The Complete Microservices & Event-Driven Architecture

By Michael Pogrebinsky

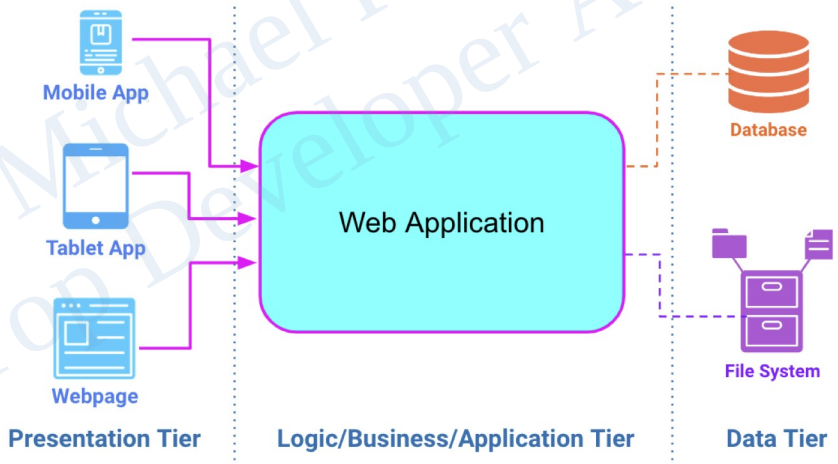
Copyright Notice

Introduction **2**
Introduction to Microservices and Event-Driven Architecture 2
Microservices Architecture - Benefits and Challenges 3
Migration to Microservices Architecture **4**
Microservices Boundaries - Core Principles 4
Decomposition of a Monolithic Application to Microservices 6
Migration to Microservices - Steps, Tips and Patterns 7
Microservices - Principles and Best Practices **8**
Databases in Microservices Architecture 8
The DRY Principle In Microservices and Shared Libraries 9
Structured Autonomy for Development Teams 10
Micro-frontends Architecture Pattern 11
Event-Driven Architecture **12**
Introduction to Event-Driven Architecture 12
Use Cases and Patterns of Event-Driven Architecture 13
Message Delivery Semantics in Event-Driven Architecture 14
Event-Driven Microservices - Design Patterns **15**
Saga Pattern 15
CQRS Pattern 16
Event Sourcing Pattern 17
Testing Microservices and Event-Driven Architecture **18**
Testing Pyramid for Microservices - Introduction and Challenges 18
Contract Tests and Production Testing 19
Observability in Microservices Architecture **20**
Introduction to the Three Pillars of Observability in Microservices 20
Distributed Logging 21
Metrics 22
Distributed Tracing 23
Deployment of Microservices and Event-Driven Architecture in Production **24**
Microservices Deployment - Cloud Virtual Machine, Dedicated Hosts and Instances 24
Serverless Deployment for Microservices using Function as a Service 25
Containers for Microservices in Dev, QA and Production 26
Container Orchestration and Kubernetes for Microservices Architecture 28

Introduction

Introduction to Microservices and Event-Driven Architecture

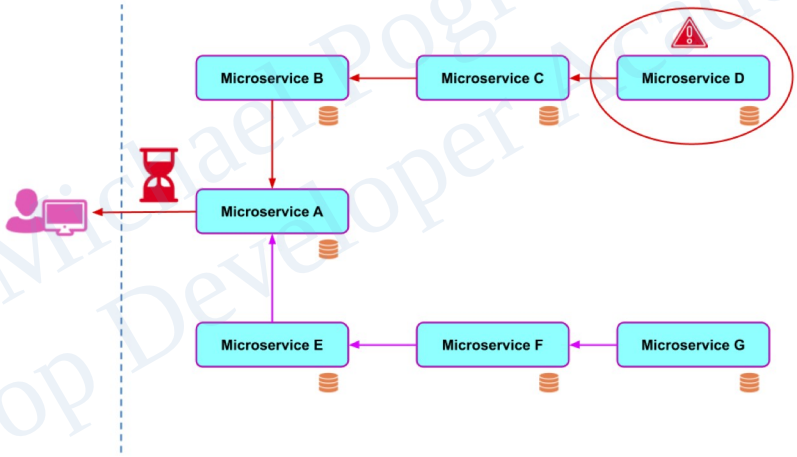
- **Monolithic Architecture:**
  - Benefits:
    - Easy to design
    - Easy to implement
  - Issues:
    - Low Organizational Scalability
    - Low System Scalability



Notes:

Microservices Architecture - Benefits and Challenges

- **Microservices Architecture:**
  - Benefits:
    - High Organizational Scalability
    - High System Scalability
  - Challenges:
    - Highly distributed system
    - Harder to test together
    - Difficult to troubleshoot and debug
    - Potential to lower Organizational Scalability (if done incorrectly)

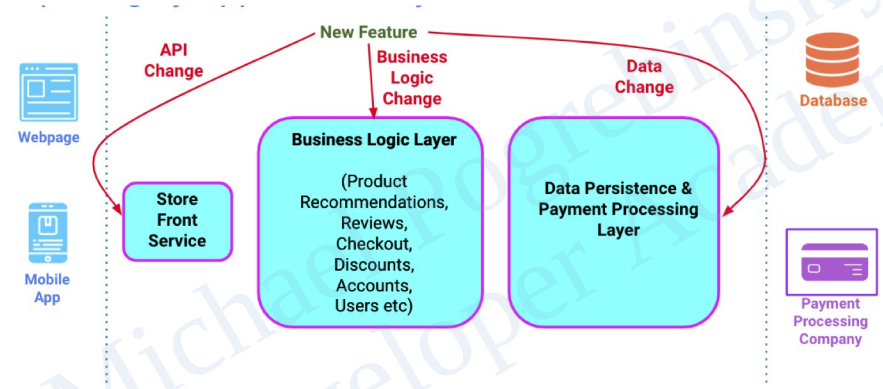


Notes:

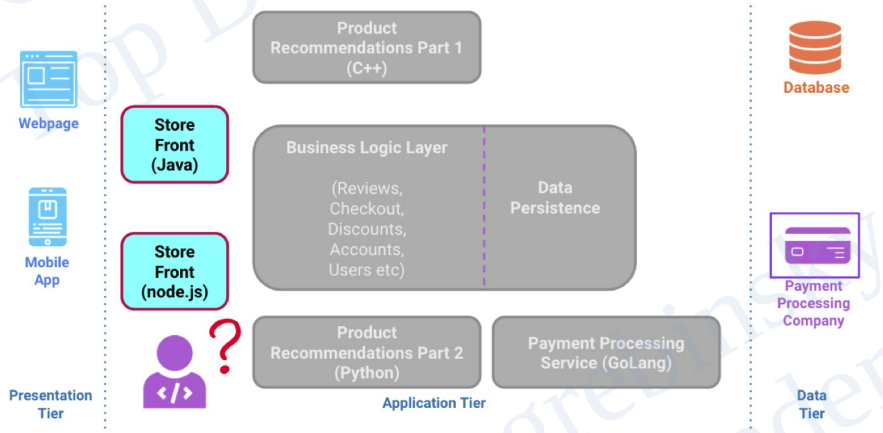
Migration to Microservices Architecture

Microservices Boundaries - Core Principles

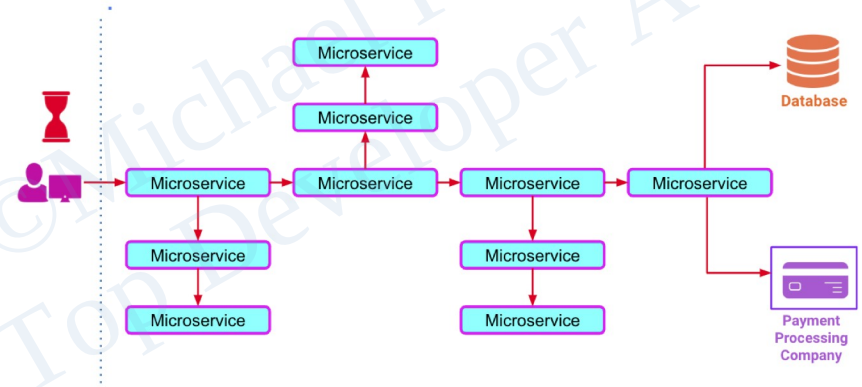
- **Claim**: *Just breaking a large codebase into an arbitrary set of microservices won't give us any benefit*
- **Boundaries for Microservices - Core Principles:**
  - *Cohesion*



  - *Single Responsibility Principle (SRP)*



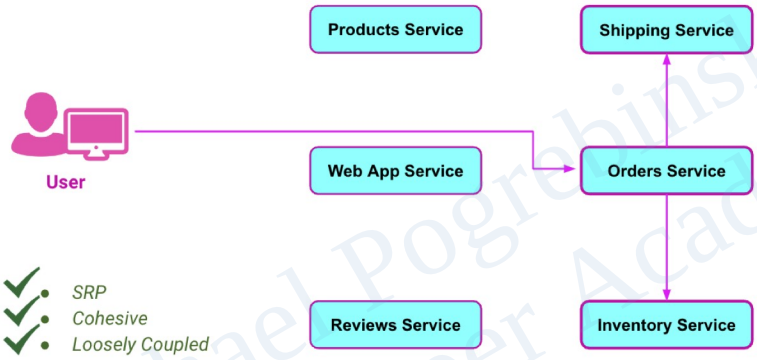
  - *Loose Coupling*



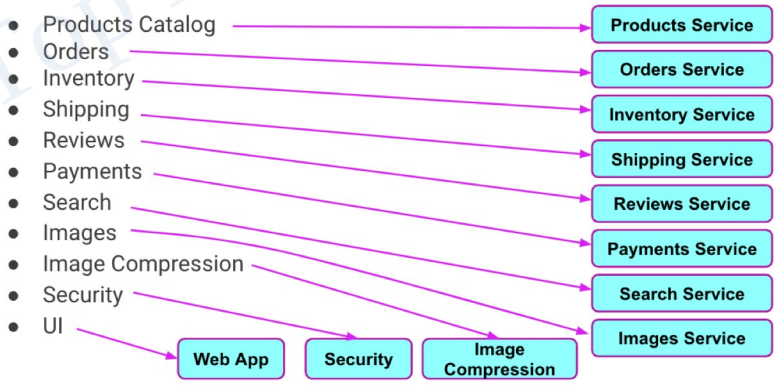
Notes:

Decomposition of a Monolithic Application to Microservices

- **Decomposition by:**
  - Business capability



  - Domain and Subdomain
    - Core
    - Supporting
    - Generic

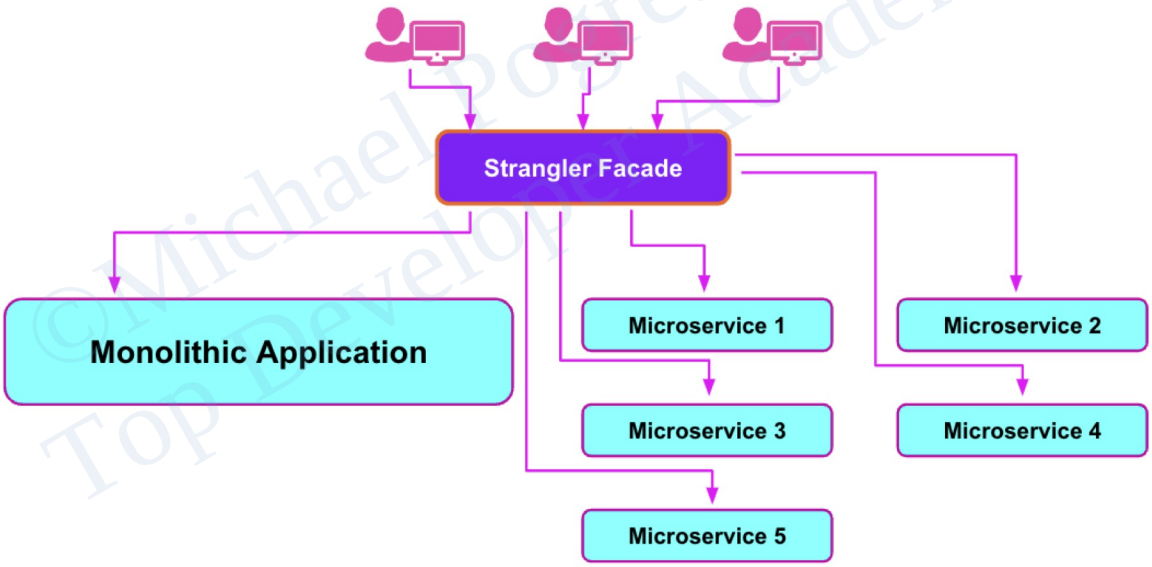


Notes:

Migration to Microservices - Steps, Tips and Patterns

- **Steps to Migrations:**
  a. Add/ensure code test coverage
  b. Define component API
  c. Isolate the component by removing interdependencies to the rest of the application

