



TGS Reference

Provider Onboarding Guide

AWS Transact Gateway Service



Authored by: Nicholas Leuci

Version 2024-08-20

PRELIMINARY

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Transact Gateway Service: Provider Onboarding Guide

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

Table of Contents

Table of Contents	3
Transact Order Orchestration e-Commerce.....	1
Resources.....	3
AWS Account	3
Connector HTTP Service	3
Transact SDK	3
Current Transact SDKs.....	4
Future / Planned Transact SDKs.....	4
Retailer With Required Capabilities.....	4
Multi-platform Deployment	5
Prerequisites	6
Active AWS Account	6
AWS Account Sign-Up	6
Encryption Key Option	7
Transact Engine ID	7
AWS IAM Account Settings	7
Transact Supported Language and SDK.....	9
Client App e-Commerce Website and Development Project.....	13
Software Tools (Optional But Useful).....	14
Insomnia.....	14
IntelliJ IDEA	15
Postman	15
Swagger	15
Visual Studio Code	17
Public AWS Websites	17
Interfacing Client App / Connector With Transact	21
Configuring The Client App.....	21
Transact API Webservice Calls	22
Handlers For Exceptions and Constraint Violations.....	23
Integrating the Provider Capability Connectors	25
Vendor (Third Party) API Capability Connectors.....	26
Resources for Building Provider (Vendor) API Capability Connectors.....	27
Using the Open API Spec to Build the Connector	28
Using the Transact SDK to Build the Connector	30
Common HTTP Error Codes.....	32

Constraint Violation (CV) Structures	35
Categories / Subcategories of Transact CVs	35
CV Detection Mechanisms	36
Language-Specific AWS SDKs	39

Transact Order Orchestration e-Commerce

Amazon Transact is a new AWS product that helps Retail IT organizations reduce the burden of building, integrating, and maintaining undifferentiated e-commerce functionality, letting them focus on building shopping features that distinguish them from competitors. AWS Transact is an order orchestration system which allows Retailers to use their preferred vendor service for providing some aspects of the order such as catalog, pricing, taxes, fulfillment, etc.

AWS Transact is a composable system with three other principal elements: Retailers, Providers, and Shoppers. Transact is essentially middleware that integrates the other elements, and it mediates transactions and stores permanent records of all orders. Retailers provide the 'front end' or UI of this system. Providers are partner vendors who provide additional, composable capabilities such as the product catalog, inventory, pricing, taxes, promotions, payment, order fulfillment and returns, etc. Retailers can provide these needed capabilities themselves, or they can use one or more preferred provider vendors for these services. Shoppers are the Retailer's e-commerce customers.

The diagram below provides a high-level abstraction that illustrates the basic organization of this composable e-commerce system:

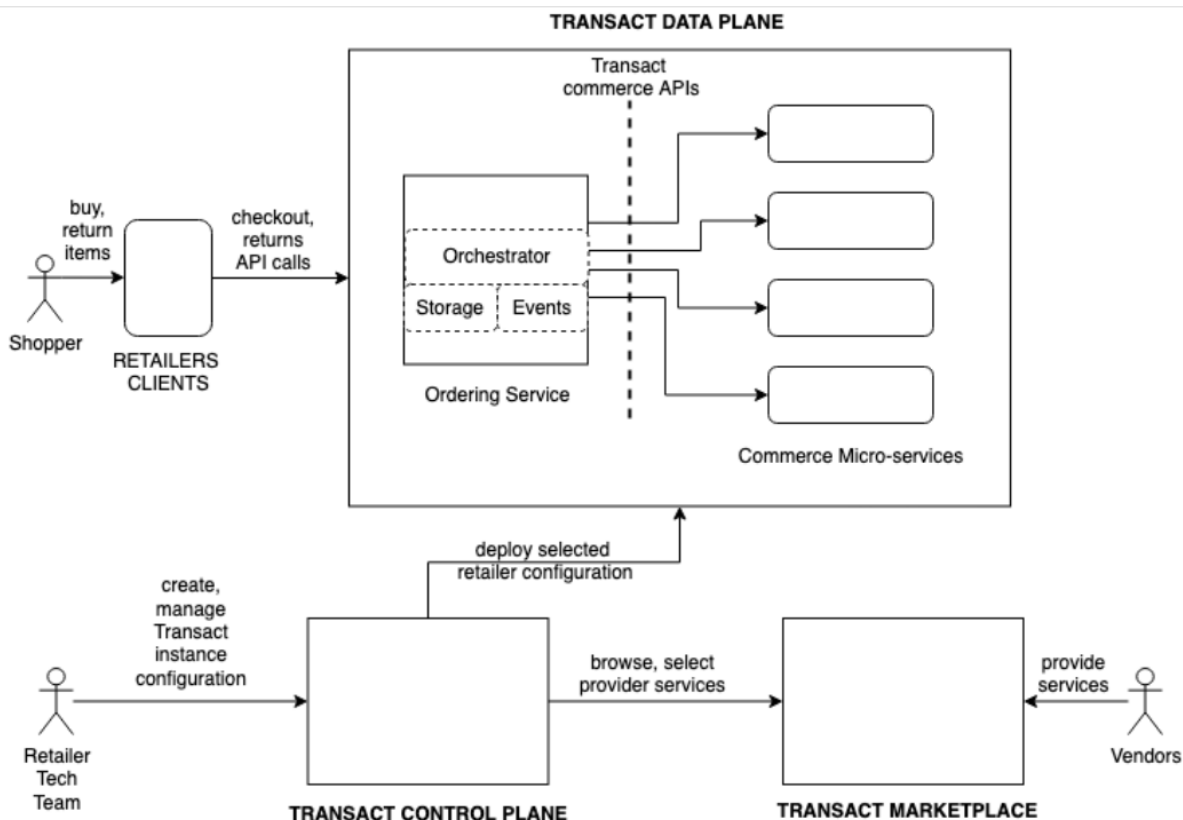


Fig: Simplified High Level Transact Diagram

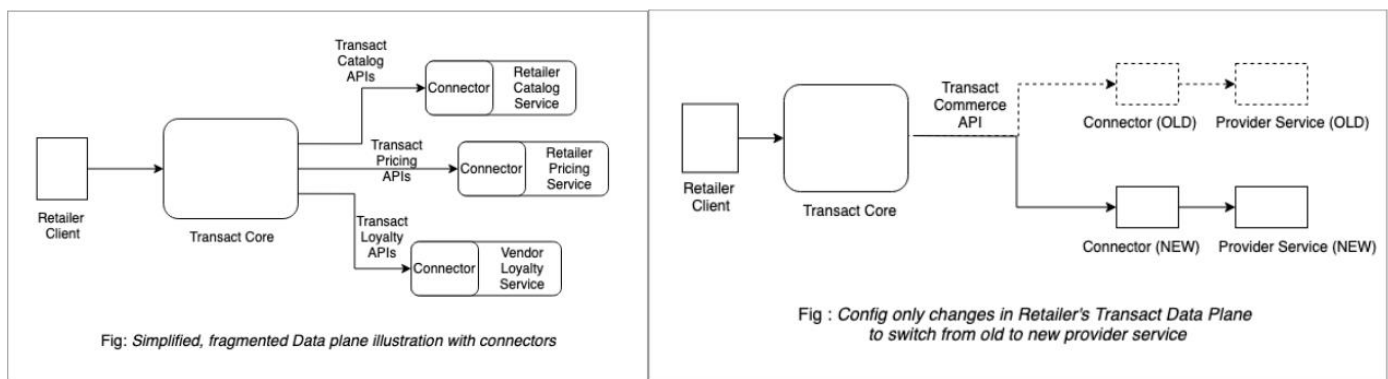
Retailers integrate their existing e-commerce system with Transact and Provider capabilities. This guide describes the details of how the Provider (vendor) constructs the Connectors to integrate Capability services

for Transact. When this integration is properly configured and built, a transactional paradigm is put in place. This is described at a very general and high level below.

First, a Shopper connects to the Retailer's Client App. The Shopper browses a product catalog. This catalog is presented by means of a Provider's **Connector**. When the Shopper selects an item from the catalog, it will be added to a cart object instanced via Client App webservice API calls to Transact. Transact creates a cart record, the Cart Document, which contains the selected item's detailed information. There may be other Provider connector calls from Transact to get the item's price, inventory status, applicable tax, shopper payment info, etc. This information is added to the cart document.

Transact may detect a Constraint Violation which blocks but does not abort the order. Typical kinds of Constraint Violations are events such as missing payment information, or item not in inventory, etc. Transact notifies the Retailer's Client App, which in turn must handle in some way the Constraint Violation. The Shopper's order can't be placed until all such Constraints are resolved. When the Shopper places the order, Transact creates a new Order Document (persistent record) of the transaction, and then Transact notifies both the Client App and the Provider's connector that the transaction has been processed. The Shopper can continue shopping. Note that the permanent record of the shopping transaction is maintained in Transact. The Retailer's Client App or the capability Provider may additionally make their own record of the transaction. Transact doesn't depend on any external transaction records.

As noted, the Retailer's Client App interacts via Transact with Provider capability Connectors. This interaction is described in greater detail later in this guide. However, it is necessary to understand that the Retailer has control of these Provider capabilities. Each capability has its own unique Connector for interaction with Transact. The Retailer can opt to implement some or all of these capabilities and connectors, or the Retailer can use an external Vendor to provide some or all capabilities and connectors. This flexibility is why Transact is characterized as 'composable e-commerce.' The diagrams below illustrate this flexible feature.



Each capability has its own connector. A connector can be replaced by another or newer one.

Resources

The following resources are necessary for Providers to construct a Connector for seamless integration with Transact.

AWS Account

An AWS account is required for the Retailer, but it is optional and highly recommended for the Provider. This AWS account will facilitate robust testing for the Provider. The Provider should create and use a public [AWS account](#). The Provider presents the AWS Account ID to the AWS channel Sales Representative or AWS Solution Architect. This AWS Account ID will be used to configure internal Transact resources and permissions, etc., and to create and configure the Provider's Transact Engine ID. The Provider will be given the unique Engine ID to allow their Connector to interface with Transact.

Connector HTTP Service

Transact TGS is headless middleware. Transact has no front-end user interface. The Retailer furnishes the front end UI as a client app, a website suitable for their shoppers to use. This website needs to have a front-end UI to allow customers to create a cart, browse a catalog, and place orders. The cart and orders are processed by interfacing with Transact. The catalog browsing, and order pricing, taxing and fulfillment services are capabilities accessed via Provided Connectors. Each built Connector is an HTTP Service. For each capability offered, a dedicated Connector must be built, tested and integrated. Providers are the vendor partners who offer these connectors.

