

# **Serverless Application Center**

## **Framework Support**

## **Product Documentation**



## Copyright Notice

©2013-2024 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

## Trademark Notice



All trademarks associated with Tencent Cloud and its services are owned by Tencent Cloud Computing (Beijing) Company Limited and its affiliated companies. Trademarks of third parties referred to in this document are owned by their respective proprietors.

## Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

# Contents

## Framework Support

Deploying Framework on Command Line

Quickly Deploying Egg Framework

Quickly Deploying Koa Framework

Quickly Deploying Express Framework

Quickly Deploying Next.js Framework

Quickly Deploying Nuxt.js Framework

Quickly Deploying Flask Framework

Quickly Deploying Laravel Framework

Quickly Deploying Nest.js Framework

Quickly Deploying Django Framework

Quickly Deploying Native WordPress Application

# Framework Support

## Deploying Framework on Command Line

Last updated : 2021-08-20 16:21:26

In addition to the console, you can also quickly deploy a web framework on the command line. This document describes how to use the [HTTP component](#) of Serverless Framework to complete the local deployment of web applications.

### Prerequisites

You have activated the service and completed the [permission configuration](#) for Serverless Framework.

### Supported frameworks

Supported Framework	Document
Express	<a href="#">Quickly Deploying Express Framework</a>
Koa	<a href="#">Quickly Deploying Koa Framework</a>
Egg	<a href="#">Quickly Deploying Egg Framework</a>
Next.js	<a href="#">Quickly Deploying Next.js Framework</a>
Nuxt.js	<a href="#">Quickly Deploying Nuxt.js Framework</a>
Nest.js	<a href="#">Quickly Deploying Nest.js Framework</a>
Flask	<a href="#">Quickly Deploying Flask Framework</a>
Django	<a href="#">Quickly Deploying Django Framework</a>
Laravel	<a href="#">Quickly Deploying Laravel Framework</a>

### Directions

#### 1. Develop an application locally

Complete the development locally according to your actual business scenario. For more information, please see the documents in the [Supported frameworks](#) section.

#### 2. Configure the .yaml file

Create a `serverless.yaml` file in the project root directory and write the configuration by referring to the

following sample. For the full configuration, please see [Configuration Document](#).

```
# serverless.yml
component: http # Component name, which is required
name: webDemo # Component instance name, which is required

inputs:
region: ap-guangzhou # Function region
src: # Deploy the file code under `src`, package it into a zip file, and upload it to the bucket
src: ./ # The file directory that needs to be packaged locally
exclude: # Excluded files or directories
- .env
- 'node_modules/**'
faas: # Function configuration
framework: express # Select the framework. Express is used here as an example
runtime: Nodejs12.16
name: webDemo # Function name
timeout: 10 # Timeout period in seconds
memorySize: 512 # Memory size, which is 512 MB by default
layers:
- name: layerName # Layer name
version: 1 # Version

apigw: # # The HTTP component will create an API Gateway service by default
isDisabled: false # Specify whether to disable automatic API Gateway creation
id: service-xxx # API Gateway service ID. If you leave it empty, a gateway will be automatically created
name: serverless # API Gateway service ID
api: # Relevant configuration of the created API
cors: true # Specify whether to allow CORS
timeout: 15 # API timeout period
name: apiName # API name
qualifier: $DEFAULT # Version associated with the API
protocols:
- http
- https
environment: test
```

3. After the creation is completed, run `sls deploy` in the root directory to deploy. The component will automatically generate the `scf_bootstrap` bootstrap file for deployment according to the selected framework.

Note :

As the bootstrap file logic is strongly related to your business logic, the generated default bootstrap file may cause the framework start to fail. We recommend you manually configure the bootstrap file according to your actual business needs. For more information, please see the deployment guide document of each framework.

### Sample `scf_bootstrap` :

- express:

```
#!/usr/bin/env bash
/var/lang/node12/bin/node app.js
```

- koa

```
#!/usr/bin/env bash
/var/lang/node12/bin/node app.js
```

- egg

```
#!/var/lang/node12/bin/node

/**
 * Node path in docker: /var/lang/node12/bin/node
 * As only `/tmp` is readable/writable in SCF, two environment variables need to
 * be modified at startup
 * `NODE_LOG_DIR` changes the default node write path of `egg-scripts` (`~/logs`)
 * to `/tmp`
 * `EGG_APP_CONFIG` changes the default directory of the Egg application to `/t
 * mp`
 */

process.env.EGG_SERVER_ENV = 'prod';
process.env.NODE_ENV = 'production';
process.env.NODE_LOG_DIR = '/tmp';
process.env.EGG_APP_CONFIG = '{"rundir":"/tmp","logger":{"dir":"/tmp"}}';

const { Application } = require('egg');

// If you deploy `node_modules` through layers, you need to modify `eggPath`
Object.defineProperty(Application.prototype, Symbol.for('egg#eggPath'), {
  value: '/opt',
});
```

```
const app = new Application({
  mode: 'single',
  env: 'prod',
});

app.listen(9000, '0.0.0.0', () => {
  console.log('Server start on http://0.0.0.0:9000');
});
```

- nextjs

```
#!/var/lang/node12/bin/node

/*
# As the HTTP passthrough function runs based on the docker image, the listeni
ng address must be 0.0.0.0, and the port 9000
*/
const { nextStart } = require('next/dist/cli/next-start');
nextStart(['--port', '9000', '--hostname', '0.0.0.0']);
```

- nuxtjs

```
#!/var/lang/node12/bin/node

/*
# As the HTTP passthrough function runs based on the docker image, the listeni
ng address must be 0.0.0.0, and the port 9000
*/
require('@nuxt/cli')
  .run(['start', '--port', '9000', '--hostname', '0.0.0.0'])
  .catch((error) => {
    require('consola').fatal(error);
    require('exit')(2);
  });
```

- nestjs

```
#!/bin/bash
# SERVERLESS=1 /var/lang/node12/bin/npm run start -- -e /var/lang/node12/bin/no
de
SERVERLESS=1 /var/lang/node12/bin/node ./dist/main.js
```

- flask

```
#!/bin/bash
# As the HTTP passthrough function runs based on the docker image, the listenin
g address must be 0.0.0.0, and the port 9000
/var/lang/python3/bin/python3 app.py
```

- django

```
#!/bin/bash
# As the HTTP passthrough function runs based on the docker image, the listeni
ng address must be 0.0.0.0, and the port 9000
/var/lang/python3/bin/python3 manage.py runserver 0.0.0.0:9000
```

- laravel

```
#!/bin/bash

#####
# Inject environment variables in the serverless environment
#####
# Inject the SERVERLESS flag
export SERVERLESS=1
# Modify the template compilation cache path, as only `/tmp` is readable/writa
ble in SCF
export VIEW_COMPILED_PATH=/tmp/storage/framework/views
# Modify `session` to store it in the memory (array type)
export SESSION_DRIVER=array
# Output logs to `stderr`
export LOG_CHANNEL=stderr
# Modify the application storage path
export APP_STORAGE=/tmp/storage

# Initialize the template cache directory
mkdir -p /tmp/storage/framework/views

# As the HTTP passthrough function runs based on the docker image, the listeni
ng address must be 0.0.0.0, and the port 9000
# Path of the executable file in the cloud: /var/lang/php7/bin/php
/var/lang/php7/bin/php artisan serve --host 0.0.0.0 --port 9000
```



# Quickly Deploying Egg Framework

Last updated : 2022-07-22 15:10:58

The SLS framework deployment scheme has been upgraded. You can use an SCF HTTP-triggered function to quickly deploy your Egg service to the cloud.

Note :

## What are the differences between SLS console deployment and direct function deployment?

Both SLS console deployment and function deployment can be based on HTTP-triggered functions, and quick deployment is usually used for web frameworks.

- If you only need to develop code logic and do not need to create additional resources, you can perform quick deployment through the Serverless console.
- If you need to create more capabilities or resources, such as automatic creation of layer hosting dependencies, quick implementation of static resource isolation, and support for direct code repository pulling, in addition to code deployment, you can use the SLS console to create web applications.

This document introduces the SLS console deployment scheme. You can also complete the deployment in CLI by referring to [Deploying Web Function on Command Line](#).

## Template Deployment - Deploying Egg Sample Code

1. Log in to the [SLS console](#).
2. Choose **Create Application** and select **Web Application > Egg Framework**.
3. Click **Next** and complete basic configuration.
4. Select **Sample Code** as the upload mode and click **Complete**. The application deployment starts.
5. After the application deployment is completed, you can view the basic information of the sample application on the application details page. In addition, you can use access the deployed Egg project at the access path URL generated by API Gateway.

## Custom Deployment - Quickly Deploying Web Application

### Prerequisites

The Node.js runtime environment has been installed locally.

## Local development

1. Refer to [Quick Start](#) to quickly initialize the sample project as follows:

```
mkdir egg-example && cd egg-example
npm init egg --type=simple
npm i
```

2. In the root directory, run the following command to directly start the service locally.

```
npm run dev
open http://localhost:7001
```

3. Start a browser, and you can access the sample Egg project locally.

## Deployment in cloud

Next, perform the following steps to make simple modifications to the locally created project, so that it can be quickly deployed through an HTTP-triggered function. The steps of project transformation for the Egg framework are as follows:

- Change the listening address and port to `0.0.0.0:9000`.
- Modify the write path. Only the `/tmp` directory is readable/writable in the SCF environment.
- Add the `scf_bootstrap` file.

### 1. (Optional) Configure the `scf_bootstrap` file.

Note :

You can also complete the configuration in the console.

Create the `scf_bootstrap` file in the root directory of the project and add the following content to it (the file is used to configure environment variables and start the service. Here is only a sample. Please adjust the configuration according to your actual business scenario):

```
#!/var/lang/node12/bin/node
'use strict';
/**
 * Node path in docker: /var/lang/node12/bin/node
 * As only `/tmp` is readable/writable in SCF, two environment variables need to be modified at startup
```

```
* `NODE_LOG_DIR` changes the default node write path of `egg-scripts` (~/.logs) to
`/tmp`
* `EGG_APP_CONFIG` changes the default directory of the Egg application to `/tmp`
*/
process.env.EGG_SERVER_ENV = 'prod';
process.env.NODE_ENV = 'production';
process.env.NODE_LOG_DIR = '/tmp';
process.env.EGG_APP_CONFIG = '{"rundir":"/tmp","logger":{"dir":"/tmp"}}';
const { Application } = require('egg');
// If you deploy `node_modules` through layers, you need to modify `eggPath`
Object.defineProperty(Application.prototype, Symbol.for('egg#eggPath'), {
  value: '/opt',
});
const app = new Application({
  mode: 'single',
  env: 'prod',
});
app.listen(9000, '0.0.0.0', () => {
  console.log('Server start on http://0.0.0.0:9000');
});
```

After the file is created, you need to run the following command to modify the executable permission of the file. By default, the permission `777` or `755` is required for the service to start normally. Below is the sample code:

```
chmod 777 scf_bootstrap
```

## 2. Console upload

Log in to the [SLS console](#), select **Web Application > Egg Framework**, and select **Local Upload** or **Code Repository Pull** as the upload mode.

