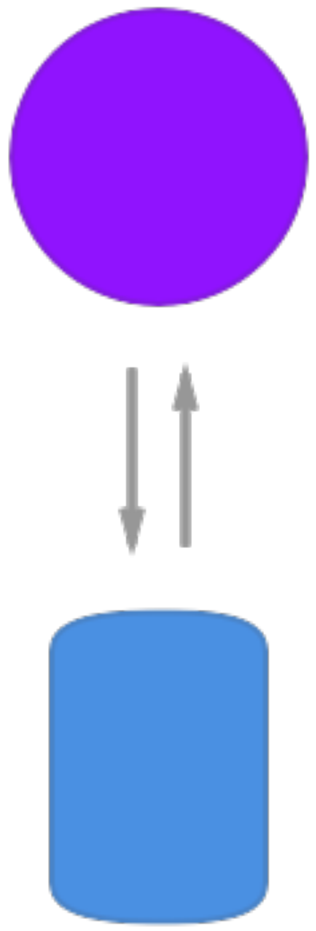
Microservices at Wunderlist

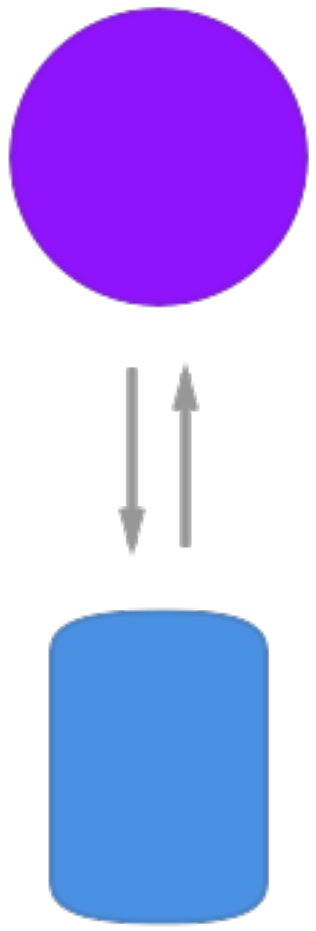
Wunderlist 1

PHP + MySQL

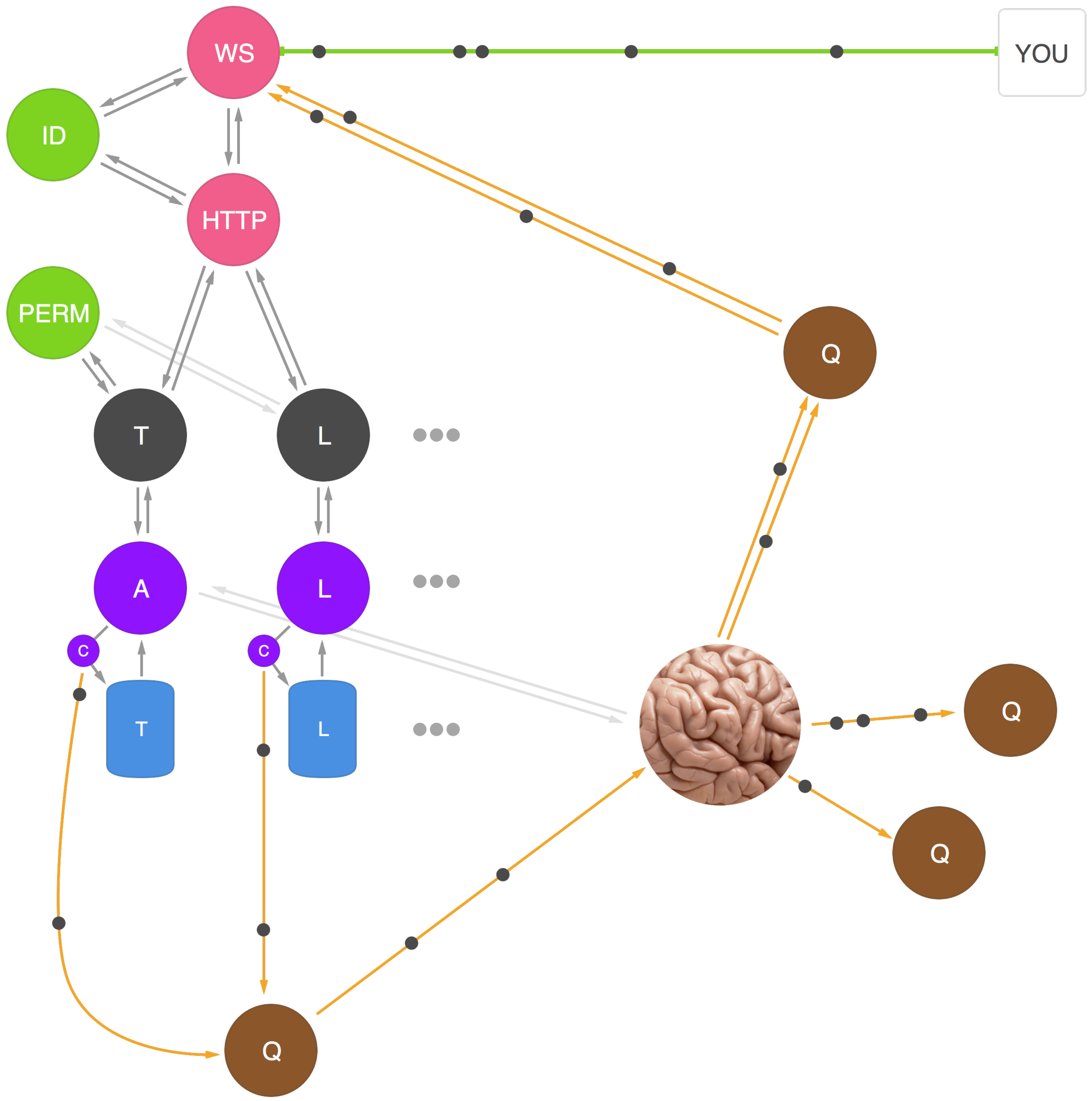


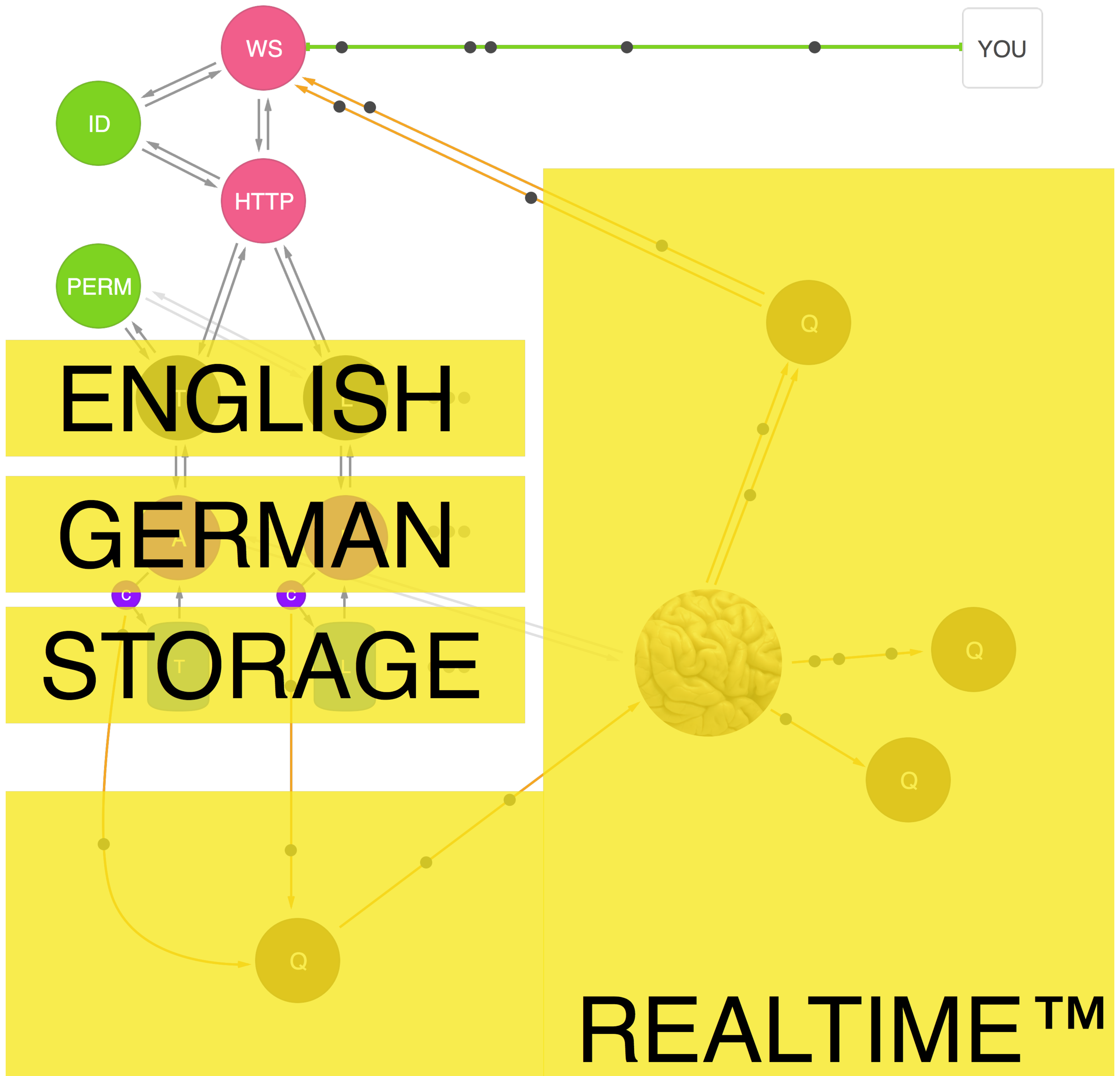
Wunderlist 2

Rails + Postgres



Wunderlist 3


WS
YOU
ID
HTTP
PERM
Q
T
L
A
L
C
C
T
L
Q
Q
Q


WS
YOU
ID
HTTP
PERM
ENGLISH
GERMAN
STORAGE
C
C
Q
Q
Q
Q
REALTIME™

Synchronous & Asynchronous

Every change generates a mutation

```json
{
  "recipient_id": 123456,
  "version": 1,
  "data": {
    "created_by_id": 123456,
    "revision": 1,
    "starred": false,
    "completed": false,
    "is_recurrence_child": false,
    "title": "Hello Microservices",
    "updated_at": "2015-07-16T17:44:51.735Z",
    "created_by_request_id": "...",
    "id": 123456,