Transact Retailers can optionally implement Connectors directly, or use capability service Connectors from the designated Provider(s). Whether built by Retailers or Providers, Connectors are servers that must be constructed as an HTTP Service.

Transact SDK

The [Transact SDK](#) is a software development kit used to build an e-commerce app or connectors for interfacing with Transact TGS. This SDK is a set of documents, software sources and related information necessary to interface with Transact or build the connector. The SDK software sources have a specific software language binding that is supported by Transact. The current Transact SDK is baselined in Java 2 (The Java 2 SDK), but other language bindings are also available.

Transact is internally defined in **Smithy**, a proprietary Interface Component Definition Language, as per the Open API specification. The Transact SDK can be configured with a variety of current and supported software languages. Transact maintains several current software development languages, and each supported language has a bound Transact SDK. The Provider can use any software development environment that can build a robust HTTP Service. However, if the Provider needs to build test apps, the Provider can build the test app with any of these supported languages. There is currently a growing set of software development languages, and many more are in various stages of completion, listed in the next sections. Contact your AWS Sales Representative or Solution Architect for more information.

Current Transact SDKs

The following Transact SDK software development language bindings are available. This guide uses the Java 2x SDK for general descriptions. Contact your AWS Sales Representative or Solution Architect to get the Transact SDK you prefer. The list of currently supported Transact SDK languages is below:

- Java-2x Transact SDK (the baseline SDK)
- Go
- JavaScript v2
- JavaScript v3
- AWS CLI v2
- Python and AWS CLI v1
- iOS
- Android

Future / Planned Transact SDKs

The following Transact SDK language bindings may be available at a later date. This guide uses the Java 2x SDK for general descriptions. Contact your AWS Sales Representative or Solution Architect to get the Transact SDK you prefer, and for availability of planned SDKs.

- C++
- Ruby V3
- .Net v3
- Kotlin
- Go v2
- PHP v3
- Rust
- Java v1

Retailer With Required Capabilities

Transact TGS is designed as Composable E-Commerce Middleware. The capabilities needed for shopping include a product Catalog for browsing, Item Pricing, Retail Taxing, Order Placing and Fulfillment, Promotions, and Returns, etc. If the Provider is NOT also the Retailer, the Provider needs access to a Transact Retailer for testing and integration.

The Retailer can develop their own provider capabilities available for integration with Transact TGS. Or the Retailer can select an eligible third-party Provider to make these capabilities available for integration as a service. However, the **Provider who is not a Retailer** must have a Retailer to partner with. The Provider offers essential capabilities. These capabilities are required to onboard a Retailer, whether from the Retailer itself or a designated **Provider** who offers them.

Transact TGS has a separate **Retailer Onboarding Guide** which describes how to interface with Transact. If you need this document, contact your AWS Sales Representative or Solution Architect. This onboarding guide is intended for Providers to describes how to construct capability Connectors for integration with a Transact Retailer.

It is possible for a Provider to develop Connectors as services for sale in a Transact marketplace. Or the Provider may need to partner with a Retailer to develop and test Connector capabilities. The Provider in this situation should contact their AWS Sales Representative or Solution Architect.

Multi-platform Deployment

This guide describes the common e-commerce website deployment for online shopping services. However, Transact is platform independent and it can be seamlessly deployed on several platforms. These platforms include:

- Websites (web e-commerce)
- In-store POS systems
- Mobile apps
 - iPhone
 - Android

Contact your Sales Representative or Solution Architect for additional information on these deployment options.

Prerequisites

The following elements **must be in place before** configuring your Connector and integrating with the Transact Gateway Service.

Active AWS Account

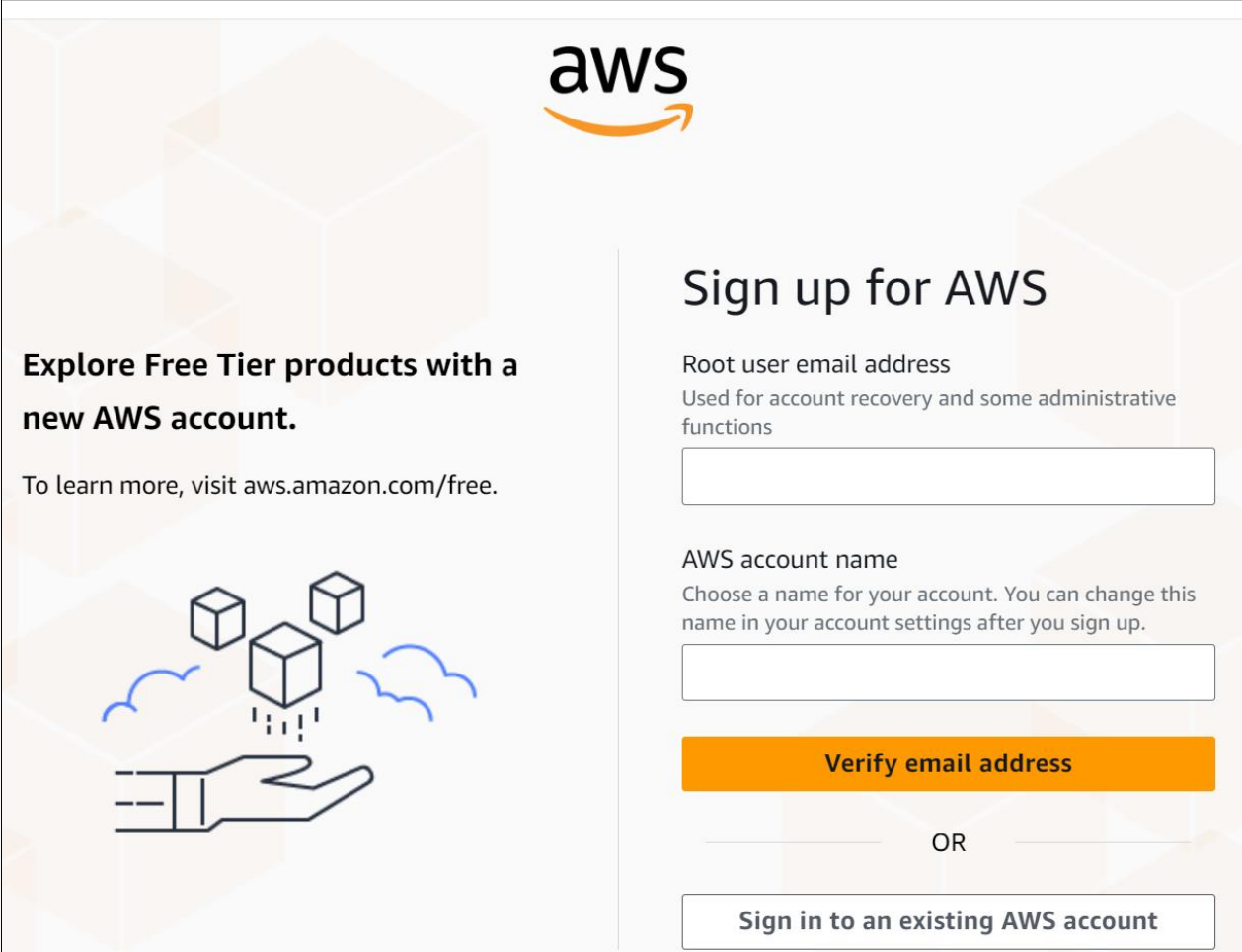
The first step the Provider can take is optionally configuring a public AWS account. This account is necessary to facilitate Connector testing. The Provider signs up for the account and provides the account information to the AWS Sales Representative of Solution Architect.

AWS Account Sign-Up

AWS accounts are free and readily available as a public resource. Navigate to this website to sign up for an account:

<https://aws.amazon.com/>

Click the button to sign up for your free AWS account and you will be presented with the sign-up page:



aws

Explore Free Tier products with a new AWS account.

To learn more, visit aws.amazon.com/free.

Sign up for AWS

Root user email address
Used for account recovery and some administrative functions

AWS account name
Choose a name for your account. You can change this name in your account settings after you sign up.

Verify email address

OR

Sign in to an existing AWS account

Encryption Key Option

All AWS account transactions are secure and encrypted. By default, AWS manages account encryption keys directly. If the Provider does not choose to use their own encryption key, Transact will pass an AWS owned key in its place. This key will live in the Transact Infrastructure Management Authority (TIMA) AWS account.

However, AWS allows users of Transact TGS to provide their own encryption key, referred to as the Customer Master Key (CMK). The CMK is also often referred to as the customer's **symmetric** key. This option allows Providers to have more fine-grained control over their AWS account's transaction usage.

The Provider secures their CMK independently of Amazon AWS as their own resource which is then provided to AWS in an authorized link. When AWS receives the CMK, it will create an Amazon Resource Name (ARN) for it. This selected CMK will be stored in the TIMA IOService as a data key. It will be used to create the Provider's Transact Engine ID and to secure all of the Provider's TGS transactions.

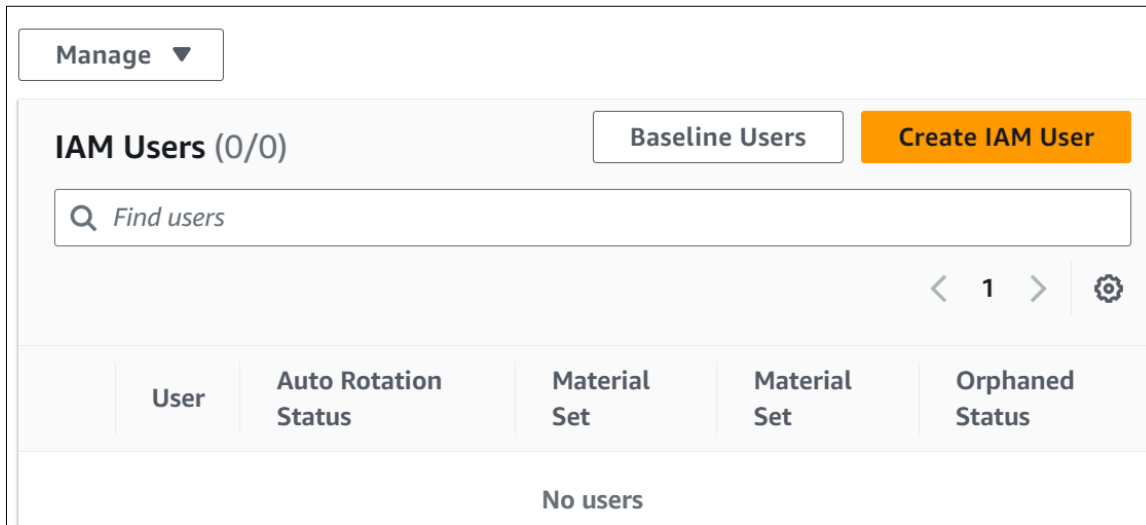
Transact Engine ID

After establishing an AWS account, the Provider needs to keep a secure record of the **AWS Account ID**. The Provider must also keep a secure record of the CMK encryption key, if that option is selected. The Provider needs to contact the AWS Sales Representative or Solution Architect and arrange to securely transmit the AWS Account ID and optional CMK resource link. If the Provider opts for their CMK, they **must** provide this link together with the AWS Account ID.

