Service-Oriented Architecture

a practical introduction to the service-oriented mind set

Thorben Schröder
FH Salzburg 2014

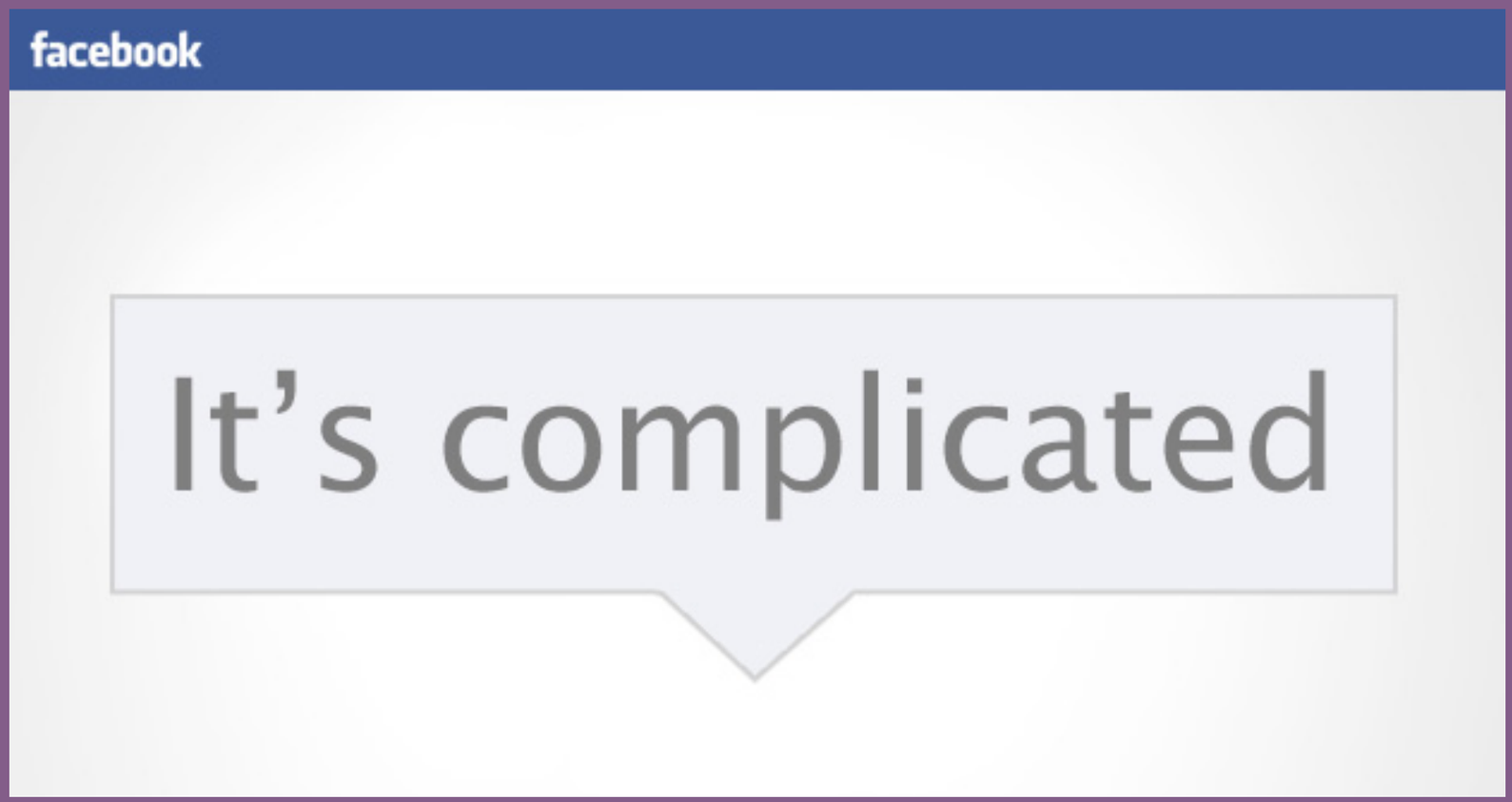
Hi there

I am Thorben.

Reach me at info@thorbenschroeder.de

Or find me on Twitter, GitHub & Co as @walski.

What I do



What I do

so coded

Universität Bremen

Quarter Spiral

KOPF MASCHINE

Engine Yard™

WunderCar

WHAT DO YOU DO?

Good to know

- Basics
  - git, (a little) UNIX, vocabulary (deployments, …)
- Ruby (≥ 2.0)
- Ruby ecosystem
  - RubyGems, bundler, RVM
- Test driven methodology

AGENDA

1. Intro to SOA
2. Modular Web Applications
3. Documentation
4. Testing
5. Development Setups
6. Service Authentication
7. Inter Service Communication
8. Deployments
9. Browsers as Clients

Services!

What it's not

- monolithic applications

What it's like

- UNIX philosophy:
  "Make each program do one thing well."

- Cluster of communicating applications

History

- term coined by 1996
- got popular with the web as it's foundation by around 2001
- not very well defined

WHY?

- Applications' requirements keep growing
- Big applications are hard to maintain & debug
- Splitting up helps to focus!
- Bonus points: An internal API can be used externally at very little extra cost

FOUR TENETS OF SOA

Don Box of Microsoft:

1. Boundaries are explicit
2. Services are autonomous
3. Services share schema and contract, not class
4. Service compatibility is based on policy

DOESN'T SAY MUCH...

