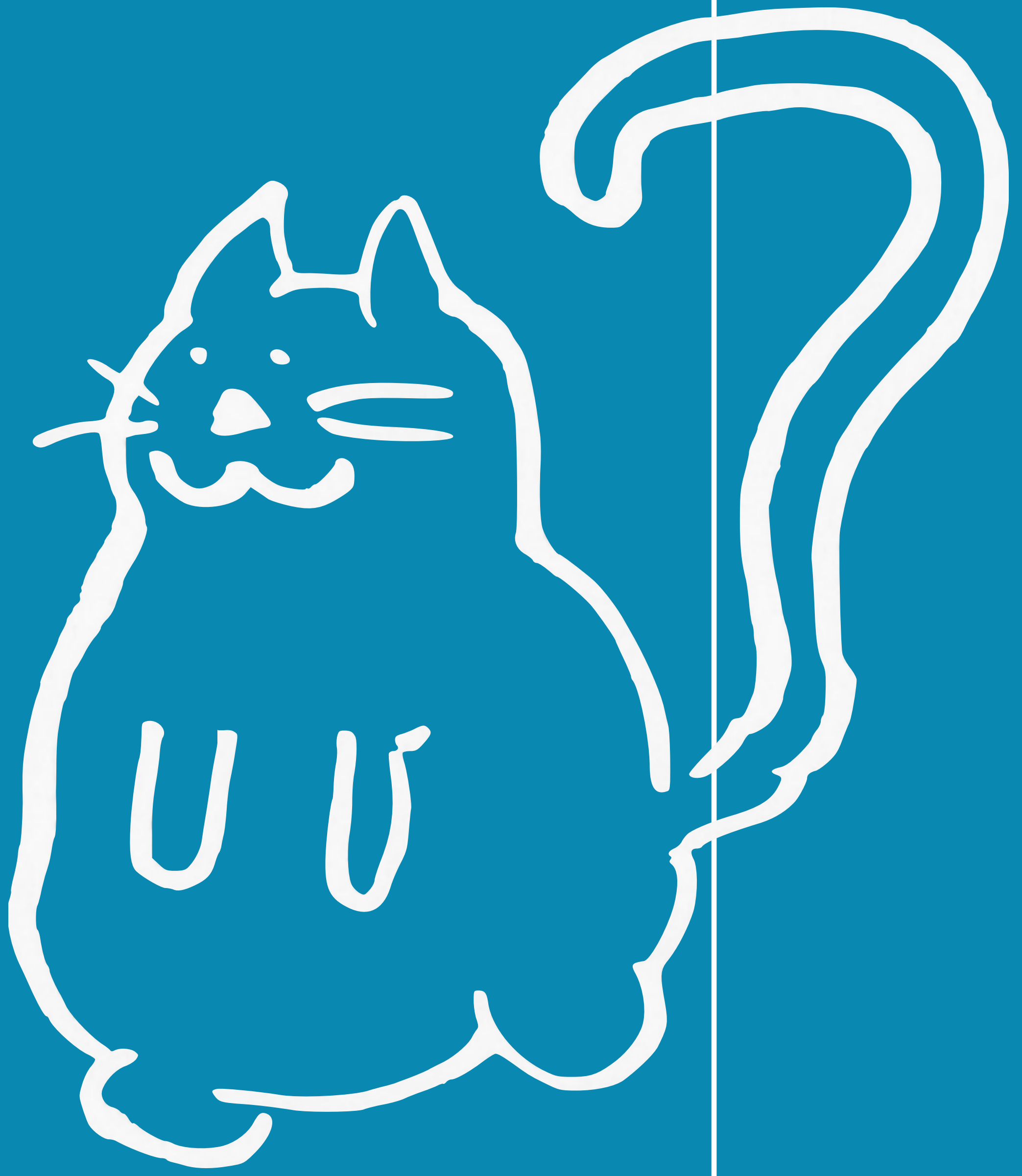
IBM

Java EE Microservices by Example: from Raspberry Pis to the Cloud

Holly Cummins
September 2016
@holly_cummins



http://ibm.biz/bluemixgaragelondon

http://ibm.biz/bluemixgaragelondon

IBM Bluemix™

Microservices are
guaranteed bug-free.

*Microservices make your
colleagues less annoying.*

Microservices: Good
design built-in!

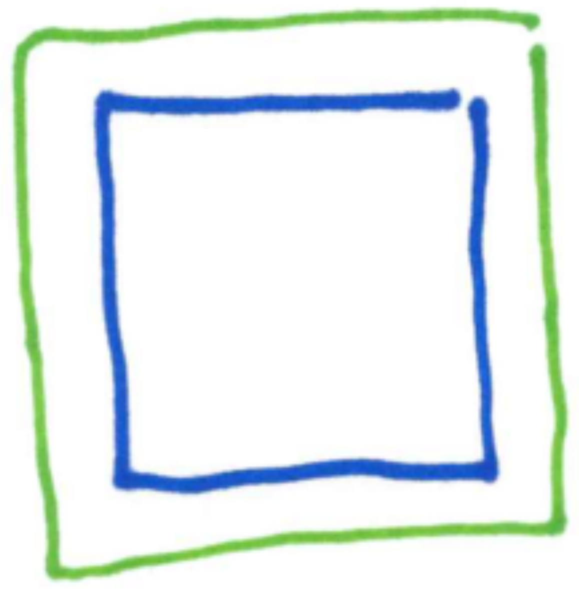
Kittens love
microservices.

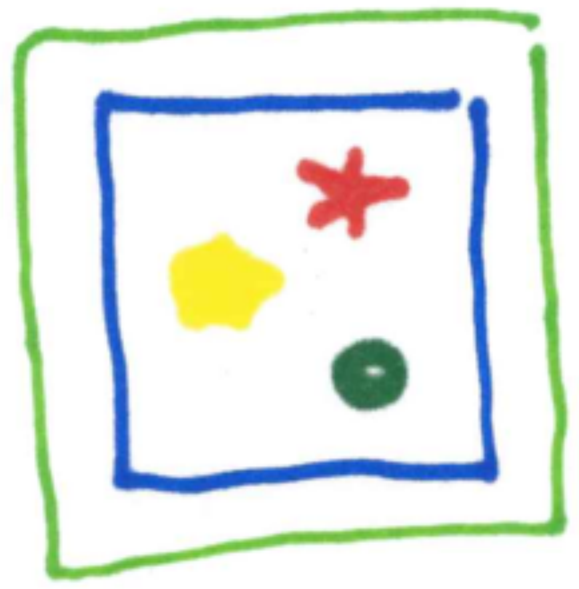
Microservices vaporize
unclean code.

Microservices. The best
thing since sliced bread.

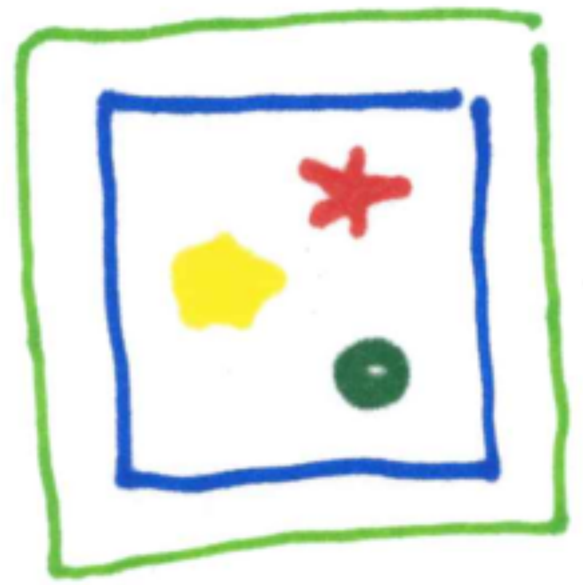
Every microservice
comes with a free puppy.

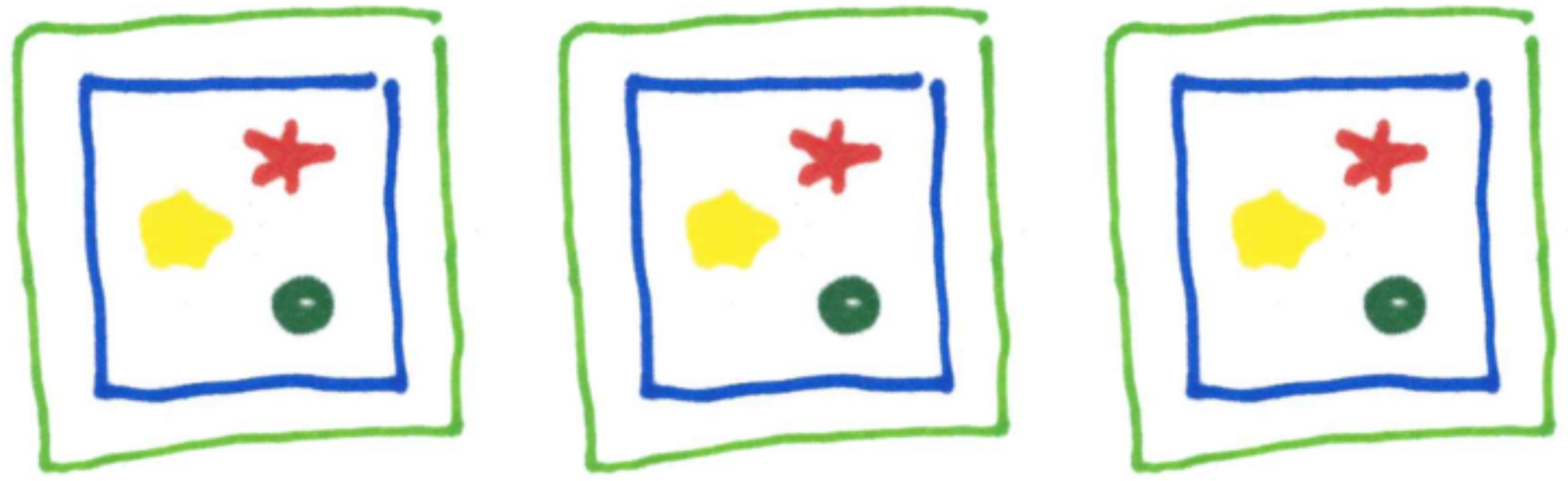
Wait. What problem are we *actually* trying to solve?



