Red Hat OpenShift Serverless 1.33

Serverless Logic

OpenShift Serverless Logic 简介

Last Updated: 2024-07-16

Red Hat OpenShift Serverless 1.33 Serverless Logic

OpenShift Serverless Logic 简介

法律通告

摘要

本文档概述 OpenShift Serverless 日志功能。

目录

第 1 章 开始使用

1.1. 使用 KNATIVE WORKFLOW 插件创建并运行工作流

您可以在本地创建并运行 OpenShift Serverless Logic 工作流。

1.1.1. 创建工作流

您可以使用带有 **kn 工作流** 的 **create** 命令在当前目录中设置新的 OpenShift Serverless Logic 项目。

先决条件

- 已安装 OpenShift Serverless Logic **kn-workflow** CLI 插件。

流程

1. 运行以下命令，创建一个新的 OpenShift Serverless Logic 工作流项目：

```
$ kn workflow create
```

默认情况下，生成的项目名称是 **new-project**。您可以使用 **[-n|--name]** 标志来更改项目名称，如下所示：

示例命令

```
$ kn workflow create --name my-project
```

1.1.2. 在本地运行工作流

您可以使用带有 **kn 工作流** 的 **run** 命令，在当前目录中构建并运行 OpenShift Serverless Logic 工作流项目。

先决条件

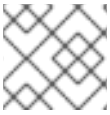
- 您已在本地机器上安装了 Podman。
- 已安装 OpenShift Serverless Logic **kn-workflow** CLI 插件。
- 您已创建了 OpenShift Serverless Logic 工作流项目。

流程

1. 运行以下命令来构建并运行 OpenShift Serverless Logic 工作流项目：

```
$ kn workflow run
```

当项目就绪时，Dev Development UI 会在 **localhost:8080/q/dev-ui** 的浏览器中自动打开，您将找到可用的 **Serverless Workflow Tools** 标题。另外，您可以使用 **http://localhost:8080/q/dev-ui/org.apache.kie.sonataflow.sonataflow-quarkus-devui/workflows** 直接访问该工具。



注意

您可以使用机器中运行的容器在本地执行工作流。使用 Ctrl+C 停止容器。

1.2. 部署工作流

您可以以两种模式(Dev)模式和预览模式在集群中部署 Serverless Logic 工作流。

1.2.1. 在 Dev 模式中部署工作流

您可以在 OpenShift Container Platform 中以 Dev 模式部署本地工作流。您可以使用此部署在集群中直接试验和修改工作流，查看几乎立即的更改。Dev 模式专为开发和测试目的而设计。它是初始开发阶段以及测试新更改的理想选择。

先决条件

- 已在集群中安装了 OpenShift Serverless Logic Operator。
- 您可以使用适当的角色和权限访问 OpenShift Serverless Logic 项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 已安装 OpenShift CLI **(oc)**。

流程

1. 创建工作流配置 YAML 文件。

   workflow-dev.yaml 文件示例

```
apiVersion: sonataflow.org/v1alpha08
kind: SonataFlow
metadata:
  name: greeting (1)
  annotations:
    sonataflow.org/description: Greeting example on k8s!
    sonataflow.org/version: 0.0.1
    sonataflow.org/profile: dev (2)
spec:
  flow:
    start: ChooseOnLanguage
    functions:
      - name: greetFunction
        type: custom
        operation: sysout
    states:
      - name: ChooseOnLanguage
        type: switch
        dataConditions:
          - condition: "${ .language == \"English\" }"
            transition: GreetInEnglish
          - condition: "${ .language == \"Spanish\" }"
            transition: GreetInSpanish
        defaultCondition: GreetInEnglish
      - name: GreetInEnglish
```

```
    type: inject
    data:
      greeting: "Hello from JSON Workflow, "
    transition: GreetPerson
  - name: GreetInSpanish
    type: inject
    data:
      greeting: "Saludos desde JSON Workflow, "
    transition: GreetPerson
  - name: GreetPerson
    type: operation
    actions:
      - name: greetAction
        functionRef:
          refName: greetFunction
          arguments:
            message: ".greeting + .name"
    end: true
```

1. 是 workflow_name
2. 表示您必须在 Dev 模式中部署工作流

2. 要部署应用程序，请输入以下命令应用 YAML 文件：

```
$ oc apply -f <filename> -n <your_namespace>
```

3. 输入以下命令验证部署并检查部署工作流的状态：

```
$ oc get workflow -n <your_namespace> -w
```

确定列出了您的工作流，其状态为 **Running** 或 **Completed**。

4. 输入以下命令直接在集群中编辑工作流：

```
$ oc edit sonataflow <workflow_name> -n <your_namespace>
```

5. 编辑后，保存更改。OpenShift Serverless Logic Operator 会检测更改并相应地更新工作流。

验证

1. 要确保正确应用更改，请输入以下命令验证工作流的状态和日志：
   a. 运行以下命令，查看工作流的状态：

```
$ oc get sonataflows -n <your_namespace>
```

   b. 运行以下命令来查看工作流日志：

```
$ oc logs <workflow_pod_name> -n <your_namespace>
```

后续步骤

1. 完成测试后，运行以下命令来删除资源以避免不必要的用法：

```
$ oc delete sonataflow <workflow_name> -n <your_namespace>
```

1.2.2. 以预览模式部署工作流

您可以以 Preview 模式在 OpenShift Container Platform 上部署本地工作流。这可让您在集群中直接试验和修改工作流，查看几乎立即的更改。在部署到生产环境前，预览模式用于最终测试和验证。它还确保工作流在类似于生产的设置中平稳运行。

