

# Alibaba Cloud

## FunctionFlow Service Integration

Document Version: 20220120

# Legal disclaimer

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

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.
2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.
3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.
4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).
5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.
6. Please directly contact Alibaba Cloud for any errors of this document.

# Document conventions

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

---

# Table of Contents

1.Overview	05
2.Child flows of Serverless Workflow	08
3.Asynchronous function invocation	12
4.MNS queues	17
5.Integrate MNS topics	21

# 1. Overview

Serverless Workflow can integrate with multiple Alibaba Cloud services. After integration, these services are executed in task steps in Serverless Workflow. Service integration modes are defined in Flow Definition Language (FDL). In a task step, you can use `resourceArn` to define the service that you want to integrate, and use `pattern` to define the integration mode that you want to use. This topic describes information about service integration, including integration modes, context objects, and integrated cloud services.

For more information about the available cloud services, see [Integrated cloud services](#).

## Integration modes

Serverless Workflow supports the following integration modes:

- Request-response mode: In this mode, when you invoke a third-party service, Serverless Workflow proceeds to the next step immediately after it gets an HTTP response. This is the default mode. In a step of a flow defined in FDL, the `resourceArn` parameter defines the service, and the `pattern: requestResponse` parameter defines the service integration mode. The latter parameter is optional. If it is not specified, the default integration mode is used. In this mode, Serverless Workflow proceeds to the next step immediately after it receives an invocation response. The following example shows a child flow, in which Serverless Workflow is the integrated service:

```
version: v1
type: flow
steps:
  - type: task
    name: testSubflow
    resourceArn: acs:fnf::flow/flowABC #Defines the child flow.
    pattern: requestResponse #Sets the service integration mode to the request-response mode, which is the default mode.
  - type: pass
    name: dummy
```

In this example, when the `testSubflow` step is executed, the `flowABC` child flow is triggered. After `flowABC` is triggered, the flow proceeds to the `dummy` step, while `flowABC` may still be running.

- Synchronous mode: In this mode, a service generally provides an API for asynchronous execution. After Serverless Workflow invokes the API, Serverless Workflow waits for the relevant task to complete and the execution result to be returned before it proceeds to the next step. For specific integrated services, Serverless Workflow waits for the relevant task to complete and then proceeds to the next step. This type of service provides an asynchronous API to start a task. You must submit the task and wait for the task to complete before the next step starts. In a step of a flow defined in FDL, the `resourceArn` parameter defines the service, and the `pattern: sync` parameter defines the service integration mode. The following example shows a child flow, in which Serverless Workflow is the integrated service:

```

version: v1
type: flow
steps:
  - type: task
    name: testTask
    resourceArn: acs:fnf::flow/flowABC #Defines the child flow.
    pattern: sync #Sets the service integration mode to the synchronous mode.
  - type: pass
    name: dummy

```

In this example, when the task step is executed, a Serverless Workflow child flow is triggered. After the child flow is triggered, the flow waits for its execution result and then proceeds to the next step. When the `testTask` step is executed, the `flowABC` child flow is triggered. After `flowABC` is triggered, the flow waits for its execution result. After `flowABC` is completed, the flow proceeds to the `dummy` step. At this point, `flowABC` has been completed.

- **Wait-for-callback mode:** In this mode, when you invoke a third-party service and pass in the task token, Serverless Workflow waits for you to use the token to notify the flow of the execution result before the flow proceeds to the next step. The callback task suspends the current flow at the task scheduling point until the callback instruction for the corresponding task token is received. In a step of a flow defined in FDL, the `resourceArn` parameter defines the service, and the `pattern: waitForCallback` parameter defines the service integration mode. The following example shows a child flow, in which Serverless Workflow is the integrated service:

```

version: v1
type: flow
steps:
  - type: task
    name: testSubflow
    resourceArn: acs:fnf::flow/flowABC #Defines the child flow.
    pattern: waitForCallback #Set the service integration mode to the wait-for-callback mode.
  - type: pass
    name: dummy

```

In this example, when the `testSubflow` step is executed, the `flowABC` child flow is triggered. After `flowABC` is triggered, the flow is paused to wait for a callback that is implemented with the invocation of the `ReportTaskSucceed` or `ReportTaskFailed` operation. After the flow receives and processes a callback request, it proceeds to the `dummy` step, regardless of whether `flowABC` has been completed. The callback is initiated by you.

