Google Cloud

Edge API Lifecycle and Tools

Testing & Mocking

Static Code Analysis

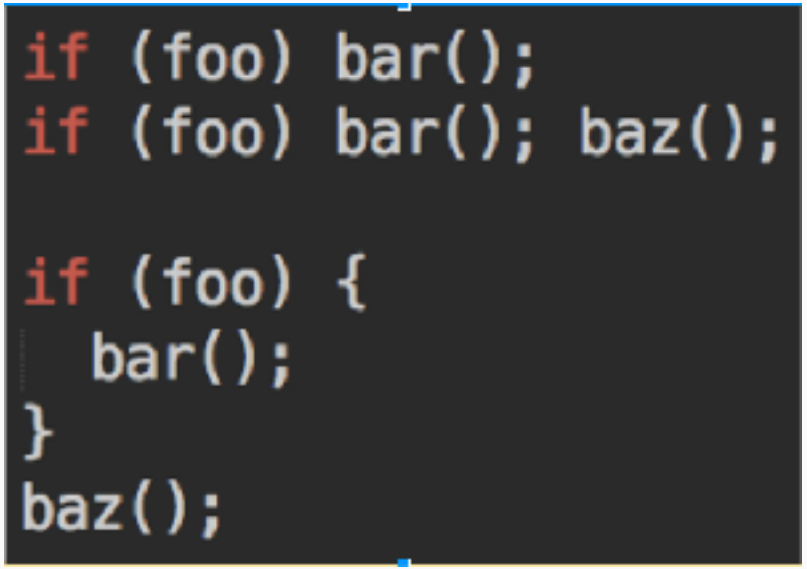
Static Code Analysis

- We have the functionality to attach custom code to any flow within Edge. These can be written in Java, JS, Python and Node.js.
- Recommendation is to run static code analysis for these custom code in Edge proxy.
- Promote consistency and follow best practices for the language you are using.

Automation

- Integrate with your editor
- Use hooks with source control
- Use Grunt/Gulp watcher
- Use continuous integration build systems

Static Code Analysis



```
if (foo) bar();
if (foo) bar(); baz();

if (foo) {
  bar();
}
baz();
```

Static Code Analysis

```
switch (new Date().getDay()) {
    case 0:
        day = 'sunday';
    case 1:
        day = 'monday';
    case 2:
        day = 'tuesday';
    case 3:
        day = 'wednesday';
    case 4:
        day = 'thursday';
    case 5:
        day = 'friday';
    case 6:
        day = 'saturday';
}
```

Testing

Unit Testing

API Proxy Testing

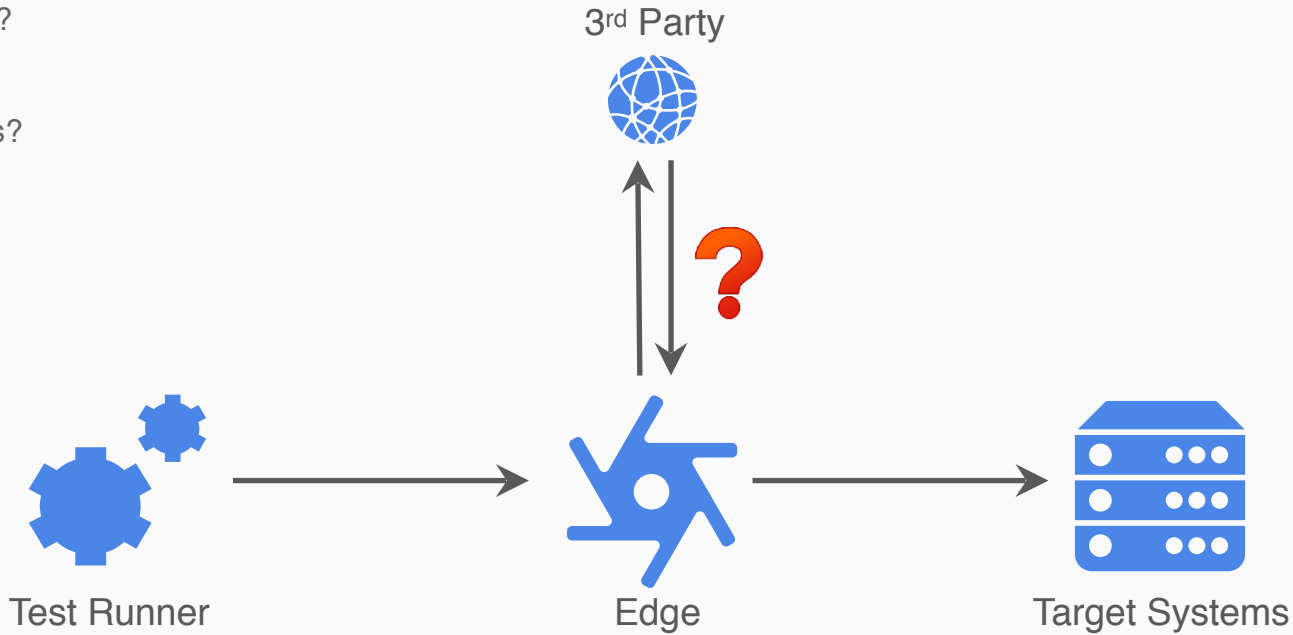
Do you think it is possible to test an API proxy fully with integration testing?

