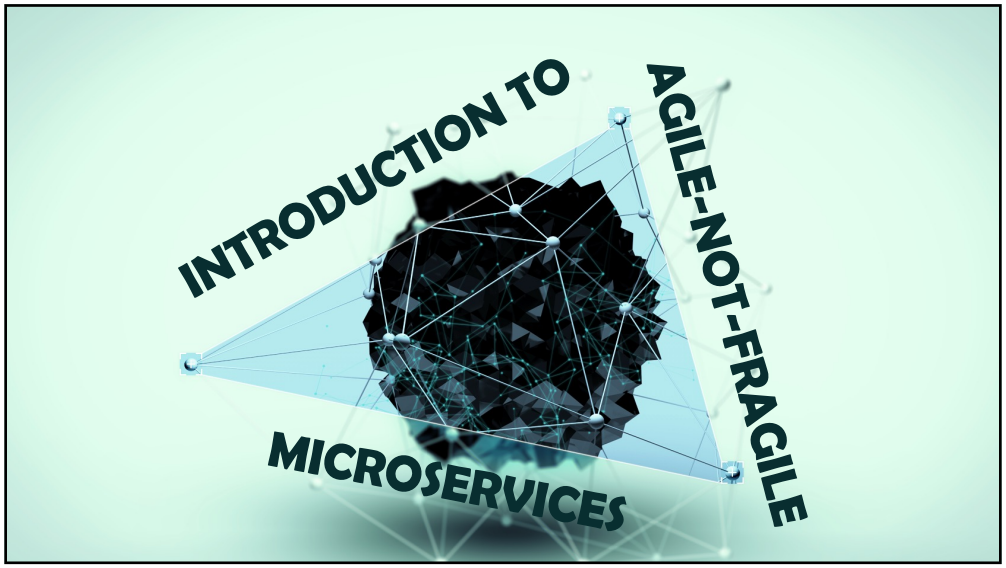

INTRODUCTION TO
AGILE-NOT-FRAGILE
MICROSERVICES

1


Tariq King
FLORIDA TECH
FIU FLORIDA INTERNATIONAL UNIVERSITY
IBM
NDSU
Ultimate SOFTWARE
test.ai
epam

2


Delegates

3


ARCHITECT
It's official team! we're moving to microservices.

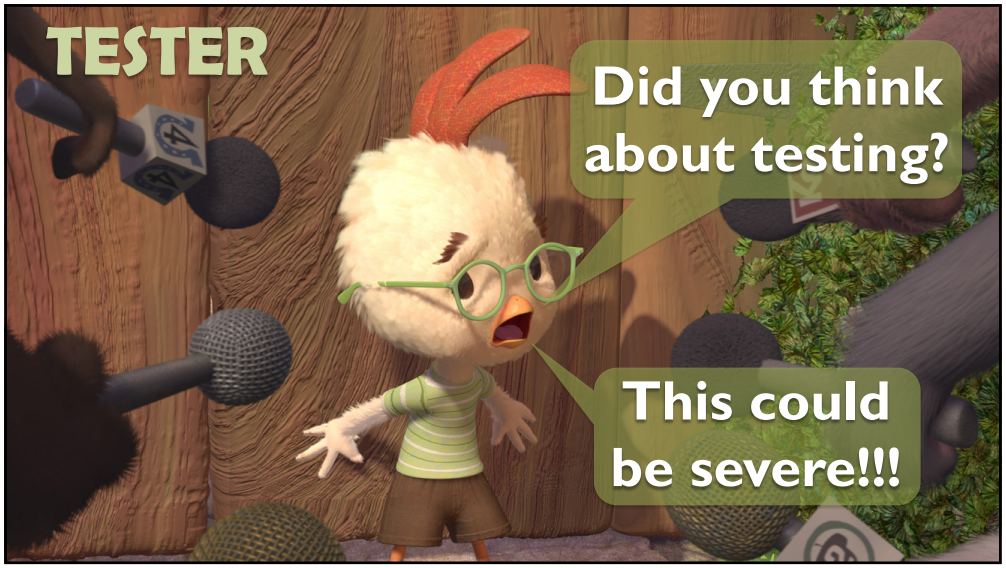
4


DEVELOPER
Yes, finally some new tech in here!

5


ANALYST
MICRO- I don't really care!

6


TESTER
Did you think about testing?
This could be severe!!!

7


ARCHITECT
Err… my bad.

8


The sky is falling.
The end is near.

9

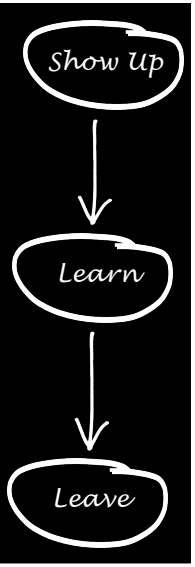

But have no fear.
I am the champion my friend...

10

11

AGENDA

- INTRODUCTION TO MICROSERVICES
  - DEFINITION, CHARACTERISTICS, MOTIVATION
  - STRUCTURE, BEHAVIOR
  - CONTAINERIZATION, PROJECT: FIRST LOOK
- AGILE-NOT-FRAGILE MICROSERVICES DEVELOPMENT
  - COMMUNICATION STYLES AND PATTERNS
  - MICROSERVICE DESIGN AND TESTABILITY
  - FULL-STACK TEST AUTOMATION
  - CHALLENGES, PITFALLS AND ANTIPATTERNS
  - RESLIENCY THROUGH CHAOS ENGINEERING
- WRAP UP
  - QUESTIONS, DISCUSSION, FEEDBACK



12

INTRODUCTION TO MICROSERVICES

13

MICROSERVICES?

14

DEFINITION

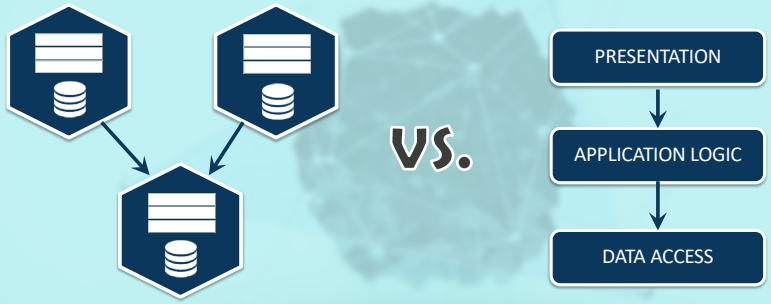
Functional decomposition of system into small, independently deployable services.

Functional decomposition means vertical slicing
as opposed to horizontal slicing into layers.

15

DEFINITION

Functional decomposition of system into small, independently deployable services.



VS.

16

DEFINITION

Functional decomposition of system into small, independently deployable services.

'Micro' refers to small size but there is no universal definition of small.

17

DEFINITION

Functional decomposition of system into small, independently deployable services.

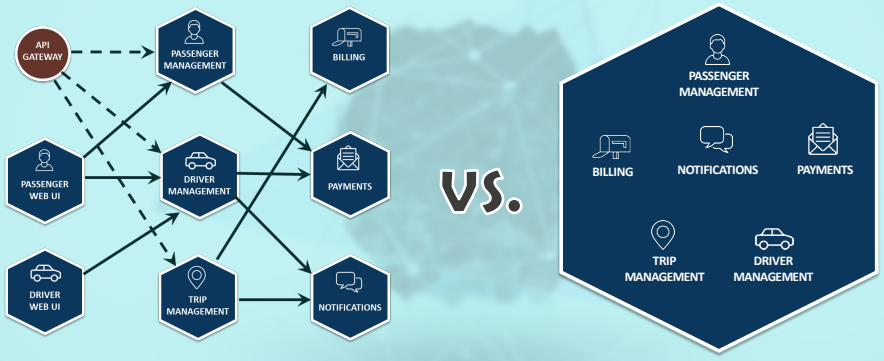
As small as possible but as big as necessary to represent domain.

Manageable by a team of five to six persons.

18

DEFINITION

Functional decomposition of system into small, independently deployable services.



19

DEFINITION

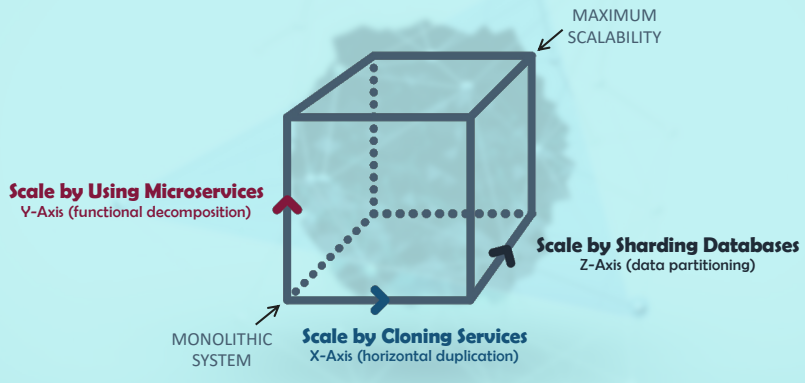
Functional decomposition of system into small, independently deployable services.

Independently deployable also implies that each microservice is independently scalable.

20

DEFINITION

Functional decomposition of system into small, independently deployable services.



21

CHARACTERISTICS

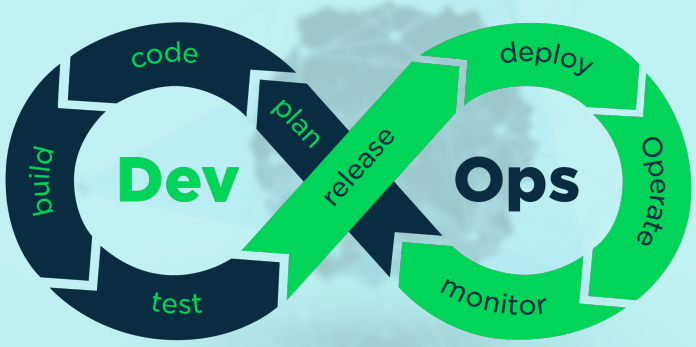
Apply SOLID and Design for Testability Principles at the Service Level

Single Responsibility | Encapsulation (Private) | Monitored (Observable) | Clustered (Instanced) | Technology Agnostic

22

MOTIVATION

Aligns Well with DevOps Practices

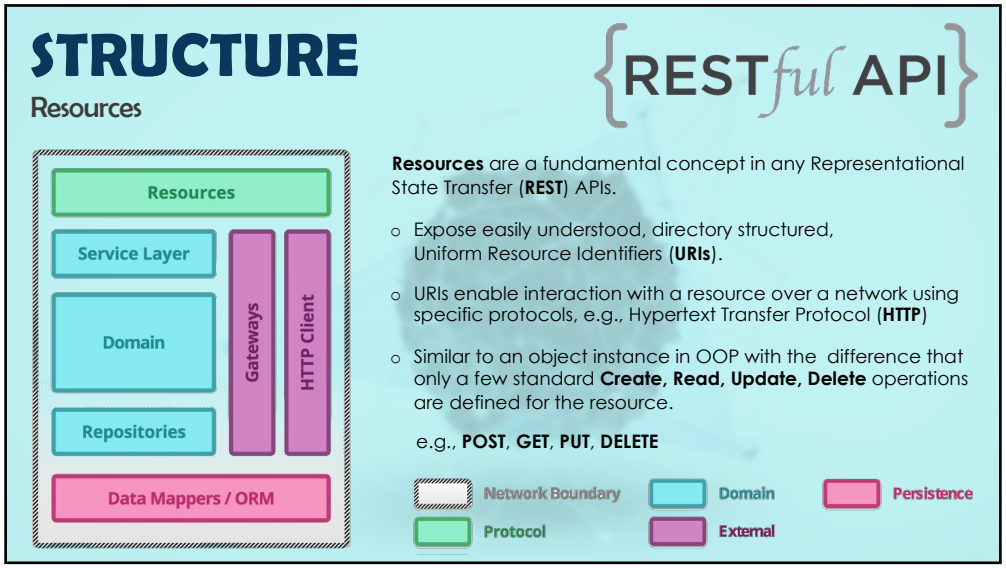


23

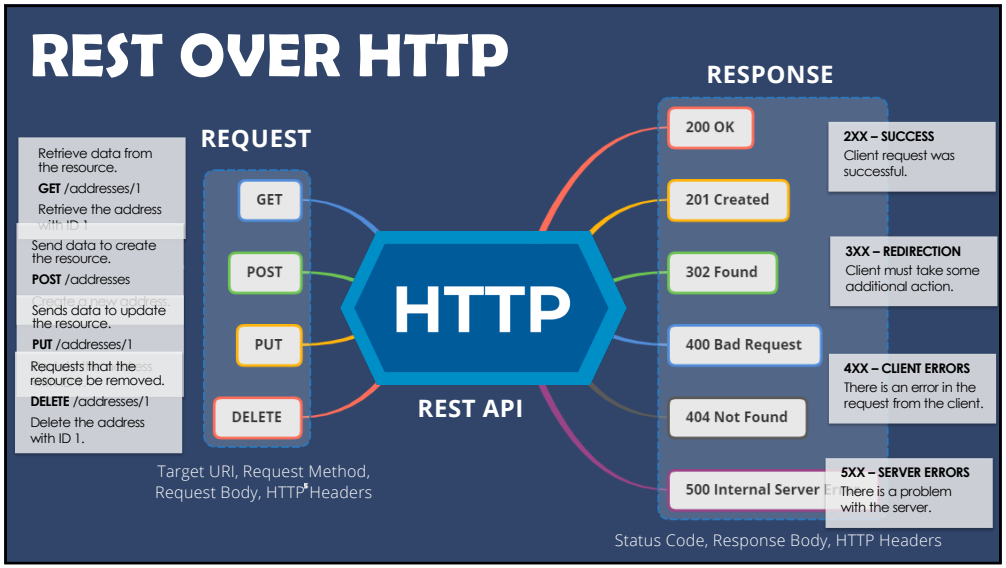
MOTIVATION

Fits Easily into Agile Development Processes

24


STRUCTURE
Resources
{RESTful API}
Resources
Service Layer
Domain
Repositories
Data Mappers / ORM
Gateways
HTTP Client
Resources are a fundamental concept in any Representational State Transfer (REST) APIs.
Expose easily understood, directory structured, Uniform Resource Identifiers (URIs).
URIs enable interaction with a resource over a network using specific protocols, e.g., Hypertext Transfer Protocol (HTTP)
Similar to an object instance in OOP with the difference that only a few standard Create, Read, Update, Delete operations are defined for the resource.
e.g., POST, GET, PUT, DELETE
Network Boundary
Protocol
Domain
External
Persistence

25


REST OVER HTTP
REQUEST
RESPONSE
Retrieve data from the resource.
GET /addresses/1
Retrieve the address with ID 1
Send data to create the resource.
POST /addresses
Sends data to update the resource.
PUT /addresses/1
Requests that the resource be removed.
DELETE /addresses/1
Delete the address with ID 1.
GET
POST
PUT
DELETE
HTTP
REST API
200 OK
201 Created
302 Found
400 Bad Request
404 Not Found
500 Internal Server Error
2XX – SUCCESS
Client request was successful.
3XX – REDIRECTION
Client must take some additional action.
4XX – CLIENT ERRORS
There is an error in the request from the client.
5XX – SERVER ERRORS
There is a problem with the server.
Target URI, Request Method, Request Body, HTTP Headers
Status Code, Response Body, HTTP Headers

26

REST API TESTING

REQ | RES

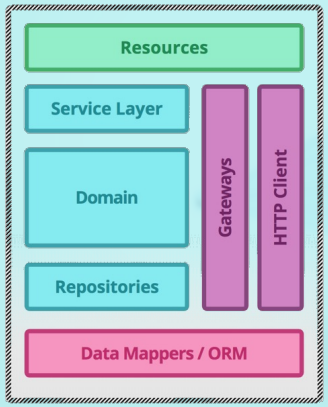
reqres.in

POSTMAN

27

STRUCTURE

Domain, Services, Repositories, Gateways, Data Mappers.



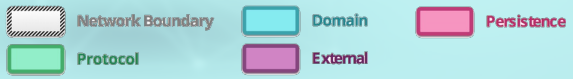
Domain models represent different aspects of the real-world business problem being solved.

