CONTINUOUS INTEGRATION
THE VIRTUOUS CYCLE

Ozan Seymen – Apigee Principal Architect
Diego Zuluaga – Apigee Principal Architect

apigee

The ultimate goal of CI is Faster Time to Market TTM

Reality Check: code that doesn't go to production...

Why is it so hard?

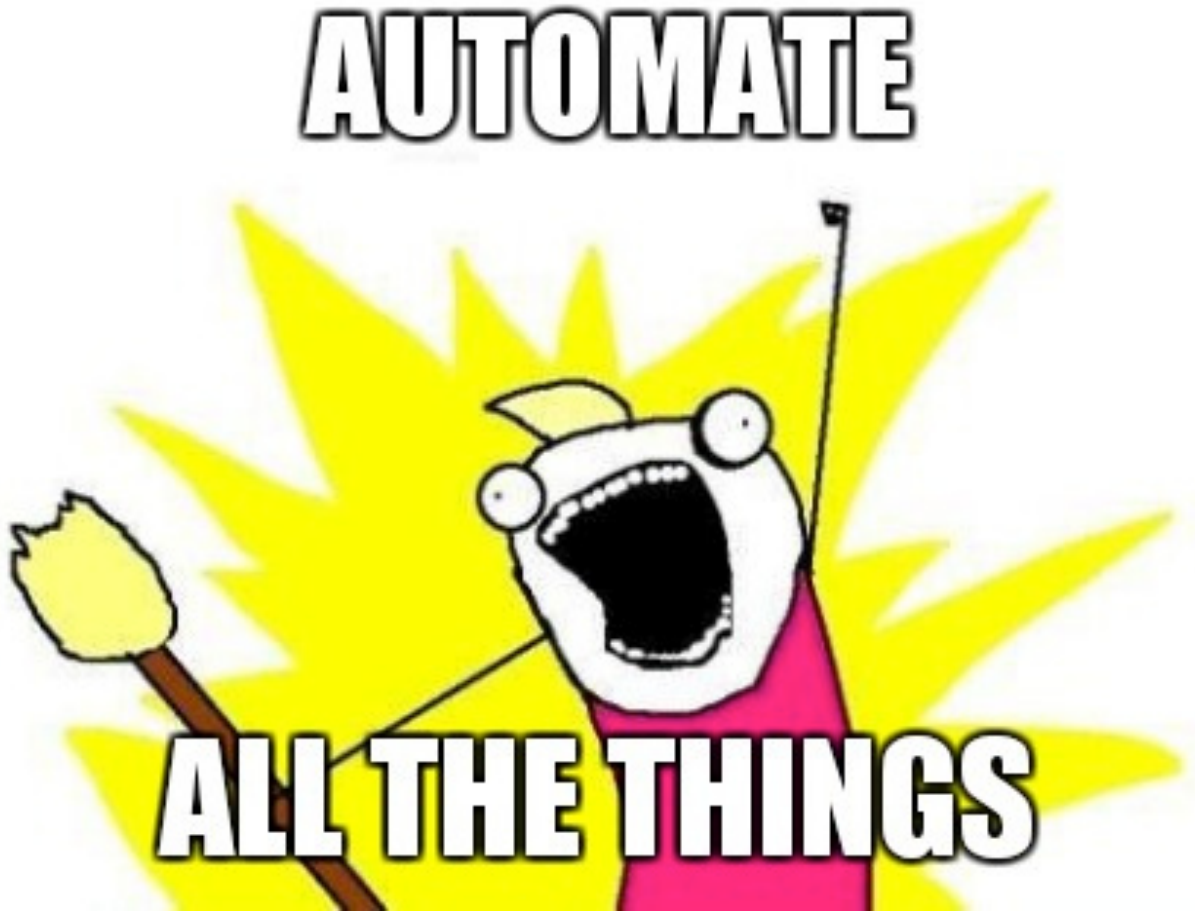
Let's take a look at the tasks required:

- Clean API Bundle Files and Folders
- Copy Artifacts
- Configure DEV, QA, STG, PROD, etc.
- Package Artifacts (zip)
- Deactivate last revision
- Import and deploy API bundle
- Execute Unit Tests
- Execute Functional Tests
- Execute Performance Tests
- Deploy and Test Documentation on CMS
- Any other manual tasks…

There must be a way to automate all this tedious work!

How do we solve this?

A common reaction is...



Fortunately, there's a solution for that

Who's doing it? and Why?

- Experience has taught us that developers and API Teams prefer:
  - Short development iterations and fast feedback
  - Battle-tested frameworks embraced by open communities
  - Developer-friendly scripting languages
  - Frameworks that easily integrate with CI infrastructure

Who's doing it? and Why?

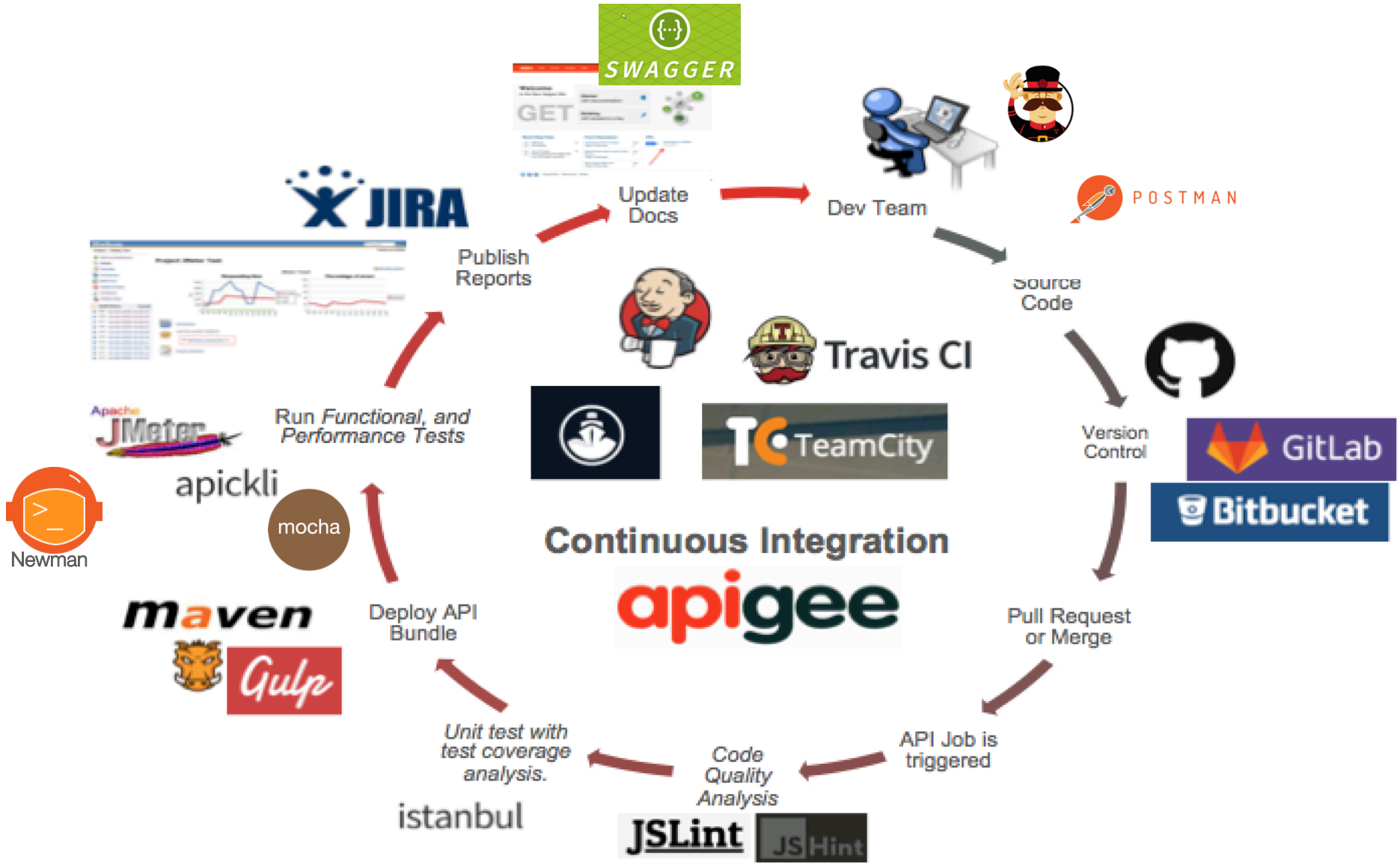
Customers: BBC WorldWide, Telefonica Chile, AT&T, T-Mobile, Everything Everywhere, Autodesk, Intralinks, L.L.Bean, FX Networks, Morrisons, Equinix, Cambia, Waitrose, Millicom (Tigo), GLH Hotels, among others.

