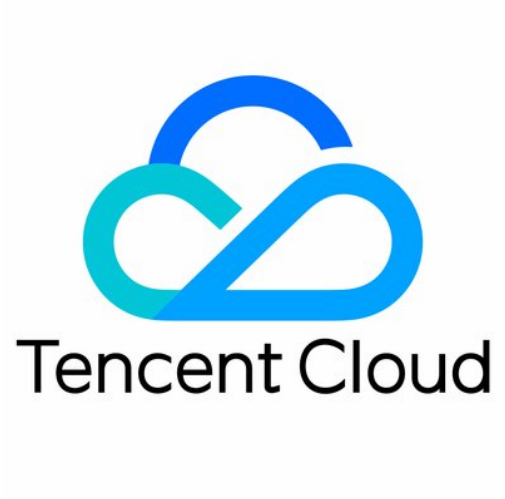
Serverless Cloud Function

Web Framework Development

Product Documentation



Contents

Web Framework Development
- Deploying Framework on Command Line
- Quickly Deploying Egg Framework
- Quickly Deploying Express Framework
- Quickly Deploying Flask Framework
- Quickly Deploying Koa Framework
- Quickly Deploying Laravel Framework
- Quickly Deploying Nest.js Framework
- Quickly Deploying Next.js Framework
- Quickly Deploying Nuxt.js Framework
- Quickly Deploying Django Framework

Web Framework Development
Deploying Framework on Command Line

Last updated：2025-02-12 11:08:57

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	Quickly Deploying Express Framework
Koa	Quickly Deploying Koa Framework
Egg	Quickly Deploying Egg Framework
Next.js	Quickly Deploying Next.js Framework
Nuxt.js	Quickly Deploying Nuxt.js Framework
Nest.js	Quickly Deploying Nest.js Framework
Flask	Quickly Deploying Flask Framework
Django	Quickly Deploying Django Framework
Laravel	Quickly Deploying Laravel Framework

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 .yml file

Create a `serverless.yml` 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 i
    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
    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
`/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');
});
```

nextjs

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

/*
# As the HTTP passthrough function runs based on the docker image, the
listening 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
listening 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/node
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
listening 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
listening 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/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

# As the HTTP passthrough function runs based on the docker image, the
listening 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：2025-02-12 11:08:57

Overview

This document describes how to quickly deploy a local Egg project to the cloud through a web function.

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.

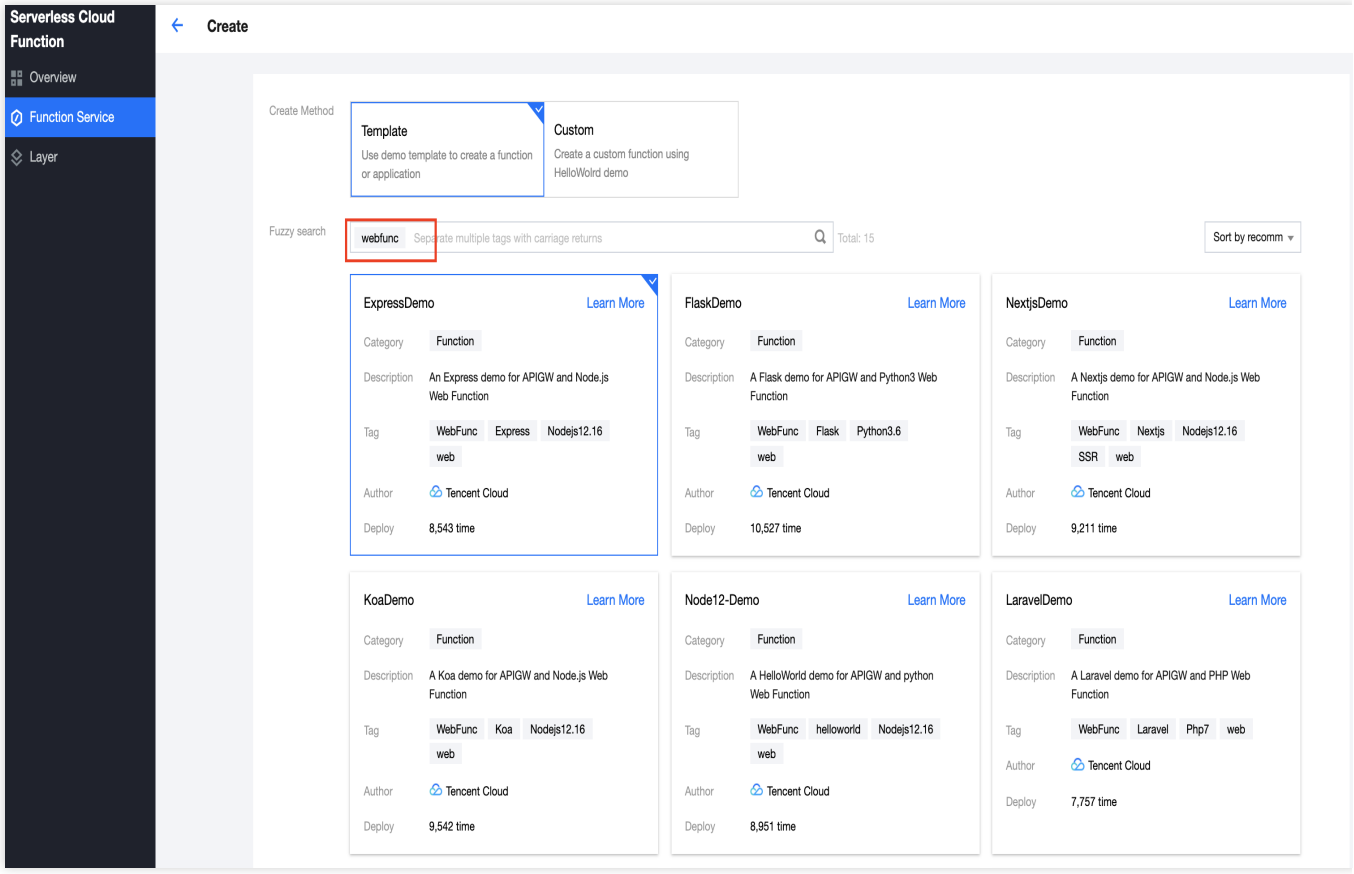
Prerequisites

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

Directions

Template deployment - quick deployment of Egg project

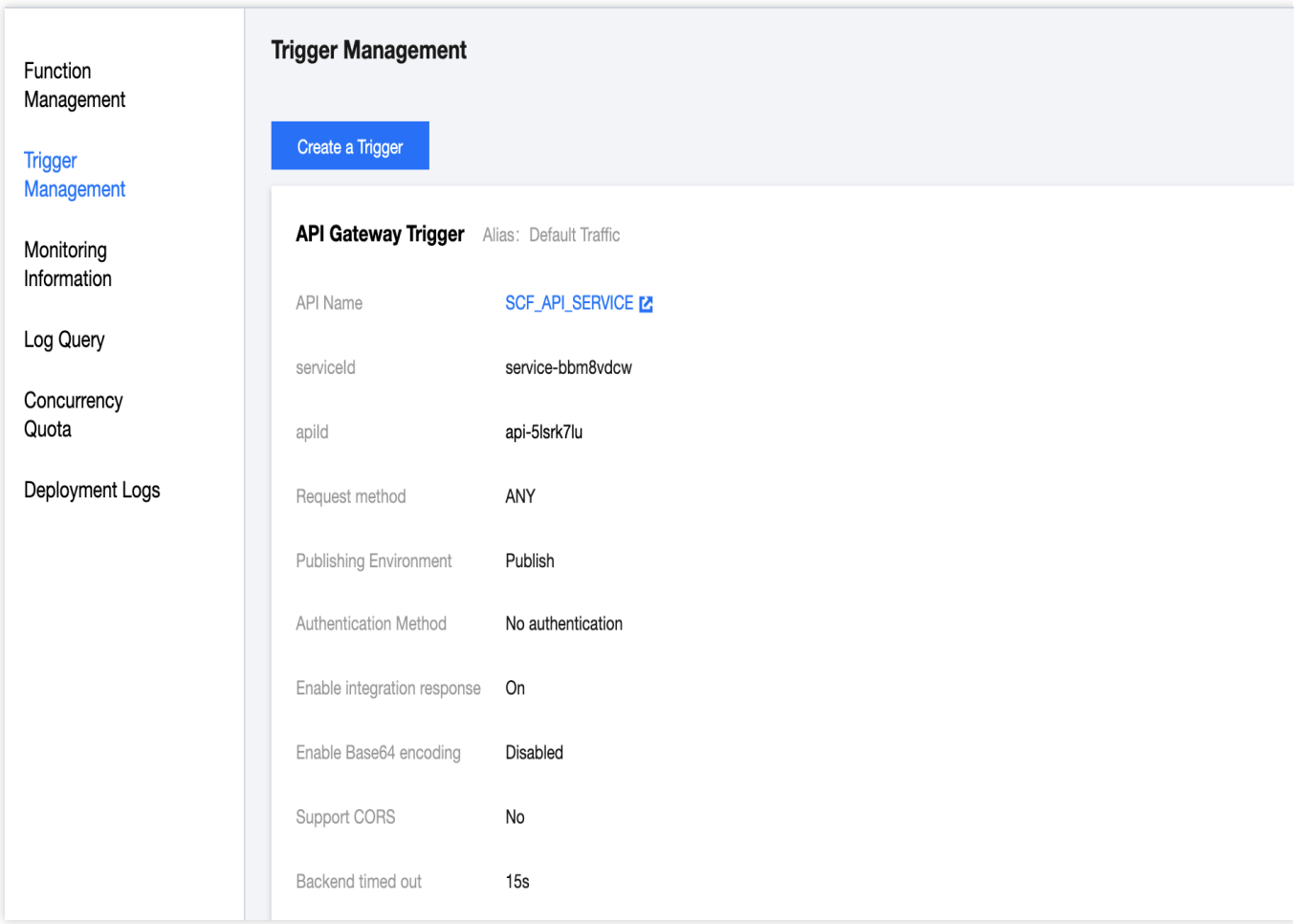
1. Log in to the SCF console and click **Function Service** on the left sidebar.
2. Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.
3. Select **Template** for **Creation Method**, enter `Egg` in the search box to filter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



4. On the **Configuration** page, you can view and modify the specific configuration information of the template project.

5. Click **Complete**. After creating the web function, you can view its basic information on the **Function Management** page.

6. You can access the deployed Egg project at the access path URL generated by API Gateway. Click **Trigger Management** on the left to view the access path as shown below:



7. Click the access path URL to access the Egg project.

Custom deployment - quick migration of local project to cloud

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. Visit `http://localhost:7001` in a browser, and you can access the sample Egg project locally.

Cloud deployment

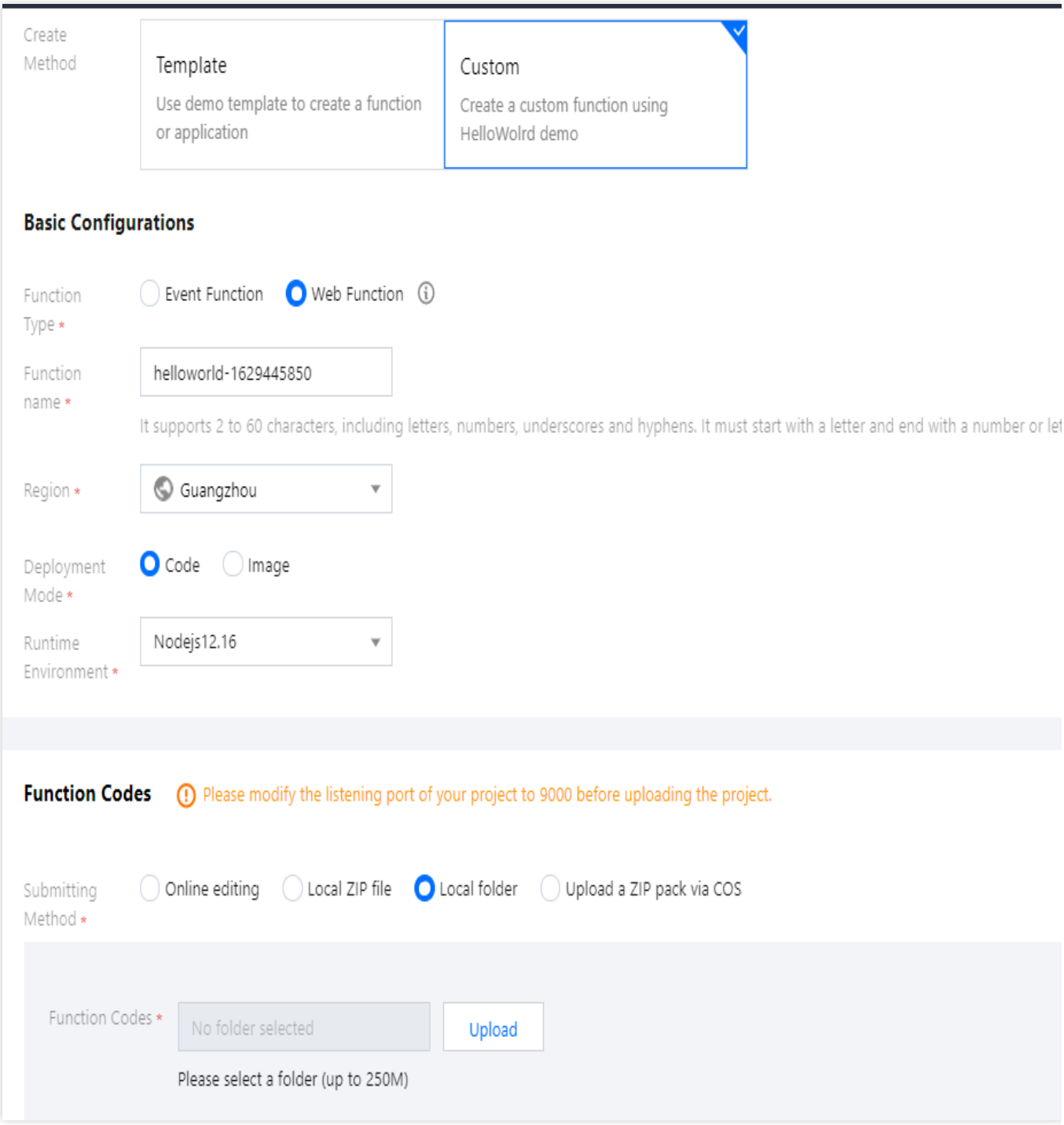
Next, perform the following steps to make simple modifications to the initialized project, so that it can be quickly deployed through a web function. The project transformation here is usually divided into the following three steps:
Change the listening address and port to `0.0.0.0:9000` .
Modify the write path. Only the `/tmp` directory is readable/writable in the SCF environment.
Add the `scf_bootstrap` bootstrap file.
The specific steps are as follows:
1. Create the `scf_bootstrap` bootstrap file in the project root directory and add the following content to it (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):

```
#!/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
 * `NODE_LOG_DIR` changes the default node write path of `egg-scripts` (~/logs) to
 * `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');
});
```

2. After the creation is completed, 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 it to start normally. Below is the sample code:

```
chmod 777 scf_bootstrap
```

3. Log in to the SCF console and click **Function Service** on the left sidebar.
4. Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.
5. Select **Custom Creation** for **Creation Method** and configure the options as prompted as shown below:



Function Type: select **Web function**.

Function Name: enter the name of your function.

Region: enter your function deployment region, which is **Guangzhou** by default.

Runtime Environment: select **Nodejs 12.16**.

Deployment Method: select **Code deployment** and upload your local project.

Submitting Method: select **Local folder**.

Function Code: select the specific local folder where the function code is.

6. Click **Complete**.

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 Express Framework

Last updated：2025-02-12 11:08:57

Overview

This document describes how to quickly deploy a local Express project to the cloud through an HTTP-triggered function.

Note:

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

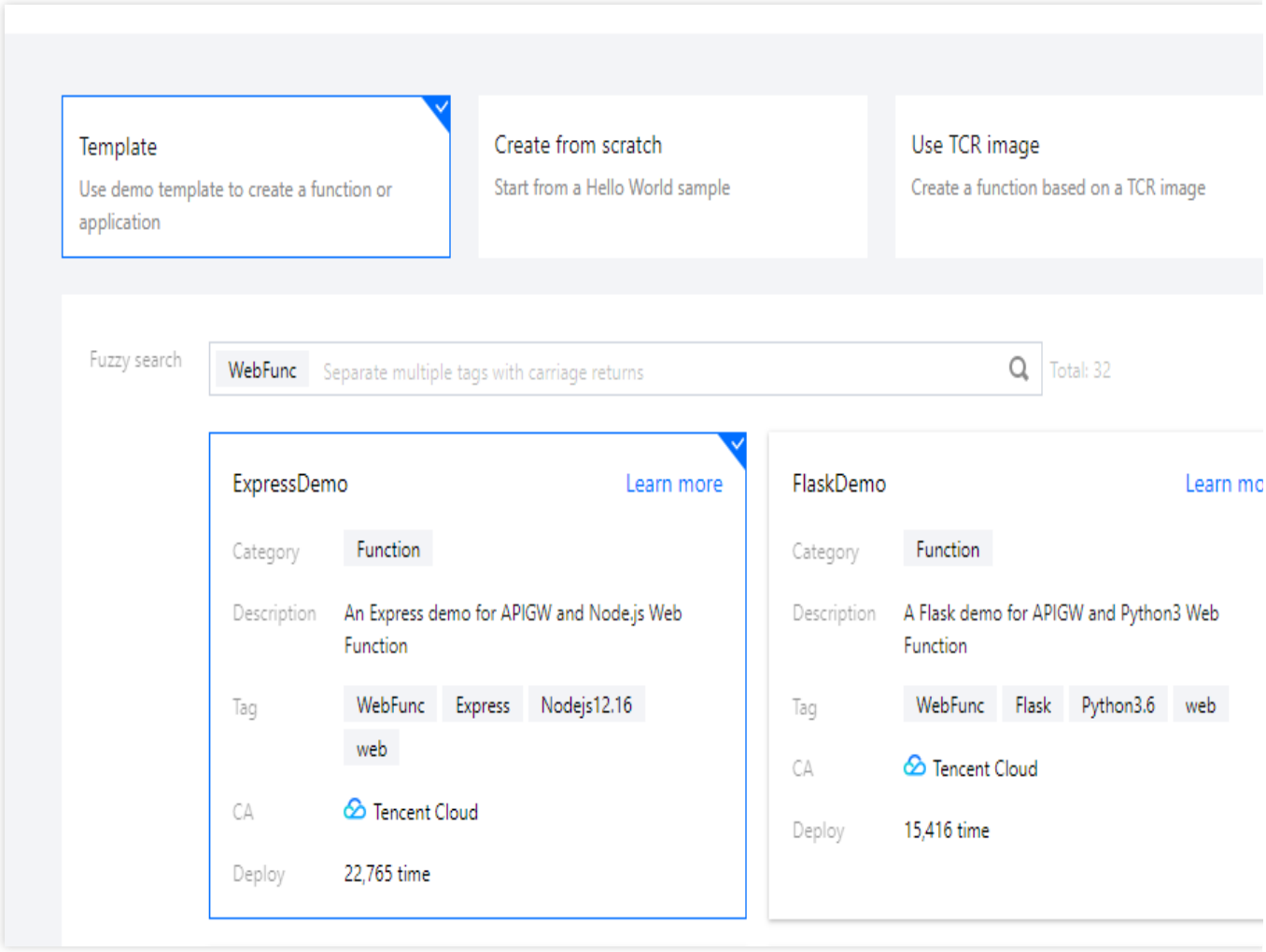
Prerequisites

Before using SCF, you need to sign up for a Tencent Cloud account and complete the identity verification first.

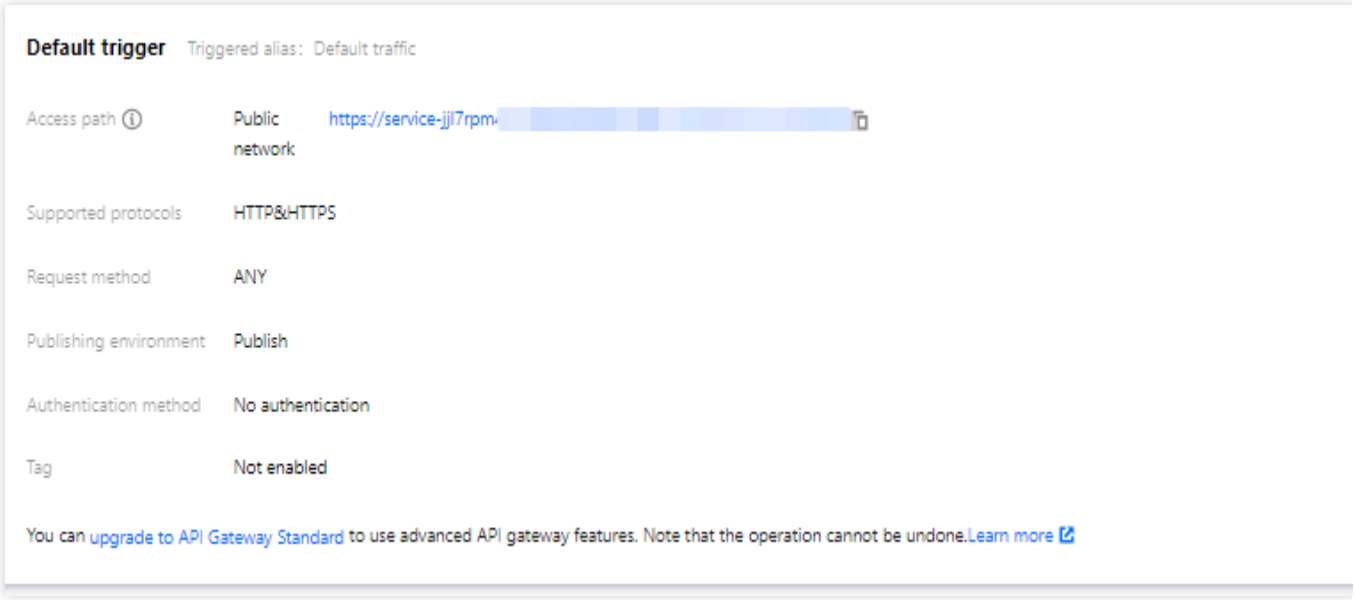
Directions

Template deployment: Quick deployment of Express project

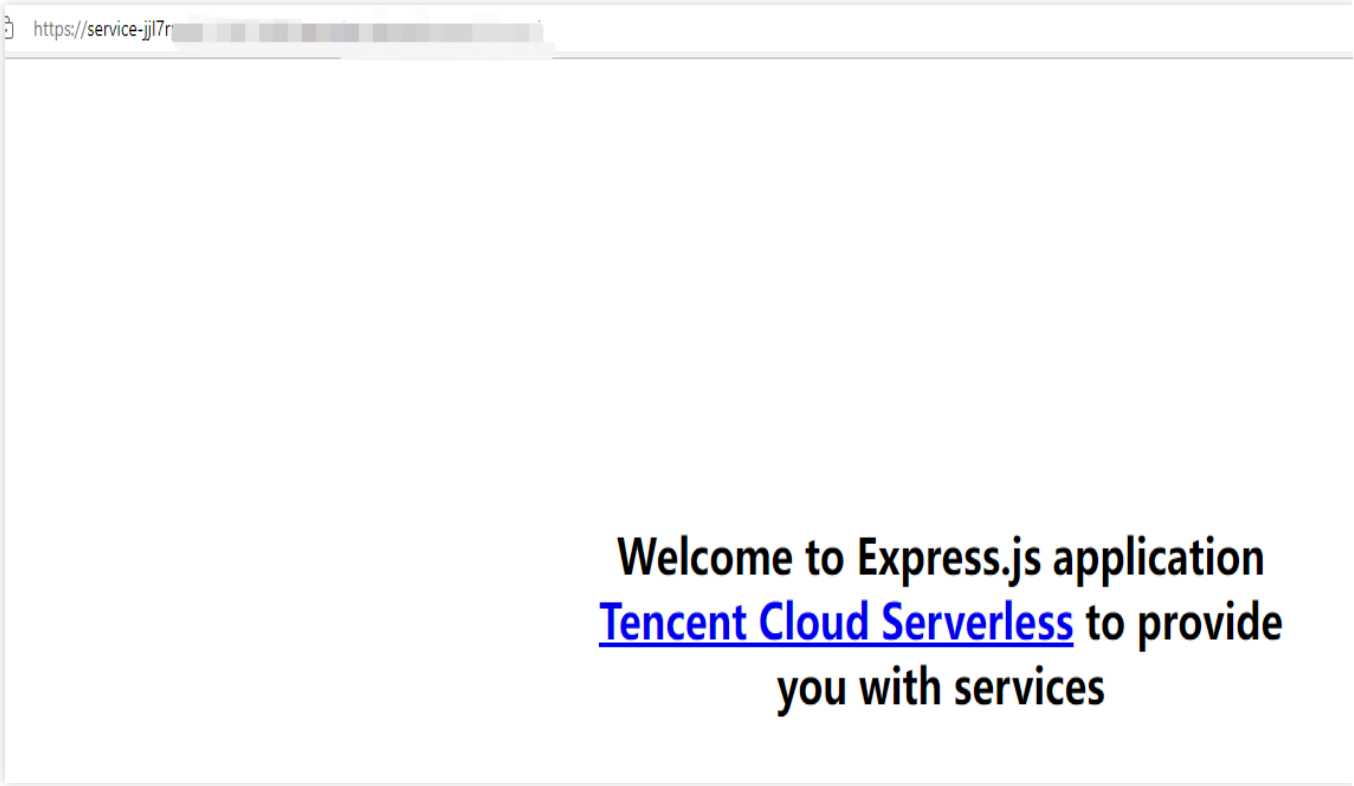
1. Log in to the SCF console and click **Functions** on the left sidebar.
2. Select the region and namespace where to create a function at the top of the page and click **Create** to enter the function creation process.
3. Select **Template**, enter `WebFunc` in the search box to filter all HTTP-triggered function templates, select **ExpressDemo**, and click **Next** as shown below:



4. On the **Create** page, you can view and modify the specific configuration information of the template project.
5. Click **Complete**. After creating the HTTP-triggered function, you can view its basic information on the **Function management** page.
6. Click **Trigger management** on the left to view the access path and access your deployed Express project as shown below:



7. Click the access path URL to access the Express project as shown below:



Custom deployment: Quick migration of local project to cloud

Prerequisites

The Node.js runtime environment has been installed locally.

Local development

1. Run the following command to install the Express framework and `express-generator` scaffold and initialize the sample Express project.

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

2. Run the following command to enter the project directory and install the required dependencies:

```
cd WebApp
npm install
```

3. After the installation is completed, run the following command to directly start the service locally.

```
npm start
```

4. Visit `http://localhost:3000` in a browser, and you can access the sample Express 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` bootstrap file.
The detailed directions are as follows:
1. In the sample Express project, you can specify the listening address and port through the environment variable in the `./bin/www` file. If you don't specify it, port **3000** will be listened on by default as shown below:

```
/**
 * 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` bootstrap file in the project root directory and add the following content to it (which is used to configure environment variables and start services):

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

3. After the creation is completed, 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 it to start normally. Below is the sample code:

```
chmod 777 scf_bootstrap
```

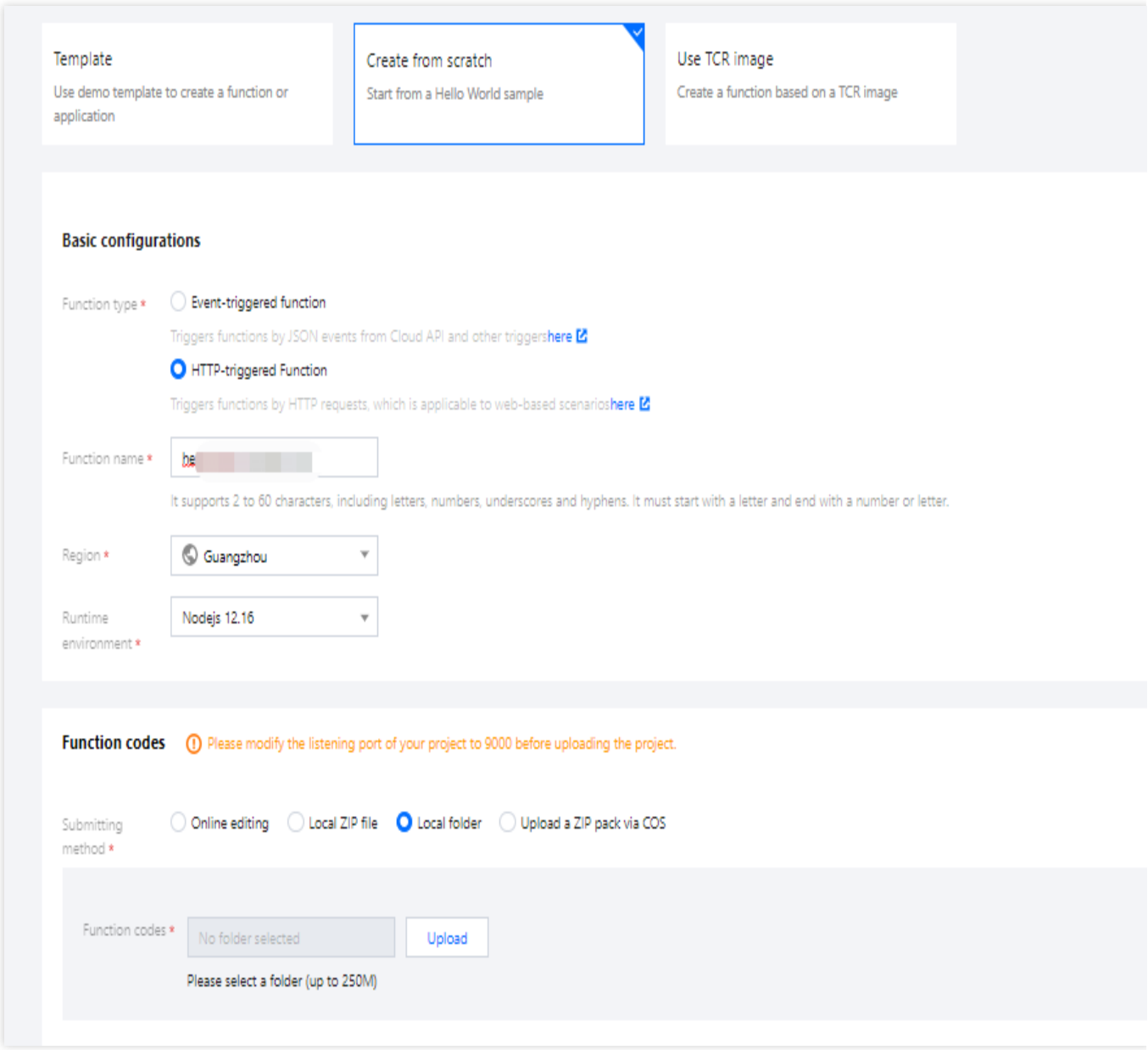
4. After the local configuration is completed, run the following command to start the file (with execution in the `scf_bootstrap` directory as an example) and make sure that your service can be normally started locally.

```
./scf_bootstrap
```

5. Log in to the SCF console and click **Functions** on the left sidebar.

6. Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.

