

Radek Rochmalski, KRUG

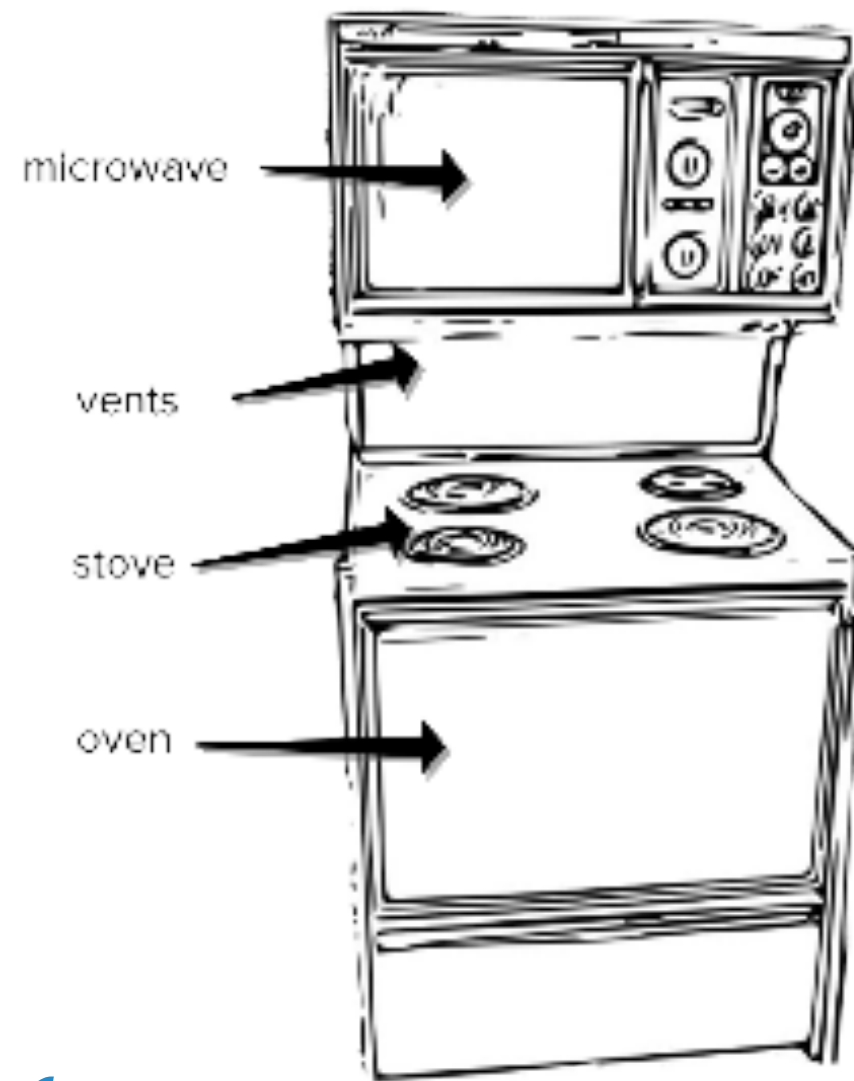
# **MICROSERVICES AND THEIR COMMUNICATION**

**(RABBITMQ / SIDEKIQ)**

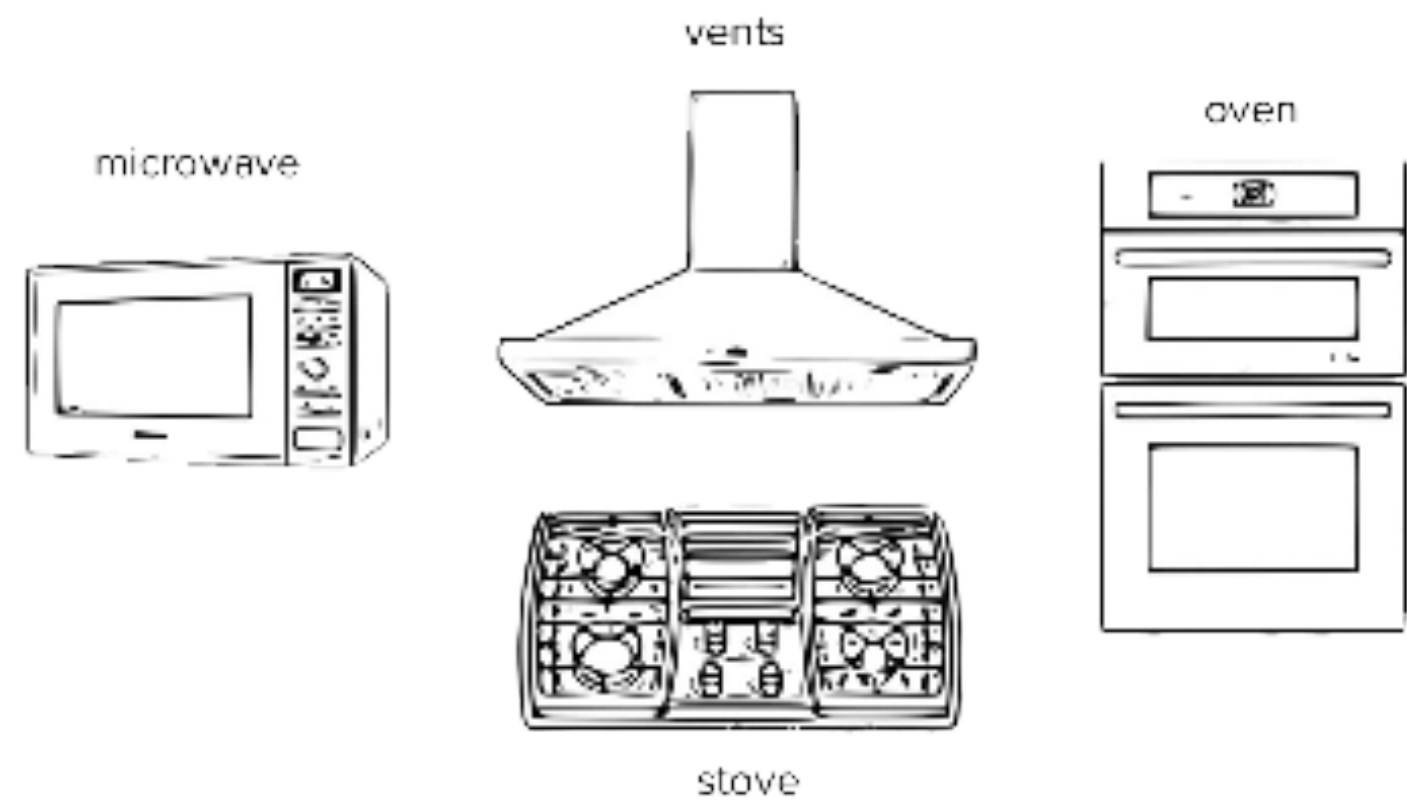
**Czym są mikroserysy?**

# Czym są mikroserwisy?

Monolithic Kitchen



Microservices Kitchen



**Dlaczego warto?**

**Dlaczego warto?**

**Sprawne wytwarzanie  
i rozwój oprogramowania.**

***inFakt***

**Dlaczego warto?**

**Niezależność  
od innych usług**

***inFakt***

Dlaczego warto?

Dobór technologii  
do problemu

***inFakt***

**Dlaczego warto?**

**Utrzymywanie  
i skalowalność**

***inFakt***



**Co może być wyzwaniem?**

Co może być wyzwaniem?

# Testowanie systemu usług

***inFakt***

Co może być wyzwaniem?

# Dekompozycja systemu monolitycznego

***inFakt***

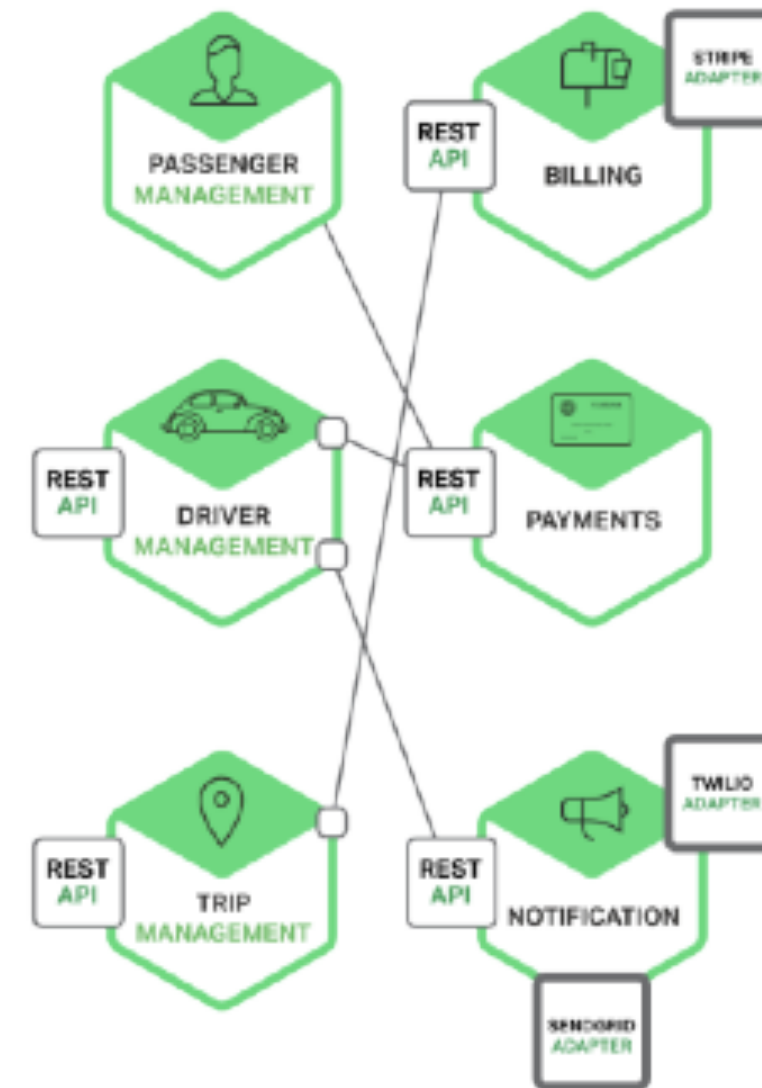
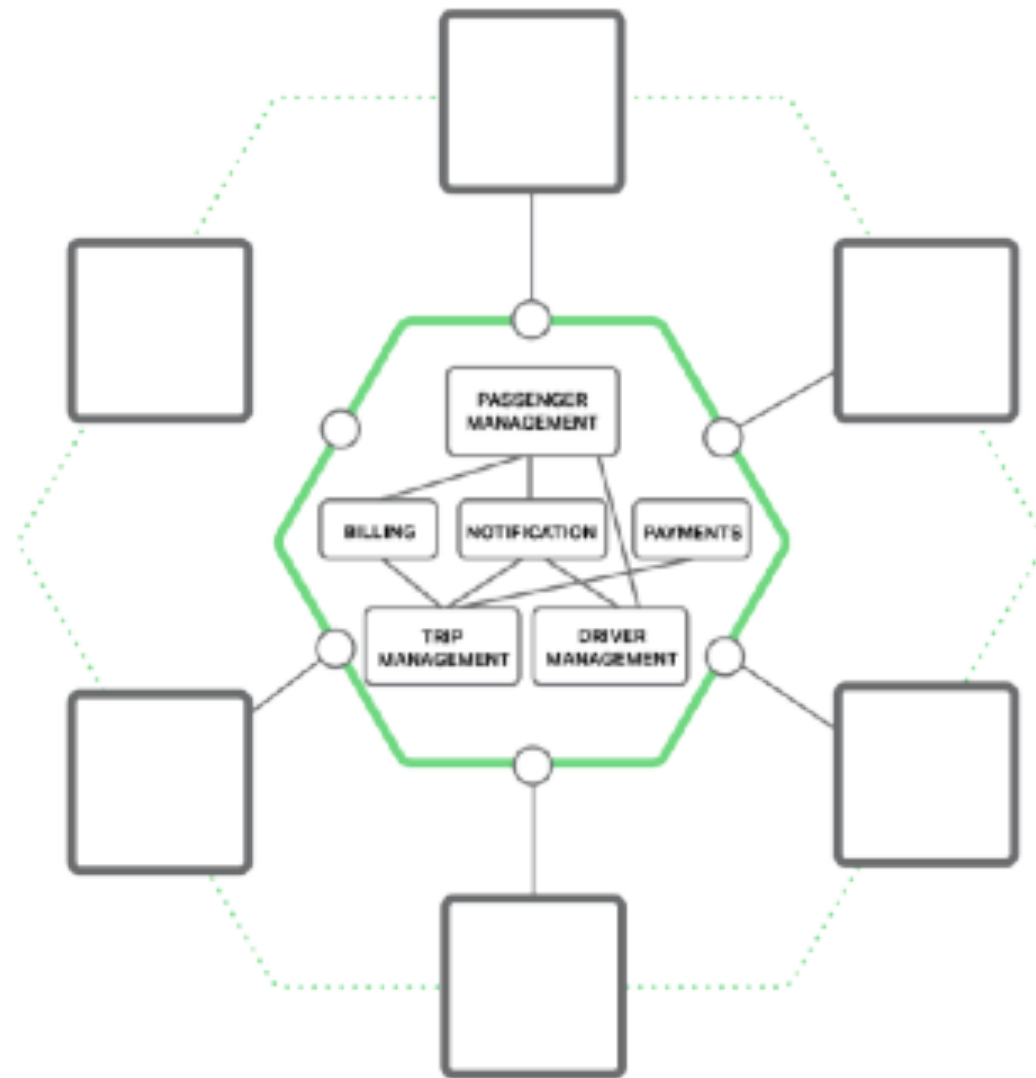
Co może być wyzwaniem?

Zarządzanie  
i monitorowanie

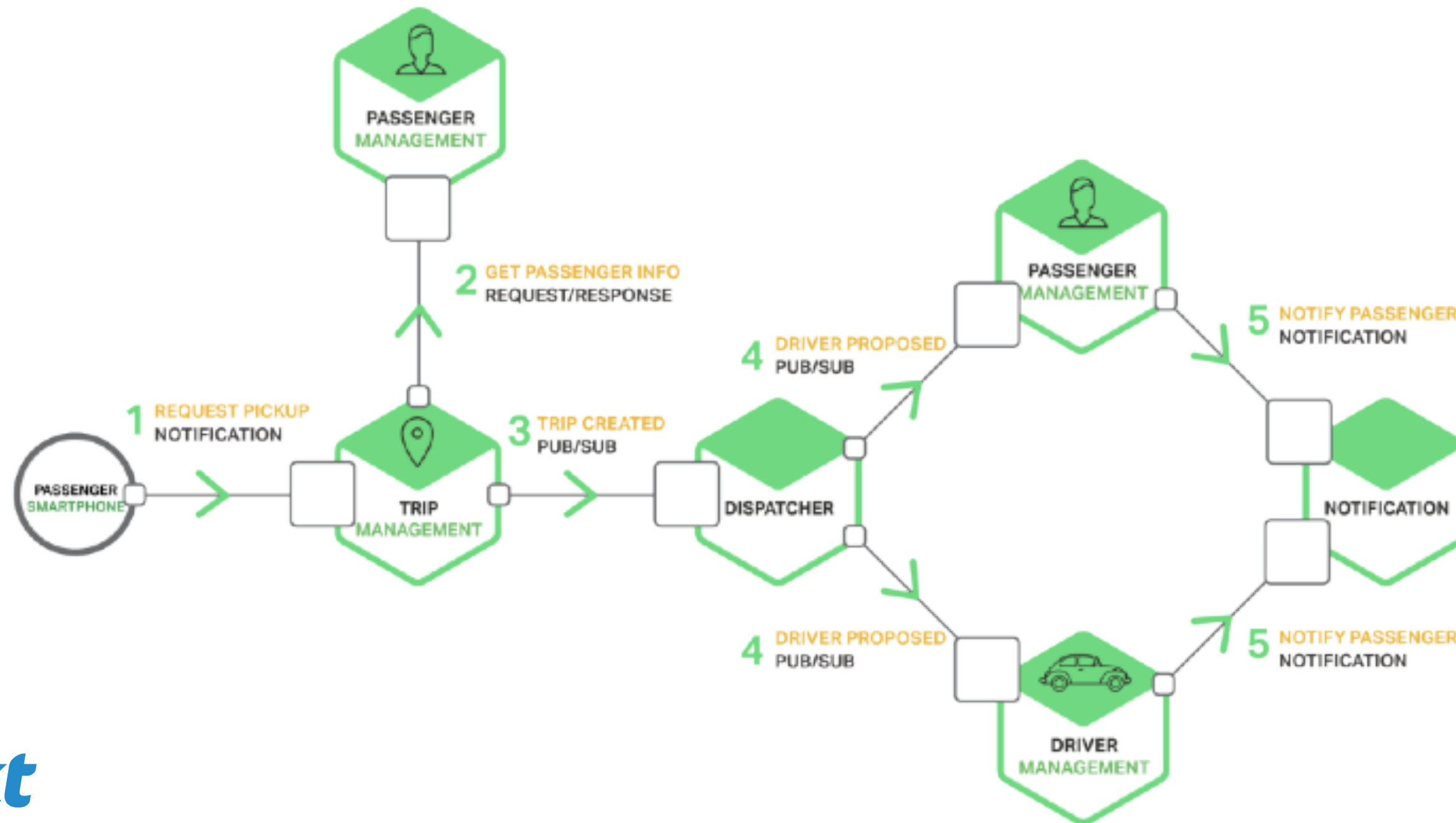
***inFakt***

# Komunikacja między usługami

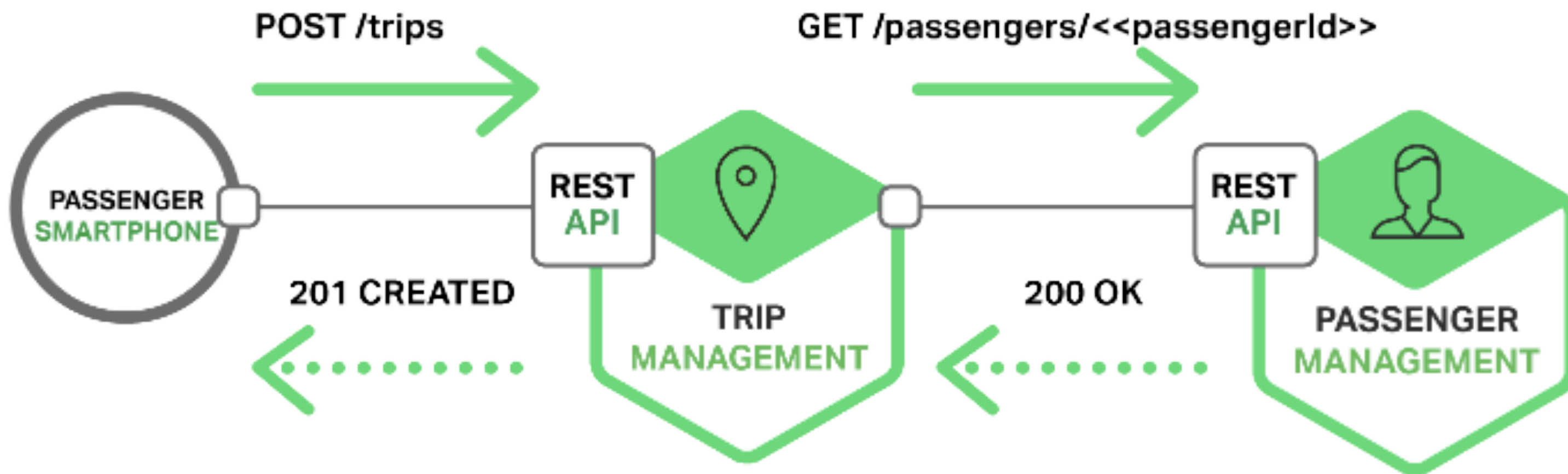
# Komunikacja między usługami



# Komunikacja między usługami



## Komunikacja między usługami





The background is a solid blue color with a repeating pattern of white, stylized icons. These icons include various symbols such as percentages (%), dollar signs (\$), numbers (1, 2, 3, 18, 23), arrows, envelopes, and question marks, all rendered in a clean, modern font style.

# Klient HTTP

# Klient HTTP - EXCON

## gemfile

```
gem 'excon'
```

## simple\_client.rb

```
connection = Excon.new('https://example.com')
get_response = connection.get(path: '/resources')
post_response = connection.post(path: '/resources')
delete_response = connection.delete(path: '/resource/1')
```

## api\_client.rb

```
connection = Excon.new('https://example.com/api/v1/resources', headers: {'Content-Type' =>
'application/json', 'X-API-Key' => 'secret'})

connection.request(method: :get, query: { page: 1, per_page: 10 })
connection.request(method: :post, body: object.attributes.to_json)
Excon.new('https://example.com', retry_limit: 5, idempotent: true)
```

# Klient HTTP - FARADAY

## gemfile

```
gem 'faraday'
```

## simple\_client.rb

```
connection = Faraday.new('https://example.com')
get_response = connection.get('/resources')
post_response = connection.post('/resources')
delete_response = connection.delete('/resource/1')
```

## api\_client.rb

```
connection = Faraday.new('https://example.com/api/v1', headers: {'Content-Type' =>
'application/json', 'X-API-Key' => 'secret'})

connection.get '/resources', { page: 1, per_page: 10 }
connection.post '/resources', object.attributes.to_json
connection.get '/resources', options: { timeout: 5, open_timeout: 2 }
```

# Klient HTTP - REST-CLIENT

## gemfile

```
gem 'rest-client'
```

## simple\_client.rb

```
get_response = RestClient.get('https://example.com/resources')
post_response = RestClient.post('https://example.com/resources', {})
delete_response = RestClient.delete('https://example.com/resource/1')
```

## api\_client.rb

```
RestClient.get('http://example.com/api/v1/resources', {content_type: :json, accept: :json,
'X-API-Key' => 'secret', params: { page: 1, per_page: 10 }})

RestClient.post('https://example.com/api/v1/resources', {content_type: :json, accept: :json,
'X-API-Key' => 'secret'}, object.attributes.to_json)
```

# Kolejkowanie i obsługa zdarzeń SIDEKIQ

# Kolejkowanie i obsługa zdarzeń SIDEKIQ

## **Prosty w integracji**

priorytety, ponawiane zadań, kontrola współbieżności

## **Wydajny, wielowątkowy**

4 500 requests per second (req/s)

## **Oparty na Redis'ie**

# Kolejkowanie i obsługa zdarzeń SIDEKIQ

## gemfile

```
gem 'sidekiq'
```

## my\_worker.rb

```
class MyWorker
  include Sidekiq::Worker sidekiq_options queue: 'critical'

  def perform(url, json_data)

    connection = Excon.new(url, headers:
    {'Content-Type' => 'application/json', 'X-API-Key' => 'secret'})
    connection.request(method: :post, body: json_data)

  end end
```

## usage

```
MyWorker.perform_async('http://...', { name: 'Jhon' }.to_json)
```

```
MyWorker.perform_in(3.minutes, 'http://...', { name: 'Jhon' }.to_json)
```

# Kolejkowanie i obsługa zdarzeń RABBITMQ



## Kolejkowanie i obsługa zdarzeń RABBITMQ

### **Bardzo wydajny**

44 824 req/s (max. wysyłka 149 910 req/s, max. odbiór 64 315 req/s)

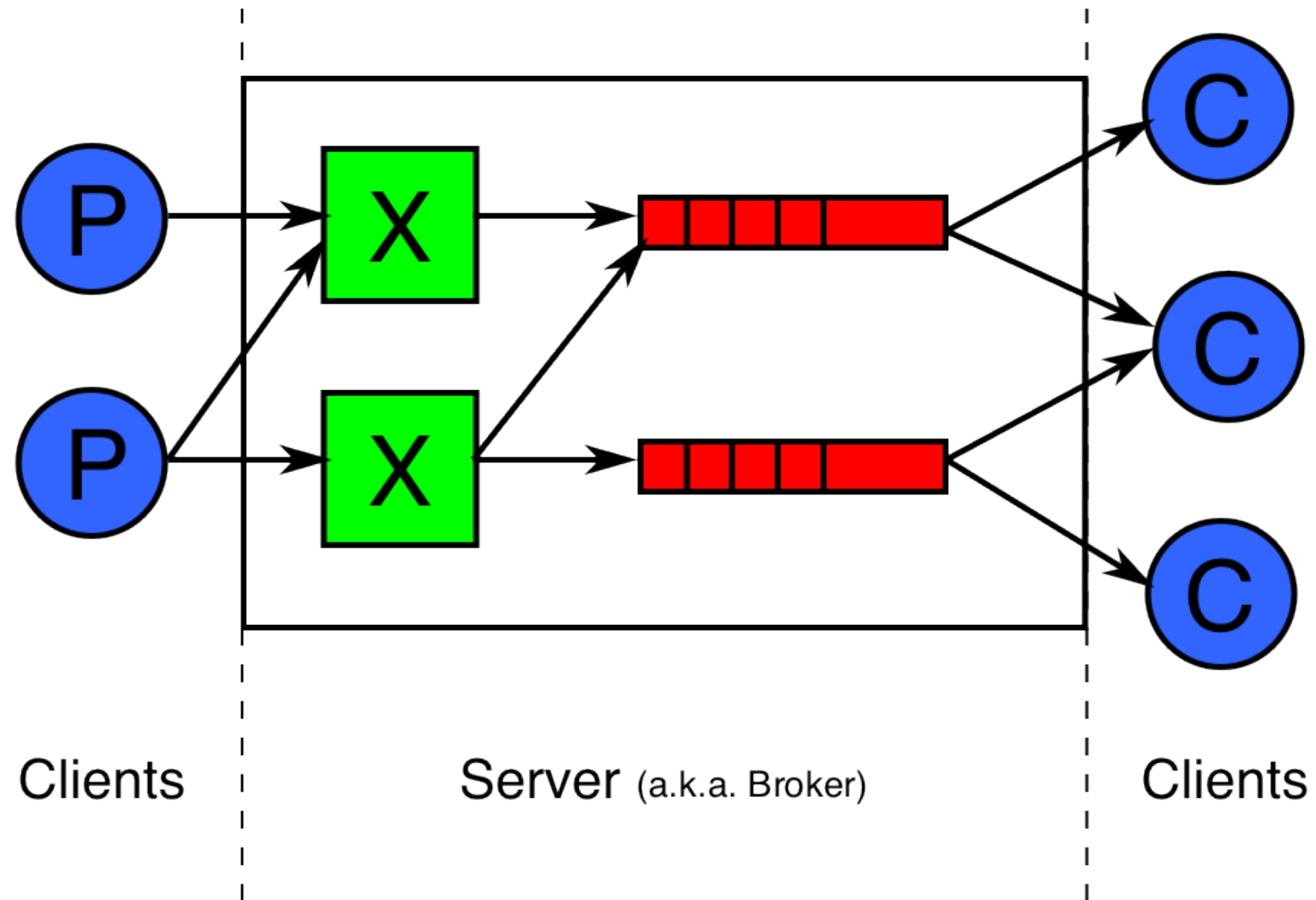
### **Obsługuje różne platformy programistyczne**

Ruby, Java, Python, Node.js, C / C++, Erlang, Go

### **Dostępny dla większości systemów**

MacOS X | Debian / Ubuntu | Amazon EC2 | Cloud Foundry | Windows

## Kolejkowanie i obsługa zdarzeń RABBITMQ



## Kolejkowanie i obsługa zdarzeń RABBITMQ - podstawowe pojęcia

**Connection** - połączenie TCP z brokerem wiadomości

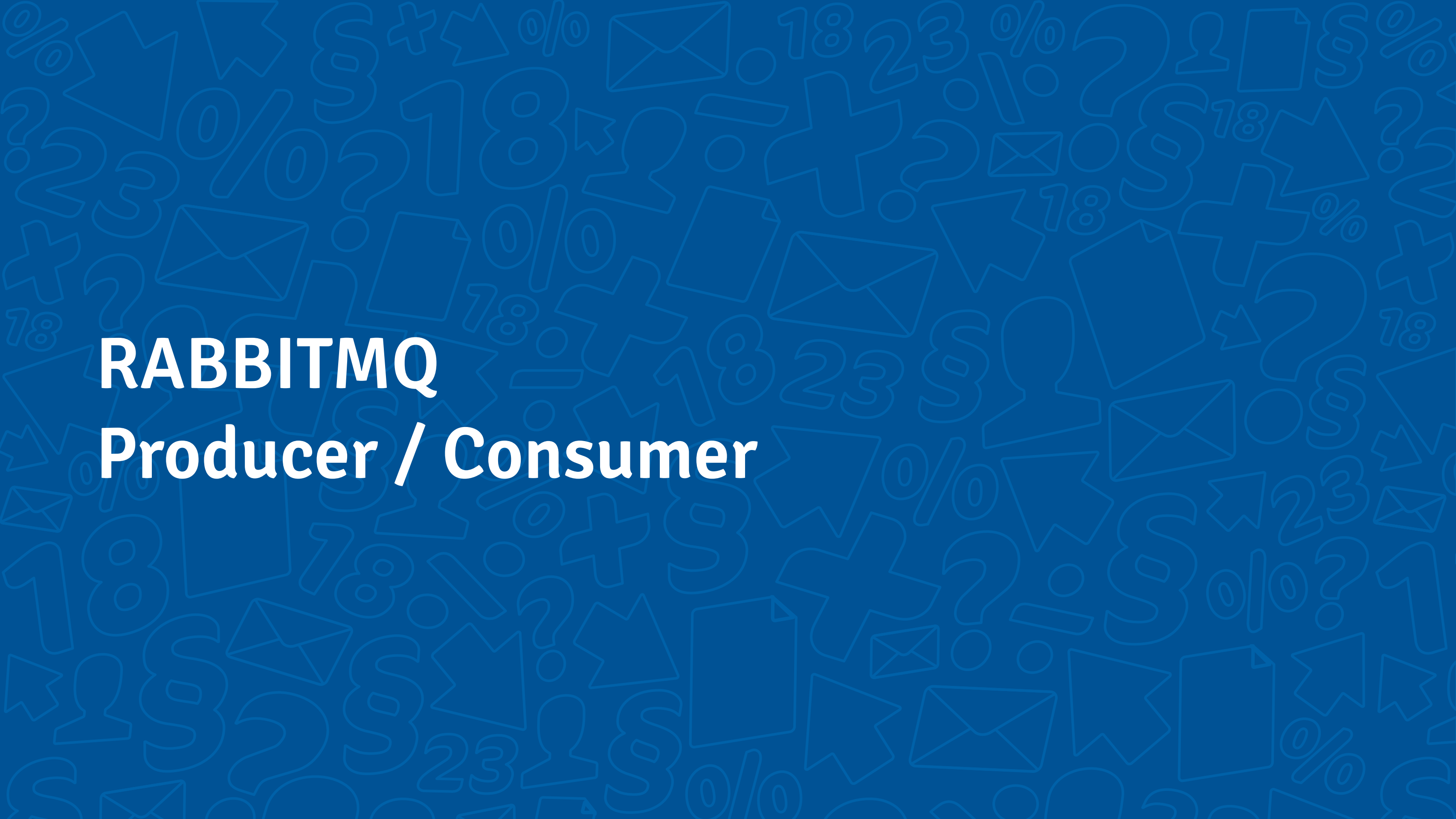
**Channel** - wirtualne połączenie wewnątrz TCP

**Message** - wiadomość o dowolnej formie i treści

**Queue** - kolejka do której wysyłane są wiadomości

**Exchange** - przyjmuje wysłane wiadomości i rozsyła je do kolejek

Exchange: Direct, Fanout, Topic, Headers



# **RABBITMQ**

## **Producer / Consumer**

## Kolejkowanie i obsługa zdarzeń BUNNY

### **Funkcjonalny**

Oferuje wszystko co posiada RabbitMQ

### **Wygodny**

Sam odzyskuje połączenia i wiadomości po awarii

### **Zaawansowany**

Dowolność w sposobie konfiguracji

# Kolejkowanie i obsługa zdarzeń BUNNY

## gemfile

```
gem 'bunny'
```

## producer.rb

```
require 'bunny' connection = Bunny.new

connection.start

channel = connection.create_channel
queue = channel.queue('app.events.user.created.development') exchange =
channel.default_exchange

exchange.publish({ url: 'https://example.com/api/v1/users',
data: { name: 'John', age: 21 } }.to_json, routing_key: queue.name)

connection.close
```

## usage

```
ruby producer.rb
```

# Kolejkowanie i obsługa zdarzeń BUNNY

## producer.rb

```
require 'bunny'

connection = Bunny.new

connection.start
channel = connection.create_channel

queue = channel.queue( 'app.events.user.created.development' )

loop do
  queue.subscribe do |delivery_info, metadata, payload|
    # Excon...
  end
end

connection.close
```

## usage

```
ruby consumer.rb
```

## Kolejkowanie i obsługa zdarzeń SNEAKERS

**Bardzo wydajny**

Ogranicznony jedynie prędkością brokera (min. 1000 req/s)

**Prosty w integracji**

Mocno wzorowany na Sidekiq

**Zaawansowana semantyka wiadomości**

Reject, requeue, acknowledge



# Kolejkowanie i obsługa zdarzeń SNEAKERS

## gemfile

```
gem 'sneakers'
```

## my\_consumer.rb

```
class Consumer
  include Sneakers::Worker
  from_queue 'app.events.user.created.development',
  prefetch: 25, threads: 20, timeout_job_after: 15

  def process(message)
    # Excon...
    ack!
  end
end
```

## usage

```
sneakers work Consumer --require consumer.rb
require 'sneakers/tasks'
$ WORKERS=Consumer rake sneakers:run
```

## Kolejkowanie i obsługa zdarzeń HUTCH

**Prosty w integracji**  
connect, publish, consume

**Specyficzny**  
Wysyłka wiadomości tylko Topic

**Oparty na Bunny**  
lub March Hare (JRuby)

***inFakt***

# Kolejkowanie i obsługa zdarzeń HUTCH

## gemfile

```
gem 'hutch'
```

## producer.rb

```
require 'hutch'
```

```
Hutch.connect
```

```
Hutch.publish('app.events.user.created.development',  
url: 'https://example.com/api/v1/users',  
data: { name: 'Jhon', age: 21, email: 'jhon@mail.com' }.to_json)
```

## usage

```
hutch --require hutch/producer.rb
```

```
hutch --require consumer.rb --pidfile path_to_hutch_pid --daemonise
```

# Kolejkowanie i obsługa zdarzeń HUTCH

## consumer.rb

```
require 'hutch'

class Consumer
  include Hutch::Consumer
  consume 'app.events.user.*.development'

  def process(message)
    url, data = message.url, message.user_data

    connection = Excon.new(url, headers:
      {'Content-Type' => 'application/json', 'X-API-Key' => 'secret'})
    connection.request(method: :post, body: data) end
  end
end
```

## usage

```
hutch --require hutch/consumer.rb
```

# **Aplikacje zewnętrzne na przykładzie mikro-frameworka Roda**

## Aplikacje zewnętrzne na przykładzie mikro-frameworka RODA

**Szybka i prosta**

Core aplikacji to drzewo routing'u

**Niezawodna i wydajna**

Inteligentny cache struktur danych

**Elastyczna**

Wiele wtyczek oraz dodatków

***inFakt***

# Aplikacje zewnętrzne na przykładzie mikro-frameworka RODA

## app.rb

```
require 'roda'
class App < Roda
  plugin :render
  route do |r|
    r.root { view :index } # GET / request
    r.on 'resources' do # /resources branch
      r.is do
        # GET /resources request
        r.get { view :resources }
        # POST /resources request
        r.post { Creator.perform(r.params); r.redirect }
      end
    end
  end
end
```

# Aplikacje zewnętrzne na przykładzie mikro-frameworka RODA

## app.rb

```
require 'roda'
class App < Roda
  plugin :render
  plugin :rest_api
  route do |r|
    r.root { view :index }
  end
  r.api
  r.version 1 do
    unless r.env['HTTP_X_API_KEY'] == 'secret'
      r.redirect('Unauthorized', 401)
    end
  end
end
```



# Aplikacje zewnętrzne na przykładzie mikro-frameworka RODA

## app.rb

```
require 'roda'
class App < Roda
  plugin :render
  plugin :rest_api
  route do |r|
    r.api
    r.version 1 do
      r.resource :resources do |resources|
        resources.save { ResourceCreator.perform(r.params) }
        resources.list { |params| ResourceCollector.perform(params) }
        resources.routes :index, :create
        resources.permit r.params

      end
    end
  end
end
end
```

# Aplikacje zewnętrzne na przykładzie mikro-frameworka RODA

## app.rb

```
require 'roda'
class App < Roda
  plugin :path
  plugin :assets
  plugin :render, views: 'app/views', engine: 'erb'
  plugin :error_handler
  plugin :json_parser
  plugin :json
  plugin :basic_auth

  route do |r|
    r.root { view :index }
  end
end
```

# Podsumowanie

**Podsumowanie**

**Pytania?**

***inFakt***

**Dziękuję za uwagę!**

**Dane kontaktowe**

**Radosław Rochmalski**  
**Ruby Developer**

e-mail: [radoslaw.rochmalski@infakt.pl](mailto:radoslaw.rochmalski@infakt.pl)

***inFakt***