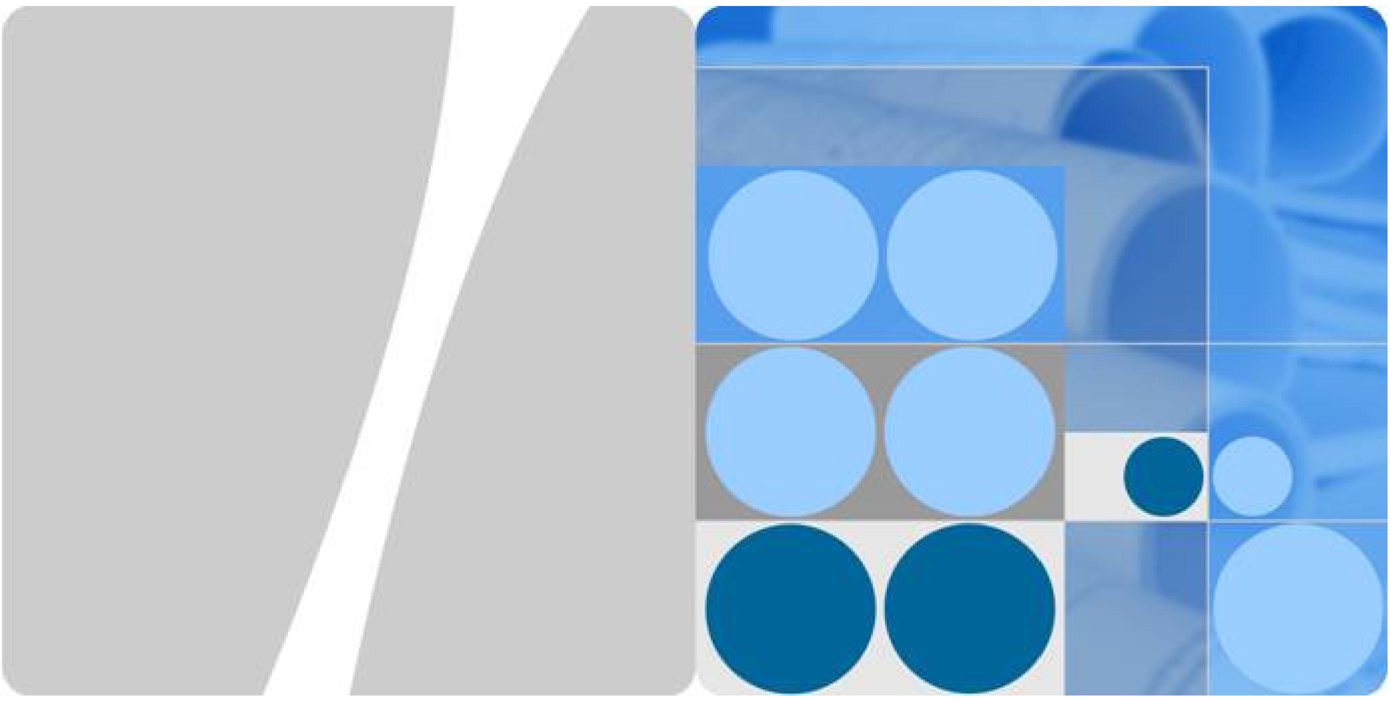


API Gateway

Developer Guide

Date 2023-12-19

Contents

1 Application Scenarios

APIG involves the following development scenarios:

- API calling authentication: For APIs that use app or IAM authentication, API calling authentication should be developed for the service system to add authentication information to API requests.
  - **API calling through app authentication (signature authentication)**: API callers provide a key, secret, and SDK through APIG to complete signature authentication for their API requests.
  - **API calling through IAM authentication (token authentication)**: API callers obtain a token from the cloud service platform and add the token to their API requests.
  - **API calling through IAM authentication (AK/SK authentication)**: API callers sign their API requests using AK (Access Key ID) /SK (Secret Access Key) and SDKs provided by APIG.
- Function for custom authentication: When using the custom authentication mode, create a function in FunctionGraph for authentication.
  - **Function for frontend custom authentication**: APIG uses the function to perform security authentication on received API requests.
  - **Function for backend custom authentication**: The API backend service uses the function to perform security authentication on API requests forwarded by APIG.
- **Signature verification by backend service**: If an API is bound with a signature key on APIG, the request sent by APIG to the backend service of the API carries the corresponding signature information. The backend service of the API must integrate the SDK provided by APIG and verify the signature information in the request.

2 Calling APIs Through App Authentication

2.1 Preparation

Before calling APIs in app authentication mode, complete the following operations:

- Obtain API request information

  On the console of a gateway, choose **API Management** > **APIs**, and click an API name to go to the details page. On the **APIs** tab, view the domain name, request path, and request method.
- Publish APIs in an environment

  On the console of a gateway, choose **API Management** > **APIs**, and click an API name to go to the details page. On the **APIs** tab, navigate to **Frontend Configuration** > **Frontend Definition**, and view the environment in which the API has been published.
- Obtain API authentication information

  To sign an API request cryptographically through app authentication (signature authentication), the key and secret of a credential authorized to call the API are required. On the console of a gateway, choose **API Management** > **Credentials**. Go to the details page of a credential, and obtain the key and secret.

  NOTE

  - **AppKey** or **Key**: access key ID of an app. It is the unique ID associated with a secret access key. The access key ID and secret access key are together used to obtain an encrypted signature for a request.
  - **AppSecret** or **Secret**: secret access key used together with an access key ID to sign requests. The access key ID and secret access key can be together used to identify a request sender to prevent the request from being modified.
- When sending an API request, add the current time to the **X-Sdk-Date** header and the signature information to the **Authorization** header.

CAUTION

The local time on the client must be synchronized with the clock server to avoid a large error in the value of the **X-Sdk-Date** request header.

APIG checks the time format and compares the time with the time when APIG receives the request. If the time difference exceeds 15 minutes, APIG will reject the request.

2.2 App Authentication

1. Construct a standard request.

   Assemble the request content according to the rules of APIG (API Management), ensuring that the client signature is consistent with that in the backend request.
2. Create a to-be-signed string using the standard request and other related information.
3. Calculate a signature using the AK/SK and to-be-signed string.
4. Add the generated signature to an HTTP request as a header or query parameter.
5. After receiving the request, APIG performs **1** to **3** to calculate a signature.
6. The new signature is compared with the signature generated in **3**. If they are consistent, the request is processed; otherwise, the request is rejected.

NOTE

The body of a signing request in app authentication mode cannot exceed 12 MB.

Step 1: Constructing a Standard Request

To access an API through app authentication, standardize the request content, and then sign the request. The client must follow the same request specifications as APIG so that each HTTP request can obtain the same signing result from the frontend and backend to complete identity authentication.

The pseudocode of standard HTTP requests is as follows:

```
CanonicalRequest =
    HTTPRequestMethod + '\n' +
    CanonicalURI + '\n' +
    CanonicalQueryString + '\n' +
    CanonicalHeaders + '\n' +
    SignedHeaders + '\n' +
    HexEncode(Hash(RequestPayload))
```

The following example shows how to construct a standard request.

Original request:

```
GET https://c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com/app1?b=2&a=1 HTTP/1.1
Host: c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com
X-Sdk-Date: 20191111T093443Z
```

1. Specify an HTTP request method (**HTTPRequestMethod**) and end with a carriage return line feed (CRLF).

For example:

```
GET
```

2. Add a standard URI (**CanonicalURI**) and end with a CRLF.

   Description

   Path of the requested resource, which is the URI code of the absolute path.

   Format

   According to RFC 3986, each part of a standard URI except the redundant and relative paths must be URI-encoded. If a URI does not end with a slash (/), add a slash at its end.

   Example

   For the URI **/app1**, the standard URI code is as follows:

   ```
   GET
   /app1/
   ```

   NOTE

   During signature calculation, the URI must end with a slash (/). When a request is sent, the URI does not need to end with a slash (/).

3. Add a standard query string (**CanonicalQueryString**) and end with a CRLF.

   Description

   Query parameters. If no query parameters are configured, the query string is an empty string.

   Format

   Standard query strings must meet the following requirements:

   – Perform URI encoding on each parameter and value according to the following rules:

     ▪ Do not perform URI encoding on any non-reserved characters defined in RFC 3986, including A–Z, a–z, 0–9, hyphen (-), underscore (_), period (.), and tilde (~).

     ▪ Use **%XY** to perform percent encoding on all non-reserved characters. **X** and **Y** indicate hexadecimal characters (0–9 and A–F). For example, the space character must be encoded as **%20**, and an extended UTF-8 character must be encoded in the "%XY%ZA%BC" format.

   – Add "*URI-encoded parameter name* = *URI-encoded parameter value*" to each parameter. If no value is specified, use a null string instead. The equal sign (=) is required.

     For example, in the following string that contains two parameters, the value of parameter **parm2** is null.

     ```
     parm1=value1&parm2=
     ```

   – Sort the parameters in alphabetically ascending order. For example, a parameter starting with uppercase letter **F** precedes another parameter starting with lowercase letter **b**.

   – Construct standard query strings from the first parameter after sorting.

   Example

   The following example contains two optional parameters **a** and **b**.

```
GET
/app1/
a=1&b=2
```

4. Add standard headers (**CanonicalHeaders**) and end with a CRLF.

   Description

   List of standard request headers, including all HTTP message headers in the to-be-signed request. The **X-Sdk-Date** header must be included to verify the signing time, which is in the UTC time format *YYYYMMDDTHHMMSSZ* as specified in ISO 8601. When publishing an API in a non-RELEASE environment, you need to specify an environment name.

   NOTICE

   The local time on the client must be synchronized with the clock server to avoid a large error in the value of the **X-Sdk-Date** request header.

   APIG checks the time format and compares the time with the time when APIG receives the request. If the time difference exceeds 15 minutes, APIG will reject the request.

   Format

   CanonicalHeaders consists of multiple message headers, for example, **CanonicalHeadersEntry0 + CanonicalHeadersEntry1 + ...**. Each message header (**CanonicalHeadersEntry**) is in the format of **Lowercase(HeaderName) + ':' + Trimall(HeaderValue) + '\n'**.

   NOTE

   - **Lowercase**: a function for converting all letters into lowercase letters.
   - **Trimall**: a function for deleting the spaces before and after a value.
   - The last message header carries a CRLF. Therefore, an empty line appears because the **CanonicalHeaders** field also contains a CRLF according to the specifications.
   - The message header name must be unique. Otherwise, authentication fails.

   Example

```
GET
/app1/
a=1&b=2
host:c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com
x-sdk-date:20191111T093443Z
```

NOTICE

Standard message headers must meet the following requirements:

- All letters in a header are converted to lowercase letters, and all spaces before and after the header are deleted.
- All headers are sorted in alphabetically ascending order.

For example, the original headers are as follows:

```
Host: c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com\n
Content-Type: application/json;charset=utf8\n
My-header1:    a   b   c  \n
X-Sdk-Date:20191111T093443Z\n
My-Header2:    "a   b   c"  \n
```

A standard header is as follows:

```
content-type:application/json;charset=utf8\n
host:c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com\n
my-header1:a   b   c\n
my-header2:"a   b   c"\n
x-sdk-date:20191111T093443Z\n
```

5. Add message headers (**SignedHeaders**) for request signing, and end with a CRLF.

   Description

   List of message headers used for request signing. This step is to determine which headers are used for signing the request and which headers can be ignored during request verification. The **X-Sdk-date** header must be included.

   Format

   SignedHeaders = Lowercase(HeaderName0) + ';' + Lowercase(HeaderName1) + ";" + ...

   Letters in the message headers are converted to lowercase letters. All headers are sorted alphabetically and separated with commas.

   Lowercase is a function for converting all letters into lowercase letters.

   Example

   In the following example, two message headers **host** and **x-sdk-date** are used for signing the request.

```
GET
/app1/
a=1&b=2
host:c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com
x-sdk-date:20191111T093443Z

host;x-sdk-date
```

6. Use the hash function SHA-256 to create a hash value based on the body (**RequestPayload**) of the HTTP or HTTPS request.

   Description

   Request message body. The message body needs two layers of conversion (**HexEncode(Hash(*RequestPayload*))**). **Hash** is a function for generating message digest. Currently, SHA-256 is supported. **HexEncode**: the Base16 encoding function for returning a digest consisting of lowercase letters. For example, **HexEncode ("m")** returns **6d** instead of **6D**. Each byte you enter is expressed as two hexadecimal characters.

NOTE

If **RequestPayload** is null, the null value is used for calculating a hash value.

Example

This example uses GET as an example and leaves the request body empty. After hash processing, the request body (empty string) is as follows:

```
GET
/app1/
a=1&b=2
host:c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com
x-sdk-date:20191111T093443Z

host;x-sdk-date
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

7. Perform hash processing on the standard request in the same way as that on the **RequestPayload**. After hash processing, the standard request is expressed with lowercase hexadecimal strings.

   Algorithm pseudocode: **Lowercase(HexEncode(Hash.SHA256(CanonicalRequest)))**

   Example of standard request after hash processing:

```
af71c5a7ef45310b8dc05ab15f7da50189ffa81a95cc284379ebaa5eb61155c0
```

Step 2: Creating a To-Be-Signed String

After a standard HTTP request is constructed and the request hash value is obtained, create a to-be-signed string by combining them with the signature algorithm and signing time.

```
StringToSign =
   Algorithm + \n +
   RequestDateTime + \n +
   HashedCanonicalRequest
```

Parameters in the pseudocode are described as follows:

- **Algorithm**

  Signature algorithm. For SHA256, the value is **SDK-HMAC-SHA256**.
- **RequestDateTime**

  Request timestamp, which is the same as **X-Sdk-Date** in the request header. The format is *YYYYMMDDTHHMMSSZ*.
- **HashedCanonicalRequest**

  Standard request generated after hash processing.

In this example, the following to-be-signed string is obtained:

```
SDK-HMAC-SHA256
20191111T093443Z
af71c5a7ef45310b8dc05ab15f7da50189ffa81a95cc284379ebaa5eb61155c0
```

Step 3: Calculating the Signature

Use the AppSecret and created character string as the input of the encryption hash function, and convert the calculated binary signature into a hexadecimal expression.

The pseudocode is as follows:

```
signature = HexEncode(HMAC(APP secret, string to sign))
```

HMAC indicates hash calculation, and **HexEncode** indicates hexadecimal conversion. **Table 2-1** describes the parameters in the pseudocode.

Table 2-1 Parameter description

Parameter	Description
AppSecret	Signature key.
To-be-signed string	Character string to be signed.

If the AppSecret is **FWTh5tqu2Pb9ZGt8NI09XYZti2V1LTa8useKXMD8**, the calculated signature is as follows:

```
01cc37e53d821da93bb7239c5b6e1640b184a748f8c20e61987b491e00b15822
```

Step 4: Adding the Signature to the Request Header

Add the signature to the HTTP Authorization header. The Authorization header is used for identity authentication and not included in the signed headers.

The pseudocode is as follows:

```
Authorization header creation pseudocode:
Authorization: algorithm Access=APP key, SignedHeaders=SignedHeaders, Signature=signature
```

There is no comma before the algorithm and **Access**. **SignedHeaders** and **Signature** must be separated with commas.

The signed headers are as follows:

```
Authorization: SDK-HMAC-SHA256 Access=FM9RLCN************NAXISK, SignedHeaders=host;x-sdk-date, 
Signature=01cc37e53d821da93bb7239c5b6e1640b184a748f8c20e61987b491e00b15822
```

The signed headers are added to the HTTP request for identity authentication. If the identity authentication is successful, the request is sent to the corresponding backend service for processing.

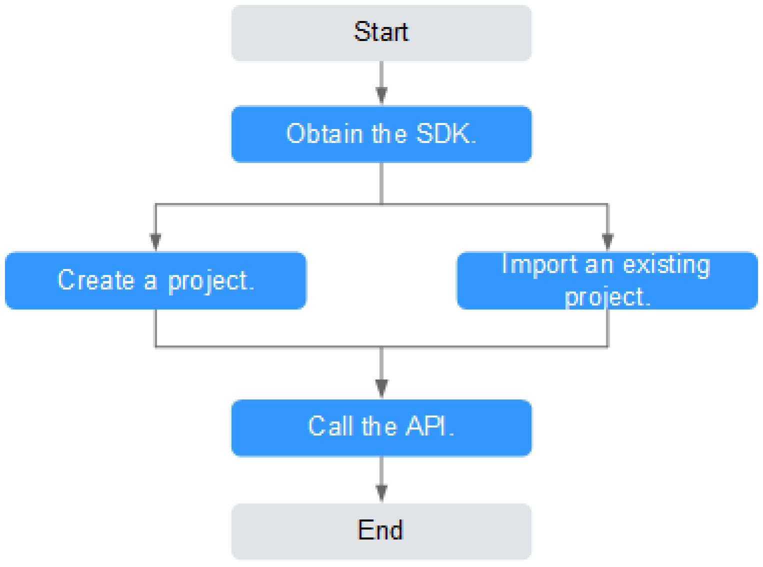
2.3 Java

Scenarios

To use Java to call an API through app authentication, obtain the Java SDK, create a project or import an existing project, and then call the API by referring to the API calling example.

This section uses Eclipse 4.5.2 as an example.

Figure 2-1 API calling process



Prerequisites

- You have obtained API calling information. For details, see **Preparation**.
- You have installed Eclipse 3.6.0 or a later version. If not, download Eclipse from the **official Eclipse website** and install it.
- You have installed Java Development Kit (JDK) 1.8.111 or a later version. If not, download JDK from the **official Oracle website** and install it. JDK 17 or later is not supported.

Obtaining the SDK

Log in to the APIG console, and download the SDK on the **SDKs** page by referring to section "SDKs" in the *API Gateway User Guide*.

Then obtain the **ApiGateway-java-sdk.zip** package. The following table shows the files decompressed from the package.

Name	Description
libs\	SDK dependencies
libs\java-sdk-core-*x.x.x*.jar	SDK package
src\com\apig\sdk\demo\Main.java	Sample code for signing requests
src\com\apig\sdk\demo\OkHttpDemo.java	
src\com\apig\sdk\demo\LargeFileUpload-Demo.java	
.classpath	Java project configuration files
.project	

Importing a Project

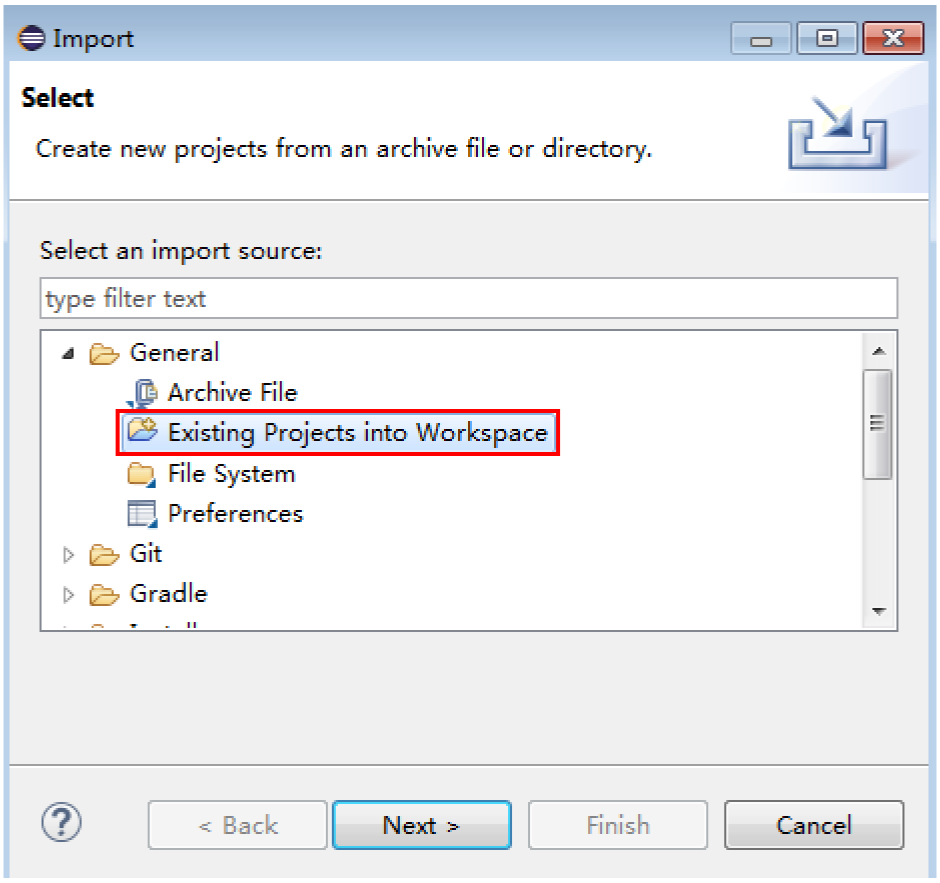
Step 1 Open Eclipse and choose **File** > **Import**.

The **Import** dialog box is displayed.

Step 2 Choose **General** > **Existing Projects into Workspace** and click **Next**.

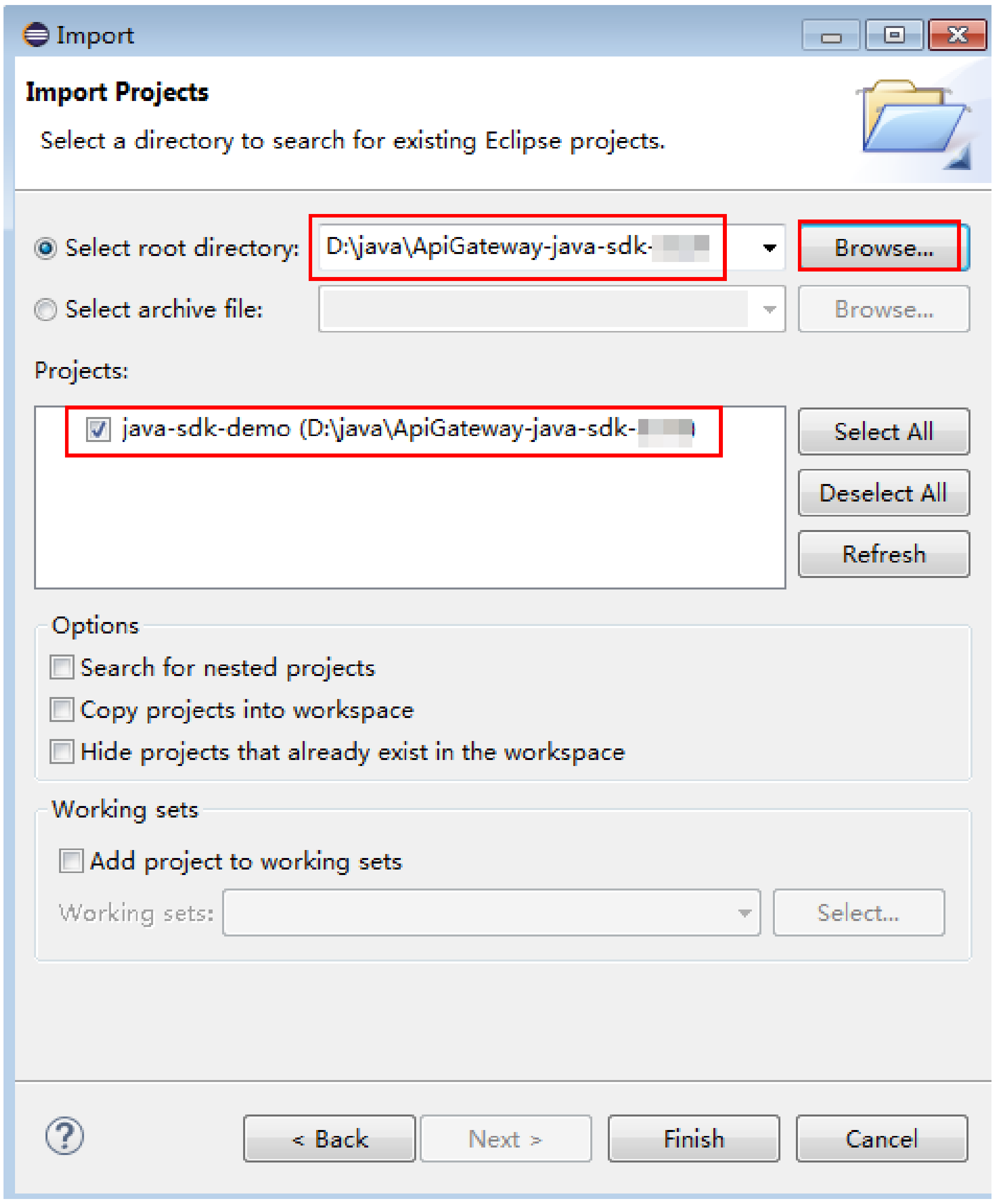
The **Import Projects** dialog box is displayed.

Figure 2-2 Importing a project



Step 3 Click **Browse** and select the directory where the SDK is decompressed.

Figure 2-3 Selecting the demo project



Import

Import Projects

Select a directory to search for existing Eclipse projects.

(●) Select root directory: D:\java\ApiGateway-java-sdk-[redacted] Browse...

( ) Select archive file: Browse...

Projects:

☑ java-sdk-demo (D:\java\ApiGateway-java-sdk-[redacted]

Options

☐ Search for nested projects

☐ Copy projects into workspace

☐ Hide projects that already exist in the workspace

Working sets

☐ Add project to working sets

Working sets:

Step 4 Click **Finish**.

Modify the parameters in sample code **Main.java** as required. For details about the sample code, see **API Calling Example**.

----End

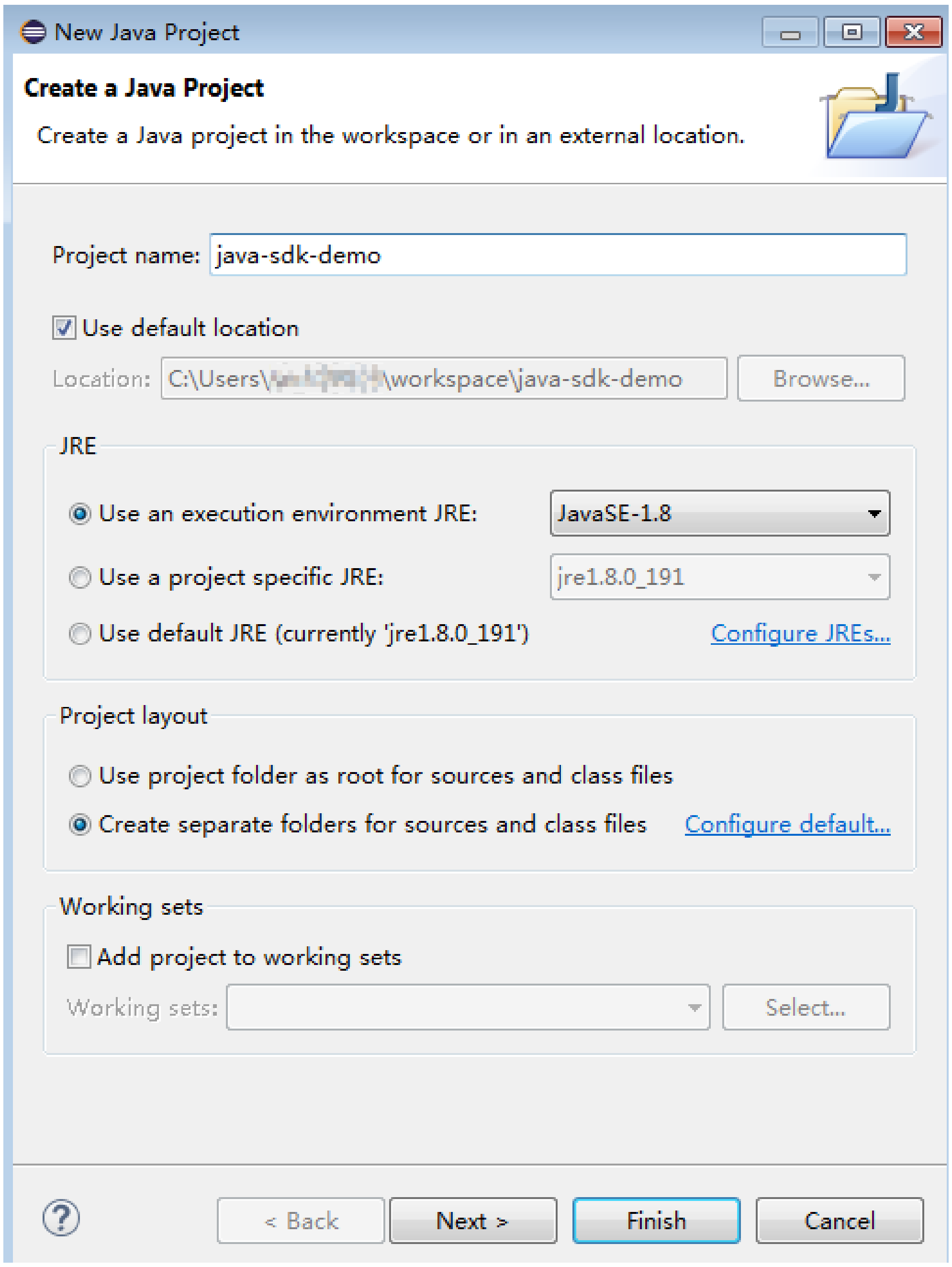
Creating a Project

Step 1 Open Eclipse and choose **File** > **New** > **Java Project**.

The **New Java Project** dialog box is displayed.

Step 2 Enter a project name, for example, **java-sdk-demo**, retain the default settings for other parameters, and click **Finish**.

Figure 2-4 Creating a project


New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: java-sdk-demo

[✓] Use default location

Location: C:\Users\[redacted]\workspace\java-sdk-demo

JRE

(●) Use an execution environment JRE: JavaSE-1.8

( ) Use a project specific JRE: jre1.8.0_191

( ) Use default JRE (currently 'jre1.8.0_191')

Project layout

( ) Use project folder as root for sources and class files

(●) Create separate folders for sources and class files

Working sets

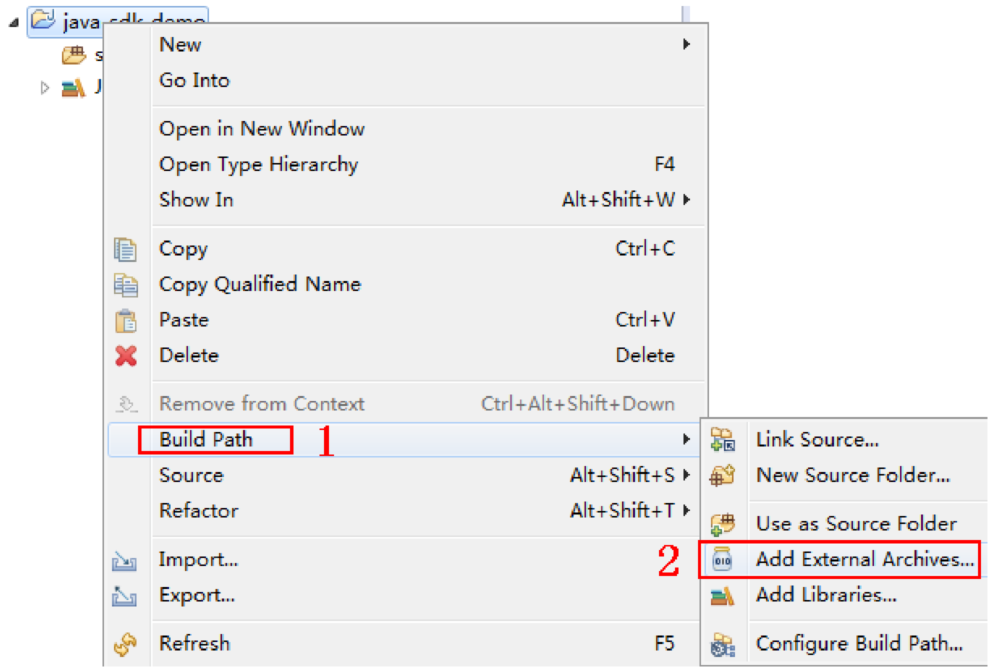
[ ] Add project to working sets

Working sets:

Step 3 Import the .jar files in the APIG Java SDK.

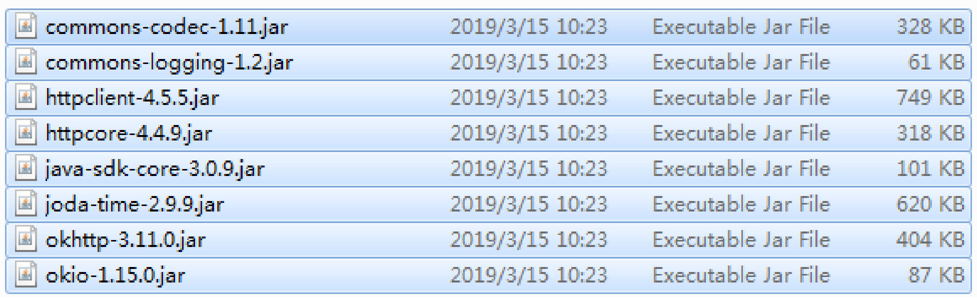
1. Choose **java-sdk-demo**, right-click, and choose **Build Path** > **Add External Archives** from the shortcut menu.

Figure 2-5 Importing the .jar files



2. Select all .jar files in the **\libs** directory.

Figure 2-6 Selecting all .jar files

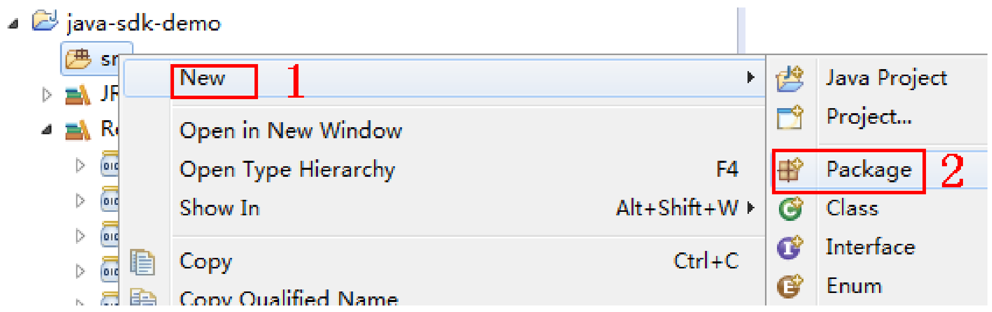


commons-codec-1.11.jar	2019/3/15 10:23	Executable Jar File	328 KB
commons-logging-1.2.jar	2019/3/15 10:23	Executable Jar File	61 KB
httpclient-4.5.5.jar	2019/3/15 10:23	Executable Jar File	749 KB
httpcore-4.4.9.jar	2019/3/15 10:23	Executable Jar File	318 KB
java-sdk-core-3.0.9.jar	2019/3/15 10:23	Executable Jar File	101 KB
joda-time-2.9.9.jar	2019/3/15 10:23	Executable Jar File	620 KB
okhttp-3.11.0.jar	2019/3/15 10:23	Executable Jar File	404 KB
okio-1.15.0.jar	2019/3/15 10:23	Executable Jar File	87 KB

Step 4 Create a package and **Main** file.

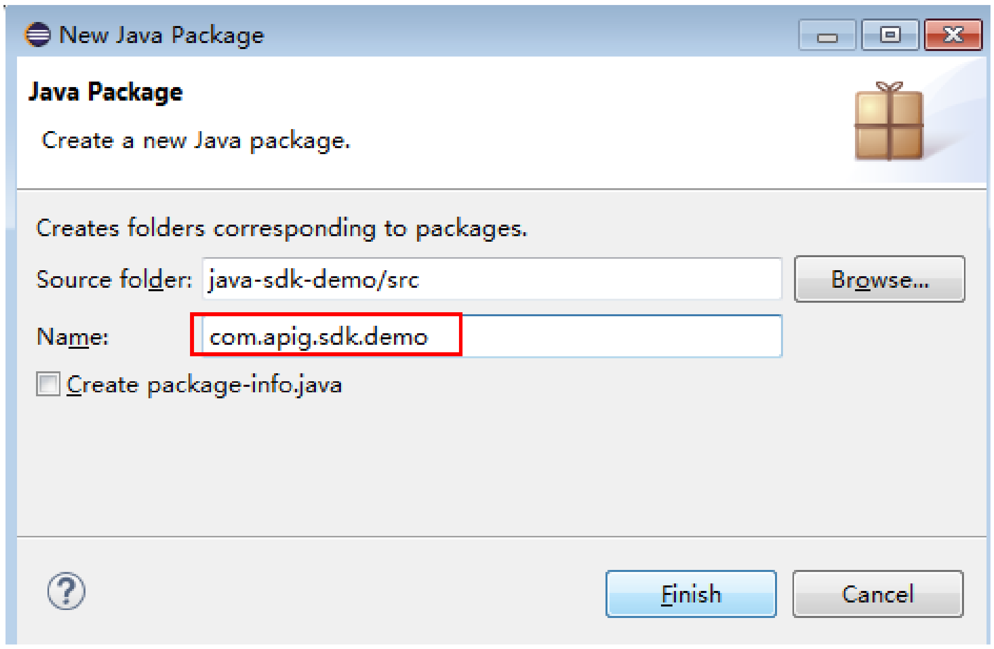
1. Choose **src**, right-click, and choose **New** > **Package** from the shortcut menu.

Figure 2-7 Creating a package



2. Enter **com.apig.sdk.demo** for **Name**.

Figure 2-8 Setting a package name



Creates folders corresponding to packages.

Source folder: java-sdk-demo/src

Name: com.apig.sdk.demo

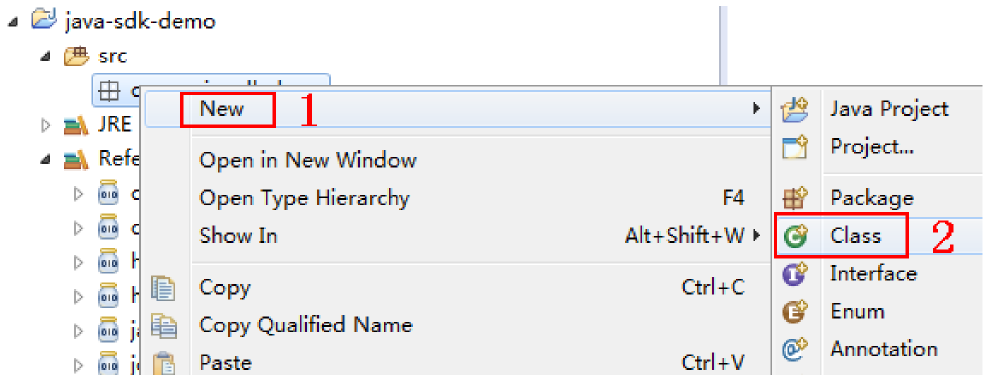
Create package-info.java

3. Click **Finish**.

   The package is created.

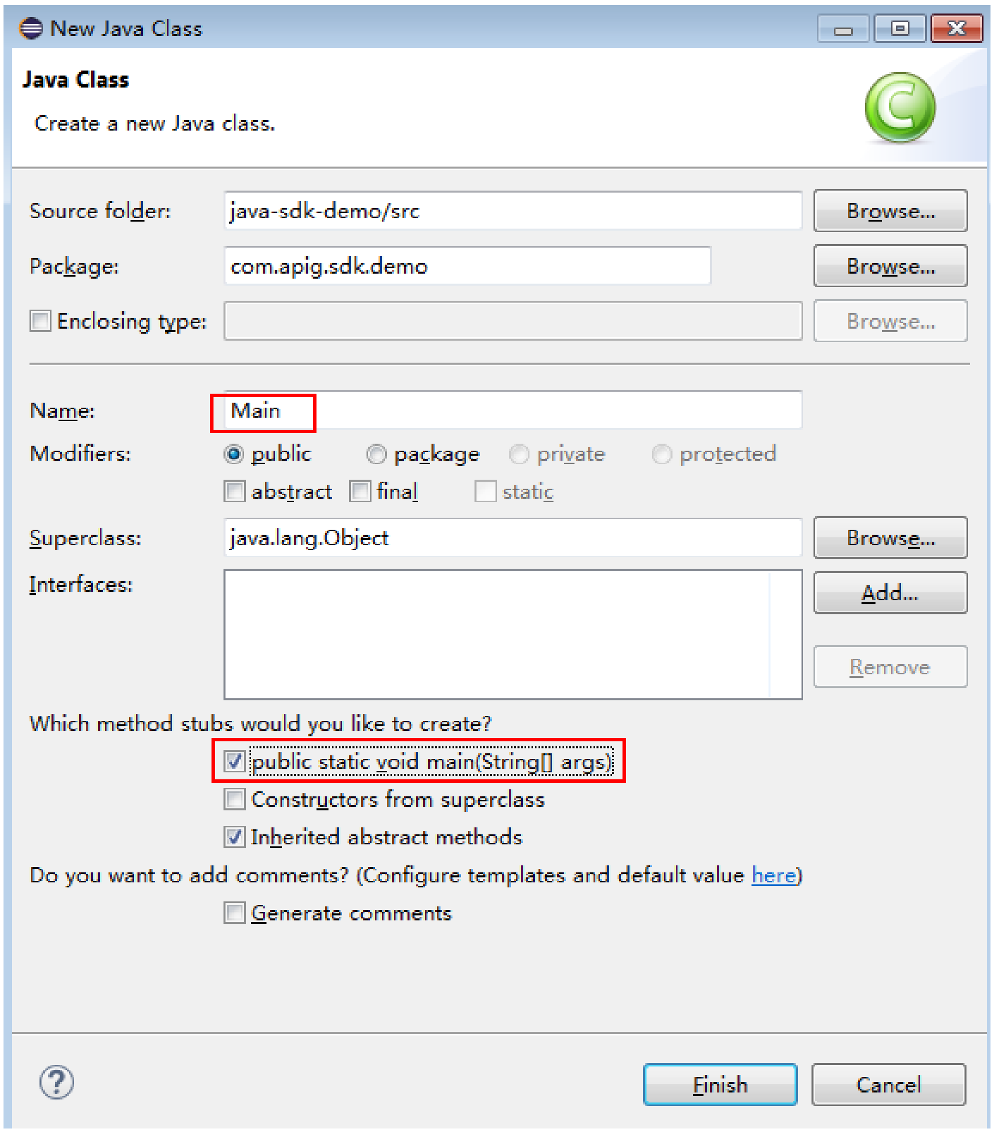
4. Choose **com.apig.sdk.demo**, right-click, and choose **New** > **Class** from the shortcut menu.

Figure 2-9 Creating a class



5. Enter **Main** for **Name** and select **public static void main(String[] args)**.

Figure 2-10 Configuring the class


New Java Class

Java Class

Create a new Java class.

Source folder: java-sdk-demo/src

Package: com.apig.sdk.demo

☐ Enclosing type:

Name: Main

Modifiers: [✓ public] package private protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☑ public static void main(String[] args)

☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

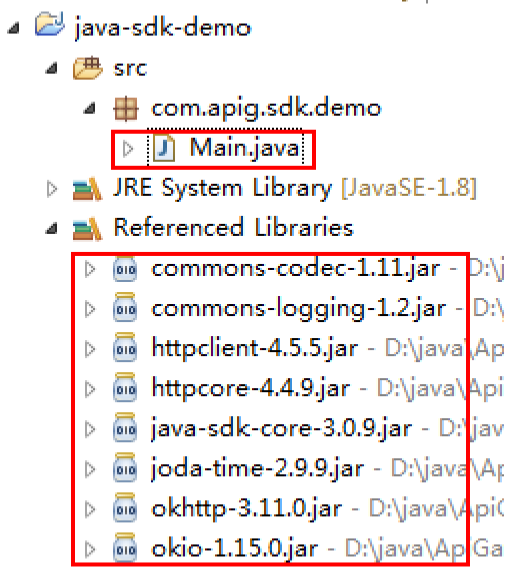
☐ Generate comments

6. Click **Finish**.

   The **Main** file is created.

Step 5 View the directory structure of the project.

Figure 2-11 Directory structure of the new project Main


java-sdk-demo
src
com.apig.sdk.demo
Main.java
JRE System Library [JavaSE-1.8]
Referenced Libraries
commons-codec-1.11.jar - D:\j
commons-logging-1.2.jar - D:\
httpclient-4.5.5.jar - D:\java\Ap
httpcore-4.4.9.jar - D:\java\Api
java-sdk-core-3.0.9.jar - D:\jav
joda-time-2.9.9.jar - D:\java\Ap
okhttp-3.11.0.jar - D:\java\ApiC
okio-1.15.0.jar - D:\java\ApiGa

Before using **Main.java**, enter the required code according to the API calling example provided in this section.

----End

API Calling Example

NOTE

- You need to create an API with the Mock backend on the console, and then publish the API. For details about how to create and publish an API, see the *API Gateway User Guide*.
- The backend of this API is a fake HTTP service, which returns response code **200** and message body **Congratulations, sdk demo is running**.

Step 1 Add the following references to **Main.java**:

```
import java.io.IOException;
import javax.net.ssl.SSLContext;

import org.apache.http.Header;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.methods.HttpRequestBase;
import org.apache.http.conn.ssl.AllowAllHostnameVerifier;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.conn.ssl.SSLContexts;
import org.apache.http.conn.ssl.TrustSelfSignedStrategy;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

import com.cloud.apigateway.sdk.utils.Client;
import com.cloud.apigateway.sdk.utils.Request;
```

Step 2 Construct a request by configuring the following parameters:

- **AppKey**: Obtain it by referring to **Preparation**. The sample code uses **4f5f****100c**.

- **AppSecret**: Obtain it by referring to **Preparation**. The sample code uses **********.
- **Method**: Specify a request method. The sample code uses **POST**.
- **url**: Request URL of the API, excluding the QueryString and fragment parts. Use your own independent domain name. The sample code uses **http://c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com/java-sdk**.
- **queryString**: Specify query parameters to be carried in the URL. Characters (0-9a-zA-Z./;[]\-=~#%^&_+:") are allowed. The sample code uses **name=value**.
- **header**: Specify request headers. The sample code uses **Content-Type:text/plain**. If you are going to publish the API in a non-RELEASE environment, specify an environment name. The sample code uses **x-stage:publish_env_name**.
- **body**: Specify the request body. The sample code uses **demo**.

The sample code is as follows:

```
        Request request = new Request();
        try
        {
            request.setKey("4f5f****100c"); //Obtained when an app is created.
            request.setSecret("*****"); //Obtained when an app is created.
            request.setMethod("POST");
            request.setUrl("http://c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com/java-
sdk");
             //URL
            request.addQueryStringParam("name", "value");
            request.addHeader("Content-Type", "text/plain");
            //request.addHeader("x-stage", "publish_env_name"); //Specify an environment name before
publishing the API in a non-RELEASE environment.
            request.setBody("demo");
        } catch (Exception e)
        {
            e.printStackTrace();
            return;
        }
```

Step 3 Sign the request, access the API, and print the result.

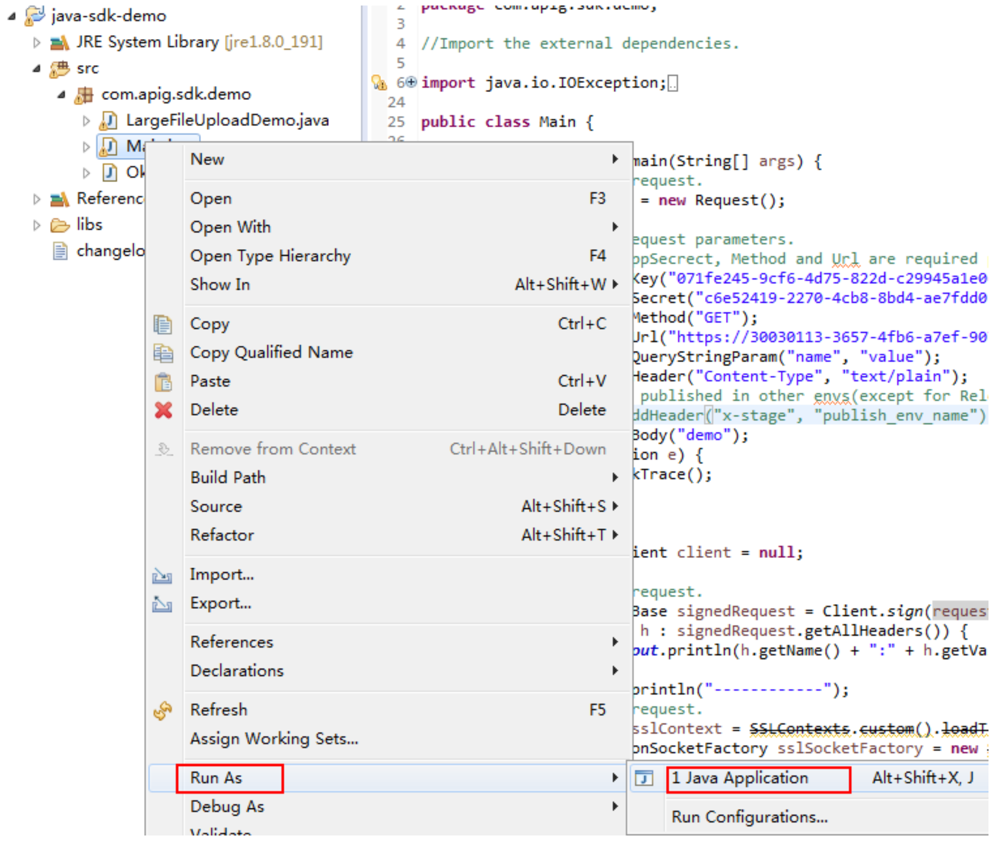
The sample code is as follows:

```
        CloseableHttpClient client = null;
        try
        {
            HttpRequestBase signedRequest = Client.sign(request);

            client = HttpClients.custom().build();
            HttpResponse response = client.execute(signedRequest);
            System.out.println(response.getStatusLine().toString());
            Header[] resHeaders = response.getAllHeaders();
            for (Header h : resHeaders)
            {
                System.out.println(h.getName() + ":" + h.getValue());
            }
            HttpEntity resEntity = response.getEntity();
            if (resEntity != null)
            {
                System.out.println(System.getProperty("line.separator") + EntityUtils.toString(resEntity, "UTF-8"));
            }

        } catch (Exception e)
        {
            e.printStackTrace();
        } finally
```

```
    {
        try
        {
            if (client != null)
            {
                client.close();
            }
        } catch (IOException e)
        {
            e.printStackTrace();
        }
    }
```

Step 4 Choose **Main.java**, right-click, and choose **Run As** > **Java Application** to run the project test code.

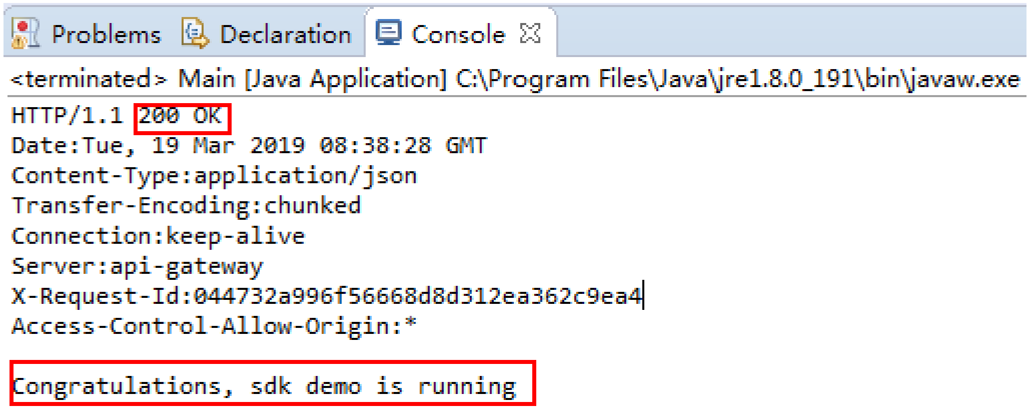
Figure 2-12 Running the project test code



```
//Import the external dependencies.

import java.io.IOException;

public class Main {

main(String[] args) {
request.
 = new Request();

equest parameters.
ppSecrect, Method and Url are required
Key("071fe245-9cf6-4d75-822d-c29945a1e0
Secret("c6e52419-2270-4cb8-8bd4-ae7fdd0
Method("GET");
Url("https://30030113-3657-4fb6-a7ef-90
QueryStringParam("name", "value");
Header("Content-Type", "text/plain");
 published in other envs(except for Rel
ddHeader("x-stage", "publish_env_name")
Body("demo");
ion e) {
kTrace();

ient client = null;

request.
Base signedRequest = Client.sign(reques
 h : signedRequest.getAllHeaders()) {
out.println(h.getName() + ":" + h.getVa

println("------------");
request.
sslContext = ~~SSLContexts~~.~~custom~~().~~loadT~~
onSocketFactory sslSocketFactory = new
```

Step 5 On the **Console** tab page, view the running result.

Figure 2-13 Response displayed if the calling is successful



```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_191\bin\javaw.exe
HTTP/1.1 200 OK
Date:Tue, 19 Mar 2019 08:38:28 GMT
Content-Type:application/json
Transfer-Encoding:chunked
Connection:keep-alive
Server:api-gateway
X-Request-Id:044732a996f56668d8d312ea362c9ea4
Access-Control-Allow-Origin:*

Congratulations, sdk demo is running
```

----End

2.4 Go

Scenarios

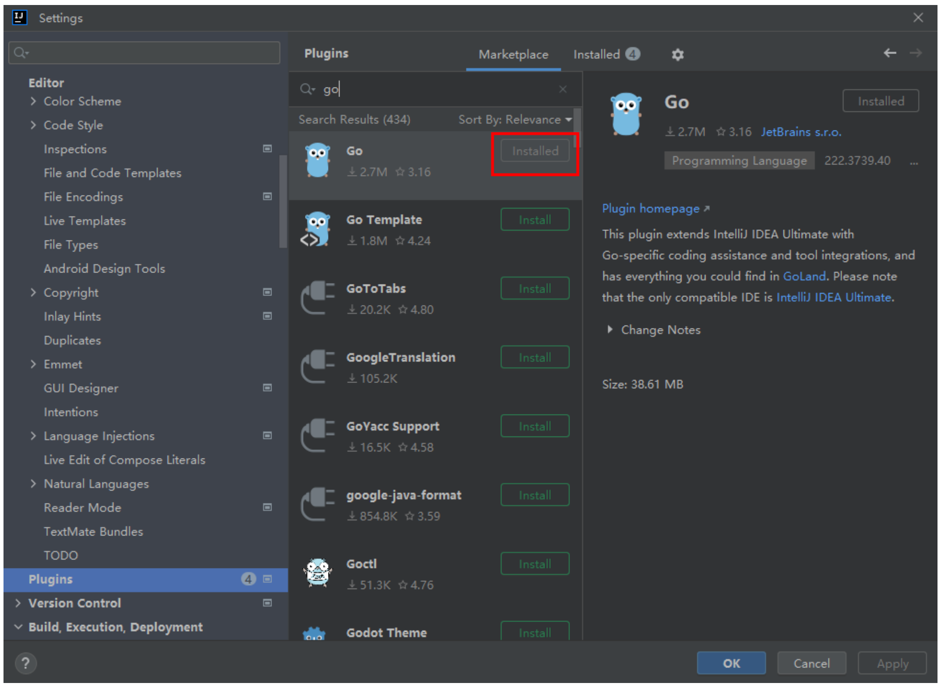
To use Go to call an API through app authentication, obtain the Go SDK, create a new project, and then call the API by referring to the API calling example.

This section uses IntelliJ IDEA 2022.2.1 as an example.

Prerequisites

- You have obtained API calling information. For details, see **Preparation**.
- You have installed Go 1.14 or a later version. If not, download the Go installation package from the **official Go website** and install it.
- You have installed IntelliJ IDEA 2022.2.1 or a later version. If not, download the installation package from the **official IntelliJ IDEA website** and install it.
- You have installed the Go plug-in on IntelliJ IDEA. If not, install the Go plug-in according to **Figure 2-14**.

Figure 2-14 Installing the Go plug-in



This plugin extends IntelliJ IDEA Ultimate with Go-specific coding assistance and tool integrations, and has everything you could find in GoLand. Please note that the only compatible IDE is IntelliJ IDEA Ultimate.

Obtaining the SDK

Log in to the APIG console, and download the SDK on the **SDKs** page by referring to section "SDKs" in the *API Gateway User Guide*.

Then obtain the **ApiGateway-go-sdk.zip** package. The following table shows the files decompressed from the package.

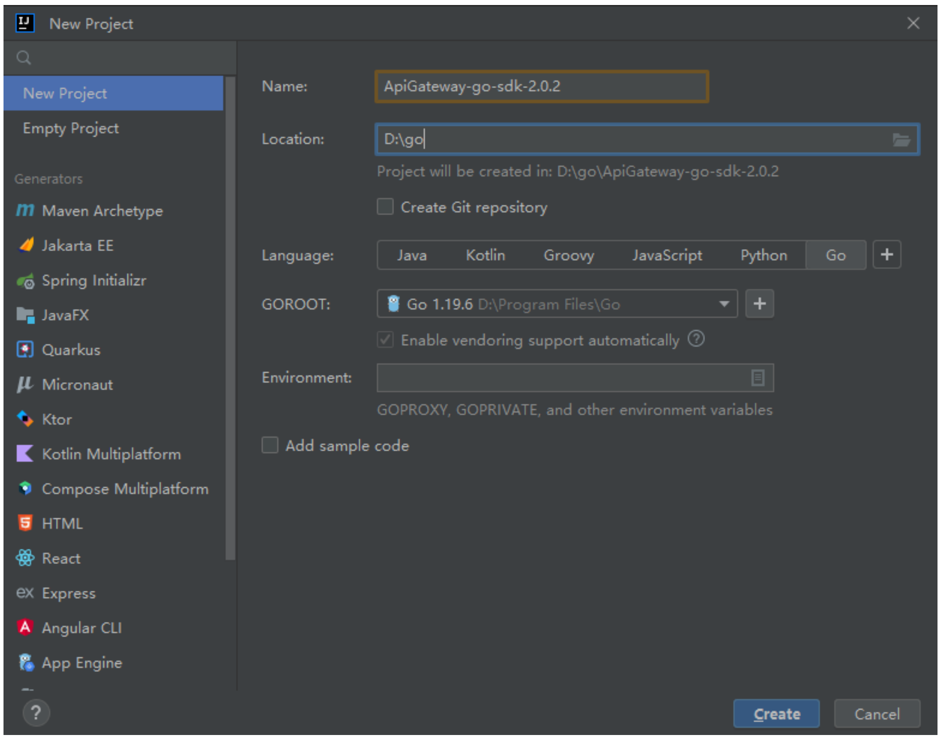
Name	Description
core\escape.go	SDK code
core\signer.go	
demo.go	Sample code

Creating a Project

Step 1 Start IntelliJ IDEA and choose **File** > **New** > **Project**.

In the displayed **New Project** dialog box, set **Name** to the name of the folder in the SDK package, **Location** to the decompression path of the folder, **Language** to **Go**, and click **Create**.

Figure 2-15 Go


New Project

Name: ApiGateway-go-sdk-2.0.2

Location: D:\go

Project will be created in: D:\go\ApiGateway-go-sdk-2.0.2

Create Git repository

Language: Java | Kotlin | Groovy | JavaScript | Python | Go

GOROOT: Go 1.19.6 D:\Program Files\Go

Enable vendoring support automatically

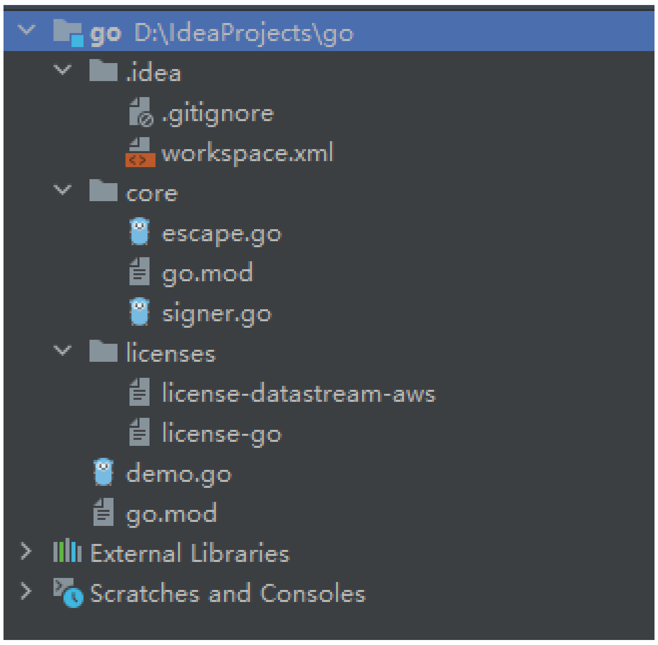
Environment:

GOPROXY, GOPRIVATE, and other environment variables

Add sample code

Step 2 View the directory structure shown in the following figure.

Figure 2-16 Directory structure of the new project go


```
go D:\IdeaProjects\go
  .idea
    .gitignore
    workspace.xml
  core
    escape.go
    go.mod
    signer.go
  licenses
    license-datastream-aws
    license-go
  demo.go
  go.mod
External Libraries
Scratches and Consoles
```

Modify the parameters in sample code **demo.go** as required. For details about the sample code, see **API Calling Example**.

----End

API Calling Example

Step 1 Import the Go SDK (signer.go) to the project.

```
import "apig-sdk/go/core"
```

Step 2 Generate a new signer and enter the AppKey and AppSecret.

```
s := core.Signer{
        Key: os.Getenv("CLOUD_SDK_AK"),
        Secret: os.Getenv("CLOUD_SDK_SK"),
}
```

Step 3 Generate a new request, and specify the domain name, method, request URL, query parameters, and body.

```
r, _ := http.NewRequest("POST", "http://c967a237-cd6c-470e-906f-
a8655461897e.apigw.exampleRegion.com/api?a=1&b=2",
                        ioutil.NopCloser(bytes.NewBuffer([]byte("foo=bar"))))
```

Step 4 Add the x-stage header to the request to specify an environment name. Add other headers to be signed as necessary.

```
r.Header.Add("x-stage", "RELEASE")
```

Step 5 Execute the following function to add the X-Sdk-Date and Authorization headers for signing:

```
s.Sign(r)
```

Step 6 Access the API and view the access result.

```
resp, err := http.DefaultClient.Do(r)
body, err := ioutil.ReadAll(resp.Body)
```

----End

2.5 Python

Scenarios

To use Python to call an API through app authentication, obtain the Python SDK, create a new project, and then call the API by referring to the API calling example.

This section uses IntelliJ IDEA 2018.3.5 as an example.

Prerequisites

- You have obtained API calling information. For details, see **Preparation**.
- You have installed Python 2.7.9 or 3.X. If not, download the Python installation package from the **official Python website** and install it.

  After Python is installed, run the **pip** command to install the **requests** library in the CLI or Shell window.

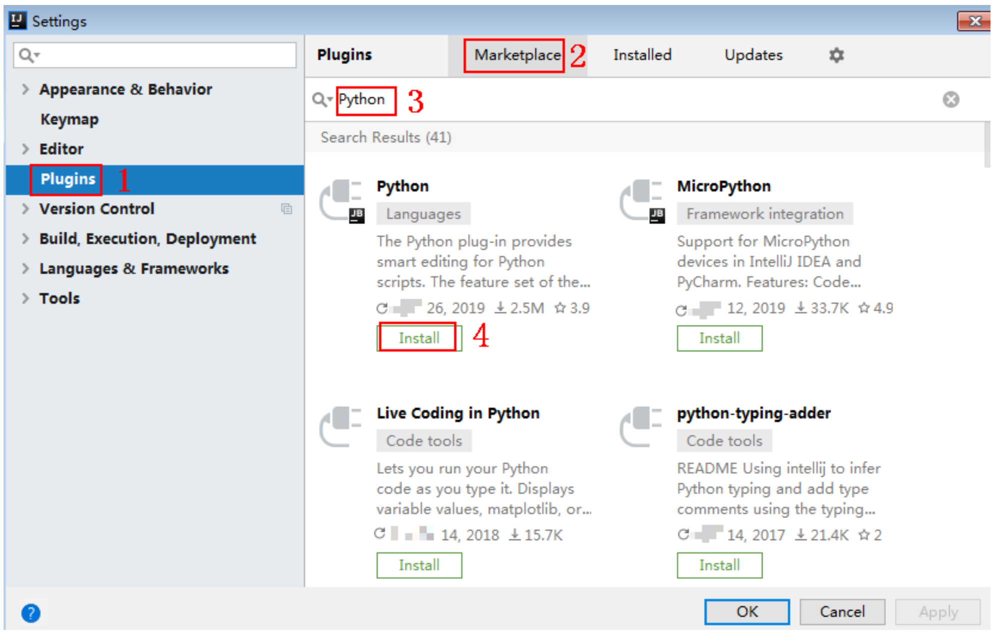
  ```
  pip install requests
  ```

  NOTE

  If a certificate error occurs during the installation, download the **get-pip.py** file to upgrade the pip environment, and try again.
- You have installed IntelliJ IDEA 2018.3.5 or a later version. If not, download the installation package from the **official IntelliJ IDEA website** and install it.

- You have installed the Python plug-in on IntelliJ IDEA. If not, install the Python plug-in according to **Figure 2-17**.

Figure 2-17 Installing the Python plug-in



The Python plug-in provides smart editing for Python scripts. The feature set of the...

Support for MicroPython devices in IntelliJ IDEA and PyCharm. Features: Code...

Lets you run your Python code as you type it. Displays variable values, matplotlib, or...

README Using intellij to infer Python typing and add type comments using the typing...

Obtaining the SDK

Log in to the APIG console, and download the SDK on the **SDKs** page by referring to section "SDKs" in the *API Gateway User Guide*.

Then obtain the **ApiGateway-python-sdk.zip** package. The following table shows the files decompressed from the package.

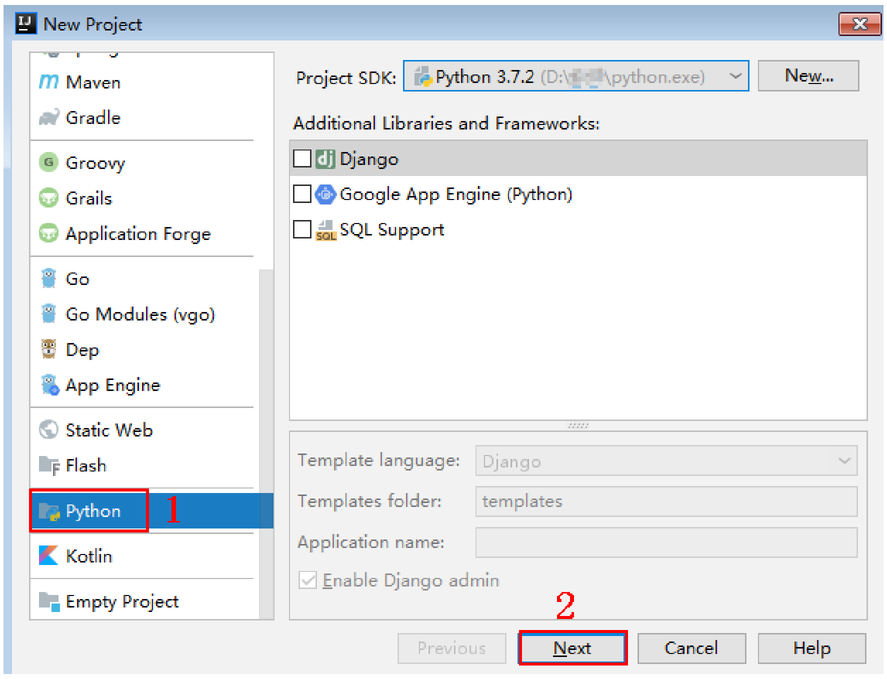
Name	Description
apig_sdk\__init__.py	SDK code
apig_sdk\signer.py	
main.py	Sample code
backend_signature.py	Sample code for backend signing
licenses\license-requests	Third-party license

Creating a Project

Step 1 Start IDEA and choose **File** > **New** > **Project**.

On the displayed **New Project** page, choose **Python** and click **Next**.

Figure 2-18 Python



Project SDK: Python 3.7.2 (D:\ \python.exe)

Additional Libraries and Frameworks:

Template language: Django

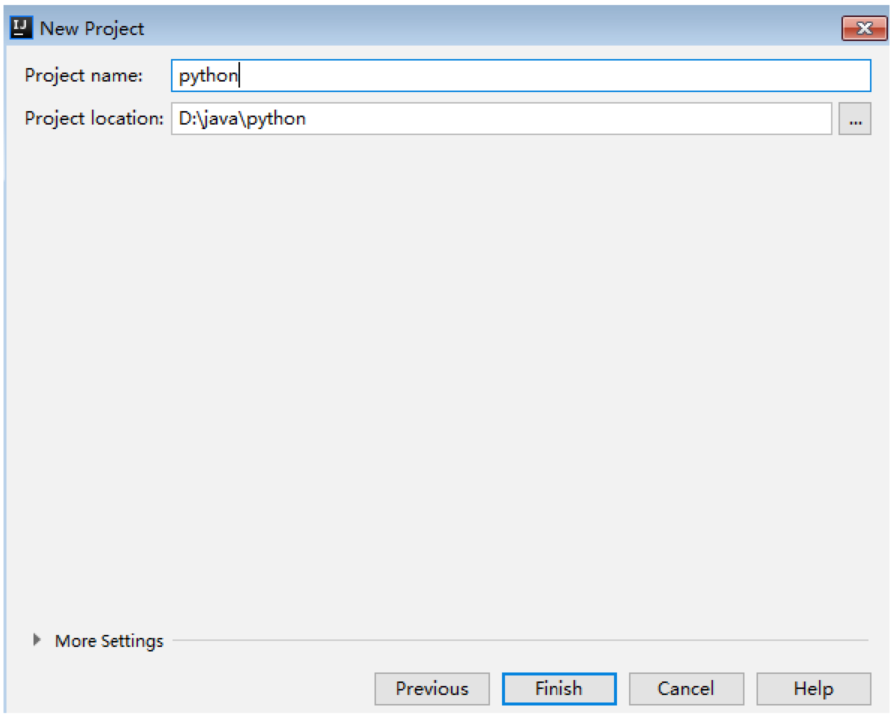
Templates folder: templates

Application name:

Enable Django admin

Step 2 Click **Next**. Click **...**, select the directory where the SDK is decompressed, and click **Finish**.

Figure 2-19 Selecting the SDK directory after decompression

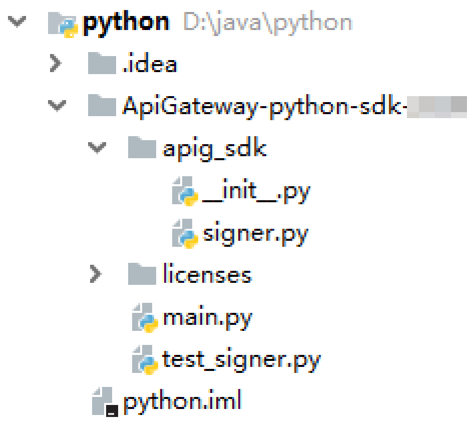


Project name: python

Project location: D:\java\python

Step 3 View the directory structure shown in the following figure.

Figure 2-20 Directory structure of the new project python



```
python D:\java\python
  .idea
  ApiGateway-python-sdk-[redacted]
    apig_sdk
      __init__.py
      signer.py
    licenses
    main.py
    test_signer.py
  python.iml
```

Modify the parameters in sample code **main.py** as required. For details about the sample code, see **API Calling Example**.

----End

API Calling Example

Step 1 Import **apig_sdk** to the project.

```
from apig_sdk import signer
import requests
```

Step 2 Generate a new signer and enter the AppKey and AppSecret.

```
sig = signer.Signer()
sig.Key = "4f5f***100c"
sig.Secret = "******"
```

Step 3 Generate a request, and specify the method, request URI, header, and request body.

```
r = signer.HttpRequest("POST",
                    "https://c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com/app1?a=1",
                    {"x-stage": "RELEASE"},
                    "body")
```

Step 4 Execute the following function to add the X-Sdk-Date and Authorization headers for signing:

NOTE

X-Sdk-Date is a request header parameter required for signing requests.

```
sig.Sign(r)
```

Step 5 Access the API and view the access result.

```
resp = requests.request(r.method, r.scheme + "://" + r.host + r.uri, headers=r.headers, data=r.body)
print(resp.status_code, resp.reason)
print(resp.content)
```

----End

2.6 C#

Scenarios

To use C# to call an API through app authentication, obtain the C# SDK, open the project file in the SDK, and then call the API by referring to the API calling example.

Prerequisites

- You have obtained API calling information. For details, see **Preparation**.
- You have installed Visual Studio 2019 version 16.8.4 or a later version. If not, download it from the **official Visual Studio website** and install it.

Obtaining the SDK

Log in to the APIG console, and download the SDK on the **SDKs** page by referring to section "SDKs" in the *API Gateway User Guide*.

Then obtain the **ApiGateway-csharp-sdk.zip** package. The following table shows the files decompressed from the package.

Name	Description
apigateway-signature\Signer.cs	SDK code
apigateway-signature\HttpEncoder.cs	
sdk-request\Program.cs	Sample code for signing requests
backend-signature\	Sample project for backend signing
csharp.sln	Project file
licenses\license-referencesource	Third-party license

Opening a Project

Double-click **csharp.sln** in the SDK package to open the project. The project contains the following:

- **apigateway-signature**: Shared library that implements the signature algorithm. It can be used in the .Net Framework and .Net Core projects.
- **backend-signature**: Example of a backend service signature.
- **sdk-request**: Example of invoking the signature algorithm. Modify the parameters as required. For details about the sample code, see **API Calling Example**.

API Calling Example

Step 1 Import the C# SDK to your project.

```
using APIGATEWAY_SDK;
```

Step 2 Generate a new signer and enter the AppKey and AppSecret.

```
Signer signer = new Signer();
signer.Key = "4f5f****6100c";
signer.Secret = "******";
```

Step 3 Generate an HttpRequest, and specify the method, request URL, and body.