- **Method**: *Strangler Fig Pattern*

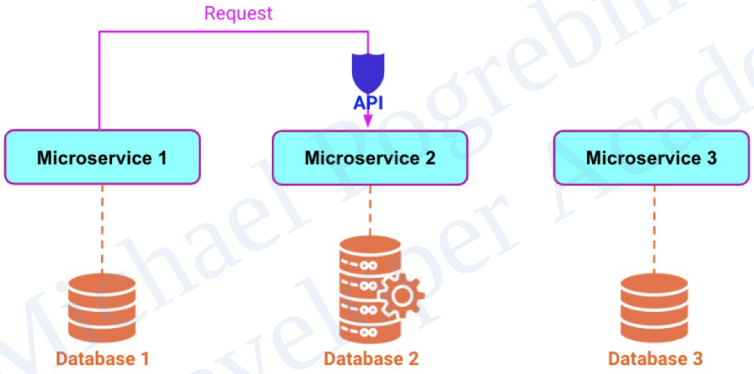


Notes:

Microservices - Principles and Best Practices

Databases in Microservices Architecture

- **Benefits:**
  - Each microservice team fully owns its data

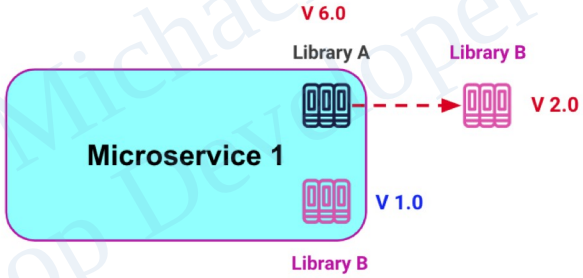


- **Downsides**:
  - Added latency
  - No “Join” operation
  - No support for transactions

Notes:

The DRY Principle In Microservices and Shared Libraries

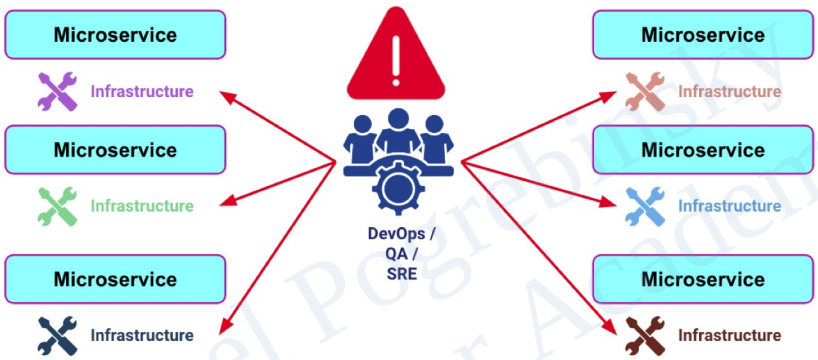
- **Challenges of shared libraries:**
  - Tight coupling
  - Every change requires:
    - Rebuild
    - Retest
    - Redeploy
  - Bug/vulnerability in a shared library impacts all microservices
  - “Dependency Hell”



- **Solutions:**
  - New Microservice
  - Code generation
  - Code duplication
  - Sidecar Pattern
- **Data duplication -** Only one owner/source of truth

Notes:

Structured Autonomy for Development Teams



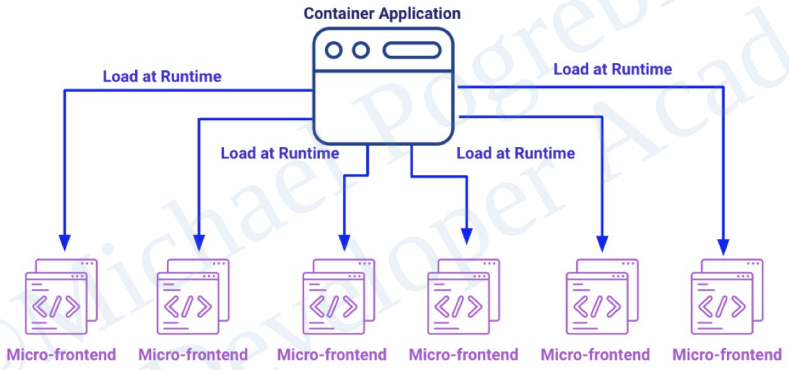
The 3 tiers of team autonomy:

1. Full restrictive
   - Infrastructure
   - API
   - Security
2. Freedom with boundaries
   - Programming languages
   - Database technologies
3. Complete autonomy
   - Release process
   - Release schedule and frequency
   - Custom scripts for local development and testing
   - Documentation
   - The onboarding process for new developers

Notes:

Micro-frontends Architecture Pattern

- **Roles of the Container Application:**
  - Render common elements
  - Take care of common functionality
  - Tell each micro-frontend where/when to be rendered



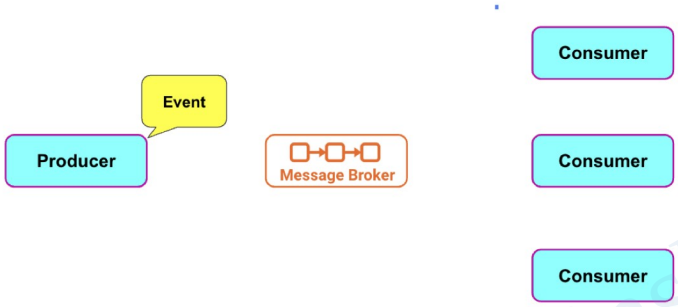
- **Benefits of Micro-frontends:**
  - Replacement of the complex monolithic codebase with small and manageable micro-frontends
  - Full-stack ownership of each micro-frontend
  - Easier/Faster to test in isolation
  - Separate CI/CD pipeline
  - Separate release schedule

Notes:

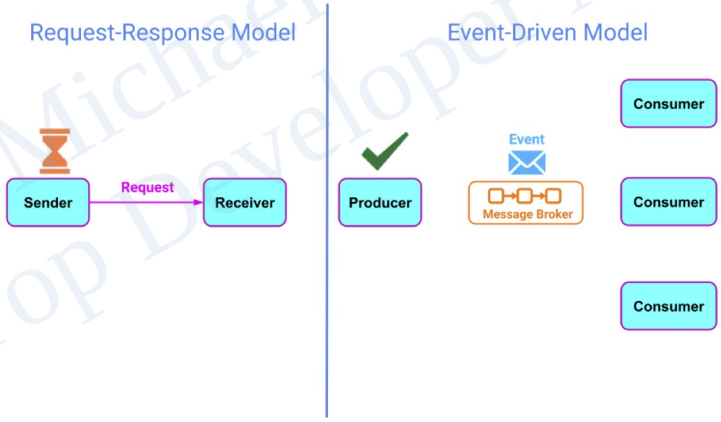
Event-Driven Architecture

Introduction to Event-Driven Architecture

- **Fundamentals of Event-Driven Architecture:**
  - Event - Fact, Action, State Change



