# Фоновые процессы - на чём реализовывать и почему мы выбрали Temporal

Пётр Сальников

СТО

- Больше **20 лет** опыта в IT
- **14+** лет на позиции **Team Lead / Software Architect**
- **6+** лет на позиции **Head of DevOps**
- Проходил путь **монолит – микросервисы - модульный монолит**
- Изучал **психоанализ** 6 лет

# Мои цели и о чём буду говорить

- Какие фоновые процессы бывают и как их правильно использовать
- Расскажу как Temporal помогает с фоновым выполнением логики
- Покажу применение Temporal в поле.рф

# Где и когда нам нужны фоновые процессы и джобы?

1. Выполнение действий по расписанию
2. Batch обработка в длительном процессе
3. Отложенная обработка данных
4. Асинхронное выполнение кода

## Какие частые проблемы встречаются

- Отсутствие лимитирования числа параллельных процессов
- Лавина нагрузки при одновременном запуске множества джобов
- Сложность в траблшутинге (логи, метрики, повторение неуспешных джобов)

## Какие частые проблемы встречаются

- Сложность оркестрации/взаимодействия при распределённой обработке
- Неправильное использование шедулера там, где правильнее будет работать в рамках Workflow

# Обзор самых частых способов реализации

- Coroutines / Threads
- Cron из Spring / Micronaut / <Your Favorite Framework>
- Quartz / JobRunr
- Kubernetes Jobs
- Temporal

# Как было и стало у нас

- Исторически использовали Kubernetes Jobs
- Сейчас перешли на Temporal

# Что такое Temporal?

- Детали в моём выступлении на JPoint 2024
- https://jpoint.ru/talks/784ea8f1cd2349e9ab9139157471a34d/
- https://youtu.be/uttzd6pod3Q

поле.рф

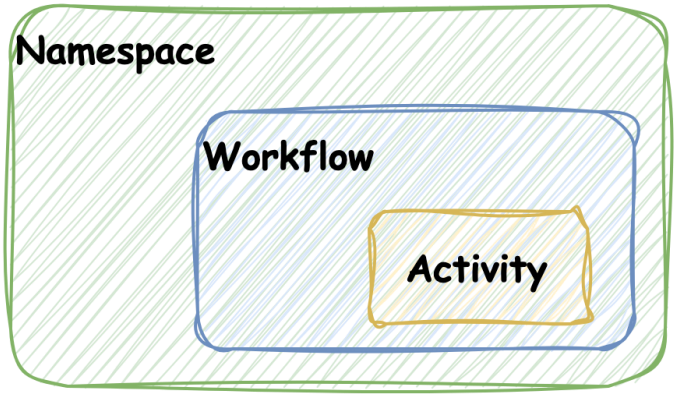
# Что такое Temporal?

- Оркестратор кода
- Бизнес процесс описывает кодом – Workflow
- Работает на многих языках (Java, GO, TS, PHP, Python, .NET)
- Отказоустойчивость через Retry, Save Point, Rate Limit
- Фоновое выполнение процессов
- Запуск по расписанию или с интервалом

# Ключевые понятия в Temporal

**Workflow** - гибкая программа, которая выполняет задачи, реагирует на внешние события, включая таймеры и таймауты

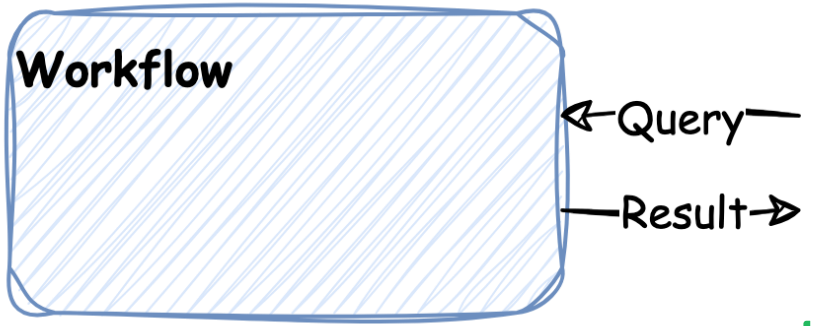
- **Namespace**
- **Workflow**
- **Activity**



# Ключевые понятия в Temporal

**Workflow** - гибкая программа, которая выполняет задачи, реагирует на внешние события, включая таймеры и таймауты

- **Namespace**
- **Workflow**
- **Activity**
- **Signal**
- **Query**

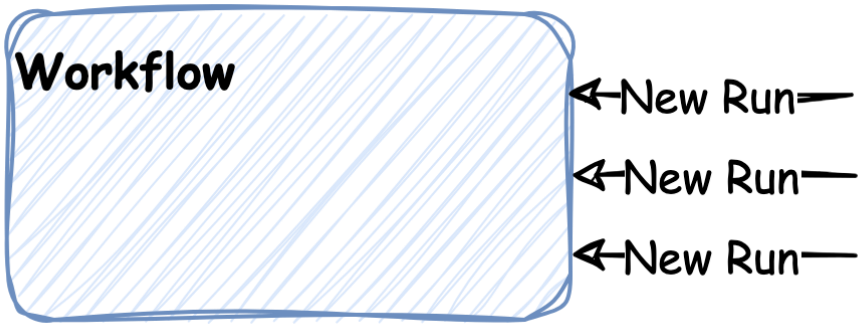




поле.рф

# Ключевые понятия в Temporal

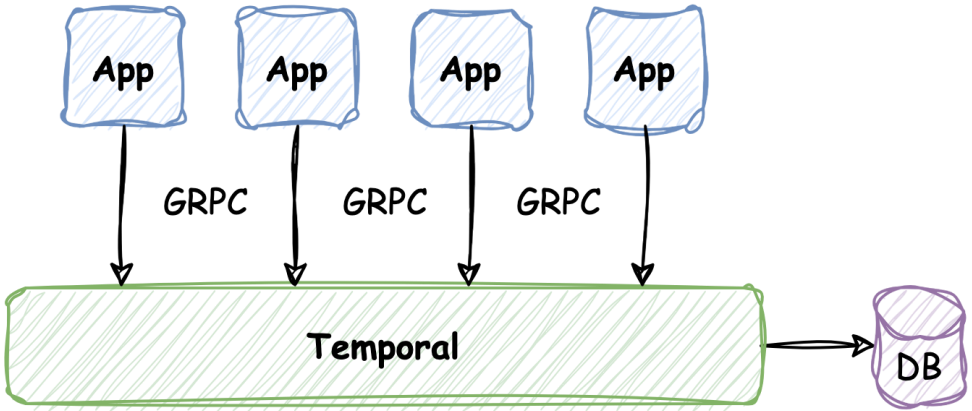
**Workflow** - гибкая программа, которая выполняет задачи, реагирует на внешние события, включая таймеры и таймауты

- **Namespace**
- **Workflow**
- **Activity**
- **Signal**
- **Query**
- **Timer**



# Развёртывание

- Helm для Kubernetes
- Docker Compose как пример для Standalone
- На уровне сети: App –> Temporal <– App




поле.рф

# Как реализовать фоновые процессы и джобы?

1. Выполнение действий по расписанию
2. Batch обработка в длительном процессе
3. Отложенная обработка данных
4. Асинхронное выполнение кода

# Выполнение действий по расписанию - Temporal Schedules

```java
// Что запускаем по расписанию
ScheduleActionStartWorkflow action =
    ScheduleActionStartWorkflow.newBuilder()
        .setWorkflowType(GreetingWorkflow.class)
        .setArguments("World")
        .setOptions(workflowOptions)
        .build();

// Объект расписание - сохранется в БД Temporal
Schedule schedule =
    Schedule.newBuilder()
      // Что именно делаем по расписанию
      .setAction(action)

      // Политика параллельных запусков
      .setPolicies(SchedulePolicies.newBuilder()
        .setOverlapPolicy(ScheduleOverlapPolicy.SCHEDULE_OVERLAP_POLICY_TERMINATE_OTHER)
      )

      // Когда запускать - расписание + интервалы
      .setSpec(
          ScheduleSpec.newBuilder()
              // Run the schedule at 5pm on Friday
              .setCalendars(
                  Collections.singletonList(
                      ScheduleCalendarSpec.newBuilder()
                          .setHour(Collections.singletonList(new ScheduleRange(17)))
                          .setDayOfWeek(Collections.singletonList(new ScheduleRange(5)))
                          .build()))
              // Run the schedule every 5s
              .setIntervals(
                  Collections.singletonList(new ScheduleIntervalSpec(Duration.ofSeconds(5))))
              .build()).build();

// Сохраняем расписание на сервер
ScheduleHandle handle =
    scheduleClient.createSchedule("ScheduleId", schedule, ScheduleOptions.newBuilder().build());
```

# Технически – такой же Workflow

```java
// Что запускаем по расписанию
ScheduleActionStartWorkflow action =
    ScheduleActionStartWorkflow.newBuilder()
        .setWorkflowType(GreetingWorkflow.class)
        .setArguments("World")
        .setOptions(workflowOptions)
        .build();

// Объект расписание - сохранется в БД Temporal
Schedule schedule =
    Schedule.newBuilder()
      // Что именно делаем по расписанию
      .setAction(action)

      // Политика параллельных запусков
      .setPolicies(SchedulePolicies.newBuilder()
        .setOverlapPolicy(ScheduleOverlapPolicy.SCHEDULE_OVERLAP_POLICY_TERMINATE_OTHER)
      )

      // Когда запускать - расписание + интервалы
      .setSpec(
          ScheduleSpec.newBuilder()
              // Run the schedule at 5pm on Friday
              .setCalendars(
                  Collections.singletonList(
                      ScheduleCalendarSpec.newBuilder()
                          .setHour(Collections.singletonList(new ScheduleRange(17)))
                          .setDayOfWeek(Collections.singletonList(new ScheduleRange(5)))
                          .build()))
              // Run the schedule every 5s
              .setIntervals(
                  Collections.singletonList(new ScheduleIntervalSpec(Duration.ofSeconds(5))))
              .build()).build();

// Сохраняем расписание на сервер
ScheduleHandle handle =
    scheduleClient.createSchedule("ScheduleId", schedule, ScheduleOptions.newBuilder().build());
```

