aws

Developer Guide

AWS Infrastructure Composer

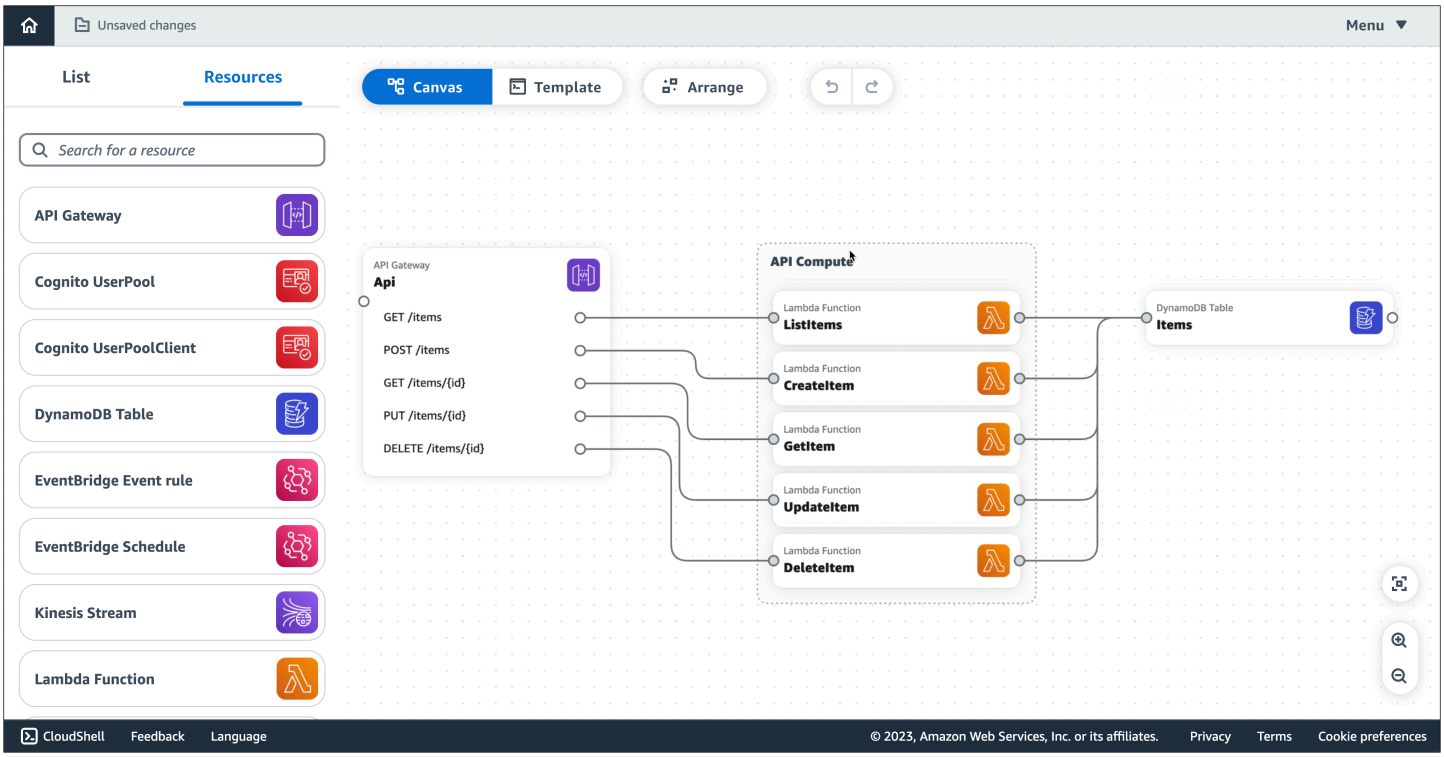
AWS Infrastructure Composer: Developer Guide

Table of Contents

What is AWS Infrastructure Composer?

AWS Infrastructure Composer allows you to visually compose modern applications on AWS. More specifically, you can use Infrastructure Composer to visualize, build, and deploy modern applications from all AWS services that are supported by AWS CloudFormation without needing to be an expert in CloudFormation.

As you compose your AWS CloudFormation infrastructure, through a delightful drag-and-drop interface, Infrastructure Composer creates your infrastructure as code (IaC) templates, all while following AWS best practices. The following image shows how easy it is to drag, drop, configure, and connect resources on Infrastructure Composer's visual canvas.



Infrastructure Composer can be used from the Infrastructure Composer console, the AWS Toolkit for Visual Studio Code, and in CloudFormation console mode.

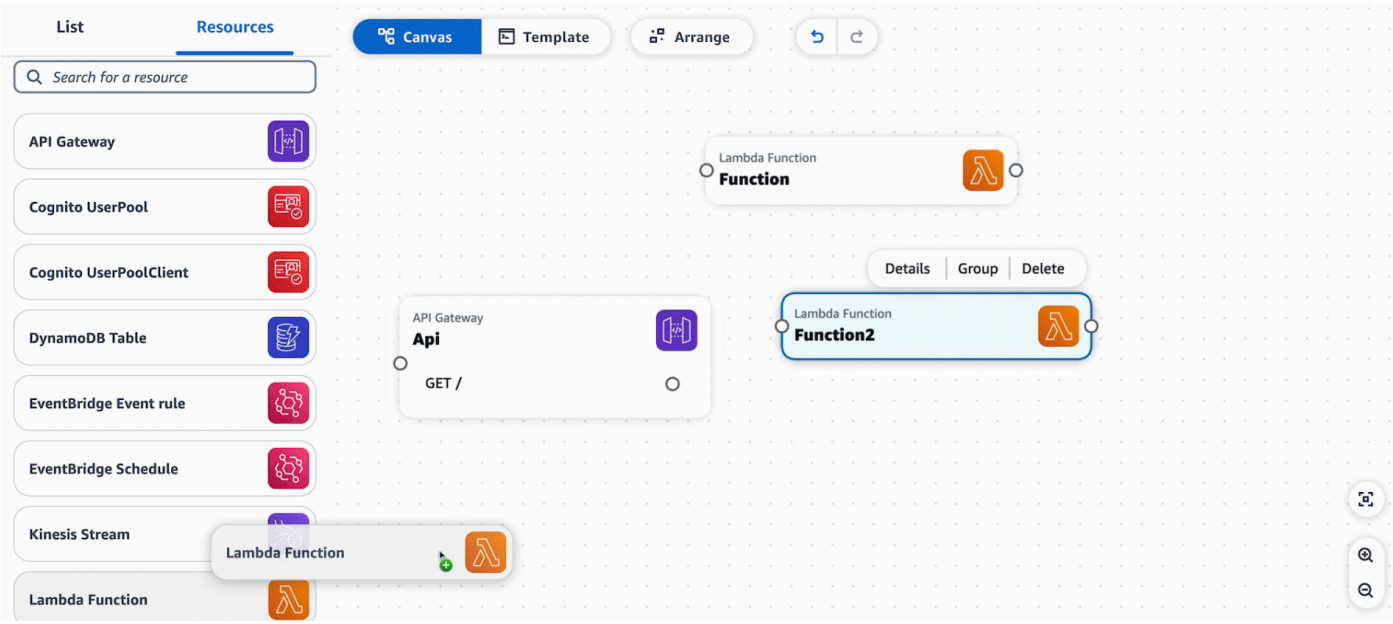
Topics

- Learn more
- Next steps
- Serverless concepts for AWS Infrastructure Composer

Compose your application architecture

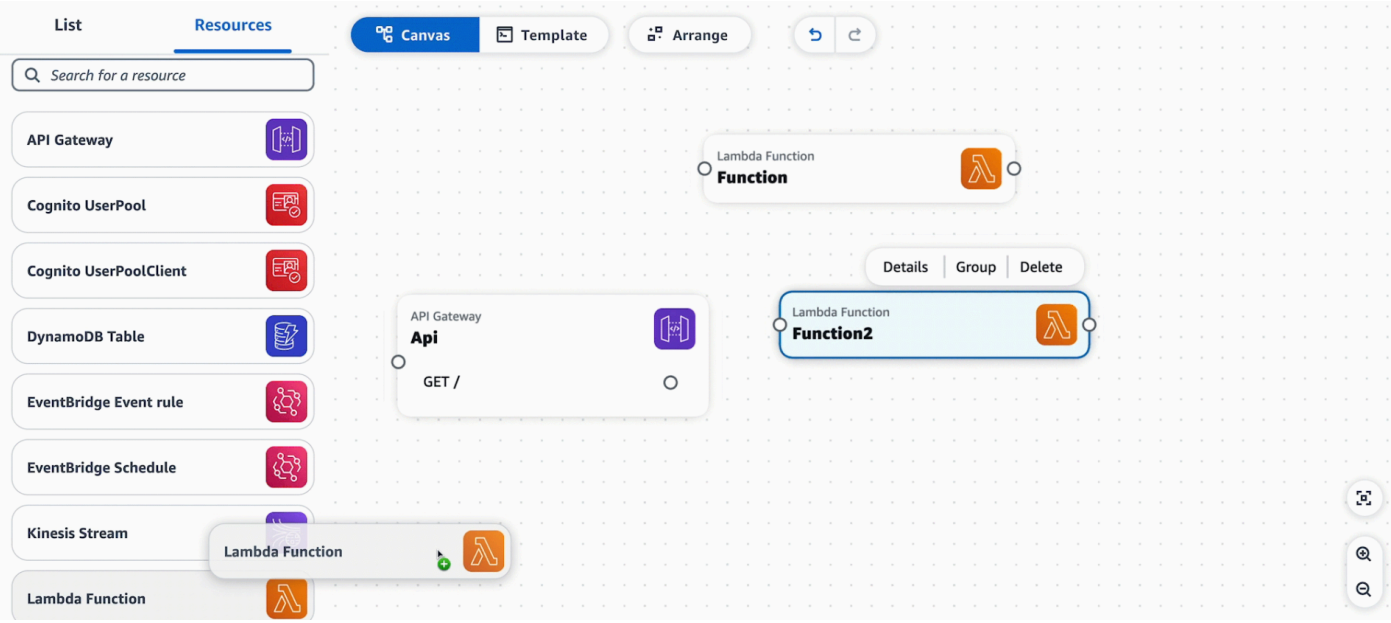
Build with cards

Place cards on the Infrastructure Composer canvas to visualize and build your application architecture.



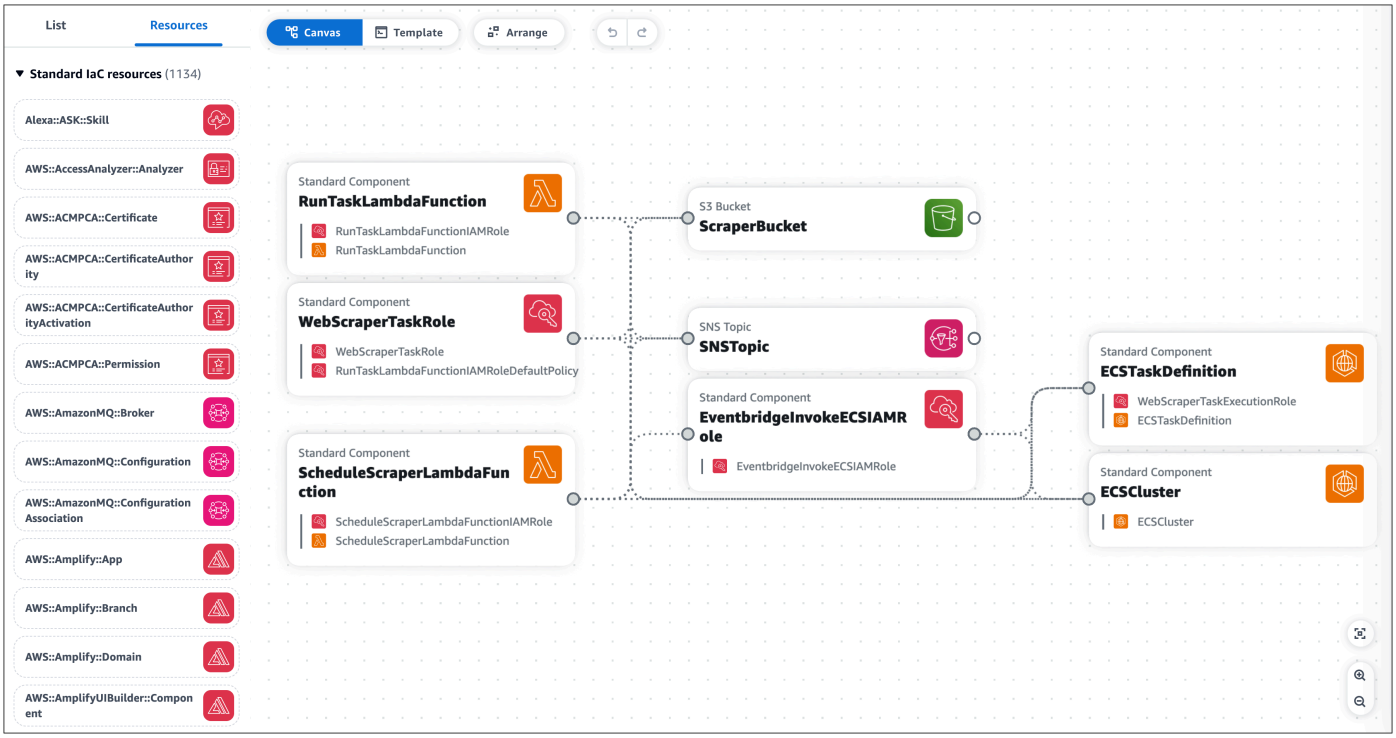
Connect cards together

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

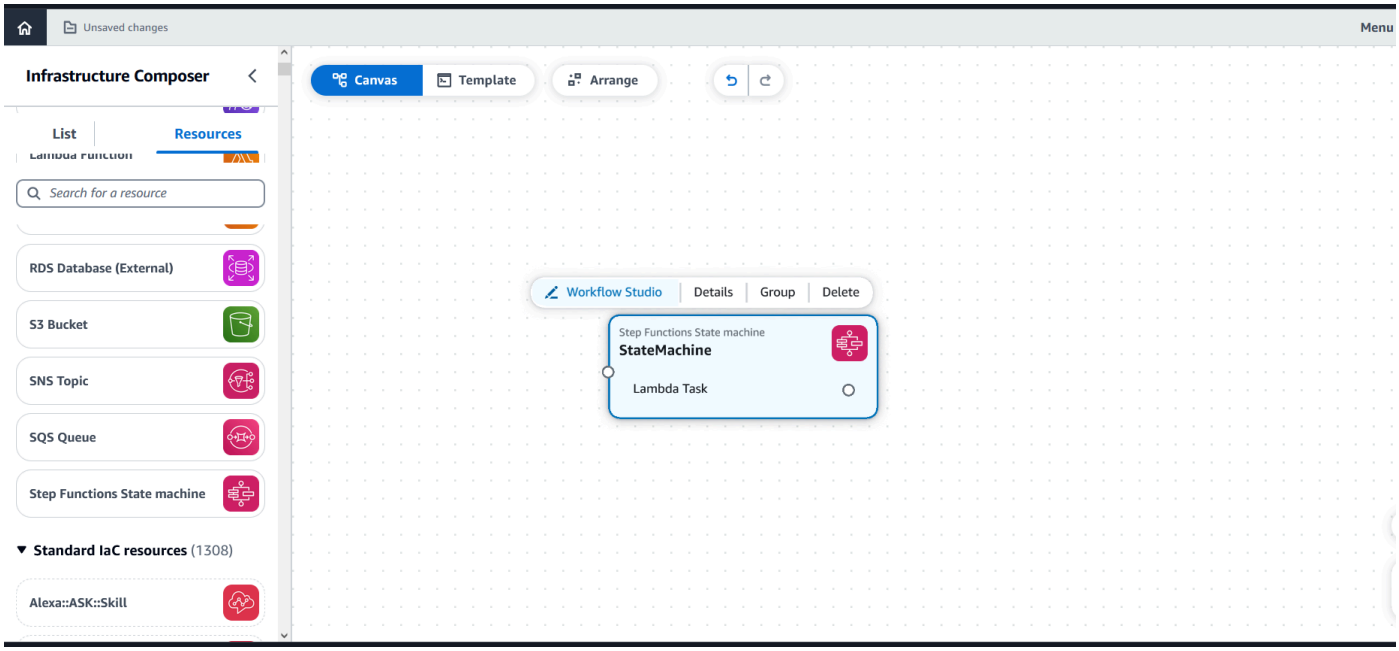
Drag any CloudFormation resource onto the canvas to compose your application architecture. Infrastructure Composer provides a starting IaC template that you can use to specify the properties of your resource. To learn more, see [Configure and modify cards in Infrastructure Composer](#).



Access additional capabilities with featured AWS services

Infrastructure Composer features AWS services that are commonly used or configured together when building applications. To learn more, see Integrate with Amazon VPC.

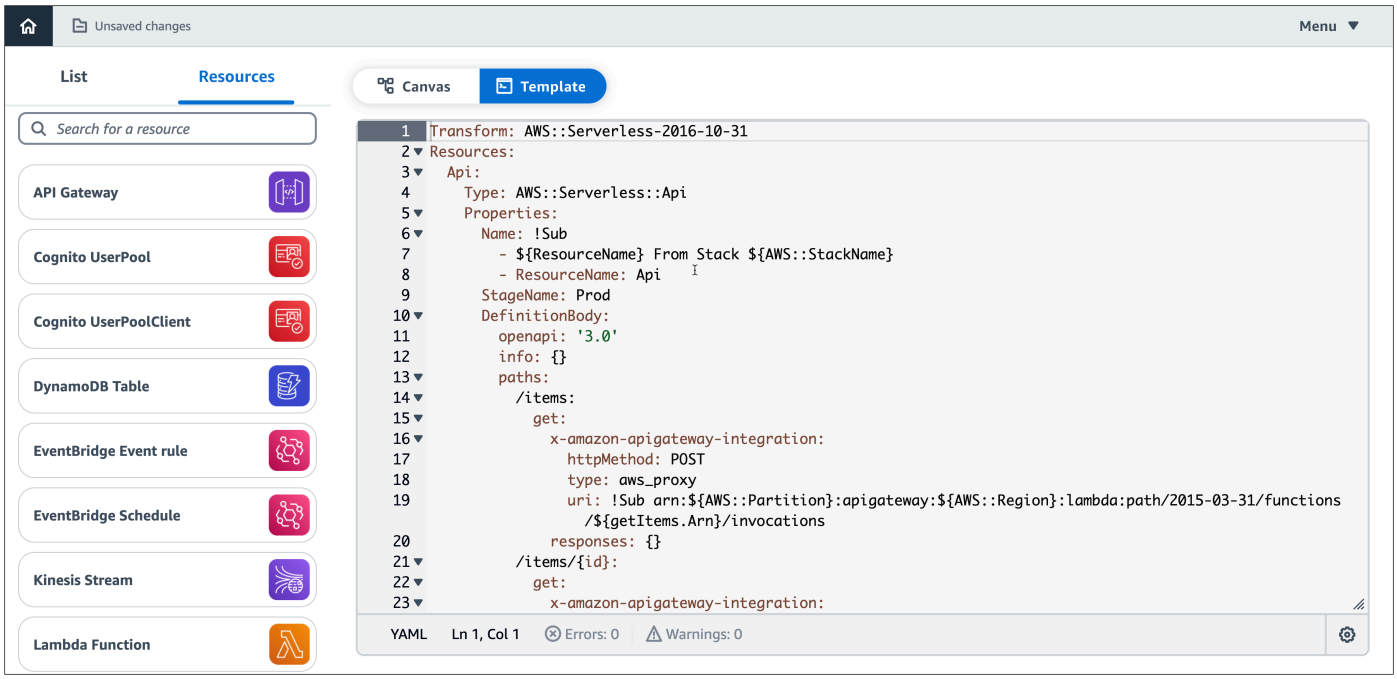
The following is an example of the AWS Step Functions feature, which provides an integration to launch Step Functions Workflow Studio directly within the Infrastructure Composer canvas.



Define your infrastructure as code (IaC) templates

Infrastructure Composer creates your infrastructure code

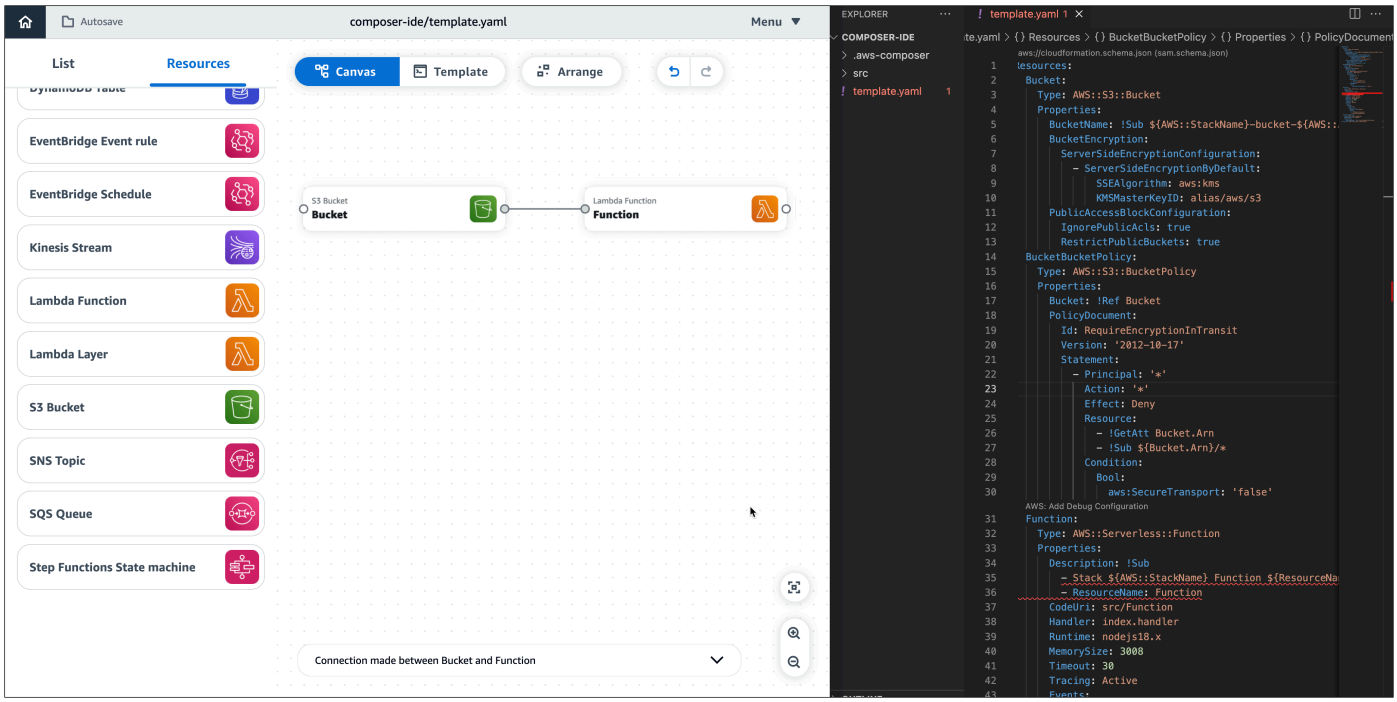
As you compose, Infrastructure Composer automatically creates your AWS CloudFormation and AWS Serverless Application Model (AWS SAM) templates, following AWS best practices. You can view and modify your templates directly from within Infrastructure Composer. Infrastructure Composer automatically syncs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



Integrate with your existing workflows

Import existing templates and projects

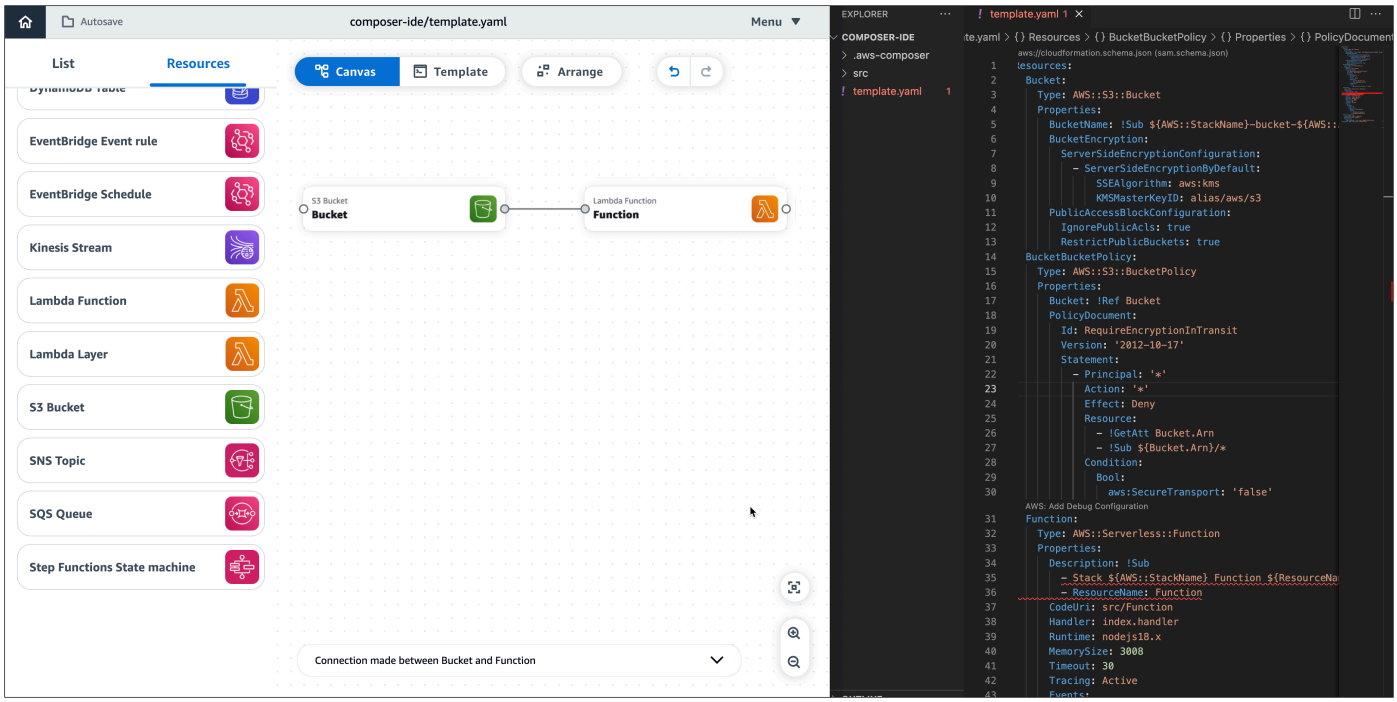
Import existing CloudFormation and AWS SAM templates to visualize them for better understanding and modify their design. Export the templates that you create within Infrastructure Composer and integrate them into your existing workflows towards deployment.



Ways to access Infrastructure Composer

From the Infrastructure Composer console

Access Infrastructure Composer through the Infrastructure Composer console to get started quickly. Additionally, you can use **local sync** mode to automatically sync and save Infrastructure Composer with your local machine.



From the CloudFormation console

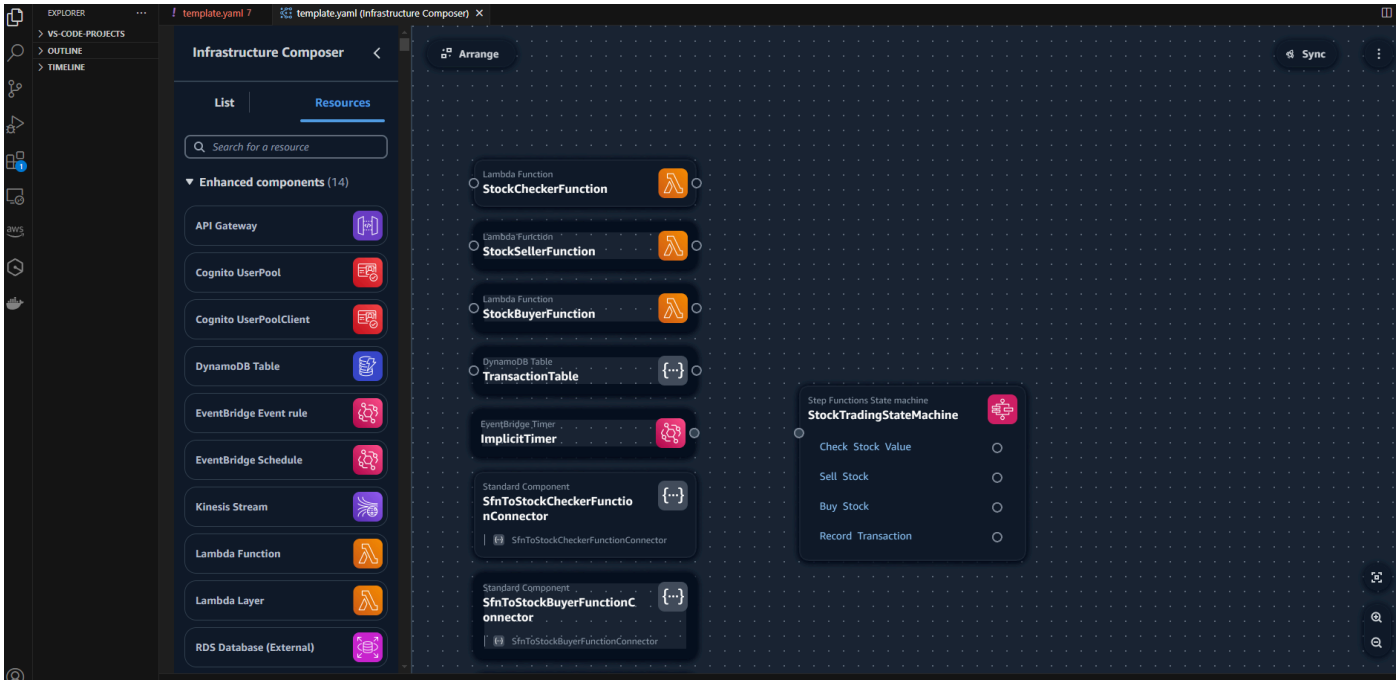
The Infrastructure Composer console also supports CloudFormation console mode, an improvement from CloudFormation Designer that is integrated with the CloudFormation stack workflow. This new tool is now the recommended tool to visualize your CloudFormation templates.

From the Lambda console

With Infrastructure Composer, you can also import Lambda functions from the Lambda console. To learn more, see Import functions into Infrastructure Composer from the Lambda console.

From the AWS Toolkit for Visual Studio Code

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



Learn more

To continue learning about Infrastructure Composer, see the following resources:

- Infrastructure Composer cards
- Visually compose and create serverless applications | Serverless Office Hours – Overview and demo of Infrastructure Composer.

Next steps

To set up Infrastructure Composer, see Getting started with the Infrastructure Composer console.

Serverless concepts for AWS Infrastructure Composer

Learn about basic serverless concepts before using AWS Infrastructure 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*.

Infrastructure Composer cards

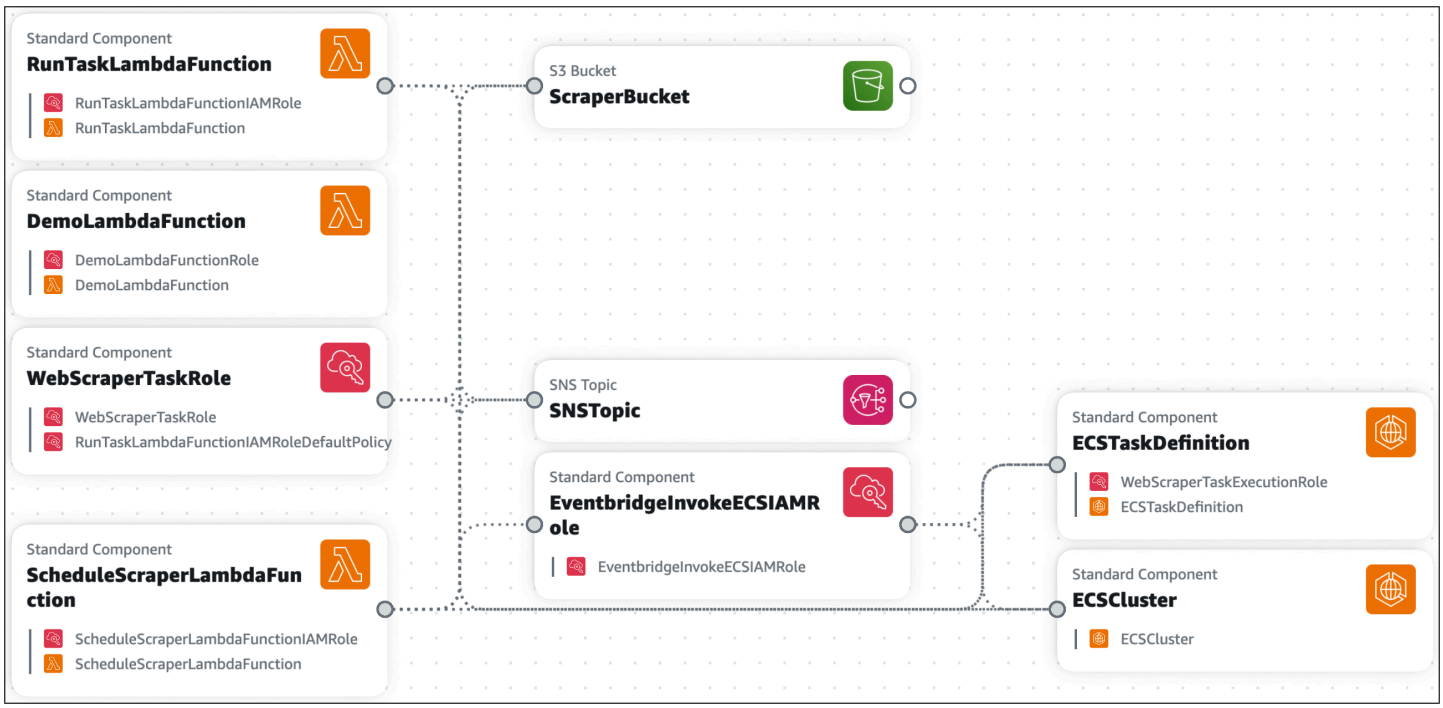
Infrastructure Composer simplifies the process of writing infrastructure as code (IaC) for CloudFormation resources. To effectively use Infrastructure Composer, there are two basic concepts you should first understand: Infrastructure Composer cards and card connections.

In Infrastructure Composer, cards represent CloudFormation resources. there are two general categories of cards:

- Enhanced component card – A collection of CloudFormation resources that have been combined into a single curated card that enhances ease of use, functionality, and are designed for a wide variety of use cases. Enhanced component cards are the first cards listed in the **Resources** palette in Infrastructure Composer.
- Standard IaC resource card – A single AWS CloudFormation resource. Each standard IaC resource card, once dragged onto the canvas, is labeled **Standard component** and may be combined into multiple resources.

> **Note**
>
> Depending on the card, a *Standard IaC resource* card may be labeled a **Standard component** card after it has been dragged onto the visual canvas. This simply means the card is a collection of one or more standard IaC resource cards.

While some types of cards are available from the **Resources** palette, cards can also appear on the canvas when you import an existing CloudFormation or AWS Serverless Application Model (AWS SAM) template into Infrastructure Composer. The following image is an example of an imported application that contains various card types:



Topics

- Enhanced component cards in Infrastructure Composer
- Standard component cards in Infrastructure Composer
- Card connections in Infrastructure Composer

Enhanced component cards in Infrastructure Composer

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

Enhanced component cards are available from the *Resources* palette, under the *Enhanced components* section.

Enhanced component cards can be fully configured and used within Infrastructure Composer to design and build your serverless applications. We recommend using enhanced component cards when designing your applications with no existing code.