## Context objects

A context object is an internal JSON object in a flow execution instance, which contains information about the execution and steps. This object allows access from external services. To implement the access, you can map the context object to a specific variable in input Mappings. The following example shows the structure of the available context objects:

```

"context": {
  "flow": {
    // The unique ID, name, and string type of the flow.
    "id": "val1",
    "name": "val2",
  },
  "execution": {
    // The name of the execution.
    "name": "val3"
  },
  "step": {
    // The name of the step.
    "name": "val4"
    // The event ID of the step.
    "eventId": "val5"
    // The iteration index. This parameter is valid in a foreach step.
    "IterationIndex": "val6",
  },
  "task": {
    // The identifier of the step, which is a string. This parameter is valid in the wait
    -for-callback mode.
    "token": "val7",
  },
}

```

To integrate the Serverless Workflow service, you must obtain from the child flow the information about the parent flow that invokes this child flow, and obtain `taskToken` in this invocation step. `taskToken` will be used in the callback. You can obtain the target and source fields in the following way:

```

...
inputMappings:
- target: current_flow_name
  source: $context.flow.name
- target: current_execution_name
  source: $context.execution.name
- target: current_step_task_token
  source: $context.task.token

```

## Integrated cloud services

Service	Request-response mode	Synchronous mode	Wait-for-callback mode
Function Compute	Supported	Not supported	Not supported
Message Service (MNS) queues	Supported	Not supported	Supported
MNS topics	Supported	Not supported	Supported
Serverless Workflow	Supported	Supported	Supported

## 2. Child flows of Serverless Workflow

The feature of integrating child flows of Serverless workflow allows you to execute another flow in a flow. This topic describes scenarios, integration modes, context objects, and input and output rules of child flows of Serverless workflow.

### Scenarios

The feature of integrating child flows of Serverless workflow can be applied in the following scenarios:

- Divide a flow into multiple child flows to reduce the flow complexity.
- Facilitate the reuse of flows. You can put some common steps in a flow and reuse these steps in other flows.
- Remove limits of the current single flow. For example, a single flow can contain a maximum of 5,000 events by default and the maximum execution time is one year.
- Implement error handling for the overall output of the flow. For example, you can design a parallel step as a child flow and handle errors that occur in the execution of the child flows in the parent flow.

### Integration modes

Serverless workflow supports three integration modes: request/response, synchronous, and wait-for-callback modes.

- Request/response mode  
In this mode, the parent flow starts to execute the next step immediately after the child flow is started. The following piece of code defines the flow:

```
version: v1
type: flow
steps:
  - type: task
    name: fnfInvoke
    resourceArn: acs:fnf::flow/subflow_demo_child
    pattern: requestResponse # The default mode. You do not need to specify this parameter.
    inputMappings: # If inputMappings is not specified, the default mapping rule is used.
      # In other words, the parameters of the parent flow are used as the input into the child flow.
    - target: childName # The execution name of the child flow that will be initiated in the service.
      source: $input.childName
    serviceParams: # The parameters of the service that is integrated in Serverless workflow.
      # You do not need to specify this parameter. If this parameter is not specified, a random string
      # is used as the name of this execution, and the parameter included in inputMappings is used
      # as the input into the child flow.
    Input: $ # Use the mapped input parameter as the input parameter to initiate the child flow.
    ExecutionName: $.childName # If a variable is specified in serviceParams, ensure that this variable
      # exists in inputMappings.
```

