MoneyLion

Invincible SRE Workflows with Temporal

Apr 08, 2025: SRE KL User Group

Michael Leow, Chee Lim Toh

Slide + Code: *https://github.com/cheelim1/go-temporal-sre*

Disclaimer: *Talks is opinion of speaker; does not reflect position of employer.*

Agenda

What is Temporal?

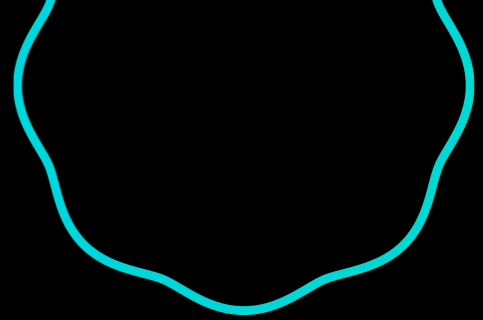
Getting Rid of Cron Forever for long-running jobs

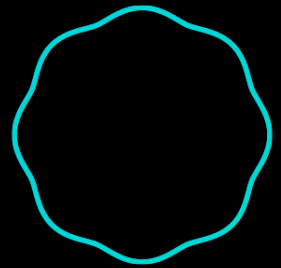
Granting Superpowers to your humble scripts

Just In Time (JIT) Access Demo

Alternatives to Temporal

Q&A





01
What is Temporal

What is Temporal

Durable Execution Platform: An abstraction for building simple, sophisticated, resilient applications

Code like it never fails

Write your business logic as code. Create Workflows that guarantee execution; idempotency guaranteed. Code Activities to handle and retry failure-prone logic. Support patterns: Event-Driven, Saga, Batch, Schedules, State-Machines

Testing + Observability

Comprehensive test suites; including time travel (workflows that takes days, months, years). Event Replays and audit logs with minimal effort. Metrics, tracing, logging available including search to troubleshoot and scaling.

Cross-Platform Support

Write business logic using native SDKs (major languages, communities). Inter-communicate + mix-match as needed. Strong access boundaries within namespace. Teams can securely communicate across namespaces via Nexus

Open Source + Commercial Managed

Full local-dev capabilities in OSS. Fully self-host with own controlled Cassandra cluster. Leverage Managed Temporal Cloud for 200ms SLA; scaling to millions++ of workflows and support

02
Trouble with Cron

Problems with Cron

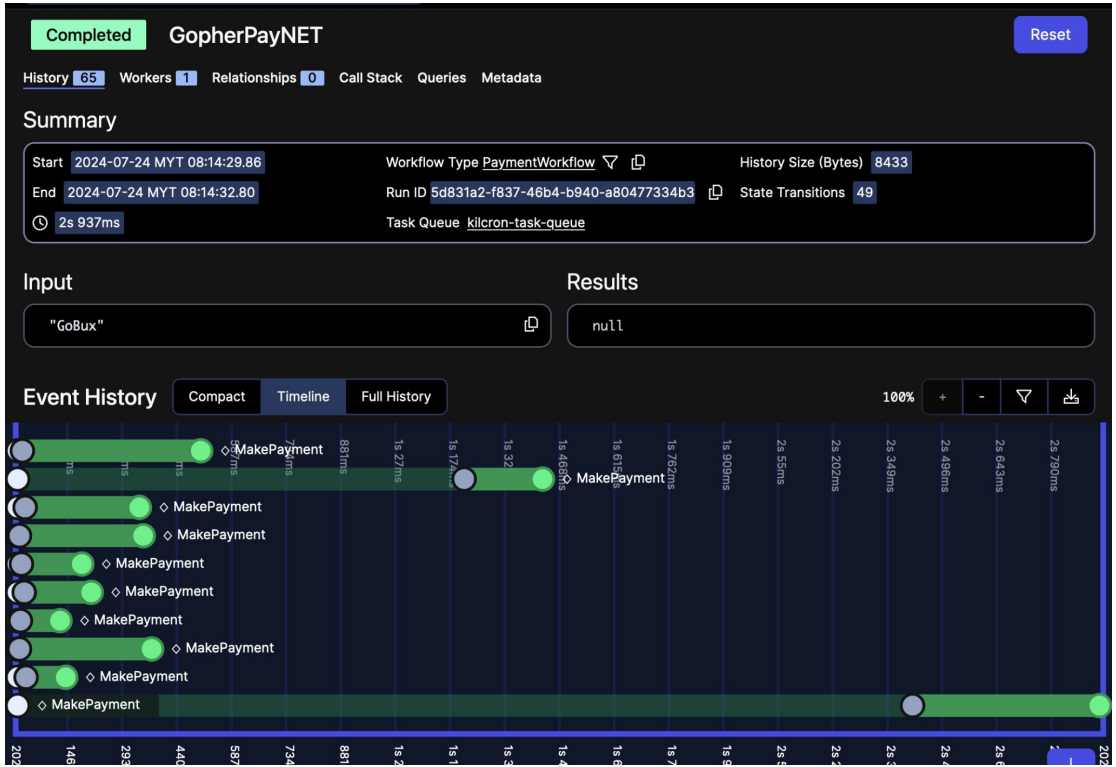
- **Scene:** Startup getting traction
- Any long running day-to-day process: (e.g reports, payments, data processing)
- **Don’t:** Extend your web server timeout!
- Cron to the rescue!!
- Now got more problems; backfill failures

Cron - Wishful Thinking

- Cron jobs start immediately; no latency, no failure!

