

# Alibaba Cloud

## Function Compute Programming Languages

Document Version: 20220712

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

# Table of Contents

1. Overview	07
2. Terms	08
3. Go	13
3.1. Runtime environment	13
3.2. Handlers	13
3.3. Event handlers	14
3.4. HTTP handlers	16
3.5. Context	18
3.6. Compilation and deployment of code packages	21
3.7. Logs	22
3.8. Error handling	23
3.9. Lifecycle callbacks for function instances	26
4. Custom Runtime	29
4.1. Overview	29
4.2. Basic principles	35
4.3. CDN	37
4.4. Event functions	37
4.5. HTTP functions	41
4.6. Lifecycle hooks for function instances	43
4.7. Specification details	44
5. Custom Container	49
5.1. Overview	49
5.2. Introduction	51
5.3. Event functions	53
5.4. HTTP functions	53
5.5. Create a function	54

5.6. Accelerate image pull for Container Registry Personal Editi...	57
5.7. Accelerate image pull for Container Registry Enterprise Edi...	61
6.Programming model extensions	64
6.1. Feature overview	64
7.FAQ about code development	68
7.1. General FAQ	68
7.1.1. In which programming languages can I define functions...	68
7.1.2. How does Function Compute ensure code security?	68
7.1.3. What do I do if the "The Access Key ID does not exist"...	68
7.1.4. Considering that Function Compute only supports Node...	69
7.1.5. How does Function Compute automatically install depe...	69
7.2. Custom Runtime FAQ	69
7.2.1. Must the listening port of a custom runtime be the sa...	70
7.2.2. What do I do if the CAExited error occurs when the b...	70
7.2.3. What do I do if the CAFilePermission error occurs whe...	70
7.2.4. What do I do if the FunctionNotStarted error occurs w...	71
7.2.5. What do I do if the HTTP server fails to start within 3...	71
7.2.6. What format is required for the bootstrap file if I use ...	71
7.2.7. What do I do if the "Process exited unexpectedly befor...	71
7.2.8. What do I do if a 404 error occurs when I use a brow...	72
7.2.9. What do I do if a 502 error that contains the "Process...	73
7.3. Custom Container FAQ	74
7.3.1. Must the listening port of a custom container runtime ...	74
7.3.2. What do I do if the FunctionNotStarted error occurs w...	75
7.3.3. What do I do if the HTTP server fails to start within 3...	75
7.3.4. What do I do if a 502 error that contains the "Process...	75
7.3.5. What do I do if a 404 error occurs when I use a brow...	76
7.3.6. What permissions must I grant to Function Compute se...	76



# 1. Overview

This topic describes information about runtimes in multiple programming languages that are supported by .

## Background information

Runtimes provide environments that run in the execution environment for different programming languages. As a relay between and your functions, a runtime passes the values of the event and the context parameters, and the response of a function invocation. You can use a runtime provided by . You can also build a custom runtime or a custom container image.

## runtime

### Custom runtime

The following list provides examples of using a custom runtime for multiple programming languages. For more information, see [Overview](#).

- Event functions
- HTTP functions

### Custom container

The following list provides examples of using a custom container for multiple programming languages. For more information, see [Overview](#).

- Event functions
- HTTP functions

## 2.Terms

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

### Function handler

When you create a function, you must specify a handler for the function. Function Compute starts executing the function from its handler. The handler 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 is a function that is specified by a function handler. The handler function is similar to the `main()` function in on-premises 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 regular handler function that provides a segment of business 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 invoke other functions defined in the code. Except for the definition, the handler function has the same code organization logic as an on-premises function.

For more information about the handler function model for each programming language, see [Overview](#).

### Initializer function

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

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

Prior to handler functions, initializer functions are executed after Function Compute allocates functions to be executed on different 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 invoked. 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 based on trigger conditions.

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

The following sample code provides an example on 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 Function Compute. Designed by Function Compute, the data structure of the context parameter contains the runtime information of a function. The context parameter has the following data structure:

### ◦ Data 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 AccessKey pair of a user from `context.credentials` and use the temporary AccessKey pair in `context` to access other Alibaba Cloud services, such as OSS. This prevents the use of hard-coded keys in code.
- Obtain the basic information of the current execution from `context`. Such basic information includes the request ID, service name, function name, and service version or alias.

### ◦ Example

The following sample Python code provides an example on how to use the personal information in the `context` parameter to access OSS:

```
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 invocation records and the logs printed in function code into Logstores. You can make a record of function logs by executing the logging statements provided by Function Compute. This helps you debug the function and troubleshoot issues.

#### Note

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

Programming language	Built-in log printing statement of the programming language	Logging statement provided by Function Compute	References
Node.js	<code>console.log()</code>	<code>console.log()</code>	<a href="#">Print logs</a>
Python	<code>print()</code>	<code>logging.getLogger().info()</code>	<a href="#">HTTP handlers</a>
Java	<code>System.out.println()</code>	<code>context.getLogger().info()</code>	<a href="#">Print logs</a>
PHP	<code>echo "" . PHP_EOL</code>	<code>\$GLOBALS['fcLogger']-&gt;info()</code>	<a href="#">Log printing</a>
C#	<code>Console.WriteLine("")</code>	<code>context.Logger.LogInformation()</code>	<a href="#">Print logs</a>

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 the request ID.

```
# Log 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.

The following example shows the data structure of a function execution log:

```

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

## Troubleshooting

Errors in Function Compute include two types: `HandledInvocationError` and

`UnhandledInvocationError` .

- **HandledInvocationError**

Only the errors returned by the `callback` parameter 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');
}

```

The following example shows a response that contains error information.

```

{"errorMessage":"this is error"}

```

- **UnhandledInvocationError**

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

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

# 3.Go

## 3.1. Runtime environment

This topic describes the runtime environment for writing function code in Go in .

### Go runtime

supports Go 1.x. We recommend that you use Go 1.8 or later.

Name	OS	Architecture
Go 1.x	Linux	x86_64



**Notice** The Go runtime must be used on Linux and does not support the ARM64 architecture.

### SDK for Go and related tools

provides the following SDK for Go and related tools:

- **Function Compute SDK for Go**: the implementation of the programming model for the Go runtime. processes handlers based on the SDK.
- **fccontext**: the auxiliary library that provides context information for function execution.
- **examples**: the sample code for the Go runtime.

### Related information

- [Handlers](#)
- [Event handlers](#)
- [HTTP handlers](#)
- [Context](#)
- [Compilation and deployment of code packages](#)
- [Logs](#)
- [Error handling](#)
- [Lifecycle callbacks for function instances](#)

## 3.2. Handlers

This topic describes handlers and how to configure handlers in the Go runtime in .

### What is a handler?

A handler for a function in is the method that is used to process requests in the function code. When a function is invoked, uses the handler that you configure to process requests. You can configure a handler by specifying a value for the **Request Handler** parameter in the .

The handler for functions in the Go language is compiled into an executable binary file. You only need to set the Request Handler parameter of the function to the name of the executable file.

For more information about the definitions and operations of functions in , see [Manage functions](#).

### Configure a handler

When you configure a handler, make sure that you follow the configuration specifications that are described in . The configuration specifications vary based on the handler type.

Handlers are classified into event handlers and HTTP handlers. Event requests are generated by event sources, and HTTP requests are generated by HTTP triggers. For more information, see [Function types](#).

For more information about how to configure handlers, see [Event handlers](#) and [HTTP handlers](#).

## References

For more information about how to compile and deploy your code package to , see [Compilation and deployment of code packages](#).

## 3.3. Event handlers

This topic describes the structure and characteristics of event handlers for Go.

### Sample code for using an event handler

Import an official SDK library named `aliyun/serverless/fc-runtime-go-sdk/fc` for Go and implement the `handler` and `main` functions. Sample code:

```
package main
import (
    "fmt"
    "context"
    "github.com/aliyun/fc-runtime-go-sdk/fc"
)
type StructEvent struct {
    Key string `json:"key"`
}
func HandleRequest(ctx context.Context, event StructEvent) (string, error) {
    return fmt.Sprintf("hello, %s!", event.Key), nil
}
func main() {
    fc.Start(HandleRequest)
}
```

The value of the `event` input parameter is a JSON string that contains the `key` information, as shown in the following sample code:


```
{
  "key": "value"
}
```

The following content describes the code snippets in the sample code:

- `package main` : the `main` package. Each Go application contains a main package.
- `import` : imports Function Compute dependencies. You need to import the following dependencies:
  - `github.com/aliyun/fc-runtime-go-sdk/fc` : the core library of SDK for Go.
  - `context` : the context object of SDK for Go.
- `func HandleRequest(ctx context.Context, event StructEvent) (string, error)` : the handler

used to process event requests. The handler contains the code to be run and involves the following parameters:

- `ctx context.Context` : provides the runtime context information when a function is invoked in . For more information, see [Context](#).
- `event StructEvent` : specifies the data to be passed in when the function is invoked. Multiple data types are supported.
- `string, error` : returns the STRING error type and the error message. For more information, see [Error handling](#).
- `return fmt.Sprintf("Hi,%s ! ", event.Key), nil` : returns `hello` and the value of the `event` parameter that is passed in. If `nil` is returned, no error occurs.
- `func main()` : the entry point for running function code in . A Go application must contain the `main` function. You can add `fc.Start(HandleRequest)` to run your application in .

 **Notice** The methods used to start an HTTP handler and an event handlers are different. To start an event handler, invoke the `fc.Start` function in the `main` function. To start an HTTP handler, invoke the `fc.StartHttp` function in the `main` function.

## Signatures for event handlers

The following signatures for event handlers are valid:

- `func ()`
- `func () error`
- `func (InputType) error`
- `func () (OutputType, error)`
- `func (InputType) (OutputType, error)`
- `func (context.Context) error`
- `func (context.Context, InputType) error`
- `func (context.Context) (OutputType, error)`
- `func (context.Context, InputType) (OutputType, error)`

The `InputType` and `OutputType` objects are compatible with the `encoding/json` standard library.

You must use an event handler based on the following rules:

- The handler must be a function.
- The handler can contain up to two input parameters. If the handler contains two input parameters, the first input parameter must be `context.Context` .
- The handler can return up to two values. If only one value is returned, the value must indicate the `error` type. If two values are returned, the second value must indicate the `error` message.

Sample code for event handlers for Go:

- [event-struct.go](#): the sample code for a handler whose `event` object is of the STRUCT type.
- [event-string.go](#): the sample code for a handler whose `event` object is of the STRING type.
- [event-map.go](#): the sample code for a handler whose `event` object is of the `map[string]interfa`

`ce{} type.`

For more information about the sample code for other handlers, visit [examples](#).

## Context

For more information about the context, see [Context](#).

# 3.4. HTTP handlers

You can use HTTP handlers to efficiently process HTTP requests. When you invoke a function, runs the handler that you specify in the function code to process requests. This topic describes the structure and characteristics of HTTP handlers for Go.

## Sample code for using an HTTP handler


Import an official SDK library named `aliyun/serverless/fc-runtime-go-sdk/fc` for Go and implement the `handler` and `main` functions. Sample code:

```
package main
import (
    "context"
    "fmt"
    "net/http"
    "io/ioutil"
    "github.com/aliyun/fc-runtime-go-sdk/fc"
)
func HandleHttpRequest(ctx context.Context, w http.ResponseWriter, req *http.Request) error {
    body, err := ioutil.ReadAll(req.Body)
    if err != nil {
        w.WriteHeader(http.StatusBadRequest)
        w.Header().Add("Content-Type", "text/plain")
        w.Write([]byte(err.Error()))
        return nil
    }
    w.WriteHeader(http.StatusOK)
    w.Header().Add("Content-Type", "text/plain")
    w.Write([]byte(fmt.Sprintf("Hi,%s! \n", body)))
    return nil
}
func main() {
    fc.StartHttp(HandleHttpRequest)
}
```

The following content describes the code snippets in the sample code:

- `package main` : the `main` package. Each Go application contains a main package.
- `import` : imports dependencies. You need to import the following dependencies:
  - `github.com/aliyun/fc-runtime-go-sdk/fc` : the core library of SDK for Go.
  - `context` : the context object of SDK for Go.
  - `net/http` : provides the `Request` and `ResponseWriter` methods in the HTTP package for the HTTP handler.

- `HandleHttpRequest(ctx context.Context, w http.ResponseWriter, req *http.Request) error` : the HTTP handler used to process HTTP requests. The handler contains the code to be run and involves the following parameters:
  - `ctx context.Context` : provides the runtime context information when a function is invoked. For more information, see [Context](#).
  - `w http.ResponseWriter` : the response method of the HTTP handler. You can call this method to set the status code, response header, and response body. For more information, see the [Response method](#) section.
  - `req *http.Request` : the request method of the HTTP handler. You can call this method to set the request line, request header, and request body. For more information, see [Request struct](#).
  - `w.WriteHeader(http.StatusOK)` : Specify the HTTP status code of the response.
  - `w.Header().Add("Content-Type", "text/plain")` : Specify the response header.
  - `w.Write([]byte(fmt.Sprintf("Hi,%s! \n", body)))` : Specify the response body.
  - `return nil` : returns a simple error message. If `nil` is returned, no error occurs. If an error message is returned, a function error occurs.
- `func main()` : the entry point for running function code in . A Go application must contain the `main` function. To run your application in , invoke the `fc.StartHttp(HandleHttpRequest)` function in the `main()` function.

 **Notice** The methods used to start an HTTP handler and an event handlers are different. To start an event handler, invoke the `fc.Start` function in the `main` function. To start an HTTP handler, invoke the `fc.StartHttp` function in the `main` function.

## Definition

An HTTP handler for Go is defined based on [Handler interface](#) for HTTP requests in the Go standard library but contains an additional `context` parameter. Syntax of an HTTP handler:

```
function(ctx context.Context, w http.ResponseWriter, req *http.Request) error
```

An HTTP handler consists of the following three objects:

- `context` : provides the runtime context information when a function is invoked in . For more information, see [Context](#).
- `http.Request` : the request struct. For more information, see [Request struct](#).
- `http.ResponseWriter` : the response method. For more information, see [Response method](#).

