ALIBABA CLOUD

Alibaba Cloud

Function Compute

Function Management

Document Version: 20210331

Alibaba Cloud

Legal disclaimer

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

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

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

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

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

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

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

Document conventions

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

Table of Contents

1.CI/CD deployment 06
1.1. Use the Function Compute console to deploy an applicatio... 06
1.2. Use Apsara Devops Flow 2020 to deploy a function 08
2.Configure Log Service resources and view function execution lo.. 11
3.Permission management 14
3.1. Permissions 14
3.2. Configure service permissions 19
4.Install third-party dependencies on Function Compute 23
5.Environment variables 25
5.1. Overview 25
5.2. Configure environment variables 25
5.3. Use environment variables 28
6.Allow functions to be invoked only in specified VPCs 29
7.Access resources in a VPC 33
7.1. Configure functions to access VPC resources 33
7.2. Troubleshooting 39
8.Instances in reserved mode 42
8.1. Overview 42
8.2. Manage provisioned instances 42
8.3. Auto-scaling of reserved instances 46
9.Tags 52
9.1. Overview 52
9.2. Manage tags 52
9.3. Use tags to perform group-based service authorization 55
10.Access database 58
10.1. Overview 58

10.2. Make preparations 58
10.3. Access an ApsaraDB RDS for MySQL database 59
10.4. Access an ApsaraDB for MongoDB database 63
10.5. Access an ApsaraDB for Redis database 67
10.6. Access an ApsaraDB RDS for SQL Server database 71
10.7. Access an ApsaraDB RDS for PostgreSQL database 76
10.8. Access an ApsaraDB for Lindorm database 80
11.Instance concurrency management 85
11.1. A single instance that concurrently processes multiple req... 85
11.2. Set the request concurrency in a single instance 90
11.3. Overview of configuring the maximum number of on-dem... 92
11.4. Set the maximum number of on-demand instances 94

1.CI/CD deployment

1.1. Use the Function Compute console to deploy an application

This topic shows you how to use a GitHub or Gitee code repository to implement a continuous integration or continuous delivery (CI/CD) deployment on a template-based application in the Function Compute console.

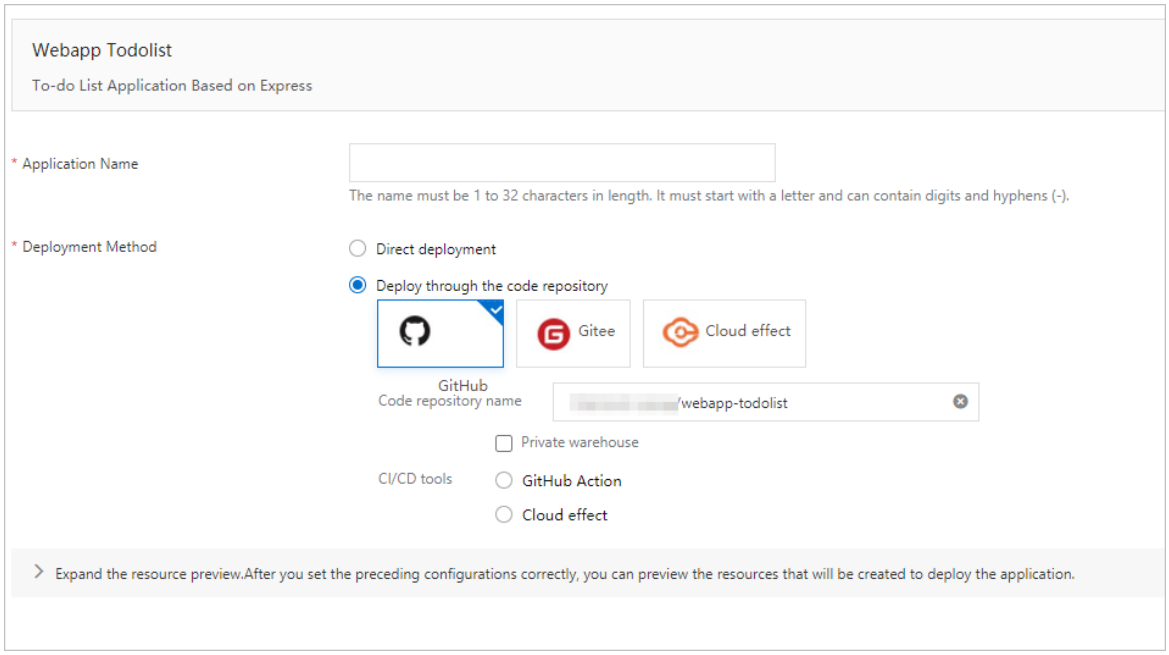
Prerequisites

The following services are activated:

- Function Compute
- GitHub or Gitee. For more information, visit the GitHub official website or the Gitee official website.

Procedure

1. Log on to the Function Compute console.
2. In the top navigation bar, select a region.
3. In the left-side navigation pane, click **Application Center**. In the **Application Templates** section of the **Application Center** page, move the pointer over the template based on which you want to deploy an application and click **Configure and Deploy**.
4. On the **Create Template Application** page, set the parameters as required and click **Deploy**.



The following section describes the parameters:

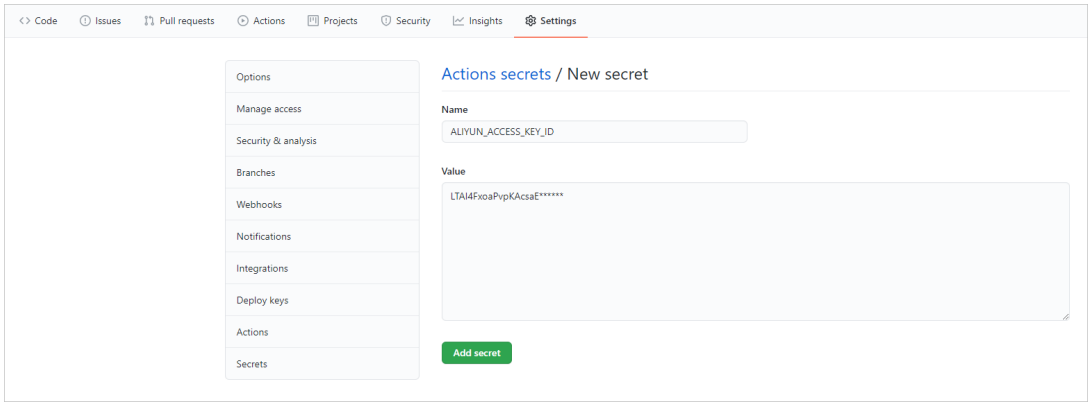
- **Application Name**: the name of the application.
- **Deployment Method**: the method in which the application is to be deployed. Set the parameter to **Deploy from Code Repository** and select a code repository type.

> **Notice**
>
> - When you use GitHub or Gitee to deploy an application in the Function Compute console for the first time, you must authorize Function Compute to manage your code repository that is hosted on GitHub or Gitee. After you select a code repository type, the authorization page appears. Complete the authorization as prompted.
> - To use Apsara Devops to implement the CI/CD deployment, you must manually configure a code repository and a pipeline. For more information, see Use Apsara Devops Flow 2020 to deploy a function.
> - Parameters that you need to set vary between application templates. Set the parameters based on the actual interface.

- **Code Repository Name**: the name of the code repository that is used to deploy the application.
- **Private Repository**: specifies whether the code repository is visible to other users. If you select **Private Repository**, the code repository is visible only to you.
- **CI/CD Tool**: the tool that is used for the CI/CD deployment. To use Apsara Devops to implement the CI/CD deployment, you must manually configure a code repository and a pipeline. For more information, see Use Apsara Devops Flow 2020 to deploy a function.

5. Create secrets or environment variables for your code repository to complete the CI/CD deployment.
   - GitHub
     a. Log on to GitHub.
     b. Go to the details page of the code repository that you use and click the **Settings** tab.
     c. In the left-side navigation pane of the **Settings** tab, click **Secrets**.
     d. In the upper-right corner of the **Actions secrets** page, click **New repository secret**.

e. Set the parameters for creating a secret and click **Add secret**.



Create the following secrets as instructed:

- **ALIYUN_ACCESS_KEY_ID**: the AccessKey ID of the Alibaba Cloud account or RAM user that you use.
- **ALIYUN_ACCESS_KEY_SECRET**: the AccessKey secret of the Alibaba Cloud account or RAM user that you use.
- **ALIYUN_ACCOUNT_ID**: the ID of the Alibaba Cloud account that you use. If you use a RAM user, specify the ID of the Alibaba Cloud account to which the RAM user belongs.

- Gitee
  a. Log on to Gitee.
  b. Go to the details page of the code repository that you use and click the **Settings** tab.
  c. In the left-side navigation pane of the **Settings** tab, click **Environment Variables**.
  d. In the upper-right corner of the **Environment Variables** page, click **New Variable**.
  e. In the **New Variable** dialog box, set the parameters as required and click **Ok**.

  Create the following environment variables as instructed:

  - **ACCESS_KEY_ID**: the AccessKey ID of the Alibaba Cloud account or RAM user that you use.
  - **ACCESS_KEY_SECRET**: the AccessKey secret of the Alibaba Cloud account or RAM user that you use.
  - **ACCOUNT_ID**: the ID of the Alibaba Cloud account that you use. If you use a RAM user, specify the ID of the Alibaba Cloud account to which the RAM user belongs.

What to do next

After the application is deployed, you can view the overview, events, monitoring data, and resources of the application in the Function Compute console. For more information, see Manage web applications.

1.2. Use Apsara Devops Flow 2020 to deploy a function

This topic shows you how to use Serverless Devs in the Apsara Devops Flow console to deploy a function to Function Compute.

Prerequisites

Before you begin, make sure that you have completed the following operations:

- Activate Function Compute.
- Activate Flow.
- Upload your business code to a code repository.

> **Note**
>
> - For more information about the types of code repositories that are supported by Flow, see Configure a source code repository.
> - In addition to your business code, you must also add the **template.yml** file for Funcraft and the **s.yml** file for Serverless Devs to the code repository. You can use the following configuration in the **s.yml** file:
>
> ```
> ProjectName:
>  Component: fun
>  Provider: alibaba
>  Properties:
>   Region: cn-qingdao #The region where you want to deploy a function.
>   Config: s
> ```

Procedure

1. Log on to the Flow console.
2. In the upper-right corner of the **My Pipelines** page, click **Create Pipeline**.
3. In the **Select a pipeline template** dialog box, select **Empty Template** in the Others section and click **Create**.
4. In the **Add Pipeline Source** panel, select a code repository type, specify the code repository that you want to use and the default branch, and then click **Add**.
5. In the **Phase 1** section of the **Process Configuration** tab, click **Empty Task**. In the **Edit** panel, set the parameters as required.
6. Click **Save and Run** in the upper-right corner. In the **Confirm** message, click **OK** to run the pipeline. After you click **OK**, the **Recently Run** tab appears. If the pipeline enters the **Successful** state, the function is deployed to Function Compute.

FAQ

Problem description: I have used a third-party cloud service or configured a trigger for a function. What can I do if I fail to use Flow to deploy the function to Function Compute?

Cause: The **AliyunRDCDefaultRole** RAM role that is assumed by Flow has limited permissions. Therefore, Flow may fail to deploy a function to Function Compute in the following scenarios:

Scenario	Required permission and its feature

Scenario	Required permission and its feature
No specific RAM role is assumed by the function and a third-party cloud service is involved in the function configuration.	• AliyunFCFullAccess: deploys the function. • AliyunRAMFullAccess: Based on the third-party cloud service involved, the Funcraft component of Serverless Devs automatically creates the fc-default-role-${function_name} RAM role and attaches the required permission policy to the RAM role. • ram:PassRole: assigns the specified RAM role to the third-party cloud service. • ram:GetRole: checks whether the fc-default-role-${function_name} RAM role exists and creates the RAM role and attaches the required permission policy to the RAM role if it does not exist.
A trigger is configured for the function. Function Compute supports the following types of triggers: Object Storage Service (OSS) event trigger, Message Service (MNS) topic trigger, Log Service trigger, Tablestore trigger, CDN event trigger, API Gateway trigger, and Cloud Monitor trigger.	The read permissions or full access permissions on the Alibaba Cloud service involved: ensures that resources of the Alibaba Cloud service can be accessed when trigger resources are being created.

You can troubleshoot the issue by using one of the following methods:

- Log on to the RAM console and attach the required permission policy to the AliyunRDCDefaultRole RAM role.

  Notice This method is applicable if you use an Alibaba Cloud account or a RAM user that is authorized to modify the permissions of a RAM role.

- Configure Serverless Devs by using an Alibaba Cloud account or a RAM user that has sufficient permissions. For more information about the procedure, see Key configuration.

Note We recommend that you use the first method to configure permissions instead of configuring Serverless Devs by using an Alibaba Cloud account or a RAM user in the Flow console.

2.Configure Log Service resources and view function execution logs

You can store function execution logs to Log Service, and then perform operations such as debugging, fault analysis, and data analysis based on the logs. This topic shows you how to configure projects and Logstores for Function Compute in the Function Compute console and view function execution logs in the Log Service console.

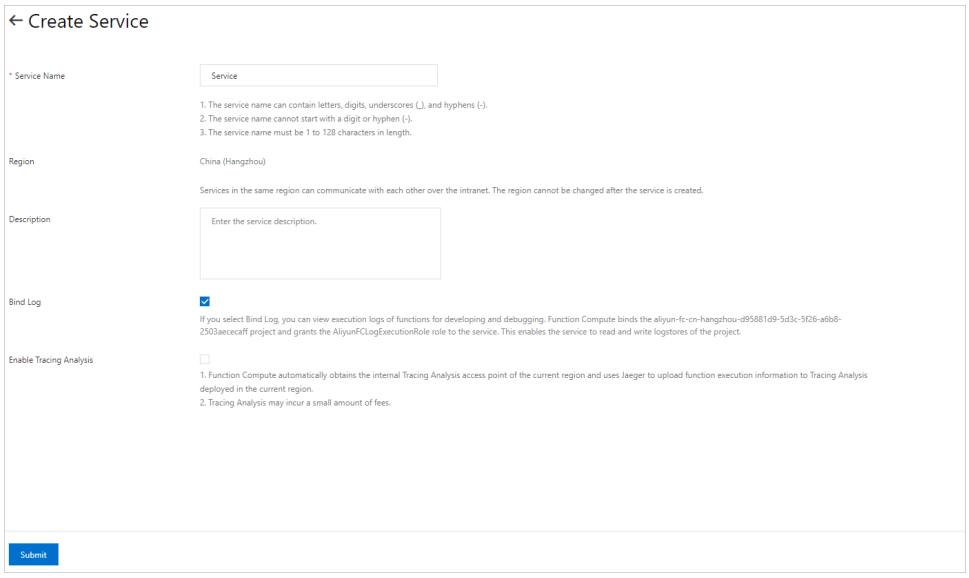
Context

Log Service is an end-to-end logging service developed by Alibaba Cloud. To store function execution logs by using Log Service, you must configure projects and Logstores in corresponding services and authorize the services to access Log Service. Function execution logs are stored in Logstores. All function execution logs of a service are stored in the same Logstore.

Procedure

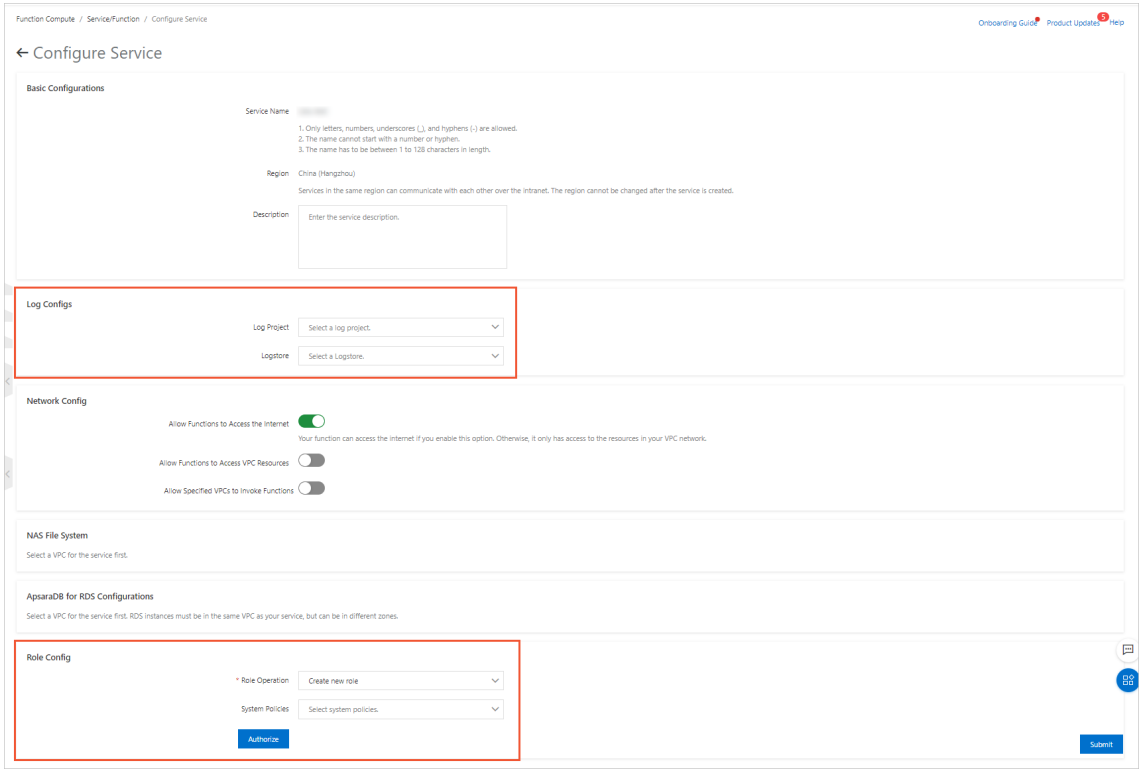
1. Log on to the Function Compute console and configure a project and a Logstore for a service.
   - You can select **Bind Log** when you create a service. For more information, see Create a service. After the service is created, Function Compute creates and binds the corresponding project and Logstore in the background and authorizes you to write function execution logs to Log Service resources.

     > **Note** Log Service resources created by Function Compute in the background are charged in pay-as-you-go mode. For more information, see 计量项和计费项.

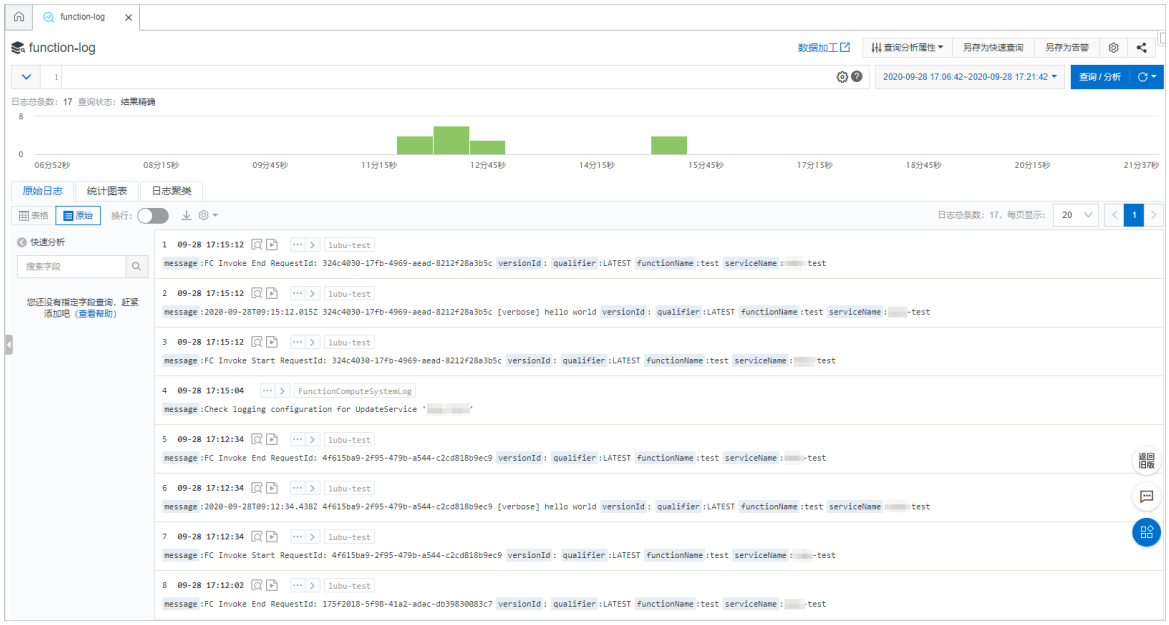
     

   - You can configure a project and a Logstore when you update a service. For more information, see Modify a service. Before you configure them, make sure that the corresponding Log Service resources have been created. For more information about how to create the Log Service resources, see the "Step 1: Create a project and a Logstore" section of the Quick Start of Log Service topic.

You must set the **Log Project** and **Logstore** parameters to the created project and Logstore in the **Log Config** section. In addition, you must set the parameters in the **Role Config** section to authorize Function Compute to write function execution logs to Log Service, as shown in the following figure. For more information, see Permissions.



2. Log on to the Log Service console to view logs. For more information, see the "Step 3: Query and analyze logs" section of the Quick Start of Log Service topic.
   In the service for which the project and Logstore are configured, create a default function whose output is hello world. When the function is executed, the generated logs are stored in the Logstore. You can view the logs in the Log Service console.



References

You can also use Function Compute Command Line Interface (fcli) to configure Log Service resources and view function execution logs. For more information, see Use fcli for the first time.

3.Permission management

3.1. Permissions

This topic describes the scenarios, types, and management mechanisms of permissions in Function Compute.

Scenarios

When you use Function Compute to build an application, you may require various permissions, as shown in the following examples:

- To use Alibaba Cloud Log Service to collect function execution logs, you must authorize Function Compute to write function execution logs to the specified Logstore.
- To use an Alibaba Cloud Object Storage Service (OSS) trigger, you must authorize OSS to invoke functions.
- When functions need to be managed by different personnel, different personnel must be granted different permissions to access resources such as OSS data under an Alibaba Cloud account. You can create Resource Access Management (RAM) roles and grant different personnel the permissions to access Alibaba Cloud resources by using the RAM roles.

Permission types

To access an Alibaba Cloud service, you must have the access permission on this service. You may require the following permissions related to Function Compute:

- To access other Alibaba Cloud services, Function Compute must be granted with corresponding permissions.
  Such a permission is granted to a corresponding service. After a permission is granted to a service, all functions of the service have the permission. The following sample code shows how to use `context.credentials` to access other Alibaba Cloud services:

```
// Uses the context parameter to access OSS.
var OSSClient = require('ali-oss').Wrapper;
exports.handler = function (event, context, callback) {
  console.log(event.toString());
  var ossClient = new OSSClient({
    accessKeyId: context.credentials.accessKeyId,
    accessKeySecret: context.credentials.accessKeySecret,
    stsToken: context.credentials.securityToken,
    region: 'oss-cn-shanghai',
    bucket: 'my-bucket',
  });
  ossClient.put('my-object', new Buffer('hello, fc')).then(function (res) {
    callback(null, 'put object');
  }).catch(function (err) {
    callback(err);
  });
};
```

  For more information about how to configure service permissions, see Configure service permissions.
- To trigger function execution by using an event source, you must authorize the event source to access Function Compute.

Such a permission is granted to the corresponding trigger. You must set permissions for each trigger based on your requirements.
You can configure permissions for a trigger when you create the trigger. For more information, see the following topics:

- Create an OSS trigger
- Create an MNS topic trigger
- Create a Log Service trigger
- Create a Tablestore trigger
- Create a CDN event trigger
- Create a time trigger

- To allow a RAM user to access Function Compute resources, you must grant the corresponding permission to the RAM user.
  This permission is only granted to RAM users. RAM allows you to create and manage multiple identities under an Alibaba Cloud account, and grant diverse permissions to a single identity or a group of identities. In this way, you can authorize different identities to access different Alibaba Cloud resources. For more information, see What is RAM You can grant permissions to RAM users by using your Alibaba Cloud account, which allows RAM users to perform operations on Function Compute resources.
  For more information about how to set permissions for a RAM user, see Grant permissions to a RAM user.
  The following tables list the operation permissions, resource access permissions, and system permissions on Function Compute, which can be granted to RAM users.
  - RAM custom policies for permissions on Function Compute

Resource	Action	Description
acs:fc:<region>:<account-id>:services/<serviceName>	fc:GetService	The resources of a specified service.
	fc:UpdateService	
	fc:DeleteService	
acs:fc:<region>:<account-id>:services/*	fc:CreateService	All service resources.
	fc:ListServices	
acs:fc:<region>:<account-id>:services/<serviceName>.<qualifier>	fc:GetService	The service resources of a specified version.
acs:fc:<region>:<account-id>:services/<serviceName>/functions/<functionName>	fc:GetFunction	The specified function resources in a specified service.
	fc:UpdateFunction	
	fc:DeleteFunction	
	fc:InvokeFunction	
acs:fc:<region>:<account-id>:services/<serviceName>/functions/*	fc:CreateFunction	All function resources in a specified service.
	fc:ListFunctions	

Resource	Action	Description
acs:fc:<region>:<account-id>:services/<serviceName>.*/functions/<functionName>	fc:GetFunction	All function resources of all versions for a specified service.
	fc:UpdateFunction	
	fc:DeleteFunction	
	fc:InvokeFunction	
	fc:PutProvisionConfig	
	fc:GetProvisionConfig	
	fc:PutFunctionOnDemandConfig	
	fc:DeleteFunctionOnDemandConfig	
	fc:GetFunctionOnDemandConfig	
acs:fc:<region>:<account-id>:services/<serviceName>/functions/<functionName>/triggers/<triggerName>	fc:GetTrigger	The specified trigger resource of a specified function in a specified service.
	fc:UpdateTrigger	
	fc:DeleteTrigger	
acs:fc:<region>:<account-id>:services/<serviceName>/functions/<functionName>/triggers/*	fc:CreateTrigger	All trigger resources of the specified function in a specified service.
	fc:ListTriggers	
acs:fc:<region>:<account-id>:services/<serviceName>/versions	fc:PublishServiceVersion	All versions.
	fc:ListServiceVersions	
acs:fc:<region>:<account-id>:services/<serviceName>/versions/<versionID>	fc:DeleteServiceVersion	Specified versions.
acs:fc:<region>:<account-id>:services/<serviceName>/aliases/*	fc:CreateAlias	All aliases.
	fc:ListAliases	
acs:fc:<region>:<account-id>:services/<serviceName>/aliases/<aliasName>	fc:GetAlias	Specified aliases.
	fc:UpdateAlias	
	fc:DeleteAlias	
acs:fc:<region>:<account-id>:custom-domains/*	fc:CreateCustomDomain	All custom domains.
	fc:ListCustomDomains	

Resource	Action	Description
acs:fc:<region>:<account-id>:custom-domains/<domainName>	fc:GetCustomDomain	Specified custom domains.
	fc:UpdateCustomDomain	
	fc:DeleteCustomDomain	
acs:fc:<region>:<account-id>:tag	fc:TagResource	A single tag.
	fc:GetResourceTags	
	fc:UnTagResource	
acs:fc:<region>:<account-id>:tags/*	fc:ListTaggedResources	All tags.

