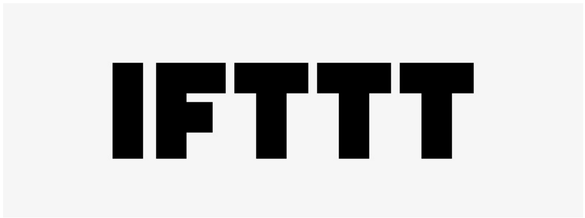
Workflow Tools for Devops

Mohammed Firdaus
April 21 2022

About Me

- I mostly program in Java, Kotlin and Python
- https://www.linkedin.com/in/mohammed-firdaus-mohammed-ab-halim-55684515/
- https://github.com/firdaus
- Blog @ https://onepointzero.app/
- Youtube - https://onepointzero.app/yt
- Email me - onepointzeroapp@outlook.com

What in the world does the term “workflow” mean anyway?



Workflow

- A workflow is a sequence of tasks or operations. Analogous terms: Pipelines, Business Process

- A workflow engine is a software application that helps you define, execute and manage workflows.

Problem Statement: scan_and_fix()

```
def scan_and_fix(host):
    issues = scan_host(host)
    apply_fixes(host, issues)
    unresolved_issues = rescan(host)
    if unresolved_issues:
        failure_report(host, unresolved_issues)
    else:
        success_report(host)


if __name__ == '__main__':
    scan_and_fix("192.168.56.1")
```

Issues

- What happens when the machine running this function dies when the code is still running?
- What happens if any of the operations fail?

Positive vs Negative Engineering

- Positive engineering is what we typically think engineers do: write code to achieve an objective.

- Negative engineering is when engineers write defensive code to make sure the positive code actually runs. For example: what happens if data arrives malformed? What if the database goes down? What if the computer running the code fails? What if the code succeeds but the computer fails before it can report the success? Negative engineering is characterized by needing to anticipate this infinity of possible failures.

- Workflow tools handle the negative engineering so that you can focus on the positive engineering.

Adapted From https://docs.prefect.io/core/ and https://medium.com/the-prefect-blog/positive-and-negative-data-engineering-a02cb497583d

Types of Workflow Engines

- We are going to compare and contrast workflow engines by how you define the workflow i.e. how you do the positive engineering.
  - Embedded DSL - the workflow structure is defined as a DAG in a host language e.g. Airflow, Prefect
  - External DSL - The workflow is defined in a separate format (JSON, YAML, XML, Graphical) e.g. Step Functions, BPMN tools, Netflix conductor, NiFi, Stackstorm. These tools are often paired with a graphical workflow editor.
  - Language SDK - the workflow features are integrated into the language itself e.g. Temporal, Cadence, Azure Durable Functions, Infinitic

Embedded DSL:Airflow

- Workflows are defined as DAGs - Directed acyclic graphs (means no loops) - in the host language of Python
- Operators define business logic that you can add to a workflow e.g. a PythonOperator is used to execute Python Code, S3CreateBucketOperator is used to create an S3 bucket (additional functionality available by installing packages called “providers”)
- A task is an instance of an Operator
- The sequence/order of tasks is defined using the “>>” symbol in Python
- Very popular in the data engineering space for data pipelines
- xcom is used to communicate between tasks

https://www.astronomer.io/guides/what-is-an-operator/

Airflow: Defining Tasks

Creating a task from an operator

```
download_file = SFTPOperator(
    task_id="get-file",
    ssh_conn_id="my_sftp_server",
    remote_filepath="/{{ ds }}/input.csv",
    local_filepath="/tmp/{{ run_id }}/input.csv",
    operation="get",
    create_intermediate_dirs=True
)
```

Creating a task to execute arbitrary Python code (Python operator)

```
@task()
def do_custom_python_stuff:
    ..........
```

Problem Statement: scan_and_fix()

```
def scan_and_fix(host):
    issues = scan_host(host)
    apply_fixes(host, issues)
    unresolved_issues = rescan(host)
    if unresolved_issues:
        failure_report(host, unresolved_issues)
    else:
        success_report(host)


if __name__ == '__main__':
    scan_and_fix("192.168.56.1")
```

