_/ TEMPORAL REPLAY 2026

From Bottlenecks to Self-Service

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

Zhihao Wang
Staff Software Engineer @ Duolingo

_/ DUOLINGO MISSION

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.

_/ ABOUT ME

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.

_/ AGENDA

What We'll Cover

01
What is Temporal Nexus?
Coss-namespace Communication

02
Duolingo + Temporal
How Cloud Operations team uses Temporal

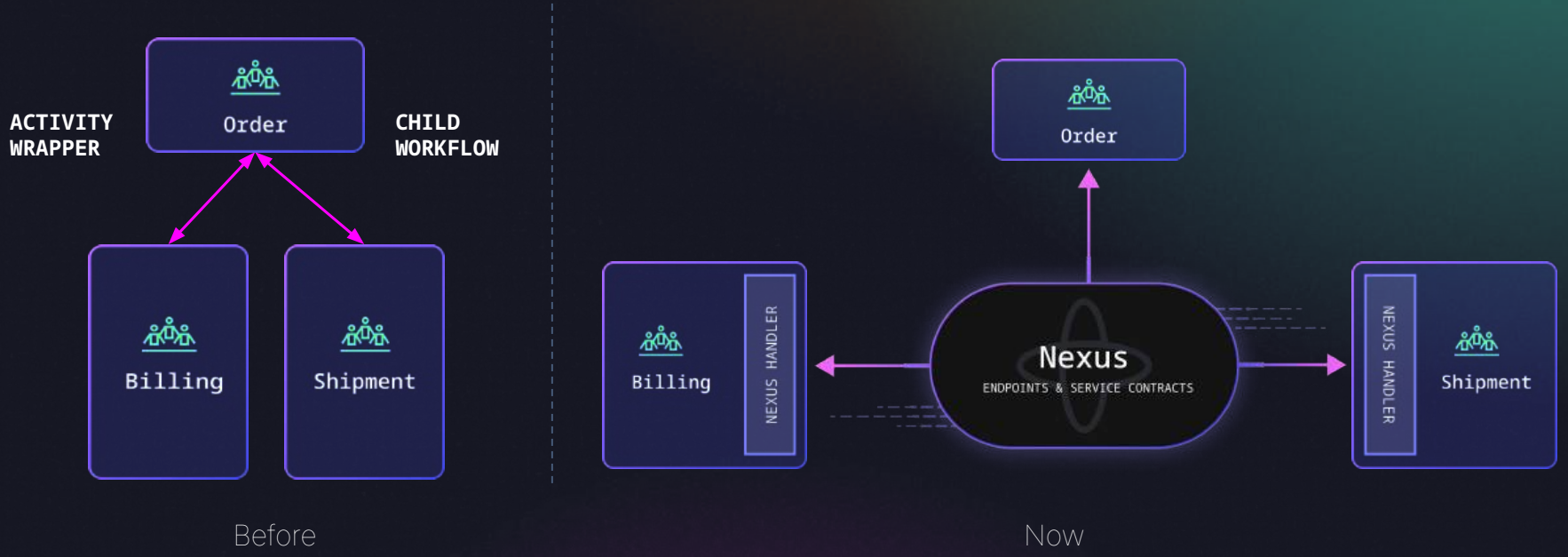
03
Building Self-Service
Architecture, access control, and the portal

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



_/ BACKGROUND

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

_/ BACKGROUND

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

_/ BACKGROUND

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~~

_/ BACKGROUND

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

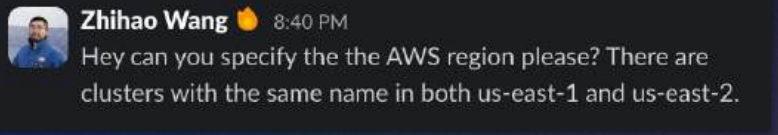
EARLY SOLUTION

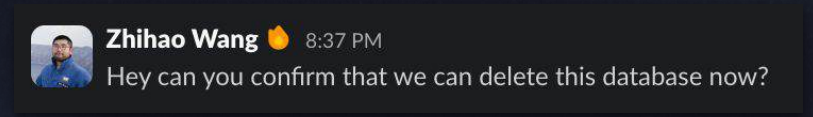
- Built Temporal workflows for repeatable infrastructure tasks
- Workflows executed in a Cloud Ops team-based Temporal namespace
- Strict access control on workflow execution
- Removed the need for manual execution using elevated AWS IAM roles

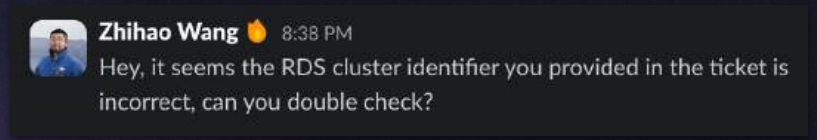
CHALLENGES

- 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.







_/ REQUIREMENTS

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**

_/ REQUIREMENTS

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!**

```python
@workflow.signal
async def confirm_deletion(self, params) -> None:
    self._confirm_deletion = True

@workflow.run
async def run(self, params) -> dict:
    ...
    await workflow.wait_condition(lambda: self._confirm_deletion or self._exit)
    ...

```

_/ REQUIREMENTS

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!**

```python
@workflow.signal
async def confirm_deletion(self, params) -> None:
    self._confirm_deletion = True

@workflow.run
async def run(self, params) -> dict:
    ...
    await workflow.wait_condition(lambda: self._confirm_deletion or self._exit)
    ...

```

_/ REQUIREMENTS

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**

```python
@workflow.signal
async def confirm_deletion(self, params) -> None:
    self._confirm_deletion = True

@workflow.run
async def run(self, params) -> dict:
    ...
    await workflow.wait_condition(lambda: self._confirm_deletion or self._exit)
    ...

```

_/ REQUIREMENTS

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**

```python
@workflow.signal
async def confirm_deletion(self, params) -> None:
    self._confirm_deletion = True

@workflow.run
async def run(self, params) -> dict:
    ...
    await workflow.wait_condition(lambda: self._confirm_deletion or self._exit)
    ...

```

_/ CHALLENGES

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**

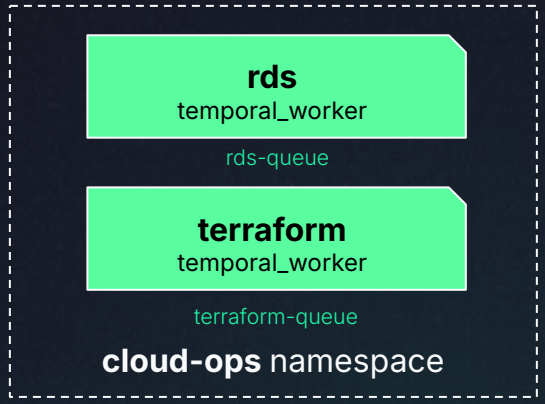
HARD CONSTRAINT

cloud-ops namespace access must remain restricted.

THE QUESTION NOW BECOMES

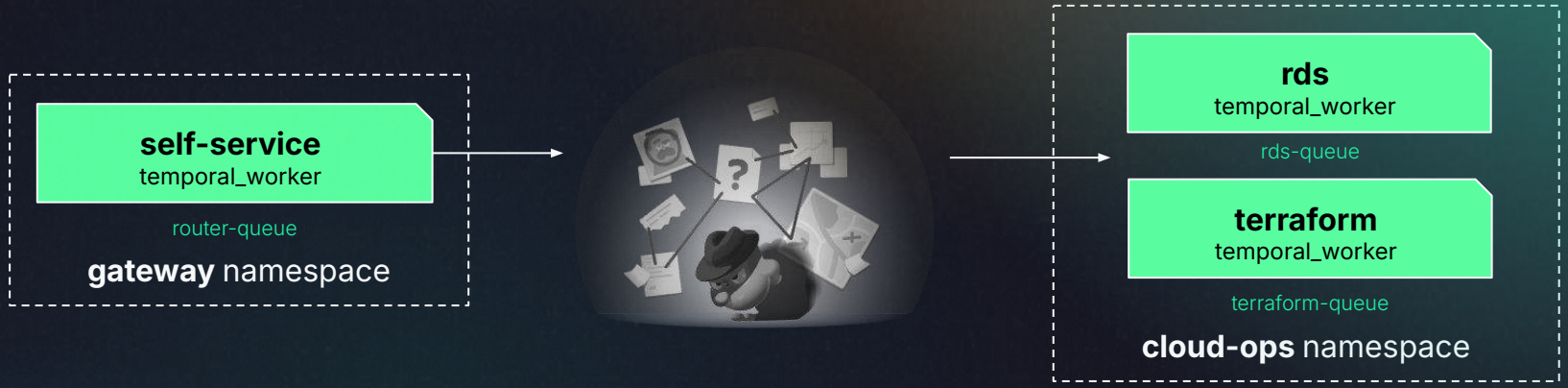
How do we expose these workflows safely?

