Google Cloud

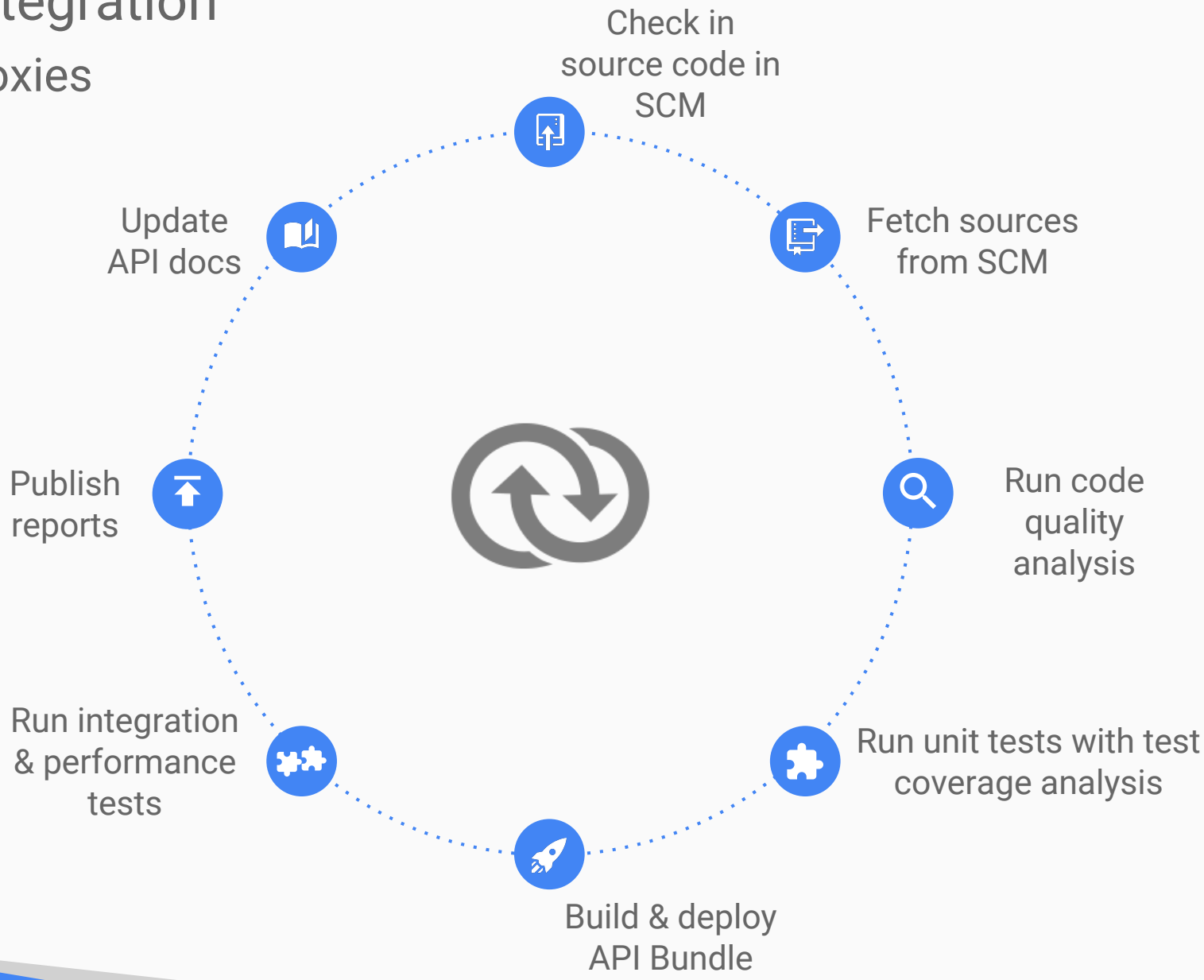
Edge

Continuous Integration and Delivery

Proprietary + Confidential

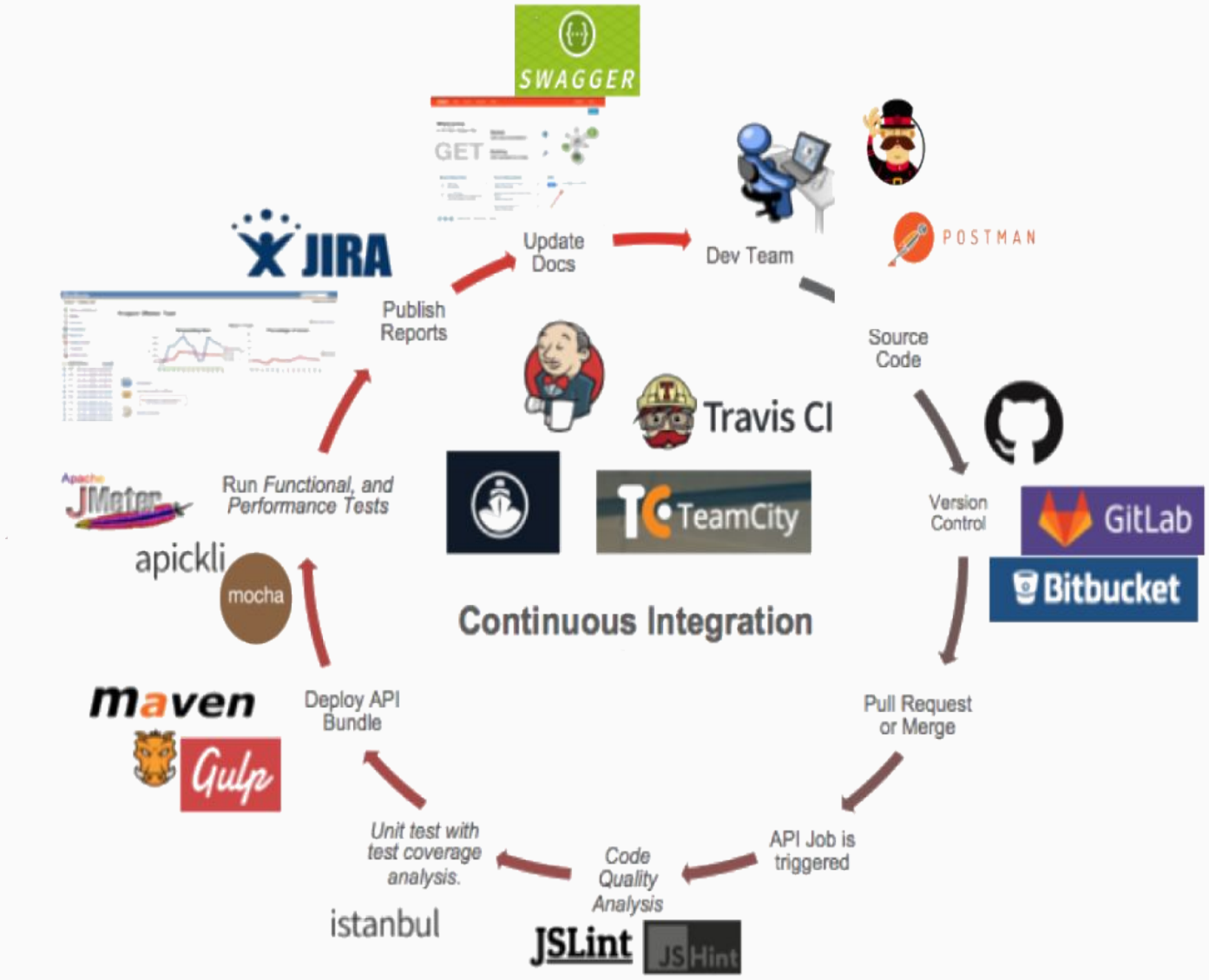
Continuous Integration

Cycle for API Proxies



Proprietary + Confidential

Lifecycle for API Proxies



CI/CD Toolset

Apigee	Customer	Usage	Description
Eclipse	Eclipse	IDE	Integrated Development Environment
GitHub	Github (pending app) otherwise VSTS Git	Source Code Management	Server hosted One repository per API proxy / shared flow Separate repository for global configurations
Jenkins	Jenkins	Continuous Integration / Continuous Delivery Build Server	Server hosted
Maven aigee-edge-maven-plugin apigee-config-maven-plugin	Maven aigee-edge-maven-plugin apigee-config-maven-plugin	Build Management	Standalone via mvn Runs in Maven via Apigee provided plugins
Mocha, Sinen.js, Express	Mocha, Sinen.js, Express	Unit Testing	Standalone via node Runs in Maven via node
JSHint, Apigeelint		Static code quality analysis	JSHint runs standalone via node, in Maven via plugin Apigeelint runs standalone via node, in Maven via node
Apickli + CucumberJS	Apickli + Cucumber.JS	Integration Testing, BDD	Standalone via node Runs in Maven via node
Jmeter		Performance Testing	Standalone via jmeter Runs in Maven via plugin

Proprietary + Confidential

What *IS* API Versioning?

