Mastering Temporal CLI Batch Operations for Distributed Systems Management

1. Introduction to Temporal Batch Operations

In the realm of distributed systems, managing a multitude of long-running processes is a common yet complex challenge. Temporal, as a robust orchestration engine, provides powerful mechanisms to define and execute durable workflows. Beyond individual workflow management, there often arises a critical need to interact with or modify a large number of these workflows simultaneously. This is where Temporal's batch operations become indispensable.

Batch operations in Temporal refer to the capability to initiate a single command that systematically affects multiple Workflow Executions. These operations are designed to run asynchronously in the background, processing each targeted Workflow Execution one at a time. This sequential, yet automated, approach ensures the durability and reliability of the operations, even when dealing with thousands or millions of workflows. The underlying Temporal platform guarantees that these bulk actions are completed resiliently, even in the face of system failures or restarts.

The importance of such capabilities in complex distributed environments cannot be overstated. Scenarios frequently emerge where system administrators or developers must perform bulk administrative tasks, react to system-wide events, or apply broad changes across their application landscape. Examples include:

- Updating a feature flag for all active customer order workflows
- Terminating obsolete long-running data processing jobs
- Initiating a mass reset of workflows after a critical bug fix has been deployed

Without dedicated batch capabilities, these tasks would necessitate tedious, error-prone, and often manual interventions, making large-scale system management impractical and risky. Temporal's batch operations streamline these processes, offering a controlled and auditable way to manage distributed application state at scale.

2. Understanding Temporal CLI and tctl Deprecation

The command-line interface (CLI) serves as a primary tool for developers and operators to interact with a Temporal Cluster. Historically, `tctl` was the foundational CLI tool for these interactions, enabling users to perform various operations on namespaces, workflows, task queues, and more. However, the Temporal project has evolved its tooling, and `tctl` has since been deprecated. It is no longer actively supported, and the official recommendation is for users to transition to the newer `temporal` CLI for all cluster interactions.

This transition is crucial for:

- Accessing new features
- Benefiting from ongoing support
- Aligning with the future direction of the Temporal ecosystem

Despite the deprecation, `tctl-v1` retains the ability to execute batch and batch-v2 commands. This backward compatibility layer is a deliberate design choice to facilitate a smoother migration path for existing users and their operational scripts. It acknowledges the practical challenges of a hard cut-over in production environments, allowing organizations to gradually transition their automation and procedures to the `temporal` CLI without immediate disruption.

Important: For all new development, automation, and long-term operational strategies, exclusive adoption of the `temporal` CLI is strongly advised.

The shift from `tctl` to the `temporal` CLI represents more than a mere renaming of a tool; it signifies a strategic evolution in Temporal's operational interface. The continued, albeit limited, support for `tctl-v1`'s batch commands underscores Temporal's commitment to operational continuity for its user base during this transition.

3. Core Batch Management Commands (`temporal batch`)

The `temporal batch` command set is specifically designed for managing the lifecycle and status of batch jobs themselves, which are long-running operations executed by the Temporal Service. These commands provide visibility into, and control over, the batch processes that are affecting multiple workflow executions.

`temporal batch describe` - Monitoring Batch Job Progress

The `temporal batch describe` command is used to retrieve and display detailed information about an ongoing or completed batch job. This functionality is essential for operators to monitor the progress of large-scale operations, understand their current status, and diagnose any issues that may arise during execution.

Usage:

```
temporal batch describe --job-id <YourJobId> [--namespace <YourNamespace>]
```

Parameters:

- `--job-id <string>`: This is a mandatory parameter that specifies the unique identifier of the batch job whose details are to be displayed.
- `--namespace, -n <string>`: This optional parameter specifies the Temporal Service Namespace where the batch job is located. If omitted, the command defaults to the "default" namespace.

Example:

```
temporal batch describe --job-id my-batch-reset-job-123 --namespace production
```

This command would return information such as the job's status (e.g., running, completed, failed), the number of workflows processed, and any associated error messages.

`temporal batch list` - Listing Active and Completed Batch Jobs

The `temporal batch list` command provides an overview of batch jobs, allowing operators to retrieve a list of active or completed batch operations within a specified namespace or across the entire Temporal Service.

Usage:

```
temporal batch list [--namespace <YourNamespace>] [--limit <int>]
```

Parameters:

- `--namespace, -n <string>`: This optional parameter specifies the Temporal Service Namespace from which to list batch jobs. It defaults to the "default" namespace if not provided.
- `--limit <int>`: This optional parameter specifies the maximum number of batch jobs to display in the output. This is useful for managing the volume of information returned, especially in environments with many batch operations.