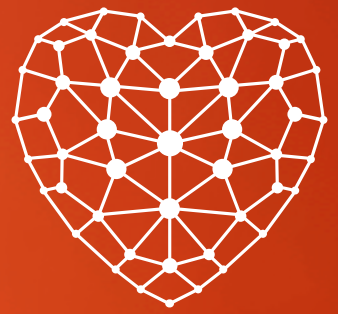
Trivia…

How many weeks in average would take to pump code to prod? For API Teams under Apigee Blueprint Program

Apigee has enabled companies releasing code to prod on a daily basis instead of every six months without losing any traffic!

Enter The Virtuous Cycle

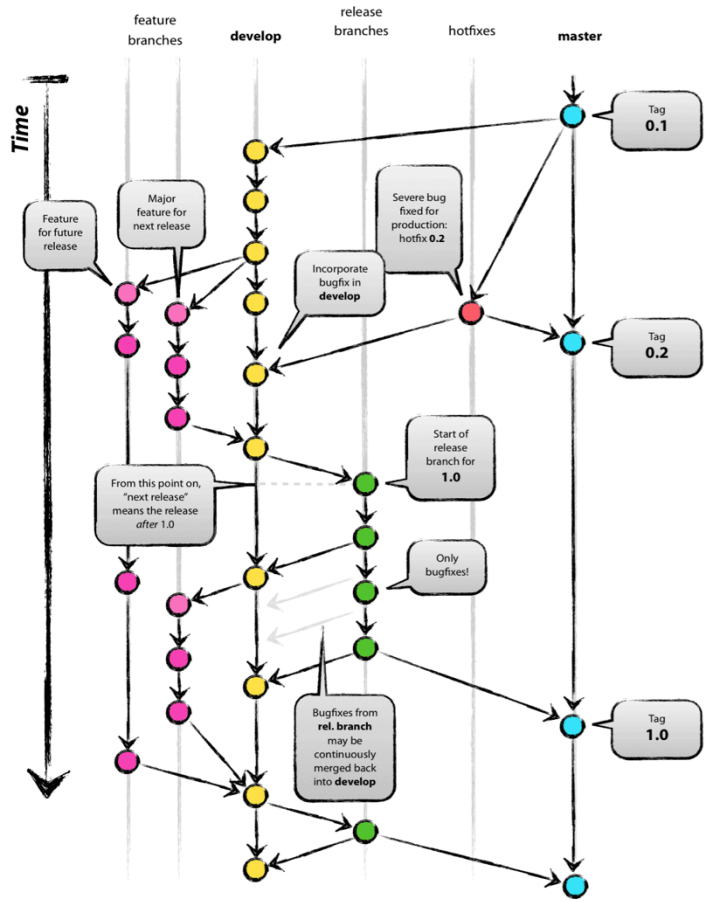




Version Control

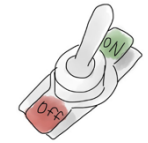
Version Control: You need a winning branching Model!

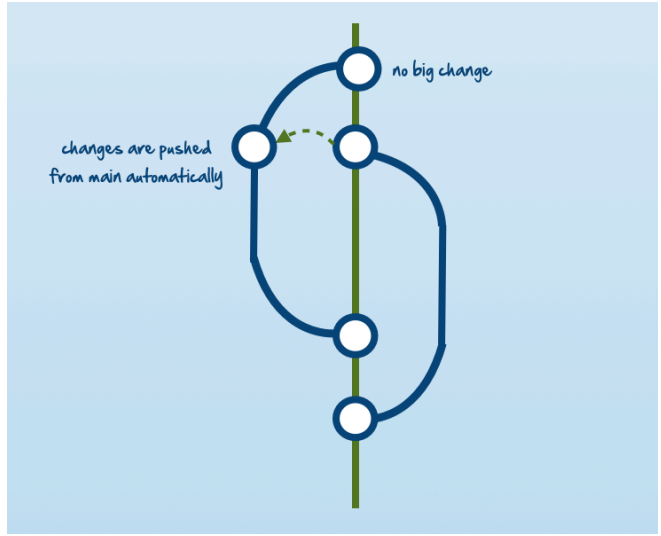
Gitflow

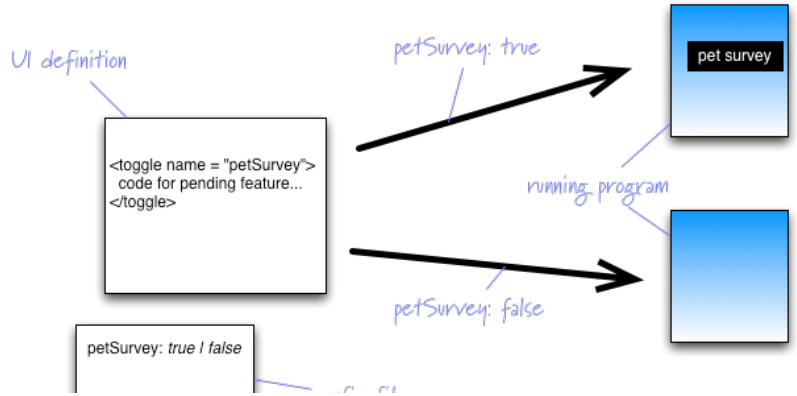


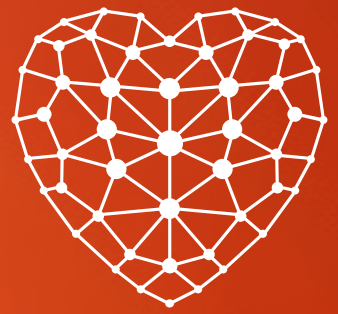
and/or

feature on









Code Quality Analysis

Static Code Quality Analysis

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

Static Code Quality Analysis

AUTOMATE!

