Temporal 102

20230911a

Temporal 102

▶ **00. About this Workshop**

01. Understanding Key Concepts in Temporal
02. Improving Your Temporal Application Code
03. Using Timers in a Workflow Definition
04. Testing Your Temporal Application Code
05. Understanding Event History
06. Debugging Workflow Execution
07. Deploying Your Application to Production
08. Understanding Workflow Determinism
09. Conclusion

Logistics

- **Schedule**
- **Availability of the self-serve online version of *Temporal 102 with Go***
  - That version provides additional detail, plus coverage of Workflow Versioning
- **Asking questions**
- **Feedback about the course**
- **Course conventions: Activity vs activity**
- **Prerequisite: Did *everyone* already complete Temporal 101?**

During this workshop, you will

- Evaluate what a **production deployment** of Temporal looks like
- Use **Timers** to introduce delays in Workflow Execution
- Capture runtime information through **logging** in Workflow and Activity code
- Leverage the SDK's **testing support** to validate application behavior
- Differentiate **completion, failure, cancelation, and termination** of Workflow Executions
- Interpret **Event History** and debug problems with Workflow Execution
- Recognize **how Workflow code maps to Commands and Events** during Workflow Execution
- Consider **why Temporal requires determinism** for Workflow code
- Observe **how Temporal uses History Replay** to achieve durable execution of Workflows

Exercise Environment

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

https://t.mp/replay-102-go-code

GitPod Overview

Code editor

Embedded browser
(displays Temporal Web UI)

File browser
source code for exercises

Refresh button
(for Web UI)

Terminals

Temporal 102

00. About this Workshop
▶ 01. Understanding Key Concepts in Temporal
02. Improving Your Temporal Application Code
03. Using Timers in a Workflow Definition
04. Testing Your Temporal Application Code
05. Understanding Event History
06. Debugging Workflow Execution
07. Deploying Your Application to Production
08. Understanding Workflow Determinism
09. Conclusion

Temporal: A Durable Execution System

- **What is a durable execution system?**
  - Ensures that your application runs reliably despite adverse conditions
  - Automatically maintains application state and recovers from failure
  - Improves developer productivity by making applications easier to develop, scale, and support

Temporal Workflows

- **Workflows are the core abstraction in Temporal**
  - It represents the sequence of steps used to carry out your business logic
  - They are durable: Temporal automatically recreates state if execution ends unexpectedly
  - In the Go SDK, a Temporal Workflow is defined through a function
  - Temporal requires that Workflows are *deterministic*

< / > Workflow Definition

Temporal Activities

- **Activities encapsulate unreliable or non-deterministic code**
  - They are automatically retried upon failure
  - In the Go SDK, Activities are defined through functions

< / > Activity Definitions

< / > Workflow Definition

Temporal Workers

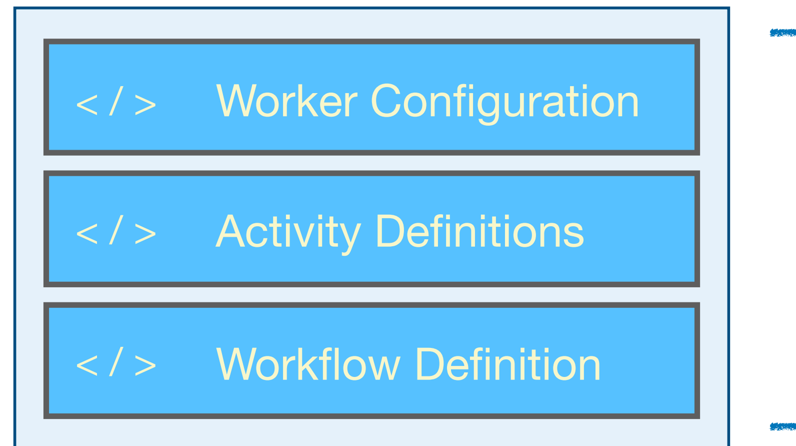
- **Workers are responsible for executing Workflow and Activity Definitions**
  - They poll a Task Queue maintained by the Temporal Cluster
- **The Worker implementation is provided by the Temporal SDK**
  - Your application will configure and start the Workers

< / > Worker Configuration

< / > Activity Definitions

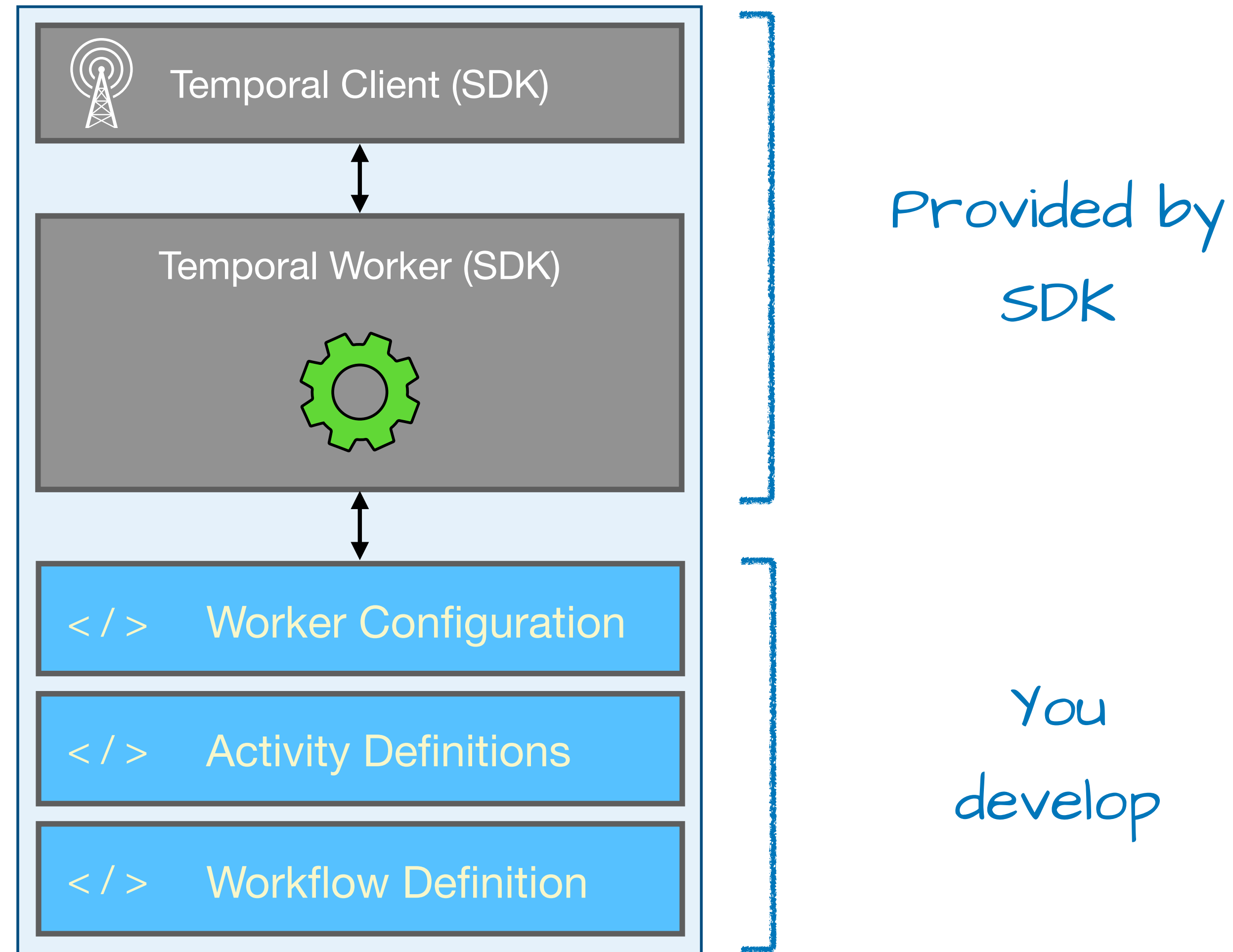
< / > Workflow Definition

Code You Develop



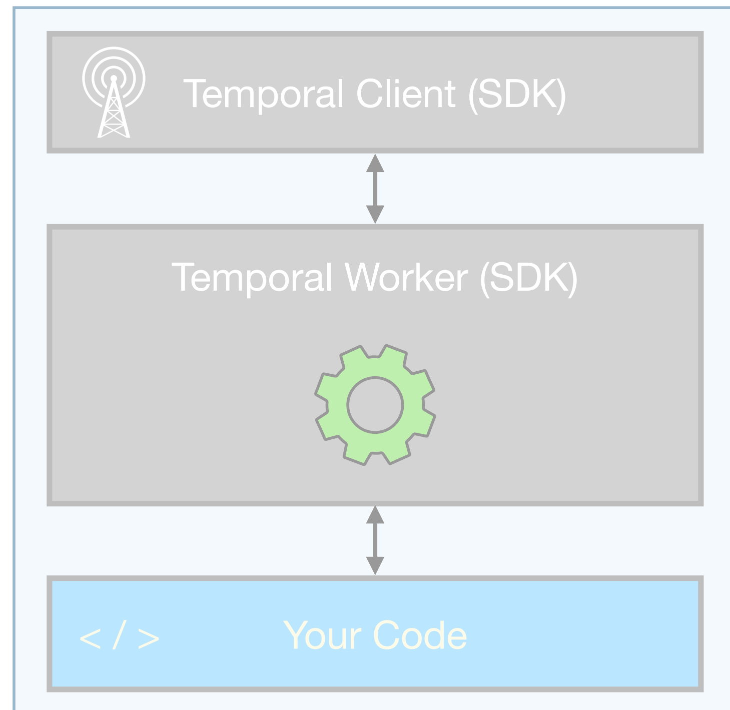
Temporal Application Code

A Complete Temporal Application

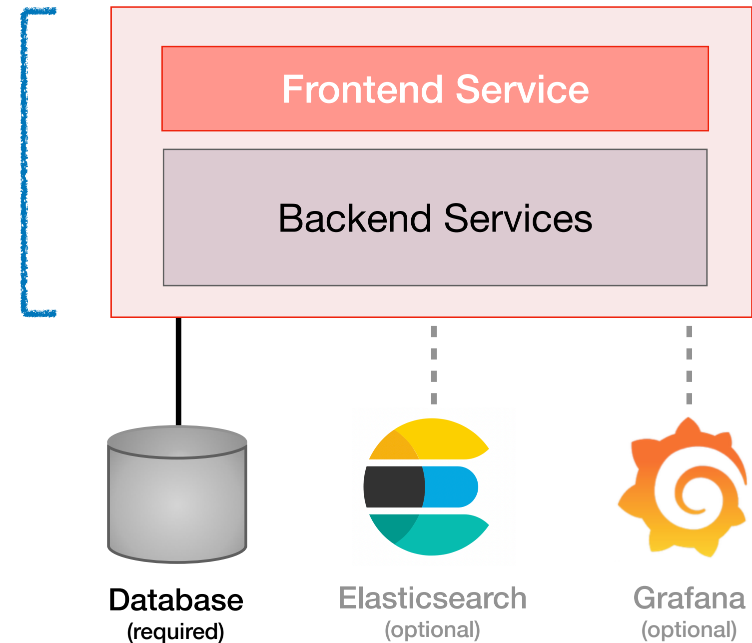


The Role of Temporal Cluster

Temporal Application



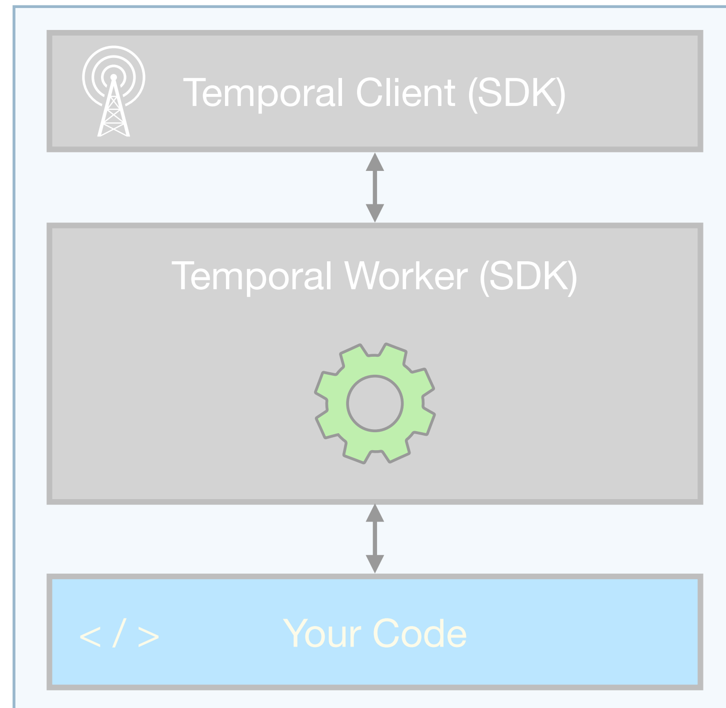
Temporal Cluster



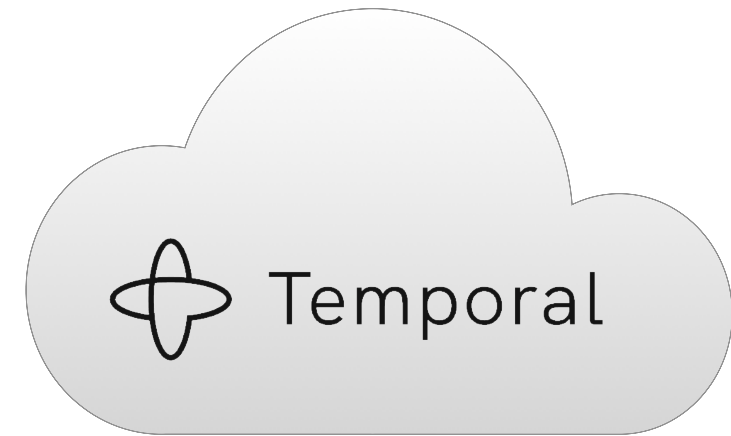
Temporal Server

The Role of Temporal Cloud

Temporal Application

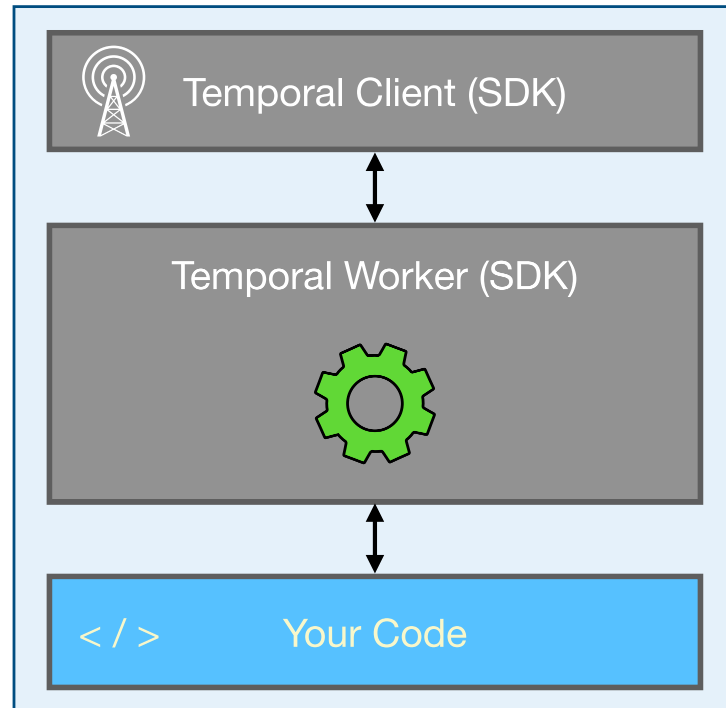


Temporal Cloud



Applications Are External to the Cluster

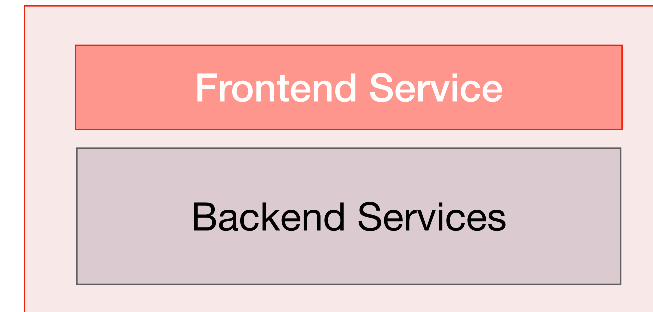
Temporal Application



Execution

Orchestration

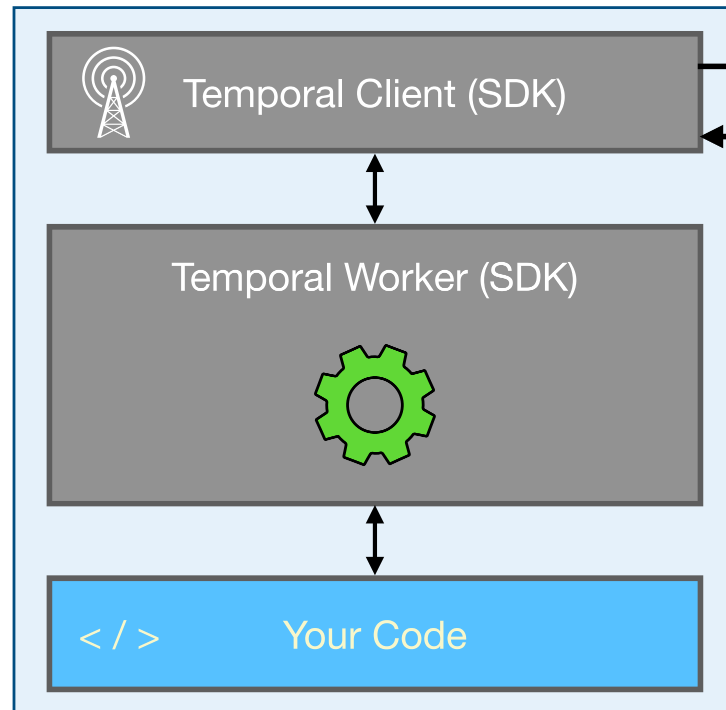
Temporal Cluster or Cloud



Temporal Uses gRPC for Communication

Temporal Application

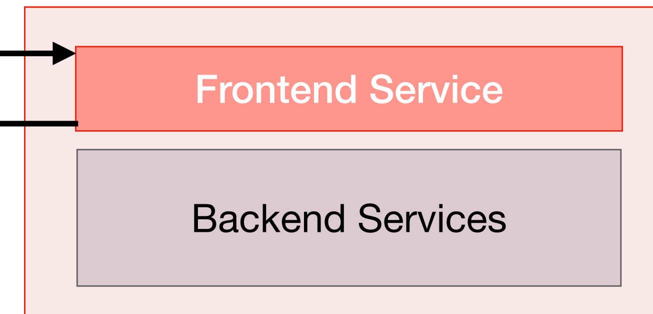
Temporal Cluster or Cloud



Request

Port 7233

Response



Temporal 102

00. About this Workshop
01. Understanding Key Concepts in Temporal
▶ **02. Improving Your Temporal Application Code**
03. Using Timers in a Workflow Definition
04. Testing Your Temporal Application Code
05. Understanding Event History
06. Debugging Workflow Execution
07. Deploying Your Application to Production
08. Understanding Workflow Determinism
09. Conclusion

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**

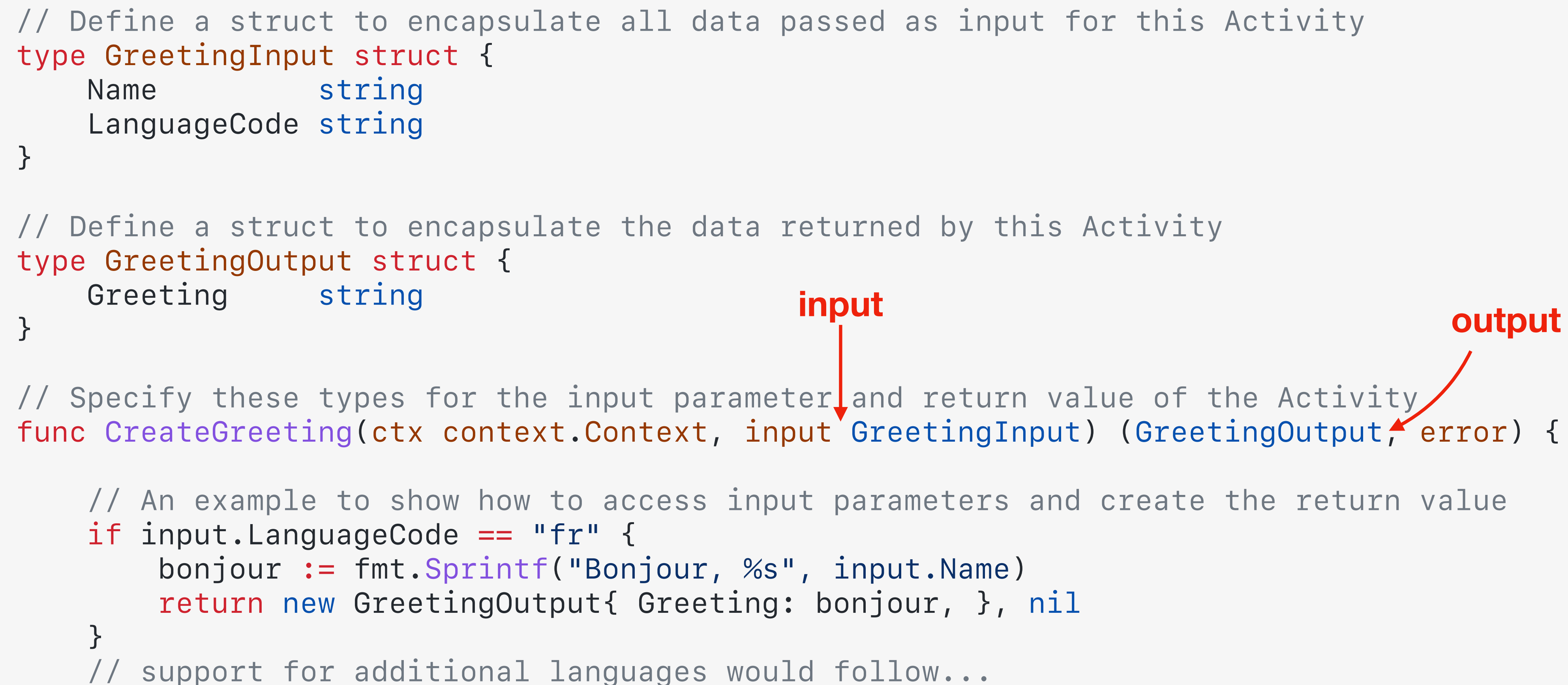
```
// This Activity returns a customized greeting in English, using the provided name
func CreateGreeting(ctx context.Context, name string) (string, error) {
   // implementation omitted for brevity
```

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...
```

Exercise #1: Using Structs for Data

- **During this exercise, you will**
  - Examine how the Workflow uses structs for input parameters and return values
  - Define structs to represent input and output of an Activity Definition
  - Update the code to use the structs you've defined for the Activity
  - Run the Workflow to ensure that it works as expected
- **Refer to this exercise's `README.md` file for details**
  - Don't forget to make your changes in the `practice` subdirectory

Task Queues

- **Temporal Clusters coordinate 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{})
```

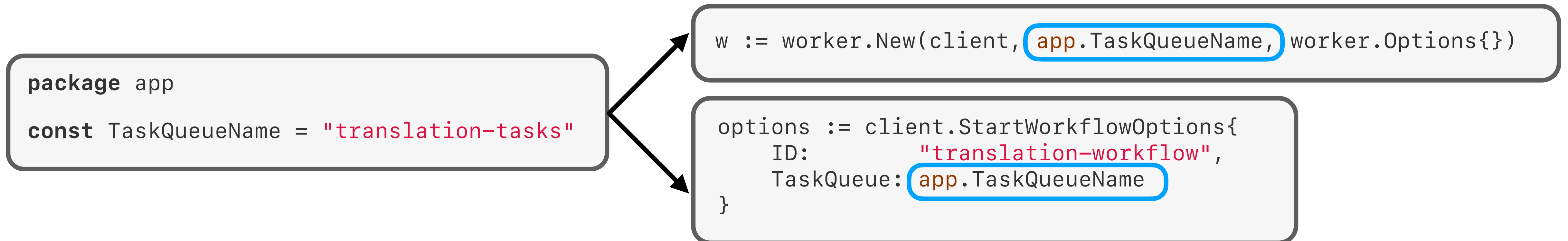
  - The Task Queue name is also specified by a Client when starting a Workflow

```
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")
```

The Default Logging Implementation

- **Limitations of the default logger**
  - Writes messages to the Client's standard output stream
  - Does not support setting a minimum level
  - Does not support customizing the output

Integrating Another Logging Implementation

- **You can integrate a different logging system into Temporal**
  - Provide this when creating a Client, via the `Logger` attribute in `client.Options`

```
// Override the default implementation and use some other logger
customLogger := NewAlternateLogger()
c, err := client.Dial(client.Options{
    Logger: customLogger,
})
```

- **It must conform to the Go SDK's `log.Logger` interface**
  - It's common to use an adapter to integrate with third-party logging packages
  - See the `zapadapter` subdirectory in the `samples-go` repository

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 simply submit *execution requests* to the cluster
  - They do not block while waiting for execution to complete

```
// Use a client to request Workflow execution
client.ExecuteWorkflow(context.Background(), options, MyWorkflow, input)
```

```
// Request Activity Execution from within a Workflow
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)
```

Temporal 102

00. About this Workshop

01. Understanding Key Concepts in Temporal

02. Improving Your Temporal Application Code

▶ **03. Using Timers in a Workflow Definition**

04. Testing Your Temporal Application Code

05. Understanding Event History

06. Debugging Workflow Execution

07. Deploying Your Application to Production

08. Understanding Workflow Determinism

09. Conclusion

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 (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 Cluster**
  - 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**
  - Don't forget to make your changes in the `practice` subdirectory

Temporal 102

00. About this Workshop
01. Understanding Key Concepts in Temporal
02. Improving Your Temporal Application Code
03. Using Timers in a Workflow Definition
▶ **04. Testing Your Temporal Application Code**
05. Understanding Event History
06. Debugging Workflow Execution
07. Deploying Your Application to Production
08. Understanding Workflow Determinism
09. Conclusion

Overview: Unit Testing in Go

- **Testing validates that your code behaves as intended**
  - Unit tests are automated tests that verify a "unit" of code in isolation
  - For example, you could write unit tests to verify that a function works correctly

```go
func Sum(first int, second int) int {
        return first + second
}
```

- **Go's `testing` package provides built-in support for unit testing**

Overview: Writing a Unit Test

- **Here is an example of a unit test for the `Sum` function you just saw**
  - The name of test functions *must* begin with `Test`

```
func TestSum(t *testing.T) {
        result := Sum(2, 5)
        expected := 7

        if result != expected {
                t.Fatalf("got %d, but expected %d", result, expected)
        }
}
```

- **This file should have same package and path as code under test**
  - Its file name should end with `_test.go` (e.g., `addition_test.go`)

Overview: Running Unit Tests

- **Use the `go test` command to run all tests in the current directory**
  - The output lists any failures, as well as the final result

```
$ go test

PASS
ok          example/testing      0.108s
```

- **Adding the `-v` option will list each test executed**

```
$ go test -v

=== RUN   TestSum
--- PASS: TestSum (0.00s)
PASS
ok          example/testing      0.121s
```

The `testify` Library

- **Go's built-in testing support is basic**
  - It lacks features such as assertions, mock objects, and test suites
- **The open source `testify` library adds these features**
  - You will execute your tests the same way, whether or not you use `testify`
- **This example illustrates how assertions reduce code needed for a unit test**
  - Replace `assert` with `require` if you want halt all tests upon the first failure

```
func TestSum(t *testing.T) {
    assert.Equal(t, 7, Sum(2, 5))
}
```

Validating Correctness of Temporal Application Code

- **Go SDK's `testsuite` package supports Workflow and Activity testing**
- **Most tests involve some combination of three types from this package**
  - `TestWorkflowEnvironment`
    - Provides a runtime environment used to test a Workflow
    - Some aspects of its execution will work differently to better support testing (e.g., time skipping)
  - `TestActivityEnvironment`
    - Provides a runtime environment used to test Activities
  - `WorkflowTestSuite`
    - A test suite used to define a collection of tests, and optionally, `Setup` and `TearDown` functions

Activity Definition Example

- **Imagine that you have written the following Activity Definition**

```go
package example

import (
	"context"

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

func SquareActivity(ctx context.Context, number int) (int, error) {
	logger := activity.GetLogger(ctx)
	logger.Info("Preparing to calculate the square")

	result := number * number

	logger.Debug("Finished calculating the square")
	return result, nil
}
```

Activity Test Example (Slide 1 of 3)

- **The following code can verify that it behaves as expected**
  - Observe the packages imported: Go's `testing`, Testify's `assert`, and SDK's `testsuite`

```
package example

import (
        "testing"

        "github.com/stretchr/testify/assert"
        "go.temporal.io/sdk/testsuite"
)

// code continues on next slide
```

Activity Test Example (Slide 2 of 3)

- **The following code verifies that the Activity works with positive numbers**

```
// code continued from previous slide

func Test_SquareActivityPositive(t *testing.T) {
	testSuite := &testsuite.WorkflowTestSuite{}
	env := testSuite.NewTestActivityEnvironment()
	env.RegisterActivity(SquareActivity)

	val, err := env.ExecuteActivity(SquareActivity, 3)
	assert.NoError(t, err)

	var result int
	assert.NoError(t, val.Get(&result))
	assert.Equal(t, 9, result)
}

// code continues on next slide
```

Activity Test Example (Slide 3 of 3)

- **The following code verifies that the Activity works with negative numbers**

```
// code continued from previous slide

func Test_SquareActivityNegative(t *testing.T) {
        testSuite := &testsuite.WorkflowTestSuite{}
        env := testSuite.NewTestActivityEnvironment()
        env.RegisterActivity(SquareActivity)

        val, err := env.ExecuteActivity(SquareActivity, -4)
        assert.NoError(t, err)

        var result int
        assert.NoError(t, val.Get(&result))
        assert.Equal(t, 16, result)
}
```

Workflow Definition Example (Slide 2 of 2)

- **Imagine that you have written the following Workflow Definition**

```go
package example

import (
	"time"

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

func SumOfSquaresWorkflow(ctx workflow.Context, numbers PairOfInts) (int, error) {
	options := workflow.ActivityOptions{
		StartToCloseTimeout: time.Second * 5,
	}
	ctx = workflow.WithActivityOptions(ctx, options)

	n1 := numbers.First
	n2 := numbers.Second

	// code continues on next slide
```

Workflow Definition Example (Slide 2 of 2)

- **Imagine that you have written the following Workflow Definition**

```
        // code continued from previous slide

        var squareOne int
        err := workflow.ExecuteActivity(ctx, SquareActivity, n1).Get(ctx, &squareOne)
        if err != nil {
                return -1, err
        }

        var squareTwo int
        err = workflow.ExecuteActivity(ctx, SquareActivity, n2).Get(ctx, &squareTwo)
        if err != nil {
                return -1, err
        }

        result := squareOne + squareTwo

        return result, nil
}
```

Workflow Test Example (Slide 1 of 3)

- **The following code can verify that the Workflow behaves as expected**
  - The same packages are imported as with the Activity Definition test

```
package example

import (
        "testing"

        "github.com/stretchr/testify/assert"
        "go.temporal.io/sdk/testsuite"
)

// code continues on next slide
```

Workflow Test Example (Slide 2 of 3)

```go
// code continued from previous slide

func Test_SumOfSquaresWorkflowPositive(t *testing.T) {
	testSuite := &testsuite.WorkflowTestSuite{}
	env := testSuite.NewTestWorkflowEnvironment()
	env.RegisterActivity(SquareActivity)

	input := PairOfInts{
		First:  2,
		Second: 5,
	}

	env.ExecuteWorkflow(SumOfSquaresWorkflow, input)
	assert.True(t, env.IsWorkflowCompleted())
	assert.NoError(t, env.GetWorkflowError())

	var result int
	assert.NoError(t, env.GetWorkflowResult(&result))
	assert.Equal(t, 29, result)
}

// code continued on next slide
```

Workflow Test Example (Slide 3 of 3)

```go
// code continued from previous slide

func Test_SumOfSquaresWorkflowNegative(t *testing.T) {
	testSuite := &testsuite.WorkflowTestSuite{}
	env := testSuite.NewTestWorkflowEnvironment()
	env.RegisterActivity(SquareActivity)

	input := PairOfInts{
		First:  -3,
		Second: 7,
	}

	env.ExecuteWorkflow(SumOfSquaresWorkflow, input)
	assert.True(t, env.IsWorkflowCompleted())
	assert.NoError(t, env.GetWorkflowError())

	var result int
	assert.NoError(t, env.GetWorkflowResult(&result))
	assert.Equal(t, 58, result)
}
```

Mocking Activities for Workflow Tests

- **The previous Workflow invoked two Activities**
  - These Activities were called as part of the test
  - Therefore, the Workflow is tightly coupled to the Activity implementation
- **Using mock Activities allows you to test your Workflow logic in isolation**
  - This is a function used in place of the Activity during the test
  - Mocking also makes it easier to set up and test unusual conditions
    - For example, if the Activity returns a certain type of error

Workflow Test Example - No Mock (Slide 1 of 2)

- **Scenario: A Workflow uses an Activity to call a microservice that estimates someone's age based on their first name**

```go
package example

import (
	"testing"

	"github.com/stretchr/testify/assert"
	"go.temporal.io/sdk/testsuite"
)

// code continues on next slide
```

Workflow Test Example - No Mock (Slide 2 of 2)

```go
// code continued from previous slide

func Test_EstimateAge_EndToEnd(t *testing.T) {
        testSuite := &testsuite.WorkflowTestSuite{}
        env := testSuite.NewTestWorkflowEnvironment()
        env.RegisterActivity(RetrieveEstimate)

        // Provide a name as input to the Workflow. This value is
        // passed to the Activity, which calls the remote API
        env.ExecuteWorkflow(EstimateAge, "Betty")
        assert.True(t, env.IsWorkflowCompleted())

        // The Workflow combines the name and age into a message,
        // and the code below verifies that it is as expected
        var result string
        assert.NoError(t, env.GetWorkflowResult(&result))
        expected := "Betty has an estimated age of 76"
        assert.Equal(t, expected, result)
}
```

Workflow Test Example - with Mock (Slide 1 of 2)

- **Same scenario as before, but this time using a mock Activity**

```go
package example

import (
	"testing"

	"github.com/stretchr/testify/assert"
	"github.com/stretchr/testify/mock"
	"go.temporal.io/sdk/testsuite"
)

// code continues on next slide
```

Workflow Test Example - with Mock (Slide 2 of 2)

```
// code continued from previous slide

func Test_EstimateAge_WithMockActivity(t *testing.T) {
        testSuite := &testsuite.WorkflowTestSuite{}
        env := testSuite.NewTestWorkflowEnvironment()

        // Define a mock Activity that returns an estimated age for Betty
        env.OnActivity(RetrieveEstimate, mock.Anything, "Betty").Return(76, nil)

        env.ExecuteWorkflow(EstimateAge, "Betty")
        assert.True(t, env.IsWorkflowCompleted())

        var result string
        assert.NoError(t, env.GetWorkflowResult(&result))
        expected := "Betty has an estimated age of 76"
        assert.Equal(t, expected, result)
}
```

Exercise #3: Testing the Translation Workflow

- **During this exercise, you will**
  - Write code to execute the Workflow in the test environment
  - Develop a Mock Activity for the translation service call
  - Observe time-skipping in the test environment
  - Write unit tests for the Activity implementation
  - Run the tests from the command line to verify correct behavior
- **Refer to this exercise's `README.md` file for details**
  - Don't forget to make your changes in the `practice` subdirectory

Temporal 102

00. About this Workshop
01. Understanding Key Concepts in Temporal
02. Improving Your Temporal Application Code
03. Using Timers in a Workflow Definition
04. Testing Your Temporal Application Code
▶ **05. Understanding Event History**
06. Debugging Workflow Execution
07. Deploying Your Application to Production
08. Understanding Workflow Determinism
09. Conclusion

```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
}
```

Workflow Definition

combined with

+

Execution Request

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

results in

=

Workflow Execution

Running Workflow

```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
}
```

1 Workflow Definition

combined with

n Execution Requests

```
client.ExecuteWorkflow(..., {"ID": 812} )
client.ExecuteWorkflow(..., {"ID": 947} )
```

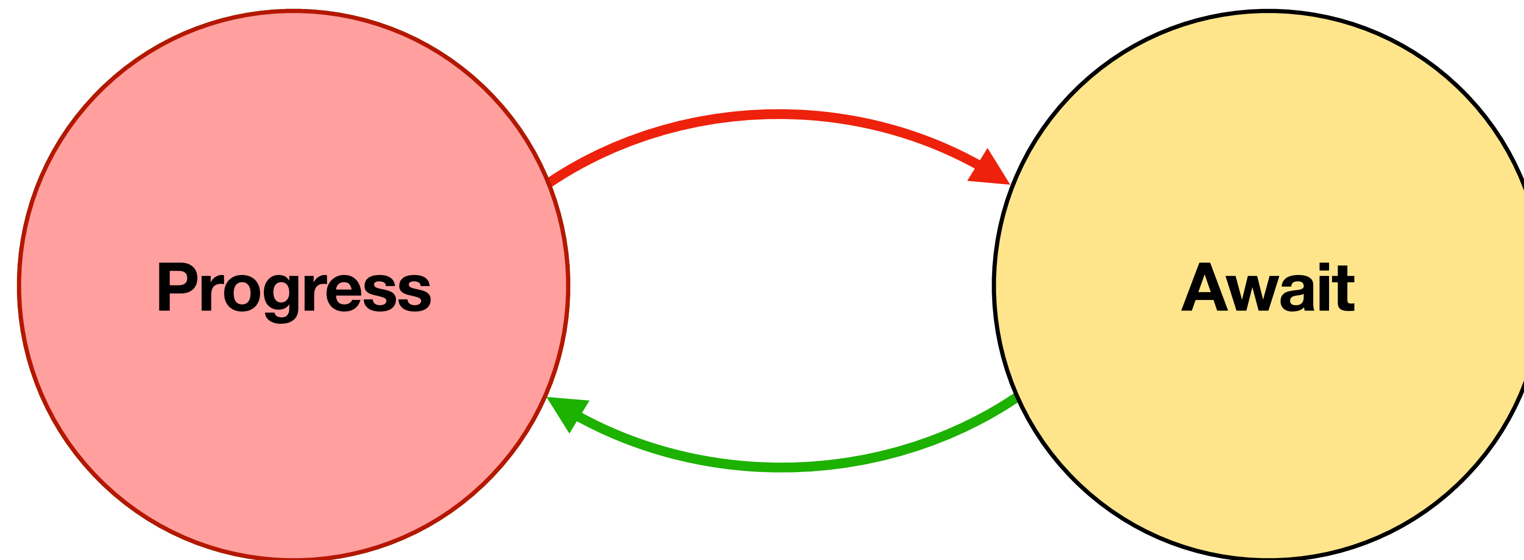
results in

n Workflow Executions

Workflow Execution 1

Workflow Execution 2

What Happens During Workflow Execution



This cycle continues throughout Workflow Execution

Workflow Execution States

This is a one-way transition

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

Basic Temporal Workflow Definition

```
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
}
```

- A Workflow is a sequence of steps
- Some steps are *internal to the Workflow*
  - Do not involve interaction with the Cluster
- 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

- Other steps *do* involve interaction with the cluster
- 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

```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
}
```

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 Temporal Cluster's perspective
  - Durably persisted by the Temporal Cluster
- **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 the Temporal Cluster 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)

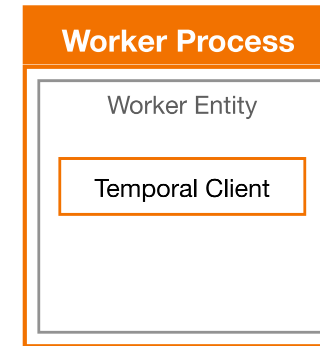
Attributes Vary by Event Type

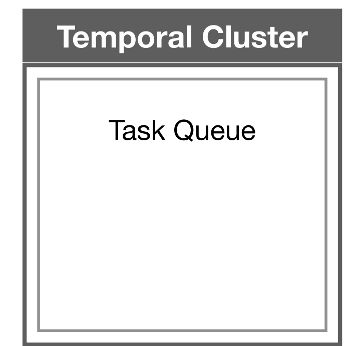
- **Additionally, each Event contains attributes specific to its type**
  - `WorkflowExecutionStarted` contains the Workflow Type and input parameters
  - `WorkflowExecutionCompleted` contains the result returned by the Workflow function
  - `WorkflowExecutionFailed` contains the error returned by the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

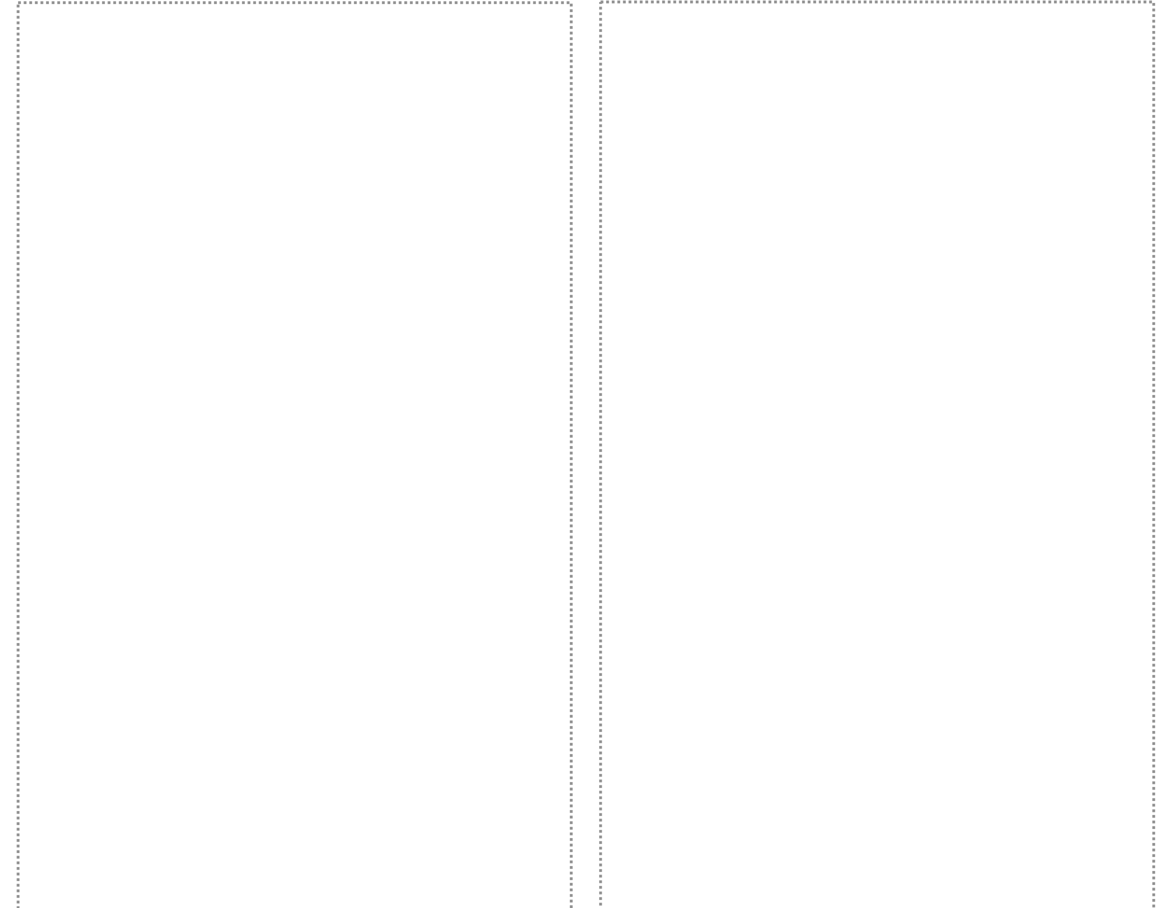
How Commands Map to Events

pseudocode

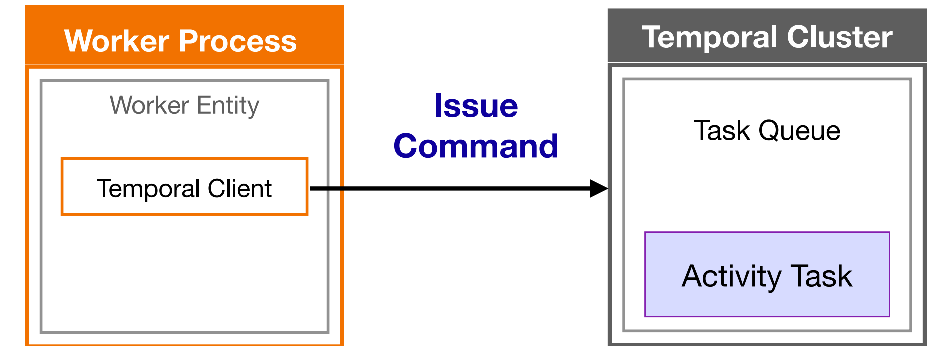
```
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   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
}
```







```
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
}
```



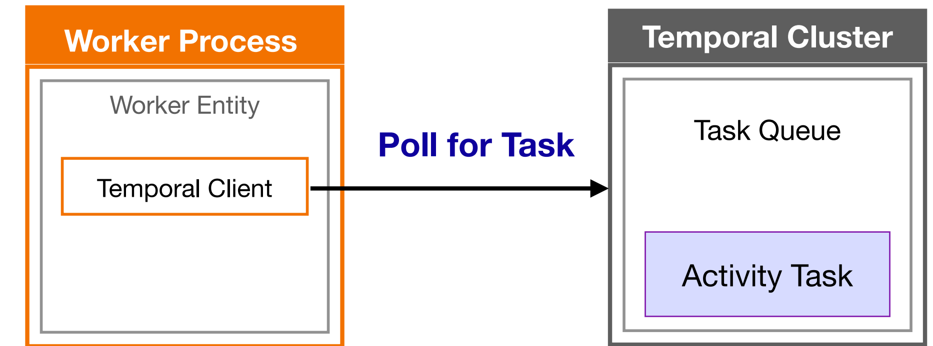
Commands

ScheduleActivityTask (GetDistance)

Events

ActivityTaskScheduled

```
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
}
```



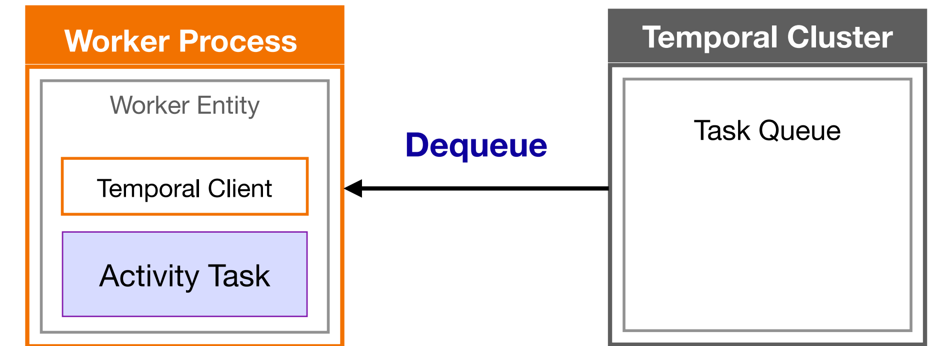
Commands

ScheduleActivityTask
(GetDistance)

Events

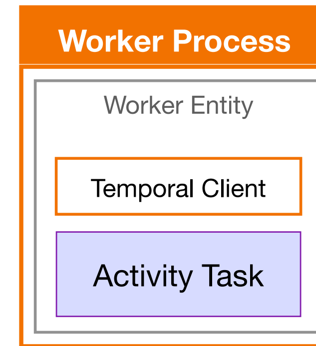
ActivityTaskScheduled

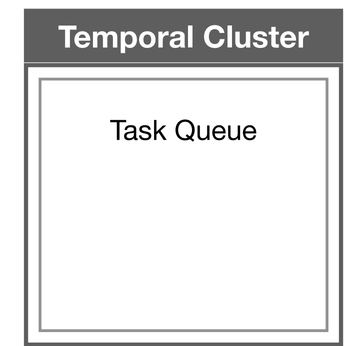
```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   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	Events
ScheduleActivityTask (GetDistance)	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

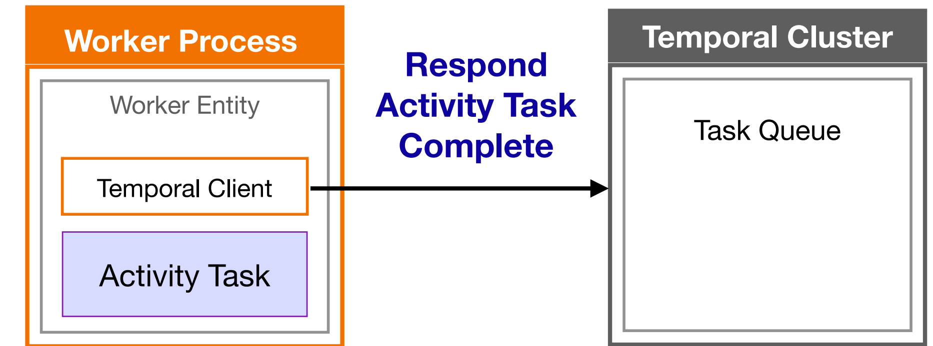
ScheduleActivityTask
(GetDistance)

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)

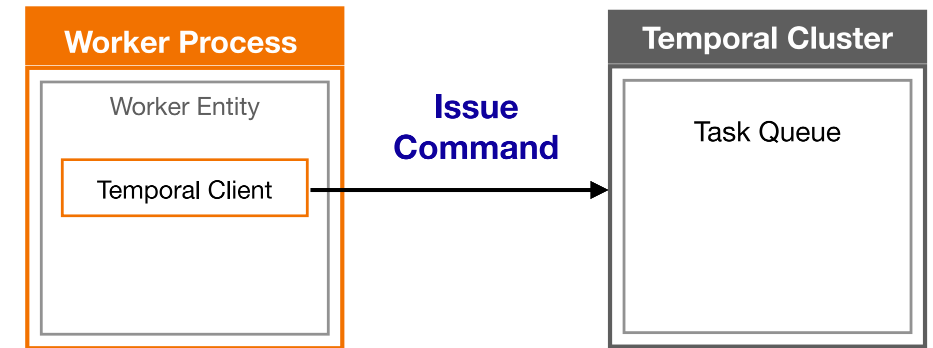
Events

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)

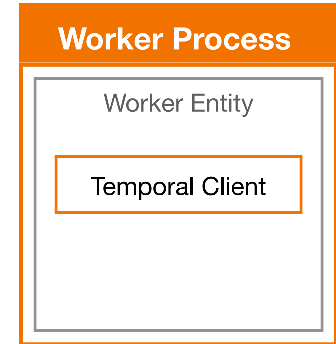
Events

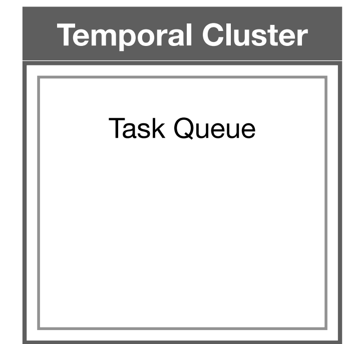
ActivityTaskScheduled

ActivityTaskStarted

ActivityTaskCompleted

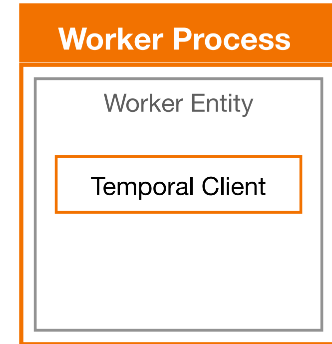
```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   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
}
```

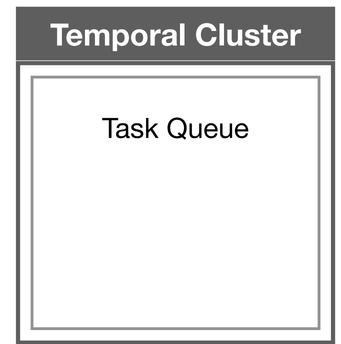




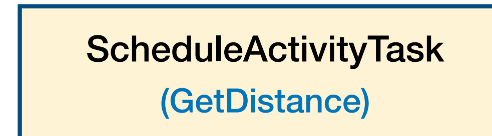
Commands	Events
ScheduleActivityTask (GetDistance)	ActivityTaskScheduled
	ActivityTaskStarted
	ActivityTaskCompleted
StartTimer (30 Minutes)	TimerStarted

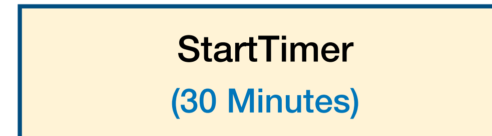
```
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   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
}
```



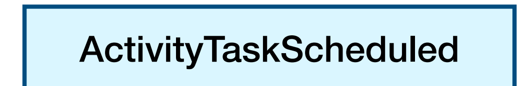


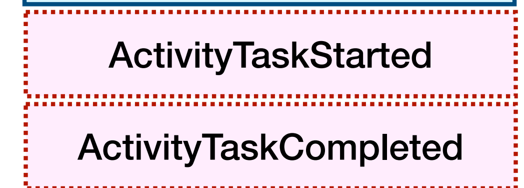
Commands

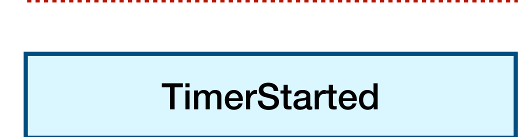




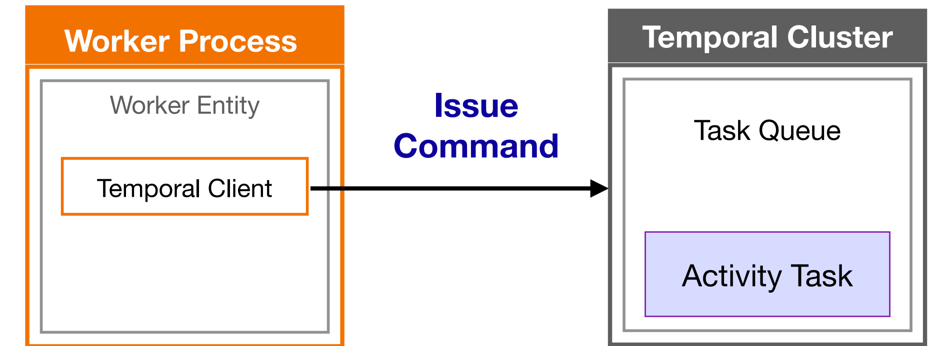
Events





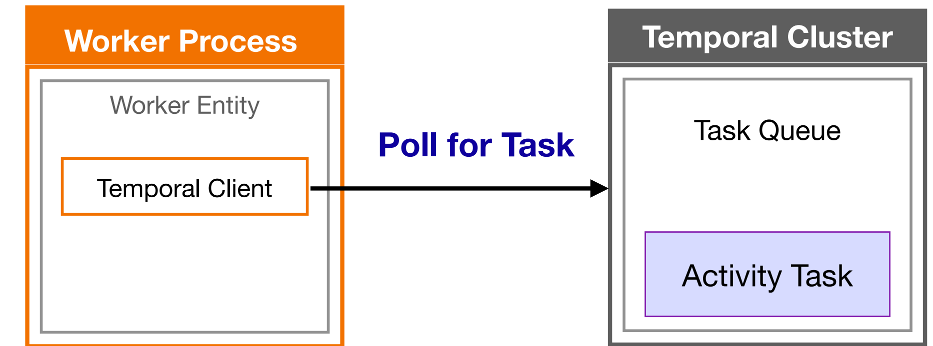


```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   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
}
```



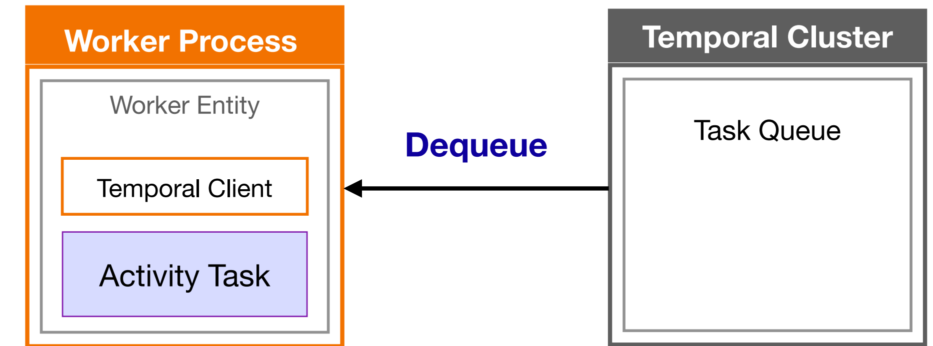
Commands	Events
ScheduleActivityTask (GetDistance)	ActivityTaskScheduled ActivityTaskStarted ActivityTaskCompleted
StartTimer (30 Minutes)	TimerStarted TimerFired
ScheduleActivityTask (SendBill)	ActivityTaskScheduled

```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	Events
ScheduleActivityTask (GetDistance)	ActivityTaskScheduled
	ActivityTaskStarted
	ActivityTaskCompleted
StartTimer (30 Minutes)	TimerStarted
	TimerFired
ScheduleActivityTask (SendBill)	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
}
```



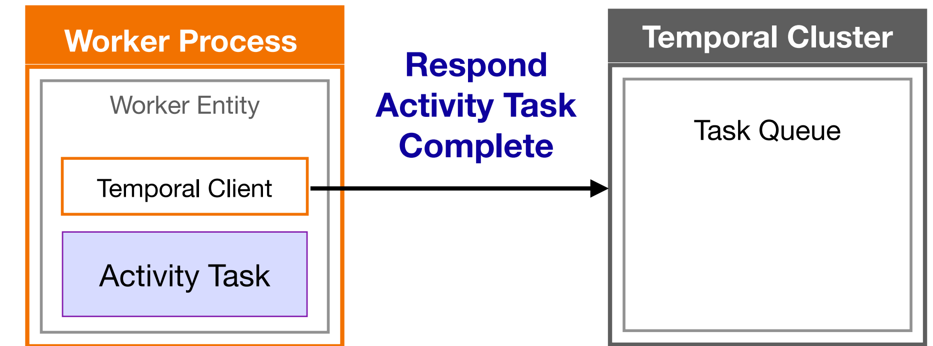
Commands

- ScheduleActivityTask (GetDistance)
- StartTimer (30 Minutes)
- ScheduleActivityTask (SendBill)

Events

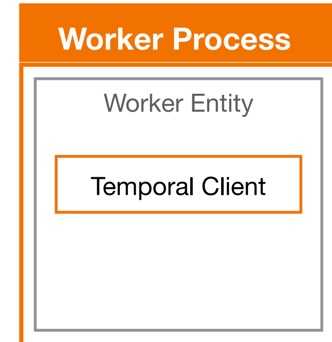
- ActivityTaskScheduled
- ActivityTaskStarted
- ActivityTaskCompleted
- TimerStarted
- TimerFired
- ActivityTaskScheduled
- ActivityTaskStarted

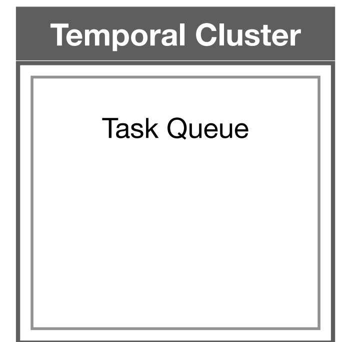
```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   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
}
```



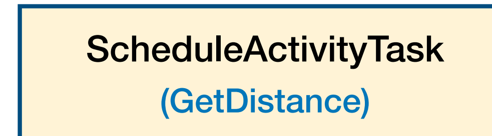
Commands	Events
ScheduleActivityTask (GetDistance)	ActivityTaskScheduled ActivityTaskStarted ActivityTaskCompleted
StartTimer (30 Minutes)	TimerStarted TimerFired
ScheduleActivityTask (SendBill)	ActivityTaskScheduled ActivityTaskStarted ActivityTaskCompleted

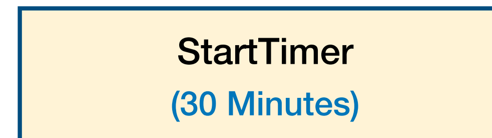
```go
func PizzaWorkflow(ctx workflow.Context, order Order) (string, error) {

   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
}
```

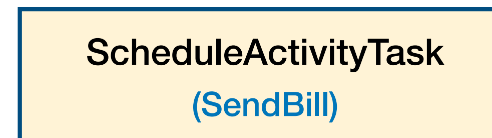




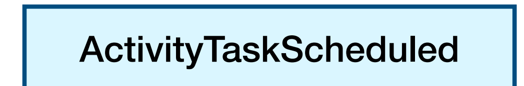
Commands

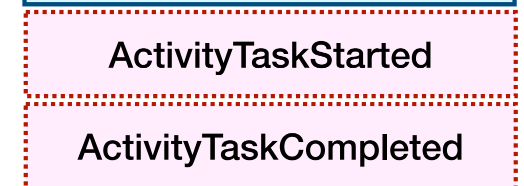


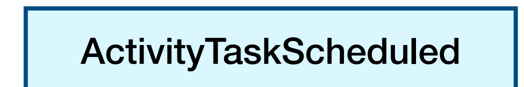


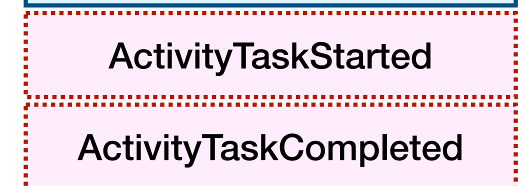


Events





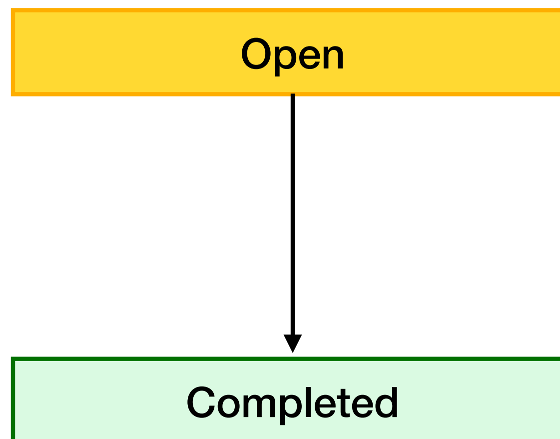




Workflow Execution States

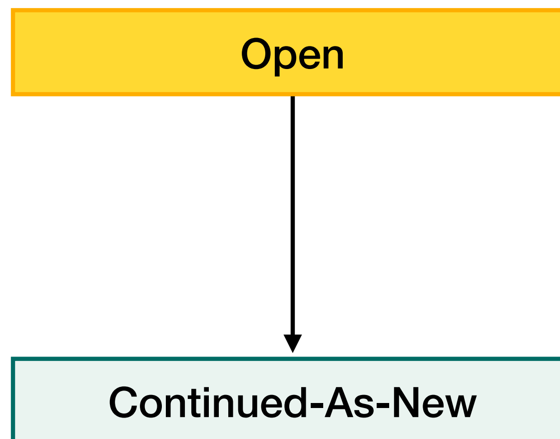
Completed

Meaning: The Workflow function returned a result



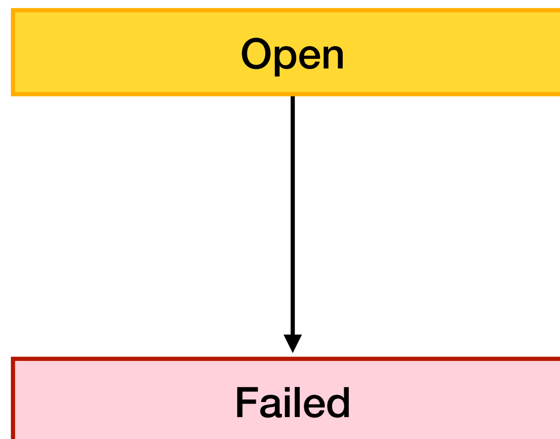
Continued-As-New

Meaning: Future progress will take place in a new Workflow Execution



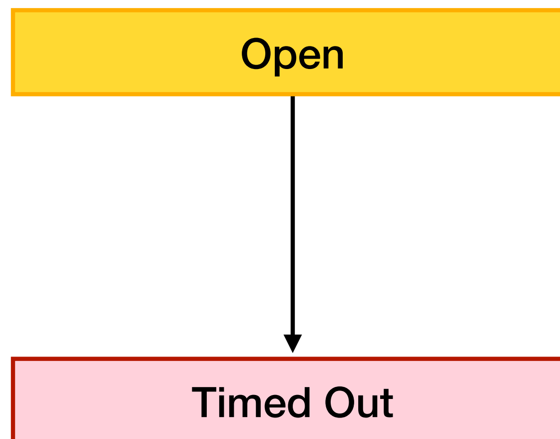
Failed

Meaning: The Workflow function returned an error



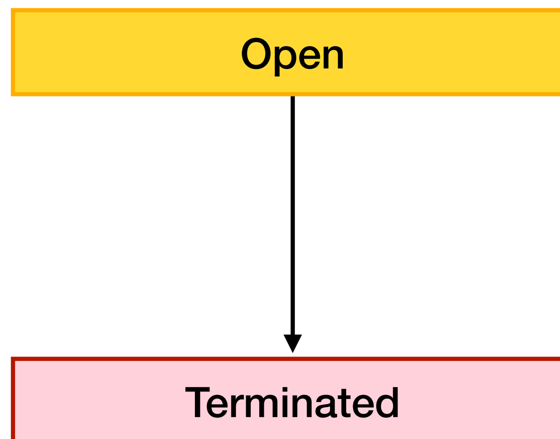
Timed Out

Meaning: Execution exceeded a specified time limit



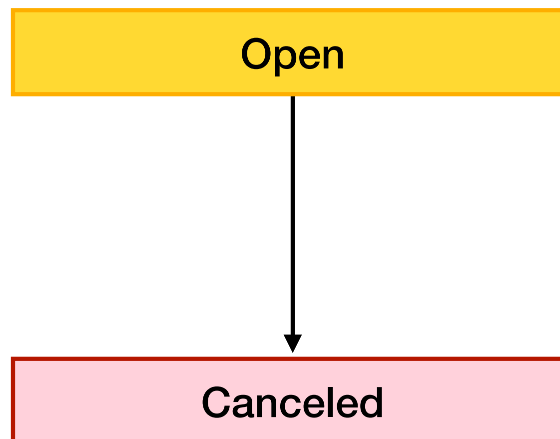
Terminated

Meaning: Temporal Cluster acted upon a termination request

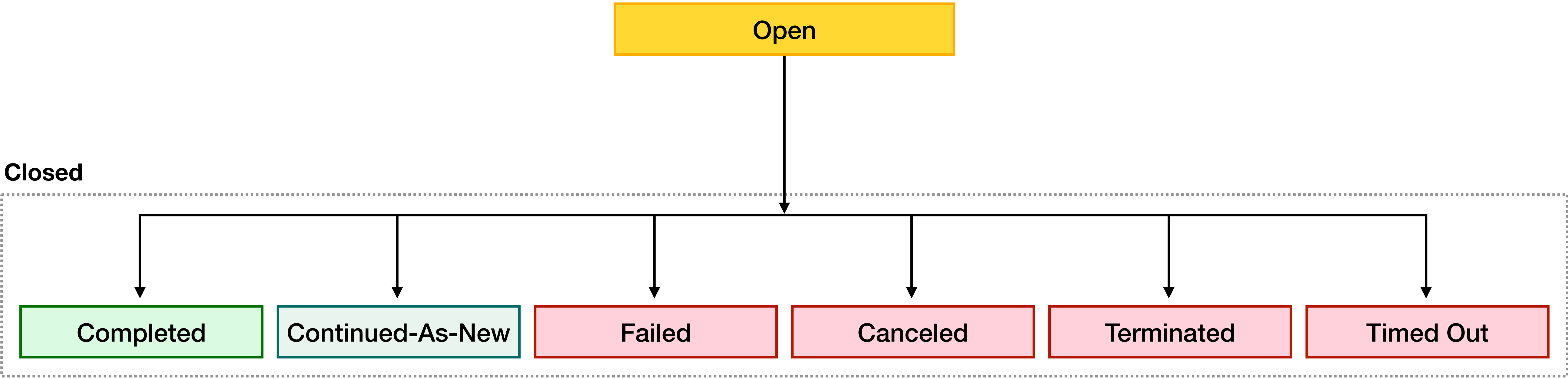


Canceled

Meaning: Temporal Cluster acted upon a request to cancel execution



Summary of Workflow Execution States



Workflow and Activity Task States

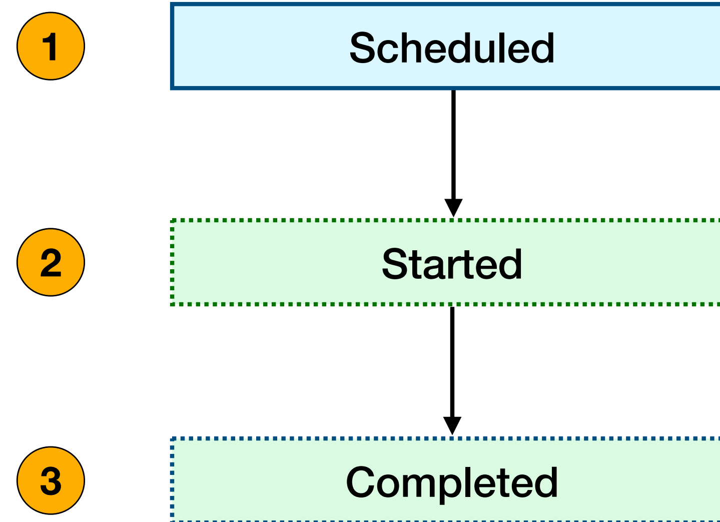
Activity Task Event Sequence

ActivityTaskScheduled

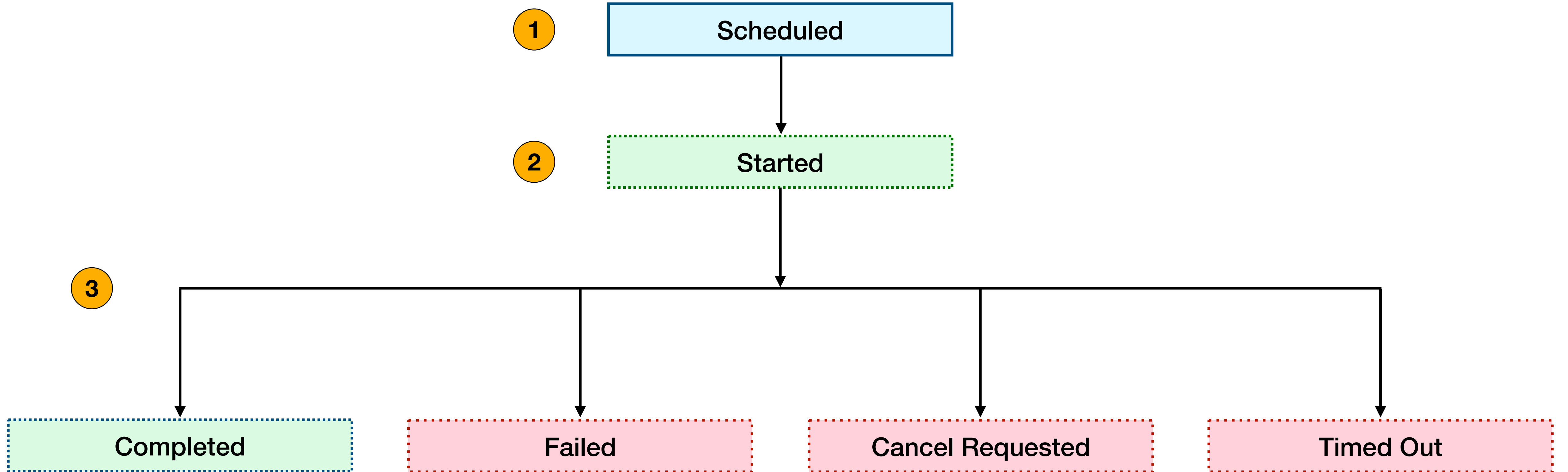
ActivityTaskStarted

ActivityTaskCompleted

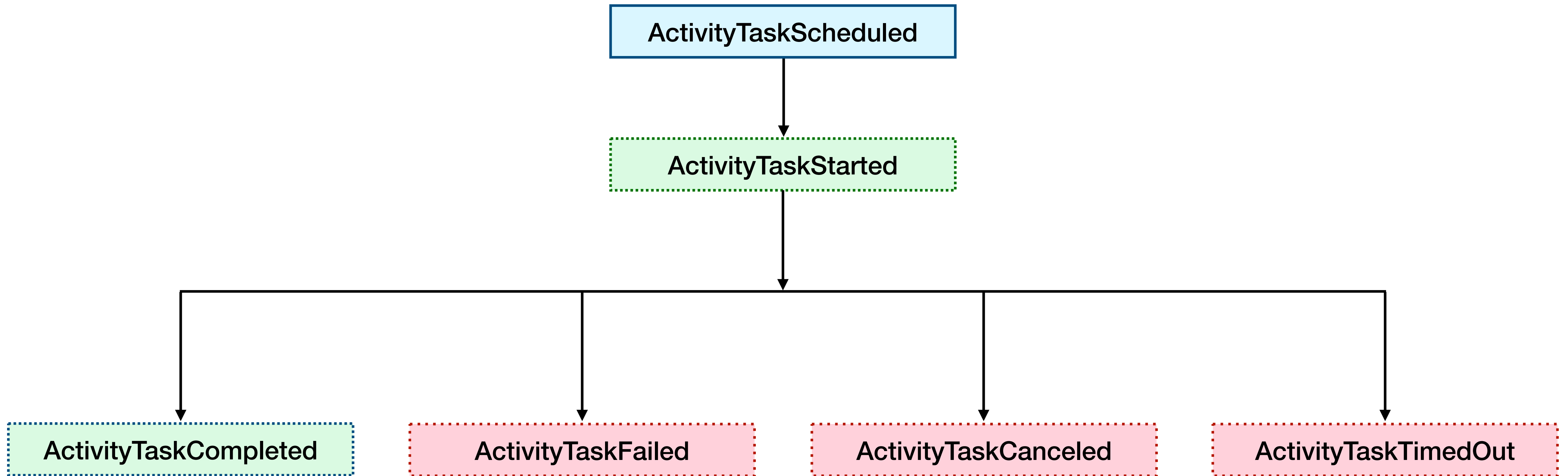
Activity States in that Sequence



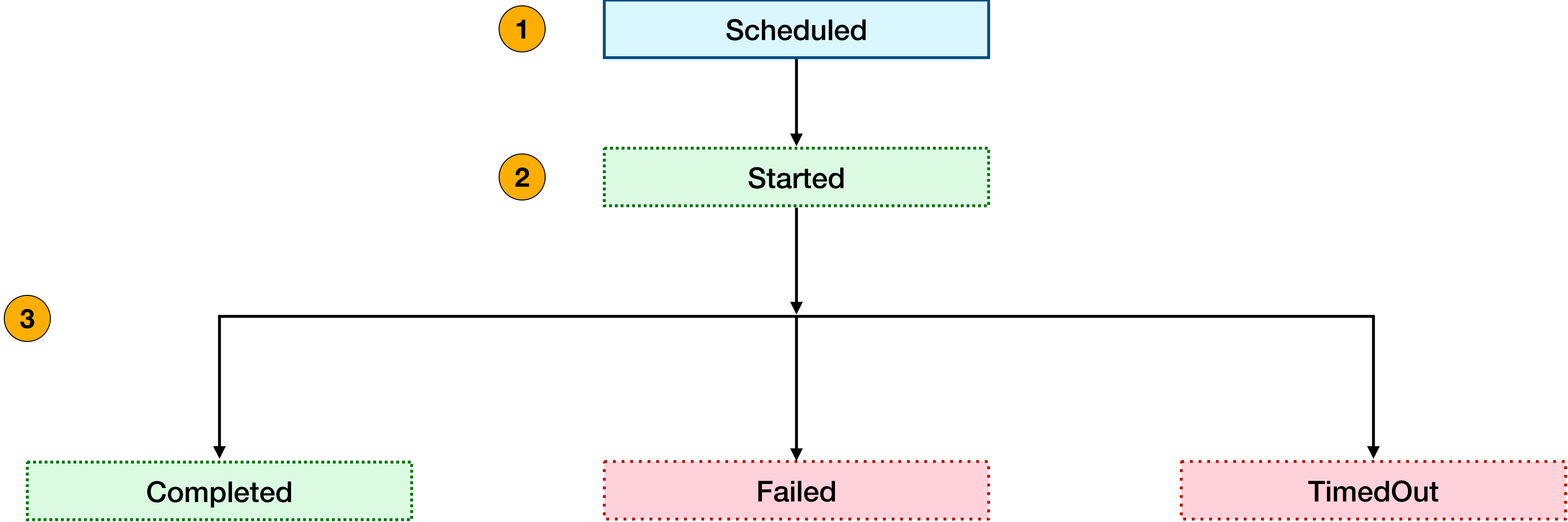
Activity Task States



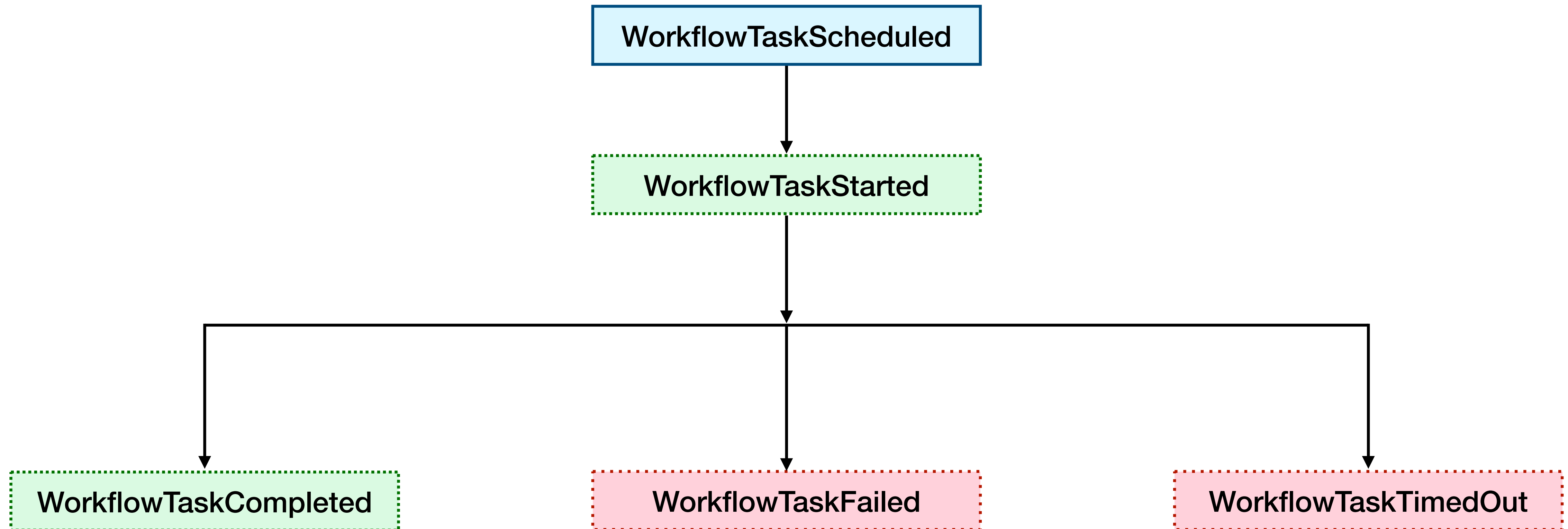
Activity Task Events



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

2	2023-07-19 UTC 17:02:31.35	WorkflowTaskScheduled
Summary	Task Queue	
Task Queue Name	durable-exec-tasks	
Task Queue Kind	Normal	

Later Workflow Task

8	2023-07-19 UTC 17:02:31.36	WorkflowTaskScheduled
Summary	Task Queue	
Task Queue Name	twwmbp:b7b2434d-4fb5-4ca6-b05f-bb98d6565a96	
Task Queue Kind	Sticky	
Task Queue Normal Name	durable-exec-tasks	

Temporal 102

00. About this Workshop
01. Understanding Key Concepts in Temporal
02. Improving Your Temporal Application Code
03. Using Timers in a Workflow Definition
04. Testing Your Temporal Application Code
05. Understanding Event History
▶ **06. Debugging Workflow History**
07. Deploying Your Application to Production
08. Understanding Workflow Determinism
09. Conclusion

Instructor-Led Demo #1

Debugging a Workflow that Does Not Progress

Instructor-Led Demo #2

Interpreting Event History for Workflow Executions

Instructor-Led Demo #3

Terminating a Workflow Execution with the Web UI

Instructor-Led Demo #4

Identifying and Fixing a Bug in an Activity Definition

Exercise #4: Debugging and Fixing an Activity Failure

- **During this exercise, you will**
  - Start a Worker and run a basic Workflow for processing a pizza order
  - Use the Web UI to find details about the execution
  - Diagnose and fix a latent bug in the Activity Definition
  - Test and deploy the fix
  - Verify that the Workflow now completes successfully
- **Refer to this exercise's `README.md` file for details**
  - Don't forget to make your changes in the `practice` subdirectory

Temporal 102

00. About this Workshop
01. Understanding Key Concepts in Temporal
02. Improving Your Temporal Application Code
03. Using Timers in a Workflow Definition
04. Testing Your Temporal Application Code
05. Understanding Event History
06. Debugging Workflow History
▶ **07. Deploying Your Application to Production**
08. Understanding Workflow Determinism
09. Conclusion

Temporal Cluster Services

Frontend

An API Gateway that validates and routes inbound calls

History

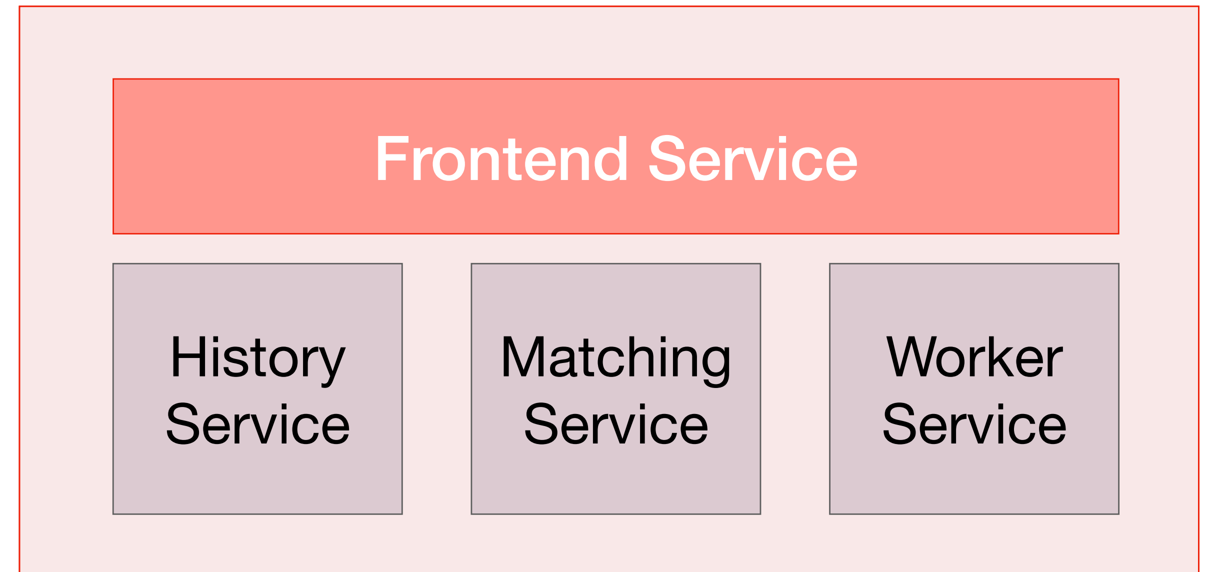
Maintains history and moves execution progress forward

Matching

Hosts Task Queues and matches Workers with Tasks

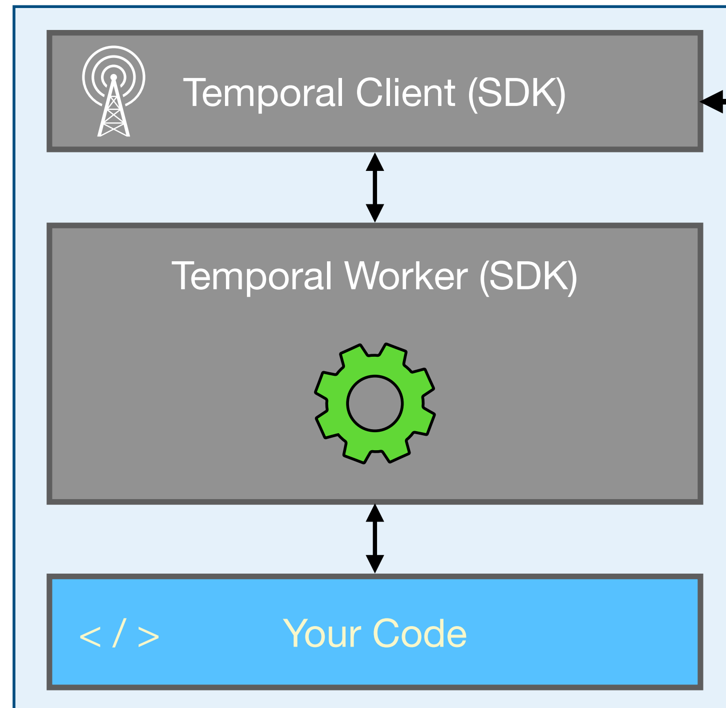
Worker Service

Runs internal system Workflows

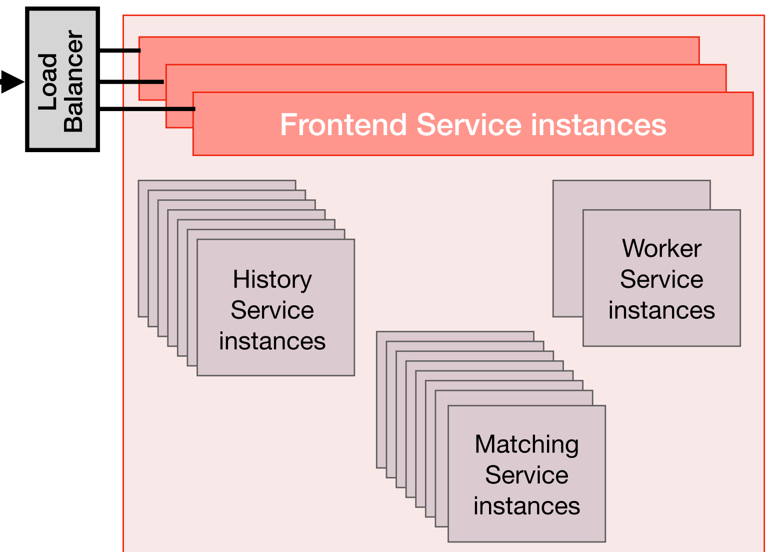


Cluster Scalability

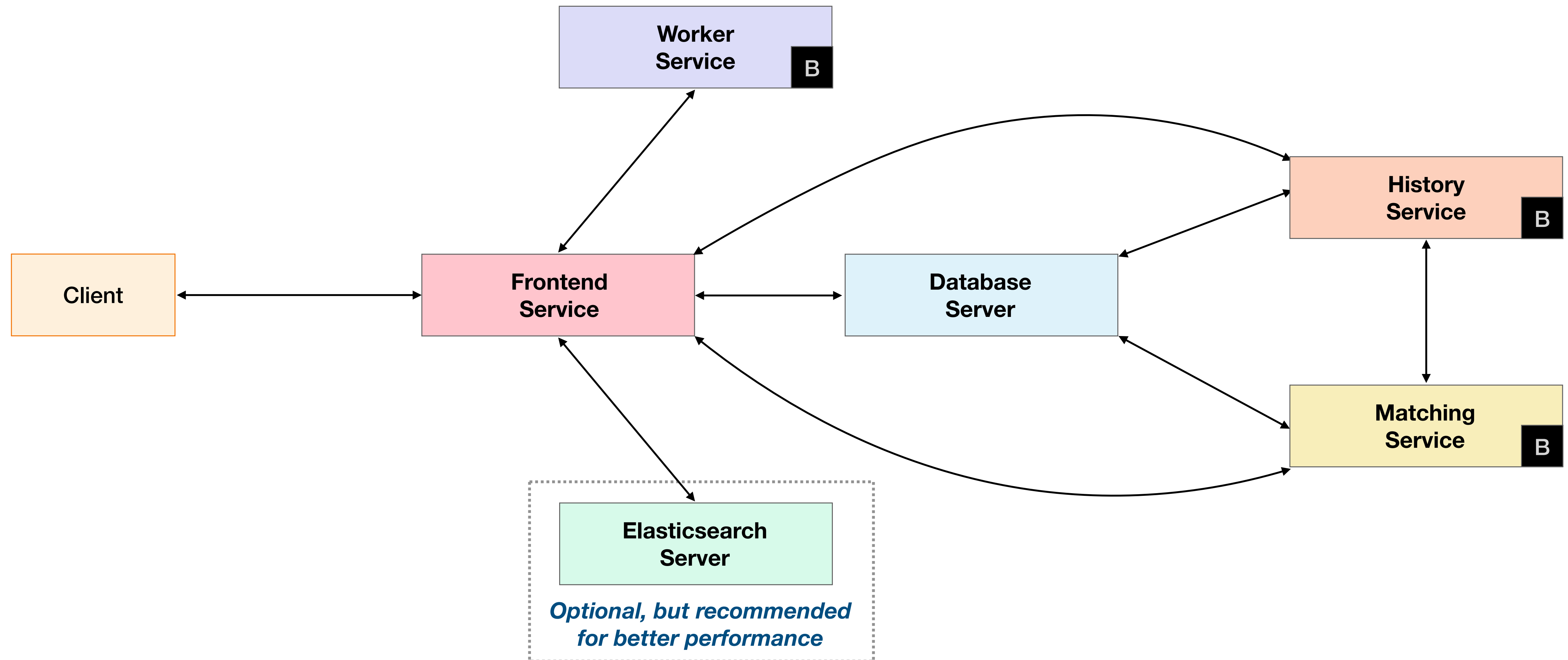
Temporal Application



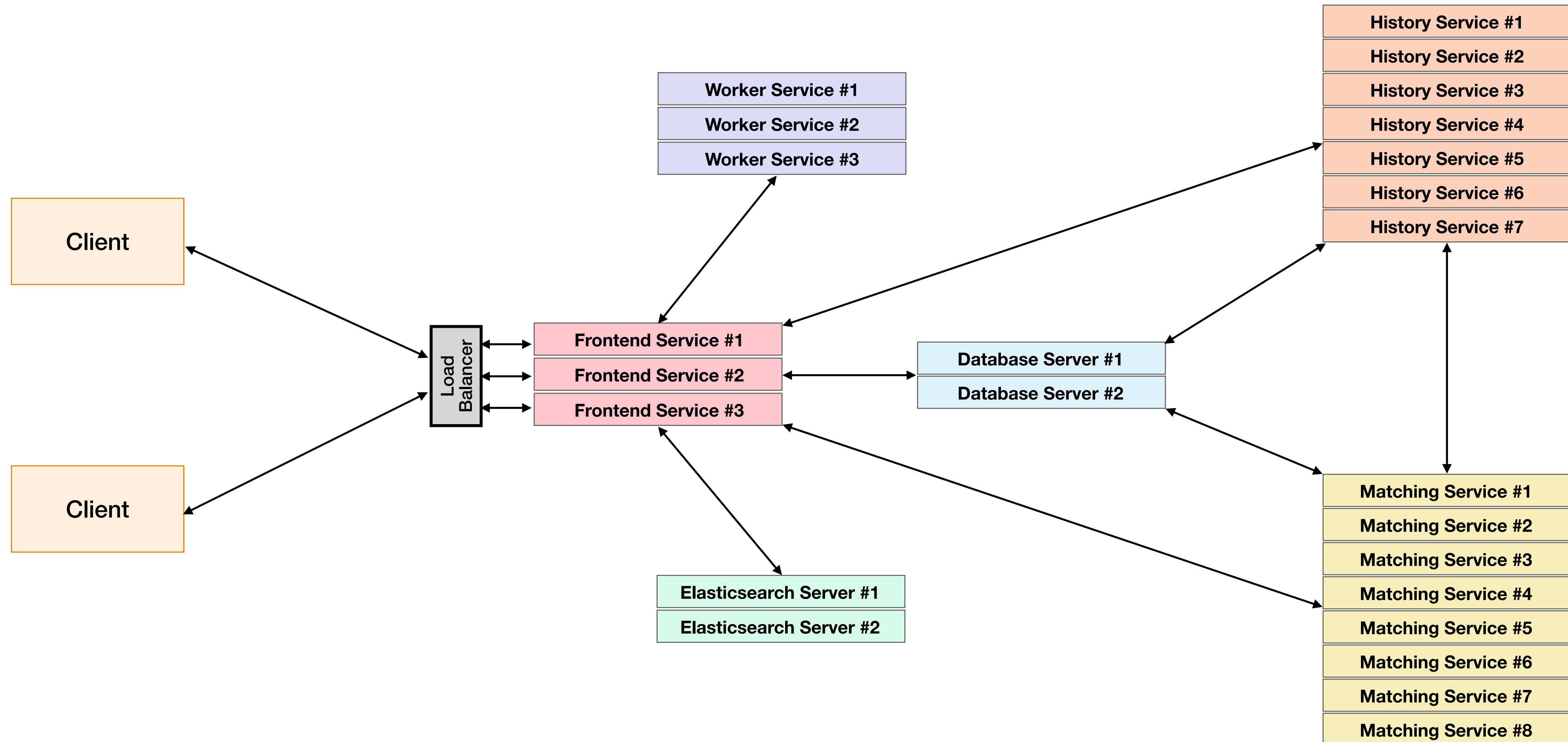
Temporal Cluster



Connectivity (Logical)



Connectivity (Physical)



Default Options for a Temporal Client

- **The following code example shows how to create a Temporal Client**
  - This will expects a Frontend Service running on `localhost` at TCP port 7233

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

Customizing a Temporal Client

- **Specify attributes in `client.Options` to configure the Client**
  - **`HostPort`**: A colon-delimited string containing the hostname and port for the Frontend Service
    - Example: `fe.example.com:7233`
  - **`Namespace`**: A string specifying the namespace to use for requests sent by this Client
  - **`ConnectionOptions`**: A **`ConnectionOptions`** instance used to specify TLS parameters

Configuring Client for a Non-Local Cluster

- **This example specifies a namespace, but not parameters needed for TLS**

```
clientOptions := client.Options{
  HostPort:  "mycluster.example.com:7233",
  Namespace: "operations",
}
c, err := client.NewClient(clientOptions)
if err != nil {
  log.Fatalln("unable to create Temporal client", err)
}
defer c.Close()
```

- The options shown above are equivalent to those in the following `tctl` command

```
$ tctl --address mycluster.example.com:7233 --namespace operations workflow list
```

Configuring Client for a Secure Cluster

- **This example shows Client configuration for a secure non-local cluster**

```
ClientCertFile = "/home/myuser/tls/certificate.pem"
ClientCertPrivateKey = "/home/myuser/tls/private.key"

clientCert, err := tls.LoadX509KeyPair(ClientCertFile, ClientCertPrivateKey)
if err != nil {
  return nil, err
}

ServerName = "mycluster.example.com"
opts := client.Options{
  HostPort:  "mycluster.example.com:7233",
  Namespace: "operations",
  ConnectionOptions: client.ConnectionOptions{
    TLS: &tls.Config{
      Certificates: []tls.Certificate{clientCert},
    },
  },
}

return client.NewClient(opts)
```

Building a Temporal Application

- **Application deployment is usually preceded by a build process**
  - The tools used to do this vary by language, based on the SDK(s) used
  - Temporal does not require the use of any particular tools
  - You can use what is typical for the language or mandated by your organization
- **With the Go SDK, you can build the Worker to create an executable**
  - The result is what you would deploy and run in production
  - It must contain all dependencies required at runtime

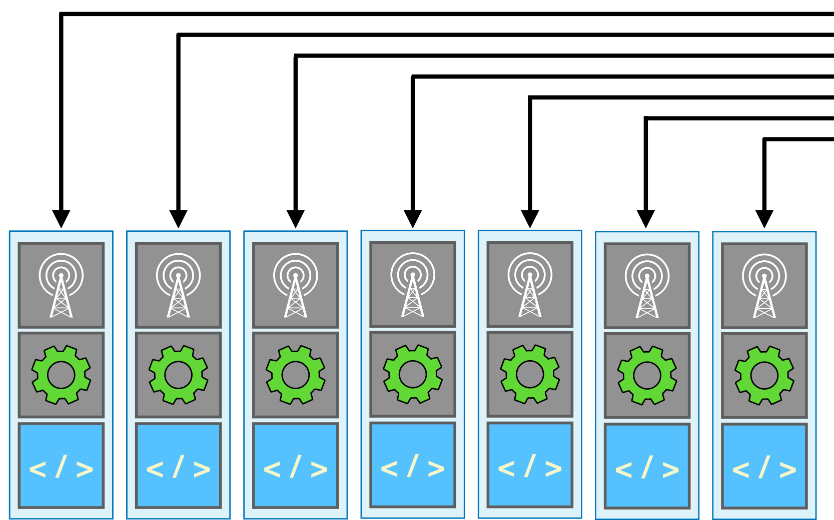
```
$ go build worker/main.go
```

Temporal Application Deployment

- **Once built, you'll deploy the application to production**
  - This will contain your compiled code, plus compile-time dependencies (e.g., Worker, Client, etc.)
  - Ensure any needed dependencies are available at runtime
    - For example, database drivers used by your application
    - For example, the Java runtime or Python interpreter for polyglot Temporal applications
- **Temporal is not opinionated about how or where you deploy the code**
  - Key point: Workers run externally to Temporal Cluster or Cloud
  - It's up to you how you run the Workers: bare metal, virtual machines, containers, etc.
  - Let's quickly look at two possible examples

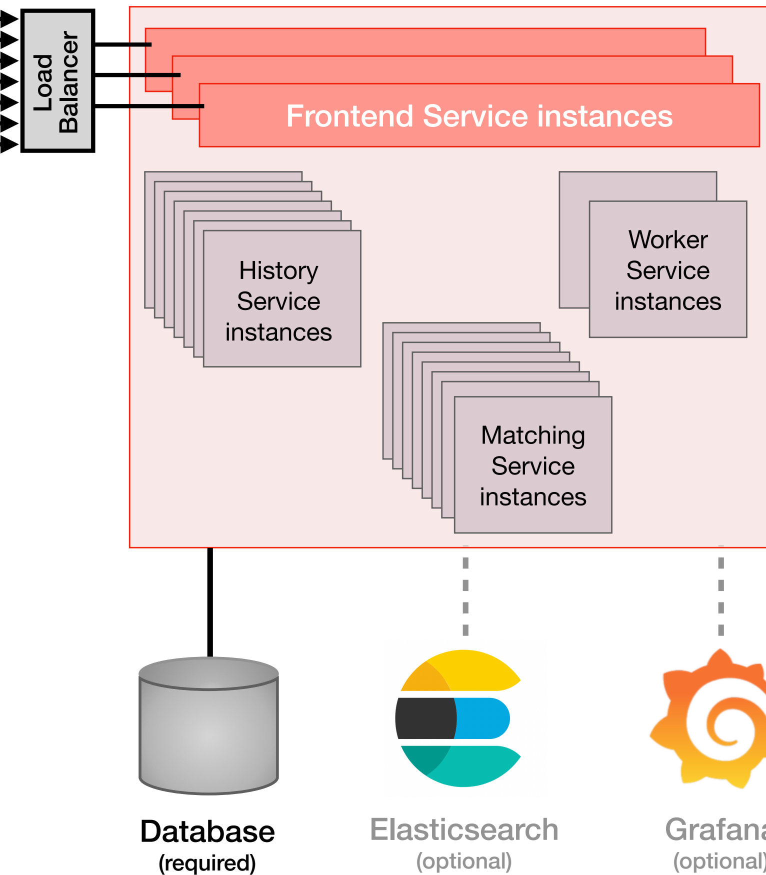
Deployment Scenario #1

Your Application

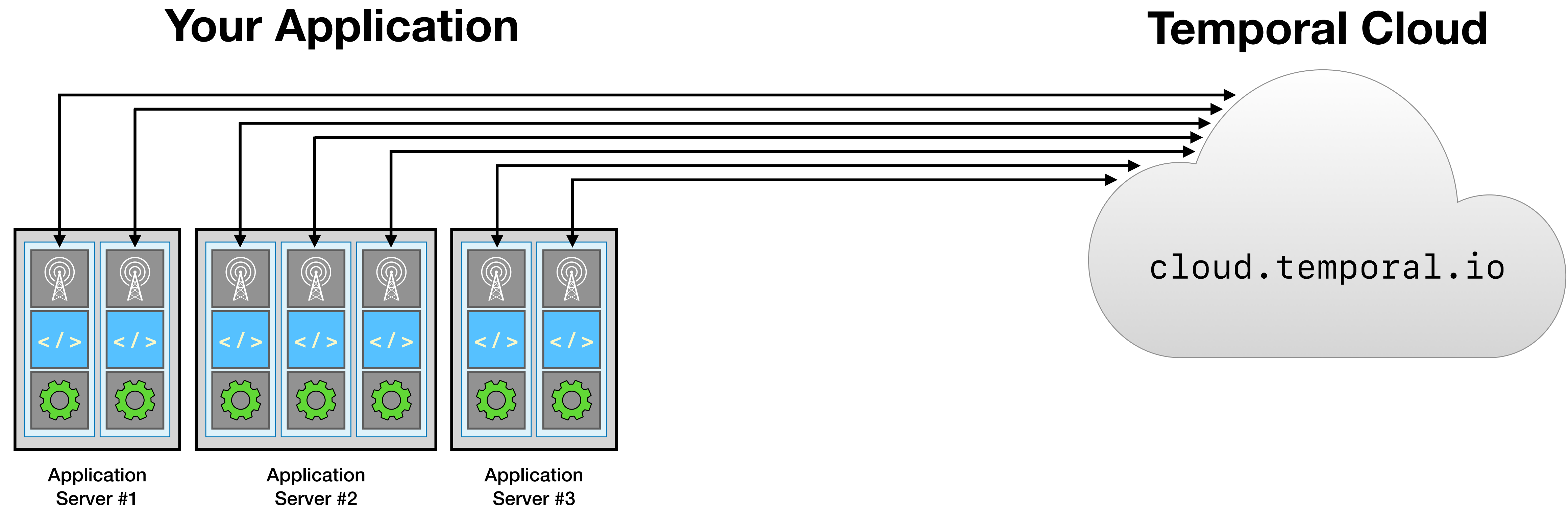


Example: Each Worker running in its own container

Temporal Cluster



Deployment Scenario #2



Temporal 102

00. About this Workshop
01. Understanding Key Concepts in Temporal
02. Improving Your Temporal Application Code
03. Using Timers in a Workflow Definition
04. Testing Your Temporal Application Code
05. Understanding Event History
06. Debugging Workflow History
07. Deploying Your Application to Production
08. Understanding Workflow Determinism
09. Conclusion

History Replay:

How Temporal Provides Durable Execution

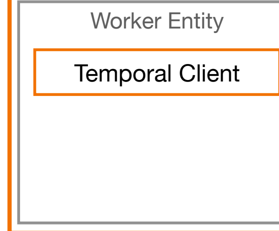
```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```

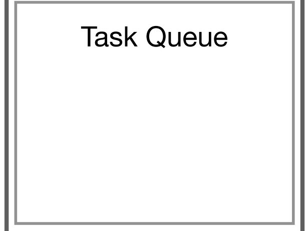
Start Workflow Execution

```
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"
      }
    }
]
```

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```

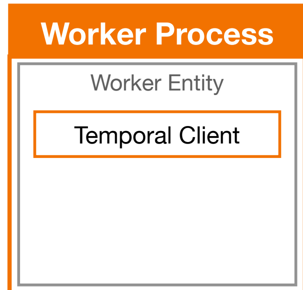


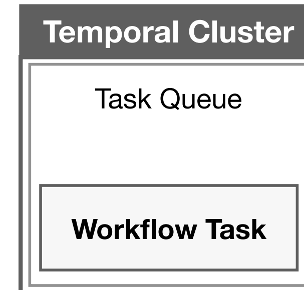


Commands

Events

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```

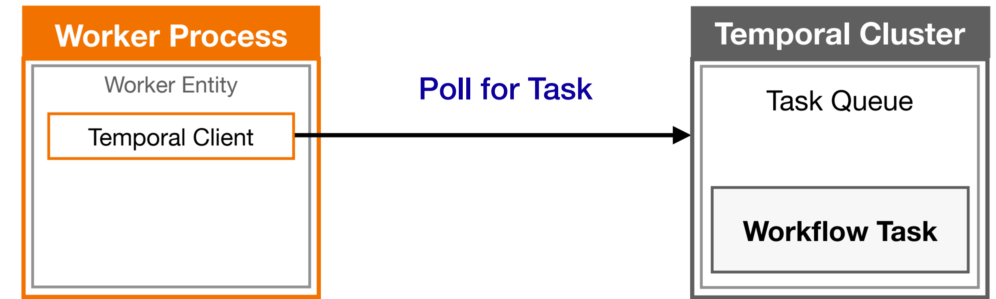




Commands

Events

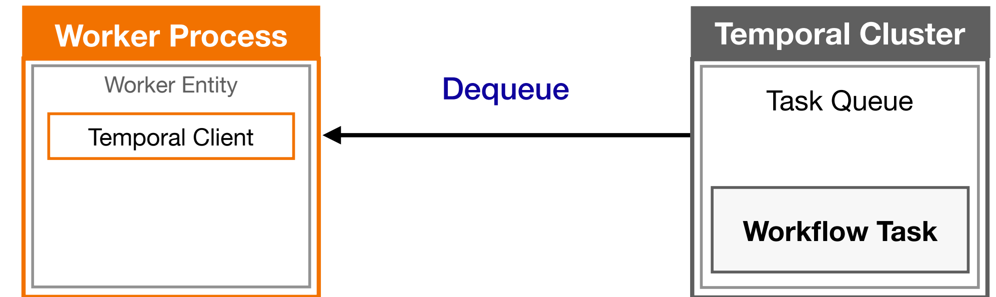
```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



Commands

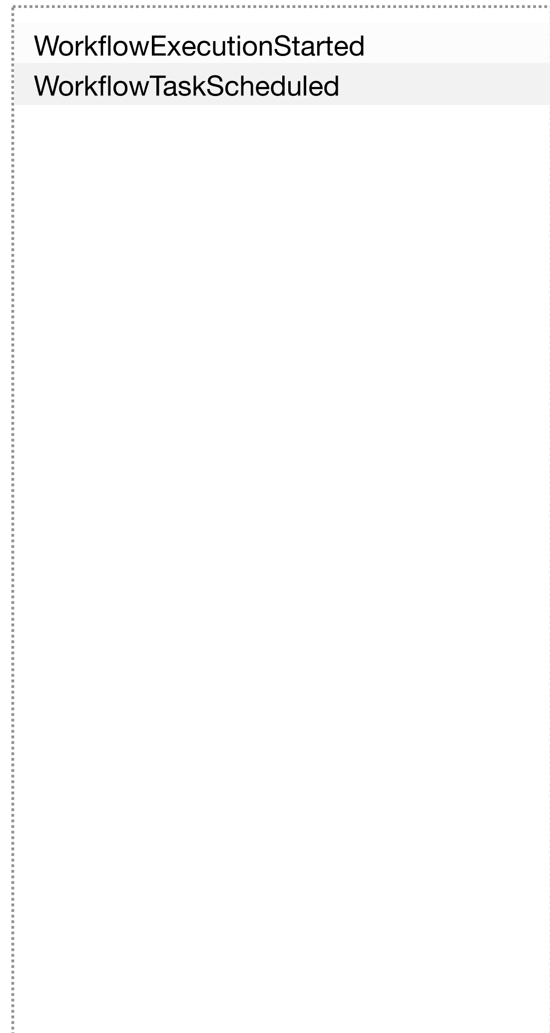
Events

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```

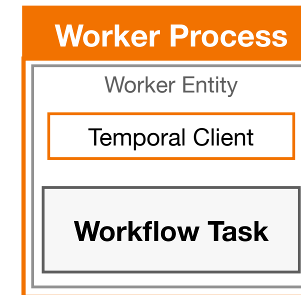


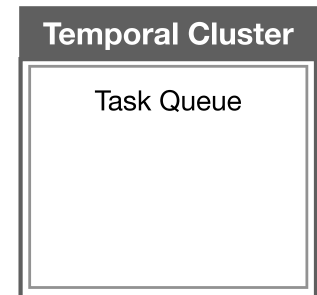
Commands

Events



```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```

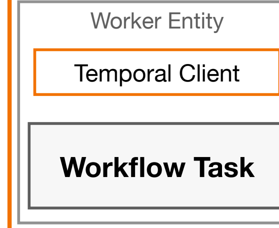


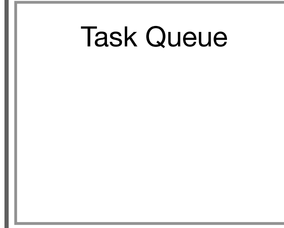


Commands

Events

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03 
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08 
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13 
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15 
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19 
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23 
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25 
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28 
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32 
33     return confirmation, nil
34 }
```



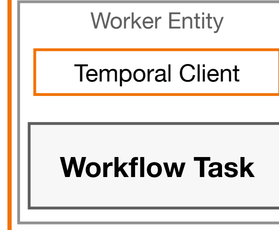


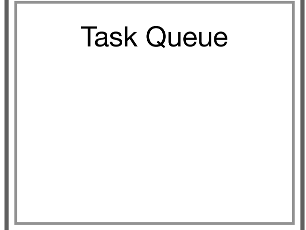
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03 
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08 
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13 
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15 
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19 
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23 
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25 
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28 
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32 
33     return confirmation, nil
34 }
```



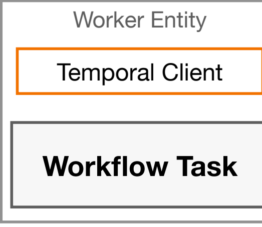


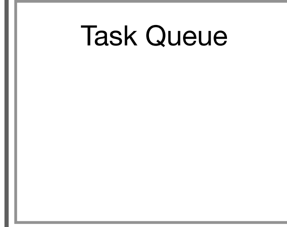
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



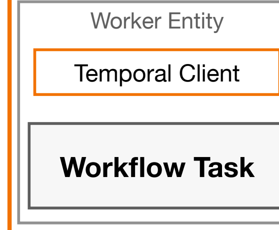


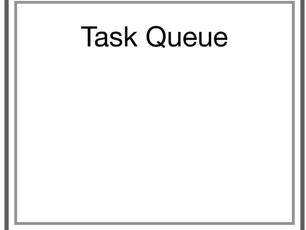
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03 
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08 
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13 
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15 
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19 
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23 
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25 
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28 
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32 
33     return confirmation, nil
34 }
```



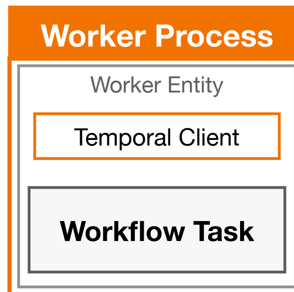


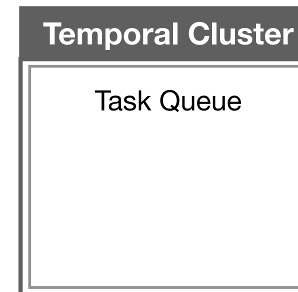
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



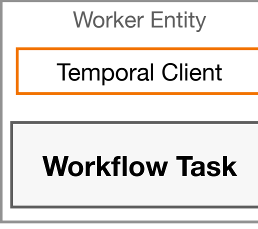


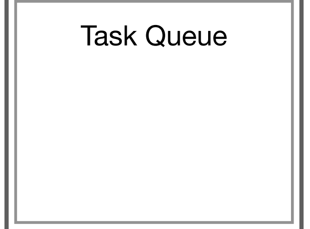
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```

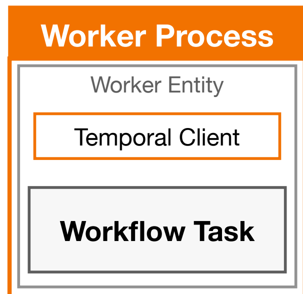


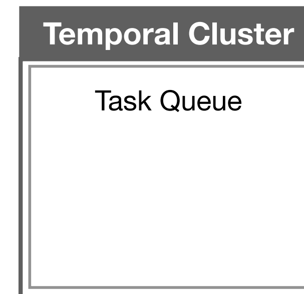


Commands

Events

```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
}
```



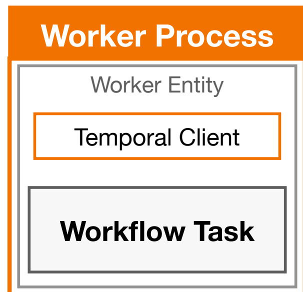


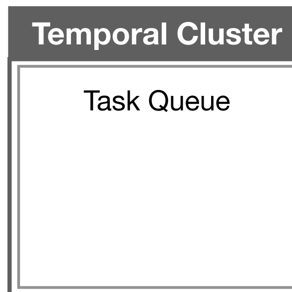
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```

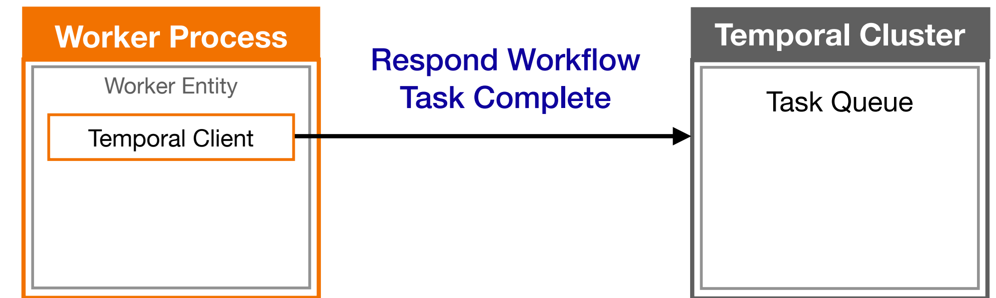




Commands

Events

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```

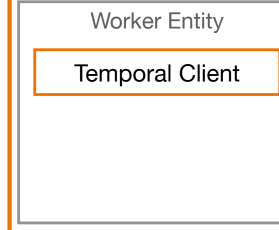


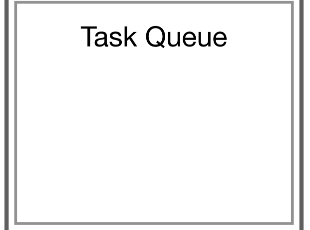
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



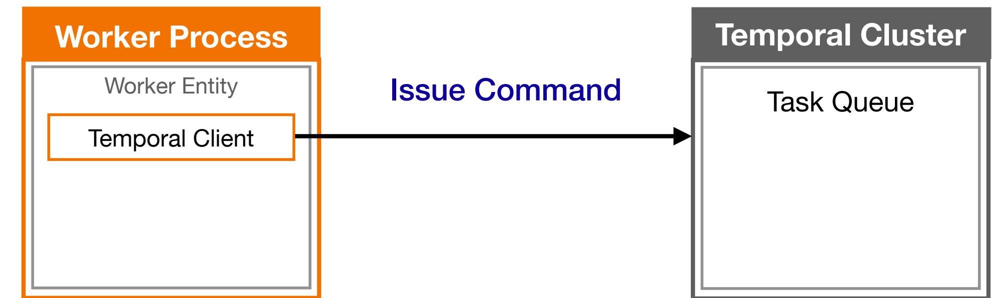


Commands

Events

- WorkflowExecutionStarted
- WorkflowTaskScheduled
- WorkflowTaskStarted
- **WorkflowTaskCompleted**

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



Commands

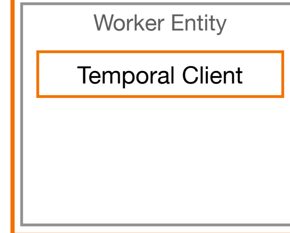
ScheduleActivityTask

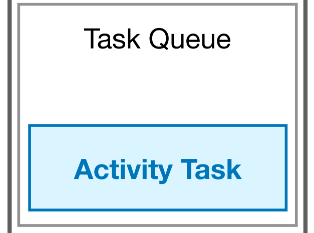
Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03 
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08 
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13 
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15 
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19 
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23 
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25 
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28 
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32 
33     return confirmation, nil
34 }
```





Commands

ScheduleActivityTask

Queue: pizza-tasks

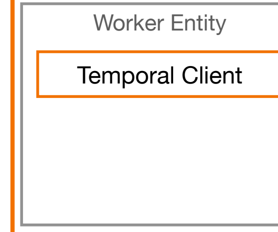
Type: GetDistance

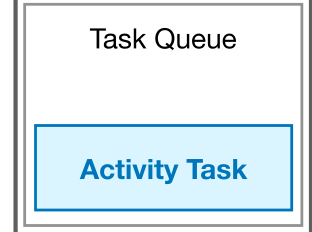
Input: "OrderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled (GetDistance)

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

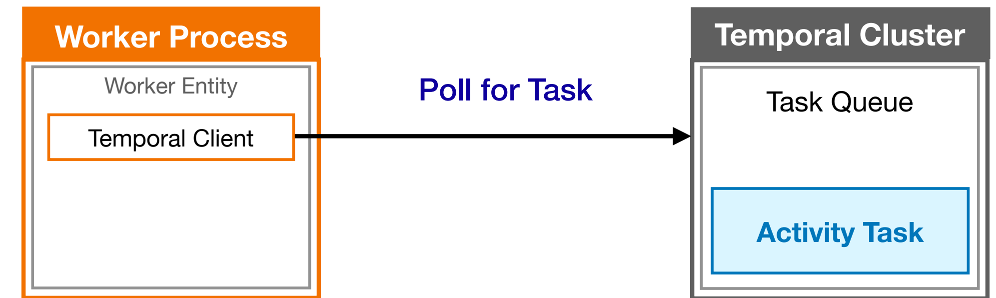
ScheduleActivityTask

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

Events

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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



Commands

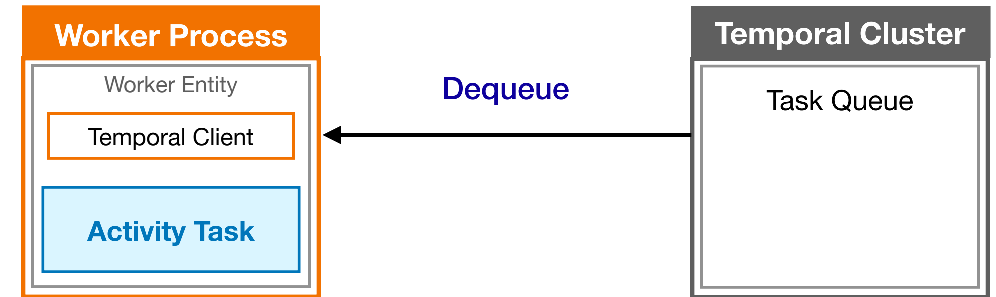
ScheduleActivityTask

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

Events

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

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



Commands

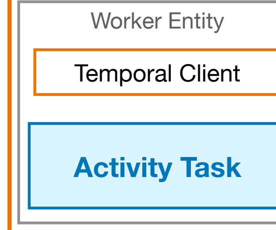
ScheduleActivityTask

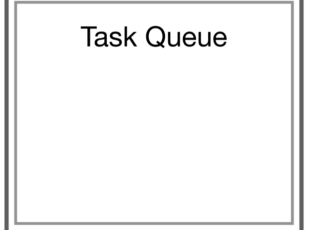
Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

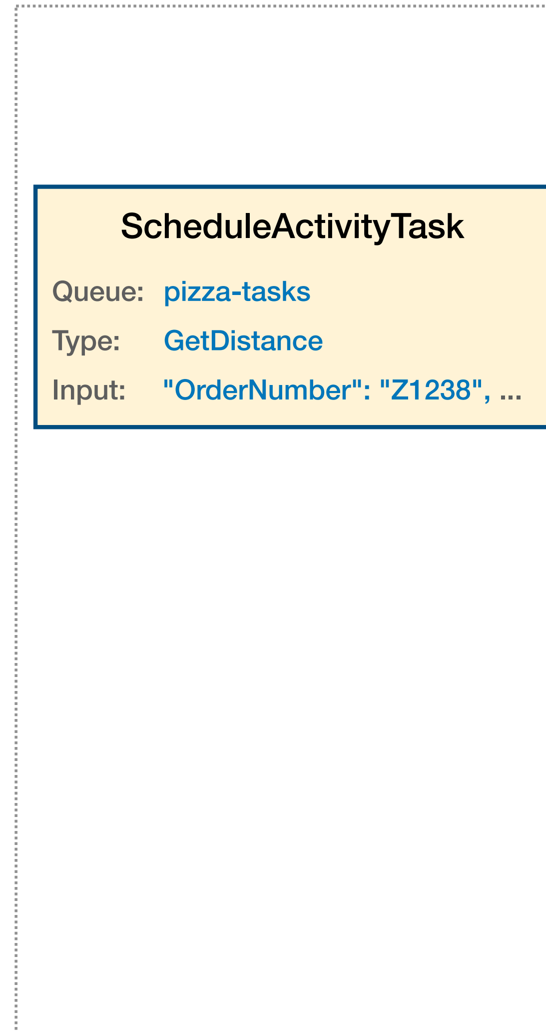
Events

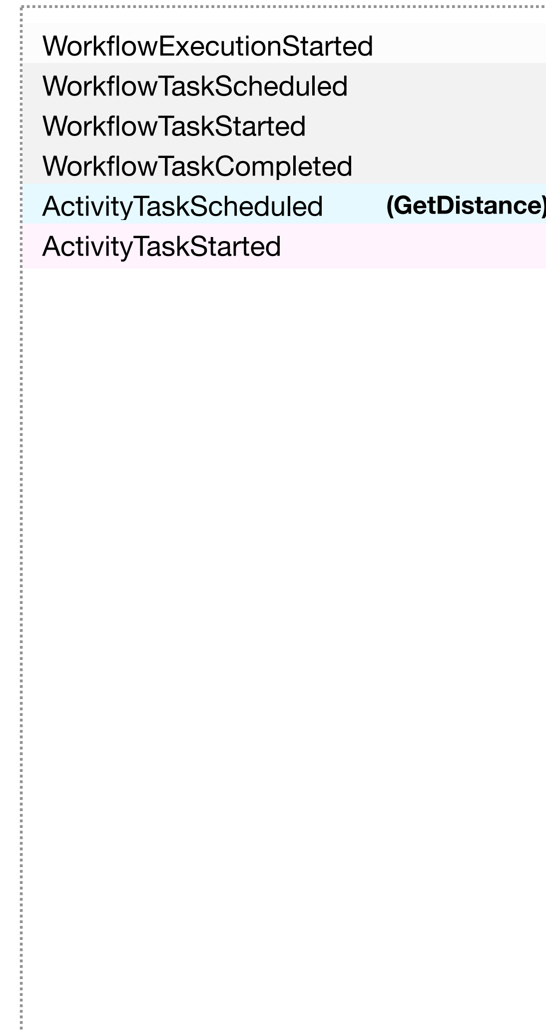
- WorkflowExecutionStarted
- WorkflowTaskScheduled
- WorkflowTaskStarted
- WorkflowTaskCompleted
- ActivityTaskScheduled **(GetDistance)**
- **ActivityTaskStarted**

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```

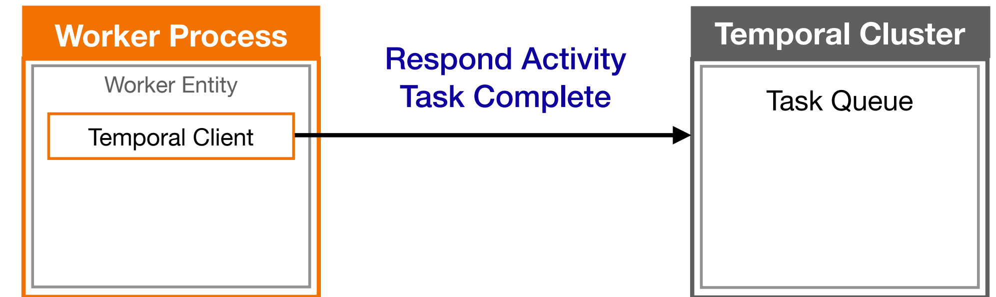








```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



Commands

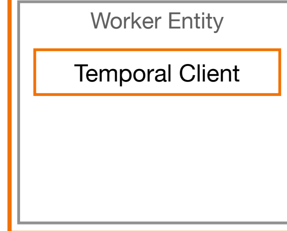
ScheduleActivityTask

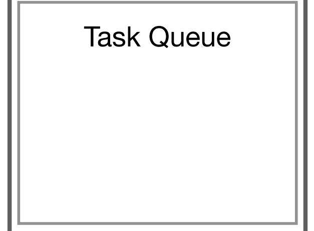
Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

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

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





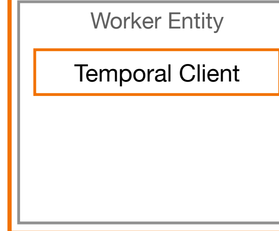
Commands

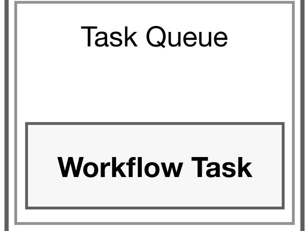
ScheduleActivityTask	
Queue:	pizza-tasks
Type:	GetDistance
Input:	"OrderNumber": "Z1238", ...

Events

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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

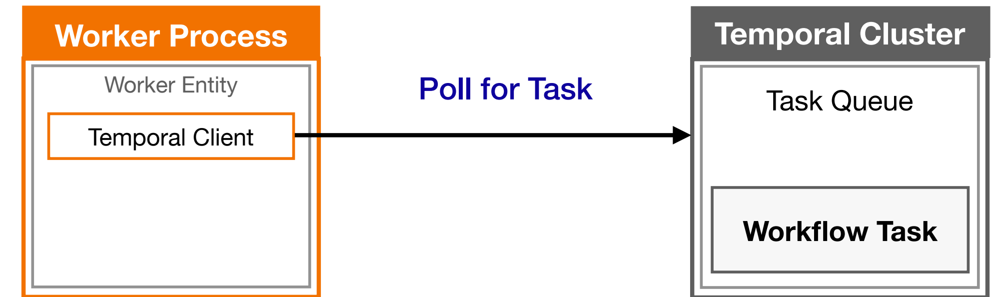
ScheduleActivityTask

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

Events

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

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



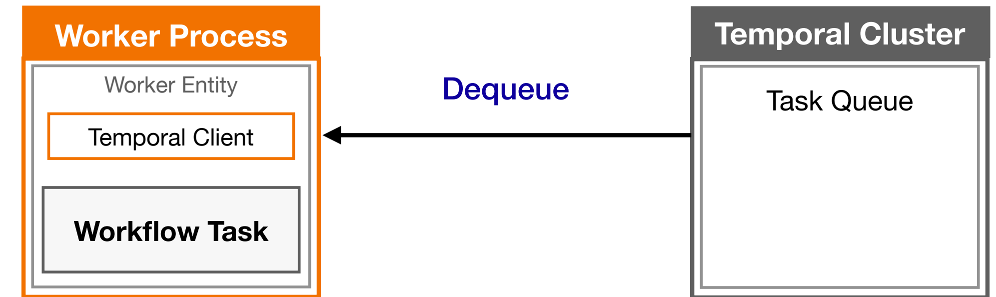
Commands

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

Events

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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



Commands

ScheduleActivityTask

Queue: pizza-tasks

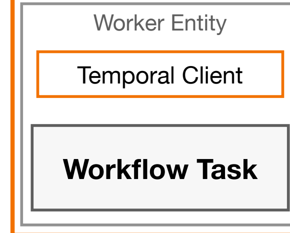
Type: GetDistance

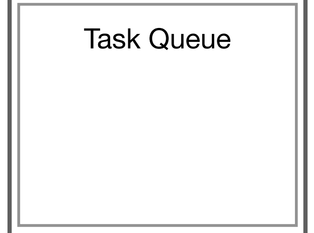
Input: "OrderNumber": "Z1238", ...

Events

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

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03 
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08 
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13 
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15 
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19 
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23 
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25 
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28 
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32 
33     return confirmation, nil
34 }
```





Commands

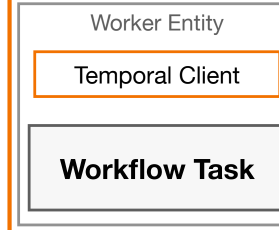
ScheduleActivityTask

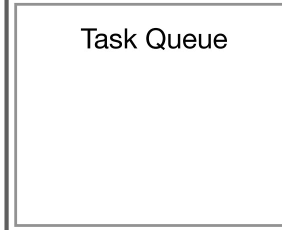
Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

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

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

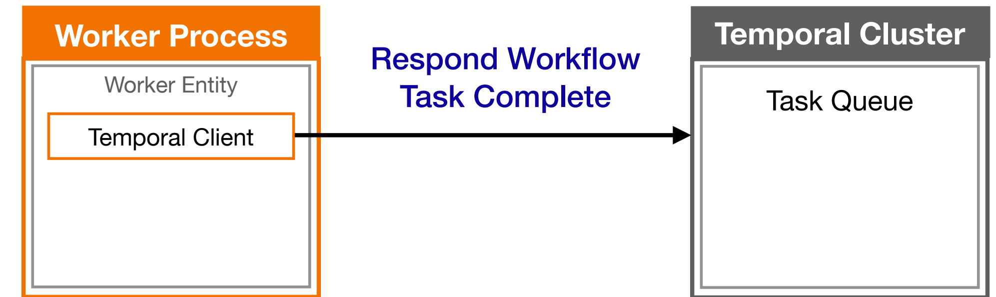
ScheduleActivityTask

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

Events

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

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03 
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08 
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13 
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15 
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19 
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23 
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25 
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28 
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32 
33     return confirmation, nil
34 }
```



Commands

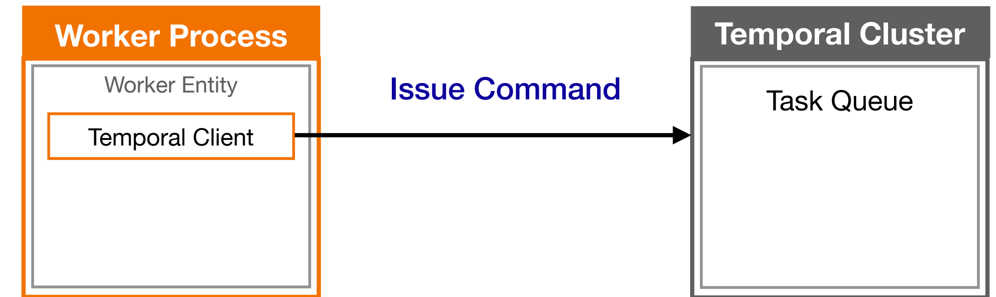
ScheduleActivityTask

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

Events

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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03 
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08 
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13 
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15 
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19 
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23 
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25 
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28 
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32 
33     return confirmation, nil
34 }
```



Commands

ScheduleActivityTask

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

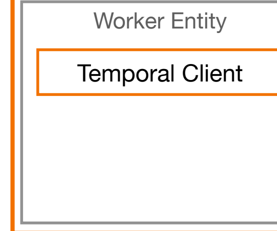
StartTimer

Duration: 30 minutes

Events

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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



Temporal Cluster

Task Queue

Commands

ScheduleActivityTask

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

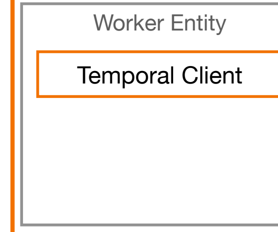
StartTimer

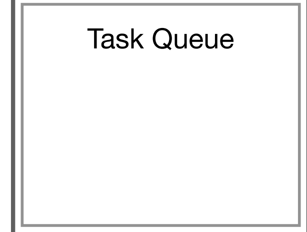
Duration: 30 minutes

Events

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

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

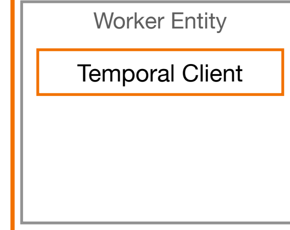
ScheduleActivityTask
Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

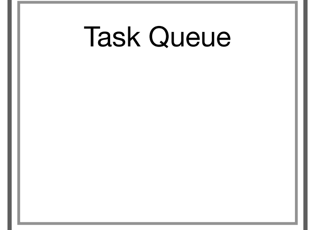
StartTimer
Duration: 30 minutes

Events

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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

ScheduleActivityTask

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

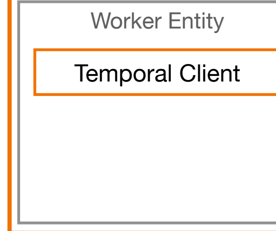
StartTimer

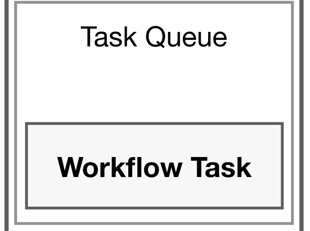
Duration: 30 minutes

Events

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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

ScheduleActivityTask

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

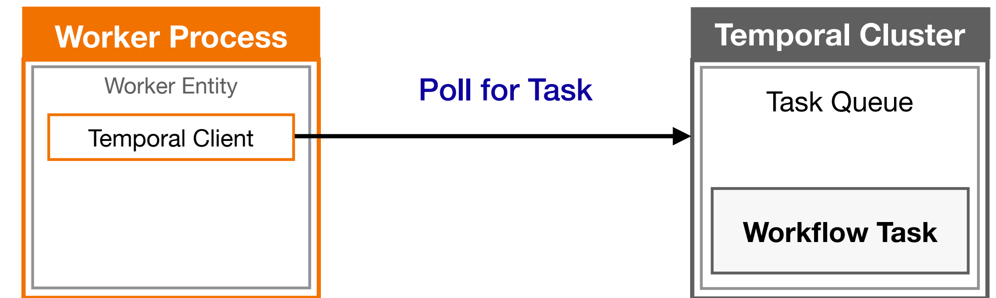
StartTimer

Duration: 30 minutes

Events

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

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



Commands

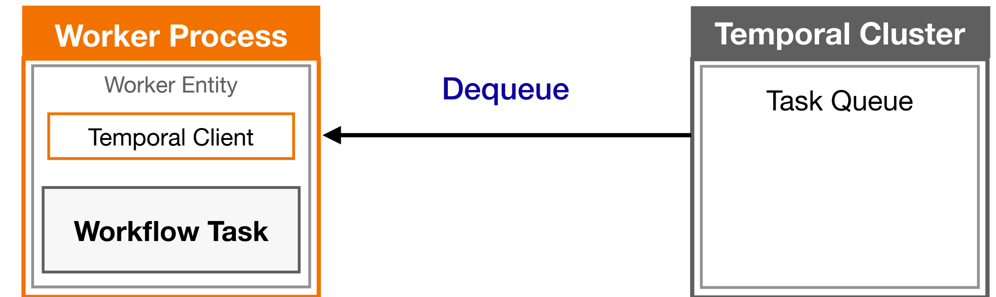
ScheduleActivityTask
Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

StartTimer
Duration: 30 minutes

Events

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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



Commands

ScheduleActivityTask

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

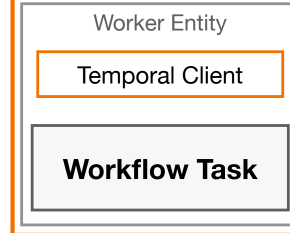
StartTimer

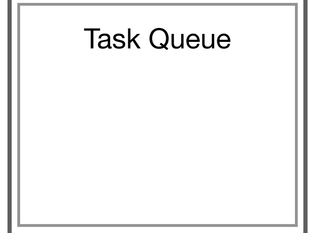
Duration: 30 minutes

Events

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

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

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

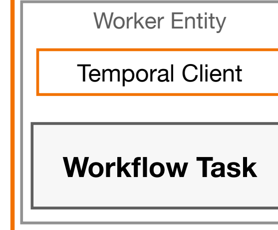
StartTimer	
Duration:	30 minutes

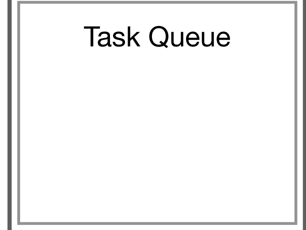
Events

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

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```

Worker crashes here





Commands

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

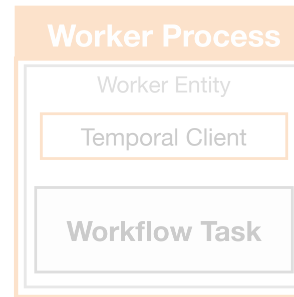
StartTimer
Duration: 30 minutes

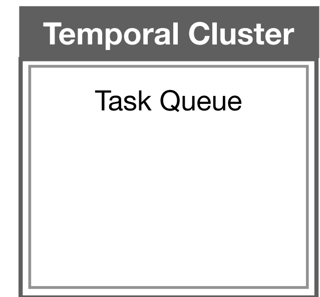
Events

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

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```

Worker crashes here



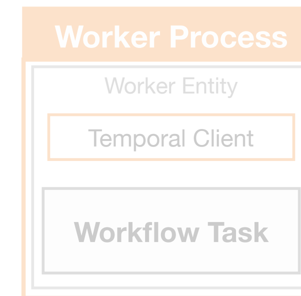


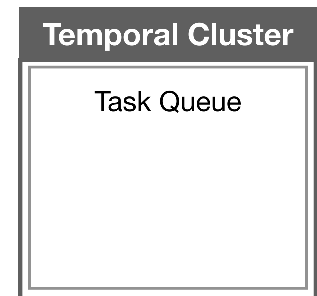
Commands

Events

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

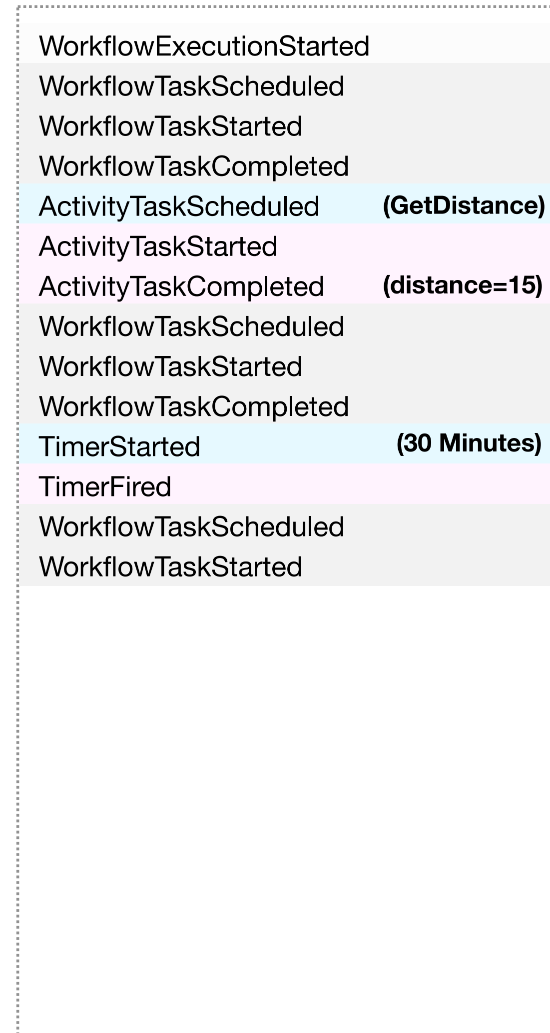
```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

Events



```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```

Worker Process

Workflow Task Timeout Exceeded

Temporal Cluster

Task Queue

Commands

Events

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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03 
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08 
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13 
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15 
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19 
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23 
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25 
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28 
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32 
33     return confirmation, nil
34 }
```

Temporal Cluster

Task Queue

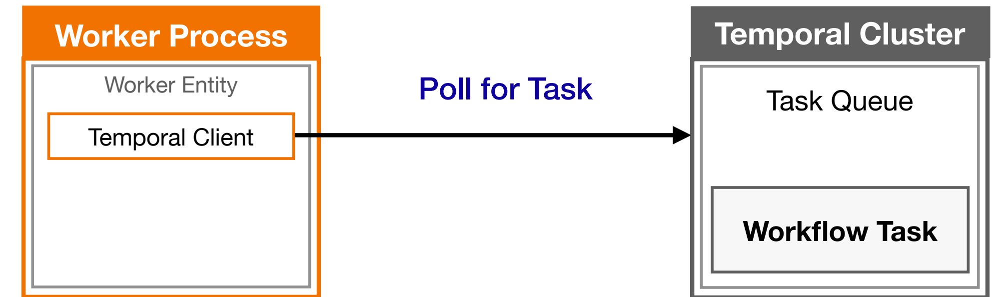
Workflow Task

Commands

Events

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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```

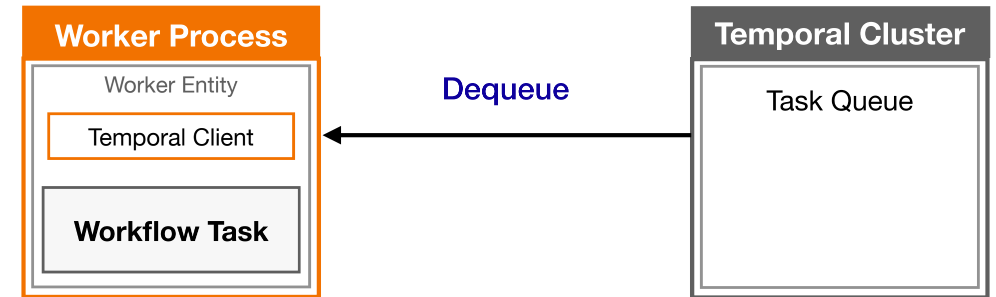


Commands

Events

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

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```

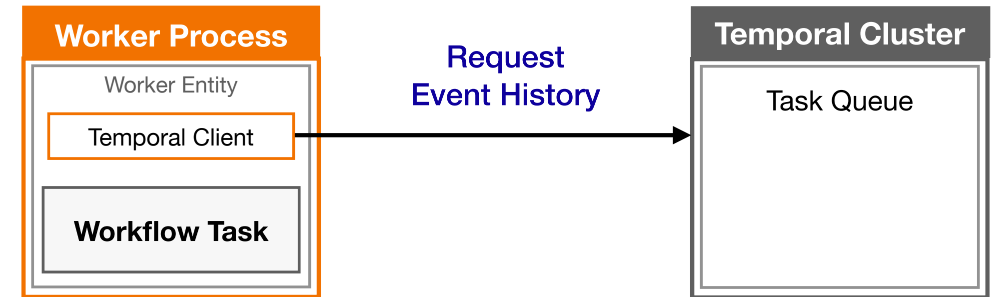


Commands

Events

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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```

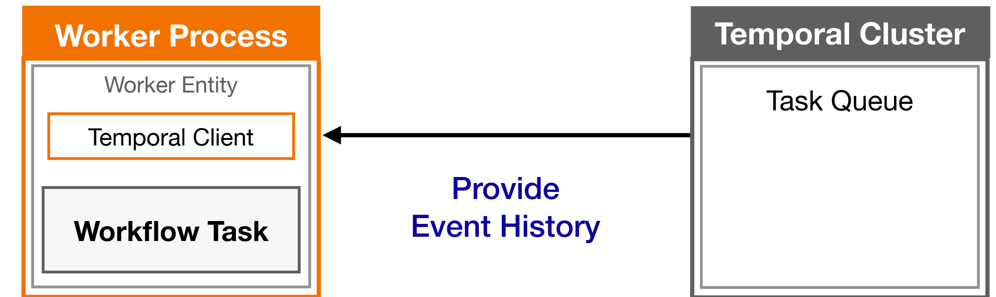


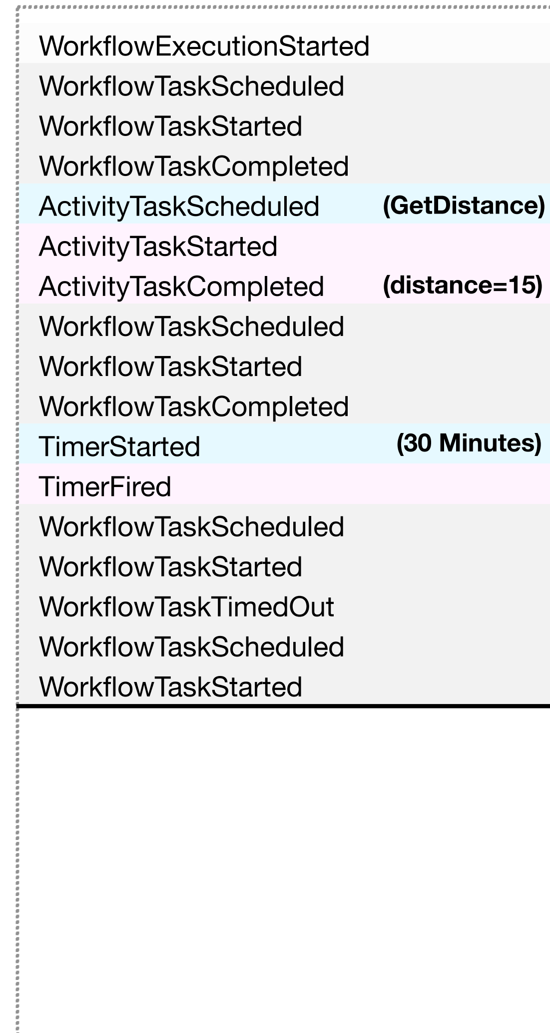
Commands

Events

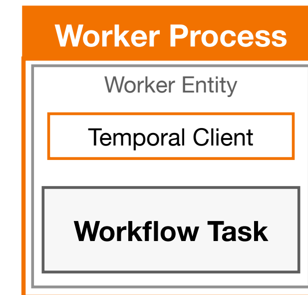
- WorkflowExecutionStarted
- WorkflowTaskScheduled
- WorkflowTaskStarted
- WorkflowTaskCompleted
- ActivityTaskScheduled **(GetDistance)**
- ActivityTaskStarted
- ActivityTaskCompleted **(distance=15)**
- WorkflowTaskScheduled
- WorkflowTaskStarted
- WorkflowTaskCompleted
- TimerStarted **(30 Minutes)**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

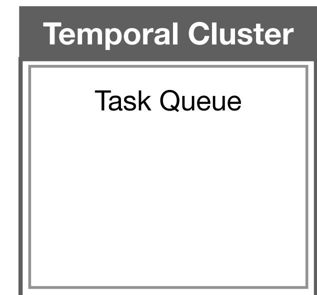
```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





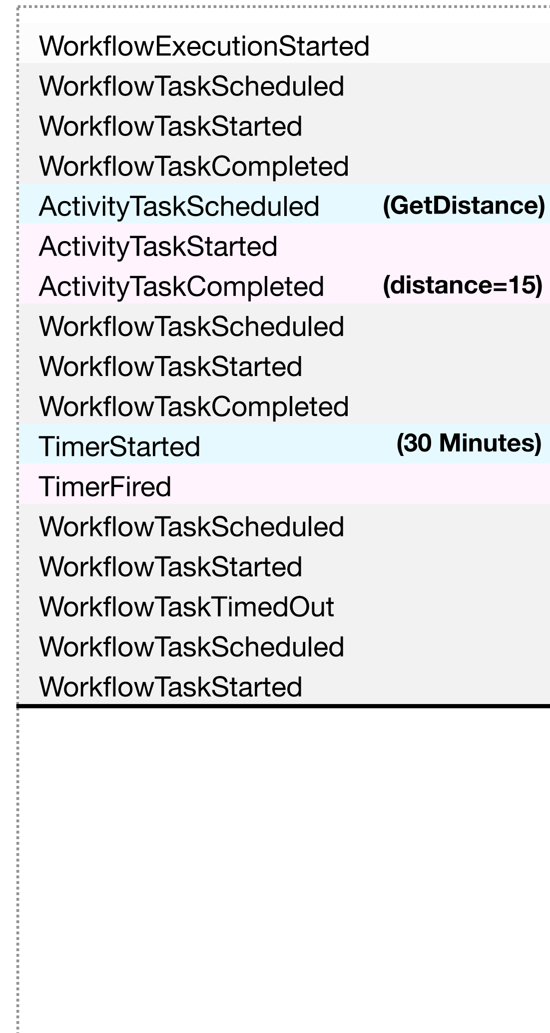
```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



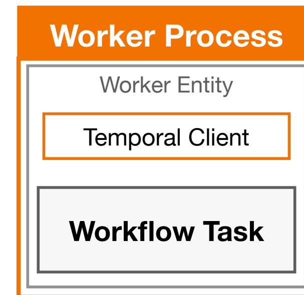


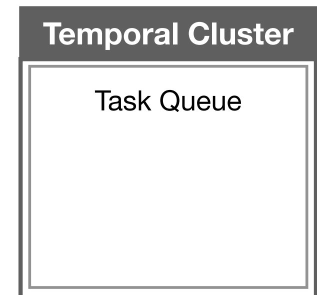
Commands

Events



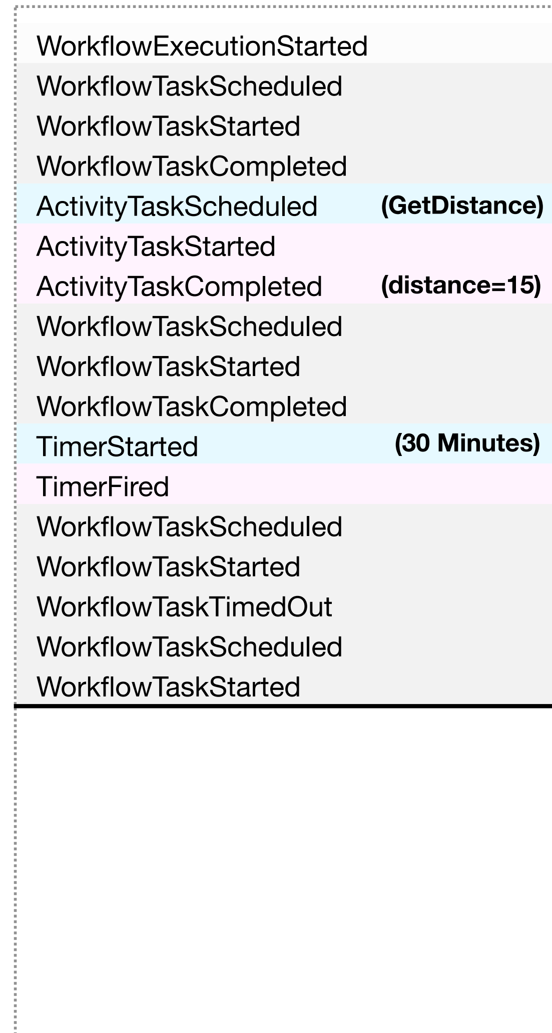
```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



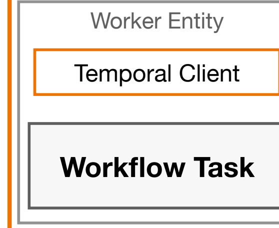


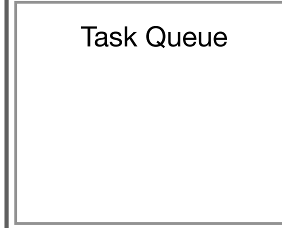
Commands

Events



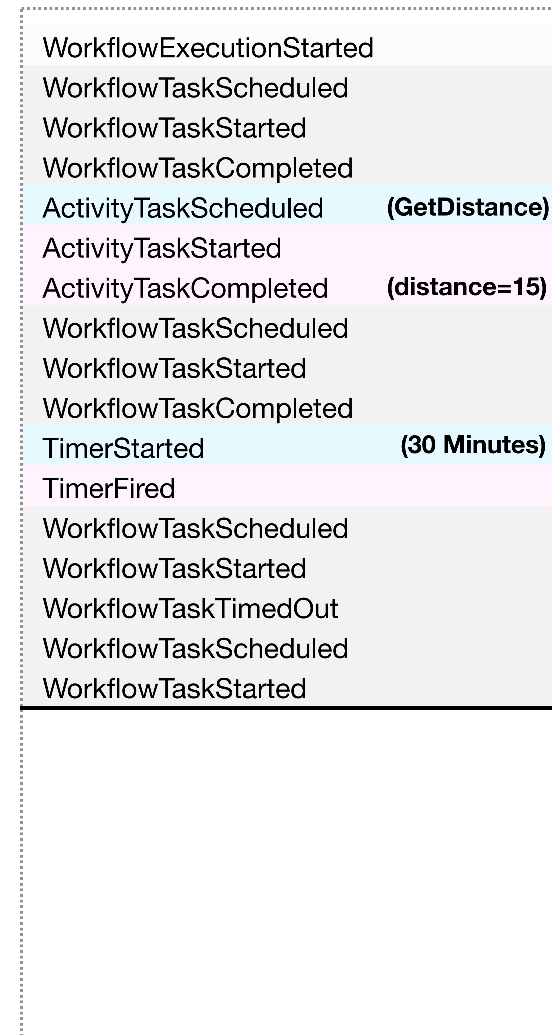
```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



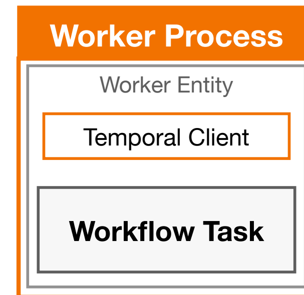


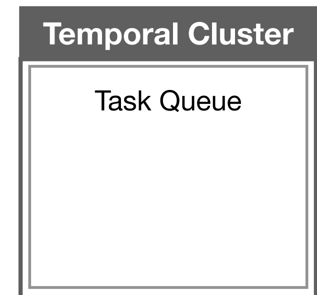
Commands

Events



```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



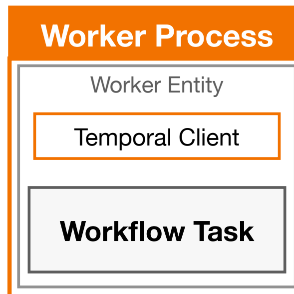


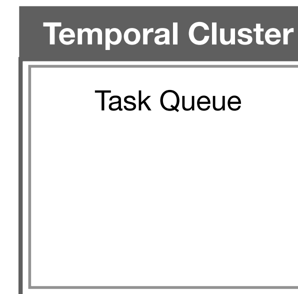
Commands

Events

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

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



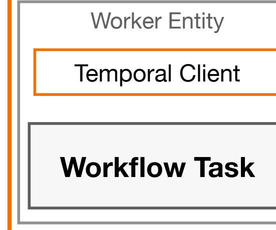


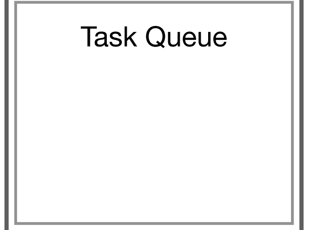
Commands

Events

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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



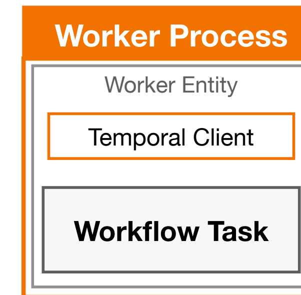


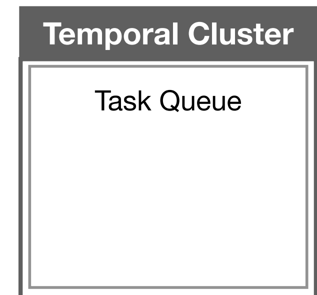
Commands

Events

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

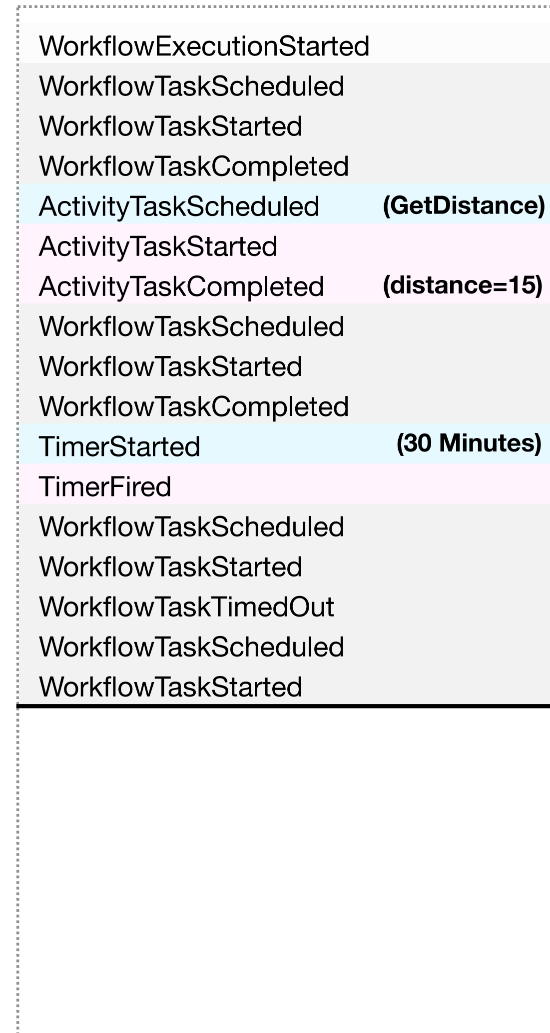
```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



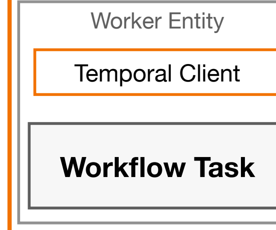


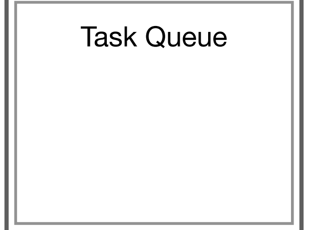
Commands

Events



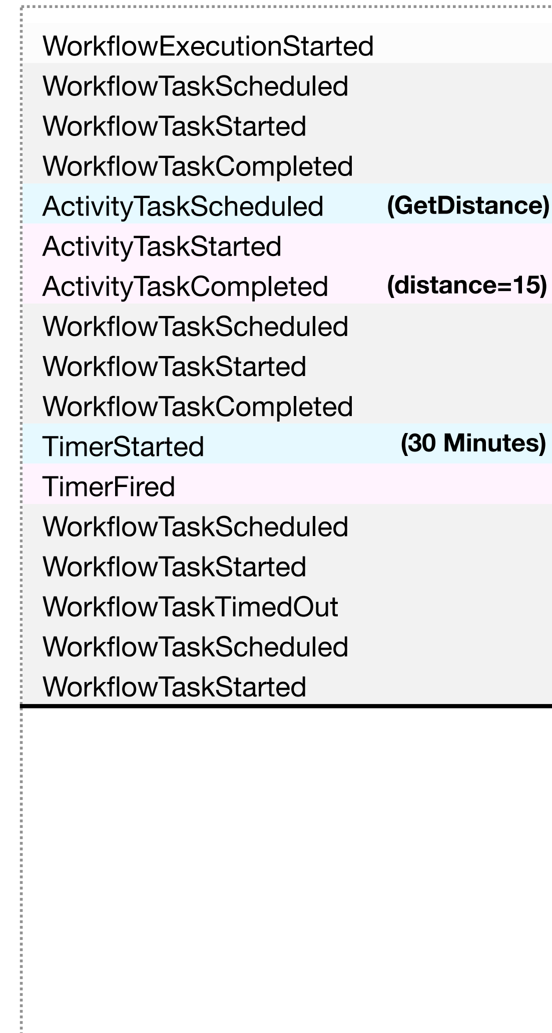
```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03 
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08 
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13 
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15 
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19 
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23 
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25 
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28 
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32 
33     return confirmation, nil
34 }
```



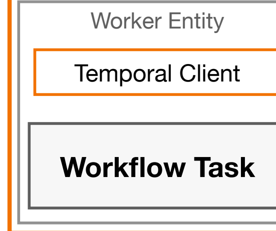


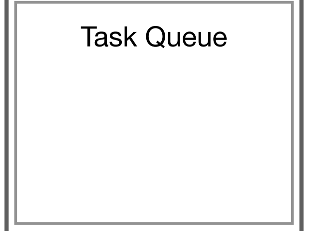
Commands

Events



```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

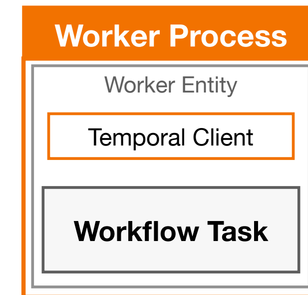
ScheduleActivityTask

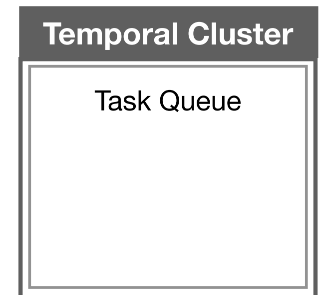
Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

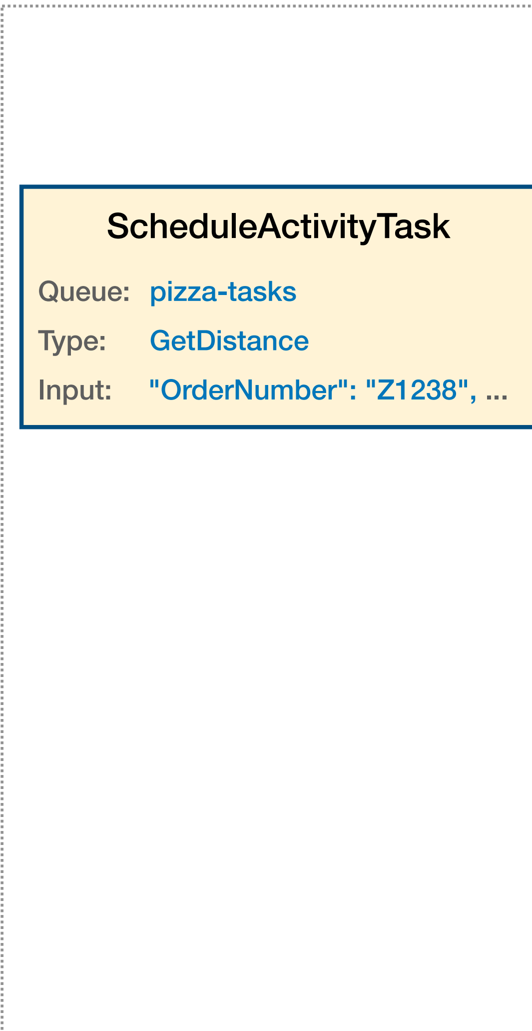
Events

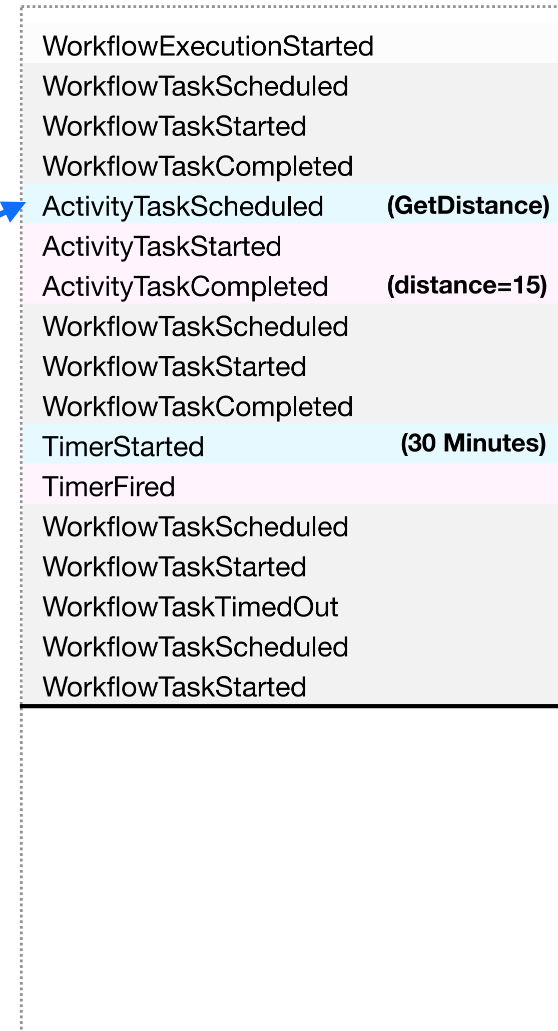
WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(GetDistance)**
ActivityTaskStarted
ActivityTaskCompleted **(distance=15)**
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
TimerStarted **(30 Minutes)**
TimerFired
WorkflowTaskScheduled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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```

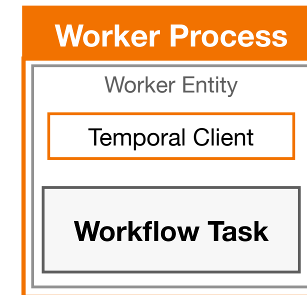


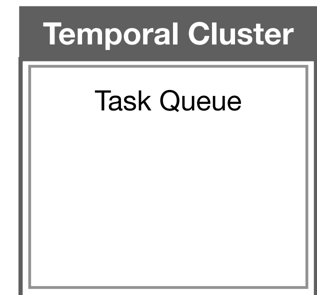






```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03 
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08 
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13 
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15 
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19 
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23 
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25 
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28 
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32 
33     return confirmation, nil
34 }
```





Commands

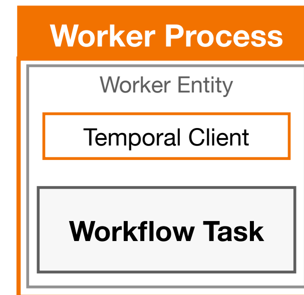
ScheduleActivityTask

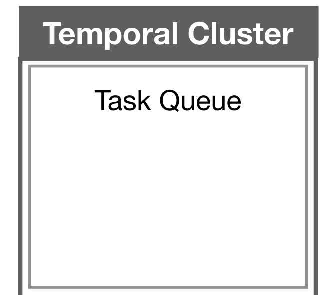
Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

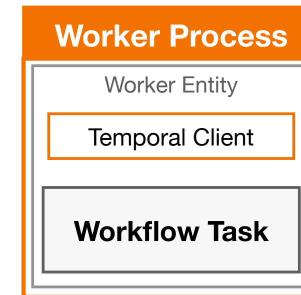
WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(GetDistance)**
ActivityTaskStarted
ActivityTaskCompleted **(distance=15)**
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
TimerStarted **(30 Minutes)**
TimerFired
WorkflowTaskScheduled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

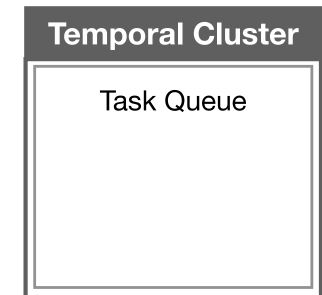
```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)  ← Worker assigns 15 to this variable
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

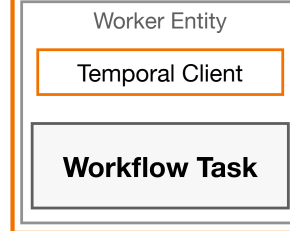
ScheduleActivityTask

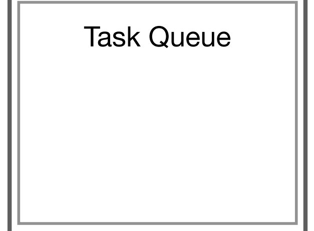
Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

Events

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

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





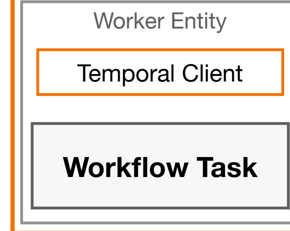
Commands

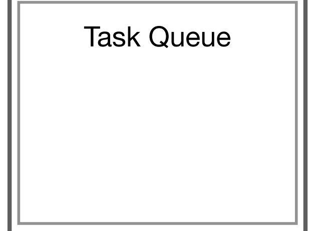
ScheduleActivityTask	
Queue:	pizza-tasks
Type:	GetDistance
Input:	"OrderNumber": "Z1238", ...

Events

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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

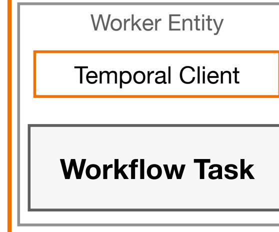
ScheduleActivityTask

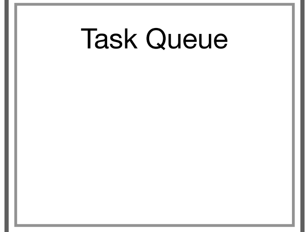
Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

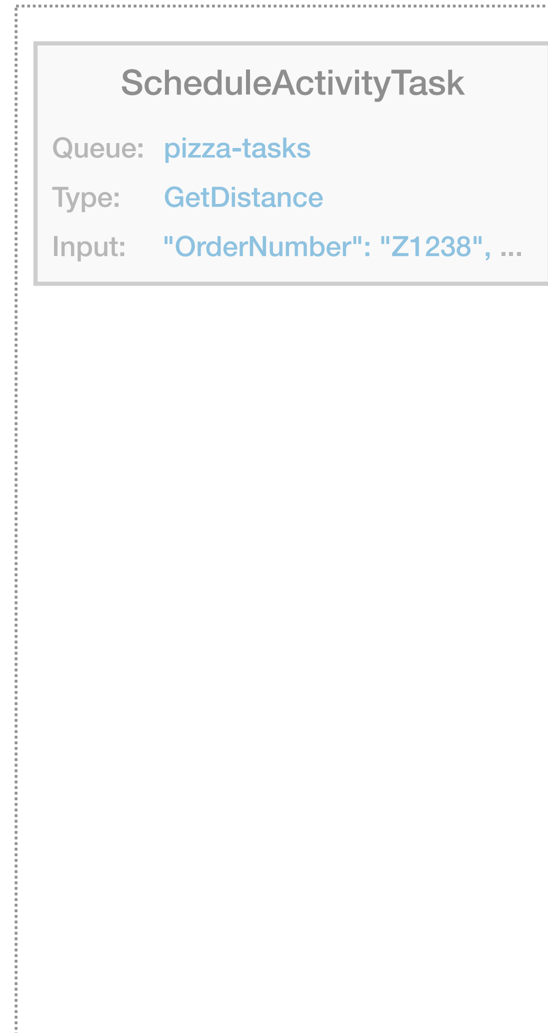
Events

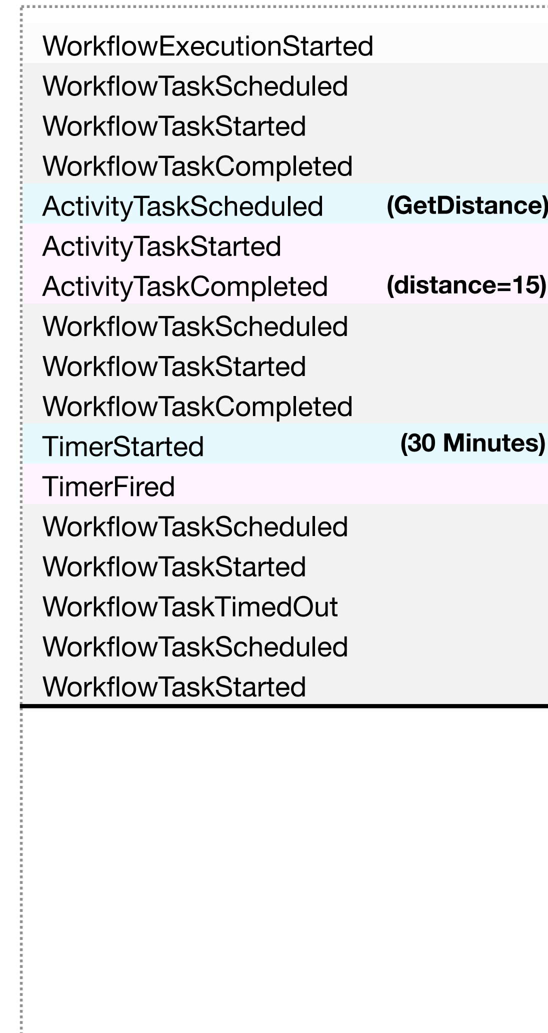
Event	
WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(GetDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```

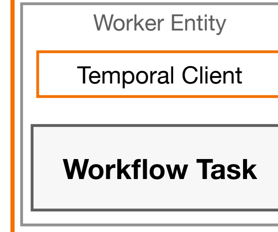


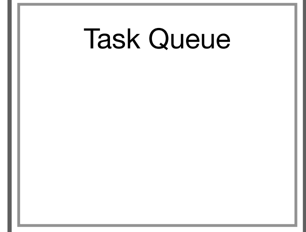






```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

ScheduleActivityTask

Queue: pizza-tasks

Type: GetDistance

Input: "OrderNumber": "Z1238", ...

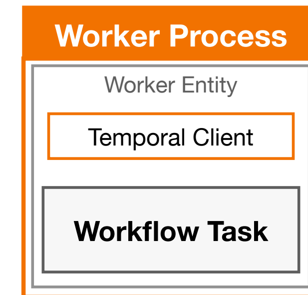
StartTimer

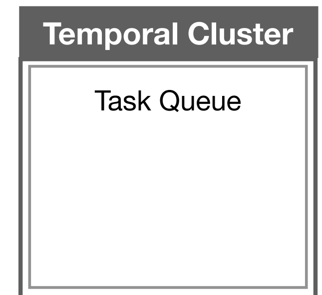
Duration: 30 minutes

Events

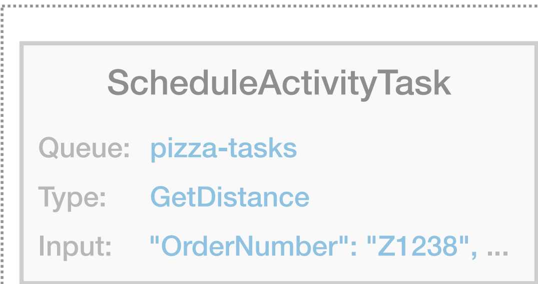
WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(GetDistance)**
ActivityTaskStarted
ActivityTaskCompleted **(distance=15)**
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
TimerStarted **(30 Minutes)**
TimerFired
WorkflowTaskScheduled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

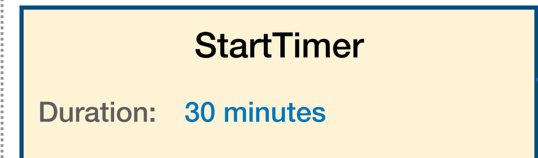
```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```

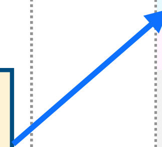




Commands



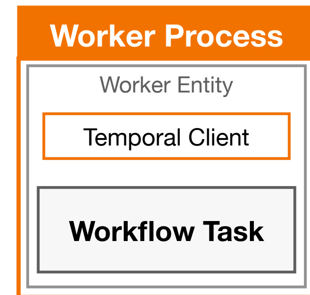


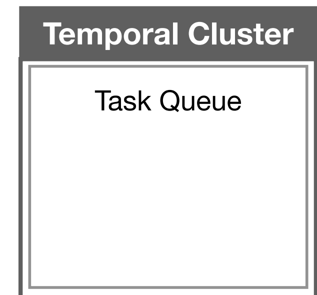


Events

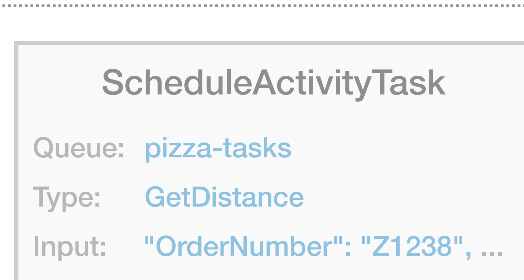
WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(GetDistance)**
ActivityTaskStarted
ActivityTaskCompleted **(distance=15)**
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
TimerStarted **(30 Minutes)**
TimerFired
WorkflowTaskScheduled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

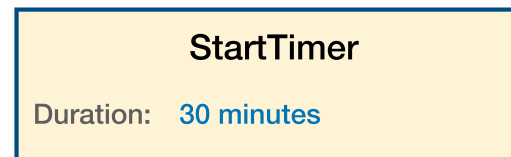
```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



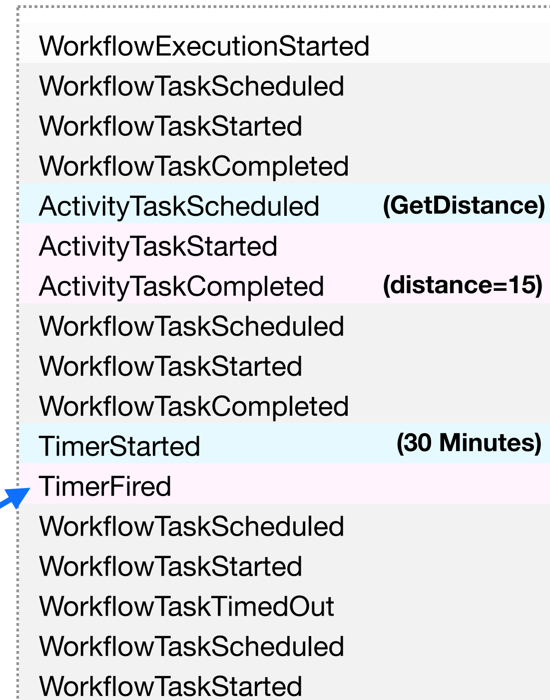


Commands

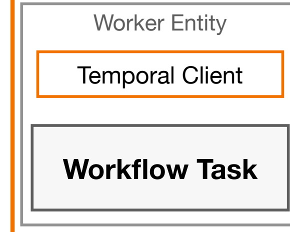


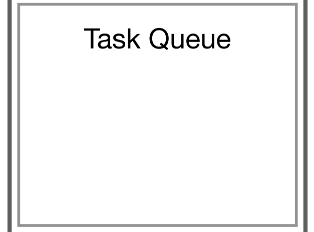


Events



```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03 
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08 
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13 
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15 
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19 
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23 
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25 
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28 
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32 
33     return confirmation, nil
34 }
```





Commands

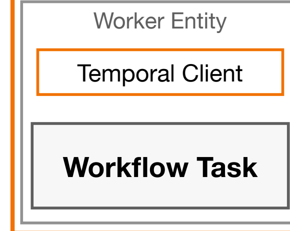
ScheduleActivityTask
Queue: pizza-tasks
Type: GetDistance
Input: "OrderNumber": "Z1238", ...

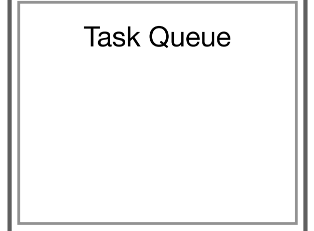
StartTimer
30 minutes

Events

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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

ScheduleActivityTask

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

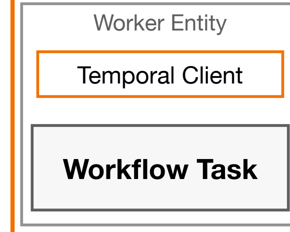
StartTimer

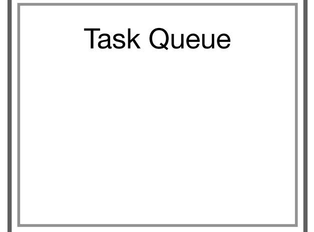
30 minutes

Events

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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

ScheduleActivityTask

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

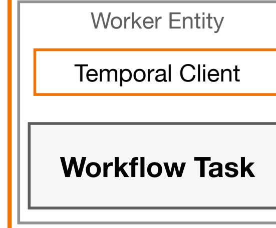
StartTimer

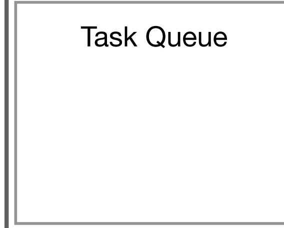
30 minutes

Events

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

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

ScheduleActivityTask

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

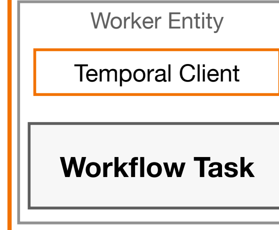
StartTimer

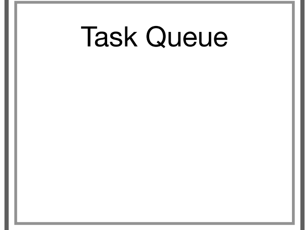
30 minutes

Events

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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

ScheduleActivityTask

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

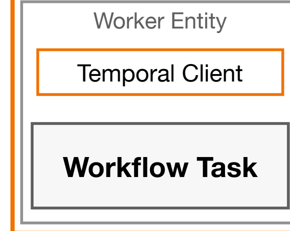
StartTimer

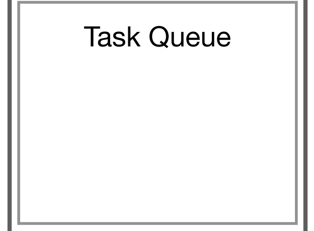
30 minutes

Events

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

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03 
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08 
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13 
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15 
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19 
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23 
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25 
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28 
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32 
33     return confirmation, nil
34 }
```





Commands

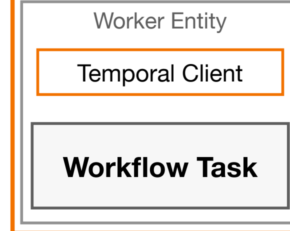
ScheduleActivityTask	
Queue:	pizza-tasks
Type:	GetDistance
Input:	"OrderNumber": "Z1238", ...

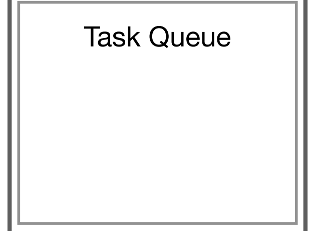
StartTimer
30 minutes

Events

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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

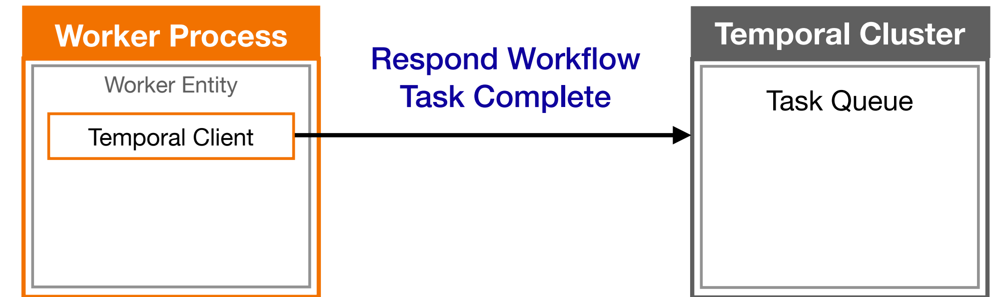
ScheduleActivityTask	
Queue:	pizza-tasks
Type:	GetDistance
Input:	"OrderNumber": "Z1238", ...

StartTimer
30 minutes

Events

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

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



Commands

ScheduleActivityTask

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

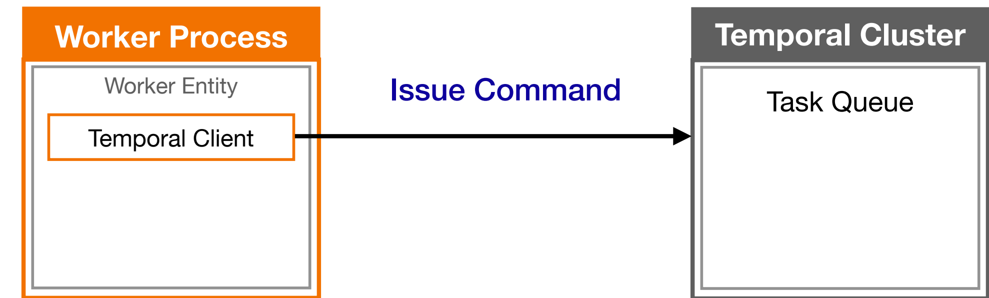
StartTimer

30 minutes

Events

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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



Commands

ScheduleActivityTask

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

StartTimer

30 minutes

ScheduleActivityTask

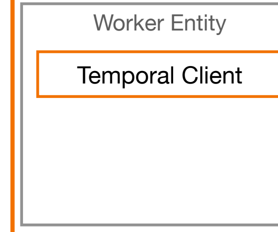
Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

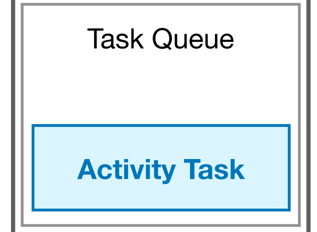
Events

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

WorkflowTaskCompleted

```go
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

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

StartTimer
30 minutes

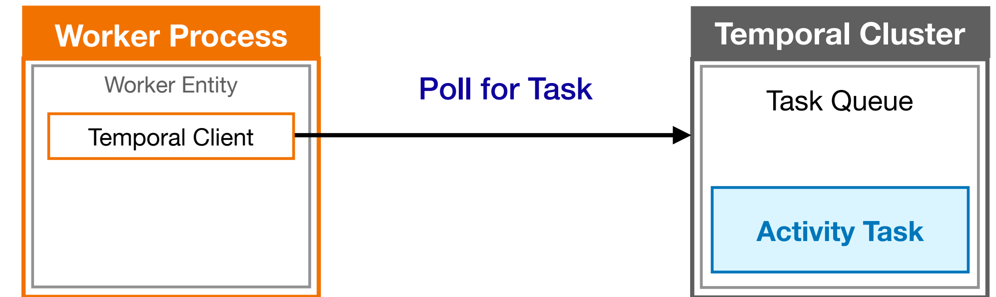
ScheduleActivityTask
Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

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)**

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



Commands

ScheduleActivityTask

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

StartTimer

30 minutes

ScheduleActivityTask

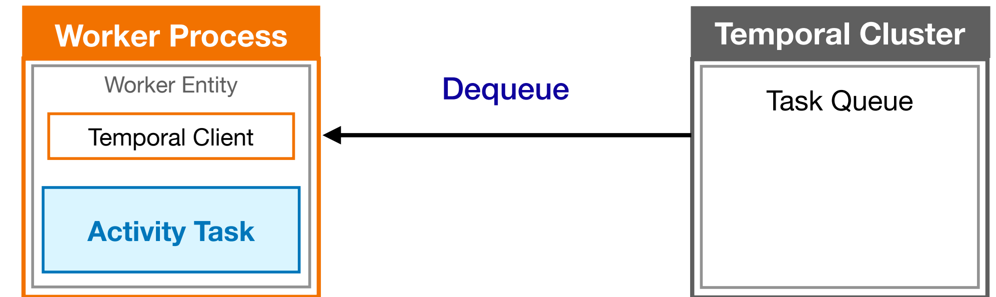
Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

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
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



Commands

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

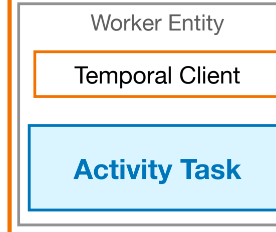
StartTimer
30 minutes

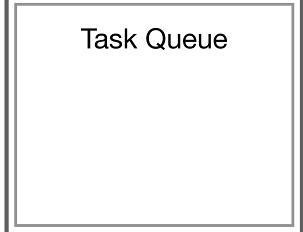
ScheduleActivityTask
Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

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

StartTimer
30 minutes

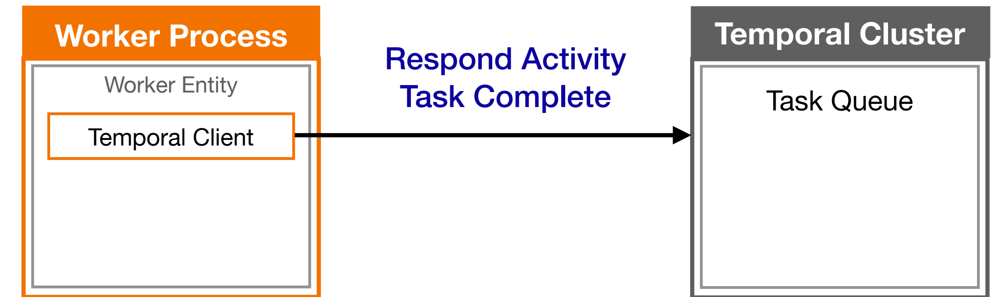
ScheduleActivityTask
Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

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
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



Commands

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

StartTimer
30 minutes

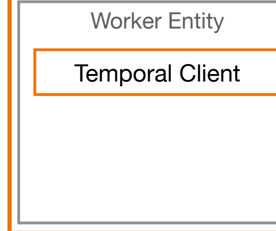
ScheduleActivityTask
Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

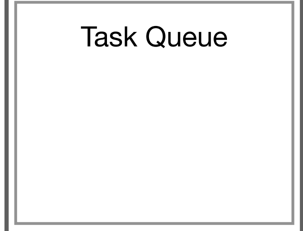
Events

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
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

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

StartTimer
30 minutes

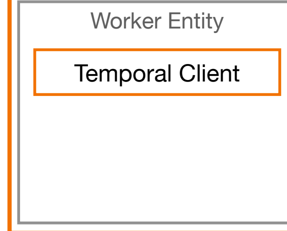
ScheduleActivityTask
Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

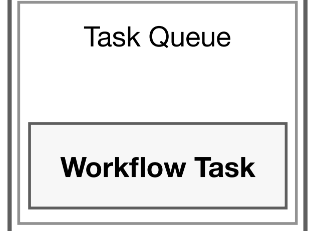
Events

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
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03 
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08 
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13 
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15 
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19 
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23 
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25 
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28 
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32 
33     return confirmation, nil
34 }
```





Commands

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

StartTimer
30 minutes

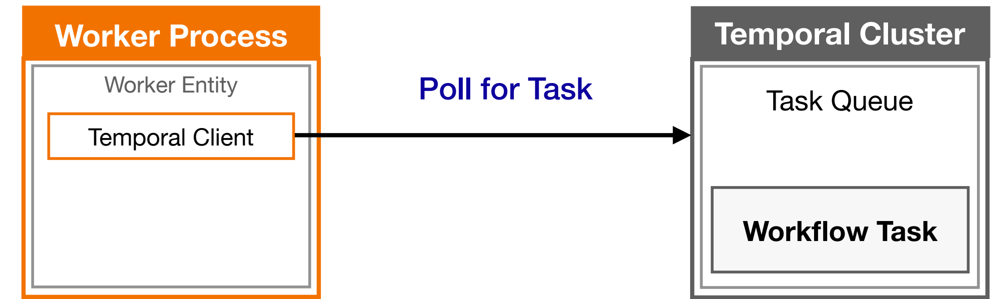
ScheduleActivityTask
Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

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
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



Commands

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

StartTimer
30 minutes

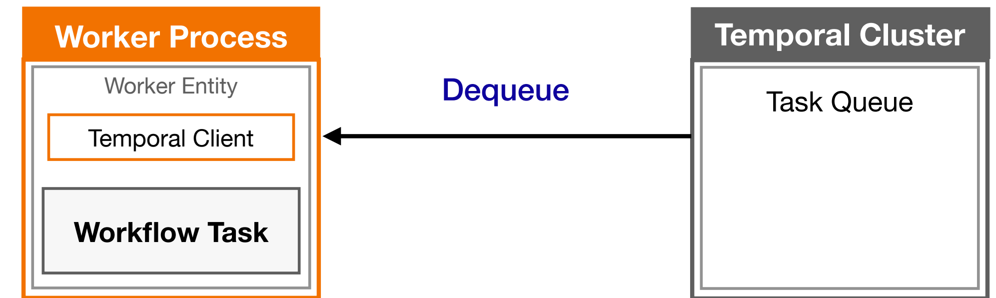
ScheduleActivityTask
Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

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
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



Commands

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

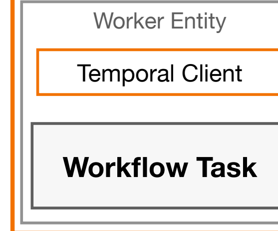
StartTimer
30 minutes

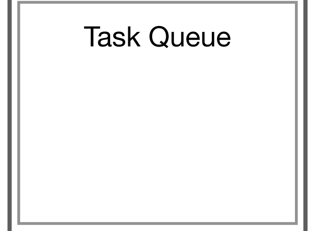
ScheduleActivityTask	
Queue:	pizza-tasks
Type:	SendBill
Input:	"CustomerID": 12983, ...

Events

Event	
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	

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

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

StartTimer
30 minutes

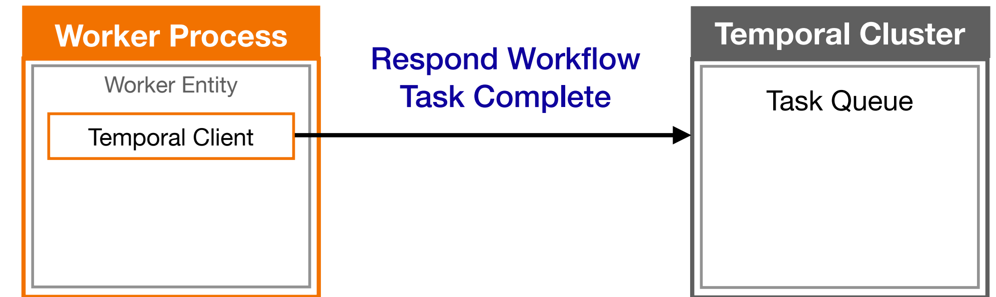
ScheduleActivityTask
Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

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

```
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03 
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08 
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13 
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15 
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19 
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23 
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25 
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28 
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32 
33     return confirmation, nil
34 }
```



Commands

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

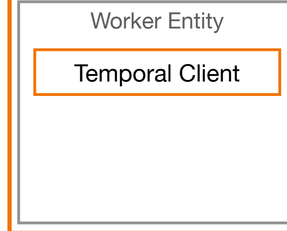
StartTimer
30 minutes

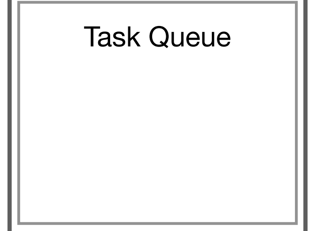
ScheduleActivityTask	
Queue:	pizza-tasks
Type:	SendBill
Input:	"CustomerID": 12983, ...

Events

Event	
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
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





Commands

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

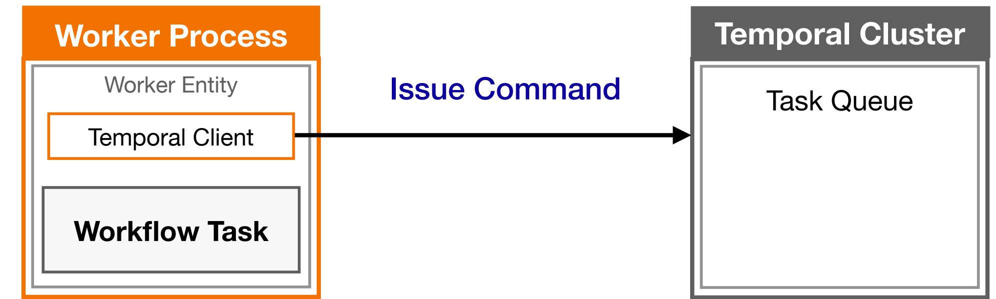
StartTimer
30 minutes

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
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```



Commands

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

StartTimer
30 minutes

ScheduleActivityTask
Queue: pizza-tasks
Type: SendBill
Input: "CustomerID": 12983, ...

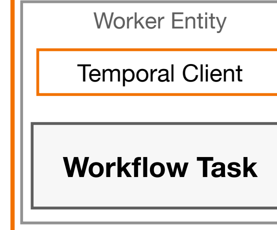
CompleteWorkflowExecution
Result: "ConfirmationNumber": "TPD-26074139"

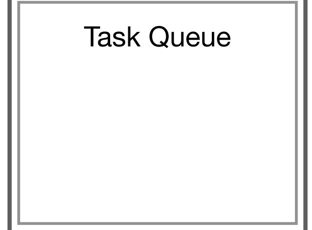
Events

- 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
01 func PizzaWorkflow(ctx workflow.Context, order PizzaOrder) (string, error) {
02     logger := workflow.GetLogger(ctx)
03
04     options := workflow.ActivityOptions{
05         StartToCloseTimeout: time.Second * 5,
06     }
07     ctx = workflow.WithActivityOptions(ctx, options)
08
09     var totalPrice int
10     for _, pizza := range order.Items {
11         totalPrice += pizza.Price
12     }
13
14     logger.Info("Calculated cost of order", "Total", totalPrice)
15
16     var distance Distance
17     future := workflow.ExecuteActivity(ctx, GetDistance, order.Address)
18     _ = future.Get(ctx, &distance)
19
20     if order.IsDelivery && distance.Kilometers > 25 {
21         return "", errors.New("customer lives too far away for delivery")
22     }
23
24     _ = workflow.Sleep(ctx, time.Minute * 30)
25
26     // call a local function to create the input passed to next Activity
27     bill := createBill(order, totalPrice)
28
29     var confirmation OrderConfirmation
30     future = workflow.ExecuteActivity(ctx, SendBill, bill)
31     _ = future.Get(ctx, &confirmation)
32
33     return confirmation, nil
34 }
```





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

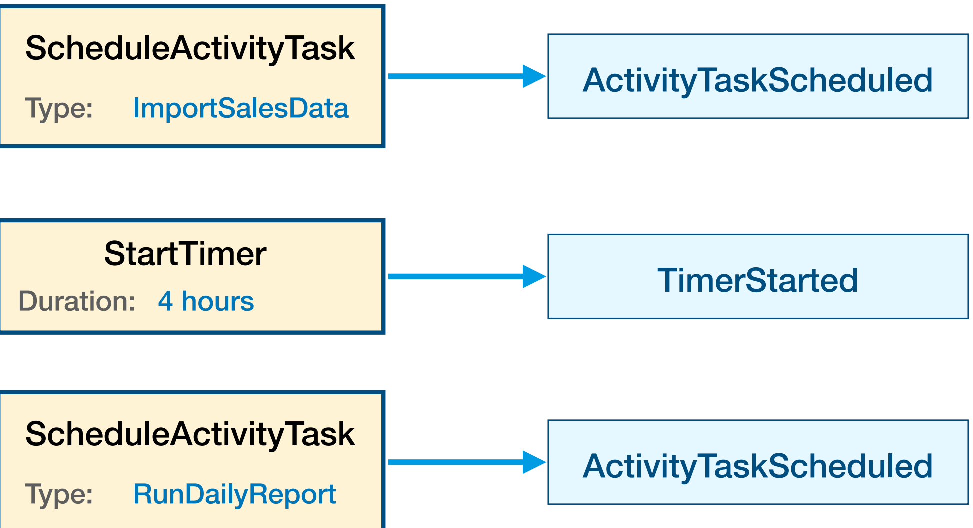
```
01 func DeterministicWorkflow(ctx workflow.Context) error {
02
03     options := workflow.ActivityOptions{
04         StartToCloseTimeout: time.Minute * 45,
05     }
06     ctx = workflow.WithActivityOptions(ctx, options)
07
08     err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
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
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 **Events**



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

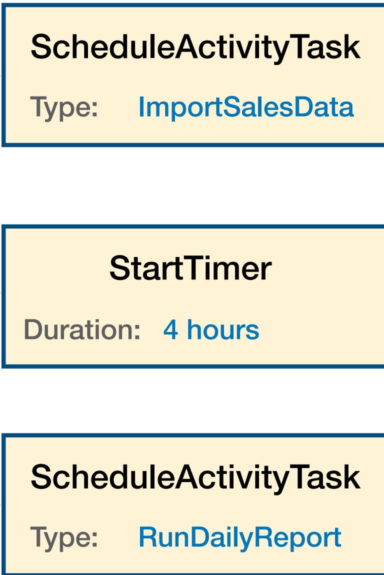
```
01 func DeterministicWorkflow(ctx workflow.Context) error {
02
03     options := workflow.ActivityOptions{
04         StartToCloseTimeout: time.Minute * 45,
05     }
06     ctx = workflow.WithActivityOptions(ctx, options)
07
08     err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
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

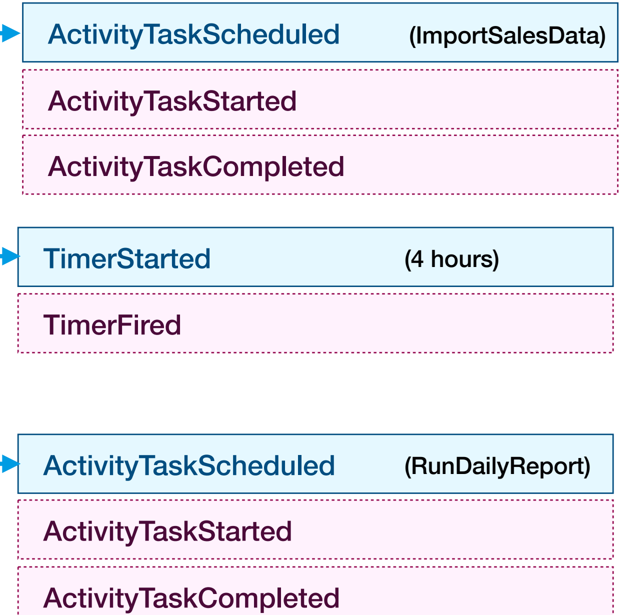
Workflow Definition

```
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 }
```

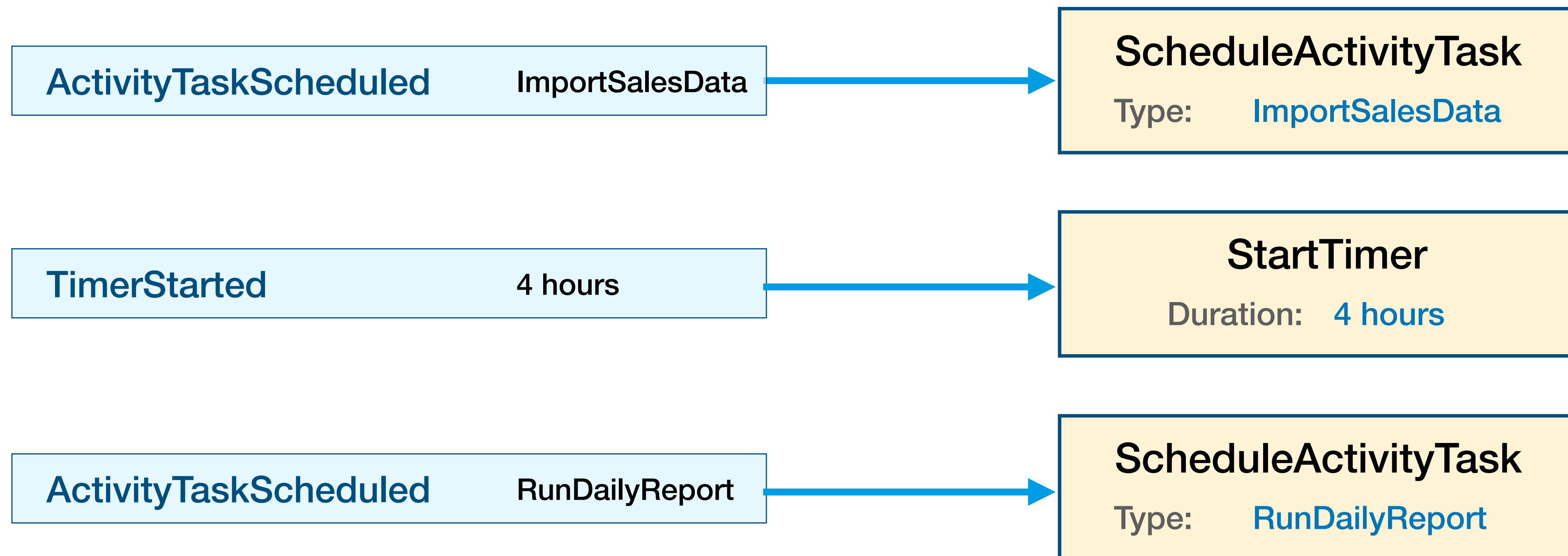
Commands



Events



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

```
01 func NonDeterministicWorkflow(ctx workflow.Context) error {
02
03     options := workflow.ActivityOptions{
04         StartToCloseTimeout: time.Minute * 45,
05     }
06     ctx = workflow.WithActivityOptions(ctx, options)
07
08     // this Activity is always executed
09     err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
10     if err != nil {
11         return err
12     }
13
14     if rand.Intn(100) >= 50 {
15        workflow.Sleep(ctx, time.Hour * 4)
16     }
17
18     workflow.GetLogger(ctx).Info("Preparing to run daily report")
19     err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
20     if err != nil {
21         return err
22     }
23
24     return nil
25 }
```

Commands Created

ScheduleActivityTask
Type: ImportSalesData

Relevant Events Logged

ActivityTaskScheduled (ImportSalesData)
ActivityTaskStarted
ActivityTaskCompleted

A Non-Deterministic Workflow Definition

```
01 func NonDeterministicWorkflow(ctx workflow.Context) error {
02
03     options := workflow.ActivityOptions{
04         StartToCloseTimeout: time.Minute * 45,
05     }
06     ctx = workflow.WithActivityOptions(ctx, options)
07
08     // this Activity is always executed
09     err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
10     if err != nil {
11         return err
12     }
13
14     if rand.Intn(100) >= 50 {
15        workflow.Sleep(ctx, time.Hour * 4)
16     }
17
18     workflow.GetLogger(ctx).Info("Preparing to run daily report")
19     err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
20     if err != nil {
21         return err
22     }
23
24     return nil
25 }
```

Commands Created

ScheduleActivityTask
Type: ImportSalesData

Relevant Events Logged

ActivityTaskScheduled (ImportSalesData)
ActivityTaskStarted
ActivityTaskCompleted

A Non-Deterministic Workflow Definition

```
01 func NonDeterministicWorkflow(ctx workflow.Context) error {
02
03     options := workflow.ActivityOptions{
04         StartToCloseTimeout: time.Minute * 45,
05     }
06     ctx = workflow.WithActivityOptions(ctx, options)
07
08     // this Activity is always executed
09     err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
10     if err != nil {
11         return err
12     }
13
14     if rand.Intn(100) >= 50 {
15        workflow.Sleep(ctx, time.Hour * 4)
16     }
17
18     workflow.GetLogger(ctx).Info("Preparing to run daily report")
19     err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
20     if err != nil {
21         return err
22     }
23
24     return nil
25 }
```

Happens to return 84 during this execution

Commands Created

ScheduleActivityTask
Type: ImportSalesData

Relevant Events Logged

ActivityTaskScheduled (ImportSalesData)
ActivityTaskStarted
ActivityTaskCompleted

A Non-Deterministic Workflow Definition

```
01 func NonDeterministicWorkflow(ctx workflow.Context) error {
02
03     options := workflow.ActivityOptions{
04         StartToCloseTimeout: time.Minute * 45,
05     }
06     ctx = workflow.WithActivityOptions(ctx, options)
07
08     // this Activity is always executed
09     err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
10     if err != nil {
11         return err
12     }
13
14     if rand.Intn(100) >= 50 {
15        workflow.Sleep(ctx, time.Hour * 4)
16     }
17
18     workflow.GetLogger(ctx).Info("Preparing to run daily report")
19     err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
20     if err != nil {
21         return err
22     }
23
24     return nil
25 }
```

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

```
01 func NonDeterministicWorkflow(ctx workflow.Context) error {
02
03     options := workflow.ActivityOptions{
04         StartToCloseTimeout: time.Minute * 45,
05     }
06     ctx = workflow.WithActivityOptions(ctx, options)
07
08     // this Activity is always executed
09     err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
10     if err != nil {
11         return err
12     }
13
14     if rand.Intn(100) >= 50 {
15        workflow.Sleep(ctx, time.Hour * 4)
16     }
17
18     workflow.GetLogger(ctx).Info("Preparing to run daily report")
19     err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
20     if err != nil {
21         return err
22     }
23
24     return nil
25 }
```

Worker crashes here

Commands Created

ScheduleActivityTask
Type: ImportSalesData

StartTimer
Duration: 4 hours

Relevant Events Logged

ActivityTaskScheduled (ImportSalesData)
ActivityTaskStarted
ActivityTaskCompleted

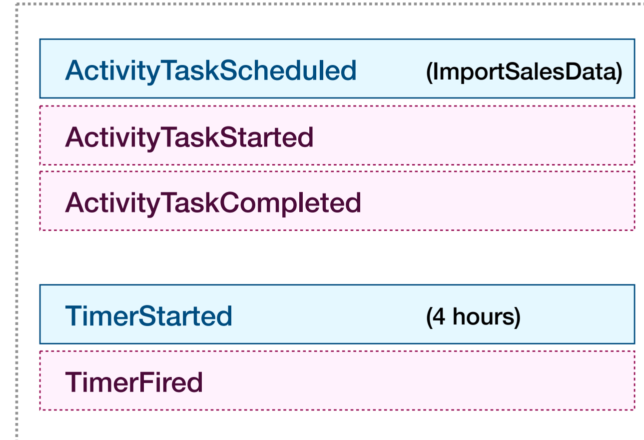
TimerStarted (4 hours)
TimerFired

A Non-Deterministic Workflow Definition

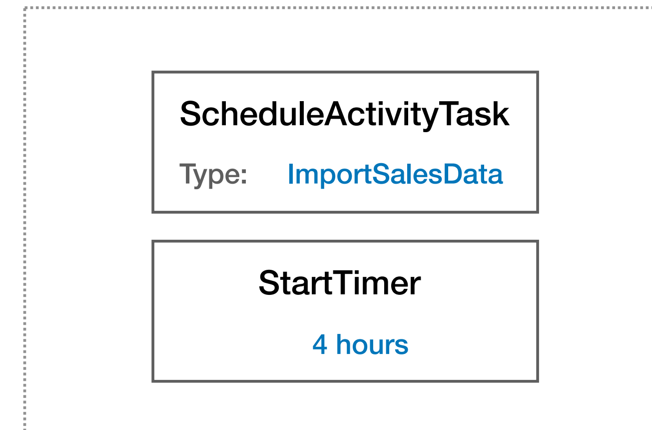
```
01 func NonDeterministicWorkflow(ctx workflow.Context) error {
02
03     options := workflow.ActivityOptions{
04         StartToCloseTimeout: time.Minute * 45,
05     }
06     ctx = workflow.WithActivityOptions(ctx, options)
07
08     // this Activity is always executed
09     err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
10     if err != nil {
11         return err
12     }
13
14     if rand.Intn(100) >= 50 {
15        workflow.Sleep(ctx, time.Hour * 4)
16     }
17
18     workflow.GetLogger(ctx).Info("Preparing to run daily report")
19     err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
20     if err != nil {
21         return err
22     }
23
24     return nil
25 }
```

Commands Created

Relevant History Events



Commands Expected (Based on History)



A Non-Deterministic Workflow Definition

```
01 func NonDeterministicWorkflow(ctx workflow.Context) error {
02
03     options := workflow.ActivityOptions{
04         StartToCloseTimeout: time.Minute * 45,
05     }
06     ctx = workflow.WithActivityOptions(ctx, options)
07
08     // this Activity is always executed
09     err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
10     if err != nil {
11         return err
12     }
13
14     if rand.Intn(100) >= 50 {
15        workflow.Sleep(ctx, time.Hour * 4)
16     }
17
18     workflow.GetLogger(ctx).Info("Preparing to run daily report")
19     err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
20     if err != nil {
21         return err
22     }
23
24     return nil
25 }
```

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

```
01 func NonDeterministicWorkflow(ctx workflow.Context) error {
02
03     options := workflow.ActivityOptions{
04         StartToCloseTimeout: time.Minute * 45,
05     }
06     ctx = workflow.WithActivityOptions(ctx, options)
07
08     // this Activity is always executed
09     err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
10     if err != nil {
11         return err
12     }
13
14     if rand.Intn(100) >= 50 {
15        workflow.Sleep(ctx, time.Hour * 4)
16     }
17
18     workflow.GetLogger(ctx).Info("Preparing to run daily report")
19     err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
20     if err != nil {
21         return err
22     }
23
24     return nil
25 }
```

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

```
01 func NonDeterministicWorkflow(ctx workflow.Context) error {
02
03     options := workflow.ActivityOptions{
04         StartToCloseTimeout: time.Minute * 45,
05     }
06     ctx = workflow.WithActivityOptions(ctx, options)
07
08     // this Activity is always executed
09     err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
10     if err != nil {
11         return err
12     }
13
14     if rand.Intn(100) >= 50 {
15        workflow.Sleep(ctx, time.Hour * 4)
16     }
17
18     workflow.GetLogger(ctx).Info("Preparing to run daily report")
19     err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
20     if err != nil {
21         return err
22     }
23
24     return nil
25 }
```

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

```
01 func NonDeterministicWorkflow(ctx workflow.Context) error {
02
03     options := workflow.ActivityOptions{
04         StartToCloseTimeout: time.Minute * 45,
05     }
06     ctx = workflow.WithActivityOptions(ctx, options)
07
08     // this Activity is always executed
09     err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
10     if err != nil {
11         return err
12     }
13
14     if rand.Intn(100) >= 50 {
15        workflow.Sleep(ctx, time.Hour * 4)
16     }
17
18     workflow.GetLogger(ctx).Info("Preparing to run daily report")
19     err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
20     if err != nil {
21         return err
22     }
23
24     return nil
25 }
```

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
01 func NonDeterministicWorkflow(ctx workflow.Context) error {
02
03     options := workflow.ActivityOptions{
04         StartToCloseTimeout: time.Minute * 45,
05     }
06     ctx = workflow.WithActivityOptions(ctx, options)
07
08     // this Activity is always executed
09     err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
10     if err != nil {
11         return err
12     }
13
14     if rand.Intn(100) >= 50 {
15        workflow.Sleep(ctx, time.Hour * 4)
16     }
17
18     workflow.GetLogger(ctx).Info("Preparing to run daily report")
19     err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
20     if err != nil {
21         return err
22     }
23
24     return nil
25 }
```

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

```
01 func NonDeterministicWorkflow(ctx workflow.Context) error {
02
03     options := workflow.ActivityOptions{
04         StartToCloseTimeout: time.Minute * 45,
05     }
06     ctx = workflow.WithActivityOptions(ctx, options)
07
08     // this Activity is always executed
09     err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
10     if err != nil {
11         return err
12     }
13
14     if rand.Intn(100) >= 50 {
15        workflow.Sleep(ctx, time.Hour * 4)
16     }
17
18     workflow.GetLogger(ctx).Info("Preparing to run daily report")
19     err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
20     if err != nil {
21         return err
22     }
23
24     return nil
25 }
```

A Non-Deterministic Workflow Definition

```
01 func NonDeterministicWorkflow(ctx workflow.Context) error {
02
03     options := workflow.ActivityOptions{
04         StartToCloseTimeout: time.Minute * 45,
05     }
06     ctx = workflow.WithActivityOptions(ctx, options)
07
08     // this Activity is always executed
09     err := workflow.ExecuteActivity(ctx, ImportSalesData).Get(ctx, nil)
10     if err != nil {
11         return err
12     }
13
14     if rand.Intn(100) >= 50 {
15        workflow.Sleep(ctx, time.Hour * 4)
16     }
17
18     workflow.GetLogger(ctx).Info("Preparing to run daily report")
19     err = workflow.ExecuteActivity(ctx, RunDailyReport).Get(ctx, nil)
20     if err != nil {
21         return err
22     }
23
24     return nil
25 }
```

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`

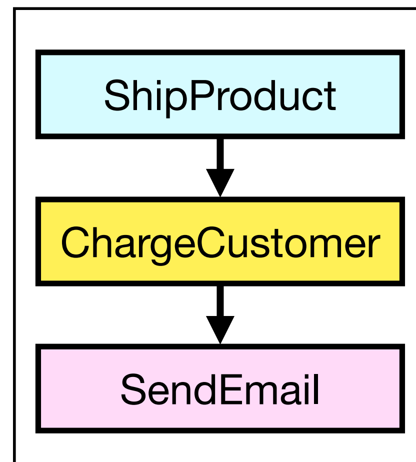
Things to Avoid in a Workflow Definition (2)

- **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

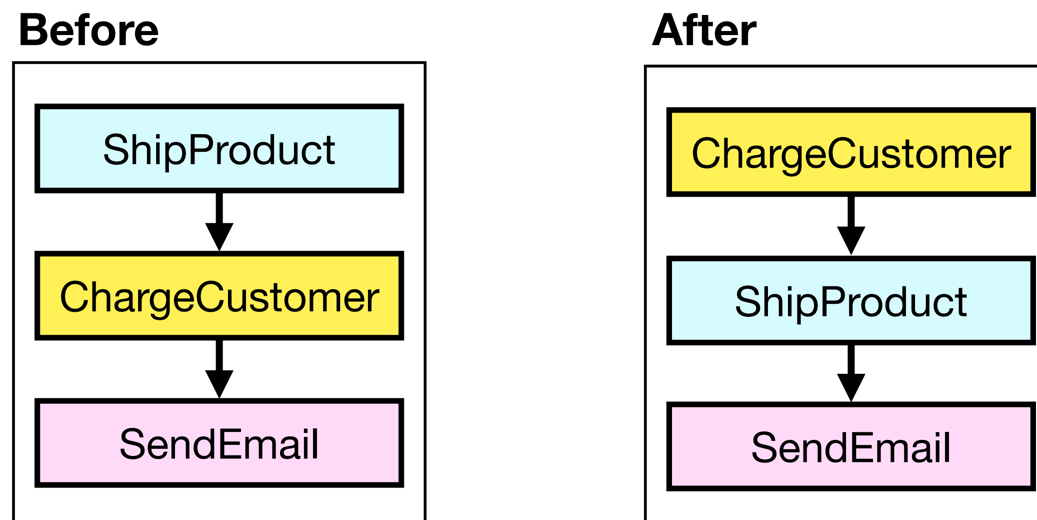
Non-Deterministic *Code* Isn't the Only Danger

- **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)

Back to Workflows
process-order-24577
Completed
Reset
History 24 Workers 0 Pending Activities 0 Stack Trace Queries
Summary
Workflow Type: ProcessOrder
77cd8a20-8de4-4681-b854-4568fbfc02e7
Task Queue: wiffle-workflow-tasks
State Transitions: 17
Start & Close Time
Start Time: 2023-08-23 CDT 18:59:21.29
Close Time: 2023-08-23 CDT 19:00:37.57
Relationships 0 Parents 0 Pending Children 0 Children 0 First 0 Previous 0 Next
Input and Results
Recent Events 100 1–24 of 24 History Compact JSON Download
Date & Time | Workflow Events | Expand All
24 2023-08-23 CDT 19:00:37.57 WorkflowExecutionCompleted Workflow Task Completed Event ID 23

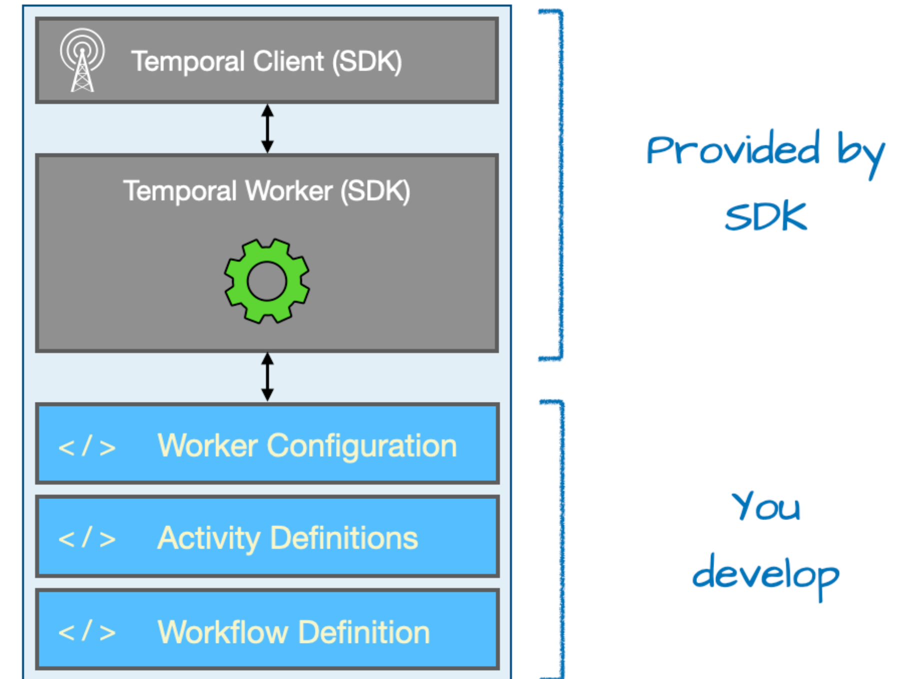
```
replayer := worker.NewWorkflowReplayer()
replayer.RegisterWorkflow("ProcessOrder")
err := replayer.ReplayWorkflowHistoryFromJSONFile("/Users/twheeler/Downloads/myhistory.json")
```

Temporal 102

00. About this Workshop
01. Understanding Key Concepts in Temporal
02. Improving Your Temporal Application Code
03. Using Timers in a Workflow Definition
04. Testing Your Temporal Application Code
05. Understanding Event History
06. Debugging Workflow History
07. Deploying Your Application to Production
08. Understanding Workflow Determinism
▶ **09. Conclusion**

Essential Points (1)

- **Temporal applications contain code that you develop**
    - Workflow and Activity Definitions, Worker Configuration, etc.
- **Temporal applications also contain SDK-provided code**
    - Such as the implementations of the Worker and Temporal Client
- **Temporal guarantees durable execution of Workflows**
    - If the Worker crashes, another Worker uses History Replay to automatically recreate pre-crash state, then continues execution
    - From the developer perspective, it's as if the crash never even happened



Essential Points (2)

- **Temporal Cluster / Cloud perform orchestration via Task Queues**
  - A Worker polls a Task Queue, accepts a Task, executes the code, and reports back with status/results
  - Communication takes place by Workers initiating requests via gRPC to the Frontend Service
  - **Key point**: Execution of the code is external to Temporal Cluster / Cloud
- **As Workers run your code, they send Commands to Temporal Cluster/Cloud**
  - For example, when encountering calls to `workflow.ExecuteActivity` or `workflow.Sleep` or when returning a result from the Workflow Definition
- **Commands sent by the Worker lead to Events logged by Temporal Cluster / Cloud**

Essential Points (3)

- **The Event History documents the details of a Workflow Execution**
  - It's an ordered append-only list of Events
  - Temporal enforces limits on the size and item count of the Event History
- **Every Event has three attributes in common: ID, timestamp, and type**
  - They will also have additional attributes, which vary by Event Type
  - Examining the Event History and attributes of individual Events can help you debug Workflow Executions

Essential Points (4)

- **A single Workflow Definition can be executed any number of times**
  - Each time potentially having different input data and a different Workflow ID
    - At most, one open Workflow Execution with a given Workflow ID is allowed per Namespace
    - This rule applies to *all* Workflow Executions, not just ones of the same Workflow Type
- **Once started, Workflow Execution enters the Open state**
  - Execution typically alternates between making progress and awaiting a condition
  - When execution concludes, it transitions to the Closed state
  - There are several subtypes of Closed, including Completed, Failed, and Terminated

Essential Points (5)

- **Temporal requires that your Workflow code is deterministic**
    - This constraint is what makes durable execution possible
    - Temporal's definition of determinism: Every execution of a given Workflow Definition must produce an identical sequence of Commands, given the same input
    - Non-deterministic errors can occur because of something inherently non-deterministic in the code
        - Can also occur after deploying a code change that changes the Command sequence, if there were open executions of the same Workflow Type at the time of deployment
- **Activities are used for code that interacts with the outside world**
    - Activity code isn't required to be deterministic (but it should be idempotent)
    - Activities are automatically retried upon failure, according to a configurable Retry Policy

Essential Points (6)

- **Recommended best practices for Temporal app development**
  - Use structs (not individual fields) as input/output of your Workflow and Activity definitions
  - Be aware of the platform's limits on Event History size and item count
  - Use `workflowcheck` (Go-specific) to scan your Workflow Definitions for non-deterministic code
  - Replace non-deterministic code in Workflow Definitions with Workflow-safe counterparts
  - Use Temporal's replay-aware logging API, ideally integrating with a 3rd-party logging package

Essential Points (7)

- **We don't dictate how to build, deploy, or run Temporal applications**
    - Typical advice: Build, deploy, and run as you would any other application in that language
    - However, we recommend running >= 2 Workers per Task Queue (availability/scalability)

Thank You

Unpublished version - TW20220829a