BUT

My Little Application

Shop

Billing

Warehouse

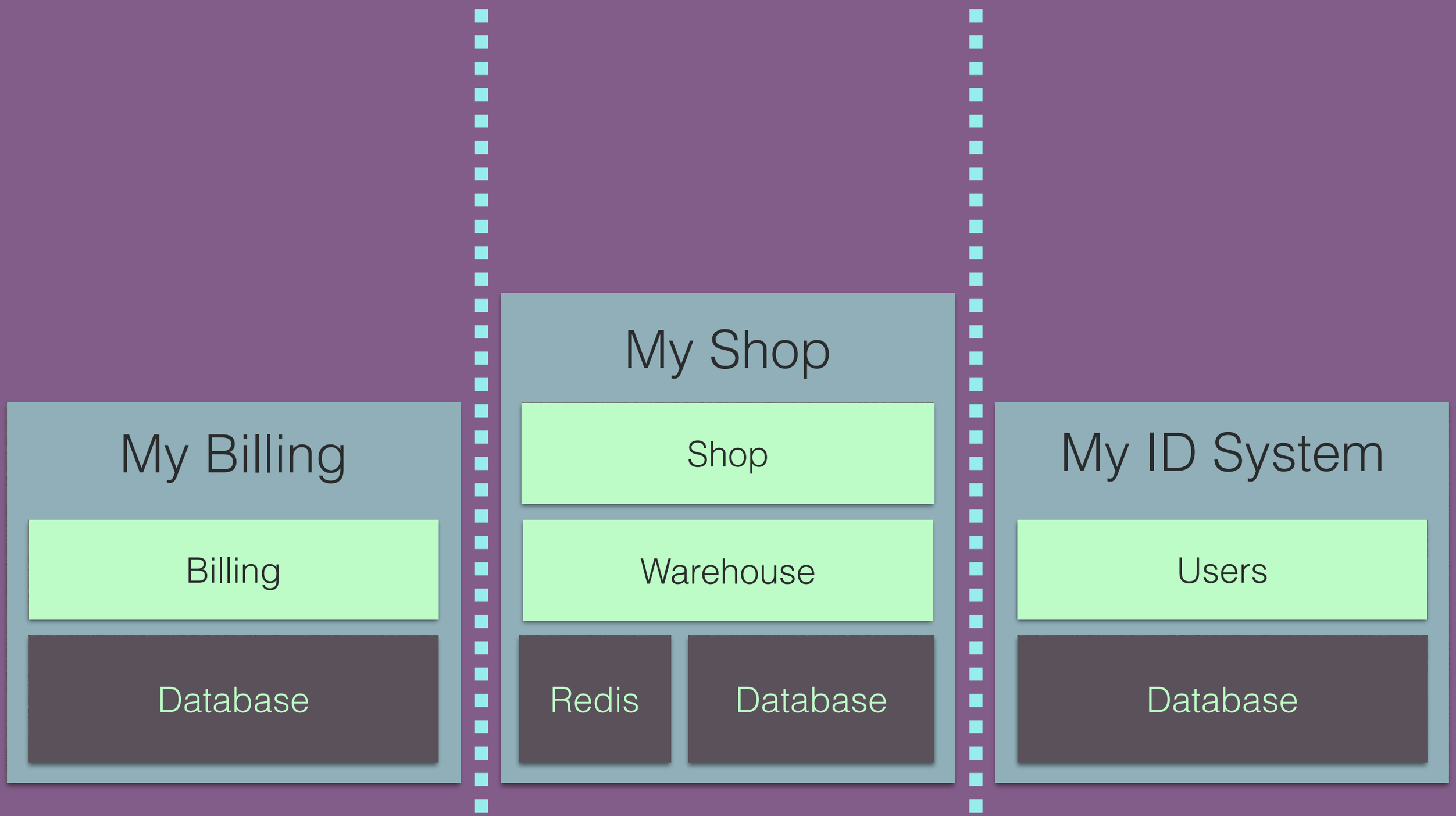
Users

Redis

Database


My Billing
Billing
Database
My Shop
Shop
Warehouse
Redis
Database
My ID System
Users
Database