The Solution: Cross-namespace Communication



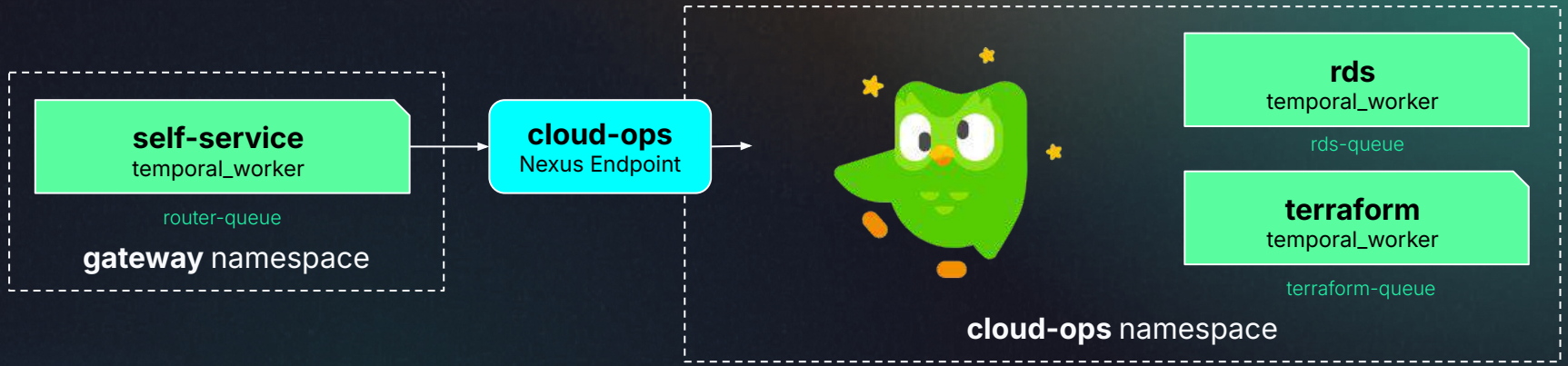
We do **not** want to change anything within existing workers within cloud-ops namespace.

The Solution: Cross-namespace Communication



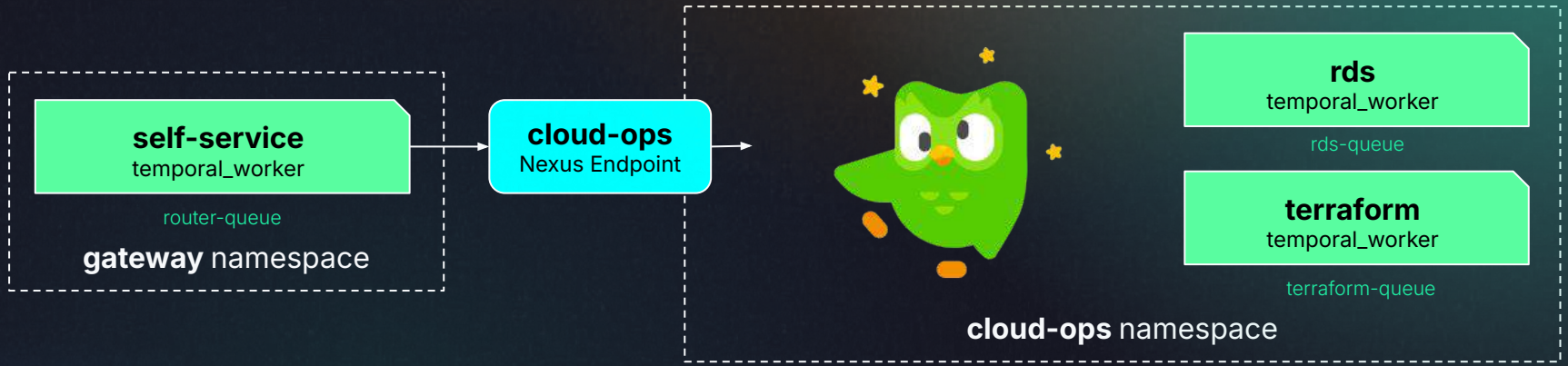
We need a central **gateway** namespace that everyone has write access to.

The Solution: Temporal Nexus

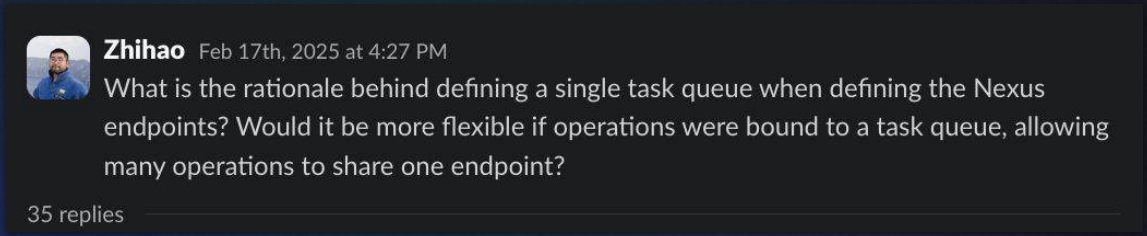


Wait... one Nexus endpoint has only **one** corresponding queue...

The Solution: Temporal Nexus



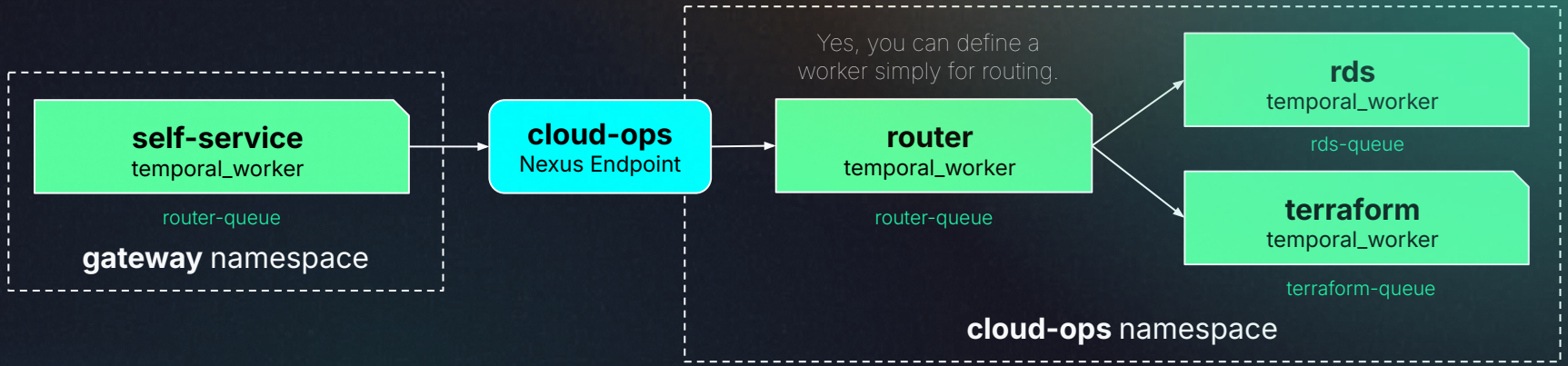
Wait... one Nexus endpoint has only **one** corresponding queue...