Service Layer coordinates business activities.

Repositories act on domain entity collections and are often persistence backed.

Gateways encapsulate message passing with remote services, marshalling requests and responses from or to domain objects. Relies on communication client.

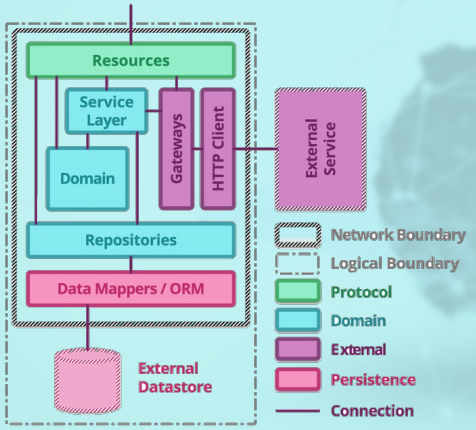
Data Mappers handle conversion of domain entities to/from underlying persistence schemas.



28

BEHAVIOR

Domain, Services, Repositories, Gateways, Data Mappers.



Workflow:

1. A **resource** receives a request, validates it, and calls into the domain to handle the request.
2. Fulfilling the request may involve the resource delegating to a **service**.
3. The resource may communicate directly with a **gateway**, **repository** or a **domain-object**.

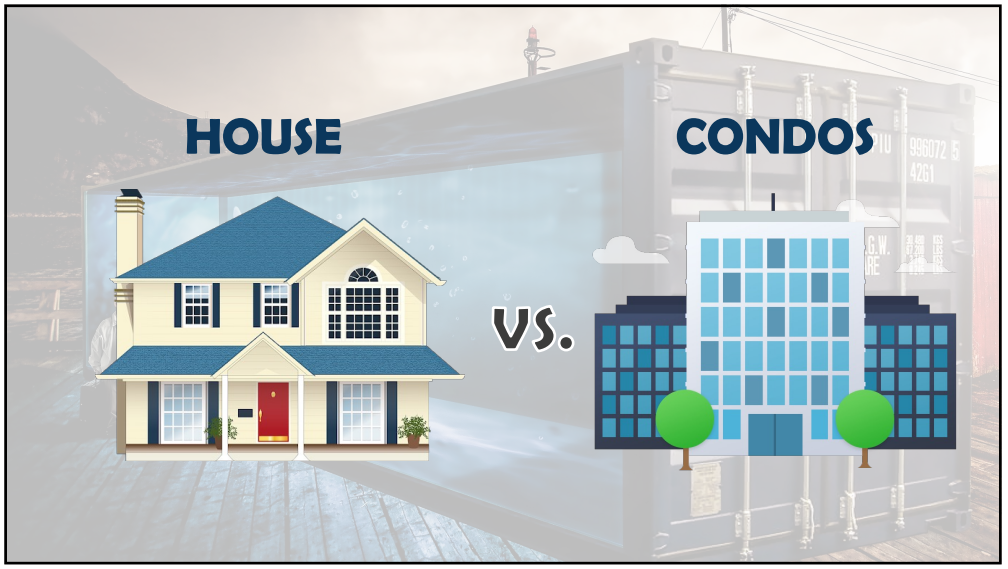
Notes:

- Gateways generally contain logic for handling network outages.
- External datastore communication occurs over a network boundary, and there is also risk of incurring latency and outage.

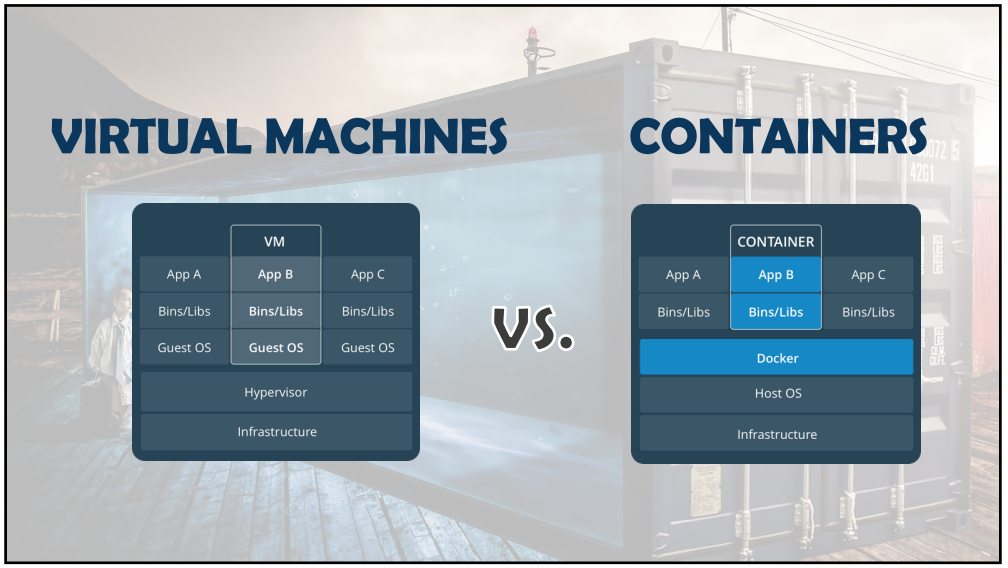
29

30

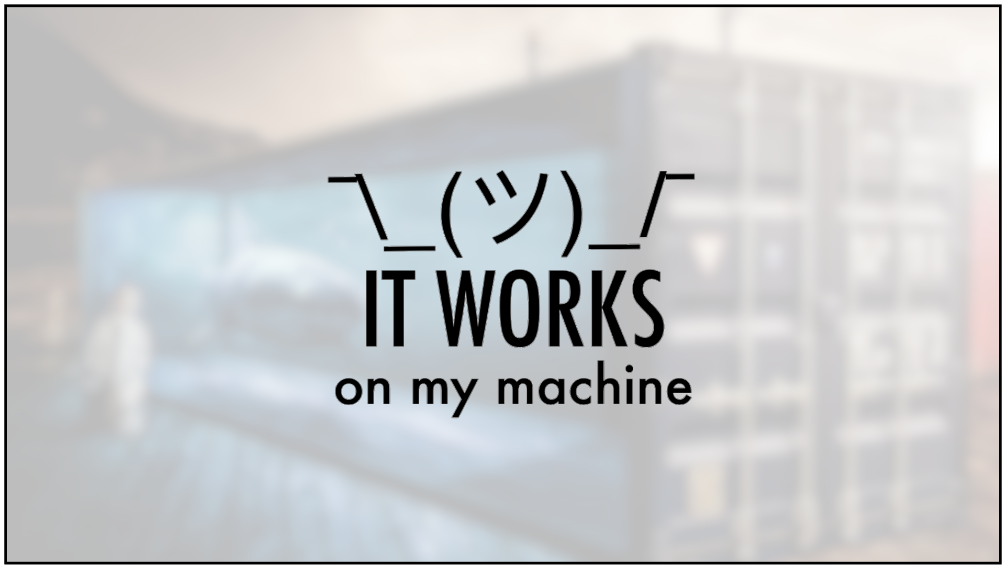
31



32


VIRTUAL MACHINES
CONTAINERS
VM
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Guest OS
Hypervisor
Infrastructure
VS.
CONTAINER
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Docker
Host OS
Infrastructure

33


¯_(ツ)_/¯
IT WORKS
on my machine

34

DOCKER CHEATSHEET

Container Lifecycle

docker create [IMAGE]	create a container without starting it
docker rename [CONTAINER_NAME] [NEW_CONTAINER_NAME]	rename a container
docker run [IMAGE]	create and start a container
docker run --rm [IMAGE]	remove a container after it stops
docker run -td [IMAGE]	start a container and keep it running
docker run -it [IMAGE]	create, start the container, and run a command in it
docker run -it-rm [IMAGE]	create, start the container, and run a command in it, after executing, the container is removed
docker rm [CONTAINER]	delete a container if it isn't running
docker update [CONTAINER]	update the configuration of a container

Image Lifecycle

docker build [URL]	create an image from a Dockerfile
docker build -t [URL]	build an image from a Dockerfile and tags it
docker pull [IMAGE]	pull an image from a registry
docker push [IMAGE]	push an image to a registry
docker import [URL/FILE]	create an image from a tarball
docker commit [CONTAINER] [NEW_IMAGE_NAME]	create an image from a container
docker rmi [IMAGE]	remove an image
docker load [TAR_FILE/STDIN_FILE]	load an image from a tar archieve as stdin
docker save [IMAGE] > [TAR_FILE]	save an image to a tar archive stream to stdout with all parent layers, tags, and versions

Networking

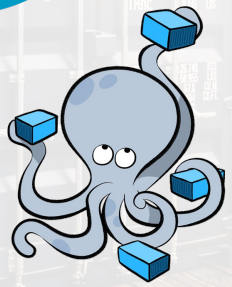
docker network ls	list networks
docker network rm [NETWORK]	remove one or more networks
docker network inspect [NETWORK]	show information on one or more networks
docker network connect [NETWORK] [CONTAINER]	connect a container to a network
docker network disconnect [NETWORK] [CONTINAER]	disconnect a container from a network

Start & Stop

docker start [CONTAINER]	start a container
docker stop [CONTAINER]	stop a running container
docker restart [CONTAINER]	stop a running container and start it up again
docker pause [CONTAINER]	pause processes in a running container
docker unpause [CONTAINER]	unpause processes in a container
docker wait [CONTAINER]	block a container until other containers stop
docker kill [CONTAINER]	kill a container by sending SIGKILL to a running container
docker attach [CONTAINER]	attach local standard input, output, and error streams to a running container

Information

docker ps	list running containers
docker ps -a	list running and stopped containers
docker logs [CONTAINER]	list the logs from a running container
docker inspect [OBJECT_NAME/ID]	list low-level information on an object
docker events [CONTAINER]	list real time events from a container
docker port [CONTAINER]	show port (or specific) mapping from a container
docker top [CONTAINER]	show running processes in a container
docker stats [CONTAINER]	show live resource usage statistics of containers
docker diff [CONTINAER]	show changes to files (or directories) on a filesystem
docker images ls	show all locally stored images
docker history [IMAGE]	show history of an image