Integration Testing Enough?

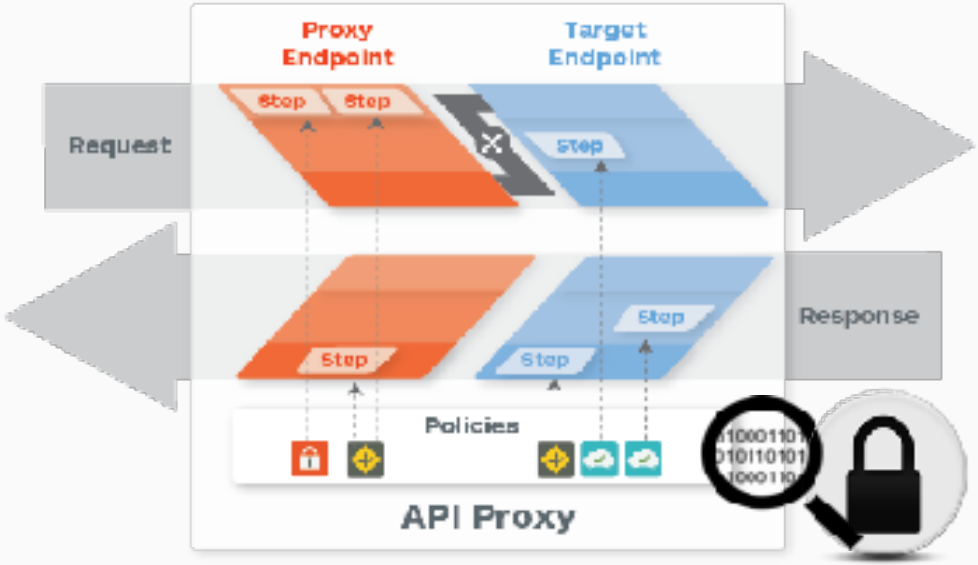
Service Callouts?

Script Callouts?

async operations?



Testing in Isolation



Unit Testing Benefits

- Code can be tested locally without deployment to Edge first
- Can create hooks to enforce testing during commit
- Much faster than integration testing

