Beginner to Builder Bootcamp

Replay 2026 - San Francisco

20260503a

Beginner to Builder Bootcamp - Go

During this workshop, you will

- Learn the basic architecture of the Temporal platform
- Develop and execute Workflows and Activities using the Go SDK
- Use the Web UI to gain insight into current and previous executions
- Define, send, and handle Workflow Signals and Queries
- Understand how Temporal handles crashes, timeouts, and failures
- Customize a Retry Policy for Activity Execution
- Observe how Activity Heartbeating detects failure in a long-running Activity
- Apply the Saga pattern to recover from a failure in a Workflow Execution

Logistics

- Schedule
- Asking questions
- Getting help with exercises
- Course conventions ("workflow" vs. "Workflow")
- Introductions

About the Instructor

Tom Wheeler

St. Louis, Missouri, USA | Information Technology and Services

Current: **Principal Developer Advocate** at Temporal Technologies Inc.

Past: Principal Curriculum Developer at Cloudera

Principal Software Engineer at Object Computing Inc.

Software Engineer at WebMD

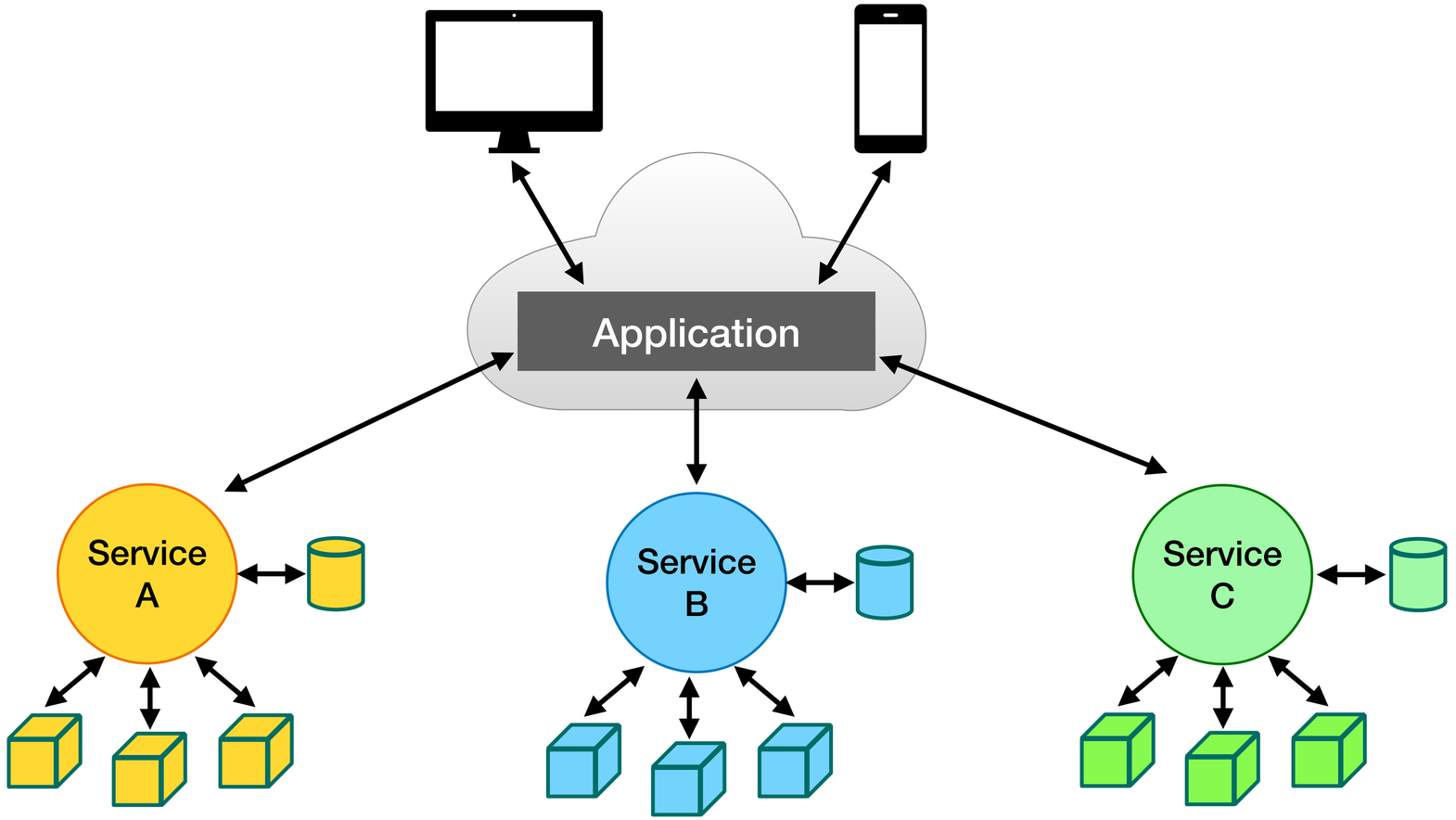
Senior Programmer / Analyst at A.G. Edwards and Sons, Inc.

Beginner to Builder Bootcamp - Go

What Is Temporal?

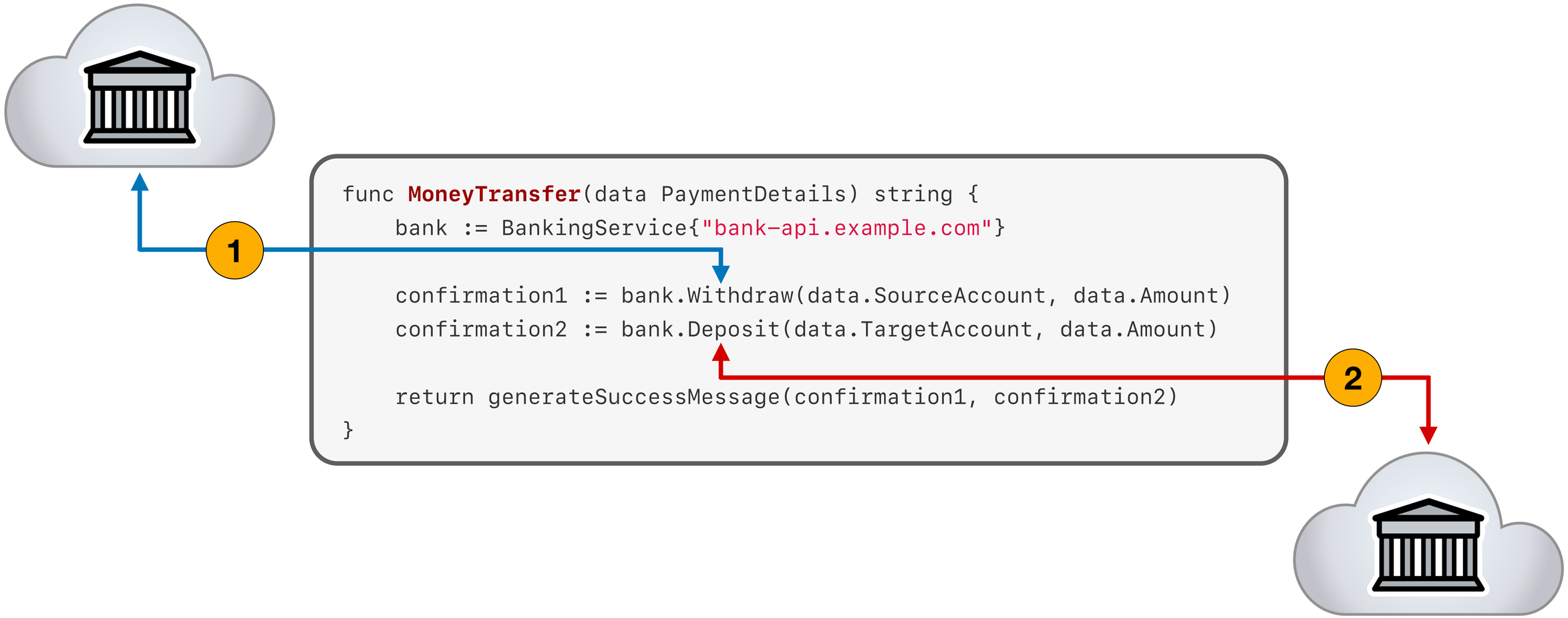
Let’s Begin with “Why Temporal Exists”

Modern Applications Are Distributed Systems



More potential for scalability, but more opportunities for failure

This Is a Distributed System




```
func MoneyTransfer(data PaymentDetails) string {
    bank := BankingService{"bank-api.example.com"}

    confirmation1 := bank.Withdraw(data.SourceAccount, data.Amount)
    confirmation2 := bank.Deposit(data.TargetAccount, data.Amount)

    return generateSuccessMessage(confirmation1, confirmation2)
}
```

There is a Big Problem Here

What if it crashes between steps 1 and 2?

```
func MoneyTransfer(data PaymentDetails) string {
    bank := BankingService{"bank-api.example.com"}

    confirmation1 := bank.Withdraw(data.SourceAccount, data.Amount)
    confirmation2 := bank.Deposit(data.TargetAccount, data.Amount)

    return generateSuccessMessage(confirmation1, confirmation2)
}
```

1

2

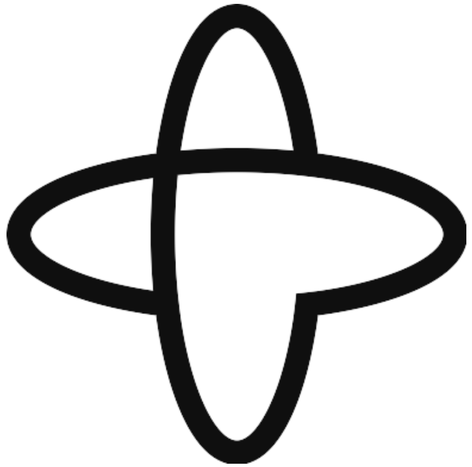
Correctness requires exactly-once execution

Durable Execution

- **Crashes are inevitable, but they don't have to be disruptive**
  - Durable Execution is *crash-proof* execution
  - It virtualizes the execution of your application, allowing it to span a series of processes
  - These processes can even span multiple machines over time, thus overcoming hardware failures
- **This simple concept has profound implications**
  - Significantly improves application reliability
  - Lets your code focus on the *goal*, not the *potential problems*
  - Accelerates development cycles and reduces maintenance effort
  - It will fundamentally change how you design software

What Is Temporal?

- **Temporal is an open source Durable Execution platform**
  - Automatically maintains application state
  - Offers built-in support for managing retries and timeouts
  - Enables applications to recover from crashes and failures—as if they *never even happened*
  - Improves developer productivity by making applications easier to develop, scale, and support



Architectural Overview

Temporal Application Overview: Workflows

- **Workflows are the core abstraction in a Temporal application**
  - They define the sequence of steps for your business logic
  - They have the benefit of Durable Execution
    - If a crash occurs, the state is automatically reconstructed and execution will continue
  - Temporal requires that Workflows are *deterministic*

Temporal Workflows Are Code

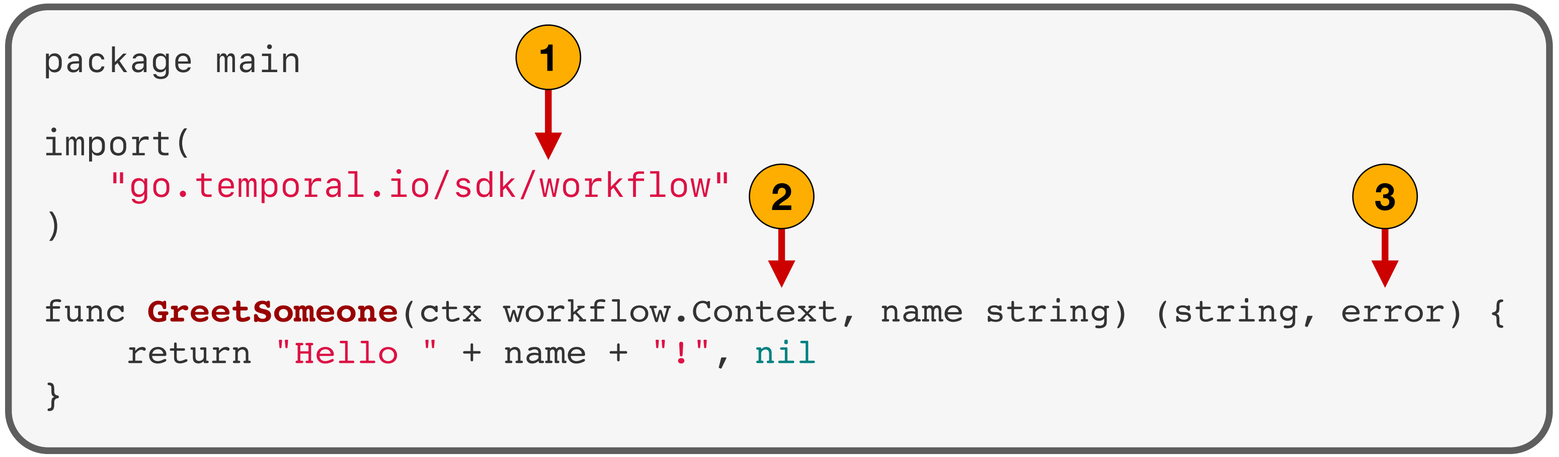
- **Temporal Workflows are defined in a standard programming language**
  - A *Temporal SDK* provides support for using Temporal in a specific language
  - Temporal currently offers seven SDKs…with an eighth (Rust) currently in development

Workflow Definition

- **With Temporal's Go SDK, you create a Workflow by writing a Go function**
  - The code for this function is known as a *Workflow Definition*
  - Each Workflow has a name, known as its *Workflow Type*
    - In the Go SDK, the Workflow Type is the name of the function (by default)

Writing a Workflow Definition

- **With Temporal's Go SDK, you define a Workflow by writing a Go function**
  - The code for this function is known as a *Workflow Definition*
  - Each Workflow has a name, known as its *Workflow Type*
    - In the Go SDK, the Workflow Type is the name of the function (by default)



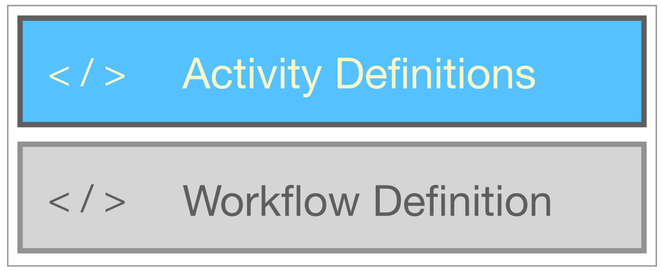
Temporal does not impose a naming convention on the function

Input Parameters and Return Values

- **The Temporal Service stores the history of your Workflow Executions**
  - Allows you to view input / output of running and completed Workflows
  - Workers uses this data to reconstruct the state in the event of a crash
- **Input parameters and return values must be serializable**
  - Allowed: Null values, binary data, and anything serializable via JSON or Protocol Buffers
  - Prohibited: Channels, functions, and unsafe pointers
- **Avoid passing in or returning large amounts of data from your Workflow**
  - This negatively affects performance and may result in the execution being terminated

Temporal Application Overview: Activities

- **Activities encapsulate unreliable or non-deterministic code**
  - They are automatically retried upon failure
  - They are referenced in the Workflow and invoked as the Workflow executes
    - In the earlier example, `MoneyTransfer` is the Workflow and `Withdraw` and `Deposit` are Activities
  - In the Go SDK, Activities are also defined through functions
  - Like Workflows, Activities can accept input and return results



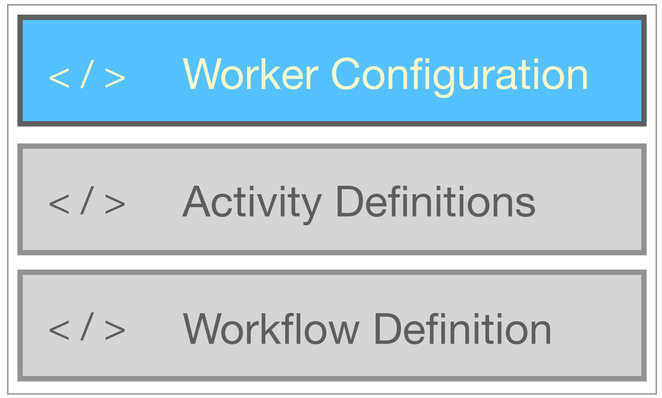
Executing Activities

```go
package app

import (
  "go.temporal.io/sdk/workflow"
  "time"
)

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
  }

  return spanishGreeting, nil
}
```

Excerpt from a Workflow Definition

Temporal Application Overview: Workers

- **Workers are responsible for executing Workflow and Activity Definitions**
- **The Worker implementation is provided by the Temporal SDK**
  - You must configure and start one or more Workers
  - It’s common to run multiple Workers: This increases the application’s availability and throughput



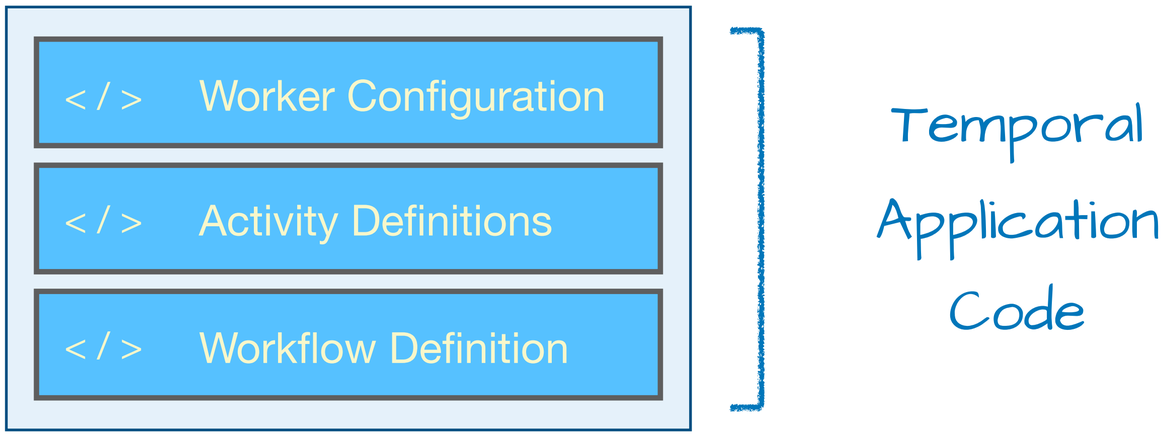
Configuring the Worker

- **How to configure a Worker**

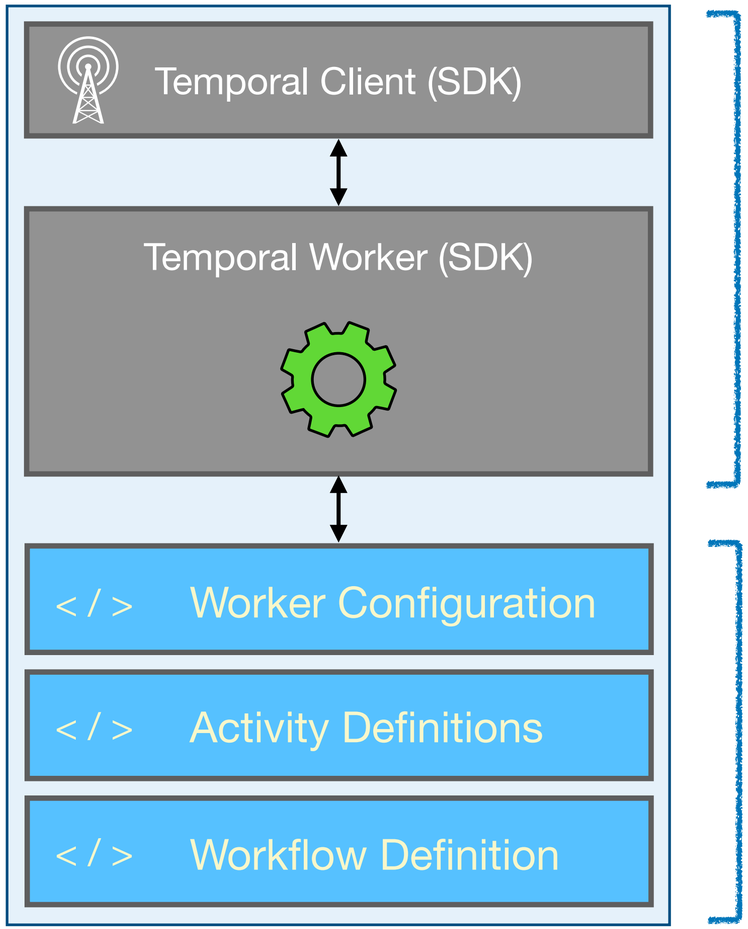
  1. Create a Temporal Client, which the Worker uses for communication
  2. Specify the name of a Task Queue for this Worker to poll
  3. Register the function(s) that this Worker will run
  4. Begin polling the Task Queue so it can find work to perform

```
import (
    "app"
    "log"
    "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(app.GreetSomeone)
    w.RegisterActivity(app.GetSpanishTranslation)

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

This Is the Code that You Develop



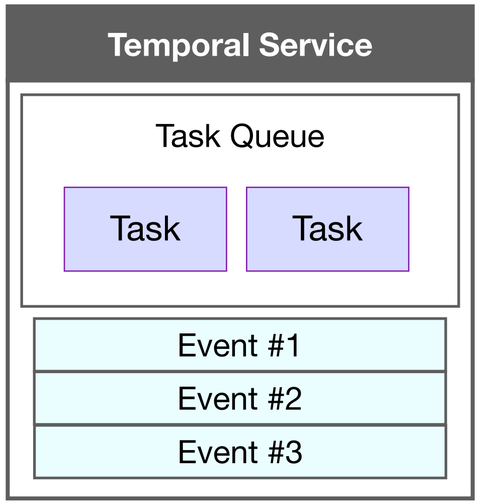
This Is a *Complete* Temporal Application



Provided by SDK

You develop

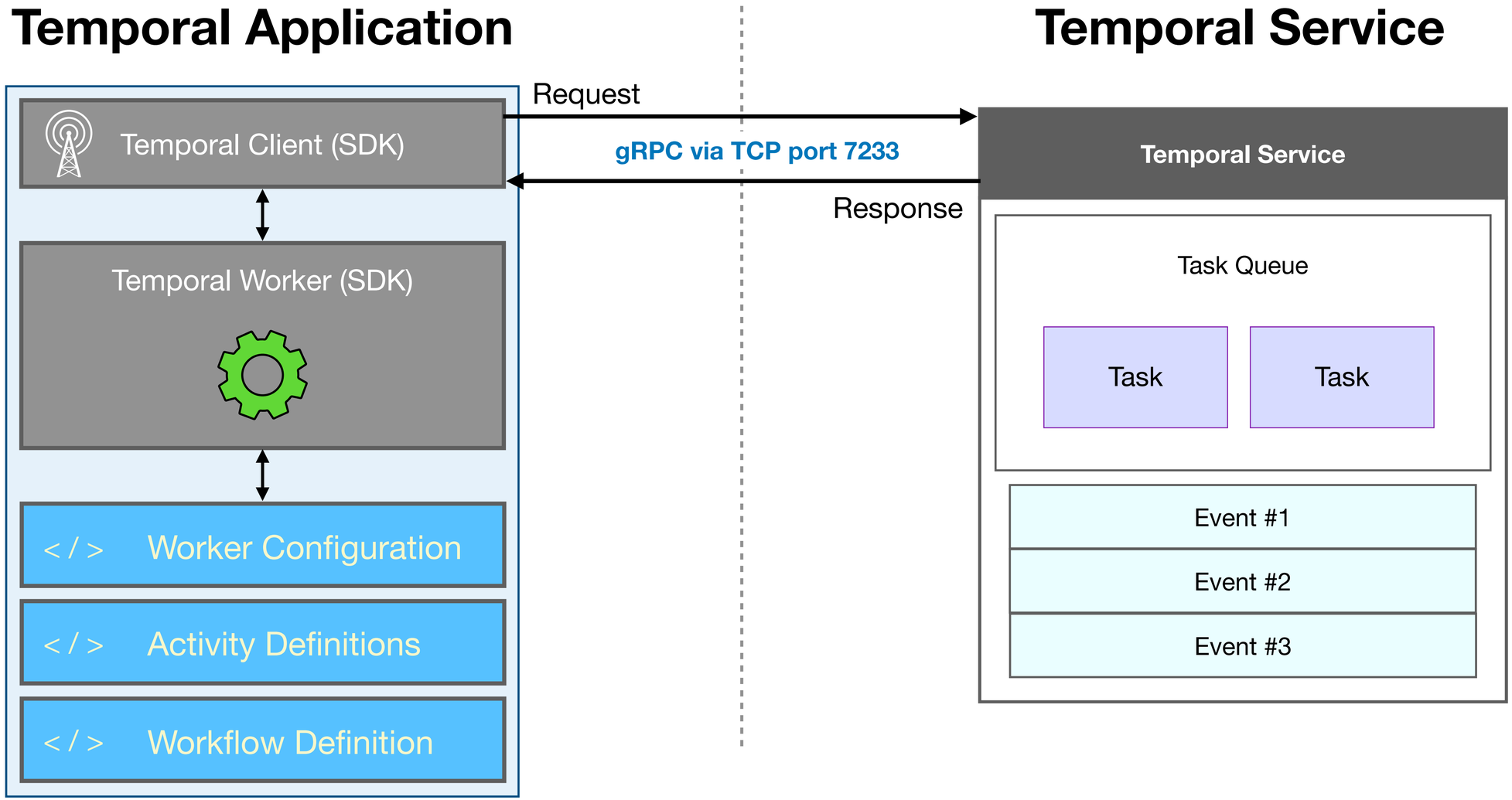
The Temporal Service



Manages tasks and history

- A backend system that *manages* Workflow Executions (it does not execute your code!)
- Creates tasks that instruct Workers on when and how to run Workflow and Activity code
- Maintains Event Histories that detail progress made by a given Workflow Execution

Applications Are External to the Temporal Service



Putting It All Together

- **Workflows are stateful and resilient, thanks to Durable Execution**
  - They’re used to orchestrate the business logic, but must behave deterministically
- **Activities are functions that encapsulate non-deterministic operations**
  - Use them to interact with the outside world (e.g., to call a service or query a database)
  - Activities are automatically retried if they fail
- **Temporal Service manages tasks and history of each Workflow Execution**
  - Workers poll for tasks, execute code, and report the result back to the Temporal Service
  - The Worker and Temporal Service are separate, typically deployed to different machines

Tools

Temporal Command-Line Interface

- **`temporal` is a CLI for interacting with the Temporal Service**
  - You can use it to start, terminate, and view details of Workflow Executions
  - You can also use it to run a local Temporal Service for development
  - Append `--help` to any command or subcommand to see usage info
  - See installation instructions in the Temporal documentation

```
$ temporal --help
The Temporal CLI manages, monitors, and debugs
Temporal apps. It lets you run a local Temporal
Service, start Workflow Executions, pass messages
to running Workflows, inspect state, and more.

* Start a local development service:
      temporal server start-dev
* View help: pass --help to any command:
      temporal activity complete --help

Usage:
  temporal [command]

Available Commands:
  activity    Complete or fail an Activity
  batch       Manage running batch jobs
```

Temporal Service for Development

- **The `temporal` CLI provides a Temporal Service for local development**
  - Install it (using Homebrew on a Mac)

```
$ brew install temporal
```

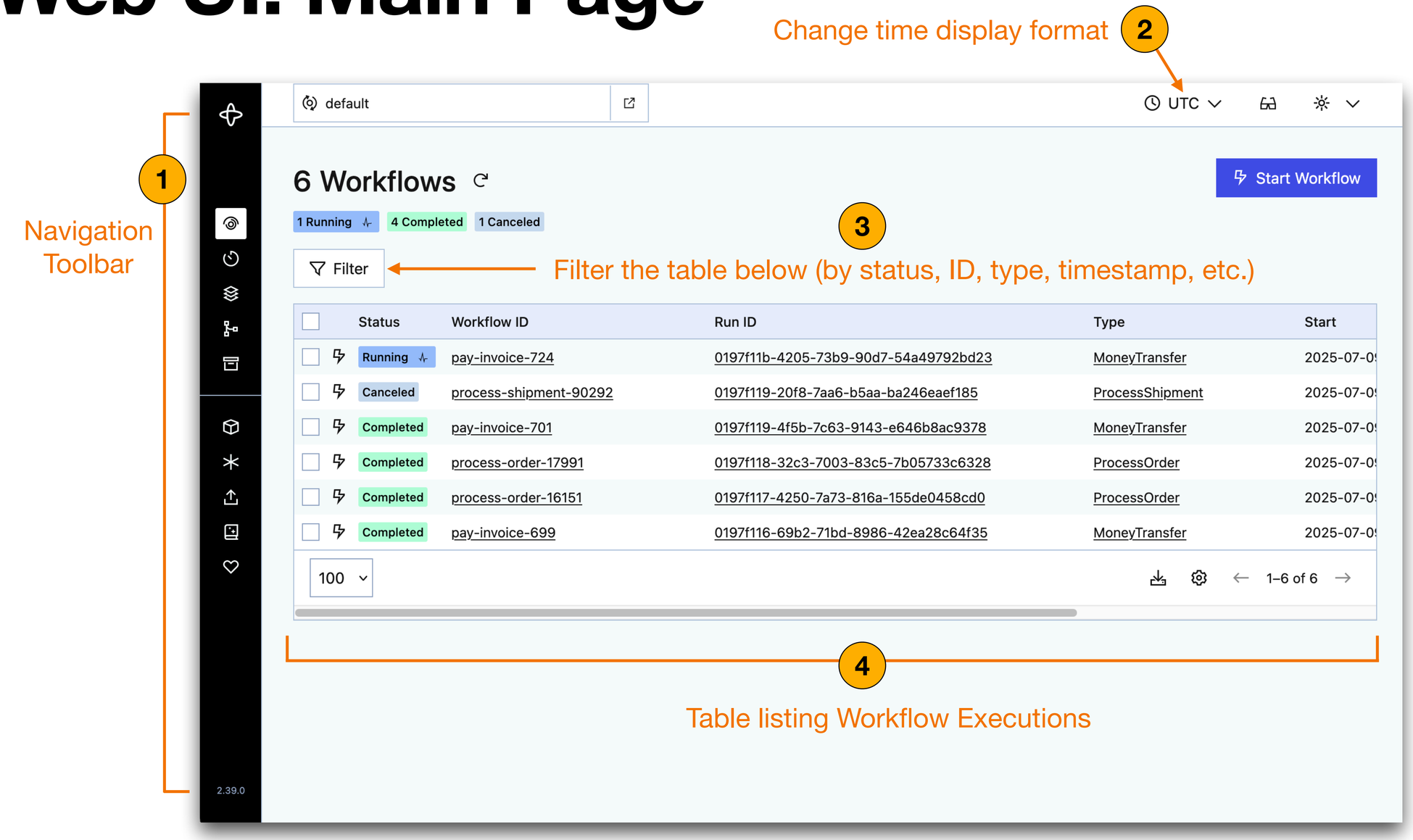
  - Start the Temporal Service (using default settings)

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

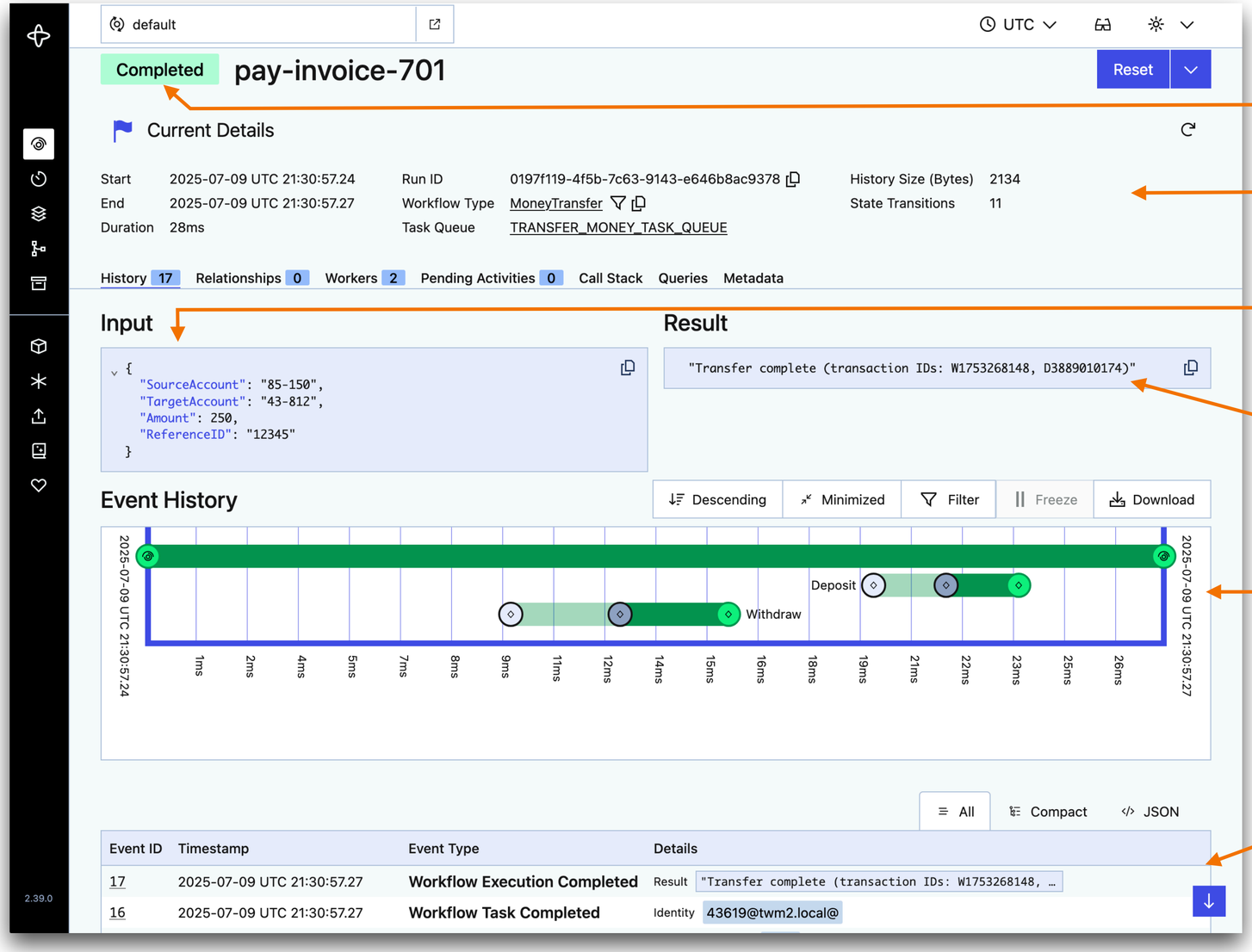
The Temporal Web UI

- **Temporal's Web UI is a powerful tool for Workflow Execution visibility**
  - Let's you explore the details of current and past execution
  - Helps you to identify problems, right down to the line of code that caused them
- **The port number used to access it may vary based on how it's deployed**
  - The `temporal server start-dev` command runs it on port 8233 by default
    - You would therefore access it with the URL `http://localhost:8233/`
    - You can override the port number with a command-line option (example: `--ui-port 8080`)

Web UI: Main Page



Web UI: Detail Page



1. Status
2. Workflow Execution details
3. Input data passed to Workflow Execution
4. Output data returned by Workflow Executio
5. Timeline display
6. Event History table

Executing a Basic Workflow

Executing a Workflow from the Command Line

- **One way to start a Workflow is with `temporal 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 unmarshalled and passed as Workflow function parameter

```
$ temporal workflow start \
    --type GreetSomeone \
    --task-queue greeting-tasks \
    --workflow-id my-first-workflow \
    --input '"Donna"'

Running execution:
  WorkflowId  my-first-workflow
  RunId       ab62c808-44d7-4b3e-97ef-777493c4da09
  Type        GreetSomeone
  Namespace   default
  TaskQueue   greeting-tasks
```

Executing a Workflow from Application Code (1)

- **An alternative to using the CLI 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
    1. Import the client package from the SDK
    2. Create and configure a Client
    3. Use the API to request execution
  - You will use similar code to run Workflows in the exercises

```
package main

import(
    "context"
    "log"
    "app"
    "os"
    "go.temporal.io/sdk/client"  1
)

func main() {
    c, err := client.Dial(client.Options{})  2
    if err != nil {
        log.Fatalln("Unable to create client", err)
    }
    defer c.Close()

    // example continues on next slide
```

Executing a Workflow from Application Code (2)

```
    // continued from previous slide

    options := client.StartWorkflowOptions{
       ID:        "my-first-workflow",
       TaskQueue: "greeting-tasks",
    }

    we, err := c.ExecuteWorkflow(context.Background(), options, app.GreetSomeone, os.Args[1])
    if err != nil {
        log.Fatalln("Unable to execute workflow", err)
    }
    log.Println("Started workflow", "WorkflowID", we.GetID(), "RunID", we.GetRunID())

    var result string
    err = we.Get(context.Background(), &result)
    if err != nil {
        log.Fatalln("Unable get workflow result", err)
    }
    log.Println("Workflow result:", result)
}
```

Making Changes to a Workflow

- **Backwards compatibility is an important consideration in Temporal**
- **You must also ensure that your Workflow is *deterministic***
  - Each execution of a given Workflow must produce the same output, given the same input
  - Careless updates can result in Workflows getting stuck due to non-deterministic errors
- **Workers use caching for better performance**
  - Changes to Workflow or Activity Definitions don’t take effect until Workers are restarted

Exercise Environment Orientation

- You're just about to begin the first hands-on exercise
- Follow the link below to see the GitHub repository containing the code
- Your instructor will explain and demonstrate the exercise environment

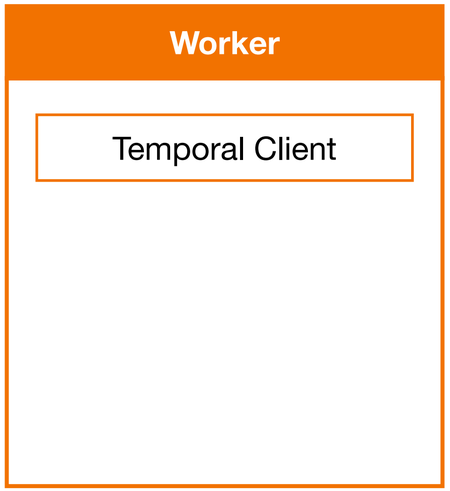
https://t.mp/bbgo26

Exercise #1: Farewell Workflow

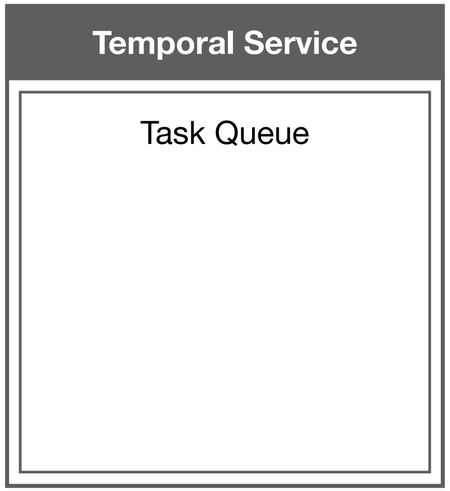
- **During this exercise, you will**
  - Write an Activity function
  - Register the Activity function
  - Modify the Workflow to execute your new Activity
  - Run the Workflow
- **Refer to the README.md file in the exercise environment for details**
  - The code is below the **`exercises/farewell-workflow`** 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

Understanding Workflow Execution

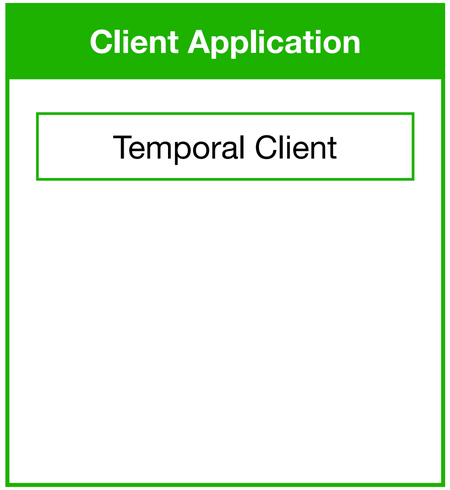
Actors in this Workflow Execution Scenario



Executes the code

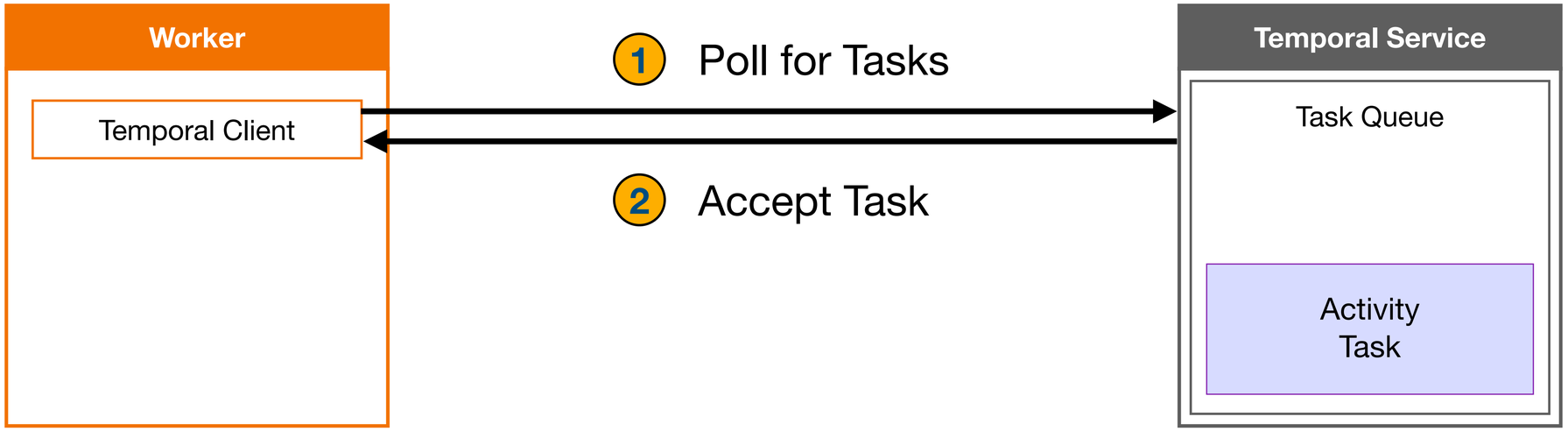


Orchestrates code execution



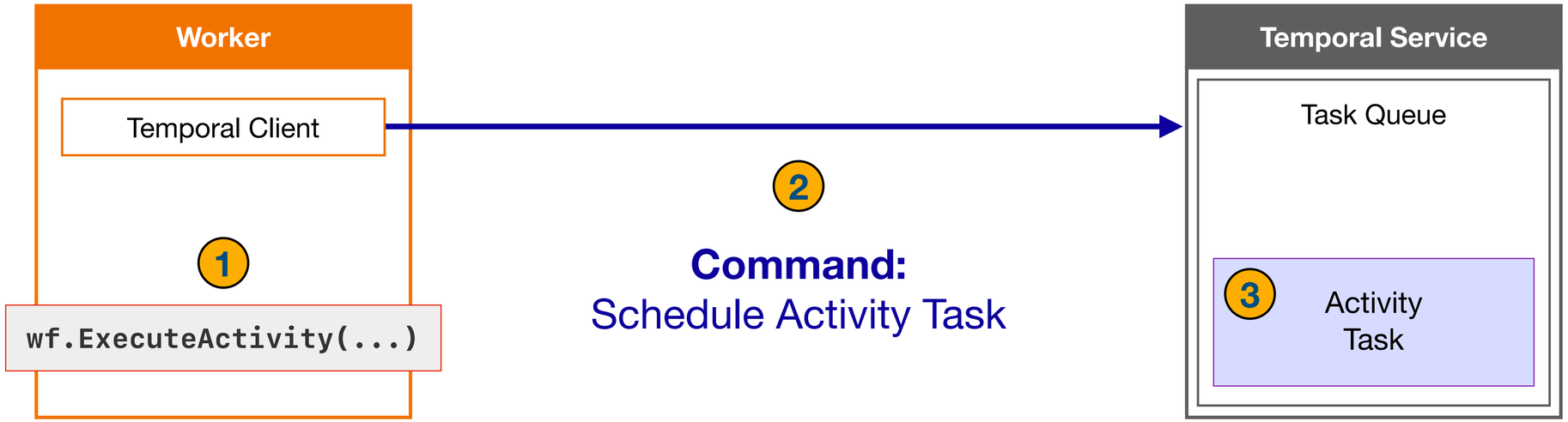
Requests code execution
and retrieves the result

Workers and Tasks



- Temporal does not assign tasks to Workers
- Workers continuously poll, accepting tasks when they have spare capacity
- You can increase throughput and scalability by adding Workers

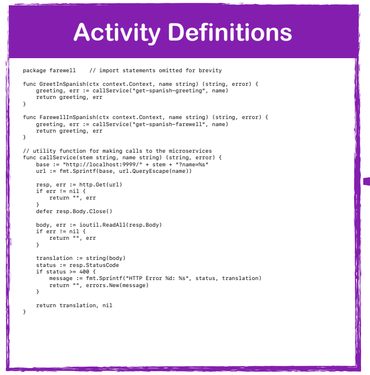
Commands



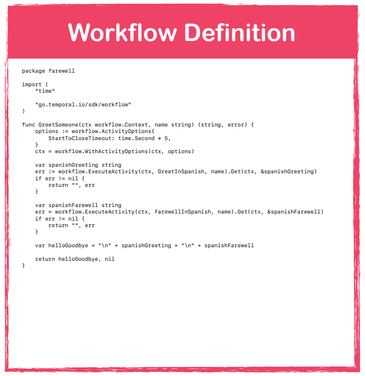
- Certain API calls result in the Worker issuing a Command to the Temporal Service
- The Temporal Service acts on these Commands, but also stores them
- This allows the Worker to recreate the state of a Workflow Execution following a crash

Activity Definitions

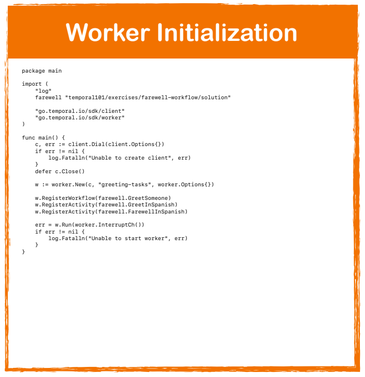
Activity Definitions



Workflow Definition



Worker Initialization



```go
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=%s"
    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
}
```

This is just a utility function for calling the microservice and is not specific to Temporal

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=%s"
    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)
    }
}
```


```
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

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=%s"
    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)
    }
}
```


```
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)
    }
}
```

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=%s"
    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

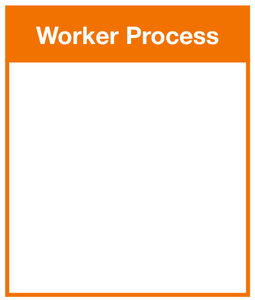
```
package farewell

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
}
```

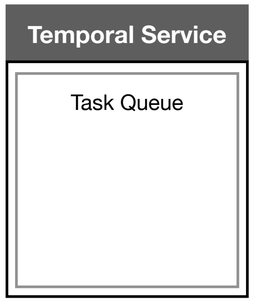
Worker Initialization

```
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)
    }
}
```

Launch

```
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)
    }
}
```





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=%s"
    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
}
```

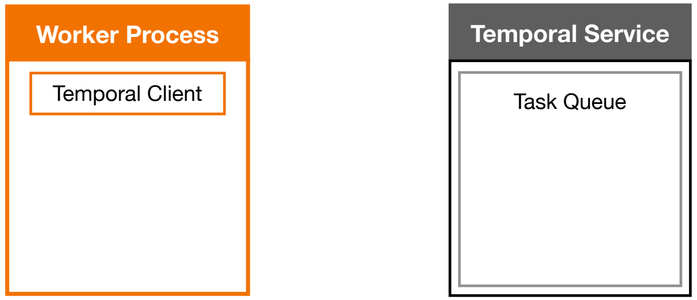
Workflow Definition

```
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)
    }
}
```


```
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)
    }
}
```

Activity Definitions

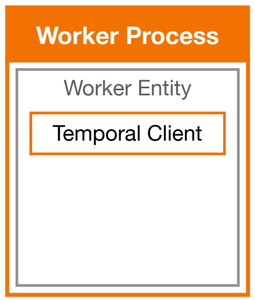
```
package farewell   // import statements omitted for brevity

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=%s"
    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
}
```

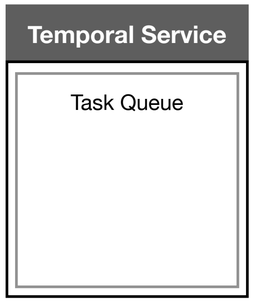
Workflow Definition

```
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)
    }
}
```





Activity Definitions

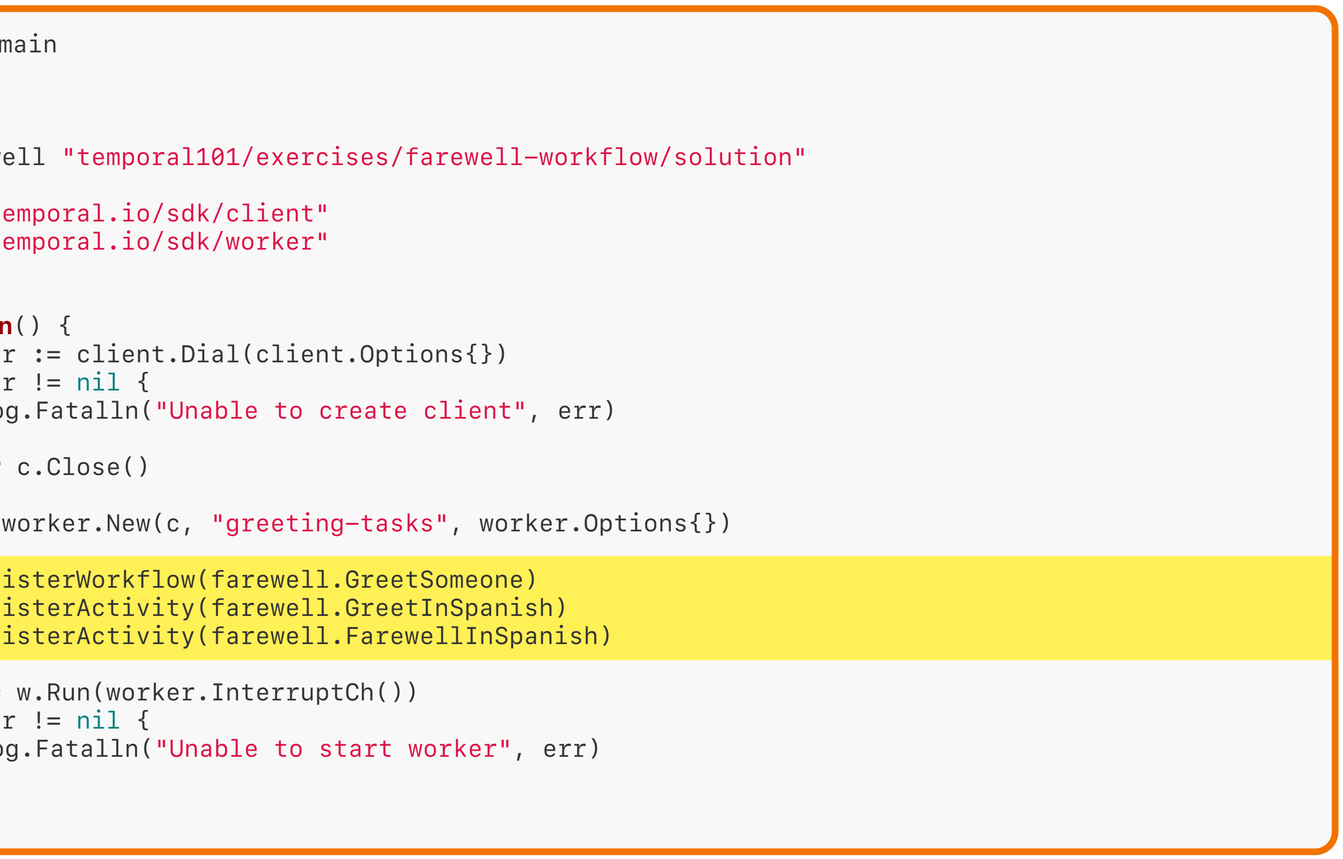
```
package farewell   // import statements omitted for brevity

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=%s"
    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
}
```

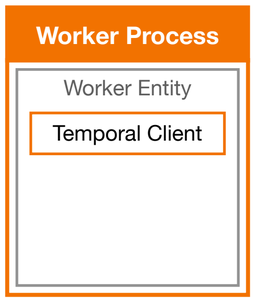
Workflow Definition

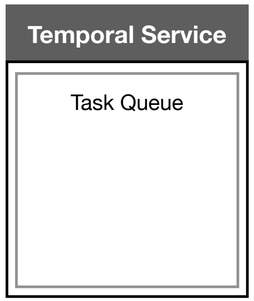
```
package farewell

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)
    }
}
```







Activity Definitions

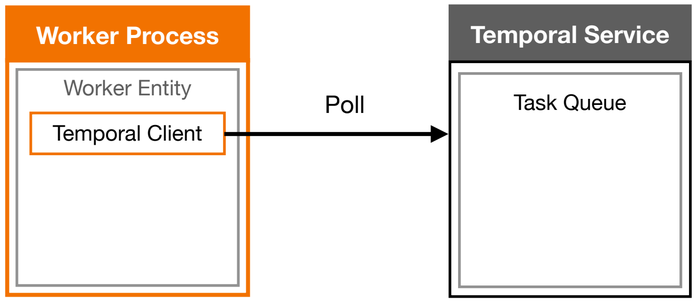
```
package farewell   // import statements omitted for brevity

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=%s"
    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)
    }
}
```



Start the Workflow

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=%s"
    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
}
```

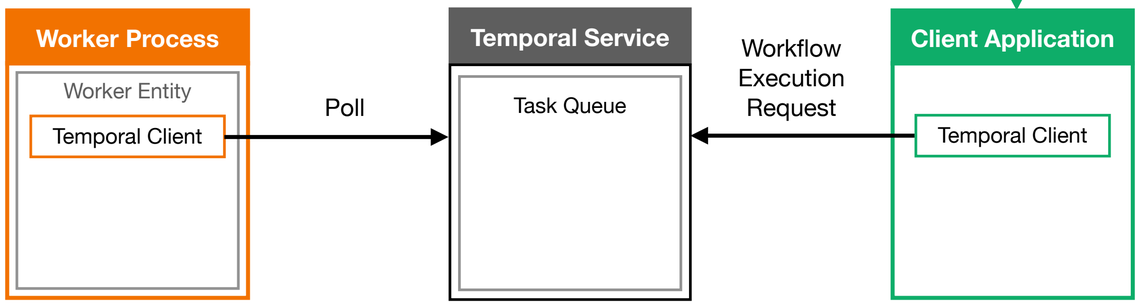
Workflow Definition

```
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)
    }
}
```

```
// ... this is code within your own application (for example, a web application, mobile app, etc.)

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

options := client.StartWorkflowOptions{
    ID:        "greeting-workflow",
    TaskQueue: "greeting-tasks",
}

we, err := c.ExecuteWorkflow(context.Background(), options, farewell.GreetSomeone, os.Args[1])
if err != nil {
    log.Fatalln("Unable to execute workflow", err)
}
log.Println("Started workflow", "WorkflowID", we.GetID(), "RunID", we.GetRunID())

var result string
err = we.Get(context.Background(), &result)
if err != nil {
    log.Fatalln("Unable get workflow result", err)
}
log.Println("Workflow result:", result)


// ... other application-specific code might follow
```



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=%s"
    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

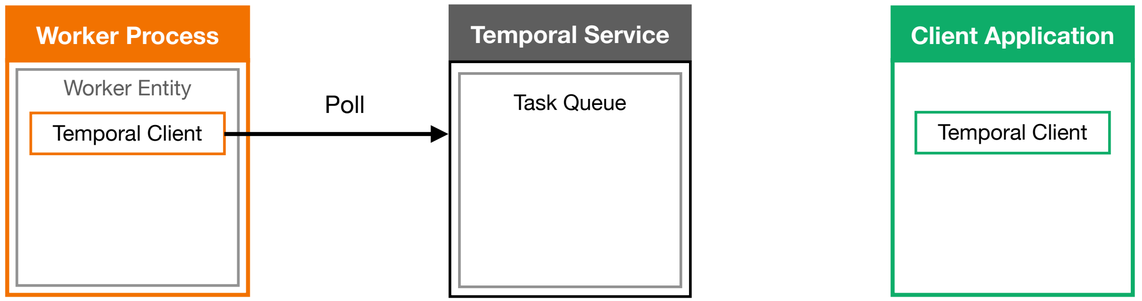
```
package farewell

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=%s"
    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

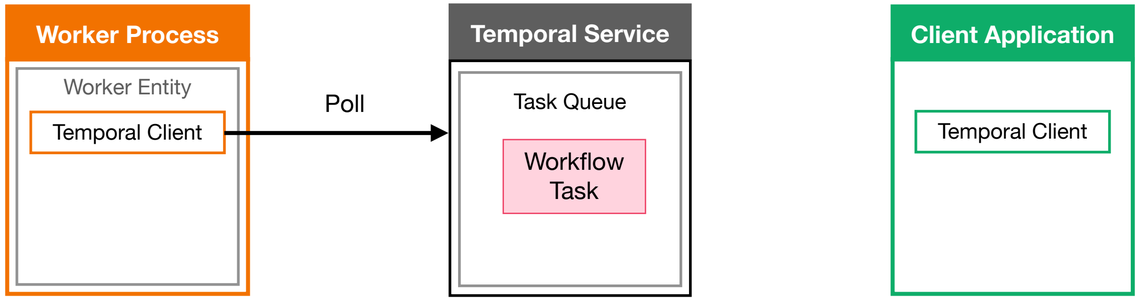
```
package farewell

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=%s"
    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

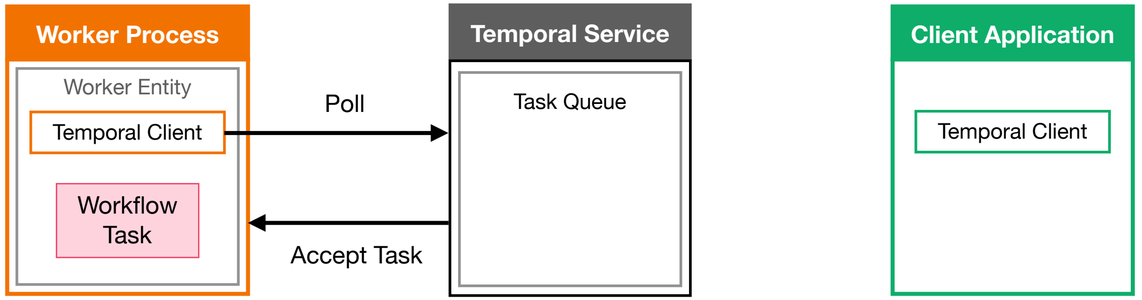
```
package farewell

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=%s"
    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
}
```

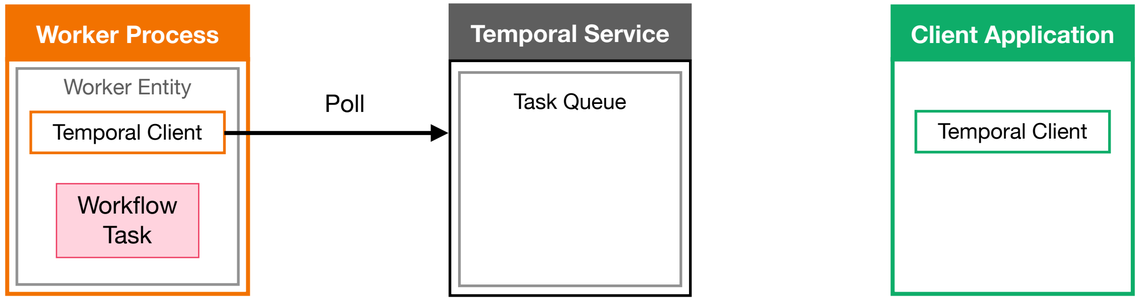
Worker Initialization

```
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)
    }
}
```

```
// ... code above has been omitted from this excerpt


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
}
```

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=%s"
    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
}
```

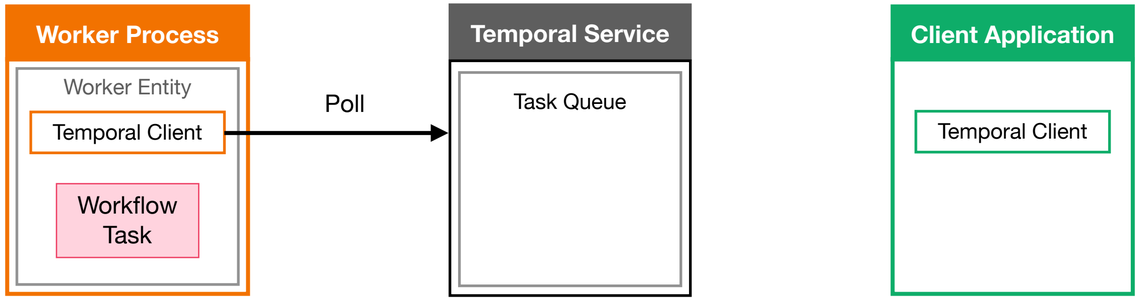
Worker Initialization

```
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)
    }
}
```

```
// ... code above has been omitted from this excerpt


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
}
```

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=%s"
    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
}
```

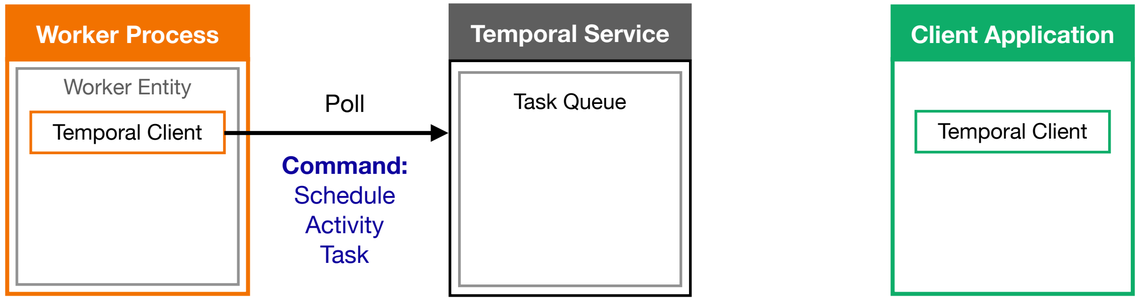
Worker Initialization

```
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)
    }
}
```

```
// ... code above has been omitted from this excerpt


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
}
```

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=%s"
    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

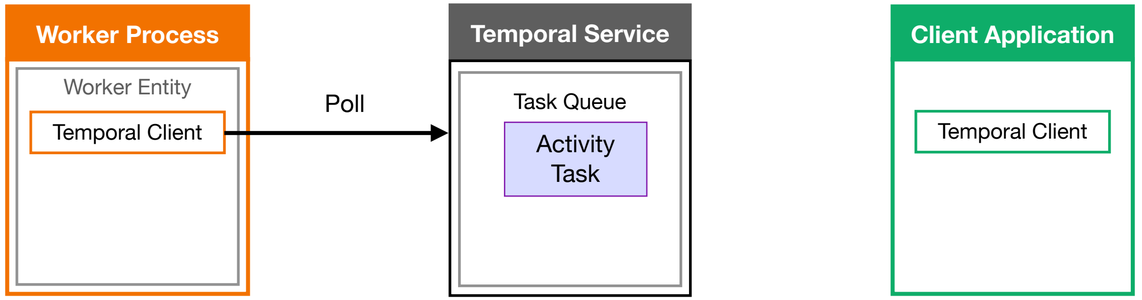
```
package farewell

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=%s"
    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

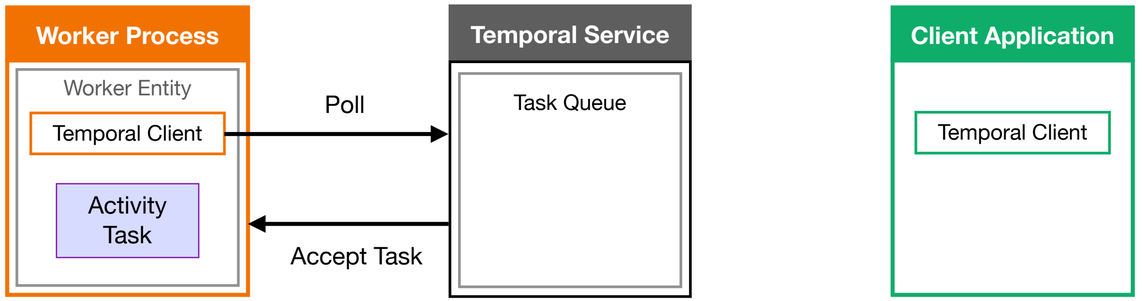
```
package farewell

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

```go
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=%s"
    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

```go
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
}
```

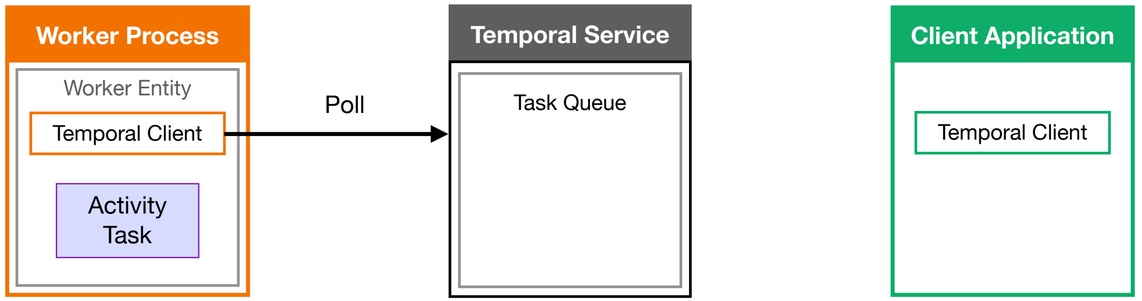
Worker Initialization

```go
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)
    }
}
```

```go
// import statements and unused code omitted from this example
func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", 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=%s"
    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
}
```

Event History

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



Activity Definitions

```go
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=%s"
    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

```go
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
}
```

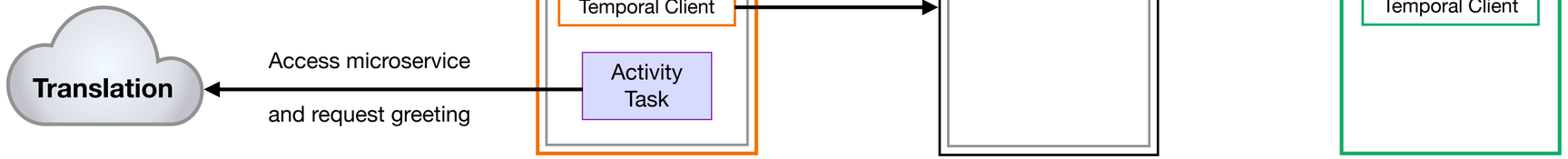
Worker Initialization

```go
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)
    }
}
```

```go
// import statements and unused code omitted from this example
func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", 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=%s"
    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
}
```

Event History

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



Activity Definitions

```go
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=%s"
    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

```go
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
}
```

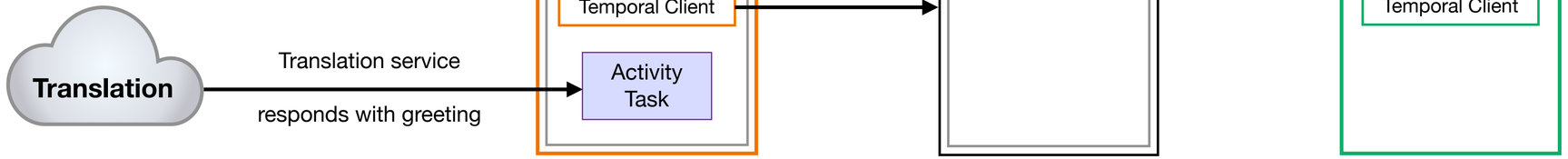
Worker Initialization

```go
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)
    }
}
```

```go
// import statements and unused code omitted from this example
func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", 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=%s"
    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
}
```

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=%s"
    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
}
```

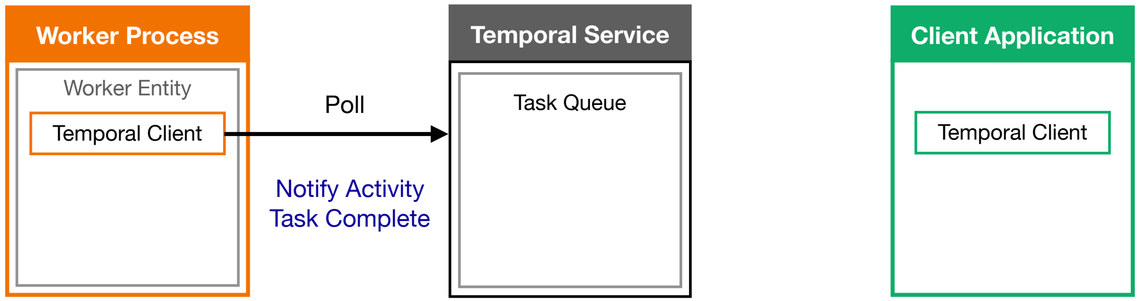
Worker Initialization

```
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)
    }
}
```

```go
// import statements and unused code omitted from this example
func GreetInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", 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=%s"
    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
}
```

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=%s"
    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

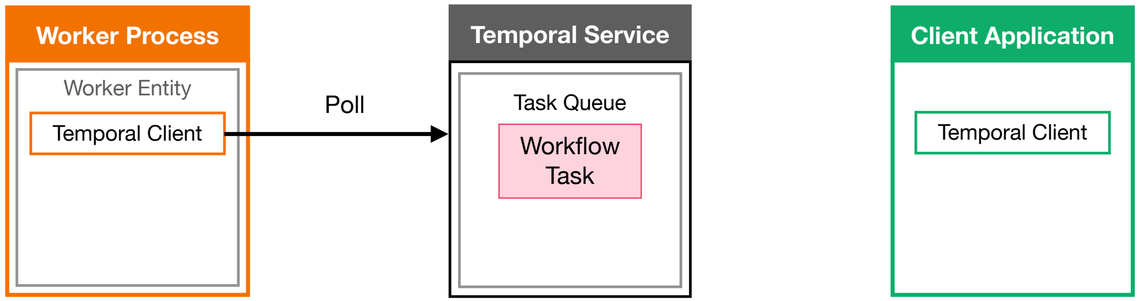
```
package farewell

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=%s"
    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

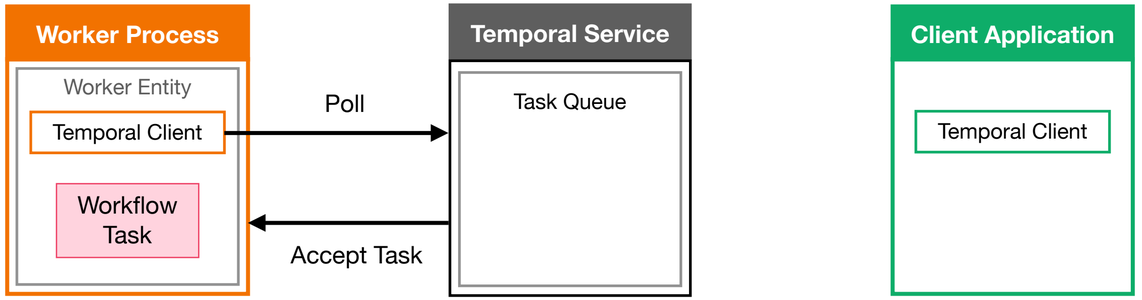
```
package farewell

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=%s"
    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
}
```

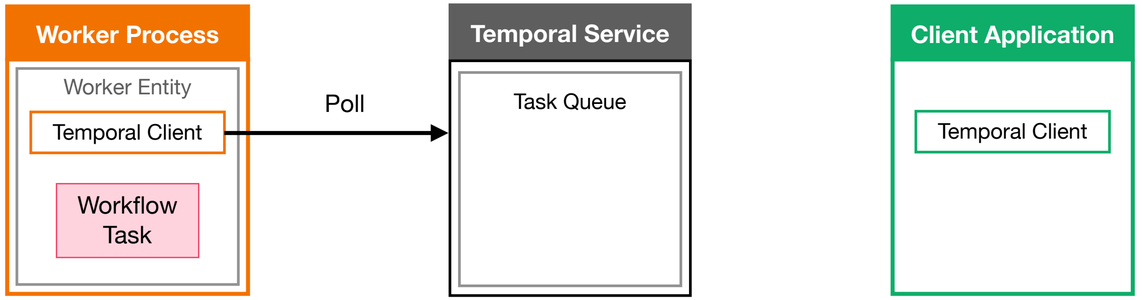
Worker Initialization

```
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)
    }
}
```

```
// ... code above has been omitted from this excerpt


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
}
```

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=%s"
    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
}
```

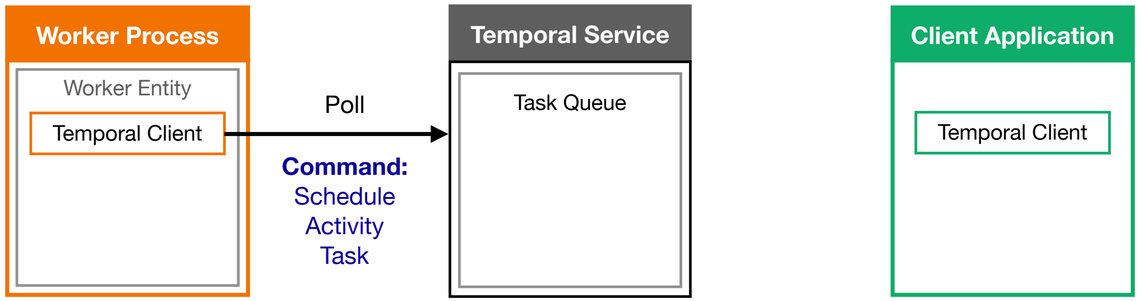
Worker Initialization

```
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)
    }
}
```

```
// ... code above has been omitted from this excerpt


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
}
```

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=%s"
    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

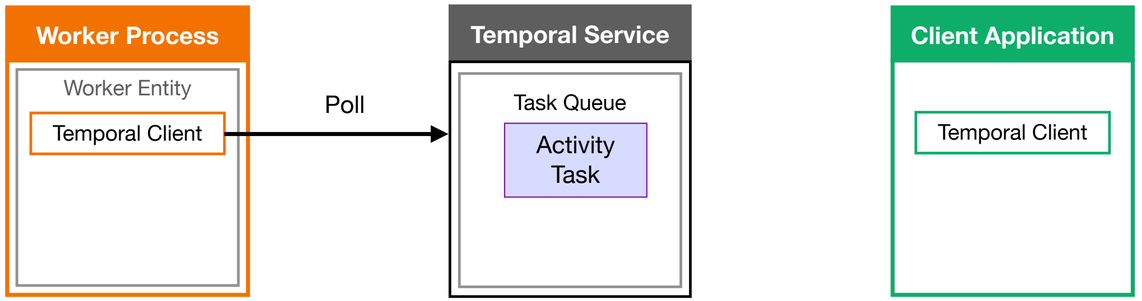
```
package farewell

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=%s"
    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
}
```

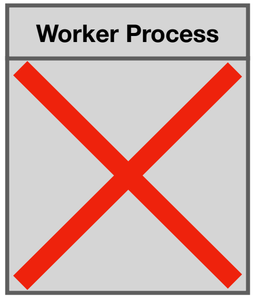
Workflow Definition

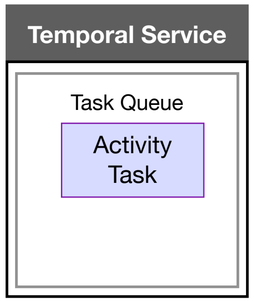
```
package farewell

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
}
```

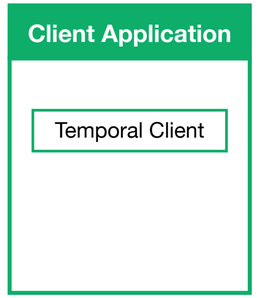
Worker Initialization

```
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=%s"
    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

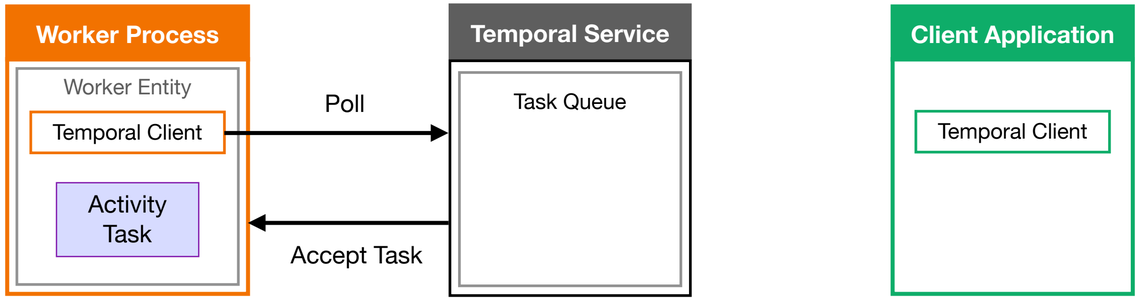
```
package farewell

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=%s"
    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
}
```

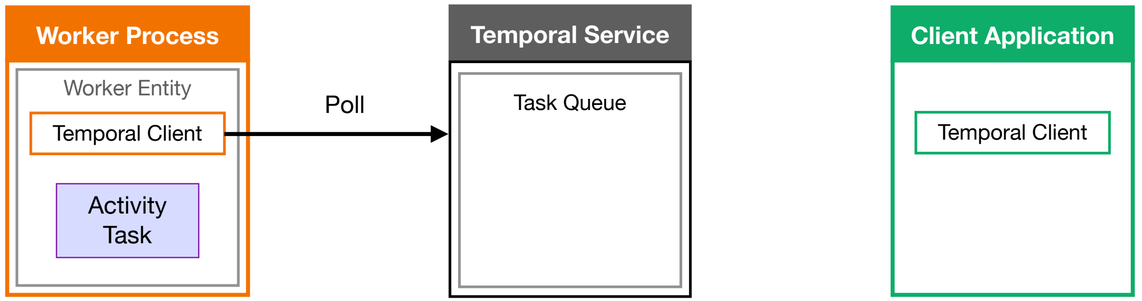
Worker Initialization

```
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)
    }
}
```

```go
// import statements and unused code omitted from this example
func FarewellInSpanish(ctx context.Context, name string) (string, error) {
    greeting, err := callService("get-spanish-greeting", 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=%s"
    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
}
```

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=%s"
    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
}
```

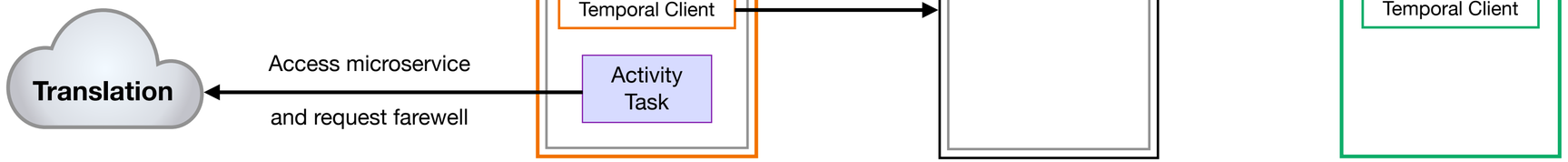
Worker Initialization

```
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)
    }
}
```

```go
// import statements and unused code omitted from this example
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=%s"
    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
}
```

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=%s"
    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

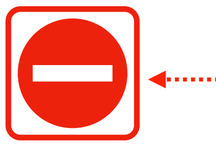
```
package main

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 statements and unused code omitted from this example
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=%s"
    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
}
```

Error

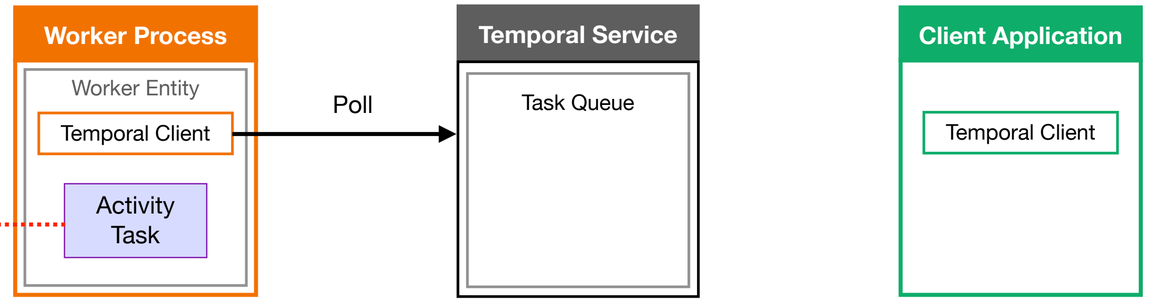
Event History

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=%s"
    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

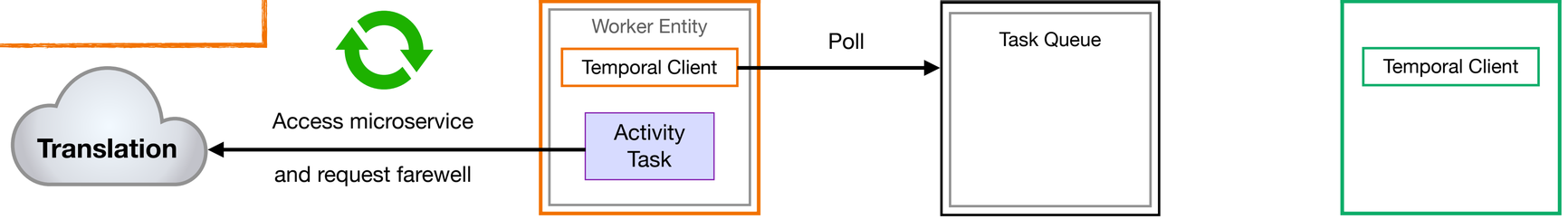
```
package main

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 statements and unused code omitted from this example
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=%s"
    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
}
```

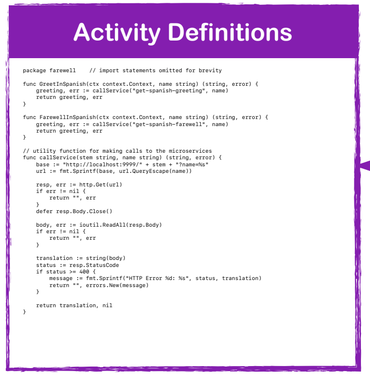
Activity is invoked again during retry

Event History

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

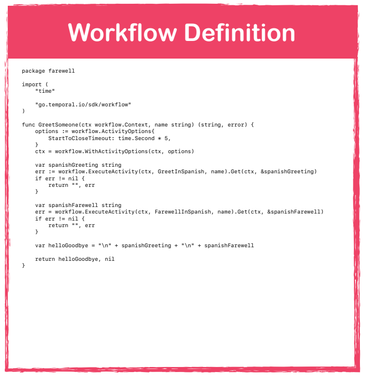


Activity Definitions



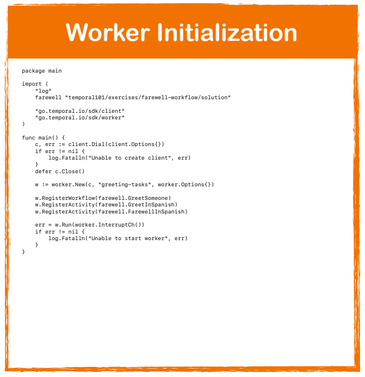
```
package farewell   // import statements omitted for brevity

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=%s"
    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

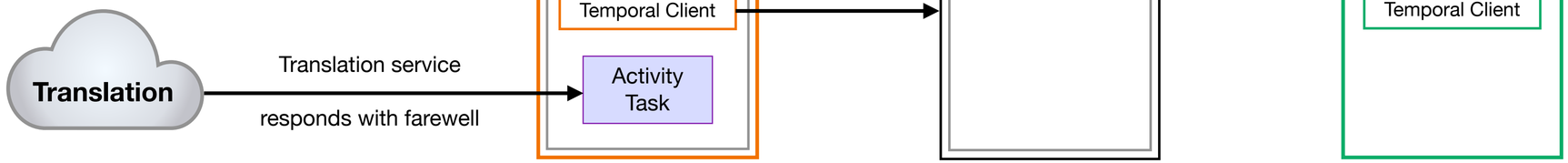


```
package main

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 statements and unused code omitted from this example
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=%s"
    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
}
```

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=%s"
    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
}
```

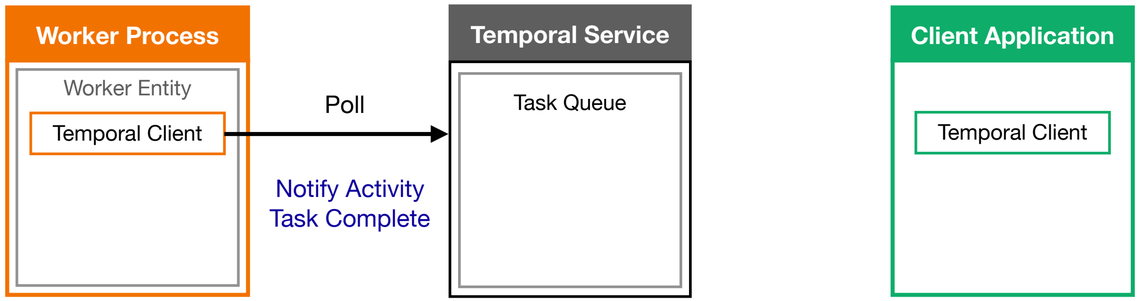
Workflow Definition

```
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)
    }
}
```

```go
// import statements and unused code omitted from this example
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=%s"
    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
}
```



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=%s"
    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

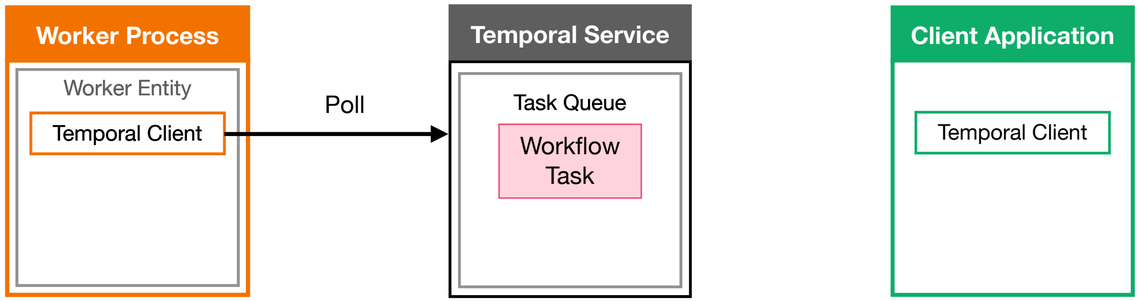
```
package farewell

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

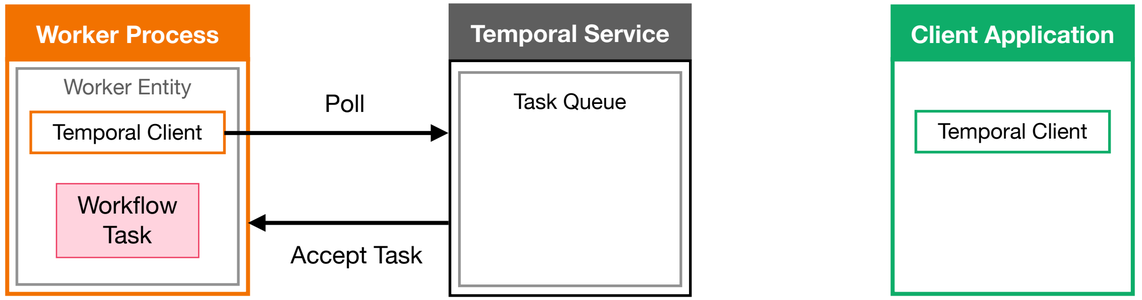
```
package farewell   // import statements omitted for brevity

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=%s"
    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=%s"
    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
}
```

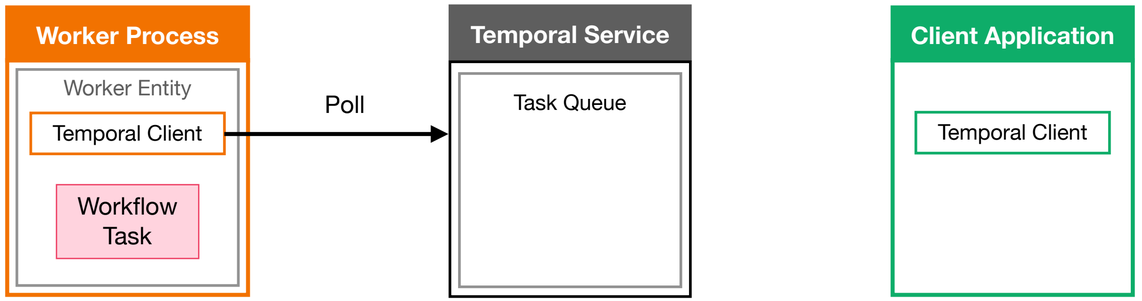
Worker Initialization

```
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)
    }
}
```

```
// ... code above has been omitted from this excerpt


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
}
```

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=%s"
    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
}
```

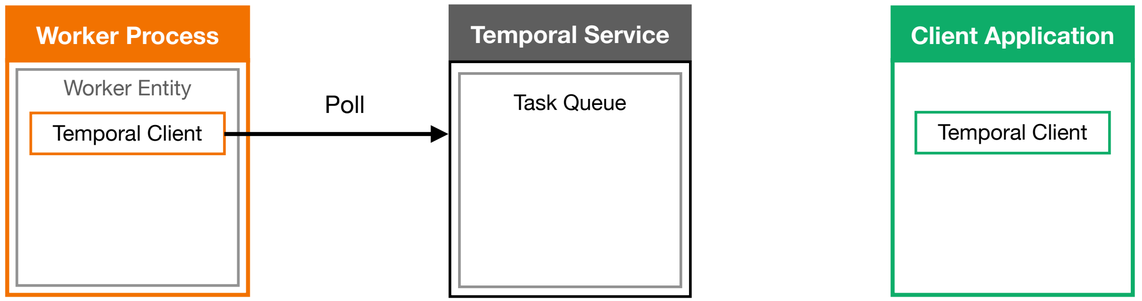
Worker Initialization

```
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)
    }
}
```

```
// ... code above has been omitted from this excerpt


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
}
```

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=%s"
    url := fmt.Sprintf(fbase, 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
}
```

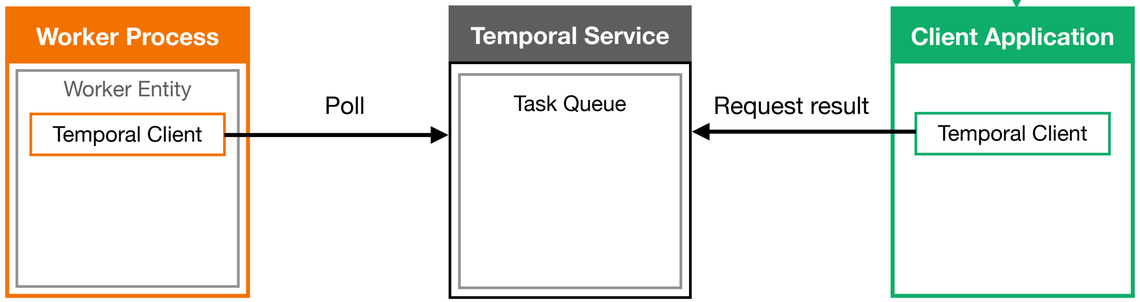
Workflow Definition

```
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)
    }
}
```

```
// ... this is code within your own application (for example, a web application, mobile app, etc.)


options := client.StartWorkflowOptions{
    ID:        "greeting-workflow",
    TaskQueue: "greeting-tasks",
}

we, err := c.ExecuteWorkflow(context.Background(), options, farewell.GreetSomeone, os.Args[1])
if err != nil {
    log.Fatalln("Unable to execute workflow", err)
}
log.Println("Started workflow", "WorkflowID", we.GetID(), "RunID", we.GetRunID())

var result string
err = we.Get(context.Background(), &result)
if err != nil {
    log.Fatalln("Unable get workflow result", err)
}
log.Println("Workflow result:", result)


// ... other application-specific code might follow
```



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=%s"
    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
}
```

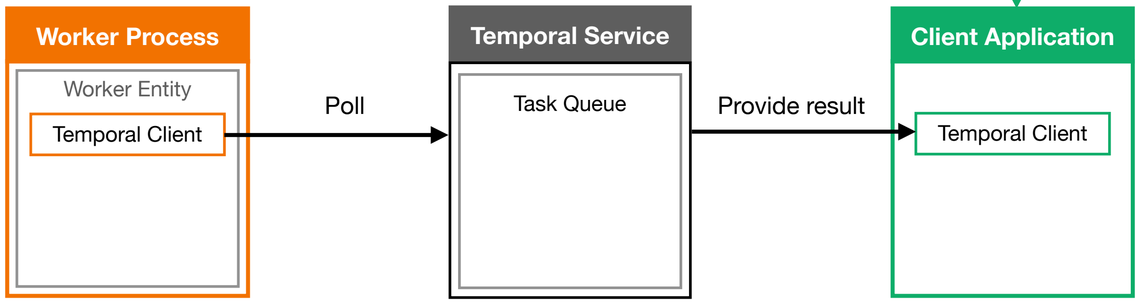
Workflow Definition

```
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)
    }
}
```

```
// ... this is code within your own application (for example, a web application, mobile app, etc.)


options := client.StartWorkflowOptions{
    ID:        "greeting-workflow",
    TaskQueue: "greeting-tasks",
}

we, err := c.ExecuteWorkflow(context.Background(), options, farewell.GreetSomeone, os.Args[1])
if err != nil {
    log.Fatalln("Unable to execute workflow", err)
}
log.Println("Started workflow", "WorkflowID", we.GetID(), "RunID", we.GetRunID())

var result string
err = we.Get(context.Background(), &result)
if err != nil {
    log.Fatalln("Unable get workflow result", err)
}
log.Println("Workflow result:", result)


// ... other application-specific code might follow
```



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	

Beginner to Builder Bootcamp - Go

Compatible Evolution of Input Parameters

- Workflows and Activities can take any number of parameters as input
  - Changing the number, position, or type of these parameters can affect backwards compatibility
- It is a best practice to pass all input in a single `struct`
  - Changes to the composition of this struct does not affect the function signature
- This is also the recommended approach for return values
  - Using structs in both places allows for evolution of input and output data

Example: Using a `struct` in an Activity (1)

- Imagine that you have the following Activity

```
// This Activity returns a customized greeting in English, using the provided name

func CreateGreeting(ctx context.Context, name string) (string, error) {
```

input output

- You later need to update it to support other languages, such as Spanish
  - Changing what is passed into or returned from the function changes its signature
  - Changes to the struct composition don't affect the signature of the functions that use it

Example: Using a `struct` in an Activity (2)

- **The following code sample illustrates how you could support this**

```
// Define a struct to encapsulate all data passed as input for this Activity
type GreetingInput struct {
    Name         string
    LanguageCode string
}

// Define a struct to encapsulate the data returned by this Activity
type GreetingOutput struct {
    Greeting     string
}

// Specify these types for the input parameter and return value of the Activity
func CreateGreeting(ctx context.Context, input GreetingInput) (GreetingOutput, error) {

    // An example to show how to access input parameters and create the return value
    if input.LanguageCode == "fr" {
        bonjour := fmt.Sprintf("Bonjour, %s", input.Name)
        return new GreetingOutput{ Greeting: bonjour, }, nil
    }
    // support for additional languages would follow...
```

Task Queues

- **Temporal Service coordinates with Workers through named Task Queues**
  - The name of this Task Queue is specified in the Worker configuration

```
w := worker.New(client, "translation-tasks", worker.Options{})
```

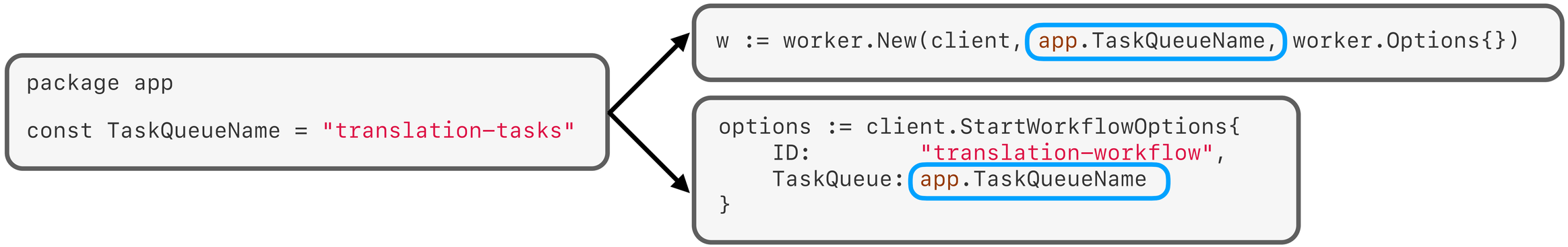
  - The Task Queue name is also specified by a Client when starting a Workflow

```
options := client.StartWorkflowOptions{
    ID:        "translation-workflow",
    TaskQueue: "translation-tasks",
}
```

  - Task Queues are dynamically created, so a name mismatch does not result in an error!

Recommendations for Task Queues

- **Use a shared constant to avoid hardcoding the name in multiple places**



  - Avoid mixed case: Task Queue names are case sensitive
  - Use descriptive names, but make them as short and simple as practical

- **Plan to run *at least* two Worker Processes per Task Queue**
  - Improves scalability: Load will be distributed among multiple Workers
  - Improves availability: If one Worker crashes, other Workers can take over for it

Workflow IDs

- **You specify a Workflow ID when starting a Workflow Execution**
  - This should be a value that is meaningful to your business logic

```
// Example: An order processing Workflow might include order number in the Workflow ID
options := client.StartWorkflowOptions{
    ID:         "process-order-number-", + input.OrderNumber
    TaskQueue: app.TaskQueueName,
}

run, err := c.ExecuteWorkflow(context.Background(), options, ProcessOrderWorkflow, input)
```

- **Must be unique among all *running* Workflow Executions in the Namespace**
  - This constraint applies across *all* Workflow Types, not just those of the *same Type*
  - This is an important consideration for choosing a Workflow ID

How Errors Affect Workflow Execution

- **An Activity that returns an error is considered as failed**
  - It may or may not retried, based on the Retry Policy associated with its execution
  - By default, Activity Execution is associated with a Retry Policy
    - The default policy results in retrying until execution succeeds or is canceled
- **A Workflow that returns an error is also considered as failed**
  - By default, Workflow Execution is *not* associated with a Retry Policy
  - Failing an Activity is common, but failing a Workflow is considered unusual
    - It is considered a better practice to fix the Workflow

How to Return Errors in Application Code

- **You can return errors as necessary in Workflows or Activities**

```
resp, err := http.Get(url)
if err != nil {
   return "", errors.New("request failed")
}
```

```
resp, err := http.Get(url)
if err != nil {
    // rethrow the error from the failed HTTP request
    return "", err
}
```

- **Developers are not *required* to use a Temporal-specific API for errors**
  - Application errors are automatically converted into a language-neutral format

Logging in Temporal Applications

- **The recommended way of logging is via the interface in the Go SDK**
  - The SDK also provides a very basic logging implementation, which you can replace
- **This interface defines four log levels, in increasing order of importance**
  - `Debug`
  - `Info`
  - `Warn`
  - `Error`

Using the Logger Interface

- **Accessing and using the Workflow logger**
  - Log statements can include any number of key-value pairs

```
logger := workflow.GetLogger(ctx)

logger.Debug("Preparing to execute an Activity")
logger.Info("Calculated cost of order", "Tax", tax, "Total", total)
```

- **Accessing and using the Activity logger is similar**

```
logger := activity.GetLogger(ctx)

logger.Info("Looking up customer in the database", "Key", customerID)
logger.Error("Database connection failed")
```

Long-Running Executions

- **Temporal Workflows may have executions that span several years**
  - Activities may also run for long periods of time
- **Workflow and Activity Executions are asynchronous operations**
  - The following calls submit *execution requests* to the Temporal Service
  - They do not block while waiting for execution to complete

```
// Use a Temporal Client to request Workflow execution
future, err := client.ExecuteWorkflow(context.Background(), options, MyWorkflow, input)
```

```
// Request Activity execution from within a Workflow
future := workflow.ExecuteActivity(ctx, MyActivity, input)
```

Waiting on Execution Results

- **It is common to chain the Execution request and result retrieval**
  - Many Temporal APIs use a Future to provide access to results from asynchronous execution
  - Calling `Get` on this value blocks until the execution is complete

```
// This chained call blocks until Activity Execution returns a result or error
var result string
err := workflow.ExecuteActivity(ctx, MyActivity, input).Get(ctx, &result)
```

Deferring Access to Execution Results

- **Deferring access to results *may* reduce overall execution time**
  - This is a good strategy when a Workflow needs to call unrelated Activities
  - It allows these Activities to execute in parallel, blocking only while accessing their results

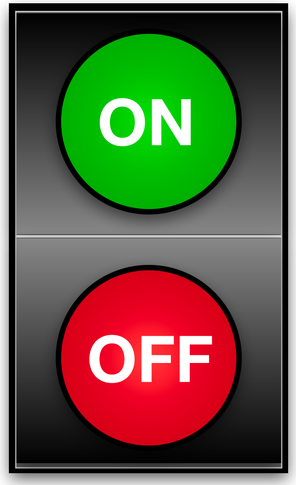
```
// Request execution of multiple Activities: these calls do not block
futureA := workflow.ExecuteActivity(ctx, MyActivityA, inputA)
futureB := workflow.ExecuteActivity(ctx, MyActivityB, inputB)
futureC := workflow.ExecuteActivity(ctx, MyActivityC, inputC)

// The following lines block until their respective executions have finished
var resultA string
errA := futureA.Get(ctx, &resultA)

var resultB string
errB := futureB.Get(ctx, &resultB)

var resultC string
errC := futureC.Get(ctx, &resultC)
```

Idempotence

- An operation is idempotent if subsequent invocations do not adversely change state beyond that of the initial invocation

- Consider the idempotence of buttons used to control device power

Toggle Button



Separate On/Off Buttons

Activity Idempotence

- **It is strongly recommended that you make your Activities idempotent**
  - A non-idempotent Activity could adversely affect the state of the system
- **For example, consider an Activity that performs the following steps**
  1. Queries a database
  2. Calls a microservice using data returned by the query
  3. Writes the result of the microservice call to the filesystem
- **This will be retried if any one of those steps fails**
  - You should balance the granularity of your Activities with the need to keep Event History small

Idempotence and At-Least-Once Execution

- **Idempotence is also important due to a distributed systems edge case**
- **Consider the following scenario**
  - Worker polls the Temporal Service and accepts an Activity Task
  - Worker begins executing the Activity
  - Worker finishes executing the Activity
  - Worker crashes just before reporting the result to the Temporal Service
- **Activity will be retried since Event History does not indicate completion**
  - Therefore, idempotence is essential for preventing unwanted changes in application state

Idempotency Keys

- **You can achieve idempotency by ignoring duplicate requests**
  - This raises a question: How can one distinguish a *duplicate* request from one that looks similar?
- **Idempotency keys are unique identifiers associated with a request**
  - They are interpreted by the system receiving the request (e.g., a payment processor)
  - In a Temporal Activity, you can compose one from a Workflow Run ID and Activity ID
  - Guaranteed to be consistent across retry attempts, but unique among Workflow Executions

```
workflowID := workflow.GetInfo(ctx).WorkflowExecution.ID
activityID := activity.GetInfo(ctx).ActivityID
idempotencyKey := workflowID + "-" + activityID
```

Beginner to Builder Bootcamp - Go

What is a Timer?

- **Timers are used to introduce delays into a Workflow Execution**
  - Code that awaits the Timer pauses execution for the specified duration
  - The duration is fixed and may range from seconds to years
  - Once the time has elapsed, the Timer fires, and execution continues
- **Workflow code must not use Go's built-in timers (non-deterministic)**

Use Cases for Timers

- **Execute an Activity multiple times at predefined intervals**
  - Send reminder e-mails to a new customer after 1, 7, and 30 days
- **Execute an Activity multiple times at dynamically-calculated intervals**
  - Delay calling the next Activity based on a value returned by a previous one
- **Allow time for offline steps to complete**
  - Wait five business days for a check to clear before proceeding

Timer APIs Provided by the Go SDK

- **The Go SDK offers two Workflow-safe ways to start a Timer**
  - These correspond to two functions in the Go `time` package
  - Workflow code must not use Go's functions for timers (this is non-deterministic)

Pausing Workflow Execution for a Specified Duration

- **Use the `workflow.Sleep` function for this**
  - This is an alternative to Go's `time.Sleep` function
  - It blocks until the Timer is fired (or is canceled)

```go
// This will pause Workflow Execution for 10 seconds
// The first parameter (ctx) is the context passed to the Workflow
err := workflow.Sleep(ctx, time.Second*10)
```

Running Code a Specific Point in the Future

- **Use the `workflow.NewTimer` function for this**
  - This is an alternative to Go's `time.NewTimer` function
  - This returns a `Future`, which becomes ready when the Timer fires (or is canceled)

```
// workflow.Sleep is a Workflow-safe counterpart to time.Sleep
timerFuture := workflow.NewTimer(ctx, time.Second * 5)
logger.Info("The timer was set")


// Unlike workflow.Sleep, waiting for the timer here is a separate operation
logger.Info("Waiting until the timer has fired")
err := timerFuture.Get(ctx, nil)
```

What Happens to a Timer if the Worker Crashes?

- **Timers are maintained by the Temporal Service**
  - Once set, they fire regardless of whether any Workers are running
- **Scenario: Timer set for 10 seconds and Worker crashes 3 seconds later**
  - If Worker is restarted within 7 seconds, it will be running when the Timer fires
    - It will be as if the Worker had never crashed at all
  - If Worker is restarted 5 *minutes* later, the Timer will have already fired
    - In this case, the Worker will resume executing the Workflow code without delay

Exercise #2: Observing Durable Execution

- **During this exercise, you will**
  - Create Workflow and Activity loggers
  - Add logging statements to the code
  - Add a Timer to the Workflow Definition
  - Launch two Workers, run the Workflow, and kill one of the Workers, observing that the remaining Worker completes the execution

- **Refer to this exercise's `README.md` file for details**
  - The code is below the **`exercises/durable-execution`** directory
  - Don't forget to make your changes in the `practice` subdirectory

Beginner to Builder Bootcamp - Go

Workflow Definition

combined with

Execution Request

results in

Workflow Execution

```go
package example

import (
    "time"

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

func MyWorkflow(ctx workflow.Context, input MyWorkflowInput) (MyWorkflowOutput, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var activityResult MyActivityOutput
    err := workflow.ExecuteActivity(ctx, MyActivity).Get(ctx, &activityResult)
    if err != nil {
        return MyWorkflowOutput{}, err
    }

    return MyWorkflowOutput{ activityResult.Name }, nil
}
```

+

```go
client.ExecuteWorkflow(context.Background(), options, example.MyWorkflow, input)
```

=

Running Workflow

1 Workflow Definition

combined with

n Execution Requests

results in

n Workflow Executions

```go
package example

import (
    "time"

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

func MyWorkflow(ctx workflow.Context, input MyWorkflowInput) (MyWorkflowOutput, error) {
    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var activityResult MyActivityOutput
    err := workflow.ExecuteActivity(ctx, MyActivity).Get(ctx, &activityResult)
    if err != nil {
        return MyWorkflowOutput{}, err
    }

    return MyWorkflowOutput{ activityResult.Name }, nil
}
```

+ +

```
client.ExecuteWorkflow(..., {"ID": 812} )
```

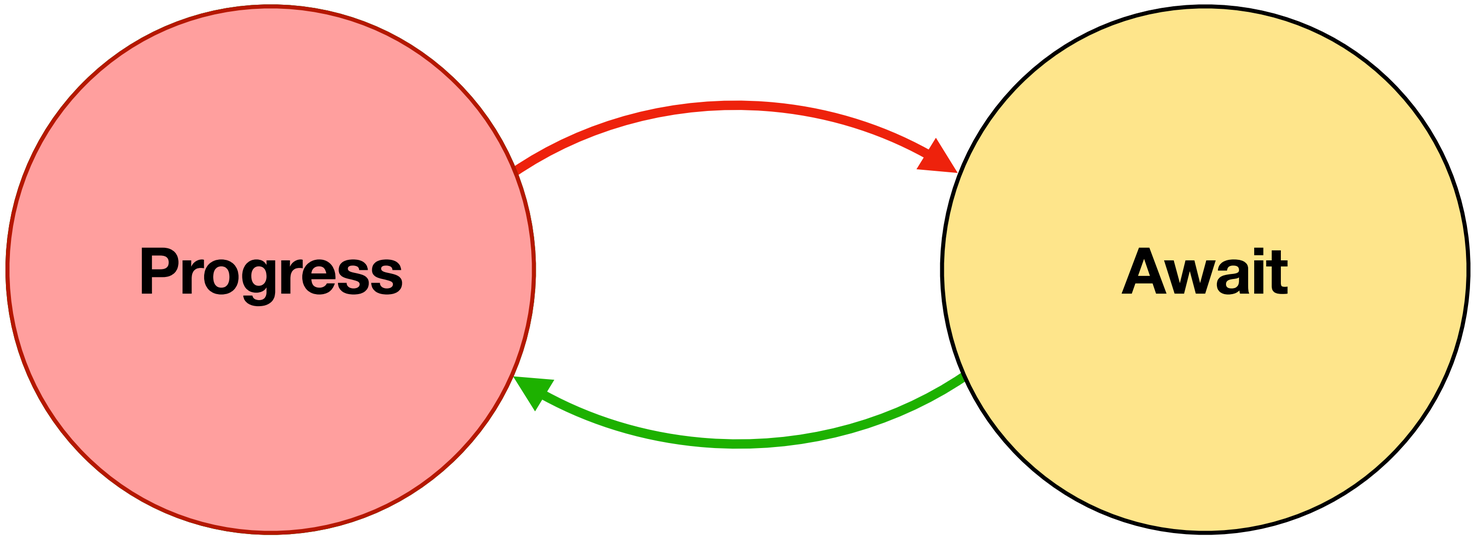
```
client.ExecuteWorkflow(..., {"ID": 947} )
```

= =

Workflow Execution 1

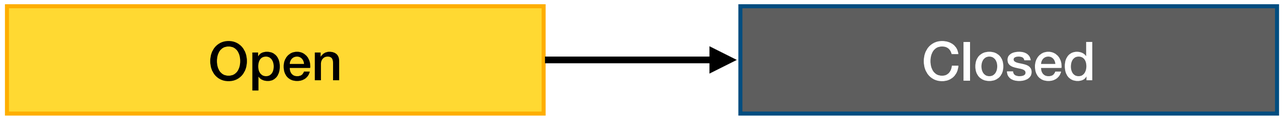
Workflow Execution 2

What Happens During Workflow Execution



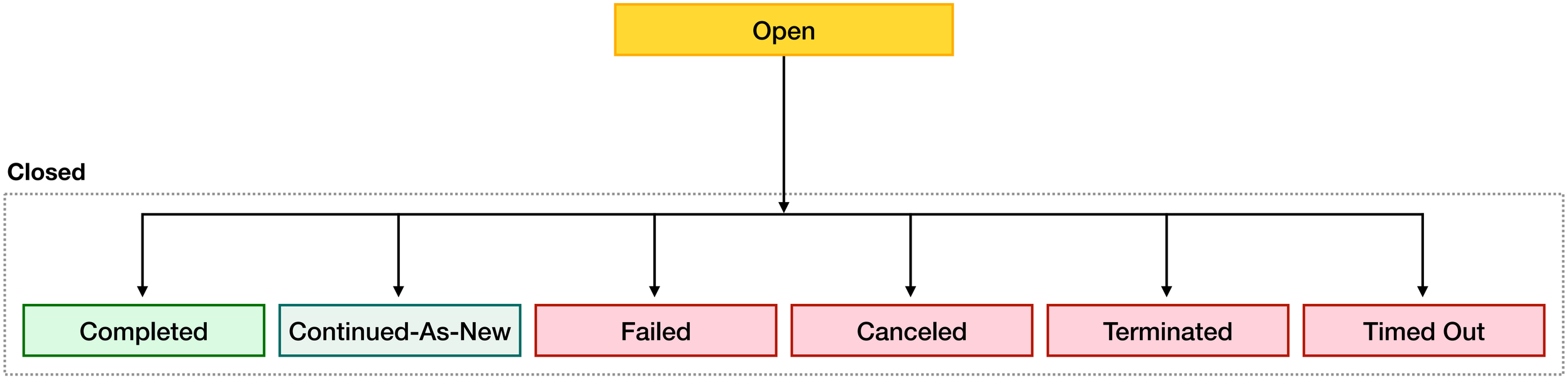
This cycle continues throughout Workflow Execution

Workflow Executions transition from Open (running) to Closed (ended)



This is a one-way transition

Summary of Workflow Execution States



How Workflow Code Maps to Commands

pseudocode

```
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```

Basic Temporal Workflow Definition

- Defines a Start-to-Close Timeout
- Calculates total price of the pizzas
- Determines distance to customer
- Fails if customer is too far away for delivery
- Sleeps for 30 minutes
- Populates a struct with billing information
- Sends a bill to the customer

```
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```

Basic Temporal Workflow Definition

- A Workflow is a sequence of steps
- Some steps are *internal to the Workflow*
  - Involve no interaction with Temporal Service
- Examples include
  - Setting configuration parameters
  - Performing calculations
  - Evaluating variables or expressions
  - Populating data structures
- These internal steps are highlighted in white

```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```

Basic Temporal Workflow Definition

- Some steps *do* interact with the Temporal Service
- Examples include
  - Executing an Activity
  - Setting a Timer
  - Returning an error from the Workflow
  - Returning a value from the Workflow
- These external steps are highlighted in yellow

```
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {
   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```

```
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```

```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```

Command

ScheduleActivityTask

("pizza-tasks", GetDistance, { Line1: "123 Oak St.", Line2: "", ... })

```
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```

```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```

Command

StartTimer

(30 minutes)

```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```

```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```

Command

ScheduleActivityTask

("pizza-tasks", SendBill, { Amount: 2750, Description: "Pizzas", ... }

```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```

Command

CompleteWorkflowExecution

({ConfirmationNumber: "TPD-26074139"})

Workflow Execution Event History

- **Each Workflow Execution is associated with an Event History**
- **Represents the source of truth for what transpired during execution**
  - As viewed from the perspective of the Temporal Service
  - Durably persisted by the Temporal Service
- **Event Histories serve two key purposes in Temporal**
  - Allow reconstruction of Workflow state following a crash
  - Enable developers to investigate both current and past executions
- **You can access them from code, command line, and Web UI**

Event History Content

- **An Event History acts as an ordered append-only log of Events**
  - Begins with the `WorkflowExecutionStarted` Event
  - New Events are appended as Workflow Execution progresses
  - Ends when the Workflow Execution closes

Event History Limits

- **Temporal places limits on a Workflow Execution's Event History**
- **Warnings begin after 10K (10,240) Events**
  - These say "history size exceeds warn limit" and will appear in Temporal Cluster’s logs
  - They identify the Workflow ID, Run ID, and namespace for the Workflow Execution
- **Workflow Execution will be *terminated* after exceeding additional limits**
  - If its Event History exceeds 50K (51,200) Events
  - If its Event History exceeds 50 MB of storage

Event Structure and Characteristics

- **Every Event always contains the following three attributes**
  - ID (uniquely identifies this Event within the History)
  - Time (timestamp representing when the Event occurred)
  - Type (the kind of Event it is)

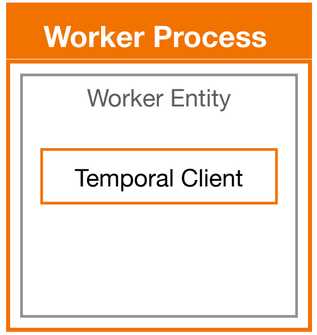
Attributes Vary by Event Type

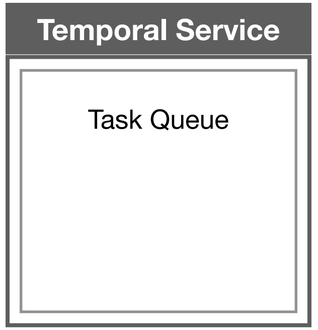
- **Additionally, each Event contains attributes specific to its type**
  - **`WorkflowExecutionStarted`** contains the Workflow Type and input parameters
  - **`WorkflowExecutionCompleted`** contains the result returned by the Workflow function
  - **`WorkflowExecutionFailed`** contains the error returned by the Workflow function
  - **`ActivityTaskScheduled`** contains the Activity Type and input parameters
  - **`ActivityTaskCompleted`** contains the result returned by the Activity function

How Commands Map to Events

pseudocode

```
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```

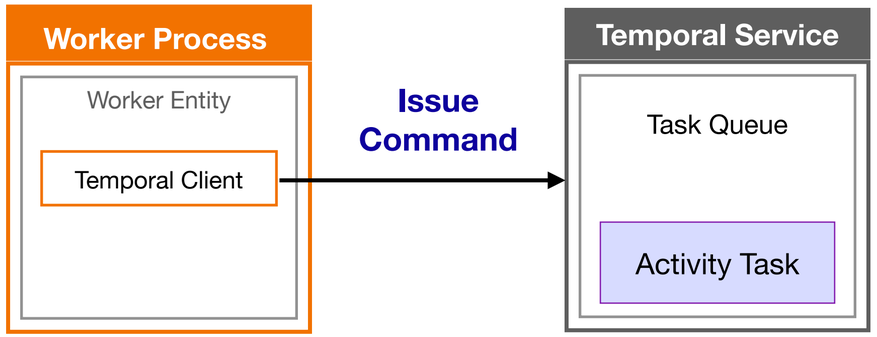




Commands

Events

```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```



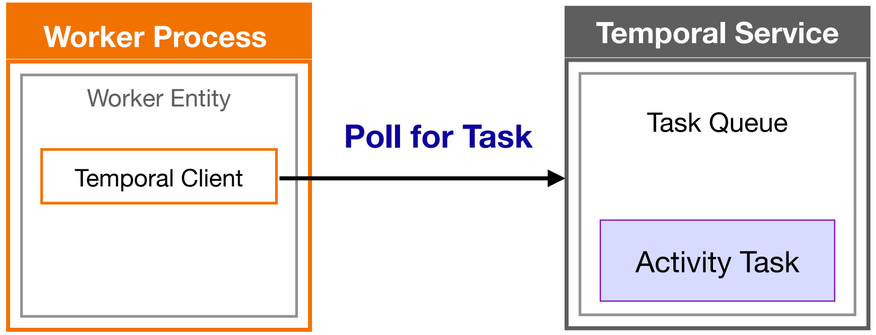
Commands

ScheduleActivityTask
(GetDistance)

Events

ActivityTaskScheduled

```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```



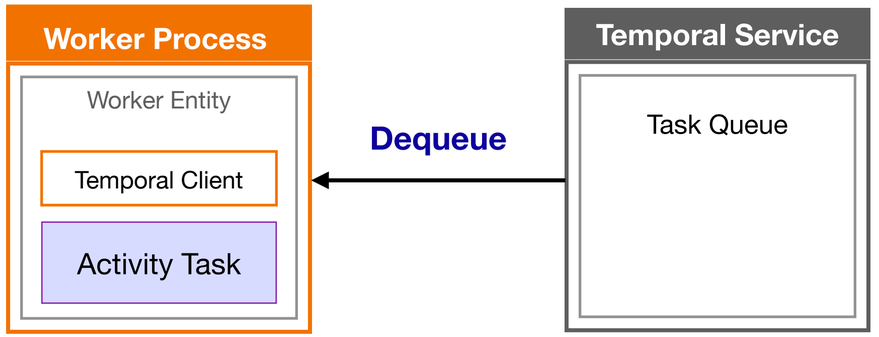
Commands

ScheduleActivityTask
(GetDistance)

Events

ActivityTaskScheduled

```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```



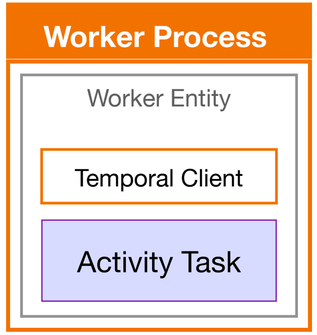
Commands

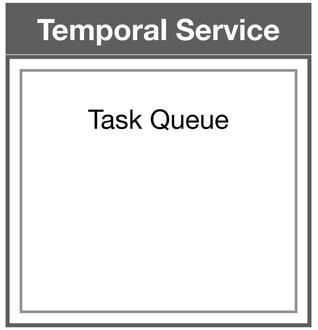
ScheduleActivityTask
(GetDistance)

Events

ActivityTaskScheduled

```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```





Commands

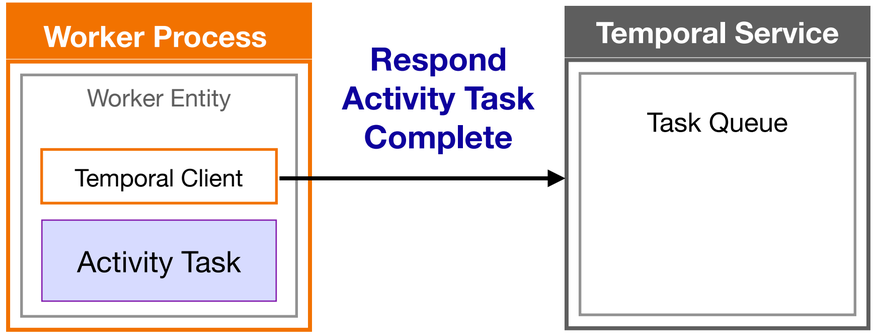
ScheduleActivityTask
(GetDistance)

Events

ActivityTaskScheduled

ActivityTaskStarted

```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```



Commands

ScheduleActivityTask
(GetDistance)

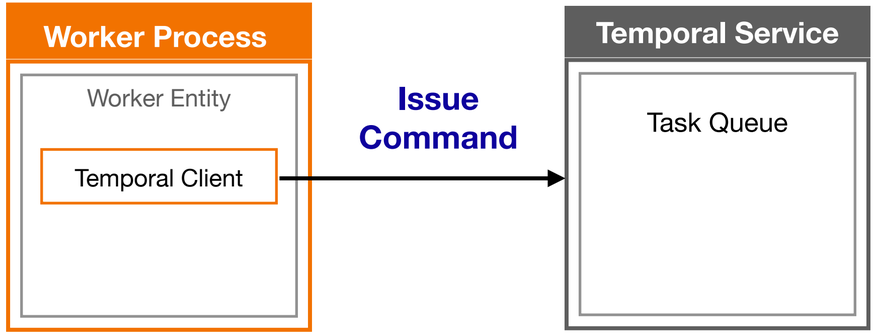
Events

ActivityTaskScheduled

ActivityTaskStarted

ActivityTaskCompleted

```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```



Commands

ScheduleActivityTask
(GetDistance)

StartTimer
(30 Minutes)

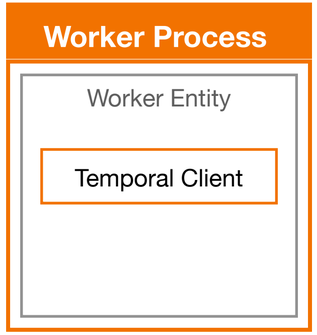
Events

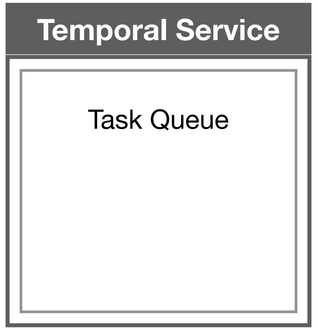
ActivityTaskScheduled

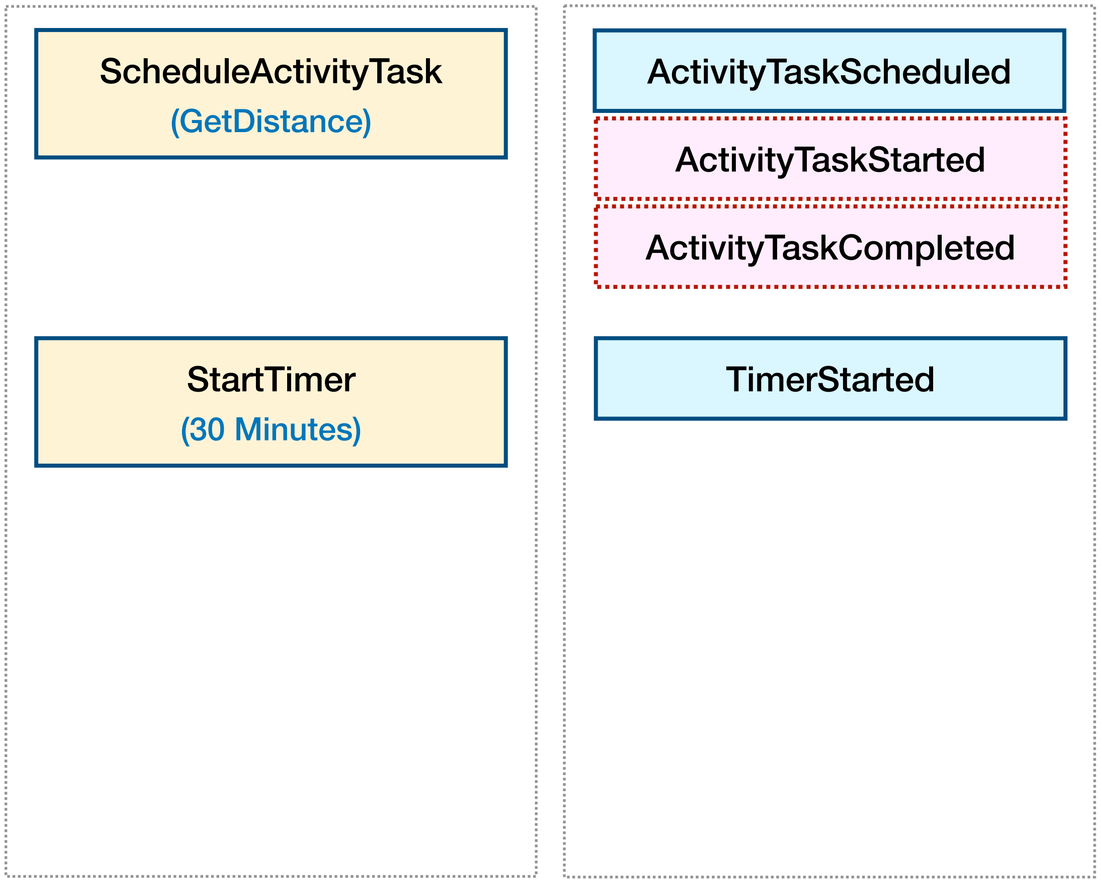
ActivityTaskStarted

ActivityTaskCompleted

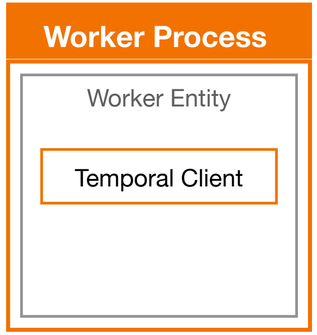
```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```

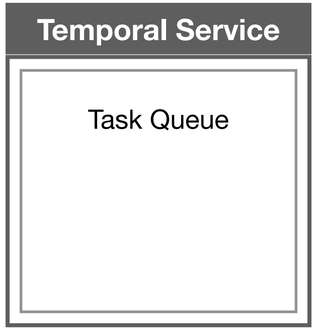


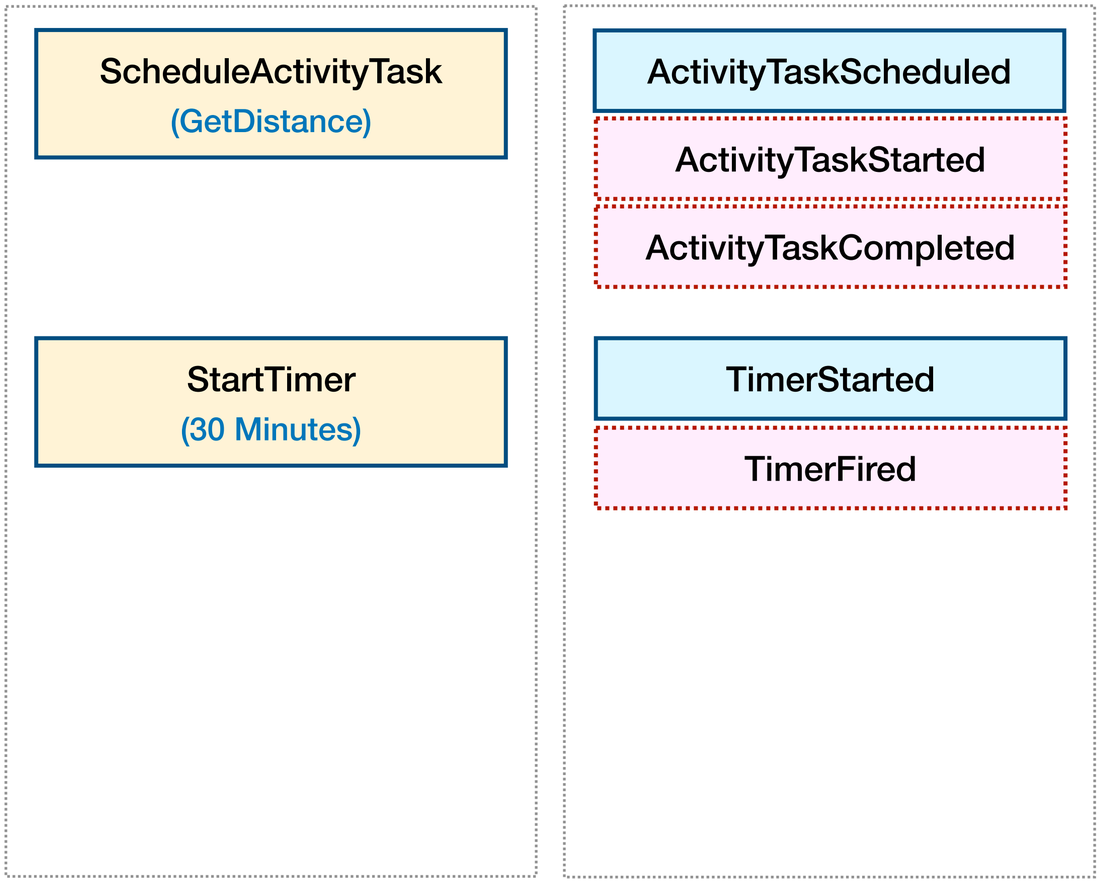




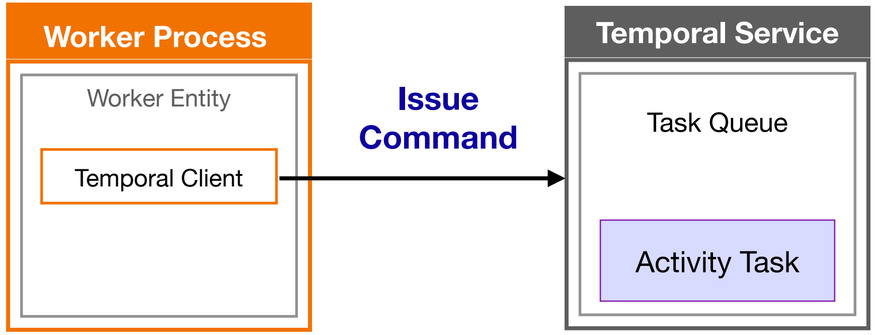
```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```







```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```



Commands

ScheduleActivityTask
(GetDistance)

StartTimer
(30 Minutes)

ScheduleActivityTask
(SendBill)

Events

ActivityTaskScheduled

ActivityTaskStarted

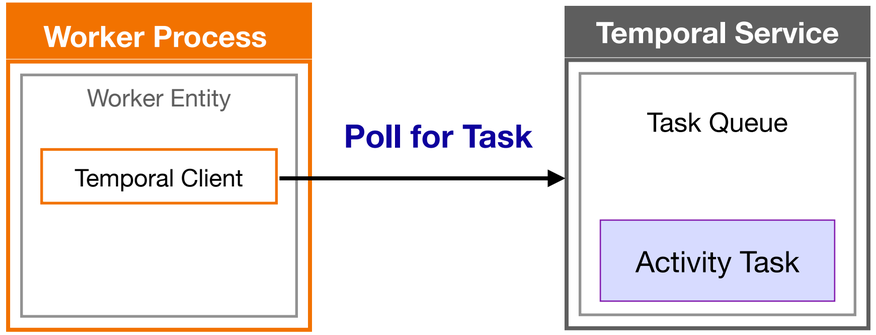
ActivityTaskCompleted

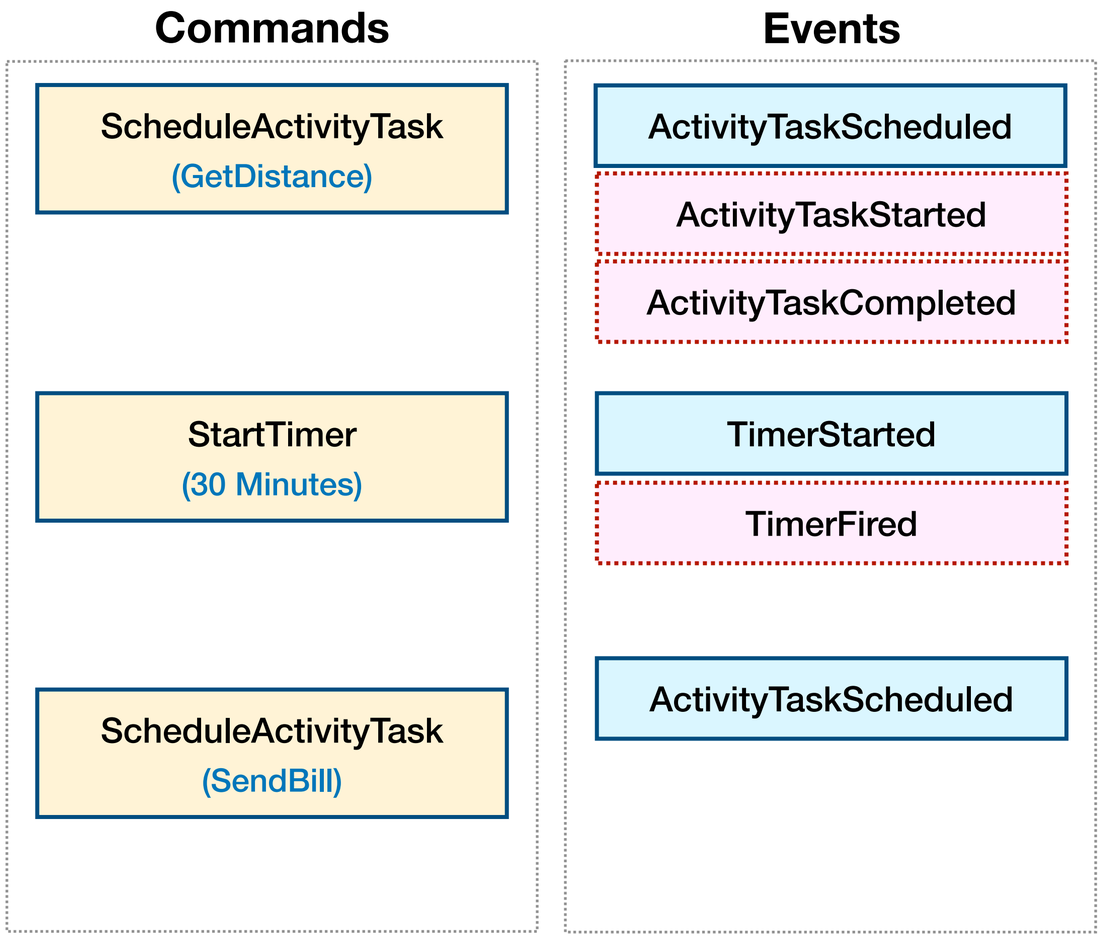
TimerStarted

TimerFired

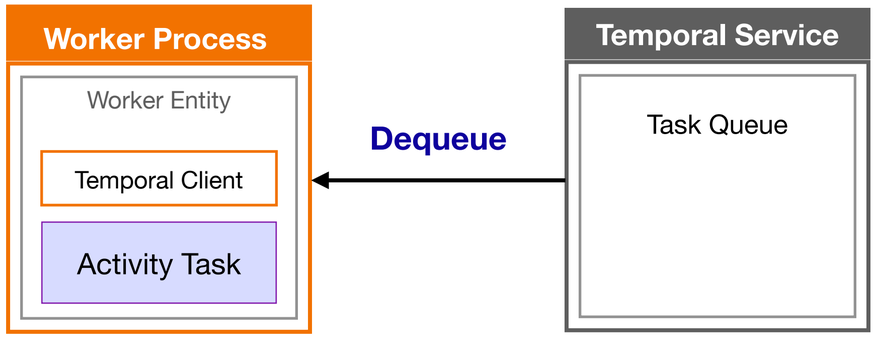
ActivityTaskScheduled

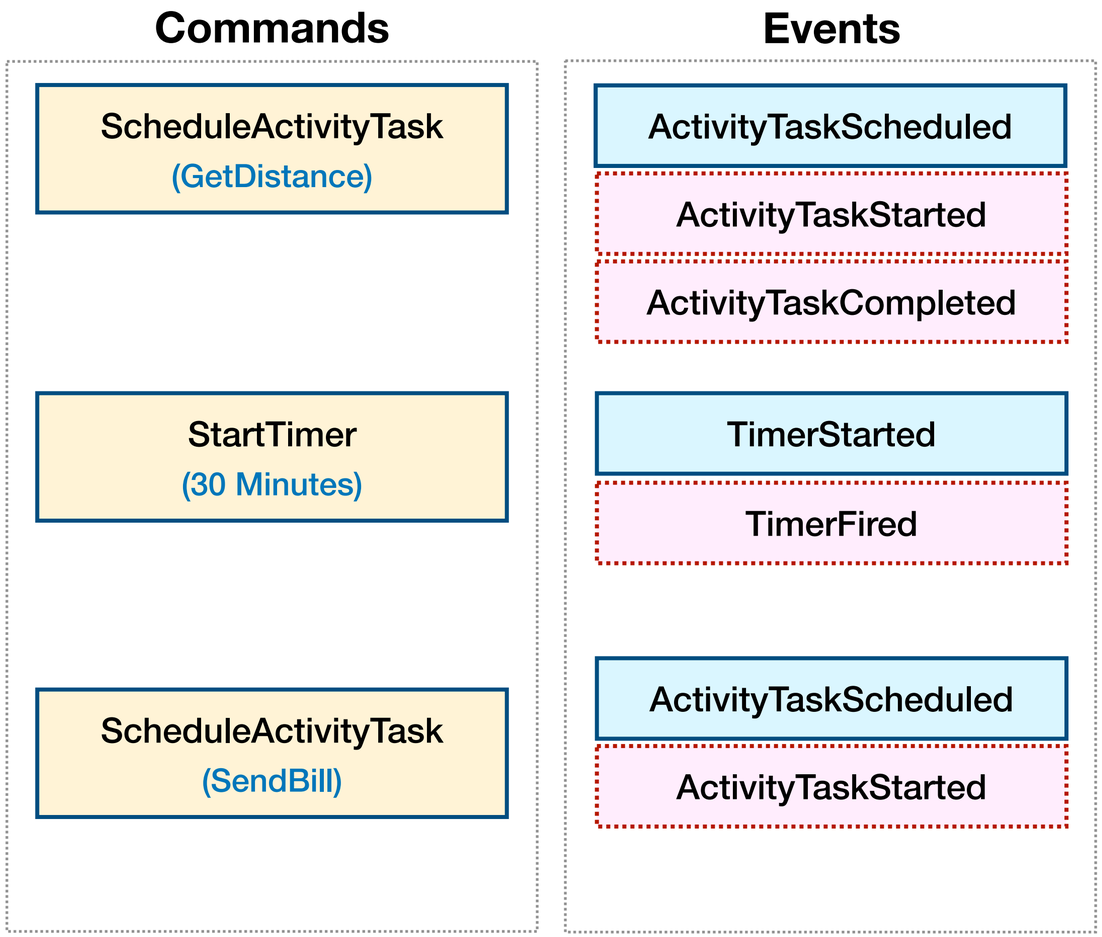
```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```



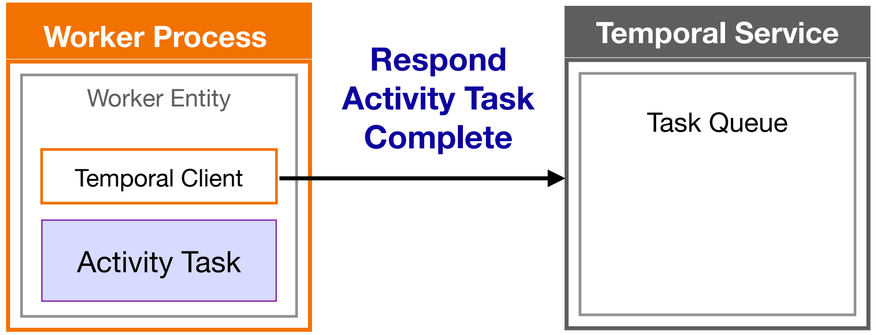


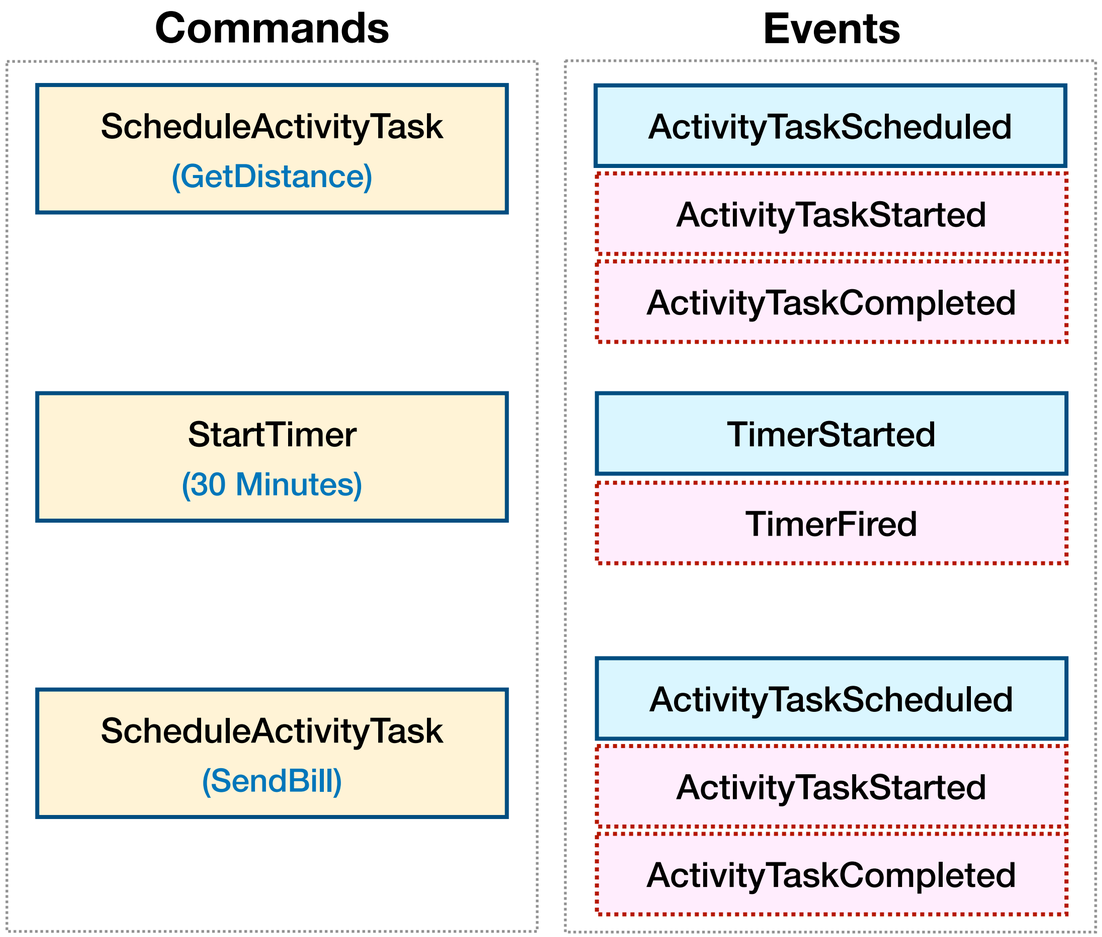
```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```



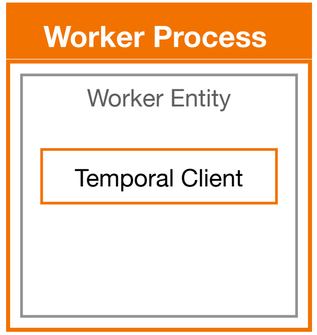


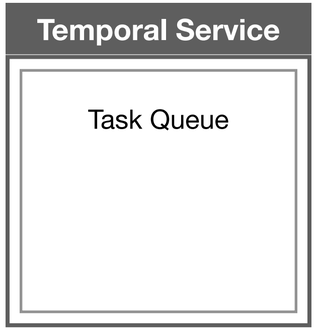
```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```

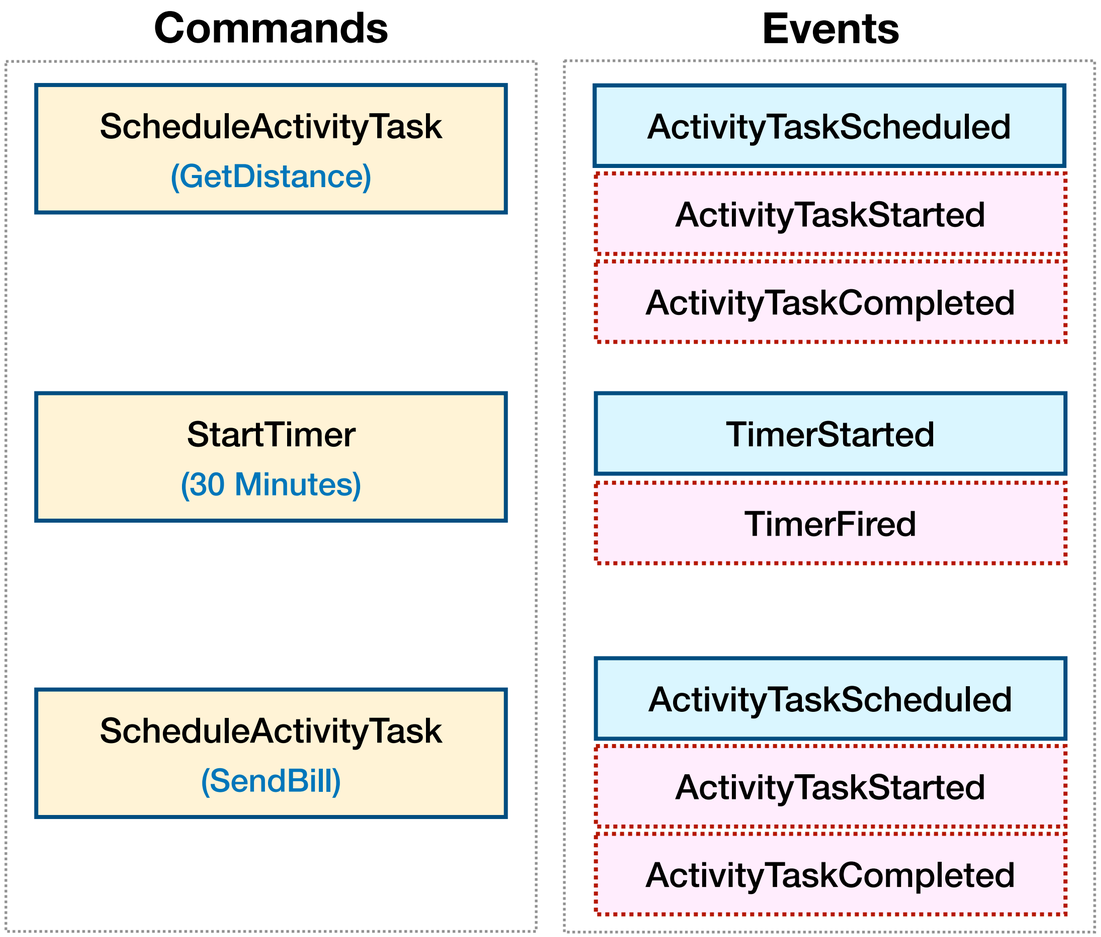




```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   options := workflow.ActivityOptions{
      StartToCloseTimeout: time.Second * 5,
   }
   ctx = workflow.WithActivityOptions(ctx, options)

   // Iterate over the items and calculate the cost of the order
   var totalPrice int
   for pizza : order.Items {
      totalPrice += pizza.Price
   }

   distance := workflow.ExecuteActivity(ctx, GetDistance, order.Address)

   if order.IsDelivery && distance > 25 {
      return "", errors.New("customer too far away for delivery")
   }

   // Wait 30 minutes before billing the customer
   workflow.Sleep(ctx, time.Minute * 30)

   bill := Bill{
      CustomerId:  order.Customer.CustomerId,
      Amount:      totalPrice,
      Description: order.OrderNumber,
   }

   confirmation := workflow.ExecuteActivity(ctx, SendBill, bill)

   return confirmation, nil
}
```

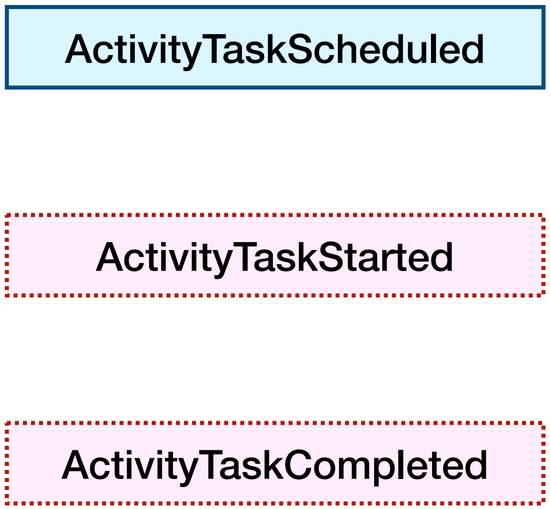




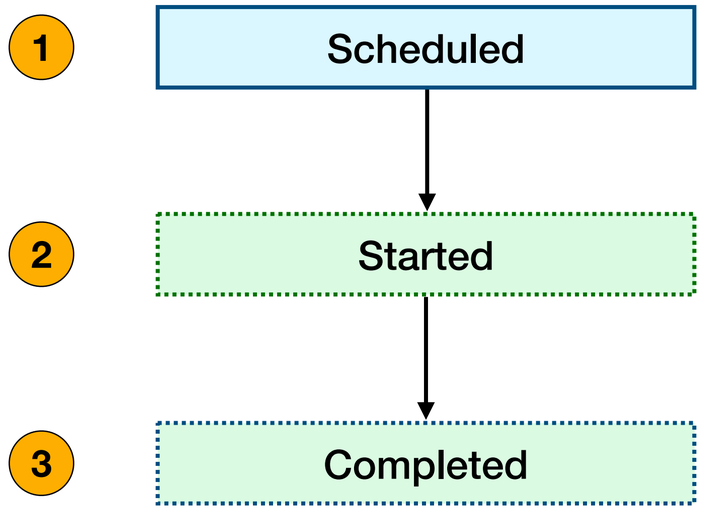


Workflow and Activity Task States

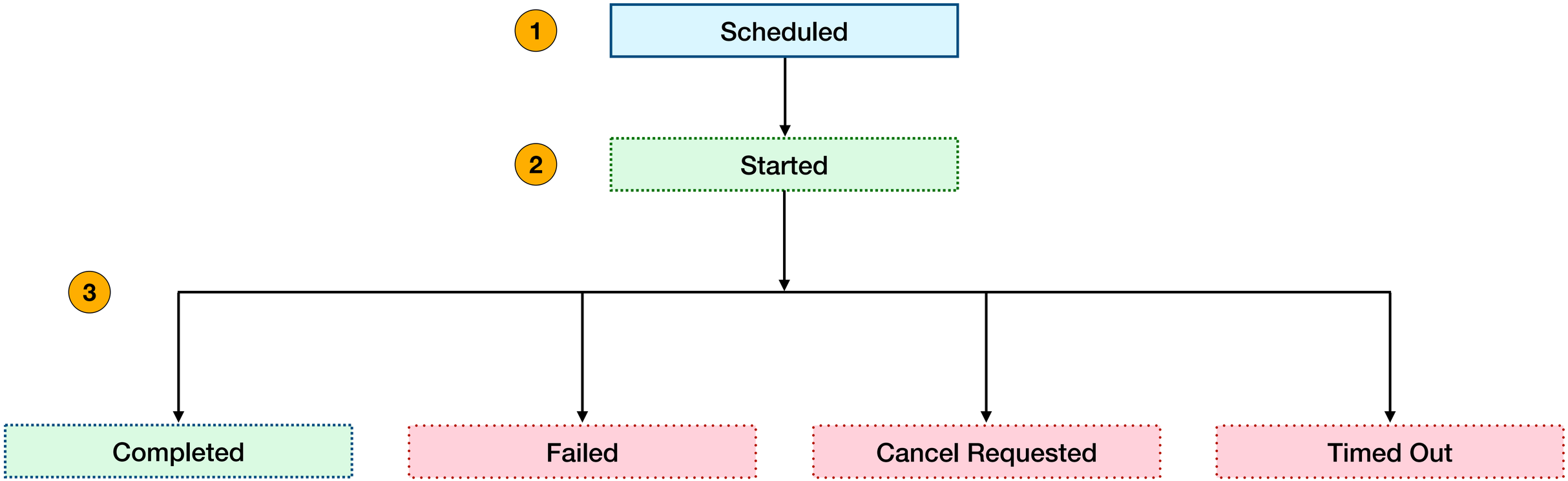
Activity Task Event Sequence



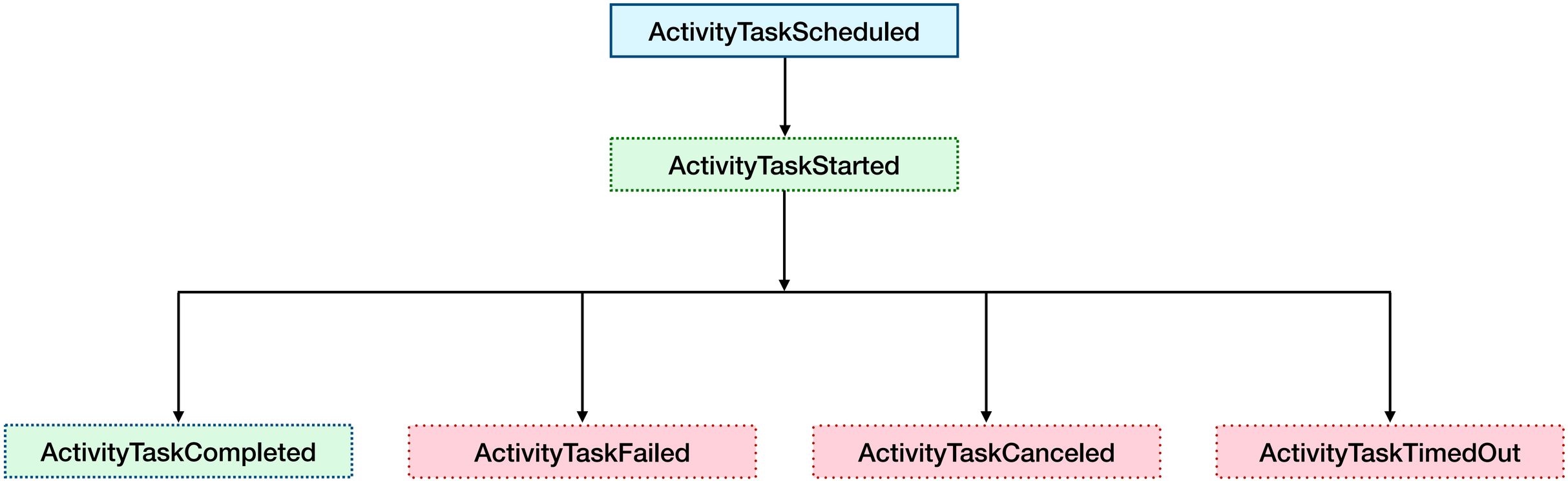
Activity States in that Sequence



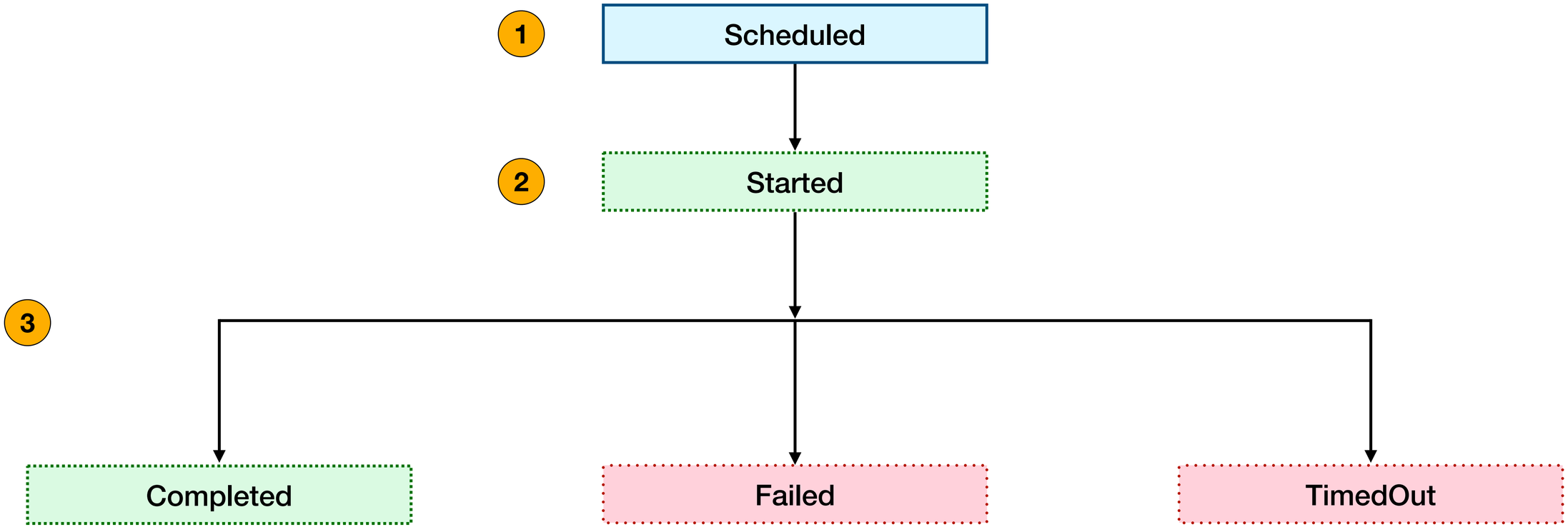
Activity Task States



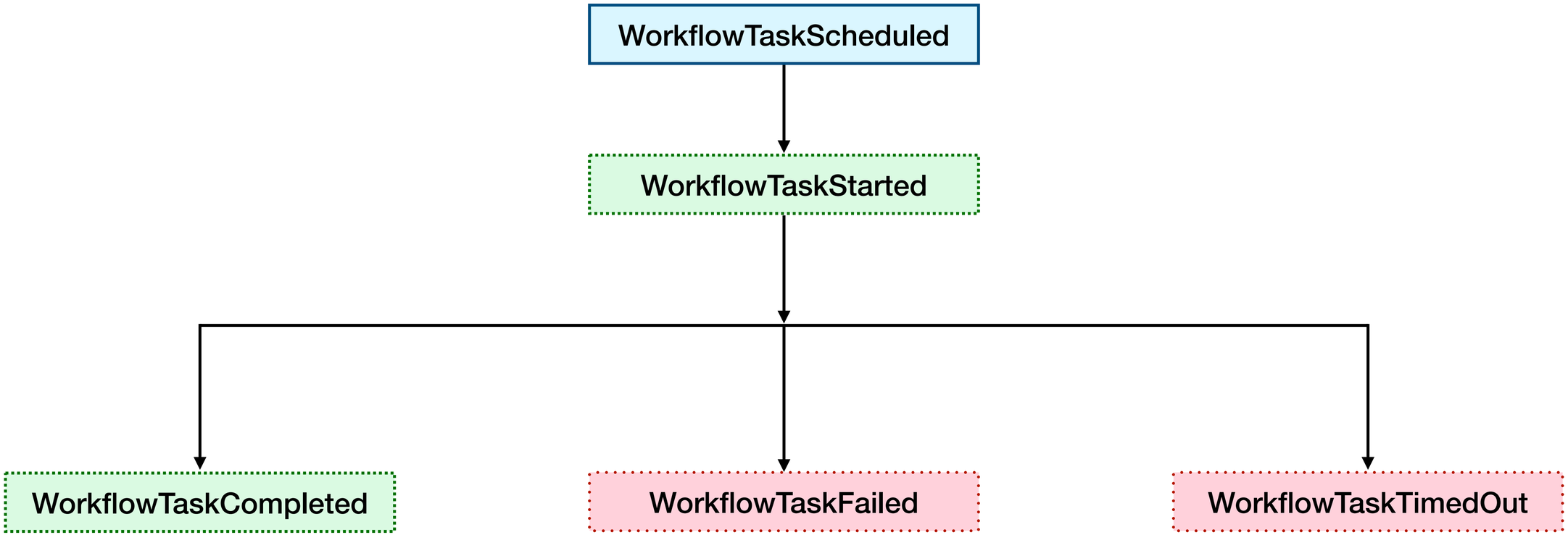
Activity Task Events



Workflow Task States



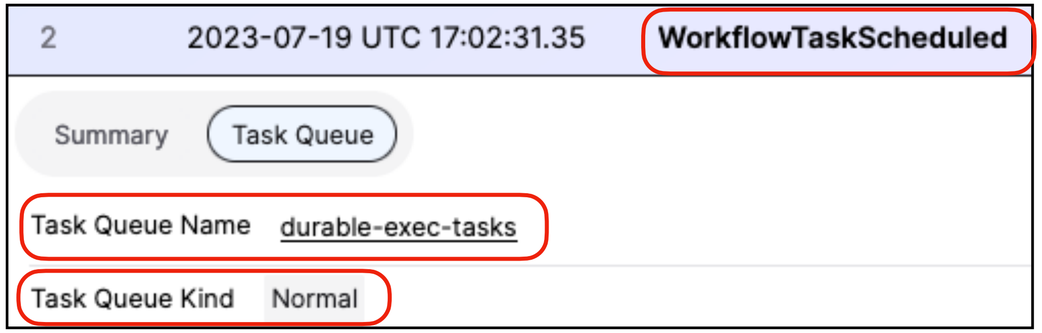
Workflow Task Events



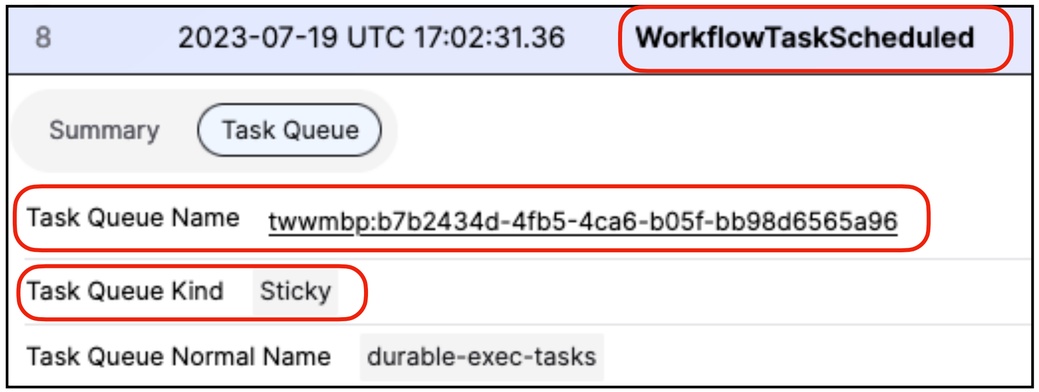
Sticky Execution

- **To improve effectiveness of Worker's caching, Temporal use "sticky" execution for Workflow Tasks**
  - A Worker which completed the first Workflow Task is given preference for subsequent Workflow Tasks in the same execution via a Worker-specific Task Queue
- **Sticky execution is visible in the Web UI**
  - See the Task Queue Name / Kind fields
- **This does not apply to Activity Tasks**

First Workflow Task



Later Workflow Task



Beginner to Builder Bootcamp - Go

History Replay:

How Temporal Provides Durable Execution

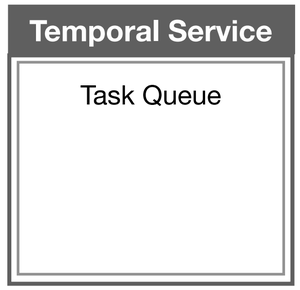
```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```

Start Workflow Execution

```go
client.ExecuteWorkflow(ctx, options, PizzaWorkflow, input)
```

```
[
    {
      "OrderNumber": "Z1238",
      "Customer": {
        "CustomerID": 12983,
        "Name": "María García",
        "Email": "maria1985@example.com",
        "Phone": "415-555-7418"
      },
      "Items": [
        {
          "Description": "Large, with pepperoni",
          "Price": 1500
        },
        {
          "Description": "Small, with mushrooms and onions",
          "Price": 1000
        }
      ],
      "IsDelivery": true,
      "Address": {
        "Line1": "701 Mission Street",
        "Line2": "Apartment 9C",
        "City": "San Francisco",
        "State": "CA",
        "PostalCode": "94103"
      }
    }
]
```

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```

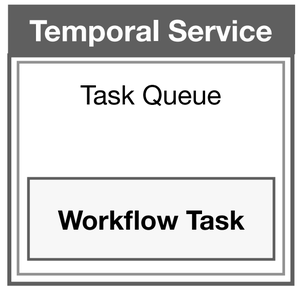


Commands

Events

WorkflowExecutionStarted

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



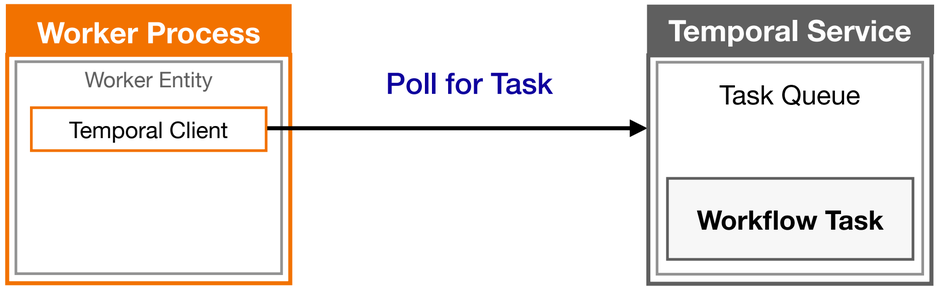
Commands

Events

WorkflowExecutionStarted

WorkflowTaskScheduled

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```

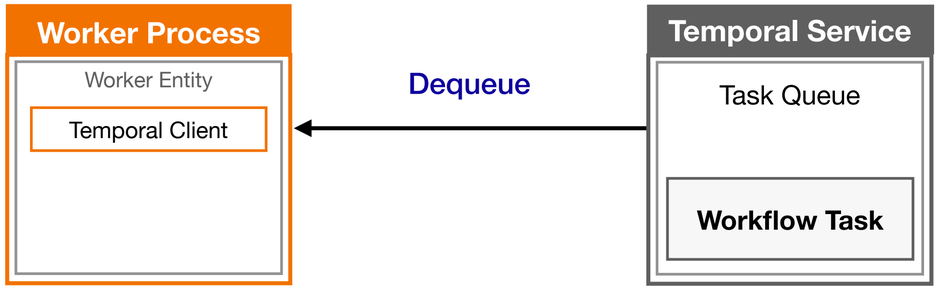


Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```

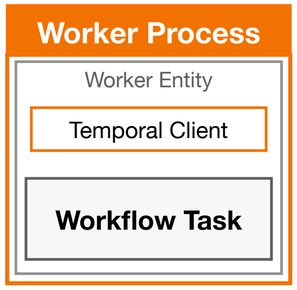


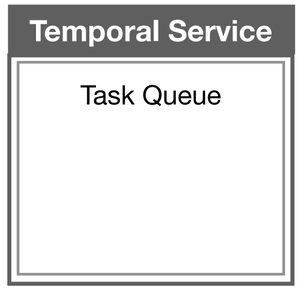
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



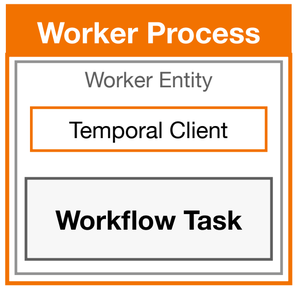


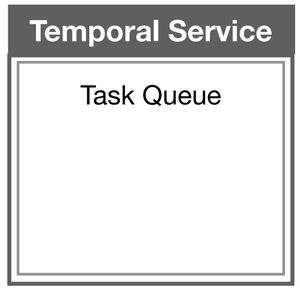
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



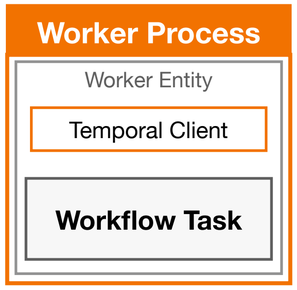


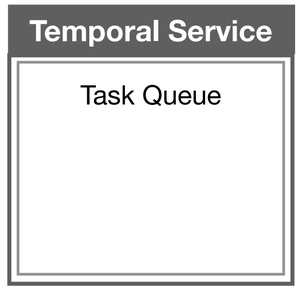
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



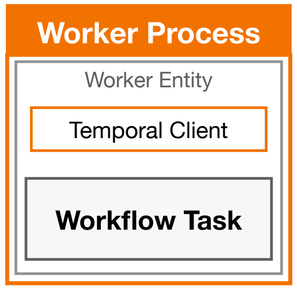


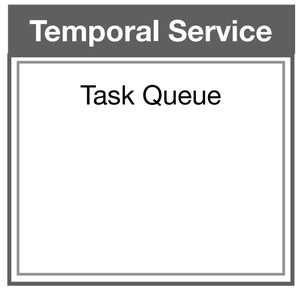
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



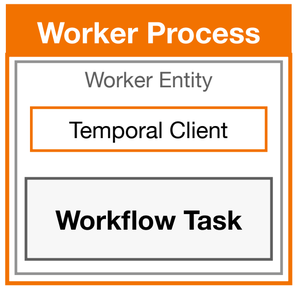


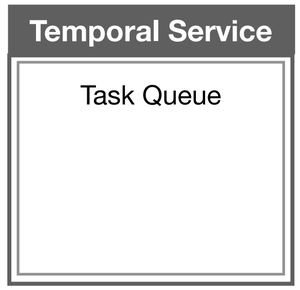
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



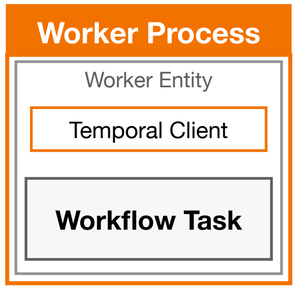


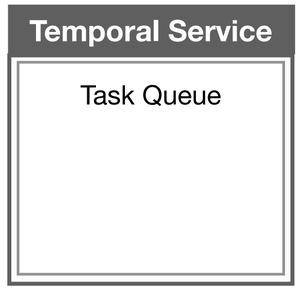
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



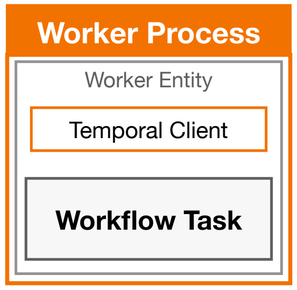


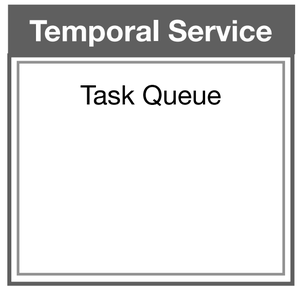
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



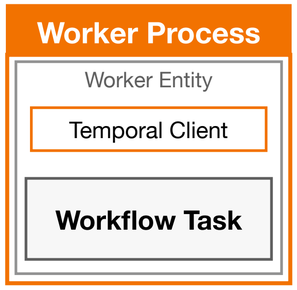


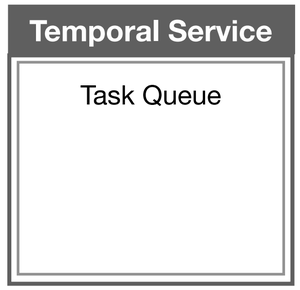
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



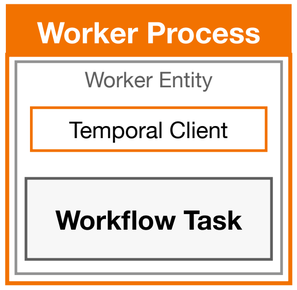


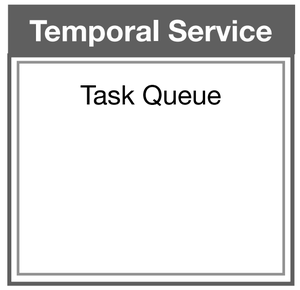
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



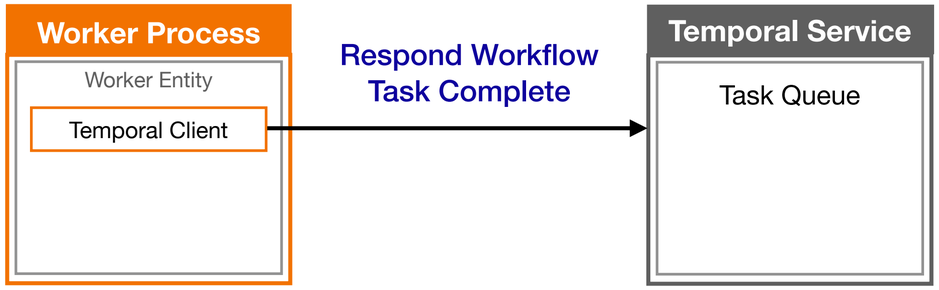


Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```

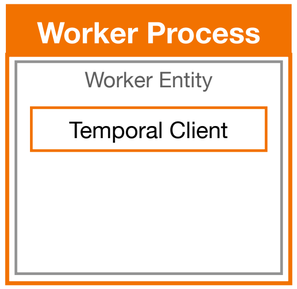


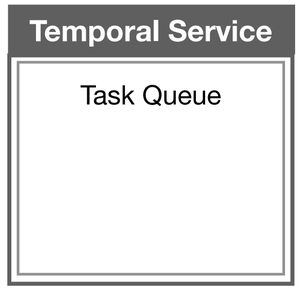
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



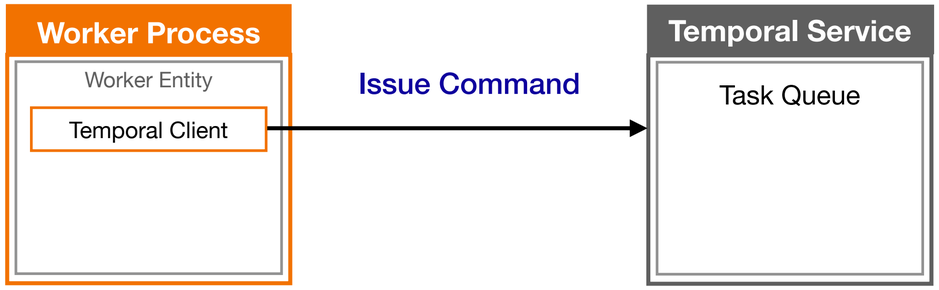


Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

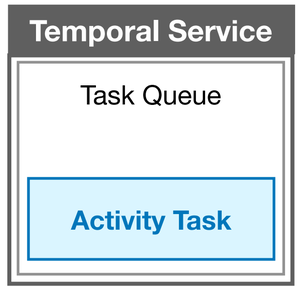
ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

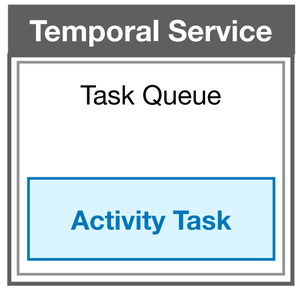
ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(GetDistance)**

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

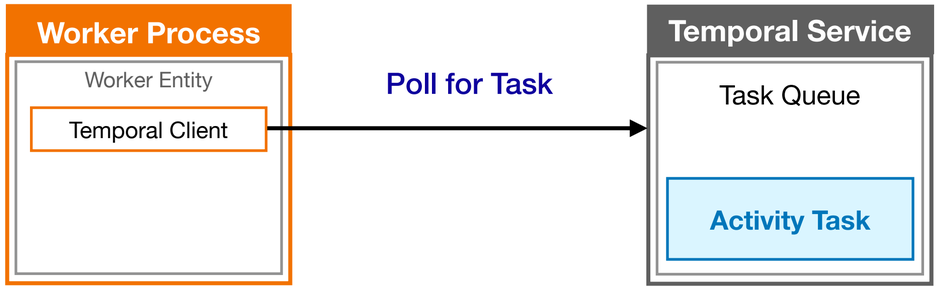
ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(GetDistance)**

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

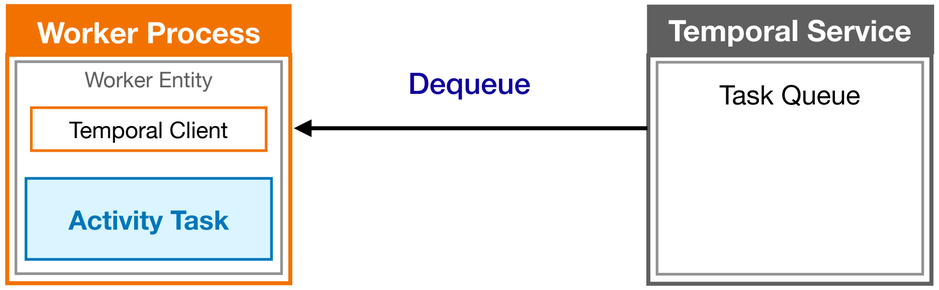
ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(GetDistance)**

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

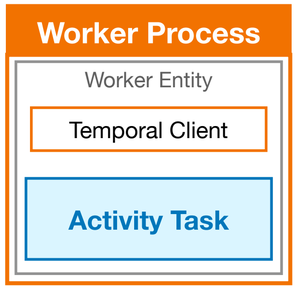
ScheduleActivityTask

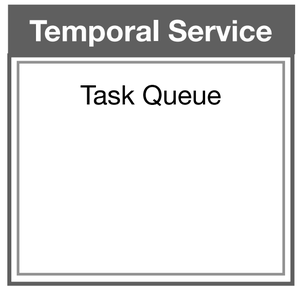
Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(GetDistance)**
ActivityTaskStarted

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

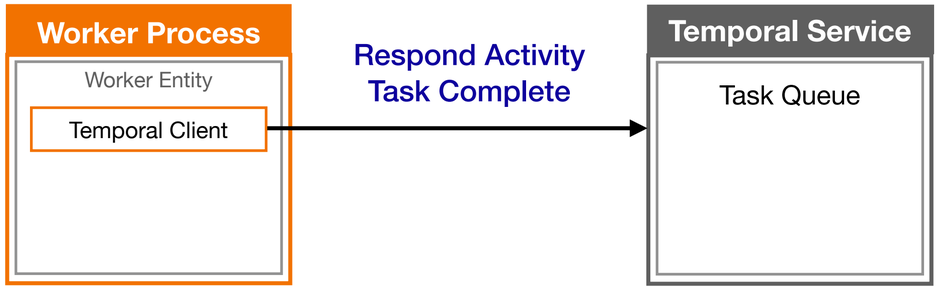
ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(GetDistance)**
ActivityTaskStarted

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

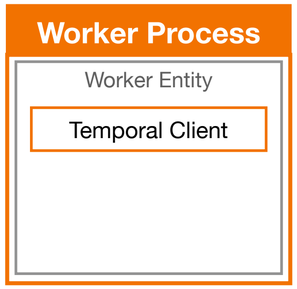
ScheduleActivityTask

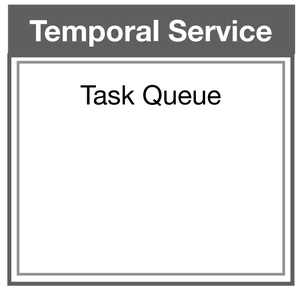
Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(GetDistance)**
ActivityTaskStarted

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

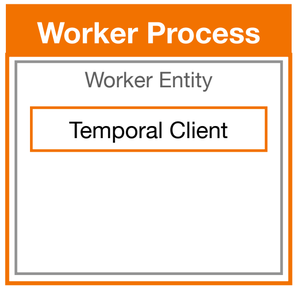
ScheduleActivityTask

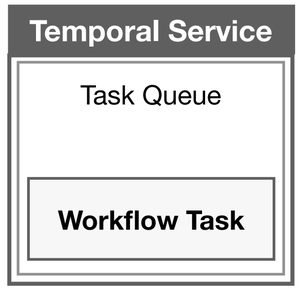
Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(GetDistance)**
ActivityTaskStarted
ActivityTaskCompleted **(distance=15)**

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

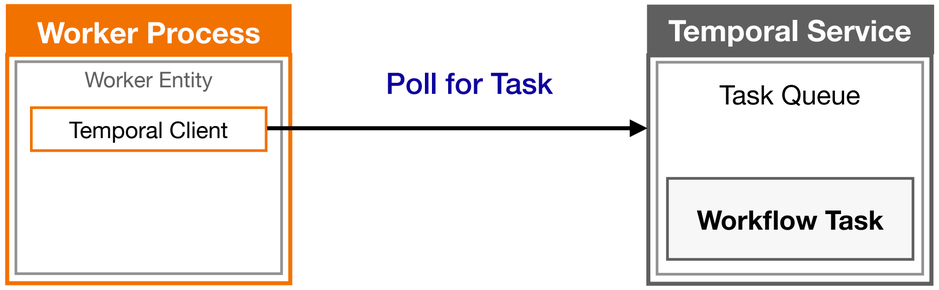
ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

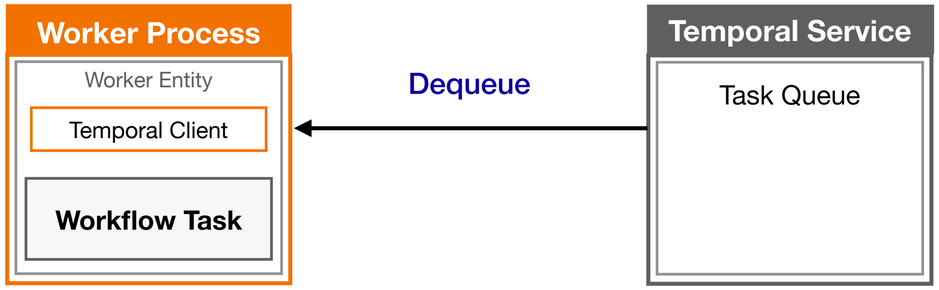
ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

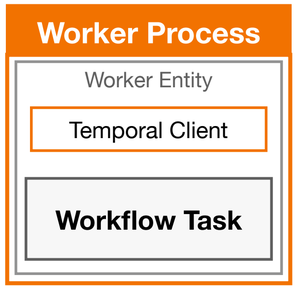
ScheduleActivityTask

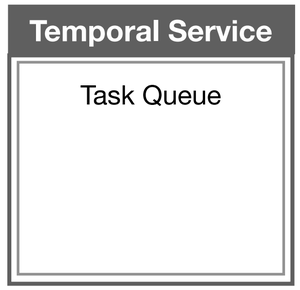
Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

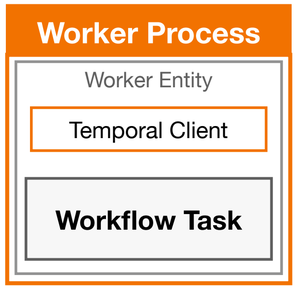
ScheduleActivityTask

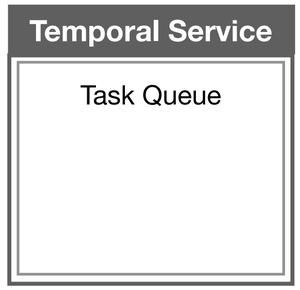
Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

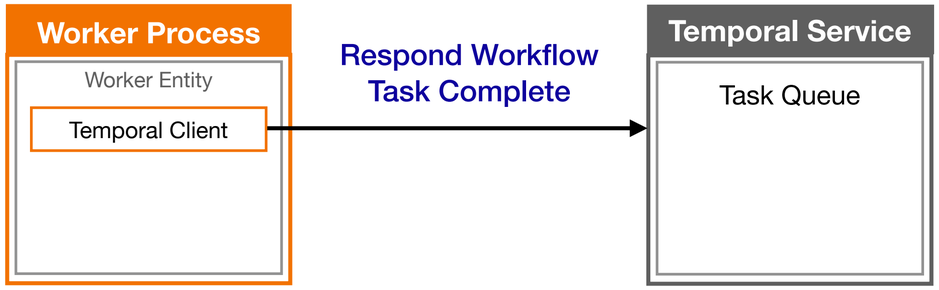
ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

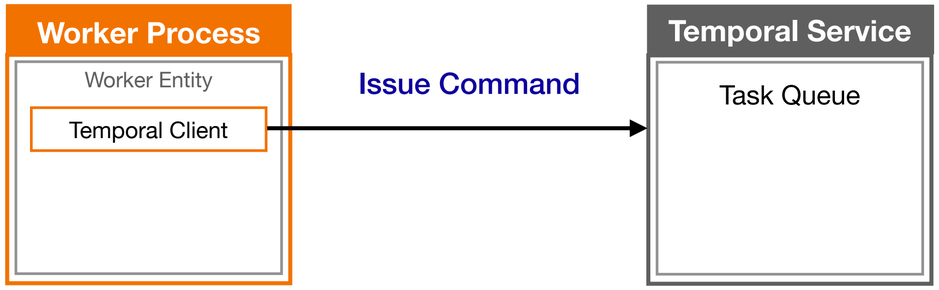
ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

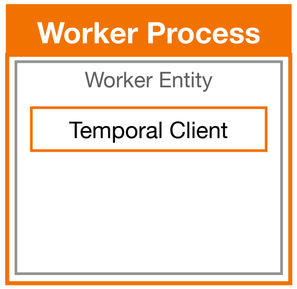
StartTimer

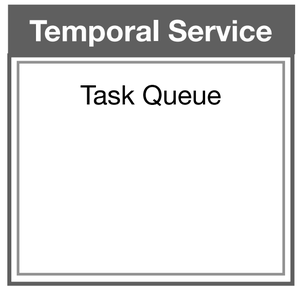
Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

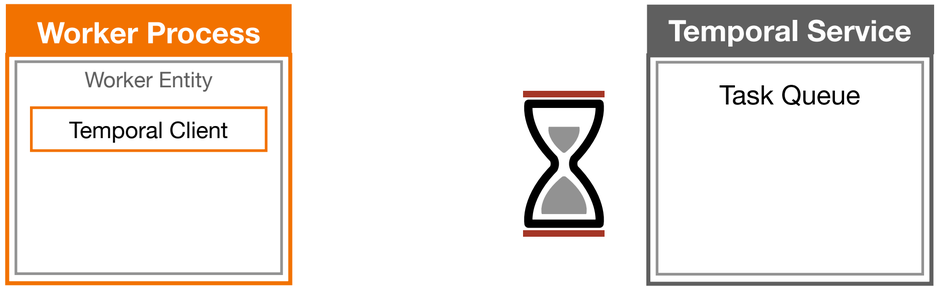
StartTimer

Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

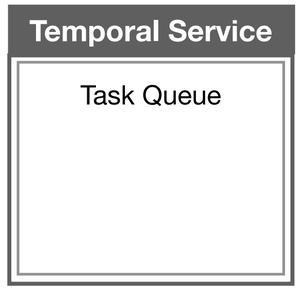
StartTimer

Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

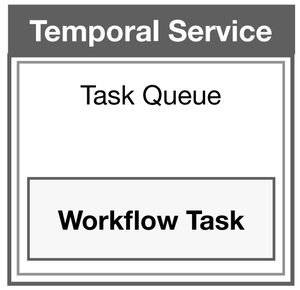
StartTimer

Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

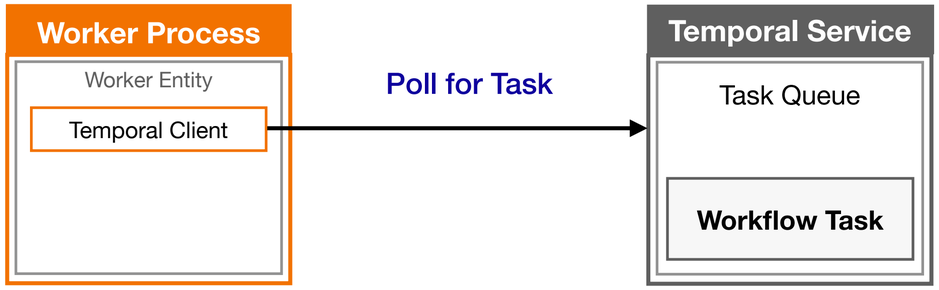
StartTimer

Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

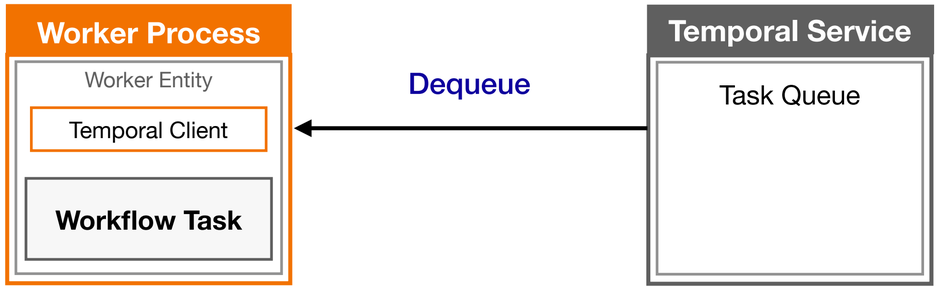
StartTimer

Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

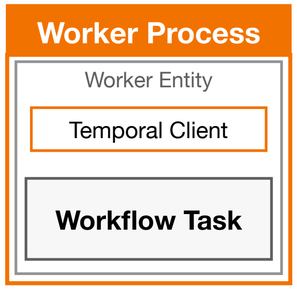
StartTimer

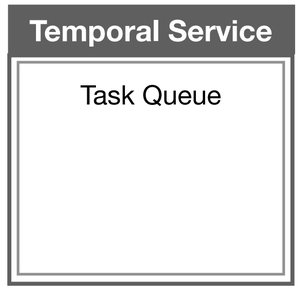
Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

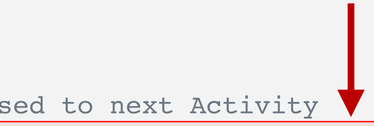
Duration: 30 minutes

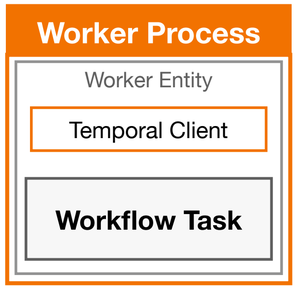
Events

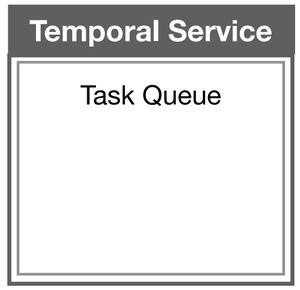
WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```

Worker crashes here







Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

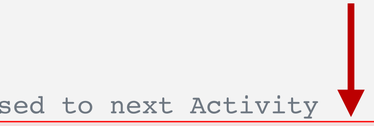
Duration: 30 minutes

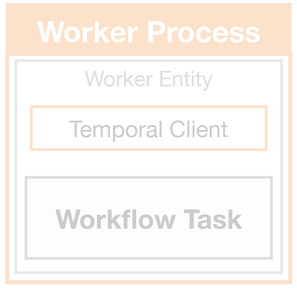
Events

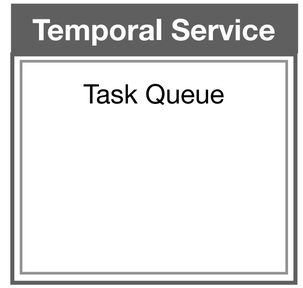
WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```

Worker crashes here





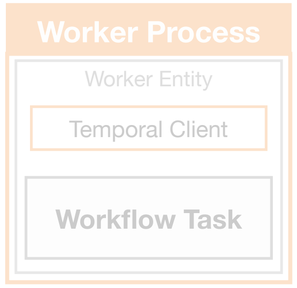


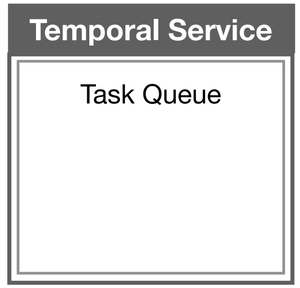
Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```

Worker Process

Workflow Task Timeout Exceeded

Temporal Service

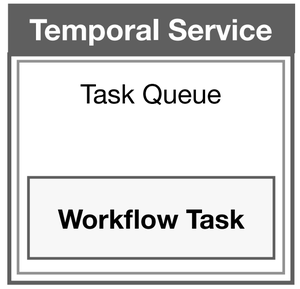
Task Queue

Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```

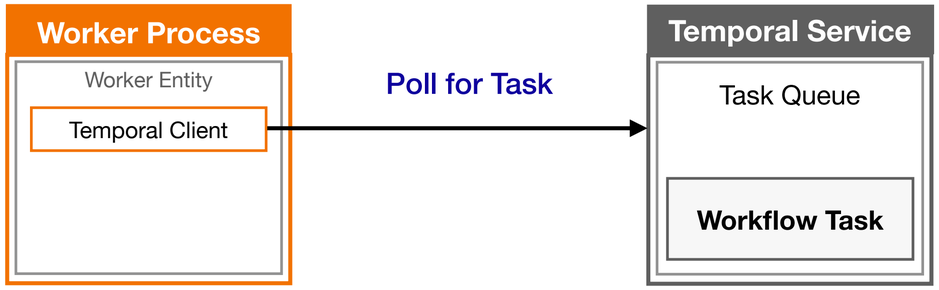


Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```

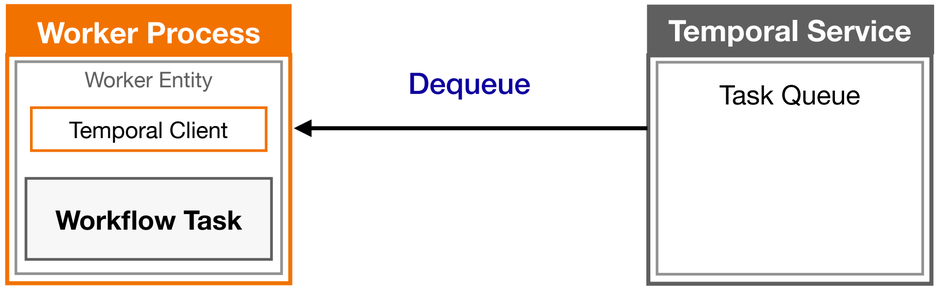


Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

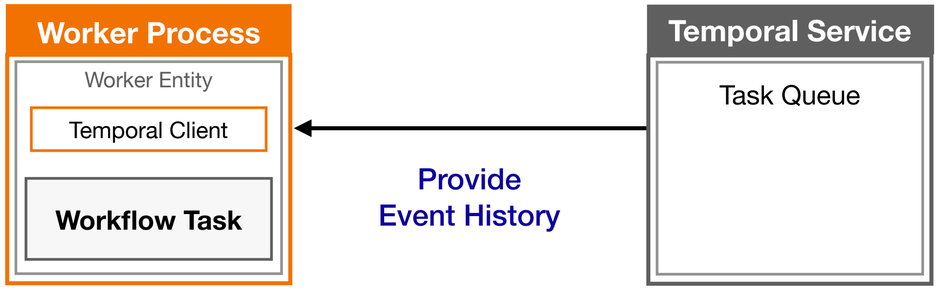
```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```

Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```

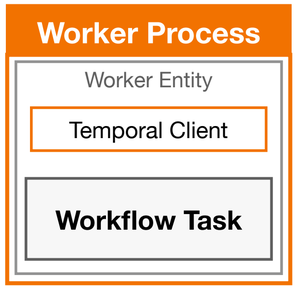


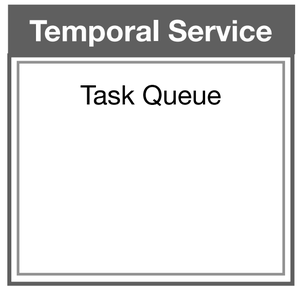
Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



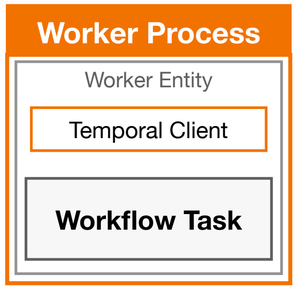


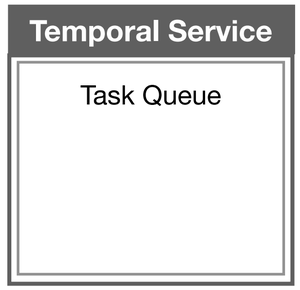
Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



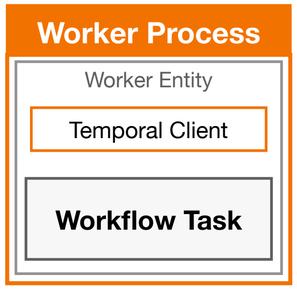


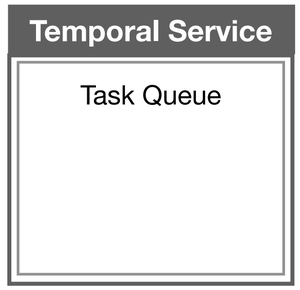
Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



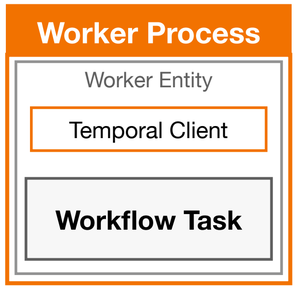


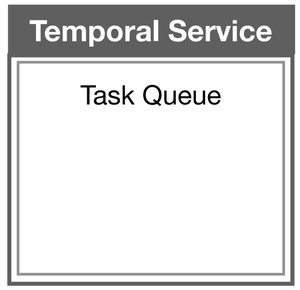
Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



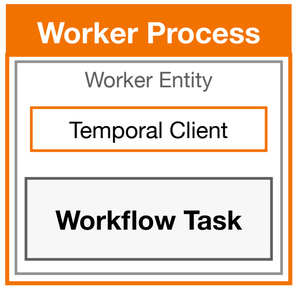


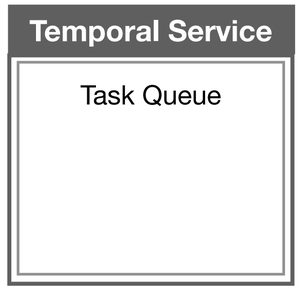
Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



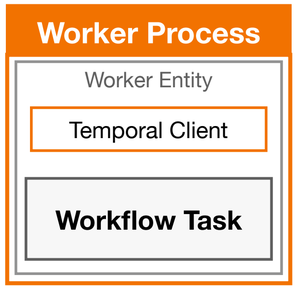


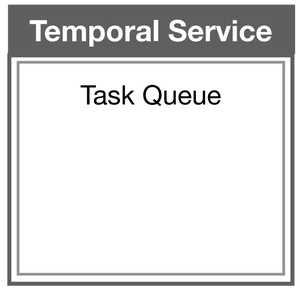
Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



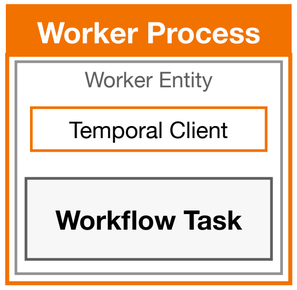


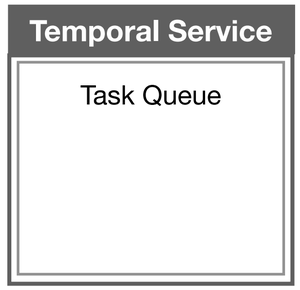
Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



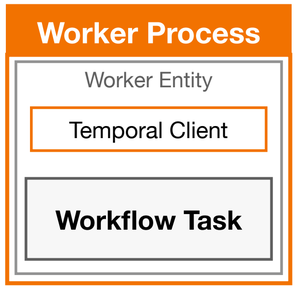


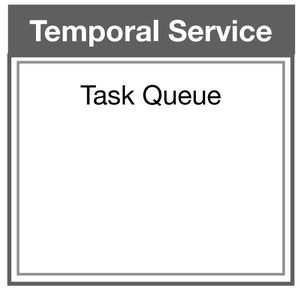
Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



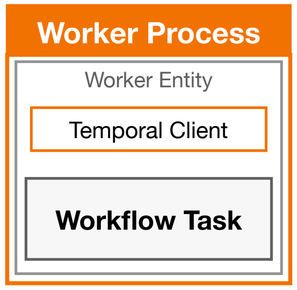


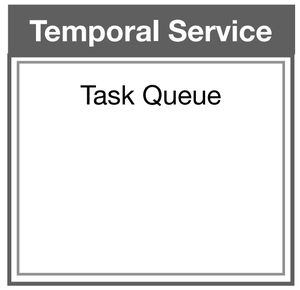
Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

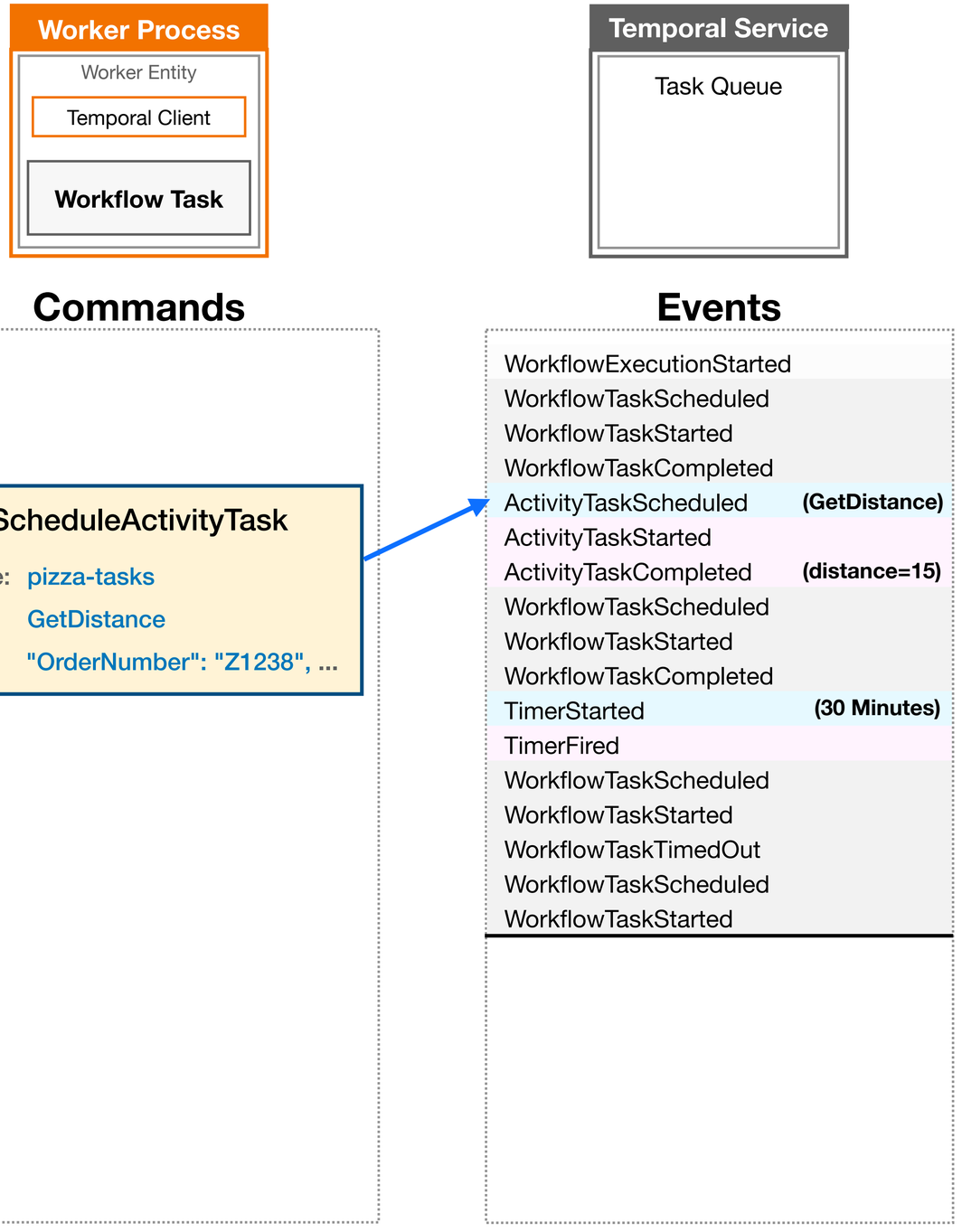
ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

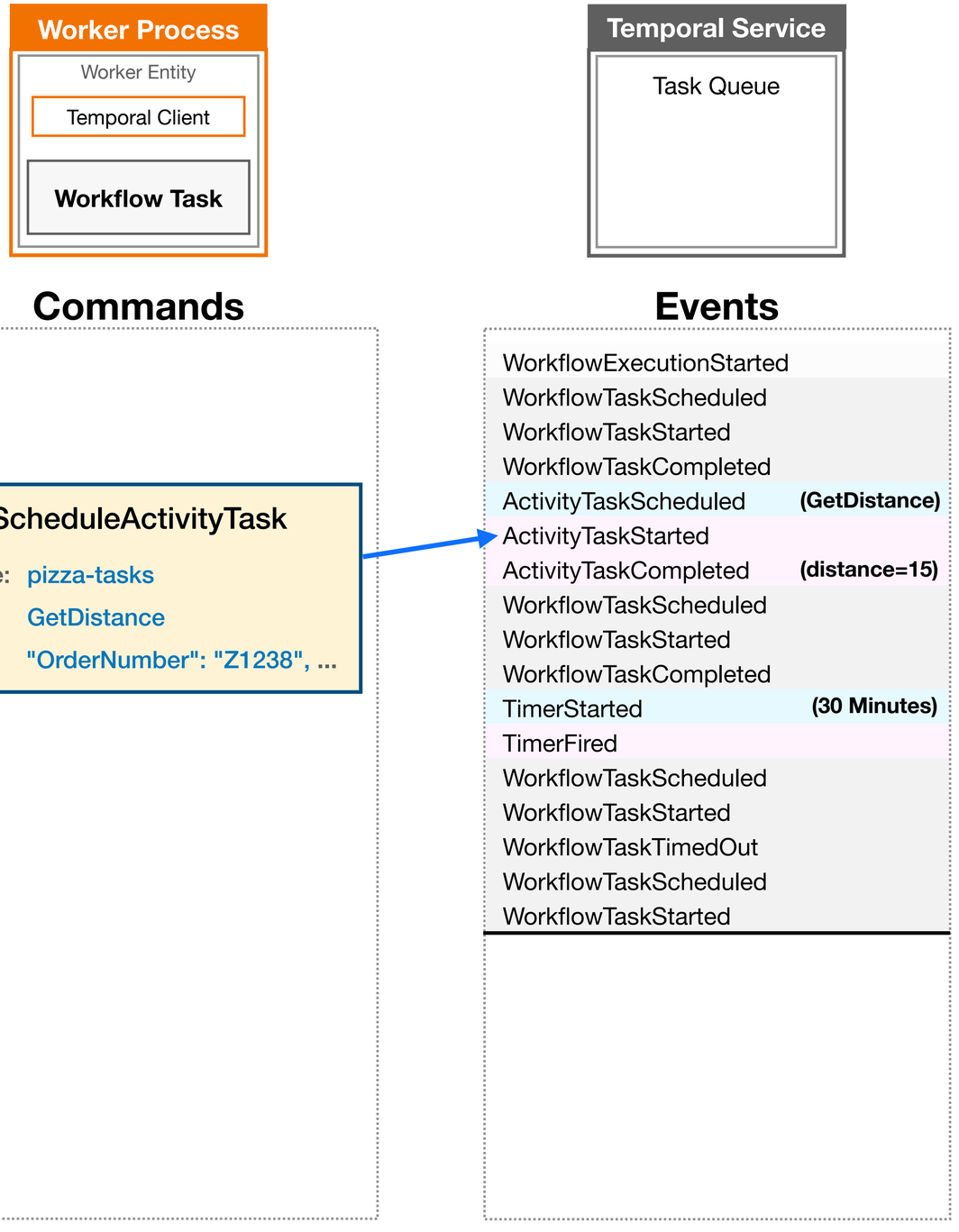
Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

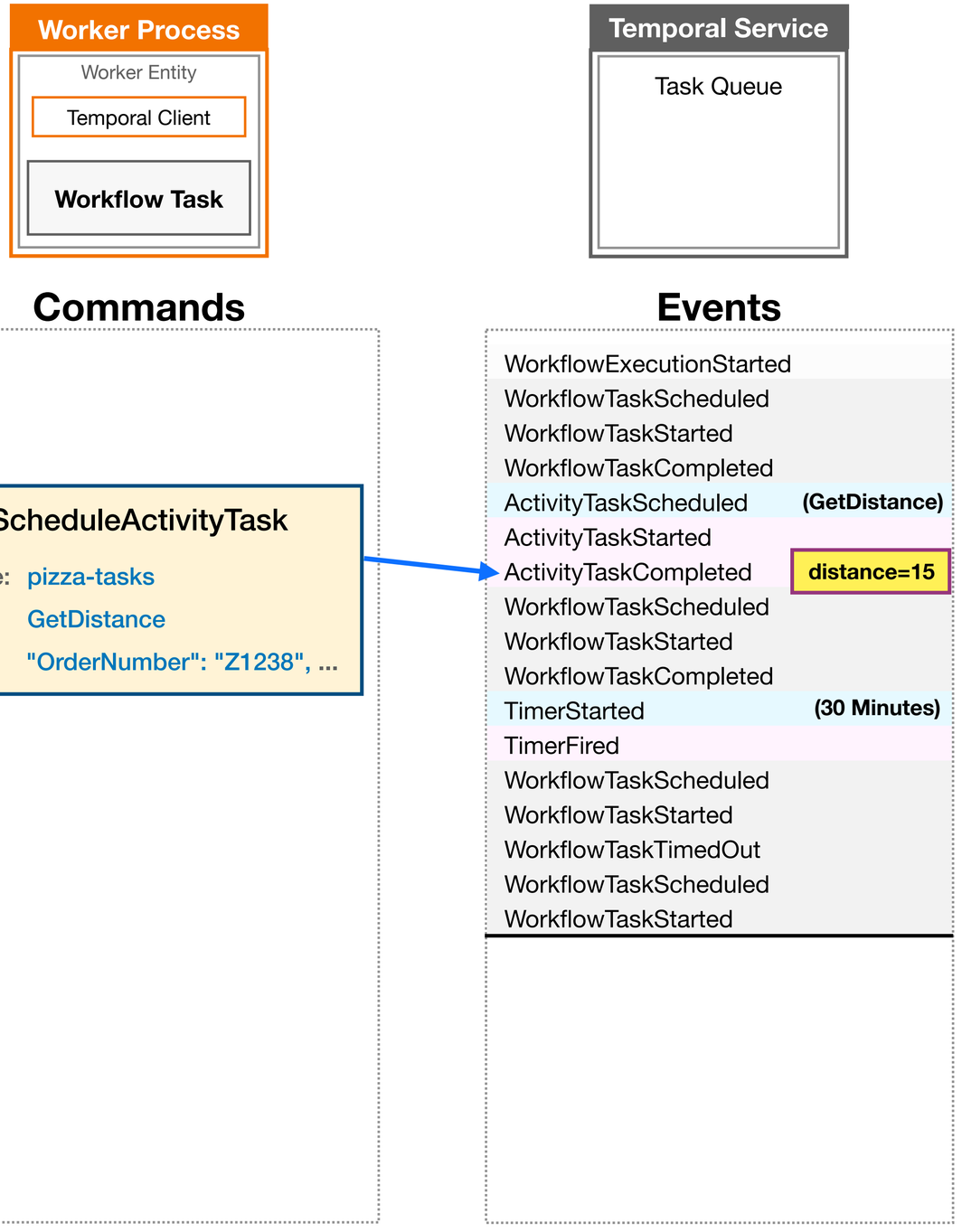
```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```

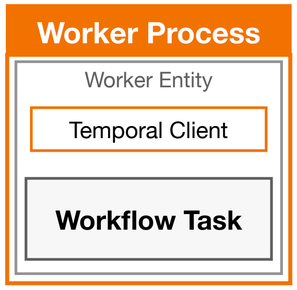


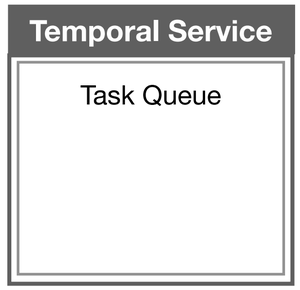
```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)
    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```

Worker assigns 15 to this variable





Commands

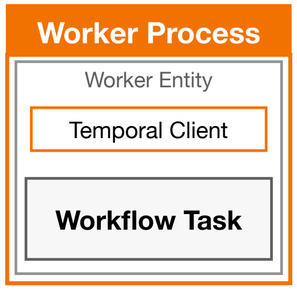
ScheduleActivityTask

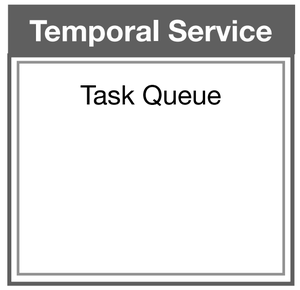
Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

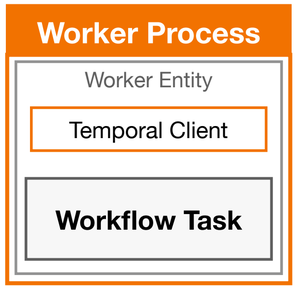
ScheduleActivityTask

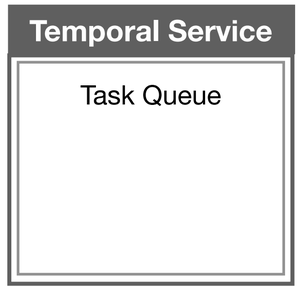
Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

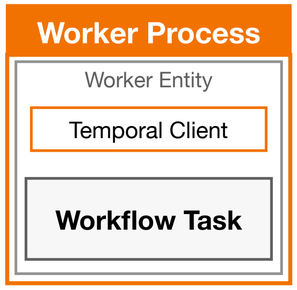
ScheduleActivityTask

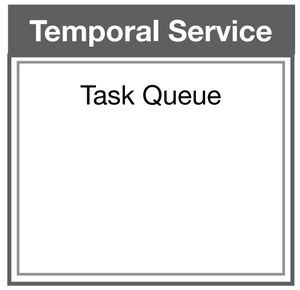
Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

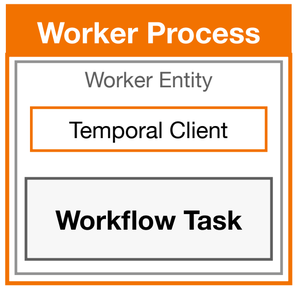
ScheduleActivityTask

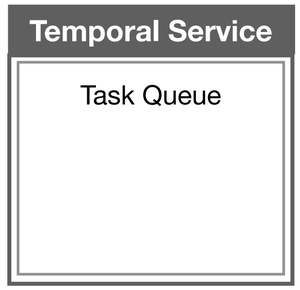
Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

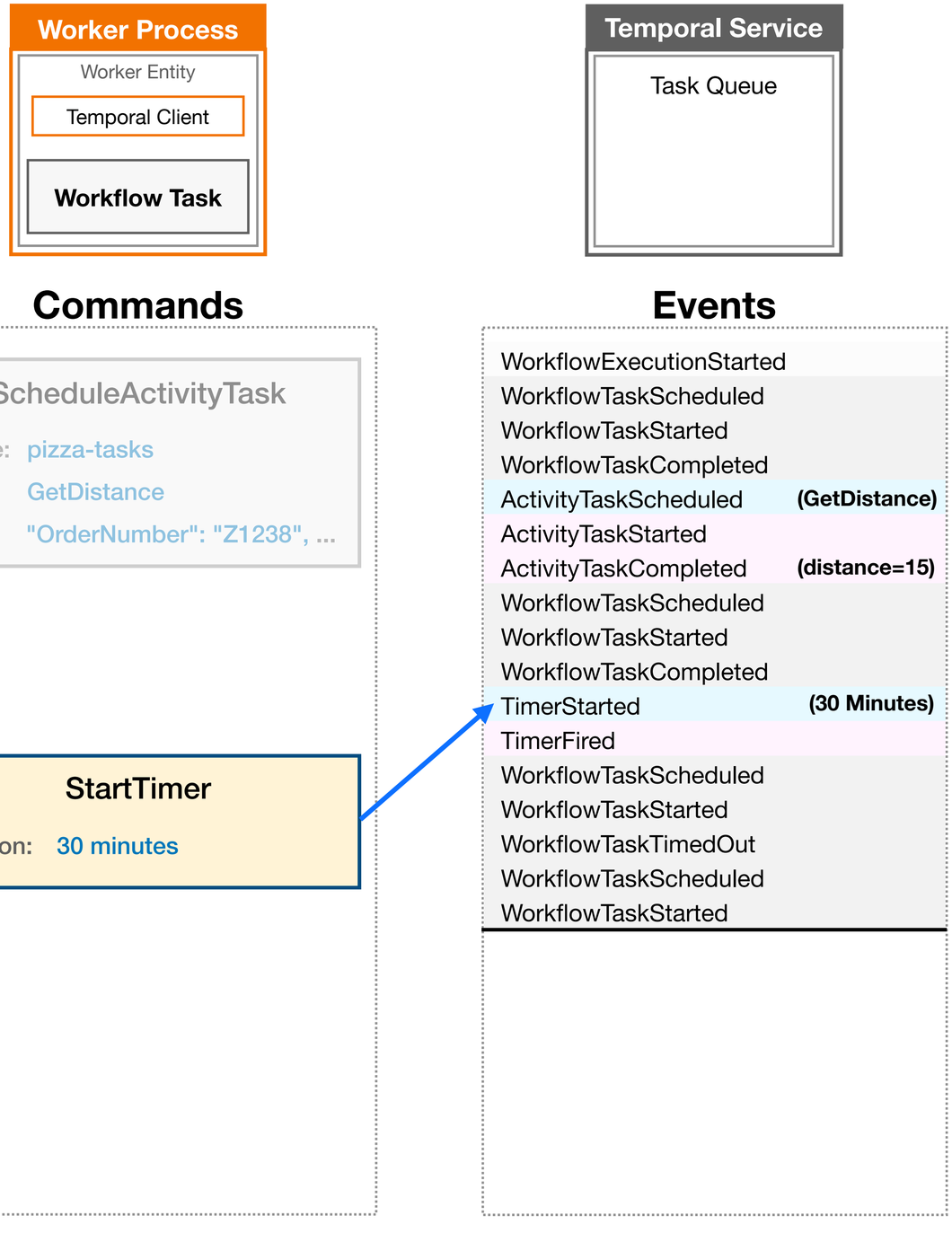
StartTimer

Duration: 30 minutes

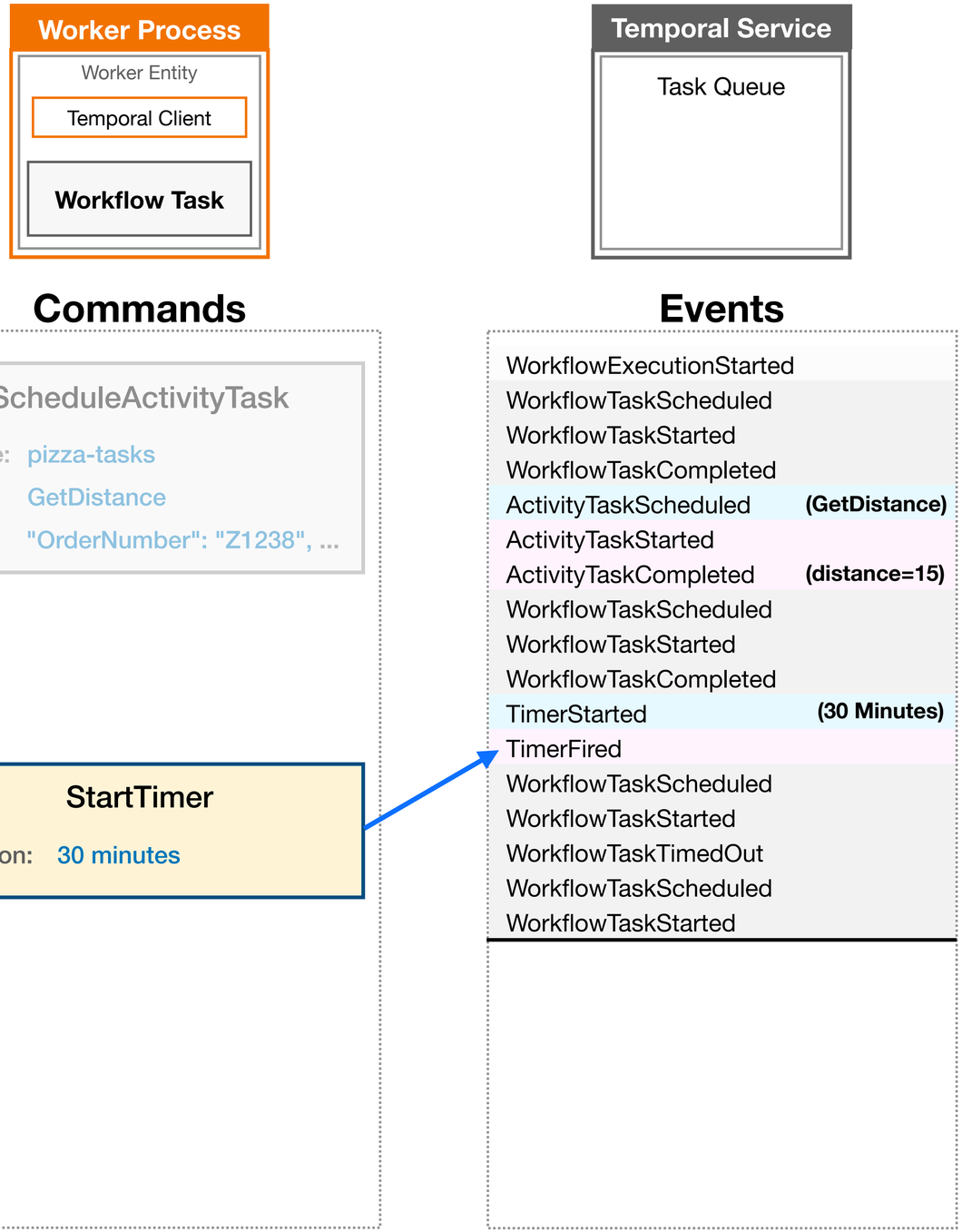
Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

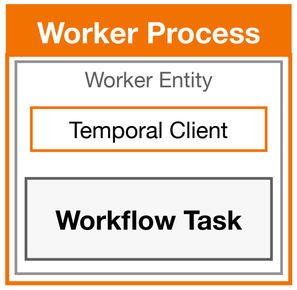
```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```

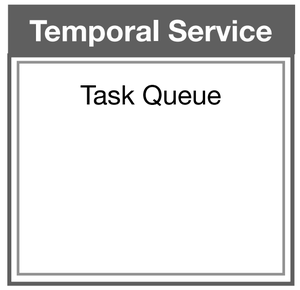


```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

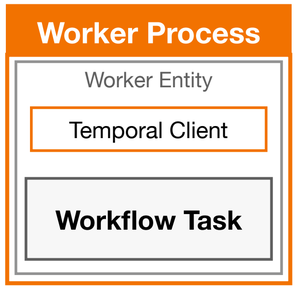
StartTimer

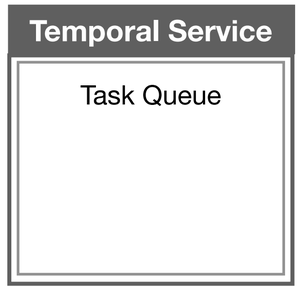
30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

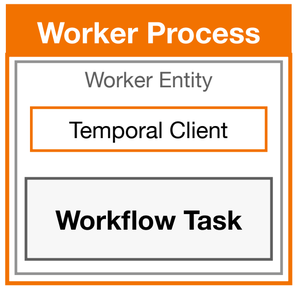
StartTimer

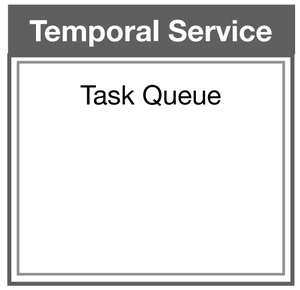
30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

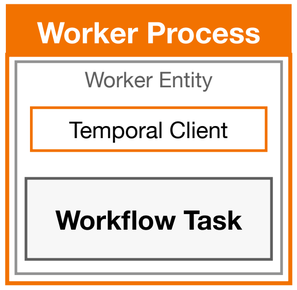
StartTimer

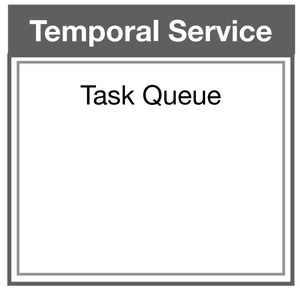
30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

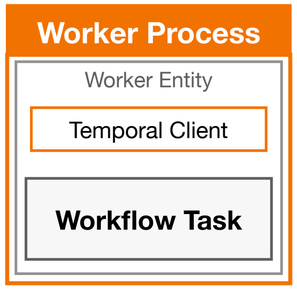
StartTimer

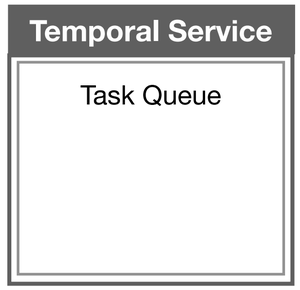
30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

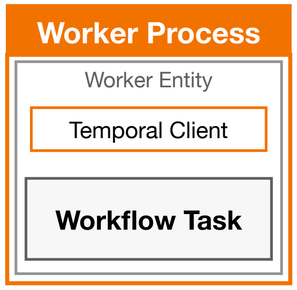
StartTimer

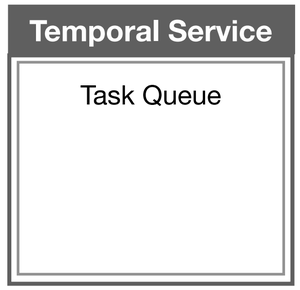
30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

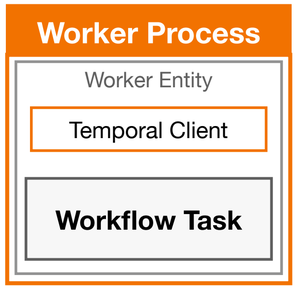
StartTimer

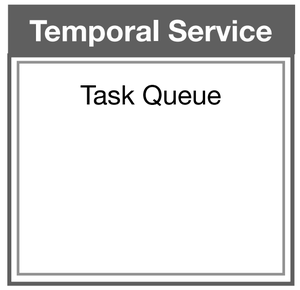
30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

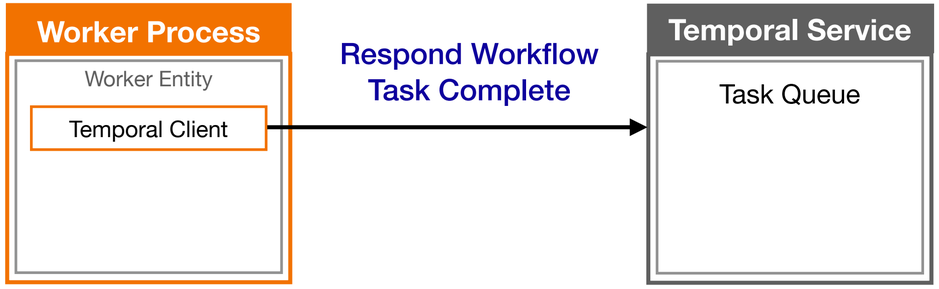
StartTimer

30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

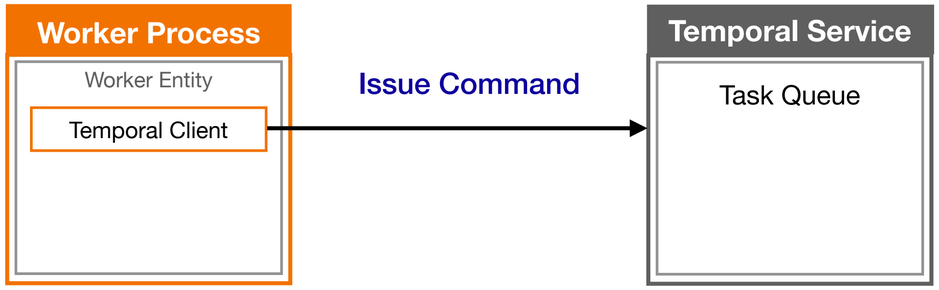
StartTimer

30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

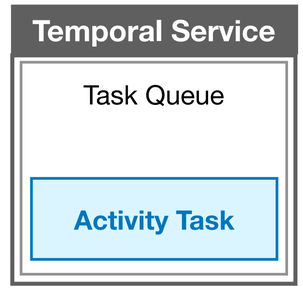
ScheduleActivityTask

Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

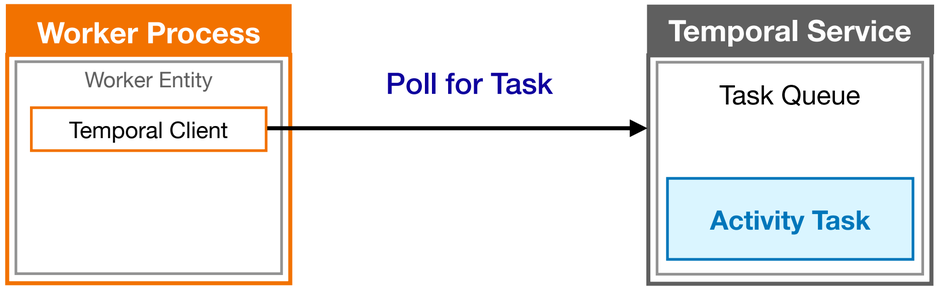
ScheduleActivityTask

Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(SendBill)**

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

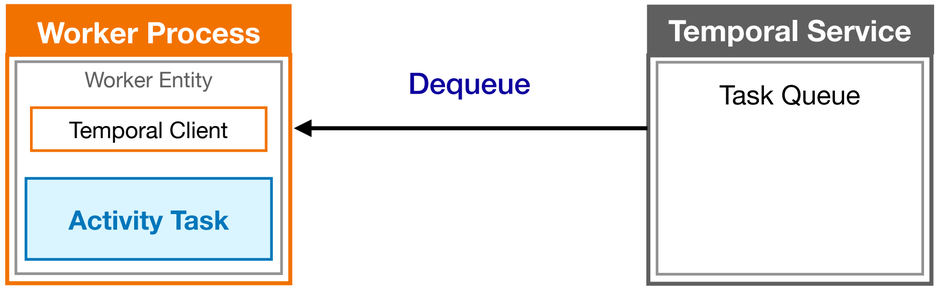
ScheduleActivityTask

Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(SendBill)**

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

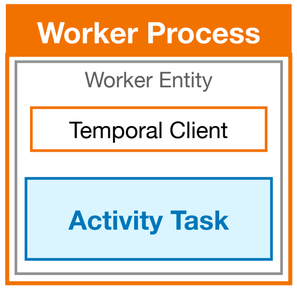
ScheduleActivityTask

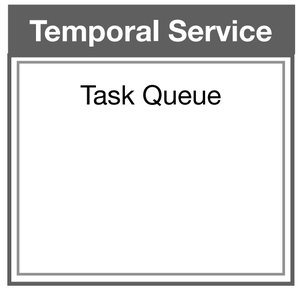
Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(SendBill)**
ActivityTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

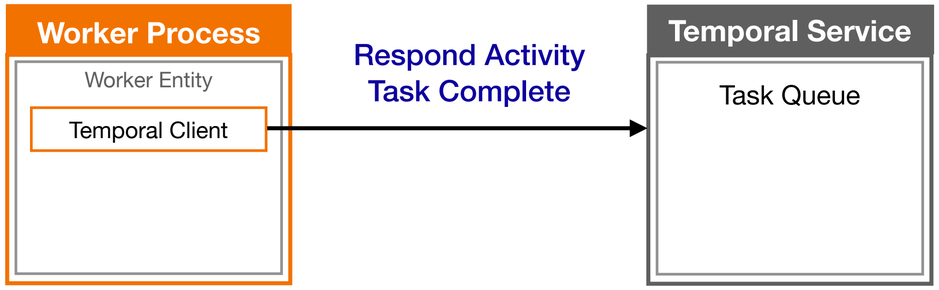
ScheduleActivityTask

Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(SendBill)**
ActivityTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

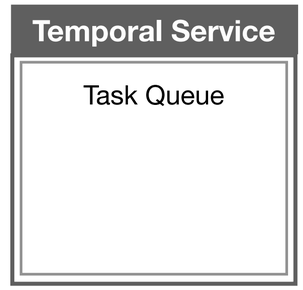
ScheduleActivityTask

Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(SendBill)**
ActivityTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

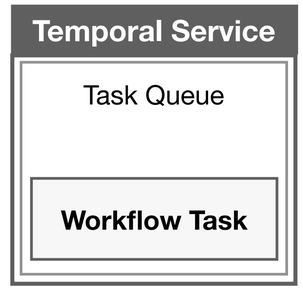
ScheduleActivityTask

Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(SendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

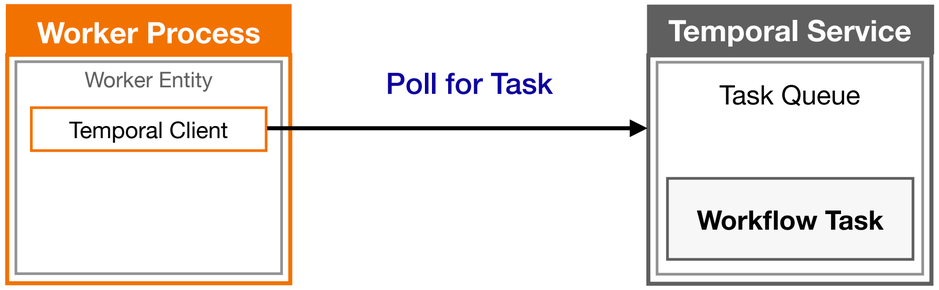
ScheduleActivityTask

Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(SendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

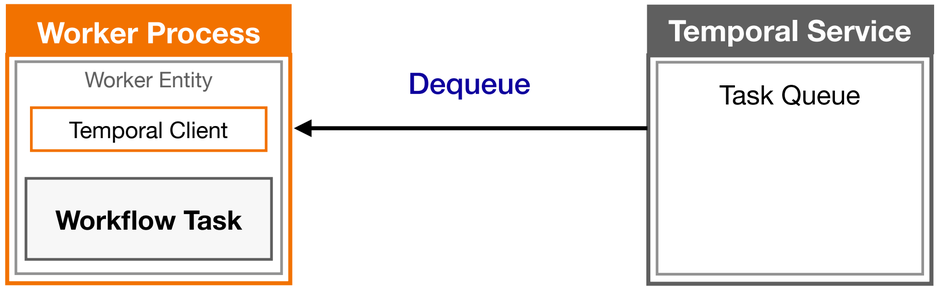
ScheduleActivityTask

Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(SendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

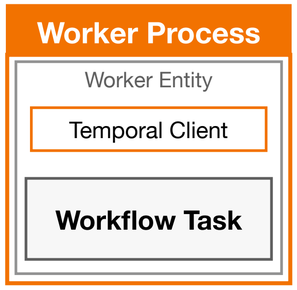
ScheduleActivityTask

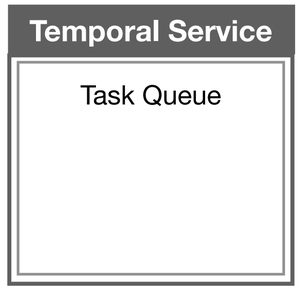
Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(SendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

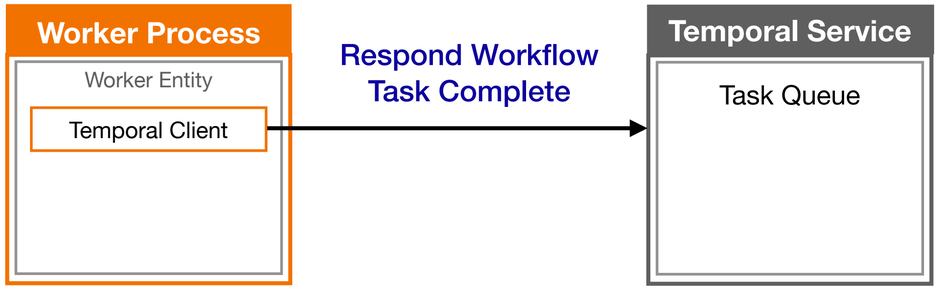
ScheduleActivityTask

Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(SendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

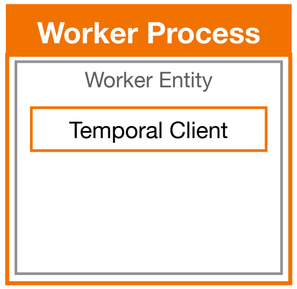
ScheduleActivityTask

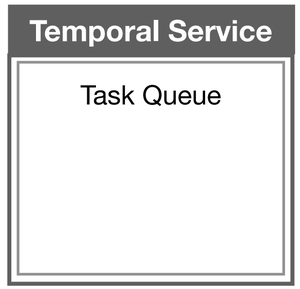
Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(SendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	
WorkflowTaskStarted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

ScheduleActivityTask

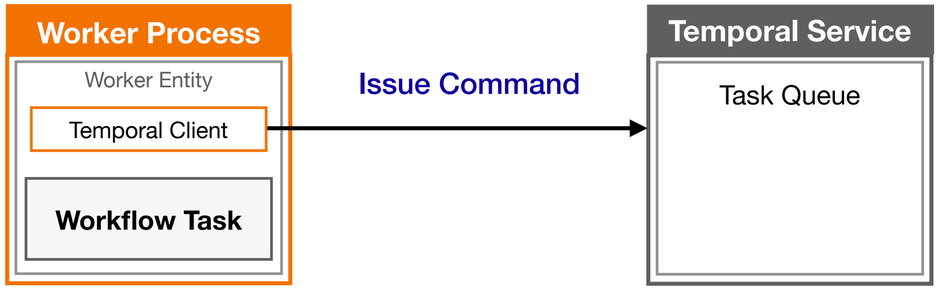
Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

Events

- WorkflowExecutionStarted
- WorkflowTaskScheduled
- WorkflowTaskStarted
- WorkflowTaskCompleted
- ActivityTaskScheduled **(GetDistance)**
- ActivityTaskStarted
- ActivityTaskCompleted **(distance=15)**
- WorkflowTaskScheduled
- WorkflowTaskStarted
- WorkflowTaskCompleted
- TimerStarted **(30 Minutes)**
- TimerFired
- WorkflowTaskScheduled
- WorkflowTaskStarted
- WorkflowTaskTimedOut
- WorkflowTaskScheduled
- WorkflowTaskStarted

- WorkflowTaskCompleted
- ActivityTaskScheduled **(SendBill)**
- ActivityTaskStarted
- ActivityTaskCompleted **(confirmation=...)**
- WorkflowTaskScheduled
- WorkflowTaskStarted
- **WorkflowTaskCompleted**

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

ScheduleActivityTask

Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

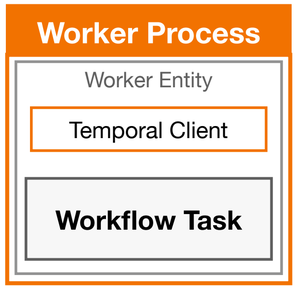
CompleteWorkflowExecution

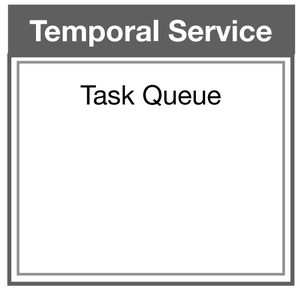
Result: "ConfirmationNumber": "TPD-26074139"

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(SendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```go
func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
    logger := workflow.GetLogger(ctx)

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Second * 5,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    var totalPrice int
    for _, pizza := range order.Items {
        totalPrice += pizza.Price
    }

    logger.Info("Calculated cost of order", "Total", totalPrice)

    var distance Distance
    future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
    _ = future.Get(ctx, &distance)

    if order.IsDelivery && distance.Kilometers > 25 {
        return "", errors.New("customer lives too far away for delivery")
    }

    _ = workflow.Sleep(ctx, time.Minute * 30)

    // call a local function to create the input passed to next Activity
    bill := createBill(order, totalPrice)

    var confirmation OrderConfirmation
    future = workflow.ExecuteActivity(ctx, SendBill, bill)
    _ = future.Get(ctx, &confirmation)

    return confirmation, nil
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

ScheduleActivityTask

Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

CompleteWorkflowExecution

Result: "ConfirmationNumber": "TPD-26074139"

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(SendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
WorkflowExecutionCompleted	

Why Temporal Requires Determinism for Workflows

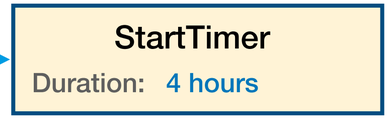
Workflow Definition

```
01 func DeterministicWorkflow(ctx workflow.Context) error {
02
03     options := workflow.ActivityOptions{
04         StartToCloseTimeout: time.Minute * 45,
05     }
06     ctx = workflow.WithActivityOptions(ctx, options)
07
08     err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
09     if err != nil {
10         return err
11     }
12
13     workflow.Sleep(ctx, time.Hour * 4)
14
15     err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
16     if err != nil {
17         return err
18     }
19
20     return nil
21 }
```

Workflow Definition

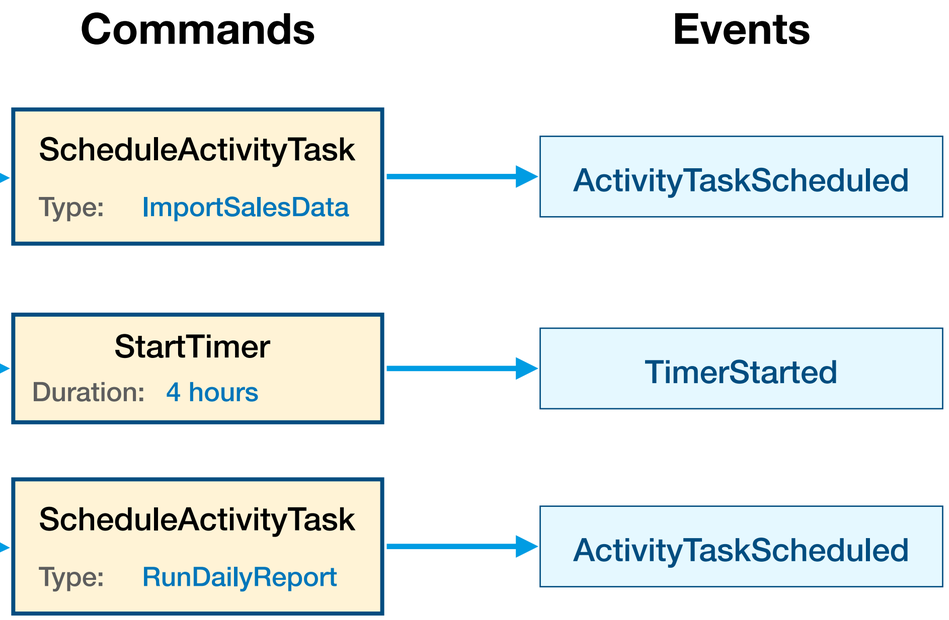
```
01 func DeterministicWorkflow(ctx workflow.Context) error {
02
03     options := workflow.ActivityOptions{
04         StartToCloseTimeout: time.Minute * 45,
05     }
06     ctx = workflow.WithActivityOptions(ctx, options)
07
08     err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
09     if err != nil {
10         return err
11     }
12
13     workflow.Sleep(ctx, time.Hour * 4)
14
15     err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
16     if err != nil {
17         return err
18     }
19
20     return nil
21 }
```

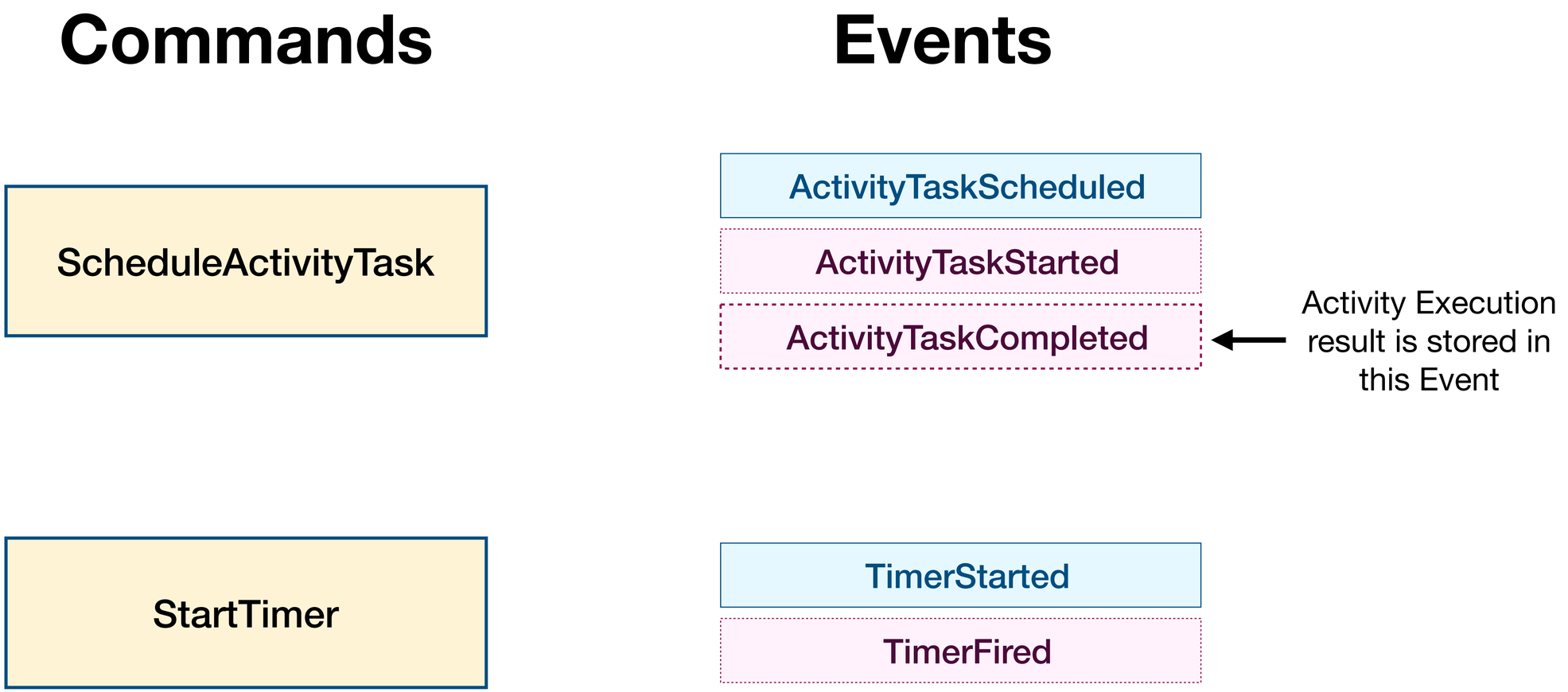
Commands



Workflow Definition

```go
func DeterministicWorkflow(ctx workflow.Context) error {

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Minute * 45,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
    if err != nil {
        return err
    }

    workflow.Sleep(ctx, time.Hour * 4)

    err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
    if err != nil {
        return err
    }

    return nil
}
```




Commands
Events
ScheduleActivityTask
ActivityTaskScheduled
ActivityTaskStarted
ActivityTaskCompleted
Activity Execution result is stored in this Event
StartTimer
TimerStarted
TimerFired

Workflow Definition

```
01 func DeterministicWorkflow(ctx workflow.Context) error {
02
03     options := workflow.ActivityOptions{
04         StartToCloseTimeout: time.Minute * 45,
05     }
06     ctx = workflow.WithActivityOptions(ctx, options)
07
08     err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
09     if err != nil {
10         return err
11     }
12
13     workflow.Sleep(ctx, time.Hour * 4)
14
15     err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
16     if err != nil {
17         return err
18     }
19
20     return nil
21 }
```

Workflow Definition

```go
func DeterministicWorkflow(ctx workflow.Context) error {

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Minute * 45,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
    if err != nil {
        return err
    }

    workflow.Sleep(ctx, time.Hour * 4)

    err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
    if err != nil {
        return err
    }

    return nil
}
```

Commands

ScheduleActivityTask

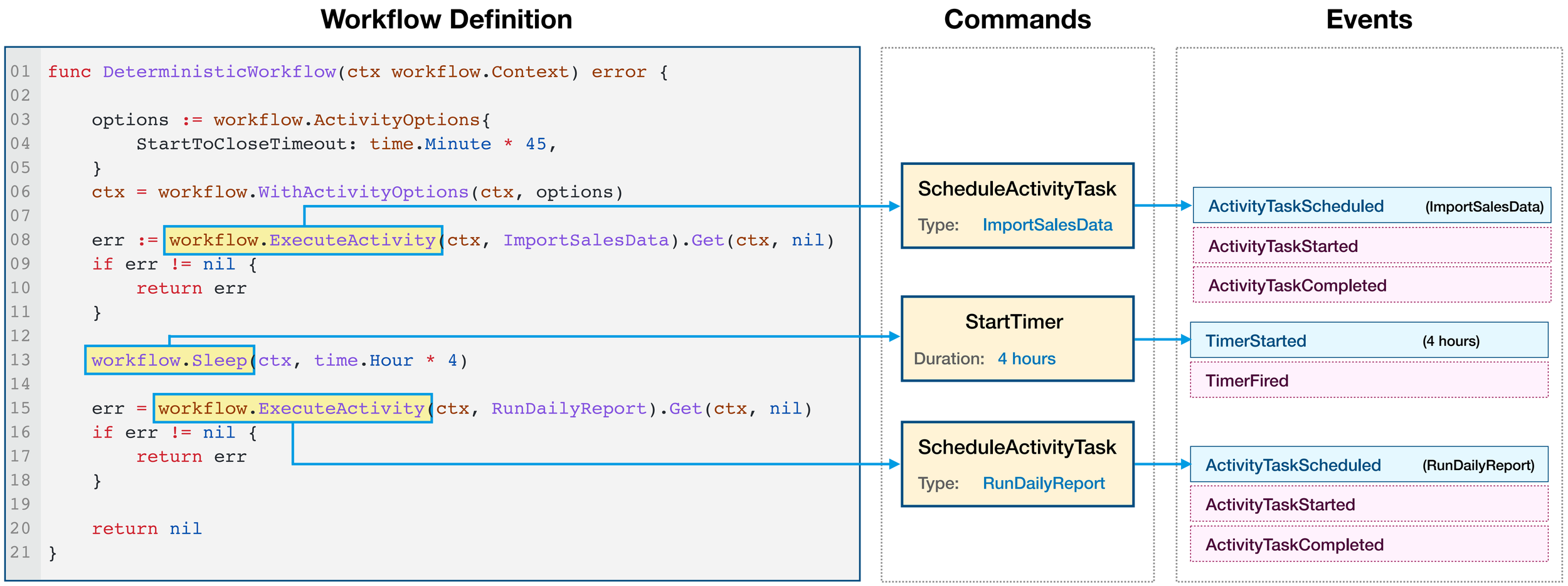
Type: ImportSalesData

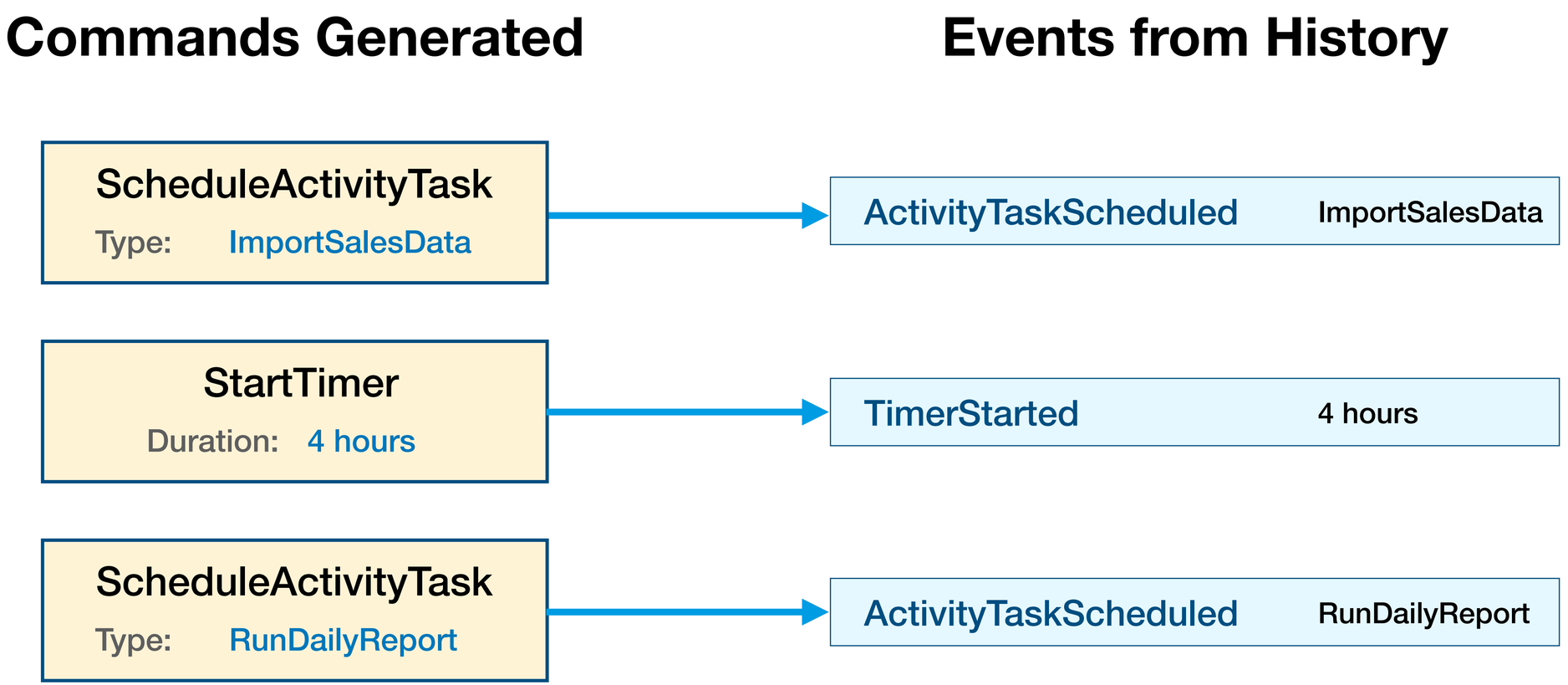
StartTimer

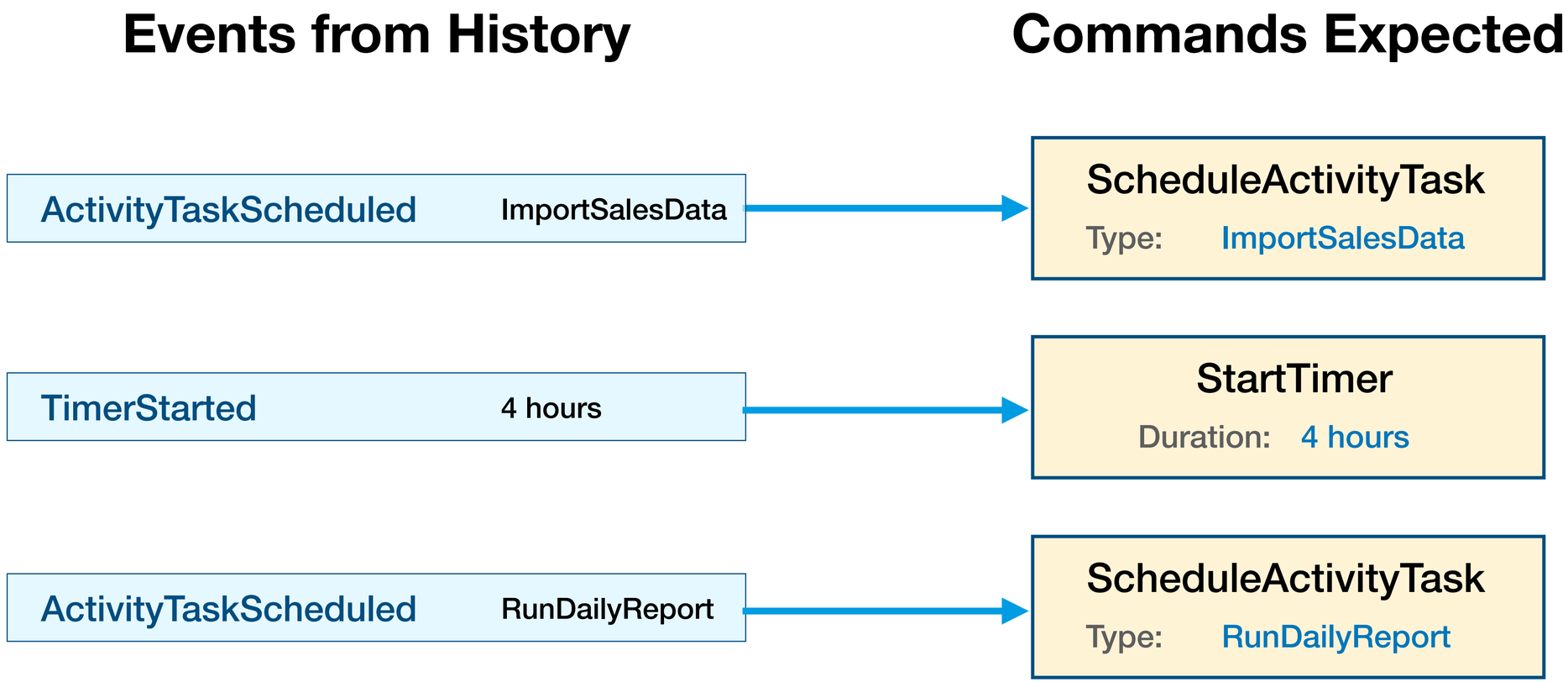
Duration: 4 hours

ScheduleActivityTask

Type: RunDailyReport


Workflow Definition
Commands
Events
func DeterministicWorkflow(ctx workflow.Context) error {
options := workflow.ActivityOptions{
StartToCloseTimeout: time.Minute * 45,
}
ctx = workflow.WithActivityOptions(ctx, options)
err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
if err != nil {
return err
}
workflow.Sleep(ctx, time.Hour * 4)
err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
if err != nil {
return err
}
return nil
}
ScheduleActivityTask
Type: ImportSalesData
StartTimer
Duration: 4 hours
ScheduleActivityTask
Type: RunDailyReport
ActivityTaskScheduled (ImportSalesData)
ActivityTaskStarted
ActivityTaskCompleted
TimerStarted (4 hours)
TimerFired
ActivityTaskScheduled (RunDailyReport)
ActivityTaskStarted
ActivityTaskCompleted


Commands Generated
Events from History
ScheduleActivityTask
Type: ImportSalesData
ActivityTaskScheduled
ImportSalesData
StartTimer
Duration: 4 hours
TimerStarted
4 hours
ScheduleActivityTask
Type: RunDailyReport
ActivityTaskScheduled
RunDailyReport


Events from History
Commands Expected
ActivityTaskScheduled
ImportSalesData
ScheduleActivityTask
Type:
ImportSalesData
TimerStarted
4 hours
StartTimer
Duration:
4 hours
ActivityTaskScheduled
RunDailyReport
ScheduleActivityTask
Type:
RunDailyReport

Example of a Non-Deterministic Workflow

A Non-Deterministic Workflow Definition

```go
func NonDeterministicWorkflow(ctx workflow.Context) error {

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Minute * 45,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    // this Activity is always executed
    err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
    if err != nil {
        return err
    }

    if rand.Intn(100) >= 50 {
       workflow.Sleep(ctx, time.Hour * 4)
    }

    workflow.GetLogger(ctx).Info("Preparing to run daily report")
    err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
    if err != nil {
        return err
    }

    return nil
}
```

Commands Created

ScheduleActivityTask

Type: ImportSalesData

Relevant Events Logged

- ActivityTaskScheduled (ImportSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted

A Non-Deterministic Workflow Definition

```go
func NonDeterministicWorkflow(ctx workflow.Context) error {

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Minute * 45,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    // this Activity is always executed
    err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
    if err != nil {
        return err
    }

    if rand.Intn(100) >= 50 {
       workflow.Sleep(ctx, time.Hour * 4)
    }

    workflow.GetLogger(ctx).Info("Preparing to run daily report")
    err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
    if err != nil {
        return err
    }

    return nil
}
```

Commands Created

ScheduleActivityTask

Type: ImportSalesData

Relevant Events Logged

ActivityTaskScheduled (ImportSalesData)

ActivityTaskStarted

ActivityTaskCompleted

A Non-Deterministic Workflow Definition

```go
func NonDeterministicWorkflow(ctx workflow.Context) error {

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Minute * 45,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    // this Activity is always executed
    err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
    if err != nil {
        return err
    }

    if rand.Intn(100) >= 50 {
       workflow.Sleep(ctx, time.Hour * 4)
    }

    workflow.GetLogger(ctx).Info("Preparing to run daily report")
    err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
    if err != nil {
        return err
    }

    return nil
}
```

Happens to return 84 during this execution

Commands Created

ScheduleActivityTask

Type: ImportSalesData

Relevant Events Logged

- ActivityTaskScheduled (ImportSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted

A Non-Deterministic Workflow Definition

```go
func NonDeterministicWorkflow(ctx workflow.Context) error {

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Minute * 45,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    // this Activity is always executed
    err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
    if err != nil {
        return err
    }

    if rand.Intn(100) >= 50 {
       workflow.Sleep(ctx, time.Hour * 4)
    }

    workflow.GetLogger(ctx).Info("Preparing to run daily report")
    err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
    if err != nil {
        return err
    }

    return nil
}
```

Commands Created

ScheduleActivityTask

Type: ImportSalesData

StartTimer

Duration: 4 hours

Relevant Events Logged

- ActivityTaskScheduled (ImportSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted

- TimerStarted (4 hours)
- TimerFired

A Non-Deterministic Workflow Definition

```go
func NonDeterministicWorkflow(ctx workflow.Context) error {

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Minute * 45,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    // this Activity is always executed
    err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
    if err != nil {
        return err
    }

    if rand.Intn(100) >= 50 {
       workflow.Sleep(ctx, time.Hour * 4)
    }

    workflow.GetLogger(ctx).Info("Preparing to run daily report")
    err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
    if err != nil {
        return err
    }

    return nil
}
```

Worker crashes here

Commands Created

ScheduleActivityTask

Type: ImportSalesData

StartTimer

Duration: 4 hours

Relevant Events Logged

ActivityTaskScheduled (ImportSalesData)

ActivityTaskStarted

ActivityTaskCompleted

TimerStarted (4 hours)

TimerFired

A Non-Deterministic Workflow Definition

```go
func NonDeterministicWorkflow(ctx workflow.Context) error {

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Minute * 45,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    // this Activity is always executed
    err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
    if err != nil {
        return err
    }

    if rand.Intn(100) >= 50 {
       workflow.Sleep(ctx, time.Hour * 4)
    }

    workflow.GetLogger(ctx).Info("Preparing to run daily report")
    err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
    if err != nil {
        return err
    }

    return nil
}
```

Commands Created

Relevant History Events

- ActivityTaskScheduled (ImportSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted

- TimerStarted (4 hours)
- TimerFired

Commands Expected (Based on History)

ScheduleActivityTask

Type: ImportSalesData

StartTimer

4 hours

A Non-Deterministic Workflow Definition

```go
func NonDeterministicWorkflow(ctx workflow.Context) error {

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Minute * 45,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    // this Activity is always executed
    err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
    if err != nil {
        return err
    }

    if rand.Intn(100) >= 50 {
       workflow.Sleep(ctx, time.Hour * 4)
    }

    workflow.GetLogger(ctx).Info("Preparing to run daily report")
    err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
    if err != nil {
        return err
    }

    return nil
}
```

Commands Created

ScheduleActivityTask

Type: ImportSalesData

Relevant History Events

ActivityTaskScheduled (ImportSalesData)

ActivityTaskStarted

ActivityTaskCompleted

TimerStarted (4 hours)

TimerFired

Commands Expected (Based on History)

ScheduleActivityTask

Type: ImportSalesData

StartTimer

4 hours

A Non-Deterministic Workflow Definition

```go
func NonDeterministicWorkflow(ctx workflow.Context) error {

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Minute * 45,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    // this Activity is always executed
    err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
    if err != nil {
        return err
    }

    if rand.Intn(100) >= 50 {
       workflow.Sleep(ctx, time.Hour * 4)
    }

    workflow.GetLogger(ctx).Info("Preparing to run daily report")
    err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
    if err != nil {
        return err
    }

    return nil
}
```

Commands Created

ScheduleActivityTask

Type: ImportSalesData

Relevant History Events

ActivityTaskScheduled (ImportSalesData)

ActivityTaskStarted

ActivityTaskCompleted

TimerStarted (4 hours)

TimerFired

Commands Expected (Based on History)

ScheduleActivityTask

Type: ImportSalesData

StartTimer

4 hours

A Non-Deterministic Workflow Definition

```go
func NonDeterministicWorkflow(ctx workflow.Context) error {

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Minute * 45,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    // this Activity is always executed
    err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
    if err != nil {
        return err
    }

    if rand.Intn(100) >= 50 {
       workflow.Sleep(ctx, time.Hour * 4)
    }

    workflow.GetLogger(ctx).Info("Preparing to run daily report")
    err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
    if err != nil {
        return err
    }

    return nil
}
```

Commands Created

ScheduleActivityTask

Type: ImportSalesData

Relevant History Events

- ActivityTaskScheduled (ImportSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted

- TimerStarted (4 hours)
- TimerFired

Commands Expected (Based on History)

ScheduleActivityTask

Type: ImportSalesData

StartTimer

4 hours

A Non-Deterministic Workflow Definition

```go
func NonDeterministicWorkflow(ctx workflow.Context) error {

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Minute * 45,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    // this Activity is always executed
    err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
    if err != nil {
        return err
    }

    if rand.Intn(100) >= 50 {
       workflow.Sleep(ctx, time.Hour * 4)
    }

    workflow.GetLogger(ctx).Info("Preparing to run daily report")
    err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
    if err != nil {
        return err
    }

    return nil
}
```

Happens to return 14 during this execution

Commands Created

ScheduleActivityTask

Type: ImportSalesData

Relevant History Events

- ActivityTaskScheduled (ImportSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted

- TimerStarted (4 hours)
- TimerFired

Commands Expected (Based on History)

ScheduleActivityTask

Type: ImportSalesData

StartTimer

4 hours

A Non-Deterministic Workflow Definition

```go
func NonDeterministicWorkflow(ctx workflow.Context) error {

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Minute * 45,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    // this Activity is always executed
    err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
    if err != nil {
        return err
    }

    if rand.Intn(100) >= 50 {
       workflow.Sleep(ctx, time.Hour * 4)
    }

    workflow.GetLogger(ctx).Info("Preparing to run daily report")
    err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
    if err != nil {
        return err
    }

    return nil
}
```

Commands Created

ScheduleActivityTask
Type: ImportSalesData

ScheduleActivityTask
Type: RunDailyReport

Relevant History Events

ActivityTaskScheduled (ImportSalesData)
ActivityTaskStarted
ActivityTaskCompleted

TimerStarted (4 hours)
TimerFired

Commands Expected (Based on History)

ScheduleActivityTask
Type: ImportSalesData

StartTimer
4 hours

A Non-Deterministic Workflow Definition

```go
func NonDeterministicWorkflow(ctx workflow.Context) error {

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Minute * 45,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    // this Activity is always executed
    err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
    if err != nil {
        return err
    }

    if rand.Intn(100) >= 50 {
       workflow.Sleep(ctx, time.Hour * 4)
    }

    workflow.GetLogger(ctx).Info("Preparing to run daily report")
    err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
    if err != nil {
        return err
    }

    return nil
}
```

Commands Created

ScheduleActivityTask
Type: ImportSalesData

ScheduleActivityTask
Type: RunDailyReport

Relevant History Events

ActivityTaskScheduled (ImportSalesData)
ActivityTaskStarted
ActivityTaskCompleted

TimerStarted (4 hours)
TimerFired

Commands Expected (Based on History)

ScheduleActivityTask
Type: ImportSalesData

StartTimer
4 hours

A Non-Deterministic Workflow Definition

```go
func NonDeterministicWorkflow(ctx workflow.Context) error {

    options := workflow.ActivityOptions{
        StartToCloseTimeout: time.Minute * 45,
    }
    ctx = workflow.WithActivityOptions(ctx, options)

    // this Activity is always executed
    err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
    if err != nil {
        return err
    }

    if rand.Intn(100) >= 50 {
       workflow.Sleep(ctx, time.Hour * 4)
    }

    workflow.GetLogger(ctx).Info("Preparing to run daily report")
    err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
    if err != nil {
        return err
    }

    return nil
}
```

Commands Created

ScheduleActivityTask
Type: ImportSalesData

ScheduleActivityTask
Type: RunDailyReport

Relevant History Events

ActivityTaskScheduled (ImportSalesData)
ActivityTaskStarted
ActivityTaskCompleted

TimerStarted (4 hours)
TimerFired

Commands Expected (Based on History)

ScheduleActivityTask
Type: ImportSalesData

StartTimer
4 hours

Using random numbers in a Workflow Definition has resulted in Non-Deterministic Error

Common Sources of Non-Determinism

Things to Avoid in a Workflow Definition (1)

- **Accessing external systems, such as databases or network services**
  - Instead, use Activities to perform these operations
- **Writing business logic or calling functions that rely on system time**
  - Instead, use Workflow-safe functions such as `workflow.Now` and `workflow.Sleep`

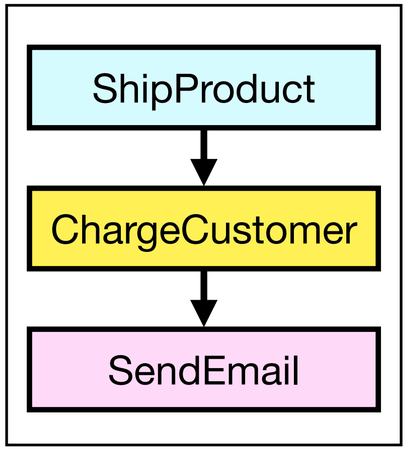
Things to Avoid in a Workflow Definition (2)

- **Working directly with threads or goroutines**
  - Instead, use the Workflow-safe `workflow.Go` function
  - To work with channels and selectors, use `workflow.Channel` and `workflow.Selector`
- **Do not iterate over data structures with unknown ordering**
- **We offer a static analyzer (`workflowcheck`) for Go**
  - This can identify many common non-deterministic violations in your code

How Workflow Changes Can Lead to Non-Deterministic Errors

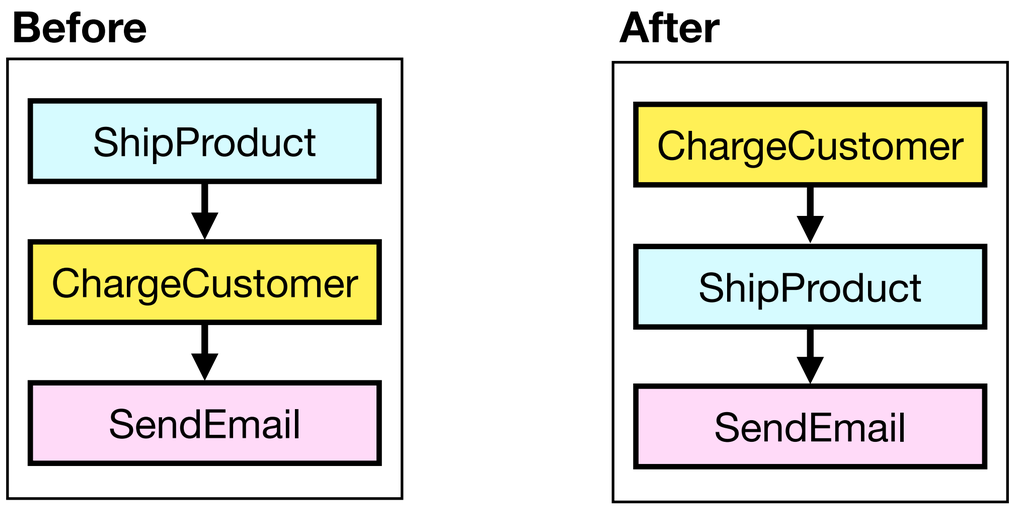
Non-Deterministic *Code* Isn't the Only Danger

- **As you've just learned, non-deterministic code can cause problems**
  - However, there's also another source of non-deterministic errors
  - This is more subtle and can't be detected through static analysis
- **Consider the following scenario**
  - You deploy and execute the following Workflow, which calls three Activities...



Deployment Leads to Non-Deterministic Error

- **While that Workflow is running, you decide to update the code**
  - You now want to charge the customer before shipping the product

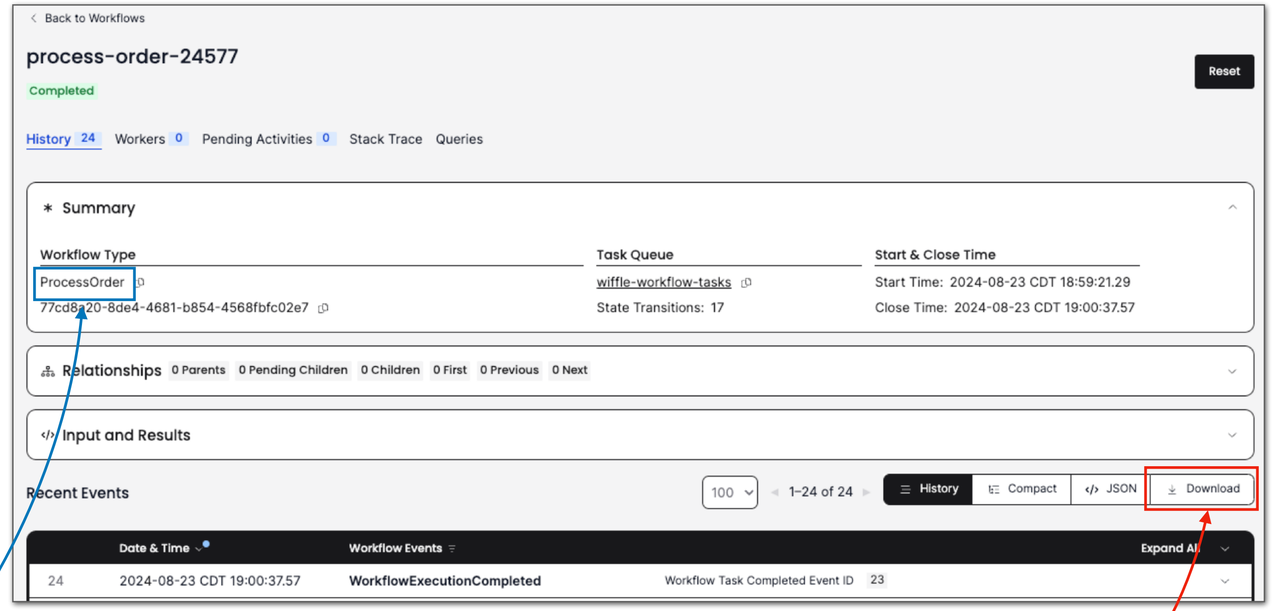


  - You deploy the updated code and restart the Worker(s) so that the change takes effect

- **What happens to the open execution when you restart the Worker?**

Deployment Leads to Non-Deterministic Error

- **Problem: Worker cannot restore previous state with the updated code**
- **How to detect?**
  - Test changes by replaying history of previous executions using new code before deploying
  - Only necessary if there are open executions at time of deployment
- **How to solve?**
  - Versioning (see documentation for details)




```
replayer := worker.NewWorkflowReplayer()
replayer.RegisterWorkflow("ProcessOrder")
err := replayer.ReplayWorkflowHistoryFromJSONFile("/Users/twheeler/Downloads/myhistory.json")
```

Beginner to Builder Bootcamp - Go

Signaling Your Workflows

What are Signals?

- **Signals provide a way to message open Workflow Executions**
  - They are sent asynchronously and may contain data
  - In response to receiving a Signal, the Workflow may change its state or flow of execution
- **They enable a Workflow Execution to respond to external stimuli**
- **Example use cases for Signals**
  - Handling user interactions, such as a button click in a GUI or web application
  - Reacting to events received from an external system, such as Apache Kafka
  - Refreshing data that changes periodically, such as product prices or sensor readings

Developing Signals

- **There are two steps to adding support for a Signal in your Workflow code**
- **Defining the Signal**
  - Specify the name and data structure that Temporal Clients will use to send the Signal
- **Handling the Signal**
  - Write code that will be invoked upon receiving the Signal from a Temporal Client

Defining Signals

- **The Signal is typically defined in the Workflow code**
  - A Workflow Definition can support multiple Signals
- **Each Signal is uniquely identified by the Signal Type**
  - The Temporal Client specifies this case-sensitive string when sending the Signal
- **The Signal can optionally accept input data**
  - If it does, this data must be serializable (the rules for Workflow input data apply here)

```
type JoinSignal struct {
  UserId string
  GroupId string
}
```

Handling Signals

- **Must specify what Workflow should do upon receiving Signal of that type**
  - The Go SDK uses Signal Channels to listen for Signals of a given type
  - Use the `GetSignalChannel` function in your Workflow Definition to listen for Signals

```
func YourWorkflowDefinition(ctx workflow.Context, param YourWorkflowParam) error {
    // ...
    var signal JoinSignal
    signalChan := workflow.GetSignalChannel(ctx, "join-signal")
    signalChan.Receive(ctx, &signal)
    if len(signal.GroupId) > 0 && len(signal.UserId) > 0 {
        // Add the user to the group
    }
    // ...
}
```

Awaiting within a Workflow

- **It is common for the Workflow to await some condition**
  - For example, an approval or confirmation (human-in-the-loop)
- **You can implement this by using `workflow.Await(ctx, bool)`**
  - This pauses execution until the expression supplied to this function resolves to `true`
  - Once it does resolve to true, execution continues with the next statement
- **The `workflow.AwaitWithTimeout` function is closely related**
  - It allows you to limit the wait time by passing a maximum duration as a parameter
  - If this duration is exceeded, the function cancels the context and returns `false`

Example of Using `workflow.Await`

```go
func ApprovalWorkflow(ctx workflow.Context) error {
   var isApproved bool // begins in unapproved state

   // Define and handle the Signal used for approval
   signalCh := workflow.GetSignalChannel(ctx, "approval-signal")
   workflow.Go(ctx, func(ctx workflow.Context) {
      for {
         var approved bool
         signalCh.Receive(ctx, &approved)

         if approved {
            isApproved = true   // this will resolve the await condition
            return
         }
      }
   })

   // Pause the Workflow Execution here until the Signal is approved
   err := workflow.Await(ctx, func() bool {
      return isApproved
   })
   if err != nil {
      return err
   }

   // Continue after approval
   return nil
}
```

How to Send Signals from Client

```
import(
    "go.temporal.io/sdk/client"
)

// ...

type MySignal struct {
    Message string
}

signal := MySignal {
    Message: "Some important data",
}

err = client.SignalWorkflow(context.Background(), "your-workflow-id", runID, "your-signal-
name", signal)
if err != nil {
    log.Fatalln("Error sending the Signal", err)
    return
}
```

Exercise #3: Sending Signals from the Client

- **During this exercise, you will**
  - Define and handle a Signal
  - Retrieve a handle on the Workflow to Signal
  - Send a Signal from the Client
- **Refer to the README.md file in the exercise environment for details**
  - The code is below the **`exercises/sending-signals-client`**
    - 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

How to Send Signals from Within a Workflow

- **Workflows can also send Signals to other Workflows**
  - This is known as an External Signal
- **Use the `SignalExternalWorkflow` to send an External Signal**

```
func YourWorkflowDefinition(ctx workflow.Context, param YourWorkflowParam) error {
  signal := MySignal { Message: "Some important data" }
  err :=  workflow.SignalExternalWorkflow(ctx, "workflow-id", "", "signal-name", signal).Get(ctx, nil)
  if err != nil {
    // ...
  }
}
```

Signal-With-Start

- **Signals a Workflow Execution by Workflow ID, starting it if necessary**
  - If there is currently an open Workflow Execution with this Workflow ID, it will be signaled
  - If not, one is started and the Signal immediately delivered to it

```
signal := MySignal {
    Message: "Some important data",
}

err = client.SignalWithStartWorkflow(context.Background(), "your-workflow-id",
"your-signal-name", signal)
if err != nil {
    log.Fatalln("Error sending the Signal", err)
    return
}
```

Signal Limits

- **There is a limit of 2,000 pending Signals sent per Workflow Execution**
  - Sending Signals in batches smaller than 2,000 can help you to stay within this limit
  - The Continue-As-New feature may also be helpful here
- **The Temporal Service also limits the total number of Signals *received***
  - Each Workflow Execution is limited to receiving 10,000 Signals during its entire lifespan

Unhandled Command

- **This error may occur when a Workflow receives a high volume of Signals in a short period of time**
  - It may be unable to process them all before attempting to complete or transition to another state.
  - Acceptable to ignore in Go SDK
  - Potentially performance degradation

Using `ReceiveAsync` to Drain Signals

- **Because of how Go implements Signal channels, it is possible for your Workflows to complete without having processed every waiting Signal**
  - To avoid this, you can call `ReceiveAsync` in a Go `defer` block or before using Continue-As-New
- **To use `ReceiveAsync` to drain signals in Temporal, you need to call `ReceiveAsync` in a loop until it returns `false`**
  - This indicates that the Signal channel is empty and all Signals have been processed

Using ReceiveAsync to Drain Signals

```
for {
    var signalVal string
    ok := signalChan.ReceiveAsync(&signalVal)
    if !ok {
        break;
    }
    workflow.GetLogger(ctx).Info("Received signal!", "signal", signalName,
"value", signalVal)
    // Process Signal
}
```

Important Considerations

- **Handling asynchronous work**
- **Payload size limitations**
- **Processing wait time**

Optional Exercise #4: Sending an External Signal

- **This is an optional exercise**
  - We will do this together if time allows; otherwise, you may do it on your own later
- **During this exercise, you will**
  - Define and handle a Signal
  - Retrieve a handle on the Workflow to Signal
  - Send an External Signal
  - Use a Temporal Client to submit execution requests for both Workflows
- **Refer to the README.md file in the exercise environment for details**
  - The code is below the **`exercises/sending-signals-external`**
    - 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

Querying Your Workflows

What are Queries?

- **Used to retrieve the state during or even after a Workflow Execution**
- **Examples of when you would use Queries**
  - Monitoring progress of long-running Workflows
  - Retrieving results

Query Attributes

- Do not mutate state of Workflow Execution
- Do not execute any Workflow command
- Queries are read-only and must complete synchronously
- Possible to send to a closed Workflow Execution
- Queries will return the most recent state
- A single Workflow can support multiple Queries

Developing Queries

1. Defining the Query Type (name)
2. Handling the Query

Defining a Query

- **Query Type: A unique name used to identify the query**
- **Can have arguments, but these are defined later**

Defining a Query

- **Query Type: A unique name used to identify the query**
- **Can have arguments, but these are defined later**

```
queryType := "your_query_name"
```

Query Handlers

- **These specify the actions a Workflow should take upon receiving a Query**
- **Use the `SetQueryHandler` method to handle Queries in a Workflow**

```
func YourWorkflow(ctx workflow.Context, input string) error {
    currentState := "started" // This could be any serializable struct.
    queryType := "current_state"
    err := workflow.SetQueryHandler(ctx, "current_state", func(prefix string,
suffix string) (string, error) {
        return prefix + currentState + suffix, nil
  })

  // ... unrelated Workflow code follows
```

Sending a Query with the SDK

- **The SDK provides the `QueryWorkflow` method to query a running or complete Workflow Execution**

```
response, err := temporalClient.QueryWorkflow(context.Background(),
workflowID, runID, queryType, "foo", "baz")

if err != nil {

  // ...

}
```

Sending a Query with the CLI

```
$ temporal workflow query \
    --workflow-id="state-id-0" \
    --type="getValueQuery" \
    --input=\"meaning-of-life\"
```

Exercise #5: Querying Workflows

- **During this exercise, you will**
  - Define and handle a Query
  - Create a handle on a Workflow to be queried
  - Call the Query from the Client
  - Send a Query from the Command Line
- **Refer to the README.md file in the exercise environment for details**
  - The code is below the **`exercises/querying-workflows`**
    - 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

Workflow Updates

What are Updates?

- **An update is a trackable, synchronous request to a Workflow Execution**
  - Comparable to sending a Signal and a Query at the same time
- **Unlike Signals, sender *must* wait until Worker processes the Update**
  - The sender *may* wait further to receive a returned value
  - If you need an asynchronous response, use a Signal instead
- **A Workflow cannot send an Update to another Workflow**

Update Handlers

- **Defining an Update Handlers is similar to a Signal and Query Handler**

```
var Language string

const SetLanguageUpdate = "set-language"

func GreetingWorkflow(ctx workflow.Context) error {
    language := English

    err = workflow.SetUpdateHandler(ctx, SetLanguageUpdate, func(ctx workflow.Context, newLanguage Language)
(Language, error) {
    // An Update handler can mutate the Workflow state and return a value
        previousLanguage := language
        language = newLanguage
        return previousLanguage, nil
    }
    //…
}
```

Sending Updates via the Go API

- **Use the Temporal Client's `UpdateWorkflow` method to send an update**
  - This returns a handle, which you can use to retrieve the Update's result

```
ctxWithTimeout, cancel := context.WithTimeout(context.Background(), 15*time.Second)
defer cancel()

updateHandle, err := temporalClient.UpdateWorkflow(ctxWithTimeout, client.UpdateWorkflowOptions{
    WorkflowID:   we.GetID(),
    RunID:        we.GetRunID(),
    UpdateName:   message.SetLanguageUpdate,
    WaitForStage: client.WorkflowUpdateStageAccepted,
    Args:         []interface{}{message.Chinese},
})
if err != nil {
    log.Fatalf("Unable to update workflow: %v", err)
}

var previousLang message.Language
err = updateHandle.Get(ctxWithTimeout, &previousLang)
if err != nil {
    log.Fatalf("Unable to get update result: %v", err)
}
```

Update Validators

- **Validators can be used to reject an Update before it is written to history**
  - To reject an Update, return an error in the Validator
- **If there is no Validator, each Update will be accepted**

Adding Validators to Update Handlers

```
type Language string

const SetLanguageUpdate = "set-language"

func GreetingWorkflow(ctx workflow.Context) error {
   language := English

   err = workflow.SetUpdateHandlerWithOptions(ctx, SetLanguageUpdate, func(ctx workflow.Context, newLanguage
Language) (Language, error) {
      // An Update handler can mutate the Workflow state and return a value.
      var previousLanguage Language
      previousLanguage, language = language, newLanguage
      return previousLanguage, nil
   }, workflow.UpdateHandlerOptions{
      Validator: func(ctx workflow.Context, newLanguage Language) error {
         if _, ok := greeting[newLanguage]; !ok {
            // In an Update validator you return any error to reject the Update
            return fmt.Errorf("%s unsupported language", newLanguage)
         }
         return nil
      },
   })
   ...
}
```

Update-with-Start and Dynamic Updates

- **Update-With-Start checks if there is currently a running Workflow Execution with the given Workflow ID**
  - Just as Signal-With-Start does
- **If one exists, it will be sent the Update**
  - If it does not, one will be started and then immediately sent the Update
- **Dynamic Update Handlers work the same as Dynamic Signal Handlers**

Beginner to Builder Bootcamp - Go

Timeouts

What are Timeouts?

- **A predefined duration provided for an operation to complete**
- **Temporal uses timeouts for two primary reasons:**
  - Detect failure
  - Establish a maximum time duration for your business logic

Activity Timeouts

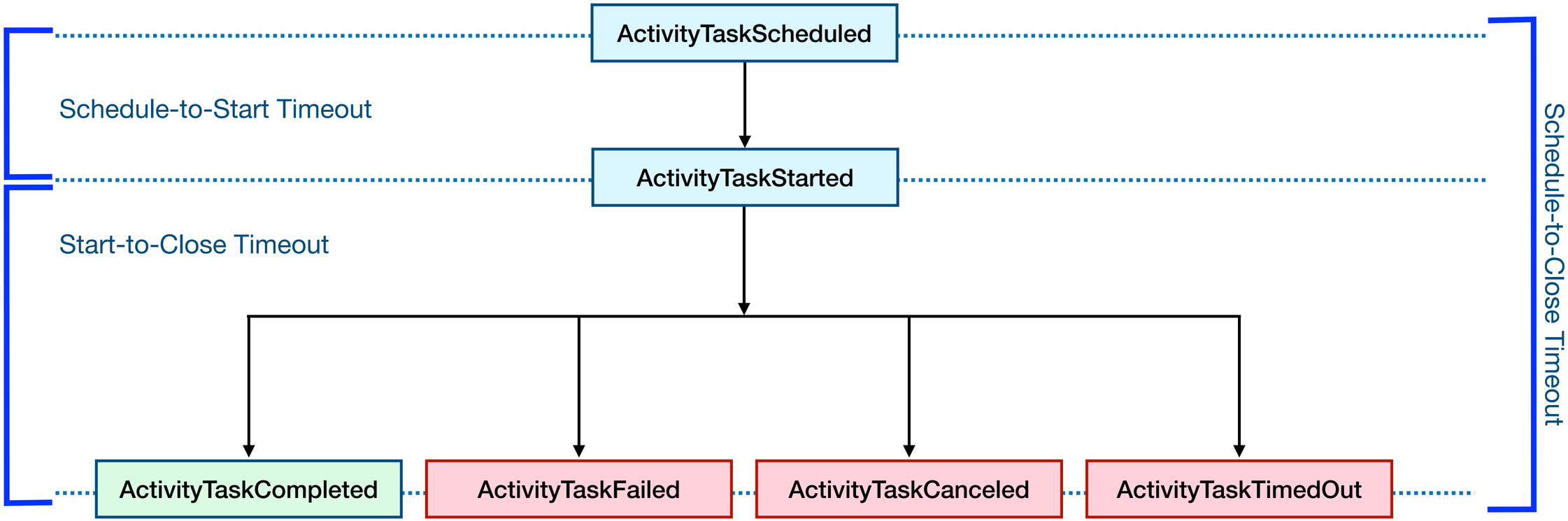
- **Controls the maximum duration of an aspect of an Activity Execution**
- **A measure of the time it takes to transition between one state to another**
- **Specified as an argument in the `ActivityOptions` struct**
- **As with an Activity that fails, an Activity that times out will be retried**
  - Based on details specified in the Retry Policy

Review of Activity Task States

Order	Event Type	Event Description
1	ActivityTaskScheduled	Temporal Service adds the Activity Task to the Task Queue
2	ActivityTaskStarted	Worker accepts the Activity Task; it's removed from the Task Queue)
3	ActivityTaskCompleted	Worker reports result of Activity Execution to the Temporal Service

(One of many closed states)

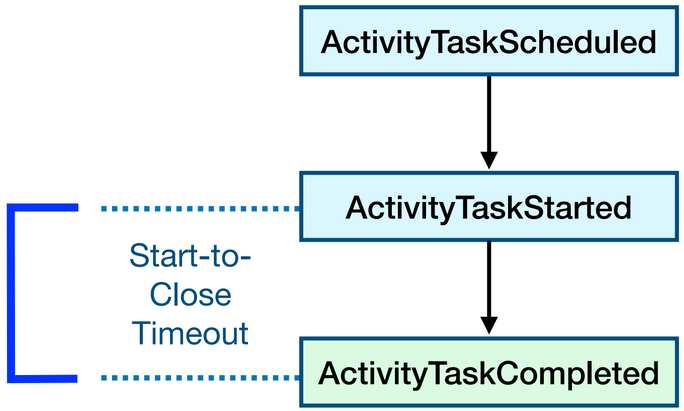
Understanding Activity Timeout Names



Start-to-Close Timeout

- **Limits maximum time allowed for a single Activity *Task* Execution**
  - This is the most common Activity Timeout to set
  - Time is reset for each retry attempt, since that will take place in a new Activity Task
  - Recommended: Set duration slightly longer than *maximum* time you expect the Activity will take

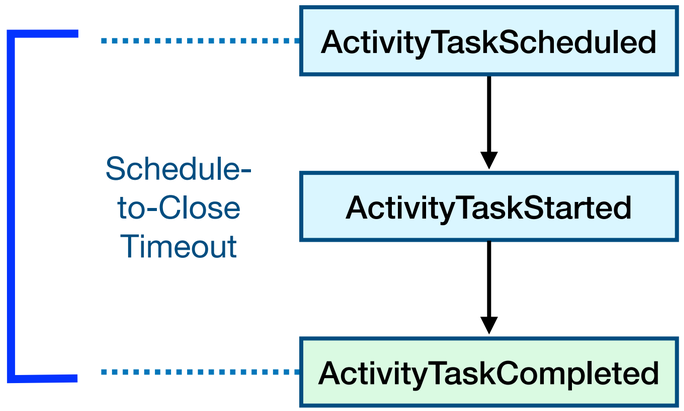
```
activityOptions := workflow.ActivityOptions{
  StartToCloseTimeout: 10 * time.Second,
}
ctx = workflow.WithActivityOptions(ctx, activityOptions)
var yourActivityResult YourActivityResult

err = workflow.ExecuteActivity(ctx,
    YourActivityDefinition,
    yourActivityParam).Get(ctx, &yourActivityResult)

if err != nil {
  // ...
}
```



Schedule-to-Close Timeout

- **Limits maximum time allowed for entire Activity Execution**
  - Because it includes all retries, it is typically less predictable than a Start-to-Close Timeout

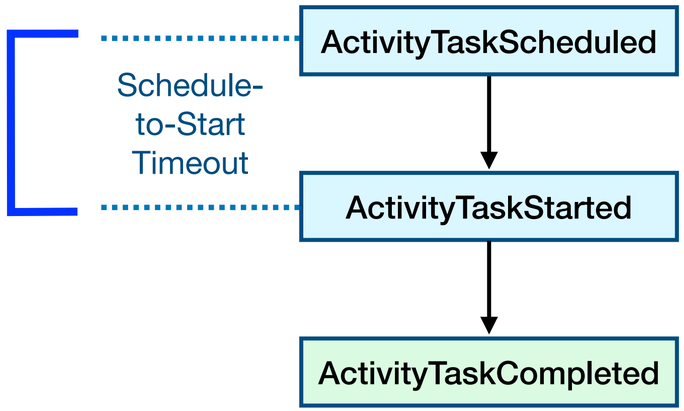
```
activityOptions := workflow.ActivityOptions{
  ScheduleToCloseTimeout: 10 * time.Second,
}
ctx = workflow.WithActivityOptions(ctx, activityOptions)
var yourActivityResult YourActivityResult

err = workflow.ExecuteActivity(ctx,
    YourActivityDefinition,
    yourActivityParam).Get(ctx, &yourActivityResult)

if err != nil {
  // ...
}
```



Schedule-to-Start Timeout

- **Limits maximum time allowed for Activity Task to remain in Task Queue**
  - Ensures the Activity is started within a specified time frame
  - It is very seldom recommended to use this type of Timeout
  - If set, it is done *in addition to* a Start-to-Close or Schedule-to-Close Timeout

```
activityOptions := workflow.ActivityOptions{
  ScheduleToStartTimeout: 10 * time.Second,
}
ctx = workflow.WithActivityOptions(ctx, activityOptions)
var yourActivityResult YourActivityResult

err = workflow.ExecuteActivity(ctx,
    YourActivityDefinition,
    yourActivityParam).Get(ctx, &yourActivityResult)

if err != nil {
  // ...
}
```



Activity Timeout Best Practices

- **You are required to set a Schedule-to-Close or Start-to-Close Timeout**
  - It can be difficult to predict how long execution might take when retries are involved
  - Therefore, setting Start-to-Close is usually the better choice
- **Retry Policies allow you to specify a maximum number of retry attempts**
  - However, using Timeouts to limit the duration is typically more useful
  - Business logic tends to be concerned with how long something takes (for example, SLAs)

Workflow Timeouts

- **These limit the duration of various aspects of Workflow Execution**
  - Such as max time allowed for the entire execution or an individual Workflow Task
  - These are set when starting the Workflow Execution
- **We generally do not recommend setting Workflow Timeouts**

Activity Heartbeats

- **Periodic messages sent by the Activity to the Temporal Service that**
  - Indicate that the Worker is still running
  - Communicate progress made during Activity Execution
  - Allow the Activity to handle Cancellation

How to Send a Heartbeat Message

```
func YourActivityDefinition(ctx, YourActivityParam) (YourActivityResult, error) {
    // ...
    activity.RecordHeartbeat(ctx, details)
    // ...
}
```

Heartbeats and Cancellations

- **For an Activity to be cancellable, it must perform Heartbeating**
  - When an Activity sends a Heartbeat, the Temporal Service checks if there has been a cancellation request
  - If a cancellation request exists, it is included in the response to the Heartbeat
  - This allows the Activity to respond promptly to cancellation requests
- **If you need to cancel a long-running Activity Execution, make sure it is configured to send Heartbeats periodically**

Heartbeat Timeout

- **The maximum time allowed between Activity Heartbeats**
- **The Heartbeat Timeout must be set in order for Temporal to track the Heartbeats sent by the Activity**
  - If this Timeout is not set, any Heartbeats sent by the Activity will be ignored.

```
activityoptions := workflow.ActivityOptions{
  HeartbeatTimeout: 10 * time.Second,
}
```

Heartbeat Timeout Best Practices

- **To ensure efficient, handling of long-running Activities:**
  - Set Start-to-Close Timeout slightly longer than the maximum duration of your Activity
  - Your Heartbeat Timeout should be fairly short
- **Must send Heartbeats at intervals shorter than the Heartbeat Timeout**

Heartbeat Throttling

- **Heartbeats may be throttled by the Worker**
  - Throttled Heartbeats are not sent to the Temporal Service
- **Throttling reduces network traffic and load on the Temporal Service**
  - Suppresses Heartbeats that aren't necessary to prevent a Heartbeat Timeout
- **Does not apply to the final Heartbeat in the case of Activity Failure**

Heartbeat Throttling

Activity ID	Details	
4	**Activity Type**	**pollDeliveryDriver**
	Attempt	1
	Maximum Attempts	5
	Last Heartbeat	
	State	PENDING_ACTIVITY_STATE_STARTED
	Last Started Time	2024-08-08 UTC 01:28:12.76
	Last Worker Identity	45943@Angelas-MBP

Retry Policies

Retry Policies

- **By default, Temporal automatically retries an Activity that fails**
  - Activities are associated with a Retry Policy by default
  - A Retry Policy defines the details of how those retries are carried out
- **Unlike Activities, Workflow Executions are not retried by default**
  - Workflows are not associated with a Retry Policy by default
  - While failed *Workflow Executions* are not retried automatically, failed *Workflow Tasks* are
  - Workflow Tasks retry automatically and indefinitely

Customizing the Retry Policy for an Activity

Customize Retry Policy by creating a `RetryPolicy{}` object

Method	Specifies	Default Value
`InitialInterval`	Duration before the first retry	1 second
`BackoffCoefficient`	Multiplier used for subsequent retries	`2.0`
`MaximumInterval`	Maximum duration between retries, in seconds	`100 * InitialInterval`
`MaximumAttempts`	Maximum number of retry attempts before giving up	`0` (unlimited)
`NonRetryableErrorTypes`	List of application failure types that won't be retried	`[]` (empty array)

```
retrypolicy := &temporal.RetryPolicy{
    InitialInterval:        time.Second * 5,
    BackoffCoefficient:     4.0,
    MaximumInterval:        time.Second * 10,
    MaximumAttempts:        3,
    NonRetryableErrorTypes: []string{"ExpiredCreditCardError"},
}
```

Defining Errors as Non-Retryable Types

- **Not always appropriate to designate an Activity Execution as retryable or not when returning an `ApplicationFailure` from the Activity**
  - The implementer likely knows which errors are retryable or not
  - Others using a shared library may not wish to fail on that specific error
- **Return the error, designate it as non-retryable in the Retry Policy**
  - Known as a non-retryable error type

```
retrypolicy := &temporal.RetryPolicy{
    NonRetryableErrorTypes: []string{"ExpiredCreditCardError"},
}
```

Non-Retryable Errors vs. Error Types

- **Non-Retryable Errors**
  - Decision not to retry is made at run time on a per-occurrence basis
  - Implemented through code in your Activity Definition
    - One way is to return a `temporal.NewNonRetryableApplicationError`
    - Another way is to set the `NonRetryable` attribute to `true` on any `ApplicationError`
- **Non-Retryable Error Types**
  - Decision not to retry is made at design time for all occurrences of a given type
  - These types are specified in a Retry Policy

Retry Policy Configurations for Failures

- **Transient failure**
  - Resolved by retrying the operation immediately after the failure
  - Default Policy typically a good option
- **Intermittent failure**
  - Addressed by retrying the operation, spread out over a longer period of time
  - Configure a custom Backoff Coefficient and Maximum Interval
- **Permanent failure**
  - Cannot be resolved solely through retries, needs manual intervention
  - Configure Non-Retryable Error Types
- **Customize the Retry Policy according to anticipated failures for your Activity**

Defining and Using a Custom Retry Policy

```
retrypolicy := &temporal.RetryPolicy{
  MaximumInterval:         time.Second * 10,
  MaximumAttempts:         3,
}

options := workflow.ActivityOptions{
  StartToCloseTimeout: time.Second * 5,
  HeartbeatTimeout:    10 * time.Second,
  RetryPolicy:         retrypolicy,
}

activityRun, err := workflow.ExecuteActivity(ctx, options, ActivityDefinition)
```

1 Specify your policy values

2 Associate the Policy with the Activity Execution

Common Use Cases for Defining a Custom Retry Policy

- **Making calls to a service experiencing heavy load**
  - Setting a higher Maximum Interval to allow for longer delays between retries
- **If an external service implements rate limiting**
  - Use a longer Backoff Coefficient to avoid triggering this limit
- **A service charges for each call received**
  - One of the few instances where setting Maximum Attempts is appropriate

Best Practices for Retry Policies

- **Don’t unnecessarily set maximum attempts to 1**
  - Not a substitute for idempotency
  - Setting `MaximumAttempts` to **DOES NOT** mean only-once execution
- **Recognize that each Activity Execution can have its own retry policy**
  - You can call the same Activity with different Retry Policies
- **Avoid Retry Policies for Workflow Executions**

Customizing a Retry Policy for a Specific Activity

```
retrypolicy_lowbackoff := &temporal.RetryPolicy{
    InitialInterval:    time.Second,
    BackoffCoefficient: 2.0,
    MaximumInterval:    time.Second * 100,
}

activityOptions_lowbackoff := workflow.ActivityOptions{
    RetryPolicy: retrypolicy_lowbackoff,
}

retrypolicy_highbackoff := &temporal.RetryPolicy{
    InitialInterval:    time.Second,
    BackoffCoefficient: 20.0,
    MaximumInterval:    time.Second * 100,
}

activityOptions_highbackoff := workflow.ActivityOptions{
    RetryPolicy: retrypolicy_highbackoff,
}

if x == true {
    activityRun, err := workflow.ExecuteActivity(ctx, activityOptions_lowbackoff, ActivityDefinition)
} else {
    activityRun, err := workflow.ExecuteActivity(ctx, activityOptions_highbackoff, ActivityDefinition)
}
```

Retry Policy for Workflow Executions

- **Workflow Executions do not retry by default**
  - Workflow failures typically indicate a permanent failure
- **We don’t recommend using a Retry Policy with a Workflow Execution**

Exercise #6: Non-Retryable Error Types

- **During this exercise, you will**
  - Configure non-retry able error types for Activities
  - Implement customized retry policies for Activities
  - Add Heartbeats and Heartbeat timeouts to help users monitor the health of Activities
- **Refer to the README.md file in the exercise environment for details**
  - The code is below the **`exercises/non-retryable-error-types`**
    - 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

Beginner to Builder Bootcamp - Go

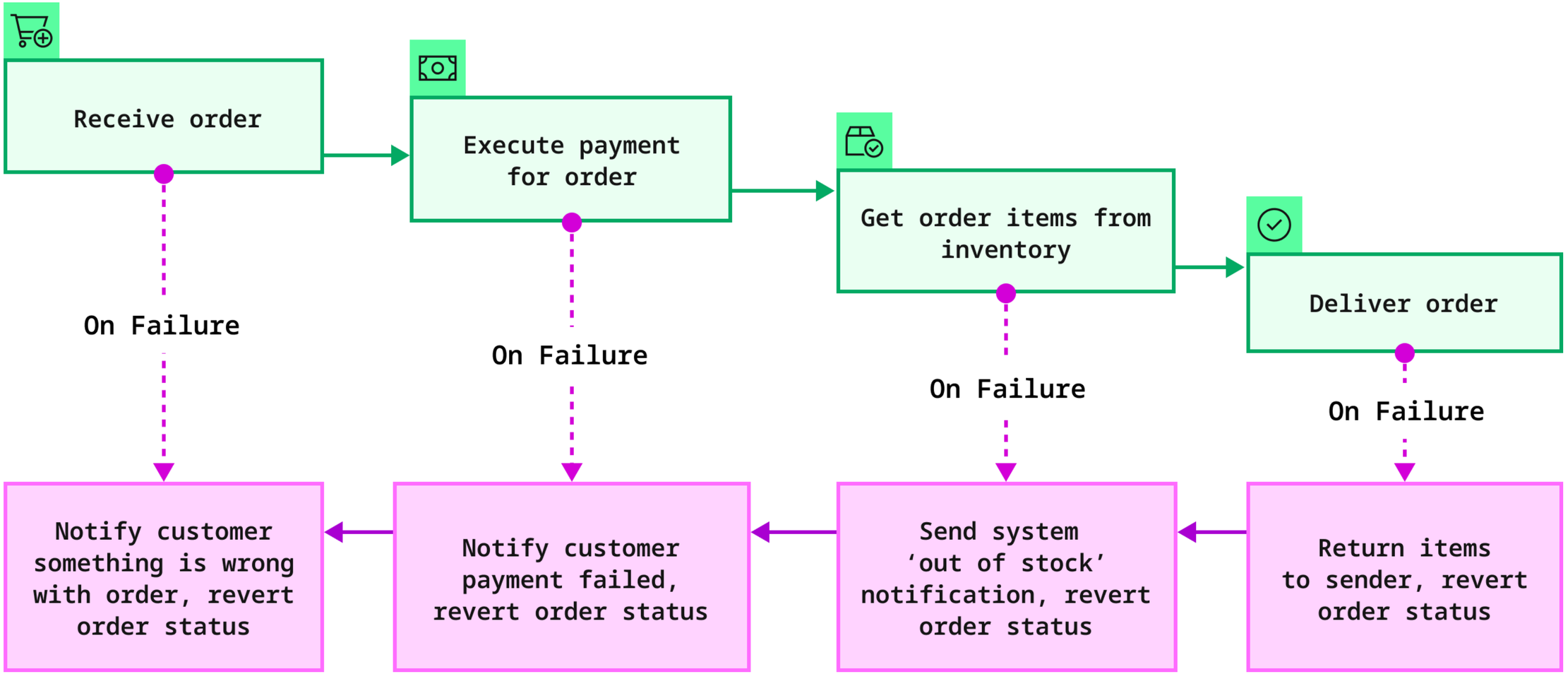
Handling a Workflow Execution that Cannot Complete

- **Cancel your Workflow Execution**
  - Provides a graceful way to stop the Workflow Execution
- **Terminate your Workflow Execution**
  - Forcefully halts the Workflow Execution
- **Reset your Workflow Execution**
  - Resets the Workflow Execution state to a previous point in the Event History

Rollback Actions and the Saga Pattern

- **A saga is a pattern used in distributed systems to manage a sequence of local transactions**
- **If any transaction in the sequence fails, the saga executes actions to rollback the previous operations**
  - This is known as a compensating action or compensation
- **Examples:**
  - Money movement
  - Trip booking
  - Order processing

Rollback Actions and the Saga Pattern



Rollback Actions and the Saga Pattern

```
err = workflow.ExecuteActivity(ctx, UpdateInventory, order.Items).Get(ctx, nil)
if err != nil {
    return OrderConfirmation{}, err
} else {
    // ...
}
```

Rollback Actions and the Saga Pattern

```
err = workflow.ExecuteActivity(ctx, UpdateInventory, order.Items).Get(ctx, nil)
if err != nil {
    return OrderConfirmation{}, err
}

defer func() {
    if err != nil {
        errCompensation := workflow.ExecuteActivity(ctx, RevertInventory,
order.Items).Get(ctx, nil)
    }
}()
```

Optional Exercise #7
Implementing a Rollback Action with the Saga Pattern

- **This is an optional exercise**
  - We will do this together if time allows; otherwise, you may do it on your own later
- **During this exercise, you will**
  - Orchestrate Activities using a Saga pattern to implement compensating transactions
  - Handle failures with rollback logic
- **Refer to the README.md file in the exercise environment for details**
  - The code is below the **`exercises/rollback-with-saga`**
    - 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

Beginner to Builder Bootcamp - Go

00. About this Workshop
01. The Basics of Temporal
02. Improving Your Temporal Application Code
03. Using Timers in a Workflow Definition
04. Understanding Event History
05. Understanding Workflow Determinism
06. Signals, Queries, and Workflow Updates
07. Timeouts and Retry Policies
08. Recovering from Failure
09. **Conclusion**

Conclusion (1)

- **Temporal is an open-source Durable Execution platform**
  - Temporal offers SDKs for several programming languages, including Go
- **Workflows are the core abstraction in a Temporal application**
  - They are functions that have the benefit of Durable Execution
  - Workflows are required to be deterministic
- **Activities are functions that encapsulate non-deterministic code**
  - They are automatically retried upon failure
  - You can change this behavior with a custom Retry Policy

Conclusion (2)

- **Workers are responsible for executing your application code**
  - You must restart Workers after deploying a code change
- **The Temporal Service orchestrates code execution**
  - The Temporal Service maintains dynamically-created Task Queues
  - Workers continuously poll a Task Queue and accept tasks if they have spare capacity
- **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, failures, and the Event History

Conclusion (3)

- **Timers are used to pause a Workflow Execution**
  - These are maintained by the Temporal Service, so they will survive a Worker crash
  - Because of Durable Execution, you can reliably use Timers with arbitrarily long durations
- **The Event History is an append-only log detailing a Workflow Execution**
  - It is maintained by the Temporal Service
  - This data is helpful for debugging and is the key to how Temporal achieves Durable Execution
- **Temporal uses History Replay to achieve Durable Execution**
  - Essentially, the Worker selectively re-runs code based on what is in the Event History

Conclusion (4)

- **Signals are a way of passing data to a running Workflow Execution**
  - This can be used to change its internal state
- **Queries are a way of retrieving data from a Workflow Execution**
  - This is used to inspect its state, but is not allowed to change it
- **Workflow Updates are similar to combining a Signal with a Query**
  - It enables you to supply and retrieve data to a Workflow Execution with a single call
  - Workflow Update Handlers can optionally use a Validator to reject changes

Conclusion (5)

- **Temporal provides built-in support for Retries and Timeouts**
  - It is common to customize Timeouts and Retry Policies for Activity Executions
  - It is unusual and seldom recommended to customize them for Workflow Executions
  - You can use Activity Heartbeating to quickly detect failure of a long-running Activity
- **Saga is a common pattern for failure recovery in distributed systems**
- **There are two options for using the Temporal Service**
  - Self-hosted: install, configure, and operate it yourself
  - Temporal Cloud: a fully-managed service that’s an alternative to self-hosting

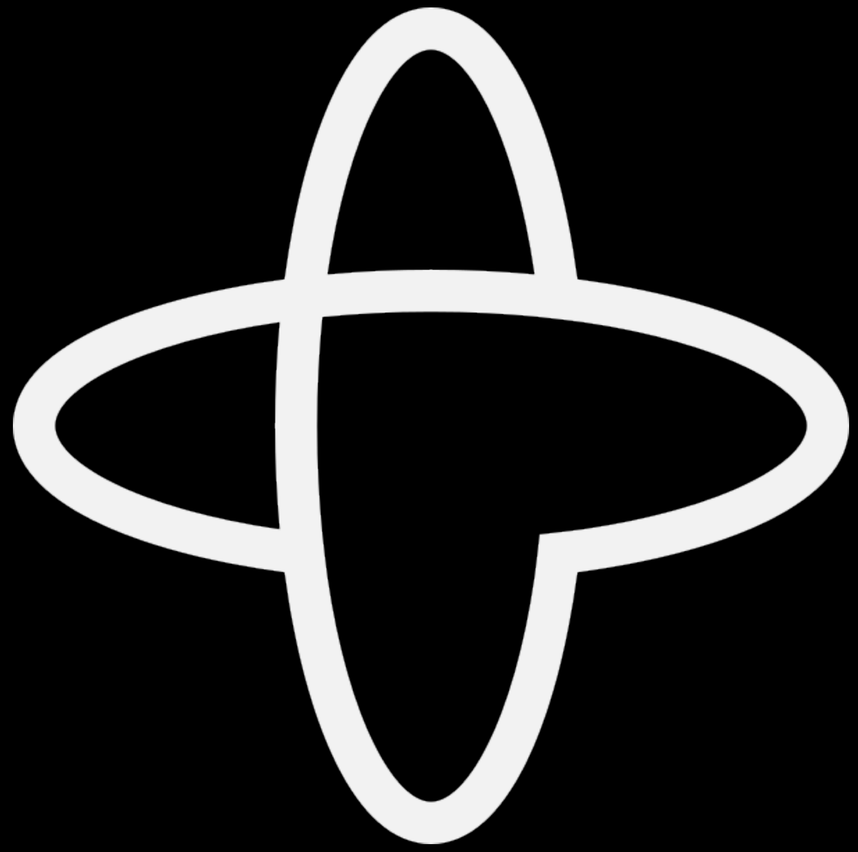
Temporal Service Options for Production

- **Self-Hosted**
  - The `temporal` CLI provides an easy way to run the Temporal Service for development
  - Using Docker Compose is another alternative for small-scale development
  - Production deployments often run on Kubernetes
- **Temporal Cloud**
  - Access to a Temporal Service run by experts via our fully-managed cloud service
    - Dependable: 99.9% uptime SLA and 24x7 production support
    - Frees your organization from having to plan, deploy, operate, and support a self-hosted Temporal Service
  - Your application runs on your own infrastructure

We’d appreciate your feedback on this workshop

Please follow the link below
or scan the QR code to access the survey

https://t.mp/replay26-ws-feedback



Thank You