**1) What is folder structure of camel –quarkus project?**

Below is folder structure of camel –quarkus project

Project Folder

src

main

docker

all-docker files

java

all-java files under packages

resources

application.properties

META-INF

resources

index.html

routes

camel-rout.xml

target

pom.xml

**2) How can we expose rest service in camel –quarkus?**

* Quarkus REST is a new Jakarta REST (formerly known as JAX-RS) implementation written from the ground up to work on our common Vert.x layer and is thus fully reactive, while also being very tightly integrated with Quarkus and consequently moving a lot of work to build time.
* We need to use below dependency to used rest API
  + quarkus-resteasy-reactive(rest API)
  + camel-quarkus-xml-io-dsl(quarkus to camel)
  + camel-quarkus-core(core jar)
  + camel-quarkus-log(logging)
  + camel-quarkus-direct(direct component)
* To expose rest service we can create Java class & use bellow class level annotation

@Path("/esb/cms/genericCMSPosting")

@Consumes(MediaType.APPLICATION\_JSON)

@Produces(MediaType.APPLICATION\_JSON)

* Create method with @GET or @POST annotation, from which we can send request to camel from route using ProducerTemplate. sendBodyAndHeaders() method as below

@Path("/esb/cms/genericCMSPosting")

@Consumes(MediaType.APPLICATION\_JSON)

@Produces(MediaType.APPLICATION\_JSON)

@Slf4j

public class Resource {

@Inject

ProducerTemplate clientProcessingTemplate;

@WithSpan

@POST

@Path("")

public Response getResponse(String apiRequest, @Context HttpHeaders httpHeaders, @Context Request request) {

Map<String, Object> headers = new HashMap<>();

headers.put("CamelHttpMethod", request.getMethod());

// Get all headers

MultivaluedMap<String, String> allHeaders = httpHeaders.getRequestHeaders();

for (Map.Entry<String, List<String>> entry : allHeaders.entrySet()){

headers.put(entry.getKey(), entry.getValue().get(0));

}

Object responseObject = null;

APIResponse apiResponse = new APIResponse();

try {

responseObject = clientProcessingTemplate.sendBodyAndHeaders("direct:genericcmsposting",

ExchangePattern.InOut, apiRequest, headers);

} catch (CamelExecutionException e) {

apiResponse.setReturnCode("400");

apiResponse.setResponseMessage(e.getMessage());

return Response.status(Response.Status.OK)

.entity(apiResponse).build();

}

return Response.status(Response.Status.OK)

.entity(responseObject).build();

}

}

**3) How can we use ActiveMQ in camel –quarkus?**

* We need to use below dependencies:
  + camel-quarkus-xml-io-dsl
  + camel-quarkus-core
  + camel-quarkus-activemq
  + org.messaginghub
* to use active MQ in camel-quarkus we need to declare active MQ configuration in application properties
  + camel.component.activemq.broker-url=failover:(tcp://10.15.15.76:61716,tcp://10.15.15.76:61716)?jms.rmIdFromConnectionId=true&maxReconnectAttempts=0
  + camel.component.activemq.username=admin
  + [camel.component.activemq.password=esbadmin@123](mailto:camel.component.activemq.password%3Desbadmin@123)
* after that we can use active MQ bean in route.xml file as below
  + <to pattern="InOnly" uri="activemq:queue:TransactionQueue" />

**4) How can we connect DB in camel –quarkus?**

* We can use 2 components to connect to DB, SQL & JDBC
* We need to use below dependencies:
  + camel-quarkus-sql
  + camel-quarkus-jdbc
  + driver jar
* To use SQL component in camel-quarkus we need to declare active MQ configuration in application properties
  + quarkus.datasource.esbDataSource.db-kind=mssql
  + quarkus.datasource.esbDataSource.username=a\_devcomdb\_prd
  + quarkus.datasource.esbDataSource.password=Sql@admin
  + quarkus.datasource.esbDataSource.jdbc.driver=com.microsoft.sqlserver.jdbc.SQLServerDriver
  + quarkus.datasource.esbDataSource.jdbc.url=jdbc:sqlserver://10.15.20.235;databaseName=ESB\_Transactions;integratedSecurity=false;encrypt=false;trustServerCertificate=true;
  + quarkus.datasource.esbDataSource.jdbc.max-size=16
* To use **SQL component** we can write code like below
  + <to uri="sql:select top 1 \* from TransactionsResponse" />
* To use **JDBC component** we can write code like below
  + <setBody><simple>select top 1 \* from TransactionsResponse</simple></setBody>
  + <to uri="jdbc:esbDataSource" />

**5) How can we call rest API using rest-client in camel –quarkus?**

* To call rest API using rest-client we need to use below jar
  + quarkus-rest-client-jackson
* Create interface which will call REST API as below, we can

**package** com.santosh.QuarkusCamelDemo.client;

**import** org.apache.camel.Body;

**import** org.eclipse.microprofile.rest.client.inject.RegisterRestClient;

**import** jakarta.inject.Named;

**import** jakarta.ws.rs.Consumes;

**import** jakarta.ws.rs.HeaderParam;

**import** jakarta.ws.rs.POST;

**import** jakarta.ws.rs.Produces;

**import** jakarta.ws.rs.core.Response;

@RegisterRestClient

**public** **interface** RestClient {

@POST

@Consumes("application/vnd.fisglobal-v1.0+json;charset=UTF-8")

@Produces("application/vnd.fisglobal-v1.0+json;charset=UTF-8")

**public** Response headerAPI(@Body String bodyPara, @HeaderParam("X-Auth-Token") String token);

}

* Here we have used below annotation to inject REST API URL

@RegisterRestClient

* We need to add rest URL in application properties as below.
* quarkus.rest-client."com.santosh.QuarkusCamelDemo.client.RestClient".uri= https://FIS/fis-rest-api-uat/app/rest/v1.0/account/addaccounthold
* Note that **com.santosh.QuarkusCamenDemo.client.RestClient** must match the fully qualified name of interface RestClient
* To call the rest interface we need to create Service class bean, in this class client interface is injected with @RestClient annotation

**package** com.santosh.QuarkusCamelDemo.service;

**import** org.eclipse.microprofile.rest.client.inject.RestClient;

**import** jakarta.enterprise.context.ApplicationScoped;

**import** jakarta.ws.rs.WebApplicationException;

**import** jakarta.ws.rs.core.Response;

@ApplicationScoped

**public** **class** RestClientServiceImpl **implements** RestClientService {

@RestClient

com.santosh.QuarkusCamelDemo.client.RestClient rs;

@Override

**public** Response callheaderAPI(String bodyPara, String token) {

**try** {

**return** rs.headerAPI(bodyPara, token);

} **catch** (WebApplicationException exception) {

**return** exception.getResponse();

} **catch** (Exception e) {

**return** exception.getResponse();

}

**return** **null**;

}

}

* create request process bean which will implement org.apache.camel.Processor interface, in which we can override process method & write our logic to process in-out message

**package** com.santosh.QuarkusCamelDemo;

**import** org.apache.camel.Exchange;

**import** org.apache.camel.Processor;

**import** com.santosh.QuarkusCamelDemo.service.RestClientService;

**import** jakarta.enterprise.context.ApplicationScoped;

**import** jakarta.inject.Inject;

**import** jakarta.inject.Named;

**import** jakarta.ws.rs.core.Response;

**import** lombok.extern.slf4j.Slf4j;

@ApplicationScoped

@Named("requestProcessor")

@Slf4j

**public** **class** RequestProcessor **implements** Processor{

@Inject

RestClientService clientRestService;

@Override

**public** **void** process(Exchange exchange) **throws** Exception {

Response callheaderAPI = clientRestService.callheaderAPI(exchange.getIn().getBody(String.**class**), exchange.getIn().getHeader("Token",String.**class**));

exchange.getIn().setBody(callheaderAPI.getEntity());

exchange.getIn().setHeader("CamelHttpResponseCode",callheaderAPI.getStatus());

}

}

* We can call HTTPS API by configuring SSL key-store path in below property
  + quarkus.rest-client."com.santosh.QuarkusCamelDemo.client.RestClient".trust-store= /home/104146@santosh.com/JKS/FIS.jks
  + quarkus.rest-client."com.santosh.QuarkusCamelDemo.client.RestClient".trust-store-password=123456
* We can disable SSL handshake by using below property
  + quarkus.rest-client."com.santosh.QuarkusCamelDemo.client.RestClient".verify-host=false

**4) How can we call Seda component in camel –quarkus?**

* To use SEDA component we need to use below maven dependencies
  + camel-quarkus-seda
* Create SEDA route as below

<route id=*"SMSCall"* streamCache=*"true"*>

<from uri=*"seda:SMSRoute?concurrentConsumers={{minConcurrentConsumers}}"* />

<log message=*"X-Correlation-Id\_${header.XCorrelationId} | UserId\_${header.RequestorId} | QuarkusCamelDemo QUARK\_STEP\_4 Seda Body before marshal\_${body}"* loggingLevel=*"INFO"* logName=*"com.fino.asynclog"* />

<bean ref=*"requestProcessor"* method=*"showHeaders"*></bean>

<log message=*"X-Correlation-Id\_${header.XCorrelationId} | UserId\_${header.RequestorId} | QuarkusCamelDemo QUARK\_STEP\_4 Seda Body after marshal\_${body}"* loggingLevel=*"INFO"* logName=*"com.fino.asynclog"* />

</route>

* call SEDA route as below from main route

<to uri=*"seda:SMSRoute?waitForTaskToComplete=Never"* />

**6) What is use of marshal & un-marshal object in camel –quarkus?How can we create it.**

* We need to add below dependencies for marshalling & unmarshalling object
  + camel-quarkus-jackson
* **Un-Marshaling** is used when there is requirement to convert JSON object into java object for processing & performing some operation on body.
* Suppose we have JSON string body & we want to do some change in that body, to do that we need to create process method & do change as below.
* In blueprint we can write code below

<setBody>

<simple>{"returnCode":"401", "responseMessage":"Full authentication is required to access this resource."}</simple>

</setBody>

<bean ref=*"requestProcessor"* method=*"checkBodyType"* />

<unmarshal ><json /></unmarshal>

<bean ref=*"requestProcessor"* method=*"checkBodyTypeAfter"* />

* In process method we can write code as below

**public** **void** checkBodyType(Exchange ex) {

System.***out***.println("body type"+ex.getIn().getBody().getClass());//body typeclass java.lang.String

}

**public** **void** checkBodyTypeAfter(Exchange ex) {

System.***out***.println("body type after "+ex.getIn().getBody().getClass());//body type after class java.util.LinkedHashMap

Map m=ex.getIn().getBody(Map.**class**);

System.***out***.println("body type after returnCode "+m.get("returnCode"));

m.put("returnCode", "5000");

}

* Marshaling is used to convert java object to JSON object, we can write below blueprint code to do that

<setBody><simple>{"returnCode":"401", "responseMessage":"Full authentication is required to access this resource."}</simple></setBody>

<unmarshal ><json /></unmarshal>

<bean ref=*"requestProcessor"* method=*"checkBodyTypeAfter"* />

<marshal ><json /></marshal>

<bean ref=*"requestProcessor"* method=*"checkBodyType"* />

* Here we have converted JSON object into Java Object → performed some processing on body → converted back Java Object into JSON object.

**7) How we can use File component in camel –quarkus?**

* File component is used to read file & write file.
* To use file component we need to used below dependencies
  + camel-quarkus-file
* To read file from specific location we need to use below code

<route id=*"fileRead"*>

<from uri=*"file:{{File\_InputPath}}?idempotent=true&amp;include=(?i).\*\.txt&amp;moveFailed=./failed"* />

<log message=*"FileDemo | Started Processing File Name: ${header.CamelFileName}"*></log>

<log message=*"FileDemo | Started Processing File Size: ${file:length}"*></log>

<log message=*"FileDemo | Started Processing File body ${body}"*></log>

</route>

* **We can use below option**
* **idempotent: Option to use the Idempotent Consumer EIP pattern to let Camel skip already processed files. Will by default use a memory based LRUCache that holds 1000 entries. If noop=true then idempotent will be enabled as well to avoid consuming the same files over and over again.**
* **Include: Is used to include files, if filename matches the regex pattern (matching is case in-sensitive). Notice if you use symbols such as plus sign and others you would need to configure this using the RAW() syntax if configuring this as an endpoint uri. See more details at configuring endpoint uris.**
* **MoveFailed: Sets the move failure expression based on Simple language. For example, to move files into a .failed subdirectory use: .failed. Note: When moving the files to the fail location Camel will handle the error and will not pick up the file again.**