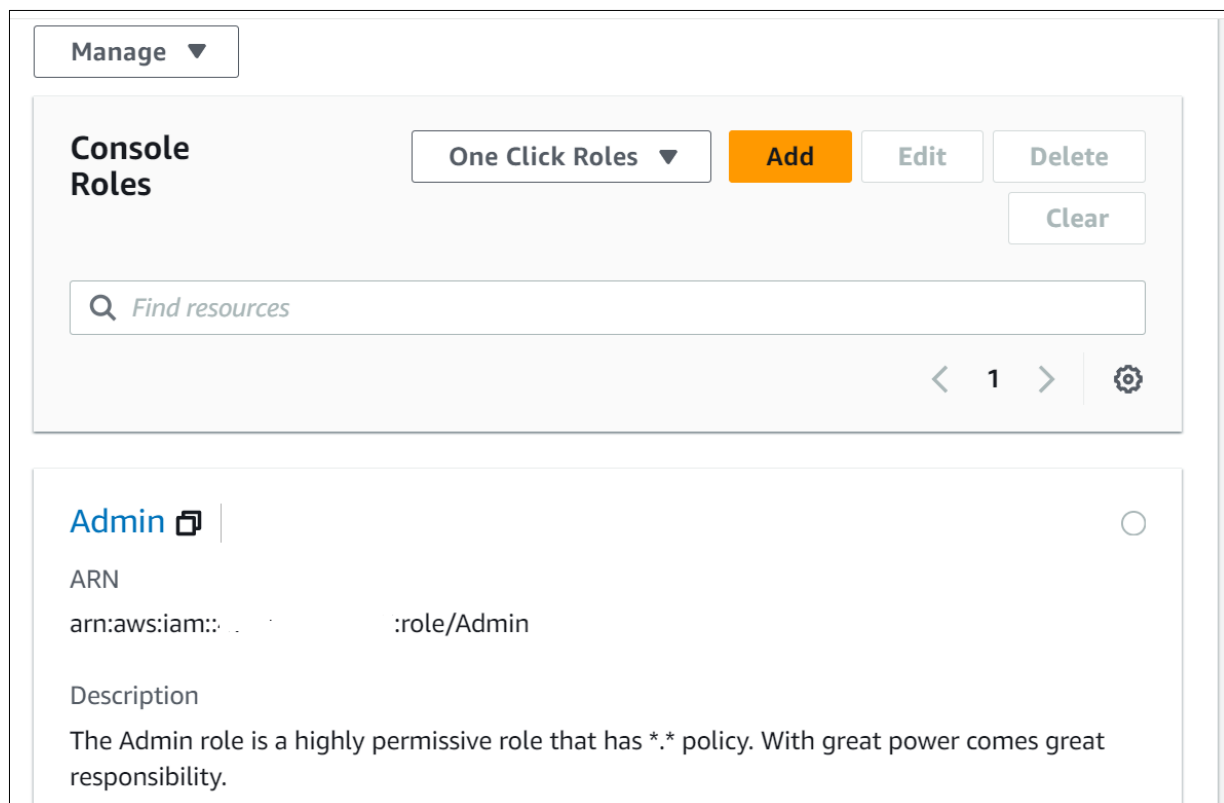
AWS will ingest the Provider's AWS Account ID and optional CMK resource link and launch an internal process that results in the creation of the unique Transact Engine ID. The AWS Sales Representative or Solution Architect will then pass this configured Transact Engine ID to the Provider, who needs to keep a secure record of it. The Provider uses this Engine ID for **all** Transact operations.

AWS IAM Account Settings

The Provider's AWS account needs to be configured as an Admin account using IAM authentication. The Amazon AWS IAM is an internal Identity Access Management facility that ensures the proper permissions and settings are in place for valid AWS account access. Be sure to setup the AWS account as an IAM user, as shown below and using the step-by-step procedure described.



The Provider must also designate the account as having the **Admin role**, as shown here:



IAM Role Step-by-Step Procedure:

Follow this step-by-step procedure for setting up the IAM role in the Provider's AWS account console:

1. Login to your AWS account where the Transact Engine is deployed, and navigate to IAM > Roles, and click on "Create Role".
2. Choose the "AWS Account" trusted entity type, keep "This account" selected, and click "Next".
3. Click "Next" on the first step, then create a name (e.g., "TransactAccessRole"), review the details, and create the role. Leave the trust policy generated unmodified.

4. Do not add permissions at this point; you can do this after the role is saved. Click "Save".
5. After the role is saved, search for the role name from step 3. On the role page, click "Add Permissions" and choose the "Inline Editor".
6. Switch to the JSON editor, copy and paste the below-provided JSON text into the editor pane.
7. Replace **<AWS_ACCOUNT_ID>** and **<ENGINE_ID>** with your own AWS account ID and Transact Engine ID, respectively. You can use "*" for the ENGINE_ID to create a universal role with access to all engines in the account.
8. Ignore any errors and click "Next", then add a policy name (e.g., "InlineTransactEngineAccess"). Click "Save".

Please follow the above steps carefully to create the role with the necessary permissions for accessing the Transact engine.

Below is the JSON text for the IAM role definition which allows access to your specified ENGINE_ID.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "transact:*"
      ],
      "Resource": "arn:aws:transact:us-east-1:<AWS_ACCOUNT_ID>:<ENGINE_ID>",
      "Effect": "Allow"
    }
  ]
}
```






Remember to replace **<AWS_ACCOUNT_ID>** with your Account ID and the **<ENGINE_ID>** with your Transact Engine ID. Do not keep the placeholder angle < > brackets.

The IAM auth setup is necessary for the Provider's Connector to access Transact. The AWS Sales Representative or Solution Architect can assist the Provider in applying these settings to the AWS account, as well as configuring the Transact Engine ID, if further assistance is needed.

Transact Supported Language and SDK

The Transact SDK is a folder of software development components specifically for Retailers and Providers. Retailers use the Transact SDK to add Transact to their client app. Providers use the Transact SDK to build capability connectors for use with the Retailer's client app. As noted, the Transact SDK is baselined with the Java 2 SDK. The components in this version were developed with the Java language. Other Transact SDK language bindings are available. This guide refers to the Java 2 version of the Transact SDK for descriptive purposes. The basic steps and issues are applicable to all Transact SDK language bindings. Your AWS Sales Representative or Solution Architect can provide you with a current version of your Transact SDK.

Here is a snapshot view of the top folder of the Transact SDK (Java 2 version):

0.0.4-20240618				
Name ^	Last Modified v	Size ^	Type ^	
 ReadMe.pdf	17 days ago	2.35 MB	PDF	
 Capability-APIs-Summary.pdf	17 days ago	60.32 KB	PDF	
 AWSTransactGatewayService...	17 days ago	199.04 KB	File	
 AwsJavaSdk-Transact-2.0.jar	17 days ago	1.11 MB	File	
 javadoc.zip	17 days ago	1.46 MB	File	

The Transact SDK is essential for both integrating your Client App and also for constructing the capability connectors. The **Readme.pdf** file provides a roadmap to using the SDK. It also has a high-level guide to building a client project using the **IntelliJ** development tool, a public resource available as a free download. See the [Software Tools](#) section of this guide.

The **Capability-APIs-Summary.pdf** file contains a table of current provider capabilities APIs and their uses. It is intended as an informative overview and not as a software reference.

The **AWSTransactGatewayService.Open API.json** file is the Transact Open API spec. This is a JSON text file that contains all API services currently implemented in Transact. This file is the definitive reference for Transact, but it is a versioned moving target and it is updated regularly with each iteration of the Transact SDK. You can load this file into software tools such as Visual Studio Code, Insomnia or Swagger for use as a dynamic technical software reference for your own development needs. See the [Software Tools](#) section of this guide. Below is an example of the contents of this file, opened in Visual Studio Code:
