Cron - Closer to Reality

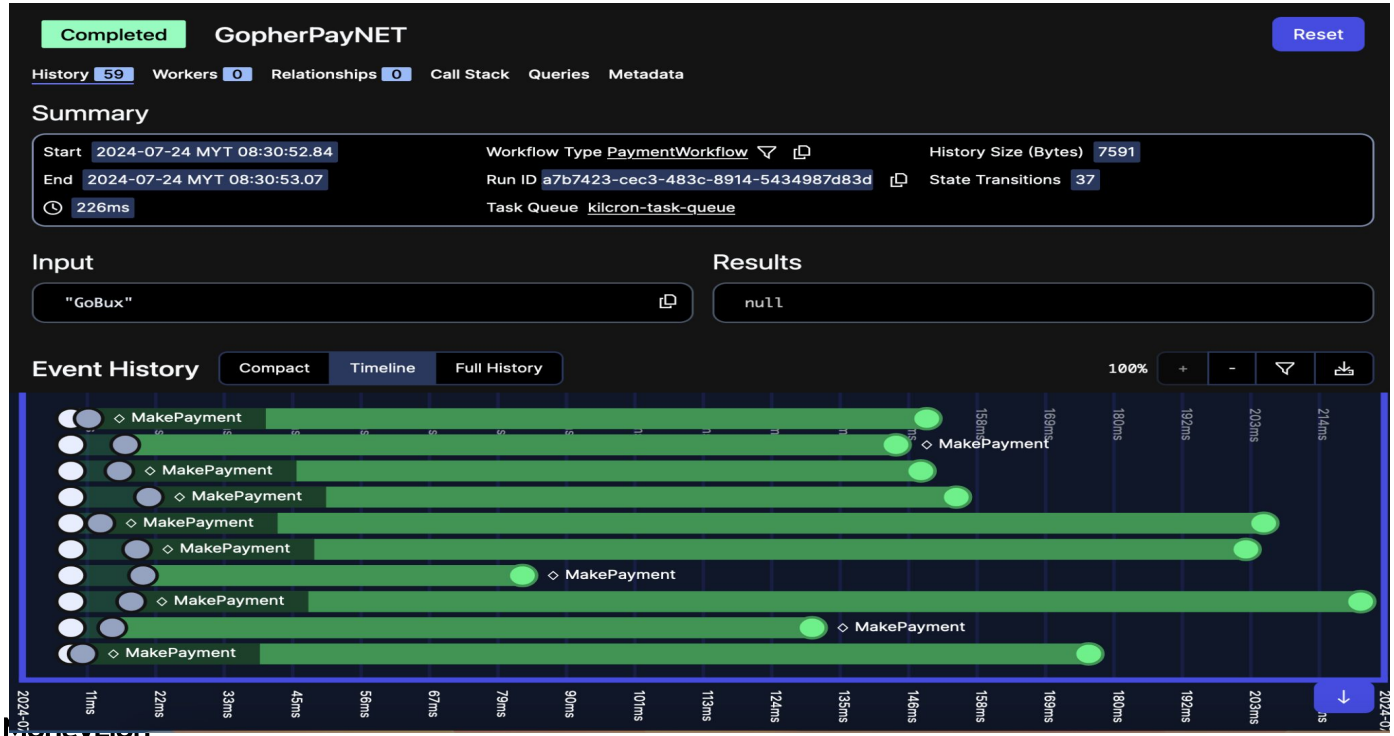
- Cron jobs have variable latency; no failure!




KilCron
TEMPORAL
MODERN
ANTQURTED
CRON
CRON
CRON
KilCron
TEMPORAL
MODERN
ANTQURTED
CRON
CRON
CRON

Cron - Reality

- Use Temporal Schedules instead. Can start, pause or signal
- Rethink the whole flow; break it down to smaller parts (HOW?)