  - Request-Response Model vs. Event-Driven Model:
    - Synchronous vs. Asynchronous
    - Inversion of Control
    - Loose Coupling



Notes:

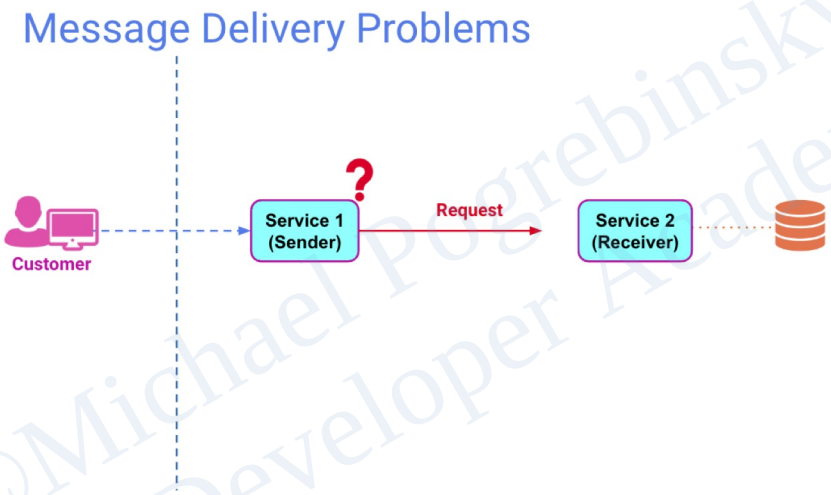
Use Cases and Patterns of Event-Driven Architecture

- **Event-Driven Architecture use cases:**
  - *Fire and Forget*
  - *Reliable delivery*
  - *Infinite stream of events*
  - *Anomaly detection/pattern recognition*
  - *Broadcasting*
  - *Buffering*
- **Request-Response Model use cases:**
  - Immediate response with data is needed
  - Simple interaction

- **Event-Delivery Patterns:**
  - Event Streaming
  - Publisher / Subscribe

Notes:

Message Delivery Semantics in Event-Driven Architecture



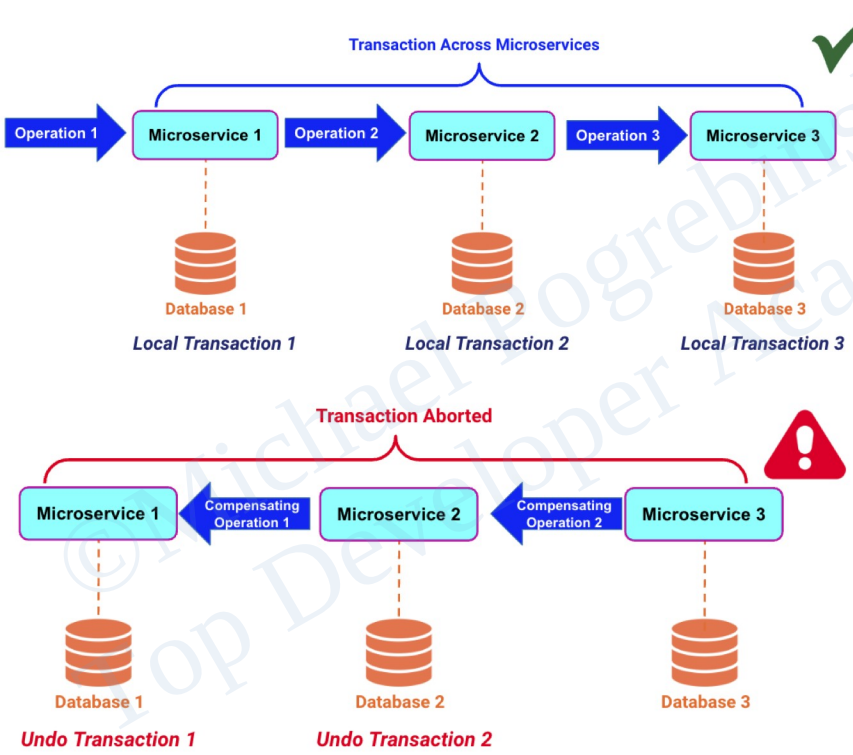
- **Message Deliery Semantics in Event-Driven Architecture:**
  - At-Most-Once
    - Data loss is “OK”
    - Least overhead /lowest latency
  - At-Least-Once
    - Data loss is unacceptable
    - Data duplication is OK
    - Increased latency
  - Exactly-Once
    - Most difficult to achieve
    - Highest overhead / latency

Notes:

Event-Driven Microservices - Design Patterns

Saga Pattern

- **Problem**: Loss of ACID Transactions in Microservices
- **Solution**: Saga Pattern

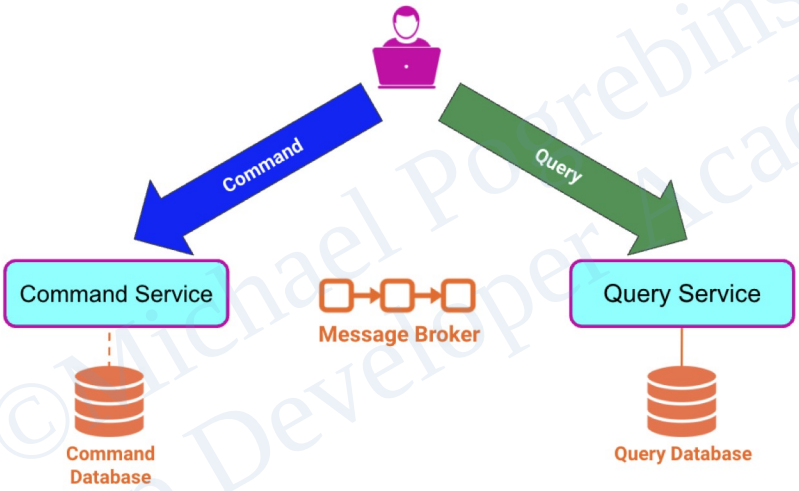


- **Two implementations:**
  - Workflow Orchestration
  - Event-Driven Model

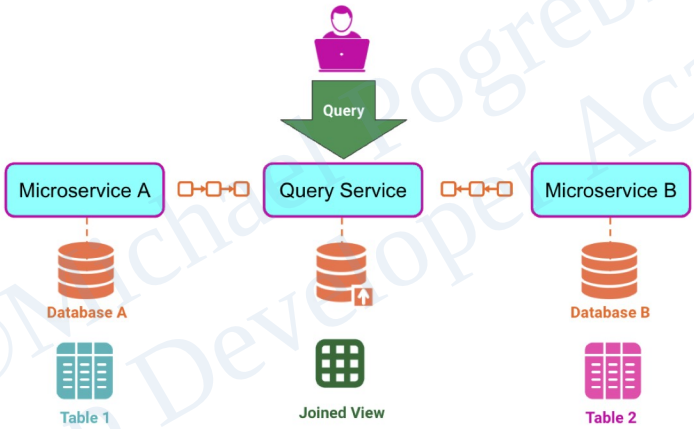
Notes:

CQRS Pattern

- **CQRS** - Command and Query Responsibility Segregation

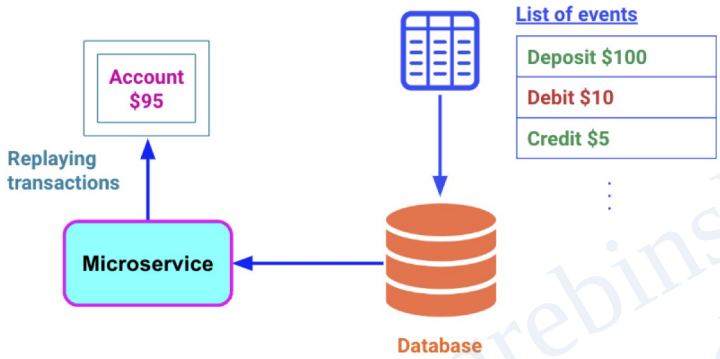


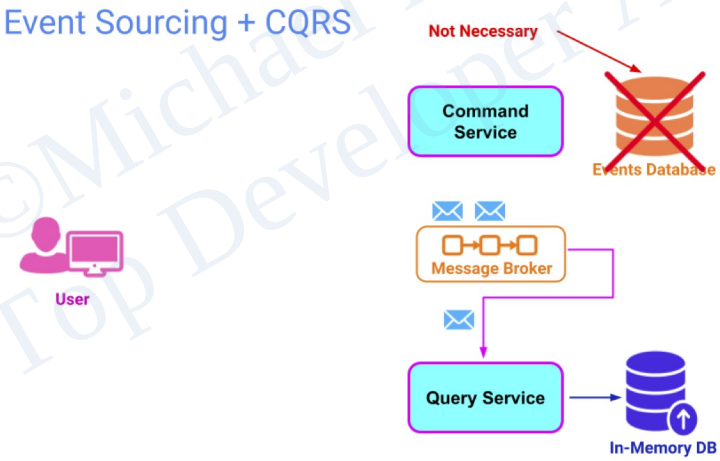
- **Benefits**:
  - Single Responsibility Principle (SRP)
  - Higher Performance
  - Higher Scalability
  - “Join” Operation in Microservices



Notes:

Event Sourcing Pattern





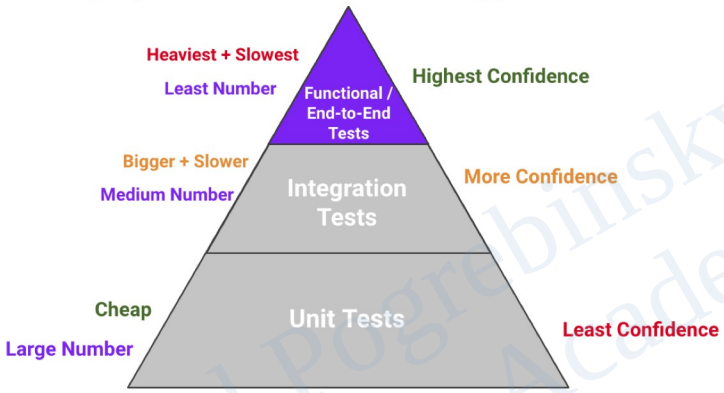
- **Benefits:**
  - Visualization
  - Auditing
  - Corrections
  - High Write Performance

Notes:

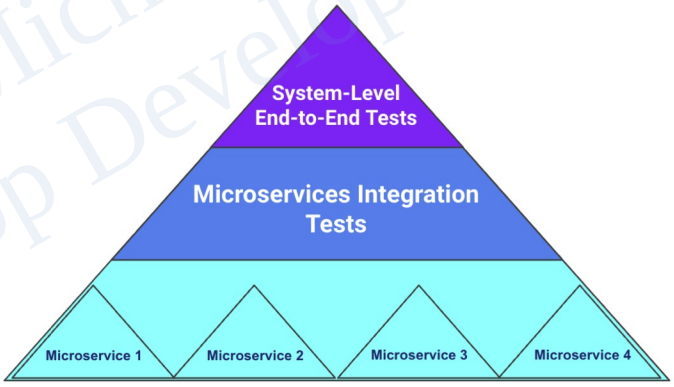
Testing Microservices and Event-Driven Architecture

Testing Pyramid for Microservices - Introduction and Challenges

- **Testing Pyramid for a Monolithic Application:**



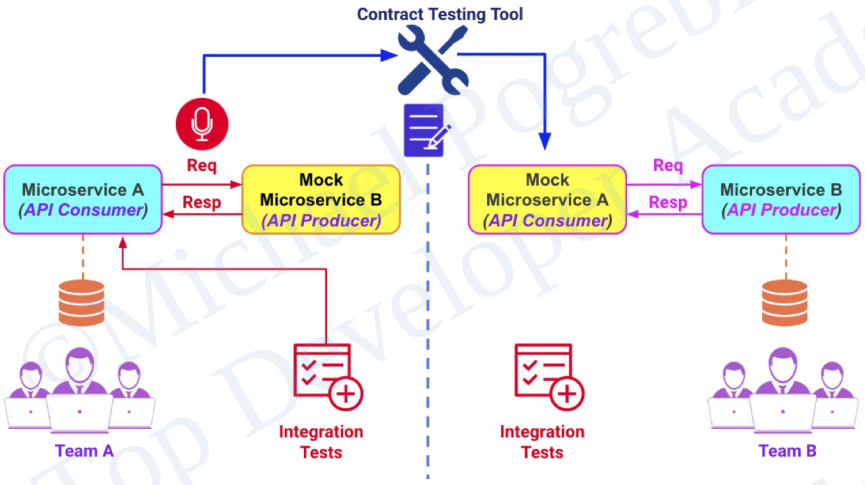
- **Testing Pyramid for a Microservices Architecture:**