- Integrate with your editor
- Hooks for source control
- Grunt/gulp watcher
- Continuous integration/build systems

Static Code Quality Analysis

JSHint

FindBugs™
because it's easy

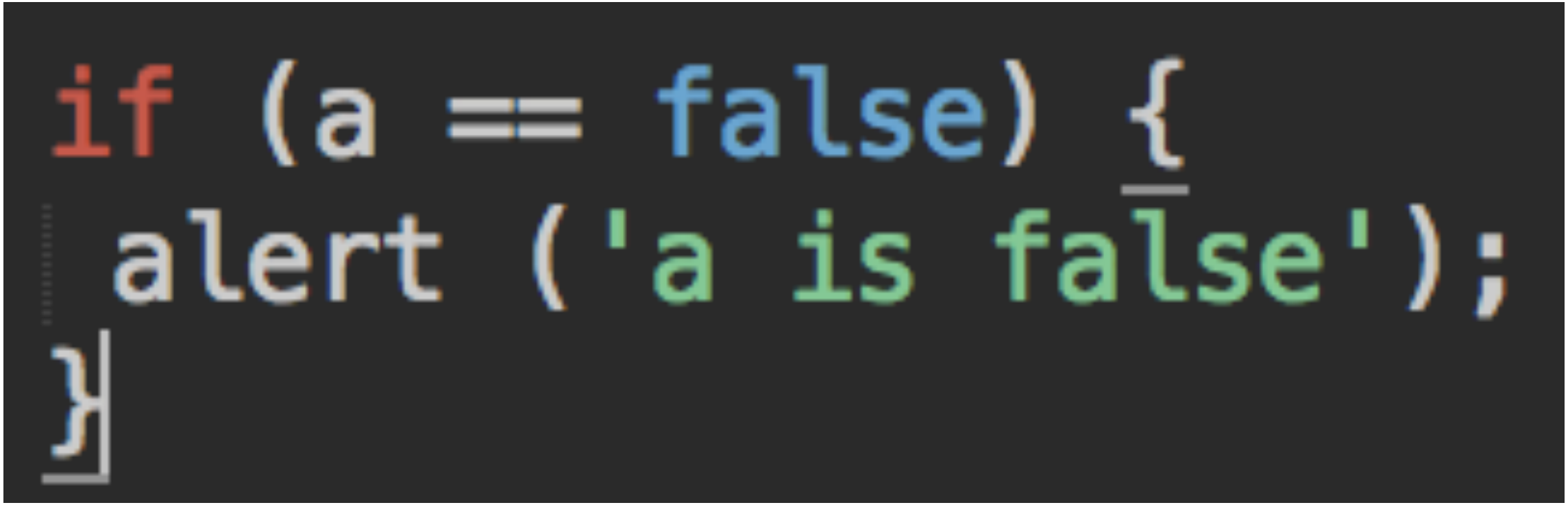
LINT4J

Pylint
Star your Python code!

checkstyle

Static Code Quality Analysis

Static Code Quality Analysis



```
if (a == false) {
  alert ('a is false');
}
```

Static Code Quality Analysis

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

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

Static Code Quality Analysis

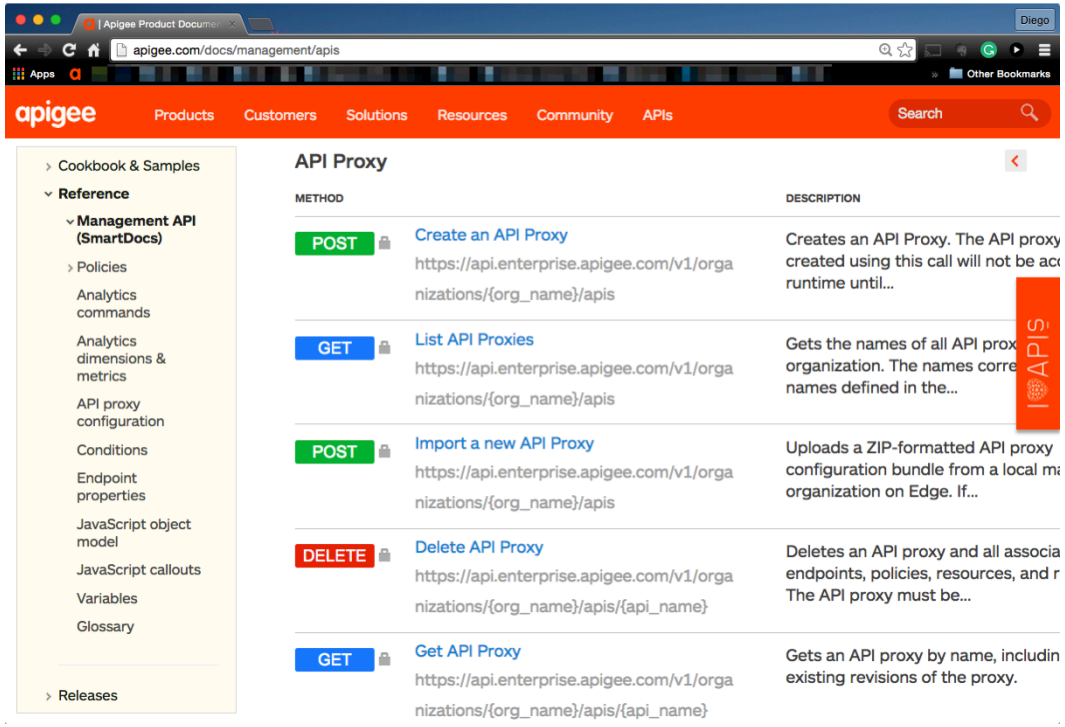
```
switch (new Date().getDay()) {
    case 0:
        day = 'sunday';
    case 1:
        day = 'monday';
    case 2:
        day = 'tuesday';
    case 3:
        day = 'wednesday';
    case 4:
        day = 'thursday';
    case 5:
        day = 'friday';
    case 6:
        day = 'saturday';
}
```

Deployment

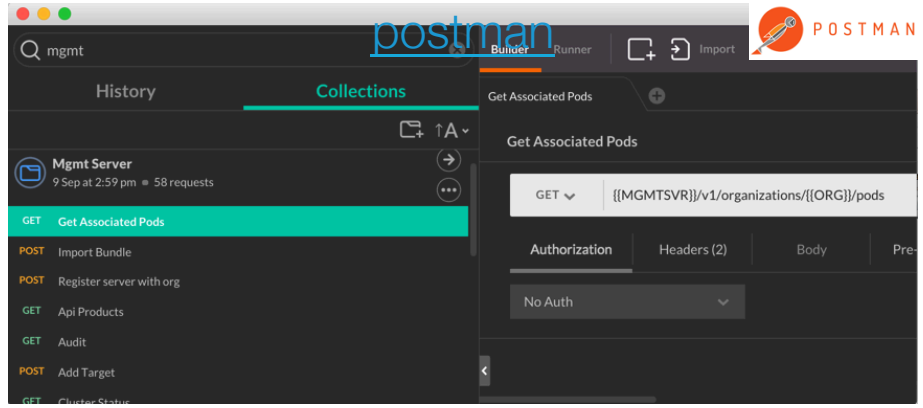
Management API is part the secret sauce

All tools leverage, Apigee Management API

Get it here:
http://apigee.com/docs/management/apis



Get it here:
https://github.com/apigee/apigee-management-api-postman



Deployment through build tools

Coming soon

Apigee Grunt Generator/Plugin

maven

Apigee Maven Plugin

Apigeetool

Deployment through build tools

- Get Apigee Grunt plugin now! https://github.com/apigeecs/apigee-deploy-grunt-plugin
- Yeoman as proxy generator. Fork it and customize it to your needs!

```
ApigeeCorporation on Diegos-MacBook-Pro.local in ~TOOLS_APIGEE/iloveapis
$ sudo yo apigee-deploy-grunt-api

     _-----_
    |       |    .--------------------------.
    |--(o)--|    |    Welcome to Yeoman,    |
   `---------´   |   ladies and gentlemen!  |
    ( _´U`_ )    '--------------------------'
    /___A___\
     |  ~  |
   __'.___.'__
 ´   `  |° ´ Y `

Out of the box I include scaffolding apiproxy, Node.js, and Gruntfile to build your app.
? Your api name: test-yeoman
? Basepath: /test-yeoman
? Management API URL Endpoint: https://api.enterprise.apigee.com
? Organization Name: (testmyapi)
```