#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.*

_/ CHALLENGES

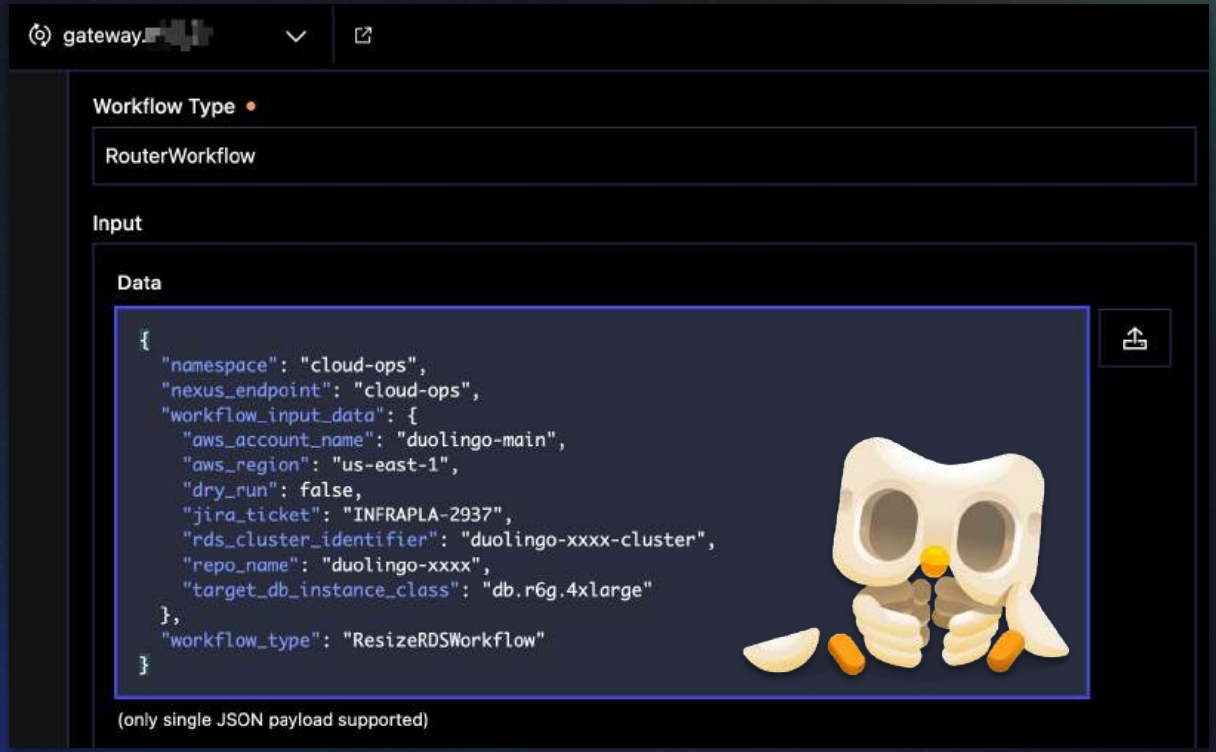
Have we solved all the challenges?

- 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~~

_/ CHALLENGES

We haven’t won yet...

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



_/ REQUIREMENTS

What We Actually Wanted

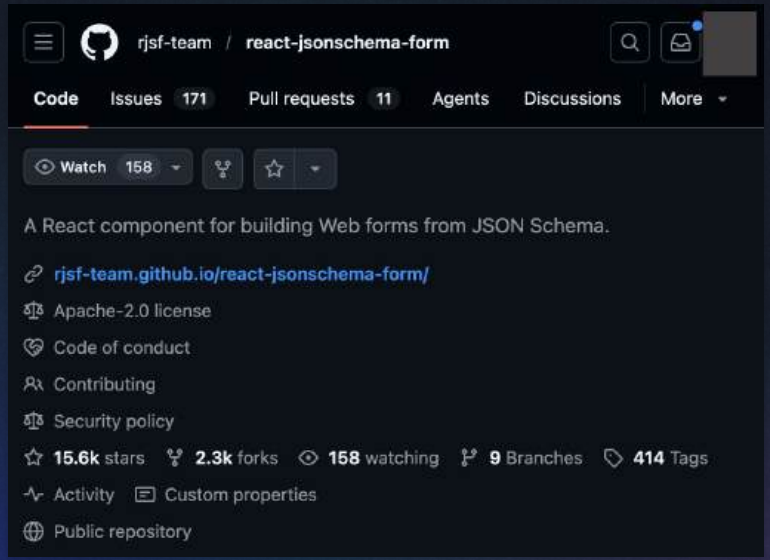
- Static, schema-driven website with minimal maintenance overhead
  - Declarative workflow definitions (JSON / YAML) that automatically render in the UI
- **Basic input validation to catch errors before execution**
- Ability to trigger workflows directly from the website

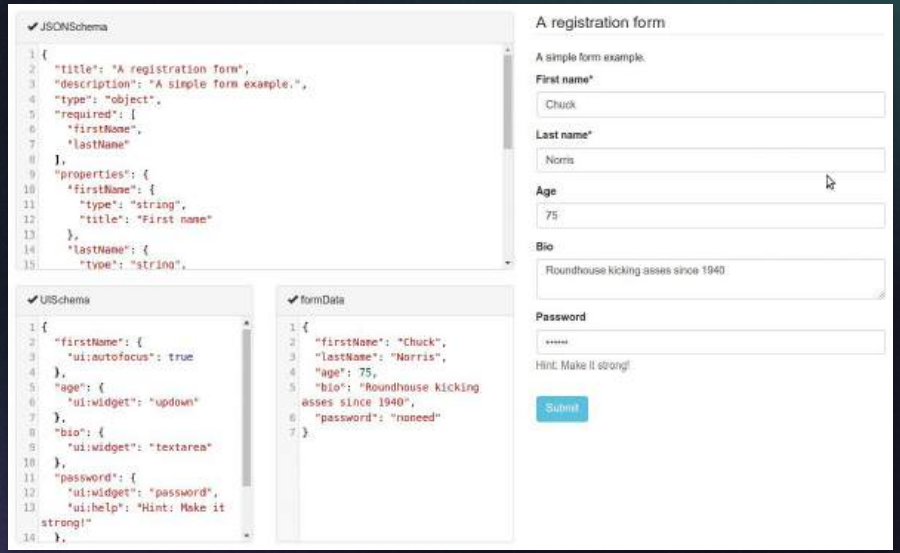
_/ REQUIREMENTS

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





_/ CHALLENGES

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.

_/ CHALLENGES

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!"

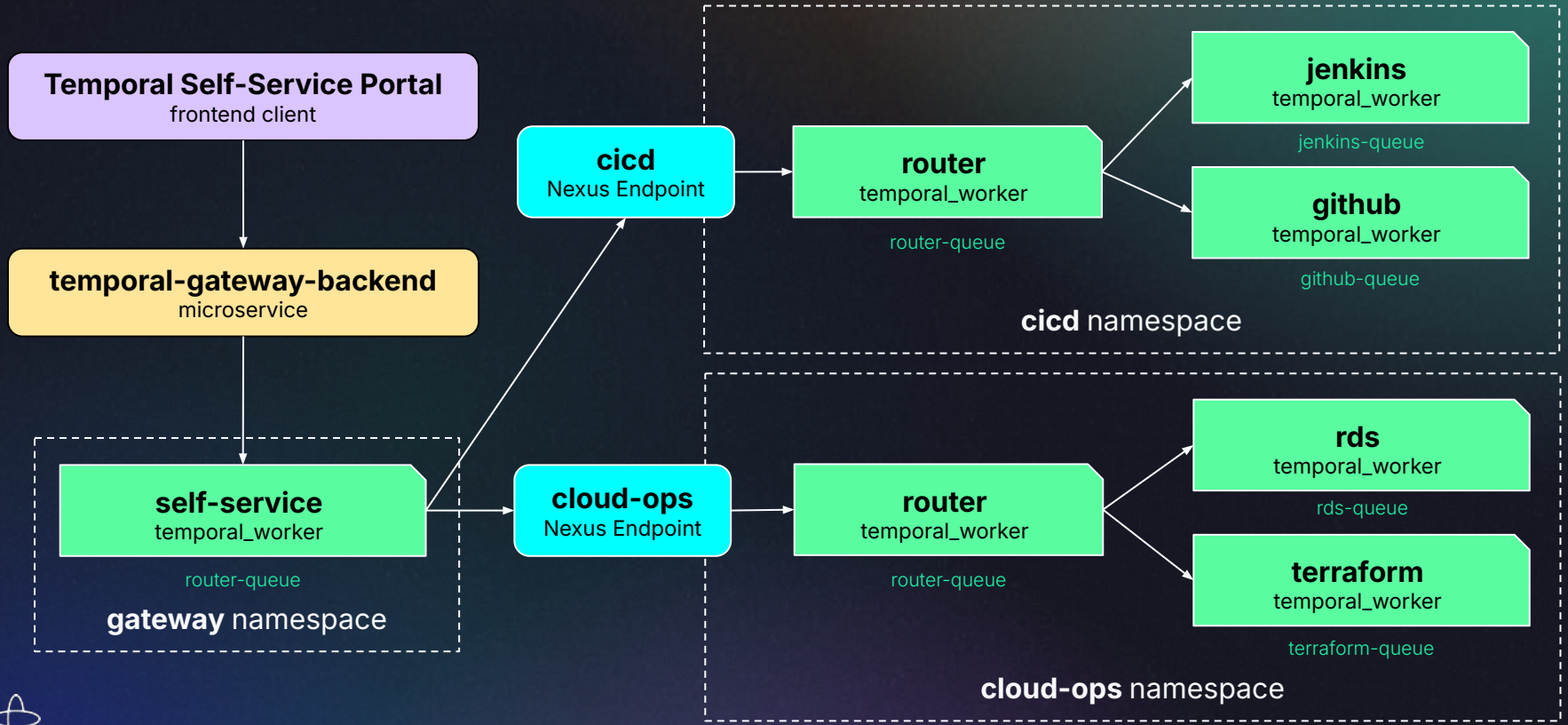
System Design Spec: Temporal Gateway

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

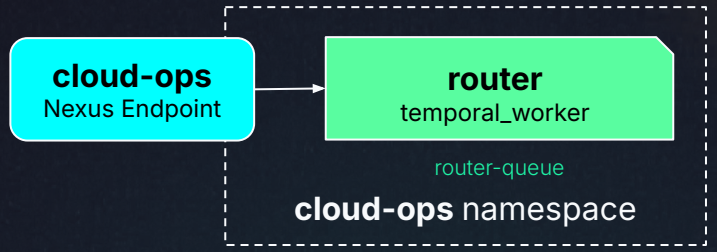
gateway namespace

```python
@dataclass
class RouterWorkflowParams:
    """
    Parameters required for running the RouterWorkflow.
    """

    workflow_type: str
    workflow_input_data: Dict[str, Any]
    namespace: str
    nexus_endpoint: str
    duo_email: str
```

```python
@workflow.defn
class RouterWorkflow:
    """
    RouterWorkflow is a Temporal workflow that triggers a workflow
    in other namespaces via Temporal Nexus.

    This workflow resides in the gateway namespace and can be triggered by all Duos,
    while the workflow in the target namespace typically has more restricted access.
    """

    @workflow.run
    async def run(self, params: RouterWorkflowParams) -> RouterWorkflowResponse:
        # Set DuoEmail search attribute
        workflow.upsert_search_attributes({"DuoEmail": [params.duo_email]})

        # Convention: Use endpoint name as the service name
        nexus_client = workflow.create_nexus_client(
            service=params.nexus_endpoint, endpoint=params.nexus_endpoint
        )

        # Convention: Use WorkflowName as the operation name
        operation_name = params.workflow_type

        # Execute the Nexus operation and wait for the result
        operation_handle = await nexus_client.start_operation(
            operation=operation_name,
            input=params.workflow_input_data,
            schedule_to_close_timeout=timedelta(days=30),
        )
        workflow_response: Dict[str, Any] = await operation_handle

        return RouterWorkflowResponse(
            workflow_type=params.workflow_type,
```

_/ SHOW, DON'T TELL

cloud-ops Namespace




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

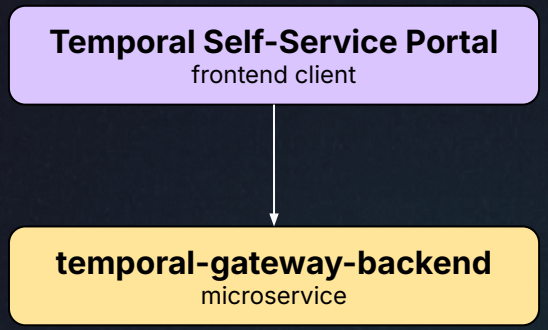
```go
var deleteRDSWorkflowOperation, _ = base.NewWorkflowNexusOperation[rds.DeleteRDSWorkflowParams, any](
  "DeleteRDSWorkflow",
  "datastore-queue",
  func(input rds.DeleteRDSWorkflowParams) string {
    return stringutils.AddDefaultTimestamp("DeleteRDSWorkflow-" + input.RDSIdentifier)
  },
)
```

invoke the workflow on a different queue in the same namespace

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

_/ SHOW, DON'T TELL

Temporal Self-Service Portal

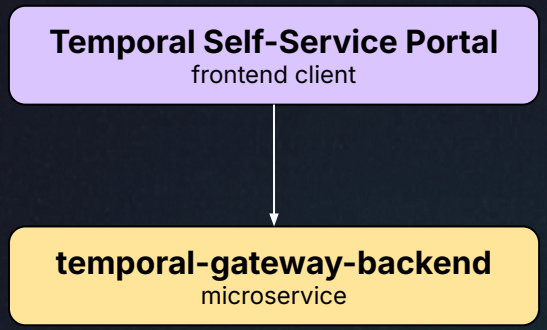



```typescript
export interface TemporalWorkflowInput {
  duoEmail: string;
  inputData?: Record<string, unknown>;
  namespace: string;
  region: string;
  taskQueue: string;
  workflowType: string;
}

export const triggerWorkflow = async (
  temporalWorkflowInput: TemporalWorkflowInput,
  env: Environment,
): Promise<WorkflowInfo> =>
  post(`${getGatewayUrl(env)}`, temporalWorkflowInput);
```

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

_/ SHOW, DON'T TELL

Temporal Self-Service Portal




```
const generateTemporalWorkflowInput = (...) => {
  const useStageBackend = env === "stage" || env === "local";

  return {
    duoEmail,
    inputData: {
      duo_email: duoEmail,
      namespace: useStageBackend ? stageNamespace : namespace,
      nexus_endpoint: useStageBackend ? stageNexusEndpoint : nexusEndpoint,
      workflow_input_data: {
        ...formData,
        duo_email: duoEmail,
      },
      workflow_type: workflowType,
    },
    namespace: useStageBackend ? "gateway-stage" : "gateway",
    region,
    taskQueue: "router-queue",
    // append uuid to workflowId to avoid collisions
    workflowId: `${workflowType}-${uuidv4()}`,
    workflowType: "RouterWorkflow",
  };
};
```

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

_/ SHOW, DON'T TELL

Temporal Self-Service Portal

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

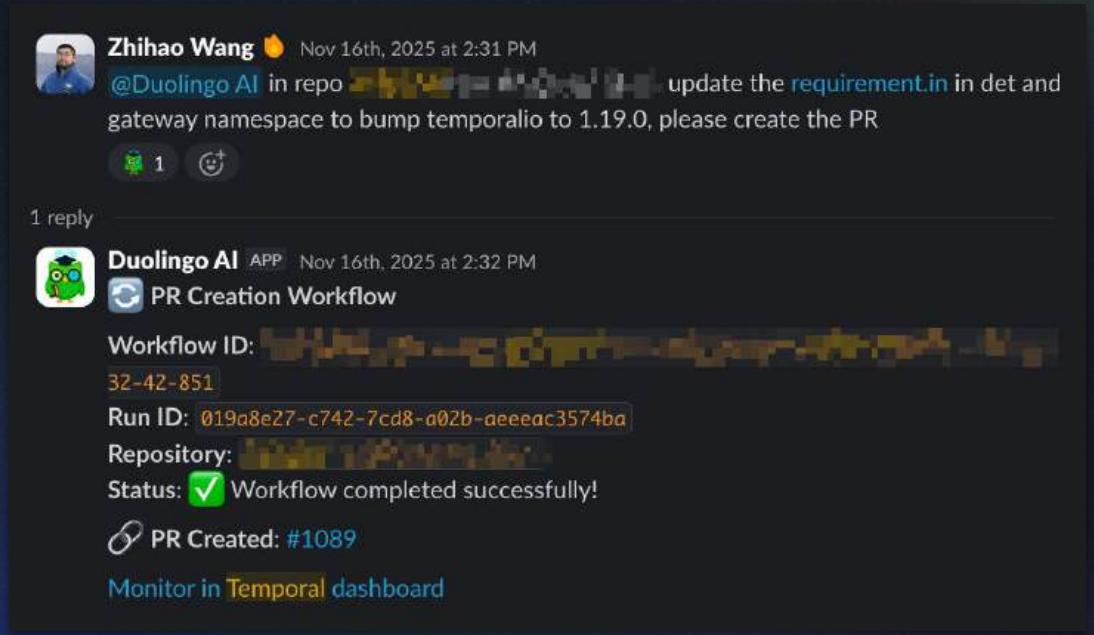
```
{
    "name": "Delete DynamoDB Table",
    "workflow_type": "DeleteDynamoWorkflow",
    "task_queue": "datastore-queue",
    "region": "us-east-1",
    "namespace": "cloud-ops",
    "stage_namespace": "cloud-ops-stage",
    "nexus_endpoint": "cloud-ops",
    "stage_nexus_endpoint": "cloud-ops-stage",
    "schema": {
        "title": "Delete DynamoDB Table",
        "description": "Delete a DynamoDB table",
        "type": "object",
        "required": [
            "aws_account_name",
            "aws_region",
            "table_name"
        ],
        "properties": {
            "aws_account_name": {
                "$ref": "#/definitions/aws_account_name"
            },
            "aws_region": {
                "$ref": "#/definitions/aws_region",
                "default": "us-east-1"
            },
            "table_name": {
                "description": "The DynamoDB table to delete",
                "title": "Table Name",
                "type": "string"
            },
            "repo_name": {
                "type": "string",
                "title": "GitHub Repository Name"
            },
            "dry_run": {
                "$ref": "#/definitions/dry_run"
            }
        }
    }
}
```

Many workflow configurations are now generated with AI, and the error rate is low thanks to the large number of existing examples.

_/ NEW PARADIGM

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.

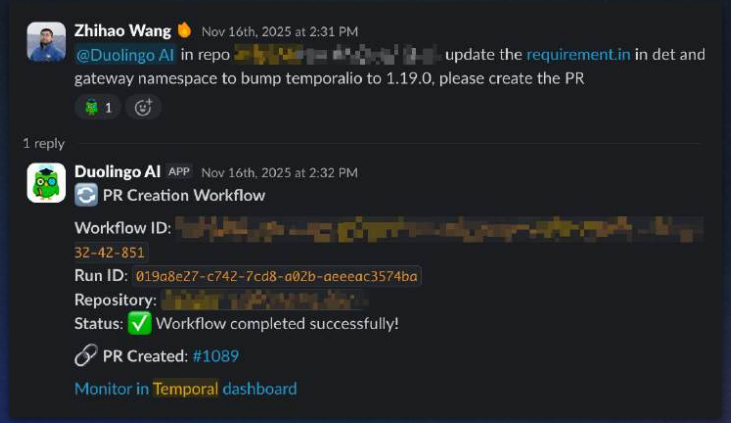


Duolingo AI bot triggers a Temporal workflow to create PR.

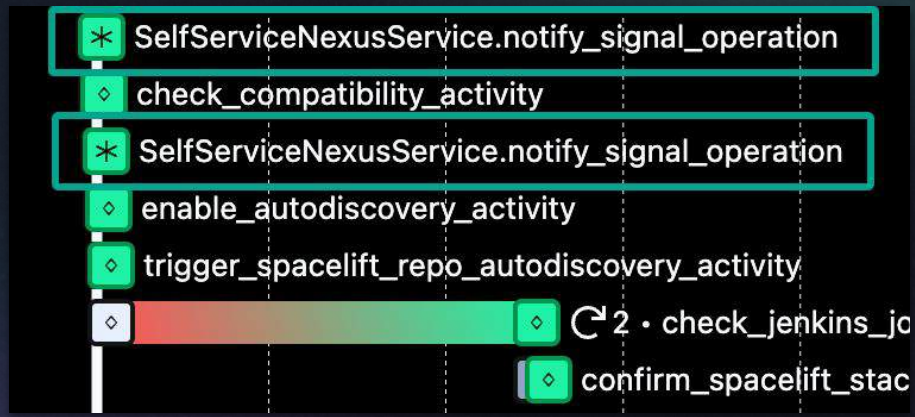
_/ NEW PARADIGM

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.



Duolingo AI bot triggers a Temporal workflow to create PR.

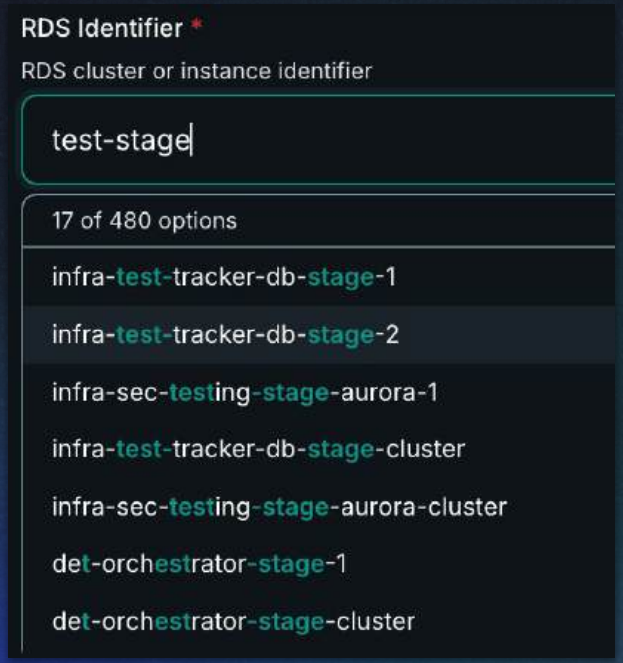


Self-service workflows send Slack notifications via Nexus operations.

_/ EVOLVING THE PLATFORM

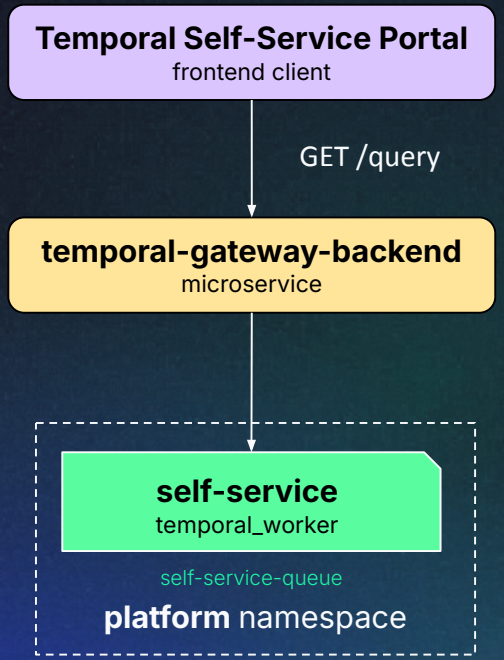
Dynamic Fields

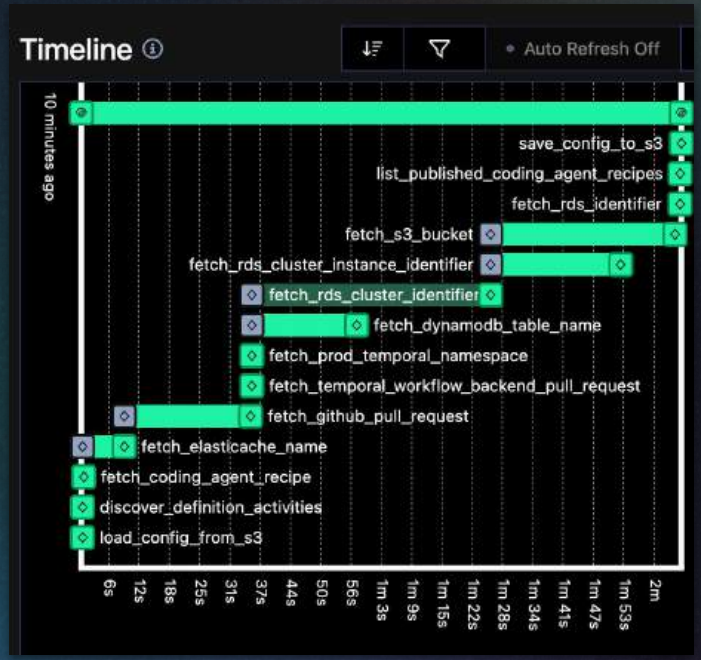
Engineers are lazy. Search beats copy and paste every time.

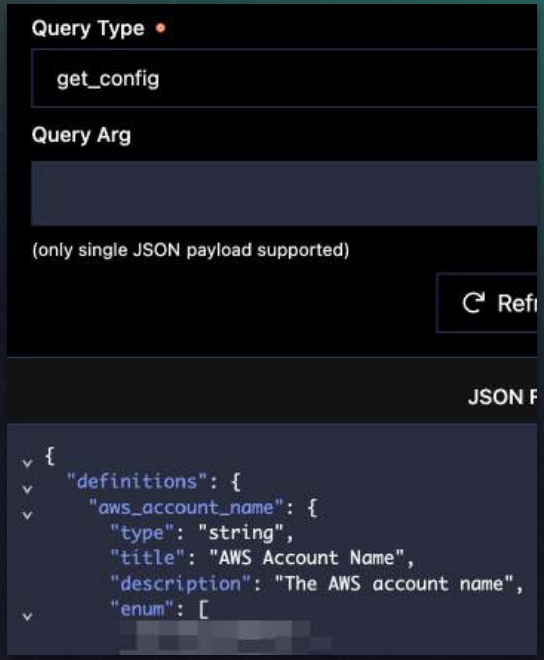


Dynamic Fields

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

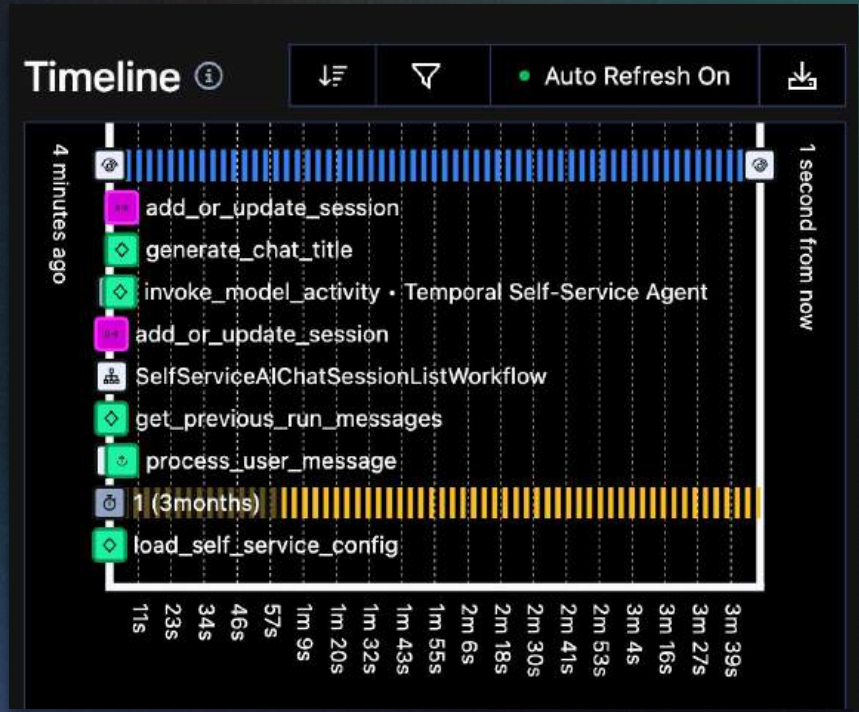
Upgrade Redis to Valkey

`RedisUpgradeValkeyWorkflow`

Pre-filled parameters:

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

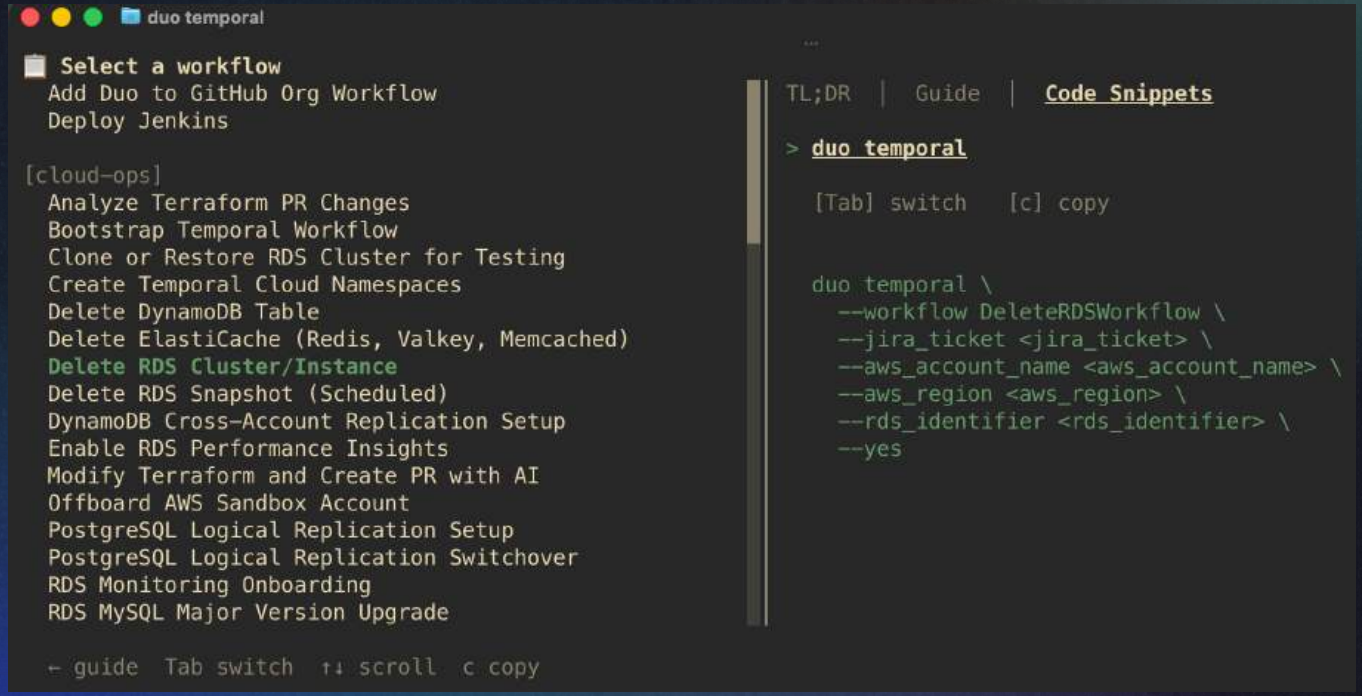
RedisUpgradeValkeyWorkflow →



_/ EVOLVING THE PLATFORM

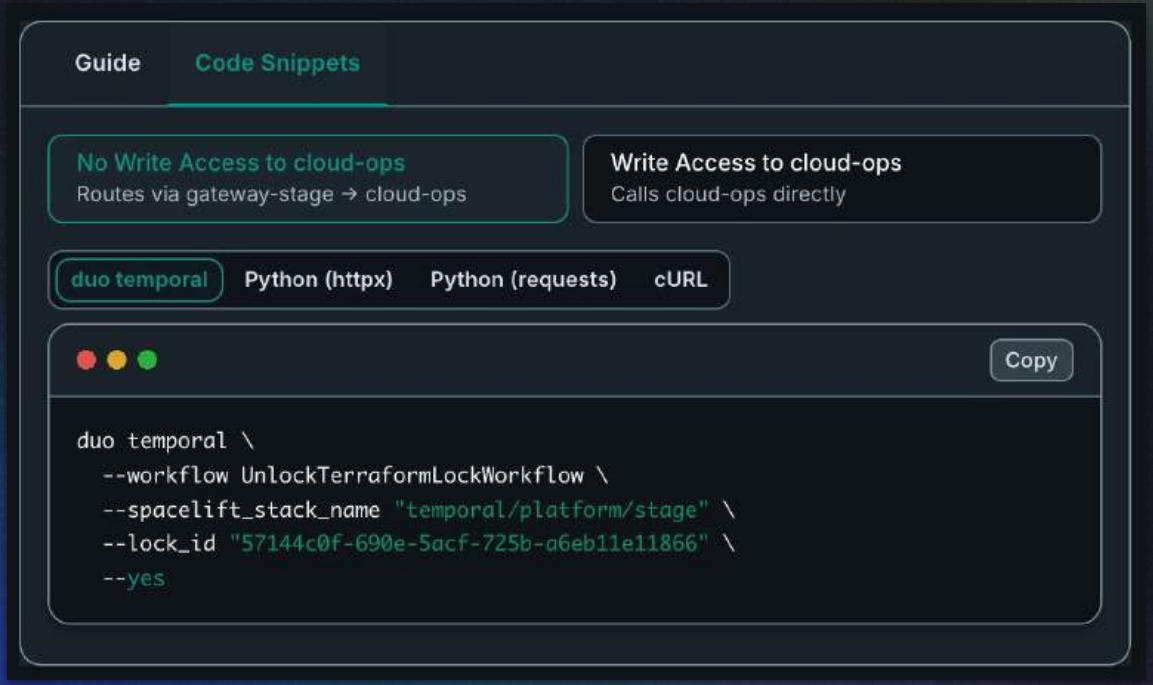
duo temporal cli

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



Code Snippets

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



Claude Code Skills

/deploy-stage

STEP 1: PRE-DEPLOYMENT CHECK

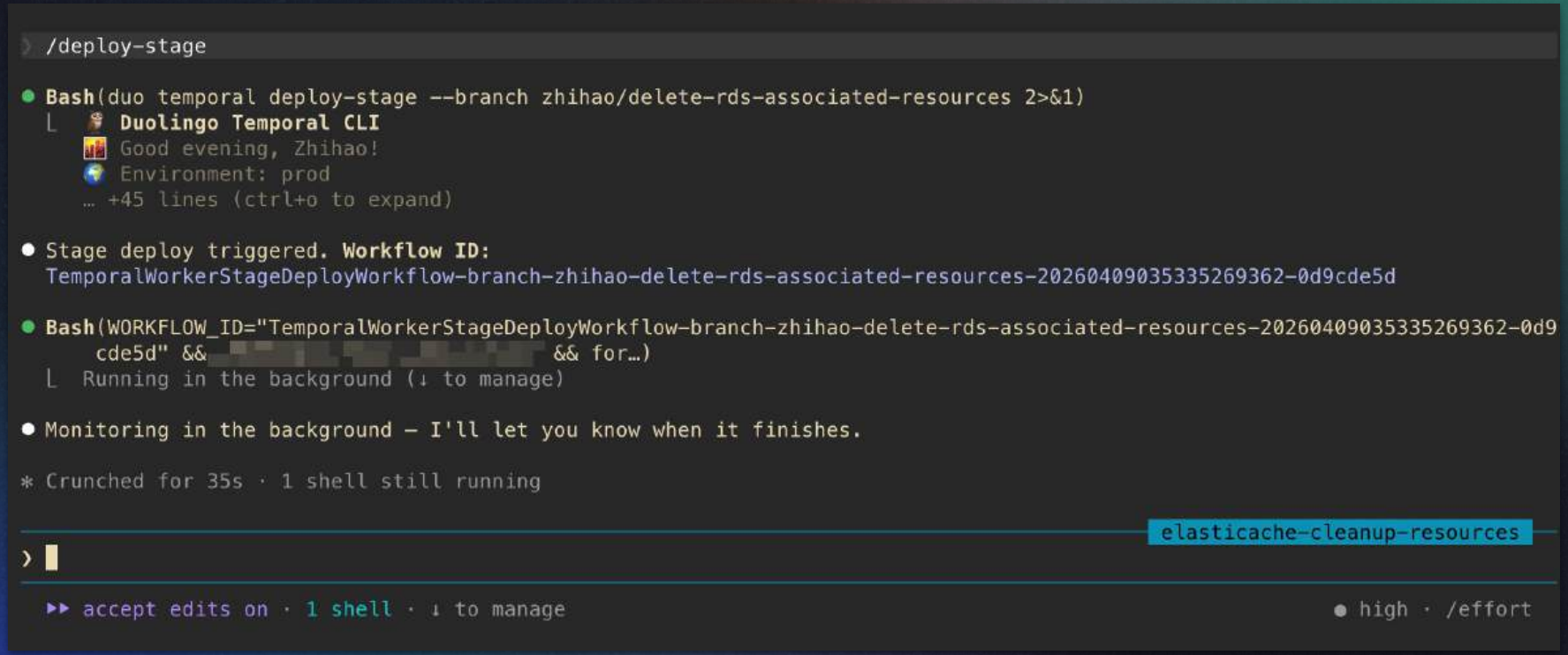
Run `git status`. If uncommitted changes exist, prompt the user to commit. Proceed based on user choice.

STEP 2: EXECUTE DEPLOYMENT

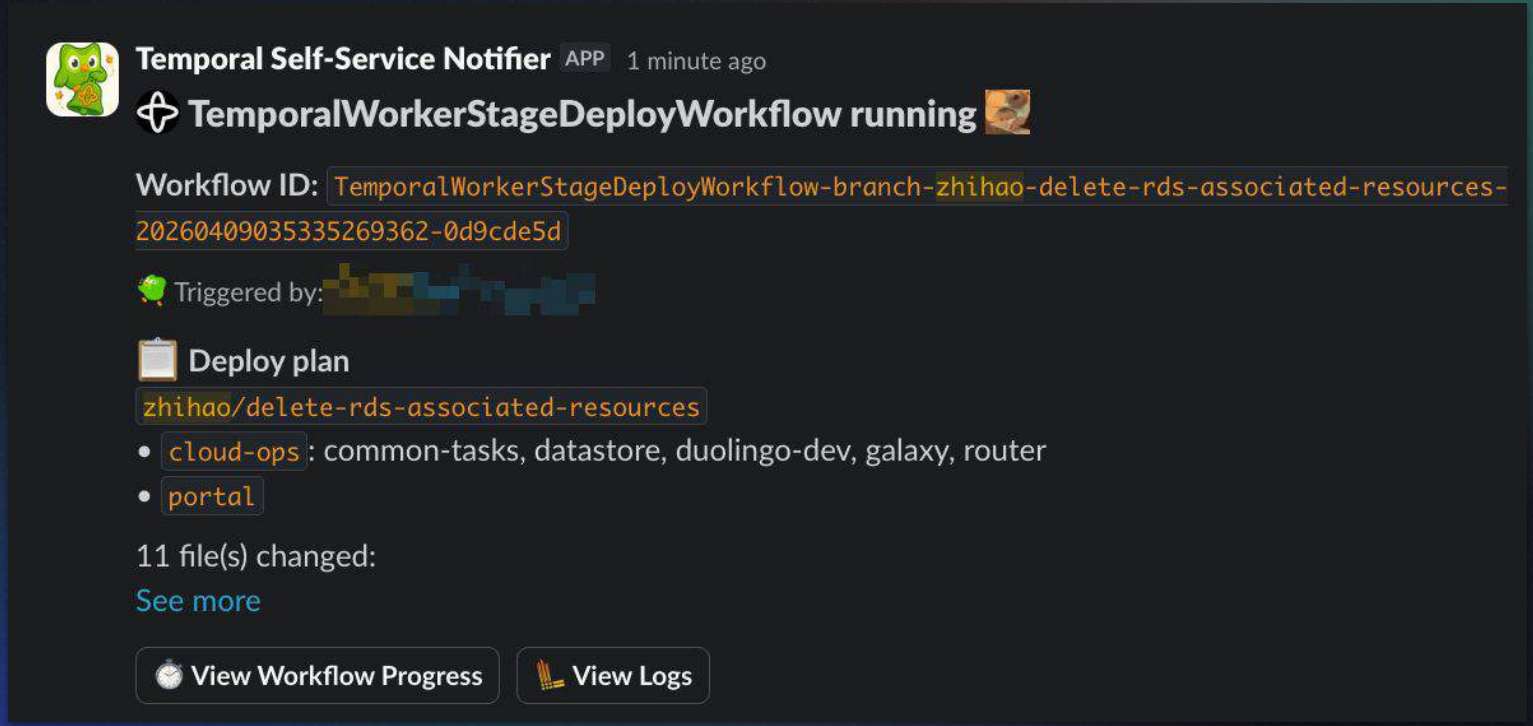
duo temporal deploy-stage [--branch <branch-name>] [--dry_run true]