```
HttpRequest r = new HttpRequest("POST",
                    new Uri("https://c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com/app1?
query=value"));
r.body = "{\"a\":1}";
```

Step 4 Add the x-stage header to the request to specify an environment name. Add other headers to be signed as necessary.

```
r.headers.Add("x-stage", "RELEASE");
```

Step 5 Execute the following function to generate **HttpWebRequest**, and add the X-Sdk-Date and Authorization headers for signing the request:

```
HttpWebRequest req = signer.Sign(r);
```

Step 6 Access the API and view the access result.

```
var writer = new StreamWriter(req.GetRequestStream());
writer.Write(r.body);
writer.Flush();
HttpWebResponse resp = (HttpWebResponse)req.GetResponse();
var reader = newStreamReader(resp.GetResponseStream());
Console.WriteLine(reader.ReadToEnd());
```

----End

2.7 JavaScript

Scenarios

To use JavaScript to call an API through app authentication, obtain the JavaScript SDK, create a project, and then call the API by referring to the API calling example.

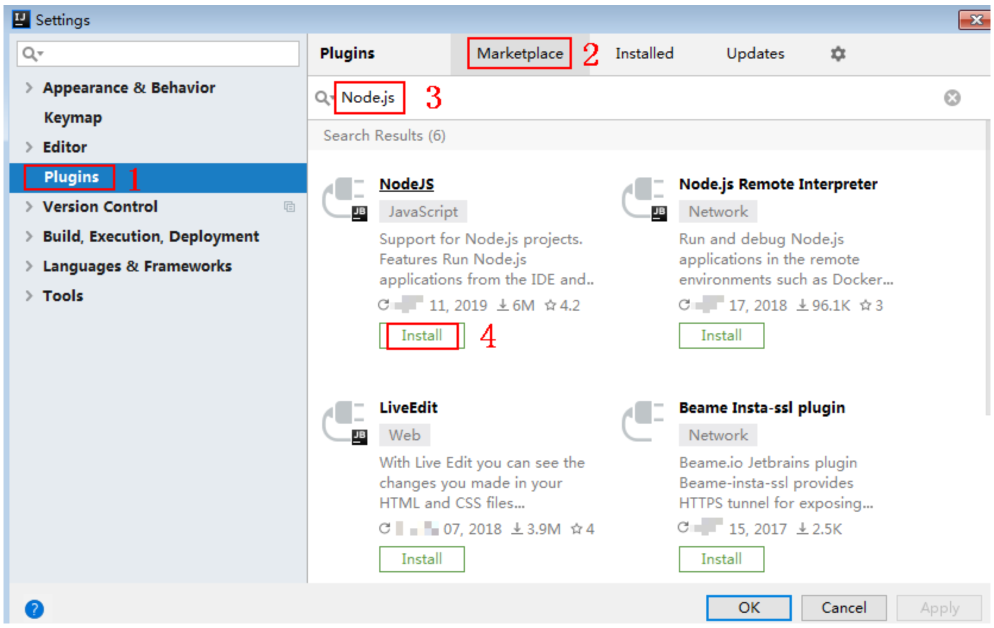
The JavaScript SDK can run in a **Node.js** or **browser** environment.

This section describes the procedure of using IntelliJ IDEA 2018.3.5 to establish a Node.js environment. It also provides examples of calling APIs using a browser.

Preparing the Environment

- You have obtained API calling information. For details, see **Preparation**.
- You have installed Node.js 15.10.0 or a later version. If not, download it from the **official Node.js website** and install it.
- You have installed IntelliJ IDEA 2018.3.5 or a later version. If not, download it from the **official IntelliJ IDEA website** and install it.
- You have installed the Node.js plug-in on IntelliJ IDEA. If not, install the Node.js plug-in according to **Figure 2-21**.

Figure 2-21 Installing the Node.js plug-in


Settings
Appearance & Behavior
Keymap
Editor
Plugins
1
Version Control
Build, Execution, Deployment
Languages & Frameworks
Tools
Plugins
Marketplace
2
Installed
Updates
Node.js
3
Search Results (6)
NodeJS
JavaScript
Support for Node.js projects. Features Run Node.js applications from the IDE and..
Install
4
Node.js Remote Interpreter
Network
Run and debug Node.js applications in the remote environments such as Docker...
Install
LiveEdit
Web
With Live Edit you can see the changes you made in your HTML and CSS files...
Install
Beame Insta-ssl plugin
Network
Beame.io Jetbrains plugin Beame-insta-ssl provides HTTPS tunnel for exposing...
Install
OK
Cancel
Apply

Obtaining the SDK

Log in to the APIG console, and download the SDK on the **SDKs** page by referring to section "SDKs" in the *API Gateway User Guide*.

Then obtain the **ApiGateway-javascript-sdk.zip** package. The following table shows the files decompressed from the package.

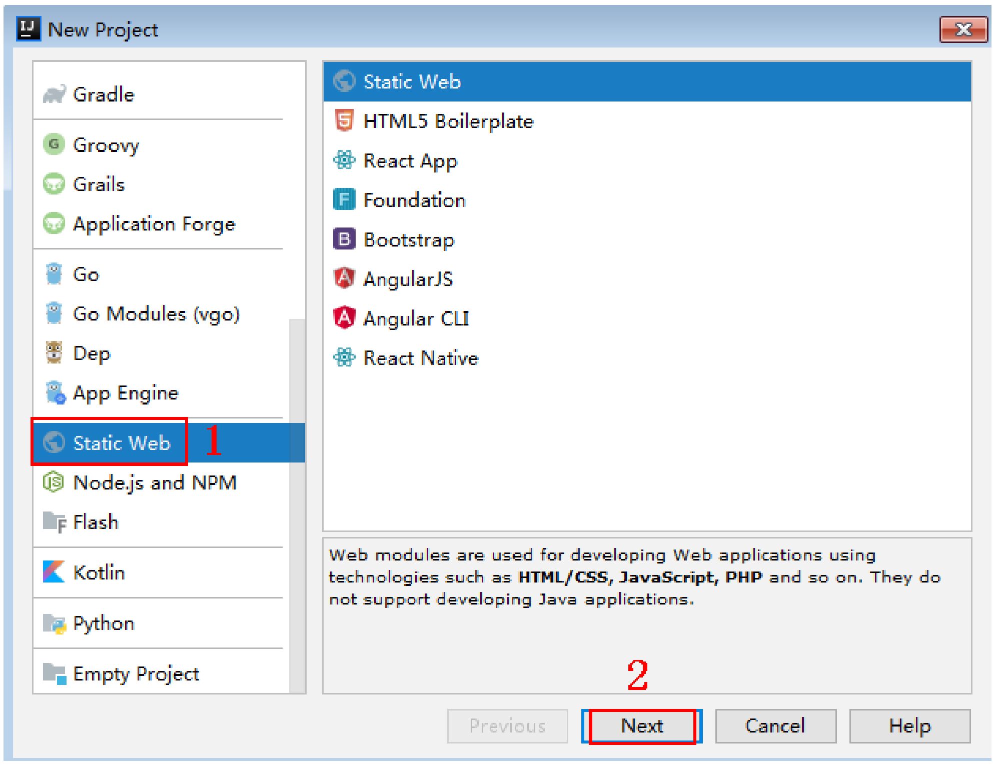
Name	Description
signer.js	SDK code
node_demo.js	Node.js sample code
demo.html	Browser sample code
demo_require.html	Browser sample code (loaded using **require**)
test.js	Test case
js\hmac-sha256.js	Dependencies
licenses\license-crypto-js	Third-party licenses
licenses\license-node	

Creating a Project

Step 1 Start IntelliJ IDEA and choose **File** > **New** > **Project**.

In the **New Project** dialog box, choose **Static Web** and click **Next**.

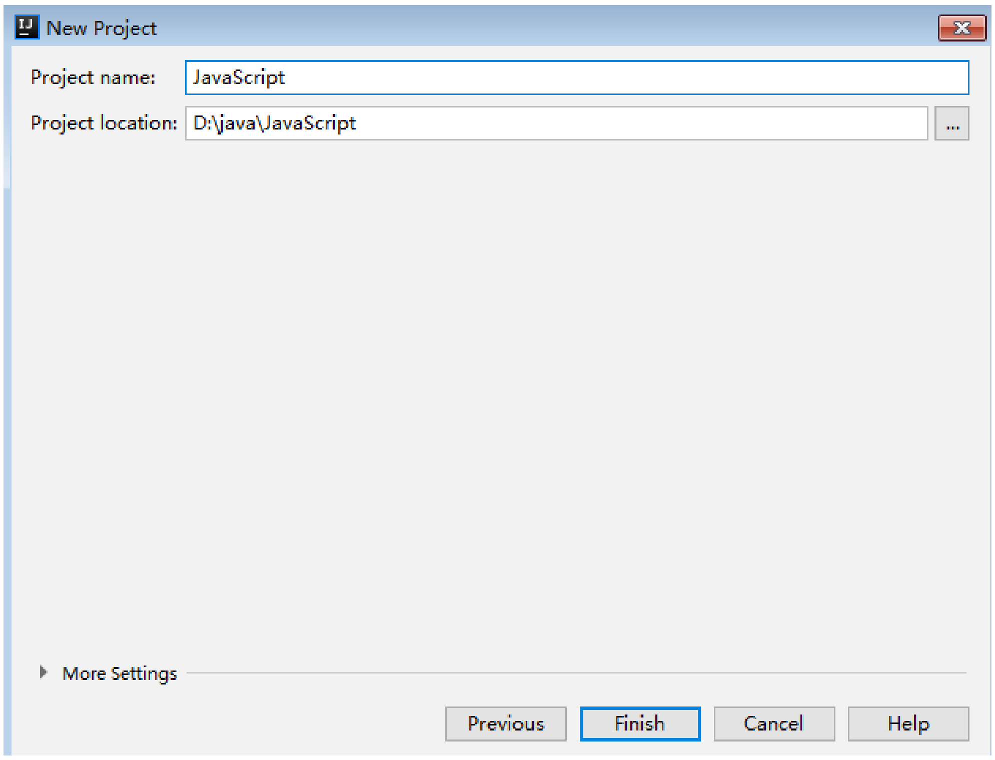
Figure 2-22 Static Web



Web modules are used for developing Web applications using technologies such as **HTML/CSS, JavaScript, PHP** and so on. They do not support developing Java applications.

Step 2 Click **...**, select the directory where the SDK is decompressed, and click **Finish**.

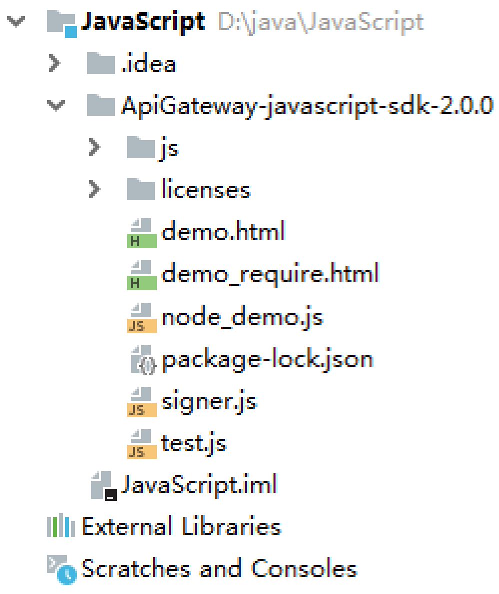
Figure 2-23 Selecting the SDK directory of JavaScript after decompression



Project name: JavaScript
Project location: D:\java\JavaScript ...

Step 3 View the directory structure shown in the following figure.

Figure 2-24 Directory structure of the new project JavaScript

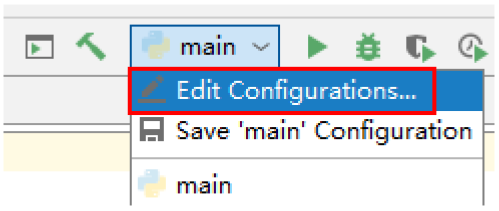


```
JavaScript D:\java\JavaScript
  .idea
  ApiGateway-javascript-sdk-2.0.0
    js
    licenses
    demo.html
    demo_require.html
    node_demo.js
    package-lock.json
    signer.js
    test.js
  JavaScript.iml
External Libraries
Scratches and Consoles
```

- **node_demo.js**: Sample code in Node.js. Modify the parameters in the sample code as required. For details about the sample code, see **API Calling Example (Node.js)**.
- **demo.html**: Browser sample code. Modify the parameters in the sample code as required. For details about the sample code, see **API Calling Example (Browser)**.

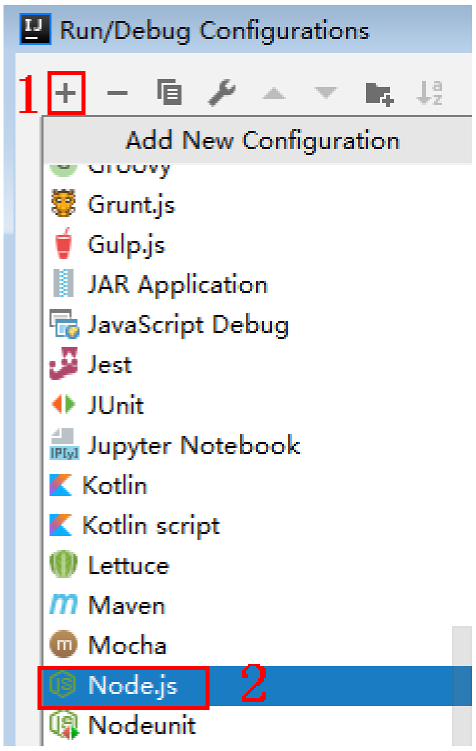
Step 4 Click **Edit Configurations**.

Figure 2-25 Click Edit Configurations



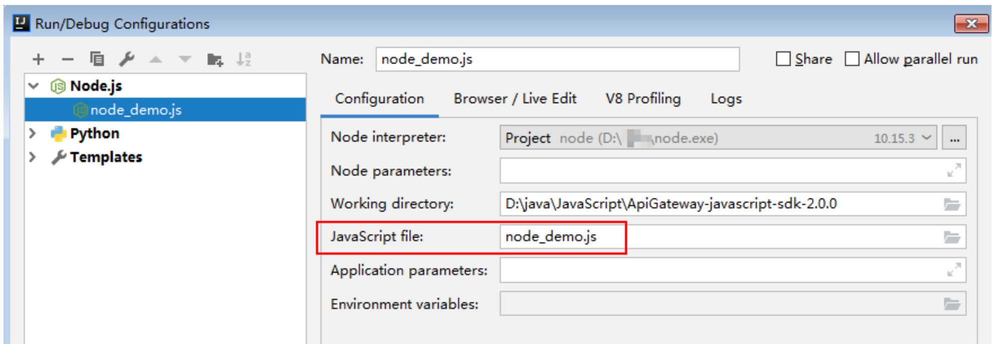
Step 5 Click **+** and select **Node.js**.

Figure 2-26 Selecting Node.js



Step 6 Set **JavaScript file** to **node_demo.js** and click **OK**.

Figure 2-27 Selecting node_demo.js



Name: node_demo.js

Node interpreter: Project node (D:\ [redacted] node.exe) 10.15.3

Node parameters:

Working directory: D:\java\JavaScript\ApiGateway-javascript-sdk-2.0.0

JavaScript file: node_demo.js

Application parameters:

Environment variables:

----End

API Calling Example (Node.js)

Step 1 Import **signer.js** to your project.

```
var signer = require('./signer')
var http = require('http')
```

Step 2 Generate a new signer and enter the AppKey and AppSecret.

```
var sig = new signer.Signer()
sig.Key = "4f5f****100c"
sig.Secret = "******"
```

Step 3 Generate a request, and specify the method, request URI, and request body.

```
var r = new signer.HttpRequest("POST", "c967a237-cd6c-470e-906f-
a8655461897e.apigw.exampleRegion.com/app1?a=1");
r.body = '{"a":1}'
```

Step 4 Add the x-stage header to the request to specify an environment name. Add other headers to be signed as necessary.

```
r.headers = { "x-stage":"RELEASE" }
```

Step 5 Execute the following function to generate HTTP(s) request parameters, and add the X-Sdk-Date and Authorization headers for signing the request:

```
var opts = sig.Sign(r)
```

Step 6 Access the API and view the access result. If you access the API using HTTPS, change **http.request** to **https.request**.

```
var req=http.request(opts, function(res){
        console.log(res.statusCode)
        res.on("data",    function(chunk){
        console.log(chunk.toString())
    })
})
req.on("error",function(err){
    console.log(err.message)
})
req.write(r.body)
req.end()
```

----End

API Calling Example (Browser)

Before calling an API with a browser, create an API that supports the OPTIONS method by following the instructions in section "Enabling CORS" in the *API Gateway User Guide*. When creating the API, enable CORS and add **Access-Control-Allow-*** fields in the response header.

Step 1 Import **signer.js** and dependencies to the HTML page.

```
<script src="js/hmac-sha256.js"></script>
<script src="js/moment.min.js"></script>
<script src="js/moment-timezone-with-data.min.js"></script>
<script src='signer.js'></script>
```

Step 2 Sign the request and access the API.

```
var sig = new signer.Signer()
sig.Key = "4f5f****100c"
sig.Secret = "******"
var r= new signer.HttpRequest()
r.host = "c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com"
r.method = "POST"
r.uri = "/app1"
r.body = '{"a":1}'
r.query = { "a":"1","b":"2" }
r.headers = { "Content-Type":"application/json" }
var opts = sig.Sign(r)
var scheme = "https"
$.ajax({
    type: opts.method,
    data: req.body,
    processData: false,
    url: scheme + "://" + opts.hostname + opts.path,
    headers: opts.headers,
    success: function (data) {
        $('#status').html('200')
        $('#recv').html(data)
    },
    error: function (resp) {
        if (resp.readyState === 4) {
            $('#status').html(resp.status)
            $('#recv').html(resp.responseText)
        } else {
            $('#status').html(resp.state())
        }
    },
    timeout: 1000
});
```

----End

2.8 PHP

Scenarios

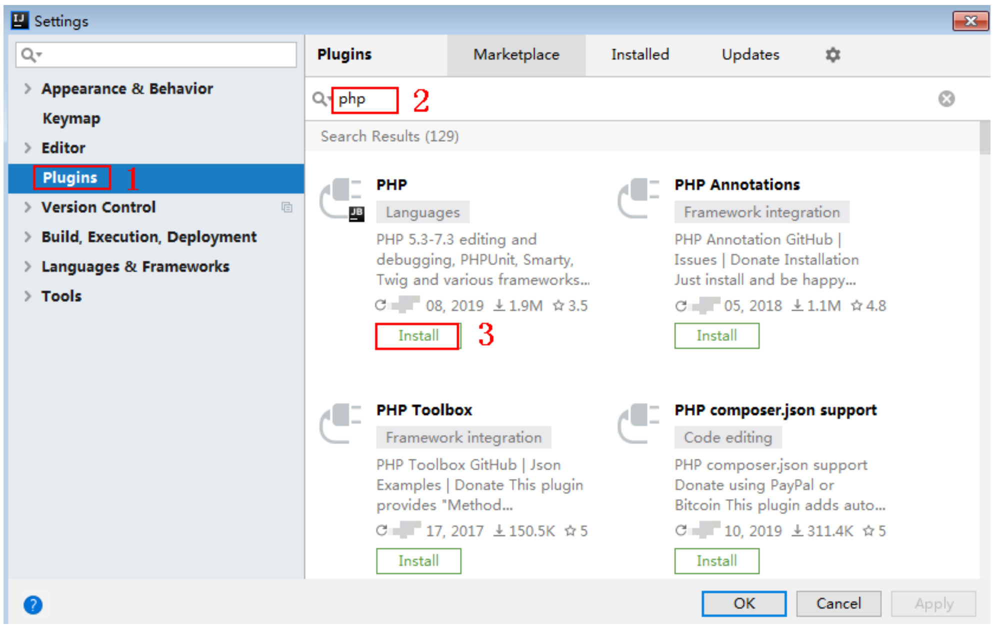
To use PHP to call an API through app authentication, obtain the PHP SDK, create a new project, and then call the API by referring to the API calling example.

This section uses IntelliJ IDEA 2018.3.5 as an example.

Prerequisites

- You have obtained API calling information. For details, see **Preparation**.
- You have installed IntelliJ IDEA 2018.3.5 or a later version. If not, download it from the **official IntelliJ IDEA website** and install it.
- You have installed PHP 8.0.3 or a later version. If not, download it from the **official PHP website** and install it.
- Copy the **php.ini-production** file from the PHP installation directory to the **C:\windows\** directory, rename the file as **php.ini**, and then add the following lines to the file:
  ```
  extension_dir = "PHP installation directory/ext"
  extension=openssl
  extension=curl
  ```
- You have installed the PHP plug-in on IntelliJ IDEA. If not, install the PHP plug-in according to **Figure 2-28**.

Figure 2-28 Installing the PHP plug-in


Settings
Appearance & Behavior
Keymap
Editor
Plugins
1
Version Control
Build, Execution, Deployment
Languages & Frameworks
Tools
Plugins
Marketplace
Installed
Updates
php
2
Search Results (129)
PHP
Languages
PHP 5.3-7.3 editing and debugging, PHPUnit, Smarty, Twig and various frameworks...
08, 2019 1.9M 3.5
Install
3
PHP Annotations
Framework integration
PHP Annotation GitHub | Issues | Donate Installation Just install and be happy...
05, 2018 1.1M 4.8
Install
PHP Toolbox
Framework integration
PHP Toolbox GitHub | Json Examples | Donate This plugin provides "Method...
17, 2017 150.5K 5
Install
PHP composer.json support
Code editing
PHP composer.json support Donate using PayPal or Bitcoin This plugin adds auto...
10, 2019 311.4K 5
Install
OK
Cancel
Apply

Obtaining the SDK

Log in to the APIG console, and download the SDK on the **SDKs** page by referring to section "SDKs" in the *API Gateway User Guide*.

Then obtain the **ApiGateway-php-sdk.zip** package. The following table shows the files decompressed from the package.

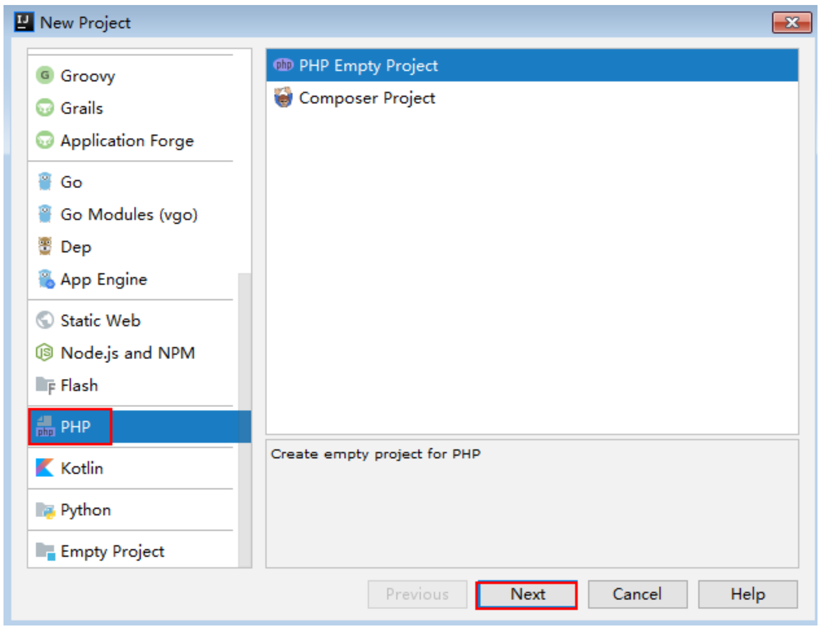
Name	Description
signer.php	SDK code
index.php	Sample code

Creating a Project

Step 1 Start IDEA and choose **File** > **New** > **Project**.

On the displayed **New Project** page, choose **PHP** and click **Next**.

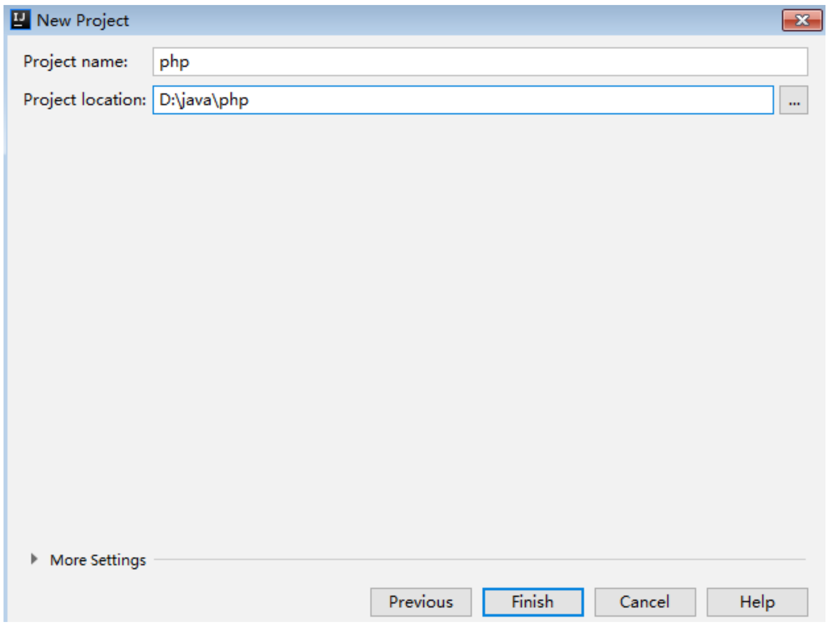
Figure 2-29 PHP



Create empty project for PHP

Step 2 Click **...**, select the directory where the SDK is decompressed, and click **Finish**.

Figure 2-30 Selecting the SDK directory of PHP after decompression

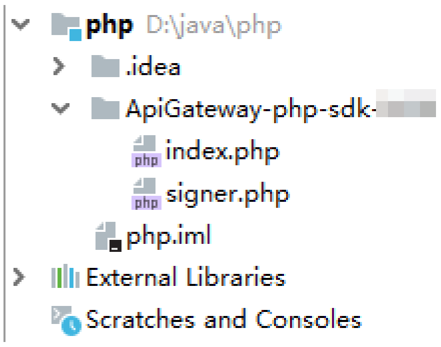


Project name: php

Project location: D:\java\php

Step 3 View the directory structure shown in the following figure.

Figure 2-31 Directory structure of the new project php



Modify the parameters in sample code **signer.php** as required. For details about the sample code, see **API Calling Example**.

----End

API Calling Example

Step 1 Import the PHP SDK to your code.