Unit Testing – Boundary Principal

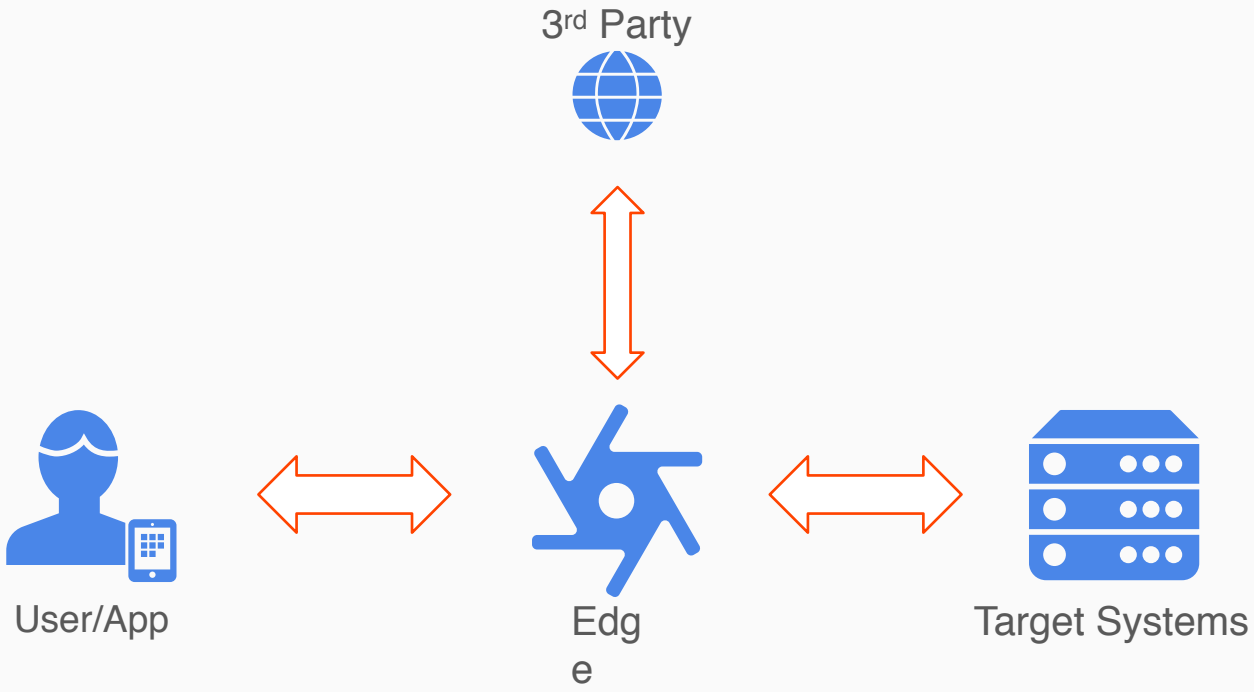
Test within your boundaries – don’t test libraries you don’t control

Unit Testing – Types of policies

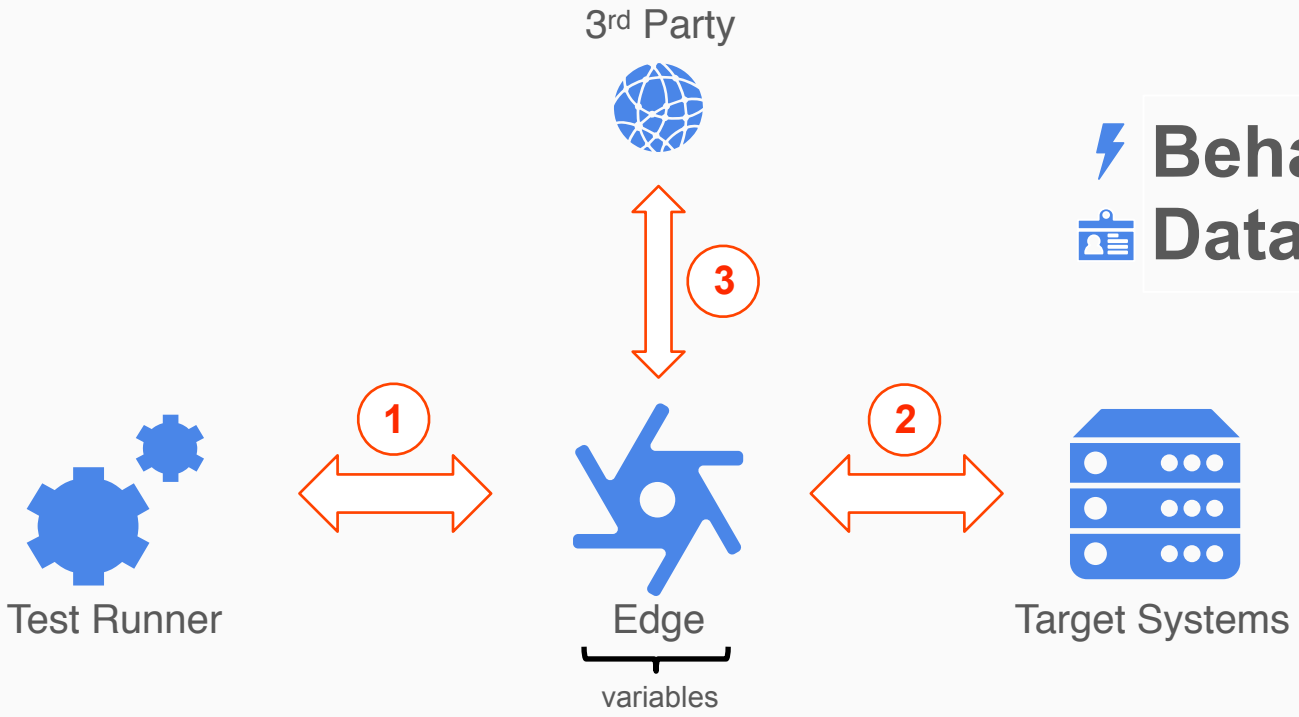
Traffic management policies	Mediation policies	Security policies	Extension policies
• Cache policies	• Access Entity policy	• Access Control policy	• Java Callout policy *
• Concurrent Rate Limit policy	• Assign Message policy	• Basic Authentication policy	• JavaScript policy
• Quota policy	• Extract Variables policy	• JSON Threat Protection policy	• Message Logging policy
• Reset Quota policy	• JSON to XML policy	• LDAP policy *|	• Python Script policy *
• Spike Arrest policy	• Key Value Map Operations policy	• OAuth v2.0 policies	• Service Callout policy
	• Raise Fault policy	• OAuth v1.0a policy	• Statistics Collector policy
	• SOAP Message Validation policy	• Regular Expression Protection policy	
	• XML to JSON policy	• SAML Assertion policies	
	• XSL Transform policy	• Verify API Key policy	
		• XML Threat Protection policy	

Integration Testing

What to Test



What to Test



Consistent Data

- Can we mock responses at specific points?
- Can we do string matching or regex in tests?
- Can we recreate data in target systems?

Consistent Data

- Asserting functional is *easier* than consistent data!
  - Normal API operations
  - OAuth handshake (esp. authorization code grant type)
- What about unexpected error conditions?
  - Timeouts
  - 500 server unavailable
- What about non-functional tests?
  - Caching
  - Traffic management (quota, spike)
  - Security (JSON/XML threat protection)

API Testing from UI

- This is highly not recommended.
- This tests the application – not the APIs.
- Cannot sufficiently verify all functional paths for the entire API resource space.
- Test needs to change when UI changes which is much more frequent than API changes.
- The API team needs to be responsible for API testing.
- There can be a lot of API clients.

Integration Testing Disadvantages

- Deployment to Edge is required. Another option is OPDK if you have access to it, but need to keep configurations in sync.
- It will be SLOW – especially compared to unit testing. Consider deployment time, network time, data size variations, and other variables.

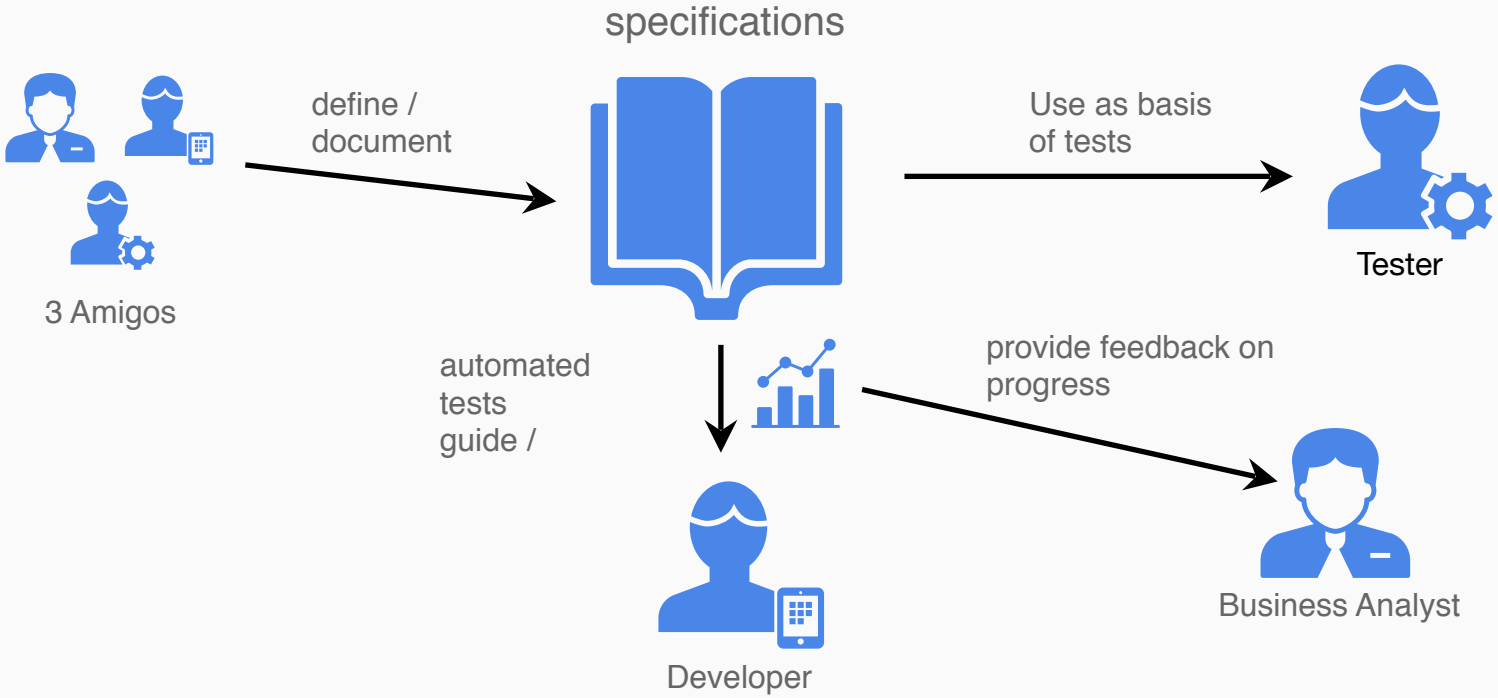
Behaviour Driven Development

Behavior Driven Development

BDD is a software development process with TDD in its heart

BDD – a software development process

BDD – a software development process



Behavior Driven Development

Non (less?) technical test script

Ideally does not require technical skills

Closer alignment to the business

Still requires...

- Knowledge of the data
- Test centric mind set
  - Edge Cases
  - Error Scenarios

```
Feature: Delete time entry
    As an employee
    I want to delete an existing time entry
    So that I can remove an invalid/incorrect time entry from my timesheet

    Scenario: Delete time entry with write scope
        Given I have a valid access token with write scope
        And I have an existing time entry
        When I delete that time entry
        Then that time entry should not exist

    Scenario: Delete time entry with read scope
        Given I have a valid access token with read scope
        And I have an existing time entry
        When I delete that time entry
        Then I should get an error with message "you don't have enough permissions to perform this
        operation" and code "400.02.003"

    Scenario: Delete time entry with invalid access token
        Given I have an invalid access token
        And I have an existing time entry
        When I delete that time entry
        Then I should get an error with message "access token is invalid" and code "400.02.001"

    Scenario: Delete time entry with expired access token
        Given I have an expired access token
        And I have an existing time entry
        When I delete that time entry
        Then I should get an error with message "access token has expired" and code "400.02.002"

    Scenario: Delete non-existing time entry
        Given I have a valid access token with write scope
        When I delete time entry with id "non-existing"
        Then I should get an error with message "time entry coesn't exist" and code "404.02.001"

    @mock
    Scenario: SAP error
        Given I have a valid access token with write scope
        And I have an existing time entry
        And SAP is broken
        When I delete that time entry
        Then I should get an error with message "internal server error" and code "500.02.001"
```

BDD – Specifications

Express using examples

Describe context, trigger and expected behaviour

Prioritize behaviours with business value

Integration Testing – Disadvantages

Deployment to Apigee is required. Other option – OPDK if you have access to it but need to keep configurations in sync.

It will be “SLOW” – especially compared to unit testing

Deployment time, network time, data sizes

Mocking

Benefits

- Workaround target API availability issues
  - Network, ops, deployment, migrations, patches
  - Parallel development
- Workaround constantly changing/unpredictable data
- Simulate certain scenarios for testing
- Easy testing when tests rely on previous data population
  - e.g. forgotten password
- Improves test execution speed for high latency targets

Implementation

- API proxy – respond using policies
- API proxy with Node.js
- apimocker (node.js) - https://github.com/apigeecs/apigee-apimocker
- Amok - https://github.com/sauliuz/amok/tree/master/examples/apigee-amok
- import OpenAPI spec and download server code from http://editor.swagger.io/#/
- other tools:
  - Mock Server
  - Wire Mock

Lab

https://github.com/apigeecs/apigee-apimocker/tree/master/shop

This includes mocking using apigee-mocker and also BDD using apickli (https://github.com/apickli/apickli)

THANK YOU