Airflow: Scan and Fix Workflow

```
@dag(
    schedule_interval=None,
    start_date=pendulum.datetime(2021, 1, 1, tz="UTC"),
    params={
        "host": Param(type='string')
    }
)
def scan_and_fix():
    choose_report_task = BranchPythonOperator(task_id="choose_report",
                                              python_callable=choose_report)

    scan_host() >> apply_fixes() >> rescan() >> choose_report_task

    choose_report_task >> failure_report()
    choose_report_task >> success_report()
```

Airflow: Scan Host task

```
@task(retries=10, retry_delay=timedelta(seconds=5))
def scan_host(params=None):
    host = params["host"]
    print(f"Scanning host {host}")
    #
    # Real scanning code goes here
    #
    return [Issue()]
```

External DSL: AWS Step Functions

AWS Step Functions is a low-code, visual workflow service that developers use to build distributed applications, automate IT and business processes, and build data and machine learning pipelines using AWS services. Workflows manage failures, retries, parallelization, service integrations, and observability so developers can focus on higher-value business logic.

https://aws.amazon.com/step-functions/

AWS Step Functions

- Workflows be authored in the Amazon States Language or using the graphical “Workflow Studio” (support round-trip engineering)

- “The Amazon States Language is a JSON-based, structured language used to define your state machine, a collection of states, that can do work (Task states), determine which states to transition to next (Choice states), stop an execution with an error (Fail states), and so on.”

- “All work in your state machine is done by *tasks*. A task performs work by using an activity or an AWS Lambda function, or by passing parameters to the API actions of other services.”

https://docs.aws.amazon.com/step-functions/latest/dg/concepts-amazon-states-language.html

https://docs.aws.amazon.com/step-functions/latest/dg/amazon-states-language-task-state.html

Step Functions: Scan and Fix Workflow

```
{
  "Comment": "A description of my state machine",
  "StartAt": "Scan host",
  "States": {
    "Scan host": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "OutputPath": "$.Payload",
      "Parameters": {
        "Payload.$": "$",
        "FunctionName": "arn:aws:lambda:ap-southeast-1:263872182581:function:scan_host:$LATEST"
      },
      "Retry": [...],
      "Next": "Apply Fixes"
    },
    "Apply Fixes": {"Type": "Task"...},
    "Rescan host": {"Type": "Task"...},
    "Any Unresolved Issues?": {"Type": "Choice"...},
    "Send Success Report": {"Type": "Task"...},
    "Send Failure Report": {"Type": "Task"...}
  }
}
```

