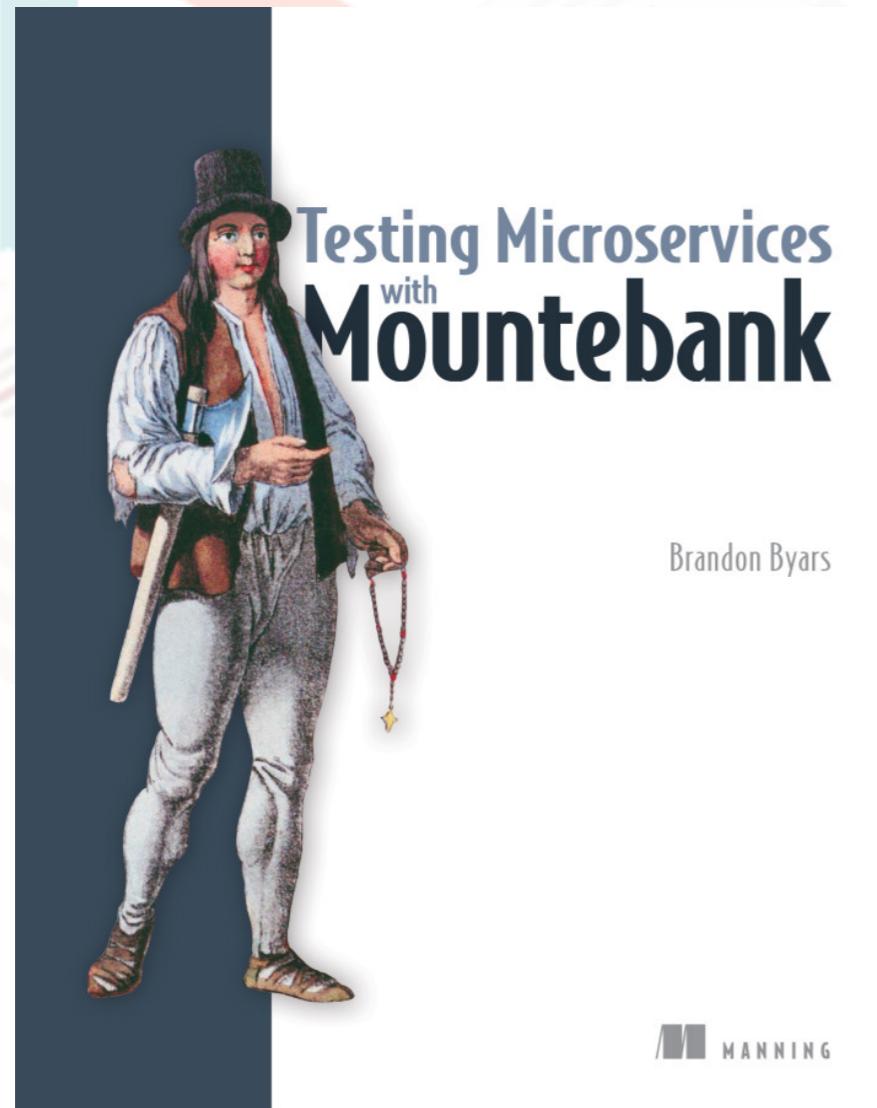


จำลองระบบเพื่อใช้ในการทดสอบ ด้วย

Mountebank 101



- Hello, World! Mountebank
- Predicates
- Record/Replay
- Behavior and Programming mountebank
- Adding Behaviors



