

Beyond CRUD: the PostgreSQL techniques your Rails app is missing 🚀



Beyond CRUD: the PostgreSQL techniques your Rails app is missing



Shayon Mukherjee

 @shayonj





 @shayonj



Incidents Happen 🙈



Overview

- The time when we lost all dynamically scheduled customer jobs ☐
- Optimistic and Pessimistic Locking 
- Skip'em with SKIP LOCKED 
- Synchronizing workloads with Advisory Locks 
- Timeouts, Reconnects and Retries 



I am Shayon Mukherjee



@shayonj



@shayonj



Railsconf
ATLANTA 2023



Railsconf
ATLANTA 2023



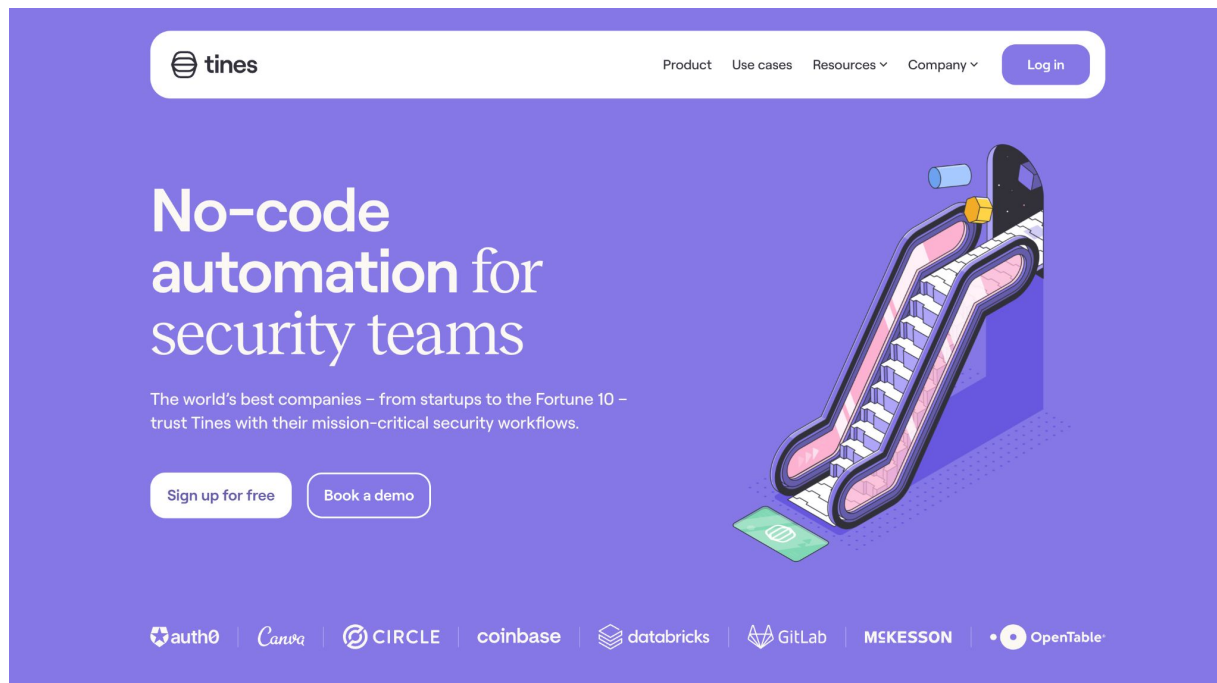
Olly (O11y?)

 Cambridge, MA

Beyond CRUD: the PostgreSQL techniques your Rails app is missing | Rails Conf 2023



