T7
DevOps/Continuous Delivery
Thursday, May 3rd, 2018
11:15 AM

Testing in a Microservices and Continuous Delivery Environment

Presented by:

Robert Williams

CA Technologies

Brought to you by:

Robert Williams

CA Technologies

Robert Williams has been in the software development business for twenty years, in fields ranging from semiconductor manufacturing automation and reporting systems to mobile security solutions to the market's leading service virtualization product. He has experience building, testing, and deploying software in multiple scenarios, whether it's an infrequently deployed internal system, a commercial product installed by customers on-premise, or multi-tenant hosted solutions. Robert has been a developer, manager, ScrumMaster, architect, and agile trainer/coach, but for the past decade he’s been keenly interested in tools and techniques that improve organizations’ ability to smoothly turn ideas into functioning, deployed software.

Testing in a Microservices and Continuous Delivery Environment

Robert Williams

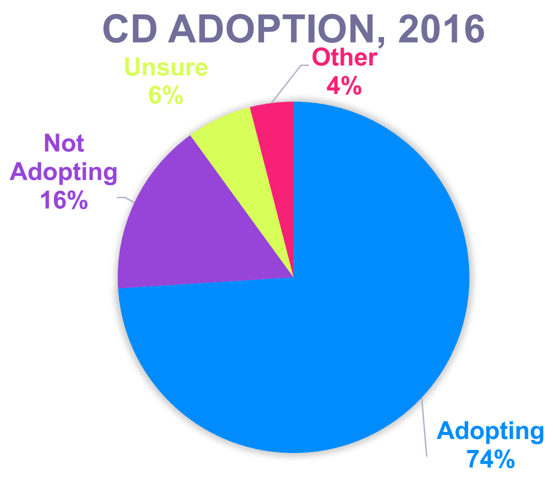
Continuous Delivery

Continuous delivery is the ability to get changes of all types—including new features, configuration changes, bug fixes and experiments—into production, or into the hands of users, safely and quickly in a sustainable way.