You can configure the `scf_bootstrap` file in the console. When the configuration is completed, the console automatically generates the `scf_bootstrap` file and packages it and the project code for deployment.

Note :

The actual `scf_bootstrap` file in your project prevails. If the `scf_bootstrap` file already exists in your project, its content will not be overwritten.

When the configuration is completed, click **Complete** to deploy your Egg project.

## Advanced configuration management

In **Advanced Configuration**, you can perform more application management operations, such as creating layers, binding custom domains, and configuring environment variables.

# Quickly Deploying Koa Framework

Last updated : 2022-07-22 15:09:26

The SLS framework deployment scheme has been upgraded. You can use an SCF HTTP-triggered function to quickly deploy your Koa service to the cloud.

Note :

## What are the differences between SLS console deployment and direct function deployment?

Both SLS console deployment and function deployment can be based on HTTP-triggered functions, and quick deployment is usually used for web frameworks.

- If you only need to develop code logic and do not need to create additional resources, you can perform quick deployment through the Serverless console.
- If you need to create more capabilities or resources, such as automatic creation of layer hosting dependencies, quick implementation of static resource isolation, and support for direct code repository pulling, in addition to code deployment, you can use the SLS console to create web applications.

## Prerequisites

- Before using Tencent Cloud Serverless, sign up for a [Tencent Cloud account](#) and complete [identity verification](#).

Note :

This document introduces the SLS console deployment scheme. You can also complete the deployment in CLI by referring to [Deploying Web Function on Command Line](#).

## Directions

### Template deployment - deploying Koa sample code

1. Log in to the [SLS console](#).
2. Choose **Create Application** and select **Web Application > Koa Framework**.
3. Click **Next** and complete basic configuration.
4. Select **Sample Code** as the upload mode and click **Complete**. The application deployment starts.

5. After the application deployment is completed, you can view the basic information of the sample application on the application details page. In addition, you can use access the deployed Koa project at the access path URL generated by API Gateway.

## Custom deployment - quickly deploying web application

### Prerequisites

The Node.js runtime environment has been installed locally.

### Local development

1. Refer to the [Koa.js](#) official documentation to install the Koa environment and initialize your Koa project. The following takes `hello world` as an example. The content of `app.js` is as follows:

```
// app.js
const Koa = require('koa');
const app = new Koa();
const main = ctx => {
  ctx.response.body = 'Hello World';
};
app.use(main);
app.listen(3000);
```

2. In the root directory, run the following command to directly start the service locally.

```
node app.js
```

3. Visit `http://localhost:3000` in a browser, and you can access the sample Koa project locally.

### Deployment in cloud

You need to make simple modifications to the initialized project, so that the project can be quickly deployed through an HTTP-triggered function. The project transformation here is usually divided into the following two steps:

- Change the listening address and port to `0.0.0.0:9000`.
- Add the `scf_bootstrap` file.

The detailed steps are as follows:

1. In the sample Koa project, change the listening port to `9000` .

```
1  const Koa = require('koa');
2  const app = new Koa();
3
4  app.use(async ctx => {
5    |   ctx.body = 'Hello World';
6  });
7
8  app.listen(9000);
9
```

2. Create the `scf_bootstrap` file in the root directory of the project and add the following content to it (the file is used to configure environment variables and start the service):

Note :

You can also complete the configuration in the console.

```
#!/bin/bash
/var/lang/node12/bin/node app.js
```

After the file is created, you need to run the following command to modify the executable permission of the file. By default, the permission `777` or `755` is required for the service to start normally. Below is the sample code:

```
chmod 777 scf_bootstrap
```

3. After the local configuration is completed, run the bootstrap file and make sure that your service can be normally started locally. Then log in to the [SLS console](#), select **Web Application > Koa Framework**, and select **Local Upload** or **Code Repository Pull** as the upload mode.

You can configure the `scf_bootstrap` file in the console. When the configuration is completed, the console automatically generates the `scf_bootstrap` file and packages it and the project code for deployment.

Note :

The actual `scf_bootstrap` file in your project prevails. If the `scf_bootstrap` file already exists in your project, its content will not be overwritten.

When the configuration is completed, click **Complete** to deploy your Koa project.



# Quickly Deploying Express Framework

Last updated : 2023-05-05 14:59:02

The SLS framework deployment scheme has been upgraded. You can use an SCF HTTP-triggered function to quickly deploy your Express service to the cloud.

## Note

### What are the differences between SLS console deployment and direct function deployment?

Both SLS console deployment and function deployment can be based on HTTP-triggered functions, and quick deployment is usually used for web frameworks.

If you only need to develop code logic and do not need to create additional resources, you can perform quick deployment through the SLS console.

If you need to create more capabilities or resources, such as automatic creation of layer hosting dependencies, quick implementation of static resource isolation, and support for direct code repository pulling, in addition to code deployment, you can use the SLS console to create web applications.

This document describes the SLS console deployment scheme. You can also complete the deployment in CLI by referring to [Deploying Web Function on Command Line](#).

## Template Deployment: Deploying Express Sample Code

1. Log in to the [SLS console](#).
2. Click **Create Application** and choose **Web Application > Express Framework**.
3. Click **Next** and complete the basic configuration. Select **Sample Code** as the upload mode.
4. Click **Complete**.
5. After the application is deployed, you can click the sample application name on the **Application List** page to go to the details page of the application.
6. On the **Resource List** page, click the access URL generated by the API gateway of the sample application to access the deployed Express project.

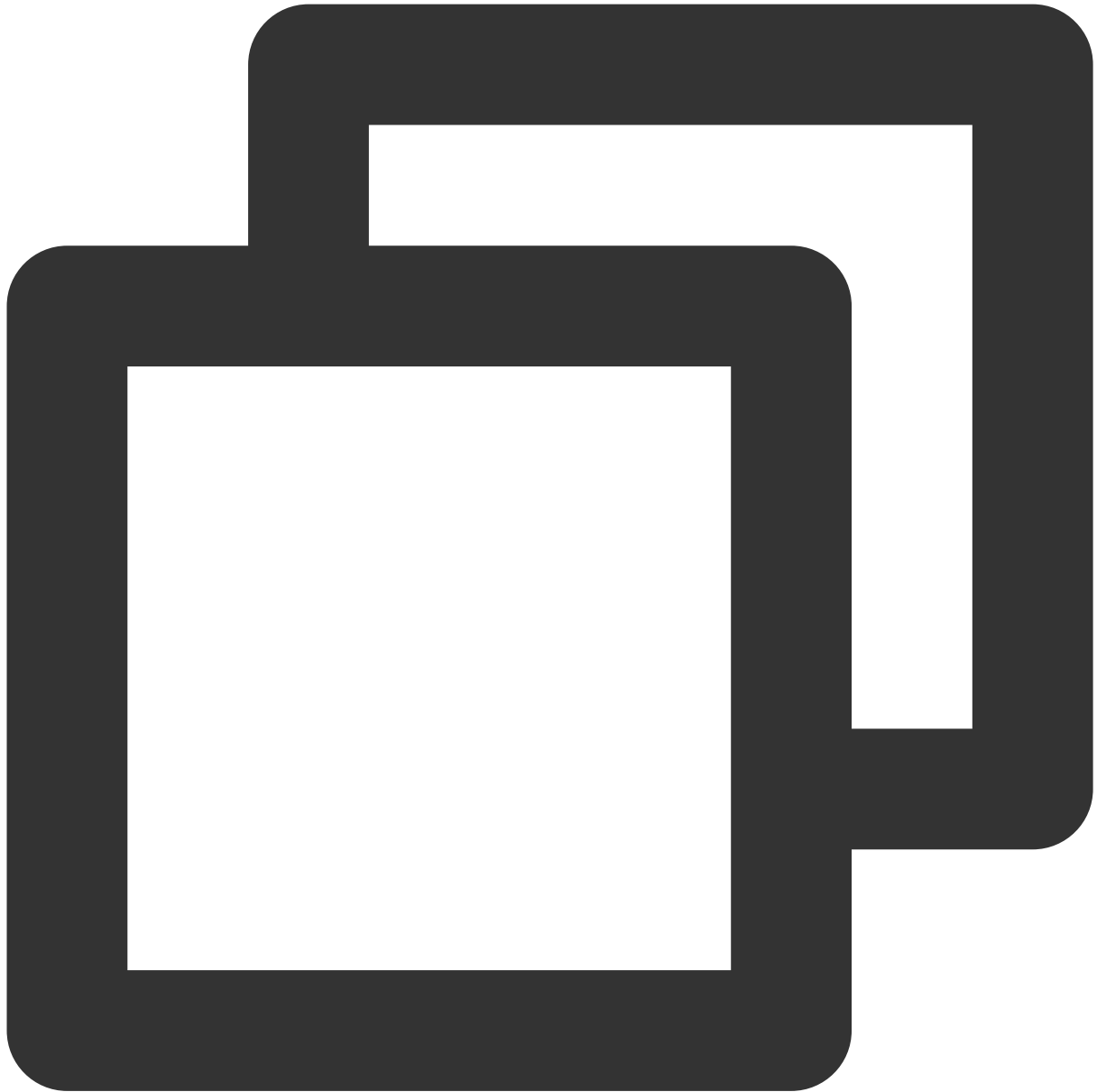
## Custom Deployment - Quickly Deploying Web Application

### Prerequisites

The Node.js runtime environment has been installed locally.

### Local development

1. After confirming that the Node.js runtime environment has been installed locally, install the Express framework and express-generator tool and initialize your sample Express project.



```
npm install express --save
npm install express-generator --save
express WebApp
```

2. Go to the project directory and install the dependency package.



```
cd WebApp  
npm install
```

3. After the installation is completed, you can directly start the service locally. Enter `http://localhost:3000` in a browser to access the sample Express project locally.



```
npm start
```

## Deployment in cloud

Next, perform the following steps to make simple modifications to the initialized project, so that it can be quickly deployed through an HTTP-triggered function. The project transformation here is usually divided into the following two steps:

Change the listening address and port to `0.0.0.0:9000` .

