



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?

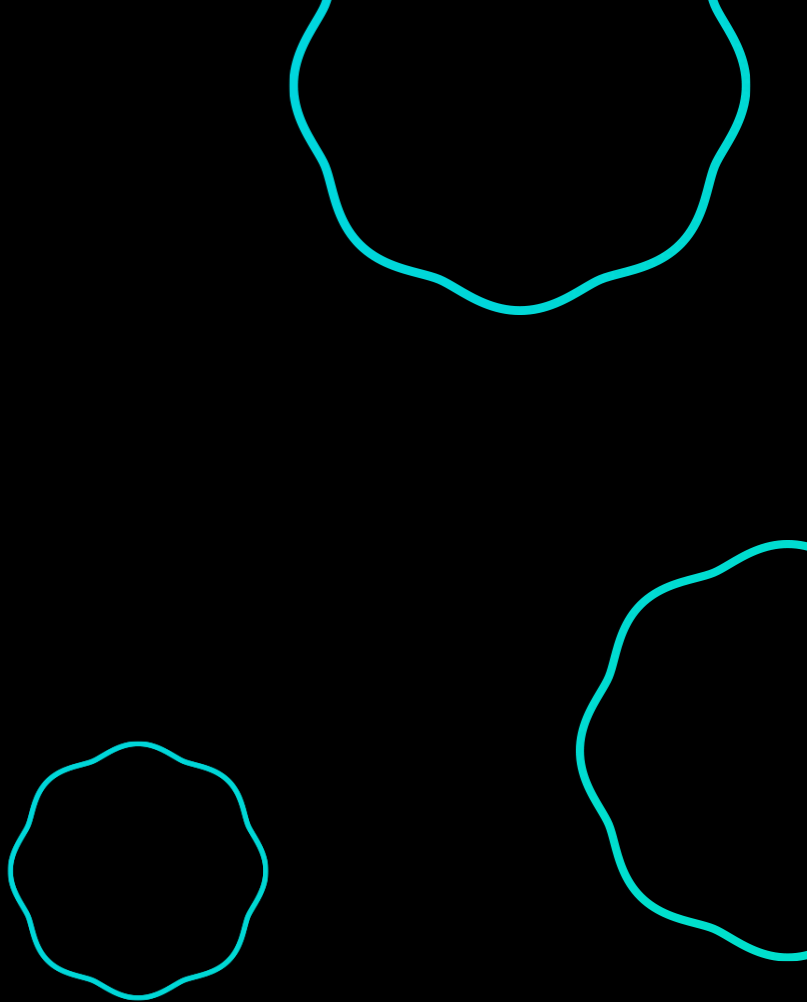
Getting Rid of Cron Forever for long-running jobs