Example:

```
temporal batch list --limit 100
```

This command provides a concise summary of each batch job, including its ID, type, status, and creation time, enabling quick assessment of the system's batch processing activity.

`temporal batch terminate` - Halting Ongoing Batch Operations

The `temporal batch terminate` command is used to stop an active batch job. This action is typically taken when a batch operation is no longer necessary, has been misconfigured, or is causing unintended side effects that require immediate cessation.

Usage:

```
temporal batch terminate --job-id <YourJobId> --reason <YourTerminationReason>
```

Parameters:

- `--job-id <string>`: This is a mandatory parameter specifying the unique identifier of the batch job to be terminated.
- `--reason <string>`: This is a mandatory parameter requiring a descriptive explanation for terminating the batch job. This reason is recorded in the system logs and is crucial for auditing and post-mortem analysis.

Example:

```
temporal batch terminate --job-id my-signal-campaign-456 --reason "Incorrect query, stopping signal campaign"
```

Irreversibility and Operational Caution

A critical aspect of `temporal batch terminate` is its operational characteristic: **terminating a batch job does not roll back any operations that have already been performed by that job**. This means that if a batch operation has already processed a certain number of workflows before termination, those workflows remain in their modified state. The termination command merely stops any further processing by the batch job.

This behavior highlights a significant design principle within Temporal's batch processing: operations are durable and committed as they occur. There is no inherent "undo" mechanism for batch operations. This necessitates a high degree of caution and meticulous planning before initiating any large-scale batch operation, particularly those with potentially destructive or irreversible effects, such as mass terminations or resets.

4. Batch Operations on Workflows via CLI

The modern `temporal` CLI handles batch operations that directly affect Workflow Executions primarily through specialized `temporal workflow` commands, leveraging the powerful `--query` flag for targeting. This approach differs from the `tctl` paradigm, which often used a `batch start` command with a `--batch_type` flag.

`temporal workflow signal --query` - Signaling Multiple Workflows

The `temporal workflow signal --query` command enables the sending of a signal to a collection of Workflow Executions that are identified by a visibility query. This is a fundamental pattern in Temporal for notifying multiple running workflows about an external event, a change in system state, or to trigger specific logic within them without directly interacting with each workflow individually.

Usage:

```
temporal workflow signal --query "<SQL-like query>" --name "<SignalName>" --input '<JSON_Input>' --reason "<Reason>"
[--namespace <YourNamespace>]
```

Parameters:

- `--query "<SQL-like query>"`: This is a mandatory parameter that specifies an SQL-like query of Search Attributes. This query filters and identifies the specific Workflow Executions that will receive the signal.
- `--name "<SignalName>"`: This mandatory parameter defines the name of the signal method that is expected to be invoked within the target workflows.
- `--input '<JSON_Input>'`: This optional parameter provides the input payload for the signal, which must be in JSON format. This allows for passing specific data along with the signal to the workflows.
- `--reason "<Reason>"`: This optional parameter allows for a descriptive reason to be provided for sending the signal. This reason is recorded in the workflow's history, aiding in auditing and debugging.
- `--namespace, -n <string>`: This optional parameter specifies the target namespace for the operation.

Example:

```
temporal workflow signal --query "WorkflowType='OrderProcessingWorkflow' AND ExecutionStatus='Running'" \
    --name "cancelOrder" \
    --input '{"reason": "Product recall"}' \
    --reason "Mass cancellation due to product recall"
```

`temporal workflow reset-batch` - Resetting Batches of Workflow Executions

The `temporal workflow reset-batch` command is a powerful administrative tool used to revert the state of multiple Workflow Executions simultaneously to a previous point in their history. This capability is invaluable for disaster recovery, applying code fixes to running workflows that encountered a bug, or correcting data inconsistencies across a large set of workflow instances.

