ALIBABA CLOUD

Alibaba Cloud

Function Compute

Programming Languages

Document Version: 20201023

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	Danger: Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	Warning: Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	Notice: If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	Note: You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Programming languages

Function Compute currently supports the following programming languages.

Language	Reference
Node.js	Node.js runtime environment
Python	Python runtime environment
PHP	PHP runtime environment
Java	Java runtime environment
C#	.NET Core runtime environment
Go	Go Custom Runtime
Ruby	Ruby Custom Runtime
PowerShell	PowerShell Custom Runtime
TypeScript	TypeScript Custom Runtime
F#	F# Custom Runtime
Others	Custom Runtime User Manual

2.Terms

This topic describes some common terms of Function Compute, including the function entry point, handler function, function input parameters, log records, and troubleshooting.

Function entry point

When you create a function, you must specify an entry point for the function. Function Compute starts executing a function from its entry point. The entry point of a function in Function Compute is in the format of `[File name].[Function name]`.

For example, when you create a Node.js function with **Handler** set to *index.handler*, Function Compute loads the handler function defined in the index.js file.

Handler function

A handler function refers to a function that is specified by a function entry point. The handler function is similar to the main() function in local development. Handler functions must comply with the programming model provided by Function Compute.

A handler function can be an event function or an HTTP function.

- An event function can be used as a common handler function that provides a segment of service logic code. All handler functions are event functions except those for HTTP functions.
- An HTTP function is a function that is configured with an HTTP trigger. HTTP functions can directly process HTTP requests and return HTTP responses. They are suitable for creating web applications. The programming model for HTTP functions is different from that for event functions.

The following example shows an event function that runs in a Python runtime:

```
def handler(event, context):
  return 'hello world'
```

In this model, the custom event parameter and the platform-defined context parameter are the input parameters of the handler function. The handler function must parse the parameters and be able to call other functions defined in the code. Except for the definition, the handler function has the same code organization logic as a local function.

For more information about the handler function model for each programming language, see the corresponding programming language.

Initializer function

Function Compute dynamically executes functions on different Docker instances based on requests. When a function is called in succession, the function can be executed on the same Docker instance for multiple times. An initializer function is used to ensure that the same instance is executed successfully only once.

In database-related scenarios, you can add time-consuming service logic to the initializer function, such as the service logic for creating a connection pool and loading function dependency libraries. This prevents such service logic from being repeated each time the function is called, reducing the latency of the function.

Prior to handler functions, initializer functions are executed after Function Compute allocates functions to be executed on different Docker instances. Although HTTP functions and event functions have different programming models, their corresponding initializer functions have the same programming model.

For more information about the initializer function model for each programming language, see the corresponding programming language.

Input parameters of functions

Input parameters of a function are the content passed to the function when the function is called. Input parameters of a function usually consist of the event parameter and the context parameter. However, the number of input parameters may vary with the programming language and environment.

- event parameter

  The event parameter is a custom input parameter for a function. It is passed to the function in the format of byte streams. The data structure of the event parameter can be customized and can be a simple string, a JSON object, or a picture (which is binary data). Function Compute does not interpret the content of the event parameter. You must convert byte streams to the corresponding data type in the function.

  The value of the event parameter varies according to trigger conditions:

  - If a function is triggered by an event source service, the event source service passes the event to the function as an event parameter in a format predefined by the Function Compute platform. You can write code in this format and obtain information from the event parameter. For example, when you use an Object Storage Service (OSS) trigger, OSS passes the information about the bucket and its files to the event parameter in JSON format.
  - If a function is directly called by using the SDK, you can customize the event parameter between the caller and the function code. The caller passes in data in the defined format and the function code obtains the data in the same format. For example, you can define `{"key":"val"}`, a data structure in JSON format, as the `event` parameter. When a caller passes in the data `{"key":"val"}`, the function code converts byte streams to data in JSON format and then calls `event["key"]` to obtain the value `val`.

  The following example describes how to use the event parameter in Python:

  ```
  import json
  def handler(event, context):
    evt = json.loads(event)
    print(evt['key'])
    return 'success'
  ```

- context parameter

  The context parameter is a function input parameter defined by the Function Compute platform. Designed by Function Compute, the data structure of the context parameter contains the function runtime information. The data structure is described as follows:

- Function structure

```
{
  requestId: '9cda63c3-1ac9-45ba-8a59-2593bb9bc101',
  credentials: {
    accessKeyId: 'xxx',
    accessKeySecret: 'xxx',
    securityToken: 'xxx'
  },
  function: {
    name: 'xxx',
    handler: 'index.handler',
    memory: 512,
    timeout: 60,
    initializer: 'index.initializer',
    initializationTimeout: 10
  },
  service: {
    name: 'xxx',
    logProject: 'xxx',
    logStore: 'xxx',
    qualifier: 'xxx',
    versionId: 'xxx'
  },
  region: 'xxx',
  accountId: 'xxx'
}
```

- Scenarios
  - Obtain the temporary key information of a user from `context.credentials` and use the temporary key in `context` to access other Alibaba Cloud services. In this example, the key is used to access OSS. This avoids the use of hard-coded keys in code.
  - Obtain the basic information of the current execution from `context`. Such basic information includes requestId, serviceName, functionName, and qualifier.