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

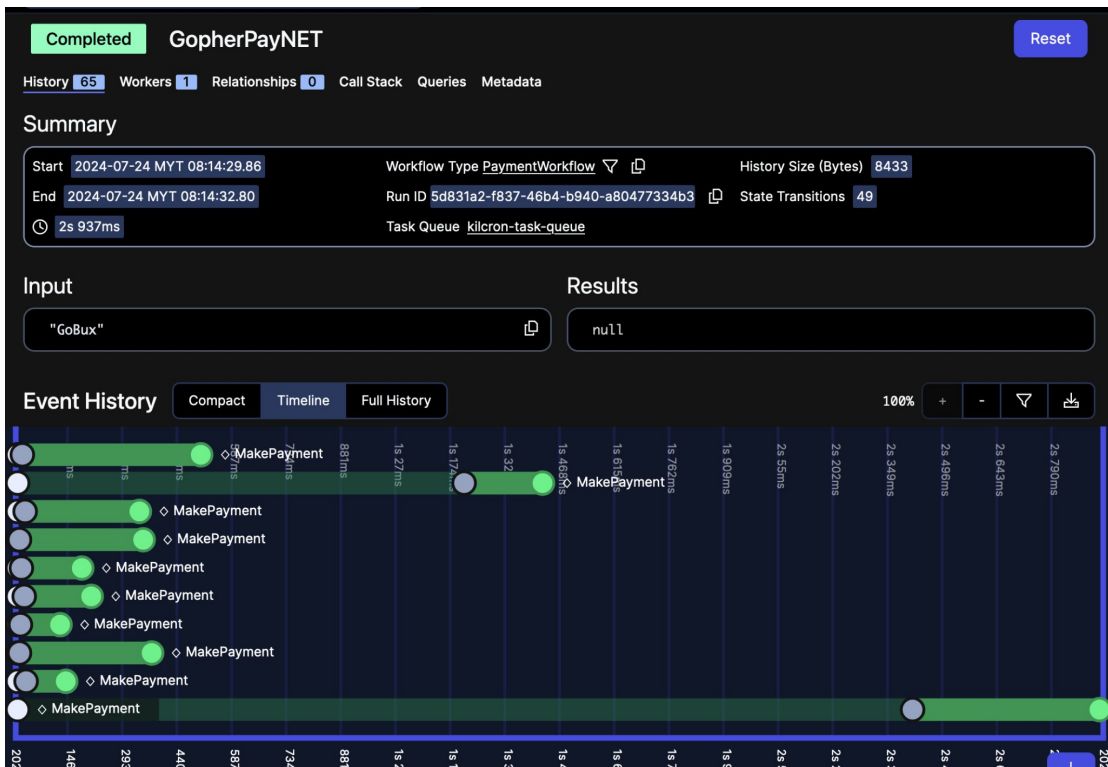
Cron - Wishful Thinking

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



Cron - Closer to Reality

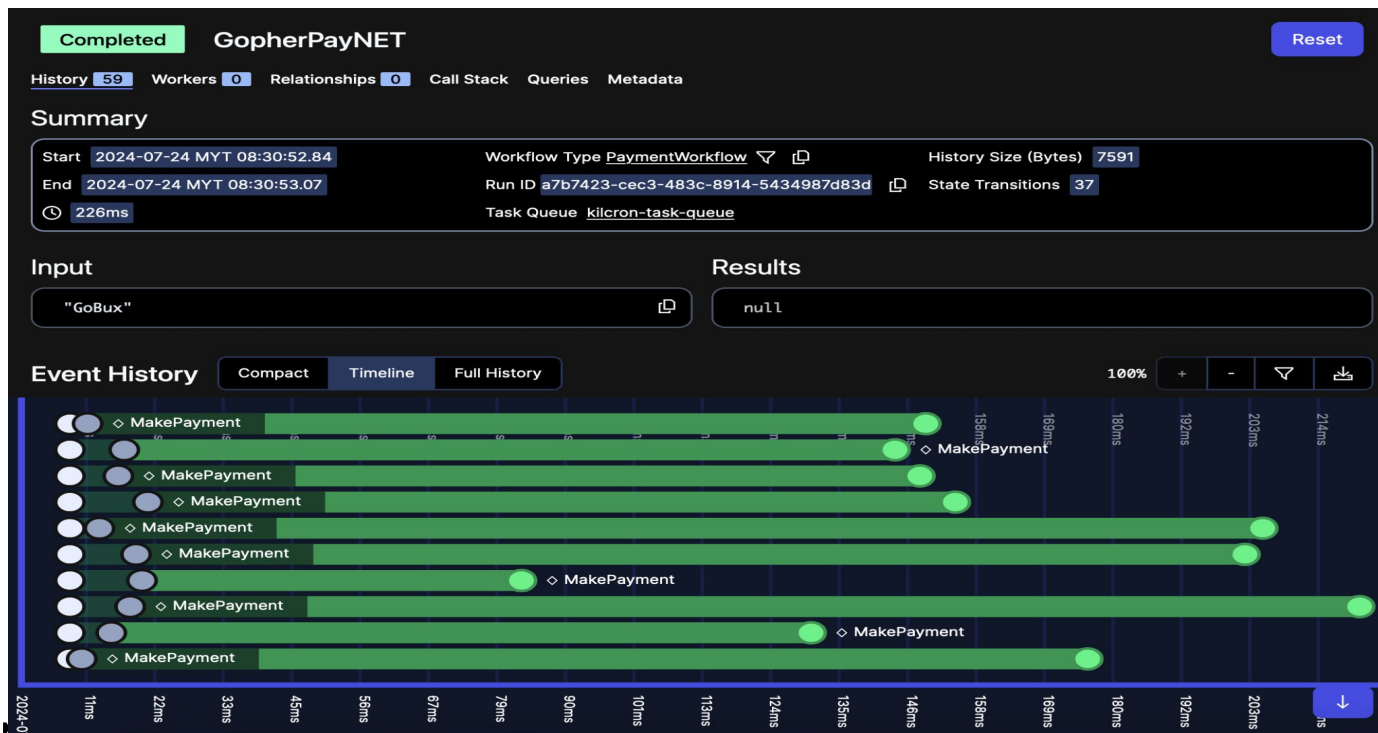
- Cron jobs have variable latency; no failure!