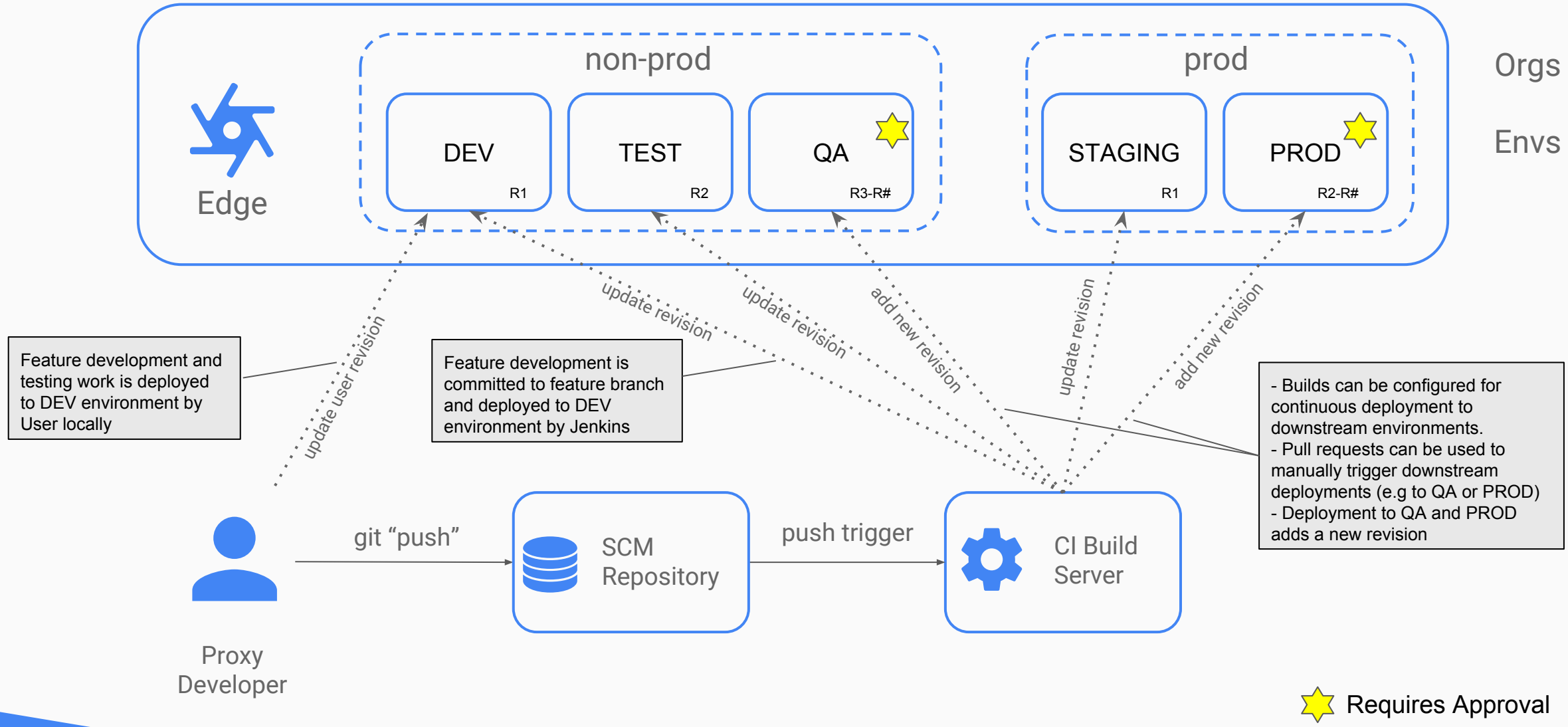
- An API version represents the contract with the consumer.
- Version changes communicates when the API is likely to break backward compatibility.
- Versioning only needs to happen when new required fields are added to queries, OR previously available data are removed from payloads.
- Versioning changes the version identifier, typically the base path of the API proxy

What *ARE* API Revisions

- Revisions represent historical changes to an API as a result of ongoing development and bug fixes.
- A revision change does not change the version identifier for the API.
- Revisions are shared across the organization and can be deployed to any environment in the organization