03
Granting Super Powers to your Humble Scripts

Real Life is Messy (as a SRE)

- Real life; unexpected events can happen! Not deterministic
- Bash or Python scripts used for automation are flaky
- Many dependencies out of control: DBs overloaded, network, vendors, cosmic-rays
- **Consequence:** Double billing of customers, Unnecessary cloud resources activated, Database upgrade left in unrecoverable state
- **Solution:** Idempotency allows safe retries. An operation that can be applied multiple times without changing the result
- Temporal to the rescue! (of course)

Traditional non-Idempotent

- Each time the script runs it is different! Not deterministic

```
→  go-temporal-sre git:(main) × make superscript-demo-2
Running SuperScript Demo 2: Traditional Non-Idempotent Script

Script Run from IP: 14.1.247.54
=======================================================
Starting batch processing of 10 OrderIDs
=======================================================

Processing OrderID: 7307 (1/10)
ERROR: OrderID 7307 failed with exit code 1 in 0s
  Starting payment processing for OrderID: 7307
  Starting processing step 1...
  Step 1 failed: FAILED: Processing Step 1 for OrderID 7307
  Cleaning up resources...
  ERROR: Script terminated with exit code: 1 - Step 1 failed: FAILED: Proc
-------------------------------------------------------

Processing OrderID: 5493 (2/10)
SUCCESS: OrderID 5493 processed successfully in 5s
  Starting payment processing for OrderID: 5493
  Starting processing step 1...
  Step 1 completed successfully: Step1 5493
  Starting processing step 2...
  Step 2 completed successfully: Step2 5493
  Payment processing completed successfully for OrderID: 5493
  Cleaning up resources...
-------------------------------------------------------
```

```
Processing OrderID: 6606 (9/10)
ERROR: OrderID 6606 failed with exit code 1 in 0s
  Starting payment processing for OrderID: 6606
  Starting processing step 1...
  Step 1 failed: FAILED: Processing Step 1 for OrderID 6606
  Cleaning up resources...
  ERROR: Script terminated with exit code: 1 - Step 1 failed: FAILED: Pro
-------------------------------------------------------

Processing OrderID: 8448 (10/10)
SUCCESS: OrderID 8448 processed successfully in 4s
  Starting payment processing for OrderID: 8448
  Starting processing step 1...
  Step 1 completed successfully: Step1 8448
  Starting processing step 2...
  Step 2 completed successfully: Step2 8448
  Payment processing completed successfully for OrderID: 8448
  Cleaning up resources...
-------------------------------------------------------

Batch Processing Summ
==========
Total OrderIDs proce
Successful: 5
Failed: 5
Success rate: 50%
```

```
SUCCESS: OrderID 3078 processed successfully in 3s
  Starting payment processing for OrderID: 3078
  Starting processing step 1...
  Step 1 completed successfully: Step1 3078
  Starting processing step 2...
  Step 2 completed successfully: Step2 3078
  Payment processing completed successfully for OrderID: 3078
  Cleaning up resources...
-------------------------------------------------------

Processing OrderID: 8577 (7/10)
ERROR: OrderID 8577 failed with exit code 1 in 0s
  Starting payment processing for OrderID: 8577
  Starting processing step 1...
  Step 1 failed: FAILED: Processing Step 1 for OrderID 8577
  Cleaning up resources...
  ERROR: Script terminated with exit code: 1 - Step 1 failed: FAILED: Processing Step 1
-------------------------------------------------------

Processing OrderID: 5479 (8/10)
ERROR: OrderID 5479 failed with exit code 2 in 4s
  Starting payment processing for OrderID: 5479
  Starting processing step 1...
  Step 1 completed successfully: Step1 5479
  Starting processing step 2...
  Step 2 failed: ERROR: Timeout occurred after 3s for OrderID 5479
  Cleaning up resources...
  ERROR: Script terminated with exit code: 2 - Step 2 failed: ERROR: Timeout occurred af
79
-------------------------------------------------------
```