- System policies that can be attached to RAM users in Function Compute
  By default, Function Compute provides three system policies: AliyunFCReadOnlyAccess, AliyunFCInvocationAccess, and AliyunFCFullAccess. You can create custom policies for finer-grained permission management. For more information, see Policy elements.
  - AliyunFCReadOnlyAccess: allows you to read all Function Compute resources.

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "fc:Get*",
        "fc:List*"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

  - AliyunFCInvocationAccess: allows you to invoke all functions.

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "fc:InvokeFunction"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

- AliyunFCFullAccess: allows you to perform operations on all Function Compute resources.

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": "fc:*",
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

- Custom policy: allows you to execute the bar function in the foo service in the China (Hangzhou) region.

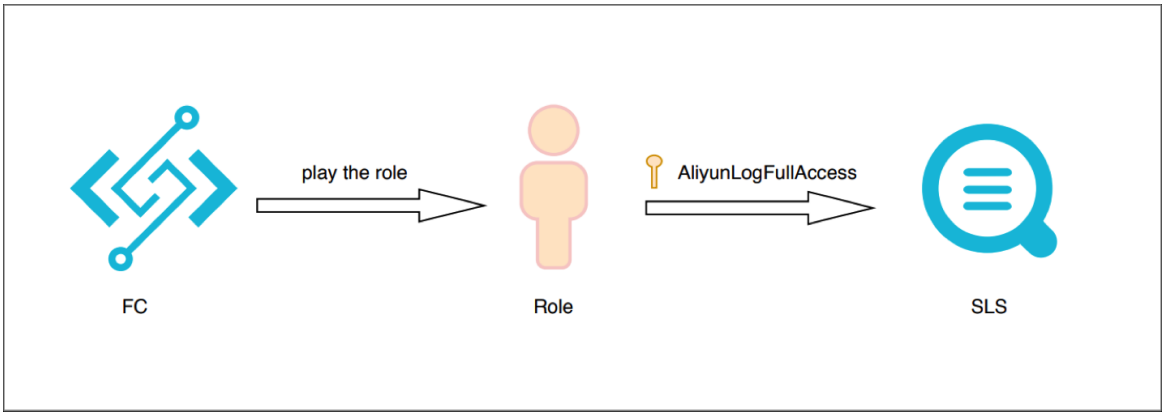
```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "fc:InvokeFunction"
      ],
      "Resource": "acs:fc:cn-hangzhou:*:services/foo/functions/bar",
      "Effect": "Allow"
    }
  ]
}
```

Permission management mechanisms

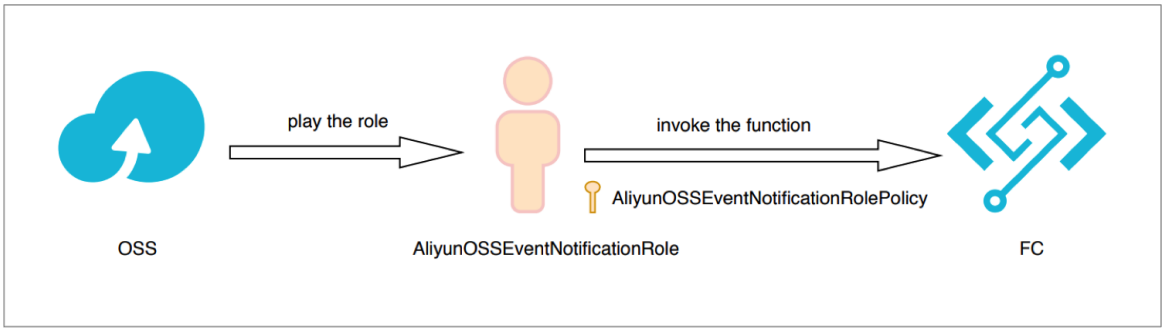
RAM is a resource access control service provided by Alibaba Cloud. Function Compute uses permission management mechanisms based on RAM roles.

- Basic idea of authorization
  A policy indicates the capability to access a service. After the policy is bound to a role, this role can access the service. When a third party needs to access this service, it only needs to assume the role that can access the service. For more information about policies and roles, see Terms for RAM.
- Authorization examples

- Authorize Function Compute to access other Alibaba Cloud services
  For example, use Function Compute to access Log Service. RAM provides the system policy **AliyunLogFullAccess**. This policy allows you to have all permissions on Log Service. When you configure permissions for a service, you can bind the service to a new role or an existing role. Then, attach the **AliyunLogFullAccess** policy to the role, which allows access from Function Compute to Log Service.



- Use event sources to access Function Compute
  For example, use OSS event sources to trigger Function Compute code execution. RAM provides the system policy **AliyunOSSEventNotificationRole**. This policy allows you to trigger execution of Function Compute code by using OSS event sources. When you create a trigger, you can bind the trigger to a new role or an existing role. Then, attach the **AliyunOSSEventNotificationRole** policy to the role, which allows you to trigger execution of Function Compute code by using OSS event sources.



- Use a RAM user to access Function Compute
  For example, grant a RAM user the read permission on all Function Compute resources. Function Compute provides the system policy **AliyunFCReadOnlyAccess**. After you create a RAM user, you can bind it to a new role or an existing role. Then, attach the **AliyunFCReadOnlyAccess** policy to the role, which allows the RAM user to read all Function Compute resources.

3.2. Configure service permissions

This topic describes how to grant Function Compute permissions to access other Alibaba Cloud services in the Function Compute console.

Prerequisites

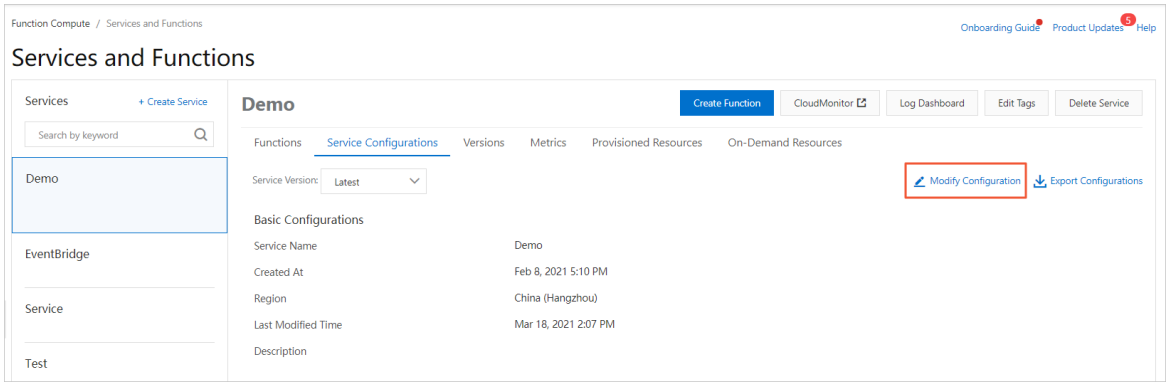
A service is created.

Context

To access other Alibaba Cloud services, Function Compute must be granted relevant permissions. Such permissions are granted to a corresponding service. After the permissions are granted to a service, all functions in the service have the permissions.

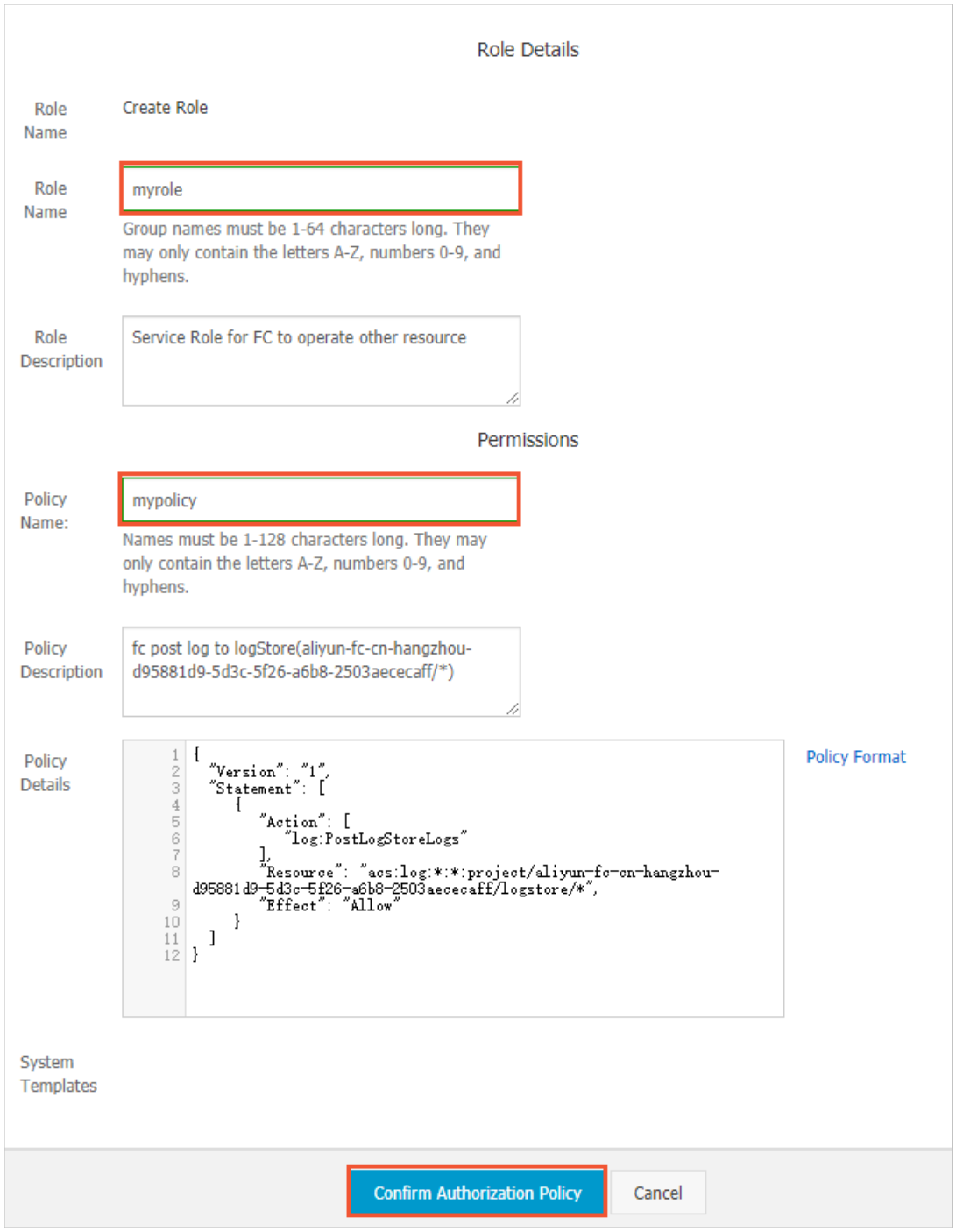
Procedure

1. Log on to the Function Compute console.
2. In the top navigation bar, select a region.
3. In the left-side navigation pane, click **Services and Functions**.
4. On the **Services and Functions** page, click the service that you require. Then, click the **Service Configurations** tab. On the **Service Configurations** tab, click **Modify Configuration**.



5. In the **Role Config** section, set the parameters and click **Submit**.
   - You have not created a role.
     a. Click **Create Role** to go to the **Role Templates** page.

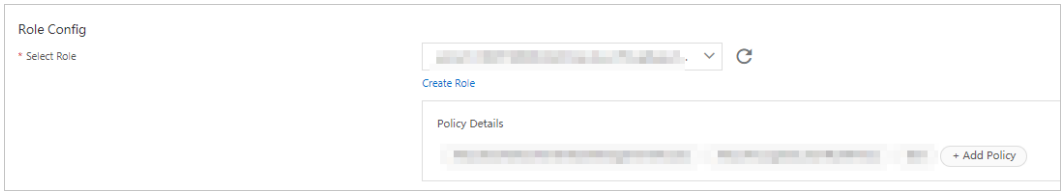
     b. On the **Role Templates** page, set the parameters and click **Confirm Authorization Policy**.



c. Click **Submit**.

- You have created a role.

  a. In the **Role Config** section, select the role to be assigned from the **Select Role** drop-down list.

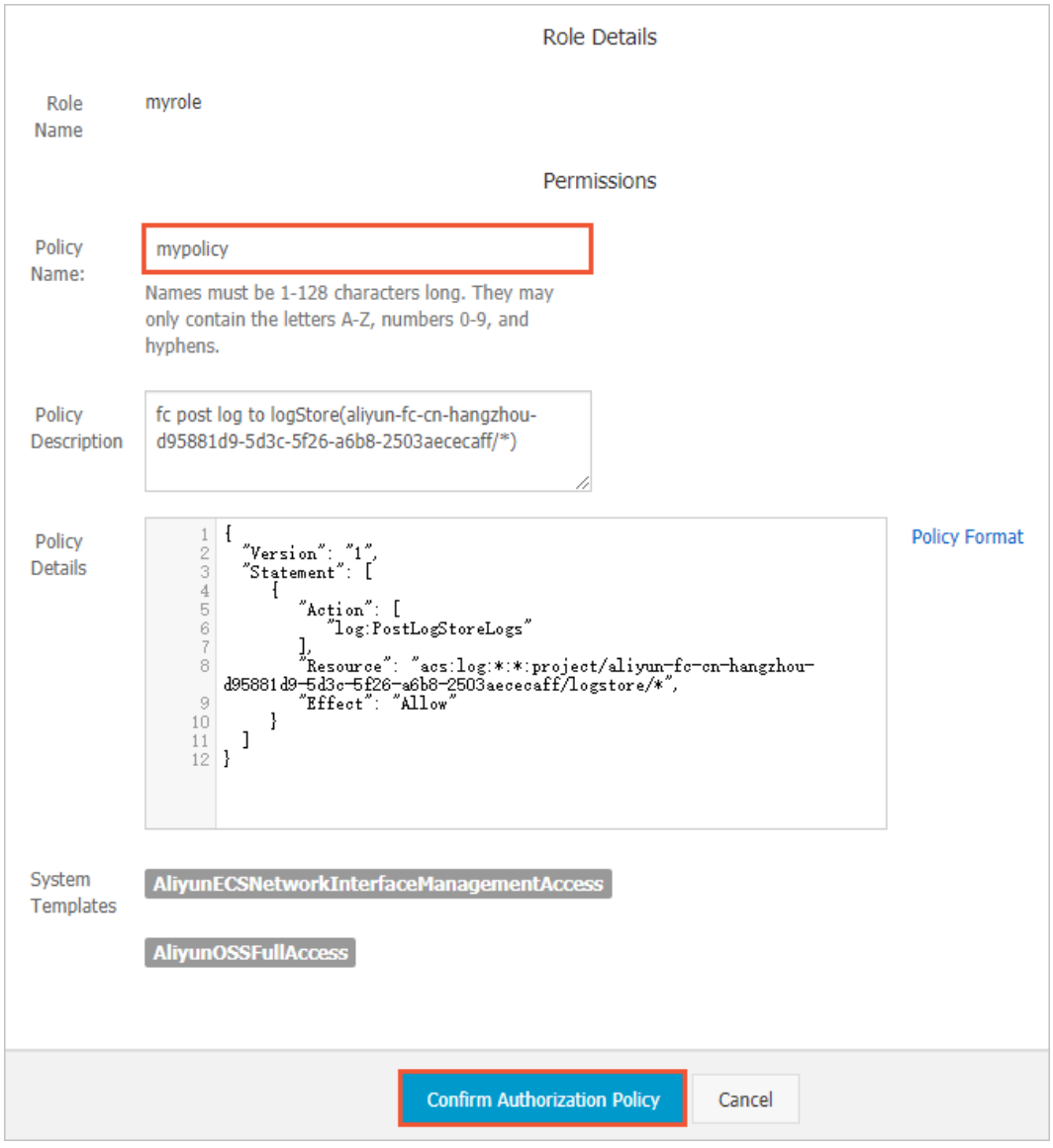


b. In the **Policy Details** section, click **+ Add Policy**.

c. In the **Add Policy** dialog box, select one or more policies that you want to attach to the role from the **Select Policy Template** drop-down list.

d. Click **RAM Authorization**.

e. On the **Role Templates** page, set the **Policy Name** parameter and click **Confirm Authorization Policy**.



f. Click **Submit**.

4.Install third-party dependencies on Function Compute

Function Compute provides built-in common dependencies for you to reference in the runtime environment. Function Compute also supports third-party dependencies. This topic describes how to install third-party dependencies on Function Compute.

Context

For more information about the common dependencies that are built in Function Compute, see the "Use built-in modules" section of the following topics:

- Node.js runtime environments
- Python runtime environment
- PHP runtime environment
- Java runtime environment
- .NET Core runtime environment
- Custom runtime environment

Installation guide

You can install third-party dependencies by using one of the following methods:

- Use the Failed to resolve content from t1882558_v2_1_0.dita#task_2470088/xref_ub0_mhr_t6h:
  i. Compress the third-party dependencies and code files into a package.

  > **Notice**
  > - You must compress all the files in the code directory into a package. After you compress the files, make sure that the files of the handler function are placed in the root directory of the package.
  > - The packaging method varies with the operating system. Select a feasible packaging method as needed.

  ii. Log on to the Failed to resolve content from t1882558_v2_1_0.dita#task_2470088/xref_ub0_mhr_t6h. Then, upload the on-premises code package to deploy functions. Alternatively, you can upload the code package to Object Storage Service (OSS) and import the code package from OSS.
- Use Funcraft: You can use Funcraft, which is a command line tool provided by Function Compute, to create and deploy functions. For more information, see Run the fun install command to install third-party dependencies.
- Use Aliyun Serverless VSCode Extension: You can use Aliyun Serverless VSCode Extension to create and deploy functions. For more information, see Aliyun Serverless VSCode Extension.

References

For more information about how to install third-party dependencies in different runtime environments, see the "Use custom modules" section of the following topics:

- Node.js runtime environments
- Python runtime environment

- PHP runtime environment
- Java runtime environment
- .NET Core runtime environment
- Custom runtime environment

5.Environment variables

5.1. Overview

Environment variables can decouple function code from configurations to improve the flexibility and portability of the code. You can use environment variables in Function Compute to dynamically pass configuration information to function code. This prevents configuration information from being hard-coded in the function code. As a part of function configuration, environment variables are stored as key-value pairs. Different functions can contain different environment variables that are mutually independent.

Security

After you configure environment variables, the environment variables are configured to the operating system environment when a function is executed. You can read the configured environment variables from the system environment variables.

When you create or update environment variables, Function Compute encrypts and stores the environment variables by using Advanced Encryption Standard (AES) 256. When you execute the function, the environment variables are automatically decrypted in reverse. This ensures data security.

Scenarios

- Deploy a function on different platforms or services
  The configurations of the same function code may be different in the test environment and production environment. For example, you can use environment variables to select different Object Storage Service (OSS) buckets, databases, or tables. This way, you can deploy the same function to different platforms or environments without modifying the function code.
- Configure a key
  You can use environment variables to configure security-sensitive authentication information, such as a username and a password for connecting to a database and your Alibaba Cloud AccessKey pair.
- Configure system variables
  You can configure the PATH and HOME variables to system directories.

Limits

- Character set rules
  - A key must start with a letter and can contain letters and digits.
  - A value must contain printable ASCII characters and cannot contain other characters such as Chinese characters.
- Size limits
  The total size of environment variables cannot exceed 4 KB.
- Reserved variables
  To prevent confusion, you cannot use the environment variables that Function Compute reserves. Reserved variables include FC_*, accessKeyID, accessKeySecret, securityToken, and topic.

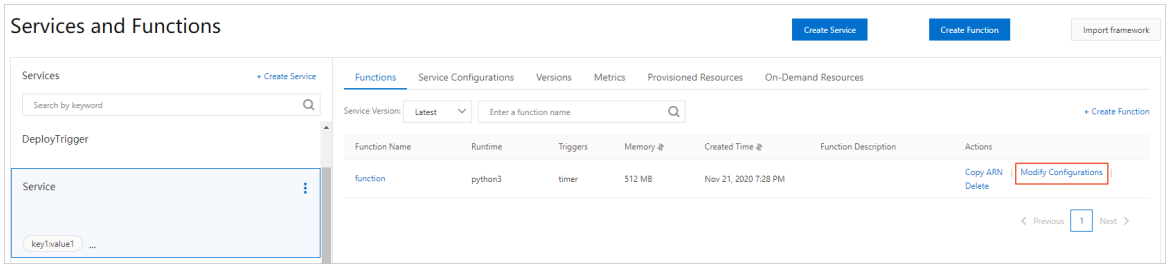
5.2. Configure environment variables

As a part of function configuration, environment variables are stored as key-value pairs. This topic describes how to configure environment variables in the Function Compute console, by using the FunCraft tool, and by using an SDK.

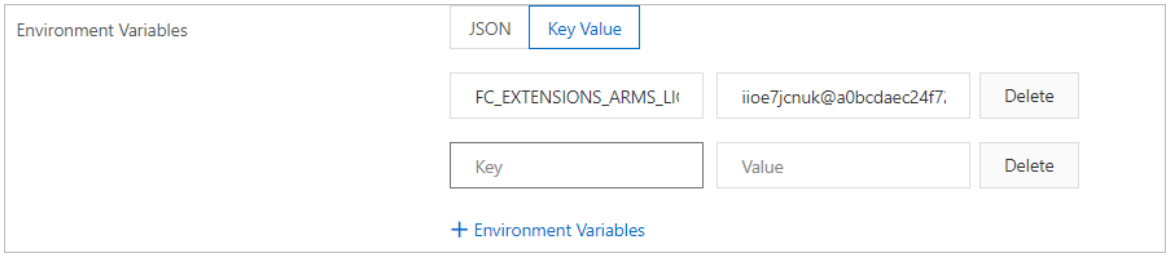
Configure environment variables in the console

Configure environment variables in the console

1. Log on to the Function Compute console.
2. In the top navigation bar, select a region.
3. In the left-side navigation pane, click **Services and Functions**.
4. Click the target service.
5. Find the target function and click **Configure** in the **Modify Configurations** column.



6. Add the key-value pairs of environment variables and click **Submit**.
   Click the **Code** tab.



7. Enter code in the code editor. The sample code is as follows. Click **Invoke** to debug the function.

```
module.exports.handler = function(event, context, callback) {
 var bucket_name = process.env['BUCKET_NAME']
 var table_name = process.env['TABLE_NAME']
 console.log('BUCKET_NAME: ',bucket_name)
 console.log('TABLE_NAME: ',table_name)
 callback(null, "success")
}
```

   Click the **Log** tab. According to the printed log, you can find that the environment variables have been created.

```
FC Invoke Start RequestId: 14d9736e-642d-6134-3d50-976cfbc1cfde
2018-04-13T03:31:27.100Z 14d9736e-642d-6134-3d50-976cfbc1cfde [verbose] BUCKET_NAME: MY_BUCKET
2018-04-13T03:31:27.100Z 14d9736e-642d-6134-3d50-976cfbc1cfde [verbose] TABLE_NAME: MY_TABLE
FC Invoke End RequestId: 14d9736e-642d-6134-3d50-976cfbc1cfde
```

Configure environment variables by using FunCraft

You can configure environment variables by specifying the EnvironmentVariables property in the FunCraft specifications.

1. Run the following sample code to configure a function and environment variables:

```
ROSTemplateFormatVersion: '2015-09-01'
Transform: 'Aliyun::Serverless-2018-04-03'
Resources:
 FunDemo:
  Type: 'Aliyun::Serverless::Service'
  envdemo:
   Type: 'Aliyun::Serverless::Function'
   Properties:
    Handler: index.handler
    CodeUri: . /
    Runtime: python2.7
    EnvironmentVariables:
     OSSEndpoint: oss-cn-hangzhou.aliyuncs.com
     BucketName: fun-local-test
```

The description of the preceding sample code is as follows: Declare a service named FunDemo and then declare a function named envdemo for the FunDemo service. Set the function handler to index.handler and the function runtime to python2.7. In addition, set the CodeUri property to the current directory. During deployment, FunCraft packages the directory specified by the CodeUri property and then uploads the package. You can store dependencies in the directory specified by the CodeUri property.
An environment variable with the key OSSEndpoint and the value oss-cn-hangzhou.aliyuncs.com and an environment variable with the key BucketName and the value fun-local-test are further configured for the function.

2. Run the **fun deploy** command to deploy the function.
   After the deployment is complete, log on to function Compute console to view the created FunDemo service, the envdemo function, and environment variables with the keys OSSEndpoint and BucketName.

Configure environment variables by using an SDK

The following description uses the Python SDK as an example. The environmentVariables parameter specifies environment variables. The values of this parameter are stored in alphabetical order. The sample code for creating, updating, and obtaining environment variables is as follows:

- Create an environment variable

```
# coding: utf-8
import fc2
client = fc2.Client(
  endpoint='your endpoint',
  accessKeyID='your accessKeyID',
  accessKeySecret='your accessKeySecret')
client.create_service('test')
client.create_function(
  'test', 'test_env', 'python2.7', 'main.handler',
  codeDir='/path/to/code/', environmentVariables={'testKey': 'testValue'})
res = client.get_function('test', 'test_env')
print res.data
```

- Update an environment variable

```
client.update_function(
  'test', 'test_env', 'python2.7', 'main.handler',
  codeDir='/path/to/code/', environmentVariables={'newKey': 'newValue'})
res = client.get_function('test', 'test_env')
print res.data
```

- Obtain environment variables

```
resp = client.get_function('test', 'test_env')
env = func['environmentVariables']
```

5.3. Use environment variables

You can use code to read system environment variables and configure the environment variables to the runtime environment of a function to run the function. This topic describes the sample code for using environment variables in different runtime environments.

Sample code

Assume that the environment variable `{"key":"val"}` is configured. The following code provides an example on how to read the environment variable and print the value of the environment variable:

Node.js | Python | Java | PHP | C#

```
var value = process.env.key
console.log(value)
```

6.Allow functions to be invoked only in specified VPCs

By default, you can use public and internal endpoints to invoke a function after you create the function. For security, you can allow functions to be invoked only over a specified virtual private cloud (VPC), but not the public and internal networks. In this case, you must bind the specified VPC to the service where the functions reside. This topic shows you how to bind a specified VPC to allow functions to be invoked only in the VPC.