## Request struct

`http.Request` is the HTTP object defined in the Go standard library. The following table describes the parameters that are supported by the `http.Request` object.

Parameter	Type	Description
Method	String	The HTTP request method, such as PUT, POST, or DELETE.
URL	*url.URL	The request URL.

Parameter	Type	Description
Header	http.Header	The key-value pair of the HTTP request header.
Body	io.ReadCloser	The request struct.
ContentLength	Int64	The data length in the request struct.

## Response method

The following sample code shows the three methods declared by `http.ResponseWriter`:

```
type ResponseWriter interface {
    Header() Header
    Write([]byte) (int, error)
    WriteHeader(statusCode int)
}
```

Note:

- `WriteHeader(statusCode int)` : sets the status code.
- `Header() Header` : sets the response header.
- `Write([]byte) (int, error)` : sets the response body.

## Context

For more information about the context, see [Context](#).

# 3.5. Context

This topic describes the context in the Go runtime in and provides sample code.

## What is context?

When runs your function code, it passes the context object `context.Context` to the handler. This object contains information about invocations, services, functions, tracing analysis, and runtime environments.

You can use the context object as an input parameter for event handlers and HTTP handlers. The format and content of the context input parameter for event handlers and HTTP handlers are the same. The following table describes the parameters that are supported by the context object.

Context

Parameter	Description
<b>Variables</b>	
RequestID	The unique ID of the request for invoking the function. You can record the ID for troubleshooting if an error occurs.

Parameter	Description
Credentials	The temporary AccessKey pair that obtains by assuming your service-linked role. The temporary AccessKey pair is valid for 5 minutes. You can use credentials in your code to access related services such as Object Storage Service (OSS). This way, you do not need to write your AccessKey pair in the function code. For more information, see <a href="#">Grant permissions across Alibaba Cloud accounts by using a RAM role</a> .
Function	The basic information of the invoked function, such as the name, handler, memory, and timeout period of the function.
Service	The information about the service to which the function belongs, such as the name, the related project and Logstore in Log Service, the version, and the alias of the service. The <code>qualifier</code> parameter specifies the version or alias of the service. The <code>version_id</code> parameter specifies the version of the service.
Region	The ID of the region in which the function is invoked. For example, if the function is invoked in the China (Shanghai) region, the region ID is cn-shanghai. For more information, see <a href="#">Endpoints</a> .
AccountId	The ID of the Alibaba Cloud account that is used to invoke the function.
<b>Method</b>	
deadline	The timeout period of function execution. The value is in the UNIX timestamp format. Unit: milliseconds.

For more information about the complete data structure, see [context.go](#).

## Sample code

### Sample code for displaying the context information

Add the `context` parameter to the `handler` of your function. passes the variable information described in the preceding [Context](#) table to the `context` parameter. Then, import the `aliyun/fc-runtime-go-sdk/fccontext` package and call the `fccontext.FromContext` method to obtain `fccontext`.

```

package main
import (
    "context"
    "encoding/json"
    "log"
    "github.com/aliyun/fc-runtime-go-sdk/fc"
    "github.com/aliyun/fc-runtime-go-sdk/fccontext"
)
func main() {
    fc.Start(echoContext)
}
func echoContext(ctx context.Context) (string, error) {
    fctx, _ := fccontext.FromContext(ctx)
    log.Println(fctx.AccountId)
    log.Printf("%#v\n", fctx)
    res, _ := json.Marshal(fctx)
    return string(res), nil
}

```

## Sample code for obtain the remaining execution time of a function

The following sample code provides an example on how to use the `deadline` parameter to obtain the remaining execution time of a function:

```

package main
import (
    "context"
    "fmt"
    "log"
    "time"
    "github.com/aliyun/fc-runtime-go-sdk/fc"
)
func LongRunningHandler(ctx context.Context) (string, error) {
    deadline, _ := ctx.Deadline()
    fmt.Printf("now: %s\ndeadline: %s\n", time.Now().String(), deadline.String())
    deadline = deadline.Add(-100 * time.Millisecond)
    timeoutChannel := time.After(time.Until(deadline))
    for {
        select {
        case <-timeoutChannel:
            return "Finished before timing out.", nil
        default:
            log.Print("hello!")
            time.Sleep(50 * time.Millisecond)
        }
    }
}
func main() {
    fc.Start(LongRunningHandler)
}

```

## 3.6. Compilation and deployment of code packages

Go is a static compilation language. If you use Go, you must locally compile and package your code in the ZIP format. This topic describes how to package official SDK for Go with the code that you compile.

### Prerequisites

The **Go** programming language is installed, supports Go 1.x. We recommend that you use Go 1.8 or later.

### Compile and package code on Linux or macOS

1. Download SDK for Go.

```
go get github.com/aliyun/fc-runtime-go-sdk/fc
```

2. Compile files.

```
GOOS=linux go build main.go
```

 **Note** The `main.go` file is only for reference. Replace it with your actual file name.

Set the `GOOS` parameter to `linux` so that the compiled executable files are compatible with the Go runtime of . Pay special attention to this when you compile your files in a non-Linux environment.

Note:

- On Linux, we recommend that you use pure static compilation and set the `CGO_ENABLED` parameter to 0. This way, the executable files do not require external dependencies such as the `libc` library. This prevents incompatibility between the compilation environment and the dependencies of the Go runtime. Sample code:

```
GOOS=linux CGO_ENABLED=0 go build main.go
```

- If your machine uses M1 macOS or other ARM architectures, set the `GOARCH` parameter to `amd64` to allow cross-platform compilation. Sample code:

```
GOOS=linux GOARCH=amd64 go build main.go
```

3. Package the files.

```
zip fc-golang-demo.zip main
```

### Compile and package files on Windows

1. Compile the executable files.
  - i. Press the **Win+R** keys to open the Run dialog box.
  - ii. Enter `cmd` and press the **Enter** key.
  - iii. In the command prompt window, run the following commands:

```
set GOOS=linux
set GOARCH=amd64
go build -o main main.go
```

2. Use the **build-fc-zip** tool to package the files.

i. Run the **go get** command to download the build-fc-zip tool from GitHub.

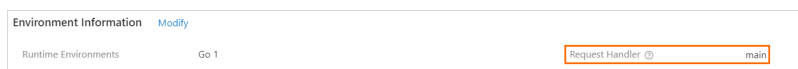
```
go get -u github.com/aliyun/fc-runtime-go-sdk/cmd/build-fc-zip
```

ii. Use the build-fc-zip tool to package the files. If you use the default installation method for Go, the tool is installed in the `%USERPROFILE%\go\bin` directory.

```
~\go\bin\build-fc-zip.exe -output main.zip main
```

## Configure function handlers in

1. **Create a service.**
2. **Create a function.** Set the Runtime Environments to Go 1.



Go is a compilation language. You need to upload the executable binary file after the file is locally compiled. When you configure the **function handler** for a Go function in the , set the **Request Handler** parameter to `[File name]` . The file name is the name of the compiled binary file. When the function is invoked, executes the binary file.

The following figure shows an example of the ZIP package. In this case, if you set the Request Handler parameter to `main` , executes the `main` binary file in the root directory of the ZIP package.

```
→ code zip -sf fc-golang-demo.zip
Archive contains:
  main
Total 1 entries (6657596 bytes)
```

If the compiled binary file is not stored in the root directory of the ZIP package but in another directory such as `bin/`, you need to set the **Request Handler** parameter to `bin/main` .

```
→ code zip -sf fc-golang-demo.zip
Archive contains:
  bin/main
Total 1 entries (6657596 bytes)
```

For more information about other deployment methods, see the following topics:

- [Use SDKs to deploy functions](#)
- [Use Serverless Devs to deploy functions](#)

## 3.7. Logs

This topic describes how to display logs in the Go runtime environment.

### Display logs

To display logs, you can use the methods provided by the built-in logging library of Go, called `log`, or other Logstores that are used to write logs to `stdout` or `stderr`.

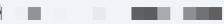
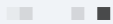
The logs displayed by using the methods provided by the `log` library contain the date and time information. The following sample code provides an example on how to display logs by using the methods provided by the `log` library:

```
package main
import (
    "log"
    "aliyun/serverless/fc-runtime-go-sdk/fc"
)
func HandleRequest() (string, error) {
    log.Println("hello world")
    return "hello world", nil
}
func main() {
    fc.Start(HandleRequest)
}
```

The following logs are displayed:

```
FC Invoke Start RequestId: a15f8f85-9ed3-47c9-a61c-75678db61831
2022/02/17 04:29:02.228870 hello world
FC Invoke End RequestId: a15f8f85-9ed3-47c9-a61c-75678db61831
```

The Go runtime records the Start and End lines and the execution summary for each function invocation.

Execution Summary			
Request ID	c32efff5 	Code Check Value	715 
Function Execution Time	1.62 ms	Function Billing Time	2 ms
Function Memory	4096 MB	Actual Used Memory	74.39 MB

The following table describes the parameters in the execution summary.

Parameter	Description
Request ID	The unique ID of the request for invoking the function.
Code verification code	The verification code of the code package used for invoking the function.
Function execution time	The time that the function handler actually takes to run.
Function billing time	The amount of time in which you are charged for invoking the function.
Function memory	The amount of memory allocated for the function.
Actual memory usage	The maximum amount of memory that is actually used by the function.

## 3.8. Error handling

This topic describes how to handle errors in the Go runtime environment.

If an error occurs, the HTTP response header contains `X-Fc-Error-Type` such as `X-Fc-Error-Type: UnhandledInvocationError`. For more information about error types of , see [Error handling](#).

can return errors in the following ways:

- Specify the error message to return in the handler function. Sample code:

```
package main
import (
    "errors"
    "fmt"
    "aliyun/serverless/fc-runtime-go-sdk/fc"
)
func HandleRequest() error {
    fmt.Println("hello world")
    return errors.New("something is wrong")
}
func main() {
    fc.Start(HandleRequest)
}
```

After the function is invoked, the following response is returned:

```
{
  "errorMessage": "something is wrong!",
  "errorType": "errorString"
}
```

- Use panic to return an error message. Sample code:

```
package main
import (
    "fmt"
    "aliyun/serverless/fc-runtime-go-sdk/fc"
)
func HandleRequest() error {
    fmt.Println("hello world")
    panic("Error: something is wrong")
    return nil
}
func main() {
    fc.Start(HandleRequest)
}
```

After the function is invoked, the following response is returned. In this example, some stack information is omitted.

```
{
  errorMessage: 'Error: something is wrong',
  errorType: 'string',
  stackTrace: [
    {
      path: 'aliyun/serverless/fc-runtime-go-sdk/fc/errors.go',
      line: 39,
      label: 'fcPanicResponse'
    },
    {
      path: 'aliyun/serverless/fc-runtime-go-sdk/fc/function.go',
      line: 84,
      label: '(*Function).Invoke.func1'
    },
    ...
    ...
    ...
    { path: 'code/main.go', line: 22, label: 'main' },
    { path: 'runtime/proc.go', line: 255, label: 'main' },
    null
  ]
}
```

- Use error handling code such as `log.Fatal` that contains `os.Exit(1)` .



**Notice** We recommend that you do not use this method. If you use this method, no error message or stack information is returned upon an exit.

```
package main
import (
    "fmt"
    "log"
    "aliyun/serverless/fc-runtime-go-sdk/fc"
)
func HandleRequest() error {
    fmt.Println("hello world")
    log.Fatal("something is wrong")
    return nil
}
func main() {
    fc.Start(HandleRequest)
}
```

After the function is invoked, the following response is returned:

```
{
  errorMessage: 'Process exited unexpectedly before completing request (duration: 0ms, maxMemoryUsage: 8MB) '
}
```

## 3.9. Lifecycle callbacks for function instances

This topic describes how to invoke lifecycle callbacks for function instances in the Go runtime environment.

### Context

After you implement and configure a lifecycle callback for a function instance, invokes the callback if a related lifecycle event for the instance occurs. A function instance involves the following lifecycle callbacks: initializer, PreFreeze, and PreStop. The Go runtime supports only the initializer callback. For more information, see [Lifecycle callbacks for function instances](#).

### Initializer callback

The initializer callback is invoked after the function instance is started but before the handler runs. ensures that the initializer callback can be successfully invoked at most once within the lifecycle of a function instance. For example, if the initializer callback fails, the system keeps retrying until the initializer callback is successfully invoked, and then runs your handler. Make sure that the initializer callback can be successfully invoked when you implement the initializer callback.

The initializer callback contains only the context input parameter and can be invoked in the same way as a handler.

```
function(ctx context.Context)
```

### Use the initializer callback

To use the initializer callback, perform the following steps:

1. Use `fc.RegistryInitializerFunction(Init)` in your application code to register an initializer callback.

Sample code:


```

package main
import (
    "context"
    "log"
    "github.com/aliyun/fc-runtime-go-sdk/fc"
)
var (
    count int = 1
)
func Init(ctx context.Context) {
    count += 1000
}
func main() {
    fc.RegisterInitializerFunction(Init)
    fc.Start(HandleRequest)
}
func HandleRequest() (int, error) {
    count += 1
    log.Println("count: ", count)
    return count, nil
}

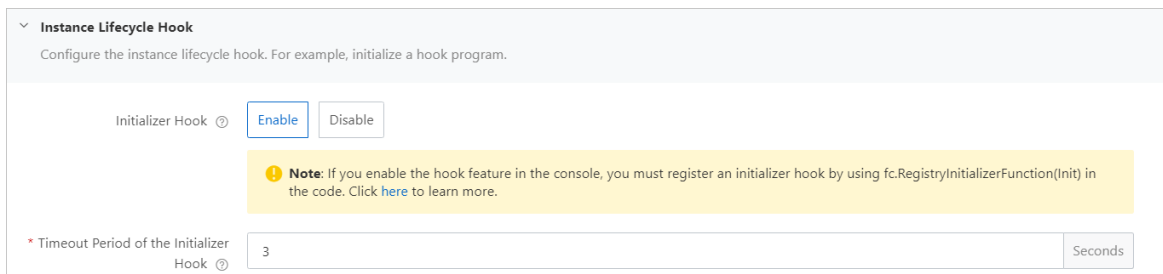
```

The following content describes the code snippets in the sample code:

- o `func Init(ctx context.Context)` : the initializer callback. The `ctx context.Context` parameter specifies the runtime information for invoking a function in . For more information, see [Context](#).
- o `func main()`: the entry point for running function code in . A Go application must contain the `main` function. You can add `fc.Start(HandleRequest)` to your code to specify the method to execute the handler, and add `fc.RegisterInitializerFunction(Init)` to register the initializer callback.