1. Boundaries are explicit



2. Services are autonomous

Just an invoice system

My Billing

Billing

Database

Just a shop

Shop

Warehouse

Redis

Database

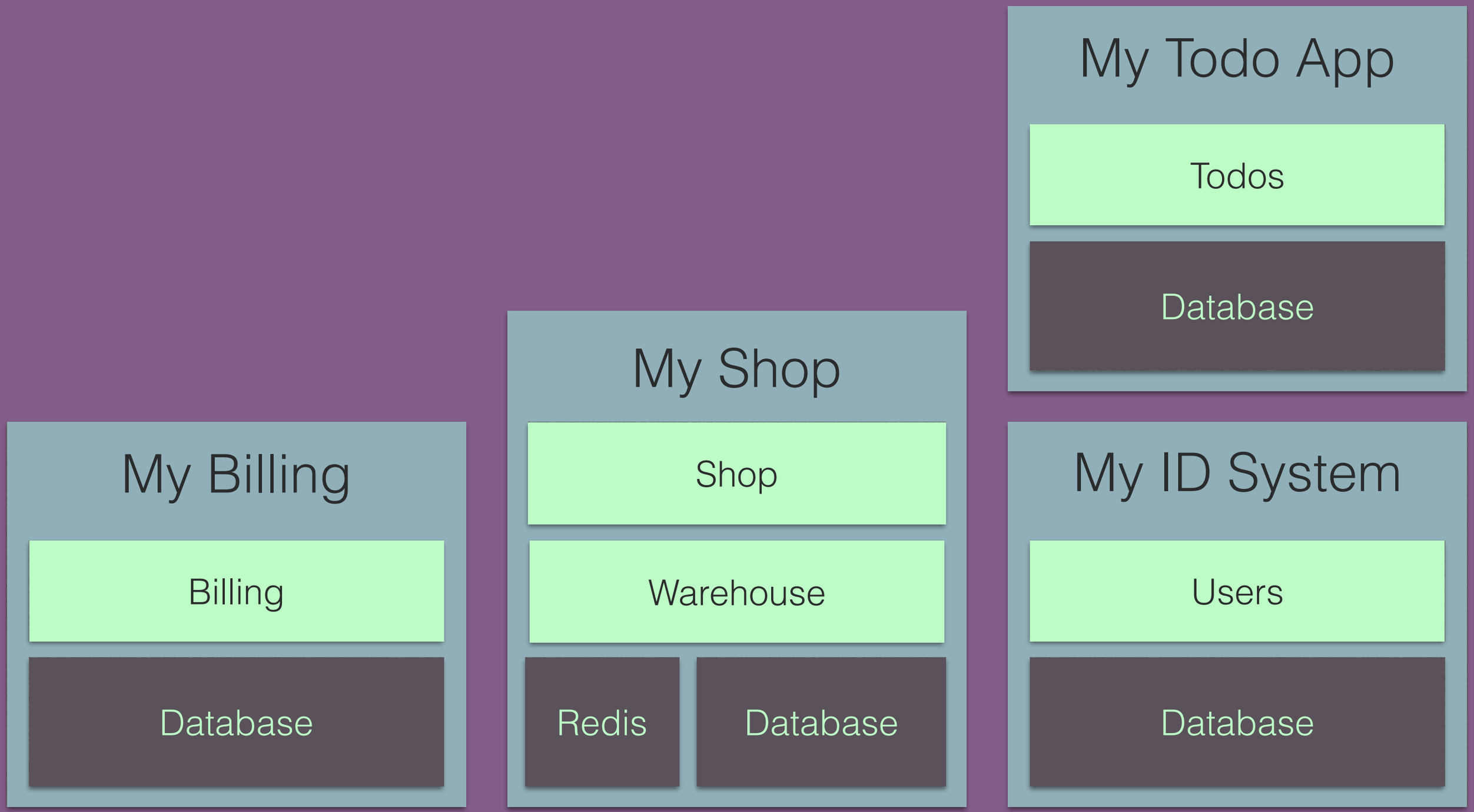
Just an authorization provider

My ID System