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

Example

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

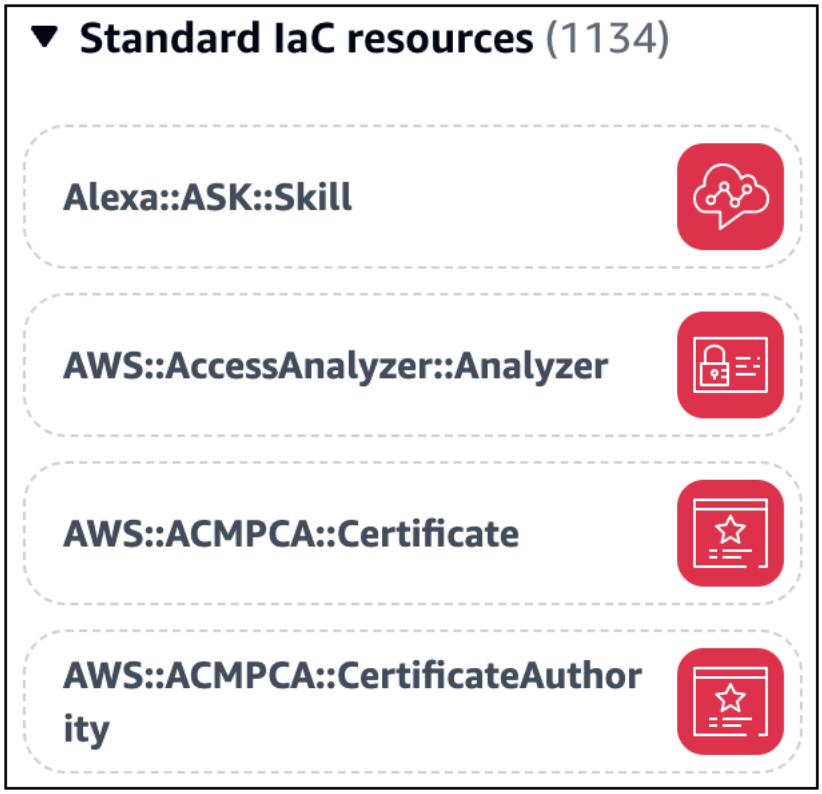
When you drag an **S3 Bucket** component card onto the canvas and view your template, you will see the following two CloudFormation resources added to your template:

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

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

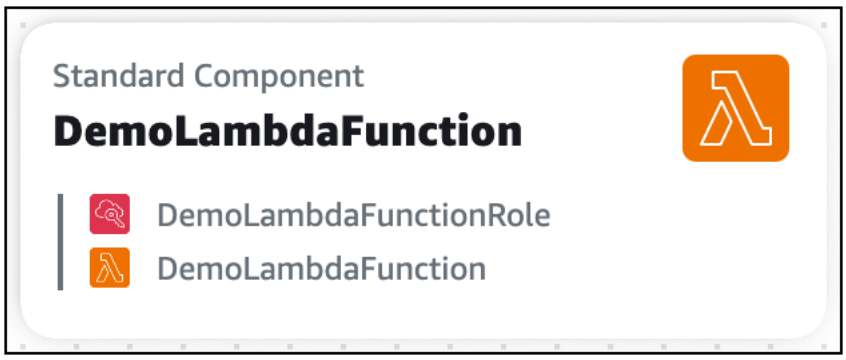
Standard component cards in Infrastructure Composer

Before a standard component card is placed on Infrastructure Composer's visual canvas, it is listed as a **Standard (IaC) resource** card on the **Resources** palette in Infrastructure Composer. A standard (IaC) resource card represents a single CloudFormation resource. Each standard IaC resource card, once placed on the visual canvas, becomes a card labeled **Standard component**, and may be combined to represent multiple CloudFormation resources.

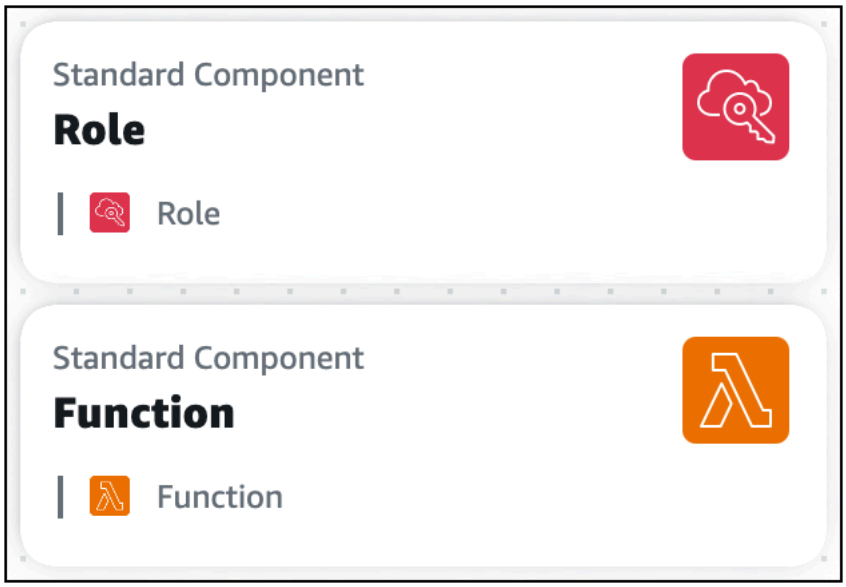


Each standard IaC resource card can be identified by its CloudFormation resource type. The following is an example of a standard IaC resource card that represents an `AWS::ECS::Cluster` CloudFormation resource type:

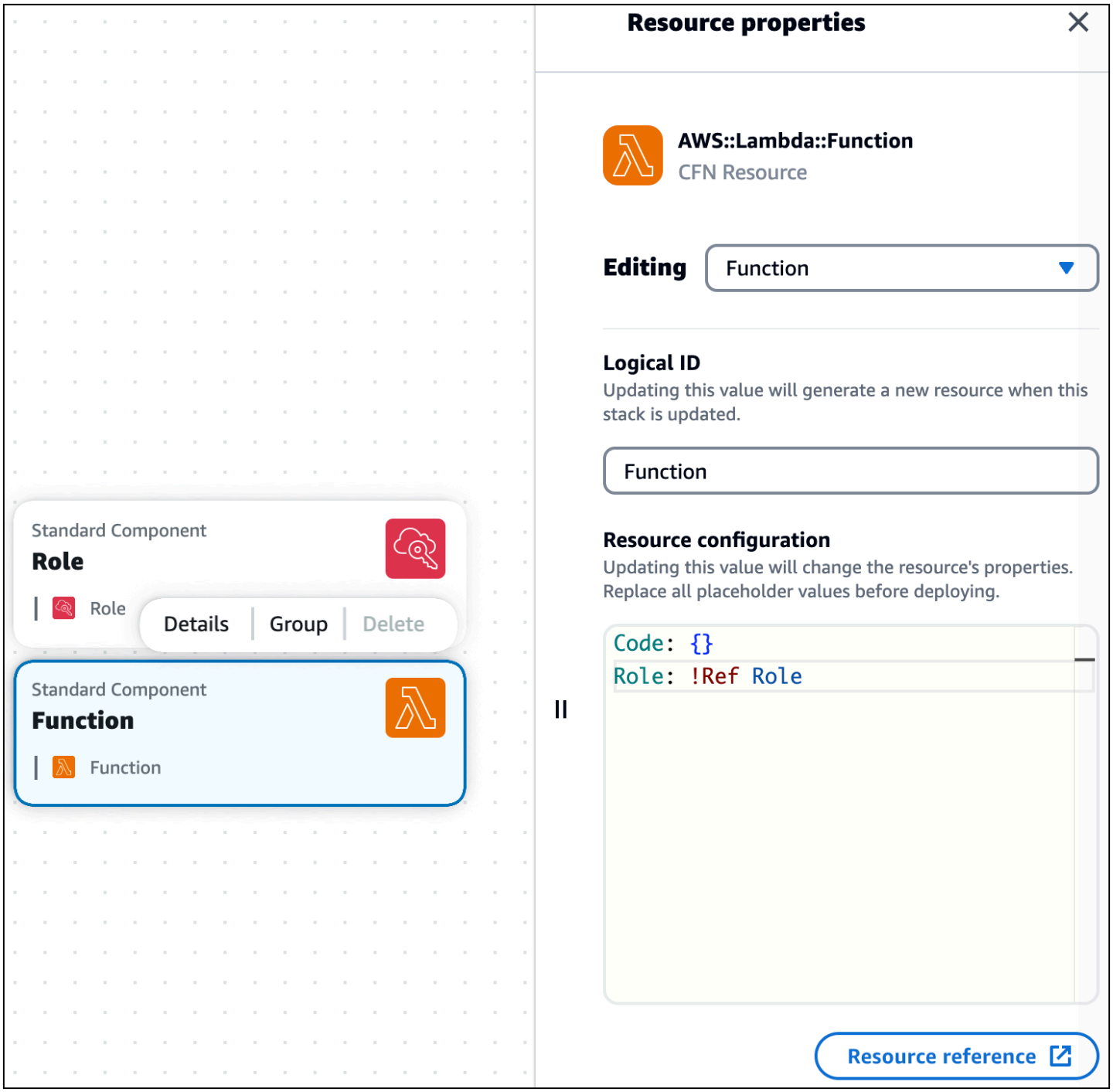
Each standard component card visualizes the CloudFormation resources that it contains. The following is an example of a standard component card that includes two standard IaC resources:



As you configure the properties of your standard component cards, Infrastructure 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.

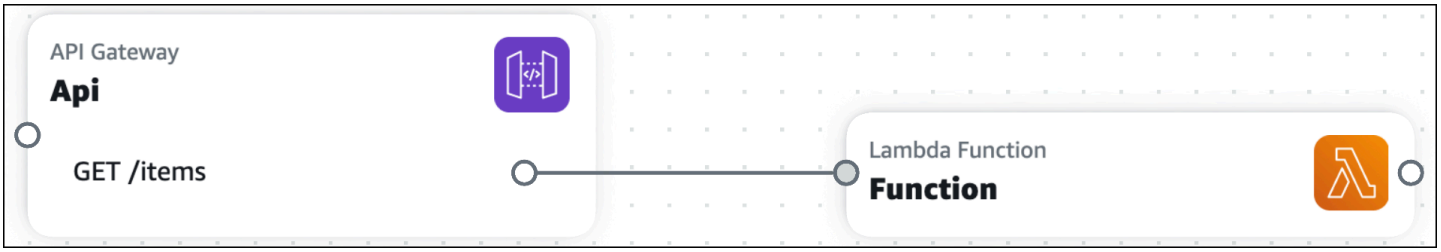
Card connections in Infrastructure Composer

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

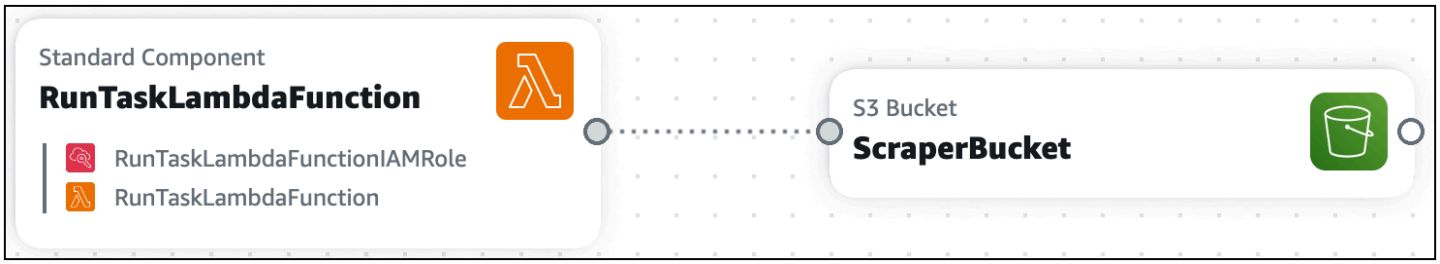
Topics

Connections between cards

How you connect cards together varies depending on the card type. Each enhanced card has at least one connector port. To connect them, you simply select one connector port and drag it to the port of another card, and Infrastructure Composer will connect the two resources or display a message stating this configuration isn't supported.



As seen above, lines between enhanced component cards are solid. Conversely, standard IaC resource cards (also referred to as standard component cards) do not have connector ports. For these cards, you must specify these event-driven relationships in your application's template, and Infrastructure Composer will automatically detect their connections and visualize them with a dotted line between your cards.

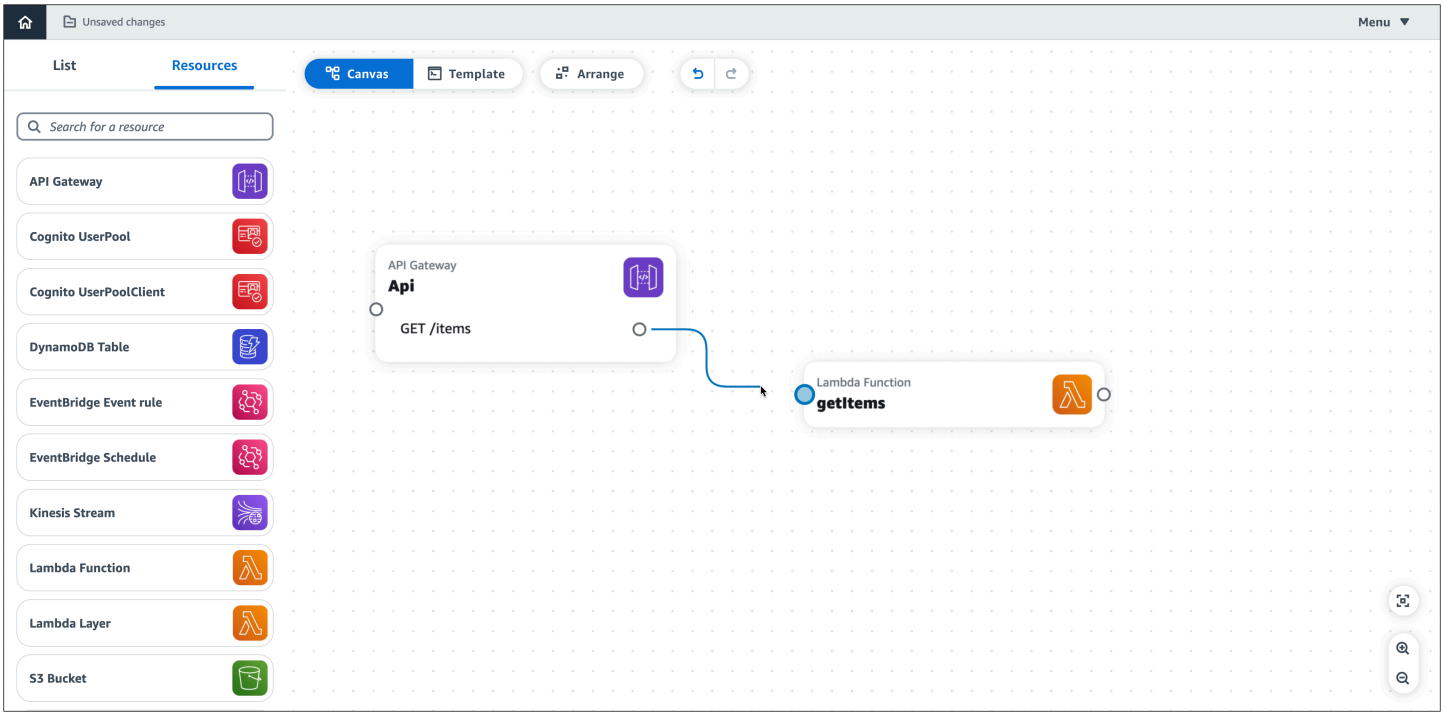


To learn more, see the sections below.

Connections between enhanced component cards

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

To connect two cards, click on a port from one card and drag it onto a port on another card.



> **Note**
>
> Standard IaC resource cards do not have connector ports. For these cards, you must specify their event-driven relationships in your application's template, and Infrastructure Composer will automatically detect their connections and visualize them with a dotted line between your cards.

For more information, see Connect cards on Infrastructure Composer's visual canvas.

What enhanced component cards provision

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

- AWS Identity and Access Management (IAM) policies
- Environment variables
- Events

IAM policies

When a resource needs permission to invoke another resource, Infrastructure 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, Infrastructure Composer defines the infrastructure code to utilize environment variables between resources.

Events

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

Connections to and from standard IaC resource cards

All CloudFormation resources are available to use as standard IaC resource cards from the **Resources** palette. When you drag a standard IaC resource card onto the canvas, a standard IaC resource card becomes a standard component card, and this prompts Infrastructure Composer to create a starting template for your resource in your application.

For more information, see Standard cards in Infrastructure Composer.

Getting started with the Infrastructure Composer console

Use the topics in this section to set up AWS Infrastructure Composer and learn how to design an application using its visual canvas. The tour and tutorials in this section are shown in the Infrastructure Composer console, which is the default user experience. The topics in this section shows you how to complete pre-requisites for using Infrastructure Composer, use the Infrastructure Composer console, load and modify a project, and build your first application.

Infrastructure Composer is also available from the AWS Toolkit for Visual Studio Code and in CloudFormation console mode. Experiences between tools are generally the same but there are some differences between each. For details on using Infrastructure Composer in each of these tools, see Where you can use Infrastructure Composer.

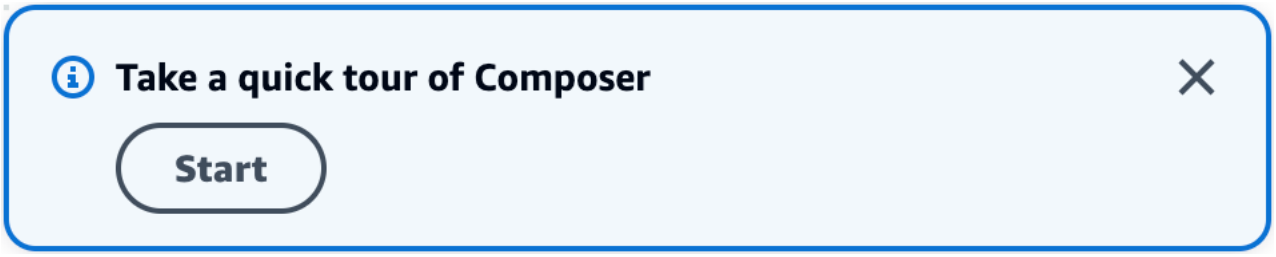
Topics

Take a tour in the Infrastructure Composer console

To get a general idea of how AWS Infrastructure Composer works, take the tour that is built into the Infrastructure Composer console. For an overview of the Infrastructure Composer console, see Take a tour in the Infrastructure Composer console. For in depth guidance on using Infrastructure Composer, refer to How to compose in AWS Infrastructure Composer.

To take a tour of Infrastructure Composer

1. Sign in to the Infrastructure 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**.

   The tour provides a short overview of basic Infrastructure Composer functonality, like using, configuring, and connecting cards. For more information, refer to How to compose in AWS Infrastructure Composer.

Next steps

To load and modify a project in Infrastructure Composer, see Load and modify the Infrastructure Composer demo project.

Load and modify the Infrastructure Composer demo project

Use this tutorial to become familier with Infrastructure Composer's user interface and learn how to load, modify, and save the Infrastructure Composer demo project.

This tutorial is done in the Infrastructure Composer console. Once completed, you'll be ready to start Build your first application with Infrastructure Composer.

Topics

Step 1: Open the demo

Start using Infrastructure Composer by creating a demo project.

To create a demo project

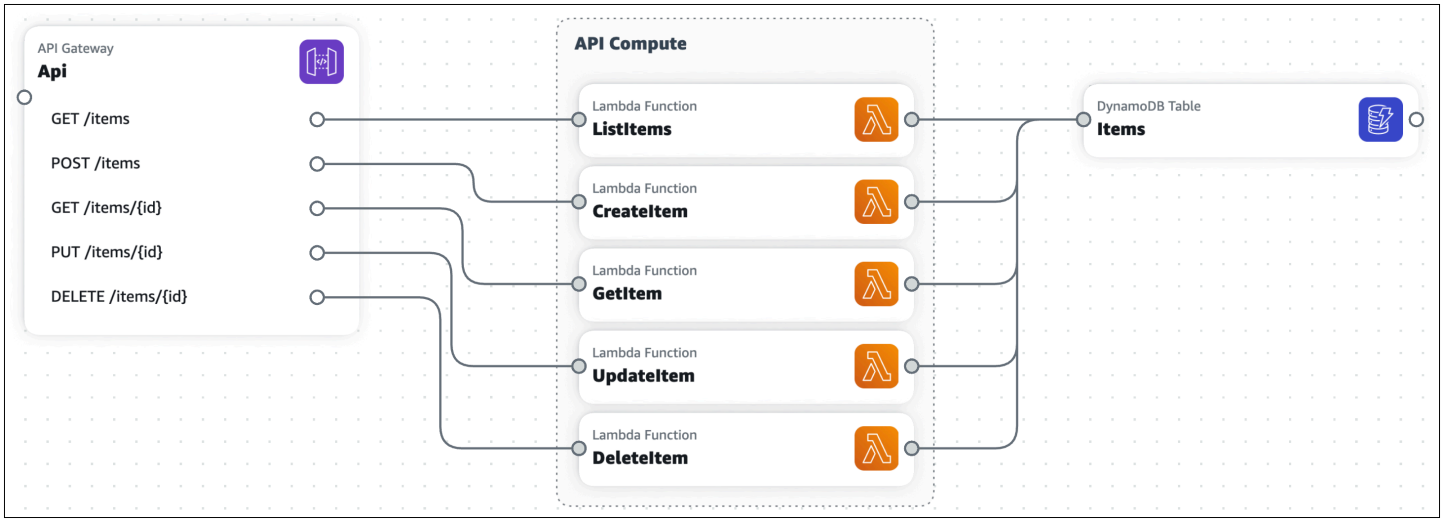
1. Sign in to the Infrastructure Composer console.

2. On the **Home** page, choose **Open demo**.

The demo application is a basic create, read, delete, and update (CRUD) serverless application that includes:

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

The following image is of the demo:

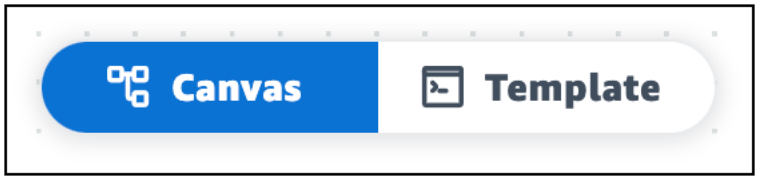


Step 2: Explore the visual canvas of Infrastructure Composer

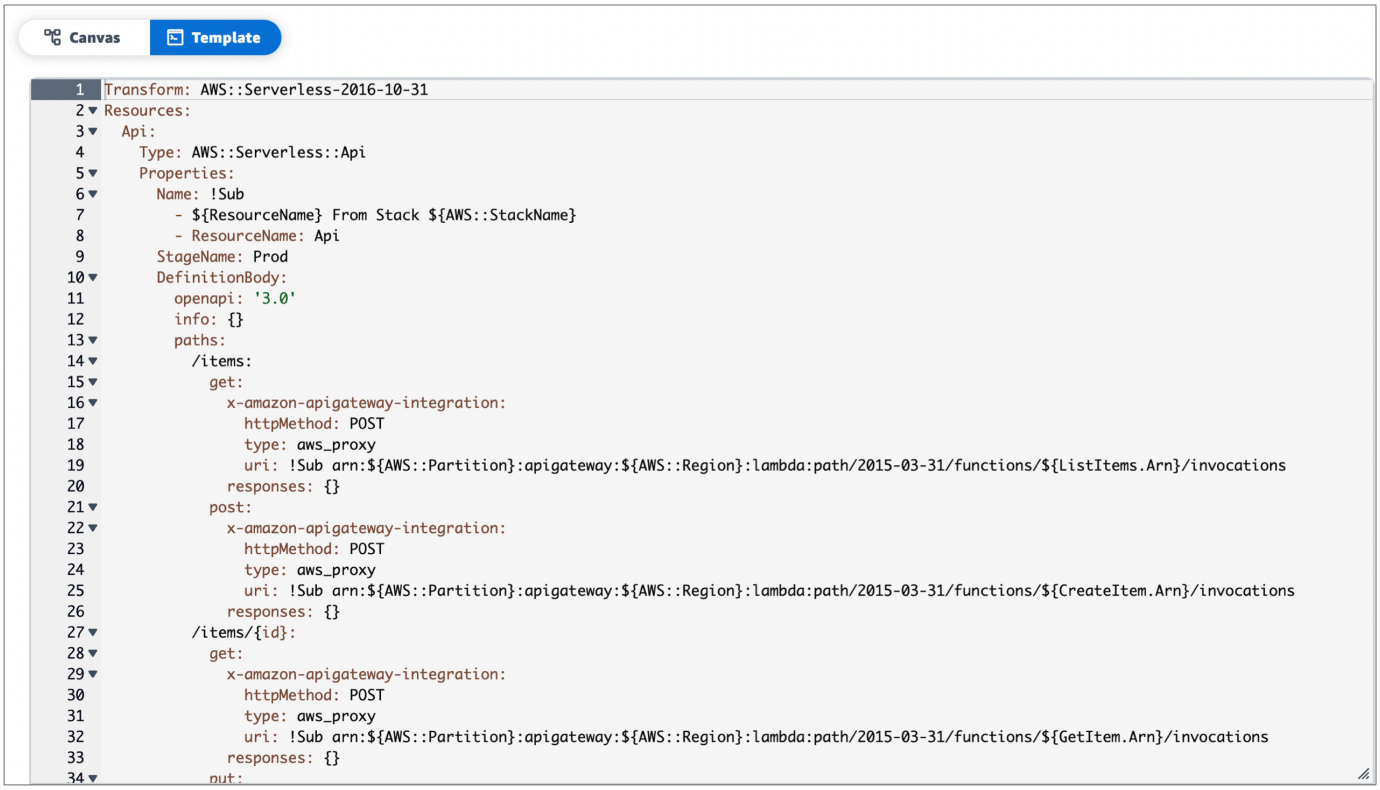
Learn the features of the visual canvas to build out your Infrastructure Composer demo project. For an overview of the visual canvas layout, see Visual overview.

To explore the features of the visual canvas

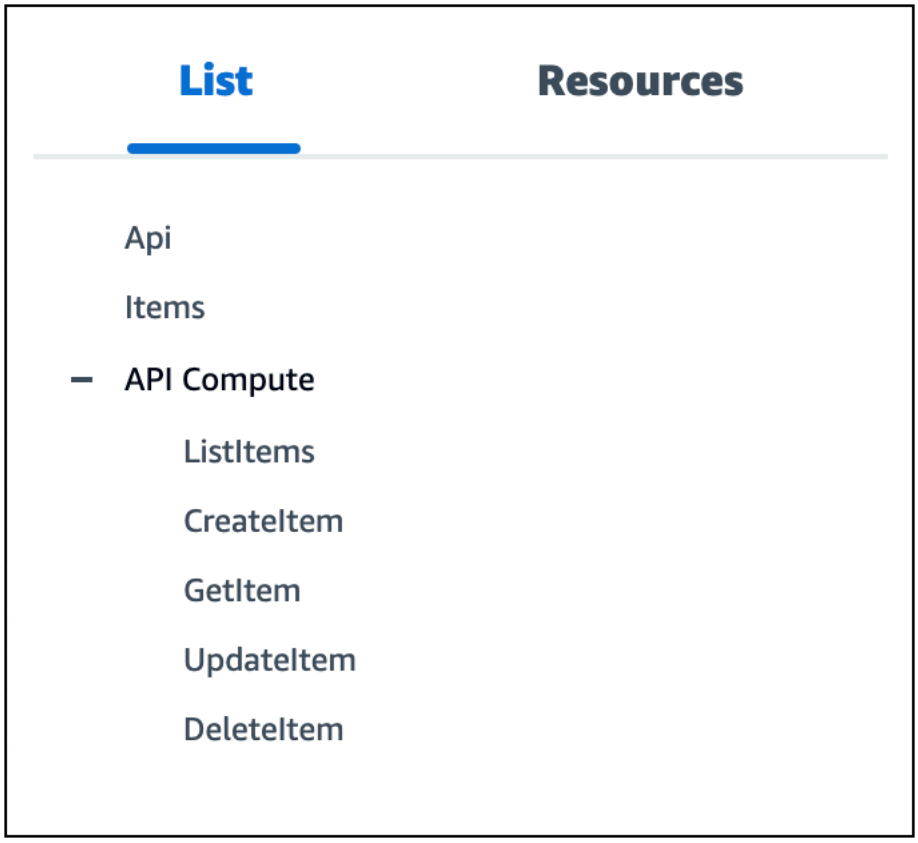
1. When you open a new or existing application project, Infrastructure Composer loads the canvas view, as indicated above the main view area.



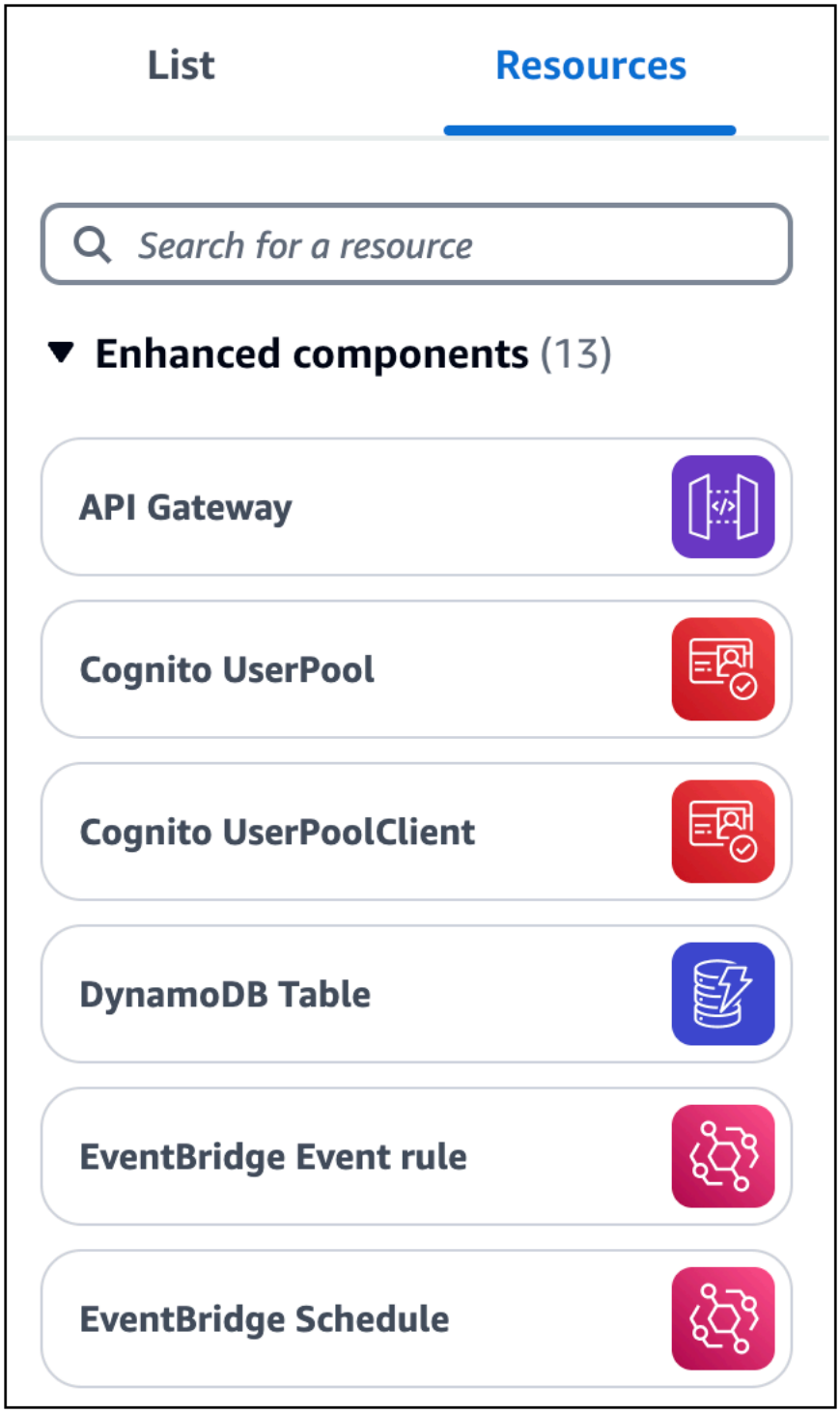
To show your application's infrastructure code in the main view area, choose **Template**. For example, here is the AWS Serverless Application Model (AWS SAM) template view of the Infrastructure Composer demo project.



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.



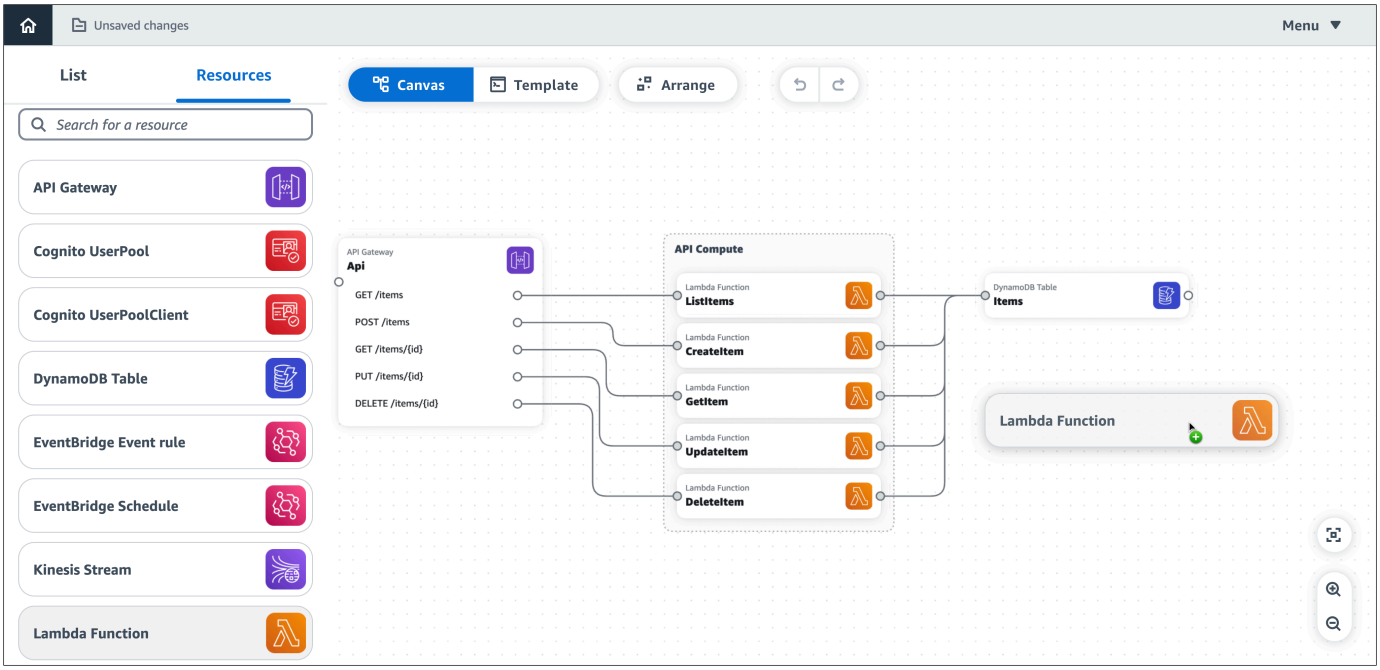
5. To move around the visual canvas, use basic gestures. For more information, see Place cards on the canvas.