MoneyLion

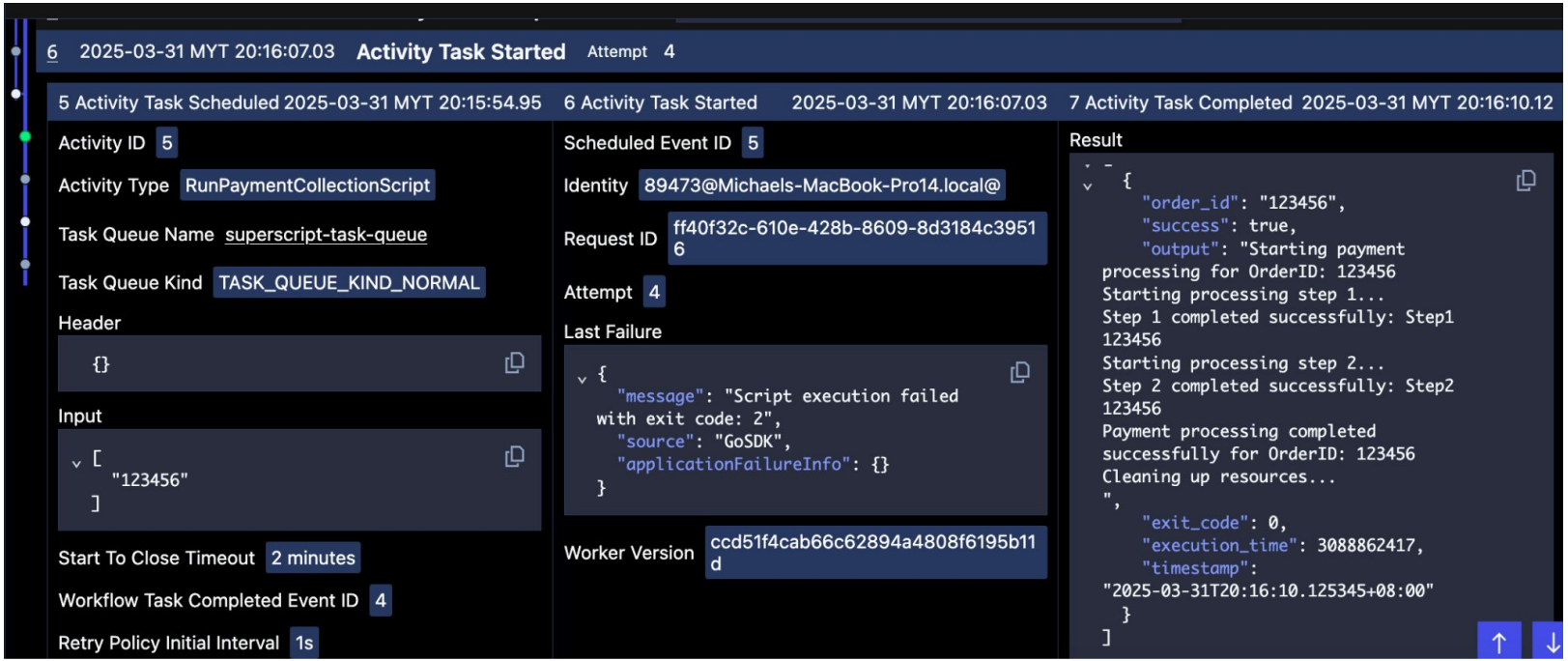
```
# Process each OrderID in the list
for order_id in "${ORDER_IDS[@]}"; do
    TOTAL_COUNT=$((TOTAL_COUNT + 1))
    echo -e "\n${YELLOW}Processing OrderID: $order_id (${TOTAL_COUNT}/${#ORDE

    # Record start time
    start_time=$(date +%s)

    # Call the single payment collection script and capture output
    # We use set +e to prevent the loop from exiting if the script fails
    set +e
    output=$($SOURCE_DIR/single_payment_collection.sh "$order_id" 2>&1)
    exit_code=$?
    set -e

    # Record end time and calculate duration
    end_time=$(date +%s)
    duration=$((end_time - start_time))

    # Display result based on exit code
```

```
echo "Starting payment processing for OrderID: $ORDER_ID"

# Process Step 1
echo "Starting processing step 1..."
# Turn off errexit temporarily to capture the output and ret
set +e
step1_result=$(process_step1 "$ORDER_ID")
step1_code=$?
set -e

if [[ $step1_code -ne 0 ]]; then
    LAST_ERROR_MSG="Step 1 failed: $step1_result"
    echo "$LAST_ERROR_MSG" >&2
    exit $step1_code
fi

echo "Step 1 completed successfully: $step1_result"
```

```
# Process Step 2
echo "Starting processing step 2..."
# Turn off errexit temporarily to capture the output and re
set +e
step2_result=$(process_step2 "$ORDER_ID")
step2_code=$?
set -e

if [[ $step2_code -ne 0 ]]; then
    LAST_ERROR_MSG="Step 2 failed: $step2_result"
    echo "$LAST_ERROR_MSG" >&2
    exit $step2_code
fi

echo "Step 2 completed successfully: $step2_result"

# All steps completed successfully
echo "Payment processing completed successfully for OrderID
exit 0
```

Single Workflow made Deterministic

- From chaos to order; now idempotent
- Ensure WorkflowID no reuse; retry for free