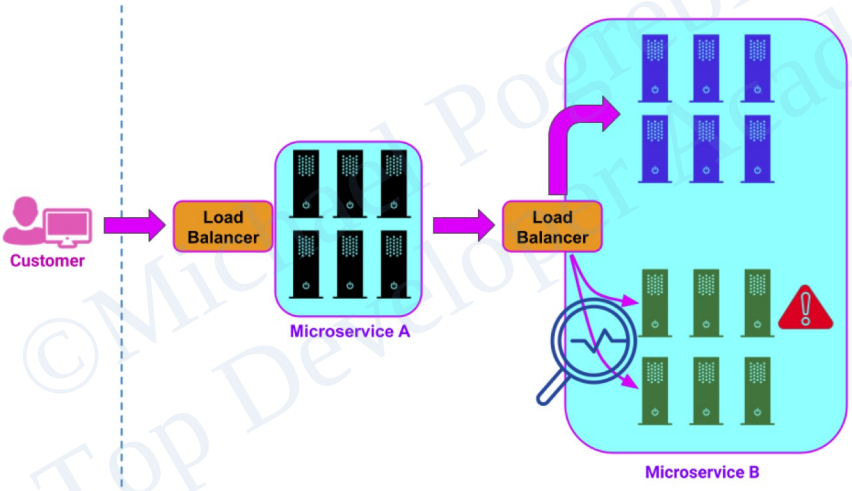
Notes:

Contract Tests and Production Testing

- **Alternative to Integration Tests:**
  - *Contract Tests*



- **Alternative to End-to-End Tests:**
  - *Blue Green Deployment + Canary Testing*

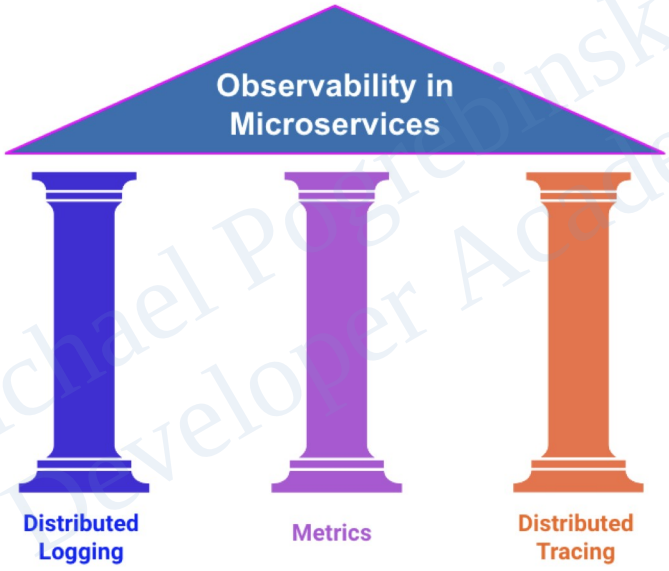


Notes:

Observability in Microservices Architecture

Introduction to the Three Pillars of Observability in Microservices

- **Observability:**
  - Allows us to follow individual, requests, transactions and events
  - Discover and isolate performance bottlenecks
  - Point us to the source of the problem



Notes:

Distributed Logging

- **Best Practices:**
  - Centralized System
  - Predefined structure/schema

logfmt

```
time=2023-11-01T23:07:42+00:00 app=review level=INFO duration=432 message="example of a logfmt"
```

JSON

```
{
   "timestamp": "2023-11-01T23:07:42+00:00",
   "level": "info",
   "message": "Server starting",
   "server_id": "baser3e",
   "start_time": "2023-11-01T23:07:42"
}
```

XML

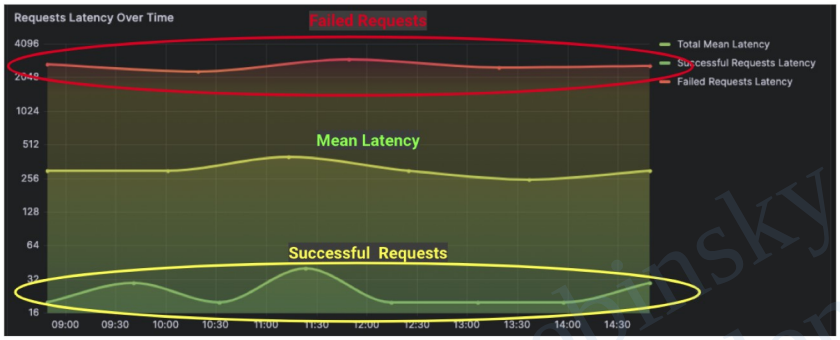
```
<Event timeMillis="43234" thread="main" level="DEBUG"
threadPriority="3">
        <Instant epochSecond="234323" nanoOfSecond="54322000"/>
        <Message>Sample debug message</Message>
</Event>
```

  - Log Level / Log Severity
    - *TRACE*
    - *DEBUG*
    - *INFO*
    - *WARN*
    - *ERROR*
    - *FATAL*
    - Correlation Id
  - Adding Contextual information

Notes:

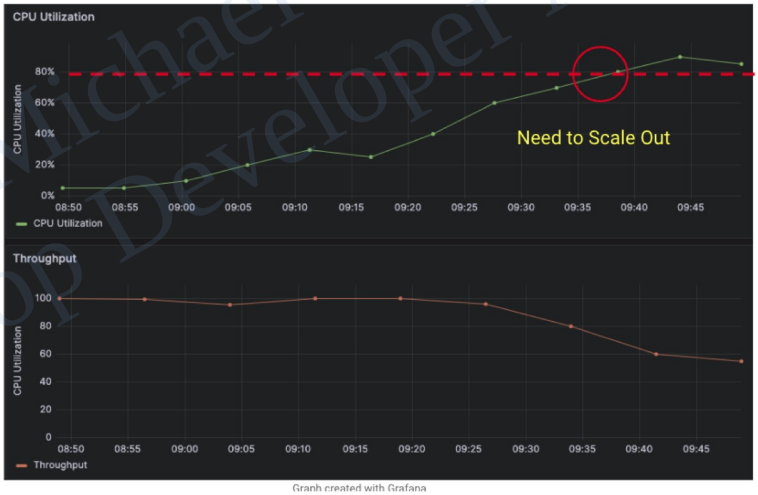
Metrics

- **Metrics Definition: "***Measurable or Countable signals of software that help us monitor the system's health and performance"*
- **The Five (Golden) Types of Signals**
  - *Traffic*
  - *Errors*
  - *Latency*