    "list_id": 123456,
    "created_at": "2015-07-16T17:44:51.735Z"
  },
  "operation": "create",
  "subject": {
    "id": 123456,
    "type": "task",
    "revision": 1,
    "previous_revision": 0,
    "parents": [{
      "id": 123456,
      "type": "list"
    }]
  },
  "client": {
    "id": "abc...",
    "request_id": "...",
    "device_id": "...",
    "instance_id": "...",
    "user_id": "123456"
  },
  "type": "mutation"
}
```

Object Oriented Server Architecture

```
class TaskFetch
  def get(id:)                                  # GET /api/v1/tasks/123
  end

  def all(list_id:)                             # GET /api/v1/tasks
  end
end

class TaskWrite
  def create(attributes:)                       # POST /api/v1/tasks
  end

  def update(id:, attributes:)                  # PUT /api/v1/tasks/123
  end

  def delete(id:)                               # DELETE /api/v1/tasks/123
  end

  private def valid?(attributes:)
  end
end
```

Object Oriented Server Architecture

```
class Tasks
  def get(id:)
    Api(:aufgaben, :v1).get("tasks/#{id}").as(Task) do |task|
      Api(:exestenz, :v1).get("exists", list_id: task.list_id, user_id: user_id).success?
    end
  end
end
```

Deployment: wake

```
$ cd aufgaben

$ wake pack
+ some output while installing the app code or binary
dockersha

$ wake deploy --sha dockersha -n 6
+ creates hosts if necessary
+ launches 6 instances

$ wake count
6

$ wake expand -n 3
+ creates hosts if necessary
+ launches 3 instances
9

$ wake contract -n 3
+ terminates 3 instances (oldest first)
6
```

Deployment: wake

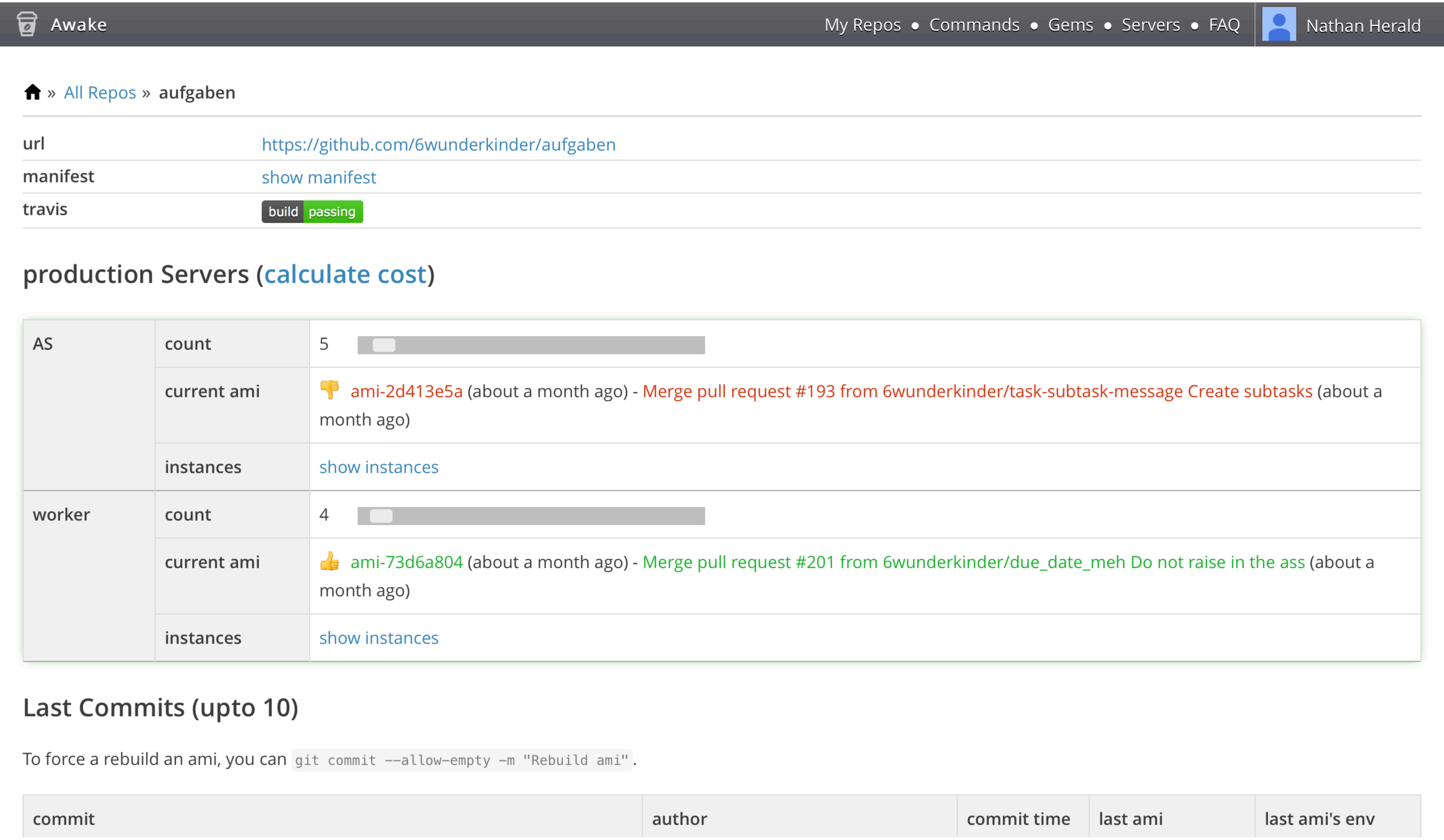
```
$ wake replace --sha dockersha
+ counts current instances
+ launches current amount
+ contracts

$ wake count
6

$ wake scale -n 12
+ counts current instances
+ if current value is more than 12, runs contract
+ else if current value is less than 12, runs expand
+ in this case, it would expand
```

Deployment: awake

- Github commit hook | build container
- Button to replace
- Button to scale



» All Repos » aufgaben

url	https://github.com/6wunderkinder/aufgaben
manifest	show manifest
travis	build passing

production Servers (calculate cost)

AS	count	5
	current ami	👎 ami-2d413e5a (about a month ago) - Merge pull request #193 from 6wunderkinder/task-subtask-message Create subtasks (about a month ago)
	instances	show instances
worker	count	4
	current ami	👍 ami-73d6a804 (about a month ago) - Merge pull request #201 from 6wunderkinder/due_date_meh Do not raise in the ass (about a month ago)
	instances	show instances

Last Commits (upto 10)

To force a rebuild an ami, you can `git commit --allow-empty -m "Rebuild ami"`.

commit	author	commit time	last ami	last ami's env

Example core service: aufgaben

```ruby
Aufgaben::Application.routes.draw do
  get '/api/health' => ->(env){
    [200, {"Content-Type" => "application/json"}, ['{"up":true}']]
  }

  namespace :api do
    namespace :v1 do
      resources :tasks
    end
  end
end
```

```ruby
class Api::V1::TasksController < ApplicationController
  before_filter :reject_conflicts, only: [:update, :destroy]

  def create
    attributes = Coor.create! attributes: create_params, client: current_client_info
    stats.increment :task, :create
    respond_with_created TaskRepresentation.new(task: attributes).to_hash
  end

  # ...
end
```

Task recently moved from aufgaben to coor.

```
class Task < ActiveRecord::Base
  attr_accessor :completed

  validates :list_id, presence: true
  validates :direct_owner_id, presence: true
  validates :title, presence: true, length: 1..255
  validates :created_by_request_id, uniqueness: true, allow_nil: true
  validate :do_not_allow_due_dates_very_far_in_the_future

  # ...
end
```

Example core service: tasks

```
GET     /api/v1/tasks                    controllers.Tasks.index
GET     /api/v1/tasks/:id                controllers.Tasks.show(id: Long)
POST    /api/v1/tasks                    controllers.Tasks.create
PATCH   /api/v1/tasks/:id                controllers.Tasks.update(id: Long)
PUT     /api/v1/tasks/:id                controllers.Tasks.update(id: Long)
DELETE  /api/v1/tasks/:id                controllers.Tasks.delete(id: Long, revision: Long)
```

```scala
trait TasksController extends Controller {
  def index = Authenticated.async { implicit req =>
    for {
      tasks <- fetchTasks
    } yield Ok(serializeTasks(tasks))
  }

  def show(id: Long) = Authenticated.async  { implicit req =>
    for {
      task <- fetchTask(id)
    } yield Ok(Json.toJson(task.write))
  }

  implicit val taskCreateReads = Json.reads[IncomingTaskCreateParams]

  def create = Authenticated.async(parse.json)  { implicit req =>
    for {
      params         <- parseBody(taskCreateReads.reads)
      _              <- hasPermissions(Some(params.listId), req.userId, false)
      outgoingParams = Some(outgoingCreateParams(params))
      task           <- Api("aufgaben", "v1").post("/tasks", outgoingParams).as[Task]
    } yield Created(taskWrites.writes(task.write))
  }

  // ...
}
```

Example stream service: webhooks

```ruby
Rails.application.routes.draw do
  namespace :api do
    namespace :v1 do
      resources :webhooks
    end
  end
end
```

Old version:

```ruby
def subscribe_and_work
  queue.bind(exchange, routing_key: "#").subscribe(ack: true, block: true) do |info, properties, payload|
    begin
      if work(payload, info.routing_key.split('.')[1]) == :retry
        queue.class.channel.reject info.delivery_tag, true # the last true is important, it means to re-enqueue
      else
        queue.class.channel.ack info.delivery_tag
      end
    rescue StandardError => e
      p e
    end
  end
end
```

```
class MutationProcessor extends Actor {
  import MutationProcessor._

  implicit val timeout = akka.util.Timeout(2, TimeUnit.SECONDS)
  import context.dispatcher
  import akka.pattern.pipe

  def receive = {
    case HandleListDeletion(mutation, webhook) =>
      println(s"[MutationHandler!HandeListDeletion] Received a delete-list mutation. Deleting webhook #${webhook.id}.")
      val deleteOperation = ApiCalls.deleteWebhook(webhook)
      pipe(deleteOperation.map(_ => Done).toFutureEither) to sender

    case ProcessMutation(listId, mutation, webhook) =>
      println(s"[MutationHandler!ProcessMutation] processing mutation for list_id $listId")
      val future: ApiFuture[Option[JsValue]] = processWebhook(mutation, webhook)
      pipe(future.toFutureEither) to sender
  }

  def processWebhook(mutation: Mutation, webhook: Webhook): ApiFuture[Option[JsValue]] = {
    new Processor(mutation).process(webhook)
  }
}
```

Polyglot Development

Why?

Challenges?

"What about shared libraries?"

No

Service vs Library

Logging

stdout | syslog | rsyslog cluster

Service vs Library

Metrics

statsd | librato

Serialization

Migrating from a ruby gem to a ruby service for mutations

```
class TaskRepresentation
  include Rep

  initialize_with :task

  fields [
    :id,
    :assignee_id,
    :completed,
    :completed_at,
    :completed_by_id,
    :created_at,
    :created_by_id,
    :created_by_request_id,
    :recurrence_type,
    # ...
  ] => :default
end
```

Conventions

- Version in url (`/api/v1/tasks`)
- Flat routes (no regexps, no nesting, use query params)
- Shared `api-client` for HMAC, discovery
- Shared `api-controller` for permissions, `null` removal
- Every change creates a mutation object
- Shared service for writes for emitting mutations

Takeaway

Over half of our microservices have been updated or rewritten since we launched, all without major interruption or even a launch party.

Questions?