- Synchronous mode



In this mode, the parent flow starts a child flow and proceeds to the next step after the execution of the child flow is completed. The following piece of code defines the flow:

```
version: v1
type: flow
steps:
  - type: parallel
    name: parallelTask
    branches:
      - steps:
          # In this step, Serverless workflow is integrated in the synchronous mode, where
          # inputMappings is used as the input into the child flow, and the execution name of the child
          # flow is dynamically specified based on the input into the parent flow.
          - type: task
            name: fnfSync
            resourceArn: acs:fnf::flow/subflow_demo_child
            pattern: sync
            inputMappings: # If inputMappings is not specified, the default mapping rule is
            # used. In other words, the parameters of the parent flow are used as the input into the child
            # flow.
            - target: childSyncName # The execution name of the child flow that will be initiated
            # in the service. If you want to specify an execution name for the child flow, perform
            # input mapping for the target execution name and use the mapped result in serviceParams,
            # as shown in this example.
              source: $input.childSyncName.
            serviceParams: # The parameters of the service that is integrated in Serverless
            # workflow.
            Input: $ # Use the mapped input parameter as the input parameter to initiate
            # the child flow. We recommend that you use the provided method unless you explicitly specify
            # the behavior and syntax for Input in other ways.
            ExecutionName: $.childSyncName # If a variable is specified in serviceParams,
            # ensure that this variable exists in inputMappings.
```

- Wait-for-callback mode

In this mode, the parent flow starts a child flow and enters the suspended state until it receives a callback notification. The following piece of code defines the flow:

```

version: v1
type: flow
steps:
  - steps:
      # In this step, Serverless workflow is integrated in the wait-for-callback mode, where the inputMappings is used as the input into the child flow, and the execution name of the child flow is dynamically specified based on the input into the parent flow.
      - type: task
        name: fnfWaitForCallback
        resourceArn: acs:fnf::flow/subflow_demo_child
        pattern: waitForCallback
        inputMappings: # If inputMappings is not specified, the default mapping rule is used. In other words, the parameters of the parent flow are used as the input into the child flow.
          -target: task_token # To ensure that callbacks can be used in the child flow, map task_token to a custom name.
            source: $context.task.token # Obtain the task token that identifies the task from the context object.
          - target: childCallbackName
            source: $input.childCallbackName
        serviceParams: # The parameters of the service that is integrated in Serverless workflow.
          Input: $ # Use the mapped input parameter as the input parameter to initiate the child flow.
          ExecutionName: $.childCallbackName # If a variable is specified in serviceParams, ensure that this variable exists in inputMappings.

```

## Context object description

In the integration modes for child flows, you can pass the `$context.execution.name` and `$context.flow.name` variables to a child flow to identify the parent flow that starts the child flow. In the `wait-for-callback` mode, `$context.task.token` is used to pass the identifier of the parent flow to the child flow to implement the callback.

## Input and output rules for child flows

- Request/response mode

In this mode, the input into a child flow derives from that of the corresponding task step. You can obtain the input by specifying `$Input` in the child flow.

The start information of the child flow (response from the call to `StartExecution`) is used as the output, whereas the output of the child flow is ignored by the parent flow. After a child flow is started, the following start information of the child flow is provided by default: `$local.ExecutionName`, `$local.FlowName`, and `$local.RequestId`. If you want to map other output parameters of the child flow to the parent flow, you can map them by performing `output mapping` in the corresponding step in the parent flow.

```

- type: task
  pattern: requestResponse
  ...
  outputMappings: # In the request/response mode, you can obtain the $local.ExecutionName, $local.FlowName, and $local.RequestId parameters.
    - target: subflow_children_request_id
      source: $local.RequestId # The ID of the request that initiates the child flow.
    - target: subflow_children_exec_name
      source: $local.ExecutionName # The execution name of the child flow that will be initiated.
    - target: subflow_children_flow_name
      source: $local.FlowName # The flow name of the child flow that will be initiated
  .

```

- Synchronous mode

In this mode, the input into a child flow derives from that of the corresponding task step. You can obtain the input by specifying `$Input` in the child flow.

The output of the child flow, which is the `Output` in the response from the call to `DescribeExecution`, is returned to the parent flow as the output of this step in the parent flow. You can use this output in subsequent steps of the parent flow. If you want to map other output parameters of the child flow to the parent flow, you can map them by performing `output mapping`.

- Wait-for-callback mode

In this mode, the input into a child flow derives from that of the corresponding task step. You can obtain the input by specifying `$Input` in the child flow.

The output of the callback is used as the output of this step in the parent flow. The value of the `Output` parameter that you pass in when you call the `ReportTaskSucceeded` operation is used as the output of this step in the parent flow. The values of the `Error` and `Cause` parameters that you pass in when you call the `ReportTaskFailed` operation are used as the output of this step in the parent flow. If you want to map other output parameters of the child flow to the parent flow, you can map them by performing `output mapping`.

