Accelerating QA with Microservices Automation

Sai Subramanian, Test Architect

Cognizant Technology Solutions

Cognizant

Abstract

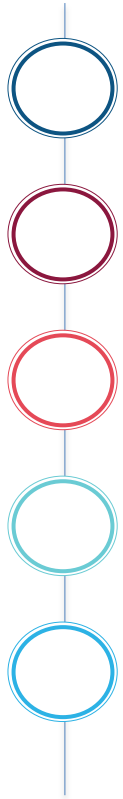
With enterprises increasingly embracing Microservices architecture for scalable and quicker deployments, it's critical to adopt a robust QA strategy that enables zero touch automation. This paper illustrates a Microservices test automation approach including component, contract testing and end-to-end validation. The solution also integrates with the CI/CD pipeline for Microservices using Jenkins pipeline-as-a-code, encompassing Static analysis, regression automation, performance testing and static security analysis. It also addresses the common bottlenecks associated with test infrastructure and mitigates them by deploying docker containers for test execution.

Key take-away for the audience:

- ✓ Using Microservices to accelerate the DevOps delivery
- ✓ Test Automation approach for Microservices
- ✓ Building CI/CD pipeline using Jenkins pipeline-as-a-code
- ✓ Provisioning On-demand docker containers for executing tests
- ✓ Value Proposition and Benefits of Microservices Automation

Agenda

1. Key Business Imperatives
2. Microservices Characteristics
3. Comparison with Monolith
4. Microservices Testing types
5. Deep dive into Contract Testing
6. CI/CD pipeline using Kubernetes / AWS
7. Case Study



Key Business Imperatives

Agility

Release the new features much faster scale

API Economy

Exposure of an organization's digital services and assets through API's

Availability

Zero deployment downtimes. Better fault isolation.

Cloud Native

Leverage scalable, on-demand cloud infrastructure

NETFLIX				
Netflix runs on MS - micro services on AWS	Over **500** micro services	1/3 of NA downstream traffic	**125** million watch each day **4** billion each month	Over **2000** production changes