Usage:

```
temporal workflow reset-batch --query "<SQL-like query>" --reason "<Reason>" --type <ResetType> [--dry-run] \
    [--input_file <filename>] [--exclude_file <filename>] [--only_non_deterministic] [--reset_bad_binary_checksum
<checksum>]
```

Parameters:

- `--query "<SQL-like query>"`: This is a mandatory parameter that uses an SQL-like query of Search Attributes to identify the specific Workflow Executions that are targeted for the reset operation.
- `--reason "<Reason>"`: This is a mandatory parameter requiring a descriptive reason for performing the batch reset. This reason is crucial for auditing and is recorded in the Event History of each affected workflow.
- `--type <ResetType>`: This mandatory parameter specifies the exact point in the Workflow's Event History to which it will be reverted:
  - `FirstWorkflowTask`: Resets the workflow's state to the very beginning of its Event History.
  - `LastWorkflowTask`: Resets the workflow's state to the last successfully executed Workflow task.
  - `LastContinuedAsNew`: Resets the workflow's state to the last point where the workflow was "continued as new."
  - `WorkflowExecutionSignaled`: Resets to the point where a WorkflowExecutionSignaled event occurred.
- `--dry-run`: This optional but highly recommended flag simulates the batch reset operation without actually modifying any workflow states.
- `--input_file <filename>`: Provides an input file containing a list of Workflow IDs to reset.
- `--exclude_file <filename>`: Provides an input file containing Workflow IDs to exclude from the reset operation.
- `--only_non_deterministic`: If specified, the workflow execution will be reset only if its last event was a WorkflowTaskFailed with a non-determinism error.
- `--reset_bad_binary_checksum <checksum>`: This parameter is used in conjunction with a BadBinary reset type to specify the binary checksum of the problematic worker code.

Example:

```
# Dry run first
temporal workflow reset-batch --query "WorkflowType='LoyaltyProgram'" \
    --reason "Applying fix for loyalty calculation bug" \
    --type LastWorkflowTask \
    --dry-run

# After verifying the dry run output, execute the actual reset
temporal workflow reset-batch --query "WorkflowType='LoyaltyProgram'" \
    --reason "Applying fix for loyalty calculation bug" \
    --type LastWorkflowTask
```

`temporal workflow list --query` - Querying Multiple Workflows in a Batch Context

While `temporal workflow query` is typically used to retrieve the state of a single workflow execution, the `temporal workflow list --query` command is the primary and most effective tool for identifying and filtering multiple Workflow Executions based on their Search Attributes.

Usage:

```
temporal workflow list --query "<SQL-like query>" [--limit <int>] [--open]
```

Parameters:

- `--query "<SQL-like query>"`: This is a mandatory parameter that accepts an SQL-like query string to filter workflow executions based on their Search Attributes.
- `--status <Status>`: This optional parameter allows filtering by the workflow execution status, such as Running, Completed, or Failed.
- `--limit <int>`: This optional parameter specifies the maximum number of workflow executions to list in the output.
- `--open`: This optional flag, when present, restricts the listing to only open (currently running) workflow executions.

Example:

```
# List all running workflows of type DataProcessingWorkflow started within the last 24 hours
temporal workflow list --query "WorkflowType='DataProcessingWorkflow' AND StartTime > '24h'" --status Running

# List all completed workflows with a specific custom search attribute
temporal workflow list --query "CustomID = 'XYZ789'" --status Completed
```

Visibility as the Foundation for Batch Operations

The extensive reliance on the `--query` flag across `temporal workflow signal`, `temporal workflow reset-batch`, and `temporal workflow list` commands underscores a fundamental architectural dependency within Temporal's operational capabilities. The accuracy and efficiency of these batch operations are directly tied to the underlying visibility store and its Search Attributes.

Important Considerations:

1. **Advanced Visibility Required**: The `--query` flag is only supported when Advanced Visibility is configured. This implies that for organizations to fully leverage the robust, large-scale batch management features of Temporal, a basic Temporal setup might not suffice.
2. **Eventual Consistency**: The visibility store operates with eventual consistency. This means that while powerful, queries against this store might not reflect real-time state changes instantaneously. There can be a slight delay between a workflow's actual state transition and its propagation and indexing within the visibility store.

5. Common Flags and Global Options

Temporal CLI commands, including those used for batch operations, support a variety of global flags. These flags allow for consistent configuration of connection details, logging behavior, and other operational parameters across different commands.