## 3. Asynchronous function invocation

Serverless Workflow can asynchronously invoke functions of Function Compute. Asynchronous invocation is ideal for some scenarios such as long-term tasks and manual audits. This can help you prevent throttling errors and simplify troubleshooting procedures and retry logic. This topic describes the scenarios, integration patterns. This topic also provides an example on how to integrate the asynchronous function invocation feature into Serverless Workflow.

### Prerequisites

The following operations are complete:

- (Optional) Asynchronous invocation is enabled. For more information, see [Asynchronous invocation](#). After you enable stateful asynchronous invocation for your functions, you can stop the function instance when the task is being executed. This helps you view the flow from a finer-grained perspective.
- Function Compute is granted the permissions to access Serverless Workflow. This ensures that Serverless Workflow can be called back to execute subsequent steps after the asynchronous function execution is complete. The following snippet shows how to configure the permission policy.

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

For more information about how to grant permission to Function Compute, see [Grant Function Compute permissions to access other Alibaba Cloud services](#).

### Background information

By default, Serverless Workflow uses the synchronous pattern when Serverless Workflow orchestrates Function Compute tasks to implement task flows. In the synchronous pattern, Serverless Workflow does not proceed to the next step until the flow receives the execution results of the previous function. The following snippet shows a flow that uses the synchronous pattern.

```
version: v1
type: flow
steps:
  - type: task
    name: mytask
    resourceArn: acs:fc:{region}:{account}:services/{serviceName}.{qualifier}/functions/{functionName}
```

The following issues may occur when you use the synchronous pattern for function invocation:

- The resource limits to invocation of functions of Function Compute may cause throttling errors. By default, a maximum of 300 pay-as-you-go instances are allowed in a single region for an Alibaba Cloud account of Function Compute. When you use Serverless Workflow to invoke a function, the invocation shares the same quota with the function invocations that are initiated by other services. This may cause throttling errors. In this case, you need to define a complex retry policy in your Serverless Workflow flow. However, this does not ensure that the flow is executed successfully.
- If the task requires a long time to execute, a persistent connection must be established between Serverless Workflow and Function Compute. Unstable network conditions may cause unexpected errors.

Serverless Workflow allows you to combine the asynchronous function invocation feature with various integration patterns to solve the preceding issues. At the same time, the combination can meet the following requirements:

- In some scenarios, the system needs to perform subsequent steps without waiting for the execution completion of the current step.
- If unexpected errors occur when the flow is being executed, the system needs to skip the current step and proceed to the next step.

## Scenarios

In different scenarios, Serverless Workflow uses the request-response pattern (requestResponse), synchronous pattern(sync), or wait-for-callback pattern (waitForCallback) to asynchronously invoke functions of Function Compute.

Integration pattern	Parameter	Scenario
Request-response pattern (default pattern)	<div>requestResponse</div> <div>Sample flow:</div> <pre>- type: task   ...   pattern: requestResponse   serviceParams:     InvocationType: Async</pre>	Tasks are long-running and the task execution results are non-consequential.
Synchronization pattern	<div>sync</div> <div>Sample flow:</div> <pre>- type: task   ...   pattern: sync   serviceParams:     InvocationType: Async</pre>	Tasks are long-running and may be throttled.

Integration pattern	Parameter	Scenario
Wait-for-callback pattern	<pre>waitForCallback</pre> <p>Sample flow:</p> <pre>- type: task   ...   pattern: waitForCallback   serviceParams:     InvocationType: Async</pre>	When a flow arrives at a specific step, the flow needs to proceed to subsequent steps regardless of whether the execution of the current step is complete. A sample scenario is manual audit.

## Service parameters

In the following examples, Serverless Workflow uses Function Compute as the task node, uses `resourceArn` to specify Function Compute as the destination service, and uses `serviceParams` to specify the parameters that are used to invoke functions of Function Compute. `serviceParams` supports the following parameters:

- `InvocationType`: the invocation type of functions. The valid values are `Sync` and `Async`. `Sync` indicates synchronous invocation. `Async` indicates asynchronous invocation.
- (Optional) `StatefulAsyncInvocationID`: the ID of a stateful asynchronous invocation. You can use this parameter to search for the name of the destination task in the Function Compute console.