- Example

  The following example describes how to use the personal information in `context` to access OSS in Python:

```
import json
import oss2
def handler(event, context):
    creds = context.credentials
    auth = oss2.StsAuth(creds.access_key_id, creds.access_key_secret, creds.security_token)
    bucket = oss2.Bucket(auth, 'oss-endpoint', 'your-bucket-name')
    bucket.put_object('object-name', 'Awesome FC')
    return 'success'
```

Log records

Function Compute is integrated with log records. Function Compute stores all function call records and the logs printed in function code into Logstores. You can make a record of function logs by running the logging statements provided by Function Compute. This helps you debug the function and locate problems in the system.

> **Note**
> - You need to configure a Logstore at the service level so that Function Compute sends function logs to the specified Logstore.
> - Function Compute automatically creates and configures a Logstore for functions and services that are created in the Function Compute console.

Programming language	Log printing statement in the function code	Logging statement provided by Function Compute	Reference
Node.js	console.log()	console.log()	Print logs
Python	print()	logging.getLogger().info()	Print logs
Java	System.out.println()	context.getLogger().info()	Print logs
PHP	echo "" . PHP_EOL	$GLOBALS['fcLogger']->info()	Print logs
C#	Console.WriteLine("")	context.Logger.LogInformation()	Print logs

Logs printed by using the built-in log printing statements of programming languages are also collected to Logstores. To facilitate log filtering, each log printed by using the logging statements provided by Function Compute is tagged with a request ID.

```
# Logs printed by using the built-in log printing statements of programming languages
# print('hello world')
message: hello world

# Log printed by using the logging statements provided by Function Compute
# logger.info('hello world')
message: 2020-03-13T04:06:49.099Z f84a9f4f-2dfb-41b0-9d6c-1682a2f3a650 [INFO] hello world
```

Log elements

A function execution log contains the service name, function name, the current function version, alias, and code logs.

For example, the data structure of a function execution log is as follows:

```
__source__:
__tag__:__receive_time__: 1584072413
__topic__: myService
functionName: myFunction
message: 2020-03-13T04:06:49.099Z f84a9f4f-2dfb-41b0-9d6c-1682a2f3a650 [INFO] hello world
qualifier: LATEST
serviceName: myService
versionId:
```

- When the execution of a function starts, the system prints `FC Invoke Start RequestId: f84a9f4f-2dfb-41b0-9d6c-1682a2f3a650`, indicating that function execution starts.
- When the execution of a function is completed, the system prints `FC Invoke End RequestId: f84a9f4f-2dfb-41b0-9d6c-1682a2f3a650`, indicating that function execution ends.

Handle errors

Errors in Function Compute are classified into `HandledInvocationError` and `UnhandledInvocationError`.

- HandledInvocationError

  Only the errors returned by `callback` in Node.js functions are of the `HandledInvocationError` type. Error information is contained in responses.

```
'use strict';
 module.exports.handler = function(event, context, callback) {
  console.log('hello world');
  callback('this is error', 'hello world');
 }
```

For example, the following information shows a response that contains error information.

```
{"errorMessage":"this is error"}
```

- UnhandledInvocationError

  All function errors are of the `UnhandledInvocationError` type except those of the `HandledInvocationError` type.

  The `stackTrace` property of an `UnhandledInvocationError` error is printed in logs. You can view the logs to find the corresponding `StackTrace` based on the context.

3.Custom Runtime

3.1. Examples

3.1.1. F#

With the Custom Runtime and HTTP triggers of Function Compute, you can migrate F# ASP.NETCore projects with one click, and thus you can access the function using a browser or an HTTP client tool such as curl.

Prerequisites

Before you install Funcraft, make sure that you have completed the following steps:

- Install Node.js 8.6.0 or later. For more information, see the official download page.
- Install Docker. For more information, see Docker Hub.

> Note
>
> - The examples in this topic apply to scenarios where Docker is installed. If you do not want to use Docker, you need to install. NET Core 3.1. For more information about installation, see dot.net.core3.1. For more information about the command, see the following command.
>
>   ```
>   https://github.com/awesome-fc/fc-custom-demo/blob/master/FSharp-demo/Makefile#L10-L18
>   ```
>
> - If you have installed Funcraft, you can go to Step 2: Deploy and invoke the function.

Step 1: Prepare the environment

The easiest way to install Funcraft is to download the executable binary files.

1. Install Funcraft on the local machine. For more information, see Install Funcraft.
2. Run the `fun --version` command to check whether the installation is successful.
3. Run the `fun config` command to configure Funcraft. Set the Account ID, Access Key ID, Access Key Secret, and Default region name parameters as prompted.

   ```
   fun config
   Aliyun Account ID 1234xxx
   Aliyun Access Key ID xxxx
   Aliyun Access Key Secret xxxx
   Default region name cn-xxxx
   The timeout in seconds for each SDK client invoking 300
   The maximum number of retries for each SDK client 5
   Allow to anonynously report usage statistics to improve the tool over time? (Y/n)
   ```

Step 2: Deploy and invoke the function

1. Run the following command to clone the sample project to your local device.

```
git clone https://github.com/awesome-fc/fc-custom-demo
```

> **Note** If you have not installed Git, download it from https://github.com/awesome-fc/fc-custom-demo and install it.