35

SPINNING UP MICROSERVICES

Using Containers and Service Discovery to Manage Microservices

BUILD PROJECT

```
./gradlew assemble
```

SPIN UP

```
docker-compose up
```

LIST RUNNING CONTAINERS

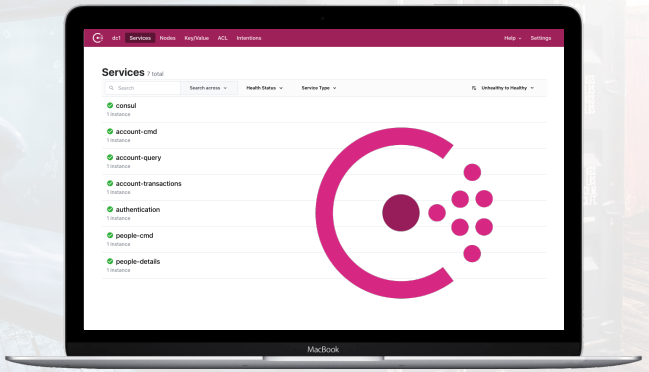
```
docker ps
```

SUSPEND AND RESUME

```
docker pause/unpause
```

SPIN DOWN

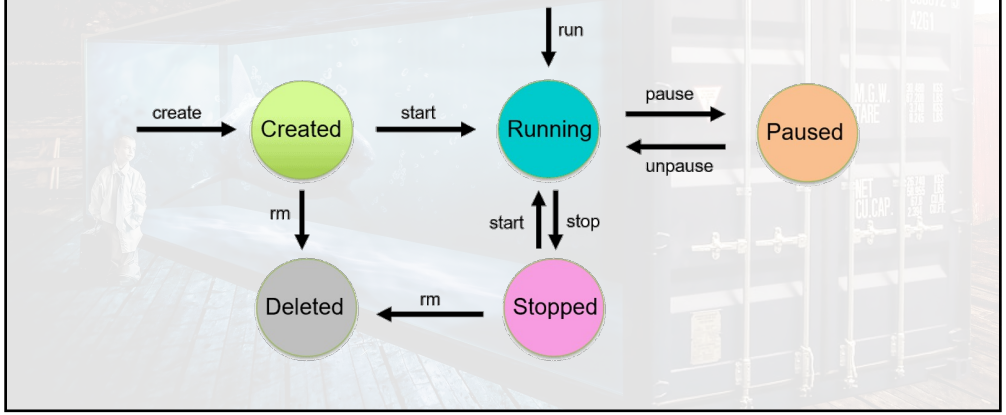
```
docker stop $(docker ps -aq)
```



36

SPINNING UP MICROSERVICES

Docker Container Lifecycle Management



37

PROJECT: FIRST LOOK

Banking Application

38

PROJECT: FIRST LOOK

Technology Stack

express gateway

MICRONAUT

gRPC

HashiCorp Consul

WIREMOCK

mongoDB

JUnit 5

redis

PACT

Java

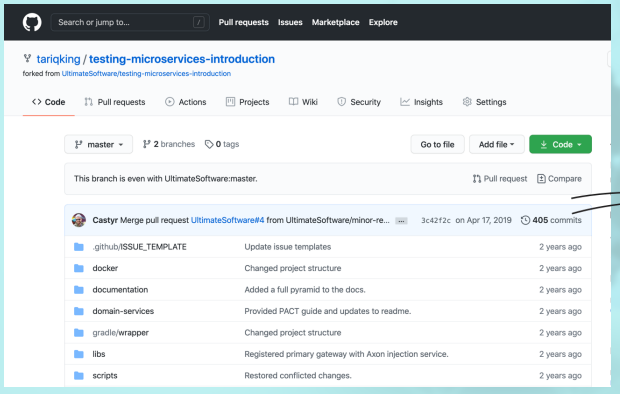
AxonIQ

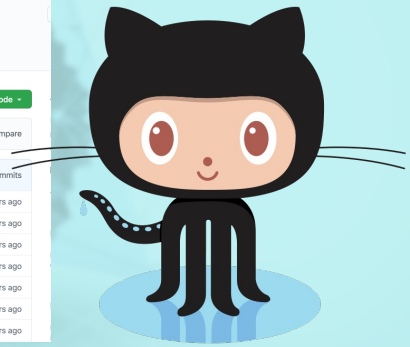
mockito

REST-assured

PROJECT: FIRST LOOK

Open-Source and Available on GitHub

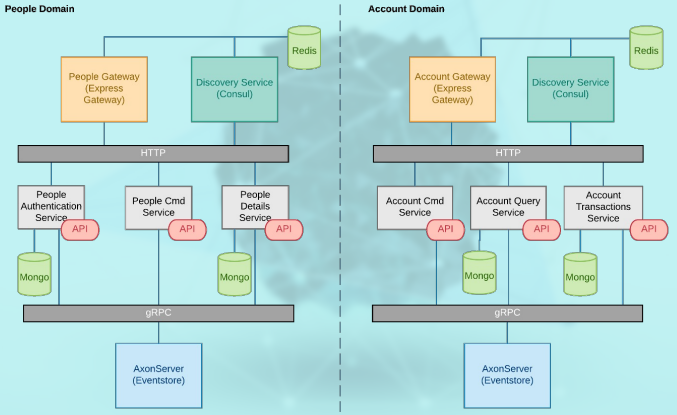




http://github.com/tariqking/testing-microservices-introduction

PROJECT: FIRST LOOK

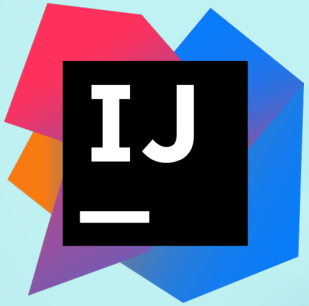
Architecture

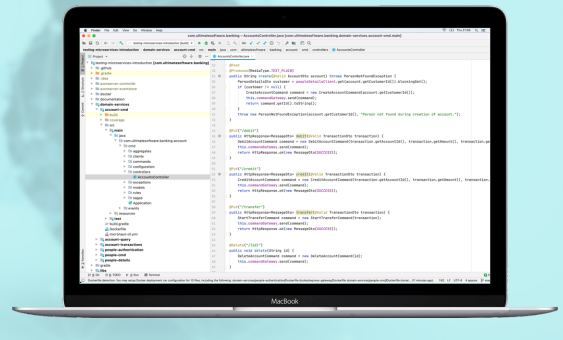


41

PROJECT: FIRST LOOK

Implementation





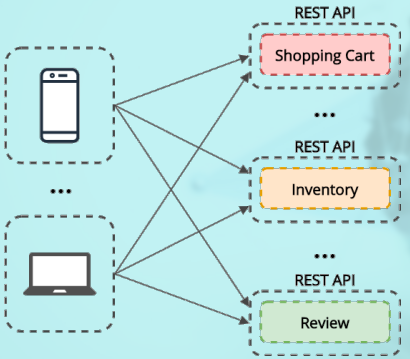
42

AGILE-NOT-FRAGILE MICROSERVICES DEVELOPMENT

COMMUNICATION STYLES AND PATTERNS

COMMUNICATION STYLES

Direct Client to Microservice Communication



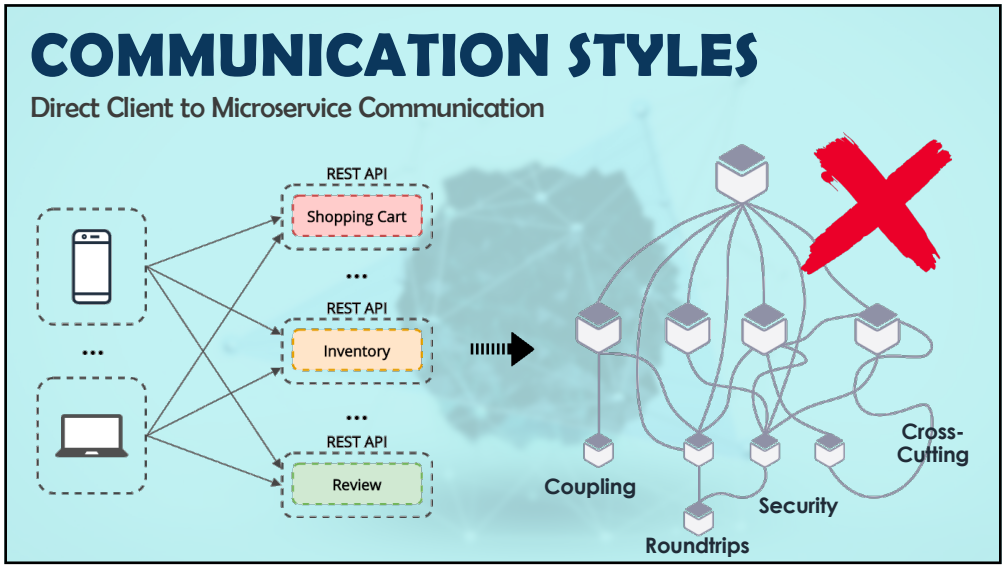
In a microservices architecture, each microservice typically exposes a set of fine-grained end-points.

A possible communication style is using **direct client-to-microservice communication.**

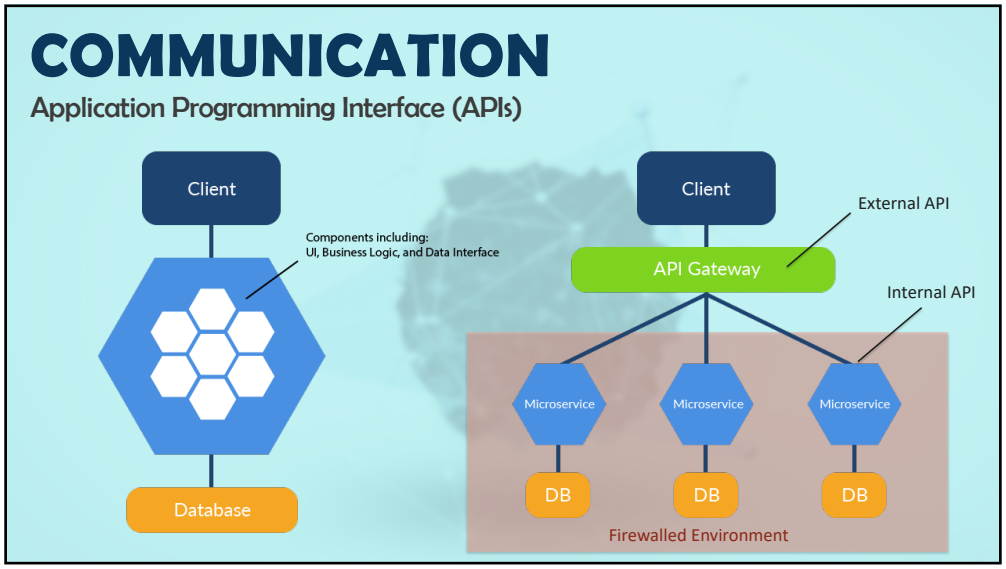
- Endpoints are made public
- Clients make direct calls

Any benefits to this approach?

How about foreseeable issues?


COMMUNICATION STYLES
Direct Client to Microservice Communication
REST API
Shopping Cart
REST API
Inventory
REST API
Review
Coupling
Cross-
Cutting
Security
Roundtrips

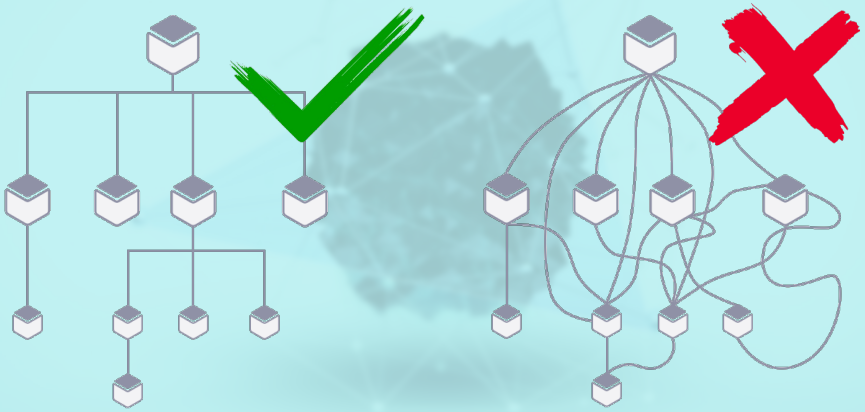
45


COMMUNICATION
Application Programming Interface (APIs)
Client
Components including:
UI, Business Logic, and Data Interface
Database
Client
External API
API Gateway
Internal API
Microservice
Microservice
Microservice
DB
DB
DB
Firewalled Environment

46

COMMUNICATION STYLES

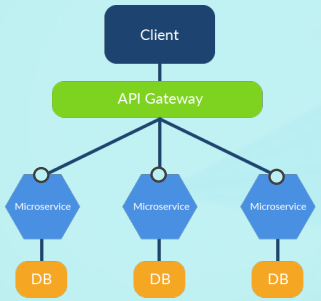
API Gateway Pattern

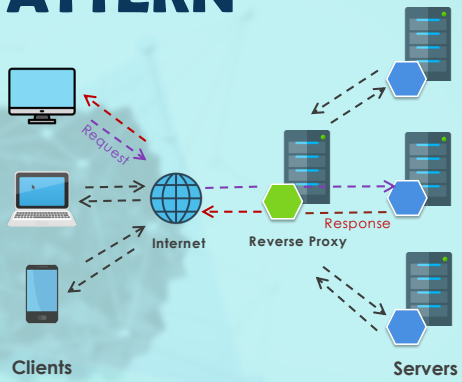


47

API GATEWAY PATTERN

Overview





API gateway sits between the client applications and the microservices.

Acts as a reverse proxy, routing requests from clients to services.

48

API GATEWAY PATTERN

Design Features

GATEWAY ROUTING

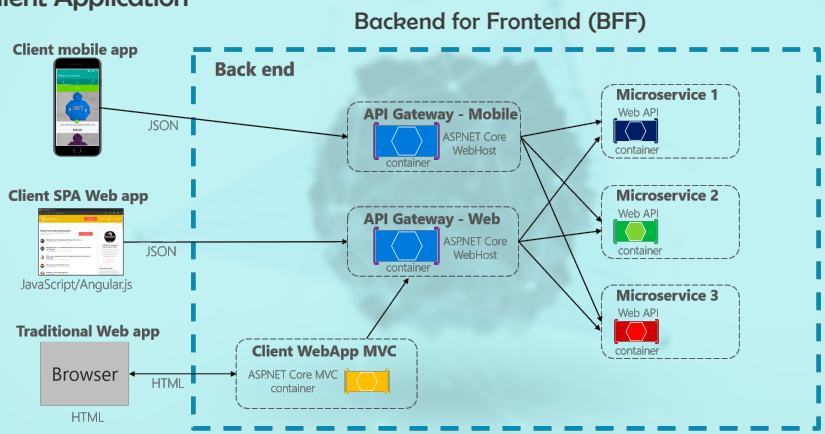
REQUEST AGGREGATION

GATEWAY OFFLOADING

49

API GATEWAY PATTERN

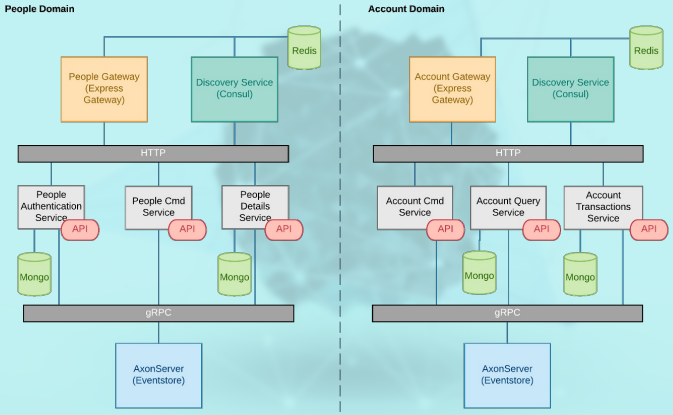
By Client Application



50

API GATEWAY PATTERN

By Application Domain

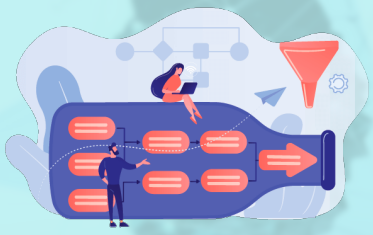


51

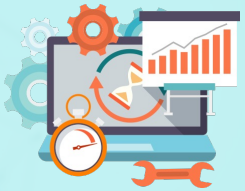
COMMUNICATION

Drawbacks of the API Gateway Pattern

SINGLE POINT OF FAILURE



PERFORMANCE IMPACTS

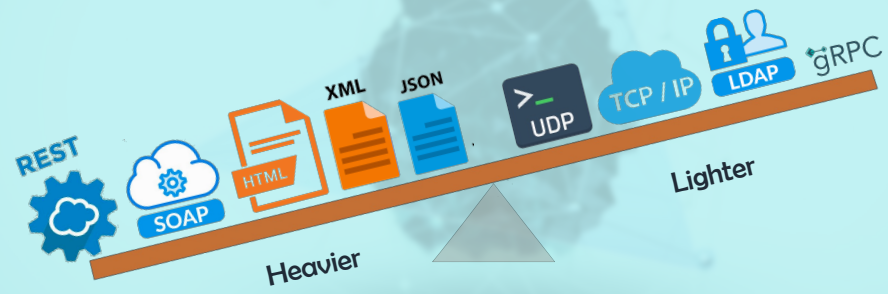


DEVELOPMENT COSTS

52

COMMUNICATION

Communication is important but it must be cheap!



53

54

EVENT SOURCING

Domain Events

Something that happened.

A withdrawal was performed.

Immutable and Immortal.

55

EVENT SOURCING

Commands vs. Events

Enter Card

Verify PIN

Withdraw Cash

Print Receipt

56

EVENT SOURCING

Commands vs. Events

Card Entered | PIN Verified | Cash Withdrawn | Receipt Printed

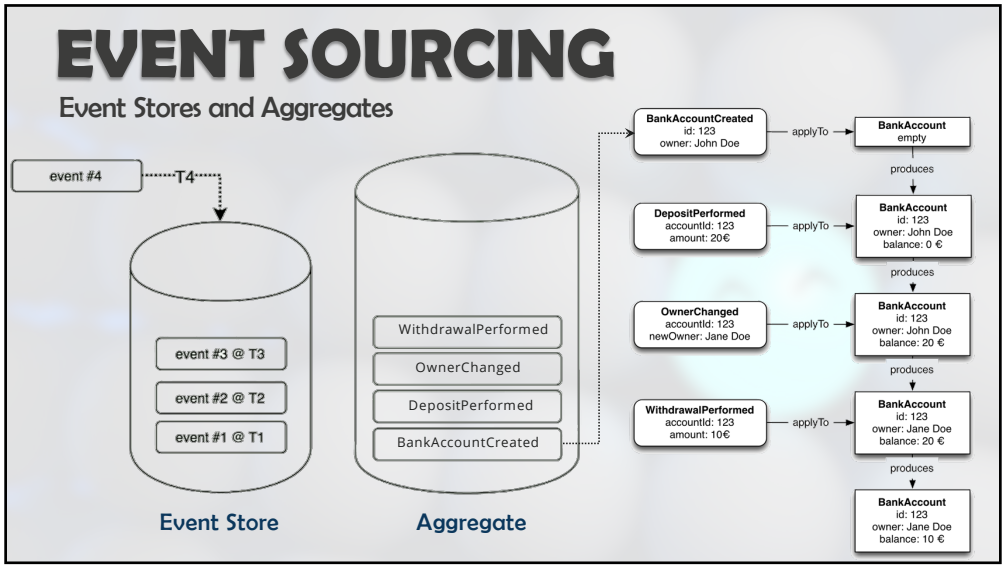
57

EVENT SOURCING

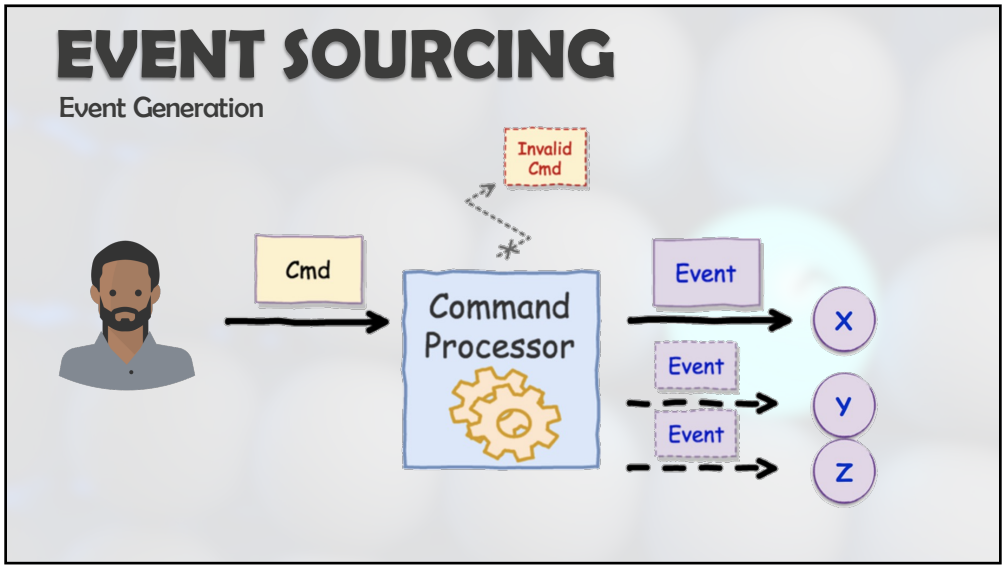
Commands vs. Events

Card Entered | PIN Verified | Cash Withdrawn | Receipt Printed

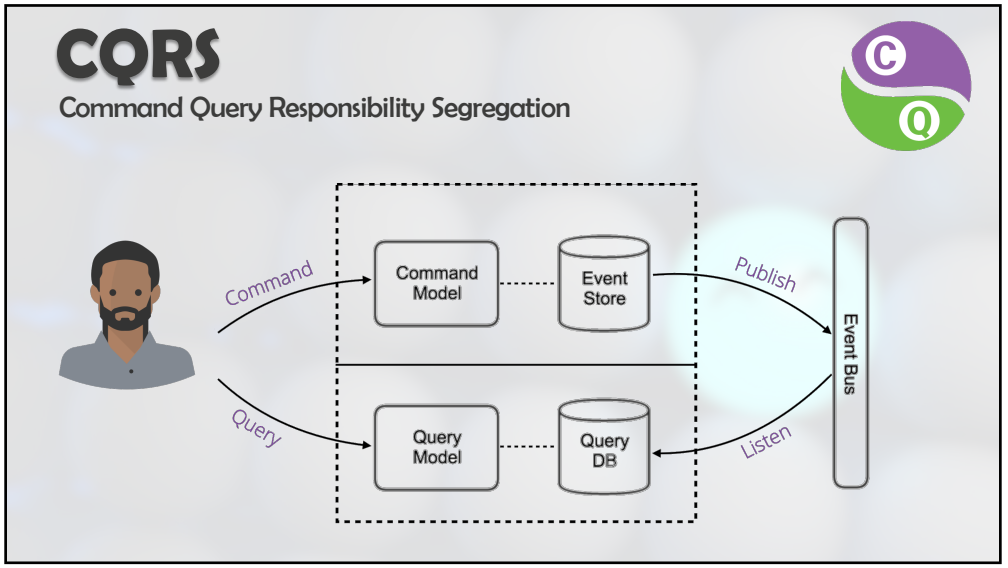
58


EVENT SOURCING
Event Stores and Aggregates
event #4
T4
event #3 @ T3
event #2 @ T2
event #1 @ T1
WithdrawalPerformed
OwnerChanged
DepositPerformed
BankAccountCreated
Event Store
Aggregate
BankAccountCreated
id: 123
owner: John Doe
applyTo
BankAccount
empty
produces
DepositPerformed
accountId: 123
amount: 20€
applyTo
BankAccount
id: 123
owner: John Doe
balance: 0 €
produces
OwnerChanged
accountId: 123
newOwner: Jane Doe
applyTo
BankAccount
id: 123
owner: John Doe
balance: 20 €
produces
WithdrawalPerformed
accountId: 123
amount: 10€
applyTo
BankAccount
id: 123
owner: Jane Doe
balance: 20 €
produces
BankAccount
id: 123
owner: Jane Doe
balance: 10 €

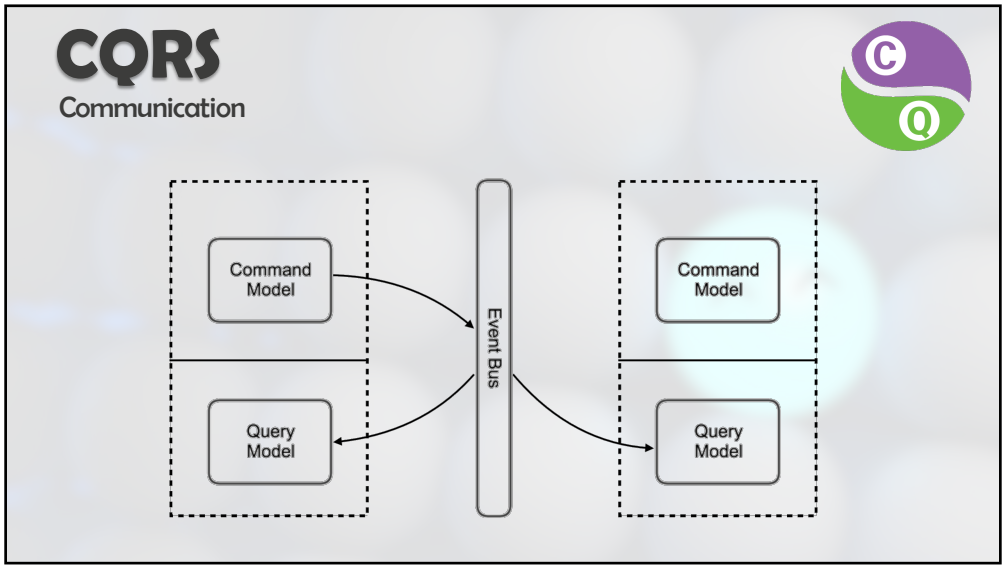
59


EVENT SOURCING
Event Generation
Invalid Cmd
Cmd
Command Processor
Event
Event
Event
X
Y
Z

60

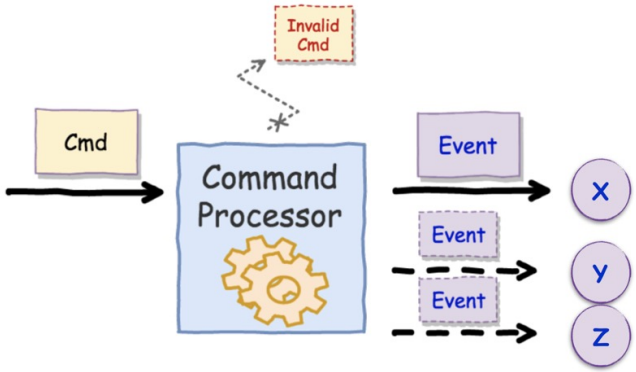

CQRS
Command Query Responsibility Segregation
C
Q
Command
Query
Command Model
Event Store
Query Model
Query DB
Publish
Listen
Event Bus

61


CQRS
Communication
C
Q
Command Model
Query Model
Event Bus
Command Model
Query Model

62

COMMANDS TO EVENTS



63

AGILE-NOT-FRAGILE MICROSERVICES DEVELOPMENT

MICROSERVICE DESIGN AND TESTABILITY

64


DEFENSIVE
NO TRUST
CONTRACT
MUTUAL AGREEMENT

65


CONTRACT
MUTUAL AGREEMENT
INTRODUCTION
Assumes that client components (consumers) that invoke operations on a server component (provider) will meet specified preconditions for that operation.
Otherwise...
=
Garbage In
Garbage Out

66


CONTRACT
MUTUAL AGREEMENT
BENEFITS & OBLIGATIONS
Benefit for consumer is an obligation for the provider, and vice-versa.
Full contract (obligation) for provider is the sum of all consumer contracts (benefits).
$obligation(p) = \sum_{i=1}^{n} benefit(c_i)$

67


PACT
[docs.pact.io]

68

PROJECT: PACT CONTRACTS

Let's Make a PACT and Run with it

SPIN UP ENVIRONMENT

START PACT BROKER

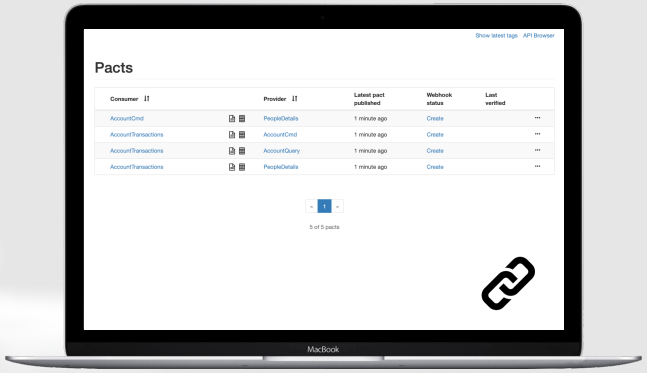
`localhost:8089`

GENERATE THE PACTS

RUN PACTS

VIEW THE RESULTS

SPIN DOWN EVERYTHING!



69

GUIDANCE

- Consumers define their own contracts.
- Provider facilitates resolution to any contradictions among consumer contracts.
- As service is updated, provider executes contract tests from all of its consumers.
- If behavior changes in a way that violates contract, provider must issue a new version of the service with a new contract.

70

GUIDANCE

TESTING THE PROVIDER

Test using valid and invalid inputs.

CONSUMER-PROVIDER INTEGRATION

Test that consumers handle all the error responses generated by providers.

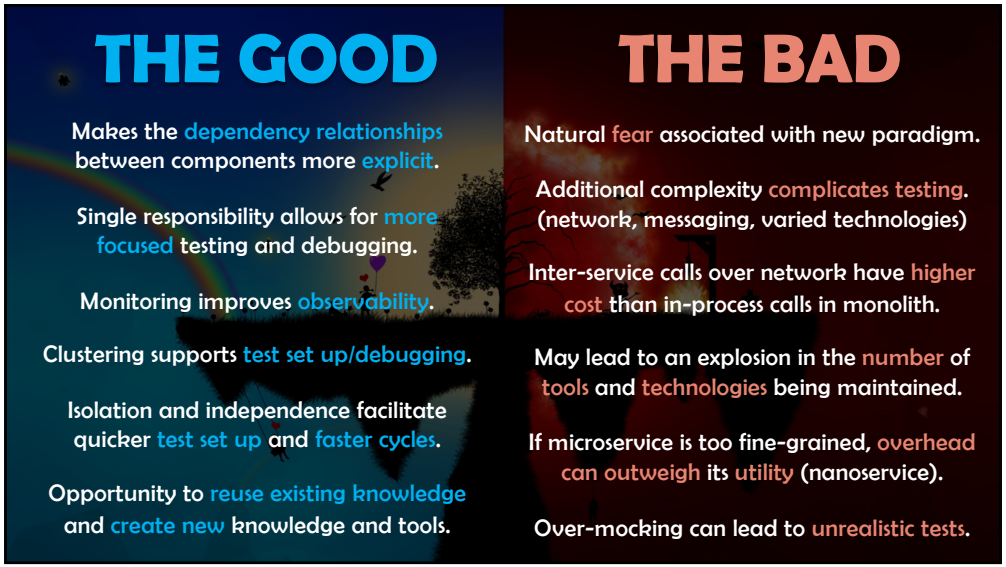

Improves Security

Code Impact?


Reduces Performance

71

72


THE GOOD
Makes the dependency relationships between components more explicit.
Single responsibility allows for more focused testing and debugging.
Monitoring improves observability.
Clustering supports test set up/debugging.
Isolation and independence facilitate quicker test set up and faster cycles.
Opportunity to reuse existing knowledge and create new knowledge and tools.
THE BAD
Natural fear associated with new paradigm.
Additional complexity complicates testing. (network, messaging, varied technologies)
Inter-service calls over network have higher cost than in-process calls in monolith.
May lead to an explosion in the number of tools and technologies being maintained.
If microservice is too fine-grained, overhead can outweigh its utility (nanoservice).
Over-mocking can lead to unrealistic tests.

73


THE REALITY
Opportunities
NEW APPROACHES, TOOLS
Containment
TEST FOCUS, DEBUGGING
Knowledge
REUSE CREATE
Overhead
COMPLEXITY, PERFORMANCE
Unanswered Questions
WHAT? WHERE? WHEN?
Explicit Relationships
INTEGRATION, PROCESS
Natural Fear
CHANGE
Isolation
MORE SEAMS
Extremism
NANOSERVICE ANTI-PATTERN, OVER-MOCKING

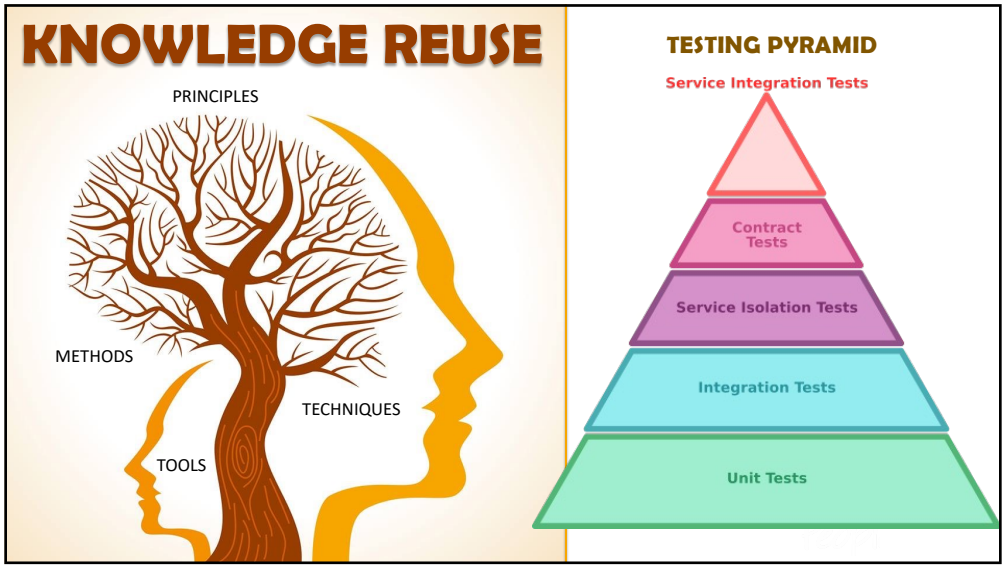
74

AGILE-NOT-FRAGILE MICROSERVICES DEVELOPMENT

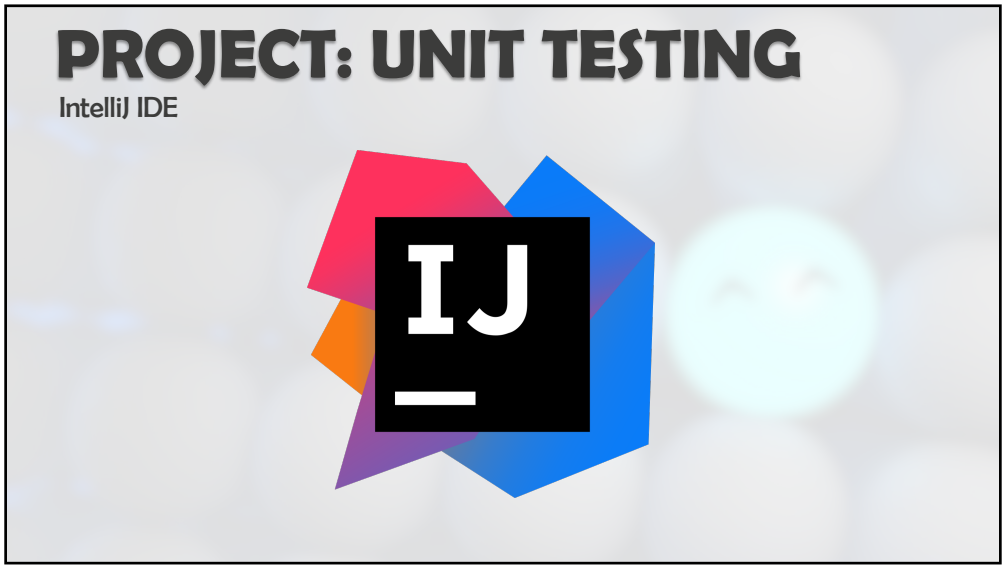
FULL-STACK TEST AUTOMATION

75

76


KNOWLEDGE REUSE
PRINCIPLES
METHODS
TECHNIQUES
TOOLS
TESTING PYRAMID
Service Integration Tests
Contract Tests
Service Isolation Tests
Integration Tests
Unit Tests

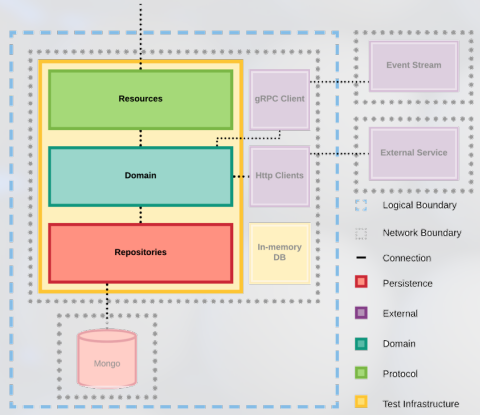
77


PROJECT: UNIT TESTING
IntelliJ IDE
IJ

78

PROJECT: UNIT TESTING

Class as the Unit: Testing Resources, Domain and Repository Classes in Isolation



JUnit 5

mockito

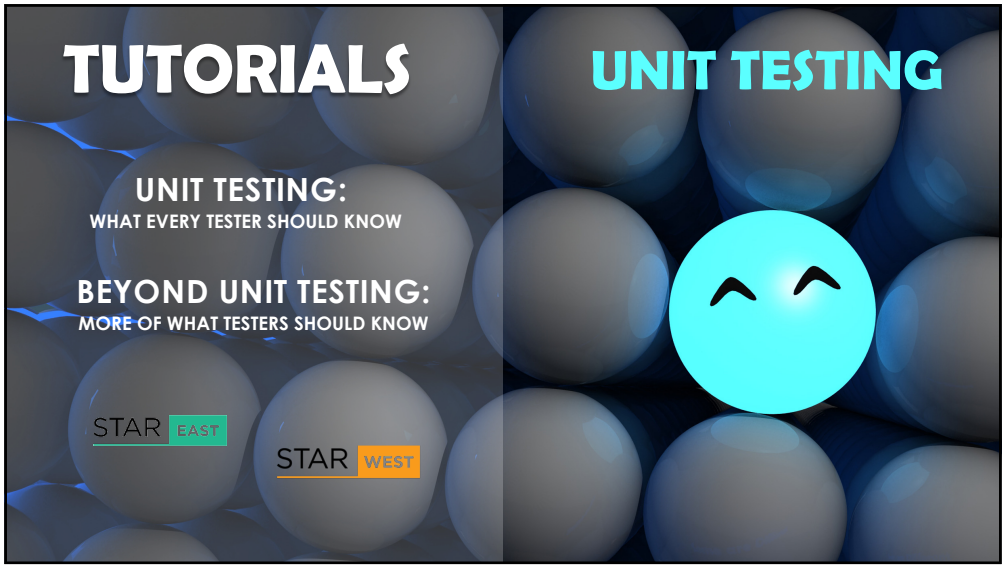
JACOCO
Java Code Coverage

79

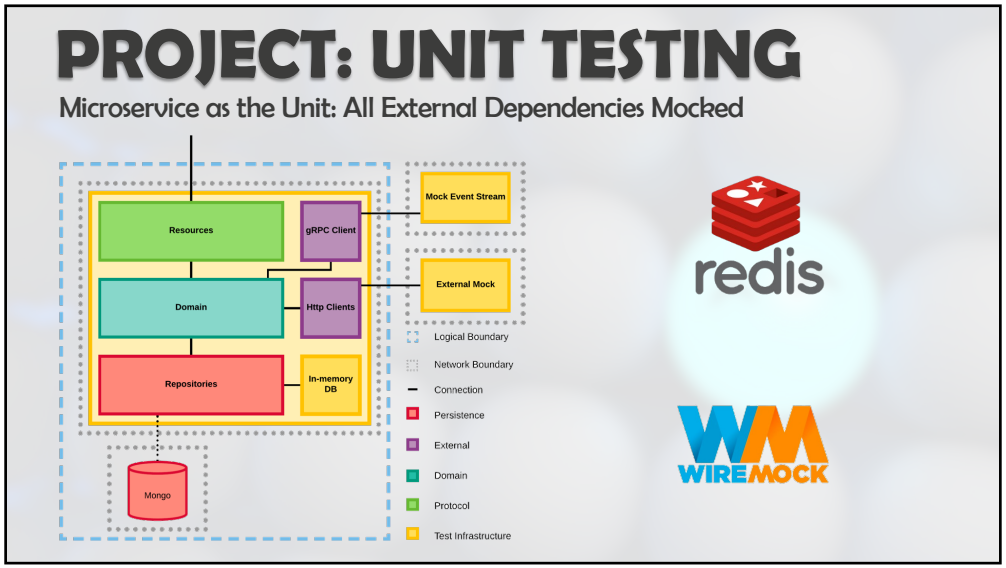
GUIDELINES

1. Know what you are testing.
2. Test results not implementation.
3. Test one thing at a time.
4. Make tests readable and understandable.
5. Make tests independent and self-sufficient.
6. Make tests deterministic.
7. Repeat yourself when necessary.
8. Measure code coverage but focus on test coverage.

80


TUTORIALS
UNIT TESTING:
WHAT EVERY TESTER SHOULD KNOW
BEYOND UNIT TESTING:
MORE OF WHAT TESTERS SHOULD KNOW
STAR EAST
STAR WEST
UNIT TESTING

81

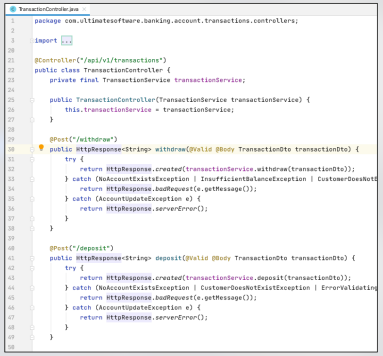

PROJECT: UNIT TESTING
Microservice as the Unit: All External Dependencies Mocked
Resources
gRPC Client
Domain
Http Clients
Repositories
In-memory DB
Mongo
Mock Event Stream
External Mock
Logical Boundary
Network Boundary
Connection
Persistence
External
Domain
Protocol
Test Infrastructure
redis
WIREMOCK

82

PROJECT: UNIT TESTING

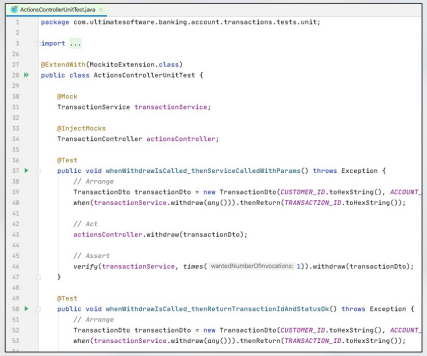
Class-Level Solitary Unit Test for Account Transactions Service

TransactionsController.java



Class Under Test

ActionsControllerUnitTest.java



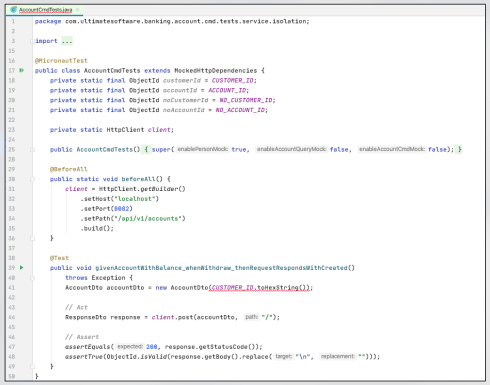
Test Driver Class

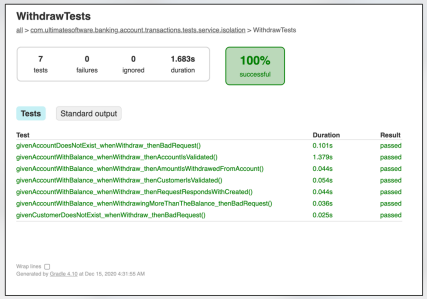
83

PROJECT: UNIT TESTING

Microservice as the Unit

AccountCmdTests.java



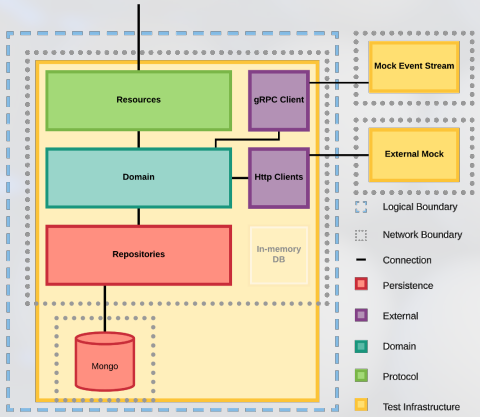


... /reports/tests/test/ ...

84

PROJECT: UNIT TESTING

Microservice as the Unit: Same Kind of Tests, Different Configuration



mongoDB

AxonIQ

WIREMOCK

85

PROJECT: UNIT TESTING

Microservice as the Unit: Testing Event-Driven Services

```
@Test
public void onAccountDelete_WhenPositiveBalance_NoEventsSent() {

    CreateAccountCommand createCommand = new CreateAccountCommand(customerId);
    CreditAccountCommand creditCommand = new CreditAccountCommand(createCommand.getId(), 100.0, transactionId);
    DeleteAccountCommand command = new DeleteAccountCommand(createCommand.getId());
    fixture.givenCommands(createCommand, creditCommand)
            .when(command)
            .expectNoEvents();
}
```

JUnit 5

AxonFramework

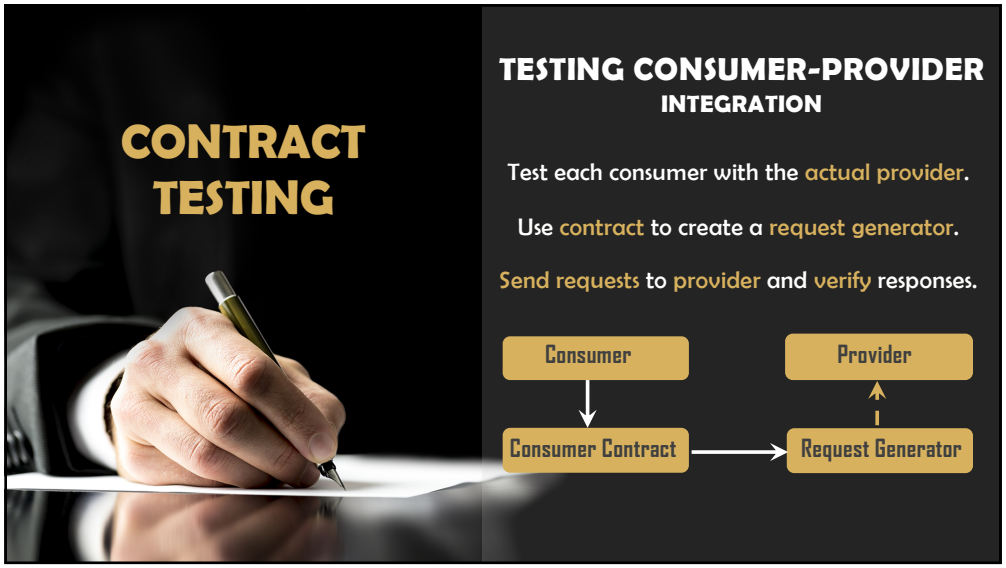
86


CONTRACT TESTING
TESTING THE PROVIDER
IN ISOLATION
Focus on checking that valid inputs produce correct responses.
No need to test invalid inputs.
=
Garbage In
Garbage Out

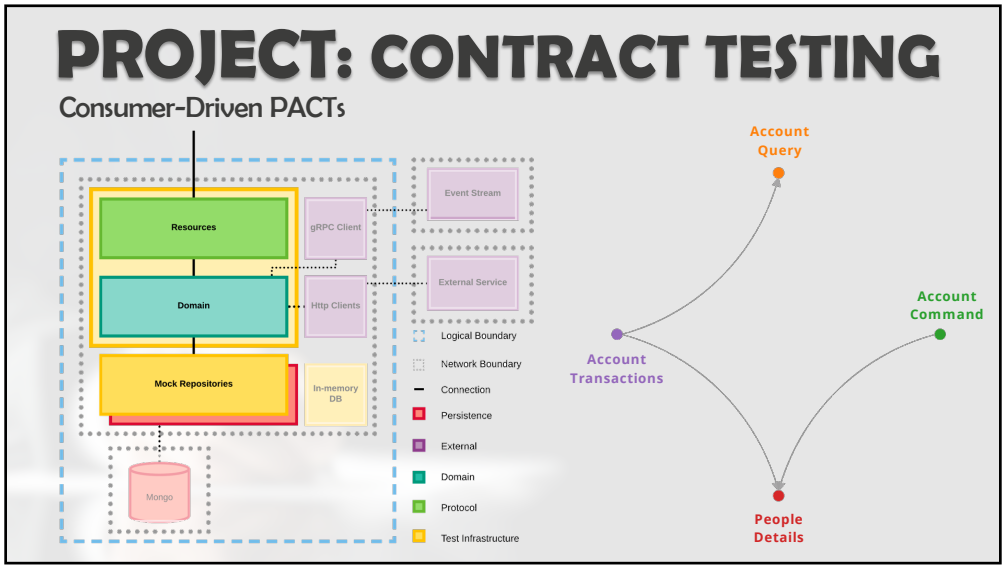
87


CONTRACT TESTING
TESTING THE CONSUMER
IN ISOLATION
Test that each consumer respects its contract.
Use contract to create a stub of the provider.
Invoke consumer behavior that calls the stub.
Consumer
Provider Stub
Consumer Contract

88


CONTRACT TESTING
TESTING CONSUMER-PROVIDER INTEGRATION
Test each consumer with the actual provider.
Use contract to create a request generator.
Send requests to provider and verify responses.
Consumer
Provider
Consumer Contract
Request Generator

89


PROJECT: CONTRACT TESTING
Consumer-Driven PACTs
Resources
Domain
Mock Repositories
Mongo
gRPC Client
Http Clients
In-memory DB
Event Stream
External Service
Logical Boundary
Network Boundary
Connection
Persistence
External
Domain
Protocol
Test Infrastructure
Account Query
Account Command
Account Transactions
People Details

90

PROJECT: CONTRACT TESTING

Let's Take a Look at How it Works

Environment Configuration

```yaml
version: '2.1'
services:
  peopleauthentication:
    build: domain-services/people-authentication/
    mem_limit: 400m
    ports:
      - "8888:8888"
    restart: on-failure
    environment:
      - MICRONAUT_ENVIRONMENTS=test,internal_mocks
      - HOST_PORT=8888
      - ES_JAVA_OPTS= "-Xms450mb -Xmx450mb"

  peoplecmd:
    build: domain-services/people-cmd/
    mem_limit: 400m
    ports:
      - "8887:8887"
    restart: on-failure
    environment:
      - MICRONAUT_ENVIRONMENTS=test,internal_mocks
      - HOST_PORT=8887
      - ES_JAVA_OPTS= "-Xms450mb -Xmx450mb"

  peopledetails:
    build: domain-services/people-details/
    mem_limit: 400m
    ports:
      - "8885:8885"
    restart: on-failure
    environment:
      - MICRONAUT_ENVIRONMENTS=test,internal_mocks
      - HOST_PORT=8885
      - ES_JAVA_OPTS= "-Xms450mb -Xmx450mb"
```

Mock Repositories

```
public MockCustomerRepository() {
    entities = new ArrayList<>();
    entities.add(new PersonDetails(new ObjectId( hexString: "5c86d84877970c1fd879a36b"), firstName: "Jack", lastName: "Oneill"));
    entities.add(new PersonDetails(new ObjectId( hexString: "5c892dbef72465ad7e7dde42"), firstName: "Samantha", lastName: "Carter"));
    entities.add(new PersonDetails(new ObjectId( hexString: "5c89342ef72465c5981bc1fc"), firstName: "Daniel", lastName: "Jackson"));
}
```

PACT Generation and Behavioral Verification

```
@ExtendWith(PactConsumerTestExt.class)
@PactTestFor(providerName = "PeopleDetails", port = "8885")
public class ConsumeSinglePersonDetails {
    @Pact(provider = "PeopleDetails", consumer = "AccountCmd")
    public RequestResponsePact createGetOnePact(PactDslWithProvider builder) {
        return builder
                .given( state: "A person exist.") PactDslWithState
                .uponReceiving( description: "Single valid person") PactDslRequestWithoutPath
                .path("/api/v1/people/5c892dbef72465ad7e7dde42") PactDslRequestWithPath
                .method("GET")
                .headers(getHeaders())
                .willRespondWith() PactDslResponse
                .status(200)
                .body("{\"id\":\"5c892dbef72465ad7e7dde42\",\"firstName\":\"Samantha\",\"lastName\":\"Carter\"}")
                .toPact();
    }

    @Test
    void testGetOne(MockServer mockServer) throws IOException {
        given()
                .headers(getHeaders()).
                when()
                .get( path: mockServer.getUrl() + "/api/v1/people/5c892dbef72465ad7e7dde42").
                then()
                .statusCode(200);
    }
```

MICRONAUT

PACT

REST-assured

91

PROJECT: CONTRACT TESTING

Try it out for yourself!

Contract Testing with PACT

Given all the moving parts and communication between these parts in microservices, contract tests provide valuable, fast feedback. Ideally, not only are all API methods covered, but also different rejection cases, such as bad requests or server errors. For this project, PACT is used as the contract testing framework. PACT is one of the most popular and mature contract testing frameworks which provides plug-ins and libraries for a range of languages.

Consumers Create Contracts

Each consumer creates a contract with a provider. In this example, we are going to develop a contract for the deposit action between the Account Transactions consumer and the Account Command. The contract are stored as JSON files like the one below:

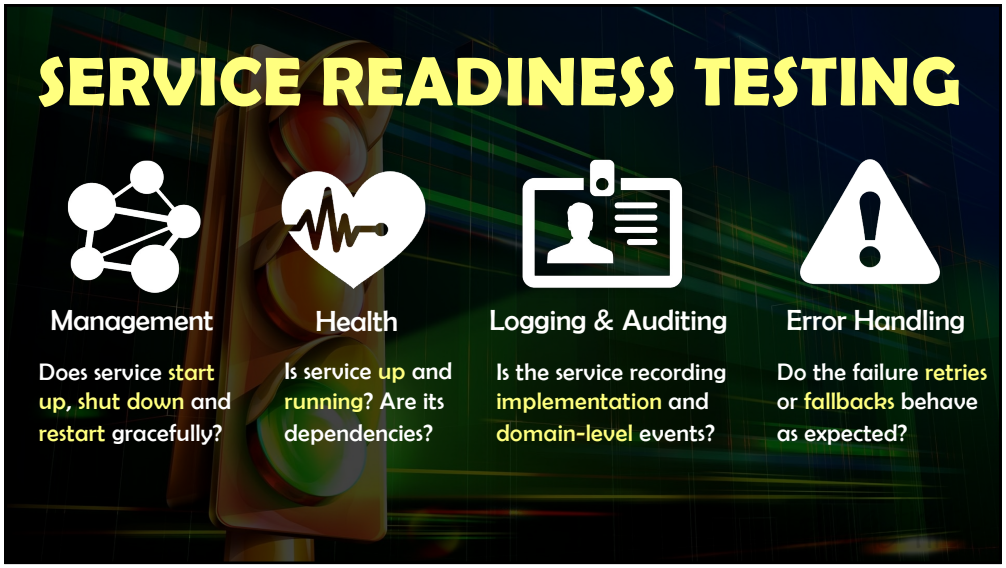
Example of a PACT Contract

```
{
  "state": "I have a picture that can be downloaded",
  "uponReceiving": "a request to download some-file",
  "withRequest": {
      "method": "GET",
      "path": "/download/somefile"
  },
  "willRespondWith": {
      "status": 200,
      "headers":
      {
          "Content-disposition": "attachment; filename=some-file.jpg"
      }
  }
}
```

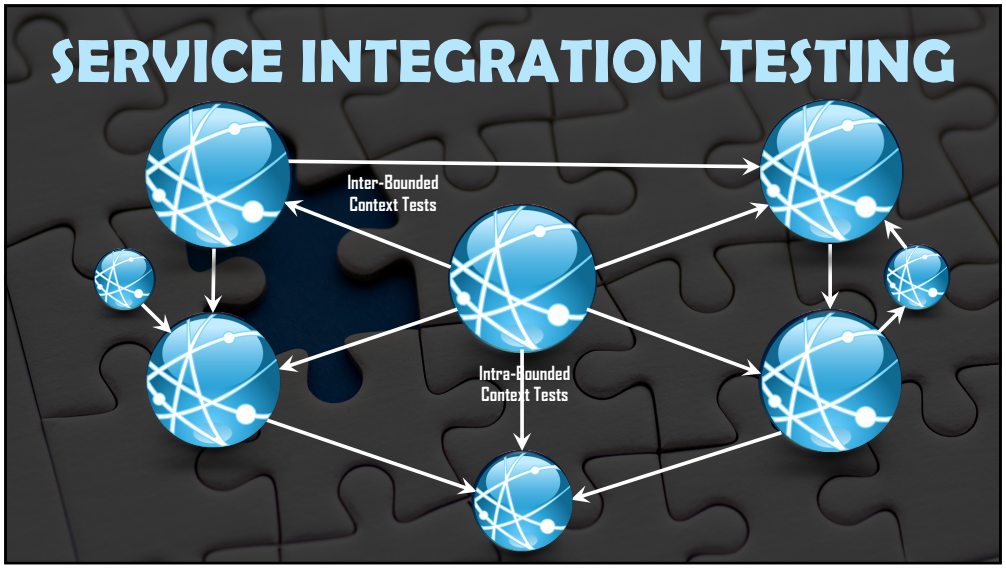
PACT

GUIDE

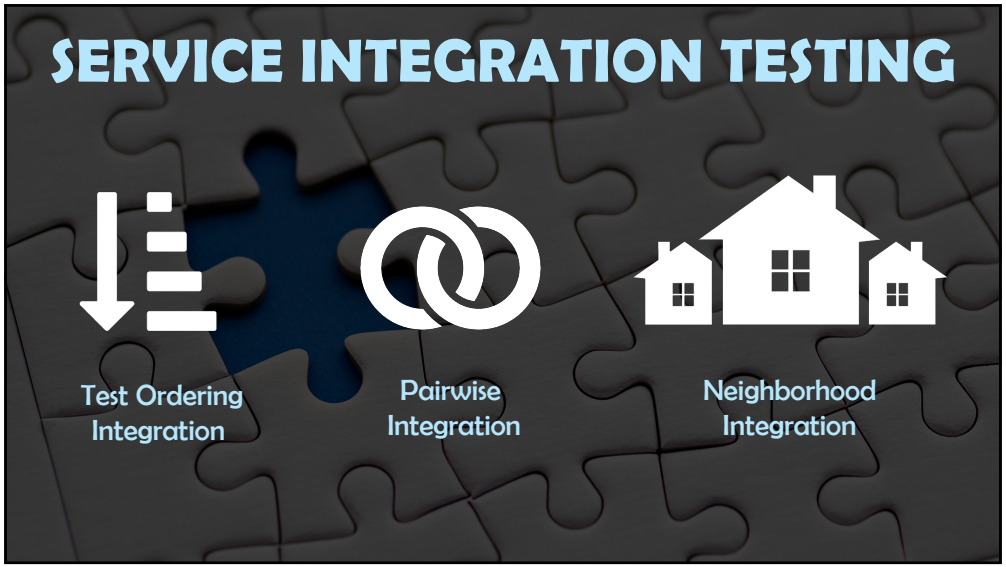
92


SERVICE READINESS TESTING
Management
Does service start up, shut down and restart gracefully?
Health
Is service up and running? Are its dependencies?
Logging & Auditing
Is the service recording implementation and domain-level events?
Error Handling
Do the failure retries or fallbacks behave as expected?

93


SERVICE INTEGRATION TESTING
Inter-Bounded Context Tests
Intra-Bounded Context Tests

94


SERVICE INTEGRATION TESTING
Test Ordering Integration
Pairwise Integration
Neighborhood Integration

95


INTEGRATION TEST ORDERING

96

INTEGRATION TEST ORDERING

Seeks to find the "optimal" order for integration testing that reduces the cost of creating stubs.

Assumption:

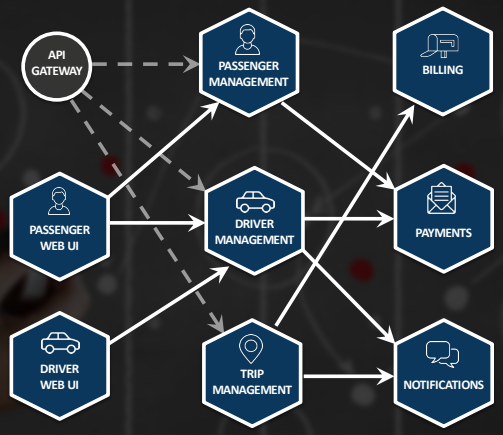
- Developing stubs is time-consuming and error prone.

Techniques:

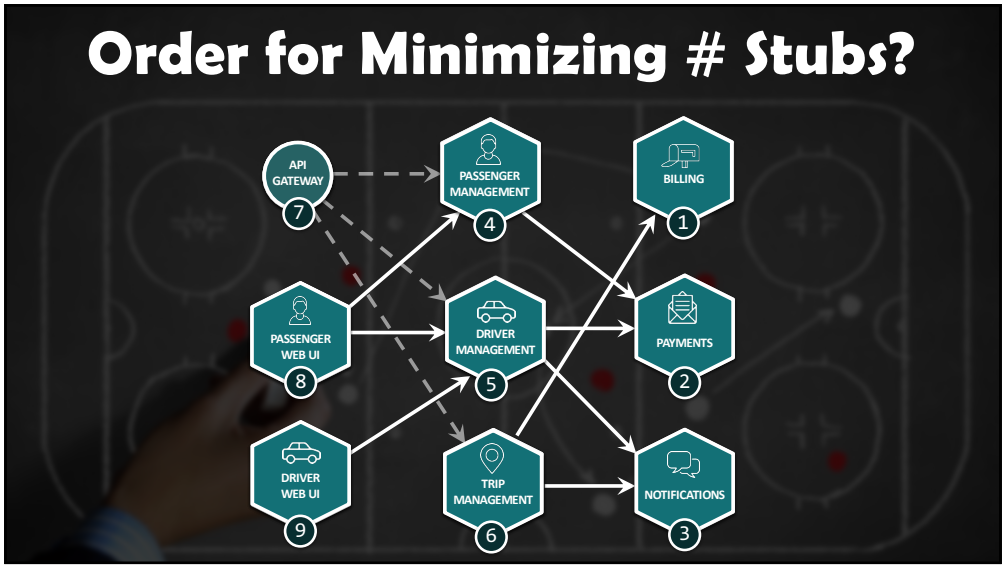
- Minimize the number of stubs
- Minimize the overall stub complexity.

97

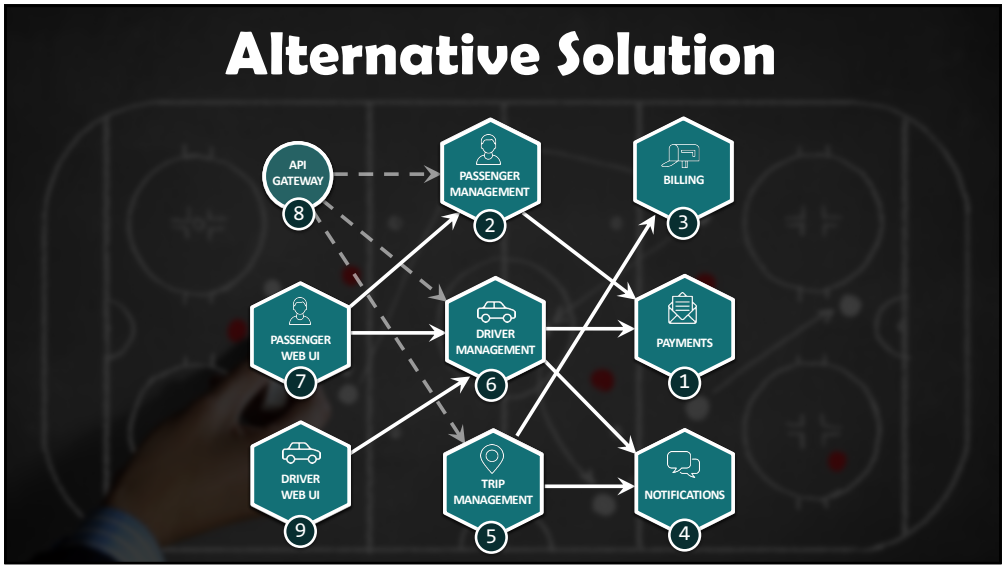
Order for Minimizing # Stubs?



98


Order for Minimizing # Stubs?
API GATEWAY
7
PASSENGER MANAGEMENT
4
BILLING
1
PASSENGER WEB UI
8
DRIVER MANAGEMENT
5
PAYMENTS
2
DRIVER WEB UI
9
TRIP MANAGEMENT
6
NOTIFICATIONS
3

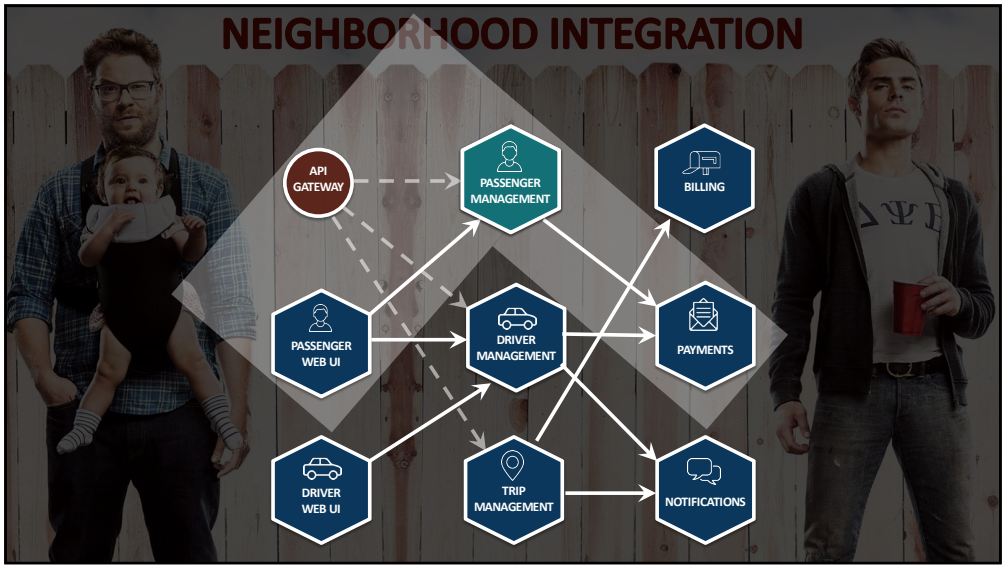
99


Alternative Solution
API GATEWAY
8
PASSENGER MANAGEMENT
2
BILLING
3
PASSENGER WEB UI
7
DRIVER MANAGEMENT
6
PAYMENTS
1
DRIVER WEB UI
9
TRIP MANAGEMENT
5
NOTIFICATIONS
4

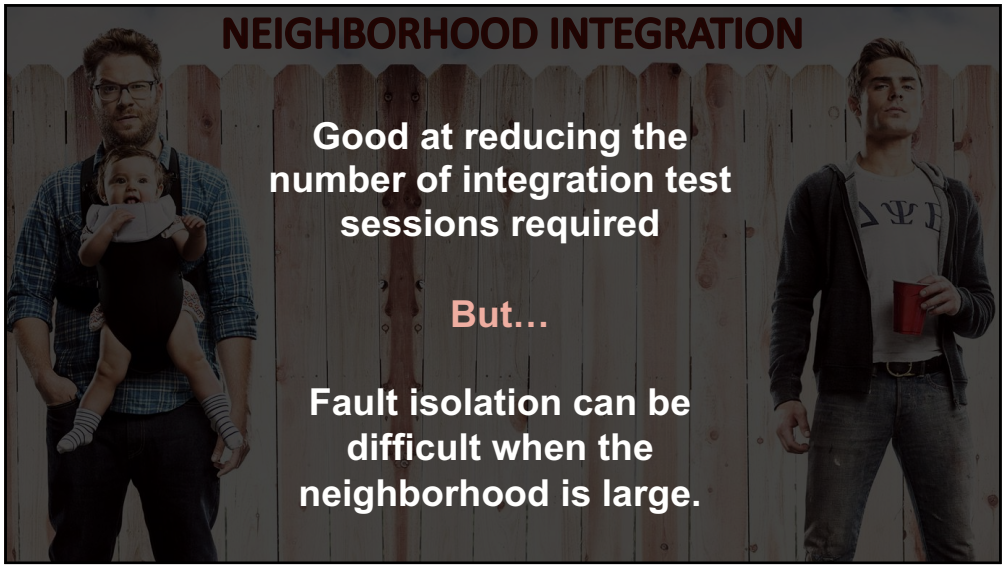
100


NEIGHBORHOOD INTEGRATION
The neighborhood of a service is the set of services that are one edge away from the given service.
That is, all the immediate predecessor services and immediate successor services of a given service.