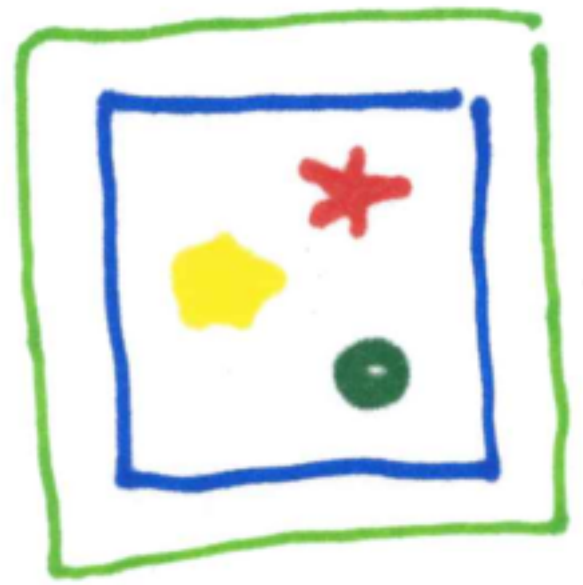


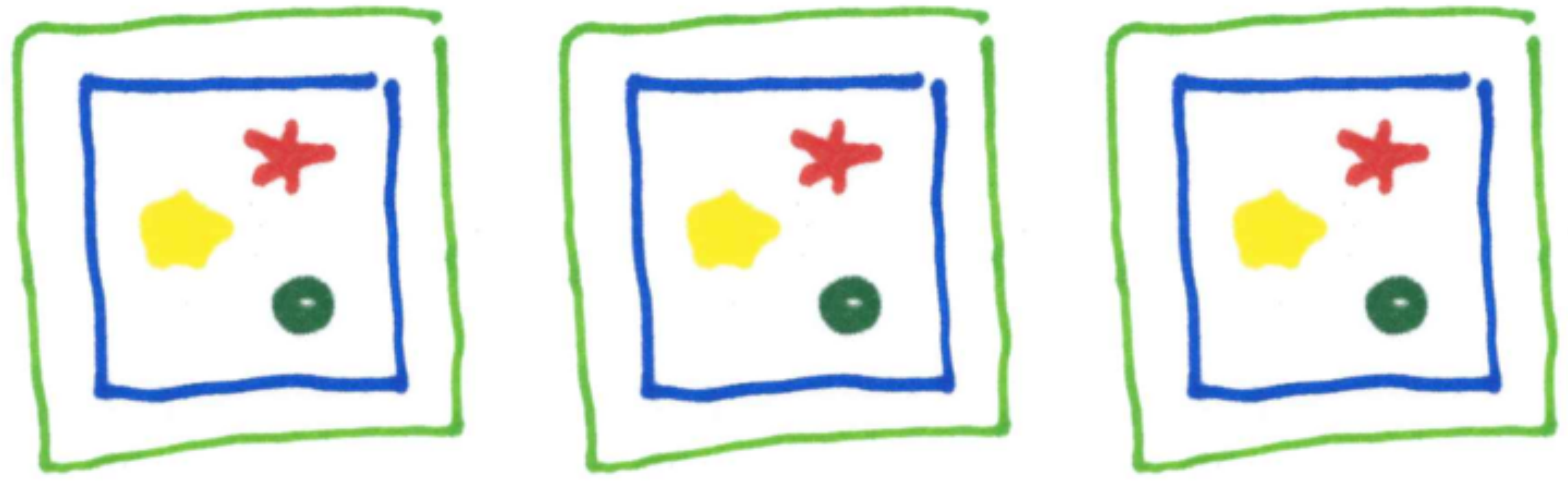
Monolithic Modularity



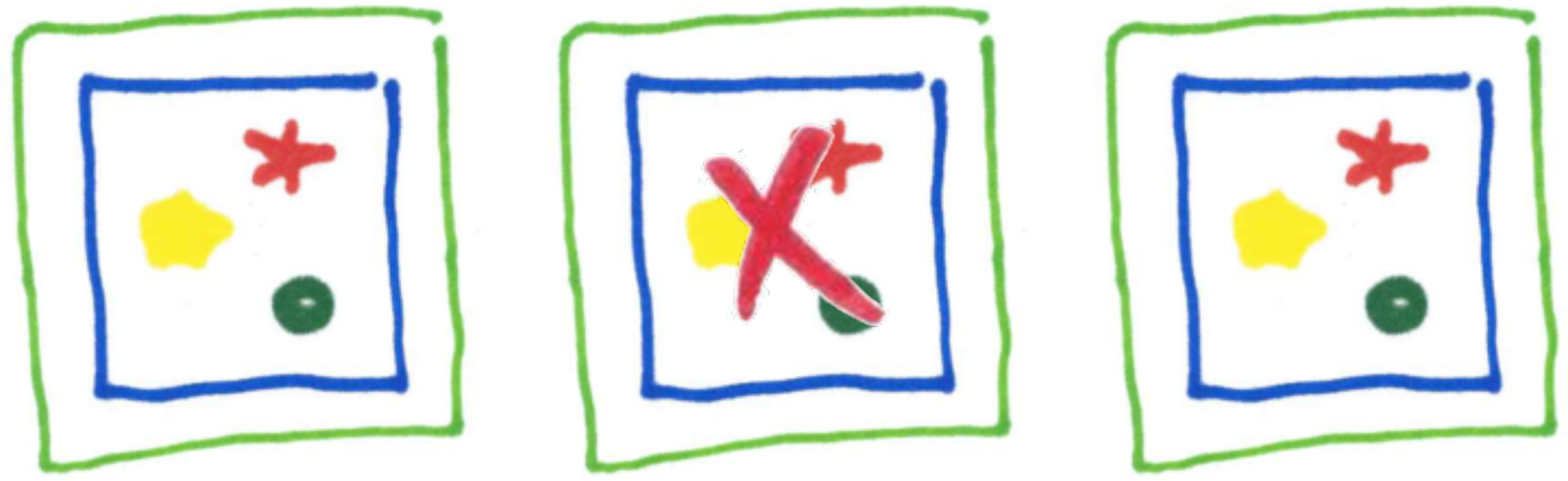


Monolithic Scaling

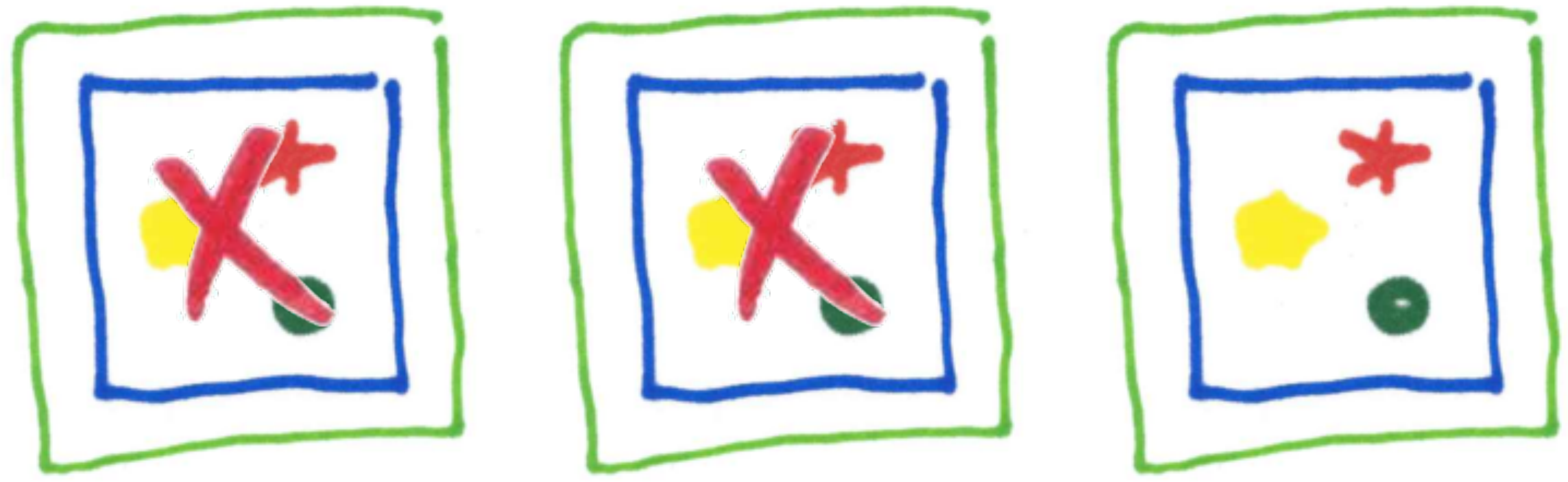




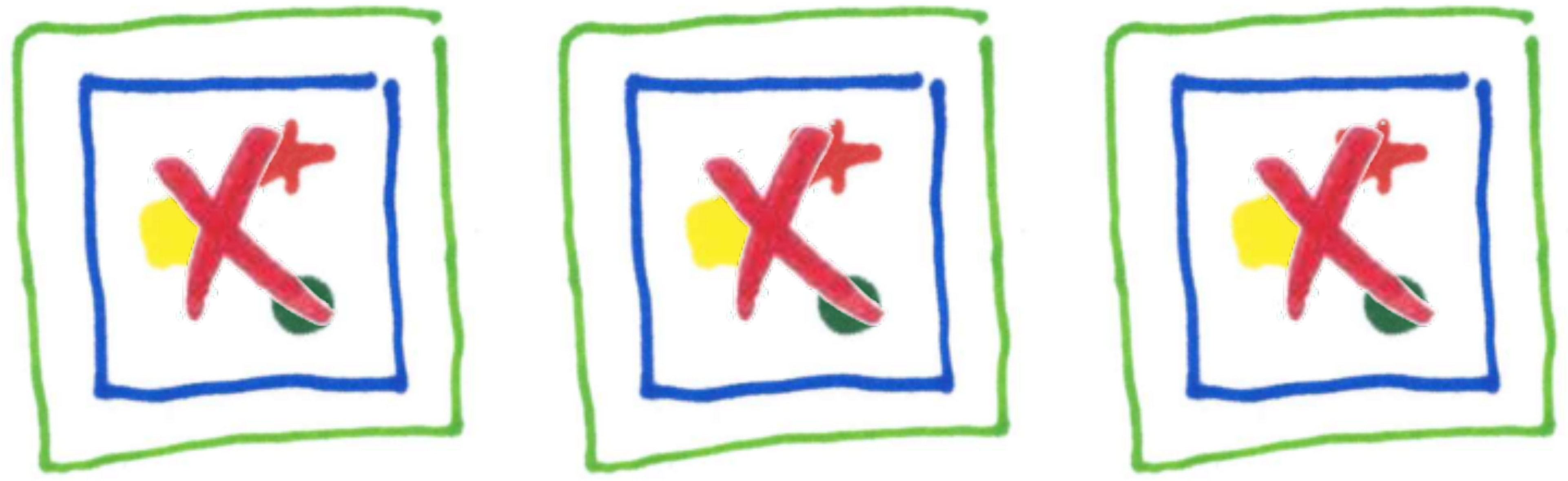
Monolithic Failing



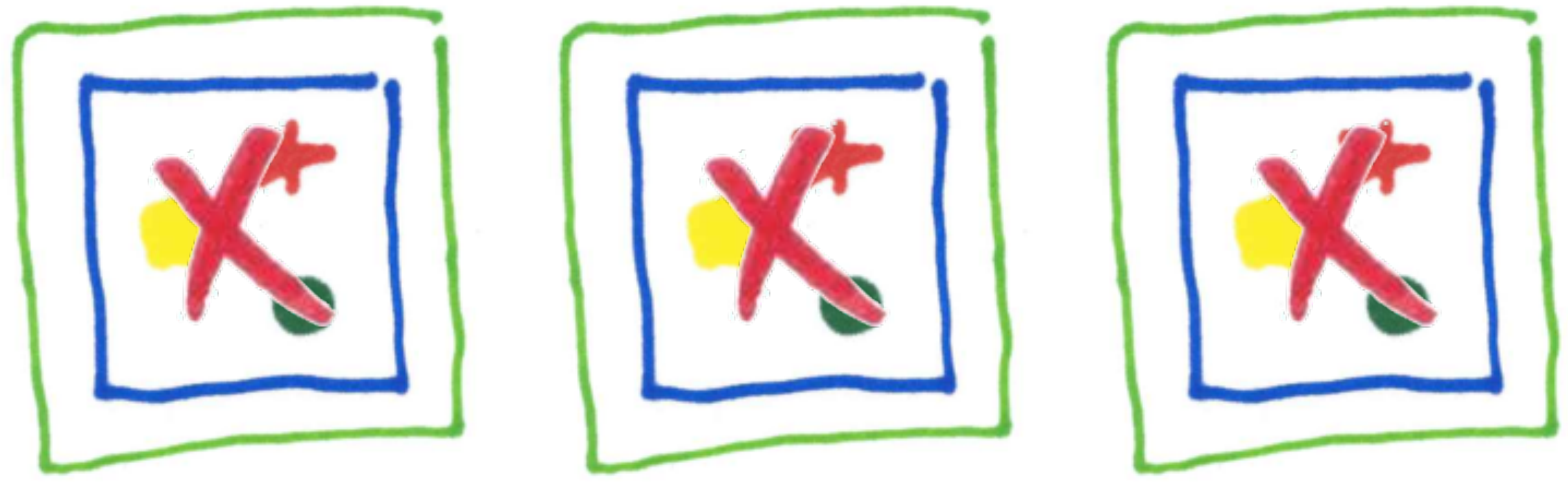
Monolithic Failing



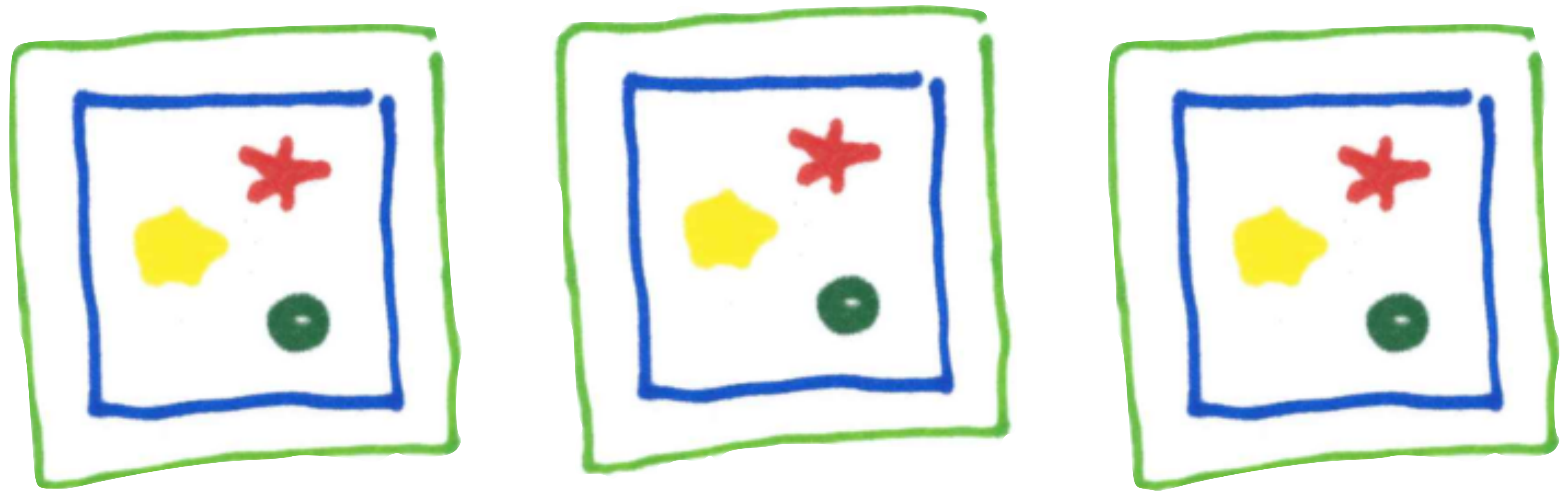
Monolithic Failing



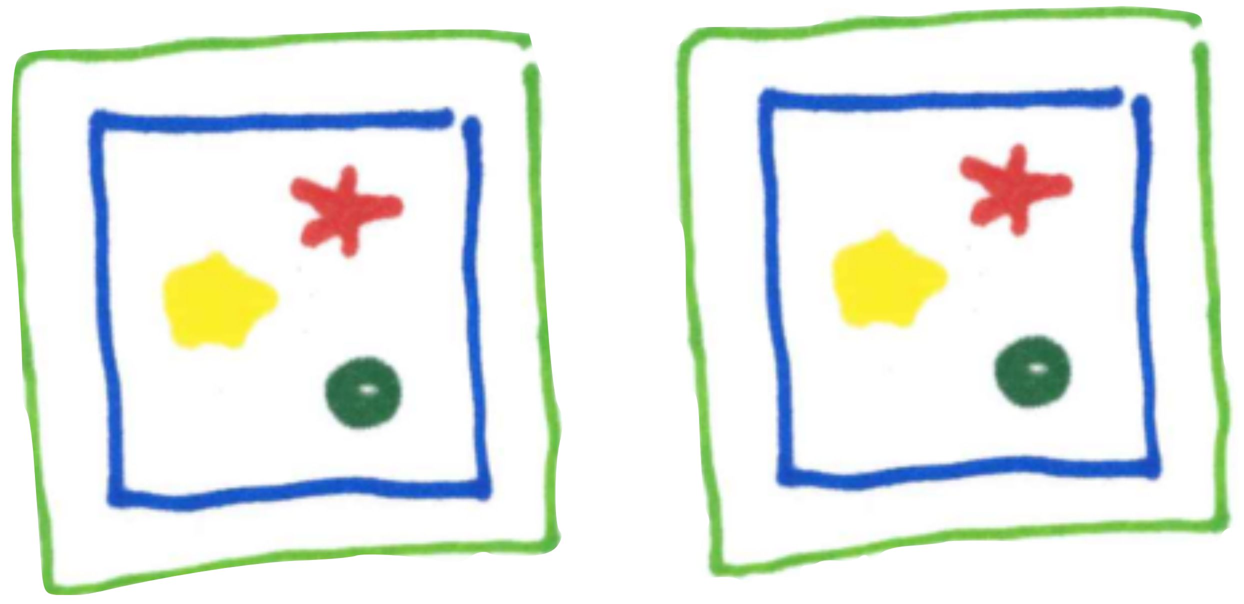
Monolithic Failing



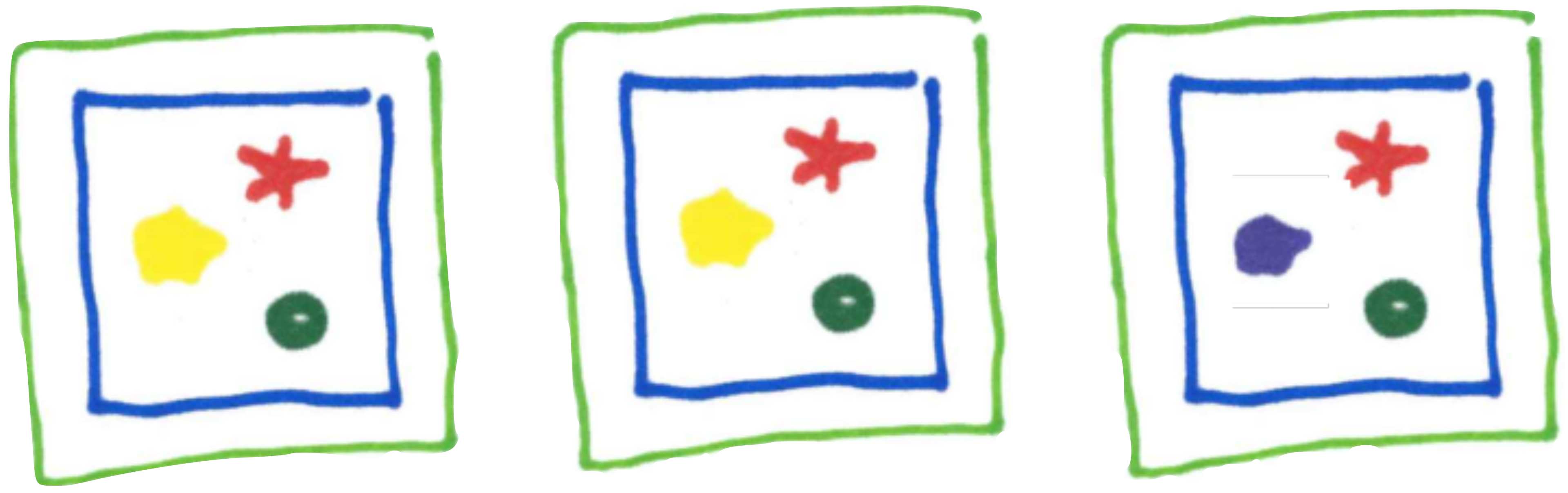
Monolithic Failure



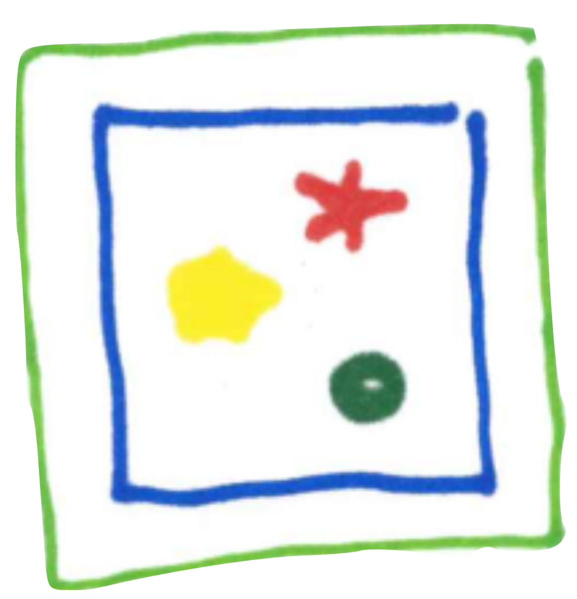
Monolithic Update

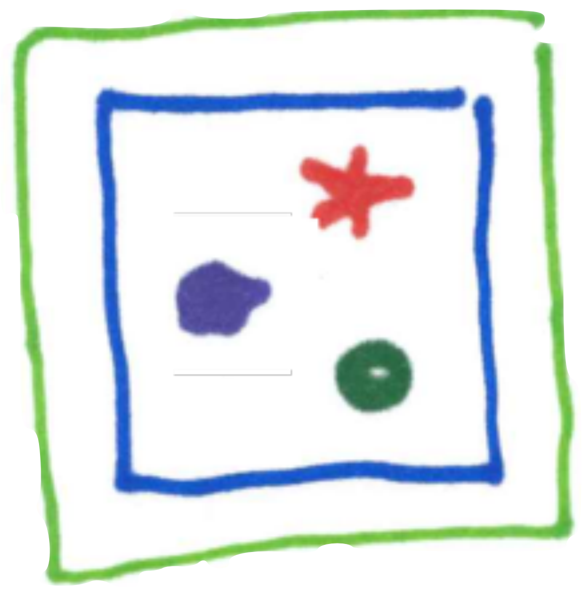


Monolithic Update

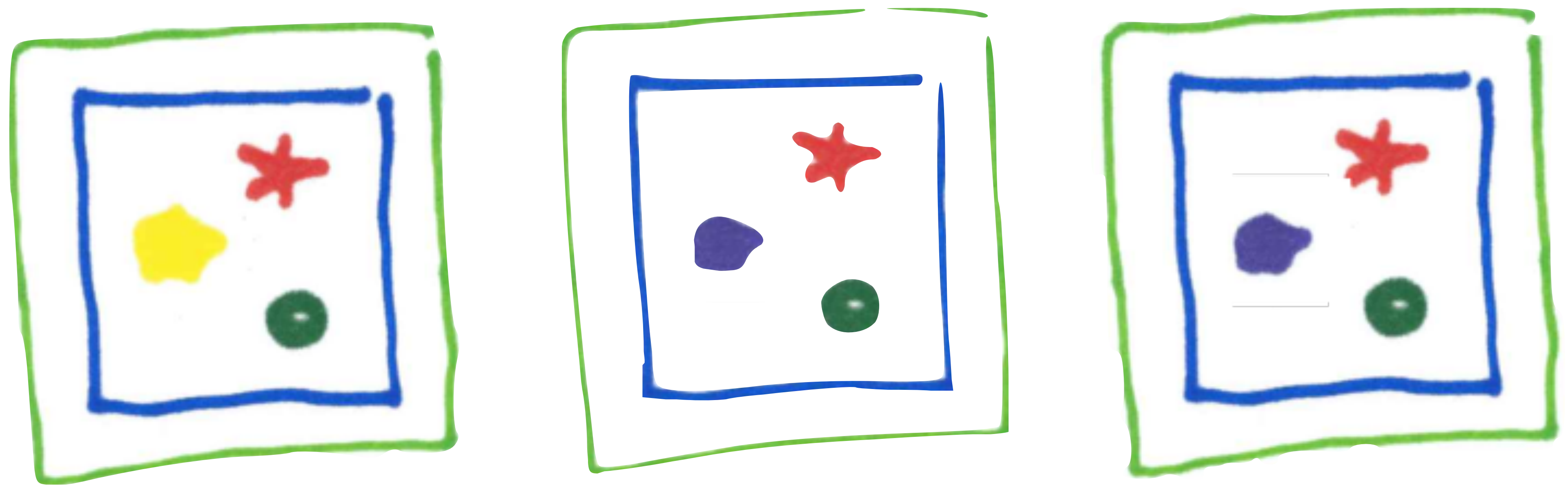


Monolithic Update

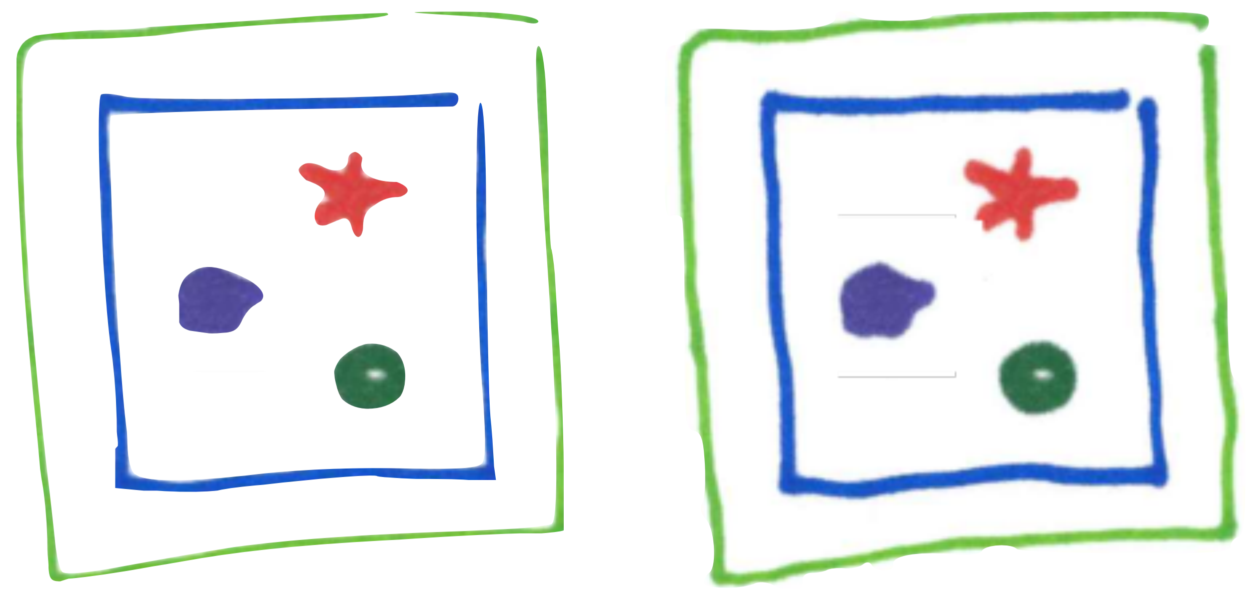




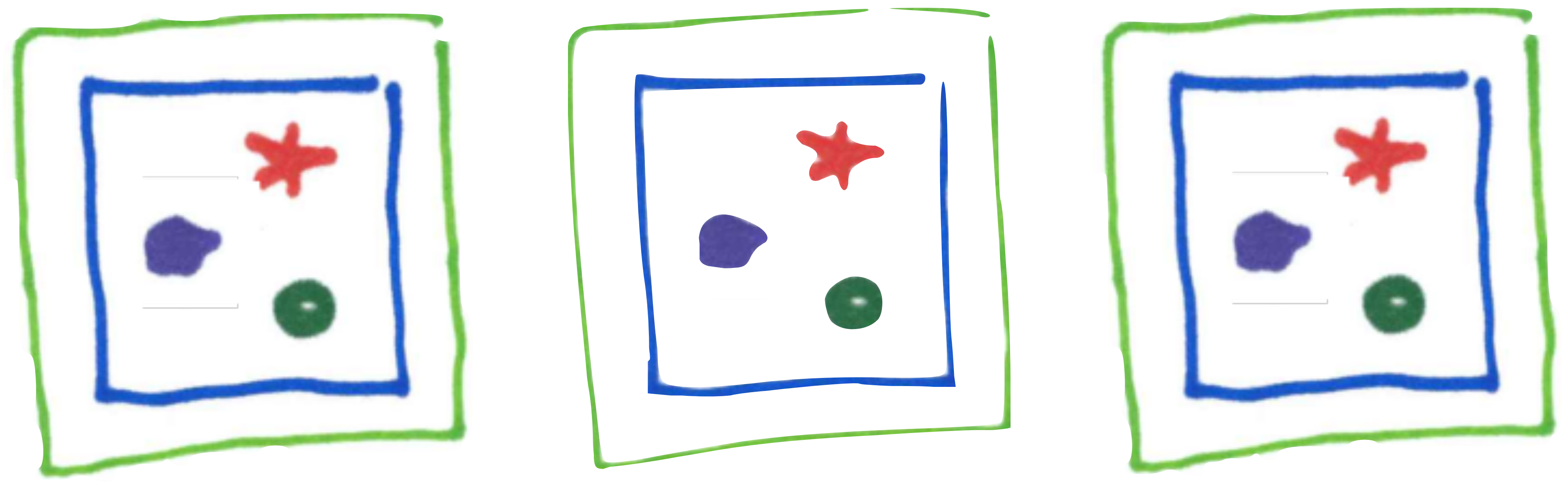
Monolithic Update



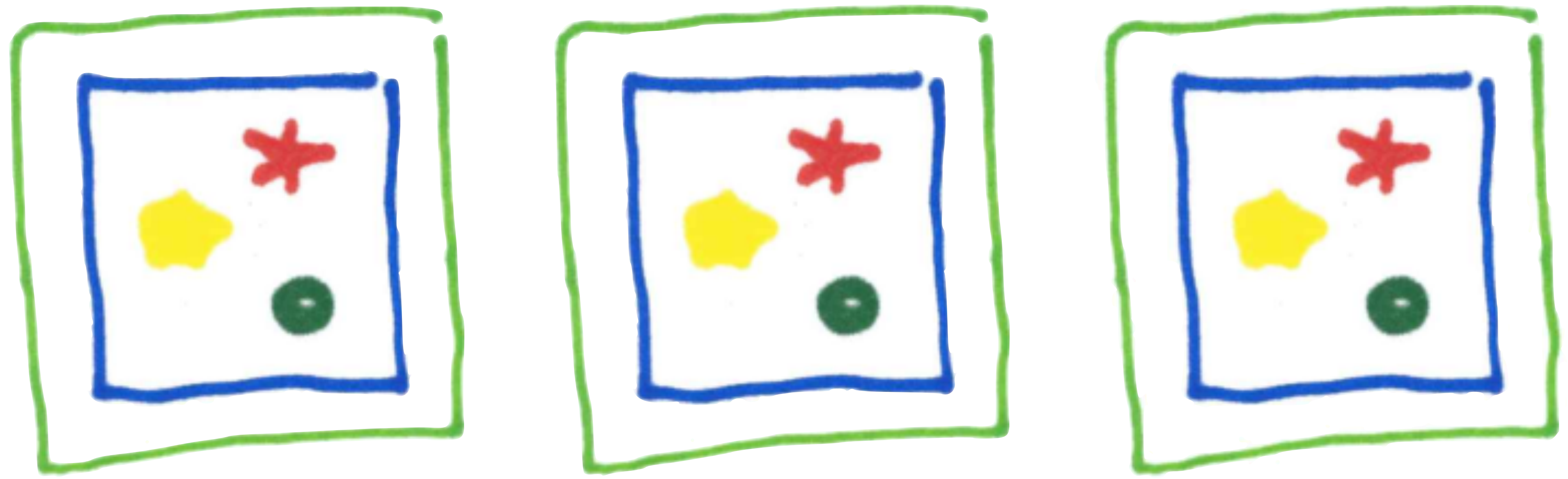
Monolithic Update



Monolithic Update



Monolithic Update

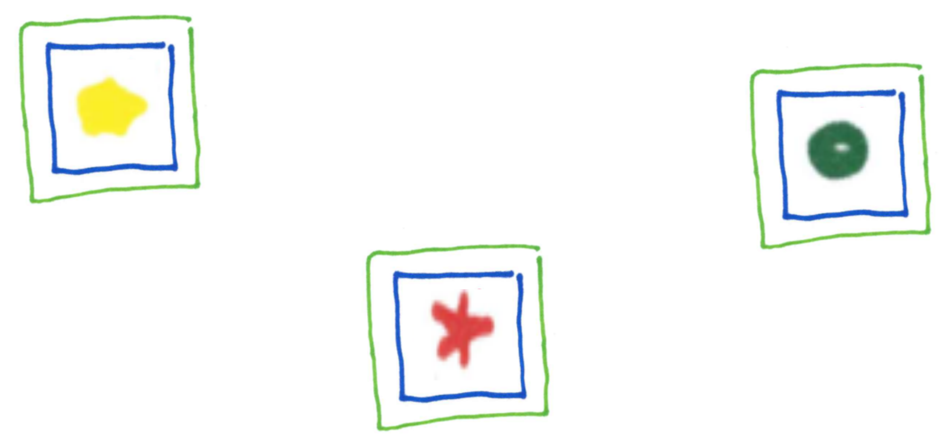


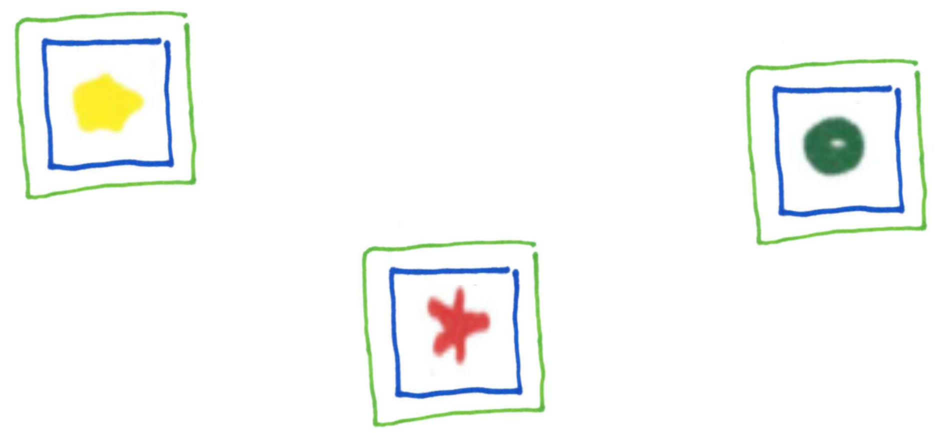
Monolithic Redesign

Monolithic Redesign

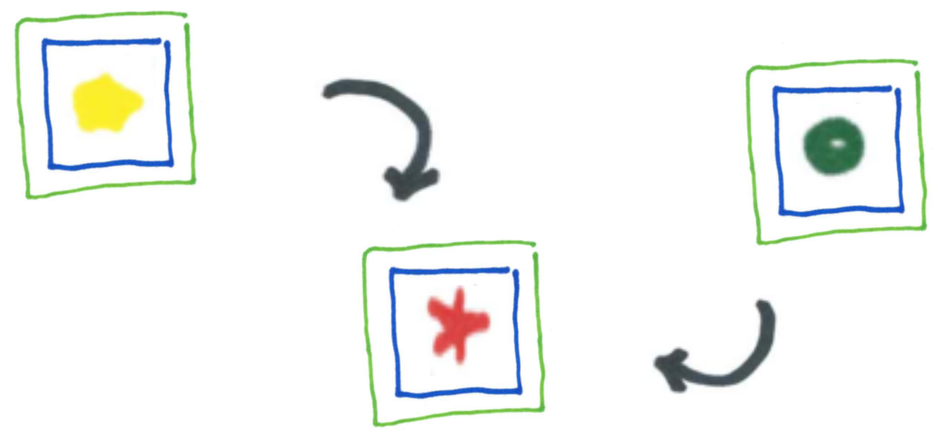
Monolithic Redesign

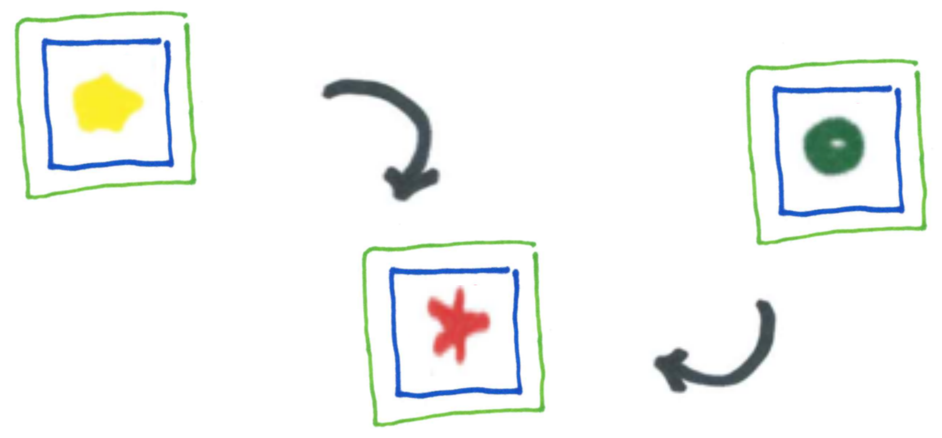
(Revolution required.)



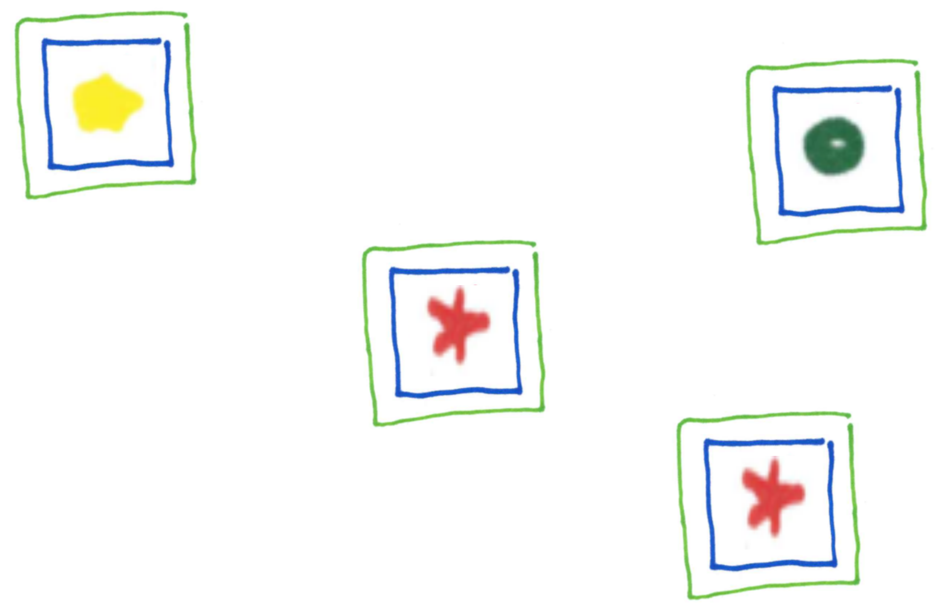


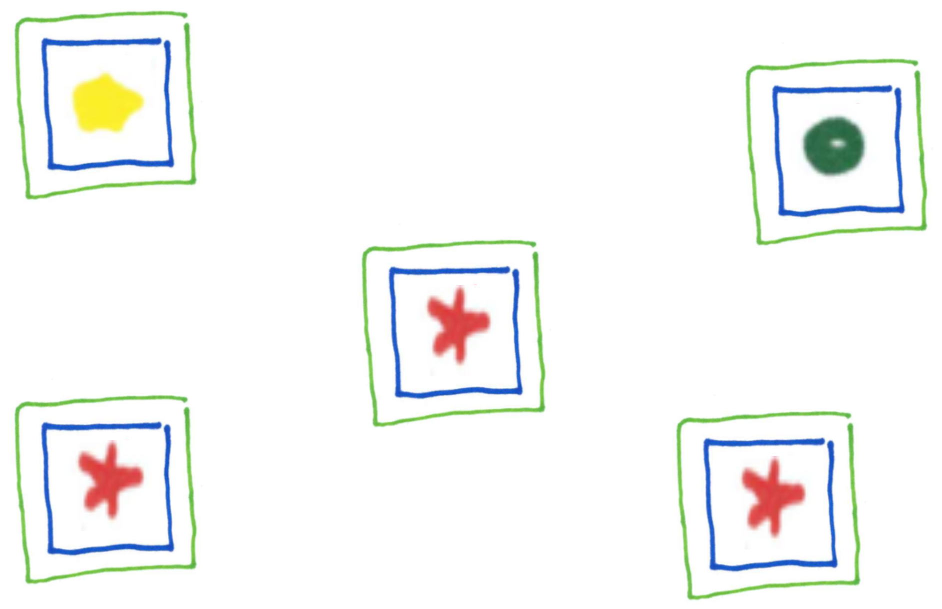
Microservice Modularity



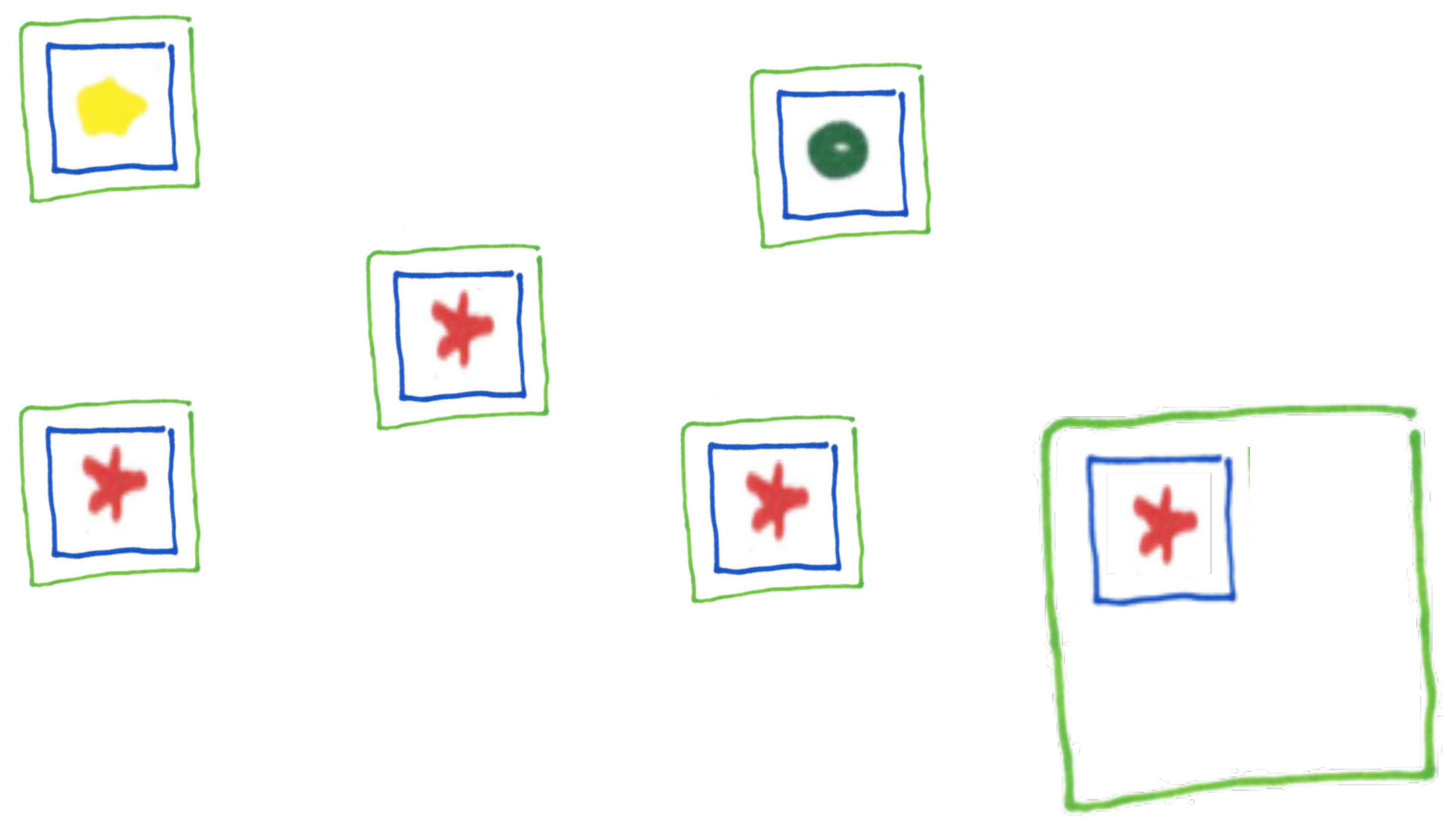


Microservice Interactions

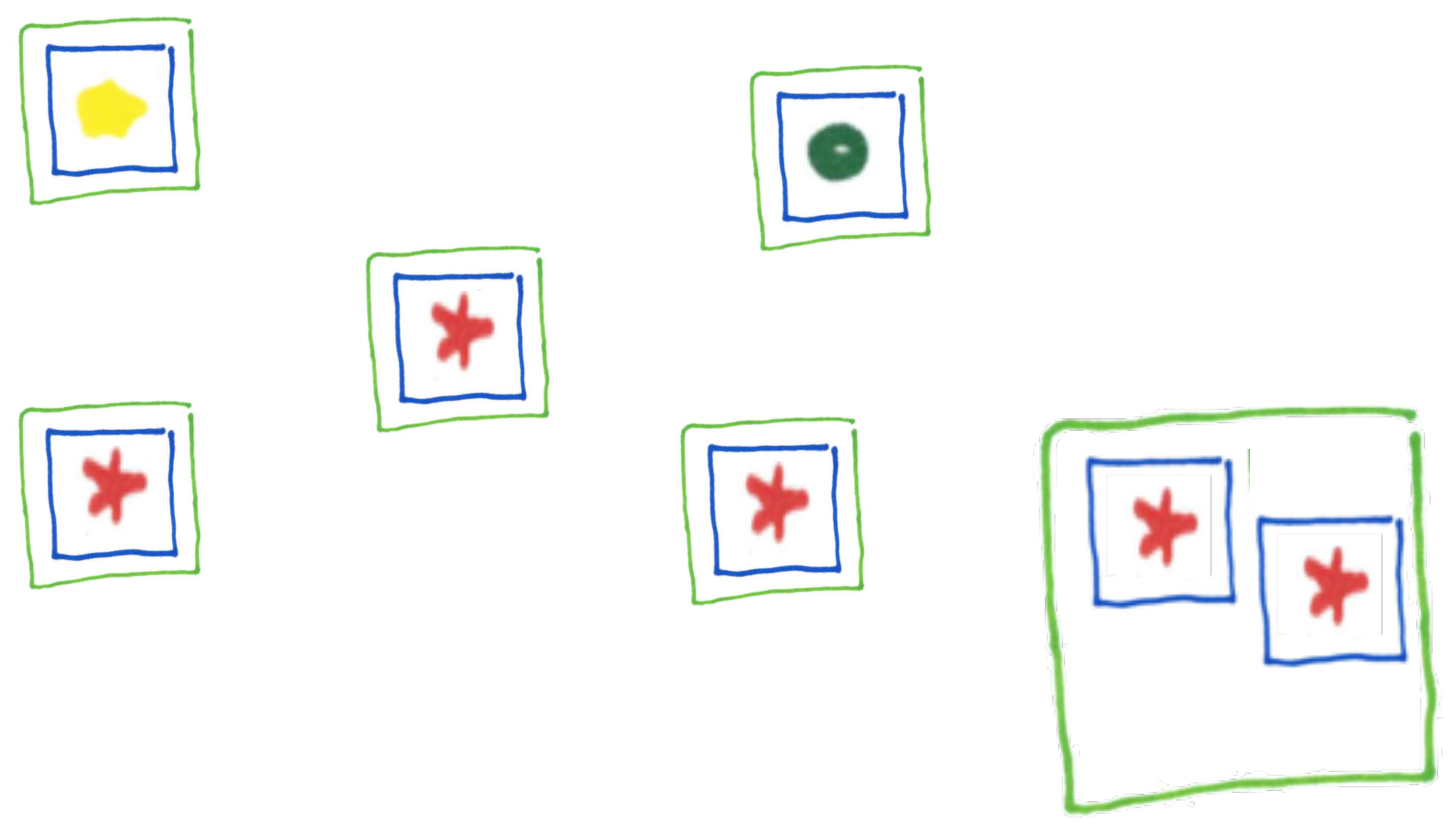




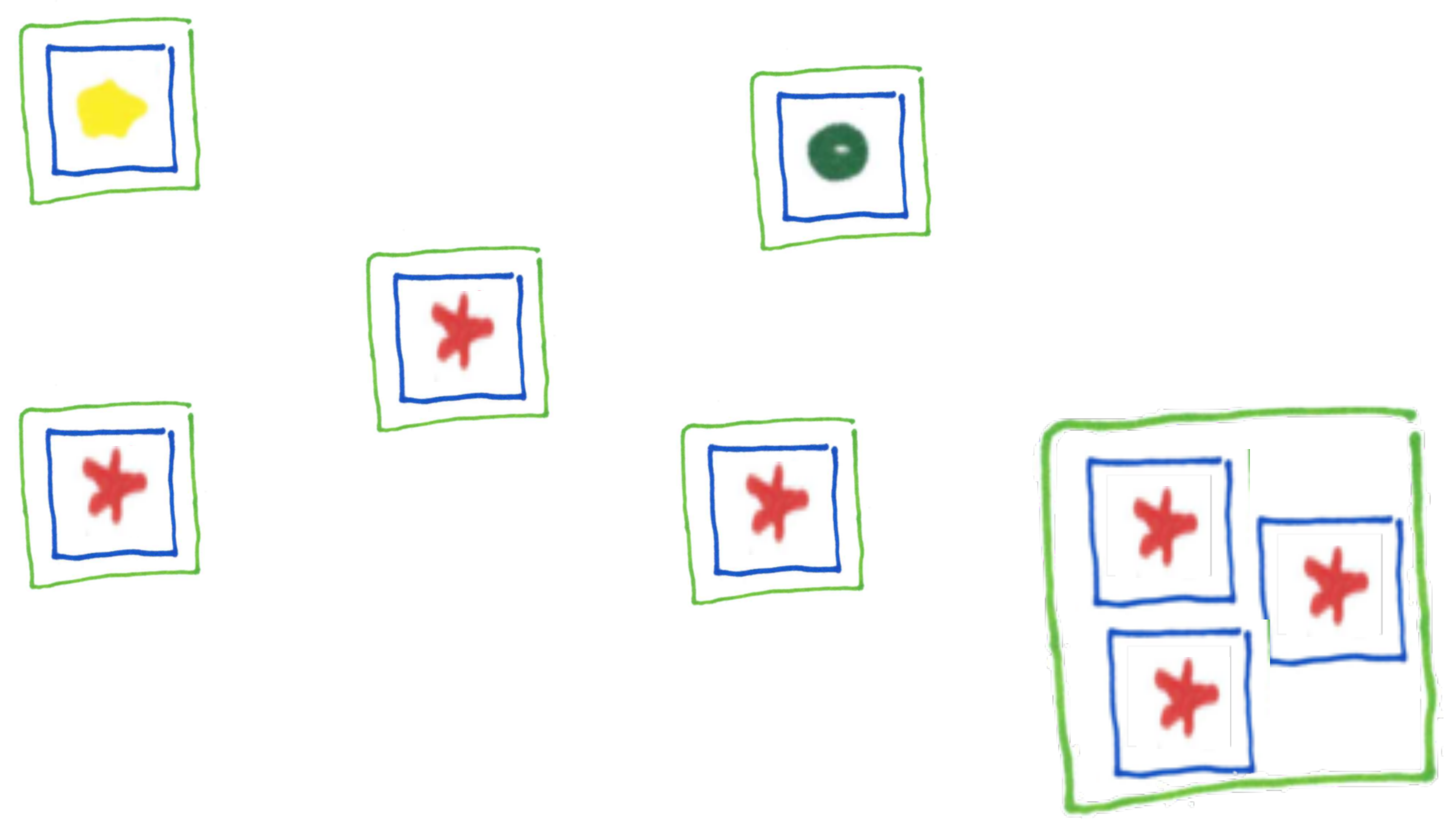
Microservices Scaled



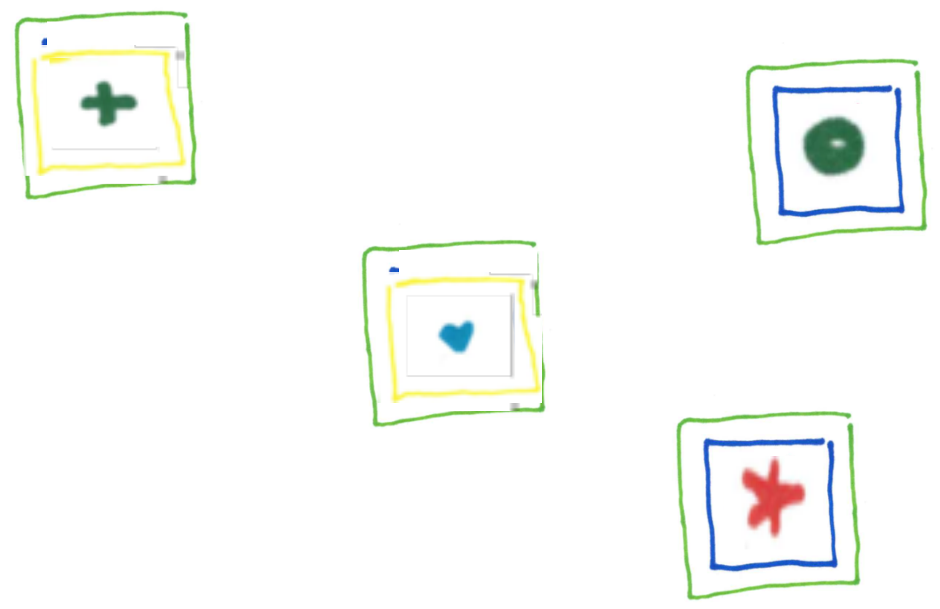
Microservices Scaled

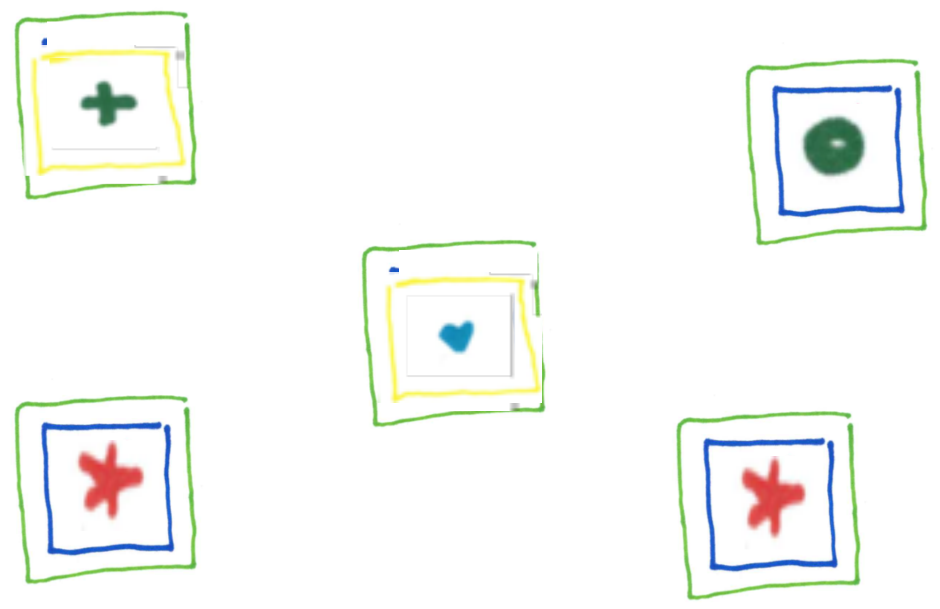


Microservices Scaled

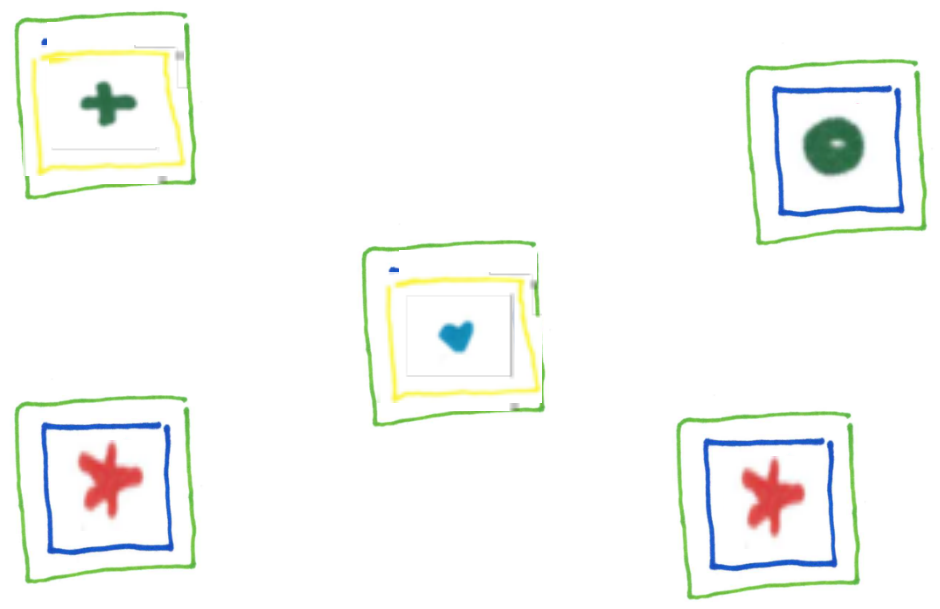


Microservices Scaled





Microservices Redesign

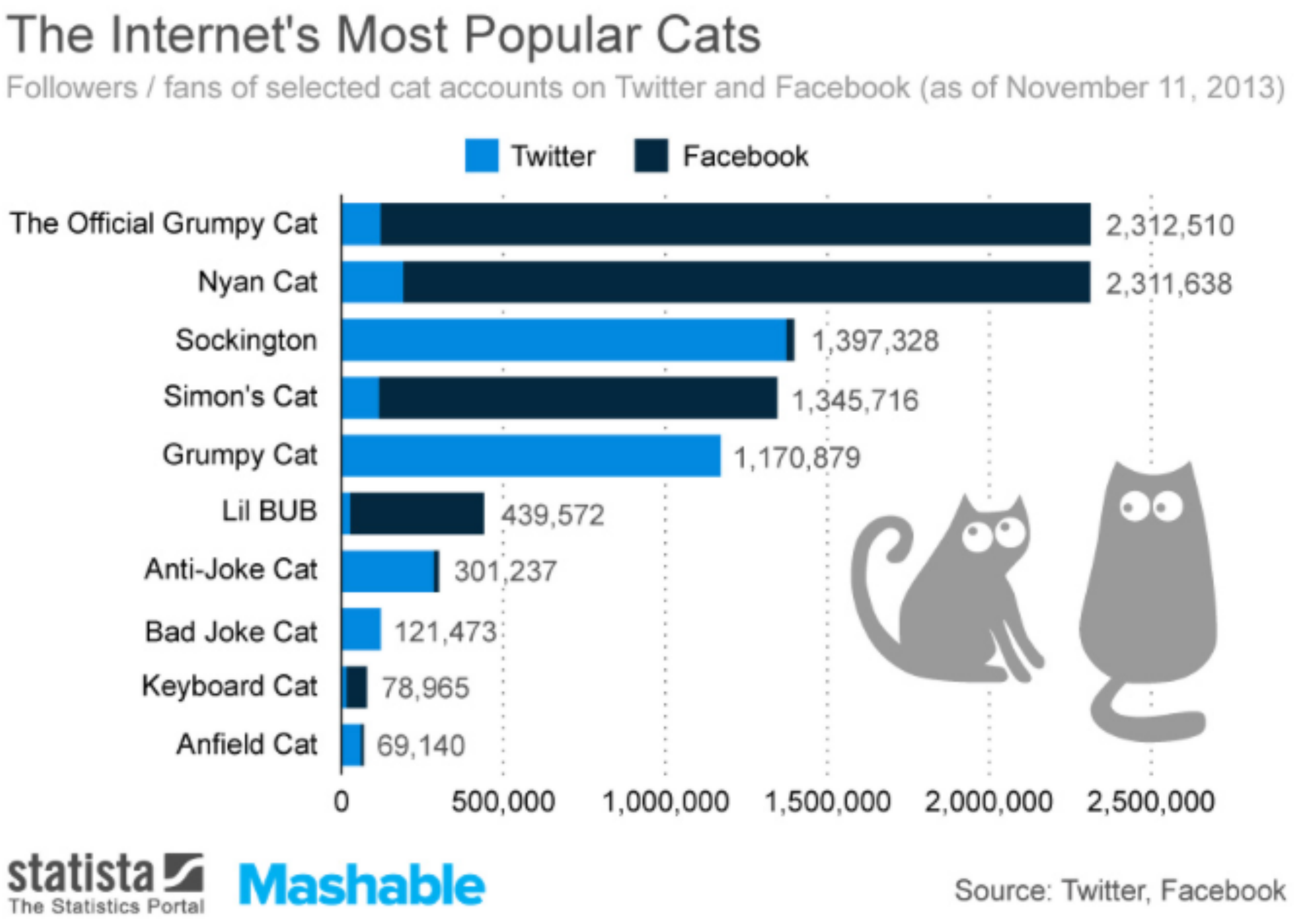


Microservices Redesign

(Evolution reduces risk.)

All good demos involve …

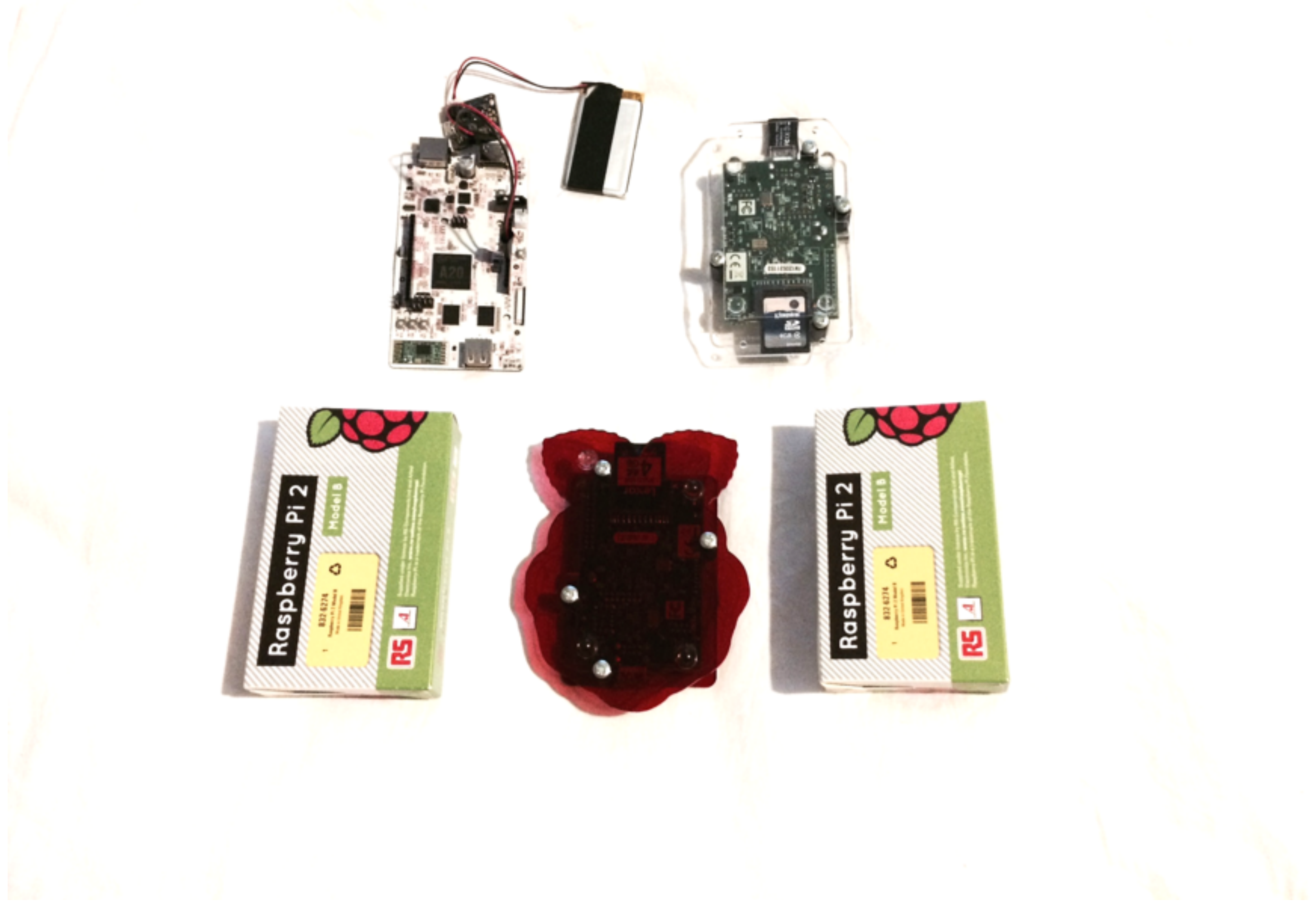
All good demos involve cats.