Расписание хранится в БД Temporal

Технически – такой же Workflow

```java
// Что запускаем по расписанию
ScheduleActionStartWorkflow action =
    ScheduleActionStartWorkflow.newBuilder()
        .setWorkflowType(GreetingWorkflow.class)
        .setArguments("World")
        .setOptions(workflowOptions)
        .build();

// Объект расписание – сохранется в БД Temporal
Schedule schedule =
    Schedule.newBuilder()
      // Что именно делаем по расписанию
      .setAction(action)

      // Политика параллельных запусков
      .setPolicies(SchedulePolicies.newBuilder()
        .setOverlapPolicy(ScheduleOverlapPolicy.SCHEDULE_OVERLAP_POLICY_TERMINATE_OTHER)
      )

      // Когда запускать – расписание + интервалы
      .setSpec(
          ScheduleSpec.newBuilder()
              // Run the schedule at 5pm on Friday
              .setCalendars(
                  Collections.singletonList(
                      ScheduleCalendarSpec.newBuilder()
                          .setHour(Collections.singletonList(new ScheduleRange(17)))
                          .setDayOfWeek(Collections.singletonList(new ScheduleRange(5)))
                          .build()))
              // Run the schedule every 5s
              .setIntervals(
                  Collections.singletonList(new ScheduleIntervalSpec(Duration.ofSeconds(5))))
              .build()).build();

// Сохраняем расписание на сервер
ScheduleHandle handle =
    scheduleClient.createSchedule("ScheduleId", schedule, ScheduleOptions.newBuilder().build());
```

Настраиваем: руками через cli и ui  или flyway java migrations

Расписание хранится в БД Temporal

Технически – такой же Workflow

```java
// Что запускаем по расписанию
ScheduleActionStartWorkflow action =
    ScheduleActionStartWorkflow.newBuilder()
        .setWorkflowType(GreetingWorkflow.class)
        .setArguments("World")
        .setOptions(workflowOptions)
        .build();

// Объект расписание - сохранется в БД Temporal
Schedule schedule =
    Schedule.newBuilder()
      // Что именно делаем по расписанию
      .setAction(action)

      // Политика параллельных запусков
      .setPolicies(SchedulePolicies.newBuilder()
        .setOverlapPolicy(ScheduleOverlapPolicy.SCHEDULE_OVERLAP_POLICY_TERMINATE_OTHER)
      )

      // Когда запускать - расписание + интервалы
      .setSpec(
          ScheduleSpec.newBuilder()
              // Run the schedule at 5pm on Friday
              .setCalendars(
                  Collections.singletonList(
                      ScheduleCalendarSpec.newBuilder()
                          .setHour(Collections.singletonList(new ScheduleRange(17)))
                          .setDayOfWeek(Collections.singletonList(new ScheduleRange(5)))
                          .build()))
              // Run the schedule every 5s
              .setIntervals(
                  Collections.singletonList(new ScheduleIntervalSpec(Duration.ofSeconds(5))))
              .build()).build();

// Сохраняем расписание на сервер
ScheduleHandle handle =
    scheduleClient.createSchedule("ScheduleId", schedule, ScheduleOptions.newBuilder().build());
```

Можно задать расписание или интервал запуска

Настраиваем: руками через cli и ui  или flyway java migrations

Расписание хранится в БД Temporal

```java
// Что запускаем по расписанию
ScheduleActionStartWorkflow action =
    ScheduleActionStartWorkflow.newBuilder()
        .setWorkflowType(GreetingWorkflow.class)
        .setArguments("World")
        .setOptions(workflowOptions)
        .build();

// Объект расписание - сохранется в БД Temporal
Schedule schedule =
    Schedule.newBuilder()
      // Что именно делаем по расписанию
      .setAction(action)

      // Политика параллельных запусков
      .setPolicies(SchedulePolicies.newBuilder()
        .setOverlapPolicy(ScheduleOverlapPolicy.SCHEDULE_OVERLAP_POLICY_TERMINATE_OTHER)
      )

      // Когда запускать - расписание + интервалы
      .setSpec(
          ScheduleSpec.newBuilder()
              // Run the schedule at 5pm on Friday
              .setCalendars(
                  Collections.singletonList(
                      ScheduleCalendarSpec.newBuilder()
                          .setHour(Collections.singletonList(new ScheduleRange(17)))
                          .setDayOfWeek(Collections.singletonList(new ScheduleRange(5)))
                          .build()))
              // Run the schedule every 5s
              .setIntervals(
                  Collections.singletonList(new ScheduleIntervalSpec(Duration.ofSeconds(5))))
              .build()).build();

// Сохраняем расписание на сервер
ScheduleHandle handle =
    scheduleClient.createSchedule("ScheduleId", schedule, ScheduleOptions.newBuilder().build());
```

Temporal помогает с параллелизмом и лимитами

Можно задать расписание или интервал запуска

Настраиваем: руками через cli и ui  или flyway java migrations

# Фоновая обработка в длительном процессе - Batch

```
IteratorBatchWorkflow batchWorkflowStub = workflowClient.newWorkflowStub(
            IteratorBatchWorkflow.class,
            WorkflowOptions.newBuilder().setTaskQueue(Worker.TASK_QUEUE_NAME).build());

WorkflowClient.start(batchWorkflowStub::processBatch, config.pageSize, 0)
```

```
public int processBatch(int pageSize, int offset) {
    // Получаем очередной список сущностей для обработки
    List<SingleRecord> records = recordLoader.getRecords(pageSize, offset);

    // На каждую сущность стартуем Child Workflow
    // в котором будет происходить обработка
    List<Promise<Void>> results = new ArrayList<>(records.size());
    for (SingleRecord record : records) {
      // Uses human friendly child id.
      String childId =
        Workflow.getInfo().getWorkflowId() + "/" + record.getId();
      RecordProcessorWorkflow processor =
          Workflow.newChildWorkflowStub(
              RecordProcessorWorkflow.class,
              ChildWorkflowOptions.newBuilder().setWorkflowId(childId).build());
      Promise<Void> result = Async.procedure(processor::processRecord, record);
      results.add(result);
    }

    // Ждём когда все запущенные в текущем батче Child Workflow завершатся
    Promise.allOf(results).get();

    // Если больше записей для обработки нет - завершаем текущий Workflow
    if (records.isEmpty()) {
      return offset;
    }

    // Запускаем Workflow для следующей очереди записей
    return nextRun.processBatch(pageSize, offset + records.size());
}
```

В параллель запускаем Activity или Child Workflow

```
IteratorBatchWorkflow batchWorkflowStub = workflowClient.newWorkflowStub(
            IteratorBatchWorkflow.class,
            WorkflowOptions.newBuilder().setTaskQueue(Worker.TASK_QUEUE_NAME).build());

WorkflowClient.start(batchWorkflowStub::processBatch, config.pageSize, 0)
```

```
public int processBatch(int pageSize, int offset) {
    // Получаем очередной список сущностей для обработки
    List<SingleRecord> records = recordLoader.getRecords(pageSize, offset);

    // На каждую сущность стартуем Child Workflow
    // в котором будет происходить обработка
    List<Promise<Void>> results = new ArrayList<>(records.size());
    for (SingleRecord record : records) {
      // Uses human friendly child id.
      String childId =
        Workflow.getInfo().getWorkflowId() + "/" + record.getId();
      RecordProcessorWorkflow processor =
          Workflow.newChildWorkflowStub(
              RecordProcessorWorkflow.class,
              ChildWorkflowOptions.newBuilder().setWorkflowId(childId).build());
      Promise<Void> result = Async.procedure(processor::processRecord, record);
      results.add(result);
    }

    // Ждём когда все запущенные в текущем батче Child Workflow завершатся
    Promise.allOf(results).get();

    // Если больше записей для обработки нет — завершаем текущий Workflow
    if (records.isEmpty()) {
      return offset;
    }

    // Запускаем Workflow для следующей очереди записей
    return nextRun.processBatch(pageSize, offset + records.size());
}
```

Легко определять нагрузку и скорость обработки

В параллель запускаем Activity или Child Workflow

```
IteratorBatchWorkflow batchWorkflowStub = workflowClient.newWorkflowStub(
            IteratorBatchWorkflow.class,
            WorkflowOptions.newBuilder().setTaskQueue(Worker.TASK_QUEUE_NAME).build());

WorkflowClient.start(batchWorkflowStub::processBatch, config.pageSize, 0)
```

```
public int processBatch(int pageSize, int offset) {
    // Получаем очередной список сущностей для обработки
    List<SingleRecord> records = recordLoader.getRecords(pageSize, offset);

    // На каждую сущность стартуем Child Workflow
    // в котором будет происходить обработка
    List<Promise<Void>> results = new ArrayList<>(records.size());
    for (SingleRecord record : records) {
      // Uses human friendly child id.
      String childId =
        Workflow.getInfo().getWorkflowId() + "/" + record.getId();
      RecordProcessorWorkflow processor =
          Workflow.newChildWorkflowStub(
              RecordProcessorWorkflow.class,
              ChildWorkflowOptions.newBuilder().setWorkflowId(childId).build());
      Promise<Void> result = Async.procedure(processor::processRecord, record);
      results.add(result);
    }

    // Ждём когда все запущенные в текущем батче Child Workflow завершатся
    Promise.allOf(results).get();

    // Если больше записей для обработки нет — завершаем текущий Workflow
    if (records.isEmpty()) {
      return offset;
    }

    // Запускаем Workflow для следующей очереди записей
    return nextRun.processBatch(pageSize, offset + records.size());
}
```

Запускающий и выполняющие процессы можно разнести

Легко определять нагрузку и скорость обработки

# Отложенная обработка данных

```java
NotificationsWorkflow notifications = workflowClient.newWorkflowStub(
    NotificationsWorkflow.class,
    WorkflowOptions.newBuilder().setTaskQueue(NS_NOTIFICATIONS).build());

// Отправлем смс
WorkflowClient.start(notifications::sendSms, phone,
    // Параметры для шаблона смс
    Map.of(
        "amount", "1234.00",
        "reason", "списание"
    )
)

// Отправляем EMail с приложениями
WorkflowClient.start(notifications::sendEmail, emailTo,
    // Параметры для шаблона email
    Map.of(
        "amount", "1234.00",
        "reason", "списание",
    ),
    // Ссылки на приложения
    Map.of(
        "summary.pdf", "s3-signed-url",
        "details.pdf", "s3-signed-url",
    ),
)
```

Технически – Триггер + Workflow в фоне

```java
NotificationsWorkflow notifications = workflowClient.newWorkflowStub(
    NotificationsWorkflow.class,
    WorkflowOptions.newBuilder().setTaskQueue(NS_NOTIFICATIONS).build());

// Отправлем смс
WorkflowClient.start(notifications::sendSms, phone,
    // Параметры для шаблона смс
    Map.of(
        "amount", "1234.00",
        "reason", "списание"
    )
)

// Отправляем EMail с приложениями
WorkflowClient.start(notifications::sendEmail, emailTo,
    // Параметры для шаблона email
    Map.of(
        "amount", "1234.00",
        "reason", "списание",
    ),
    // Ссылки на приложения
    Map.of(
        "summary.pdf", "s3-signed-url",
        "details.pdf", "s3-signed-url",
    ),
)
```

Легко контролировать нагрузку и скорость настройками

Технически – Триггер + Workflow в фоне

```java
NotificationsWorkflow notifications = workflowClient.newWorkflowStub(
    NotificationsWorkflow.class,
    WorkflowOptions.newBuilder().setTaskQueue(NS_NOTIFICATIONS).build());

// Отправлем смс
WorkflowClient.start(notifications::sendSms, phone,
    // Параметры для шаблона смс
    Map.of(
        "amount", "1234.00",
        "reason", "списание"
    )
)

// Отправляем EMail с приложениями
WorkflowClient.start(notifications::sendEmail, emailTo,
    // Параметры для шаблона email
    Map.of(
        "amount", "1234.00",
        "reason", "списание",
    ),
    // Ссылки на приложения
    Map.of(
        "summary.pdf", "s3-signed-url",
        "details.pdf", "s3-signed-url",
    ),
)
```

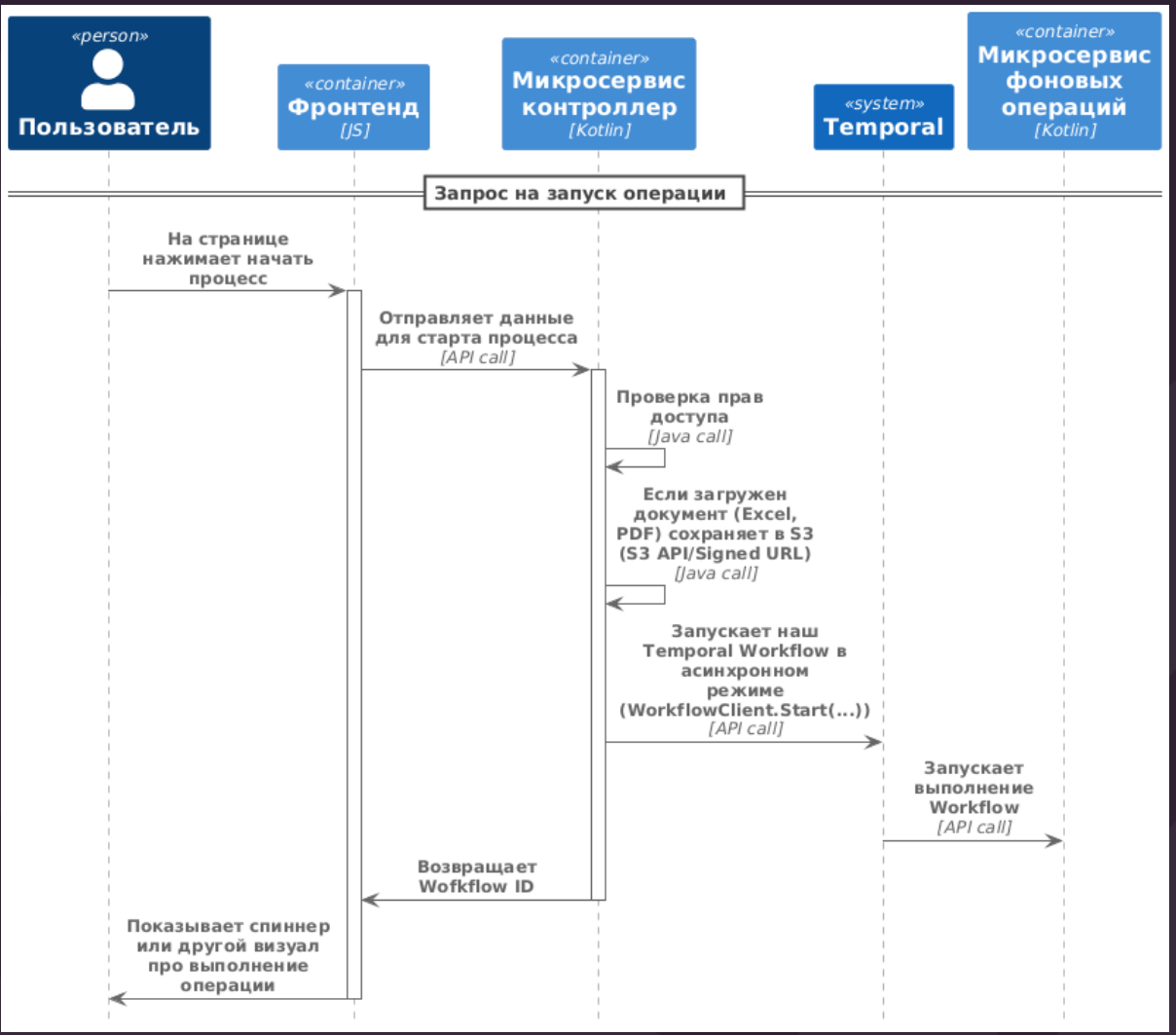
Простой аналог очереди из Kafka/Rabbit

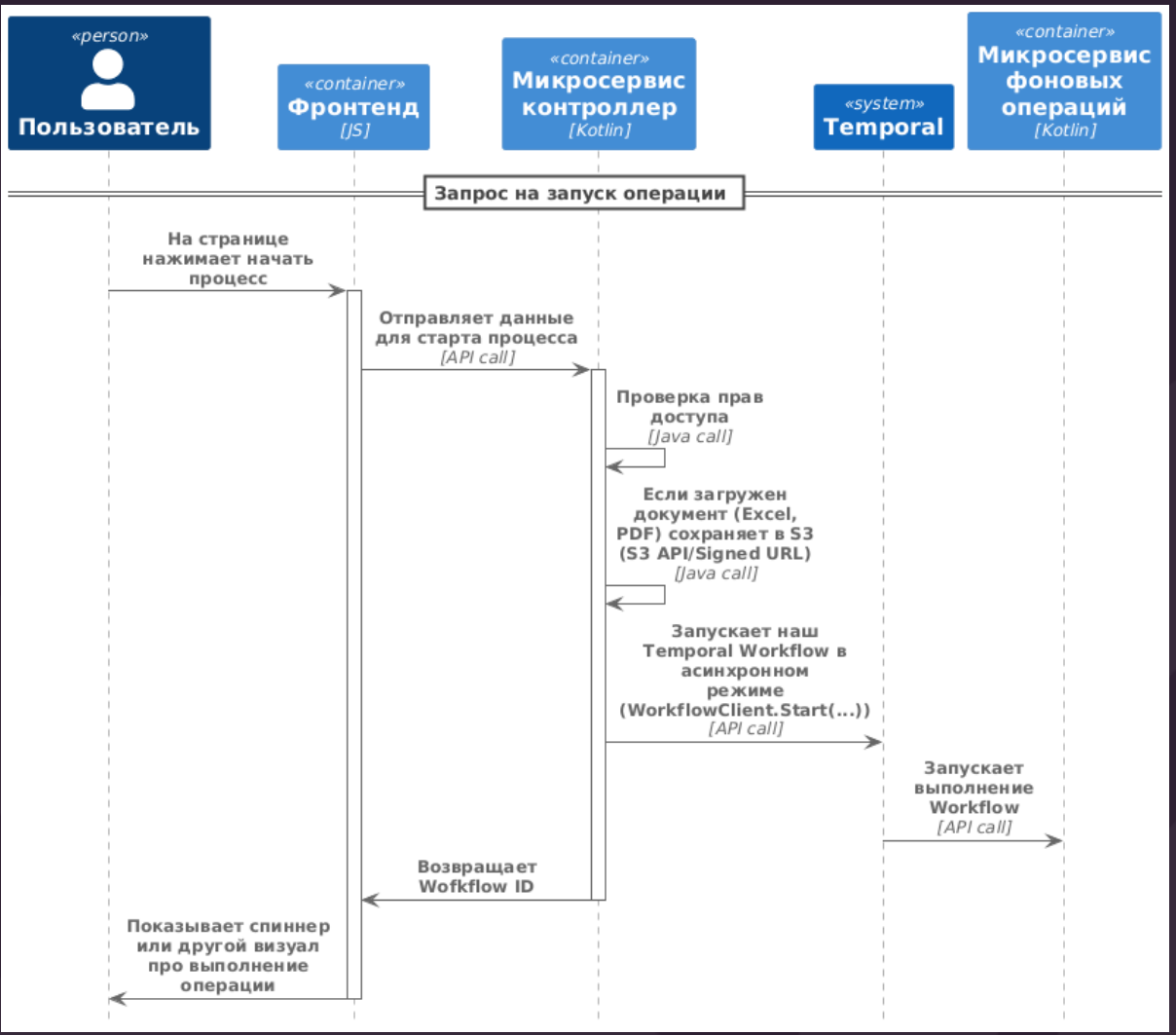
Легко контролировать нагрузку и скорость настройками

Технически – Триггер + Workflow в фоне

# Асинхронное выполнение кода

# Технически – API Endpoint + Workflow

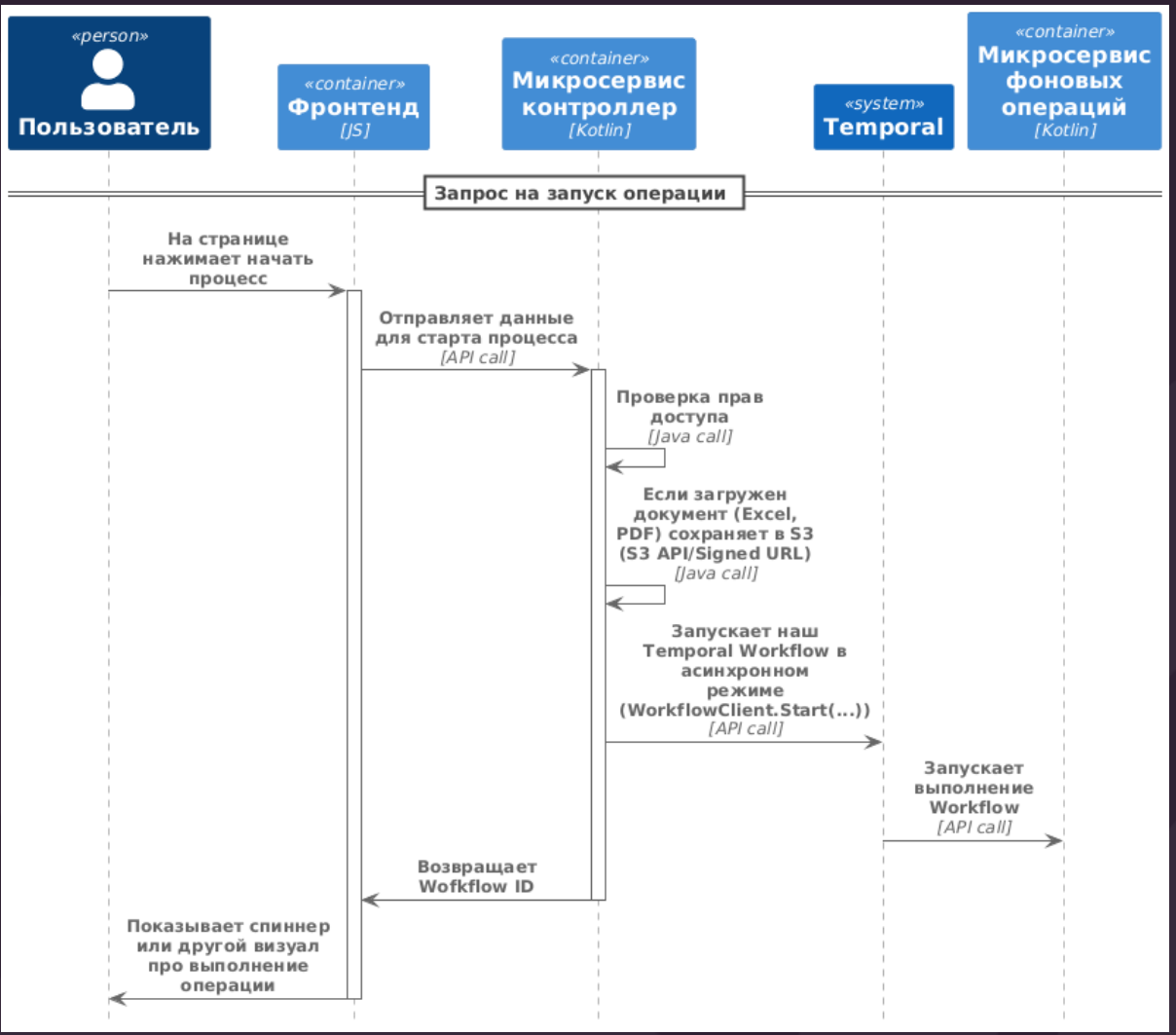





Можно контролировать нагрузку/скорость

Технически – API Endpoint + Workflow

поле.рф

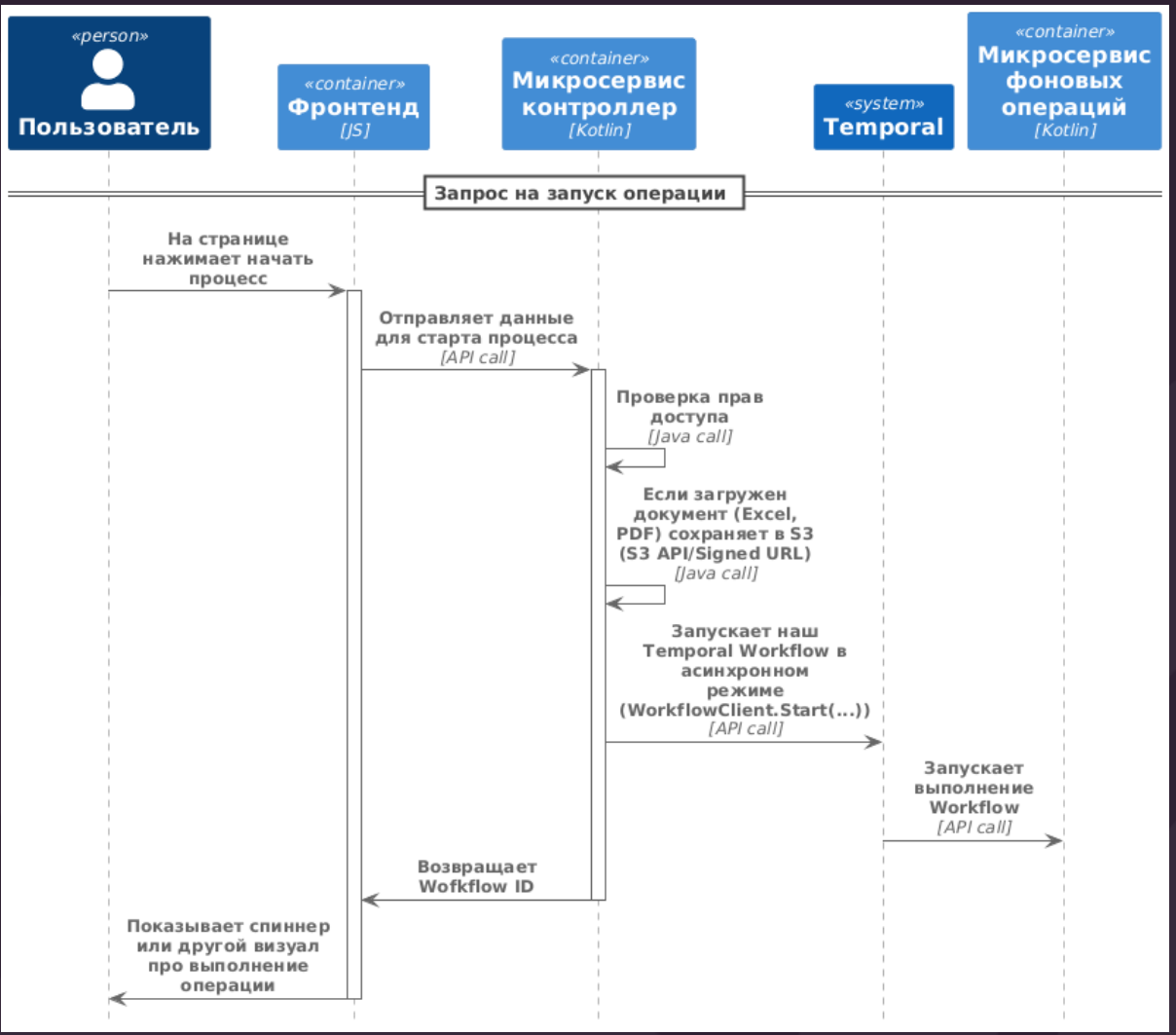



Запускающий и выполняющие процессы можно разнести

Можно контролировать нагрузку/скорость

Технически – API Endpoint + Workflow

поле.рф



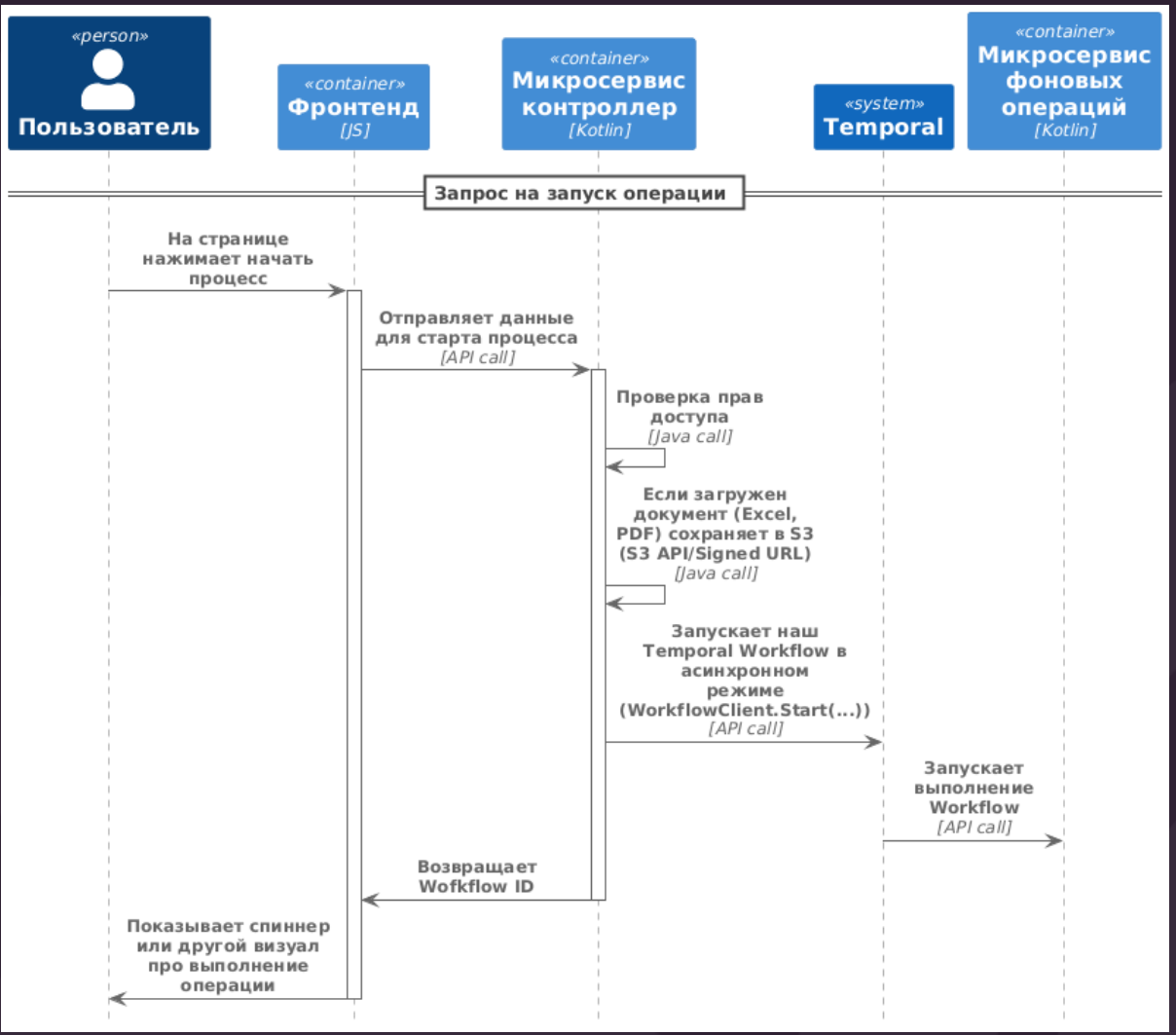

Поэтапное выполнение и SAGA

Запускающий и выполняющие процессы можно разнести

Можно контролировать нагрузку/скорость

Технически – API Endpoint + Workflow

поле.рф




Есть нюансы с большими данными (>256Kb)

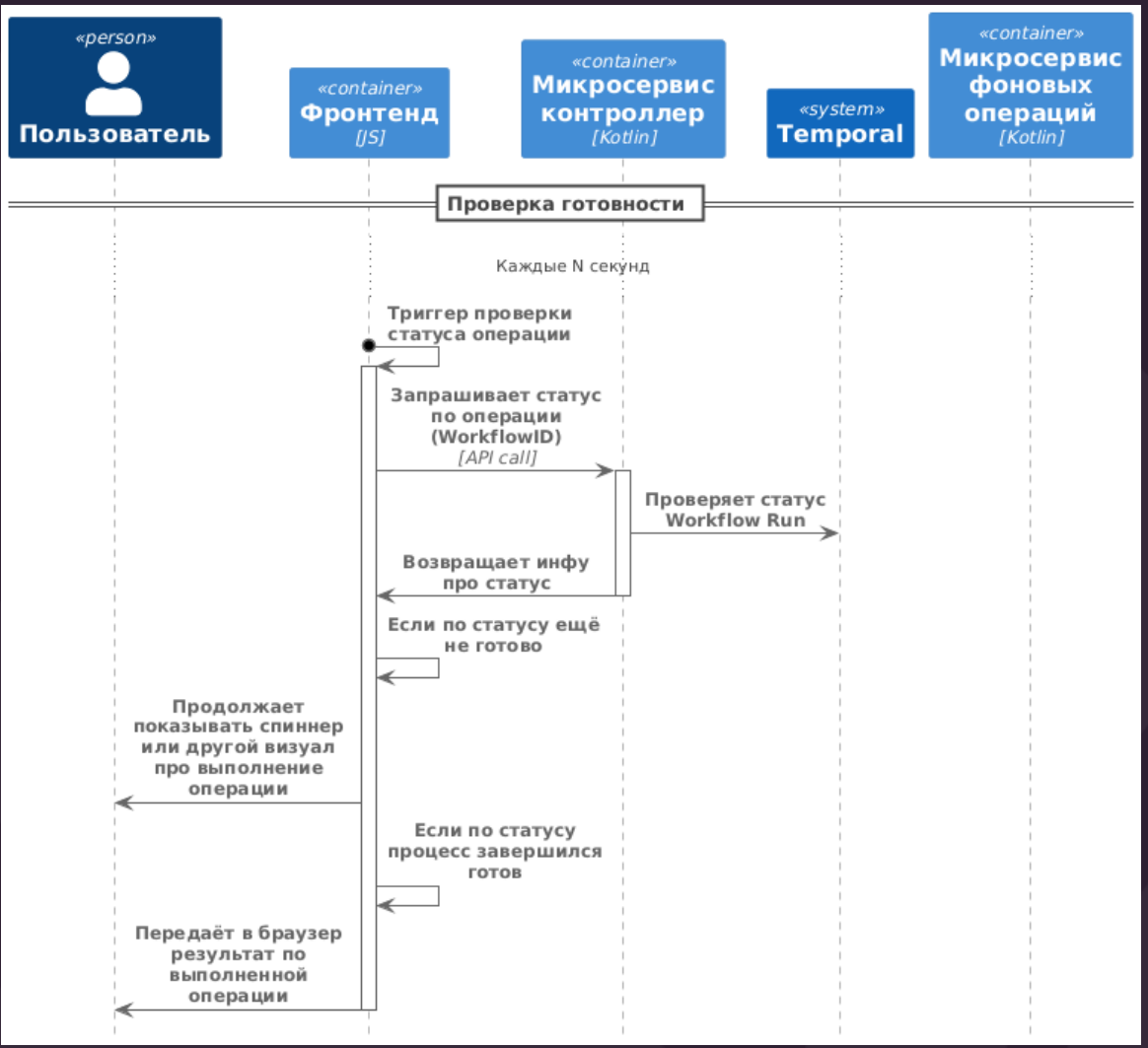
Поэтапное выполнение и SAGA

Запускающий и выполняющие процессы можно разнести

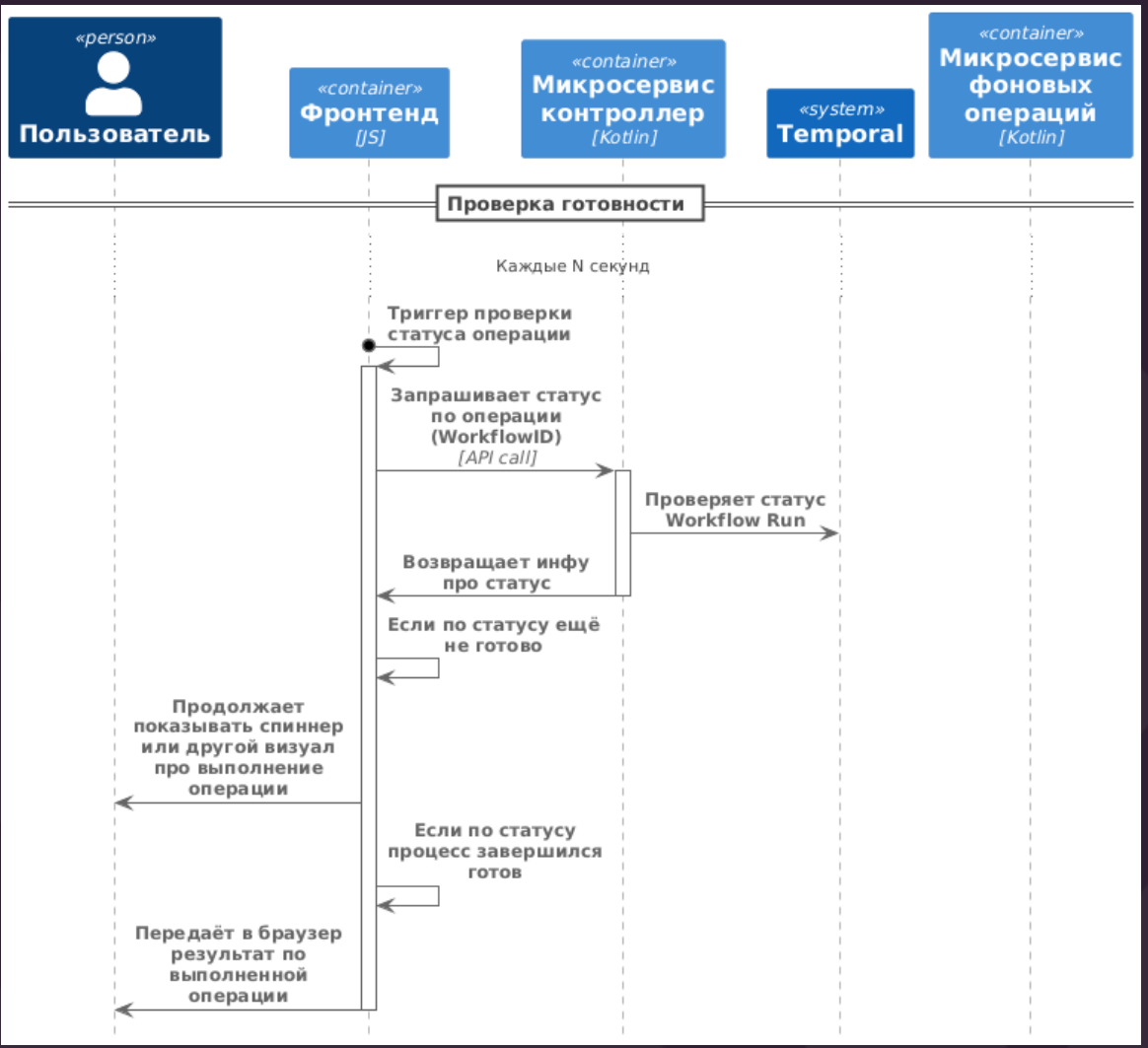
Можно контролировать нагрузку/скорость

поле.рф

# Важен подходящий UX



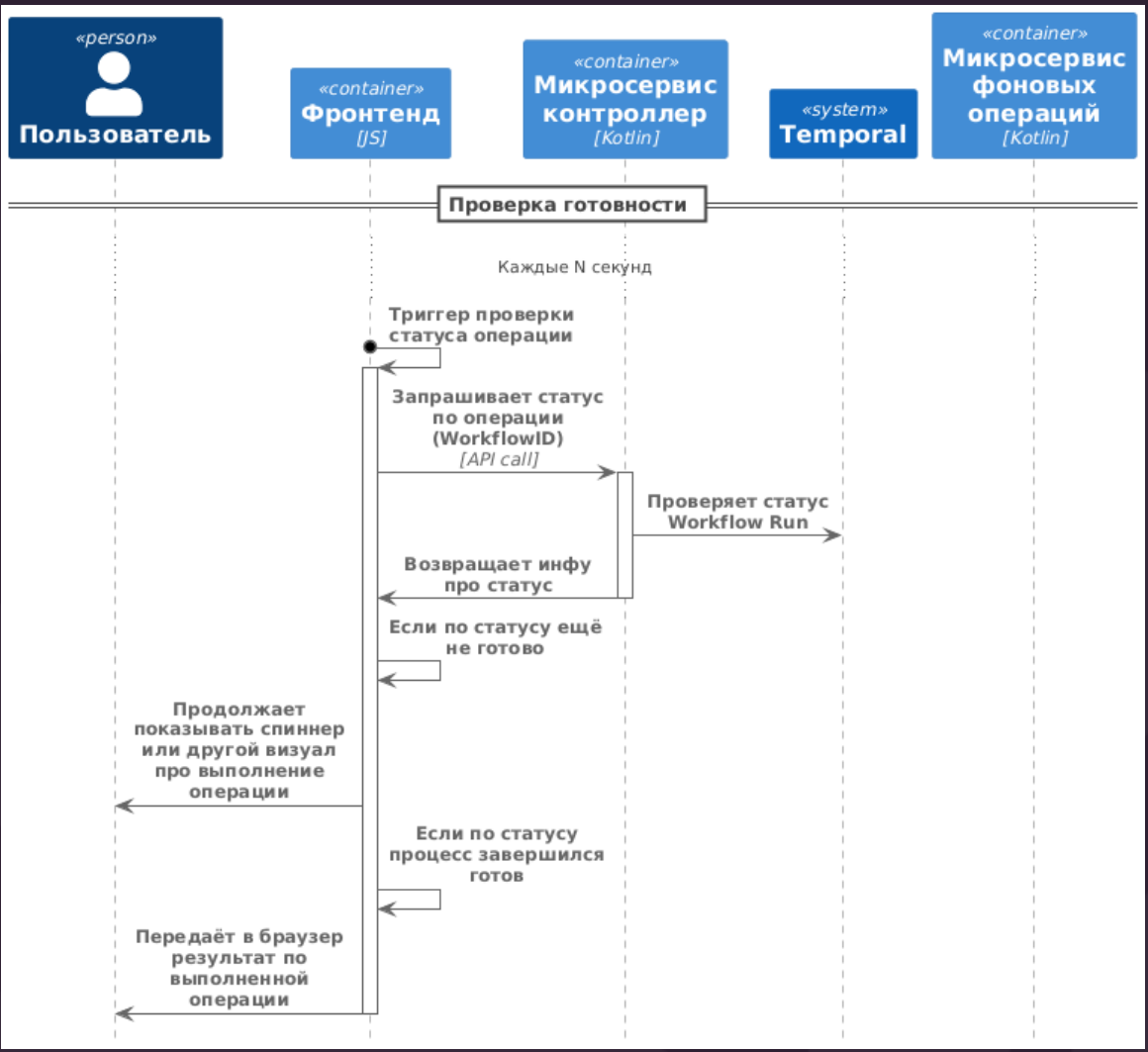

поле.рф




Можно контролировать нагрузку/скорость

Важен подходящий UX

поле.рф




К результатам фоновой операции пользователь может вернутся через любое время

Можно контролировать нагрузку/скорость

Важен подходящий UX

поле.рф

# Работа с файлами
# в асинхронных процессах

```
1   // Выбираем уникальное имя для TaskQueue
2   // Удобнее будет брать имя POD-а
3   String uniqueTaskQueueName = "someQueue_" + UUID.randomUUID().toString()
4
5   // Сохраняем имя пода внутри активити
6   // Это нужно чтобы первым активити привязать воркфлоу к определённому POD-у
7   StoreActivitiesImpl storeActivityImpl =
8       new StoreActivitiesImpl(uniqueTaskQueueName)
9
10  // Создаём объект Worker привязанный к текущему pod-у
11  // И регистрируем наш Activity в этом воркере
12  Worker workerLinkedToCurrentPod = workerFactory.newWorker(uniqueTaskQueueName);
13  workerLinkedToCurrentPod.registerActivitiesImplementations(storeActivityImpl);
14
15
16
17  // Внутри Workflow блок Sticky Activity нужно обернуть для общего ретрая
18  Workflow.retry(retryOptions, Optional.empty(), () -> processFileImpl(source, destination))
19
20
21
```

# Особенности Temporal – лимиты 256Kb / 2Mb

поле.рф

```
1   // Выбираем уникальное имя для TaskQueue
2   // Удобнее будет брать имя POD-а
3   String uniqueTaskQueueName = "someQueue_" + UUID.randomUUID().toString()
4
5   // Сохраняем имя пода внутри активити
6   // Это нужно чтобы первым активити привязать воркфлоу к определённому POD-у
7   StoreActivitiesImpl storeActivityImpl =
8       new StoreActivitiesImpl(uniqueTaskQueueName)
9
10  // Создаём объект Worker привязанный к текущему pod-у
11  // И регистрируем наш Activity в этом воркере
12  Worker workerLinkedToCurrentPod = workerFactory.newWorker(uniqueTaskQueueName);
13  workerLinkedToCurrentPod.registerActivitiesImplementations(storeActivityImpl);
14
15
16
17  // Внутри Workflow блок Sticky Activity нужно обернуть для общего ретрая
18  Workflow.retry(retryOptions, Optional.empty(), () -> processFileImpl(source, destination))
19
20
21
```

Поможет паттерн Sticky Activity

Особенности Temporal – лимиты 256Kb / 2Mb

поле.рф

```
// Выбираем уникальное имя для TaskQueue
// Удобнее будет брать имя POD-а
String uniqueTaskQueueName = "someQueue_" + UUID.randomUUID().toString()

// Сохраняем имя пода внутри активити
// Это нужно чтобы первым активити привязать воркфлоу к определённому POD-у
StoreActivitiesImpl storeActivityImpl =
    new StoreActivitiesImpl(uniqueTaskQueueName)

// Создаём объект Worker привязанный к текущему pod-у
// И регистрируем наш Activity в этом воркере
Worker workerLinkedToCurrentPod = workerFactory.newWorker(uniqueTaskQueueName);
workerLinkedToCurrentPod.registerActivitiesImplementations(storeActivityImpl);



// Внутри Workflow блок Sticky Activity нужно обернуть для общего ретрая
Workflow.retry(retryOptions, Optional.empty(), () -> processFileImpl(source, destination))
```

Особенности Temporal Task Queue

Поможет паттерн Sticky Activity

Особенности Temporal – лимиты 256Kb / 2Mb

поле.рф

```java
// Выбираем уникальное имя для TaskQueue
// Удобнее будет брать имя POD-а
String uniqueTaskQueueName = "someQueue_" + UUID.randomUUID().toString()

// Сохраняем имя пода внутри активити
// Это нужно чтобы первым активити привязать воркфлоу к определённому POD-у
StoreActivitiesImpl storeActivityImpl =
    new StoreActivitiesImpl(uniqueTaskQueueName)

// Создаём объект Worker привязанный к текущему pod-у
// И регистрируем наш Activity в этом воркере
Worker workerLinkedToCurrentPod = workerFactory.newWorker(uniqueTaskQueueName);
workerLinkedToCurrentPod.registerActivitiesImplementations(storeActivityImpl);



// Внутри Workflow блок Sticky Activity нужно обернуть для общего ретрая
Workflow.retry(retryOptions, Optional.empty(), () -> processFileImpl(source, destination))
```

Temporal Task Queue – связать с именем POD-а

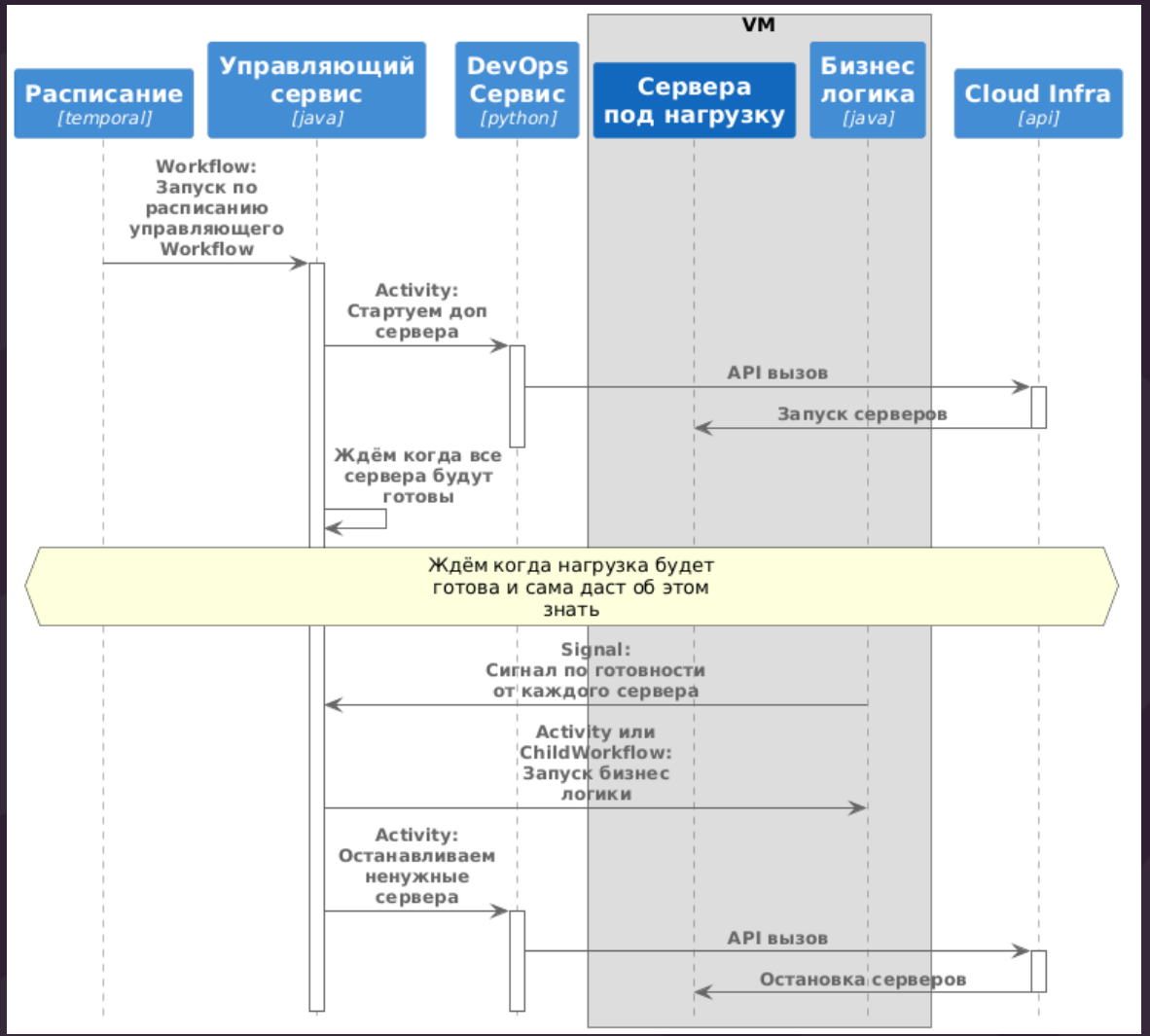
Особенности Temporal Task Queue

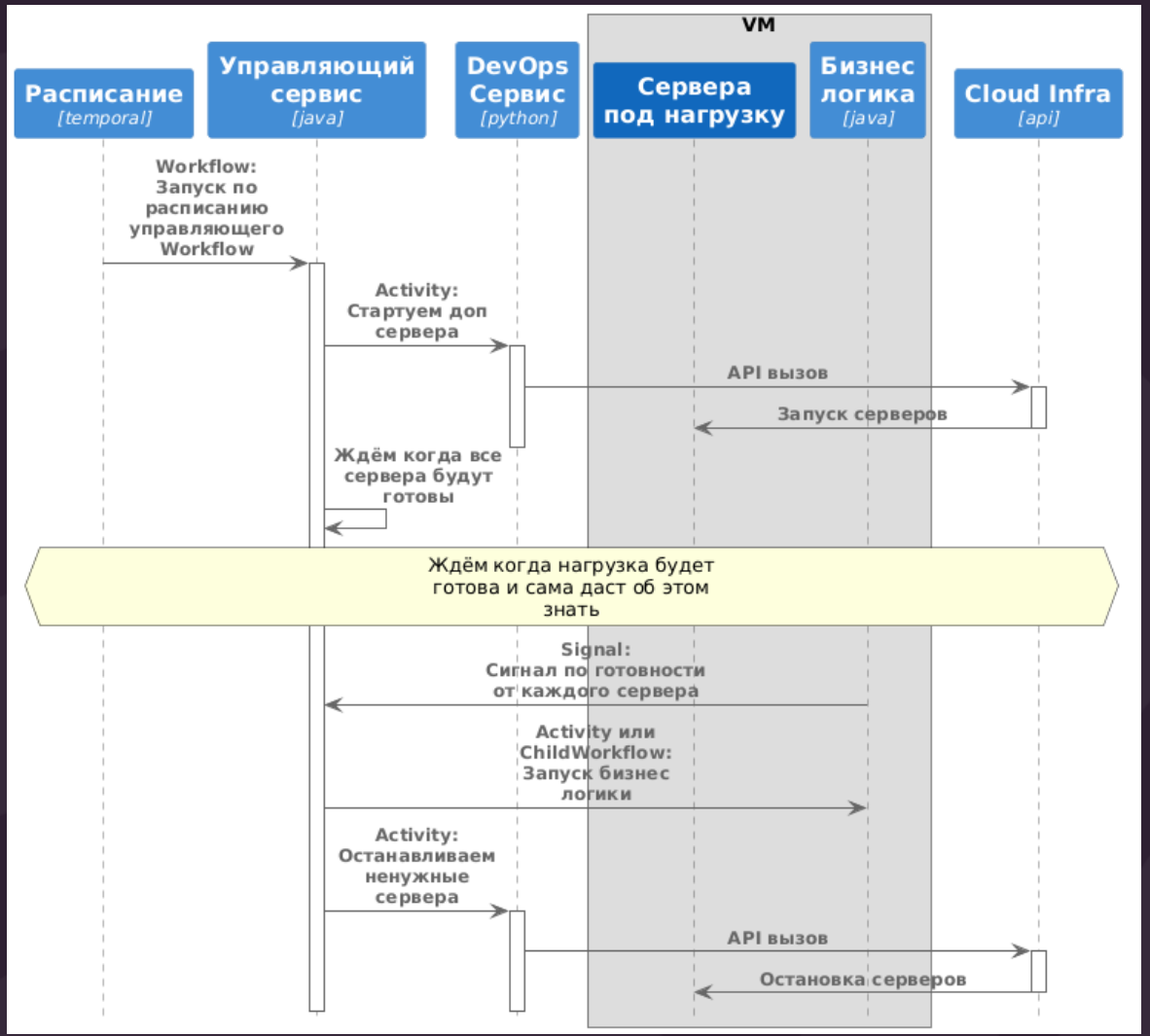
Поможет паттерн Sticky Activity

поле.рф

# Способы скейлинга нагрузки

# Допустима сложная логика скейлинга

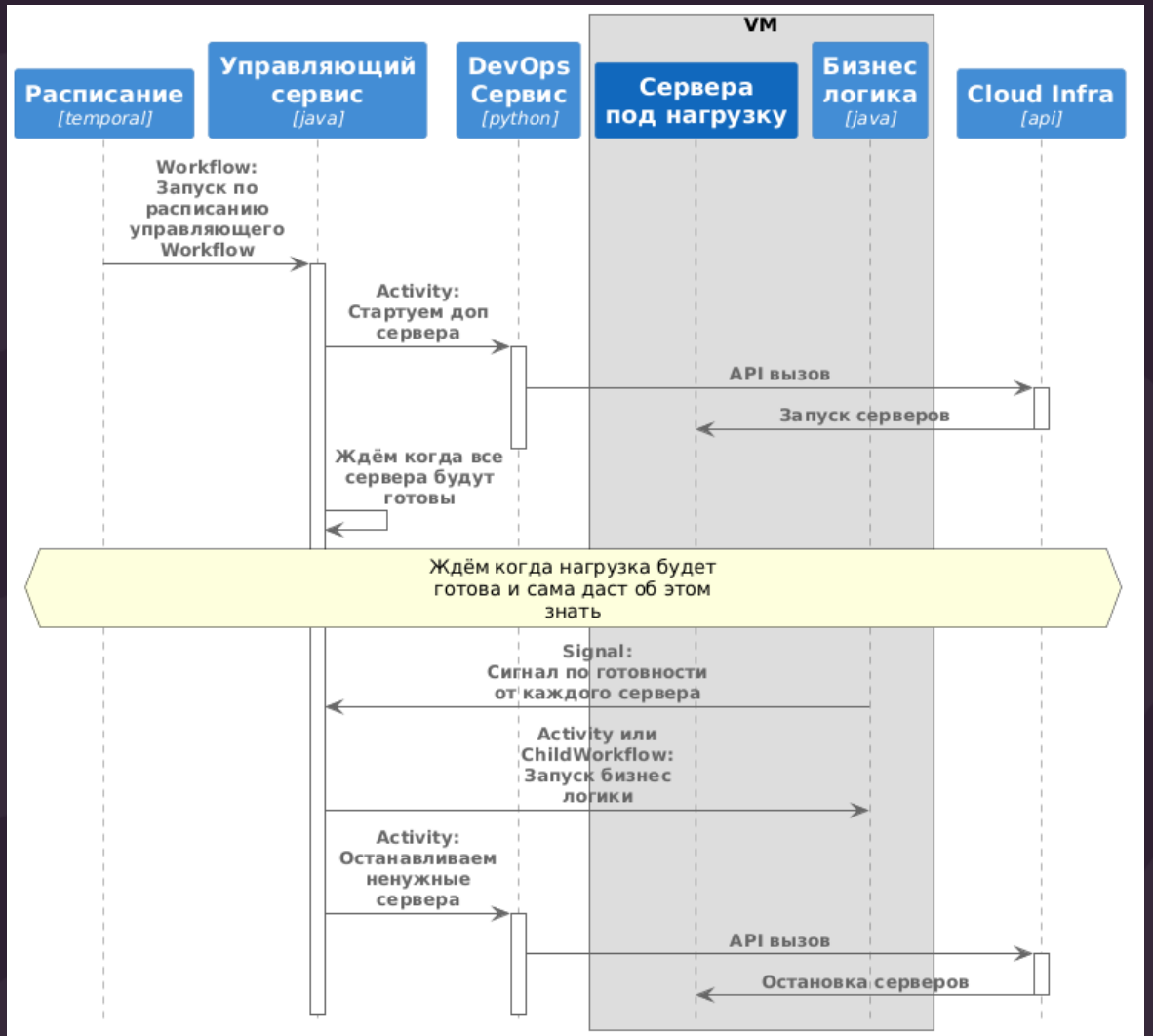





Валидно для GPU или других специфических нагрузок

Допустима сложная логика скейлинга

поле.рф

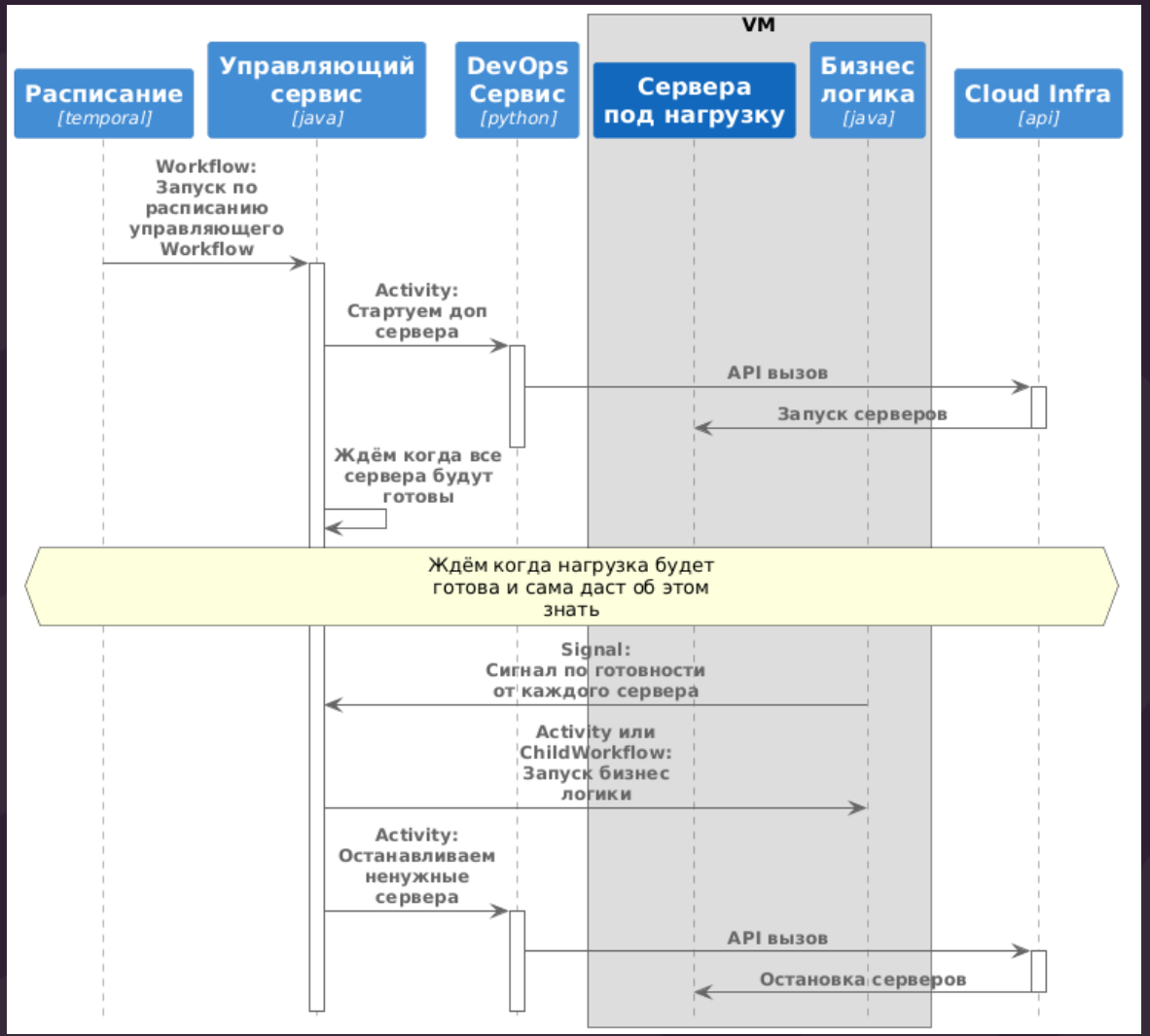



Скрипты инфраструктуры реализовать на Python, а логику на Java/Kotlin

Валидно для GPU или других специфических нагрузок

Допустима сложная логика скейлинга

поле.рф




Можно под расписание/планируемую нагрузку скейлить виртуальный машины

Скрипты инфраструктуры реализовать на Python, а логику на Java/Kotlin

Валидно для GPU или других специфических нагрузок

Допустима сложная логика с

поле.рф

# Когда использовать Schedule, а когда в рамках бизнес процесса/Workflow

**Schedule ->**

Когда есть явное расписание под нашу логику

**Workflow Activity =>**

Когда над сущностью нужны фоновые обработки при переходе между шагами процесса

# Способы реализации data backfilling и data migration

**Терминология**

- data migration - миграция данных, трансформация или перенос формата хранения при релизе.
- data backfilling - наполнение упущенных/пустых мест недостающими данными

# Способы реализации data backfilling и data migration

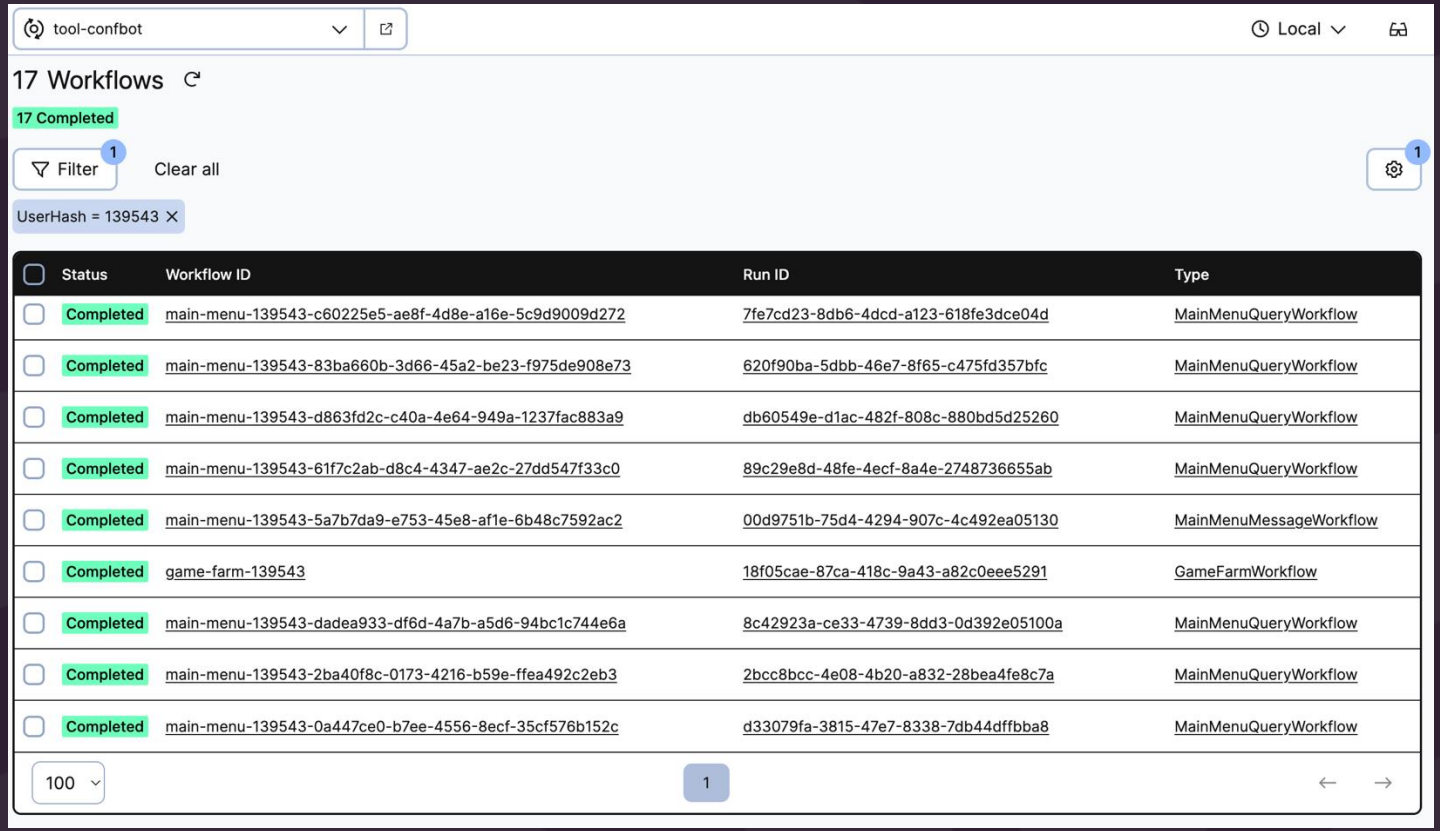
- Фоновое выполнение нужно, когда процесс миграции очень долгий и требует дополнительных вычислений
- Можно подготовить данные и в несколько шагов/релизов смигрировать
- А можно в выделенном сервисе запустить Workflow и мигрировать данные кодом в рамках фонового процесса

# Как делать Troubleshooting

- В логах при запуске обязательно оставляем WorkflowId
- На уровне кода Workflow обёртка для логгера
- Используем Custom Search Attributes – ID сущности или пользователя

- ПРИМЕР КОДА ПО ССЫЛКЕ

# Как делать Troubleshooting




| Status | Workflow ID | Run ID | Type |
|---|---|---|---|
| Completed | main-menu-139543-c60225e5-ae8f-4d8e-a16e-5c9d9009d272 | 7fe7cd23-8db6-4dcd-a123-618fe3dce04d | MainMenuQueryWorkflow |
| Completed | main-menu-139543-83ba660b-3d66-45a2-be23-f975de908e73 | 620f90ba-5dbb-46e7-8f65-c475fd357bfc | MainMenuQueryWorkflow |
| Completed | main-menu-139543-d863fd2c-c40a-4e64-949a-1237fac883a9 | db60549e-d1ac-482f-808c-880bd5d25260 | MainMenuQueryWorkflow |
| Completed | main-menu-139543-61f7c2ab-d8c4-4347-ae2c-27dd547f33c0 | 89c29e8d-48fe-4ecf-8a4e-2748736655ab | MainMenuQueryWorkflow |
| Completed | main-menu-139543-5a7b7da9-e753-45e8-af1e-6b48c7592ac2 | 00d9751b-75d4-4294-907c-4c492ea05130 | MainMenuMessageWorkflow |
| Completed | game-farm-139543 | 18f05cae-87ca-418c-9a43-a82c0eee5291 | GameFarmWorkflow |
| Completed | main-menu-139543-dadea933-df6d-4a7b-a5d6-94bc1c744e6a | 8c42923a-ce33-4739-8dd3-0d392e05100a | MainMenuQueryWorkflow |
| Completed | main-menu-139543-2ba40f8c-0173-4216-b59e-ffea492c2eb3 | 2bcc8bcc-4e08-4b20-a832-28bea4fe8c7a | MainMenuQueryWorkflow |
| Completed | main-menu-139543-0a447ce0-b7ee-4556-8ecf-35cf576b152c | d33079fa-3815-47e7-8338-7db44dffbba8 | MainMenuQueryWorkflow |

# Когда какой из вариантов применим и оправдан

- Spring Cron / Quartz - для простых проектов, как хороший старт
- Kubernetes Jobs - если DevOps практика сильная и реализация от них
- JobRunr - большое число джобов, выделенная технология исключительно под фоновые задачи
- Temporal - когда есть необходимость в воркфлоу оркестраторе или когда нужно уметь лимитировать нагрузку/параллельное выполнение

# Happy End

- Основные сценарии фонового выполнения логики
- Детали по каждому из них
- Нюансы вокруг передачи файлов и траблшутинга
- Исходный код тут: https://github.com/pole-rf/temporal-demo
- Вопросы можно мне: @PeterSalnikov
- Ждём в гости на стенде поле.рф

поле.рф

**Примеры**

**Задать вопросы**