Temporal 101 - TypeScript

Temporal 101

00	**About this Workshop**
01	What is Temporal?
02	What is an Activity?
03	Parts of Temporal
04	Options for running a Temporal Cluster
05	Interacting with Temporal
06	Developing a Workflow, Activity, Worker
07	Executing a Workflow
08	Handling Failures
09	Putting it Together: Visualizing a Workflow Execution
10	Conclusion

Logistics

- Introductions
- Schedule
- Facilities
- WiFi
- Asking questions
- Getting help with exercises

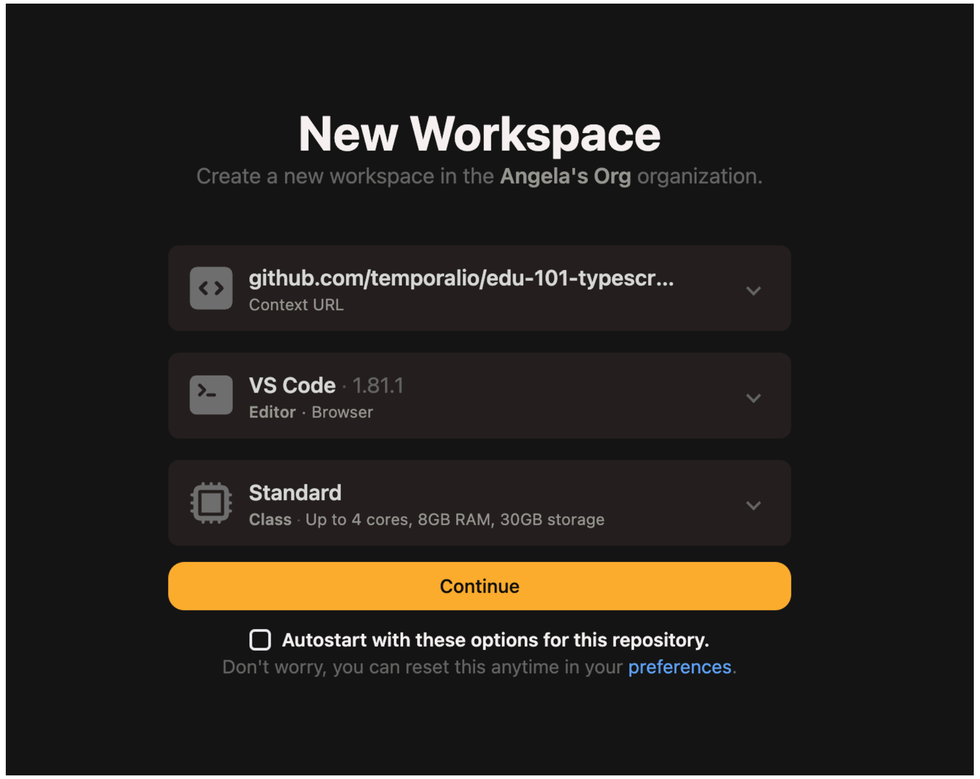
During this course, you will:

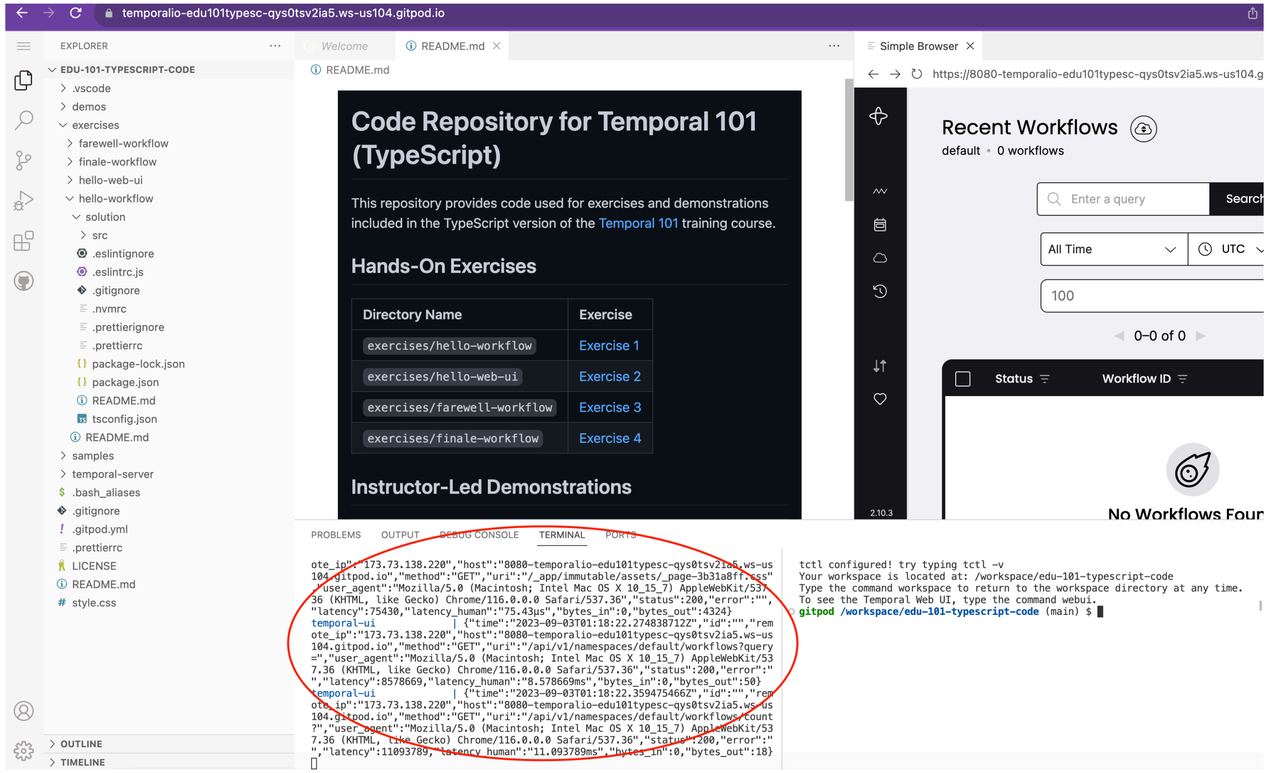
- Learn the basic architecture of the Temporal platform
- Develop and execute Workflows and Activities using the TypeScript SDK
- Use the Web UI to gain insight into current and previous executions
- Experiment with failures and retries
- Understand how a Temporal Cluster orchestrates execution

Exercise Environment

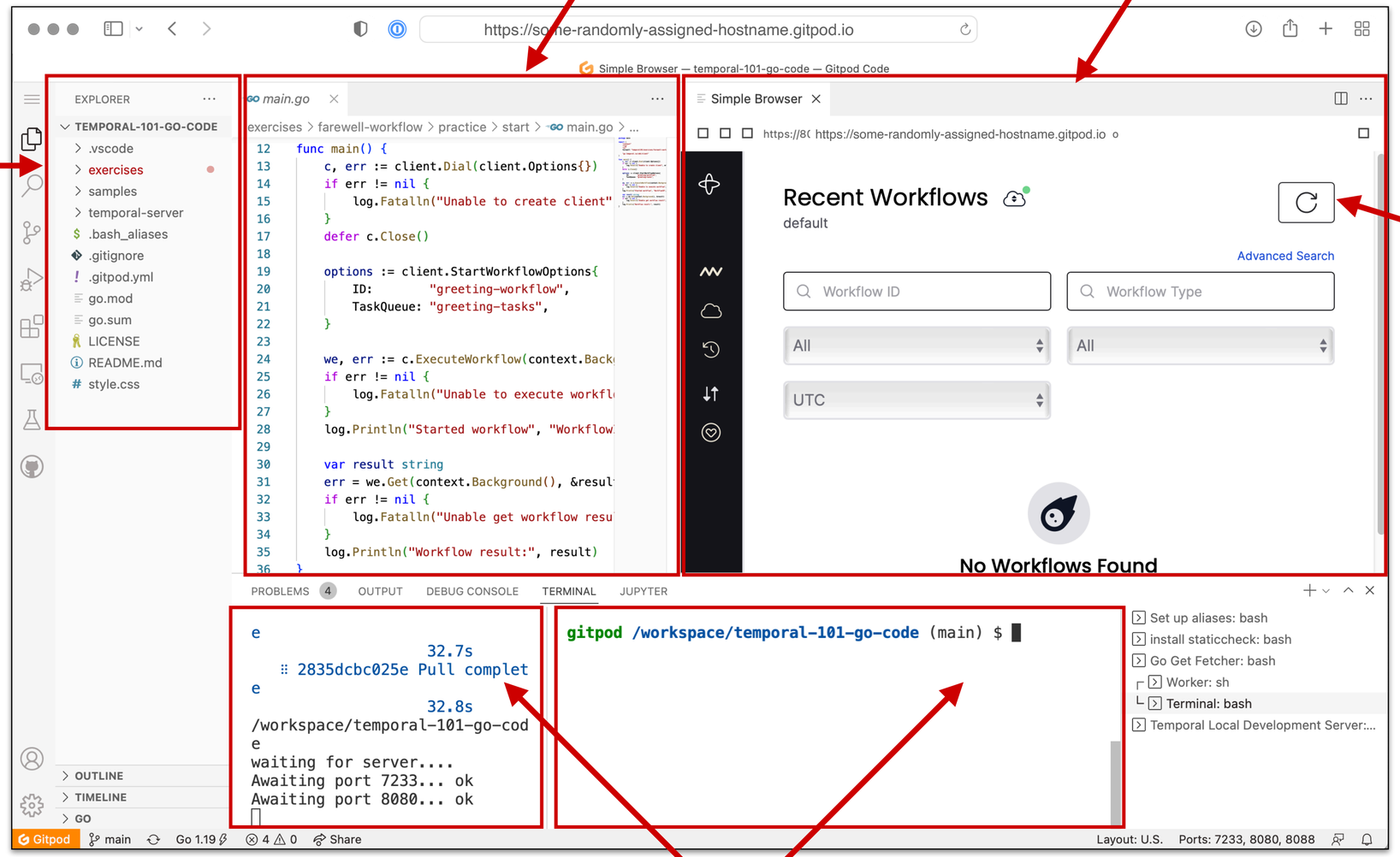
- **We provide a development environment for you in this course**
  - It uses the GitPod service to deploy a private cluster, plus a code editor and terminal
  - You access it through your browser (may require you to log in to GitHub)

GitPod link: t.mp/replay-101-typescript.


New Workspace
Create a new workspace in the Angela's Org organization.
github.com/temporalio/edu-101-typescr...
Context URL
VS Code · 1.81.1
Editor · Browser
Standard
Class · Up to 4 cores, 8GB RAM, 30GB storage
Continue
Autostart with these options for this repository.
Don't worry, you can reset this anytime in your preferences.


temporalio-edu101typesc-qys0tsv2ia5.ws-us104.gitpod.io
EXPLORER
EDU-101-TYPESCRIPT-CODE
README.md
Code Repository for Temporal 101 (TypeScript)
This repository provides code used for exercises and demonstrations included in the TypeScript version of the Temporal 101 training course.
Hands-On Exercises
Directory Name
Exercise
exercises/hello-workflow
Exercise 1
exercises/hello-web-ui
Exercise 2
exercises/farewell-workflow
Exercise 3
exercises/finale-workflow
Exercise 4
Instructor-Led Demonstrations
Simple Browser
Recent Workflows
default • 0 workflows
Enter a query
All Time
UTC
100
0–0 of 0
Status
Workflow ID
No Workflows Four
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
tctl configured! try typing tctl -v
Your workspace is located at: /workspace/edu-101-typescript-code
Type the command workspace to return to the workspace directory at any time.
To see the Temporal Web UI, type the command webui.
gitpod /workspace/edu-101-typescript-code (main) $

GitPod Overview



Temporal 101

Introducing Temporal

- Temporal is the open source runtime for managing a distributed application state at scale

- The Temporal Platform is a durable execution system for your code

What is a Workflow?

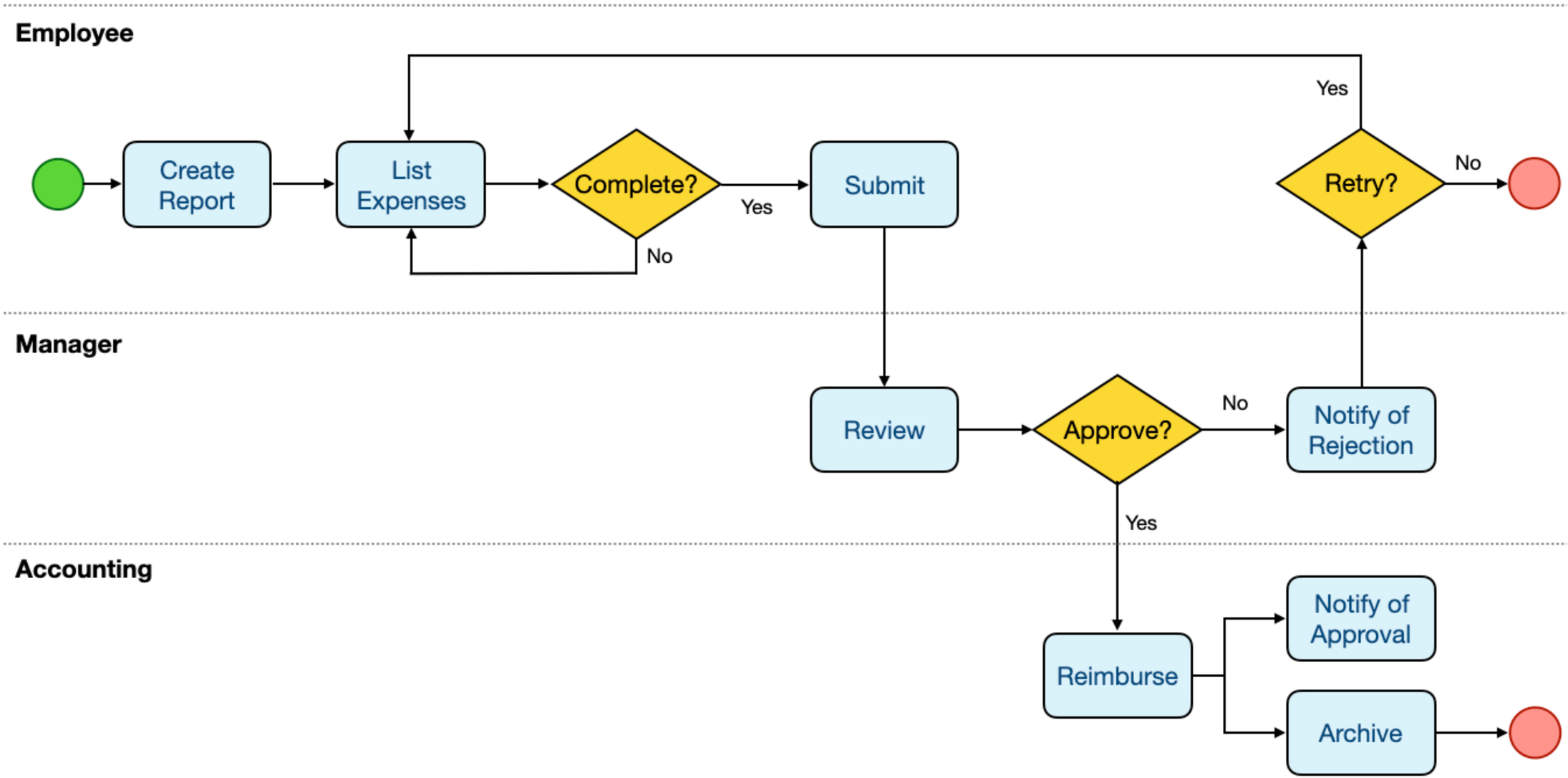
- **Conceptually, a workflow is a sequence of steps**

- **You probably have experience with workflows from everyday life**
  - Using a mobile app to transfer money
  - Buying concert tickets
  - Booking a vacation
  - Ordering a pizza
  - Filing an expense report

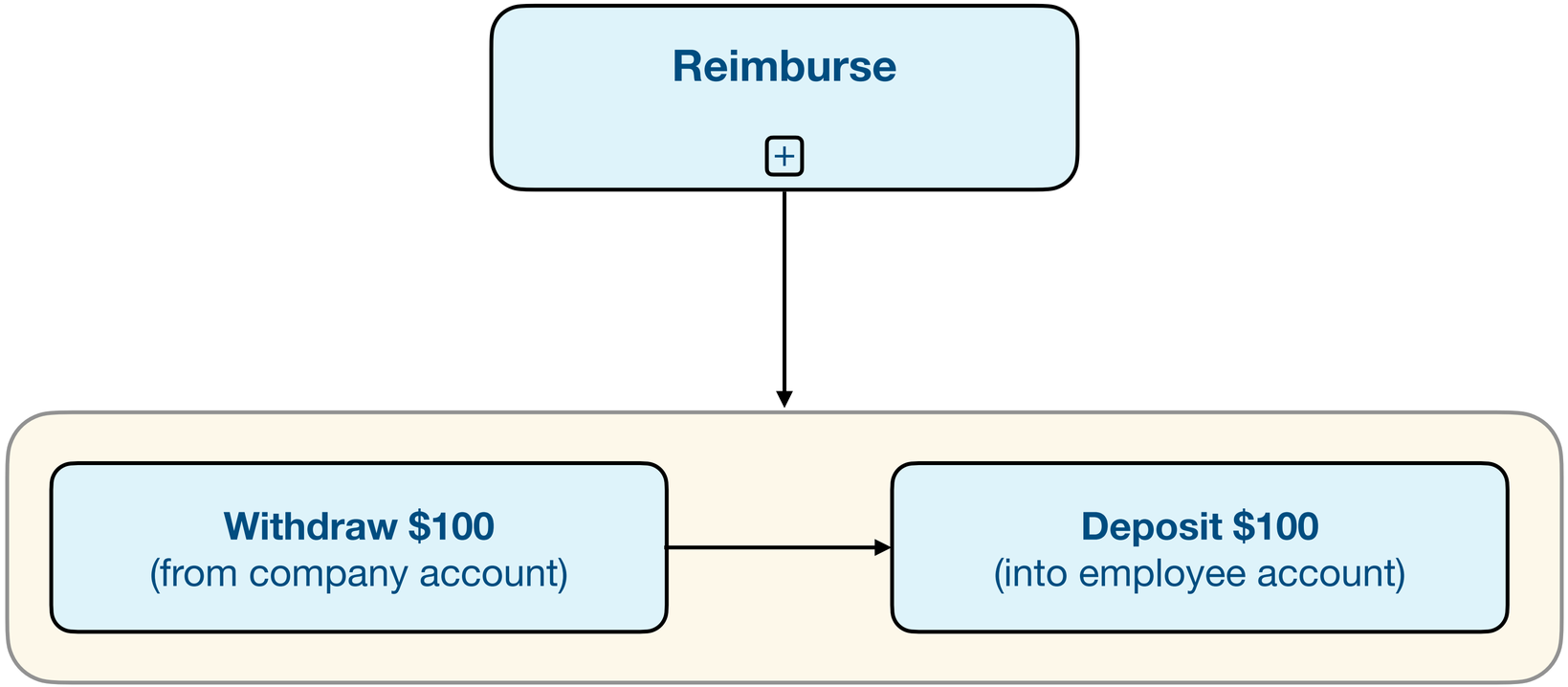
Long-running Possibilities:

- Implement a subscription like charging a card every 30 days
- Manages patient’s chronic conditions
- Track progress of participants over years for clinical trials
- Track retirement funds

Workflow Example: Expense Report



Workflow Example: Reimbursement



Correctness requires exactly-once execution

This Workflow is a Distributed System

```
async function withdrawMoney(details: PaymentDetails):
Promise<string> {
  const sourceBank = new BankingClient('bank1.example.com');
  const targetBank = new BankingClient('bank2.example.com');

  const withdrawConfirmation = await
sourceBank.withdraw(details.sourceAccount, details.amount);
  const depositConfirmation = await
targetBank.deposit(details.targetAccount, details.amount);

  return generateSuccessMessage(confirmation1,
confirmation2);
}
```

1

2

Correctness requires exactly-once execution

Introducing Temporal

- **The Temporal Platform guarantees durable execution of your code allowing you to just focus on business logic**

Why Workflows?

- Unlike other functions, Temporal Workflows are resilient
- They can run for years, surviving both server and application crashes

Why Workflows?

- Unlike other functions, Temporal Workflows are resilient
- They can run for years, surviving both server and application crashes
- Temporal will automatically recreate its pre-failure state

Writing Workflows

- Like other applications, you develop Workflows by writing code
- The code you write is executed at runtime

Determinism

- Workflows need to be deterministic
- Determinism: Each execution of a function will produce the same output given the same input

Determinism

- Workflows need to be deterministic
- Determinism: Each execution of a function will produce the same output given the same input
- Determinism is crucial for the consistency and reliability of Workflow executions

Determinism

- Workflows need to be deterministic
- Determinism: Each execution of a function will produce the same output given the same input
- Determinism is crucial for the consistency and reliability of Workflow executions
- Workflows -> Deterministic
  Activities -> Non-deterministic

Temporal 101

What Are Activities?

- They are executed during Workflow Execution

What Are Activities?

- They are executed during Workflow Execution
- Activities encapsulate business logic that is prone to failure or change

What Are Activities?

- They are executed during Workflow Execution
- Activities encapsulate business logic that is prone to failure or change
- Typically interact with external systems

What Are Activities?

- They are executed during Workflow Execution
- Activities encapsulate business logic that is prone to failure or change
- Typically interact with external systems
- They are retried upon failure

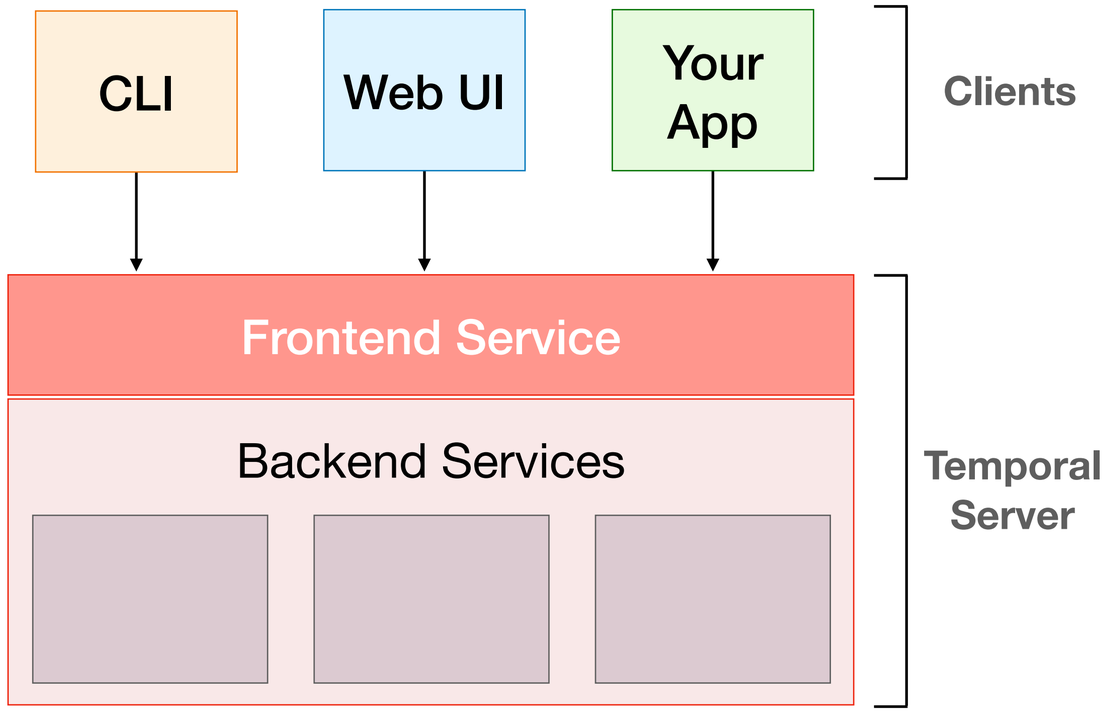
What Are Activities?

- They are executed during Workflow Execution
- Activities encapsulate business logic that is prone to failure or change
- If the Workflow fails, a previously stored value is supplied to the Workflow that encapsulates it

Temporal 101

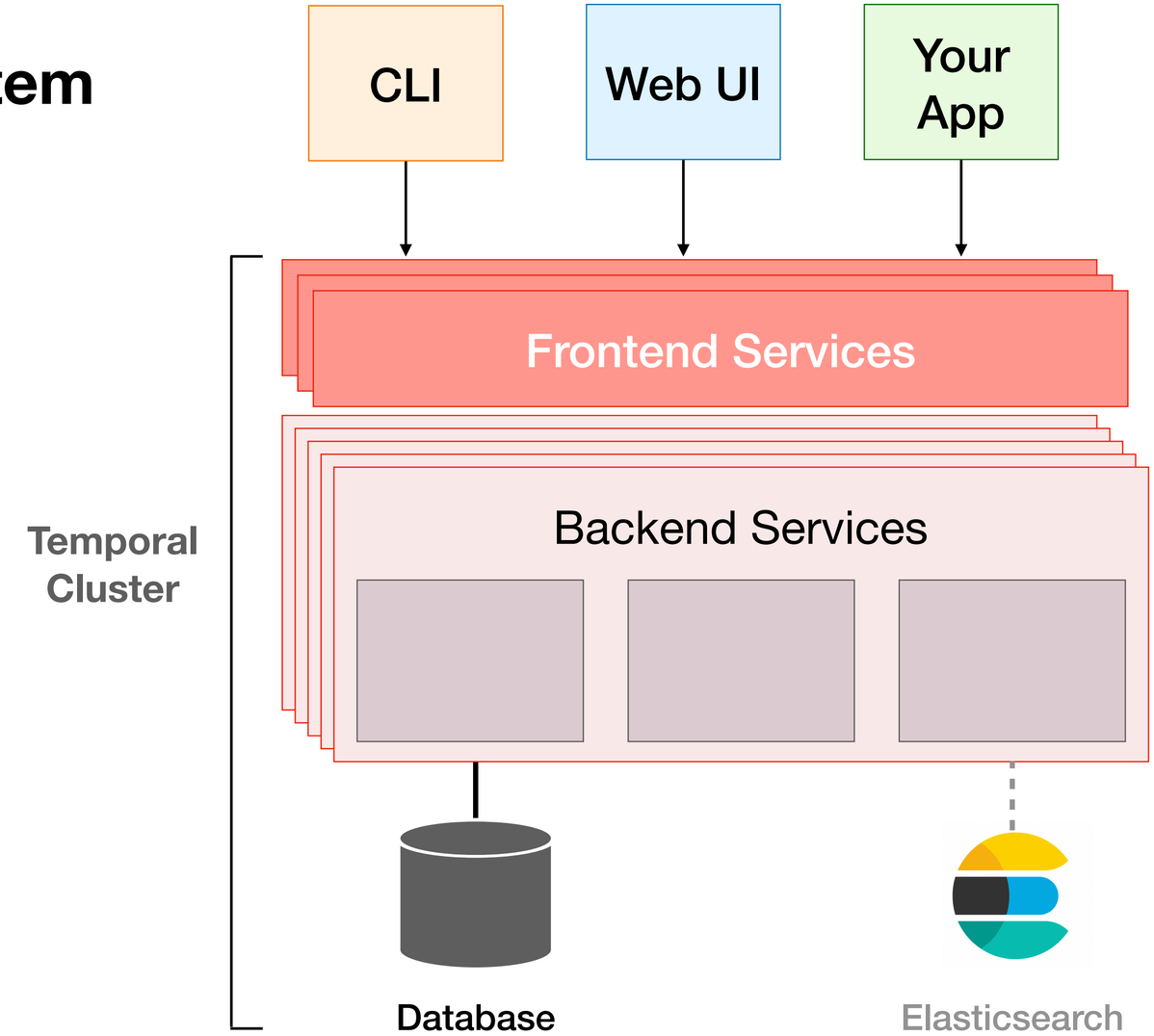
Architectural Overview: Temporal Server

- **Consists of multiple services**
  - Each service is horizontally scalable
  - The frontend service is an API gateway
  - Clients are external to the server and communicate with the server



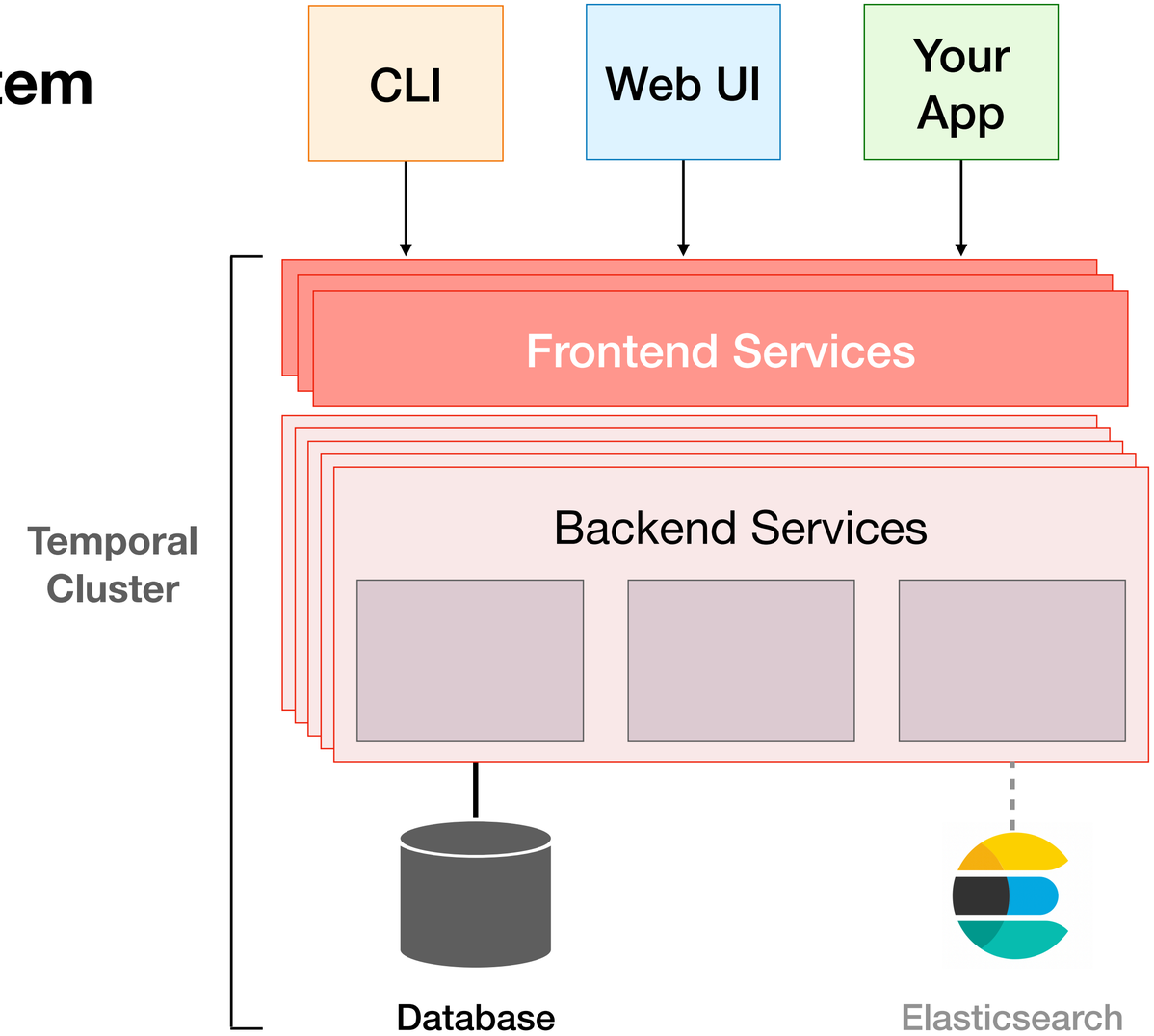
Architectural Overview: Temporal Cluster

- **Temporal Cluster is a complete system**
  - It is a deployment of the Temporal Server software and the components used with it
  - A database is a required component
    - Persists Workflow state and Event History
    - Also stores data for durable timers and queues



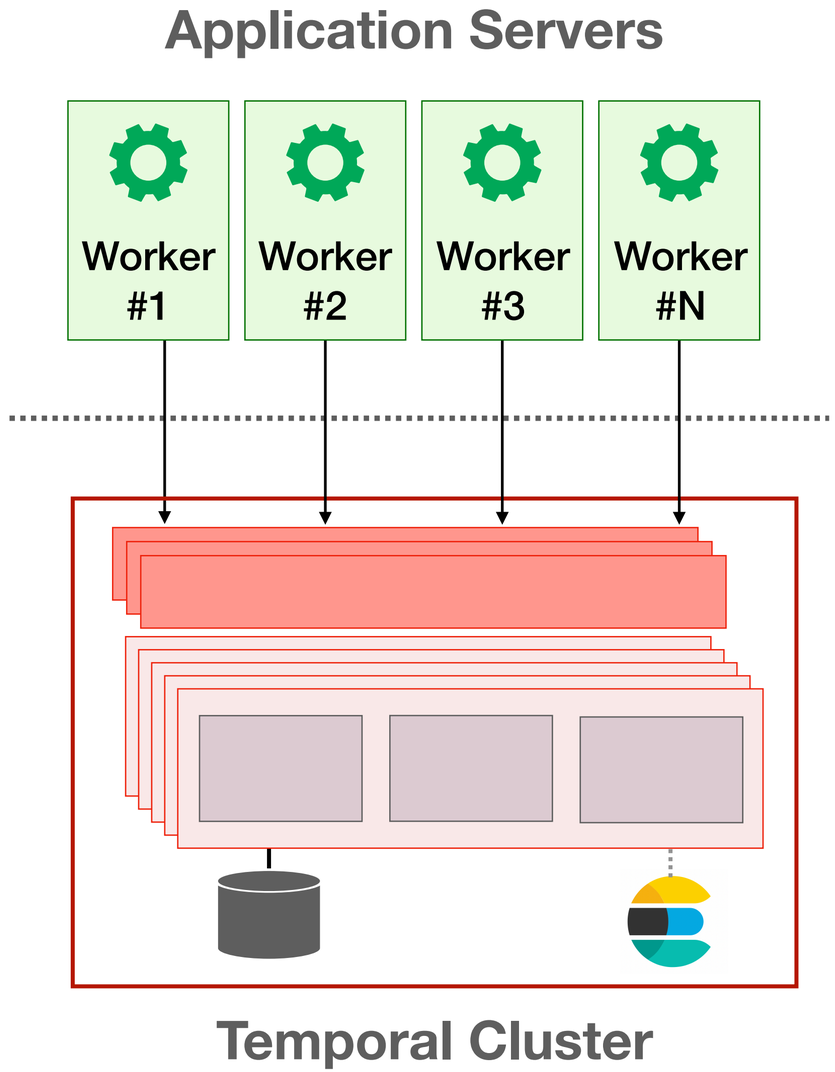
Architectural Overview: Temporal Cluster

- **Temporal Cluster is a complete system**
  - It is a deployment of the Temporal Server software and the components used with it
  - A database is a required component
    - Persists Workflow state and Event History
    - Also stores data for durable timers and queues
  - Elasticsearch is an optional component
    - Adds advanced search capabilities for information about Workflow Executions
  - Prometheus and Grafana



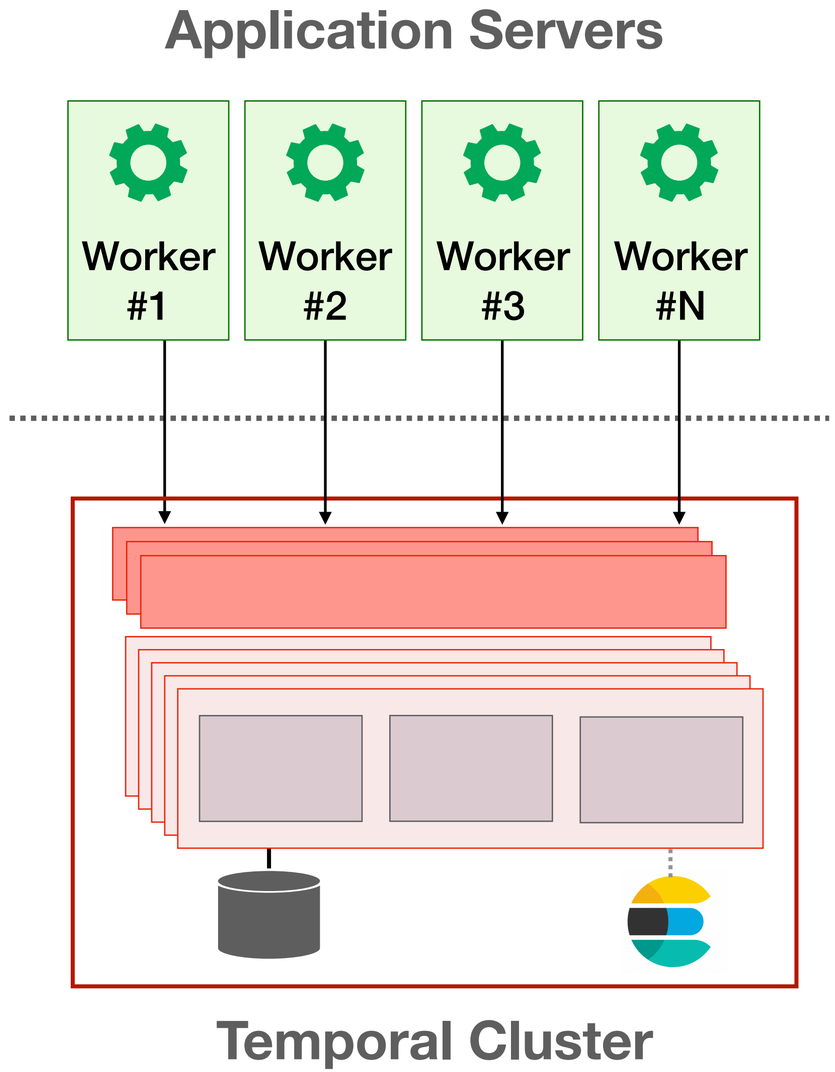
What are Workers?

- **Temporal Cluster *does not* execute your code**
  - It *orchestrates* the execution of your code
- ***Workers* execute your code**

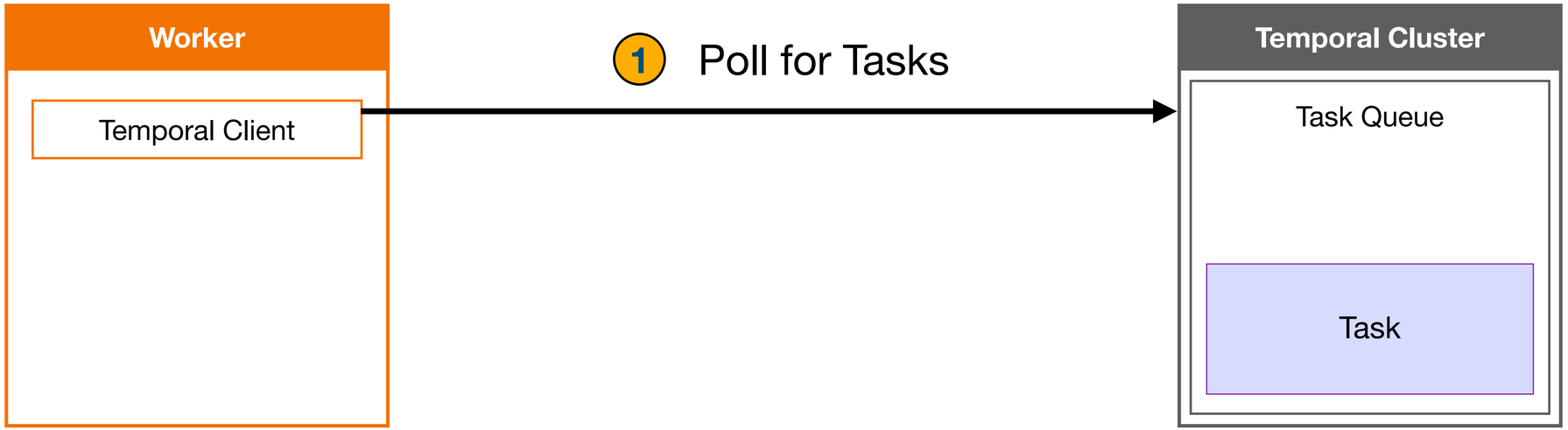


What are Workers?

- **Temporal Cluster *does not* execute your code**
  - It *orchestrates* the execution of your code
- ***Workers* execute your code**
  - They are part of your application and contain a Client
  - They coordinate with the Temporal Cluster
  - It's common to run them on multiple servers

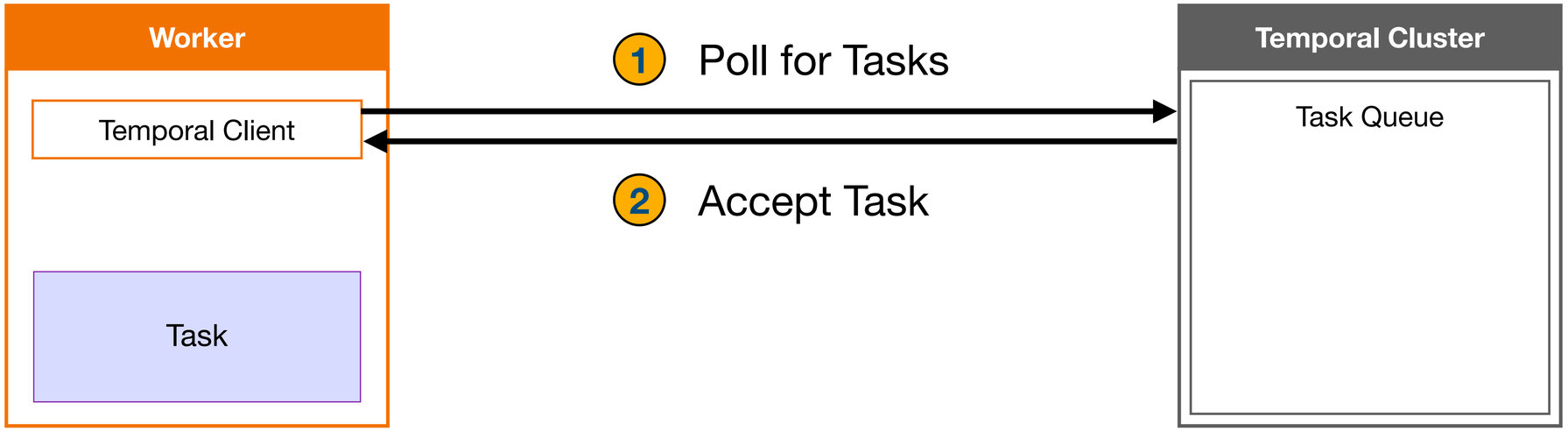


Task Queue



- A **Task Queue** is a queue Workers polls for tasks
- Temporal Cluster manages Task Queue
- Workers continuously poll for tasks, seeking work to perform

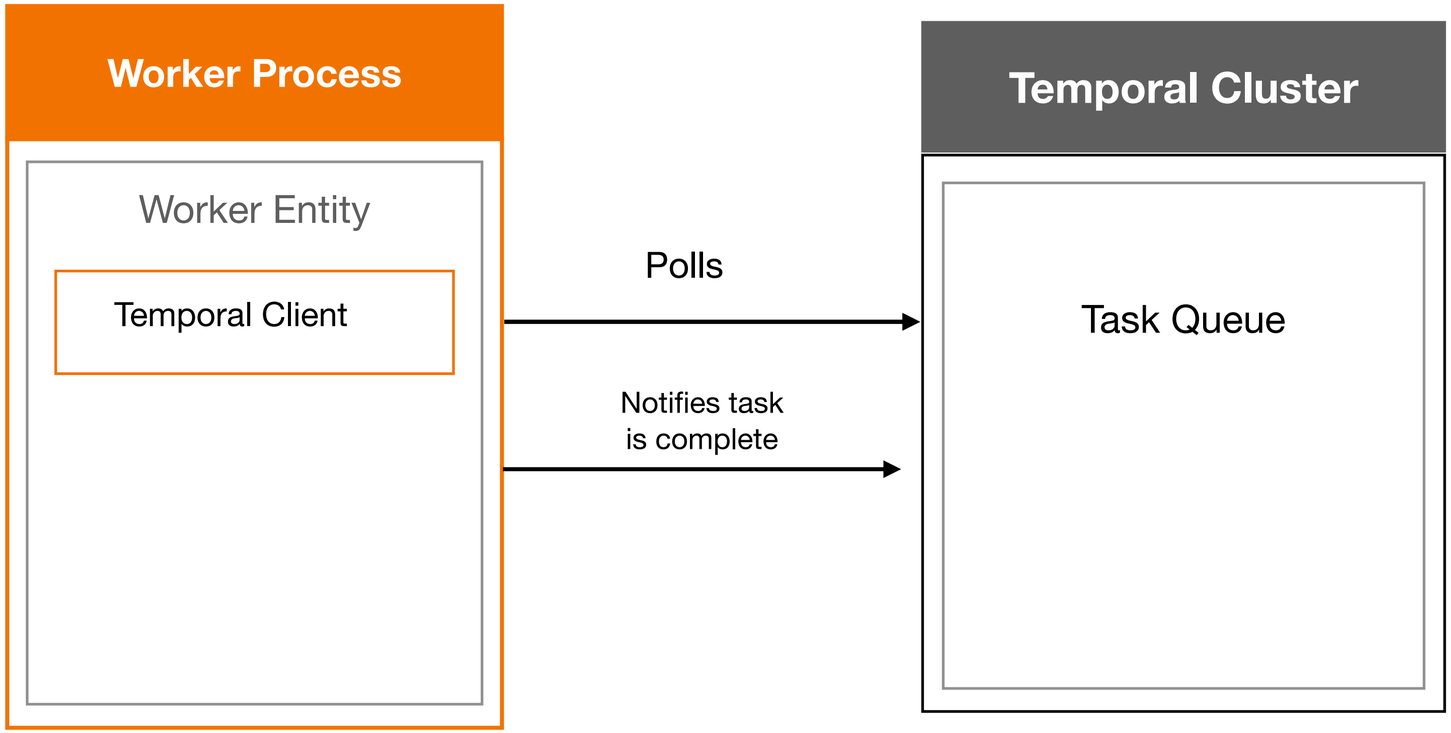
Workers and Tasks



- Workers accept task when they have the capacity

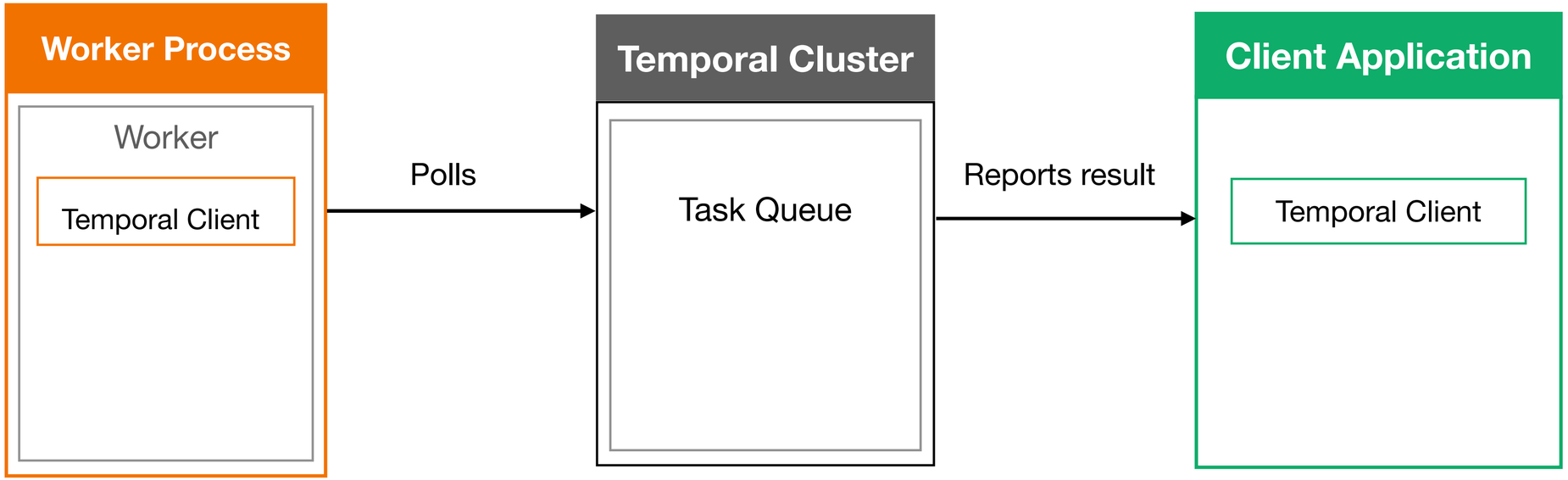
Task Queue

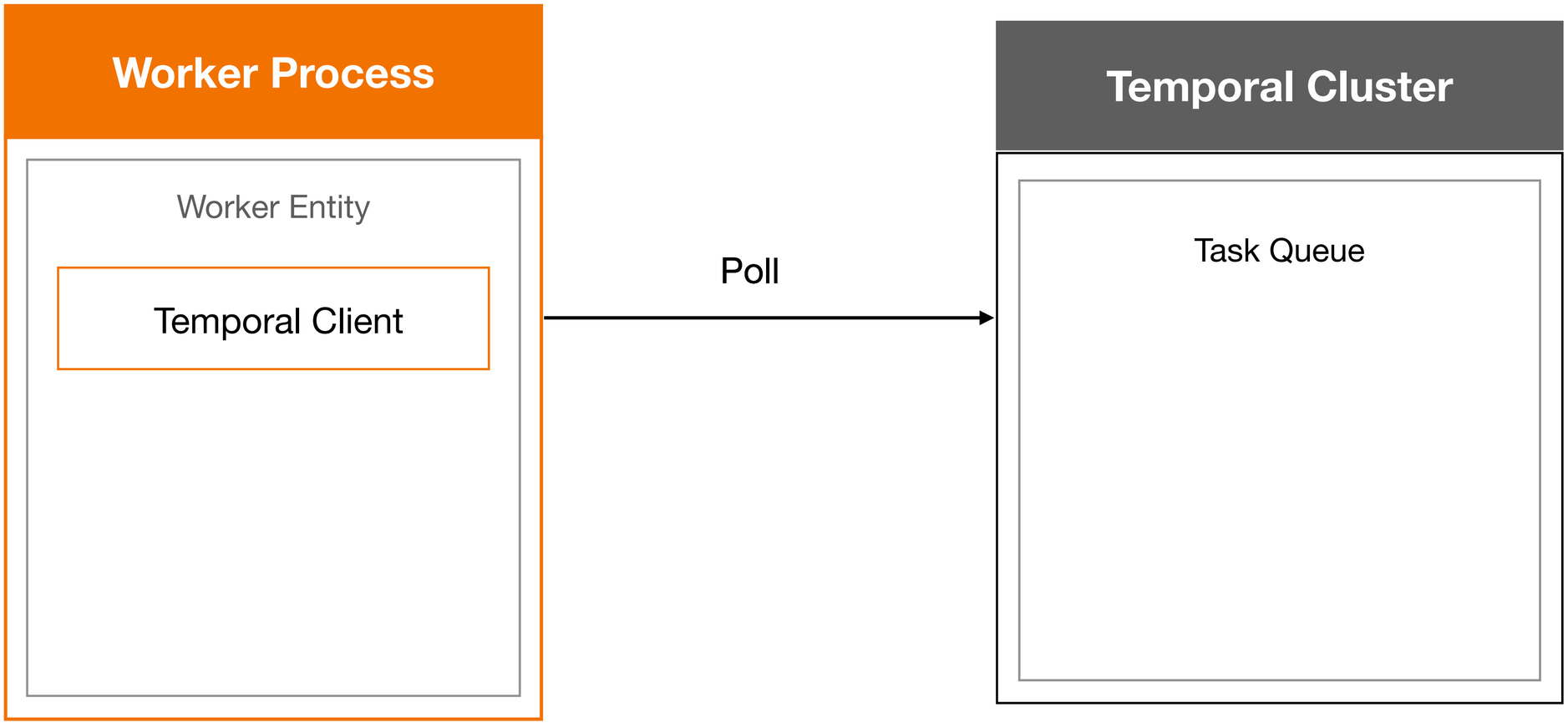
- After task is done, worker notifies Cluster task is done
- Workflow moves on to next task

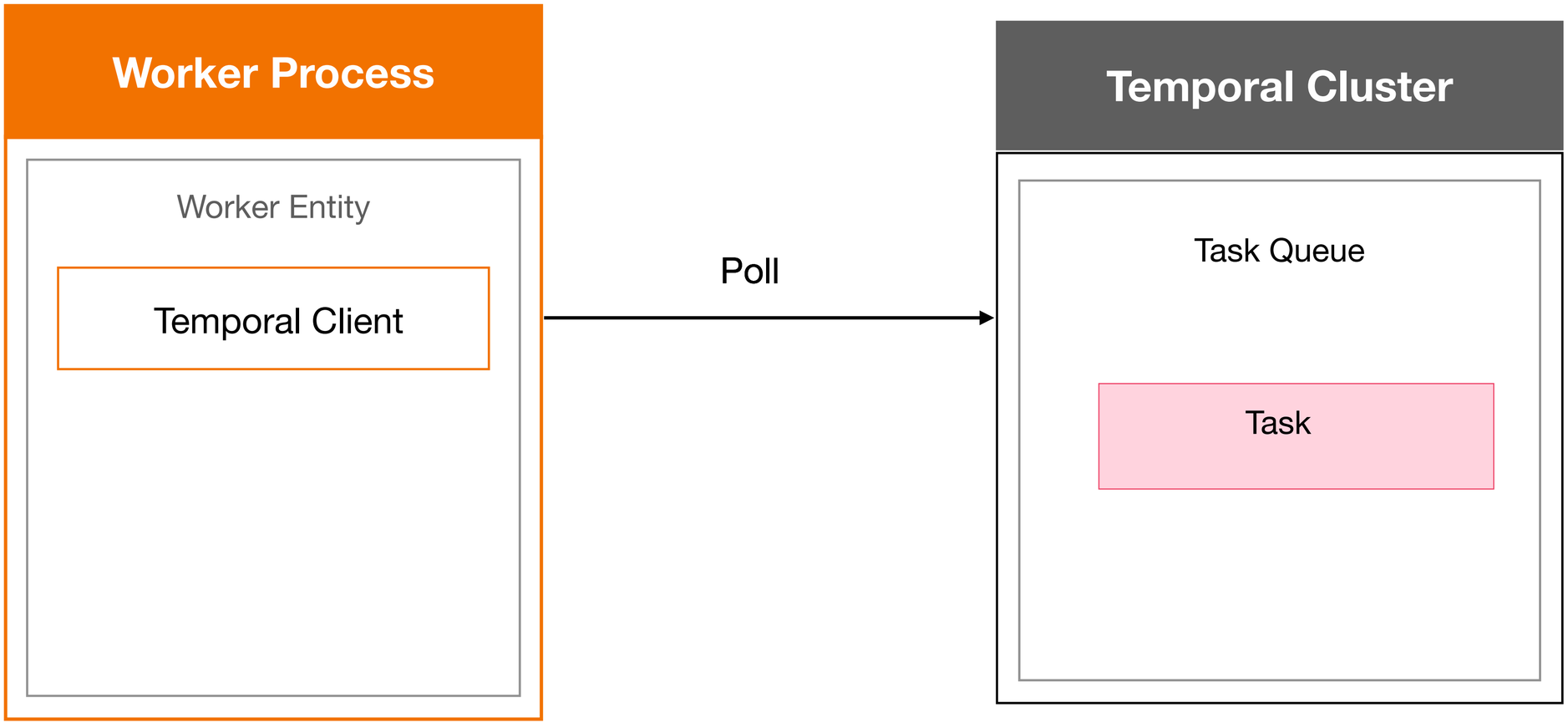


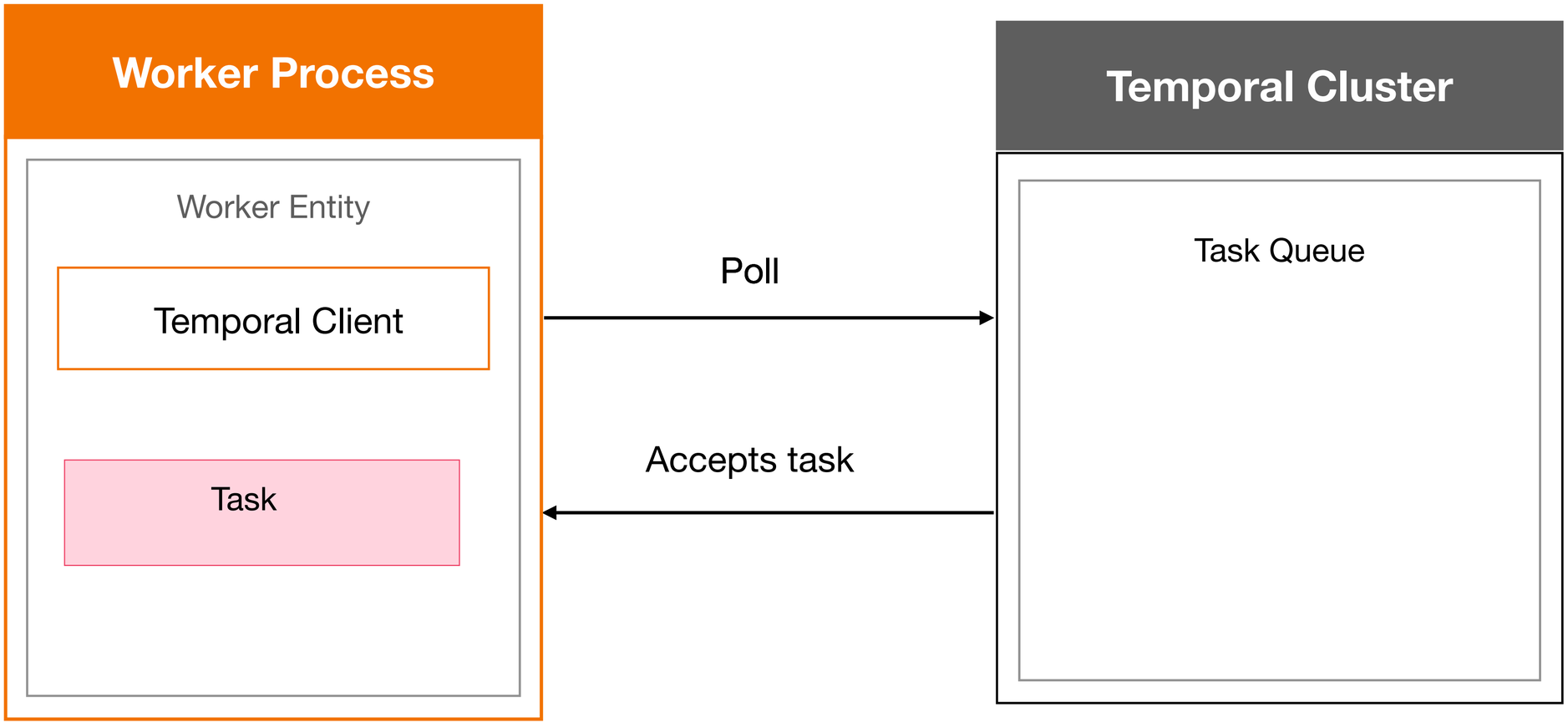
Task Queue

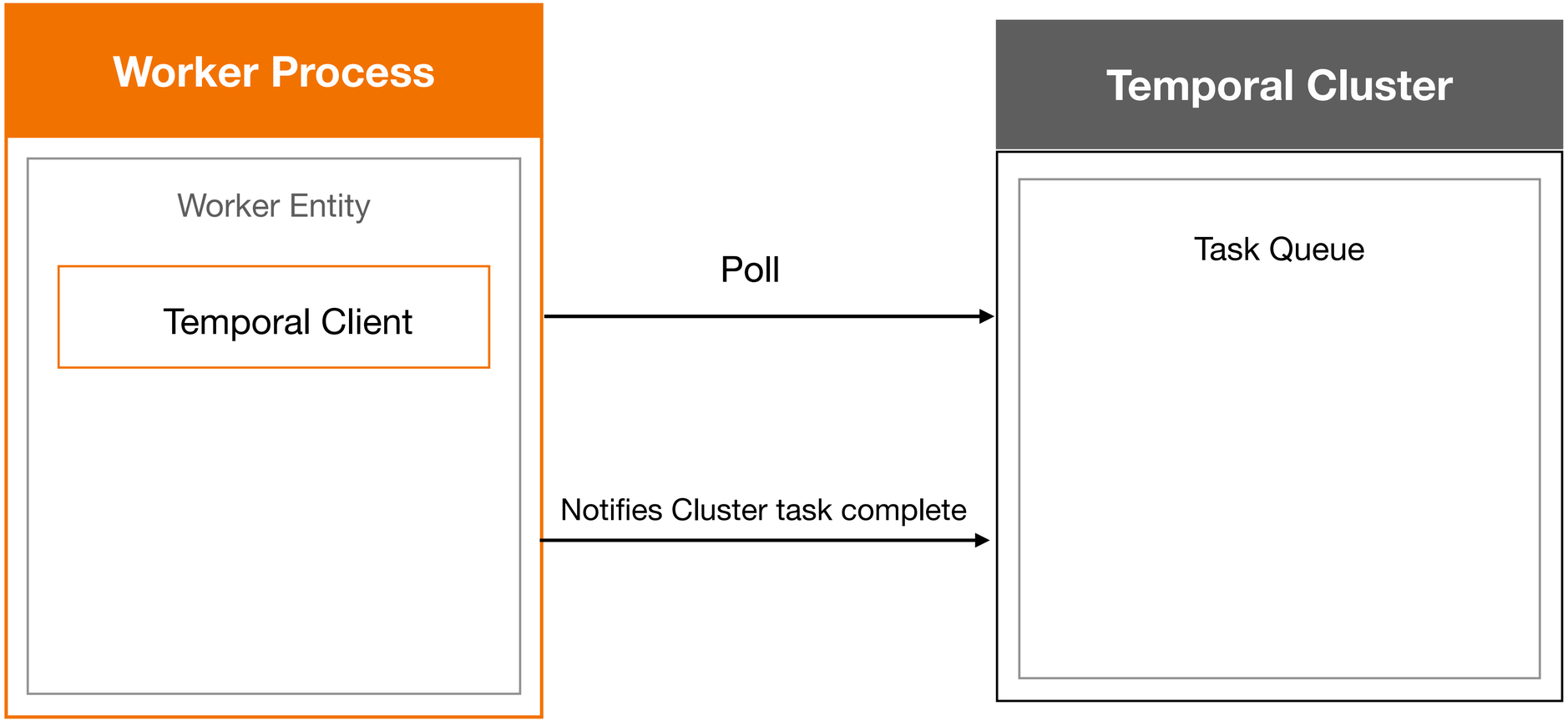
- When the Workflow is done, the result gets reported to the Temporal Client

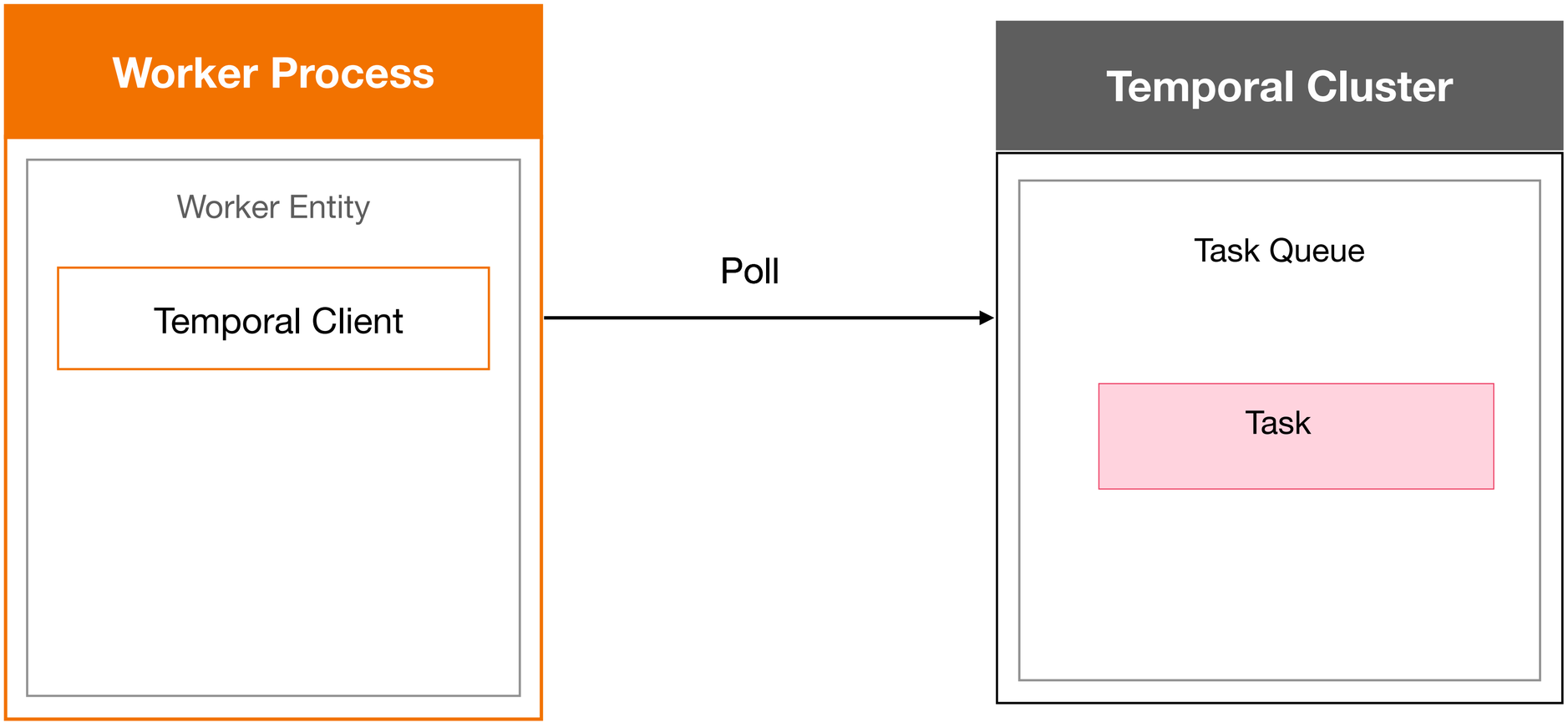


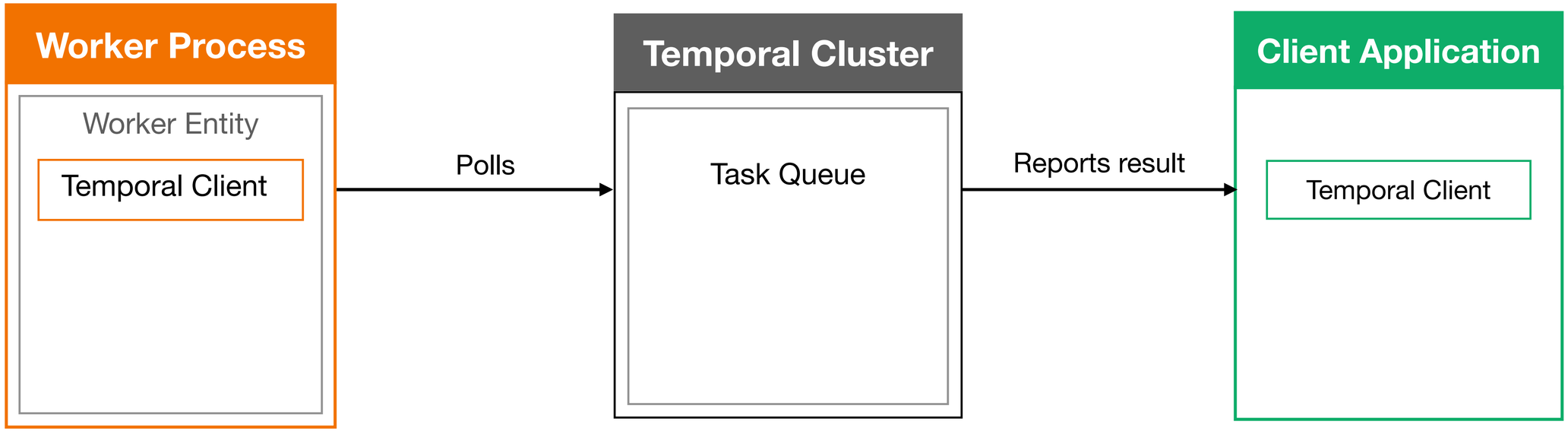

Worker Process
Worker Entity
Temporal Client
Poll
Temporal Cluster
Task Queue


Worker Process
Worker Entity
Temporal Client
Poll
Temporal Cluster
Task Queue
Task


Worker Process
Worker Entity
Temporal Client
Task
Poll
Accepts task
Temporal Cluster
Task Queue


Worker Process
Worker Entity
Temporal Client
Poll
Notifies Cluster task complete
Temporal Cluster
Task Queue


Worker Process
Worker Entity
Temporal Client
Poll
Temporal Cluster
Task Queue
Task


Worker Process
Worker Entity
Temporal Client
Polls
Temporal Cluster
Task Queue
Reports result
Client Application
Temporal Client

Temporal 101

Options for Running a Temporal Cluster

- **Self-Hosted**
  - Using Docker Compose is common for development
  - Temporal CLI - provides a small Temporal Cluster that runs in a single process
  - Production deployments often run on Kubernetes

Temporal Cluster

- **The new temporal CLI is the fast & easy way to run a development cluster**
  - Install this CLI tool (on a Mac; see docs for other systems)

```
$ brew install temporal
```

  - Start a development cluster (using default settings)

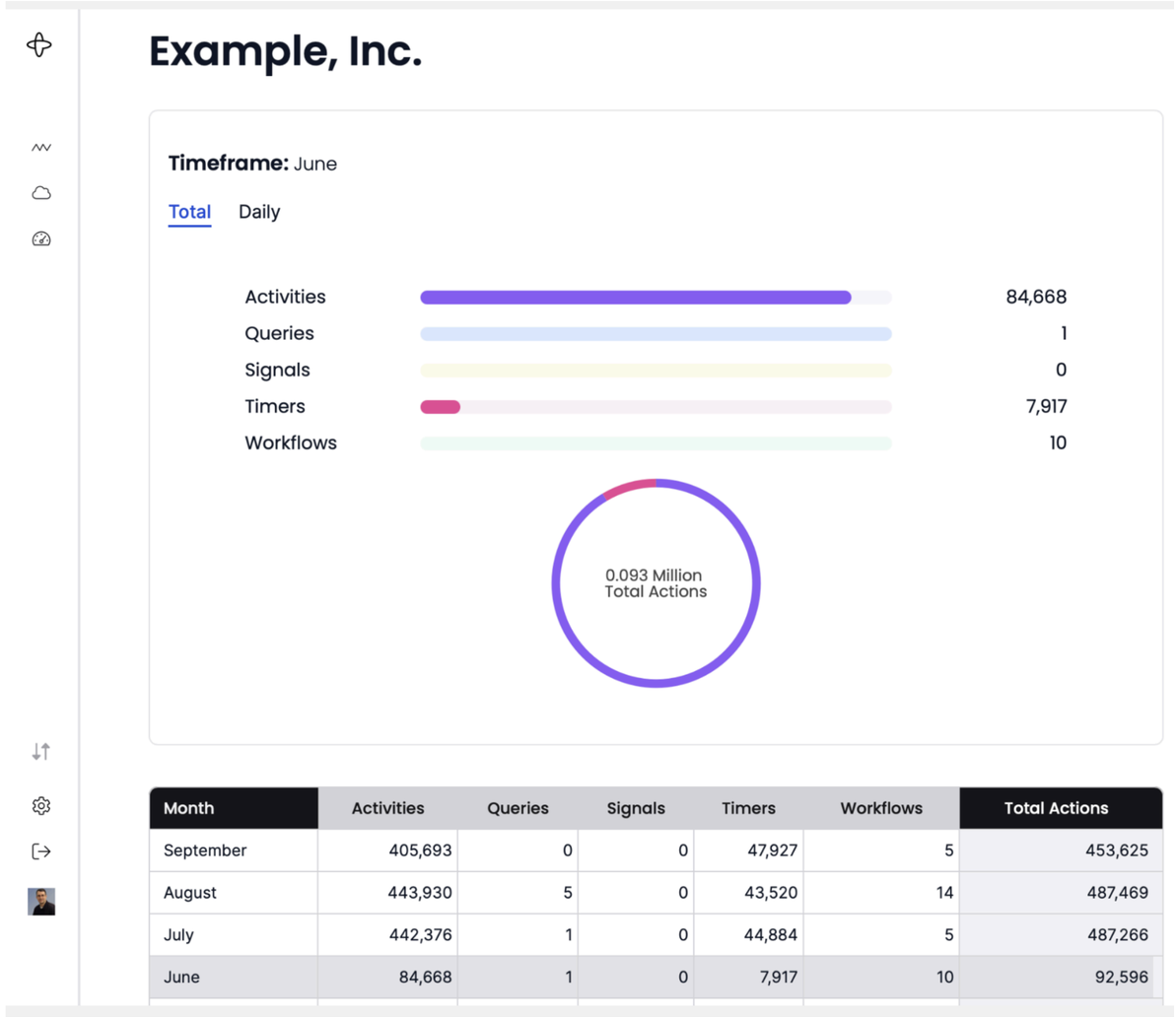
```
$ temporal server start-dev
```

  - Start a development cluster (specifying path for durable storage and a custom Web UI port)
  -

```
$ temporal server start-dev \
    --db-filename /Users/azhou/dev/mycluster.db \
    --ui-port 8080
```

Options for Running a Temporal Cluster

- **Temporal Cloud**
  - Access to a Temporal Cluster run by experts via our fully-managed cloud service
    - Frees your organization from having to plan, deploy, and operate your own cluster
  - Your application runs on your own infrastructure



Month	Activities	Queries	Signals	Timers	Workflows	Total Actions
September	405,693	0	0	47,927	5	453,625
August	443,930	5	0	43,520	14	487,469
July	442,376	1	0	44,884	5	487,266
June	84,668	1	0	7,917	10	92,596

Temporal 101

Temporal SDK

- A collection of **tools, libraries, and APIs** that **provides a framework** for Temporal application development.
- Language-specific
- SDK Provides:
  - A Temporal Client
  - APIs to develop Workflows
  - APIs to create and manage Worker Processes
  - API to author Activities
  - A common library for code that’s used across the Client, Worker, and/or Workflow

Installing Temporal SDK

- This command will create a new project with Temporal

```
$ npx @temporalio/create@latest ./your-app
```

- This command adds Temporal to an existing project

```
$ npm install @temporalio/client @temporalio/worker @temporalio/workflow @temporalio/activity @temporalio/
common
```

Temporal Command-Line Interface (tctl)

- **tctl is provides a CLI for interacting with a Temporal cluster**
  - You'll use it to start a Workflow in this workshop, but it has many other capabilities
  - See documentation for installation instructions
  - *brew install tctl*

Temporal Command-Line Interface (tctl)

- Append --help to any command or subcommand to see usage info
- This will soon be superseded by the temporal command

```
$ tctl --help
NAME:
   tctl - A command-line tool for Temporal users

USAGE:
   tctl [global options] command [command options...

VERSION:
   1.18.0

COMMANDS:
   namespace, n       Operate Temporal namespace
   workflow, wf       Operate Temporal workflow
   activity, act      Operate activities of workflow
   ...
```

Temporal 101

Workflow Definitions

- **With Temporal's TypeScript SDK, you create a Workflow by writing a TS function**
  - The code for this function is known as a ***Workflow Definition***
  - Each Workflow has a name, known as its ***Workflow Type***
  - Each invocation of a Workflow is known as a ***Workflow Execution***

Input Parameters and Return Values

- **Input parameters and return values must be serializable**
  - Allowed: Null values, binary data, and anything serializable via JSON or Protocol Buffers
  - Prohibited: Date, BigInt.
- **Avoid passing in or returning large amounts of data from your Workflow**
  - May rapidly expand the size of your Temporal Cluster's database

Business Logic

- **We will begin with an example**
  - Input: string (a person's name)
  - Output: string (a greeting containing that name)
- **This is simply a TypeScript function**
  - It is not (yet) a Temporal Workflow

```
function greetSomeone(name: string): string {
  return "Hello " + name + "!"
}
```

Writing Activities

- **Activity Definitions are TypeScript functions**
  - Rules for input and output types are the same as for Workflow Definitions
  - Temporal does not impose a naming convention on the function name
- **Activities are single, well-defined actions (either short or long-running) that generally interface with external services, such as calling an API.**

```
import axios from 'axios';

const url = 'http://localhost:9999';

export async function getSpanishGreeting(name:
string): Promise<string> {
  const response = await axios.get(`${url}/get-
spanish-greeting?name=${name}`);

  return response.data;
}
```

Writing a Workflow Function

- **Four steps for turning a TS function into a Workflow Definition**

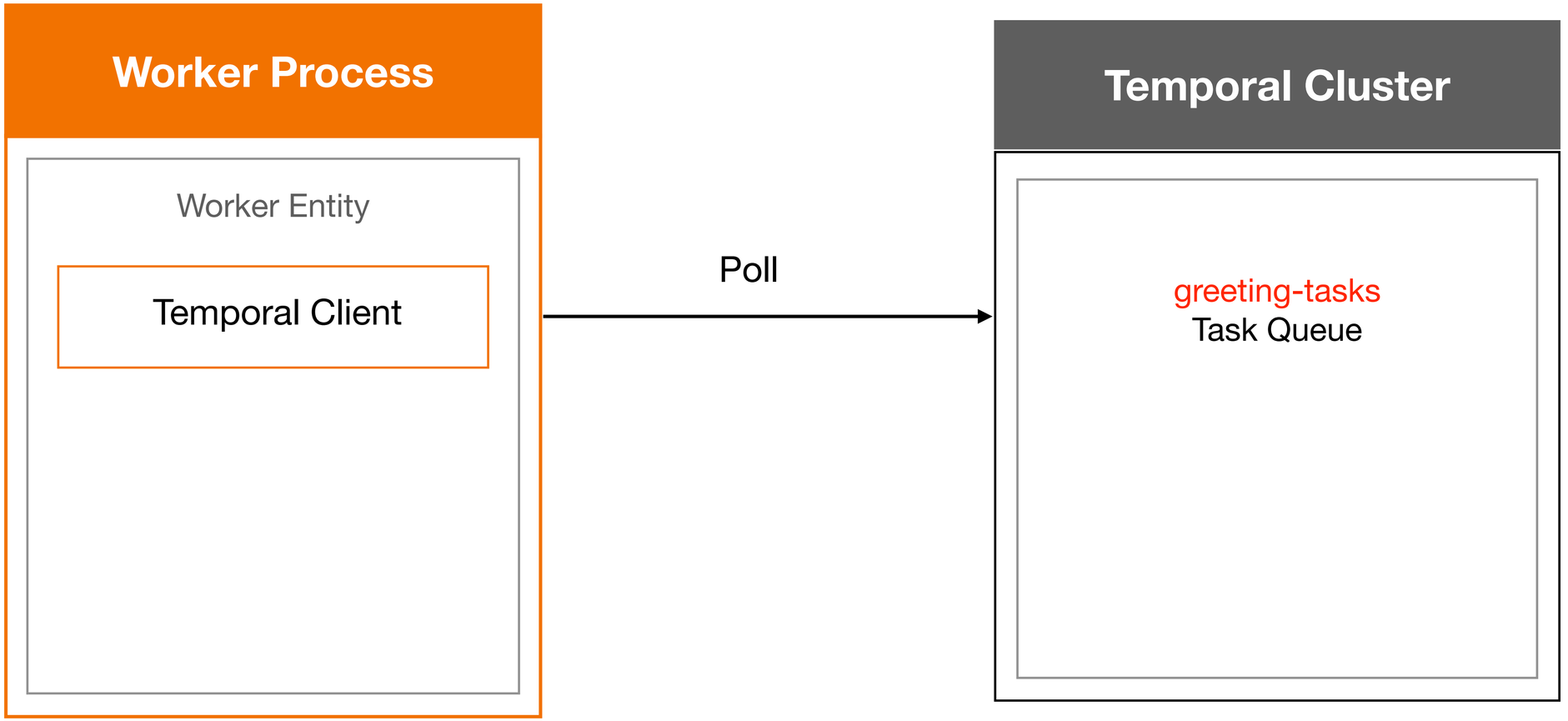
1. Import the proxyActivities package from the Temporal TypesScript SDK
2. Import your Activity Types from the Activities module
3. Define your Activity Options using the proxyActivities function
4. Write and export a function that calls your Activity

```
import { proxyActivities } from '@temporalio/workflow';
// Only import the activity types
import type * as activities from './activities';

const { getSpanishGreeting } = proxyActivities<typeof activities>({
  startToCloseTimeout: '1 minute',
});

/** A workflow that simply calls an activity */
export async function greetSomeone(name: string): Promise<string> {
  return await getSpanishGreeting(name);
}
```

Initializing the Worker

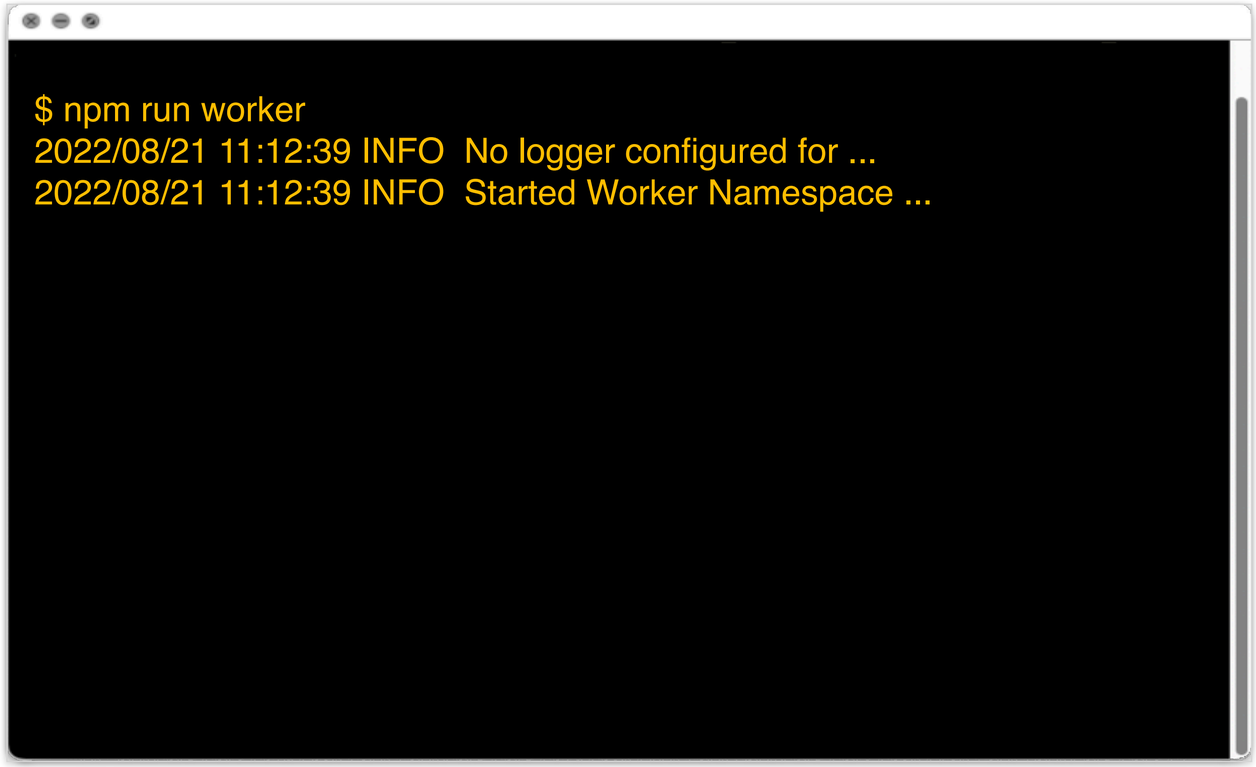
- **Workers execute your code**
- **How to initialize a Worker**
  1. Import the Worker package
  2. Register Workflows and Activities within the Worker and connect to the Temporal server
  3. Specify the name of a task queue on the Temporal Cluster
  4. Call the function to start the Worker

```
import { Worker } from '@temporalio/worker';                    ← 1
import * as activities from './activities';

async function run() {
  // Step 1: Register Workflows and Activities
  const worker = await Worker.create({                          ┐
    workflowsPath: require.resolve('./workflows'),              │ 2
    activities,                                                 ┘
    taskQueue: 'greeting-tasks',                                ← 3
  });
  // Step 2: Start accepting tasks
  await worker.run();
}

run().catch((err) => {                                          ← 4
  console.error(err);
  process.exit(1);
});
```


Worker Process
Worker Entity
Temporal Client
Poll
Temporal Cluster
greeting-tasks
Task Queue

Starting the Worker Program

- **Since Workers runs your code, there is no progress unless Worker is running**
  - After starting it, the Worker outputs a few lines and then appears to do nothing
  - This is expected behavior, as it is busy polling the task queue and executing your code
  - The Worker will keep running after this Workflow completes, because it then waits for more work to appear in the task queue



Starting the Worker Program

- A worker can take over for a crashed Worker
- If there are no other Workers available, Workflow Execution will continue where it left off as soon as original Worker is restarted

Temporal 101

Executing a Workflow from the Command Line

- **One way to start a Workflow is with tctl workflow start**
  - The task queue value must match the value specified in your Worker initialization code
  - The workflow_id is a user-defined identifier, which typically has some business meaning
  - The input argument's value is deserialized and passed as a Workflow function parameter

```
$ tctl workflow start \
    --workflow_type greetSomeone \
    --taskqueue greeting-tasks \
    --workflow_id my-first-workflow \
    --input "'Donna"'

Started Workflow Id: my-first-workflow,
run Id: e8f9217e-344e-4f7b-98bc-7703bc8c7c76
```

Executing a Workflow from Application Code

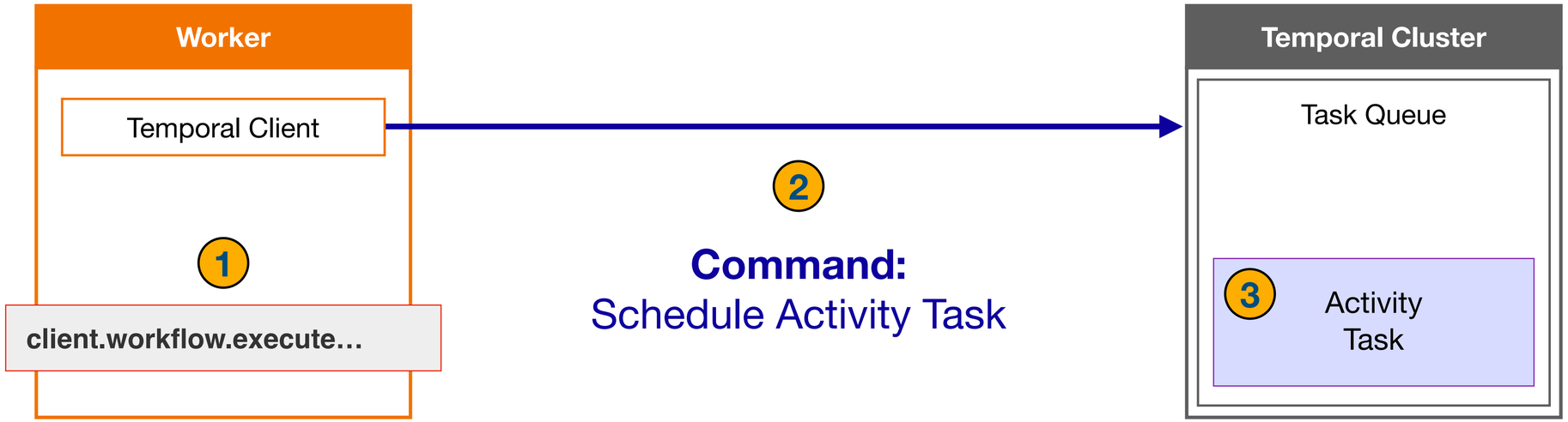
- **An alternative to using tctl is to execute the Workflow from code**
  - This provides a way of integrating Temporal into your own applications
  - You can do this in three steps:
    - Import the Client class from the SDK's @temporalio/client package
    - Create an async function called run and within the function:
      - Set up a connection
      - Set up a client
      - Start a Workflow
    - Run the function

```
import { Client } from '@temporalio/client';                    ← 1
import { greetSomeone } from './workflows';
import { nanoid } from 'nanoid';

async function run() {                                          ← 2
  const client = new Client();

  const handle = await client.workflow.start(greetSomeone, {
    args: ['Tina'],
    taskQueue: 'greeting-tasks',
    // in practice, use a meaningful business id, eg customerId or transactionId
    workflowId: 'workflow-' + nanoid(),
  });
  console.log(`Started workflow ${handle.workflowId}`);

  // optional: wait for client result
  console.log(await handle.result()); // Hello,Tina!
}

run().catch((err) => {                                          ← 3
  console.error(err);
  process.exit(1);
});
```

Exercise #1: Hello Workflow

- **During this exercise, you will**
  - Review the business logic of the provided Workflow Definition to understand its behavior
  - Modify the Worker initialization code to specify a task queue name (**greeting-tasks**)
  - Run the Worker initialization code to start the Worker process
  - Use **tctl** to execute the Workflow from the command line, specifying your name as input
- **Refer to the README.md file in the exercise environment for details**
  - The code is below the **exercises/hello-workflow** directory
    - Make your changes to the code in the **practice** subdirectory (look for TODO comments)
    - If you need a hint or want to verify your changes, look at the complete version in the **solution** subdirectory

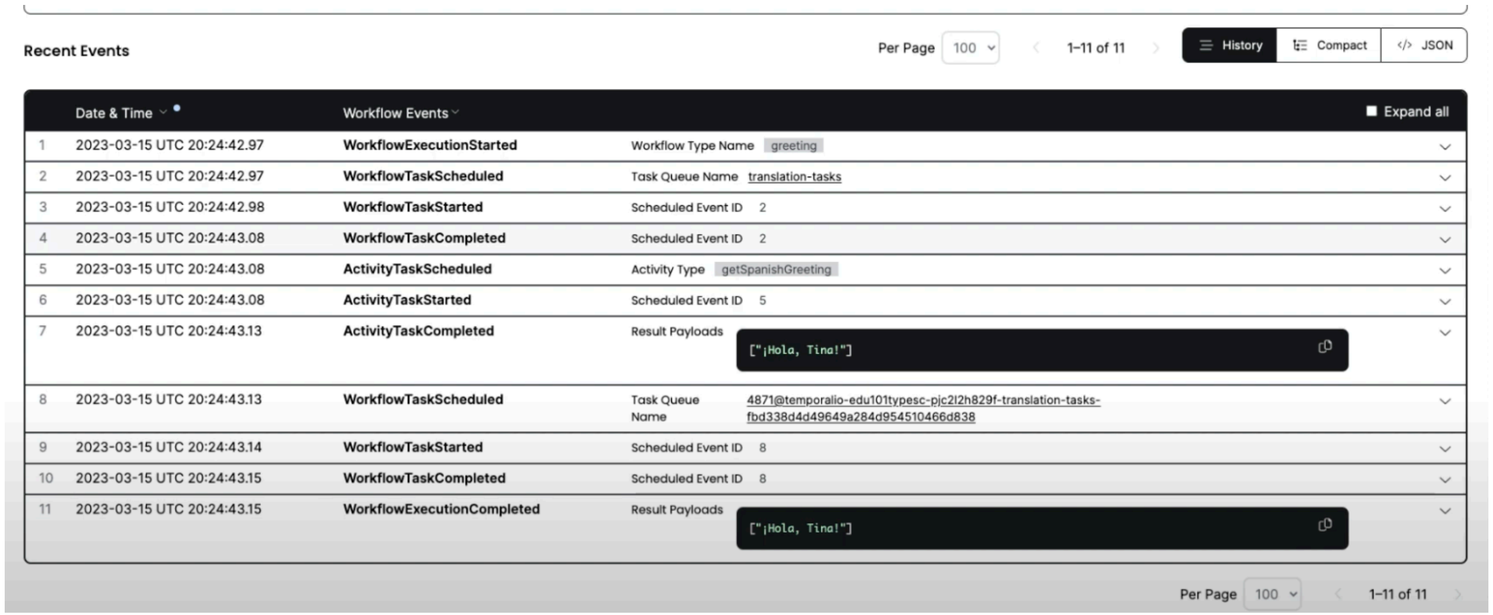
Commands



- Certain API calls result in the Worker issuing a Command to the Temporal Cluster
- The Cluster acts on these Commands, but also stores them
- This allows the Worker to recreate the state of a Workflow Execution following a crash

Event History

- **Temporal stores the history of your Workflow Executions**
  - Events are stored in a durable history
  - Events represent changes to Workflow state
  - If a failure occurs, Temporal uses the history to replay the Workflow code



Viewing Workflow History with tctl

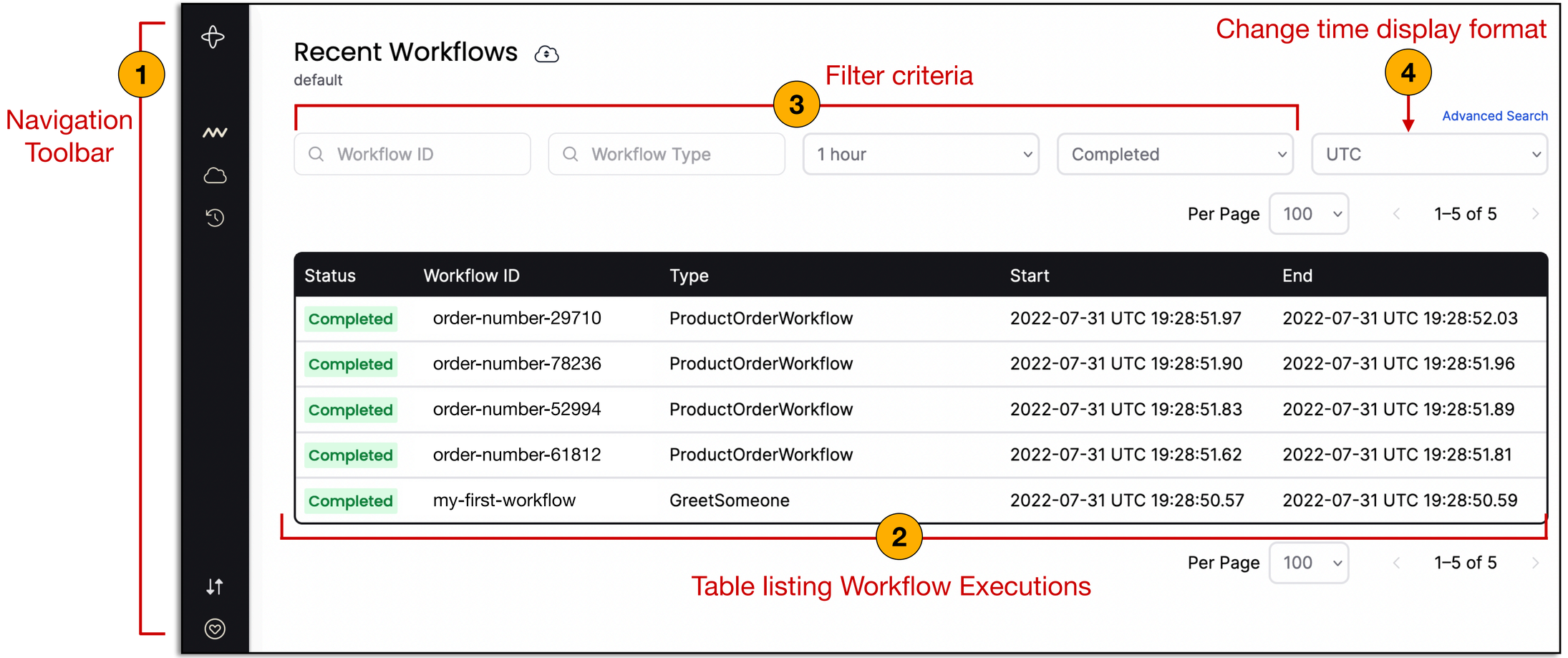
- **The Temporal Web UI displays Workflow status and history**
  - It's also a powerful tool for gaining insight into Workflow Execution

```
$ tctl wf show --workflow_id my-first-workflow

  1  WorkflowExecutionStarted    {WorkflowType:{Name:GreetSomeone},
                                 ParentInitiatedEventId:0, TaskQueue:{Name:greeting-tasks,
                                 Kind:Normal}, Input:["Donna"],
                                 WorkflowExecutionTimeout:0s, WorkflowRunTimeout:0s,
                                 WorkflowTaskTimeout:10s, Initiator:Unspecified,
                                 OriginalExecutionRunId:e8f9217e-344e-4f7b-98bc-7703bc8c7c76,
                                 Identity:tctl@twwmbp,
                                 FirstExecutionRunId:e8f9217e-344e-4f7b-98bc-7703bc8c7c76,
                                 Attempt:1, FirstWorkflowTaskBackoff:0s,
                                 ParentInitiatedEventVersion:0}
  2  WorkflowTaskScheduled       {TaskQueue:{Name:greeting-tasks,
                                 Kind:Normal},
                                 StartToCloseTimeout:10s,
                                 Attempt:1}
  3  WorkflowTaskStarted         {ScheduledEventId:2, Identity:93592@twwmbp@,
                                 RequestId:10535889-9c10-4073-b38f-4876bbae4db3,
                                 SuggestContinueAsNew:false, HistorySizeBytes:0}
```

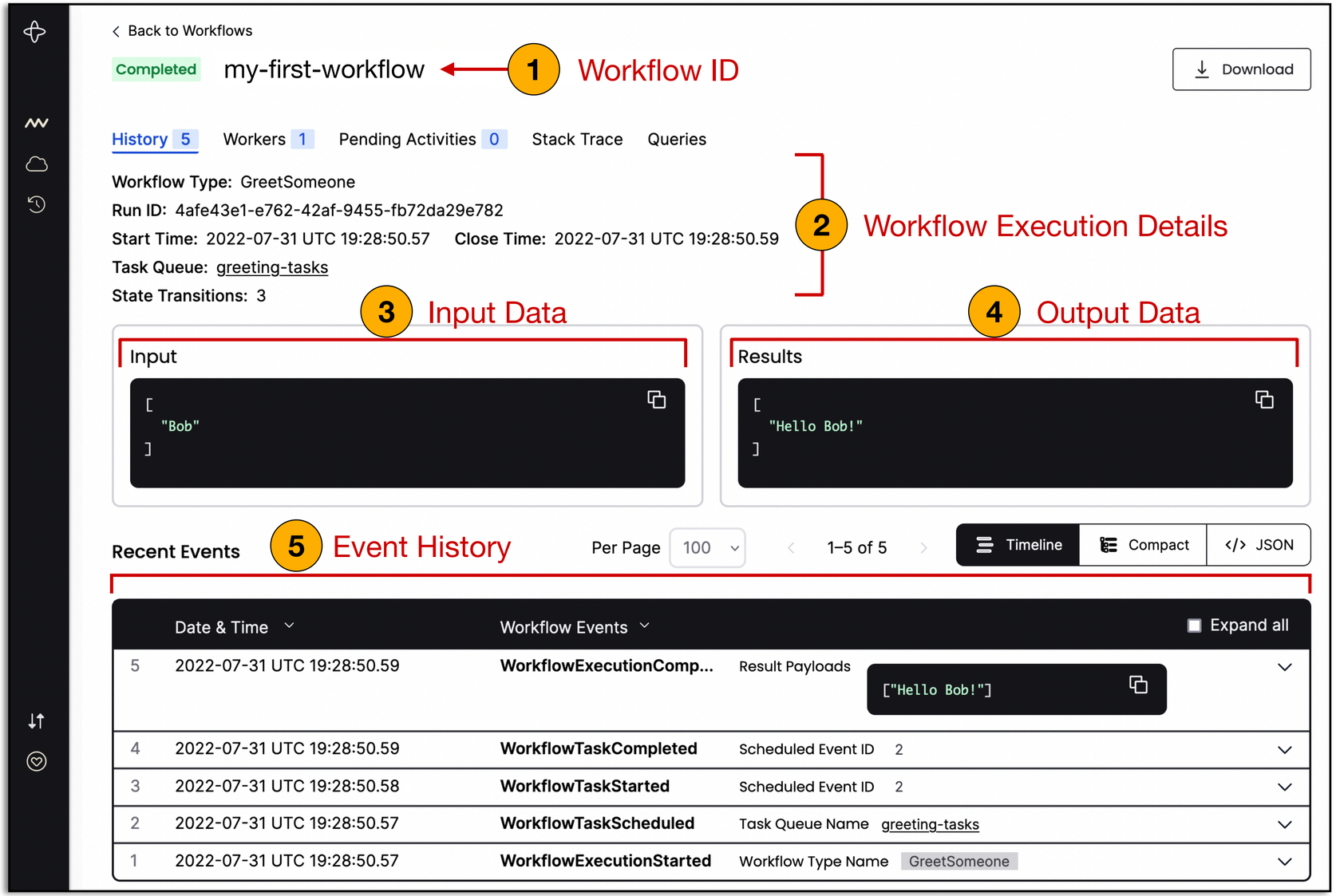
Viewing Workflow History from Web UI

- **The port number used to access it may vary by deployment type**
  - If using Docker Compose on your laptop: http://localhost:8080/
  - If you're using Temporal Cloud, you'll be using cloud.temporal.io
  - If you're using Temporal CLI, you'll be using http://localhost:8233
  - In our GitPod environment, the Web UI is shown in an embedded browser tab

Web UI: Main Page

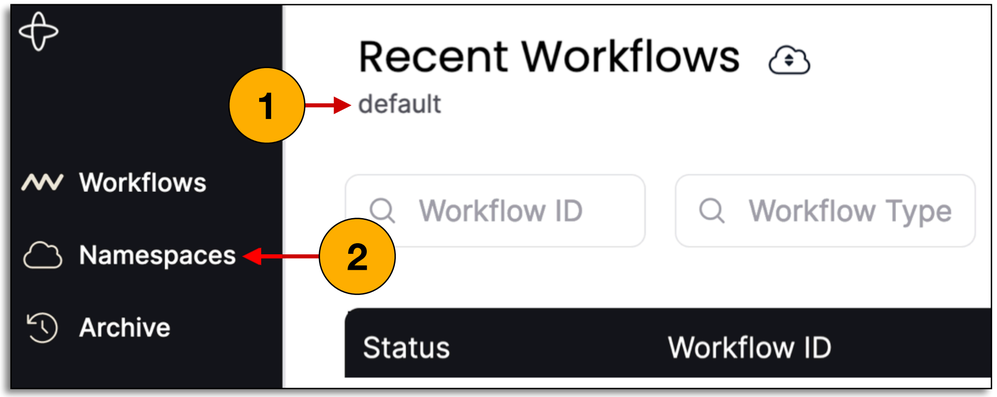


Web UI: Workflow Execution Detail Page



Namespaces

- **The Web UI lists recent Workflow Executions within a given *namespace***
  - You can see the selected namespace (1) and switch among available namespaces (2)
- **Namespaces are a means of isolation within a Temporal cluster**
  - Used to logically separate Workflows according to your needs
    - For example, by lifecycle (development vs. production) or department (Marketing vs. Accounting)
  - Some settings are applied at a per-namespace level
  - The default namespace is named default



Exercise #2: Hello Web UI

- **During this exercise, you will**
  - Use the Temporal Web UI to display the list of recent Workflow Executions
  - View the detail page for the Workflow Execution from the previous exercise
  - See if you can find the following information on the detail page
    - Name of the task queue
    - Start time
    - Close time (this is the time of completion)
    - Input and output for this Workflow execution (hint: click the "</> Input and Results" section)

Making Changes to a Workflow

- **Backwards compatibility is an important consideration in Temporal**
- **Avoid changing the number or types of input parameters**
  - We recommend that your Workflow uses an object as the only input parameter
  - Changing the keys used to create the object does not change its type

```
export type Person = {
  name: string;
  age: number;
};

export async function someWorkflow(person: Person):
Promise<void> {
    //some logic
}
```

```
export type Person = {
  name: string;
  age: number;
  email: string;
};

export async function someWorkflow(person: Person):
Promise<void> {
    //some logic
}
```

Restarting the Worker Process

- **Workers cache the application state for better performance**
  - After making changes, you must restart the Worker(s) before changes take effect

Temporal 101

How Temporal Handles Activity Failure

- By default, Temporal automatically retries failed Activities forever with a short delay between retries

How Temporal Handles Activity Failure

- **Four properties determine the timing and number of retry attempts**
  - You can override one or more of these defaults with a custom Retry Policy

Property	Description	Default Value
InitialInterval	Duration before the first retry	1 second
BackoffCoefficient	Multiplier used for subsequent retries	2.0
MaximumInterval	Maximum duration between retries	100 * InitialInterval
MaximumAttempts	Maximum number of retry attempts before giving up	0 (unlimited)

Activity Retry Policy Example

```
import { proxyActivities } from '@temporalio/workflow';
import type * as activities from './activities';

const { withdraw, deposit } = proxyActivities<typeof activities>({
    retry: {
    initialInterval:    ‘15s’,  // first retry will occur after 15 seconds
    backoffCoefficient: 3,      // triple the delay after each retry
    maximumInterval:    ‘1m’,   // up to a maximum delay of 60 seconds
    maximumAttempts:    100,   // fail the Activity after 100 attempts
    nonRetryableErrorTypes: [“InsufficientFundsError”,“InvalidAccountError”],
},
   startToCloseTimeout: ‘90s’
});

export async function makeTransfer(name: string): Promise<string> {
  const withdrawOutput = await withdraw(details);
  const depositOutput = await deposit(details);
  return `transfer complete (transactionIDs: ${withdrawOutput},
  ${depositOutput}`
};
```

1 Import this package from the SDK

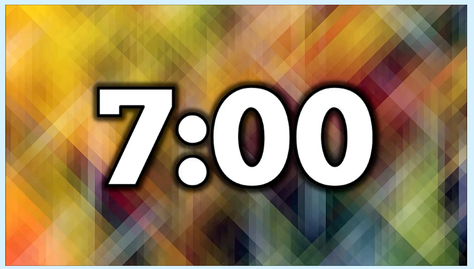
2 Specify your policy values

3 Create a Workflow that will call your Activity

Optional Demo: Fixing Errors at Runtime

- **If there is time, your instructor will demonstrate how Temporal allows you to fix application errors at runtime, even for running workflows**
  1. Edit the Activity to throw a new Error
  2. Run the Workflow and use the Web UI to show the error
  3. Fix the error
  4. Restart the Worker
  5. Observe that the running Workflow now completes successfully

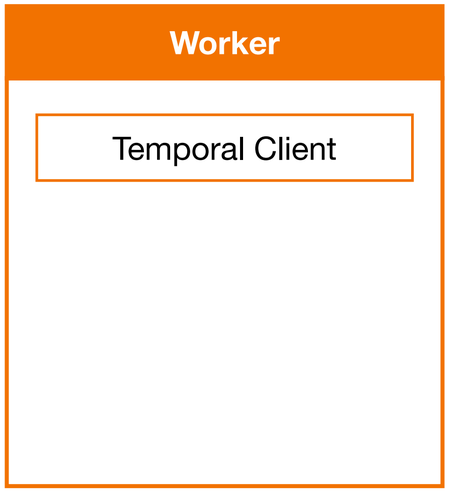
Exercise #3: Farewell Workflow



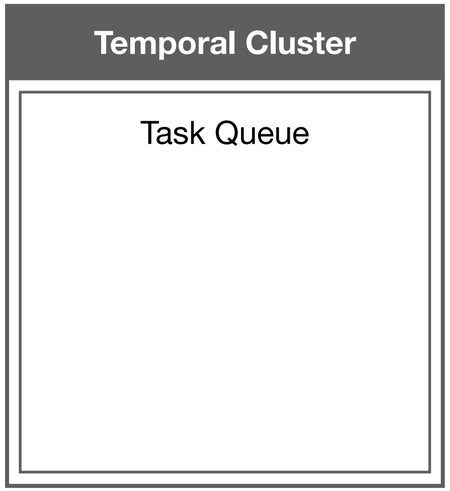
- **During this exercise, you will**
  - Write an Activity function
  - Register the Activity function
  - Modify the Workflow to execute your new Activity
  - Run the Workflow
- **Refer to the README.md file in the exercise environment for details**
  - The code is below the **exercises/farewell-workflow** directory
    - Make your changes to the code in the **practice** subdirectory (look for TODO comments)
    - If you need a hint or want to verify your changes, look at the complete version in the **solution** subdirectory

Temporal 101

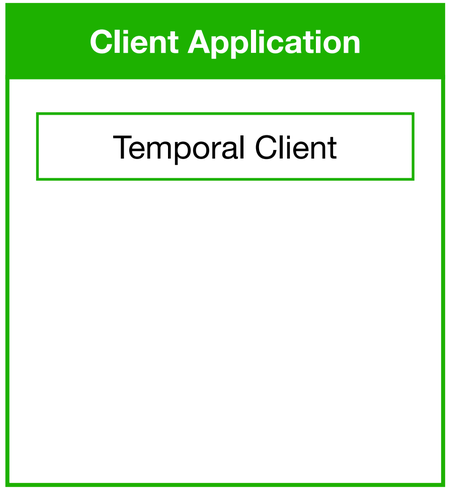
Actors in this Workflow Execution Scenario



Executes the code



Orchestrates code execution



Requests code execution
and retrieves the result

Activity Definitions

Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="us"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }
    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

```
import axios from 'axios';

const url = 'http://localhost:9999';

export async function getSpanishGreeting(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-greeting?name=${name}
`);

  return response.data;
}

export async function getSpanishFarewell(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-farewell?name=${name}
`);

  return response.data;
}
```

Workflow Definition

Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

```typescript
import { proxyActivities } from '@temporalio/workflow';
import type * as activities from './activities';

const { getSpanishGreeting, getSpanishFarewell } =
proxyActivities<
  typeof activities
>({
  startToCloseTimeout: '10 seconds',
});

export async function greeting(name: string):
Promise<string> {
  const greeting = await getSpanishGreeting(name);
  const farewell = await getSpanishFarewell(name);
  const helloGoodbye = "\n" + greeting + "\n" + farewell;
  return helloGoodbye;
}
```

Worker Initialization

Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
  greeting, err := callService("get-spanish-greeting", name)
  return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
  greeting, err := callService("get-spanish-farewell", name)
  return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
  base := "http://localhost:9999/" + stem + "?name="
  url := fmt.Sprintf(base, url.QueryEscape(name))

  resp, err := http.Get(url)
  if err != nil {
    return "", err
  }
  defer resp.Body.Close()

  body, err := ioutil.ReadAll(resp.Body)
  if err != nil {
    return "", err
  }

  translation := string(body)
  status := resp.StatusCode
  if status >= 400 {
    message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
    return "", errors.New(message)
  }

  return translation, nil
}
```

Workflow Definition

```
package farewell

import (
  "time"

  "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
  options := workflow.ActivityOptions{
    StartToCloseTimeout: time.Second * 5,
  }
  ctx = workflow.WithActivityOptions(ctx, options)

  var spanishGreeting string
  err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
  if err != nil {
    return "", err
  }

  var spanishFarewell string
  err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
  if err != nil {
    return "", err
  }

  var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

  return helloGoodbye, nil
}
```

Worker Initialization

```
package main

import (
  "log"
  farewell "temporal101.exercises/farewell-workflow/solution"

  "go.temporal.io/sdk/client"
  "go.temporal.io/sdk/worker"
)

func main() {
  c, err := client.Dial(client.Options{})
  if err != nil {
    log.Fatalln("Unable to create client", err)
  }
  defer c.Close()

  w := worker.New(c, "greeting-tasks", worker.Options{})

  w.RegisterWorkflow(farewell.GreetSomeone)
  w.RegisterActivity(farewell.GreetInSpanish)
  w.RegisterActivity(farewell.FarewellInSpanish)

  err = w.Run(worker.InterruptCh())
  if err != nil {
    log.Fatalln("Unable to start worker", err)
  }
}
```

```
import { Worker } from '@temporalio/worker';
import * as activities from './activities';

async function run() {
  const worker = await Worker.create({
    workflowsPath: require.resolve('./workflows'),
    activities,
    taskQueue: 'translation-tasks',
  });

  await worker.run();
}

run().catch((err) => {
  console.error(err);
  process.exit(1);
});
```

Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

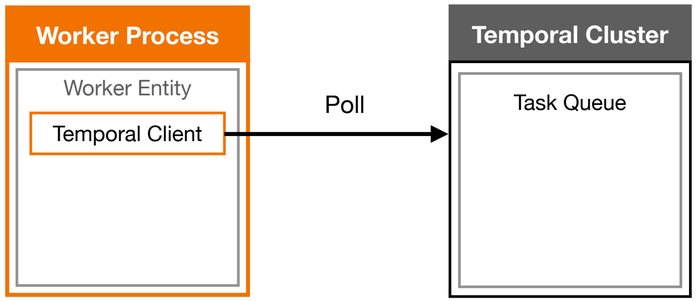
```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```


```
import { Worker } from '@temporalio/worker';
import * as activities from './activities';

async function run() {
  const worker = await Worker.create({
    workflowsPath: require.resolve('./workflows'),
    activities,
    taskQueue: 'translation-tasks',
  });

  await worker.run();
}

run().catch((err) => {
  console.error(err);
  process.exit(1);
});
```

npm run start.watch



Launching from Command Line

Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

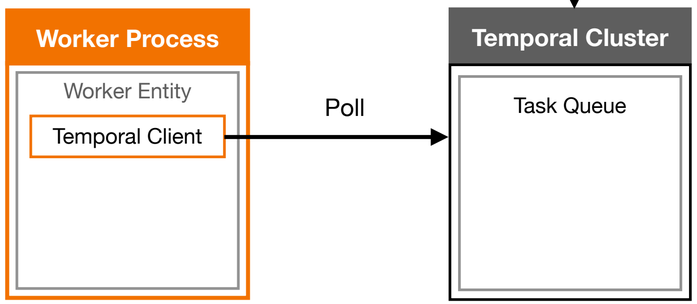
```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

```
$ tctl workflow run \
    --taskqueue translation-tasks \
    --workflow_id greeting-task-tina
    --workflow_type greeting \
    —input “‘Tina”’
```

Workflow Execution Request



Launching from Application Code

Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }
    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

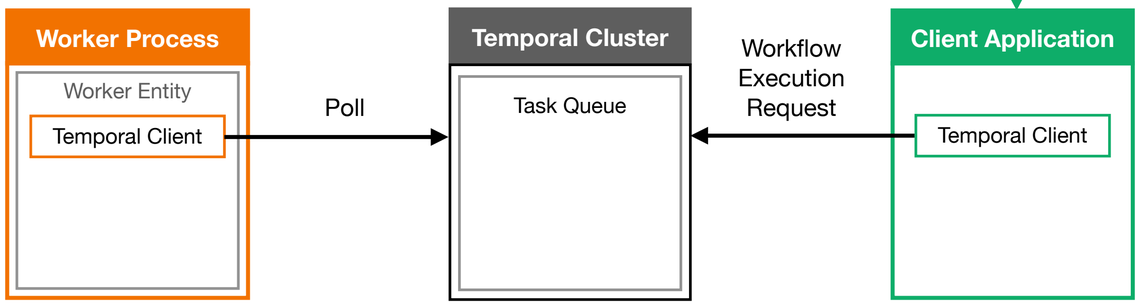
```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

```
import { Client } from '@temporalio/client';
import { randomUUID } from 'node:crypto';
import { greeting } from '../workflows';

async function run() {
  const client = new Client();
  const result = await client.workflow.execute(greeting, {
    args: ['Tina'],
    taskQueue: 'translation-tasks',
    workflowId: 'workflow-' + randomUUID(),
  });
  console.log(`The greeting Workflow returned: ${result}`);
}

run().catch((err) => {
  console.error(err);
  process.exit(1);
});
```

npm run greeting



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

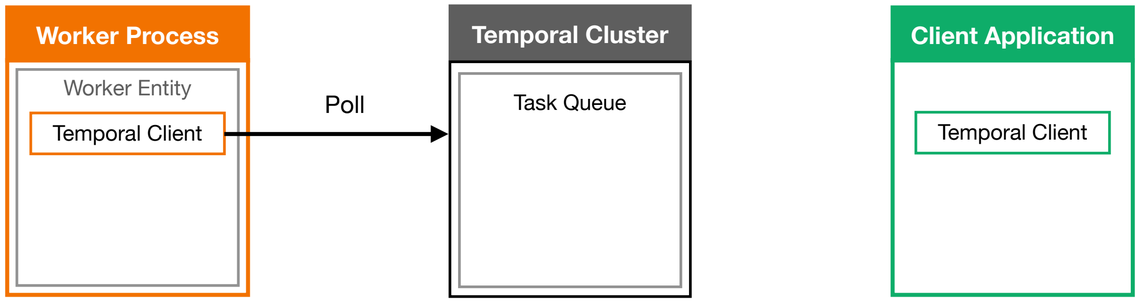
```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

Event History

WorkflowExecutionStarted



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

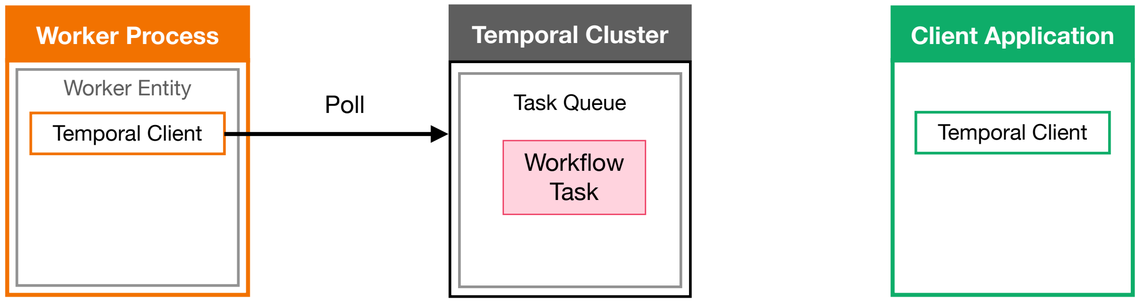
```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

Event History

WorkflowExecutionStarted
WorkflowTaskScheduled



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

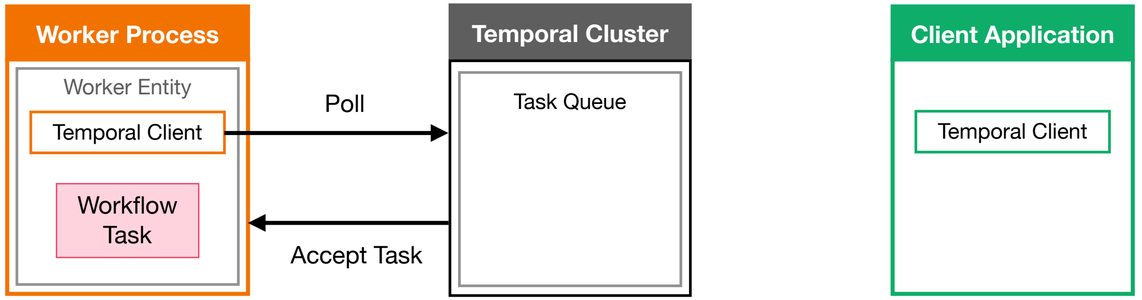
```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

Event History

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

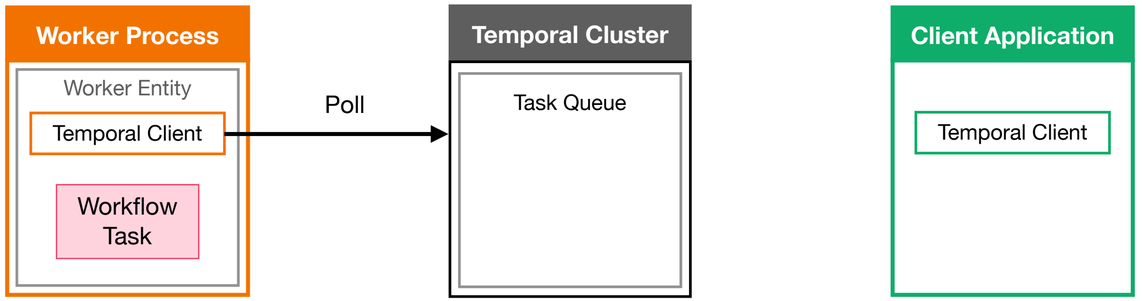
Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

```
import { proxyActivities } from '@temporalio/workflow';
import type * as activities from './activities';

const { getSpanishGreeting, getSpanishFarewell } = proxyActivities<
  typeof activities
>({
  startToCloseTimeout: '10 seconds',
});

export async function greeting(name: string): Promise<string> {
  const greeting = await getSpanishGreeting(name);
  const farewell = await getSpanishFarewell(name);
  const helloGoodbye = "\n" + greeting + "\n" + farewell;
  return helloGoodbye;
}
```

Event History

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

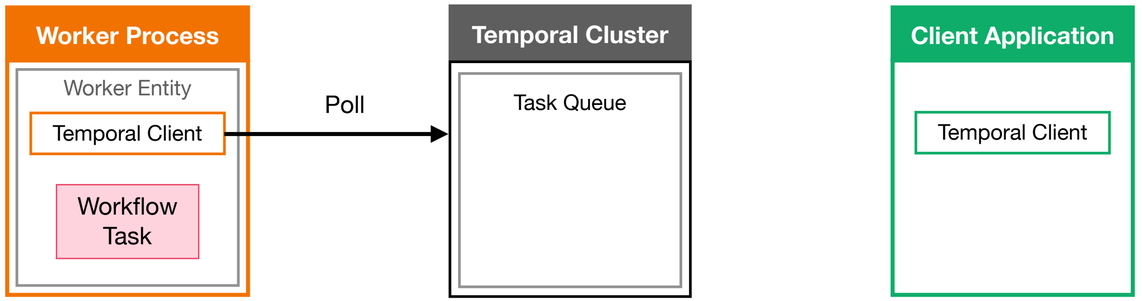
Workflow Definition

```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

```typescript
import { proxyActivities } from '@temporalio/workflow';
import type * as activities from './activities';

const { getSpanishGreeting, getSpanishFarewell } = proxyActivities<
  typeof activities
>({
  startToCloseTimeout: '10 seconds',
});

export async function greeting(name: string): Promise<string> {
  const greeting = await getSpanishGreeting(name);
  const farewell = await getSpanishFarewell(name);
  const helloGoodbye = "\n" + greeting + "\n" + farewell;
  return helloGoodbye;
}
```



Event History

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

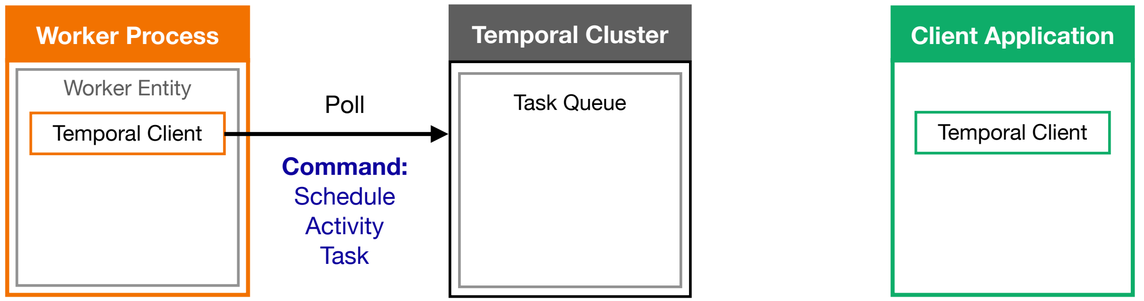
Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

```ts
import { proxyActivities } from '@temporalio/workflow';
import type * as activities from './activities';

const { getSpanishGreeting, getSpanishFarewell } = proxyActivities<
  typeof activities
>({
  startToCloseTimeout: '10 seconds',
});

export async function greeting(name: string): Promise<string> {
  const greeting = await getSpanishGreeting(name);
  const farewell = await getSpanishFarewell(name);
  const helloGoodbye = "\n" + greeting + "\n" + farewell;
  return helloGoodbye;
}
```

Event History

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

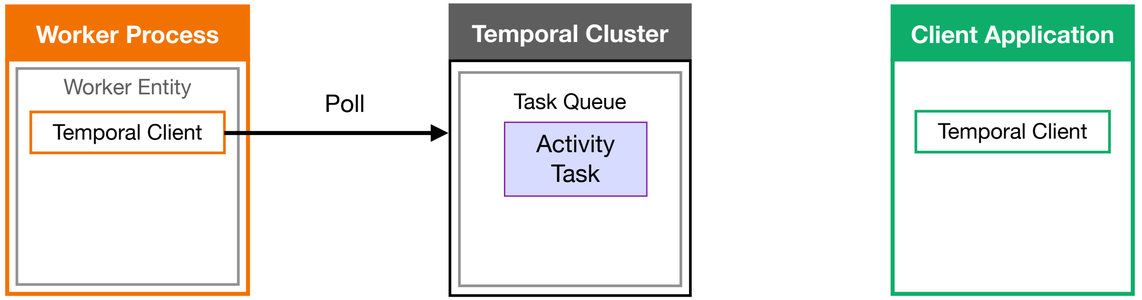
```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

Event History

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled (Greeting)



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

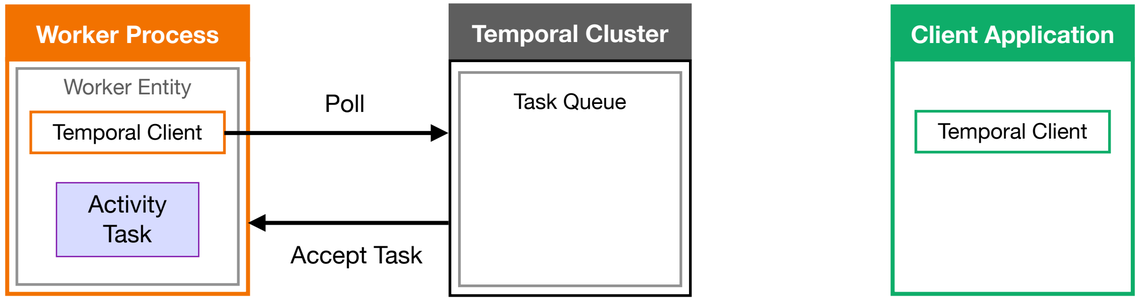
```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
  greeting, err := callService("get-spanish-greeting", name)
  return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
  greeting, err := callService("get-spanish-farewell", name)
  return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
  base := "http://localhost:9999/" + stem + "?name="
  url := fmt.Sprintf(base, url.QueryEscape(name))

  resp, err := http.Get(url)
  if err != nil {
    return "", err
  }
  defer resp.Body.Close()

  body, err := ioutil.ReadAll(resp.Body)
  if err != nil {
    return "", err
  }

  translation := string(body)
  status := resp.StatusCode
  if status >= 400 {
    message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
    return "", errors.New(message)
  }

  return translation, nil
}
```

Workflow Definition

```
package farewell

import (
  "time"

  "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
  options := workflow.ActivityOptions{
    StartToCloseTimeout: time.Second * 5,
  }
  ctx = workflow.WithActivityOptions(ctx, options)

  var spanishGreeting string
  err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
  if err != nil {
    return "", err
  }

  var spanishFarewell string
  err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
  if err != nil {
    return "", err
  }

  var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

  return helloGoodbye, nil
}
```

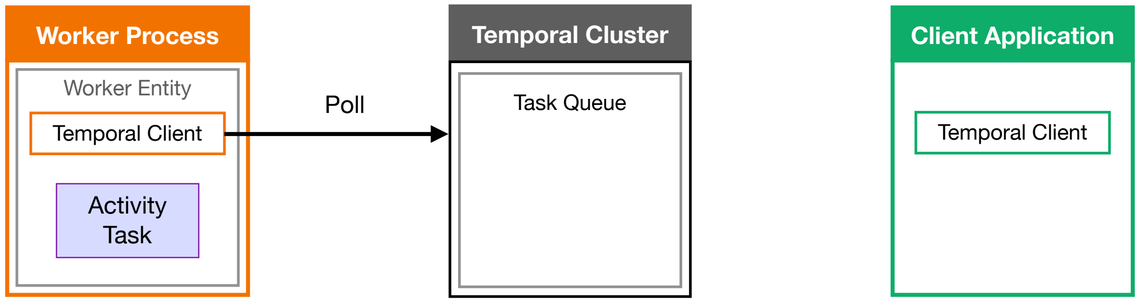
Worker Initialization

```
package main

import (
  "log"
  farewell "temporal101.exercises/farewell-workflow/solution"

  "go.temporal.io/sdk/client"
  "go.temporal.io/sdk/worker"
)

func main() {
  c, err := client.Dial(client.Options{})
  if err != nil {
    log.Fatalln("Unable to create client", err)
  }
  defer c.Close()

  w := worker.New(c, "greeting-tasks", worker.Options{})

  w.RegisterWorkflow(farewell.GreetSomeone)
  w.RegisterActivity(farewell.GreetInSpanish)
  w.RegisterActivity(farewell.FarewellInSpanish)

  err = w.Run(worker.InterruptCh())
  if err != nil {
    log.Fatalln("Unable to start worker", err)
  }
}
```

```
import axios from 'axios';

const url = 'http://localhost:9999';

export async function getSpanishGreeting(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-greeting?name=${name}`);

  return response.data;
}

export async function getSpanishFarewell(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-farewell?name=${name}`);

  return response.data;
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
  greeting, err := callService("get-spanish-greeting", name)
  return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
  greeting, err := callService("get-spanish-farewell", name)
  return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
  base := "http://localhost:9999/" + stem + "?name="
  url := fmt.Sprint(base, url.QueryEscape(name))

  resp, err := http.Get(url)
  if err != nil {
    return "", err
  }
  defer resp.Body.Close()

  body, err := ioutil.ReadAll(resp.Body)
  if err != nil {
    return "", err
  }

  translation := string(body)
  status := resp.StatusCode
  if status >= 400 {
    message := fmt.Sprint("HTTP Error %d: %s", status, translation)
    return "", errors.New(message)
  }

  return translation, nil
}
```

Workflow Definition

```
package farewell

import (
  "time"

  "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
  options := workflow.ActivityOptions{
    StartToCloseTimeout: time.Second * 5,
  }
  ctx = workflow.WithActivityOptions(ctx, options)

  var spanishGreeting string
  err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
  if err != nil {
    return "", err
  }

  var spanishFarewell string
  err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
  if err != nil {
    return "", err
  }

  var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

  return helloGoodbye, nil
}
```

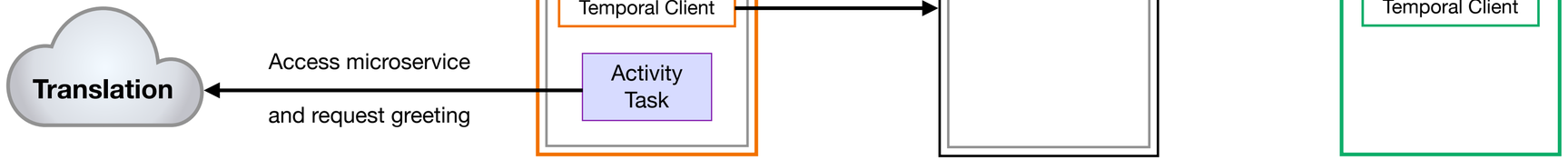
Worker Initialization

```
package main

import (
  "log"
  farewell "temporal101.exercises/farewell-workflow/solution"

  "go.temporal.io/sdk/client"
  "go.temporal.io/sdk/worker"
)

func main() {
  c, err := client.Dial(client.Options{})
  if err != nil {
    log.Fatalln("Unable to create client", err)
  }
  defer c.Close()

  w := worker.New(c, "greeting-tasks", worker.Options{})

  w.RegisterWorkflow(farewell.GreetSomeone)
  w.RegisterActivity(farewell.GreetInSpanish)
  w.RegisterActivity(farewell.FarewellInSpanish)

  err = w.Run(worker.InterruptCh())
  if err != nil {
    log.Fatalln("Unable to start worker", err)
  }
}
```

```
import axios from 'axios';

const url = 'http://localhost:9999';

export async function getSpanishGreeting(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-greeting?name=${name}`);

  return response.data;
}

export async function getSpanishFarewell(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-farewell?name=${name}`);

  return response.data;
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

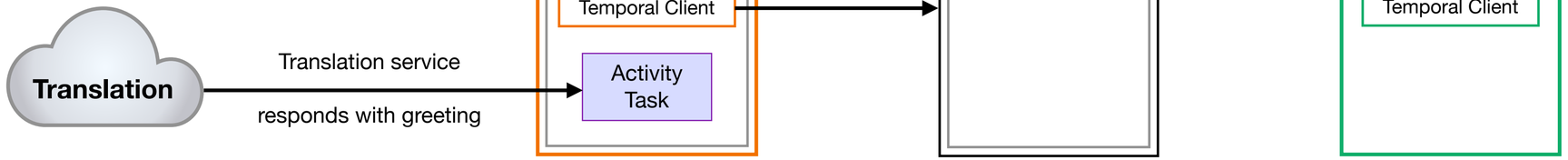
Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

```
import axios from 'axios';

const url = 'http://localhost:9999';

export async function getSpanishGreeting(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-greeting?name=${name}`);

  return response.data;
}

export async function getSpanishFarewell(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-farewell?name=${name}`);

  return response.data;
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="
    url := fmt.Sprint(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprint("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

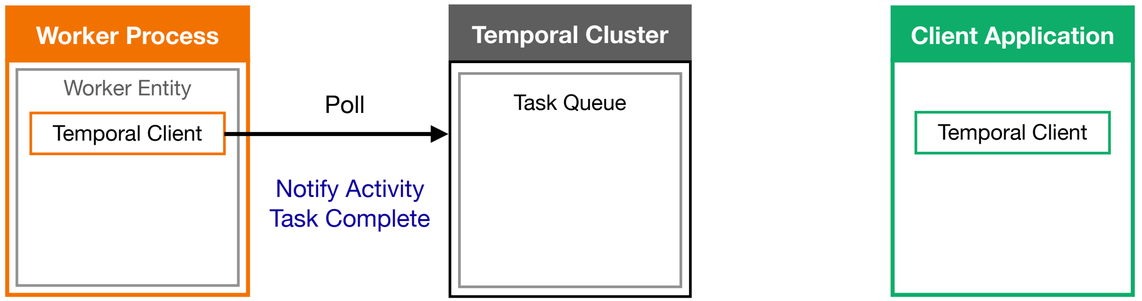
Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

```
import axios from 'axios';

const url = 'http://localhost:9999';

export async function getSpanishGreeting(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-greeting?name=${name}`);

  return response.data;
}

export async function getSpanishFarewell(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-farewell?name=${name}`);

  return response.data;
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

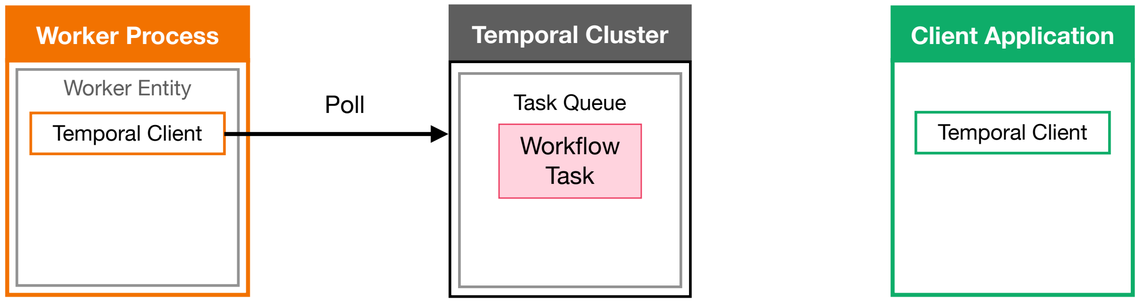
```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

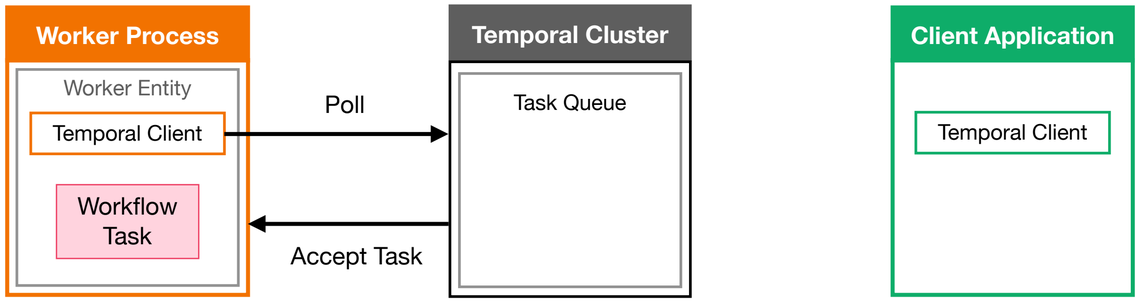
```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

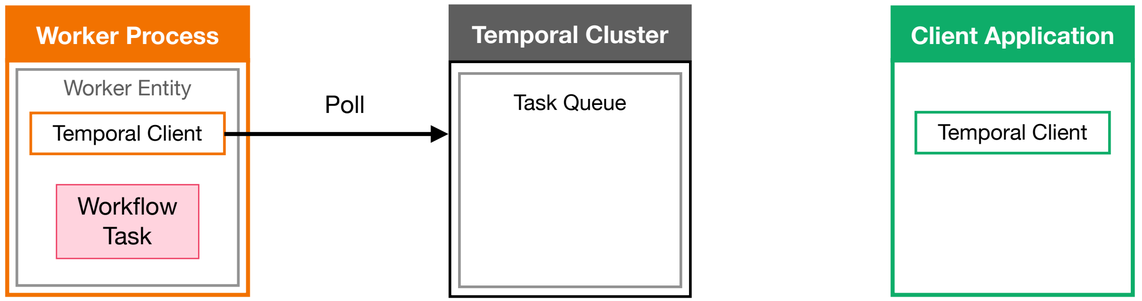
Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

```
import { proxyActivities } from '@temporalio/workflow';
import type * as activities from './activities';

const { getSpanishGreeting, getSpanishFarewell } = proxyActivities<
  typeof activities
>({
  startToCloseTimeout: '10 seconds',
});

export async function greeting(name: string): Promise<string> {
  const greeting = await getSpanishGreeting(name);
  const farewell = await getSpanishFarewell(name);
  const helloGoodbye = "\n" + greeting + "\n" + farewell;
  return helloGoodbye;
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

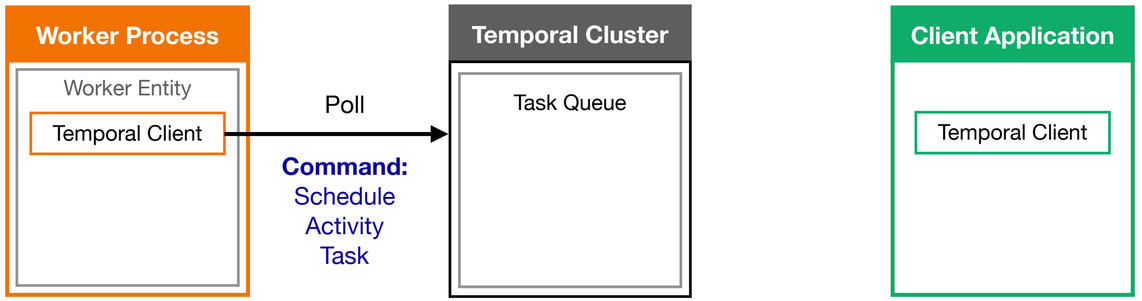
Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

```
import { proxyActivities } from '@temporalio/workflow';
import type * as activities from './activities';

const { getSpanishGreeting, getSpanishFarewell } = proxyActivities<
  typeof activities
>({
  startToCloseTimeout: '10 seconds',
});

export async function greeting(name: string): Promise<string> {
  const greeting = await getSpanishGreeting(name);
  const farewell = await getSpanishFarewell(name);
  const helloGoodbye = "\n" + greeting + "\n" + farewell;
  return helloGoodbye;
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

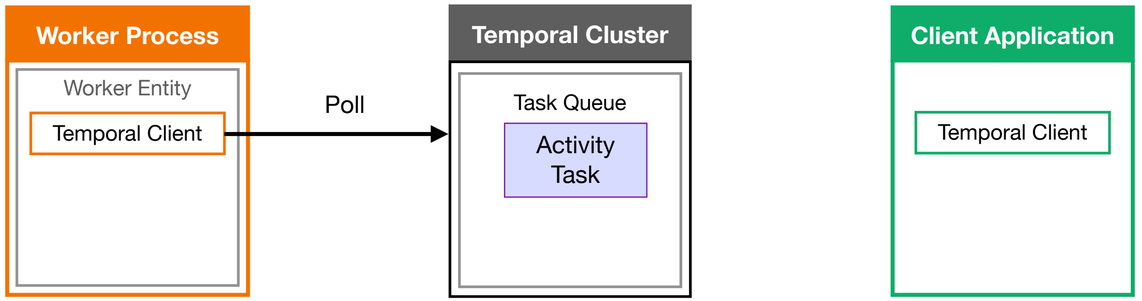
```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

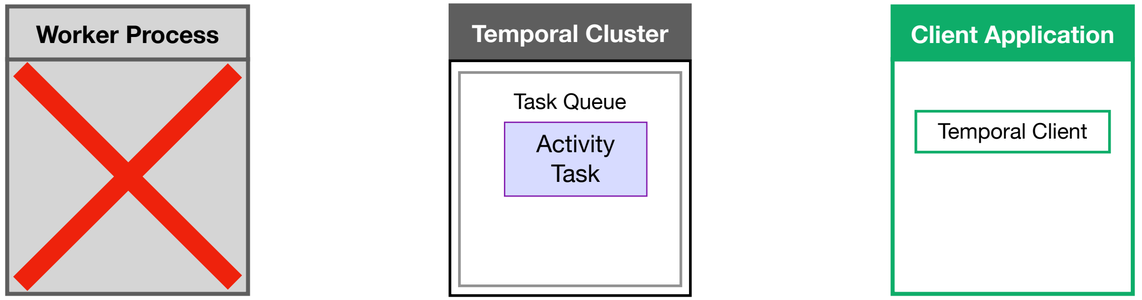
Workflow Definition

```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

What happens if the Worker crashes?



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)

Activity Definitions

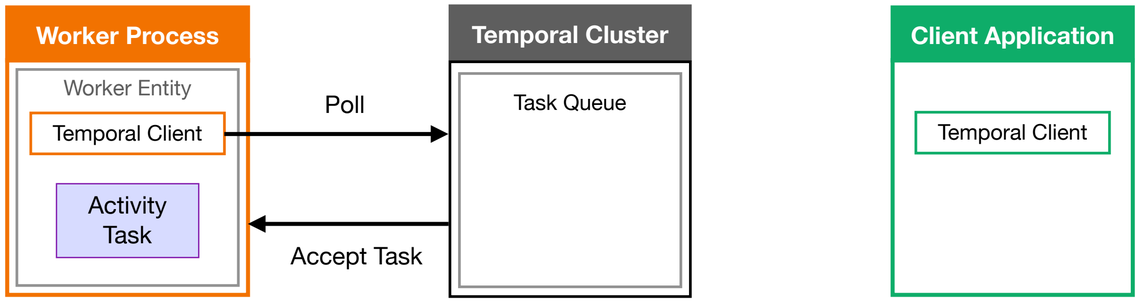
```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)

Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

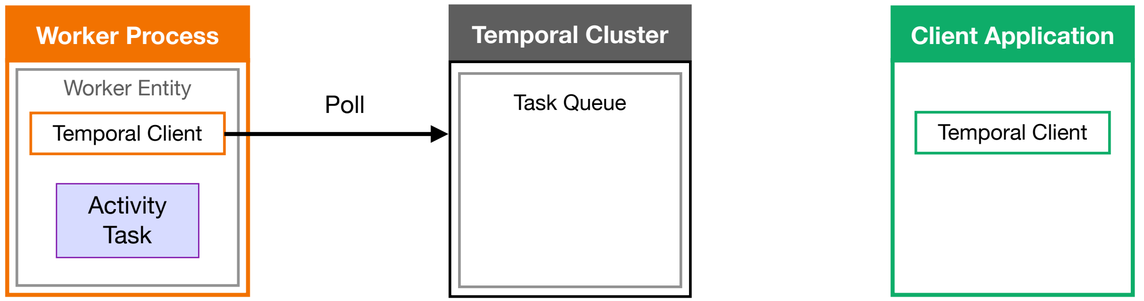
Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

```
import axios from 'axios';

const url = 'http://localhost:9999';

export async function getSpanishGreeting(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-greeting?name=${name}`);

  return response.data;
}

export async function getSpanishFarewell(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-farewell?name=${name}`);

  return response.data;
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
  greeting, err := callService("get-spanish-greeting", name)
  return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
  greeting, err := callService("get-spanish-farewell", name)
  return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
  base := "http://localhost:9999/" + stem + "?name="
  url := fmt.Sprint(base, url.QueryEscape(name))

  resp, err := http.Get(url)
  if err != nil {
    return "", err
  }
  defer resp.Body.Close()

  body, err := ioutil.ReadAll(resp.Body)
  if err != nil {
    return "", err
  }

  translation := string(body)
  status := resp.StatusCode
  if status >= 400 {
    message := fmt.Sprint("HTTP Error %d: %s", status, translation)
    return "", errors.New(message)
  }

  return translation, nil
}
```

Workflow Definition

```
package farewell

import (
  "time"

  "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
  options := workflow.ActivityOptions{
    StartToCloseTimeout: time.Second * 5,
  }
  ctx = workflow.WithActivityOptions(ctx, options)

  var spanishGreeting string
  err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
  if err != nil {
    return "", err
  }

  var spanishFarewell string
  err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
  if err != nil {
    return "", err
  }

  var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

  return helloGoodbye, nil
}
```

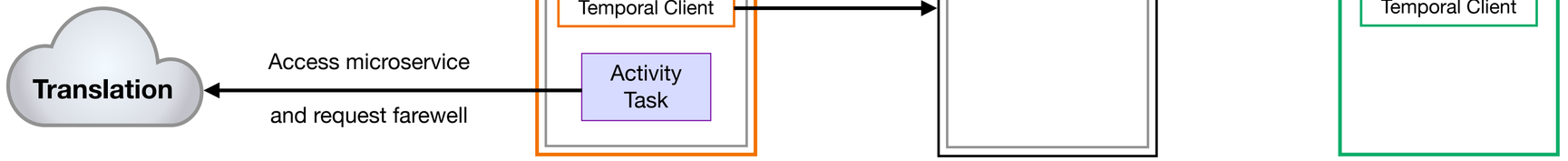
Worker Initialization

```
package main

import (
  "log"
  farewell "temporal101.exercises/farewell-workflow/solution"

  "go.temporal.io/sdk/client"
  "go.temporal.io/sdk/worker"
)

func main() {
  c, err := client.Dial(client.Options{})
  if err != nil {
    log.Fatalln("Unable to create client", err)
  }
  defer c.Close()

  w := worker.New(c, "greeting-tasks", worker.Options{})

  w.RegisterWorkflow(farewell.GreetSomeone)
  w.RegisterActivity(farewell.GreetInSpanish)
  w.RegisterActivity(farewell.FarewellInSpanish)

  err = w.Run(worker.InterruptCh())
  if err != nil {
    log.Fatalln("Unable to start worker", err)
  }
}
```

```
import axios from 'axios';

const url = 'http://localhost:9999';

export async function getSpanishGreeting(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-greeting?name=${name}`);

  return response.data;
}

export async function getSpanishFarewell(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-farewell?name=${name}`);

  return response.data;
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
  greeting, err := callService("get-spanish-greeting", name)
  return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
  greeting, err := callService("get-spanish-farewell", name)
  return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
  base := "http://localhost:9999/" + stem + "?name="
  url := fmt.Sprintf(base, url.QueryEscape(name))

  resp, err := http.Get(url)
  if err != nil {
    return "", err
  }
  defer resp.Body.Close()

  body, err := ioutil.ReadAll(resp.Body)
  if err != nil {
    return "", err
  }

  translation := string(body)
  status := resp.StatusCode
  if status >= 400 {
    message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
    return "", errors.New(message)
  }

  return translation, nil
}
```

Workflow Definition

```
package farewell

import (
  "time"

  "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
  options := workflow.ActivityOptions{
    StartToCloseTimeout: time.Second * 5,
  }
  ctx = workflow.WithActivityOptions(ctx, options)

  var spanishGreeting string
  err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
  if err != nil {
    return "", err
  }

  var spanishFarewell string
  err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
  if err != nil {
    return "", err
  }

  var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

  return helloGoodbye, nil
}
```

Worker Initialization

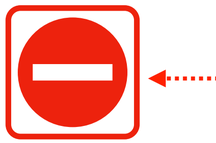
```
package main

import (
  "log"
  farewell "temporal101.exercises/farewell-workflow/solution"

  "go.temporal.io/sdk/client"
  "go.temporal.io/sdk/worker"
)

func main() {
  c, err := client.Dial(client.Options{})
  if err != nil {
    log.Fatalln("Unable to create client", err)
  }
  defer c.Close()

  w := worker.New(c, "greeting-tasks", worker.Options{})

  w.RegisterWorkflow(farewell.GreetSomeone)
  w.RegisterActivity(farewell.GreetInSpanish)
  w.RegisterActivity(farewell.FarewellInSpanish)

  err = w.Run(worker.InterruptCh())
  if err != nil {
    log.Fatalln("Unable to start worker", err)
  }
}
```

```
import axios from 'axios';

const url = 'http://localhost:9999';

export async function getSpanishGreeting(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-greeting?name=${name}`);

  return response.data;
}

export async function getSpanishFarewell(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-farewell?name=${name}`);

  return response.data;
}
```

Error

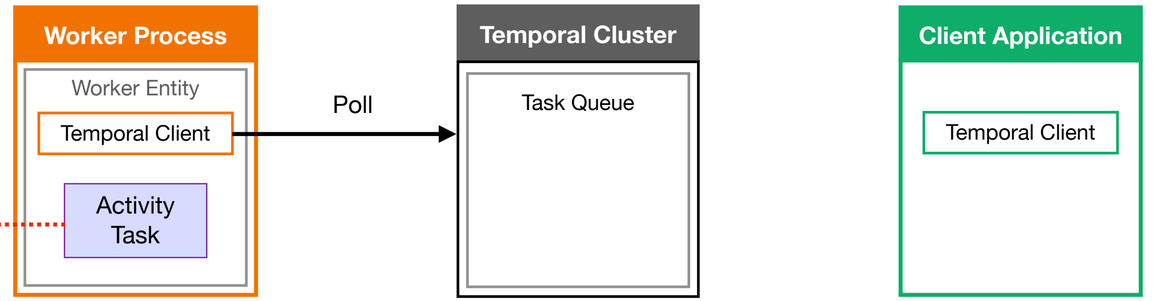
Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)



Service Unavailable

Execution fails due to service outage



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

Worker Initialization

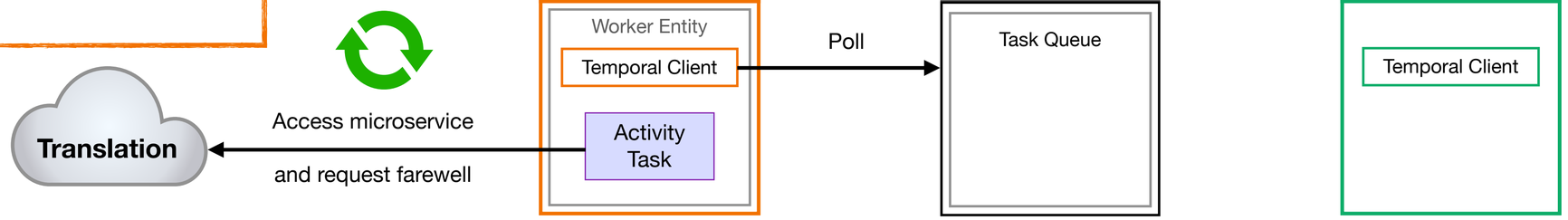
```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

```
import axios from 'axios';

const url = 'http://localhost:9999';

export async function getSpanishGreeting(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-greeting?name=${name}`);

  return response.data;
}

export async function getSpanishFarewell(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-farewell?name=${name}`);

  return response.data;
}
```

Activity is invoked again during retry

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)



Activity Definitions

```
package farewell   // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
   greeting, err := callService("get-spanish-greeting", name)
   return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
   greeting, err := callService("get-spanish-farewell", name)
   return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
   base := "http://localhost:9999/" + stem + "?name="
   url := fmt.Sprintf(base, url.QueryEscape(name))

   resp, err := http.Get(url)
   if err != nil {
      return "", err
   }
   defer resp.Body.Close()

   body, err := ioutil.ReadAll(resp.Body)
   if err != nil {
      return "", err
   }

   translation := string(body)
   status := resp.StatusCode
   if status >= 400 {
      message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
      return "", errors.New(message)
   }

   return translation, nil
}
```

Workflow Definition

```
package farewell

import (
   "time"

   "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   var spanishGreeting string
   err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
   if err != nil {
      return "", err
   }

   var spanishFarewell string
   err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
   if err != nil {
      return "", err
   }

   var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

   return helloGoodbye, nil
}
```

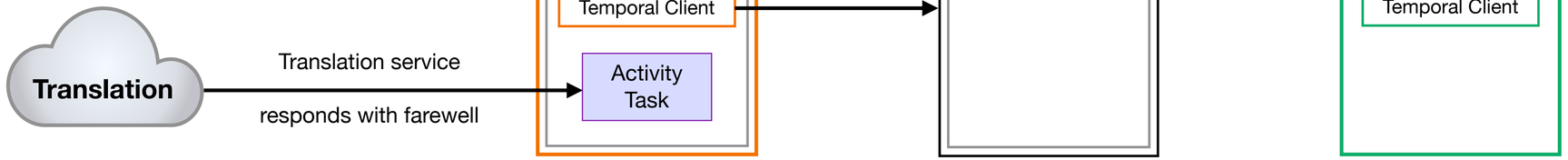
Worker Initialization

```
package main

import (
   "log"
   farewell "temporal101.exercises/farewell-workflow/solution"

   "go.temporal.io/sdk/client"
   "go.temporal.io/sdk/worker"
)

func main() {
   c, err := client.Dial(client.Options{})
   if err != nil {
      log.Fatalln("Unable to create client", err)
   }
   defer c.Close()

   w := worker.New(c, "greeting-tasks", worker.Options{})

   w.RegisterWorkflow(farewell.GreetSomeone)
   w.RegisterActivity(farewell.GreetInSpanish)
   w.RegisterActivity(farewell.FarewellInSpanish)

   err = w.Run(worker.InterruptCh())
   if err != nil {
      log.Fatalln("Unable to start worker", err)
   }
}
```

```
import axios from 'axios';

const url = 'http://localhost:9999';

export async function getSpanishGreeting(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-greeting?name=${name}`);

  return response.data;
}

export async function getSpanishFarewell(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-farewell?name=${name}`);

  return response.data;
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
  greeting, err := callService("get-spanish-greeting", name)
  return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
  greeting, err := callService("get-spanish-farewell", name)
  return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
  base := "http://localhost:9999/" + stem + "?name="
  url := fmt.Sprintf(base, url.QueryEscape(name))

  resp, err := http.Get(url)
  if err != nil {
    return "", err
  }
  defer resp.Body.Close()

  body, err := ioutil.ReadAll(resp.Body)
  if err != nil {
    return "", err
  }

  translation := string(body)
  status := resp.StatusCode
  if status >= 400 {
    message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
    return "", errors.New(message)
  }

  return translation, nil
}
```

Workflow Definition

```
package farewell

import (
  "time"

  "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
  options := workflow.ActivityOptions{
    StartToCloseTimeout: time.Second * 5,
  }
  ctx = workflow.WithActivityOptions(ctx, options)

  var spanishGreeting string
  err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
  if err != nil {
    return "", err
  }

  var spanishFarewell string
  err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
  if err != nil {
    return "", err
  }

  var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

  return helloGoodbye, nil
}
```

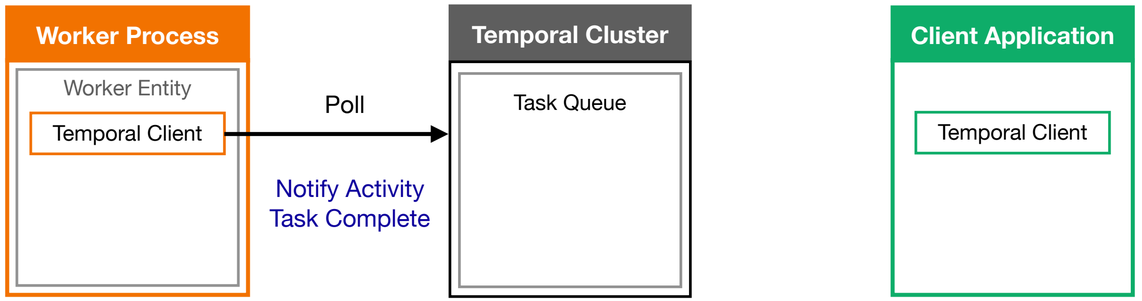
Worker Initialization

```
package main

import (
  "log"
  farewell "temporal101.exercises/farewell-workflow/solution"

  "go.temporal.io/sdk/client"
  "go.temporal.io/sdk/worker"
)

func main() {
  c, err := client.Dial(client.Options{})
  if err != nil {
    log.Fatalln("Unable to create client", err)
  }
  defer c.Close()

  w := worker.New(c, "greeting-tasks", worker.Options{})

  w.RegisterWorkflow(farewell.GreetSomeone)
  w.RegisterActivity(farewell.GreetInSpanish)
  w.RegisterActivity(farewell.FarewellInSpanish)

  err = w.Run(worker.InterruptCh())
  if err != nil {
    log.Fatalln("Unable to start worker", err)
  }
}
```

```
import axios from 'axios';

const url = 'http://localhost:9999';

export async function getSpanishGreeting(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-greeting?name=${name}`);

  return response.data;
}

export async function getSpanishFarewell(name: string): Promise<string> {
  const response = await axios.get(`${url}/get-spanish-farewell?name=${name}`);

  return response.data;
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)
ActivityTaskCompleted	(Farewell)



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

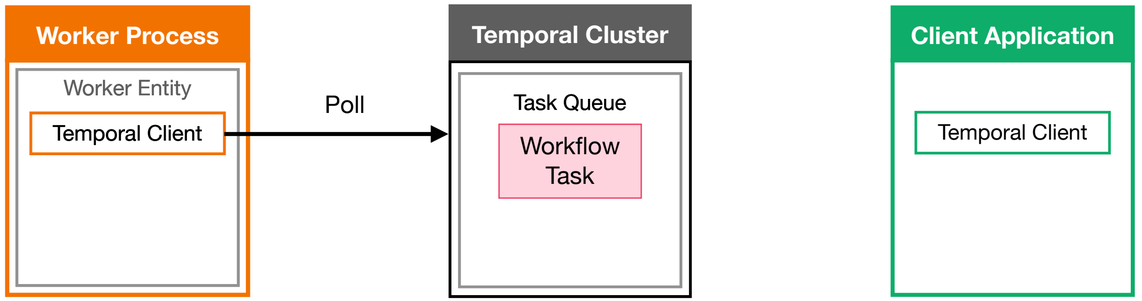
```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)
ActivityTaskCompleted	(Farewell)
WorkflowTaskScheduled	



Activity Definitions

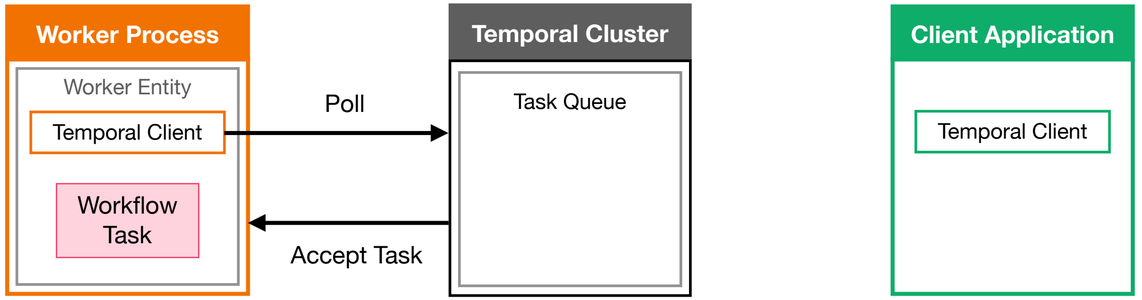
```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)
ActivityTaskCompleted	(Farewell)
WorkflowTaskScheduled	
WorkflowTaskStarted	

Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

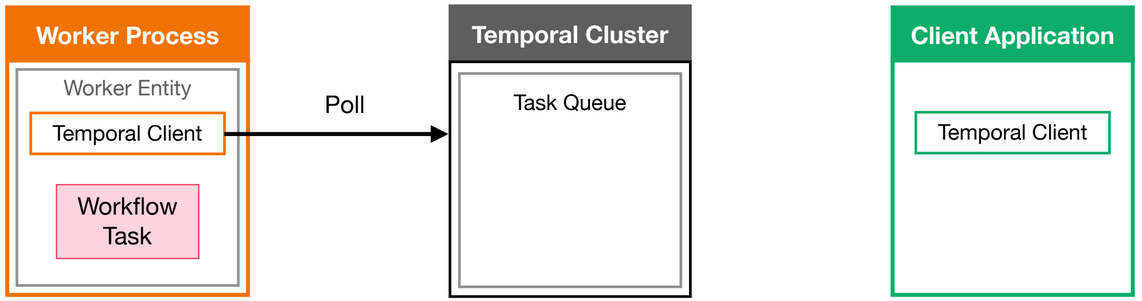
Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

```ts
import { proxyActivities } from '@temporalio/workflow';
import type * as activities from './activities';

const { getSpanishGreeting, getSpanishFarewell } = proxyActivities<
  typeof activities
>({
  startToCloseTimeout: '10 seconds',
});

export async function greeting(name: string): Promise<string> {
  const greeting = await getSpanishGreeting(name);
  const farewell = await getSpanishFarewell(name);
  const helloGoodbye = "\n" + greeting + "\n" + farewell;
  return helloGoodbye;
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)
ActivityTaskCompleted	(Farewell)
WorkflowTaskScheduled	
WorkflowTaskStarted	



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

Workflow Definition

```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

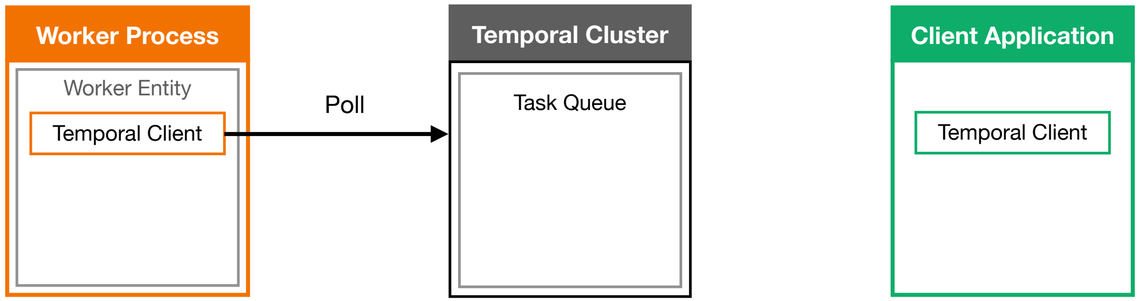
Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

```ts
import { proxyActivities } from '@temporalio/workflow';
import type * as activities from './activities';

const { getSpanishGreeting, getSpanishFarewell } = proxyActivities<
  typeof activities
>({
  startToCloseTimeout: '10 seconds',
});

export async function greeting(name: string): Promise<string> {
  const greeting = await getSpanishGreeting(name);
  const farewell = await getSpanishFarewell(name);
  const helloGoodbye = "\n" + greeting + "\n" + farewell;
  return helloGoodbye;
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)
ActivityTaskCompleted	(Farewell)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	



Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
  greeting, err := callService("get-spanish-greeting", name)
  return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
  greeting, err := callService("get-spanish-farewell", name)
  return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
  base := "http://localhost:9999/" + stem + "?name="+a"
  url := fmt.Sprintf(base, url.QueryEscape(name))

  resp, err := http.Get(url)
  if err != nil {
      return "", err
  }
  defer resp.Body.Close()

  body, err := ioutil.ReadAll(resp.Body)
  if err != nil {
      return "", err
  }

  translation := string(body)
  status := resp.StatusCode
  if status >= 400 {
      message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
      return "", errors.New(message)
  }

  return translation, nil
}
```

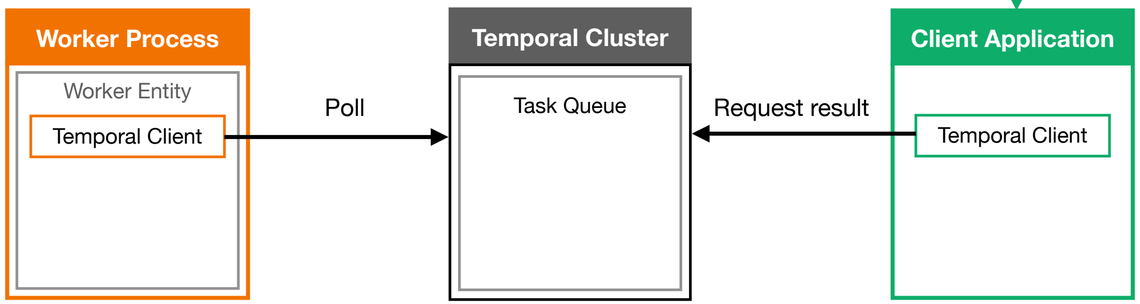
Workflow Definition

```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "Hi" + spanishGreeting + "Hi" + spanishFarewell

    return helloGoodbye, nil
}
```

Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

```
import { Client } from '@temporalio/client';
import { randomUUID } from 'node:crypto';
import { greeting } from '../workflows';

async function run() {
  const client = new Client();
  const result = await client.workflow.execute(greeting, {
    args: ['Tina'],
    taskQueue: 'translation-tasks',
    workflowId: 'workflow-' + randomUUID(),
  });
  console.log(`The greeting Workflow returned: ${result}`);
}

run().catch((err) => {
  console.error(err);
  process.exit(1);
});
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)
ActivityTaskCompleted	(Farewell)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
WorkflowExecutionCompleted	

The End

Activity Definitions

```
package farewell  // import statements omitted for brevity

func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", name)
    return greeting, err
}

func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-farewell", name)
    return greeting, err
}

// utility function for making calls to the microservices
func callService(stem string, name string) (string, error) {
    base := "http://localhost:9999/" + stem + "?name="+a"
    url := fmt.Sprintf(base, url.QueryEscape(name))

    resp, err := http.Get(url)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return "", err
    }

    translation := string(body)
    status := resp.StatusCode
    if status >= 400 {
        message := fmt.Sprintf("HTTP Error %d: %s", status, translation)
        return "", errors.New(message)
    }

    return translation, nil
}
```

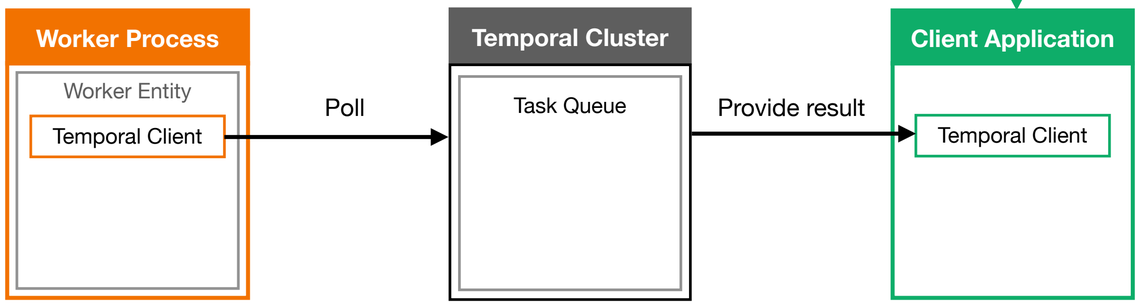
Workflow Definition

```
package farewell

import (
    "time"

    "go.temporal.io/sdk/workflow"
)

func GreetSomeone(ctx workflow.Context, name string) (string, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var spanishGreeting string
    err := workflow.ExecuteActivity(ctx, GreetInSpanish, name).Get(ctx, &spanishGreeting)
    if err != nil {
        return "", err
    }

    var spanishFarewell string
    err = workflow.ExecuteActivity(ctx, FarewellInSpanish, name).Get(ctx, &spanishFarewell)
    if err != nil {
        return "", err
    }

    var helloGoodbye = "\n" + spanishGreeting + "\n" + spanishFarewell

    return helloGoodbye, nil
}
```

Worker Initialization

```
package main

import (
    "log"
    farewell "temporal101.exercises/farewell-workflow/solution"

    "go.temporal.io/sdk/client"
    "go.temporal.io/sdk/worker"
)

func main() {
    c, err := client.Dial(client.Options{})
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    w := worker.New(c, "greeting-tasks", worker.Options{})

    w.RegisterWorkflow(farewell.GreetSomeone)
    w.RegisterActivity(farewell.GreetInSpanish)
    w.RegisterActivity(farewell.FarewellInSpanish)

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start worker", err)
    }
}
```

```
import { Client } from '@temporalio/client';
import { randomUUID } from 'node:crypto';
import { greeting } from '../workflows';

async function run() {
  const client = new Client();
  const result = await client.workflow.execute(greeting, {
    args: ['Tina'],
    taskQueue: 'translation-tasks',
    workflowId: 'workflow-' + randomUUID(),
  });
  console.log(`The greeting Workflow returned: ${result}`);
}

run().catch((err) => {
  console.error(err);
  process.exit(1);
});
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)
ActivityTaskCompleted	(Farewell)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
WorkflowExecutionCompleted	

Temporal 101

Conclusion (1)

- **Temporal guarantees the durable execution of your applications**
  - In Temporal, Workflows are defined through code (using a Temporal SDK)
- **Temporal Clusters orchestrate code execution**
  - Workers are responsible for actually executing the code
- **The Temporal Cluster maintains dynamically-created task queues**
  - Workers continuously poll a task queue and accept tasks if they have spare capacity
  - You can increase application scalability by adding more Workers
  - You must restart Workers after deploying a code change

Conclusion (2)

- **There are multiple ways of deploying a self-hosted Temporal cluster**
  - Temporal Cloud is an alternative to hosting your own cluster
- **Namespaces are used for isolation within a cluster**
  - The name is often chosen to indicate a specific team, department, or other category
- **In the TS SDK, a Temporal Workflow is defined through a function**
  - Activities are also defined through functions

Conclusion (3)

- **Activities encapsulate unreliable or non-deterministic code**
  - They are automatically retried upon failure
  - You can change this behavior with a custom Retry Policy
- **The Web UI is a powerful tool for gaining insight into your application**
  - It displays current and recent Workflow Executions
  - The Web UI shows inputs, outputs, and event history

Exercise #4: Finale Workflow

- **During this exercise, you will**
  - Observe that a Workflow and its Activities can be implemented in different languages
    - This example provides a Java Activity and a Go Workflow for you to run
- **Refer to the README.md file in the exercise environment for details**
  - The code is below the **exercises/finale-workflow** directory

Thank You