先决条件

- 在集群中安装了 OpenShift Serverless Logic Operator。
- 您可以使用适当的角色和权限访问 OpenShift Serverless Logic 项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 已安装 OpenShift CLI **(oc)**。

要以 Preview 模式部署工作流，OpenShift Serverless Logic Operator 使用 OpenShift Container Platform 上的构建系统，这会自动创建用于部署工作流的镜像。

以下小节解释了如何使用带有 **SonataFlow** 自定义资源的 OpenShift Serverless Logic Operator 在集群中构建和部署您的工作流。

1.2.2.1. 在预览**模式中配置工作流**

1.2.2.1.1. **配置工作流基础构建器镜像**

如果您的场景需要严格策略用于镜像使用，如安全或强化约束，请替换 OpenShift Serverless Logic Operator 用来构建最终工作流容器镜像的默认镜像。

默认情况下，OpenShift Serverless Logic Operator 使用官方 Red Hat Registry 中分发的镜像来构建工作流。如果您的场景需要严格的策略用于镜像，如 security 或 hardening 约束，您可以替换默认镜像。

要更改此镜像，您可以编辑部署工作流的命名空间中 **SonataFlowPlatform** 自定义资源(CR)。

先决条件

- 在集群中安装了 OpenShift Serverless Logic Operator。
- 您可以使用适当的角色和权限访问 OpenShift Serverless Logic 项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 已安装 OpenShift CLI **(oc)**。

流程

1. 运行以下命令，列出命名空间中的 **SonataFlowPlatform** 资源：

```
$ oc get sonataflowplatform -n <your_namespace>
```
(1)

(1) 将 **<your_namespace** > 替换为命名空间的名称。

2. 运行以下命令，使用新构建器镜像修补 **SonataFlowPlatform** 资源：

```
$ oc patch sonataflowplatform <name> --patch 'spec:\n  build:\n    config:\n      baseImage:
<your_new_image_full_name_with_tag>' -n <your_namespace>
```

验证

1. 运行以下命令，验证 **SonataFlowPlatform** CR 是否已正确修补：

```
$ oc describe sonataflowplatform <name> -n <your_namespace>
```
(1)

(1) 将 **<name>** 替换为 **SonataFlowPlatform** 资源的名称，**将 <your_namespace>** 替换为命名空间的名称。

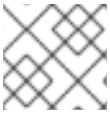
确保 **spec.build.config** 下的 **baseImage** 字段反映了新镜像。

1.2.2.1.2. 自定义基础构建器 Dockerfile

OpenShift Serverless Logic Operator 使用 **openshift-serverless-logic**OpenShift Serverless Logic Operator 安装命名空间中的 **logic-operator-rhel8-builder-config** 配置映射自定义资源(CR)来配置和运行工作流构建过程。您可以更改此配置映射中的 Dockerfile 条目，以根据您的需要调整 Dockerfile。

重要

修改 Dockerfile 可能会破坏构建过程。



注意

这个示例仅供参考。实际版本可能稍有不同。不要在您的安装中使用这个示例。

logic-operator-rhel8-builder-config 配置映射 CR 示例

```
apiVersion: v1
data:
  DEFAULT_WORKFLOW_EXTENSION: .sw.json
  Dockerfile: |
    FROM registry.redhat.io/openshift-serverless-1/logic-swf-builder-rhel8:1.33.0 AS builder

    # Variables that can be overridden by the builder
    # To add a Quarkus extension to your application
    ARG QUARKUS_EXTENSIONS
    # Args to pass to the Quarkus CLI add extension command
    ARG QUARKUS_ADD_EXTENSION_ARGS
    # Additional java/mvn arguments to pass to the builder
    ARG MAVEN_ARGS_APPEND

    # Copy from build context to skeleton resources project
    COPY --chown=1001 . ./resources

    RUN /home/kogito/launch/build-app.sh ./resources

    #============================
```

```
    # Runtime Run
    #============================
    FROM registry.access.redhat.com/ubi9/openjdk-17:latest

    ENV LANG='en_US.UTF-8' LANGUAGE='en_US:en'

    # We make four distinct layers so if there are application changes, the library layers can be re-used
    COPY --from=builder --chown=185 /home/kogito/serverless-workflow-project/target/quarkus-app/lib/ /deployments/lib/
    COPY --from=builder --chown=185 /home/kogito/serverless-workflow-project/target/quarkus-app/*.jar /deployments/
    COPY --from=builder --chown=185 /home/kogito/serverless-workflow-project/target/quarkus-app/app/ /deployments/app/
    COPY --from=builder --chown=185 /home/kogito/serverless-workflow-project/target/quarkus-app/quarkus/ /deployments/quarkus/

    EXPOSE 8080
    USER 185
    ENV AB_JOLOKIA_OFF=""
    ENV JAVA_OPTS="-Dquarkus.http.host=0.0.0.0 -Djava.util.logging.manager=org.jboss.logmanager.LogManager"
    ENV JAVA_APP_JAR="/deployments/quarkus-run.jar"
kind: ConfigMap
metadata:
  name: sonataflow-operator-builder-config
  namespace: sonataflow-operator-system
```

1.2.2.1.3. 更改资源要求

您可以通过在工作流命名空间中创建或编辑 **SonataFlowPlatform** 资源来指定内部构建器 pod 的资源要求。

SonataFlowPlatform 资源示例

```
apiVersion: sonataflow.org/v1alpha08
kind: SonataFlowPlatform
metadata:
  name: sonataflow-platform
spec:
  build:
    template:
      resources:
        requests:
          memory: "64Mi"
          cpu: "250m"
        limits:
          memory: "128Mi"
          cpu: "500m"
```

注意

每个命名空间只允许一个 **SonataFlowPlatform** 资源。获取并编辑 OpenShift Serverless Logic Operator 为您创建的资源，而不是尝试创建另一个资源。

您可以微调特定工作流的资源要求。每个工作流实例都有一个 **SonataFlowBuild** 实例，其名称与工作流相同。您可以编辑 **SonataFlowBuild** 自定义资源(CR)并指定参数，如下所示：

SonataFlowBuild CR 示例

```
apiVersion: sonataflow.org/v1alpha08
kind: SonataFlowBuild
metadata:
  name: my-workflow
spec:
  resources:
    requests:
      memory: "64Mi"
      cpu: "250m"
    limits:
      memory: "128Mi"
      cpu: "500m"
```

这些参数仅适用于新的构建实例。

1.2.2.1.4. 将参数传递给内部构建器

您可以通过将构建参数传递给 **SonataFlowBuild** 实例，或者在 **SonataFlowPlatform** 资源中设置默认构建参数来自定义构建流程。

先决条件

- 在集群中安装了 OpenShift Serverless Logic Operator。
- 您可以使用适当的角色和权限访问 OpenShift Serverless Logic 项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 已安装 OpenShift CLI **(oc)**。

流程

1. 运行以下命令，检查现有的 **SonataFlowBuild** 实例：

```
$ oc get sonataflowbuild <name> -n <namespace>
```
(1)

(1) 将 **<name** > 替换为 **SonataFlowBuild** 实例的名称，将 &lt **;namespace&** gt; 替换为您的命名空间。

2. 运行以下命令，在 **SonataFlowBuild** 实例中添加构建参数：

```
$ oc edit sonataflowbuild <name> -n <namespace>
```

3. 在 **SonataFlowBuild** 实例的 **.spec.buildArgs** 字段中添加所需的构建参数：

```
apiVersion: sonataflow.org/v1alpha08
kind: SonataFlowBuild
metadata:
  name: <name>
```
(1)

```
spec:
  buildArgs:
    - name: <argument_1>
      value: <value_1>
    - name: <argument_2>
      value: <value_2>
```

1 现有 **SonataFlowBuild** 实例的名称。

4. 保存文件并退出。
   将启动一个带有更新的配置的新构建。

5. 运行以下命令，在 **SonataFlowPlatform** 资源中设置默认构建参数：

```
$ oc edit sonataflowplatform <name> -n <namespace>
```

6. 在 **SonataFlowPlatform** 资源的 **.spec.buildArgs** 字段中添加所需的构建参数：

```
apiVersion: sonataflow.org/v1alpha08
kind: SonataFlowPlatform
metadata:
  name: <name> 1
spec:
  build:
    template:
      buildArgs:
        - name: <argument_1>
          value: <value_1>
        - name: <argument_2>
          value: <value_2>
```

1 现有 **SonataFlowPlatform** 资源的名称。

7. 保存文件并退出。

1.2.2.1.5. 在内部构建器中设置环境变量

您可以将环境变量设置为 **SonataFlowBuild** 内部构建器 pod。这些变量仅对构建上下文有效，且不会在最终构建的工作流镜像上设置。

先决条件

- 在集群中安装了 OpenShift Serverless Logic Operator。
- 您可以使用适当的角色和权限访问 OpenShift Serverless Logic 项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 已安装 OpenShift CLI **(oc)**。

流程

1. 运行以下命令，检查现有的 **SonataFlowBuild** 实例：

```
$ oc get sonataflowbuild <name> -n <namespace>
```

将 **<name** > 替换为 **SonataFlowBuild** 实例的名称，将 &lt **;namespace&** gt; 替换为您的命名空间。

2. 运行以下命令来编辑 **SonataFlowBuild** 实例：

```
$ oc edit sonataflowbuild <name> -n <namespace>
```

SonataFlowBuild 实**例示例**

```
apiVersion: sonataflow.org/v1alpha08
kind: SonataFlowBuild
metadata:
  name: <name>
spec:
  envs:
    - name: <env_variable_1>
      value: <value_1>
    - name: <env_variable_2>
      value: <value_2>
```

3. 保存文件并退出。
一个新的带有更新后的配置会启动。

或者，您可以在 **SonataFlowPlatform** 中设置 enviroments，以便每个新构建实例都会将其用作模板。

SonataFlowPlatform 实**例示例**

```
apiVersion: sonataflow.org/v1alpha08
kind: SonataFlowPlatform
metadata:
  name: <name>
spec:
  build:
    template:
      envs:
        - name: <env_variable_1>
          value: <value_1>
        - name: <env_variable_2>
          value: <value_2>
```

1.2.2.1.6. 更改基本构建器镜像

您可以通过编辑 **logic-operator-rhel8-builder-config** 配置映射来修改 OpenShift Serverless Logic Operator 使用的默认构建器镜像。

先决条件

- 在集群中安装了 OpenShift Serverless Logic Operator。
- 您可以使用适当的角色和权限访问 OpenShift Serverless Logic 项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

- 已安装 OpenShift CLI **(oc)**。

流程

1. 运行以下命令来编辑 **logic-operator-rhel8-builder-config** 配置映射：

```
$ oc edit cm/logic-operator-rhel8-builder-config -n openshift-serverless-logic
```

2. 修改 dockerfile 条目。
在编辑器中，找到 Dockerfile 条目并将第一行更改为所需的镜像。

Example

```
data:
  Dockerfile: |
    FROM registry.redhat.io/openshift-serverless-1/logic-swf-builder-rhel8:1.33.0
    # Change the image to the desired one
```