Add the `scf_bootstrap` file.

The detailed directions are as follows:

1. In the sample Express project, you can specify the listening address and port by using the environment variable in the `./bin/www` file. If you don't specify it, port 3000 will be listened on by default.

```
/**
 * Get port from environment and store in Express.
 */
var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

/**
 * Create HTTP server.
 */
var server = http.createServer(app);

/**
 * Listen on provided port, on all network interfaces.
 */
server.listen(port);
server.on('error', onError);
server.on('listening', onListening);
```

2. Create the `scf_bootstrap` file in the root directory of the project. This file is used to configure environment variables and specify service startup commands.

#### Note

You can also complete the configuration in the console.



```
#!/bin/bash
export PORT=9000
npm run start
```

After the file is created, you need to run the following command to modify the executable permission of the file. By default, the permission `777` or `755` is required for the service to start normally.



```
chmod 777 scf_bootstrap
```

3. After the local configuration is completed, run the bootstrap file and make sure that your service can be normally started locally. Then log in to the [SLS console](#), choose **Web Application > Express Framework**, and select **Local Upload** or **Code Repository Pull** as the upload mode.

You can configure the `scf_bootstrap` file in the console. When the configuration is completed, the console automatically generates the `scf_bootstrap` file and packages it and the project code for deployment.

**Note**

The actual `scf_bootstrap` file in your project prevails. If the `scf_bootstrap` file already exists in your project, its content will not be overwritten.

When the configuration is completed, click **Complete** to deploy your Express project.



# Quickly Deploying Next.js Framework

Last updated : 2022-07-22 16:12:28

The SLS framework deployment scheme has been upgraded. You can use an SCF HTTP-triggered function to quickly deploy your Next.js service to the cloud.

Note :

## What are the differences between SLS console deployment and direct function deployment?

Both SLS console deployment and function deployment can be based on HTTP-triggered functions, and quick deployment is usually used for web frameworks.

- If you only need to develop code logic and do not need to create additional resources, you can perform quick deployment through the Serverless console.
- If you need to create more capabilities or resources, such as automatic creation of layer hosting dependencies, quick implementation of static resource isolation, and support for direct code repository pulling, in addition to code deployment, you can use the SLS console to create web applications.

## Prerequisites

Before using Tencent Cloud Serverless, you need to sign up for a [Tencent Cloud account](#) and complete the [identity verification](#).

Note :

This document introduces the SLS console deployment scheme. You can also complete the deployment in CLI by referring to [Deploying Web Function on Command Line](#).

## Directions

### Template deployment - deploying Next.js sample code

1. Log in to the [SLS console](#).
2. Choose **Create Application** and select **Web Application > Next.js Framework**.
3. Click **Next** and complete basic configuration.

4. Select **Sample Code** as the upload mode and click **Complete**. The application deployment starts.
5. After the application deployment is completed, you can view the basic information of the sample application on the application details page. In addition, you can use access the deployed Next.js project at the access path URL generated by API Gateway.

## Custom deployment - quickly deploying web application

### Prerequisites

The Node.js runtime environment has been installed locally.

### Local development

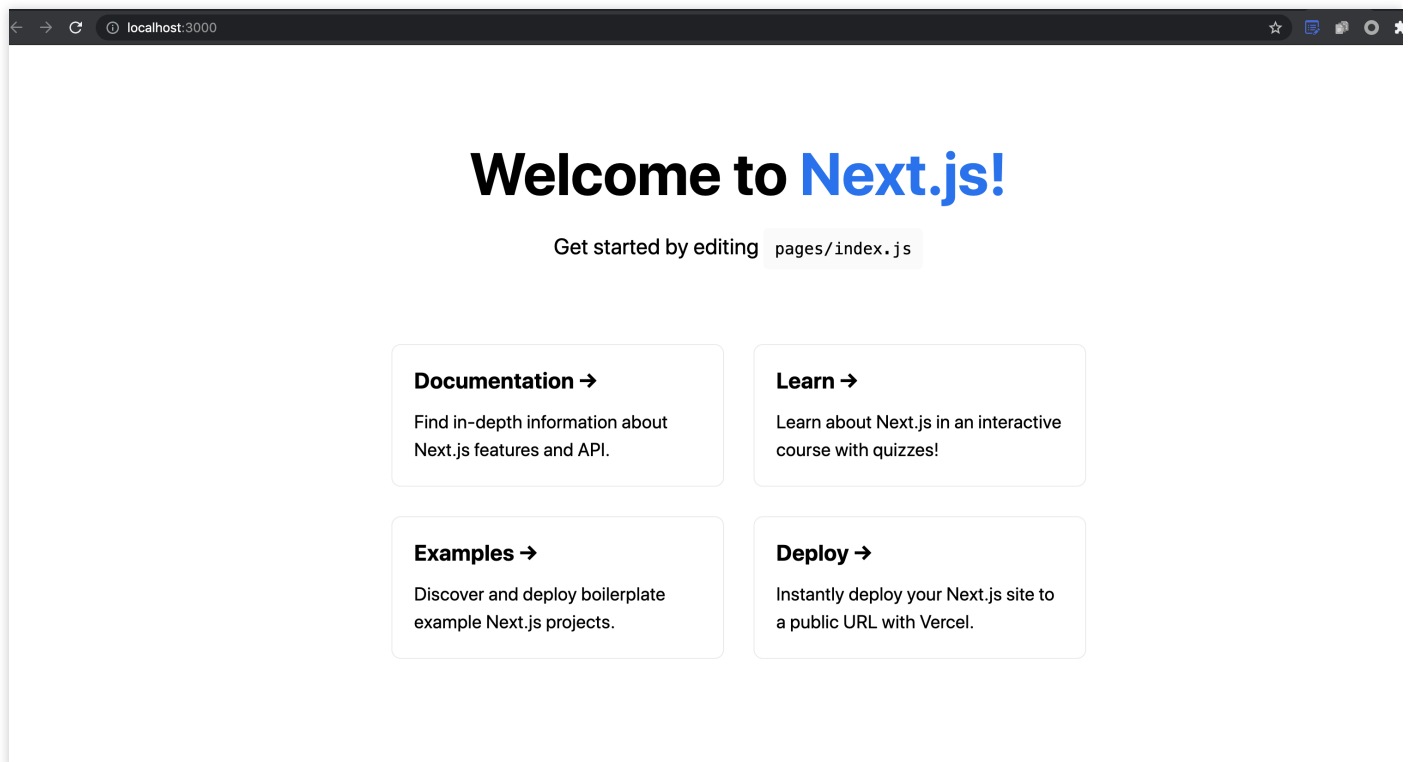
1. Refer to [Getting Started](#) to install and initialize your Next.js project:

```
npx create-next-app
```

2. In the root directory, run the following command to directly start the service locally.

```
cd my-app && npm run dev
```

3. Visit `http://localhost:3000` in a browser, and you can access the sample Next.js project locally.



## Deployment in cloud

You need to make simple modifications to the initialized project, so that the project can be quickly deployed through an HTTP-triggered function. The project transformation here is usually divided into the following two steps:

- Change the listening address and port to `0.0.0.0:9000`.
- Add the `scf_bootstrap` file.

The detailed steps are as follows:

1. Create the `scf_bootstrap` file in the root directory of the project and add the following content to it (the file is used to start the service and specify the start port):

Note :

You can also complete the configuration in the console.

```
#!/var/lang/node12/bin/node
const { nextStart } = require('next/dist/cli/next-start');
nextStart([ '--port', '9000', '--hostname', '0.0.0.0' ])
```

Note

1. Here is only a sample bootstrap file. Please adjust the configuration according to your actual business scenario.
2. The sample uses the standard node environment path of SCF. When debugging locally, you need to change it to your local path.

After the file is created, you need to run the following command to modify the executable permission of the file. By default, the permission `777` or `755` is required for the service to start normally. Below is the sample code:

```
chmod 777 scf_bootstrap
```

2. After the local configuration is completed, run the bootstrap file and make sure that your service can be normally started locally. Then log in to the [SLS console](#), select **Web Application > Next.js Framework**, and select **Local Upload** or **Code Repository Pull** as the upload mode.

You can configure the `scf_bootstrap` file in the console. When the configuration is completed, the console automatically generates the `scf_bootstrap` file and packages it and the project code for deployment.

Note :

The actual `scf_bootstrap` file in your project prevails. If the `scf_bootstrap` file already exists in your project, its content will not be overwritten.

When the configuration is completed, click **Complete** to deploy your Next.js project.

# Quickly Deploying Nuxt.js Framework

Last updated : 2022-07-22 17:49:02

The SLS framework deployment scheme has been upgraded. You can use an SCF HTTP-triggered function to quickly deploy your Nuxt.js service to the cloud.

Note :

## What are the differences between SLS console deployment and direct function deployment?

Both SLS console deployment and function deployment can be based on HTTP-triggered functions, and quick deployment is usually used for web frameworks.

- If you only need to develop code logic and do not need to create additional resources, you can perform quick deployment through the Serverless console.
- If you need to create more capabilities or resources, such as automatic creation of layer hosting dependencies, quick implementation of static resource isolation, and support for direct code repository pulling, in addition to code deployment, you can use the SLS console to create web applications.

## Prerequisites

- Before using Tencent Cloud Serverless, you need to sign up for a [Tencent Cloud account](#) and complete [identity verification](#).

Note :

This document introduces the SLS console deployment scheme. You can also complete the deployment in CLI by referring to [Deploying Web Function on Command Line](#).

## Directions

### Template deployment - deploying Nuxt.js sample code

1. Log in to the [SLS console](#).
2. Choose **Create Application** and select **Web Application > Nuxt.js Framework**.
3. Click **Next** and complete basic configuration.

4. Select **Sample Code** as the upload mode and click **Complete**. The application deployment starts.
5. After the application deployment is completed, you can view the basic information of the sample application on the application details page. In addition, you can use access the deployed Nuxt.js project at the access path URL generated by API Gateway.

## Custom deployment - quickly deploying web application

### Prerequisites

The Node.js runtime environment has been installed locally.