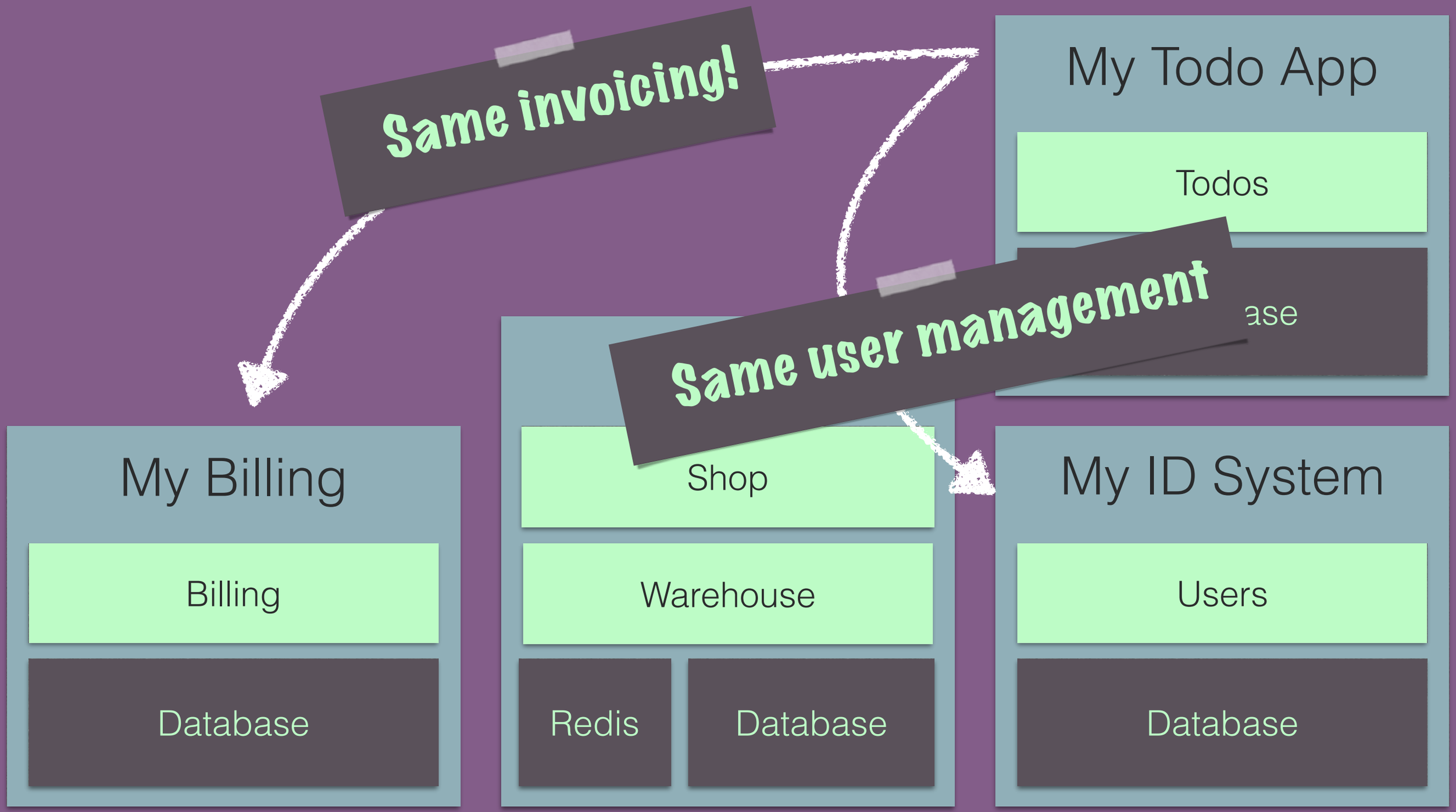
Users

Database

2. Services are autonomous



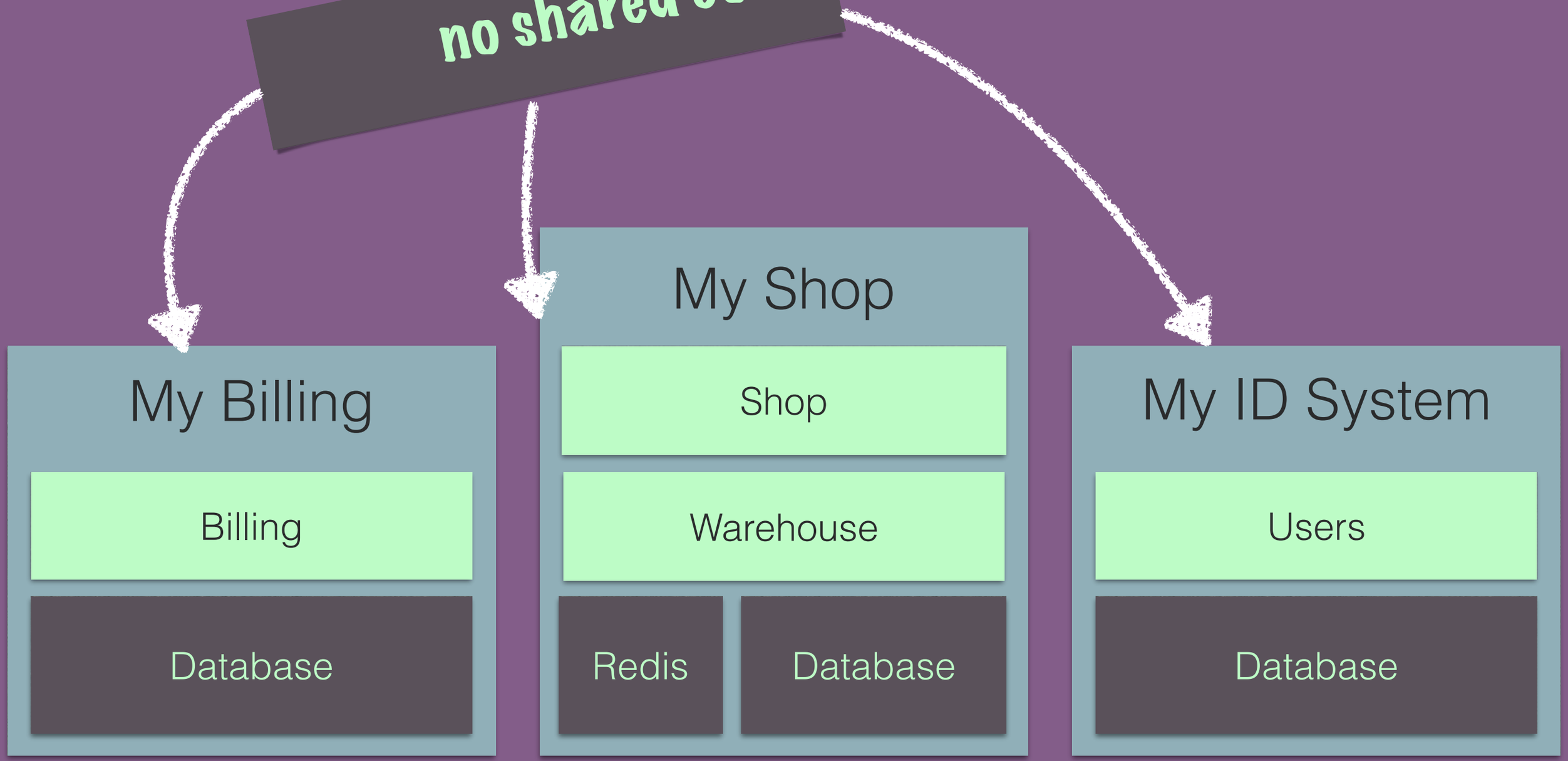
2. Services are autonomous



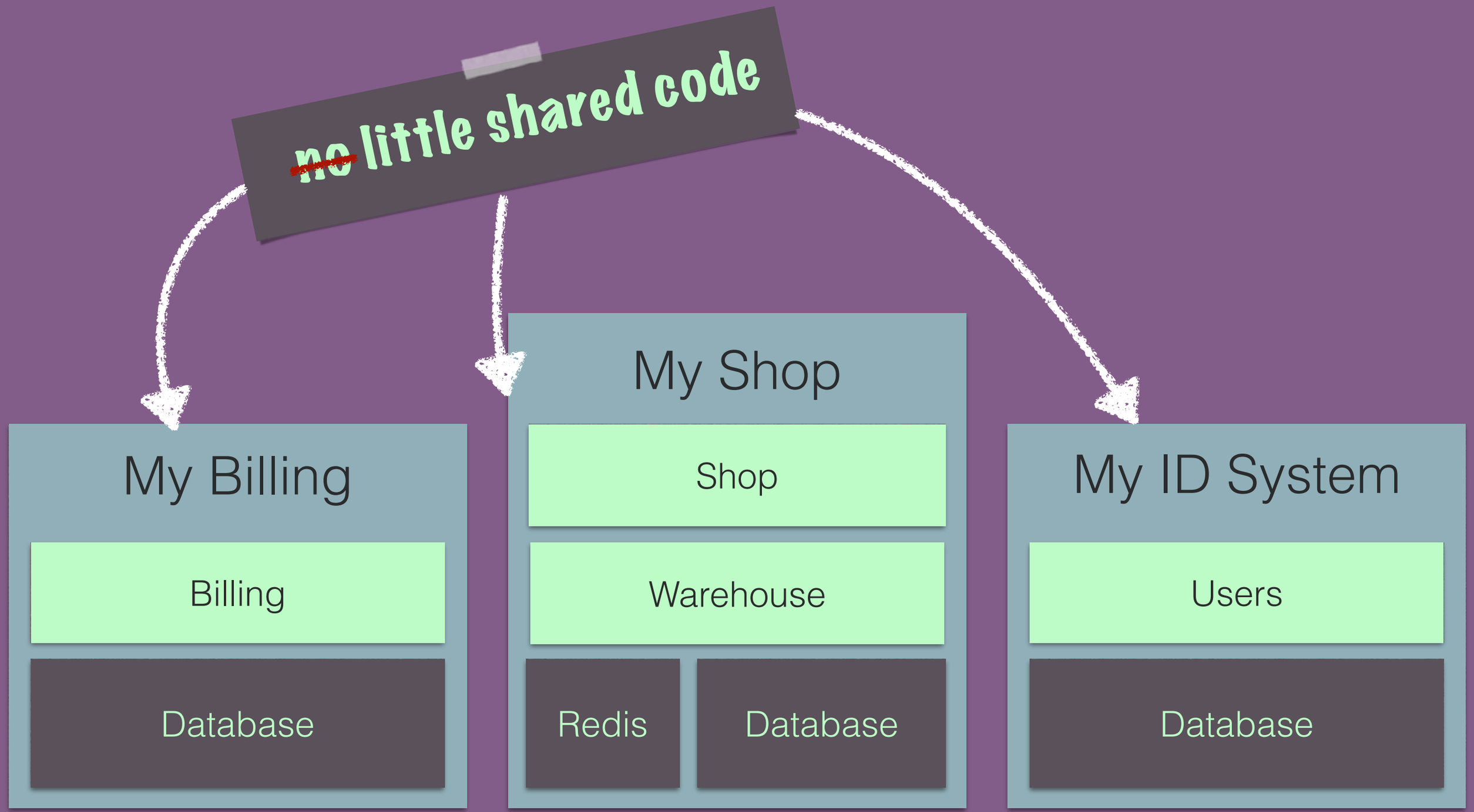
3. Services share schema and contract, not class