www.tines.com

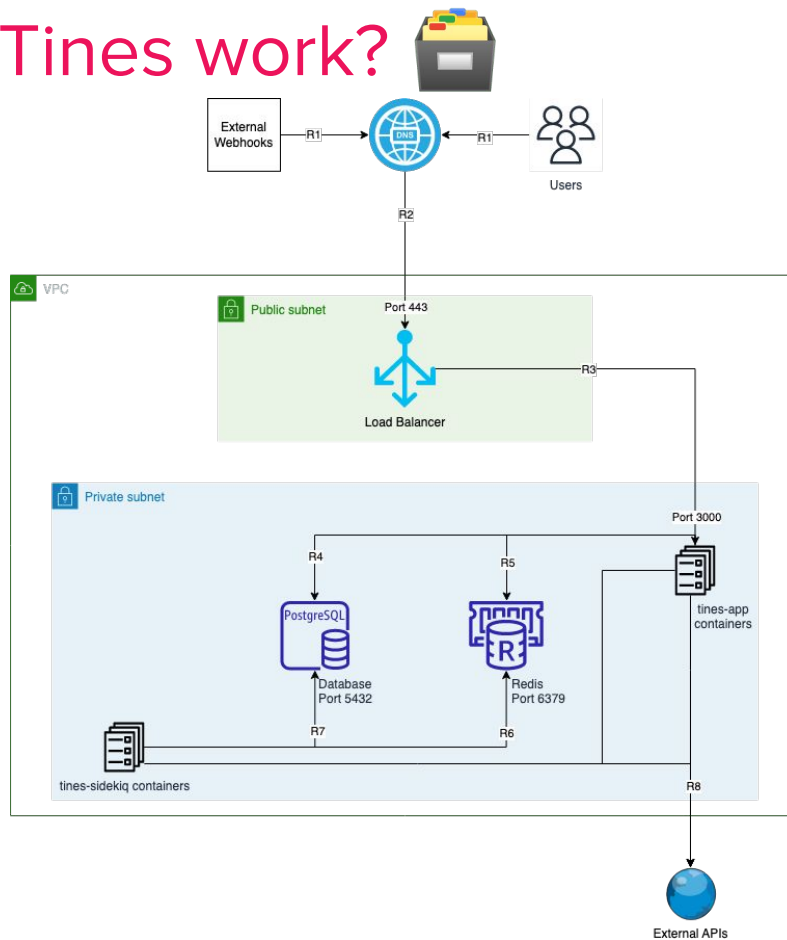


Railsconf
ATLANTA 2023

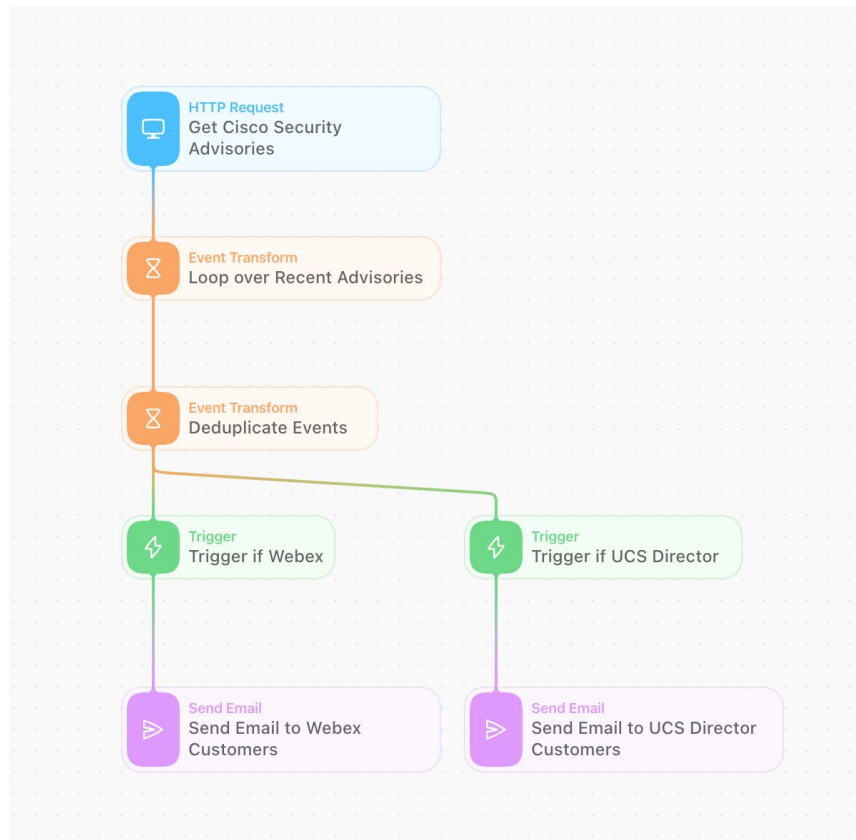
The time when we
lost all dynamically
scheduled customer
jobs 😊



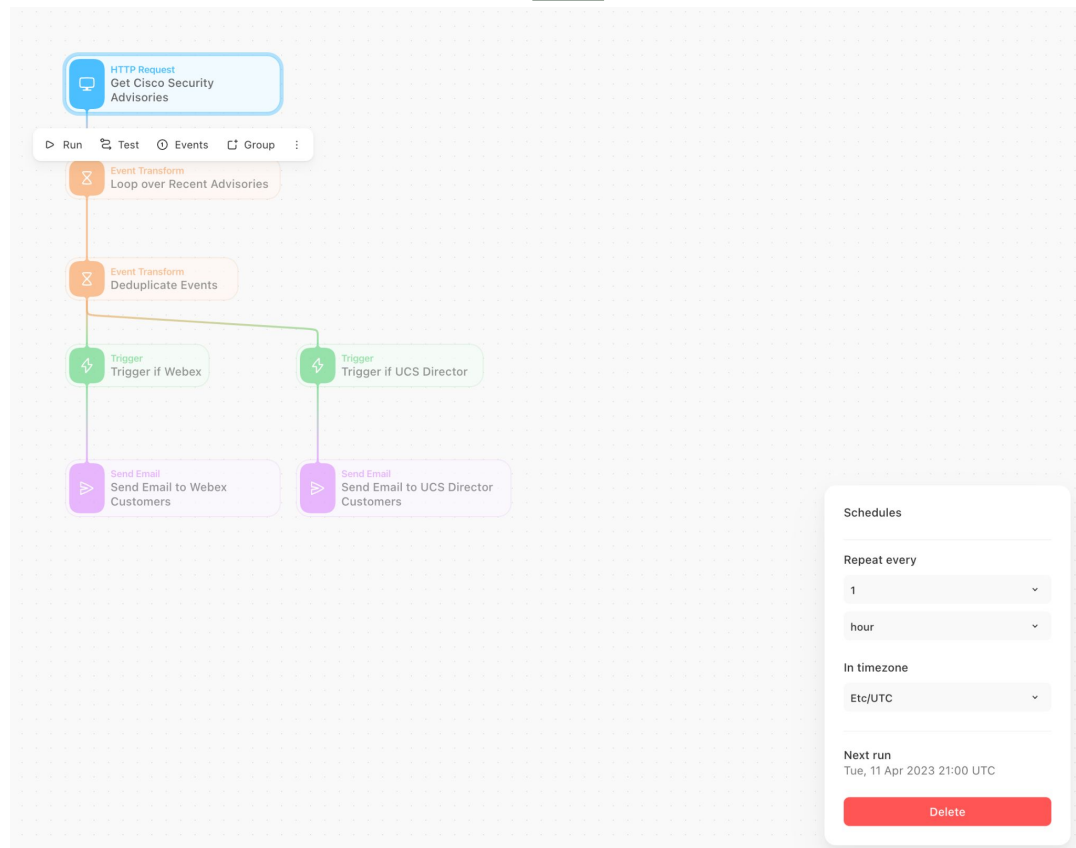
How does Tines work?



How does Tines work?



How does Tines work?



How does Tines work?