Day2: Predicates

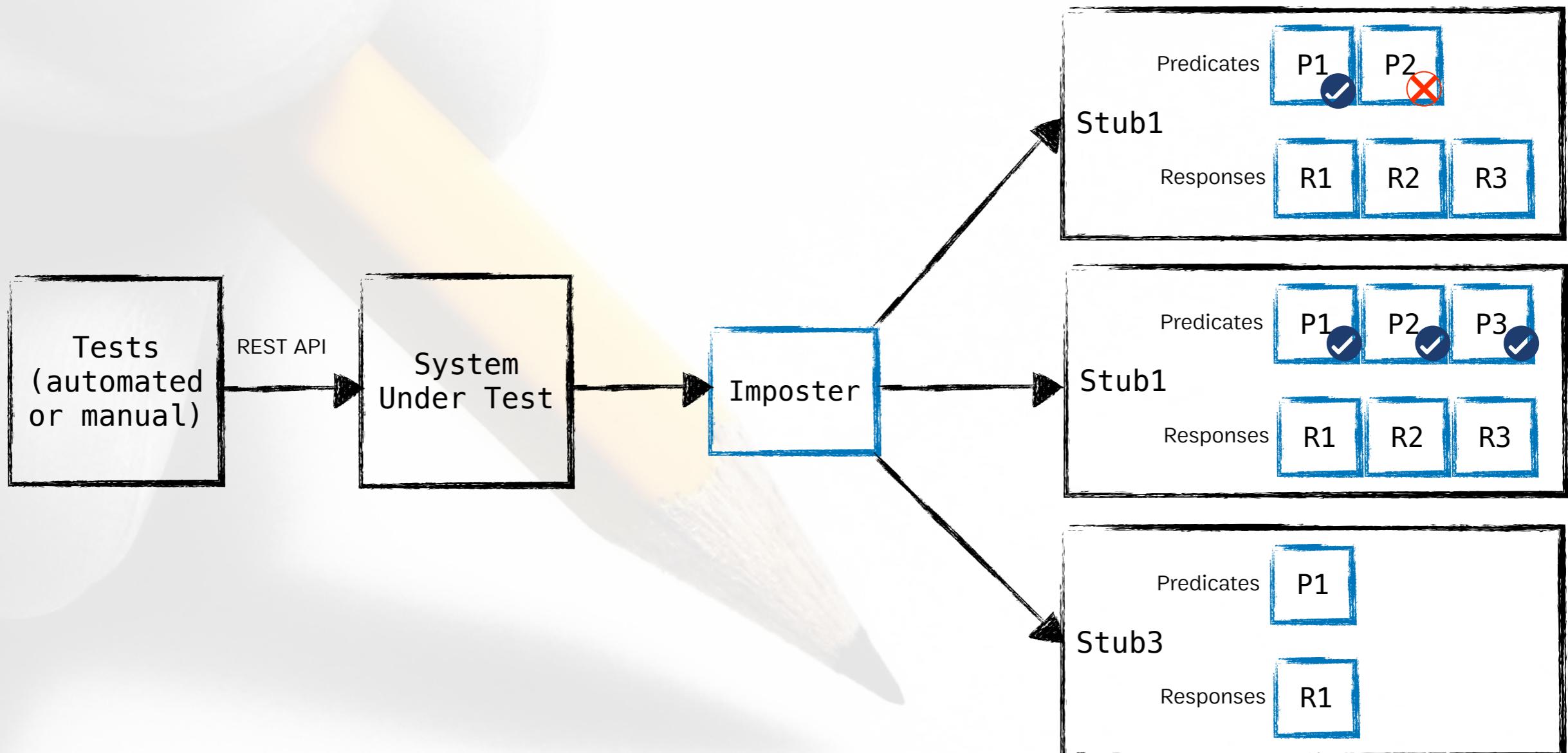


The Basic of Predicates

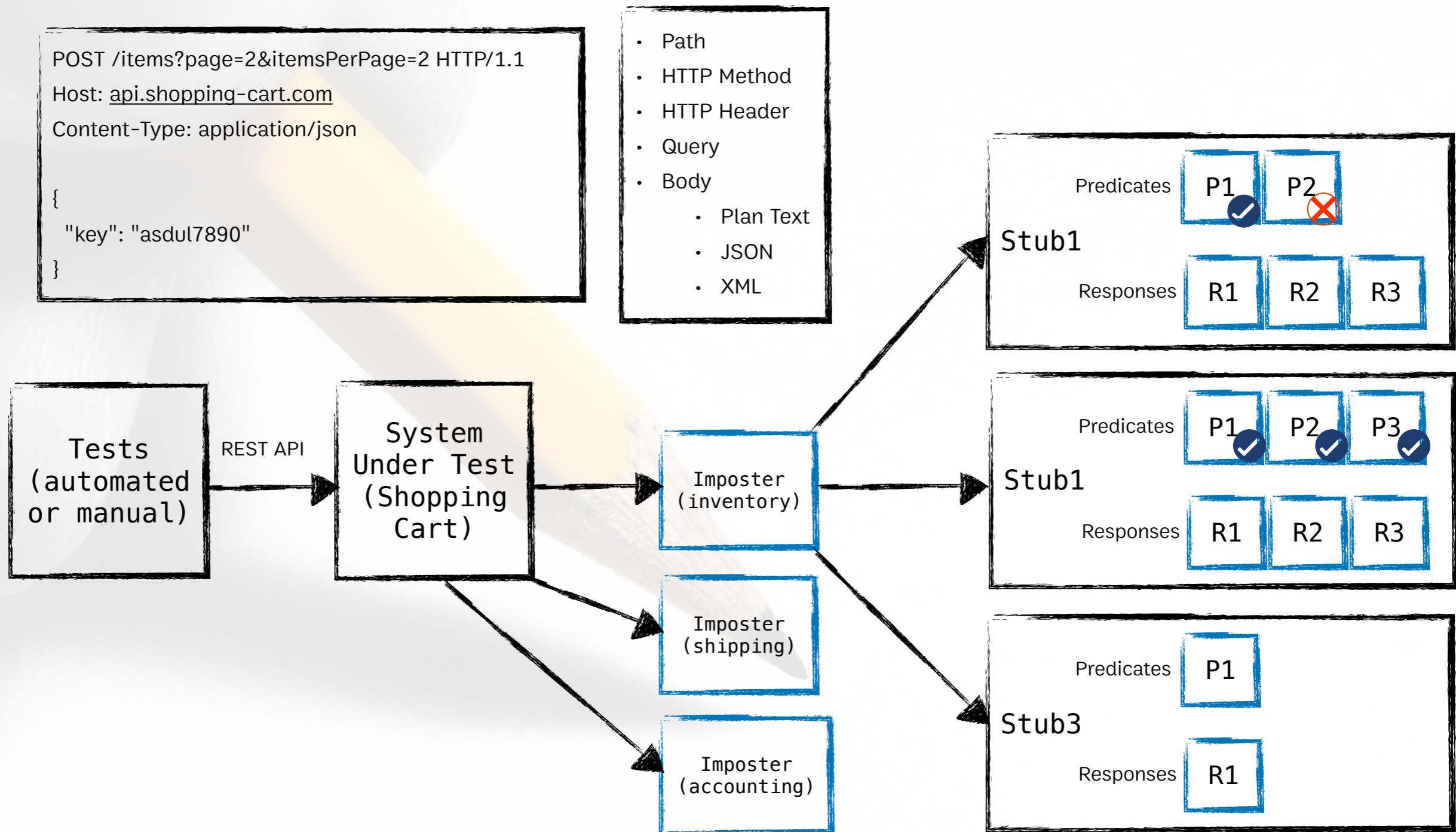
- Using predicates to send different responses for different requests
- Simplifying predicates on JSON request bodies
- Using XPath to simplify predicates on XML request bodies



How Mountebank matches the request against each stub's predicates?



The HTTP Shopping Cart



Type of Predicates



The Matches Predicates

Imposter: matches-predicate.json

```
{  
  "predicates": [  
    { "equals": { "method": "POST" }},  
    { "startsWith": { "body": "smart" }},  
    { "endsWith": { "body": "robot" }}  
,  
  "responses": [  
    {  
      "is": {  
        "body": "Hello, smart robot"  
      }  
    }  
  ]  
}
```

HTTP Method

POST

URL

<http://localhost:3000/items>

Body

smart alpha3 robot



Matching any identifier on the path

Imposter: path-identifier-predicate.json

```
{  
  "protocol": "http",  
  "port": 3000,  
  "stubs": [  
    "predicates": [  
      "matches": { "path": "/items/\\d+" }  
    ],  
    "responses": [  
      "is": {  
        "statusCode": 200,  
        "headers": { "Content-Type": "text/plain" },  
        "body": {  
          "name": "43 Piece Dinner Set",  
          "price": 12.95,  
          "quantity": 10  
        }  
      }  
    ]  
  ]  
}
```

Postman: request with

HTTP Method	GET
URL	http://localhost:3000/items/1

The metacharacters used here, \d+, represent one or more digits, so the pattern will match /items/123 and /items/2 but not items/robot.