 **Notice** The preceding sample code applies only to an event handler. If you use an HTTP handler, replace `fc.Start(HandleRequest)` in the sample code with `fc.StartHttp(HandleRequest)` .

2. Configure the initializer callback on the function configuration page of the , as shown in the following figure.



Instance Lifecycle Hook

Configure the instance lifecycle hook. For example, initialize a hook program.

Initializer Hook ⓘ Enable Disable

**Note:** If you enable the hook feature in the console, you must register an initializer hook by using `fc.Registry/InitializerFunction(Init)` in the code. Click [here](#) to learn more.

\* Timeout Period of the Initializer Hook ⓘ  Seconds

For more information, see [Manage functions](#).

## Sample code

provides sample code for invoking the initializer callback. The sample code provides an example on how to use the initializer callback in the Go runtime environment to initialize a MySQL connection pool. In the sample code, the MySQL database is configured by using an environment variable of the function. For more information, see the `s.yaml` file. The initializer callback obtains the database configuration based on the environment variable, creates a MySQL connection pool, and then tests the connectivity.

For more information, see [go-initializer-mysql](#).

# 4. Custom Runtime

## 4.1. Overview

This topic describes the background information, container environment, basic principles, configuration requirements, common request headers, and log formats of a custom runtime.

### Background information

Custom runtimes allow you to define runtime environments. In a custom runtime, you can define the runtime environment based on your requirements. For example, you can perform the following operations:

- Use a specified programming language, such as Lua.
- Use a runtime environment of the specified version for a programming language, such as Node.js 16.

If you want to create a custom runtime by using a language that is not a built-in language of the custom runtime, you must compress the parser or runtime of the language and your code file into a package and deploy the package in . For example, if the runtime environment is Node.js 16, you must download the interpreter of Node.js 16, compress the interpreter and your code file into a package, and then deploy the package in .

A custom runtime supports the following built-in languages of the specified versions. You can create custom runtimes of the following languages without the need to install third-party interpreters:

- Python 3.7.4
- Node.js 10.16.2
- Ruby 2.7
- PowerShell 7.1.0
- Nginx 1.10.3
- OpenJDK 1.8.0\_232
- PHP 7.4.12

Built-in extensions for PHP

bcmath	calendar	Core
ctype	curl	date
dom	exif	FFI
fileinfo	filter	ftp
gd	gettext	hash
iconv	imagick	imap
intl	json	libxml
mbstring	mcrypt	memcached
mysqli	mysqlnd	openssl
pcntl	pcre	PDO

pdo_mysql	pdo_pgsql	pdo_sqlite
pgsql	Phar	posix
protobuf	readline	redis
Reflection	session	shmop
SimpleXML	soap	sockets
sodium	SPL	sqlite3
standard	swoole	sysvmsg
sysvsem	sysvshm	tokenizer
xml	xmlreader	xmlrpc
xmlwriter	xsl	Zend OPcache
zip	zlib	

For more information about the built-in software of a custom runtime, visit [GitHub](#).

## Container environment

A custom runtime runs in the following container environment:

- Operating system: Debian 9
- User permissions:
  - If functions are created at and after 00:00:00 December 1, 2021, the functions must be executed by the root user.
  - If functions are created before 00:00:00 December 1, 2021, the functions must be executed by a non-root user.
- Directory permissions:
  - If functions are created at and after 00:00:00 December 1, 2021, all the directories are writable.
  - If functions are created before 00:00:00 December 1, 2021, only the `/tmp` directory is writable.

## Basic principles

To create a custom runtime, you must create an executable file named *bootstrap* to store the command that is used to start an HTTP server whose default port is port 9000. Compress the bootstrap file and your code file into a ZIP package, and then create a custom runtime function by using the ZIP package as the code package.

For example, you can name the deployment package of a function *function.zip*. The following examples show the files contained in the deployment package and the content of the *bootstrap* file based on the programming language that is used to develop the function.

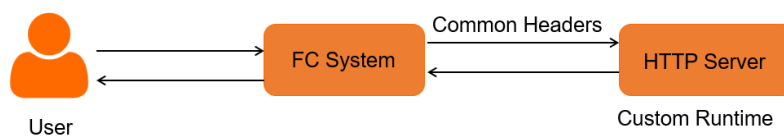
[Java or Spring Boot](#)[Python](#)[Node.js](#)[PHP](#)

**Note** If an executable *bootstrap* file is found in the code package of a function, executes the file when a user invokes the function. If no bootstrap file is found in the code package of a function or the file is not executable, Function Compute returns an error when a user invokes the function.

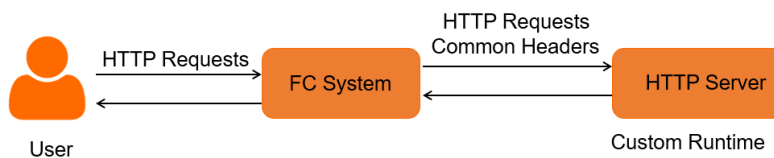
To cold start the custom runtime, calls the *bootstrap* file by default to start your custom HTTP server. Then, the HTTP server takes over all requests from , including the invocations of event and HTTP functions.

Before you develop the logic of a function, you must determine whether the function is an event function or an HTTP function. The following figures show how an event function and an HTTP function work.

- Event function



- HTTP function



## Configuration requirements on an HTTP server

When you create an HTTP server, make sure that the following requirements are met:

- Services that are started in a custom runtime must listen on `0.0.0.0:CAPort` or `*:CAPort` . If you configure the services to listen on `127.0.0.1:CAPort` , a request times out and the following error is returned:

```
{
  "ErrorCode": "FunctionNotStarted",
  "ErrorMessage": "The CA's http server cannot be started:ContainerStartDuration:2500000000. Ping CA failed due to: dial tcp 21.0.5.7:9000: getsockopt: connection refused Logs : 2019-11-29T09:53:30.859837462Z Listening on port 9000"
}
```

The default listening port (CAPort) of a custom runtime is port 9000. If a custom runtime uses the default listening port, the listening port of its HTTP server must be port 9000. If the listening port of the custom runtime is port 8080, the listening port of its HTTP server must be port 8080.

- If the *bootstrap* file of the custom runtime is a Shell script, add `#!/bin/bash` . Otherwise, the following error is returned:

```
{
  "ErrorCode": "CAExited",
  "ErrorMessage": "The CA process either cannot be started or exited: ContainerStartDuration: 25037266905. CA process cannot be started or exited already: rpc error: code = 106 desc = ContainerStartDuration: 250000000000. Ping CA failed due to: dial tcp 21.0.7.2:9000: i/o timeout Logs : 2019-11-29T07:27:50.759658265Z panic: standard_init_linux.go:178: exec user process caused \"exec format error\""
}
```

If the bootstrap file is a binary executable file, such as a binary file compiled by using Go or C++, you do not need to add `#!/bin/bash`.

- The *bootstrap* file of a custom runtime must be executable. Otherwise, the following error is returned:

```
{
  "ErrorCode": "CAFilePermission",
  "ErrorMessage": "The CA process cannot be started due to bootstrap file don't have execute permissions"
}
```

You can use one of the following methods to resolve the error:

- Run the `chmod +x bootstrap` or `chmod 755 bootstrap` command before you compress files.
- If you use the Windows operating system, convert the format of the *bootstrap* file to UNIX.
- You must set the Connection parameter to Keep-Alive and the request timeout period to 15 minutes or longer. The following sample code provides an example on how to specify the values:

```
// In this example, the express framework for Node.js is used.
var server = app.listen(PORT, HOST);
server.timeout = 0; // never timeout
server.keepAliveTimeout = 0; // keepalive, never timeout
```

- The HTTP server must complete the startup within 120 seconds.

## Common request headers in Function Compute

The following table describes the common request headers that a custom runtime may receive from . If you want to access other Alibaba Cloud services, you may need to use the request headers that specify a temporary AccessKey pair. If you migrate existing applications to Function Compute, skip the following headers.

### Note

- Both event functions and HTTP functions contain common request headers.
- generates common request headers that contain basic information about permissions and functions.

Header	Description
x-fc-request-id	The ID of the request.
x-fc-access-key-id	The temporary AccessKey ID.

Header	Description
x-fc-access-key-secret	The temporary AccessKey secret.
x-fc-security-token	The temporary security token.
x-fc-function-handler	The handler of the function. This value is not required if the custom runtime is a function.
x-fc-function-memory	The maximum memory that the function can use.
x-fc-function-initializer	The handler of the initializer function. This value is not required if the custom runtime is a function.
x-fc-initialization-timeout	The timeout period of the initializer function.
x-fc-region	The region of the function.
x-fc-account-id	The UID of the function owner.
x-fc-qualifier	The service version or alias that is specified when the function is invoked. For more information, see <a href="#">View the version for a function</a> .
x-fc-version-id	The version of the function.
x-fc-service-name	The name of the Function Compute service to which the function belongs.
x-fc-service-logproject	The Log Service project that is configured for the Function Compute service to which the function belongs.
x-fc-service-logstore	The Log Service Logstore that is configured for the Function Compute service to which the function belongs.

Header	Description
x-fc-control-path	<p>The request type of the function.</p> <p>For a custom runtime or custom container, you can determine whether a request is used to invoke an HTTP function or an event function based on the value of this parameter. Valid values:</p> <ul style="list-style-type: none"> <li><i>/invoke</i>: The request is used to invoke an event function.</li> <li><i>/http-invoke</i>: The request is used to invoke an HTTP function. adds common headers to your request that contains the request path, request body, and request headers, and forwards the request to the custom runtime or custom container. Then, Function Compute forwards the response headers and body returned by the custom runtime or custom container to the client.</li> <li><i>/initialize</i>: The request is automatically initiated by to invoke an initializer function when a runtime environment is created and runs for the first time. Similar to a class constructor, the initializer function is invoked only once in the lifecycle of a container.</li> </ul>

## Function log formats

We recommend that you use [Log Service](#) when you create a service in Function Compute. Then, all the generated stdout logs are stored in the specified Log Service project. For more information, see [Configure the logging feature](#).

(Optional)If invokes a function in a runtime that is not a custom runtime or a custom container, and the request contains the `x-fc-log-type = "Tail"` header, the content that contains the `x-fc-log-result` header in the response is the log generated when the function is executed. The maximum size of a log is 4 KB. You can view the log in function execution results in the . If you want to view the logs of a custom runtime in function execution results in the console, you must record the start and end logs of requests in the code.

Log content	Required	Code format
-------------	----------	-------------

Log content	Required	Code format
A runtime starts	No <div> <p><b>Note</b> The cold start flag of the function is recorded.</p> </div>	<pre>FunctionCompute \${runtime} runtime initied.</pre> <div> <p><b>Note</b> You can specify a custom value for <code>\${runtime}</code>. We recommend that you do not specify an official language name, such as Node.js, Python, or PHP.</p> </div>
Invocation starts	Yes	<pre>FC Invoke Start RequestId: \${RequestId}</pre>
Invocation ends	Yes	<pre>FC Invoke End RequestId: \${RequestId}</pre>
Initialization starts	No <div> <p><b>Note</b> For an initializer function, this log must be recorded.</p> </div>	<pre>FC Initialize Start RequestId: \${RequestId}</pre>
Initialization ends	No <div> <p><b>Note</b> For an initializer function, this log must be recorded.</p> </div>	<pre>FC Initialize End RequestId: \${RequestId}</pre>

In addition to the preceding information, we recommend that you include request IDs in your logs in the `$utcdatetime (yyyy-MM-ddTHH:mm:ss.fff) $requestId [$Level] $message` format for future troubleshooting.

**Note** The API operation to specify the log level varies based on the programming language. For more information, see [Log records](#).

## References

For more information about the limits of custom runtimes, see [Limits](#).

## 4.2. Basic principles

For a custom runtime, the code file in the ZIP format is an HTTP server program. This topic describes the basic principles of cold starts of a custom runtime and the requirements for the configurations of HTTP servers.

## Basic principles

For a custom runtime, the code file in the ZIP format is an HTTP server program. You need to only configure the **Startup Command** and **Startup Parameter** parameters for the function to start the HTTP server. When performs a cold start in a custom runtime, the **Startup Command** and **Startup Parameter** parameters are used to start your custom HTTP server. The HTTP server takes over all requests from . The default port of an HTTP server is 9000. If you use another port, such as 8080, for the HTTP server, you can set the listening port in the function configurations to 8080.

For example, the name of the code package of a function is *function.zip*. The following examples show the files that are contained in the package and the **Startup Command** and **Startup Parameter** parameters based on the programming language that is used to develop the function.

Java 8 or Spring Boot

Python 3.7

Node.js 10

PHP 7.4

```
.
├─ demo.jar
```

```
customRuntimeConfig:
  command:
    - java
  args:
    - '-jar'
    - 'demo.jar'
```

**Note** `customRuntimeConfig` specifies the custom startup commands for the function. `command` specifies the startup commands for the container. `args` specifies the startup parameters. concatenates the content in `command` and `args` to form a complete startup command.

## Requirements on HTTP server configurations

When you create an HTTP server, make sure that the following requirements are met:

- Services that are started in a custom runtime must listen on `0.0.0.0:CAPort` or `*:CAPort` . If you use the `127.0.0.1:CAPort` port, a request times out, and the following error is returned:

```
{
  "ErrorCode": "FunctionNotStarted",
  "ErrorMessage": "The CA's http server cannot be started:ContainerStartDuration:250000000. Ping CA failed due to: dial tcp 21.0.XX.XX:9000: getsockopt: connection refused Logs : 2019-11-29T09:53:30.859837462Z Listening on port 9000"
}
```

The default listening port of a custom runtime is port 9000. If a custom runtime uses the default listening port, the listening port of its HTTP server must be port 9000. If the listening port of the custom runtime is port 8080, the listening port of its HTTP server must be port 8080.