7. Select **Create from scratch** and configure the options as prompted as shown below:



Function type: Select **HTTP-triggered function**.

Function Name: Enter the name of your function.

Region: Enter your function deployment region, which is **Guangzhou** by default.

Runtime environment: Select **Nodejs 12.16**.

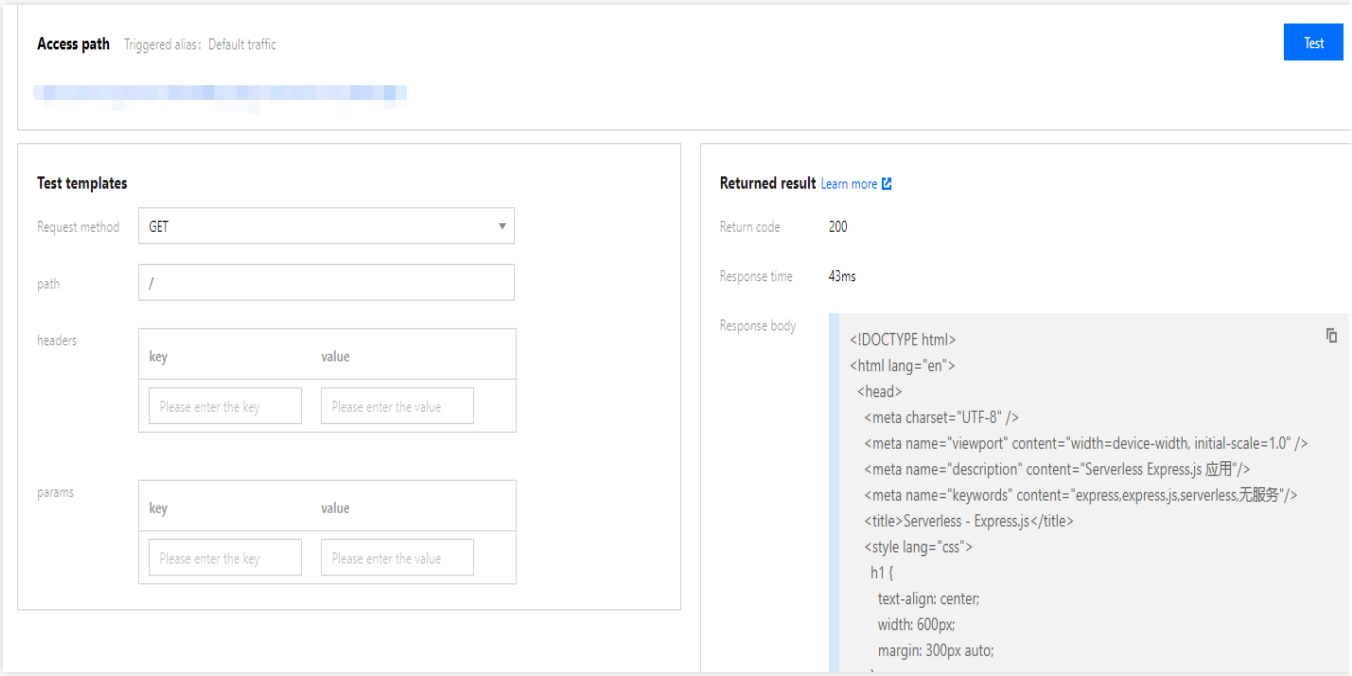
Submitting method: Select **Local folder** and upload your local project.

-**Function codes**: Select the specific local folder where the function code is.

8. Click **Complete**.

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 as shown below:



Quickly Deploying Flask Framework

Last updated：2025-02-12 11:08:57

Overview

This document describes how to quickly deploy a Flask business to the cloud through an SCF web function.

Note:

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

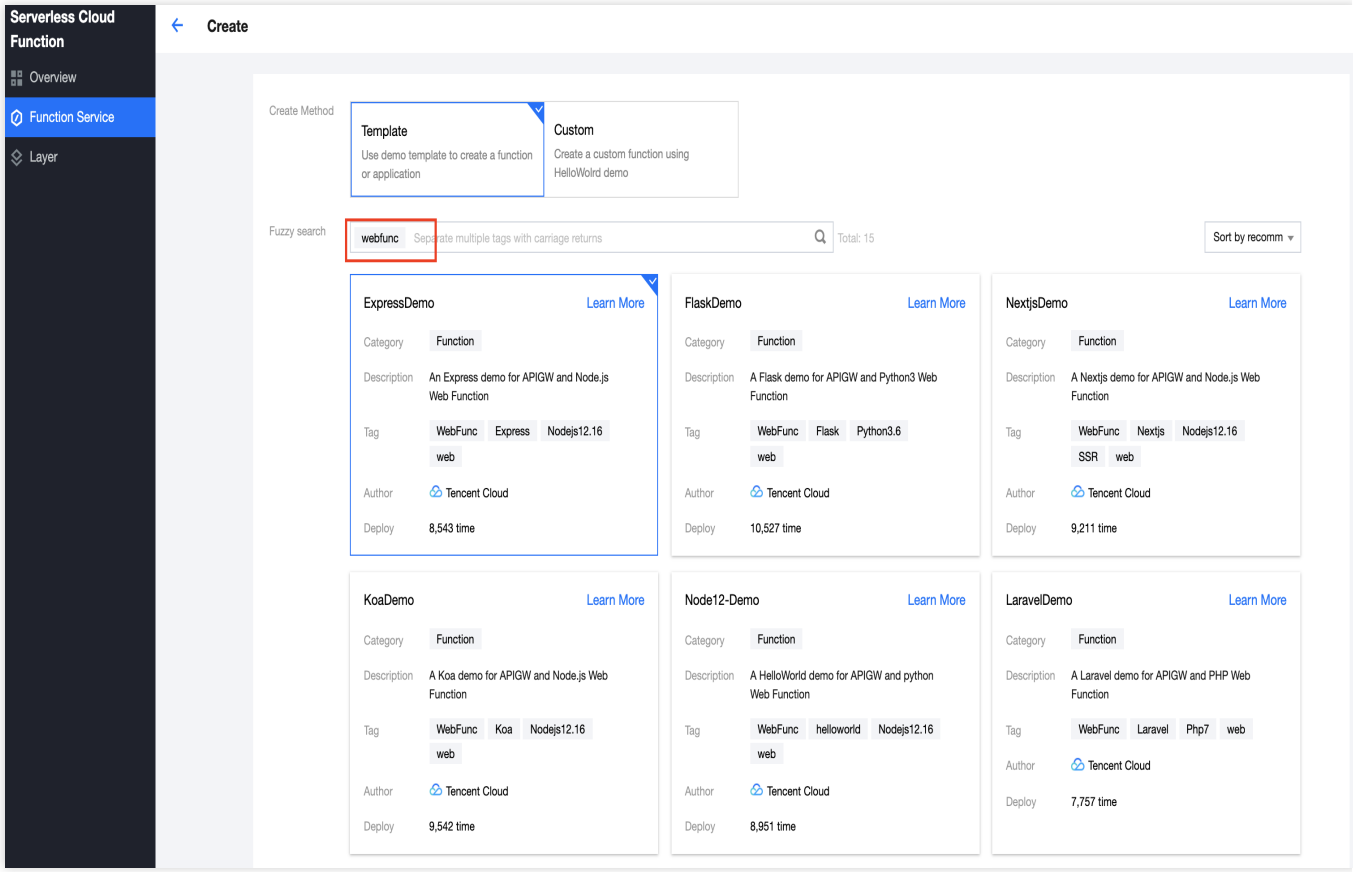
Prerequisites

Before using SCF, you need to sign up for a Tencent Cloud account and complete the identity verification first.

Directions

Template deployment - quick deployment of Flask project

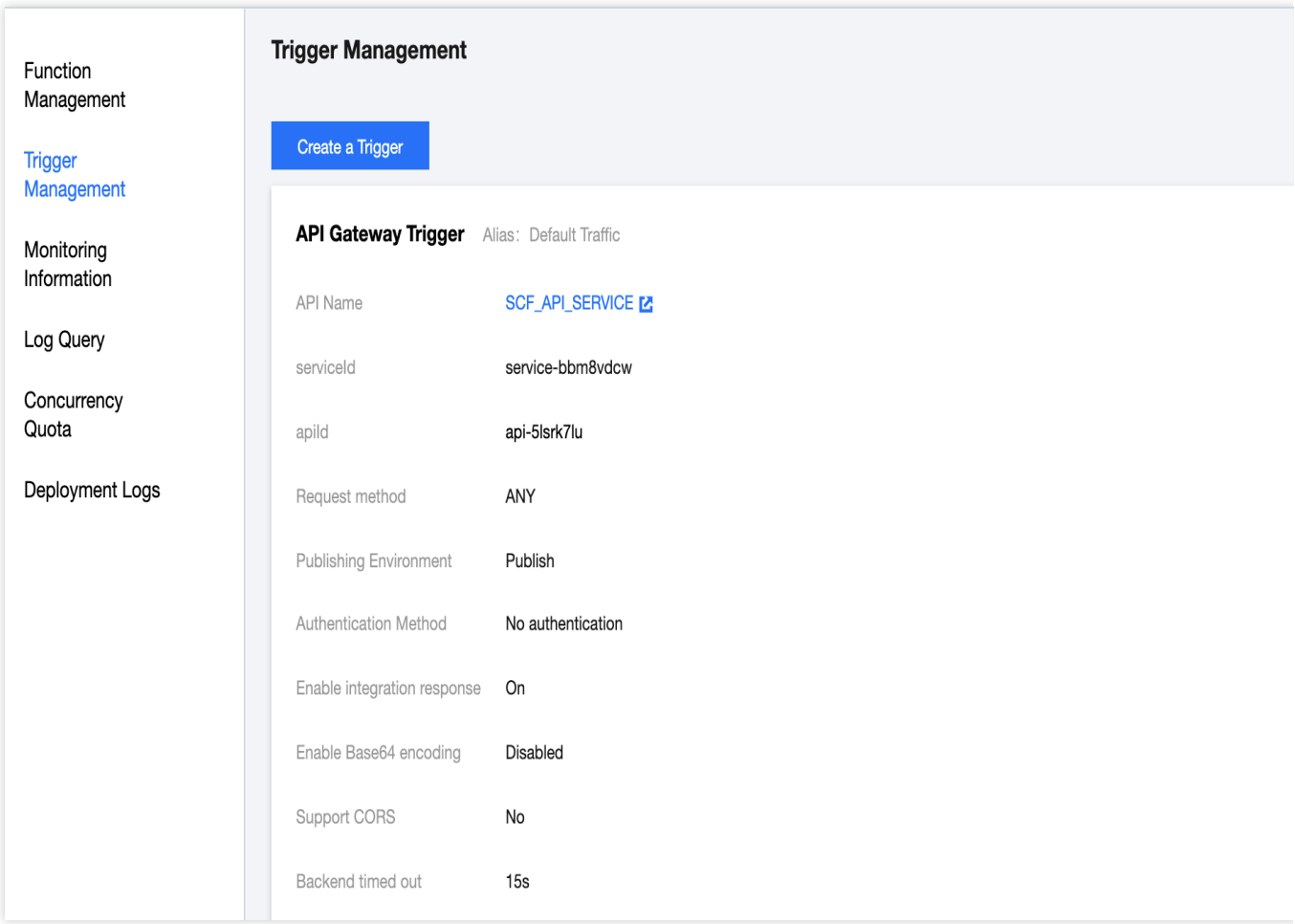
1. Log in to the SCF console and click **Functions** on the left sidebar.
2. Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.
3. Select **Template**, enter `WebFunc` in the search box to filter all HTTP-triggered function templates, select **FlaskDemo**, and click **Next** as shown below:



4. On the **Create** page, you can view and modify the specific configuration information of the template project.

5. Click **Complete**. After creating the HTTP-triggered function, you can view its basic information on the **Function management** page.

6. You can access the deployed Flask project at the access path URL generated automatically. Click **Trigger management** on the left to view the access path as shown below:



7. Click the access path URL to access the Flask project.

Custom deployment - quick migration of local project to cloud

Local development

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

```
pip install Flask
```

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

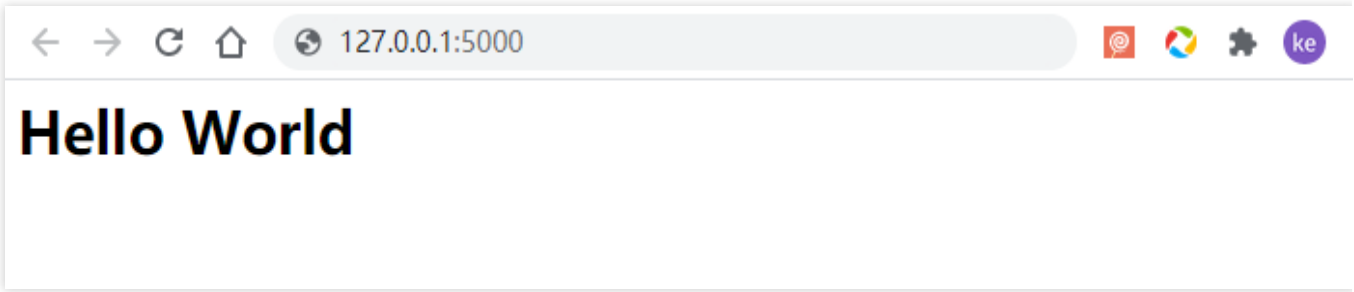
In the project directory, create the `app.py` file to implement the Hello World application. Below is the sample code:

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

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

if __name__ == '__main__':
   app.run()
```

3. Run the `python3 app.py` command locally to start the `app.py` file. Below is the sample code:

```
$ 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 -
```

4. Visit `http://127.0.0.1:5000` in a browser, and you can access the sample Flask project locally as shown below:



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 a web function. The steps of project transformation for Flask are as follows:

1. **Install dependencies**

1.1 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. Create the `requirements.txt` file first with the following content:

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

Note:

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

1.2 Run the following command to install:

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

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` as shown below:

```
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. **Add the `scf_bootstrap` bootstrap file**

3.1 Create the `scf_bootstrap` bootstrap file in the project root directory and add the following content to it (which is used to configure environment variables, specify service startup commands, and make sure that your service can be started normally through this file):

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

3.2 After the creation is completed, 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 it to start normally. Below is the sample code:

```
chmod 777 scf_bootstrap
```

Note:

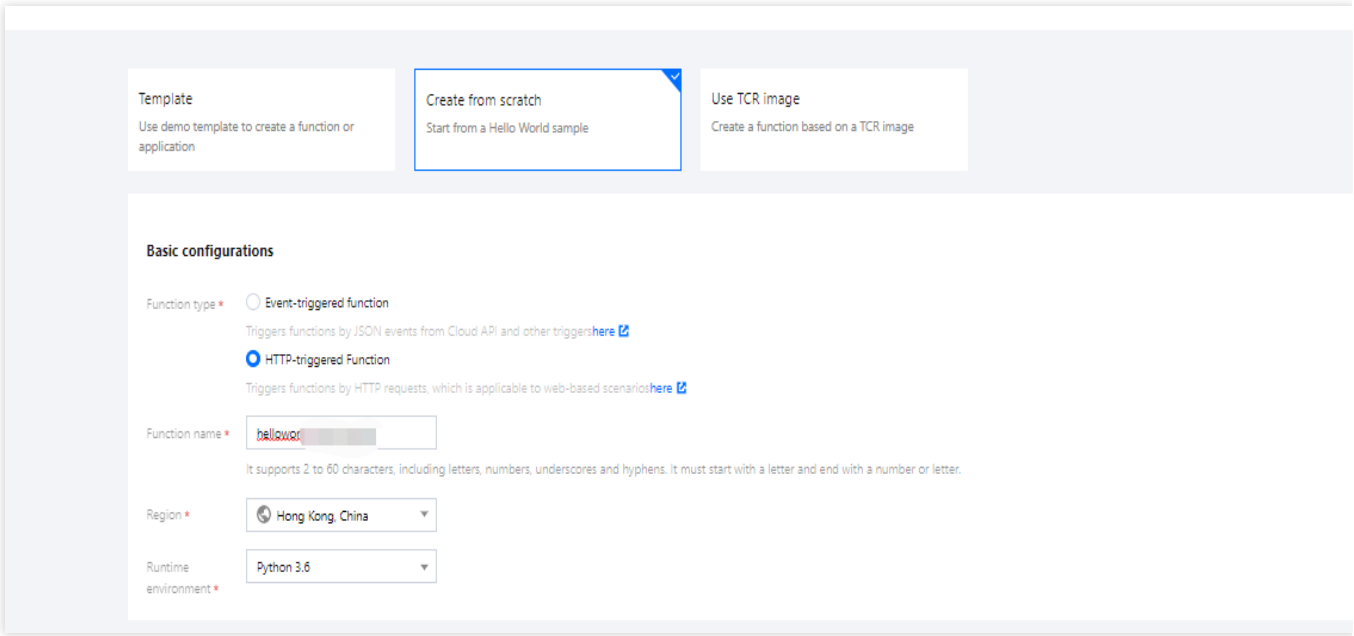
In the SCF environment, only files in the `/tmp` directory are readable/writable. We recommend you select `/tmp` when outputting files. If you select other directories, write will fail due to the lack of permissions.

If you want to output environment variables in the log, you need to add the `-u` parameter before the startup command, such as `python -u app.py` .

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

```
./scf_bootstrap
```

5. Log in to the SCF console and click **Functions** on the left sidebar.
6. Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.
7. Select **Create from scratch** and configure the options as prompted as shown below:



Function type: Select **HTTP-triggered function**.
Function name: Enter the name of your function.
Region: Enter your function deployment region.
Deployment mode: Select **Code deployment** and upload your local project.
Runtime Environment: Select **Python 3.6**.
8. Click **Complete**.

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 Koa Framework

Last updated：2025-02-12 11:08:57

Overview

This document describes how to quickly deploy a local Koa project to the cloud through an HTTP-triggered function.

Note:

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

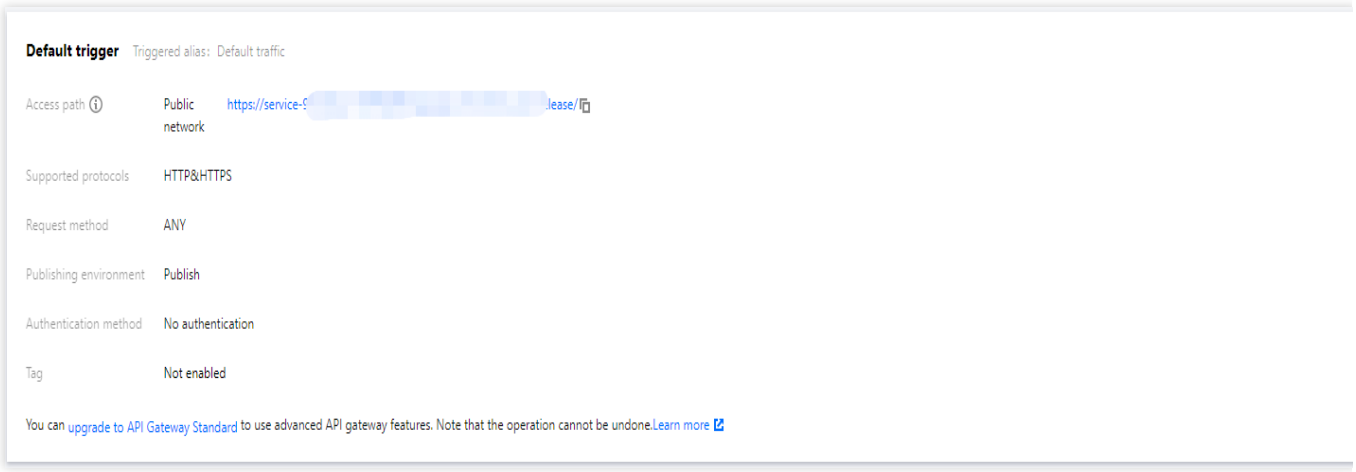
Prerequisites

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

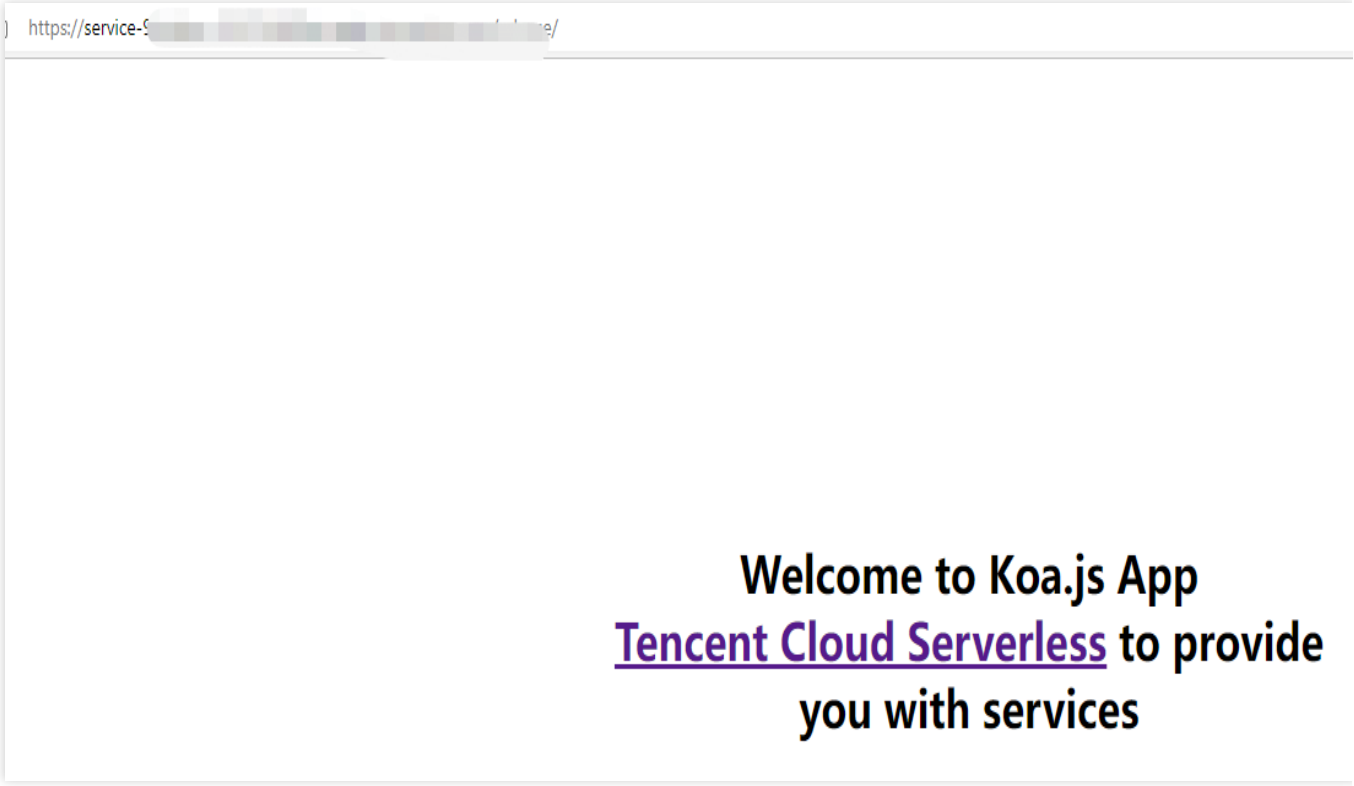
Directions

Template deployment: Quick deployment of Koa project

1. Log in to the SCF console and click **Functions** on the left sidebar.
2. Select the region and namespace where to create a function at the top of the page and click **Create** to enter the function creation process.
3. Select **Template**, enter `koa` in the search box to filter function templates, select the **Koa template**, and click **Next**.
4. On the **Create** page, you can view and modify the specific configuration information of the template project.
5. Click **Complete**. After creating the HTTP-triggered function, you can view its basic information on the **Function management** page.
6. Click **Trigger management** on the left to view the access path and access your deployed Koa project as shown below:



7. Click the access path URL to access the Koa project as shown below:



Custom deployment: Quick migration of local project to cloud

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` bootstrap file.

The detailed directions are as follows:

1. In the sample Koa project, change the listening port to `9000` as shown below:

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

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

3. After the creation is completed, 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 it to start normally. Below is the sample code:

```
chmod 777 scf_bootstrap
```

4. Log in to the SCF console and click **Functions** on the left sidebar.
5. Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.
6. Select **Create from scratch** and configure the options as prompted:
Function type: Select **HTTP-triggered function**.
Function name: Enter the name of your function.
Region: Enter your function deployment region, which is **Guangzhou** by default.
Runtime environment: Select **Nodejs 12.16**.
Submitting method: Select **Local folder** and upload your local project.
Function codes: Select the specific local folder where the function code is.
7. Click **Complete**.

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 Laravel Framework

Last updated：2025-02-12 11:08:57

Overview

This document describes how to use an HTTP-triggered 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, see Deploying Framework on Command Line.

Prerequisites

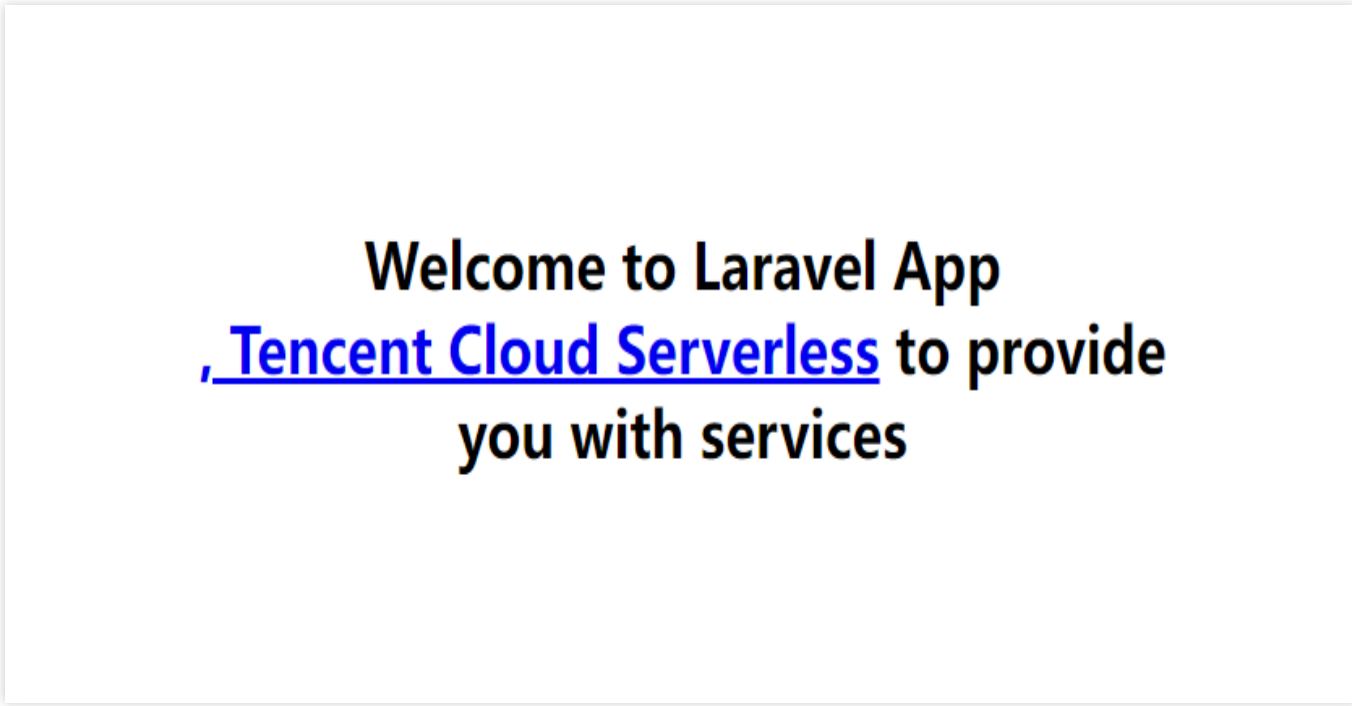
Before using SCF, you need to sign up for a Tencent Cloud account and complete identity verification first.

Directions

Template deployment: Quick deployment of Laravel project

1. Log in to the SCF console and click **Functions** on the left sidebar.
2. Select the region and namespace where to create a function at the top of the page and click **Create** to enter the function creation process.
3. Select **Template**, enter `WebFunc` in the search box to filter all HTTP-triggered function templates, select the **Laravel template**, and click **Next**.
4. On the **Create** page, you can view and modify the specific configuration information of the template project.
5. Click **Complete**. After creating the HTTP-triggered function, you can view its basic information on the **Function management** page.
6. Click **Trigger management** on the left to view the access path and access your deployed Laravel project as shown below:

7. Click the access path URL to access the Laravel project as shown below:



Custom deployment: Quick migration of local project to cloud

Local development

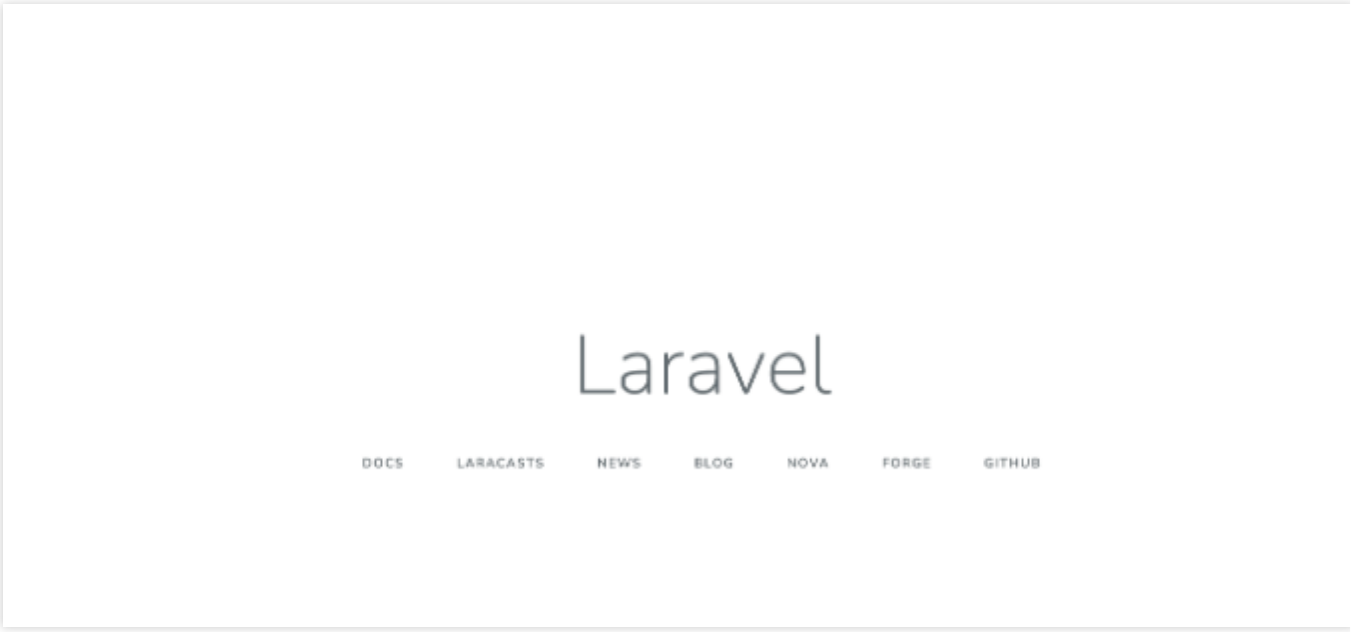
1. Refer to Getting Started on macOS to set up the Laravel development environment locally.
2. Create the sample Laravel project locally. Enter the project directory and run the following command to initialize it.

```
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

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 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:

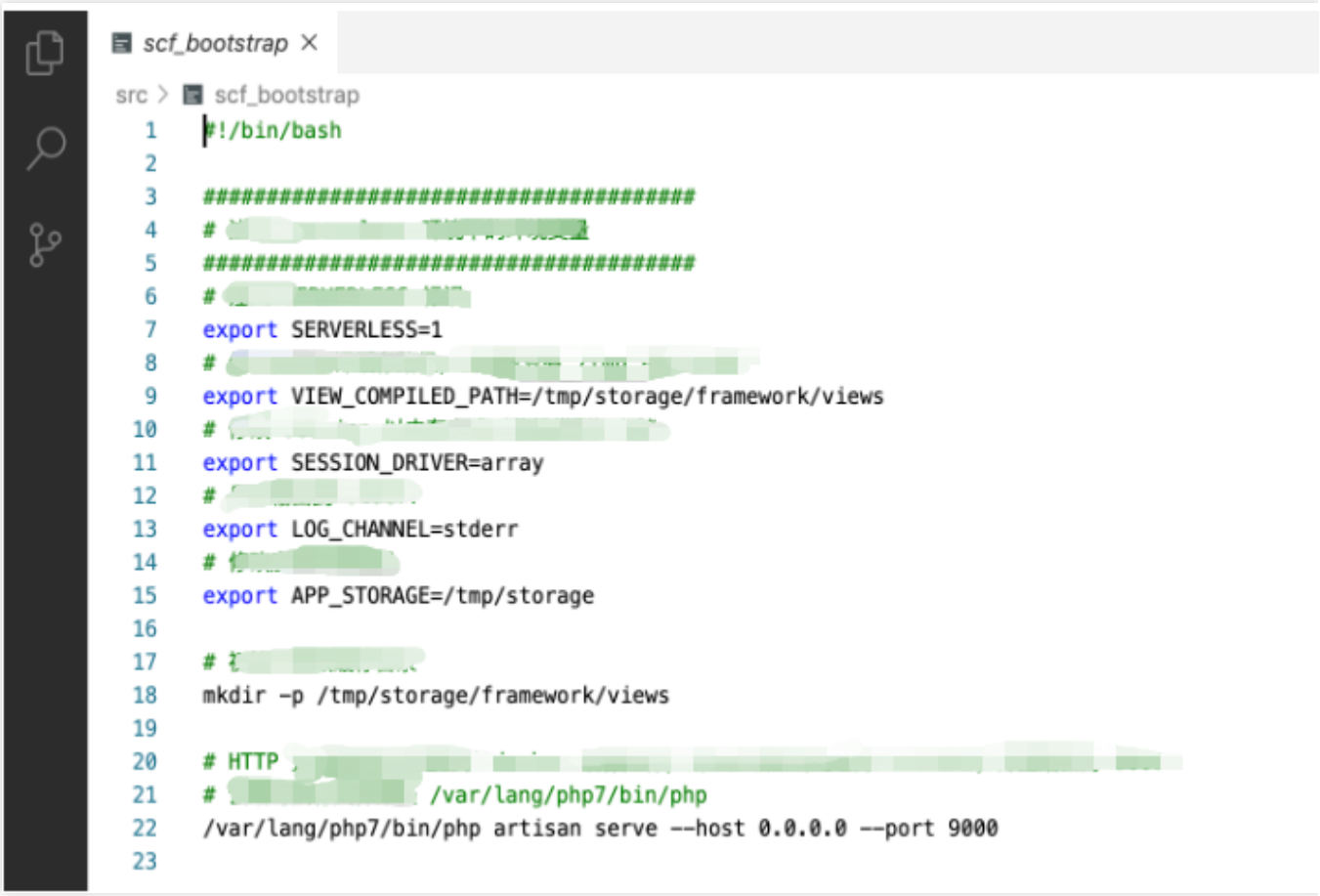
```
#!/bin/bash

# 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 HTTP-triggered 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:



4. **Deploy Laravel**

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:

4.1 Log in to the SCF console and click **Functions** on the left sidebar.

4.2 Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.

4.3 Select **Create from scratch** and configure the options as prompted:

Function type: Select **HTTP-triggered function**.

Function name: Enter the name of your function.

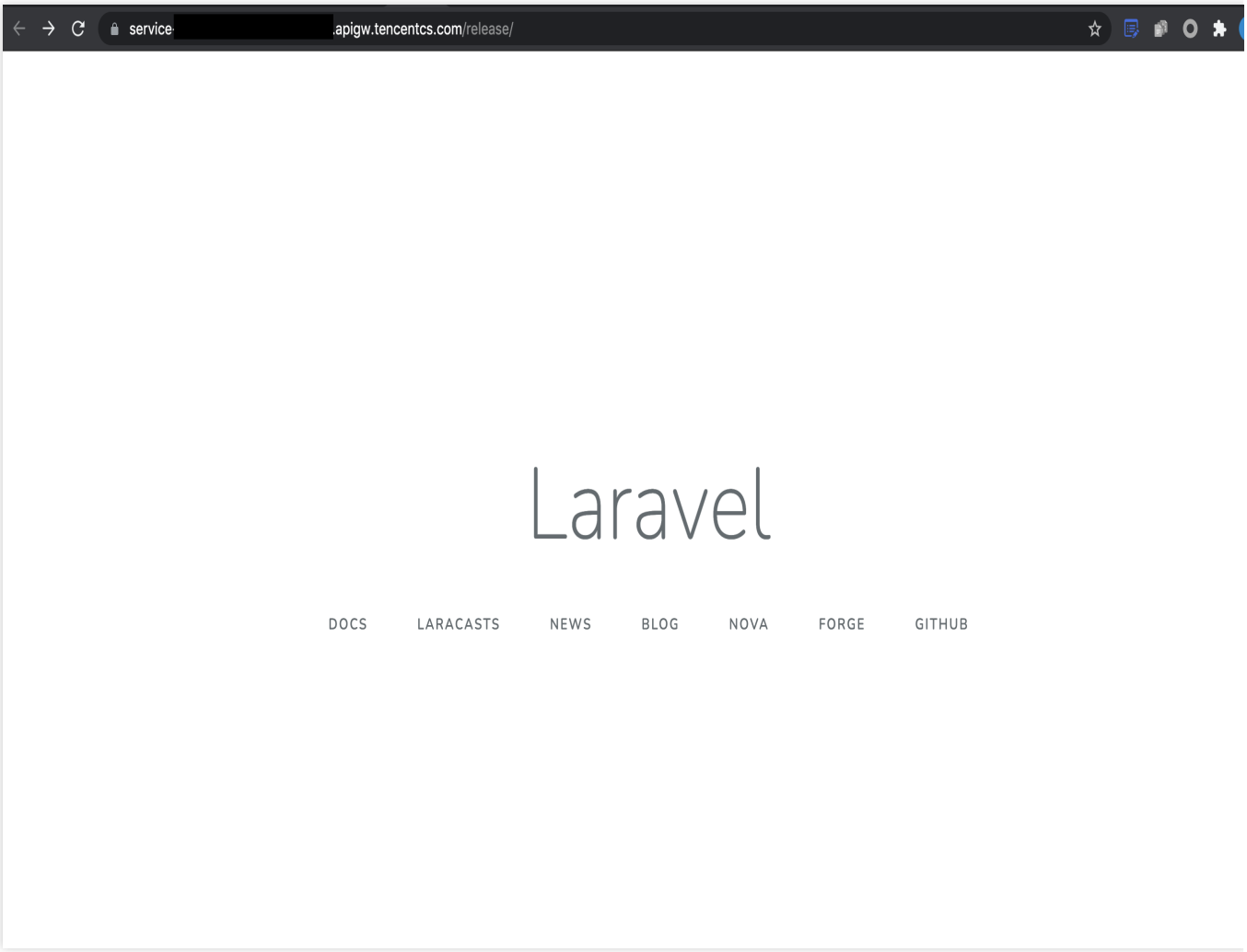
Region: Enter your function deployment region, such as Chengdu.

Runtime environment: Select **Php7**.

Submitting method: Select **Local folder** and upload your local project.

-**Function codes**: Select the specific local folder where the function code is.

4.4 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：2023-02-02 11:00:57

Overview

This document describes how to quickly deploy a local Nest.js project to the cloud through an HTTP-triggered function.

> Note：
> This document mainly describes how to deploy in the console. You can also complete the deployment on the command line. For more information, see Deploying Framework on Command Line.

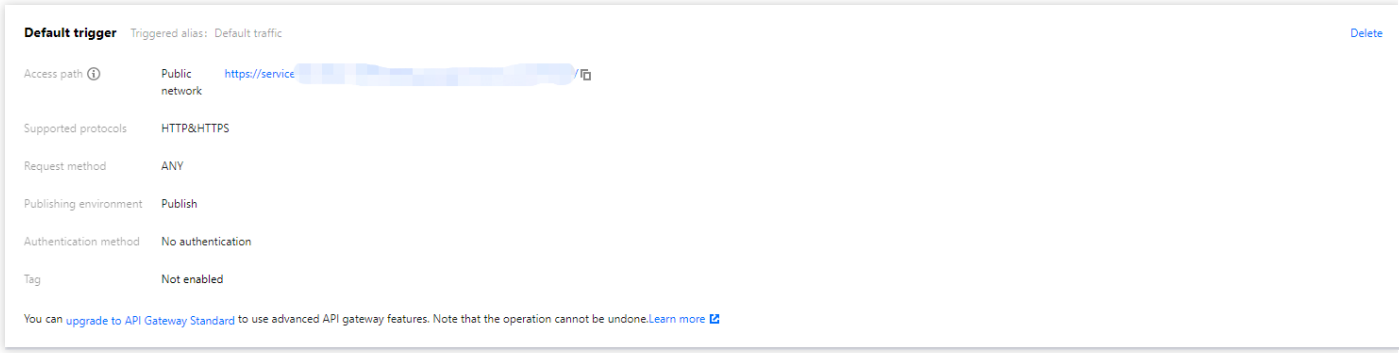
Prerequisites

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

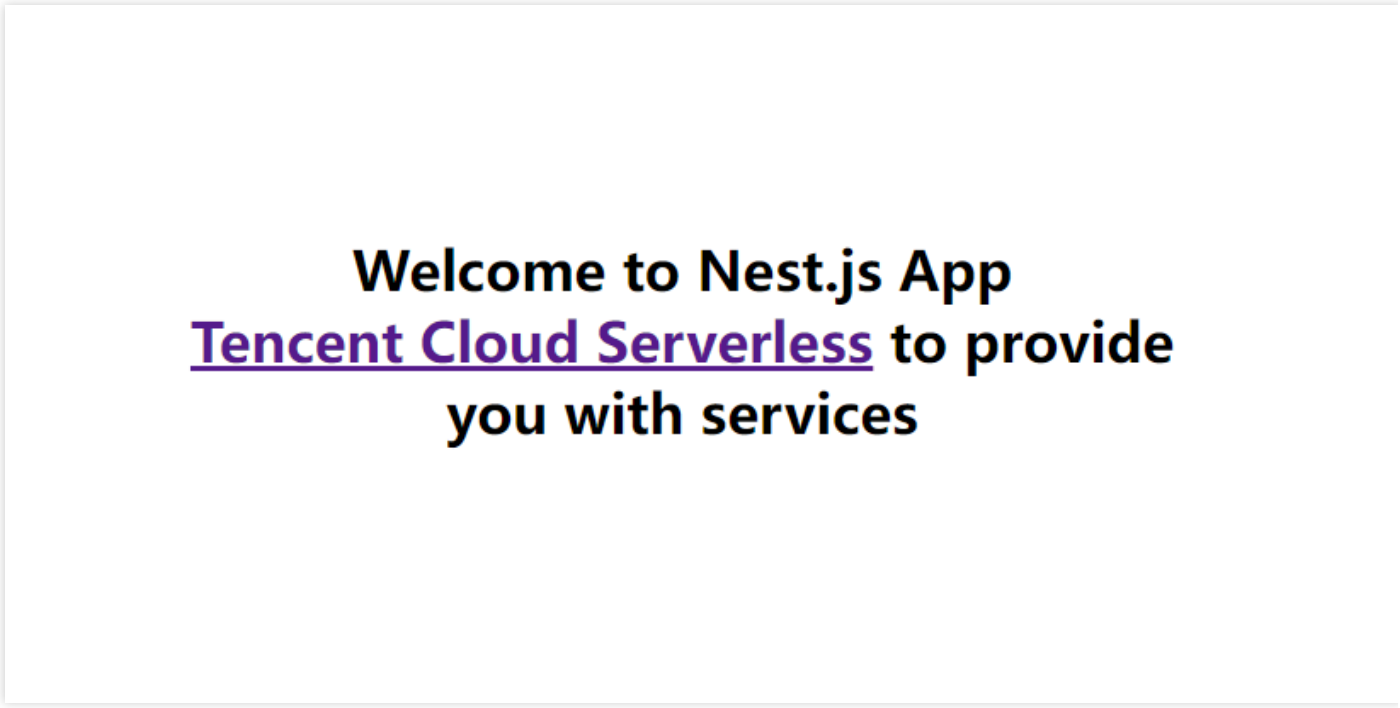
Directions

Template deployment: Quick deployment of Nest.js project

1. Log in to the SCF console and click **Functions** on the left sidebar.
2. Select the region and namespace where to create a function at the top of the page and click **Create** to enter the function creation process.
3. Select **Template**, enter `nest` in the search box to filter function templates, select the **Nest template**, and click **Next**.
4. On the **Create** page, you can view and modify the specific configuration information of the template project.
5. Click **Complete**. After creating the HTTP-triggered function, you can view its basic information on the **Function management** page.
6. Click **Trigger management** on the left to view the access path and access your deployed Nest.js project as shown below:



7. Click the access path URL to access the Nest.js project as shown below:



Custom deployment: Quick migration of local project to cloud

Prerequisites

The Node.js runtime environment has been installed locally.

Local development

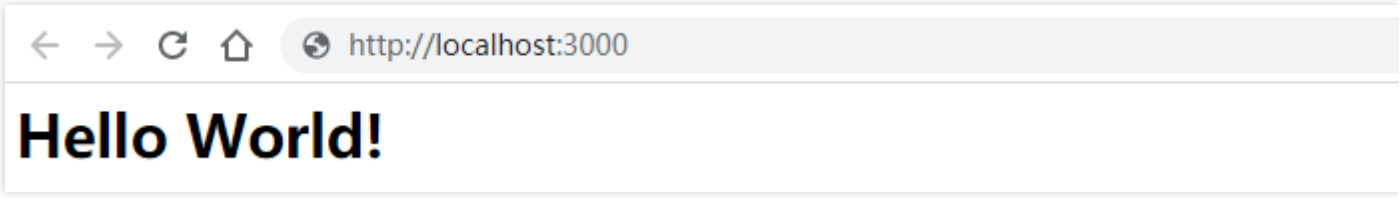
1. Refer to First steps to initialize your Nest.js project:

```
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 as shown below:



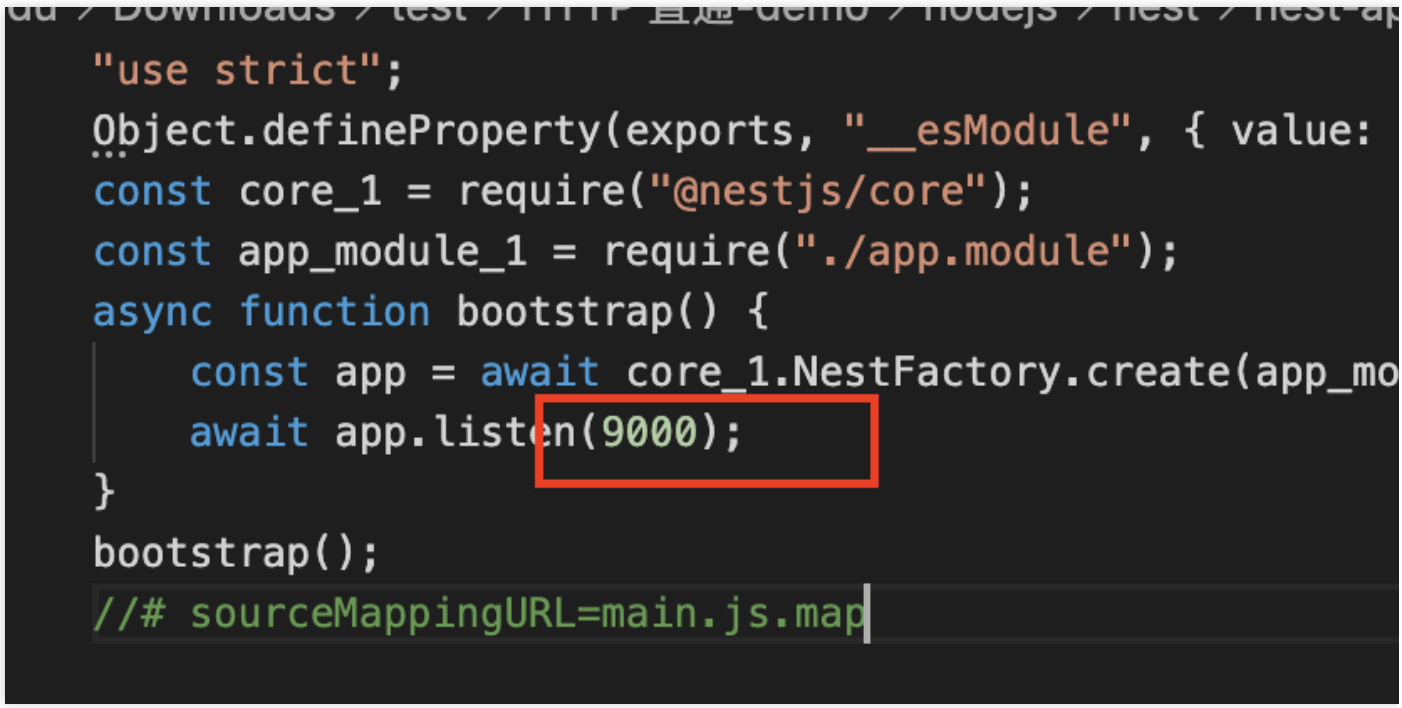
Deployment in cloud

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

The detailed directions are as follows:

1. Modify the bootstrap file `./dist/main.js` to change the listening port to `9000` as shown below:



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

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

Note
- Here is only a sample bootstrap file. Adjust the specific operations 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.

3. After the creation is completed, 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 it to start normally. Below is the sample code:

```
chmod 777 scf_bootstrap
```

4. Log in to the SCF console and click **Functions** on the left sidebar.
5. Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.
6. Select **Create from scratch** and configure the options as prompted:
   - **Function type**: Select **HTTP-triggered function**.
   - **Function name**: Enter the name of your function.
   - **Region**: Enter your function deployment region, which is **Guangzhou** by default.
   - **Runtime environment**: Select **Nodejs 12.16**.
   - **Submitting method**: Select **Local folder** and upload your local project.
   - **Function codes**: Select the specific local folder where the function code is.
7. Click **Complete**.

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 Next.js Framework

Last updated：2023-02-02 11:00:57

Overview

This document describes how to quickly deploy a local Next.js SSR project to the cloud through an HTTP-triggered function.

> Note：
> This document mainly describes how to deploy in the console. You can also complete the deployment on the command line. For more information, see Deploying Framework on Command Line.

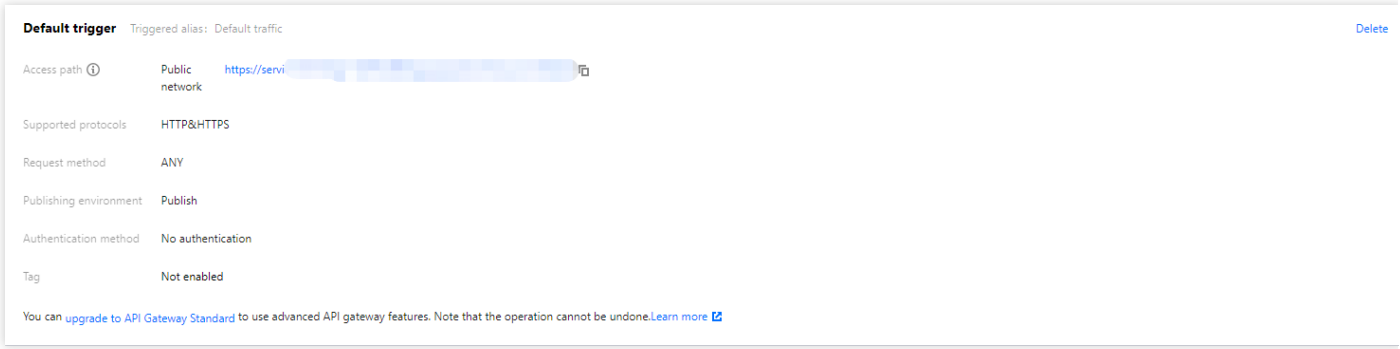
Prerequisites

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

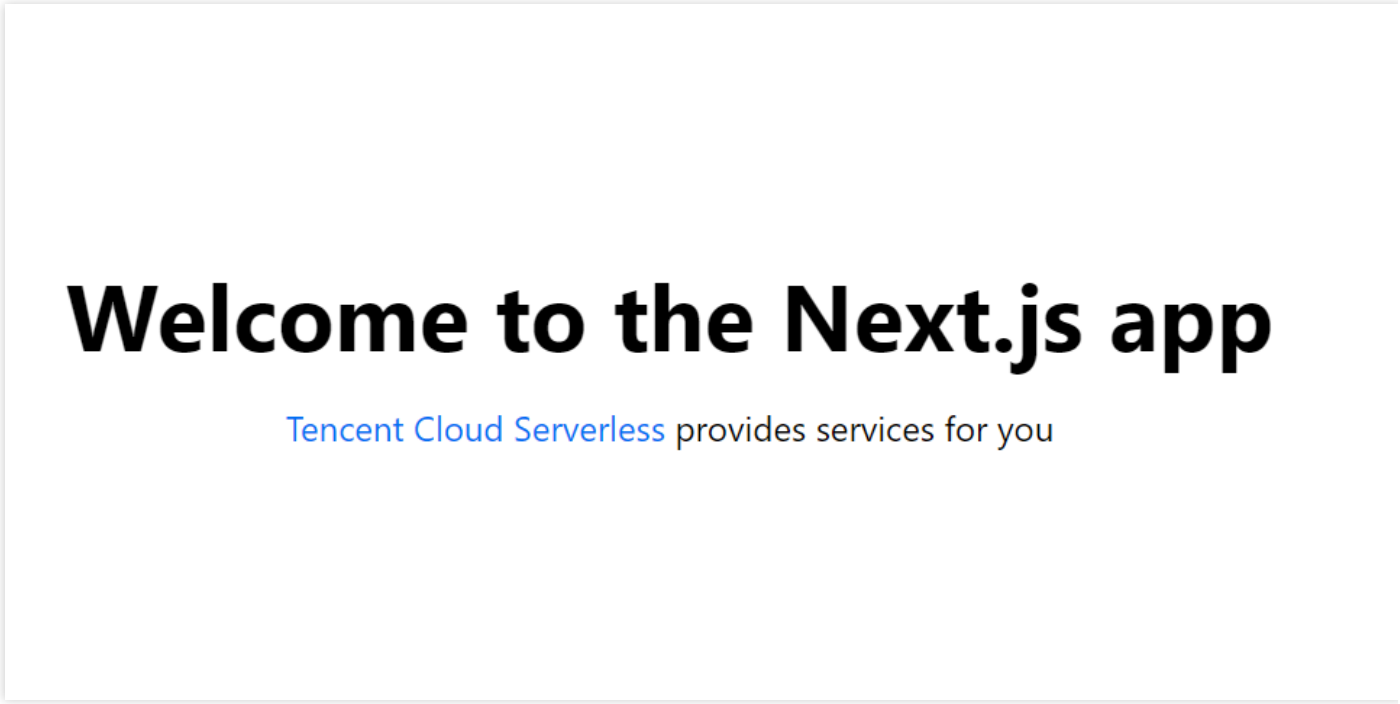
Directions

Template deployment: Quick deployment of Next.js project

1. Log in to the SCF console and click **Functions** on the left sidebar.
2. Select the region and namespace where to create a function at the top of the page and click **Create** to enter the function creation process.
3. Select **Template**, enter `webfunc` in the search box to filter function templates, select the **Next.js template**, and click **Next**.
4. On the **Create** page, you can view and modify the specific configuration information of the template project.
5. Click **Complete**. After creating the HTTP-triggered function, you can view its basic information on the **Function management** page.
6. Click **Trigger management** on the left to view the access path and access your deployed Next.js project as shown below:



7. Click the access path URL to access the Next.js project as shown below:



> Note：
> As the Next.js framework needs to be rebuilt before each deployment, be sure to update the code locally and run `build` again before deploying.

Custom deployment: Quick migration of local project to cloud

Prerequisites

The Node.js runtime environment has been installed locally.

Local development

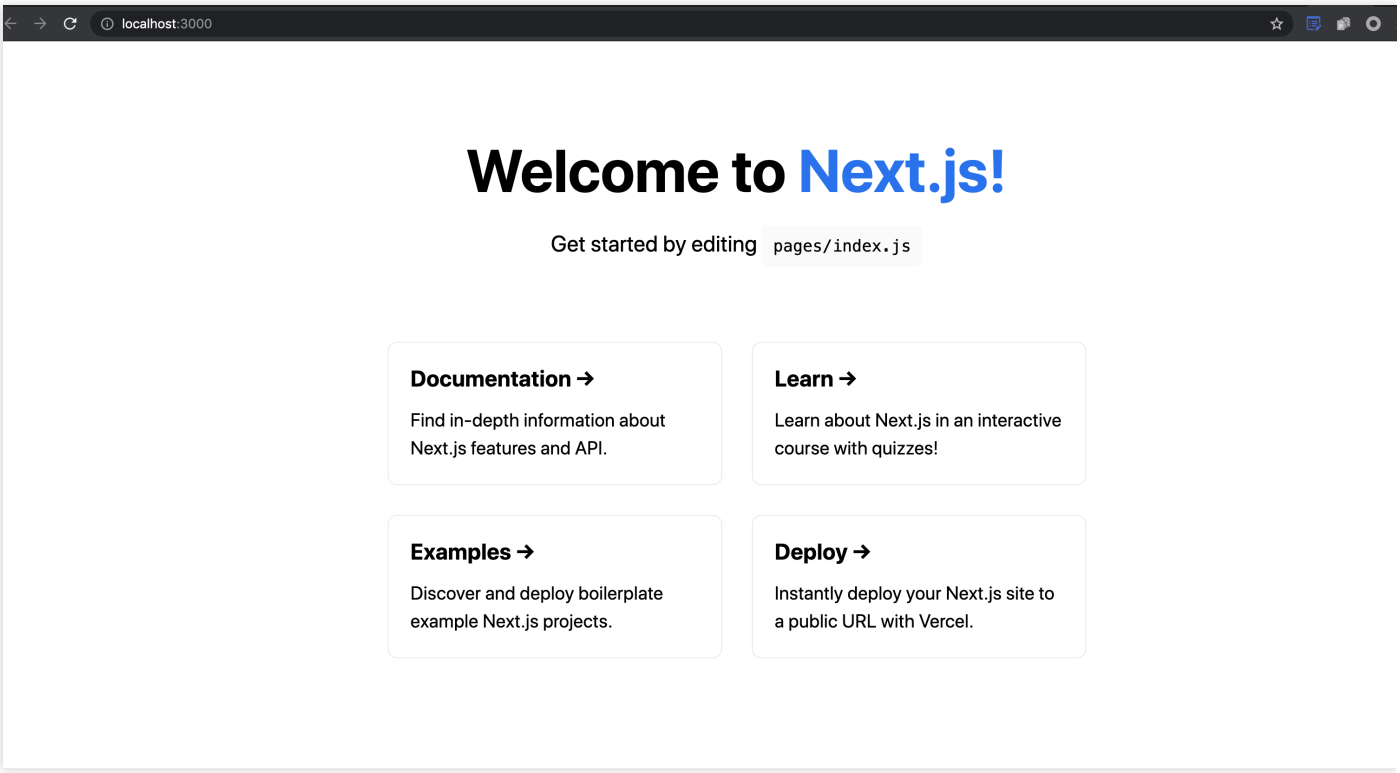
1. Refer to Getting Started to install and initialize your Next.js project:

```
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 as shown below:



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` bootstrap file.

The detailed directions are as follows:

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

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

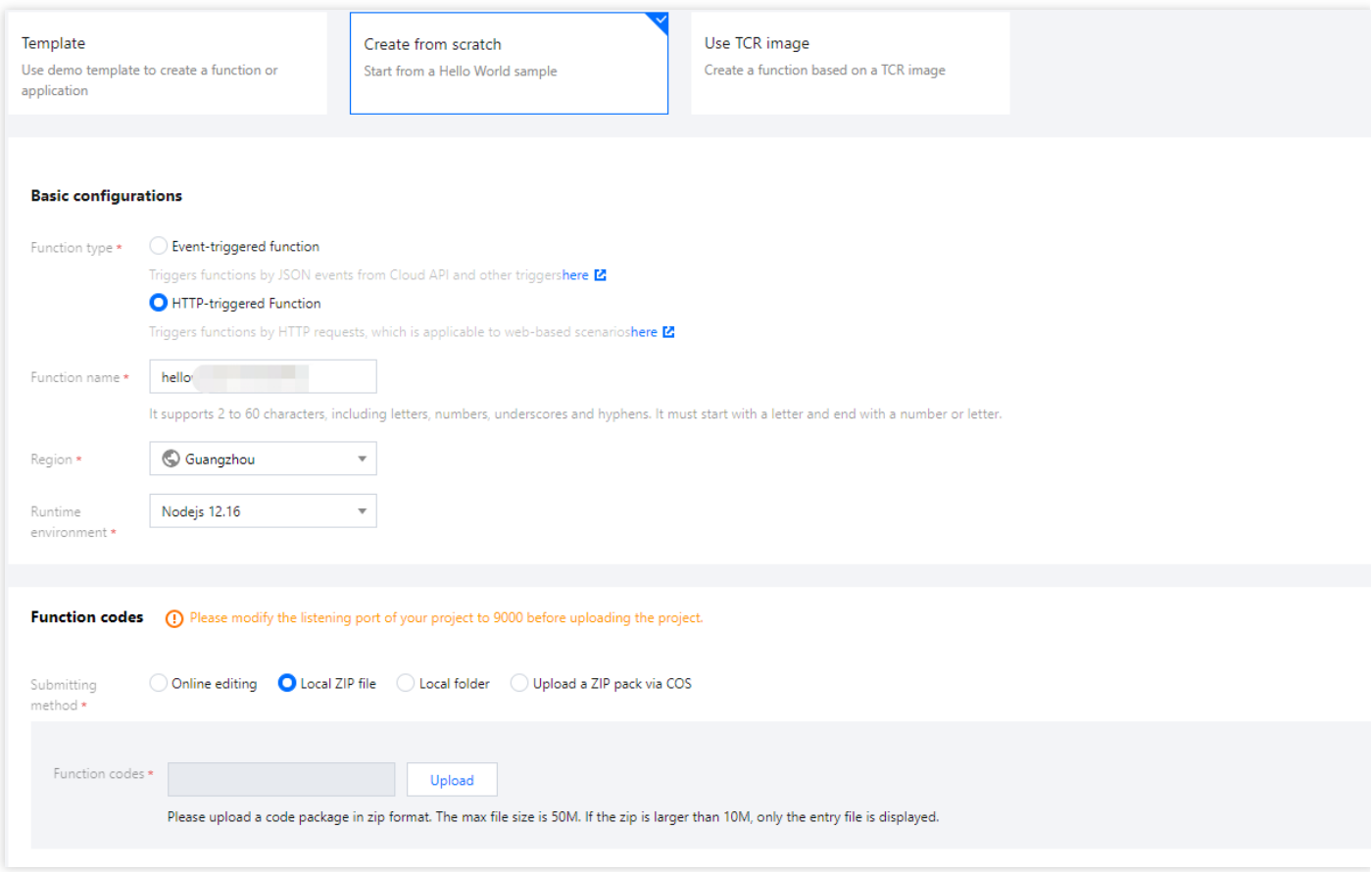
> Note
> - Here is only a sample bootstrap file. Adjust the specific operations according to your actual business scenario.
> - The sample uses the standard node environment path of SCF. When debugging locally, you need to change it to your local path.

2. After the creation is completed, 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 it to start normally. Below is the sample code:

```
chmod 777 scf_bootstrap
```

3. Log in to the SCF console and click **Functions** on the left sidebar.
4. Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.

5. Select **Create from scratch** and configure the options as prompted as shown below:



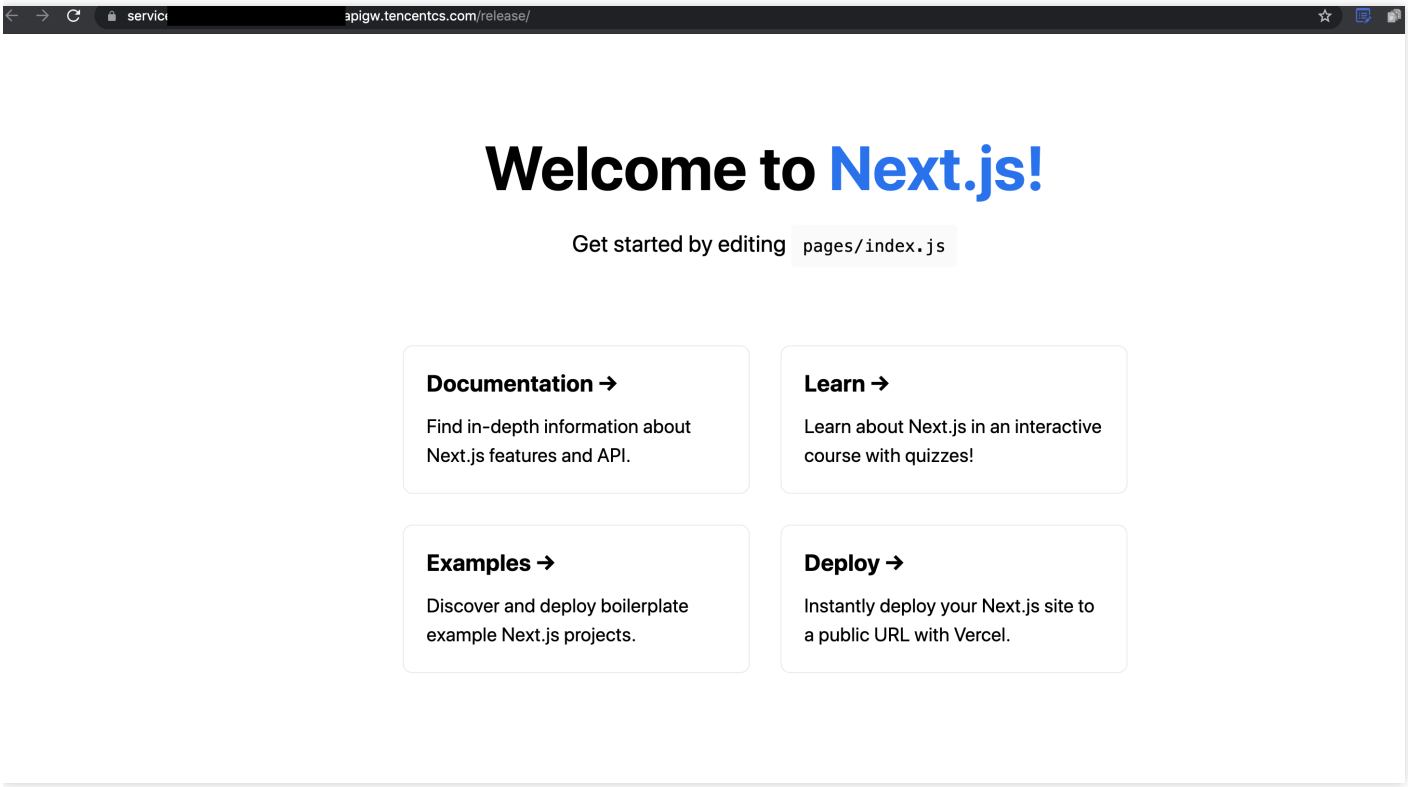
- **Function type**: Select **HTTP-triggered function**.
- **Function name**: Enter the name of your function.
- **Region**: Enter your function deployment region, which is **Guangzhou** by default.
- **Runtime environment**: Select **Nodejs 12.16**.
- **Submitting method**: Select **Local folder** and upload your local project.
- **Function codes**: Select the specific local folder where the function code is.

6. Click **Complete**.

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 Nuxt.js Framework

Last updated：2025-02-12 11:08:57

Overview

This document describes how to quickly deploy a local Nuxt.js SSR project to the cloud through an HTTP-triggered function.

Note:

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

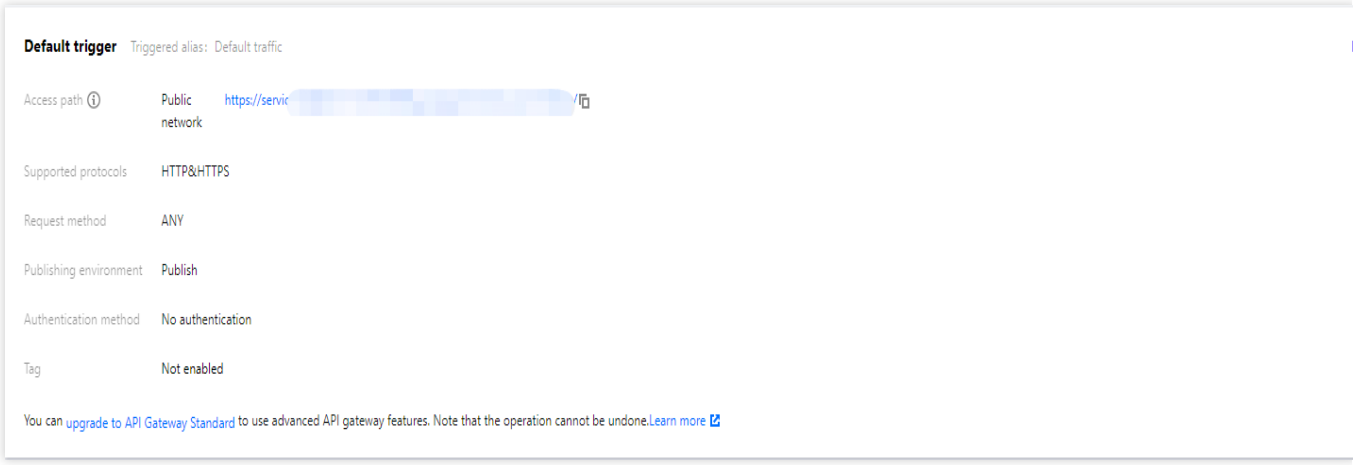
Prerequisites

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

Directions

Template deployment: Quick deployment of Nuxt.js project

1. Log in to the SCF console and click **Functions** on the left sidebar.
2. Select the region and namespace where to create a function at the top of the page and click **Create** to enter the function creation process.
3. Select **Template**, enter `webfunc` in the search box to filter function templates, select the **Nuxt.js template**, and click **Next**.
4. On the **Create** page, you can view and modify the specific configuration information of the template project.
5. Click **Complete**. After creating the HTTP-triggered function, you can view its basic information on the **Function management** page.
6. Click **Trigger management** on the left to view the access path and access your deployed Nuxt.js project as shown below:



7. Click the access path URL to access the Nuxt.js project as shown below:

Note:

As the Nuxt.js framework needs to be rebuilt before each deployment, be sure to update the code locally and run `build` again before deploying.

Custom deployment: Quick migration of local project to cloud

Prerequisites

The Node.js runtime environment has been installed locally.

Local development

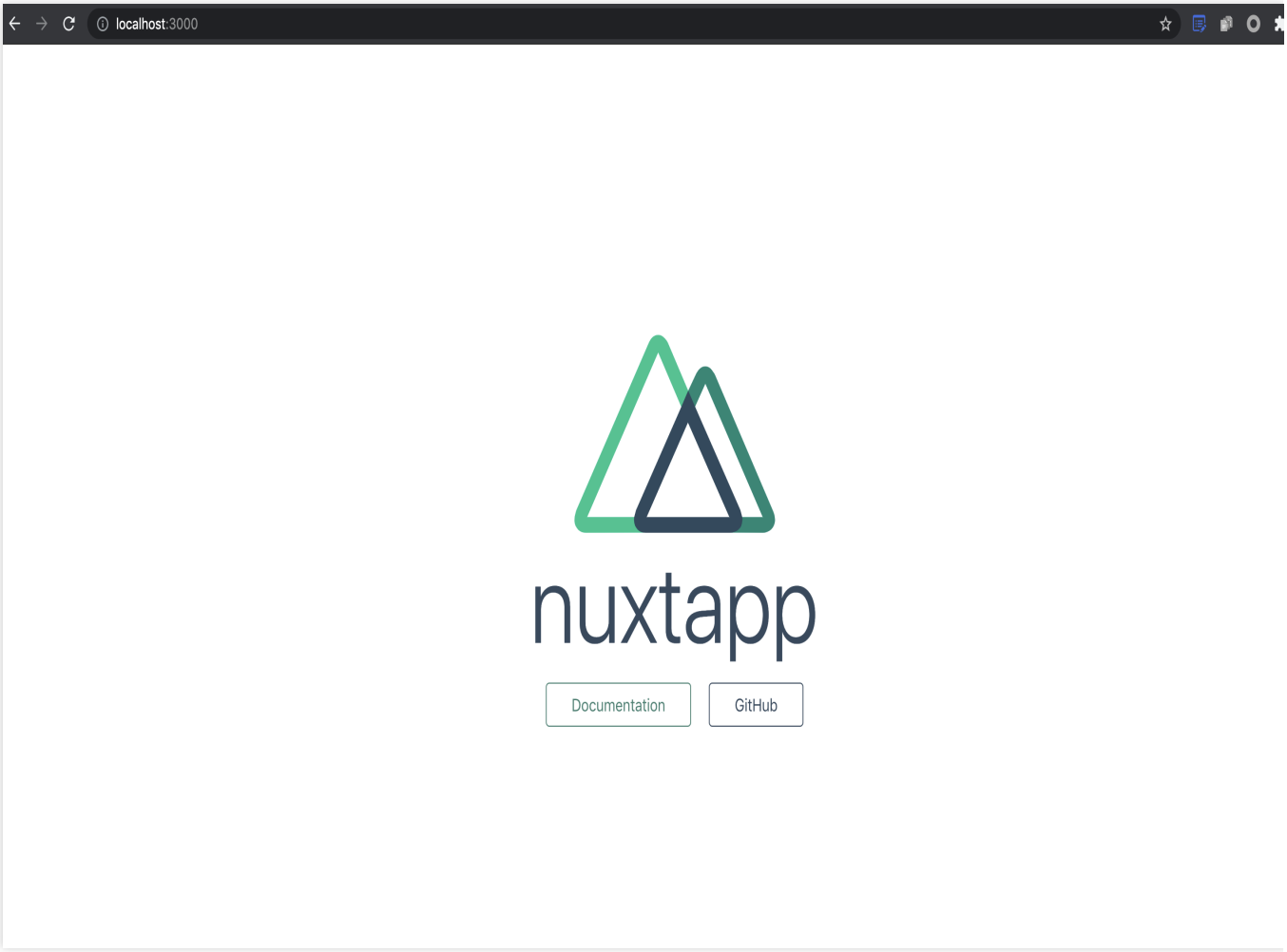
1. Refer to Installation to install and initialize your Nuxt.js project:

```
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 as shown below:



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` bootstrap file.

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

The detailed directions are as follows:

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

```
#!/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);
    });
```

Note:

Here is only a sample bootstrap file. Adjust the specific operations 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.

2. After the creation is completed, 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 it to start normally. Below is the sample code:

```
chmod 777 scf_bootstrap
```

3. Log in to the SCF console and click **Functions** on the left sidebar.

4. Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.

5. Select **Create from scratch** and configure the options as prompted:

Function type: Select **HTTP-triggered function**.

Function name: Enter the name of your function.

Region: Enter your function deployment region, which is **Guangzhou** by default.

Runtime environment: Select **Nodejs 12.16**.

Submitting method: Select **Local folder** and upload your local project.

Function codes: Select the specific local folder where the function code is.

6. Click **Complete**.

Note:

When you access the URL, the access may fail due to frontend routing. Therefore, you need to remove the `/release` path when accessing.

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 Django Framework

Last updated：2025-02-12 11:08:58

Overview

This document describes how to quickly deploy a local Django project to the cloud through an HTTP-triggered function.

Note:

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

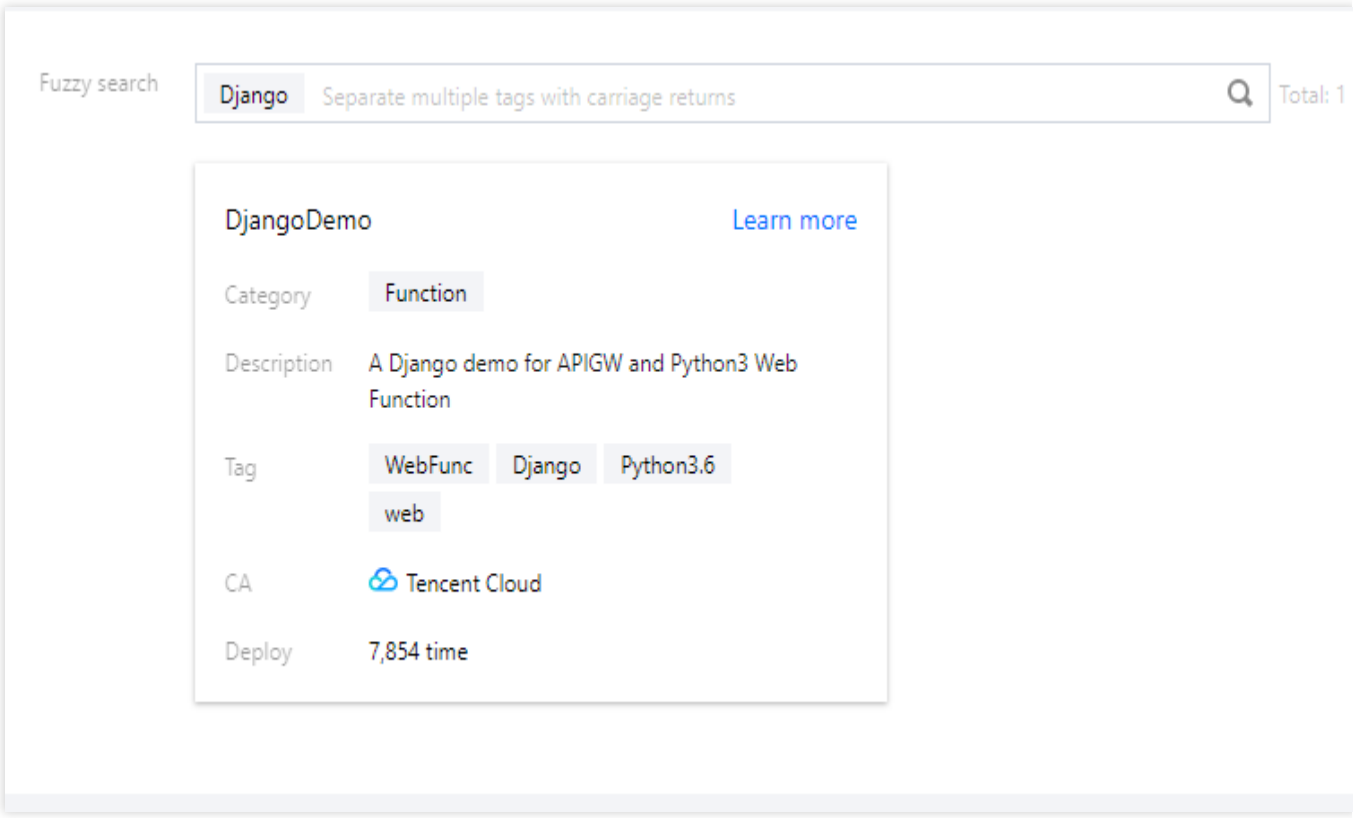
Prerequisites

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

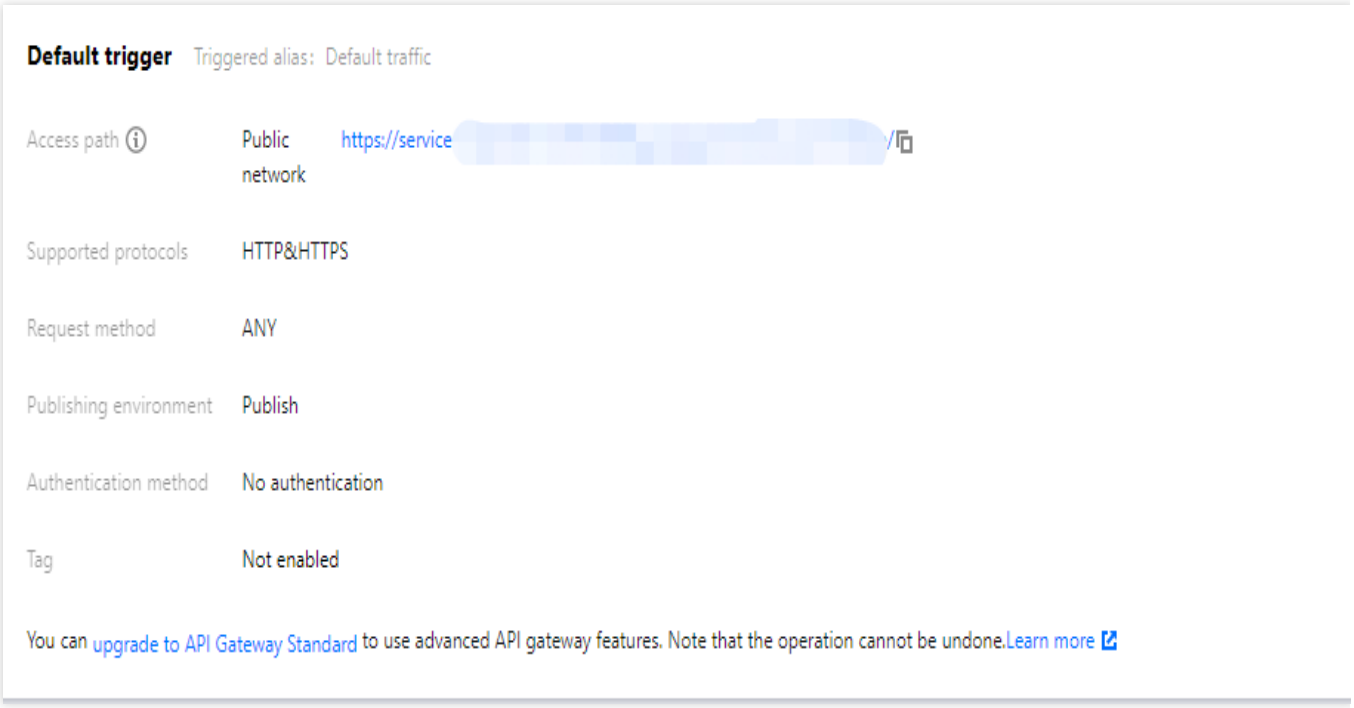
Directions

Template deployment: Quick deployment of Django project

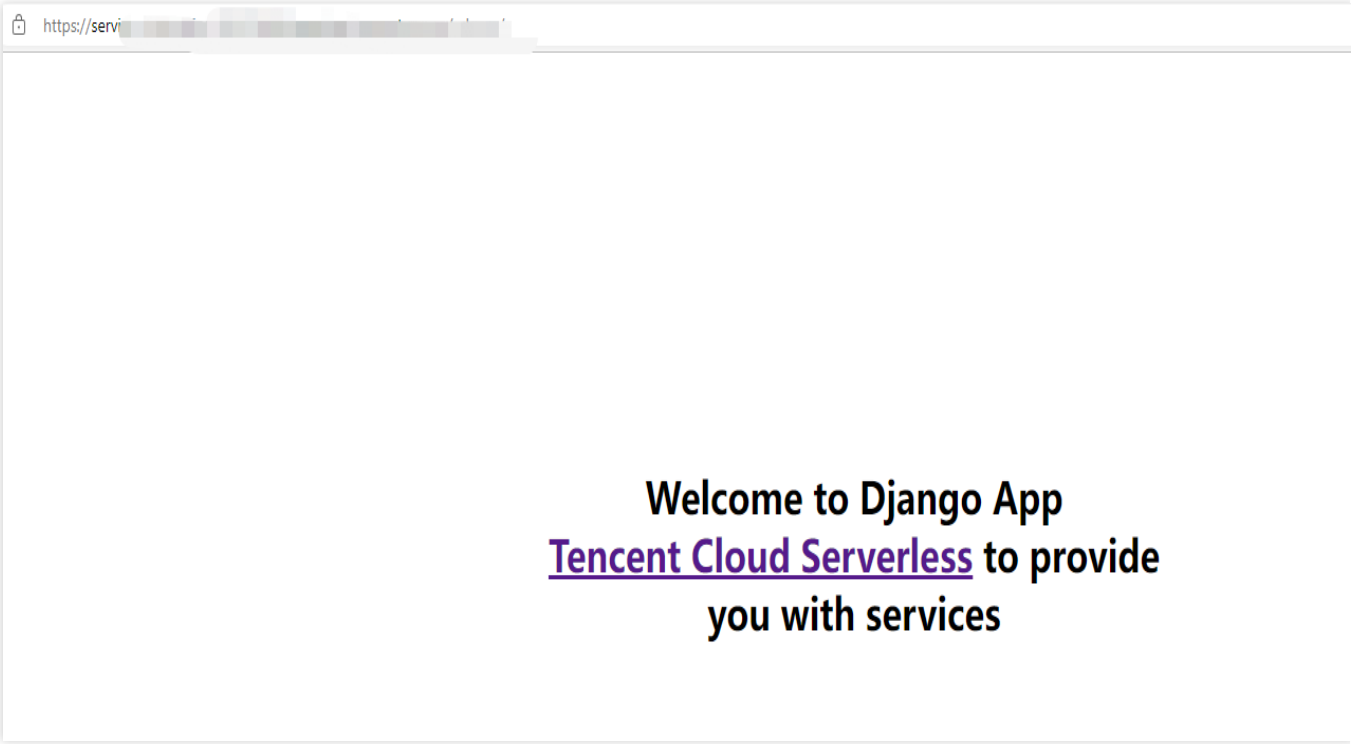
1. Log in to the SCF console and click **Functions** on the left sidebar.
2. Select the region and namespace where to create a function at the top of the page and click **Create** to enter the function creation process.
3. Select **Template**, enter `Django` in the search box, select the **Django template**, and click **Next** as shown below:



4. On the **Create** page, you can view and modify the specific configuration information of the template project.
5. Click **Complete**. After creating the HTTP-triggered function, you can view its basic information on the **Function management** page.
6. Click **Trigger management** on the left to view the access path and access your deployed Django project as shown below:



7. Click the access path URL to access the Django project as shown below:



Custom deployment: Quick migration of local project to cloud

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
```

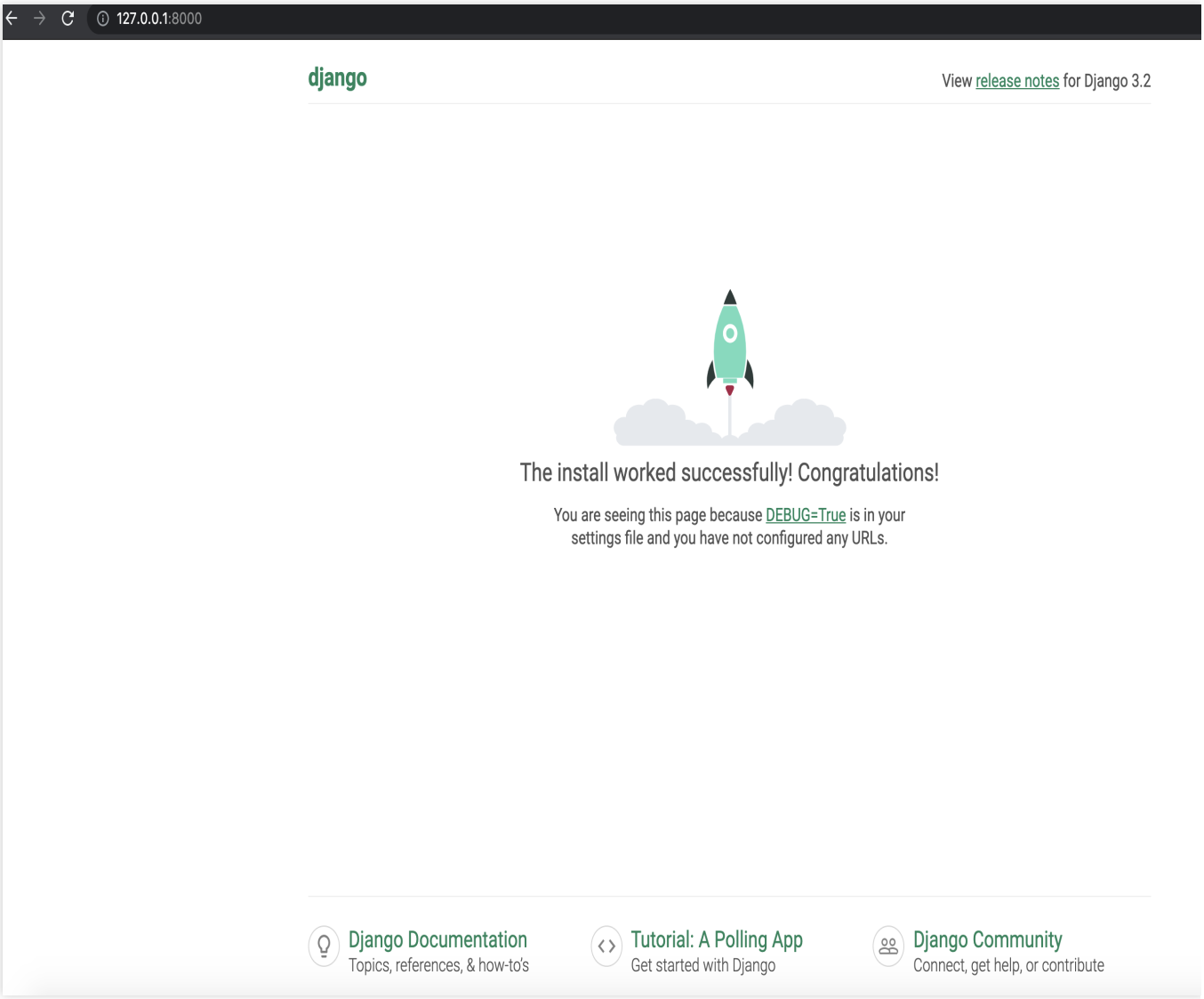
The directory structure is as follows:

```
$ 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

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 Django are as follows:

1. **Install dependencies**

1.1 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. Create the `requirements.txt` file first with the following content:

```
Django==3.1.3
```

1.2 Run the following command to install:

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

Note:

As the initialized default project imports the `db.sqlite3` library, install this dependency synchronously or configure comments for the `DATABASES` field in the `setting.py` file of the project.

2. **Add the `scf_bootstrap` bootstrap file**

The listening port in the HTTP-triggered function must be **9000**, so you need to change the listening address and port in the following way: create the `scf_bootstrap` bootstrap file in the project root directory and add the following content to it (which is used to configure environment variables, specify service bootstrap commands, and make sure that your service can be started normally through this file):

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

3. After the creation is completed, 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 it 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 the log, you need to add the `-u` parameter before the startup command, such as `python -u app.py` .

4. After the local configuration is completed, run the following command to start the service (with execution 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
```

5. Log in to the SCF console and click **Functions** on the left sidebar.

6. Select the region where to create a function at the top of the page and click **Create** to enter the function creation process.

7. Select **Create from scratch** and configure the options as prompted:

Function type: Select **HTTP-triggered function**.

Function name: Enter the name of your function.

Region: Enter your function deployment region, such as Chengdu.

Runtime environment: Select **Python 3.6**.

Submitting method: Select **Local folder** and upload your local project.

Function codes: Select the specific local folder where the function code is.

8. Click **Complete**.

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 as shown below: