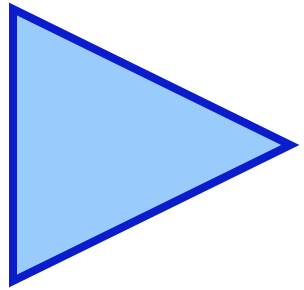
Quality Never Goes Out of Style: Applying Lessons Learned from SOA, Web Service, and REST API Testing to Microservices

Robert Schneider
robert.schneider@wiseclouds.com

Agenda

- Introduction
- About the attendees
- Four decades of distributed computing evolution
- Similarities and differences for testing
- Leveraging proven best practices



About the Attendees

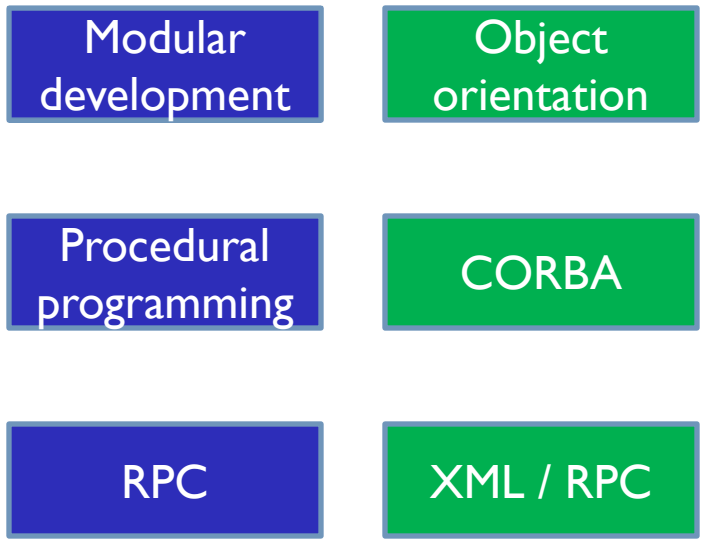
- How many using APIs as gateways to Microservices?
- How many using direct GUI front-end to Microservices?
- How many applying API testing tools for Microservices?
- How many applying GUI testing tools for Microservices?

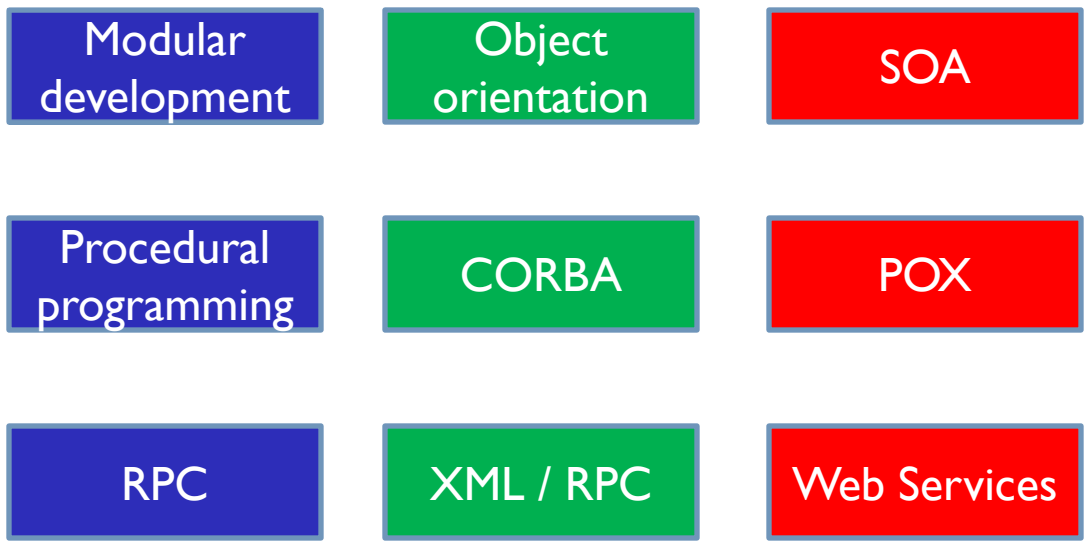
Modular development

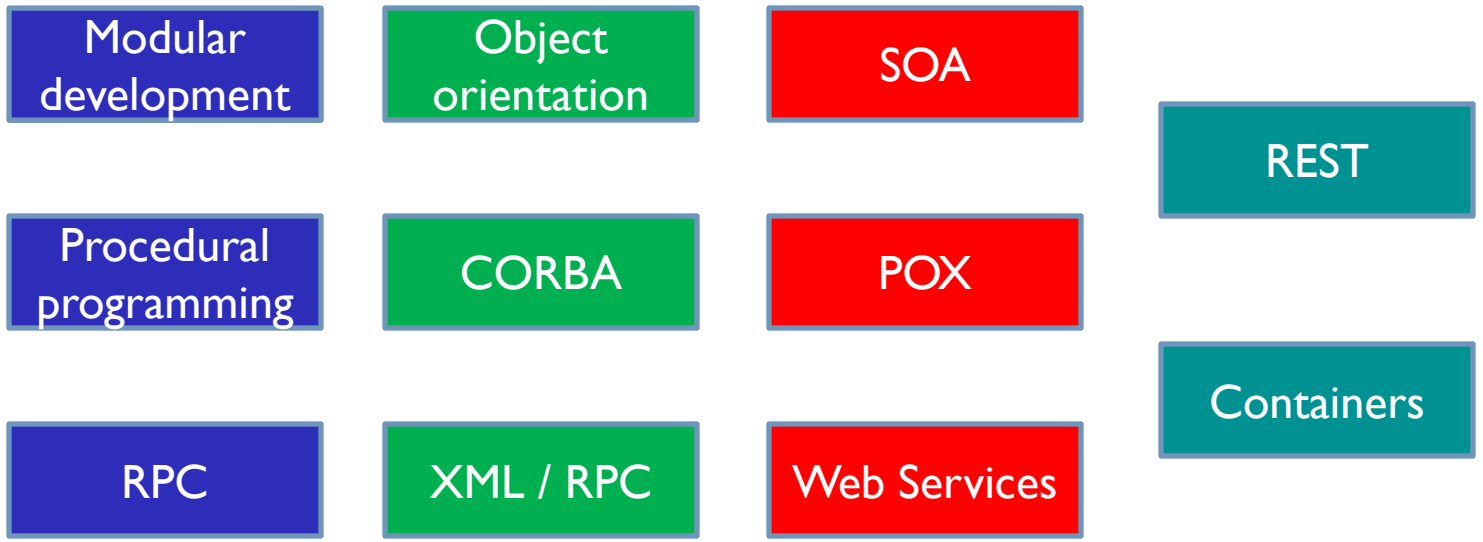
Procedural programming

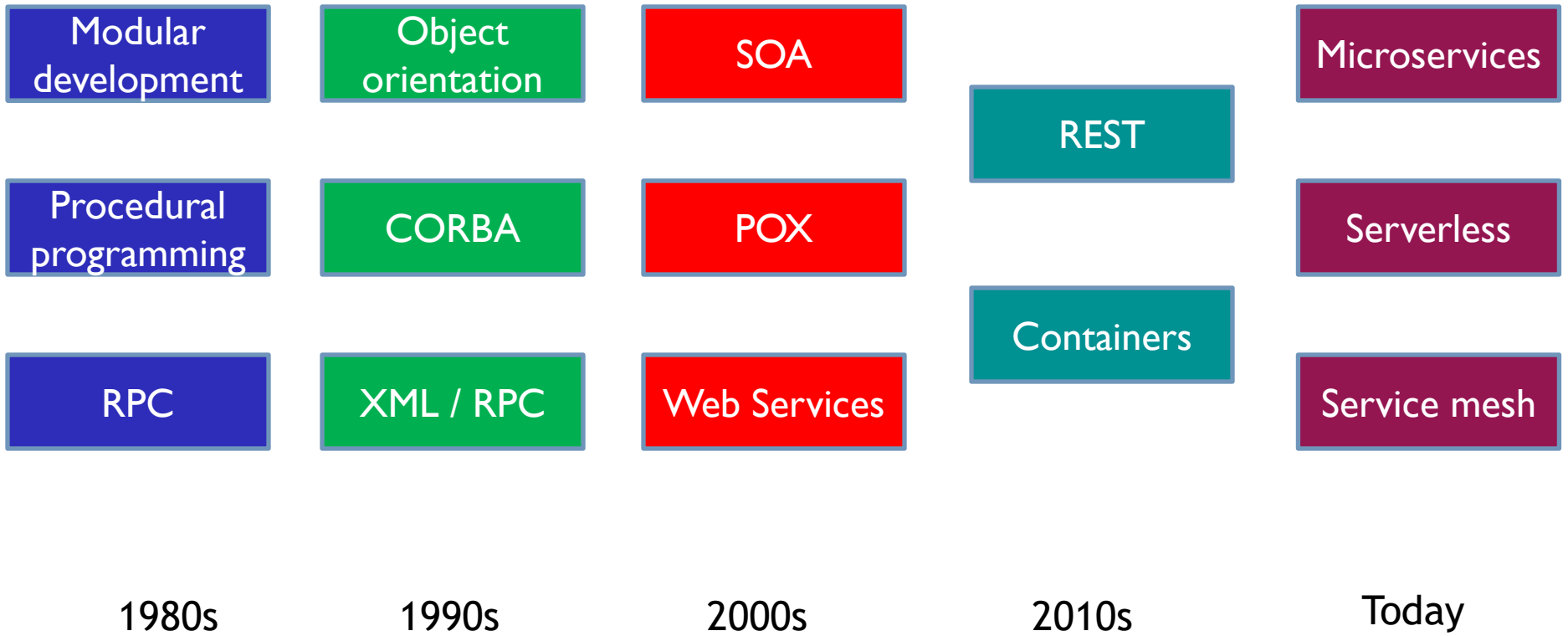
RPC

1980s


Modular development
Object orientation
Procedural programming
CORBA
RPC
XML / RPC
1980s
1990s


Modular development
Object orientation
SOA
Procedural programming
CORBA
POX
RPC
XML / RPC
Web Services
1980s
1990s
2000s


Modular development
Procedural programming
RPC
Object orientation
CORBA
XML / RPC
SOA
POX
Web Services
REST
Containers
1980s
1990s
2000s
2010s


Modular development
Procedural programming
RPC
Object orientation
CORBA
XML / RPC
SOA
POX
Web Services
REST
Containers
Microservices
Serverless
Service mesh
1980s
1990s
2000s
2010s
Today

The More Things Change…

- Like fingerprints and snowflakes, no two SOA implementations, Web services, or REST APIs are identical
  - The same is true for Microservices

- Designing these assets is a combination of art and science
  - But there are universal, well-proven testing patterns
  - And these best practices remain relevant today

SOA vs. Microservices

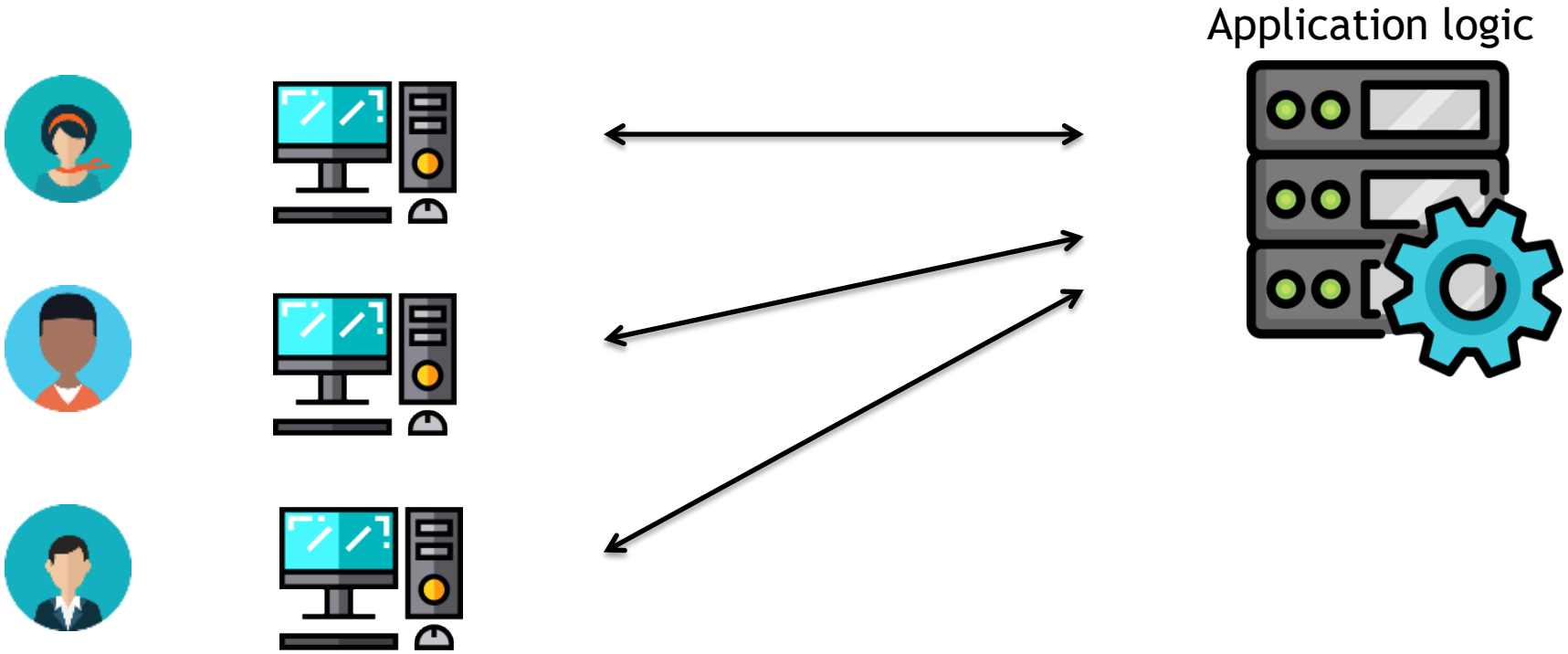
SOA

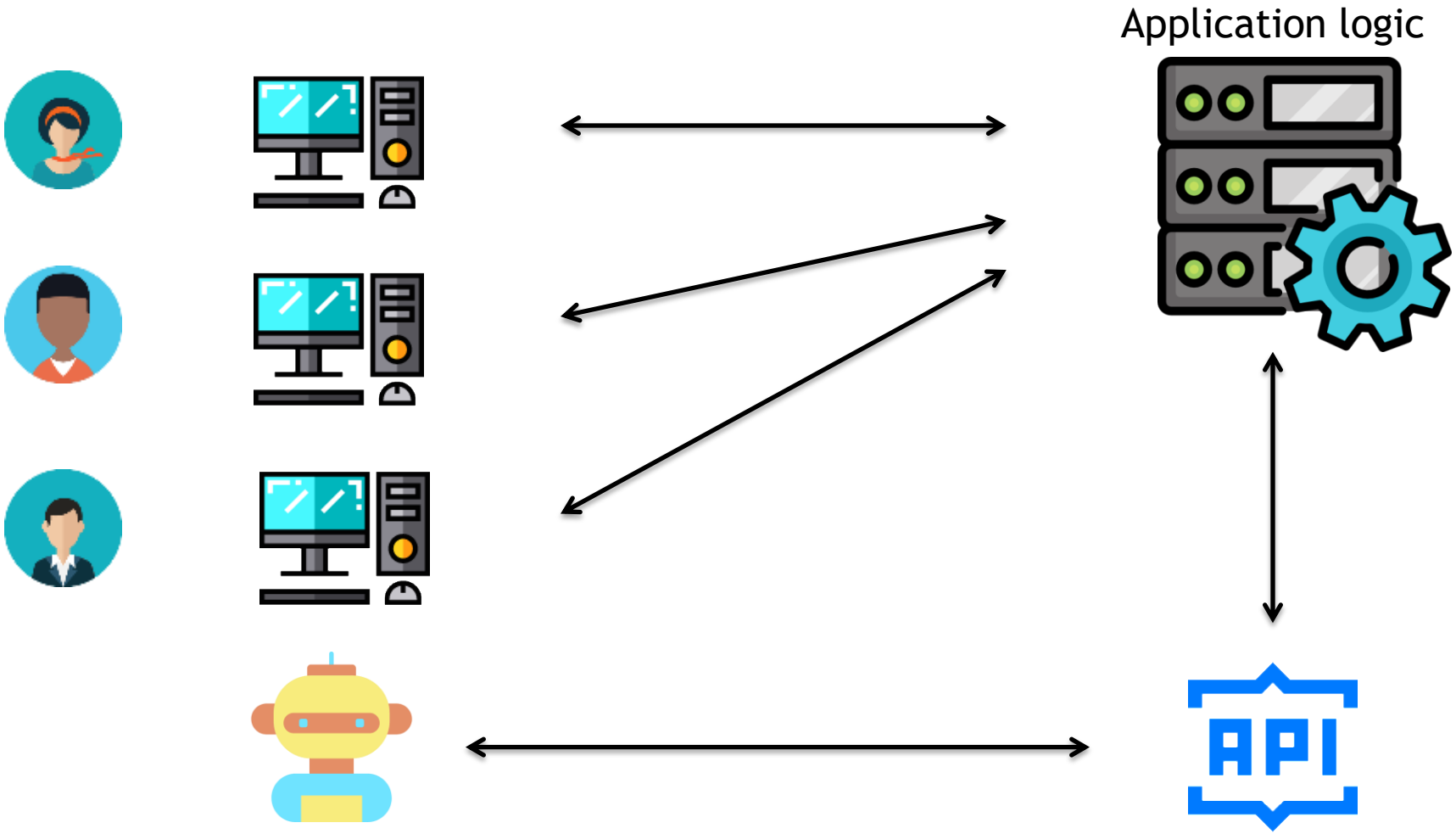
- Focus on reusability
- Not reliant on DevOps
- Multiple protocols
- Deployed on base platform
- Not meant for containers
- Governance is essential

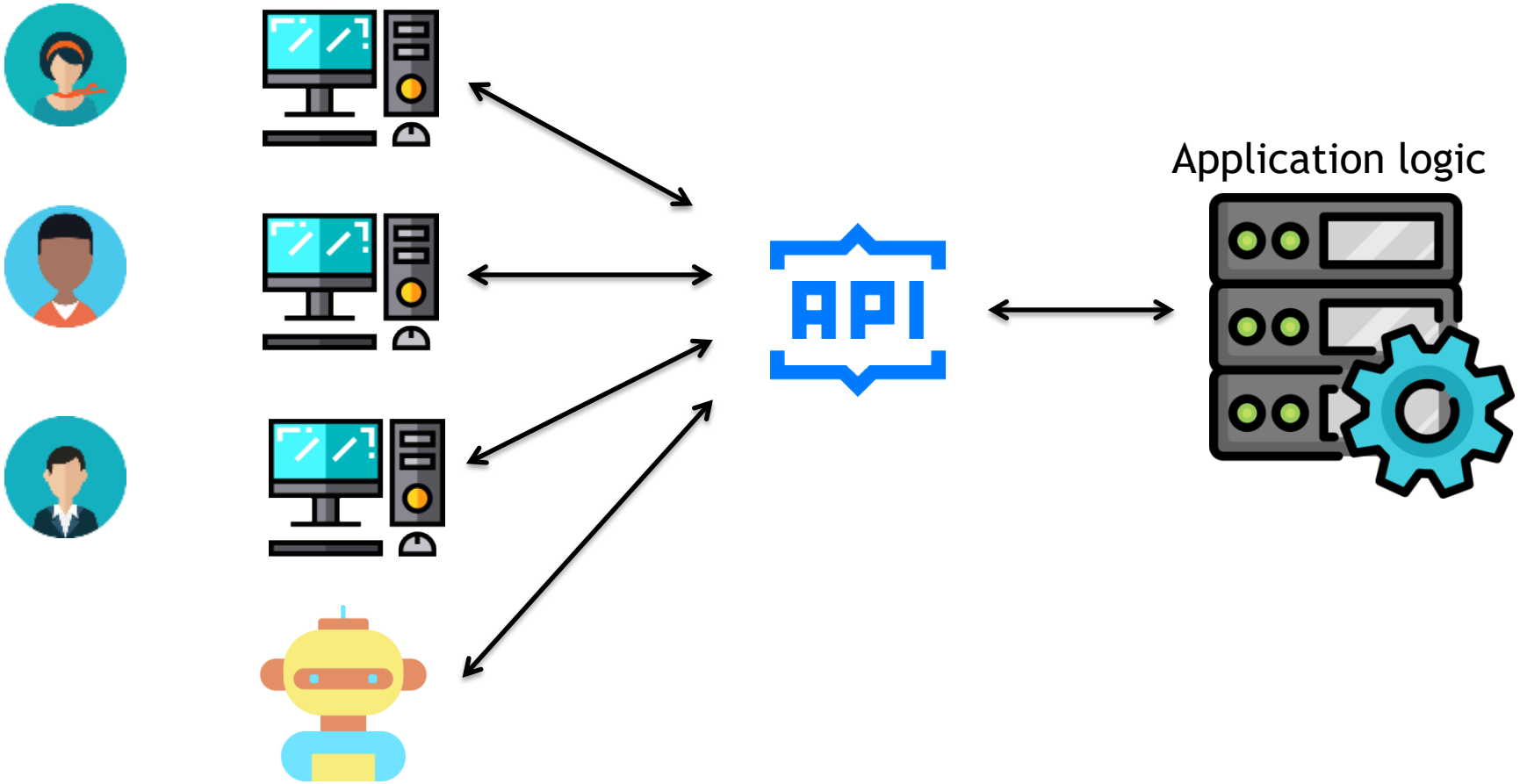
Microservices

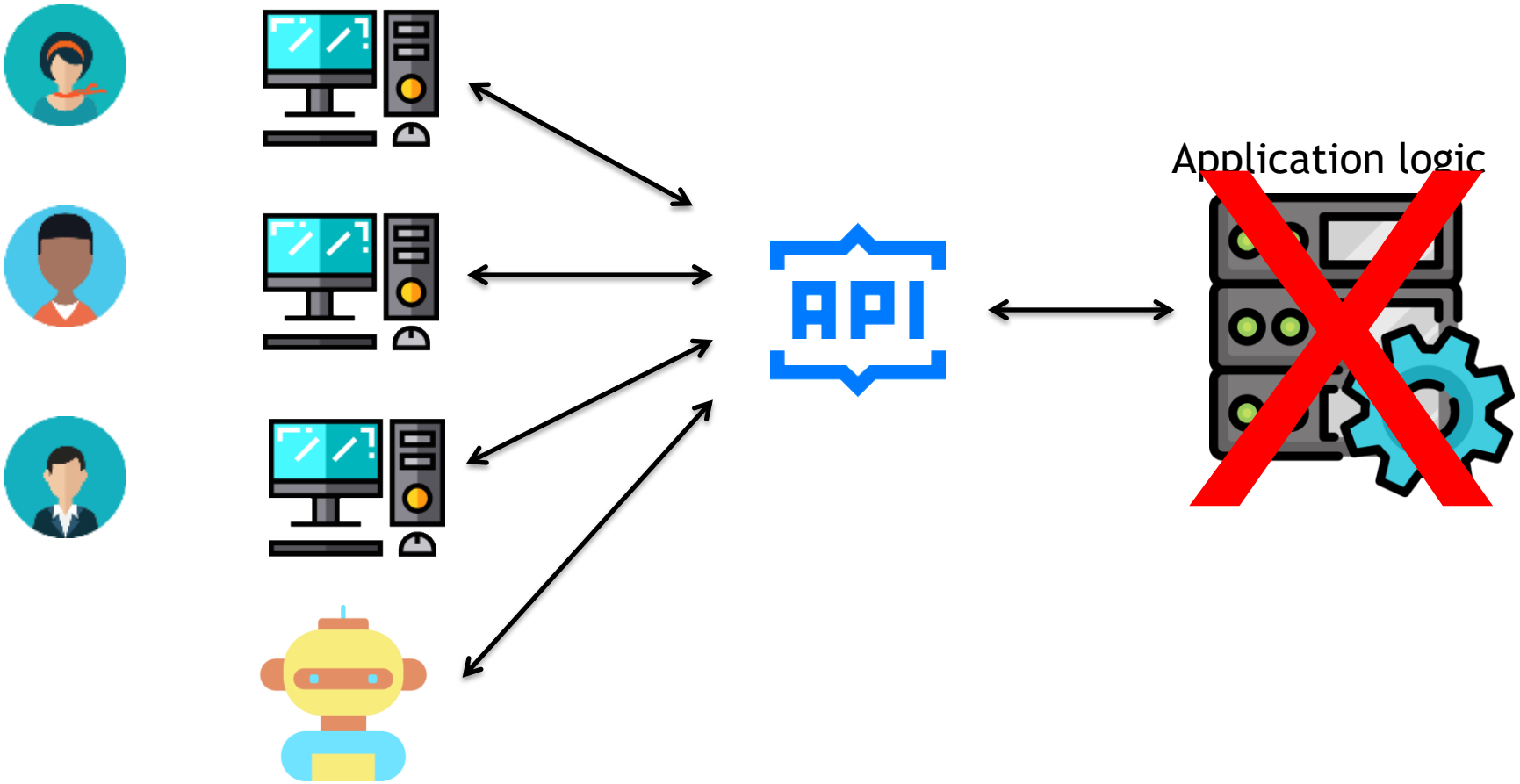
- Focus on decoupling
- Uses DevOps/CD
- Lightweight protocols
- Deployed in cloud
- Ideal for containers
- Governance is optional

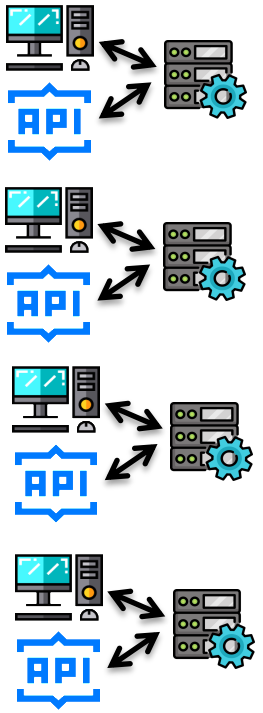
Courtesy Rade Despodovski/Dzone.com

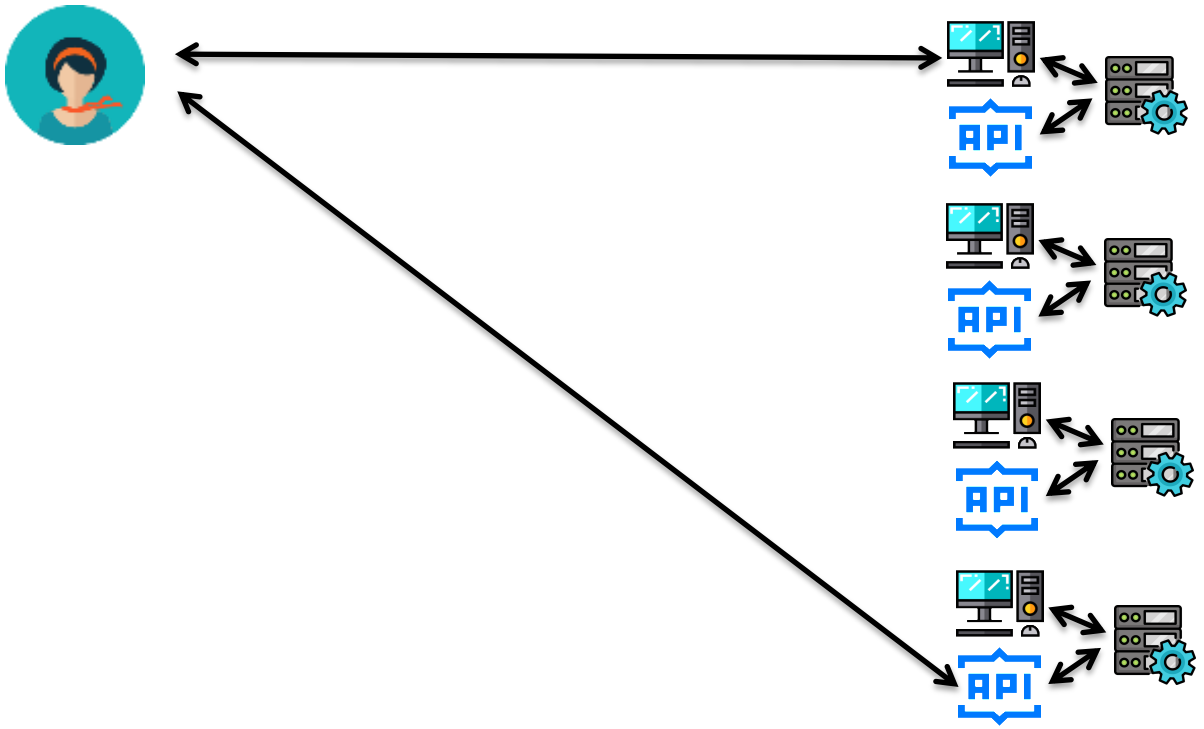

Application logic

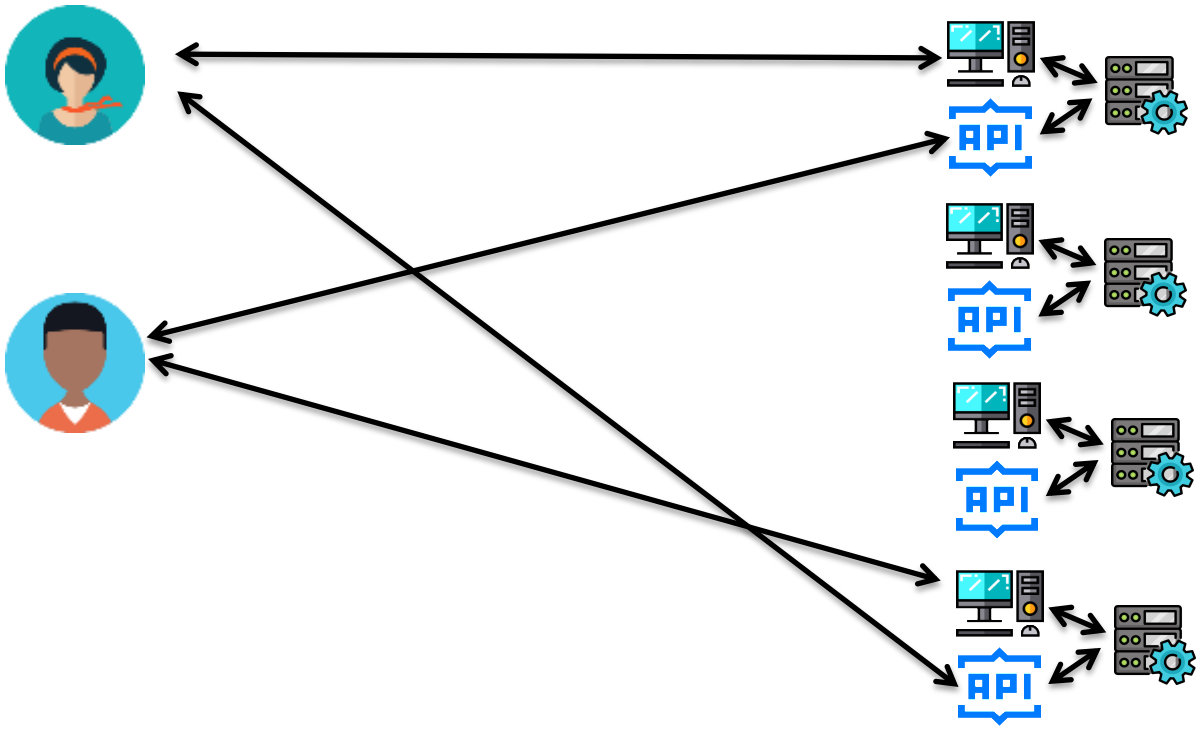

Application logic
API

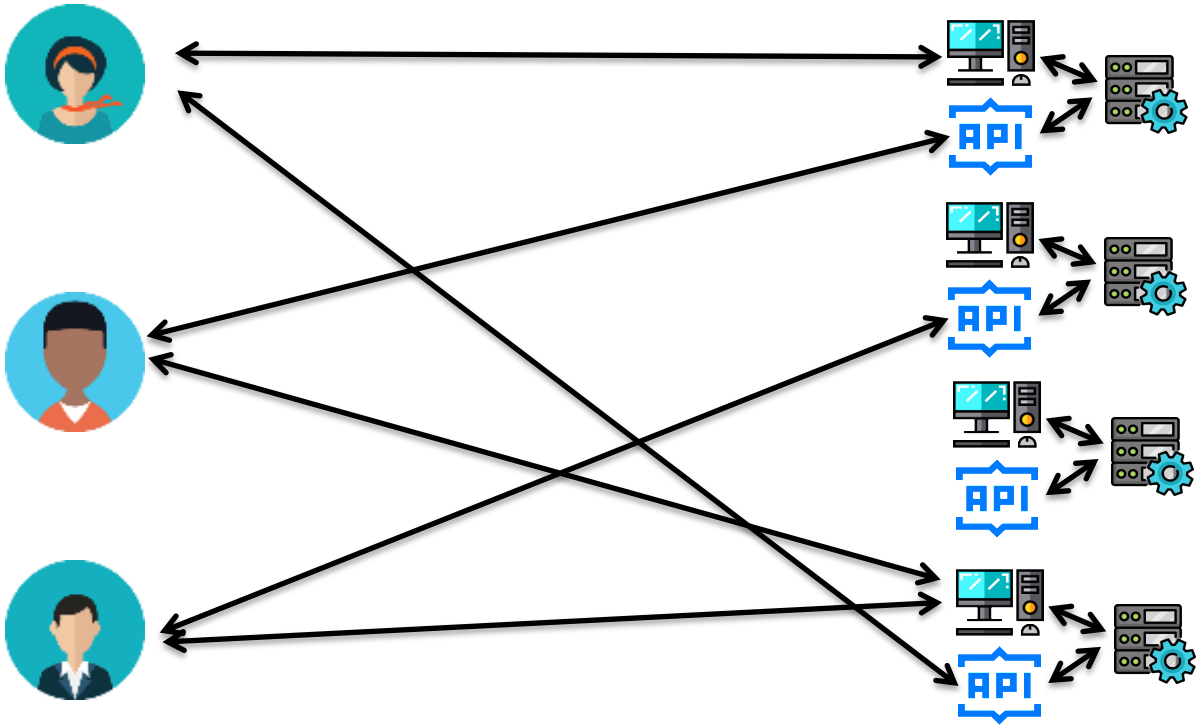

API
Application logic


Application logic
API

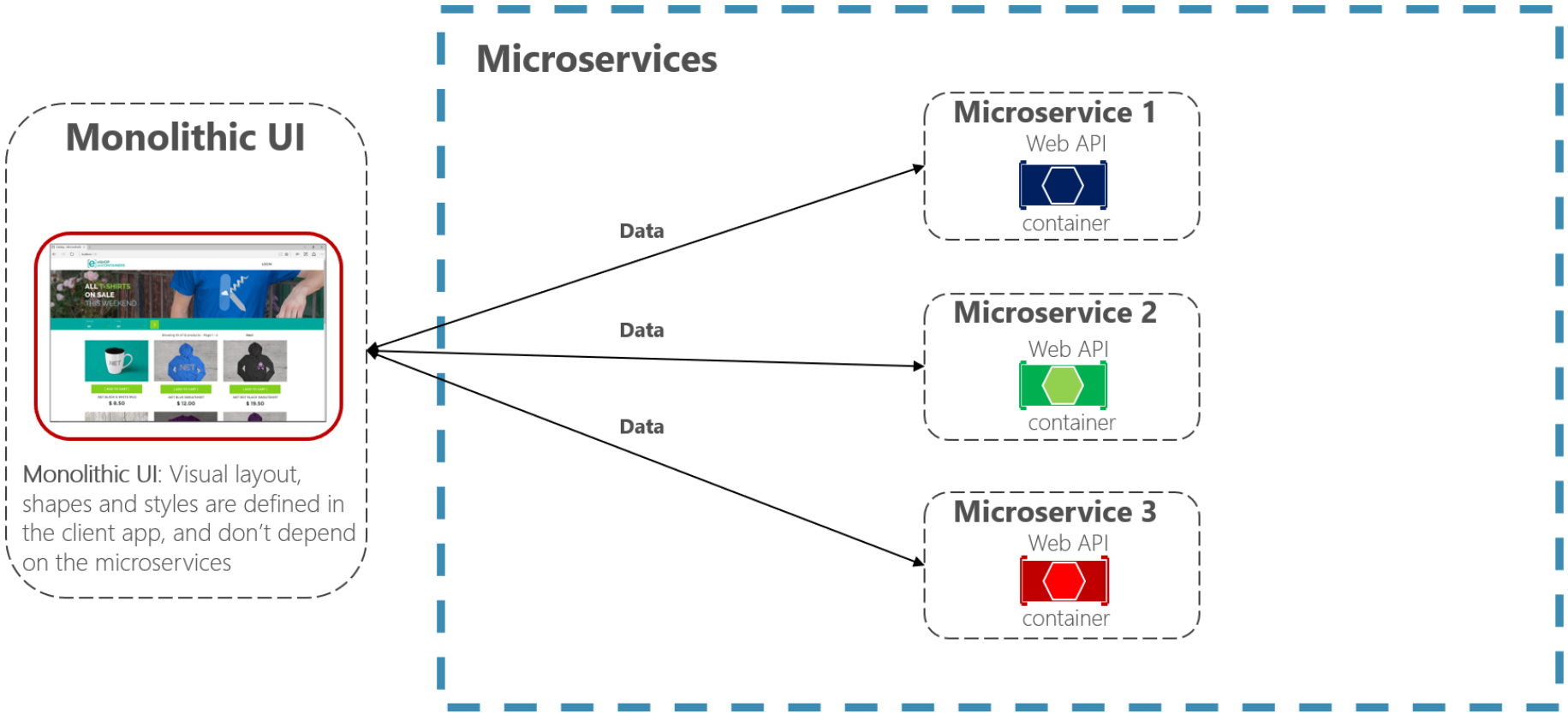

API
API
API
API


API
API
API
API


API
API
API
API

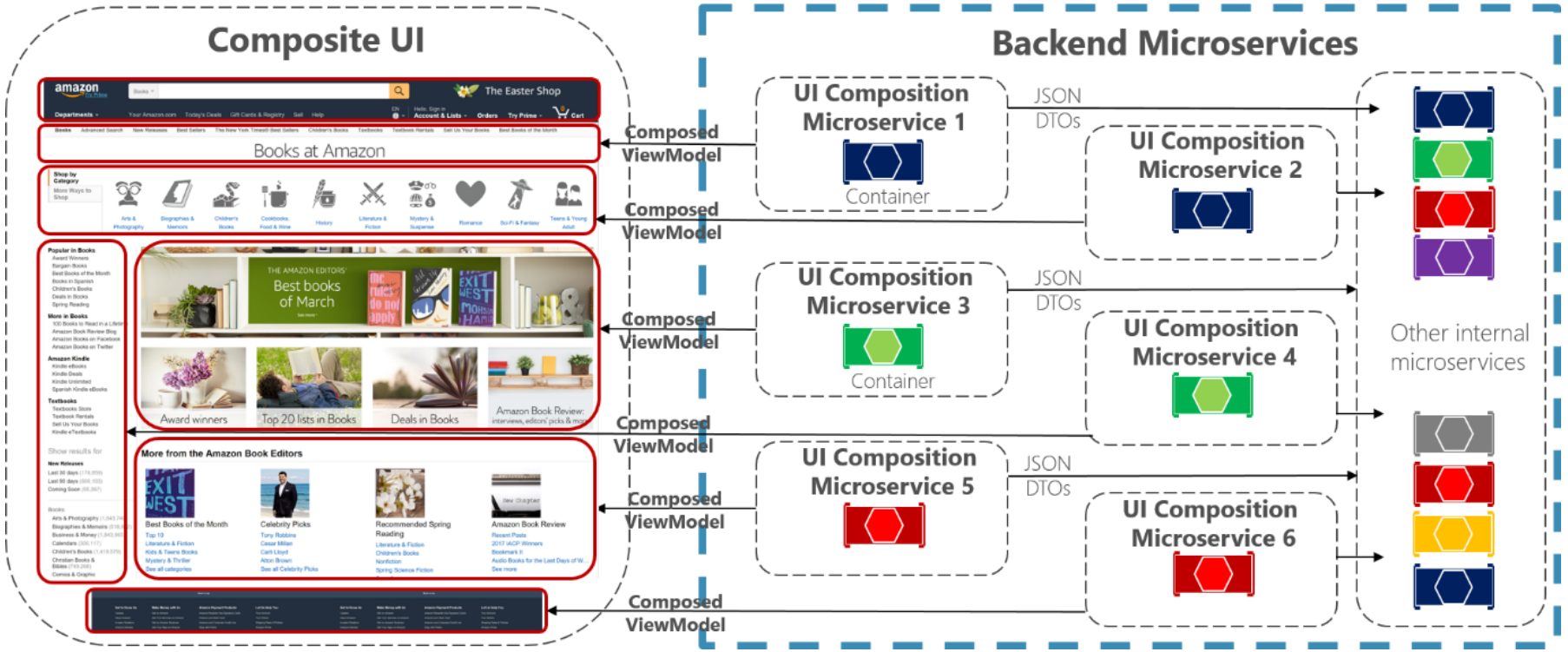

API
API
API
API

Monolithic UI consuming microservices



Courtesy Microsoft

Composite UI *generated* by microservices



Courtesy Microsoft

What Hasn't Changed About Testing?

- Lots of resources (i.e. APIs, Microservices) to evaluate
- Mission-critical assets
- Yet often under-examined
  - Complexity
  - Time pressure
  - Lack of process
  - Insufficient tooling

What's Different about Testing?

- Microservices often are a much more fine-grained implementation
- More likely to leverage cutting-edge architecture
  - Containers
  - Serverless
  - Service mesh
- GUI front-end complications

Leveraging Proven Best Practices

- There's no need to reinvent the wheel when testing Microservices

- Each of these patterns is still relevant:
  - Apply testing automation
  - Use meaningful amounts of data
  - Test for performance
  - Employ governance whenever possible
  - Go beyond unit testing

Applying Test Automation

- What hasn't changed?
  - CI/CD
  - Agile

- Microservice differences
  - Developers regularly build unit tests in the front end
  - This purely GUI approach makes it difficult to attain full code coverage

Recommendations

- Carry out testing automation on two fronts
  - Use BDD tools such as Cucumber to drive End-to-End (E2E) tests
  - Use the OpenAPI specification + test tools to automate heavy volumes of testing via the API
  -
- This two-pronged strategy is both scalable and thorough

Data-Driven Testing

- What hasn't changed?
  - Hard-coded data
  - Very few scenarios
  - Minimal assertion coverage on responses

- Microservice differences
  - GUI-driven testing can make it far too time consuming to develop hundreds, thousands, or millions of permutations
  - Not easy to use randomized data

Recommendations

- Use ETL tools to copy data from production to staging
  - Remove PII or other sensitive information
- Consider data generators to power API side of testing

Test for Performance

- What hasn't changed?
  - High traffic volume in production
  - Yet this type of testing is frequently overlooked

- Microservice differences
  - Autoscaling infrastructure has made load testing less important
    - "Throw hardware at the problem"
  - But this can result in unnecessary costs for non optimized microservices

Recommendations

- Modern API testing tools integrate nicely with performance testing technologies
  - This two-pronged strategy is both scalable and thorough
- Include orchestrated functional tests in performance evaluations
  - Not just unit tests at high volume

Governance

- What hasn't changed?
  - Brittleness of test assets
  - Lots of 'reinventing the wheel'

- Microservice differences
  - Tooling has greatly improved
  - So have procedures

Recommendations

- Employ technology wherever possible
  - Git
  - Wikis
- Create a ‘single point of truth’
- Build a culture of sharing and reuse
  - Including keeping track of test results over time
  - Leverage dashboards of test platforms, DevOps metrics, ALM tools

Orchestration

- What hasn't changed?
  - Applications are composed of many individual assets working in a logical sequence
- Microservice differences
  - Developers are more focused on unit testing
  - Diminished attention to longer-running business processes
  - Integration testing is particularly hard

Recommendations

- Gain knowledge of BDD
- Use this information to drive both front-end E2E and API tests
  - Stories are relevant regardless of the testing tool being applied
- Apply Gherkin's `Given-When-Then` as a natural language approach to capturing behaviors for a user story
- Use API mocking/virtualization to help with integration

Quality Never Goes Out of Style: Applying Lessons Learned from SOA, Web Service, and REST API Testing to Microservices

Robert Schneider
robert.schneider@wiseclouds.com