

# **PRINCIPLES OF MICROSERVICES**

# THE TWELVE FACTORS

## I. Codebase

One codebase tracked in revision control, many deploys

## II. Dependencies

Explicitly declare and isolate dependencies

## III. Config

Store config in the environment

## IV. Backing Services

Treat backing services as attached resources

## V. Build, release, run

Strictly separate build and run stages

## VI. Processes

Execute the app as one or more stateless processes

## VII. Port binding

Export services via port binding

## VIII. Concurrency

Scale out via the process model

## IX. Disposability

Maximize robustness with fast startup and graceful shutdown

## X. Dev/prod parity

Keep development, staging, and production as similar as possible

## XI. Logs

Treat logs as event streams

## Strategic Goals

### Enable scalable business

More customers/transactions  
Self-service for customers

### Support entry into new markets

Flexible operational processes  
New products and operational processes

### Support innovation in existing markets

Flexible operational processes  
New products and operational processes

## Architectural Principles

### Reduce inertia

Make choices that favour rapid feedback and change, with reduced dependencies across teams.

### Eliminate accidental complexity

Aggressively retire and replace unnecessarily complex processes, systems, and integrations so that we can focus on the essential complexity.

### Consistent interfaces and data flows

Eliminate duplication of data and create clear systems of record, with consistent integration interfaces.

### No silver bullets

Off the shelf solutions deliver early value but create inertia and accidental complexity.

## Design and Delivery Practices

### Standard REST/HTTP

### Encapsulate legacy

### Eliminate integration databases

### Consolidate and cleanse data

### Published integration model

### Small independent Services

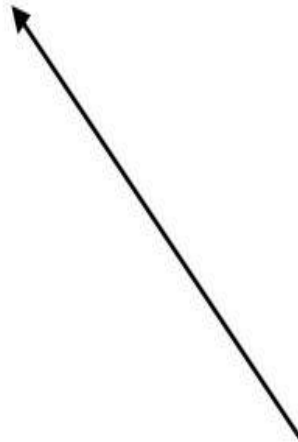
### Continuous deployment

### Minimal customisation of COTS/SAAS

Small ***Autonomous*** services  
that ***work together***

# **Principles Of Microservices**

Modelled Around  
Business Domain

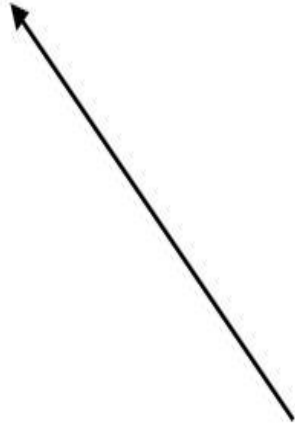


**Principles Of  
Microservices**

Modelled Around  
Business Domain

Culture Of  
Automation

**Principles Of  
Microservices**



Modelled Around  
Business Domain

Culture Of  
Automation

Hide Implementation  
Details

**Principles Of  
Microservices**



```
graph TD; A[Principles Of Microservices] --> B[Modelled Around Business Domain]; A --> C[Culture Of Automation]; A --> D[Hide Implementation Details];
```



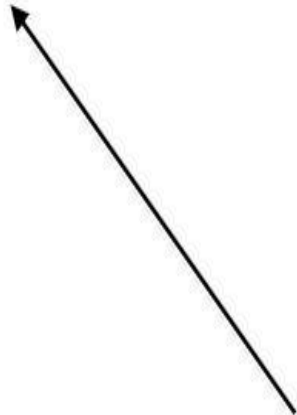
Modelled Around  
Business Domain

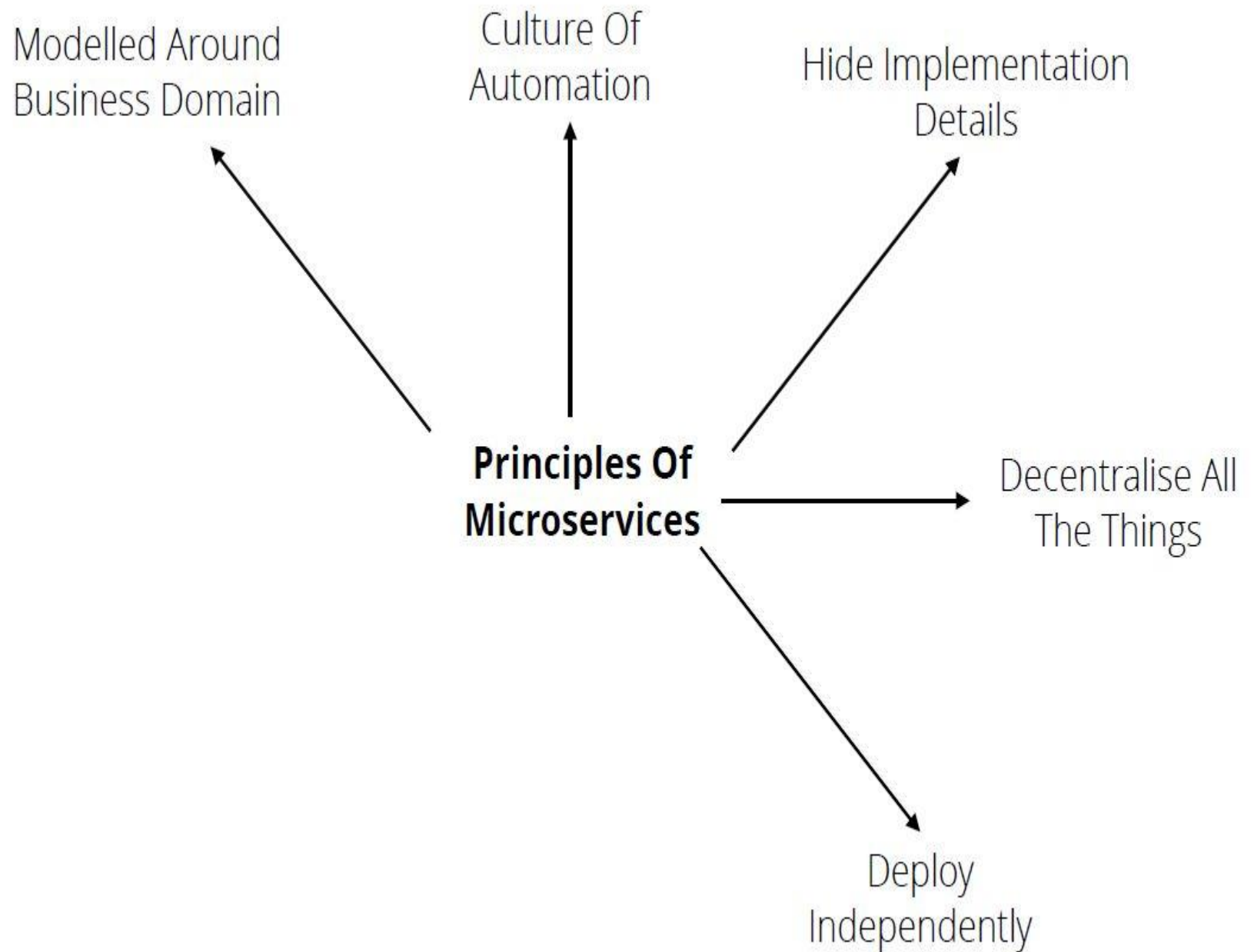
Culture Of  
Automation

Hide Implementation  
Details

**Principles Of  
Microservices**

Decentralise All  
The Things





Modelled Around  
Business Domain

Culture Of  
Automation

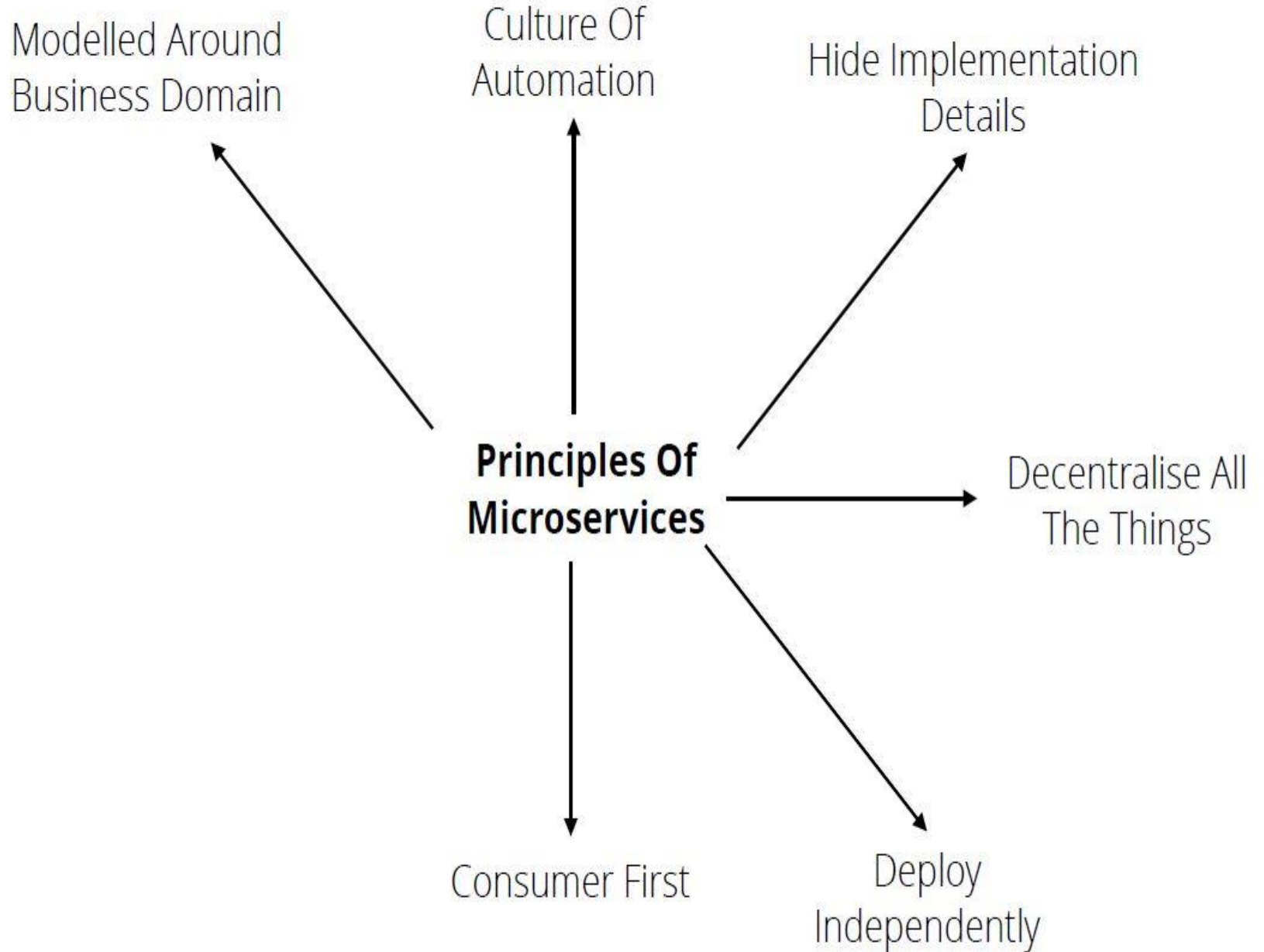
Hide Implementation  
Details

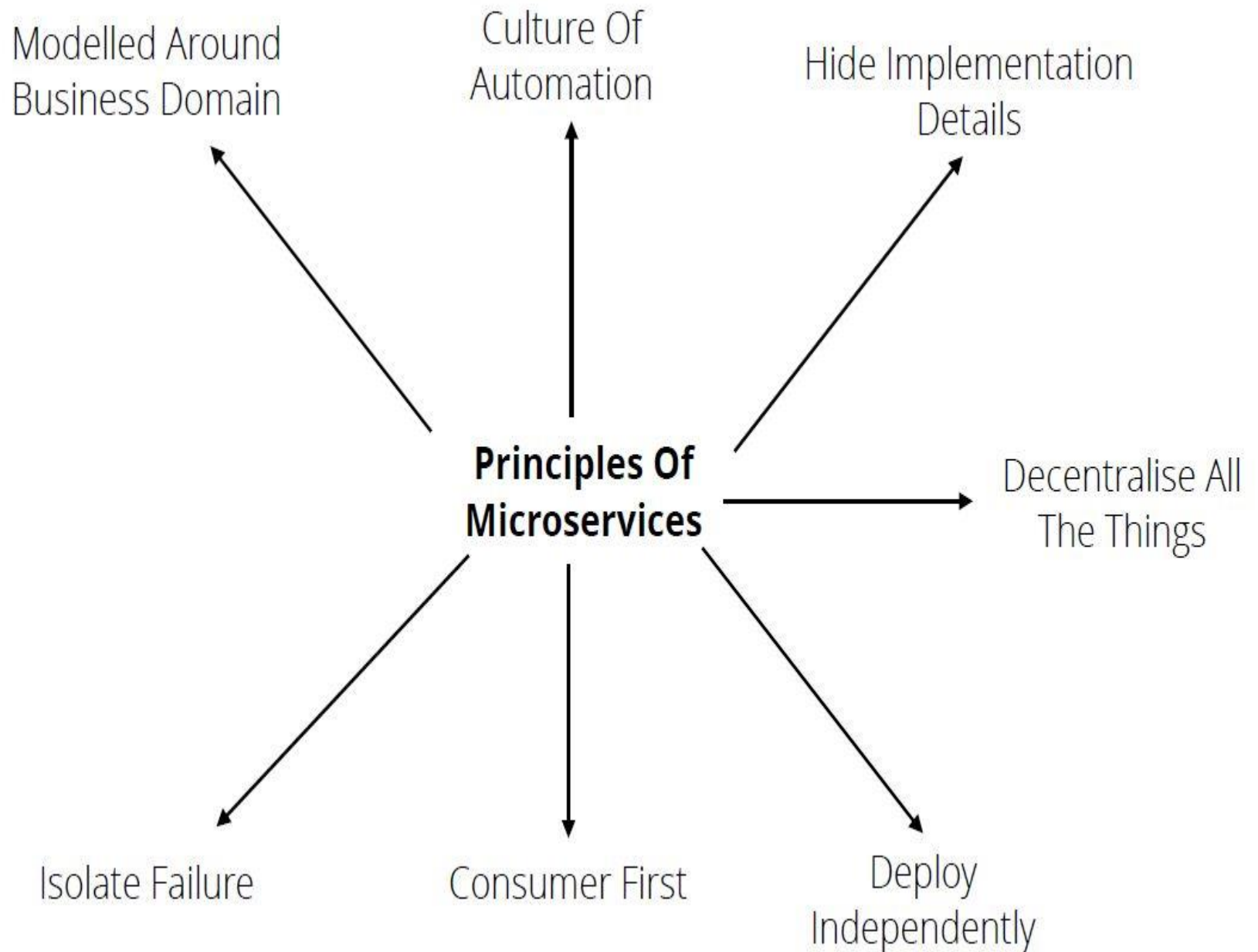
**Principles Of  
Microservices**

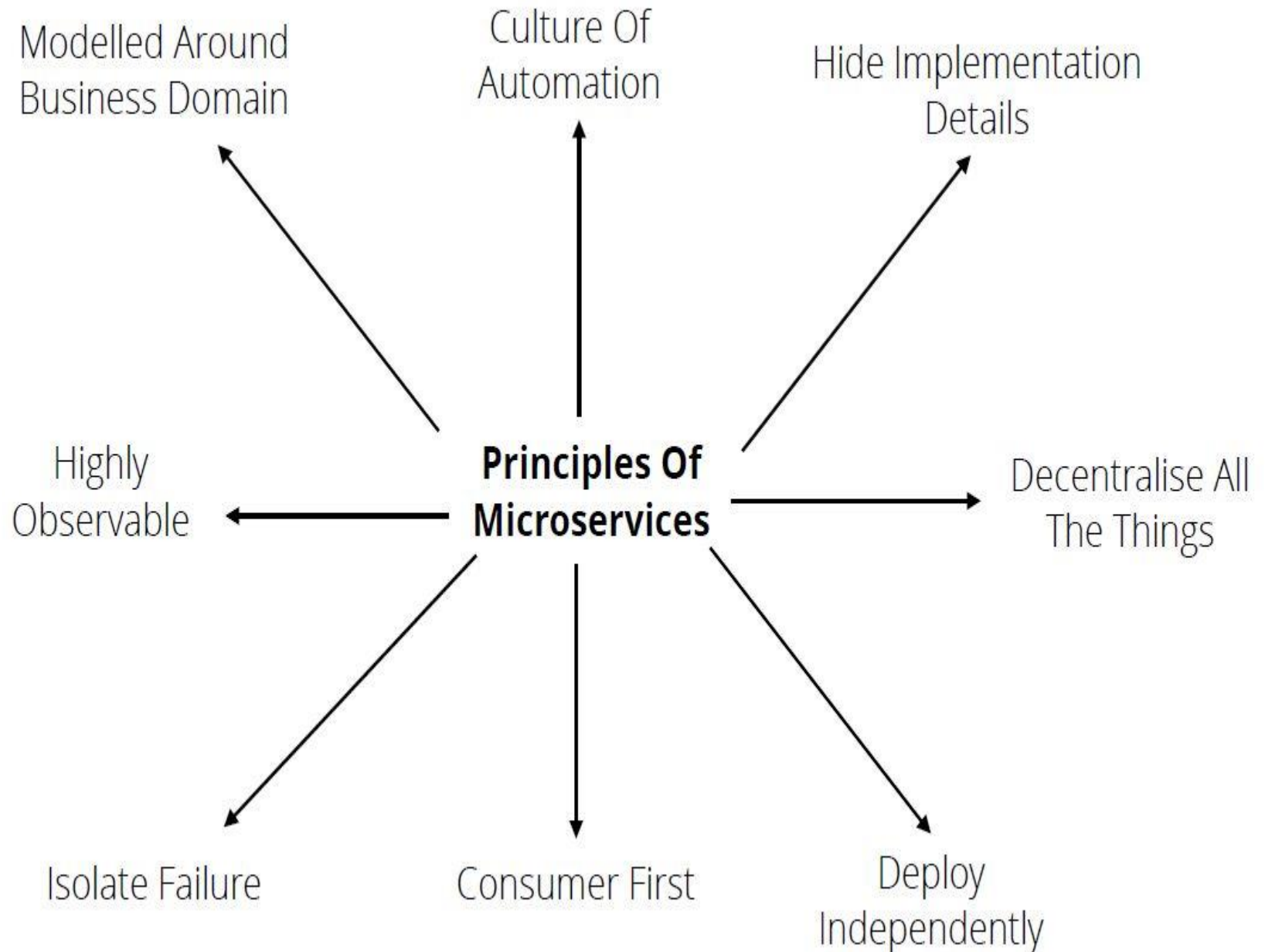
Decentralise All  
The Things

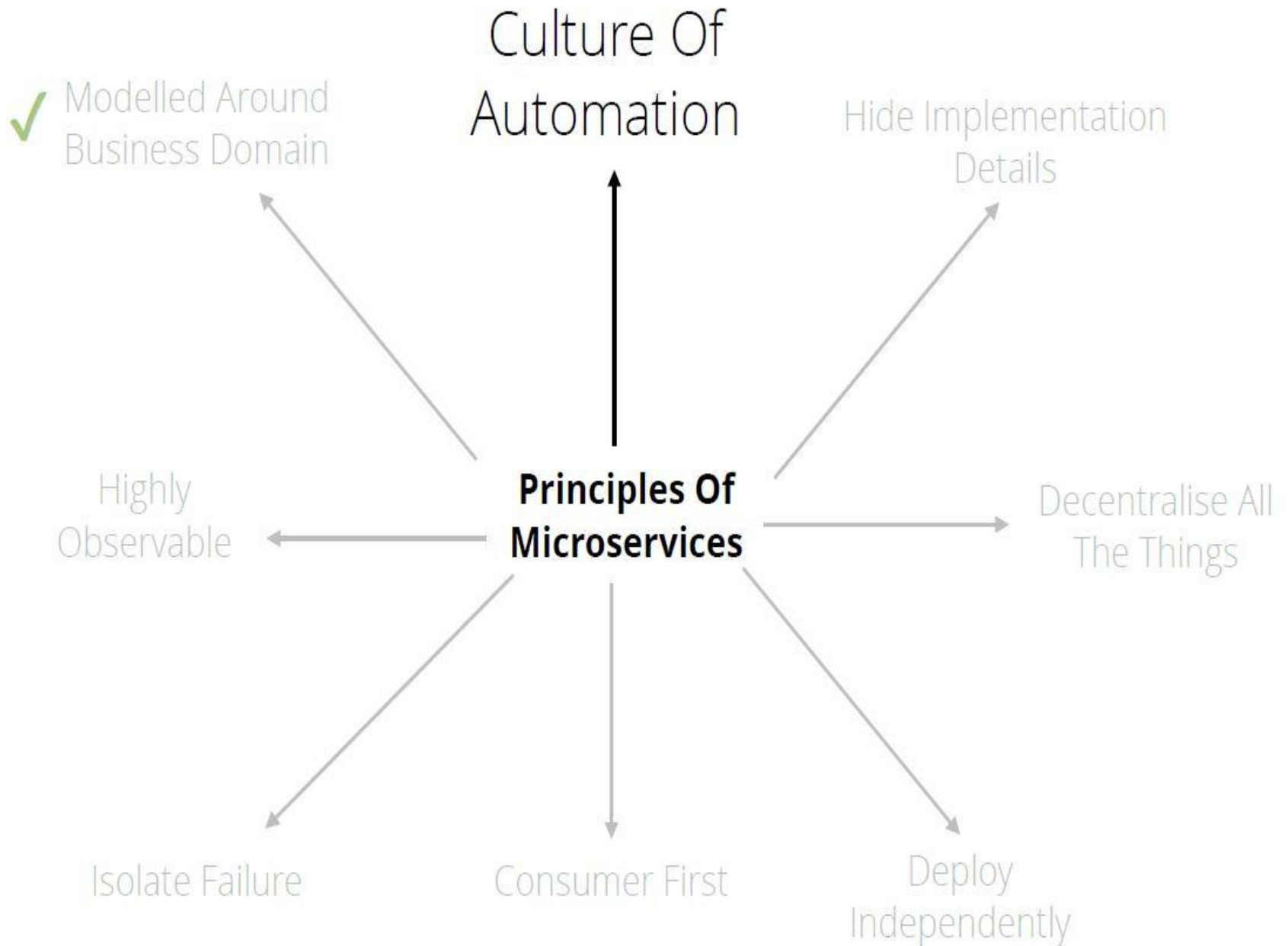
Consumer First

Deploy  
Independently









## ***2 Microservices***



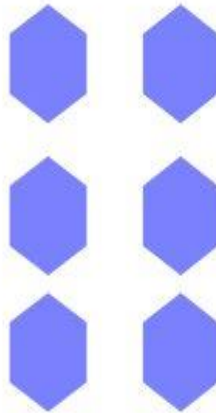
3 Months

***2 Microservices***



3 Months

***10 Microservices***



12 Months

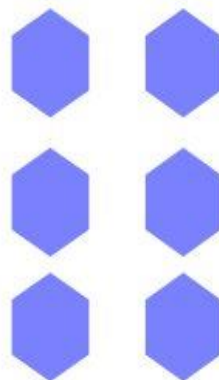


***2 Microservices***



3 Months

***10 Microservices***



12 Months

***60 Microservices***



18 Months

# Infrastructure Automation

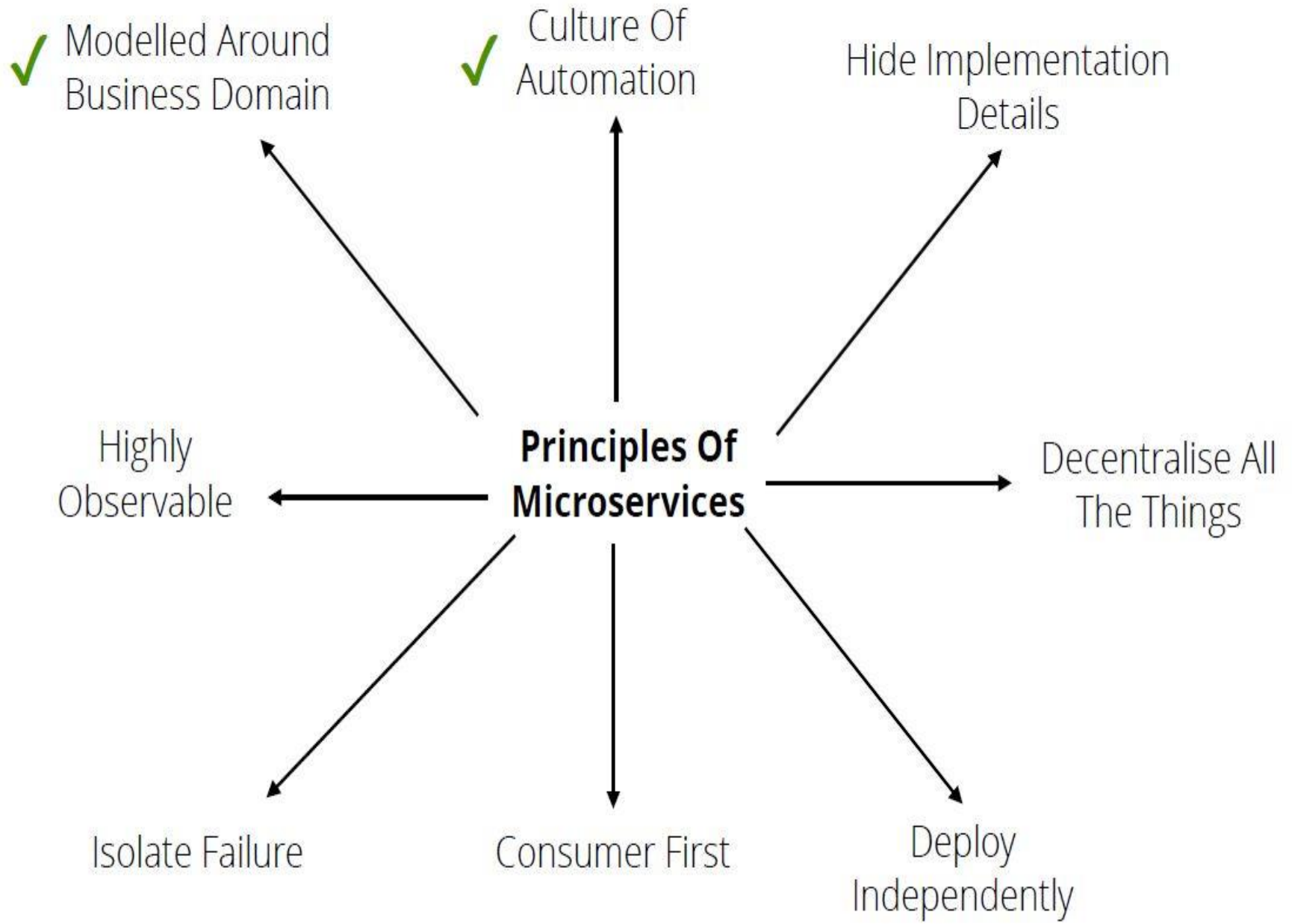
Infrastructure Automation

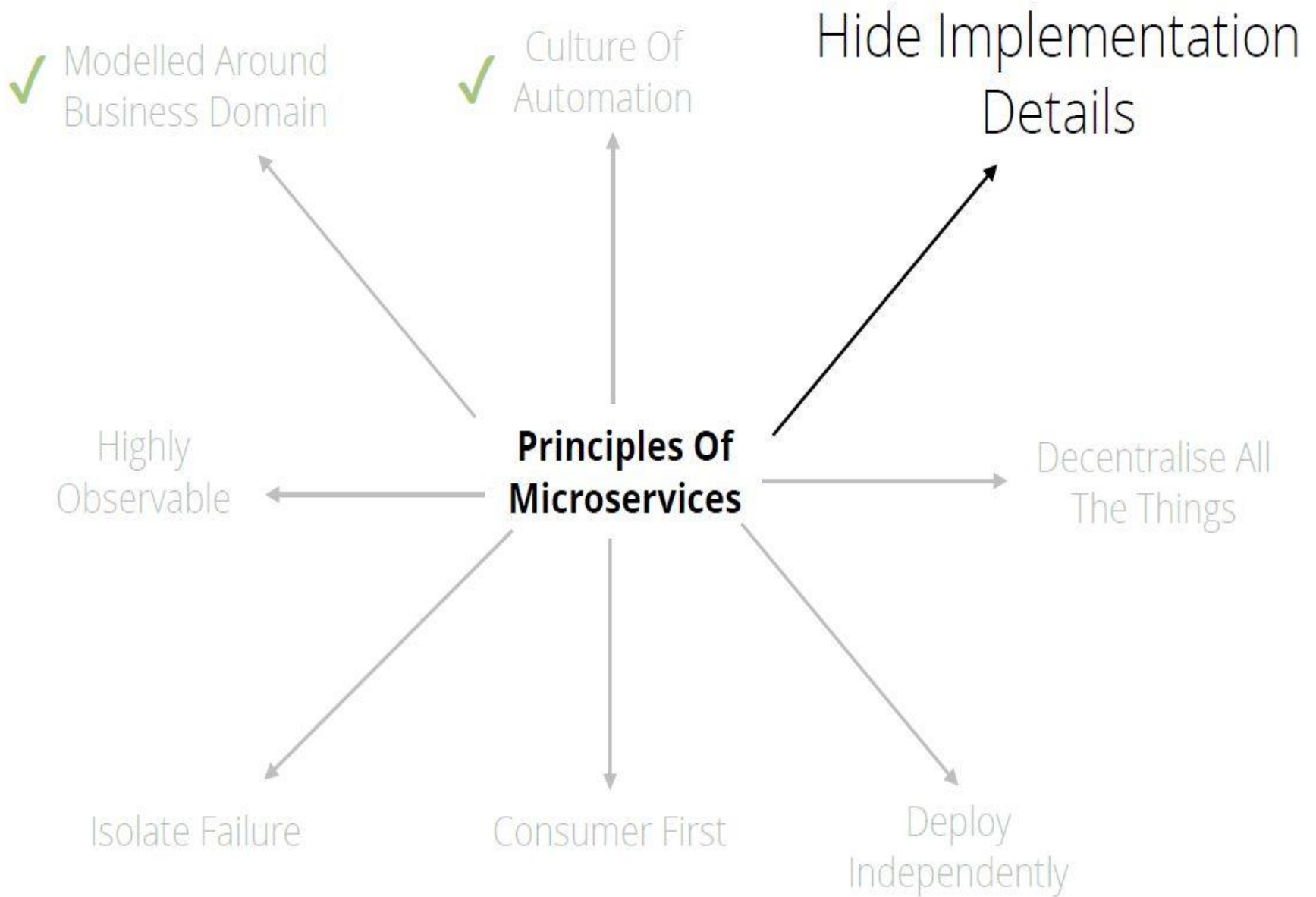
Automated Testing

Infrastructure Automation

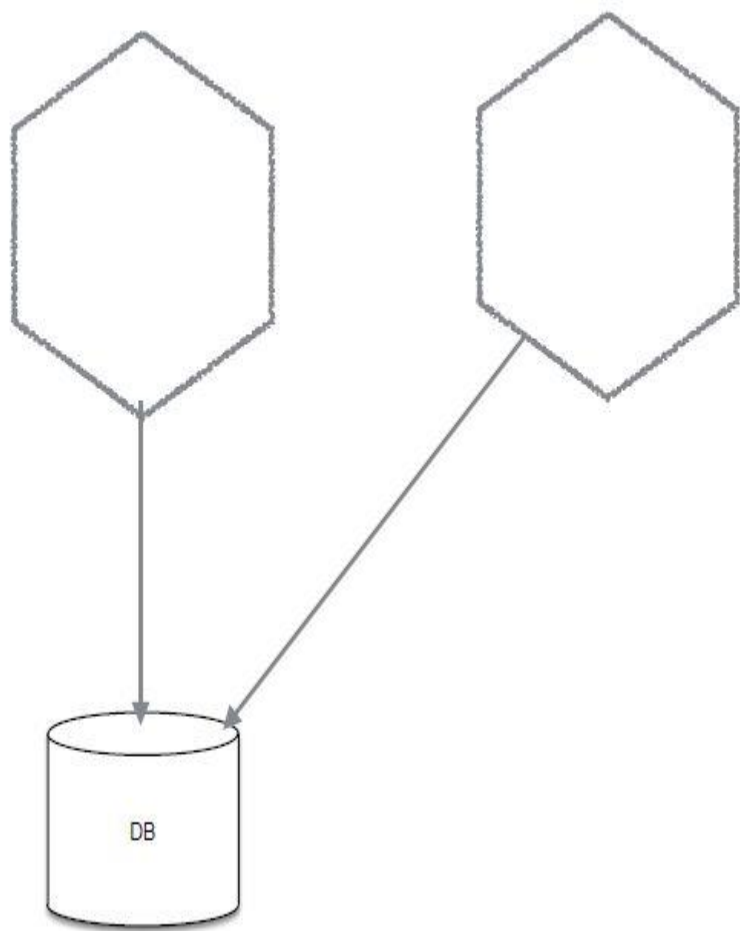
Automated Testing

Continuous Delivery

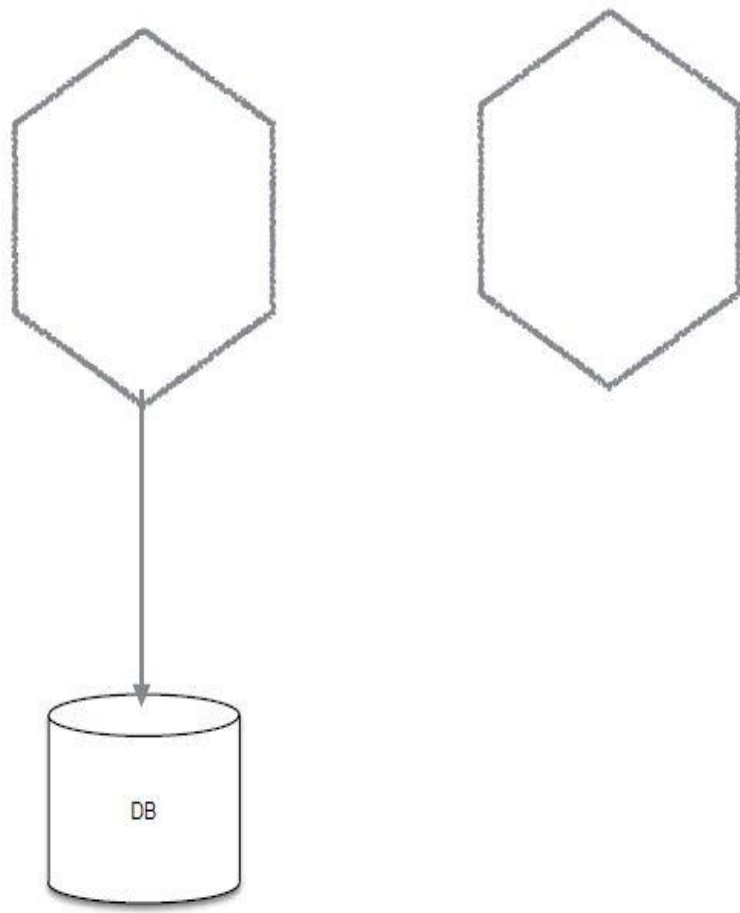


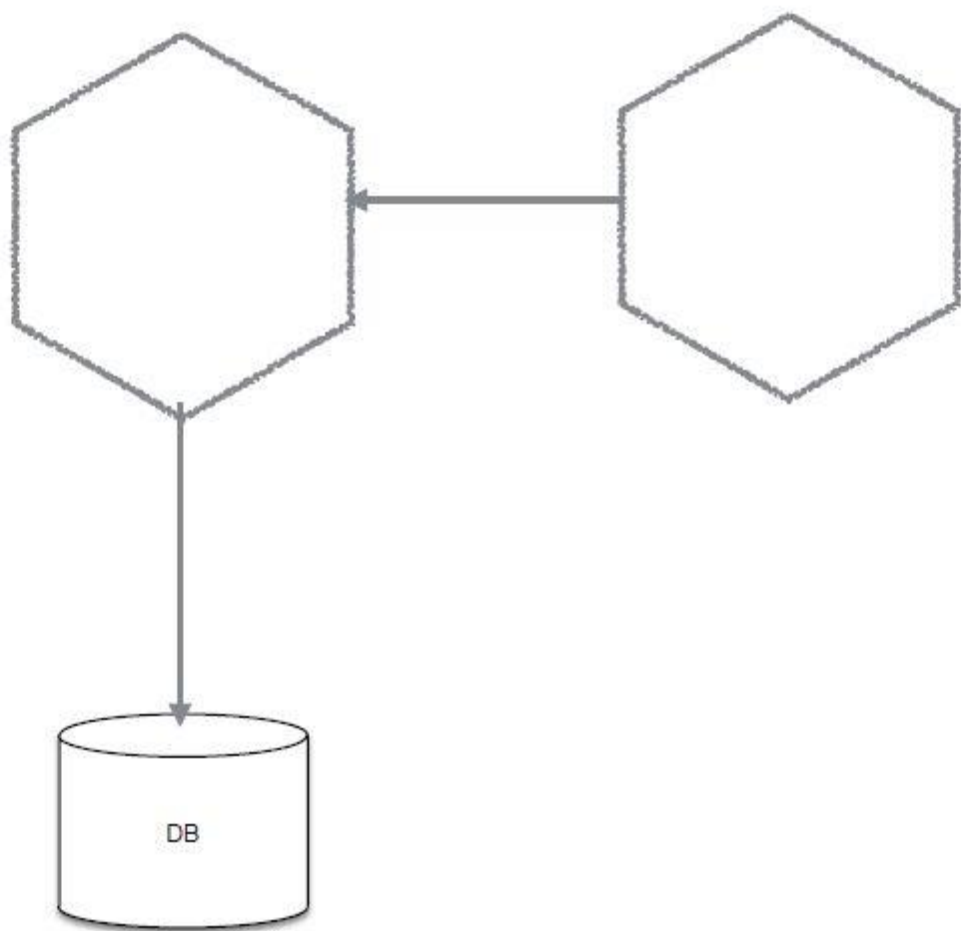




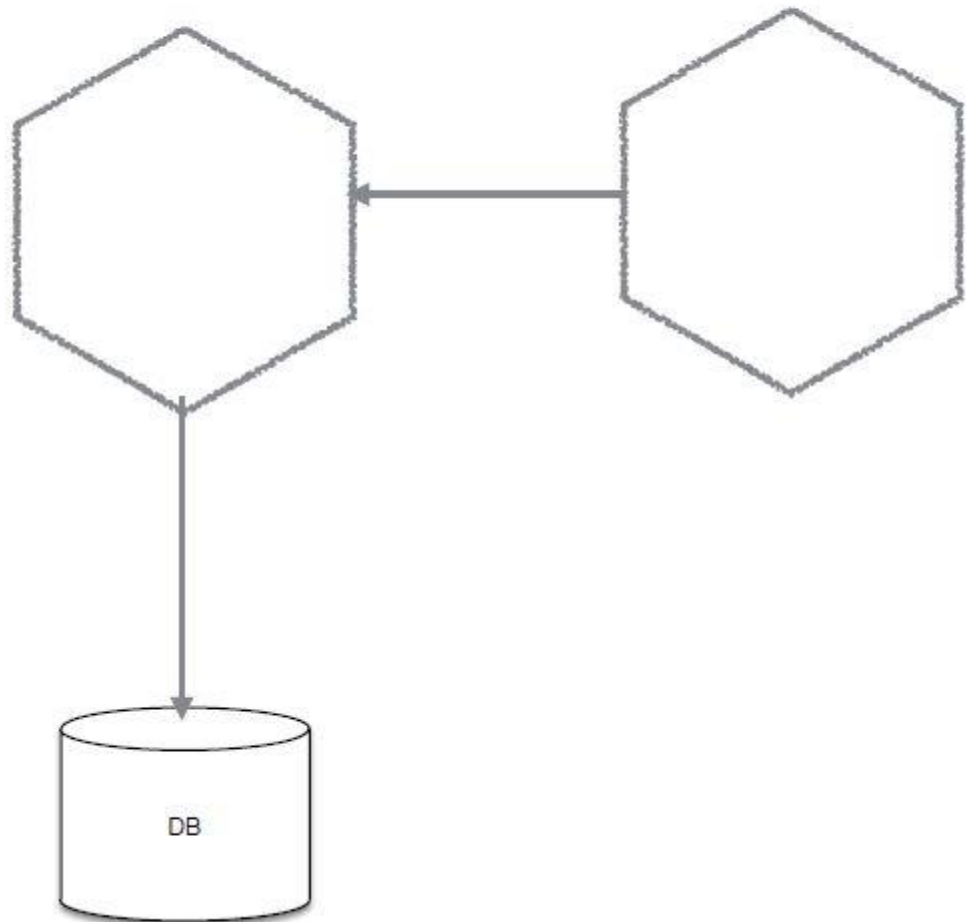




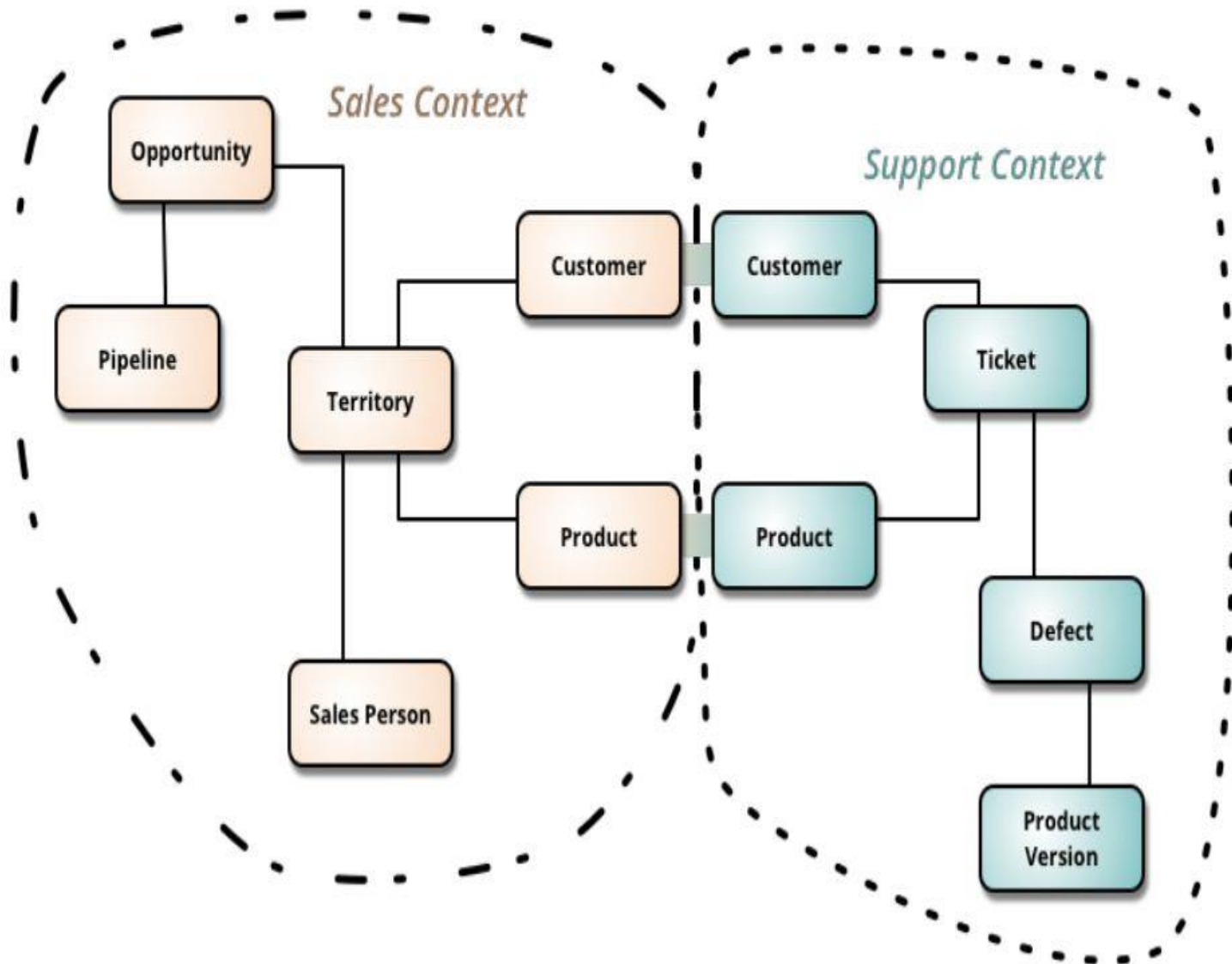




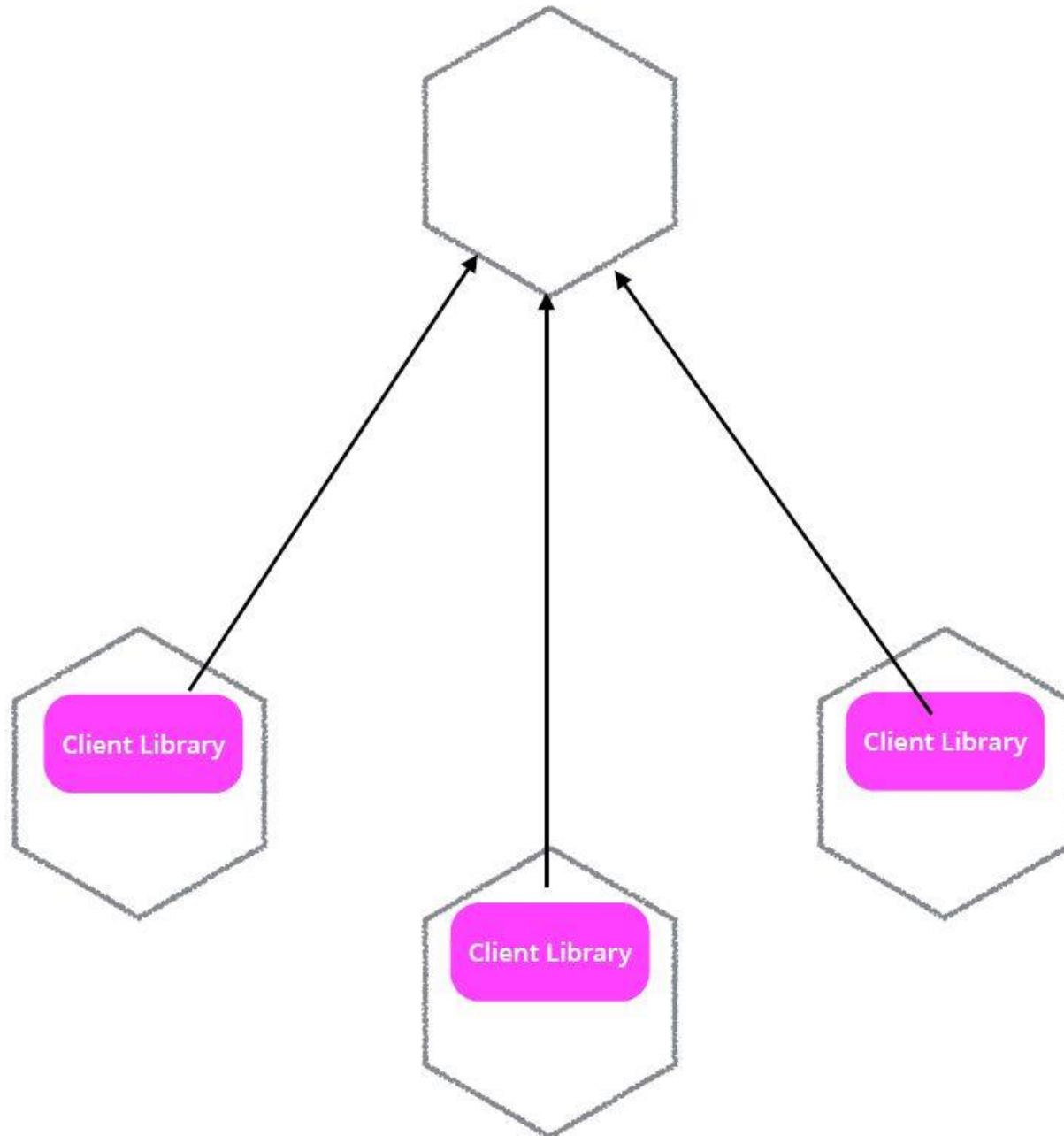
# HIDE YOUR DATABASE

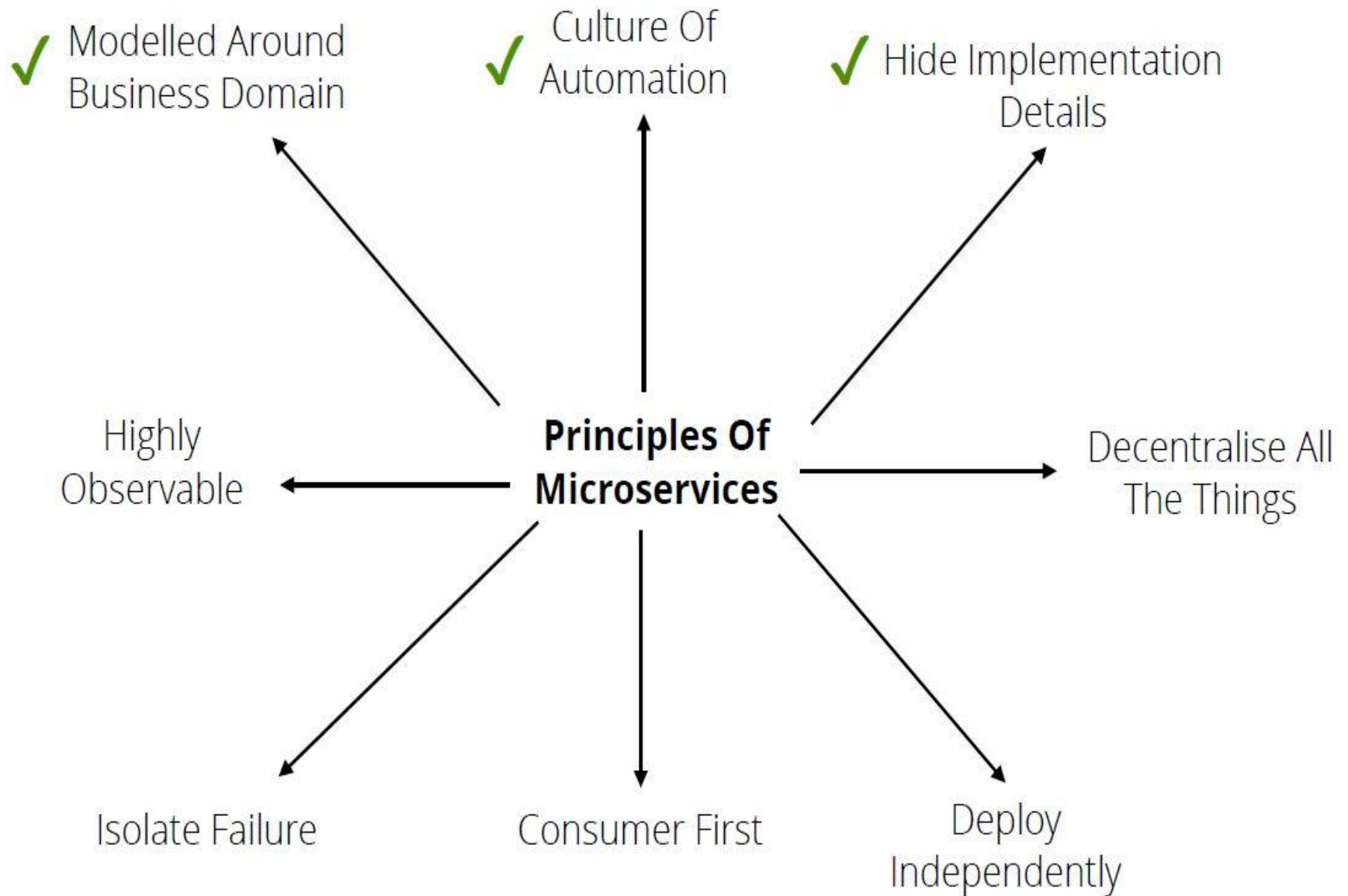


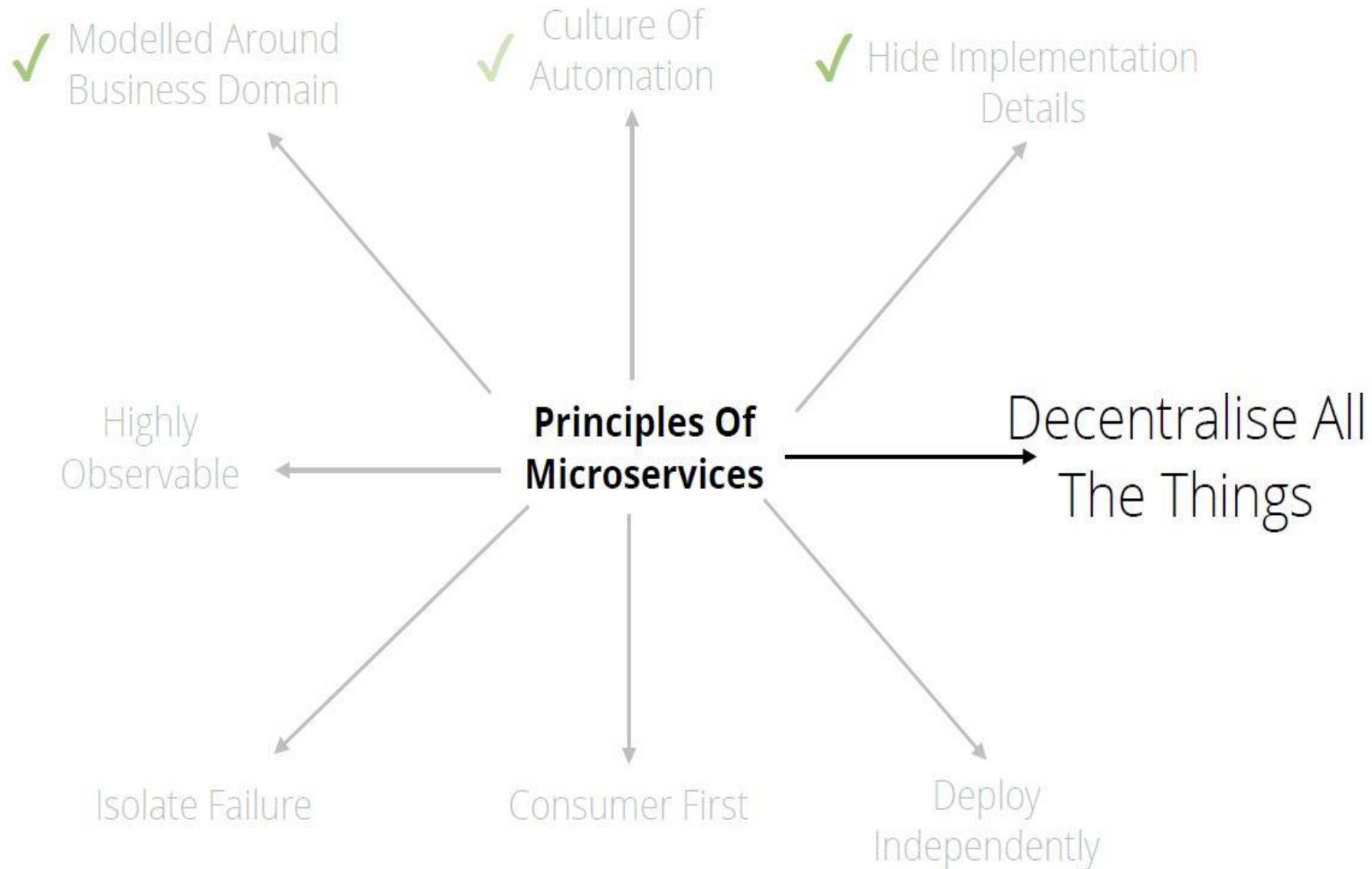
# BOUNDED CONTEXTS



## BE CAREFUL OF CLIENT LIBRARIES





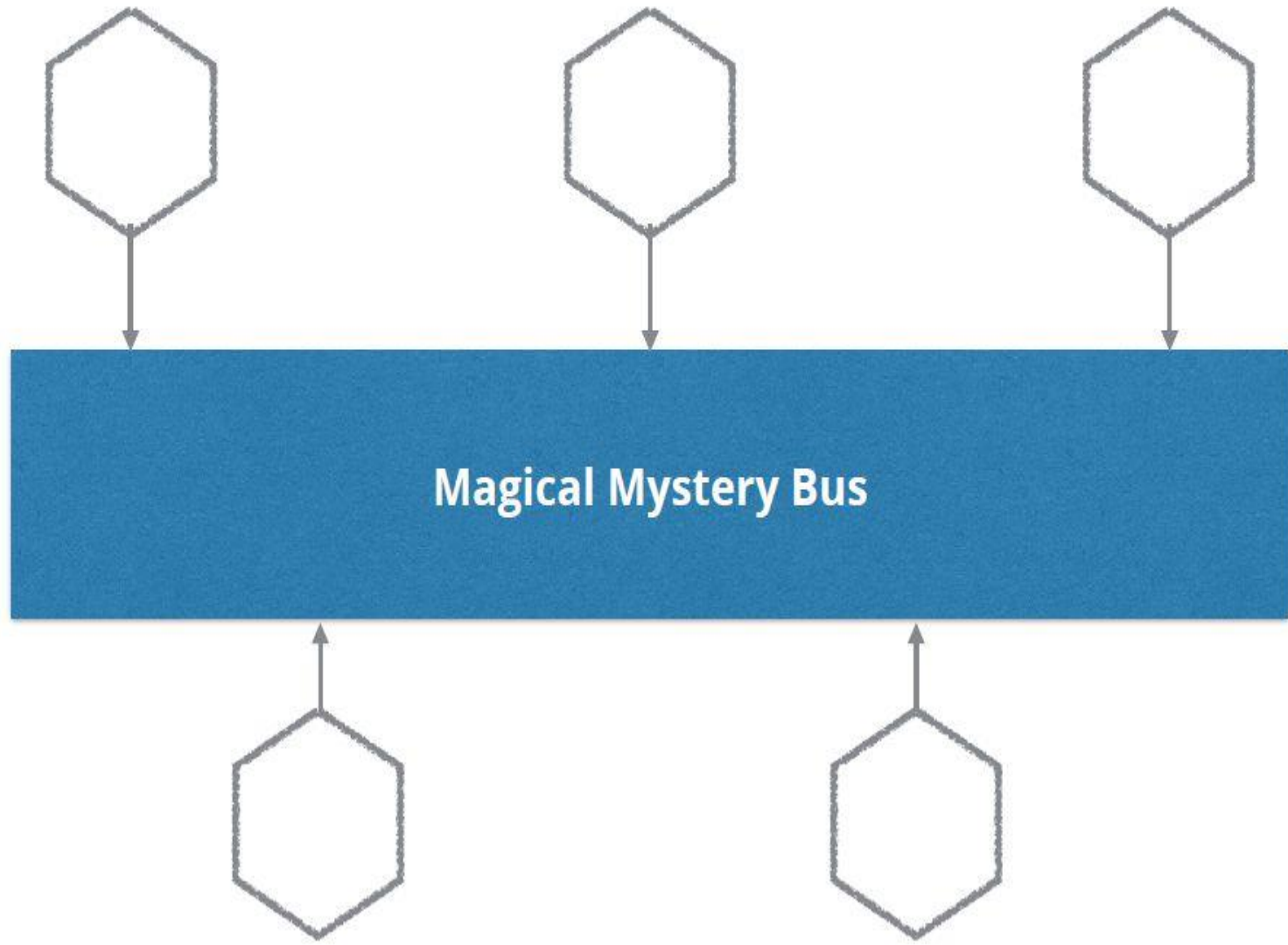


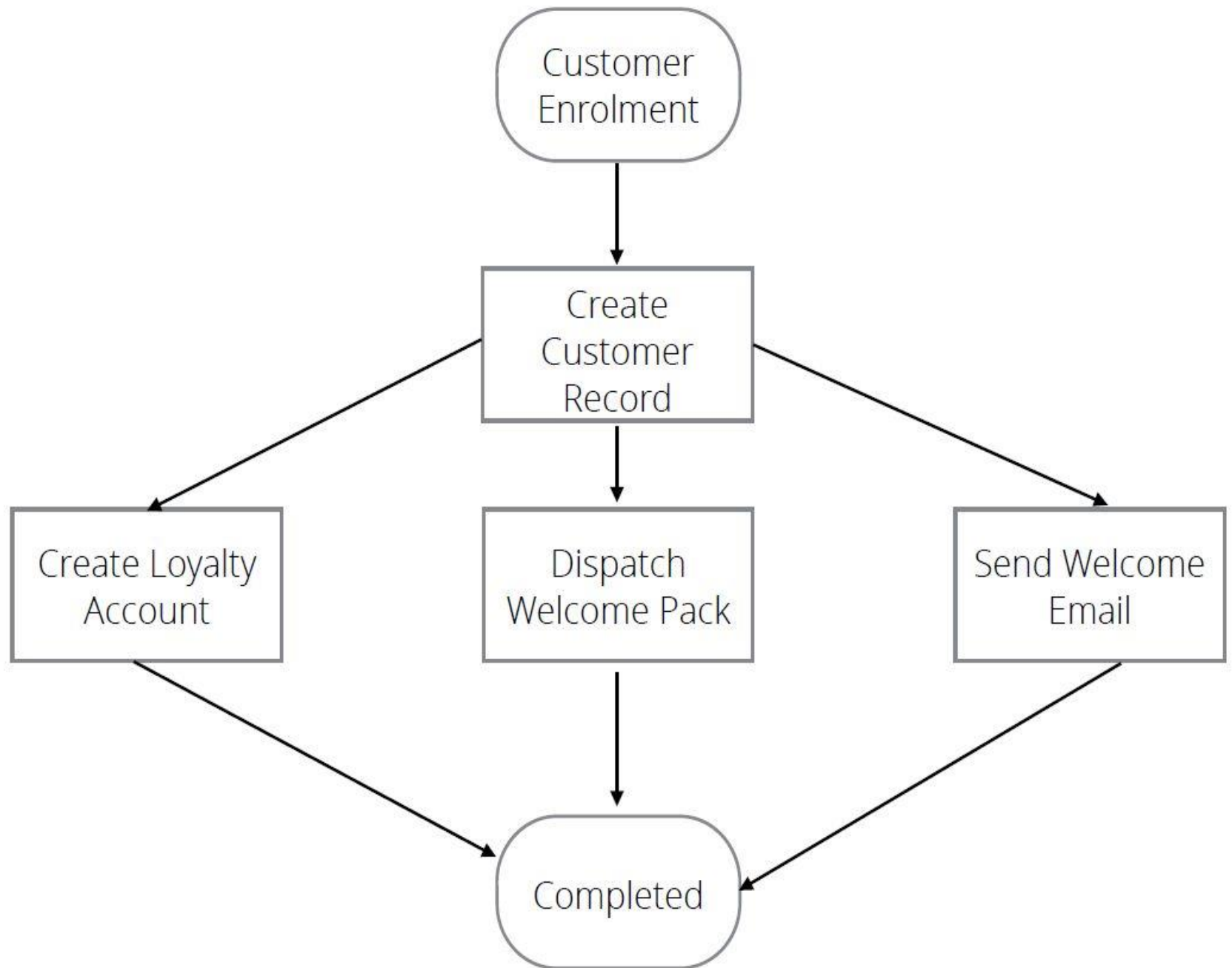
What is autonomy?

Giving people as much freedom as possible  
to do the job at hand

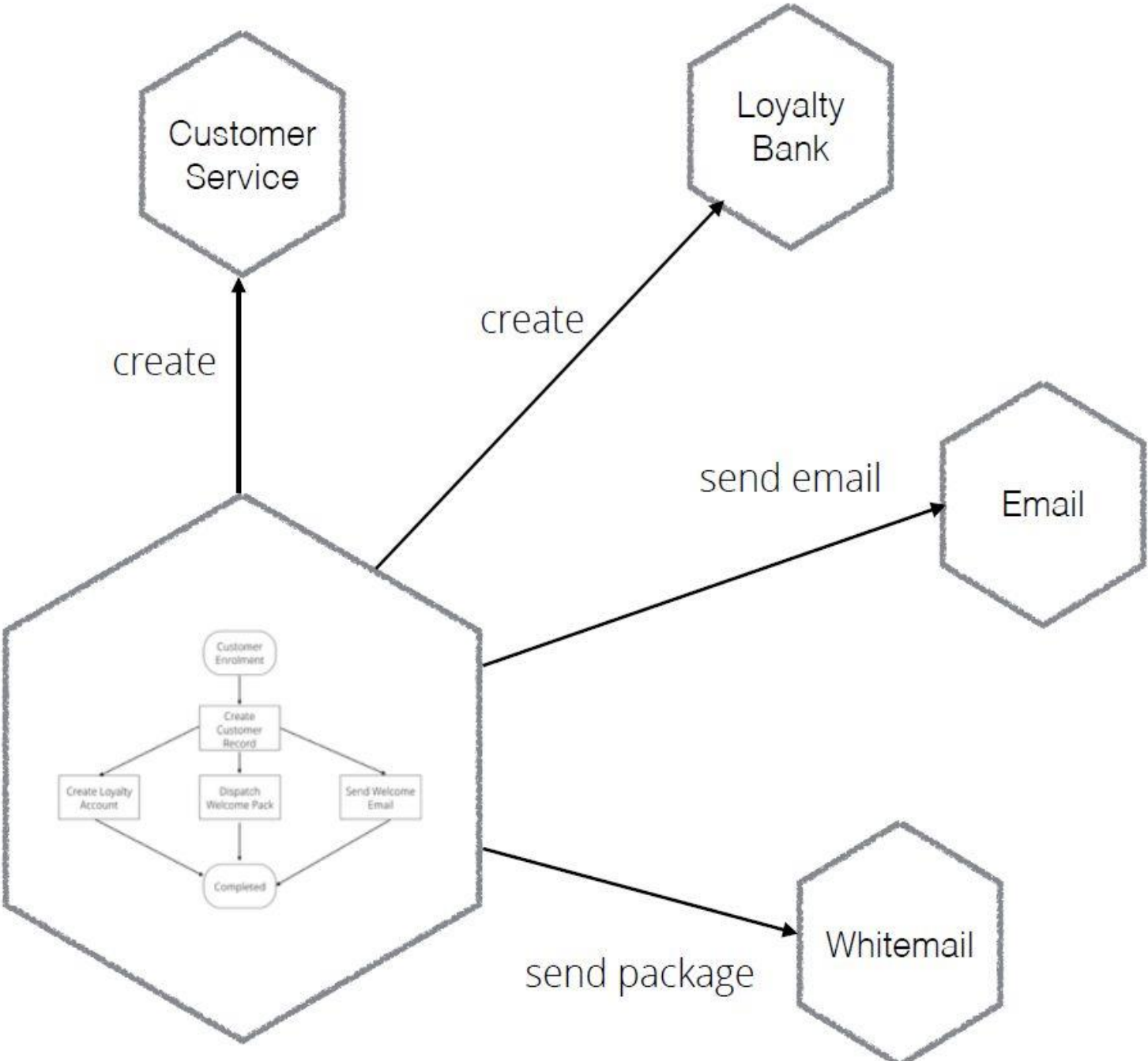


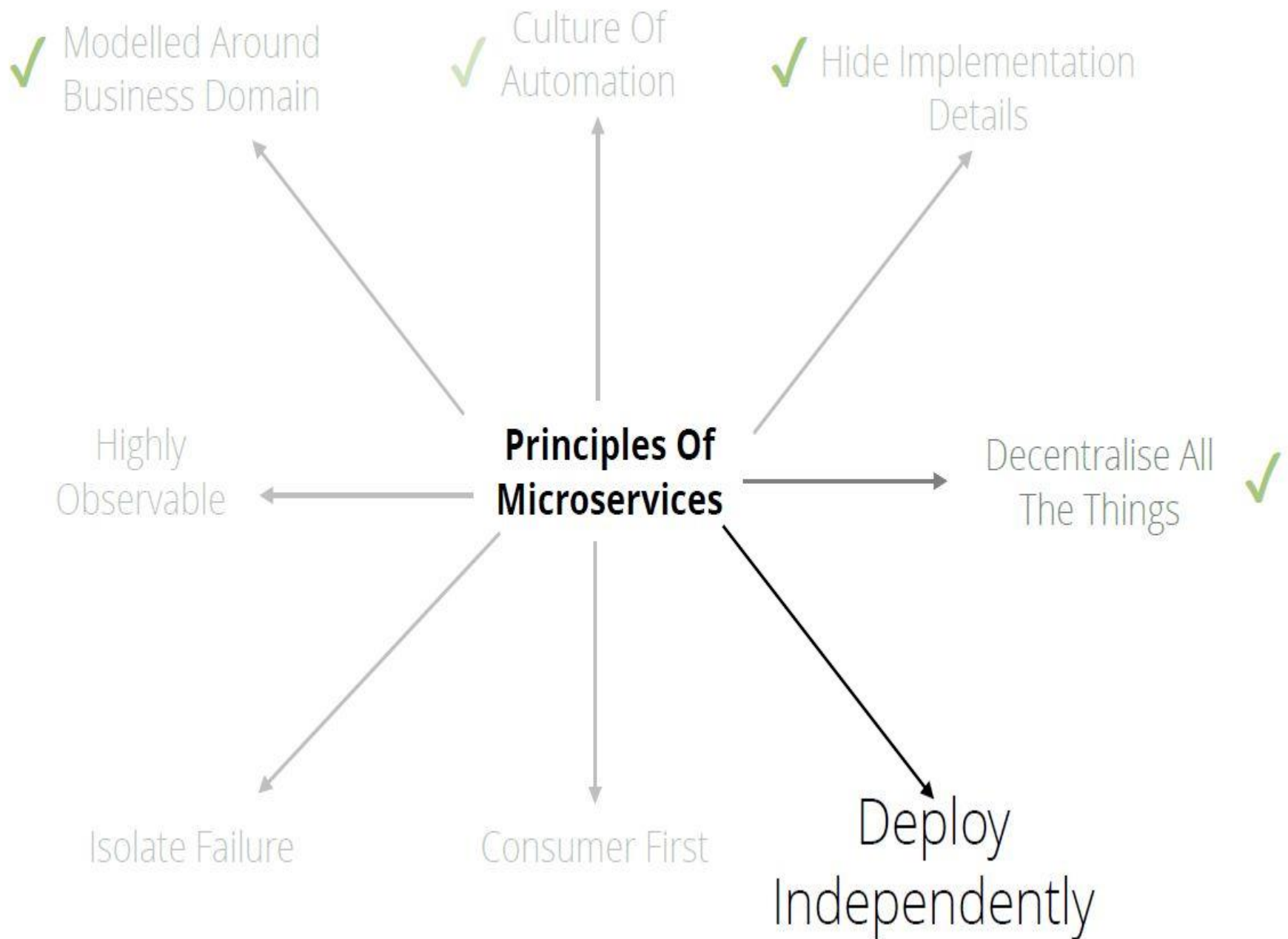
## DUMB-PIPES, SMART ENDPOINTS



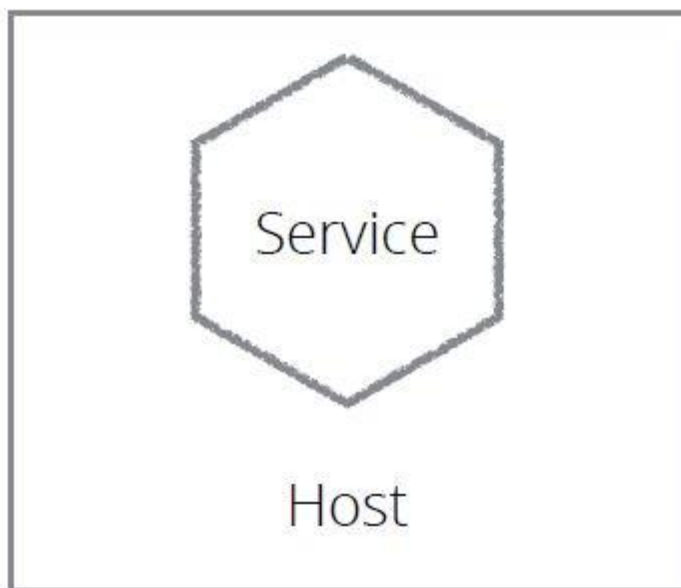


# ORCHESTRATION

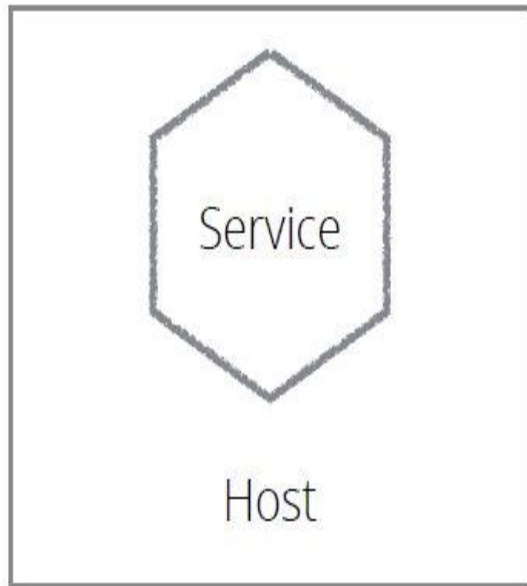




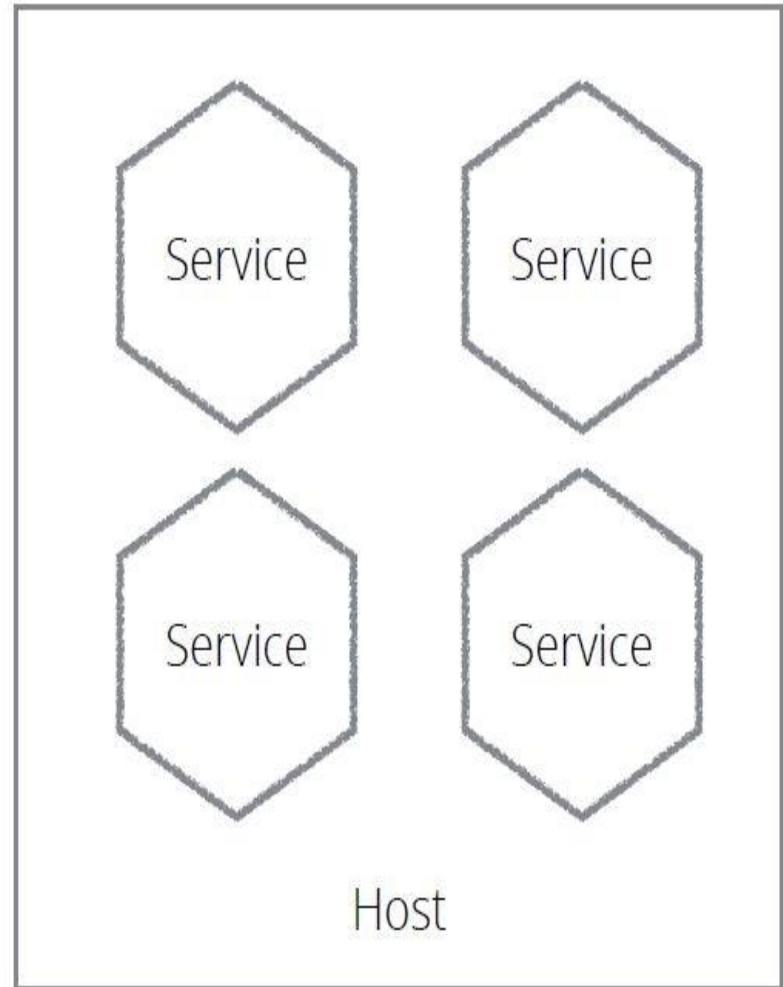
# ONE SERVICE PER-HOST



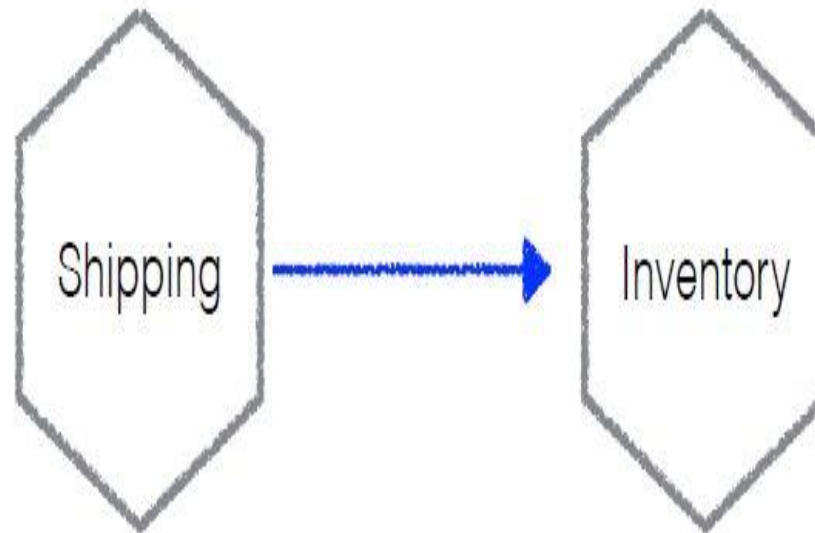
## ONE SERVICE PER-HOST



VS



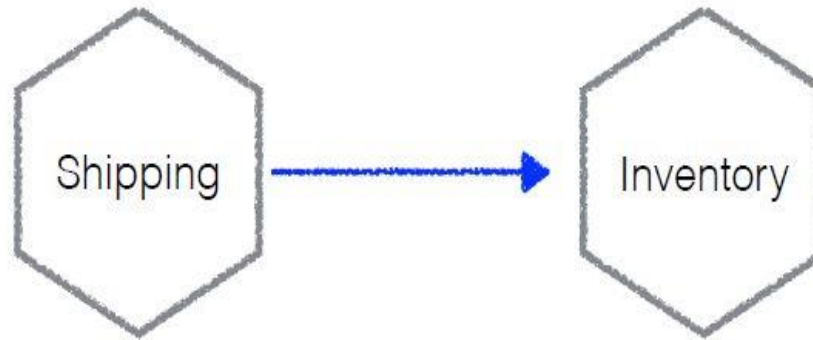
# CONSUMER-DRIVEN CONTRACTS







## CONSUMER-DRIVEN CONTRACTS



## Expectations



```

17 def concat( s1, s2 ):
18     return s1 + s2
19
20 class Node:
21     def __init__( self, data ):
22         self.data = data
23         self.next = None
24
25 class LinkedList:
26     def __init__( self ):
27         self.head = None
28
29     def insert( self, data ):
30         new_node = Node( data )
31         new_node.next = self.head
32         self.head = new_node
33
34     def display( self ):
35         if self.head is None:
36             print "List is empty"
37         else:
38             temp = self.head
39             while temp:
40                 print temp.data,
41                 temp = temp.next
42
43     def search( self, data ):
44         temp = self.head
45         while temp:
46             if temp.data == data:
47                 return True
48             temp = temp.next
49         return False
50
51     def delete( self, data ):
52         temp = self.head
53         prev = None
54         while temp:
55             if temp.data == data:
56                 if prev is None:
57                     self.head = temp.next
58                 else:
59                     prev.next = temp.next
60             prev = temp
61             temp = temp.next
62
63     def reverse( self ):
64         prev = None
65         temp = self.head
66         while temp:
67             next = temp.next
68             temp.next = prev
69             prev = temp
70             temp = next
71
72     def merge( self, l2 ):
73         if self.head is None:
74             self.head = l2.head
75         else:
76             temp = self.head
77             while temp.next:
78                 temp = temp.next
79             temp.next = l2.head
80
81     def sort( self ):
82         if self.head is None:
83             return
84         if self.head.next is None:
85             return
86         temp = self.head
87         while temp:
88             next = temp.next
89             temp.next = None
90             temp = next
91
92     def merge_sort( self ):
93         if self.head is None:
94             return
95         if self.head.next is None:
96             return
97         temp = self.head
98         while temp:
99             next = temp.next
100             temp.next = None
101             temp = next
102
103     def merge_sort_recursive( self ):
104         if self.head is None:
105             return
106         if self.head.next is None:
107             return
108         temp = self.head
109         while temp:
110             next = temp.next
111             temp.next = None
112             temp = next
113
114     def merge_sort_recursive(self):
115         if self.head is None:
116             return
117         if self.head.next is None:
118             return
119         temp = self.head
120         while temp:
121             next = temp.next
122             temp.next = None
123             temp = next
124
125     def merge_sort_recursive(self):
126         if self.head is None:
127             return
128         if self.head.next is None:
129             return
130         temp = self.head
131         while temp:
132             next = temp.next
133             temp.next = None
134             temp = next
135
136     def merge_sort_recursive(self):
137         if self.head is None:
138             return
139         if self.head.next is None:
140             return
141         temp = self.head
142         while temp:
143             next = temp.next
144             temp.next = None
145             temp = next
146
147     def merge_sort_recursive(self):
148         if self.head is None:
149             return
150         if self.head.next is None:
151             return
152         temp = self.head
153         while temp:
154             next = temp.next
155             temp.next = None
156             temp = next
157
158     def merge_sort_recursive(self):
159         if self.head is None:
160             return
161         if self.head.next is None:
162             return
163         temp = self.head
164         while temp:
165             next = temp.next
166             temp.next = None
167             temp = next
168
169     def merge_sort_recursive(self):
170         if self.head is None:
171             return
172         if self.head.next is None:
173             return
174         temp = self.head
175         while temp:
176             next = temp.next
177             temp.next = None
178             temp = next
179
180     def merge_sort_recursive(self):
181         if self.head is None:
182             return
183         if self.head.next is None:
184             return
185         temp = self.head
186         while temp:
187             next = temp.next
188             temp.next = None
189             temp = next
190
191     def merge_sort_recursive(self):
192         if self.head is None:
193             return
194         if self.head.next is None:
195             return
196         temp = self.head
197         while temp:
198             next = temp.next
199             temp.next = None
200             temp = next
201
202     def merge_sort_recursive(self):
203         if self.head is None:
204             return
205         if self.head.next is None:
206             return
207         temp = self.head
208         while temp:
209             next = temp.next
210             temp.next = None
211             temp = next
212
213     def merge_sort_recursive(self):
214         if self.head is None:
215             return
216         if self.head.next is None:
217             return
218         temp = self.head
219         while temp:
220             next = temp.next
221             temp.next = None
222             temp = next
223
224     def merge_sort_recursive(self):
225         if self.head is None:
226             return
227         if self.head.next is None:
228             return
229         temp = self.head
230         while temp:
231             next = temp.next
232             temp.next = None
233             temp = next
234
235     def merge_sort_recursive(self):
236         if self.head is None:
237             return
238         if self.head.next is None:
239             return
240         temp = self.head
241         while temp:
242             next = temp.next
243             temp.next = None
244             temp = next
245
246     def merge_sort_recursive(self):
247         if self.head is None:
248             return
249         if self.head.next is None:
250             return
251         temp = self.head
252         while temp:
253             next = temp.next
254             temp.next = None
255             temp = next
256
257     def merge_sort_recursive(self):
258         if self.head is None:
259             return
260         if self.head.next is None:
261             return
262         temp = self.head
263         while temp:
264             next = temp.next
265             temp.next = None
266             temp = next
267
268     def merge_sort_recursive(self):
269         if self.head is None:
270             return
271         if self.head.next is None:
272             return
273         temp = self.head
274         while temp:
275             next = temp.next
276             temp.next = None
277             temp = next
278
279     def merge_sort_recursive(self):
280         if self.head is None:
281             return
282         if self.head.next is None:
283             return
284         temp = self.head
285         while temp:
286             next = temp.next
287             temp.next = None
288             temp = next
289
290     def merge_sort_recursive(self):
291         if self.head is None:
292             return
293         if self.head.next is None:
294             return
295         temp = self.head
296         while temp:
297             next = temp.next
298             temp.next = None
299             temp = next
300
301     def merge_sort_recursive(self):
302         if self.head is None:
303             return
304         if self.head.next is None:
305             return
306         temp = self.head
307         while temp:
308             next = temp.next
309             temp.next = None
310             temp = next
311
312     def merge_sort_recursive(self):
313         if self.head is None:
314             return
315         if self.head.next is None:
316             return
317         temp = self.head
318         while temp:
319             next = temp.next
320             temp.next = None
321             temp = next
322
323     def merge_sort_recursive(self):
324         if self.head is None:
325             return
326         if self.head.next is None:
327             return
328         temp = self.head
329         while temp:
330             next = temp.next
331             temp.next = None
332             temp = next
333
334     def merge_sort_recursive(self):
335         if self.head is None:
336             return
337         if self.head.next is None:
338             return
339         temp = self.head
340         while temp:
341             next = temp.next
342             temp.next = None
343             temp = next
344
345     def merge_sort_recursive(self):
346         if self.head is None:
347             return
348         if self.head.next is None:
349             return
350         temp = self.head
351         while temp:
352             next = temp.next
353             temp.next = None
354             temp = next
355
356     def merge_sort_recursive(self):
357         if self.head is None:
358             return
359         if self.head.next is None:
360             return
361         temp = self.head
362         while temp:
363             next = temp.next
364             temp.next = None
365             temp = next
366
367     def merge_sort_recursive(self):
368         if self.head is None:
369             return
370         if self.head.next is None:
371             return
372         temp = self.head
373         while temp:
374             next = temp.next
375             temp.next = None
376             temp = next
377
378     def merge_sort_recursive(self):
379         if self.head is None:
380             return
381         if self.head.next is None:
382             return
383         temp = self.head
384         while temp:
385             next = temp.next
386             temp.next = None
387             temp = next
388
389     def merge_sort_recursive(self):
390         if self.head is None:
391             return
392         if self.head.next is None:
393             return
394         temp = self.head
395         while temp:
396             next = temp.next
397             temp.next = None
398             temp = next
399
400     def merge_sort_recursive(self):
401         if self.head is None:
402             return
403         if self.head.next is None:
404             return
405         temp = self.head
406         while temp:
407             next = temp.next
408             temp.next = None
409             temp = next
410
411     def merge_sort_recursive(self):
412         if self.head is None:
413             return
414         if self.head.next is None:
415             return
416         temp = self.head
417         while temp:
418             next = temp.next
419             temp.next = None
420             temp = next
421
422     def merge_sort_recursive(self):
423         if self.head is None:
424             return
425         if self.head.next is None:
426             return
427         temp = self.head
428         while temp:
429             next = temp.next
430             temp.next = None
431             temp = next
432
433     def merge_sort_recursive(self):
434         if self.head is None:
435             return
436         if self.head.next is None:
437             return
438         temp = self.head
439         while temp:
440             next = temp.next
441             temp.next = None
442             temp = next
443
444     def merge_sort_recursive(self):
445         if self.head is None:
446             return
447         if self.head.next is None:
448             return
449         temp = self.head
450         while temp:
451             next = temp.next
452             temp.next = None
453             temp = next
454
455     def merge_sort_recursive(self):
456         if self.head is None:
457             return
458         if self.head.next is None:
459             return
460         temp = self.head
461         while temp:
462             next = temp.next
463             temp.next = None
464             temp = next
465
466     def merge_sort_recursive(self):
467         if self.head is None:
468             return
469         if self.head.next is None:
470             return
471         temp = self.head
472         while temp:
473             next = temp.next
474             temp.next = None
475             temp = next
476
477     def merge_sort_recursive(self):
478         if self.head is None:
479             return
480         if self.head.next is None:
481             return
482         temp = self.head
483         while temp:
484             next = temp.next
485             temp.next = None
486             temp = next
487
488     def merge_sort_recursive(self):
489         if self.head is None:
490             return
491         if self.head.next is None:
492             return
493         temp = self.head
494         while temp:
495             next = temp.next
496             temp.next = None
497             temp = next
498
499     def merge_sort_recursive(self):
500         if self.head is None:
501             return
502         if self.head.next is None:
503             return
504         temp = self.head
505         while temp:
506             next = temp.next
507             temp.next = None
508
```

# Pact

---

Define a pact between service consumers and providers, enabling "consumer driven contract" testing.

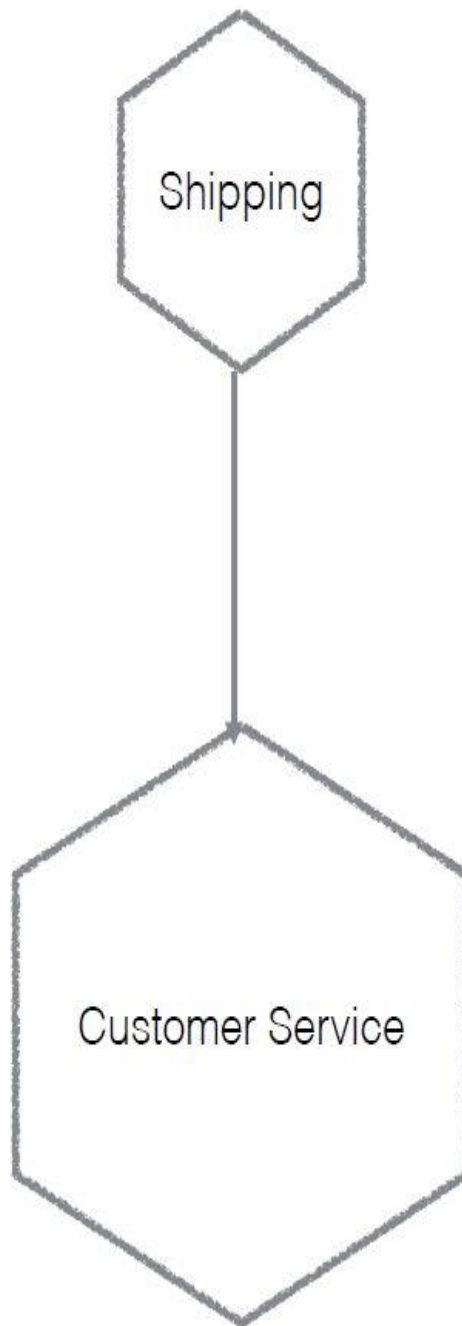
Pact provides an RSpec DSL for service consumers to define the HTTP requests they will make to a service provider and the HTTP responses they expect back. These expectations are used in the consumers specs to provide a mock service provider. The interactions are recorded, and played back in the service provider specs to ensure the service provider actually does provide the response the consumer expects.

This allows testing of both sides of an integration point using fast unit tests.

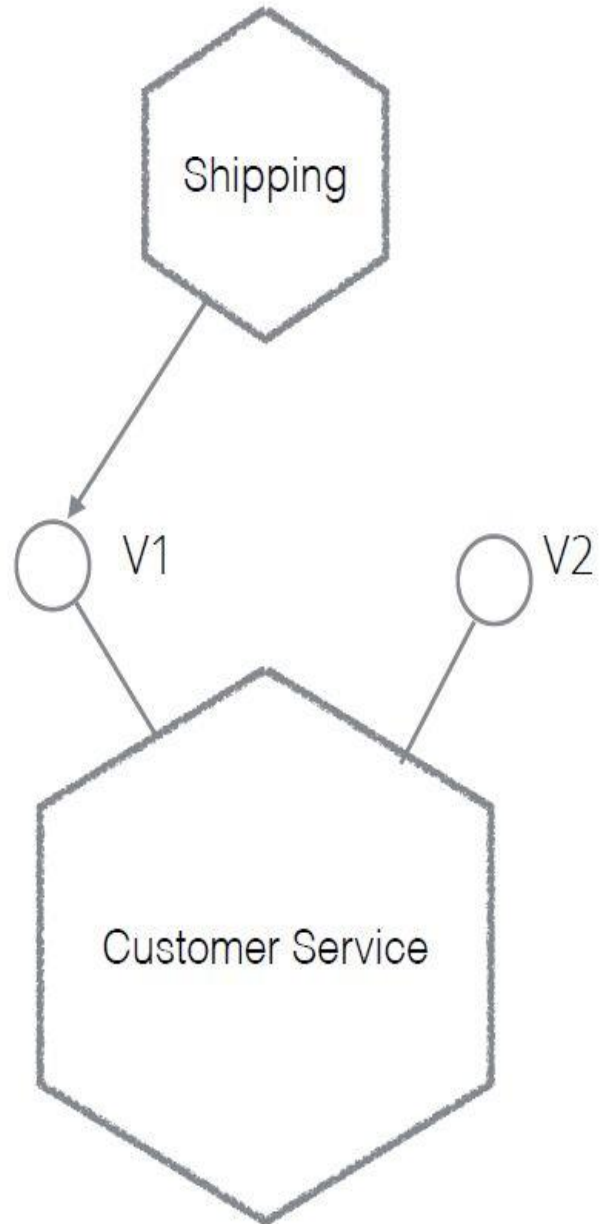
This gem is inspired by the concept of "Consumer driven contracts". See <http://martinfowler.com/articles/consumerDrivenContracts.html> for more information.

Travis CI Status:  build passing

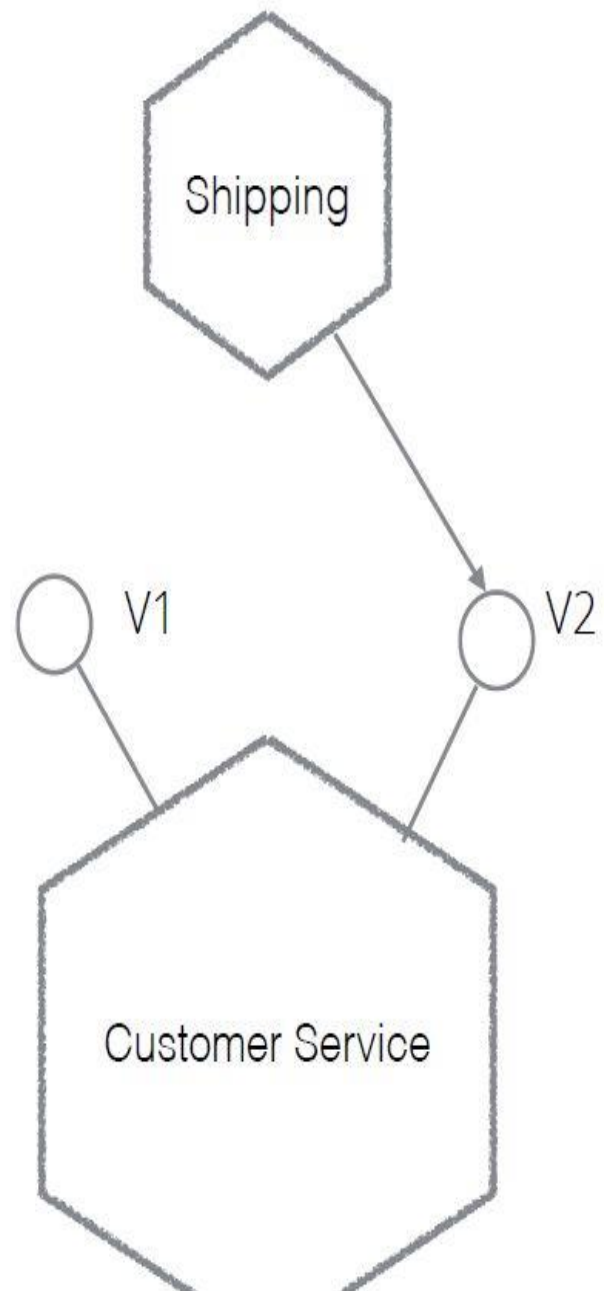
<https://github.com/realestate-com-au/pact>



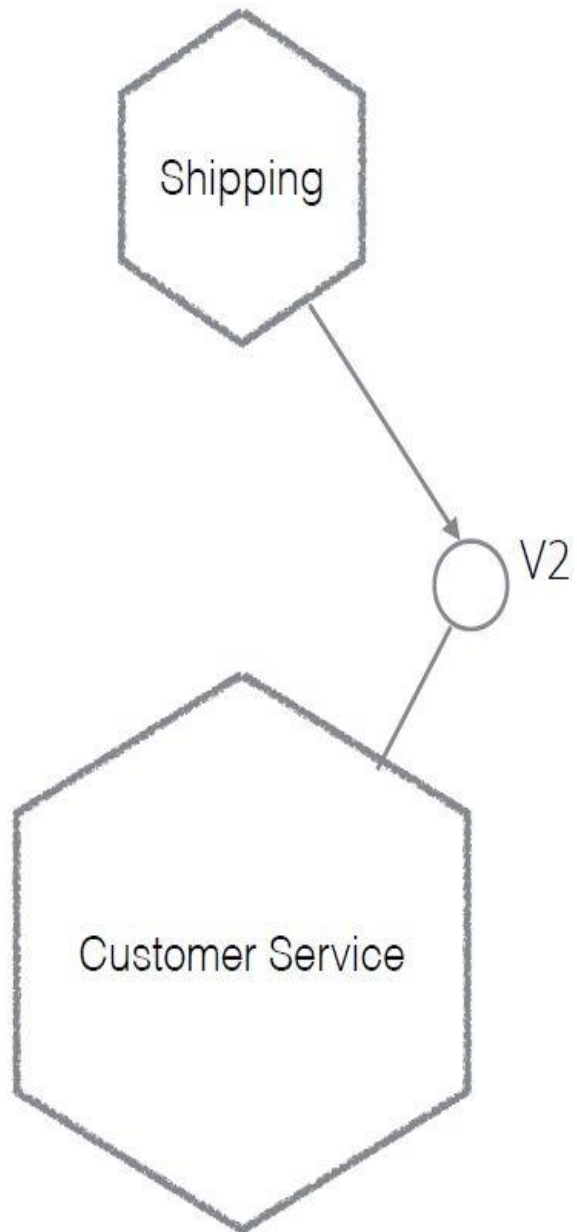
## CO-EXIST ENDPOINTS

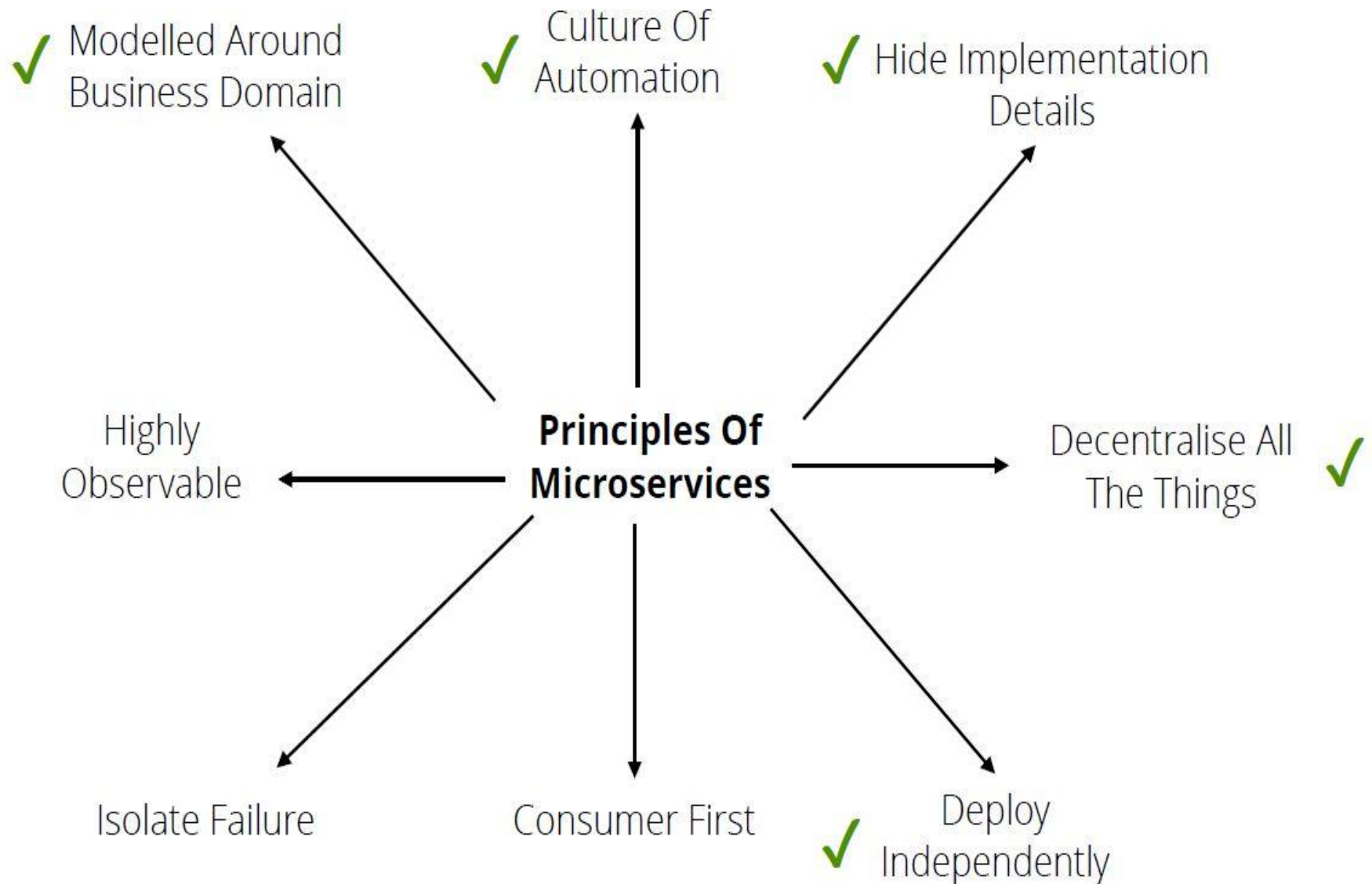


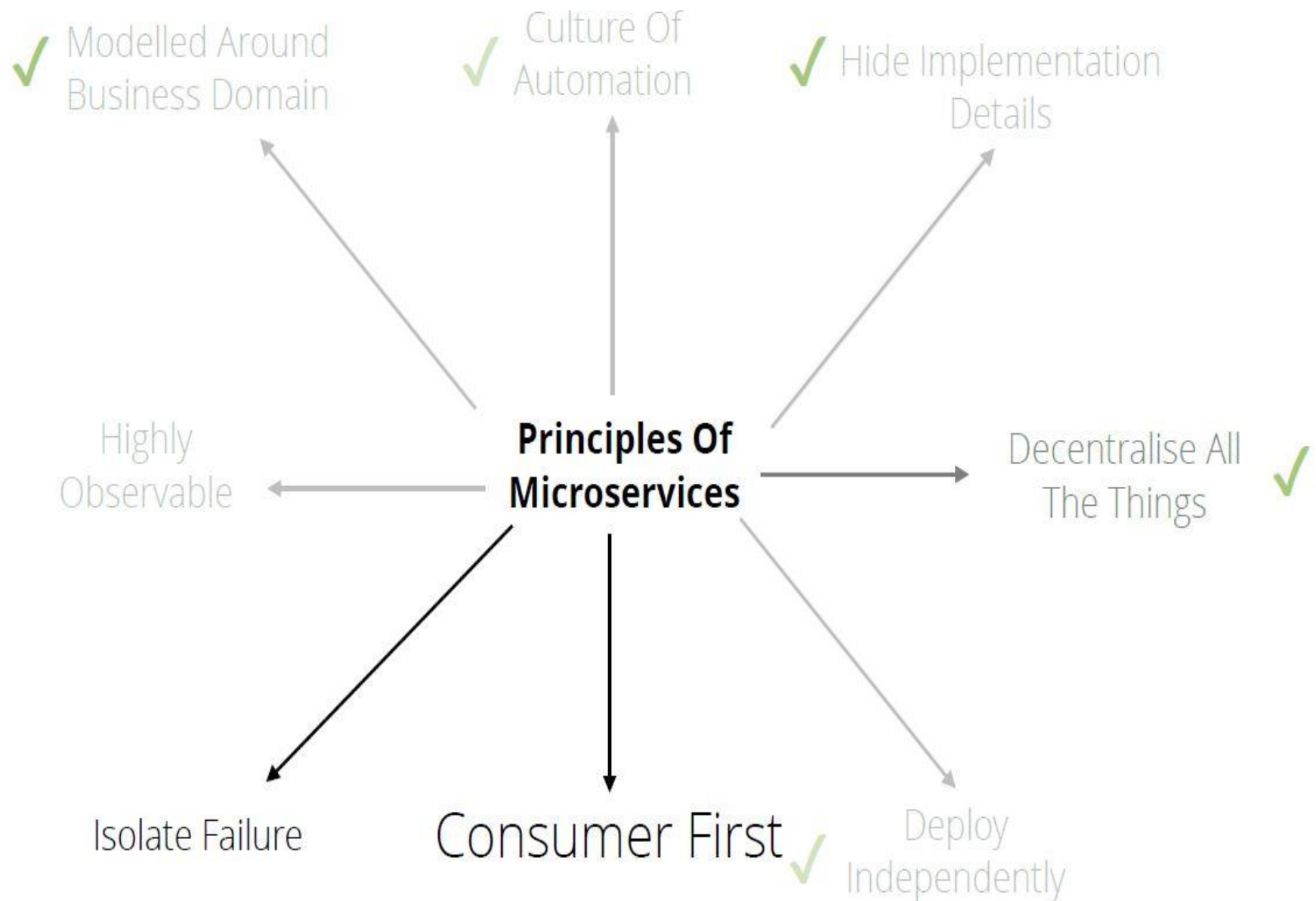
## CO-EXIST ENDPOINTS



## CO-EXIST ENDPOINTS







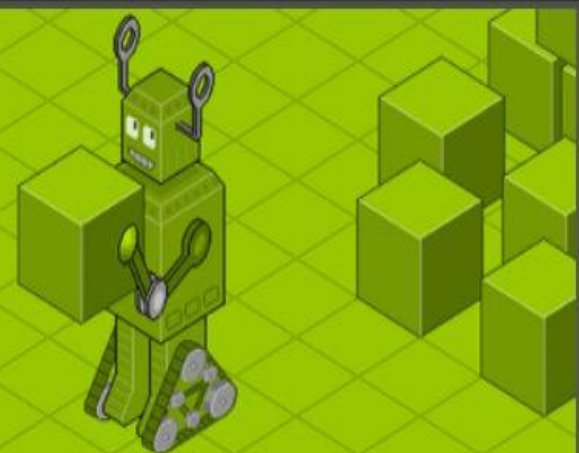


**DOCUMENTATION**



# ***SWAGGER***

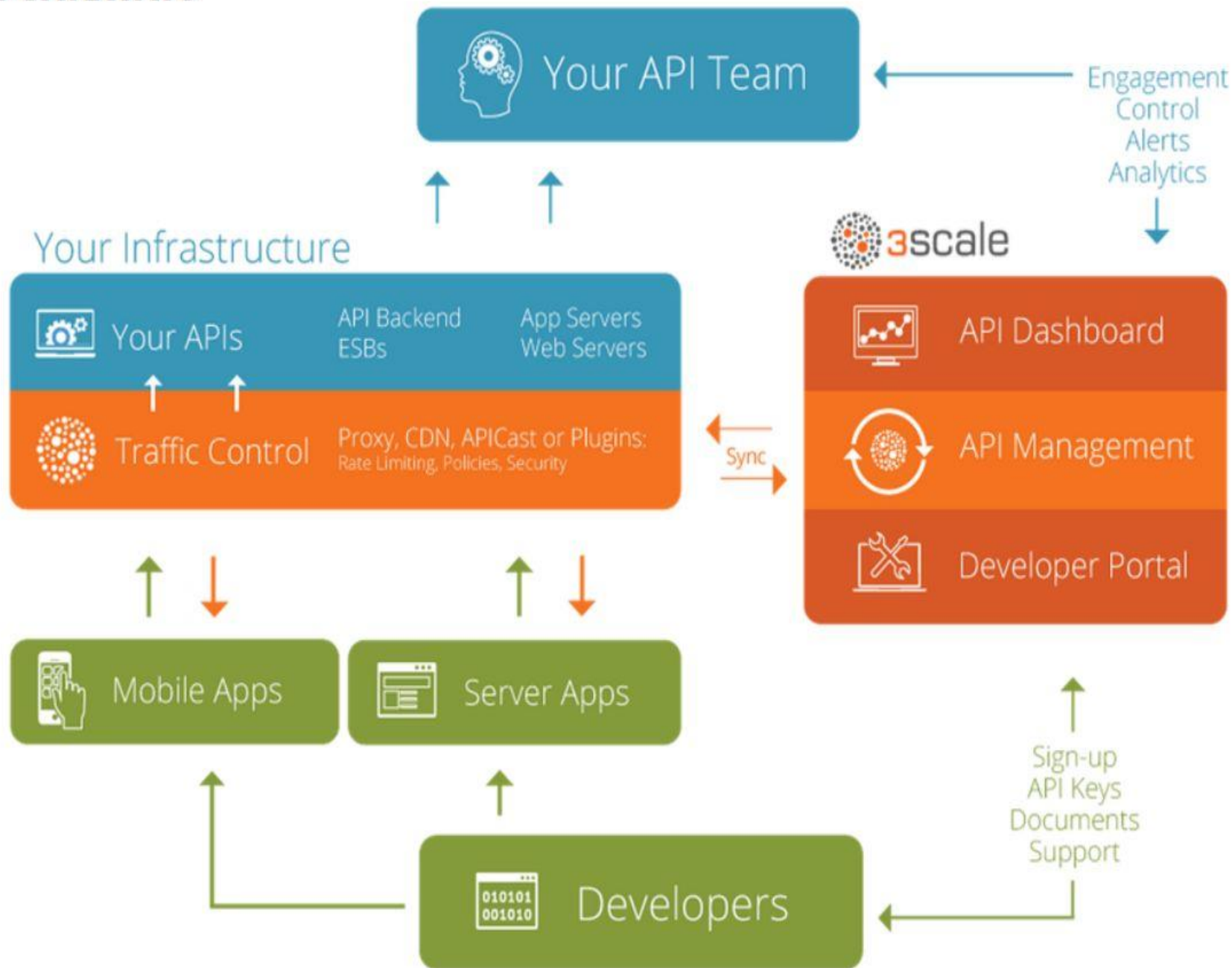
The World's Most Popular Framework for APIs.



## A POWERFUL INTERFACE TO YOUR API

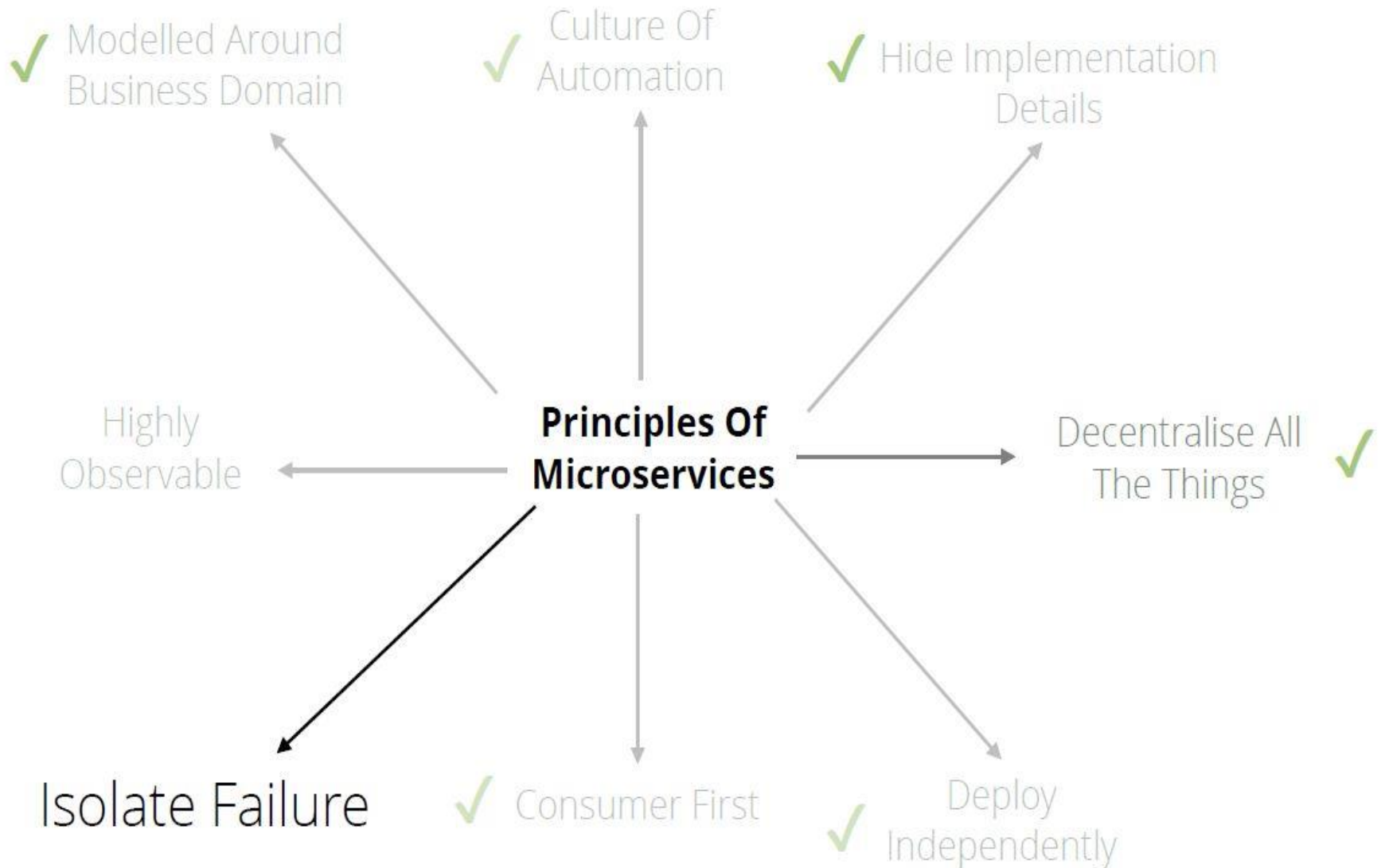
Swagger is a simple yet powerful representation of your RESTful API. With the largest ecosystem of API tooling on the planet, thousands of developers are supporting Swagger in almost every modern programming language and deployment environment. With a Swagger-enabled API, you get interactive documentation, client SDK generation and discoverability.

# API GATEWAYS



## SERVICE DISCOVERY







**David Brady**

@dbrady



Follow

If 1 service dies and your whole system breaks, you don't have SOA. You have a monolith whose brain has been chopped up and stuck in jars.



RETWEETS

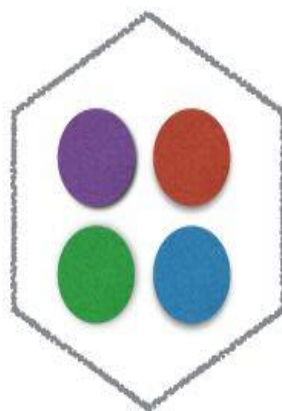
856

FAVORITES

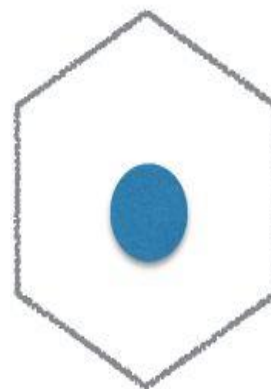
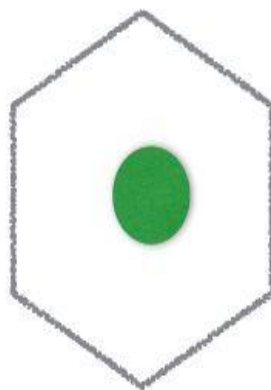
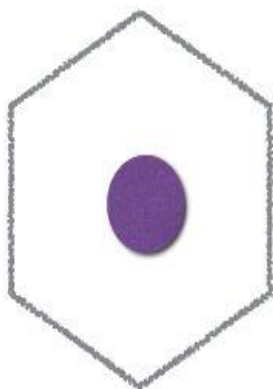
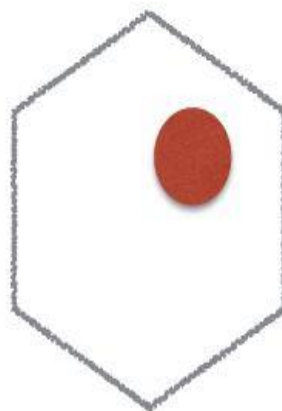
456

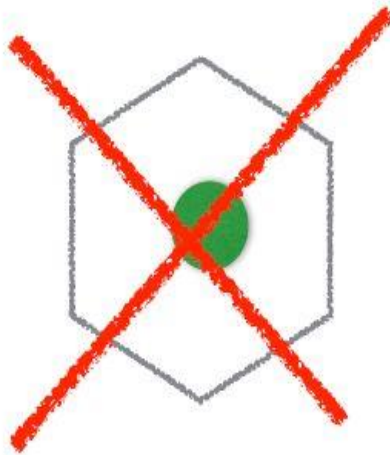
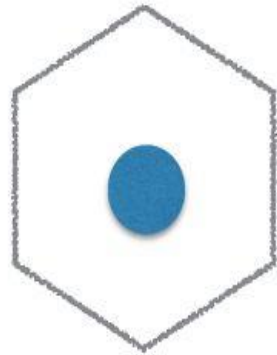
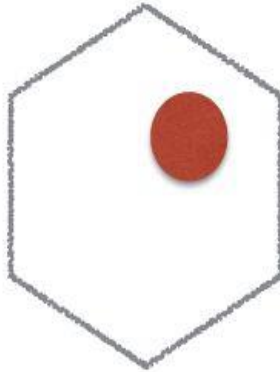
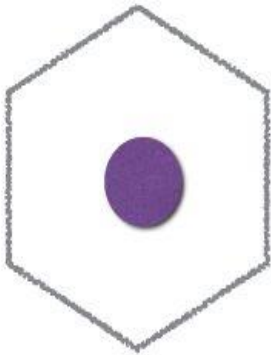


4:15 PM - 26 Mar 2015

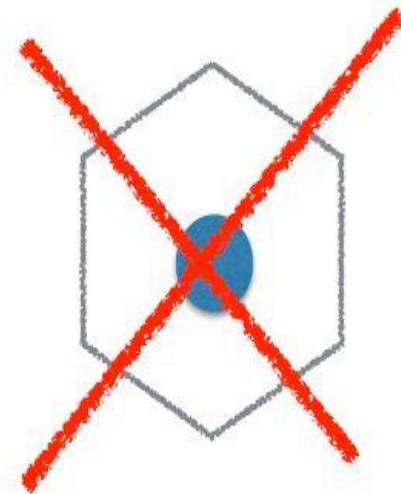
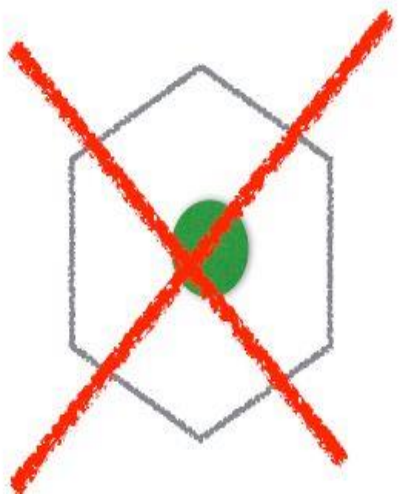
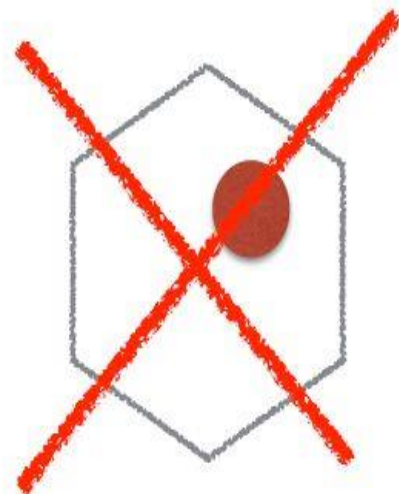
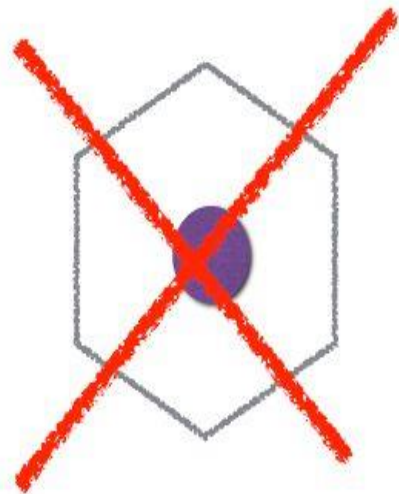




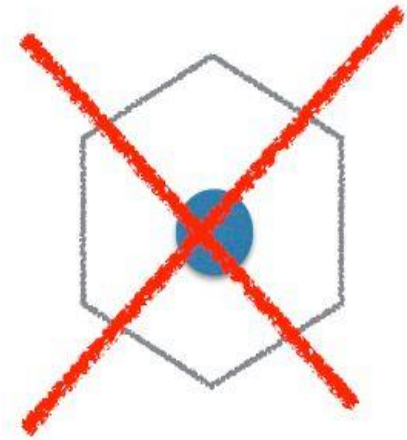
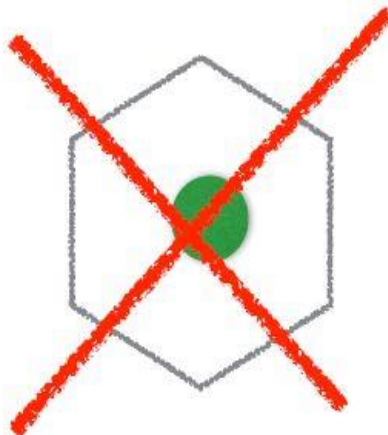
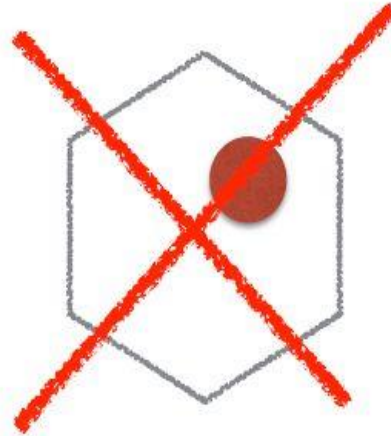
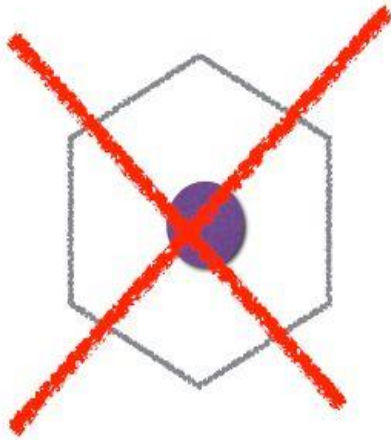








## AVOID THE DISTRIBUTED SINGLE POINT OF FAILURE!



Requests  
Building Up

Strangler App

*Thread Pool*

Thread-pool  
exhausted

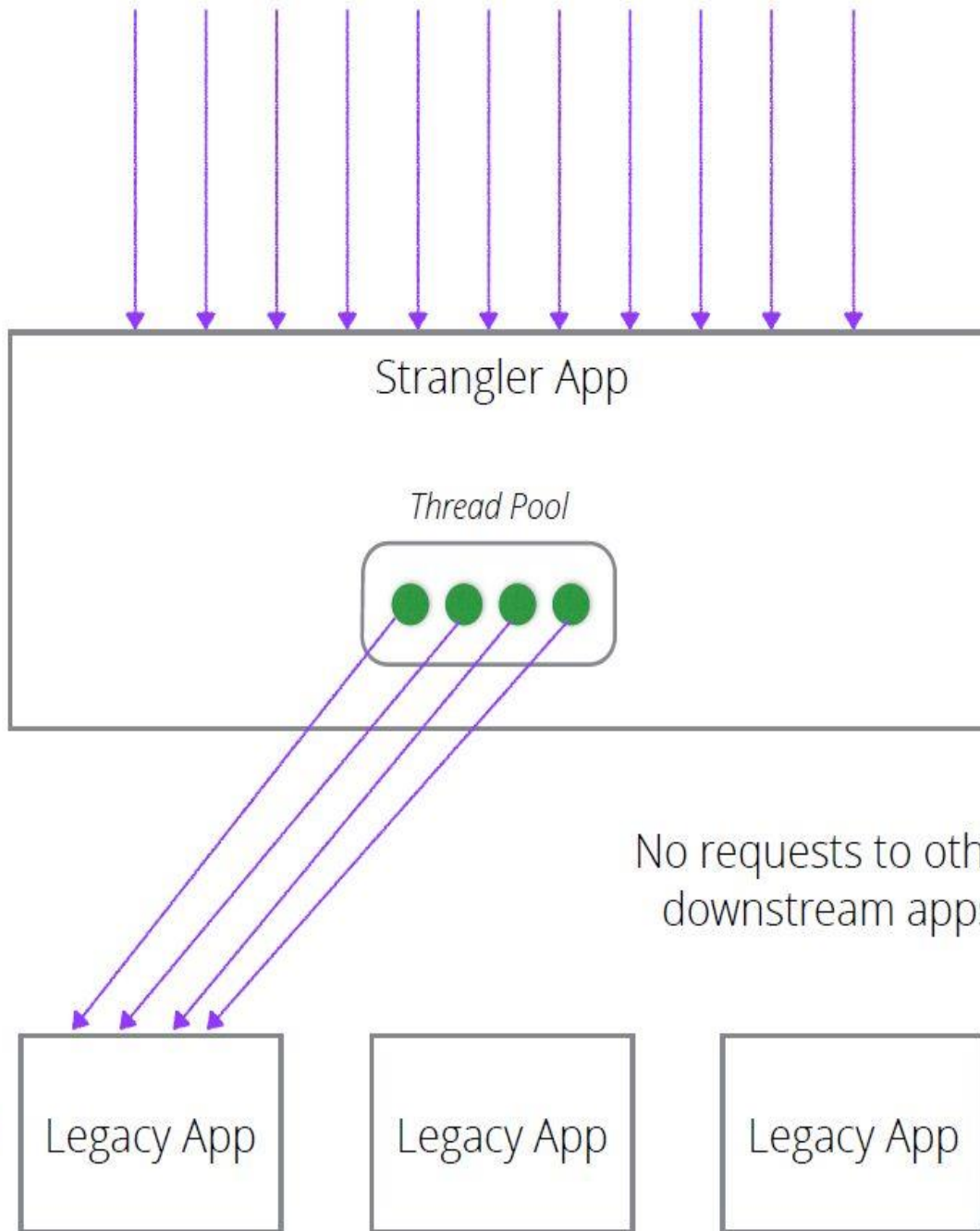
No requests to other  
downstream apps

Failing...slowly!

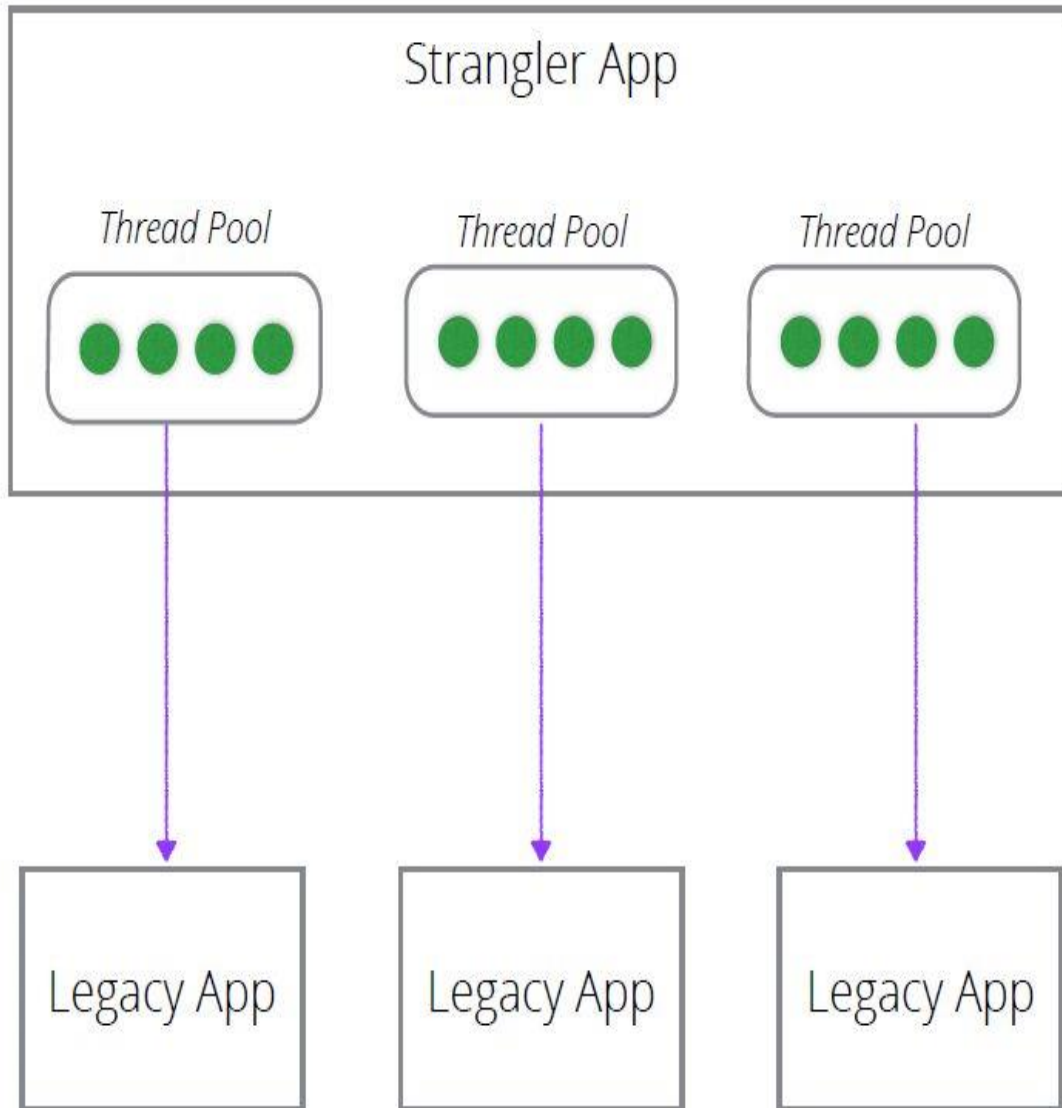
Legacy App

Legacy App

Legacy App

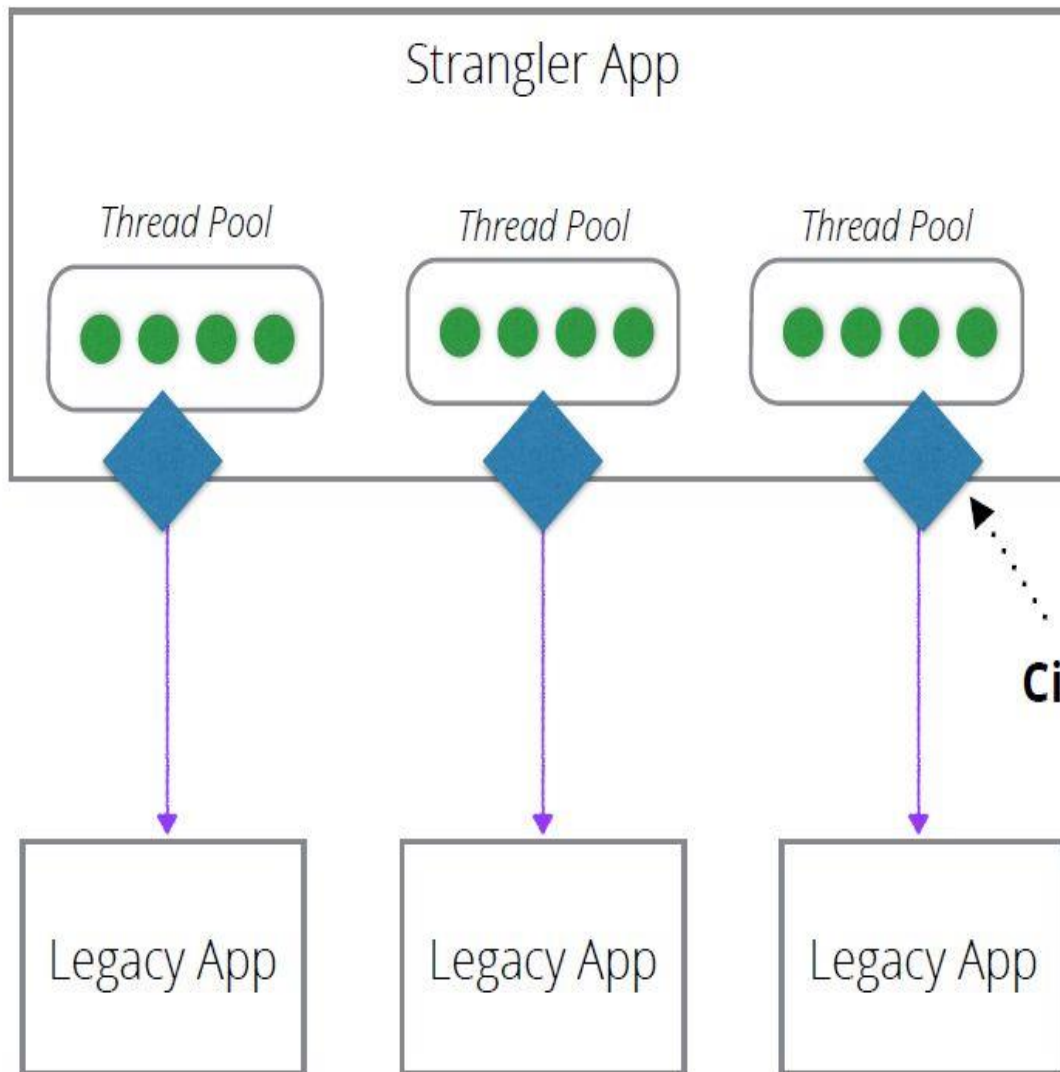


Fix **Timeouts**



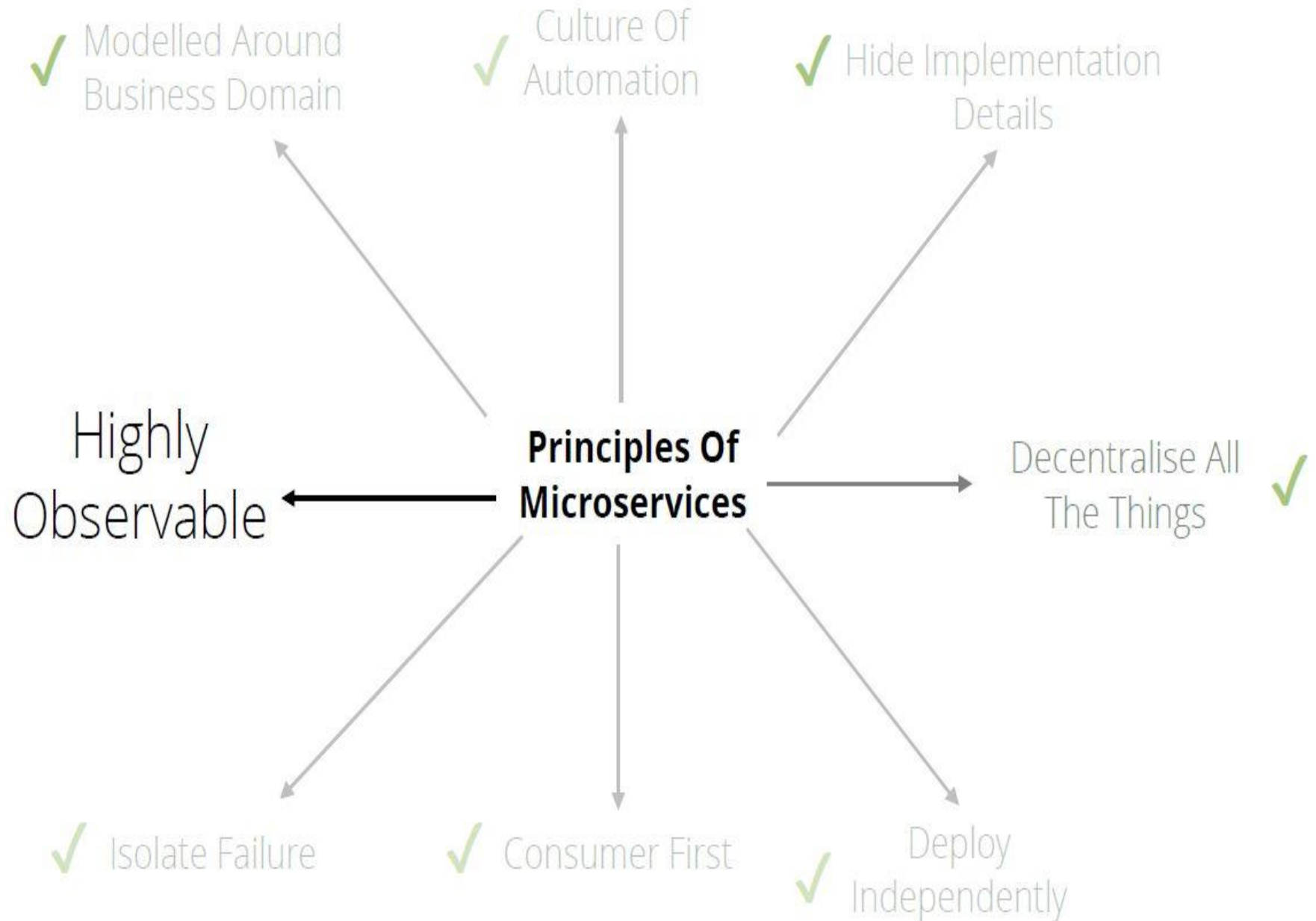
**Bulkhead**  
Downstream  
Connections

Fix **Timeouts**

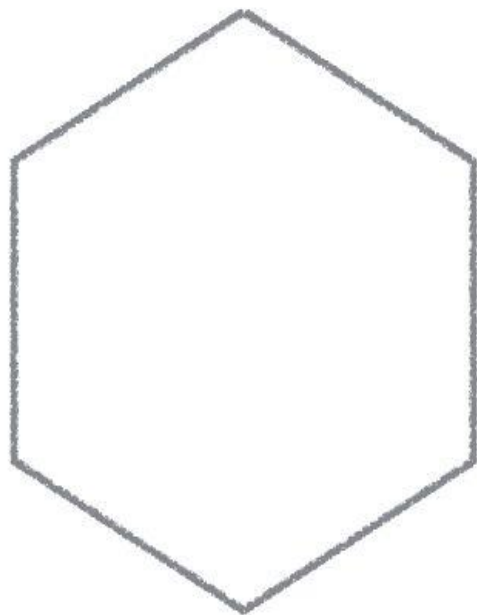


**Bulkhead**  
Downstream  
Connections

**Circuit Breakers**



# STATS PAGES



**numberOfApplicationErrors**

57

**numberOfServicedRequestsWithResponse200**

136711

**numberOfServicedRequestsWithResponse304**

27782

**numberOfServicedRequestsWithResponse404**

303

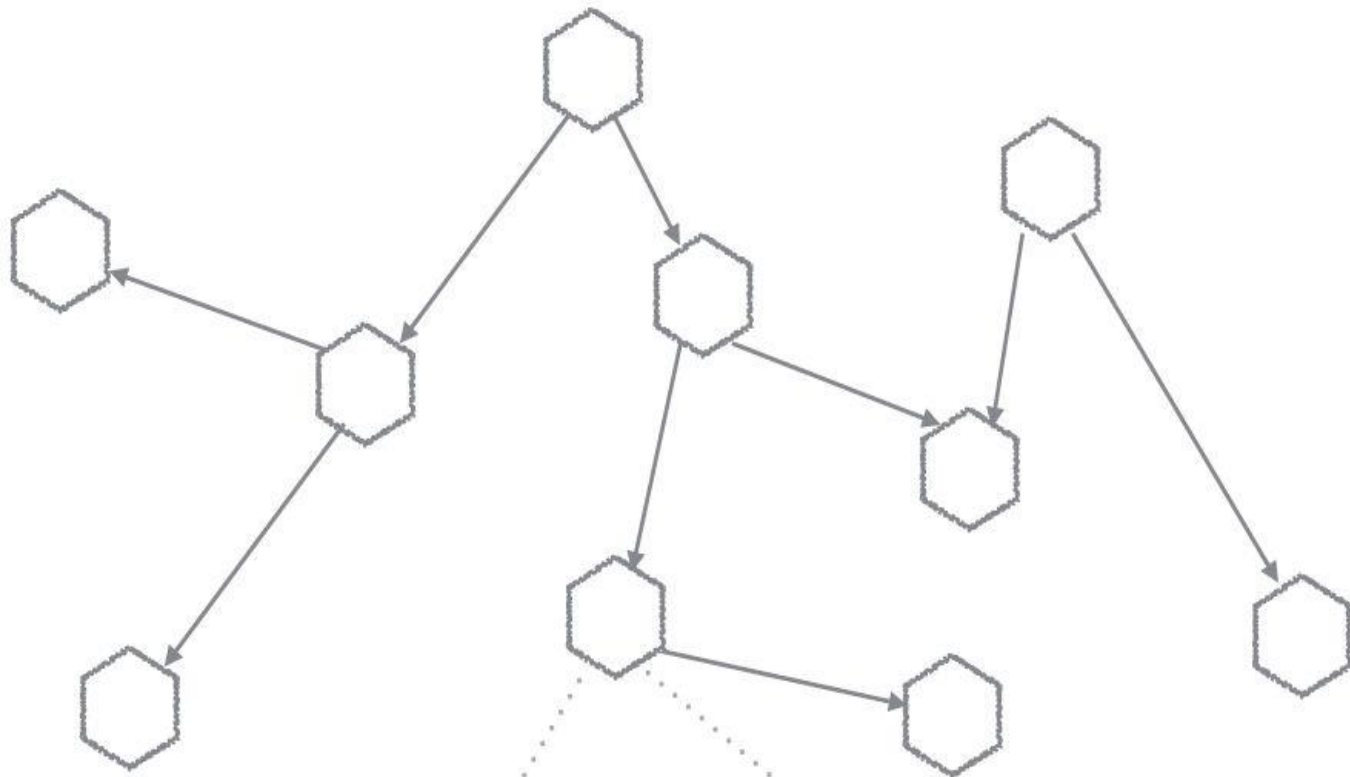
**numberOfServicedRequestsWithResponse500**

141

**totalNumberOfServicedRequests**

172383

# AGGREGATION



**LOGS**



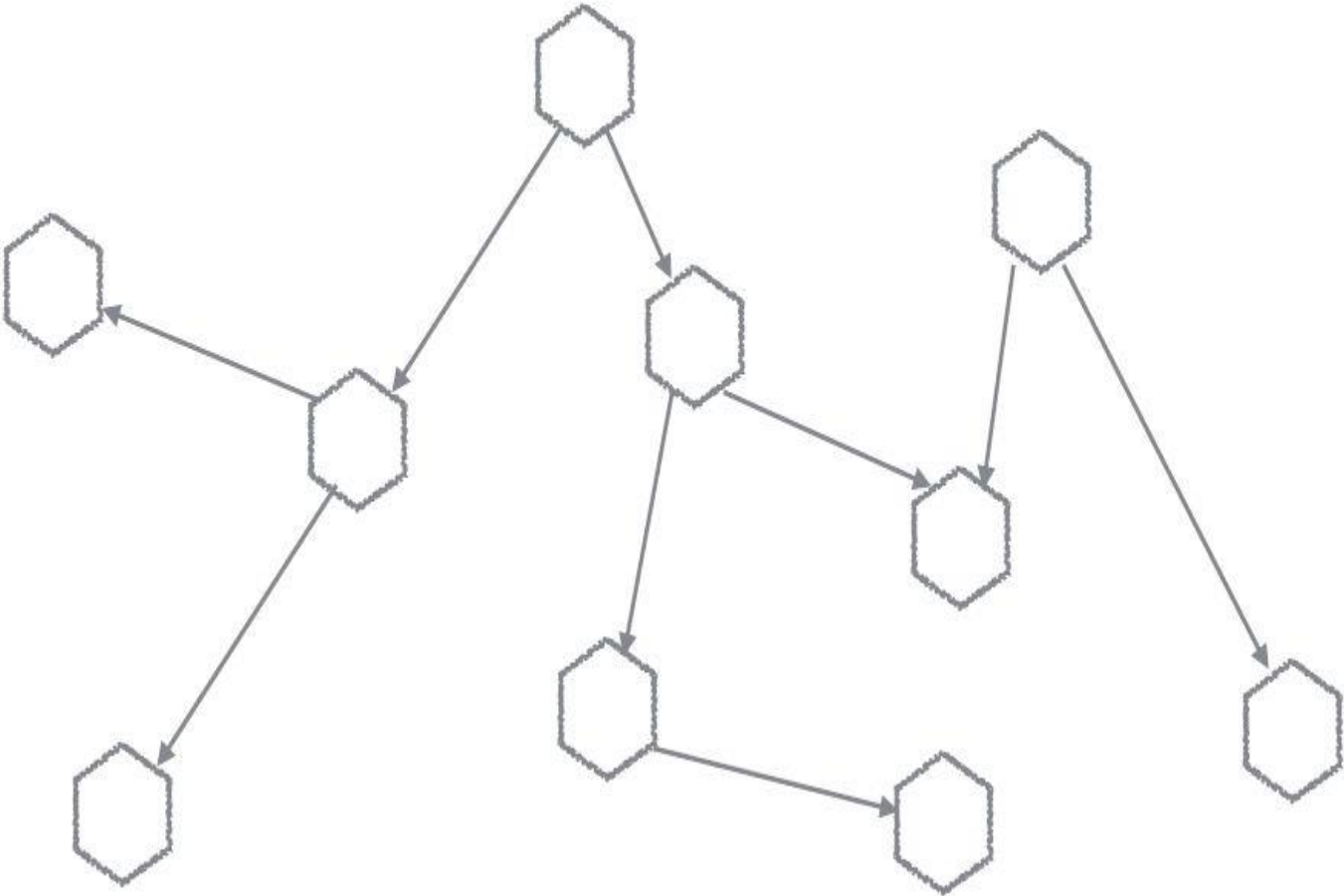
**STATS**





**CORRELATION IDS**

*ID 8964*



## CORRELATION IDS