```
require 'signer.php';
```

Step 2 Generate a new signer and enter the AppKey and AppSecret.

```
$signer = new Signer();
$signer->Key = '4f5f****100c';
$signer->Secret = "******";
```

Step 3 Generate a new request and specify the method, request URL, and body. (The body should contain the actual request content.)

```
$req = new Request('GET', "https://c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com/
app1?a=1");
$req->body = '';
```

Step 4 Add the x-stage header to the request to specify an environment name. Add other headers to be signed as necessary.

```
$req->headers = array(
    'x-stage' => 'RELEASE',
);
```

Step 5 Execute the following function to generate a **$curl** context variable.

```
$curl = $signer->Sign($req);
```

Step 6 Access the API and view the access result.

```
$response = curl_exec($curl);
echo curl_getinfo($curl, CURLINFO_HTTP_CODE);
echo $response;
curl_close($curl);
```

----End

2.9 C++

Scenarios

To use C++ to call an API through App authentication, obtain the C++ SDK, and then call the API by referring to the API calling example.

Prerequisites

1. You have obtained API calling information. For details, see **Preparation**.
2. Install the OpenSSL library.
```
apt-get install libssl-dev
```
3. Install the curl library.
```
apt-get install libcurl4-openssl-dev
```

Obtaining the SDK

Log in to the APIG console, and download the SDK on the **SDKs** page by referring to section "SDKs" in the *API Gateway User Guide*.

Then obtain the **ApiGateway-cpp-sdk.zip** package. The following table shows the files decompressed from the package.

Name	Description
hasher.cpp	SDK code
hasher.h	
header.h	
RequestParams.cpp	
RequestParams.h	
signer.cpp	
signer.h	
Makefile	**Makefile** file
main.cpp	Sample code

API Calling Example

Step 1 Add the following references to **main.cpp**:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <curl/curl.h>
#include "signer.h"
```

Step 2 Generate a new signer and enter the AppKey and AppSecret.

```

Signer signer("4f5f****100c", "******");
```

Step 3 Generate a new **RequestParams** request, and specify the method, domain name, request URI, query strings, and request body.

```
 RequestParams* request = new RequestParams("POST", "c967a237-cd6c-470e-906f-
a8655461897e.apigw.exampleRegion.com", "/app1",
        "Action=ListUsers&Version=2010-05-08", "demo");
```

Step 4 Add the x-stage header to the request to specify an environment name. Add other headers to be signed as necessary.

```
request->addHeader("x-stage", "RELEASE");
```

Step 5 Execute the following function to add the generated headers to the request variable.

```
signer.createSignature(request);
```

Step 6 Use the curl library to access the API and view the access result.

```
static size_t
WriteMemoryCallback(void *contents, size_t size, size_t nmemb, void *userp)
{
    size_t realsize = size * nmemb;
    struct MemoryStruct *mem = (struct MemoryStruct *)userp;

    mem->memory = (char*)realloc(mem->memory, mem->size + realsize + 1);
    if (mem->memory == NULL) {
        /* out of memory! */
        printf("not enough memory (realloc returned NULL)\n");
        return 0;
    }

    memcpy(&(mem->memory[mem->size]), contents, realsize);
    mem->size += realsize;
    mem->memory[mem->size] = 0;

    return realsize;
}

//send http request using curl library
int perform_request(RequestParams* request)
{
    CURL *curl;
    CURLcode res;
    struct MemoryStruct resp_header;
    resp_header.memory = (char*)malloc(1);
    resp_header.size = 0;
    struct MemoryStruct resp_body;
    resp_body.memory = (char*)malloc(1);
    resp_body.size = 0;

    curl_global_init(CURL_GLOBAL_ALL);
    curl = curl_easy_init();

    curl_easy_setopt(curl, CURLOPT_CUSTOMREQUEST, request->getMethod().c_str());
    std::string url = "http://" + request->getHost() + request->getUri() + "?" + request->getQueryParams();
    curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
    struct curl_slist *chunk = NULL;
    std::set<Header>::iterator it;
    for (auto header : *request->getHeaders()) {
        std::string headerEntry = header.getKey() + ": " + header.getValue();
        printf("%s\n", headerEntry.c_str());
        chunk = curl_slist_append(chunk, headerEntry.c_str());
    }
    printf("-------------\n");
    curl_easy_setopt(curl, CURLOPT_HTTPHEADER, chunk);
    curl_easy_setopt(curl, CURLOPT_COPYPOSTFIELDS, request->getPayload().c_str());
    curl_easy_setopt(curl, CURLOPT_NOBODY, 0L);
    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteMemoryCallback);
    curl_easy_setopt(curl, CURLOPT_HEADERDATA, (void *)&resp_header);
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, (void *)&resp_body);
    //curl_easy_setopt(curl, CURLOPT_VERBOSE, 1L);
    res = curl_easy_perform(curl);
    if (res != CURLE_OK) {
        fprintf(stderr, "curl_easy_perform() failed: %s\n", curl_easy_strerror(res));
    }
    else {
        long status;
        curl_easy_getinfo(curl, CURLINFO_HTTP_CODE, &status);
        printf("status %d\n", status);
        printf(resp_header.memory);
        printf(resp_body.memory);
    }
    free(resp_header.memory);
    free(resp_body.memory);
    curl_easy_cleanup(curl);

    curl_global_cleanup();

    return 0;
}
```

Step 7 Run the **make** command to obtain a **main** executable file, execute the file, and then view the execution result.

----End

2.10 C

Scenarios

To use C to call an API through app authentication, obtain the C SDK, and then call the API by referring to the API calling example.

Prerequisites

1. You have obtained API calling information. For details, see **Preparation**.
2. Install the OpenSSL library.
   ```
   apt-get install libssl-dev
   ```
3. Install the curl library.
   ```
   apt-get install libcurl4-openssl-dev
   ```

Obtaining the SDK

Log in to the APIG console, and download the SDK on the **SDKs** page by referring to section "SDKs" in the *API Gateway User Guide*.

Then obtain the **ApiGateway-c-sdk.zip** package. The following table shows the files decompressed from the package.

Name	Description
signer_common.c	SDK code
signer_common.h	
signer.c	
signer.h	
Makefile	**Makefile** file
main.c	Sample code

API Calling Example

Step 1 Add the following references to **main.c**:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <curl/curl.h>
#include "signer.h"
```

Step 2 Generate a sig_params_t variable, and enter the AppKey and AppSecret.

```
sig_params_t params;
sig_params_init(&params);
sig_str_t app_key = sig_str("4f5f****100c");
sig_str_t app_secret = sig_str("******");
params.key = app_key;
params.secret = app_secret;
```

Step 3 Specify the method, domain name, request URI, query strings, and request body.

```
sig_str_t host = sig_str("c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com");
sig_str_t method = sig_str("GET");
sig_str_t uri = sig_str("/app1");
sig_str_t query_str = sig_str("a=1&b=2");
sig_str_t payload = sig_str("");
params.host = host;
params.method = method;
params.uri = uri;
params.query_str = query_str;
params.payload = payload;
```

Step 4 Add the x-stage header to the request to specify an environment name. Add other headers to be signed as necessary.

```
sig_headers_add(&params.headers, "x-stage", "RELEASE");
```

Step 5 Execute the following function to add the generated headers to the request variable.

```
sig_sign(&params);
```

Step 6 Use the curl library to access the API and view the access result.

```
static size_t
WriteMemoryCallback(void *contents, size_t size, size_t nmemb, void *userp)
{
    size_t realsize = size * nmemb;
    struct MemoryStruct *mem = (struct MemoryStruct *)userp;

    mem->memory = (char*)realloc(mem->memory, mem->size + realsize + 1);
    if (mem->memory == NULL) {
        /* out of memory! */
        printf("not enough memory (realloc returned NULL)\n");
        return 0;
    }

    memcpy(&(mem->memory[mem->size]), contents, realsize);
    mem->size += realsize;
    mem->memory[mem->size] = 0;

    return realsize;
}

//send http request using curl library
int perform_request(RequestParams* request)
{
    CURL *curl;
    CURLcode res;
    struct MemoryStruct resp_header;
    resp_header.memory = malloc(1);
    resp_header.size = 0;
    struct MemoryStruct resp_body;
    resp_body.memory = malloc(1);
    resp_body.size = 0;

    curl_global_init(CURL_GLOBAL_ALL);
    curl = curl_easy_init();

    curl_easy_setopt(curl, CURLOPT_CUSTOMREQUEST, params.method.data);
    char url[1024];
    sig_snprintf(url, 1024, "http://%V%V?%V", &params.host, &params.uri, &params.query_str);
    curl_easy_setopt(curl, CURLOPT_URL, url);
    struct curl_slist *chunk = NULL;
    for (int i = 0; i < params.headers.len; i++) {
        char header[1024];
        sig_snprintf(header, 1024, "%V: %V", &params.headers.data[i].name, &params.headers.data[i].value);
        printf("%s\n", header);
        chunk = curl_slist_append(chunk, header);
    }
    printf("-------------\n");
    curl_easy_setopt(curl, CURLOPT_HTTPHEADER, chunk);
    curl_easy_setopt(curl, CURLOPT_POSTFIELDS, params.payload.data);
    curl_easy_setopt(curl, CURLOPT_NOBODY, 0L);
    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteMemoryCallback);
    curl_easy_setopt(curl, CURLOPT_HEADERDATA, (void *)&resp_header);
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, (void *)&resp_body);
    //curl_easy_setopt(curl, CURLOPT_VERBOSE, 1L);
    res = curl_easy_perform(curl);
    if (res != CURLE_OK) {
        fprintf(stderr, "curl_easy_perform() failed: %s\n", curl_easy_strerror(res));
    }
    else {
        long status;
        curl_easy_getinfo(curl, CURLINFO_HTTP_CODE, &status);
        printf("status %d\n", status);
        printf(resp_header.memory);
        printf(resp_body.memory);
    }
    free(resp_header.memory);
    free(resp_body.memory);
    curl_easy_cleanup(curl);

    curl_global_cleanup();

    //free signature params
    sig_params_free(&params);
    return 0;
}
```

Step 7 Run the **make** command to obtain a **main** executable file, execute the file, and then view the execution result.

----End

2.11 Android

Scenarios

To use Android to call an API through app authentication, obtain the Android SDK, create a new project, and then call the API by referring to the API calling example.

Prerequisites

- You have obtained API calling information. For details, see **Preparation**.

- You have installed Android Studio 4.1.2 or a later version. If not, download it from the **official Android Studio website** and install it.

Obtaining the SDK

Log in to the APIG console, and download the SDK on the **SDKs** page by referring to section "SDKs" in the *API Gateway User Guide*.

Then obtain the **ApiGateway-android-sdk.zip** package. The following table shows the files decompressed from the package.

Name	Description
app\	Android project code
gradle\	Gradle files
build.gradle	Gradle configuration files
gradle.properties	
settings.gradle	
gradlew	Gradle Wrapper scripts
gradlew.bat	

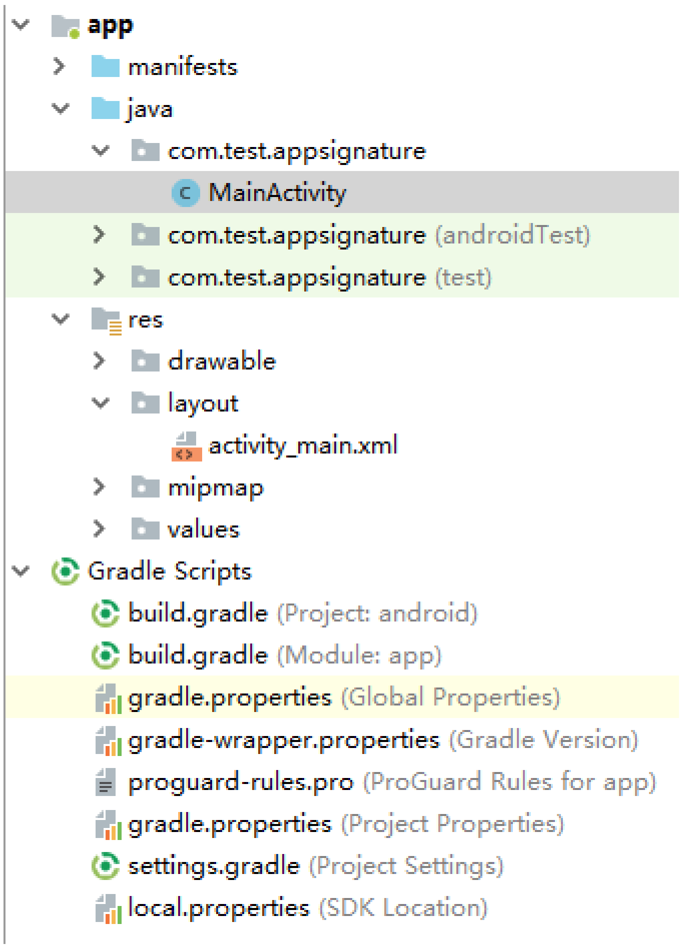
Opening a Project

Step 1 Start the Android Studio and choose **File** > **Open**.

Select the directory where the SDK is decompressed.

Step 2 View the directory structure of the project shown in the following figure.

Figure 2-32 Project directory structure


app
manifests
java
com.test.appsignature
MainActivity
com.test.appsignature (androidTest)
com.test.appsignature (test)
res
drawable
layout
activity_main.xml
mipmap
values
Gradle Scripts
build.gradle (Project: android)
build.gradle (Module: app)
gradle.properties (Global Properties)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)

----End

API Calling Example

Step 1 Add required JAR files to the **app/libs** directory of the Android project. The following JAR files must be included:

- java-sdk-core-x.x.x.jar
- commons-logging-1.2.jar
- joda-time-2.10.jar

Step 2 Add dependencies of the **okhttp** library to the **build.gradle** file.

Add **implementation 'com.squareup.okhttp3:okhttp:3.14.2'** in the **dependencies** field of the **build.gradle** file.

```
dependencies {
    ...
    ...
    implementation 'com.squareup.okhttp3:okhttp:3.14.3'
}
```

Step 3 Create a request, enter an AppKey and AppSecret, and specify the domain name, method, request URI, and body.

```
Request request = new Request();
try {
    request.setKey("4f5f****100c");
    request.setSecrect("******");
    request.setMethod("POST");
    request.setUrl("https://c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com/app1");
    request.addQueryStringParam("name", "value");
    request.addHeader("Content-Type", "text/plain");
    request.setBody("demo");
} catch (Exception e) {
    e.printStackTrace();
    return;
}
```

Step 4 Sign the request to generate an **okhttp3.Request** object for API access.

```
okhttp3.Request signedRequest = Client.signOkhttp(request);
OkHttpClient client = new OkHttpClient.Builder().build();
Response response = client.newCall(signedRequest).execute();
```

----End

2.12 curl

Scenarios

To use the curl command to call an API through app authentication, download the JavaScript SDK to generate the curl command, and copy the command to the CLI to call the API.

Prerequisites

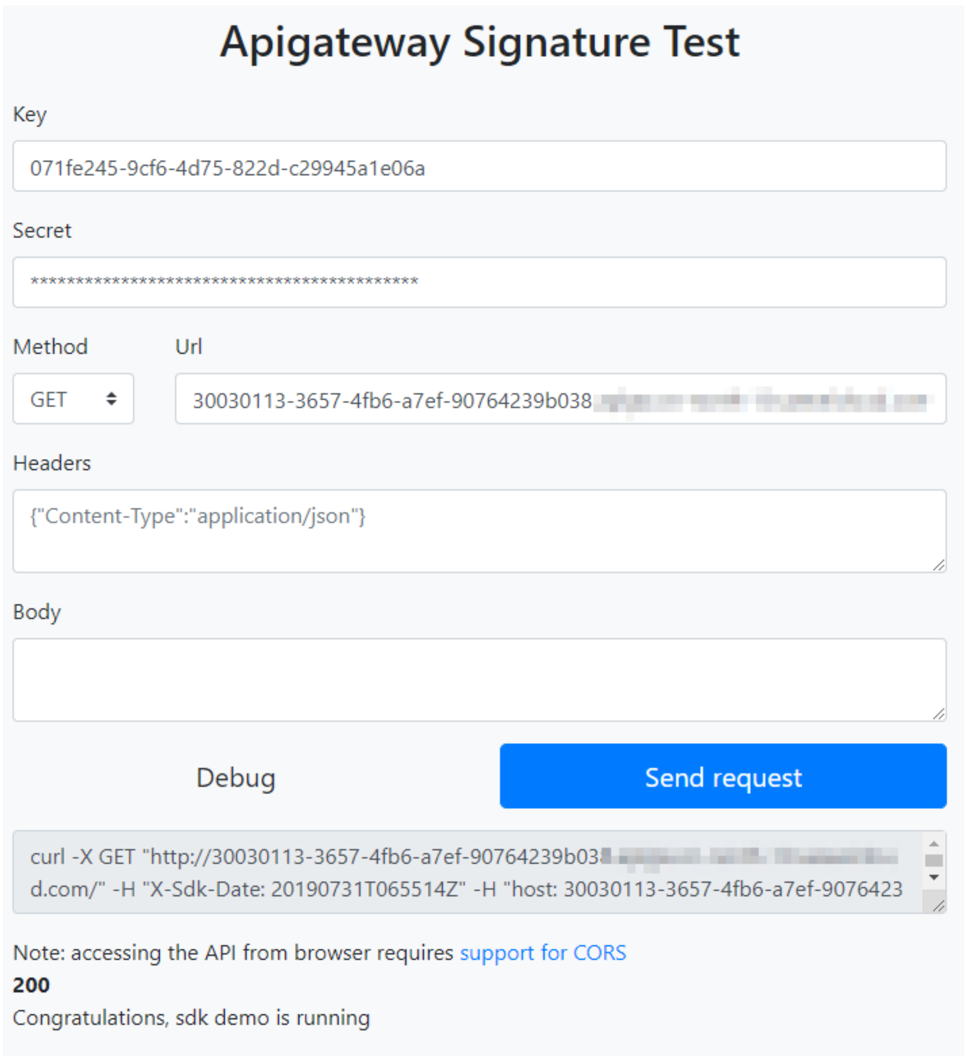
You have obtained API calling information. For details, see **Preparation**.

API Calling Example

Step 1 Use the JavaScript SDK to generate the curl command.

Log in to the APIG console, and download the SDK on the **SDKs** page by referring to section "SDKs" in the *API Gateway User Guide*.

Open demo.html in a browser. The following figure shows the demo page.


Apigateway Signature Test
Key
071fe245-9cf6-4d75-822d-c29945a1e06a
Secret
****
Method
Url
GET
30030113-3657-4fb6-a7ef-90764239b038
Headers
{"Content-Type":"application/json"}
Body
Debug
Send request

```
curl -X GET "http://30030113-3657-4fb6-a7ef-90764239b03
d.com/" -H "X-Sdk-Date: 20190731T065514Z" -H "host: 30030113-3657-4fb6-a7ef-9076423
```

Note: accessing the API from browser requires support for CORS
200
Congratulations, sdk demo is running

Step 2 Specify the key, secret, method, protocol, domain name, and URL. For example:

```
Key=4f5f****100c
Secret=******
Method=POST
Url=https://c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com
```

Step 3 Specify query and header parameters in JSON format, and set the request body.

Step 4 Click **Send request** to generate a **curl** command. Copy the **curl** command to the CLI to access the API.

```
$ curl -X POST "https://c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com/" -H "X-Sdk-
Date: 20180530T115847Z" -H "Authorization: SDK-HMAC-SHA256 Access=071fe245-9cf6-4d75-822d-
c29945a1e06a, SignedHeaders=host;x-sdk-date, Signature=9e5314bd156d517******dd3e5765fdde4" -d ""
Congratulations, sdk demo is running
```

NOTE

The **curl** command generated using an SDK does not meet the format requirements of Windows. Please run the **curl** command in Git Bash.

----End

3 Calling APIs Through IAM Authentication

3.1 Token Authentication

Scenarios

To call APIs using a token, add the token to the **X-Auth-Token** header in API requests.

NOTE

You can use either of the following authentication methods to call APIs:

- Token authentication: Requests are authenticated using a token.
- AK/SK authentication: Requests are encrypted using an access key ID (AK) and a secret access key (SK).

API Calling Example

1. Obtain a token and set it as an environment variable for authenticating the calling of other APIs.
   a. Obtain a token. An example request is as follows:

```
curl -X POST https://{iam_endpoint}/v3/auth/tokens -H 'content-type: application/json' -d '{
    "auth": {
        "identity": {
            "methods": [
                "password"
            ],
            "password": {
                "user": {
                "name": "{user_name}",
                    "domain": {
                        "name": "{user_name}"
                    },
            "password": "{password}"
                }
            }
        },
        "scope": {
            "project": {
```

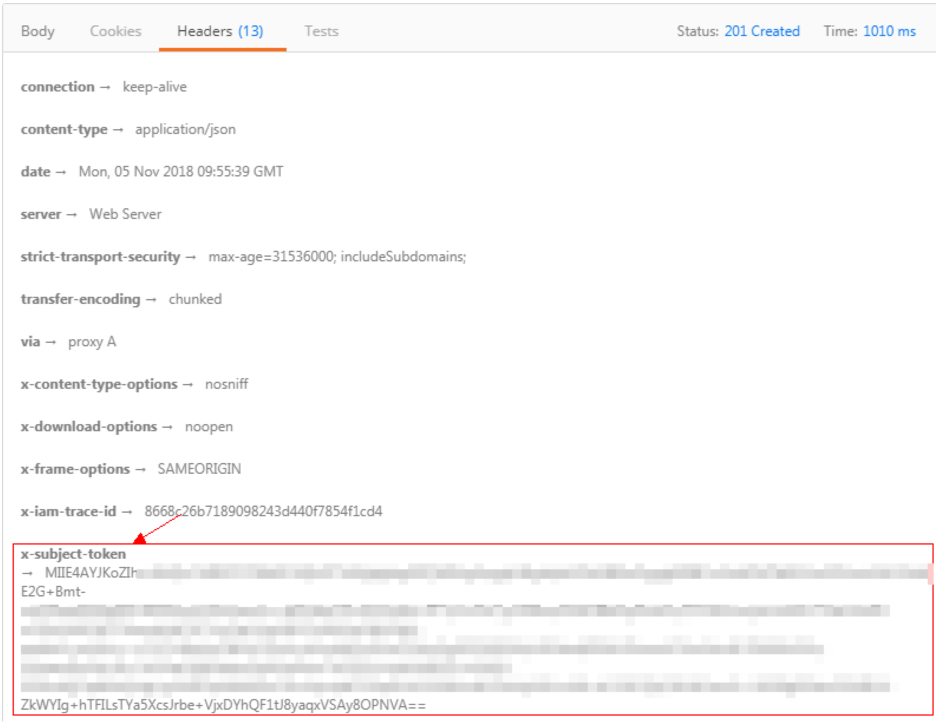
```
            "id": "{project_id}"
          }
        }
      }
}' -vk
```

Modify the parameters in the preceding command according to the following description. For details, see the API forObtaining a User Token in the *Identity and Access Management API Reference*.

- Obtain the ***{iam_endpoint}*** from **Regions and Endpoints**.
- Replace ***{user_name}*** and ***{password}*** respectively with the username and password of the IAM server.
- ***{project_id}***: The project ID. On the management console, hover the mouse pointer over the username in the upper right corner, choose **My Credentials** from the drop-down list, and then view the project ID on the displayed page.

The token value is the **X-Subject-Token** value in the response header as shown below:

Figure 3-1 Obtaining the X-Subject-Token response header



```
connection → keep-alive
content-type → application/json
date → Mon, 05 Nov 2018 09:55:39 GMT
server → Web Server
strict-transport-security → max-age=31536000; includeSubdomains;
transfer-encoding → chunked
via → proxy A
x-content-type-options → nosniff
x-download-options → noopen
x-frame-options → SAMEORIGIN
x-iam-trace-id → 8668c26b7189098243d440f7854f1cd4
x-subject-token
→ MIIE4AYJKoZIh[redacted]
E2G+Bmt-
ZkWYIg+hTFILsTYa5XcsJrbe+VjxDYhQF1tJ8yaqxVSAy8OPNVA==
```

b. Run the following command to set an environment variable for passing the token:

export Token=*{X-Subject-Token}*

X-Subject-Token is the token obtained in **1.a**. Example:

```
export Token=MIIDkg******BZQMEAgEwg
```

2. To call an API, obtain the domain name, request method, and URL by referring to **Preparation**, add **X-Auth-Token** to the request header, and set the value of **X-Auth-Token** to the token obtained in **1**. Set the parameters based on the site requirements.

```
curl -X Request_method Domain name+URL  -H "x-auth-token: $Token" -vk
```

3.2 AK/SK Authentication

This section describes how to use AK and SK to sign requests.

NOTE

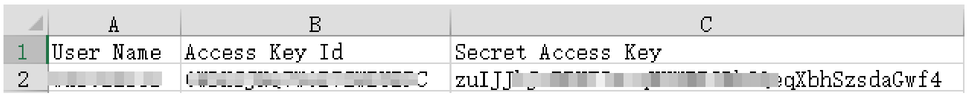
- AK: access key ID, which is a unique identifier used in conjunction with a secret access key to sign requests cryptographically.
- SK: secret access key used in conjunction with an AK to sign requests cryptographically. It identifies a request sender and prevents the request from being modified.

Generating an AK and SK Pair

If an AK/SK pair has already been generated, skip this step. Find the downloaded AK/SK file, which is usually named **credentials.csv**.

As shown in the following figure, the file contains the username, access key ID, and secret access key.

Figure 3-2 Content of the credential.csv file



	A	B	C
1	User Name	Access Key Id	Secret Access Key
2	[redacted]	[redacted]	zuIJJ[redacted]qXbhSzsdaGwf4

Perform the following procedure to generate an AK/SK pair:

1. Log in to the console.
2. Hover the mouse pointer over the username and choose **My Credentials** from the drop-down list.
3. Click the **Access Keys** tab.
4. Click **Create Access Key**.
5. Enter the verification code or login password as prompted, and click **OK** to download the access key. Keep the access key secure.

Generating a Signature

Generate a signature in the same way as in App authentication mode. Replace AppKey with AK and replace AppSecret with SK to complete the signing and request processing. You can sign requests to access APIs by using **Java**, **Go**, **Python**, **C#**, **JavaScript**, **PHP**, **C++**, **C**, and **Android**.

NOTICE

The local time on the client must be synchronized with the clock server to avoid a large error in the value of the **X-Sdk-Date** request header.

APIG (API Management) checks the time format and compares the time with the time when APIG (API Management) receives the request. If the time difference exceeds 15 minutes, APIG will reject the request.

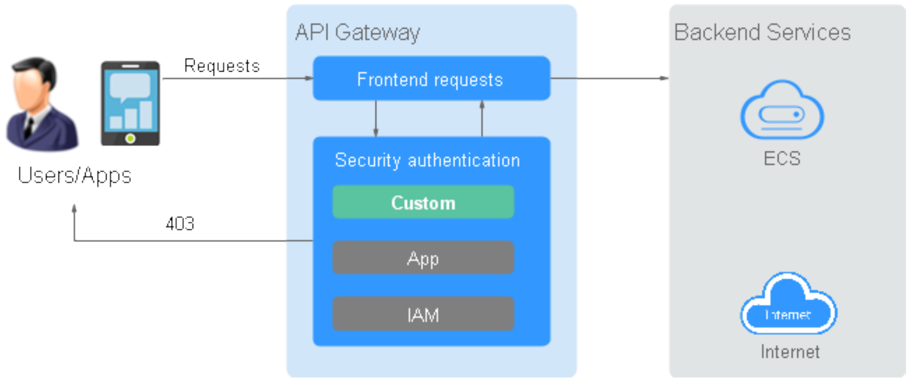
4 Creating a Function for Frontend Custom Authentication

Scenarios

API requests can be authenticated using a custom authorizer. You need to create a FunctionGraph function for frontend custom authentication and define the required authentication information in the function. The function then serves as a custom authentication backend to authenticate requests for the API.

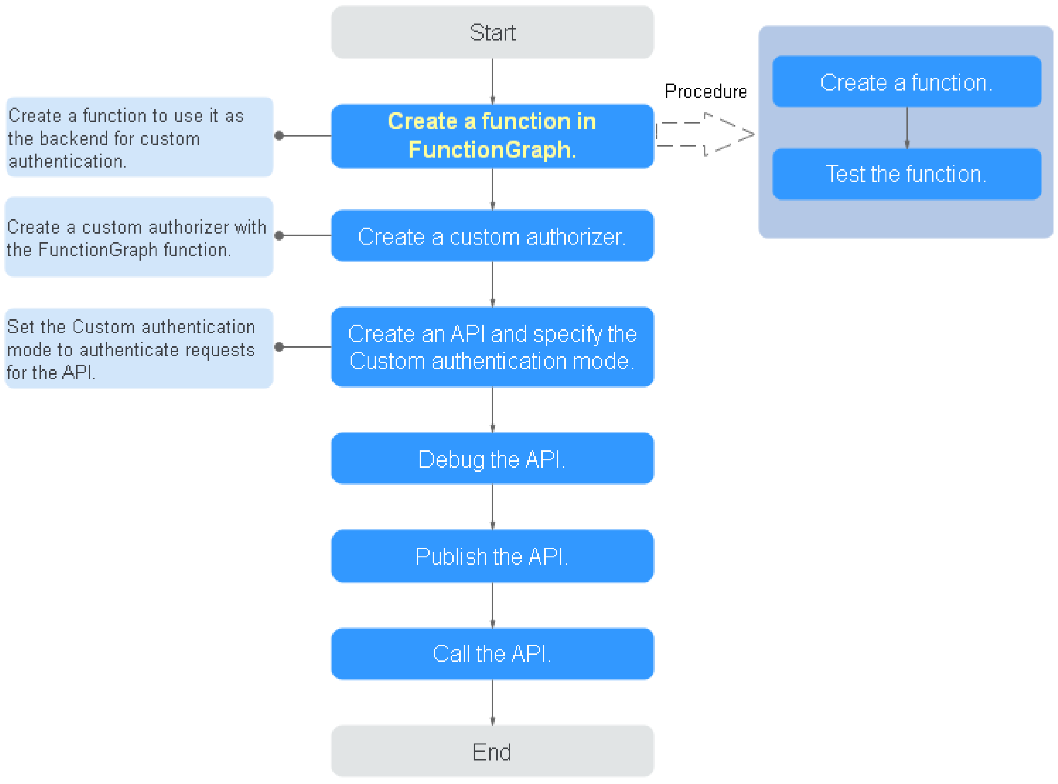
This section describes how to encapsulate a function into a "custom authorizer" and the important operations that need to be noted.

Figure 4-1 Schematic diagram of frontend custom authentication



The following figure shows the process of calling APIs using custom authentication.

Figure 4-2 Calling APIs by using custom authentication


Start
Create a function to use it as the backend for custom authentication.
Create a function in FunctionGraph.
Procedure
Create a function.
Test the function.
Create a custom authorizer with the FunctionGraph function.
Create a custom authorizer.
Set the Custom authentication mode to authenticate requests for the API.
Create an API and specify the Custom authentication mode.
Debug the API.
Publish the API.
Call the API.
End

NOTE

FunctionGraph is required for custom authorizers. If FunctionGraph is unavailable in the selected region, custom authorizers are not supported.

Procedure

Step 1 Create a function in FunctionGraph.

The function code must meet the following requirements (Python 2.7 is used as an example):

- The function code has defined three types of request parameters in the following formats:
  - Header parameters: event["headers"]["*Parameter name*"]
  - Query parameters: event["queryStringParameters"]["*Parameter name*"]
  - Custom user data: event["user_data"]
- The three types of request parameters obtained by the function are mapped to the custom authentication parameters defined in APIG.
  - Header parameter: Corresponds to the identity source specified in **Header** for custom authentication. The parameter value is transferred when the API that uses custom authentication is called.
  - Query parameter: Corresponds to the identity source specified in **Query** for custom authentication. The parameter value is transferred when the API that uses custom authentication is called.

   – Custom user data: Corresponds to the user data for custom authentication. The parameter value is specified when the custom authentication system is created.

- The response of the function cannot be greater than 1 MB and must be displayed in the following format:

```
{
    "statusCode":200,
    "body": "{\"status\": \"allow\", \"context\": {\"user\": \"abc\"}}"
}
```

  The **body** field is a character string, which is JSON-decoded as follows:

```
{
    "status": "allow/deny",
    "context": {
        "user": "abc"
    }
}
```

   – The **status** field is mandatory and is used to identify the authentication result. The authentication result can only be **allow** or **deny**. **allow** indicates that the authentication is successful, and **deny** indicates that the authentication fails.

   – The **context** field is optional. It can be key-value pairs, but the key value cannot be a JSON object or an array.

     The **context** field contains custom user data. After successful authentication, the user data is mapped to the backend parameters. The parameter name in **context** is case-sensitive and must be the same as the system parameter name. The parameter name must start with a letter and can contain 1 to 32 characters, including letters, digits, hyphens (-), and underscores (_).

     After successful frontend authentication, the value **abc** of **user** in **context** is mapped to the **test** parameter in the **Header** location of backend requests.

Example header parameters:

```
# -*- coding:utf-8 -*-
import json
def handler(event, context):
    if event["headers"].get("test")=='abc':
        resp = {
            'statusCode': 200,
            'body': json.dumps({
                "status":"allow",
                "context":{
                    "user":"abcd"
                }
            })
        }
    else:
        resp = {
            'statusCode': 200,
            'body': json.dumps({
                "status":"deny",
            })
        }
    return json.dumps(resp)
```

Example query parameters:

```
# -*- coding:utf-8 -*-
import json
def handler(event, context):
    if event["queryStringParameters"].get("test")=='abc':
        resp = {
            'statusCode': 200,
            'body': json.dumps({
                "status":"allow",
                "context":{
                    "user":"abcd"
                }
            })
        }
    else:
        resp = {
            'statusCode': 200,
            'body': json.dumps({
                "status":"deny",
            })
        }
    return json.dumps(resp)
```

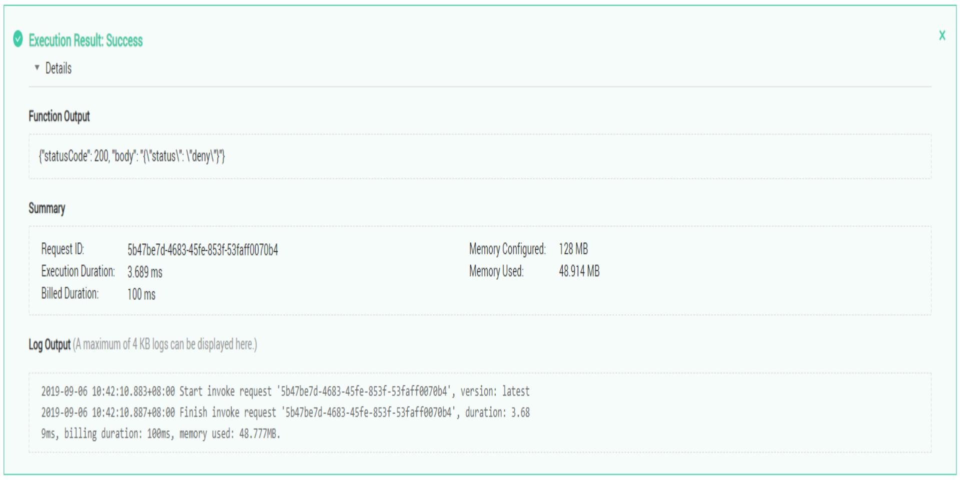
Example user data:

```
# -*- coding:utf-8 -*-
import json
def handler(event, context):
    if event.get("user_data")=='abc':
        resp = {
            'statusCode': 200,
            'body': json.dumps({
                "status":"allow",
                "context":{
                    "user":"abcd"
                }
            })
        }
    else:
        resp = {
            'statusCode': 200,
            'body': json.dumps({
                "status":"deny",
            })
        }
    return json.dumps(resp)
```

Step 2 Test the function. In the **Configure Test Event** dialog box, select **apig-event-template**, edit the test event, and click **Save**. Then click **Test**.

If the execution result is **Success**, the test is successful.

Next, you need to go to the APIG console to create a frontend custom authorizer.



Function Output

```
{"statusCode": 200, "body": "{\"status\": \"deny\"}"}
```

Summary

Request ID:	5b47be7d-4683-45fe-853f-53faff0070b4	Memory Configured:	128 MB
Execution Duration:	3.689 ms	Memory Used:	48.914 MB
Billed Duration:	100 ms		

Log Output (A maximum of 4 KB logs can be displayed here.)

```
2019-09-06 10:42:10.883+08:00 Start invoke request '5b47be7d-4683-45fe-853f-53faff0070b4', version: latest
2019-09-06 10:42:10.887+08:00 Finish invoke request '5b47be7d-4683-45fe-853f-53faff0070b4', duration: 3.68
9ms, billing duration: 100ms, memory used: 48.777MB.
```

----End

Follow-Up Operations

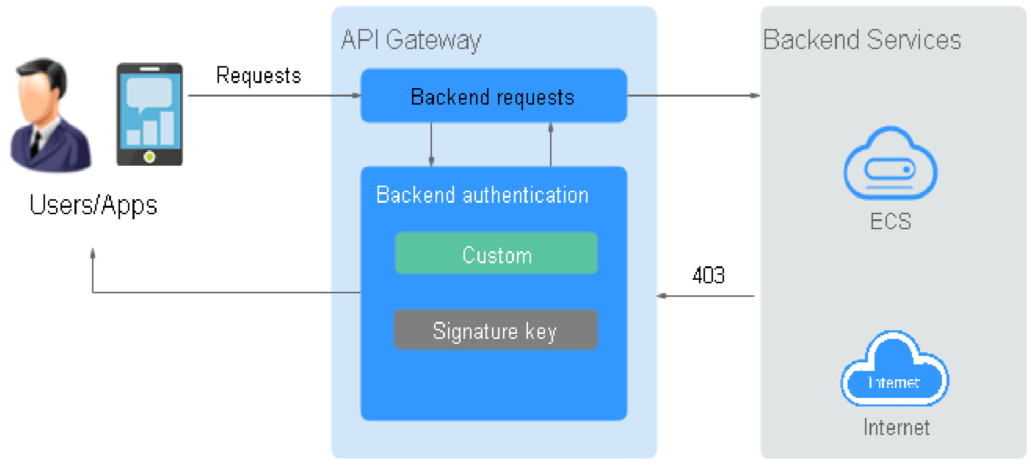
Create a custom authorizer for frontend authentication on the APIG console. For details, see section "Create a custom authorizer" in the *API Gateway User Guide*.

5 Creating a Function for Backend Custom Authentication

Scenarios

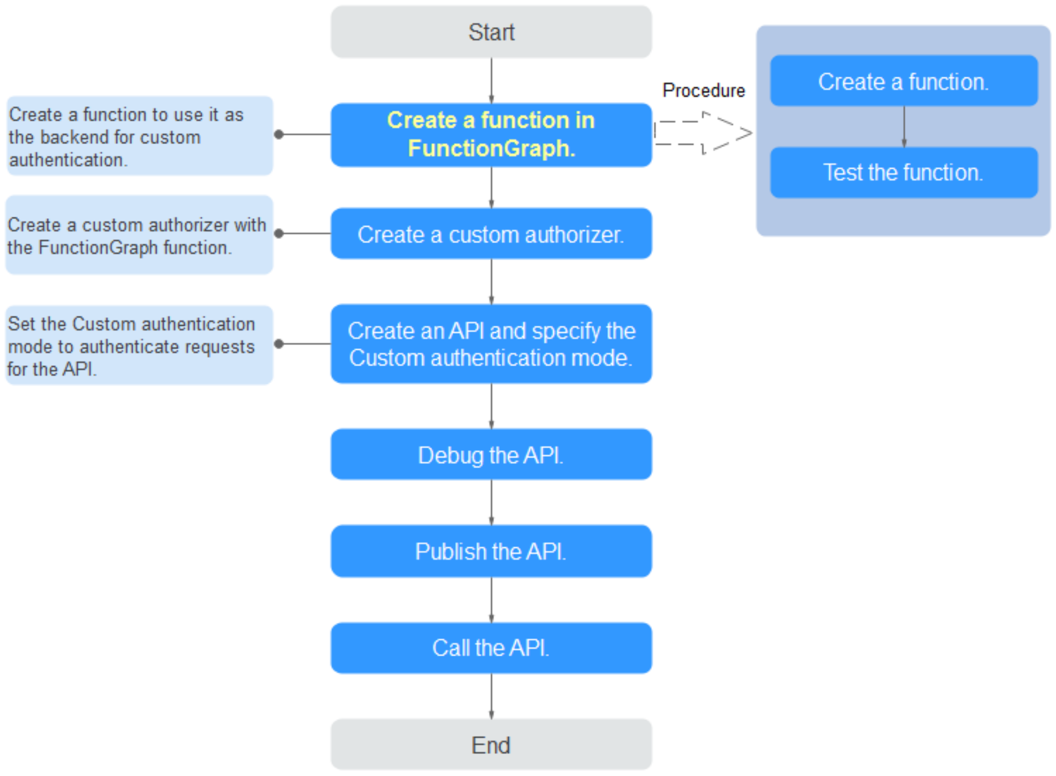
To protect backend services, you can use multiple external authentication systems by configuring one authentication mechanism. You need to create a FunctionGraph function for backend custom authentication and define the required authentication information in the function. The function then serves as a custom authentication backend to authenticate requests forwarded by APIG.

Figure 5-1 Schematic diagram of backend custom authentication



The following figure shows the process of calling APIs using custom authentication.

Figure 5-2 Calling APIs by using custom authentication


Start
Create a function to use it as the backend for custom authentication.
Create a function in FunctionGraph.
Procedure
Create a function.
Test the function.
Create a custom authorizer with the FunctionGraph function.
Create a custom authorizer.
Set the Custom authentication mode to authenticate requests for the API.
Create an API and specify the Custom authentication mode.
Debug the API.
Publish the API.
Call the API.
End

NOTE

FunctionGraph is required for custom authorizers. If FunctionGraph is unavailable in the selected region, custom authorizers are not supported.

Procedure

Step 1 Create a function in FunctionGraph.

The function code must meet the following requirements (Python 2.7 is used as an example):

- The custom user data contained in the function code must be in the following format: event["user_data"].
- The custom user data corresponds to the user data defined for the custom authorizer. You can define the user data in any format.
- The response of the function cannot be greater than 1 MB and must be displayed in the following format:

```
{
    "statusCode":200,
    "body": "{\"status\": \"allow\", \"context\": {\"user\": \"abc\"}}"
}
```

The **body** field is a character string, which is JSON-decoded as follows:

```
{
    "status": "allow/deny",
    "context": {
        "user": "abc"
    }
}
```

– The **status** field is mandatory and is used to identify the authentication result. The authentication result can only be **allow** or **deny**. **allow** indicates that the authentication is successful, and **deny** indicates that the authentication fails.

– The **context** field is optional. It can be key-value pairs, but the key value cannot be a JSON object or an array.

  The **context** field contains custom user data. After successful authentication, the user data is mapped to the backend parameters. The parameter name in **context** is case-sensitive and must be the same as the system parameter name. The parameter name in **context** must start with a letter and contain 1 to 32 characters, including uppercase letters, lowercase letters, digits, underscores (_), and hyphens (-).

  After successful backend authentication, the value **abc** of **user** in **context** is mapped to the **test** parameter in the **Header** location of backend requests and passed to the backend service.

Example user data:

```
# -*- coding:utf-8 -*-
import json
import base64
def handler(event, context):
    exampleuserdata=base64.b64encode(event["user_data"])
    resp = {
        'statusCode': 200,
        'body': json.dumps({
            "status":"allow",
            "context":{
                "user":exampleuserdata
            }
        })
    }
    return json.dumps(resp)
```

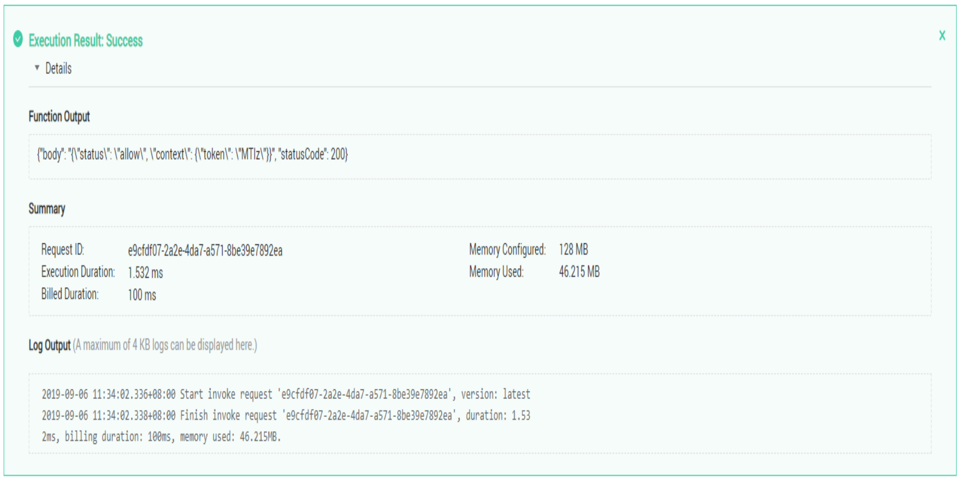
Step 2 Test the function. In the **Configure Test Event** dialog box, select **blank-template**, and set the following test event:

```
{"user_data": "123"}
```

Click **Save**. Then click **Test**.

If the execution result is **Success**, the test is successful.

Next, you need to go to the APIG console to create a backend custom authorizer.



Function Output

```
{"body": "{\"status\": \"allow\", \"context\": {\"token\": \"MTIz\"}}", "statusCode": 200}
```

Summary

```
Request ID:          e9cfdf07-2a2e-4da7-a571-8be39e7892ea        Memory Configured:   128 MB
Execution Duration:  1.532 ms                                     Memory Used:         46.215 MB
Billed Duration:     100 ms
```

Log Output (A maximum of 4 KB logs can be displayed here.)

```
2019-09-06 11:34:02.336+08:00 Start invoke request 'e9cfdf07-2a2e-4da7-a571-8be39e7892ea', version: latest
2019-09-06 11:34:02.338+08:00 Finish invoke request 'e9cfdf07-2a2e-4da7-a571-8be39e7892ea', duration: 1.53
2ms, billing duration: 100ms, memory used: 46.215MB.
```

----End

Follow-Up Operations

Create a custom authorizer for backend authentication on the APIG console. For details, see section "Create a custom authorizer" in the *API Gateway User Guide*.

6 Creating Signatures for Backend Requests

6.1 Pre-signature Preparations

Obtaining signature key information:

On the APIG instance console, choose **API Management** > **API Policies** > **Policies**, click the *signature key name* to go to the details page, and view the key and secret in the **Policy Content** area.

6.2 Java

Scenarios

To use Java to sign backend requests, obtain the Java SDK, import the project, and verify the backend signature by referring to the example provided in this section.

This section uses IntelliJ IDEA 2018.3.5 as an example.

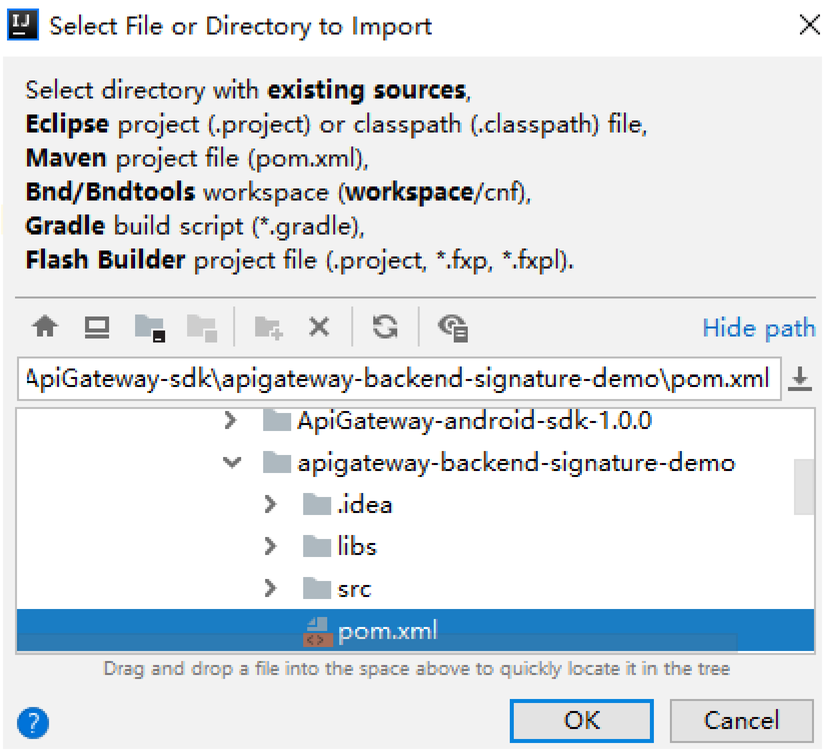
Prerequisites

- You have created a signature key on the APIG console and bound it to the API to be called. For more information, see section "Signature Keys" in the *API Gateway User Guide*.
- You have obtained the signature key and secret. For details, see **Pre-signature Preparations**.
- In the APIG console, download the SDK on the **SDKs** page by referring to section "SDKs" in the *API Gateway User Guide*.
- You have installed IntelliJ IDEA 2018.3.5 or a later version. If not, download it from the **official IntelliJ IDEA website** and install it.
- You have installed Java Development Kit (JDK) 1.8.111 or a later version. If not, download JDK from the **official Oracle website** and install it. JDK 17 or later is not supported.

Importing a Project

Step 1 Open IntelliJ IDEA, choose **File** > **New** > **Project from Existing Sources**, select the **apigateway-backend-signature-demo\pom.xml** file, and click **OK**.

Figure 6-1 Select File or Directory to Import


Select File or Directory to Import

Select directory with **existing sources**,
Eclipse project (.project) or classpath (.classpath) file,
Maven project file (pom.xml),
Bnd/Bndtools workspace (**workspace**/cnf),
Gradle build script (*.gradle),
Flash Builder project file (.project, *.fxp, *.fxpl).

ApiGateway-sdk\apigateway-backend-signature-demo\pom.xml

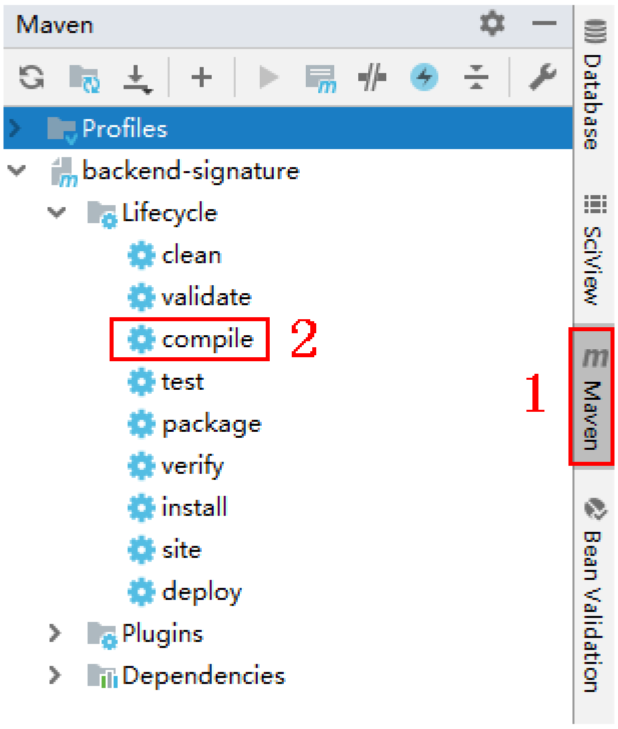
```
> ApiGateway-android-sdk-1.0.0
v apigateway-backend-signature-demo
    > .idea
    > libs
    > src
      pom.xml
```

Drag and drop a file into the space above to quickly locate it in the tree

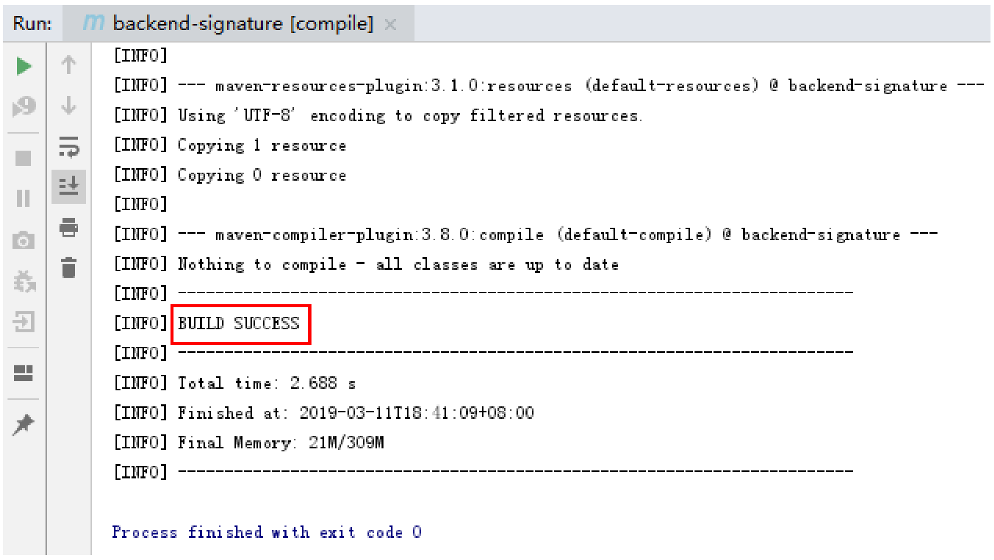
Step 2 Retain the default settings, click **Next** for the following four steps, and then click **Finish**.

Step 3 On the **Maven** tab page on the right, double-click **compile** to compile the file.

Figure 6-2 Compiling the project



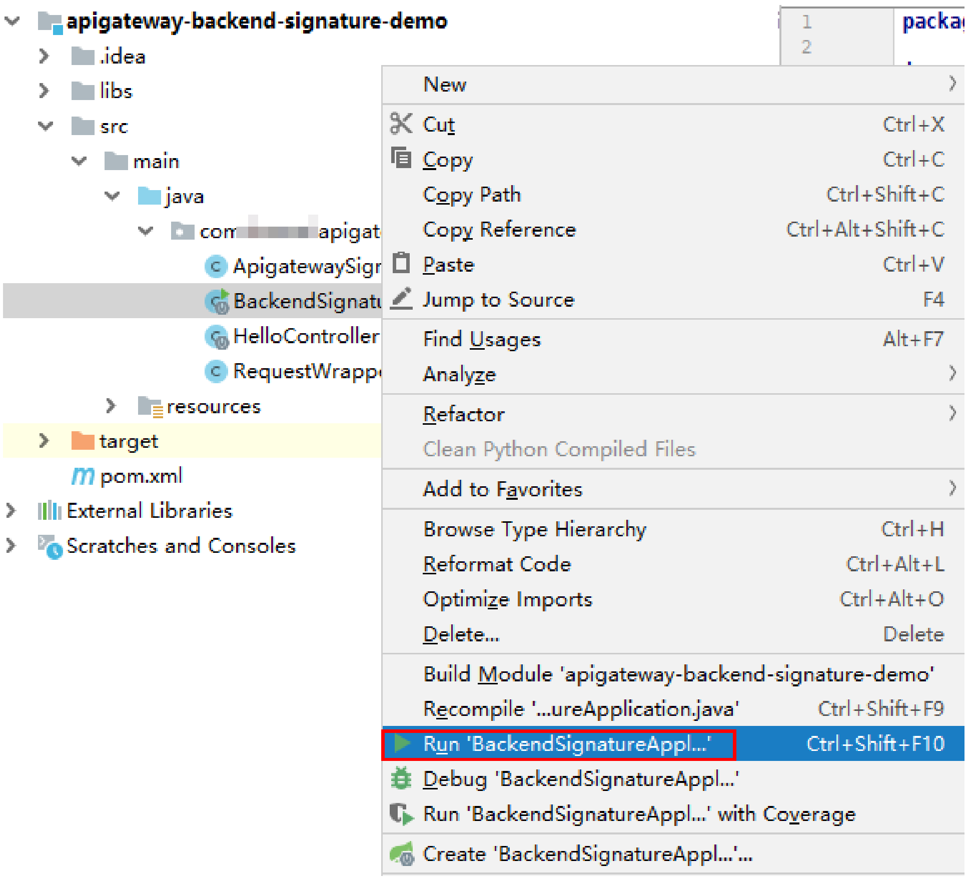
If the message "BUILD SUCCESS" is displayed, the compilation is successful.



```
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:resources (default-resources) @ backend-signature ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:compile (default-compile) @ backend-signature ---
[INFO] Nothing to compile - all classes are up to date
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.688 s
[INFO] Finished at: 2019-03-11T18:41:09+08:00
[INFO] Final Memory: 21M/309M
[INFO] ------------------------------------------------------------------------

Process finished with exit code 0
```

Step 4 Right-click **BackendSignatureApplication** and choose **Run**.

Figure 6-3 Running the BackendSignatureApplication service


apigateway-backend-signature-demo
.idea
libs
src
main
java
com apigat
ApigatewaySign
BackendSignatu
HelloController
RequestWrappe
resources
target
pom.xml
External Libraries
Scratches and Consoles
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Shift+C
Copy Reference Ctrl+Alt+Shift+C
Paste Ctrl+V
Jump to Source F4
Find Usages Alt+F7
Analyze
Refactor
Clean Python Compiled Files
Add to Favorites
Browse Type Hierarchy Ctrl+H
Reformat Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
Delete... Delete
Build Module 'apigateway-backend-signature-demo'
Recompile '...ureApplication.java' Ctrl+Shift+F9
Run 'BackendSignatureAppl...' Ctrl+Shift+F10
Debug 'BackendSignatureAppl...'
Run 'BackendSignatureAppl...' with Coverage
Create 'BackendSignatureAppl...'...

Modify the parameters in sample code **ApigatewaySignatureFilter.java** as required. For details about the sample code, see **Backend Signature Verification Example**.

----End

Backend Signature Verification Example

This example demonstrates how to build a Spring boot–based server as the backend of an API and implement a filter to verify the signature of requests sent from APIG (API Management).

NOTE

Signature information is added to requests sent to access the backend of an API only after a signature key is bound to the API.

Step 1 Compile a controller that matches all request paths and methods and set the return body to **Hello World!**

```
// HelloController.java

@RestController
@EnableAutoConfiguration
public class HelloController {

    @RequestMapping("/*")
    private String index() {
        return "Hello World!";
    }
}
```

Step 2 Compile a filter that matches all request paths and methods, and put the signature key and secret in a **Map**.

```
// ApigatewaySignatureFilter.java

@Component
@WebFilter(filterName = "ApigatewaySignatureFilter", urlPatterns = "/*")
public class ApigatewaySignatureFilter implements Filter {
    private static Map<String, String> secrets = new HashMap<>();
    static {
        secrets.put("signature_key1", "signature_secret1");
        secrets.put("signature_key2", "signature_secret2");
    }

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain chain) {
        //Signature verification code
        ...
    }
}
```

Step 3 To ensure that the body can be read in the filter and controller, wrap the request and send it to the filter and controller. The doFilter function is used for signature verification. For the implementation of wrapper classes, see RequestWrapper.java.

```
RequestWrapper request = new RequestWrapper((HttpServletRequest) servletRequest);
```

Step 4 Use a regular expression to parse the **Authorization** header to obtain **signingKey** and **signedHeaders**.

```
private static final Pattern authorizationPattern = Pattern.compile("SDK-HMAC-SHA256\\s+Access=([^,]+),\
\s?SignedHeaders=([^,]+),\\s?Signature=(\\w+)");

...

String authorization = request.getHeader("Authorization");
if (authorization == null || authorization.length() == 0) {
    response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Authorization not found.");
    return;
}

Matcher m = authorizationPattern.matcher(authorization);
if (!m.find()) {
    response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Authorization format incorrect.");
    return;
}
String signingKey = m.group(1);
String signingSecret = secrets.get(signingKey);
if (signingSecret == null) {
    response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Signing key not found.");
    return;
}
String[] signedHeaders = m.group(2).split(";");
```

For example, for **Authorization** header:

```
SDK-HMAC-SHA256 Access=signature_key1, SignedHeaders=host;x-sdk-date,
Signature=e11adf65a20d1b82c25419b5********8d0ba12fed1ceb13ed00
```

The parsing result is as follows:

```
signingKey=signature_key1
signedHeaders=host;x-sdk-date
```

Step 5 Find **signingSecret** based on **signingKey**. If **signingKey** does not exist, the authentication failed.

```
String signingSecret = secrets.get(signingKey);
if (signingSecret == null) {
    response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Signing key not found.");
    return;
}
```

Step 6 Create a request, and add the method, URL, query, and signedHeaders headers to the request. Determine whether the body needs to be set.

The body is read if there is no **x-sdk-content-sha256** header with value **UNSIGNED-PAYLOAD**.

```
Request apiRequest = new DefaultRequest();
apiRequest.setHttpMethod(HttpMethodName.valueOf(request.getMethod()));
String url = request.getRequestURL().toString();
String queryString = request.getQueryString();
try {
    apiRequest.setEndpoint((new URL(url)).toURI());
    Map<String, String> parametersmap = new HashMap<>();
    if (null != queryString && !"".equals(queryString)) {
        String[] parameterarray = queryString.split("&");
        for (String p : parameterarray) {
            String[] p_split = p.split("=", 2);
            String key = p_split[0];
            String value = "";
            if (p_split.length >= 2) {
                value = p_split[1];
            }
            parametersmap.put(URLDecoder.decode(key, "UTF-8"), URLDecoder.decode(value, "UTF-8"));
        }
        apiRequest.setParameters(parametersmap); //set query
    }
} catch (URISyntaxException e) {
    e.printStackTrace();
}

boolean needbody = true;
String dateHeader = null;
for (int i = 0; i < signedHeaders.length; i++) {
    String headerValue = request.getHeader(signedHeaders[i]);
    if (headerValue == null || headerValue.length() == 0) {
        ((HttpServletResponse) response).sendError(HttpServletResponse.SC_UNAUTHORIZED, "signed
header" + signedHeaders[i] + " not found.");
    } else {
        apiRequest.addHeader(signedHeaders[i], headerValue);//set header
        if (signedHeaders[i].toLowerCase().equals("x-sdk-content-sha256") &&
headerValue.equals("UNSIGNED-PAYLOAD")) {
            needbody = false;
        }
        if (signedHeaders[i].toLowerCase().equals("x-sdk-date")) {
            dateHeader = headerValue;
        }
    }
}

if (needbody) {
        apiRequest.setContent(new ByteArrayInputStream(request.getBody()));    //set body
}
```

Step 7 Check whether the signature has expired. Obtain the time from the **X-Sdk-Date** header, and check whether the difference between this time and the server time is within 15 minutes. If **signedHeaders** does not contain **X-Sdk-Date**, the authentication failed.

```
private static final DateTimeFormatter timeFormatter =
DateTimeFormat.forPattern("yyyyMMdd'T'HHmmss'Z'").withZoneUTC();
```

```
...

if (dateHeader == null) {
    response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Header x-sdk-date not found.");
    return;
}
long date = timeFormatter.parseMillis(dateHeader);
long duration = Math.abs(DateTime.now().getMillis() - date);
if (duration > 15 * 60 * 1000) {
    response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Signature expired.");
    return;
}
```

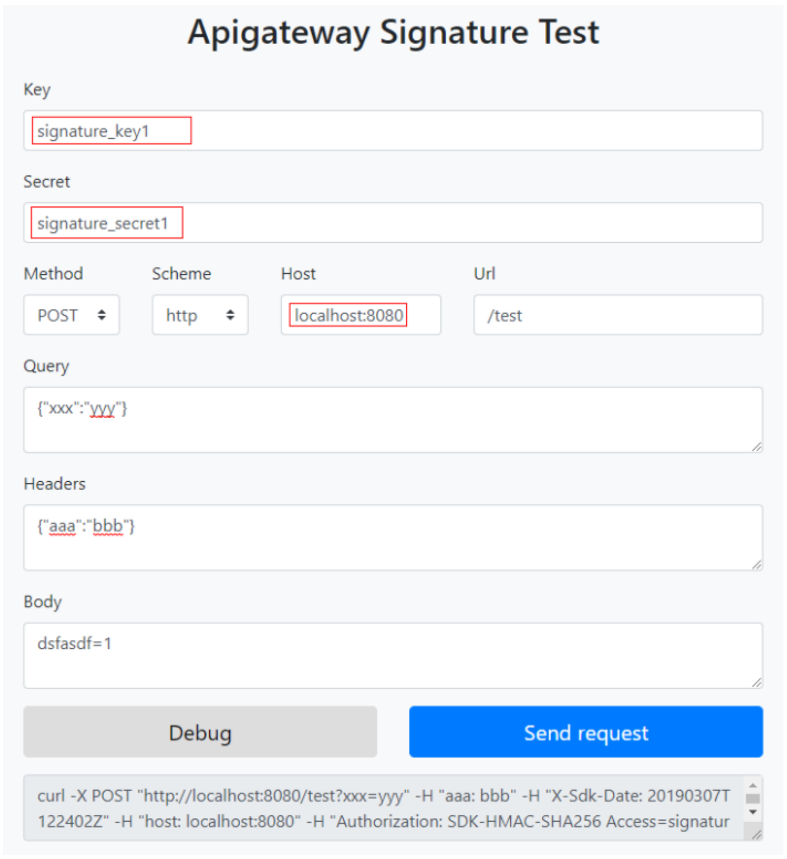
Step 8 Add the **Authorization** header to the request, and invoke the **verify** method to verify the request signature. If the verification is successful, the next filter is executed. Otherwise, the authentication failed.

```
apiRequest.addHeader("authorization", authorization);
apiRequest.setKey(signingKey);
apiRequest.setSecret(signingSecret);
Signer signer = new Signer();
boolean verify = signer.verify(apiRequest);
if (verify) {
    chain.doFilter(request, response);
} else {
    response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Verify authroization failed.");
}
```

Step 9 Run the server to verify the code. The following example uses the **HTML signature tool** in the JavaScript SDK to generate a signature.

Set the parameters according to the following figure, and click **Send request**. Copy the generated curl command, execute it in the CLI, and check whether the server returns **Hello World!**

If an incorrect key or secret is used, the server returns **401**, which means authentication failure.



Key: signature_key1

Secret: signature_secret1

Method: POST | Scheme: http | Host: localhost:8080 | Url: /test

Query: {"xxx":"yyy"}

Headers: {"aaa":"bbb"}

Body: dsfasdf=1

```
curl -X POST "http://localhost:8080/test?xxx=yyy" -H "aaa: bbb" -H "X-Sdk-Date: 20190307T
122402Z" -H "host: localhost:8080" -H "Authorization: SDK-HMAC-SHA256 Access=signatur
```

----End

6.3 Python

Scenarios

To use Python to sign backend requests, obtain the Python SDK, import the project, and verify the backend signature by referring to the example provided in this section.

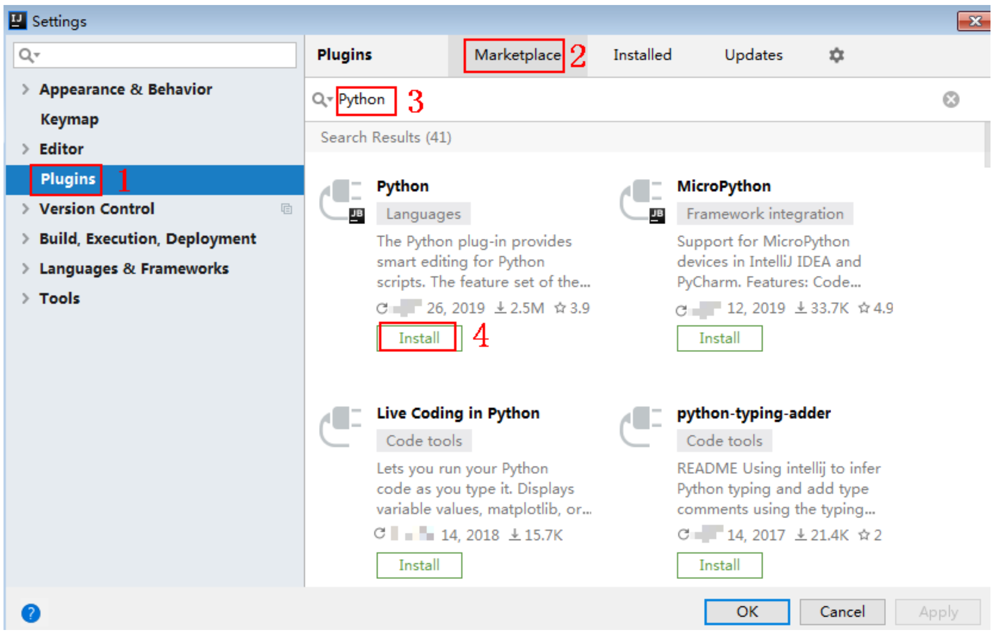
This section uses IntelliJ IDEA 2018.3.5 as an example.

Preparing the Environment

- You have created a signature key on the APIG console and bound it to the API to be called. For more information, see section "Signature Keys" in the *API Gateway User Guide*.
- You have obtained the signature key and secret. For details, see **Pre-signature Preparations**.
- Log in to the APIG console, and download the SDK on the **SDKs** page by referring to section "SDKs" in the *API Gateway User Guide*.
- You have installed Python 2.7.9 or 3.X. If not, download the Python installation package from the **official Python website** and install it.
- You have installed IntelliJ IDEA 2018.3.5 or a later version. If not, download it from the **official IntelliJ IDEA website** and install it.

- You have installed the Python plug-in on IntelliJ IDEA. If not, install the Python plug-in according to **Figure 6-4**.

Figure 6-4 Installing the Python plug-in

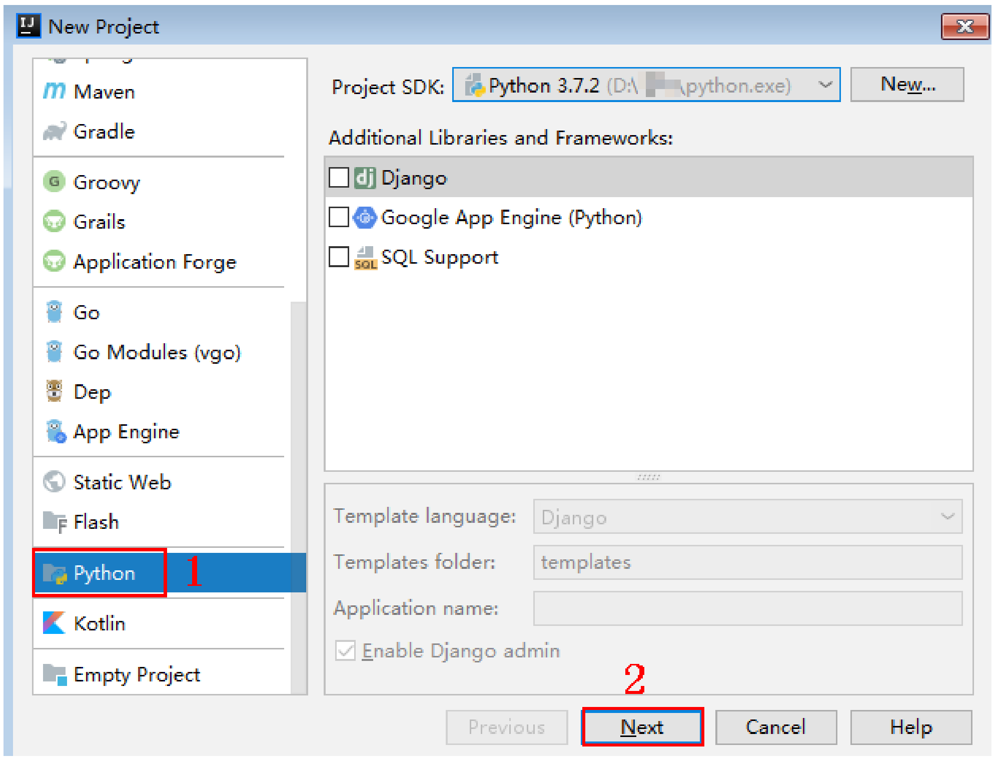

Settings
Appearance & Behavior
Keymap
Editor
Plugins 1
Version Control
Build, Execution, Deployment
Languages & Frameworks
Tools
Plugins
Marketplace 2
Installed
Updates
Python 3
Search Results (41)
Python
Languages
The Python plug-in provides smart editing for Python scripts. The feature set of the...
Install 4
MicroPython
Framework integration
Support for MicroPython devices in IntelliJ IDEA and PyCharm. Features: Code...
Install
Live Coding in Python
Code tools
Lets you run your Python code as you type it. Displays variable values, matplotlib, or...
Install
python-typing-adder
Code tools
README Using intellij to infer Python typing and add type comments using the typing...
Install
OK
Cancel
Apply

Importing a Project

Step 1 Start IntelliJ IDEA and choose **File** > **New** > **Project**.

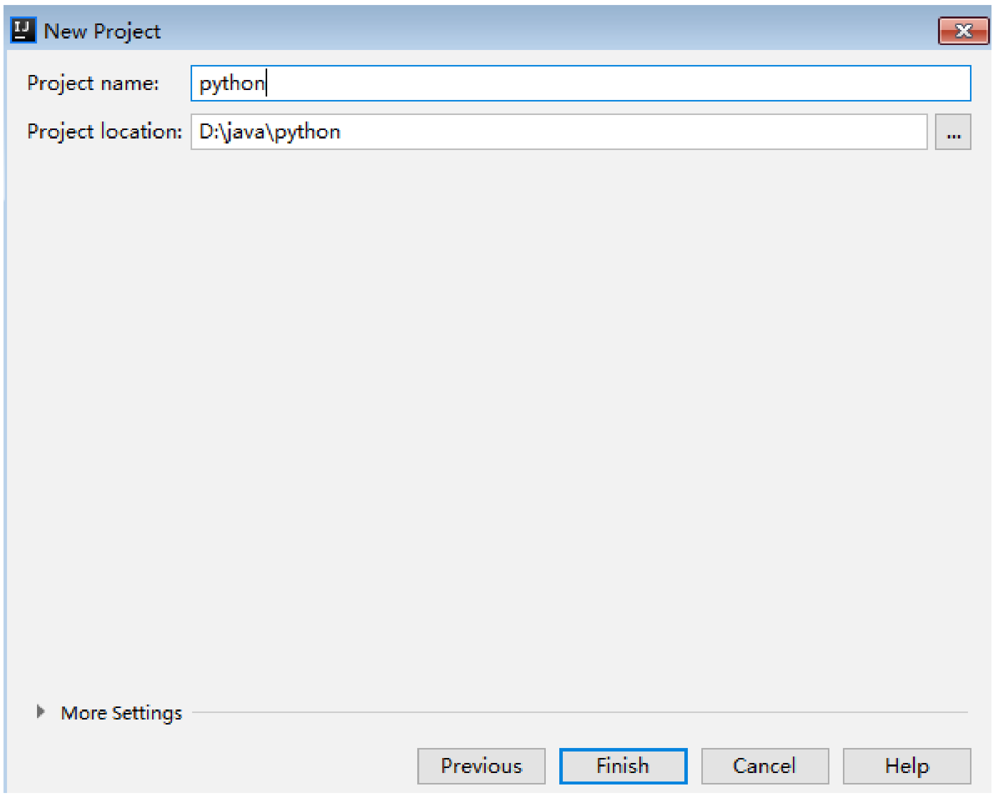
On the displayed **New Project** page, choose **Python** and click **Next**.

Figure 6-5 New Python



Step 2 Click **Next**. Click **...**, select the directory where the SDK is decompressed, and click **Finish**.

Figure 6-6 Selecting the SDK directory of Python after decompression

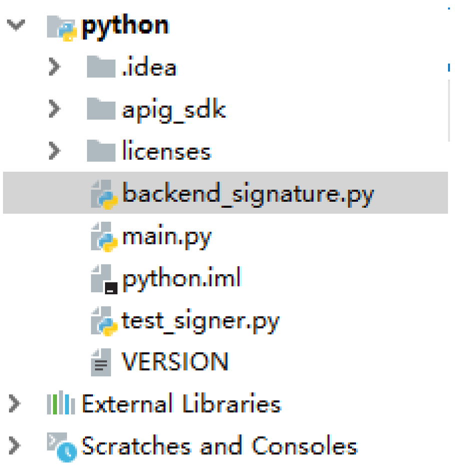

New Project

Project name: python

Project location: D:\java\python

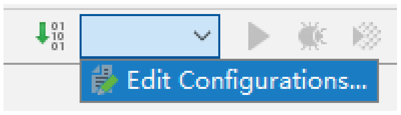
Step 3 View the directory structure shown in the following figure.

Figure 6-7 Directory structure


```
python
  .idea
  apig_sdk
  licenses
  backend_signature.py
  main.py
  python.iml
  test_signer.py
  VERSION
External Libraries
Scratches and Consoles
```

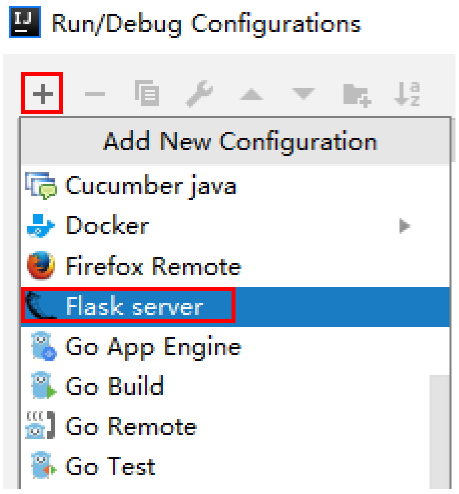
Step 4 Click **Edit Configurations**.

Figure 6-8 Edit Configurations

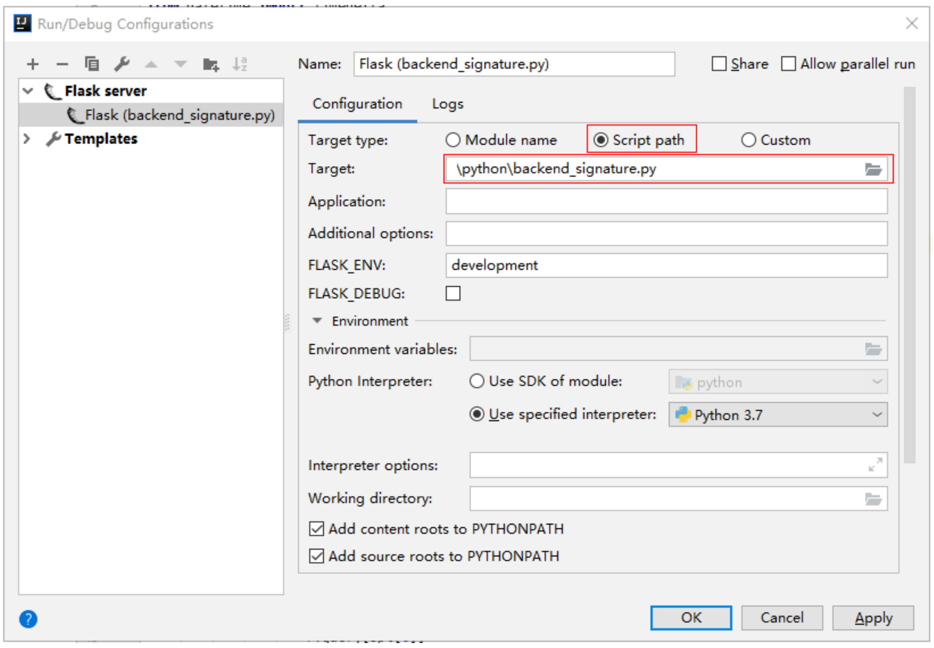


Step 5 Click **+** and choose **Flask server**.

Figure 6-9 Choosing Flask server



Step 6 Set **Target type** to **Script path**, select **backend_signature.py** from the **Target** drop-down list box, and click **OK**.



Name: Flask (backend_signature.py)
Target type: Module name [✓ Script path] Custom
Target: \python\backend_signature.py
Application:
Additional options:
FLASK_ENV: development
FLASK_DEBUG: ☐
Environment variables:
Python Interpreter: Use SDK of module: python
[✓ Use specified interpreter:] Python 3.7
Interpreter options:
Working directory:
☑ Add content roots to PYTHONPATH
☑ Add source roots to PYTHONPATH

----End

Backend Signature Verification Example

This example demonstrates how to build a Flask-based server as the backend of an API and implement a wrapper to verify the signature of requests sent from APIG (API Management).

NOTE

Signature information is added to requests sent to access the backend of an API only after a signature key is bound to the API.

Step 1 Compile an interface that returns **Hello World!** Configure the GET, POST, PUT, and DELETE methods and the **requires_apigateway_signature** wrapper.

```
app = Flask(__name__)

@app.route("/<id>", methods=['GET', 'POST', 'PUT', 'DELETE'])
@requires_apigateway_signature()
def hello(id):
    return "Hello World!"
```

Step 2 Implement **requires_apigateway_signature** by putting the signature key and secret in a **dict**.

```
def requires_apigateway_signature():
    def wrapper(f):

        secrets = {
            "signature_key1": "signature_secret1",
            "signature_key2": "signature_secret2",
        }
        authorizationPattern = re.compile(
            r'SDK-HMAC-SHA256\s+Access=([^,]+),\s?SignedHeaders=([^,]+),\s?Signature=(\w+)')
        BasicDateFormat = "%Y%m%dT%H%M%SZ"

        @wraps(f)
        def wrapped(*args, **kwargs):
            //Signature verification code
            ...

            return f(*args, **kwargs)
        return wrapped
    return wrapper
```

Step 3 Use a regular expression to parse the **Authorization** header. The key and signedHeaders are obtained. The wrapped function is used for signature verification.

```
if "authorization" not in request.headers:
    return 'Authorization not found.', 401
authorization = request.headers['authorization']
m = authorizationPattern.match(authorization)
if m is None:
    return 'Authorization format incorrect.', 401
signingKey = m.group(1)
signedHeaders = m.group(2).split(";")
```

For example, for **Authorization** header:

```
SDK-HMAC-SHA256 Access=signature_key1, SignedHeaders=host;x-sdk-date,
Signature=e11adf65a20d1b82c25419b5********8d0ba12fed1ceb13ed00
```

The parsing result is as follows:

```
signingKey=signature_key1
signedHeaders=host;x-sdk-date
```

Step 4 Find **secret** based on **key**. If **key** does not exist, the authentication failed.

```
if signingKey not in secrets:
    return 'Signing key not found.', 401
signingSecret = secrets[signingKey]
```

Step 5 Create an HttpRequest, and add the method, URL, query, and signedHeaders headers to the request. Determine whether the body needs to be set.

The body is read if there is no **x-sdk-content-sha256** header with value **UNSIGNED-PAYLOAD**.

```
r = signer.HttpRequest()
r.method = request.method
```

```
r.uri = request.path
r.query = {}
for k in request.query_string.decode('utf-8').split('&'):
    spl = k.split("=", 1)
    if len(spl) < 2:
        r.query[spl[0]] = ""
    else:
        r.query[spl[0]] = spl[1]
r.headers = {}
needbody = True
dateHeader = None
for k in signedHeaders:
    if k not in request.headers:
        return 'Signed header ' + k + ' not found', 401
    v = request.headers[k]
    if k.lower() == 'x-sdk-content-sha256' and v == 'UNSIGNED-PAYLOAD':
        needbody = False
    if k.lower() == 'x-sdk-date':
        dateHeader = v
    r.headers[k] = v
if needbody:
    r.body = request.get_data()
```

Step 6 Check whether the signature has expired. Obtain the time from the **X-Sdk-Date** header, and check whether the difference between this time and the server time is within 15 minutes. If **signedHeaders** does not contain **X-Sdk-Date**, the authentication failed.

```
if dateHeader is None:
    return 'Header x-sdk-date not found.', 401
t = datetime.strptime(dateHeader, BasicDateFormat)
if abs(t - datetime.utcnow()) > timedelta(minutes=15):
    return 'Signature expired.', 401
```

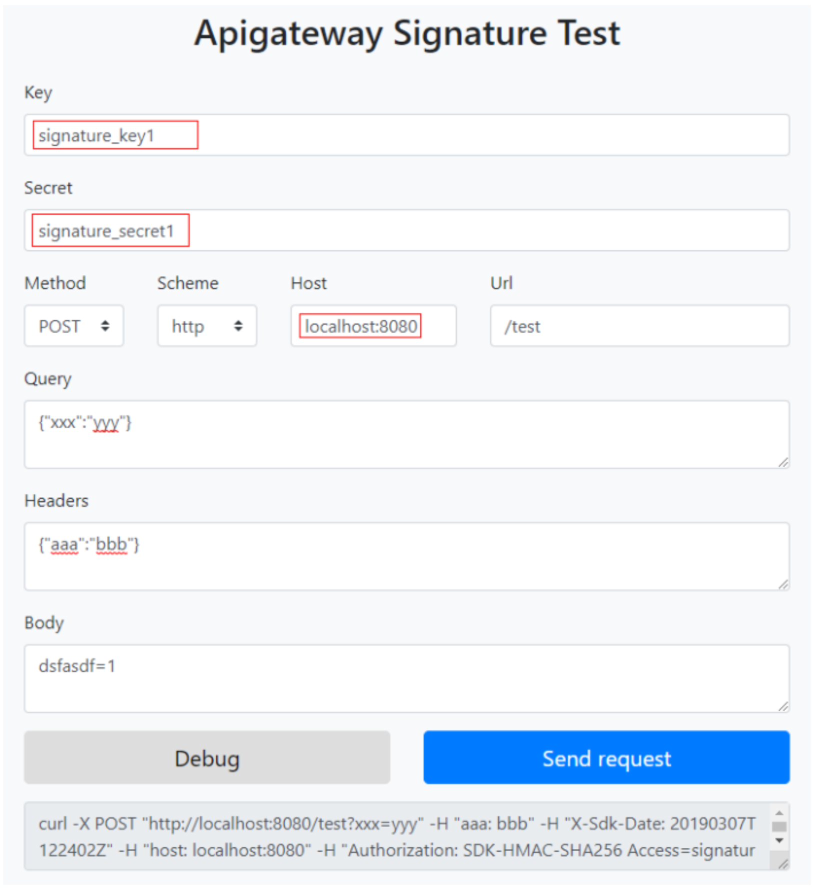
Step 7 Invoke the **verify** method to verify the signature of the request, and check whether the verification is successful.

```
sig = signer.Signer()
sig.Key = signingKey
sig.Secret = signingSecret
if not sig.Verify(r, m.group(3)):
    return 'Verify authroization failed.', 401
```

Step 8 Run the server to verify the code. The following example uses the **HTML signature tool** in the JavaScript SDK to generate a signature.

Set the parameters according to the following figure, and click **Send request**. Copy the generated curl command, execute it in the CLI, and check whether the server returns **200**.

If an incorrect key or secret is used, the server returns **401**, which means authentication failure.



Key: signature_key1

Secret: signature_secret1

Method	Scheme	Host	Url
POST	http	localhost:8080	/test

Query: {"xxx":"yyy"}

Headers: {"aaa":"bbb"}

Body: dsfasdf=1

```
curl -X POST "http://localhost:8080/test?xxx=yyy" -H "aaa: bbb" -H "X-Sdk-Date: 20190307T
122402Z" -H "host: localhost:8080" -H "Authorization: SDK-HMAC-SHA256 Access=signatur
```

----End

6.4 C#

Scenarios

To use C# to sign backend requests, obtain the C# SDK, import the project, and verify the backend signature by referring to the example provided in this section.

Preparing the Environment

- You have obtained the signature key and secret. For details, see **Pre-signature Preparations**.
- You have created a signature key on the APIG console and bound it to the API to be called. For more information, see section "Signature Keys" in the *API Gateway User Guide*.
- Log in to the APIG console, and download the SDK on the **SDKs** page by referring to section "SDKs" in the *API Gateway User Guide*.
- You have installed Visual Studio 2019 version 16.8.4 or a later version. If not, download it from the **official Visual Studio website** and install it.

Opening a Project

Double-click **csharp.sln** in the SDK package to open the project. The project contains the following:

- **apigateway-signature**: Shared library that implements the signature algorithm. It can be used in the .Net Framework and .Net Core projects.
- **backend-signature**: Example of a backend signature. Modify the parameters as required. For details about the sample code, see **Backend Signature Verification Example**.
- **sdk-request**: Example of invoking the signature algorithm.

Backend Signature Verification Example

This example demonstrates how to build an ASP.Net Core–based server as the backend of an API and implement an IAuthorizationFilter to verify the signature of requests sent from APIG (API Management).

NOTE

Signature information is added to requests sent to access the backend of an API only after a signature key is bound to the API.

Step 1 Write a controller that provides the GET, POST, PUT, and DELETE interfaces, and add the **ApigatewaySignatureFilter** attribute.

```
// ValuesController.cs

namespace backend_signature.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    [ApigatewaySignatureFilter]
    public class ValuesController : ControllerBase
    {
        // GET api/values
        [HttpGet]
        public ActionResult<IEnumerable<string>> Get()
        {
            return new string[] { "value1", "value2" };
        }

        // POST api/values
        [HttpPost]
        public void Post([FromBody] string value)
        {
        }

        // PUT api/values/5
        [HttpPut("{id}")]
        public void Put(int id, [FromBody] string value)
        {
        }

        // DELETE api/values/5
        [HttpDelete("{id}")]
        public void Delete(int id)
        {
        }
    }
}
```

Step 2 Implement **ApigatewaySignatureFilter** by putting the signature key and secret in a **Dictionary**.

```
// ApigatewaySignatureFilter.cs

namespace backend_signature.Filters
{
    public class ApigatewaySignatureFilter : Attribute, IAuthorizationFilter
    {
        private Dictionary<string, string> secrets = new Dictionary<string, string>
        {
            {"signature_key1", "signature_secret1" },
            {"signature_key2", "signature_secret2" },
        };

        public void OnAuthorization(AuthorizationFilterContext context) {
            //Signature verification code
            ...
        }
    }
}
```

Step 3 Use a regular expression to parse the **Authorization** header. The key and signedHeaders are obtained. The OnAuthorization function is used for signature verification.

```
private Regex authorizationPattern = new Regex("SDK-HMAC-SHA256\\s+Access=([^,]+),\\s?
SignedHeaders=([^,]+),\\s?Signature=(\\w+)");

...

string authorization = request.Headers["Authorization"];
if (authorization == null)
{
    context.Result = new UnauthorizedResult();
    return;
}
var matches = authorizationPattern.Matches(authorization);
if (matches.Count == 0)
{
    context.Result = new UnauthorizedResult();
    return;
}
var groups = matches[0].Groups;
string key = groups[1].Value;
string[] signedHeaders = groups[2].Value.Split(';');
```

For example, for **Authorization** header:

```
SDK-HMAC-SHA256 Access=signature_key1, SignedHeaders=host;x-sdk-date,
Signature=e11adf65a20d1b82c25419b5********8d0ba12fed1ceb13ed00
```

The parsing result is as follows:

```
signingKey=signature_key1
signedHeaders=host;x-sdk-date
```

Step 4 Find **secret** based on **key**. If **key** does not exist, the authentication failed.

```
if (!secrets.ContainsKey(key))
{
    context.Result = new UnauthorizedResult();
    return;
}
string secret = secrets[key];
```

Step 5 Create an HttpRequest, and add the method, URL, query, and signedHeaders headers to the request. Determine whether the body needs to be set.

The body is read if there is no **x-sdk-content-sha256** header with value **UNSIGNED-PAYLOAD**.

```
HttpRequest sdkRequest = new HttpRequest();
sdkRequest.method = request.Method;
sdkRequest.host = request.Host.Value;
sdkRequest.uri = request.Path;
Dictionary<string, string> query = new Dictionary<string, string>();
foreach (var pair in request.Query)
```

```
{
    query[pair.Key] = pair.Value;
}
sdkRequest.query = query;
WebHeaderCollection headers = new WebHeaderCollection();
string dateHeader = null;
bool needBody = true;
foreach (var h in signedHeaders)
{
    var value = request.Headers[h];
    headers[h] = value;
    if (h.ToLower() == "x-sdk-date")
    {
        dateHeader = value;
    }
    if (h.ToLower() == "x-sdk-content-sha256" && value == "UNSIGNED-PAYLOAD")
    {
        needBody = false;
    }
}
sdkRequest.headers = headers;
if (needBody)
{
    request.EnableRewind();
    using (MemoryStream ms = new MemoryStream())
    {
        request.Body.CopyTo(ms);
        sdkRequest.body = Encoding.UTF8.GetString(ms.ToArray());
    }
    request.Body.Position = 0;
}
```

Step 6 Check whether the signature has expired. Obtain the time from the **X-Sdk-Date** header, and check whether the difference between this time and the server time is within 15 minutes. If **signedHeaders** does not contain **X-Sdk-Date**, the authentication failed.

```
private const string BasicDateFormat = "yyyyMMddTHHmmssZ";

...

if(dateHeader == null)
{
    context.Result = new UnauthorizedResult();
    return;
}
DateTime t = DateTime.ParseExact(dateHeader, BasicDateFormat, CultureInfo.CurrentCulture);
if (Math.Abs((t - DateTime.Now).Minutes) > 15)
{
    context.Result = new UnauthorizedResult();
    return;
}
```

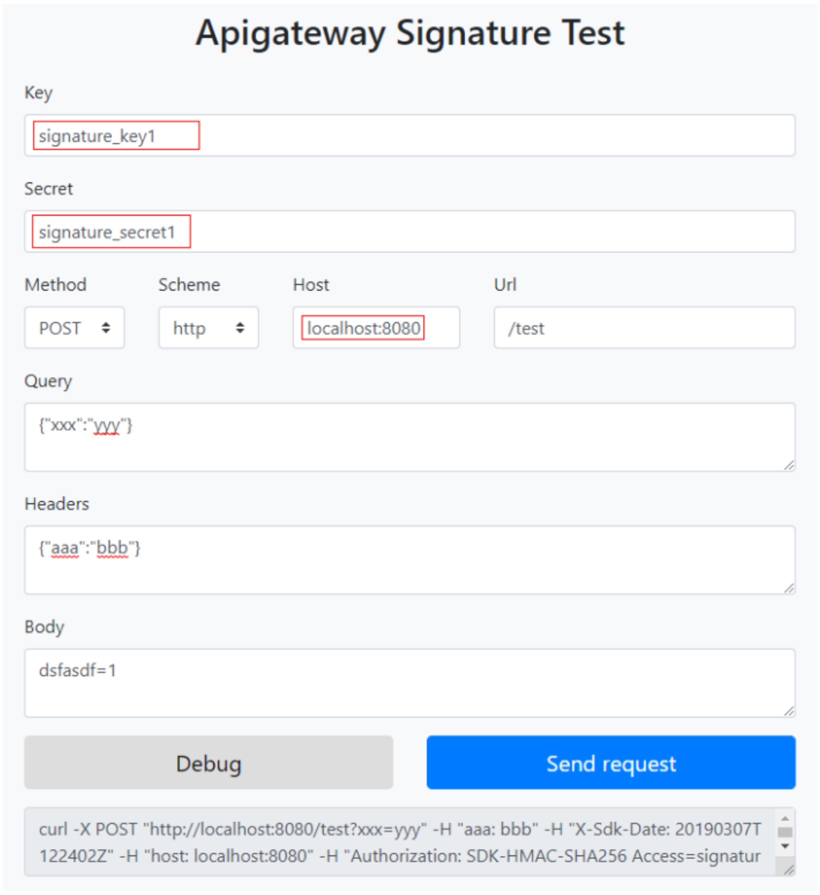
Step 7 Invoke the **verify** method to verify the signature of the request, and check whether the verification is successful.

```
Signer signer = new Signer();
signer.Key = key;
signer.Secret = secret;
if (!signer.Verify(sdkRequest, groups[3].Value))
{
    context.Result = new UnauthorizedResult();
}
```

Step 8 Run the server to verify the code. The following example uses the **HTML signature tool** in the JavaScript SDK to generate a signature.

Set the parameters according to the following figure, and click **Send request**. Copy the generated curl command, execute it in the CLI, and check whether the server returns **200**.

If an incorrect key or secret is used, the server returns **401**, which means authentication failure.



Key: signature_key1

Secret: signature_secret1

Method	Scheme	Host	Url
POST	http	localhost:8080	/test

Query: {"xxx":"yyy"}

Headers: {"aaa":"bbb"}

Body: dsfasdf=1

```
curl -X POST "http://localhost:8080/test?xxx=yyy" -H "aaa: bbb" -H "X-Sdk-Date: 20190307T
122402Z" -H "host: localhost:8080" -H "Authorization: SDK-HMAC-SHA256 Access=signatur
```

----End

A Change History

Table A-1 Change history

Date	Description
2023-12-19	This issue is the first official release.