Step 3: Expand your application architecture

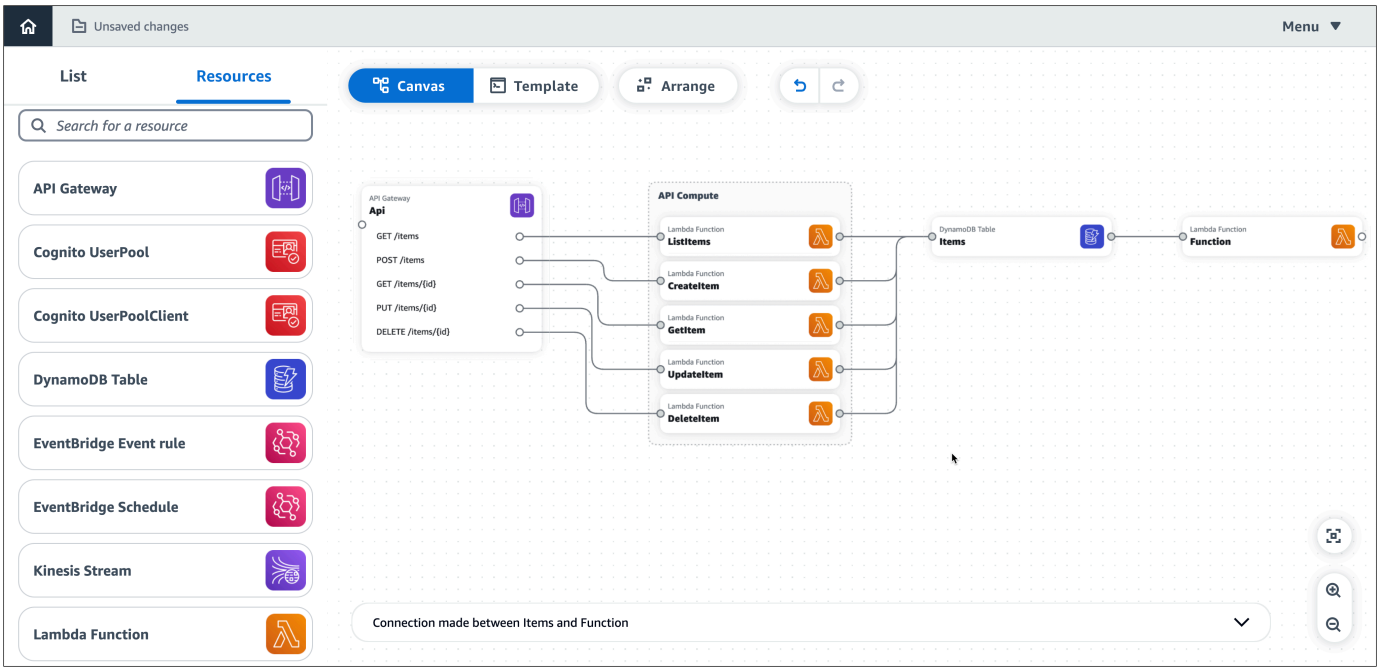
In this step, you will expand your application architecture by adding a Lambda function to your DynamoDB table.

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`). Infrastructure 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 Locally sync and save your project in the Infrastructure Composer console.

Next steps

To get started with building your first application, see Build your first application with Infrastructure Composer.

Build your first application with Infrastructure Composer

In this tutorial, you use AWS Infrastructure Composer to build a create, read, update, and delete (CRUD) serverless application that manages users in a database.

For this tutorial, we use Infrastructure 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 for AWS Infrastructure Composer.

Topics

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

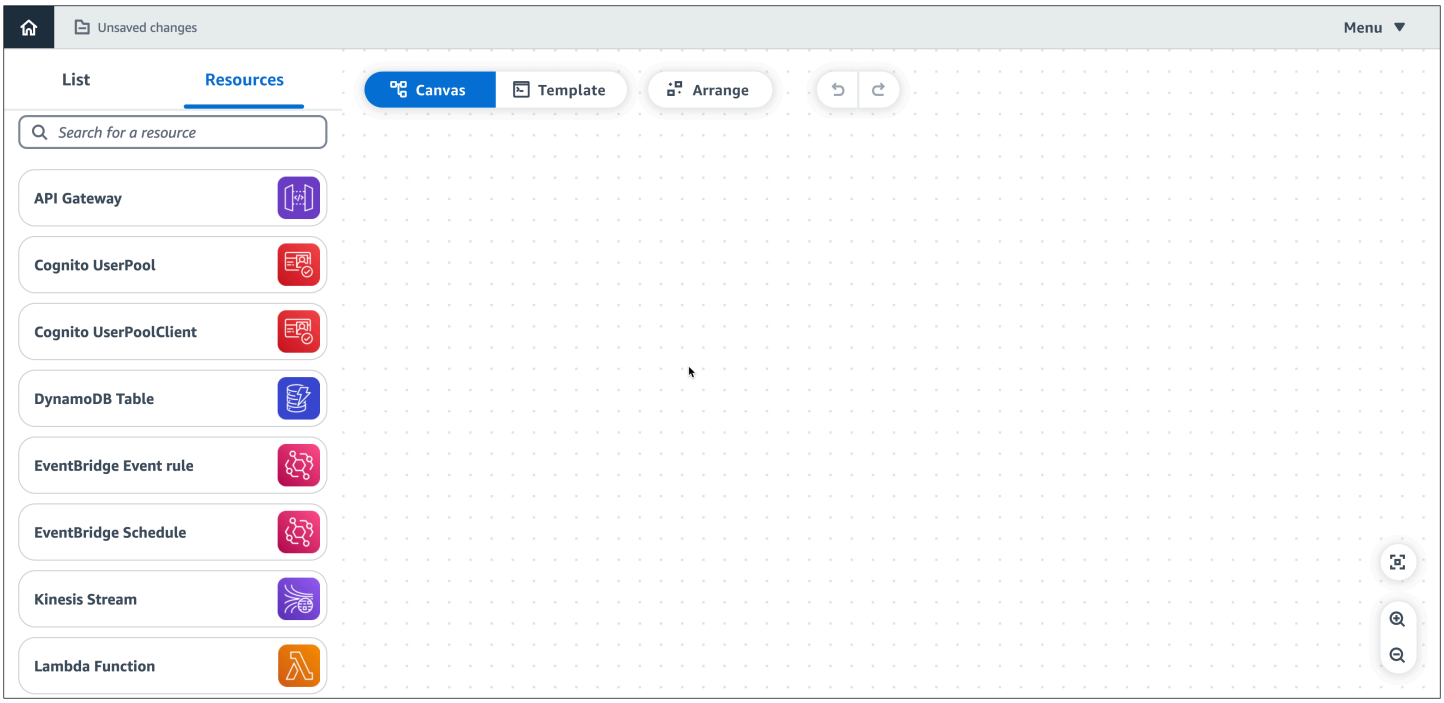
Step 1: Create your project

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

To create a new blank project

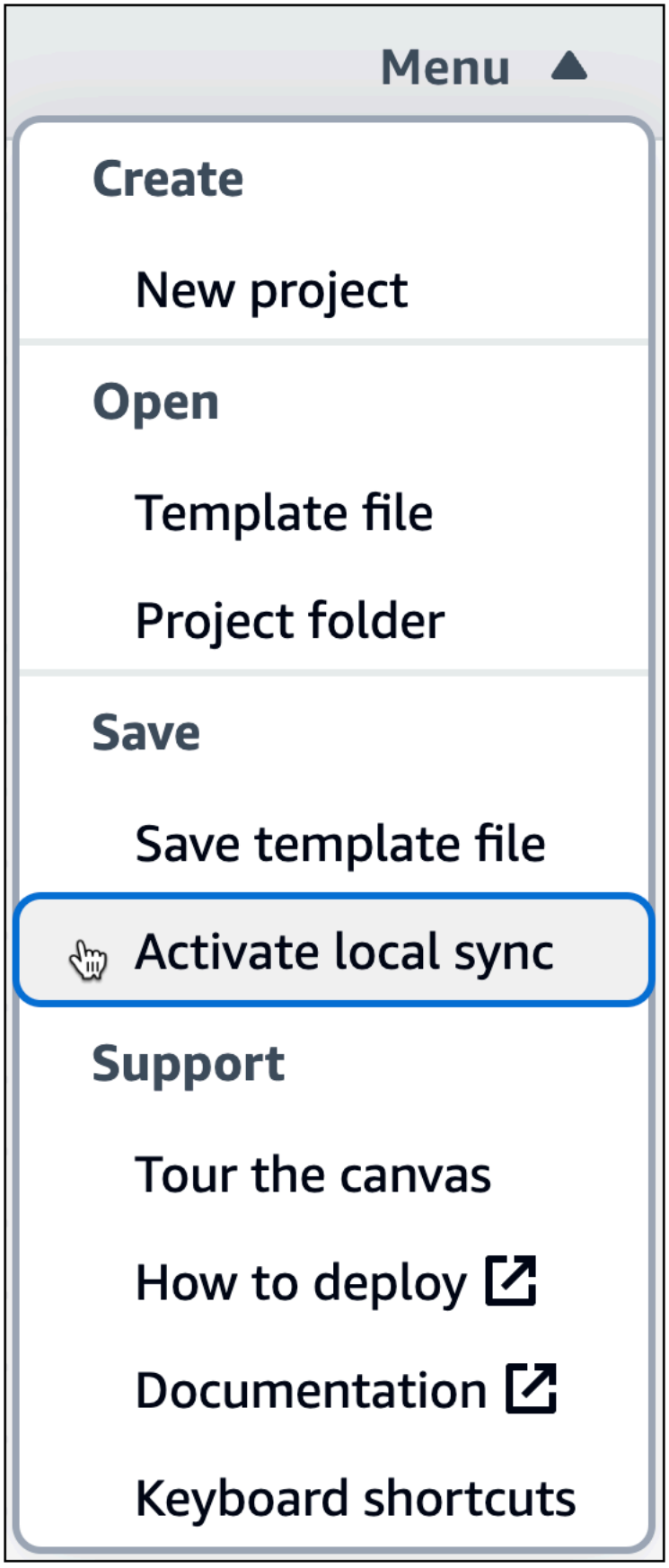
1. Sign in to the Infrastructure Composer console.
2. On the **Home** page, choose **Create project**.

As shown in the following image, Infrastructure Composer opens the visual canvas and loads a starting (blank) application template.



To activate local sync

1. From the Infrastructure 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 Infrastructure 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 Data Infrastructure Composer gains access to.

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.

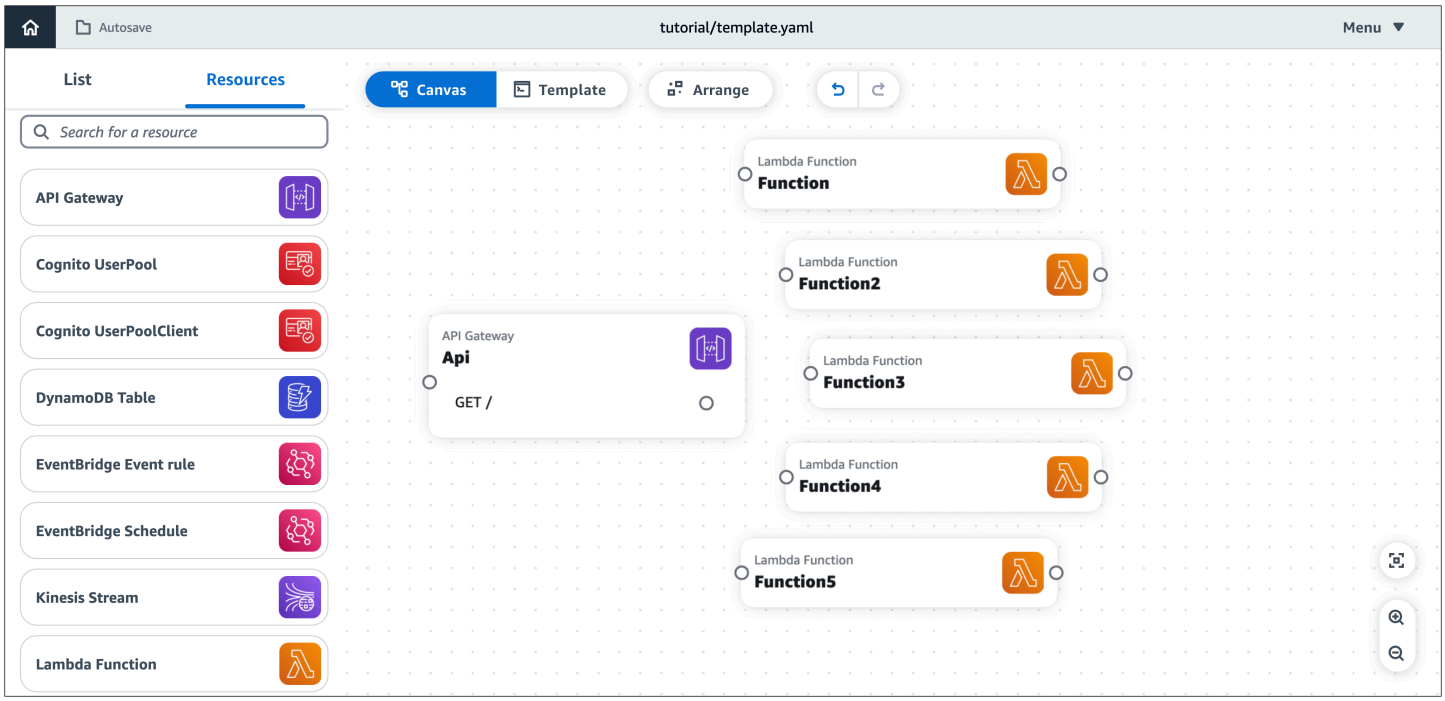
Step 2: Add cards to the canvas

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:

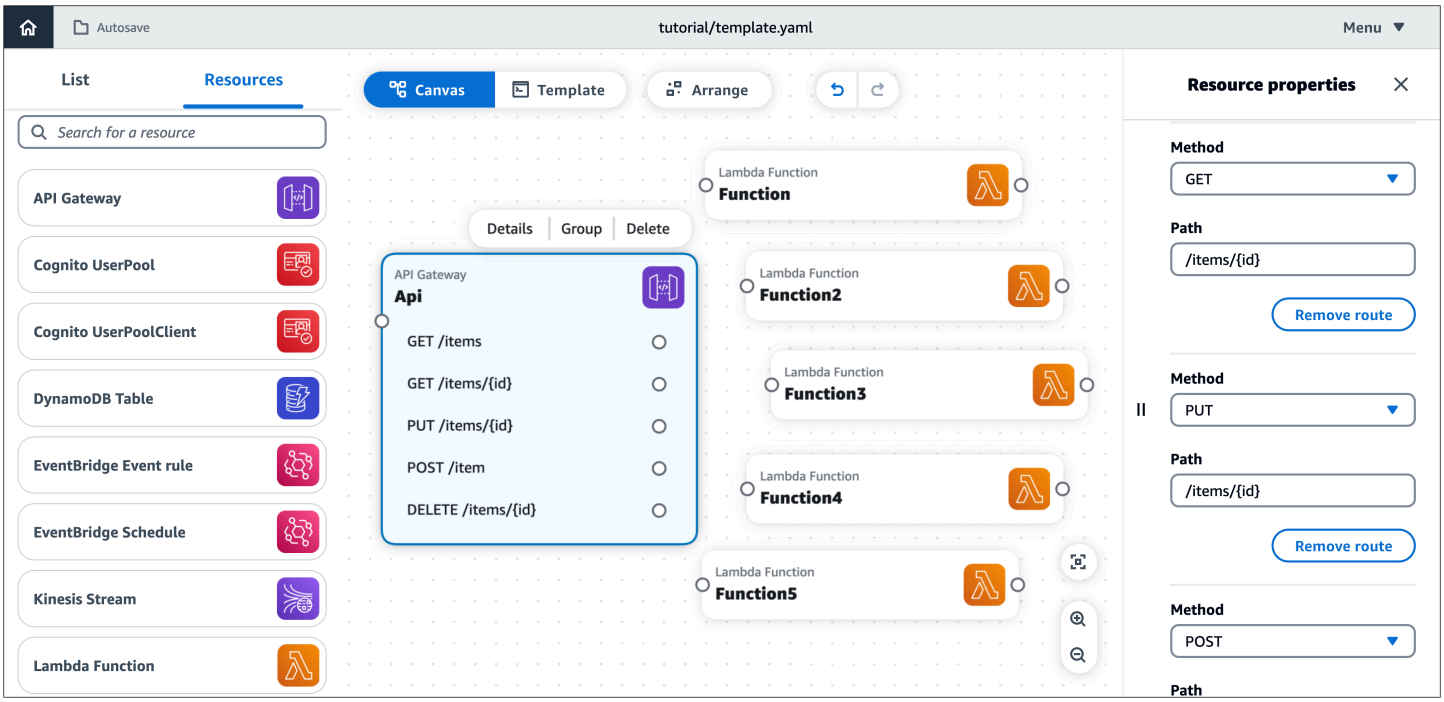
   > **Note**
   >
   > For each of the following routes, use the HTTP method and path values specified in the resource properties reference table.

   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**.

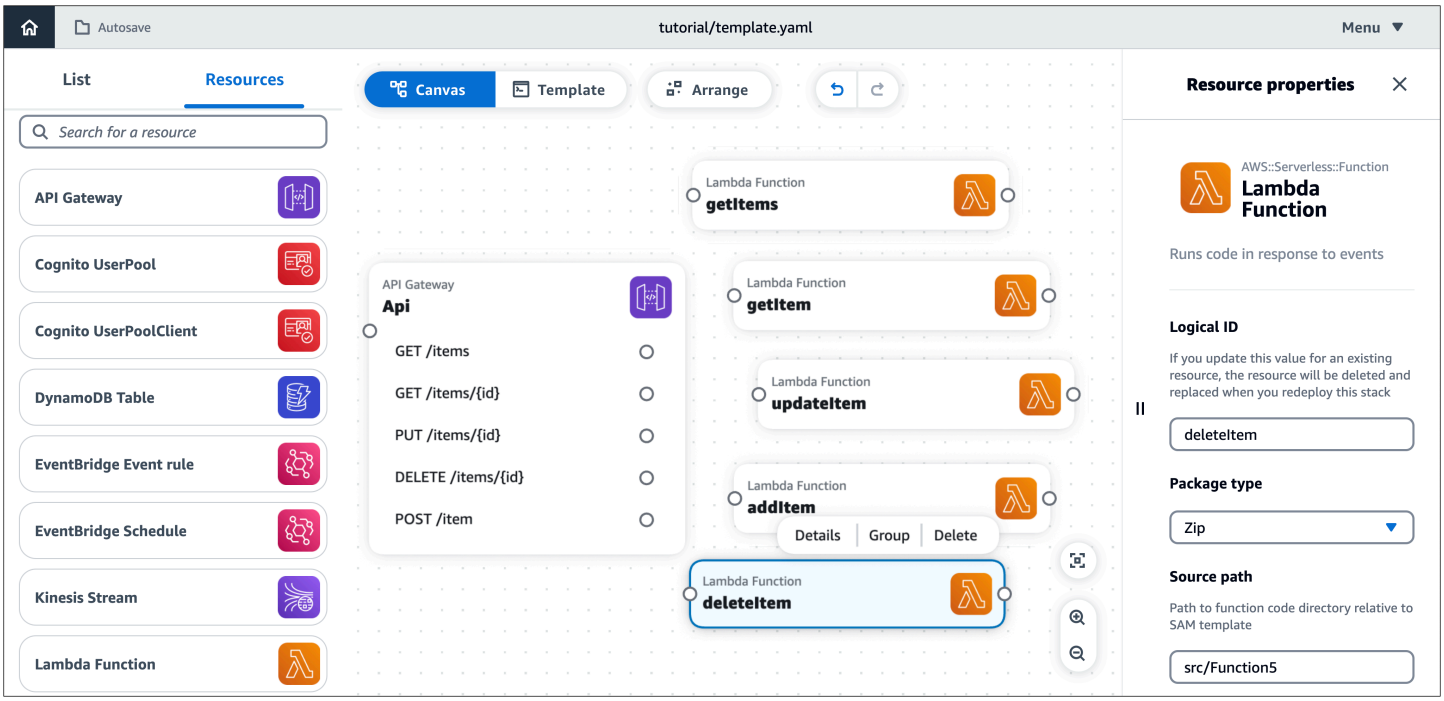


Step 4: Configure your Lambda functions

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.

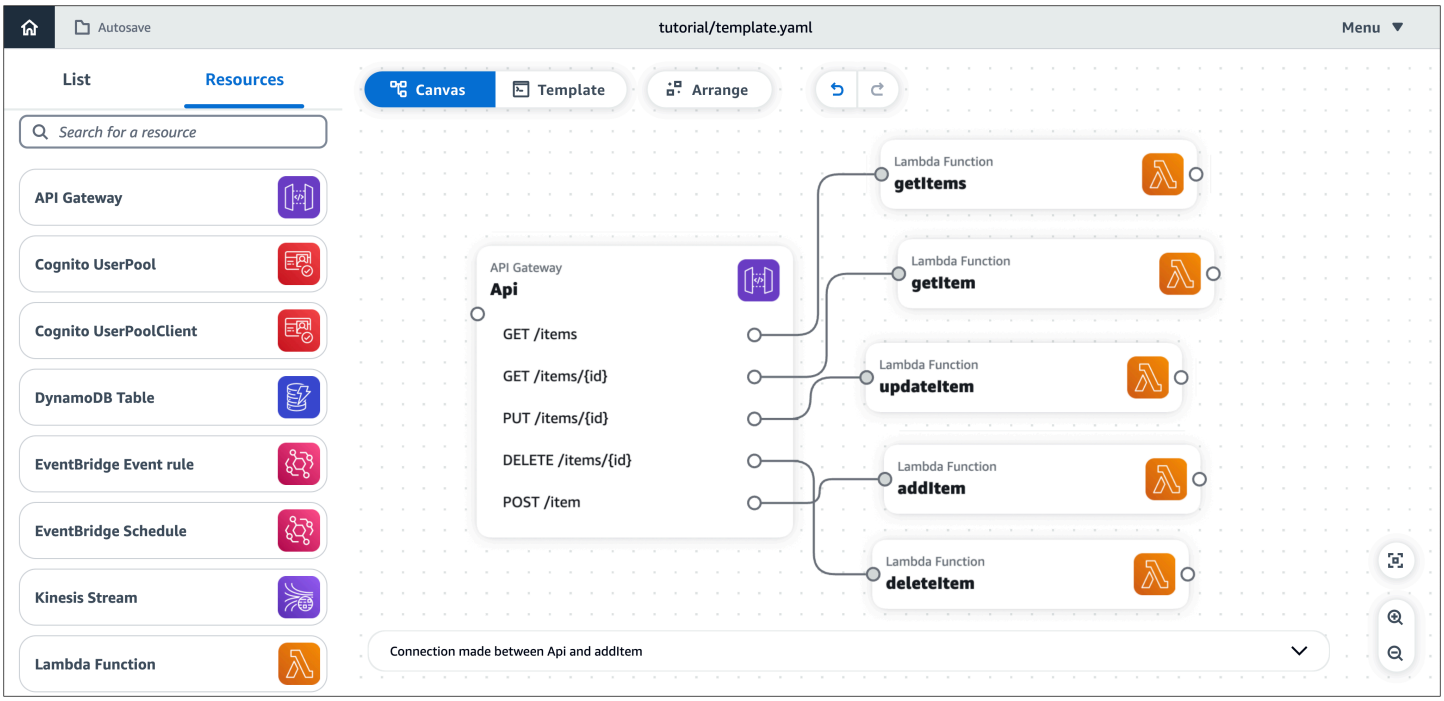


Step 5: Connect your cards

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

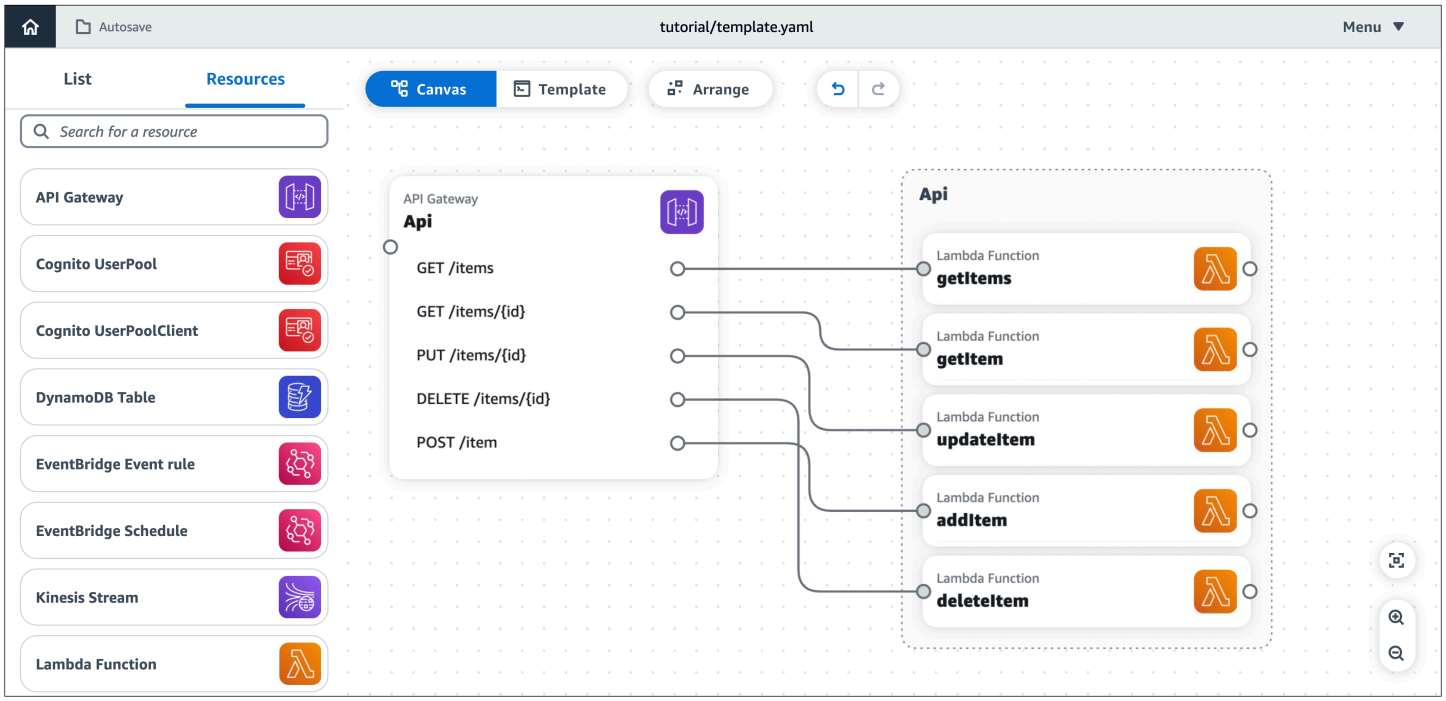
1. Double-click the top of the group, near the group name (**Group**).

   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**.

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

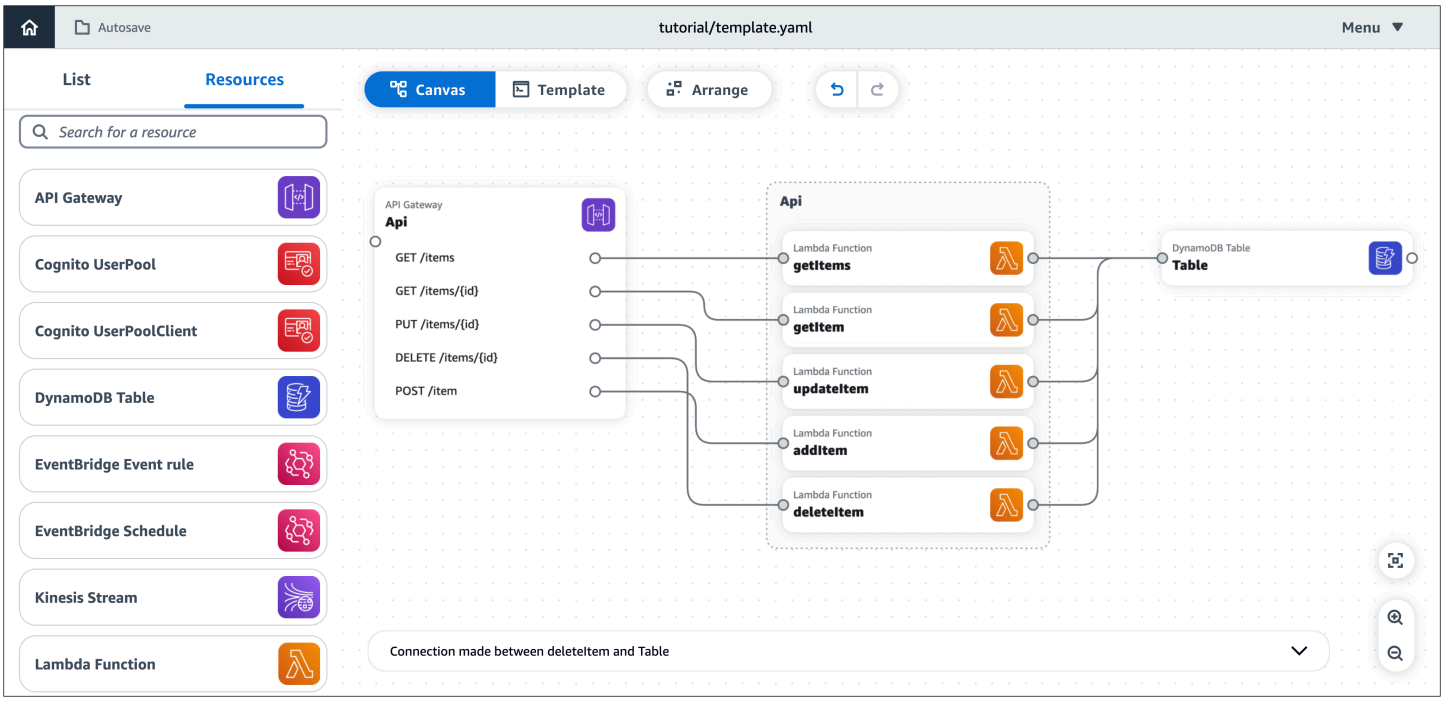


Step 7: Add and connect a DynamoDB table

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**.

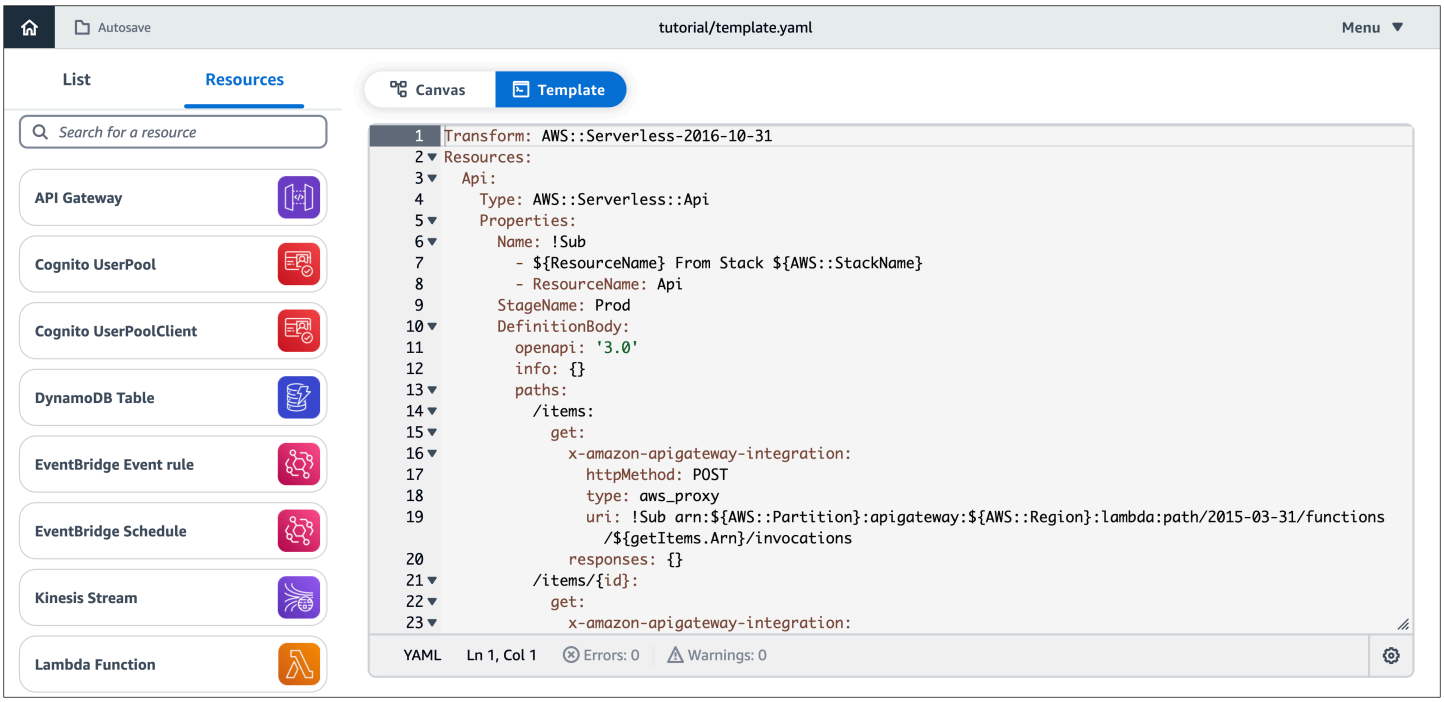


Step 8: Review your AWS CloudFormation template

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

In the template, Infrastructure 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 Infrastructure Composer created for further testing and deployment.

- With **local sync**, you can connect Infrastructure Composer to the IDE on your local machine to speed up development. To learn more, see Connect the Infrastructure Composer console 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 Deploy your Infrastructure Composer serverless application to the AWS Cloud.

Next steps

You're now ready to build your own applications with Infrastructure Composer. For in-depth details on using Infrastructure Composer, refer to How to compose in AWS Infrastructure Composer. When you are ready to deploy your application, refer to Deploy your Infrastructure Composer serverless application to the AWS Cloud.

Where you can use Infrastructure Composer

You can use Infrastructure Composer from its console, from AWS Toolkit for Visual Studio Code, and in Infrastructure Composer in CloudFormation console mode. While each varies for slightly different use cases, overall they are similar experiences. This section provides details of each experience.

The topic Using the AWS Infrastructure Composer console is a comprehensive overview of the default console experience. The topic CloudFormation console mode provides details on a version of Infrastructure Composer that is integrated with the CloudFormation stack workflow. AWS Toolkit for Visual Studio Code provides information on accessing and using Infrastructure Composer in VS Code.

Topics

Using the AWS Infrastructure Composer console

This section provides details on accessing and using AWS Infrastructure Composer from the Infrastructure Composer console. This is the default experience for Infrastructure Composer and is a good way to become familiar with Infrastructure Composer. You can also integrate the Infrastructure Composer console with your local IDE. For details, see Connect the Infrastructure Composer console with your local IDE.

You can also access Infrastructure Composer from the AWS Toolkit in VS Code, and you can use a mode of Infrastructure Composer that is specifically designed to be used in CloudFormation.

For general documentation on using Infrastructure Composer, see How to compose.