### Local development

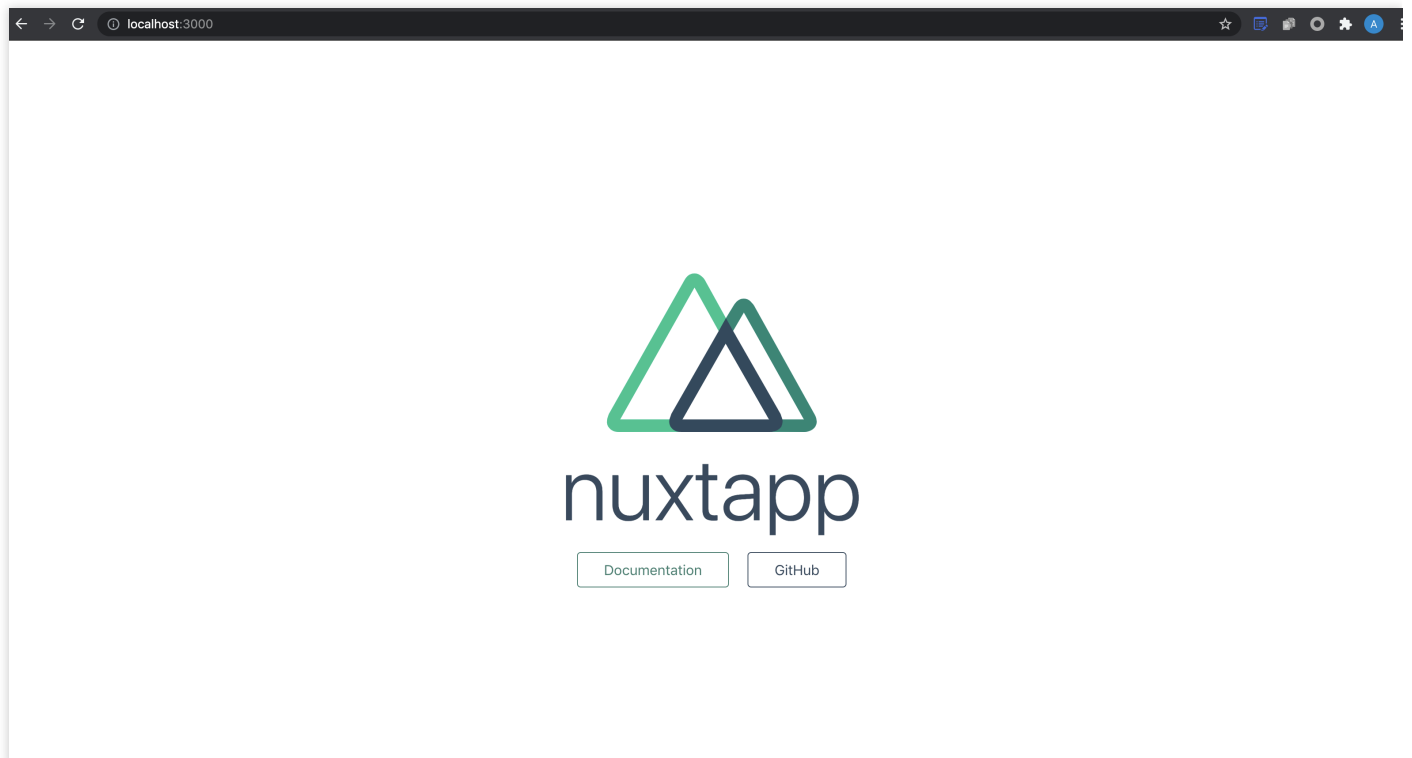
1. Refer to [Installation](#) to install and initialize your Nuxt.js project:

```
npx create-nuxt-app nuxt-app
```

2. In the root directory, run the following command to directly start the service locally.

```
cd nuxt-app && npm run dev
```

3. Visit `http://localhost:3000` in a browser, and you can access the sample Nuxt.js project locally.



## Deployment in cloud

You need to make simple modifications to the initialized project, so that the project can be quickly deployed through an HTTP-triggered function. The project transformation here is usually divided into the following two steps:

- Add the `scf_bootstrap` file.
- Change the listening address and port to `0.0.0.0:9000`.

The detailed steps are as follows:

1. Create the `scf_bootstrap` file in the root directory of the project and add the following content to it (the file is used to start the service and specify the start port):

Note :

- Here is only a sample bootstrap file. Please adjust the configuration according to your actual business scenario.
- The sample uses the standard node environment path of SCF. When debugging locally, you need to change it to your local path.

```
#!/var/lang/node12/bin/node
require("@nuxt/cli")
.run(["start", "--port", "9000", "--hostname", "0.0.0.0"])
.catch(error => {
  require("consola").fatal(error);
  require("exit")(2);
});
```

After the file is created, you need to run the following command to modify the executable permission of the file. By default, the permission `777` or `755` is required for the service to start normally. Below is the sample code:

```
chmod 777 scf_bootstrap
```

2. After the local configuration is completed, run the bootstrap file and make sure that your service can be normally started locally. Then log in to the [SLS console](#), select **Web Application > Nuxt.js Framework**, and select **Local Upload** or **Code Repository Pull** as the upload mode.

You can configure the `scf_bootstrap` file in the console. When the configuration is completed, the console automatically generates the `scf_bootstrap` file and packages it and the project code for deployment.

Note :

The actual `scf_bootstrap` file in your project prevails. If the `scf_bootstrap` file already exists in your project, its content will not be overwritten.

When the configuration is completed, click **Complete** to deploy your Nuxt.js project.



# Quickly Deploying Flask Framework

Last updated : 2022-06-13 15:10:23

The SLS framework deployment scheme has been upgraded. You can use an SCF HTTP-triggered function to quickly deploy your Flask service to the cloud.

Note :

## What are the differences between SLS console deployment and direct function deployment?

Both SLS console deployment and function deployment can be based on HTTP-triggered functions, and quick deployment is usually used for web frameworks.

- If you only need to develop code logic and do not need to create additional resources, you can perform quick deployment through the SCF console.
- If you need to create more capabilities or resources, such as automatic creation of layer hosting dependencies, quick implementation of static resource isolation, and support for direct code repository pulling, in addition to code deployment, you can use the SLS console to create web applications.

This document introduces the SLS console deployment scheme. You can also complete the deployment in CLI by referring to [Deploying Web Function on Command Line](#).

## Template Deployment - Deploying Flask Sample Code

1. Log in to the [SLS console](#).
2. Choose **Create Application** and select **Web Application** > **Flask Framework**.
3. Click **Next** and complete basic configuration.
4. Select **Sample Code** as the upload mode and click **Complete**. The application deployment starts.
5. After the application deployment is completed, you can view the basic information of the sample application on the application details page. In addition, you can use access the deployed Flask project at the access path URL generated by API Gateway.

## Custom Deployment - Quickly Deploying Web Application

### Local development

1. Confirm that Flask has been installed in your local environment.

```
pip install Flask
```

2. Create the `Hello World` sample project locally.

In the project directory, create the `app.py` file to implement the Hello World application. Below is the sample code:

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello_world():
    return 'Hello World'
if __name__ == '__main__':
    app.run()
```

3. Run the `app.py` file locally. Visit `http://127.0.0.1:5000` in a browser, and you can access the sample Express project locally.

```
$ python3 app.py
```

- Serving Flask app "app" (lazy loading)
- Environment: production  
WARNING: Do not use the development server in a production environment.  
Use a production WSGI server instead.
- Debug mode: off
- Running on `http://127.0.0.1:5000/` (Press CTRL+C to quit)

```
127.0.0.1 - - [22/Jun/2021 09:41:04] "GET / HTTP/1.1" 200 -
```



# Hello World

## Deployment in cloud

Next, perform the following steps to make simple modifications to the locally created project, so that it can be quickly deployed through an HTTP-triggered function. The steps of project transformation for Flask are as follows:

### 1. Install the dependency package

As the Flask dependency library is not provided in the standard cloud environment of SCF, you must install the dependencies and upload them together with the project code. Please create the `requirements.txt` file first:

```
#requirements.txt
Flask==1.0.2
werkzeug==0.16.0
```

Then install the `requirements.txt` file:

```
pip install -r requirements.txt
```

Note :

Due to the limitation of SCF's built-in runtime environment version (Python 3.6), only lower versions (1.0.x or earlier) of Werkzeug can be used, while higher versions may not work. The runtime environment version upgrade has been planned. Please stay tuned.

### 2. Modify the listening address and port

The listening port in the HTTP-triggered function must be `9000` , so you need to change the listening address and port to `0.0.0.0:9000` .

```
from flask import Flask
app = Flask(__name__)
```

```
@app.route('/')
def hello_world():
    return 'Hello World'
```

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=9000)
```

Note :

You can also configure the listening port through the environment variable in `scf_bootstrap` .

### 3. (Optional) Configure the `scf_bootstrap` file.

Create the `scf_bootstrap` file in the root directory of the project. This file is used to configure environment variables, specify service start commands, and make sure that your service can be started normally through this file.

```
#!/bin/bash
/var/lang/python3/bin/python3 app.py
```

After the file is created, you need to run the following command to modify the executable permission of the file. By default, the permission `777` or `755` is required for the service to start normally.

```
chmod 777 scf_bootstrap
```

Note :

- You can also complete the configuration in the console.
- In the SCF environment, only files in the `/tmp` directory are readable/writable. We recommend you select `/tmp` when outputting files. If you select other directories, write will fail due to the lack of permissions.
- If you want to output environment variables in the log, you need to add the `-u` parameter before the bootstrap command, such as `python -u app.py` .

#### 4. Console upload

Log in to the [SLS console](#), select **Web Application > Flask Framework**, and select **Local Upload** or **Code Repository Pull** as the upload mode.

You can configure the `scf_bootstrap` file in the console. When the configuration is completed, the console automatically generates the `scf_bootstrap` file and packages it and the project code for deployment.

Note :

The actual `scf_bootstrap` file in your project prevails. If the `scf_bootstrap` file already exists in your project, its content will not be overwritten.

When the configuration is completed, click **Complete** to deploy your Flask project.

#### Advanced configuration management

In **Advanced Configuration**, you can perform more application management operations, such as creating layers, binding custom domains, and configuring environment variables.

# Quickly Deploying Laravel Framework

Last updated : 2021-08-20 16:21:27

## Overview

This document describes how to use a web function to quickly migrate a local Laravel service to the cloud.

Note :

This document mainly describes how to deploy in the console. You can also complete the deployment on the command line. For more information, please see [Deploying Framework on Command Line](#).