Deployment through build tools - Apigee Grunt


```
grunt.registerTask('buildApiBundle',
    'Build zip without importing to Edge',
        ['apigeeGruntPluginBanner',
            'prompt',
                'clean',
                    'saveGitRevision',
                        'shell:apigee_npm_node_modules',
                            'mkdir',
                                'copy',
                                    'xmlpoke',
                                        'string-replace',
                                            'jshint',
                                                'eslint',
                                                    'complexity',
                                                        'compressAlias',
                                                              'istambul_coverage',]);

//3. import revision and run seamless deployment
  grunt.registerTask('DEPLOY_IMPORT_BUMP_SEAMLESS_REVISION',
    [ 'buildApiBundle',
        'getDeployedApiRevisions',
            'apigee_import_api_bundle',
                'installNpmRevisionAlias',
                    'deployApiRevisionAlias',
                        'executeTests',
                              'shell:run_mocha_tests',
                                    'shell:run_jmeter_tests',
                                          'notify:ApiDeployed']);

// importKVM at Organization and Environment level. See apigee_kvm task above
grunt.registerTask('importKVMs',
    ['apigee_kvm:' + grunt.config.get("apigee_profiles")[grunt.option('env')].org + '-' + grunt.option("env"),
        'apigee_kvm:' + grunt.config.get("apigee_profiles")[grunt.option('env')].org]);
```

Deployment through build tools

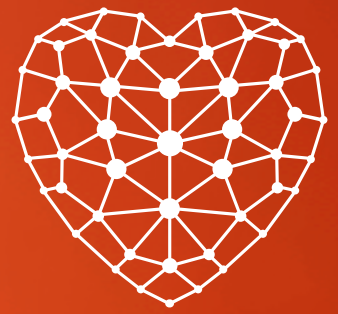
- Get it now! https://github.com/apigee/apigee-deploy-maven-plugin

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>apigee</groupId>
    <artifactId>parent-pom</artifactId>
    <packaging>pom</packaging>
    <version>1.0</version>
    <build>
        <plugins>
            <plugin>
                <artifactId>maven-clean-plugin</artifactId>
                <version>2.5</version>
            </plugin>
            <plugin>
                <groupId>io.apigee.build-tools.enterprise4g</groupId>
                <artifactId>apigee-edge-maven-plugin</artifactId>
                <version>1.0.0</version>
                <executions>
                    <execution>
                        <id>configure-bundle</id>
                        <phase>package</phase>
                        <goals>
                            <goal>configure</goal>
                        </goals>
                    </execution>
                    <execution>
                        <id>deploy-bundle</id>
                        <phase>install</phase>
                        <goals>
                            <goal>deploy</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
```

Deployment through build tools

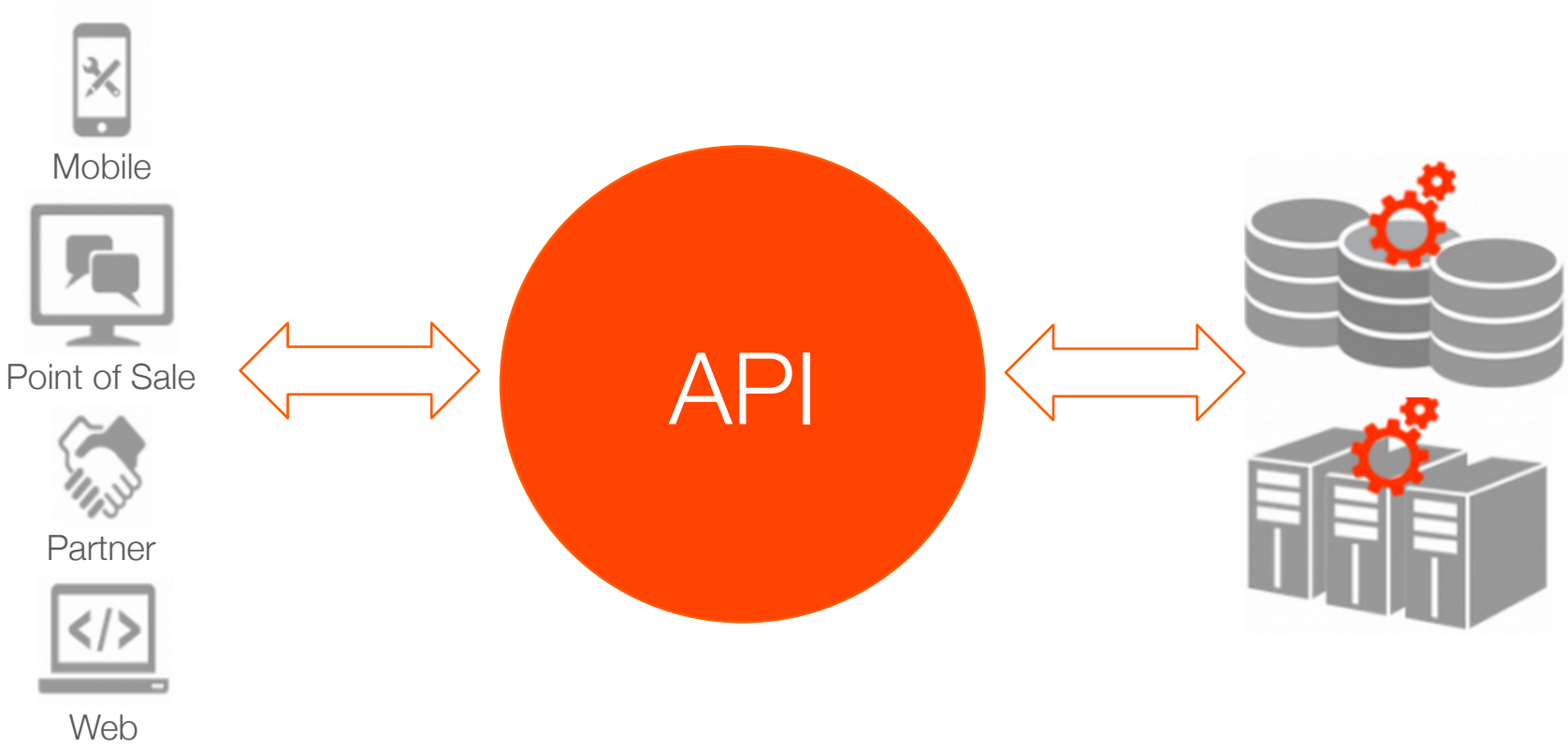
- Get it now! https://github.com/apigee/apigee-deploy-maven-plugin

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>apigee</groupId>
    <artifactId>parent-pom</artifactId>
    <packaging>pom</packaging>
    <version>1.0</version>
<!-- This is where you add the environment specific properties under various profile names -->
 <profiles>
        <profile>
            <id>test</id>
            <properties>
                <org>testmyapi</org>  <!-- default org, replace with default org to avoid passing parameter e.g. -Dorg testmyapi -->
                <options>validate</options>  <!-- default org, replace with default org to avoid passing parameter e.g. -Dorg testmyapi -->
                <apigee.profile>test</apigee.profile>
                <apigee.env>test</apigee.env>
                <apigee.hosturl>https://api.enterprise.apigee.com</apigee.hosturl>
                <apigee.apiversion>v1</apigee.apiversion>
                <apigee.org>${org}</apigee.org>
                <apigee.username>${username}</apigee.username>
                <apigee.password>${password}</apigee.password>
                <apigee.options>${options}</apigee.options>
                <!--apigee.override.delay>10</apigee.override.delay-->
                <!--apigee.delay>1000</apigee.delay-->
            </properties>
        </profile>
    </profiles>
</project>
```



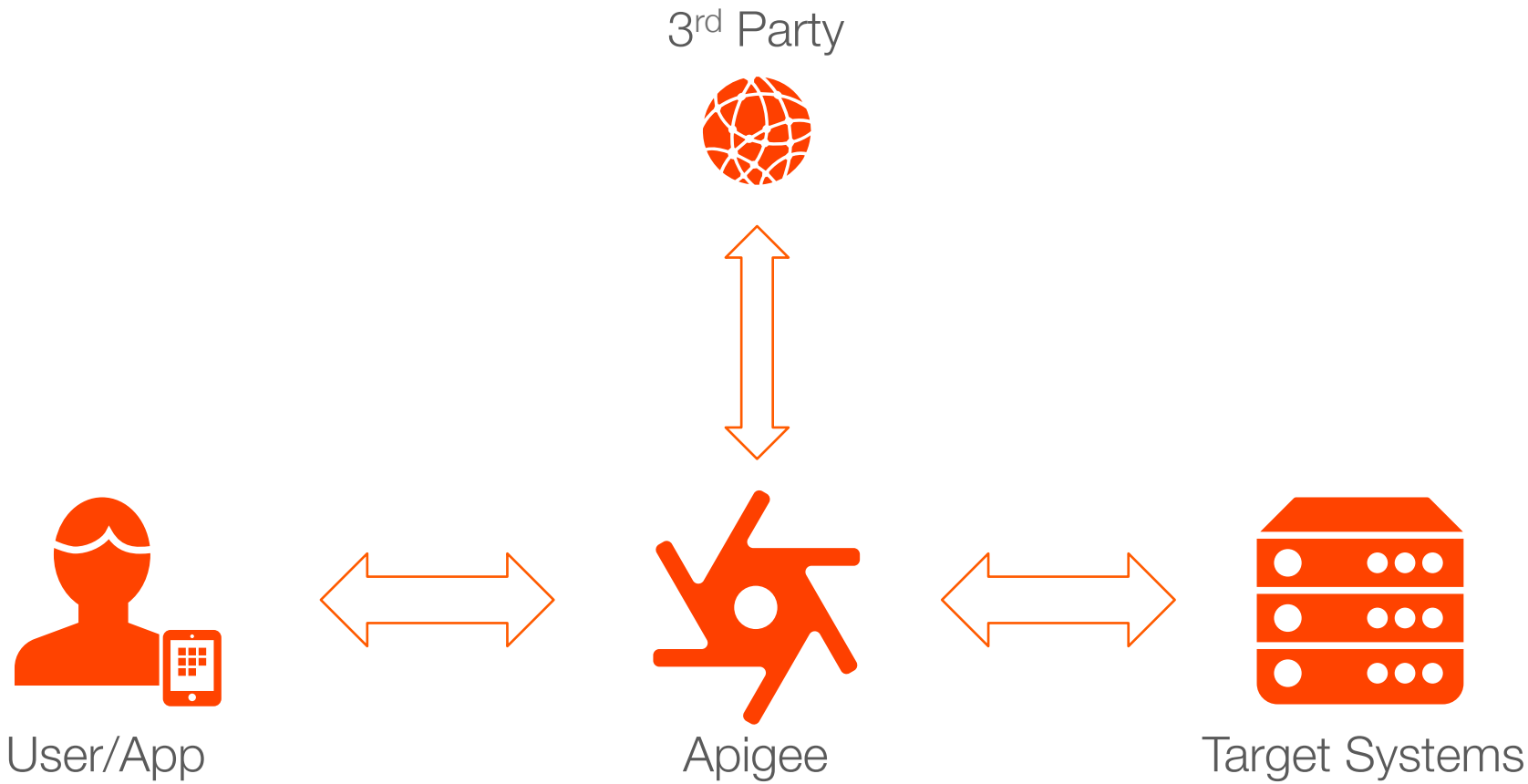
Integration Testing

Integration Testing

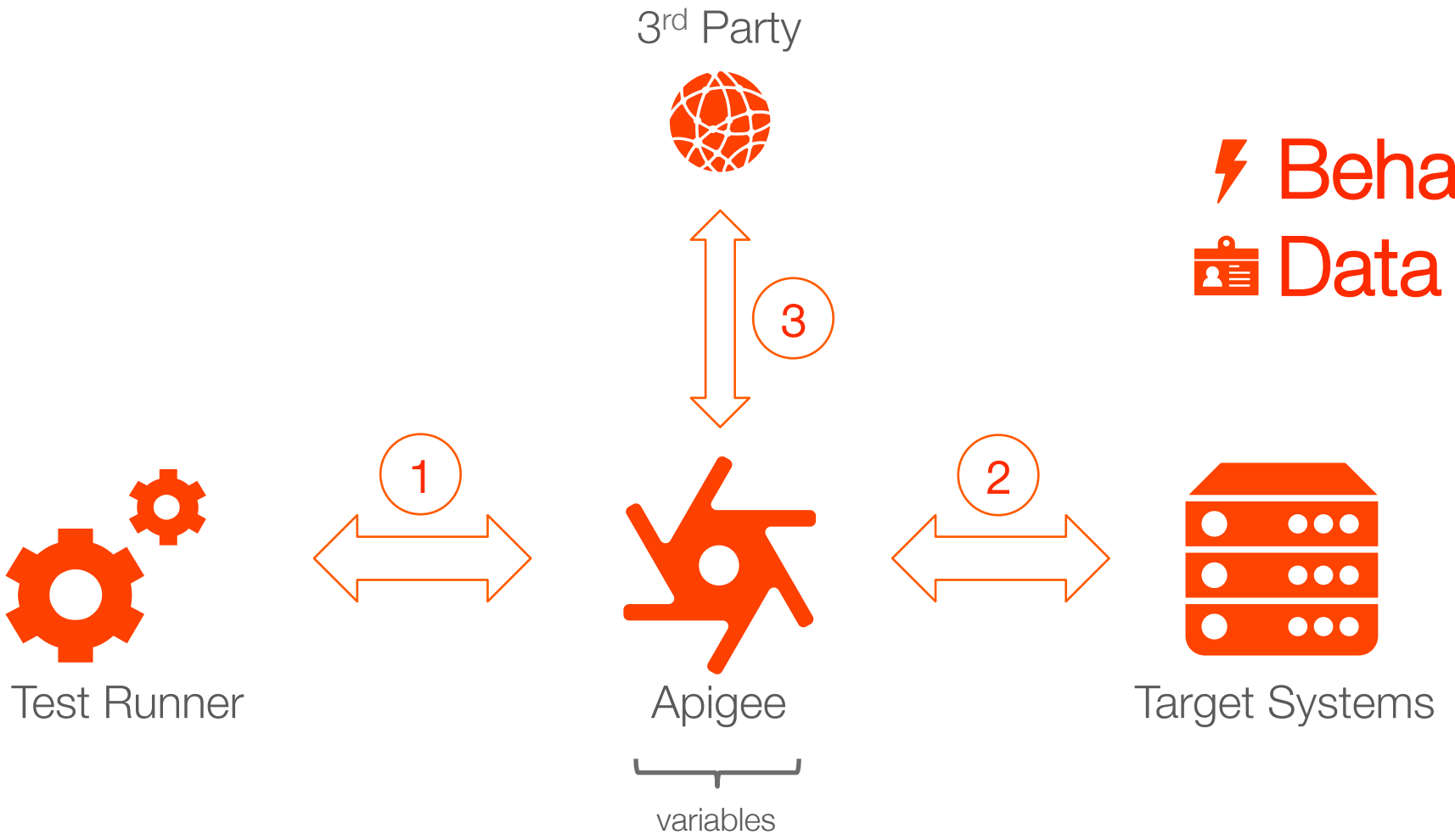


"Most ~~obvious~~ important type of testing for APIs"

Integration Testing – What to test?



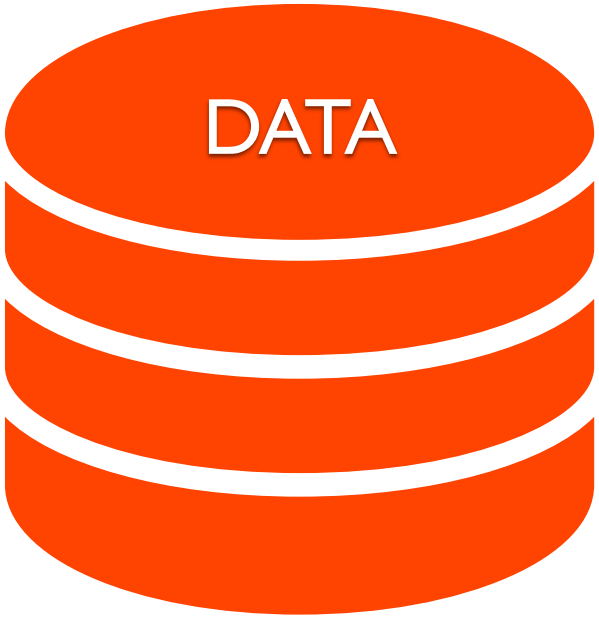
Integration Testing – What to test?



Behavior

Data

Integration Testing – Consistent Data



PAIN!!!

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

Integration Testing – Behavior

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

Integration Testing – From UI?

- This only tests the application – not APIs
- Can't 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
- API team needs to be responsible for API testing
- There are a lot of API clients

"WE DON'T INTEGRATION TEST APIs FROM CLIENT APPs UI"