Topics

- Locally sync and save your project in the Infrastructure Composer console
- Import functions into Infrastructure Composer from the Lambda console
- Export an image of Infrastructure Composer's visual canvas

AWS Infrastructure Composer console visual overview

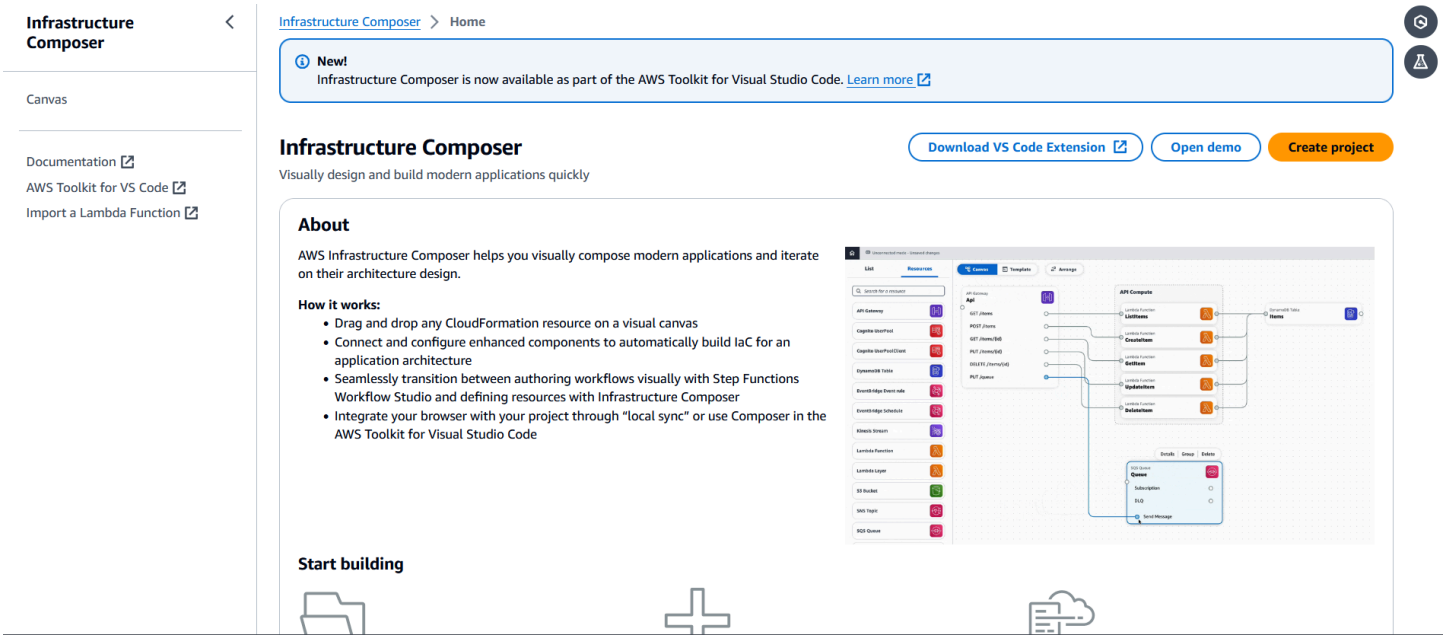
This section provides a visual overview of the AWS Infrastructure Composer console.

Topics

- Home page
- Visual designer and visual canvas

Home page

The following image is of the home page in the Infrastructure Composer console:

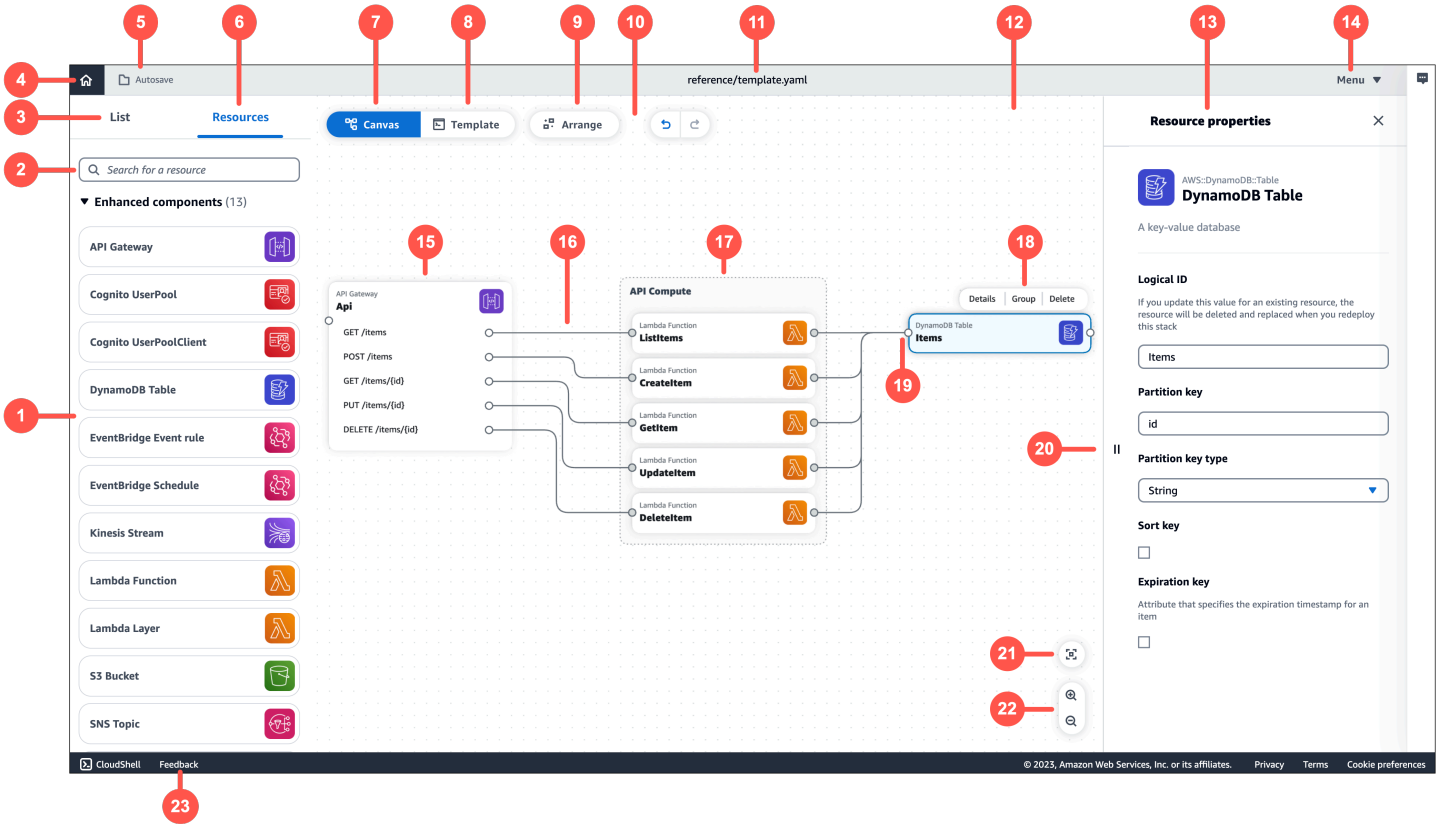


1. **Documentation** – Go to Infrastructure Composer documentation.
2. **Canvas** – Go to the canvas and create or load a project.
3. **Demo** – Open the Infrastructure Composer demo application.
4. **Create project** – Create or load a project.
5. **Start building** – Quick links to start building an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

6. **Feedback** – Go here to submit feedback.

Visual designer and visual canvas

The following image is of Infrastructure Composer's visual designer and visual canvas:



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 Infrastructure Composer homepage.
5. **Save status** – Indicates whether Infrastructure 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.

Manage your project from the Infrastructure Composer console

This topic provides guidance on the basic tasks you perform to manage your project from the Infrastructure Composer console. This includes common tasks like creating a new project, saving a project, and importing a project or template. You can also load an existing project if you activate local sync mode. 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.
- Use Infrastructure Composer to manage your templates and folders

With local sync mode, Infrastructure 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 Infrastructure 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.

> **Note**
>
> Infrastructure Composer supports applications that consist of the following:
>
> - An CloudFormation or AWS Serverless Application Model template that defines your infrastructure code.
> - A folder structure that organizes your project files, such as Lambda function code, configuration files, and build folders.

Topics

Create a new project in the Infrastructure Composer console

When you create a new project, Infrastructure 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 Infrastructure Composer console.
2. On the **Home** page, choose **Create project**.

> **Note**
>
> You can also load an existing in Infrastructure Composer, but you must first activate local sync mode. Once activated, see Load an existing Infrastructure Composer project with local sync activated to load an existing project.

Import an existing project folder in the Infrastructure Composer console

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 Infrastructure 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 Allow web page access to local files in Infrastructure Composer.

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, Infrastructure 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 Allow web page access to local files in Infrastructure Composer.

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, Infrastructure Composer automatically selects it for you.
6. Choose **Create**.

When you import an existing project folder, Infrastructure 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 in the Infrastructure Composer console

When you import an existing CloudFormation or AWS SAM template, Infrastructure 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 Infrastructure 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 in the Infrastructure Composer console

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 Infrastructure 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**.

Connect the Infrastructure Composer console with your local IDE

To connect the Infrastructure Composer console with your local integrated development environment (IDE), use local sync mode. This mode automatically syncs and saves data to your local machine. For more information about local sync mode, see Locally sync and save your project in the Infrastructure Composer console. For instructions on using local sync mode, see Locally sync and save your project in the Infrastructure Composer console.

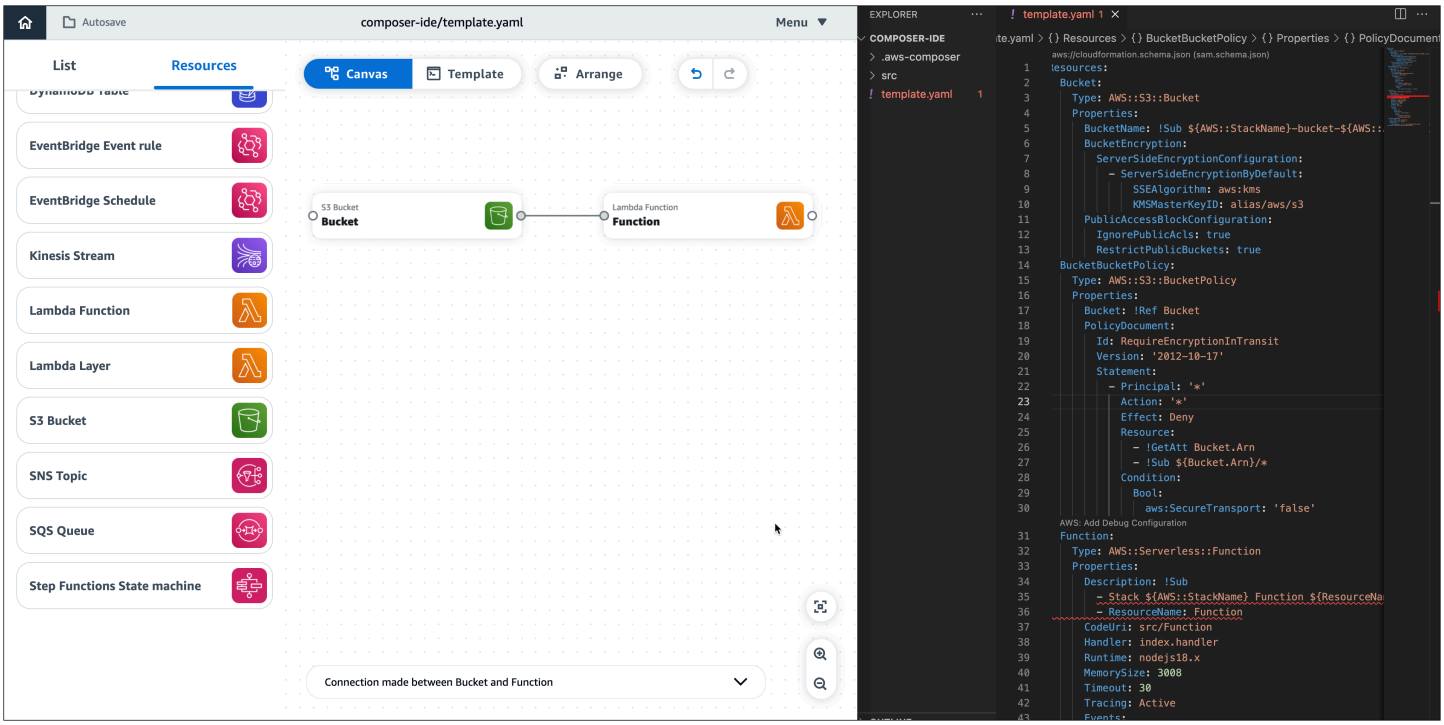
> **Note**
>
> The **Activate local sync** option is not available in every browser. It is available in Google Chrome and Microsoft Edge.

Benefits of using Infrastructure Composer with your local IDE

As you design in Infrastructure 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 Infrastructure 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. The following example shows how you can drag and drop resources onto Infrastructure Composer's visual canvas which, in turn, creates markup in your AWS SAM template in your local IDE.



Integrate Infrastructure Composer with your local IDE

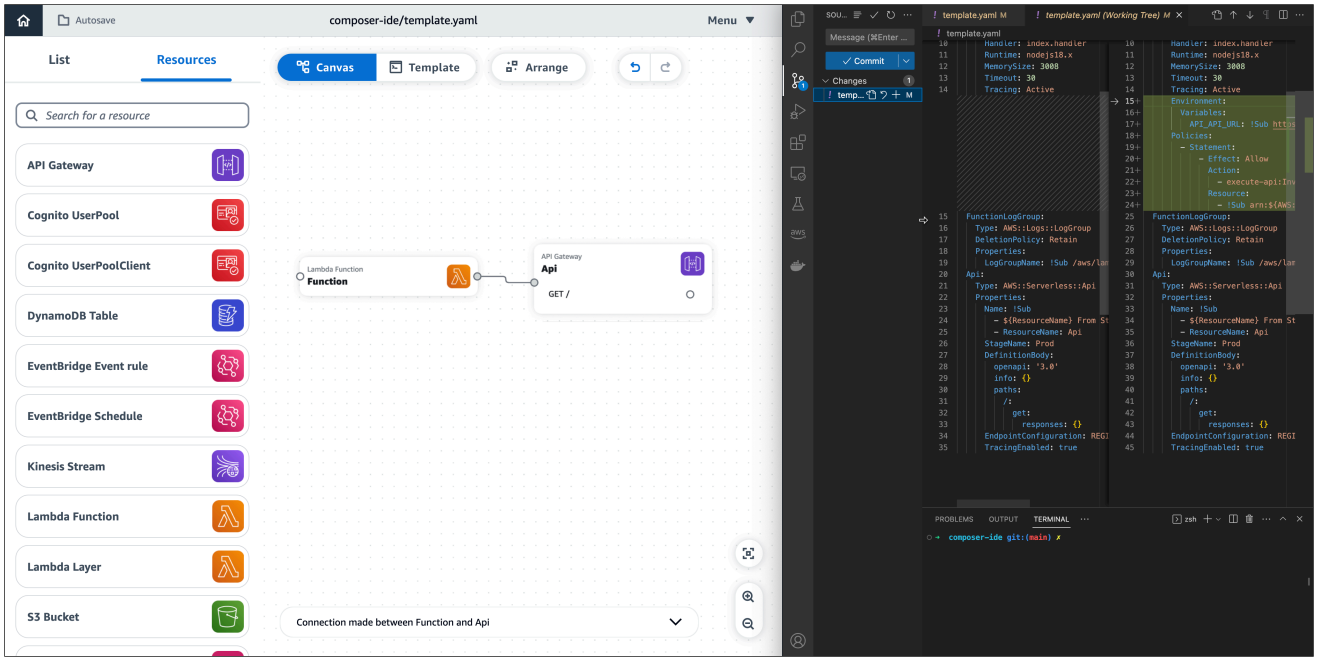
To integrate Infrastructure Composer with your local IDE

1. In Infrastructure Composer, create or load a project, and activate local sync by selecting the **Menu** button at the top-right side of the screen and choosing **Activate local sync**.

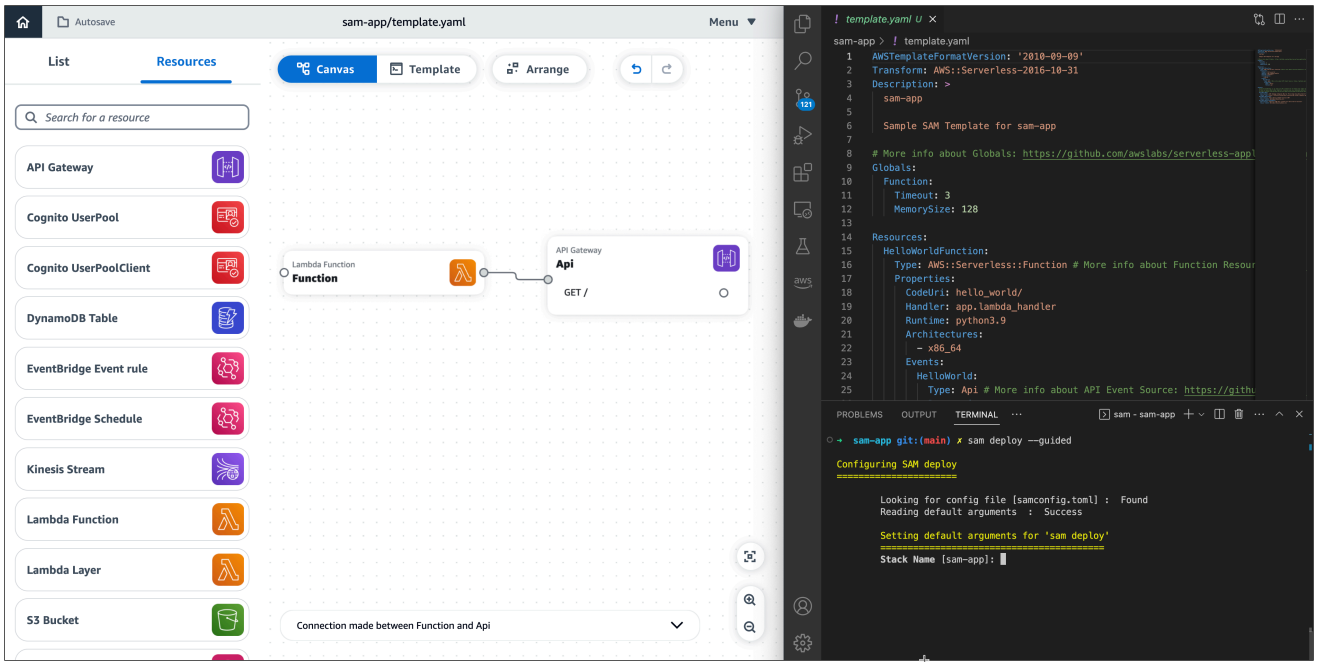
   > **Note**
   >
   > The **Activate local sync** option is not available in every browser. It is available in Google Chrome and Microsoft Edge.

2. In your local IDE, open the same project folder as Infrastructure Composer.
3. Use Infrastructure Composer with your local IDE. Updates made in Infrastructure 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 Infrastructure Composer.



b. Use the AWS SAM CLI locally to build, test, deploy your application, and more. To learn more, see [Deploy your Infrastructure Composer serverless application to the AWS Cloud](#).



Allow web page access to local files in Infrastructure Composer

The Infrastructure Composer console supports local sync mode and Importing functions from the Lambda console. To use these features, a web browser that supports the File System Access API is required. 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 Infrastructure Composer.

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.

local sync mode

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

Data Infrastructure Composer gains access to

Infrastructure 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 Infrastructure 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 Infrastructure Composer *local sync* mode. You can choose another local directory to connect with or use Infrastructure 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 Infrastructure 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 Infrastructure Composer *local sync* mode with your Windows directory.

Locally sync and save your project in the Infrastructure Composer console

This section provides information on using Infrastructure Composer's **local sync** mode to automatically sync and save your project to your local machine.

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

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

- 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 Infrastructure Composer with your local IDE to speed up development. To learn more, see Connect the Infrastructure Composer console with your local IDE.

What local sync mode saves

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.

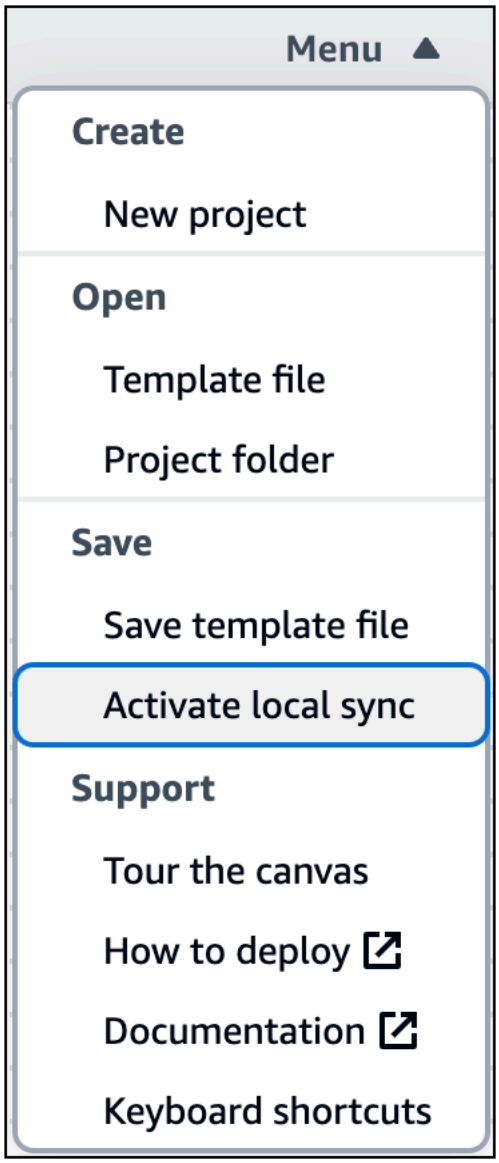
- **External files** – Supported external files that you can use within Infrastructure Composer. To learn more, see Reference external files in Infrastructure Composer.

Browser requirements

Local sync mode requires a browser that supports the File System Access API. For more information, see Allow web page access to local files in Infrastructure Composer.

Activating local sync mode

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



For instructions on activating **local sync** and existing loading projects, see the following topics:.

- Activate local sync in Infrastructure Composer
- Load an existing Infrastructure Composer project with local sync activated

Activate local sync in Infrastructure Composer

To activate local sync, complete the following steps:

1. From the Infrastructure Composer **home** page, select **Create project**.
2. From the Infrastructure Composer **menu**, select **Activate local sync**.
3. For **Project location**, press **Select folder** and choose a directory. This is where Infrastructure 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.

Load an existing Infrastructure Composer project with local sync activated

To load an existing project with local sync activated, complete the following steps:

1. From the Infrastructure Composer **home** page, select **Load a CloudFormation template**.
2. From the Infrastructure 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.

Import functions into Infrastructure Composer from the Lambda console

Infrastructure Composer provides an integration with the AWS Lambda console. You can import a Lambda function from the Lambda console into the Infrastructure Composer console. Then, use the Infrastructure Composer canvas to design your application architecture further.

- This integration requires a browser that supports the File System Access API. For more information, see Allow web page access to local files in Infrastructure Composer.
- When you import your Lambda function into Infrastructure Composer, you must activate local sync mode to save any changes. For more information, see Locally sync and save your project in the Infrastructure Composer console.

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

Export an image of Infrastructure Composer's visual canvas

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

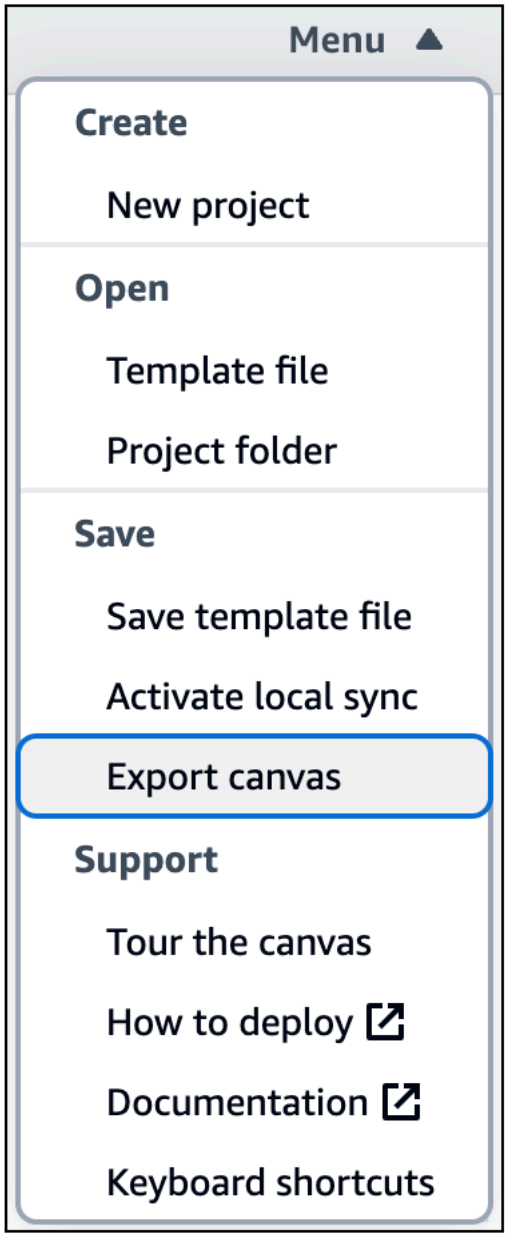
For a visual overview of all Infrastructure Composer features, see AWS Infrastructure Composer console visual overview.

About export canvas

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

- Infrastructure 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.

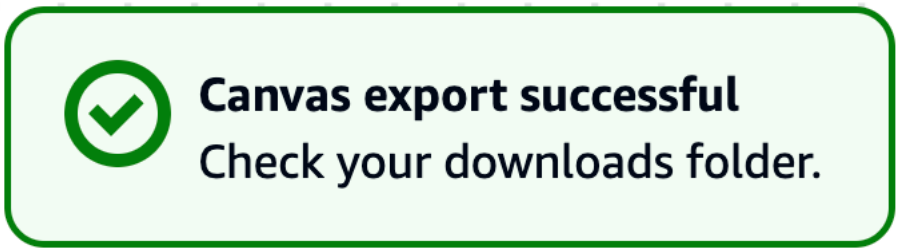
You can access the **export canvas** feature from the **Menu**.



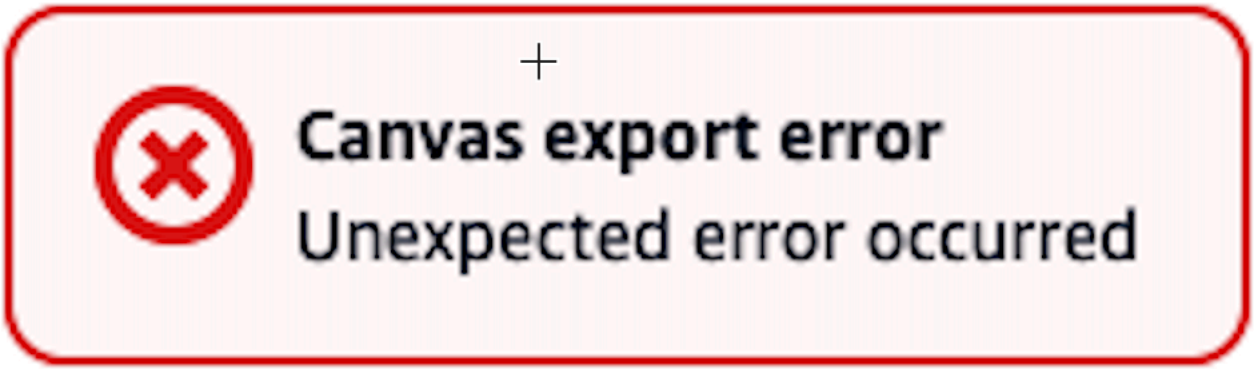
Exporting canvas

When you export your canvas, Infrastructure 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.



Using Infrastructure Composer in CloudFormation console mode

Infrastructure Composer in CloudFormation console mode is the recommended tool to visualize your CloudFormation templates. You can also use this tool to create and edit CloudFormation templates.

How is this mode different than the Infrastructure Composer console?

Infrastructure Composer in CloudFormation console mode generally has the same functionality as the default Infrastructure Composer console, but there are a few differences to note.

- This mode is integrated with the stack workflow in the CloudFormation console. This allows you to use Infrastructure Composer directly in CloudFormation.
- Locally sync and save your project in the Infrastructure Composer console, a feature that automatically syncs and saves data to your local machine, is not supported.
- Lambda-related cards (**Lambda Function** and **Lambda Layer**) require code builds and packaging solutions that are not available in this mode.

> **Note**
>
> These cards and local sync can be used in the Infrastructure Composer Console or the AWS Toolkit for Visual Studio Code.

When you open Infrastructure Composer from the CloudFormation console, Infrastructure Composer opens in CloudFormation console mode. In this mode, you can use Infrastructure Composer to visualize, create, and update your templates.

How to access Infrastructure Composer in CloudFormation console mode

Infrastructure Composer in CloudFormation console mode is an upgrade from CloudFormation Designer. We recommend using Infrastructure Composer to visualize your CloudFormation templates. You can also use this tool to create and edit CloudFormation templates.

1. Go to the Cloudformation console and log in.
2. Select **Infrastructure Composer** from the left-side navigation menu. This will take you to Infrastructure Composer in CloudFormation console mode.

> **Note**
>
> For information on using Infrastructure Composer in CloudFormation console mode, see Using Infrastructure Composer in CloudFormation console mode.

Visualize a deployment in Infrastructure Composer in CloudFormation console mode

Follow the instructions in this topic to visualize a deployed CloudFormation stack/ Infrastructure Composer template.

1. Go to the CloudFormation console and log in.
2. Select the stack you want to edit.
3. Select the **Template** tab.
4. Select **Infrastructure Composer**.

Infrastructure Composer will visualize your stack/template. Changes can be made here as well.

Create a new template in Infrastructure Composer in CloudFormation console mode

Follow the instructions in this topic to create a new template.

1. Go to the CloudFormation console and log in.
2. Select **Infrastructure Composer** from the left-side navigation menu. This will open Infrastructure Composer in CloudFormation console mode.
3. Drag, drop, configure, and connect the resources (cards) you need from the **Resources** pallete.

   > **Note**
   >
   > See How to compose for details on using Infrastructure Composer, and note that Lambda-related cards (**Lambda Function** and **Lambda Layer**) require code builds and packaging solutions that are not available in Infrastructure Composer in CloudFormation console mode. These cards can be used in the Infrastructure Composer console or the AWS Toolkit for Visual Studio Code. For information on using these tools, refer to Where you can use Infrastructure Composer.

4. Double click cards to use the **Resource properties** panel to specify how cards are configured.
5. Connect your cards to specify your application's event-driven workflow.
6. Select **Template** to view and edit your infrastructure code. Changes are automatically synced with your canvas view.
7. Once your template is ready to be exported into a stack, select **Create template**.
8. Select the **Confirm and export to CloudFormation** button. This will take you back to the create stack workflow with a message confirming your template was successfully imported.

   > **Note**
   >
   > Only templates with resources in them can be exported.

9. In the **Create stack** workflow, select **Next**.
10. Provide a stack name, review any listed parameters, and select **Next**.

> **Note**
>
> The stack name must start with a letter and contain only letters, numbers, dashes.

11. Select **Next** after providing the following information:

    - Tags associated with the stack
    - Stack permissions
    - The stack's failure options

> **Note**
>
> For guidance on managing stacks, see CloudFormation best practices in the *CloudFormation User Guide*.

12. Confirm your stack details are correct, check acknowledgements at the bottom of the page, and select the **Submit** button.

CloudFormation will begin creating the stack based on the data in your template.

Update an existing stack in Infrastructure Composer in CloudFormation console mode

Follow the instructions in this topic to update an existing CloudFormation stack.

> **Note**
>
> If your file is saved locally, we recommend using AWS Toolkit for Visual Studio Code.

1. Go to the CloudFormation console and log in.
2. Select the stack you want to edit.
3. Select the **Update** button. Doing this will take you to the update stack wizard.
4. On the right, select **Edit in Infrastructure Composer**.

5. Select the button below that's labeled **Edit in Infrastructure Composer**. This will take you to Infrastructure Composer in CloudFormation console mode.
6. Here, you can drag, drop, configure, and connect resources (cards) from the **Resources** pallete.

> **Note**
>
> See How to compose for details on using Infrastructure Composer, and note that Lambda-related cards (**Lambda Function** and **Lambda Layer**) require code builds and packaging solutions that are not available in Infrastructure Composer in CloudFormation console mode. These cards can be used in the Infrastructure Composer console or the AWS Toolkit for Visual Studio Code. For information on using these tools, refer to Where you can use Infrastructure Composer.

7. When you are ready to export changes to CloudFormation, select **Update template**.
8. Select **Confirm and continue to CloudFormation**. This will take you back to the **Update stack** workflow with a message confirming your template was successfully imported.

> **Note**
>
> Only templates with resources in them can be exported.

9. In the **Update stack** workflow, select **Next**.
10. Review any listed parameters and select **Next**.
11. Select **Next** after providing the following information:

    - Tags associated with the stack
    - Stack permissions
    - The stack's failure options

> **Note**
>
> For guidance on managing stacks, see CloudFormation best practices in the *CloudFormation User Guide*.

12. Confirm your stack details are correct, check acknowledgements at the bottom of the page, and select the **Submit** button.

CloudFormation will begin updating the stack based on the updates you made in your template.

Using Infrastructure Composer from the AWS Toolkit for Visual Studio Code

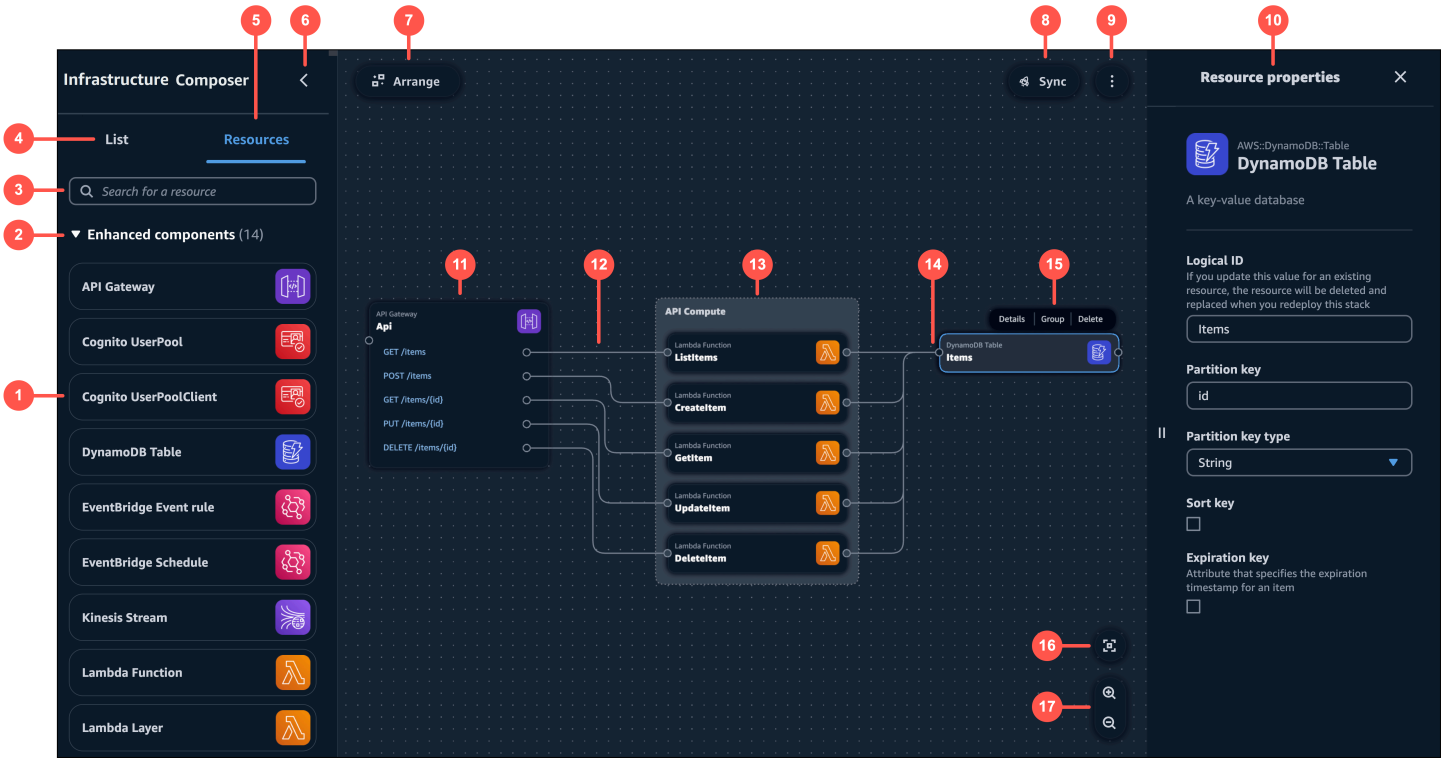
This section describes how you can use AWS Infrastructure Composer from the AWS Toolkit for Visual Studio Code. This includes a visual overview of Infrastructure Composer from the AWS Toolkit for Visual Studio Code. It also includes instructions showing how you can access this experience and sync your project from VS Code to the AWS cloud. To sync, you use the **sam sync** command from the AWS SAM CLI. This section also provides guidance on using Amazon Q while in Infrastructure Composer from the AWS Toolkit for Visual Studio Code.

