Migrating Workflows

Namespace to Namespace

This paper recommends a strategy that can be used to migrate workflows from one Temporal namespace to another.
Note that all of these considerations are *application level* solutions. Direct data migration of execution history is not currently available.

Principles

1. *The principle to aim for in all of these is to do the migration from **within** the current workflow executions. Anything else will be ambiguous and quite possibly lead to inconsistency or data loss.*
2. *The state being migrated to your target Namespace must carry with it all of the appropriate context to resume the Workflow, including execution details like elapsed time or child relationships.*

Cloud Constraints

We rarely see customers hit our limits, but be sure you are acquainted with Cloud Platform Limits enumerated here as part of your migration strategy.

Sample

https://github.com/temporalio/migration-example

Preparing For Your Workflow Migration

There are some key questions to ask about the workflows you are migrating before performing the act of migrating them to another Namespace.

Closed Workflows

Workflows that have been Closed due to Completion, Failure, etc have state you may need to expose.

- Do you need to preserve closed workflow state for querying?
    - eg, Some UI experience queries recently completed workflows.
- Do you use Advanced Visibility queries that might include closed workflow state?
    - eg, A support team queries for workflow status to help customers.

If the answer to all the above is “no”, no action is needed.
If “yes”, keep your current Temporal services running to serve Query SDK invocations until the workflow state has been removed based on Namespace Retention Policy.

Running Workflows

Short-running Workflow Recommendation: Task Queue Draining

Workflows that execute for a **short duration** can often be simply drained off after clients have been deployed to start workflows with the new namespace client.
This is the simplest type of migration, requiring no downtime or coding. The definition for “short” is determined by how long you are willing to let existing workers run that are connected to your current namespace. If the workflows will be running longer than this, read on.

Long-running Workflow Considerations

Review Execution State Considerations to help think through what action needs to be taken to resume an execution in another Namespace.

Migration Strategy: Interceptor

References

- *DOCS*: https://javadoc.io/doc/io.temporal/temporal-sdk/latest/io/temporal/common/interceptors/package-summary.html
- *IMPLEMENTATION*: https://github.com/temporalio/migration-example/tree/a5a3ba50494e2e30916284337bb09801867d50c8/src/main/java/io/temporal/migration/interceptor
- *EXAMPLE*: https://github.com/temporalio/migration-example/tree/a5a3ba50494e2e30916284337bb09801867d50c8/src/main/java/io/temporal/migration/example

How It Works

The WorkerInterceptor manages its own CancellationScope to perform a cancellation *on-demand* when a WorkflowType which is deemed isMigrateable by your Migrator implementation.

Cancellation of the legacy workflows will hence start the migration. This is started by sending a *batch* migrateIt signal using the Temporal CLI.

The interceptor will invoke your migrate implementation inside the cancellation scope as it exits, passing along the result of the Query you expose. The execution is effectively "resumed" in the target Namespace.

Signals could arrive while the new execution is being started in the target Namespace. These can be forwarded to the execution in the target Namespace. See the forwardSignal method in the sample.

Workflows in the legacy Namespace will be Completed (swallowing the CanceledFailure) with the result of the Query described previously.

Requirements

Changes To Existing Workflow Definition

1. Required: Implement a Query named getMigrationState on your workflow definition.
    a. This can return any value but should include what is needed for resuming execution in the target Namespace. // TODO Create "Execution State Considerations" (from content below)
2. As Needed: Update workflow code to avoid duplicating steps or resume timers in the target Namespace.

Interface Implementations

1. Implement the interceptor.Migrator interface.
    a. **NOTE:** Use only **Local Activity** for starting execution in the target Namespace.
    b. See this sample implementation.
2. Inject your Migrator implementation into the WorkerInterceptor.
    a. See this sample.

Caller Adaptations

1. Callers sending signals to legacy workflows should be updated to catch NotFound exceptions and attempt to send the signal to the target Namespace.
2. Callers querying legacy workflows should be updated to first check the target Namespace and fallback to the legacy implementation if the execution is NotFound.
3. // TODO provide sample for failover at Caller client

Performing The Migration

There are a couple of ways to kick off the migration from your cluster.
Please review the Deployment Considerations before doing so.

Option 1: Batch Signal

1. Send a batch signal named migrateIt to your legacy Namespace.
    a. Example: `temporal workflow signal --query 'WorkflowType="LongRunningWorkflow" AND ExecutionStatus="Running"' --name migrateIt --reason 'migration'`
    b. **Note:** Rate limiting this command is being tracked here: https://github.com/temporalio/temporal/issues/4926

Option 2: Workflow with Long Running Activity

// TODO

a.

Execution State Considerations

1. *Skipping Steps*: If your workflow steps cannot guarantee idempotency, determine if you need to skip those steps when resuming the execution in the target Namespace.
2. *Elapsed Time*: If your workflow is "resuming sleep" when in the target Namespace, determine how you will calculate the delta for the `sleep` invocation in the new execution.
3. *Child Relationships*: If your workflow has `ChildWorkflow` relationships (other than `Detached` ParentClosePolicy children), determine how you can pass the state of those children into the parent to execute the child in a resumed state.
4. *Heartbeat state*: If you have long running activities relying on heartbeat state, determine how you can resume these activities in the target Namespace.

Gotchas

1. If ChildWorkflows are the same type as their Parent types, those will be returned in ListFilters being used to gather relevant executions. Unless these are Detached ParentClosePolicy children, this is not what you want since the Parent/Child relationship will not be carried over to the target Namespace.
2. Long running activities that use heartbeat details will not receive the "latest" details in the target Namespace.
3. Duration *between* Awaitables inside workflow definition needs to be considered for elapsed time accuracy when resuming in the target Namespace.
4. Signaling directly from one workflow to another needs to handle `NotFound` executions in the target Namespace since they may resume out of order.

Deployment Considerations

Your Worker fleet deployment will need to support the load being caused by batch signals, queries of so many executions, and the resumption of executions in the target Namespace.

Self-hosted Cluster Constraints

Recall that both a Signal and a Query will be executed against your workflow during the course of this migration. Also, recall that Query API loads the entire history of workflows into Workers to compute the result (if they are not already cached).
That means that your Worker capacity will need to support having those executions *in memory* to serve those requests and that the volume of these requests might be quite high to execute against all the matches to a ListFilter.
You should have a general idea how many of these workflow executions might be in a Running state that will be matched by your ListFilter if you are doing a batch command so you can plan accordingly.

Cloud Rate Limits

Temporal Cloud will impose rate limits on the target Namespace (by default, 200-400 Actions Per Second). Take care to throttle the invocation of the migrateIt signal so that you are not met with ResourceExhausted errors indicating rate limiting in Workers that are connected to Temporal Cloud.
Invoking low-level gRPC API requests (e.g., `DescribeWorkflowExecution`) are also subject to rate limiting, so note your implementation of `Migrator` interface will contribute to the load against Cloud during the migration.