3. Services share schema and contract, not class

no shared code

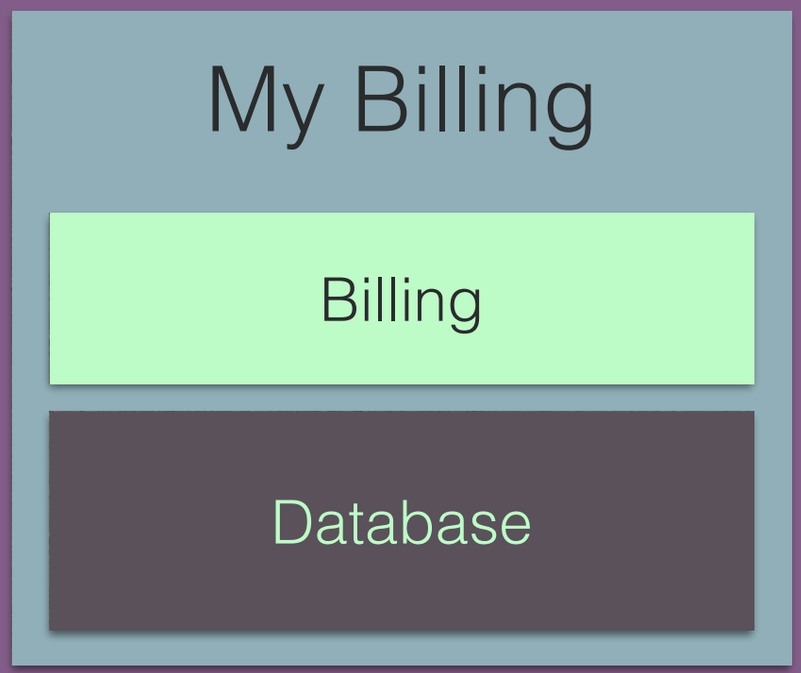


3. Services share schema and contract, not class

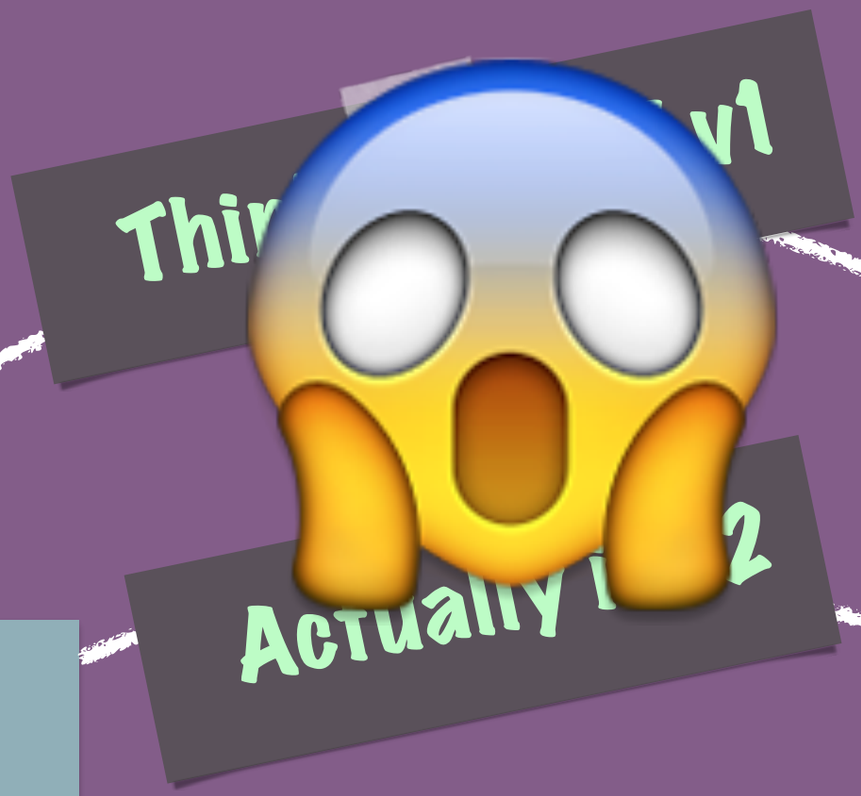


4. Service compatibility is based on policy

4. Service compatibility is based on policy



4. Service compatibility is based on policy



My Billing

Billing

Database

My Shop

Shop

Warehouse

Redis

Database

FOUR TENETS OF SOA

Thorben's version:

1. Think in autonomous apps
2. Make them communicate by messages only
3. Integration test everything

Modular

Web Applications

The Twelve-Factor App

http://12factor.net/

The Twelve-Factor App

- "a methodology for building software-as-a-service apps"

- made for "cloud" deployments
- scalability in mind
- very practical approach to SOA development

The Twelve-Factor App

I. Codebase

II. Dependencies

III. Config

IV. Backing Services

V. Build, release, run

VI. Processes

VII. Port binding

VIII. Concurrency

IX. Disposability

X. Dev/prod parity

XI. Logs

XII. Admin processes

The Twelve-Factor App

I. Codebase

Use git

II. Dependencies

III. Config

IV. Backing Services

V. Build, release, run

VI. Processes

VII. Port binding

VIII. Concurrency

IX. Disposability

X. Dev/prod parity

XI. Logs

XII.

The Twelve-Factor App

I. Codebase

II. Dependencies

III. Config

IV. Backing Services

V. Build, release, run

VI. Processes

VII. Port binding

Use bundler

ncurrency

IX. Disposability

X. Dev/prod parity

XI. Logs

XII.

The Twelve-Factor App

I. Codebase

II. Dependencies

III. Config

IV. Backing Services

V. Build, release, run

VI. Processes

VII. Port binding

VIII. Concurrency

sposability

X. Dev/prod parity

XI. Logs

XII.

Configuration is part of the environment

The Twelve-Factor App

I. Codebase

II. Dependencies

III. Config

IV. Backing Services

V. Build, release, run

VI. Processes

VII. Port binding

VIII. Concurrency

IX. Disposability

X. Dev/prod parity

XI. Logs

XII.