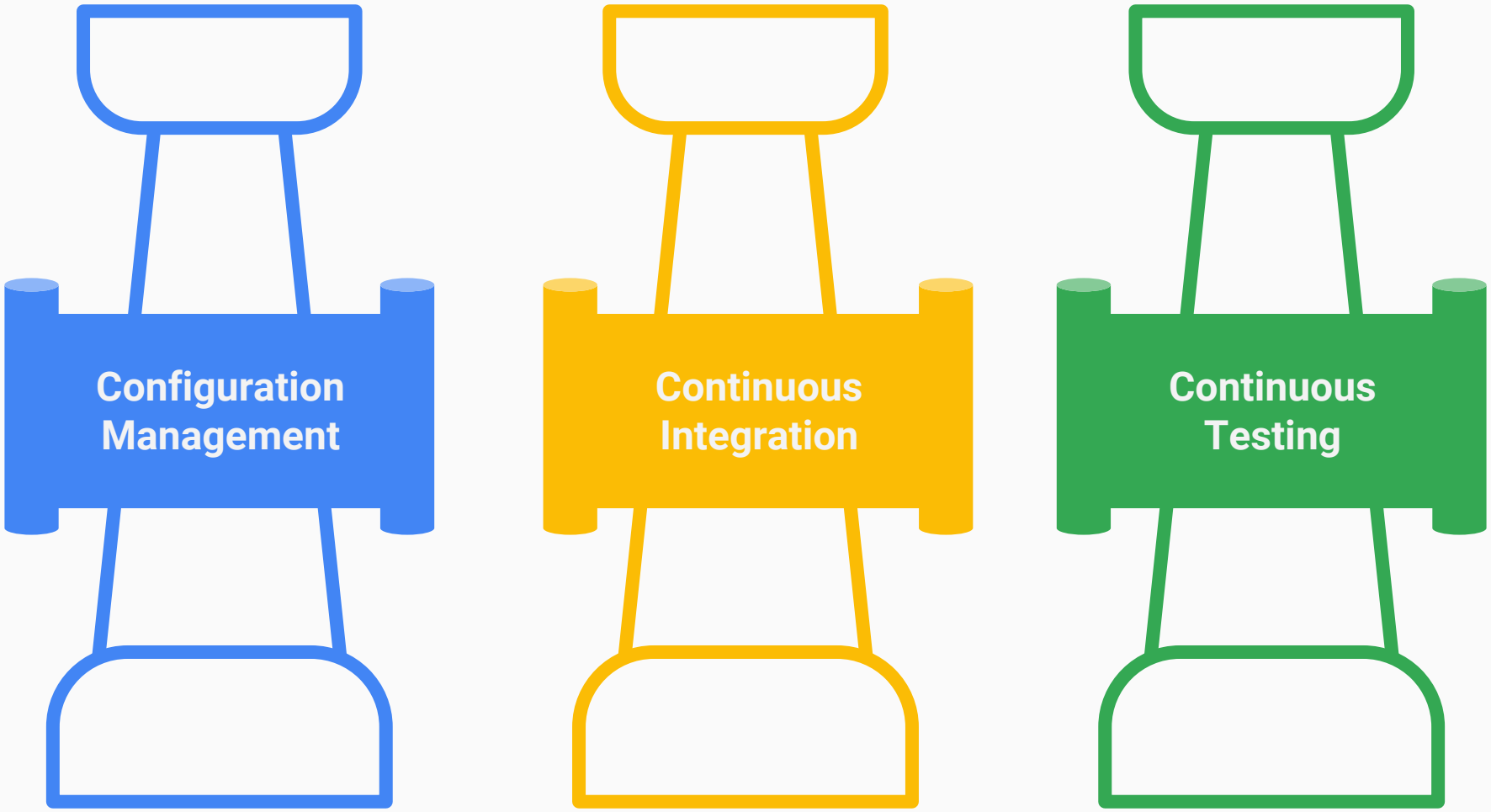
Proprietary + Confidential

Revisions and Deployments for API Proxies



Proprietary + Confidential

Foundations



Proprietary + Confidential

Configuration Management

Goals

Reproducibility

Provision environment configurations in a fully automated fashion, so each environment can be reproduced.

Allow API Proxy code to move between environments without change.

Traceability

Compare previous versions of an environment configuration and see what has changed between them.

Know when a change was incorporated.