2. Run the following command to go to the cloned sample project.

```
cd FSharp-demo
```

3. Run the following command to deploy the function.

```
$ make deploy
docker run -it -v $(pwd):/tmp mcr.microsoft.com/dotnet/core/sdk:3.1 bash -c "cd /tmp/FSharpDem
o && dotnet publish -r linux-x64 -c Release --self-contained true && cd /tmp/FSharpDemo/bin/Rel
ease/netcoreapp3.1/linux-x64/publish && mv FSharpDemo bootstrap && chmod +x bootstrap"
Microsoft (R) Build Engine version 16.5.0+d4cbfca49 for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

  Restore completed in 15.2 sec for /tmp/FSharpDemo/FSharpDemo.fsproj.
  FSharpDemo -> /tmp/FSharpDemo/bin/Release/netcoreapp3.1/linux-x64/FSharpDemo.dll
  FSharpDemo -> /tmp/FSharpDemo/bin/Release/netcoreapp3.1/linux-x64/publish/
fun deploy -y
...
Waiting for service fsharp_demo to be deployed...
    Waiting for function fc_fsharp to be deployed...
        Waiting for packaging function fc_fsharp code...
        The function fc_fsharp has been packaged. A total of 338 files were compressed and the
final size was 39.3 MB
        Waiting for HTTP trigger http_t to be deployed...
        triggerName: http_t
        methods: [ 'GET', 'POST', 'PUT', 'DELETE' ]
        url: https://19861144305****.cn-hangzhou.fc.aliyuncs.com/2016-08-15/proxy/fsharp_demo
/fc_fsharp/
        Http Trigger will forcefully add a 'Content-Disposition: attachment' field to the response
header, which cannot be overwritten
        and will cause the response to be downloaded as an attachment in the browser. This iss
ue can be avoided by using CustomDomain.

        trigger http_t deploy success
    function fc_fsharp deploy success
service fsharp_demo deploy success

Detect 'DomainName:Auto' of custom domain 'my_domain'
Fun will reuse the temporary domain 16225220-19861144****.test.functioncompute.com, expired at
2020-04-17 10:07:00, limited by 1000 per day.

Waiting for custom domain my_domain to be deployed...
custom domain my_domain deploy success
```

4. Run the following command to invoke the deployed function.

In the example, directly use curl or the browser to access the temporary domain name `1622 5220-198611443 *****. test.functioncompute.com` after the deployment is successful .

```
$ curl 16225220-1986114430****.test.functioncompute.com/weatherforecast
[{"date":"2020-04-07T07:46:29.4315198+00:00","temperatureC":49,"summary":"Hot","temperatureF"
:120},{"date":"2020-04-08T07:46:29.431527+00:00","temperatureC":11,"summary":"Bracing","temper
atureF":51},{"date":"2020-04-09T07:46:29.4315276+00:00","temperatureC":45,"summary":"Bracing","
temperatureF":112},{"date":"2020-04-10T07:46:29.431528+00:00","temperatureC":13,"summary":"Ch
illy","temperatureF":55},{"date":"2020-04-11T07:46:29.4315284+00:00","temperatureC":-8,"summary
":"Mild","temperatureF":18}]
```

3.1.2. Ruby

You can use the Custom Runtime of Function Compute to write functions in Ruby. This topic describes how to deploy and invoke Ruby functions.

Prerequisites

You have installed Node.js 8.6.0 or later. For more information, see the official download page.

> **Note** If you have installed Funcraft, you can go to Step 2: Deploy and invoke the function.

Step 1: Prepare the environment

The easiest way to install Funcraft is to download the executable binary files.

1. Install Funcraft on the local machine. For more information, see Install Funcraft.
2. Run the `fun --version` command to check whether the installation is successful.
3. Run the `fun config` command to configure Funcraft. Set the Account ID, Access Key ID, Access Key Secret, and Default region name parameters as prompted.

```
fun config
Aliyun Account ID 1234xxx
Aliyun Access Key ID xxxx
Aliyun Access Key Secret xxxx
Default region name cn-xxxx
The timeout in seconds for each SDK client invoking 300
The maximum number of retries for each SDK client 5
Allow to anonynously report usage statistics to improve the tool over time? (Y/n)
```

Step 2: Deploy and invoke the function

1. Run the following command to clone the sample project to your local device.

```
git clone https://github.com/awesome-fc/fc-custom-demo
```

> Note If you have not installed Git, download it from https://github.com/awesome-fc/fc-custom-demo and install it.

2. Run the following command to go to the cloned sample project.

```
cd ruby-demo
```

3. Run the following command to deploy the project to Function Compute.

```
$fun deploy -y
...
Waiting for service ruby-demo to be deployed...
    Waiting for function fc-ruby to be deployed...
        Waiting for packaging function fc-ruby code...
        The function fc-ruby has been packaged. A total of 4 files were compressed and the final size was 1.45 KB
    function fc-ruby deploy success
service ruby-demo deploy success
```

4. Run the following command to invoke the deployed function.

```
$ fun invoke -e "Hello World"
...
========= FC invoke Logs begin =========
FC Invoke Start RequestId: a9c4dc8a-4b5b-48e0-9a3f-70310531ae61
Hello World
FC Invoke End RequestId: a9c4dc8a-4b5b-48e0-9a3f-70310531ae61

Duration: 0.69 ms, Billed Duration: 100 ms, Memory Size: 512 MB, Max Memory Used: 5.06 MB
========= FC invoke Logs end =========

FC Invoke Result:
Hello World
```

3.1.3. TypeScript

You can use TypeScript to write functions in the Custom Runtime of Function Compute. This topic describes how to deploy and invoke TypeScript functions.

Prerequisites

You have installed Node.js 8.6.0 or later. For more information, see the official download page.

> Note If you have installed Funcraft, you can go to Step 2: deploy and call the function.

Step 1: Prepare the environment

The easiest way to install Funcraft is to download the executable binary files.