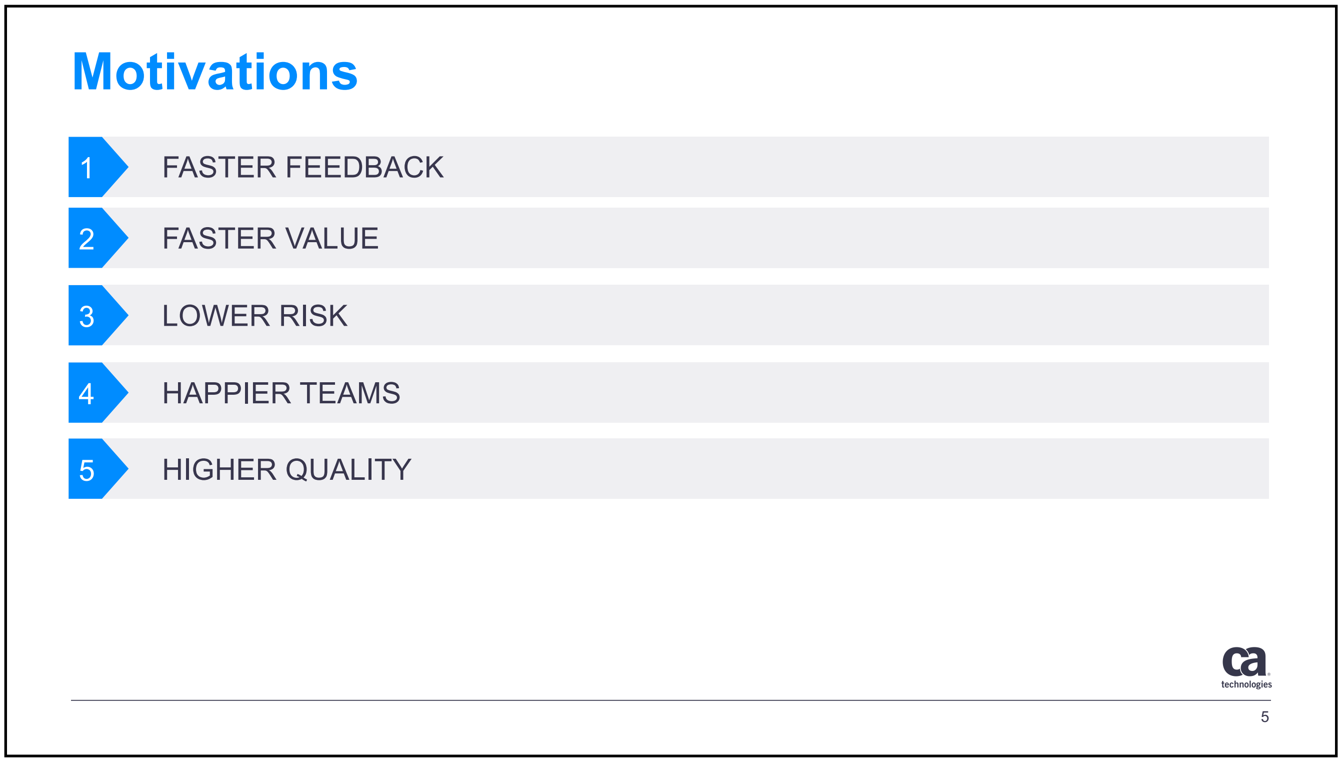
3

The New Normal

CD has “crossed the chasm” and is widely adopted - but not yet for all projects



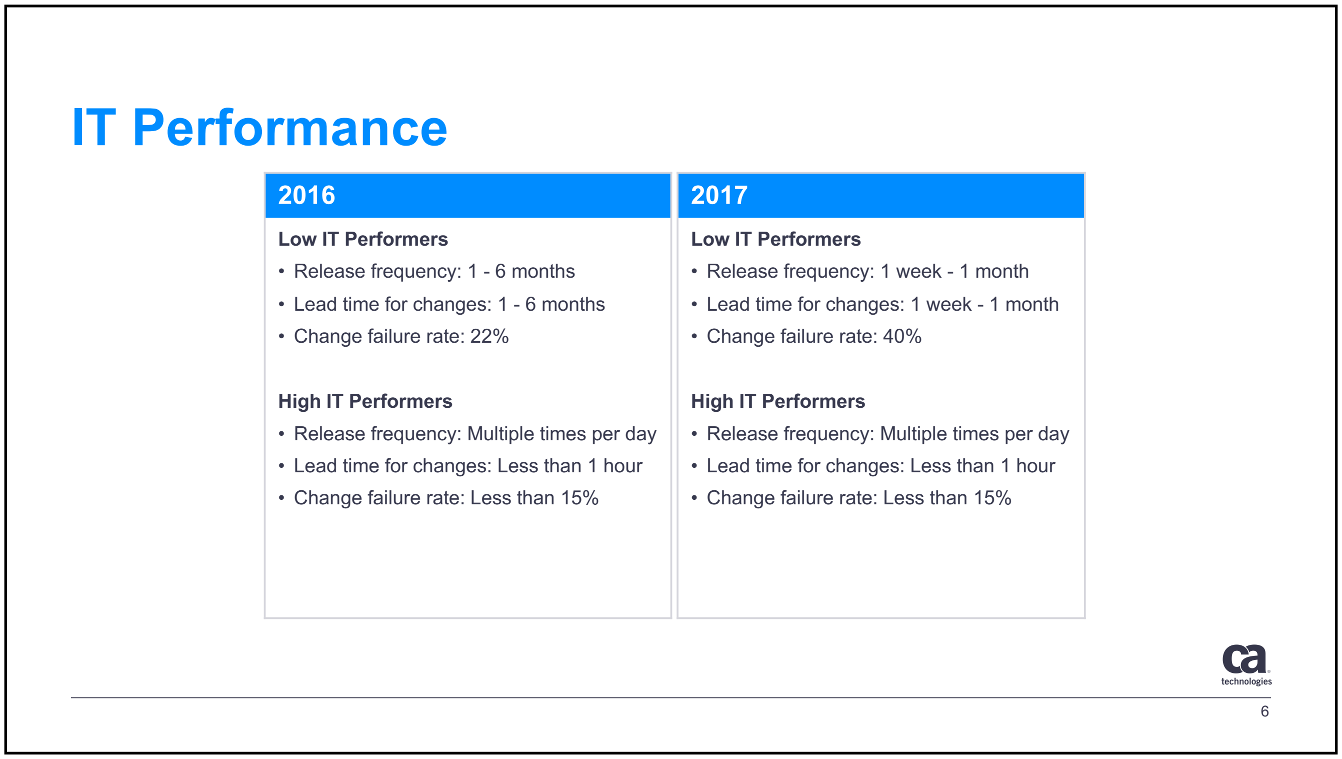
4



Motivations

1. FASTER FEEDBACK
2. FASTER VALUE
3. LOWER RISK
4. HAPPIER TEAMS
5. HIGHER QUALITY

5



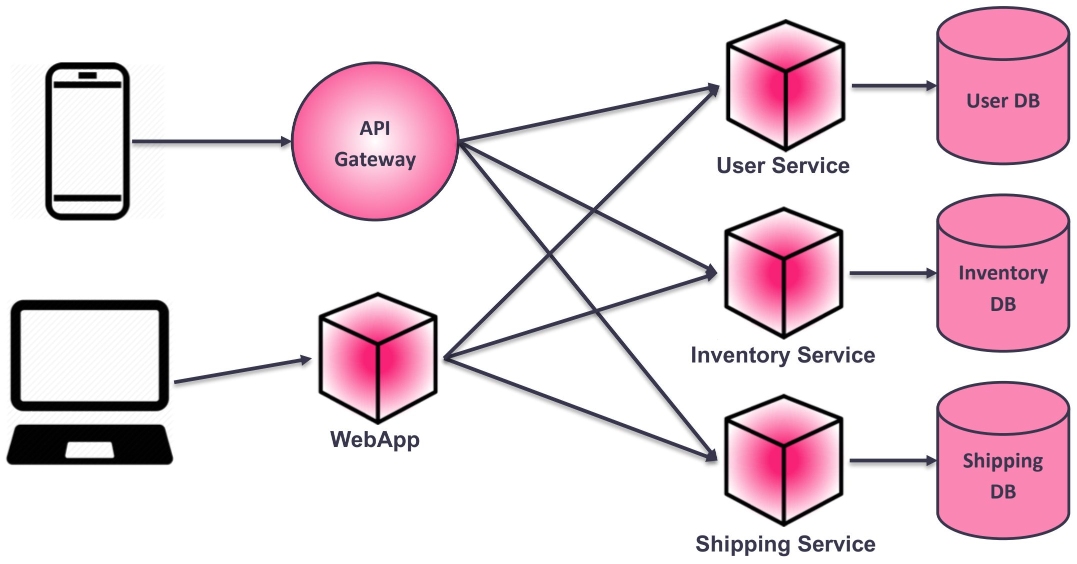
IT Performance

2016	2017
Low IT Performers	**Low IT Performers**
• Release frequency: 1 - 6 months	• Release frequency: 1 week - 1 month
• Lead time for changes: 1 - 6 months	• Lead time for changes: 1 week - 1 month
• Change failure rate: 22%	• Change failure rate: 40%
High IT Performers	**High IT Performers**
• Release frequency: Multiple times per day	• Release frequency: Multiple times per day
• Lead time for changes: Less than 1 hour	• Lead time for changes: Less than 1 hour
• Change failure rate: Less than 15%	• Change failure rate: Less than 15%

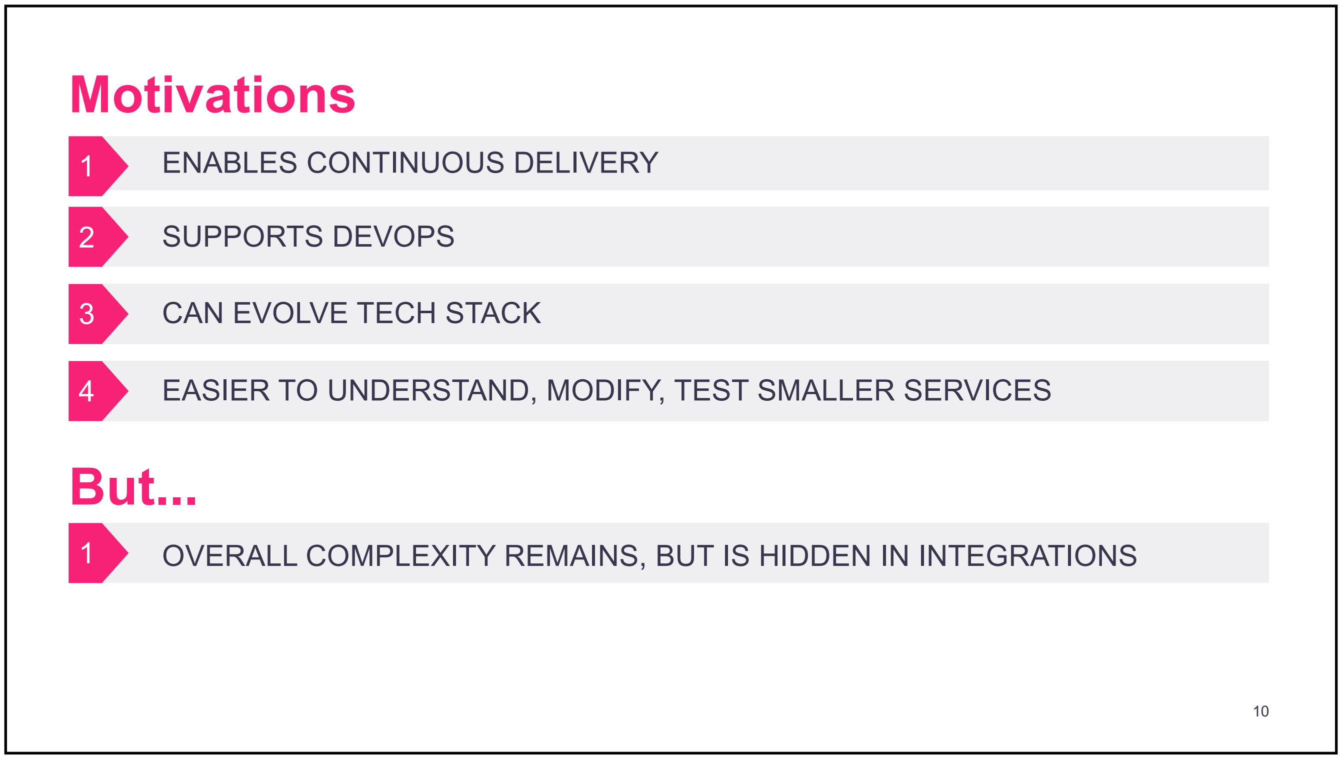
6

Microservices

Microservice-based architecture is an architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities



9



Motivations

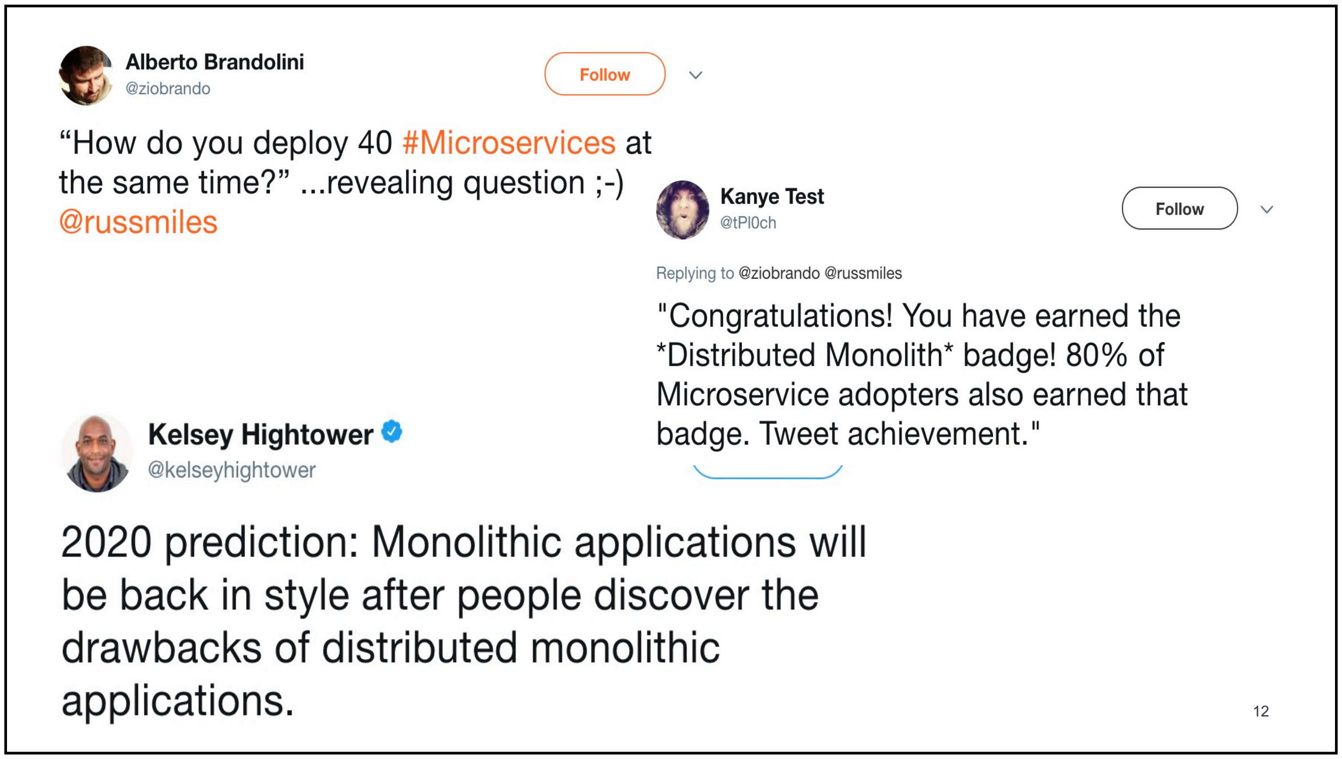
1. ENABLES CONTINUOUS DELIVERY
2. SUPPORTS DEVOPS
3. CAN EVOLVE TECH STACK
4. EASIER TO UNDERSTAND, MODIFY, TEST SMALLER SERVICES

But...

1. OVERALL COMPLEXITY REMAINS, BUT IS HIDDEN IN INTEGRATIONS

10

How it Fails



Alberto Brandolini
@ziobrando

“How do you deploy 40 #Microservices at the same time?” ...revealing question ;-) @russmiles

Kanye Test
@tPl0ch

Replying to @ziobrando @russmiles

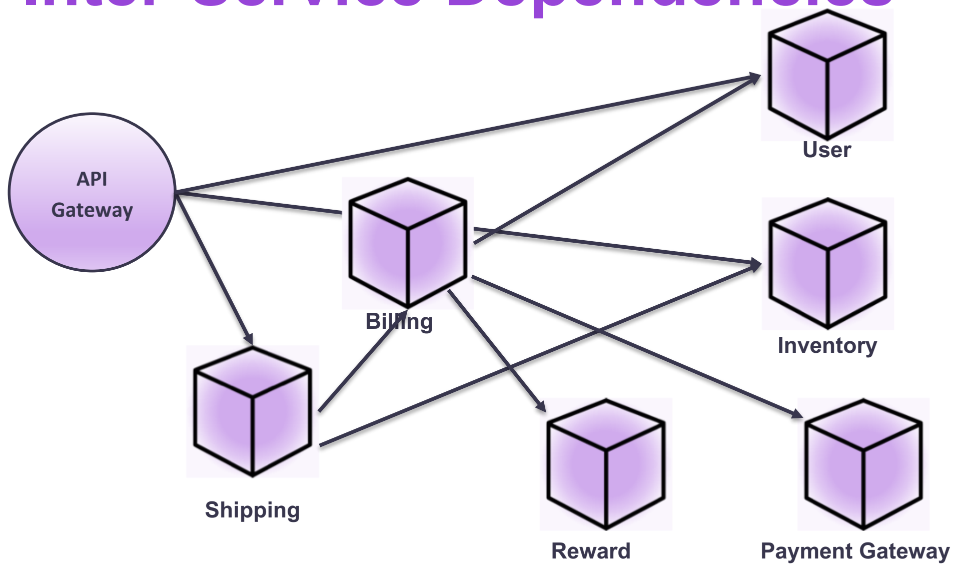
"Congratulations! You have earned the *Distributed Monolith* badge! 80% of Microservice adopters also earned that badge. Tweet achievement."

Kelsey Hightower
@kelseyhightower

2020 prediction: Monolithic applications will be back in style after people discover the drawbacks of distributed monolithic applications.

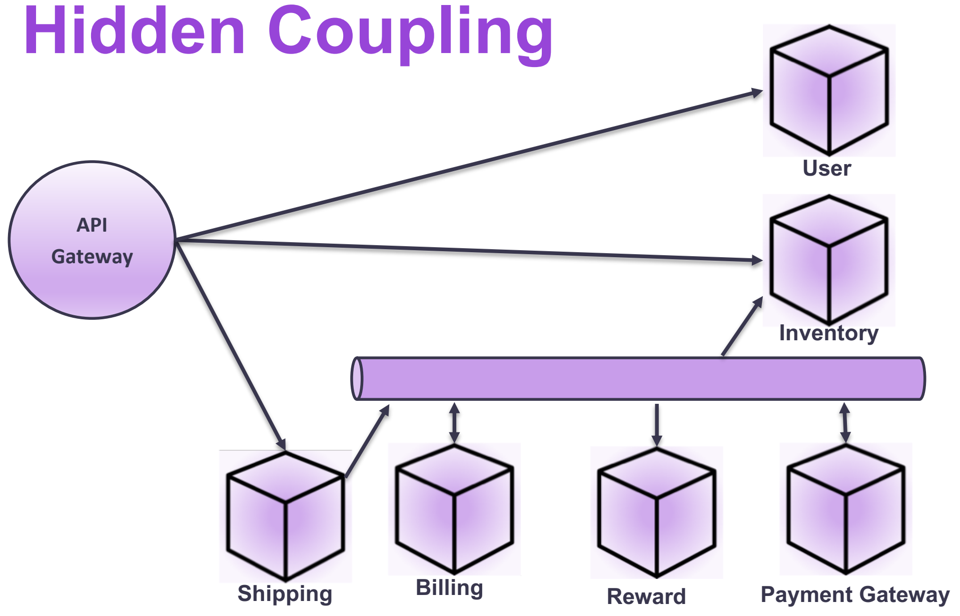
12

Inter-Service Dependencies



13

Hidden Coupling



14

Continuous Testing

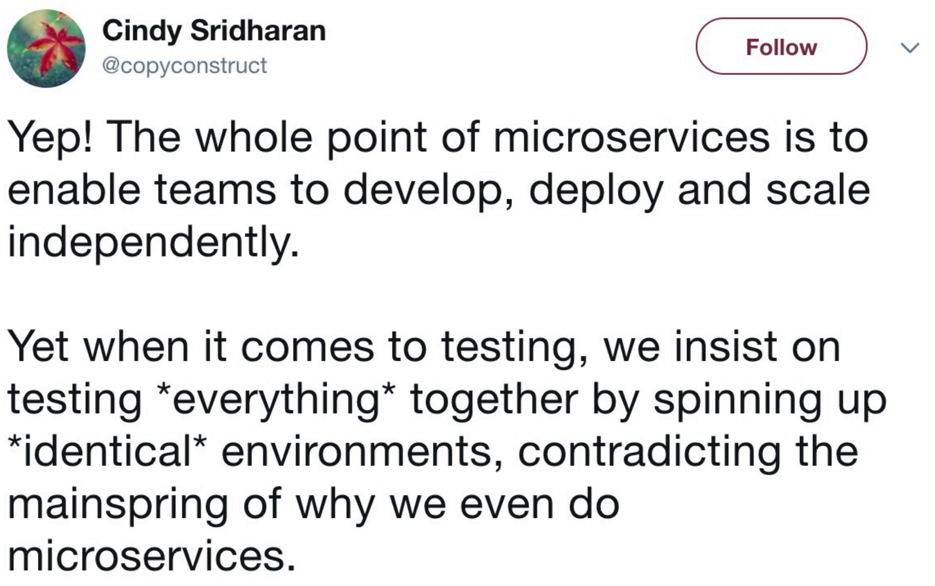
Testing your system at the appropriate level, measuring appropriate characteristics, in the appropriate context, at every step in the SDLC

16

Level	Characteristics	Context
Unit / API	Does the service behave as designed? API compatibility?	All dependencies virtualized
Integration	Basic functionality, selected negative cases, API interoperability	Transitive, expensive, unstable, or unavailable dependencies virtualized
Functional	Overall system behavior	Only unstable / unavailable dependencies virtualized
Performance	Performance characteristics – speed, memory, disk, network, latency, degradation	Expensive, unstable / unavailable, or artificially slow dependencies virtualized