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

Integration Testing – Tooling

YADDA

APICLI

cucumber

Simple, human collaboration

behat

mocha

Chai Assertion Library

mockito

Apache JMeter™

Sinon.JS

Jasmine

PyUnit

NSpec

jbehave

specflow

Cucumber for .NET

Unit Testing

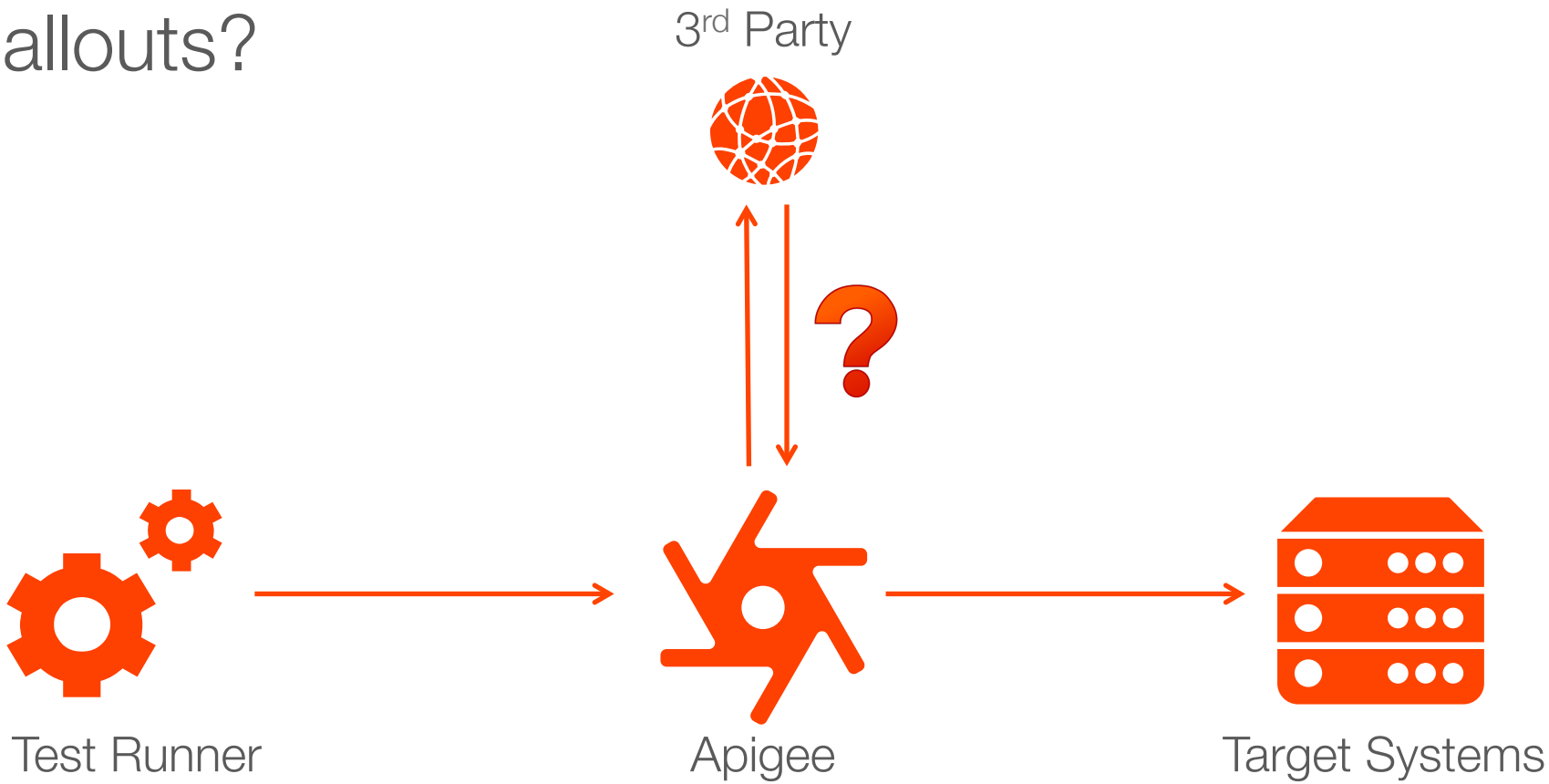
API Proxy Testing

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

No…

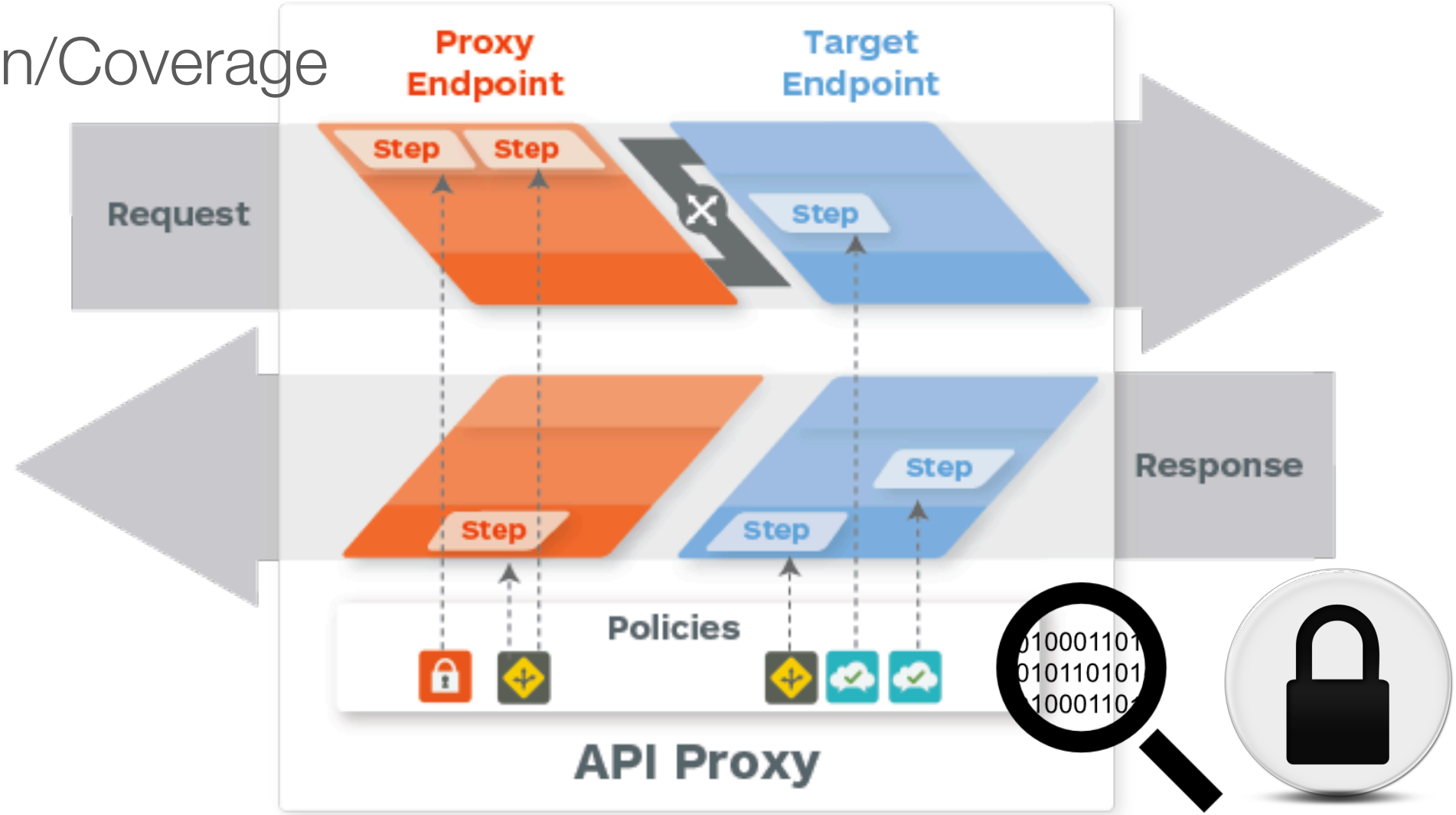
API Proxy Testing – Integration test enough?

Service Callouts?
Async?



API Proxy Testing – Integration test enough?

Isolation/Coverage



Unit Testing – Other benefits

- Code can be tested locally without deployment to Apigee 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

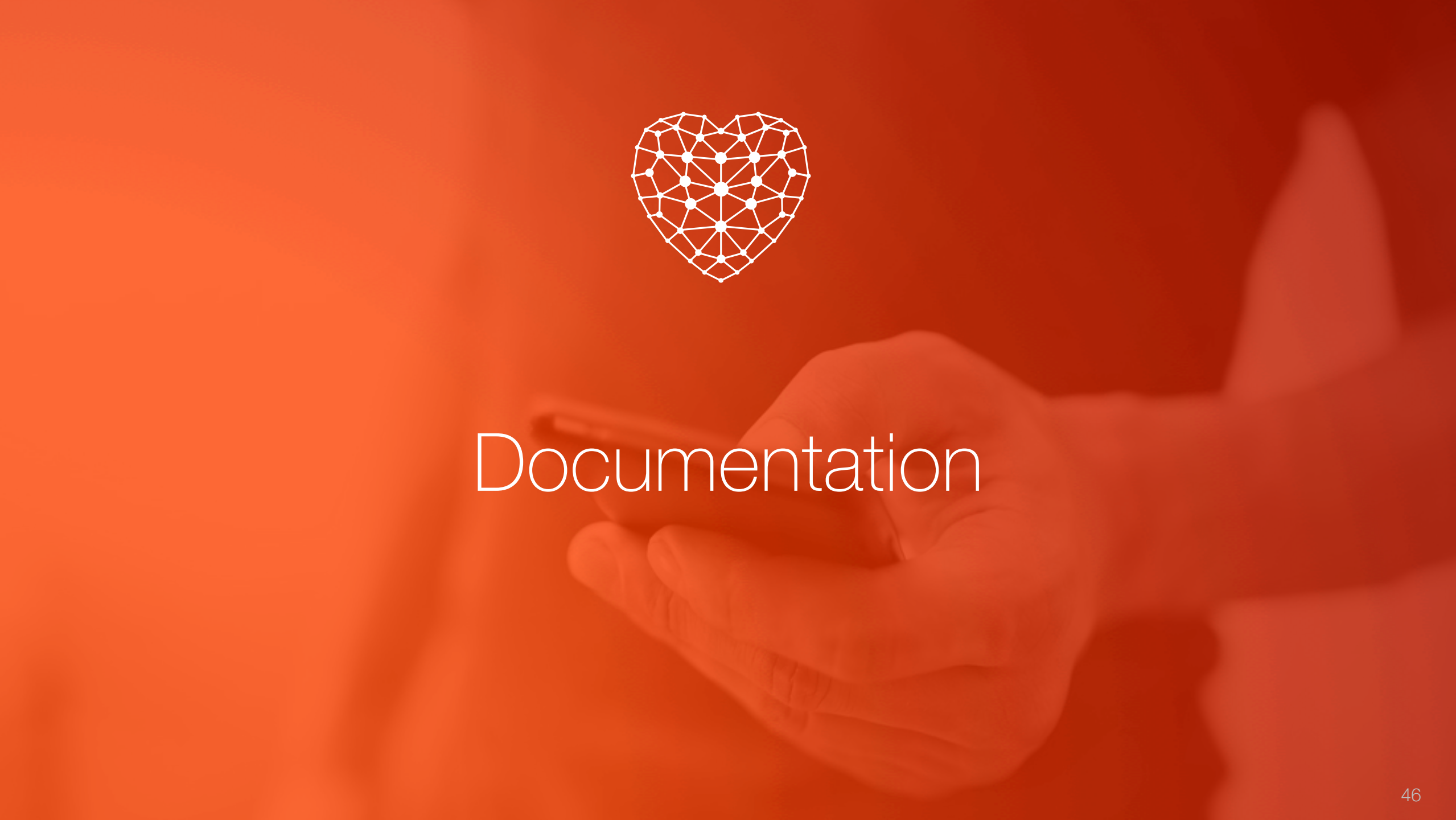
Traffic management policies	Mediation policies	Security policies	Extension policies
Traffic management policies let you configure cache, control traffic quotas and spikes, set concurrent rate limits, and so on. • Cache policies • Concurrent Rate Limit policy • Quota policy • Reset Quota policy • Spike Arrest policy	Mediation policies let you perform message transformation, parsing, and validation, as well as raise faults and alerts. • Access Entity policy • Assign Message policy • Extract Variables policy • JSON to XML policy • Key Value Map Operations policy • Raise Fault policy • SOAP Message Validation policy • XML to JSON policy • XSL Transform policy	Security policies let you control access to your APIs with OAuth, API key validation, and other threat protection features. • Access Control policy • Basic Authentication policy • JSON Threat Protection policy • LDAP policy *† • OAuth v2.0 policies • OAuth v1.0a policy • Regular Expression Protection policy • SAML Assertion policies • Verify API Key policy • XML Threat Protection policy	Extension policies let you provide custom policy functionality, with support for such features as service callout, message data collection, and calling Java, JavaScript, and Python behavior you have created. • Java Callout policy * • JavaScript policy • Message Logging policy • Python Script policy * • Service Callout policy • Statistics Collector policy