Graph created with Grafana

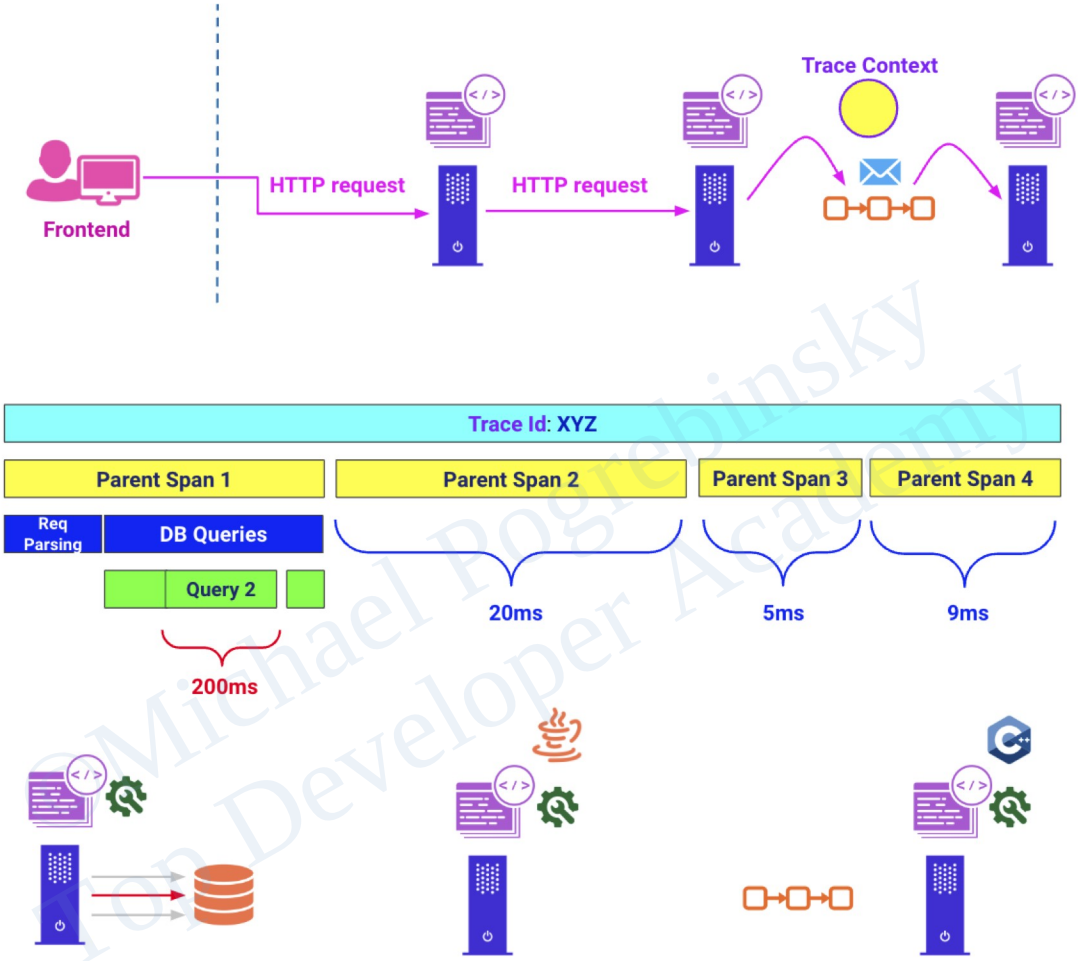
  - *Saturation*
  - *Utilization*



Graph created with Grafana

Notes:

Distributed Tracing

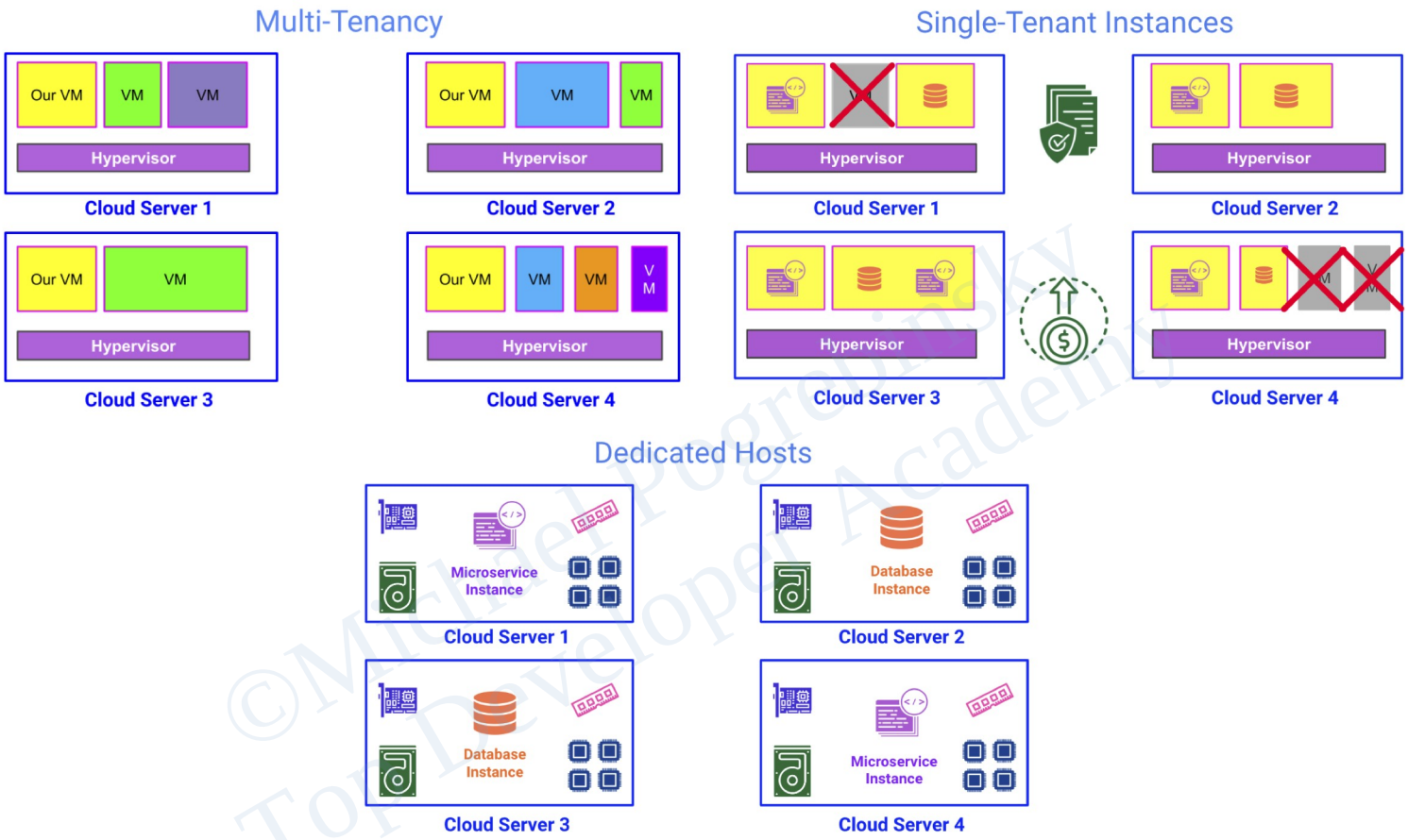


- **Challenges:**
  - Manual instrumentation of code
  - Cost
  - Big Traces / too much data