The Internet's Most Popular Cats
Followers / fans of selected cat accounts on Twitter and Facebook (as of November 11, 2013)
Twitter
Facebook
The Official Grumpy Cat
2,312,510
Nyan Cat
2,311,638
Sockington
1,397,328
Simon's Cat
1,345,716
Grumpy Cat
1,170,879
Lil BUB
439,572
Anti-Joke Cat
301,237
Bad Joke Cat
121,473
Keyboard Cat
78,965
Anfield Cat
69,140
0
500,000
1,000,000
1,500,000
2,000,000
2,500,000
statista The Statistics Portal
Mashable
Source: Twitter, Facebook

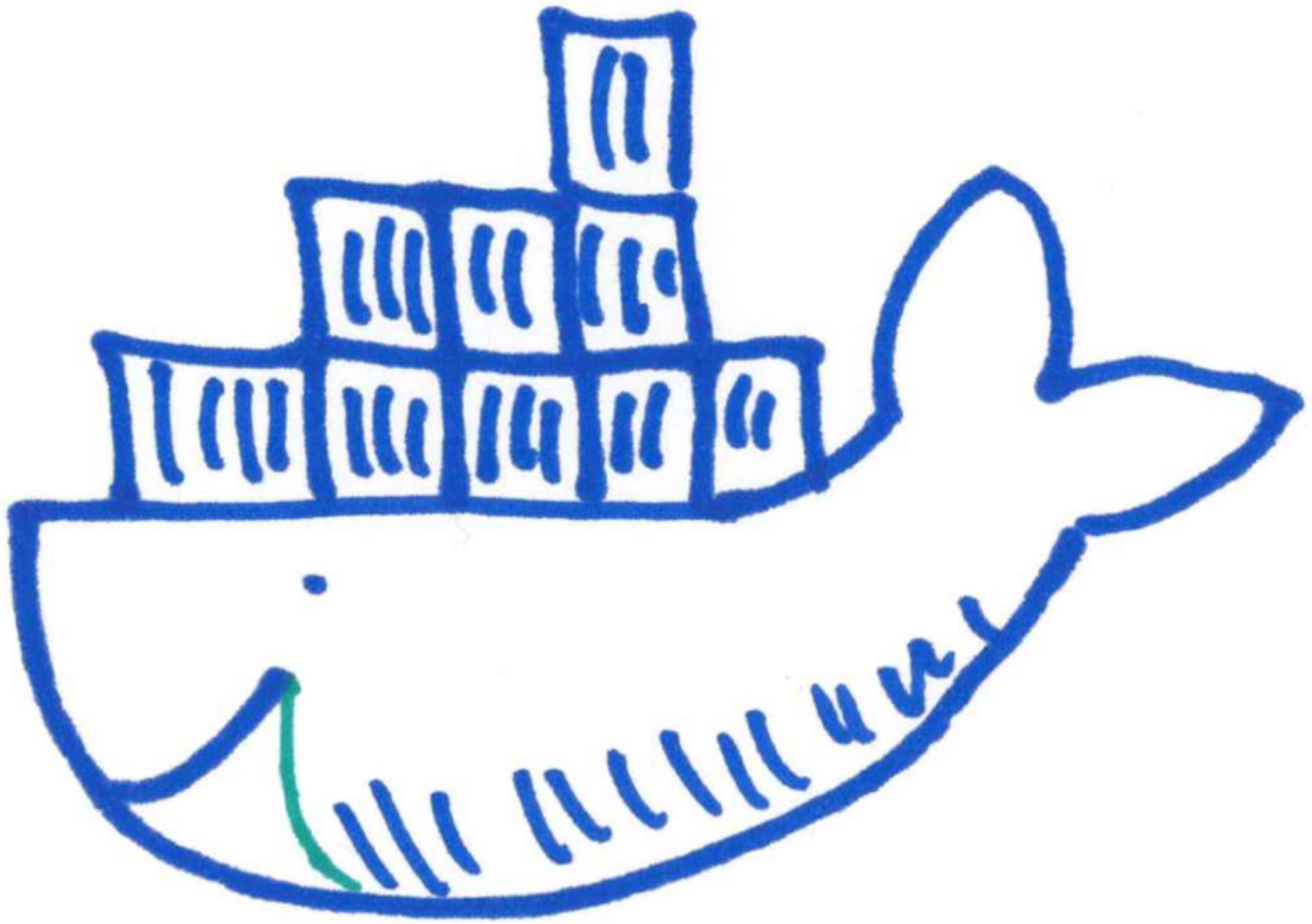
ANKER

All good demos involve cats and ...

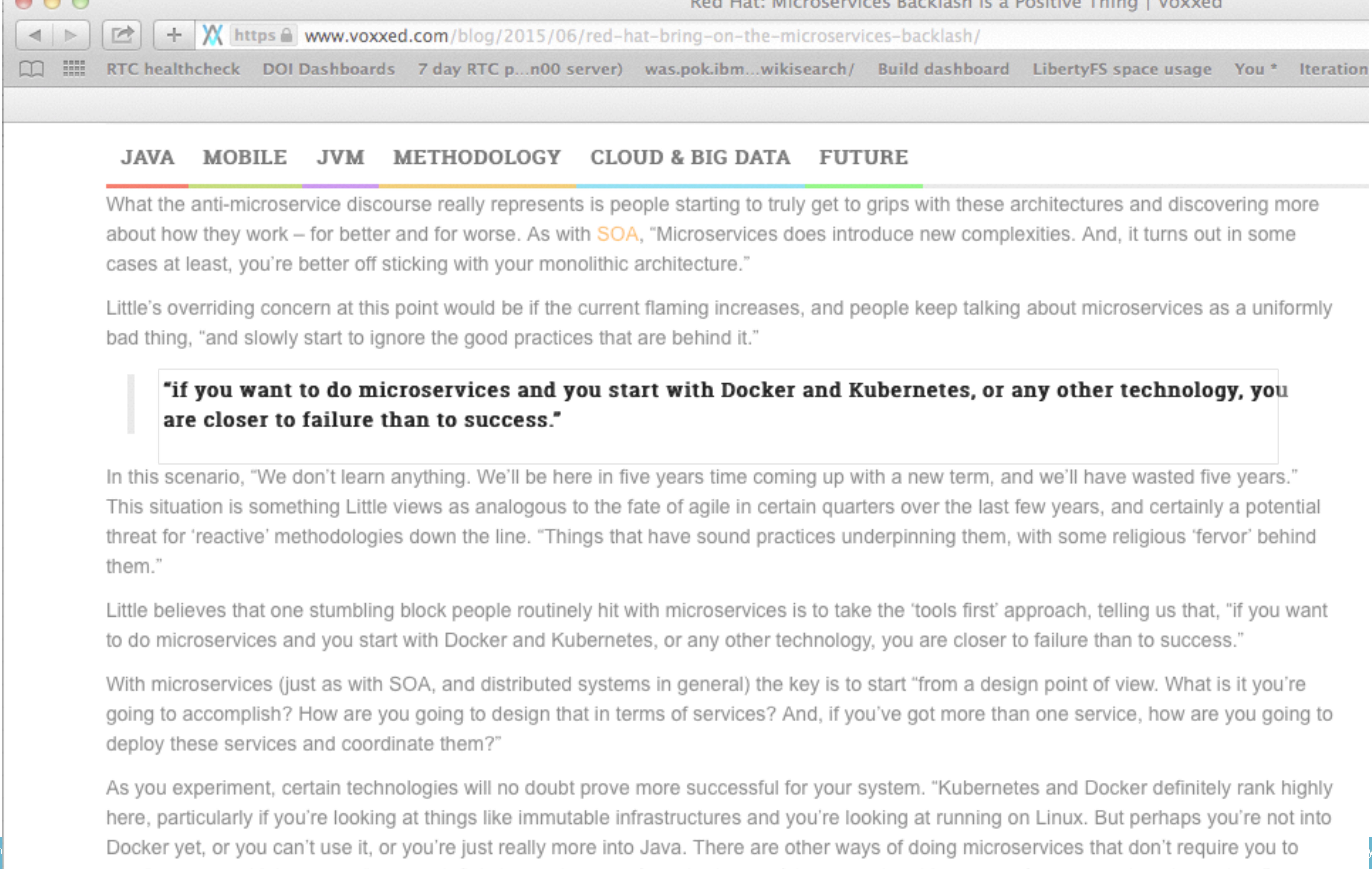
All good demos involve cats and raspberry pis.


Raspberry Pi 2
Model B
Raspberry Pi 2
Model B

Datacentre in a handbag



What, no Docker?


Red Hat: Microservices Backlash is a Positive Thing | Voxxed

https www.voxxed.com/blog/2015/06/red-hat-bring-on-the-microservices-backlash/

What the anti-microservice discourse really represents is people starting to truly get to grips with these architectures and discovering more about how they work – for better and for worse. As with SOA, "Microservices does introduce new complexities. And, it turns out in some cases at least, you're better off sticking with your monolithic architecture."

Little's overriding concern at this point would be if the current flaming increases, and people keep talking about microservices as a uniformly bad thing, "and slowly start to ignore the good practices that are behind it."

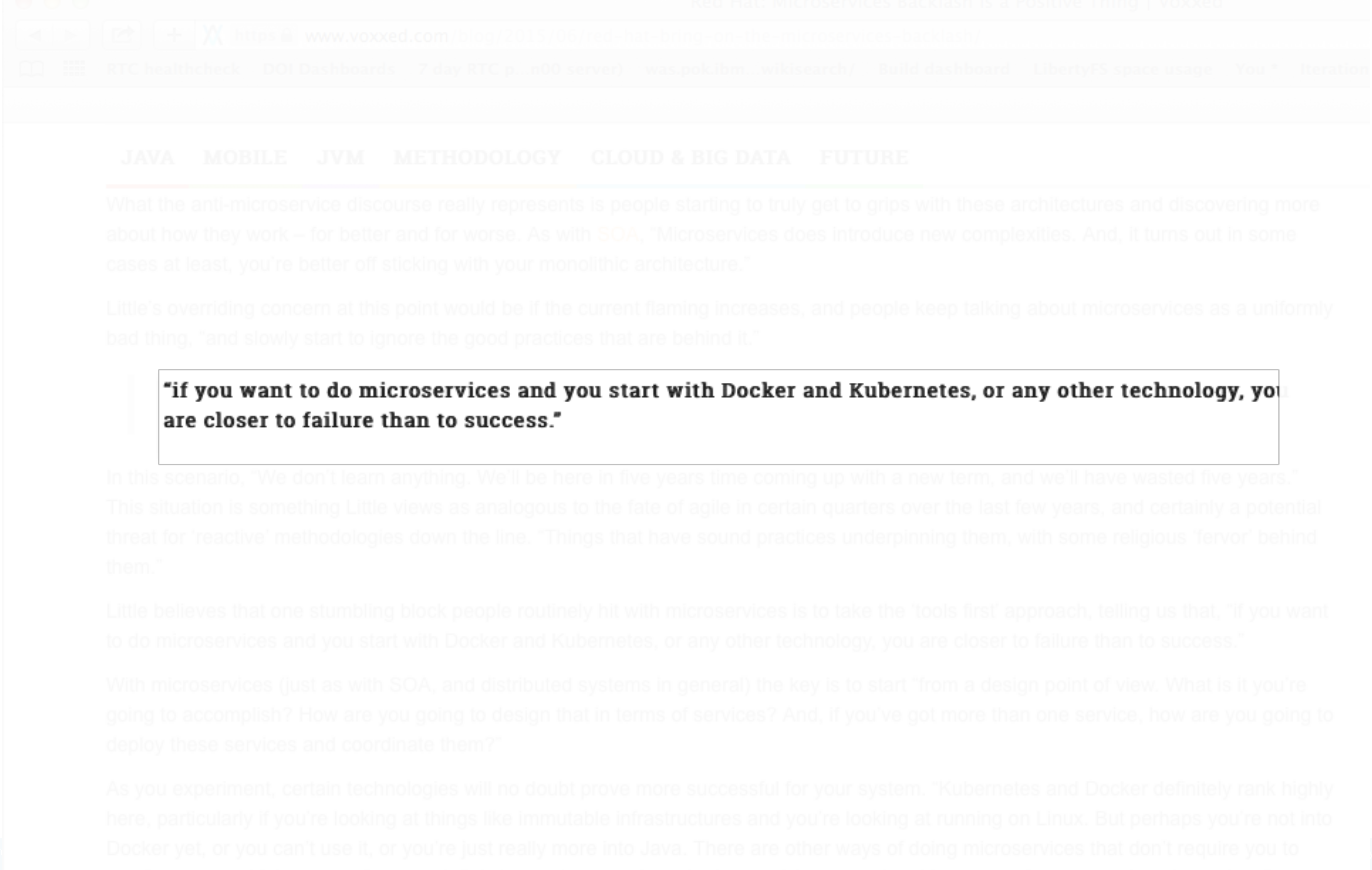
> **"if you want to do microservices and you start with Docker and Kubernetes, or any other technology, you are closer to failure than to success."**

In this scenario, "We don't learn anything. We'll be here in five years time coming up with a new term, and we'll have wasted five years." This situation is something Little views as analogous to the fate of agile in certain quarters over the last few years, and certainly a potential threat for 'reactive' methodologies down the line. "Things that have sound practices underpinning them, with some religious 'fervor' behind them."

Little believes that one stumbling block people routinely hit with microservices is to take the 'tools first' approach, telling us that, "if you want to do microservices and you start with Docker and Kubernetes, or any other technology, you are closer to failure than to success."

With microservices (just as with SOA, and distributed systems in general) the key is to start "from a design point of view. What is it you're going to accomplish? How are you going to design that in terms of services? And, if you've got more than one service, how are you going to deploy these services and coordinate them?"

As you experiment, certain technologies will no doubt prove more successful for your system. "Kubernetes and Docker definitely rank highly here, particularly if you're looking at things like immutable infrastructures and you're looking at running on Linux. But perhaps you're not into Docker yet, or you can't use it, or you're just really more into Java. There are other ways of doing microservices that don't require you to



Red Hat: Microservices backlash is a positive thing | Voxxed

www.voxxed.com/blog/2015/06/red-hat-bring-on-the-microservices-backlash/

RTC healthcheck DOI Dashboards 7 day RTC p...n00 server) was.pok.ibm...wikisearch/ Build dashboard LibertyFS space usage You * Iteration

JAVA MOBILE JVM METHODOLOGY CLOUD & BIG DATA FUTURE

What the anti-microservice discourse really represents is people starting to truly get to grips with these architectures and discovering more about how they work – for better and for worse. As with SOA, "Microservices does introduce new complexities. And, it turns out in some cases at least, you're better off sticking with your monolithic architecture."

Little's overriding concern at this point would be if the current flaming increases, and people keep talking about microservices as a uniformly bad thing, "and slowly start to ignore the good practices that are behind it."

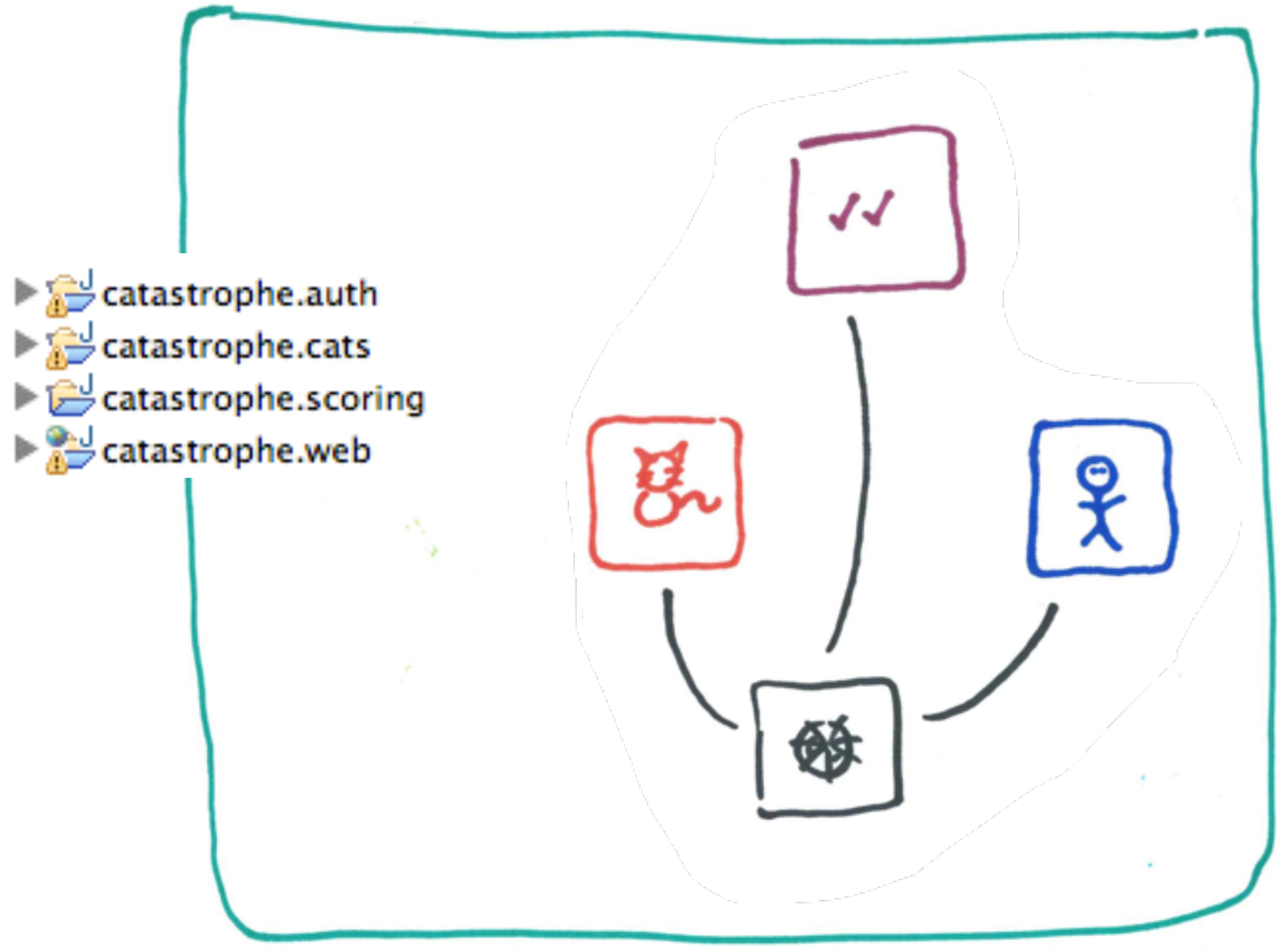
"if you want to do microservices and you start with Docker and Kubernetes, or any other technology, you are closer to failure than to success."

In this scenario, "We don't learn anything. We'll be here in five years time coming up with a new term, and we'll have wasted five years." This situation is something Little views as analogous to the fate of agile in certain quarters over the last few years, and certainly a potential threat for 'reactive' methodologies down the line. "Things that have sound practices underpinning them, with some religious 'fervor' behind them."

Little believes that one stumbling block people routinely hit with microservices is to take the 'tools first' approach, telling us that, "if you want to do microservices and you start with Docker and Kubernetes, or any other technology, you are closer to failure than to success."

With microservices (just as with SOA, and distributed systems in general) the key is to start "from a design point of view. What is it you're going to accomplish? How are you going to design that in terms of services? And, if you've got more than one service, how are you going to deploy these services and coordinate them?"

As you experiment, certain technologies will no doubt prove more successful for your system. "Kubernetes and Docker definitely rank highly here, particularly if you're looking at things like immutable infrastructures and you're looking at running on Linux. But perhaps you're not into Docker yet, or you can't use it, or you're just really more into Java. There are other ways of doing microservices that don't require you to

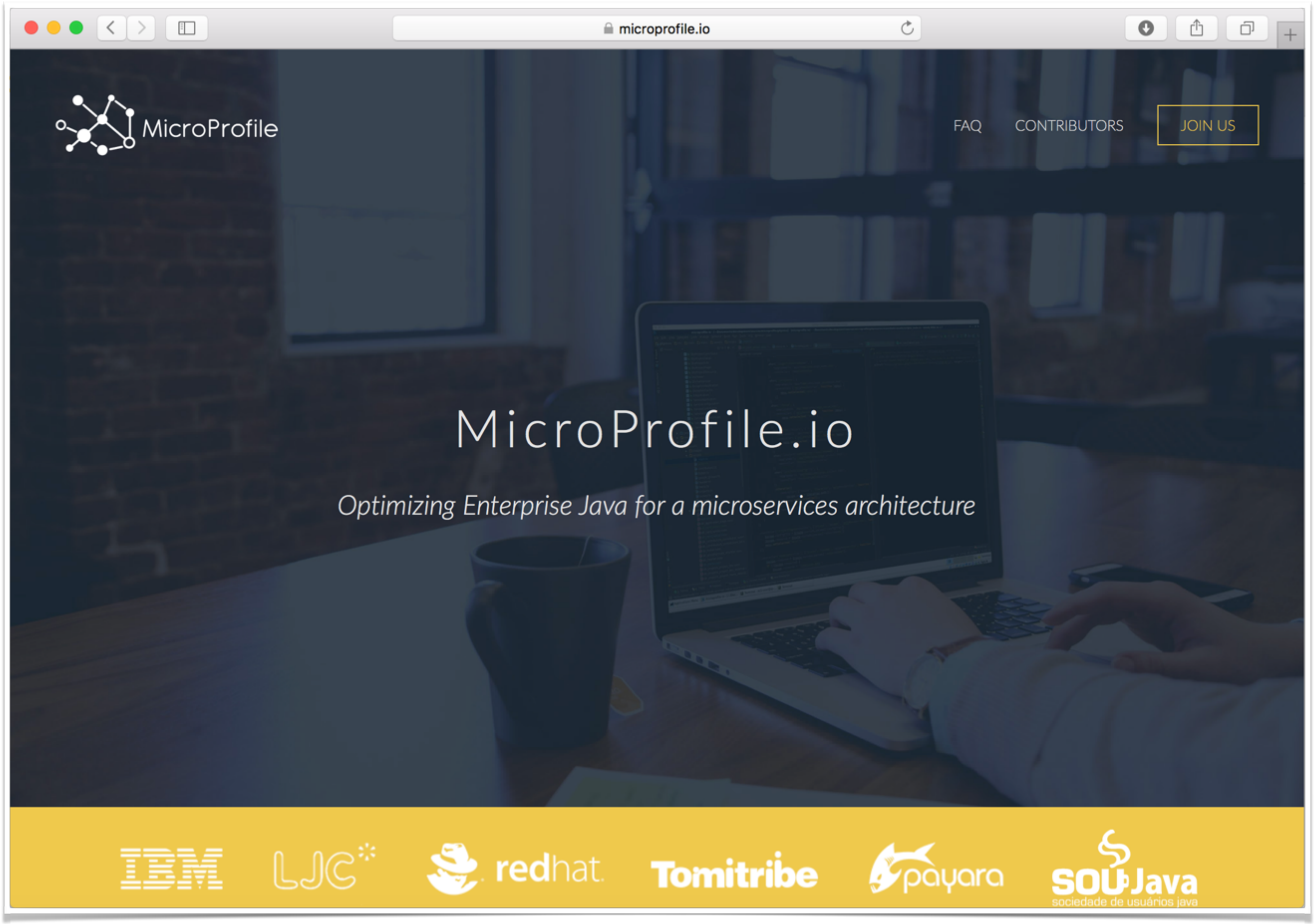


Cat-astrophe

Powered by

WebSphere Liberty …

of course



microprofile.io

MicroProfile

FAQ CONTRIBUTORS JOIN US

MicroProfile.io

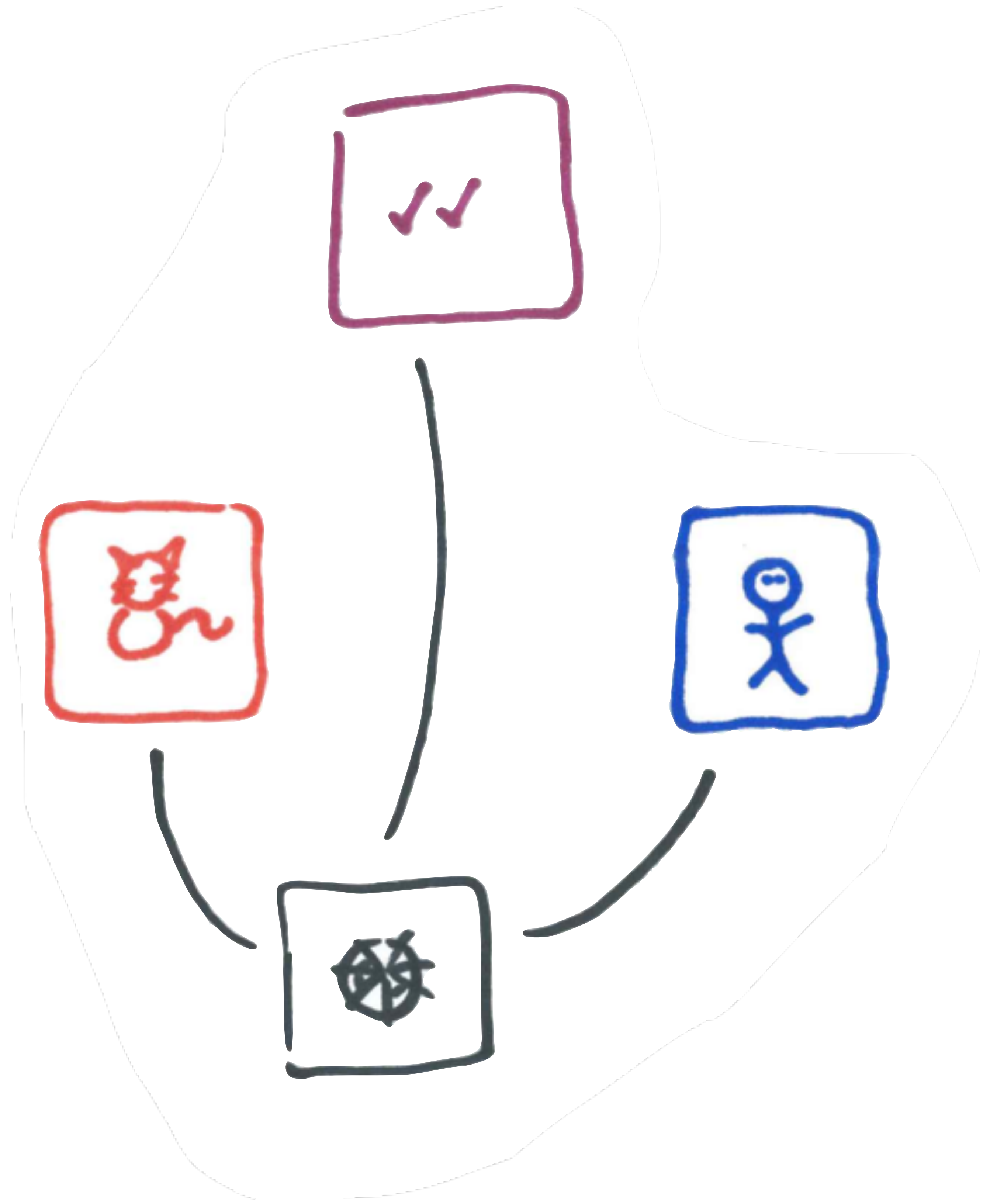
Optimizing Enterprise Java for a microservices architecture

http://raspberrypi.local:8080/

What happens if things fail?

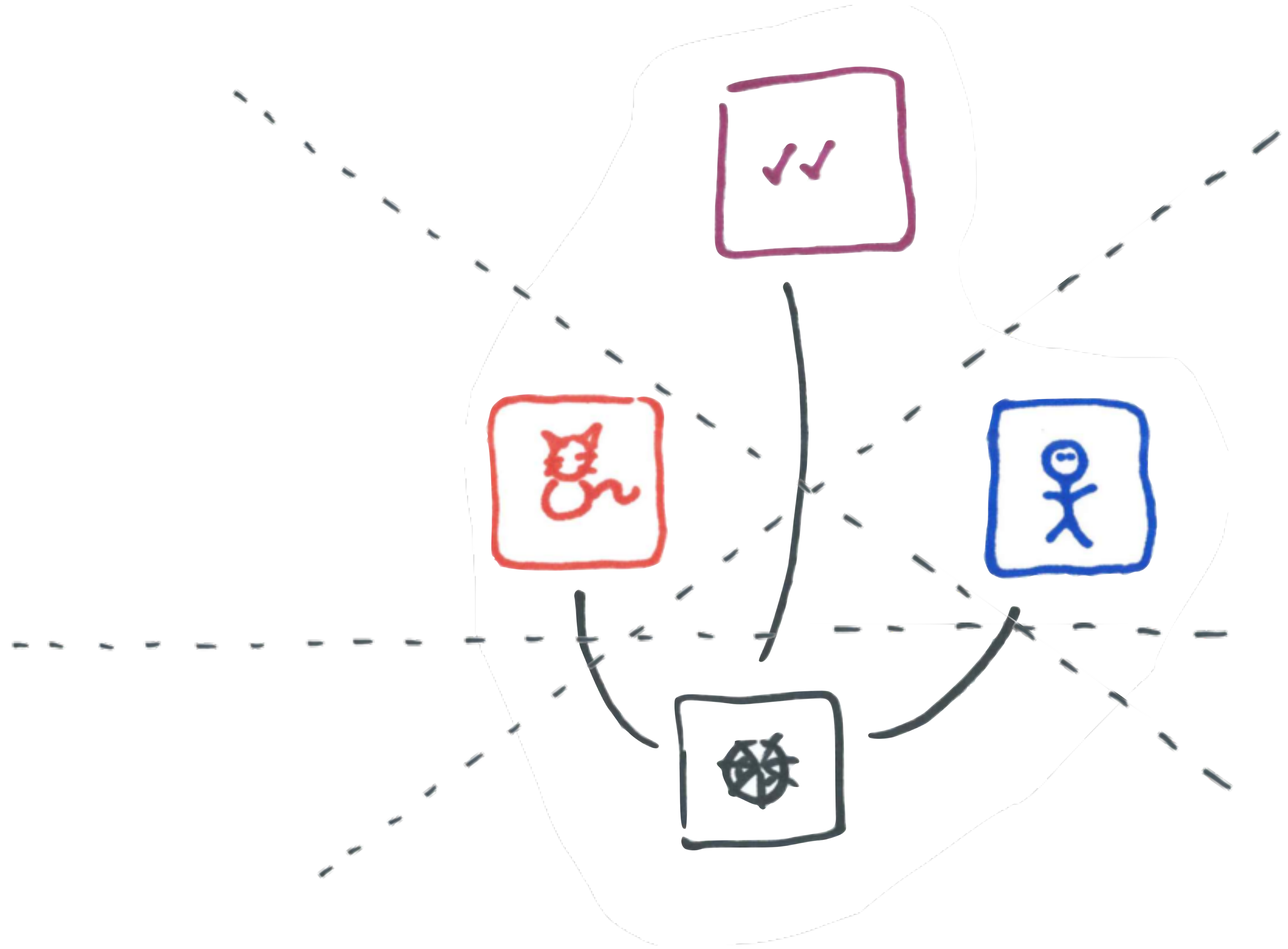
Refactoring your way to the **microservices** dream

```
http://github.com/holly-cummins/catastrophe-microservices
```



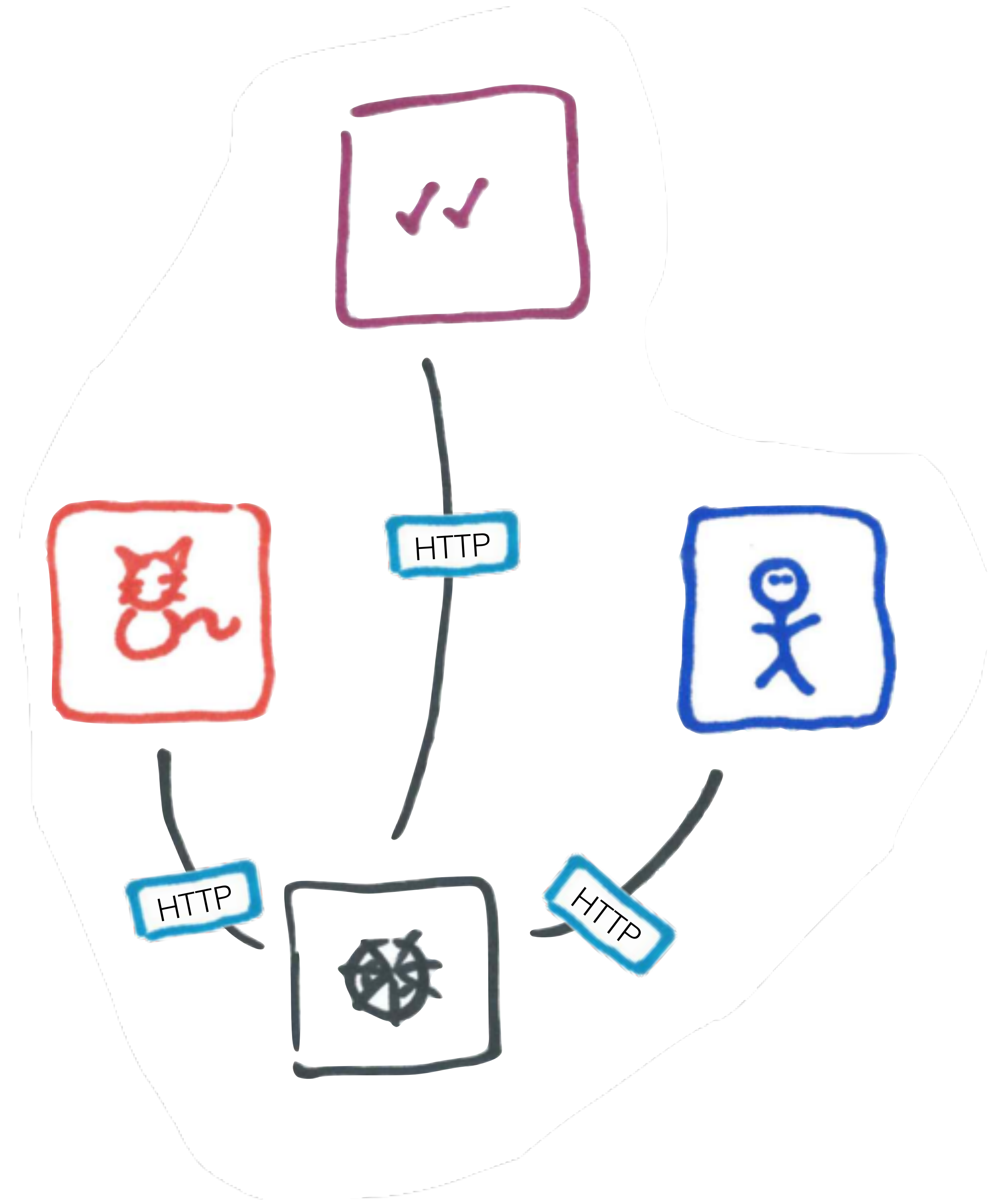
Slice it up!

```
http://github.com/holly-cummins/catastrophe-microservices
```



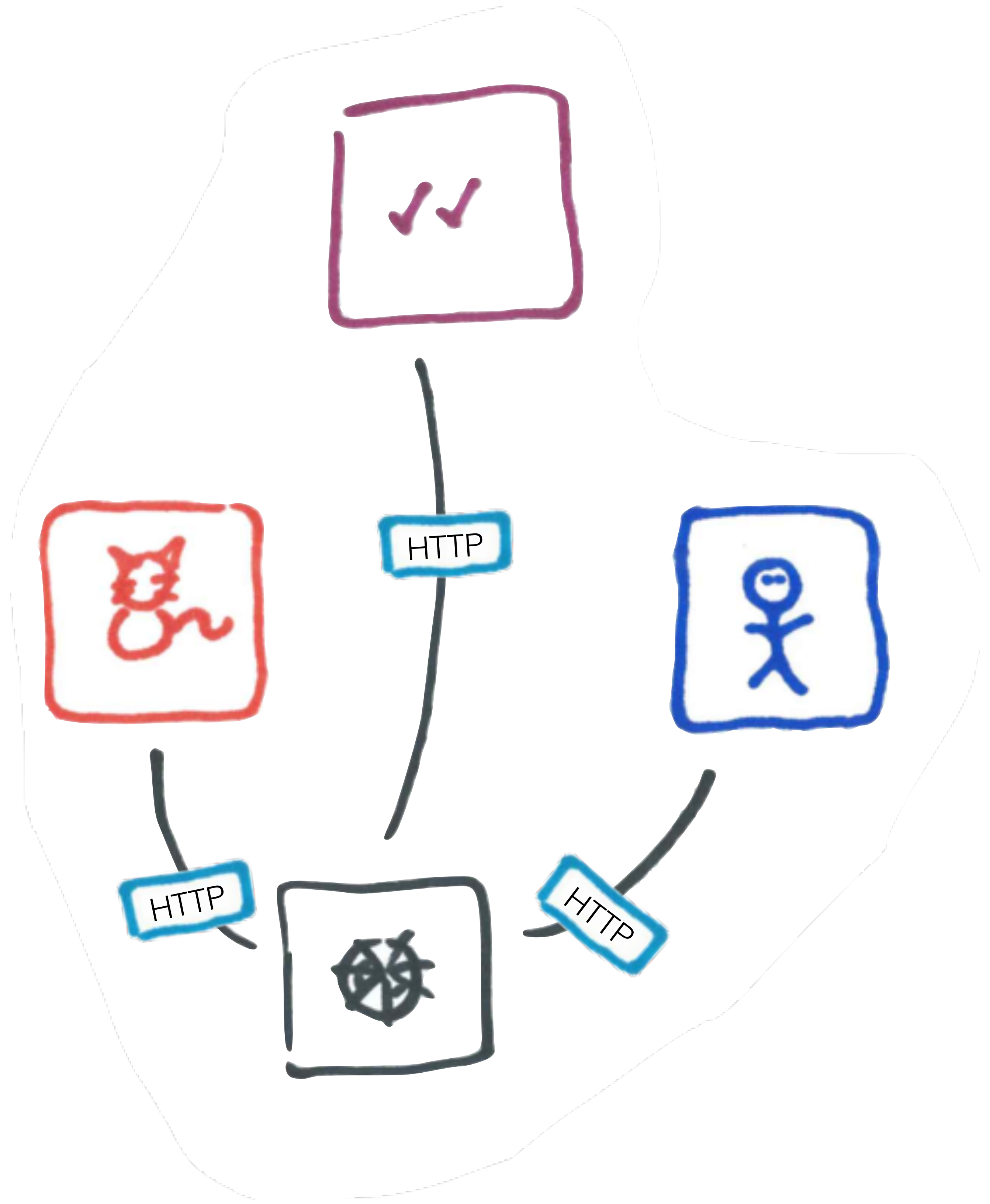
Slice it up!

http://github.com/holly-cummins/catastrophe-microservices



Slice it up!

http://github.com/holly-cummins/catastrophe-microservices



Peel it off.