```

1  {
2    "openapi": "3.0.2",
3    "info": {
4      "title": "AWS Transact Service",
5      "version": "2022-07-26"
6    },
7    "paths": {
8      "/carts": {
9        "get": {
10         "operationId": "ListCarts",
11         "parameters": [
12           {
13             "name": "shopperId",
14             "in": "query",
15             "description": "shopperId used to uniquely identify a shopper for the correspondi",
16             "schema": {
17               "type": "string",
18               "pattern": "^[A-Za-z0-9_-]+$",
19               "description": "shopperId used to uniquely identify a shopper for the corresp
20             }
21           }
22         ]
23       }
24     }
25   }

```

The **AwsJavaSdk-Transact-2.0.jar** file is the compiled bytecode version of the Transact SDK Java classes. Do **NOT** edit or manipulate this file. Add it *as is* to your software project, and add its fully qualified path name to your app's Classes path setting. When opened in a properly configured Java development environment, the classes in the jar file will be visible for developers to instance and use. This is described in more detail in the [Client App](#) section of this guide.

The **javadoc.zip** file has the compiled Javadoc utility's HTML references of the Transact SDK Java classes. This file can be viewed with the Windows file manager, or tools like 7-Zip, etc. Click each reference for browser viewing. You can drill down into each HTML reference for more information about the Java classes. This reference is updated with each iteration of the Transact SDK. An example listing is shown below:

Name	Type	Compressed size	Passw
 software	File folder		
 allclasses-frame.html	Chrome HTML Document	5 KB	No
 allclasses-noframe.html	Chrome HTML Document	5 KB	No
 constant-values.html	Chrome HTML Document	3 KB	No
 deprecated-list.html	Chrome HTML Document	1 KB	No
 help-doc.html	Chrome HTML Document	3 KB	No
 index.html	Chrome HTML Document	2 KB	No
 index-all.html	Chrome HTML Document	74 KB	No
 overview-frame.html	Chrome HTML Document	1 KB	No
 overview-summary.html	Chrome HTML Document	2 KB	No
 overview-tree.html	Chrome HTML Document	13 KB	No
 package-list	File	1 KB	No
 script.js	JavaScript Source File	1 KB	No
 serialized-form.html	Chrome HTML Document	6 KB	No
 stylesheet.css	Cascading Style Sheet Do...	3 KB	No

Client App e-Commerce Website and Development Project

The Retailer must have full access to a Client App developed in a software language that is compatible with the provided [Transact SDK](#). The Retailer needs to configure the Client App for seamless integration with the Transact SDK. This [integration](#) is described later in this guide.

The Retailer must also have access to one or more Transact Capability Connectors. For example, a key capability is the Catalog, which allows for browsing products and selecting them in a cart for placing orders. These essential capabilities must be provided directly by the Retailer, or selected as a Provider by the Retailer. What must be understood is that each capability must have its own customized Transact Connector. These capabilities include components such as:

- Catalog (Product)
- Pricing
- Taxes
- Promotions
- Fulfillment
- Returns
- Etc.

[Integrating these connectors](#) is described later in this guide

As a rule of thumb, the Retailer's Client App is for REQUESTS. Each Provider Connector is for RESPONSES. Transact is the middleware that manages the operations. The Retailer will need to integrate and interface with the Transact SDK and also the capability connectors, which will entail editing the Client App source code, compiling, testing, debugging and deploying the development project.

When successfully done, the Retailer's Client App will be ready for shoppers to place orders, etc. The Provider may wish to implement a mock Client App for testing purposes.

Software Tools (Optional But Useful)

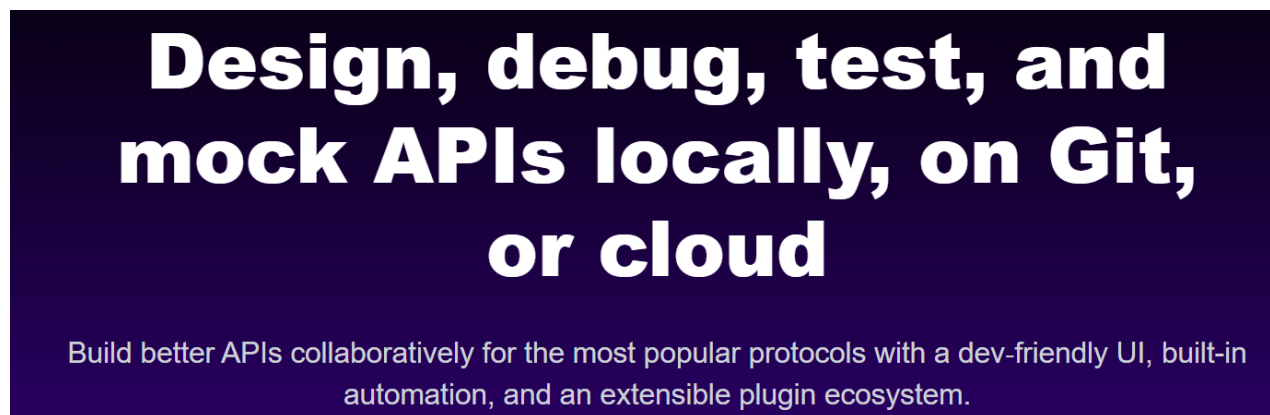
It is necessary for the Retailer to have a functioning Client App to integrate with Transact and provider capability connectors. This Client App could typically be an e-commerce website, which the Retailer would need to develop and maintain. The Provider needs a development environment able to build an HTTP Service for each Connector. The Provider may also wish to build a Client App for testing needs. There are many software tools and resources available for these purposes.

Some of these technologies and tools are established, reliable and have open-source licensing. For example, Oracle makes Java available for download and use on such a basis. Git, the Git Desktop, and Git for Windows, etc. are similarly available. There are also specific software tools that are optimized for web services functionality and are in use by Amazon AWS developers.

This section has an annotated list of very useful, widely available open-source tools. They are described here not as an endorsement or even recommendation. Rather, they are described here because the Provider's or Retailer's staff may find them useful. These tools are NOT required to use with Transact. Their use is optional and at the Provider or Retailer's discretion.

Insomnia

This is an open source, licensed tool available for free download and use. The company describes it this way on its splash screen:



It is a very functional and easy to use tool, and it is particularly effective for working with web services API calls. It is similar in scope to Postman, but it allows users to have their authentication credentials integrated in its use, which Postman does not. Insomnia is highly configurable and very flexible.

You can load the Transact Open API spec (the **AWSTransactGatewayService.Open API.json** file) into Insomnia for direct testing. Here is the main website for Insomnia:

<https://insomnia.rest/>

IntelliJ IDEA

IntelliJ IDEA is a powerful development IDE focused primarily on enterprise scale Java or Kotlin development. The parent company has no direct association with Oracle. It is available for download and use with an open-source license. The corporate owner, JetBrains, also has paid licensing programs with additional features and benefits.

If your main software development language is Java, IntelliJ is definitely worth considering. IntelliJ can also be readily used to manage multiple development projects. Note that Transact currently does not have a Transact SDK bound to Kotlin, but does have such a binding [planned for a future release](#). Here is the IntelliJ corporate website:

<https://www.jetbrains.com/idea/>

Postman

Postman is a widely used tool for website and web services APIs development and maintenance. It is available for download and use with an open-source license. On their website splash page, they describe Postman this way:

Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration so you can create better APIs—faster.

Postman has a host of tools for API development and managing API repositories. It is language independent and works well with Git. Because it is so widely used, many add-on extensions are available, and it has a large developer support community. Here is the corporate website:

<https://www.postman.com/>

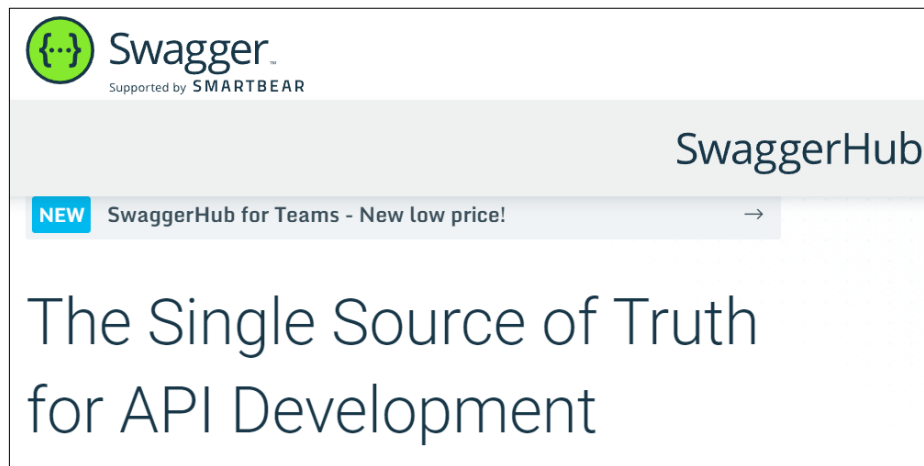
Swagger

Unlike the other software tools in this section, Swagger is not free. In fact, it is a very expensive product. Swagger can be downloaded on a 'free' trial basis. But the license is enforced and it is not open-source. Swagger is a full life-cycle suite of tools and facilities for designing, developing, implementing, deploying, documenting and maintaining enterprise APIs. Swagger is a member of the [Open API foundation](#), and support for the [Open API Specification \(OAS\)](#) and RESTful APIs is built into the product.

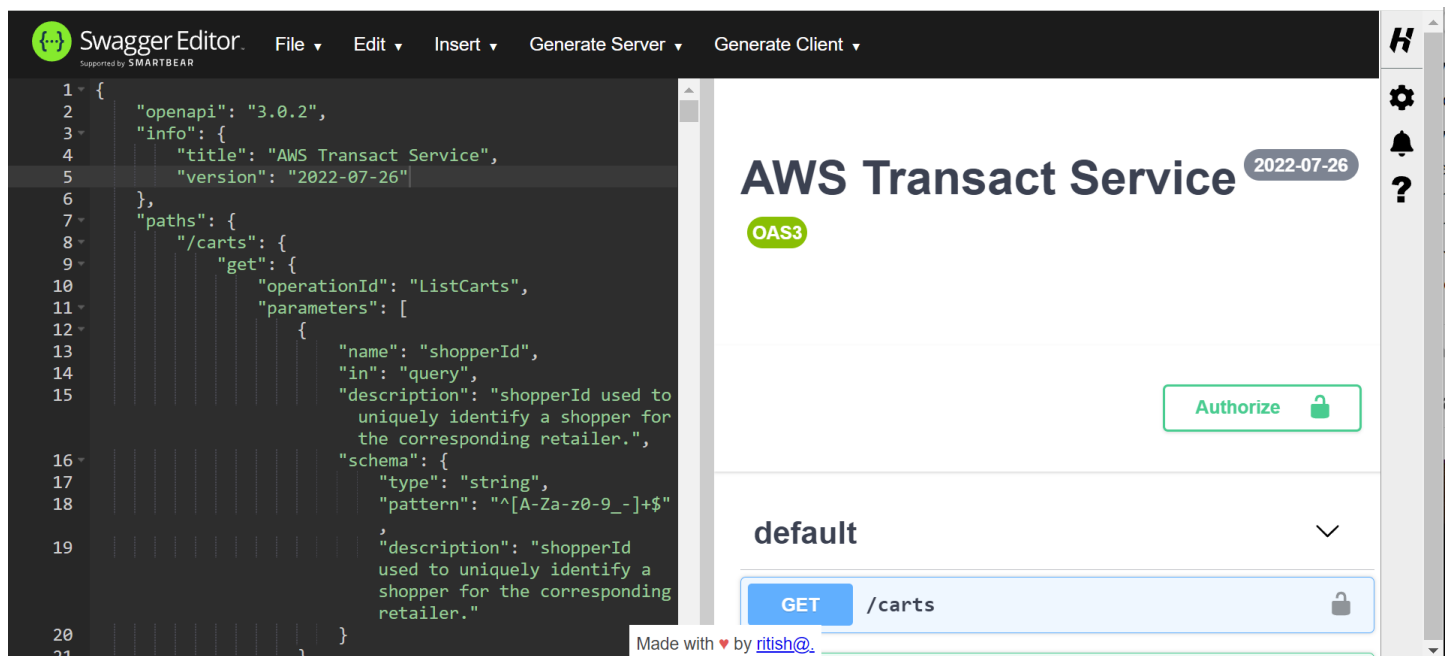
Swagger can be found at this website:

<https://swagger.io/>

The product also includes a development hub with a large community of users. The hub product's splash screen describes it this way:



Swagger is very powerful with a market-leading feature set. The **Swagger Editor** is the core product in the tool suite. Swagger can be found here: <https://editor.swagger.io/> Below is an example of the Transact Open API spec loaded into the Swagger Editor, ready for work:



To view the APIs in an easy readable format, navigate to the Swagger Editor at <https://console.harmony.a2z.com/swagger-editor/> and copy the content from above json file and paste it into the editor. This will show different resources like Cart, Order, Catalog, etc. with PUT, POST operations. Note that this resource is behind Amazon's firewall and it requires Amazon credentials to access it. However, Retailers can choose to acquire Swagger and take advantage of this tool.

AWS is a licensed user of Swagger. Once again, Amazon makes no endorsement or recommendation of Swagger. Providers and Retailers do NOT need to use Swagger to build and maintain their Connectors or Client App, but they may find it useful.

Visual Studio Code

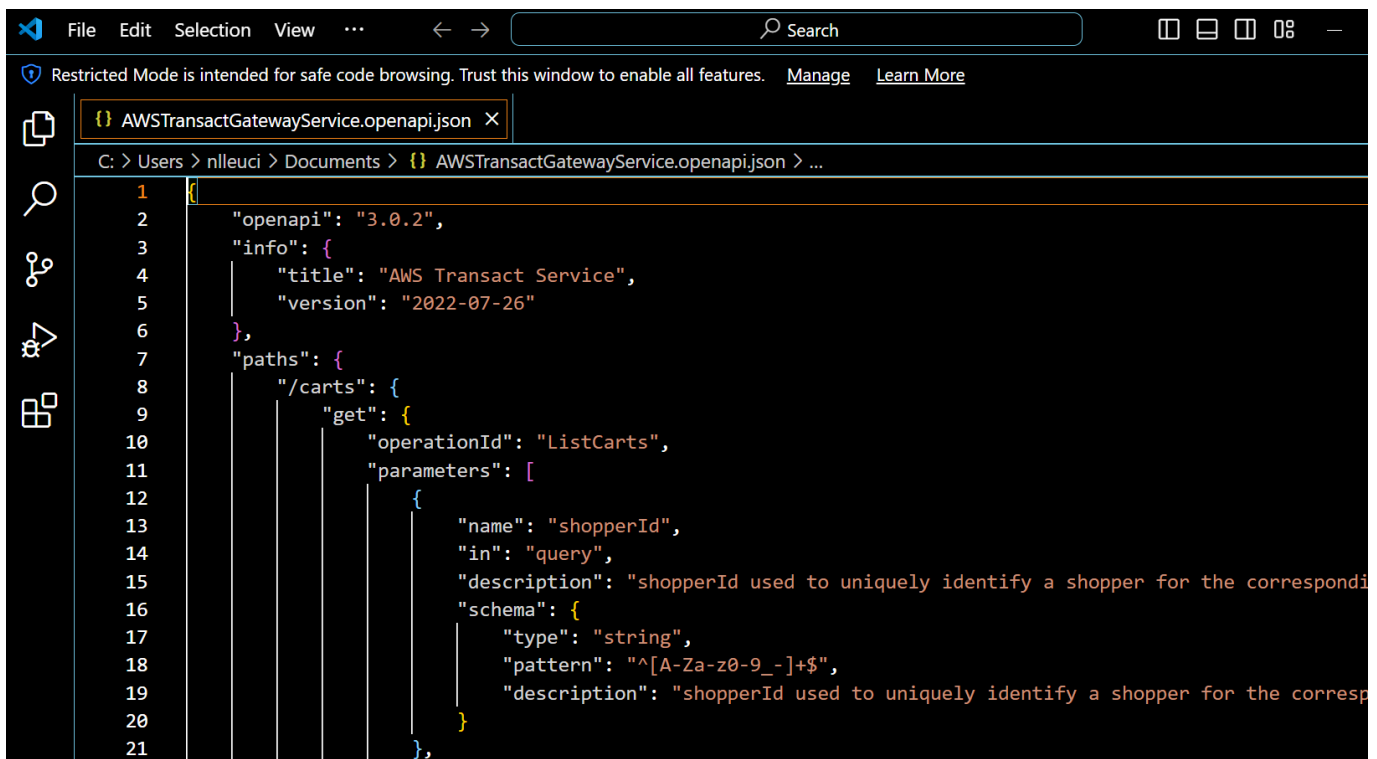
Visual Studio Code is a freeware software development and editing tool from Microsoft. It is available for many platforms including Apple iOS, Linux, and others, as well as Microsoft operating systems. Visual Studio Code is not to be confused with Microsoft's Visual Studio.Net development environment. The two products are independent of each other.