Prerequisites

Before you begin, make sure that the following operations are complete:

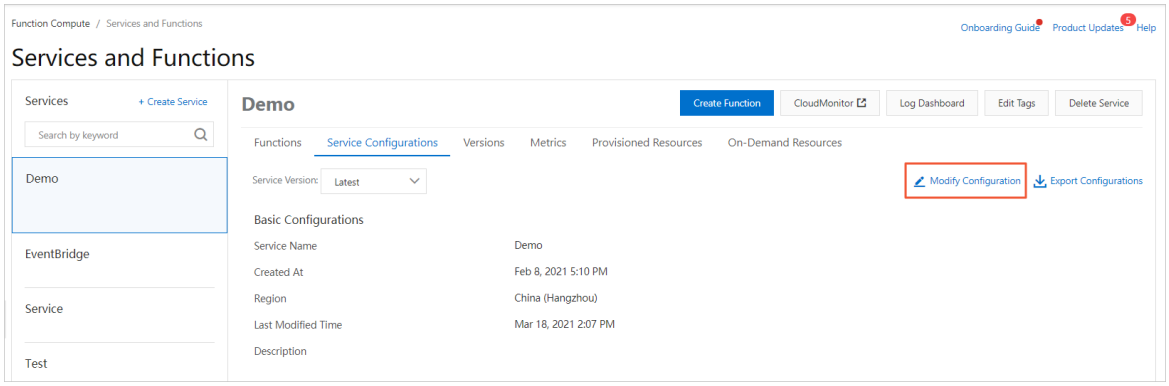
- Create a service.
- Create a function.

Usage notes

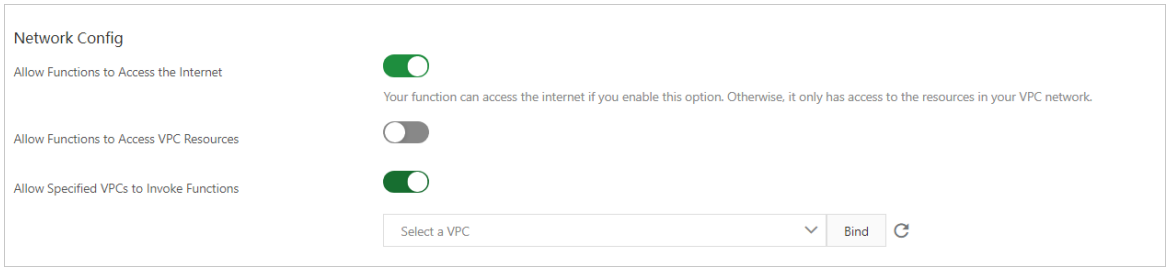
- You can bind a maximum of 20 VPCs to a service.
- If you allow functions to be invoked only in a specified VPC, functions invoked by triggers are not affected.
- After a VPC is bound to a service, the VPC is bound to all versions and aliases of the service.
- After you allow functions to be invoked only in a specified VPC, invocation requests from the Internet and other VPCs are denied. In this case, the HTTP status code is 403, the error code is `AccessDenied`, and the error message is `Resource access is bound by VPC: VPCID`.

Bind a VPC

1.
2.
3. On the **Services and Functions** page, click the service that you require. Then, click the **Service Configurations** tab. On the **Service Configurations** tab, click **Modify Configuration**.

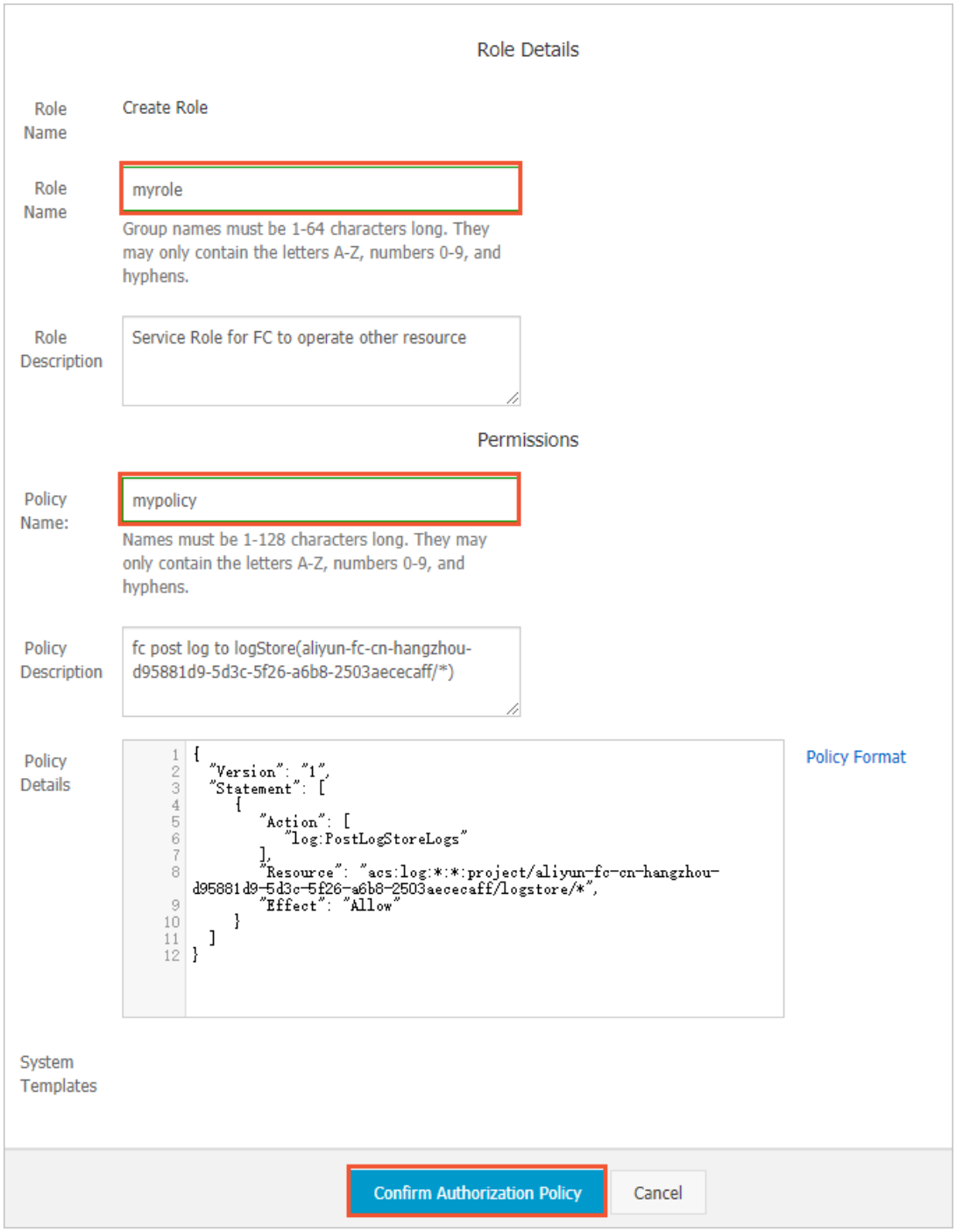


4. In the **Network Config** section on the **Configure Service** page, turn on **Allow Specified VPCs to Invoke Functions** and select the VPC to be bound.

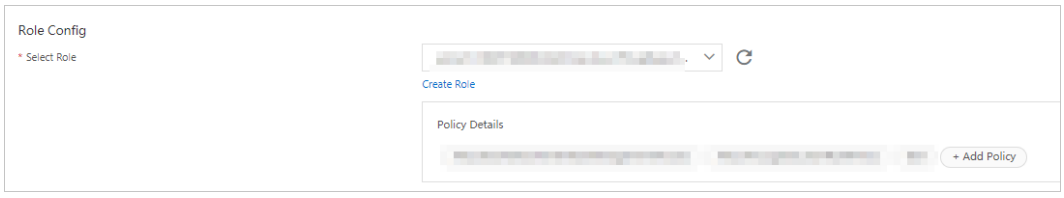


5. (Optional)Click **Bind**. You can bind multiple VPCs to the service.
6. In the **Role Config** section, set the parameters.
   - You have not created a role.
     a. Click **Create Role** to go to the **Role Templates** page.

     b. On the **Role Templates** page, set the parameters and click **Confirm Authorization Policy**.



- You have created a role.
  a. In the **Role Config** section, select the role to be assigned from the **Select Role** drop-down list.

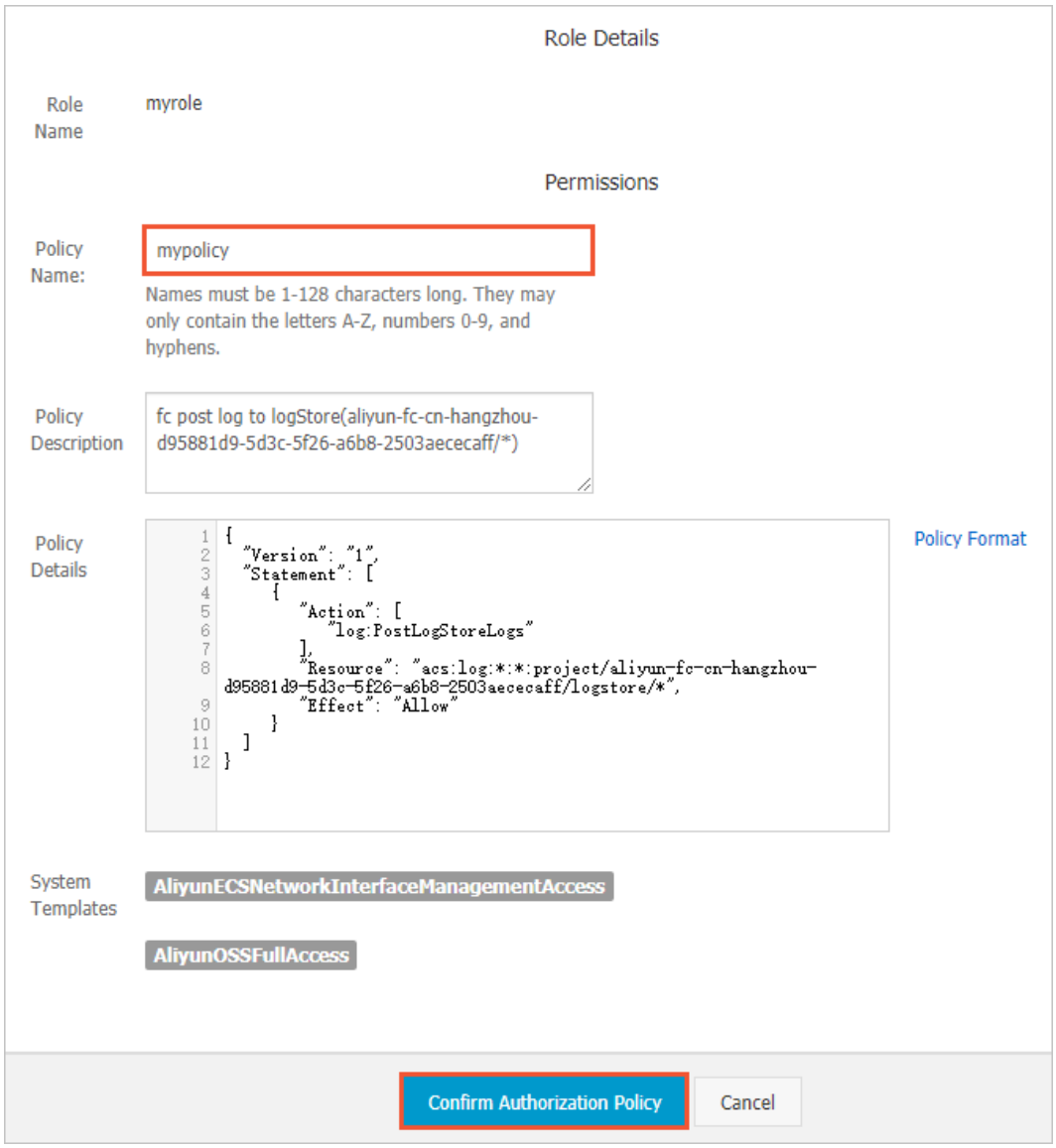


  b. In the **Policy Details** section, click **+ Add Policy**.

c. In the **Add Policy** dialog box, select one or more policies that you want to attach to the role from the **Select Policy Template** drop-down list.

d. Click **RAM Authorization**.

e. On the **Role Templates** page, set the **Policy Name** parameter and click **Confirm Authorization Policy**.



7. Click **Submit**.
   After the operations are complete, all functions in the service can be invoked only in the specified VPC.

7.Access resources in a VPC

7.1. Configure functions to access VPC resources

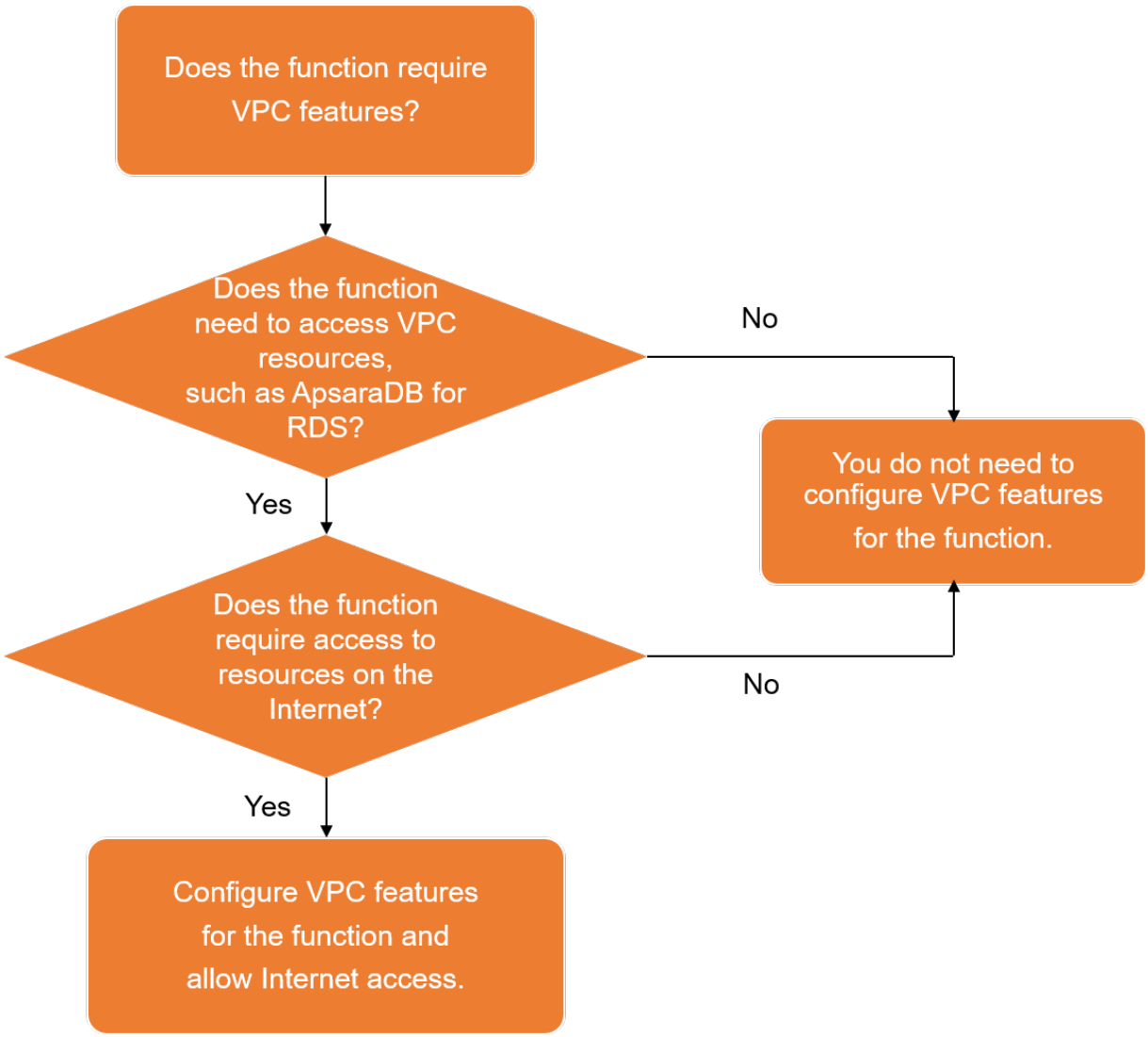
By default, Function Compute cannot access resources that you have created in a virtual private cloud (VPC). You must manually set the VPC configuration for Function Compute to authorize Function Compute to access resources deployed in the VPC.

Prerequisites

- A service is created.
- A function is created.
- A VPC and a vSwitch are created.
- A security group is created.

Context

Additional fees are charged for VPC. We recommend that you use Resource Access Management (RAM) rather than the VPC configuration to grant access permissions on a service such as Tablestore. Therefore, before you configure a VPC, you must determine whether the VPC configuration is required.



The VPC configuration is set on the service level. When you grant access permissions to a service in Function Compute, all functions in the service are authorized to access the specified VPC.

> **Note** If your resources are not deployed in a zone where Function Compute is available, create a vSwitch in your VPC. The vSwitch must be in the same zone as Function Compute. In addition, you must specify the vSwitch ID in the configuration of the specified service in Function Compute. vSwitches in the same VPC can communicate with each other. Therefore, Function Compute can use the vSwitch to access resources that are deployed in the VPC and reside in other zones.

The vpcId, vSwitchIds, and securityGroupId fields are defined in the VPC configuration. All the fields must be specified.

- vpcId: the ID of the VPC to be accessed.
- vSwitchIds: the vSwitches. You must specify at least one vSwitch ID.
  The vSwitchIds field specifies the subnets that Function Compute can access. We recommend that you specify two or more vSwitches in the vSwitchIds field. This allows your functions to be executed in other subnets when an error occurs in the zone or IP addresses are insufficient. If multiple vSwitch IDs are specified in the vSwitchIds field, Function Compute selects one when it creates an elastic network interface (ENI).

- securityGroupId: the ID of the security group that is associated with the ENI.

```
"vpcConfig": {
    "vpcId": "string",
    "vSwitchIds": [ "string" ],
    "securityGroupId": "string"
 }
```

  The securityGroupId field specifies the security group with which the ENI and Function Compute are associated. A security group defines the inbound and outbound rules for Function Compute in the specified VPC. In this security group, configure a rule to allow access from the security group with which Function Compute is associated. Otherwise, Function Compute cannot access resources that are deployed in the specified VPC.

Function Compute services contain a Boolean field internetAccess that indicates whether a service is allowed to access the Internet. The default value is true, which indicates that the service can access the Internet. You can set the internetAccess field to false, which disallows all functions in the service to access the Internet.

Determine whether the VPC configuration is required

Fields defined in the VPC configuration

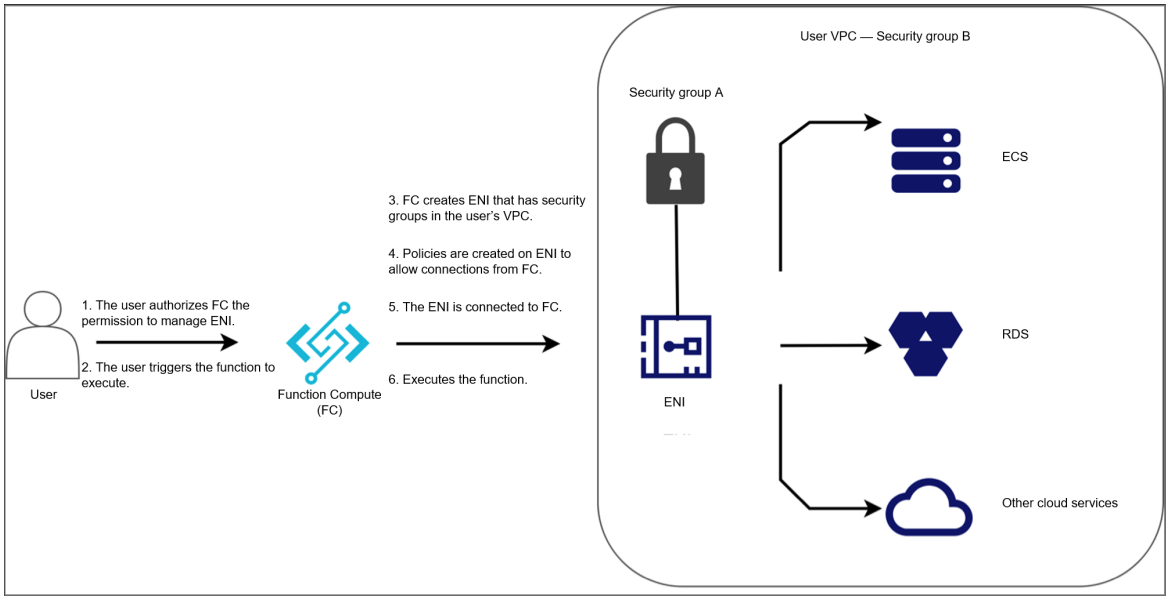
Access to the Internet

How it works

A VPC is a custom private network created on Alibaba Cloud. VPCs are logically isolated from each other. You can create and manage your Alibaba Cloud service instances in your VPC, such as Elastic Compute Service (ECS) instances, Server Load Balancer (SLB) instances, and ApsaraDB RDS instances. This prevents these resources from being accessed over the Internet.

The following part describes how Function Compute accesses resources in a VPC:

A VPC is a private network dedicated for your use. You must authorize your ENI to access the VPC and attach this ENI to the instance used to execute your functions. This allows the functions to access the resources in your VPC. For more information about ENIs, see ENI overview.



When you create an ENI, you must provide configuration information such as the VPC ID, security group ID, and vSwitch ID. Function Compute configures the ENI based on this information. This allows your functions to access resources in the specified VPC by using the ENI.

For more information about how Function Compute accesses resources in a VPC, see Overview.

Usage notes

- If you cannot use Alibaba Cloud VPC in the China (Hangzhou), China (Shanghai), China (Beijing), and China (Shenzhen) regions, you must activate it as prompted in the console.
- The following table describes the regions where Function Compute is available. If the region where your resources reside is not in the following table, see How can I resolve the "VSwitch is in unsupported zone" error?.

Region	Region ID	VPC
China (Hangzhou)	cn-hangzhou	cn-hangzhou-f,cn-hangzhou-g,cn-hangzhou-h
China (Shanghai)	cn-shanghai	cn-shanghai-b,cn-shanghai-e,cn-shanghai-g,cn-shanghai-f
China (Qingdao)	cn-qingdao	cn-qingdao-c
China (Beijing)	cn-beijing	cn-beijing-h,cn-beijing-c,cn-beijing-e,cn-beijing-f
China (Zhangjiakou)	cn-zhangjiakou	cn-zhangjiakou-b,cn-zhangjiakou-a
China (Hohhot)	cn-huhehaote	cn-huhehaote-a,cn-huhehaote-b
China (Shenzhen)	cn-shenzhen	cn-shenzhen-e,cn-shenzhen-d
China (Chengdu)	cn-chengdu	cn-chengdu-a, cn-chengdu-b
China (Hong Kong)	cn-hongkong	cn-hongkong-c
Singapore (Singapore)	ap-southeast-1	ap-southeast-1a,ap-southeast-1b
Australia (Sydney)	ap-southeast-2	ap-southeast-2a,ap-southeast-2b
Malaysia (Kuala Lumpur)	ap-southeast-3	ap-southeast-3a
Indonesia (Jakarta)	ap-southeast-5	ap-southeast-5a,ap-southeast-5b
Japan (Tokyo)	ap-northeast-1	ap-northeast-1b,ap-northeast-1a
UK (London)	eu-west-1	eu-west-1a

Region	Region ID	VPC
Germany (Frankfurt)	eu-central-1	eu-central-a,eu-central-1a,eu-central-1b
US (Silicon Valley)	us-west-1	us-west-1a,us-west-1b
US (Virginia)	us-east-1	us-east-1b, us-east-1a
India (Mumbai)	ap-south-1	ap-south-1a,ap-south-1b

Network access modes

Functions can access resources in four network access modes based on network settings. You can set networks for your functions as needed.

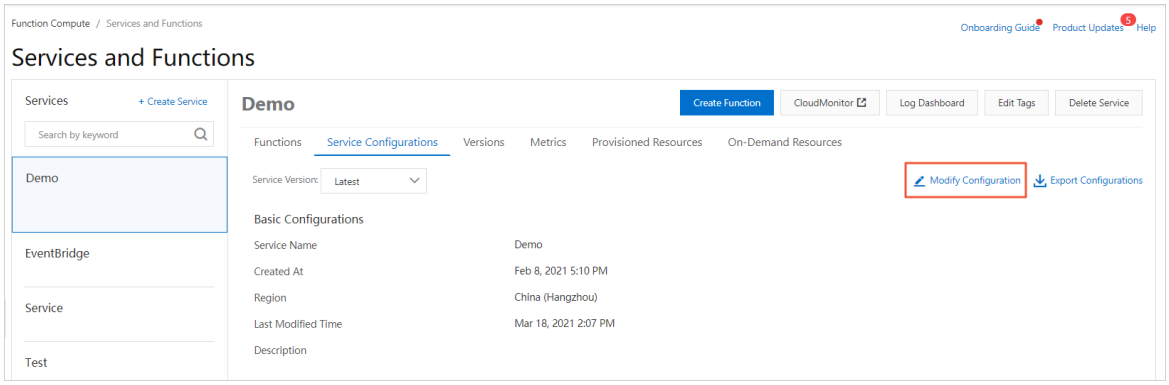
Allow functions to access the Internet	Allow functions to access VPC resources	Network access mode
Yes	Yes	Functions can access the Internet and a specified VPC.
Yes	Not	Functions can access only the Internet.
Not	Yes	Functions can access only a specified VPC.
Not	Not	Functions cannot access the Internet or a specified VPC.

Configure networks and permissions

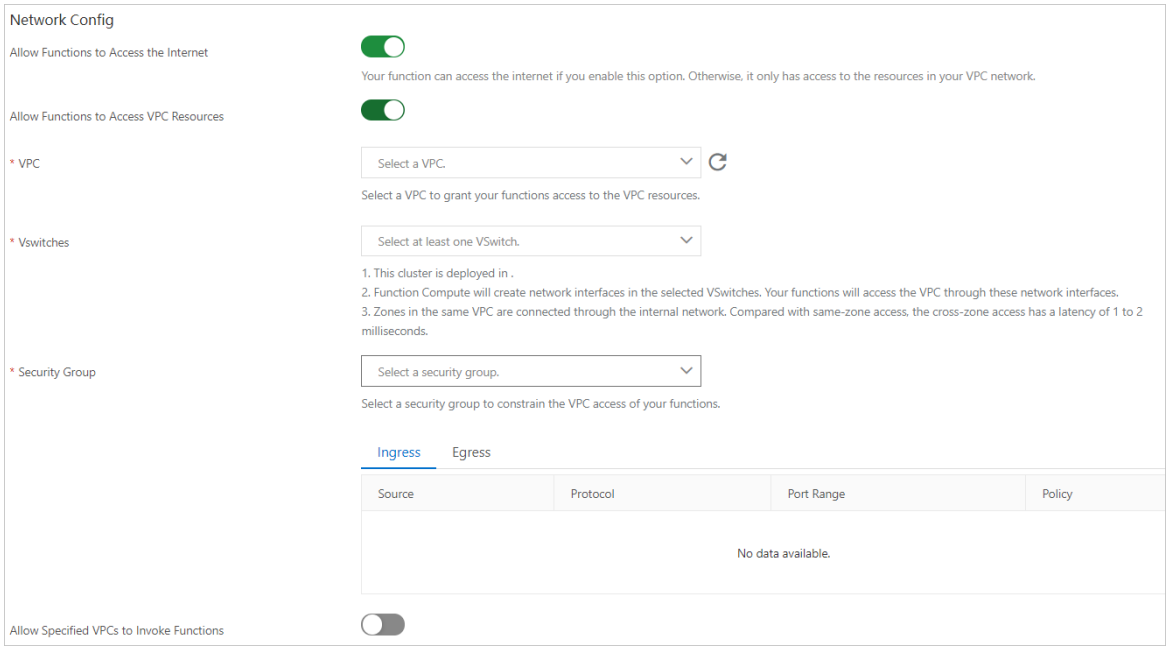
The VPC and permissions are configured on the service level. When you grant access permissions to a service in Function Compute, all functions in the service are authorized to access the specified VPC.