1. Install Funcraft on the local machine. For more information, see Install Funcraft.
2. Run the `fun --version` command to check whether the installation is successful.
3. Run the `fun config` command to configure Funcraft. Set the Account ID, Access Key ID, Access Key Secret, and Default region name parameters as prompted.

```
fun config
Aliyun Account ID 1234xxx
Aliyun Access Key ID xxxx
Aliyun Access Key Secret xxxx
Default region name cn-xxxx
The timeout in seconds for each SDK client invoking 300
The maximum number of retries for each SDK client 5
Allow to anonynously report usage statistics to improve the tool over time? (Y/n)
```

Step 2: deploy and call the function

1. Run the following command to clone the sample project to your local device.

```
git clone https://github.com/awesome-fc/fc-custom-demo
```

> **Note** If you have not installed Git, download it from https://github.com/awesome-fc/fc-custom-demo and install it.

2. Run the following command to go to the cloned sample project.

```
cd ts-demo
```

3. Run the following command to deploy the project to Function Compute.

```
$ fun deploy -y
...
Waiting for service ts-demo to be deployed...
    Waiting for function fc-ts to be deployed...
        Waiting for packaging function fc-ts code...
        The function fc-ts has been packaged. A total of 336 files were compressed and the final
size was 9.41 MB
    function fc-ts deploy success
service ts-demo deploy success
```

4. Run the following command to invoke the deployed function.

```
$ fun invoke -e "Hello World"
...
========= FC invoke Logs begin =========
FC Invoke Start RequestId: 7ab0a86a-be32-4086-ac17-3ce0797cda41
Hello World
FC Invoke End RequestId: 7ab0a86a-be32-4086-ac17-3ce0797cda41

Duration: 13.48 ms, Billed Duration: 100 ms, Memory Size: 512 MB, Max Memory Used: 162.38 MB
========= FC invoke Logs end =========

FC Invoke Result:
Hello World
```

3.1.4. PowerShell

You can use the Custom Runtime of Function Compute to write functions in PowerShell. This topic describes how to deploy and invoke PowerShell functions.

Prerequisites

You have installed Node.js 8.6.0 or later. For more information, see the official download page.

> **Note** If you have installed Funcraft, you can go to Step 2: Deploy and invoke the function.

Step 1: Prepare the environment

The easiest way to install Funcraft is to download the executable binary files.

1. Install Funcraft on the local machine. For more information, see Install Funcraft.
2. Run the `fun --version` command to check whether the installation is successful.
3. Run the `fun config` command to configure Funcraft. Set the Account ID, Access Key ID, Access Key Secret, and Default region name parameters as prompted.

```
fun config
Aliyun Account ID 1234xxx
Aliyun Access Key ID xxxx
Aliyun Access Key Secret xxxx
Default region name cn-xxxx
The timeout in seconds for each SDK client invoking 300
The maximum number of retries for each SDK client 5
Allow to anonynously report usage statistics to improve the tool over time? (Y/n)
```

Step 2: Deploy and invoke the function

1. Run the following command to clone the sample project to your local device.

```
git clone https://github.com/awesome-fc/fc-custom-demo
```

> **Note** If you have not installed Git, download it from https://github.com/awesome-fc/fc-custom-demo and install it.

2. Run the following command to go to the cloned sample project.

```
cd powershell-demo
```

3. Run the following command to install local dependencies.

```
$ fun install -v
start installing function dependencies without docker

building powershell-demo/fc-powershell
Funfile exist, Fun will use container to build forcely
Step 1/7 : FROM registry.cn-beijing.aliyuncs.com/aliyunfc/runtime-custom:build-1.9.4
...
Step 7/7 : RUN fun-install apt-get install powershell
...
sha256:7c6476a3a4496bbf3da83bfb8966acc90fa94f4f8baccd248cd79c18c4fae409
Successfully built 7c6476a3a449
Successfully tagged fun-cache-965bbabb-ab6f-44e7-9910-7dd1df1d3c3e:latest
copying function artifact to /Users/songluo/gitpro/fc-custom-demo/powershell-demo
copy from container /mnt/auto/. to localNasDir

Install Success
```

4. Run the following command to deploy the project to Function Compute.

```
$ fun deploy -y
....
? Do you want to let fun to help you automate the configuration? Yes
? You have already configured 'NasConfig: Auto'. We want to use this configuration to store your function dependencies. Yes

Fun automatically backups the original template.yml file to /Users/txd123/Desktop/jack/fc-custom-demo/powershell-demo/.template.yml.backup
Fun add .fun/root to /Users/txd123/Desktop/jack/fc-custom-demo/powershell-demo/.nas.yml
Fun add environment variables to 'powershellDemo/fc-powershell' in /Users/txd123/Desktop/jack/fc-custom-demo/powershell-demo/template.yml

starting upload /Users/txd123/Desktop/jack/fc-custom-demo/powershell-demo/.fun/root to nas://powershellDemo/mnt/auto/root/

start fun nas init...
....
fun nas init Success

zipping /Users/txd123/Desktop/jack/fc-custom-demo/powershell-demo/.fun/root
✔ upload done
unzipping file
✔ unzip done
✔ upload completed!

....
? Region cn-qingdao only supports capacity NAS. Do you want to create it automatically? Yes
      ....
   Waiting for function fc-powershell to be deployed...
      Waiting for packaging function fc-powershell code...
      The function fc-powershell has been packaged. A total of 3 files were compressed and the final size was 1.31 KB
   function fc-powershell deploy success
   function fc-powershell deploy success
service powershellDemo deploy success
```

5. Run the following command to invoke the deployed function.

```
$ fun invoke -e "Hello World"
...
========= FC invoke Logs begin =========
FC Invoke Start RequestId: cd30369e-7dfa-439c-a68d-7fe16d5a7e05
Hello World
FC Invoke End RequestId: cd30369e-7dfa-439c-a68d-7fe16d5a7e05

Duration: 54.13 ms, Billed Duration: 100 ms, Memory Size: 512 MB, Max Memory Used: 133.70 MB
========= FC invoke Logs end =========

FC Invoke Result:
Hello World
```

3.1.5. Go

You can deploy a Go Project to Function Compute with one click using the Custom Runtime of Function Compute. This topic describes how to deploy and invoke Go functions.