When installed, Visual Studio Code natively supports Markdown and HTML development. All Markdown source can be presented in HTML with a button click. Visual Studio Code is backed by Microsoft in a major way. There are many language plug-ins and extensions available from Microsoft and the wider development community. It has a huge installed base of users. It can be used to develop apps in Java, JSON, Python, and many other contemporary languages.

Visual Studio Code seamlessly integrates with Git and it can be readily adapted to Git workflows. Visual Studio Code's internal OS shell can be configured as a Bash shell. If the Git desktop environment is installed on the development platform, the Visual Studio developer can work exclusively in Bash.

Visual Studio Code is available for download and use on an open-source license. Here is the corporate website: <https://code.visualstudio.com/>

Below is an example of the Transact Open API spec loaded into Visual Studio Code:

A screenshot of the Visual Studio Code editor interface. The top menu bar includes File, Edit, Selection, View, and a search bar. Below the menu, a status bar indicates 'Restricted Mode is intended for safe code browsing. Trust this window to enable all features.' The main editor area displays a file named 'AWSTransactGatewayService.openapi.json'. The file content is a JSON document representing an OpenAPI specification. The JSON is formatted with syntax highlighting and line numbers on the left. The visible portion of the JSON includes the 'openapi' version, 'info' (title and version), and the start of the 'paths' section, specifically the '/carts' endpoint with a 'get' method. The 'parameters' array for the 'get' method is partially visible, showing a parameter named 'shopperId' with a query type and a specific pattern and description.

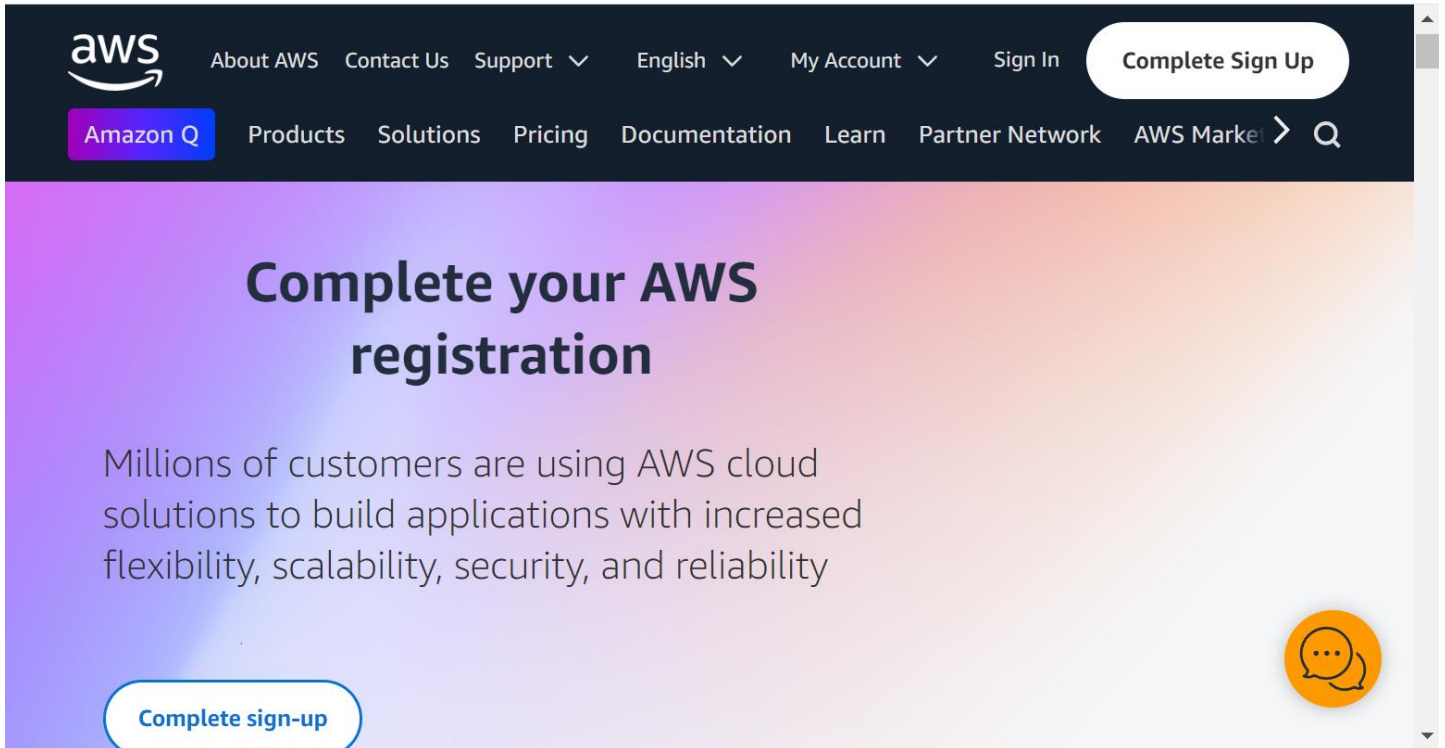
```
1 {
2   "openapi": "3.0.2",
3   "info": {
4     "title": "AWS Transact Service",
5     "version": "2022-07-26"
6   },
7   "paths": {
8     "/carts": {
9       "get": {
10        "operationId": "ListCarts",
11        "parameters": [
12          {
13            "name": "shopperId",
14            "in": "query",
15            "description": "shopperId used to uniquely identify a shopper for the correspondi",
16            "schema": {
17              "type": "string",
18              "pattern": "^[A-Za-z0-9_-]+$",
19              "description": "shopperId used to uniquely identify a shopper for the corresp",
20            }
21          }
22        ]
23      }
24    }
25  }
26 }
```

Public AWS Websites

There are three Amazon AWS website resources that will be quite useful for webservicess developers, including those building Connectors. These resources are public websites readily available for anyone who

wants to build solutions using AWS. Users can sign up for free access with some additional paid options. These websites are:

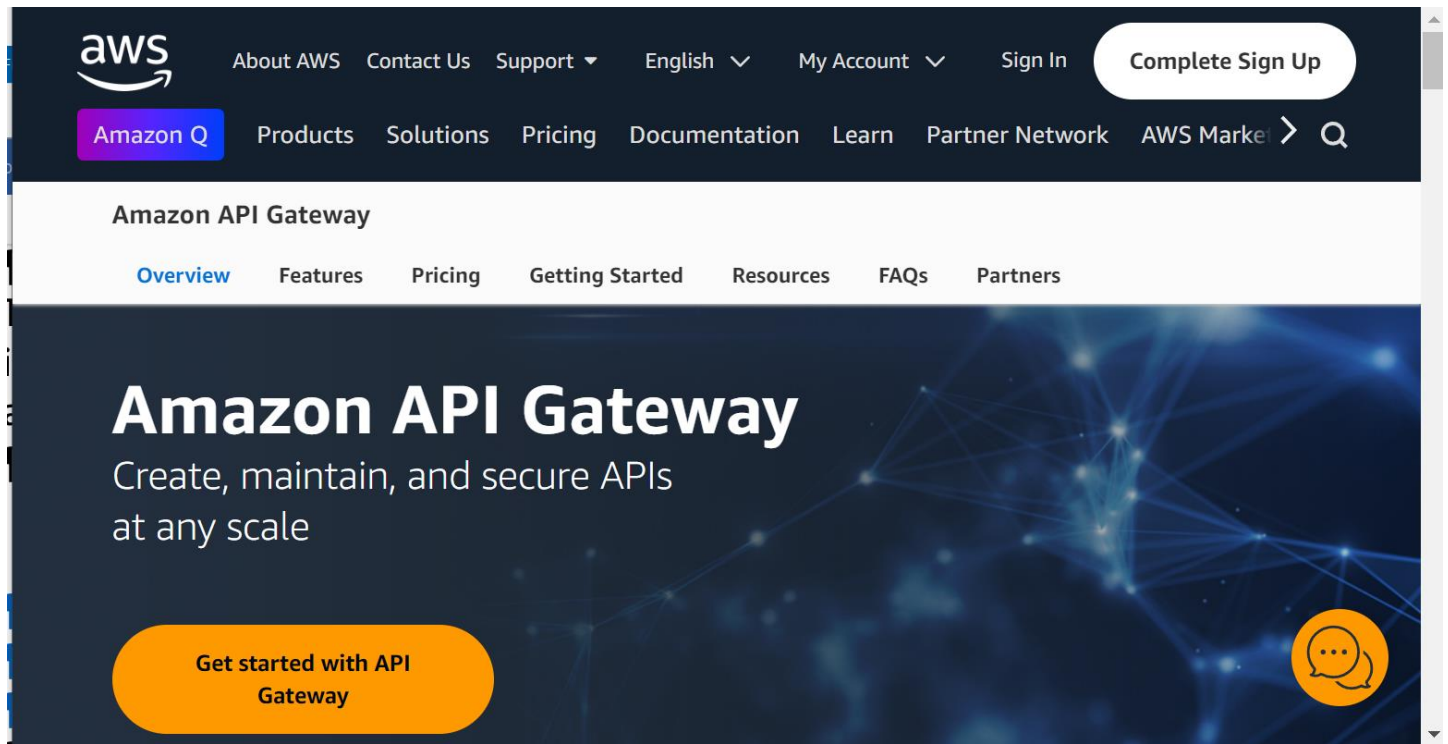
1. The AWS main site
<https://aws.amazon.com/>



The AWS main site is used in concert with your [AWS account](#), and it points to all public AWS features and support information. After completing your AWS account set up, the next step is to navigate to this site and take advantage of all that AWS has to offer to users and developers of AWS webservices.

2. The Amazon AWS API Gateway

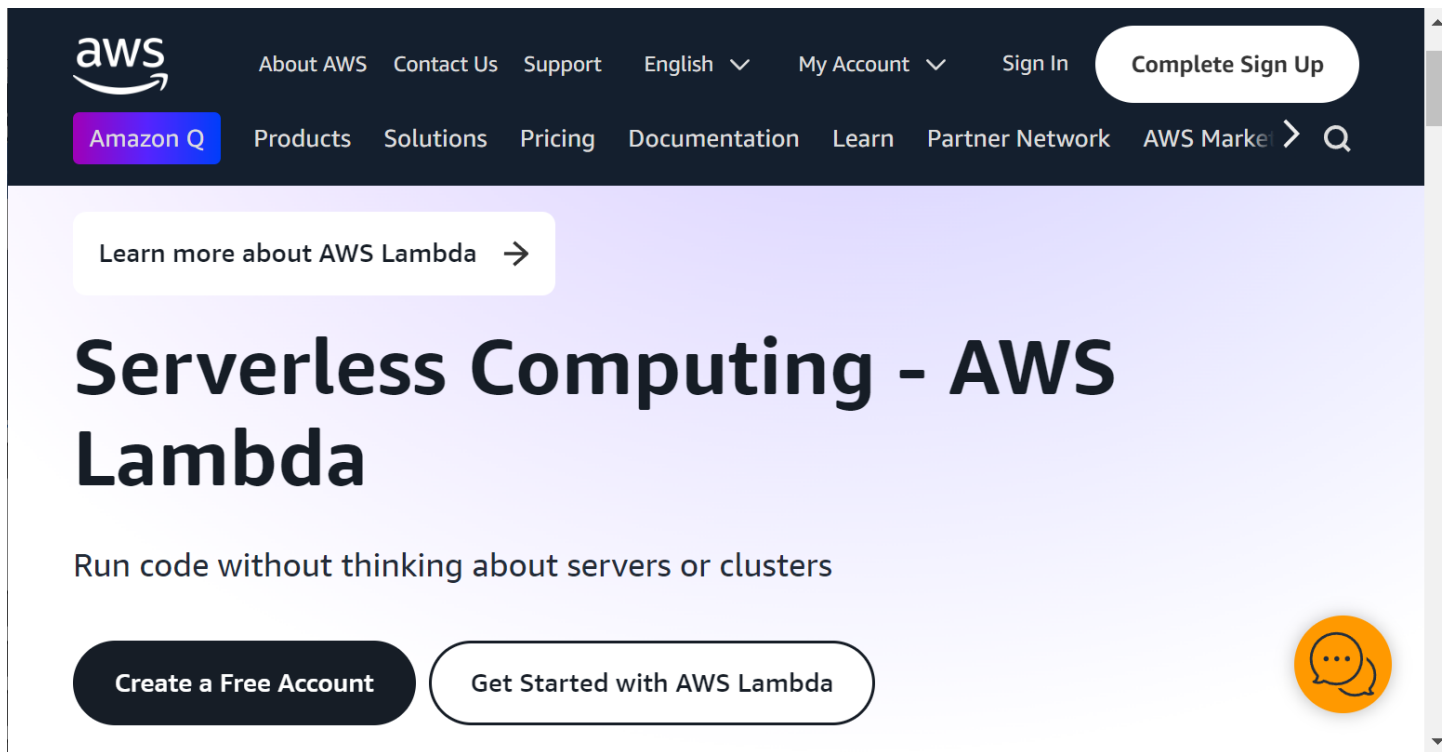
<https://aws.amazon.com/api-gateway/>



The AWS API Gateway is a full featured resource for API developers. It is intended to provide AWS webservers developers a complete solution for designing, implementing, testing and maintaining AWS webservers applications. There are no specific software language dependencies, except that it functions with HTTP request / response compatibility. You can configure your AWS account (the 'AWS console') to work with AWS.

The use of the AWS Gateway is not required for Providers who need to build Transact Connectors, but they will find it very helpful, especially for testing API webservers.

3. The AWS Serverless Resource, **AWS Lambda** <https://aws.amazon.com/pm/lambda/>



The AWS Lambda website is an elegantly simple way to use a serverless environment to test any webservice function. **It allows developers to execute code without setting up a server.** The developers drop in their code in a zip file, check some settings, and the service is ready to go. AWS Lambda also has a simple text editor that is JavaScript friendly. Some versions of Java are supported, but the built-in editor does not support Java itself. Use the zip drop-in to load and run Java code. AWS Lambda also will work with Python, Ruby and other languages. Check the site for software language specifics.

AWS Lambda can be used independently of API Gateway. However, these two websites can be optimally used together. The AWS API Gateway is used to develop, test and manage AWS webservices. Set up the API Gateway to invoke an AWS Lambda function. The API Gateway works as a **listener** for HTTP requests. It will call the AWS Lambda function to act on the request and return the response.

Transact does not require Providers to use these three AWS websites. However, these sites are intended for testing API webservices by making them easy to set up and execute. They are extensively used internally by Amazon AWS developers and testers.

Interfacing Client App / Connector With Transact

The task of building and operating the Client App for Transact rests with Retailers. Providers build and operate Connectors that work in concert with Retailers and Transact. A Provider can't be deployed without a Retailer. Also, a Provider may choose to build and operate a Client App either as a complete solution or as a mock testing facility. For these reasons, a minimal description of the Client App here is necessary for the Provider. However, the detailed procedure of how to configure and integrate the Client App is out of scope for this guide. The AWS Transact team does have a **Retailer Onboarding Guide** which does have the detailed procedure for the Client App. For those Providers who do need it, contact your AWS Sales Representative or Solution Architect for the Retailer Onboarding Guide.

Once all of the prerequisites have been addressed, the Retailer is ready to perform the crucial task of modifying the Client App so that it can interface with Transact. This can't be readily described in a general way. The Transact SDK is bound to a specific software language and development context. The Transact SDK must be compatible with the Client App development language and environment.

Since the baseline Transact SDK is focused on Java 2, this guide will use that software language and development environment as a basis to describe integration with the Client App. Even though a specific software language is referenced here, the process is conceptually the same for any other Transact SDK software language binding.

The Client App is assumed to be a fully functional e-commerce web-site. This guide does not indicate how to describe the feature set needed for an e-commerce website, or how to implement it. This discussion assumes the Provider or Retailer has this functionality already and is now ready to integrate with (or plug-in) to the Transact Gateway Service middleware. Succinctly, Transact has no UI, as the Provider or Retailer must use their existing UI to integrate with Transact.

Configuring The Client App

A summary procedure to get started integrating the Client App with Transact is detailed below. The Provider or Retailer conceptually needs to do the following in order to Interface with Transact:

1. Add the Transact SDK to the Client App development project.
 - a. Add the Transact SDK **jar** file (***AwsJavaSdk-Transact-2.0.jar***) to the development project.
 - b. Add the fully qualified path and file name of the Jar file to the project's **Classes** setting.
 - c. Add an **import** statement to put the Transact Java classes in scope.
 - d. Rebuild the Client App so that the build output is error free and ideally warning free
 - e. Visually confirm in the build environment that the Transact Java classes are in scope
2. Find the right location in the Client App to insert a **CreateCart** method call.
 - a. Make sure the selected code module has the **import** statement (see 1.c)
 - b. Find the effective place to insert the **CreateCart** method call with the necessary inputs and types, and the handling for the call's return code.

- c. Rebuild the Client App so that the build output is error free and ideally warning free
3. Verify in the runtime code and the UI that a **Cart** object has been instantiated.
4. Add the Catalog capability connector to the project.
5. Verify in the runtime code and the UI that a **Catalog** connector object has been instantiated.
6. Create a dummy order to select an item and make a call to the **CreateOrder** method.
7. Verify in the runtime code and the UI that an **Order** object has been instantiated.

The following sections highlight key considerations for the Provider or Retailer to consider in building out their integrated Client App.

Transact API Webservice Calls

The AWS Transact Gateway Service offers a suite of API web services for e-commerce transactions. The Client App must provide a context for all of these APIs to be called and handled as needed. Using the example from the procedure in the previous section, the Provider or Retailer needs to do the following:

1. Define a working application making appropriate use of the various Transact APIs.
2. Use the **TransactClient** class to locate and call from the Client App each web service API call.
3. Handle each API call's HTTP return code.
 - If the Return is an error, the transaction will be aborted and the Client App must recover.
 - If the Return is valid, proceed to the next logical step in the Client App design.

Below is a comprehensive list of Transact API web services. This list is growing and is subject to change. These APIs cover all Transact e-commerce functional areas. They include:

Core APIs:

- **Cart APIs**
 - CreateCart
 - CreateCartLineItems
 - GetCart
 - UpdateCart
 - DeleteCart
 - ListCarts
- **Order APIs**
 - CreateOrder
 - CreateOrderLineItems
 - GetOrder
 - UpdateOrder
 - UpdateOrderFulfillments
 - UpdateOrderReturns X

- UpdateSignedOrder X
- DeleteOrder
- ListOrders
- SignOrder
- CancelOrder X

Capability APIs:

- **Catalog APIs**
 - GetProduct
 - BatchGetProduct
- **Pricing APIs**
 - GetProductPrices
 - GetProductPricesForLineItems
- **Tax APIs**
 - GetTaxesForLineItems
- **Fulfillment APIs**
 - FulfillOrder
 - CancelLineItemFulfillment
- **Inventory APIs**
 - GetItemsInventoryRequest
 - AdjustReservedInventoryRequest
 - AdjustAvailableInventoryRequest
- **DeliveryPromise**
 - GetDeliveryPromiseRequest
- **Promotions APIs**
 - *pending*
- **Returns APIs**
 - *pending*

Handlers For Exceptions and Constraint Violations

All of the API calls in the previous section will generate a successful HTTP return code, or an HTTP Error code. All Error codes mean the initiated transaction has completely **aborted**. The Client App **must** handle both successful and error returns from each Transact API call. An aborted API webservice call should NEVER result in a Client App crash. The Retailer must carefully and robustly design and implement a strategy to handle any API error return.

A Detailed list of all [Transact HTTP error codes](#) is provided in this guide.

Additionally, many API webservice calls may cause a **Constraint Violation** during the operation. A Constraint Violation means that the transaction is **temporarily blocked but NOT aborted**. This block remains in effect until all Constraint Violations are handled and resolved. When that occurs, the transaction continues.

When Transact detects a Constraint Violation, for example a missing payment provider, it will block the transaction benignly: it won't abort, alter or remove any data. It will signal the Client App of the Constraint Violation and wait for the Client App to resolve the Constraint Violation. The Client App needs to provide handlers for any kind of Constraint Violation for any API call. How the Constraint Violation is handled is up to the Provider or Retailer. Whatever action is taken will be signaled to Transact, and Transact will then re-evaluate the current transaction. When any and all Constraint Violation blocks have been lifted, the transaction will resume.

The following kinds of Transact API webservices may result in a blocking Constraint Violation:

Transact Core API Services

- **CART**
- **ORDER**

Capability API Services:

- **CATALOG**
- **PRICING**
- **TAX**
- **FULFILLMENT**
- **INVENTORY**
- **DELIVERY_ESTIMATE**
- **RETURNS**
- **PROMOTIONS**
- **LOYALTY**

An expanded reference of [Constraint Violations](#) and their identifiers is provided in this guide.

Integrating the Provider Capability Connectors

Transact has core API webservices such as Cart and Order APIs. Transact also has capability API webservices. The Retailer needs to handle core services in the Client App. The Retailer needs to handle capability services as well. Capability services are invoked from the Client App but are handled via **Connectors**. It is important to understand that EACH capability API must have its own dedicated connector. A Provider implements and integrates each API Connector.

Below is a comprehensive list of Transact APIs. This list is growing and is subject to change. These APIs cover all Transact e-commerce functional areas. They include:

Core APIs: Implemented in the Retailer Client App

- **Cart APIs**
 - CreateCart
 - CreateCartLineItems
 - GetCart
 - UpdateCart
 - DeleteCart
 - ListCarts
- **Order APIs**
 - CreateOrder
 - CreateOrderLineItems
 - GetOrder
 - UpdateOrder
 - UpdateOrderFulfillments
 - UpdateOrderReturns X
 - UpdateSignedOrder X
 - DeleteOrder
 - ListOrders
 - SignOrder
 - CancelOrder X

Capability APIs: Implemented in the Provider API Connector

- **Catalog APIs**
 - GetProduct
 - BatchGetProduct
- **Pricing APIs**
 - GetProductPrices
 - GetProductPricesForLineItems
- **Tax APIs**

- GetTaxesForLineItems
- **Fulfillment APIs**
 - FulfillOrder
 - CancelLineItemFulfillment
- **Inventory APIs**
 - GetItemsInventoryRequest
 - AdjustReservedInventoryRequest
 - AdjustAvailableInventoryRequest
- **DeliveryPromise APIs**
 - GetDeliveryPromiseRequest
- **Promotions APIs**
 - *pending*
- **Returns APIs**
 - *pending*

The Retailer has a choice to directly implement some or all capability APIs via connectors, or to integrate some or all capability APIs with Vendor (Provider) connectors. Transact is neutral about whether the Retailer or Vendor provides the connectors. Both scenarios are implemented the same way. Only the Transact role of the implementer differs. It is key to understand that the Client App has no awareness of any Connectors. Transact is the middleware that handles both. All Connectors are pluggable components in the Transact composable architecture. The Client App should always work the same whenever an API Connector is replaced.

Vendor (Third Party) API Capability Connectors

Each capability API connector is a constructed HTTP access point. It functions as a server for the capability API, using [IAM](#) authentication. The Retailer needs to capture the URL for each Provider (Vendor) API capability connector and make a secure record of it along with all pertinent details. The Retailer must contact the AWS Sales Representative or Solution Architect to securely pass the Provider (Vendor) connector URLs for integration with Transact TGS.

The Retailer and the Provider (Vendor) are both responsible for the proper testing and robust use of all such API capability connectors. Transact will pass Client App requests to the Connectors and then pass Connector responses back to the Client App.

The Retailer can also choose to directly implement one or more or all capability Connectors. Each connector is an HTTP access point and it must be constructed by a development technology that can build an HTTP service. Each connector must use [IAM](#) authentication. In this scenario, the Retailer is also a Provider, and the retailer would use this guide as a Provider.

How an API capability Connector is built or how it works is the same to Transact, regardless of who (a Retailer or a Provider) implemented it. A Retailer can use multiple Providers, or the Retailer can directly implement each connector, or any combination, as long as each API capability service has its own dedicated Connector.

Resources for Building Provider (Vendor) API Capability Connectors

From this point on in the guide, the focus is on constructing the API capability Connector. The methodology is the same for a third-party Vendor Provider or for a Retailer who is also a Provider.

There are two approaches to build the Transact API capability Connector, and both are described here. They are:

- Using the [Transact Open API Spec](#)
- And using the [Transact SDK](#)

CAVEAT: Both approaches describe hands-on steps that refer to basic webservices technology and concepts. It is recommended to have developers with some webservices experience use this guide to build the API capability Connectors.

There are three Amazon AWS resources that will be quite useful for building Connectors. These resources are [free public websites](#) readily available for anyone who wants to build solutions using AWS. They are:

1. The **AWS** main site
<https://aws.amazon.com/>
2. The Amazon AWS **API Gateway**
<https://aws.amazon.com/api-gateway/>
3. The AWS Serverless Resource, **AWS Lambda**
<https://aws.amazon.com/pm/lambda/>

The AWS main site is the principal starting point for all AWS public information and support. The AWS API Gateway is used to develop, test and manage AWS webservices. AWS Lambda is an elegantly simple way to use a serverless environment to test any webservice function. Set up the API Gateway to invoke an AWS Lambda function. The API Gateway works as a **listener** for HTTP requests. It will call the AWS Lambda function to act on the request and return the response.

The procedures described below do not require Providers to use these three AWS websites. However, these sites are very useful for testing API webservices.

The AWS Golden Path

Amazon AWS has an internal process for building webservices. This is often referred to as the *Golden Path*. It requires an Amazon AWS account and permissions to access this internal website:

<https://code.amazon.com/>

If you have the requisite access, go to the above, request a service, select the API Gateway after setting it up for AWS Lambda. The API Gateway is a listener for HTTP requests; so it will call the AWS Lambda function to act on the request and return the response. This *Golden Path* is the preferred AWS **internal** way to test,

Using the Open API Spec to Build the Connector

The entire Transact SDK is not necessarily needed to construct a Connector. The **Transact Open API Spec** itself is useful for building a Connector. This is the [JSON file located in the Transact SDK](#) (**AWSTransactGatewayService.Open API.json**). Extract this JSON file from the Transact SDK to start this procedure.

Follow and implement this procedure in the described order:

Step	Owner	Description
1. Download the latest Transact Open API Spec	Provider	Obtain the latest Transact Open API specification file in JSON format from the Transact Team
2. Extract out particular capability webservice from the Transact Open API Spec	Provider	Extract out a specific capability API spec webservice from Open API Spec for which you will create a providerAPI with connector logic
3. Create and import your own Open API spec file with extracted information for the capability in your project	Provider	Ensure you have the necessary project structure to store extracted Open API spec file
4. Add Open API Generator Plugin of your choice. Example - Swagger Codegen	Provider	Configure the Open API generator plugin. Specify the input specification file location, generator name, output directory and package names.
5. Generate Server Stubs	Provider	Execute the Open API generator plugin and generate the server stubs. Verify generated server stub code is available in output directory.
6. Implement Application code for your project	Provider	Define controllers, services and other components. Use the generated server stubs by importing them into your controllers or services
7. Implement connector logic for your provider API	Provider	Add the connector logic to convert generated Models, which adheres to the Open API standards, into models that are specific to your provider API and vice-versa i.e., from provider models to Open API models.

8. Test your project application locally	Provider	Run your application and test the provider API endpoints to verify that the integration with generated server stubs and connector logic are functioning as expected
9. Deploy your provider capability API	Provider	Deploy your provider API for capability to any AWS infrastructure of your choice
10. Create an IAM Role	Provider	<p>Create IAM Role for provider API in order to <i>allowlist</i> Transact Service</p> <ol style="list-style-type: none"> Policy - Attach full access policy of the infrastructure to which your provider API for capability is deployed Trust Relations - Add the following accounts and services URL for Transact so that Transact can assume this Role to invoke your provider API for capability.
11. Transmit the provider API information to Transact Team	Provider	Transmit provider Name , Base Endpoint , Auth Strategy Type , RoleArn , SignRegion, SignService, CustomAttributes informations to Transact Team
12. Add the configuration for the Provider API	Transact Team	Add the configuration given by Provider to a particular retailer engine id so that particular retailer will use Provider capability API to fetch capability information
13. Deploy the retailer and Provider configuration	Transact Team	Deploy this updated retailer configuration to particular Transact Services i.e., the AWSTransactGatewayService and the AWSTransactCapabilityService
14. Provide Retailer information in order to test new Provider API	Transact Team	Provide the retailer engine id endpoint and auth strategy information
15. Test your Provider API which is attached to a particular Retailer	Provider	Call Transact service endpoint i.e., Retailer engine endpoint to test that your Provider API is able to serialize and de-serialize request / response to Transact Capability Request / Response

Using the Transact SDK to Build the Connector

Download the current Transact SDK from the Transact Team, as this procedure requires full and ready access to the [Transact SDK](#). Follow and implement this alternate procedure in the described order:

Step	Owner	Description
1. Download the latest version of the Transact SDK to your project	Provider	Obtain the latest Transact SDK from Transact Team
2. Add dependencies in your project	Provider	Add Transact SDK to your project directory and add Transact SDK and other dependencies required in your project
3. Create partner capability API and implement connector logic inside it	Provider	Add the connector logic to convert Transact Models, which adhere to the Open API standards as defined by Transact, into models that are specific to each partner and vice-versa i.e., from partner models to Transact Open API models.
4. Test your project application locally	Provider	Run your application and test the partner API endpoints to verify that the integration with Transact SDK capability models and connector logic are functioning as expected
5. Deploy your partner capability API	Provider	Deploy your partner capability API to any AWS infrastructure of your choice
6. Create an IAM Role	Provider	Create IAM Role for partner API in order to <i>allowlist</i> Transact Service 1. <u>Policy</u> - Attach full access policy of the infrastructure to which your partner API for capability is deployed 2. <u>Trust Relations</u> - Add the following accounts and services URL for Transact so that Transact can assume this Role to invoke your partner API for capability.
7. Provide the partner API information to Transact Team	Provider	Provide partner Name , Base Endpoint , Auth Strategy Type , RoleArn , SignRegion, SignService, CustomAttributes informations to Transact Team
8. Add the configuration for the Partner API	Transact Team	Add the configuration provided by Partner to a particular Retailer engine id so that particular Retailer will use Partner capability partner API to fetch capability information
9. Deploy the Retailer and Partner configuration	Transact Team	Deploy this updated Retailer configuration to particular Transact Services i.e., the AWSTransactGatewayService and the AWSTransactCapabilityService
10. Provide Retailer information in order to test new Partner API	Transact Team	Provide partner Retailer engine id endpoint and auth strategy information
11. Test your Partner API which is attached to a particular Retailer	Provider	Call Transact service endpoint i.e., Retailer engine endpoint to test that your partner API is able to serialize and de-serialize request / response to Transact Capability Request / Response

Common HTTP Error Codes

This section lists the HTTP errors common to the API actions of all AWS services. For errors specific to an API action for this service, see the topic for that API action.

AccessDeniedException

You do not have sufficient access to perform this action.
HTTP Status Code: **403**

ExpiredTokenException

The security token included in the request is expired
HTTP Status Code: **403**

IncompleteSignature

The request signature does not conform to AWS standards.
HTTP Status Code: **403**

InternalFailure

The request processing has failed because of an unknown error, exception or failure. This is generically characterized as an '**Internal Server Error**'.
HTTP Status Code: **500**

MalformedHttpRequestException

Problems with the request at the HTTP level, e.g. we can't decompress the body according to the decompression algorithm specified by the content-encoding.
HTTP Status Code: **400**

NotAuthorized

You do not have permission to perform this action.
HTTP Status Code: **401**

OptInRequired

The AWS access key ID needs a subscription for the service.
HTTP Status Code: **403**

RequestAbortedException

Convenient exception that can be used when a request is aborted before a reply is sent back (e.g. client closed connection).

HTTP Status Code: **400**

RequestEntityTooLargeException

Problems with the request at the HTTP level. The request entity is too large.

HTTP Status Code: **413**

RequestExpired

The request reached the service more than 15 minutes after the date stamp on the request or more than 15 minutes after the request expiration date (such as for pre-signed URLs), or the date stamp on the request is more than 15 minutes in the future.

HTTP Status Code: **400**

RequestTimeoutException

Problems with the request at the HTTP level. Reading the Request timed out.

HTTP Status Code: **408**

ServiceUnavailable

The request has failed due to a temporary failure of the server.

HTTP Status Code: **503**

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: **400**

UnrecognizedClientException

The X.509 certificate or AWS access key ID provided does not exist in our records.

HTTP Status Code: **403**

UnknownOperationException

The action or operation requested is invalid. Verify that the action is typed correctly.

HTTP Status Code: **404**

ValidationError

The input fails to satisfy the constraints specified by an AWS service.

HTTP Status Code: **400**

Constraint Violation (CV) Structures

Constraint Violations are events that block a Transact AWS call, mostly due to missing or incorrect information. Constraint Violations are NOT errors. They do not cause transaction aborts. Constraint Violations are **blocking**, meaning that an AWS call or transaction is temporarily halted until the Constraint Violation is resolved (removed). When a Constraint Violation occurs, Transact does not alter or remove data, but it does stop the operation and signals the Client App. The Retailer's Client App must resolve the Constraint Violation for the operation to continue. Note that a given operation can potentially have multiple Constraint Violations. It is up to the Retailer to decide how best to handle and remove all Constraint Violations.

This section lists the Constraint Violations common to the API actions of all AWS services. All Transact AWS webservice calls potentially can have Constraint Violations, with the exception of *GetCart* and *GetOrder*. This section provides a summary reference of the types of Constraint Violations (CVs) that can occur, long with their identifiers.

Categories / Subcategories of Transact CVs

@documentation("Denotes the category of the CV")