Matching object request fields

Imposter: request-fields-predicate.json

```
{  
  "predicates": [  
    {"equals": {  
      "query": { "q": "robot" }  
    }  
  ],  
  "responses": [  
    {"is": {  
      "statusCode": 200,  
      "headers": { "Content-Type": "text/plain" },  
      "body": {  
        "items": [  
          {  
            "name": "TA Robot",  
            "price": 109.99,  
            "quantity": 50  
          },  
          {  
            "name": "AlphaBot2",  
            "price": 71.0,  
            "quantity": 73  
          }  
        ]  
      }  
    }  
  ]  
}
```

Postman: request with

```
HTTP Method  
GET  
  
URL  
http://localhost:3000/items?q=robot
```



The Deep Equals Predicate

Imposter: deep-equals-predicate.json

```
{  
  "predicates": [  
    {"deepEquals": {  
      "query": { "q": "robot", "page": 1 }  
    }  
  ],  
  "responses": [  
    {"is": {  
      "statusCode": 200,  
      "headers": { "Content-Type": "text/plain" },  
      "body": {  
        "items": [  
          {  
            "name": "TA Robot",  
            "price": 109.99,  
            "quantity": 50  
          },  
          {  
            "name": "AlphaBot2",  
            "price": 71.0,  
            "quantity": 73  
          }  
        ]  
      }  
    }  
  ]  
}
```

Postman: request with

```
HTTP Method  
GET  
  
URL  
http://localhost:3000/items?q=robot&page=1
```

With this predicate, a query string of ?
q=robot&page=1 would match, but a query string
of ?q=robot&page=1&sort=desc wouldn't.



Matching Multivalued Fields

Imposter: multivalued-field-predicate.json

```
{  
  "predicates": [  
    {"deepEquals": {  
      "query": { "q": ["smart", "robot"] }  
    }  
  ]  
}
```

Requires exact match

```
{  
  "predicates": [  
    {"equals": {  
      "query": { "q": ["smart", "robot"] }  
    }  
  ]  
}
```

Requires these elements to be present

Postman: request with

HTTP Method

GET

URL

<http://localhost:3000/items?q=smart&q=robot>



The Exists Predicate

Imposter: exists-predicate.json

```
{  
  "predicates": [  
    {  
      "exists": {  
        "headers": { "Authorization": false }  
      }  
    },  
    {  
      "responses": [  
        {  
          "is": { "statusCode": 401 }  
        }  
      ]  
    }  
  ]  
}
```

Postman: request with

HTTP Method

GET

URL

http://localhost:3000/items



Conjunction Junction

Imposter: conjunction-predicate1.json

```
{  
  "predicates": [  
    { "startsWith": { "body": "smart" }},  
    { "endsWith": { "body": "robot" }}  
  ]  
}
```

without conjunction

Imposter: conjunction-predicate2.json

```
{  
  "predicates": [  
    {  
      "and": [  
        { "startsWith": { "body": "smart" }},  
        { "endsWith": { "body": "robot" }}  
      ]  
    }  
  ]  
}
```

with conjunction

HTTP Method

POST

URL

<http://localhost:3000/items>

Body

smart alpha3 robot



A Complete List of Predicate Types

OPERATOR	DESCRIPTION
equals	Requires the request field to equal the predicate value
deepEquals	Performs nested set equality on object request fields
contains	Requires the request field to contain the predicate value
startsWith	Requires the request field to start with the predicate value
endsWith	Requires the request field to end with the predicate value
Matches	Requires the request field to match the regular expression provided as the predicate value
exists	Requires the request field to exist as a nonempty value (if true) or not (if false)
not	Inverts the subpredicate
or	Requires any of the subpredicates to be satisfied
and	Requires all of the subpredicates to be satisfied