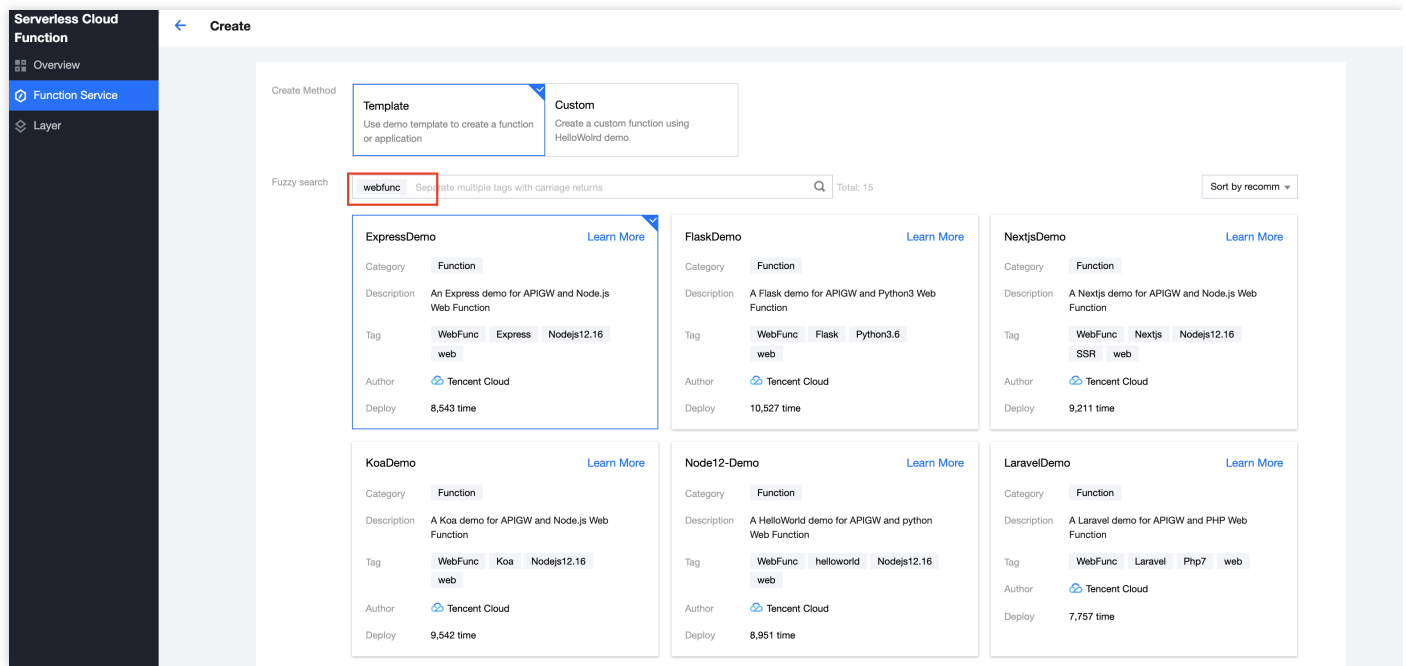
## Prerequisites

Before using SCF, you need to sign up for a [Tencent Cloud account](#) and complete [identity verification](#) first.

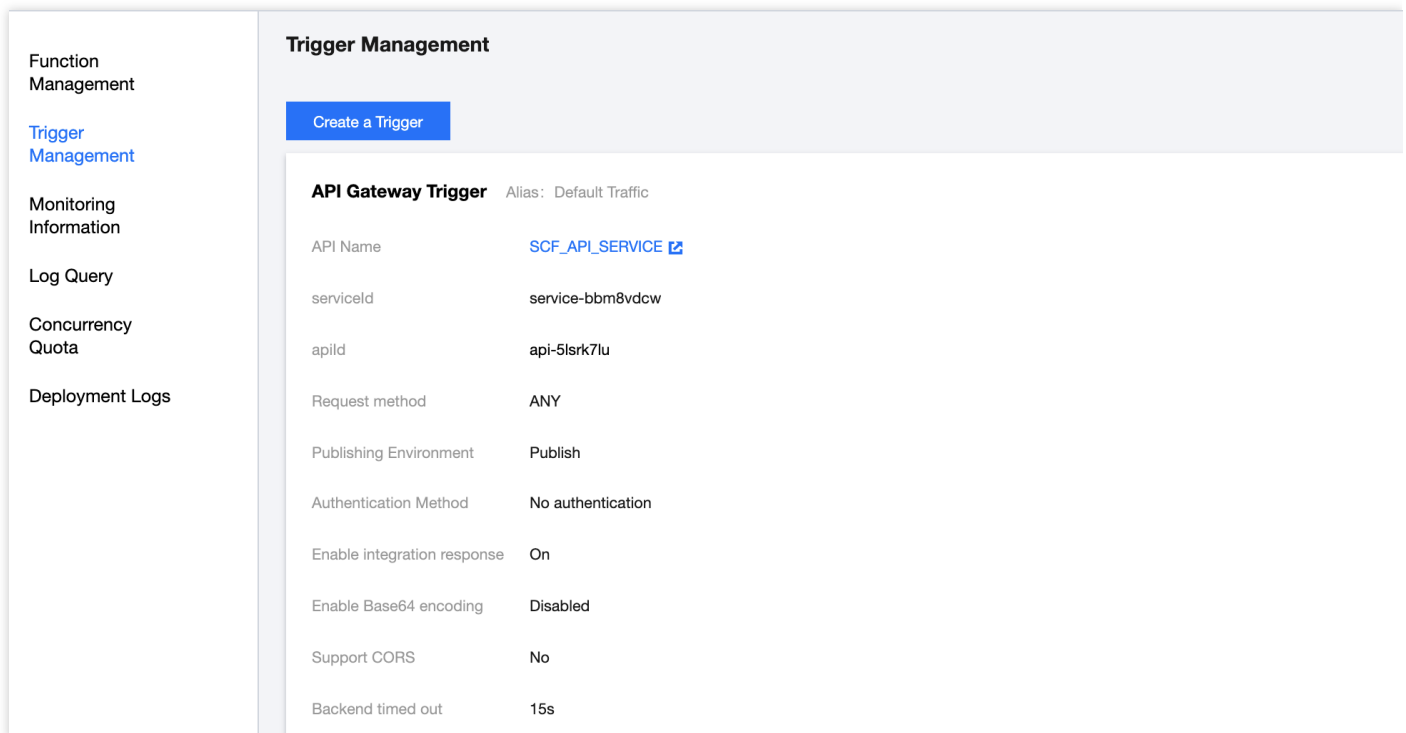
## Directions

### Template deployment - quick deployment of Laravel project

1. Log in to the [SCF console](#) and click **Function Service** on the left sidebar.
2. Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.
3. Select **Template** for **Creation Method**, enter `WebFunc` in the search box to filter all web function templates, select **Laravel Framework Template**, and click **Next** as shown below:



- On the **Configuration** page, you can view and modify the specific configuration information of the template project.
- Click **Complete**. After creating the web function, you can view its basic information on the **Function Management** page.
- You can access the deployed Laravel project at the access path URL generated by API Gateway. Click **Trigger Management** on the left to view the access path as shown below:



- Click the access path URL to access the Laravel project.

## Custom deployment - quick migration of local project to cloud

### Local development

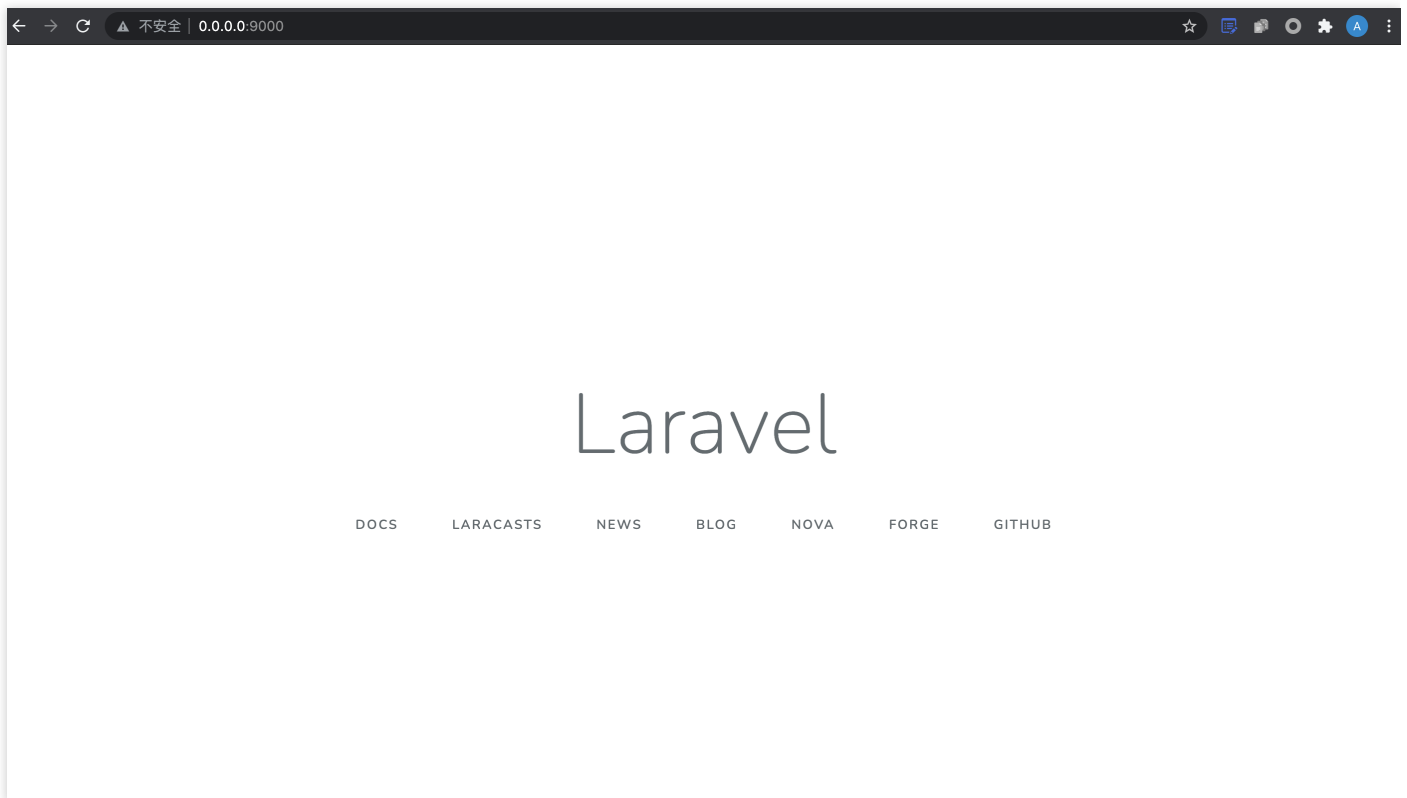
1. Refer to [Getting Started on macOS](#) to set up the Laravel development environment locally.
2. Create the sample Laravel project locally. Enter the project directory and run the following command to initialize it.

```
composer create-project --prefer-dist laravel/laravel blog
```

3. Run the following command to start the sample project locally. Below is the sample code:

```
$ php artisan serve --host 0.0.0.0 --port 9000
Laravel development server started: <http: 0.0.0.0:9000="">
[Wed Jul 7 11:22:05 2021] 127.0.0.1:54350 [200]: /favicon.ico
```

4. Visit `http://0.0.0.0:9000` in a browser, and you can access the sample Laravel project locally as shown below:



### Deployment in cloud

Add the following content to the file (which is used to configure environment variables and start services. Here is only a sample. Please adjust the specific operations according to your actual business scenario):

Next, perform the following steps to make simple modifications to the initialized project, so that it can be quickly deployed through a web function. The steps of project transformation are as follows:



### 1. Add the `scf_bootstrap` bootstrap file

Create the `scf_bootstrap` bootstrap file in the project root directory. This file is used to configure environment variables, specify service bootstrap commands, and make sure that your service can be started normally through this file.