- Reuse Policy: **WORKFLOW_ID_REUSE_POLICY_REJECT_DUPLICATE**
- ActivityOptions to Retry

```go
// This workflow wraps a potentially non-ide
func SinglePaymentCollectionWorkflow(ctx wor
    logger := workflow.GetLogger(ctx)
    logger.Info( msg: "Starting SinglePayment
    startTime := workflow.Now(ctx)

    // Define activity options
    ao := workflow.ActivityOptions{
        StartToCloseTimeout: 2 * time.Minute,
        RetryPolicy: &temporal.RetryPolicy{
            InitialInterval:    time.Second,
            BackoffCoefficient: 2.0,
            MaximumInterval:    30 * time.Second,
            MaximumAttempts:    5,
        },
    }
    ctx = workflow.WithActivityOptions(ctx, ao)

    var activityResult PaymentResult // Activity should return this structure or similar
    err := workflow.ExecuteActivity(ctx, activity: "RunPaymentCollectionScript", params.OrderID).Get(ctx
```

```go
// Create a workflow ID based on the order ID
workflowID := fmt.Sprintf( format: "%s-%s", superscript.SinglePaymentWorkflowTy

// Start the workflow with idempotency guaranteed by Temporal
workflowOptions := client.StartWorkflowOptions{
    ID:        workflowID,
    TaskQueue: superscript.SuperscriptTaskQueue,
    // Reject duplicate ensures idempotency
    WorkflowIDReusePolicy: enums.WORKFLOW_ID_REUSE_POLICY_REJECT_DUPLICATE,
}
```

Completed **single-payment-workflow-4242**

Reset

Current Details

Start	2025-03-31 MYT 19:33:20.36	Run ID	0195ebfa-3d6d-70cf-9ad3-301d8ccbea77	History Size (Bytes)	2318
End	2025-03-31 MYT 19:33:24.40	Workflow Type	SinglePaymentCollectionWorkflow		
	4s 42ms	Task Queue	superscript-task-queue		

History 11 | Relationships 0 | Workers 1 | Pending Activities 0 | Call Stack | Queries | Metadata

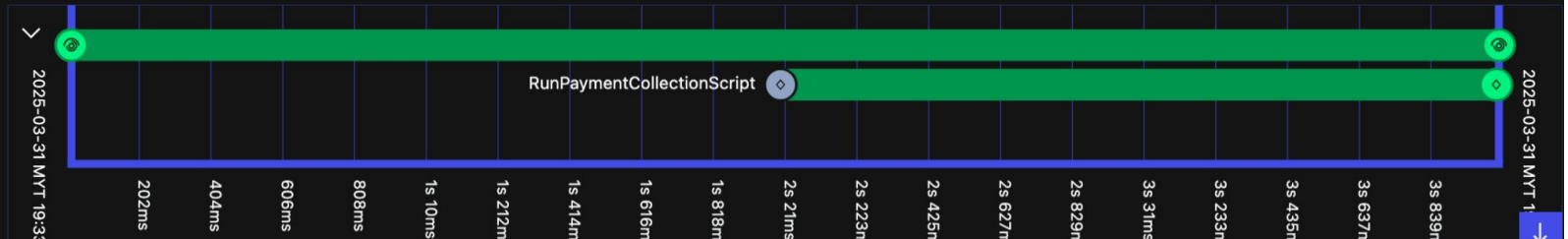
Input


```
{
  "OrderID": "4242"
}
```


Result

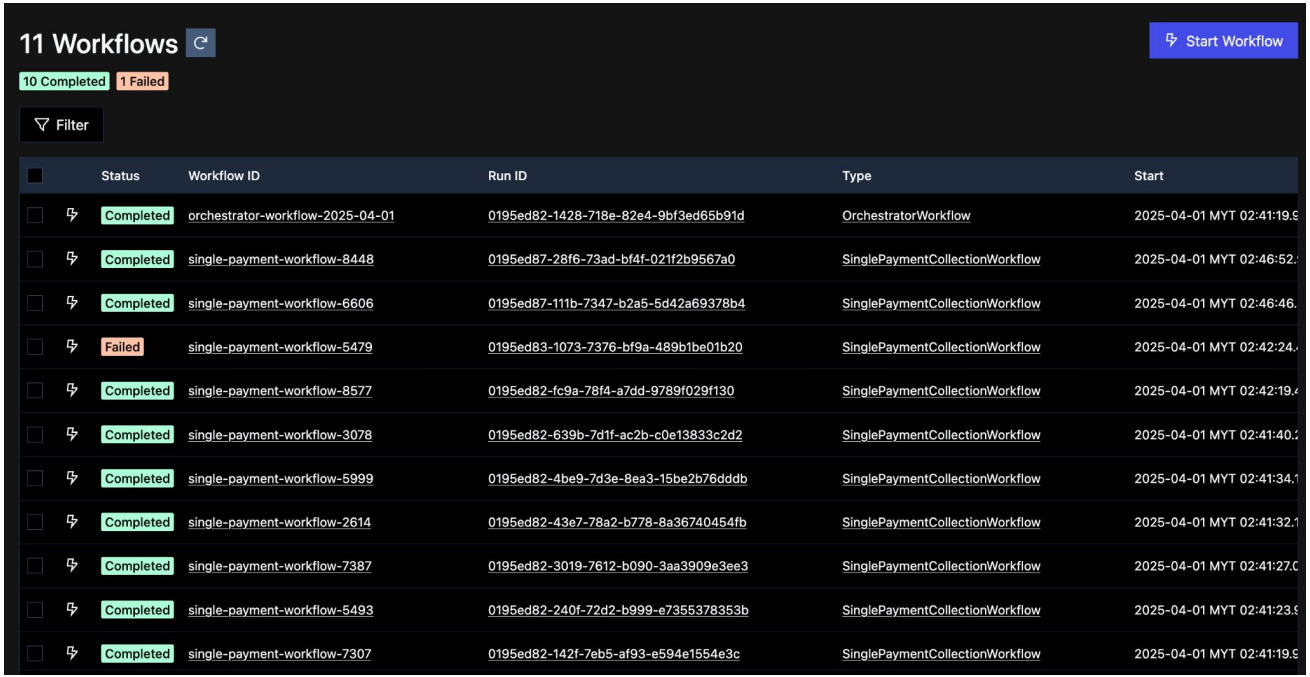

```
{
  "order_id": "4242",
  "success": true,
  "output": "Starting payment processing for OrderID: 4242
Starting processing step 1...
Step 1 completed successfully: Step1 4242
Starting processing step 2...
Step 2 completed successfully: Step2 4242
Payment processing completed successfully for OrderID: 4242
Cleaning up resources...
",
  "exit_code": 0,
  "execution_time": 2024431334,
  "timestamp": "2025-03-31T19:33:24.398353+08:00"
}
```


