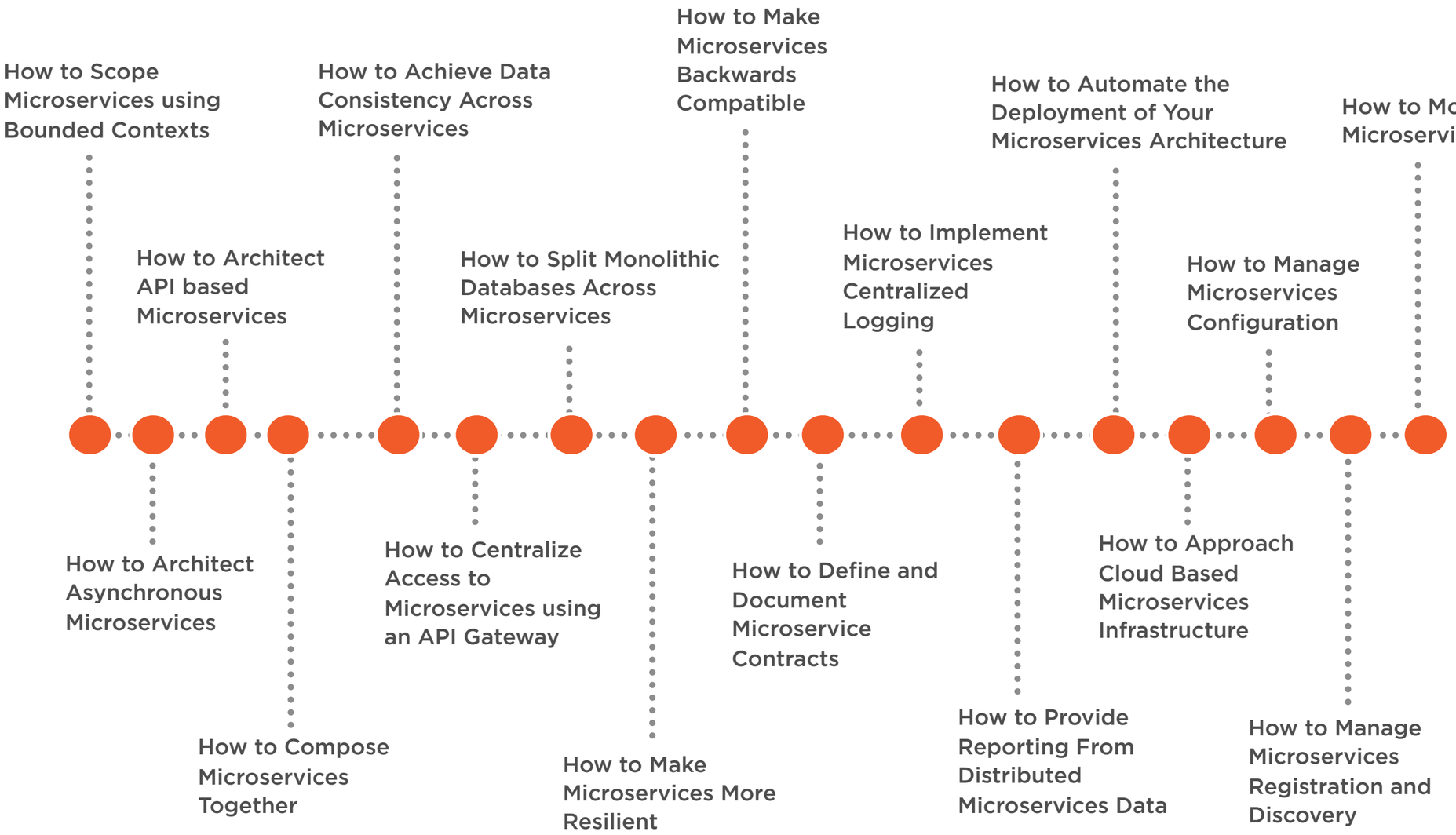
How to Make Microservices Backwards Compatible

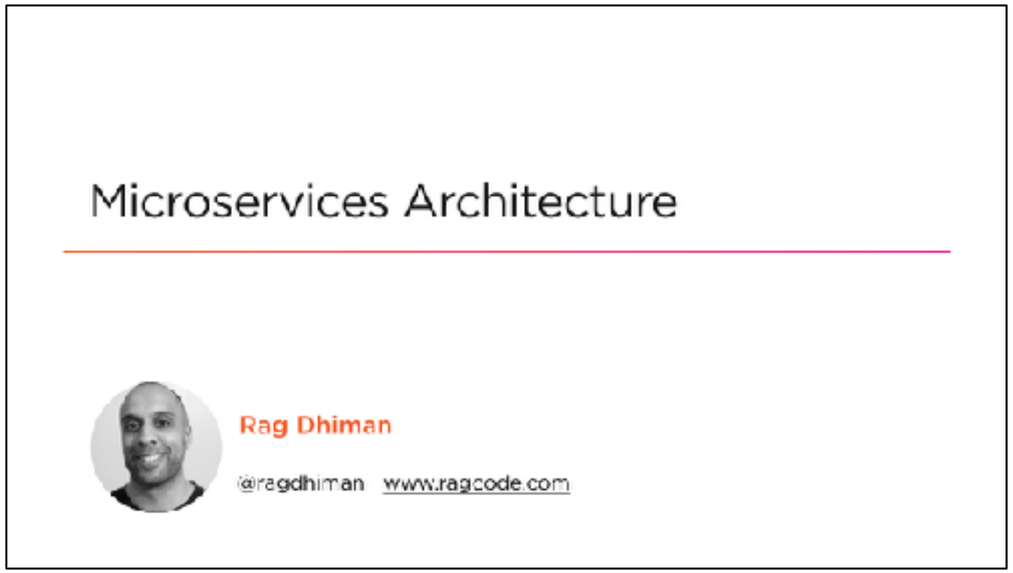
Rag Dhiman

@ragdhiman www.ragcode.com

Microservices Architectural Design Patterns Playbook



Microservices Architectural Design Patterns Playbook



Overview

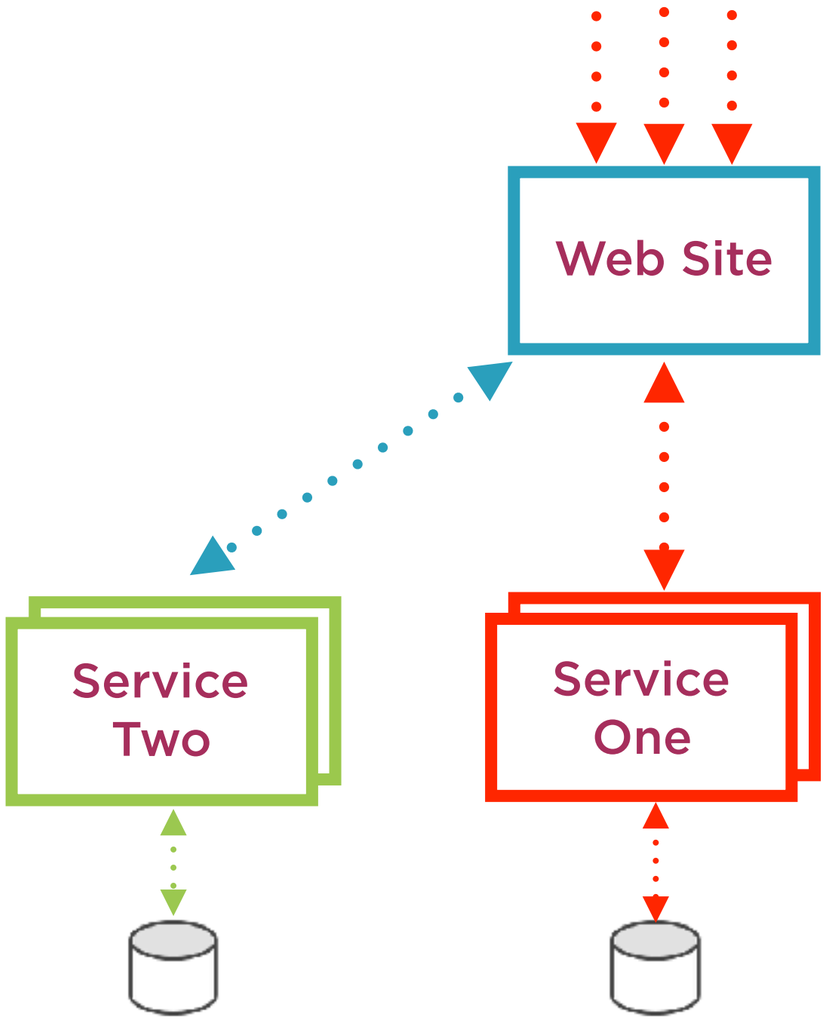
Introduction

Awareness of Compatibility

Testing Compatibility

Versioning Strategies

Introduction



Microservices USPs

- Independently changeable
- Independently deployable

Need for compatibility

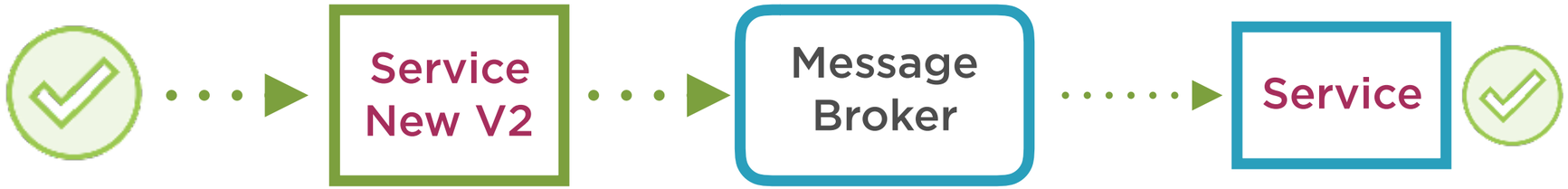
- Backwards compatibility
- Forwards compatibility

Compatibility with:

- Client applications
- Client services
- Overall architecture
- Third parties

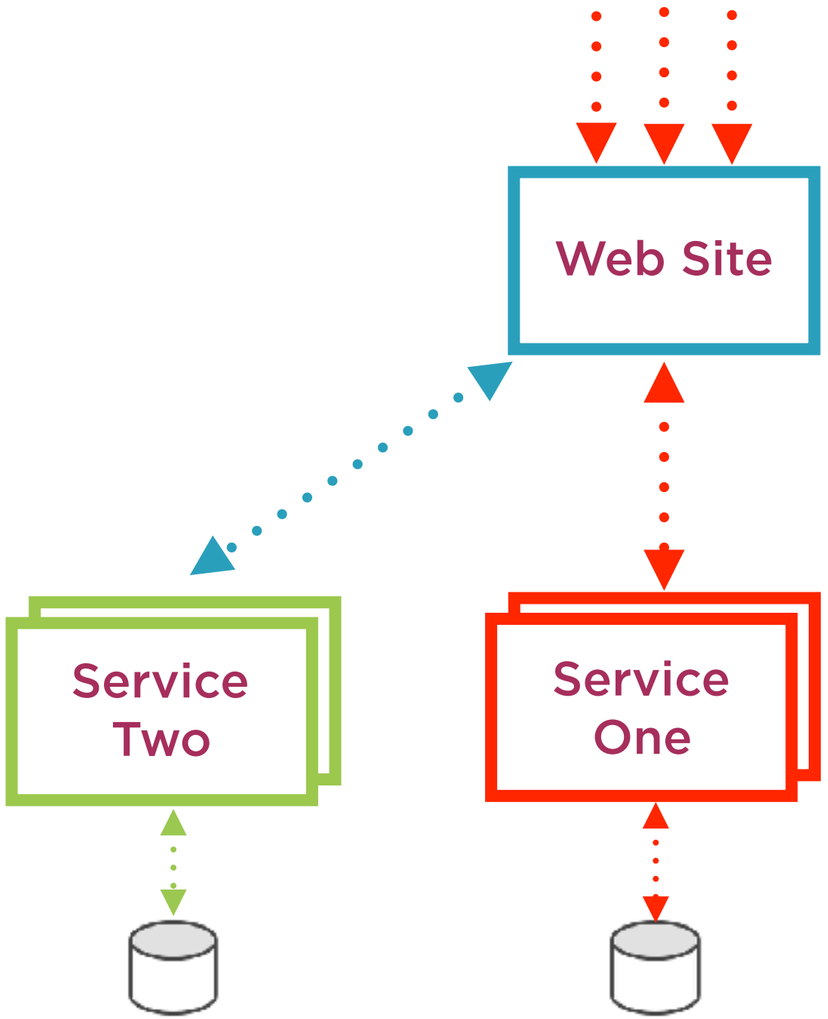
Backwards Compatibility

Backwards Compatible



Backwards Incompatible

Backwards Compatibility Issues



Contract breakage

- Interaction agreement is broken

What is a contract

- Defines interactions between applications
- Endpoints, parameters, payloads and responses

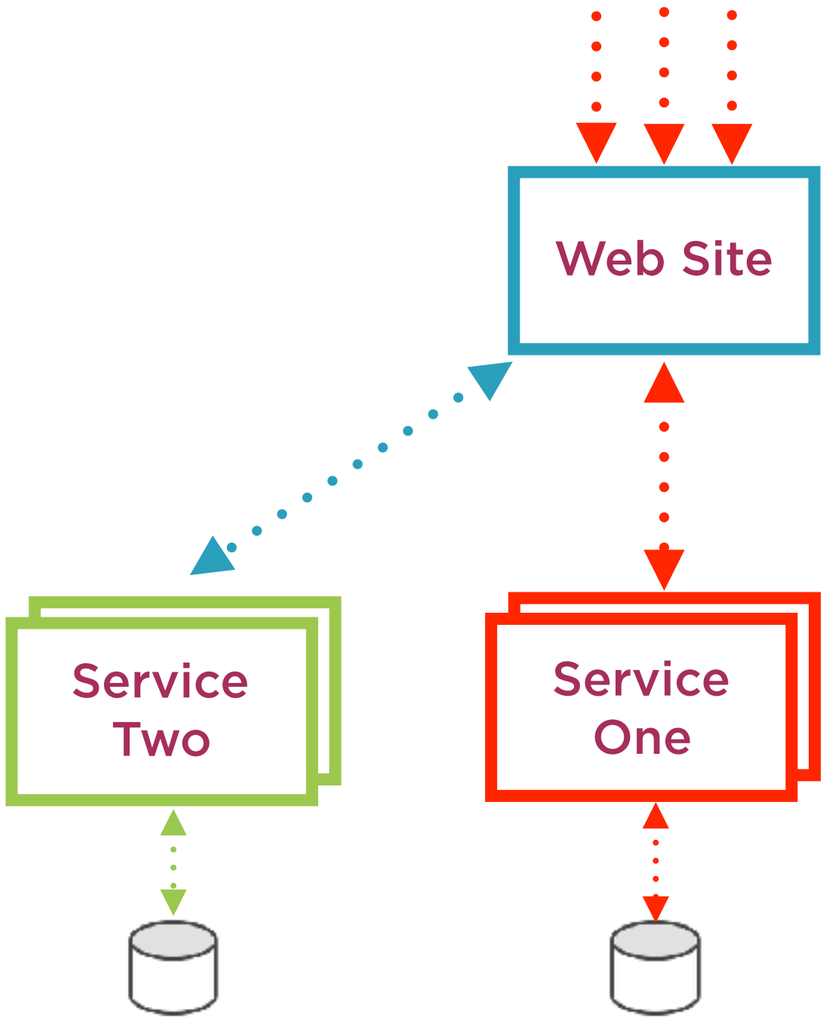
Contract consumer

- Service or app that consumes your API

Contract provider

- Service that exposes an API

Backwards Compatibility: How?



Awareness of compatibility

- Incompatible contract changes
- Compatible contract changes
- Contract consistency

Testing compatibility

- Automation

Communicating compatibility

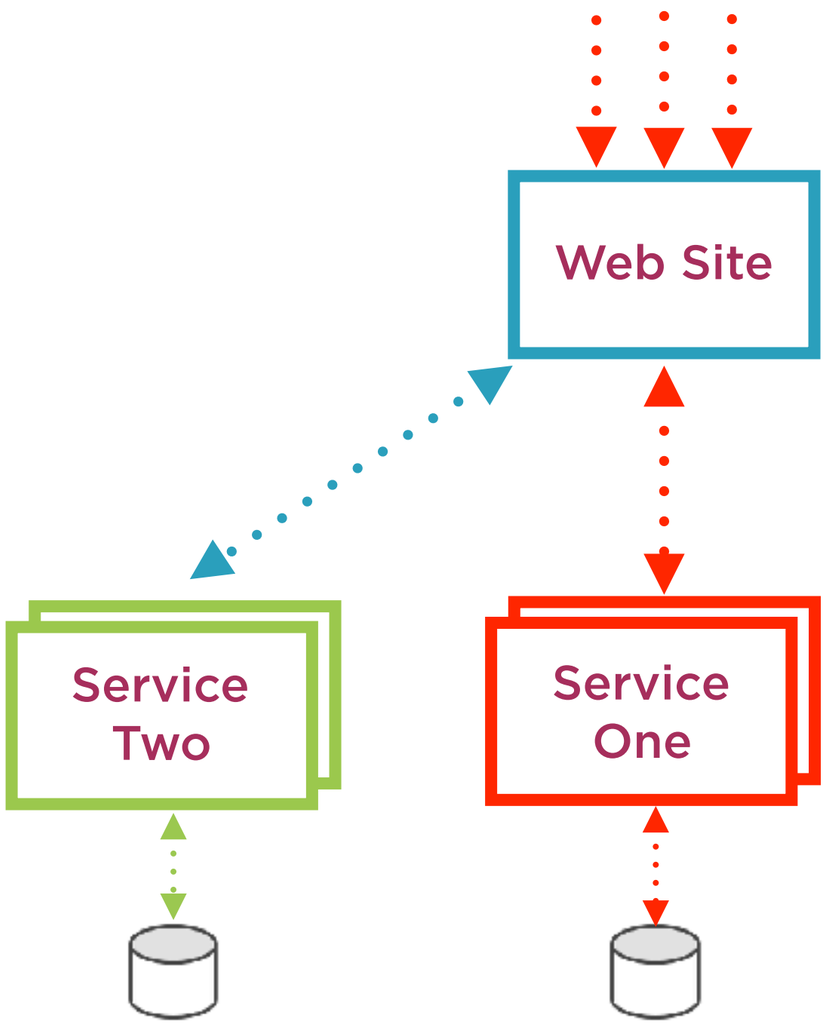
- Versioning strategy

Managing breaking changes

- Migration plan

Awareness of Compatibility

Incompatible Changes



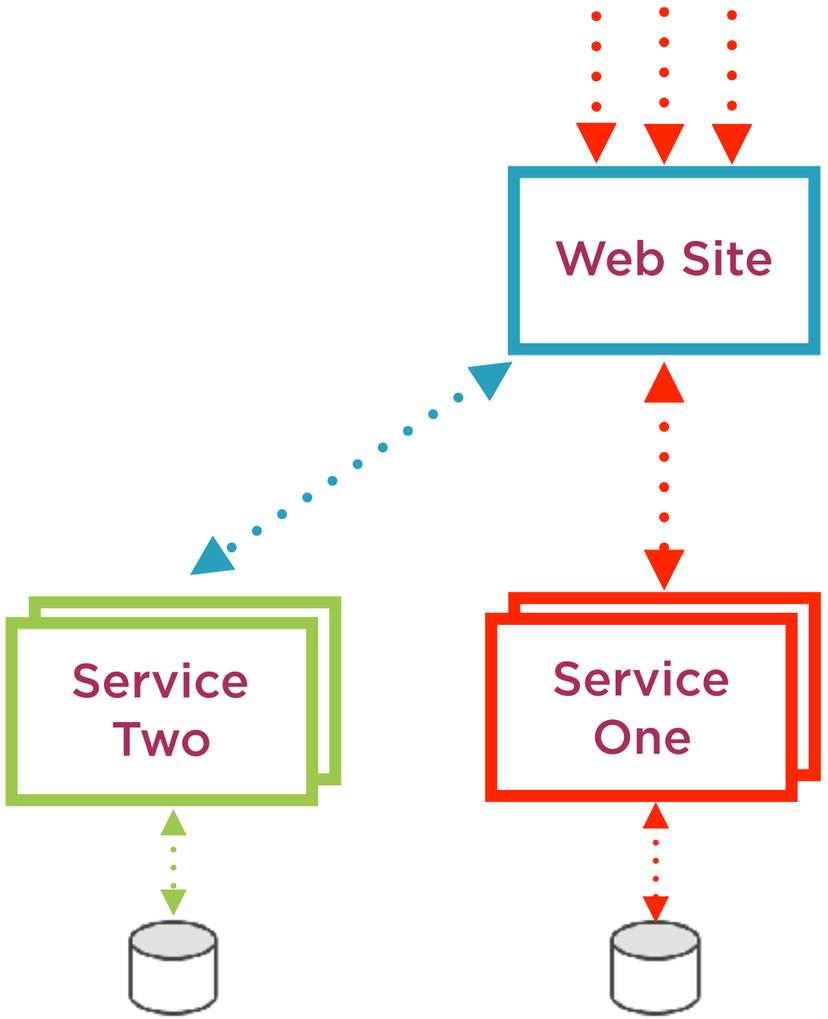
Breaking the contract for client calls

- Renaming operation/resource or data
- Removing operation/resource or data
- Optional data becomes mandatory
- Adding a new validation/exception
- Changing data constraints and types
- Changing endpoints
- Changing security
- Changing a standard

Breaking the contract for recipients

- Renaming or removing data
- Changing data format and types

Compatible Changes



Contract for client calls

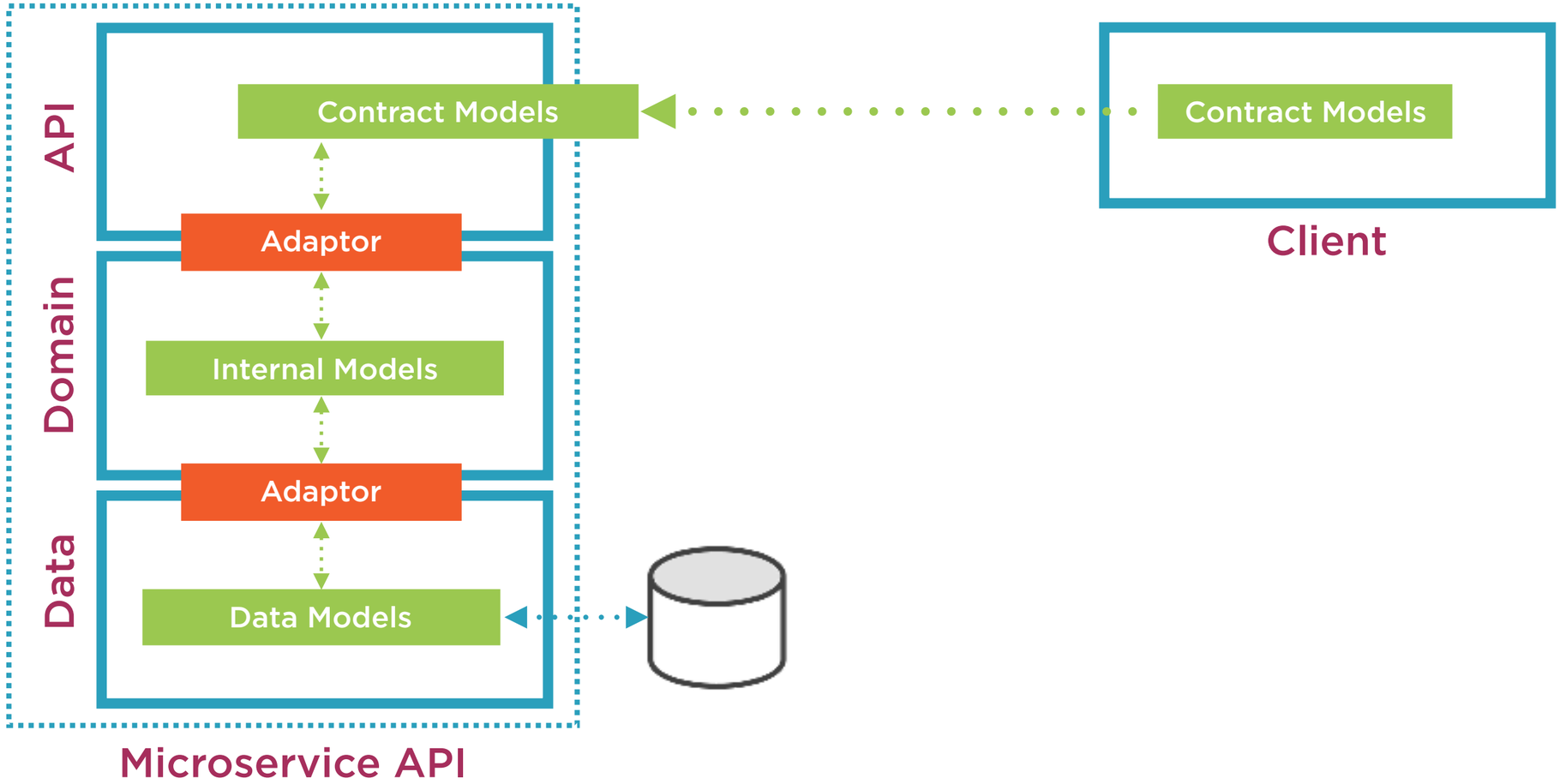
- Adding an operation/resource
- Adding optional data
- Making mandatory data optional
- Reducing the constraints on data
- Implementing an existing operation/resource
- Redirection response codes

Contract for recipients

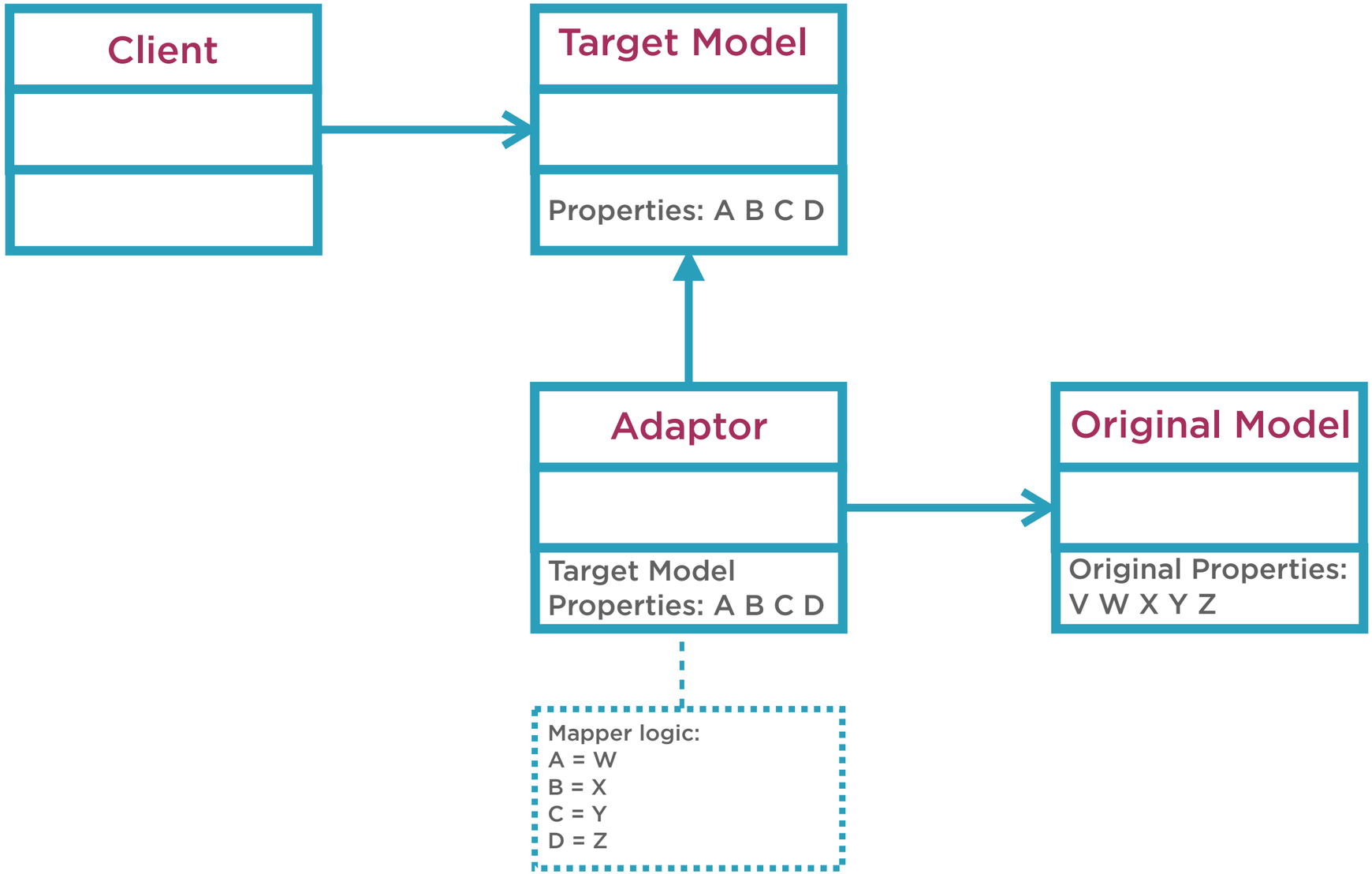
- Adding additional data to the schema

How to Achieve Contract Consistency

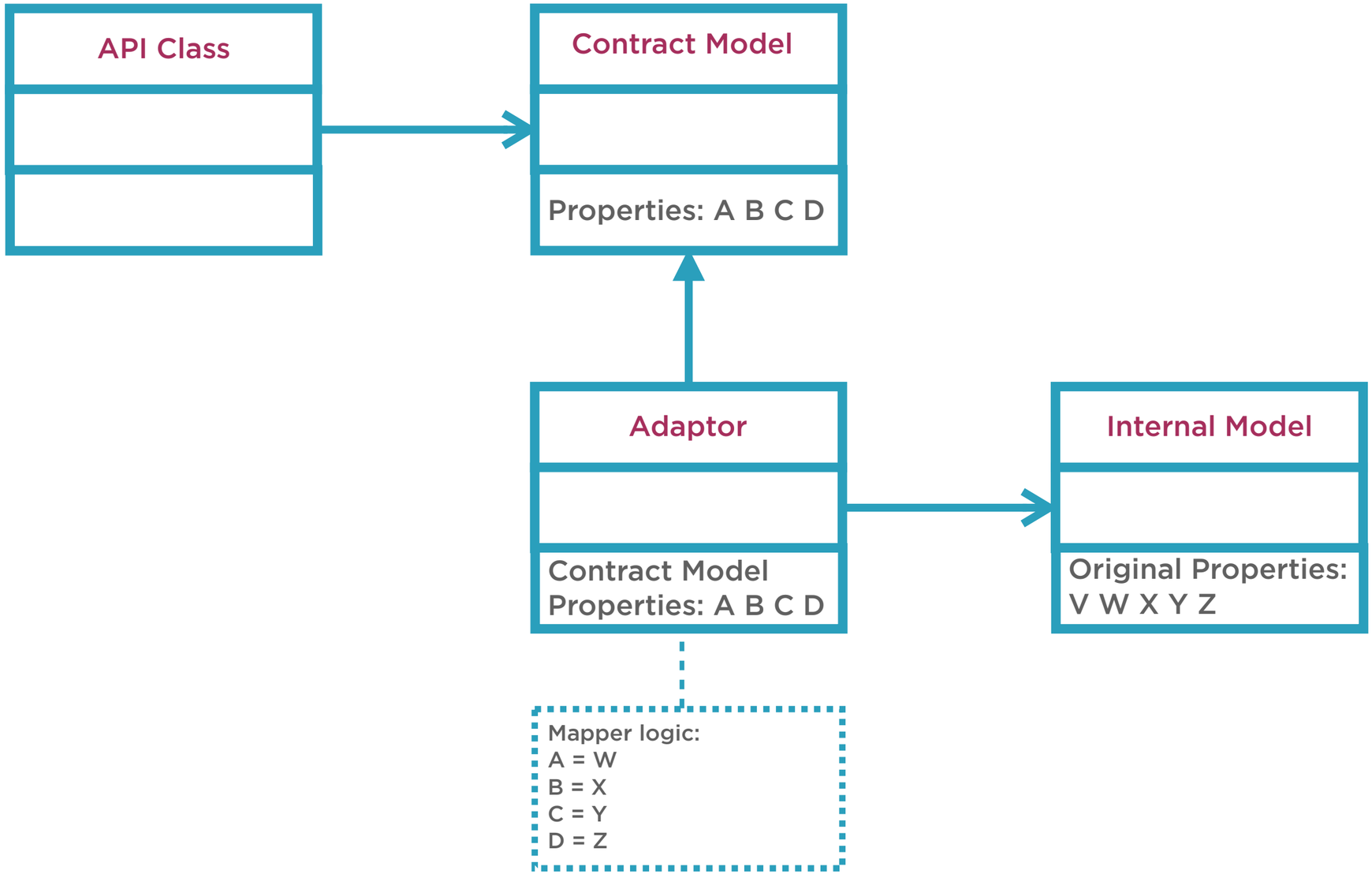
Shared Models and Separation of Concerns



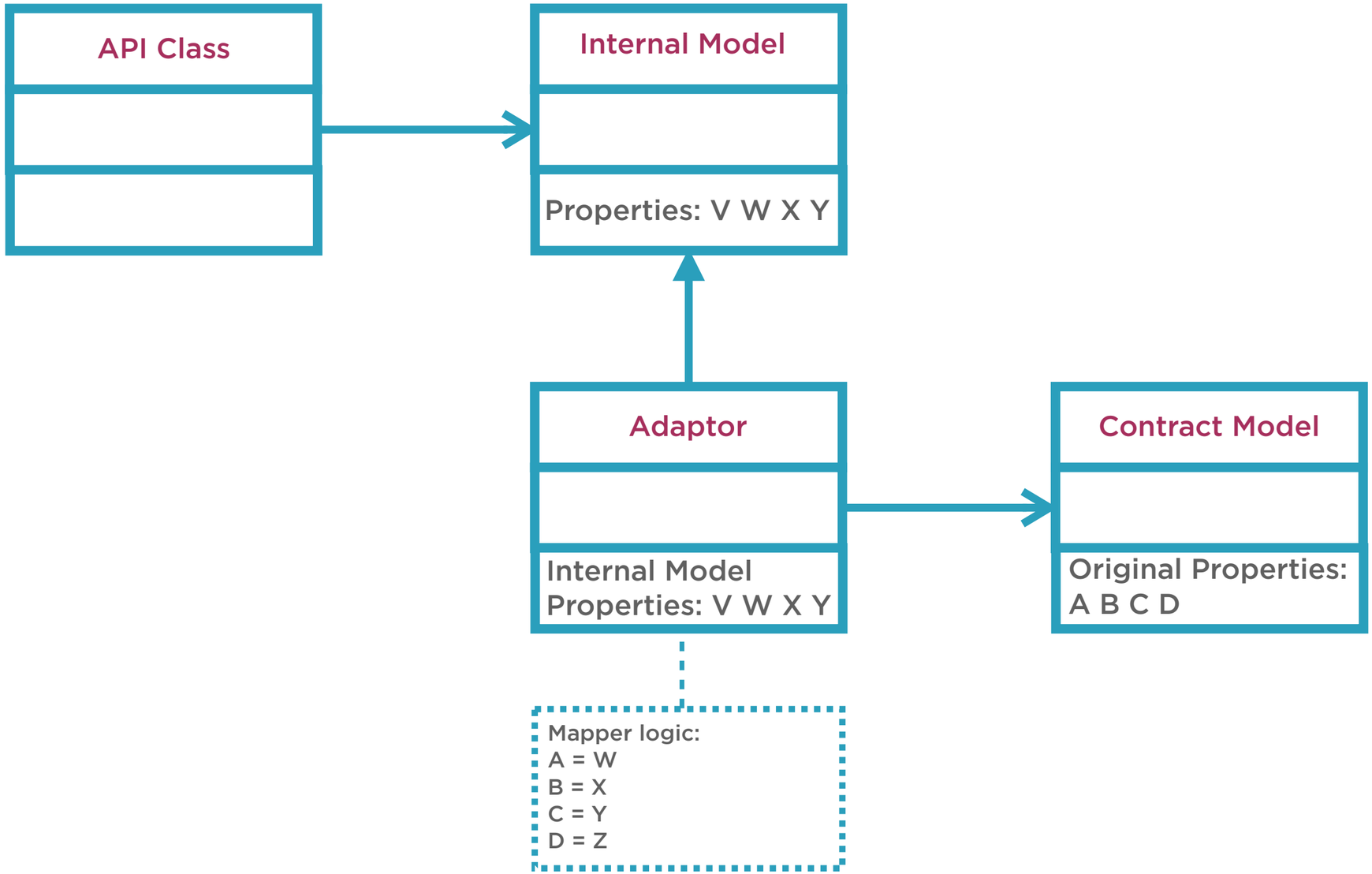
Adaptor Design Pattern



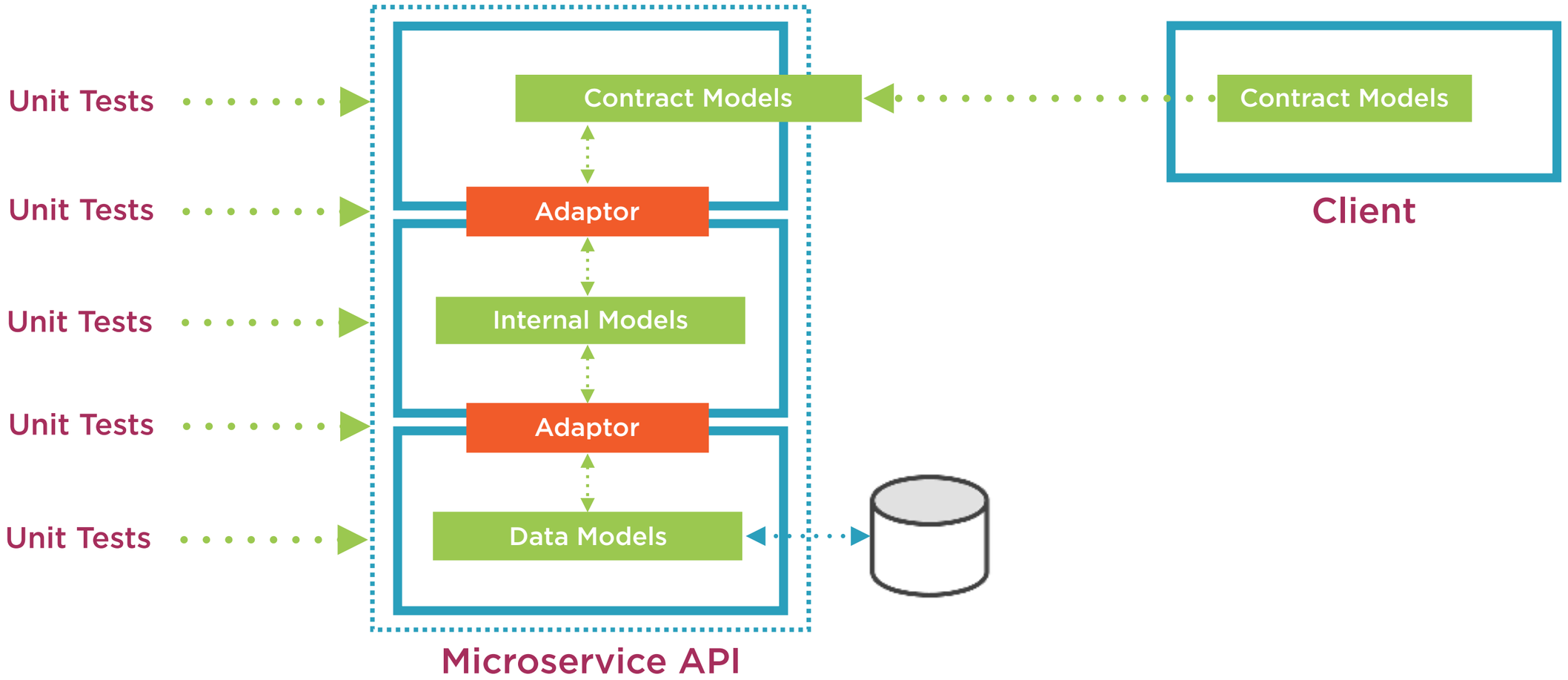
Adaptor Design Pattern



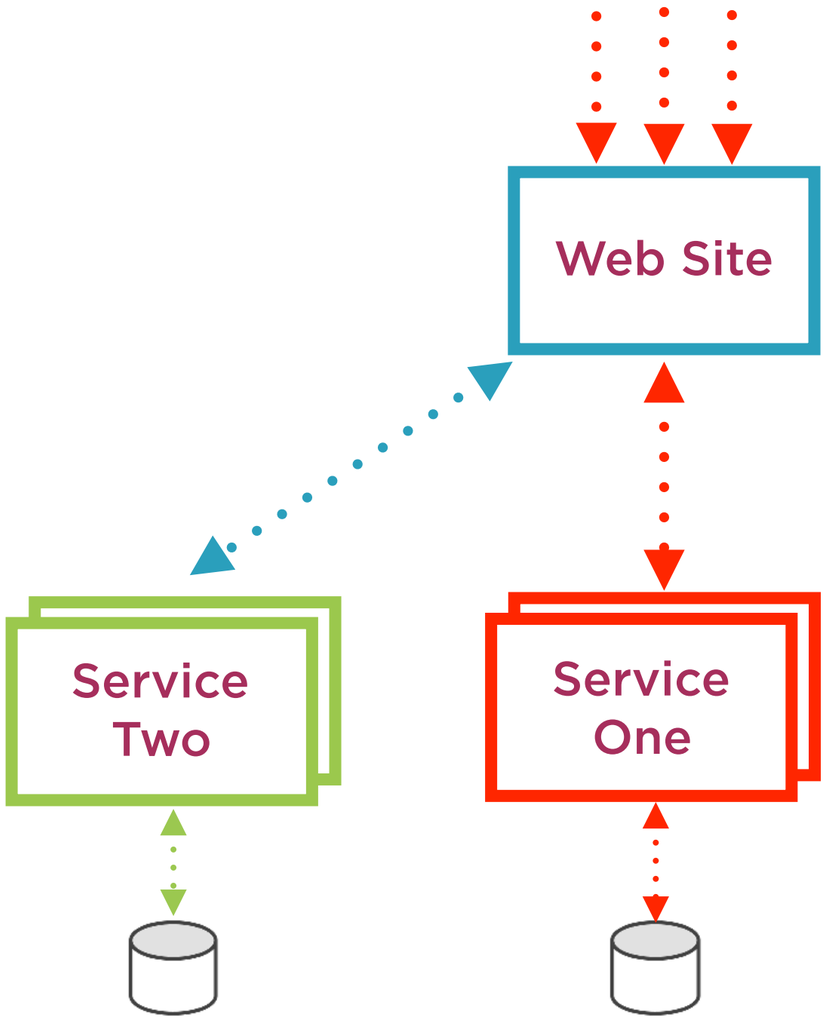
Adaptor Design Pattern



Unit Test Models and Adaptors



How to Achieve Contract Consistency



Make awareness part of the design

- Incompatible vs. compatible

Make awareness part of code reviews

Automated integration testing

- Part of continuous integration strategy

Separate internal models from contract

- Use shared models
- Separation of concern
- Use mappers and adaptor design pattern
- Unit tests to validate mappers and adaptors

Testing Compatibility

Automation

End to end testing

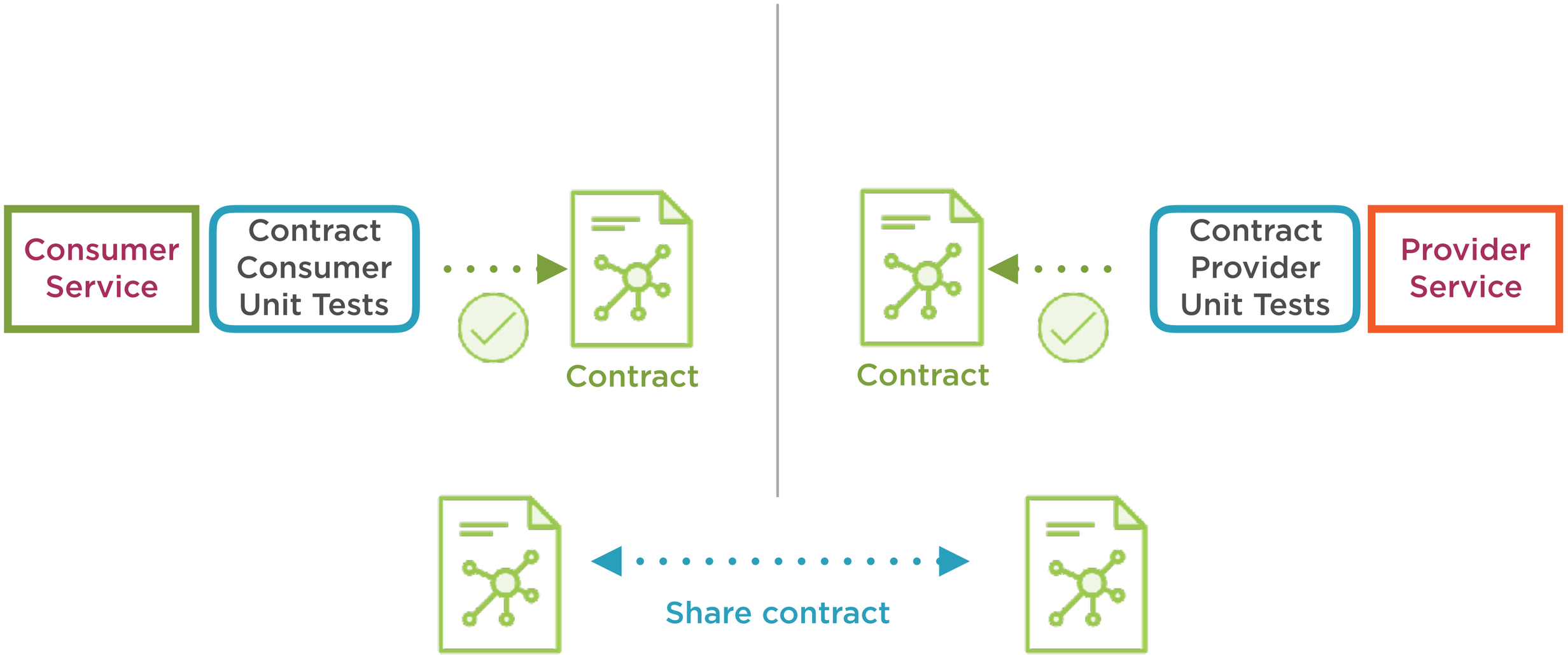
Component testing

Contract testing

Integration testing

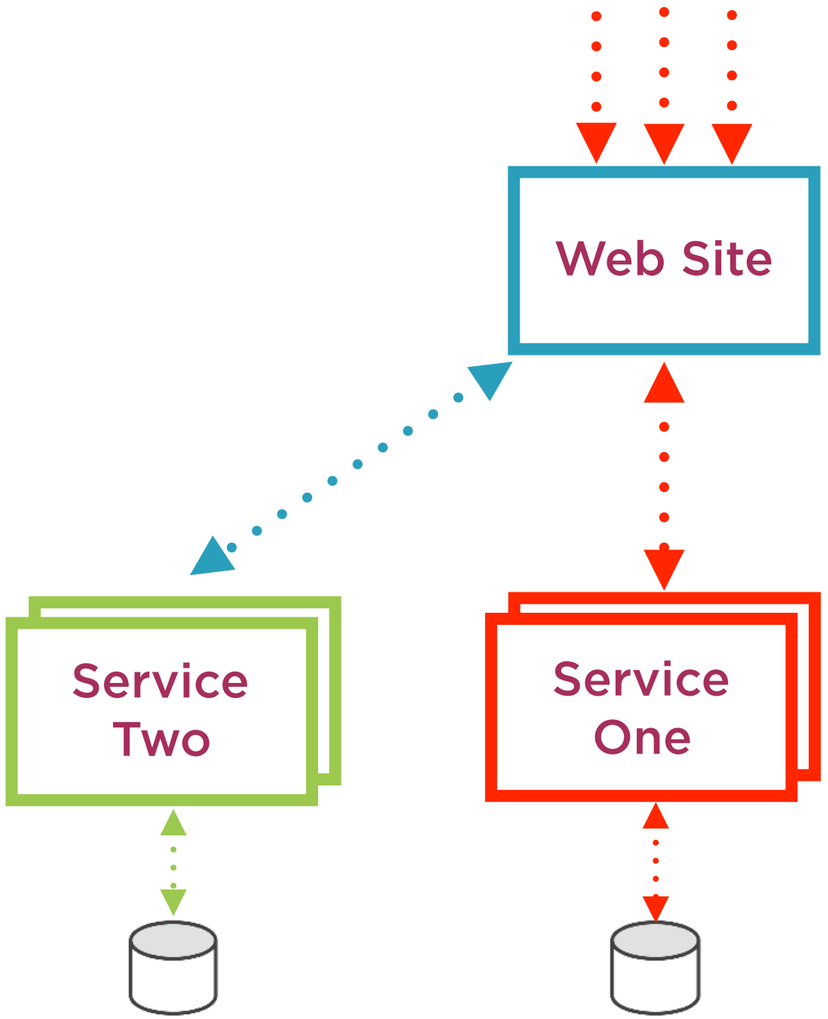
Unit testing

Contract Testing High-Level



Versioning Strategies

Versioning Strategy



Versioning identification pattern

Canonical versioning pattern

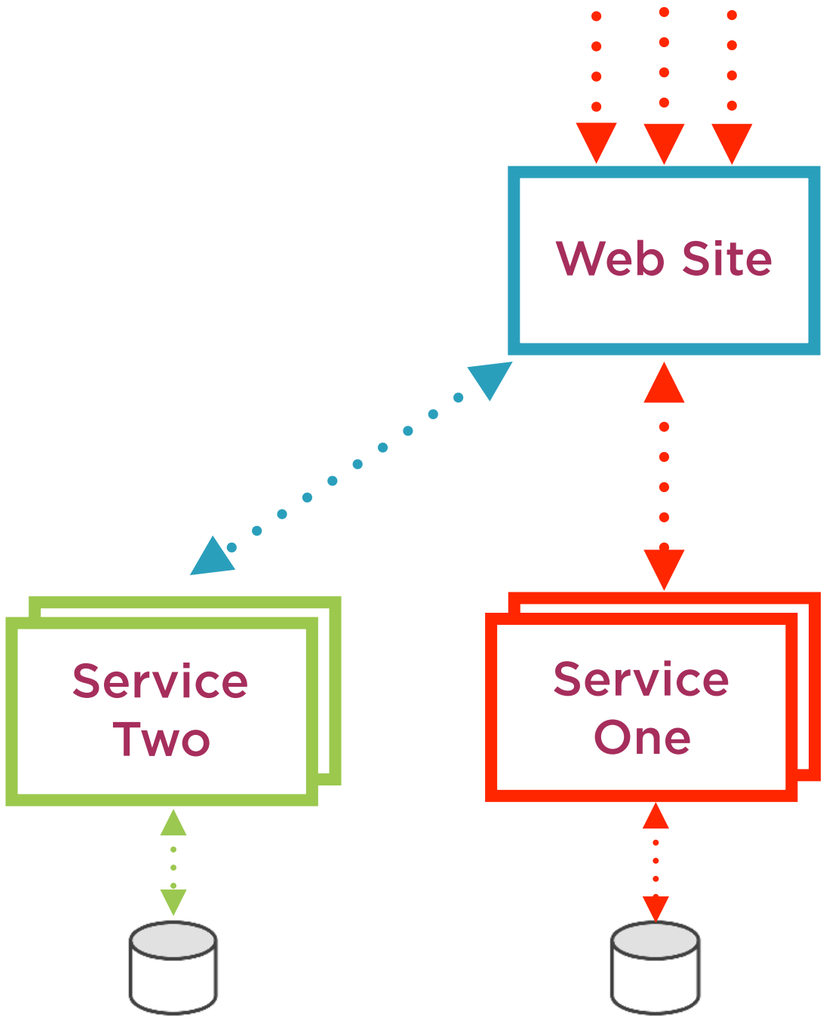
How

- HTTP headers
- Request path

Format

- Consecutive numbers for endpoints
- Dates
- Semantic versioning. For example 2.0.1
  - Major, minor and patch

HTTP Headers



HTTP headers to specify versions

- Accept header used for the request
- Content-type header used for the response

Client uses accept header

Accept: application/myApi.net; **version=2**

Server response includes resource version

