# **Contributing**

We would ❤️ for you to contribute to Appwrite and help make it better! As a contributor, here are the guidelines we would like you to follow:

## **Code of Conduct**

Help us keep Appwrite open and inclusive. Please read and follow our [Code of Conduct](https://github.com/appwrite/sdk-generator/blob/master/CODE_OF_CONDUCT.md).

## **Installation**

To install a working development environment, please follow these instructions:

1. Fork or clone the appwrite/sdk-generator repository.
2. Install Composer dependencies using one of the following options:

Composer CLI

composer update --ignore-platform-reqs --optimize-autoloader --no-plugins --no-scripts --prefer-dist

Docker (UNIX)

docker run --rm --interactive --tty --volume "$(pwd)":/app composer update --ignore-platform-reqs --optimize-autoloader --no-plugins --no-scripts --prefer-dist

Docker (Windows)

docker run --rm --interactive --tty --volume "%cd%":/app composer update --ignore-platform-reqs --optimize-autoloader --no-plugins --no-scripts --prefer-dist

1. Follow our contribution guide to learn how you can add support for more languages.

## **Creating a Language Class**

First, create a new class for the new language in this directory: [/src/SDK/Language](https://github.com/appwrite/sdk-generator/tree/master/src/SDK/Language)

You can use the interface to know which methods are required to be implemented: [/src/SDK/Language.php](https://github.com/appwrite/sdk-generator/blob/master/src/SDK/Language.php)

getName SDK Language name (JS, PHP…).

getKeywords An array with language keywords to avoid using as param or function names, template engine will solve conflicts.

getIdentifierOverrides Returns an associative array that can be used to override keywords with pre-defined word using overrideIdentifier filter.

getFiles An array with a list of language template files in [twig format](https://twig.symfony.com/). Each file scope determines what template parameters will be available.

* Default scope - Basic SDK and language-specific params (package name, language name, etc.).
* Service scope - Generate x templates where x is the number of API services, adds service-specific params to the template (service name, methods, etc.).
* Method scope - Generate x\*y templates where x is the number of API services and y is the number of methods, adds service and method-specific params to the template (service name, method name, method params, etc.), which is good for generating MD files with examples for using each method.
* Copy scope - Static files (like images, etc.) that will just get copied and not processed by twig.

getTypeName This method receives the API param type and should return the equivalent param in the implemented language.

getParamDefault This method receives the API param and should return the equivalent default value of param in the implemented language, for example, a default array param in PHP is represented as [].

getParamExample This method receives the API param and should return the equivalent example value of param in the implemented language, for example, if an example value is some text in PHP, the return value should be 'some text' (with quotes).

Note: The easiest way to get started is to copy an existing language class close to the new language about to be implemented and just edit it.

## **Adding Templates**

Add your new templates as listed in your language class getFiles method. Make sure to follow the [checklist](https://github.com/appwrite/sdk-generator/blob/master/CONTRIBUTING.md#sdk-checklist) when building the language templates.

Make sure to follow the objects structure and service separation architecture. We aim to keep the developer experience as consistent as possible across different SDKs to make the learning curve as short as possible.

The Appwrite SDK generator adds some filters to the TWIG templates to allow common code formatting options like converting text to camelCase, dash-case, and others. The full list is available in the [SDK class](https://github.com/appwrite/sdk-generator/blob/master/src/SDK/SDK.php#L62)

When you need to test the API templates output, add your new language instance to the example.php file like this:

sdk-generator/blob/master/example.php:

 use Appwrite\Spec\Swagger2;

 use Appwrite\SDK\SDK;

 use Appwrite\SDK\Language\NewLang;

 function getSSLPage($url) {

 $ch = curl\_init();

 curl\_setopt($ch, CURLOPT\_HEADER, false);

 curl\_setopt($ch, CURLOPT\_URL, $url);

 curl\_setopt($ch, CURLOPT\_SSL\_VERIFYPEER, false);

 curl\_setopt($ch, CURLOPT\_SSL\_VERIFYHOST, false);

 curl\_setopt($ch, CURLOPT\_RETURNTRANSFER, true);

 $result = curl\_exec($ch);

 curl\_close($ch);

 return $result;

 }

 $spec = getSSLPage('https://appwrite.io/v1/open-api-2.json?extensions=1');

 // NewLang

 $sdk = new SDK(new NewLang(), new Swagger2($spec));

 $sdk

 ->setLogo('https://appwrite.io/v1/images/console.png')

 ->setLicenseContent('test test test')

 ->setWarning('\*\*WORK IN PROGRESS - NOT READY FOR USAGE\*\*')

 ;

 $sdk->generate(\_\_DIR\_\_ . '/examples/new-lang');

Run the following command (make sure you have an updated docker version on your machine):

docker run --rm -v $(pwd):/app -w /app php:7.4-cli php example.php

Note: You can just add the new language next to the other languages in example.php file; there is no need to rewrite the file completely.