Event History



Superscript Demo

- Real world is messy; but now under control - idempotent + auto-retry
- It may take time but run to completion successfully



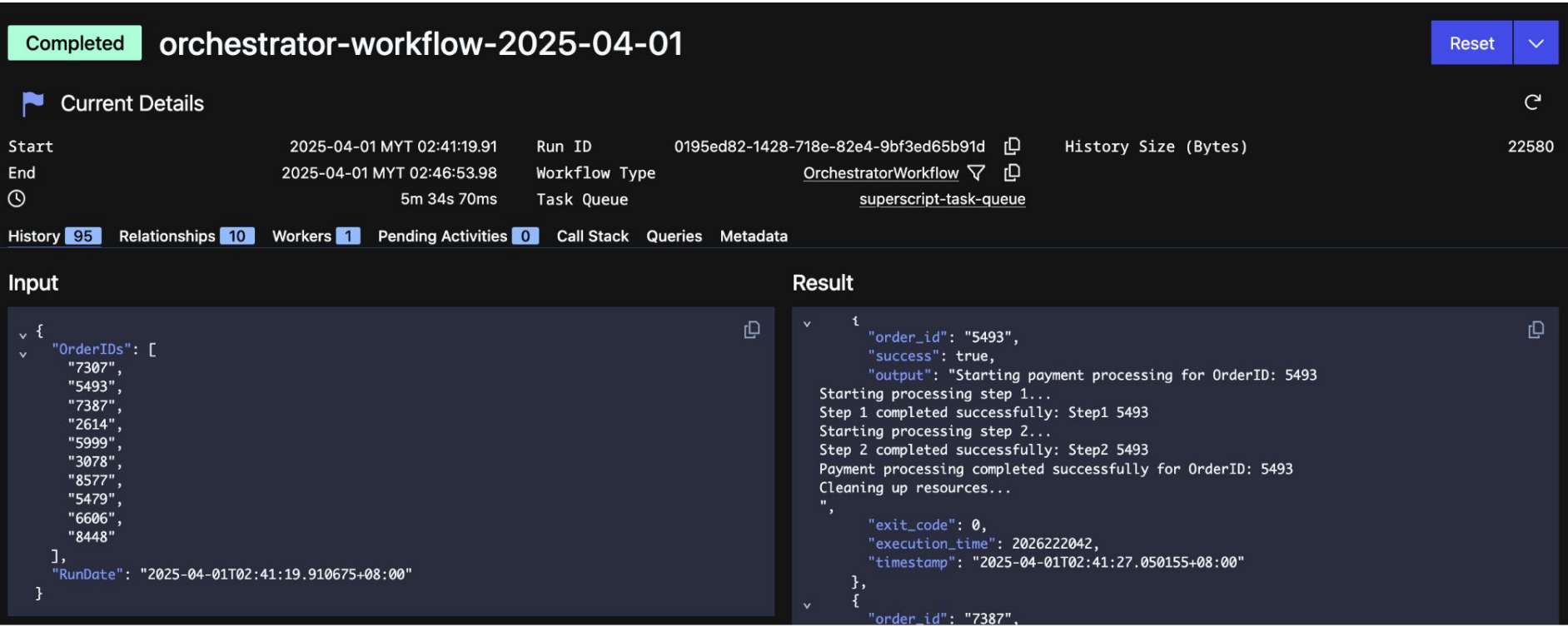
Superscript - Code / Flow

- Straight-forward, composable; calls earlier SinglePaymentWorkflow

```go
func OrchestratorWorkflow(ctx workflow.Context, params OrchestratorWorkflowParams) (*BatchResult, error) {

    if len(params.OrderIDs) == 0 {...}

    selector := workflow.NewSelector(ctx)
    sem := workflow.NewSemaphore(ctx, int64(concurrency))
    numScheduled := 0
    numCompleted := 0
    futuresMap := make(map[workflow.Future]int) // Map future to original index

    logger.Info( msg: "Starting concurrent child workflow execution",  keyvals...: "concurrency", concurrency)

    for numCompleted < len(params.OrderIDs) {
        // Schedule new workflows if concurrency l
        // Reverting to standard TryAcquire(1) bas
        if numScheduled < len(params.OrderIDs) &&
    }
```

```go
    workflowID := fmt.Sprintf( format: "%s-%s", SinglePaymentWorkflowType, orderID)
    childCtx := workflow.WithChildOptions(ctx, workflow.ChildWorkflowOptions{
        WorkflowID:            workflowID,
        WorkflowIDReusePolicy: enums.WORKFLOW_ID_REUSE_POLICY_REJECT_DUPLICATE,
        TaskQueue:             SuperscriptTaskQueue,
    })

    exFuture := workflow.ExecuteChildWorkflow(childCtx, SinglePaymentWorkflowType, SinglePaymentWorkflowParams{Order
    futuresMap[exFuture] = idx // Store mapping

    selector.AddFuture(exFuture, func(f workflow.Future) {
        completedIdx := futuresMap[f]
        completedOrderID := params.OrderIDs[completedIdx]
        completedWorkflowID := fmt.Sprintf( format: "%s-%s", SinglePaymentWorkflowType, completedOrderID)
        var result PaymentResult

        err := f.Get(ctx, &result)
```

MoneyLion


Completed
orchestrator-workflow-2025-04-01
Reset
Current Details
Start
2025-04-01 MYT 02:41:19.91
End
2025-04-01 MYT 02:46:53.98
5m 34s 70ms
Run ID
0195ed82-1428-718e-82e4-9bf3ed65b91d
Workflow Type
OrchestratorWorkflow
Task Queue
superscript-task-queue
History Size (Bytes)