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: Proc
```

```
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 Summary: 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...
=====
Successful: 5
Failed: 5
Success rate: 50%
```

```
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
```

```
# 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}/${#ORDER_IDS})"

    # 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

The screenshot displays a workflow management interface with a dark theme. At the top, a timeline shows the sequence of events: '5 Activity Task Scheduled' at 20:15:54.95, '6 Activity Task Started' at 20:16:07.03, and '7 Activity Task Completed' at 20:16:10.12. The main panel is divided into three sections. The left section, titled 'Activity Task Started Attempt 4', contains details for 'Activity ID 5', 'Activity Type RunPaymentCollectionScript', 'Task Queue Name superscript-task-queue', 'Task Queue Kind TASK_QUEUE_KIND_NORMAL', an empty 'Header' object, 'Input' with the value '123456', 'Start To Close Timeout 2 minutes', 'Workflow Task Completed Event ID 4', and 'Retry Policy Initial Interval 1s'. The middle section shows 'Scheduled Event ID 5', 'Identity 89473@Michaels-MacBook-Pro14.local@', 'Request ID ff40f32c-610e-428b-8609-8d3184c39516', 'Attempt 4', and a 'Last Failure' log entry stating 'Script execution failed with exit code: 2'. The right section, titled 'Result', displays a JSON object with 'order_id', 'success', and 'output' fields, indicating a successful payment processing for OrderID 123456.

6 2025-03-31 MYT 20:16:07.03 Activity Task Started Attempt 4

5 Activity Task Scheduled 2025-03-31 MYT 20:15:54.95 6 Activity Task Started 2025-03-31 MYT 20:16:07.03 7 Activity Task Completed 2025-03-31 MYT 20:16:10.12

Activity ID 5

Activity Type RunPaymentCollectionScript

Task Queue Name superscript-task-queue

Task Queue Kind TASK_QUEUE_KIND_NORMAL

Header

{}

Input

["123456"]

Start To Close Timeout 2 minutes

Workflow Task Completed Event ID 4

Retry Policy Initial Interval 1s

Scheduled Event ID 5

Identity 89473@Michaels-MacBook-Pro14.local@

Request ID ff40f32c-610e-428b-8609-8d3184c39516

Attempt 4

Last Failure

{ "message": "Script execution failed with exit code: 2", "source": "GoSDK", "applicationFailureInfo": {} }

Worker Version ccd51f4cab66c62894a4808f6195b11d

Result

```
{  "order_id": "123456",  "success": true,  "output": "Starting payment processing for OrderID: 123456\nStarting processing step 1...\nStep 1 completed successfully: Step1 123456\nStarting processing step 2...\nStep 2 completed successfully: Step2 123456\nPayment processing completed successfully for OrderID: 123456\nCleaning up resources..."  ,  "exit_code": 0,  "execution_time": 3088862417,  "timestamp": "2025-03-31T20:16:10.125345+08:00"}
```


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

```
// This workflow wraps a potentially non-idempotent activity
func SinglePaymentCollectionWorkflow(ctx workflow.Context) error {
    logger := workflow.GetLogger(ctx)
    logger.Info(msg: "Starting SinglePaymentCollectionWorkflow")
    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, &activityResult)
    if err != nil {
        logger.Error(msg: "PaymentCollectionScript failed", err)
        return err
    }
    return nil
}
```

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

// 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"  
}
```

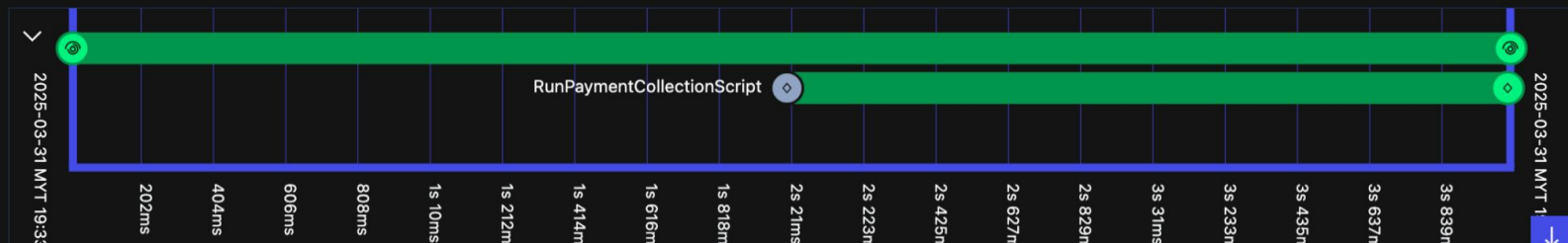


Result

```
✓ {  
  "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"  
}
```

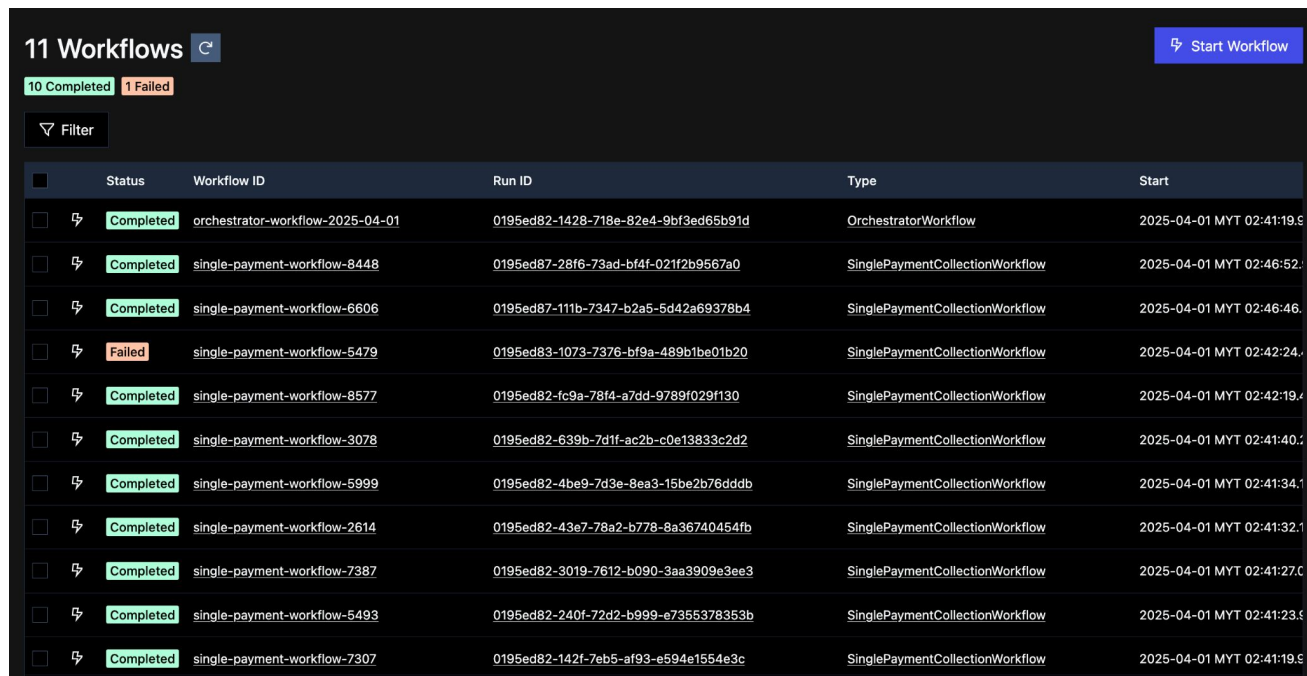


Event History



Superscript Demo

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



	Status	Workflow ID	Run ID	Type	Start
<input type="checkbox"/>	Completed	orchestrator-workflow-2025-04-01	0195ed82-1428-718e-82e4-9bf3ed65b91d	OrchestratorWorkflow	2025-04-01 MYT 02:41:19.5
<input type="checkbox"/>	Completed	single-payment-workflow-8448	0195ed87-28f6-73ad-bf4f-021f2b9567a0	SinglePaymentCollectionWorkflow	2025-04-01 MYT 02:46:52.1
<input type="checkbox"/>	Completed	single-payment-workflow-6606	0195ed87-111b-7347-b2a5-5d42a69378b4	SinglePaymentCollectionWorkflow	2025-04-01 MYT 02:46:46.6
<input type="checkbox"/>	Failed	single-payment-workflow-5479	0195ed83-1073-7376-bf9a-489b1be01b20	SinglePaymentCollectionWorkflow	2025-04-01 MYT 02:42:24.1
<input type="checkbox"/>	Completed	single-payment-workflow-8577	0195ed82-fc9a-78f4-a7dd-9789f029f130	SinglePaymentCollectionWorkflow	2025-04-01 MYT 02:42:19.4
<input type="checkbox"/>	Completed	single-payment-workflow-3078	0195ed82-639b-7d1f-ac2b-c0e13833c2d2	SinglePaymentCollectionWorkflow	2025-04-01 MYT 02:41:40.7
<input type="checkbox"/>	Completed	single-payment-workflow-5999	0195ed82-4be9-7d3e-8ea3-15be2b76dddb	SinglePaymentCollectionWorkflow	2025-04-01 MYT 02:41:34.1
<input type="checkbox"/>	Completed	single-payment-workflow-2614	0195ed82-43e7-78a2-b778-8a36740454fb	SinglePaymentCollectionWorkflow	2025-04-01 MYT 02:41:32.1
<input type="checkbox"/>	Completed	single-payment-workflow-7387	0195ed82-3019-7612-b090-3aa3909e3ee3	SinglePaymentCollectionWorkflow	2025-04-01 MYT 02:41:27.0
<input type="checkbox"/>	Completed	single-payment-workflow-5493	0195ed82-240f-72d2-b999-e7355378353b	SinglePaymentCollectionWorkflow	2025-04-01 MYT 02:41:23.5
<input type="checkbox"/>	Completed	single-payment-workflow-7307	0195ed82-142f-7eb5-af93-e594e1554e3c	SinglePaymentCollectionWorkflow	2025-04-01 MYT 02:41:19.5

Superscript - Code / Flow

- Straight-forward, composable; calls earlier SinglePaymentWorkflow

```
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 limit is reached
        // Reverting to standard TryAcquire(1) based semaphore
        if numScheduled < len(params.OrderIDs) &&
    }

    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)
```

Completed

orchestrator-workflow-2025-04-01

Reset



Current Details



Start	2025-04-01 MYT 02:41:19.91	Run ID	0195ed82-1428-718e-82e4-9bf3ed65b91d	History Size (Bytes)	22580
End	2025-04-01 MYT 02:46:53.98	Workflow Type	OrchestratorWorkflow		
🕒	5m 34s 70ms	Task Queue	superscript-task-queue		

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\nStarting processing step 1...\nStep 1 completed successfully: Step1 5493\nStarting processing step 2...\nStep 2 completed successfully: Step2 5493\nPayment processing completed successfully for OrderID: 5493\nCleaning up resources...\n",
  "exit_code": 0,
  "execution_time": 206222042,
  "timestamp": "2025-04-01T02:41:27.050155+08:00"
},
{
  "order_id": "7387",
```

Event History



			All	Compact	JSON	↓ Desc	⌵	⏸	📄	<input type="checkbox"/> 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?