Check your output files at: /examples/new-lang and make sure the SDK works. When possible, add some unit tests.

## **SDK Checklist**

It is very important for us to create a consistent structure and architecture as well as a native-like feel for the SDKs we generate. To accomplish that, we have made a checklist of points to support while adding a new language to the SDK generator.

The following checklist aims to balance consistency among languages and follow each platform's best practices and coding standards.

* Proper Coding Standards and Conventions
* Proper Skeleton Structure
* Readme Doc
* HTTP Client class or object
	+ Client Setters
		- Set Auth Keys Method
		- Set Basic Auth Method
		- Set OAuth Dialog Method
		- Set Endpoint Method
		- Set Self Signed Certificates
	+ Default Headers
		- 'appwrite-sdk-version' header
		- Add 'User-Agent' header with device/server name and version + platform name and version (ubuntu-20.04-php-7.0.1 / android-20.0-flutter-3.0).
		- Add 'origin' header with the following syntax <scheme>://<identifier> where scheme is one of http, https, appwrite-android, appwrite-ios, appwrite-macos, appwrite-windows, appwrite-linux and identifier is the host name is case of web apps and the package name in case of ios, android and other platforms.
		- All Global headers available in spec
	+ Methods
		- addHeader(key, value)
		- call(method, path = '', headers = [], params = [])
			* Concat GET params to path
			* Parse request params by content type header
			* Parse response params by content type header
			* Throw error on bad response
	+ Handle errors and throw AppwriteException with proper information
* Service Abstraction (optional)
	+ Constructor receiving an instance of the client class
* Service Class (extends the service abstraction if exists)
	+ Headers Support (Content Type)
	+ Parameters Support
		- Required Values Support
		- String Support
		- Integer Support
		- Boolean Support
		- Files Support (+array file and multiple header support and params flatten)
		- Arrays / List Support
		- Key-Value / Maps Support
* Usage Example Docs
* Definitions / Models Classes - with setters and getters

## **Tests**

Testing a single project that runs in multiple languages can be difficult. Managing dependencies with multiple package managers of different ecosystems can take the SDK Generator complexity to extreme levels.

To avoid that complexity, we have created a cross-platform mechanism that leverages Docker and a vanilla language file with no dependencies attached.

To add a new language test, you need to create a language file that initializes your SDK and call generated method in a predefined order. The test algorithm will evaluate your script output and determine whether the test passed or failed.

The test algorithm will generate your SDK from a small demo SDK JSON spec file and will run your test on different versions of your chosen language.

To get started, create a language file in this location:

./tests/languages/<language>/test.[MY-LANGUAGE-FILE-EXT]

In your new language file, init your SDK from a relative path which will be generated here: ./tests/sdks/ from this spec file: ./tests/resources/spec.json.

After you finish initializing, make a series of HTTP calls using your new generated SDKs method just like in one of these examples:

1. tests/languages/php/test.php
2. tests/languages/node/test.js

Note: In your test files, make sure that you begin the test with the following string "\nTest Started\n". We use this string to filter output from the build tool you're using.

Once you’re done, create a new test file tests/[Language]Test.php and update as the following.

<?php

namespace Tests;

class [Language]Test extends Base

{

 protected string $language = '[language]';

 protected string $class = 'Appwrite\SDK\Language\[Language]';

 protected array $build = [

 //commands required before executing the test

 ];

 protected array $envs = [

 // docker commands that can execute test file to the sdk test. Make sure to add

 // one command for each lanuage version you wish to support

 ];

 // list of expected outputs from test based on features supported

 protected array $expectedOutput = [

 ...Base::FOO\_RESPONSES,

 ...Base::BAR\_RESPONSES,

 ...Base::GENERAL\_RESPONSES,

 ...Base::EXCEPTION\_RESPONSES,

 ...Base::REALTIME\_RESPONSES

 ];

}

A good example is the Dart test:

<?php

namespace Tests;

class DartTest extends Base

{

 protected string $language = 'dart';

 protected string $class = 'Appwrite\SDK\Language\Dart';

 protected array $build = [

 'mkdir -p tests/sdks/dart/tests',

 'cp tests/languages/dart/tests.dart tests/sdks/dart/tests/tests.dart',

 ];

 protected array $envs = [

 'dart-stable' => 'docker run --rm -v $(pwd):/app -w /app/tests/sdks/dart dart:stable sh -c "dart pub get && dart pub run tests/tests.dart"',

 'dart-beta' => 'docker run --rm -v $(pwd):/app -w /app/tests/sdks/dart dart:beta sh -c "dart pub get && dart pub run tests/tests.dart"',

 ];

 protected array $expectedOutput = [

 ...Base::FOO\_RESPONSES,

 ...Base::BAR\_RESPONSES,

 ...Base::GENERAL\_RESPONSES,

 ...Base::EXCEPTION\_RESPONSES,

 ];

}

Also in .travis.yml add new env SDK=[Language] so that travis will run a test for this language as well.

