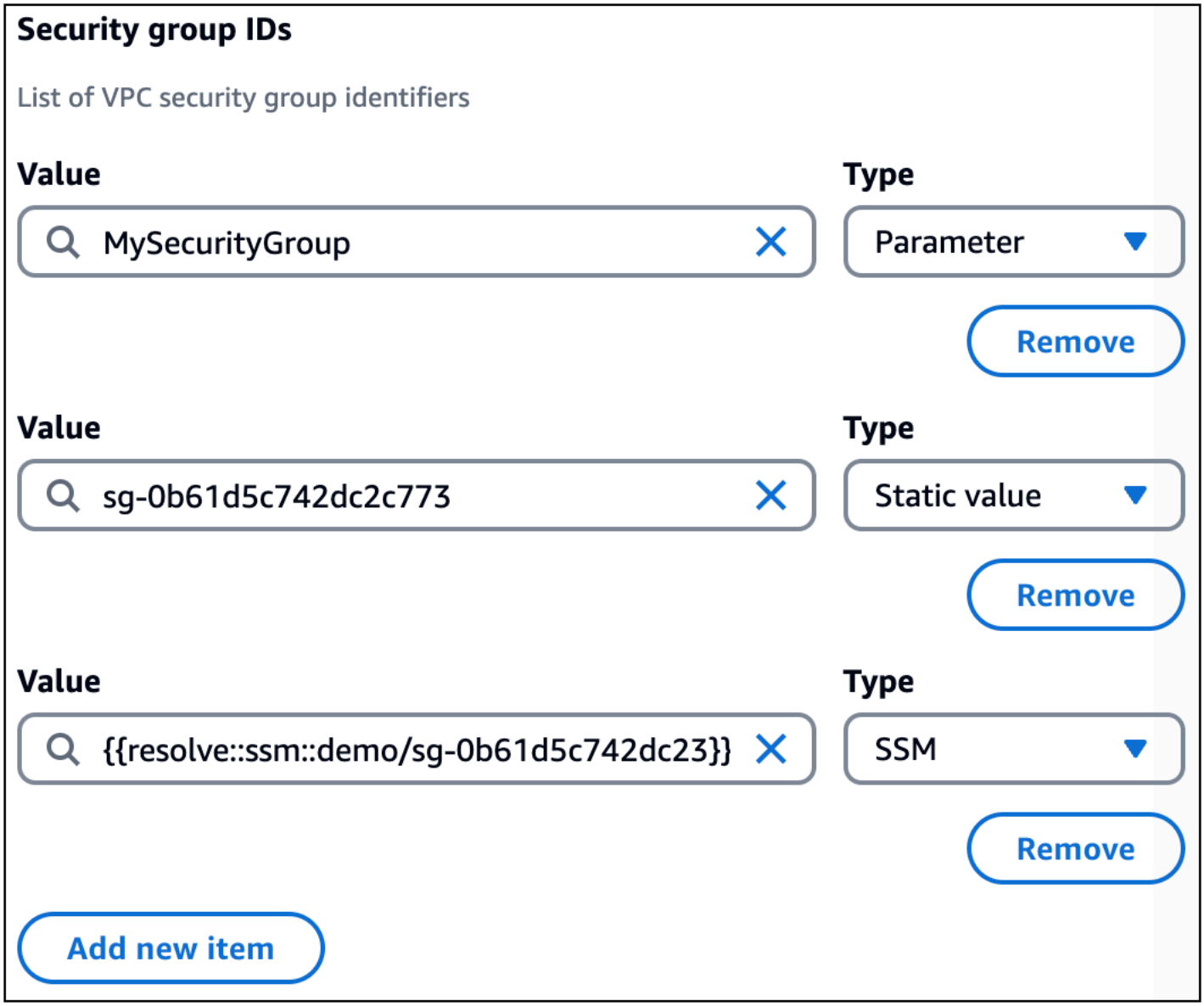
- For security groups, provide the `GroupId`. For more information, see Return values in the *AWS CloudFormation User Guide*. The following is an example: `sg-0b61d5c742dc2c773`.
- For subnets, provide the `SubnetId`. For more information, see Return values in the *AWS CloudFormation User Guide*. The following is an example: `subnet-01234567890abcdef`.

Application Composer creates the infrastructure code to configure your Lambda function with the static value. The following is an example:

```
...
Resources:
  Function:
    Type: AWS::Serverless::Function
    Properties:
      ...
      VpcConfig:
        SecurityGroupIds:
          - subnet-01234567890abcdef
        SubnetIds:
          - sg-0b61d5c742dc2c773
  ...
```

Using multiple types

For security groups and subnets, you can use multiple types together. The following is an example that configures three security groups for a Lambda function by providing values of different types:



Application Composer references all three values under the `SecurityGroupIds` property:

```
...
Resources:
  Function:
    Type: AWS::Serverless::Function
    Properties:
      ...
      VpcConfig:
        SecurityGroupIds:
          - !Ref MySecurityGroup
          - sg-0b61d5c742dc2c773
          - '{{resolve::ssm::demo/sg-0b61d5c742dc23}}'
```

```
      ...
Parameters:
  MySecurityGroup:
    Type: AWS::EC2::SecurityGroup::Id
    Description: Parameter is generated by Application Composer
```

Importing a template with parameters

When you import an existing template with parameters defined for the security groups and subnets of an external VPC, Application Composer provides a dropdown list to select your parameters from.

The following is an example of the `Parameters` section of an imported template:

```
...
Parameters:
  VPCSecurityGroups:
    Description: Security group IDs generated by Application Composer
    Type: List<AWS::EC2::SecurityGroup::Id>
  VPCSubnets:
    Description: Subnet IDs generated by Application Composer
    Type: List<AWS::EC2::Subnet::Id>
  VPCSubnet:
    Description: Subnet Id generated by Application Composer
    Type: AWS::EC2::Subnet::Id
...
```

When configuring an external VPC for a new Lambda function on the canvas, these parameters will be available from a dropdown list. The following is an example:

Limitations when importing list parameter types

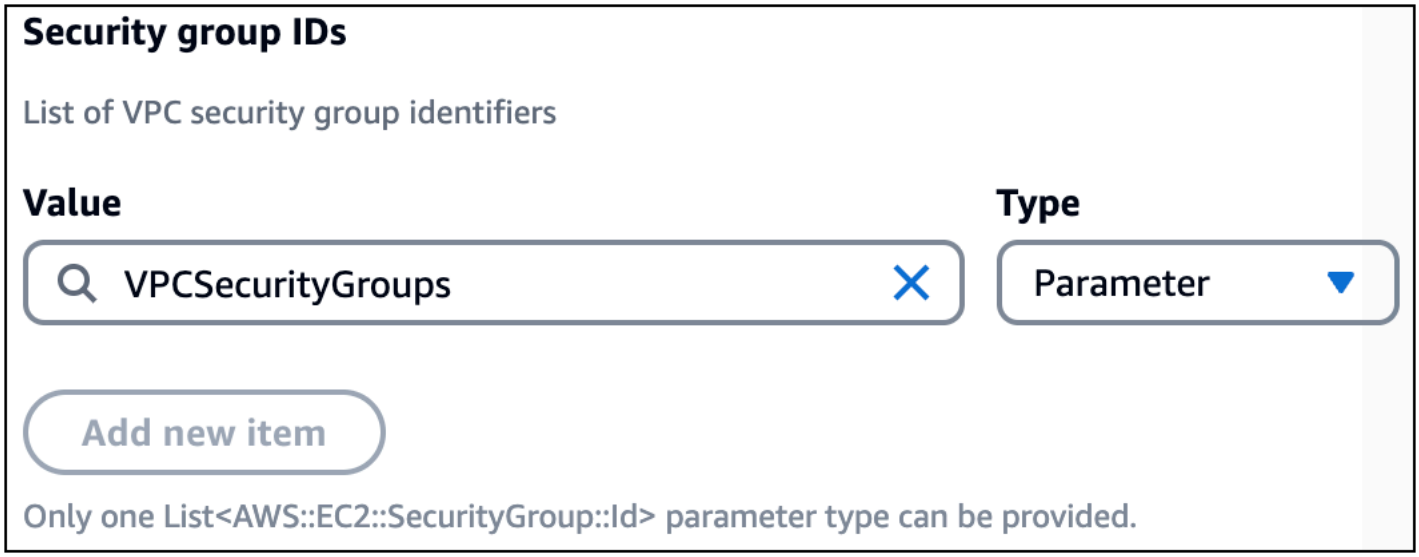
Normally, you can specify multiple security group and subnet identifiers for each Lambda function. If your existing template contains list parameter types, such as `List<AWS::EC2::SecurityGroup::Id>` or `List<AWS::EC2::Subnet::Id>`, you can only specify one identifier.

For more information on parameter lists type, see Supported AWS-specific parameter types in the *AWS CloudFormation User Guide*.

The following is an example of a template that defines `VPCSecurityGroups` as a list parameter type:

```
...
Parameters:
  VPCSecurityGroups:
    Description: Security group IDs generated by Application Composer
    Type: List<AWS::EC2::SecurityGroup::Id>
...
```

In Application Composer, if you select the `VPCSecurityGroups` value as a security group identifier for a Lambda function, you will see the following message:



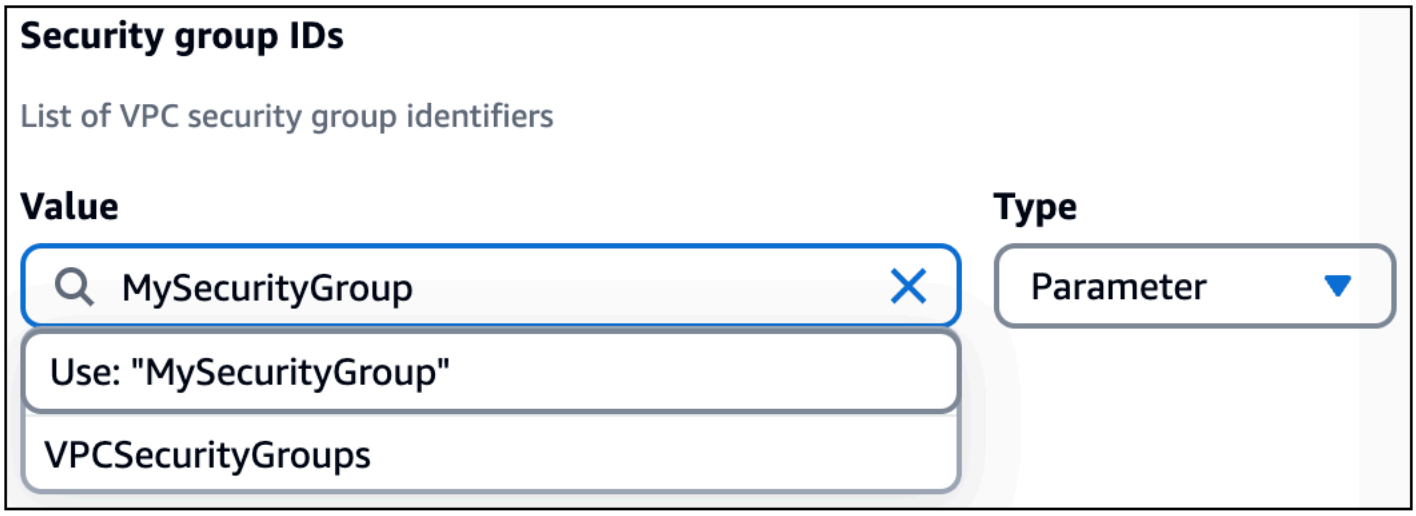
This limitation occurs because the `SecurityGroupIds` and `SubnetIds` properties of an `AWS::Lambda::Function VpcConfig` object both accept only a list of string values. Since a single list parameter type contains a list of strings, it can be the only object provided when specified.

For list parameter types, the following is an example of how they are defined in the template when configured with a Lambda function:

```
...
Parameters:
  VPCSecurityGroups:
    Description: Security group IDs generated by Application Composer
    Type: List<AWS::EC2::SecurityGroup::Id>
  VPCSubnets:
    Description: Subnet IDs generated by Application Composer
    Type: List<AWS::EC2::Subnet::Id>
Resources:
  ...
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      ...
      VpcConfig:
        SecurityGroupIds: !Ref VPCSecurityGroups
        SubnetIds: !Ref VPCSubnets
```

Adding new parameters to imported templates

When you import an existing template with parameters defined, you can also create new parameters. Instead of selecting an existing parameter from the dropdown list, provide a new type and value. The following is an example that creates a new parameter named `MySecurityGroup`:



For all new values that you provide in the **Resource properties** panel for the Lambda function, Application Composer defines them in a list under the `SecurityGroupIds` or `SubnetIds` properties of a Lambda function. The following is an example:

```
...
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      ...
      VpcConfig:
        SecurityGroupIds:
          - sg-94b3a1f6
        SubnetIds:
          - !Ref SubnetParameter
          - !Ref VPCSubnet
```

If you want to reference the logical ID of a list parameter type from an external template, we recommend that you use the **Template** view and directly modify your template. The logical ID of a list parameter type should always be provided as a single value and as the only value.

```
...
Parameters:
  VPCSecurityGroups:
    Description: Security group IDs generated by Application Composer
    Type: List<AWS::EC2::SecurityGroup::Id>
  VPCSubnets:
    Description: Subnet IDs generated by Application Composer
    Type: List<AWS::EC2::Subnet::Id>
Resources:
  ...
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      ...
      VpcConfig:
        SecurityGroupIds: !Ref VPCSecurityGroups # Valid syntax
        SubnetIds:
          - !Ref VPCSubnets # Not valid syntax
```

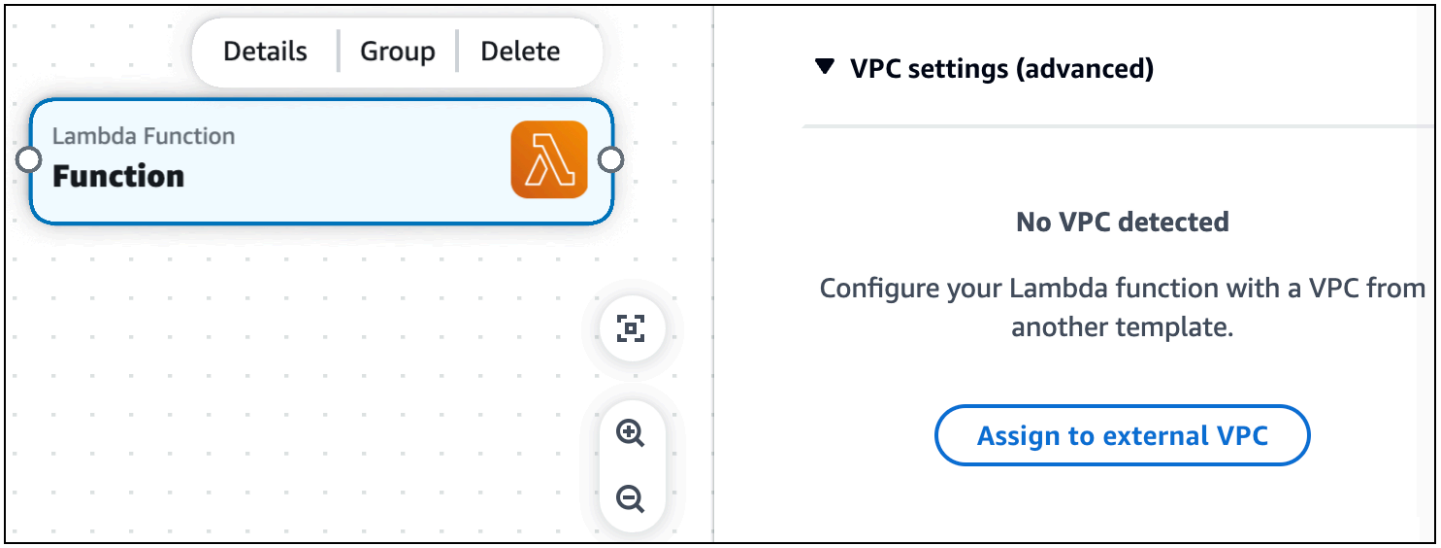
Examples

Configure a Lambda function with a VPC defined on another template

In this example, we configure a Lambda function in Application Composer with a VPC defined on another template.