17

But…


Cindy Sridharan
@copyconstruct

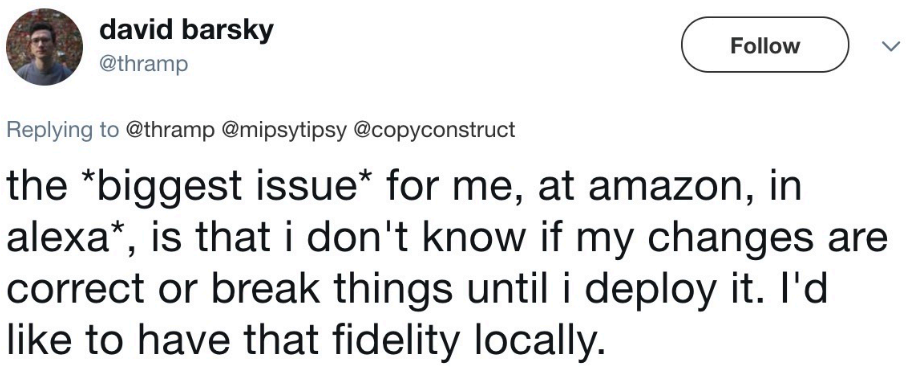
Follow

Yep! The whole point of microservices is to enable teams to develop, deploy and scale independently.

Yet when it comes to testing, we insist on testing *everything* together by spinning up *identical* environments, contradicting the mainspring of why we even do microservices.

18

But...

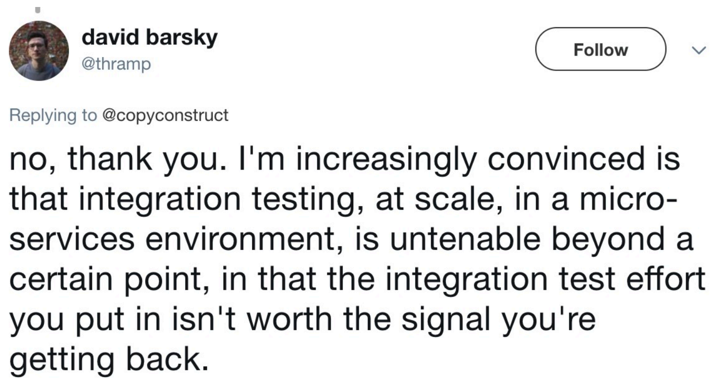

david barsky
@thramp

Replying to @thramp @mipsytipsy @copyconstruct

the *biggest issue* for me, at amazon, in alexa*, is that i don't know if my changes are correct or break things until i deploy it. I'd like to have that fidelity locally.

Options

Throw in the Towel?


david barsky
@thramp

Follow

Replying to @copyconstruct

no, thank you. I'm increasingly convinced is that integration testing, at scale, in a micro-services environment, is untenable beyond a certain point, in that the integration test effort you put in isn't worth the signal you're getting back.

Top performing IT organizations can deploy software to production in less than one hour, have failure rates of less than 15%, and can easily roll back their changes.

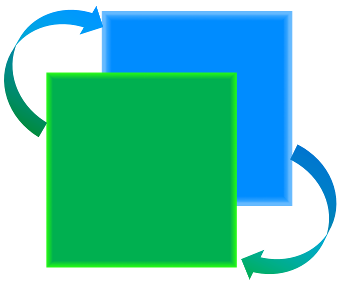
21

Throw in the Towel?

- Ensure good automated unit test coverage
- GUI testing
- Extensive API testing
- Resiliency, Scalability
- Usability / Functional Quality
- Automate, automate, automate
- Teach developers how to test
- Automated failure detection and correction