```
enum CvCategory {
    CART
    ORDER
    CATALOG
    PRICING
    TAX
    FULFILLMENT
    INVENTORY
    DELIVERY_ESTIMATE
    RETURNS
    PROMOTIONS
    LOYALTY
}
```

@documentation("Denotes the subCategory of the CV")

```
enum CvSubCategory {
    @documentation("Requested inventory size is unavailable")
    INSUFFICIENT_INVENTORY
    @documentation("Item can not be shipped to the recipient address")
    SHIPMENT_UNAVAILABLE
    @documentation("Requested quantity exceeds permissible quantity")
    QUANTITY_LIMIT_EXCEEDED
    @documentation("Mismatch in order total")
    ORDER_TOTAL_MISMATCH
}
```

@documentation("Denotes the unitType corresponding to the CV")

```
enum CvUnitType {
    @documentation("Product ID")
    PRODUCT_ID = "productId"
    @documentation("Line item ID")
    LINE_ITEM_ID = "lineItemId"
    @documentation("Fulfillment group ID")
    FULFILLMENT_GROUP_ID = "fulfillmentGroupId"
}
```

@documentation("Supported unit types")

```
enum UnitType {
    COUNTABLE,
    MEASURABLE
}
```

@documentation("Denotes the type of jurisdiction imposing a given tax")

```
enum JurisdictionType {
    CITY,
    STATE,
    COUNTRY,
    COUNTY,
    SPECIAL,
    OTHER
}
```

CV Detection Mechanisms

The following code excerpts are internal services in Transact to handle and report Constraint Violations. This information is presented here (in AWS **smithy** modeling language) to provide context to Retailers when implementing their handlers for Constraint Violations.

```
@documentation(
    "Evaluates products"
)
```

```
@http(code: 200, method: "POST", uri: "/RudolphPlugins/EvaluateItemData")
```

```
operation EvaluateItemData {
    input: PluginRequest
    output: PluginResponse
    errors: [PluginException]
}
```

```
@documentation(
    "Evaluates Taxes using the GetTaxesForLineItems API of the Tax capability"
)
```

```
@http(code: 200, method: "POST", uri: "/RudolphPlugins/EvaluateTax")
operation EvaluateTax {
  input: PluginRequest
  output: PluginResponse
  errors: [PluginException]
}
```

```
@documentation(
  "Sign order for fulfillment"
)
```

```
@http(code: 200, method: "POST", uri: "/RudolphPlugins/SignFulfillment")
operation SignFulfillment {
  input: PluginRequest
  output: PluginResponse
  errors: [PluginException]
}
```

```
@documentation(
  "Evaluates price"
)
```

```
@http(code: 200, method: "POST", uri: "/RudolphPlugins/EvaluatePrice")
operation EvaluatePrice {
  input: PluginRequest
  output: PluginResponse
  errors: [PluginException]
}
```

```
@documentation(
  "Sign items reservation"
)
```

```
@http(code: 200, method: "POST", uri: "/RudolphPlugins/SignItemReservations")
operation SignItemReservations {
  input: PluginRequest
  output: PluginResponse
  errors: [PluginException]
}
```

```
@documentation(
  "Validate Currency Code consistency across a single transact order/cart request"
)
@http(code: 200, method: "POST", uri: "/RudolphPlugins/ValidateCurrencyCode")
operation ValidateCurrencyCode {
  input: PluginRequest
  output: PluginResponse
  errors: [PluginException]
}
```

```
@documentation(  
    "Validate that each item in the cart/order has a price"  
)  
@http(code: 200, method: "POST", uri: "/RudolphPlugins/ValidatePrice")  
operation ValidatePrice {  
    input: PluginRequest  
    output: PluginResponse  
    errors: [PluginException]  
}  
  
@documentation(  
    "Evaluates promotions"  
)  
@http(code: 200, method: "POST", uri: "/RudolphPlugins/EvaluatePromotions")  
operation EvaluatePromotions {  
    input: PluginRequest,  
    output: PluginResponse,  
    errors: [PluginException]  
}
```

Language-Specific AWS SDKs

For more information about using these APIs in one of the language-specific AWS SDKs, refer to the following links:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)