Proprietary + Confidential

Continuous Integration

Best Practices

- Maintain a code repository
- Automate the build
- Make the build self-testing
- Everyone commits to the baseline every day
- Every commit (to baseline) should be built
- Keep the build fast
- Test in a clone of the production environment
- Make it easy to get the latest deliverables
- Everyone can see the results of the latest build
- Automate deployment

Proprietary + Confidential

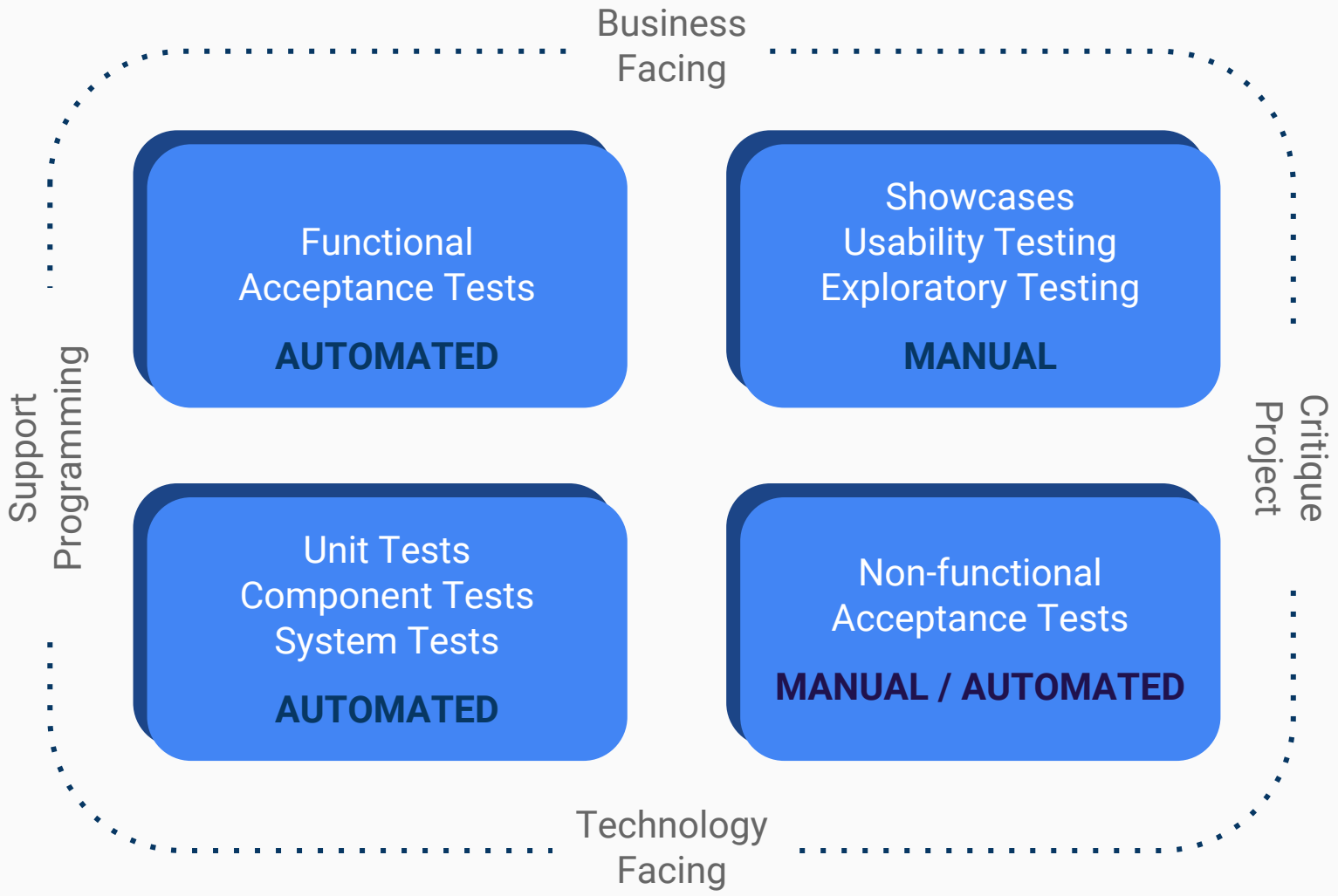
Continuous Integration

Benefits

- Integration bugs are detected early and are easy to track down.
- Last-minute chaos at release date is avoided.
- When tests fail or a bug emerges, and the code needs to be reverted to working state without debugging, only a small set of changes is lost.
- A current build is always available for testing, demo, or release purposes.
- Developers are pushed to create modular, less complex code and supply automated tests.
- Immediate feedback is provided on system-wide impact of local changes.
- Developers spend less time debugging and more time adding features.

Proprietary + Confidential

Continuous Testing



Proprietary + Confidential