Flag Name	Alias	Description	Default Value	Environment Variable
`--address`		Temporal Service gRPC endpoint.	`127.0.0.1:7233`	`TEMPORAL_CLI_ADDRESS`
`--api-key`		API key for requests.	`None`	
`--codec-auth`		Authorization header for Codec Server requests.	`None`	
`--codec-endpoint`		Remote Codec Server endpoint.	`None`	
`--codec-header`		HTTP headers for requests to codec server (KEY=VALUE).	`None`	
`--color`		Output coloring. Accepted values: always, never, auto.	`auto`	

Flag Name	Alias	Description	Default Value	Environment Variable
`--command-timeout`		The command execution timeout. 0s means no timeout.	`0s`	
`--env`		Active environment name (ENV).	`default`	
`--env-file`		Path to environment settings file.	`$HOME/.config/temporalio/temporal.yaml`	
`--grpc-meta`		HTTP headers for requests (KEY=VALUE).	`None`	
`--log-format`		Log format. Accepted values: text, json.	`text`	
`--log-level`		Log level. Accepted values: debug, info, warn, error, never.	`info`	
`--namespace`	`-n`	Temporal Service Namespace.	`default`	`TEMPORAL_CLI_NAMESPACE`
`--no-json-shorthand-payloads`		Raw payload output, even if the JSON option was used.	`false`	
`--tls_ca_path`		Path to a server Certificate Authority (CA) certificate file.	`None`	`TEMPORAL_CLI_TLS_CA`
`--tls_cert_path`		Path to a public X.509 certificate file for mutual TLS authentication.	`None`	`TEMPORAL_CLI_TLS_CERT`
`--tls_disable_host_verification`		Disable verification of the server certificate.	`false`	`TEMPORAL_CLI_TLS_DISABLE_HOST_VER`
`--tls_key_path`		Path to a private key file for mutual TLS authentication.	`None`	`TEMPORAL_CLI_TLS_KEY`
`--tls_server_name`		Override target TLS server name used for TLS host verification.	`None`	

The ability to set these global flags via environment variables is a significant advantage for operational efficiency. This practice enables the creation of cleaner, more portable scripts, as sensitive or frequently used parameters do not need to be explicitly included in every command line invocation.

6. Best Practices for Batch Operations

Effective and safe utilization of Temporal CLI batch operations requires adherence to several best practices:

1. Prioritize `temporal` CLI

Always use the modern `temporal` CLI over the deprecated `tctl` for new development and automation. This ensures access to the latest features, bug fixes, and long-term support.

2. Leverage `--dry-run` for reset-batch

The `--dry-run` flag for `temporal workflow reset-batch` is an indispensable safety mechanism. Always perform a dry run before executing any actual batch reset to verify that the query correctly identifies the target workflows and to understand the scope of the operation.

3. Understand Eventual Consistency

Be aware that the visibility store, which powers the `--query` functionality, is eventually consistent. This implies that real-time state changes might not be immediately reflected in query results. For critical operations, consider potential delays in visibility data propagation.

4. Use Descriptive `--reason` Flags

For all batch operations that support it (e.g., signal, reset-batch, terminate), provide clear and descriptive reasons. These reasons are permanently recorded in the workflow's event history and are invaluable for auditing, debugging, and understanding the context of past administrative actions.

5. Acknowledge Irreversibility of Termination

Remember that `temporal batch terminate` stops future processing but does not roll back already completed operations. This characteristic means that any actions performed by the batch job before its termination are permanent.

6. Utilize Global Flags and Environment Variables

For consistent and streamlined operations, leverage global CLI flags and their corresponding environment variables. This practice simplifies scripting, reduces command-line clutter, and ensures that operations are consistently targeting the correct cluster and namespace.

7. Monitor Batch Job Progress

Regularly use `temporal batch describe` to monitor the progress of ongoing batch jobs. This proactive monitoring allows for early detection of issues and provides the necessary information to decide whether to continue or terminate a job.

7. Conclusion

Temporal CLI batch operations provide powerful capabilities for managing and interacting with large numbers of workflow executions in a distributed system. The transition from the deprecated `tctl` to the modern `temporal` CLI marks a significant evolution in Temporal's operational tooling, offering a more streamlined and future-proof interface.

While core batch management commands like `describe`, `list`, and `terminate` provide control over the batch jobs themselves, operations directly affecting workflows, such as signaling and resetting, are now integrated with the `temporal workflow` command family, primarily leveraging the robust `--query` mechanism.

A profound understanding of the underlying visibility store, its eventual consistency, and the critical role of Advanced Visibility configuration is paramount for effective batch targeting. Furthermore, the irreversible nature of certain operations, like batch termination, and the indispensable safety net provided by features like dry-run for batch resets, underscore the necessity of meticulous planning and cautious execution.

By adhering to established best practices, including thorough testing, descriptive logging, and consistent configuration through global flags and environment variables, operators can harness the full potential of Temporal's batch capabilities to manage complex distributed applications with efficiency, reliability, and confidence.