Parameterizing Predicates



Making Case-Sensitive Predicates

Imposter: case-sensitive-predicate.json

```
{  
  "predicates": [  
    "equals": {  
      "query": { "q": "robot" }  
    },  
    "caseSensitive": true  
  ],  
  "responses": [  
    "is": {  
      "statusCode": 200,  
      "headers": { "Content-Type": "text/plain" },  
      "body": {  
        "items": [  
          {  
            "name": "TA Robot",  
            "price": 109.99,  
            "quantity": 50  
          },  
          {  
            "name": "AlphaBot2",  
            "price": 71.0,  
            "quantity": 73  
          }  
        ]  
      }  
    }  
  ]  
}
```

Predicates are case-insensitive by default.

Postman: request with

```
HTTP Method  
GET  
  
URL  
http://localhost:3000/items?q=robot
```



Using Predicates on JSON Values



Using Direct JSON Predicates

Imposter: direct-json-predicate.json

```
{  
  "predicates": [  
    {  
      "equals": {  
        "body": {  
          "name": "riderX"  
        }  
      }  
    }  
  ],  
  "responses": [  
    {  
      "is": {  
        "statusCode": 201,  
        "headers": { "Location": "/items/123" }  
      }  
    }  
  ]  
}
```

Postman: request with

HTTP Method	POST
URL	http://localhost:3000/items
HEADER	"Content-Type": "application/json"
Body	{ "name": "riderX", "price": 100, "quantity": 50 }



Selecting a JSON Value with JSONPath

Imposter: select-a-json-value-with-jsonpath-predicate.json

```
{  
  "predicates": [  
    { "equals": { "method": "POST" } },  
    { "equals": { "path": "/shipping" } },  
    {  
      "jsonpath": {  
        "selector": "$.items[(@.length-1)].location"  
      },  
      "equals": { "body": "Covid-19 Zone" }  
    },  
  ],  
  "responses": [{ "is": { "statusCode": 400 } }]  
}
```

Postman: request with

HTTP Method
POST

URL
<http://localhost:3000/shipping>

HEADER
"Content-Type": "application/json"

Body

```
{  
  "items": [  
    {  
      "name": "TA Robot",  
      "price": 109.99,  
      "quantity": 50,  
      "location": "Bangkok"  
    },  
    {  
      "name": "AlphaBot2",  
      "price": 71.0,  
      "quantity": 73,  
      "location": "Covid-19 Zone"  
    }  
  ]  
}
```

ref: [JSONPath syntax](#)



Using Predicates on XML Values



Selecting a XML Value with XPath

Imposter: select-a-xml-value-with-xpath-predicate.json

```
{  
  "predicates": [  
    { "equals": { "method": "POST" } },  
    { "equals": { "path": "/shipping" } },  
    {  
      "xpath": {  
        "selector": "//item[@name='AlphaBot2']/location"  
      },  
      "equals": { "body": "Covid-19 Zone" }  
    }  
  ],  
  "responses": [{ "is": { "statusCode": 400 } }]  
}
```

Postman: request with

HTTP Method
POST

URL
http://localhost:3000/shipping

HEADER
"Content-Type": "application/json"

Body

```
<items>  
  <item name="TA Robot">  
    <price>109.99</price>  
    <location>Bangkok</location>  
  </item>  
  <item name="AlphaBot2">  
    <price>71.50</price>  
    <location>Covid-19 Zone</location>  
  </item>  
</items>
```



Setup Node.js(Windows)

Downloads

Latest LTS Version: 12.16.1 (includes npm 6.13.4)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.