> **Note** If your resources are not deployed in a zone where Function Compute is available, create a vSwitch in your VPC. The vSwitch must be in the same zone as Function Compute. In addition, you must specify the vSwitch ID in the configuration of the specified service in Function Compute. vSwitches in the same VPC can communicate with each other. Therefore, Function Compute can use the vSwitch to access resources that are deployed in the VPC and reside in other zones.

1. Log on to the Function Compute console.
2. In the top navigation bar, select a region.
3. In the left-side navigation pane, click **Services and Functions**. In the **Services** pane, click the service that you require.
4. On the **Services and Functions** page, click the service that you require. Then, click the **Service Configurations** tab. On the **Service Configurations** tab, click **Modify Configuration**.

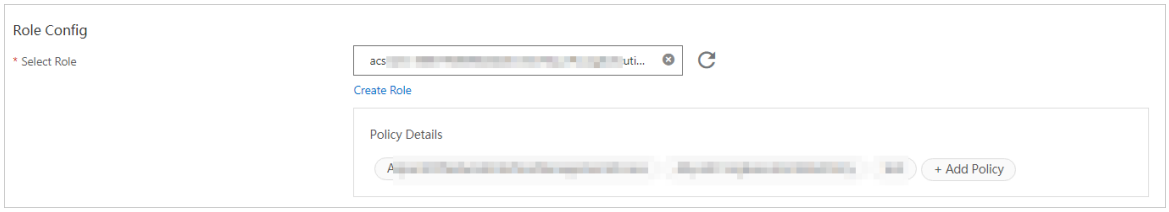


5. In the **Network Config** section, modify the network configurations as needed.



Parameter	Description
Allow Functions to Access the Internet	Specifies whether to allow functions to access the Internet. ○ On: Functions can access the Internet. ○ Off: Functions cannot access the Internet.
Allow Functions to Access VPC Resources	Specifies whether to allow functions to access VPC resources. ○ On: Functions can access VPC resources. If you turn on the switch, you must set the following parameters: ■ **VPC**: Select a VPC. ■ **Vswitches**: Select one or more vSwitches. ■ **Security Group**: Select the security group with which your ENI is associated. ○ Off: Functions cannot access VPC resources.
Allow Specified VPCs to Invoke Functions	Specifies whether to allow functions to be invoked only in specified VPCs. For more information, see Allow functions to be invoked only in specified VPCs.

6. In the **Role Config** section, set the parameters.
   Function Compute accesses resources deployed in a VPC by using an ENI. Therefore, you must grant the service that needs to access the specified VPC permissions to create, describe, and delete ENIs. For more information, see Permission management.



Parameter	Description
Select Role	Select an existing role from the drop-down list or click **Create Role**.
Select Policy Template	Select **AliyunECSNetworkInterfaceManagementAccess** from the drop-down list. This policy contains the following permissions: ○ vpc:DescribeVSwitchAttributes ○ ecs:CreateNetworkInterface ○ ecs:DeleteNetworkInterface ○ ecs:DescribeNetworkInterfaces ○ ecs:CreateNetworkInterfacePermission ○ ecs:DescribeNetworkInterfacePermissions

7. Click **Submit**.

7.2. Troubleshooting

Why am I unable to connect Function Compute to a VPC for debugging?

If you have set a virtual private cloud (VPC) configuration for a service in Function Compute but the service fails to connect to the specified VPC, the failure may occur due to the following causes:

- An error may have occurred with the subnet with which the vSwitch associates, or IP addresses are insufficient. We recommend that you specify multiple vSwitch IDs. This allows your functions to correctly run in other zones if an error occurs with the current one.
- The security group is incorrectly configured. The following requirements must be met when you configure the security group. For more information about how to configure a security group, see Add security group rules.
  - In the security group with which the specified VPC is associated, a rule is configured to allow access from the security group with which Function Compute is associated.
  - The outbound traffic of the security group must support Internet Control Message Protocol (ICMP). Function Compute checks the VPC network connectivity based on ICMP.

Why does a network connection error occur when I invoke a function to access cloud resources?

To allow Function Compute to access resources that are deployed in a VPC, the execution environment has been migrated from the classic network to the VPC. Therefore, a network connection error may occur when you invoke a function to access cloud services, such as Elastic Compute Service (ECS). You can troubleshoot the error based on the following solutions in different scenarios:

- A network connection error occurs when you invoke a function to access the internal endpoint of a cloud service. You must use the VPC endpoint of the cloud service to access the cloud service. If the destination cloud service does not provide a VPC endpoint, set the InternetAccess field for the service where the function is created to true and then access the destination cloud service by using a public endpoint that is provided by the destination cloud service.

  Notice You are charged for the network traffic that is generated when you access the destination cloud service by using a public endpoint.

- A network connection error occurs when you invoke a function to access self-managed ECS resources, such as web services and file systems, that are deployed in the classic network, or ApsaraDB RDS databases that are connected to the classic network.
  - If you need to use ECS resources or ApsaraDB RDS databases that are connected to the classic network, access them by using a public IP address or a public network. You are charged for the network traffic that is generated by using these access methods.
  - If you are able to migrate resources to a VPC, you can access the resources in the VPC by using Function Compute. For more information, see Configure functions to access VPC resources.
- A network connection error occurs when you invoke a function to access an ApsaraDB RDS instance. After you create an RDS instance, you must configure a whitelist to access the instance. For more information, see Switch an ApsaraDB RDS for MySQL instance to the enhanced whitelist mode.

  Note No security risk is incurred when you allow all IP addresses to access the RDS instance in the VPC.

Troubleshoot errors

If you have set a VPC configuration for a service in Function Compute, Function Compute cannot verify access permissions when the service accesses the specified VPC. Permissions are verified only when a function is executed. Therefore, new errors may occur when you call the InvokeFunction operation to invoke a function. The following table describes specific common errors that occur when a service in Function Compute accesses a VPC so that you can troubleshoot the errors with efficiency.

Error code	HTTP status code	Cause	Solution
InvalidArgument	400	Function Compute does not support the zone of the specified vSwitch.	Specify another vSwitch ID.
		The resources specified by the `vpcId`, `vSwitchIds`, or `securityGroupId` field defined in the VPC configuration cannot be found.	Check whether the VPC configuration is correctly set.

Error code	HTTP status code	Cause	Solution
		The vSwitch and security group are not associated with the specified VPC.	Check whether the VPC configuration is correctly set. Make sure that the resources specified by the `vSwitchId` and `securityGroupId` fields are deployed in the VPC that is specified by the vpcId field.
AccessDenied	403	You have not granted operation permissions on elastic network interfaces (ENIs) to the service in Function Compute.	Check the operation permissions of the service. For more information, see Configure functions to access VPC resources.
ResourceExhausted	429	All ENIs in the specified VPC have been used and Function Compute cannot create ENIs.	Provide more ENIs for the specified VPC.

8.Instances in reserved mode

8.1. Overview

Function Compute provides two modes for instance management: the pay-as-you-go mode and the reserved mode. This topic describes the characteristics of these two modes.

Pay-as-you-go mode

In pay-as-you-go mode, instances are allocated and released by Function Compute. When Function Compute receives function invocation requests, it dynamically schedules resources to provide an elastic and reliable execution environment and help simplify resource management.

However, cold start is unavoidable during dynamic scheduling of resources, which has negative impacts on online applications that are sensitive to response latency.

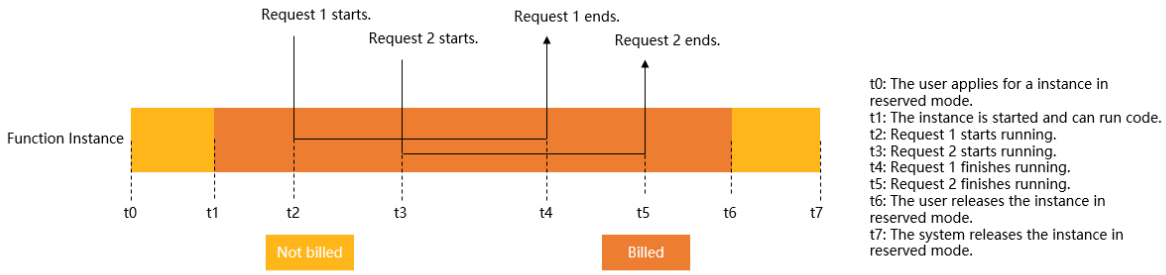
Reserved mode

In reserved mode, instances are allocated and released by users, and are billed based on their running duration.

An instance in reserved mode is ready for use after it is created. This eliminates the impacts caused by cold start.

By default, instances in reserved mode are prioritized over those in pay-as-you-go mode. When Function Compute receives function invocation requests, it preferentially uses instances in reserved mode to handle the requests. If the instances in reserved mode are insufficient to handle all the requests, Function Compute adds instances in pay-as-you-go mode as an addition to handle the remaining requests.

An instance in reserved mode is billed based on its running duration, which starts when the instance is started and ends when the instance is released. Therefore, even if an instance in reserved mode that is not released does not process any requests, you must pay for it.



> **Note** Before you call a Function Compute API operation to release an instance in reserved mode, make sure that no new requests are sent to the instance.

For more information about pricing and billing, see Billing.

8.2. Manage provisioned instances

This topic describes how to create and modify provisioned instances in the Function Compute console.

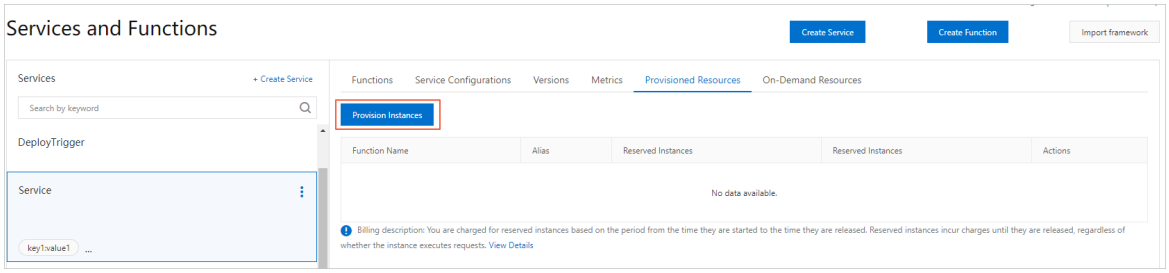
Prerequisites

Before you begin, make sure that the following operations are complete:

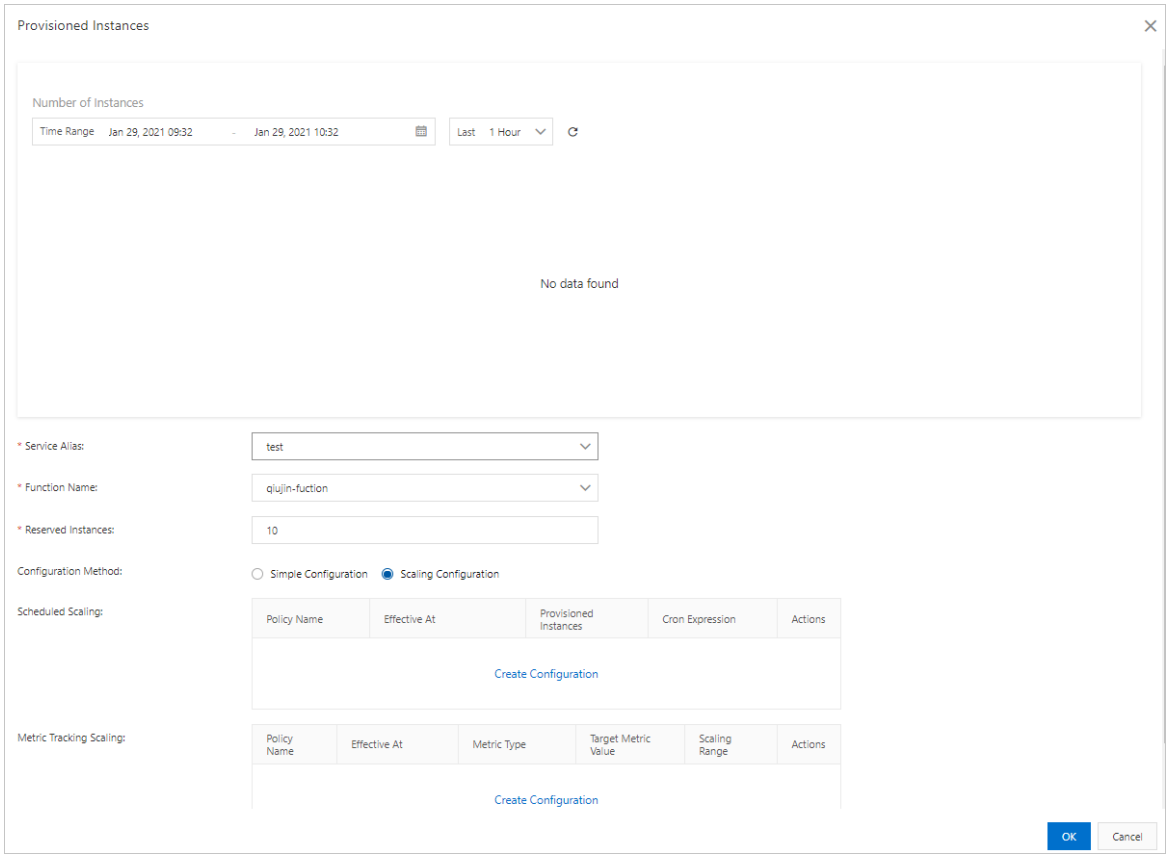
- Create a version.
- Create an alias.

Procedure

1. Log on to the Function Compute console.
2. In the top navigation bar, select a region.
3. In the left-side navigation pane, click **Services and Functions**. In the **Services** pane, click the service that you require.
4. On the **Services and Functions** page, click the service that you require. Then, click the **Provisioned Resources** tab. On the **Provisioned Resources** tab, click **Provision Instances**.



5. In the **Provisioned Instances** dialog box, set the parameters and click **OK**.

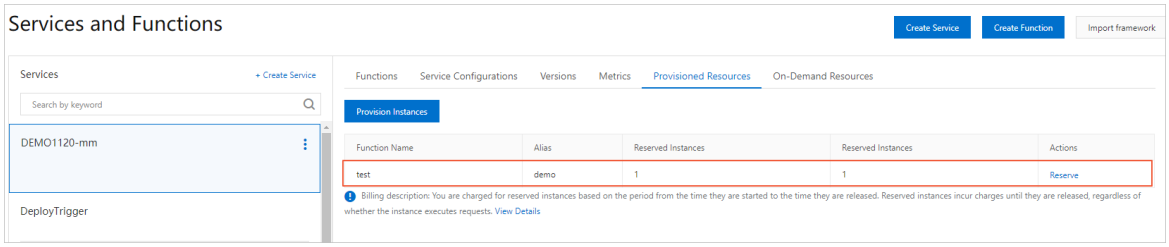


> **Note** You cannot create provisioned instances for the LATEST version. You must create a version and an alias before you create provisioned instances. For more information, see Manage aliases.

Parameter	Description
Service Alias	The alias of the service to which the function that you want to use the provisioned instances to execute belongs.
Function Name	The function that you want to use the provisioned instances to execute.
Reserved Instances	The number of provisioned instances to be created. **Note** You can set this parameter based on the number of instances in use, which is displayed in the **Number of Instances** section in the Provisioned Instances dialog box.

Parameter	Description
Configuration Method	Select a configuration method as needed. You can view the chart that displays the number of instances that are being used to execute the function based on different configuration methods. ○ Simple Configuration: If you can make full use of provisioned instances and the number of provisioned instances does not fluctuate greatly, you can select Simple Configuration. ○ Scaling Configuration: Select an auto scaling type of provisioned instances as needed to make better use of provisioned instances. ■ Scheduled Scaling: You can configure scheduled auto scaling to flexibly configure provisioned instances. You can configure the number of provisioned instances to be automatically adjusted to a specified value at a specified time. This way, the number of provisioned instances can meet the concurrency of your business. ■ **Policy Name**: Enter a custom policy name. ■ **Effective Time**: Set the time when the configuration of scheduled auto scaling starts to take effect and the time when the configuration of scheduled auto scaling expires. ■ **Provisioned Quantity**: Enter the number of provisioned instances as needed. ■ **Schedule Expression**: Enter the scheduled expression that specifies when to perform scheduling. Two formats are supported. For more information, see the Parameters table in the "Scheduled auto-scaling" section of the Auto-scaling of reserved instances topic. ■ Metric Tracking Scaling: Provisioned instances are scaled in or out every minute based on the concurrency utilization of provisioned instances. ■ **Policy Name**: Enter a custom scheduled task name. ■ **Effective Time**: Set the time when the configuration of metric tracking auto scaling starts to take effect and the time when the configuration of metric tracking auto scaling expires. ■ **Metric Type**: Select the type of the metric to be tracked from the drop-down list. ■ **Target Metric Value**: Set the expected usage rate. If the usage rate is lower than the value of this parameter, the system scales in provisioned instances. If the usage rate is higher than the value of this parameter, the system scales out provisioned instances. ■ **Scaling Range**: Set the **Min Provisioned Instances** and **Max Provisioned Instances** parameters as needed.

You can view the created provisioned instances on the Provisioned Resources tab.



Change the number of provisioned instances

1. Log on to the Function Compute console.
2. In the top navigation bar, select a region.
3. In the left-side navigation pane, click **Services and Functions**.
4. In the Services pane, click the service that you require.
5. Click the **Provisioned Resources** tab. On the Provisioned Resources tab, find the function that you require.
6. Click **Edit** in the **Actions** column.
7. In the **Provisioned Instances** dialog box, change the value of the Reserved Instances parameter and click **OK**.

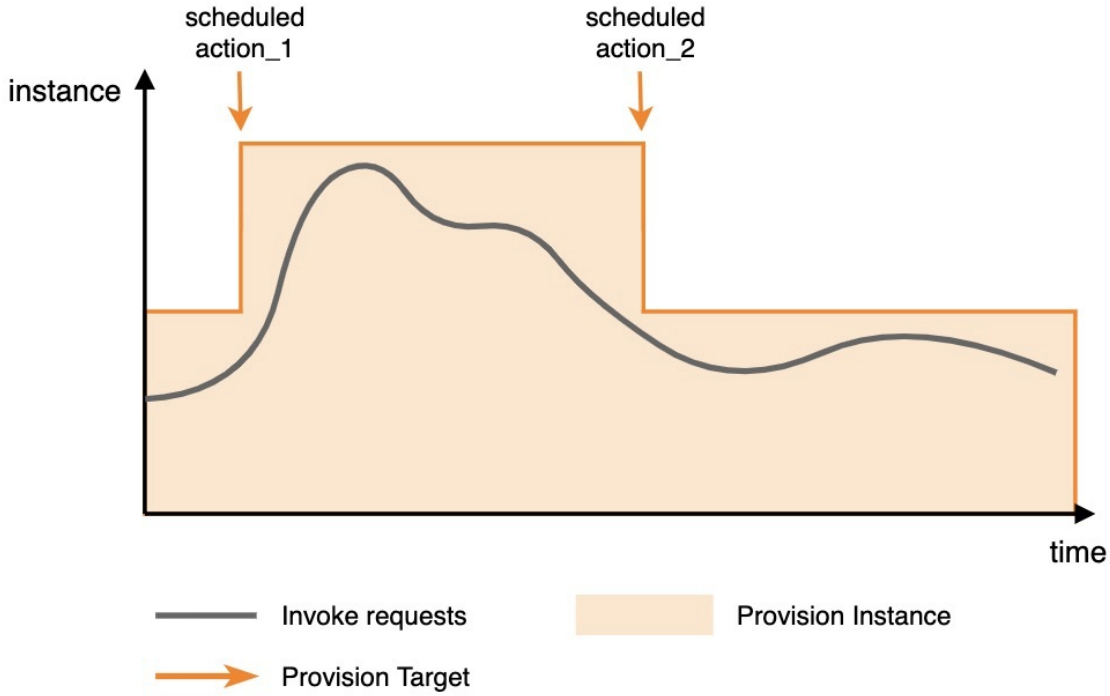
> **Note** To delete provisioned instances, set the **Reserved Instances** parameter to 0.

8.3. Auto-scaling of reserved instances

The reserved mode allows you to reserve instances in response to function invocation requests. This reduces the occurrences of cold starts and improves the response speed for latency-sensitive online services. You can perform scheduled auto-scaling or metric tracing auto-scaling to make better use of reserved instances.

Scheduled auto-scaling

- Definition: Scheduled auto-scaling is used to flexibly configure reserved instances. You can configure the number of reserved instances to be automatically adjusted to a specified value at a specified time so that the number of instances can meet the concurrency of your business.
- Scenario: You can use scheduled auto-scaling to reserve instances in advance of periodic or predicted traffic peaks for functions. When the number of instances invoked concurrently by a function is higher than the reserved instance concurrency, the excess instances are charged on a pay-as-you-go basis.
- Example of configuration: Two scheduled operations are configured. The first scheduled operation scales out the reserved instances before the traffic peak, and the second scheduled operation scales in the reserved instances after the traffic peak.



Example of parameter settings:

- In this example, a function named function_1 in a service named service_1 is configured to automatically scale in and out. Set the scaling period for function_1 to the period from 10:00:00 on November 1, 2020 to 10:00:00 on November 30, 2020 (UTC+8). The number of reserved instances is scaled out to 50 at 20:00 every day and scale in to 10 at 22:00 every day.

```
{
 "ServiceName": "service_1",
 "FunctionName": "function_1",
 "Qualifier": "alias_1",
 "SchedulerActions": [
  {
   "Name": "action_1",
   "StartTime": "2020-11-01T10:00:00Z",
   "EndTime": "2020-11-30T10:00:00Z",
   "TargetValue": 50,
   "ScheduleExpression": "cron(0 0 20 * * *)"
  },
  {
   "Name": "action_2",
   "StartTime": "2020-11-01T10:00:00Z",
   "EndTime": "2020-11-30T10:00:00Z",
   "TargetValue": 10,
   "ScheduleExpression": "cron(0 0 22 * * *)"
  }
 ]
}
```

- Parameters

Parameter	Description	Schema
Name	The name of the scheduled auto-scaling task.	string
StartTime	The time when the configuration starts to take effect. Specify the value in UTC.	string
EndTime	The time when the configuration expires. Specify the value in UTC.	string
TargetValue	The number of instances to be reached.	integer (int64)
ScheduleExpression	The scheduled expression that specifies when to run the scheduled task. The following formats are supported: ○ At expressions - "at(yyyy-mm-ddThh:mm:ss)": runs the scheduled task only once. Specify the value in UTC. ○ Cron expressions - "cron(0 0 20 * * *)": runs the scheduled task multiple times. Specify the value in the standard CRON format. For example, cron(0 0 20 * * *) indicates that the scheduled task is run at 20:00 every day.	string

The following tables describe the fields and special characters of the CRON expression (Seconds Minutes Hours Day-of-month Month Day-of-week).

Fields

Field name	Valid value	Allowed special characters
Seconds	0 to 59	None
Minutes	0 to 59	, - * /
Hours	0 to 23	, - * /
Day-of-month	1 to 31	, - * ? /
Month	1 to 12 or JAN to DEC	, - * /
Day-of-week	1 to 7 or MON to SUN	, - * ?

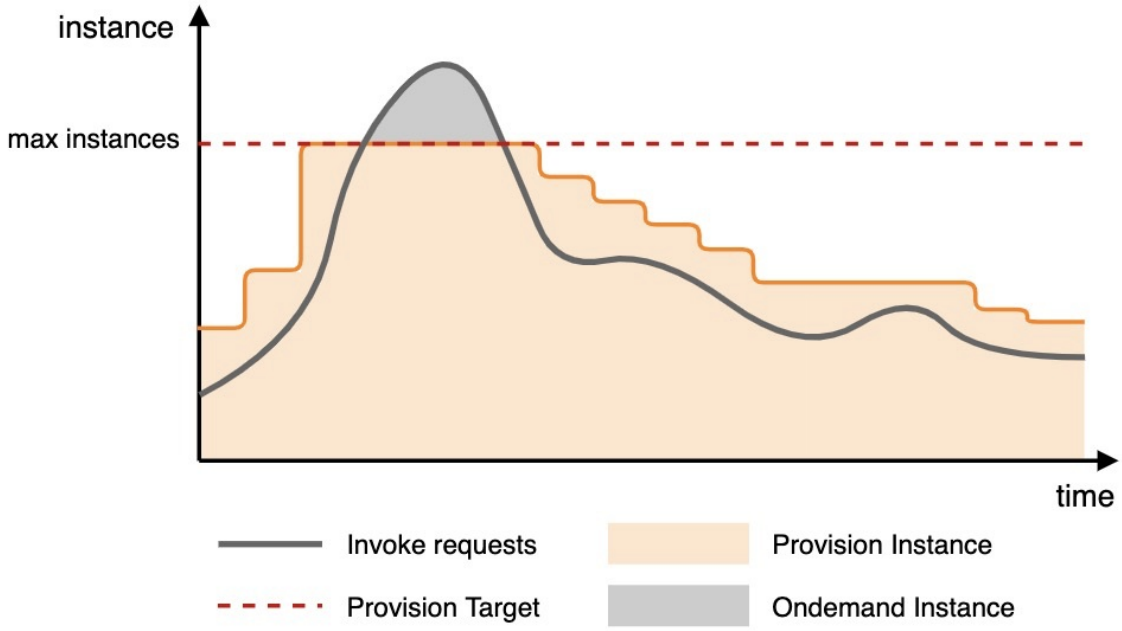
Special characters

Character	Description	Example
*	Indicates any or each.	In the **Minutes** field, 0 indicates that the task is run at the 0th second of every minute.
,	Indicates the list value.	In the **Day-of-week** field, MON, WED, and FRI indicate Monday, Wednesday, and Friday.
-	Indicates a range.	In the **Hours** field, 10-12 indicates that the time range is from 10:00 to 12:00 in UTC.
?	Indicates an uncertain value.	This value is used with other specified values. For example, if you specify a specific date, but you do not care what day of the week it is, you can use this special character in the **Day-of-week** field.
/	Indicates the increment of a value. For example, n/m means to add an increment m to n each time.	In the **minute** field, 3/5 indicates that the operation is performed every 5 minutes starting from the minute 3.