Finally, you can run tests using

docker run --rm -v $(pwd):$(pwd):rw -w $(pwd) -v /var/run/docker.sock:/var/run/docker.sock php:7.4-cli-alpine sh -c "apk add docker-cli && vendor/bin/phpunit".

## **SDK Generator Interface**

* spec -- This object is derived from the appwrite swagger spec
	+ title -> The title of the SDK you are generating (normally used as package name.)
	+ description -> Description of Appwrite SDK
	+ namespace -> SDK Namespace
	+ version -> SDK Version
	+ endpoint -> Default Endpoint (example: "<https://appwrite.io/v1>")
	+ host -> Default Host (example: "appwrite.io")
	+ basePath -> Default Path to API (example: "/v1")
	+ licenseName -> Name of licence for SDK
	+ licenseURL -> URL to SDK Licence
	+ contactName -> Name of Person/Team that created the SDK
	+ contactURL -> URL to contact for help with the SDK
	+ contactEmail -> Email Address to Contact for help with the SDK
	+ services -> Array of Services. Each service contains the following:
		- name -> The name of the service
		- methods -> Array of methods that can be used with the service
			* method -> HTTP Method to call
			* path -> Path to API without a basePath
			* fullPath -> Path to API with basePath
			* name -> Name of API Method
			* packaging -> A flag to indicate if the files at a path need to be packaged as a tar file
			* title -> Title of API Method
			* description -> Description of API Method
			* security -> Array of security methods for this API Call, primarily used for code examples.
			* consumes -> Array of Content-Type headers the API Route accepts.
			* cookies -> Are cookies required? Bool
			* type -> Response Type. Tells us whether the endpoint returns a JSON Payload, A URL or redirect to an auth mechanism.
			* headers -> Array of headers for API
			* parameters -> Parameters for API
				+ all -> Array containing all Parameters
				+ headers -> Array containing parameters that go in the header
				+ path -> Array containing parameters that go into the path of the API URL
				+ query -> Array containing parameters that go into the query of the API URL
				+ body -> Array containing parameters that go in the body
				All Parameters will have a structure like this:

name -> Name of parameter

type -> Parameter Type

description -> Parameter Description

required -> Is parameter required

default -> Parameter Defaults

example -> Parameter Example

array

type -> Array Type (only used if param type is "array")

* + global
		- headers -> An object containing all global headers
		- defaultHeaders -> An object containing all default headers
* language -- Information on the current language SDK
	+ name -> Name of language
	+ params -> Custom language specific parameters
* sdk -- Various Metadata used for packaging and categorizing
	+ namespace -> SDK Namespace
	+ name -> SDK Name
	+ description -> SDK Desc
	+ shortDescription -> SDK Short Desc
	+ version -> SDK Version
	+ license -> SDK License
	+ licenseContent -> SDK License content
	+ gitURL -> GIT URL for SDK
	+ gitRepo -> GIT Repo for SDK
	+ gitRepoName -> Git Repo Name
	+ gitUserName -> Git username of creator
	+ logo -> SDK Logo
	+ url -> SDK URL
	+ shareText -> Social Media Metadata
	+ shareURL -> Social Media Metadata
	+ shareVia -> Social Media Metadata
	+ shareTags -> Social Media Metadata
	+ warning -> Used for warnings usually communicated within the Readme.md
	+ gettingStarted -> Raw markdown for getting started
	+ readme -> Stores the raw markdown used to generate the readme.md file. [here](https://github.com/appwrite/sdk-for-flutter/blob/master/README.md)
	+ changelog -> Stores the raw markdown used to generate the changelog.md file. [here](https://github.com/appwrite/sdk-for-flutter/blob/master/CHANGELOG.md)
	+ examples -> Stores the raw markdown used to generate examples for your SDK. An example can be found [here](https://github.com/appwrite/sdk-for-flutter/tree/master/example)
	+ twitterHandle -> Twitter handle of creator
	+ discordChannel -> Discord Channel ID for SDK
	+ discordUrl -> Discord Server Invite for SDK