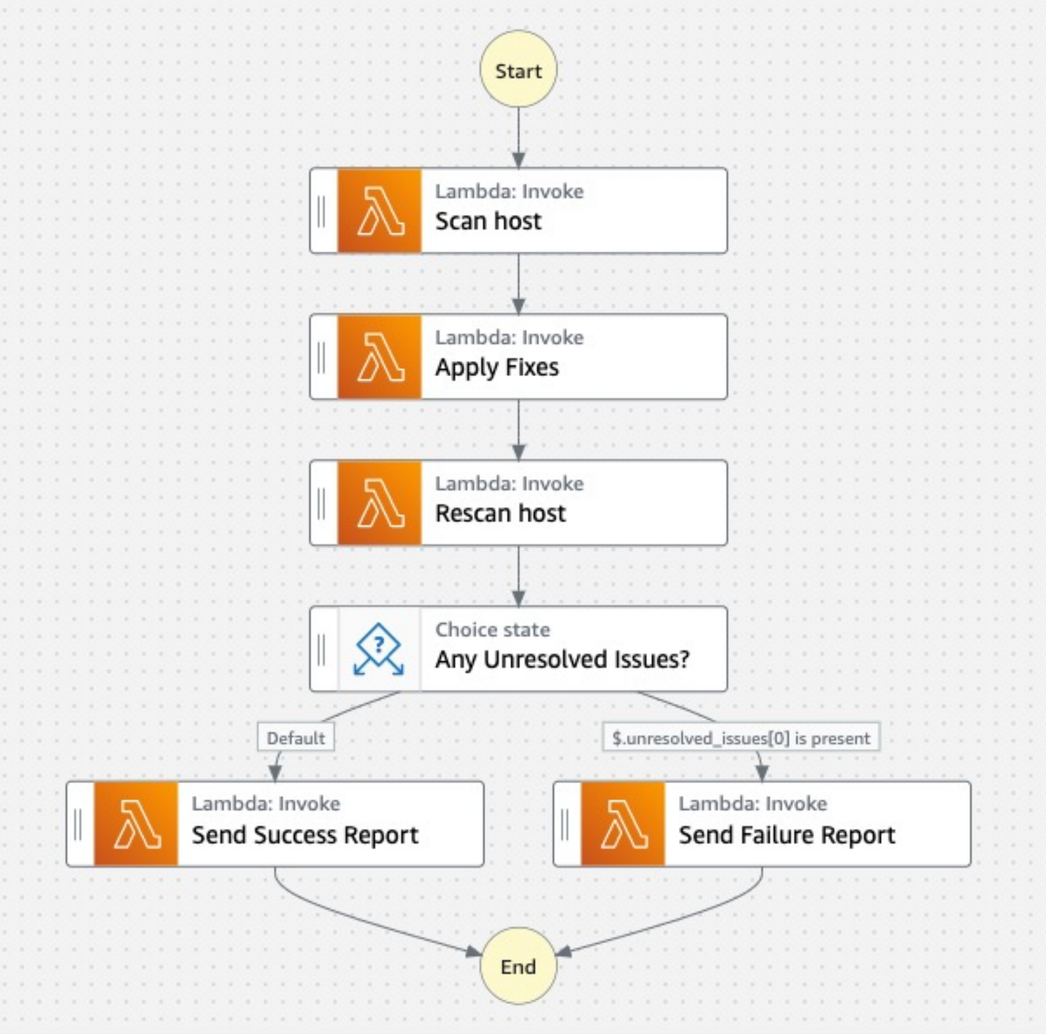
Step Functions: Scan and Fix Workflow

```
{
  "Comment": "A description of my state machine",
  "StartAt": "Scan host",
  "States": {
    "Scan host": {"Type": "Task"...},
    "Apply Fixes": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "OutputPath": "$.Payload",
      "Parameters": {
        "Payload.$": "$",
        "FunctionName": "arn:aws:lambda:ap-southeast-1:263872182581:function:apply_fixes:$LATEST"
      },
      "Retry": [...],
      "Next": "Rescan host"
    },
    "Rescan host": {"Type": "Task"...},
    "Any Unresolved Issues?": {"Type": "Choice"...},
    "Send Success Report": {"Type": "Task"...},
    "Send Failure Report": {"Type": "Task"...}
  }
}
```

Step Functions: Scan and Fix Workflow

```
{
  "Comment": "A description of my state machine",
  "StartAt": "Scan host",
  "States": {
    "Scan host": {"Type": "Task"...},
    "Apply Fixes": {"Type": "Task"...},
    "Rescan host": {"Type": "Task"...},
    "Any Unresolved Issues?": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.unresolved_issues[0]",
          "IsPresent": true,
          "Next": "Send Failure Report"
        }
      ],
      "Default": "Send Success Report"
    },
    "Send Success Report": {"Type": "Task"...},
    "Send Failure Report": {"Type": "Task"...}
  }
}
```

Step Functions: Scan and Fix Workflow



Step Functions: scan_host lambda function

```
from dataclasses import dataclass, asdict

@dataclass
class Issue:
    id: str = ""
    context: str = ""


def lambda_handler(event, context):
    host = event["host"]
    print(f"Scanning host {host}")
    #
    # Actual scanning logic goes here
    #
    event["issues"] = [asdict(Issue()), asdict(Issue())]
    return event
```

Language SDK: Temporal

- Available at - https://www.temporal.io/ - Yes, it's a workflow engine
- Written in Golang
- Fork of Uber Cadence, developed by Temporal Technologies
- Allows you to write workflows as functions in the language of your choice (provided an SDK exists for that language) - SDKs exists for Java, Golang, PHP and Typescript (Python is in development)
- Workflows in Temporal are just long running functions - can take days, months or years to complete
- Not a DSL, Not a graphical tool, Not YAML.

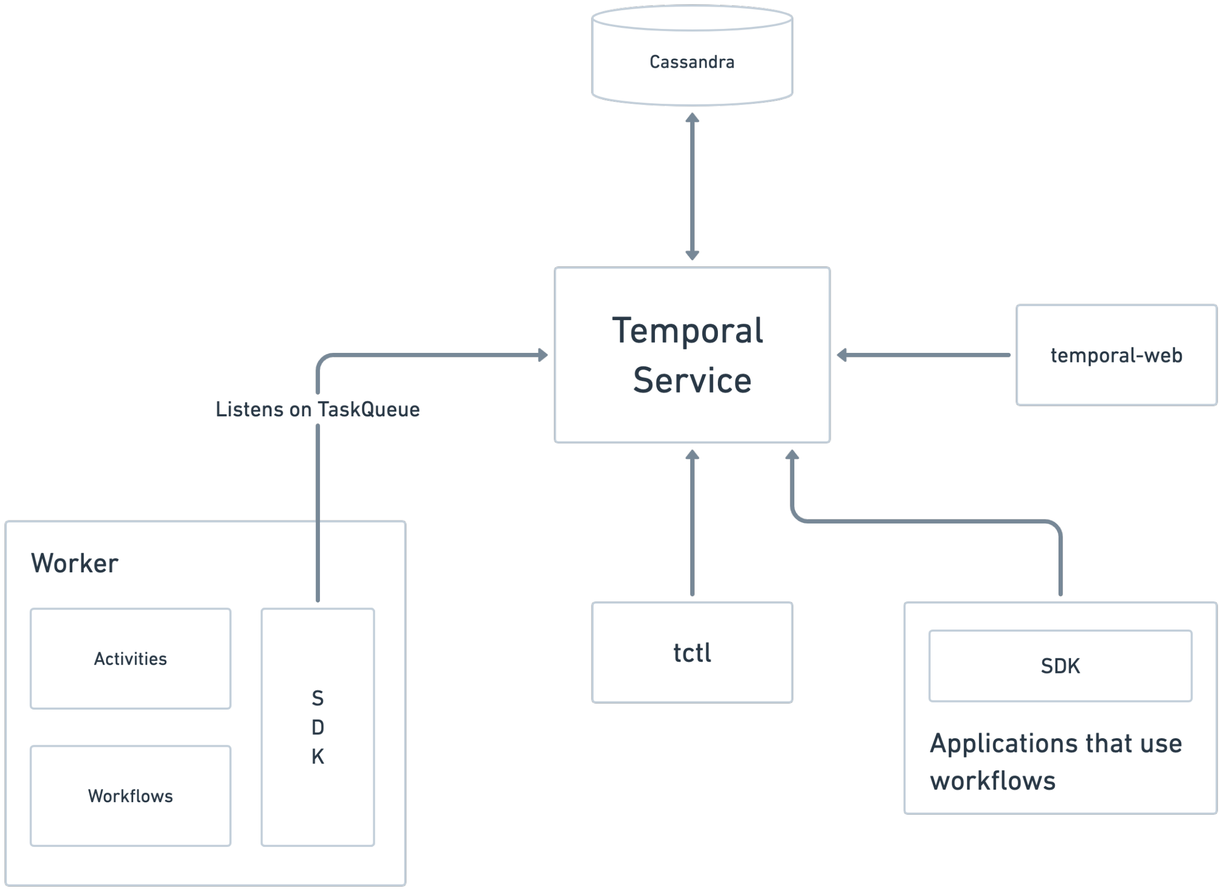
A little bit more theory

Code needs to separate into two types of methods:

- Workflow Functions - Workflows contain the business logic to be implemented. It should contain logic for coordinating/orchestration but should not “affect” the outside world directly. Generally speaking, the workflow ends when the workflow method “returns”.