3. 保存更改。

1.2.2.2. 构建和部署您的工作流

您可以在 OpenShift Container Platform 和 OpenShift Serverless Logic Operator 上创建 **SonataFlow** 自定义资源(CR)并部署工作流。

先决条件

- 在集群中安装了 OpenShift Serverless Logic Operator。
- 您可以使用适当的角色和权限访问 OpenShift Serverless Logic 项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 已安装 OpenShift CLI **(oc)**。

流程

1. 创建类似以下示例的工作流 YAML 文件：

```
apiVersion: sonataflow.org/v1alpha08
kind: SonataFlow
metadata:
  name: greeting
  annotations:
    sonataflow.org/description: Greeting example on k8s!
    sonataflow.org/version: 0.0.1
spec:
  flow:
    start: ChooseOnLanguage
    functions:
      - name: greetFunction
        type: custom
        operation: sysout
    states:
      - name: ChooseOnLanguage
```

```
    type: switch
    dataConditions:
      - condition: "${ .language == \"English\" }"
        transition: GreetInEnglish
      - condition: "${ .language == \"Spanish\" }"
        transition: GreetInSpanish
    defaultCondition: GreetInEnglish
  - name: GreetInEnglish
    type: inject
    data:
      greeting: "Hello from JSON Workflow, "
    transition: GreetPerson
  - name: GreetInSpanish
    type: inject
    data:
      greeting: "Saludos desde JSON Workflow, "
    transition: GreetPerson
  - name: GreetPerson
    type: operation
    actions:
      - name: greetAction
        functionRef:
          refName: greetFunction
          arguments:
            message:  ".greeting+.name"
    end: true
```

2. 运行以下命令，将 **SonataFlow** 工作流定义应用到 OpenShift Container Platform 命名空间：

```
$ oc apply -f <workflow-name>.yaml -n <your_namespace>
```

greetings-workflow.yaml 文件的命令示例：

```
$ oc apply -f greetings-workflow.yaml -n workflows
```

3. 运行以下命令列出所有构建配置：

```
$ oc get buildconfigs -n workflows
```

4. 运行以下命令，获取构建过程的日志：

```
$ oc logs buildconfig/<workflow-name> -n <your_namespace>
```

greetings-workflow.yaml 文件的命令示例：

```
$ oc logs buildconfig/greeting -n workflows
```

验证

1. 要验证部署，请运行以下命令列出所有 pod：

```
$ oc get pods -n <your_namespace>
```

确保与工作流对应的 pod 正在运行。

2. 运行以下命令，检查正在运行的 pod 及其日志：

```
$ oc logs pod/<pod-name> -n workflows
```

1.2.2.3. 验证工作流部署

您可以通过从工作流 pod 执行测试 HTTP 调用来验证 OpenShift Serverless Logic 工作流是否正在运行。

先决条件

- 在集群中安装了 OpenShift Serverless Logic Operator。
- 您可以使用适当的角色和权限访问 OpenShift Serverless Logic 项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 已安装 OpenShift CLI **(oc)**。

流程

1. 创建类似以下示例的工作流 **YAML** 文件：

```
apiVersion: sonataflow.org/v1alpha08
kind: SonataFlow
metadata:
  name: greeting
  annotations:
    sonataflow.org/description: Greeting example on k8s!
    sonataflow.org/version: 0.0.1
spec:
  flow:
    start: ChooseOnLanguage
    functions:
      - name: greetFunction
        type: custom
        operation: sysout
    states:
      - name: ChooseOnLanguage
        type: switch
        dataConditions:
          - condition: "${ .language == \"English\" }"
            transition: GreetInEnglish
          - condition: "${ .language == \"Spanish\" }"
            transition: GreetInSpanish
        defaultCondition: GreetInEnglish
      - name: GreetInEnglish
        type: inject
        data:
          greeting: "Hello from JSON Workflow, "
        transition: GreetPerson
      - name: GreetInSpanish
        type: inject
        data:
          greeting: "Saludos desde JSON Workflow, "
```

```
      transition: GreetPerson
    - name: GreetPerson
      type: operation
      actions:
        - name: greetAction
          functionRef:
            refName: greetFunction
            arguments:
              message: ".greeting+.name"
      end: true
```

2. 运行以下命令，为工作流服务创建路由：

```
$ oc expose svc/<workflow-service-name> -n workflows
```

此命令创建用于访问工作流服务的公共 URL。

3. 运行以下命令，为公共 URL 设置环境变量：

```
$ WORKFLOW_SVC=$(oc get route/<workflow-service-name> -n <namespace> --
template='{{.spec.host}}')
```

4. 运行以下命令，向工作流发出 HTTP 调用，以将 POST 请求发送到服务：

```
$ curl -X POST -H 'Content-Type: application/json' -H 'Accept: application/json' -d '{<"your":
"json_payload">}' http://$WORKFLOW_SVC/<endpoint>
```

输出示例

```
{
  "id": "b5fbfaa3-b125-4e6c-9311-fe5a3577efdd",
  "workflowdata": {
    "name": "John",
    "language": "English",
    "greeting": "Hello from JSON Workflow, "
  }
}
```

此输出显示工作流正在运行时的预期响应示例。

1.2.2.4. 重启构建

要重启构建，您可以在 **SonataFlowBuild** 实例中添加或编辑 **sonataflow.org/restartBuild: true** 注解。如果您的工作流或初始构建版本存在问题，则需要重启构建。

先决条件

- 在集群中安装了 OpenShift Serverless Logic Operator。
- 您可以使用适当的角色和权限访问 OpenShift Serverless Logic 项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 已安装 OpenShift CLI **(oc)**。

流程

1. 运行以下命令，检查 **SonataFlowBuild** 实例是否存在：

```
$ oc get sonataflowbuild <name> -n <namespace>
```

2. 运行以下命令来编辑 **SonataFlowBuild** 实例：

```
$ oc edit sonataflowbuild/<name> -n <namespace>
```

将 **<name** > 替换为 **SonataFlowBuild** 实例的名称，将 **<** namespace> 替换为部署了工作流的命名空间。

3. 添加 **sonataflow.org/restartBuild: true** 注解来重新启动构建。

```
apiVersion: sonataflow.org/v1alpha08
kind: SonataFlowBuild
metadata:
  name: <name>
  annotations:
    sonataflow.org/restartBuild: true
```

此操作会触发 OpenShift Serverless Logic Operator 来启动工作流的新构建。

4. 要监控构建过程，请运行以下命令来检查构建日志：

```
$ oc logs buildconfig/<name> -n <namespace>
```

将 **<name** > 替换为 **SonataFlowBuild** 实例的名称，将 **<** namespace> 替换为部署了工作流的命名空间。

1.2.3. 编辑工作流

当 OpenShift Serverless Logic Operator 部署工作流服务时，它会创建两个配置映射来存储运行时属性：

- 用户属性：在 **ConfigMap** 中定义，以带有后缀 的 **SonataFlow** 对象命名。**例如，如果您的工作流名称是 greeting，**则 **ConfigMap 名称**为 **greeting-props。**

- **受管属性：在 ConfigMap 中定义，以带有 suffix -managed-props 的 SonataFlow 对象命名。例如，如果您的工作流名称是 greeting，**则 **ConfigMap 名称**为 **greeting-managed-props。**

注意

受管属性始终覆盖具有相同密钥名称的任何 user 属性，用户无法编辑。在下一个协调周期中，Operator 都会覆盖任何更改。

先决条件

- 已在集群中安装了 OpenShift Serverless Logic Operator。
- 您可以使用适当的角色和权限访问 OpenShift Serverless Logic 项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 已安装 OpenShift CLI (oc)。

流程

1. 运行以下命令打开并编辑 ConfigMap ：

```
$ oc edit cm <workflow_name>-props -n <namespace>
```

将 <workflow_name > 替换为工作流的名称，将 <namespace> 替换为部署工作流的命名空间。

2. 在 application.properties 部分中添加属性。

存储在 ConfigMap 中的工作流属性示例：

```
apiVersion: v1
kind: ConfigMap
metadata:
  labels:
    app: greeting
  name: greeting-props
  namespace: default
data:
  application.properties: |
    my.properties.key = any-value
```

确保正确格式化属性，以防止 Operator 将配置替换为默认配置。

3. 进行必要的更改后，保存文件并退出编辑器。

1.2.4. 测试工作流

要验证 OpenShift Serverless Logic 工作流是否在正确运行，您可以从相关 pod 执行测试 HTTP 调用。

先决条件

- 已在集群中安装了 OpenShift Serverless Logic Operator。
- 您可以使用适当的角色和权限访问 OpenShift Serverless Logic 项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 已安装 OpenShift CLI (oc)。

流程

1. 运行以下命令，为命名空间中的指定服务创建路由：

```
$ oc expose svc <service_name> -n <namespace>
```

2. 运行以下命令，获取新公开的服务的 URL：

```
$ WORKFLOW_SVC=$(oc get route/<service_name> --template='{{.spec.host}}')
```

3. 运行以下命令，执行测试 HTTP 调用并发送 POST 请求：

```
$ curl -X POST -H 'Content-Type:application/json' -H 'Accept:application/json' -d
'<request_body>' http://$WORKFLOW_SVC/<endpoint>
```

4. 验证响应，以确保工作流按预期工作。

1.2.5. 工作流故障排除

OpenShift Serverless Logic Operator 使用健康检查探测部署其 pod，以确保工作流以健康状态运行。如果更改导致这些健康检查失败，pod 将停止响应。

先决条件

- 已在集群中安装了 OpenShift Serverless Logic Operator。
- 您可以使用适当的角色和权限访问 OpenShift Serverless Logic 项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 已安装 OpenShift CLI (oc)。

流程

1. 运行以下命令检查工作流状态：

```
$ oc get workflow <name> -o jsonpath={.status.conditions} | jq .
```

2. 要从工作流的部署中获取和分析日志，请运行以下命令：

```
$ oc logs deployment/<workflow_name> -f
```

1.2.6. 删除工作流

您可以使用 oc delete 命令删除当前目录中的 OpenShift Serverless Logic 工作流。

先决条件

- 已在集群中安装了 OpenShift Serverless Logic Operator。
- 您可以使用适当的角色和权限访问 OpenShift Serverless Logic 项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 已安装 OpenShift CLI (oc)。

流程

流程

1. 验证您具有定义您要删除的工作流的正确文件。例如，workflow.yaml。

2. 运行 oc delete 命令从指定的命名空间中删除工作流：

```
$ oc delete -f <your_file> -n <your_namespace>
```

将 <your_file > 替换为工作流文件的名称，将 &lt ;your_namespace& gt; 替换为您的命名空间。

第 2 章 管理服务

2.1. 配置 OPENAPI 服务

OpenAPI 规格 (OAS)为 HTTP API 定义了一个标准的编程语言无关接口。您可以在不访问源代码、额外文档或网络流量检查的情况下了解服务的功能。当使用 OpenAPI 定义服务时，您可以使用最小实施逻辑理解并与之交互。正如接口描述简化了低级别编程一样，*OpenAPI 规格* 消除了调用服务中的猜测工作。

2.1.1. OpenAPI 功能定义

OpenShift Serverless Logic 允许工作流使用函数中的 OpenAPI 规格引用与远程服务交互。

OpenAPI 功能定义示例

```
{
  "functions": [
    {
      "name": "myFunction1",
      "operation": "classpath:/myopenapi-file.yaml#myFunction1"
    }
  ]
}
```

operation 属性是由以下参数组成的 字符串 ：

- URI ：引擎使用它来定位规范文件，如 classpath。
- 操作标识符 ：您可以在 OpenAPI 规格文件中找到此标识符。

OpenShift Serverless Logic 支持以下 URI 方案：

- classpath ：将它用于位于应用程序项目的 src/main/resources 文件夹中的文件。classpath 是默认的 URI 方案。如果没有定义 URI 方案，则文件位置为

src/main/resources/myopenapifile.yaml。

- file：将它用于位于文件系统中的文件。
- HTTP 或 https ：将它们用于远程找到的文件。

确保 OpenAPI 规格文件在构建期间可用。OpenShift Serverless Logic 使用内部代码生成功能在运行时发送请求。构建应用程序镜像后，OpenShift Serverless Logic 将无法访问这些文件。

如果要添加到工作流的 OpenAPI 服务没有规格文件，您可以创建一个或多个服务来生成和公开该文件。

2.1.2. 根据 OpenAPI 规格发送 REST 请求

要发送基于 OpenAPI 规格文件的 REST 请求，您必须执行以下步骤：

- 定义功能引用
- 访问工作流状态中定义的功能

先决条件

- 已在集群中安装了 OpenShift Serverless Logic Operator。
- 您可以使用适当的角色和权限访问 OpenShift Serverless Logic 项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 您可以访问 OpenAPI 规格文件。

流程

1. 定义 OpenAPI 功能：

a. 识别并访问您要调用的服务的 OpenAPI 规格文件。

b. 将 OpenAPI 规格文件复制到工作流服务目录中，如 src/main/resources/specs。

以下示例显示了 multiplication REST 服务的 OpenAPI 规格：

multiplication REST 服务 OpenAPI 规格示例

```
openapi: 3.0.3
info:
  title: Generated API
  version: "1.0"
paths:
  /:
    post:
      operationId: doOperation
      parameters:
        - in: header
          name: notUsed
          schema:
            type: string
          required: false
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/MultiplicationOperation'
      responses:
        "200":
          description: OK
          content:
            application/json:
              schema:
                type: object
                properties:
                  product:
                    format: float
                    type: number
components:
  schemas:
    MultiplicationOperation:
      type: object
      properties:
        leftElement:
          format: float
          type: number
```

```
      rightElement:
        format: float
        type: number
```

c. 要在工作流中定义功能，请使用 OpenAPI 规格中的 operationId 来引用功能定义中的所需操作。

温度转换应用中的函数定义示例

```
{
  "functions": [
    {
      "name": "multiplication",
      "operation": "specs/multiplication.yaml#doOperation"
    },
    {
      "name": "subtraction",
      "operation": "specs/subtraction.yaml#doOperation"
    }
  ]
}
```

d. 确保您的功能定义引用存储在 src/main/resources/specs 目录中的 OpenAPI 文件的正确路径。

2. 访问工作流状态中定义的功能：

a. 定义工作流操作来调用您添加的功能定义。确保每个操作引用之前定义的函数。

b. 使用 functionRef 属性根据其名称引用特定功能。使用 OpenAPI 规格中定义的参数来映射 functionRef 中的参数。

以下示例演示了在请求路径而不是请求正文中映射参数，您可以参考以下 PetStore API 示例：

工作流中的映射功能参数示例

```
{
  "states": [
   {
    "name": "SetConstants",
    "type": "inject",
    "data": {
     "subtractValue": 32.0,
     "multiplyValue": 0.5556
    },
    "transition": "Computation"
   },
   {
    "name": "Computation",
    "actionMode": "sequential",
    "type": "operation",
    "actions": [
     {
      "name": "subtract",
      "functionRef": {
       "refName": "subtraction",
       "arguments": {
        "leftElement": ".fahrenheit",
        "rightElement": ".subtractValue"
       }
      }
     },
     {
      "name": "multiply",
      "functionRef": {
       "refName": "multiplication",
       "arguments": {
         "leftElement": ".difference",
         "rightElement": ".multiplyValue"
       }
      }
     }
    ],
    "end": {
     "terminate": true
    }
   }
  ]
}
```

c.

检查 **OpenAPI** 规格的 **Operation Object** 部分，以了解如何在请求中结构参数。

d.

使用 jq 表达式从有效负载中提取数据并将其映射到所需参数。确保引擎根据 OpenAPI 规格映射参数名称。

e.

对于在请求路径而不是正文中需要参数的操作，请参阅 OpenAPI 规格中的参数定义。

有关请求路径而不是请求正文中映射参数的更多信息，您可以参阅以下 PetStore API 示例：

映射路径参数示例

```
{
  "/pet/{petId}": {
    "get": {
      "tags": ["pet"],
      "summary": "Find pet by ID",
      "description": "Returns a single pet",
      "operationId": "getPetById",
      "parameters": [
        {
          "name": "petId",
          "in": "path",
          "description": "ID of pet to return",
          "required": true,
          "schema": {
            "type": "integer",
            "format": "int64"
          }
        }
      ]
    }
  }
}
```

以下是调用函数的示例，在请求路径中只添加了一个名为 petId 的参数：

调用 PetStore 功能的示例

```
{
  "name": "CallPetStore", 1
```

```
  "actionMode": "sequential",
  "type": "operation",
  "actions": [
   {
    "name": "getPet",
    "functionRef": {
     "refName": "getPetById", 2
     "arguments": { 3
      "petId": ".petId"
     }
    }
   }
  ]
}
```

1

状态定义，如 CallPetStore。

2

功能定义参考。在上例中，函数定义 getPetById 用于 PetStore OpenAPI 规格。

3

参数定义。OpenShift Serverless Logic 在发送请求前将参数 petId 添加到请求路径中。

2.1.3. 配置 OpenAPI 服务的端点 URL

在访问工作流状态中的函数定义后，您可以配置 OpenAPI 服务的端点 URL。

先决条件

- 您可以使用适当的角色和权限访问 OpenShift Serverless Logic 项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 您已创建了 OpenShift Serverless Logic 项目。

- 您可以访问 OpenAPI 规格文件。
- 您已在工作流中定义了函数定义。
- 您可以访问工作流状态中定义的功能。

流程

1. 找到您要配置的 OpenAPI 规格文件。例如，substraction.yaml。
2. 通过将特殊字符（如 .）替换为下划线并将字母转换为小写，将文件名转换为有效的配置键。例如，将 substraction.yaml 更改为 substraction_yaml。
3. 要定义配置密钥，请使用转换的文件名作为 REST 客户端配置密钥。将此键设置为环境变量，如下例所示：

```
quarkus.rest-client.subtraction_yaml.url=http://myserver.com
```

4. 要防止 application.properties 文件中的硬编码 URL，请使用环境变量替换，如下例所示：

```
quarkus.rest-client.subtraction_yaml.url=${SUBTRACTION_URL:http://myserver.com}
```

在本例中：

- 配置密钥： quarkus.rest-client.subtraction_yaml.url
- 环境变量：SUBTRACTION_URL
- 回退 URL: http://myserver.com

5. 确保在系统或部署环境中设置了 (SUBTRACTION_URL) 环境变量。如果没有找到变量，应用程序将使用回退 URL (http://myserver.com)。

6. 将配置键和 URL 替换添加到 application.properties 文件中：

```
quarkus.rest-client.subtraction_yaml.url=${SUBTRACTION_URL:http://myserver.com}
```

7. 部署或重启您的应用程序以应用新的配置设置。

2.2. 配置 OPENAPI 服务端点

OpenShift Serverless Logic 使用 kogito.sw.operationIdStrategy 属性来生成 REST 客户端，用于调用 OpenAPI 文档中定义的服务。此属性决定了如何为 REST 客户端配置派生配置密钥。

kogito.sw.operationIdStrategy 属性支持以下值： FILE_NAME、FULL_URI、FUNCTION_NAME 和 SPEC_TITLE。

FILE_NAME

OpenShift Serverless Logic 使用 OpenAPI 文档文件名来创建配置密钥。键基于文件名，其中将特殊字符替换为下划线。

配置示例：

```
quarkus.rest-client.stock_portfolio_svc_yaml.url=http://localhost:8282/ (1)
```

(1) OpenAPI File Path 是 src/main/resources/openapi/stock-portfolio-svc.yaml。为 REST 客户端配置 URL 生成的密钥是 stock_portfolio_svc_yaml

FULL_URI

OpenShift Serverless Logic 使用 OpenAPI 文档的完整 URI 路径作为配置密钥。整个 URI 被清理以组成密钥。

Serverless 工作流示例

```
{
  "id": "myworkflow",
  "functions": [
    {
      "name": "myfunction",
      "operation": "https://my.remote.host/apicatalog/apis/123/document"
    }
  ]
  ...
}
```

配置示例：

```
quarkus.rest-client.apicatalog_apis_123_document.url=http://localhost:8282/ 1
```

1

URI 路径为 https://my.remote.host/apicatalog/apis/123/document。为 REST 客户端配置 URL 生成的密钥是 apicatalog_apis_123_document。

FUNCTION_NAME

OpenShift Serverless Logic 组合了工作流 ID 和引用 OpenAPI 文档的功能名称来生成配置密钥。

Serverless 工作流示例

```
{
  "id": "myworkflow",
  "functions": [
    {
      "name": "myfunction",
      "operation": "https://my.remote.host/apicatalog/apis/123/document"
    }
```

```
    ]
    ...
}
```

配置示例：

```
quarkus.rest-client.myworkflow_myfunction.url=http://localhost:8282/ 1
```

1

工作流 ID 是 myworkflow。函数名称是 myfunction。为 REST 客户端配置 URL 生成的密钥是 myworkflow_myfunction。

SPEC_TITLE

OpenShift Serverless Logic 使用 OpenAPI 文档中的 info.title 值来创建配置键。标题被清理为组成密钥。

OpenAPI 文档示例

```
openapi: 3.0.3
info:
  title: stock-service API
  version: 2.0.0-SNAPSHOT
paths:
  /stock-price/{symbol}:
...
```

配置示例：

```
quarkus.rest-client.stock-service_API.url=http://localhost:8282/ 1
```

1

OpenAPI 文档标题为 stock-service API。为 REST 客户端配置 URL 生成的密钥是 stock-service_API。

2.2.1. 使用 URI 别名

作为 kogito.sw.operationIdStrategy 属性的替代选择，您可以使用 workflow-uri-definitions 自定义扩展为 URI 分配别名。此别名简化了配置过程，并可用作 REST 客户端设置和功能定义中的配置密钥。

workflow-uri-definitions 扩展允许您将 URI 映射到别名，您可以在整个工作流和配置文件中引用。这种方法提供了一种集中管理 URI 及其配置的集中方法。

先决条件

- 您可以使用适当的角色和权限访问 OpenShift Serverless Logic 项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 您可以访问 OpenAPI 规格文件。