Table: actions

id	cron_expression	Timezone	
1	* * * * *	America/Chicago	
...	
923	0 * * * *	Etc/UTC	



Scheduling an Action



Scheduling an Action

Static Jobs

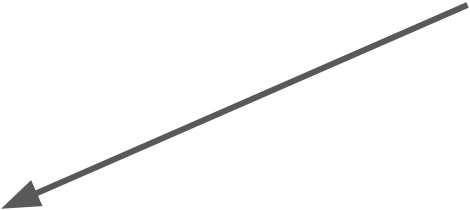
```
# sidekiq.yml
scheduler:
  schedule:
    "Scheduled::PopulateGravatarJob":
      cron: "0 * * * *" # Runs every hour
      description: "Populates gravatar url"
    "Scheduled::PoolAnalyticsEventJob":
      cron: "*/5 * * * *" #Runs every 5 min
      description: "Pool and send events to Analytics store"
```



Scheduling an Action

Dynamic Jobs

```
# sidekiq.yml
scheduler:
  dynamic: true
  schedule:
    "Scheduled::PopulateGravatarJob":
      cron: "0 * * * *" # Runs every hour
      description: "Populates gravatar url"
    "Scheduled::PoolMixpanelEventJob":
      cron: "*/5 * * * *" #Runs every 5 min
      description: "Pool and send events to Analytics store"
```



Scheduling an Action

Dynamic Jobs

```
Sidekiq.set_schedule(action.unique_key,  
  {  
    "cron" => cron_schedule_with_timezone,  
    "class" => AgentReceiveJob.to_s,  
    "args" => [action.id],  
  },  
)
```



Scheduling an Action

Internal Job

...

```
def perform
```

```
  Scheduler.reset_schedules!
```

```
  Scheduler.load_default_schedules!
```

```
  Action
```

```
    .joins(:story)
```

```
    .scheduled
```

```
    .available
```

```
    .find_each { |action| Scheduler.populate_action_schedules(action) }
```

```
  Scheduler.reload_schedule!
```

```
end
```

...



Scheduling an Action

Internal Job

...

```
def perform
```

```
  Scheduler.reset_schedules!
```

```
  Scheduler.load_default_schedules!
```

```
  Agent
```

```
    .joins(:story)
```

```
    .scheduled
```

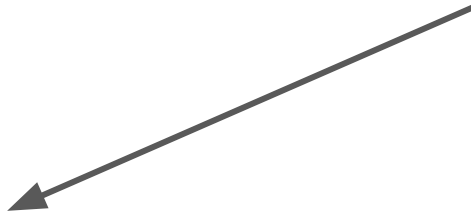
```
    .available
```

```
    .find_each { |action| Scheduler.populate_action_schedules(action) }
```

```
  Scheduler.reload_schedule!
```

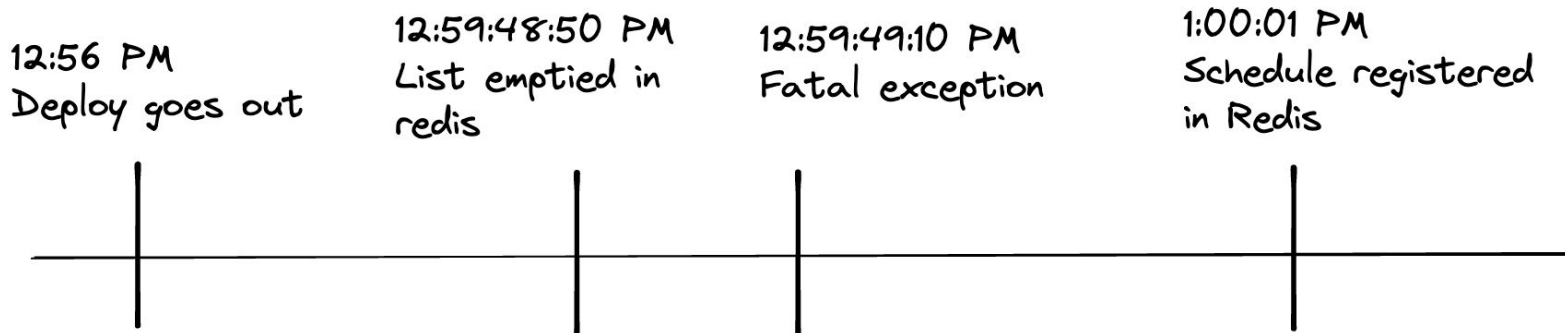
```
end
```

...



Scheduling an Action

Deploy



Scheduling an Action

Sidekiq scheduler

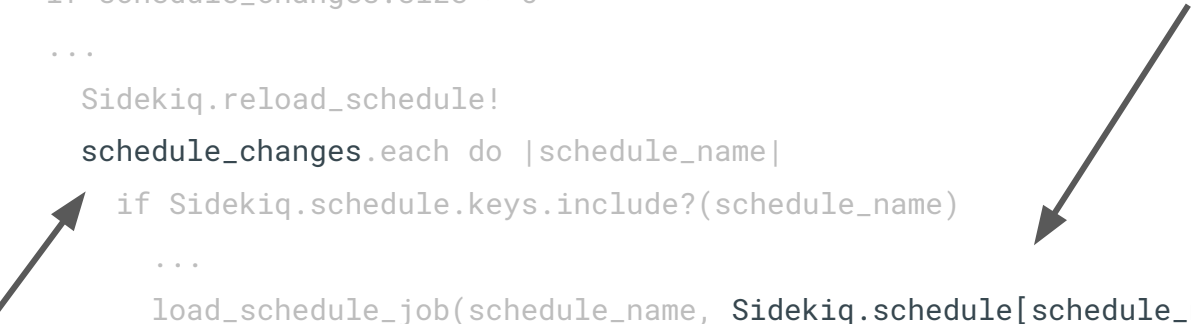
```
def update_schedule  
  ...  
  if schedule_changes.size > 0  
    ...  
    Sidekiq.reload_schedule!  
    schedule_changes.each do |schedule_name|  
      if Sidekiq.schedule.keys.include?(schedule_name)  
        ...  
        load_schedule_job(schedule_name, Sidekiq.schedule[schedule_name])  
      end  
    end  
  end  
end
```