Note :

- `scf_bootstrap` must have the executable permission of `755` or `777` .
- If you want to output environment variables in the log, you need to add the `-u` parameter before the bootstrap command, such as `python -u app.py` .

### 2. Modify the file read/write path

In the SCF environment, only files in the `/tmp` directory are readable/writable. If you select other directories, write will fail due to the lack of permissions. Therefore, you need to inject environment variables in the `scf_bootstrap` file to adjust the output directory of the Laravel framework:

```
#!/bin/bash
# Inject the SERVERLESS flag
export SERVERLESS=1
# Modify the template compilation cache path, as only `/tmp` is readable/writable in SCF
export VIEW_COMPILED_PATH=/tmp/storage/framework/views
# Modify `session` to store it in the memory (array type)
export SESSION_DRIVER=array
# Output logs to `stderr`
export LOG_CHANNEL=stderr
# Modify the application storage path
export APP_STORAGE=/tmp/storage
# Initialize the template cache directory
mkdir -p /tmp/storage/framework/views
```

### 3. Modify the listening address and port

The listening port in the web function must be `9000` , so you need to specify the listening port in `scf_bootstrap` through the following command:

```
/var/lang/php7/bin/php artisan serve --host 0.0.0.0 --port 9000
```

The content of `scf_bootstrap` is as follows:

```
src > scf_bootstrap
1  #!/bin/bash
2
3  #####
4  # 注入 serverless 环境下的环境变量
5  #####
6  # 注入 SERVERLESS 标识
7  export SERVERLESS=1
8  # 修改模板编译缓存路径，云函数只有 /tmp 目录可读写
9  export VIEW_COMPILED_PATH=/tmp/storage/framework/views
10 # 修改 session 以内存方式（数组类型）存储
11 export SESSION_DRIVER=array
12 # 日志输出到 stderr
13 export LOG_CHANNEL=stderr
14 # 修改应用存储路径
15 export APP_STORAGE=/tmp/storage
16
17 # 初始化模板缓存目录
18 mkdir -p /tmp/storage/framework/views
19
20 # HTTP 直通函数由于是基于 docker 镜像运行，所以必须监听地址为 0.0.0.0，并且端口为 9000
21 # 云端可执行文件路径 /var/lang/php7/bin/php
22 /var/lang/php7/bin/php artisan serve --host 0.0.0.0 --port 9000
23
```

#### 4. Deploy in the cloud

After the local configuration is completed, run the bootstrap file and make sure that your service can be normally started locally. Then, perform the following steps to deploy Laravel:

- Log in to the [SCF console](#) and click **Function Service** on the left sidebar.
- Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.

iii. Select **Custom Creation** for **Creation Method** and configure the options as prompted as shown below:

Create Method

Template  
Use demo template to create a function or application

Custom  
Create a custom function using HelloWolrd demo

**Basic Configurations**

Function Type \* ☐ Event Function ☒ Web Function ⓘ

Function name \*   
It supports 2 to 60 characters, including letters, numbers, underscores and hyphens. It must start with a letter and end with a number or letter.

Region \*

Deployment Mode \* ☒ Code ☐ Image

Runtime Environment \*

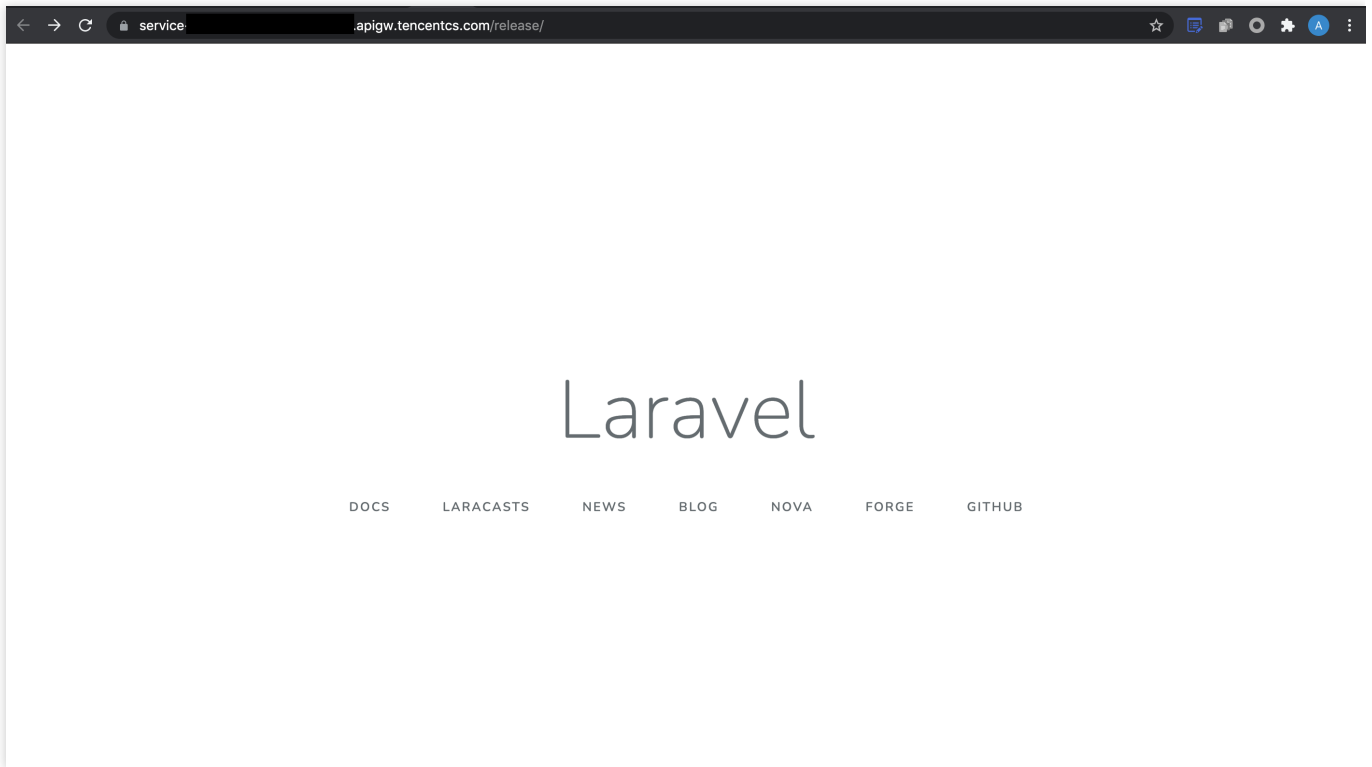
**Function Codes** ⓘ Please modify the listening port of your project to 9000 before uploading the project.

Submitting Method \* ☐ Online editing ☐ Local ZIP file ☒ Local folder ☐ Upload a ZIP pack via COS

Function Codes \*    
Please select a folder (up to 250M)

- **Function Type:** select **Web function**.
- **Function Name:** enter the name of your function.
- **Region:** enter your function deployment region, such as Chengdu.
- **Deployment Method:** select **Code deployment** and upload your local project.
- **Runtime environment:** select "Php7".

iv. After the deployment is completed, click the generated URL to access your Laravel application as shown below:



### Development management

After the deployment is completed, you can quickly access and test your web service in the SCF console and try out various features of SCF, such as layer binding and log management. In this way, you can enjoy the advantages of low cost and flexible scaling brought by the serverless architecture.

# Quickly Deploying Nest.js Framework

Last updated : 2022-06-13 15:10:23

The SLS framework deployment scheme has been upgraded. You can use an SCF HTTP-triggered function to quickly deploy your Nest.js service to the cloud.

Note :

## What are the differences between SLS console deployment and direct function deployment?

Both SLS console deployment and function deployment can be based on HTTP-triggered functions, and quick deployment is usually used for web frameworks.

- If you only need to develop code logic and do not need to create additional resources, you can perform quick deployment through the SCF console.
- If you need to create more capabilities or resources, such as automatic creation of layer hosting dependencies, quick implementation of static resource isolation, and support for direct code repository pulling, in addition to code deployment, you can use the SLS console to create web applications.

## Prerequisites

- Before using Tencent Cloud SCF, you need to sign up for a [Tencent Cloud account](#) and complete [identity verification](#).

Note :

This document introduces the SLS console deployment scheme. You can also complete the deployment in CLI by referring to [Deploying Web Function on Command Line](#).

## Directions

### Template deployment - deploying Nest.js sample code

1. Log in to the [SLS console](#).
2. Choose **Create Application** and select **Web Application** > **Nest.js Framework**.
3. Click **Next** and complete basic configuration.

4. Select **Sample Code** as the upload mode and click **Complete**. The application deployment starts.
5. After the application deployment is completed, you can view the basic information of the sample application on the application details page. In addition, you can use access the deployed Nest.js project at the access path URL generated by API Gateway.

## Custom deployment - quickly deploying web application

### Prerequisites

The Node.js runtime environment has been installed locally.