- Activity Functions - For defining tasks. Activity methods are how the workflow methods can change the outside world. Any type of logic that could possibly have a side effect should be done in an activity method e.g. API calls to other systems, database queries etc..

Temporal: High Level View



Temporal: Scan and Fix Workflow

```
@WorkflowInterface
public interface ScanAndFixWorkflow {
    @WorkflowMethod
    void scanAndFix(String host);
}
```

Temporal: Scan and Fix Workflow

```
public static class ScanAndFixWorkflowImpl implements ScanAndFixWorkflow {
    ActivityOptions options = ActivityOptions.newBuilder()
            .setScheduleToCloseTimeout(Duration.ofSeconds(2))
            .build();
    private ScanAndFixActivities activities = Workflow.newActivityStub(ScanAndFixActivities.class,
                                                                       options);
    private Logger logger = Workflow.getLogger(ScanAndFixWorkflowImpl.class);

    @Override
    public void scanAndFix(String host) {
        logger.info("Scanning and fixing: {}", host);
        List<Issue> issues = activities.scanHost(host);
        activities.applyFixes(host, issues);
        List<Issue> unresolvedIssues = activities.rescan(host);
        if (unresolvedIssues.isEmpty()) {
            activities.successReport(host);
        } else {
            activities.failureReport(host, unresolvedIssues);
        }
    }
}
```

Temporal: ScanAndFix Activities

```java
@ActivityInterface
public interface ScanAndFixActivities {

    List<Issue> scanHost(String host);

    void applyFixes(String host, List<Issue> issues);

    List<Issue> rescan(String host);

    void successReport(String host);

    void failureReport(String host, List<Issue> unresolvedIssues);
}
```

Temporal: ScanAndFix Activities

```
public static class ScanAndFixActivitiesImpl implements ScanAndFixActivities  {
    public Logger logger = LoggerFactory.getLogger(ScanAndFixActivitiesImpl.class);

    @Override
    public List<Issue> scanHost(String host) {
        logger.info("Scanning {}", host);
        //
        // Real scanning code goes here
        //
        return List.of(new Issue(), new Issue());
    }

    // Other activity method definitions
    // ...
    // ...
}
```

Temporal: ScanAndFix Worker

```
public static void main(String[] args) {
    WorkflowServiceStubs service = WorkflowServiceStubs.newInstance();
    WorkflowClient client = WorkflowClient.newInstance(service);
    WorkerFactory factory = WorkerFactory.newInstance(client);
    Worker worker = factory.newWorker("scanandfix-tq");
    worker.registerWorkflowImplementationTypes(ScanAndFixWorkflowImpl.class);
    worker.registerActivitiesImplementations(new ScanAndFixActivitiesImpl());
    factory.start();
}
```

Starting the Temporal Service

```
$ git clone https://github.com/temporalio/docker-compose.git
$ cd  docker-compose
$ docker-compose up
$ alias tctl='docker exec temporal-admin-tools tctl'
```

```
$ alias tctl='docker run --network=host --rm temporalio/tctl:latest'

$ % tctl workflow run --taskqueue scanandfix-tq --workflow_type ScanAndFixWorkflow --input '"192.168.56.1"'
```

```
Running execution:
  Workflow Id : 22040ba1-264c-4228-adfb-5bf70d2e47a0
  Run Id      : e6c44945-4c2c-4650-9668-bc284da240e5
  Type        : ScanAndFixWorkflow
  Namespace   : default
  Task Queue  : scanandfix-tq
  Args        : ["192.168.56.1"]
Progress:
  1, 2022-04-20T05:05:20Z, WorkflowExecutionStarted
  2, 2022-04-20T05:05:20Z, WorkflowTaskScheduled
  3, 2022-04-20T05:05:20Z, WorkflowTaskStarted
  4, 2022-04-20T05:05:20Z, WorkflowTaskCompleted
  5, 2022-04-20T05:05:20Z, ActivityTaskScheduled
  6, 2022-04-20T05:05:20Z, ActivityTaskStarted
  7, 2022-04-20T05:05:20Z, ActivityTaskCompleted
  8, 2022-04-20T05:05:20Z, WorkflowTaskScheduled
  9, 2022-04-20T05:05:20Z, WorkflowTaskStarted
  10, 2022-04-20T05:05:20Z, WorkflowTaskCompleted
  11, 2022-04-20T05:05:20Z, ActivityTaskScheduled
  12, 2022-04-20T05:05:20Z, ActivityTaskStarted
  13, 2022-04-20T05:05:20Z, ActivityTaskCompleted
  14, 2022-04-20T05:05:20Z, WorkflowTaskScheduled
  15, 2022-04-20T05:05:20Z, WorkflowTaskStarted
  16, 2022-04-20T05:05:20Z, WorkflowTaskCompleted
  17, 2022-04-20T05:05:20Z, ActivityTaskScheduled
  18, 2022-04-20T05:05:20Z, ActivityTaskStarted
  19, 2022-04-20T05:05:20Z, ActivityTaskCompleted
  20, 2022-04-20T05:05:20Z, WorkflowTaskScheduled
  21, 2022-04-20T05:05:20Z, WorkflowTaskStarted
  22, 2022-04-20T05:05:20Z, WorkflowTaskCompleted
  23, 2022-04-20T05:05:20Z, ActivityTaskScheduled
  24, 2022-04-20T05:05:20Z, ActivityTaskStarted
  25, 2022-04-20T05:05:20Z, ActivityTaskCompleted
  26, 2022-04-20T05:05:20Z, WorkflowTaskScheduled
  27, 2022-04-20T05:05:20Z, WorkflowTaskStarted
  28, 2022-04-20T05:05:20Z, WorkflowTaskCompleted
  29, 2022-04-20T05:05:20Z, WorkflowExecutionCompleted

Result:
  Run Time: 1 seconds
  Status: COMPLETED
  Output:
```

ScanAndFix Output

```
13:05:20.405 [workflow-method-22040ba1-264c-4228-adfb-5bf70d2e47a0-e6c44945-4c2c-4650-9668-
bc284da240e5] INFO ScanAndFix$ScanAndFixWorkflowImpl - Scanning and fixing: 192.168.56.1
13:05:20.437 [Activity Executor taskQueue="scanandfix-tq", namespace="default": 2] INFO
ScanAndFix$ScanAndFixActivitiesImpl - Scanning 192.168.56.1
13:05:20.511 [Activity Executor taskQueue="scanandfix-tq", namespace="default": 2] INFO
ScanAndFix$ScanAndFixActivitiesImpl - Applying 2 fixes on host 192.168.56.1
13:05:20.587 [Activity Executor taskQueue="scanandfix-tq", namespace="default": 2] INFO
ScanAndFix$ScanAndFixActivitiesImpl - Rescanning 192.168.56.1
13:05:20.664 [Activity Executor taskQueue="scanandfix-tq", namespace="default": 2] INFO
ScanAndFix$ScanAndFixActivitiesImpl - Sending success report for host 192.168.56.1
```

Workflow functions can run for as long as you need them to

```
@Override
public void scanAndFix(String host) {
    logger.info("Scanning and fixing: {}", host);
    List<Issue> issues = activities.scanHost(host);
    activities.applyFixes(host, issues);

    Workflow.sleep(Duration.ofDays(2));

    List<Issue> unresolvedIssues = activities.rescan(host);
    if (unresolvedIssues.isEmpty()) {
        activities.successReport(host);
    } else {
        activities.failureReport(host, unresolvedIssues);
    }
}
```

Note about Workflow Functions

- Workflow functions need to be deterministic because they are replayed each time the state of the workflow needs to be rebuilt.
- From Wikipedia “a deterministic algorithm is an algorithm which, given a particular input, will always produce the same output, with the underlying machine always passing through the same sequence of states”.
- Practically this means that:
  - Use `Workflow.currentTimeMillis()` instead of `System.currentTimeMillis()` to get the current time
  - Use `Workflow.randomUUID()` to generate UUIDs, `Workflow.newRandom()` to generate random numbers
  - Use `Workflow.getLogger()` to obtain a logger that is replay aware
  - Anything that talks to the outside world should be done in activity functions.

More Temporal Concepts

- Signal Methods/Functions - While the workflow is running it be can be triggered by external code when certain events occur using signal methods.

- Query Methods/Functions - Query methods allow external code to pull information about the current state of the workflow.

E-commerce Loyalty Membership Tier

- Lets implement a loyalty program where customers are assigned a membership tier of MEMBER, SILVER, GOLD or PLATINUM based on the amount they spent the previous month.

- MEMBER spent < 100

- SILVER spent < 500

- GOLD spent < 1000

- PLATINUM for 1000 and above

Workflow Interface

```
@WorkflowInterface
public static interface LoyaltyWorkflow {
    @QueryMethod
    Tier currentTier();

    @QueryMethod
    long transactionsThisMonth();

    @SignalMethod
    void recordTransaction(long value);

    @WorkflowMethod
    void enroll(String customerId);
}
```

Signal and Query Methods

```
public static class LoyaltyWorkflowImpl implements LoyaltyWorkflow {
    long transactionsThisMonth = 0;
    Tier tier = Tier.MEMBER;

    @Override
    public Tier currentTier() {
        return tier;
    }

    @Override
    public long transactionsThisMonth() {
        return transactionsThisMonth;
    }


    @Override
    public void recordTransaction(long value) {
        transactionsThisMonth += value;
    }
    .
    .
```

Workflow Method

```
@Override
public void enroll(String customerId) {
    Logger logger = Workflow.getLogger("loyalty-" + customerId);
    while (true) {
        transactionsThisMonth = 0;
        Instant now = Instant.ofEpochMilli(Workflow.currentTimeMillis());
        Instant firstOfNextMonth = firstOfNextMonth();
        logger.info("Sleeping until {}", firstOfNextMonth);
        Workflow.sleep(Duration.between(now, firstOfNextMonth));
        if (transactionsThisMonth < 100) {
            tier = Tier.MEMBER;
        } else if (transactionsThisMonth < 500) {
            tier = Tier.SILVER;
        } else if (transactionsThisMonth < 1000) {
            tier = Tier.GOLD;
        } else {
            tier = Tier.PLATINUM;
        }
    }
}
```

Querying and Signaling

```
# Start the workflow specifying an ID
$ tctl workflow start --taskqueue loyalty-tq --workflow_type "LoyaltyWorkflow"  \
        --input "\"firdaus\"" --execution_timeout 315360000 \
        --workflow_id "customer-a"

# Record a transaction (signal)
$ tctl workflow signal --workflow_id "customer-a" --name "recordTransaction" --input "25"

# Query the transactions this month
$ tctl workflow query --workflow_id "customer-a" --query_type "transactionsThisMonth"

# Query the current tier
$ tctl workflow query --workflow_id "customer-a" --query_type "currentTier"
```

Other Features

- Cron workflows
- Child workflows
- Long running activities - heartbeat, activity completion
- Workflow history archival
- Untyped stubs (useful for DSLs or invoking activities / workflows across different languages)
- Workflow filtering / Search Attributes
- Namespaces
- Any many more……

Ecosystem

- Temporal Service (written in Golang)
- Database backends: Cassandra and MySQL (PostgreSQL support is beta)
- Search attributes support requires ElasticSearch and Kafka
- SDKs - Golang and Java are actively developed by Temporal Technologies. Python, Ruby and .NET have community contributed Cadence clients so it’s only a matter of time before they are ported to Temporal.
- temporal-web (http://localhost:8088)
- CLI - tctl
- Helm chart

Use Cases

- Microservice Orchestration
- Sharing economy use cases
- Subscriptions
- Loyalty / Gamification
- Marketing automation
- CI / CD Pipelines
- DSLs / Graphical Tools
- SAGA Pattern
- Improve code readability of business processes span long durations of time.

Going Deeper

- [Uber Open Summit 2018] Cadence: The Only Workflow Platform You'll Ever Need
  https://www.youtube.com/watch?v=llmsBGKOuWI

- Uber Cadence: Fault Tolerant Actor Framework
  https://www.youtube.com/watch?v=qce_AqCkFys

- Forum: https://community.temporal.io/

Thank You

Q&A