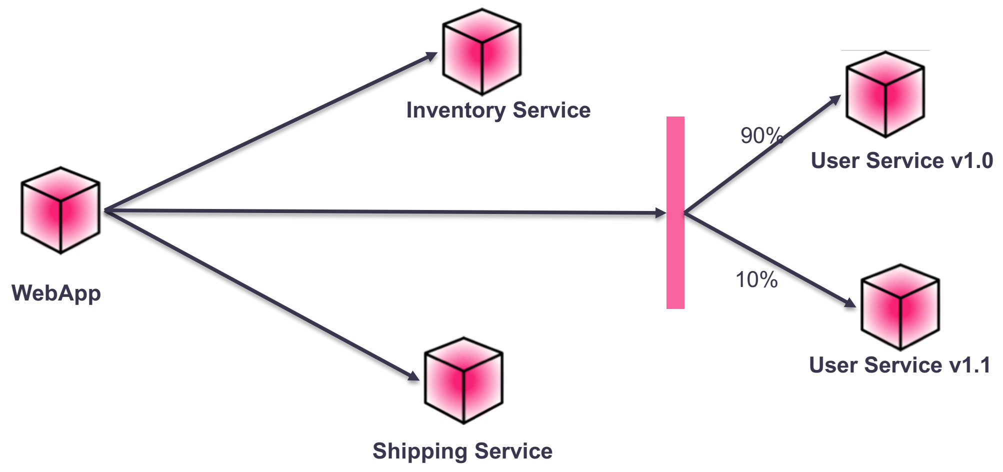
22

Testing In Production



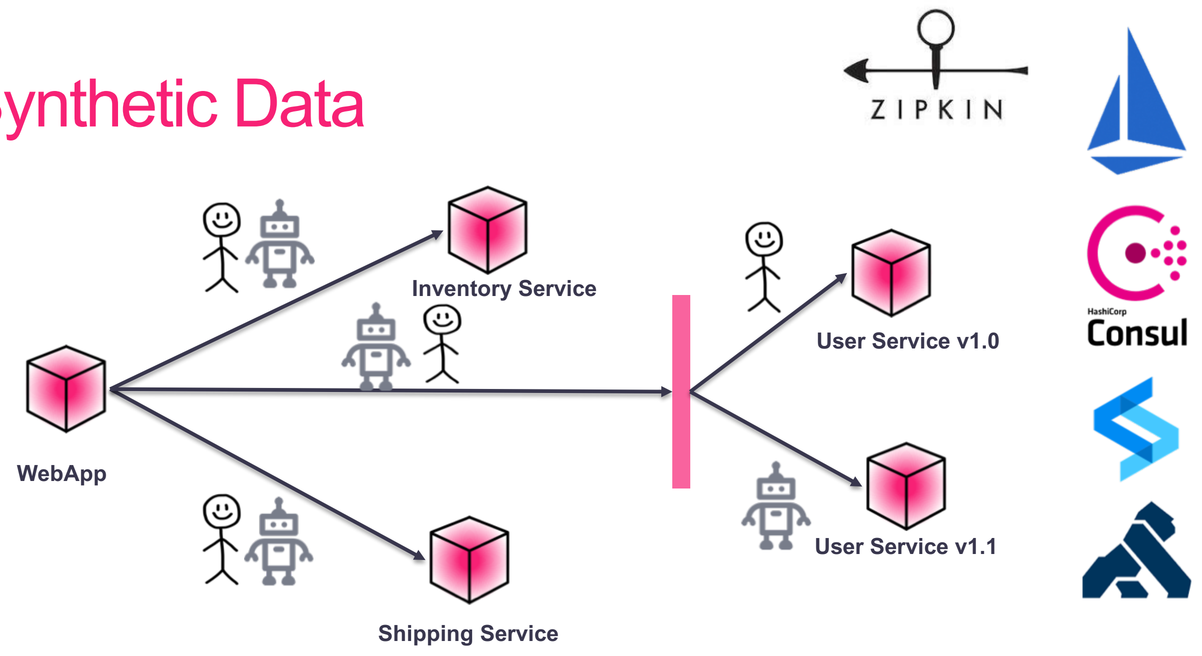
23

Canary Deployments



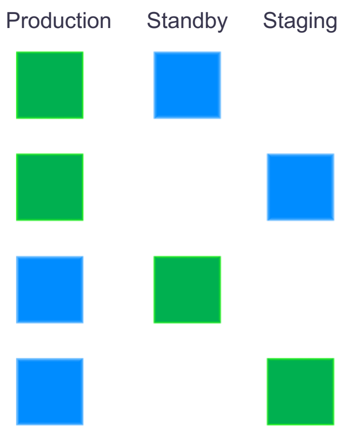
24

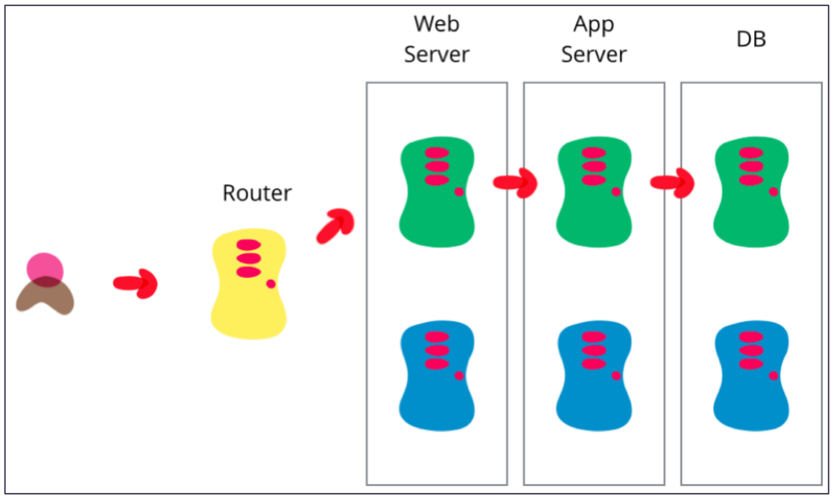
Synthetic Data



25

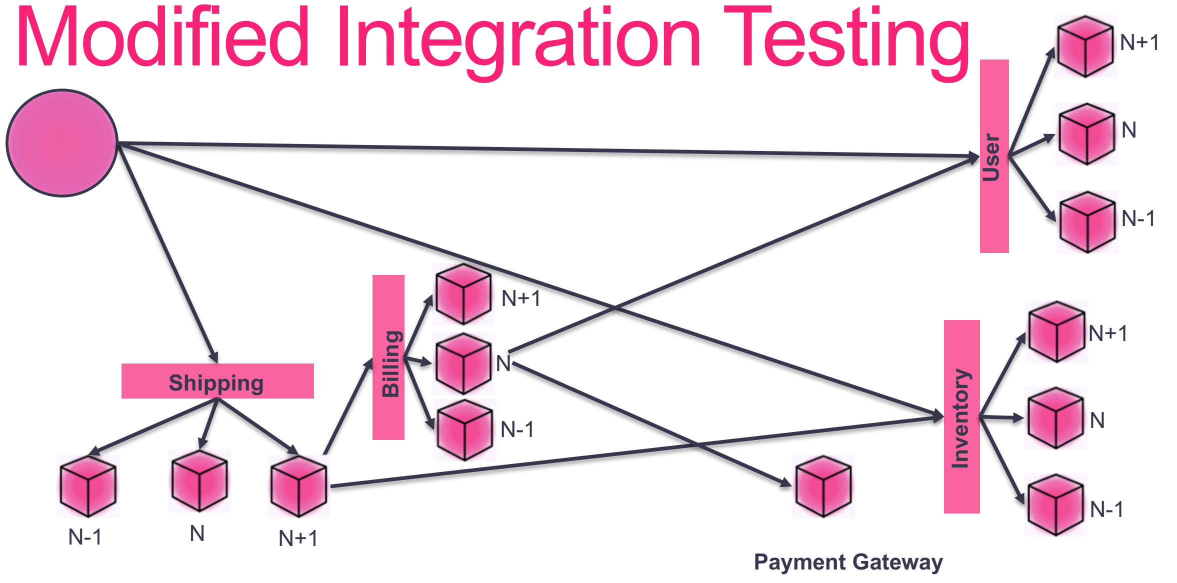
Blue / Green





26

Modified Integration Testing



27

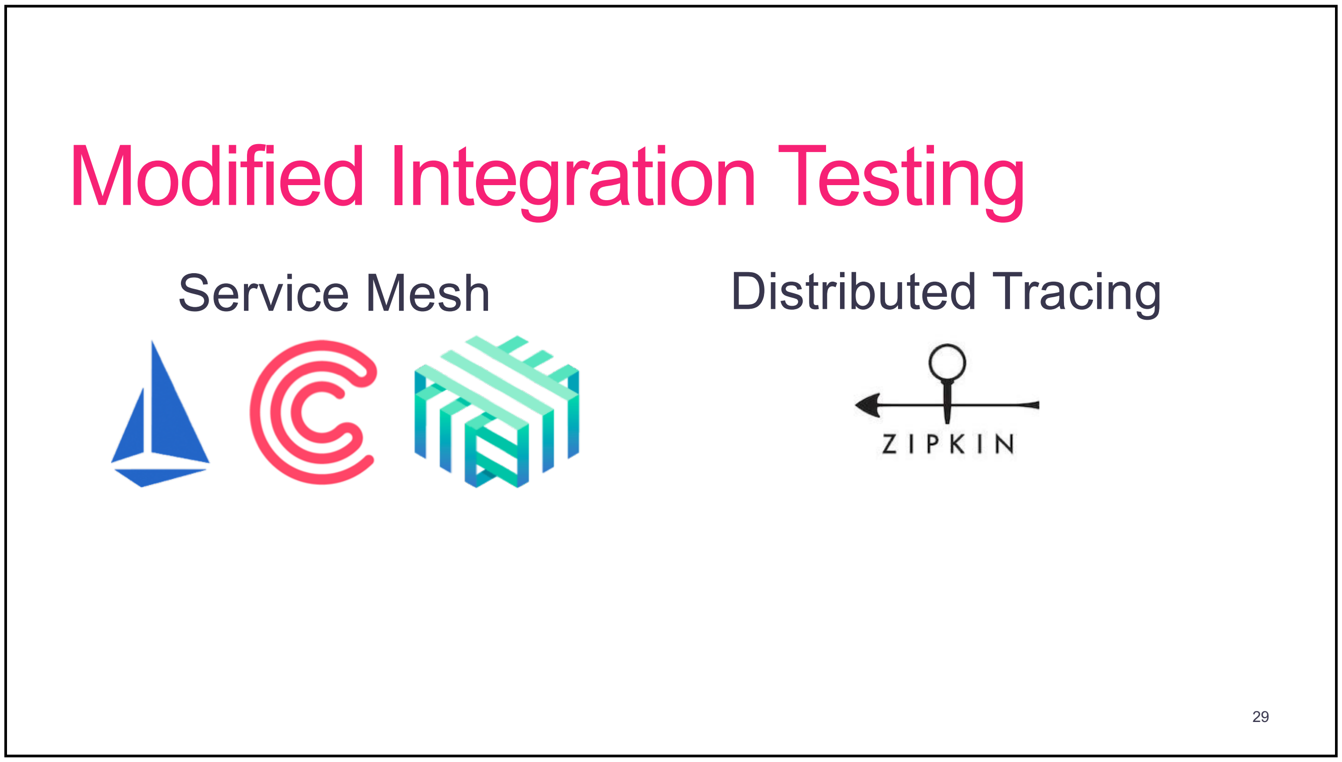
Modified Integration Testing

20 microservices
3 versions each
$3^{20} = 3.5 \times 10^9$ combinations

3^{20} combinations ÷ 1000 combinations / sec ÷ 86400 sec/day

40 days to exhaustively test all combinations