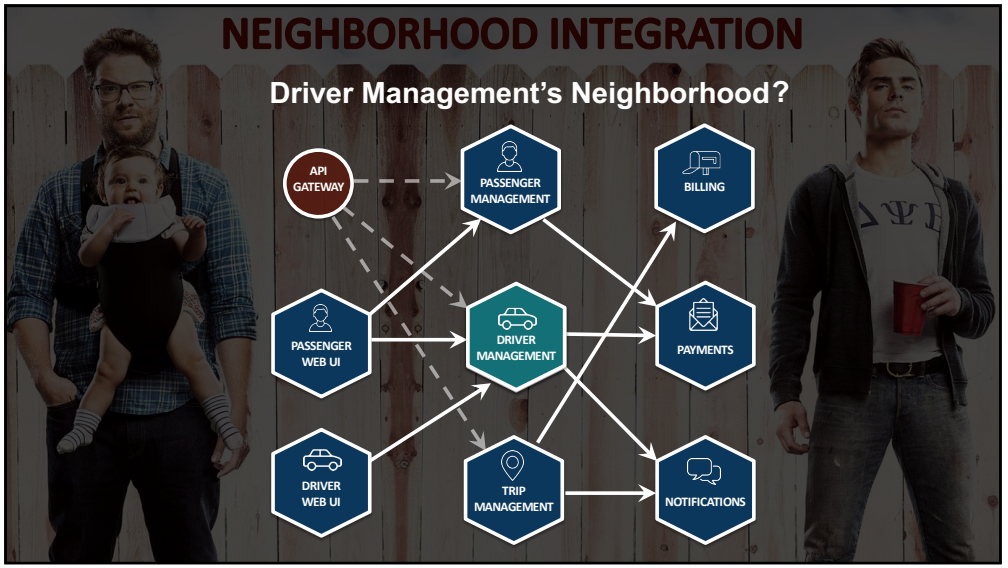
101


NEIGHBORHOOD INTEGRATION
API GATEWAY
PASSENGER MANAGEMENT
BILLING
PASSENGER WEB UI
DRIVER MANAGEMENT
PAYMENTS
DRIVER WEB UI
TRIP MANAGEMENT
NOTIFICATIONS

102


NEIGHBORHOOD INTEGRATION
Good at reducing the number of integration test sessions required
But…
Fault isolation can be difficult when the neighborhood is large.

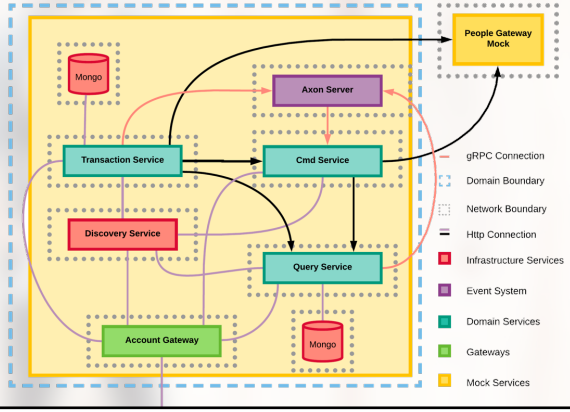
103


NEIGHBORHOOD INTEGRATION
Driver Management's Neighborhood?
API GATEWAY
PASSENGER MANAGEMENT
BILLING
PASSENGER WEB UI
DRIVER MANAGEMENT
PAYMENTS
DRIVER WEB UI
TRIP MANAGEMENT
NOTIFICATIONS

104

PROJECT: SERVICE INTEGRATION

Sub-Domain as a Neighborhood

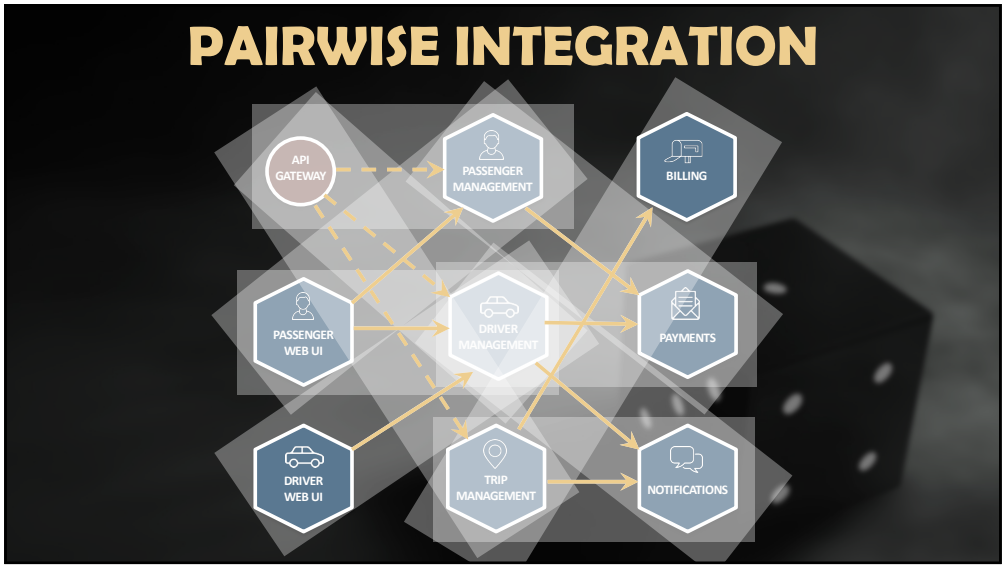


105

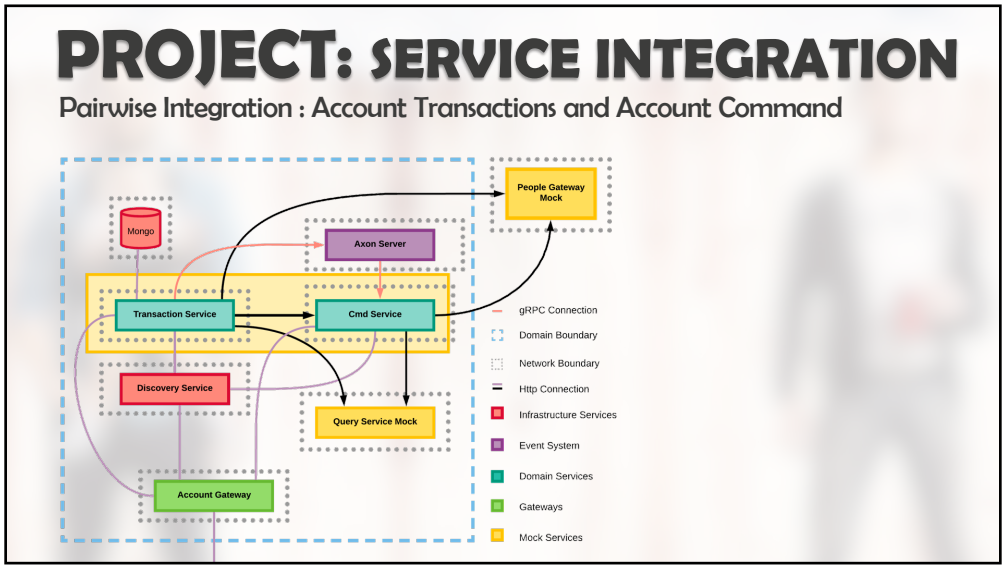
PAIRWISE INTEGRATION

- Limit testing to each pair of services.
- Results in one integration test for each edge in call graph.
- You may eliminate the need for stubs and drivers by using the actual code instead.

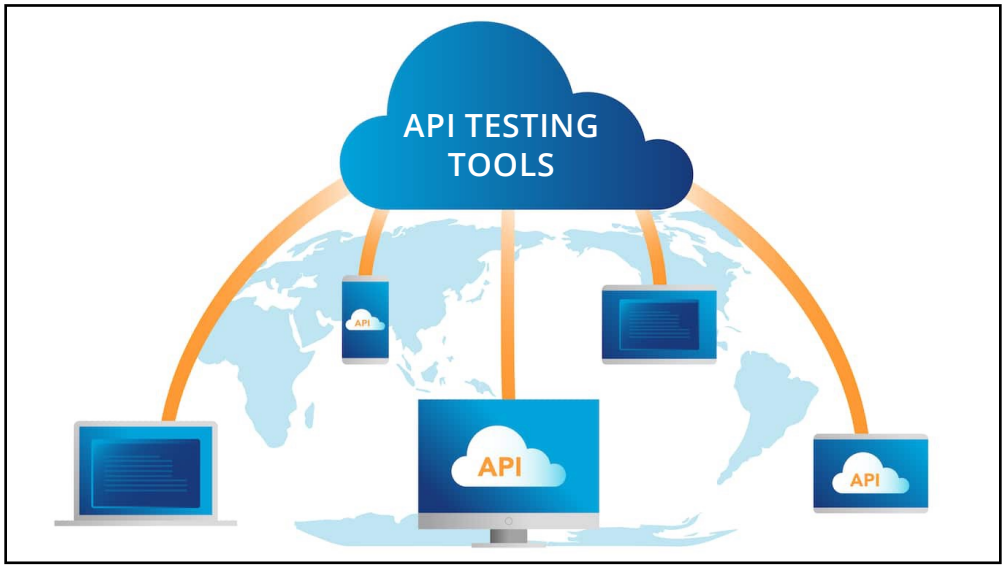
106


PAIRWISE INTEGRATION
API GATEWAY
PASSENGER MANAGEMENT
BILLING
PASSENGER WEB UI
DRIVER MANAGEMENT
PAYMENTS
DRIVER WEB UI
TRIP MANAGEMENT
NOTIFICATIONS

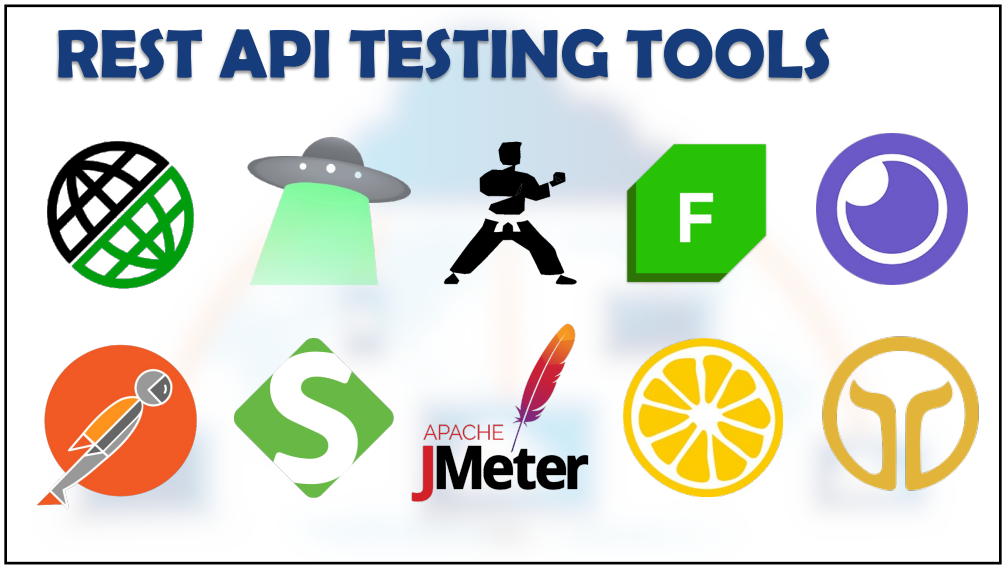
107


PROJECT: SERVICE INTEGRATION
Pairwise Integration : Account Transactions and Account Command
Mongo
Axon Server
People Gateway Mock
Transaction Service
Cmd Service
Discovery Service
Query Service Mock
Account Gateway
gRPC Connection
Domain Boundary
Network Boundary
Http Connection
Infrastructure Services
Event System
Domain Services
Gateways
Mock Services

108


API TESTING
TOOLS
API
API
API

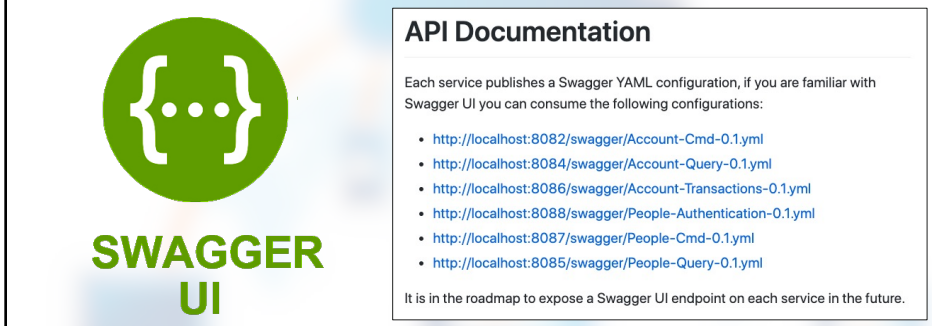
109


REST API TESTING TOOLS
F
APACHE
JMeter

110

REST API TESTING TOOLS

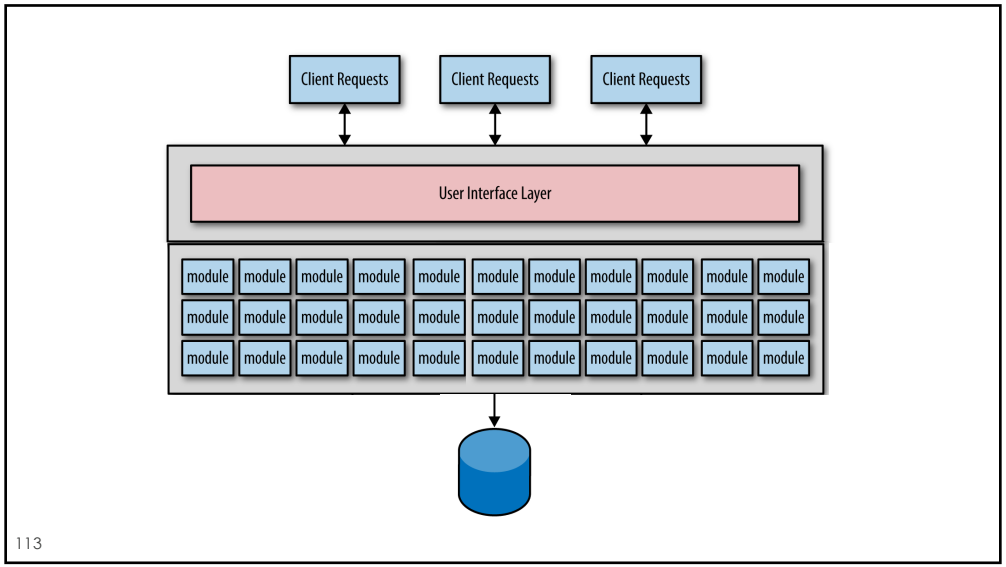
SWAGGER UI



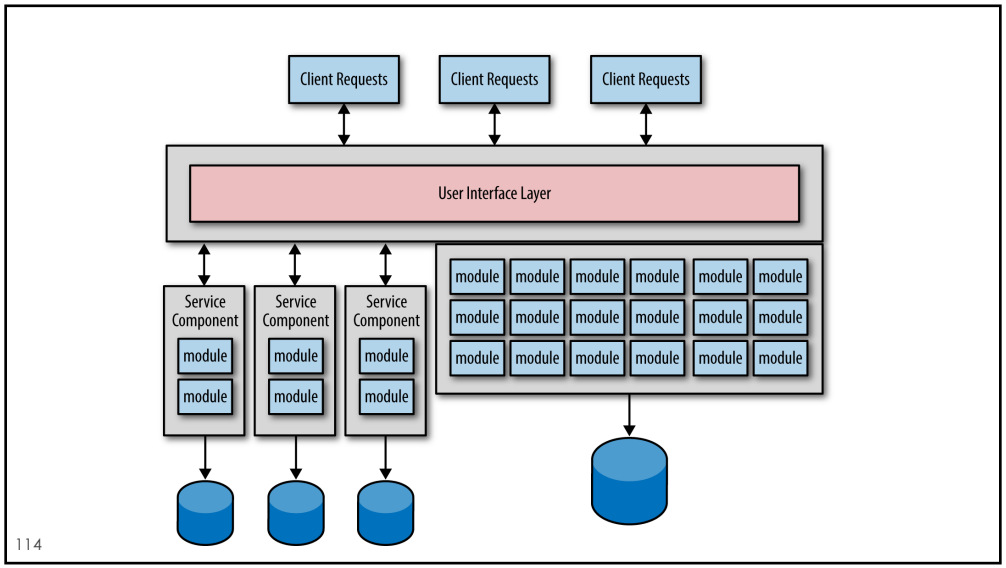
111

CHALLENGES, PITFALLS & ANTIPATTERNS

112


Client Requests
Client Requests
Client Requests
User Interface Layer
module
113

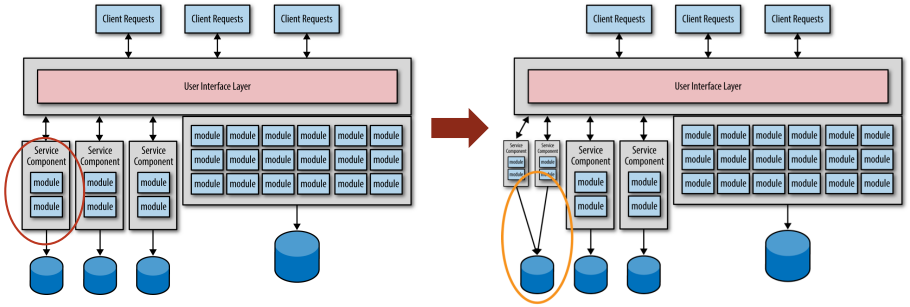
113


Client Requests
Client Requests
Client Requests
User Interface Layer
Service Component
Service Component
Service Component
module
114

114

CHALLENGE #1: SIZING

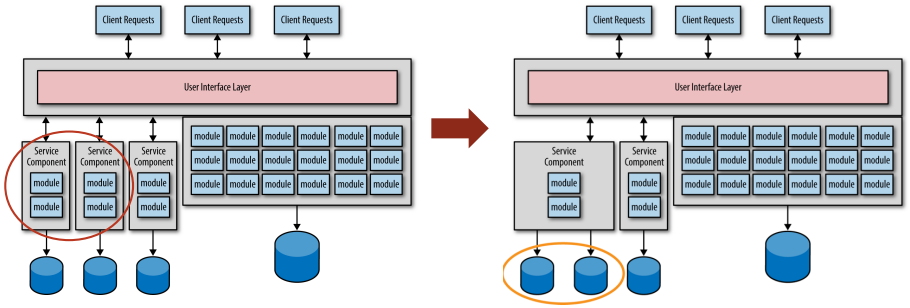
First attempt at service decomposition may be too coarse grained...



115

CHALLENGE #1: SIZING

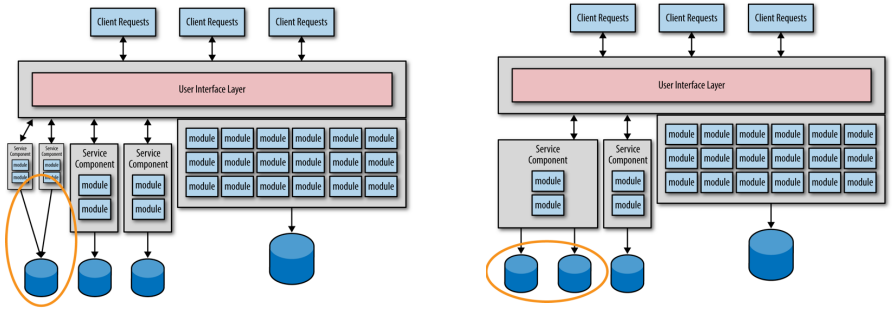
Or services may be too fine-grained...



116

PITFALL #1:

Data Migration Anti-Pattern: Too Many Data Migrations

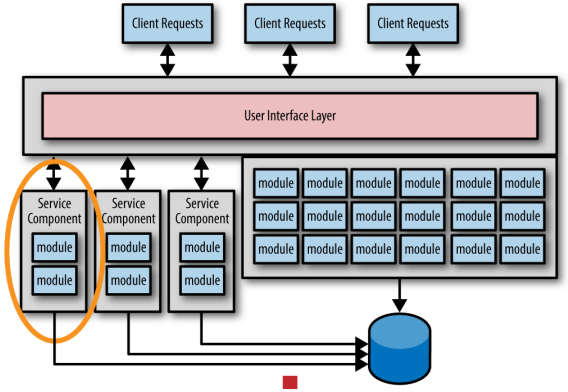


117

117

AVOIDING PITFALL #1

Migrate Functionality First, Data Last

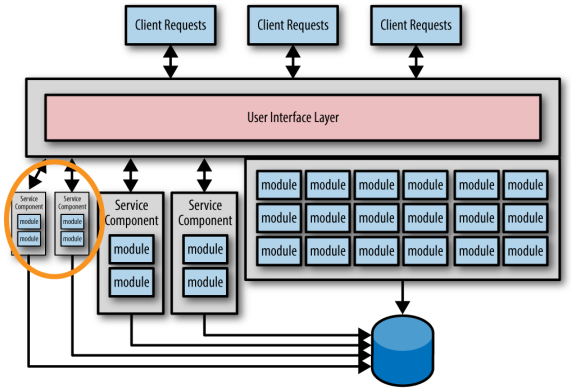


118

118

AVOIDING PITFALL #1

Migrate Functionality First, Data Last

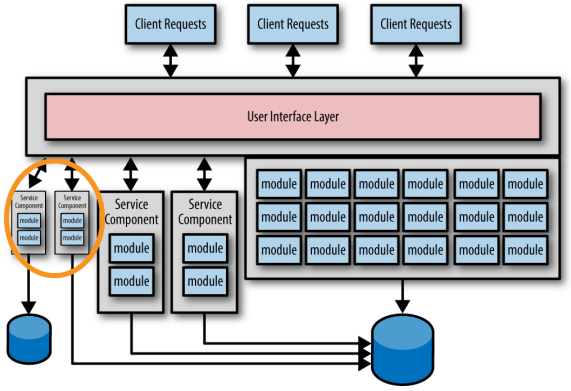


119

119

AVOIDING PITFALL #1

Migrate Functionality First, Data Last

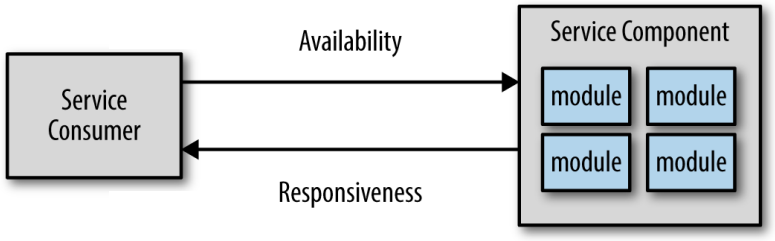


120

120

CHALLENGE #2

Managing Service Availability and Responsiveness

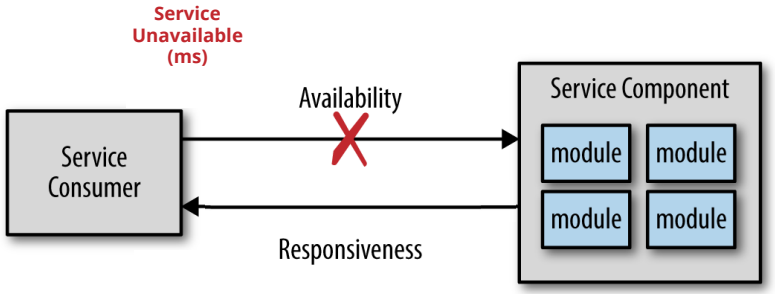


121

121

CHALLENGE #2

Consumer is unable to connect with service

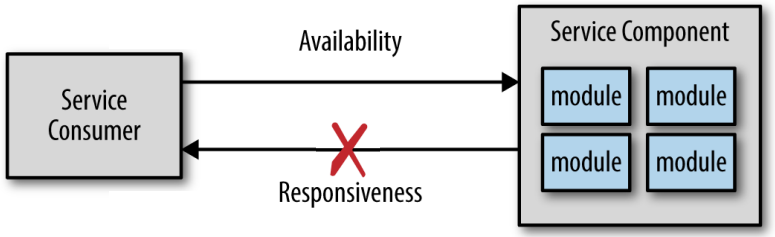


122

122

CHALLENGE #2

Consumer can connect, but service doesn't respond

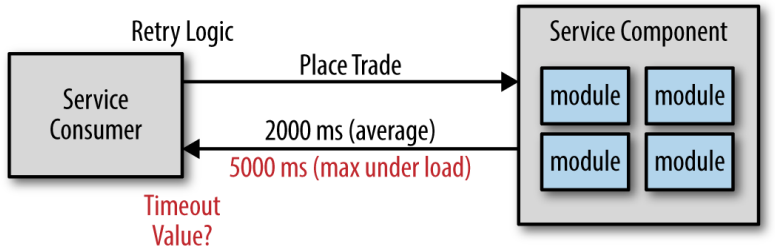


123

PITFALL #2

Timeout Anti-Pattern

Apple Inc
NASDAQ: AAPL

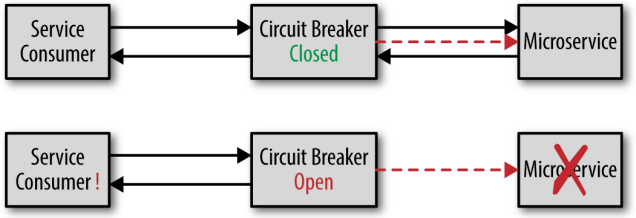


Timeout Value = 5000ms x 2 = 10000ms

124

AVOIDING PITFALL #2

Circuit Breaker Pattern



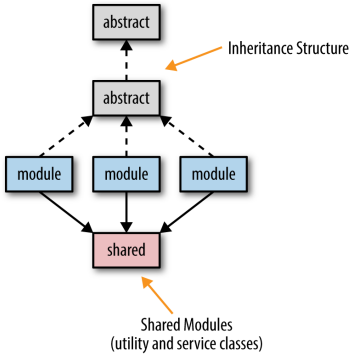
HYSTRIX

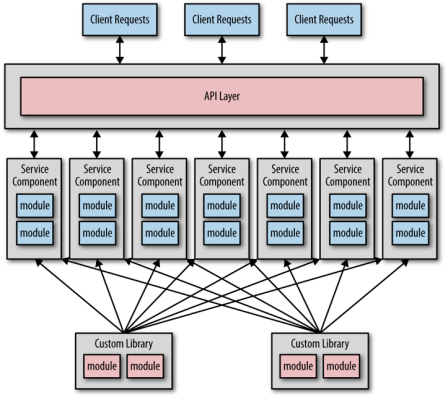
akka

125

CHALLENGE #3

Sharing and Managing Shared Dependencies

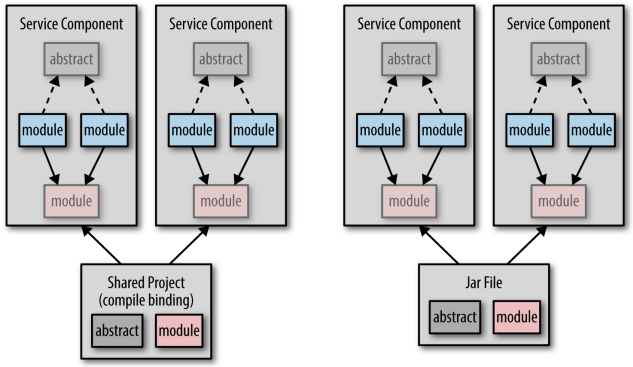




126

AVOIDING PITFALL #3

Techniques for Sharing



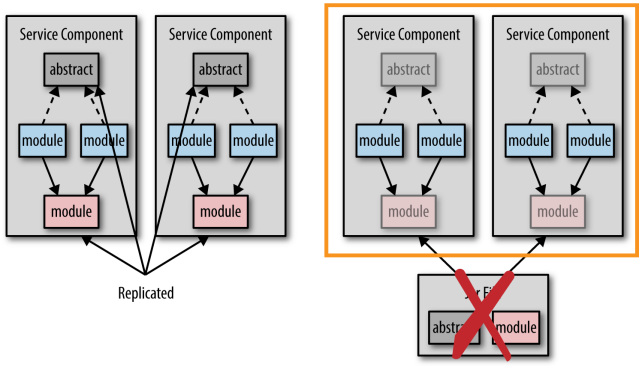
Shared Project

Shared Library

127

AVOIDING PITFALL #3

Techniques for Sharing (cont'd)



128

Replication

Consolidation

AGILE-NOT-FRAGILE MICROSERVICES DEVELOPMENT

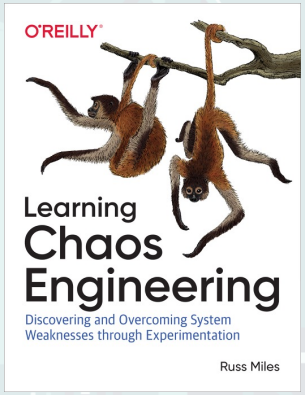
RESILIENCY THROUGH CHAOS ENGINEERING

129

130

CHAOS IN PRACTICE

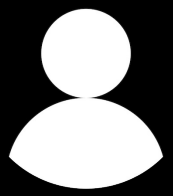
Resources



ChaosToolkit

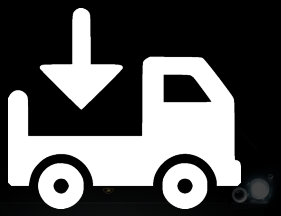
131

SYSTEM TESTING



End User Testing

Functional,
User Experience



Performance Testing

Load, Scale
Stable, Reliable

Chaos and Stress

Robust, Secure,
Resilient

132

133

THE TWELVE-FACTOR APP

[https://12factor.net/]

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

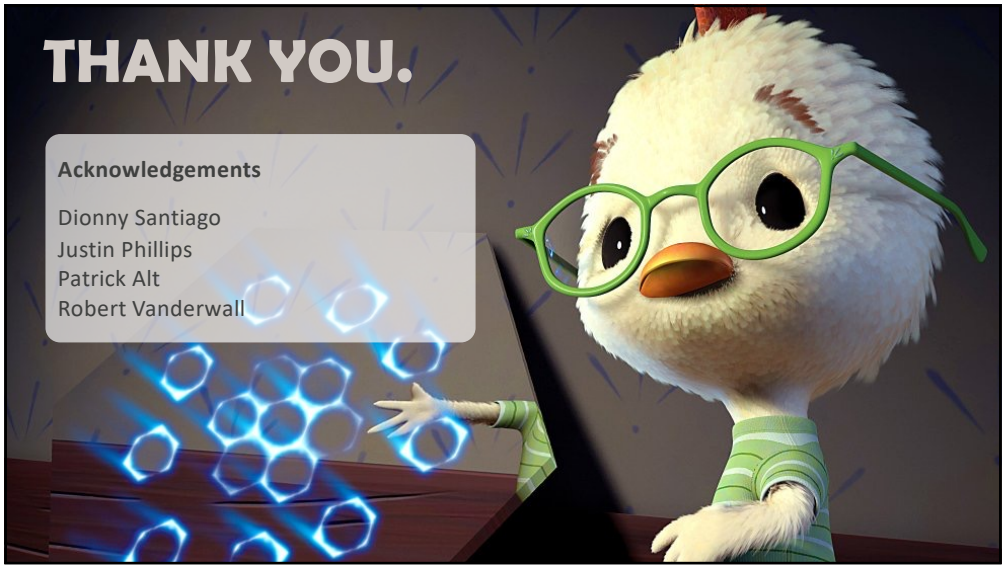
XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

134


Gatling
LOAD TESTING
REST-assured
APACHE
JMeter
OWASP
ZAP
PACT
BDD-Security
Hoverfly
CHAOS MONKEY
WM

135


THANK YOU.
Acknowledgements
Dionny Santiago
Justin Phillips
Patrick Alt
Robert Vanderwall

136