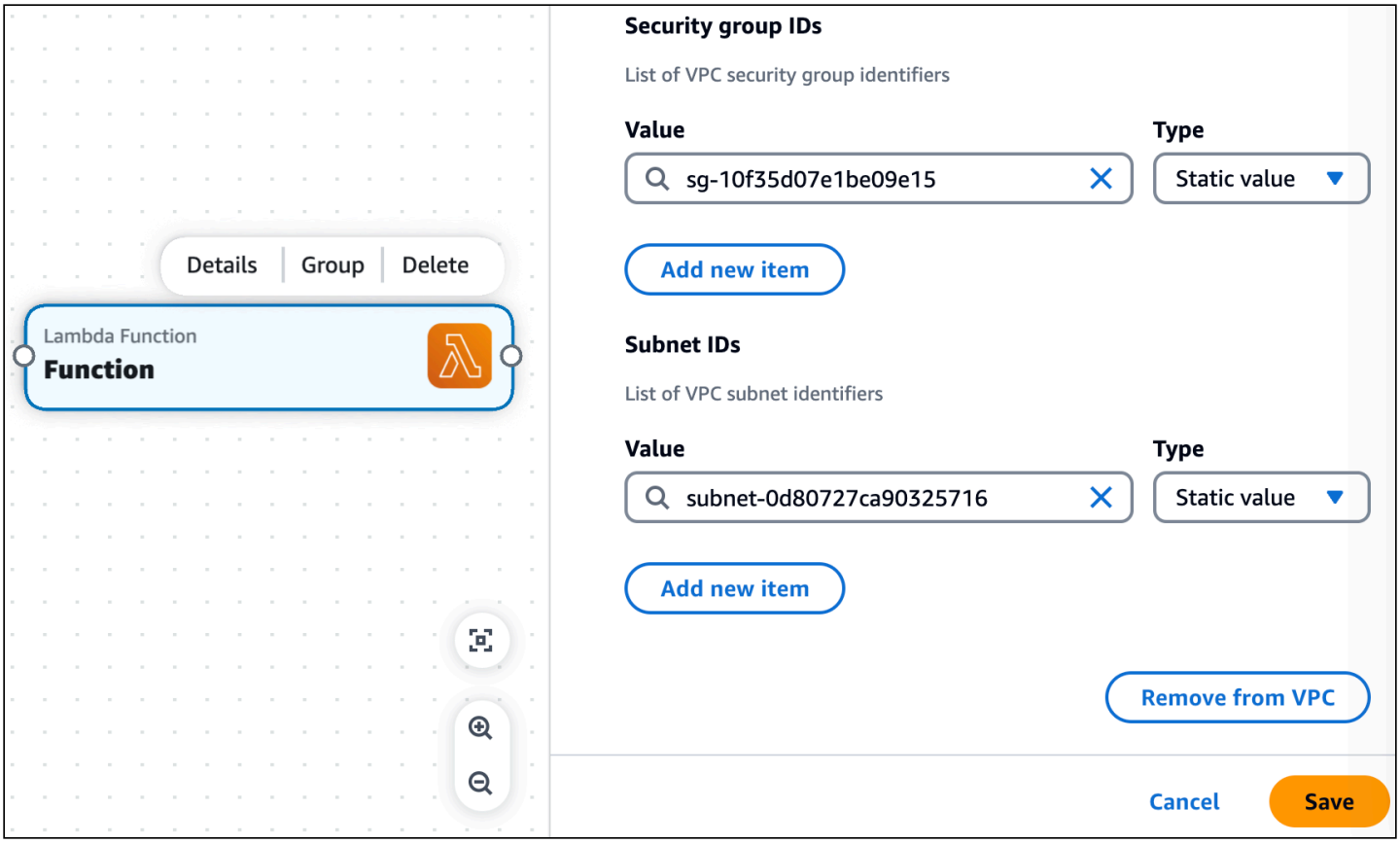
We start by dragging a **Lambda Function** enhanced component card onto the canvas.

Next, we open the card's **Resource properties** panel and expand the **VPC settings (advanced)** dropdown section.



Next, we select **Assign to external VPC** to begin configuring a VPC from an external template.

In this example, we reference a security group ID and subnet ID. These values are created when the template defining the VPC is deployed. We choose the **Static value** type and input the value of our IDs. We select **Save** when done.



Now that our Lambda function is configured with our VPC, the VPC tag is displayed on our card.

Application Composer has created the infrastructure code to configure our Lambda function with the security group and subnet of the external VPC.

```
Transform: AWS::Serverless-2016-10-31
Resources:
  Function:
    Type: AWS::Serverless::Function
    Properties:
      Description: !Sub
        - Stack ${AWS::StackName} Function ${ResourceName}
        - ResourceName: Function
      CodeUri: src/Function
```

```
      Handler: index.handler
      Runtime: nodejs18.x
      MemorySize: 3008
      Timeout: 30
      Tracing: Active
      VpcConfig:
        SecurityGroupIds:
          - sg-10f35d07e1be09e15
        SubnetIds:
          - subnet-0d80727ca90325716
  FunctionLogGroup:
    Type: AWS::Logs::LogGroup
    DeletionPolicy: Retain
    Properties:
      LogGroupName: !Sub /aws/lambda/${Function}
```

Using AWS Application Composer from the AWS Management Console

This section covers the specific use cases of using AWS Application Composer from the AWS Management Console.

For general documentation on using Application Composer, see Application Composer basics.

Topics

- Prerequisites
- Important links
- AWS Application Composer console visual overview
- Manage your project in AWS Application Composer from the AWS Management Console
- Using Application Composer with your local IDE

Prerequisites

We recommend that you complete the following topics:

Introduction

For an introduction to Application Composer, see What is AWS Application Composer?

Setting up

For instructions on setting up Application Composer, see Setting up Application Composer.

Getting started

Get started with Application Composer by following our tutorials, see Getting started with Application Composer.

Application Composer basics

To learn the basic of designing with Application Composer, see the Using AWS Application Composer basics section.

Important links

Featured services

Featured services in Application Composer provide integrations that improve the experience of designing applications with those services. To learn more, see Using Application Composer with featured AWS services.

Working with other services

Use Application Composer with other AWS services to improve your developer workflows. To learn more, see Using Application Composer with other services.

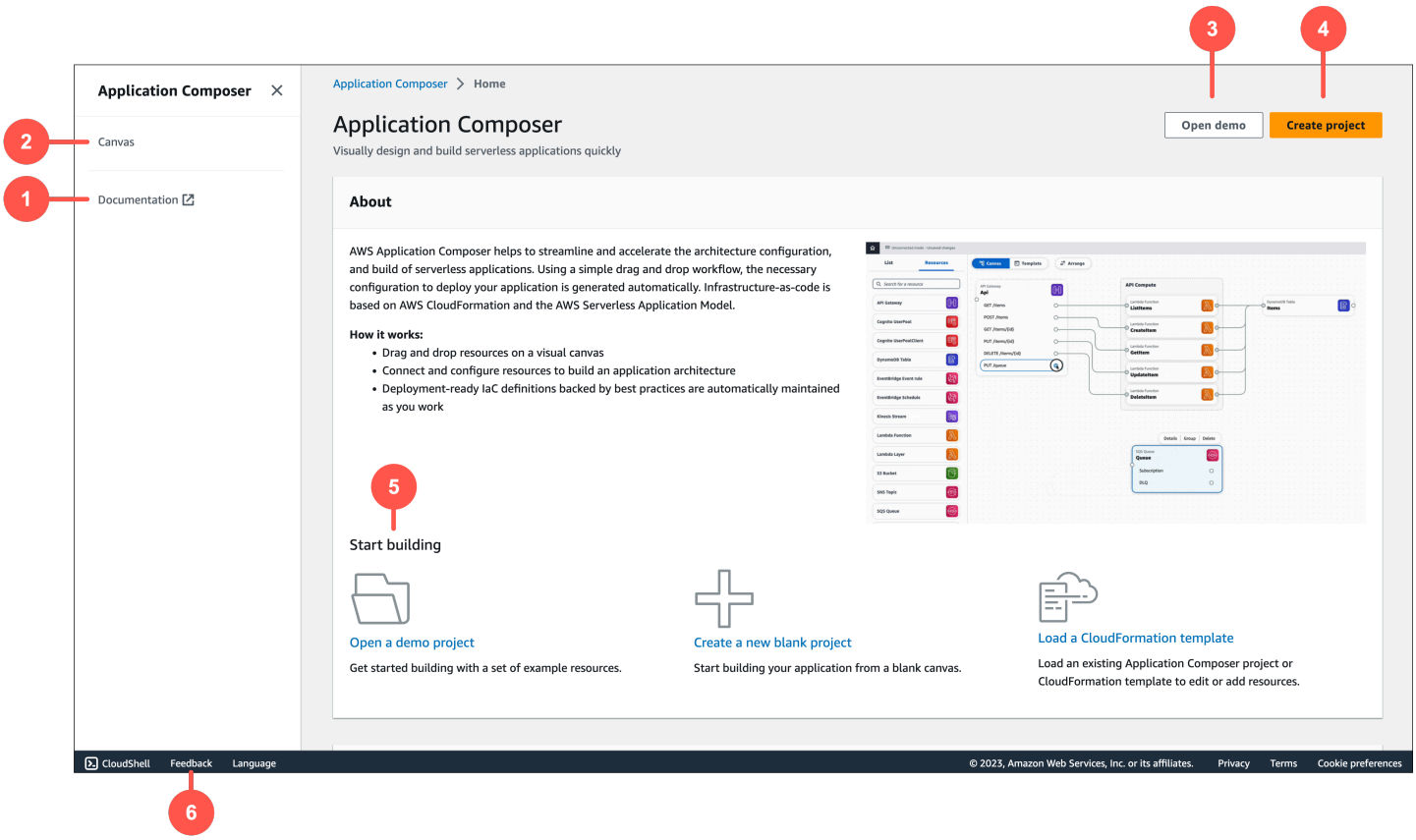
AWS Application Composer console visual overview

This section provides a visual overview of AWS Application Composer from the AWS Management Console.

Topics

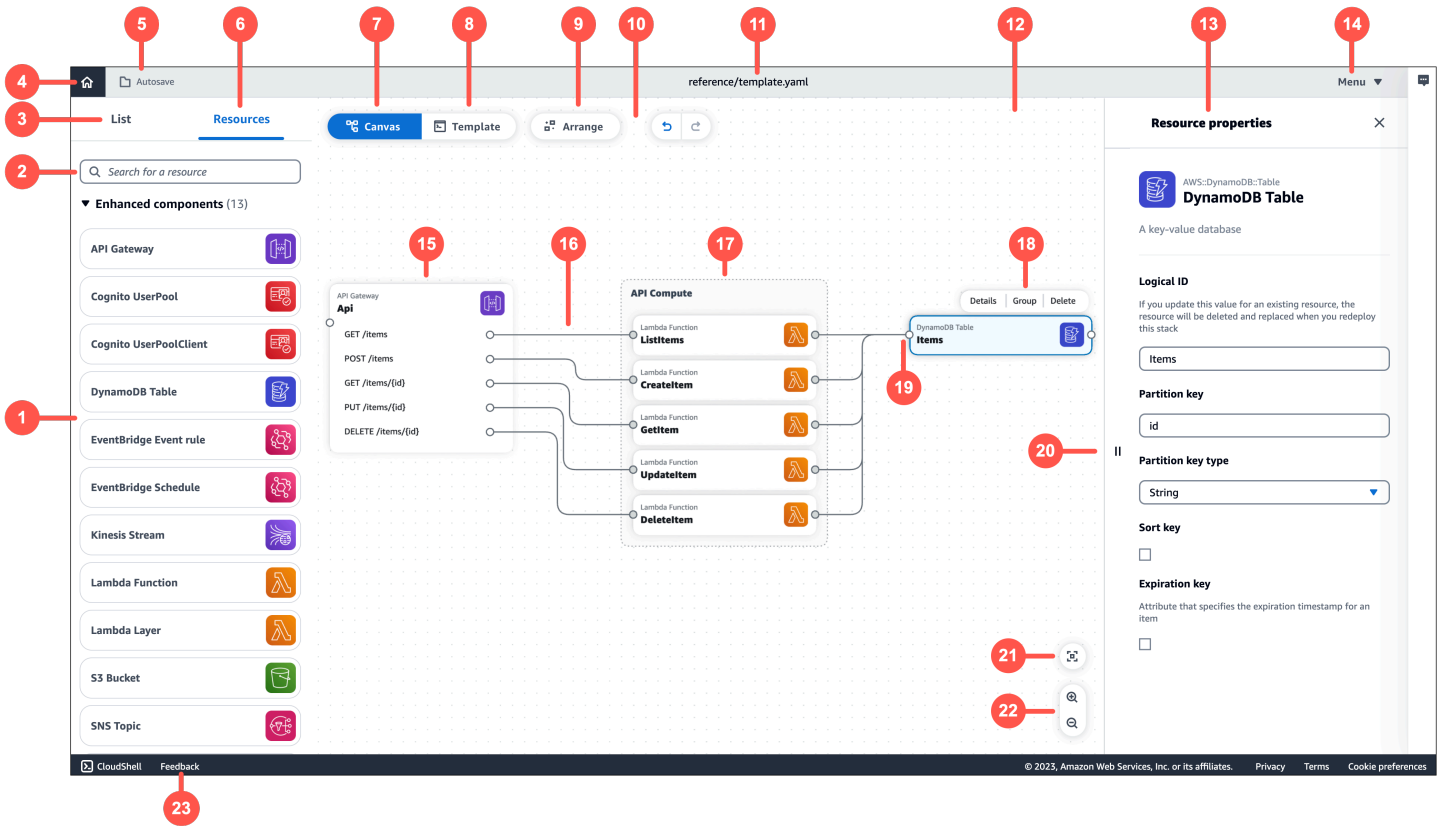
- Home page
- Visual designer
- Export canvas
- Local sync mode
- Undo and redo

Home page



1. **Documentation** – Go to Application Composer documentation.
2. **Canvas** – Go to the canvas and create or load a project.
3. **Demo** – Open the Application Composer demo application.
4. **Create project** – Create or load a project.
5. **Start building** – Quick links to start building an application.
6. **Feedback** – Go here to submit feedback.