amazon

Deployments every 11.7 sec

Walmart

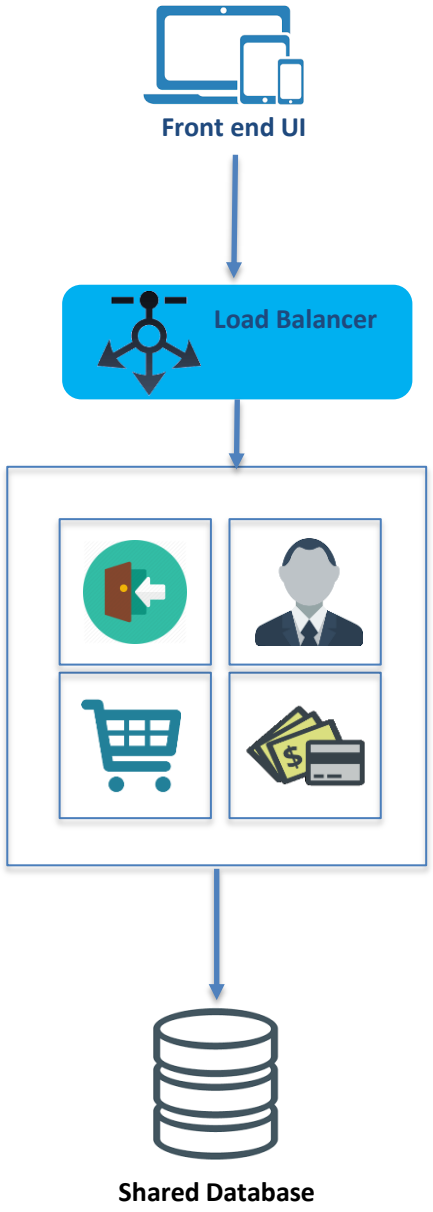
ZERO downtime on Black Friday

Spotify

150m+ users, 58 countries, 2 Billion playlists

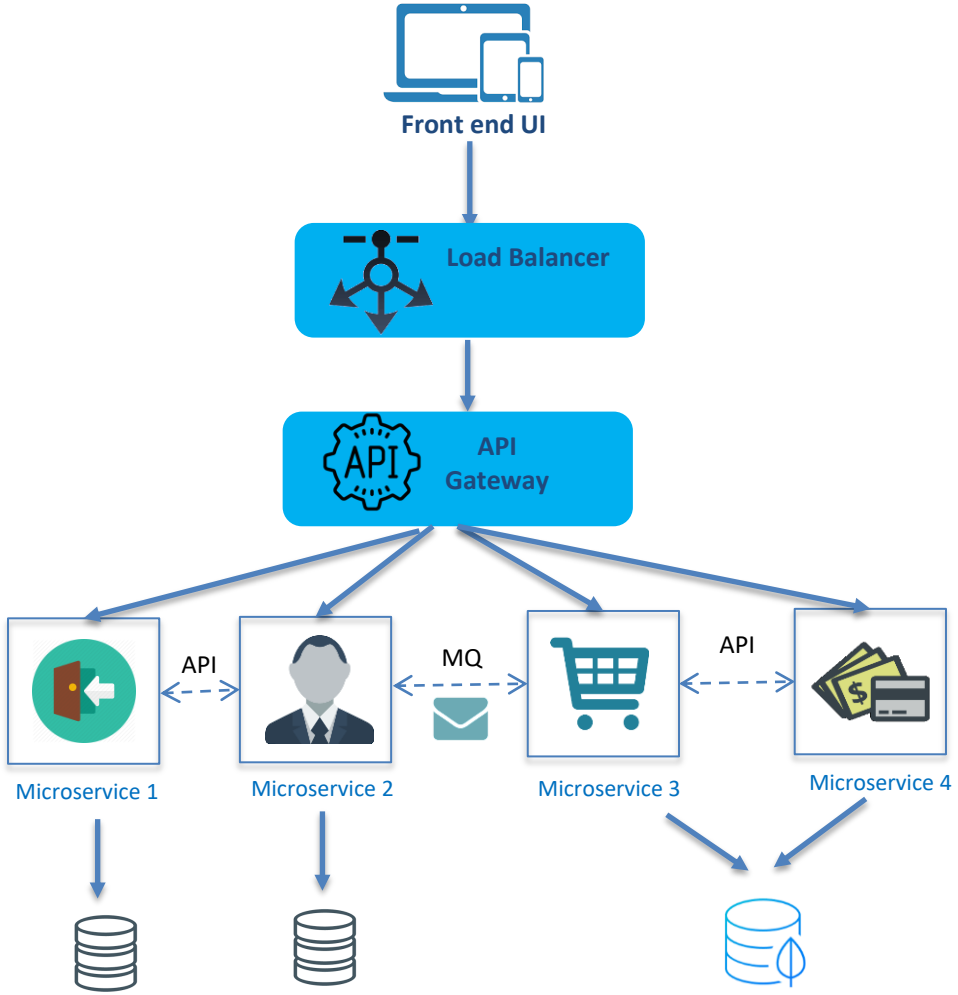
What are Microservices?

Monolithic



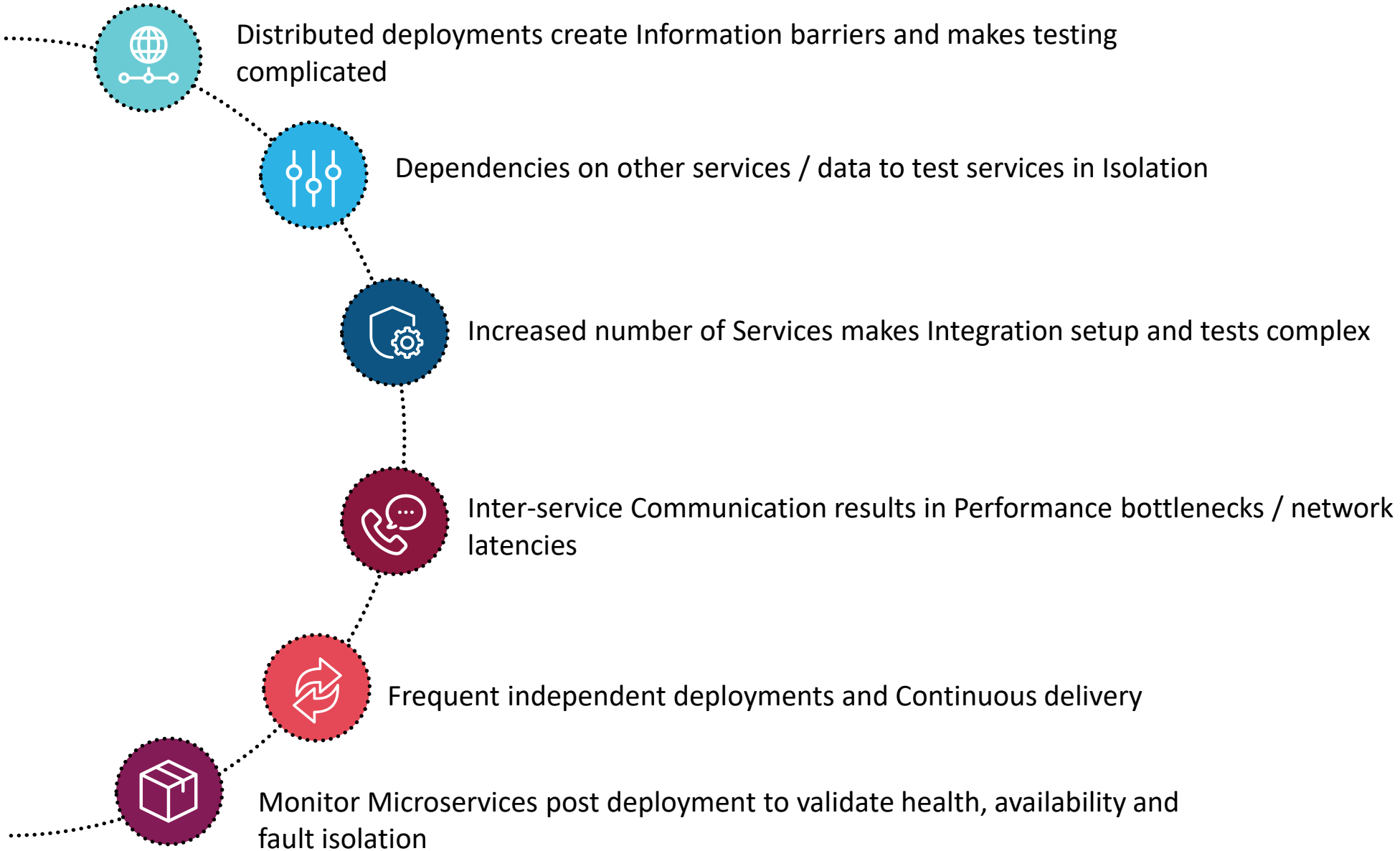
- Built as single entity
- Tightly coupled
- Entire application to be rebuilt and deployed

Microservices



- Suite of Services bounded by Business capabilities
- Loosely coupled
- Independently deployable

Testing Considerations



Distributed deployments create Information barriers and makes testing complicated

Dependencies on other services / data to test services in Isolation

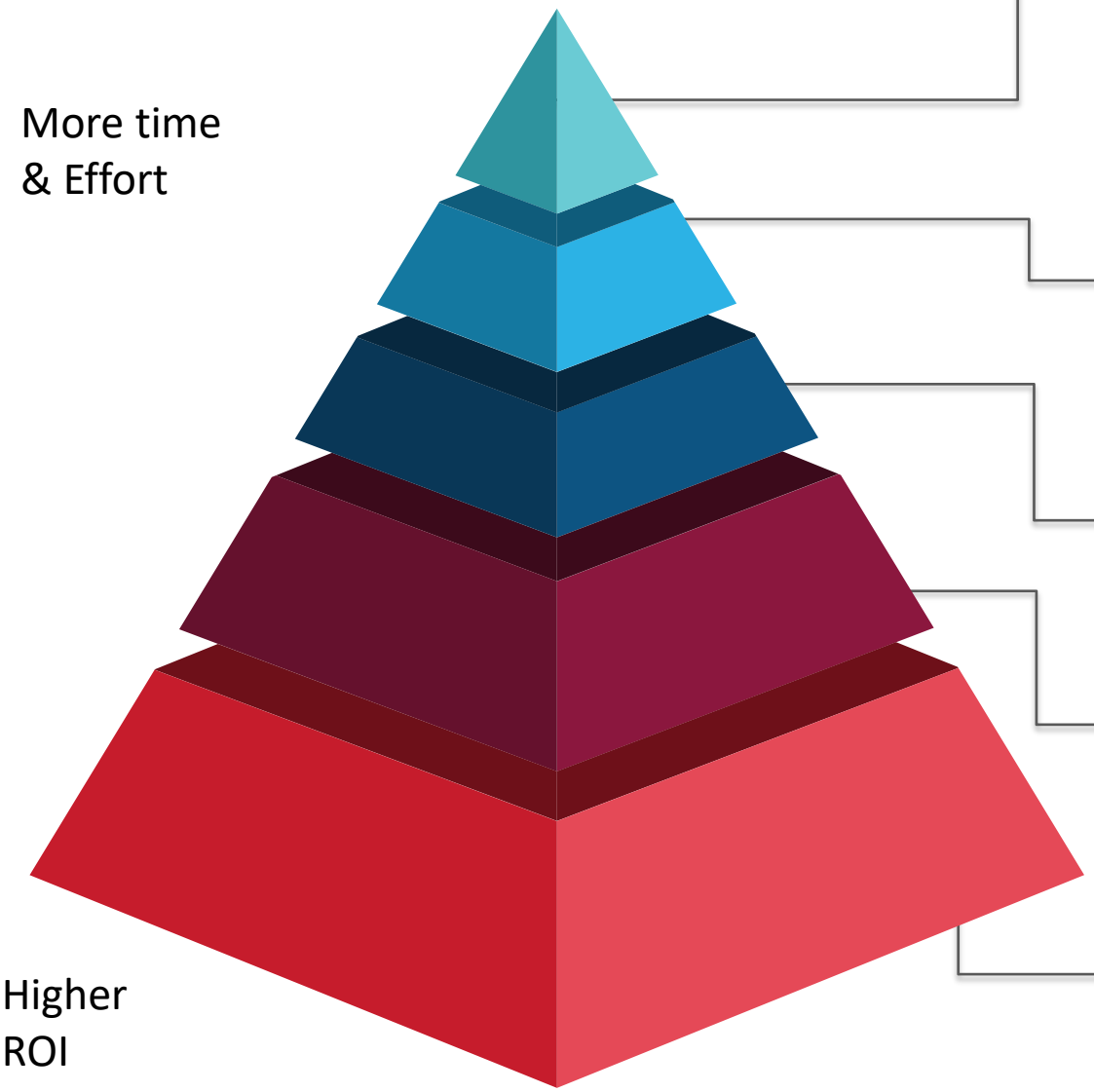
Increased number of Services makes Integration setup and tests complex

Inter-service Communication results in Performance bottlenecks / network latencies

Frequent independent deployments and Continuous delivery

Monitor Microservices post deployment to validate health, availability and fault isolation

Testing types



End-to-End Tests

- Tests the behavior of fully integrated system typically from UI using BDD style
- Ensures test coverage without any gaps between the Microservices
- Assures the correctness of messages passing between the Microservices and also uncovers any configuration issues with associated infrastructure
- Harder, Flaky to run, Expensive to write

Integration Tests

- Tests the integration of Microservices with external/internal consumers and providers.
- Verifies the communication paths and interactions between components to detect interface defects.

Component Tests

- Validate individual microservice as a separate component by virtualizing dependent API
- Publish messages to Queue to validate asynchronous microservices
- Performed via the interfaces the MS expose

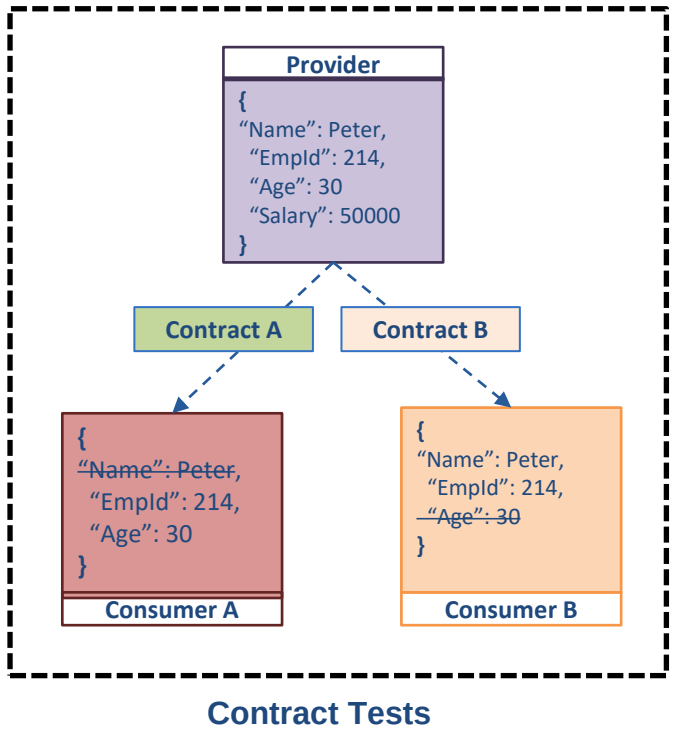
Contract Tests

- Contract is an agreement between the Consumer and Provider about the format of data that they exchange between each other

Unit Tests

- Tests the core functionality of a module with developers
- Any external dependencies / API are stubbed
- Quicker to run, Cheaper to write

Early Integration Tests? - Consumer Driven Contract (CDC) to the rescue…



- Contract test is a ***test at the boundary of an external Microservice for verifying that it meets the contract*** expected by a consuming Microservice.
- Contracts are formed based on the ***requirements of data exchange between provider and consumer***

- Both provider and consumer need not be up and running at the same time
- Fail fast - Identifies integration issues much earlier before new service deployment and running your integration tests
- Increases confidence and provides fast feedback on whether your services are functioning
- Any provider behavior not used by current consumers is free to change without breaking tests
- Eliminate expensive feedback cycles

Contract Testing frameworks: PACTS Spring Contract framework

PACT - Overview

- Open Source Framework for Consumer driven contract testing
- Implementations available for Java, Scala, Ruby etc.,
- Visualization of microservices dependencies
- Contract Test both Synchronous and asynchronous messages

Contract Tests using PACT

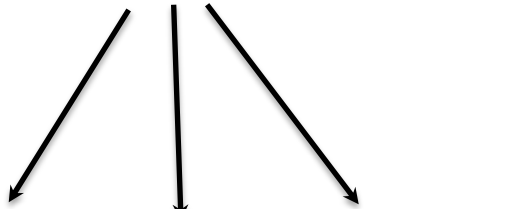
Consumer

HTTP Requests

HTTP Response

PACT File

PACT File



Cloud

File System

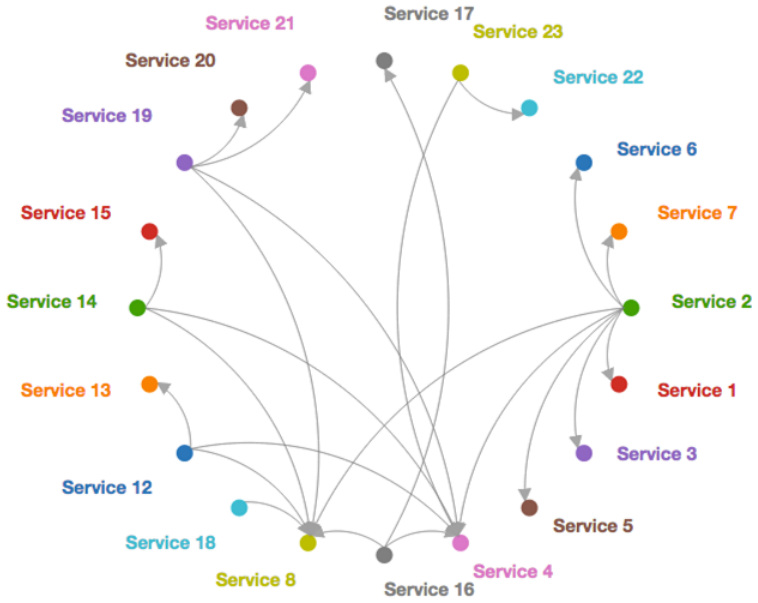
PACT Broker

PACT File

HTTP Requests

HTTP Response

Provider



1

Define Consumer Expectation and create PACT

2

Share the PACT file

3

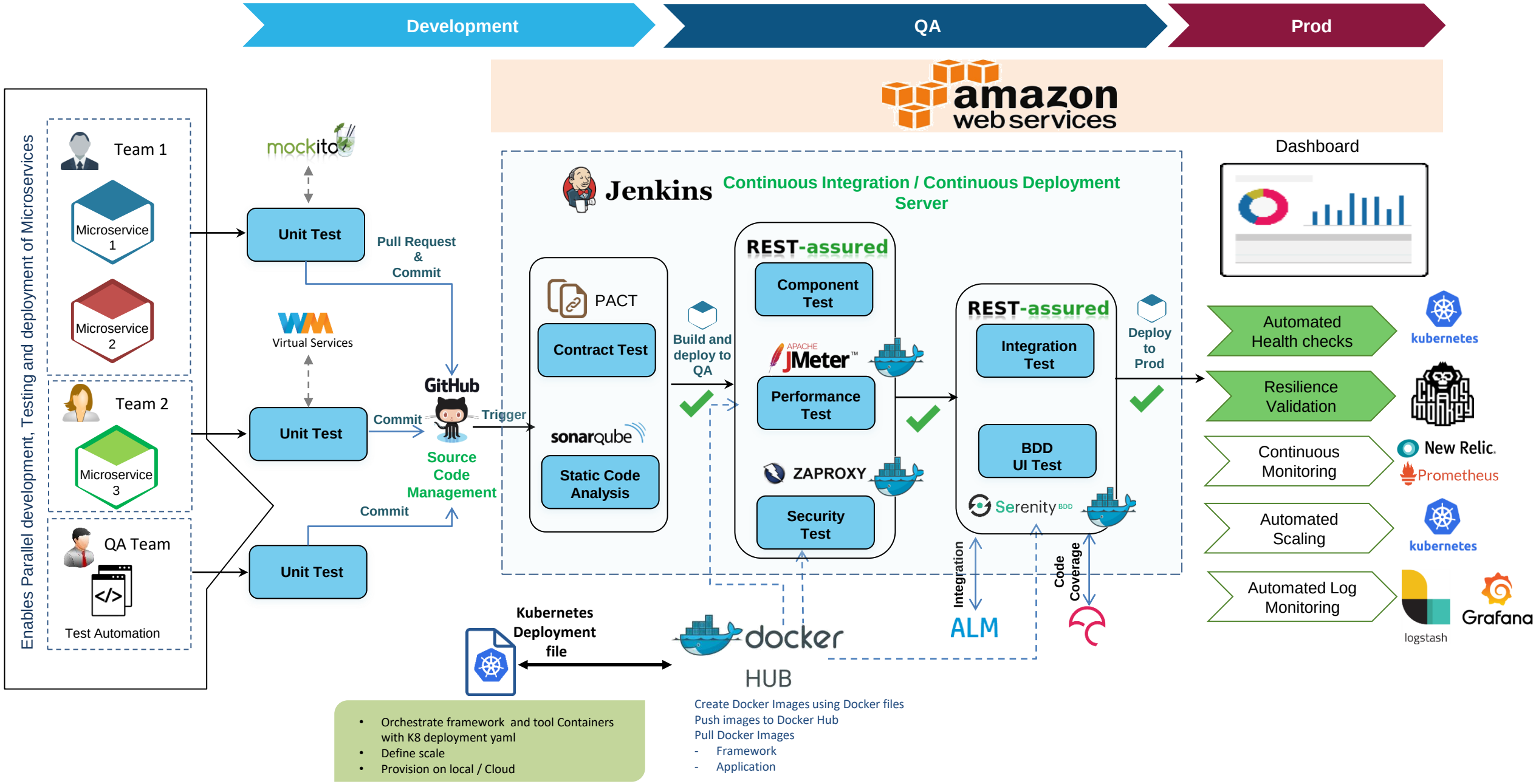
Verify the PACT file against the Provider

4

View Contract results and Dependency Graphs

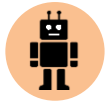
PACT Demo

Parallel Microservices deployment pipeline with AWS/Kubernetes



CI/CD Demo

Changing roles of QA



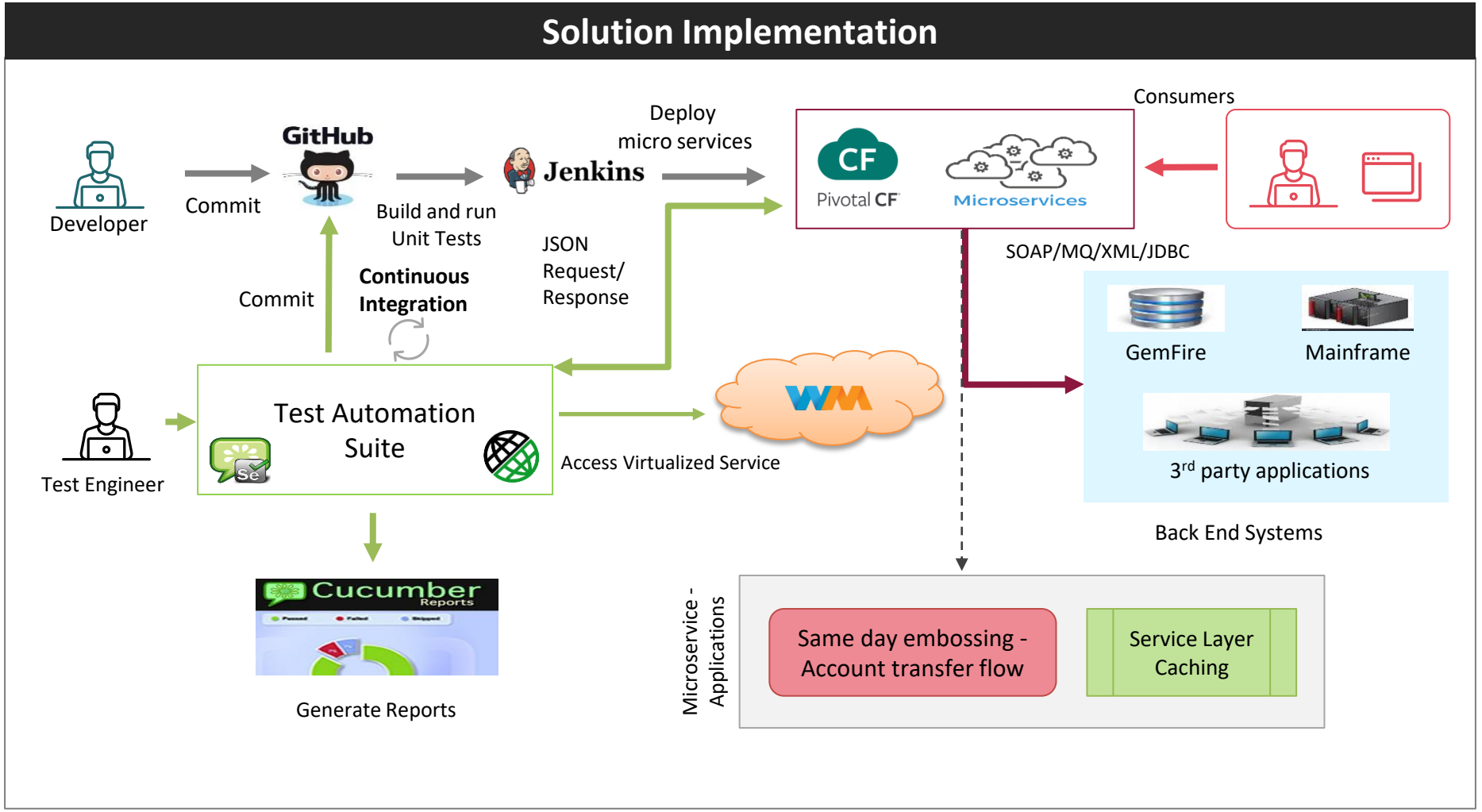
Specialized Test roles to Full Stack QA / SDET.

More involvement of QA in post production validation such as Monitoring, Resiliency validation and log analysis

A/B testing / Canary release to enable new prod release only to a limited set of users

Case Study

Legacy Modernization using Microservices for a leading Financial Services company



Learning's:

- ✓ Fail fast – Run pipeline tests in parallel. Use Jenkins declarative pipeline.
- ✓ Run early Performance tests to resolve component level bottlenecks
- ✓ Swagger documentation – write API tests early
- ✓ Shift Left with contract tests – avoid finding expensive defects late in the integration phase

Key Benefits:

- ✓ Identified 30% of the Integration defects with early contract testing
- ✓ Quicker feedback on code quality. Drastically cut down the communication lag time by over 90%
- ✓ Zero touch automation from developer code commit until build deployment in staging / prod environment

References & Appendix

https://martinfowler.com/articles/microservices.html

https://developer.ibm.com/integration/blog/2017/01/11/common-misconceptions-around-microservices-apis/

https://aws.amazon.com/blogs/compute/kubernetes-clusters-aws-kops/

https://www.nginx.com/blog/introduction-to-microservices/

https://docs.pact.io/

Author Biography

Sai Subramanian has over 14 years of experience in Quality Assurance and has managed the test delivery for strategic engagements from both Offshore and customer locations. He is currently a Test Architect with Cognizant and has expertise in Test Automation, Microservices Testing, Service Virtualization and Performance Engineering. In addition, he has implementation experience in building CI/CD pipeline and DevOps dashboards.

STC 2018

Thank You!!!

Cognizant