28



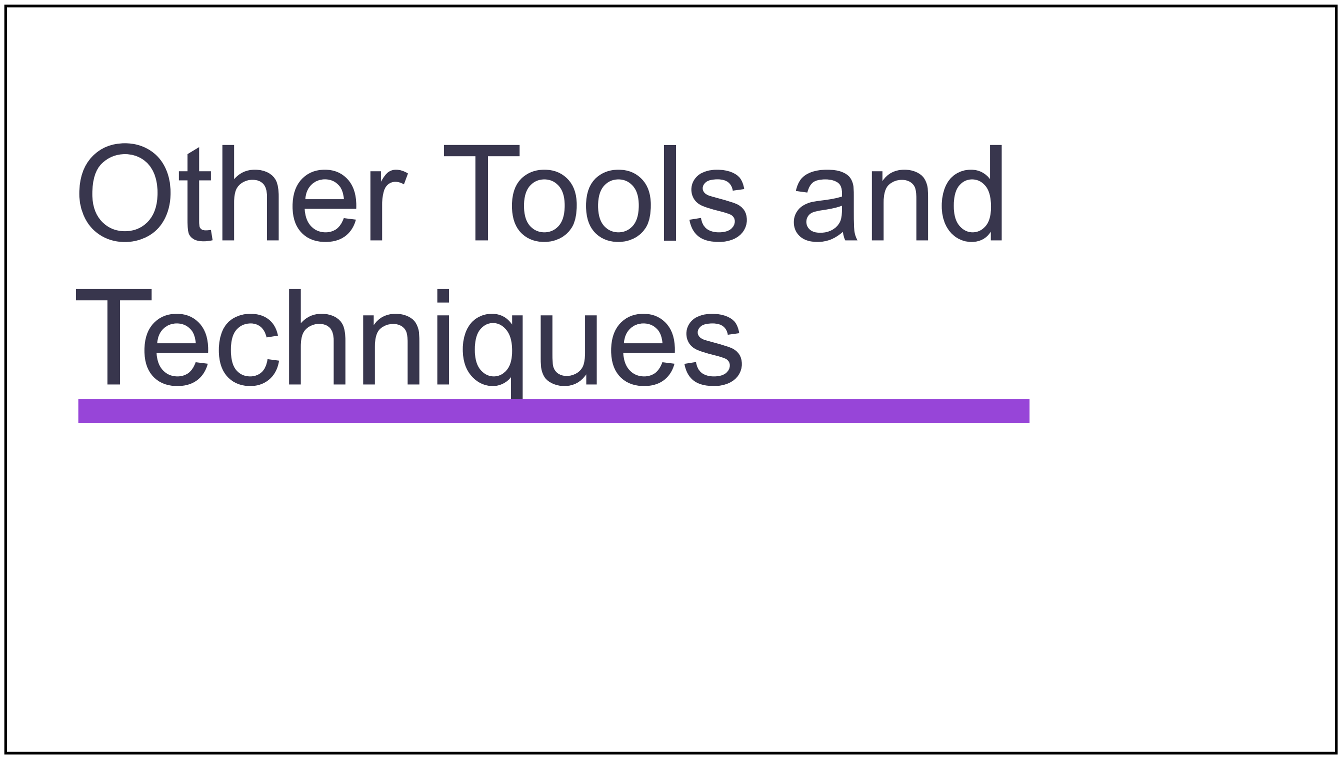
Modified Integration Testing

Service Mesh

Distributed Tracing

ZIPKIN

29



Other Tools and Techniques

Service Virtualization

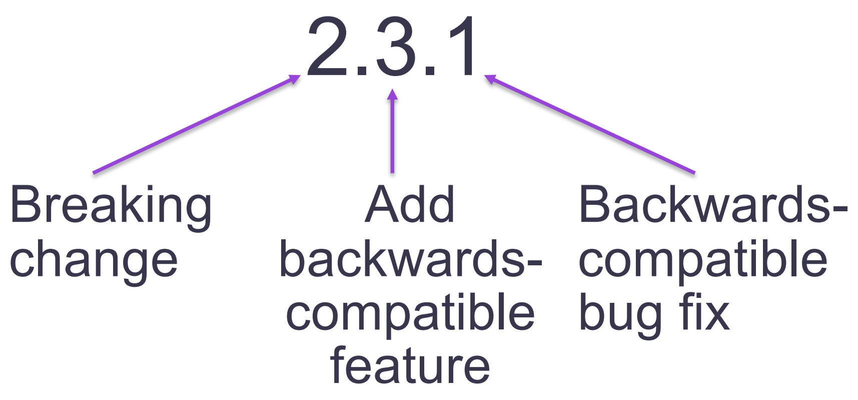
Can't always run multiple versions of a service simultaneously (e.g. database changes)

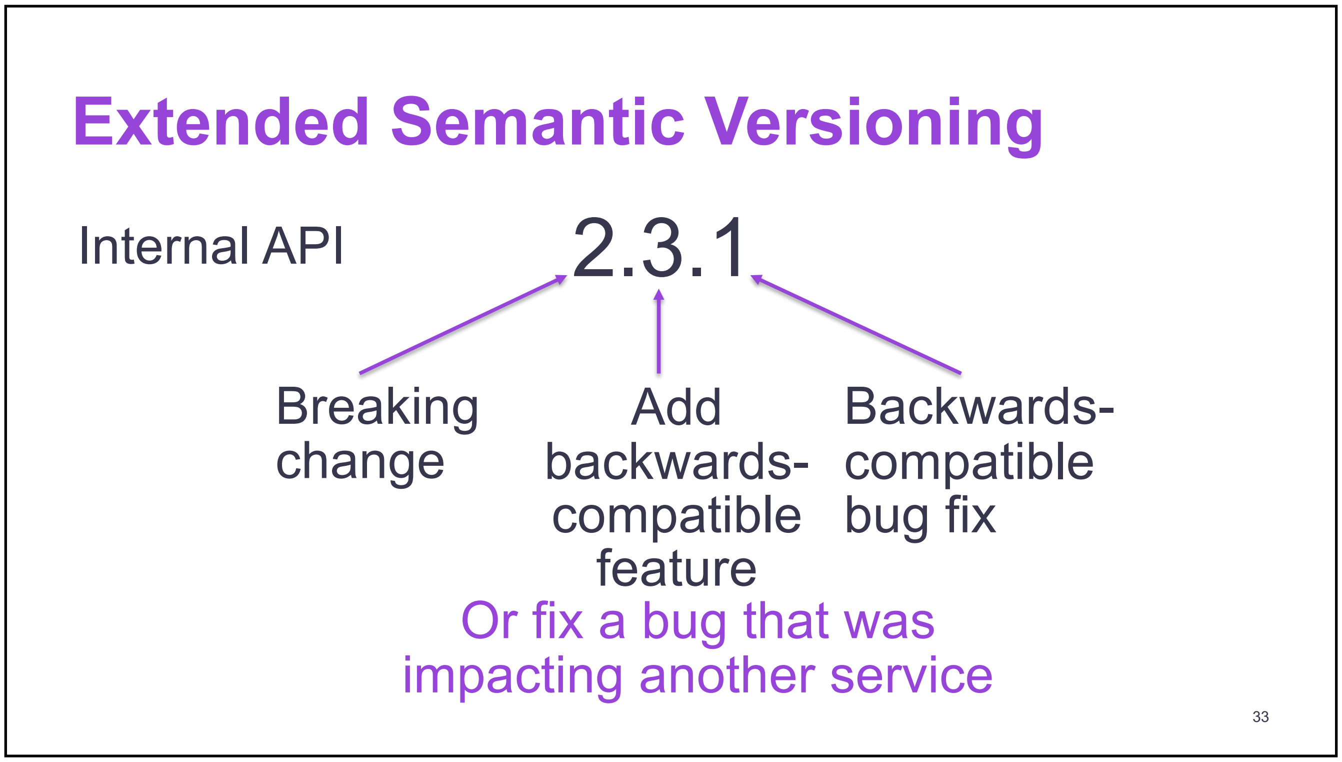
Can't generate all negative test cases using live code

Therefore... Create a virtual service for each version of your real service. Keep this in the same repo as your binary, and tag it the same way as your binary.

Extended Semantic Versioning

Public API


2.3.1
Breaking change
Add backwards-compatible feature
Backwards-compatible bug fix



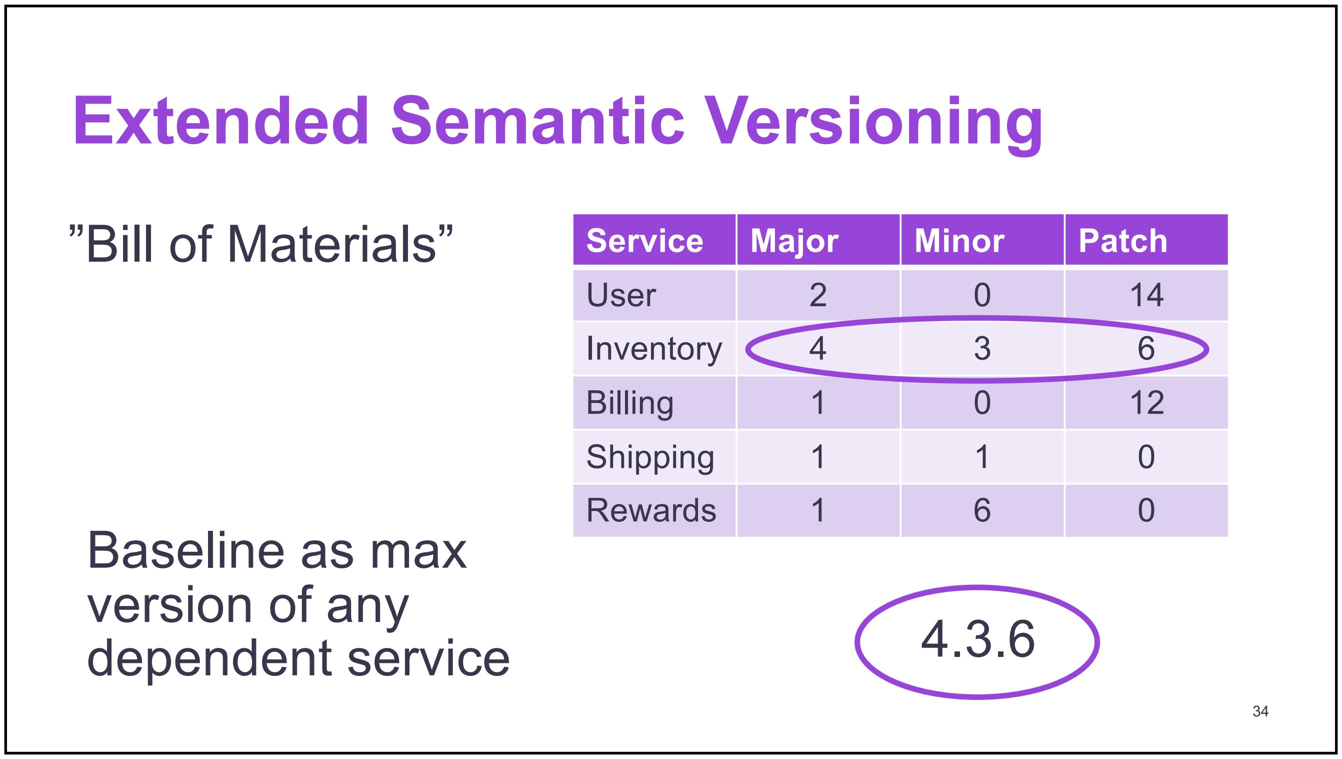
Extended Semantic Versioning

Internal API 2.3.1

Breaking change
Add backwards-compatible feature
Backwards-compatible bug fix

Or fix a bug that was impacting another service

33



Extended Semantic Versioning

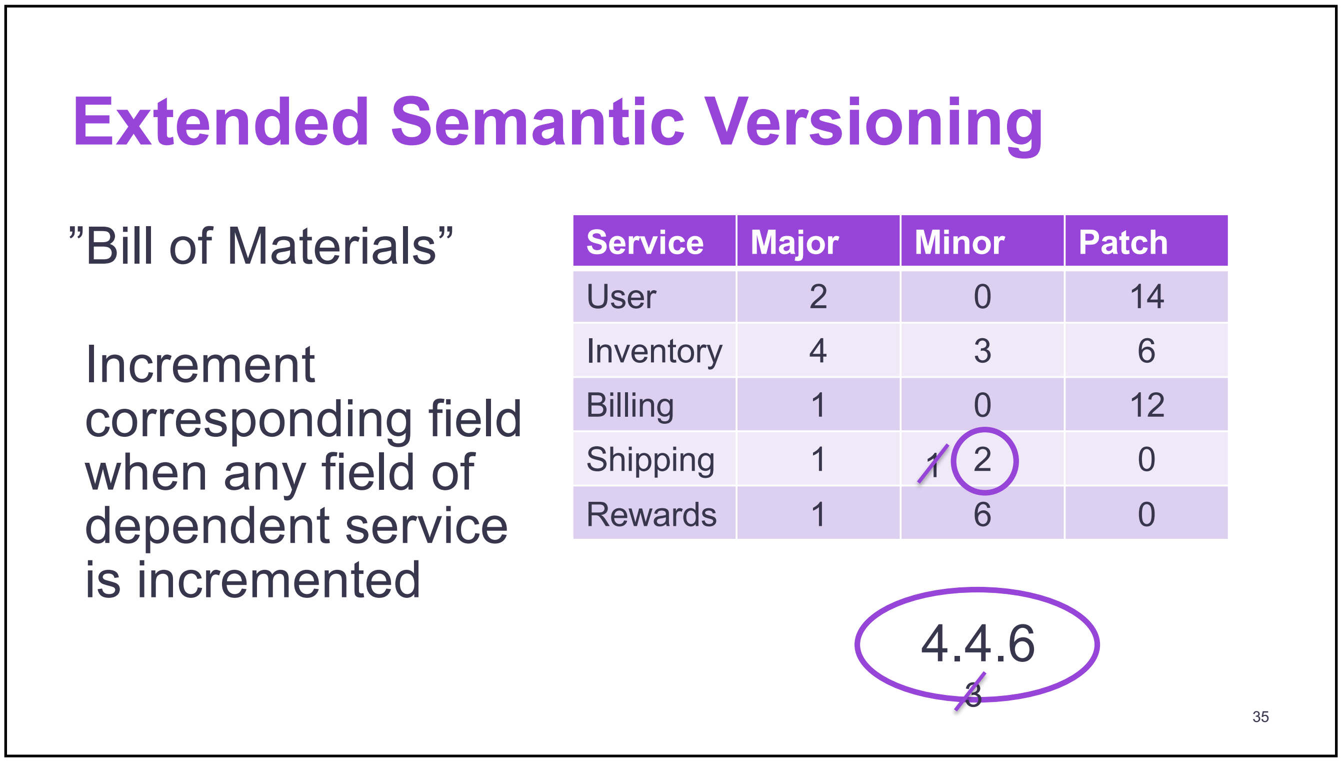
”Bill of Materials”

Service	Major	Minor	Patch
User	2	0	14
Inventory	4	3	6
Billing	1	0	12
Shipping	1	1	0
Rewards	1	6	0

Baseline as max version of any dependent service

4.3.6

34



Extended Semantic Versioning

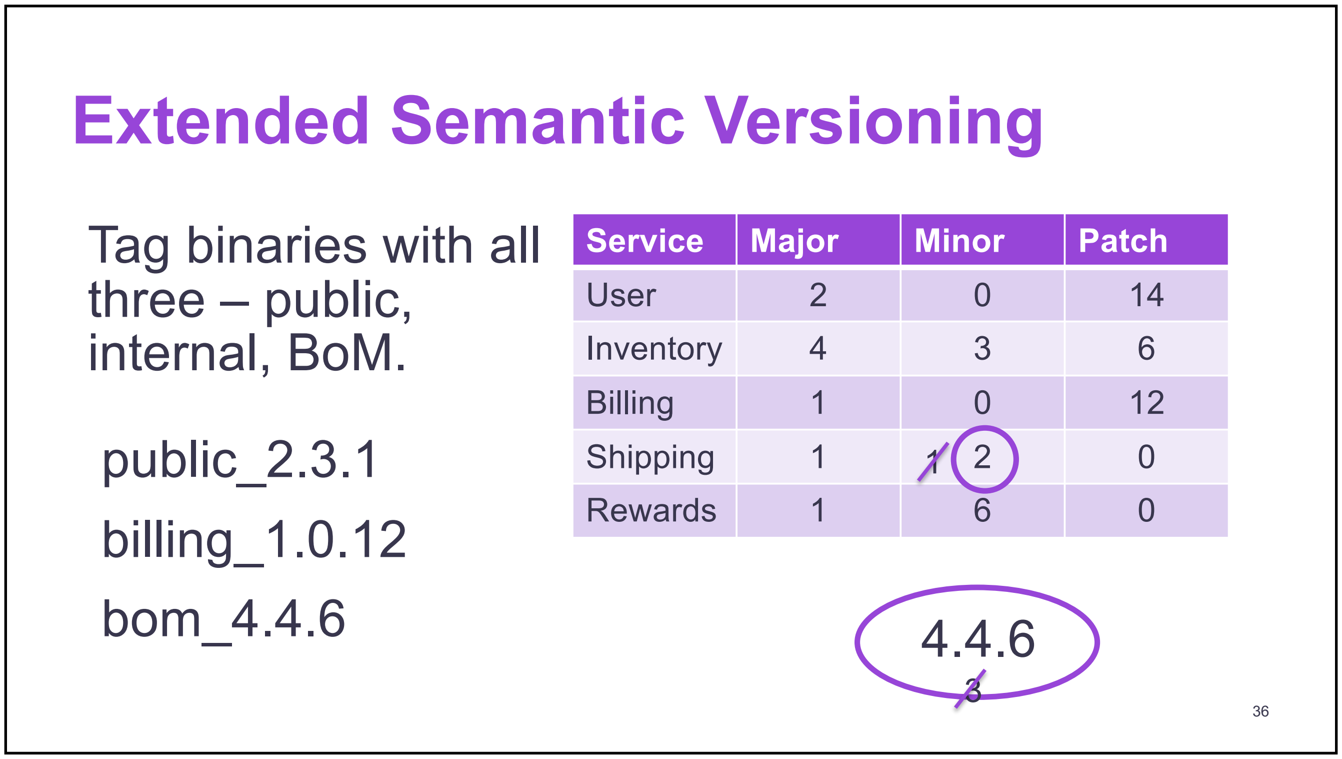
"Bill of Materials"