Visual designer



1. **Resource palette** – Displays cards that you can design with.
2. **Resource search bar** – Search for cards that you can add to the canvas.
3. **List** – Displays a tree view of your application resources.
4. **Home** – Select here to go to the Application Composer homepage.
5. **Save status** – Indicates whether Application Composer changes are saved to your local machine. States include:
   - **Autosave** – **Local sync** is activated and your project is being automatically synced and saved.
   - **Changes saved** – Your application template is saved to your local machine.
   - **Unsaved changes** – Your application template has changes that are not saved to your local machine.
6. **Resources** – Displays the resource palette.
7. **Canvas** – Displays the canvas view of your application in the main view area.
8. **Template** – Displays the template view of your application in the main view area.
9. **Arrange** – Arranges your application architecture in the canvas.
10. Undo and redo – Perform **undo** and **redo** actions when supported.

11. **Template name** – Indicates the name of the template you are designing.
12. **Main view area** – Displays either the canvas or template based on your selection.
13. **Resource properties panel** – Displays relevant properties for the card that's been selected in the canvas. This panel is dynamic. Properties displayed will change as you configure your card.
14. **Menu** – Provides general options such as the following:
    - **Create a project**
    - **Open a template file or project**
    - **Save a template file**
    - **Activate local sync**
    - **Export canvas**
    - **Get support**
    - **Keyboard shortcuts**
15. **Card** – Displays a view of your card on the canvas.
16. **Line** – Represents a connection between cards.
17. **Group** – Groups selected cards together for visual organization.
18. **Card actions** – Provides actions you can take on your card.
    a. **Details** – Brings up the resource property panel.
    b. **Group** – Group selected cards together.
    c. **Delete** – Deletes the card from your canvas.
19. **Port** – Connection points to other cards.
20. **Resource property fields** – A curated set of property fields to configure for your cards.
21. **Re-center** – Re-center your application diagram on the visual canvas.
22. **Zoom** – Zoom in and out on your canvas.
23. **Feedback** – Go here to submit feedback.

Export canvas

This topic describes the AWS Application Composer console **export canvas** feature.

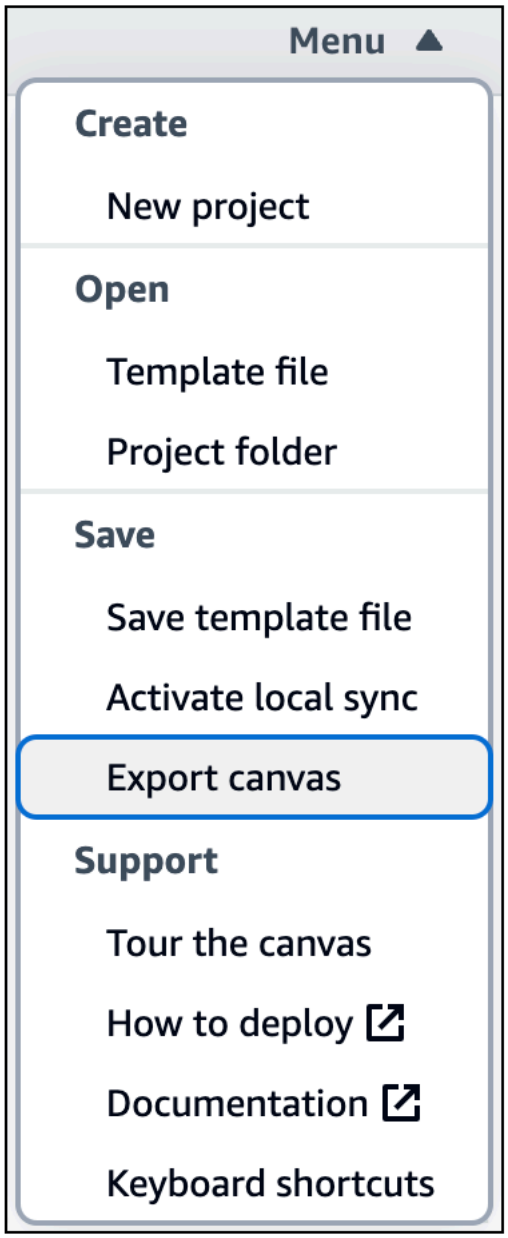
For a visual overview of all Application Composer features, see AWS Application Composer console visual overview.

About export canvas

The **export canvas** feature exports your application's canvas as an image to your local machine.

- Application Composer removes the visual designer UI elements and exports only your application's diagram.
- The default image file format is png.
- The file is exported to your local machine's default download location.

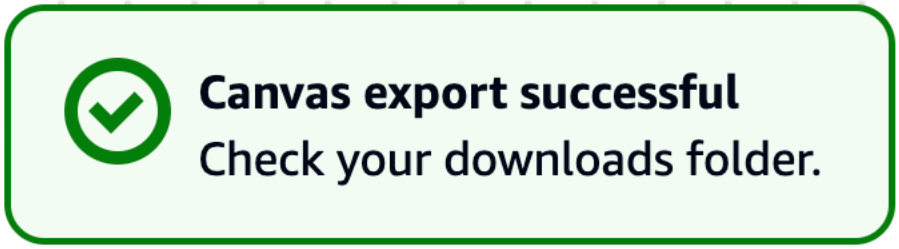
You can access the **export canvas** feature from the **Menu**.



Exporting canvas

When you export your canvas, Application Composer displays a status message.

If the export is successful, you will see the following message:



If the export was unsuccessful, you will see an error message. If you receive an error, try exporting again.

Local sync mode

This topic describes the AWS Application Composer console **local sync** mode.

For a visual overview of all Application Composer features, see AWS Application Composer console visual overview.

About local sync mode

Local sync mode automatically syncs and saves the following to your local machine:

- **Application template file** – The AWS CloudFormation or AWS Serverless Application Model (AWS SAM) template that contains your infrastructure as code (IaC).
- **Project folders** – A general directory structure that organizes your AWS Lambda functions.
- **Backup directory** – A backup directory named `.aws-composer`, created at the root of your project location. This directory contains a backup copy of your application template file and project folders.

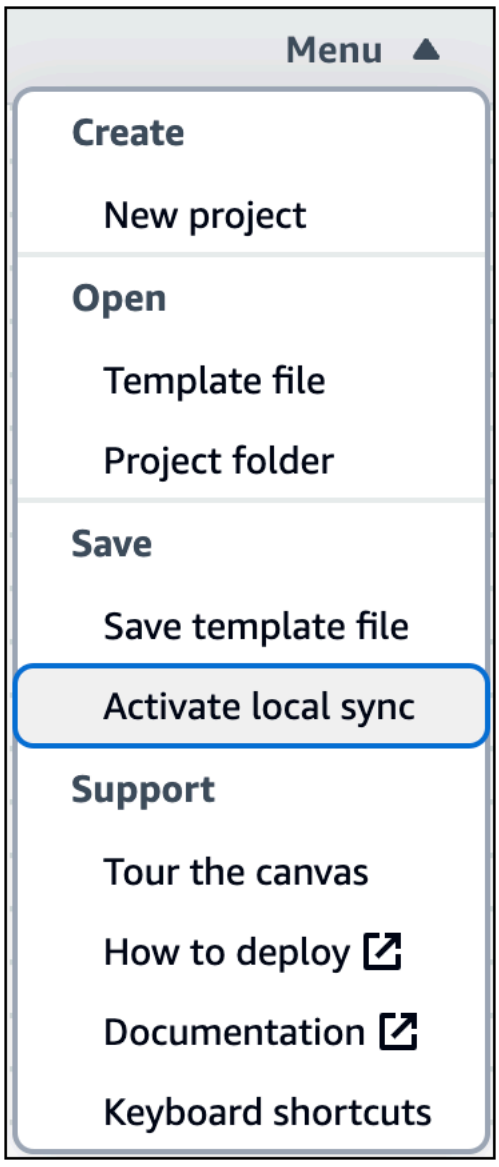
- **External files** – Supported external files that you can use within Application Composer. To learn more, see Working with templates that reference external files.

Browser requirements

Local sync mode requires a browser that supports the File System Access API. For more information, see AWS Application Composer and the File System Access API.

Activating local sync mode

Local sync mode is deactivated by default. You can activate **Local sync** mode through the Application Composer **menu**.



For instructions on activating and using **local sync**, see Automatically sync and save your project.

Undo and redo

This topic describes the AWS Application Composer console **undo** and **redo** features.

For a visual overview of all Application Composer features, see AWS Application Composer console visual overview.

About undo and redo

The **undo** and **redo** features are available as buttons on the Application Composer canvas.

- **Undo** – Revert the most recent action.
- **Redo** – Re-apply the most recently undone action.

You can also use the following keyboard shortcuts:

- **Undo** – `Control-Z | Command-Z`.
- **Redo** – `Control-Shift-Z | Command-Shift-Z`.

The **redo** feature becomes available when you perform an **undo**. Once you begin performing new actions, **redo** becomes unavailable until you perform an undo again.

Undo and redo support

You can undo and redo the following types of actions.

Application design changes in the visual canvas

This includes any modifications that you make to your application through the visual canvas. For example, dragging new resources onto the canvas or connecting resources together.

Application arrangement actions

This includes using the **Arrange** button or manually arranging a resource or group.

Application template changes

This includes any modifications to your application template from the **Template** view.

External file changes

When you modify a supported external file from within Application Composer, you can undo and redo these actions. These include changes that you make to external files within the Application Composer visual canvas, **Template** view, and **Resource properties** panel.

Application Composer remembers the last 100 actions. You can undo and redo changes up to this amount.

Local IDE support

When using Application Composer with your local IDE, Application Composer won't undo and redo any actions that you perform in your IDE. We recommend using the undo and redo features within your IDE to manage those actions.

When you use the undo and redo features of your local IDE, Application Composer will reset its memory of recent actions. This makes the Application Composer **undo** and **redo** features temporarily unavailable. Application Composer does this to prevent unintended changes that could occur when using multiple undo and redo services at the same time. As you start modifying your application in Application Composer again, the **undo** and **redo** features will become available until you use your IDE's undo and redo features again.

Manage your project in AWS Application Composer from the AWS Management Console

This topic describes how to manage your project using AWS Application Composer from the AWS Management Console.

To begin using Application Composer, you can create a new project or load an existing project. Application Composer also provides tools to integrate with your local development machine.

- For instructions on accessing Application Composer, see Setting up Application Composer.
- For an overview of the visual elements mentioned on this page, see AWS Application Composer console visual overview.

Topics

- Manage project templates and folders
- Create a new project

- Import an existing project folder
- Import an existing project template
- Save an existing project template
- Automatically sync and save your project

Manage project templates and folders

Application Composer supports applications that consist of the following:

- **Template** – An AWS CloudFormation or AWS Serverless Application Model (AWS SAM) template that defines your infrastructure code.
- **Folders** – A folder structure that organizes your project files, such as AWS Lambda function code, configuration files, and build folders.

If your browser supports Local sync mode, you can use Application Composer to manage your templates and folders. After activating local sync mode, you can do the following:

- Create a new project that consists of a starting template and folder structure.
- Load an existing project by choosing a parent folder that contains your project template and files.

With local sync mode, Application Composer automatically saves your project's template and folder changes to your local machine.

If your browser doesn't support local sync mode, or if you prefer to use Application Composer without local sync mode activated, you can create a new template or load an existing template. To save changes, you must export the template to your local machine.

Create a new project

When you create a new project, Application Composer generates a starting template. As you design your application on the canvas, your template is modified. To save your work, you must export your template or activate local sync mode.

To create a new project

1. Sign in to the Application Composer console.

2. On the **Home** page, choose **Create project**.

For instructions on activating local sync mode, see Automatically sync and save your project.

Import an existing project folder

Using local sync mode, you can import the parent folder of an existing project. If your project contains multiple templates, you can choose the template to load.

To import an existing project from the Home page

1. Sign in to the Application Composer console.
2. On the **Home** page, choose **Load a CloudFormation template**.
3. For **Project location**, choose **Select folder**. Select your project's parent folder and choose **Select**.

> **Note**
>
> If you do not receive this prompt, your browser may not support the File System Access API, which is required for local sync mode. For more information, see AWS Application Composer and the File System Access API.

4. When prompted by your browser, select **View files**.
5. For **Template file**, choose your template from the dropdown list. If your project contains a single template, Application Composer automatically selects it for you.
6. Choose **Create**.

To import an existing project from the canvas

1. From the canvas, choose **Menu** to open the menu.
2. In the **Open** section, choose **Project folder**.

> **Note**
>
> If the **Project folder** option is unavailable, your browser may not support the File System Access API, which is required for local sync mode. For more information, see AWS Application Composer and the File System Access API.

3. For **Project location**, choose **Select folder**. Select your project's parent folder and choose **Select**.
4. When prompted by your browser, select **View files**.
5. For **Template file**, choose your template from the dropdown list. If your project contains a single template, Application Composer automatically selects it for you.
6. Choose **Create**.