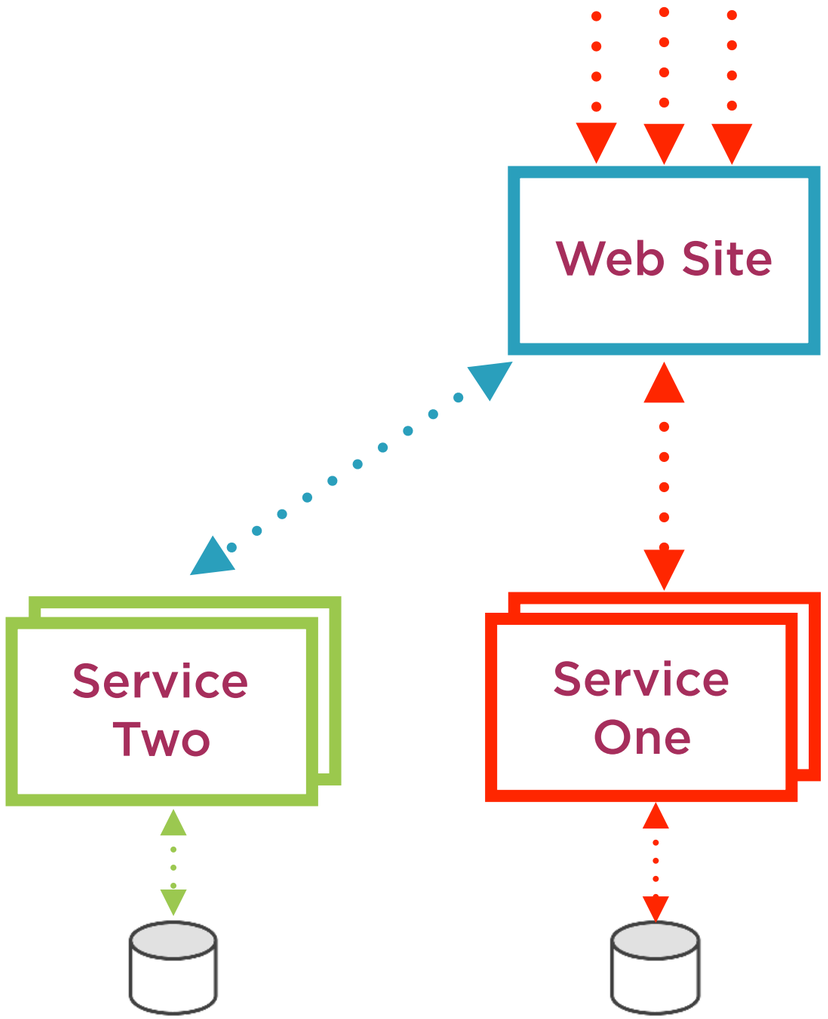
Content-type: application/myApi.net; **version=2**

Alternatively, use custom app specific headers

Extra tools required for testing

Not as convenient as request path versioning

Request Path



More convenient than HTTP headers

Can be used with HTTP headers

Consecutive numbers within request path

https://myApi.net/**v2**/order

Alternatively, dates within the request path

https://myApi.net/**v20190404**/order

Query parameter as an alternative approach

https://myApi.net/order?**version=2**

Previous versions can carry on working

https://myApi.net/**v1**/order

Summary

Introduction

Awareness of Compatibility

Testing Compatibility

Versioning Strategies

Microservices Architectural Design Patterns Playbook