22580
History 95
Relationships 10
Workers 1
Pending Activities 0
Call Stack
Queries
Metadata
Input
{
"OrderIDs": [
"7307",
"5493",
"7387",
"2614",
"5999",
"3078",
"8577",
"5479",
"6606",
"8448"
],
"RunDate": "2025-04-01T02:41:19.910675+08:00"
}
Result
{
"order_id": "5493",
"success": true,
"output": "Starting payment processing for OrderID: 5493
Starting processing step 1...
Step 1 completed successfully: Step1 5493
Starting processing step 2...
Step 2 completed successfully: Step2 5493
Payment processing completed successfully for OrderID: 5493
Cleaning up resources...
",
"exit_code": 0,
"execution_time": 2026222042,
"timestamp": "2025-04-01T02:41:27.050155+08:00"
},
{
"order_id": "7387",

Event History

All | Compact | JSON

Desc

Pending and Failed Only

Event Types

86 87 91	2025-04-01 MYT 02:46:52.91	Child Workflow	Workflow Type Name	SinglePaymentCollectionWorkflow	Result	[{"order_id":"8448","success":true,"output":"Start…	
77 78 82	2025-04-01 MYT 02:46:46.80	Child Workflow	Workflow Type Name	SinglePaymentCollectionWorkflow	Result	[{"order_id":"6606","success":true,"output":"Start…	
68 69 73	2025-04-01 MYT 02:42:24.49	Child Workflow	Workflow Type Name	SinglePaymentCollectionWorkflow	Failure Message	activity error	
59 60 64	2025-04-01 MYT 02:42:19.41	Child Workflow	Workflow Type Name	SinglePaymentCollectionWorkflow	Result	[{"order_id":"8577","success":true,"output":"Start…	
50 51 55	2025-04-01 MYT 02:41:40.25	Child Workflow	Workflow Type Name	SinglePaymentCollectionWorkflow	Result	[{"order_id":"3078","success":true,"output":"Start…	
41 42 46	2025-04-01 MYT 02:41:34.18	Child Workflow	Workflow Type Name	SinglePaymentCollectionWorkflow	Result	[{"order_id":"5999","success":true,"output":"Start…	
32 33 37	2025-04-01 MYT 02:41:32.13	Child Workflow	Workflow Type Name	SinglePaymentCollectionWorkflow	Result	[{"order_id":"2614","success":true,"output":"Start…	
23 24 28	2025-04-01 MYT 02:41:27.06	Child Workflow	Workflow Type Name	SinglePaymentCollectionWorkflow	Result	[{"order_id":"7387","success":true,"output":"Start…	

04
Just In Time (JIT) Access + Demo

What is JIT?

Is it the same as break glass?