When you import an existing project folder, Application Composer activates **local sync mode**. Changes made to your project's template or files are automatically saved to your local machine.

Import an existing project template

When you import an existing AWS CloudFormation or AWS SAM template, Application Composer automatically generates a visualization of your application architecture on the canvas.

You can import a project template from your local machine.

To import an existing project template

1. Sign in to the Application Composer console.
2. Choose **Create project** to open a blank canvas.
3. Choose **Menu** to open the menu.
4. In the **Open** section, choose **Template file**.
5. Select your template and choose **Open**.

To save changes to your template, you must export your template or activate local sync mode.

Save an existing project template

If you don't use local sync mode, you must export your template to save your changes. If you have local sync mode activated, manually saving your template is not required. Changes are automatically saved to your local machine.

To save an existing project template

1. From the Application Composer canvas, choose **Menu** to open the menu.

2. In the **Save** section, choose **Save template file**.
3. Provide a name for your template.
4. Select a location to save your template.
5. Choose **Save**.

Automatically sync and save your project

Use Application Composer **local sync** to automatically sync and save your project to your local machine. For more information about local sync, see Local sync mode.

We recommend that you use **local sync** for the following reasons:

- By default, you need to manually save your application template as you design. Use **local sync** to automatically save your application template to your local machine as you make changes.
- **Local sync** manages and automatically syncs your project folders, backup folder, and supported external files to your local machine.
- When using **local sync**, you can connect Application Composer with your local IDE to speed up development. To learn more, see Using Application Composer with your local IDE.

You can activate **local sync** for a new project, or load an existing project with **local sync** activated.

To activate local sync for a new project

1. From the Application Composer **home** page, select **Create project**.
2. From the Application Composer **menu**, select **Activate local sync**.
3. For **Project location**, press **Select folder** and choose a directory. This is where Application Composer will save and sync your template files and folders as you design.

   > **Note**
   >
   > The project location must not contain an existing application template.

4. When prompted to allow access, select **View files**.
5. Press **Activate**. When prompted to save changes, select **Save changes**.

   When activated, the **Autosave** indicator will be displayed in the upper-left area of your canvas.

To load an existing project with local sync activated

1. From the Application Composer **home** page, select **Load a AWS CloudFormation template**.
2. From the Application Composer **menu**, select **Open** > **Project folder**.
3. For **Project location**, press **Select folder** and choose the root folder of your project.
4. When prompted to allow access, select **View files**.
5. For **Template file**, select your application template and press **Create**.
6. When prompted to save changes, select **Save changes**.

   When activated, the **Autosave** indicator will be displayed in the upper-left area of your canvas.

Using Application Composer with your local IDE

Use AWS Application Composer from the AWS Management Console with **local sync** mode to connect with your local integrated development environment (IDE).

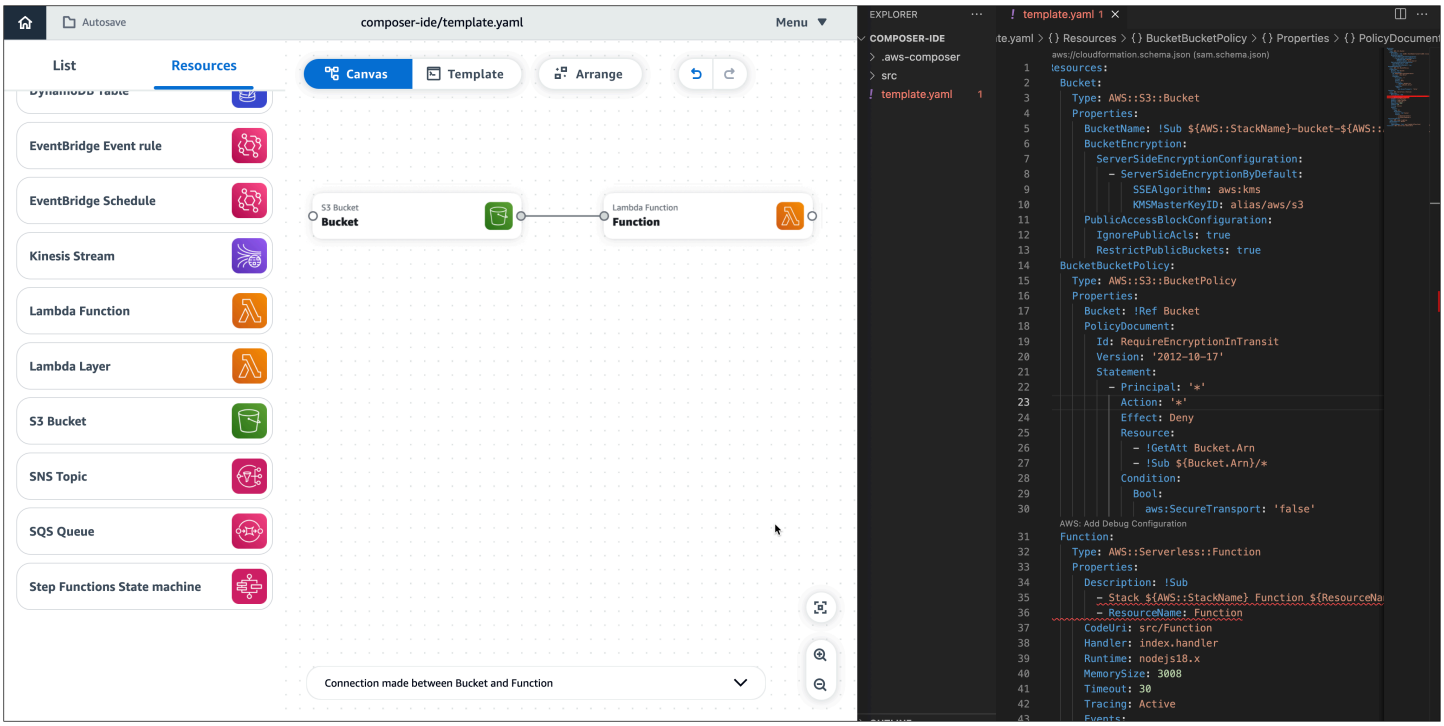
- For more information about **local sync** mode, see Local sync mode.
- For instructions on using **local sync** mode, see Automatically sync and save your project.

Benefits of using Application Composer with your local IDE

As you design in Application Composer, your local template and project directory are automatically synced and saved.

You can use your local IDE to view changes and modify your templates. Changes that you make locally are automatically synced to Application Composer.

You can use local tools such as the AWS Serverless Application Model Command Line Interface (AWS SAM CLI) to build, test, deploy your application, and more.

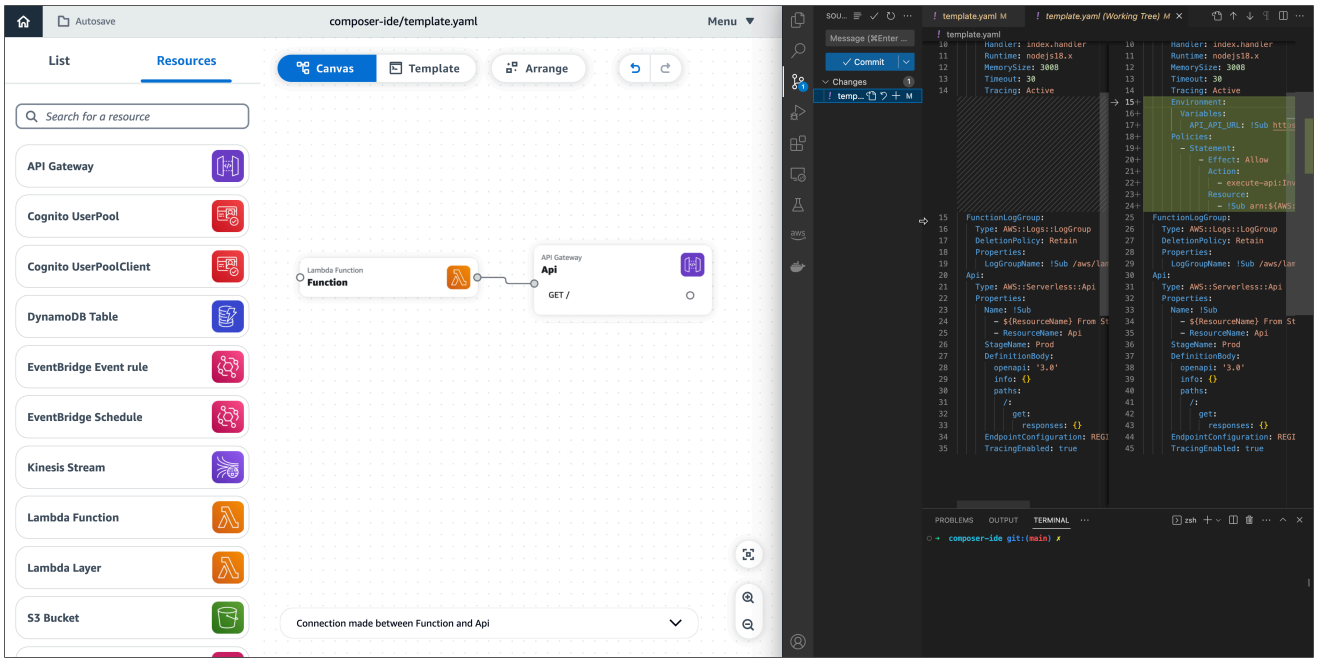


Integrate Application Composer with your local IDE

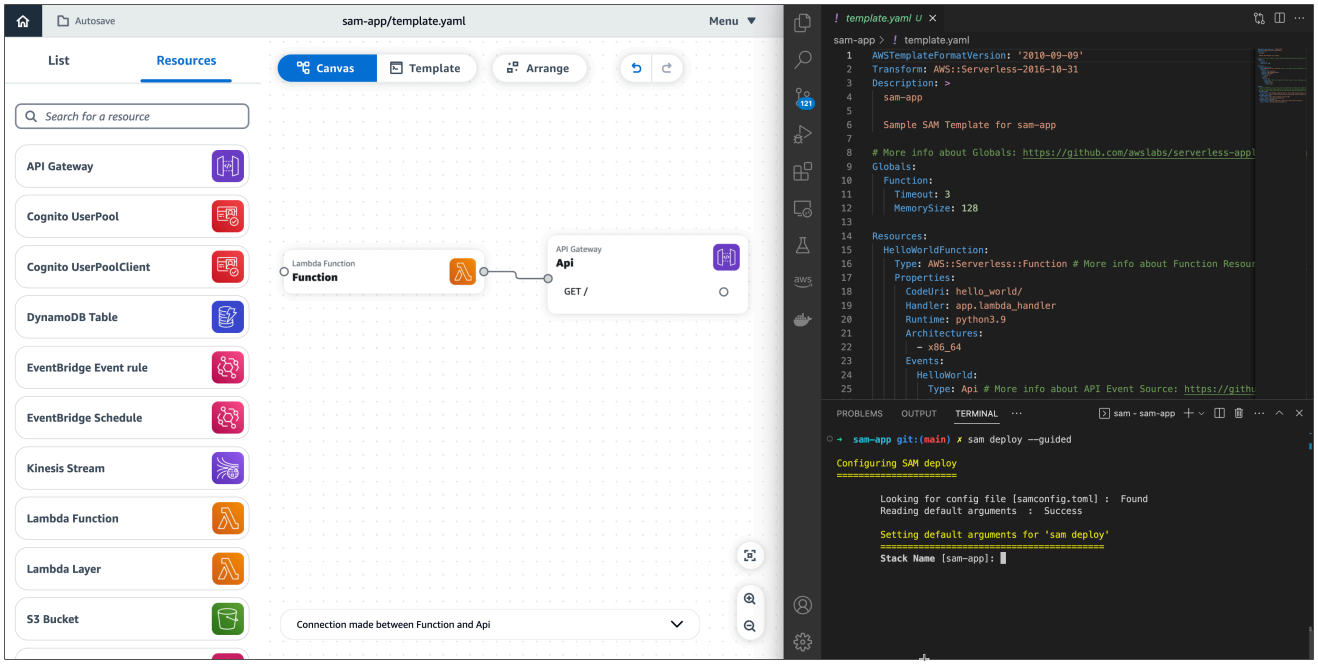
To integrate Application Composer with your local IDE

1. In Application Composer, create or load a project with **local sync** activated.
2. In your local IDE, open the same project folder as Application Composer.
3. Use Application Composer with your local IDE. Updates made in Application Composer will automatically sync with your local machine. Here are some examples of what you can do:

    a. Use your version control system of choice to track updates being performed by Application Composer.



b. Use the AWS SAM CLI locally to build, test, deploy your application, and more. To learn more, see [Using Application Composer with AWS CloudFormation and AWS SAM](#).



Using AWS Application Composer from the AWS Toolkit for Visual Studio Code

This section covers the specific use cases of using AWS Application Composer from the AWS Toolkit for Visual Studio Code.

For general documentation on using Application Composer, see Application Composer basics.

Topics

- Prerequisites
- Important links
- AWS Application Composer from the AWS Toolkit for Visual Studio Code visual overview
- Manage your project in AWS Application Composer from the Toolkit for VS Code
- Deploy your application with sam sync
- Using AWS Application Composer with Amazon CodeWhisperer

Prerequisites

We recommend that you complete the following topics:

Introduction

For an introduction to Application Composer, see What is AWS Application Composer?

Setting up

For instructions on setting up and accessing Application Composer, see Setting up Application Composer.

Getting started

Get started with Application Composer by following our tutorials, see Getting started with Application Composer.

Application Composer basics

To learn the basic of designing with Application Composer, see the Using AWS Application Composer basics section.

Important links

Featured services

Featured services in Application Composer provide integrations that improve the experience of designing applications with those services. To learn more, see Using Application Composer with featured AWS services.

Working with other services

Use Application Composer with other AWS services to improve your developer workflows. To learn more, see Using Application Composer with other services.

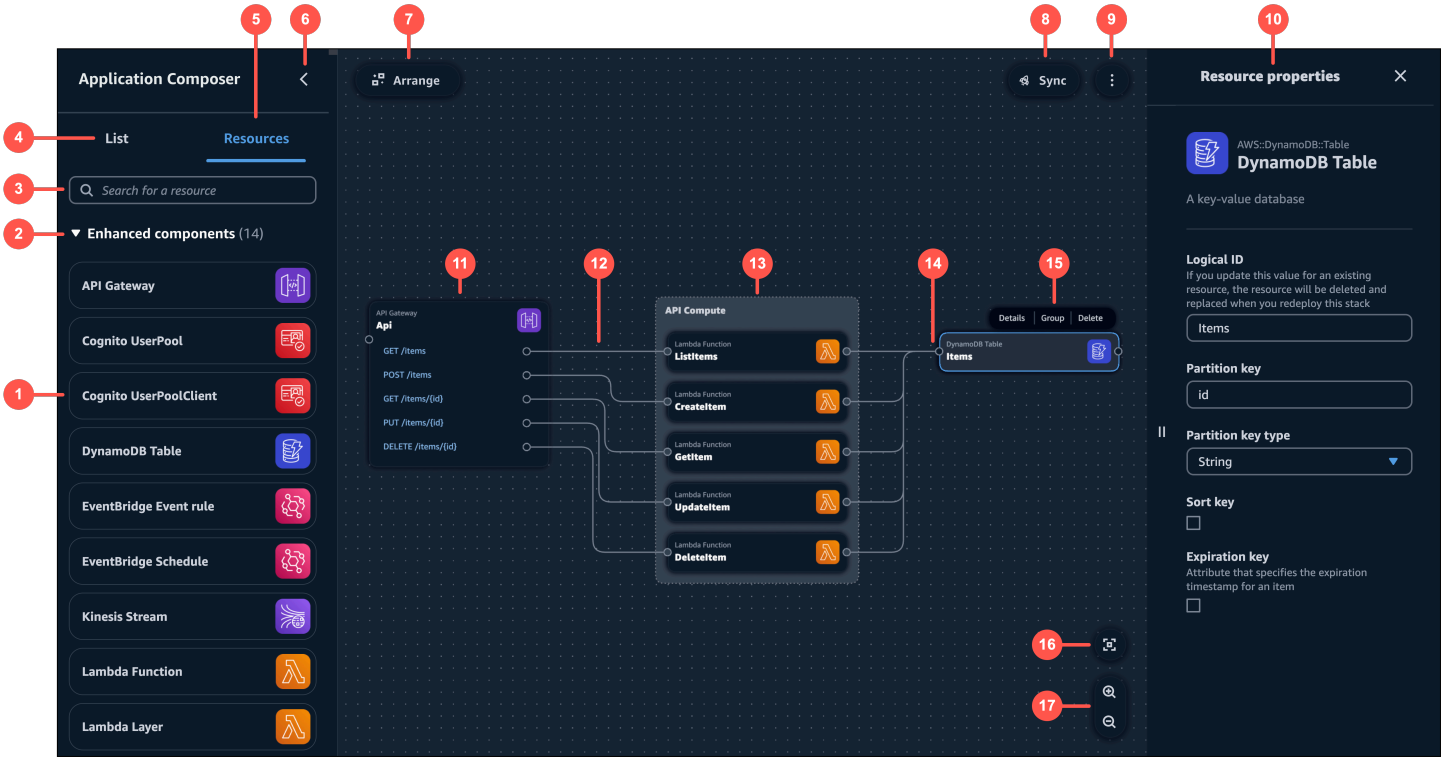
AWS Application Composer from the AWS Toolkit for Visual Studio Code visual overview

This section provides a visual overview of AWS Application Composer from the AWS Toolkit for Visual Studio Code.

Topics

- Visual designer

Visual designer



1. **Resource palette** – Displays cards that you can design with.
2. **Card categories** – Cards are organized by categories unique to Application Composer.
3. **Resource search bar** – Search for cards that you can add to the canvas.
4. **List** – Displays a tree view of your application resources.
5. **Resources** – Displays the resource palette.
6. **Left pane toggle** – Hide or show the left pane.
7. **Arrange** – Arranges your application architecture in the canvas.
8. **Sync** – Initiates the AWS Serverless Application Model (AWS SAM) CLI `sam sync` command to deploy your application.
9. **Menu** – Provides general options such as the following:
   - **Export canvas**
   - **Tour the canvas**
   - Links to **Documentation**
   - Keyboard shortcuts
10. **Resource properties panel** – Displays relevant properties for the card that's been selected in the canvas. This panel is dynamic. Properties displayed will change as you configure your card.

11. **Card** – Displays a view of your card on the canvas.
12. **Line** – Represents a connection between cards.
13. **Group** – A group of cards. You can group cards for visual organization.
14. **Port** – Connection points to other cards.
15. **Card actions** – Provides actions you can take on your card.
    - **Details** – Brings up the **Resource properties** panel.
    - **Group** – Group selected cards together.
    - **Delete** – Deletes the card from your canvas and template.
16. **Re-center** – Re-center your application diagram on the visual canvas.
17. **Zoom** – Zoom in and out on your canvas.

Manage your project in AWS Application Composer from the Toolkit for VS Code

This topic describes how to manage your project using AWS Application Composer from the AWS Toolkit for Visual Studio Code.

To visualize your application in Application Composer, you can launch Application Composer from any AWS CloudFormation or AWS Serverless Application Model (AWS SAM) template in VS Code. For instructions, see Accessing Application Composer from the AWS Toolkit for Visual Studio Code.

Deploy your application with sam sync

Use the **sync** button in AWS Application Composer from the AWS Toolkit for Visual Studio Code to deploy your application to the AWS Cloud.

The **sync** button initiates the AWS Serverless Application Model Command Line Interface (AWS SAM CLI) `sam sync` command.

Topics

- What is sam sync?
- Setting up
- Syncing your application

What is sam sync?

The `sam sync` command can deploy new applications or quickly sync changes that you make locally to the AWS Cloud. Running `sam sync` may include the following:

- Building your application with `sam build` to prepare your local application files for deployment by creating or updating a local `.aws-sam` directory.
- For resources that support AWS service APIs, the AWS SAM CLI will use the APIs to deploy your changes. The AWS SAM CLI does this to quickly update your resources in the cloud.
- If necessary, the AWS SAM CLI performs an AWS CloudFormation deployment to update your entire stack through a change set.

The `sam sync` command is best suited for rapid development environments when quickly updating your cloud resources can benefit your development and testing workflows.

To learn more about `sam sync`, see Using sam sync in the *AWS Serverless Application Model Developer Guide*.

Setting up

To use the **sync** feature in Application Composer, you must have the AWS SAM CLI installed on your local machine. For instructions, see Installing the AWS SAM CLI in the *AWS Serverless Application Model Developer Guide*.

When you use the **sync** feature in Application Composer, the AWS SAM CLI references your configuration file for the information it needs to sync your application to the AWS Cloud. For instructions on creating, modifying, and using configuration files, see Configure project settings in the *AWS Serverless Application Model Developer Guide*.

Syncing your application

To sync your application to the AWS Cloud

1. Select the **sync** button on the Application Composer canvas.
2. You may receive a prompt to confirm that you are working with a development stack. Select **OK** to continue.
3. Application Composer may prompt you to configure the following options:

- **AWS Region** – The region to sync your application to.
- **AWS CloudFormation stack name** – The name of your AWS CloudFormation stack. You can select an existing stack name or create a new one.
- **Amazon Simple Storage Service (Amazon S3) bucket** – The name of your Amazon S3 bucket. The AWS SAM CLI will package and store your application files and function code here. You can select an existing bucket or create a new one.

Application Composer will initiate the AWS SAM CLI `sam sync` command and open a terminal window in your IDE to output its progress.

Using AWS Application Composer with Amazon CodeWhisperer

AWS Application Composer from the AWS Toolkit for Visual Studio Code provides an integration with Amazon CodeWhisperer. You can use CodeWhisperer within Application Composer to generate the infrastructure code for your AWS resources as you design your application.

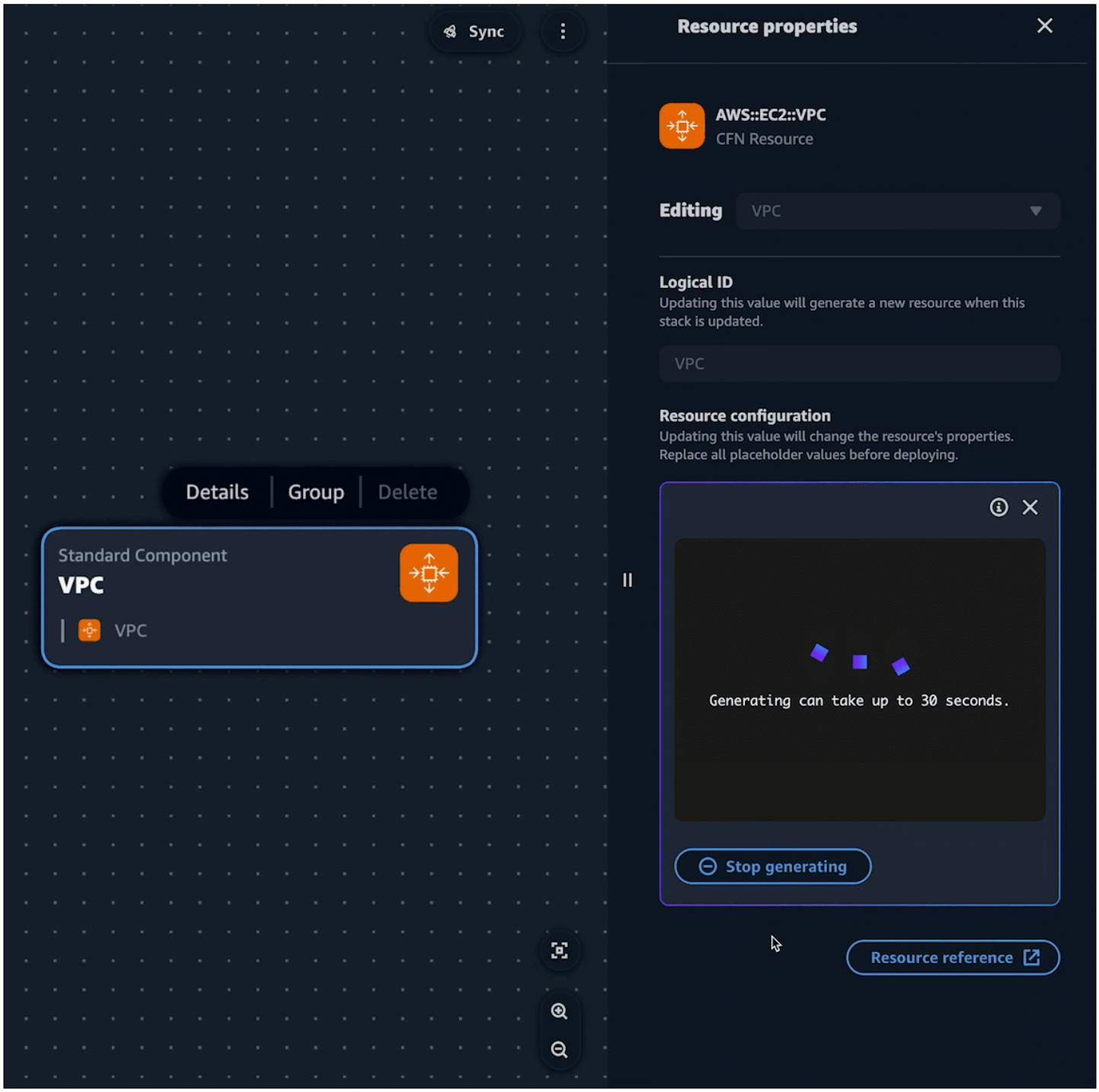
CodeWhisperer is a general purpose, machine learning-powered code generator. To learn more, see What is CodeWhisperer? in the *Amazon CodeWhisperer User Guide*.

Topics

- What is CodeWhisperer support in Application Composer?
- Setting up
- Using CodeWhisperer in Application Composer
- Learn more

What is CodeWhisperer support in Application Composer?

For **standard resource** and **standard component** cards, you can use CodeWhisperer to generate infrastructure code suggestions for your resources.



Standard resource and **standard component** cards can represent an AWS CloudFormation resource or a collection of AWS CloudFormation resources. To learn more, see Designing with cards in Application Composer.

Setting up

To use CodeWhisperer in Application Composer, you must authenticate with CodeWhisperer in the Toolkit. For instructions, see Getting started with CodeWhisperer in VS Code and JetBrains in the *Amazon CodeWhisperer User Guide*.

Using CodeWhisperer in Application Composer

You can use CodeWhisperer from the **Resource properties** panel of any **standard resource** or **standard component** card.

To use CodeWhisperer in Application Composer

1. From a **standard resource** or **standard component** card, open the **Resource properties** panel.
2. Locate the **Resource configuration** field. This field contains the infrastructure code for the card.
3. Select the **Generate suggestions** button. CodeWhisperer will generate a suggestion.

   > **Note**
   >
   > Code generated at this stage will not overwrite existing infrastructure code from your template.

4. To generate more suggestions, select **Regenerate**. You can toggle through the samples to compare results.
5. To select an option, choose **Select**. You can modify the code here before saving it to your application. To exit without saving, select the **exit icon (X)**.
6. To save the code to your application template, select **Save** from the **Resource properties** panel.

Learn more

To learn more about CodeWhisperer, see What is CodeWhisperer? in the *Amazon CodeWhisperer User Guide*.

Using Application Composer with other services

Integrate AWS Application Composer into your development workflows by using it with other services.

Topics

- Using Application Composer with AWS CloudFormation and AWS SAM
- Using AWS Application Composer with the AWS Lambda console

Using Application Composer with AWS CloudFormation and AWS SAM

Use AWS Application Composer to design deployment-ready serverless applications. To deploy, use any AWS CloudFormation compatible service. We recommend using the AWS Serverless Application Model (AWS SAM).

Topics

- What is AWS SAM?
- AWS SAM prerequisites
- Using Application Composer with the AWS SAM CLI
- Examples

What is AWS SAM?

AWS SAM is an open-source framework that provides developer tools for building and running serverless applications on AWS. The AWS SAM toolkit consists of two primary parts:

1. AWS SAM template specification
2. AWS SAM Command Line Interface (AWS SAM CLI)

AWS SAM template specification

The AWS SAM template specification contains a short-hand syntax and structure that you can use to define the infrastructure of your AWS serverless applications.

- AWS SAM templates can be defined in JSON and YAML.
- Its syntax is short-hand, making it quicker to learn and code. Less code means fewer errors and faster development.

AWS SAM templates are an extension of AWS CloudFormation, which is a service that provisions resources at AWS.

- AWS SAM templates are automatically transformed into the AWS CloudFormation template syntax at deployment.
- You can deploy AWS SAM templates directly to AWS CloudFormation to create your application resources.