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

Claude Code Skills

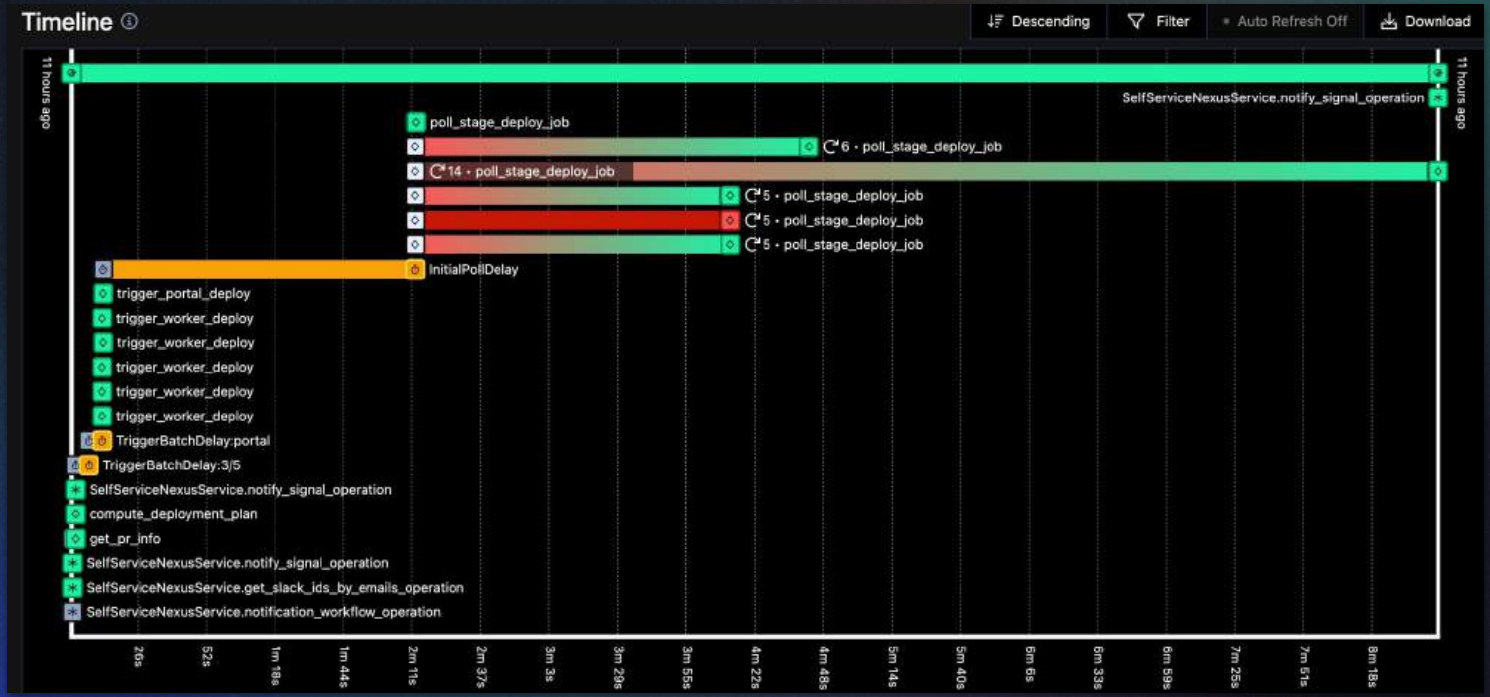


Claude Code Skills



Claude Code Skills

`/deploy-stage`



_/ WINS

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.

_/ BONUS

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 ✨

_/ BONUS

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

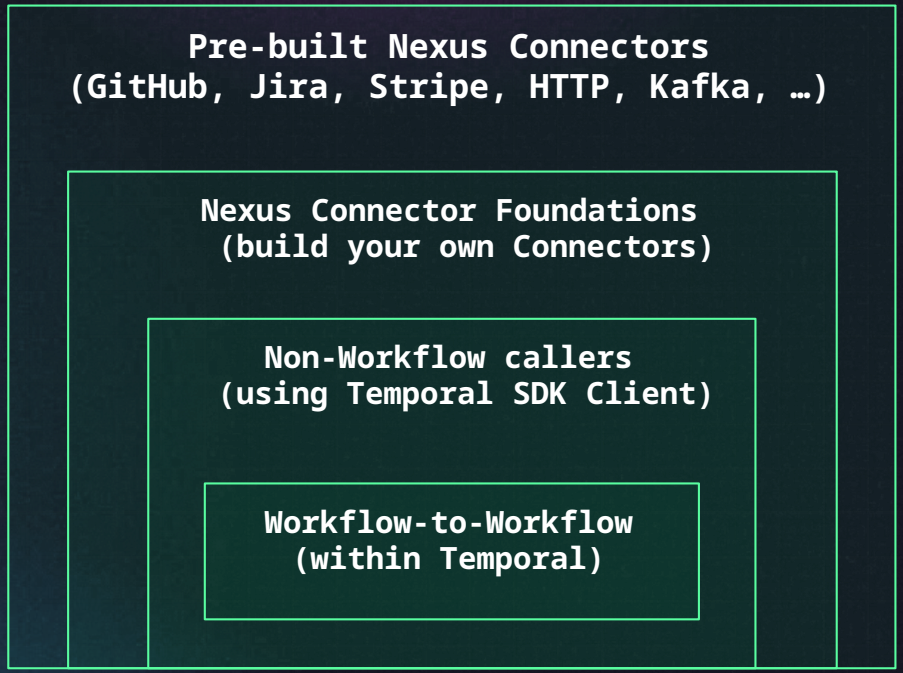
Nexus: What's next?

Today

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

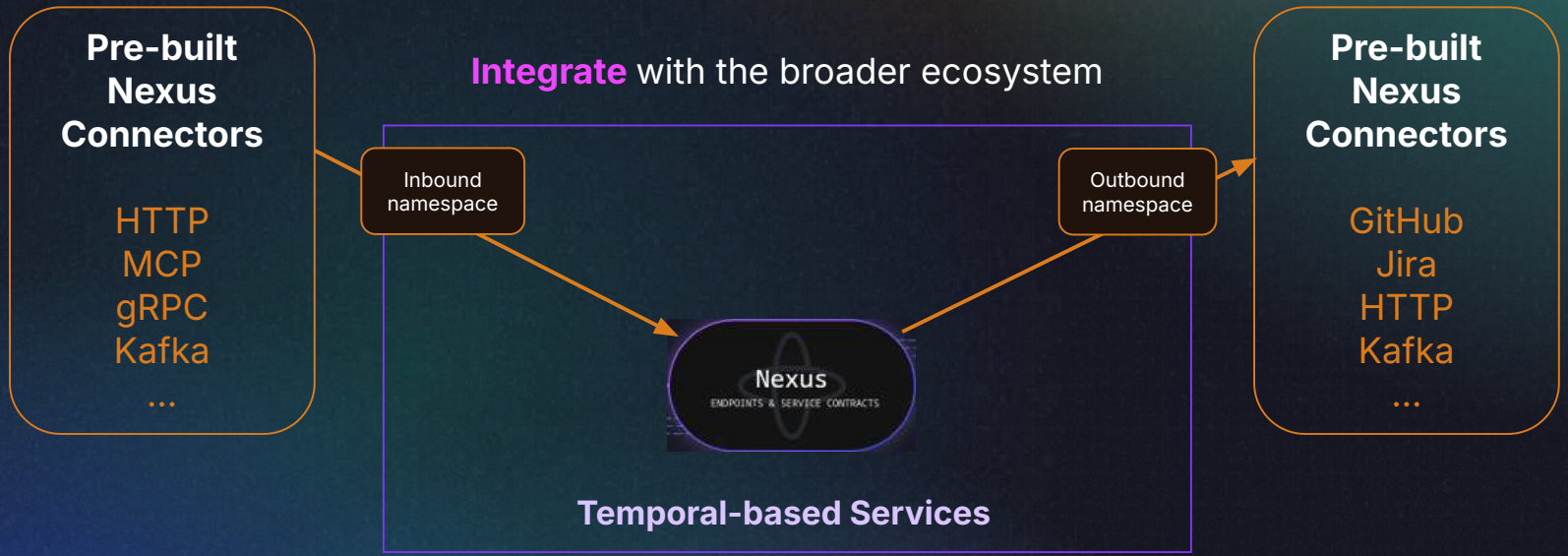
Next up

- **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
@zhihao

Linkedin