JIT vs Break glass

Use Cases of Just In Time(JIT)

Temporary AWS IAM Access

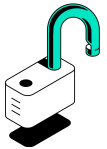
Gaining a temporary elevated role to perform a certain access on a Resource in AWS.

Temporary K8s Access

Temporary access to access k8s using IAM to perform elevated troubleshooting in the production environment cluster.

Temporary Access to approve Github Deployments

Temporary access to approve deployments when no one in the team is available to review and approve.



Temporary Database Access

Temporary access to a certain database (most likely production) to perform a certain change while being audited.

Every JIT request must be audited & comply to the audit requirements.

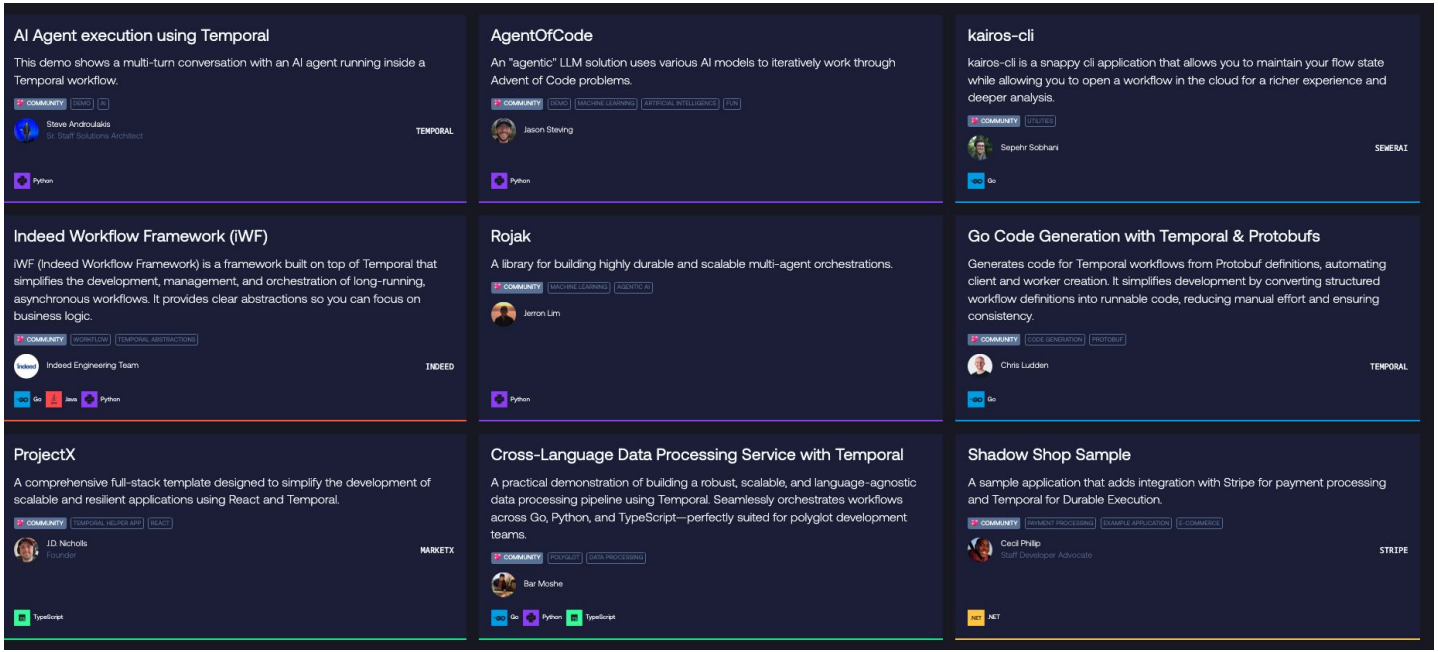
1. ***Ticket Tracked***
2. ***Required Approvers to approve requests***
3. ***Audit trail***
4. ***Access is automatically revoked after specific period of time.***

DEMO

05
Temporal Universe Expanded

Introducing: Temporal Code Exchange

Marketplace of ideas to study + learn from. Open to submission!
https://temporal.io/code-exchange



06
Alternatives to Temporal

Temporal Alternatives

Crowded market ... who wins? Who has the best DX?

Restate (Go,Java,Python,Typescript, Rust)

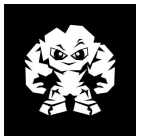
https://docs.restate.dev/

DBOS (Typescript, Python)

https://www.dbos.dev/

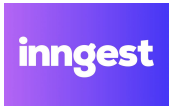
Cloudflare Workflows (Typescript)

https://cloudflare.com/



Golem Cloud (WASM)

https://www.golem.cloud/



Inngest (JS)

https://www.inngest.com/

Littlehorse (Go,Java,Python,Typescript,.NET)

https://littlehorse.io/

07
Q&A

***Slide + Code**: https://github.com/cheelim1/go-temporal-sre*

Thank you

MoneyLion®