When you design your application in Application Composer, AWS SAM templates are automatically created for you. You can select the **Template view** to view and modify your AWS SAM template.

AWS SAM CLI

The AWS SAM CLI is a command line tool that helps you manage your serverless applications through their entire development lifecycle. You can use the AWS SAM CLI to:

- Prepare your application for deployment.
- Perform local debugging and testing.
- Deploy your application.
- Develop and sync local changes to the cloud.
- And more!

The AWS SAM CLI is a great companion to Application Composer. Use Application Composer to design deployment-ready applications. Then use the AWS SAM CLI to deploy and manage your applications.

To learn more about AWS SAM, see What is AWS SAM? in the *AWS Serverless Application Model Developer Guide*.

AWS SAM prerequisites

Install the AWS CLI

We recommend installing and setting up the AWS CLI before installing the AWS SAM CLI. For instructions, see AWS SAM prerequisites in the *AWS Serverless Application Model Developer Guide*.

> **Note**
>
> After installing the AWS CLI, you must configure AWS credentials. To learn more, see Quick setup in the *AWS Command Line Interface User Guide*.

Install the AWS SAM CLI

To install the AWS SAM CLI, see Installing the AWS SAM CLI in the *AWS Serverless Application Model Developer Guide*.

Using Application Composer with the AWS SAM CLI

Application Composer from the console

If you use Application Composer from the AWS Management Console, you have the following options to use the AWS SAM CLI.

Activate local sync mode

With local sync mode, your project folder, including the AWS SAM template, are automatically saved to your local machine. Application Composer structures your project directory in a way that AWS SAM recognizes. You can run the AWS SAM CLI from the root directory of your project.

For more information about **local sync** mode, see Local sync mode.

Export your template

You can export your template to your local machine. Then, run the AWS SAM CLI from the parent folder that contains the template. You can also use the `--template-file` option with any AWS SAM CLI command and provide the path to your template.

Use Application Composer from the AWS Toolkit for Visual Studio Code

You can use Application Composer from the Toolkit for VS Code to bring Application Composer to your local machine. Then, use Application Composer and the AWS SAM CLI from VS Code.

Build your application

Building your application involves taking your AWS SAM template, AWS Lambda function code, and any language-specific files and dependencies, and placing these build artifacts in the proper structure and location for deployment. You can use the **sam build** command to build your application.

To learn more about building applications with AWS SAM, see the following from the *AWS Serverless Application Model Developer Guide*:

- Building serverless applications.
- Using sam build.

Deploy your application

Deploy your application to AWS CloudFormation to provision the resources and infrastructure defined in your AWS SAM templates. You can use the **sam deploy** command to deploy your application.

- You can deploy to create new resources or update existing resources.
- The AWS SAM CLI saves your deployment preferences in a configuration file.
- A deployed application in AWS CloudFormation is called a stack. To learn more, see Working with stacks in the *AWS CloudFormation User Guide*.

To learn more about deploying applications with AWS SAM, see the following from the *AWS Serverless Application Model Developer Guide*:

- Deploying serverless applications.
- Using sam deploy.

Test your application locally

Use the AWS SAM CLI to test your application locally. You can simulate events, start up APIs, invoke functions, and more.

Local testing requires Docker on your local machine. For more information, see Installing Docker to use with the AWS SAM CLI in the *AWS Serverless Application Model Developer Guide*.

To learn more about testing locally, see the following from the *AWS Serverless Application Model Developer Guide*:

- Testing and debugging serverless applications.
- Using sam local

Sync local changes to the cloud

As you design your application in Application Composer, you can use the **sam sync** command to have the AWS SAM CLI automatically detect local changes and deploy those changes to AWS CloudFormation.

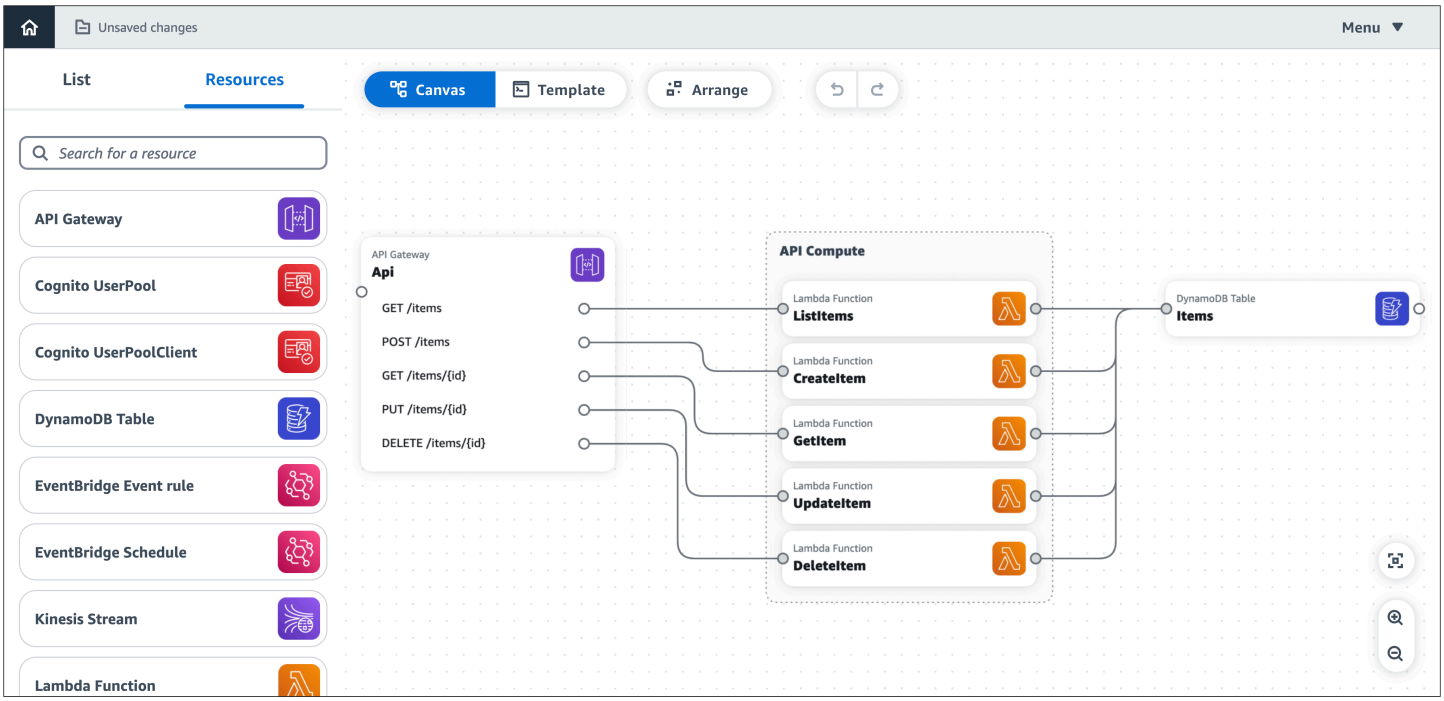
To learn more, see the following from the *AWS Serverless Application Model Developer Guide*:

- Using sam sync.

Examples

Build and deploy a serverless application

In this example, we build and deploy the Application Composer demo application.



- To learn more about the demo application, see [Tutorial 1: Load and modify the Application Composer demo project].
- For this example, we will be using Application Composer with **local sync** activated.

Use the **sam build** command to build the application.

```
$ sam build
...
Build Succeeded

Built Artifacts  : .aws-sam/build
Built Template   : .aws-sam/build/template.yaml

Commands you can use next
=========================
[*] Validate SAM template: sam validate
[*] Invoke Function: sam local invoke
[*] Test Function in the Cloud: sam sync --stack-name {{stack-name}} --watch
[*] Deploy: sam deploy --guided
```

The AWS SAM CLI creates the `./aws-sam` directory in the project folder. This directory contains build artifacts for the application's Lambda functions. Here is an output of the project directory:

```
.
### README.md
### samconfig.toml
### src
#   ### CreateItem
#   #   ### index.js
#   #   ### package.json
#   ### DeleteItem
#   #   ### index.js
#   #   ### package.json
#   ### GetItem
#   #   ### index.js
#   #   ### package.json
#   ### ListItems
#   #   ### index.js
#   #   ### package.json
#   ### UpdateItem
#       ### index.js
#       ### package.json
### template.yaml
```

Now, the application is ready to be deployed. We will use **sam deploy --guided**. This prepares your application for deployment through a series of prompts.

```
$ sam deploy --guided
...
Configuring SAM deploy
======================

    Looking for config file [samconfig.toml] :  Found
    Reading default arguments  :  Success

    Setting default arguments for 'sam deploy'
    =========================================
    Stack Name [aws-app-composer-basic-api]:
    AWS Region [us-west-2]:
    #Shows you resources changes to be deployed and require a 'Y' to initiate deploy
    Confirm changes before deploy [y/N]:
    #SAM needs permission to be able to create roles to connect to the resources in
 your template
    Allow SAM CLI IAM role creation [Y/n]:
    #Preserves the state of previously provisioned resources when an operation fails
    Disable rollback [y/N]:
```

```
    ListItems may not have authorization defined, Is this okay? [y/N]: y
    CreateItem may not have authorization defined, Is this okay? [y/N]: y
    GetItem may not have authorization defined, Is this okay? [y/N]: y
    UpdateItem may not have authorization defined, Is this okay? [y/N]: y
    DeleteItem may not have authorization defined, Is this okay? [y/N]: y
    Save arguments to configuration file [Y/n]:
    SAM configuration file [samconfig.toml]:
    SAM configuration environment [default]:
```

The AWS SAM CLI displays a summary of what will be deployed:

```
Deploying with following values
    ===============================
    Stack name                   : aws-app-composer-basic-api
    Region                       : us-west-2
    Confirm changeset            : False
    Disable rollback             : False
    Deployment s3 bucket         : aws-sam-cli-managed-default-
samclisourcebucket-1b3x26zbcdkqr
    Capabilities                 : ["CAPABILITY_IAM"]
    Parameter overrides          : {}
    Signing Profiles             : {}
```

The AWS SAM CLI deploys the application, first by creating an AWS CloudFormation changeset:

```
Initiating deployment
=====================
Uploading to aws-app-composer-basic-api/4181c909ee2440a728a7a129dafb83d4.template
 7087 / 7087  (100.00%)

Waiting for changeset to be created..
CloudFormation stack changeset
-------------------------------------------------------------------------------------------------
Operation                           LogicalResourceId                   ResourceType
                     Replacement
-------------------------------------------------------------------------------------------------
+ Add                               ApiDeploymentcc153d135b
 AWS::ApiGateway::Deployment         N/A
+ Add                               ApiProdStage
 AWS::ApiGateway::Stage              N/A
+ Add                               Api
 AWS::ApiGateway::RestApi            N/A
```

```
+ Add                                 CreateItemApiPOSTitemsPermissionP
 AWS::Lambda::Permission               N/A
                                      rod
+ Add                                 CreateItemRole                      AWS::IAM::Role
                      N/A
+ Add                                 CreateItem
 AWS::Lambda::Function                 N/A
+ Add                                 DeleteItemApiDELETEitemsidPermiss
 AWS::Lambda::Permission               N/A
                                      ionProd
+ Add                                 DeleteItemRole                      AWS::IAM::Role
                      N/A
+ Add                                 DeleteItem
 AWS::Lambda::Function                 N/A
+ Add                                 GetItemApiGETitemsidPermissionPro
 AWS::Lambda::Permission               N/A
                                      d
+ Add                                 GetItemRole                         AWS::IAM::Role
                      N/A
+ Add                                 GetItem
 AWS::Lambda::Function                 N/A
+ Add                                 Items
 AWS::DynamoDB::Table                  N/A
+ Add                                 ListItemsApiGETitemsPermissionPro
 AWS::Lambda::Permission               N/A
                                      d
+ Add                                 ListItemsRole                       AWS::IAM::Role
                      N/A
+ Add                                 ListItems
 AWS::Lambda::Function                 N/A
+ Add                                 UpdateItemApiPUTitemsidPermission
 AWS::Lambda::Permission               N/A
                                      Prod
+ Add                                 UpdateItemRole                      AWS::IAM::Role
                      N/A
+ Add                                 UpdateItem
 AWS::Lambda::Function                 N/A
-----------------------------------------------------------------------------------------------------

Changeset created successfully. arn:aws:cloudformation:us-
west-2:513423067560:changeSet/samcli-deploy1677472539/967ab543-f916-4170-b97d-
c11a6f9308ea
```