Notes:

Deployment of Microservices and Event-Driven Architecture in Production

Microservices Deployment - Cloud Virtual Machine, Dedicated Hosts and Instances



Notes:

Serverless Deployment for Microservices using Function as a Service

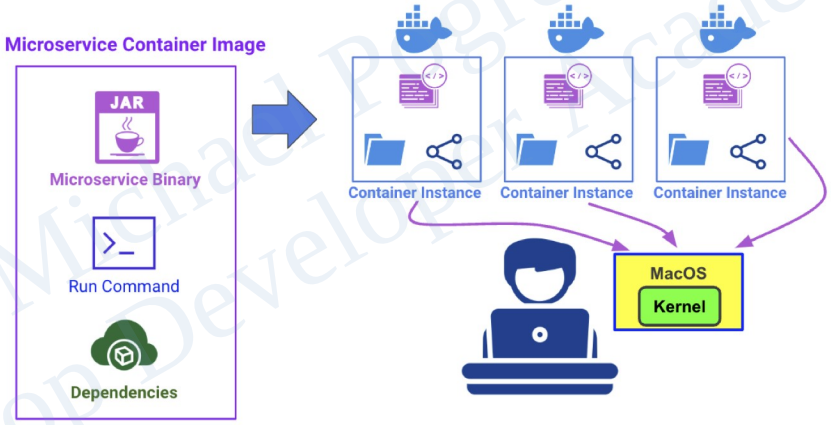
- **FaaS - Function as a Service:**
  - Event-Driven:
    - Software
    - Infrastructure
  - We provide:
    - Type of request/event to handle
    - Logic to execute
  - Pricing based on:
    - Number of requests/month
    - Execution time
    - Memory

Deployment Type	Cost	Security	Performance
Multi-Tenant Cloud VM	$		
Single-Tenant Cloud VM	$ $		
Dedicated Host	$ $ $		
Function as a Service	¢ / $ $ $		

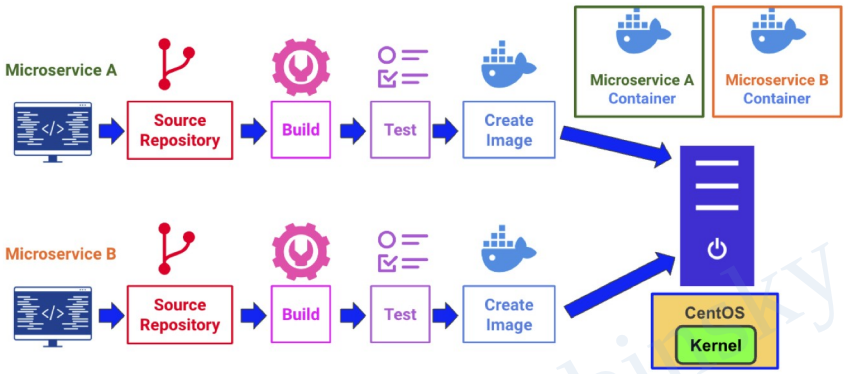
Notes:

Containers for Microservices in Dev, QA and Production

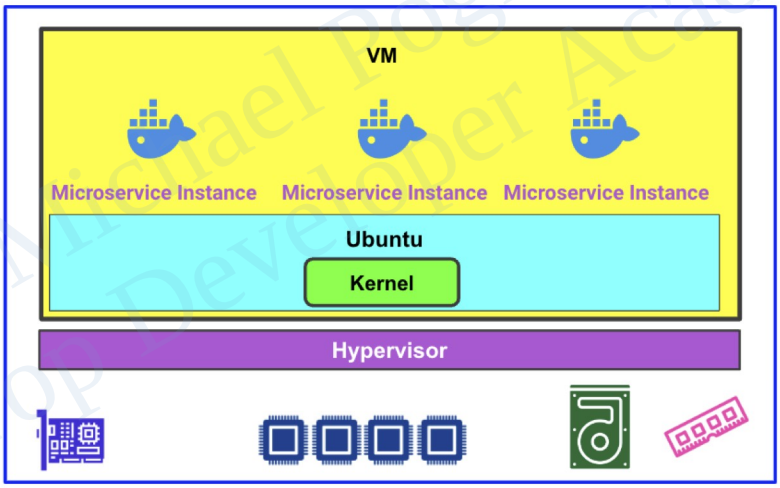
- **Deploying Microservices as Containers in:**
  - Development



  - QA / Continuous Integration



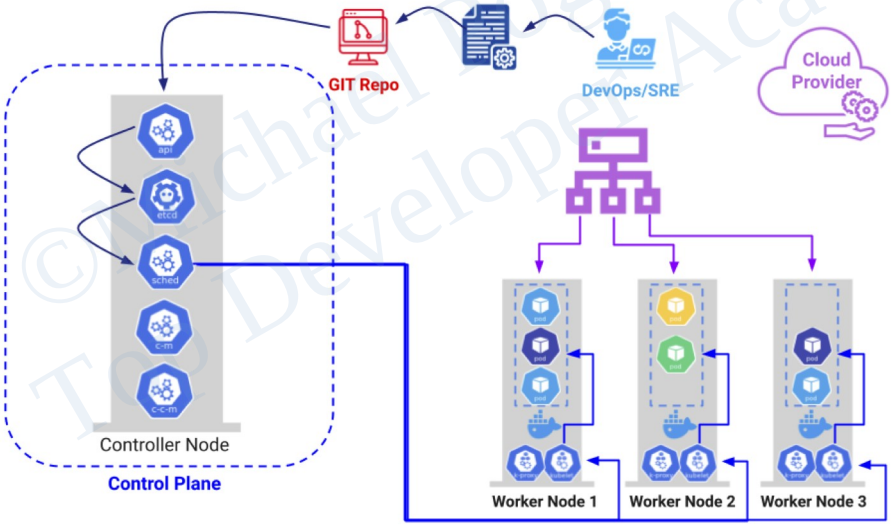
  - Production



Notes:

Container Orchestration and Kubernetes for Microservices Architecture

- **Container Orchestration - Responsibilities:**
  - Deployment automation
  - Resource allocation
  - Health monitoring
  - Self-healing
  - Bin-packing
  - Load balancing
  - Scaling services
  - Container discovery
  - Network connectivity
- **Kubernetes Architecture:**



Notes: