


_ / TEMPORAL REPLAY 2026

From Bottlenecks to Self-Service

How  duolingo Built Workflow-as-a-Service
with Temporal Nexus

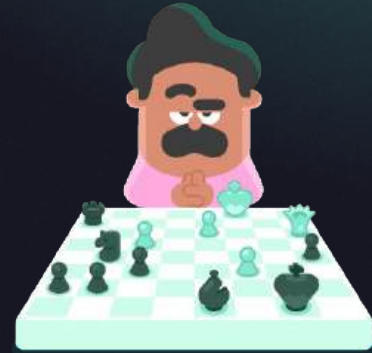
Zhihao Wang

Staff Software Engineer @ Duolingo



Develop the best education in the world and make it universally available

Duolingo is the most popular mobile learning platform and the most downloaded education app worldwide. The app makes learning new languages, math, music and chess fun with bite-sized lessons that feel like playing a game.



Zhihao Wang



Staff Software Engineer

Cloud Operations Team at Duolingo

Temporal 0→1

Turning Temporal into a true platform at Duolingo and educating engineers on how to use it effectively.

Infrastructure & Developer Productivity

Making infrastructure easier to operate through better tooling and automation

1060+ Day Streak

Japanese, French, German, Chess, and more.



What We'll Cover

01

What is Temporal Nexus?

Cross-namespaces Communication

02

Duolingo + Temporal

How Cloud Operations team uses Temporal

03

Building Self-Service

Architecture, access control, and the portal

04

Evolving the Platform

What we built beyond the portal

05

Demo

Live walkthrough

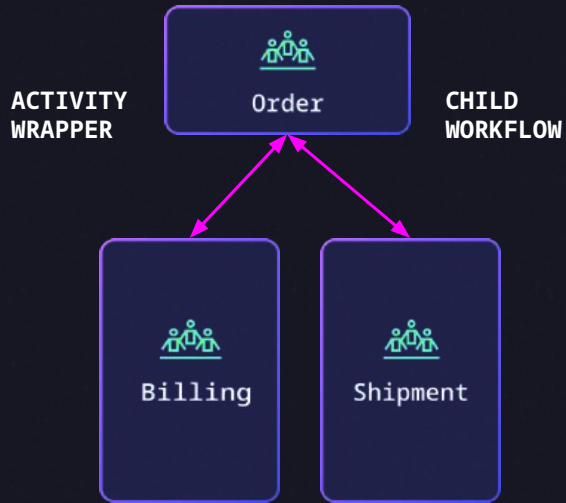
06

Results & What's Next

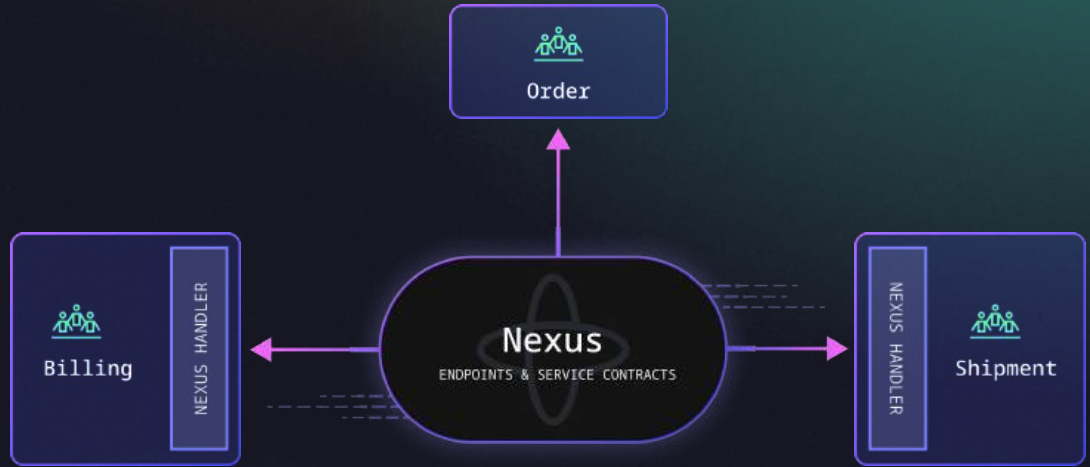
Impact, adoption, and the road ahead



Connect Workflows Across Isolated Namespaces



Before



Now



Cloud Operations: Scaling Through Automation

OUR TEAM

- Manage Duolingo's cloud infrastructures on AWS
- Enable product engineers to ship reliably at scale
- Perform high-risk, complex operations product engineers shouldn't run directly
- Provide tools, guardrails, and operational expertise



Cloud Operations: Scaling Through Automation

OUR TEAM

- Manage Duolingo's cloud infrastructures on AWS
- Enable product engineers to ship reliably at scale
- **Perform high-risk, complex operations product engineers shouldn't run directly**
- Provide tools, guardrails, and operational expertise

CHALLENGES

- Requests arrived via Slack or Jira
- Manual copy-and-paste of parameters into executions
- No validation for missing, incorrect, or mistyped inputs
- Manual execution using elevated AWS IAM roles



Cloud Operations: Scaling Through Automation

OUR TEAM

- Manage Duolingo's cloud infrastructures on AWS
- Enable product engineers to ship reliably at scale
- **Perform high-risk, complex operations product engineers shouldn't run directly**
- Provide tools, guardrails, and operational expertise

CHALLENGES

- Requests arrived via Slack or Jira
- Manual copy-and-paste of parameters into executions
- No validation for missing, incorrect, or mistyped inputs
- ~~Manual execution using elevated AWS IAM roles~~



Cloud Operations: Scaling Through Automation

OUR TEAM

- Manage Duolingo's cloud infrastructures on AWS
- Enable product engineers to ship reliably at scale
- **Perform high-risk, complex operations product engineers shouldn't run directly**
- Provide tools, guardrails, and operational expertise

CHALLENGES

- Requests arrived via Slack or Jira
- Manual copy-and-paste of parameters into executions
- No validation for missing, incorrect, or mistyped inputs
- ~~Manual execution using elevated AWS IAM roles~~

→ This can frequently be the most significant bottleneck on engineering productivity.



Cloud Operations: Scaling Through Automation

EARLY SOLUTION

- Built Temporal workflows for repeatable infrastructure tasks
- Workflows executed in a Cloud Ops team-based Temporal namespace
- Strict access control on workflow execution
- Removed the need for manual execution using elevated AWS IAM roles

CHALLENGES

- Requests arrived via Slack or Jira
- Manual copy-and-paste of parameters into executions
- No validation for missing, incorrect, or mistyped inputs
- ~~Manual execution using elevated AWS IAM roles~~

→ This can frequently be the most significant bottleneck on engineering productivity.



Zhihao Wang 🔥 8:40 PM

Hey can you specify the the AWS region please? There are clusters with the same name in both us-east-1 and us-east-2.



Zhihao Wang 🔥 8:37 PM

Hey can you confirm that we can delete this database now?



Zhihao Wang 🔥 8:38 PM

Hey, it seems the RDS cluster identifier you provided in the ticket is incorrect, can you double check?



What We Actually Wanted

- **Engineers can trigger workflows themselves**
 - This implies no manual copy-and-paste by Cloud Operations team
- **Inputs validated before execution or before core logic**
- **Cloud Operations team keeps control through approvals**



What We Actually Wanted

- Engineers can trigger workflows themselves
 - This implies no manual copy-and-paste by Cloud Operations team
- Inputs validated before execution or before core logic
- **Cloud Operations team keeps control through approvals** → **Signals!**

```
1 @workflow.signal
2 async def confirm_deletion(self, params) → None:
3     self._confirm_deletion = True
4
5 @workflow.run
6 async def run(self, params) → dict:
7     ...
8     await workflow.wait_condition(lambda: self._confirm_deletion or self._exit)
9     ...
10
```



What We Actually Wanted

- **Engineers can trigger workflows themselves** → Write Access to Namespaces
 - This implies no manual copy-and-paste by Cloud Operations team
- Inputs validated before execution or before core logic
- **Cloud Operations team keeps control through approvals** → **Signals!**

```
1 @workflow.signal
2 async def confirm_deletion(self, params) → None:
3     self._confirm_deletion = True
4
5 @workflow.run
6 async def run(self, params) → dict:
7     ...
8     await workflow.wait_condition(lambda: self._confirm_deletion or self._exit)
9     ...
10
```



What We Actually Wanted

- **Engineers can trigger workflows themselves** → Write Access to Namespaces
 - This implies no manual copy-and-paste by Cloud Operations team
- Inputs validated before execution or before core logic
- **Cloud Operations team keeps control through approvals**
 - **Other engineers cannot approve the workflows**

```
1 @workflow.signal
2 async def confirm_deletion(self, params) → None:
3     self._confirm_deletion = True
4
5 @workflow.run
6 async def run(self, params) → dict:
7     ...
8     await workflow.wait_condition(lambda: self._confirm_deletion or self._exit)
9     ...
10
```



What We Actually Wanted

- **Engineers can trigger workflows themselves** → Write Access to Namespaces
 - This implies no manual copy-and-paste by Cloud Operations team
- Inputs validated before execution or before core logic
- **Cloud Operations team keeps control through approvals**
 - **Other engineers cannot approve the workflows** → No Write Access to Namespaces



```
1 @workflow.signal
2 async def confirm_deletion(self, params) → None:
3     self._confirm_deletion = True
4
5 @workflow.run
6 async def run(self, params) → dict:
7     ...
8     await workflow.wait_condition(lambda: self._confirm_deletion or self._exit)
9     ...
10
```



The Hardest Part: Access Control

- **Engineers can trigger workflows themselves** → **Write Access to Namespaces**
 - This implies no manual copy-and-paste by Cloud Operations team
- Inputs validated before execution or before core logic
- **Cloud Operations team keeps control through approvals**
 - **Other engineers cannot approve the workflows** → **No Write Access to Namespaces**



HARD CONSTRAINT

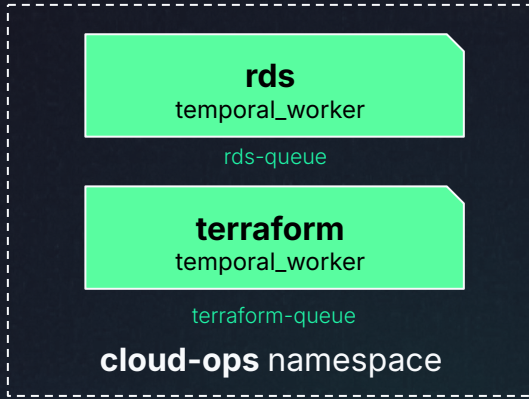
cloud-ops namespace access must remain restricted.

THE QUESTION NOW BECOMES

How do we expose these workflows safely?



The Solution: Cross-namespace Communication



We do **not** want to change anything within existing workers within cloud-ops namespace.



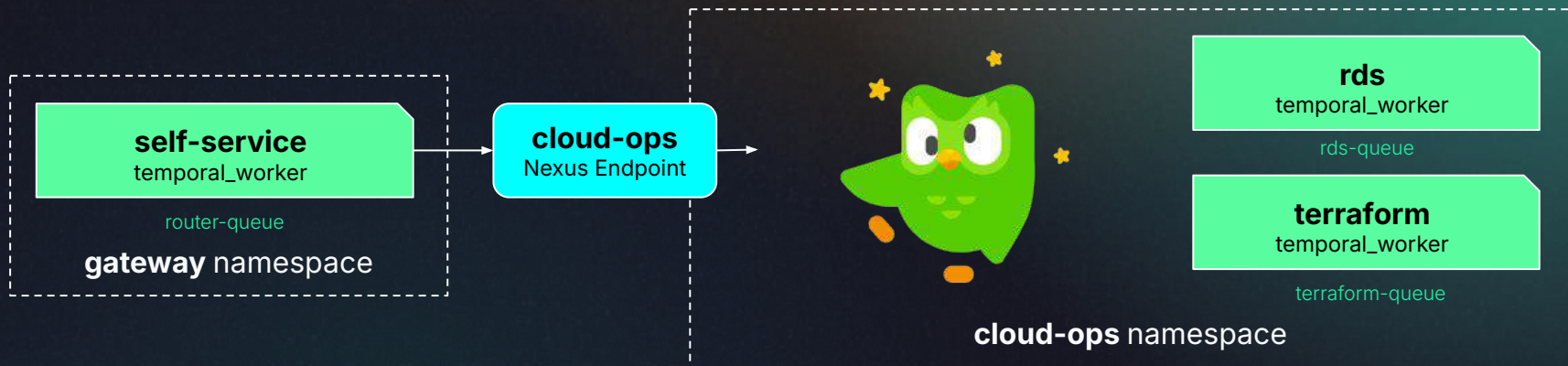
The Solution: Cross-namespace Communication



We need a central **gateway** namespace that everyone has write access to.



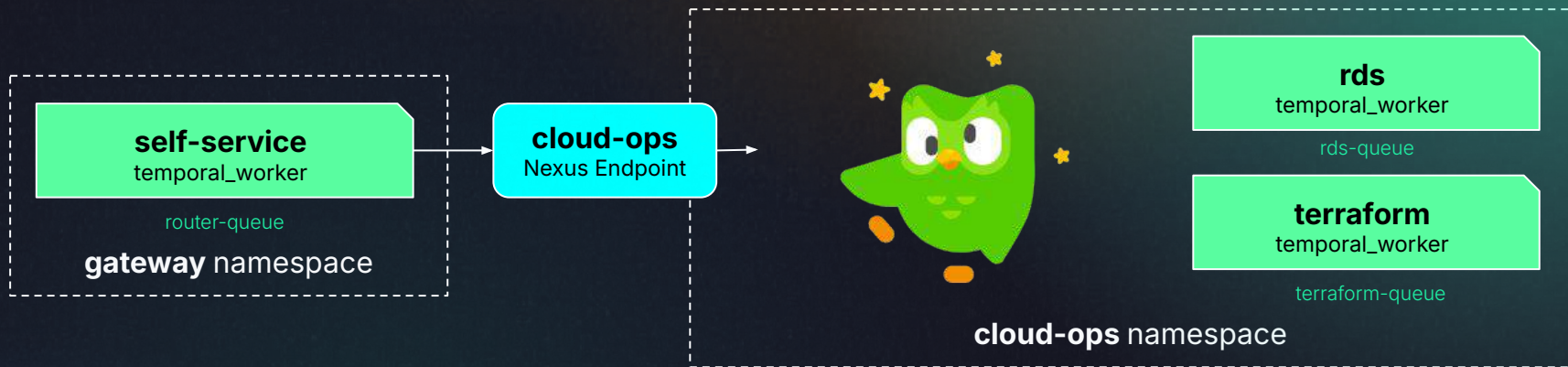
The Solution: Temporal Nexus




Wait... one Nexus endpoint has only **one** corresponding queue...



The Solution: Temporal Nexus



Wait... one Nexus endpoint has only **one** corresponding queue...

 **Zhihao** Feb 17th, 2025 at 4:27 PM
What is the rationale behind defining a single task queue when defining the Nexus endpoints? Would it be more flexible if operations were bound to a task queue, allowing many operations to share one endpoint?
35 replies

 **Zhihao** Feb 26th, 2025 at 2:34 PM
Thank you for your time today!
image.png



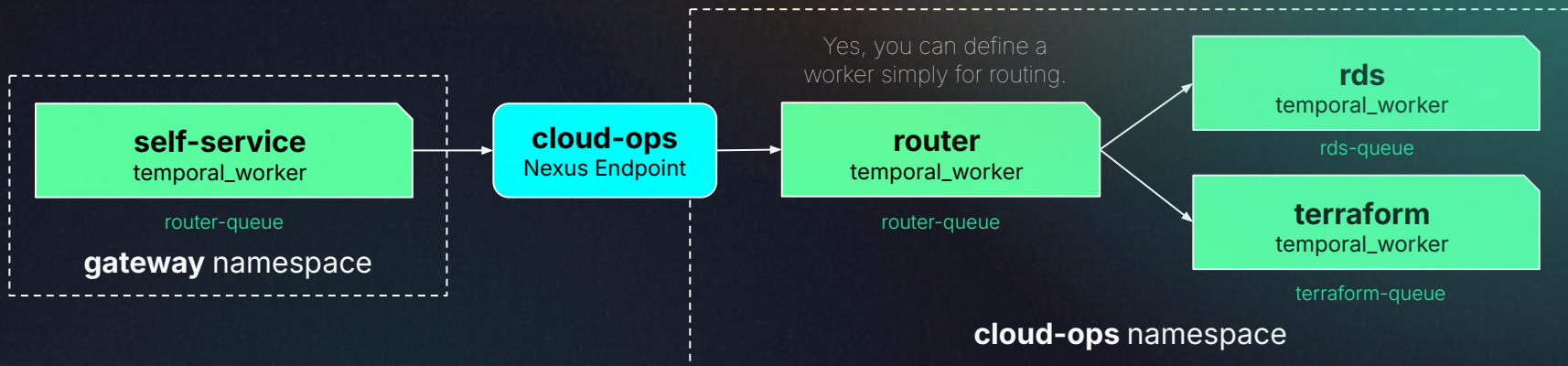
The screenshot shows a diagram of a **cloud-ops (team) namespace**. It features a **Nexus worker** connected to a **Stack?** component. The **Stack?** component is further connected to **rds worker** and **Terraform worker** components. A **gateway namespace** is also indicated at the bottom left of the diagram.

#nexus thread started on 2/17/2025

Zoom meeting on 2/26/2025



The Solution: Temporal Nexus



	gateway namespace	cloud-ops namespace
Engineering Duos*	write access	read access
Cloud Operations Team	write access	write access

* At Duolingo, an individual employee is called a Duo.



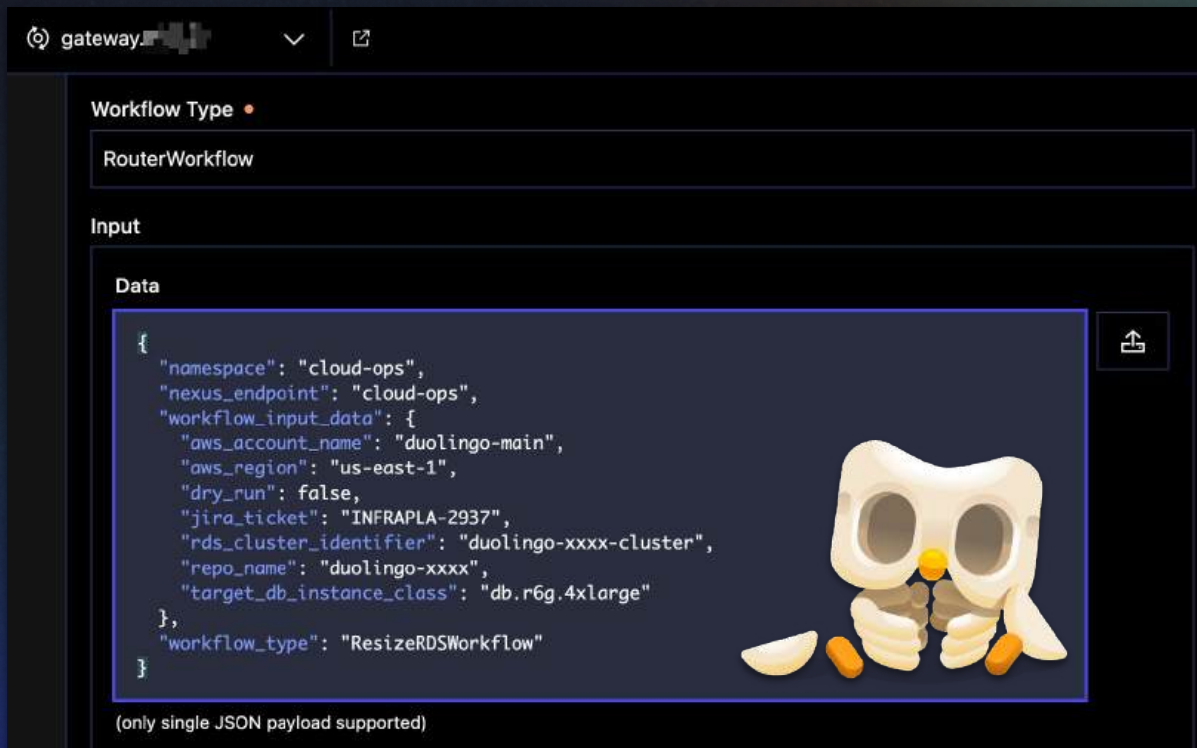
Have we solved all the challenges?

- Requests arrived via Slack or Jira
- Manual copy-and-paste of parameters into executions
- No validation for missing, incorrect, or mistyped inputs
- ~~Manual execution using elevated AWS IAM roles~~



We haven't won yet...

- Temporal UI requires **raw JSON input**, which is powerful but not a user-friendly way to submit infrastructure requests.



The screenshot shows the Temporal UI interface for configuring a workflow. At the top, there's a breadcrumb 'gateway' and a dropdown menu. Below that, the 'Workflow Type' is set to 'RouterWorkflow'. The 'Input' section is expanded to show the 'Data' field, which contains a raw JSON payload. The JSON is highlighted with a blue background. To the right of the JSON, there is a small owl mascot icon. Below the JSON, there is a note: '(only single JSON payload supported)'. A share icon is visible to the right of the JSON editor.

```
{
  "namespace": "cloud-ops",
  "nexus_endpoint": "cloud-ops",
  "workflow_input_data": {
    "aws_account_name": "duolingo-main",
    "aws_region": "us-east-1",
    "dry_run": false,
    "jira_ticket": "INFRAPLA-2937",
    "rds_cluster_identifier": "duolingo-xxxx-cluster",
    "repo_name": "duolingo-xxxx",
    "target_db_instance_class": "db.r6g.4xlarge"
  },
  "workflow_type": "ResizeRDSWorkflow"
}
```

(only single JSON payload supported)



What We Actually Wanted

- Static, schema-driven website with minimal maintenance overhead
 - Declarative workflow definitions (JSON / YAML) that automatically render in the UI
- **Basic input validation to catch errors before execution**
- Ability to trigger workflows directly from the website



What We Actually Wanted

- Static, schema-driven website with minimal maintenance overhead
 - Declarative workflow definitions (JSON / YAML) that automatically render in the UI
- **Basic input validation to catch errors before execution**
- Ability to trigger workflows directly from the website via HTTPS

→ Open Source framework

[rjsf-team/react-jsonschema-form](https://github.com/rjsf-team/react-jsonschema-form)

The screenshot shows the GitHub repository page for `rjsf-team / react-jsonschema-form`. It includes the repository name, navigation tabs (Code, Issues, Pull requests, Agents, Discussions, More), and statistics: 158 watchers, 171 issues, and 11 pull requests. The description reads: "A React component for building Web forms from JSON Schema." It also lists the Apache-2.0 license, a code of conduct, and contributing guidelines. At the bottom, it shows 15.6k stars, 2.3k forks, 158 watching, 9 branches, and 414 tags.

This screenshot illustrates the mapping from JSON Schema to a web form. On the left, a JSON Schema for a registration form is shown, defining fields like firstName, lastName, age, bio, and password. In the middle, a second schema shows how these fields are mapped to specific UI widgets like 'textinput', 'textarea', and 'password'. On the right, the rendered form is displayed, showing a registration form with fields for First name, Last name, Age, Bio, and Password, along with a Submit button and a strength hint.