Metric tracking auto-scaling

- Definition: Metric tracking auto-scaling tracks the metrics to dynamically scale reserved instances.
- Scenario: Function Compute periodically collects the concurrency usage rate of reserved instances, and uses this metric together with the scale-out and scale-in trigger values you configured to control the scaling of reserved instances. In this way, the number of reserved instances can be scaled based on your business needs.
- Principle: Reserved instances are scaled in or out every minute based on the metric.
  - When the metric exceeds the scale-out threshold, the system scales out the number of instances to the destination value as soon as possible.
  - When the metric is lower than the scale-in threshold, the system slightly scales in the number of instances to the destination value.

  If the maximum and minimum numbers of reserved instances are configured, the system scales the number of reserved instances between the maximum and minimum numbers. If the number of instances reaches the value range, scaling stops.
- Example of configuration:



  - When the traffic increases and the number of required instances reaches 80% of the scale-out threshold, the number of reserved instances starts to scale out till it reaches the scale-out threshold. Requests that cannot be processed by reserved instances are sent to pay-as-you-go instances.
  - When the traffic decreases and the number of required instances reaches 60% of the scale-in threshold, the number of reserved instances starts to scale in.

Statistics only on reserved instances are collected to calculate the concurrency usage rate of reserved instances. The statistics on the pay-as-you-go instances are not included. The metric is calculated based on the following formula: Number of concurrent requests to which reserved instances are responding/Maximum number of concurrent requests to which all reserved instances can respond. The metric value ranges from 0 to 1. The maximum number of concurrent requests to which reserved instances can respond is calculated based on different Manage a function:

- Single request processed by one instance: Maximum concurrency = Number of instances.
- Multiple requests processed by one instance: Maximum concurrency = Number of instances × Number of requests concurrently processed by one instance.

Scale-in and scale-out values:

- The values are determined by the current metric value, scaling threshold, number of reserved instances, and scaling factor.
- Calculation principle: The system scales in based on a scale-in factor. Value range: (0,1]. You do not need to set this factor. The scale-in and scale-out values are rounded-up integers of the following calculation results:
  - Scale-out value = (Current metric value/Scale-out threshold) × Number of reserved instances.
  - Scale-in ratio = (1 - Current metric value/Scale-out threshold) × Scale-in factor.
  - Scale-in value = Current number of instances × (1 - Scale-in ratio).
- Example: The current metric value is 90%, the scale-out threshold is 80%, and the number of reserved instances is 100. The scale out value = (90%/80%) × 100 = 112.5 (rounded up to 113). The number of reserved instances is increased to 113.

Example of parameter settings:

- In this example, a function named function_1 in a service named service_1 is configured to automatically scales in and out based on the ProvisionedConcurrencyUtilization metric. Set the scaling period from 10:00:00 on November 1, 2020 to 10:00:00 on November 30, 2020. When the concurrency usage rate exceeds 60%, the number of reserved instances can scale out to 100. When the concurrency usage rate is lower than 60%, the number of reserved instances can scale in to 10.

```
{
 "ServiceName": "service_1",
 "FunctionName": "function_1",
 "Qualifier": "alias_1",
 "TargetTrackingPolicies": [
  {
   "Name": "action_1",
   "StartTime": "2020-11-01T10:00:00Z",
   "EndTime": "2020-11-30T10:00:00Z",
   "MetricType": "ProvisionedConcurrencyUtilization",
   "MetricTarget": 0.6,
   "MinCapacity": 10,
   "MaxCapacity": 100,
  }
 ]
}
```

- Parameters

Parameter	Description	Schema
Name	The name of the scheduled auto-scaling task.	string
StartTime	The time when the configuration starts to take effect. Specify the value in UTC.	string
EndTime	The time when the configuration expires. Specify the value in UTC.	string
MetricType	The tracked metric: ProvisionedConcurrencyUtilization.	string
MetricTarget	The tracked value of the metric.	double
MinCapacity	The minimum scale-in value.	integer (int64)
MaxCapacity	The maximum scale-out value.	integer (int64)

9.Tags

9.1. Overview

Function Compute allows you to classify service resources that have the same features by using tags. Tags facilitate resource search and aggregation.

Scenarios

Using tags to group and classify an increasing number of services facilitates resource search and aggregation.

- You can attach different tags to services in different environments, such as the production environment and the test environment. For example, you can attach the Env:Test key-value pair to all services in the test environment. This way, you can efficiently filter services to obtain the services that meet the specified conditions.
- You can attach tags to services by group, project, or department to facilitate team or project management. For example, you can use the FinanceDept:FinanceJoshua tag to implement group-based authorization. For more information, see Use tags to perform group-based service authorization.

Usage notes

- Tags and service resources are in an N-to-N relationship.
- Each tag consists of a key-value pair.
- A tag serves as a condition to implement fine-grained authorization on resources within a specified scope.
- All resources of a service, such as versions, aliases, functions, and triggers, inherit the tag attached to this service.
  - Tag-based authorization is supported if you specify the service in your API request.
  - Different versions of a service use the same tag. Therefore, a change to the tag of a service affects tag-based authorization that involve all versions and aliases of the service.

Limits

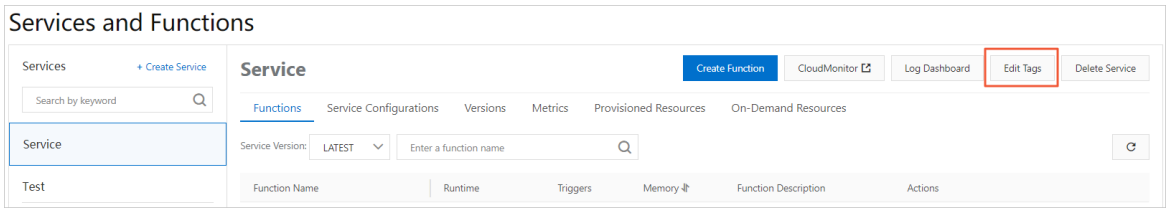
- Each tag key must be 1 to 64 case-sensitive Unicode characters in length.
- Each tag value must be 1 to 128 case-sensitive Unicode characters in length.
- Each tag key on a resource can have only one value. If you add a tag that has the same key as an existing tag on a resource, the new value overwrites the original value.
- Each resource can have a maximum of 20 tags.
- A tag key cannot start with aliyun or acs:, contain http:// or https://, or be an empty string.
- A tag value cannot contain http:// or https://, or be an empty string.
- Tags cannot be used across regions. For example, tags created in the China (Hangzhou) region are invisible in the China (Shanghai) region.

9.2. Manage tags

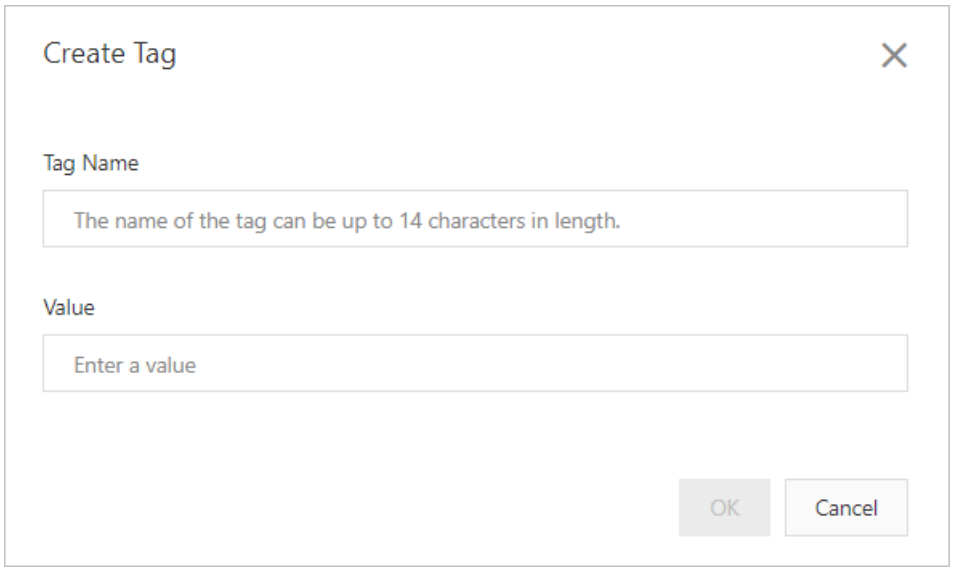
This topic shows you how to create, update, and delete tags, and query services by tag in the Function Compute console.

Create tags

1. Log on to the Function Compute console.
2. In the top navigation bar, select a region.
3. In the left-side navigation pane, click **Services and Functions**.
4. On the **Services and Functions** page, click the service that you require. Then, click **Edit Tags** in the upper-right corner.

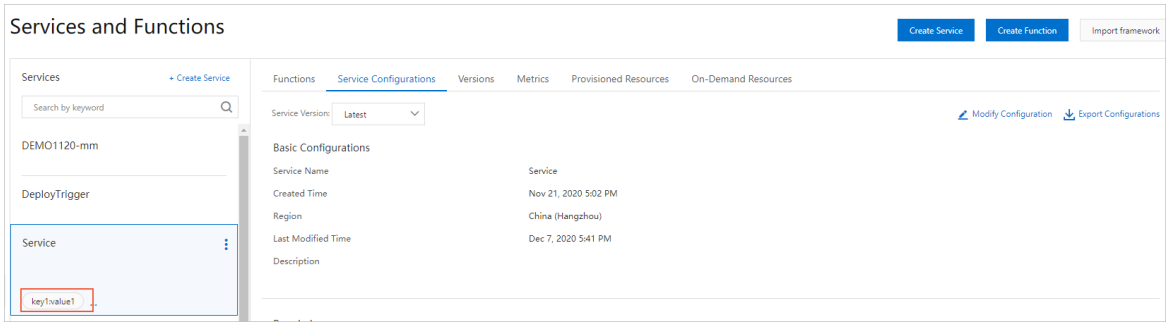


5. In the **Tags** panel, click **Create Tag**.
6. In the **Create Tag** dialog box, enter the tag name and value, and click **OK**.



To create multiple tags for the service, repeat this step and enter multiple key-value pairs. For more information about the instructions and limits of tags, see Overview.

7. In the **Tags** panel, click **Save**.
Move the pointer over the service to view the created tags.



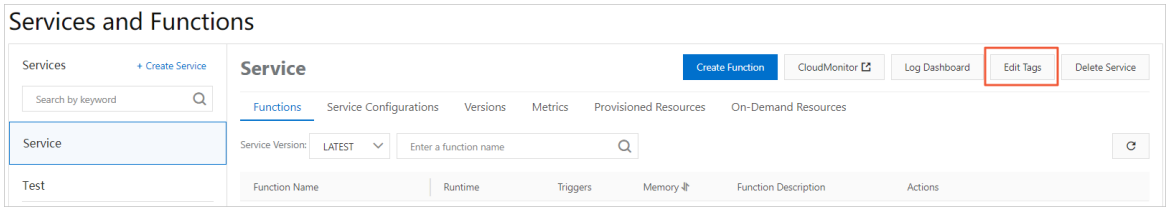
Update tags

For a resource, each tag key must be unique. If a new tag has the same key as an existing tag, the new tag overwrites the existing tag. Therefore, you can update tags by overwriting existing tags.

For example, the key:value tag exists. If you need to update the tag value to value1, you can create the key:value1 tag. This new tag overwrites the key:value tag.

> **Note** A new tag overwrites an existing tag only when they have the same key.

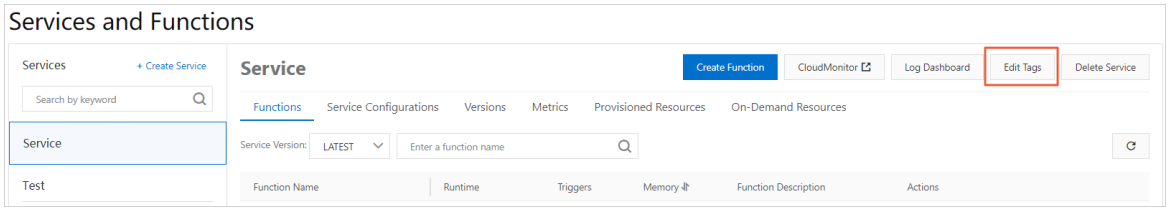
1. Log on to the Function Compute console.
2. In the top navigation bar, select a region.
3. In the left-side navigation pane, click **Services and Functions**.
4. On the **Services and Functions** page, click the service that you require. Then, click **Edit Tags** in the upper-right corner.



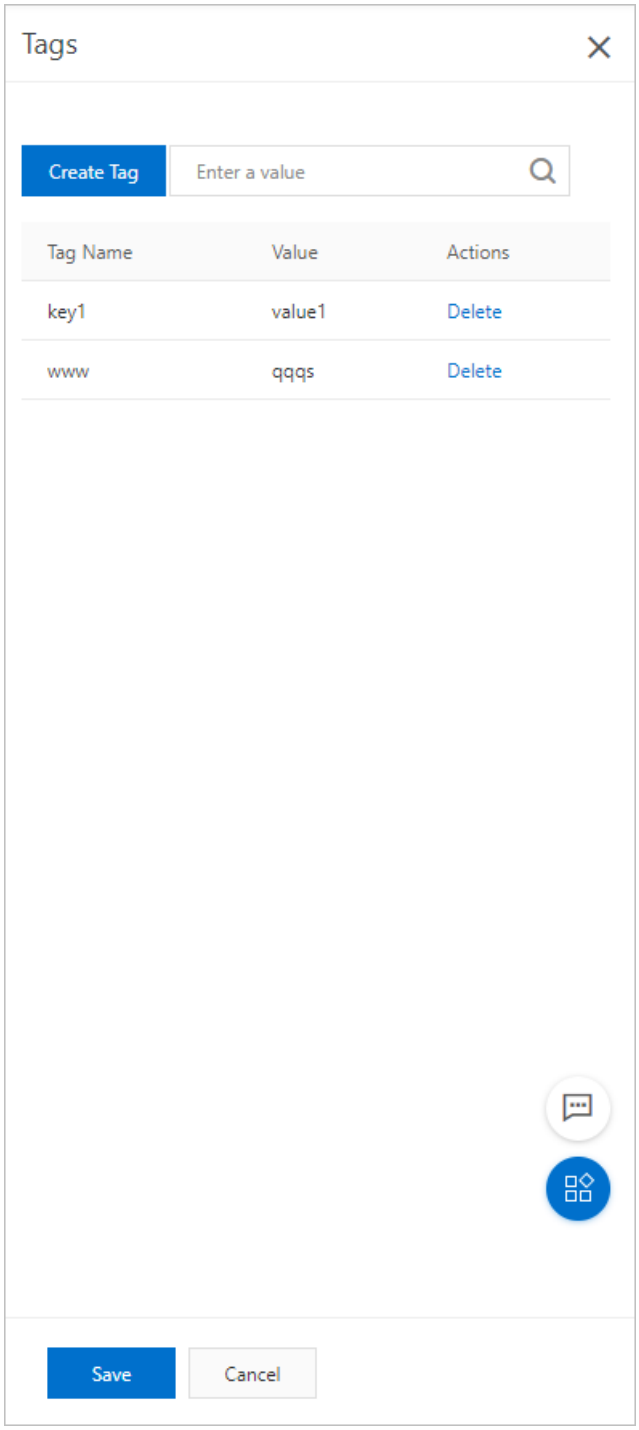
5. In the **Tags** panel, click **Create Tag**.
6. In the **Create Tag** dialog box, enter the new key-value pair and click **OK**.
7. In the **Tags** panel, click **Save**.

Delete tags

1. Log on to the Function Compute console.
2. In the top navigation bar, select a region.
3. In the left-side navigation pane, click **Services and Functions**.
4. On the **Services and Functions** page, click the service that you require. Then, click **Edit Tags** in the upper-right corner.



5. In the **Tags** panel, find the tag that you want to delete and click **Delete** in the Actions column. Then, click **Save**.



9.3. Use tags to perform group-based service authorization

You can use the tagging feature to group services and authorize different roles to manage services in different groups.

Sample scenario

Assume that you have created 10 services in the Function Compute console. You want to authorize the dev team to manage five services and the ops team to manage the other five services. The dev and ops teams can view only the services that they are authorized to manage.

You can use the tagging feature to add teams to different groups and grant different permissions to different groups. In this scenario, you can attach the team:dev tag to five services and the team:ops tag to the other five services.

Procedure

1. Attach the team:dev tag to the five services that you want to authorize the dev team to manage, and attach the team:ops tag to the five services that you want to authorize the ops team to manage. For more information, see Create tags.
2. Create a RAM user.
3. Create a RAM user group.Create two user groups named dev and ops.
4. Add a RAM user to a RAM user group.Create RAM users and add them to the corresponding user groups.
5. Grant different permissions to the two user groups.Function Compute supports system policies and custom policies. You can select an appropriate policy as needed.
   - Grant permissions to different user groups by using system policies.
     For more information, see Grant permissions to a RAM user group.
   - Grant permissions to different user groups by using custom policies.

a. Create a custom policy.
Assume that the custom policy named policyForDevTeam is used to grant permissions to the dev team. The following example shows the policy:

```
{
  "Statement": [
  {
    "Action": "fc:*",
    "Effect": "Allow",
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "fc:tag/team": "dev"
      }
    }
  },
  {
    "Action": "fc:*",
    "Effect": "Allow",
    "Resource": "*"
  }
  ],
  "Version": "1"
}
```

Assume that the custom policy named policyForOpsTeam is used to grant permissions to the ops team. The following example shows the policy:

```
{
  "Statement": [
  {
    "Action": "fc:*",
    "Effect": "Allow",
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "fc:tag/team": "ops"
      }
    }
  },
  {
    "Action": "fc:*",
    "Effect": "Allow",
    "Resource": "*"
  }
  ],
  "Version": "1"
}
```

b. Grant permissions to a RAM user group.
Select the created custom policies when you grant permissions to user groups.

After the authorization is complete, the RAM users in the dev user group can manage only the services tagged with team:dev and the RAM users in the ops user group can manage only the services tagged with team:ops.

10.Access database

10.1. Overview

This topic describes the mechanism for Function Compute to access databases in a virtual private cloud (VPC).

Access mechanism

Function Compute dynamically allocates instances for running functions. Therefore, you cannot add the dynamic IP addresses of these instances to the whitelist of a database. Specifically, you cannot control the access of Function Compute to a database by using a whitelist. In addition, based on the principle of least privilege, we recommend that you do not add the IP address 0.0.0.0/0 to the whitelist of your database in your production environment.

To resolve the preceding issue, you can grant Function Compute the permissions to access resources in a specified VPC. You can deploy your database in a secure VPC and enable the service to which the function belongs to access the VPC.

The following figure shows the process for Function Compute to access a database.

1. The client sends a request to Function Compute.
2. Function Compute accesses the database in the specified VPC when the service to which the specified function belongs is enabled to access resources in the VPC.
   A VPC is a private network. Function Compute and the database reside in different VPCs. Therefore, Function Compute must use an elastic network interface (ENI) to access the database across VPCs. You must authorize an ENI to access the specified VPC, and bind the ENI to the instance that executes functions. For more information, see Configure functions to access VPC resources.

   > **Note** vSwitches in the same VPC can communicate with each other. Assume that the vSwitch in the VPC where the database resides is not in a zone supported by Function Compute. You can create a vSwitch in a zone supported by Function Compute in this VPC and configure the ID of the created vSwitch in the VPC configuration of the service in Function Compute. This way, you can use vSwitches in different zones to achieve interconnection.
3. Function Compute returns the obtained data to the client.

References

- What is a VPC?
- Overview

10.2. Make preparations

This topic describes the preparations that you must make to enable Function Compute to access a database. Specifically, you must create a virtual private cloud (VPC) and a vSwitch, create a database instance, configure a whitelist, and create a security group.

Step 1: Create a VPC and a vSwitch

Step 1: Create a VPC and a vSwitch

You must create a vSwitch in a zone that is supported by Function Compute. For more information about the supported zones, see Configure functions to access VPC resources. For more information about how to create a VPC and a vSwitch, see Create an IPv4 VPC network.

Step 2: Create a database instance

You must create a database instance in the same VPC as Function Compute. You do not need to create the vSwitch in the same zone as Function Compute. vSwitches in the same VPC can communicate with each other regardless of whether they are in the same zone. You can refer to the following topics to create the database instances that you want to access:

- Create an ApsaraDB RDS for MySQL instance
- Create an ApsaraDB for MongoDB instance that meets your business needs
  - Create a standalone instance
  - Create a replica set instance
  - Create a sharded cluster instance
- Create an ApsaraDB for Redis instance
- Create an ApsaraDB RDS for SQL Server instance
- Create an ApsaraDB RDS for PostgreSQL instance

Step 3: Configure a whitelist

Enter the Classless Inter-Domain Routing (CIDR) blocks of the specified VPCs in the **IP Addresses** field to configure an IP address whitelist for the database instance. You can log on to the VPC console and find the CIDR block of a VPC on the **VPCs** page. For more information, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.

Step 4: Create a security group

The security group must have outbound traffic allowed from the internal CIDR block and port of the database. For more information about how to create a security group, see Create a security group.

10.3. Access an ApsaraDB RDS for MySQL database

In Function Compute, you can use a function to call the ApsaraDB RDS for MySQL API to perform operations such as insert and query on an ApsaraDB RDS for MySQL instance. In normal cases, states cannot be shared among different execution environments in Function Compute. You can persist structured data in a database so that states can be shared. This topic describes how to use FunCraft to deploy a function to access an ApsaraDB RDS for MySQL database.

Prerequisites

1. You have created a virtual private cloud (VPC) and a VSwitch.
   The VSwitch must belong to a zone supported by Function Compute. For more information about the zones supported by Function Compute, see VPC access.
2. Create an ApsaraDB RDS for MySQL instance
   You have created a database instance in same VPC as Function Compute. You do not need to create the VSwitch in the same zone as Function Compute. VSwitches under the same VPC can communicate with each other even when they are in different zones.
3. Configure an IP address whitelist for an ApsaraDB RDS for MySQL instance

If you need to control access to databases by using IP address whitelists, you must enter the internal Classless Inter-Domain Routing (CIDR) blocks of the virtaul private clouds (VPCs) or VSwitches in the **IP Addresses** field.
You can log on to the VPC console to obtain the internal CIDR block on the details page of the target VPC or VSwitch.

4. Create a security group
   You have configured the security group to allow outbound traffic from the internal CIDR block and port of the database.

Write a function

The following section describes how to use Funcraft to write a function that accesses a database. The sample code is written in Python 3.

1. On the local machine, create a directory to store code and dependent modules. In the directory, create the *template.yml* file. In this example, the */tmp/code/template.yml* file is created and contains the following content:

```
ROSTemplateFormatVersion: '2015-09-01'
Transform: 'Aliyun::Serverless-2018-04-03'
Resources:
 service:
  Type: 'Aliyun::Serverless::Service'
  Properties:
   Description: This is MYSQL service
   Policies:
    - AliyunECSNetworkInterfaceManagementAccess
   VpcConfig:
    VpcId: vpc-XXXX
    VSwitchIds:
     - vsw-XXX
    SecurityGroupId: sg-XXXX
   InternetAccess: true
  function:
   Type: 'Aliyun::Serverless::Function'
   Properties:
    Initializer: 'index.initializer'
    Handler: 'index.handler'
    Runtime: python3
    Timeout: 10
    MemorySize: 128
    CodeUri: '. /'
```

   The following list describes the main parameters:

   - A service named service is declared.
     - Policies: grants Function Compute permission to manage elastic network interfaces (ENIs) attached to Elastic Compute Service (ECS) instances. Function Compute can then access resources in the VPC.

- VpcConfig: binds the VPC to the service. You must replace the following values with the information for your VPC.
  - VpcId: the ID of the VPC.
  - VSwitchId: the ID of the VSwitch.
  - SecurityGroupId: the ID of the security group.
- A function named function is declared.
  - Initializer: the initializer function. For more information, see Initializer function.
  - Handler: the function handler. For more information, see Function entry point.
  - Runtime: the runtime environment of the function.
  - CodeUri: the directory where the code package is located.
    During function deployment, Funcraft packages and uploads the specified folders or files under the CodeUri directory.

2. In the directory that contains the *template.yml* file, create the *Funfile* file. In this example, the new file contains the following sample content:

```
RUNTIME python3
RUN fun-install pip install PyMySQL
```

3. Run the **fun install** command to install dependencies.

```
fun install
```

When the code editor shows the following content, the dependencies have been installed.

```
Install Success
```

4. In the directory that contains the *template.yml* file, create a Python file. In this example, the */tmp/code/index.py* file is created and contains the following sample content:

```
# -*- coding: utf-8 -*-
import logging
import pymysql
import os,sys
logger = logging.getLogger()
def getConnection():
  try:
  conn = pymysql.connect(
    host = 'MYSQL_HOST', //Replace the value with the internal endpoint of your ApsaraDB RDS for MySQL database.
    port = MYSQL_PORT, //Replace the value with the port number of your database.
    user = 'MYSQL_USER', //Replace the value with the user name of your database.
    passwd = 'MYSQL_PASSWORD', //Replace the value with the password for logging on to the database.
    db = 'MYSQL_DBNAME', //Replace the value with the name of your database.
    connect_timeout = 5)
  return conn
  except Exception as e:
  logger.error(e)
  logger.error("ERROR: Unexpected error: Could not connect to MySql instance.")
  sys.exit()
def conditionallyCreateUsersTable():
  try:
  conn = getConnection()
  with conn.cursor() as cursor:
    sql = """CREATE TABLE IF NOT EXISTS users (
    id      VARCHAR(64) NOT NULL,
    name    VARCHAR(128) NOT NULL,
    PRIMARY KEY(id))"""
    cursor.execute(sql)
  conn.commit()
  finally:
  conn.close()
def initializer(context):
  conditionallyCreateUsersTable()
def handler(event, context):
  try:
  conn = getConnection()
  with conn.cursor() as cursor:
    sql = "REPLACE INTO users (id, name) VALUES(%s, %s)"
    cursor.execute(sql, ('2', 'wan'))
    cursor.execute("SELECT * FROM users")
    result = cursor.fetchone()
    logger.info(result)
    return result
  finally:
  conn.close()
```

5. Run the following command to deploy the function to Function Compute by using Funcraft:

```
fun deploy
```

When the code editor shows the following content, the function has been deployed.

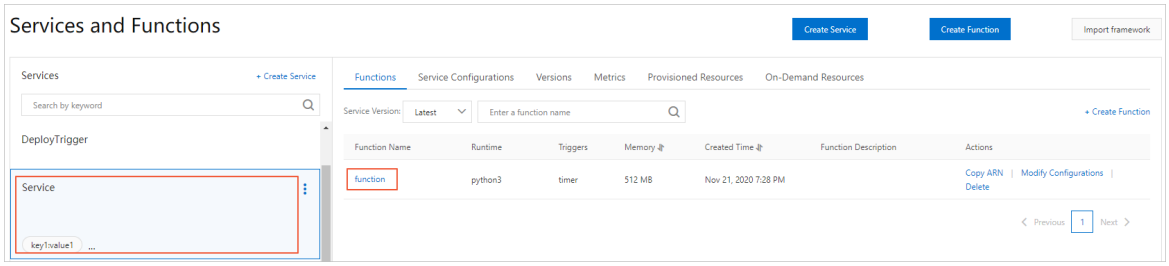
```
    function <function-name> deploy success
service <service-name> deploy success
```

After the deployment is complete, log on to the Failed to resolve content from t1882558.dita#task_2470088/xref_vds_2o7_t33 and click **Service/Function**. On the page that appears, you can see the newly deployed service and function.

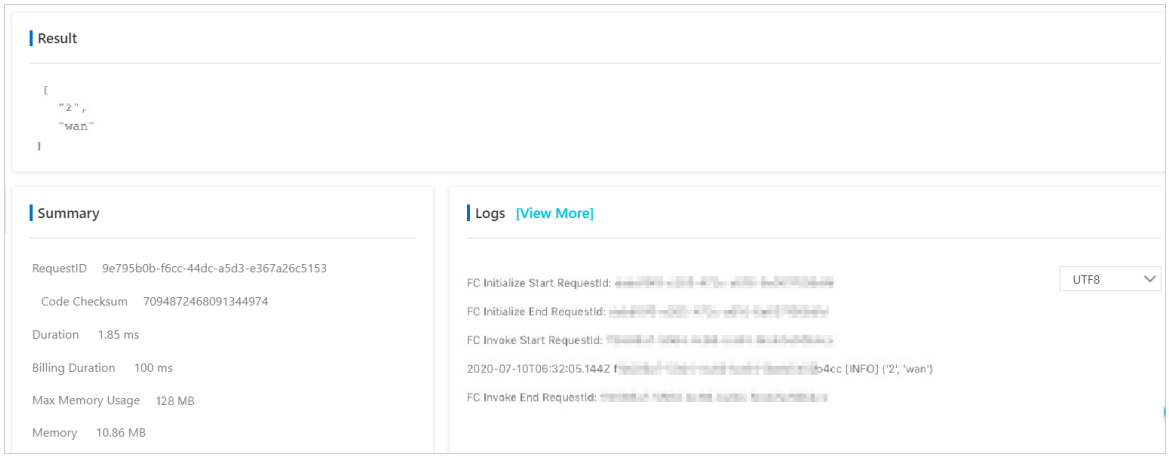
Debug the function

After the function is deployed, you can debug the function in the Function Compute console.

1.
2. In the top navigation bar, select a region.
3. In the left-side navigation pane, click **Services and Functions**.
4. In the **Services** pane, click the service that you require. On the **Functions** tab, click the name of the function that you require.



5. On the page that appears, click the **Code** tab. On this tab, click **Invoke**.
   After the execution is complete, you can view the execution results and logs.



More information

Examples of the access of Function Compute to MySQL databases

10.4. Access an ApsaraDB for MongoDB database

In Function Compute, you can use a function to call the ApsaraDB for MongoDB API to perform operations such as insert and query on an ApsaraDB for MongoDB database. In normal cases, states cannot be shared among different execution environments in Function Compute. You can persist structured data in a database so that states can be shared. This topic describes how to use Funcraft to deploy a function to access an ApsaraDB for MongoDB database.

Prerequisites

1. You have created a virtual private cloud (VPC) and a VSwitch.
   The VSwitch must belong to a zone supported by Function Compute. For more information about the zones supported by Function Compute, see VPC access.
2. An ApsaraDB for MongoDB instance that meets your business needs is created.
   - Create a standalone instance
   - Create a replica set instance
   - Create a sharded cluster instance

   You have created a database instance in same VPC as Function Compute. You do not need to create the VSwitch in the same zone as Function Compute. VSwitches under the same VPC can communicate with each other even when they are in different zones.
3. A whitelist is configured.
   - A whitelist for a standalone ApsaraDB for MongoDB instance is configured.
   - A whitelist for a replica set instance is configured.
   - A whitelist for a sharded cluster instance is configured.

   If you need to control access to databases by using an IP address whitelist, ensure that you have entered the internal Classless Inter-Domain Routing (CIDR) blocks of the target VPCs in the **IP** field. To find the CIDR block of a VPC, log on to the VPC console and view the **VPC Details** page.
4. Create a security group
   You have configured the security group to allow outbound traffic from the internal CIDR block and port of the database.

Write code for a function

The following section describes how to use Funcraft to write a function that accesses a database. The sample code is written in Python 3.

1. On the local machine, create a directory to store code and dependent modules. In the directory, create the *template.yml* file. In this example, the */tmp/code/template.yml* file is created and contains the following content:

```
ROSTemplateFormatVersion: '2015-09-01'
Transform: 'Aliyun::Serverless-2018-04-03'
Resources:
 service:
  Type: 'Aliyun::Serverless::Service'
  Properties:
   Description: This is MongoDB service
   Policies:
    - AliyunECSNetworkInterfaceManagementAccess
   VpcConfig:
    VpcId: vpc-XXXX
    VSwitchIds:
     - vsw-XXX
    SecurityGroupId: sg-XXXX
   InternetAccess: true
  function:
   Type: 'Aliyun::Serverless::Function'
   Properties:
    Handler: 'index.handler'
    Runtime: python3
    Timeout: 10
    MemorySize: 128
    CodeUri: '. /'
```

The following list describes the main parameters:

- A service named service is declared.
  - Policies: grants Function Compute permission to manage elastic network interfaces (ENIs) attached to Elastic Compute Service (ECS) instances. Function Compute can then access resources in the VPC.
  - VpcConfig: binds the VPC to the service. You must replace the following values with the information for your VPC.
    - VpcId: the ID of the VPC.
    - VSwitchId: the ID of the VSwitch.
    - SecurityGroupId: the ID of the security group.
- A service named service is declared.
  - Initializer: the initializer function. For more information, see the "Initializer function" section of the Terms topic.
  - Handler: the function handler. For more information, see Function entry point.
  - Runtime: the runtime environment of the function.
  - CodeUri: the directory where the code package is located.
    When you deploy the function, Funcraft packages and uploads the items in the directory that the CodeUri parameter specifies.

2. In the directory that contains the *template.yml* file, create the *Funfile* file. In this example, the new file contains the following sample content:

```
RUNTIME python3
RUN fun-install pip install pymongo
```

3. Run the **fun install** command to install dependencies.

```
fun install
```

When the code editor shows the following content, the dependencies have been installed.

```
Install Success
```

4. In the directory that contains the *template.yml* file, create a Python file. In this example, the */tmp/code/index.py* file is created and contains the following sample content:

```
  # -*- coding: utf-8 -*-
import uuid
from pymongo import MongoClient
def handler(event, context):
    CONN_ADDR1 = 'dds-XXX.mongodb.rds.aliyuncs.com:3717'
    CONN_ADDR2 = 'dds-XXXX.mongodb.rds.aliyuncs.com:3717'
    REPLICAT_SET = 'XXX'
    username = 'XXX'
    password = 'XXXX'
    # Obtain the MongoClient.
    client = MongoClient([CONN_ADDR1, CONN_ADDR2], replicaSet=REPLICAT_SET)
    # Grant the user the administrator permissions to access the database.
    client.admin.authenticate(username, password)
    # Insert doc and search for documents based on the demo name. The collection:testColl of the test
database is used in this example.
    demo_name = 'python-' + str(uuid.uuid1())
    print ('demo_name:'+ demo_name)
    doc = dict(DEMO=demo_name, MESG="Hello ApsaraDB For MongoDB")
    doc_id = client.test.testColl.insert(doc)
    for d in client.test.testColl.find(dict(DEMO=demo_name)):
      print ('find documents:'+ str(d))
    return 'success'
```

5. Run the following command to deploy the function to Function Compute by using Funcraft:

```
fun deploy
```

When the code editor shows the following content, the function has been deployed.

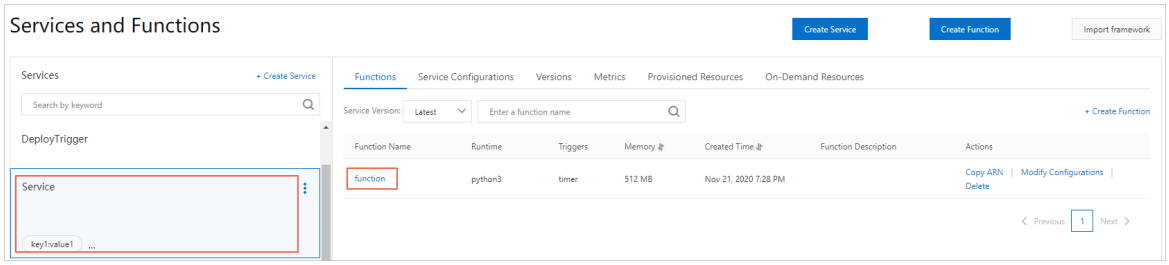
```
    function <function-name> deploy success
service <service-name> deploy success
```

After the deployment is complete, log on to the Failed to resolve content from t1882558.dita#task_2470088/xref_vds_2o7_t33 and click **Service/Function**. On the page that appears, you can see the newly deployed service and function.

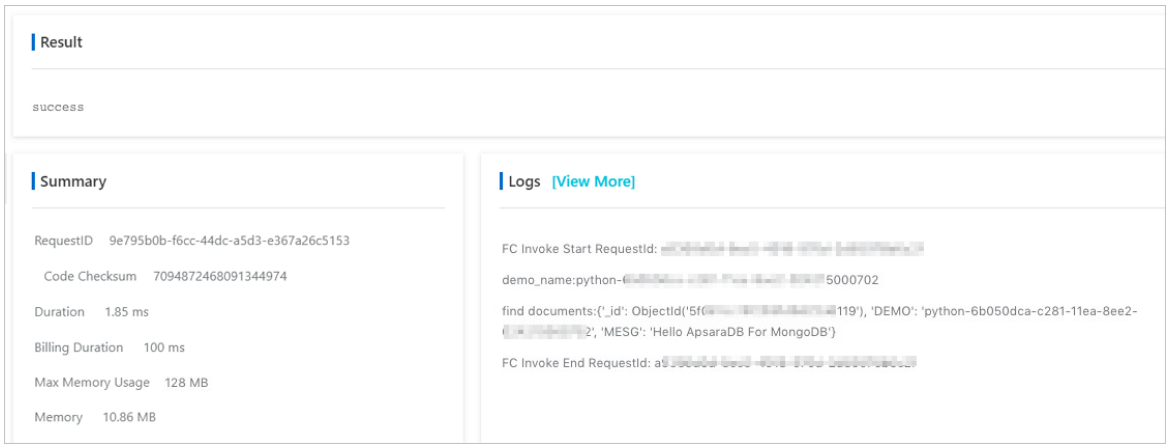
Debug the function

After the function is deployed, you can debug it in the Function Compute console.

1. Log on to the Function Compute console.
2. In the top navigation bar, select a region.
3. In the left-side navigation pane, click **Services and Functions**.
4. In the **Services** pane, click the service that you require. On the **Functions** tab, click the name of the function that you require.



5. On the page that appears, click the **Code** tab. On the Code tab, click **Invoke**. After the execution is complete, you can view the execution results and logs.



10.5. Access an ApsaraDB for Redis database

In Function Compute, you can use a function to call the ApsaraDB for Redis API to perform operations such as insert and query on an ApsaraDB for Redis database. In normal cases, states cannot be shared among different execution environments in Function Compute. You can persist structured data in a database so that states can be shared. This topic describes how to use Funcraft to deploy a function to access an ApsaraDB for Redis database.

Prerequisites

1. You have created a virtual private cloud (VPC) and a VSwitch.
   The VSwitch must belong to a zone supported by Function Compute. For more information about the zones supported by Function Compute, see VPC access.
2. An ApsaraDB for Redis instance is created.
   You have created a database instance in same VPC as Function Compute. You do not need to create the VSwitch in the same zone as Function Compute. VSwitches under the same VPC can communicate with each other even when they are in different zones.
3. A whitelist is configured for the ApsaraDB for Redis instance.
   If you need to control access to databases by using an IP address whitelist, ensure that you have entered the internal Classless Inter-Domain Routing (CIDR) blocks of the target VPCs in the **IP** field. To find the CIDR block of a VPC, log on to the VPC console and view the **VPC Details** page.
4. Create a security group
   You have configured the security group to allow outbound traffic from the internal CIDR block and port of the database.

Write code for a function

The following section describes how to use Funcraft to write a function that accesses a database. The sample code is written in Python 3.

1. On the local machine, create a directory to store code and dependent modules. In the directory, create the *template.yml* file. In this example, the */tmp/code/template.yml* file is created and contains the following content:

```
ROSTemplateFormatVersion: '2015-09-01'
Transform: 'Aliyun::Serverless-2018-04-03'
Resources:
 service:
  Type: 'Aliyun::Serverless::Service'
  Properties:
   Description: This is Redis service
   Policies:
    - AliyunECSNetworkInterfaceManagementAccess
   VpcConfig:
    VpcId: vpc-xxxx
    VSwitchIds:
     - vsw-xxxx
    SecurityGroupId: sg-xxxx
   InternetAccess: true
  function:
   Type: 'Aliyun::Serverless::Function'
   Properties:
    Initializer: 'index.initializer'
    Handler: 'index.handler'
    Runtime: python3
    CodeUri: '. /'
    EnvironmentVariables:
     REDIS_HOST: r-xxxx.redis.rds.aliyuncs.com
     REDIS_PASSWORD: Txd1xxxx
     REDIS_PORT: '6379'
```

   The following list describes the main parameters:

   - A service named service is declared.
     - Policies: grants Function Compute permission to manage elastic network interfaces (ENIs) attached to Elastic Compute Service (ECS) instances. Function Compute can then access resources in the VPC.
     - VpcConfig: binds the VPC to the service. You must replace the following values with the information for your VPC.
       - VpcId: the ID of the VPC.
       - VSwitchId: the ID of the VSwitch.
       - SecurityGroupId: the ID of the security group.
   - A function named function is declared.
     - Initializer: the initializer function. For more information, see Initializer function.
     - Handler: the function handler. For more information, see Function entry point.
     - Runtime: the runtime environment of the function.

- CodeUri: the directory where the code package is located.
  During function deployment, Funcraft packages and uploads the items in the directory specified by the CodeUri parameter.
- EnvironmentVariables: the environment variables of the function.
  - REDIS_HOST: the internal endpoint of the ApsaraDB for Redis instance.
    To find the internal endpoint, log on to the ApsaraDB for Redis console and open the **Instance Information** page for the target instance. In the **Connection Information** section, check the value of **Internal Endpoint (Host)**.
  - REDIS_PASSWORD: the password for logging on to the ApsaraDB for Redis database.
  - REDIS_PORT: the port used to connect to the ApsaraDB for Redis database.

For more information about configuration rules, see Serverless Application Model.

2. In the directory that contains the *template.yml* file, create the *Funfile* file. In this example, the new file contains the following sample content:

```
RUNTIME python3
RUN fun-install pip install redis
```

3. Run the **fun install** command to install dependencies.

```
fun install
```

```
using template: template.yml
start installing function dependencies without docker
building service/function
Funfile exist, Fun will use container to build forcely
Step 1/2 : FROM registry.cn-beijing.aliyuncs.com/aliyunfc/runtime-python3.6:build-1.7.7
---> 373f5819463b
Step 2/2 : RUN fun-install pip install redis
---> Running in f26aef48f9e5
Task => PipTask
   => PYTHONUSERBASE=/code/.fun/python pip install --user redis
Removing intermediate container f26aef48f9e5
---> 809c6655f9e9
sha256:809c6655f9e93d137840b1446f46572fbab7548c5c36b6ae66599dfc2e27555b
Successfully built 809c6655f9e9
Successfully tagged fun-cache-78c74899-5497-4205-a670-24e4daf88284:latest
copying function artifact to /Users/txd123/Desktop/Redis/Python
Install Success
```

4. In the directory that contains the *template.yml* file, create a Python file. In this example, the */tmp/code/index.py* file is created and contains the following sample content:

```
# -*- coding: utf-8 -*-
import os,sys
import redis
def initializer(context):
  global conn_pool
  conn_pool=redis.ConnectionPool(host=os.environ['REDIS_HOST'],password=os.environ['REDIS_PAS
SWORD'],port=os.environ['REDIS_PORT'],db=1,decode_responses=True)
def handler(event, context):
  r = redis.Redis(connection_pool=conn_pool)
  r.set('test','89898')
  r.set('zyh_info','{"name":"Tanya","password":"123456","account":11234}')
  print(r.get('test'))
  return r.get('zyh_info')
```

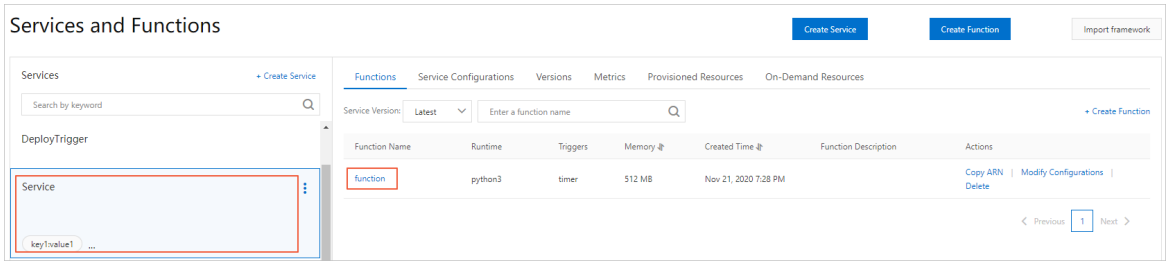
5. Run the following command to deploy the function to Function Compute by using Funcraft:

```
fun deploy
```

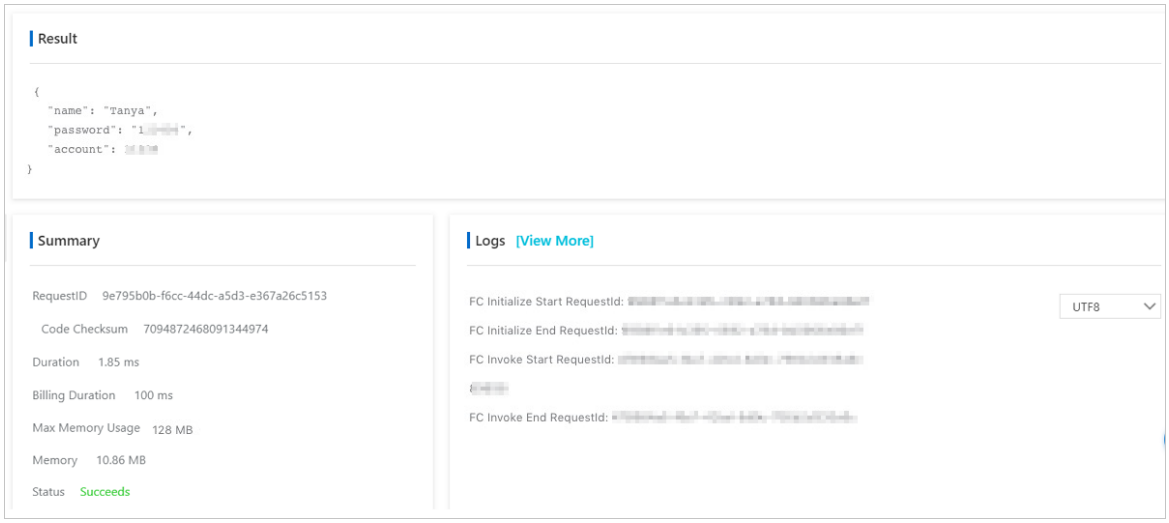
```
using template: template.yml
using region: cn-hangzhou
using accountId: ***********3743
using accessKeyId: ***********Ptgk
using timeout: 60
Waiting for service service to be deployed...
    Waiting for function function to be deployed...
        Waiting for packaging function function code...
        The function function has been packaged. A total of 25 files files were compressed and the final s
ize was 138.78 KB
    function function deploy success
service service deploy success
```

Debug the function

1. Log on to the Function Compute console.
2. In the top navigation bar, select a region.
3. In the left-side navigation pane, click **Services and Functions**.
4. In the **Services** pane, click the service that you require. On the **Functions** tab, click the name of the function that you require.



5. On the page that appears, click the **Code** tab. On the Code tab, click **Invoke**.
After the execution is complete, you can view the execution results and logs.



10.6. Access an ApsaraDB RDS for SQL Server database

In Function Compute, you can use a function to call the ApsaraDB RDS for SQL Server API to perform operations such as insert and query on an ApsaraDB RDS for SQL Server database. In normal cases, states cannot be shared among different execution environments in Function Compute. You can persist structured data in a database so that states can be shared. This topic describes how to use Funcraft to deploy a function to access an ApsaraDB RDS for SQL Server database.

Prerequisites

1. You have created a virtual private cloud (VPC) and a VSwitch.
   The VSwitch must belong to a zone supported by Function Compute. For more information about the zones supported by Function Compute, see VPC access.
2. Create an ApsaraDB RDS for SQL Server instance
   You have created a database instance in same VPC as Function Compute. You do not need to create the VSwitch in the same zone as Function Compute. VSwitches under the same VPC can communicate with each other even when they are in different zones.
3. Configure an IP address whitelist for an ApsaraDB RDS for SQL Server instance
   If you need to control access to databases by using an IP address whitelist, ensure that you have entered the internal Classless Inter-Domain Routing (CIDR) blocks of the target VPCs in the **IP** field. To find the CIDR block of a VPC, log on to the VPC console and view the **VPC Details** page.
4. Create a security group
   You have configured the security group to allow outbound traffic from the internal CIDR block and port of the database.

Write code for a function in Python 3

This section describes how to use Funcraft to write code for a function in Python 3 to access an ApsaraDB RDS for SQL Server database.

1. On the local machine, create a directory to store code and dependent modules. In the directory, create the *template.yml* file. In this example, the */tmp/code/template.yml* file is created and contains the following content:

```
ROSTemplateFormatVersion: '2015-09-01'
Transform: 'Aliyun::Serverless-2018-04-03'
Resources:
 service:
  Type: 'Aliyun::Serverless::Service'
  Properties:
   Description: This is SQL-Server service
   Policies:
    - AliyunECSNetworkInterfaceManagementAccess
   VpcConfig:
    VpcId: vpc-xxx
    VSwitchIds:
     - vsw-xxx
    SecurityGroupId: sg-xxx
   InternetAccess: true
  function:
   Type: 'Aliyun::Serverless::Function'
   Properties:
    Handler: 'index.handler'
    Runtime: python3
    Timeout: 10
    MemorySize: 128
    CodeUri: '. /'
```

The following list describes the main parameters:

- A service named service is declared.
  - Policies: grants Function Compute permission to manage elastic network interfaces (ENIs) attached to Elastic Compute Service (ECS) instances. Function Compute can then access resources in the VPC.
  - VpcConfig: binds the VPC to the service. You must replace the following values with the information for your VPC.
    - VpcId: the ID of the VPC.
    - VSwitchId: the ID of the VSwitch.
    - SecurityGroupId: the ID of the security group.
- A function named function is declared.
  - Handler: the function handler. For more information, see Function entry point.
  - Runtime: the runtime environment of the function.
  - CodeUri: the directory where the code package is located.
    When you deploy the function, Funcraft packages and uploads the items in the directory that the CodeUri parameter specifies.

2. In the directory that contains the *template.yml* file, create the *Funfile* file. In this example, the new file contains the following sample content:

```
RUNTIME python3
RUN fun-install pip install pymssql
```

3. Run the **fun install** command to install dependencies.

```
fun install
```

When the code editor shows the following content, the dependencies have been installed.

```
Install Success
```

4. In the directory that contains the *template.yml* file, create a Python file. In this example, the */tmp/code/index.py* file is created and contains the following sample content:

```
# -*- coding: utf-8 -*-
import pymssql
def handler(event, context):
  conn = pymssql.connect(host='rm-xxx.sqlserver.rds.aliyuncs.com',
                         user='xxx',
                         password='x'x'x'x',
                         database='xxx',
                         charset='utf8')
  cursor = conn.cursor()
  cursor.execute('SELECT * FROM inventory WHERE quantity > 152')
  result = ''
  for row in cursor:
    result += 'row = %r\n' % (row,)
  conn.close()
  return result
```

5. Run the following command to deploy the function to Function Compute by using Funcraft:

```
fun deploy
```

When the code editor shows the following content, the function has been deployed.

```
    function <function-name> deploy success
service <service-name> deploy success
```

After the deployment is complete, log on to the Failed to resolve content from t1882558.dita#task_2470088/xref_vds_2o7_t33 and click **Service/Function**. On the page that appears, you can see the newly deployed service and function.

Write code for a function in PHP 7.2

This section describes how to use Funcraft to write code for a function in PHP 7.2 to access an ApsaraDB RDS for SQL Server database.

1. On the local machine, create a directory to store code and dependent modules. In the directory, create the *template.yml* file. In this example, the */tmp/code/template.yml* file is created and contains the following content:

```
ROSTemplateFormatVersion: '2015-09-01'
Transform: 'Aliyun::Serverless-2018-04-03'
Resources:
  service:
    Type: 'Aliyun::Serverless::Service'
    Properties:
      Description: This is SQL-Server service
      Policies:
        - AliyunECSNetworkInterfaceManagementAccess
      VpcConfig:
        VpcId: vpc-xxx
        VSwitchIds:
          - vsw-x'x'x'x
        SecurityGroupId: sg-xxx
      InternetAccess: true
    function:
      Type: 'Aliyun::Serverless::Function'
      Properties:
        Handler: 'index.handler'
        Runtime: php7.2
        Timeout: 10
        MemorySize: 128
        CodeUri: '. /'
        EnvironmentVariables:
          ODBCINI: /code/.fun/root/etc/odbc.ini
          ODBCSYSINI: /code/.fun/root/opt/microsoft/msodbcsql17/etc
```

The following list describes the main parameters:

- A service named service is declared.
  - Policies: grants Function Compute permission to manage elastic network interfaces (ENIs) attached to Elastic Compute Service (ECS) instances. Function Compute can then access resources in the VPC.
  - VpcConfig: binds the VPC to the service. You must replace the following values with the information for your VPC.
    - VpcId: the ID of the VPC.
    - VSwitchId: the ID of the VSwitch.
    - SecurityGroupId: the ID of the security group.
- A function named function is declared.
  - Handler: the function handler. For more information, see Function entry point.
  - Runtime: the runtime environment of the function.
  - CodeUri: the directory where the code package is located.
    When you deploy the function, Funcraft packages and uploads the items in the directory that the CodeUri parameter specifies.
  - EnvironmentVariables: the environment variables of the function.
    - ODBCINI: the directory where the *odbc.ini* file to be uploaded is located.
    - ODBCSYSINI: the directory where the *odbcinst.ini* file to be uploaded is located.

2. In the directory that contains the *template.yml* file, create the *Funfile* file. In this example, the new

file contains the following sample content:

```
RUNTIME php7.2
RUN apt-get update && apt-get install -y apt-transport-https apt-utils
RUN curl https://packages.microsoft.com/keys/microsoft.asc | apt-key add -
RUN curl https://packages.microsoft.com/config/debian/8/prod.list > /etc/apt/sources.list.d/mssql-release.list
RUN fun-install apt-get install unixodbc-dev
RUN fun-install apt-get install msodbcsql17
```

3. Run the **fun install** command to install dependencies.

```
$fun install
```

When the code editor shows the following content, the dependencies have been installed.

```
Install Success
```

4. Modify the *odbc.ini* file in the root directory, such as */tmp/code/odbc.ini*. Modify the value of the Driver parameter in the file to /code/.fun/root/opt/microsoft/msodbcsql17/lib64/libmsodbcsql-17.5.so.1.1.

