# Temporal

Масштабируемая отказоустойчивая платформа для оркестрации рабочих (бизнес) процессов

написанных с помощью кода или оркестратор бизнес процессов (WORKFLOW AS CODE).

Написан на golang и исповедует событийную архитектуру.

Взаимодействие между компонентами решения происходит по GRPC протоколу.

Сущности и их описание (в порядке от конечного действия к ядру системы):

**Activity** (действие, команда, активити)

Действия или бизнес логика выполняющая конкретную задачу.

- Не имеет статуса или состояния

- Все зависимости и интеграции здесь (блокировки).

**Worlflow** (рабочий процесс)

Отказоустойчивая программа, выполняющая действия, реагирующая на события, включая таймеры и таймауты. С возможностью логического ветвления.

- Имеет статус или состояние

- Должны быть идемпотентны.

- Нет зависимостей и интеграций (блокировок быть не должно)

- Могут порождать дочерние рабочие процессы

- Дочерние рабочие процессы могут быть сиротами.

Рекомендуемый паттерн от разработчиков:

- Рабочий процесс представляет собой один основной рабочий процесс (root).

- И по необходимости дополнительные рабочие процессы.

Типы рабочих процессов

- create - Основной «долгоживущий»рабочий процесс. Или рабочий процесс полного цикла.

Дополнительные, вспомогательные рабочие процессы.

- signal — Однонаправленный асинхронный без подтверждения.

- query — Получение каких-либо данных (Не фиксируется в событиях)

- update — Двунаправленный синхронный с подтверждением и получением результата.

Все р. п. Запускаются с помощью ручек по GRPC

**Task** (очередь задач)

Это собирательный образ.

Группировка рабочих процессов по некоторому строковому идентификатору.

**Worker** (worker, рабочий)

Регистрация рабочих процессов и команд. Контроль за их работой и выполнением.

**Преимущества**:

- Абстракция над бизнес логикой или отделение бизнес логики от сервисной частью.

- Контроль повторного выполнения (statefull execution).

- Обработка ошибок.

- Саги и распределённые транзакции реализуются легко (почти из коробки)

- Статусная машина (состояния). Нет гонок и нет синхронизаций

- Отложенные задачи и кроны.

- Всё вместе высокий ТТМ.

***«Time to market (TTM)»*** *— это* ***время от начала разработки идеи до конечного запуска решения и его выхода на рынок****.*

**Детали разработки:**

Рабочий процесс создаётся через фабрику и далее запускается его основной метод работы.

(Execute)

Высокочастотные действия:

Реализуются на стороне действия

Низкочастотные действия

Реализуются на стороне рабочего процесса

**Версионирование:**

При изменении кода рабочих процессов нужно использовать механизм версий !!!

И оборачивать все изменения в Side Effect

Версии работают на уровне блоков ( {} ) кода.

Линтер для проверки детерминированного кода (workflowcheck).

Replay Tests

В коде рабочих процессов использовать альтернативы следующим конструкциям:

time.Now() -> workflow.Now()  
time.Sleep() -> workflow.Sleep()  
logger -> workflow.GetLogger()  
go -> workflow.Go  
chan -> workflow.Channel  
select -> workflow.Selector  
context.Context -> workflow.Context

**Проблема статуса объекта в темпорал и во внешнем мире**:

Все статусы храним в темпорал.

Отправляем все статусы через кролик во внешний сервер статусов.

Соответственно тому кому нужен статус идёт на этот сервер.

Дополнительно решаем проблему избыточной нагрузки на темпорал.

**Сага или компенсирующие действия:**

Для реализации этого механизма для данной архитектуры программирования хорошо подходит конструкция defer