Increment corresponding field when any field of dependent service is incremented

Service	Major	Minor	Patch
User	2	0	14
Inventory	4	3	6
Billing	1	0	12
Shipping	1	~~1~~ 2	0
Rewards	1	6	0

4.4.6 ~~3~~

35



Extended Semantic Versioning

Tag binaries with all three – public, internal, BoM.

public_2.3.1

billing_1.0.12

bom_4.4.6

Service	Major	Minor	Patch
User	2	0	14
Inventory	4	3	6
Billing	1	0	12
Shipping	1	~~1~~ 2	0
Rewards	1	6	0

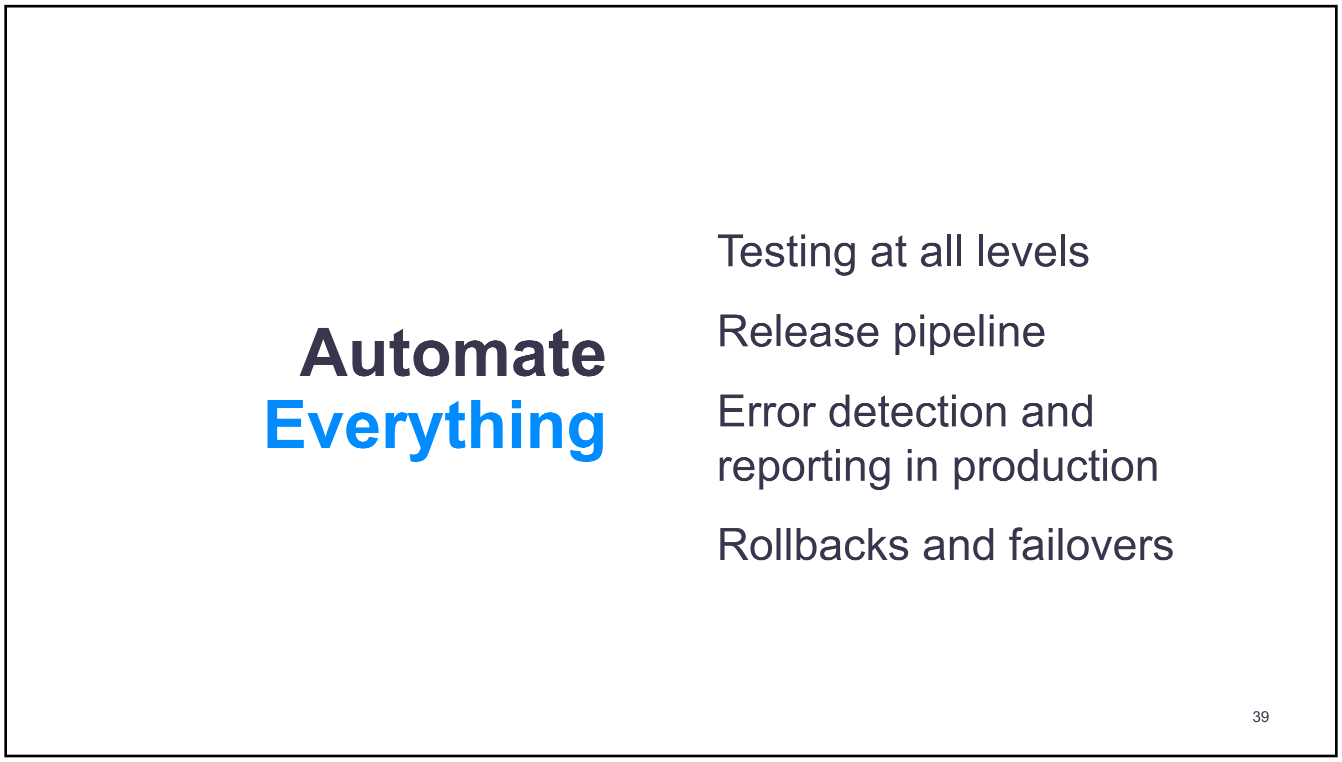
4.4.6 ~~3~~

36

Long Term Strategy

Support the Continuous Delivery Transformation

38



Automate Everything

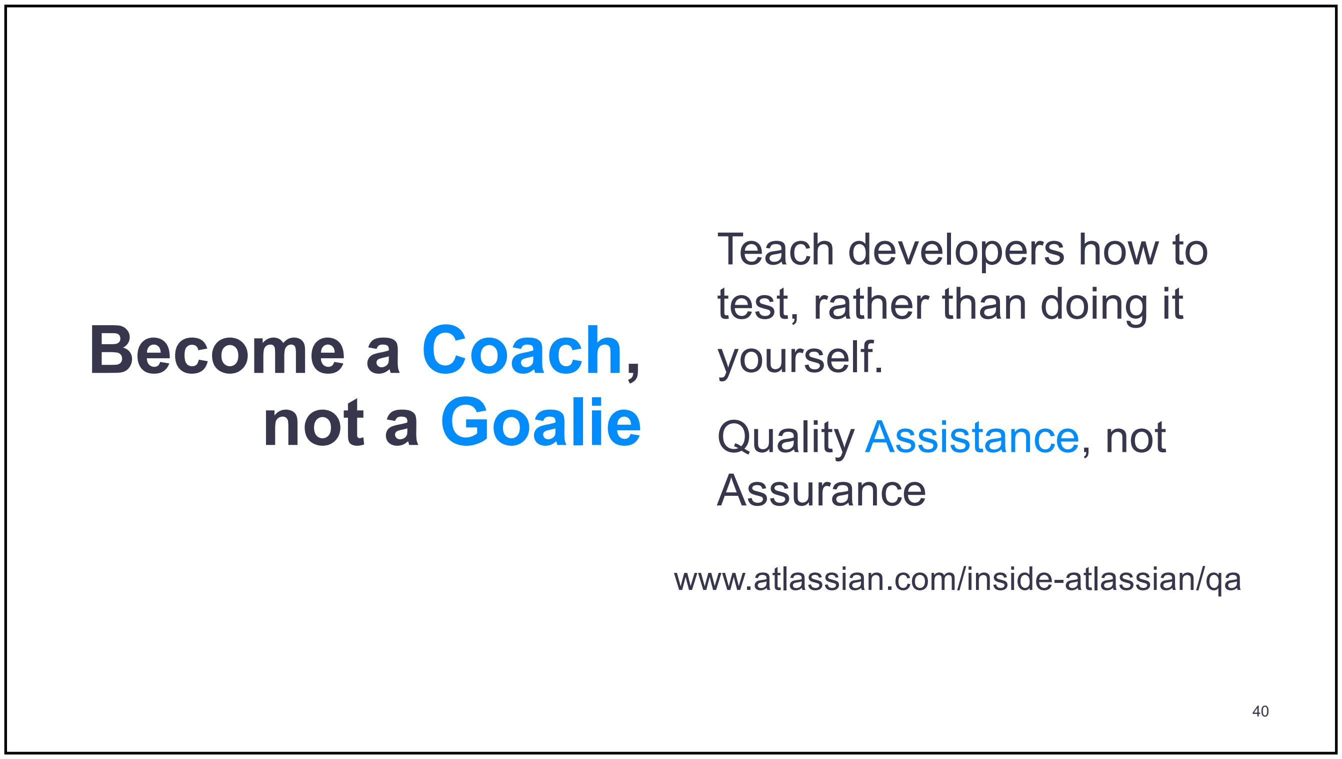
Testing at all levels

Release pipeline

Error detection and reporting in production

Rollbacks and failovers

39



Become a Coach, not a Goalie

Teach developers how to test, rather than doing it yourself.

Quality Assistance, not Assurance

www.atlassian.com/inside-atlassian/qa

40

Robert Williams
Sr. Principal Architect, Service Virtualization
Robert.Williams@ca.com