Unit Testing – Tooling

Integration Testing – End-to-End Testing Session

Oct 14 – 11:40AM

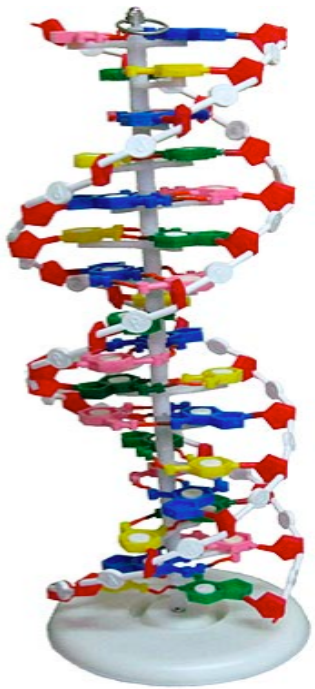
END TO END TESTING: BUG SQUASHING FOR API DEVELOPERS



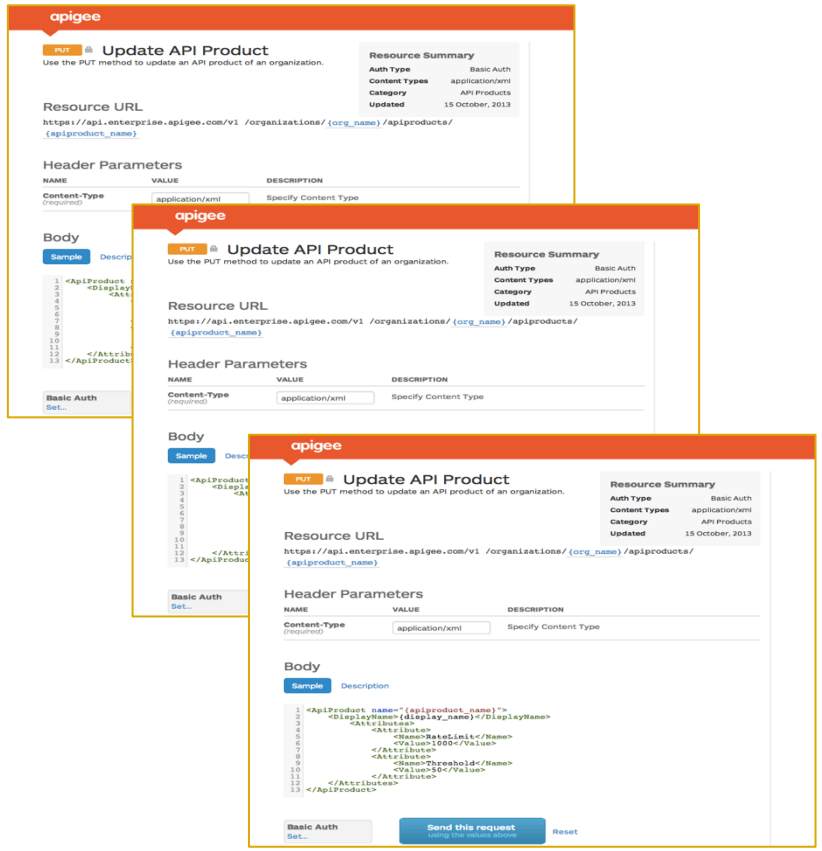
Documentation

Programmable Documentation with SmartDocs

API Modeling
Describe an API structure



SmartDocs
Generate interactive documentation



API-based
Integrate with any portal / CMS

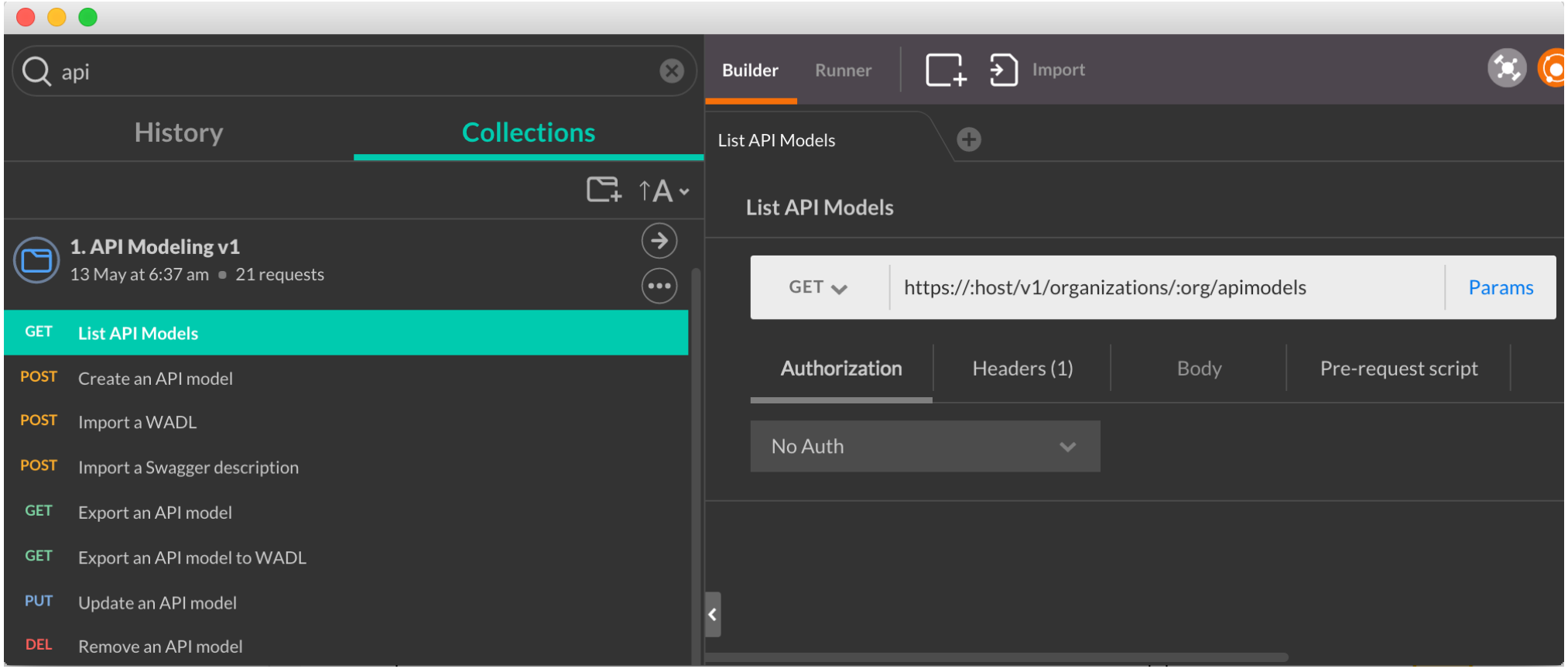
Apigee Edge Developer Services

Other CMS

gh-pages

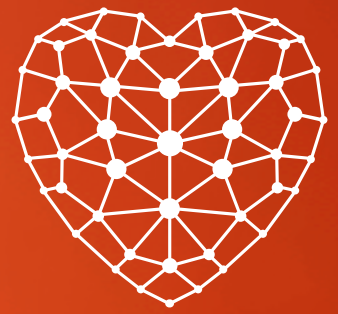
Programmable API Documentation with SmartDocs

Get it now! https://github.com/dzuluaga/apigee-tutorials/tree/master/apiproxies/sample-api-smartdocs



In Conclusion

- CI doesn't happen in a vacuum. So, enable the whole organization
- Don't reinvent the wheel. Leverage frameworks and community
- Don't let tools get on your way. They must be easy to adopt and maintain
- It's not a silver bullet, but if done right, it can make a huge difference
- It won't happen overnight. Master it with practice
- Keep improving it



Questions?