**Note** If you set `InvocationType` to `Async` and specify a value for `StatefulAsyncInvocationID`, the function invocation pattern of Serverless Workflow is stateful and asynchronous invocation. The value of the `StatefulAsyncInvocationID` parameter must be unique for all the functions.

## Integration pattern of asynchronous function invocation

When a flow arrives at the specified step, Serverless Workflow asynchronously invokes a function and executes the function. Then, the flow proceeds to the next step without waiting for a callback or the execution completion of the preceding step.


```
version: v1
type: flow
steps:
  - type: task
    name: mytask
    resourceArn: acs:fc:{region}:{account}:services/{serviceName}.{qualifier}/functions/{functionName}
    pattern: requestResponse # Async invocation with sync pattern
    serviceParams:
      InvocationType: Async
```

The preceding example shows that a function is triggered to execute when the flow arrives at the `mytask` step. After the function is triggered, the flow proceeds to the next step when the execution of the `mytask` step may not be complete.

When a flow arrives at the specified step, Serverless Workflow asynchronously invokes a function and executes the function. Then, the flow is suspended and does not proceed to the next step until the Serverless Workflow flow is notified that the function execution is complete.

```
version: v1
type: flow
steps:
  - type: task
    name: mytask
    resourceArn: acs:fc:{region}:{account}:services/{serviceName}.{qualifier}/functions/{functionName}
    pattern: sync # Async invocation with sync pattern
    serviceParams:
      InvocationType: Async
```

Serverless Workflow The preceding example shows that when the flow arrives at the `mytask` step, a function is triggered to execute. Then the flow is suspended and does not proceed to the next step until the flow is informed that the function execution is complete.

 **Note** The use of the synchronous integration pattern is the same as the use of the synchronous function invocation feature in terms of operations. Only the function invocation methods are different.

When a flow arrives at the specified step, Serverless Workflow asynchronously invokes a function, executes the function, and passes in a task token. Then, the flow is suspended. Regardless of whether the function execution is complete, the flow does not proceed to the next step until you use the task token to manually notify the flow of the function execution result.

```
version: v1
type: flow
steps:
  - type: task
    name: mytask
    resourceArn: acs:fc:{region}:{account}:services/{serviceName}.{qualifier}/functions/{functionName}
    pattern: waitForCallback # Async invocation with sync pattern
    serviceParams:
      InvocationType: Async
```

The preceding example shows that a function is triggered to execute when the flow arrives at the `mytask` step. After the function is triggered, the flow is suspended to wait for a callback that is implemented with the invocation of the `ReportTaskSucceed` or `ReportTaskFailed` API operation. The flow does not proceed to the next step until the flow receives a callback request and processes the callback request. The callback request is initiated by you. You can initiate the callback request when the function execution is complete or not.

## Request-response pattern

## Synchronization pattern

## Wait-for-callback pattern

## Example: asynchronous function invocation

Serverless Workflow combines the asynchronous function invocation feature and the stateful asynchronous invocation feature to provide supports for job-type scenarios. When you enable the stateful asynchronous invocation feature and use the asynchronous invocation pattern, you can view the function execution status and stop the function execution when and after the function is invoked. This makes the flow observable and easy to operate. The following snippet shows the flow definition language (FDL) of Serverless Workflow.

```
version: v1
type: flow
steps:
  - type: task
    name: mytask
    resourceArn: acs:fc::services/{serviceName}.{qualifier}/functions/{functionName}
    pattern: sync # Async invocation with sync pattern
    inputMappings:
      - target: id
        source: $context.execution.name
    serviceParams:
      InvocationType: Async
      StatefulAsyncInvocationID: $.id
```



## 4.MNS queues

In the task step of Serverless workflow, you can send messages to a specified Message Service (MNS) queue. This topic describes the scenarios, common parameters, integration modes, and permission configurations for integrating MNS queues.

### Scenarios

You can integrate the MNS queues in the task step in the request/response and wait-for-callback modes. The following table lists the scenarios of these two modes.

Integration mode	Parameter	Scenario	Description
Request/response mode	requestResponse	Event notification	Services other than flow execution are persistently notified, and the flow execution is irrelevant to how the notified services process the message.
Wait-for-callback mode	waitForCallback	Orchestration of custom task types	After a task executor in an environment, such as an Elastic Compute Service (ECS) virtual machine or a server in an on-premises data center, receives a message sent to a queue, the task executor calls back Serverless workflow after it executes the relevant task. For more information, see <a href="#">Best Practices</a> .

### Common parameters

In the following example, Serverless workflow sends a message to the queue specified in `resourceArn`. The specified queue name replaces `{queue-name}`. The body and parameters of the message are specified in `serviceParams`. The following content lists the supported parameters:

- **MessageBody:** The body of the message, which is required. `MessageBody: $`: The message body is generated by inputMappings. The following content describes how the message body is generated when the step enters the StepEntered event:
  - The input object is `{"key": "value_1"}`.
  - The mapped input is `{"key_1": "value_1", "key_2": "value"}`.
  - The task step uses the `{"key_1": "value_1", "key_2": "value"}` string as the message body and sends it to the MNS queue.
- **Priority:** The priority of the message, which is required.
- **DelaySeconds:** The message latency, in seconds. The parameter is optional.

For more information about parameters, see [SendMessage](#) in MNS documentation.

```
version: v1
type: flow
steps:
  - type: task
    name: Task_1
    resourceArn: acs:mns:::/queues/{queue-name}/messages # The task step sends messages to
the MNS queue named {queue-name} under the same account in the same region.
    pattern: requestResponse # The task step ends after the mes
sage is sent to the MNS queue.
    inputMappings:
      - target: key_1
        source: $input.key
      - target: key_2
        source: value
    serviceParams: # The service integration parameters.
      MessageBody: $ # The mapped input is used as the body of the message you want to sen
d.
      DelaySeconds: 0 # The message latency, in seconds.
      Priority: 1 # The priority of the MNS queue.
```

## Integration modes

- Request/response mode

The `pattern: requestResponse` parameter of the task step indicates the request/response mode. In this mode, Serverless workflow sends a message to the `{queue-name}` queue specified in `resourceArn`. After the message is sent, the step is completed. If the message fails to be sent, the step fails or is retried based on the retry configurations.

```
version: v1
type: flow
steps:
  - type: task
    name: mytask
    resourceArn: acs:mns:::/queues/{queue-name}/messages # The task step sends messages t
o an MNS queue under the same account in the same region.
    pattern: requestResponse # The task step ends after the m
essage is sent to the MNS queue.
    inputMappings:
      - target: key_3
        source: value
    serviceParams: # The service integration parameters.
      MessageBody: $ # {"key_3": "value"} is sent as the message body to the MNS queue.
      DelaySeconds: 0 # The message latency, in seconds.
      Priority: 1 # The priority of the MNS queue.
```

- Wait-for-callback mode

In the task step, `pattern: waitForCallback` indicates the wait-for-callback mode. In this mode, Serverless workflow sends a message to the `{queue-name}` queue specified in `resourceArn`. After the message is sent, the step suspends and waits for a callback. The task executor needs to use the corresponding task token to call back Serverless workflow. If the callback result indicates that the call to `ReportTaskSucceeded` is successful, the task step is successful. If the callback result indicates that the `ReportTaskFailed` operation was called, the step fails or is retried based on the retry configuration.

`taskToken`: The task token can be obtained from the context object of this step. `inputMappings` is used to assign `taskToken` to a field in the JSON object of the message body. In the following example, the context object in this step is `{"task":{"token":"my-token-1"}}`. The mapped task input is `{"task_token": "my-token-1", "key": "value"}`, which is also sent as the message body to the MNS queue.

```
version: v1
type: flow
steps:
  - type: task
    name: mytask
    resourceArn: acs:mns::/queues/{queue-name}/messages # The task step sends messages to
    the MNS queue named {queue-name} under the same account in the same region.
    pattern: waitForCallback # The task step suspends after the message is sent to the MNS
    queue and waits until it receives the callback.
    inputMappings:
      - target: task_token
        source: $context.task.token # Serverless Workflow queries the task token from the
        context object.
      - target: key
        source: value
    serviceParams: # The service integration parameters.
      MessageBody: $ # The mapped input is used as the body of the message you want to send.
    Priority: 1 # The priority of the MNS queue.
```

## Flow role configuration

Serverless workflow assumes the RAM role specified by the flow to obtain your temporary AccessKey pair to access MNS on your behalf. Therefore, you must grant the flow role the `SendMessage` permission to send messages to MNS. You can select either a system policy or a custom policy to grant Serverless workflow the permission to send messages to your MNS queue.