Then, the AWS SAM CLI deploys the application:

```
CloudFormation events from stack operations (refresh every 0.5 seconds)
---------------------------------------------------------------------------------------------------
ResourceStatus                      ResourceType
 LogicalResourceId                   ResourceStatusReason
---------------------------------------------------------------------------------------------------
CREATE_IN_PROGRESS                  AWS::DynamoDB::Table               Items
                    -
CREATE_IN_PROGRESS                  AWS::DynamoDB::Table               Items
                    Resource creation Initiated
CREATE_COMPLETE                     AWS::DynamoDB::Table               Items
                    -
CREATE_IN_PROGRESS                  AWS::IAM::Role                     DeleteItemRole
                    -
CREATE_IN_PROGRESS                  AWS::IAM::Role                     ListItemsRole
                    -
CREATE_IN_PROGRESS                  AWS::IAM::Role                     UpdateItemRole
                    -
CREATE_IN_PROGRESS                  AWS::IAM::Role                     GetItemRole
                    -
CREATE_IN_PROGRESS                  AWS::IAM::Role                     CreateItemRole
                    -
CREATE_IN_PROGRESS                  AWS::IAM::Role                     DeleteItemRole
                    Resource creation Initiated
CREATE_IN_PROGRESS                  AWS::IAM::Role                     ListItemsRole
                    Resource creation Initiated
CREATE_IN_PROGRESS                  AWS::IAM::Role                     GetItemRole
                    Resource creation Initiated
CREATE_IN_PROGRESS                  AWS::IAM::Role                     UpdateItemRole
                    Resource creation Initiated
CREATE_IN_PROGRESS                  AWS::IAM::Role                     CreateItemRole
                    Resource creation Initiated
CREATE_COMPLETE                     AWS::IAM::Role                     DeleteItemRole
                    -
CREATE_COMPLETE                     AWS::IAM::Role                     ListItemsRole
                    -
CREATE_COMPLETE                     AWS::IAM::Role                     GetItemRole
                    -
CREATE_COMPLETE                     AWS::IAM::Role                     UpdateItemRole
                    -
CREATE_COMPLETE                     AWS::IAM::Role                     CreateItemRole
                    -
```

```
CREATE_IN_PROGRESS                  AWS::Lambda::Function               DeleteItem
                    -
CREATE_IN_PROGRESS                  AWS::Lambda::Function               CreateItem
                    -
CREATE_IN_PROGRESS                  AWS::Lambda::Function               ListItems
                    -
CREATE_IN_PROGRESS                  AWS::Lambda::Function               UpdateItem
                    -
CREATE_IN_PROGRESS                  AWS::Lambda::Function               DeleteItem
                    Resource creation Initiated
CREATE_IN_PROGRESS                  AWS::Lambda::Function               GetItem
                    -
CREATE_IN_PROGRESS                  AWS::Lambda::Function               ListItems
                    Resource creation Initiated
CREATE_IN_PROGRESS                  AWS::Lambda::Function               CreateItem
                    Resource creation Initiated
CREATE_IN_PROGRESS                  AWS::Lambda::Function               UpdateItem
                    Resource creation Initiated
CREATE_IN_PROGRESS                  AWS::Lambda::Function               GetItem
                    Resource creation Initiated
CREATE_COMPLETE                     AWS::Lambda::Function               DeleteItem
                    -
CREATE_COMPLETE                     AWS::Lambda::Function               ListItems
                    -
CREATE_COMPLETE                     AWS::Lambda::Function               CreateItem
                    -
CREATE_COMPLETE                     AWS::Lambda::Function               UpdateItem
                    -
CREATE_COMPLETE                     AWS::Lambda::Function               GetItem
                    -
CREATE_IN_PROGRESS                  AWS::ApiGateway::RestApi            Api
                    -
CREATE_IN_PROGRESS                  AWS::ApiGateway::RestApi            Api
                    Resource creation Initiated
CREATE_COMPLETE                     AWS::ApiGateway::RestApi            Api
                    -
CREATE_IN_PROGRESS                  AWS::Lambda::Permission
 GetItemApiGETitemsidPermissionPro   -
                                                                        d
CREATE_IN_PROGRESS                  AWS::Lambda::Permission
 ListItemsApiGETitemsPermissionPro   -
                                                                        d
CREATE_IN_PROGRESS                  AWS::Lambda::Permission
 DeleteItemApiDELETEitemsidPermiss   -
```

```
                                                                    ionProd
CREATE_IN_PROGRESS                  AWS::ApiGateway::Deployment
 ApiDeploymentcc153d135b             -
CREATE_IN_PROGRESS                  AWS::Lambda::Permission
 UpdateItemApiPUTitemsidPermission   -
                                                                    Prod
CREATE_IN_PROGRESS                  AWS::Lambda::Permission
 CreateItemApiPOSTitemsPermissionP   -
                                                                    rod
CREATE_IN_PROGRESS                  AWS::Lambda::Permission
 GetItemApiGETitemsidPermissionPro   Resource creation Initiated
                                                                    d
CREATE_IN_PROGRESS                  AWS::Lambda::Permission
 UpdateItemApiPUTitemsidPermission   Resource creation Initiated
                                                                    Prod
CREATE_IN_PROGRESS                  AWS::Lambda::Permission
 CreateItemApiPOSTitemsPermissionP   Resource creation Initiated
                                                                    rod
CREATE_IN_PROGRESS                  AWS::Lambda::Permission
 ListItemsApiGETitemsPermissionPro   Resource creation Initiated
                                                                    d
CREATE_IN_PROGRESS                  AWS::Lambda::Permission
 DeleteItemApiDELETEitemsidPermiss   Resource creation Initiated
                                                                    ionProd
CREATE_IN_PROGRESS                  AWS::ApiGateway::Deployment
 ApiDeploymentcc153d135b             Resource creation Initiated
CREATE_COMPLETE                     AWS::ApiGateway::Deployment
 ApiDeploymentcc153d135b             -
CREATE_IN_PROGRESS                  AWS::ApiGateway::Stage              ApiProdStage
                    -
CREATE_IN_PROGRESS                  AWS::ApiGateway::Stage              ApiProdStage
                    Resource creation Initiated
CREATE_COMPLETE                     AWS::ApiGateway::Stage              ApiProdStage
                    -
CREATE_COMPLETE                     AWS::Lambda::Permission
 CreateItemApiPOSTitemsPermissionP   -
                                                                    rod
CREATE_COMPLETE                     AWS::Lambda::Permission
 UpdateItemApiPUTitemsidPermission   -
                                                                    Prod
CREATE_COMPLETE                     AWS::Lambda::Permission
 ListItemsApiGETitemsPermissionPro   -
                                                                    d
```

```
CREATE_COMPLETE                     AWS::Lambda::Permission
 DeleteItemApiDELETEitemsidPermiss   -
                                                                    ionProd
CREATE_COMPLETE                     AWS::Lambda::Permission
 GetItemApiGETitemsidPermissionPro   -
                                                                    d
CREATE_COMPLETE                     AWS::CloudFormation::Stack      aws-app-
composer-basic-api          -
-------------------------------------------------------------------------------------------
```

Finally, a message is displayed, informing you that deployment was successful:

```
Successfully created/updated stack - aws-app-composer-basic-api in us-west-2
```

Delete an AWS CloudFormation stack

To delete an AWS CloudFormation stack, use the **sam delete** command:

```
$ sam delete
 Are you sure you want to delete the stack aws-app-composer-basic-api in the region us-
west-2 ? [y/N]: y
 Do you want to delete the template file 30439348c0be6e1b85043b7a935b34ab.template in
 S3? [y/N]: y
 - Deleting S3 object with key eb226ca86d1bc4e9914ad85eb485fed8
 - Deleting S3 object with key 875e4bcf4b10a6a1144ad83158d84b6d
 - Deleting S3 object with key 20b869d98d61746dedd9aa33aa08a6fb
 - Deleting S3 object with key c513cedc4db6bc184ce30e94602741d6
 - Deleting S3 object with key c7a15d7d8d1c24b77a1eddf8caebc665
 - Deleting S3 object with key e8b8984f881c3732bfb34257cdd58f1e
 - Deleting S3 object with key 3185c59b550594ee7fca7f8c36686119.template
 - Deleting S3 object with key 30439348c0be6e1b85043b7a935b34ab.template
 - Deleting Cloudformation stack aws-app-composer-basic-api

Deleted successfully
```

Using AWS Application Composer with the AWS Lambda console

AWS Application Composer from the AWS Management Console provides an integration with the AWS Lambda console. You can import a Lambda function from the Lambda console into

Application Composer. Then, use the Application Composer canvas to design your application architecture further.

- This integration requires a browser that supports the File System Access API. For more information, see AWS Application Composer and the File System Access API.
- When you import your Lambda function into Application Composer, you must activate local sync mode to save any changes. For more information, see Automatically sync and save your project.

To get started with using this integration, see Using AWS Lambda with AWS Application Composer in the *AWS Lambda Developer Guide*.

Application Composer reference

This section contains reference information for AWS Application Composer.

Topics

- AWS Application Composer and the File System Access API
- Application Composer card reference
- AWS Application Composer troubleshooting
- Serverless concepts

AWS Application Composer and the File System Access API

To use the AWS Application Composer **local sync** mode, a web browser that supports the File System Access API is required.

Topics

- What is the File System Access API?
- What is the local sync mode?
- What web browsers are supported?
- What does Application Composer gain access to?

What is the File System Access API?

The File System Access API lets web pages gain access to your local file system in order to read, write, or save files. This feature is off by default and requires your permission through a visual prompt to allow it. Once granted, this access remains for the duration of your web page's browser session.

To learn more about the File System Access API, see:

- File System Access API in the *mdn web docs*.
- The File System Access API: simplifying access to local files in the *web.dev* website.

What is the local sync mode?

Local sync mode lets you automatically sync and save your template files and project folders locally as you design in Application Composer. To use this feature, a web browser that supports the File System Access API is required.

What web browsers are supported?

Any recent version of Google Chrome and Microsoft Edge support all capabilities of the File System Access API and can be used with **local sync** mode in Application Composer.

What does Application Composer gain access to?

Application Composer gains read and write access to the project folder you allow, along with any child folders of that project folder. This access is used to create, update, and save any template files, project folders, and backup directories that are generated as you design. Data accessed by Application Composer is not used for any other purpose and is not stored anywhere beyond your local file system.

Access to sensitive data

The File System Access API excludes or limits access to specific directories that may contain sensitive data. An error will occur if you select one of these directories to use with Application Composer **local sync** mode. You can choose another local directory to connect with or use Application Composer in its default mode with **local sync** deactivated.

For more information, including examples of sensitive directories, see Users giving access to more, or more sensitive files than they intended in the *File System Access W3C Draft Community Group Report*.

If you use Windows Subsystem for Linux (WSL), the File System Access API excludes access to the entire Linux directory because of its location within your Windows system. You can use Application Composer with **local sync** deactivated or configure a solution to sync project files from your WSL directory to a working directory in Windows. Then, use Application Composer **local sync** mode with your Windows directory.

Application Composer card reference

This topic contains reference information for cards in AWS Application Composer. To learn about using cards, see Designing with cards in Application Composer.

Topics

- Enhanced component cards
- Future enhanced component card support

Enhanced component cards

Enhanced component cards are those available from the **Resources** palette. They can be fully configured and used within Application Composer to design and build your serverless applications. We recommend using enhanced component cards when designing your applications from scratch.

This table displays our enhanced components with links to the AWS CloudFormation or AWS Serverless Application Model (AWS SAM) template specification of the card's featured resource:

Card	Reference
Amazon API Gateway	AWS::Serverless::API
Amazon Cognito UserPool	AWS::Cognito::UserPool
Amazon Cognito UserPoolClient	AWS::Cognito::UserPoolClient
Amazon DynamoDB Table	AWS::DynamoDB::Table
Amazon EventBridge Event rule	AWS::Events::Rule
EventBridge Schedule	AWS::Scheduler::Schedule
Amazon Kinesis Stream	AWS::Kinesis::Stream
AWS Lambda Function	AWS::Serverless::Function
Lambda Layer	AWS::Serverless::LayerVersion
Amazon Simple Storage Service (Amazon S3) Bucket	AWS::S3::Bucket
Amazon Simple Notification Service (Amazon SNS) Topic	AWS::SNS::Topic

Card	Reference
Amazon Simple Queue Service (Amazon SQS) Queue	AWS::SQS::Queue
AWS Step Functions State machine	AWS::Serverless::StateMachine

Future enhanced component card support

When prioritizing enhanced component cards to feature, we consider those that are popular in usage, powerful in combination with others, and challenging to configure. This is where Application Composer's visual features are most beneficial.

To provide feedback on enhanced component cards you'd like to see featured, please contact us through the feedback link located at the bottom-left corner of Application Composer.

AWS Application Composer troubleshooting

Troubleshoot error messages when using AWS Application Composer.

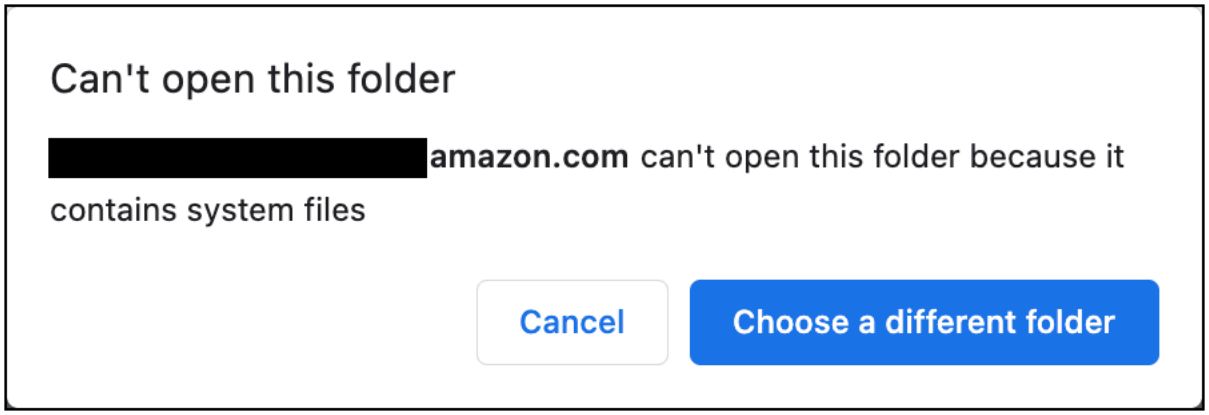
Topics

- Error messages
- Submit feedback

Error messages

"Can't open this folder"

Example error:



Possible cause: Application Composer is unable to access a sensitive directory using local sync mode.

To learn more about this error, see What does Application Composer gain access to?.

Try connecting to a different local directory or using Application Composer with **local sync** deactivated.

"Incompatible template"

Example error: When loading a new project in Application Composer, you see the following:

Possible cause: Your project contains an externally referenced file that isn't supported in Application Composer.

To learn about supported external files in Application Composer, see Referencing external files.