Should we decompose the front-end?

- Probably not.

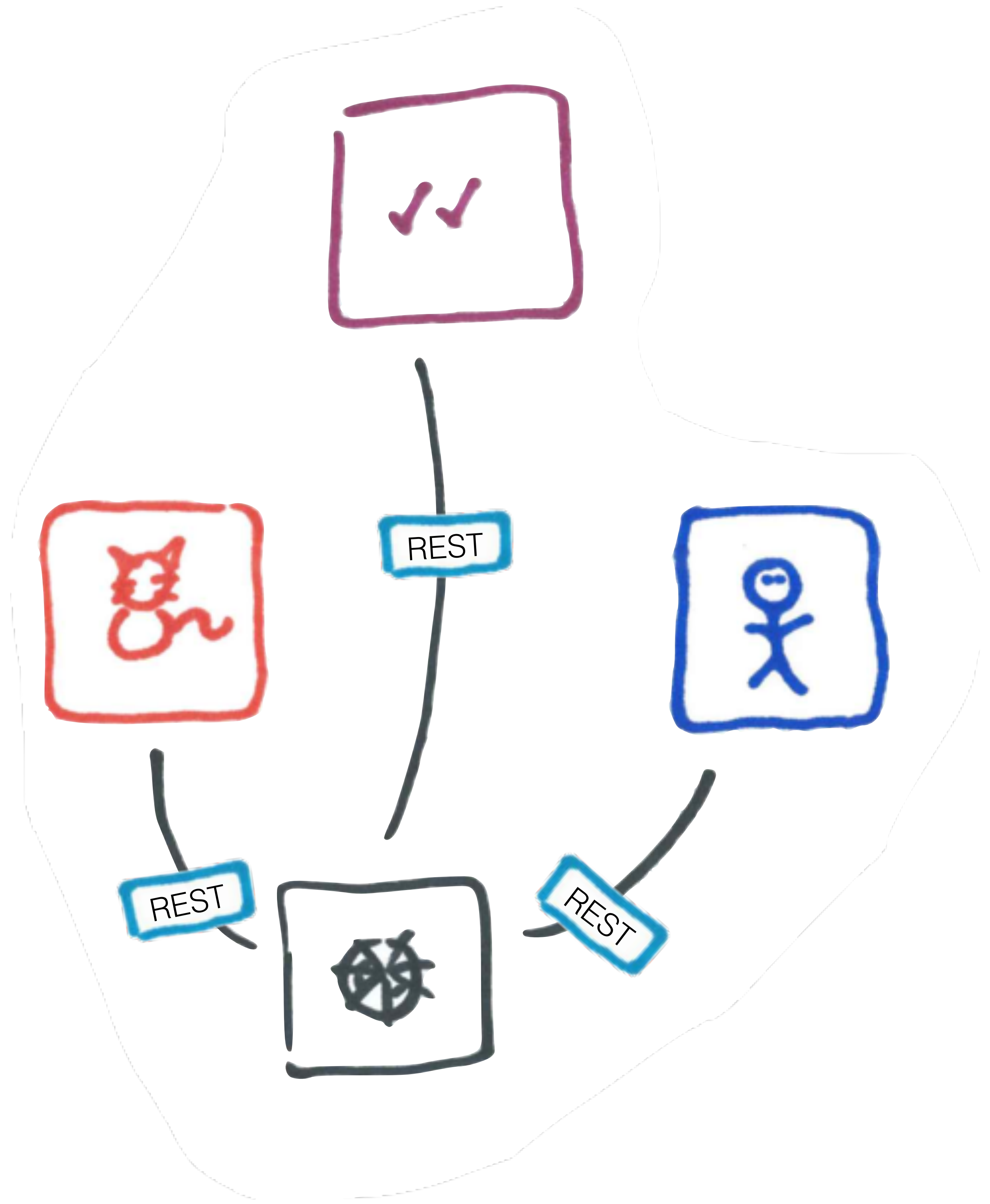
Should we decompose the front-end?

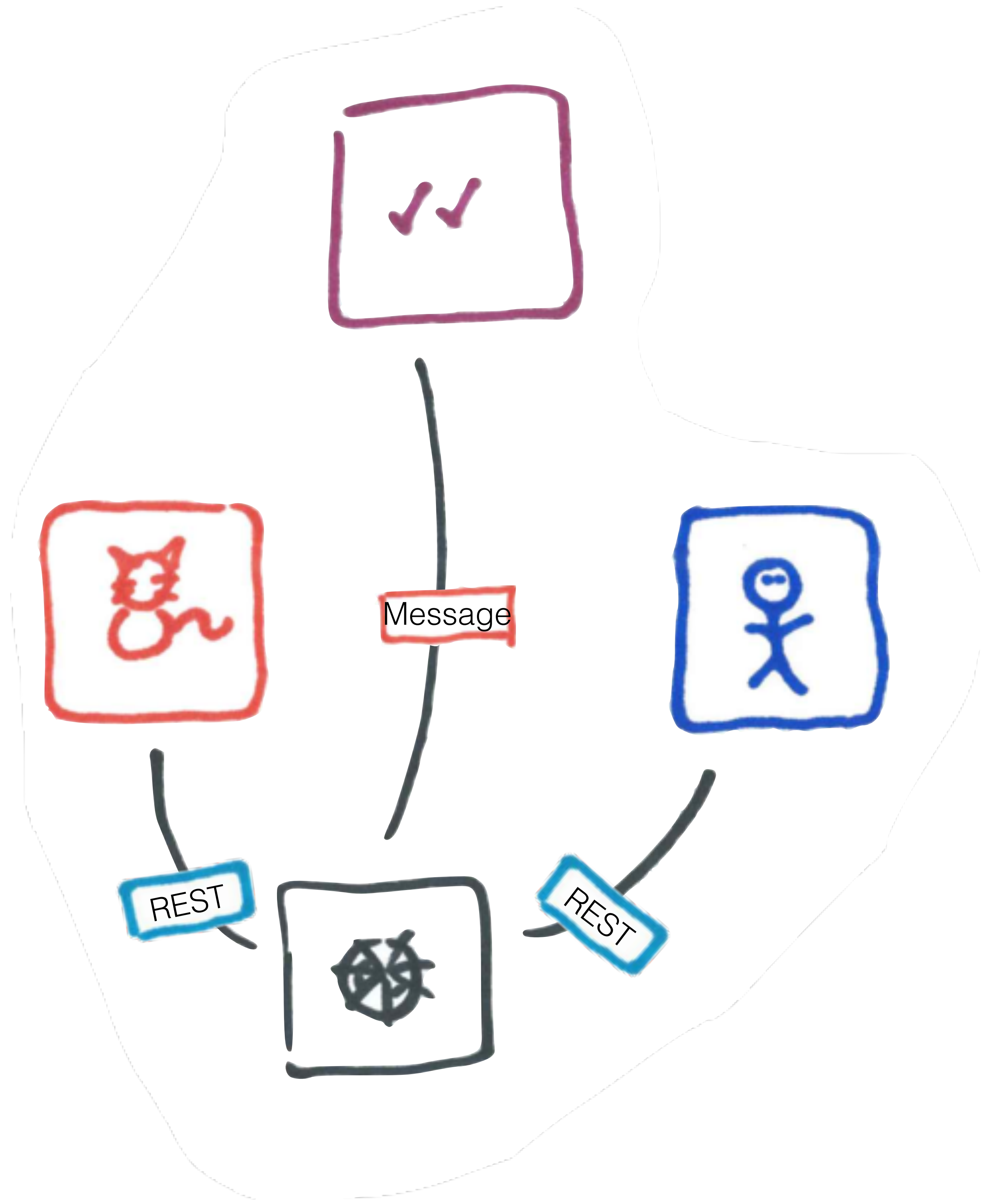
- Probably not.
    - Single Origin headaches

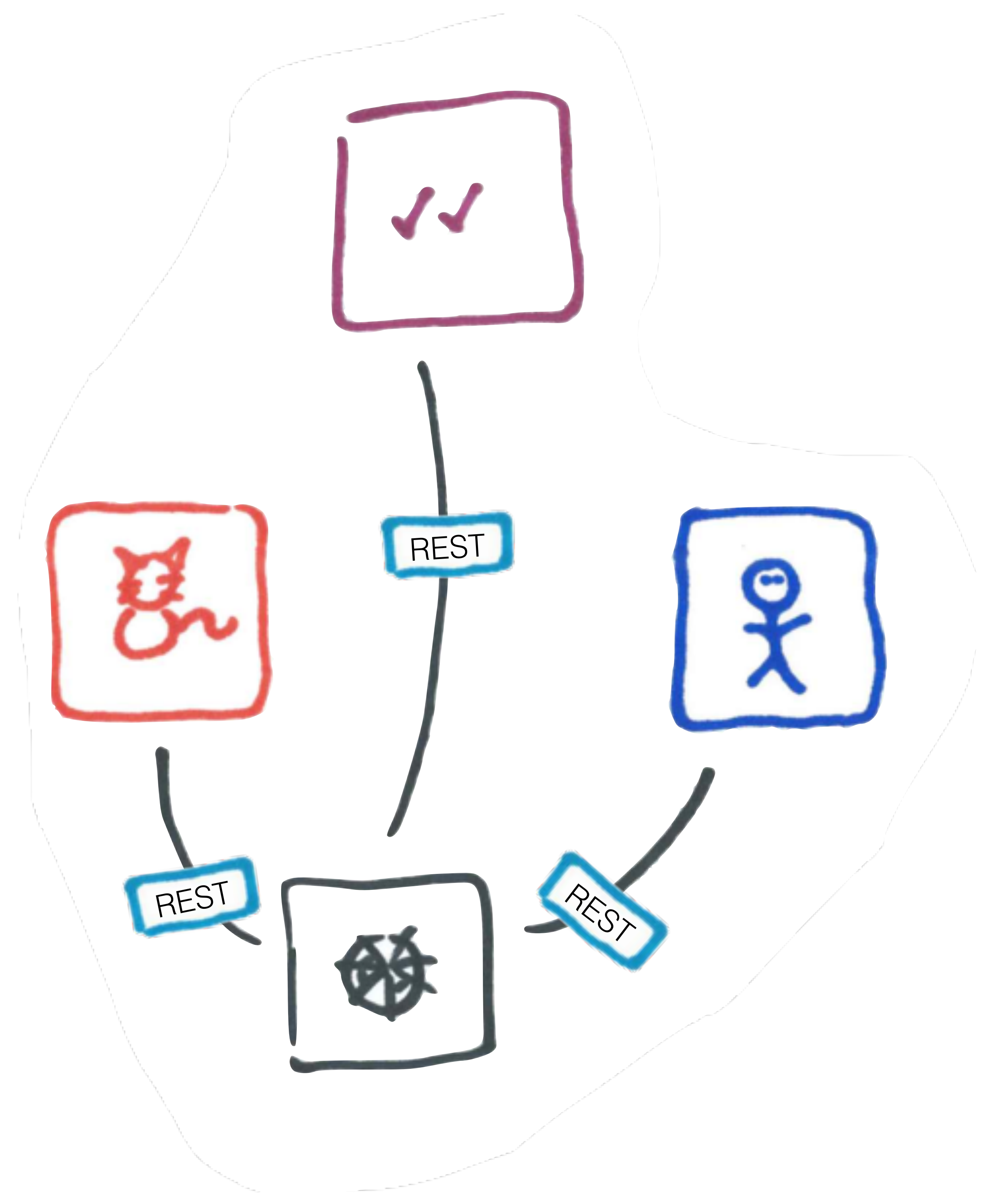
Should we decompose the front-end?

- Probably not.
  - Single Origin headaches
  - Page composition headaches

Should we decompose the front-end?


REST
REST
REST


Message
REST
REST


REST
REST
REST

- Synchronous is convenient
- Asynchronous has scalability advantages
- Consider reactive architectures

REST != synchronous (well, not necessarily)

How **hard** the refactoring is depends on **where** you started

```
@ApplicationScoped
public class CatRepository {

public Set<Cat> getAllCats()
{
```

Exposing a service in a monolith

```
@Path("cat")
public class CatRepository {

@Path("allcats")
@Produces(MediaType.APPLICATION_JSON)
@GET
public Set<Cat> getAllCats() {
    …
```

Exposing a microservice

```
@Path("cat")
public class CatRepository {

@Path("allcats")
@Produces(MediaType.APPLICATION_JSON)
@GET
public Set<Cat> getAllCats() {
    …
```

JAXRS=magic

```
@Path("allcats")
@Asynchronous
@GET
public void getAllCats(@Suspended final AsyncResponse response)
{
    // stuff
   response.resume(stuff)
```

Go asynchronous for scalability

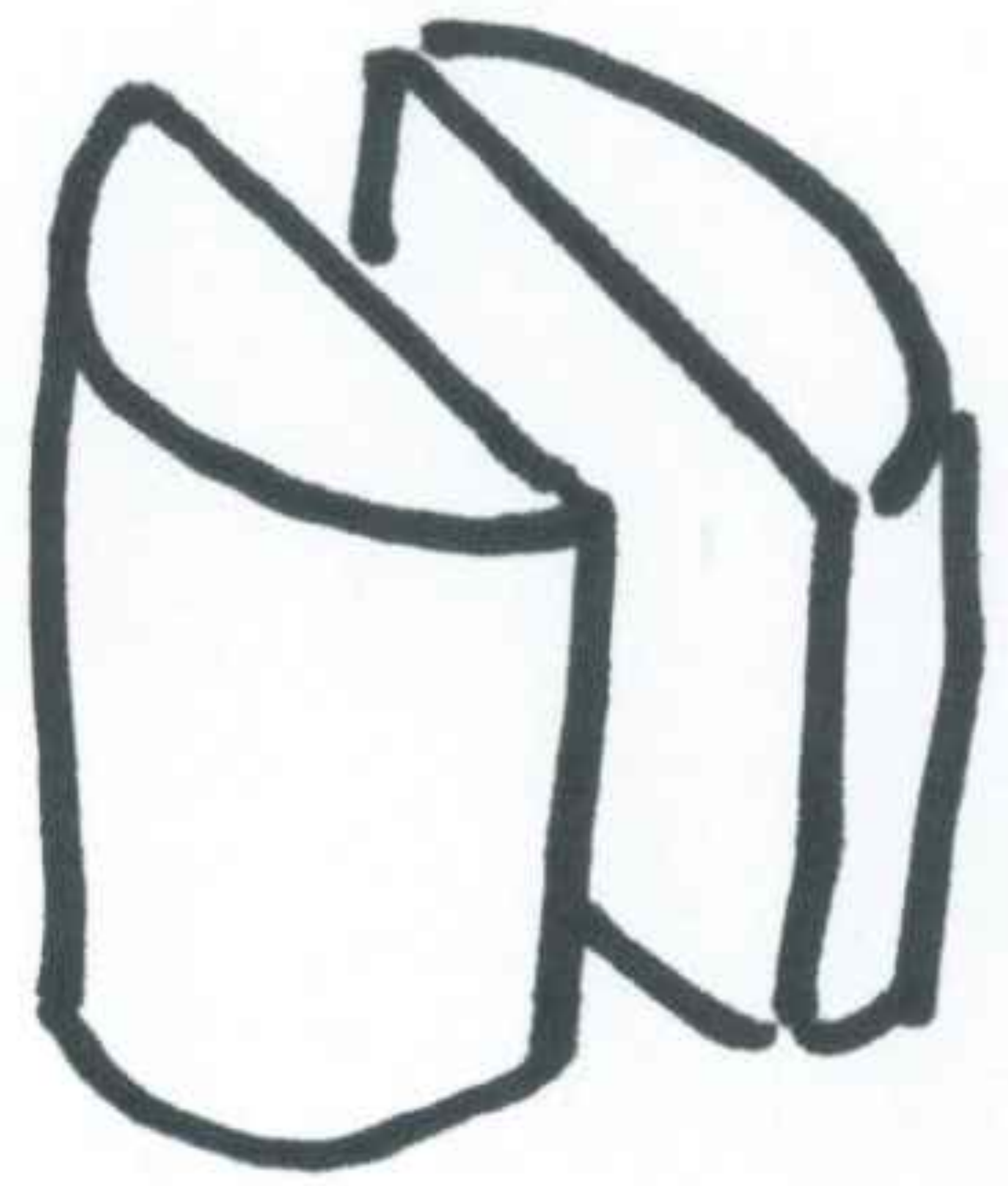
```java
@Path("allcats")
@Asynchronous
@GET
public void getAllCats(@Suspended final AsyncResponse response)
{
    // stuff
   response.resume(stuff)
```

```
@Inject
CatRepository catRepo;
  ...

Set<Cat> cats = catRepo.getAllCats();
```

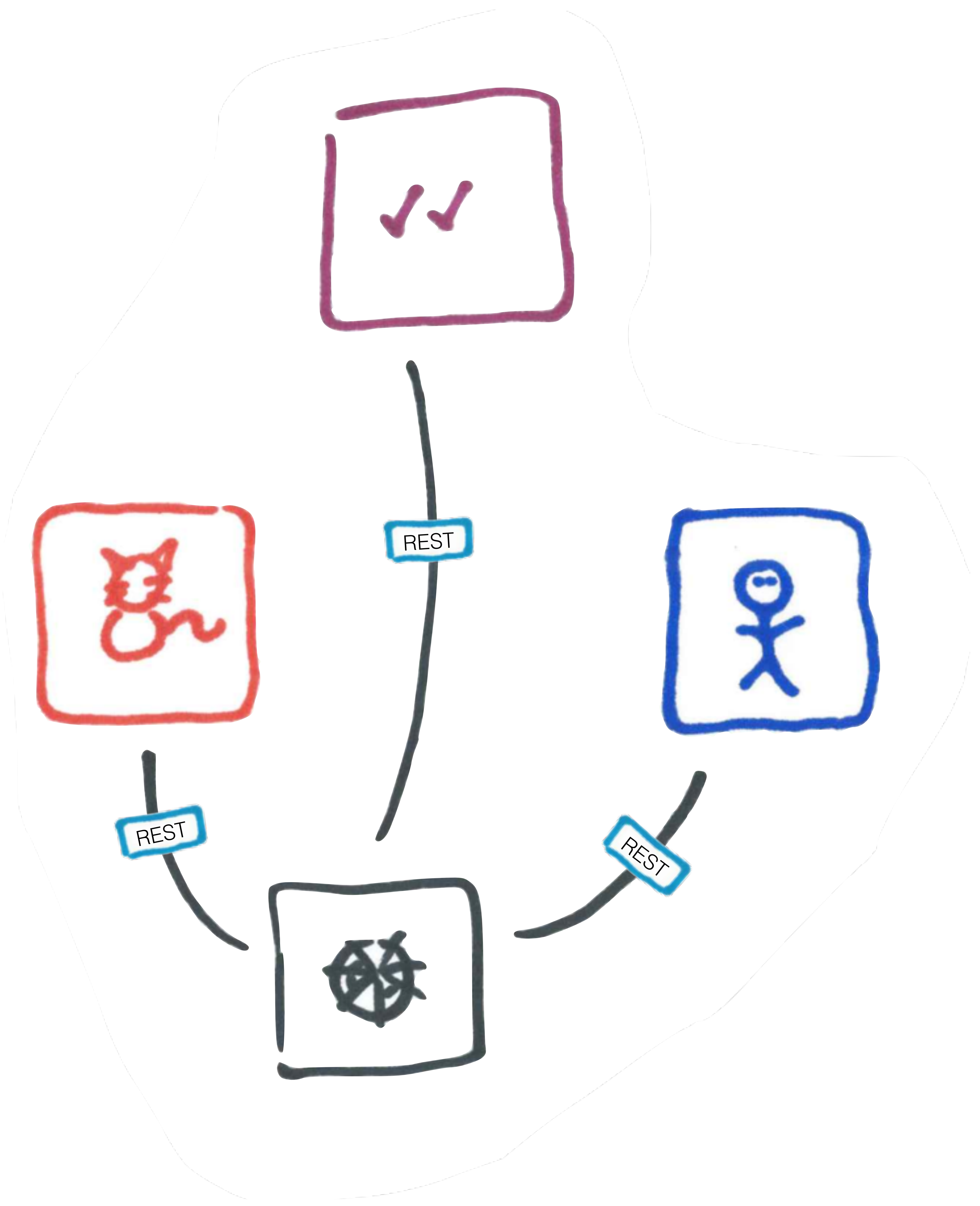
Consuming a service in a monolith

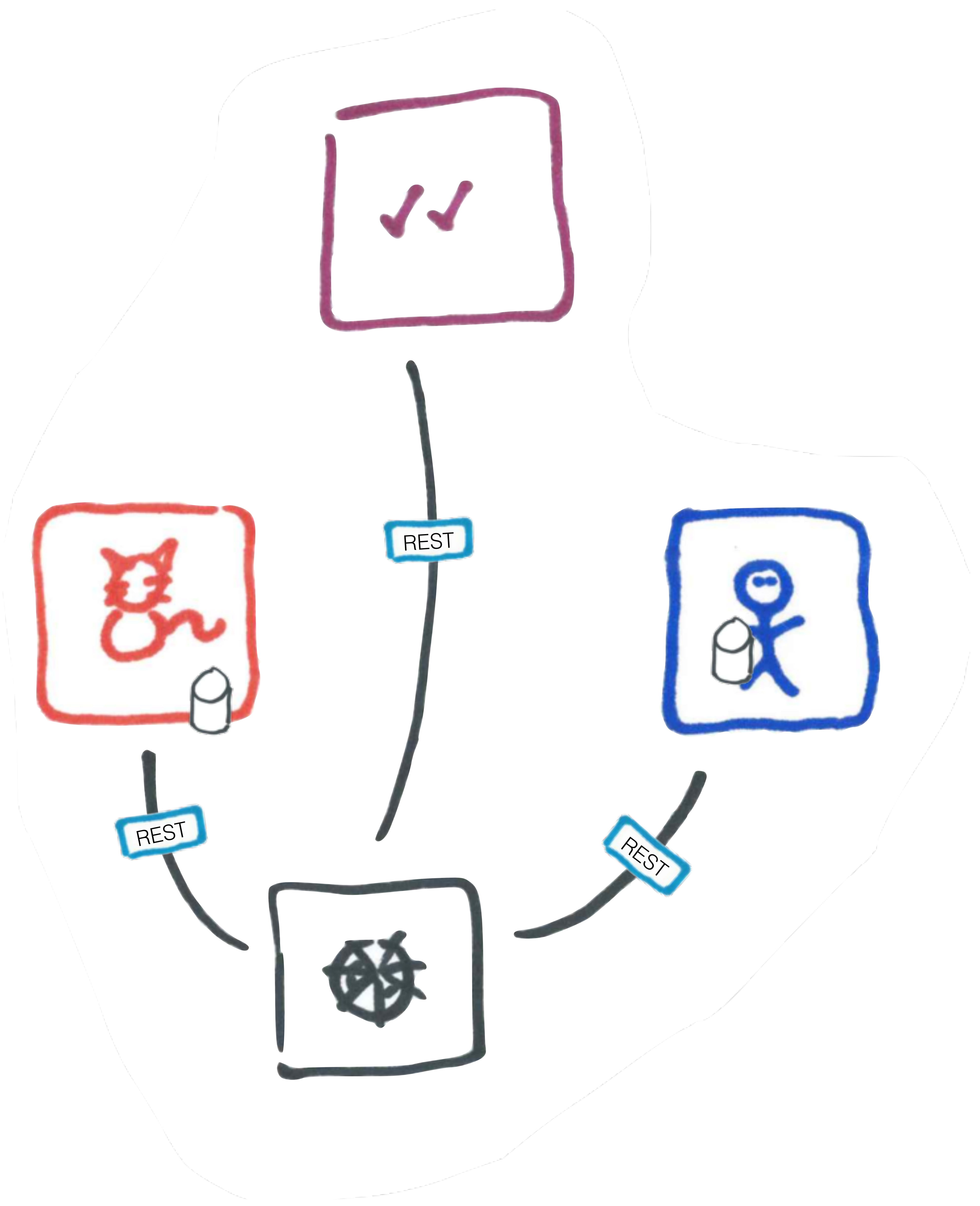
```
Client client = ClientBuilder.newClient();
WebTarget target = client.target("http://localhost:9080")
      .path("/rest/cat/cats");
Set<Cat> cats = target.request(MediaType.APPLICATION_JSON)
      .get(new GenericType<>(Set.class));
```

Consuming a REST microservice



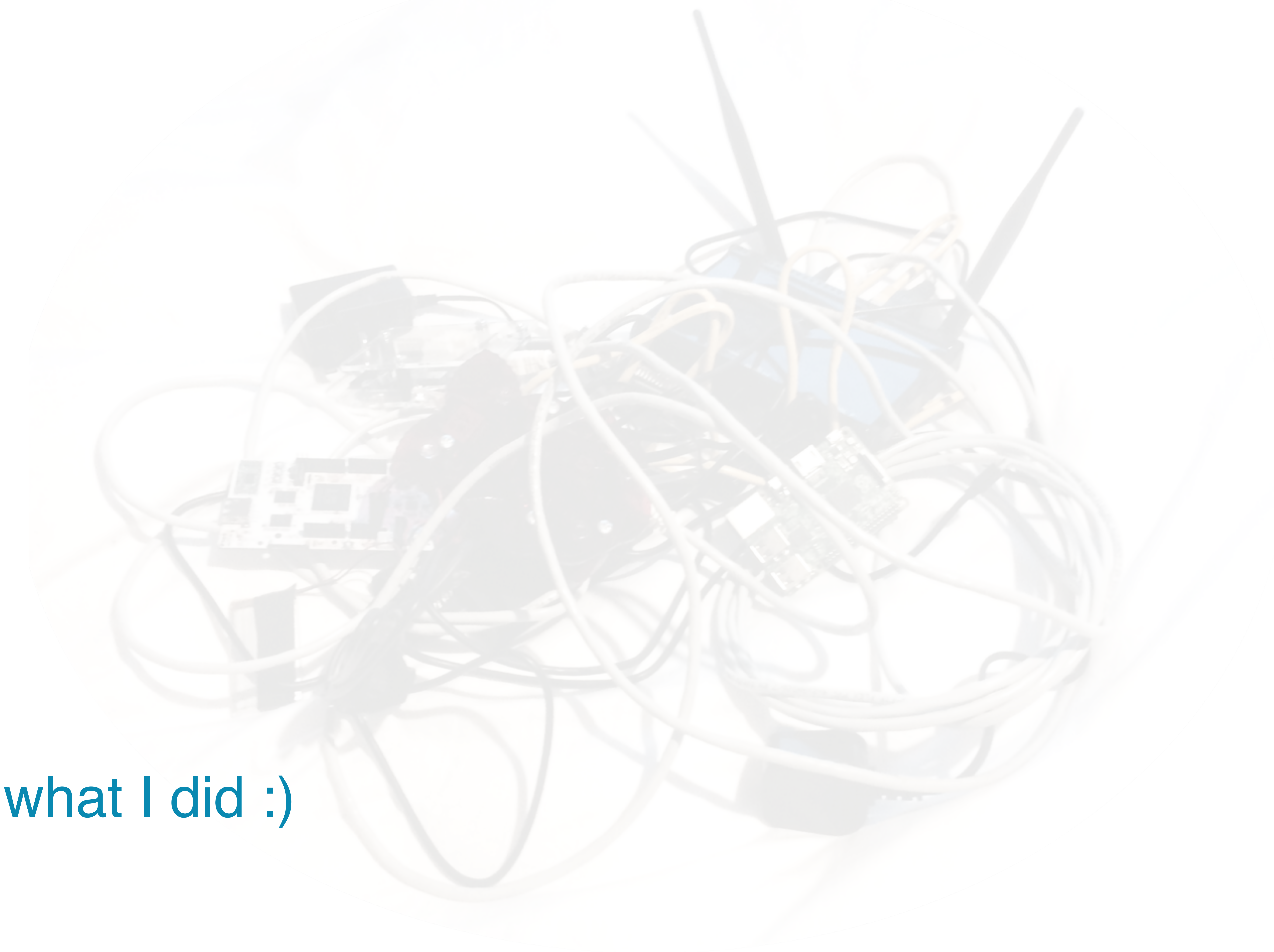
Don’t forget to slice up the database too


REST
REST
REST

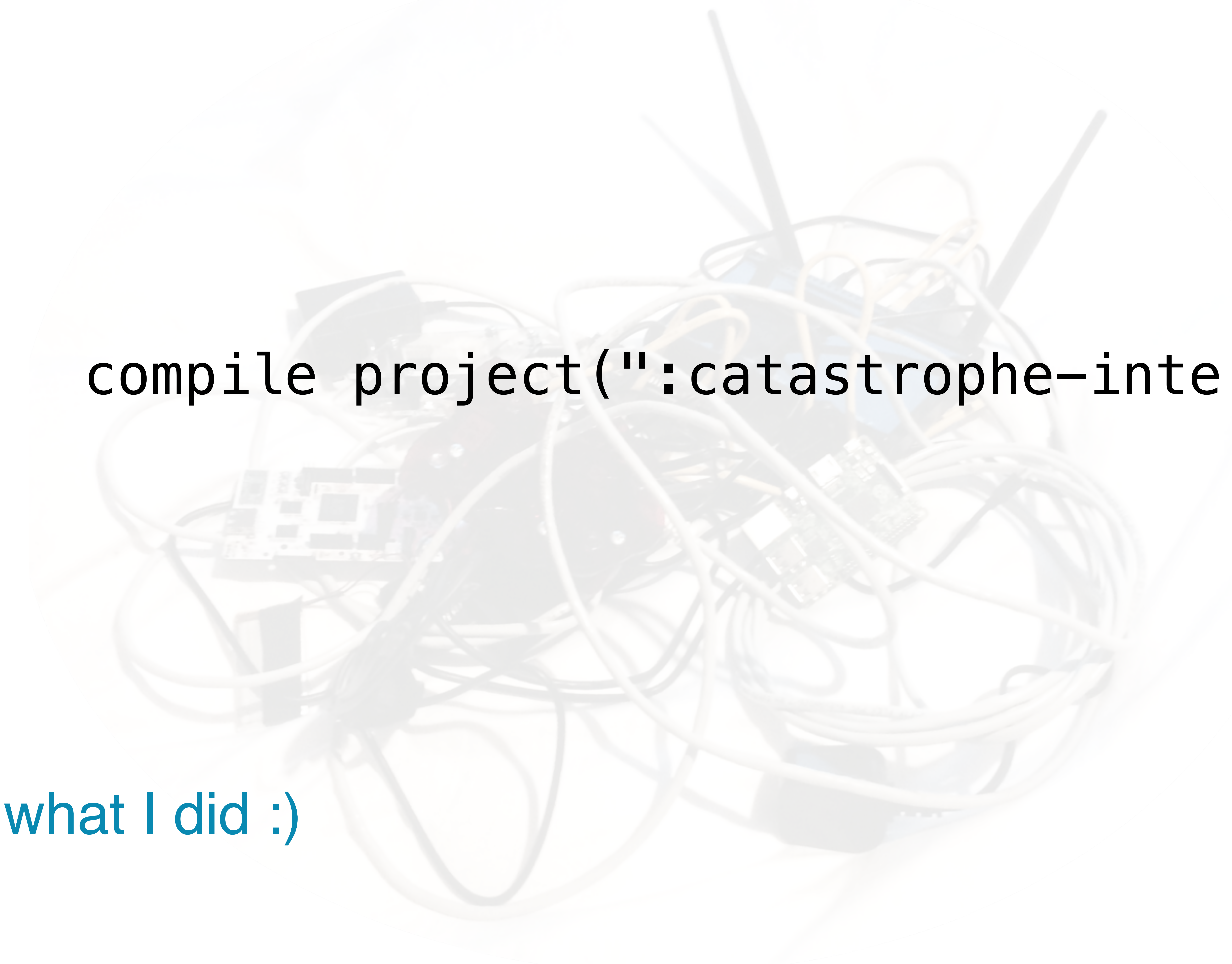

REST
REST
REST

Are we done?