[Windows Installer \(.msi\)](#)

[Windows Binary \(.zip\)](#)

[macOS Installer \(.pkg\)](#)

[macOS Binary \(.tar.gz\)](#)

[Linux Binaries \(x64\)](#)

[Linux Binaries \(ARM\)](#)

[Source Code](#)

32-bit	64-bit
32-bit	64-bit
64-bit	
64-bit	
64-bit	
ARMv7	ARMv8
node-v12.16.1.tar.gz	

- Goto [Node.js downloads](#) and download Node.js for Windows
 - LTS
 - Current



Run the Node.js Installer(Windows)

- Welcome to the Node.js setup wizard
 - Select **Next**
- End-User License Agreement (EULA)
 - Check **I accept the terms in the License Agreement**
 - Select **Next**
- Destination Folder
 - Select **Next**
- Custom Setup
 - Select **Next**
- Ready to install Node.js
 - Select **Install**
 - *Note:* This step requires Administrator privileges.
 - If prompted, authenticate as an Administrator.
- Installing Node.js
 - Let the installer run to completion.
- Completed the Node.js Setup Wizard
 - Click **Finish**



Setup Node.js(Mac)

Install Homebrew

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"
```

Install Node.js

```
brew install node
```



Verify That Node.js was Properly Installed and Install Mountebank

Run command

```
node -v
```

Update npm

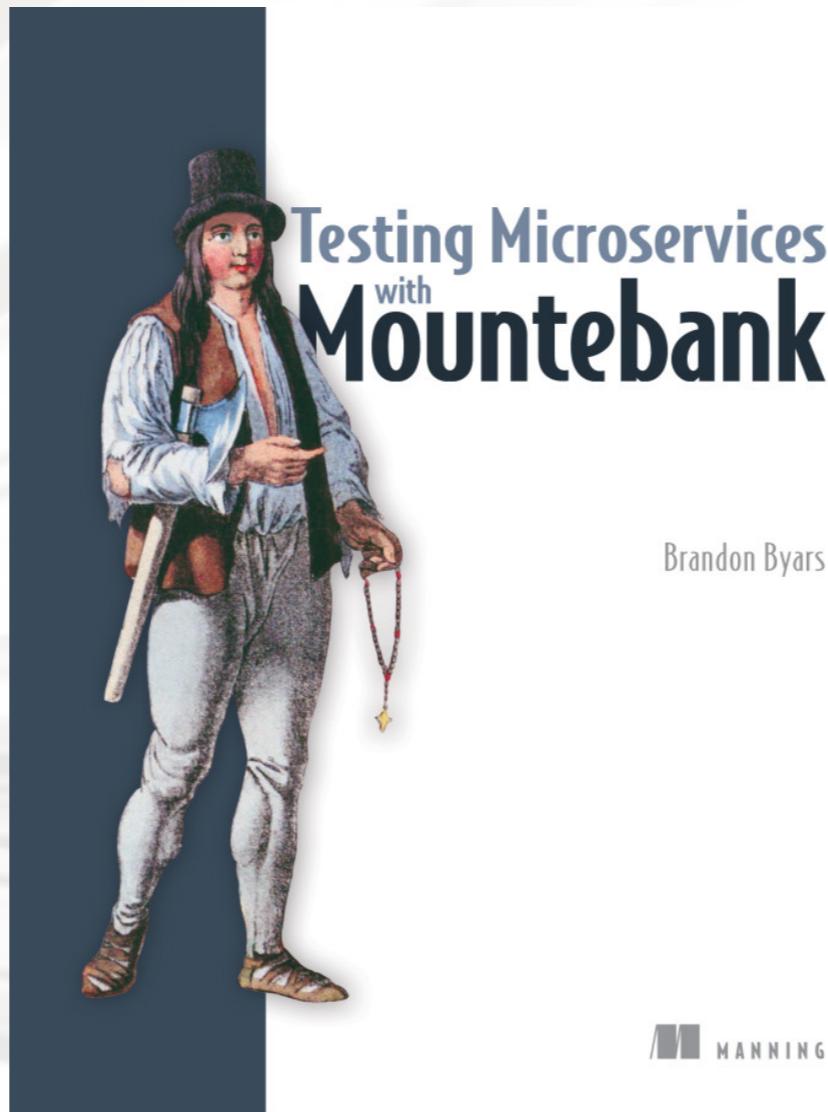
```
npm install npm -g
```

Install mountebank

```
npm install mountebank -g
```



Books to Read and Practice



<https://www.manning.com/books/testing-microservices-with-mountebank>



บริษัท สยามชำนาญกิจ จำกัด และเพื่อนพ้องน้องพี่