The Tricky Part: Triggering workflow via HTTPS

- Static, schema-driven website with minimal maintenance overhead
 - Declarative workflow definitions (JSON / YAML) that automatically render in the UI
- Basic input validation to catch errors before execution
- **Ability to trigger workflows directly from the website via HTTPS**

Temporal Workflows or Nexus Operations, *as of 02/2026*, can only be triggered by SDK/CLI/UI.



The Tricky Part: Triggering workflow via HTTPS

- Static, schema-driven website with minimal maintenance overhead
 - Declarative workflow definitions (JSON / YAML) that automatically render in the UI
- Basic input validation to catch errors before execution
- **Ability to trigger workflows directly from the website via HTTPS**

"Build a microservice!"

System Design Spec: Temporal Gateway

Author(s): Zhihao Wang

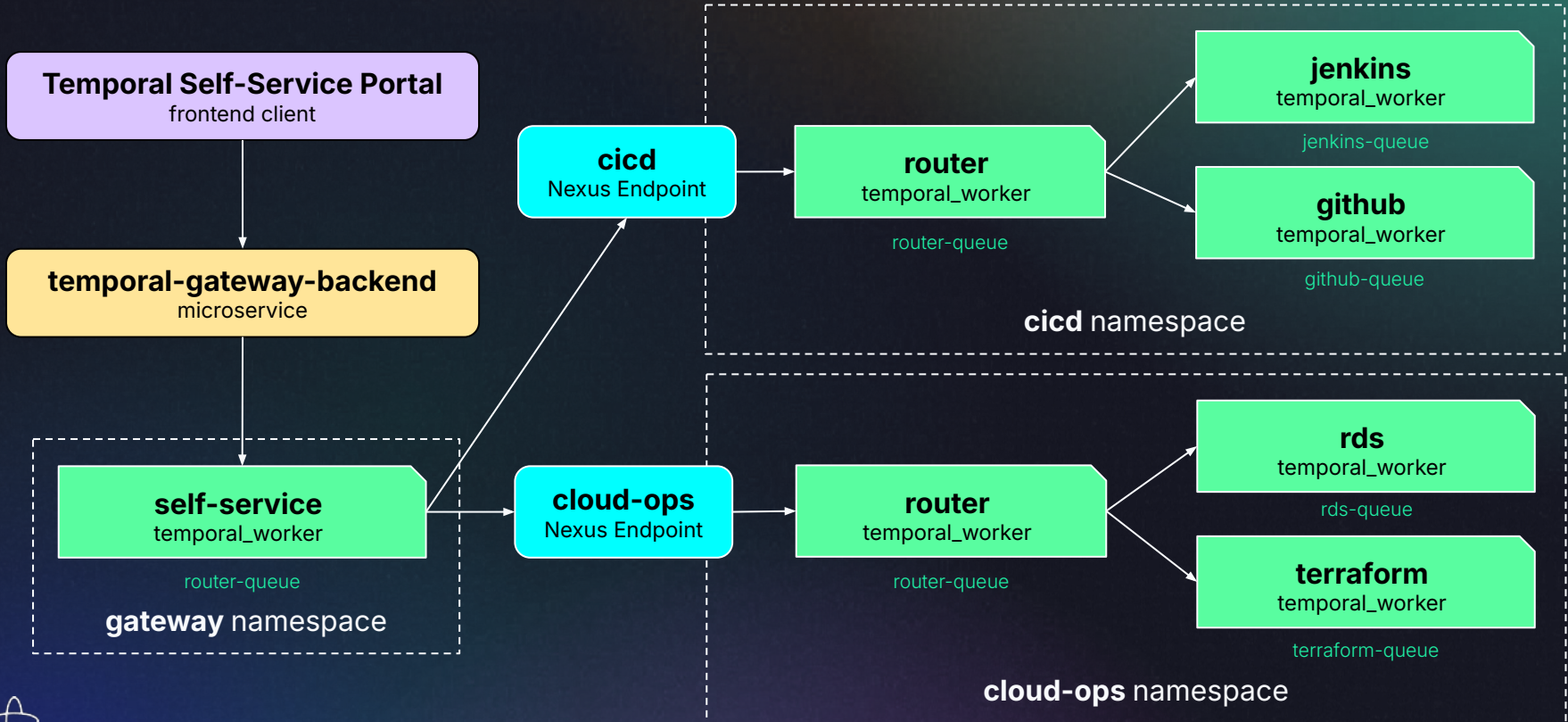
Date: Mar 12, 2025

Background

Triggering a Temporal workflow requires a client. Doing this directly from a React frontend would require [Next.js](#). The most scalable and backend-friendly solution is to introduce a simple API layer as a "gateway" to facilitate triggering workflows via HTTP requests. This will be an internal service, meaning it will be only accessible within the internal network and not exposed to external users. **This would enable all types of clients (Duos, Jenkins, services, etc) to trigger Temporal workflows via HTTP requests using user or service account API keys.**



Temporal Self-Service Platform Architecture



_/ SHOW, DON'T TELL

Gateway Namespace

self-service
temporal_worker

router-queue

gateway namespace

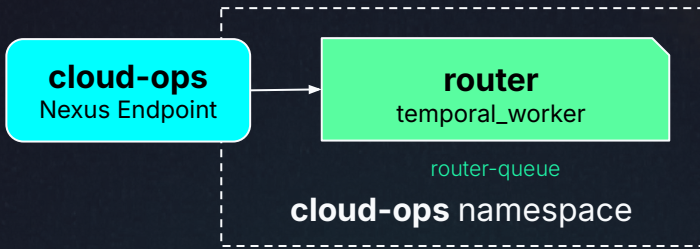
```
1 @dataclass
2 class RouterWorkflowParams:
3     """
4     Parameters required for running the RouterWorkflow.
5     """
6
7     workflow_type: str
8     workflow_input_data: Dict[str, Any]
9     namespace: str
10    nexus_endpoint: str
11    duo_email: str
```

```
1 @workflow.defn
2 class RouterWorkflow:
3     """
4     RouterWorkflow is a Temporal workflow that triggers a workflow
5     in other namespaces via Temporal Nexus.
6
7     This workflow resides in the gateway namespace and can be triggered by all Duos,
8     while the workflow in the target namespace typically has more restricted access.
9     """
10
11    @workflow.run
12    async def run(self, params: RouterWorkflowParams) → RouterWorkflowResponse:
13        # Set DuoEmail search attribute
14        workflow.upsert_search_attributes({"DuoEmail": [params.duo_email]})
15
16        # Convention: Use endpoint name as the service name
17        nexus_client = workflow.create_nexus_client(
18            service=params.nexus_endpoint, endpoint=params.nexus_endpoint
19        )
20
21        # Convention: Use WorkflowName as the operation name
22        operation_name = params.workflow_type
23
24        # Execute the Nexus operation and wait for the result
25        operation_handle = await nexus_client.start_operation(
26            operation=operation_name,
27            input=params.workflow_input_data,
28            schedule_to_close_timeout=timedelta(days=30),
29        )
30        workflow_response: Dict[str, Any] = await operation_handle
31
32    return RouterWorkflowResponse(
33        workflow_type=params.workflow_type,
```



_/ SHOW, DON'T TELL

cloud-ops Namespace



```
1 // NewWorkflowNexusOperation is a convenience wrapper for
  NewWorkflowNexusOperationWithParamsConverter when no paramsConverter is needed.
2 func NewWorkflowNexusOperation[T, R any](
3     workflowName, taskQueue string,
4     idGenerator func(T) string,
5 ) (nexus.Operation[T, R], error) {
6     return NewWorkflowNexusOperationWithParamsConverter[T, T, R](workflowName,
7         taskQueue, idGenerator, nil)
7 }
```

```
1 var deleteRDSWorkflowOperation, _ = base.NewWorkflowNexusOperation[rds.DeleteRDSWorkflowParams, any](
2     "DeleteRDSWorkflow",
3     "datastore-queue", → invoke the workflow on a different queue in the same namespace
4     func(input rds.DeleteRDSWorkflowParams) string {
5         return stringutils.AddDefaultTimestamp("DeleteRDSWorkflow-" + input.RDSIdentifier)
6     },
7 )
```

When we first implemented router workers, Temporal Nexus was not yet supported in the Python SDK, so we wrote them in Go.



Temporal Self-Service Portal

Temporal Self-Service Portal

frontend client



temporal-gateway-backend

microservice

```
1 export interface TemporalWorkflowInput {
2   duoEmail: string;
3   inputData?: Record<string, unknown>;
4   namespace: string;
5   region: string;
6   taskQueue: string;
7   workflowType: string;
8 }
9
10 export const triggerWorkflow = async (
11   temporalWorkflowInput: TemporalWorkflowInput,
12   env: Environment,
13 ): Promise<WorkflowInfo> =>
14   post(`${getGatewayUrl(env)}`, temporalWorkflowInput);
15
```

Required for publishing workflows to the self-service portal. Defined using JSON configurations.



_/ SHOW, DON'T TELL

Temporal Self-Service Portal

Temporal Self-Service Portal

frontend client



temporal-gateway-backend

microservice

```
1 const generateTemporalWorkflowInput = (...) => {
2   const useStageBackend = env === "stage" || env === "local";
3
4   return {
5     duoEmail,
6     inputData: {
7       duo_email: duoEmail,
8       namespace: useStageBackend ? stageNamespace : namespace,
9       nexus_endpoint: useStageBackend ? stageNexusEndpoint : nexusEndpoint,
10      workflow_input_data: {
11        ...formData,
12        duo_email: duoEmail,
13      },
14      workflow_type: workflowType,
15    },
16    namespace: useStageBackend ? "gateway-stage" : "gateway",
17    region,
18    taskQueue: "router-queue",
19    // append uuid to workflowId to avoid collisions
20    workflowId: `${workflowType}-${uuidv4()}`,
21    workflowType: "RouterWorkflow",
22  };
23 };
```

Engineers fill out the form and submit it. That's it.



Temporal Self-Service Portal

Workflow publishers are also users of the self-service portal, so publishing new workflows must be simple and frictionless.

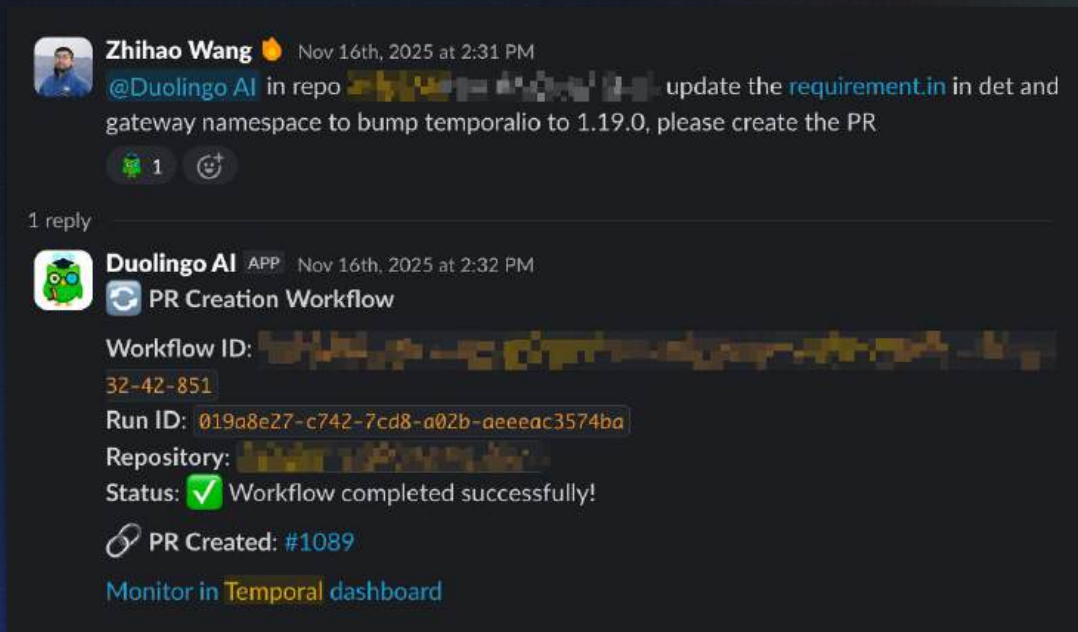
```
1 {
2   "name": "Delete DynamoDB Table",
3   "workflow_type": "DeleteDynamoWorkflow",
4   "task_queue": "datastore-queue",
5   "region": "us-east-1",
6   "namespace": "cloud-ops",
7   "stage_namespace": "cloud-ops-stage",
8   "nexus_endpoint": "cloud-ops",
9   "stage_nexus_endpoint": "cloud-ops-stage",
10  "schema": {
11    "title": "Delete DynamoDB Table",
12    "description": "Delete a DynamoDB table",
13    "type": "object",
14    "required": [
15      "aws_account_name",
16      "aws_region",
17      "table_name"
18    ],
```

```
19    "properties": {
20      "aws_account_name": {
21        "$ref": "#/definitions/aws_account_name"
22      },
23      "aws_region": {
24        "$ref": "#/definitions/aws_region",
25        "default": "us-east-1"
26      },
27      "table_name": {
28        "description": "The DynamoDB table to delete",
29        "title": "Table Name",
30        "type": "string"
31      },
32      "repo_name": {
33        "type": "string",
34        "title": "GitHub Repository Name"
35      },
36      "dry_run": {
37        "$ref": "#/definitions/dry_run"
38      }
39    }
40  }
41 }
```



Workflow-As-Service

- Teams build reusable, durable workflows that can be shared across teams via Nexus/HTTPS
 - Clients are not limited to the frontend portal: Slack bots, microservices, and more.



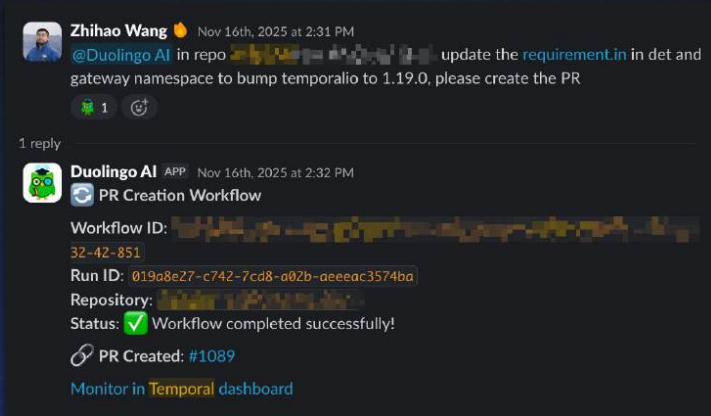
The screenshot shows a Slack conversation. The top message is from Zhihao Wang, dated Nov 16th, 2025 at 2:31 PM. The message content is: "@Duolingo AI in repo [redacted] update the [requirement.in](#) in det and gateway namespace to bump temporalio to 1.19.0, please create the PR". Below the message are 1 reaction and a plus icon for more reactions.

Below the message is a reply from Duolingo AI, dated Nov 16th, 2025 at 2:32 PM. The reply content is: "PR Creation Workflow". Below this is a list of workflow details: "Workflow ID: [redacted]", "Run ID: 019a8e27-c742-7cd8-a02b-aeaac3574ba", "Repository: [redacted]", "Status: Workflow completed successfully!", and "PR Created: #1089". At the bottom of the reply is a link: "Monitor in Temporal dashboard".



Workflow-As-Service

- **Teams build reusable, durable workflows that can be shared across teams via Nexus/HTTPS**
 - Clients are not limited to the frontend portal: Slack bots, microservices, and more.
- **Beyond shared utilities and libraries, Nexus lets us share workflows as reusable services while hiding implementation details and complex configurations.**
 - We've abstracted Slack notification functionality into shared Nexus operations, so teams don't need to install Slack SDKs in their workers or manage Slack API keys and integrations.



Dynamic Fields

Engineers are lazy. Search beats copy and paste every time.

RDS Identifier *
RDS cluster or instance identifier

17 of 480 options

- infra-**test**-tracker-db-**stage**-1
- infra-**test**-tracker-db-**stage**-2
- infra-sec-**testing**-**stage**-aurora-1
- infra-**test**-tracker-db-**stage**-cluster
- infra-sec-**testing**-**stage**-aurora-cluster
- det-orchestrator-**stage**-1
- det-orchestrator-**stage**-cluster

Setup Temporal Nexus Router

Production Temporal Namespace *

The production Temporal Cloud namespace (excludes stage)

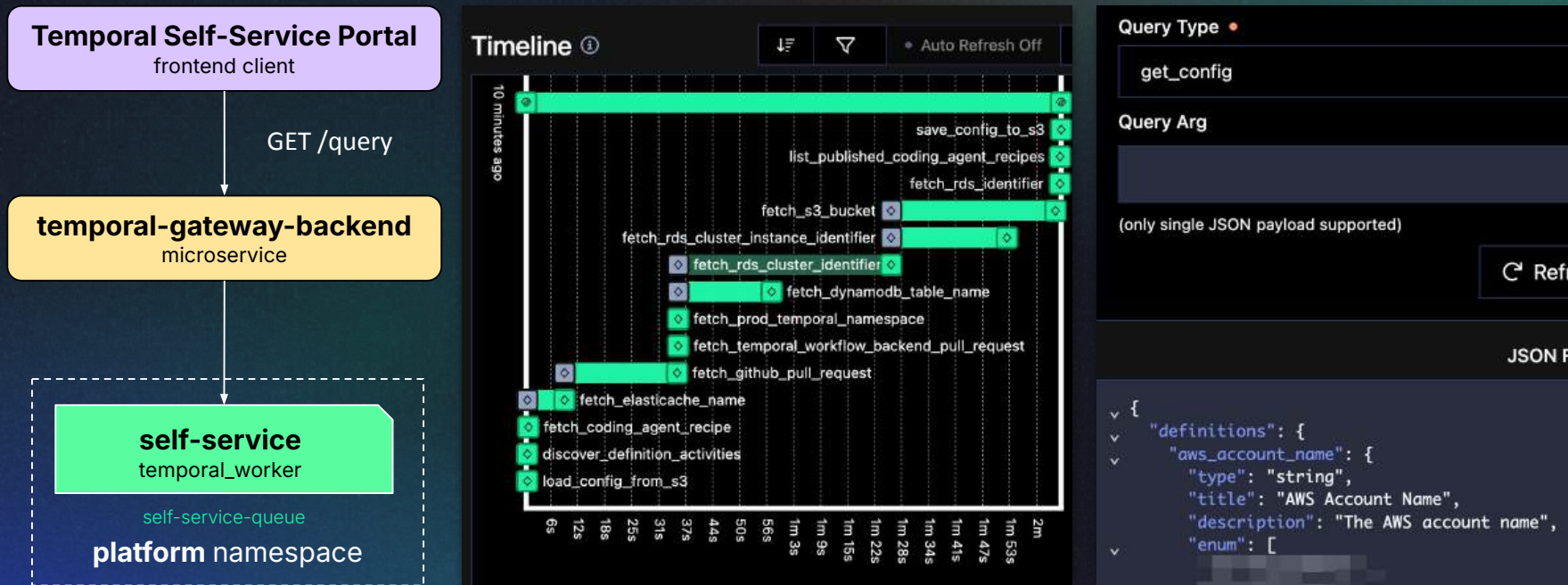
3 of 26 options

- cloud-ops
- client-architecture
- video-call



Dynamic Fields

This is powered by Temporal workflows and Queries exposed via existing gateway microservice.



Self-Service AI Assistant

Temporal-powered AI assistant that helps users choose the right workflow from natural language.

I want to upgrade zhihao-test Redis cache cluster in zhihao-sandbox AWS account to Valkey 8.0. Oh it's in us-east-2 region!

Suggested Workflow: 93% confidence

[Upgrade Redis to Valkey](#)

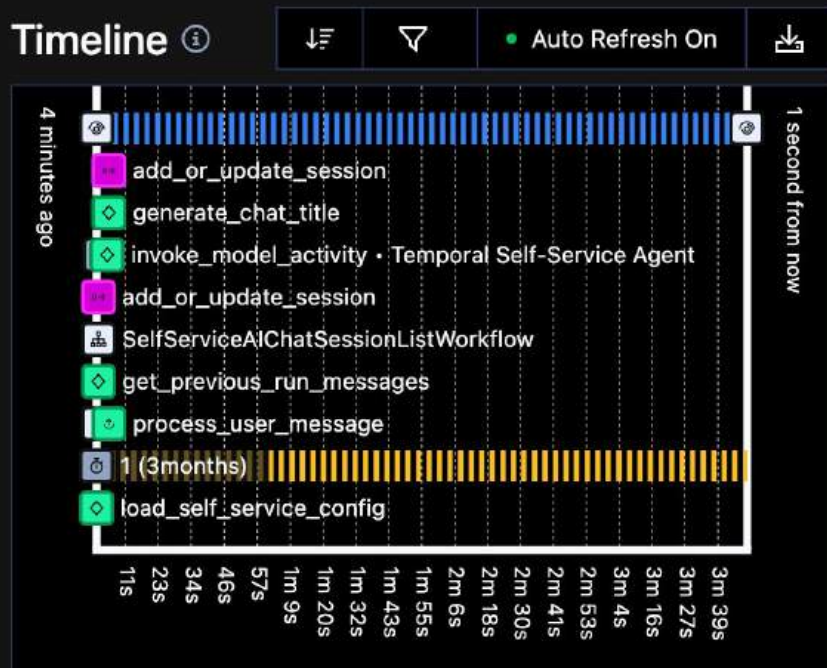
RedisUpgradeValkeyWorkflow

Pre-filled parameters:

- aws_account_name : zhihao-sandbox
- aws_region : us-east-2
- redis_name : zhihao-test
- target_engine_version : 8.0

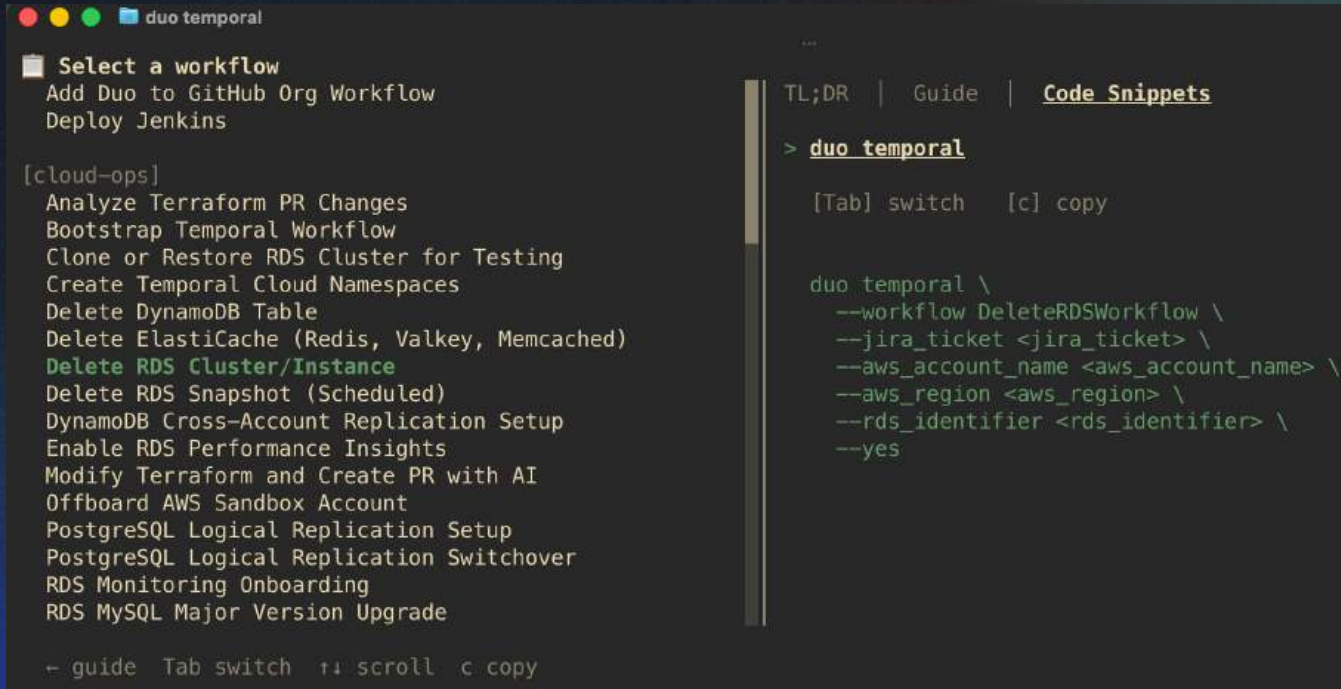
[RedisUpgradeValkeyWorkflow →](#)

Timeline



duo temporal cli

It offers the same experience as the self-service portal, and the code was generated with AI.



```
duo temporal
Select a workflow
Add Duo to GitHub Org Workflow
Deploy Jenkins

[cloud-ops]
Analyze Terraform PR Changes
Bootstrap Temporal Workflow
Clone or Restore RDS Cluster for Testing
Create Temporal Cloud Namespaces
Delete DynamoDB Table
Delete ElastiCache (Redis, Valkey, Memcached)
Delete RDS Cluster/Instance
Delete RDS Snapshot (Scheduled)
DynamoDB Cross-Account Replication Setup
Enable RDS Performance Insights
Modify Terraform and Create PR with AI
Offboard AWS Sandbox Account
PostgreSQL Logical Replication Setup
PostgreSQL Logical Replication Switchover
RDS Monitoring Onboarding
RDS MySQL Major Version Upgrade

← guide Tab switch ↑↓ scroll c copy
```

```
TL;DR | Guide | Code Snippets
> duo temporal
[Tab] switch [c] copy

duo temporal \
  --workflow DeleteRDSWorkflow \
  --jira_ticket <jira_ticket> \
  --aws_account_name <aws_account_name> \
  --aws_region <aws_region> \
  --rds_identifier <rds_identifier> \
  --yes
```



Code Snippets

"How do I trigger a Temporal workflow from Jenkins, a local script, or a microservice?"

The screenshot shows a documentation page with a dark theme. At the top, there are two tabs: 'Guide' and 'Code Snippets', with 'Code Snippets' being the active tab. Below the tabs, there are two main options for triggering a workflow:

- No Write Access to cloud-ops**: Routes via gateway-stage → cloud-ops
- Write Access to cloud-ops**: Calls cloud-ops directly

Below these options, there is a row of filters: 'duo temporal' (selected), 'Python (httpx)', 'Python (requests)', and 'cURL'. Underneath the filters is a code editor area with a 'Copy' button. The code in the editor is:

```
duo temporal \  
  --workflow UnlockTerraformLockWorkflow \  
  --spacelift_stack_name "temporal/platform/stage" \  
  --lock_id "57144c0f-690e-5acf-725b-a6eb11e11866" \  
  --yes
```



Claude Code Skills

/deploy-stage

STEP 1: PRE-DEPLOYMENT CHECK

Run `git status`. If uncommitted changes exist, prompt the user to commit. Proceed based on user choice.

STEP 2: EXECUTE DEPLOYMENT

```
duo temporal deploy-stage [--branch <branch-name>] [--dry_run true]
```

- `--branch`: Omit to auto-detect and deploy the current git branch.
- `--dry_run`: Omit to use the default (false).



Claude Code Skills



Temporal Self-Service Notifier APP 1 minute ago

 TemporalWorkerStageDeployWorkflow running 

Workflow ID: `TemporalWorkerStageDeployWorkflow-branch-zhihao-delete-rds-associated-resources-20260409035335269362-0d9cde5d`

 Triggered by: 

 Deploy plan

`zhihao/delete-rds-associated-resources`

- `cloud-ops`: common-tasks, datastore, duolingo-dev, galaxy, router
- `portal`

11 file(s) changed:

[See more](#)



View Workflow Progress

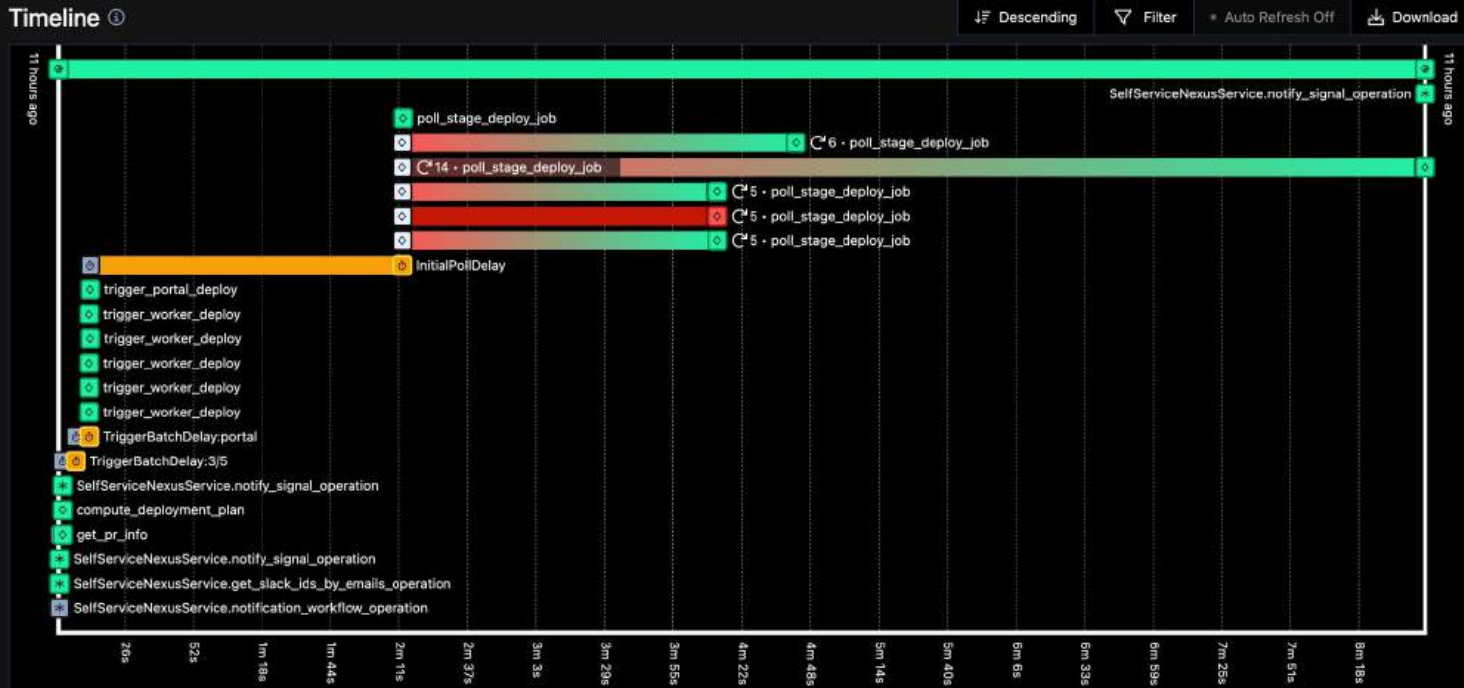


View Logs



Claude Code Skills

/deploy-stage



The Results We Planned For



- **Saved thousands of engineering hours on time-consuming operations like RDS resizing, DynamoDB deletion, and cache upgrades, while preserving human-in-the-loop approvals and significantly reducing the need for elevated access.**
 - Some workflows were instrumental during outages allowing engineers to execute critical operations safely without click-ops or dependency on Cloud Operations or PagerDuty rotation.
- **Expanded self-service beyond Cloud Operations team, with 45+ workflows now published and maintained by multiple teams across the organization.**


Month	Activities	Queries	Signals	Updates	Timers	Nexus	Workflows	Total Actions
January	16,700	0	3,006	644	1,906	602	1,820	24,678
December	11,565	24	2,240	562	1,450	775	2,109	18,725
November	7,396	0	1,384	372	1,122	430	1,252	11,956
October	15,363	16	2,590	820	2,093	810	2,388	24,080
September	6,048	0	1,132	440	1,086	566	1,584	10,856

Note: Company size: fewer than 1,000 employees globally, with <500 engineers.



The Wins We Didn't Expect



- **Adoption of Temporal within Duolingo** 

- Temporal has grown from a Cloud Ops tool into an organization-wide workflow platform, with teams increasingly aware of its capabilities.
- Much of the supporting infrastructure grew organically across teams rather than through a formal OKR, creating a virtuous cycle of demos, word-of-mouth, and shared tooling.



Zhihao Wang · Senior Software Engineer · Apr 4, 2025 (10:56)
In **All**, **Infrastructure**, **Tooling**

Generating Temporal Workflow w/ Temporal Workflow (Now You Have No Excuse Not to Give Temporal a Try)


Spinning up a new Temporal workflow used to take **1–2 hours** of boilerplate setup and Terraform configs. Thanks to our new [BootstrapTemporalWorkflow](#) (a Temporal workflow that generates workflows!), you can now run your first Temporal workflow locally and then deploy Temporal workers and workflows to AWS in **just minutes**. No more excuses, Duos!

<5min

One-time effort for namespace setup

<10min

Run your 1st Temporal workflow locally

 This workflow also lives in the self-service portal, powered by Nexus ✨

The Wins We Didn't Expect

- **Beyond the portal: an ecosystem of workflows-as-a-service and tools**
 - We started with a simple goal: build a self-service portal for Temporal workflows.
 - We ended up creating a broader ecosystem of workflows-as-a-service and tooling.
 - The portal is now just one entry point into the platform.



Demo



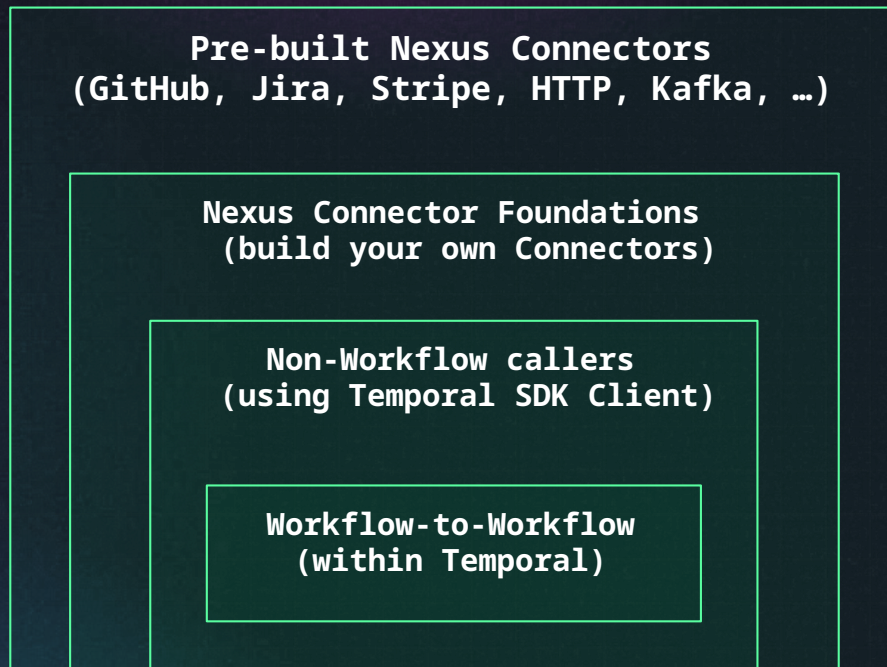
Nexus: What's next?

Today

- **Reliably connect workflows** across any namespace within Temporal

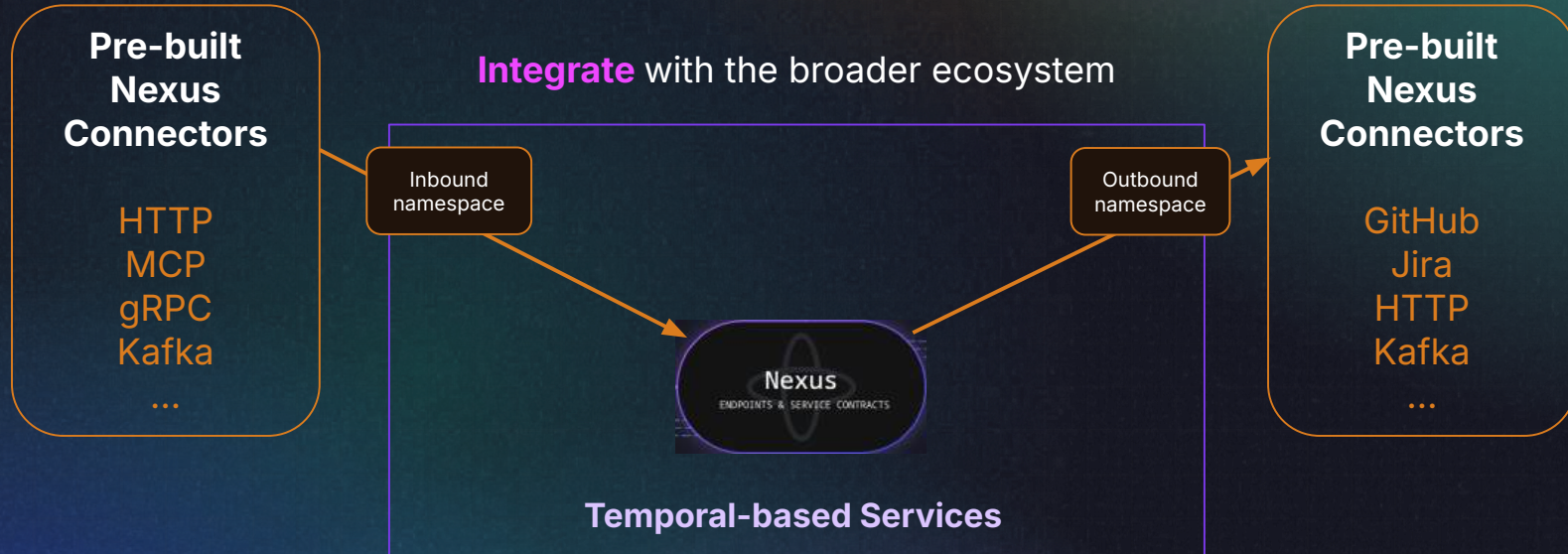
Next up

- **Make Nexus calls from anywhere**
 - other apps, services, bash scripts, ...
 - via Temporal SDK, HTTP, Kafka, ...
- **Integrate non-Temporal services**
 - Nexus facade for existing APIs
 - HTTP, Kafka, gRPC, ...
 - GitHub, Jira, etc
 - Streamlined Workflow dev experience



Nexus Connectors

Connect faster with Nexus Connectors for popular services and protocols



Q&A

Thank you for your time!

Find me on Temporal Community Slack
@zihao



LinkedIn