> **Note** If you need to use pdo_sqlsrv to extend the code in the PHP environment, see Non-built-in extension for php runtime compilation.

5. Create a file in a directory at the same level as that of the *template.yml* file, such as */tmp/code/index.php*. The following sample code is provided:

```
<? php
function handler($event, $context)
{
  try {
    $conn = new PDO("sqlsrv:Server=rm-xxx.sqlserver.rds.aliyuncs.com;Database=xxx","xxx","xxx");
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $conn->query("set names utf-8");
    $sql="SELECT * FROM inventory WHERE quantity > 152";
    $result = $conn->prepare($sql);
    $result->execute();
    print($result);
    return ("Connection successed.");
  } catch (PDOException $e) {
    return ("Connection failed: " . $e->getMessage());
  }
}
```

6. Run the following command to deploy the function to Function Compute by using Funcraft:

```
fun deploy
```

When the code editor shows the following content, the function has been deployed.

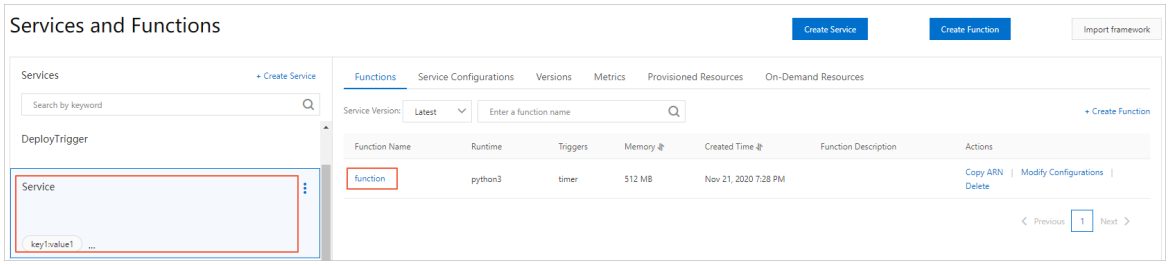
```
    function <function-name> deploy success
service <service-name> deploy success
```

After the deployment is complete, log on to the Failed to resolve content from t1882558.dita#task_2470088/xref_vds_2o7_t33 and click **Service/Function**. On the page that appears, you can see the newly deployed service and function.

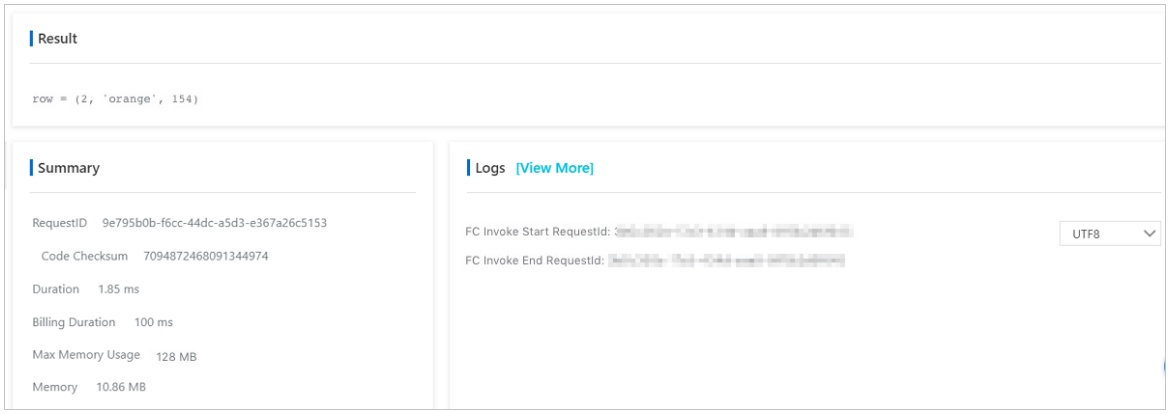
Debug the function

After the function is deployed, you can debug it in the Function Compute console.

1. Log on to the Function Compute console.
2. In the top navigation bar, select a region.
3. In the left-side navigation pane, click **Services and Functions**.
4. In the **Services** pane, click the service that you require. On the **Functions** tab, click the name of the function that you require.



5. On the page that appears, click the **Code** tab. On the Code tab, click **Invoke**. After the execution is complete, you can view the execution results and logs.



10.7. Access an ApsaraDB RDS for PostgreSQL database

In Function Compute, you can use a function to call the ApsaraDB RDS for PostgreSQL API to perform operations such as insert and query on an ApsaraDB RDS for PostgreSQL database. In normal cases, states cannot be shared among different execution environments in Function Compute. You can persist structured data in a database so that states can be shared. This topic describes how to use Funcraft to deploy a function to access an ApsaraDB RDS for PostgreSQL database.

Prerequisites

1. You have created a virtual private cloud (VPC) and a VSwitch.
   The VSwitch must belong to a zone supported by Function Compute. For more information about the zones supported by Function Compute, see VPC access.
2. Create an ApsaraDB RDS for PostgreSQL instance

You have created a database instance in same VPC as Function Compute. You do not need to create the VSwitch in the same zone as Function Compute. VSwitches under the same VPC can communicate with each other even when they are in different zones.

3. Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance
   If you need to control access to databases by using an IP address whitelist, ensure that you have entered the internal Classless Inter-Domain Routing (CIDR) blocks of the target VPCs in the **IP** field. To find the CIDR block of a VPC, log on to the VPC console and view the **VPC Details** page.
4. Create a security group
   You have configured the security group to allow outbound traffic from the internal CIDR block and port of the database.

Write code for a function

The following section describes how to use Funcraft to write a function that accesses a database. The sample code is written in Python 3.

1. On the local machine, create a directory to store code and dependent modules. In the directory, create the *template.yml* file. In this example, the */tmp/code/template.yml* file is created and contains the following content:

```
ROSTemplateFormatVersion: '2015-09-01'
Transform: 'Aliyun::Serverless-2018-04-03'
Resources:
 service:
  Type: 'Aliyun::Serverless::Service'
  Properties:
   Description: This is PostgreSQL service
   Policies:
    - AliyunECSNetworkInterfaceManagementAccess
   VpcConfig:
    VpcId: vpc-****
    VSwitchIds:
     - vsw-***
    SecurityGroupId: sg-***
   InternetAccess: true
  function:
   Type: 'Aliyun::Serverless::Function'
   Properties:
    Handler: 'index.handler'
    Initializer: 'index.initializer'
    Runtime: python3
    Timeout: 10
    MemorySize: 128
    CodeUri: '. /'
    EnvironmentVariables:
     HOST: pgm-bp1yawvyyu***.pg.rds.aliyuncs.com
     PASSWORD: Txd123**
     PORT: 1433
     DATABASE: test_123
     USER: ***
```

   The following list describes the main parameters:

   - A service named service is declared.

- Policies: grants Function Compute permission to manage elastic network interfaces (ENIs) attached to Elastic Compute Service (ECS) instances. Function Compute can then access resources in the VPC.
- VpcConfig: binds the VPC to the service. You must replace the following values with the information for your VPC.
  - VpcId: the ID of the VPC.
  - VSwitchId: the ID of the VSwitch.
  - SecurityGroupId: the ID of the security group.

- A function named function is declared.
  - Initializer: the initializer function. For more information, see Initializer function.
  - Handler: the function handler. For more information, see Function entry point.
  - Runtime: the runtime environment of the function.
  - CodeUri: the directory where the code package is located.
    During function deployment, Funcraft packages and uploads the items in the directory specified by the CodeUri parameter.
  - EnvironmentVariables: the environment variables of the function.
    - HOST: the internal endpoint of the ApsaraDB RDS for PostgreSQL instance.
      Log on to the ApsaraDB RDS console. Click the instance that you require to go to the **Basic Information** page. In the **Basic Information** section, click **See Detail** to view the internal endpoint.
    - PASSWORD: the password used to log on to the database.
    - PORT: the port of the database.
    - DATABASE: the name of the database.
    - USER: the username used to log on to the database.

2. In the directory that contains the *template.yml* file, create the *Funfile* file. In this example, the new file contains the following sample content:

```
RUNTIME python3
RUN fun-install pip install psycopg2
```

3. Run the **fun install** command to install dependencies.

```
fun install
```

When the code editor shows the following content, the dependencies have been installed.

```
Install Success
```

4. In the directory that contains the *template.yml* file, create a Python file. In this example, the */tmp/code/index.py* file is created and contains the following sample content:

```
# -*- coding: utf-8 -*-
import logging
import psycopg2
import os,sys
logger = logging.getLogger()
def getConnection():
 try:
  conn = psycopg2.connect(
   database = os.environ['DATABASE'],
   user = os.environ['USER'],
   password = os.environ['PASSWORD'],
   host = os.environ['HOST'],
   port = os.environ['PORT'],
   )
  return conn
 except Exception as e:
  logger.error(e)
  logger.error("ERROR: Unexpected error: Could not connect to PostgreSQL instance.")
  sys.exit()
def conditionallyCreateUsersTable():
  conn = getConnection()
  cur = conn.cursor()
  cur.execute('''CREATE TABLE COMPANY
    (ID INT PRIMARY KEY    NOT NULL,
    NAME        TEXT    NOT NULL,
    AGE         INT     NOT NULL,
    ADDRESS     CHAR(50),
    SALARY      REAL);''')
  conn.commit()
  conn.close()
def initializer(context):
 conditionallyCreateUsersTable()
def handler(event, context):
 try:
  conn = getConnection()
  cur = conn.cursor()
  cur.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
   VALUES (1, 'Paul', 32, 'California', 20000.00 )");
  conn.commit()
  return 'successfully'
 finally:
  conn.close()
```

5. Run the following command to deploy the function to Function Compute by using Funcraft:

```
fun deploy -y
```

When the code editor shows the following content, the function has been deployed.

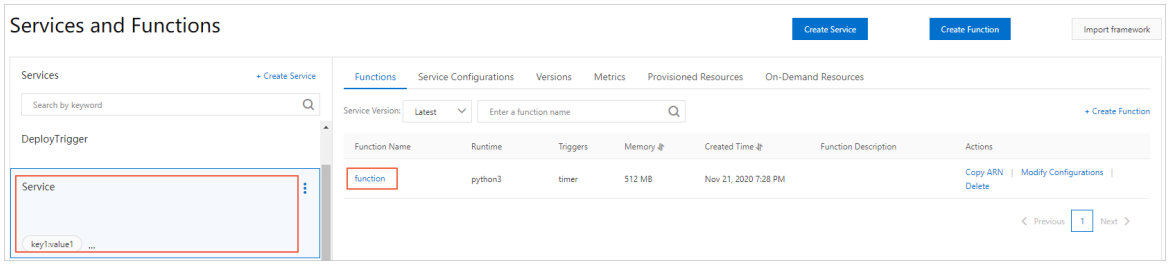
```
    function <function-name> deploy success
service <service-name> deploy success
```

After the deployment is complete, log on to the Failed to resolve content from t1882558.dita#task_2470088/xref_vds_2o7_t33 and click **Service/Function**. On the page that appears, you can see the newly deployed service and function.

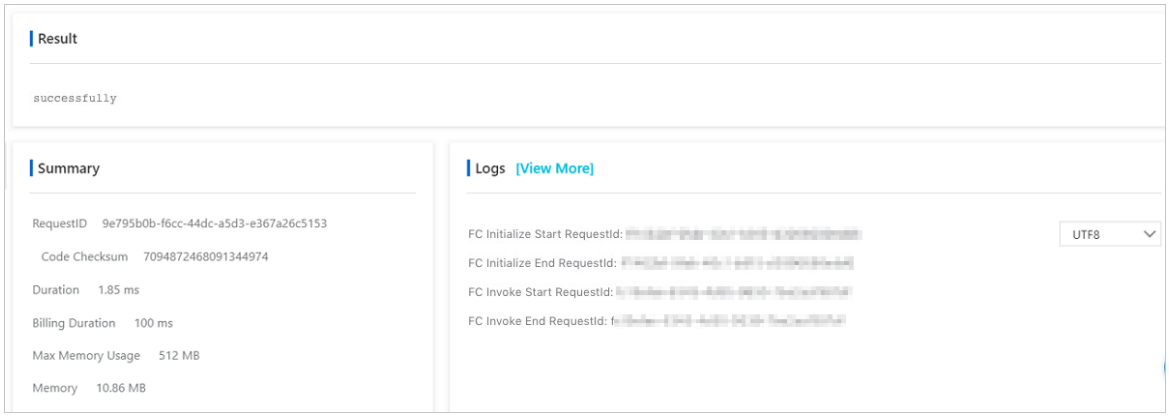
Debug the function

After the function is deployed, you can debug it in the Function Compute console.

1. Log on to the Function Compute console.
2. In the top navigation bar, select a region.
3. In the left-side navigation pane, click **Services and Functions**.
4. In the **Services** pane, click the service that you require. On the **Functions** tab, click the name of the function that you require.



5. On the page that appears, click the **Code** tab. On the Code tab, click **Invoke**. After the execution is complete, you can view the execution results and logs.



10.8. Access an ApsaraDB for Lindorm database

In Function Compute, you can use a function to call the ApsaraDB for Lindorm API to perform operations such as insert and query on a Lindorm database. In normal cases, states cannot be shared among different execution environments in Function Compute. You can persist structured data in a database so that states can be shared. This topic describes how to use Funcraft to deploy a function to access a Lindorm database.

Prerequisites

1. You have created a virtual private cloud (VPC) and a VSwitch.
   The VSwitch must belong to a zone supported by Function Compute. For more information about the zones supported by Function Compute, see VPC access.
2. A Lindorm instance that meets your business needs is created.

You have created a database instance in same VPC as Function Compute. You do not need to create the VSwitch in the same zone as Function Compute. VSwitches under the same VPC can communicate with each other even when they are in different zones.

3. Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance
   If you need to control access to databases by using an IP address whitelist, ensure that you have entered the internal Classless Inter-Domain Routing (CIDR) blocks of the target VPCs in the **IP** field. To find the CIDR block of a VPC, log on to the VPC console and view the **VPC Details** page.
4. Create a security group
   You have configured the security group to allow outbound traffic from the internal CIDR block and port of the database.

Write code for a function

The following section describes how to use Funcraft to write a function that accesses a database. The sample code is written in Python 3.

1. On the local machine, create a directory to store code and dependent modules. In the directory, create the *template.yml* file. In this example, the */tmp/code/template.yml* file is created and contains the following content:

```
ROSTemplateFormatVersion: '2015-09-01'
Transform: 'Aliyun::Serverless-2018-04-03'
Resources:
 service:
  Type: 'Aliyun::Serverless::Service'
  Properties:
   Description: This is Lindorm service
   Policies:
    - AliyunECSNetworkInterfaceManagementAccess
   VpcConfig:
    VpcId: vpc-XXXX
    VSwitchIds:
     - vsw-XXX
    SecurityGroupId: sg-XXXX
   InternetAccess: true
  function:
   Type: 'Aliyun::Serverless::Function'
   Properties:
    Handler: 'index.handler'
    Runtime: python3
    Timeout: 100
    MemorySize: 1024
    CodeUri: '. /'
```

   The following list describes the main parameters:

   - A service named service is declared.
     - Policies: grants Function Compute permission to manage elastic network interfaces (ENIs) attached to Elastic Compute Service (ECS) instances. Function Compute can then access resources in the VPC.

- VpcConfig: binds the VPC to the service. You must replace the following values with the information for your VPC.
  - VpcId: the ID of the VPC.
  - VSwitchId: the ID of the VSwitch.
  - SecurityGroupId: the ID of the security group.
- A function named function is declared.
  - Initializer: the initializer function. For more information, see Initializer function.
  - Handler: the function handler. For more information, see Function entry point.
  - Runtime: the runtime environment of the function.
  - CodeUri: the directory where the code package is located.
    During function deployment, Funcraft packages and uploads the items in the directory specified by the CodeUri parameter.

2. In the directory that contains the *template.yml* file, create the *Funfile* file. In this example, the new file contains the following sample content:

```
RUNTIME python3
RUN fun-install pip install cassandra-driver
```

3. Run the **fun install** command to install dependencies.

```
fun install
```

When the code editor shows the following content, the dependencies have been installed.

```
Install Success
```

4. In the directory that contains the *template.yml* file, create a Python file. In this example, the */tmp/code/index.py* file is created and contains the following sample content:

> **Note** You can log on to the Lindorm console, find the database, and obtain the endpoint, username, and password on the **Wide Table Engine** tab of the **Database Connection** page.

```
# -*- coding: utf-8 -*-
import logging
import sys
from cassandra.cluster import Cluster
from cassandra.auth import PlainTextAuthProvider
logger = logging.getLogger()
def handler(event, context):
  logger.info("start to test Lindorm ")
  cluster = Cluster(
    # Configure the endpoint.
    contact_points=["ld-bp10ljb1mxae5****-proxy-lindorm.lindorm.rds.aliyuncs.com"],
    # Configure the username and password.
    auth_provider=PlainTextAuthProvider("XXX", "XXX"))
  session = cluster.connect()
  session.execute("CREATE KEYSPACE IF NOT EXISTS testKeyspace WITH replication = {'class':'SimpleStrategy', 'replication_factor':1};");
  session.execute("CREATE TABLE IF NOT EXISTS testKeyspace.testTable (id int PRIMARY KEY, name text,age int,address text);");
  session.execute("INSERT INTO testKeyspace.testTable (id, name, age, address) VALUES ( 1, 'testname', 11, 'hangzhou');");
  rows = session.execute( "SELECT * FROM testKeyspace.testTable ;");
  for row in rows:
    logger.info("# row: {}".format(row))
  session.shutdown()
  cluster.shutdown()
```

5. Run the following command to deploy the function to Function Compute by using Funcraft:

   ```
   fun deploy
   ```

   When the code editor shows the following content, the function has been deployed.

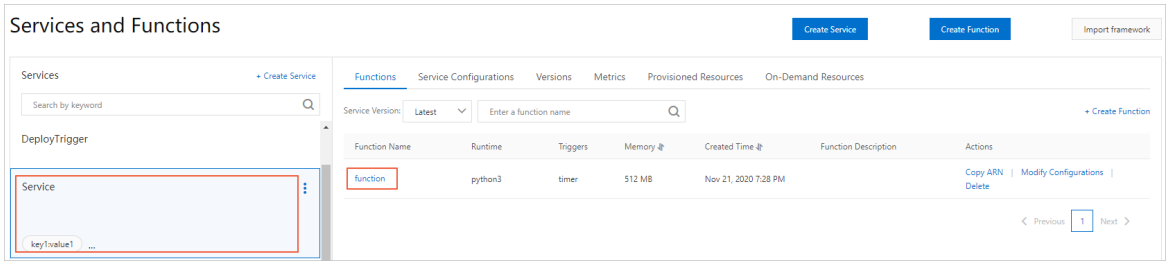
   ```
       function <function-name> deploy success
   service <service-name> deploy success
   ```

   After the deployment is complete, log on to the Failed to resolve content from t1882558.dita#task_2470088/xref_vds_2o7_t33 and click **Service/Function**. On the page that appears, you can see the newly deployed service and function.

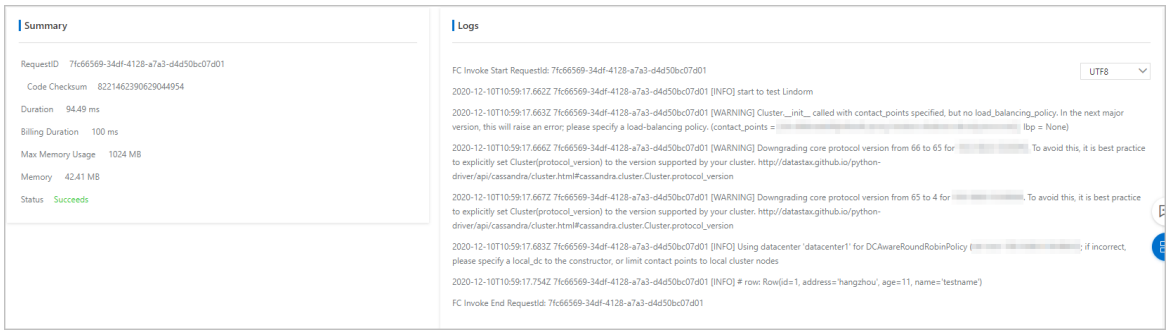
Debug the function

After the function is deployed, you can debug it in the Function Compute console.

1. Log on to the Function Compute console.
2. In the top navigation bar, select a region.
3. In the left-side navigation pane, click **Services and Functions**. In the **Services** pane, click the service that you require.
4. On the **Functions** tab, click the name of the function that you require.



5. On the page that appears, click the **Code** tab. On the Code tab, click **Invoke**. After the execution is complete, you can view the execution results and logs.



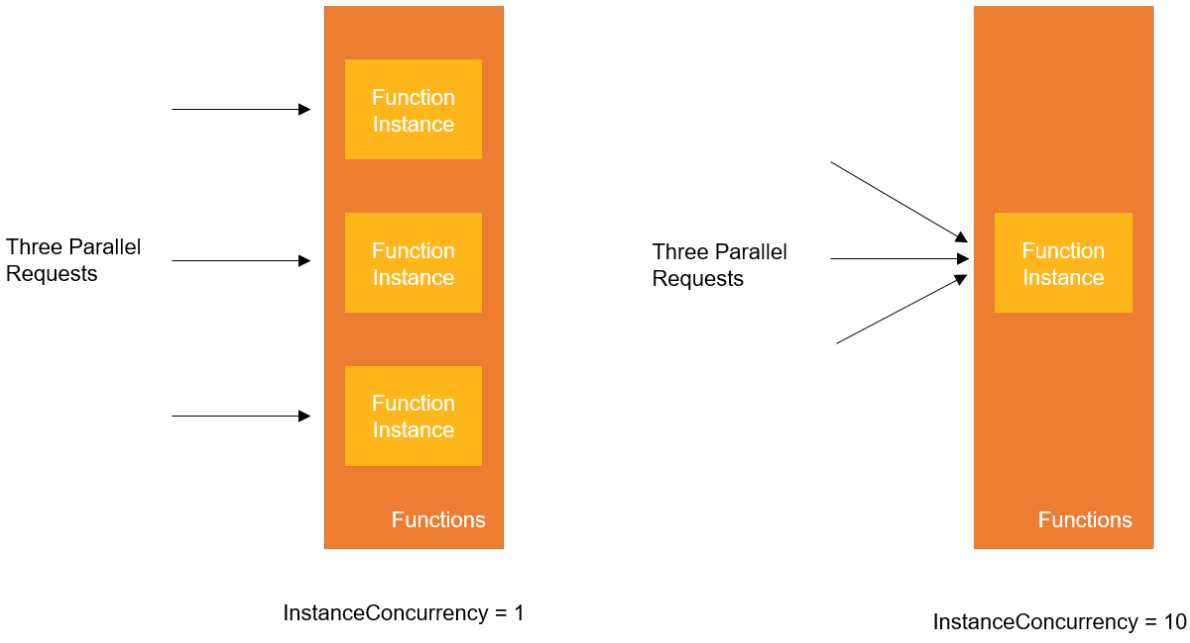
11.Instance concurrency management

11.1. A single instance that concurrently processes multiple requests

This topic describes the background information, benefits, scenarios, and impact of a single instance that concurrently processes multiple requests.

Background information

Function Compute calculates fees based on the execution duration of requests by an instance. Assume that database access latency is 10 seconds and three instances totally process three requests. The total execution duration of the requests by the three instances is 30 seconds. If one instance concurrently processes the three requests, the total execution duration of the requests by the instance is 10 seconds. Function Compute allows you to concurrently process multiple requests by using a single instance. This reduces the fees for resource usage of instances. Function Compute allows you to set the Single Instance Concurrency parameter for a function. This parameter specifies the maximum number of requests that a single instance can concurrently process. The following figure shows the differences between a single instance that processes a single request at a time and a single instance that concurrently processes multiple requests.



Assume that three requests need to be concurrently processed. If you set the Single Instance Concurrency parameter to 1, Function Compute must create three instances to process the three requests, and each instance processes one request. If you set the Single Instance Concurrency parameter to 10, one instance can concurrently process 10 requests, and Function Compute needs to create only one instance to process the three requests.

> **Note** By default, the value of the Single Instance Concurrency parameter is 1. A value of 1 indicates that a single instance can process only one request at a time. If you set the Single Instance Concurrency parameter to a value greater than 1, Function Compute makes full use of the concurrency of an instance before Function Compute creates another instance.

Benefits

- Reduces the execution duration and costs.
  For example, for functions that require I/O operations, you can use a single instance to concurrently process multiple requests. This reduces the number of instances that are used to process requests to reduce the total execution duration of requests by the instances.
- Provides a shared state for requests.
  Multiple requests can share the connection pool of a database in one instance to minimize the connections between requests and the database.
- Reduces the frequency of cold starts.
  Fewer instances need to be created because one instance can process multiple requests. This reduces the frequency of cold starts.
- Reduces the number of IP addresses used in a virtual private cloud (VPC).
  For a fixed number of requests to be processed, the number of occupied instances is reduced when each instance can process multiple requests. This reduces the number of IP addresses used in the VPC.

Scenarios

This feature is not applicable to all functions. The following table describes the scenarios to which this feature is or is not applicable.

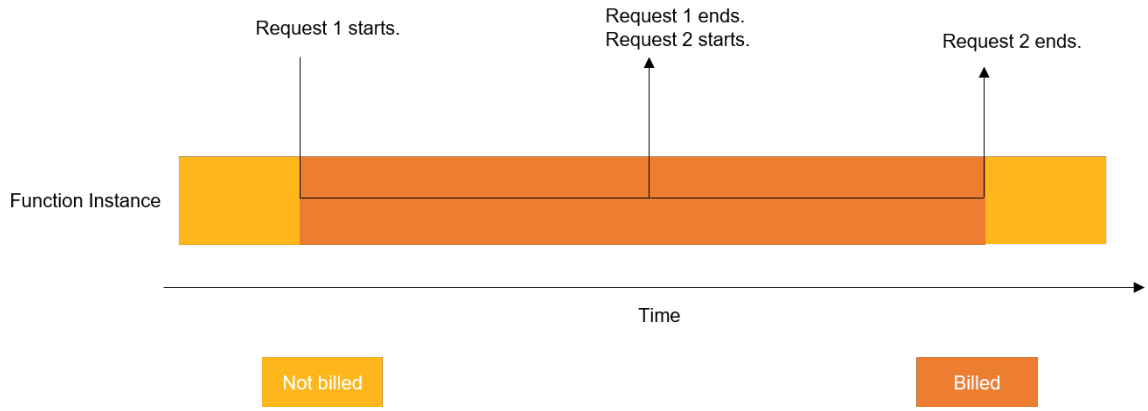
Scenario	Applicable	Reason
Requests are waiting for responses from the downstream service for an extended period of time.	Yes	Resources are generally not consumed when requests are waiting for responses. If you use a single instance to concurrently process multiple requests, costs can be reduced.
Requests are using a shared state that cannot be concurrently accessed.	No	If multiple requests are concurrently processed to modify the shared state, such as global variables, errors may occur.
A request consumes a large amount of CPU and memory resources.	No	Multiple requests compete for resources, which leads to out of memory or longer latency.

Impacts

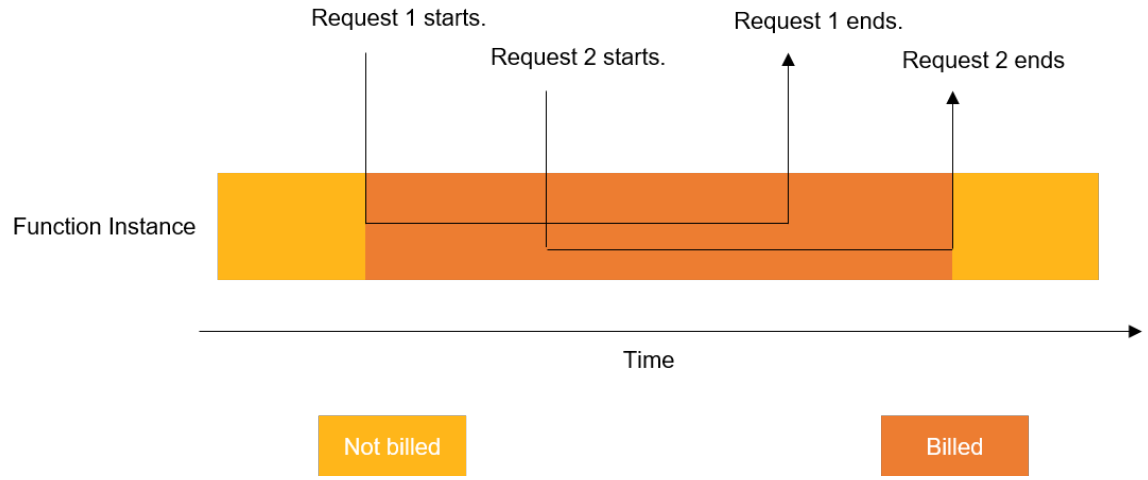
If you set the Single Instance Concurrency parameter to a value greater than 1, the differences from a value of 1 lie in the following aspects:

- Billing

- A single instance that processes a single request at a time
  An instance can process only one request at a time. The billing duration starts when the first request starts to be processed and ends when the last request is processed.



- A single instance that concurrently processes multiple requests
  For a single instance that concurrently processes multiple requests, Function Compute calculates fees based on the execution duration of the requests by the instance. This duration starts when the first request starts to be processed and ends when the last request is processed.



For more information, see Billing.

- Concurrent request limit
  By default, Function Compute supports a maximum of 300 pay-as-you-go instances in a region. Maximum number of requests that can be concurrently processed in a region = 300 × Value of the Single Instance Concurrency parameter. For example, if you set the Single Instance Concurrency parameter to 10, a maximum of 3,000 requests can be concurrently processed in a region. If the number of concurrent requests exceeds the maximum number of requests that Function Compute can process, the ResourceExhausted error occurs.

  > **Note** If you want to increase the number of pay-as-you-go instances in a region, you can contact Function Compute engineers.

- Logs

- For a single instance that processes a single request at a time, if you specify `X-Fc-Log-Type: Tail` in the HTTP header when you invoke a function, Function Compute returns the function logs in the `X-Fc-Log-Result` field that is in the response header. For a single instance that concurrently processes multiple requests, the response header does not include function logs because the logs of a specific request cannot be obtained among concurrent requests.
- For the Node.js runtime, the `console.info()` function is used to return the ID of the current request in the log. When an instance concurrently processes multiple requests, the `console.info()` function cannot display the correct IDs of all the requests. All the request IDs are changed to `req 2`. The following sample log is displayed:

```
2019-11-06T14:23:37.587Z req1 [info] logger begin
2019-11-06T14:23:37.587Z req1 [info] ctxlogger begin
2019-11-06T14:23:37.587Z req2 [info] logger begin
2019-11-06T14:23:37.587Z req2 [info] ctxlogger begin
2019-11-06T14:23:40.587Z req1 [info] ctxlogger end
2019-11-06T14:23:40.587Z req2 [info] ctxlogger end
2019-11-06T14:23:37.587Z req2 [info] logger end
2019-11-06T14:23:37.587Z req2 [info] logger end
```

  In this case, the `context.logger.info()` function can be used to display logs. This ensures that the correct ID of a request is returned. The following part shows the sample code:

```
exports.handler = (event, context, callback) => {
  console.info('logger begin');
  context.logger.info('ctxlogger begin');
  setTimeout(function() {
    context.logger.info('ctxlogger end');
    console.info('logger end');
    callback(null, 'hello world');
  }, 3000);
};
```

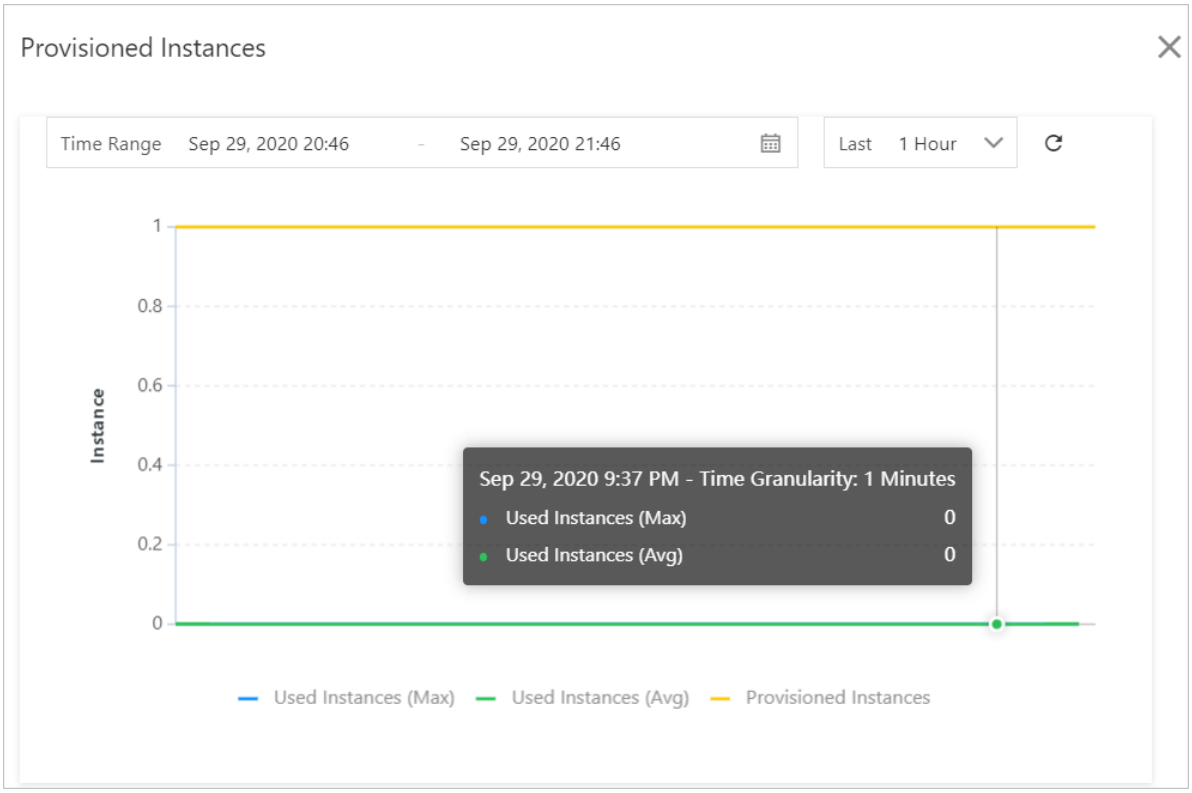
- Troubleshooting
  When an instance concurrently processes multiple requests, unexpected process quits caused by failed requests affect other concurrent requests. Therefore, you must compile troubleshooting logic to prevent impacts on other requests. The following example shows the sample Node.js code:

```
exports.handler = (event, context, callback) => {
  try {
    JSON.parse(event);
  } catch (ex) {
    callback(ex);
  }
  callback(null, 'hello world');
};
```

- Shared variables
  When an instance concurrently processes multiple requests, errors may occur if multiple requests attempt to modify the same variable at the same time. You must use the mutual exclusion method to prevent variable modifications that are not safe for threads when you define your functions. The following example shows the sample Java code:

```
public class App implements StreamRequestHandler
{
  private static int counter = 0;
  @Override
  public void handleRequest(InputStream inputStream, OutputStream outputStream, Context context) t
hrows IOException {
    synchronized (this) {
      counter = counter + 1;
    }
    outputStream.write(new String("hello world").getBytes());
  }
}
```

- Monitoring metrics
  After you set the instance concurrency for your function, you can see that the number of used instances is reduced in the instance monitoring chart.



Limits

Item	Description
Supported runtime environments	• Node.js Runtime • Java Runtime • Custom Runtime
Valid values of instance concurrency	1-100

Item	Description
Function execution logs provided in the X-Fc-Log-Result field in the response header	Not supported when the Single Instance Concurrency parameter is set to a value greater than 1

References

Set the request concurrency in a single instance

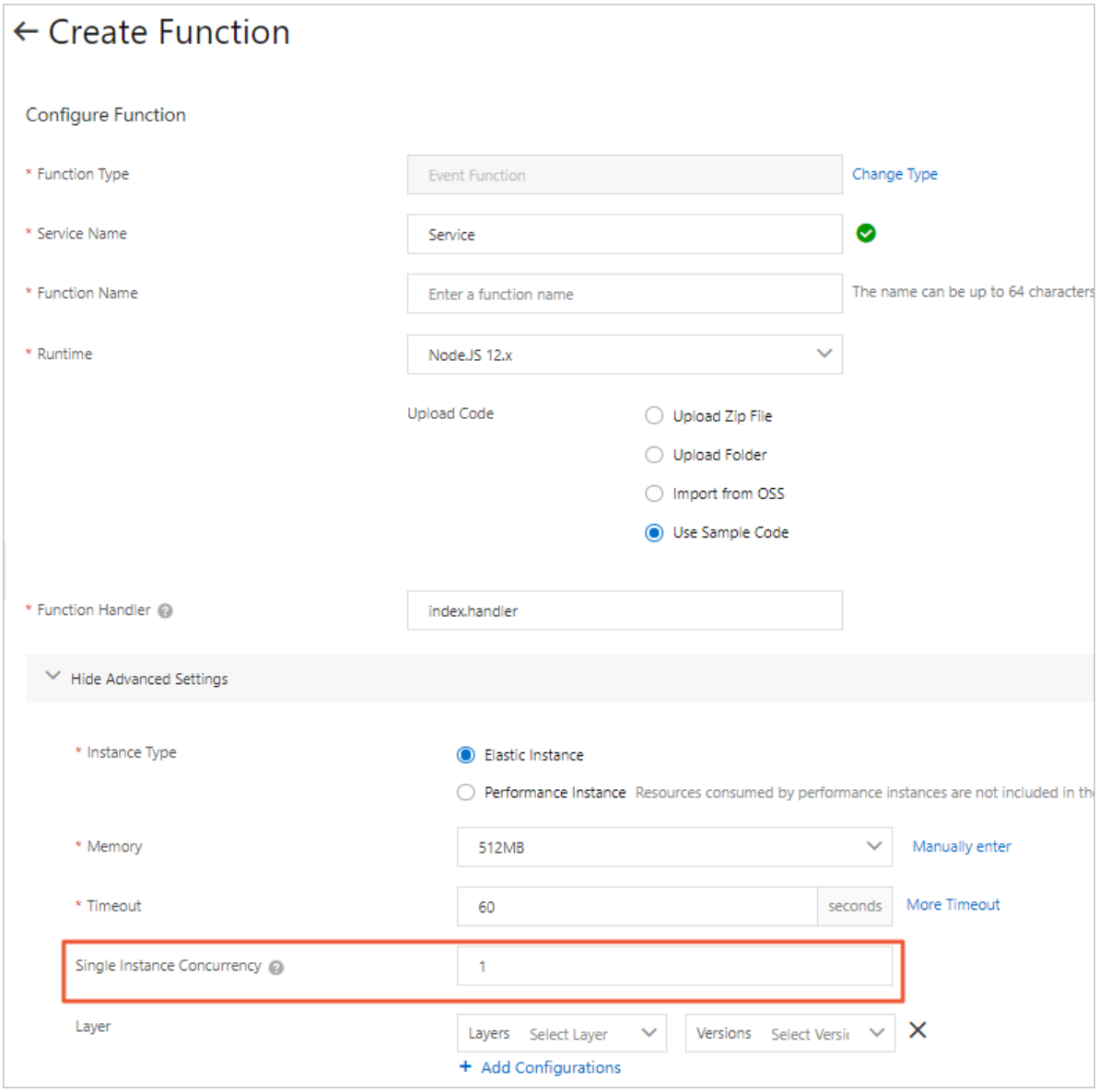
11.2. Set the request concurrency in a single instance

This topic describes how to set the request concurrency in a single instance in the Function Compute console.

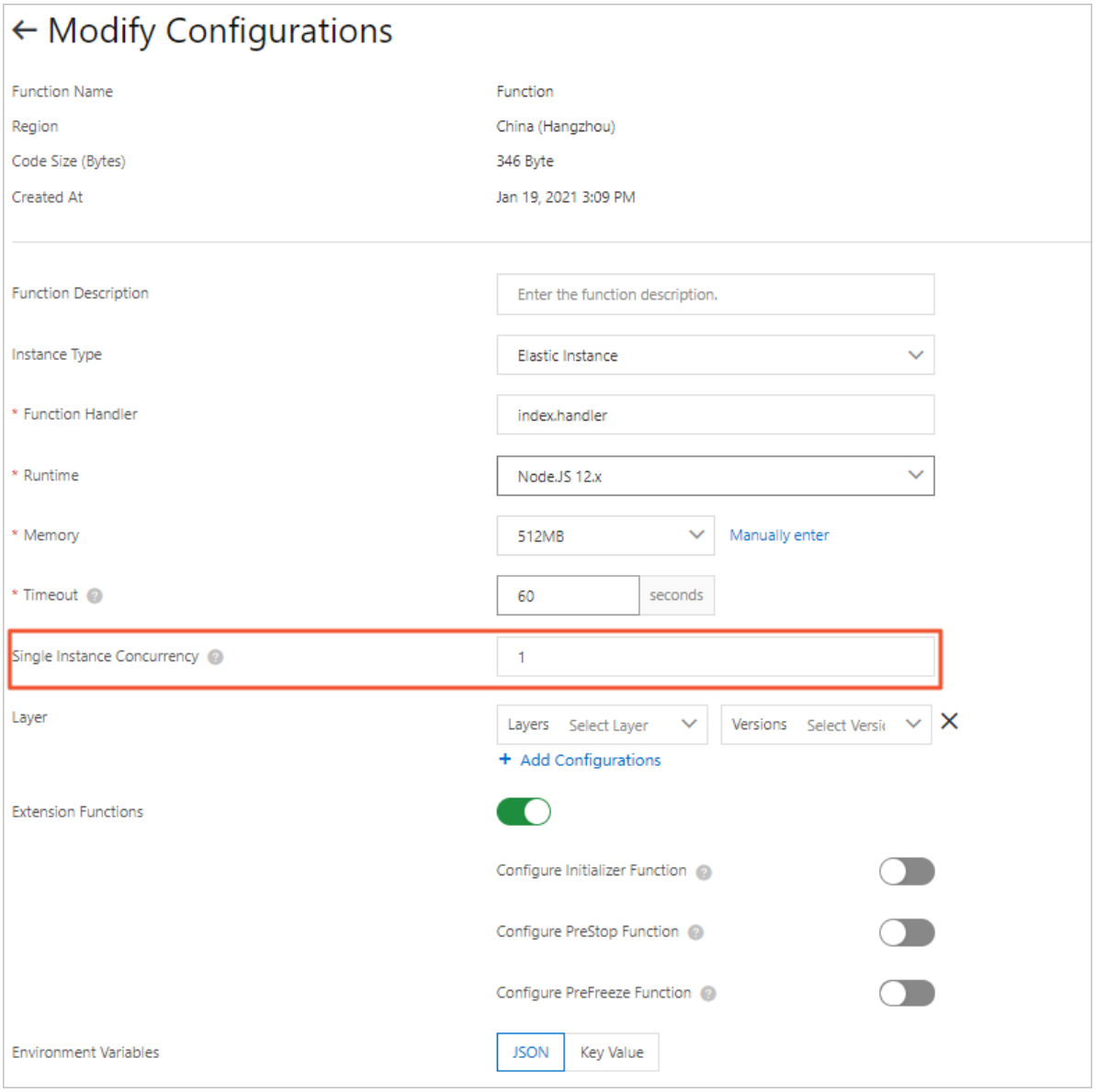
Procedure

You can set the request concurrency in a single instance when you create or update a function.

- For more information about how to set the concurrency when you create a function, see Create a function.



- For more information about how to set the concurrency when you update a function, see Modify a function.



If you use provisioned instances, the function in provisioned mode can concurrently process multiple requests. For more information, see Overview.

References

For more information about how to use the SDK for Node.js to set the concurrency, see Specify the instance concurrency.

11.3. Overview of configuring the maximum number of on-demand instances

This topic describes the background information, scenarios, limits, and instructions of configuring the maximum number of on-demand instances. This topic also describes how to calculate transactions per second (TPS).

Background information

To prevent unexpected invocations from generating unnecessary charges, you can set the maximum number of on-demand instances in each region within your account. This limit applies to all functions. For example, the account 123456789 can have up to 300 on-demand instances in a specific region. The account has three functions: function-a, function-b, and function-c. In this case, the maximum number of on-demand instances that concurrently process requests is 300.

Function Compute also allows you to set the maximum number of on-demand instances for a single function in the Function Compute console or by calling API operations. This prevents a large number of instances being occupied by a single function due to excessive invocations, protects backend resources, and prevents unexpected charges. For example, the account 123456789 has three functions: function-a, function-b, and function-c. You can set a maximum of 10 on-demand instances for function-a. In this case, a maximum of 10 instances can be used to process requests when you invoke function-a.

Scenarios

- Set a limit for a function to ensure the concurrency of another function.
  For example, function-a and function-b share the maximum number of instances within an account, and function-a is a key business function. If function-b is excessively invoked, normal requests of function-a may be affected. In this case, you can set the maximum number of on-demand instances for function-b to prevent it from consuming excessive instances.
- Protect downstream services.
  For example, if Function Compute makes a large number of access requests to ApsaraDB RDS, you can set the maximum number of on-demand instances for functions that are invoked to access ApsaraDB RDS. This way, ApsaraDB RDS stops responding requests that exceed its processing capacity.
- Terminate invocations of a function.
  If invocations of a function are abnormal, you can terminate invocations of the function by setting the maximum number of on-demand instances for the function to 0.
- Prevent unexpected invocations from generating unnecessary charges.
  You can set the maximum number of on-demand instances for a function to prevent unexpected invocations caused by the browser or client.
- Use with provisioned instances.
  You can set the maximum number of on-demand instances and provisioned instances and use one or both types of instances as needed.

Request processing methods after setting the on-demand instance limit

Type	Request processing method
Synchronous invocation	When the required number of on-demand instances exceeds the specified limit, the exceeded requests are denied and the `ResourceExhausted` error is returned.
Asynchronous invocation	When the number of required on-demand instances exceeds the specified limit, the requests are not denied but are queued up and processed in full load mode.

For more information, see Overview.

Use on-demand and provisioned instances in combination

If your function has provisioned instances, the provisioned instances are used first. When all provisioned instances are processing requests, new requests are processed by on-demand instances. The following table describes how to use on-demand instances and provisioned instances in combination.

On-demand instance limit	Provisioned instance limit	Instance usage
0	10	Up to 10 provisioned instances can be used, and no on-demand instances can be used. When provisioned instances are insufficient to process concurrent requests, new requests are throttled and error 429 is returned.
20	0	Up to 20 on-demand instances can be used, and no provisioned instances can be used.
50	30	Up to 50 on-demand instances can be used after all 30 provisioned instances are used. You can use a maximum of 80 instances.

Limits

- You can configure up to 100 instance limit rules for functions within an account in each region.
- The instance limit rules for functions must be applied to aliases or the latest version of a function.
- The maximum number of instances that is set in a rule cannot exceed the limit of the account, which is 300.
- Different limits can be set for different aliases of the function.

Calculate TPS

TPS indicates the number of requests that a function can process per second. You can set the maximum number of on-demand instances based on TPS and your business requirements.

TPS calculation formula: TPS = 1/DurationInSecond × InstanceConcurrency × MaxInstances

Assume that 0.1 second is taken to process a request, and a maximum of five instances can be used to execute the function. If a single instance can concurrently process two requests, the number of requests that the five instances can process per second is calculated based on the following formula: 1/0.1 × 2 × 5 = 100. In this case, TPS = 100.

References

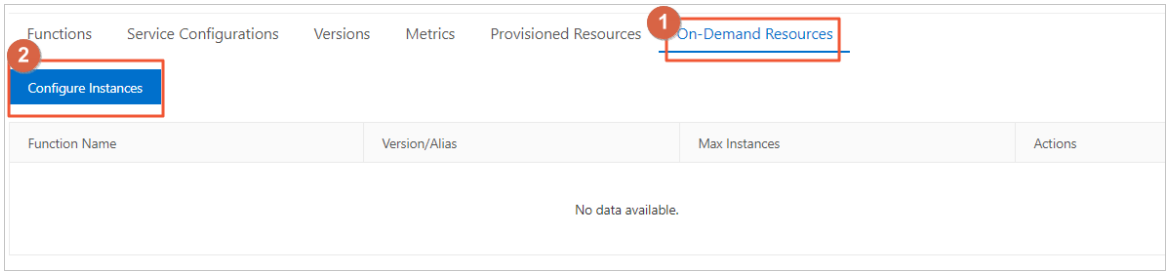
Set the maximum number of on-demand instances

11.4. Set the maximum number of on-demand instances

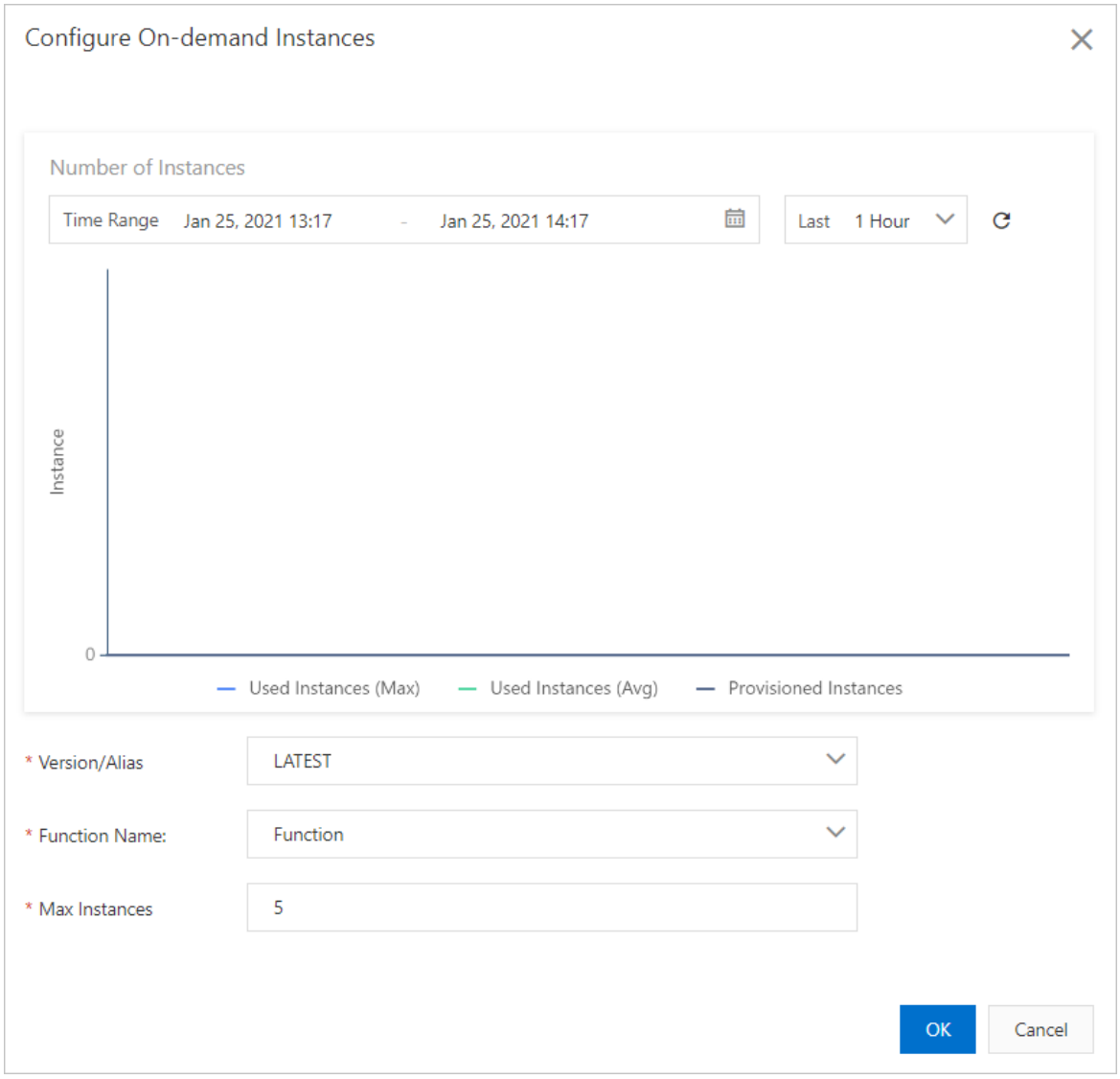
This topic describes how to set the maximum number of on-demand instances in the Function Compute console.

Procedure

1. Log on to the Function Compute console.
2. In the top navigation bar, select a region.
3. In the left-side navigation pane, click **Services and Functions**. In the **Services** pane, click the service that you require.
4. Click the **On-Demand Resources** tab. On the **On-Demand Resources** tab, click **Configure Instances**.



5. In the **Configure On-demand Instances** dialog box, set the parameters in the lower part and click **OK**.



Parameter	Description
Version/Alias	Select an alias on the Alias tab or LATEST on the Version tab for the function to be executed by the on-demand instance from the drop-down list.
Function Name	Select the function to be executed by the on-demand instance.
Max Instances	Enter the maximum number of on-demand instances that can be used to execute the function.

References

- For more information about how to use the SDK for Java to configure on-demand instances, see Example.
- For more information about how to use the SDK for Go to configure on-demand instances, see Configure pay-as-you-go instances.