For additional guidance on using Infrastructure Composer from the AWS Toolkit for Visual Studio Code, refer to How to compose. The content in this section applies to this experience, as well as the Infrastructure Composer console experience.

Topics

Visual overview of Infrastructure Composer from the AWS Toolkit for Visual Studio Code

Infrastructure Composer's visual designer in the AWS Toolkit for Visual Studio Code includes a visual canvas, which includes components that are numbered in the following image and listed below.



1. **Resource palette** – Displays cards that you can design with.
2. **Card categories** – Cards are organized by categories unique to Infrastructure 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.

Access Infrastructure Composer from the AWS Toolkit for Visual Studio Code

Follow the instructions in this topic to access Infrastructure Composer from the AWS Toolkit for Visual Studio Code.

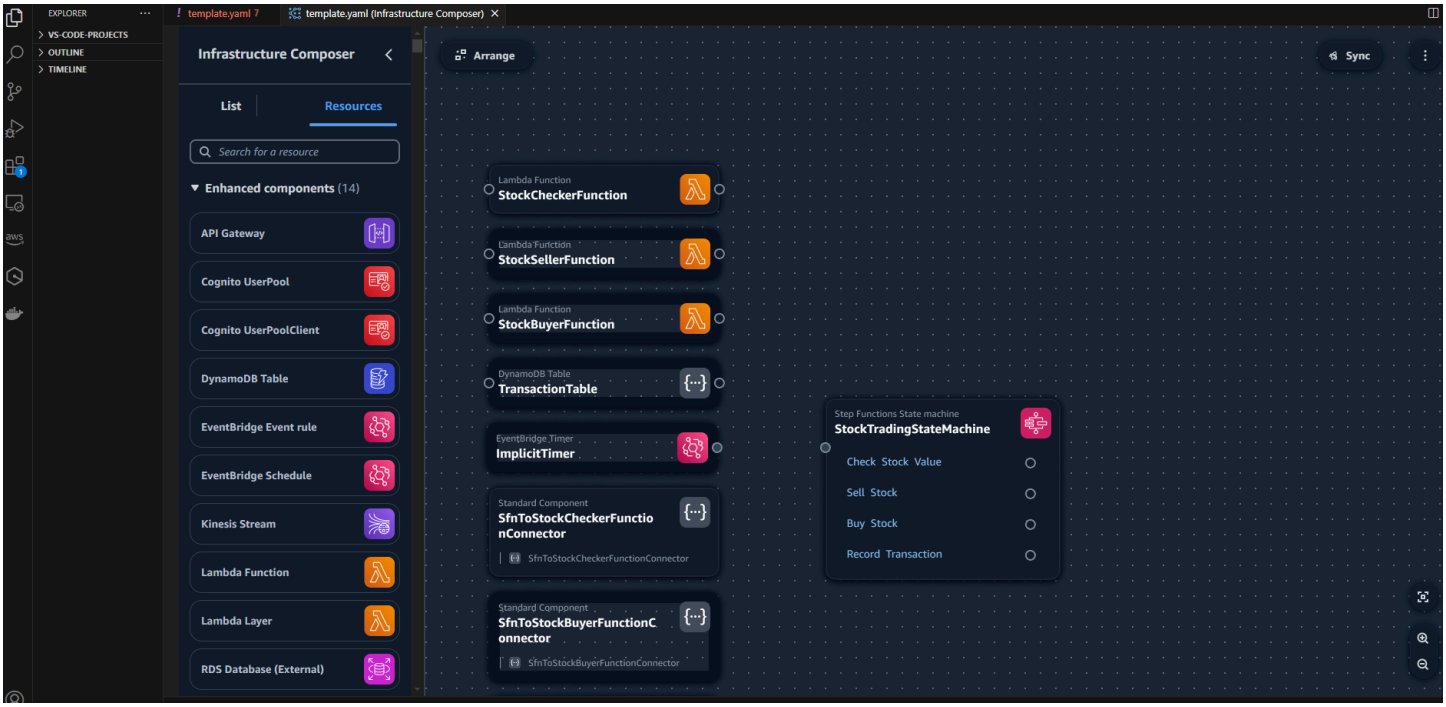
> **Note**
>
> Before you can access Infrastructure Composer from the AWS Toolkit for Visual Studio Code, you must first download and install the Toolkit for VS Code. For instructions, see Downloading the Toolkit for VS Code.

To access Infrastructure Composer from the Toolkit for VS Code

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

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

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



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

Sync Infrastructure Composer to deploy to the AWS Cloud

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

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

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*.

Set up

To use the **sync** feature in Infrastructure 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 Infrastructure 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*.

Sync and deploy your application

To sync your application to the AWS Cloud

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

   - **AWS Region** – The region to sync your application to.
   - **CloudFormation stack name** – The name of your 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.

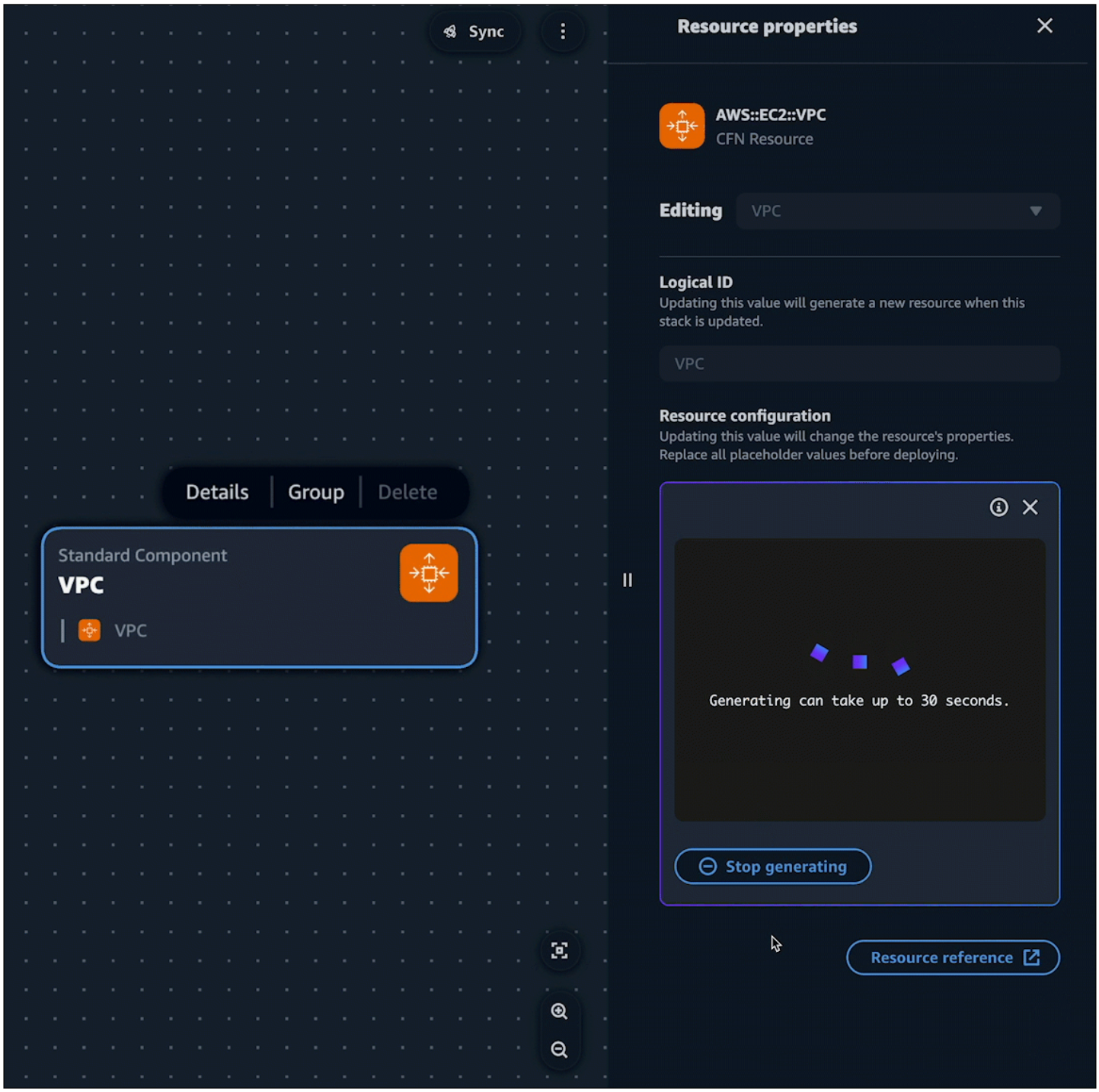
Infrastructure Composer will initiate the AWS SAM CLI `sam sync` command and open a terminal window in your IDE to output its progress.

Using AWS Infrastructure Composer with Amazon Q Developer

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

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

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



Standard resource and **standard component** cards can represent an CloudFormation resource or a collection of CloudFormation resources. To learn more, see Configure and modify cards in Infrastructure Composer.

Setting up

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

Using Amazon Q Developer in Infrastructure Composer

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

To use Amazon Q in Infrastructure 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. Amazon Q 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 Amazon Q, see What is Amazon Q? in the *Amazon Q Developer User Guide*.

How to compose in AWS Infrastructure Composer

This section covers the basics of using Infrastructure Composer from the Infrastructure Composer console, CloudFormation console mode, and the AWS Toolkit for Visual Studio Code. More specifically, the topics in this section provide key details on how to compose an application with Infrastructure Composer, and includes details on additional features and shortcuts. There are a few variations in functionality between console and VS Code experiences, and the topics in this section identifies and describes these variations where they occur.

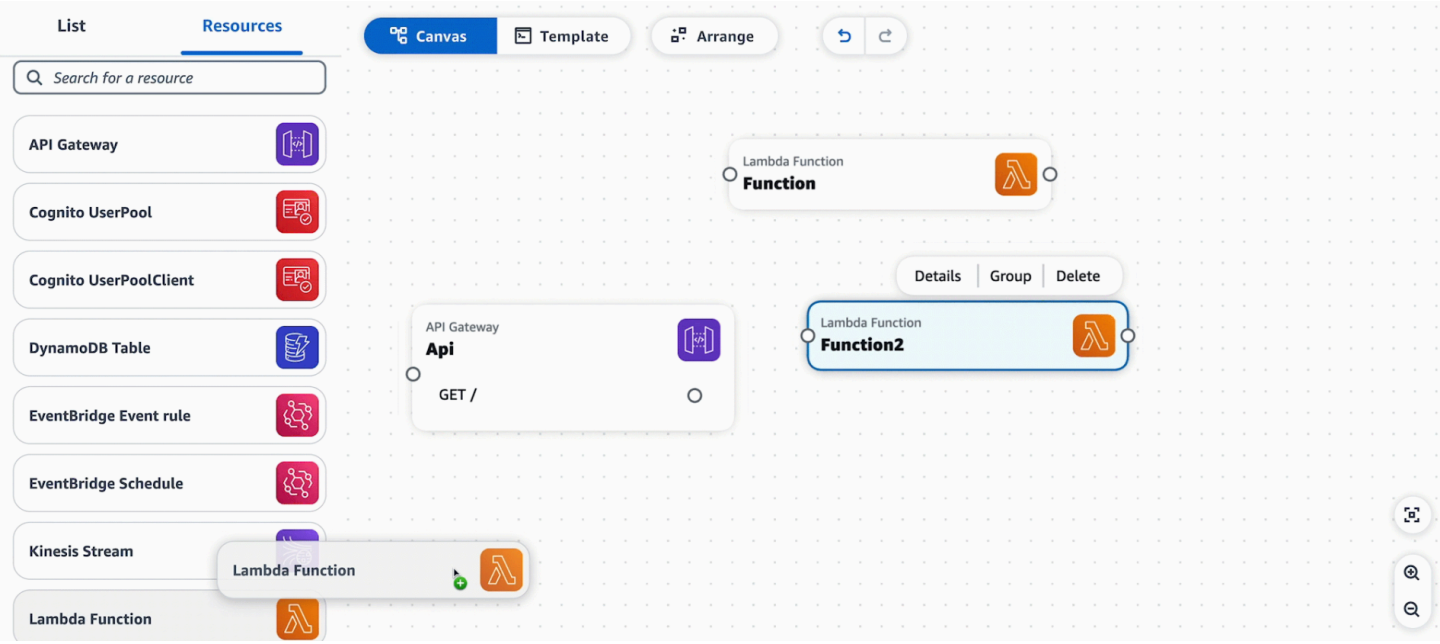
After composing your application, you will be ready to review Deploy your Infrastructure Composer serverless application to the AWS Cloud for information on deploying your application.

Topics

Place cards on Infrastructure Composer's visual canvas

This section describes how you select and drag Infrastructure Composer cards in its visual canvas. Before starting, identify what resources your application needs and how they need to interact. For tips on doing this, see Build your first application with Infrastructure Composer.

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



You can choose from two types of cards: Enhanced component cards and Standard IaC resource cards.

After placing your cards on the visual canvas, you'll be ready to group, connect, arrange, and configure your cards. See the following topics for information on doing this:

- Group cards together on Infrastructure Composer's visual canvas
- Connect cards on Infrastructure Composer's visual canvas
- Arrange cards on Infrastructure Composer's visual canvas
- Configure and modify cards in Infrastructure Composer

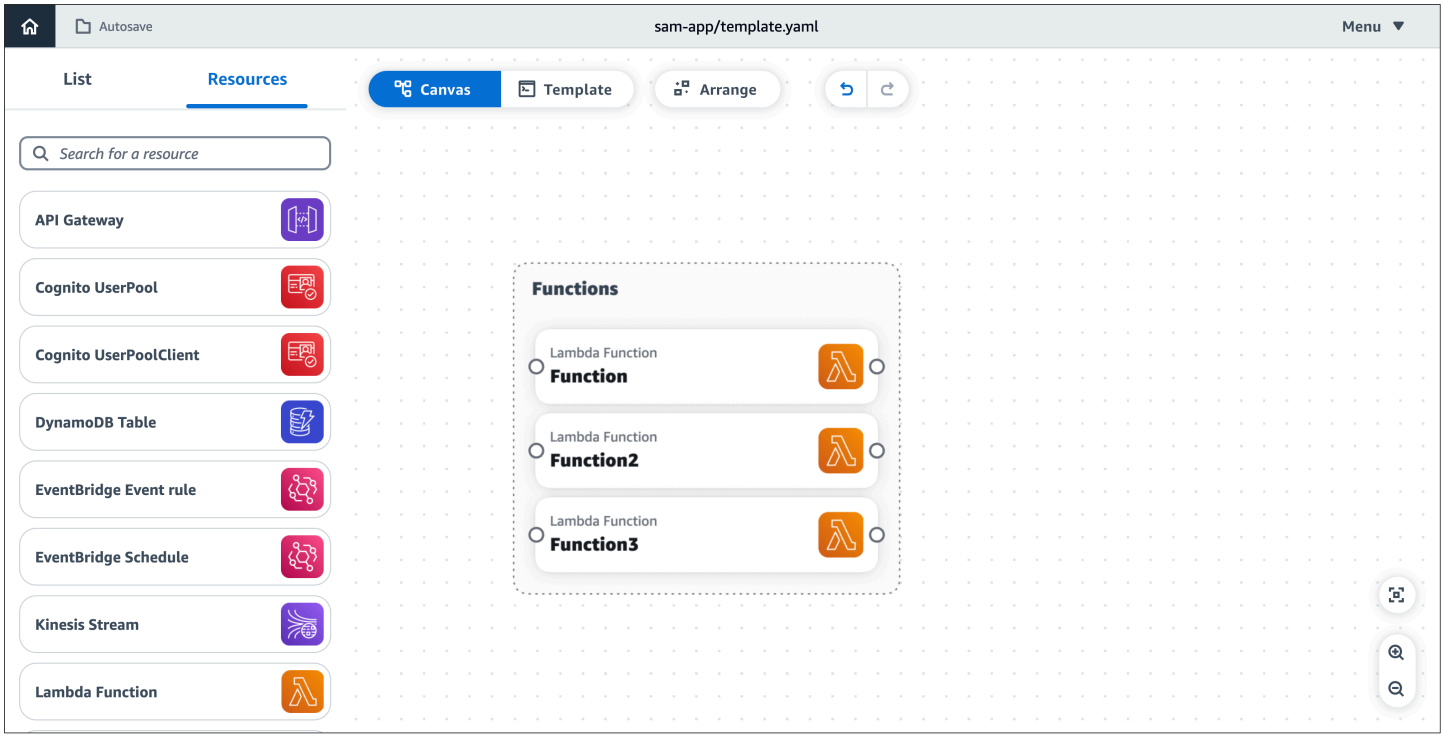
Group cards together on Infrastructure Composer's visual canvas

This topic contains details on grouping enhanced component cards and standard component cards. Grouping cards helps you categorize and organize your resources without needing to think about the code or markup you need write.

Grouping enhanced component cards

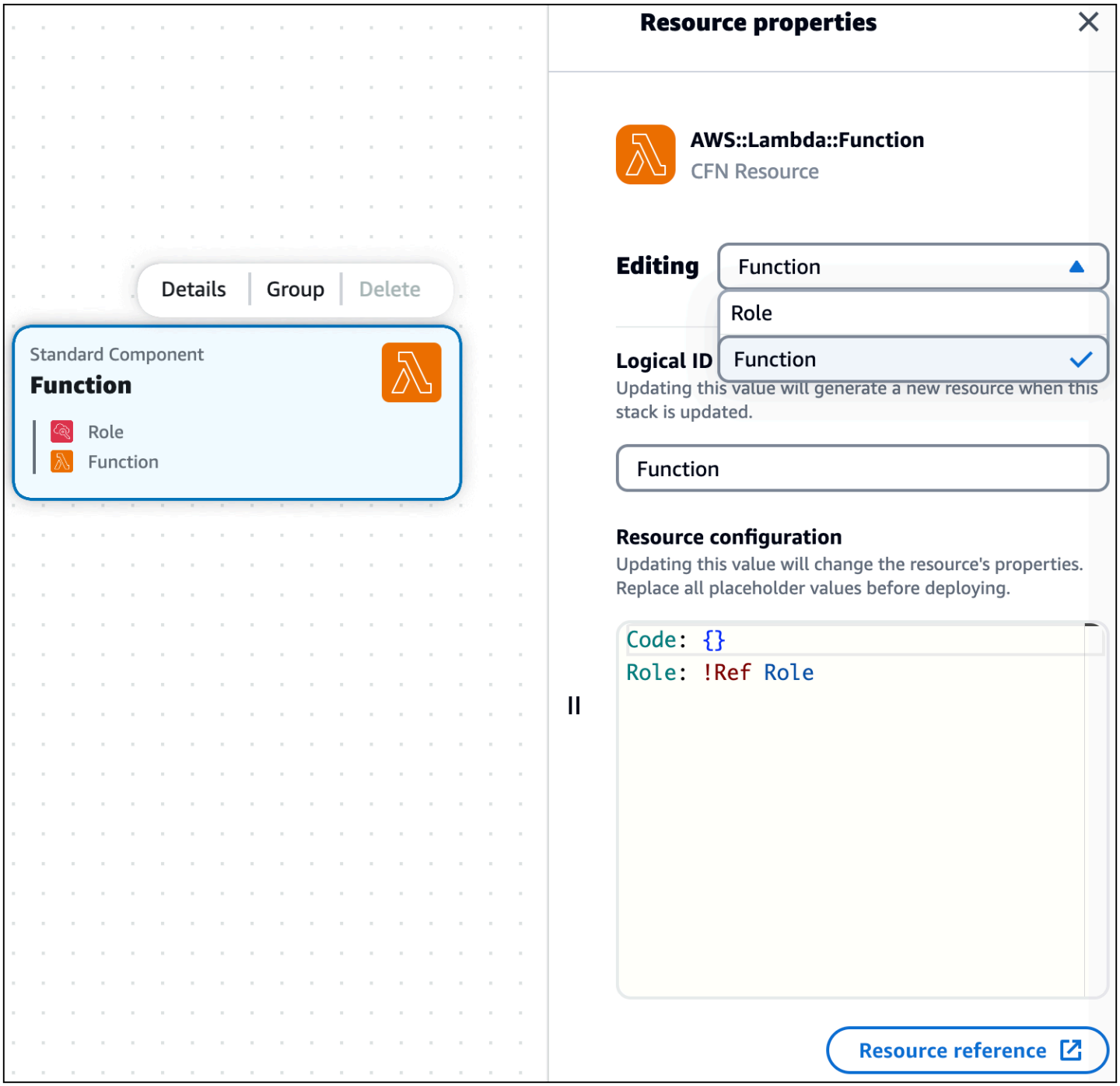
There are two ways to group enhanced component cards together:

- While pressing **Shift**, select cards to group. Then, choose **Group** from the resource actions menu.
- select a card you want in a group. From the menu that appears, select **Group**. This will create a group that you can drag and drop other cards into.



Grouping a standard component card into another

The following example shows one way a standard component card can be grouped into another card from the **Resource properties** panel:



In the **Resource configuration** field on the **Resource properties** panel, the `Role` has been referenced in the Lambda function. This results in the **Role** card being grouped into the **Function** card on the canvas.

Connect cards on Infrastructure Composer's visual canvas

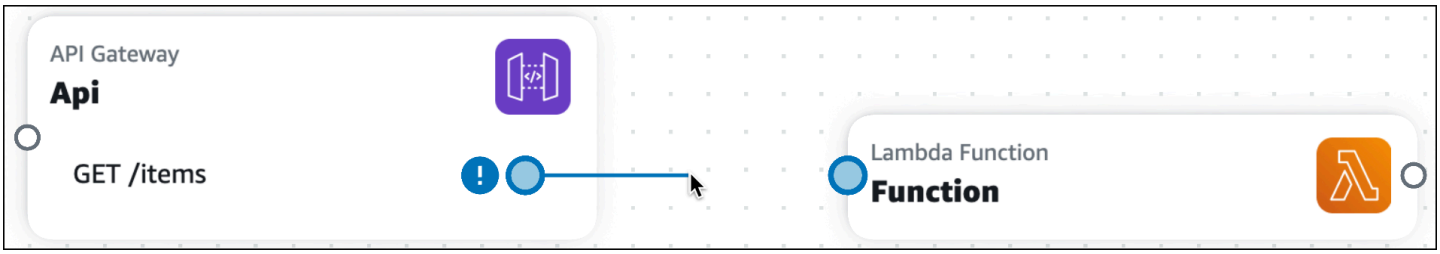
Use this topic to understand how to connect cards in Infrastructure Composer. This section includes details on connecting enhanced component cards and standard component cards. It also provides a few examples that illustrate the different ways cards can be connected.

Connecting enhanced component cards

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

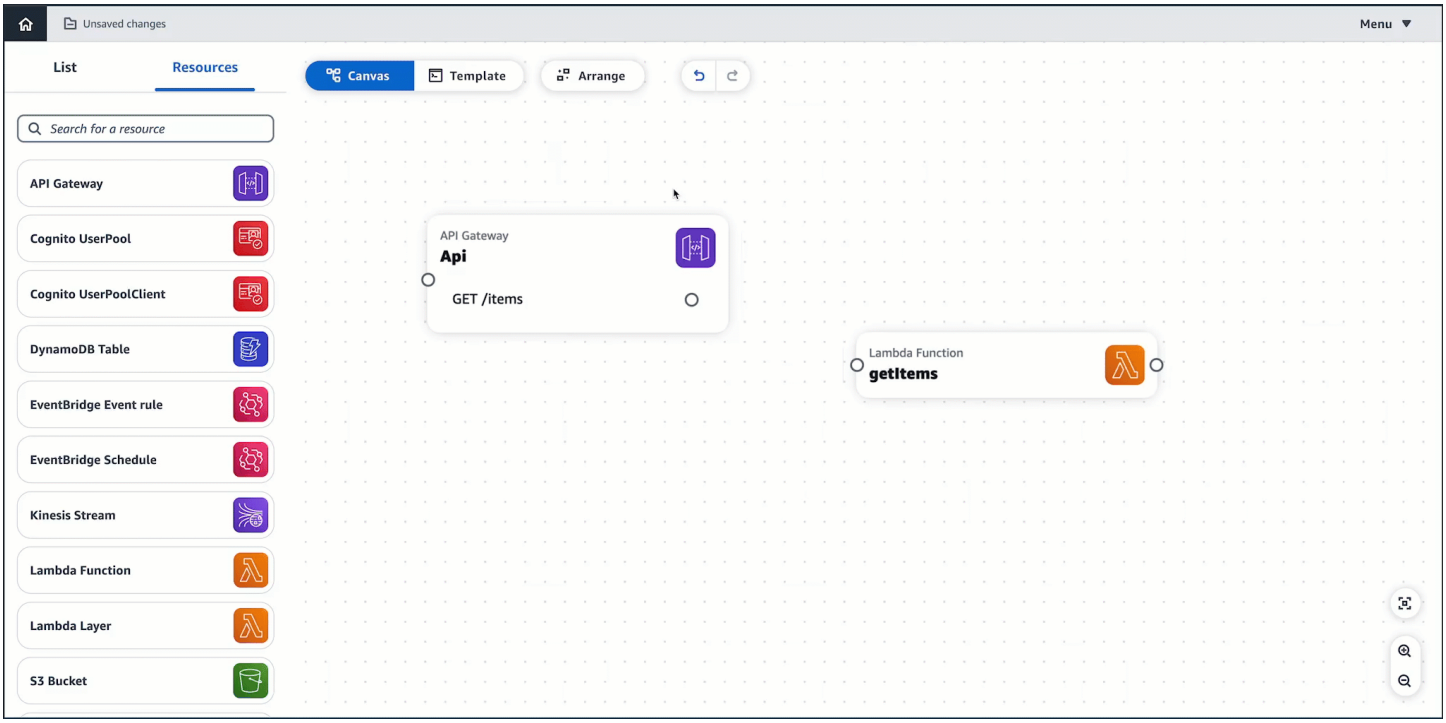
- A port on the right side of a card indicates an opportunity for the card to invoke another card.
- A port on the left side of a card indicates an opportunity for the card to be invoked by another card.

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 create a connection, a message will display, letting you know if the connection was successfully made. Select the message to see what Infrastructure Composer changed to provision a connection. If the connection was unsuccessful, you can select **Template view** to manually update your infrastructure code to provision the connection.

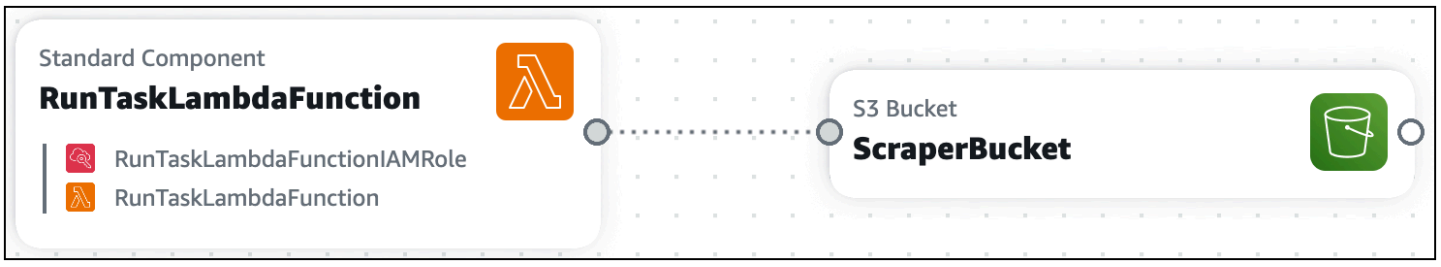
- When successful, click on the message to view the **Change inspector**. Here, you can see what Infrastructure Composer modified to provision your connection.
- When unsuccessful, a message will display. You can select the **Template view** and manually update your infrastructure code to provision the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



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

Connecting standard component cards (Standard IaC resource cards)

Standard IaC resource cards do not include ports to create connections with other resources. During card configuration, you specify event-driven relationships in the template of your application, Infrastructure 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:



The following example shows how a Lambda function can be connected with an Amazon API Gateway rest API:

```
AWSTemplateFormatVersion: '2010-09-09'
```

```
Resources:
  MyApi:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Name: MyApi

  ApiGatewayMethod:
    Type: 'AWS::ApiGateway::Method'
    Properties:
      HttpMethod: POST  # Specify the HTTP method you want to use (e.g., GET, POST,
 PUT, DELETE)
      ResourceId: !GetAtt MyApi.RootResourceId
      RestApiId: !Ref MyApi
      AuthorizationType: NONE
      Integration:
        Type: AWS_PROXY
        IntegrationHttpMethod: POST
        Uri: !Sub
          - arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/
${LambdaFunctionArn}/invocations
          - { LambdaFunctionArn: !GetAtt MyLambdaFunction.Arn }
      MethodResponses:
        - StatusCode: 200

  MyLambdaFunction:
    Type: 'AWS::Lambda::Function'
    Properties:
      Handler: index.handler
      Role: !GetAtt LambdaExecutionRole.Arn
      Runtime: nodejs14.x
      Code:
        S3Bucket: your-bucket-name
        S3Key: your-lambda-zip-file.zip

  LambdaExecutionRole:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
            Action: 'sts:AssumeRole'
```

```
      Policies:
        - PolicyName: LambdaExecutionPolicy
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Action:
                  - 'logs:CreateLogGroup'
                  - 'logs:CreateLogStream'
                  - 'logs:PutLogEvents'
                Resource: 'arn:aws:logs:*:*:*'
              - Effect: Allow
                Action:
                  - 'lambda:InvokeFunction'
                Resource: !GetAtt MyLambdaFunction.Arn
```

In the above example, the snippet of code listed in `ApiGatewayMethod:` under `Integration:` specifies the event-driven relationship that connects the two cards.

Examples for connecting cards in Infrastructure Composer

Use the examples in this section to understand how cards can be connected in Infrastructure Composer.