Make services configurable

The Twelve-Factor App

I. Codebase

II. Dependencies

III. Config

IV. Backing Services

V. Build, release, run

VI. Processes

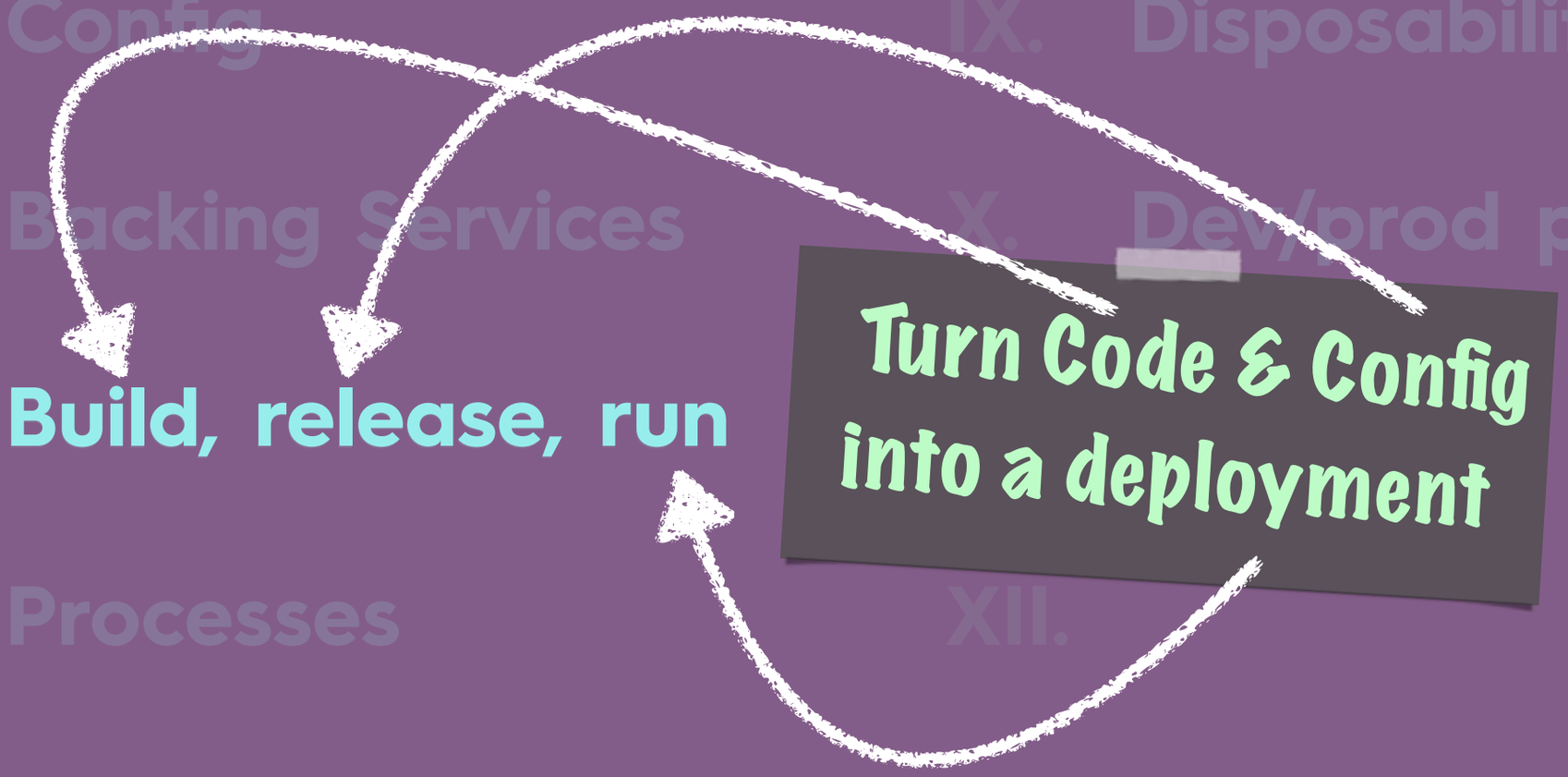
VII. Port binding

VIII. Concurrency

IX. Disposability

X. Dev/prod parity

XII.



The Twelve-Factor App

I. Codebase

II. Dependencies

III. Config

IV. Backing Services

V. Build, releas

VI. Processes

VII. Port binding

VIII. Concurrency

IX. Disposability

X. Dev/prod parity

Processes are stateless

Any "state" belongs in a backing service

The Twelve-Factor App

Apps are self-contained

Communicate over a configurable port

I. C

II. D

III. Config

IV. Backing Services

V. Build, release, run

VI. Processes

IX. Port binding

II. Concurrency

IX. Disposability

X. Dev/prod parity

XI. Logs

XII.

The Twelve-Factor App

I. Codebase

II. Dep

III. Concurrency

Scale by processes
Define process types

III. Config

IV. Backing Services

V. Build, release, run

VI. Processes

VII. Port binding

IX. Disposability

X. Dev/prod parity

XI. Logs

XII.

The Twelve-Factor App

I. Codebase

II. Depend

III. Config

IV. Backing Services

V. Build, release, run

VI. Processes

VII. Port binding

VIII. Concurrency

Quick startup
Graceful shutdown

IX. Disposability

X. Dev/prod parity

XI. Logs

XII.

The Twelve-Factor App

I. Codebase

II. Dependencies

III. C

IV. D

V. Build, release, run

VI. Processes

VII. Port binding

VIII. Concurrency

IX. Disposability

Dev/prod parity

Logs

XII.

Development == Production
Same backing services
Continuous deployments

The Twelve-Factor App

I. Codebase

II. Dependencies

III. Config

IV. Backing Services

V. Build, rele

VI. Processes

VII. Port binding

VIII. Concurrency

IX. Disposability

X. Dev/prod parity

Log to STDOUT

XI. Logs

XII.

The Twelve-Factor App

I. Codebase

II. Dependencies

III. Config

IV. Backing

V. Build, r

VI. Proces

VII. Port binding

VIII. Concurrency

IX. Disposability

Dev/prod parity

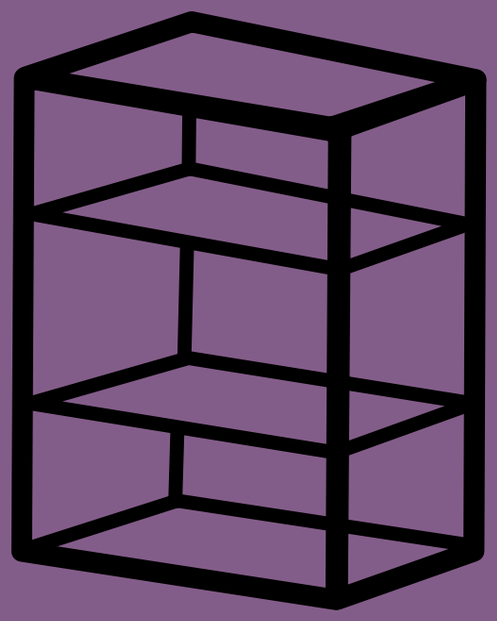
Logs

Admin processes

Same execution as app's processes

REPL should be part of the app

A Rack 101



rack

powers web applications

http://rack.github.io/

What is Rack?

Rack provides a minimal interface between web servers supporting Ruby and Ruby frameworks.

What is Rack?

- Rack Application
- Rack Middleware
- That is it

What is Rack?

- Hash in
- Array out

What is Rack?

- Hash in
- Array out

Rack environment

```
{
  'REQUEST_METHOD' => 'POST',
  'PATH_INFO' => '/api/v1/todos',
  #…
  'HTTP_AUTHORIZATION' => 'Basic bGFzdDp1bmljb3Ju',
}
```

What is Rack?

HTTP Header

Rack environment

```
{
    'REQUEST_METHOD' => 'POST',
    'PATH_INFO' => '/api/v1/todos',
    #...
    'HTTP_AUTHORIZATION' => 'Basic bGFzdDp1bmljb3Ju',
}
```

WHAT IS RACK?

- Hash in

Iterable to allow streaming

- Array out

Rack response

```
[200, {“Content-Type" => "text/plain"}, ['Body']]
```

HTTP Status

HTTP Response Header

Response Body

WHAT IS RACK?

- Hash in

Iterable to allow streaming

- Array out

Rack response

```
[200, {“Content-Type" => "text/plain"}, ['Body']]
```

HTTP Status

HTTP Response Header

Response Body

A Rack App

- must respond to “call” with the Rack environment as the only argument

A Rack App

Rack environment

```
class MyApp
  def call(env)
    [200, {'Content-Type' => 'text/plain'}, ['Hey there']]
  end
end
```

Rack response array

BUT WAIT...

CALL YOU SAID?

```
$ ri Proc#call

= Proc#call

(from ruby core)
------------------------------------------------------------
  prc.call(params,...)   -> obj
  prc[params,...]        -> obj
  prc.(params,...)       -> obj


------------------------------------------------------------

Invokes the block, setting the block's parameters to the
values in params using something close to method calling
semantics.…
```

PROCS ARE RACK APPS!

Also a Rack App

Rack environment

```
-> (env) { [200, {'Content-Type' => 'text/plain'}, ['…']] }
```

Rack response array

CHALLENGE ACCEPTED

Setting up our lab

```
$ git clone https://github.com/walski/soa-workshop.git
$ cd soa-workshop/box
$ vagrant up
$ vagrant ssh

vagrant@…$ cd /vagrant/challenges/01
vagrant@…$ bundle exec rake
```

Setting up our lab

- edit code on your machine (box/challenges)
- run the specs in Vagrant

CHALLENGE ACCEPTED

RACK'S MIGHTY FRIENDS

Rack::Request

Rack::Response

Rack::Builder

```ruby
class MyAppA
  def call(env)
    [200, {'Content-Type' => 'text/plain'}, ['Hey A']]
  end
end

class MyAppB
  def call(env)
    [200, {'Content-Type' => 'text/plain'}, ['Hey B']]
  end
end

app = Rack::Builder.new {
   map("/a") { run MyAppA }
   map("/b") { run MyAppB }
   [200, {'Content-Type' => 'text/plain'}, ['What?']]
}
```

RACKUP

1. Create a rackup config

CONFIG.RU

```
Bundler.require
require 'myapp'

run MyApp.new
```

2. Run your web app

```
$ rackup
```

RTFM

rack.rubyforge.org/doc/README_rdoc.html

CHALLENGE ACCEPTED

A Sinatra Primer

What is Sinatra?

- simple web framework
- routing / templates / “all we need”
- Sinatra apps are Rack apps
- good example

What is Sinatra?

- simple web framework
- routing / templates / “all we need”
- Sinatra apps are Rack apps
- good example

What is Sinatra?

```
require 'sinatra/base'

class MyApp < Sinatra::Base
  get '/hossa' do
    'cool!'
  end
end
```

Sinatra can do so much more.

But seriously...

RTFM

www.sinatrarb.com/intro.html

CHALLENGE ACCEPTED

A Rack middleware

- is a wrapper around an app
- gets the app passed as the only argument to it's constructor
- must respond to "call" with the Rack environment as the only argument

A Rack middleware

Parent Rack app

```ruby
class MyMiddleware
  def initialize(app)
    @app = app
  end

  def call(env)
    if env['PATH_INFO'] = '/yay'
      [200, {'Content-Type' => 'text/plain'}, ['yay']]
      return
    end
    @app.call(env)
  end
end
```

CHALLENGE ACCEPTED

I hate Ruby!

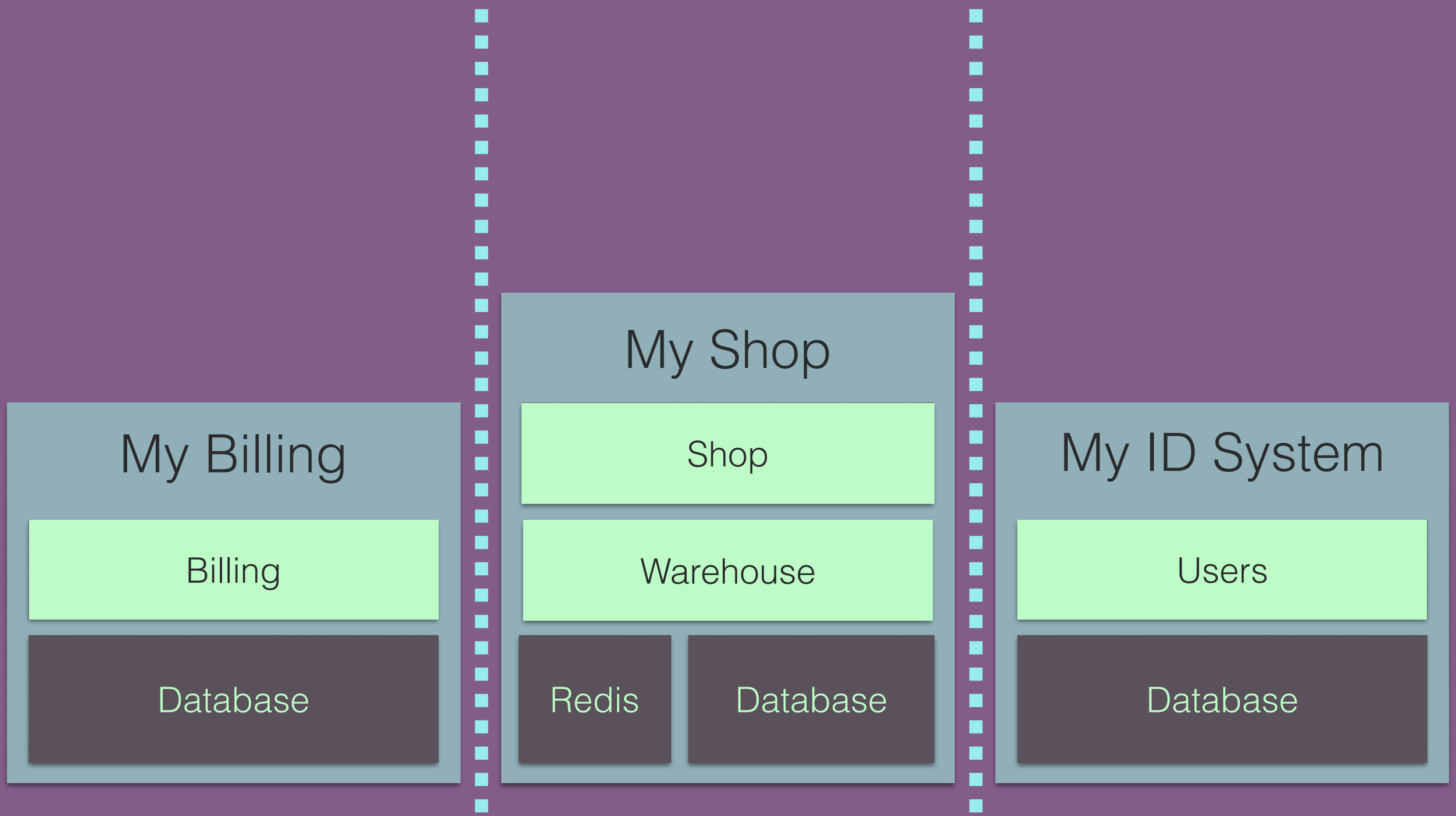
- Don't you worry:
  - Python WSGI
  - Perl Web Server Gateway Interface
  - JavaScript Gateway Interface
  - ...

SOA

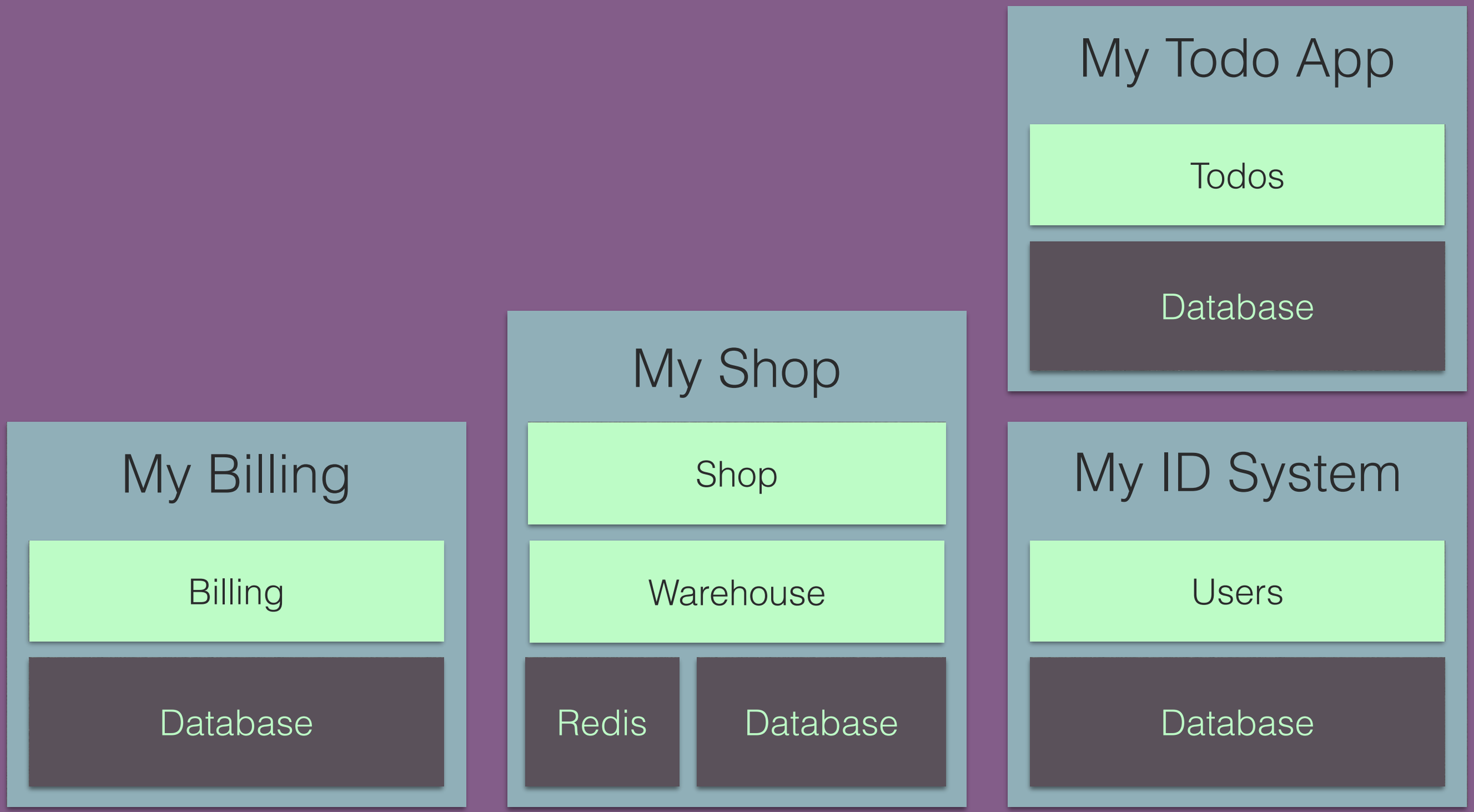
Documentation

QUICK RECAP

1. Boundaries are explicit



2. Services are autonomous



3. Services share schema and contract, not class

4. Service compatibility is based on

4. Service

compatibility is

based on policy

4. Service Compatibility is Based on Policy

Document your APIs

- APIs are your policy / contract
- Helps to frame your mindset

Document your APIs

- APIs are your policy / contract
- Helps to frame your mindset

Readme Driven Development

- Write your Readme before writing code
- Before ANY code!
- Forces you to think what you are about to build
- Enables team discussions

README.md in SOA

- The Readme can be / include your API documentation
- Good for cross functional teams

What needs to be documented?

- Take advantage of the simplicity of HTTP:
  - Request
  - Response

What needs to be documented?

- Request
  - Verb
  - Path
  - Headers
  - Body

What needs to be documented?

- Response
  - Status
  - Headers
  - Body
- One documented request might lead to several different responses depending on the request specifics (unauthenticated vs. authenticated requests...)

What needs to be documented?

- Miscellaneous
- Versioning
- Authentication
- Pagination
- ...

SAAAAAAMPLES

developer.github.com/v3/

SAAAAAAMPLES

https://devcenter.heroku.com/articles/platform-api-reference

How to document?

- Write it yourself
- Generated from code / comments
- Generated from tests / specs

How to document?

- Write it yourself
- Generated from code / comments
- Generated from tests / specs

No documentation challenge :/

CHALLENGE DENIED.

SOA Testing

WHY IS THIS EVEN A THING?

Integration Tests

SOA apps can't

They rely on the integration with each other!

vs.

Monolithic apps can be tested with unit tests

Unit Tests

Your Options

1. Have a mock for every single service
2. Find a way to run the real systems

Manage Dependencies

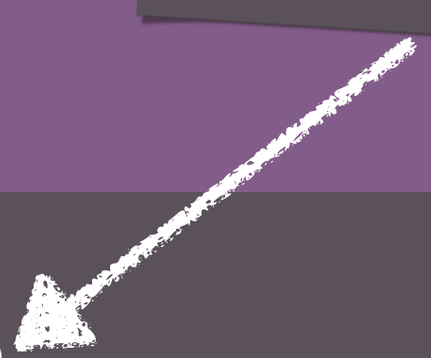
Each app is a gem

Creating Gems

```
$ bundle gem my_gem
      create  my_gem/Gemfile
      create  my_gem/Rakefile
      create  my_gem/LICENSE.txt
      create  my_gem/README.md
      create  my_gem/.gitignore
      create  my_gem/hello-world.gemspec
      create  my_gem/lib/my_gem.rb
      create  my_gem/lib/my_gem/version.rb
```

Creating Gems

Pretty much empty



```
$ bundle gem my_gem
      create  my_gem/Gemfile
      create  my_gem/Rakefile
      create  my_gem/LICENSE.txt
      create  my_gem/README.md
      create  my_gem/.gitignore
      create  my_gem/hello-world.gemspec
      create  my_gem/lib/my_gem.rb
      create  my_gem/lib/my_gem/version.rb
```

Creating Gems

Place to track your dependencies, edit name, description, etc...

```
$ bundle gem my_gem
      create  my_gem/Gemfile
      create  my_gem/Rakefile
      create  my_gem/LICENSE.txt
      create  my_gem/README.md
      create  my_gem/.gitignore
      create  my_gem/hello-world.gemspec
      create  my_gem/lib/my_gem.rb
      create  my_gem/lib/my_gem/version.rb
```

Creating Gems

```
Gem::Specification.new do |spec|
  …
  spec.add_development_dependency "bundler", "~> 1.3"
  spec.add_development_dependency "rake"
end
```

Use spec.add_dependency for runtime dependencies