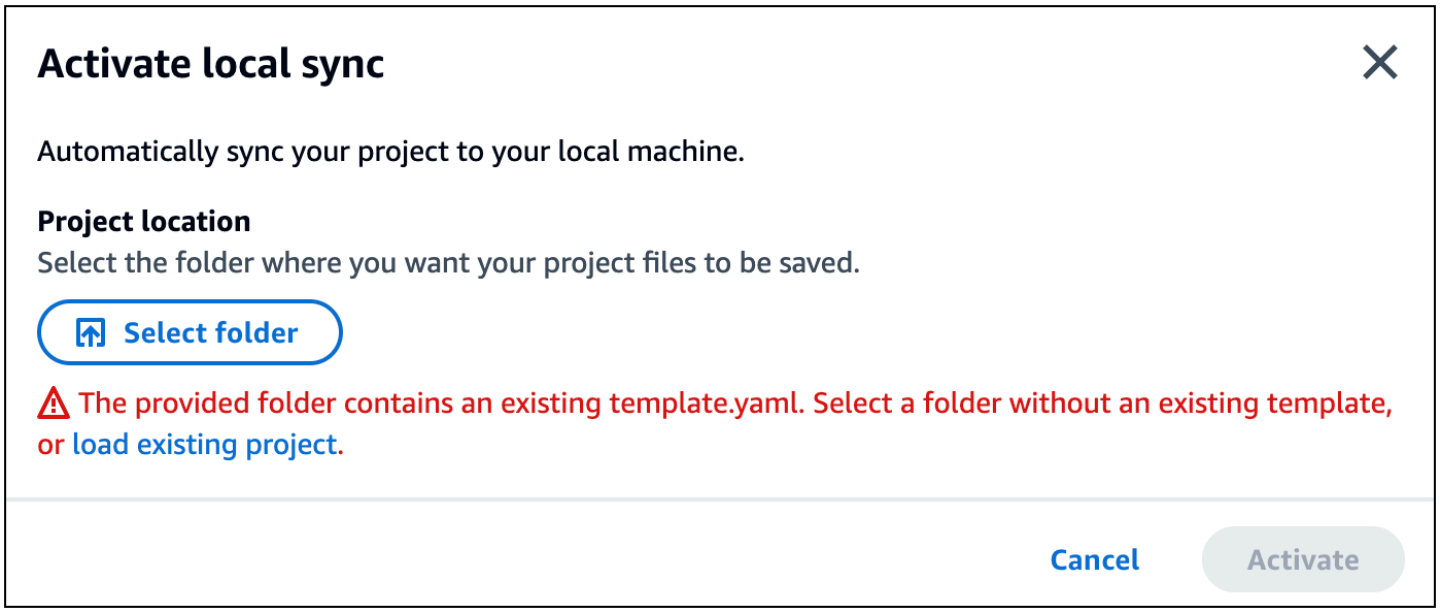
If you'd like Application Composer to support your use case, submit feedback.

Possible cause: Your project links to an external file in a different local directory.

Move your externally referenced file to a subdirectory of the directory that you select to use with Application Composer **local sync** mode.

"The provided folder contains an existing template.yaml"

When attempting to activate **local sync**, you see the following error:



Possible cause: Your selected folder already contains a template.yaml file.

Select another directory that doesn't contain an application template, or create a new directory.

"Your browser doesn't have permissions to save your project in that folder..."

Possible cause: Application Composer is unable to access a sensitive directory using local sync mode.

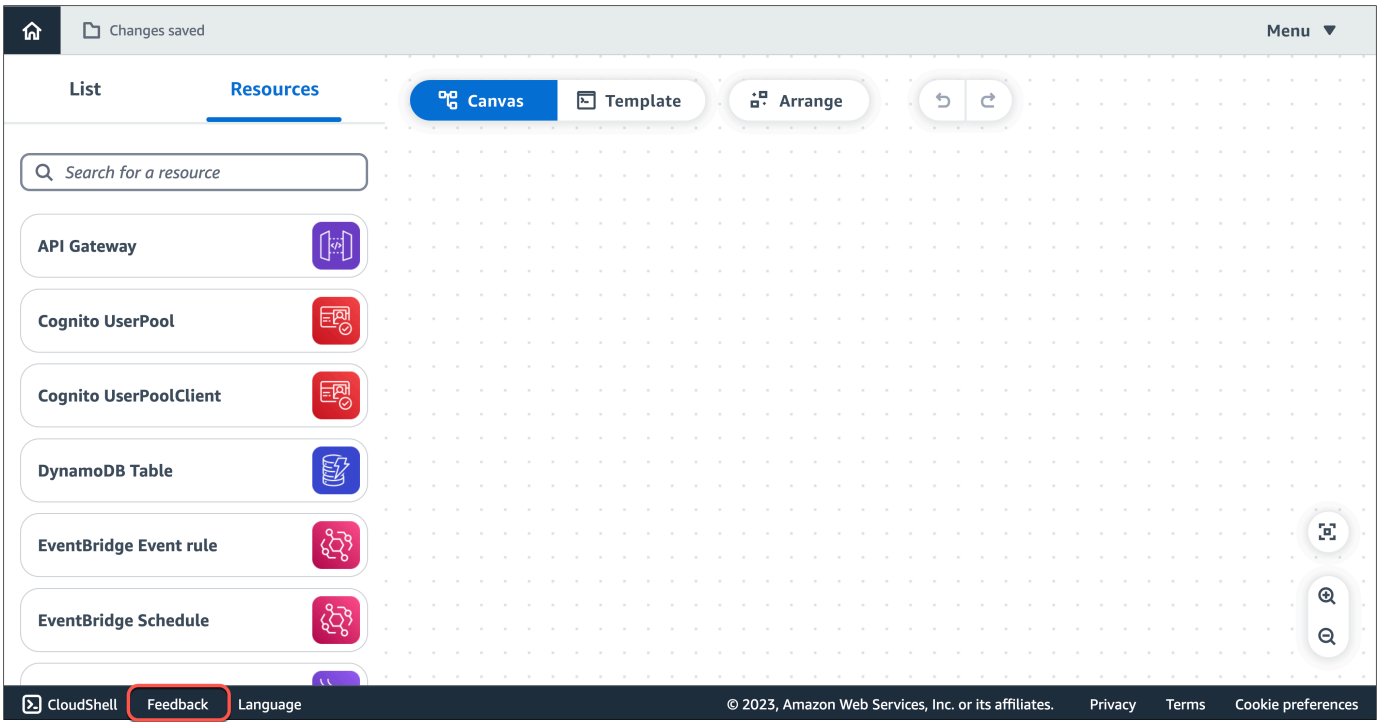
To learn more about this error, see What does Application Composer gain access to?.

Try connecting to a different local directory or use Application Composer with **local sync** deactivated.

Submit feedback

To submit feedback in Application Composer

1. Select the **Feedback** link within Application Composer.
2. Fill out the feedback form and **Submit**.



Serverless concepts

Learn about basic serverless concepts before using AWS Application Composer.

Serverless concepts

Event-driven architecture

A serverless application consists of individual AWS services, such as AWS Lambda for compute and Amazon DynamoDB for database management, that each perform a specialized role. These services are then loosely integrated with each other through an event-driven architecture. To learn more about event-driven architecture, see What is an Event-Driven Architecture?.

Infrastructure as Code (IaC)

Infrastructure as Code (IaC) is a way of treating infrastructure in the same way that developers treat code, applying the same rigor of application code development to infrastructure provisioning. You define your infrastructure in a template file, deploy it to AWS, and AWS creates the resources for you. With IAC, you define in code what you want AWS to provision. For more information, see Infrastructure as Code in the *Introduction to DevOps on AWS* AWS Whitepaper.

Serverless technologies

With AWS serverless technologies, you can build and run applications without having to manage your own servers. All server management is done by AWS, providing many benefits such as automatic scaling and built-in high availability, letting you take your idea to production quickly. Using serverless technologies, you can focus on the core of your product without having to worry about managing and operating servers. To learn more about serverless, see Serverless on AWS.

For a basic introduction to the core AWS serverless services, see Serverless 101: Understanding the serverless services at *Serverless Land*.

Security in AWS Application Composer

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

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

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

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

Topics

- Data protection in AWS Application Composer
- Identity and access management for AWS Application Composer
- Compliance validation for AWS Application Composer
- Resilience in AWS Application Composer

Data protection in AWS Application Composer

The AWS shared responsibility model applies to data protection in AWS Application Composer. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks

for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

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

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail.
- 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 Application Composer 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.

> **Note**
>
> All data that you input into Application Composer is used for the sole purpose of providing functionality within Application Composer and generating project files and directories that are saved locally to your machine. Application Composer does not save, store or transmit any of this data.

Data encryption

Application Composer does not encrypt customer content since data is not saved, stored or transmitted.

Encryption at rest

Application Composer does not encrypt customer content since data is not saved, stored or transmitted.

Encryption in transit

Application Composer does not encrypt customer content since data is not saved, stored or transmitted.

Key management

Application Composer does not support key management since customer content is not saved, stored or transmitted.

Inter-network traffic privacy

Application Composer does not generate traffic with on-premise clients and applications.

Identity and access management for AWS Application Composer

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Application Composer resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience
- Authenticating with identities
- Managing access using policies
- How AWS Application Composer works with IAM

Audience

Application Composer requires, at minimum, read-only access to the AWS Management Console. Any user with this authorization can use all features of Application Composer. Granular access to specific features of Application Composer is not supported.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see Signing AWS API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account.

We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must

have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

- **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How AWS Application Composer works with IAM

AWS Application Composer requires, at minimum, read-only access to the AWS Management Console. Any user with this authorization can use all features of Application Composer. Granular access to specific features of Application Composer is not supported.

When you deploy your project template and files to AWS CloudFormation, you will need the necessary permissions to be in place. To learn more, see Controlling access with AWS Identity and Access Management in the *AWS CloudFormation User Guide*.

IAM features you can use with AWS Application Composer

IAM feature	Application Composer support
Identity-based policies	No
Resource-based policies	No
Policy actions	No
Policy resources	No
Policy condition keys	No
ACLs	No
ABAC (tags in policies)	No
Temporary credentials	Yes
Principal permissions	No
Service roles	No
Service-linked roles	No

To get a high-level view of how Application Composer and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for Application Composer

Supports identity-based policies	No

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Resource-based policies within Application Composer

Supports resource-based policies	No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Policy actions for Application Composer

Supports policy actions	No

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Application Composer actions, see Actions Defined by AWS Application Composer in the *Service Authorization Reference*.

Policy actions in Application Composer use the following prefix before the action:

```
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      ":action1",
      ":action2"
         ]
```

Policy resources for Application Composer

Supports policy resources	No

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of Application Composer resource types and their ARNs, see Resources Defined by AWS Application Composer in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by AWS Application Composer .

Policy condition keys for Application Composer

Supports service-specific policy condition keys	No

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of Application Composer condition keys, see Condition Keys for AWS Application Composer in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions Defined by AWS Application Composer .

ACLs in Application Composer

Supports ACLs	No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with Application Composer

Supports ABAC (tags in policies)	No

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see What is ABAC? in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with Application Composer

Supports temporary credentials	Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switching to a role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

You can use temporary credentials to access Application Composer through the AWS Management Console. For an example, see Enabling custom identity broker access to the AWS console in the *IAM User Guide*.

Cross-service principal permissions for Application Composer

Supports forward access sessions (FAS)	No

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service roles for Application Composer

Supports service roles	No

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break Application Composer functionality. Edit service roles only when Application Composer provides guidance to do so.

Service-linked roles for Application Composer

Supports service-linked roles	No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Compliance validation for AWS Application Composer

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- Architecting for HIPAA Security and Compliance on Amazon Web Services – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

> **Note**
>
> Not all AWS services are HIPAA eligible. For more information, see the HIPAA Eligible Services Reference.

- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Resilience in AWS Application Composer

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

All data that you input into Application Composer is used for the sole purpose of providing functionality within Application Composer and generating project files and directories that are saved locally to your machine. Application Composer does not save or store any of this data.

Document history for Application Composer

The following table describes important documentation releases for Application Composer. For notifications about updates to this documentation, you can subscribe to an RSS feed.

- **Latest documentation update:** November 30, 2023

Change	Description	Date
Add documentation for the Application Composer integration with CodeWhisperer	AWS Application Composer from the Toolkit for VS Code provides an integration with Amazon CodeWhisperer. To learn more, see Using AWS Application Composer with Amazon CodeWhisperer.	November 30, 2023
Add documentation for deploying your application with Application Composer from the AWS Toolkit for Visual Studio Code	Use the **sync** button from the Application Composer canvas to deploy your applicati on to the AWS Cloud. To learn more, see Deploy your application with sam sync.	November 30, 2023
Add documentation for Application Composer from the AWS Toolkit for Visual Studio Code	You can now use Application Composer from VS Code with the AWS Toolkit for Visual Studio Code. To learn more, see Using AWS Applicati on Composer from the AWS Toolkit for Visual Studio Code.	November 30, 2023
Add Step Functions Workflow Studio integration	Launch Step Functions Workflow Studio from the Application Composer canvas. To learn more, see Using AWS	November 27, 2023

	Application Composer with AWS Step Functions.	
Add Lambda console and Application Composer integration	Launch the Application Composer canvas from the Lambda console. To learn more, see Using AWS Application Composer with the AWS Lambda console.	November 14, 2023
Add Amazon VPC as a featured service with Application Composer	Application Composer introduces a VPC tag to visualize resources configure d with a VPC. You can also configure Lambda functions with VPCs defined on an external template. To learn more, see Using Application Composer with Amazon VPC.	October 17, 2023
Add Amazon RDS as a featured service with Application Composer	Connect your Application Composer application to an Amazon RDS DB cluster or instance that is defined on an external template. To learn more, see Using Application Composer with Amazon RDS.	October 17, 2023
Add Application Composer support to design with all AWS CloudFormation resources	Select any AWS CloudForm ation resource from the **Resources** palette to design your applications with. To learn more, see Work with any AWS CloudFormation resource.	September 26, 2023

Add documentation for cards in Application Composer	Application Composer supports multiple types of cards that you can use to design and build your application. To learn more, see Designing with cards in Application Composer.	September 20, 2023
Add documentation for undo and redo feature	Use the **undo** and **redo** buttons on the Application Composer canvas. To learn more, see Undo and redo.	August 1, 2023
Add documentation for local sync mode	Use **local sync** mode to automatically sync and save your project to your local machine. To learn more, see Local sync mode.	August 1, 2023
Add documentation for export canvas feature	Use the **export canvas** feature to export your application's canvas as an image to your local machine. To learn more, see Export canvas.	August 1, 2023
Application Composer support for external file references	Reference external files for supported resources in Application Composer. To learn more, see Working with templates that reference external files.	May 17, 2023

New documentation on connecting resources	Connect resources together to define event-driven relations hips between resources in your application. To learn more, see Connecting resources together using the Application Composer visual canvas.	March 7, 2023
New Change Inspector feature	Use the **Change Inspector** to view your template code updates and learn what Application Composer is creating for you. To learn more, see View code updates with the Change Inspector.	March 7, 2023
Expand on benefits of using connected mode	Use Application Composer in connected mode with your local IDE to speed up development. To learn more, see Using Application Composer with your local IDE.	March 7, 2023
Application Composer now generally available	AWS Application Composer is now generally available . To learn more, see AWS Application Composer now generally available - Visually build serverless applications quickly.	March 7, 2023

Update topic on using other AWS services to deploy your application	Use Application Composer to design deployment-ready serverless applications. Use AWS SAM to deploy your serverless application. To learn more, see Using Application Composer with AWS CloudFormation and AWS SAM.	March 3, 2023
Add serverless concepts section	Learn about basic serverless concepts before using Application Composer. To learn more, see Serverless concepts.	March 2, 2023
Public release	Initial public release of Application Composer.	December 1, 2022