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 Infrastructure 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 Infrastructure 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
    ...
```

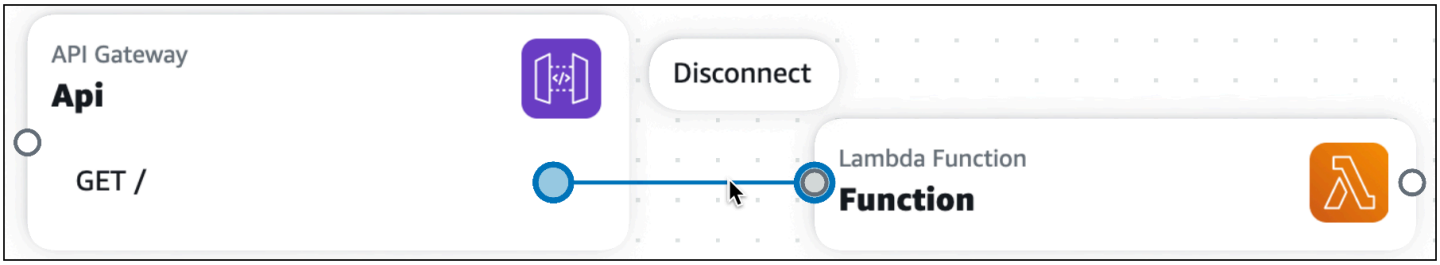
```
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      ...
      Environment:
        Variables:
          BUCKET_NAME: !Ref MyBucket
          BUCKET_ARN: !GetAtt MyBucket.Arn
      Policies:
        - S3CrudPolicy:
          BucketName: !Ref MyBucket
```

Disconnect cards in Infrastructure Composer

In Infrastructure Composer, you connect and disconnect AWS resources using *enhanced component cards* and *standard component cards*. This section describes how to disconnect both types of cards.

Enhanced component cards

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



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

Standard component cards

Standard component cards do not include ports to create connections with other resources. During card configuration, you specify event-driven relationships in the template of your application, Infrastructure Composer will automatically detect these connections and visualize them with a dotted line between your cards. To disconnect a standard component card, remove the event-driven relationship in the template of your application.

The following example shows a Lambda function that is connected with an Amazon API Gateway rest API:

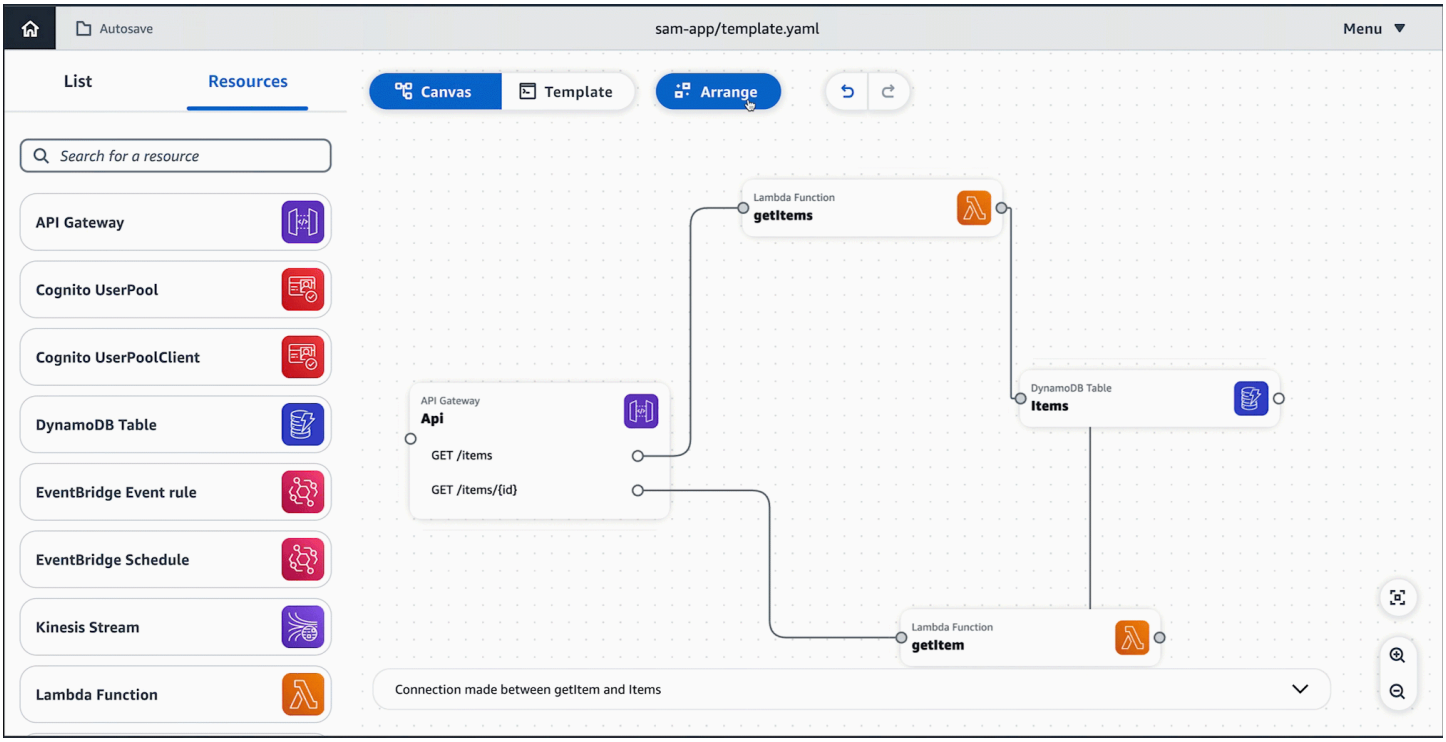
```
AWSTemplateFormatVersion: '2010-09-09'
Resources:
  MyApi:
    Type: 'AWS::ApiGateway::RestApi'
    Properties:
      Name: MyApi

  ApiGatewayMethod:
    Type: 'AWS::ApiGateway::Method'
    Properties:
      HttpMethod: POST  # Specify the HTTP method you want to use (e.g., GET, POST,
 PUT, DELETE)
      ResourceId: !GetAtt MyApi.RootResourceId
      RestApiId: !Ref MyApi
      AuthorizationType: NONE
      Integration:
        Type: AWS_PROXY
        IntegrationHttpMethod: POST
        Uri: !Sub
          - arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/
${LambdaFunctionArn}/invocations
          - { LambdaFunctionArn: !GetAtt MyLambdaFunction.Arn }
      MethodResponses:
        - StatusCode: 200

  MyLambdaFunction:
    Type: 'AWS::Lambda::Function'
    Properties:
      Handler: index.handler
      Role: !GetAtt LambdaExecutionRole.Arn
      Runtime: nodejs14.x
      Code:
        S3Bucket: your-bucket-name
        S3Key: your-lambda-zip-file.zip

  LambdaExecutionRole:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
```

```
            Service: lambda.amazonaws.com
          Action: 'sts:AssumeRole'
    Policies:
      - PolicyName: LambdaExecutionPolicy
        PolicyDocument:
          Version: '2012-10-17'
          Statement:
            - Effect: Allow
              Action:
                - 'logs:CreateLogGroup'
                - 'logs:CreateLogStream'
                - 'logs:PutLogEvents'
              Resource: 'arn:aws:logs:*:*:*'
            - Effect: Allow
              Action:
                - 'lambda:InvokeFunction'
              Resource: !GetAtt MyLambdaFunction.Arn
```

To remove the connection betweent the two cards, remove references to `MyLambdaFunction` listed in `ApiGatewayMethod:` under `Integration`.

Arrange cards on Infrastructure Composer's visual canvas

Select **Arrange** to visually arrange and organize cards on the canvas. Using the **Arrange** button is particularly useful when there many cards and connections on the canvas.



Configure and modify cards in Infrastructure Composer

In Infrastructure Composer, cards represent resources that you use to design your application architecture. When you configure a card in Infrastructure Composer, you define the details of the resources in your application. This includes details like a card's **Logical ID** and **Partition key**. The way this information is defined varies between **Enhanced component cards** and **Standard cards**.

An **Enhanced component card** is A collection of CloudFormation resources that have been combined into a single curated card that enhances ease of use, functionality, and are designed for a wide variety of use cases. A **Standard IaC resource card** represents a single AWS CloudFormation resource. Each standard IaC resource card, once dragged onto the canvas, is labeled **Standard component**.

This topic provides details on configuring **Enhanced component cards** and **Standard component cards**.

> **Note**
>
> This topic applies to using cards from the Infrastructure Composer Console, the AWS Toolkit for Visual Studio Code extension, and while in Infrastructure Composer in CloudFormation console mode. Lambda-related cards (**Lambda Function** and **Lambda**

Layer) require code builds and packaging solutions that are not available in Infrastructure Composer in CloudFormation console mode. For more information, see Using Infrastructure Composer in CloudFormation console mode.

Topics

Enhanced component cards in Infrastructure Composer

To configure enhanced component cards, Infrastructure Composer provides a form in the **Resource properties** panel. This form is curated uniquely to guide you through configuring each enhanced component card. As you fill out the form, Infrastructure Composer modifies your infrastructure code.

Some enhanced component cards do have additional features. This section reviews the basics of using enhanced component cards and offers details on cards with additional features.

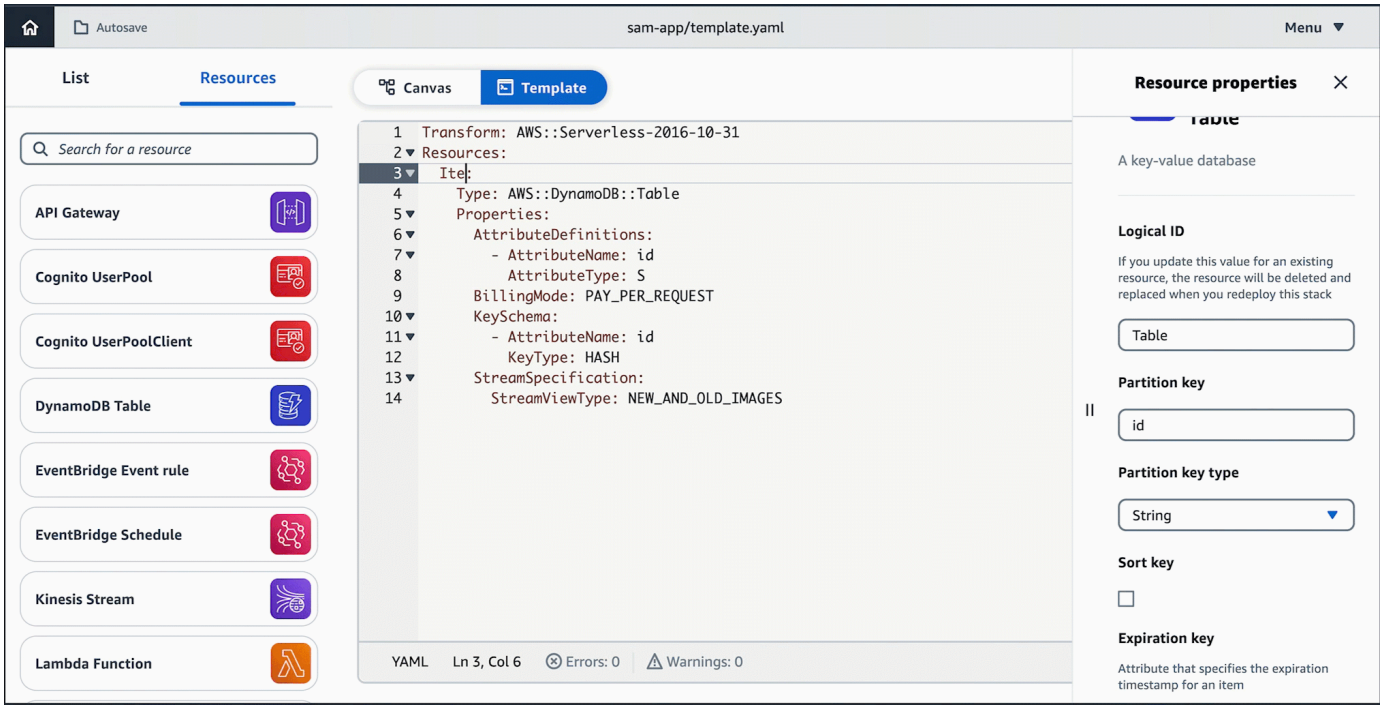
For more information on enhanced component cards, see Enhanced component cards in Infrastructure Composer and Enhanced component cards in Infrastructure Composer

Procedure

The **Resource properties** panel streamlines configuration and adds guiderails that simplifies card configuration. To use this panel, perform the following steps:

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

   The following image shows how this can be done:



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

AWS Infrastructure Composer features an integration with Amazon Relational Database Service (Amazon RDS). Using the **RDS Database (External)** enhanced component card in Infrastructure Composer, you can connect your application to Amazon RDS DB clusters, instances, and proxies that are defined on another CloudFormation or AWS Serverless Application Model (AWS SAM) template.

The **RDS Database (External)** enhanced component card represents Amazon RDS resources that are defined on another template. This includes:

- Amazon RDS DB cluster or instance that is defined on another template
- Amazon RDS DB proxy

The **RDS Database (External)** enhanced component card is available from the **Resources** palette.

To use this card, drag it onto the Infrastructure Composer canvas, configure it, and connect it to other resources.

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

Requirements

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

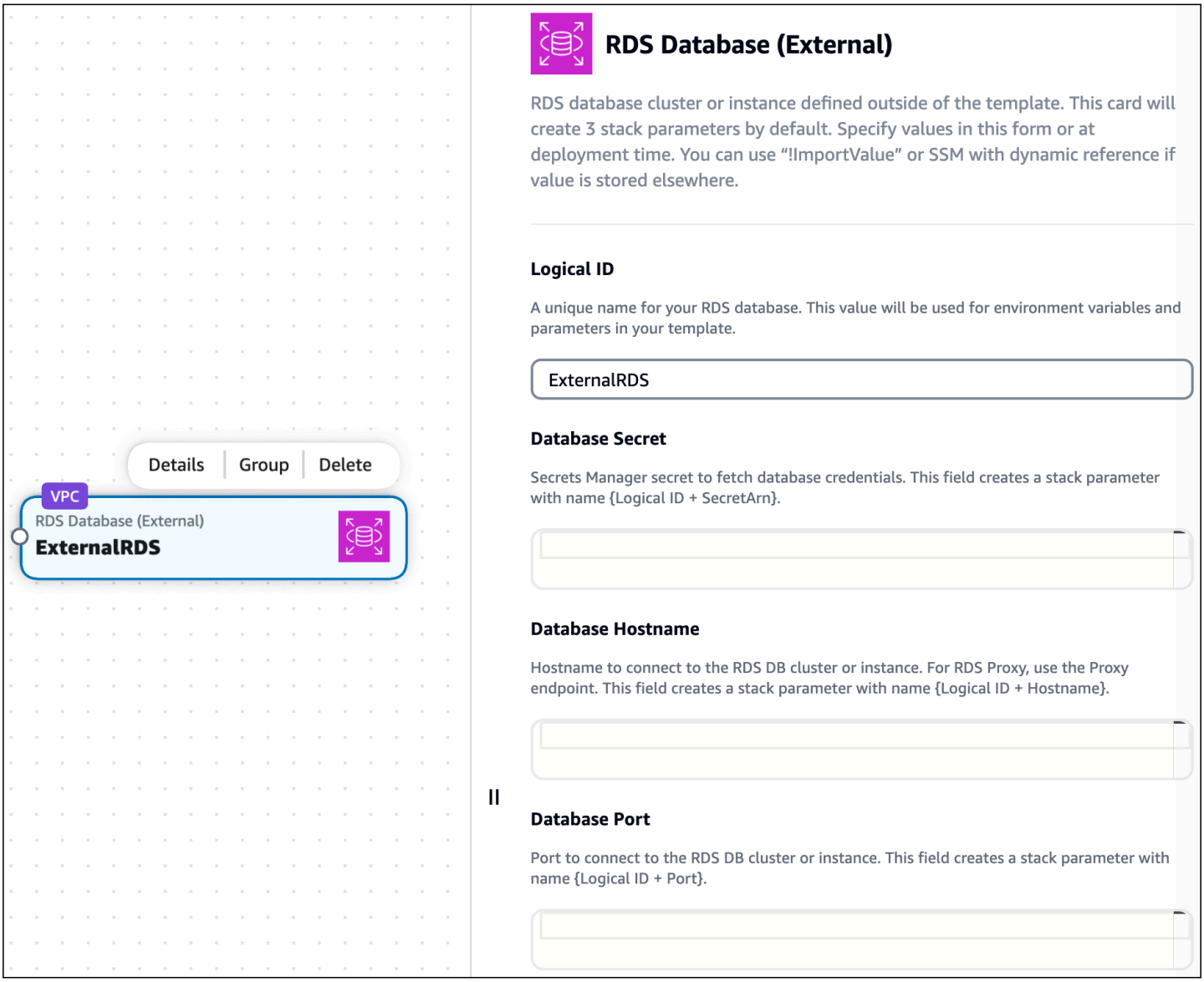
1. Your external Amazon RDS DB cluster, instance, or proxy must be using AWS Secrets Manager to manage the user password. To learn more, see Password management with Amazon RDS and AWS Secrets Manager in the *Amazon RDS User Guide*.
2. Your application in Infrastructure Composer must be a new project or must have been originally created in Infrastructure Composer.

Procedure

Step 1: Configure the external RDS Database card

From the **Resources** palette, drag an **RDS Database (external)** enhanced component card onto the canvas.

Select the card and choose **Details** or double-click on the card to bring up the **Resource properties** panel. The card's resource properties panel will appear:



You can configure the following here:

- **Logical ID** – A unique name for your external Amazon RDS DB cluster, instance, or proxy. This ID does not have to match the logical ID value of your external Amazon RDS DB resource.
- **Database secret** – An identifier for the AWS Secrets Manager secret that is associated with your Amazon RDS DB cluster, instance, or proxy. 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, instance, or proxy. This value is specified in the external template that defines your Amazon RDS resource. 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, instance, or proxy. 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, instance, or proxy. This value is specified in the external template that defines your Amazon RDS resource. 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, instance, or proxy. 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.

Infrastructure Composer will provision your template to facilitate this connection.

What Infrastructure Composer does to create your connection

When you complete the procedure listed above, Infrastructure Composer performs specific actions to connect your Lambda function to your database.

When specifying the external Amazon RDS DB cluster, instance, or proxy

When you drag an **RDS Database (external)** card onto the canvas, Infrastructure 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 CloudFormation template section that is used to store details about your template. Metadata that is specific to Infrastructure Composer is stored under the

`AWS::Composer::ExternalResources` metadata key. Here, Infrastructure Composer stores the values that you specify for your Amazon RDS DB cluster, instance, or proxy.

The Parameters section of an 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, Infrastructure Composer may store values here for your Amazon RDS DB cluster, instance, or proxy 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.

When connecting the Lambda function to your database

When you connect a **Lambda Function** card to the **RDS Database (external)** card, Infrastructure 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*.

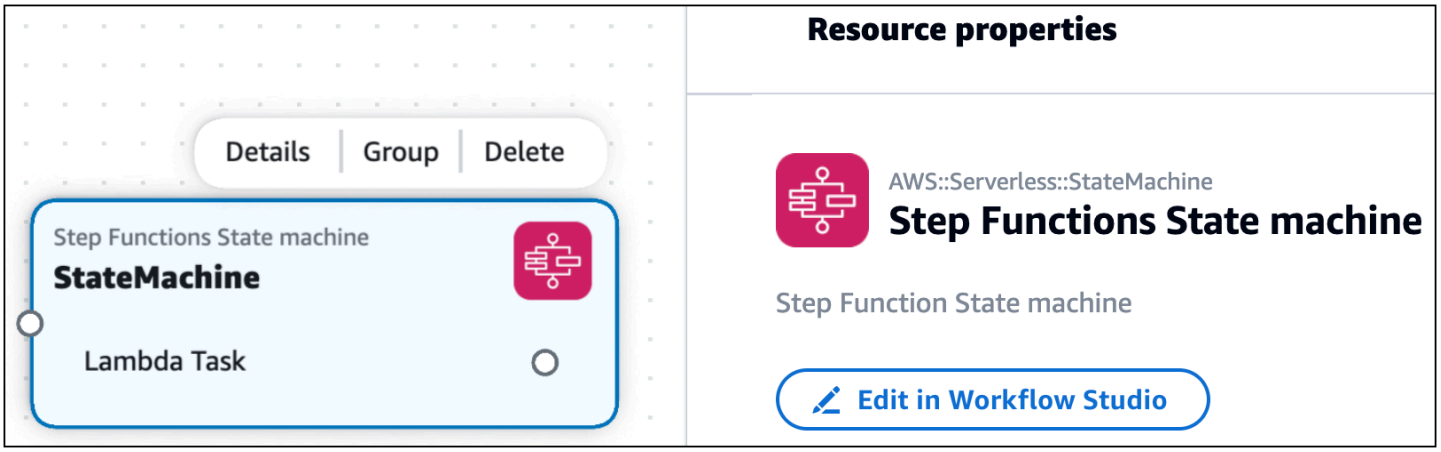
Policies provision permissions for your function. Here, Infrastructure Composer creates a policy to allow read access from your function to Secrets Manager to obtain your password for access to the Amazon RDS DB cluster, instance, or proxy.

Using AWS Infrastructure Composer with AWS Step Functions

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

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

The following image is of a Step Functions State machine card



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

Topics

- IAM policies
- Getting started with Step Functions Workflow Studio in Infrastructure Composer
- Using Step Functions Workflow Studio in Infrastructure Composer
- Learn more

IAM policies

When you connect tasks from your workflow to resources, Infrastructure 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.

Getting started with Step Functions Workflow Studio in Infrastructure Composer

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

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, Infrastructure Composer creates the following:

   - An `AWS::Serverless::StateMachine` resource that defines your state machine. By default, Infrastructure 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 Infrastructure 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 Infrastructure 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 Infrastructure Composer**.

   Infrastructure 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 Infrastructure 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. Infrastructure Composer will detect your changes and update your workflow in Infrastructure Composer.
   - For Infrastructure Composer from the Toolkit for VS Code, you can directly modify your template. Infrastructure Composer will detect your changes and update your workflow in Infrastructure Composer.

To import existing workflows

You can import workflows from applications that are defined using AWS Serverless Application Model (AWS SAM) templates. Use any state machine defined with the `AWS::Serverless::StateMachine` resource type, and it 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

Infrastructure 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 in the Infrastructure Composer console. To save changes that you make within Infrastructure 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 in the Infrastructure Composer console.

Infrastructure Composer from the Toolkit for VS Code

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

DefinitionUri property

Infrastructure 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 in the Infrastructure Composer console.

Infrastructure Composer from the Toolkit for VS Code

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

Using Step Functions Workflow Studio in Infrastructure Composer

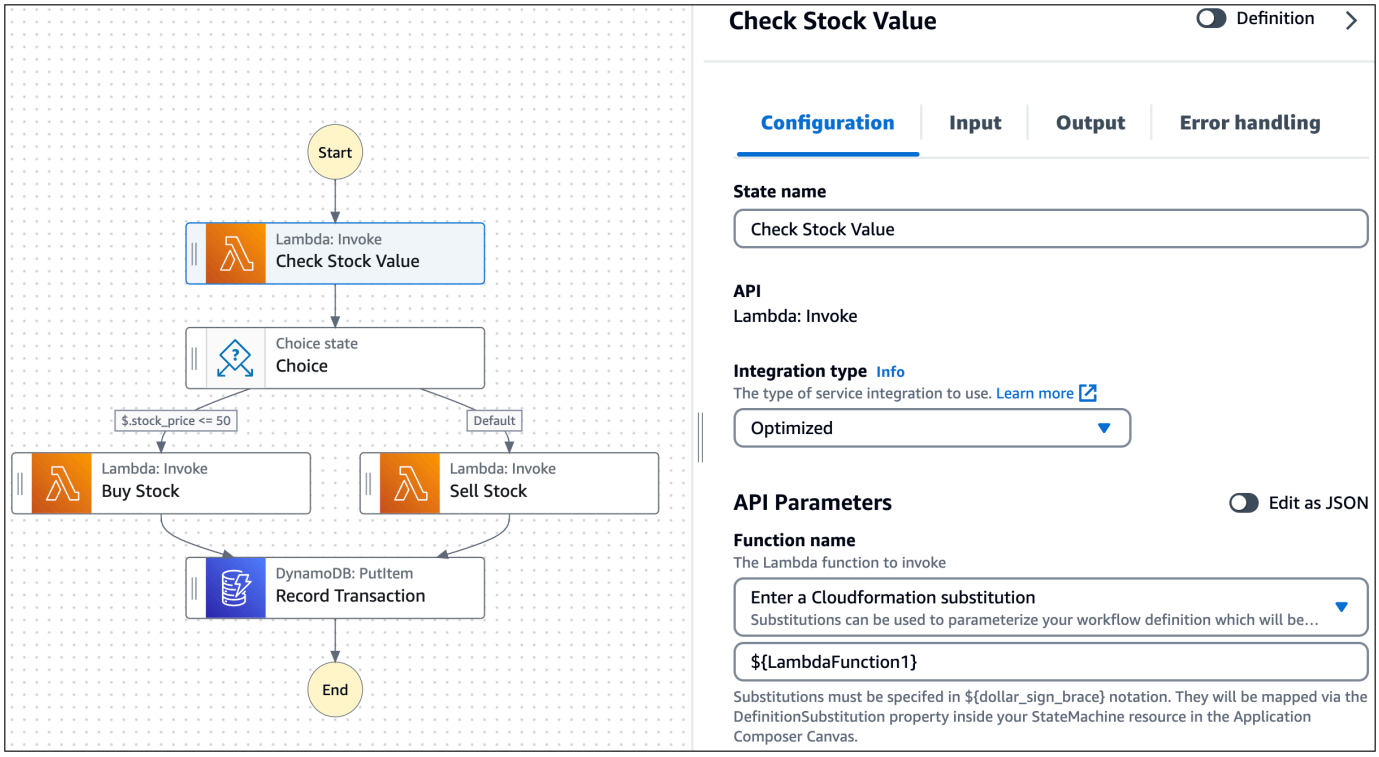
Build workflows

Infrastructure 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 Infrastructure Composer canvas.

To specify a definition substitution in Workflow Studio

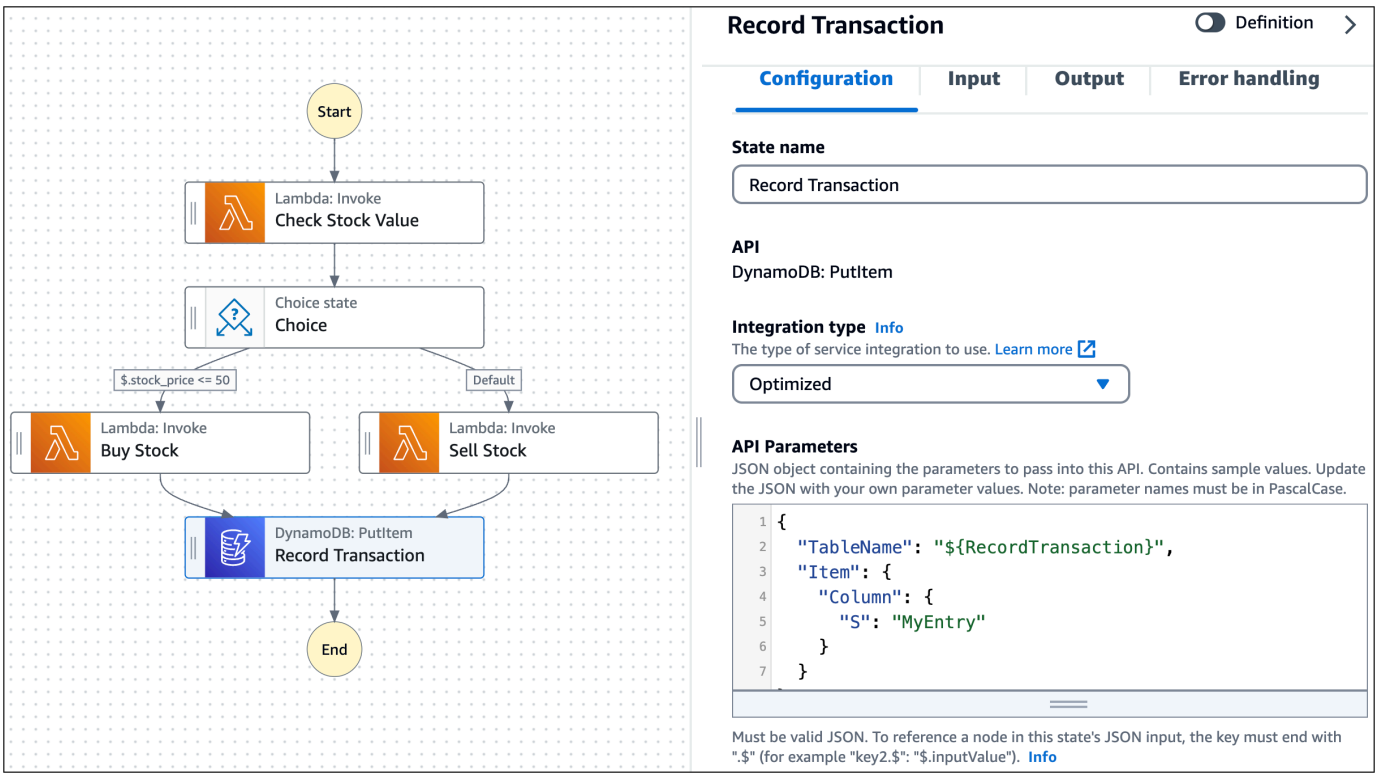
1. Open the **Configuration** tab of the task and locate the **API Parameters** field.



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

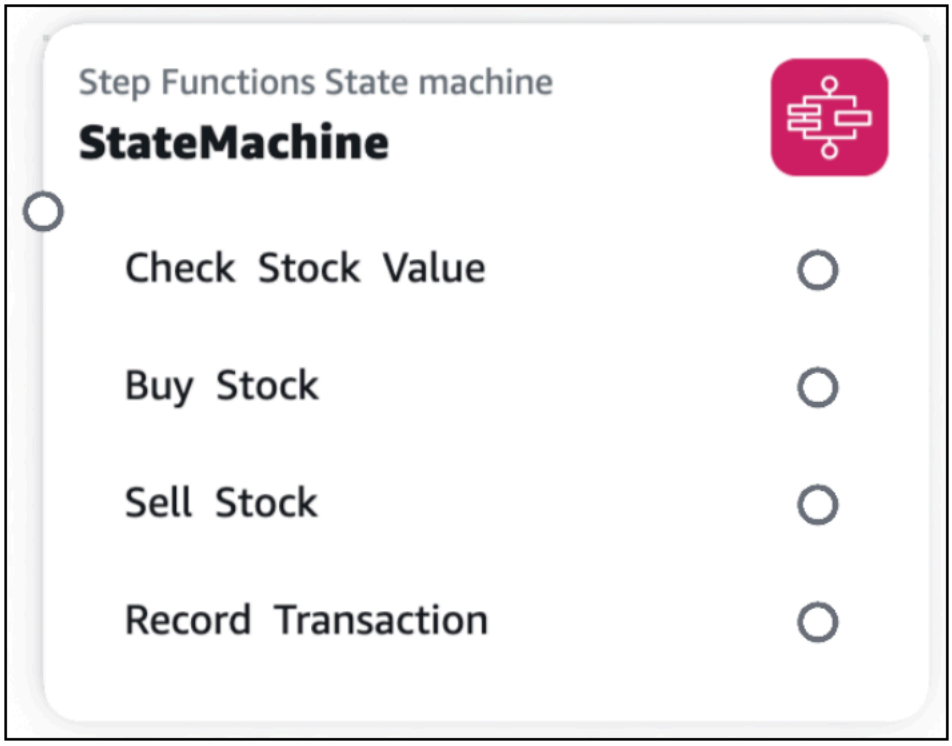
   For tasks that connect to the same resource, specify the same definition substitution for each task. To use an existing definition substitution, choose **Select a CloudFormation substitution** and select the substitution to use.

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 Infrastructure Composer**.

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



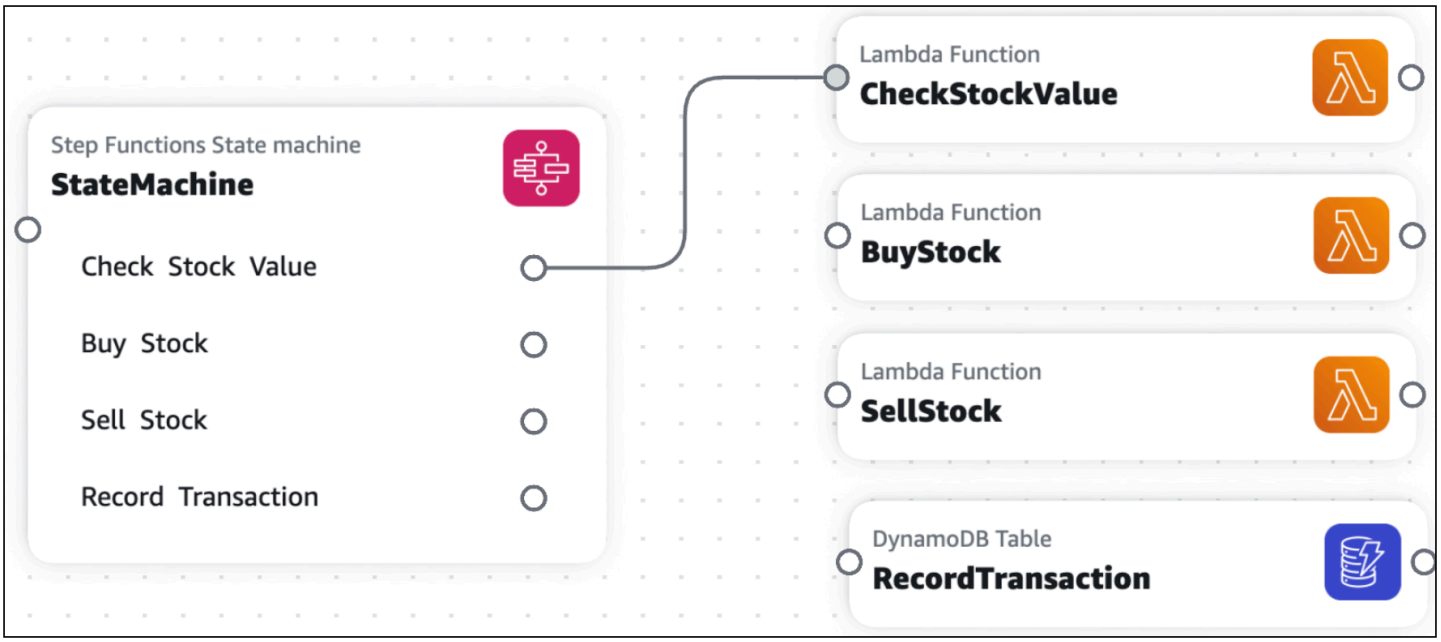
Connect resources to workflow tasks

You can create connections in Infrastructure Composer between supported workflow tasks and supported Infrastructure 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 Infrastructure Composer cards** – Enhanced component cards are supported. To learn more about cards in Infrastructure Composer, see Configure and modify cards in Infrastructure 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.



Infrastructure 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:
    ...
```

Working with external files

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

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

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**.

Infrastructure 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 Infrastructure Composer, see the following:

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

Standard cards in Infrastructure Composer

All CloudFormation resources are available to use as **standard IaC resource cards** from the **Resources** palette. After being dragged onto the visual canvas, a **standard IaC resource card** becomes a **standard component card**. This simply means the card is one or more standard IaC resources. For further examples and details, see the topics in this section.

You can modify your infrastructure code through the **Template** view and through the **Resource properties** window. For example, 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 CloudFormation resource type.
- Required or commonly used properties.
- The required type of the value to provide for each property.

> **Note**
>
> You can use Amazon Q to generate infrastructure code suggestions for standard resource cards. To learn more, see Using AWS Infrastructure Composer with Amazon Q Developer.

Procedure

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

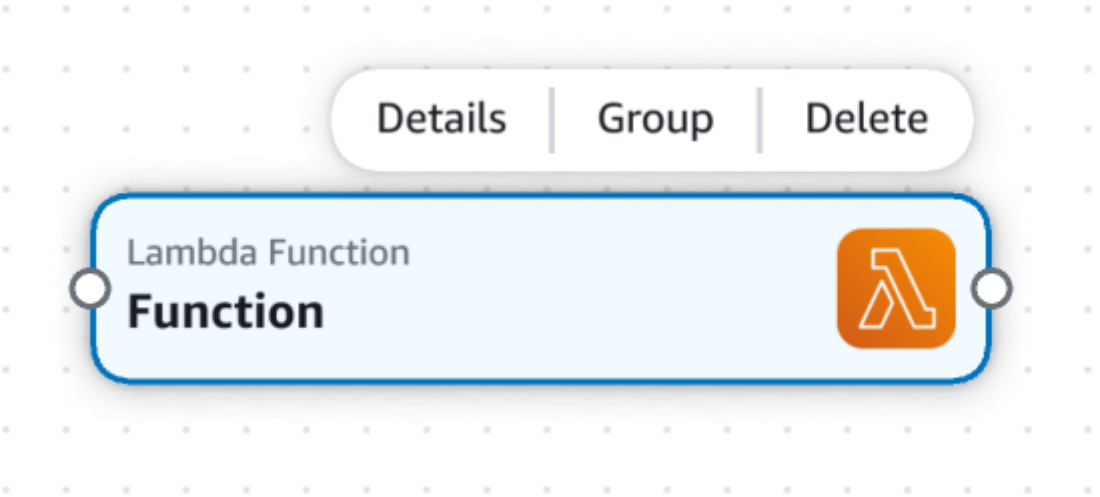
1. Open the **Resource properties** panel of the standard IaC component card.
2. In the **Editing** field, select the standard IaC resource to edit from the dropdown list.
3. Modify your infrastructure code and **Save**.

Delete cards in Infrastructure Composer

This section provides instructions for deleting cards in AWS Infrastructure Composer.

Enhanced component cards

To delete an enhanced component card, select a card you have place on the visual canvas. From the **Card actions** menu, select **Delete**.



Standard component cards

To delete standard component cards, you must manually remove the infrastructure code for each 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.