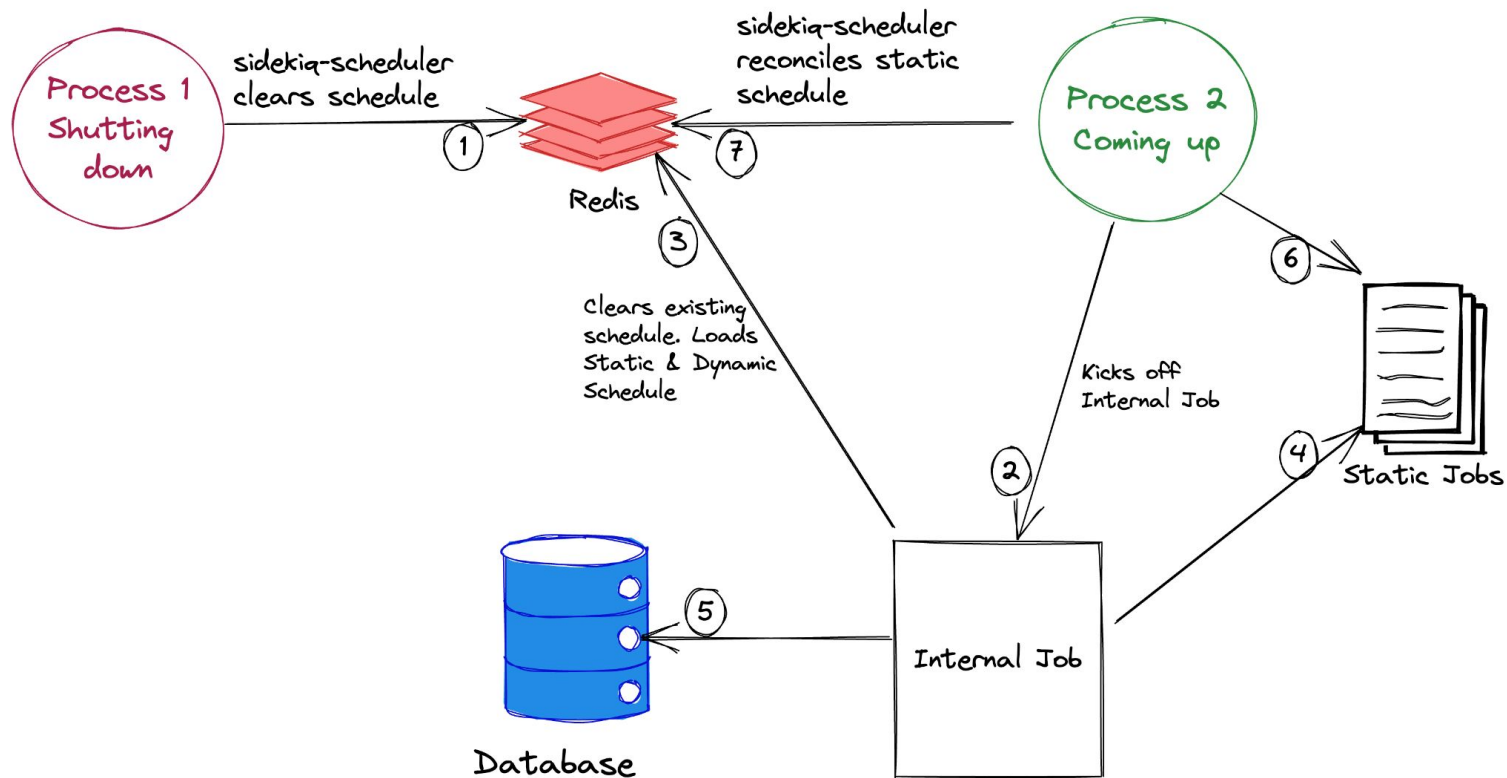
Scheduling an Action

Sidekiq scheduler

```
def update_schedule
  ...
  if schedule_changes.size > 0
    ...
    Sidekiq.reload_schedule!
    schedule_changes.each do |schedule_name|
      if Sidekiq.schedule.keys.include?(schedule_name)
        ...
        load_schedule_job(schedule_name, Sidekiq.schedule[schedule_name])
      end
    end
  end
end
```



Scheduling an Action



Back to first principles 🏆



Back to first principles 🏆

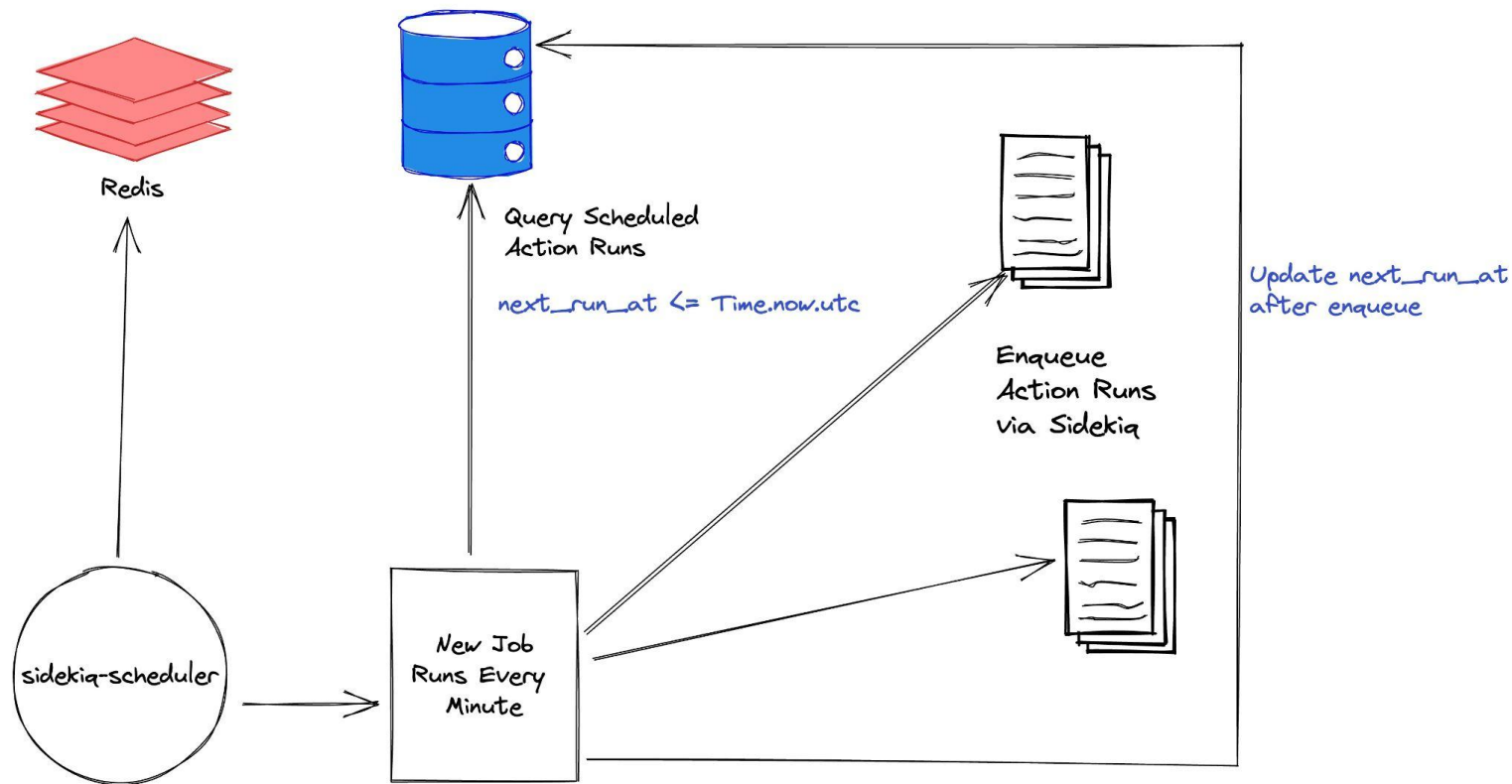
- 1 minute as lowest frequency of jobs
- Avoid bi-modality
- Avoid using new and complex technology
- Support for self hosted
- Embrace eventual consistency



New Scheduler System



New Scheduler System



New Scheduler System



```
class NewFanOutJob

  ...

  def perform

    ...

    ActionSchedule.where("next_run_at <= ?", Time.now.utc).find_each(&:run)

    ...

  end

end
```



New Scheduler System



```
class ActionSchedule
  def run
    return if next_run_at > Time.now.utc

    # optimistic concurrency control:
    rows_updated =
      self.class.where(id: id, cron: cron, timezone: timezone, next_run_at: next_run_at)
        .update_all(next_run_at: calculate_next_run_at)

    enqueue_action_run if rows_updated == 1
  end
end
```



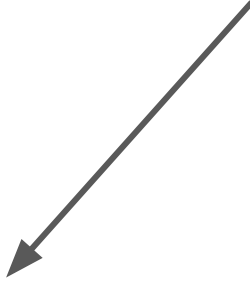
New Scheduler System



```
class ActionSchedule
  def run
    return if next_run_at > Time.now.utc

    # optimistic concurrency control:
    rows_updated =
      self.class.where(id: id, cron: cron, timezone: timezone, next_run_at: next_run_at)
        .update_all(next_run_at: calculate_next_run_at)

    enqueue_action_run if rows_updated == 1
  end
end
```



New Scheduler System



Types of locking

- Optimistic Locking (?) / Optimistic Concurrency Control
- Pessimistic Locking / Pessimistic Concurrency Control



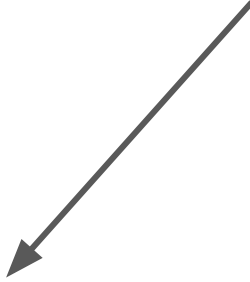
New Scheduler System



```
class ActionSchedule
  def run
    return if next_run_at > Time.now.utc

    # optimistic concurrency control:
    rows_updated =
      self.class.where(id: id, cron: cron, timezone: timezone, next_run_at: next_run_at)
        .update_all(next_run_at: calculate_next_run_at)

    enqueue_action_run if rows_updated == 1
  end
end
```



New Scheduler System



```
class ActionSchedule
  def run
    return if next_run_at > Time.now.utc

    self.with_lock do
      self.next_run_at = calculate_next_run_at
      self.save!

      enqueue_action_run
    end
  end
end
```



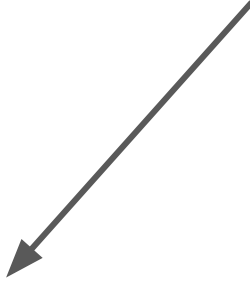
New Scheduler System



```
class ActionSchedule
  def run
    return if next_run_at > Time.now.utc

    # optimistic concurrency control:
    rows_updated =
      self.class.where(id: id, cron: cron, timezone: timezone, next_run_at: next_run_at)
        .update_all(next_run_at: calculate_next_run_at)

    enqueue_action_run if rows_updated == 1
  end
end
```



Skip'em with SKIP LOCKED

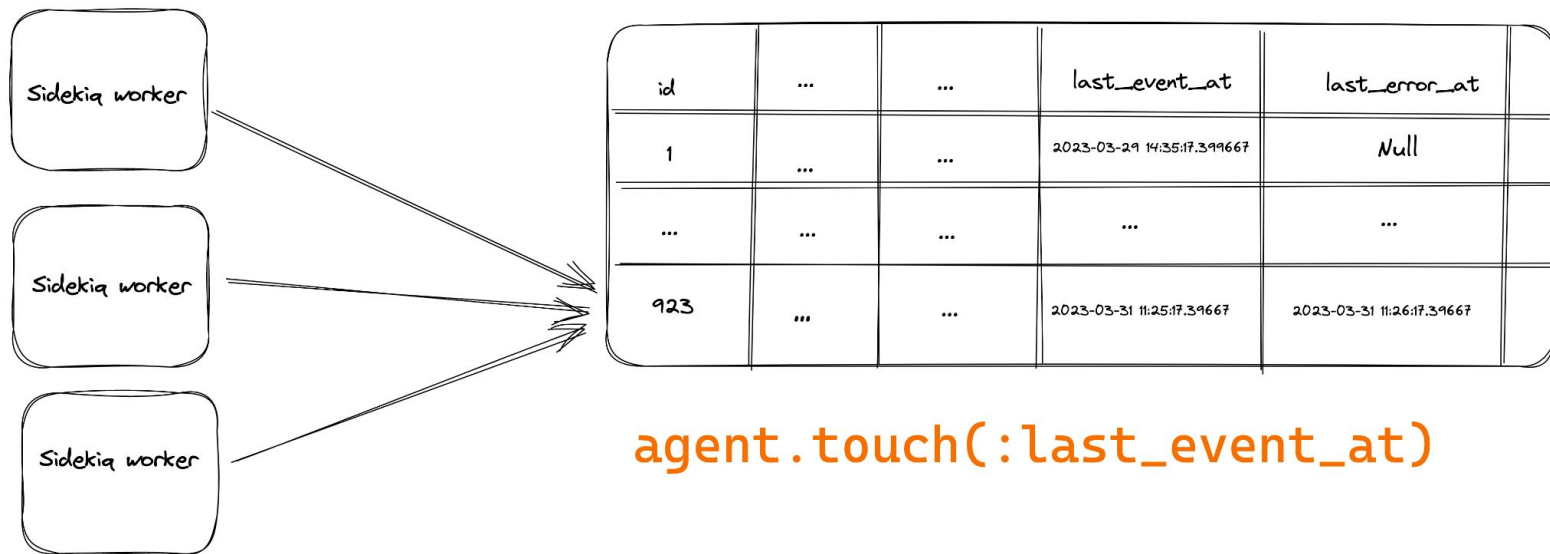


Skip'em with SKIP LOCKED

id	last_event_at	last_error_at	
1	2023-03-29 14:35:17.399667	Null	
...	
923	2023-03-31 11:25:17.39667	2023-03-31 11:26:17.39667	



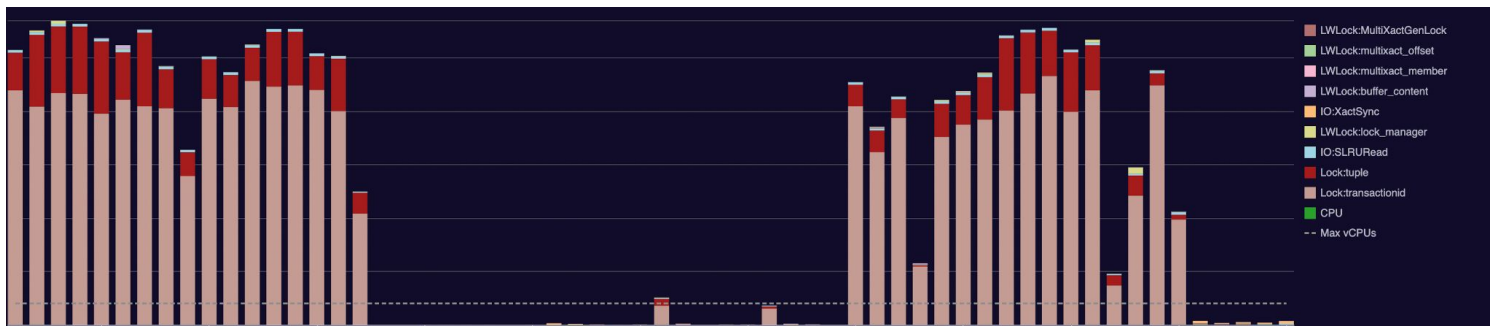
Skip'em with SKIP LOCKED



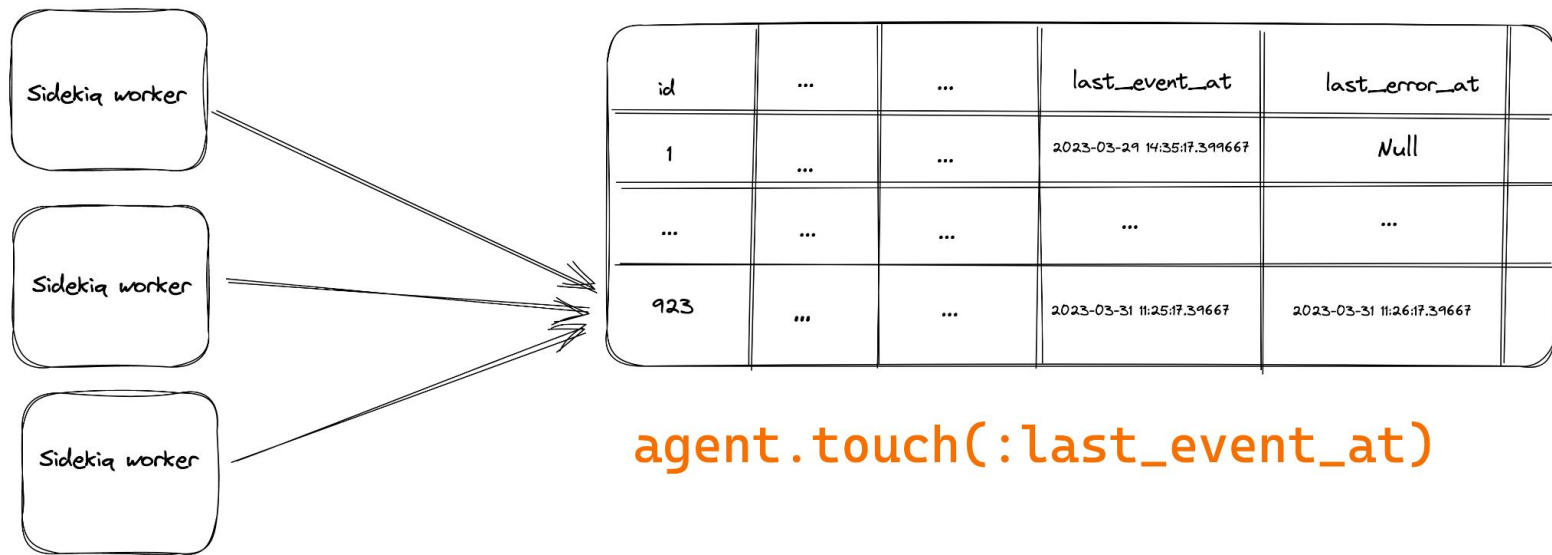
`agent.touch(:last_event_at)`



Skip'em with SKIP LOCKED



Skip'em with SKIP LOCKED



Skip'em with SKIP LOCKED

```
class ApplicationRecord < ActiveRecord::Base
  ...

  def update_columns_with_skipped_lock(args)
    record = self.class.lock("FOR UPDATE SKIP LOCKED").find_by(id: id)
    return if record.nil?
    record.update_columns(args)
  end

  ...
end
```

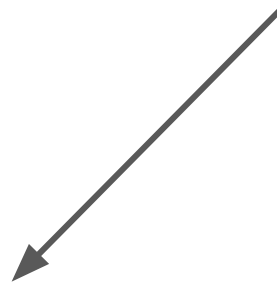


Skip'em with SKIP LOCKED

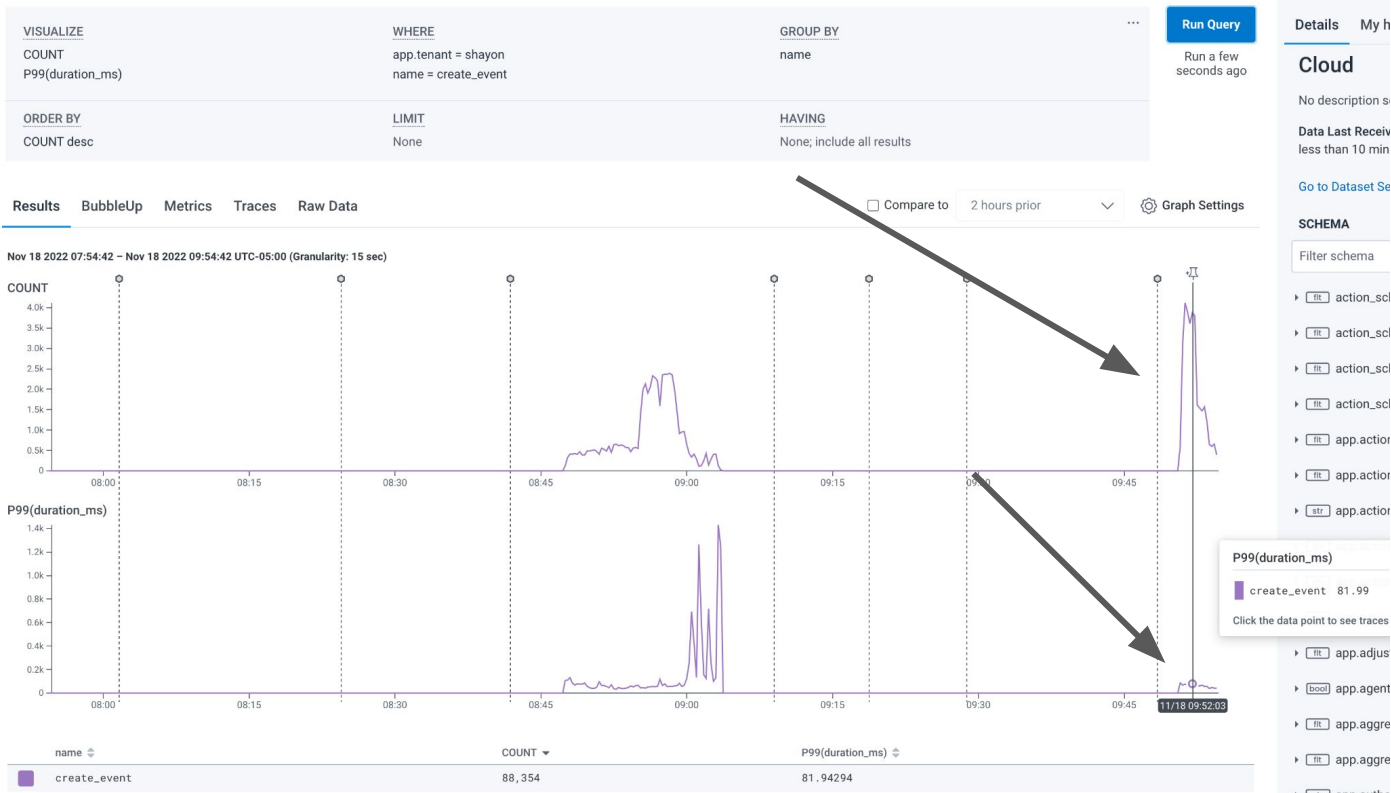
```
# action.touch(:last_event_at)
```

```
action.update_columns_with_skipped_lock(  
  last_event_at: Time.now,  
)
```

```
#> SELECT "agents".* FROM "agents" WHERE "agents"."id" = $1 LIMIT $1  
FOR UPDATE SKIP LOCKED
```



Skip'em with SKIP LOCKED



Synchronizing workloads with advisory locks



Synchronizing workloads with Postgres advisory locks



- Application defined lock
- Outside of the inbuilt MVCC (multiversion concurrency control)
- Control access to a shared resource
- Control access to a 3rd party API



Synchronizing workloads with Postgres advisory locks



acquire

```
SELECT pg_try_advisory_lock(10);
```

release

```
SELECT pg_try_advisory_unlock(10);
```



Synchronizing workloads with Postgres advisory locks



- github.com/ClosureTree/with_advisory_lock
- github.com/heroku/pg_lock



Synchronizing workloads with Postgres advisory locks



```
User.with_advisory_lock(lock_name) do
  do_something_that_needs_locking
end
```

```
User.with_advisory_lock(lock_name, { timeout_seconds: 30 }) do
  do_something_that_needs_locking
end
```



Synchronizing workloads with Postgres advisory locks



```
class Scheduled::PoolAnalyticsEventJob

  ...

  def perform

    ...

    ApplicationRecord.with_advisory_lock("analytics-job", { timeout_seconds: 30 }) do

      # send analytics events

    end

    ...

  end

end
```



Timeouts, Reconnects and Retries



Timeouts, Reconnects and Retries



production:

adapter: postgresql

encoding: utf8

...

variables:

statement_timeout: 60000 # 1min

lock_timeout: 5000 # 5s

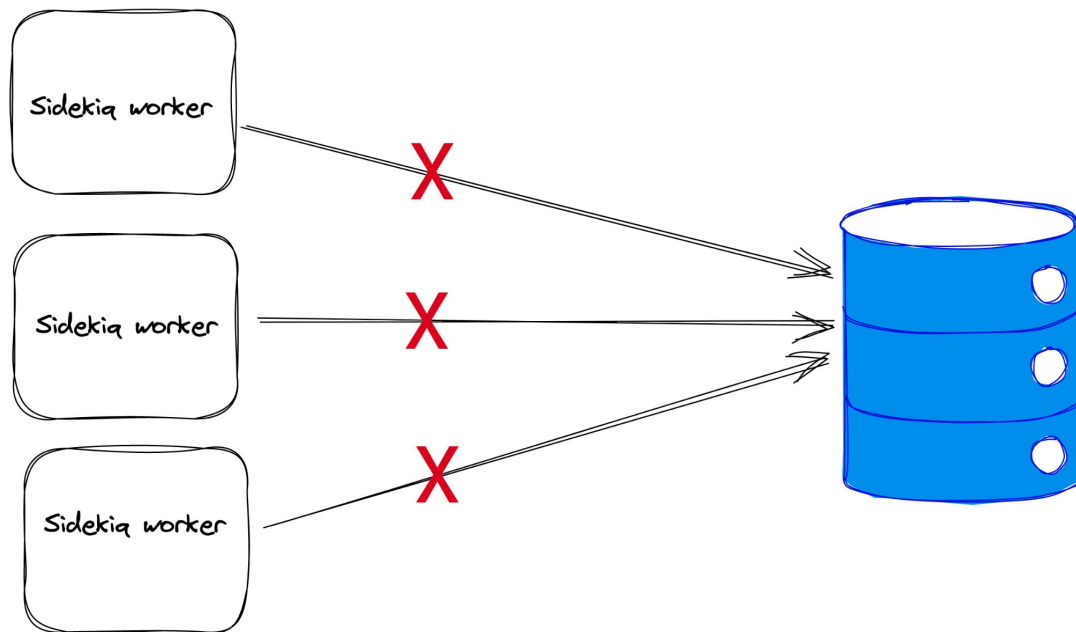
idle_in_transaction_session_timeout: 300000 # 5min



Timeouts, Reconnects and Retries



ActiveRecord Connection Pools



Timeouts, Reconnects and Retries



```
def exec_no_cache(*args)
  super(*args)
rescue ActiveRecord::StatementInvalid, ActiveRecord::ConnectionNotEstablished => e
  if failover_error?(e.message)
    disconnect!

    ActiveRecord::Base.connection_pool.remove(::ActiveRecord::Base.connection)

    raise if in_transaction?

    reconnect_and_retry _with_backoff!
  end
end
```



Timeouts, Reconnects and Retries



Rails 6.x: github.com/tines/rails-pg-adapter

Rails 7 (coming soon)

`execute(sql, name = nil, allow_retry: false)`

[Link](#)

Executes the SQL statement in the context of this connection and returns the raw result from the connection adapter.

Setting `allow_retry` to true causes the db to reconnect and retry executing the SQL statement in case of a connection-related exception. This option should only be enabled for known idempotent queries.

Note: depending on your database connector, the result returned by this method may be manually memory managed. Consider using the `exec_query` wrapper instead.

Source: [show](#) | [on GitHub](#)

❤ Rails core team and Shopify



Timeouts and Reconnects

github.com/tines/rails-pg-adapter

Auto heal from **PG::UndefinedColumn**

```
SELECT "events"."id",  
       "events"."old_message"  
FROM   "events"  
WHERE  "events"."agent_id" IS NOT NULL  
ORDER BY "events"."id" DESC  
LIMIT  $1
```



Timeouts and Reconnects

github.com/tines/rails-pg-adapter

```
def exec_no_cache(*args)
  super(*args)
rescue ActiveRecord::StatementInvalid=> e
  if column_error?(e.message)
    ActiveRecord::Base.connection_pool.connections.do |conn|
      conn.schema_cache.clear!
    end
    ActiveRecord::Base.descendants.each(&:reset_column_information)
    raise
  end
end
```



Till next time 🙌

- The time when we lost all dynamically scheduled customer jobs ☐
- Optimistic and Pessimistic Locking 🔒
- Skip'em with SKIP LOCKED ⏭
- Synchronizing workloads with Advisory Locks 🔒
- Timeouts and Reconnects ⌚



Special thanks 🙏

- @noelrap
- @serifritsch
- @ronocod
- @team-carbon Tines



Till next time 🙌



tines.com/careers



Till next time 🙌



@shayonj



@shayonj

