**Understand Temporal-Workflow basics**

1. **Introduction: --** For understanding a workflow managed by the **Temporal-Server** we need to first understand what a workflow is. A workflow is a sequence of tasks which are performed in the same sequence in which they have been defined for a workflow, and the workflow will be considered as a completed one only when all its tasks are completed successfully.

A **temporal-Server** is a **workflow managing-engine** which executes all those tasks or activities which are already defined within a workflow. But, a question arises how **Temporal-Server** will come to know which workflow it has to execute and how to execute. For providing all these details to **Temporal-Server**, we use **temporal-api** while implementing a workflow in the code. A workflow implementation through the Temporal-Server can be done in a distributed environment.

In a micro-service based architecture, different tasks or activities of a workflow are executed by different micro-services in a distributed environment. **Temporal-Server** manages a workflow whose activities are executed by different micro-service in a distributed environment.

1. **What Temporal-Workflow-Implementing-API suggests us for a workflow-implementation in the code:** -- It suggests us followings
   1. **Identify micro-Services and its activity which will participate in a workflow:**
      1. First identify **a micro-service** in your **micro-service echo-system** which will instantiate a workflow. This micro-service will be considered as a **workflow initiator.**
      2. Identify those micro-services whose single activity is going to be a task or activity of a workflow.
      3. Identify the sequence of execution of these activities.
   2. **Create a WorkflowCommon Project**: -- It tells us to create a simple java util-project with following features
      1. A **TaskQueue** enum file which will contain the name of all the activities which are going to be executed within a workflow.
      2. Corresponding to each activity executed by a micro-service, there should be a separate java interface defining that activity.
      3. A common OrderDTO class with all the required properties.
      4. A centralized exception handing, error handling and logger implementation.
   3. **Include this** **WorkflowCommon project as a dependency** in each micro-service project which is a part of the workflow.
   4. Start workflow implementation from **workflow-initiator micro-service**.
      1. Implement **Workflow-Activity**: -- Every micro-service participating in a workflow has to complete an activity. In our use-case, the workflow-initiator is **ms-order** micro-service. An activity of a workflow defined for ms-order is to completeOrder. So, we will simply implement this method in ms-order.
      2. Create a **Workflow**: -- A workflow implementing class has a single method which calls all activity implementing methods in a sequence through their respective interfaces.
      3. Create a **Temporal-Worker** and register it with Temporal-Server: -
      4. Create an **Orchestrator**: --

[1] Every workflow is instantiated though a micro-service.

[2] A set of micro-service may be a part of a workflow.

[3] Every micro-service participating in workflow will have a single responsibility i.e. called an activity.

[4] We will call an activity defined within a workflow-initiator as a local-activity whereas the activity defined within other micro-service participating in the workflow will be called an activity only.

[5] While designing a micro-service, every micro-service participating in a workflow will have two sections

(a) Local-Activity of the micro-service.

(b) Workflow-Activity of the micro-service

**Local-Activity:** -- A local activity within a micro-service is the set of activity which is not the part of a workflow activity. Such activities or API may be serving different api-calls made to the micro-service.

**Workflow-Activity:** -- This is a single task or activity of a workflow within each micro-service. It is so because a micro-service participating in a workflow will do a single task only. We can't define multiple tasks i.e. workflow-activity within a micro service participating in a workflow.

[6] **Orchestrator**: Every micro-service participating in workflow will have a workflow orchestrator implementing class whose role-play is to create a temoral-workflow client and start the workflow execution.

[7] Corresponding to an activity performed by a micro-service, we need to define an activityWorkflowImpl class. For example, we define

(a) OrderFulfilmentWorkflowImpl within ms-order micro-service: This implementation will be done within ms-order micro-service. As this micro-service will receive the order from the front-end, it has two jobs.

(i) It will do some business validation of the received order and records the order-details in its database and keeps the order-status as PENDING at the beginning of initiation of workflow.

(ii) After that, it creates an workflow and starts it. The moment it starts the workflow, it submits all the workflow activities

to the temporal server. The temporal-server registers this workflow with a unique id. Now, the temporal

server has knowledge about which activity of the workflow needs to be performed in which sequence.

The workflow-initiator keeps polling to the temporal-server for knowing whether all the activities

submitted by the workflow-initiator are completed or not. Once, it gets the confirmation from the temporal

server of workflow completion, it calls its workflow completion method and update the status of the order

in its database as COMPLETED, and the temporal server also updates its database as workflow is completed.

Remark:

(a) A workflow-initiator defines all the activities of the workflow for submitting them to the temporal server whereas

the activity performing micro-service will define its own activity only.

(b) Temporal-Server keeps calling all the activities submitted by the workflow-initiator in a sequence which was submitted

by the workflow-initiator. Separate-activity implementation is defined within a micro-service. This activity-implementing

method is a method of an interface which is a part of stub used by Temporal-Server. Temporal-Server uses this stub-method

for calling the respective implemented method in the micro-service through gRPC call. gRPC is a kind of RPC call that

Temporal-Server uses.

(b) PaymentActivityWorkflowImpl within ms-payment micro-service: This activity implementing class will

implement its own activity which will be called by its dedicated worker through Temporal-Server.

(c) InventoryReserveActivityWorkflow within ms-inventory micro-service:This activity implementing class will

implement its own activity which will be called by its dedicated worker through Temporal-Server.

(d) ShipmentActivityWorkflow within ms-shipment micro-service:This activity implementing class will

implement its own activity which will be called by its dedicated worker through Temporal-Server.

[8] Worker: Every micro-service has its own worker defined in itself which is registered with

Temporal-Server when the micro-service starts. Using this worker, Temporal-Server calls its

activity implementing method. When it is called by Temporal-Server, the respective

micro-service starts its activity and reports to the Temporal-Server via its worker.

Remark: A worker registration with Temporal-Server is done by each micro-service through a

post-construct annotation of the method which is registering it to Temporal-Server.

#################################################################### **How to start coding of workflow implementation in a micro-servicce.** ####################################################################

[1] First, we need to create a separate worflow-util project. Say its name is Workflow-Common. Why I am

giving its name as Workflow-Common has some objective. Its objective is to define a separate activity

interface corresponding to each micro-service which is participating in the workflow. Different

micro-service has different activity and we will define such method accordingly. Keep it in mind that one

micro-service will have one activity corresponding to a workflow. if one micro-service is participating

in more than one workflow, then its activity interface that you are defining in this Workflow-Common

utility project will be different one.

Remark: This activity interface that you are defining here is of sharable nature among all those

micro-services which are a part of a workflow.

[2] Include this project in each micro-service which is a part of a workflow-implementation.

[3] Create a worker class in each micro-service and register it with Temporal-Server. Each such worker is

registered with a unique TaskQueue which is defined in Workflow-Common utility project.

[4] Now, define an Orchestrator implementing class whose job is to provide a Temporal client. It will be

used by Worker of each micro-service.

[5] Lastly, we will write a code for activity implementation of each micro-service.

[6] There is an additional implementation of workflow i.e. workflowImpl which will be a part of that

micro-service which is initiating an workflow.

####################################################################

**How the control flows within a micro-service when Temporal-Server starts executing a workflow?**

####################################################################

For understanding this workflow execution, we need to divide micro-services participating in workflow in

the two categories.

All micro-services participating in the workflow first register its worker with Temporal when the

service is started. It is so because the worker registering task is being performed through a post

construct annotation means it is executed when the service is already started. It takes place just after

service is started.

[a] Worflow-Initiator: Every micro-service has its own orchestrator implementing class. In workflow

initiator micro-service, it does two tasks.

(i) It creates a Temporal client for doing two things

(1) For registering a Worker

(2) For starting a workflow

Remark: But, there will be one job of orchestrator in other micro-service which are participating

in workflow i.e. just to create a Temporal-Client for registering its worker with Temporal

Server.

(ii) The control first reaches to a method of its orchestrator i.e. createOrder() with OrderDTO

object as an input.

(iii) Within this method, a workflowStub is created using OrderFulfillmentWorkflow class. This

OrderFulfillmentWorkflowImpl class has a complete definition of workflow. By complete definition

of workflow means, we have defined which activities need to be performed in which sequence

within this workflow.

Within OrderFulfillmentWorkflowImpl class, we have defined all activities of the workflow with

different following controlling parameters.

(a) StartToCloseTimeout: We set this parameter to each activity of micro-service here. This

parameter is telling how much time a particular activity should be completed. It is user

defined parameter. We can adjust it as per the time required to complete an activity. While

coding, it has been fetched from an external configuration file available at a centralized

location GIT through a CONFIG-SERVER.

(b) setTaskQueue: We also set the TastQueue name to an activity. It is so because Temporal-Server

could recognize the activity that which worker should be called for executing this activity.

We have set this TaskQueue-name while registering its worker through a post construct.

(c) setRetryOptions: This is an integral number which is set to an activity. It helps Temporal

Server that how many times it will retry calling this activity. It is also user-defined and

controlled through a config-file available at a centralized location GIT via CONFIG-SERVER.

(d) An activity defined corresponding to a workflow initiator micro-service will be defined

under LocalActivity. It is an important step.

(iv) Once these activities of the workflow are defined, we will consume all these activities in the

workflow implementation method. While consuming these activities, we will call the activity

methods in the same sequence in which we have to complete different activities of the workflow.

(v) Now, the order micro-service-orchestrator will submit this workflow through Temporal-Client to

the Temporal-Server. From here, the Temporal-Server will take care of workflow execution.

(vi) The last activity defined in the workflow execution is completeOrder which is called by the

Temporal Server at the end when all the activities execution is called. When an ms-order gets

this call by Temporal-Server, it updates the status of an order in its database as COMPLETED.

[b] Workflow-Activity-Implemention in other Micro-Services: Using the respective worker, which is already

registered with Temporal-Server, Temporal-Server calls the respective method of activity implementing

class.