流程

1. 在您的工作流中添加 workflow-uri-definitions 扩展。在此扩展中，为您的 URI 创建别名。

工作流示例

```
{
 "extensions": [
  {
   "extensionid": "workflow-uri-definitions", 1
   "definitions": {
    "remoteCatalog": "https://my.remote.host/apicatalog/apis/123/document" 2
   }
  }
 ],
```

```
  "functions": [ 3
   {
    "name": "operation1",
    "operation": "remoteCatalog#operation1"
   },
   {
    "name": "operation2",
    "operation": "remoteCatalog#operation2"
   }
  ]
}
```

1

将扩展 ID 设置为 workflow-uri-definitions。

2

通过将 remoteCatalog 别名映射到 URI 来设置别名定义，例如 https://my.remote.host/apicatalog/apis/123/document URI。

3

使用带有操作标识符的 remoteCatalog 别名来设置功能操作，如 operation1 和 operation2 操作标识符。

1. 在 application.properties 文件中，使用工作流中定义的别名配置 REST 客户端。

属性示例

```
quarkus.rest-client.remoteCatalog.url=http://localhost:8282/
```

在上例中，配置键被设置为 quarkus.rest-client.remoteCatalog.url，URL 设置为 http://localhost:8282/，REST 客户端通过引用 remoteCatalog 别名来使用它。

2. 在工作流中，在定义在 URI 上操作的功能时使用别名。

工作流示例（续）：

```
{
  "functions": [
    {
      "name": "operation1",
      "operation": "remoteCatalog#operation1"
    },
    {
      "name": "operation2",
      "operation": "remoteCatalog#operation2"
    }
  ]
}
```

2.3. 故障排除服务

对基于 HTTP 的功能调用（如使用 OpenAPI 功能的用户）进行有效的故障排除对于维护工作流编配至关重要。

要诊断问题，您可以跟踪 HTTP 请求和响应。

2.3.1. 追踪 HTTP 请求和响应

OpenShift Serverless Logic 使用 Apache HTTP 客户端来跟踪 HTTP 请求和响应。

先决条件

- 您可以使用适当的角色和权限访问 OpenShift Serverless Logic 项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 您可以访问 OpenAPI 规格文件。
- 您可以访问工作流定义和实例 ID，以更正 HTTP 请求和响应。

- 您可以访问发生 HTTP 服务调用的应用的日志配置

流程

1. 要跟踪 HTTP 请求和响应，OpenShift Serverless Logic 通过设置以下属性来使用 Apache HTTP 客户端：

```
# Turning HTTP tracing on
quarkus.log.category."org.apache.http".level=DEBUG
```

2. 在应用程序的 application.properties 文件中添加以下配置，以打开 Apache HTTP 客户端的调试：

```
quarkus.log.category."org.apache.http".level=DEBUG
```

3. 重启应用程序以传播日志配置更改。

4. 重启后，检查 HTTP 请求追踪的日志。

追踪的 HTTP 请求的日志示例

```
2023-09-25 19:00:55,242 DEBUG Executing request POST /v2/models/yolo-model/infer
HTTP/1.1
2023-09-25 19:00:55,243 DEBUG http-outgoing-0 >> POST /v2/models/yolo-model/infer
HTTP/1.1
2023-09-25 19:00:55,243 DEBUG http-outgoing-0 >> Accept: application/json
2023-09-25 19:00:55,243 DEBUG http-outgoing-0 >> Content-Type: application/json
2023-09-25 19:00:55,243 DEBUG http-outgoing-0 >> kogitoprocid: inferencepipeline
2023-09-25 19:00:55,243 DEBUG http-outgoing-0 >> kogitoprocinstanceid: 85114b2d-
9f64-496a-bf1d-d3a0760cde8e
2023-09-25 19:00:55,243 DEBUG http-outgoing-0 >> kogitoprocist: Active
2023-09-25 19:00:55,243 DEBUG http-outgoing-0 >> kogitoproctype: SW
2023-09-25 19:00:55,243 DEBUG http-outgoing-0 >> kogitoprocversion: 1.0
2023-09-25 19:00:55,243 DEBUG http-outgoing-0 >> Content-Length: 23177723
2023-09-25 19:00:55,244 DEBUG http-outgoing-0 >> Host: yolo-model-opendatahub-
model.apps.trustyai.dzzt.p1.openshiftapps.com
```

-

5. 在请求日志后检查 **HTTP** 响应追踪的日志。

追踪的 HTTP 响应的日志示例

```
2023-09-25 19:01:00,738 DEBUG http-outgoing-0 << "HTTP/1.1 500 Internal Server
Error[\r][\n]"
2023-09-25 19:01:00,738 DEBUG http-outgoing-0 << "content-type: application/json[\r]
[\n]"
2023-09-25 19:01:00,738 DEBUG http-outgoing-0 << "date: Mon, 25 Sep 2023 19:01:00
GMT[\r][\n]"
2023-09-25 19:01:00,738 DEBUG http-outgoing-0 << "content-length: 186[\r][\n]"
2023-09-25 19:01:00,738 DEBUG http-outgoing-0 << "set-cookie:
276e4597d7fcb3b2cba7b5f037eeacf5=5427fafade21f8e7a4ee1fa6c221cf40; path=/;
HttpOnly; Secure; SameSite=None[\r][\n]"
2023-09-25 19:01:00,738 DEBUG http-outgoing-0 << "[\r][\n]"
2023-09-25 19:01:00,738 DEBUG http-outgoing-0 << "{"code":13, "message":"Failed to
load Model due to adapter error: Error calling stat on model file: stat /models/yolo-
model__isvc-1295fd6ba9/yolov5s-seg.onnx: no such file or directory"}"
```