### Local development

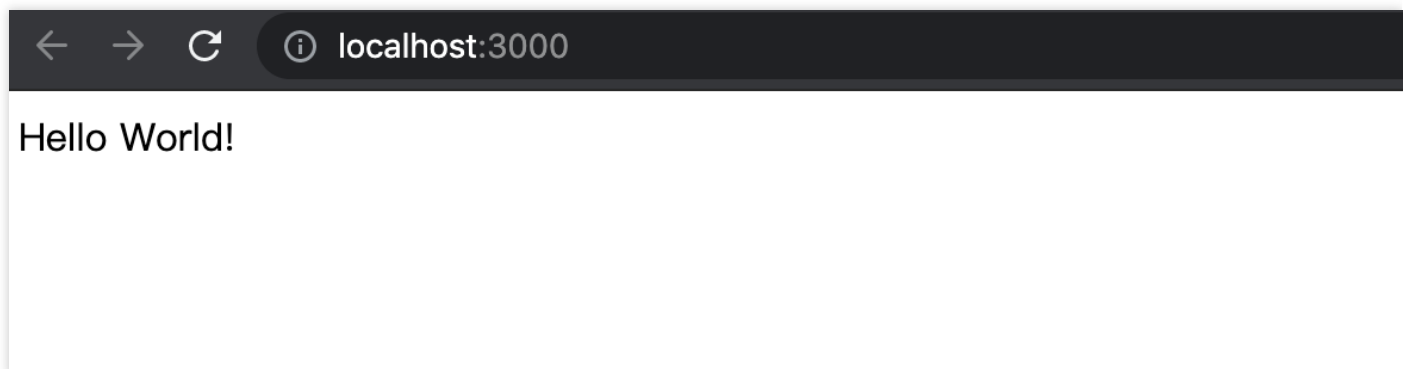
1. Refer to [First steps](#) to initialize your Nest.js project:

```
npm i -g @nestjs/cli
nest new nest-app
```

2. In the root directory, run the following command to directly start the service locally.

```
cd nest-app && npm run start
```

3. Visit `http://localhost:3000` in a browser, and you can access the sample Nest.js project locally.



### Deployment in cloud

You need to make simple modifications to the initialized project, so that the project can be quickly deployed through an HTTP-triggered function. The project transformation here is usually divided into the following two steps:

- Add the `scf_bootstrap` file.
- Change the listening address and port to `0.0.0.0:9000`.

The detailed steps are as follows:

1. Change the listening port to `9000` in the `./dist/main.js` file.

```
dd / Downloads / test / 小程序-直通-demo / nodejs / nest / nest-ap
"use strict";
Object.defineProperty(exports, "__esModule", { value: true });
const core_1 = require("@nestjs/core");
const app_module_1 = require("./app.module");
async function bootstrap() {
    const app = await core_1.NestFactory.create(app_module_1.AppModule);
    await app.listen(9000);
}
bootstrap();
//# sourceMappingURL=main.js.map
```

2. Create the `scf_bootstrap` file in the root directory of the project and add the following content to it (the file is used to start the service):

Note :

You can also complete the configuration in the console.

```
#!/bin/bash
SERVERLESS=1 /var/lang/node12/bin/node ./dist/main.js
```

Note :

1. Here is only a sample bootstrap file. Please adjust the configuration according to your actual business scenario.
2. The sample uses the standard node environment path of SCF. When debugging locally, you need to change it to your local path.

After the file is created, you need to run the following command to modify the executable permission of the file. By default, the permission `777` or `755` is required for the service to start normally. Below is the sample code:

```
chmod 777 scf_bootstrap
```

3. After the local configuration is completed, run the bootstrap file and make sure that your service can be normally started locally. Then log in to the [SLS console](#), select **Web Application > Nest.js Framework**, and select **Local Upload** or **Code Repository Pull** as the upload mode.

You can configure the `scf_bootstrap` file in the console. When the configuration is completed, the console automatically generates the `scf_bootstrap` file and packages it and the project code for deployment.

Note :

The actual `scf_bootstrap` file in your project prevails. If the `scf_bootstrap` file already exists in your project, its content will not be overwritten.

When the configuration is completed, click **Complete** to deploy your Nest.js project.



# Quickly Deploying Django Framework

Last updated : 2022-06-13 15:10:23

The SLS framework deployment scheme has been upgraded. You can use an SCF HTTP-triggered function to quickly deploy your Django service to the cloud.

Note :

## **What are the differences between SLS console deployment and direct function deployment?**

Both SLS console deployment and function deployment can be based on HTTP-triggered functions, and quick deployment is usually used for web frameworks.

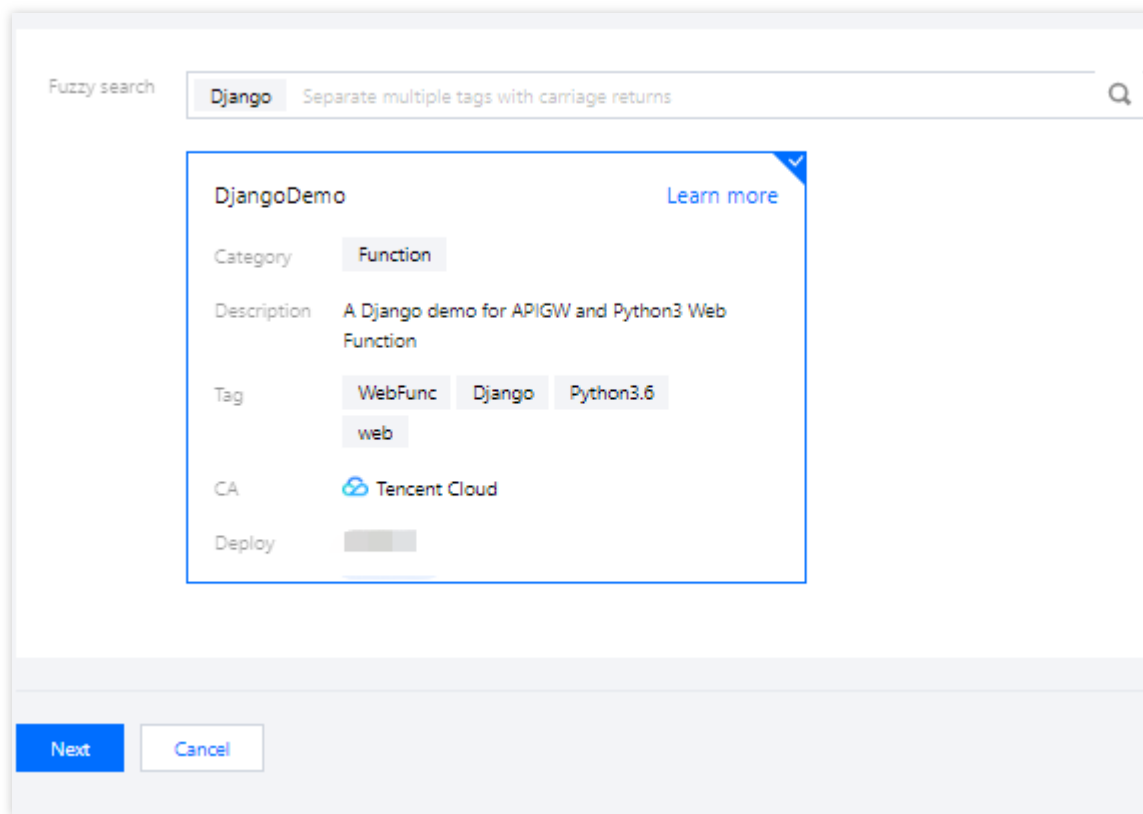
- If you only need to develop code logic and do not need to create additional resources, you can perform quick deployment through the SCF console.
- If you need to create more capabilities or resources, such as automatic creation of layer hosting dependencies, quick implementation of static resource isolation, and support for direct code repository pulling, in addition to code deployment, you can use the SLS console to create web applications.

This document introduces the SLS console deployment scheme. You can also complete the deployment in CLI by referring to [Deploying Web Function on Command Line](#).

## Template Deployment - Deploying Django Sample Code

1. Log in to the [SLS console](#).

2. Choose **Create Application** and select **Web Application** > **Django Framework**.



3. Click **Next** and complete basic configuration.

4. Select **Sample Code** as the upload mode and click **Complete**. The application deployment starts.

5. After the application deployment is completed, you can view the basic information of the sample application on the application details page. In addition, you can use access the deployed Django project at the access path URL generated by API Gateway.

## Custom Deployment - Quickly Deploying Web Application

### Local development

1. Run the following command to confirm that Django has been installed in your local environment.

```
python -m pip install Django
```

2. Create the `Hello World` sample project locally.

```
django-admin startproject helloworld && cd helloworld
```

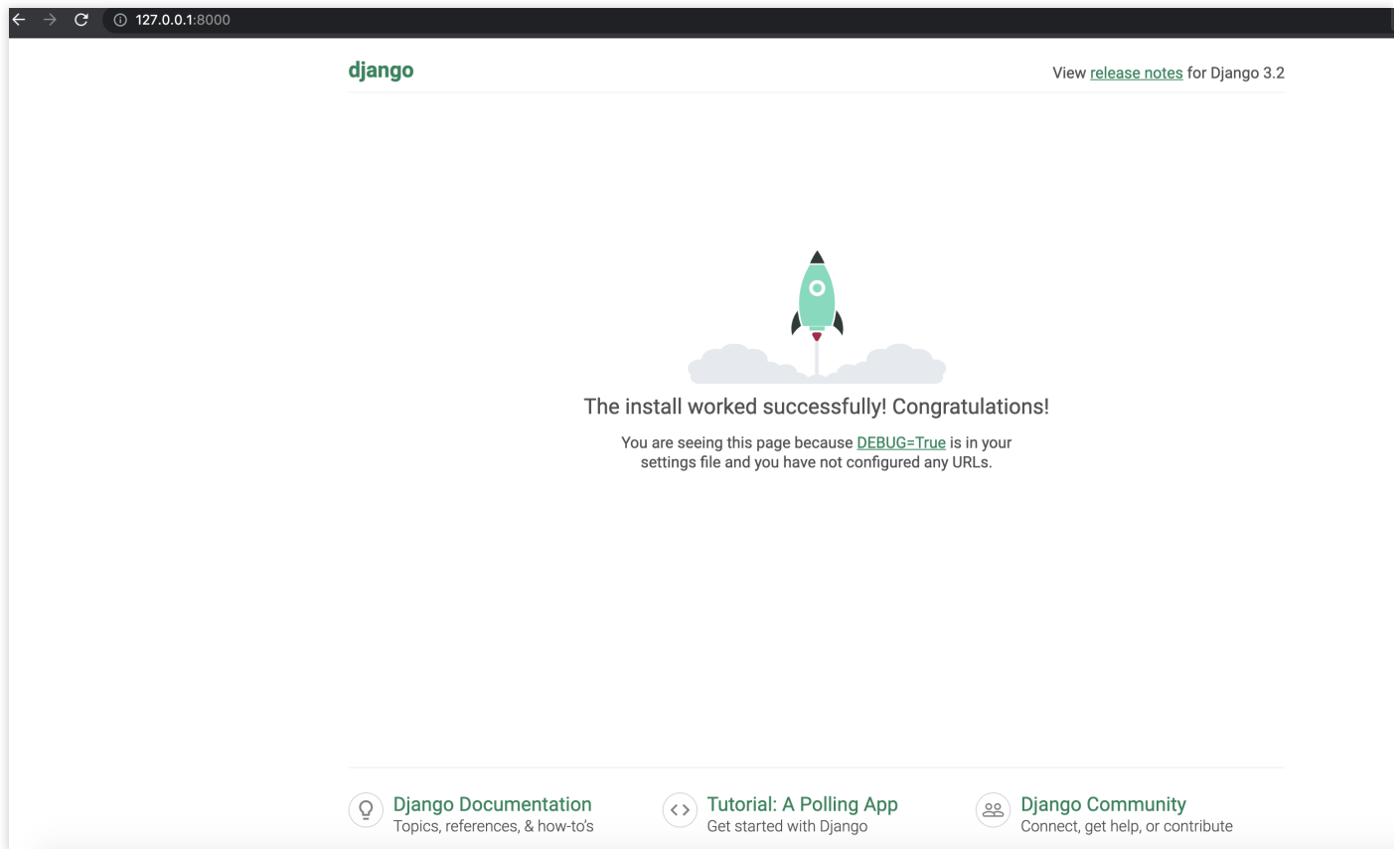
The directory structure is as follows:

```
$ tree
. manage.py Manager
|--***
| |-- __init__.py Package
| |-- settings.py Settings file
| |-- urls.py Route
| `-- wsgi.py Deployment
```