View code updates with the Change Inspector in Infrastructure Composer

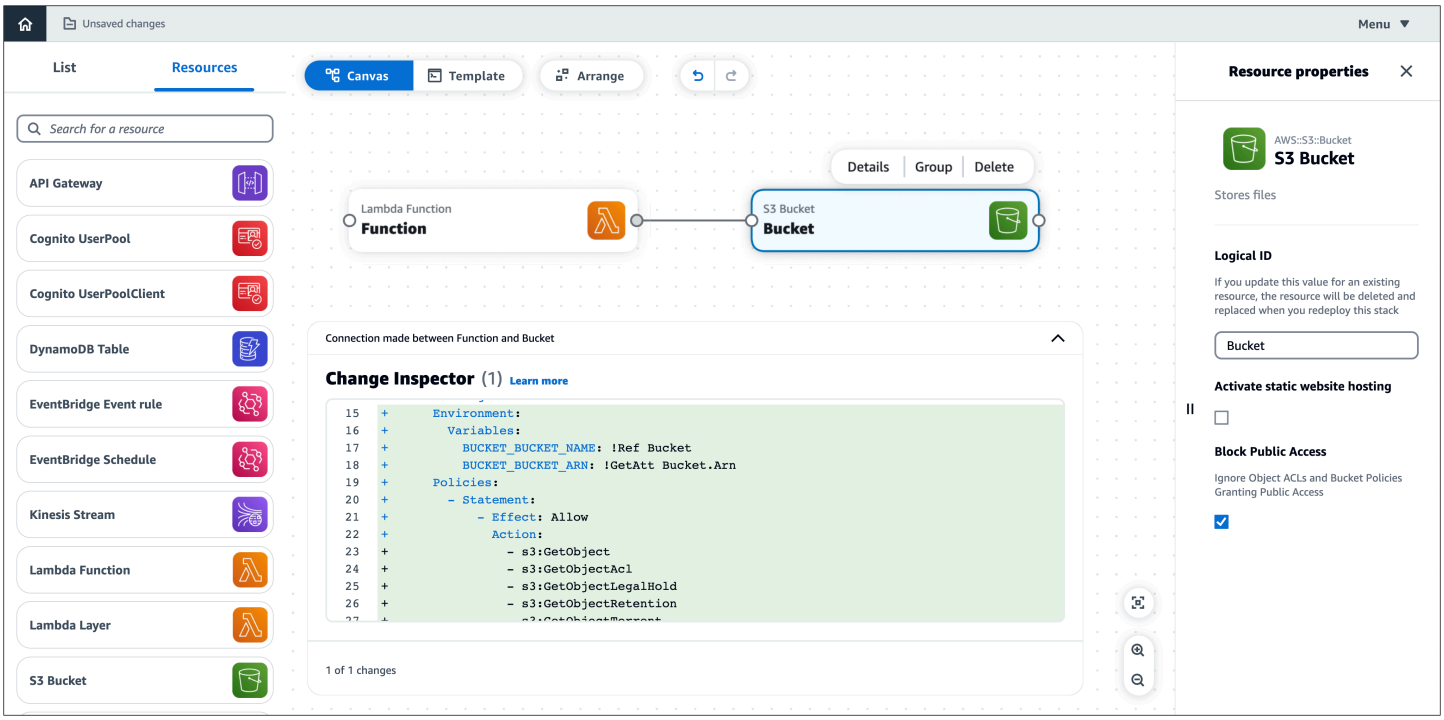
As you design in Infrastructure Composer console, your infrastructure code is automatically created. Use the **Change Inspector** to view your template code updates and learn what Infrastructure Composer is creating for you.

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

The **Change Inspector** is a visual tool within Infrastructure 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.

The following example demonstrates how change inspector works:



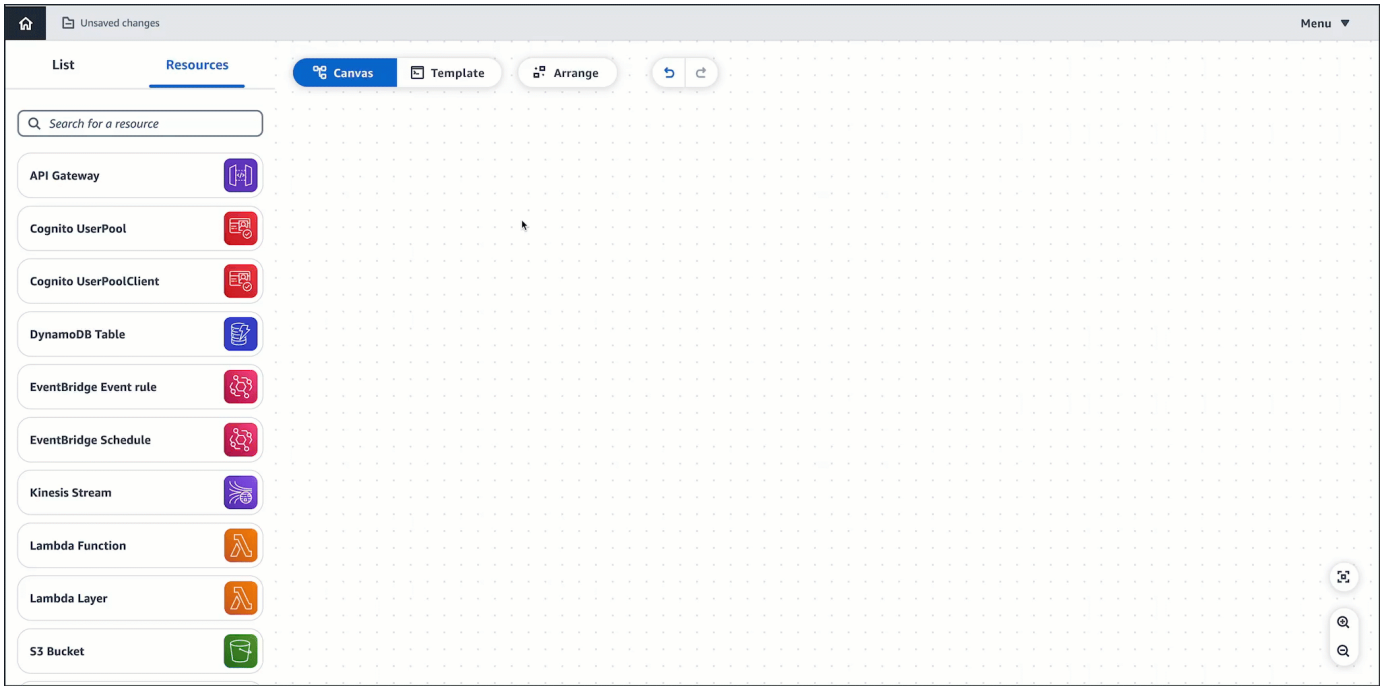
Benefits of the Change Inspector

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

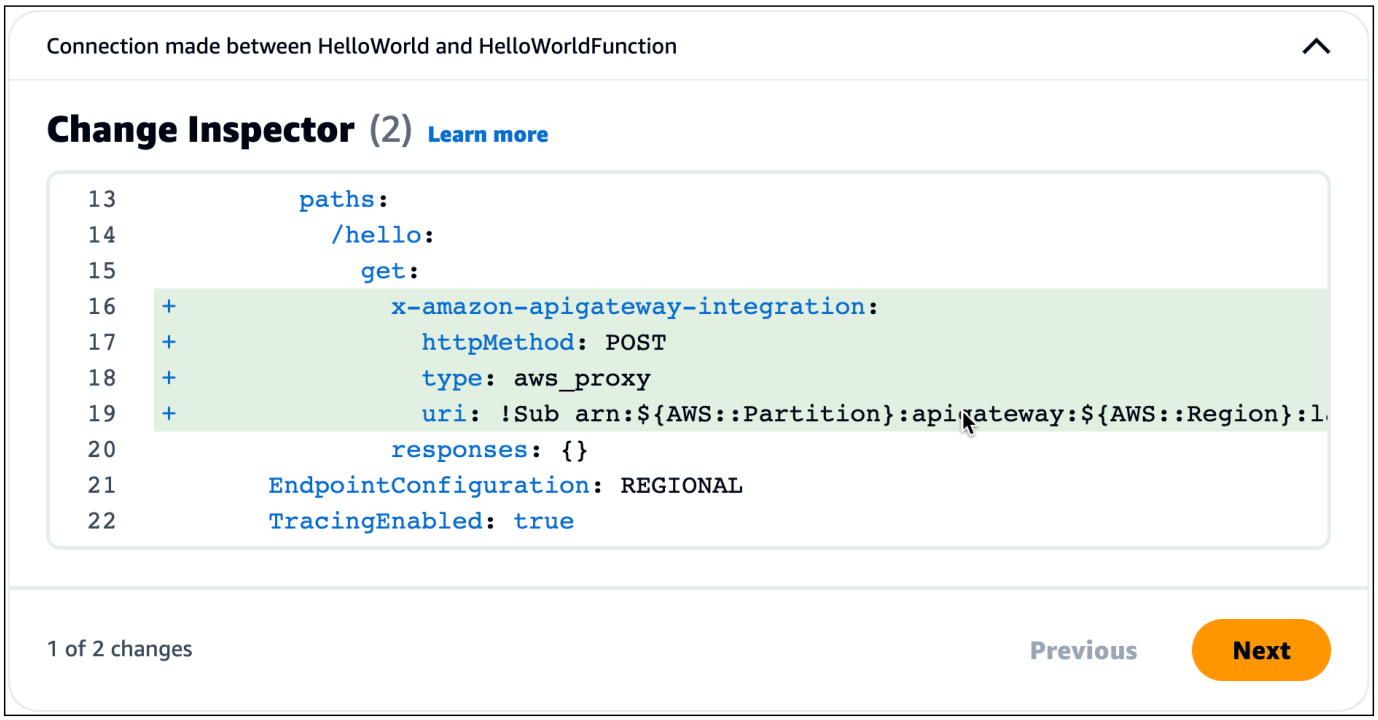
Procedure

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.

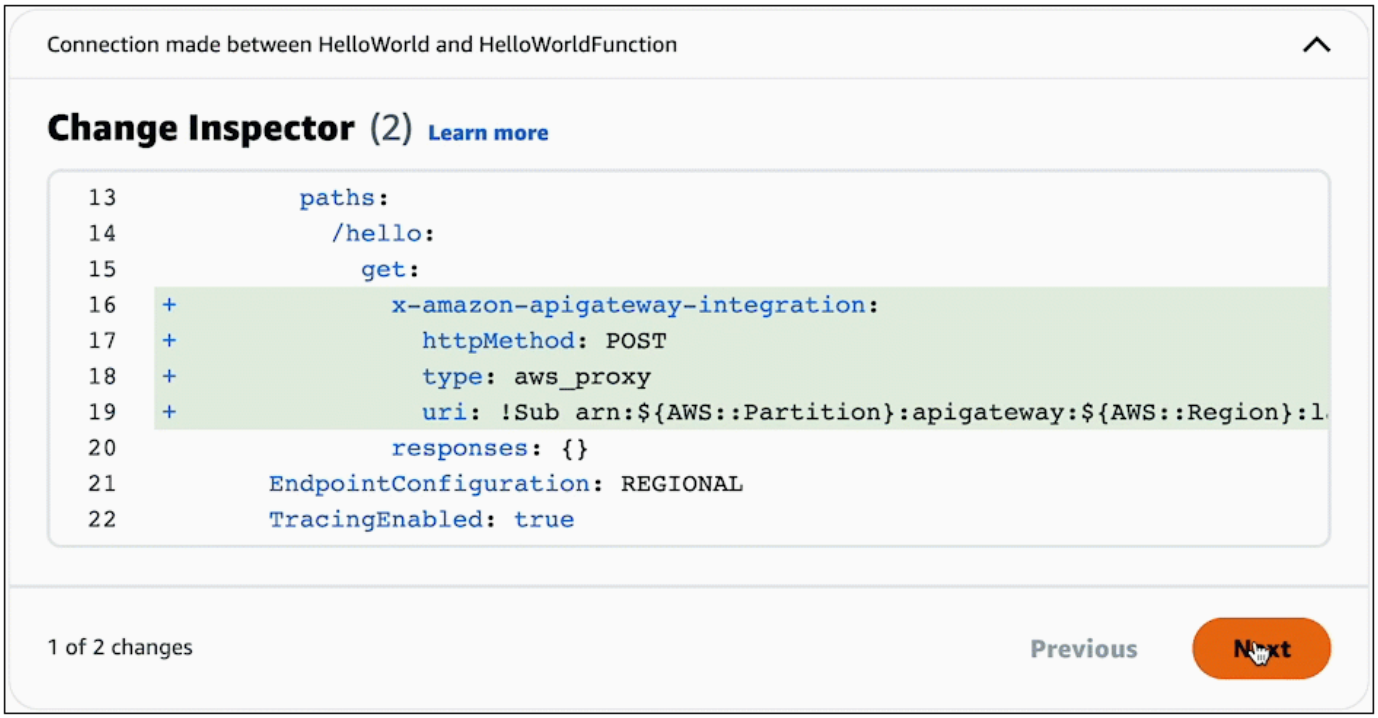


a. Code highlighted **green** indicate newly added code.

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 Infrastructure 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 Infrastructure Composer with a local IDE and view your entire template on your local machine. To learn more, see Connect the Infrastructure Composer console with your local IDE.

Learn more

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

- Card connections in Infrastructure Composer.

Reference external files in Infrastructure Composer

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 Infrastructure 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 Infrastructure Composer from the Infrastructure Composer console, you must use Infrastructure Composer in **local sync** mode. For more information, see Locally sync and save your project in the Infrastructure Composer console.

Topics

Best practices for Infrastructure Composer external reference files

Use Infrastructure Composer with a local IDE

When you use Infrastructure 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 Infrastructure Composer canvas. To learn more, see Connect the Infrastructure Composer console 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. Infrastructure 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*.

Create an external file reference in Infrastructure Composer

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, Infrastructure Composer will create it.

4. **Save** your changes.

Load a project with an external file reference in Infrastructure Composer

Follow the steps listed on this page to load an Infrastructure Composer project with an external file reference.

From the Infrastructure Composer console

1. Complete the steps listed in Import an existing project template in the Infrastructure Composer console.
2. Confirm Infrastructure Composer prompts you to connect to the root folder of your project

If your browser supports the File System Access API, Infrastructure Composer will prompt you to connect to the root folder of your project. Infrastructure 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.

From the Toolkit for VS Code

1. Complete the steps listed in Access Infrastructure Composer from the AWS Toolkit for Visual Studio Code.
2. Open the template you want to view in Infrastructure Composer.

When you access Infrastructure Composer from a template, Infrastructure 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.

Create an application that references an external file in Infrastructure Composer

This example uses the AWS SAM CLI to create an application that references an external file for its state machine definition. You then load your project in Infrastructure Composer with your external file properly referenced.

Example

1. First, use the AWS SAM CLI **sam init** command to initialize a new application named demo. During the interactive flow, 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.

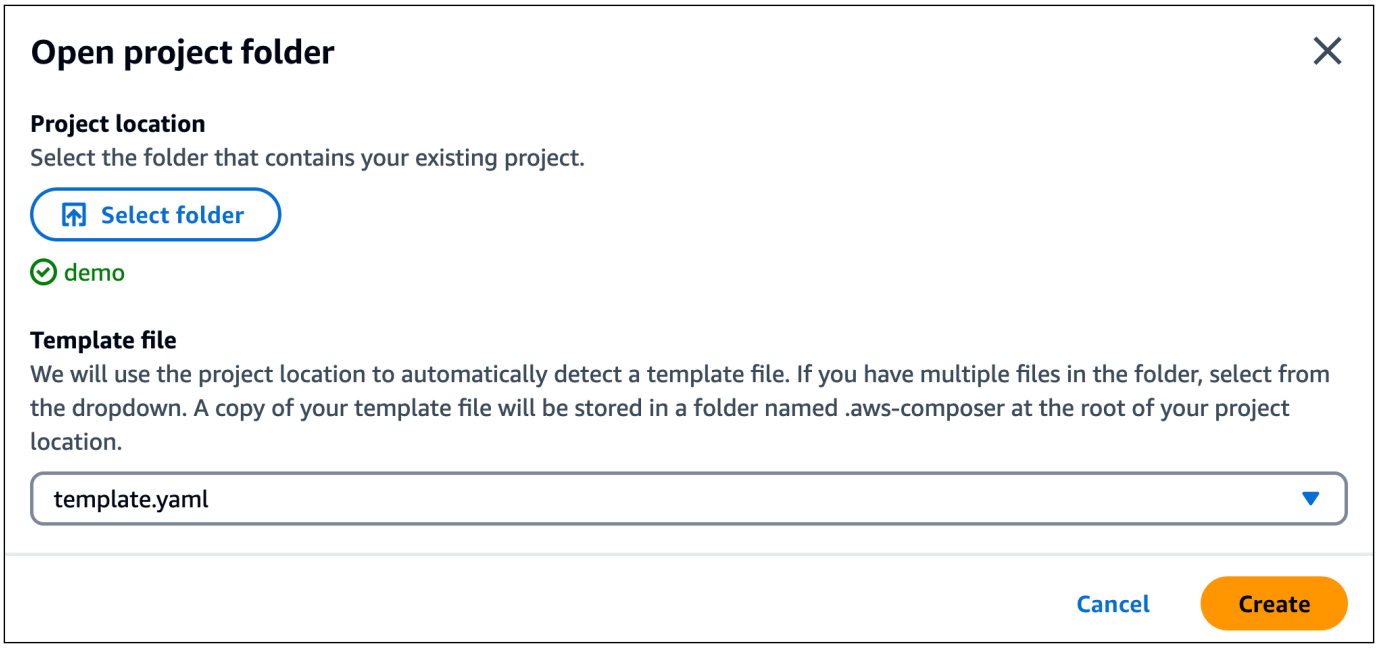
```
...
Resources:
  StockTradingStateMachine:
    Type: AWS::Serverless::StateMachine
    Properties:
      DefinitionUri: statemachine/stock_trader.asl.json
...
```

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

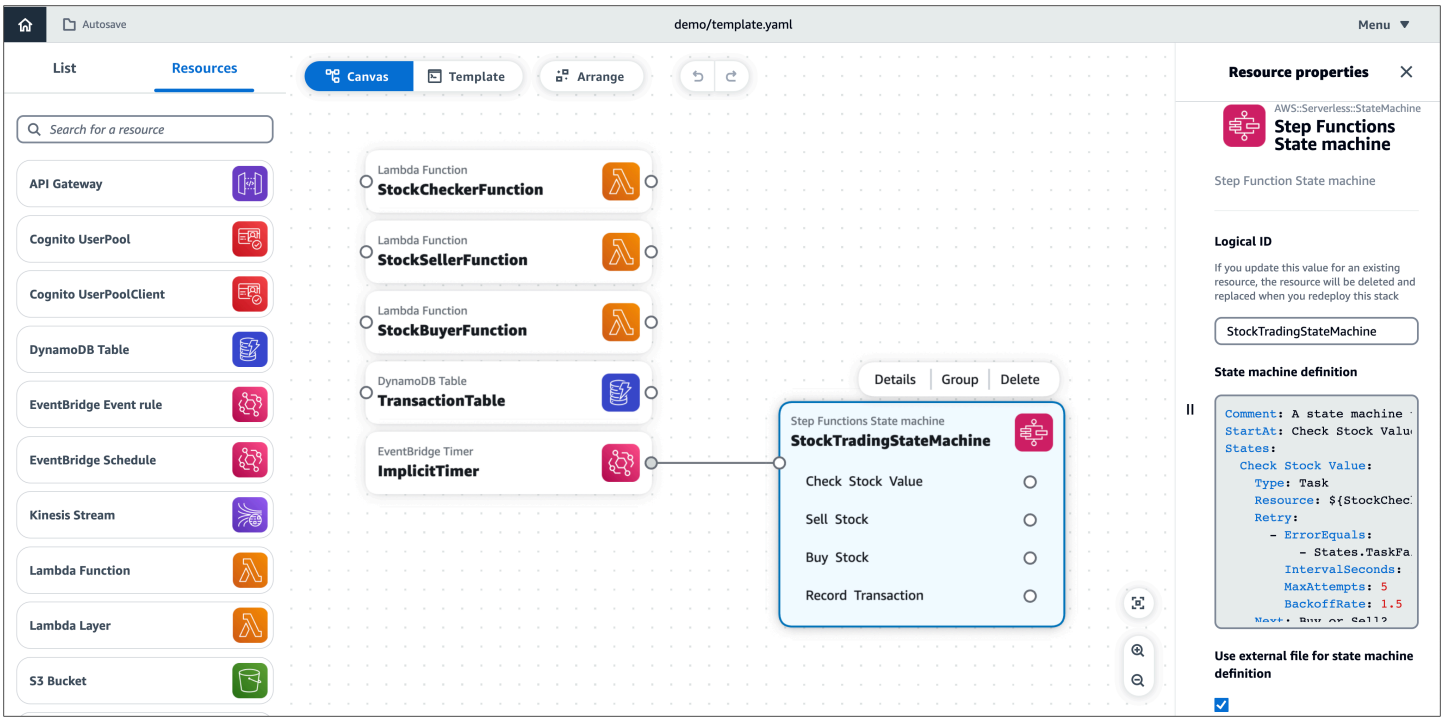
```
demo
### README.md
### __init__.py
### functions
```

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

2. Next, load your application in Infrastructure Composer from the console. From the Infrastructure Composer **home** page, select **Load a CloudFormation template**.
3. Select our `demo` project folder and allow the prompt to view files. Select our `template.yaml` file and select **Create**. When prompted, select **Save changes**.



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



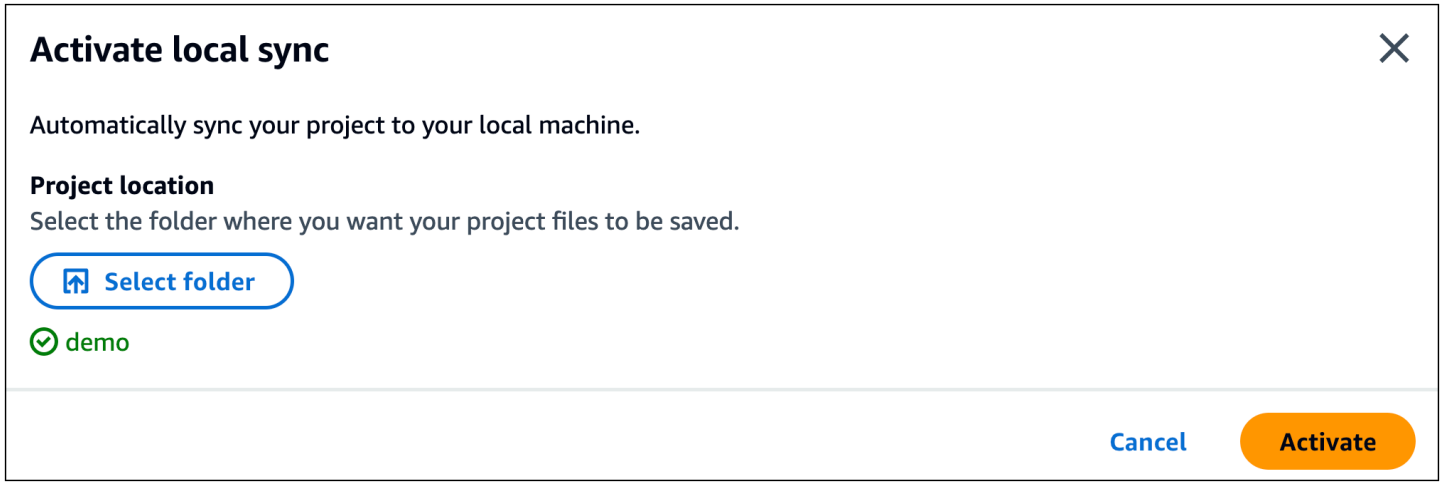
Any changes made to the state machine definition file will be automatically reflected in Infrastructure Composer.

Reference an OpenAPI specification external file with Infrastructure Composer

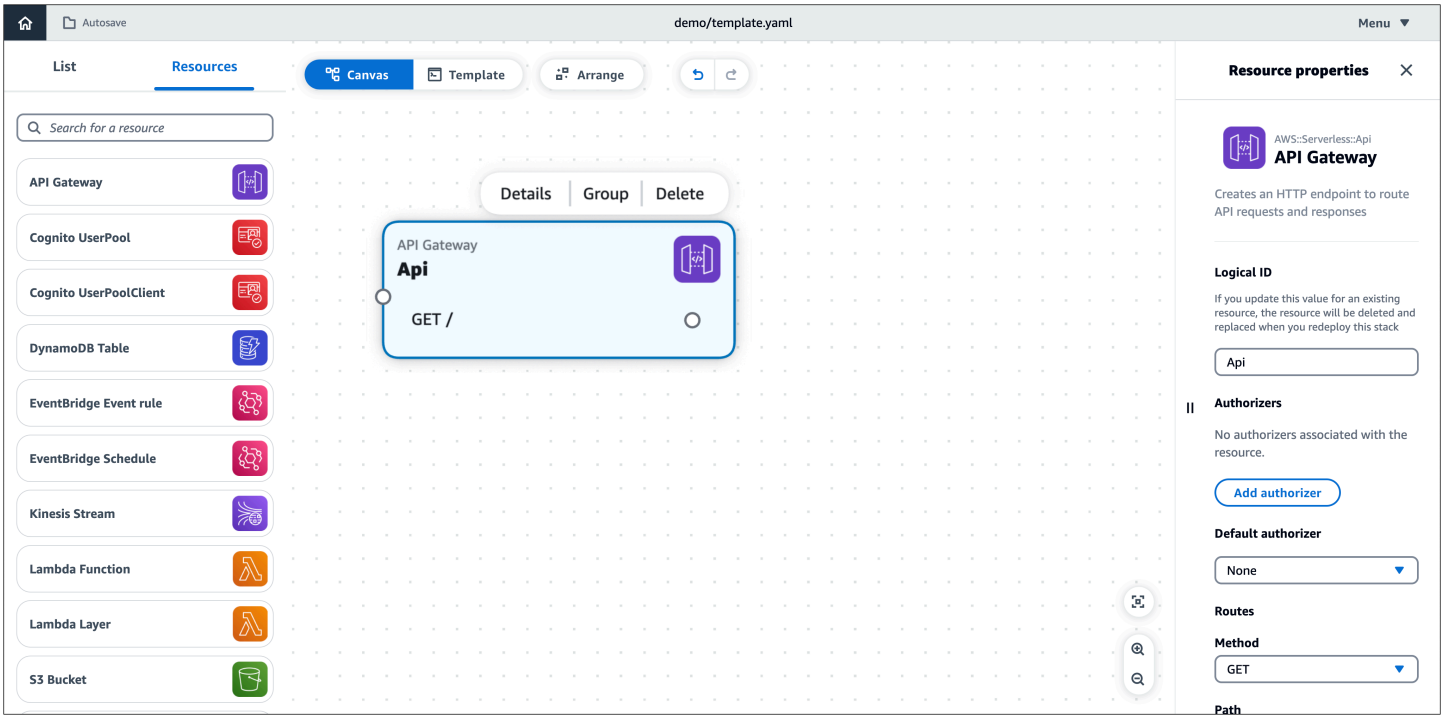
This example uses Infrastructure Composer from the console to reference an external OpenAPI specification file that defines a API Gateway REST API.

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

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

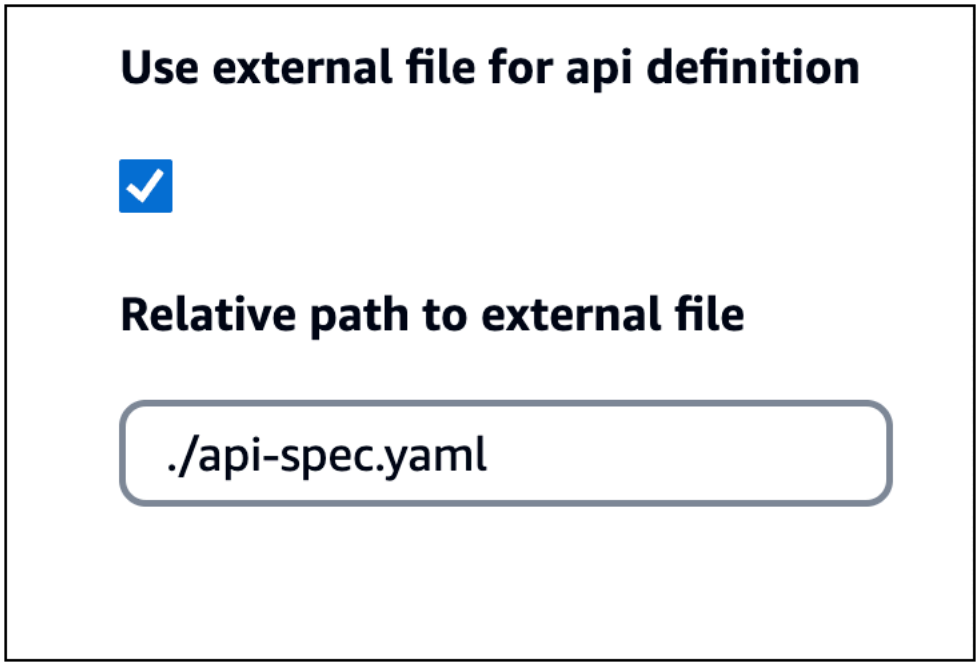


Next, drag an Amazon API Gateway card onto the canvas. Select **Details** to bring up the **Resource properties** panel.



From the **Resource properties** panel, 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:

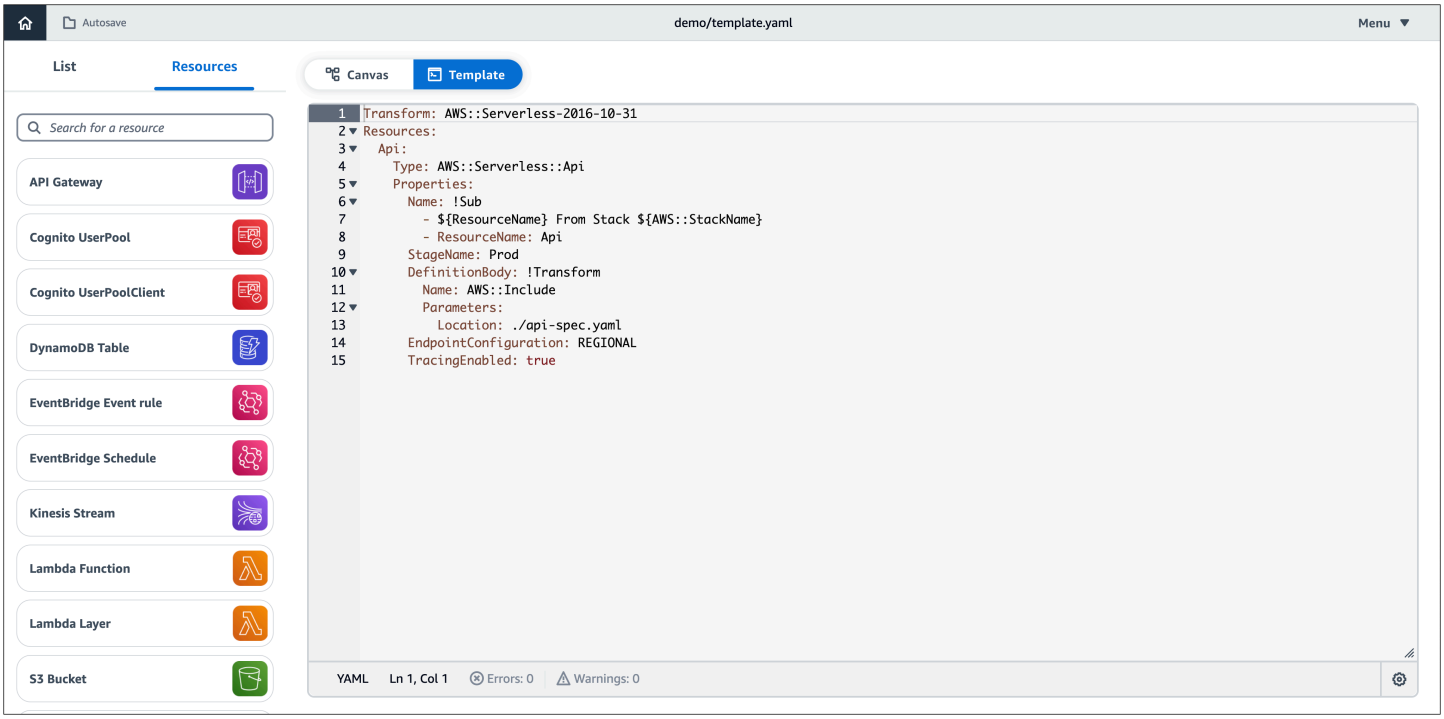
```
demo
### api-spec.yaml
```

Now, you can configure the external file on our local machine. Using our IDE, open the `api-spec.yaml` located in your project folder. 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 Infrastructure Composer **Template** view, you can see that Infrastructure Composer has automatically updated your template to reference the external file.



Integrate Infrastructure Composer with Amazon Virtual Private Cloud (Amazon VPC)

AWS Infrastructure Composer features an integration with the Amazon Virtual Private Cloud (Amazon VPC) service. Using Infrastructure 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.

The following image shows is an example of an application with a Lambda function configured with a VPC.

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

Topics

- Identify Infrastructure Composer resources and related information in a VPC
- Configure Lambda functions with external VPCs in Infrastructure Composer
- Parameters in imported templates for an external VPC with Infrastructure Composer
- Adding new parameters to imported templates with Infrastructure Composer
- Configure a Lambda function and a VPC defined in another template with Infrastructure Composer

Identify Infrastructure Composer resources and related information in a VPC

To integrate Infrastructure Composer with Amazon VPC, you must first identify resources in a VPC and the information needed to complete an integration. This also includes configuration information related to security groups, subnet identifiers, parameter types, SSM types, static value types.

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

Security group and subnet identifiers

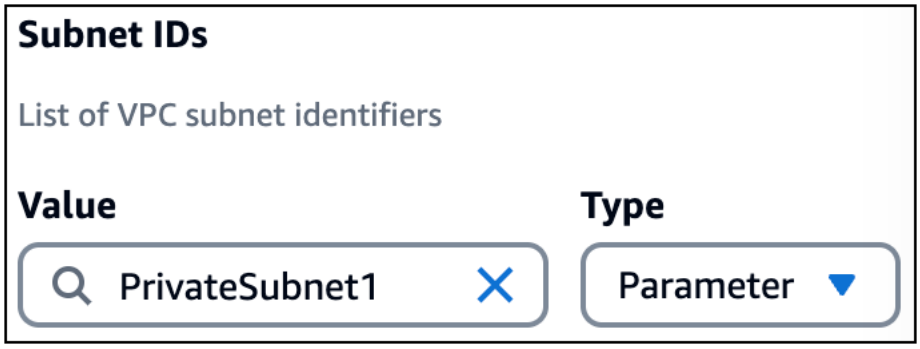
A Lambda function can be configured with multiple security groups and subnets. To configure a security group or subnet for a Lambda function, provide a value and type.

- **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, Infrastructure Composer defines it in the `Parameters` section of your template. Then, Infrastructure 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 Infrastructure 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*.

Infrastructure 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}}'
  ...
```

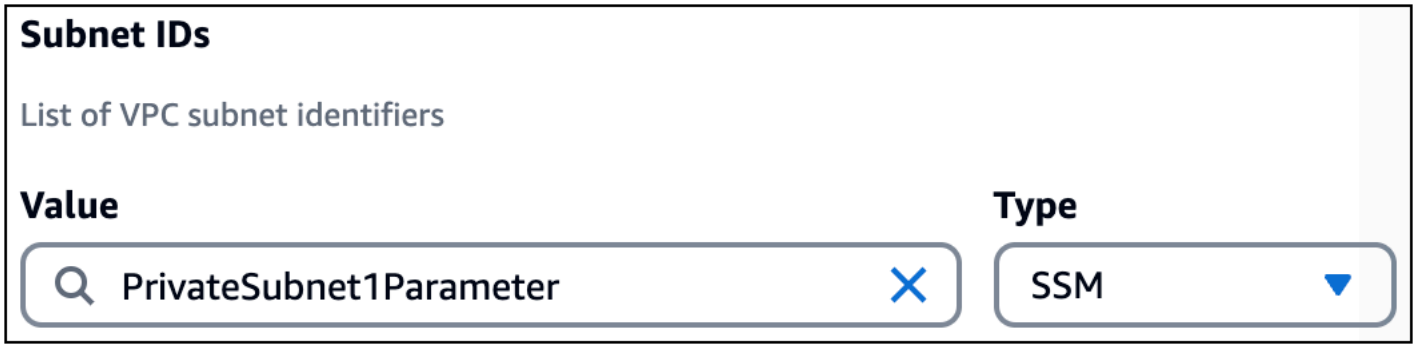
Logical ID

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:



Infrastructure 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 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:

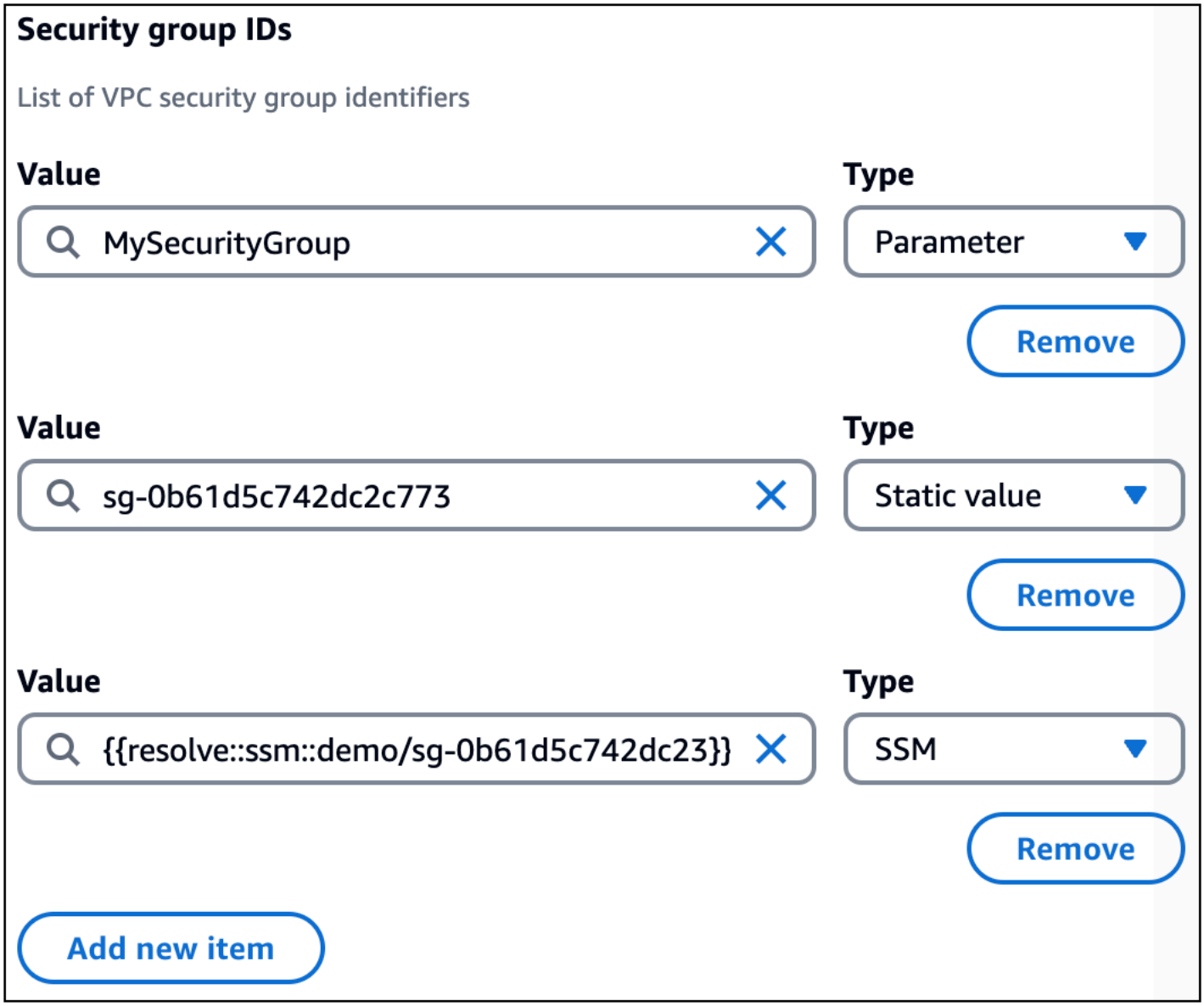
- For security groups, provide the `GroupId`. For more information, see Return values in the *AWS CloudFormation User Guide*. The following is an example: `sg-0b61d5c742dc2c773`.
- For subnets, provide the `SubnetId`. For more information, see Return values in the *AWS CloudFormation User Guide*. The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

Infrastructure 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:



Infrastructure 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 Infrastructure Composer
```

Configure Lambda functions with external VPCs in Infrastructure Composer

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. See Security group and subnet identifiers for details.
4. **Save** your changes.

Parameters in imported templates for an external VPC with Infrastructure Composer

When you import an existing template with parameters defined for the security groups and subnets of an external VPC, Infrastructure 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 Infrastructure Composer
    Type: List<AWS::EC2::SecurityGroup::Id>
  VPCSubnets:
    Description: Subnet IDs generated by Infrastructure Composer
```

```
    Type: List<AWS::EC2::Subnet::Id>
  VPCSubnet:
    Description: Subnet Id generated by Infrastructure 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

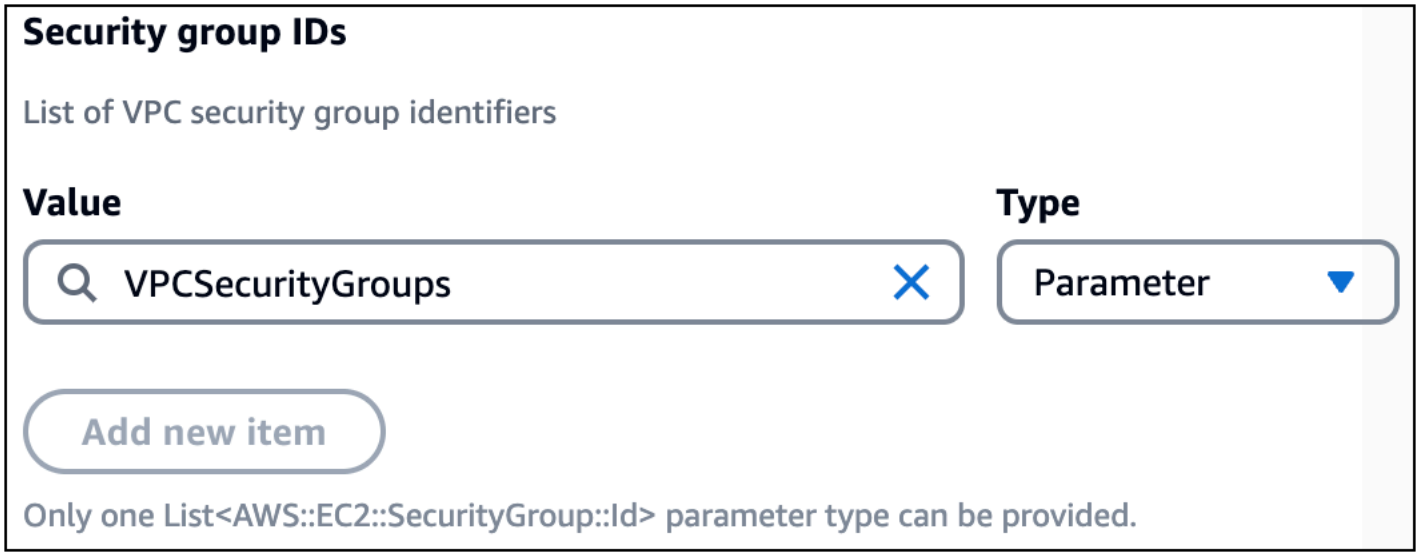
Normally, you can specify multiple security group and subnet identifiers for each Lambda function. If your existing template contains list parameter types, such as `List<AWS::EC2::SecurityGroup::Id>` or `List<AWS::EC2::Subnet::Id>`, you can only specify one identifier.

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 Infrastructure Composer
    Type: List<AWS::EC2::SecurityGroup::Id>
...
```

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



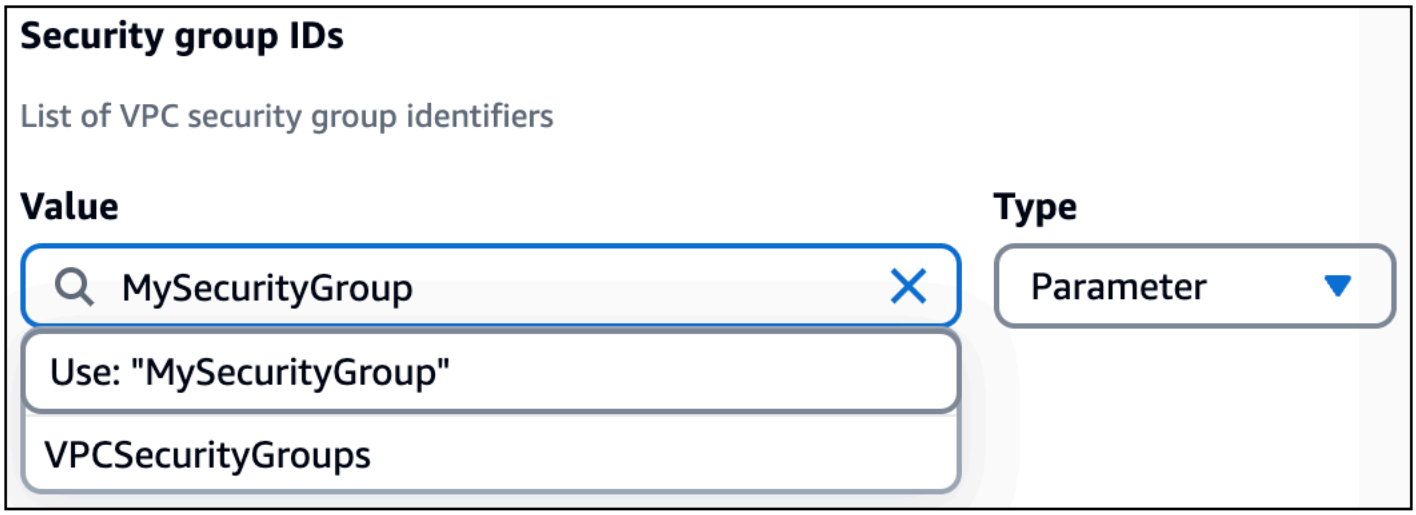
This limitation occurs because the `SecurityGroupIds` and `SubnetIds` properties of an `AWS::Lambda::Function VpcConfig` object both accept only a list of string values. Since a single list parameter type contains a list of strings, it can be the only object provided when specified.

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 Infrastructure Composer
    Type: List<AWS::EC2::SecurityGroup::Id>
  VPCSubnets:
    Description: Subnet IDs generated by Infrastructure 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 with Infrastructure Composer

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, Infrastructure 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.

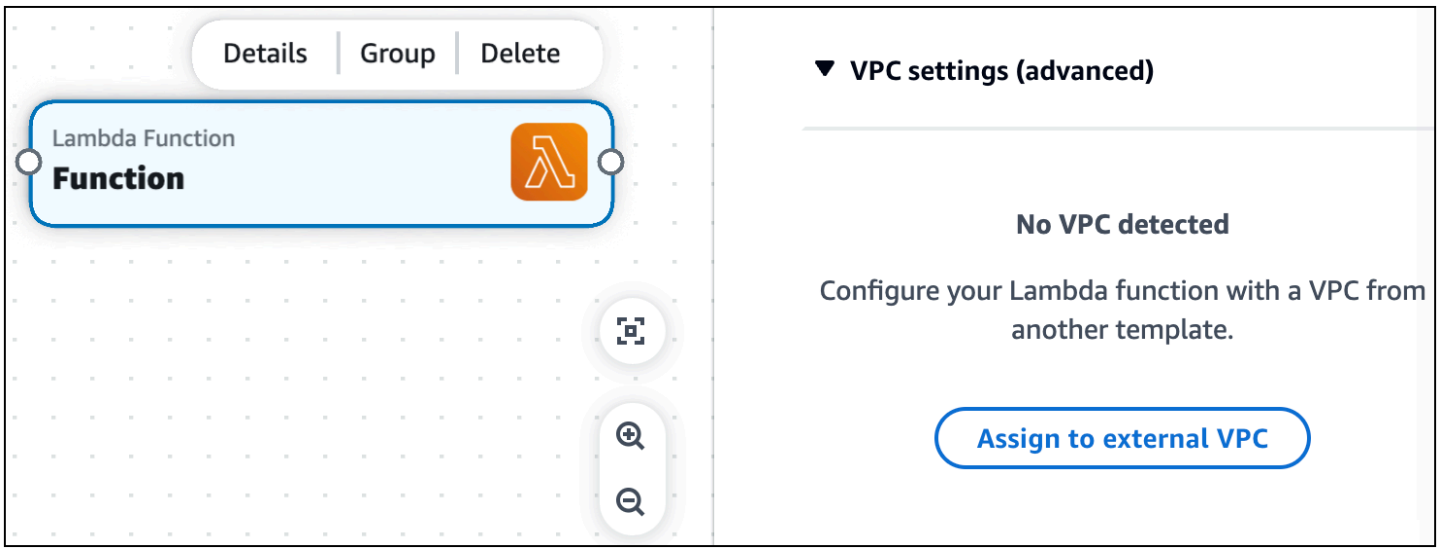
```
...
Parameters:
  VPCSecurityGroups:
    Description: Security group IDs generated by Infrastructure Composer
    Type: List<AWS::EC2::SecurityGroup::Id>
  VPCSubnets:
    Description: Subnet IDs generated by Infrastructure Composer
    Type: List<AWS::EC2::Subnet::Id>
Resources:
  ...
  MyFunction: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

Configure a Lambda function and a VPC defined in another template with Infrastructure Composer

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

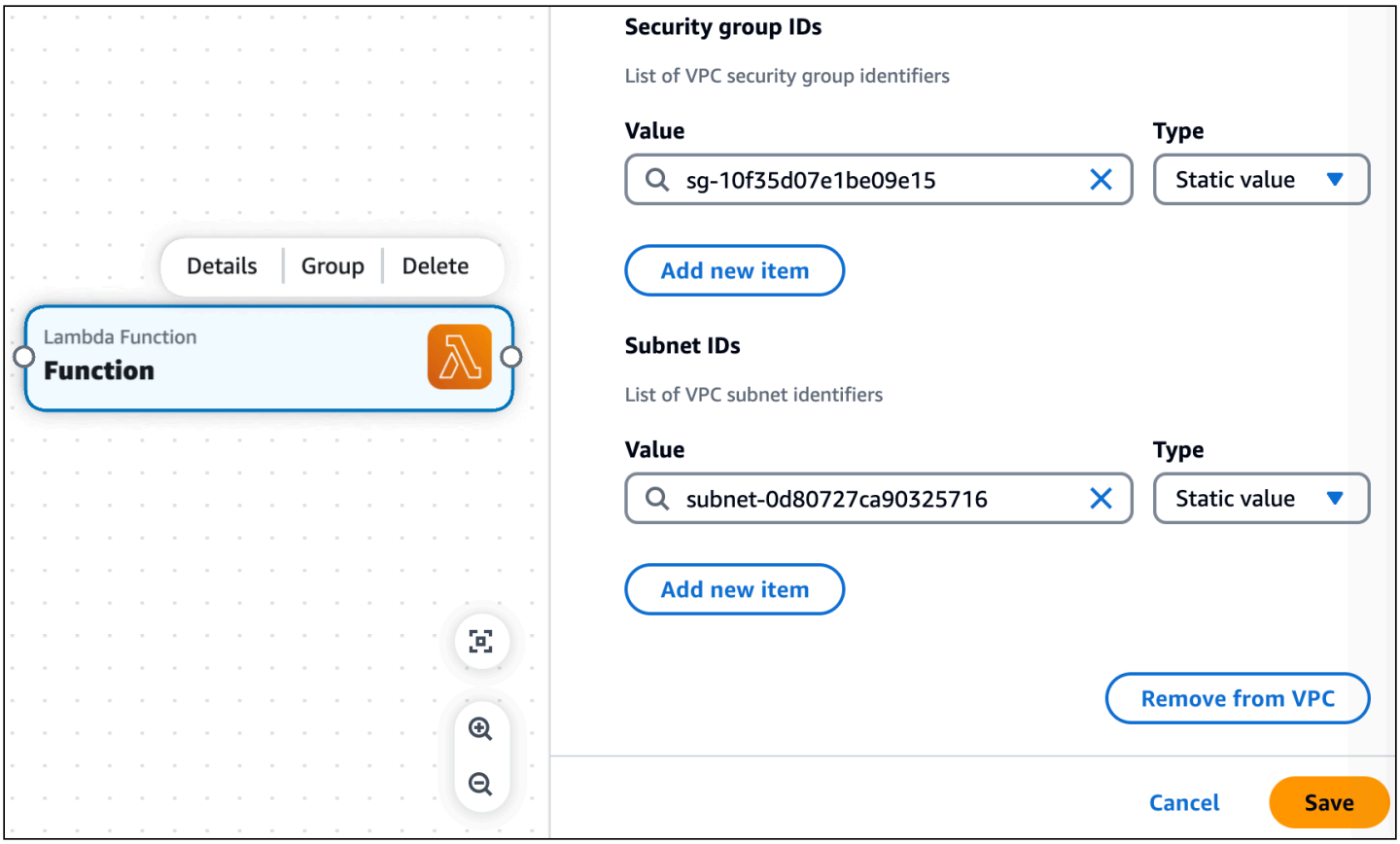
We start by dragging a **Lambda Function** enhanced component card onto the canvas.

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

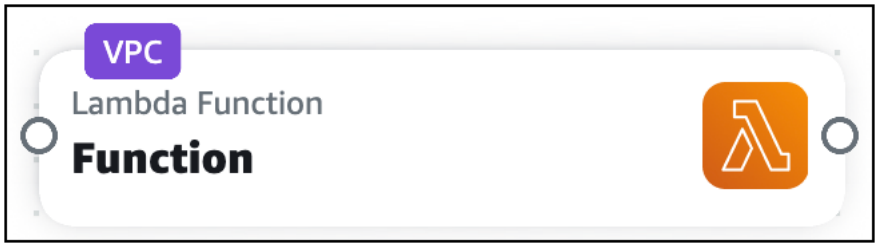


Next, we select **Assign to external VPC** to begin configuring a VPC from an external template.

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.



Infrastructure 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}
```

Deploy your Infrastructure Composer serverless application to the AWS Cloud

Use AWS Infrastructure 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).

AWS SAM is an open-source framework that provides developer tools for building and running serverless applications on AWS. With AWS SAM's shorthand syntax, developers declare CloudFormation resources and specialized serverless resources that are transformed to infrastructure during deployment.

Important AWS SAM concepts

Before you use AWS SAM, it's important you become familiar with some of its fundemental concepts.

- **How AWS SAM works**: This topic, which is in the *AWS Serverless Application Model Developer Guide*, provides important information on the primary components you use to create your serveless application: The AWS SAM CLI, the AWS SAM project, and the AWS SAM template.
- **How to use AWS Serverless Application Model (AWS SAM)**: This topic, which is in the *AWS Serverless Application Model Developer Guide*, provides a high-level overview of the steps you need to complete to use AWS SAM to deploy your application to the AWS Cloud.

As you design your application in Infrastructure Composer, you can use the **sam sync** command to have the AWS SAM CLI automatically detect local changes and deploy those changes to CloudFormation. To learn more, see Using sam sync in the *AWS Serverless Application Model Developer Guide*.

Next steps

Refer to Set up for deploying with the AWS SAM CLI and Infrastructure Composer to prepare to deploy your application.

Set up for deploying with the AWS SAM CLI and Infrastructure Composer

To deploy your application with AWS SAM, you first need to install and access the AWS CLI and the AWS SAM CLI. The topics in this section provide details on doing this.

Install the AWS CLI

We recommend installing and setting up the AWS CLI before installing the AWS SAM CLI. For instructions, see Install or update to the latest version of the AWS CLI in the *AWS Command Line Interface User 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*.

Access the AWS SAM CLI

If you use Infrastructure 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. Infrastructure 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 Locally sync and save your project in the Infrastructure Composer console.

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 Infrastructure Composer from the AWS Toolkit for Visual Studio Code

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

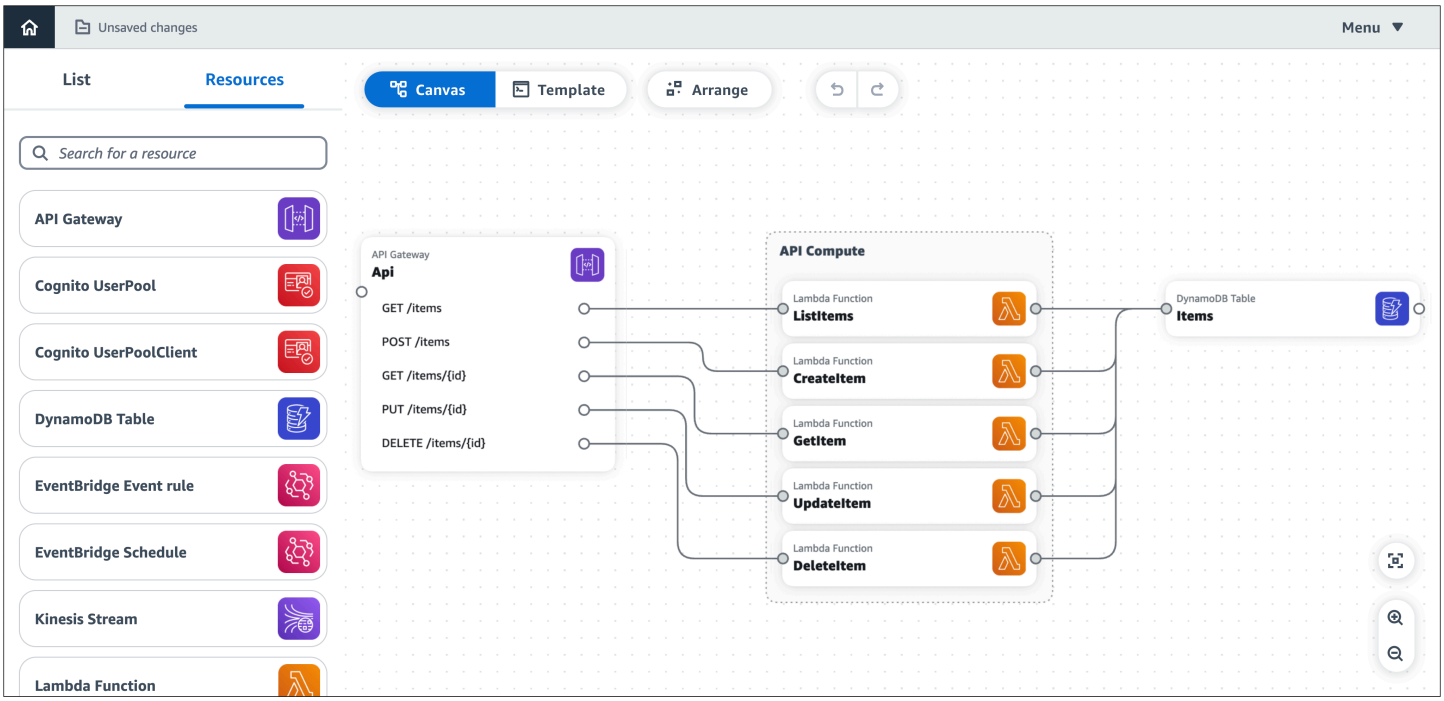
Next steps

To deploy your application, refer to Use Infrastructure Composer with AWS SAM to build and deploy.

Use Infrastructure Composer with AWS SAM to build and deploy

Now that you have completed Set up for deploying with the AWS SAM CLI and Infrastructure Composer, you can deploy your application with AWS SAM and Infrastructure Composer. This section provides an example detailing how you can do this. You can also refer to Deploy your application and resources with AWS SAM in the *AWS Serverless Application Model Developer Guide* for instructions on deploying your application with AWS SAM.

This example shows you how to build and deploy the Infrastructure Composer demo application. The demo application has the following resources:



Note

- To learn more about the demo application, see Load and modify the Infrastructure Composer demo project.
- For this example, we use Infrastructure Composer with **local sync** activated.

1. 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
```

2. 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-samclisam-s3-
demo-1b3x26zbcdkqr
    Capabilities                 : ["CAPABILITY_IAM"]
    Parameter overrides          : {}
    Signing Profiles             : {}
```

The AWS SAM CLI deploys the application, first by creating an 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
```

Use Infrastructure Composer with AWS SAM to delete a stack

This example shows you how to delete an CloudFormation stack using the **sam delete** command.

Enter the command **sam delete** in the AWS SAM CLI and confirm whether you want to delete the stack and the template:

```
$ 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
```

AWS Infrastructure Composer troubleshooting

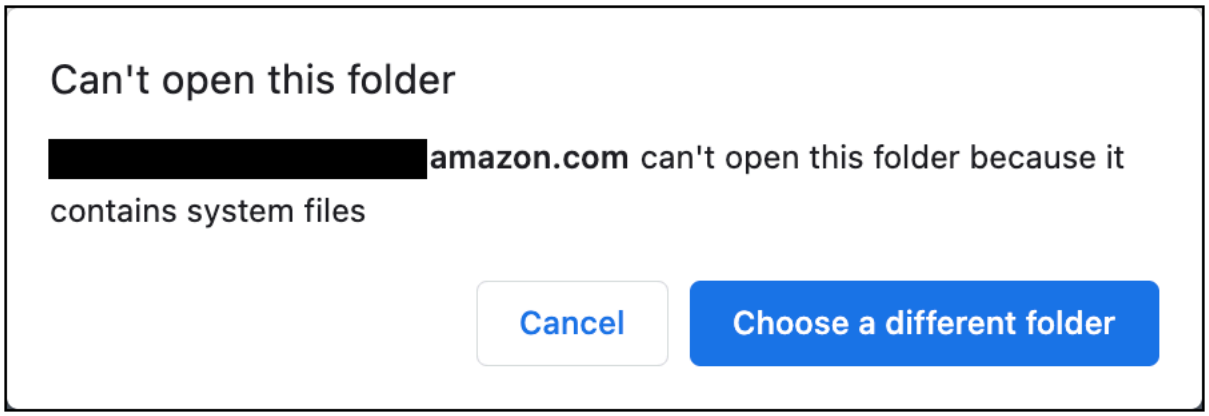
The topics in this section provide guidance on troubleshooting error messages when using AWS Infrastructure Composer.

Topics

Error messages

"Can't open this folder"

Example error:



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

To learn more about this error, see Data Infrastructure Composer gains access to.

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

"Incompatible template"

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

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

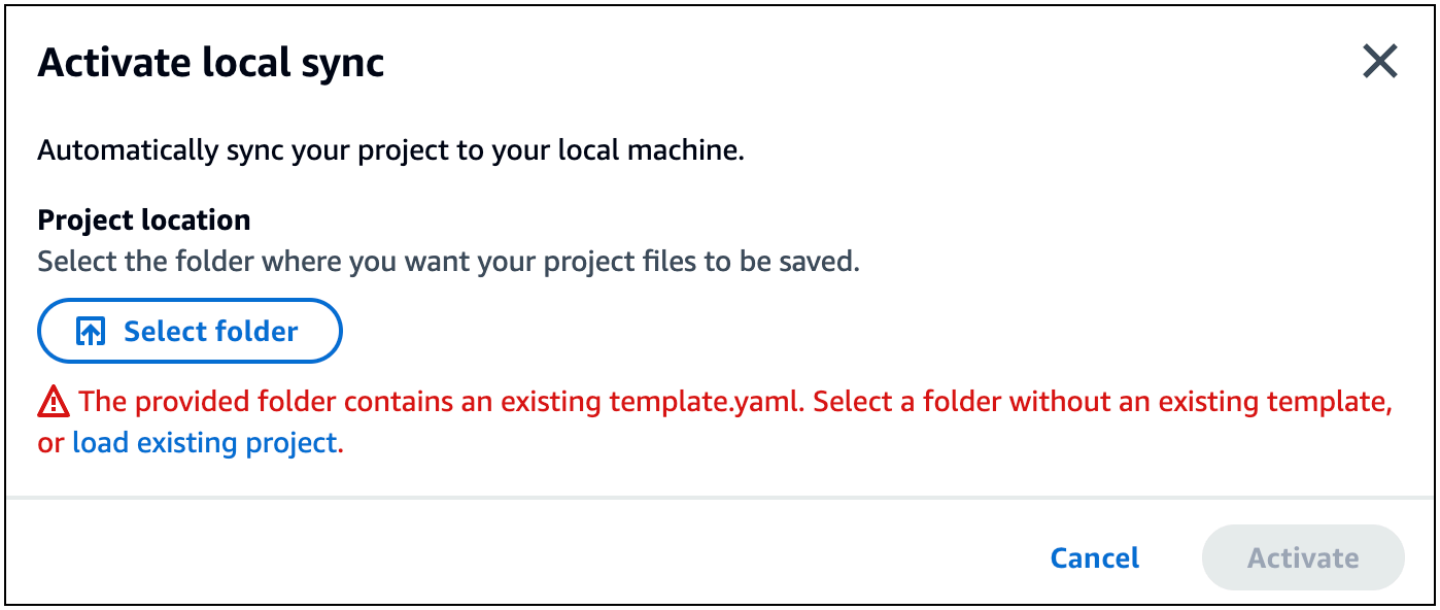
To learn about supported external files in Infrastructure Composer, see Reference external files.

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 Infrastructure 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: Infrastructure Composer is unable to access a sensitive directory using local sync mode.

To learn more about this error, see Data Infrastructure Composer gains access to.

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

Security in AWS Infrastructure 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 Infrastructure 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 Infrastructure Composer. The following topics show you how to configure Infrastructure 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 Infrastructure Composer resources.

Topics

Data protection in AWS Infrastructure Composer

The AWS shared responsibility model applies to data protection in AWS Infrastructure 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. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with Infrastructure 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 Infrastructure Composer is used for the sole purpose of providing functionality within Infrastructure Composer and generating project files and directories that are saved locally to your machine. Infrastructure Composer does not save, store or transmit any of this data.

Data encryption

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

Encryption at rest

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

Encryption in transit

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

Key management

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

Inter-network traffic privacy

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

AWS Identity and Access Management for AWS Infrastructure 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 Infrastructure Composer resources. IAM is an AWS service that you can use with no additional charge.

Topics

- How AWS Infrastructure Composer works with IAM

Audience

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

Authenticating with identities

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

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

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

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM 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 defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, see Choose 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 include 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. You must specify a principal in a resource-based policy.

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 policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.

- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. 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 Infrastructure Composer works with IAM

AWS Infrastructure Composer requires, at minimum, read-only access to the AWS Management Console. Any user with this authorization can use all features of Infrastructure Composer. Granular access to specific features of Infrastructure 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*.

The following table shows what IAM features can be used with AWS Infrastructure Composer.

IAM feature	Infrastructure 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

IAM feature	Infrastructure Composer support
Principal permissions	No
Service roles	No
Service-linked roles	No

To get a high-level view of how Infrastructure 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 Infrastructure 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 Define custom IAM permissions with customer managed 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. 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 Infrastructure 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. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

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

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

Policy resources for Infrastructure 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. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

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

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

Policy condition keys for Infrastructure 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 specifies when statements execute based on defined criteria. 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. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

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

ACLs in Infrastructure 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 Infrastructure Composer

Supports ABAC (tags in policies): No

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

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 Define permissions with ABAC authorization 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 Infrastructure Composer

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

You can use temporary credentials to access Infrastructure 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 Infrastructure Composer

Supports forward access sessions (FAS): No

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see Forward access sessions.

Service roles for Infrastructure 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 Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

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

Service-linked roles for Infrastructure 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 Infrastructure 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. For more information about your compliance responsibility when using AWS services, see AWS Security Documentation.

Resilience in AWS Infrastructure 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 Infrastructure Composer is used for the sole purpose of providing functionality within Infrastructure Composer and generating project files and directories that are saved locally to your machine. Infrastructure Composer does not save or store any of this data.

Document history for Infrastructure Composer

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

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

Change	Description	Date
Restructured and updated content throughout the developer guide	Reorganized and restructured the guide to improve discoverability and usability. Updated and improved titles. Provided additional details when introducting topics and concepts.	August 1, 2024
Added documentation for using Infrastructure Composer in CloudFormation console mode and restructured the Infrastructure Composer Developer Guide.	AWS Infrastructure Composer can now be used in CloudFormation console mode. To learn more, see Using Infrastructure Composer in CloudFormation console mode. Additionally, much of the content in the user guide has been reorganized to create a streamlined experience.	March 28, 2024
Added documentation for the Infrastructure Composer integration with CodeWhisperer	AWS Infrastructure Composer from the Toolkit for VS Code provides an integration with Amazon CodeWhisperer. To learn more, see Using AWS Infrastructure Composer with Amazon CodeWhisperer.	November 30, 2023

Added documentation for deploying your application with Infrastructure Composer from the AWS Toolkit for Visual Studio Code	Use the **sync** button from the Infrastructure Composer canvas to deploy your application to the AWS Cloud. To learn more, see Deploy your application with sam sync.	November 30, 2023
Added documentation for Infrastructure Composer from the AWS Toolkit for Visual Studio Code	You can now use Infrastructure Composer from VS Code with the AWS Toolkit for Visual Studio Code. To learn more, see Using AWS Infrastructure Composer from the AWS Toolkit for Visual Studio Code.	November 30, 2023
Added Step Functions Workflow Studio integration	Launch Step Functions Workflow Studio from the Infrastructure Composer canvas. To learn more, see Using AWS Infrastructure Composer with AWS Step Functions.	November 27, 2023
Added Lambda console and Infrastructure Composer integration	Launch the Infrastructure Composer canvas from the Lambda console. To learn more, see Using AWS Infrastructure Composer with the AWS Lambda console.	November 14, 2023

Added Amazon VPC as a featured service with Infrastructure Composer	Infrastructure Composer introduces a VPC tag to visualize resources configured with a VPC. You can also configure Lambda functions with VPCs defined on an external template. To learn more, see Using Infrastructure Composer with Amazon VPC.	October 17, 2023
Added Amazon RDS as a featured service with Infrastructure Composer	Connect your Infrastructure Composer application to an Amazon RDS DB cluster or instance that is defined on an external template. To learn more, see Using Infrastructure Composer with Amazon RDS.	October 17, 2023
Added Infrastructure Composer support to design with all CloudFormation resources	Select any CloudFormation resource from the **Resources** palette to design your applications with. To learn more, see Work with any CloudFormation resource.	September 26, 2023
Added documentation for cards in Infrastructure Composer	Infrastructure Composer supports multiple types of cards that you can use to design and build your application. To learn more, see Designing with cards in Infrastructure Composer.	September 20, 2023
Added documentation for undo and redo feature	Use the **undo** and **redo** buttons on the Infrastructure Composer canvas. To learn more, see Undo and redo.	August 1, 2023

Added 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
Added 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
Infrastructure Composer support for external file references	Reference external files for supported resources in Infrastructure 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 Infrastructure Composer visual canvas.	March 7, 2023
New Change Inspector feature	Use the **Change Inspector** to view your template code updates and learn what Infrastructure Composer is creating for you. To learn more, see View code updates with the Change Inspector.	March 7, 2023

Infrastructure Composer now generally available	AWS Infrastructure Composer is now generally available . To learn more, see AWS Infrastructure Composer now generally available - Visually build serverless applications quickly.	March 7, 2023
Expanded on benefits of using connected mode	Use Infrastructure Composer in connected mode with your local IDE to speed up development. To learn more, see Using Infrastructure Composer with your local IDE.	March 7, 2023
Updated topic on using other AWS services to deploy your application	Use Infrastructure Composer to design deployment-ready serverless applicati ons. Use AWS SAM to deploy your serverless application. To learn more, see Using Infrastructure Composer with CloudFormation and AWS SAM.	March 3, 2023
Added serverless concepts section	Learn about basic serverles s concepts before using Infrastructure Composer. To learn more, see Serverless concepts.	March 2, 2023
Public release	Initial public release of Infrastructure Composer.	December 1, 2022