- You must set the connection to the keep-alive mode and the request timeout period to 24 hours (the maximum function running period) or longer. Sample code:

```
// In this example, the express framework for Node.js is used.
var server = app.listen(PORT, HOST);
server.timeout = 0; // never timeout
server.keepAliveTimeout = 0; // keepalive, never timeout
```

- The HTTP server must be started within 120 seconds.

## 4.3. CDN

### Context

### Procedure

## 4.4. Event functions


This topic describes how to check and invoke event functions in a custom runtime and provides examples.

### Context



In a custom runtime, Function Compute forwards the common headers, request body, POST method, `/invoke`, and `/initialize` to HTTP servers. You can build an input parameter similar to `context` in an official runtime based on the information in common headers and an input parameter similar to `event` in an official runtime based on the request body of the invoked function.

### Function invocation

When you invoke an event function, only the `/invoke` and `/initialize` paths need to be implemented.

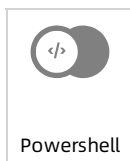
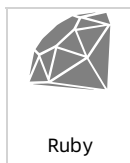
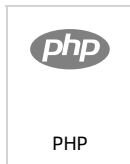
 **Notice** If you do not configure the `Initializer` parameter when you create a function, `/initialize` does not need to be implemented. Even if `/initialize` is implemented by the HTTP server, you cannot invoke or execute the `/initialize` logic in the code.

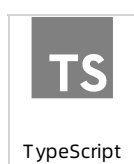
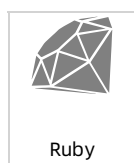
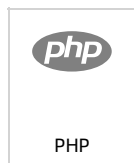
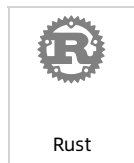
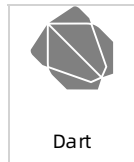
Path	Request	Expected response
------	---------	-------------------

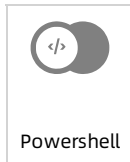
Path	Request	Expected response
POST /invoke	<ul style="list-style-type: none"> <li>Request body: the function input, which is the payload that you specify when you call the <code>InvokeFunction</code> operation.</li> <li>Request header: common request headers. For more information, see <a href="#">Common request headers in Function Compute</a>.</li> </ul> <div>  <b>Notice</b> Content-Type is set to application/octet-stream.         </div>	<p>Response body: the return value of the function handler.</p> <ul style="list-style-type: none"> <li>StatusCode             <ul style="list-style-type: none"> <li>200: succeeded</li> <li>404: failed</li> </ul> </li> <li>Header: x-fc-status             <ul style="list-style-type: none"> <li>200: succeeded</li> <li>404: failed</li> </ul> </li> </ul> <p>The <code>x-fc-status</code> field in the response header is used to report the invocation result of the function to Function Compute.</p> <ul style="list-style-type: none"> <li>If you do not configure the <code>x-fc-status</code> field, Function Compute considers the invocation successful by default. However, an exception may occur in your function and the exception is not reported to Function Compute. In this case, Function Compute considers the function invocation successful. This may not affect the business logic, but affects the monitoring observability.</li> <li>If you configure the <code>x-fc-status</code> field, the function execution failure is reported to the function system by using the <code>x-fc-status</code> field and the error stack information is recorded to logs if an exception occurs in the function.</li> </ul> <div>  <b>Note</b> We recommend that you specify both the StatusCode and x-fc-status fields in the HTTP response.         </div>

Path	Request	Expected response
(Optional) POST <code>/initialize</code>	<ul style="list-style-type: none"><li>Request body: none.</li><li>Request header: common request headers. For more information, see <a href="#">Common request headers in Function Compute</a>.</li></ul>	<p>Response body: the return value of the <code>initializer</code> function.</p> <p>Status Code</p> <ul style="list-style-type: none"><li>200: succeeded</li><li>404: failed</li></ul>

## Examples



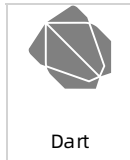




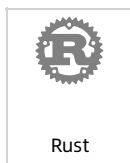
Powershell



Lua



Dart



Rust

## FAQ

- Must the listening port of a custom runtime be the same as that of the HTTP server of the custom runtime?
- What do I do if the CAExited error occurs when the bootstrap file of a custom runtime is a shell script?
- What do I do if the CAFilePermission error occurs when I do not have the permissions to execute the bootstrap file of a custom runtime?
- What do I do if the FunctionNotStarted error occurs when I invoke a third-party service in a service started in a custom runtime?
- What do I do if the HTTP server fails to start within 30s because the HTTP server implementation takes a long time?
- What format is required for the bootstrap file if I use the Windows operating system?
- What do I do if the "Process exited unexpectedly before completing request" message is returned?
- What do I do if a 404 error occurs when I use a browser or the cURL tool to access a function?

## 4.5. HTTP functions


This topic describes how to determine and invoke HTTP functions in a custom runtime and provides sample code and limits.

### Context

Function Compute forwards your requests, including the method, path, query, request headers, request body, and common headers generated by Function Compute to the HTTP server. HTTP functions allow you to migrate existing HTTP web applications.

### Invocations

The invocation of an HTTP function is like the invocation of a web API. You can initiate an invocation request by using cURL, Postman, or a browser. For more information about the solution to the case where the relevant function is downloaded in a forced manner when you use a browser to access an HTTP trigger, see [Solution](#).

 **Notice** If the `initializer` parameter is not specified when a function is created, `/initialize` does not need to be implemented. In this case, even if `/initialize` is implemented by the HTTP server, the `/initialize` logic in the code cannot be invoked or executed.


Path	Request	Expected response
(Optional) POST <code>/initialize</code>	Request body: none. Request header: common request headers. For more information, see <a href="#">Common request headers in Function Compute</a> .	Response body: the return value of the function <code>initializer</code> . StatusCode <ul style="list-style-type: none"> <li>• 200: succeeded.</li> <li>• 404: failed.</li> </ul>

Header	Description
(Optional) x-fc-base-path	If a custom domain name is not specified, the x-fc-base-path value is <code>/2016-08-15/proxy/\${servicename}/\${functionname}/</code> .
(Optional) x-fc-status	If your HTTP function is not used for a migration but as a new web API, it is similar to an event function. The <code>x-fc-status</code> field can be used to report to Function Compute whether the function invocation is successful. For more information, see <a href="#">Overview</a> .

## Limits

- After an HTTP trigger is set for a function, other types of triggers cannot be set for this function.
- Only one HTTP trigger can be created for each function.
- If version management is enabled for a function, only one HTTP trigger can be created for each version or alias of the function. In other words, one HTTP trigger can be created for one version or one alias of a function. For more information about versions and aliases, see [Introduction to versions](#).
- The following fields in a request header and request headers that start with x-fc- cannot be customized:
  - accept-encoding
  - connection
  - keep-alive
  - proxy-authorization
  - te
  - trailer
  - transfer-encoding
- The following fields in a response header and response headers that start with x-fc- cannot be customized:
  - connection
  - content-length

- content-encoding
- date
- keep-alive
- proxy-authenticate
- server
- trailer
- transfer-encoding
- upgrade
- content-disposition:attachment

 **Note** For security, when the default aliyuncs.com domain name of Function Compute is used, the server forcibly adds the content-disposition: attachment field to the response header. This field is used to download the returned result in the browser as an attachment. To remove this limit, you must set a custom domain name.

- Limits on HTTP requests  
If a request exceeds one of the following limits, the system returns status code `400` and error code `InvalidArgument`.
  - Header size: The total size of all keys and values in the headers cannot exceed 4 KB.
  - Path size: The total size of the path, including all of the query parameters, cannot exceed 4 KB.
  - Body size: The total size of the HTTP request body cannot exceed 6 MB.
- Limits on HTTP responses  
If a response exceeds one of the following limits, the system returns status code `502` and error code `BadResponse`.
  - Header size: The total size of all keys and values in the headers cannot exceed 4 KB.
- Other instructions  
You can map different HTTP paths for HTTP functions by binding a custom domain name. You can also use API Gateway, set the backend service type to HTTP, and configure the HTTP function path as the backend service address to implement similar features. For more information, see [Use Function Compute as the backend service of an API operation](#).

## 4.6. Lifecycle hooks for function instances

This topic describes how to implement lifecycle hooks for function instances in a custom runtime.


### Lifecycle hooks

After you configure a lifecycle hook for a function instance, invokes the hook when a related lifecycle event for the instance occurs. The following lifecycle hooks can be configured for a function instance: Initializer, PreFreeze, and PreStop hooks. For more information, see [Function instance lifecycle](#).

Path	Request	Expected response
------	---------	-------------------

Path	Request	Expected response
(Optional) POST <code>/initialize</code>	Request body: none. Request header: common request headers. For more information, see <a href="#">Common request headers in Function Compute</a> .	Response body: the return value of the <code>Initializer</code> hook. StatusCode <ul style="list-style-type: none"> <li>200: The request succeeded.</li> <li>404: The request failed.</li> </ul> Sample code of <code>initialize</code> in Python: <pre>@app.route('/initialize', methods=['POST']) def init_invoke():     rid = request.headers.get(x-fc-request-id)     print("FC Initialize Start RequestId: " + rid)     # do your things     print("FC Initialize End RequestId: " + rid)     return "OK"</pre>
(Optional) GET <code>/pre-freeze</code>	<ul style="list-style-type: none"> <li>Request body: none.</li> <li>Request header: common request headers. For more information, see <a href="#">Common request headers in Function Compute</a>.</li> </ul>	Response body: the return value of the PreFreeze hook. StatusCode <ul style="list-style-type: none"> <li>200: The request succeeded.</li> <li>404: The request failed.</li> </ul>
(Optional) GET <code>/pre-stop</code>	<ul style="list-style-type: none"> <li>Request body: none.</li> <li>Request header: common request headers. For more information, see <a href="#">Common request headers in Function Compute</a>.</li> </ul>	Response body: the return value of the PreStop function. StatusCode <ul style="list-style-type: none"> <li>200: The request succeeded.</li> <li>404: The request failed.</li> </ul>

If you want to use the Initializer hook in a custom runtime, you need to only implement the logic that corresponds to the `/initialize` path and the `POST` method in your HTTP server. You can refer to the sample code for `initialize` in the preceding table.

 **Notice** If you do not configure the Initializer hook for the function, the `/initialize` path does not need to be implemented. Even if the `/initialize` path is implemented by the HTTP server, you cannot invoke or execute the `/initialize` logic in the code.

The PreFreeze and PreStop methods are used in the same manner as the Initializer method.

## 4.7. Specification details

This topic describes the common request headers, HTTP status codes, response headers, and log formats of a custom runtime.

## Common request headers in Function Compute

The following table describes the common request headers that forwards to a custom runtime. If you want to access other Alibaba Cloud services, you may need to use the request headers that specify a temporary AccessKey pair. If you migrate existing applications to Function Compute, skip the following information.

### Note

- Event functions and HTTP functions contain common request headers.
- generates common request headers that contain basic information about permissions and functions.

Header	Description
x-fc-request-id	The ID of the request.
x-fc-access-key-id	The temporary AccessKey ID.
x-fc-access-key-secret	The temporary AccessKey secret.
x-fc-security-token	The temporary security token.
x-fc-function-handler	The handler of the function. If the custom runtime is a function, you do not need to specify a value for this header.
x-fc-function-memory	The maximum memory that the function can use.
x-fc-function-initializer	The handler of the initializer function. If the custom runtime is a function, you do not need to specify a value for this header.
x-fc-initialization-timeout	The timeout period of the initializer function.
x-fc-region	The region where the function resides.
x-fc-account-id	The user ID (UID) of the function owner.
x-fc-qualifier	The service version or alias that you specify when you invoke the function. For more information, see <a href="#">Overview</a> .
x-fc-version-id	The version of the function.
x-fc-service-name	The name of the service to which the function belongs.
x-fc-service-logproject	The Log Service project that is configured for the service to which the function belongs.

Header	Description
x-fc-service-logstore	The Log Service Logstore that is configured for the service to which the function belongs.
x-fc-control-path	<p>The request type of the function. For a custom runtime or custom container, you can check whether you sent a request to invoke an HTTP function or an event function based on the value of this parameter. Valid values:</p> <ul style="list-style-type: none"> <li><i>/invoke</i>: The request is sent to invoke an event function.</li> <li><i>/http-invoke</i>: The request is sent to invoke an HTTP function. adds common headers to your request that contains the request path, request body, and request headers, and forwards the request to the custom runtime or custom container. Then, Function Compute forwards the response headers and body that are returned by the custom runtime or custom container to the client.</li> <li><i>/initialize</i>: The first time you create a runtime environment, automatically initiates a request to invoke an initializer function. Similar to a class constructor, you can invoke the initializer function only once during the lifecycle of a container.</li> </ul>

## HTTP status codes and response headers in Function Compute

A custom runtime can work as an HTTP server. Each function invocation that you make is an HTTP request. This way, each response includes a response code and a response header.

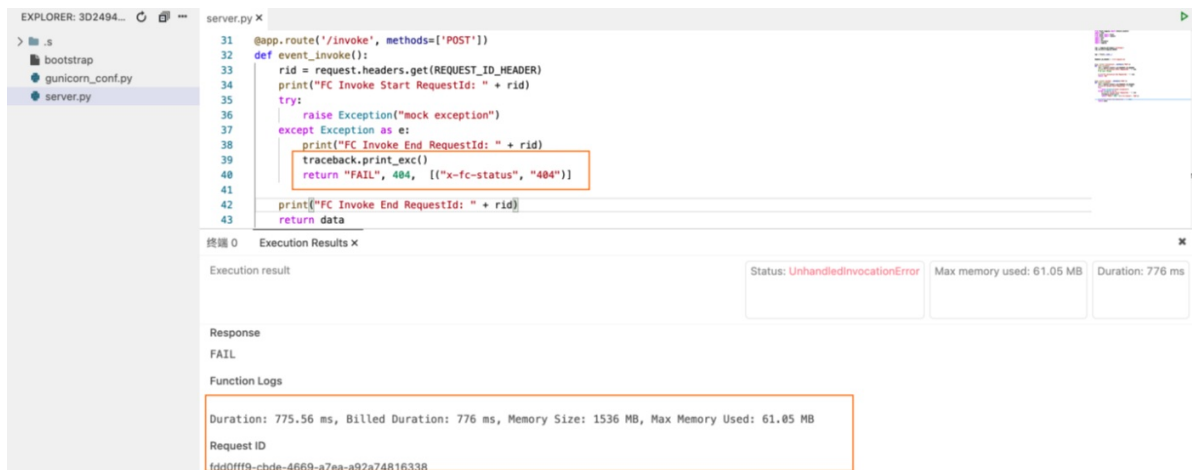
- HTTP status code ( `Status Code` )
  - `200` : succeeded
  - `404` : failed
- Response header ( `x-fc-status` )
  - `200` : succeeded
  - `404` : failed

You can include the `x-fc-status` field in response headers to report whether the local function is invoked to .

- If you do not specify a value for the `x-fc-status` field, considers the invocation successful. If an error occurs during function execution, the system does not report the error to . In this case, the business logic is not affected, but the observability of is affected. The following figure provides an example.



- If you specify a value for the `x-fc-status` field, the system reports function invocation failures to by using the `x-fc-status` field. If an error occurs during function invocation, the system records the error stack information in logs. The following figure provides an example.



**Note** We recommend that you specify the `Statuscode` and `x-fc-status` fields in the HTTP response.

## Function log formats

We recommend that you enable logging when you create a service in Function Compute. This way, all stdout logs that the system generates are stored in the specified Log Service project. For more information, see [Configure the logging feature](#).

If invokes a function in a runtime that is not a custom runtime or a custom container, and the request contains the `x-fc-log-type = "Tail"` header, the response that contains the `x-fc-log-result` header is the log that the system generates when the system invokes the function. The maximum size of a log is 4 KB. You can view the log in the results of the function invocation in the . If you want to view the logs of a custom runtime in the results of the function invocation in the console, you must record the start and end logs of the requests.

Log	Required	Code
Start of the runtime	No <div> <p><b>Note</b> The cold start flag of the function is recorded.</p> </div>	<pre>FunctionCompute \${runtime} runtime initied.</pre> <div> <p><b>Note</b> You can specify a custom value for the <code>\${runtime}</code> parameter. We recommend that you do not specify an official language name in , such as Node.js, Python, or PHP.</p> </div>
Start log of function invocation	Yes	<pre>FC Invoke Start RequestId: \${RequestId}</pre>
End log of function invocation	Yes	<pre>FC Invoke End RequestId: \${RequestId}</pre>
Start log of function initialization	No <div> <p><b>Note</b> If you use an initializer function, record this log.</p> </div>	<pre>FC Initialize Start RequestId: \${RequestId}</pre>
End log of function initialization	No <div> <p><b>Note</b> If you use an initializer function, record this log.</p> </div>	<pre>FC Initialize End RequestId: \${RequestId}</pre>

We recommend that you also include the request ID in your logs for diagnostics in the future and the request ID in your logs is in the `$utcdatetime (yyyy-MM-ddTHH:mm:ss.fff) $requestId [$Level]` `$message` format.

**Note** The API operation that you can perform to specify the log level varies based on the programming language. For more information, see [Log records](#).

## References

- [Overview](#)
- [Basic principles](#)
- [Event functions](#)
- [HTTP functions](#)
- [Lifecycle hooks for function instances](#)

# 5. Custom Container

## 5.1. Overview

This topic describes the background information, basic principles, configuration requirements on an HTTP server, log formats, cold start optimization, billing, and limits of custom containers.

### Background information

In the cloud-native era, container images have become a standard tool for software deployment and development. To improve developer experience and the development and delivery efficiency, allows developers to use custom containers as the runtime environments of functions. The developers can deliver their functions as container images and interact with over HTTP. Custom containers have the following benefits:

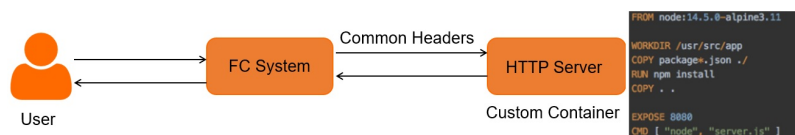
- You can perform cost-effective migration and maintain consistency between development and production environments without modifying code or recompiling binary and shared objects (\*.so).
- Compressed images can be up to 3 GB in size. This allows you to package code and dependencies together for easier distribution and deployment.
- Container images are natively stored in a cache hierarchy. This improves the efficiency of uploading and pulling code.
- Standard, replicable third-party libraries can be used for referencing, sharing, building, code uploading, storage, and version management. This offers a comprehensive open source ecosystem for continuous integration and continuous delivery (CI/CD).

### Basic principles

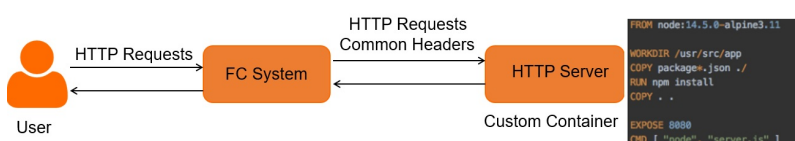
The basic principles of a custom container are the same as those of a **custom runtime**. Before initializes an instance, Function Compute assumes the service role of the function to obtain a temporary username and password and pull an image. After the image is pulled, Function Compute starts your HTTP server by using the specified startup command (Command), parameters (Args), and CAPort (port 9000 by default). Then, the HTTP server takes over all requests from , including the invocations of event and HTTP functions.

Before you develop the logic of a function, you must determine whether the function is an event function or an HTTP function. The following figures show how an event function and an HTTP function work.

- Event function



- HTTP function



### Configuration requirements on an HTTP server

- Services that are started in a custom container must listen on `0.0.0.0:CAPort` or `*:CAPort` . If

you configure the services to listen on `127.0.0.1:CAPort`, the request times out, and the following error is returned:

```
{
  "ErrorCode": "FunctionNotStarted",
  "ErrorMessage": "The CA's http server cannot be started:ContainerStartDuration:2500000000. Ping CA failed due to: dial tcp 21.0.5.7:9000: getsockopt: connection refused Logs : 2019-11-29T09:53:30.859837462Z Listening on port 9000"
}
```

The default listening port (CAPort) of a custom container is port 9000. If a custom container uses the default listening port, the listening port of its HTTP server must be port 9000. If the listening port of the custom container is port 8080, the listening port of its HTTP server must be port 8080.

- You must set the Connection parameter to Keep-Alive and the request timeout period to 15 minutes or longer. The following sample code provides an example on how to specify the values:

```
// In this example, the Express framework for Node.js is used.
var server = app.listen(PORT, HOST);
server.timeout = 0; // never timeout
server.keepAliveTimeout = 0; // keepalive, never timeout
```

- The HTTP server must complete the startup within 120 seconds.

## Common request headers in Function Compute

The common request headers that a custom container can receive are the same as those that a custom runtime can receive. For more information, see [Common request headers in Function Compute](#).

## Function log formats

A custom container uses the same log formats as a custom runtime. For more information, see [Function log formats](#).

## Best practices for cold start optimization

In comparison with code packages, container images rely on the basic environment and consume additional time to download and decompress data. To improve the cold start experience, we recommend the following best practices:

- For optimal latency and stability when a container image is pulled, we recommend that you use a virtual private cloud (VPC) endpoint of the container image in the same region as . Example: `registry-vpc.cn-hangzhou.aliyuncs.com/fc-demo/helloworld:v1beta1`.
- To minimize the size of images, you can build custom images based on minimized images such as Alpine or Ubuntu. You can include only necessary dependencies in an image and exclude unnecessary documents, data, and files.
- You can use container images together with provisioned instances. For more information, see [Configure provisioned instances and auto scaling rules](#).
- When resources are sufficient and threads are secure, you can use an instance to concurrently process multiple requests to avoid unnecessary cold starts and reduce costs.

## Billing


The billable items for a custom container are the same as those for other types of runtimes. For more information, see [Billing](#).

The execution duration of an image resource is the period from the time when an instance starts to pull the image from the image repository to the time when the image is pulled. For example, if an instance configured with 1,024 MB of memory takes 10 seconds to pull an image, the resource usage for this pull is 10 GB-s.

Container images are cached in a specific range for a specific period of time. Therefore, you may not be charged for an image pull during a cold start.

## Limits

- **Image repository**  
You can pull images from the image repositories of **Container Registry** Enterprise Edition and Personal Edition. For more information, see [What is Container Registry?](#).
- **Image access**  
You can read images only from private image repositories in the same region and within the same account.
- **Read and write permissions on files in a container**  
By default, the UID of run-as-user is that of the root user. If you specify a user in *Dockerfile*, the specified user runs the container image.
- **Storage space limit of the writable layer in a container**  
Data that is generated by a container cannot exceed 512 MB, excluding data in the read-only image layer.

 **Note** Data stored in the writable layer of a container does not persist. If the container is deleted, the data is also deleted. To persist data, you can integrate with Apsara File Storage NAS. For more information, see [Configure a NAS file system](#). You can also use other storage services to persist data, such as [Object Storage Service \(OSS\)](#) or [Tablestore](#).

## 5.2. Introduction

This topic describes the definition and parameters of a service.


### Definition

A service is a resource management unit in Function Compute. In business scenarios, a single application can be divided into multiple services. From the resource utilization perspective, a single service can consist of multiple functions. For example, a data processing service is divided into data preparation and data processing. During data preparation, you can select a low-specification instance due to undemanding requirements for function resources. During data processing, however, you need to select a high-specification instance due to demanding requirements for function resources. Before you create a function, you must create a service. All functions of the same service share the same settings such as service authorization and log configuration. You can create and manage services in the Function Compute console or by using the Funcraft tool. For more information, see [Manage services](#).

### Service parameters

When you create a service, you must specify the parameters described in the following table.

Parameter	Required	Description
-----------	----------	-------------

Parameter	Required	Description
ServiceName	Yes	<p>The name of the service. The service name must be unique in the same region and cannot be modified after the service is created. Meanwhile, the service name must comply with the following constraints:</p> <ul style="list-style-type: none"> <li>The name consists of letters, numbers (0-9), underscores (_), and hyphens (-).</li> <li>The name must start with a letter or an underscore (_).</li> <li>The name is case-sensitive.</li> <li>The name must be 1 to 128 characters in length.</li> </ul>
Description	No	The description of the service.
NasConfig	No	Network attached storage (NAS) settings for the service. After you configure these settings, functions of the specified service can access the NAS file system in the same way as accessing a local file system.
Role	No	<p>Grants required permissions for the Function Compute instance to run functions for the service. This parameter is applicable to the following scenarios:</p> <ul style="list-style-type: none"> <li>Authorize the Function Compute instance to use your Log Service resources to store and analyze function execution logs.</li> <li>Authorize the Function Compute instance to access other cloud resources.</li> </ul> <p>For more information about permissions, see <a href="#">User permission</a>.</p>
LogConfig	No	<p>Specifies a log project and a Logstore of Log Service to store and analyze function execution logs. We recommend that you enable Log Service and configure this parameter. Otherwise, you cannot view function execution logs.</p> <div>  <b>Notice</b> When you use Log Service, you will be billed for reserved resources. The minimum fee is RMB 0.04 per day, even when no logs are generated. For more information, see <a href="#">Pay-as-you-go</a>. </div>
VpcConfig	No	Configures Virtual Private Cloud (VPC) for the service. This parameter allows functions to access the specified VPC instance.

Parameter	Required	Description
InternetAccess	No	Specifies whether to allow functions to access the Internet. If this parameter is set to true, functions can access the Internet.

## 5.3. Event functions

This topic describes the background of event functions in a custom container runtime and provides sample code and examples in multiple languages.

### Background information

In a custom container runtime, Function Compute forwards the common headers, request body, POST method, `/invoke`, and `/initialize` to the HTTP server in the container. You can use function signatures, like 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 functions](#).

### Sample code

In the following Node.js Express example, Function Compute calls the POST method and the `/initialize` path to initialize a function instance. The POST method and the `/invoke` path serve as handlers when the function is invoked. Function Compute obtains the context and event parameters from

`req.headers` and `req.body`, and returns the result of function invocation as an HTTP response.

```
'use strict';
const express = require('express');
// Constants
const PORT = 9000;
const HOST = '0.0.0.0';
const app = express();
// initialize example, need config Initializer in function meta
app.post('/initialize', (req, res) => {
  res.send('Hello FunctionCompute, /initialize\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
```

### Examples

## 5.4. HTTP functions

This topic describes the background and limits of HTTP functions in a custom container runtime. This topic also describes how to determine and invoke HTTP functions, and provides sample code and examples in multiple languages.

### Background information

## Background information

Function Compute forwards your requests, including the method, path, query, request headers, request body, and common headers generated by Function Compute to the HTTP server. HTTP functions enable the smooth migration of HTTP web applications. For more information, see [HTTP functions](#).

## 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');
});
app.post('/*', (req, res) => {
  res.send('Hello FunctionCompute, http POST');
});
app.listen(PORT, HOST);
console.log(`Running on http://${HOST}:${PORT}`);
```

## Examples

# 5.5. Create a function

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

## Prerequisites

A Container Registry Enterprise or Personal Edition instance is created. We recommend that you create a Container Registry Enterprise Edition instance.

- A namespace and an image repository for the Container Registry Enterprise Edition instance are created. For more information, see the "Step 4: Create a namespace" and "Step 5: Create an image repository" sections of the [Use a Container Registry Enterprise Edition instance to push and pull images](#) topic.
- A namespace and an image repository for the Container Registry Personal Edition instance are created. For more information, see [Manage namespaces](#) and [Step 2: Create a repository](#).

## Create a function in the Function Compute console

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

In this example, the sample project is in the `/tmp` directory, the region of Function Compute is China (Shenzhen), and the name of the image repository is `nodejs-express`.

- i. Run the following command to go to the `/tmp` directory:

```
cd /tmp
```

- ii. Run the following command in the `/tmp` directory to decompress the sample project:

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

- iii. Run the following command to go to the `custom-container-docs/nodejs-express` directory:

```
cd custom-container-docs/nodejs-express
```

- iv. Run the following command to specify the image repository. Replace `your ACR image name` with the name of your image. For example, replace `your ACR image name` with `registry.cn-shenzhen.aliyuncs.com/fc-demo/nodejs-express:v0.2`.

```
export IMAGE_NAME="your ACR image name"
```

- v. Run the following command to package the image:

```
docker build -t $IMAGE_NAME .
```

- vi. Run the following command to push the image:

```
docker push $IMAGE_NAME
```

## 2. Create a service and configure permissions for it.

- i. In the [Function Compute](#) console, create a service. For more information, see [Create a service](#).
- ii. Attach the **AliyunContainerRegistryReadOnlyAccess** or **AliyunContainerRegistryFullAccess** policy to the service. For more information, see [Grant Function Compute permissions to access other Alibaba Cloud services](#).

The preceding policies allow Function Compute to obtain the temporary account for the default instance in Container Registry. Then, Function Compute uses the temporary account to push the image 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 left-side navigation pane, click **Services and Functions**.
- iii. In the top navigation bar, select the region where the service resides.
- iv. Click **Create Function**. On the Create Function page, move the pointer over the **Event Function** section and click **Configure and Deploy**.
- v. In the **Configure Function** section, set the parameters described in the following table and click **Create**.

← Create Function

Configure Function

\* Function Type

Event Function

Change Type

\* Service Name

test

\* Function Name

Enter a function name

The name can be up to 64 characters in length and can contain digits, lowercase letters, uppercase letters, underscores (\_), and hyphens (-). It must start with a letter.

\* Runtime

Custom Container

If you select custom-container, you must grant the AliyunContainerRegistryReadOnlyAccess permission to the role of the service.

\* Container Image

The image must be stored in Container Registry.

Select Container Image

Command

string array, Example: ["python", "server.py"]

Args

string array, Example: ["-port", "9000"]

Image Acceleration

After image acceleration is enabled, images will be saved to the accelerated image storage in Function Compute. [Learn More](#)

> Show Advanced Settings

Create

Cancel

Parameter	Configuration method
Service Name	Select the service that you created in <a href="#">Step 2</a> .
Function Name	Enter a custom function name.
Runtime	Select <b>Custom Container</b> from the drop-down list.
Instance Type	<p>Select a type of instance used to execute the function. Valid values:</p> <ul style="list-style-type: none"> <li>■ <b>Elastic Instance</b></li> <li>■ <b>Performance Instance</b></li> </ul> <p>For more information about instance types, see <a href="#">Instance types and usage modes</a>.</p>
Container Image	<p>Click <b>Select Container Image</b> to select an image version on the <b>Instance of Personal Edition</b> or <b>Instance of Enterprise Edition</b> tab.</p> <div> <div>?</div> <div> <b>Note</b> If you select a Container Registry Enterprise Edition instance, make sure that the default IP address to which the domain name is resolved is the address in the same VPC as Function Compute. The domain name cannot be resolved by using Alibaba Cloud DNS PrivateZone. </div> </div>
Command	<p>Enter a startup command, such as <code>["/code/myserver"]</code>.</p> <p>This parameter is optional. If you do not set this parameter, the ENTRYPOINT or CMD command included in the image is used.</p>
Args	<p>Enter additional parameters, such as <code>["-arg1", "value1"]</code>.</p> <p>This parameter is optional. If you do not set this parameter, the CMD command included in the image is used.</p>

56

> Document Version: 20220712

Parameter	Configuration method
Memory	Select the memory size used to execute the function. The memory size cannot be less than 512 MB.
Timeout	Enter the timeout period for requests.
Single Instance Concurrency	Enter the maximum number of requests that can be concurrently processed by a single instance. For more information, see <a href="#">A single instance that concurrently processes multiple requests</a> .
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 the need to modify the image.

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

## Use Funcraft to create a function

You can use Funcraft to build and push container images and deploy functions with a few clicks.

1. Run the following command to clone the [custom-container-docs example](#):

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

2. Run the following command to go to the *custom-container-docs/nodejs-express* directory:

```
cd custom-container-docs/nodejs-express
```

3. In the *template.yml* file, replace the value of the Image parameter with the address of your image in Container Registry.
4. Run the following command to build an image:

```
fun build --use-docker
```

5. Run the following command to deploy a function:

```
# 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.

## 5.6. Accelerate image pull for Container Registry Personal Edition

Compared with function code packages, container images provide better portability and a more diverse ecosystem of toolchains. However, the irrelevant data contained makes the cold start of an image that is gigabytes in size last several minutes. When you enable the image pull acceleration feature, the cold start speed can be increased in two phases by approximately 90%, and the period of time that is required to pull an image can be decreased from a few minutes to a few seconds. This topic describes how does image pull acceleration for Container Registry Personal Edition works and how to configure image pull acceleration.

## Principles

If you enable the image pull acceleration feature when you create or update a container image in Container Registry Personal Edition for a function that runs in a custom container, assumes a RAM role, uses a temporary AccessKey pair to pull the image, and then transfers the image to the image cache service of . After the image is cached, the speed for pulling the image is improved.

## Usage notes

- The image pull acceleration feature is supported in the following regions: China (Beijing), China (Zhangjiakou), China (Hangzhou), China (Shanghai), China (Shenzhen), China (Hong Kong), Singapore (Singapore), US (Silicon Valley), and US (Virginia).
- If you enable the image pull acceleration feature, is authorized to pull images from your image repository and transfer the images to the image cache service of . To ensure data security, Container Registry Personal Edition supports network isolation and identity authentication to allow all users to encrypt the data and restrict access to the image. Before you enable this feature, make sure that the operation of transferring images to the image cache service of complies with the security regulations and guidelines of your organization.
- After you create or update a function that uses a container image in Container Registry Personal Edition, a period of time is required to transfer the image. Therefore, a cache miss may occur before the cached image becomes available. The cached image becomes available approximately 5 minutes after you create or update the function.

## Configuration methods

You can configure image pull acceleration by using one of the following methods when you create or update a function:

- Use the console. For more information, see [Create a function in the console](#).

The following figure shows how to configure the image pull acceleration feature when you update a function.

The screenshot shows the Function Compute console configuration page. Under 'Basic Settings', the description is 'hello world by serverless devs', the instance category is 'Elastic Instance', and memory capacity is '256 MB'. Under 'Environment Information', the container image is 'registry', the listening port is '9000', the command is 'The format is ["python", "server.py"]', and the arguments are 'The format is ["-port", "9000"]'. The 'Image acceleration' section has 'Enable' and 'Disable' buttons, with 'Enable' selected. The 'Execution Timeout Period' is set to '60' seconds.

- Use Serverless Devs. For more information, see [YAML syntax](#). Add the `accelerationType` parameter to the `CustomContainerConfig` structure. Valid values:
  - `Default` : enables the image pull acceleration feature.
  - `None` : disables the image pull acceleration feature.

The following sample code shows how to enable the image pull acceleration feature:

```
customContainerConfig:
  image: registry-vpc.<regionId>.aliyuncs.com/fc-demo/python-flask:[Image version]
  accelerationType: Default
```

For more information about the complete procedure, see [Sample project puppeteer-pdf](#).

- [Use SDKs](#).

## View the status of image pull acceleration

You can view the status of image pull acceleration by using one of the following methods to determine whether the cached image is available:

- On the **Configurations** tab of the function details page in the , view the value of the **Image Acceleration Preparation Status** parameter in the **Environment Information** section. Valid values:
  - **Preparing**: Image pull acceleration is being prepared.
  - **Available**: Image pull acceleration is ready.
  - **Failed**: Failed to accelerate the image pull.

## Example:

Environment Information <small>Modify</small>			
Runtime Environments	Custom Container	Request Handler ⓘ	index.handler
Image	registry-vpc.cn-hangzhou.aliyuncs.com	Command ⓘ	n/a
Arguments ⓘ	n/a	Image acceleration ⓘ	Enabled
Execution Timeout Period ⓘ	60 Seconds	Image Acceleration Preparation Status ⓘ	✓ Available
Instance Concurrency ⓘ	1	Listening Port ⓘ	9000

- Call the **GetFunction** operation and check the value of the status parameter in the `accelerationInfo` structure to view the status of image pull acceleration. Valid values:
  - **Preparing** : Image pull acceleration is being prepared. If you invoke the function now, the original image is pulled, and the image pull is not accelerated.
  - **Ready** : Image pull acceleration is ready. If you invoke the function now, the image pull is accelerated.
  - **Failed** : Failed to accelerate the image pull.

## Best practices for versioning

If you use Container Registry Personal Edition and enable the image pull acceleration feature, the transfer of the new image is triggered when you update the image for the function. When you invoke the function before the cached image is available, the original image is pulled and the image pull is not accelerated. You can systematically publish the function by **managing the function** and **managing versions**:

1. Update the function. When you update this function, the LATEST version of the service is also updated.
2. After the status of image pull acceleration changes from **Preparing** to **Available**, publish a new service version.
3. Switch the service alias to the new service version.


## View the results of image pull acceleration

After you enable the image pull acceleration feature, the cold start speed is increased in two phases.

**Sample project puppeteer-pdf** provides an example on how to combine Node.js Express and Puppeteer to convert a web page into a PDF file.

Before the image pull acceleration feature is enabled, the process takes 66.51s. After you enable the image pull acceleration feature, the process takes only 15.2s for the first phase and the cold start speed is increased by 77.1%. The process takes only 4.3s for the second phase and the cold start speed is increased by 71.6%. The cold start speed is increased by 93.5% in total. Sample code:

```
time curl -H "x-fc-invocation-target: 2016-08-15/proxy/CustomContainerDemo/puppeteer-pdf-no-accl" https://$ACCOUNT_ID.$REGION.fc.aliyuncs.com/generate-pdf?url=http://example.com -o /tmp/fc-demo-puppeteer-pdf-no-accl.pdf
# Time spent: 0.06s user 0.09s system 0% cpu 1:06.51 total time
# First image cold start after the image pull acceleration feature is enabled
curl -H "x-fc-invocation-target: 2016-08-15/proxy/CustomContainerDemo/puppeteer-pdf-accl" https://$ACCOUNT_ID.$REGION.fc.aliyuncs.com/generate-pdf?url=http://example.com -o /tmp/fc-demo-puppeteer-pdf-accl.pdf
# Time spent: 0.05s user 0.06s system 0% cpu 15.200 total time
# Cold start after a period of time
curl -H "x-fc-invocation-target: 2016-08-15/proxy/CustomContainerDemo/puppeteer-pdf-accl" https://$ACCOUNT_ID.$REGION.fc.aliyuncs.com/generate-pdf?url=http://example.com -o /tmp/fc-demo-puppeteer-pdf-accl.pdf
# Time spent: 0.05s user 0.06s system 0% cpu 4.300 total time
```

 **Note** Errors may occur during testing. The period of time that is required for an actual scenario may be different.

## 5.7. Accelerate image pull for Container Registry Enterprise Edition

Compared with function code packages, container images provide better portability and a more diverse ecosystem of toolchains. However, the irrelevant data contained makes the cold start of an image that is gigabytes in size last several minutes. When you enable the image pull acceleration feature, the cold start speed can be increased in two phases by approximately 90%, and the period of time that is required to pull an image can be decreased from a few minutes to a few seconds. This topic describes how does image pull acceleration for Container Registry Enterprise Edition works and how to configure image pull acceleration.

### Benefits (compared with the image pull acceleration of Container Registry Personal Edition)

Images in Container Registry Enterprise Edition support all acceleration features of Container Registry Personal Edition and provide the following benefits:

- Network isolation is supported. You can configure independent virtual private cloud (VPC) security rules to manage access to image repositories.
- Exclusive bandwidth helps improve the flexibility of image pull tasks.
- The built-in image conversion feature of image repositories helps prevent occasional unaccelerated cold starts before a cached image is available in .

### Principles

If you enable the image pull acceleration feature for a function that runs a custom container and uses a container image in Container Registry Enterprise Edition, assumes a RAM role and uses a temporary AccessKey pair to pull the image when function requests are processed. The speed for pulling the image is improved.

### Usage notes

- The image pull acceleration feature is supported in the following regions: China (Beijing), China

(Zhangjiakou), China (Hangzhou), China (Shanghai), China (Shenzhen), China (Hong Kong), Singapore (Singapore), US (Silicon Valley), and US (Virginia).

- uses the default VPC IP address of your image repository to resolve the domain name of the container image in Container Registry Enterprise Edition.
- After you create or update a function that uses a container image in Container Registry Enterprise Edition, preferentially pulls the accelerated image from your image repository. If the accelerated image does not exist in your image repository, pulls the original image in your image repository.

## Configuration methods

1. Standard Edition and Advanced Edition instances of Container Registry Enterprise Edition provide the image acceleration feature. You can enable the image pull acceleration feature when you create or update an image repository. For more information, see [Enable image acceleration](#).
2. When you create or update a function, we recommend that you use an accelerated image whose name ends with `_accelerated` and enable the image pull acceleration feature. After the function is configured, you can immediately call the accelerated image. This ensures that the image pull is accelerated when you invoke the function. You can configure image pull acceleration by using one of the following methods when you create or update a function:
  - Use the console. For more information, see [Create a function in the console](#). The following figure shows how to configure image pull acceleration when you create a function.

The screenshot displays the 'Create a function' interface in the Alibaba Cloud console. At the top, three tabs are visible: 'Start from Scratch', 'Use Container Image' (which is selected), and 'Use Template'. Below the tabs, the 'Basic Settings' section is expanded, showing various configuration fields. The 'Function Name' is 'caree-function'. The 'Container Image' is set to 'registry-vpc.cn-hangzhou.cr.aliyuncs.com/\_accelerated'. A note indicates that the VPC of the service should be the default resolution VPC of Container Registry Enterprise Edition. The 'Listening Port' is '9000', the 'Command' is 'python server.py', and the 'Arguments' are '[-port, 9000]'. The 'Function Trigger Mode' is 'Event-triggered'. The 'Instance Category' is 'Elastic Instance'. The 'Memory Capacity' is '512 MB'. There are also links to view instance type details and billing information.

- Use Serverless Devs. For more information, see [YAML syntax](#). To configure image pull acceleration, add the `accelerationType` parameter to the `CustomContainerConfig` structure. Valid values:
  - `Default` : enables the image pull acceleration feature.
  - `None` : disables the image pull acceleration feature.

The following sample code shows how to enable the image pull acceleration feature:

```
customContainerConfig:
  image: registry-vpc.<regionId>.aliyuncs.com/fc-demo/python-flask:[Image version_acc
elerated]
  accelerationType: Default
```

- o [Use SDKs.](#)

## What to do next

You can log on to the or call the [GetFunction](#) operation to view the status of image pull acceleration. For more information, see [View the status of image pull acceleration.](#)

# 6. Programming model extensions

## 6.1. Feature overview

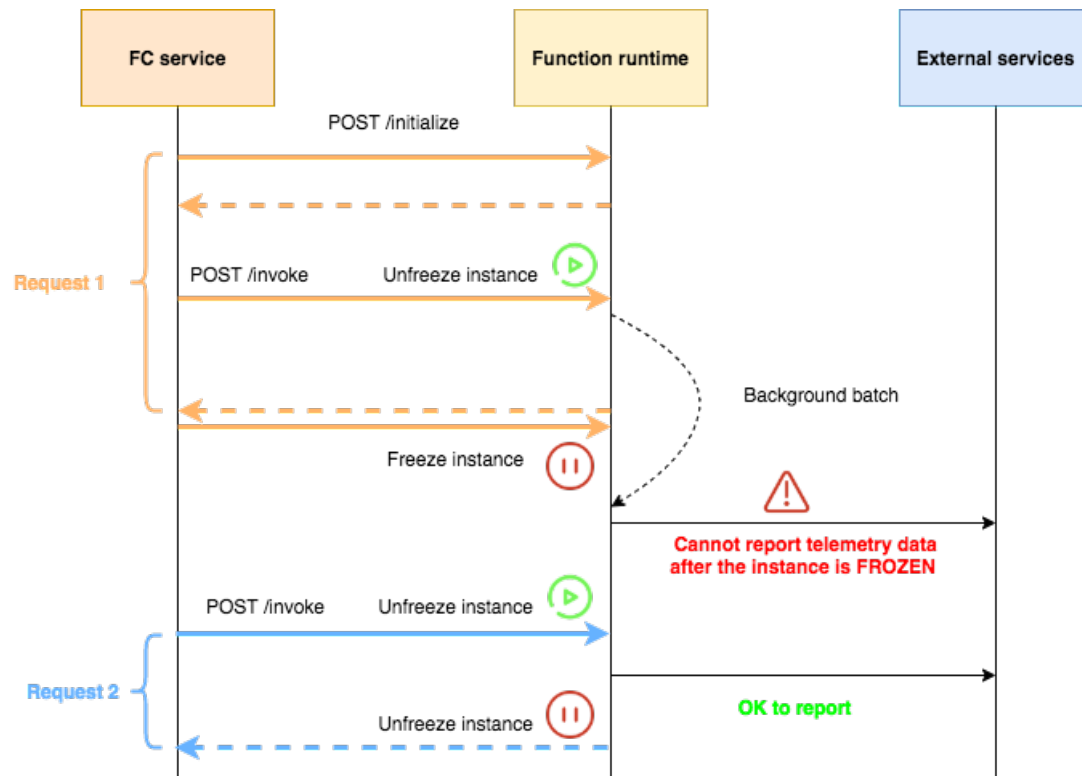
This topic describes the runtime extension feature of Function Compute that is developed based on traditional long-running applications to help you eliminate idle costs.

### Long-running applications and FaaS execution environment

Traditional long-running virtual machines or managed container services often use a billing interval that starts when an instance is started and ends when the instance is stopped. You are charged even if no request is executed during this interval. Function Compute charges you at a billing granularity of 1 ms. Instances are billed only during the execution of actual requests. The instances are frozen in time periods where no requests are executed. This basically eliminates the idle costs of a fully event-driven billing model. However, the freezing mechanism breaks the assumption of long-running processes in traditional architectures and increases the difficulty in migrating applications. For example, the commonly used open source distributed Tracing Analysis libraries or third-party application performance management (APM) solutions cannot correctly report data due to the special execution environment of Function Compute.

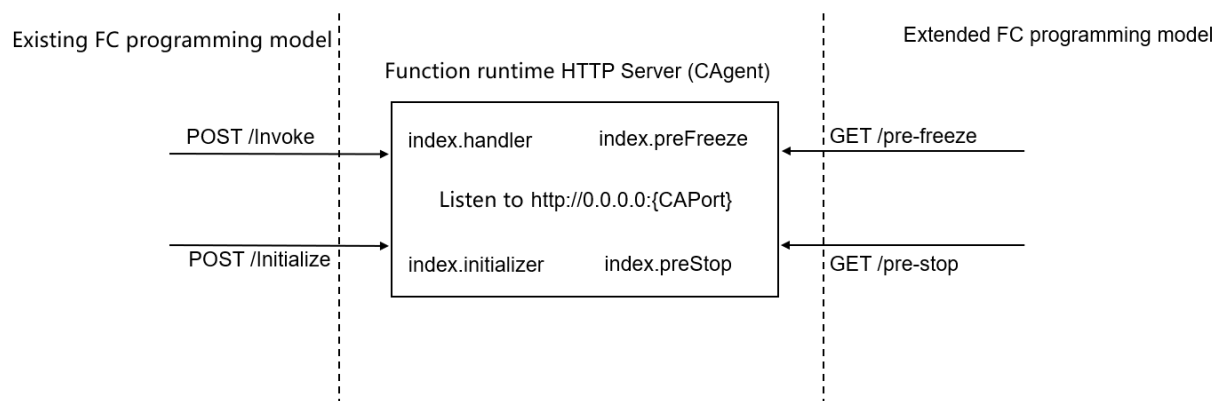
The following pain points hinder the smooth migration of traditional applications to a serverless architecture:

- Data of asynchronous background metrics is delayed or lost. If the data fails to be sent during the execution of a request, the data may be delayed until the next request or data points are discarded.
- The latency is increased if metrics are sent synchronously. If a method similar to Flush is called after each request is completed, this not only increases the latency of each request, but also causes unnecessary pressure on backend servers.
- To support graceful release of functions, applications need to close connections, stop processes, and report status when instances are stopped. Developers do not know when instances are released in Function Compute. In addition, no webhook is provided to send notifications about release events of function instances.

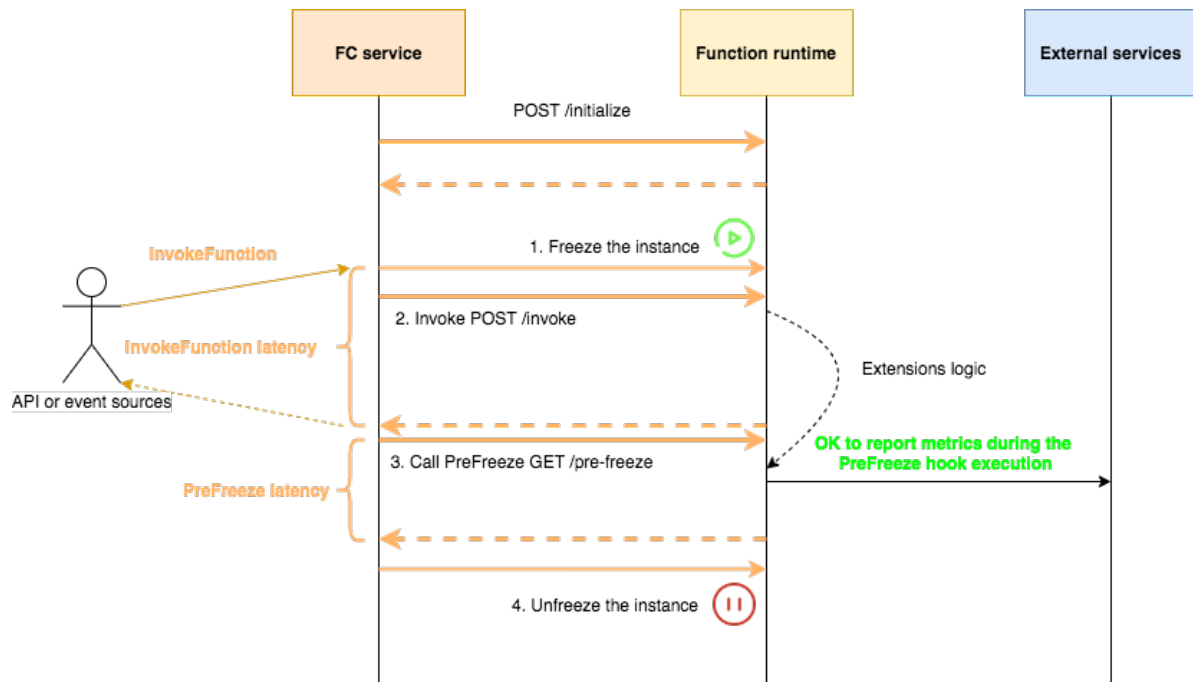


## Programming model extensions

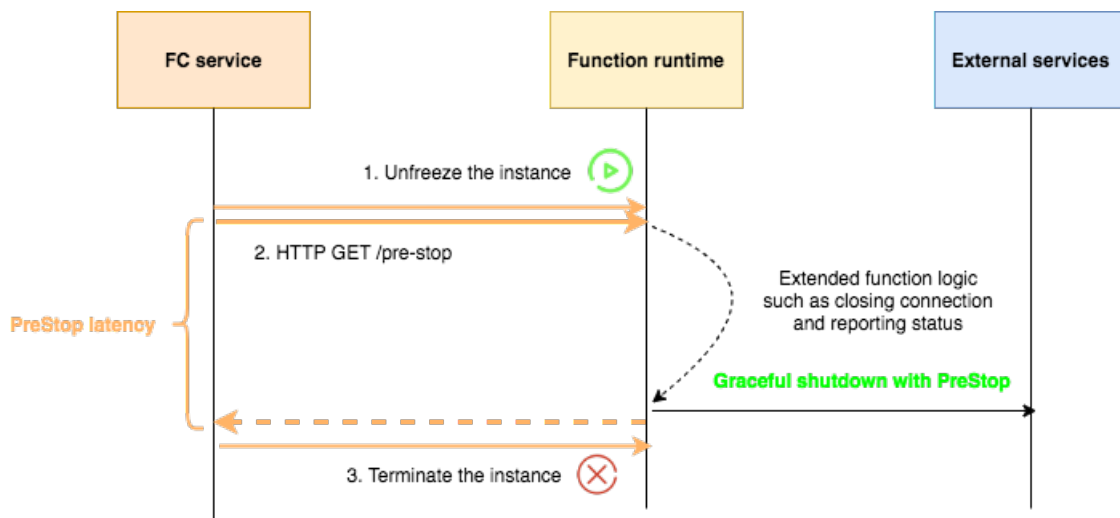
Function Compute provides the runtime extension feature to resolve the preceding pain points. The feature extends the existing programming model for HTTP servers by adding the PreFreeze and PreStop webhooks to the existing HTTP server model. Extension developers implement an HTTP handler to monitor lifecycle events of function instances.



- **PreFreeze:** Each time Function Compute decides to freeze the current function instance, Function Compute sends an HTTP GET request to the `/pre-freeze` path. Extension developers implement the logic to ensure that necessary operations are completed before the instance is frozen. For example, the instance waits until metrics are sent. The function invocation time does not include the execution time of the PreFreeze webhook.

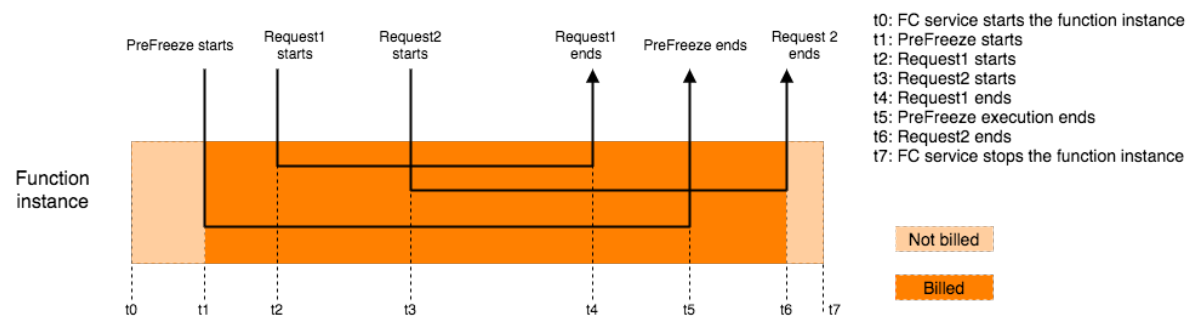


- PreStop: Each time Function Compute decides to stop the current function instance, Function Compute sends an HTTP GET request to the /pre-stop path. Extension developers implement the logic to ensure that necessary operations are completed before the instance is released. For example, database connections are closed and the status is reported or updated.



## Billing

The billing method for PreFreeze or PreStop calls is the same as that for InvokeFunction calls. You are not charged for the number of requests sent to the HTTP hooks. The extensions are also applicable to scenarios in which multiple concurrent requests are executed on a single instance. Assume that multiple invocation requests are concurrently executed on the same instance. After all the requests are completed, the PreFreeze hook is called before the instance is frozen. In the example shown in the following figure, the specification of the function is 1 GB. Assume that the period from  $t_1$  when PreFreeze starts to  $t_6$  when Request 2 is completed is 1s. The execution time of the instance is calculated based on the following formula:  $t_6 - t_1$ . The consumed resource is calculated based on the following formula:  $1s \times 1\text{ GB} = 1\text{ CU}$ .



# 7.FAQ about code development

## 7.1. General FAQ

7.1.1. In which programming languages can I define functions?

7.1.2. How does Function Compute ensure code security?

Function Compute encrypts code and stores it in Object Storage Service (OSS). Function Compute performs integrity checks whenever code is executed. Code execution is isolated from its own file system and network namespace.

7.1.3. What do I do if the "The Access Key ID does not exist" message is returned when I use the information in the context parameter of a function such as the AccessKey ID to access other cloud resources?

The context parameter of a function includes a temporary key that consists of the AccessKey ID, AccessKey secret, and security token. If the security token is not provided, the "The Access Key ID does not exist" message is returned.

The following example demonstrates how to access Object Storage Service (OSS) code in a Python function.

```
import json
import oss2
def my_handler(event, context):
    evt = json.loads(event)
    creds = context.credentials
    # Do not omit the security token for authentication.
    # Do not miss the "security_token" for the authentication!
    auth = oss2.StsAuth(creds.access_key_id, creds.access_key_secret, creds.security_token)
    bucket = oss2.Bucket(auth, evt['endpoint'], evt['bucket'])
    bucket.put_object(evt['objectName'], evt['message'])
    return 'success'
```

## 7.1.4. Considering that Function Compute only supports Node.js, how can I execute C++ code?

Function Compute supports various programming languages. For more information about the programming languages supported by Function Compute, see [Overview](#). The following list describes how to run programs that are written in an unsupported language:

- Redefine your code in a language that Function Compute supports. Node.js and Python are efficient programming languages that have a rich variety of class libraries.
- Use a custom runtime environment. For more information, see [Custom runtime environment](#).
- Compile your C or C++ program as an executable file and use system calls such as fork to run the file.
- Compile your C or C++ module as a shared library and create a Python binding to interface with the library.

The following table describes the advantages and disadvantages of these methods.

Method	Difficulty	Performance Loss	Scenario
Redefine code	Depends on program complexity	Depends on language and scenario	Applications that are not complex
Custom Runtime	Low	Low	All
Call an executable file	Low	High	Latency-insensitive applications, such as asynchronous processing of files in the background
Call a shared library	High	Low	High-performance applications

If these methods are not satisfactory, [contact us](#).

## 7.1.5. How does Function Compute automatically install dependencies in a runtime environment?

You must include all the dependencies in the code package that you upload to . The methods of package management vary with the programming language. For example, in Node.js, you can use npm to install dependencies in the code directory and compress the dependencies and the code into a package. You can install third-party dependencies by using or Serverless Devs. For more information, see [Install third-party dependencies on Function Compute](#).

## 7.2. Custom Runtime FAQ

## 7.2.1. Must the listening port of a custom runtime be the same as that of the HTTP server of the custom runtime?

Yes. The default listening port (CAPort) of a custom runtime is 9000. If a custom runtime uses the default listening port, the listening port of its HTTP server must be 9000. If the listening port of the custom runtime is 8080, the listening port of its HTTP server must be 8080.

The HTTP server started in a custom runtime must listen on the `0.0.0.0:CAPort` or `*:CAPort` port. If you select the `127.0.0.1:CAPort` port, the request times out and the following error occurs:

```
{
  "ErrorCode": "FunctionNotStarted",
  "ErrorMessage": "The CA's http server cannot be started:ContainerStartDuration:250000000000 . Ping CA failed due to: dial tcp 21.0.5.7:9000: getsockopt: connection refused Logs : 2019-11-29T09:53:30.859837462Z Listening on port 9000"
}
```

## 7.2.2. What do I do if the CAExited error occurs when the bootstrap file of a custom runtime is a shell script?

If the *bootstrap* file of a custom runtime is a shell script and the following error occurs, you must add this content to the *bootstrap* file of the custom runtime: `#!/bin/bash` .

```
{
  "ErrorCode": "CAExited",
  "ErrorMessage": "The CA process either cannot be started or exited:ContainerStartDuration:25037266905. CA process cannot be started or exited already: rpc error: code = 106 desc = ContainerStartDuration:250000000000. Ping CA failed due to: dial tcp 21.0.7.2:9000: i/o time out Logs : 2019-11-29T07:27:50.759658265Z panic: standard_init_linux.go:178: exec user process caused \"exec format error\"
}
```

If the bootstrap file is a binary executable file, such as a binary file compiled by using Go or C++, you do not need to add `#!/bin/bash` .

## 7.2.3. What do I do if the CAFilePermission error occurs when I do not have the permissions to execute the bootstrap file of a custom runtime?

To execute the *bootstrap* file of a custom runtime, you must have the 777 or 755 permissions. Otherwise, the following error occurs:

```
{
  "ErrorCode": "CAFilePermission",
  "ErrorMessage": "The CA process cannot be started due to bootstrap file don't have execute permissions"
}
```

You can run the `chmod 777 bootstrap` or `chmod 755 bootstrap` command to obtain the permissions before packaging files.

## 7.2.4. What do I do if the FunctionNotStarted error occurs when I invoke a third-party service in a service started in a custom runtime?

Check whether the third-party service has connection restrictions. If connection restrictions exist or a connection timeout occurs, the logic for starting the HTTP server is not complete and the following error occurs.

```
{
  "ErrorCode": "FunctionNotStarted",
  "ErrorMessage": "The CA's http server cannot be started:ContainerStartDuration:2500000000. Ping CA failed due to: dial tcp 21.0.3.1:9000: getsockopt: connection refused"
}
```

## 7.2.5. What do I do if the HTTP server fails to start within 30s because the HTTP server implementation takes a long time?

We recommend that you optimize the startup speed of the HTTP server.

## 7.2.6. What format is required for the bootstrap file if I use the Windows operating system?

If you use the Windows operating system, the *bootstrap* file must be in the UNIX format.

## 7.2.7. What do I do if the "Process exited unexpectedly before completing request" message is returned?

If the preceding message is returned, you can troubleshoot the error in the following ways:

- If the message is returned for a custom runtime, you can check the settings of the Connection header and server.  
Keep-Alive must be set in the Connection header and the timeout value must be 15 minutes or more at the server side. Example:

```
var server = app.listen(PORT, HOST);
server.timeout = 0; // never timeout
server.keepAliveTimeout = 0; // keepalive, never timeout
```

- The client that invokes the function initiates the cancel operation. For example, the execution period of the function is 10s, but the Timeout value for SDK to invoke the function is set to 5s at the client side. We recommend that you set a timeout value that is greater than the execution period of the function on the client that invokes the function.
- The logic problem of the function causes the execution environment to exit.

## 7.2.8. What do I do if a 404 error occurs when I use a browser or the cURL tool to access a function?

### Problem description

I create an HTTP function for a custom runtime. The service name is *CustomDemo* and the function name is *func-http*. An anonymous HTTP trigger is specified. The following sample routing code can be used to implement the HTTP server of the custom runtime:

```
@app.route('/test', methods = ['POST', 'GET'])
def test():
```


When I use the cURL tool or a browser to access the URL of the HTTP function, a **404** error occurs.

- Use the cURL tool to access the HTTP function.

```
curl -v https://123456789.cn-hangzhou.fc.aliyuncs.com/2016-08-15/proxy/CustomDemo/func-http/test
```

- Use a browser to access the HTTP function.

```
https://123456789.cn-hangzhou.fc.aliyuncs.com/2016-08-15/proxy/CustomDemo/func-http/test
```

 **Note** The URL format of the HTTP function is `https://$ACCOUNT_ID.$REGION.fc.aliyuncs.com/2016-08-15/proxy/$ServiceName/$functionName/$path`.

### Solution

#### Procedure

1. You can use one of the following methods to solve this problem:

- Add a header named `x-fc-invocation-target` to the command. Command syntax:

```
curl -v "x-fc-invocation-target: 2016-08-15/proxy/$ServiceName/$functionName" https://$ACCOUNT_ID.$REGION.fc.aliyuncs.com/$path
```

A sample command:


```
curl -v -H "x-fc-invocation-target: 2016-08-15/proxy/CustomDemo/func-http" https://123456789.cn-hangzhou.fc.aliyuncs.com/test
```

- Bind a custom domain name to the function. Then, run the following command to access the function again. For more information about how to bind a domain name, see .  
If the domain name is `test.abc.com` , the following command can be used:

```
curl -v https://test.abc.com/$path
```

A sample command:

```
curl -v https://test.abc.com/test
```

 **Notice** The path must be `/*` . The service name is *CustomDemo* and the function name is *func-http*.

- Modify your function code and deploy the function. Then, use the default URL to access the function again. An example of modified function code:

```
@app.route('/2016-08-15/proxy/CustomDemo/func-http/test', methods = ['POST','GET'])  
def test():
```

A sample command:

```
curl -v https://123456789.cn-hangzhou.fc.aliyuncs.com/2016-08-15/proxy/CustomDemo/f  
unc-http/test
```

## 7.2.9. What do I do if a 502 error that contains the "Process exited unexpectedly before completing request" error message is returned?

### Causes

The HTTP server connection is closed. Possible causes:

- The keep-alive mode is not configured for the connection.
- The connection is closed after the HTTP server stays idle for a period of time.
- The connection is closed when a read/write operation times out or an error occurs.

### Solutions

#### Procedure

1. uses the keep-alive mode to ensure that the connection to the HTTP servers remains established in the custom runtime. For idempotent requests such as the GET, HEAD, OPTIONS, and TRACE requests, the system retries to establish the connection when an error occurs such as `EOF` and `connection reset by peer` . However, for non-idempotent requests such as the POST and PATCH requests, a 502 error is returned when the connection fails. To prevent 502 errors, configure the following parameters on the server on which the custom runtime runs:
  - Set the connection mode to keep-alive.
  - Disable the idle timeout feature of the HTTP server or set the idle timeout period to more than 15 minutes.

For different HTTP server frameworks, the configurations of the preceding parameters may be different. For example, for the GoFrame framework, you must configure the `SetIdleTimeout`, `ReadTimeout`, and `python uvicorn` parameters. You must set the value of `SetIdleTimeout` to 0, and configure the `--timeout-keep-alive` parameter in the command line of `python uvicorn`. We recommend that you check whether the HTTP server is disconnected when sparse invocations are performed by an HTTP client in the keep-alive mode.

## Causes

The process exited because a function error occurred. Possible causes:

- The exit operation is called.
- The `exception` that occurred during the operation was not captured.

## Solutions

### Procedure

1. You can perform the following operations:
  - Check whether an active exit logic is specified in your code.
  - Add exception capture or overwrite mechanism at the top level of the process in the runtime environment to prevent processes from exiting when an `exception` occurs.

## 7.3. Custom Container FAQ

### 7.3.1. Must the listening port of a custom container runtime be the same as that of the HTTP server started in the custom container runtime?

Yes, the listening ports of the custom container and the HTTP server in the custom container must be the same. The default listening port (CAPort) of a custom container is 9000. If a custom container uses the default listening port, the listening port of the HTTP server started in the custom container must also be 9000. If the listening port of the custom container is 8080, the listening port of the HTTP server started in the custom container must also be 8080.

The HTTP server started in a custom container must listen on the 0.0.0.0:CAPort or \*:CAPort port. If you select the 127.0.0.1:CAPort port, the request times out and the following error occurs:

```
{
  "ErrorCode": "FunctionNotStarted",
  "ErrorMessage": "The CA's http server cannot be started:ContainerStartDuration:25000000000
. Ping CA failed due to: dial tcp 21.0.5.7:9000: getsockopt: connection refused Logs : 2019
-11-29T09:53:30.859837462Z Listening on port 9000"
}
```

## 7.3.2. What do I do if the FunctionNotStarted error occurs when I invoke a third-party service in a service started in a custom container runtime?

Check whether the third-party service has connection restrictions. If connection restrictions exist or a connection timeout occurs, the logic for starting the HTTP server is not complete and the following error occurs.

```
{
  "ErrorCode": "FunctionNotStarted",
  "ErrorMessage": "The CA's http server cannot be started:ContainerStartDuration:2500000000. Ping CA failed due to: dial tcp 21.0.3.1:9000: getsockopt: connection refused"}
```

## 7.3.3. What do I do if the HTTP server fails to start within 30s because the HTTP server implementation takes a long time?

We recommend that you optimize the startup speed of the HTTP server.

## 7.3.4. What do I do if a 502 error that contains the "Process exited unexpectedly before completing request" error message is returned?

### Causes

The HTTP server connection is closed. Possible causes:

- The keep-alive mode is not configured for the connection.
- The connection is closed after the HTTP server stays idle for a period of time.
- The connection is closed when a read/write operation times out or an error occurs.

### Solutions

#### Procedure

1. uses the keep-alive mode to ensure that the connections to the HTTP servers remains established in the custom container. For idempotent requests such as the GET, HEAD, OPTIONS, and TRACE requests, the system retries to establish the connection when an error such as EOF and connection reset by peer occurs. However, for non-idempotent requests such as the POST and PATCH requests, a 502 error is returned when the connection fails. To prevent 502 errors, configure the following parameters on the server on which the custom container runs:
  - Set the connection mode to keep-alive.
  - Disable the idle timeout feature of the HTTP server or set the idle timeout period to more than 15 minutes.

For different HTTP server frameworks, the configurations of the preceding parameters may be different. For example, for the GoFrame framework, you must configure the `SetIdleTimeout`, `ReadTimeout`, and `python uvicorn` parameters. You must set `SetIdleTimeout` to 0, and configure the `--timeout-keep-alive` parameter in the command line of `python uvicorn`. We recommend that you check whether the HTTP server is disconnected when sparse invocations are requested from an HTTP client in the keep-alive mode.

## Causes

The process exited because a function error occurred. Possible causes:

- The exit operation is called.
- The `exception` that occurred during the function execution was not captured.

## Solutions

### Procedure

1. You can perform the following operations:
  - Check whether an active exit logic is specified in your code.
  - Add exception capture or overwrite mechanisms at the top level of the process in the runtime environment to prevent processes from exiting when an `exception` occurs.

## 7.3.5. What do I do if a 404 error occurs when I use a browser or the cURL tool to access a function?

### Problem description

I create an HTTP function for a custom container runtime. The service name is *CustomDemo* and the function name is *func-http*. An anonymous HTTP trigger is specified. The following sample routing code can be used to implement the HTTP server of the custom container runtime:

### Solution

#### Procedure

- 1.

## 7.3.6. What permissions must I grant to Function Compute service role if Alibaba Cloud public container images are not used?

You must grant the `AliyunContainerRegistryReadOnlyAccess` or `AliyunContainerRegistryFullAccess` permissions to the service role. For more information, see [Grant Function Compute permissions to access other Alibaba Cloud services](#).