3. Run the `python manage.py runserver` command locally to start the bootstrap file. Below is the sample code:

```
$ python manage.py runserver
July 27, 2021 - 11:52:20
Django version 3.2.5, using settings 'helloworld.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

4. Visit `http://127.0.0.1:8000` in a browser, and you can access the sample Django project locally as shown below:



## Deployment in cloud

You need to make simple modifications to the locally created project, so that the project can be quickly deployed through an HTTP-triggered function. The project modification steps for Django are as follows:

### 1. Install the dependency package

As the Django dependency library is not provided in the standard cloud environment of SCF, you must install the dependencies and upload them together with the project code. Please create the `requirements.txt` file first with the following content:

```
Django==3.1.3
```

Run the following installation command:

```
pip install -r requirements.txt -t .
```

Note :

As the `db.sqlite3` library is referenced in the initialized default project, you need to install this dependency at the same time or comment out the `DATABASES` field in the `setting.py` file of the project.

## 2. (Optional) Configure the `scf_bootstrap` file.

Note :

You can also complete the configuration in the console.

The listening port in the HTTP-triggered function must be **9000**, so you need to change the listening address and port. To do so, you need to create the `scf_bootstrap` file in the root directory of the project and add the following content to the file to configure environment variables, specify service start commands, and so on to make sure that your service can be started normally through the file:

```
#!/bin/bash
/var/lang/python3/bin/python3 manage.py runserver 9000
```

After the file is created, you need to run the following command to modify the executable permission of the file. By default, the permission `777` or `755` is required for the service to start normally. Below is the sample code:

```
chmod 777 scf_bootstrap
```

Note :

- In the SCF environment, only files in the `/tmp` directory are readable/writable. We recommend you select `/tmp` when outputting files. If you select other directories, write will fail due to the lack of permissions.
- If you want to output environment variables in logs, you need to add the `-u` parameter before the bootstrap command, such as `python -u app.py`.

After the local configuration is completed, run the following command to start the service (take running the command in the `scf_bootstrap` directory as an example) and make sure that your service can be normally started locally.

Note :

Be sure to change the `python` path to the local path during local testing.

```
./scf_bootstrap
```

### 3. Console upload

Log in to the [SLS console](#), select **Web Application > Django Framework**, and select **Local Upload** or **Code Repository Pull** as the upload mode.

You can configure the `scf_bootstrap` file in the console. When the configuration is completed, the console automatically generates the `scf_bootstrap` file and packages it and the project code for deployment.

Note :

The actual `scf_bootstrap` file in your project prevails. If the `scf_bootstrap` file already exists in your project, its content will not be overwritten.

When the configuration is completed, click **Complete** to deploy your Django project.

### Advanced configuration management

In **Advanced Configuration**, you can perform more application management operations, such as creating layers, binding custom domains, and configuring environment variables.

# Quickly Deploying Native WordPress Application

Last updated : 2021-08-20 16:21:27

Tencent Cloud Serverless Framework provides a new deployment method for WordPress based on the serverless architecture. By using the [Serverless Framework WordPress component](#), you can quickly deploy a WordPress project in just a few steps.

## Architecture Overview

This component mainly creates the following resources for you:

Module	Description
SCF	It implements the access layer of Serverless WordPress to run WordPress
API Gateway	It is the ingress of WordPress and implements RESTful APIs
CFS	It is the serverless storage warehouse for WordPress
TDSQL-C Serverless	You can create a TDSQL-C for MySQL serverless database to implement a pay-as-you-go and automatically scalable database service
VPC	It is used to connect SCF, CFS, and TDSQL-C Serverless over the private network to ensure network isolation

## Features

- **Support for native WordPress framework**

With the Serverless WordPress component, you can directly deploy native WordPress projects without having to make any modifications, which is non-intrusive to the framework and supports subsequent version upgrade.

- **Reduced costs**

From the access layer to compute layer to storage layer, everything uses serverless resources to truly implement pay-as-you-go billing and auto scaling, greatly reducing the costs.

- **Easy deployment**

With the Serverless WordPress component, you can quickly complete WordPress application deployment by using just a few lines of configuration in a YAML file, greatly lowering the deployment threshold.

# Deployment Steps

You can deploy Serverless WordPress on the **command line** or in the **console** in the following steps:

## Prerequisites

- You have activated [SCF](#).
- You have activated [CFS](#).
- (Optional) You have prepared a custom domain name.

## Deployment in console

Note :

Currently, only four regions are supported: Beijing, Guangzhou, Nanjing, and Shanghai.

1. Log in to the [SAC console](#) and click **Create Application**.
2. Enter the application name as prompted, select **Application Template > WordPress Application**, and click **Create** to create an application.
3. On the serverless application page, click **Access Application** to access your WordPress project. You can also configure a custom domain name on the application details page.

## Deployment on command line

Note :

Currently, only four AZs are supported: `ap-guangzhou-4` , `ap-shanghai-2` , `ap-beijing-3` , and `ap-nanjing-1` .

1. Create the `wordpress-demo` folder locally and download the application from the [WordPress official website](#) into it.
2. In the folder, create the `serverless.yml` configuration file and complete application configuration as follows (for more information on the configuration, please see the [configuration document](#)):

```
app: wordpress
stage: dev
component: wordpress
name: wordpressDemo
inputs:
```



```

region: ap-shanghai # Project region
zone: ap-shanghai-2
src: # Project path, which should be your WordPress path
src: ./wordpress
exclude:
- .env
apigw: # API Gateway configuration
customDomains: # (Optional) Bind a custom domain name
- domain: abc.com # The custom domain name to be bound
certId: abcdefg # Unique certificate ID of the custom domain name to be bound
customMap: true # Whether the path is custom
pathMap:
- path: /
environment: release
protocols: # Type of the protocol of the custom domain name to be bound, which
is the same as that of the frontend service protocol
- http
- https

```

After you complete the above configuration, your project structure will be as follows:

```

.wordpress-demo
├─ wordpress # WordPress source file
├─ serverless.yml # Configuration file
└─ .env # Environment variable file

```

3. In the root directory, run `sls deploy` to complete the deployment. Below is an example:

```

$ sls deploy
serverless ⚡framework
Action: "deploy" - Stage: "dev" - App: "appDemo" - Instance: "wordpressDemo"
region: ap-shanghai
zone: ap-shanghai-2
vpc:
...
cfs:
...
db:
...
apigw:
created: true
url: https://service-xxxxxx.sh.apigw.tencentcs.com/release/
...
layer:
...
wpInitFaas:
...

```

```
wpServerFaas:
...
```

- After the deployment succeeds, click the URL output in the `apigw` part, configure the account and password as prompted, and you can start using your WordPress application.

## FAQs

### What should I do if deployment failed due to permission problems?

- Check whether the root account/sub-account has the following permissions:
  - Check roles: **SCF\_QcsRole**, **SLS\_QcsRole**, and **CODING\_QcsRole**
  - Check permissions:
    - `SCF_QcsRole` must have the **CFSFullAccess** permission
    - `CODING_QCSRole` must have the **QcloudSLSFullAccess**, **QcloudSSLFullAccess**, and **QcloudAccessForCODINGRole** permissions
- For a sub-account, you also need to check the following permissions:  
The account should have permissions to use **SLS**, **SCF**, **CFS**, **TDSQL-C**, and **CODING**.

After a custom domain name was bound, the error message `{"message": "There is no api match env_mapping '\/'"}` was reported. What should I do?

Modify the custom mapping in the [API Gateway console](#) as shown below:

Path Mapping ☐ Default path mapping ⓘ ☒ Custom path mapping

Path	Environment
<input type="text" value="/"/>	<input type="text" value="Publish"/>




[Add path mapping \(1/3\)](#)

### How do I modify `php.ini` to change the limit on the size of uploaded files?

- Modify the layer code. Move the `php.ini` file in the `etc` folder to the `etc/php.d` folder. You can also directly use the [package](#) provided by Tencent Cloud.

When packaging and uploading the layer again, pay attention to the hierarchy in the package and compress only

files in the parent directory; otherwise, function initialization will fail:

 bin	2021/6/28 14:10	File folder
 etc	2021/6/28 14:10	File folder
 lib	2021/6/28 14:10	File folder

2. Modify the bootstrap code of the `wp-server-xxx` function as follows:

```
#!/bin/bash
export PATH="/opt/bin:$PATH"
export LD_LIBRARY_PATH=/opt/lib/:$LD_LIBRARY_PATH
export PHP_INI_SCAN_DIR=/opt/etc/php.d
php -d extension_dir=/opt/lib/php/modules/ sl_handler.php 1>&2
```

### How do I troubleshoot the "event too large" error?

Currently, you can only upload an event of up to **6 MB** in size for a function. Larger files cannot be uploaded.

Currently, Base64 encoding by API Gateway will increase your code size by 1.5 times. Therefore, we recommend you upload a file below **3.5 MB** in size.

### How do I modify files in the WordPress root directory?

Currently, such files are mounted to CFS and cannot be modified directly. We recommend you install the File Manager plugin to manage files in the root directory.