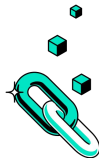
JIT vs Break glass

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.



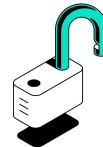
Temporary K8s Access

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>

<p>AI Agent execution using Temporal</p> <p>This demo shows a multi-turn conversation with an AI agent running inside a Temporal workflow.</p> <p> COMMUNITY VIDEO AI</p> <p> Steve Androulakis Sr. Staff Solutions Architect</p> <p>TEMPORAL</p> <p> Python</p>	<p>AgentOfCode</p> <p>An "agentic" LLM solution uses various AI models to iteratively work through Adversary of Code problems.</p> <p> COMMUNITY VIDEO MACHINE LEARNING ARTIFICIAL INTELLIGENCE OPEN</p> <p> Jason Stewig</p> <p>Python</p>	<p>kairos-cli</p> <p>kairos-cli is a snappy cli application that allows you to maintain your flow state while allowing you to open a workflow in the cloud for a richer experience and deeper analysis.</p> <p> COMMUNITY OPEN</p> <p> Sapreth Sobhana</p> <p>GENERAL</p> <p> Go</p>
<p>Indeed Workflow Framework (iWF)</p> <p>iWF (Indeed Workflow Framework) is a framework built on top of Temporal that simplifies the development, management, and orchestration of long-running, asynchronous workflows. It provides clear abstractions so you can focus on business logic.</p> <p> COMMUNITY WORKFLOW TEMPORAL ABSTRACTIONS</p> <p> Indeed Engineering Team</p> <p>INDEED</p> <p>    Python</p>	<p>Rojak</p> <p>A library for building highly durable and scalable multi-agent orchestrations.</p> <p> COMMUNITY MACHINE LEARNING AGENTS</p> <p> Jerom Lim</p> <p>Python</p>	<p>Go Code Generation with Temporal & Protobufs</p> <p>Generates code for Temporal workflows from Protobuf definitions, automating client and worker creation. It simplifies development by converting structured workflow definitions into runnable code, reducing manual effort and ensuring consistency.</p> <p> COMMUNITY CODE GENERATION PROTobuf</p> <p> Chris Ludden</p> <p>TEMPORAL</p> <p> Go</p>
<p>ProjectX</p> <p>A comprehensive full-stack template designed to simplify the development of scalable and resilient applications using React and Temporal.</p> <p> COMMUNITY TEMPORAL RELATED APP REACT</p> <p> JD Nicholls Founder</p> <p>MARKETX</p> <p> TypeScript</p>	<p>Cross-Language Data Processing Service with Temporal</p> <p>A practical demonstration of building a robust, scalable, and language-agnostic data processing pipeline using Temporal. Seamlessly orchestrates workflows across Go, Python, and TypeScript—perfectly suited for polyglot development teams.</p> <p> COMMUNITY PREVIEW DATA PROCESSING</p> <p> Bar Moshe</p> <p>   TypeScript</p>	<p>Shadow Shop Sample</p> <p>A sample application that adds integration with Stripe for payment processing and Temporal for Durable Execution.</p> <p> COMMUNITY PAYMENT PROCESSING SAMPLE APPLICATION E-COMMERCE</p> <p> Cedric Philip Staff Developer Advocate</p> <p>STRIPE</p> <p> .NET</p>

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/>



Cloudflare Workflows (Typescript)

<https://cloudflare.com/>



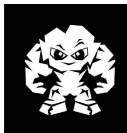
Inngest (JS)

<https://www.inngest.com/>

The DBOS logo, with the letters "DBOS" in white bold capital letters on a black rectangular background.

DBOS (Typescript, Python)

<https://www.dbos.dev/>



Golem Cloud (WASM)

<https://www.golem.cloud/>



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

<https://littlehorse.io/>

07

Q&A

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



Thank you