Don’t forget to slice up the data *model* too

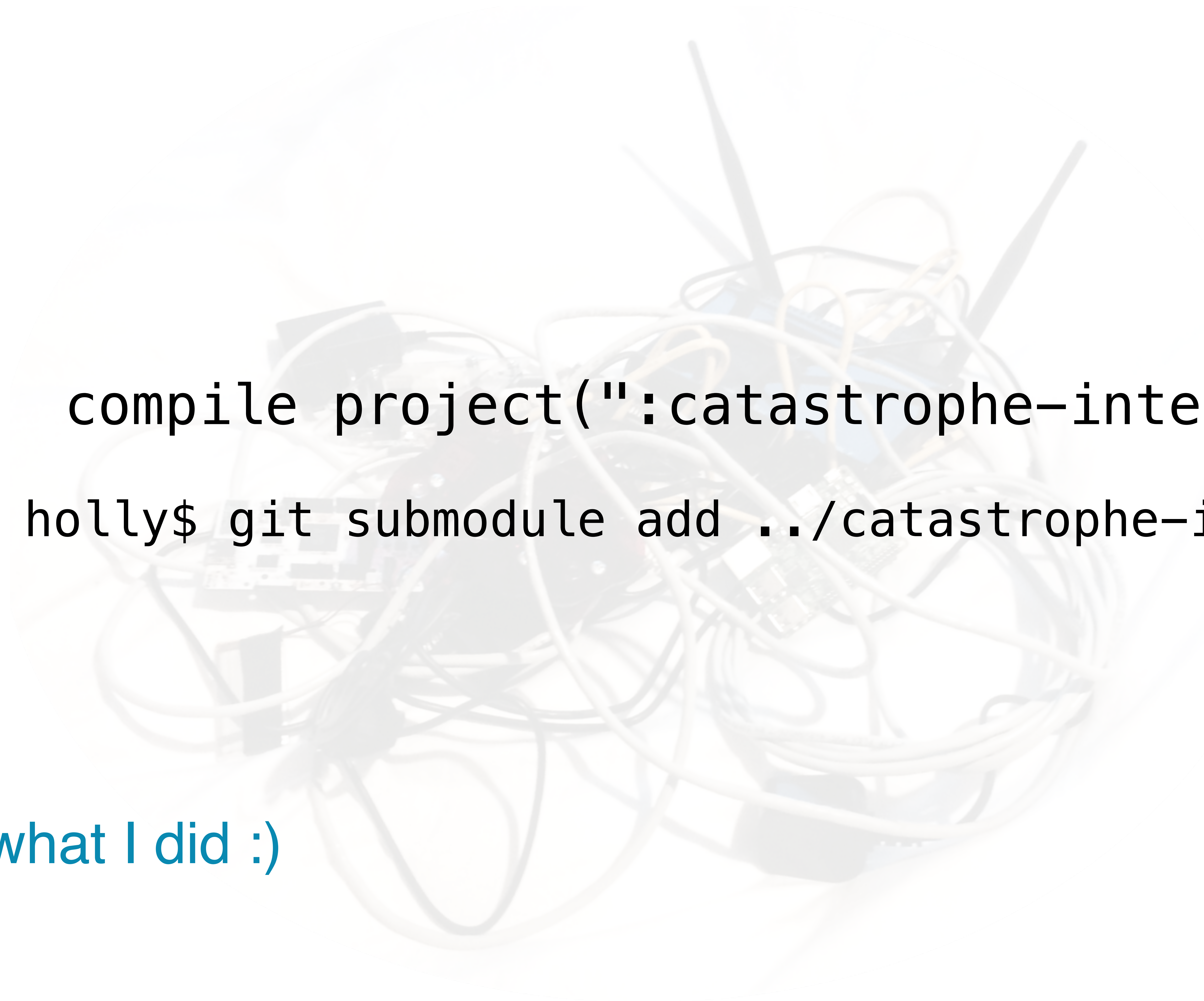


Don’t do what I did :)



```
compile project(":catastrophe-interfaces")
```

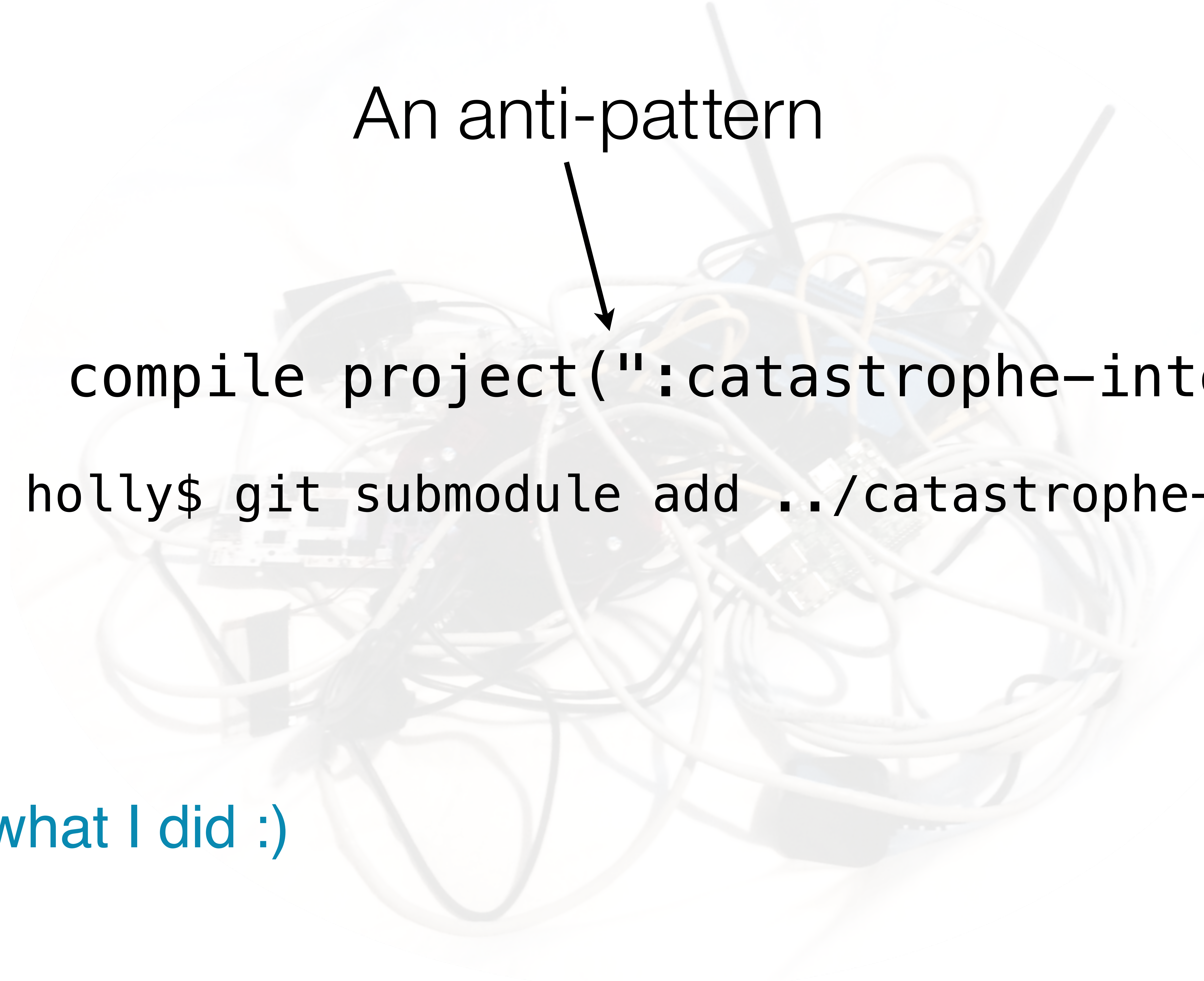
Don’t do what I did :)



```
compile project(":catastrophe-interfaces")
```

```
mymac:~ holly$ git submodule add ../catastrophe-interfaces
```

Don’t do what I did :)

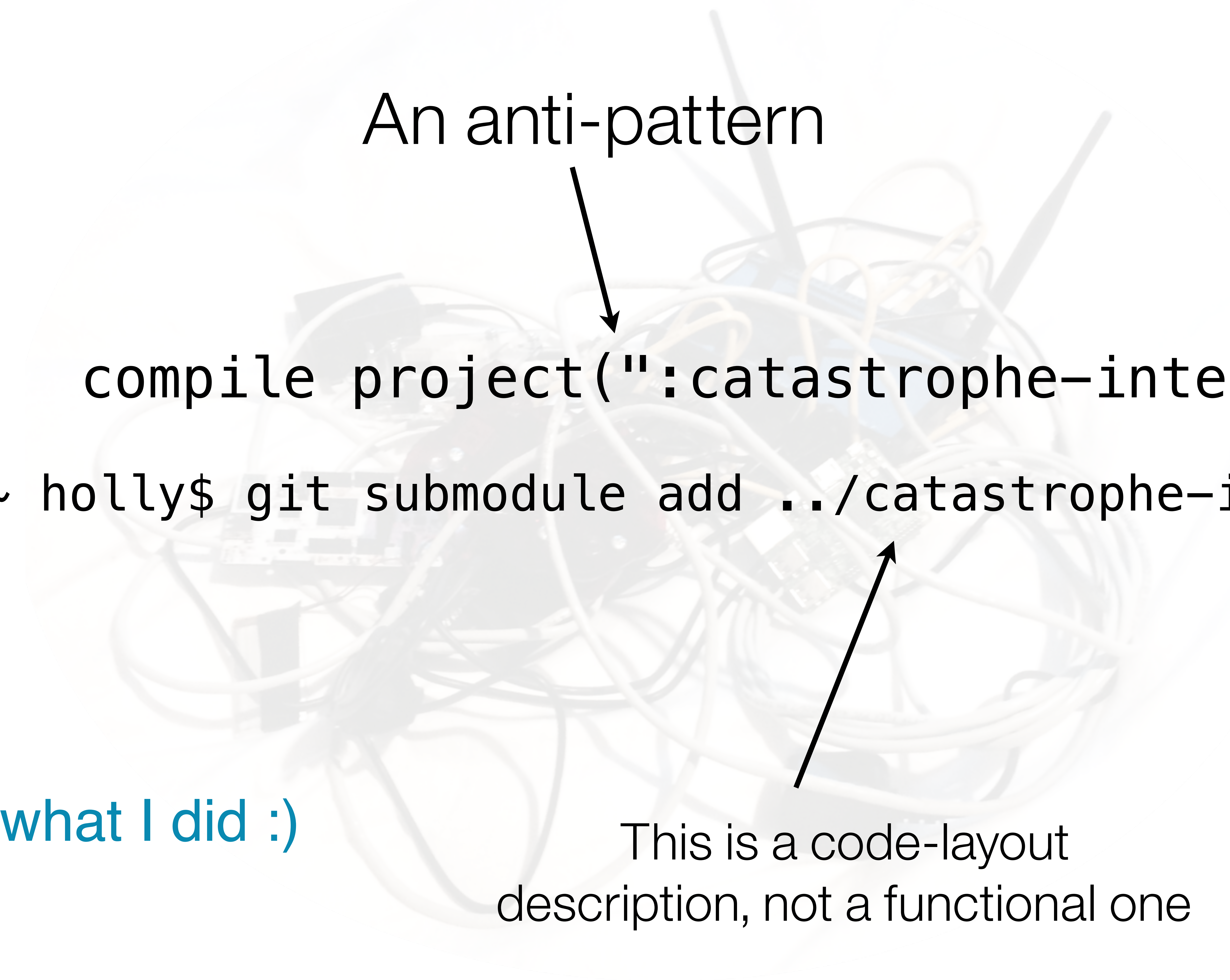


An anti-pattern

```
compile project(":catastrophe-interfaces")
```

```
mymac:~ holly$ git submodule add ../catastrophe-interfaces
```

Don’t do what I did :)



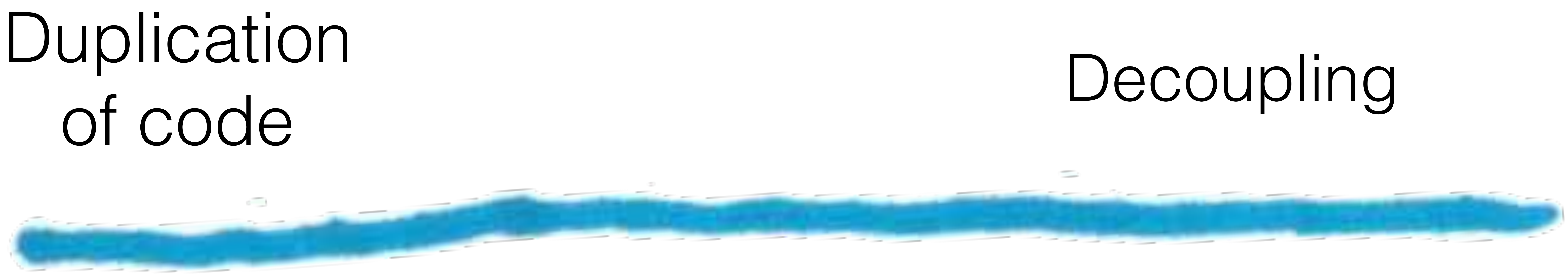
An anti-pattern

```
compile project(":catastrophe-interfaces")
```

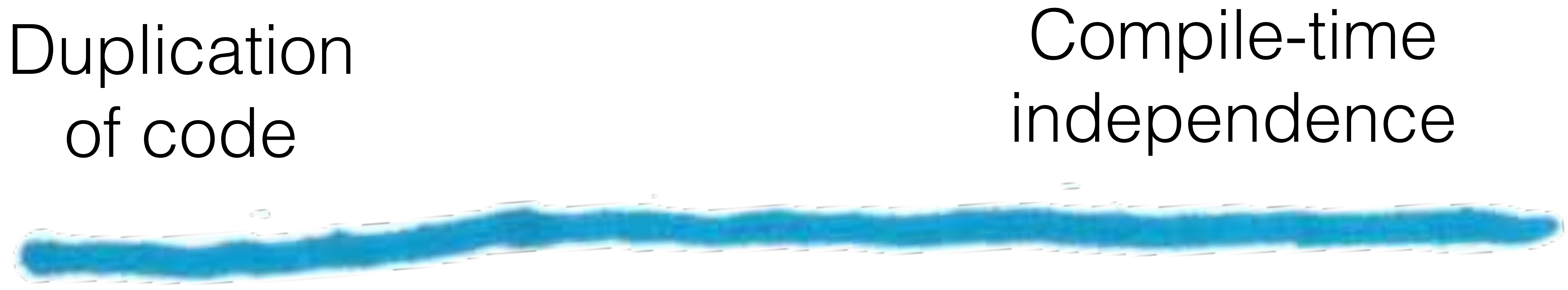
```
mymac:~ holly$ git submodule add ../catastrophe-interfaces
```

Don’t do what I did :)

This is a code-layout description, not a functional one



The tradeoff



The tradeoff

If this tradeoff is hurting, your domain model is too coupled.

If this tradeoff is hurting, your domain model is too coupled.

Have your microservices got the right granularity?

“Does this domain model make sense?”

“Does this domain model make sense?”

“Not really.”

“Does this domain model make sense?”

“Not really.”

“Does decomposing a system of this size into microservices actually make sense?”

“Does this domain model make sense?”

“Not really.”

“Does decomposing a system of this size into microservices actually make sense?”

“Well, no.”

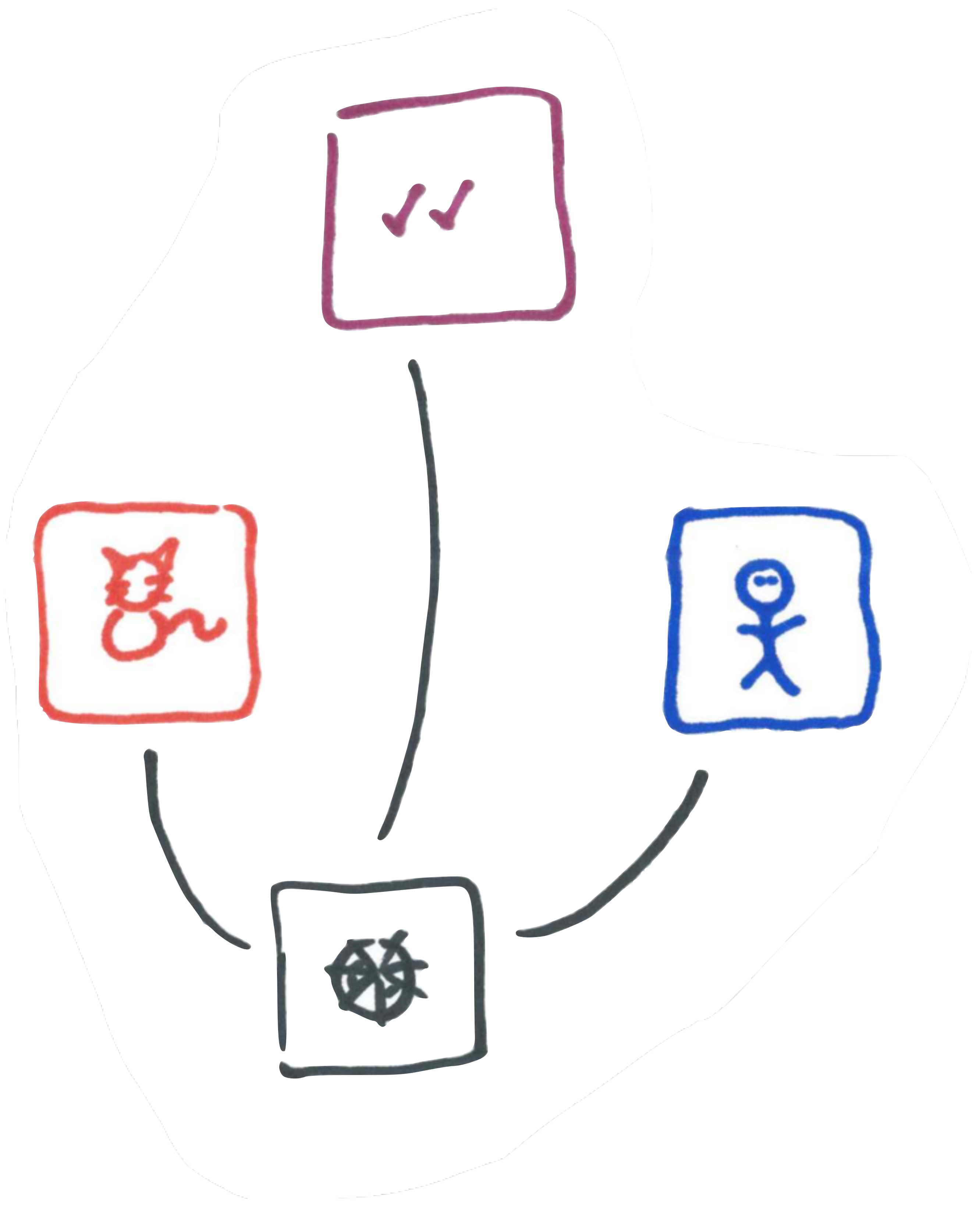
“Does this domain model make sense?”

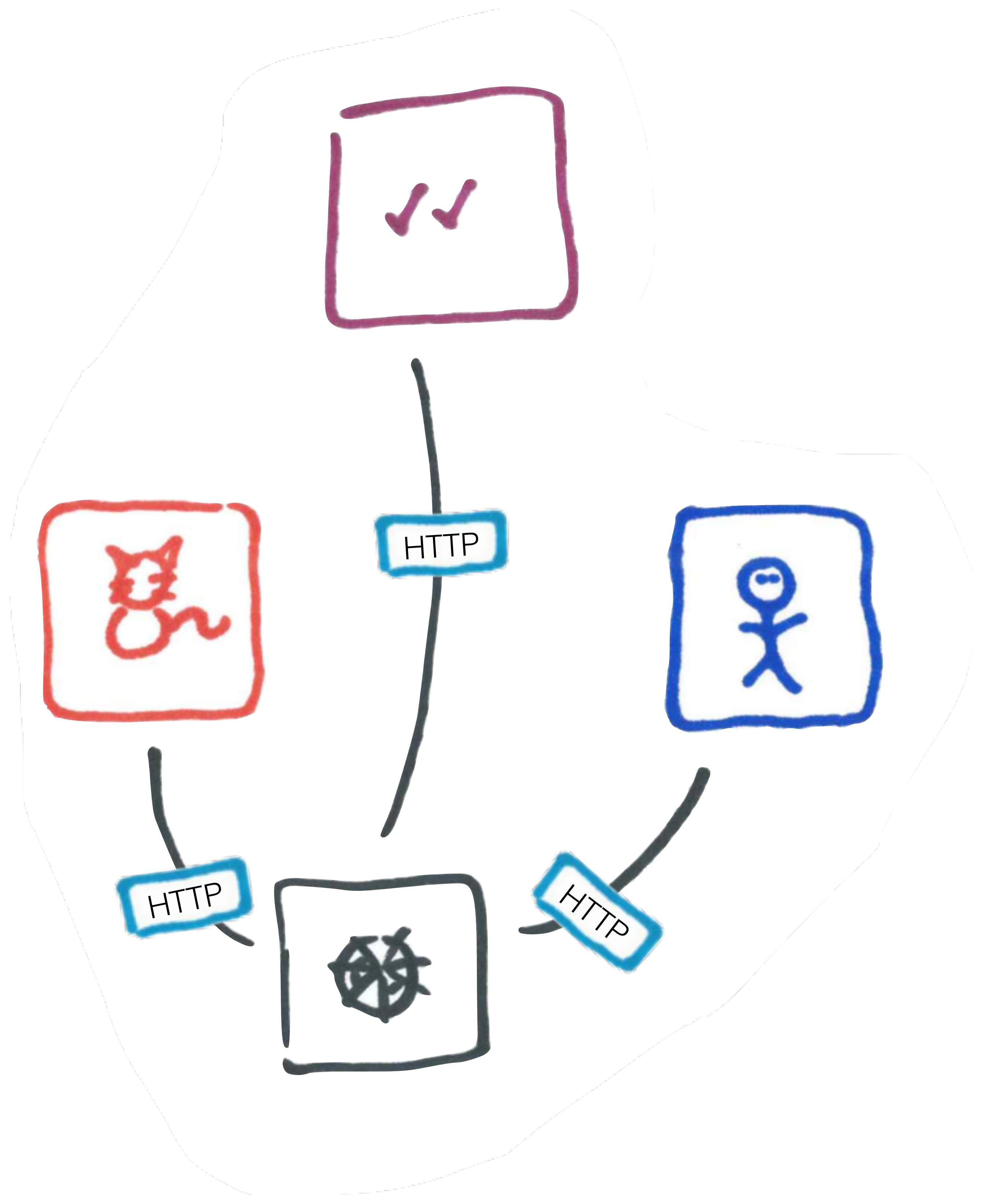
“Not really.”

“Does decomposing a system of this size into microservices actually make sense?”

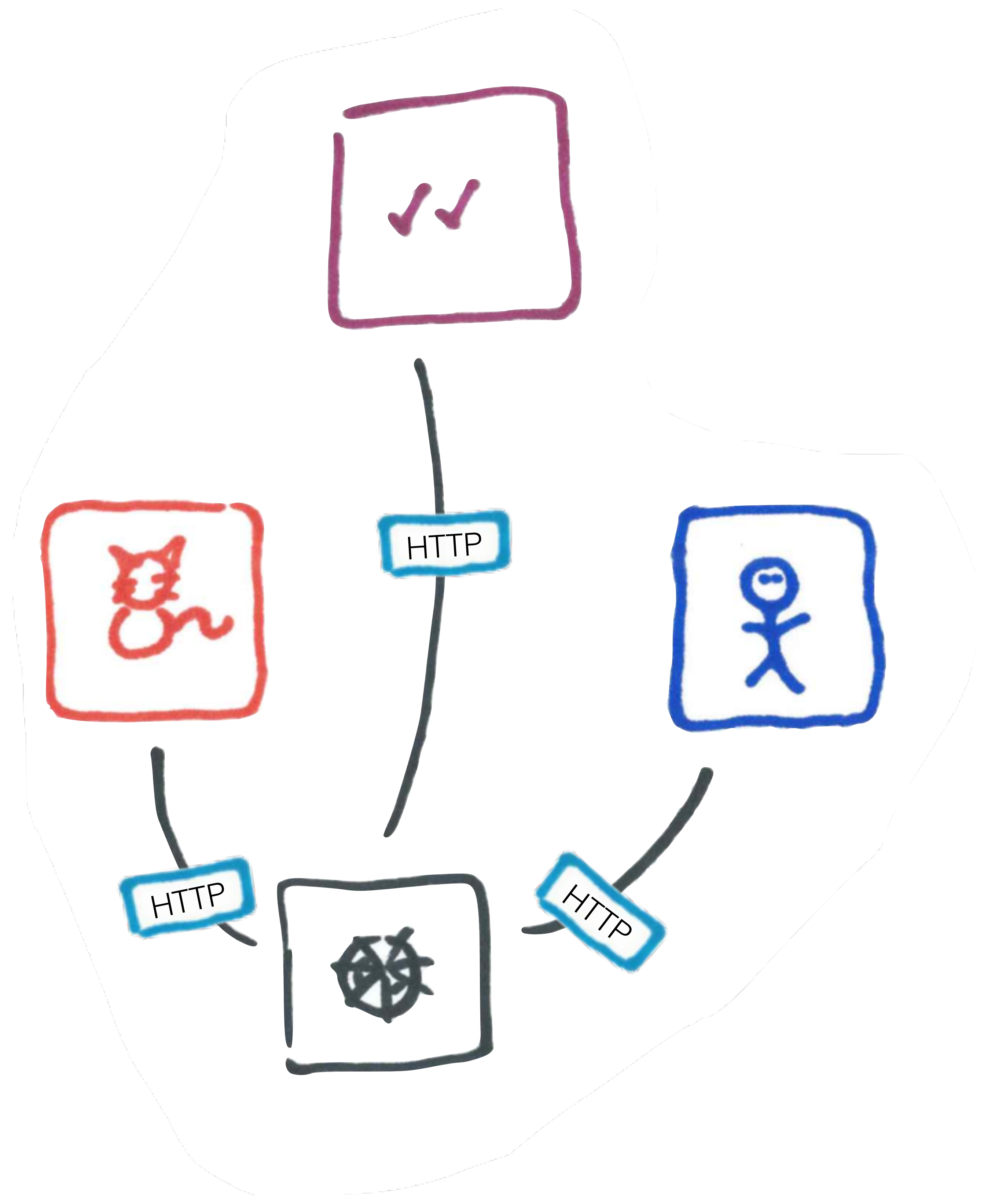
“Well, no.”

The right granularity may be “monolith.”



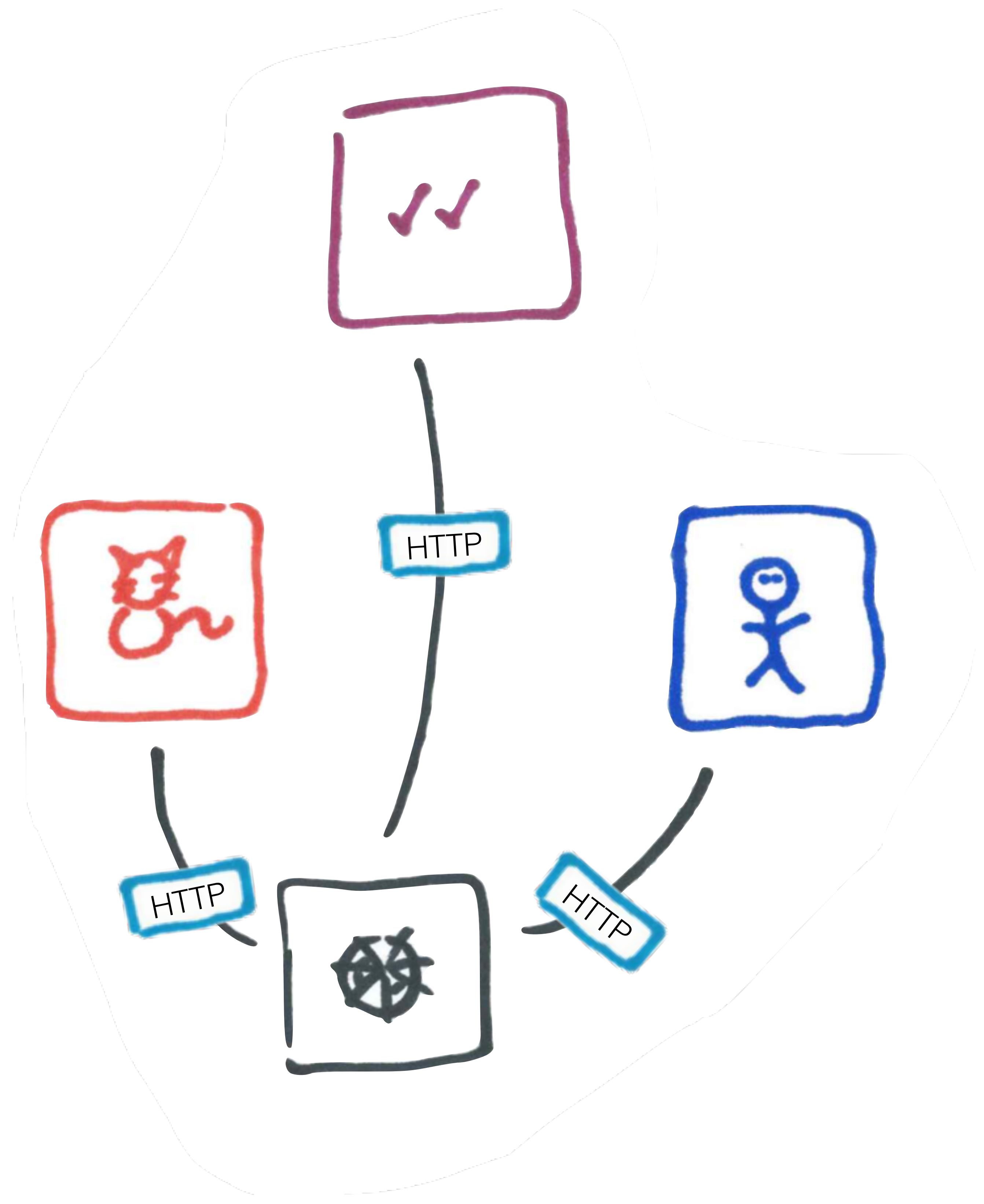

HTTP
HTTP
HTTP

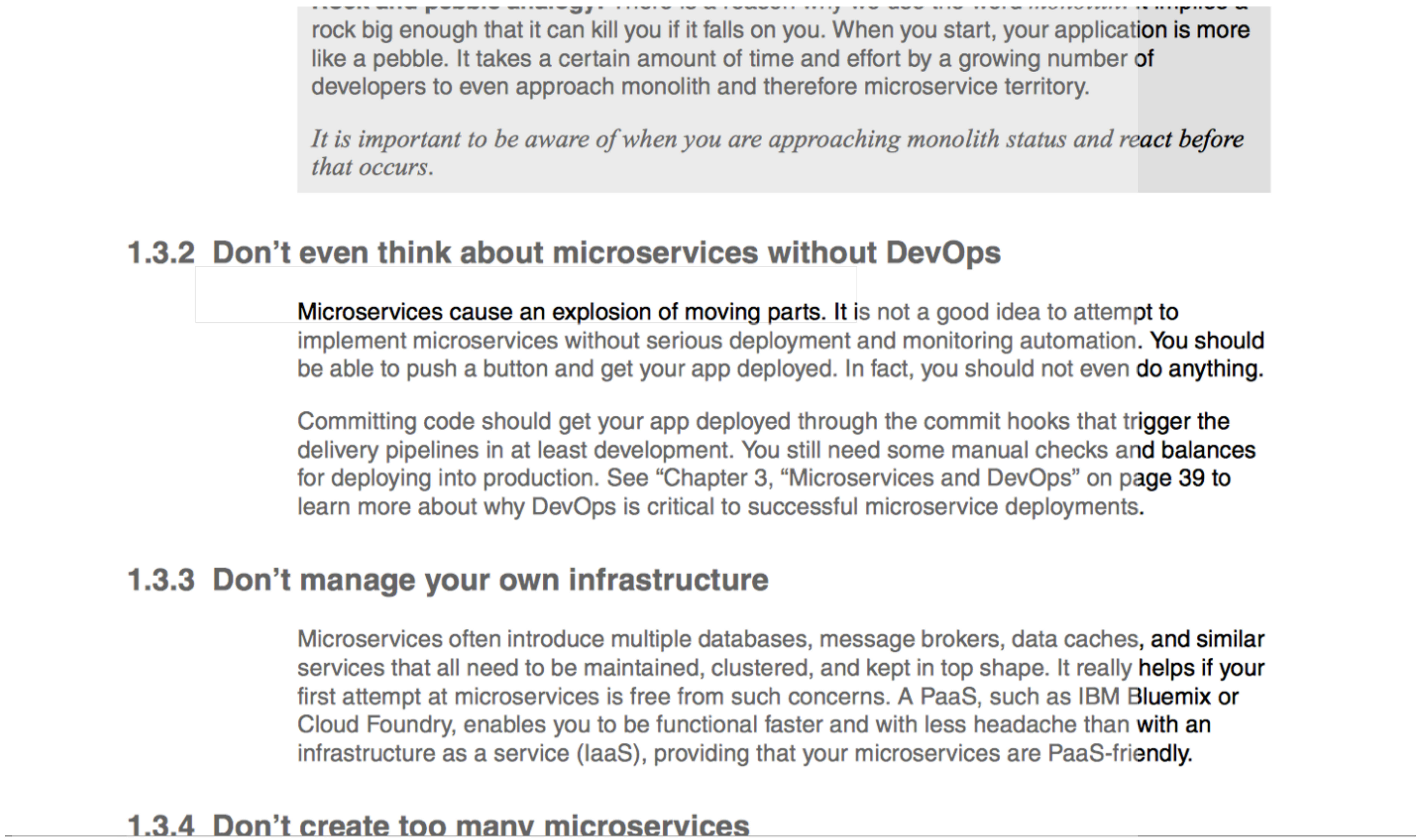
Remember the distributed computing fallacies.



Really

Remember the distributed computing fallacies.





rock big enough that it can kill you if it falls on you. When you start, your application is more like a pebble. It takes a certain amount of time and effort by a growing number of developers to even approach monolith and therefore microservice territory.

It is important to be aware of when you are approaching monolith status and react before that occurs.

1.3.2 Don’t even think about microservices without DevOps

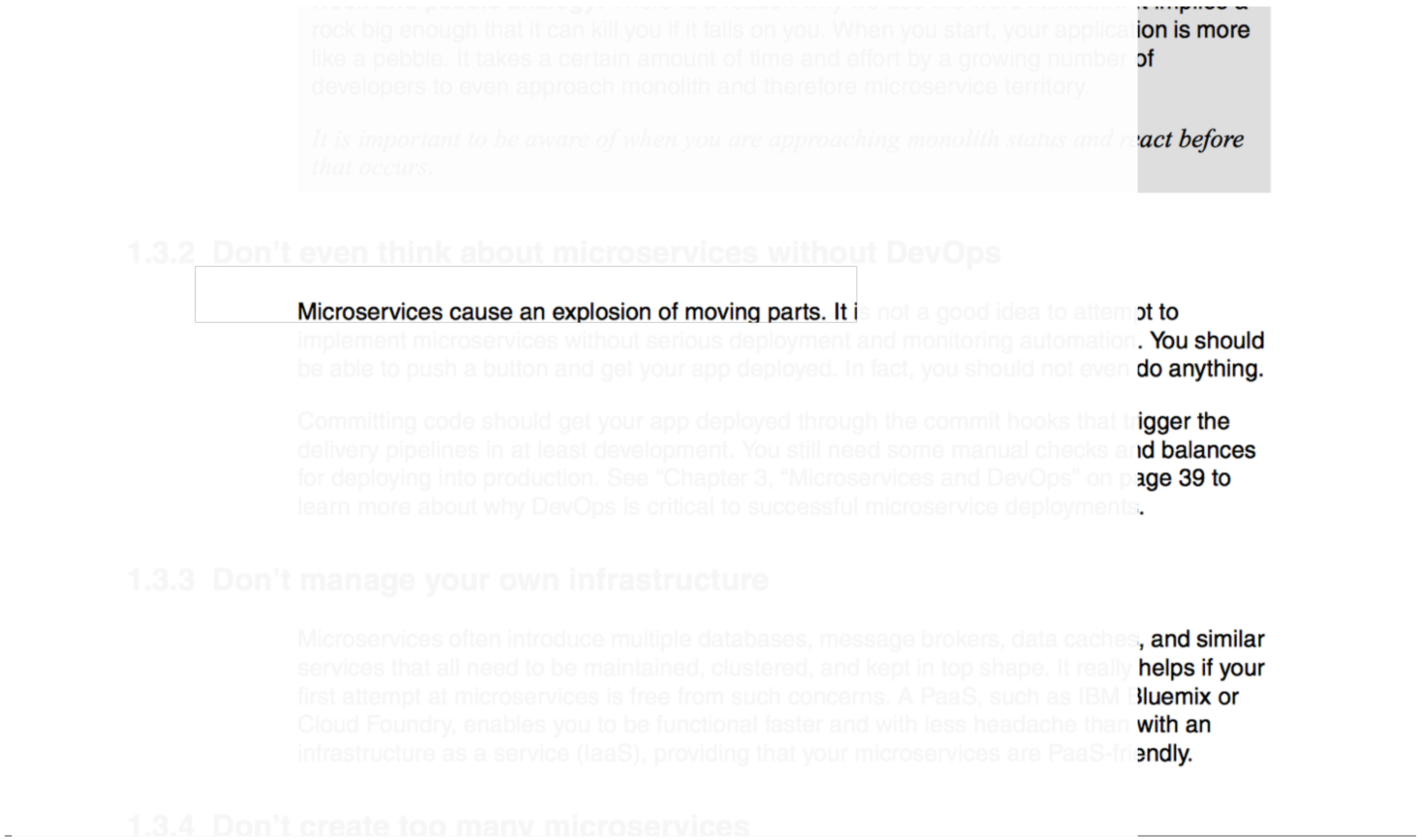
Microservices cause an explosion of moving parts. It is not a good idea to attempt to implement microservices without serious deployment and monitoring automation. You should be able to push a button and get your app deployed. In fact, you should not even do anything.

Committing code should get your app deployed through the commit hooks that trigger the delivery pipelines in at least development. You still need some manual checks and balances for deploying into production. See “Chapter 3, “Microservices and DevOps” on page 39 to learn more about why DevOps is critical to successful microservice deployments.

1.3.3 Don’t manage your own infrastructure

Microservices often introduce multiple databases, message brokers, data caches, and similar services that all need to be maintained, clustered, and kept in top shape. It really helps if your first attempt at microservices is free from such concerns. A PaaS, such as IBM Bluemix or Cloud Foundry, enables you to be functional faster and with less headache than with an infrastructure as a service (IaaS), providing that your microservices are PaaS-friendly.

1.3.4 Don’t create too many microservices



Rock and pebble analogy: There is a reason why we use the word monolith: it implies a rock big enough that it can kill you if it falls on you. When you start, your application is more like a pebble. It takes a certain amount of time and effort by a growing number of developers to even approach monolith and therefore microservice territory.

It is important to be aware of when you are approaching monolith status and react before that occurs.

1.3.2 Don't even think about microservices without DevOps

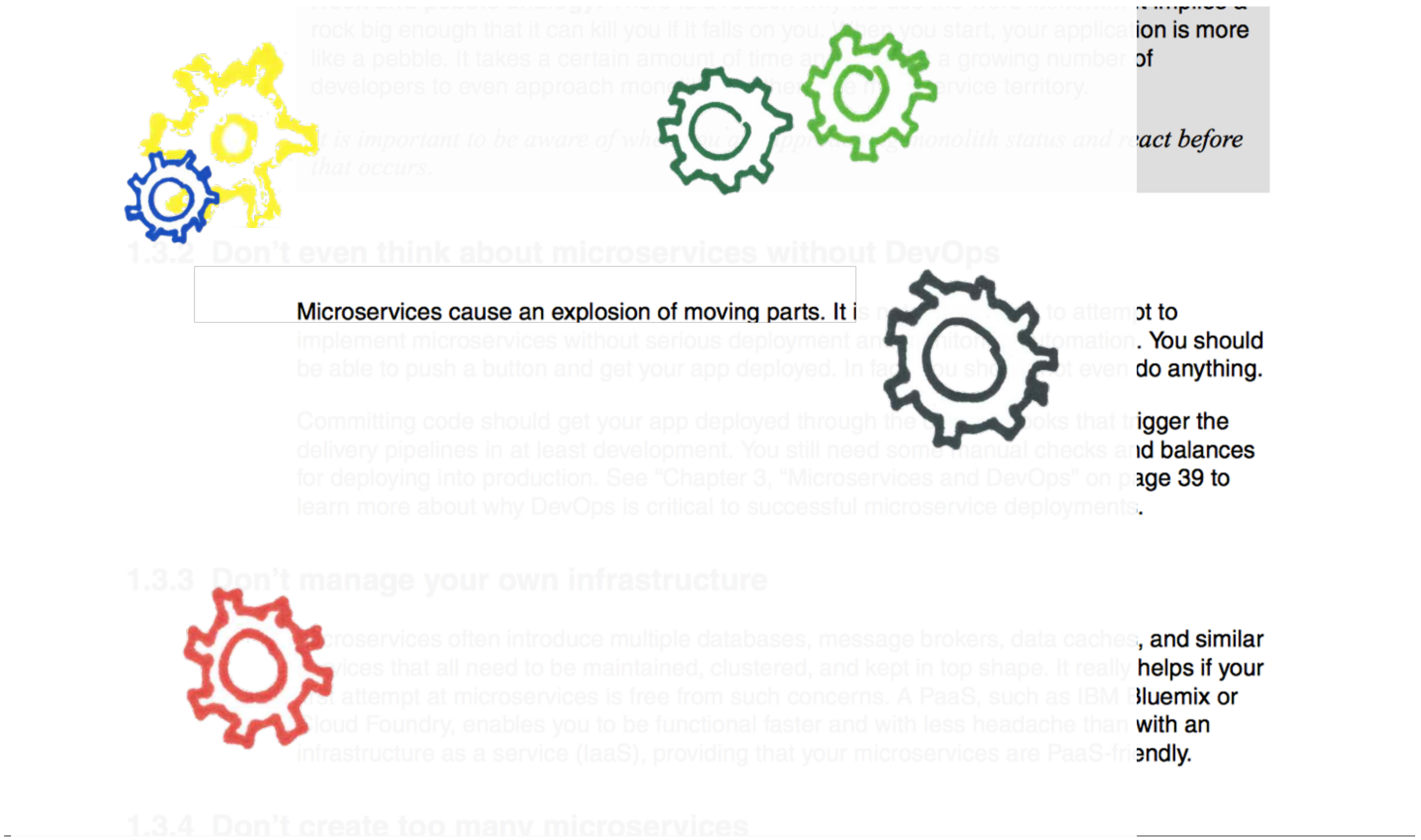
Microservices cause an explosion of moving parts. It is not a good idea to attempt to implement microservices without serious deployment and monitoring automation. You should be able to push a button and get your app deployed. In fact, you should not even do anything.

Committing code should get your app deployed through the commit hooks that trigger the delivery pipelines in at least development. You still need some manual checks and balances for deploying into production. See "Chapter 3, "Microservices and DevOps" on page 39 to learn more about why DevOps is critical to successful microservice deployments.

1.3.3 Don't manage your own infrastructure

Microservices often introduce multiple databases, message brokers, data caches, and similar services that all need to be maintained, clustered, and kept in top shape. It really helps if your first attempt at microservices is free from such concerns. A PaaS, such as IBM Bluemix or Cloud Foundry, enables you to be functional faster and with less headache than with an infrastructure as a service (IaaS), providing that your microservices are PaaS-friendly.

1.3.4 Don't create too many microservices



Rock and pebble analogy: There is a reason why we use the word monolith: it implies a rock big enough that it can kill you if it falls on you. When you start, your application is more like a pebble. It takes a certain amount of time and effort by a growing number of developers to even approach monolith and therefore microservice territory.

It is important to be aware of when you are approaching monolith status and react before that occurs.

1.3.2 Don't even think about microservices without DevOps

Microservices cause an explosion of moving parts. It is not a good idea to attempt to implement microservices without serious deployment and monitoring automation. You should be able to push a button and get your app deployed. In fact, you should not even do anything.

Committing code should get your app deployed through the commit hooks that trigger the delivery pipelines in at least development. You still need some manual checks and balances for deploying into production. See "Chapter 3, "Microservices and DevOps" on page 39 to learn more about why DevOps is critical to successful microservice deployments.

1.3.3 Don't manage your own infrastructure

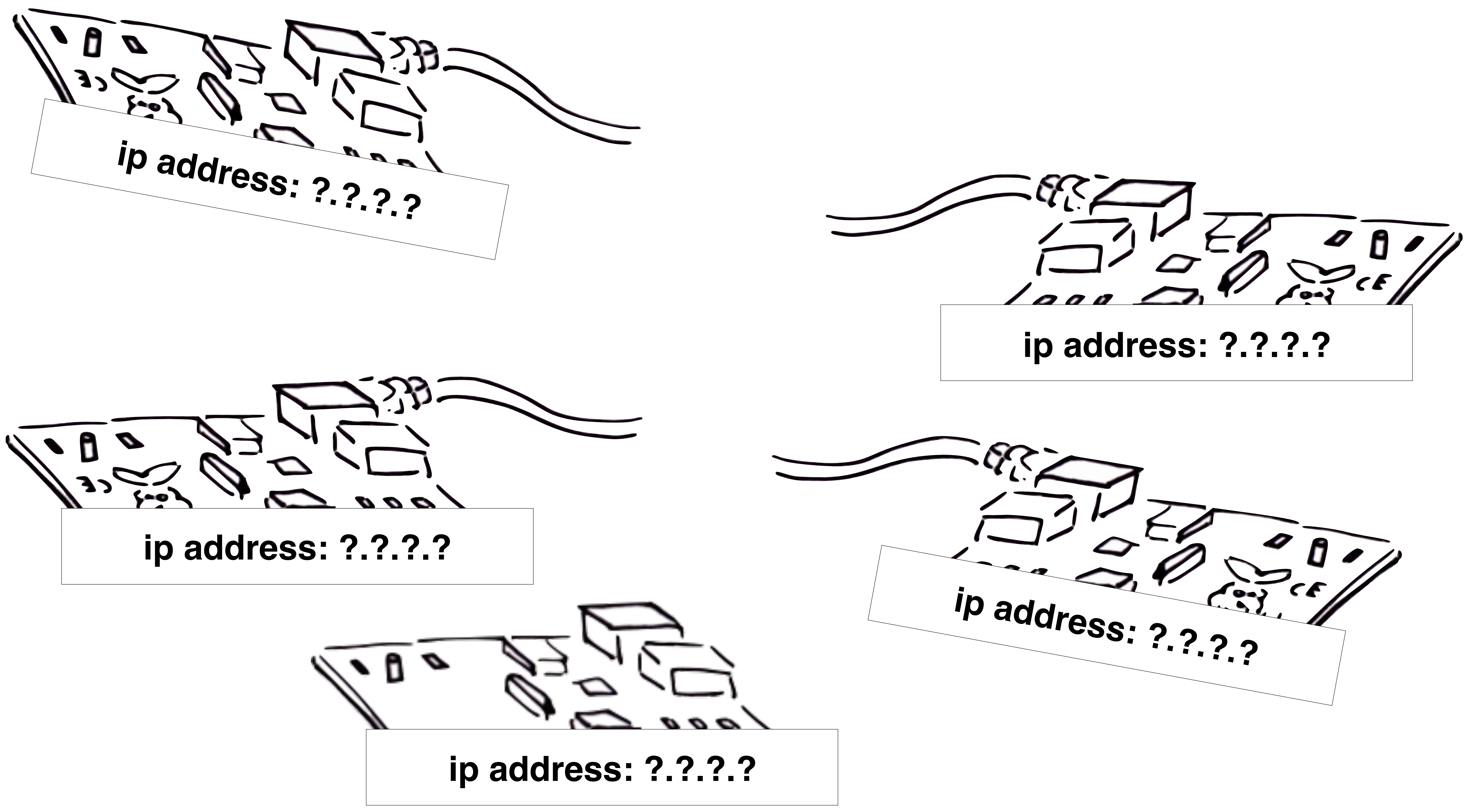
Microservices often introduce multiple databases, message brokers, data caches, and similar services that all need to be maintained, clustered, and kept in top shape. It really helps if your first attempt at microservices is free from such concerns. A PaaS, such as IBM Bluemix or Cloud Foundry, enables you to be functional faster and with less headache than with an infrastructure as a service (IaaS), providing that your microservices are PaaS-friendly.

1.3.4 Don't create too many microservices

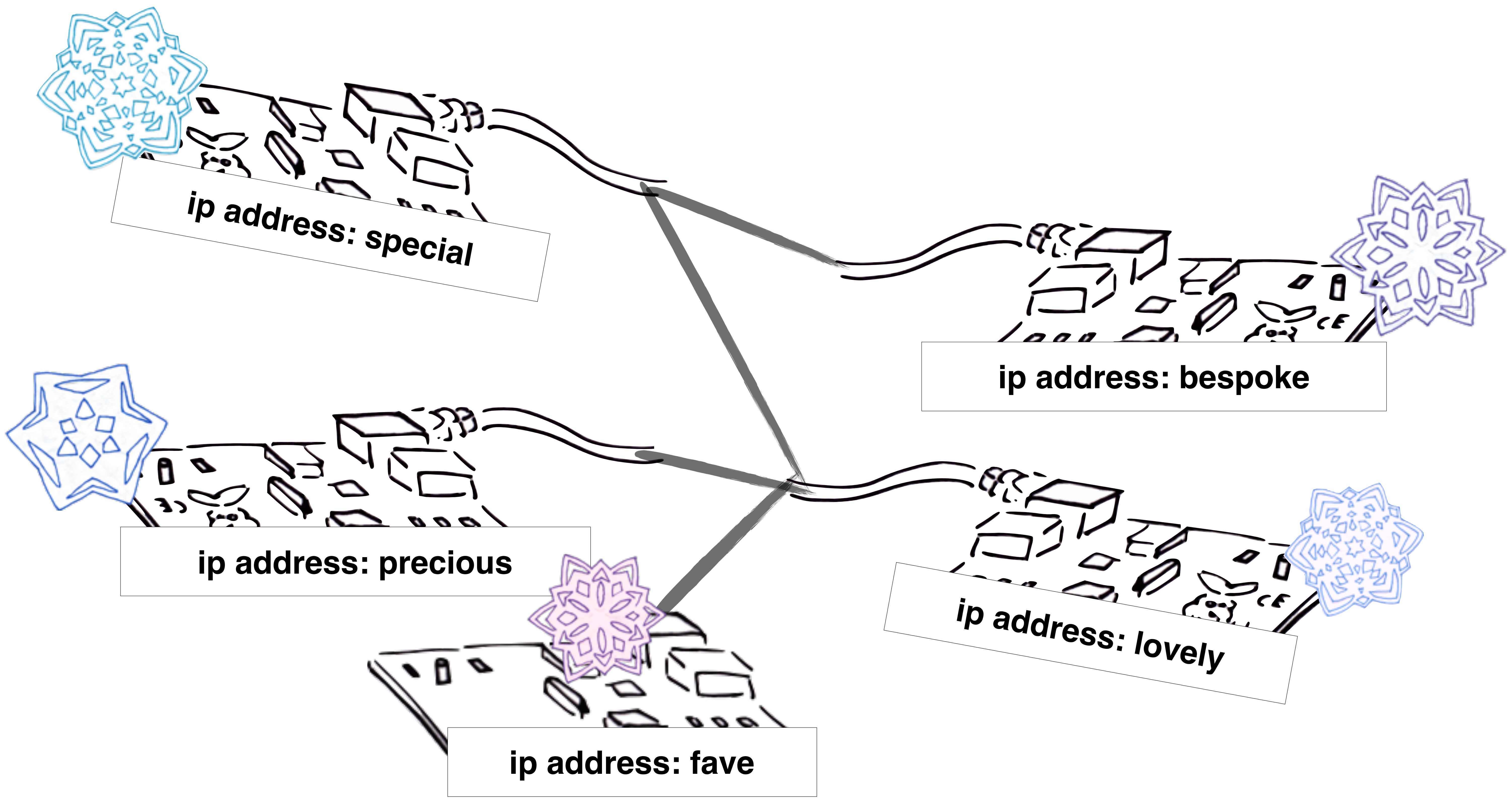
Complexity

```
WebTarget cat = client.target("http://raspberrypiclearcase.local");
WebTarget auth = client.target("http://raspberrypi2.local");
WebTarget scoring = client.target("http://raspberrypiredcase.local");
WebTarget cat = client.target(“http://raspberrypiclearcase.local”);
WebTarget auth = client.target(“http://raspberrypi2.local”);
WebTarget scoring = client.target(“http://raspberrypiredcase.local”);
```

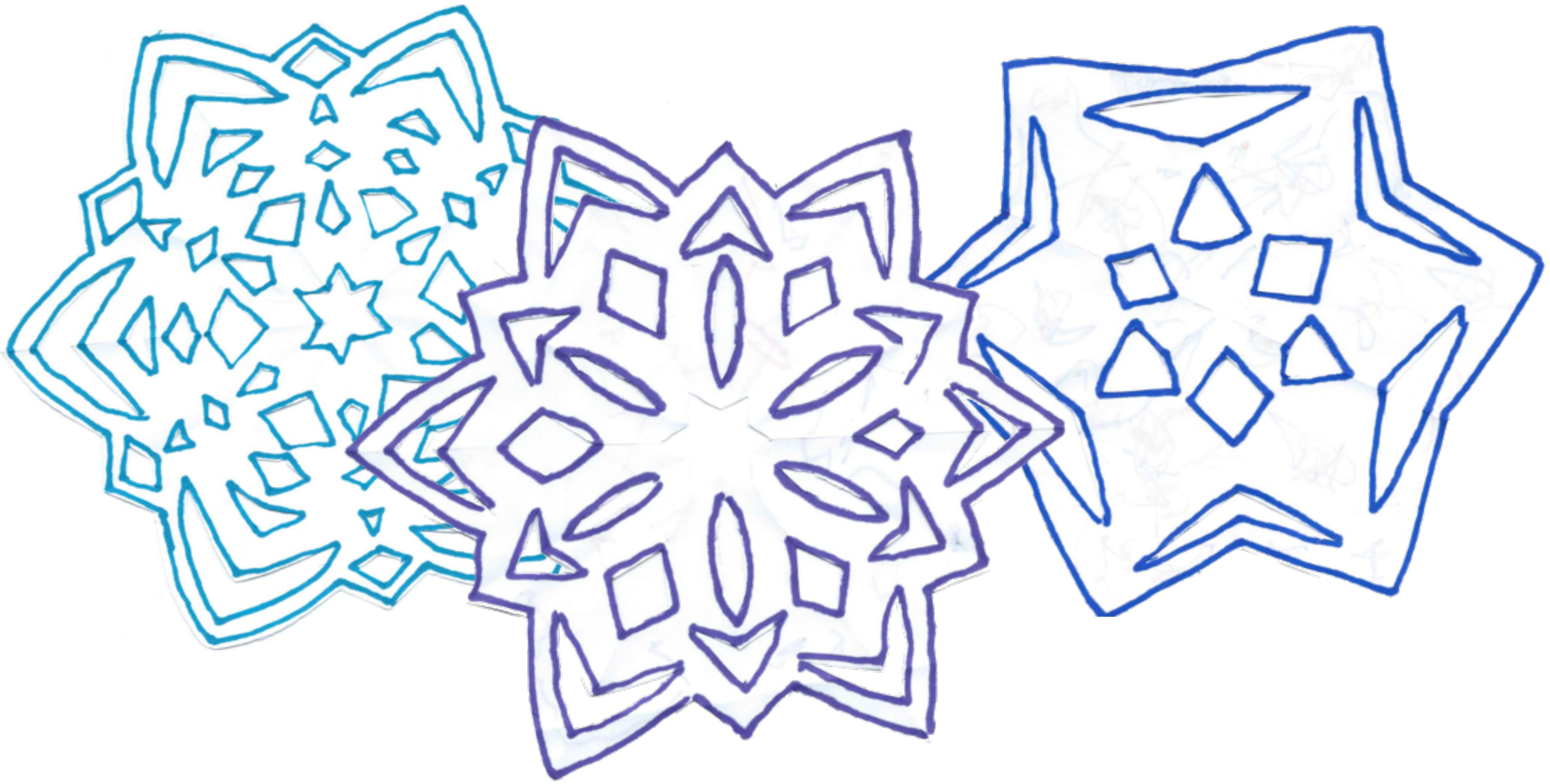
This is robust code, right?

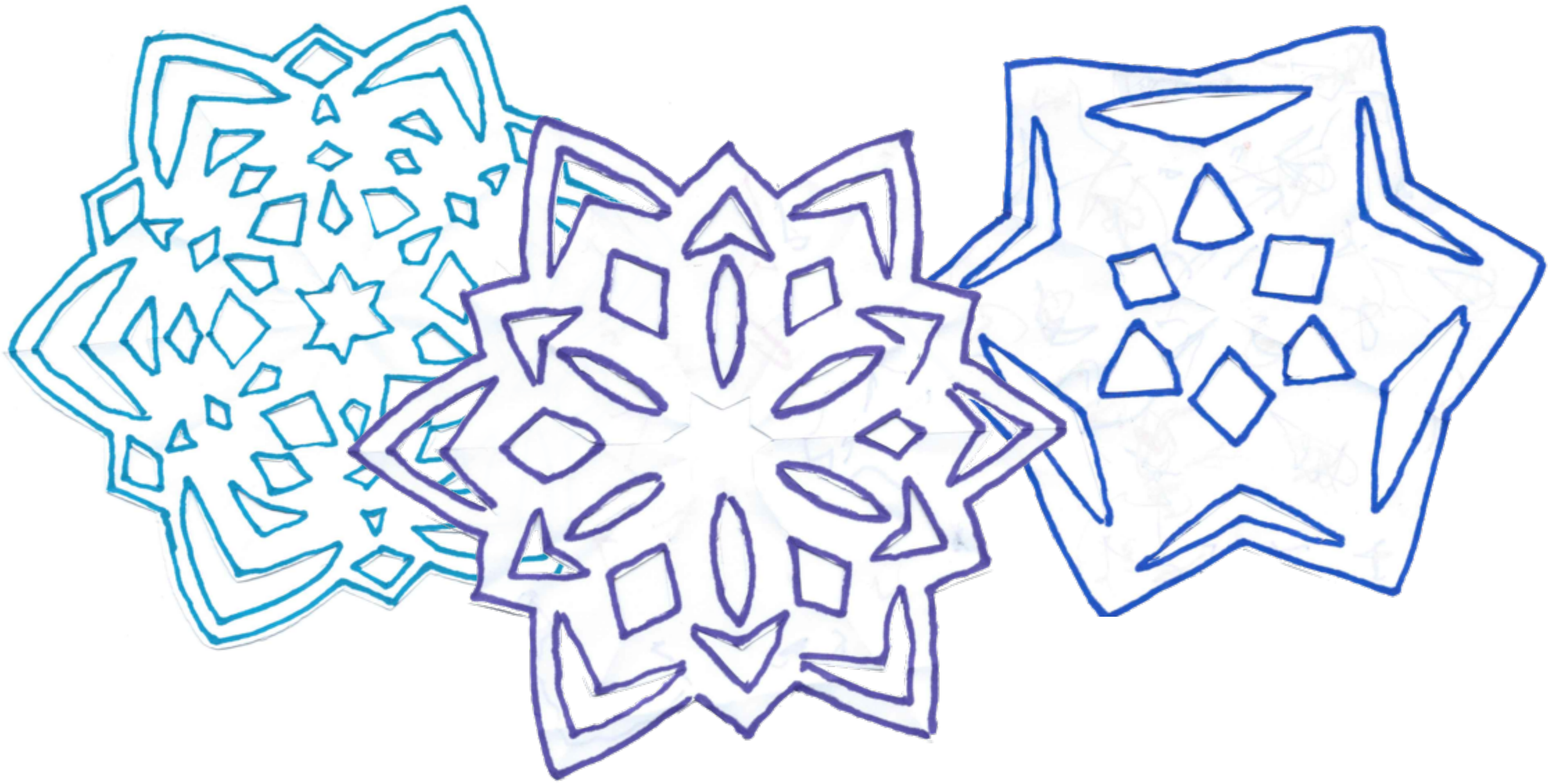


Network topology

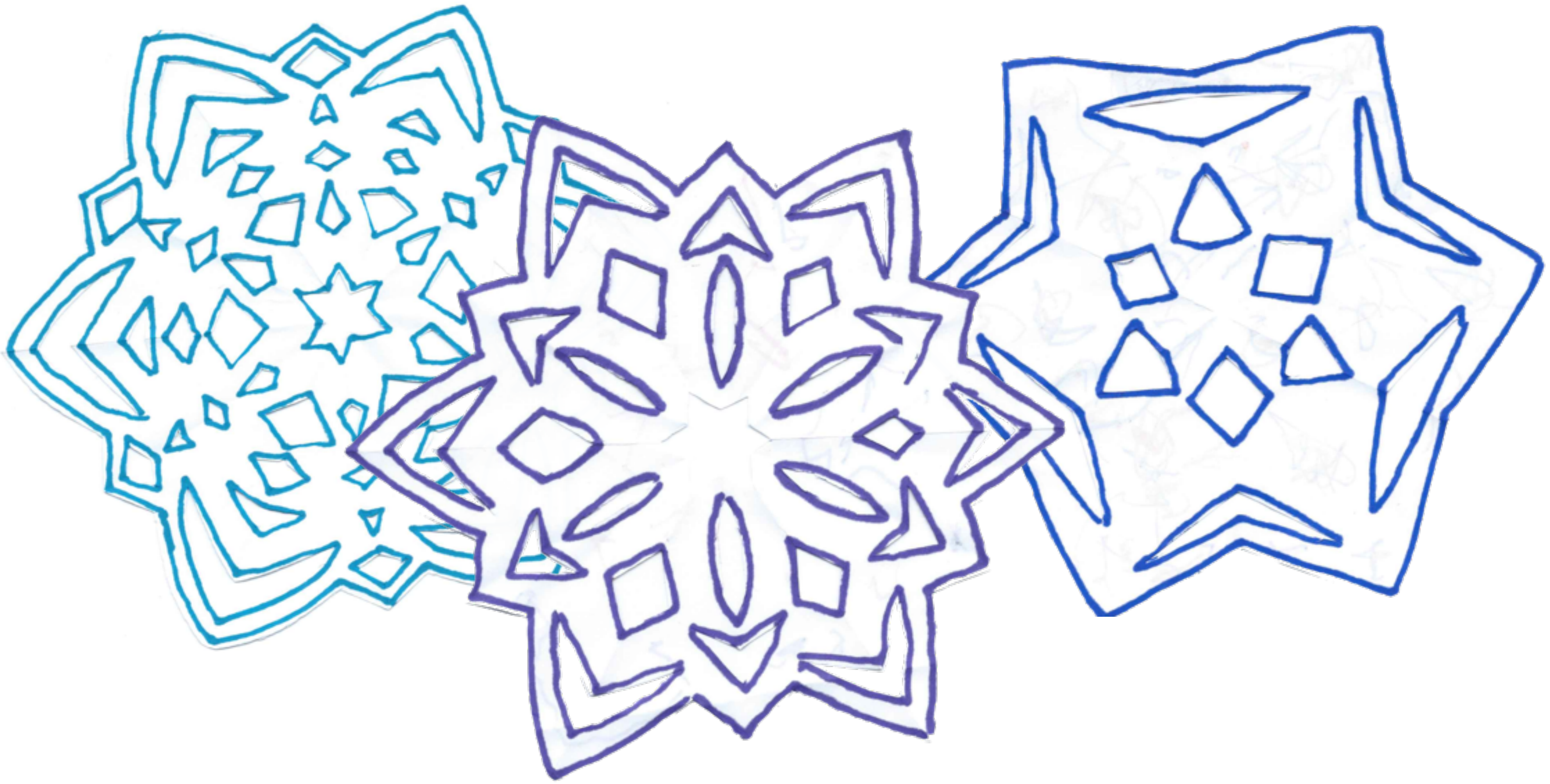


Network topology





Disposability

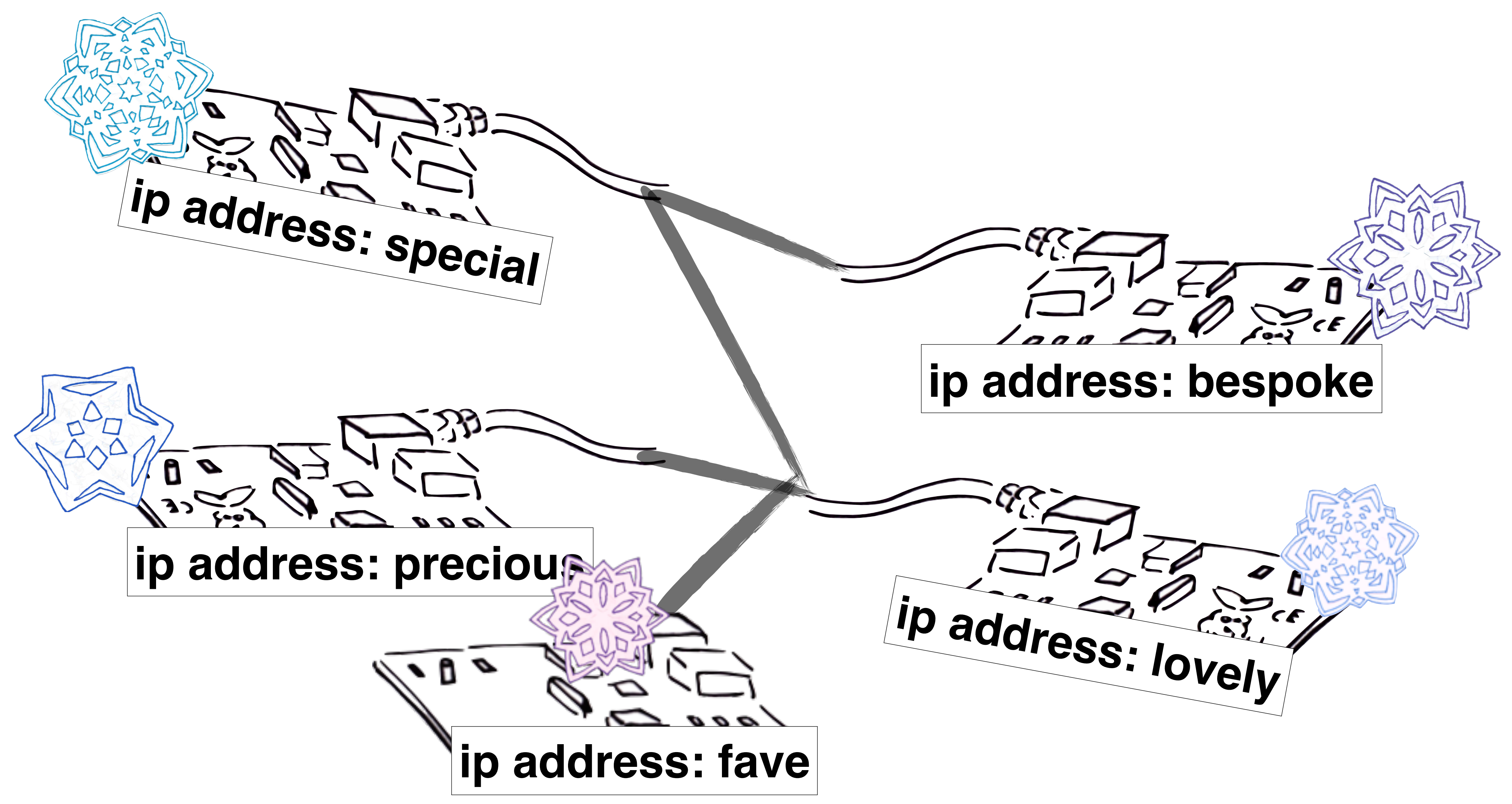


Disposability

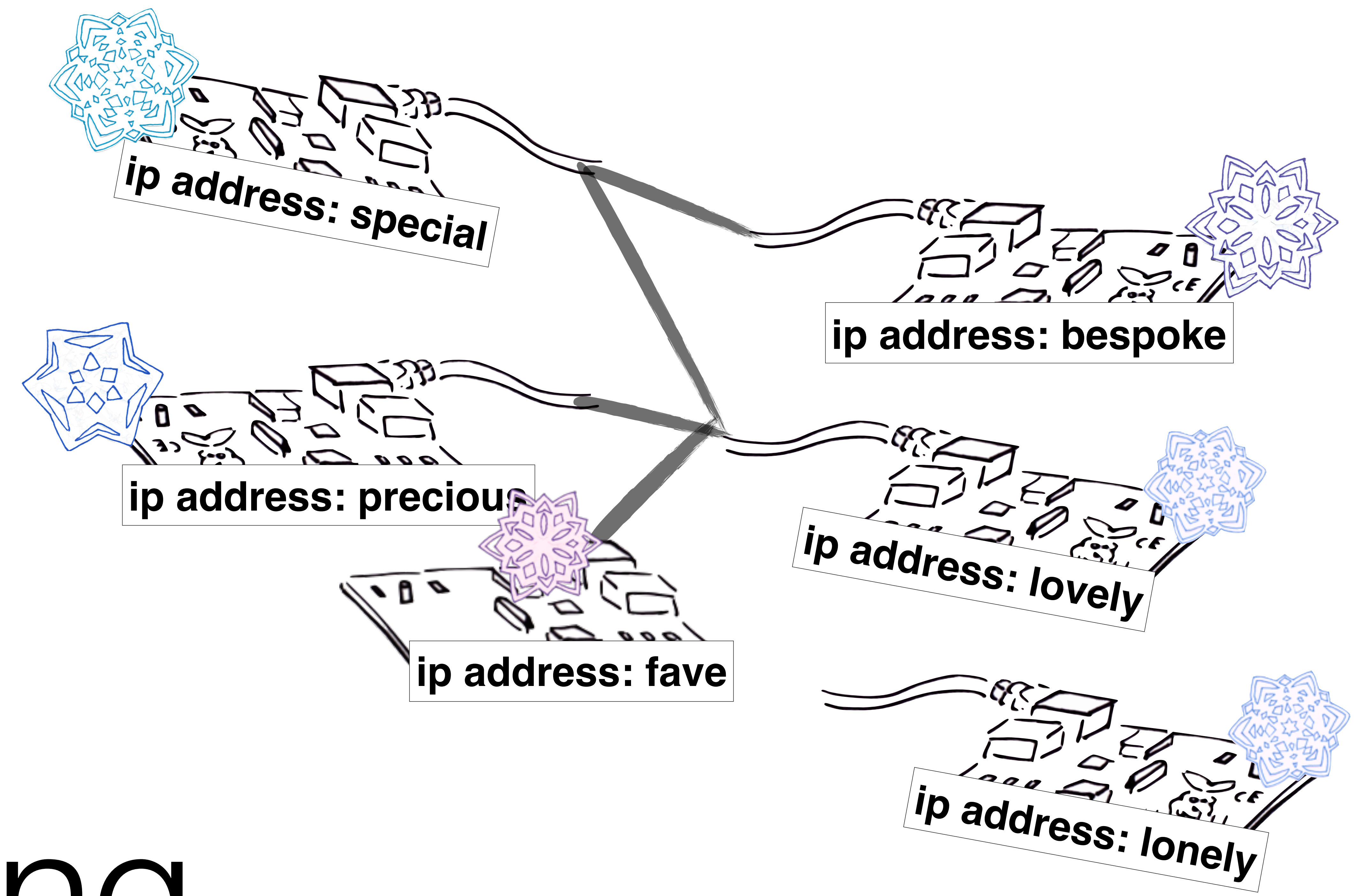
Say **no** to snowflake servers

Disposability

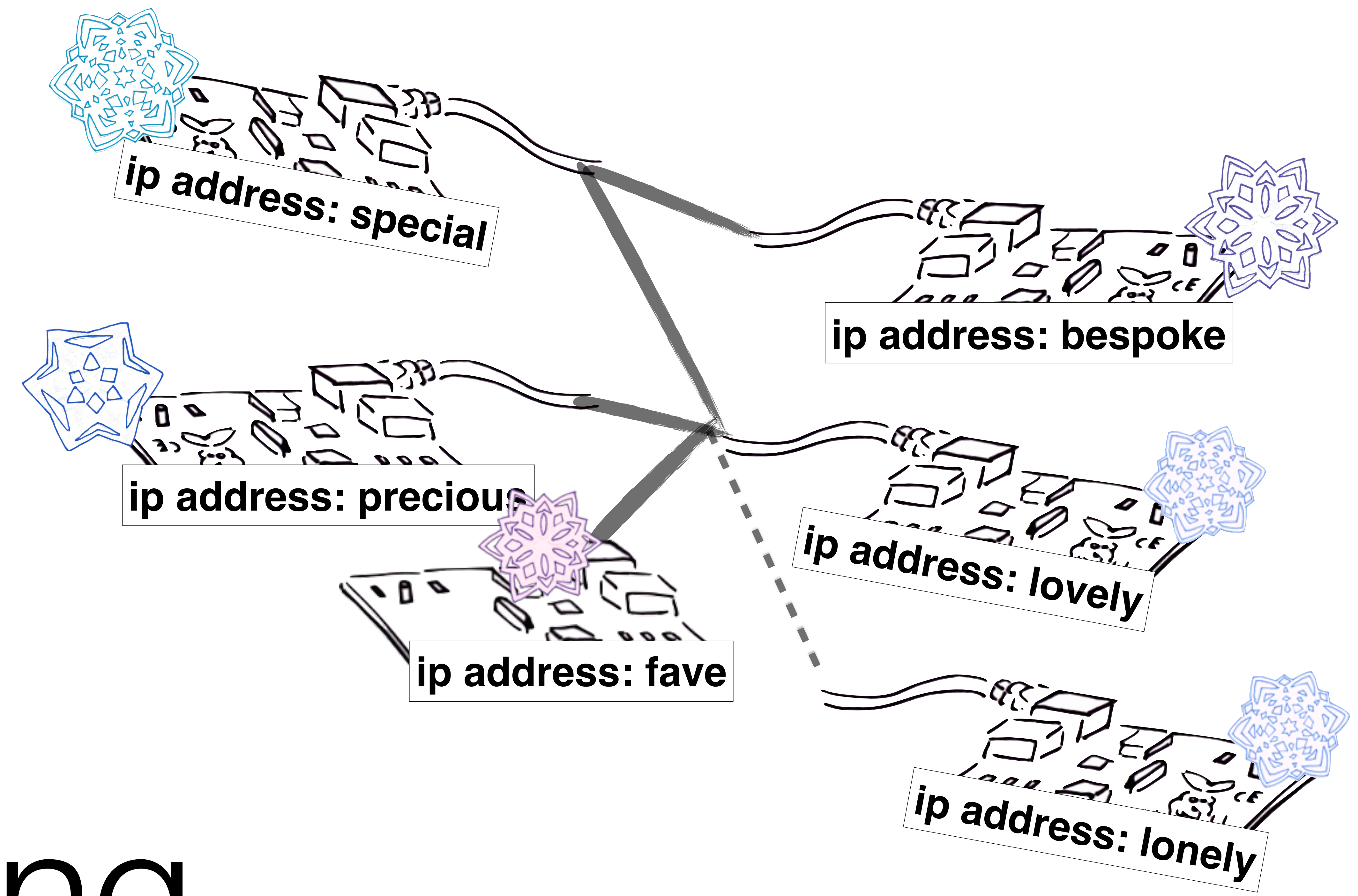
Say **no** to snowflake servers


ip address: special
ip address: bespoke
ip address: precious
ip address: lovely
ip address: fave

Scaling


ip address: special
ip address: bespoke
ip address: precious
ip address: lovely
ip address: fave
ip address: lonely

Scaling


ip address: special
ip address: bespoke
ip address: precious
ip address: lovely
ip address: fave
ip address: lonely

Scaling

- Kubernetes
- Apache Zookeeper + Curator
- Eureka
- etcd
- Consul
- Bluemix Service Discovery

Service discovery

- KuberneteDocker
- Apache Zookeeper + Curator
- Eureka
- etcd
- Consul
- Bluemix Service Discovery

Service discovery

- KuberneteDocker
- Apache Zookeeper + CuraJava
- Eureka
- etcd
- Consul
- Bluemix Service Discovery

Service discovery

- KuberneteDocker
- Apache Zookeeper + CuraJava
- EurekaAWSSoftLayer
- etcd
- Consul
- Bluemix Service Discovery

Service discovery

- KuberneteDocker
- Apache Zookeeper + CuraJava
- EurekaAWSSoftLayer
- etcd CoreOS
- Consul
- Bluemix Service Discovery

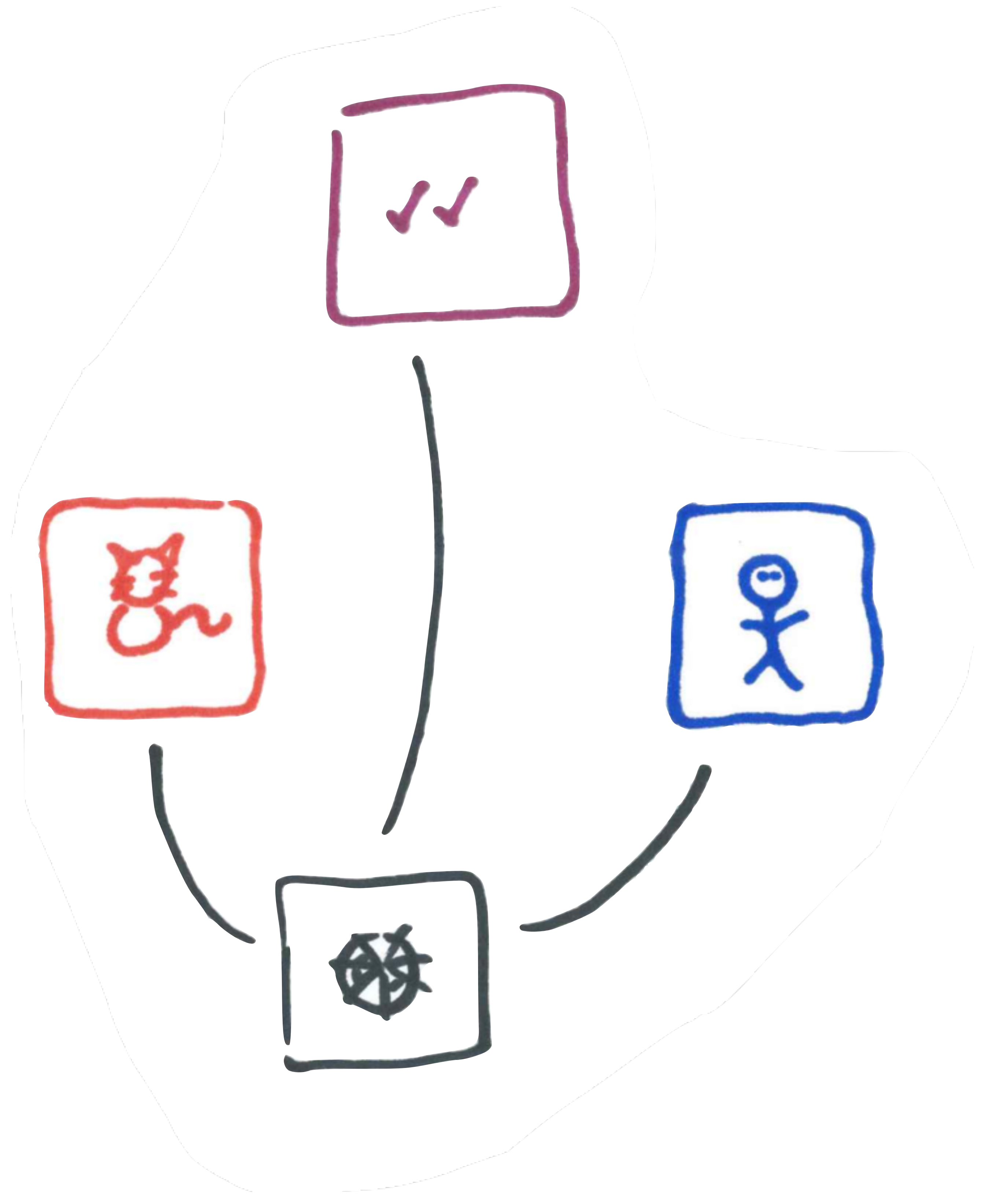
Service discovery

- KuberneteDocker
- Apache Zookeeper + CuraJava
- EurekaAWSSoftLayer
- etcd CoreOS
- ConsuDNS HTTP Java
- Bluemix Service Discovery

Service discovery

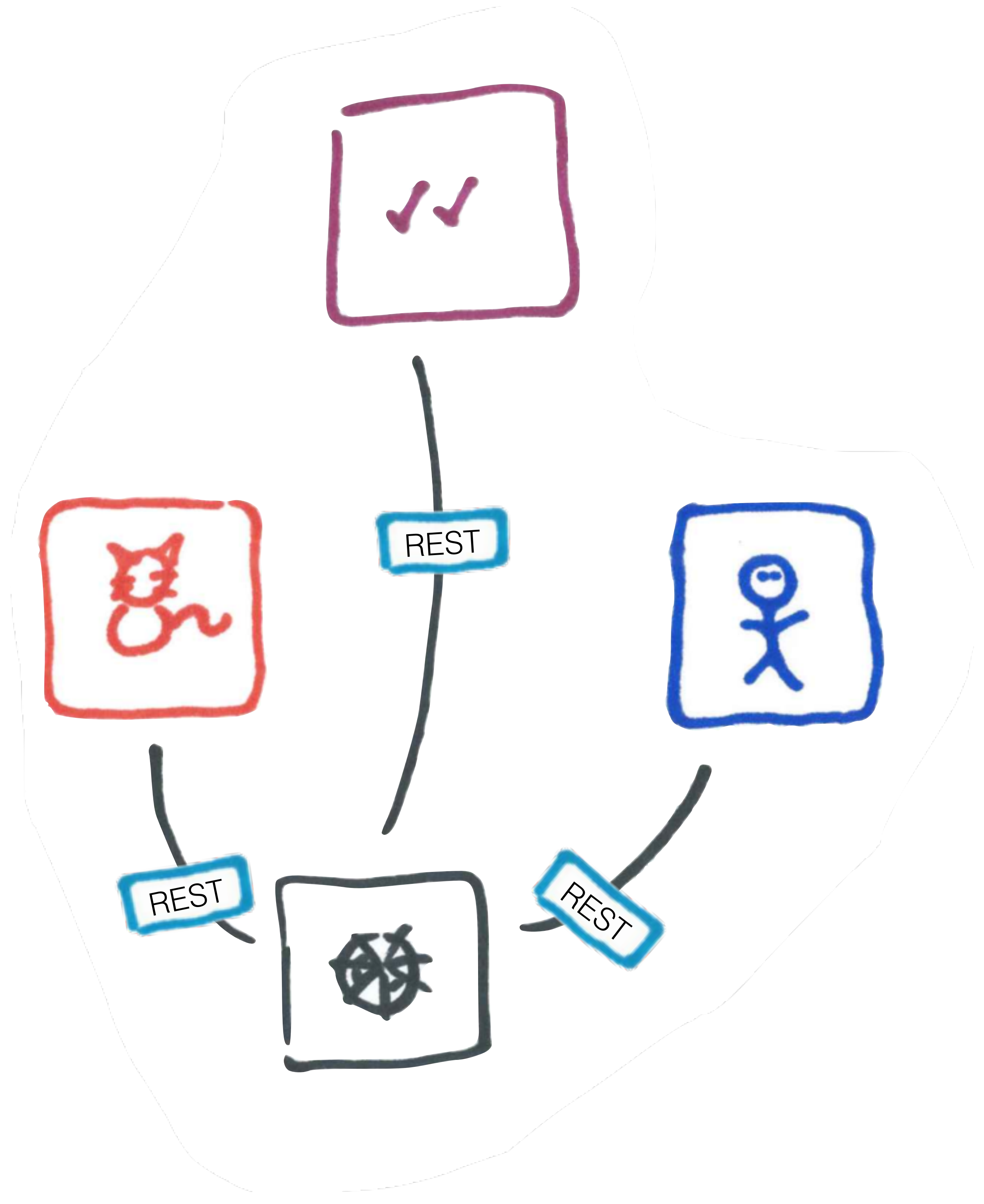
- Kubernete Docker
- Apache Zookeeper + Cura Java
- Eureka AWS SoftLayer
- etcd CoreOS
- Consu DNS HTTP Java
- Bluemix Service Discov Bluemix :)

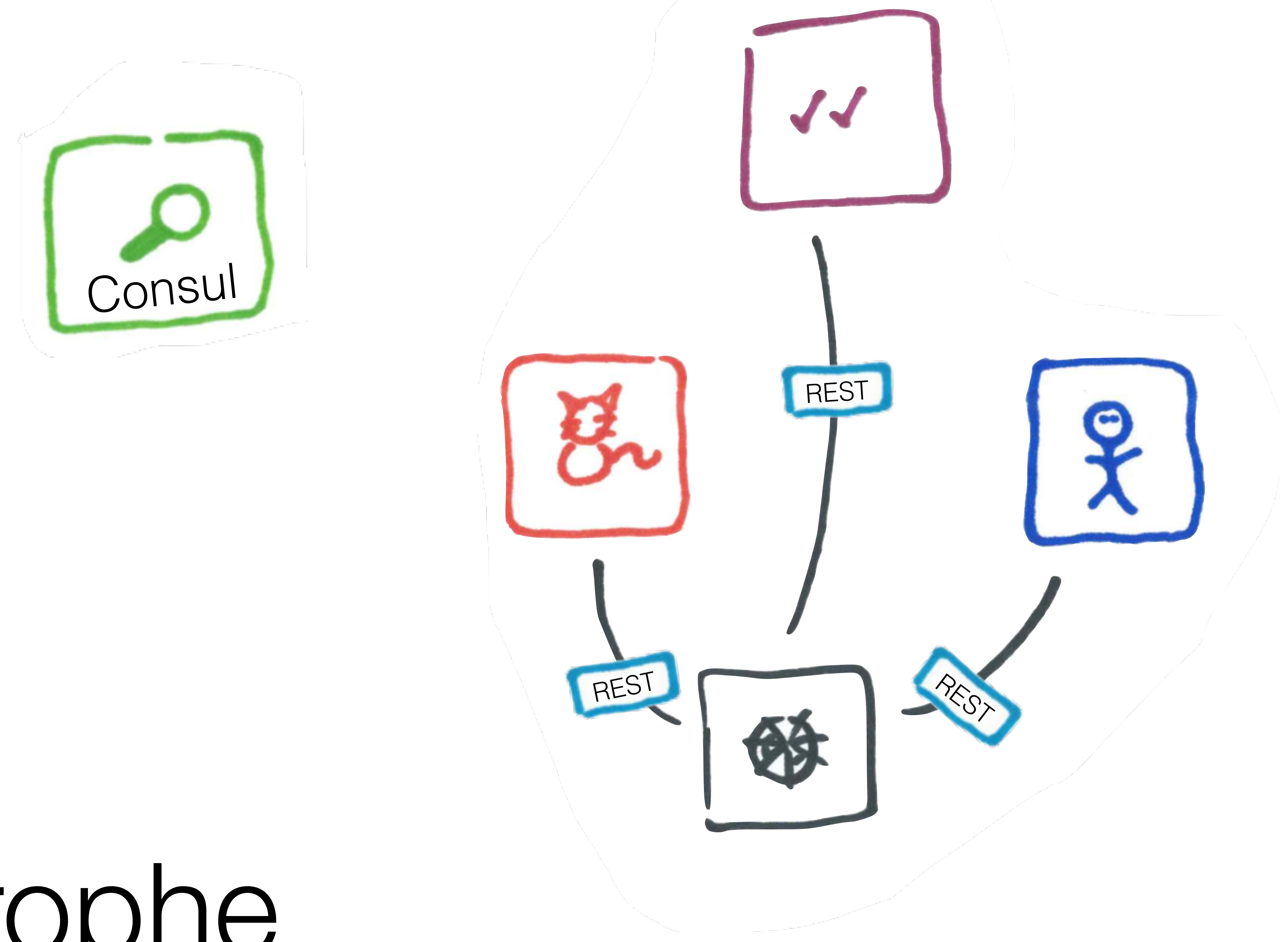
Service discovery



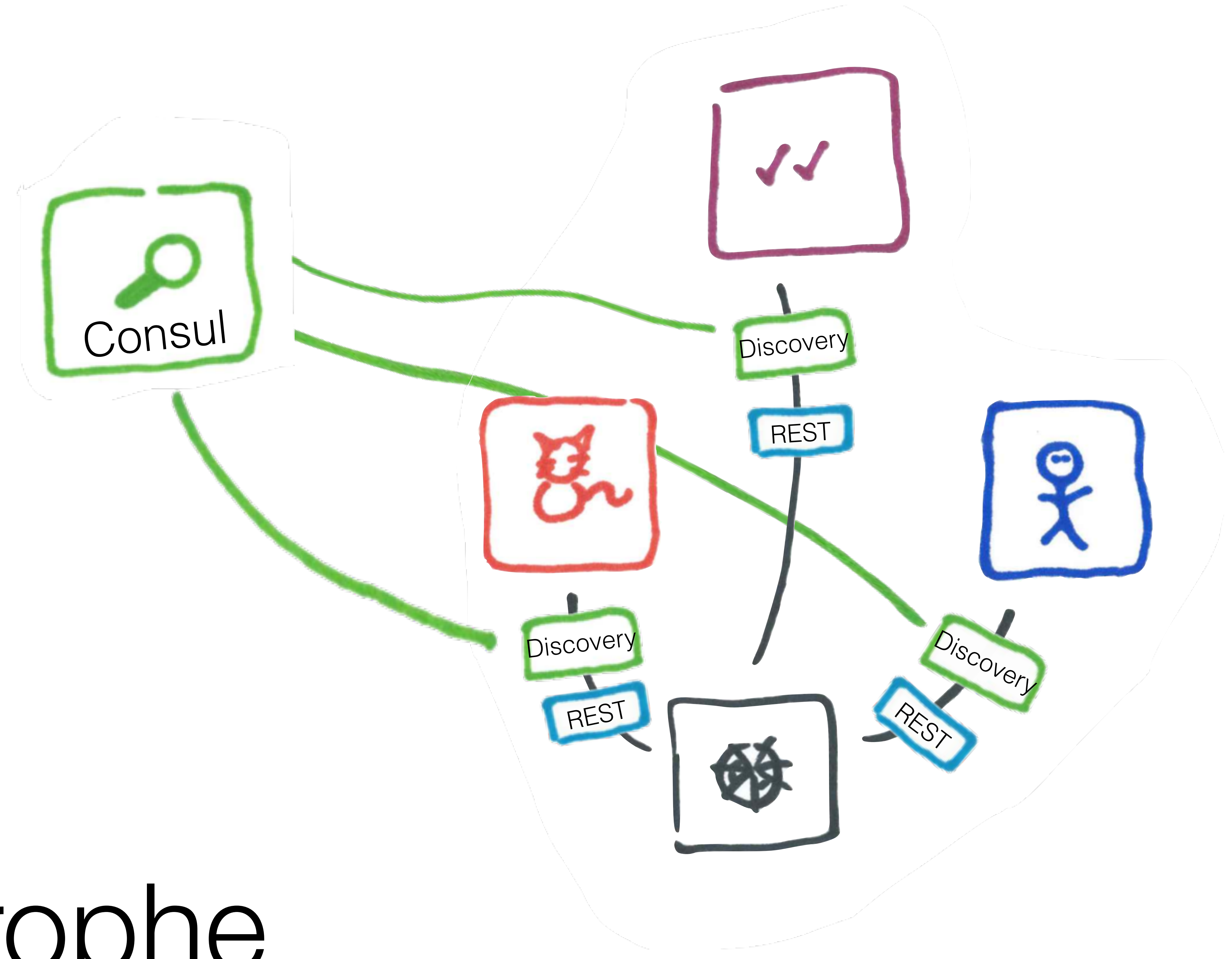
Cat-astrophe

Cat-astrophe





Cat-astrophe



Cat-astrophe

```
<featureManager>
 <feature>jaxrs-1.0</feature>
```

Server
configuration

```
<featureManager>
  <feature>jaxrs-1.0</feature>
```

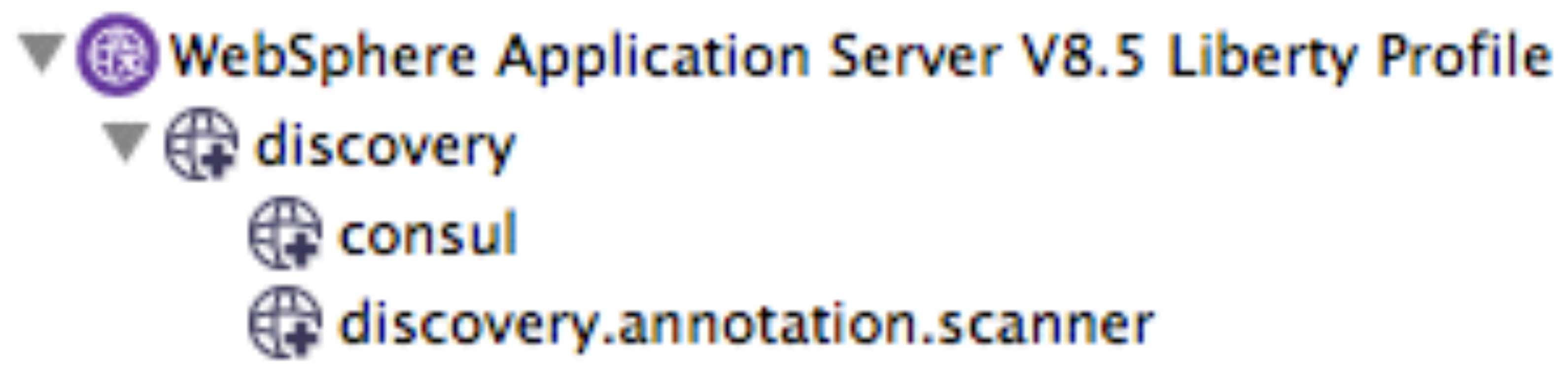
Server
configuration

```
<featureManager>
 <feature>jaxrs-1.0</feature>
 <feature>usr:discovery</feature>
     …

<consul server="catastrophe.consul" />
```

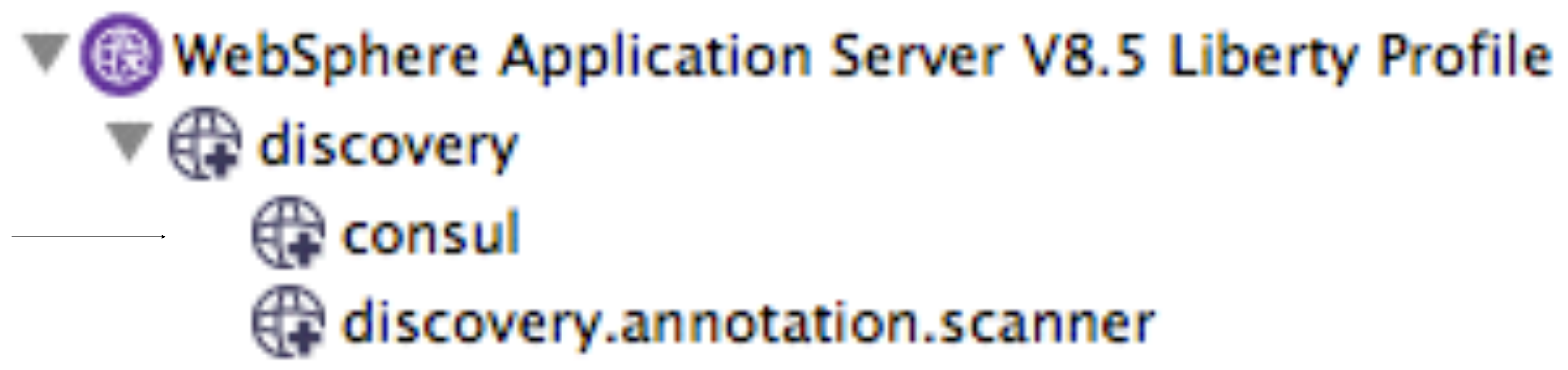
Server configuration

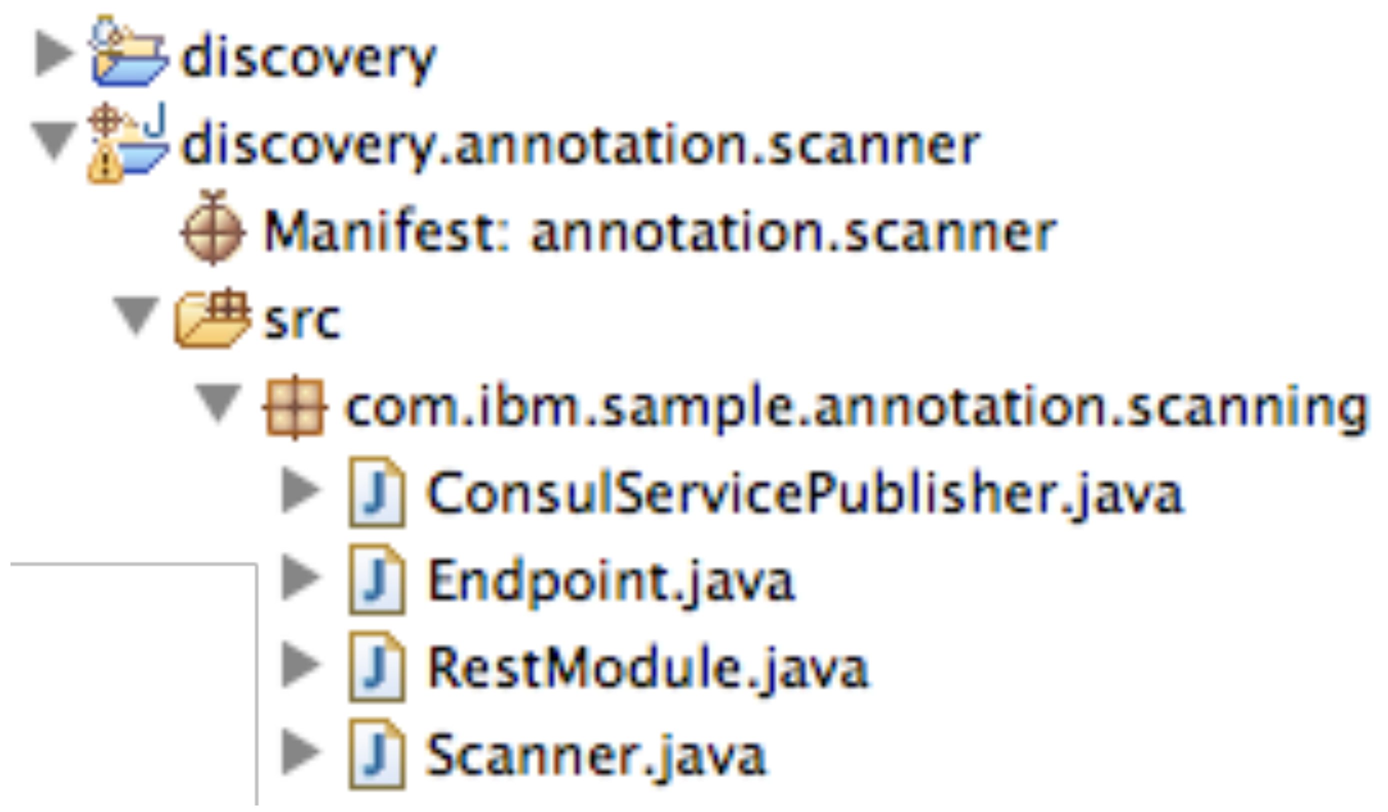
Wouldn't this be nice?

```
<featureManager>
  <feature>jaxrs-1.0</feature>
  <feature>usr:discovery</feature>
      ...

<consul server="catastrophe.consul" />
```


WebSphere Application Server V8.5 Liberty Profile
discovery
consul
discovery.annotation.scanner

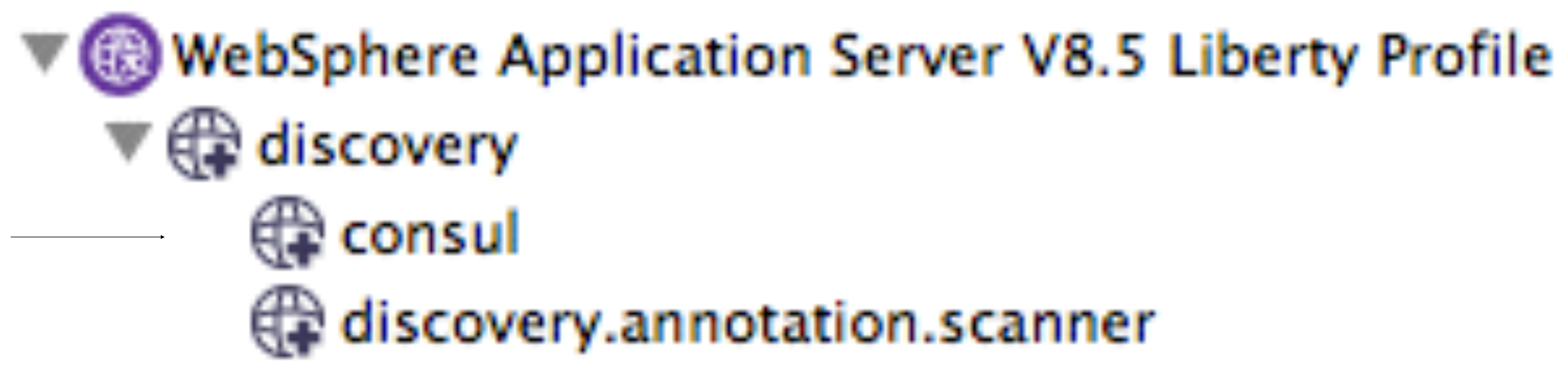
Liberty extension ("user feature")


WebSphere Application Server V8.5 Liberty Profile
discovery
consul
discovery.annotation.scanner



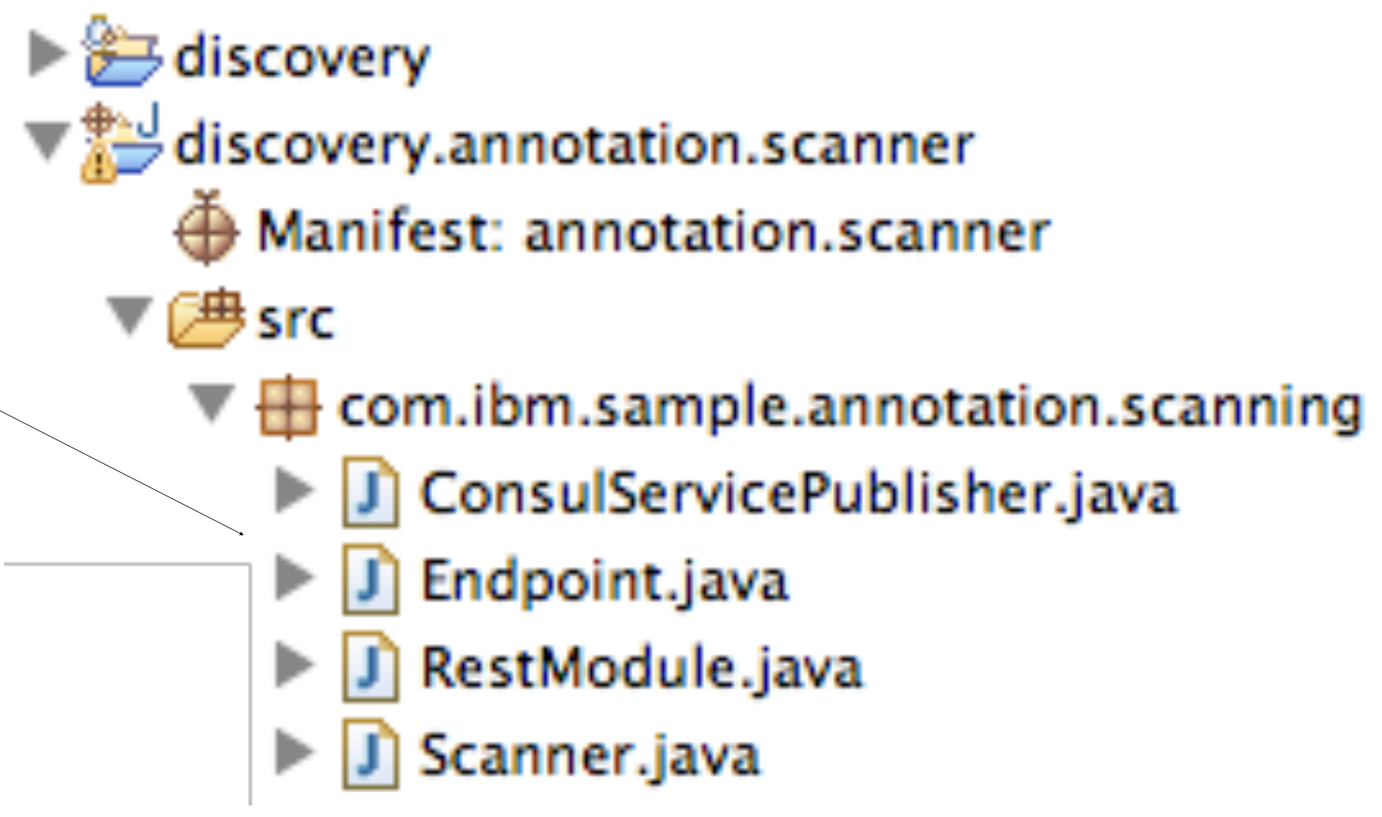
- discovery
- discovery.annotation.scanner
  - Manifest: annotation.scanner
  - src
    - com.ibm.sample.annotation.scanning
      - ConsulServicePublisher.java
      - Endpoint.java
      - RestModule.java
      - Scanner.java

Liberty extension (“user feature”)

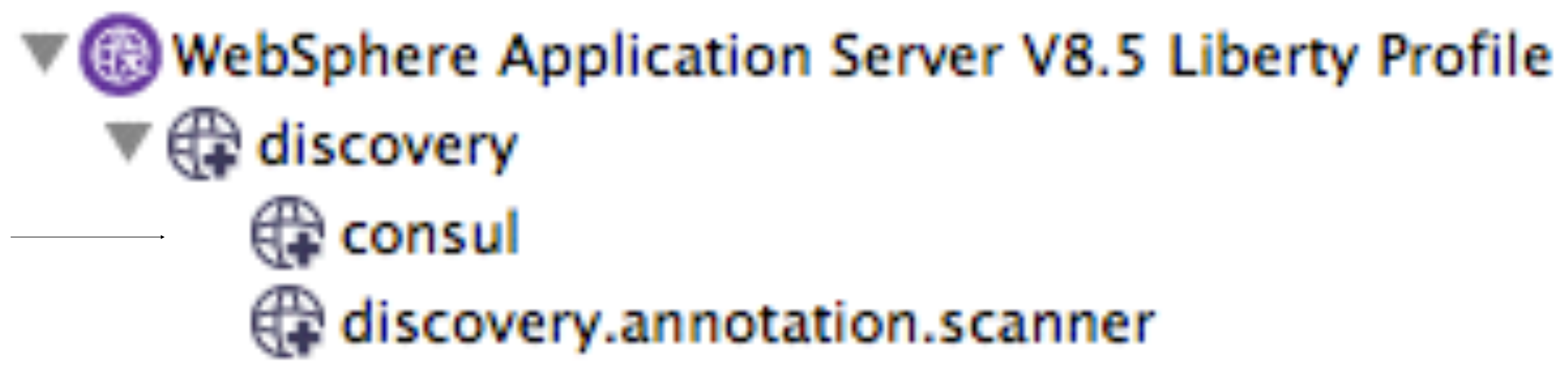


- WebSphere Application Server V8.5 Liberty Profile
  - discovery
    - consul
    - discovery.annotation.scanner

Auto-publishes REST endpoints

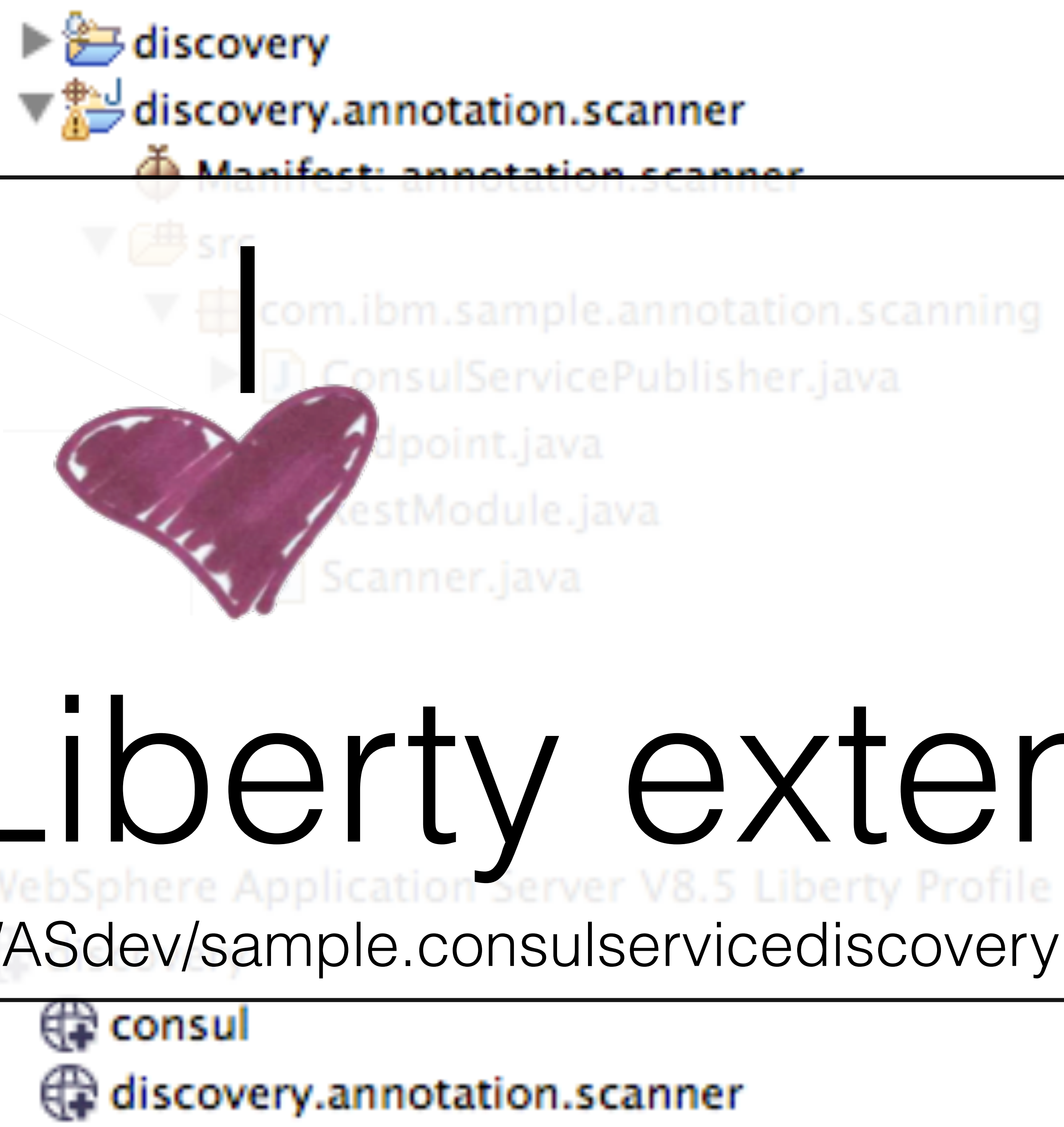

discovery
discovery.annotation.scanner
Manifest: annotation.scanner
src
com.ibm.sample.annotation.scanning
ConsulServicePublisher.java
Endpoint.java
RestModule.java
Scanner.java

Liberty extension ("user feature")


WebSphere Application Server V8.5 Liberty Profile
discovery
consul
discovery.annotation.scanner

Auto-publishes REST endpoints

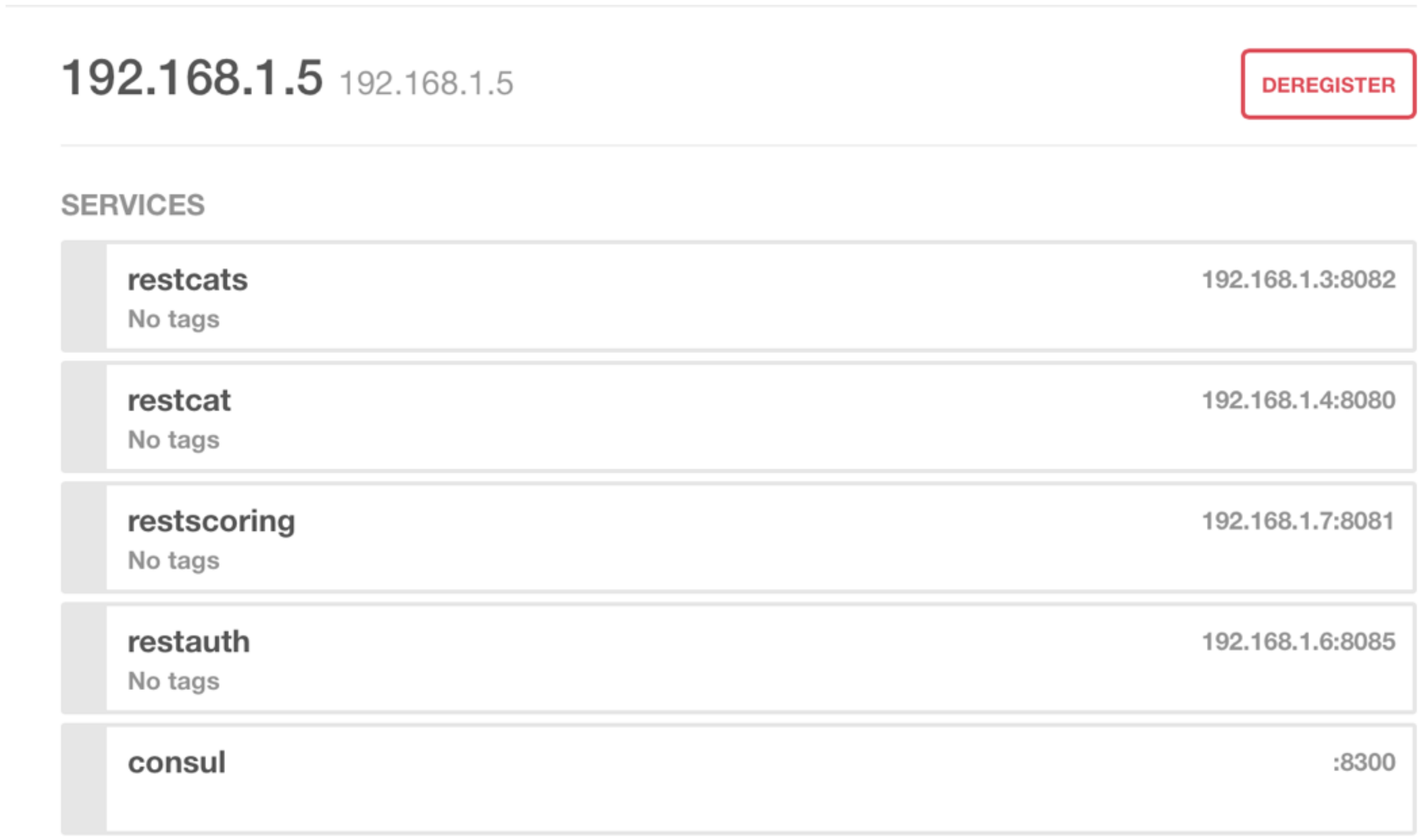
Liberty extension ("user feature")


discovery
discovery.annotation.scanner
Manifest: annotation.scanner
src
com.ibm.sample.annotation.scanning
ConsulServicePublisher.java
dpoint.java
estModule.java
Scanner.java
WebSphere Application Server V8.5 Liberty Profile
consul
discovery.annotation.scanner

I

WebSphere Liberty extensibility

https://github.com/WASdev/sample.consulservicediscovery



192.168.1.5 192.168.1.5

SERVICES

restcats 192.168.1.3:8082
No tags

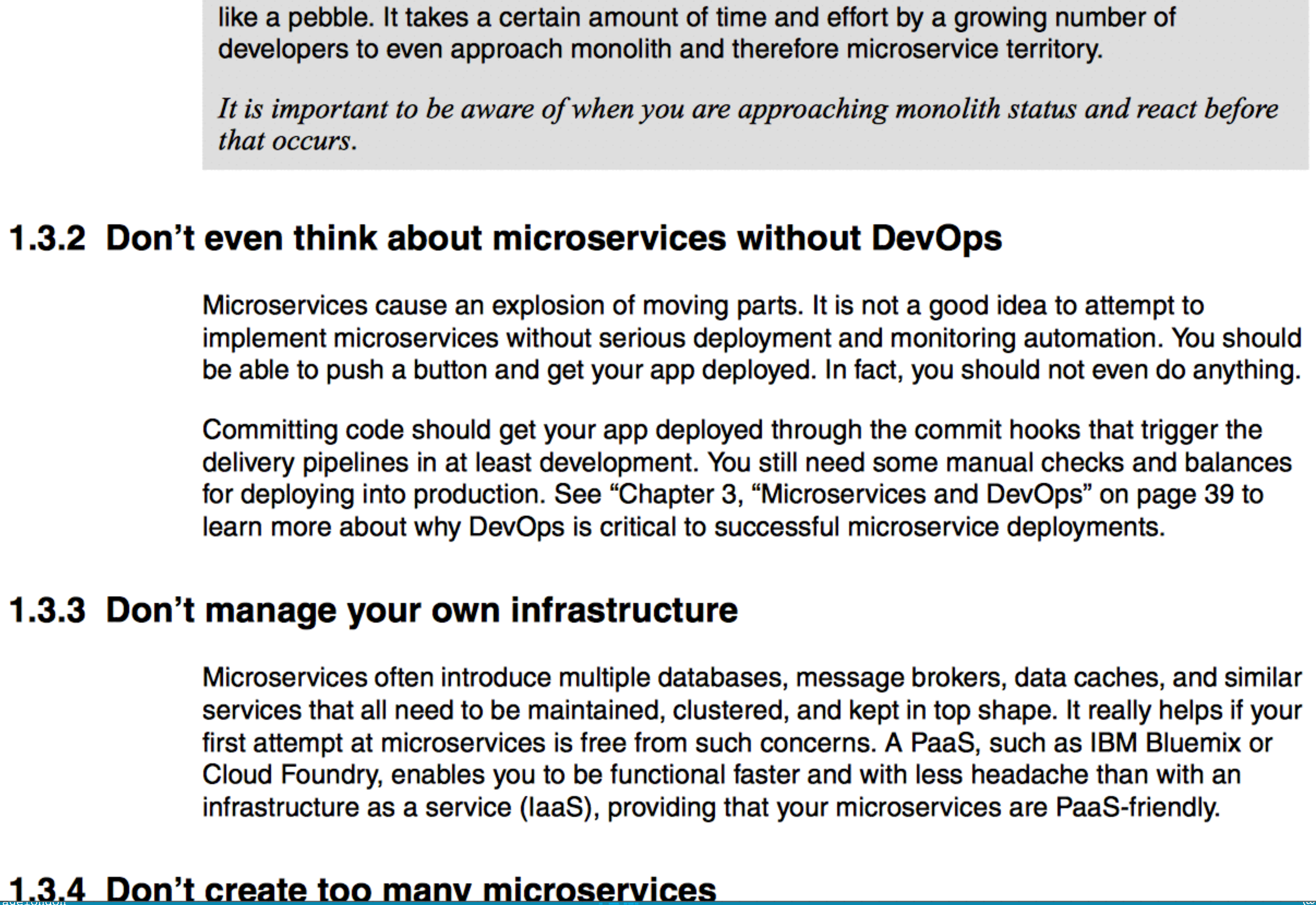
restcat 192.168.1.4:8080
No tags

restscoring 192.168.1.7:8081
No tags

restauth 192.168.1.6:8085
No tags

consul :8300

Consul view of the Catastrophe services



like a pebble. It takes a certain amount of time and effort by a growing number of developers to even approach monolith and therefore microservice territory.

It is important to be aware of when you are approaching monolith status and react before that occurs.

1.3.2 Don't even think about microservices without DevOps

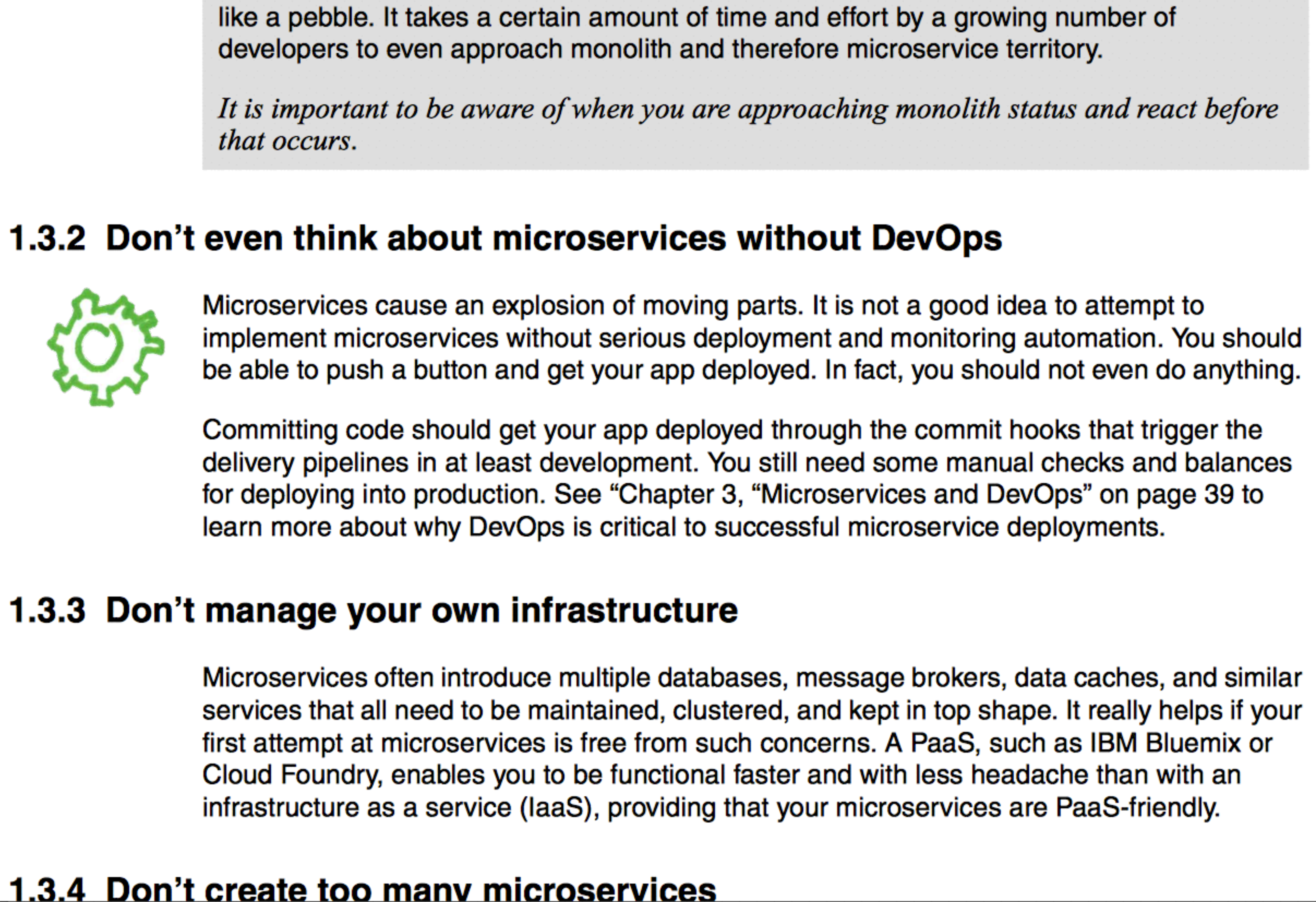
Microservices cause an explosion of moving parts. It is not a good idea to attempt to implement microservices without serious deployment and monitoring automation. You should be able to push a button and get your app deployed. In fact, you should not even do anything.

Committing code should get your app deployed through the commit hooks that trigger the delivery pipelines in at least development. You still need some manual checks and balances for deploying into production. See "Chapter 3, "Microservices and DevOps" on page 39 to learn more about why DevOps is critical to successful microservice deployments.

1.3.3 Don't manage your own infrastructure

Microservices often introduce multiple databases, message brokers, data caches, and similar services that all need to be maintained, clustered, and kept in top shape. It really helps if your first attempt at microservices is free from such concerns. A PaaS, such as IBM Bluemix or Cloud Foundry, enables you to be functional faster and with less headache than with an infrastructure as a service (IaaS), providing that your microservices are PaaS-friendly.

1.3.4 Don't create too many microservices


like a pebble. It takes a certain amount of time and effort by a growing number of developers to even approach monolith and therefore microservice territory.

It is important to be aware of when you are approaching monolith status and react before that occurs.

1.3.2 Don't even think about microservices without DevOps

Microservices cause an explosion of moving parts. It is not a good idea to attempt to implement microservices without serious deployment and monitoring automation. You should be able to push a button and get your app deployed. In fact, you should not even do anything.

Committing code should get your app deployed through the commit hooks that trigger the delivery pipelines in at least development. You still need some manual checks and balances for deploying into production. See "Chapter 3, "Microservices and DevOps" on page 39 to learn more about why DevOps is critical to successful microservice deployments.

1.3.3 Don't manage your own infrastructure

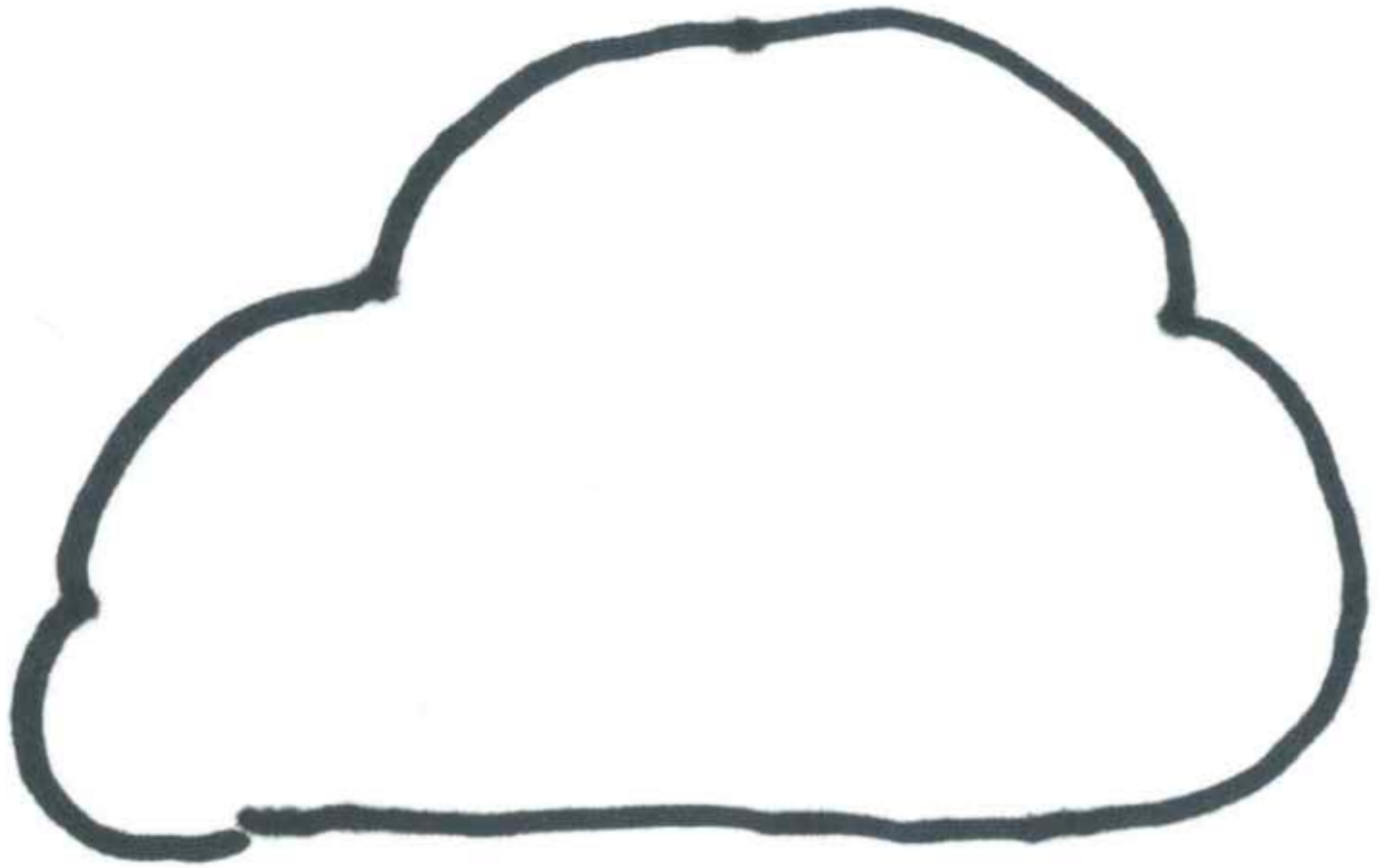
Microservices often introduce multiple databases, message brokers, data caches, and similar services that all need to be maintained, clustered, and kept in top shape. It really helps if your first attempt at microservices is free from such concerns. A PaaS, such as IBM Bluemix or Cloud Foundry, enables you to be functional faster and with less headache than with an infrastructure as a service (IaaS), providing that your microservices are PaaS-friendly.

1.3.4 Don't create too many microservices

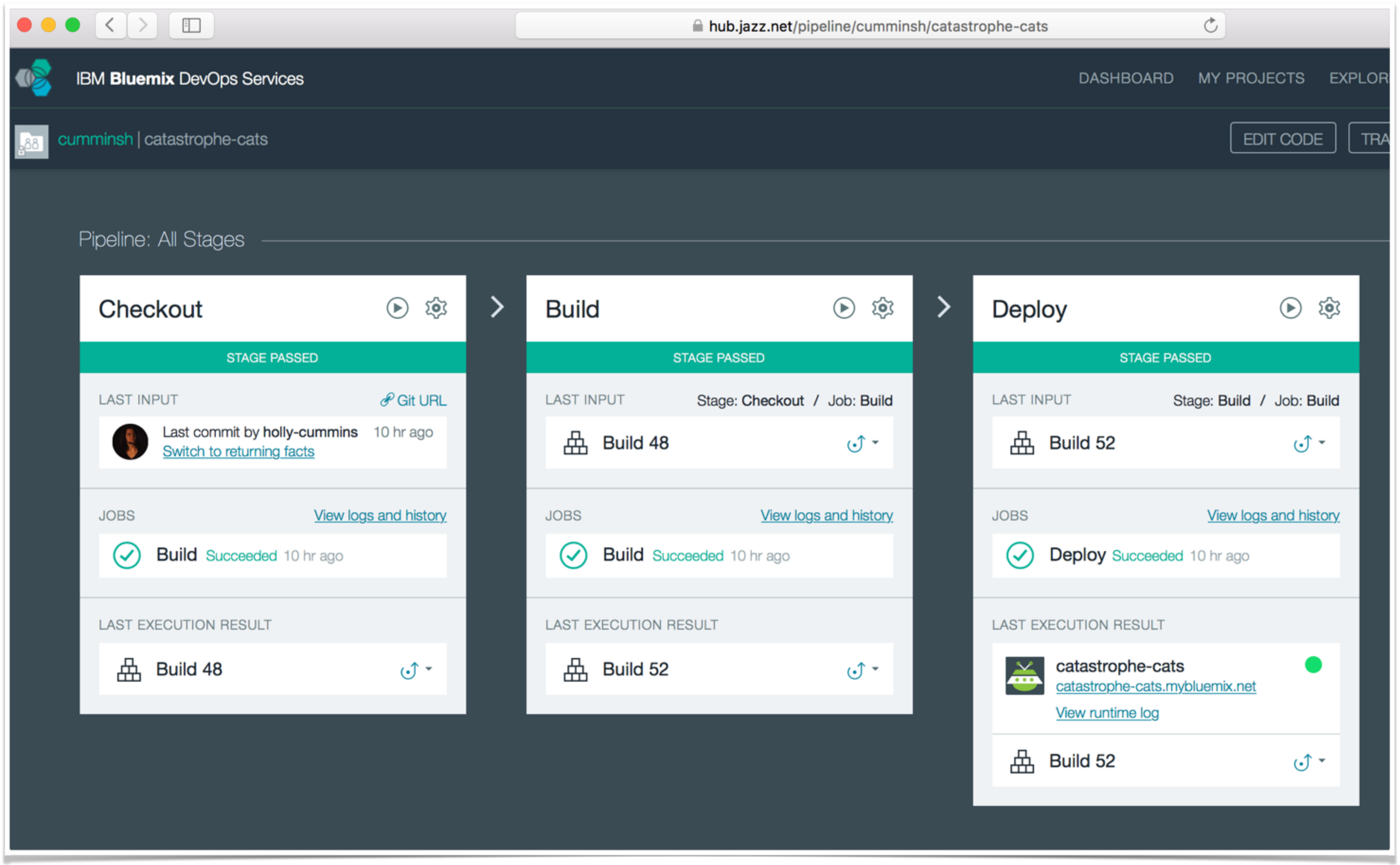
You need DevOps

An SD card is not devops :)

You need 100% automation



IBM Bluemix™


hub.jazz.net/pipeline/cumminsh/catastrophe-cats

IBM **Bluemix** DevOps Services

DASHBOARD MY PROJECTS EXPLOR

cumminsh | catastrophe-cats

EDIT CODE TRA

Pipeline: All Stages

Checkout

STAGE PASSED

LAST INPUT — Git URL

Last commit by holly-cummins 10 hr ago
Switch to returning facts

JOBS — View logs and history

Build Succeeded 10 hr ago

LAST EXECUTION RESULT

Build 48

Build

STAGE PASSED

LAST INPUT — Stage: Checkout / Job: Build

Build 48

JOBS — View logs and history

Build Succeeded 10 hr ago

LAST EXECUTION RESULT

Build 52

Deploy

STAGE PASSED

LAST INPUT — Stage: Build / Job: Build

Build 52

JOBS — View logs and history

Deploy Succeeded 10 hr ago

LAST EXECUTION RESULT

catastrophe-cats
catastrophe-cats.mybluemix.net
View runtime log

Build 52

http://catastrophe-web.mybluemix.net/

Who can draw the best cat?

http://catastrophe-web.mybluemix.net/

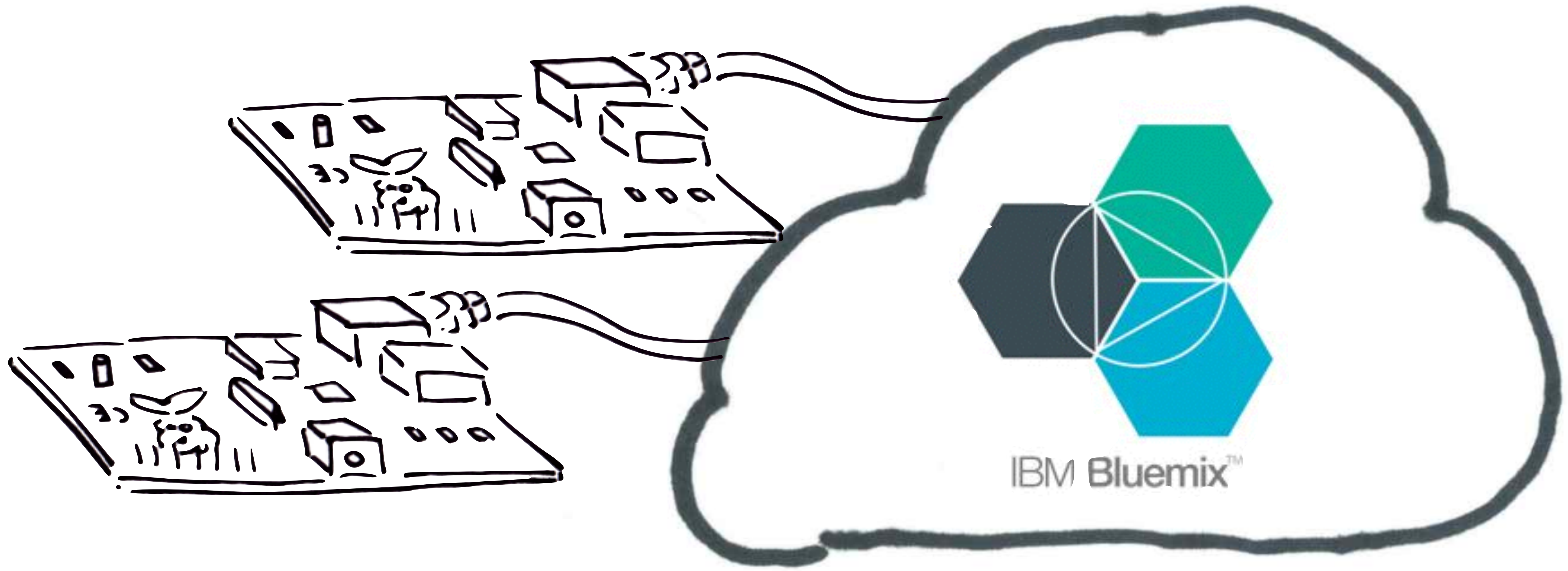
(I have THINK hats for the highest scores!)

What if I want to run on both pis and the cloud?

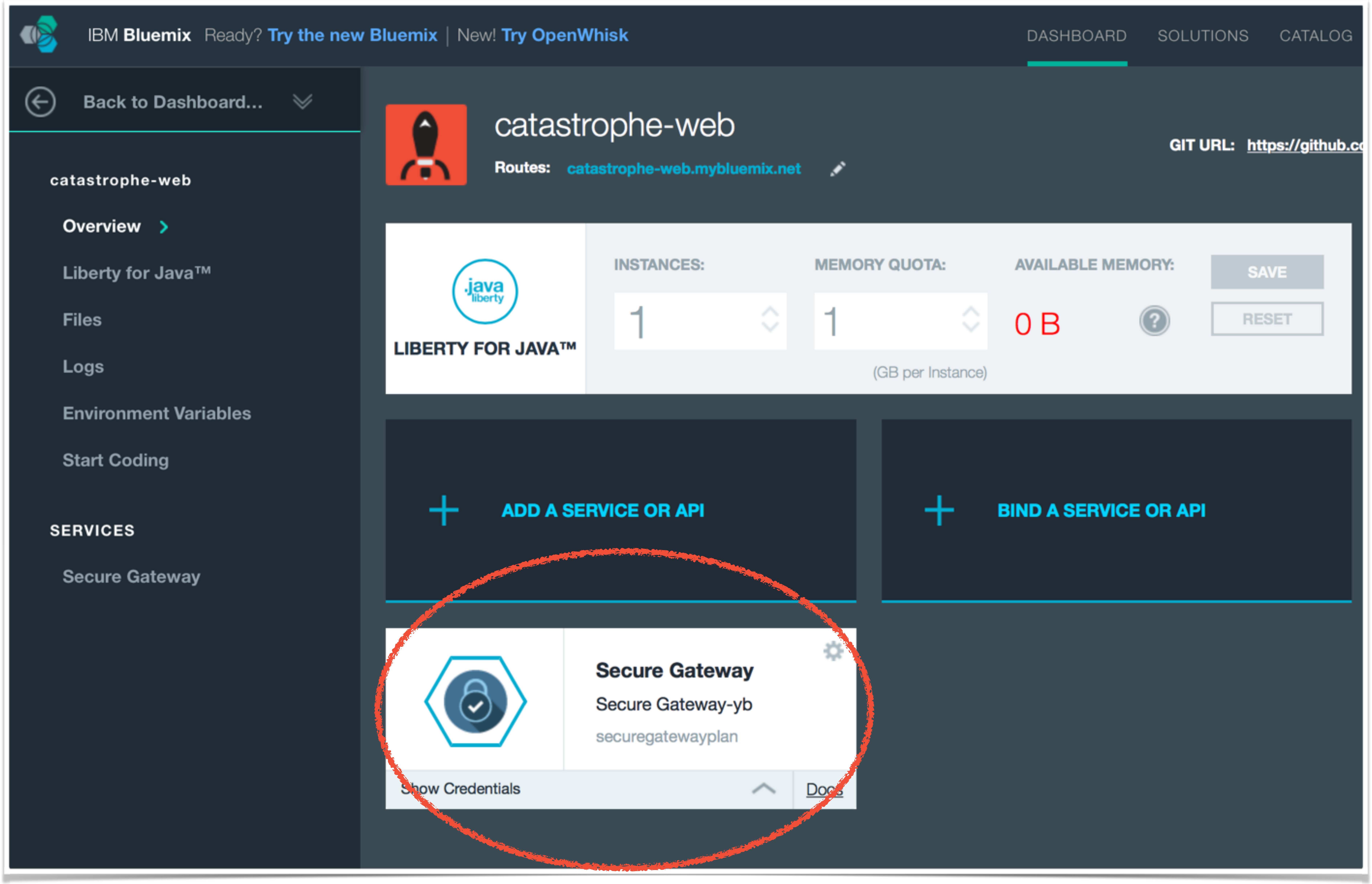
What if I want to run on both pis and the cloud?

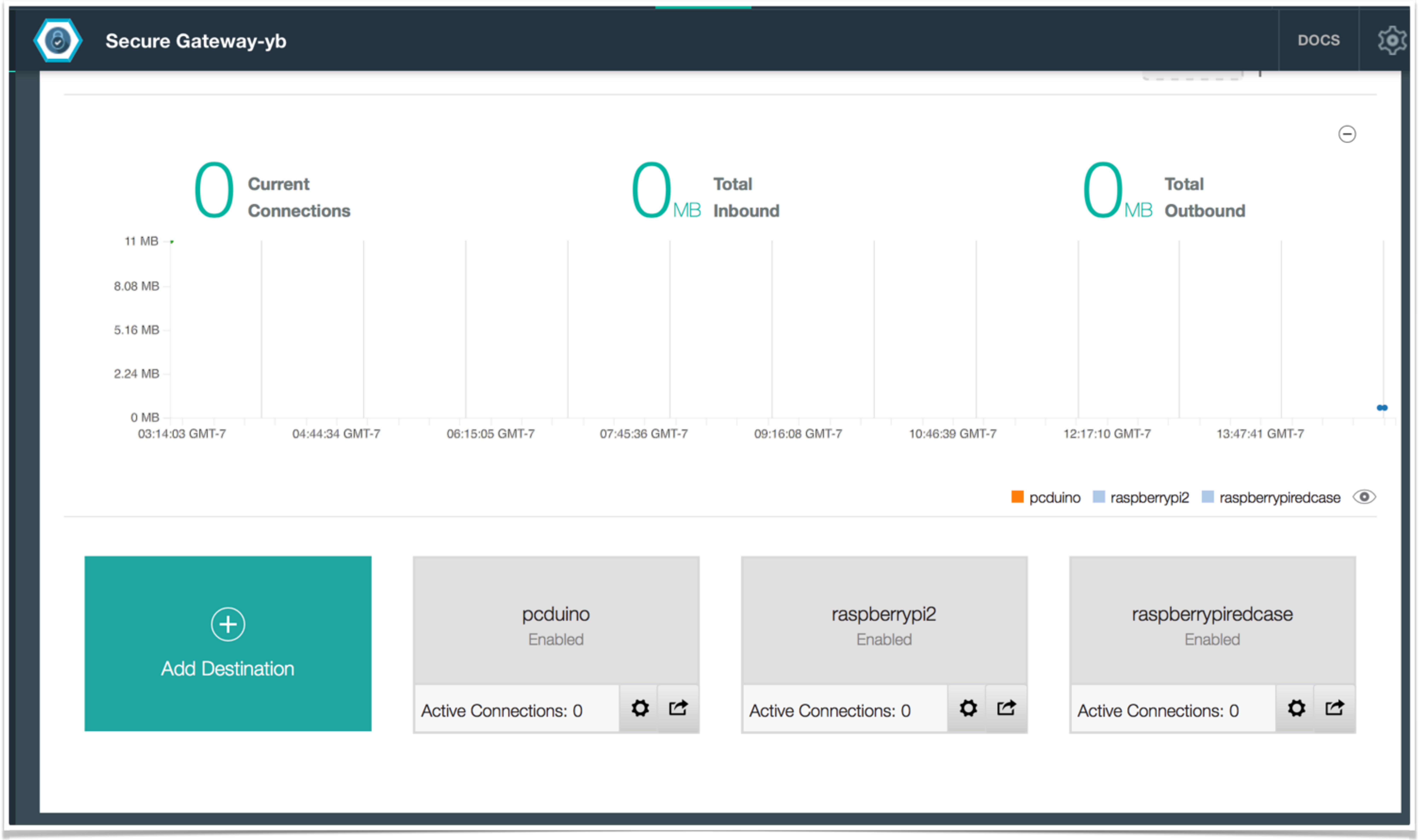
You need **Hybrid** Cloud!

What if I want to run on both pis and the cloud?



You need **Hybrid** Cloud!


IBM Bluemix Ready? Try the new Bluemix | New! Try OpenWhisk
DASHBOARD
SOLUTIONS
CATALOG
Back to Dashboard...
catastrophe-web
Overview
Liberty for Java™
Files
Logs
Environment Variables
Start Coding
SERVICES
Secure Gateway
catastrophe-web
Routes: catastrophe-web.mybluemix.net
GIT URL: https://github.co
LIBERTY FOR JAVA™
INSTANCES: 1
MEMORY QUOTA: 1
(GB per Instance)
AVAILABLE MEMORY: 0 B
SAVE
RESET
ADD A SERVICE OR API
BIND A SERVICE OR API
Secure Gateway
Secure Gateway-yb
securegatewayplan
Show Credentials
Docs



0 Current Connections

0 MB Total Inbound

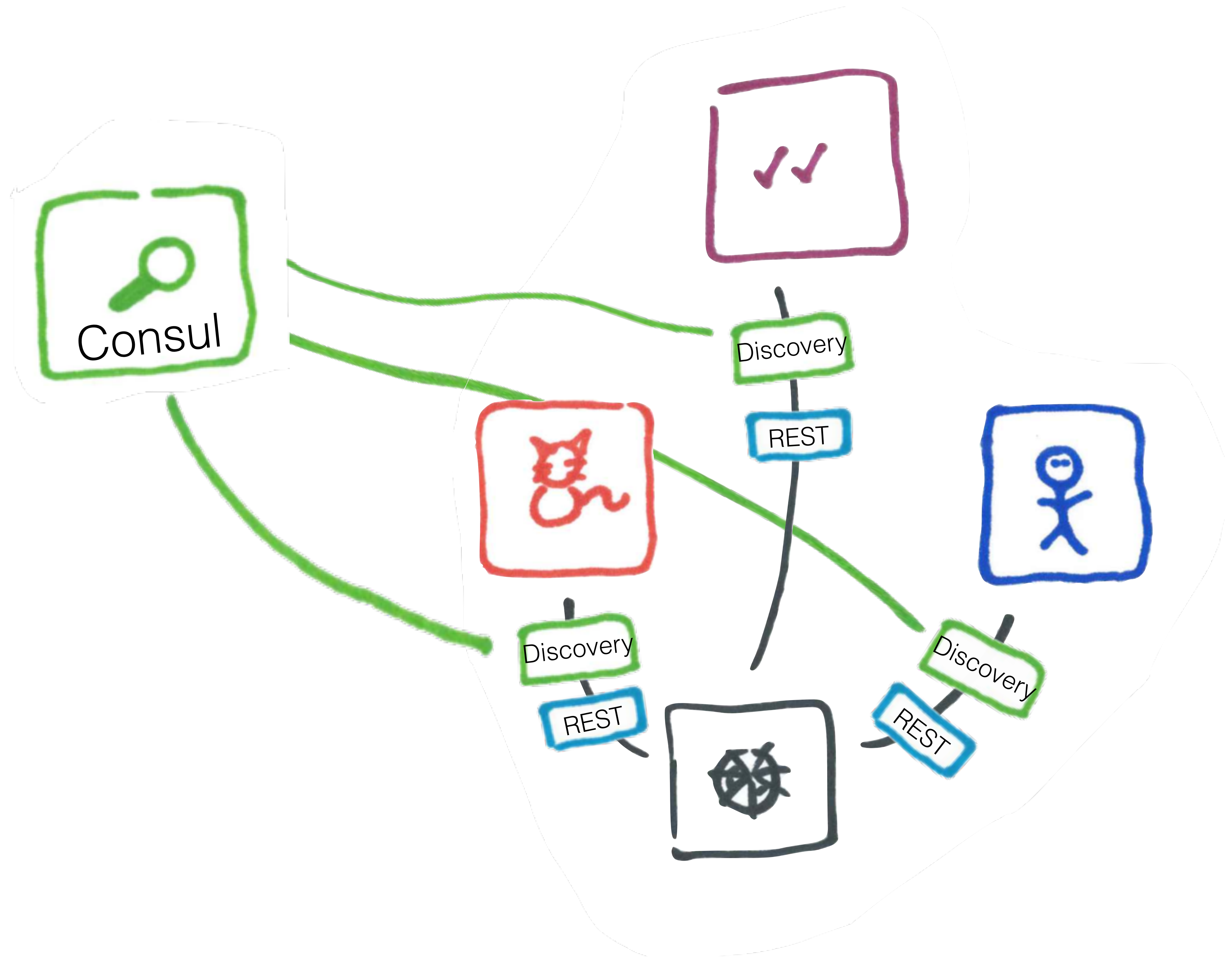
0 MB Total Outbound

pcduino
Enabled
Active Connections: 0

raspberrypi2
Enabled
Active Connections: 0

raspberrypiredcase
Enabled
Active Connections: 0

Are we done?

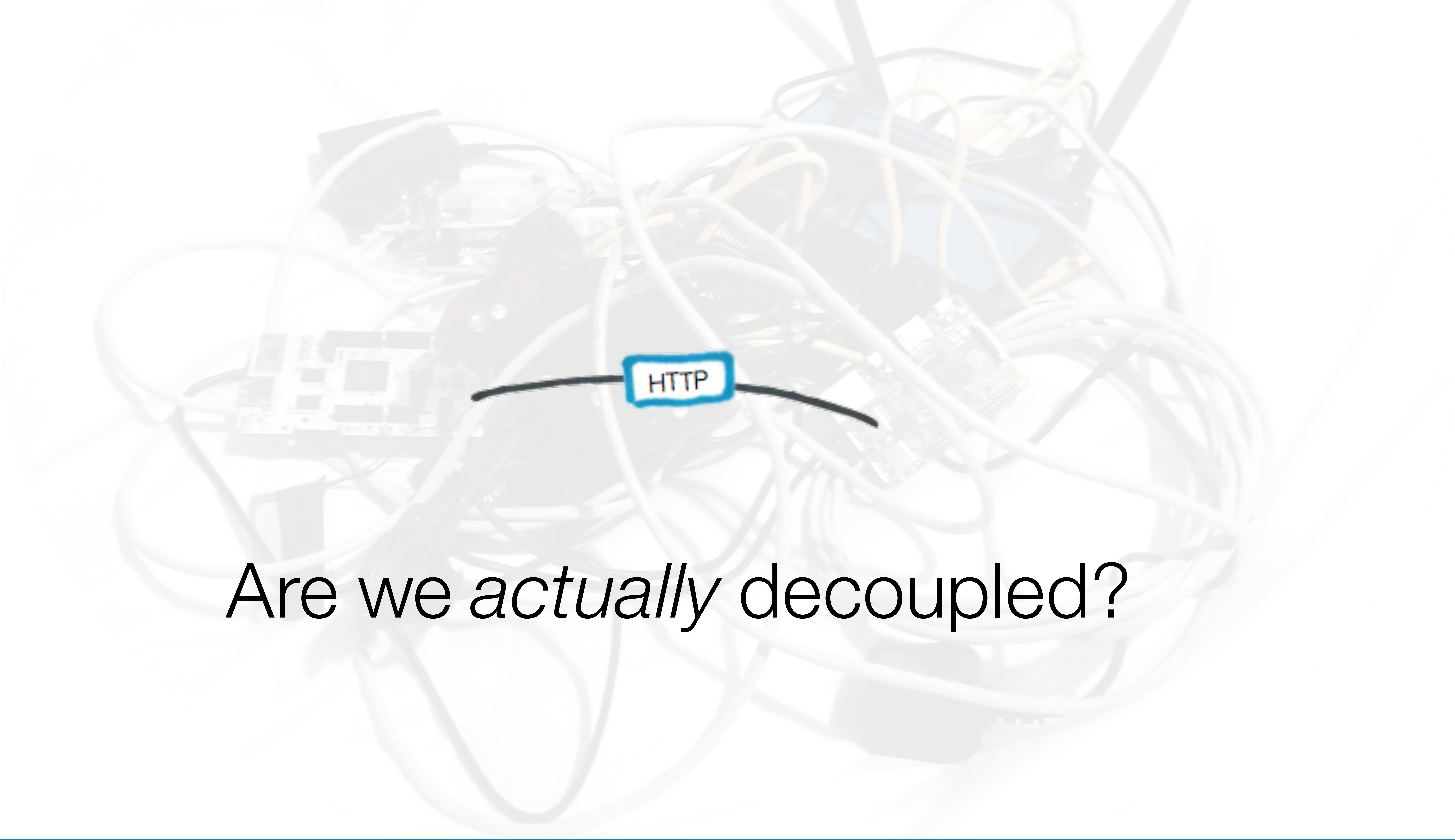


Are we done?

Have we tested it?

How de we handle failures?

Are we *actually* decoupled?



Are we *actually* decoupled?

So **remember…**

- Decoupling is more than just HTTP communication
- Some of your microservices **will** fail. Be resilient.
- I ♥ WebSphere Liberty
- JEE is great for microservices (especially with microprofile)
- Hybrid cloud makes a lot of cool stuff possible

Thank You!

```
http://ibm.biz/bluemixgaragelondon
http://github.com/holly-cummins/catastrophe-microservices
```

Holly Cummins | **@holly_cummins**