- **System Policy:** grants the `AliyunMNSFullAccess` system permission to the flow role.
- **Custom Policy:** If you only allow the flow role to send messages to the `{queue-name}` queue, create the following policy and bind it to the flow role.

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": "mns:SendMessage",
      "Resource": "acs:mns:*:*:/queues/{queue-name}/messages",
      "Effect": "Allow"
    }
  ]
}
```

## 5. Integrate MNS topics

In Serverless workflow, task steps have integrated Alibaba Cloud Message Service (MNS) topics. You can publish messages to MNS topics by defining task steps in a flow. By setting PublishMessage parameters, you can push messages that are from MNS topics to queues, HTTP endpoints, and email addresses.

### Integration modes

The following section describes two integration modes of MNS topics: request/response mode and wait-for-callback mode. You can use one of them to orchestrate MNS topics.

- Request/response mode

In this mode, after a PublishMessage request is sent, the flow proceeds with the task steps. The following piece of code defines the flow:

```
version: v1
type: flow
steps:
  - type: task
    name: mns-topic-task
    resourceArn: acs:mns::/topics/{topicName}/messages # mns topic resource arn
    pattern: requestResponse # Optional. The default mode.
    outputMappings:
      # Response parameters of PublishMessage
      - target: messageId # The ID of the message.
        source: $local.MessageId
      - target: requestId # The ID of the request.
        source: $local.RequestId
      - target: messageBodyMD5 # The MD5 value of the message body.
        source: $local.MessageBodyMD5
    serviceParams:
      # Request parameters of PublishMessage
      MessageBody: $.messageBody # The message body.
      MessageTag: $.messageTag # Optional. The message tag.
      MessageAttributes: $.messageAttributes # The additional message attributes, which must be in JSON format.
```

- Wait-for-callback mode

In this mode, after a PublishMessage request is sent, the flow holds on the task steps until a callback notification is received. The following piece of code defines the flow:

```
version: v1
type: flow
steps:
  - type: task
    name: mns-topic-task
    resourceArn: acs:mns::/topics/{topicName}/messages # mns topic resource arn
    pattern: waitForCallback # The wait-for-callback mode.
    inputMappings:
      - target: messageBody
        source: $input.messageBody
      - target: messageTag
        source: $.messageTag
      - target: messageAttributes
        source: $.messageAttributes
      - target: taskToken # The token that is automatically generated for task status cal
        lbacks
        source: $context.task.token
    serviceParams:
      # Request parameters of PublishMessage
      MessageBody: $ # The message body, in which TaskToken is encapsulated.
      MessageTag: $.messageTag # Optional. The message tag.
      MessageAttributes:
        $.messageAttributes # Optional. The additional message attributes.
```

## Description of parameters

- context
  - TaskToken  
In the wait-for-callback mode, a task step automatically generates `TaskToken` for task status callbacks. You can use `$context.task.token` to query the value. You can run task status callbacks by passing the `TaskToken` parameter to call the [ReportTaskSucceeded](#) and [ReportTaskFailed](#) operations.
- Fields in serviceParams
  - MessageBody: This is the message body string to be sent.
  - MessageTag: Optional. This is the tag of the message to be sent.
  - MessageAttributes: Optional. You must specify this parameter to push messages in specific ways. The value must be a JSON string.
  - Mail push: The parameters must include MailAttributes, as shown in the following code:

```
{
  "MailAttributes": {
    "Subject": "{Email subject}",
    "AccountName": "{Sender address (email address)}",
    "AddressType": 0,
    "IsHTML": true,
    "ReplyToAddress": 0
  }
}
```

For more information about the preceding parameters, see [PublishMessage](#).