Prerequisites

Before you install Funcraft, make sure that you have completed the following steps:

- Install Node.js 8.6.0 or later. For more information, see the official download page.
- Install Docker. For more information, see Docker Hub.

> **Note** If you have installed Funcraft, you can go to Step 2: Deploy and invoke the function.

Step 1: Prepare the environment

The easiest way to install Funcraft is to download the executable binary files.

1. Install Funcraft on the local machine. For more information, see Install Funcraft.
2. Run the `fun --version` command to check whether the installation is successful.
3. Run the `fun config` command to configure Funcraft. Set the Account ID, Access Key ID, Access Key Secret, and Default region name parameters as prompted.

```
fun config
Aliyun Account ID 1234xxx
Aliyun Access Key ID xxxx
Aliyun Access Key Secret xxxx
Default region name cn-xxxx
The timeout in seconds for each SDK client invoking 300
The maximum number of retries for each SDK client 5
Allow to anonynously report usage statistics to improve the tool over time? (Y/n)
```

Step 2: Deploy and invoke the function

1. Run the following command to clone the sample project to your local device.

```
git clone https://github.com/awesome-fc/fc-custom-demo
```

> **Note** If you have not installed Git, download it from https://github.com/awesome-fc/fc-custom-demo and install it.

2. Run the following command to go to the cloned sample project.

```
cd go_demo
```

3. Run the following command to deploy the function.

```
$ make deploy
docker build -t fc-go-runtime  -f build-image/Dockerfile build-image
Sending build context to Docker daemon  3.072kB
Step 1/5 : FROM golang:1.12.16-stretch
 ---> 7ad03a8aece5
...
Step 5/5 : RUN go get github.com/awesome-fc/golang-runtime
 ---> Using cache
 ---> 262e7f38ac05
Successfully built 262e7f38ac05
Successfully tagged fc-go-runtime:latest
docker run -it -v $(pwd):/tmp fc-go-runtime bash -c "go build -o /tmp/code//bootstrap /tmp/code
/main.go"
chmod +x code/bootstrap
fun deploy -y
using template: template.yml
...
Waiting for service go_demo to be deployed...
    Waiting for function fc_go to be deployed...
        Waiting for packaging function fc_go code...
        The function fc_go has been packaged. A total of 1 file were compressed and the final si
ze was 3.76 MB
    function fc_go deploy success
service go_demo deploy success
```

4. Run the following command to invoke the deployed function.

```
$ fun invoke -e "Hello World"
...
========= FC invoke Logs begin =========
FC Invoke Start RequestId: 4c1451b2-f29b-4554-87e5-126f3bc11fcf
2020-04-07T02:53:01.981Z: 4c1451b2-f29b-4554-87e5-126f3bc11fcf [INFO]  hello golang!
FC Invoke End RequestId: 4c1451b2-f29b-4554-87e5-126f3bc11fcf

Duration: 1.03 ms, Billed Duration: 100 ms, Memory Size: 512 MB, Max Memory Used: 4.39 MB
========= FC invoke Logs end =========

FC Invoke Result:
Hello World
```

3.1.6. CPP

Custom Runtime provided by Function Compute allows you to write functions in C++ (CPP) on the Function Compute platform. This topic describes how to quickly deploy and call a CPP function.

Prerequisites

You have installed Node.js 8.6.0 or later. For more information, see the official download page.

> Note If you have installed FunCraft, you can directly go to Step 2: Deploy and call a function.

Step 1: Prepare the environment

The easiest way to install Funcraft is to download the executable binary files.

1. Install Funcraft on the local machine. For more information, see Install Funcraft.
2. Run the `fun --version` command to check whether the installation is successful.
3. Run the `fun config` command to configure Funcraft. Set the Account ID, Access Key ID, Access Key Secret, and Default region name parameters as prompted.

```
fun config
Aliyun Account ID 1234xxx
Aliyun Access Key ID xxxx
Aliyun Access Key Secret xxxx
Default region name cn-xxxx
The timeout in seconds for each SDK client invoking 300
The maximum number of retries for each SDK client 5
Allow to anonynously report usage statistics to improve the tool over time? (Y/n)
```

Step 2: Deploy and call a function

1. Run the following command to clone the sample project to your local device.

```
git clone https://github.com/awesome-fc/fc-custom-demo
```

> Note If you have not installed Git, download it from https://github.com/awesome-fc/fc-custom-demo and install it.

2. Run the following command to access the cloned sample project:

```
cd cpp_demo
```

3. Run the following command to deploy the project to the Function Compute instance:

```
$ make deploy
docker build -t fc-cpp-runtime  -f build-image/Dockerfile build-image
Sending build context to Docker daemon  2.048kB
Step 1/3 : FROM aliyunfc/runtime-custom:base
 ---> 5bbdcf6377bd
```

```
...
Step 3/3 : RUN apt-get install -y cmake
 ---> Using cache
 ---> a244cd26cec2
Successfully built a244cd26cec2
Successfully tagged fc-cpp-runtime:latest
docker run --rm -it -v $(pwd):/tmp fc-cpp-runtime bash -c "cd /tmp && ./build.sh"
-- The C compiler identification is GNU 6.3.0
-- The CXX compiler identification is GNU 6.3.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
...
Scanning dependencies of target cppruntime
[ 33%] Building CXX object CMakeFiles/cppruntime.dir/src/handler.cpp.o
[ 66%] Building CXX object CMakeFiles/cppruntime.dir/src/logging.cpp.o
[100%] Linking CXX shared library /tmp/bin/libcppruntime.so
...
-- Build files have been written to: /tmp/sample/release
...
[100%] Built target bootstrap
fun deploy -y

Waiting for service cpp_demo to be deployed...
    Waiting for function fc_cpp_event to be deployed...
        Waiting for packaging function fc_cpp_event code...
        The function fc_cpp_event has been packaged. A total of 2 files were compressed and th
e final size was 446.51 KB
    function fc_cpp_event deploy success
    Waiting for function fc_cpp_http to be deployed...
        Waiting for packaging function fc_cpp_http code...
        The function fc_cpp_http has been packaged. A total of 2 files were compressed and the
final size was 446.51 KB
        Waiting for HTTP trigger http_t to be deployed...
        triggerName: http_t
        methods: [ 'GET', 'POST', 'PUT', 'DELETE' ]
        url: https://123456789.cn-hangzhou.fc.aliyuncs.com/2016-08-15/proxy/cpp_demo/fc_cpp_
http/
        Http Trigger will forcefully add a 'Content-Disposition: attachment' field to the response
header, which cannot be overwritten
        and will cause the response to be downloaded as an attachment in the browser. This iss
ue can be avoided by using CustomDomain
```

```
de can be avoided by using customDomain.

            trigger http_t deploy success
        function fc_cpp_http deploy success
service cpp_demo deploy success

Detect 'DomainName:Auto' of custom domain 'my_domain'
Fun will reuse the temporary domain http://17904911-123456789.test.functioncompute.com, expire
d at 2020-05-06 20:41:51, limited by 1000 per day.

Waiting for custom domain my_domain to be deployed...
custom domain my_domain deploy success
```

4. Run the following command to call the deployed function:

```
$ fun invoke cpp_demo/fc_cpp_event -e "Hello World"
...
========= FC invoke Logs begin =========
/invoke is called.
FC Invoke Start RequestId: bf282a87-0f0b-4953-b159-a31792bad22a
2020-04-27T08:01:08 bf282a87-0f0b-4953-b159-a31792bad22a [INFO] handling invoke
FC Invoke End RequestId: bf282a87-0f0b-4953-b159-a31792bad22a

Duration: 9.11 ms, Billed Duration: 100 ms, Memory Size: 512 MB, Max Memory Used: 4.25MB
========= FC invoke Logs end =========

FC Invoke Result:
Hello World
$ curl http://17904911-123456789.test.functioncompute.com -d "Hello World"
Hello World
```

3.1.7. Lua

Custom Runtime provided by Function Compute allows you to use the Lua language to write functions on the Function Compute platform. This topic describes how to quickly deploy and call a Lua function.

Prerequisites

You have installed Node.js 8.6.0 or later. For more information, see the official download page.

> **Note** If you have installed FunCraft, you can directly go to Step 2: Deploy and call a function.

Step 1: Prepare the environment

The easiest way to install Funcraft is to download the executable binary files.

1. Install Funcraft on the local machine. For more information, see Install Funcraft.
2. Run the `fun --version` command to check whether the installation is successful.
3. Run the `fun config` command to configure Funcraft. Set the Account ID, Access Key ID, Access Key Secret, and Default region name parameters as prompted.

```
fun config
Aliyun Account ID 1234xxx
Aliyun Access Key ID xxxx
Aliyun Access Key Secret xxxx
Default region name cn-xxxx
The timeout in seconds for each SDK client invoking 300
The maximum number of retries for each SDK client 5
Allow to anonynously report usage statistics to improve the tool over time? (Y/n)
```

Step 2: Deploy and call a function

1. Run the following command to clone the sample project to your local device.

```
git clone https://github.com/awesome-fc/fc-custom-demo
```

> **Note** If you have not installed Git, download it from https://github.com/awesome-fc/fc-custom-demo and install it.

2. Run the following command to access the cloned sample project:

```
cd lua-demo
```

3. Run the following command to deploy the project to the Function Compute instance:

```
fun deploy -y
...
Waiting for service lua-demo to be deployed...
    Waiting for function fc-lua to be deployed...
        Waiting for packaging function fc-lua code...
        The function fc-lua has been packaged. A total of 7 files were compressed and the final s
ize was 10.62 MB
    function fc-lua deploy success
service lua-demo deploy success
```

4. Run the following command to call the deployed function:

```
fun invoke -e "Hello World"
...
========= FC invoke Logs begin =========
FC Invoke Start RequestId: dcc2ff81-2318-4a89-abae-c181ede22b79, client: 21.0.3.254, server: , req
uest: "POST /invoke HTTP/1.1", host: "21.0.3.1:9000"
2020/05/10 13:16:21 [notice] 7#7: *2 [lua] main.lua:17: FC Invoke End RequestId: dcc2ff81-2318-4a8
9-abae-c181ede22b79, client: 21.0.3.254, server: , request: "POST /invoke HTTP/1.1", host: "21.0.3.
1:9000"
21.0.3.1 21.0.3.254 0.000 [10/May/2020:13:16:21 +0000] "POST /invoke HTTP/1.1" 200 22 "-" "Go-http-
client/1.1" "-" dcc2ff81-2318-4a89-abae-c181ede22b79

Duration: 3.42 ms, Billed Duration: 100 ms, Memory Size: 512 MB, Max Memory Used: 4.66 MB
========= FC invoke Logs end =========

FC Invoke Result:
Hello World
```

4.Custom Container

4.1. Overview

This topic describes custom container runtime, including its background, limits, operation, HTTP server configuration, log format, cold start optimization, and billing.

Background

In the cloud-native age, container images have become a standard tool for software deployment and development. Custom container runtime offers developers a simpler experience and makes development and delivery more efficient. Developers can deliver their functions as container images and interact over HTTP with Function Compute. Custom container runtime simplifies the following scenarios:

- You can perform low-cost migrations and maintain consistency between development and production environments without modifying code or recompiling binary and shared objects (*.so).
- Compressed images can be up to 1 GB in size, allowing you to package code and dependencies together for easier distribution and deployment.
- Container images are natively stored in a cache hierarchy, which improves the efficiency of uploading and pulling code.
- Standard, replicable third-party libraries are available for referencing, sharing, building, code uploading, storage, and version management. This offers a rich open-source ecosystem for continuous integration (CI) and continuous delivery (CD).

How it works

Custom container runtime operates on the same basic principles as custom runtime. Before it initializes an instance, Function Compute assumes the service role of the function to obtain a temporary user name and password and pull the image. After the image is pulled, Function Compute starts your HTTP server with the specified startup command (Command) and parameters (Args). The server listens on your configured listening port, such as CAPort. The server interacts with Function Compute and processes requests by using function logic. Function Compute forwards event calls and HTTP calls to the server to trigger functions.

HTTP server configuration

- Configure the HTTP server to listen on all local IP addresses (0.0.0.0) and the specified port. Read and use the FC_SERVER_PORT environment variable to set the port. The default value is 9000.
- Configure `connection keep-alive` on the HTTP server.
- Set the request timeout to 15 minutes or greater.
- Ensure that the HTTP server can start up within 25 seconds.

Function log format

Custom container runtime uses the same log format as custom runtime. For more information, see Log format.

Best practices for cold start optimization

The basic environment for container images brings a larger download than a code package, and the downloaded content takes longer to decompress. We recommend the following best practices to improve the cold start experience:

- To ensure sufficient bandwidth and stability for internal network transmissions, you can use the registry-vpc.Endpoint address for container images. An example address is registry-vpc.cn-hangzhou.aliyuncs.com/fc-demo/helloworld:v1beta1.
- For optimal latency and stability when images are pulled, we recommend that you use a VPC endpoint in the same region as Alibaba Cloud Container Registry (ACR) and Function Compute.
- To minimize images, you can include only necessary dependencies and code in your images. Unnecessary documents, data, or other files may cause delays.
- You can use container images with reserved instances. For more information, see Reserved instances.

Billing rules

Custom container runtime uses the same billing items as other runtime services. For more information, see Billing. The run time item under image resource usage is the period from when the repository initiates the image pull to when the pull is finished. For example, if an instance with 1,024 MB of memory takes 10 seconds to pull an image, the resource usage for this pull is 10 MBCU-second. Container images are cached for some time. A cold start does not necessarily incur image pull fees.

Limits

- Image size:
  - If the memory to run a function is less than 1 GB, the size of the image before decompression cannot exceed 256 MB.
  - If the memory to run a function is 1 GB or greater, the size of the image before decompression cannot exceed 1,024 MB.
- Image repository: Only default instance images in Container Registry are supported.
- Image access: You can read images only from private image repositories in the same region and under the same account.
- Read and write permissions of files in containers: The UID of run-as-user is randomly selected from the range of 10000 to 10999. The */tmp* directory in containers is writable by default. Read and write permissions for other directories are controlled by the image file system. If a running UID does not have read and write access to a file, you can modify the permissions of the UID in *Dockerfile*.
- The writable layer of a container is 512 MB. The data generated by a container cannot exceed this size.

> **Note** The data stored in the writable layer of a container does not persist. When the container is deleted, this data is also deleted. If persistent storage is required, you can use Function Compute to mount an Apsara File Storage NAS file system. For more information, see Configure NAS. You can also use other shared storage services, such as Object Storage Service or Tablestore.

4.2. Event functions

This topic describes the input parameters and responses of event functions for custom container runtime and provides sample code.

Background

In a custom container runtime, Function Compute forwards the common headers, request body, POST method, *and /invoke* and */initialize* paths to the HTTP server in the container. You can use function signatures similar to context and event of officially supported runtimes such as Golang Runtime. You can also use the request headers and body as input parameters to define the service logic of a function. For more information, see Event function calls in a custom runtime.

Input parameters of functions

- event: the body of the POST request
- context:
  - The x-fc-access-key-id, x-fc-access-key-secret, and x-fc-security-token headers are used to obtain temporary credentials for the service role to access other Alibaba Cloud services.
  - The x-fc-request-id header is used to obtain the current request ID.
  - For more information about request headers, see Common headers.

Function response structure

The results of a function call are returned as an HTTP response.

Sample code

In the following Node.js Express example, the POST method and */initialize* are called by Function Compute when a function instance is initialized. The POST method and */invoke* path serve as handlers when Function Compute is called. The context and event parameters are obtained from `req.headers` and `req.body`, and the results of the function call are then returned as an HTTP response.

```
'use strict';

const express = require('express');

// Constants
const PORT = 8080;
const HOST = '0.0.0.0';

// HTTP function invocation
const app = express();
app.get('/*', (req, res) => {
  res.send('Hello FunctionCompute, http function\n');
});

// Event function invocation
app.post('/invoke', (req, res) => {
  res.send('Hello FunctionCompute, event function\n');
});

var server = app.listen(PORT, HOST);
console.log(`Running on http://${HOST}:${PORT}`);

server.timeout = 0; // never timeout
server.keepAliveTimeout = 0; // keepalive, never timeout
```

4.3. HTTP functions

This topic describes the request and response specifications of HTTP functions for custom container runtime and provides sample code.

Background

Function Compute forwards user requests, including the path, request headers, request body, and common headers, to the HTTP server. Unlike with event functions, you do not need to implement `/invoke` and `/initialize`. HTTP functions enable you to smoothly migrate an existing HTTP web application. For more information, see Function calls in a custom runtime.

Input parameters of functions

- event: the body of the POST request
- context:
  - The x-fc-access-key-id, x-fc-access-key-secret, and x-fc-security-token headers are used to obtain temporary credentials for the service role to access other Alibaba Cloud services.

- The x-fc-request-id header is used to obtain the current request ID.
- For more information about request headers, see Common headers.

Sample code

In the following Node.js Express example, the GET and POST methods are routed to different handlers. You can map a path to a handler that you need.

```
'use strict';

const express = require('express');

// Constants
const PORT = 9000;
const HOST = '0.0.0.0';

// HTTP function get
const app = express();
app.get('/*', (req, res) => {
  res.send('Hello FunctionCompute, http GET');
});

const app = express();
app.post('/*', (req, res) => {
  res.send('Hello FunctionCompute, http POST');
});

app.listen(PORT, HOST);
console.log(`Running on http://${HOST}:${PORT}`);
```

4.4. Create a function

This topic describes how to create a custom container runtime function in the Function Compute console or by using Funcraft.

Prerequisites

1. You have created a namespace in Container Registry Default Instance Edition.
2. Create a repository

Create a function in the Function Compute console.

1. Upload your function image to the image repository for the default instance.

   In the following example, the region of Function Compute is China (Shenzhen), and the name of the image repository is nodejs-express.

```
cd /tmp
git clone https://github.com/awesome-fc/custom-container-docs.git && cd custom-container-docs
/nodejsexpress

# Specify the ACR image address
export IMAGE_NAME="your ACR image name"  # e.g. registry.cn-shenzhen.aliyuncs.com/fc-demo/n
odejs-express:v0.2
docker build -t $IMAGE_NAME .
docker push $IMAGE_NAME
```

2. Create a service and set permissions for it.
   i. In the Function Compute console, create a service. For more information, see Create a service.
   ii. Assign the **AliyunContainerRegistryReadOnlyAccess** or **AliyunContainerRegistryFullAccess** permission policy to the service. For more information, see Configure service permissions.In Function Compute, apply the preceding policies to the temporary account for the default instance in Alibaba Cloud Container Registry. Use the temporary account to upload images from your private image repository.
3. Create a function.In this example, an event function is created. HTTP functions use similar parameter settings.
   i. Log on to the Function Compute console.
   ii. In the top navigation bar, select your region.
   iii. In the left-side navigation pane, click **Service/Function**.
   iv. Click Create Function. On the page that appears, select Event Function and click Next.
   v. In the **Configure Function** step, set the parameters and click **Create**.

Parameter	Setting
Service Name	Select the service that you created in Step 2.
Function Name	Enter a custom function name.
Runtime	Select **custom-container** from the drop-down list.
Instance Type	Select a function instance type. Valid values: ■ **Elastic** ■ **Performance** For more information about instance types, see Instance specifications.

Parameter	Setting
Container Image	Enter the address of the container image. We recommend that you use the registry-vpc.Endpoint address for container images. An example address is registry-vpc.cn-hangzhou.aliyuncs.com/fc-demo/helloworld:v1beta1.
Command	Enter the startup command, for example, `["/code/myserver"]`. This parameter is optional. If you do not set this parameter, the ENTRYPOINT or CMD instruction included in the image is used.
Args	Enter additional parameters, for example, `["-arg1", "value1"]`. This parameter is optional. If you do not set this parameter, the CMD instruction included in the image is used.
Memory	Select the memory size used to run the function. The memory size cannot be less than 521 MB.
Timeout	Enter the timeout for requests.
Single Instance Concurrency	Enter the number of requests that can be concurrently processed by a single instance. For more information, see Concurrent requests in a single instance.
Listening Port	Enter the port on which the server listens. This parameter is optional. Default value: 9000. You can change the listening port on this page, without needing to modify the image.

After the function is created, you can view it in the function list of the corresponding service.

Use Funcraft to create a function

You can use Funcraft to build and upload container images and deploy functions with one click.

1. Run the following commands to clone the custom-container-docs example:

```
git clone https://github.com/awesome-fc/custom-container-docs.git
cd custom-container-docs/nodejsexpress
```

2. In *template.yml*, replace the value of the Image parameter with the address of your Alibaba Cloud Container Registry (ACR) image.
3. Run the following commands to build the image and deploy the function:

```
# Build the Docker image.
fun build --use-docker

# Deploy the function, push the image via the internet registry host (the function config uses the VPC registry for faster image pulling).
fun deploy --push-registry acr-internet
```

After the function is deployed, you can log on to the Function Compute console and view the function in the function list of the corresponding service.