WHAT'S NEXT IN CLOUD APP DEVELOPMENT

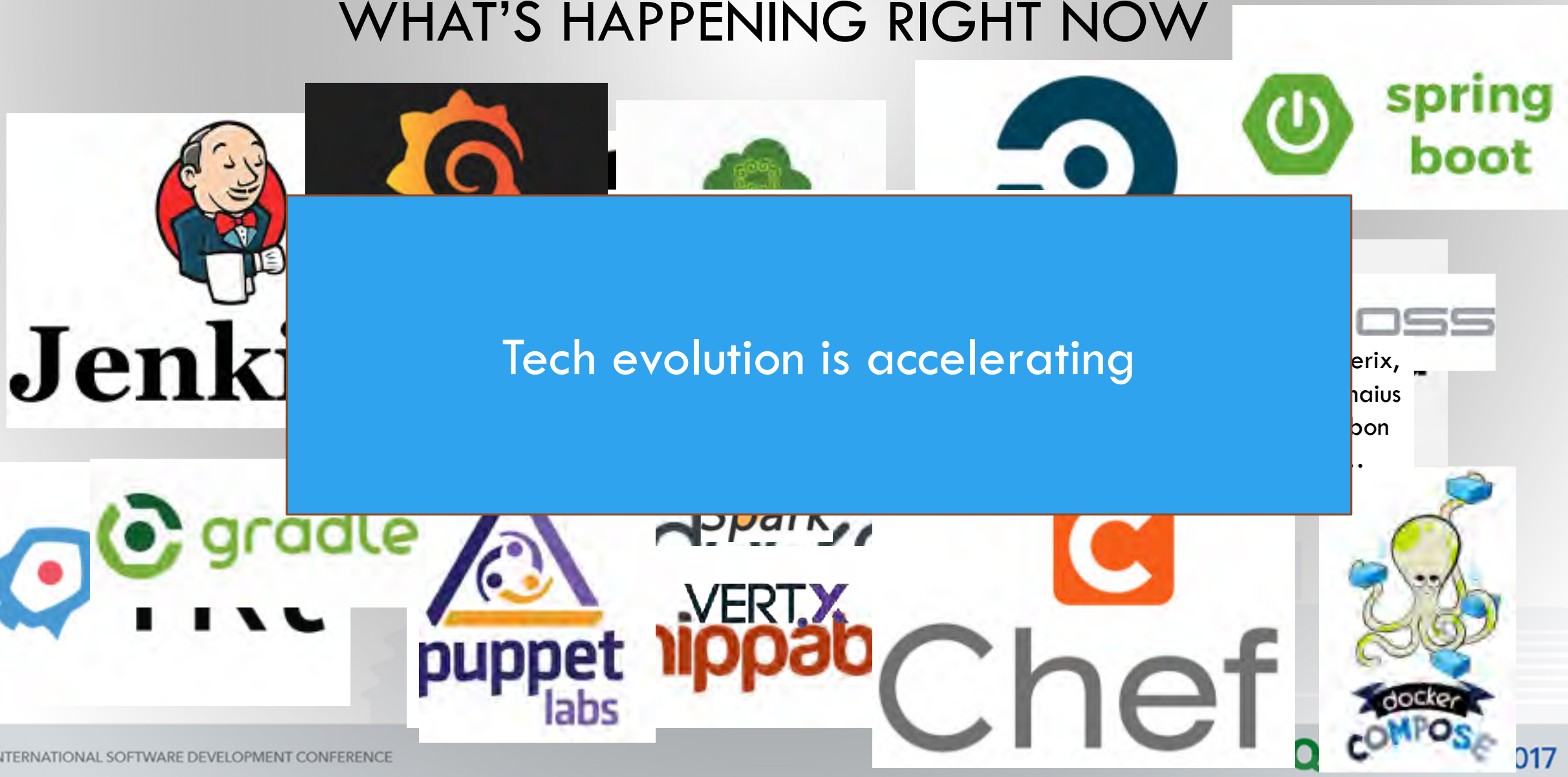
BORIS SCHOLL

VP ENGINEERING- ORACLE CLOUD

WHO AM I?

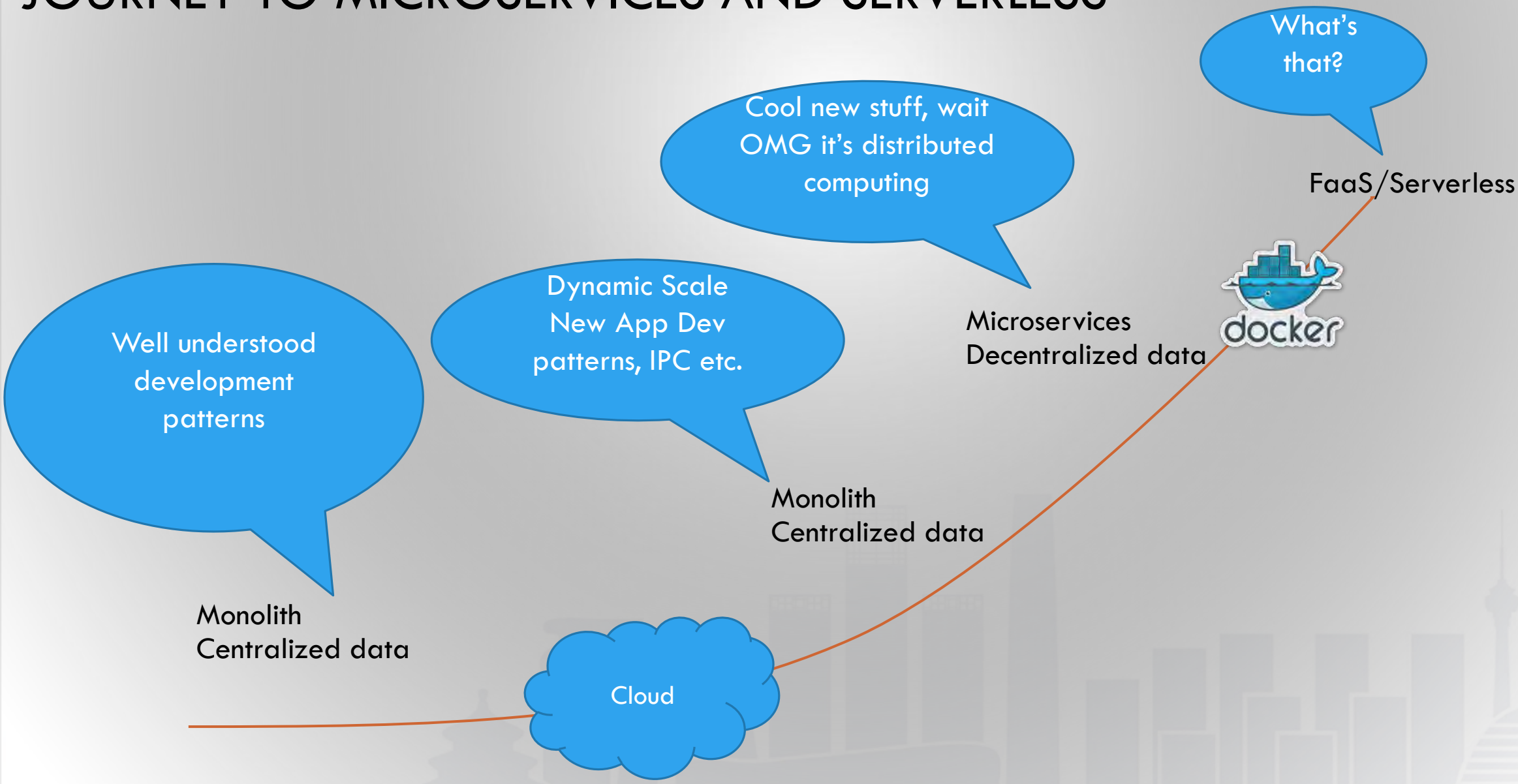
- 18+ YEARS EXPERIENCE IN SOFTWARE DEVELOPMENT
- WORKING ON CLOUD SERVICES FOR ABOUT 8 YEARS
- FOCUSING ON PATTERNS FOR LARGE SCALE CLOUD SOLUTIONS AND WORKING DISTRIBUTED SYSTEMS
- LEADING ENGINEERING FOR NEXT GEN MICROSERVICES PLATFORM @ORACLE
- AUTHOR OF BOOKS AND ARTICLES ON CLOUD DEVELOPMENT AND MICROSERVICES
- LIVE IN SEATTLE WITH MY FAMILY AND LOVE WAKEBOARDING AND SKIING

WHAT'S HAPPENING RIGHT NOW

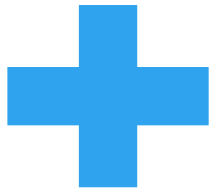


Tech evolution is accelerating

JOURNEY TO MICROSERVICES AND SERVERLESS



FAST EMERGING CLOUD DEPLOYMENT ARCHITECTURE FOR MICROSERVICES CIRCA 2017



Swarm

Kubernetes

Mesos

Cloud Server Today

12 FACTOR - MANIFESTO FOR CLOUD APPS

THE TWELVE-FACTOR APP

INTRODUCTION

In the modern era, software is commonly delivered as a service: called *web apps*, or *software-as-a-service*. The twelve-factor app is a methodology for building software-as-a-service apps that:

- Use **declarative** formats for setup automation, to minimize time and cost for new developers joining the project;
- Have a **clean contract** with the underlying operating system, offering **maximum portability** between execution environments;
- Are suitable for **deployment** on modern **cloud platforms**, obviating the need for servers and systems administration;
- **Minimize divergence** between development and production, enabling **continuous deployment** for maximum agility;
- And can **scale up** without significant changes to tooling, architecture, or development practices.

The twelve-factor methodology can be applied to apps written in any programming language, and which use any combination of backing services (database, queue, memory cache, etc).

BACKGROUND

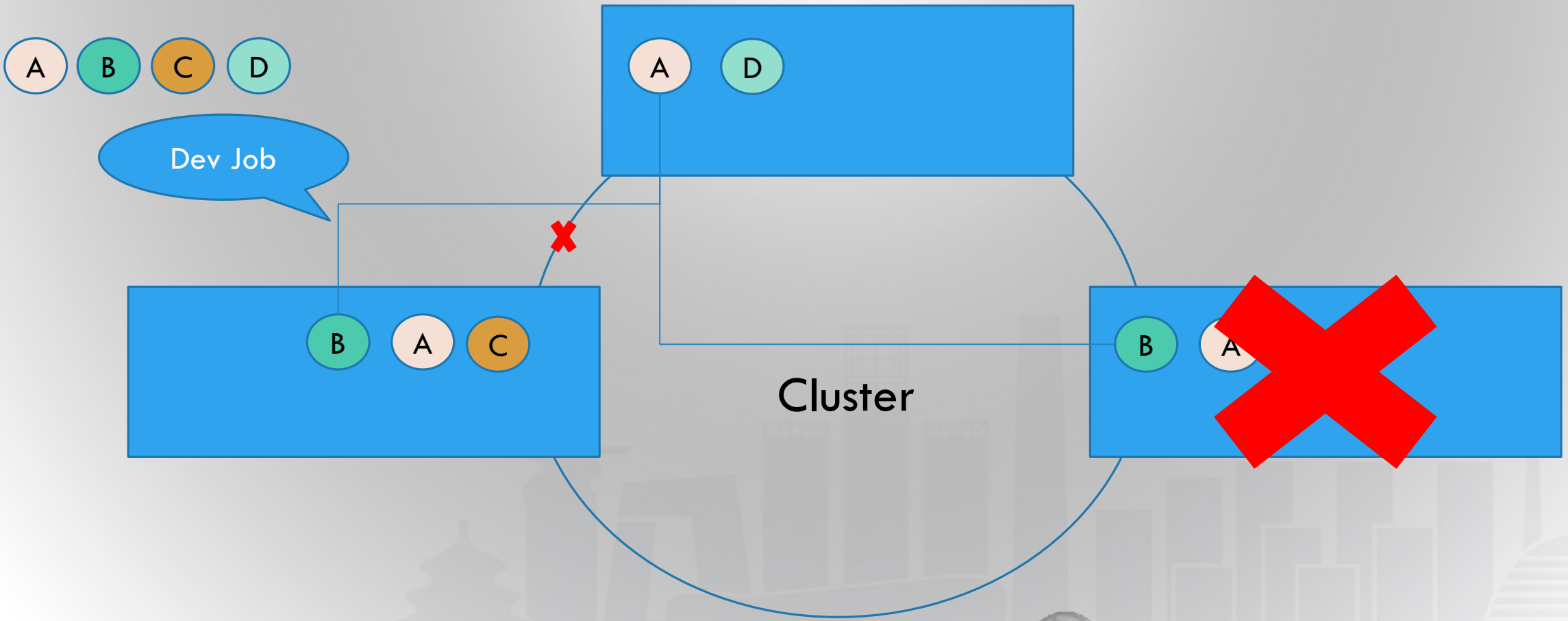
The contributors to this document have been directly involved in the development and deployment of hundreds of apps, and indirectly witnessed the development, operation, and scaling of hundreds of thousands of apps via our work on the Heroku platform.

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

MICROSERVICES, CONTAINERS AND ORCHESTRATORS HOW DOES IT HELP DEVELOPERS?

- AUTONOMOUS SERVICES
- SMALL AUTONOMOUS TEAMS
- FAST DELIVERY
- USE THE BEST TECHNOLOGY FOR THE PROB
- IMMUTABLE ENVIRONMENT
- FAST BOOT UP TIME
- ETC……

MICROSERVICES, CONTAINERS AND ORCHESTRATORS – BENEFITS FOR DEVELOPERS



COMMON THINGS DEVELOPERS NEED TO CONSIDER FOR MICROSERVICES

Autoscaling

Resiliency

Async

Idempotency

Event Sourcing

Configuration

Serialization

Stateless Compute

Gateway

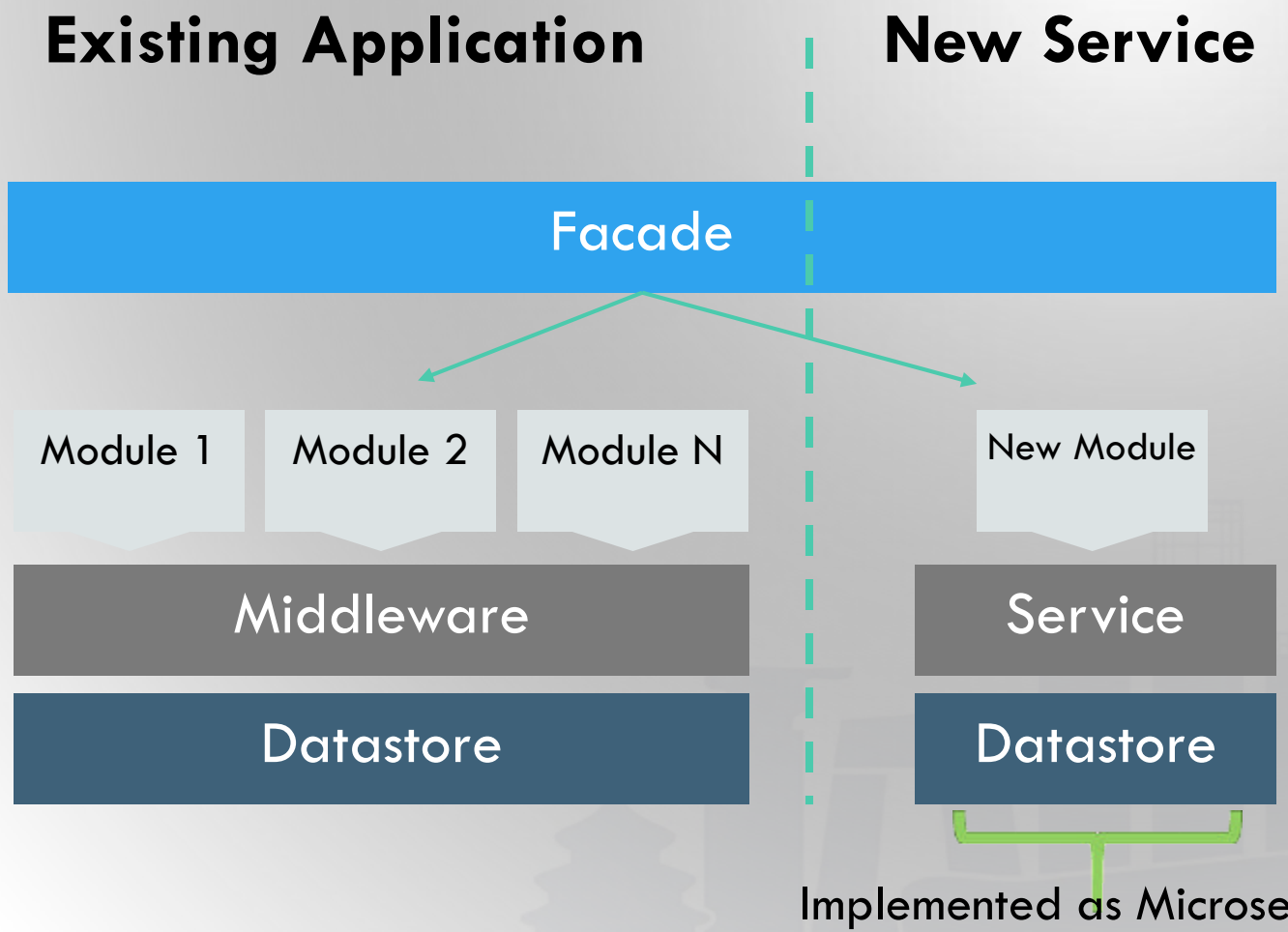
CQRS

Transaction log tailing

IPC Patterns

Distributed Tracing

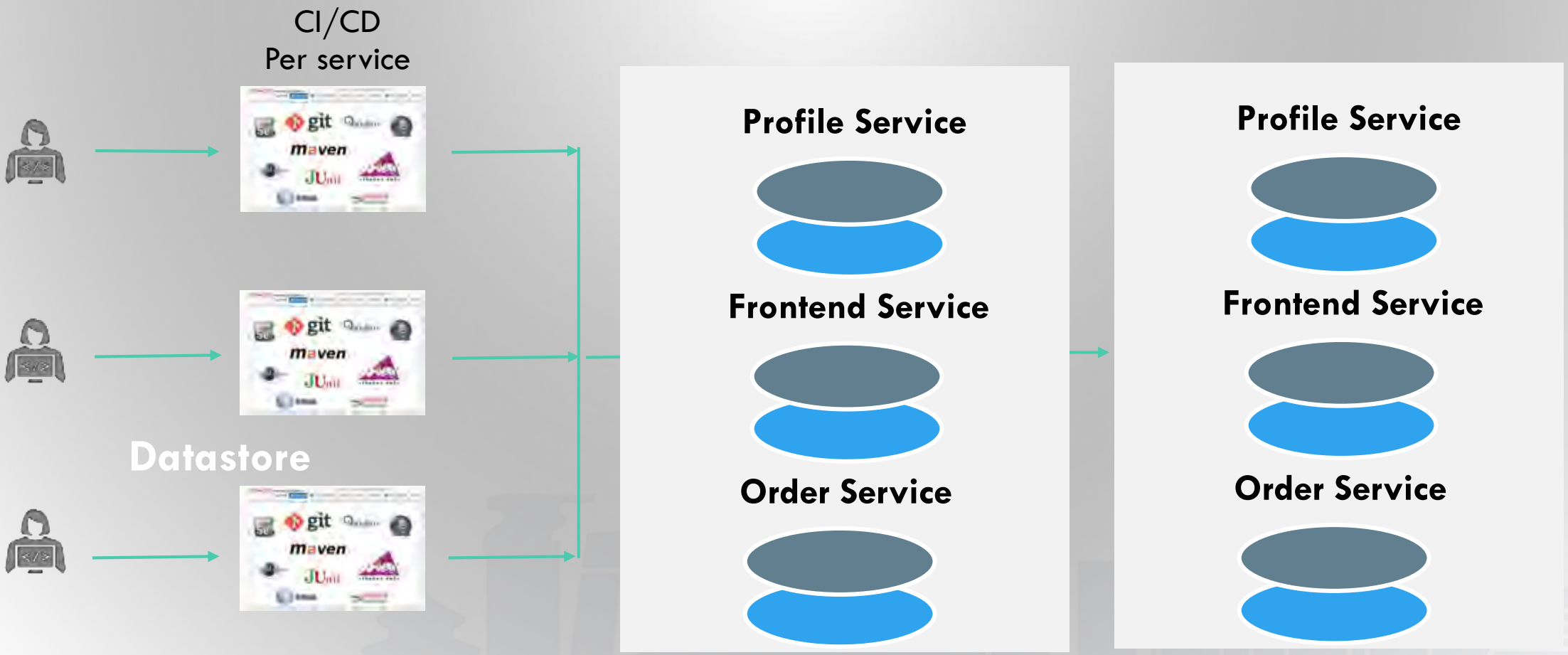
MONOLITH TO MICROSERVICES



- New services or existing components are implemented as microservices
- The façade routes user requests to the correct application
- Over time more and more features are moved to the new architecture

CI/CD MICROSERVICES

EACH SERVICE HAS ITS OWN CI/CD PIPELINE



WHAT IF…

- ….YOU COULD BREAK DOWN THE FUNCTIONALITY EVEN FURTHER?
  - SAY THE MICROSERVICE IMPLEMENTS FUNCTIONALITY THAT REALLY ONLY NEED TO DO A JOB AND BE DONE
- ….YOU JUST WANT TO SUBMIT THE CODE FOR IT AND HAVE IT REACT TO AN EVENT?
- ….YOU COULD INVOKE A FUNCTION EVEN FASTER THAN A CONTAINER BOOT?

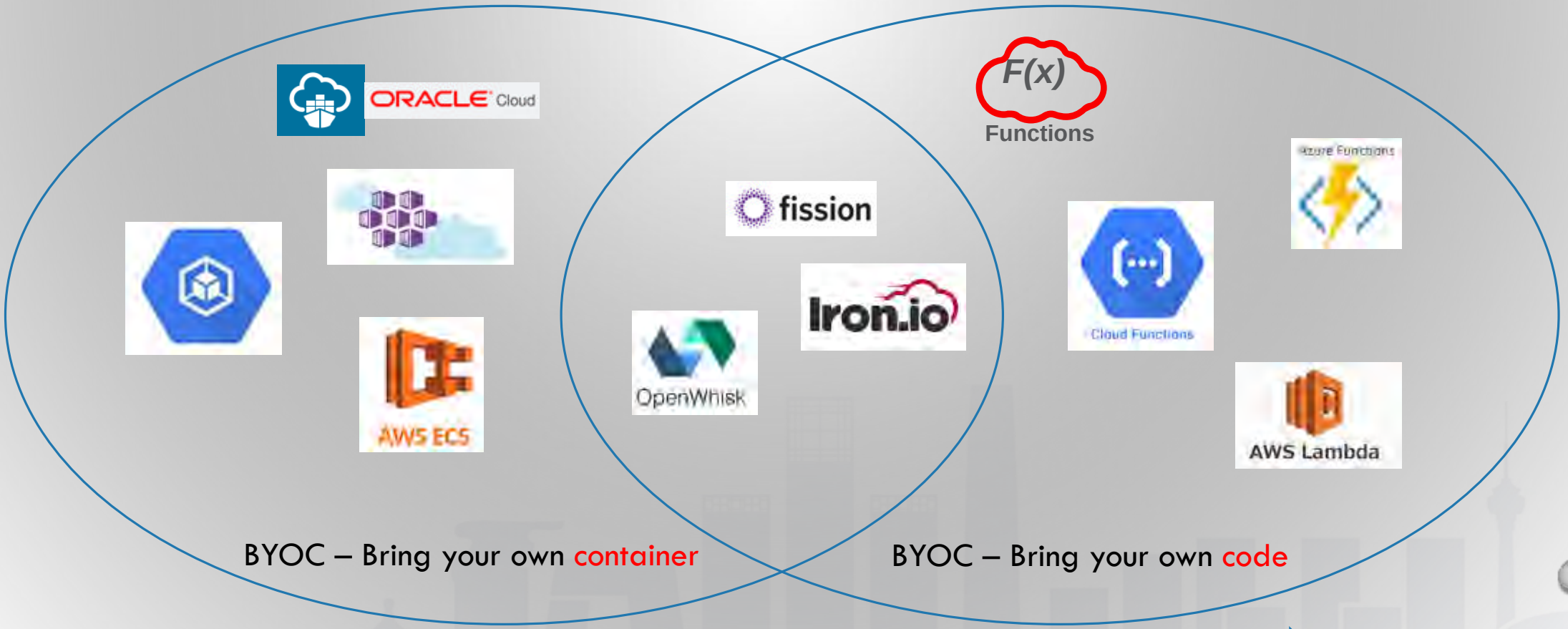
*Credit to AWS

SERVERLESS MANIFESTO*

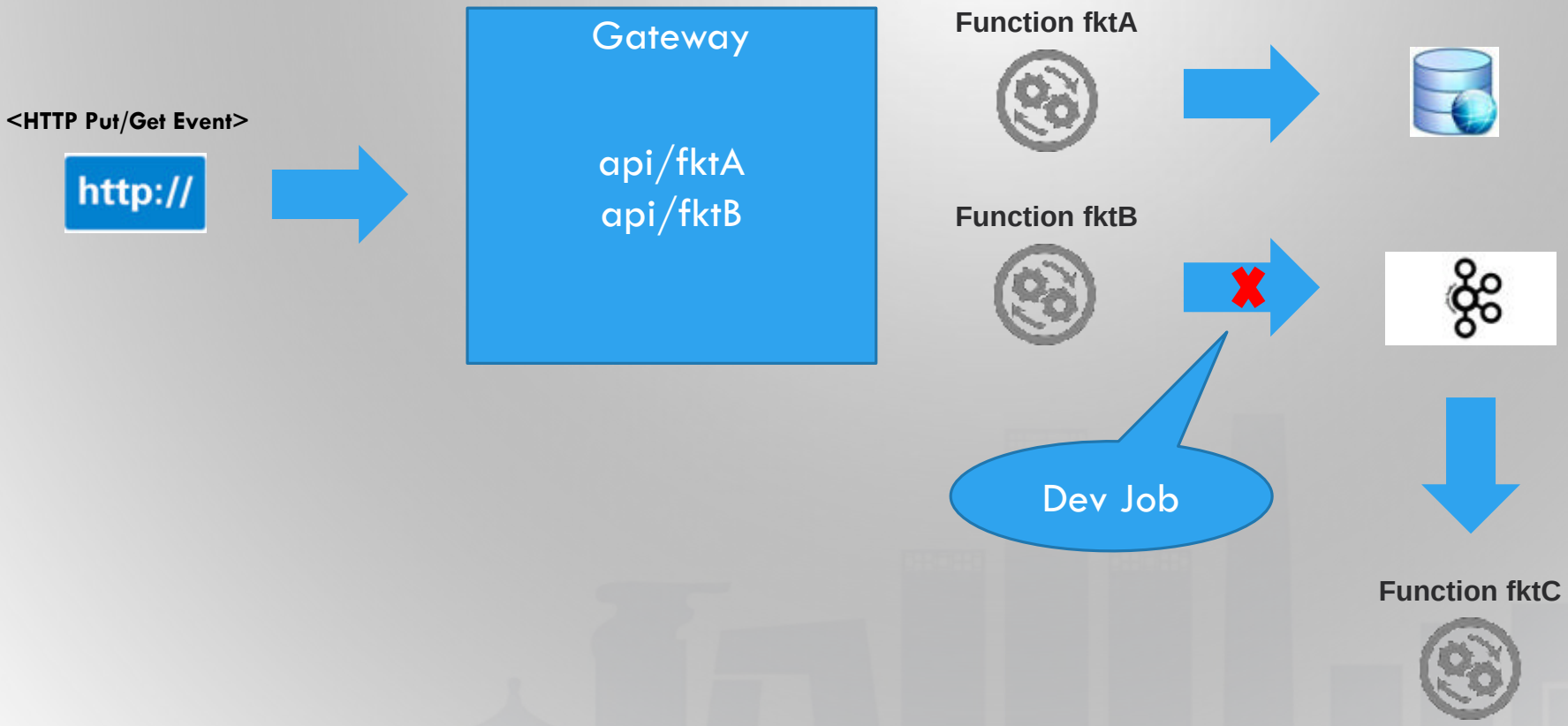
- FUNCTION ARE THE UNIT OF DEPLOYMENT AND SCALING.
- NO MACHINES, VMS, OR CONTAINERS VISIBLE IN THE PROGRAMMING MODEL.
- PERMANENT STORAGE LIVES ELSEWHERE.
- SCALES PER REQUEST; USERS CANNOT OVER- OR UNDER-PROVISION CAPACITY.
- NEVER PAY FOR IDLE (NO COLD SERVERS/CONTAINERS OR THEIR COSTS).
- IMPLICITLY FAULT-TOLERANT BECAUSE FUNCTIONS CAN RUN ANYWHERE.
- BYOC - BRING YOUR OWN CODE.
- METRICS AND LOGGING ARE A UNIVERSAL RIGHT.

*Credit to AWS

SERVERLESS LANSDCAPE



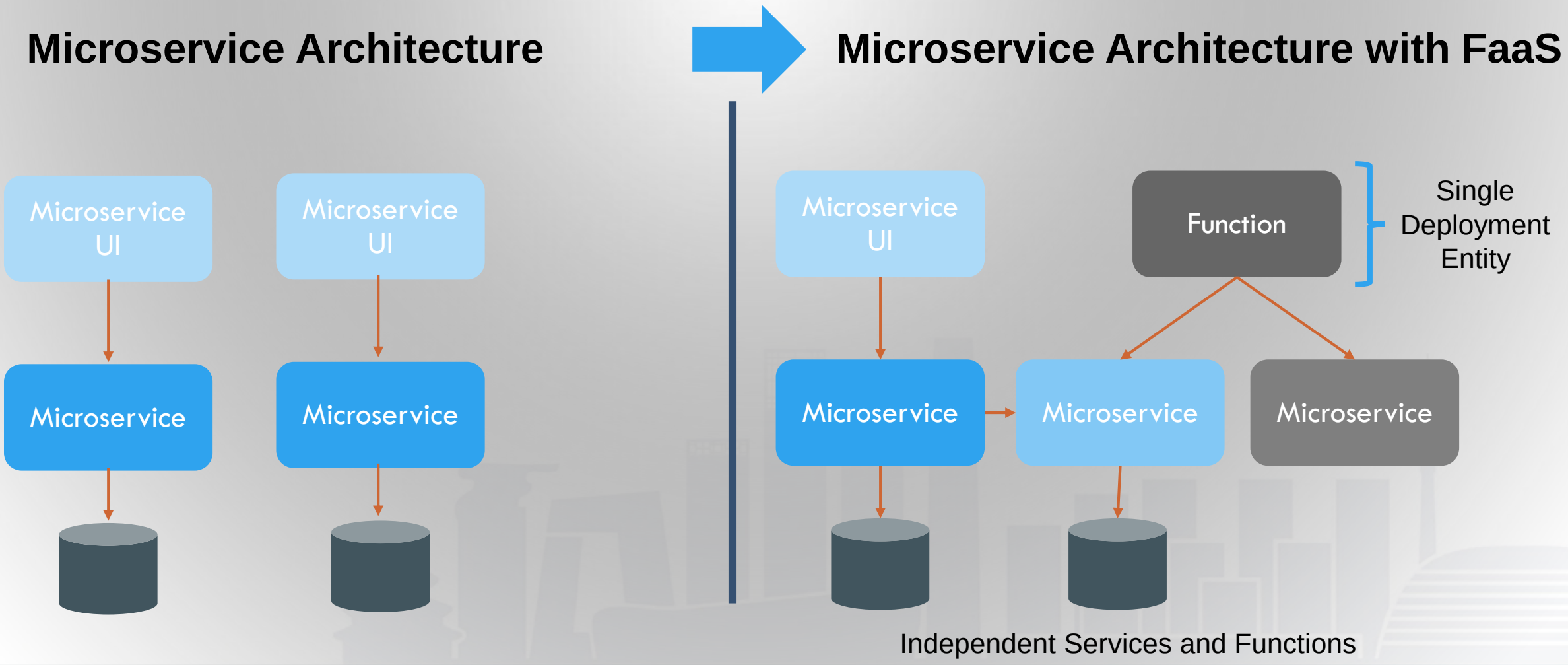
SERVERLESS APPLICATIONS



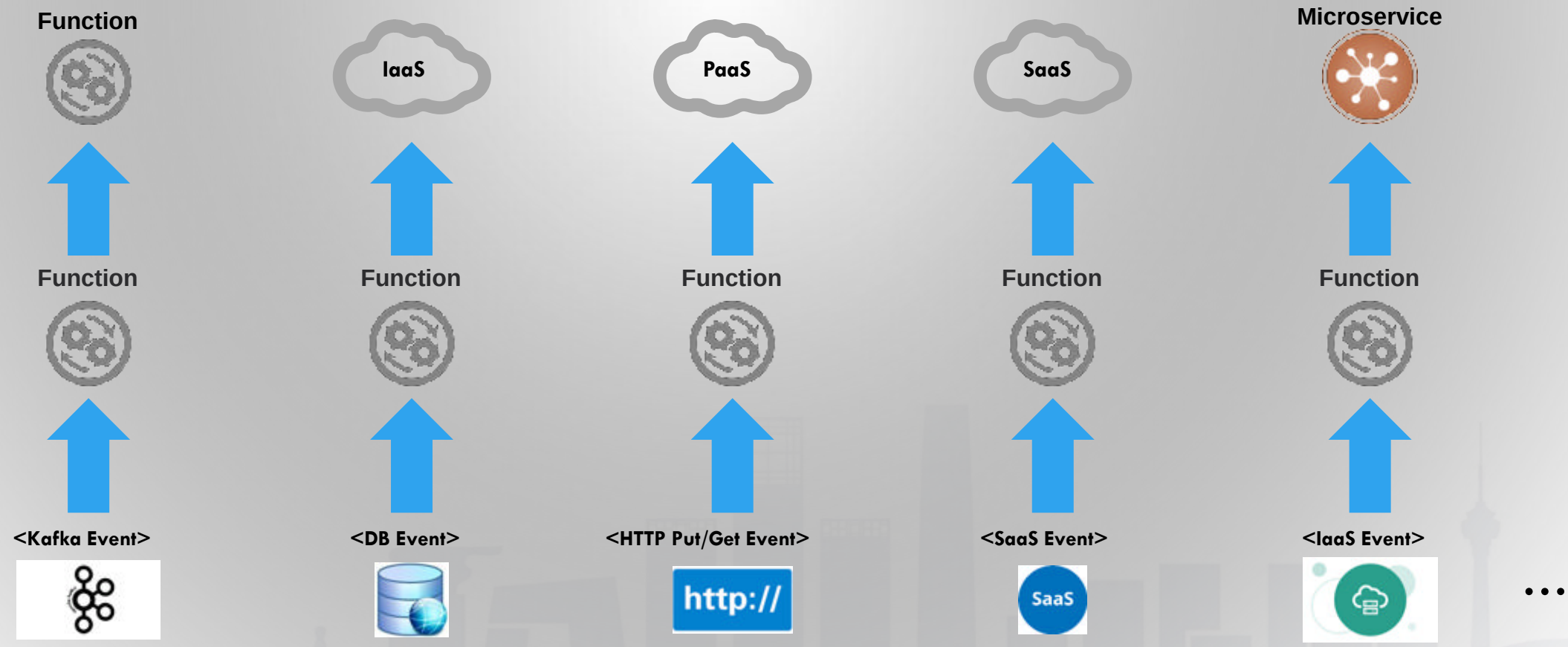
COMMON THINGS DEVELOPERS NEED TO CONSIDER FOR SERVERLESS

- THERE ARE STILL THINGS DEVELOPERS/OPS PERSONA NEED TO HANDLE
  - DEPLOYMENT, SECURITY, MONITORING ETC.
- STATELESS COMPUTE
  - NONE OF THE IN-PROCESS OR HOST STATE THAT YOU CREATE WILL BE AVAILABLE TO ANY SUBSEQUENT INVOCATION
- MANY MICROSERVICES PATTERNS ARE STILL RELEVANT
  - IDEMPOTENCY
  - DISTRIBUTED TRACING
  - MESSAGING PATTERNS

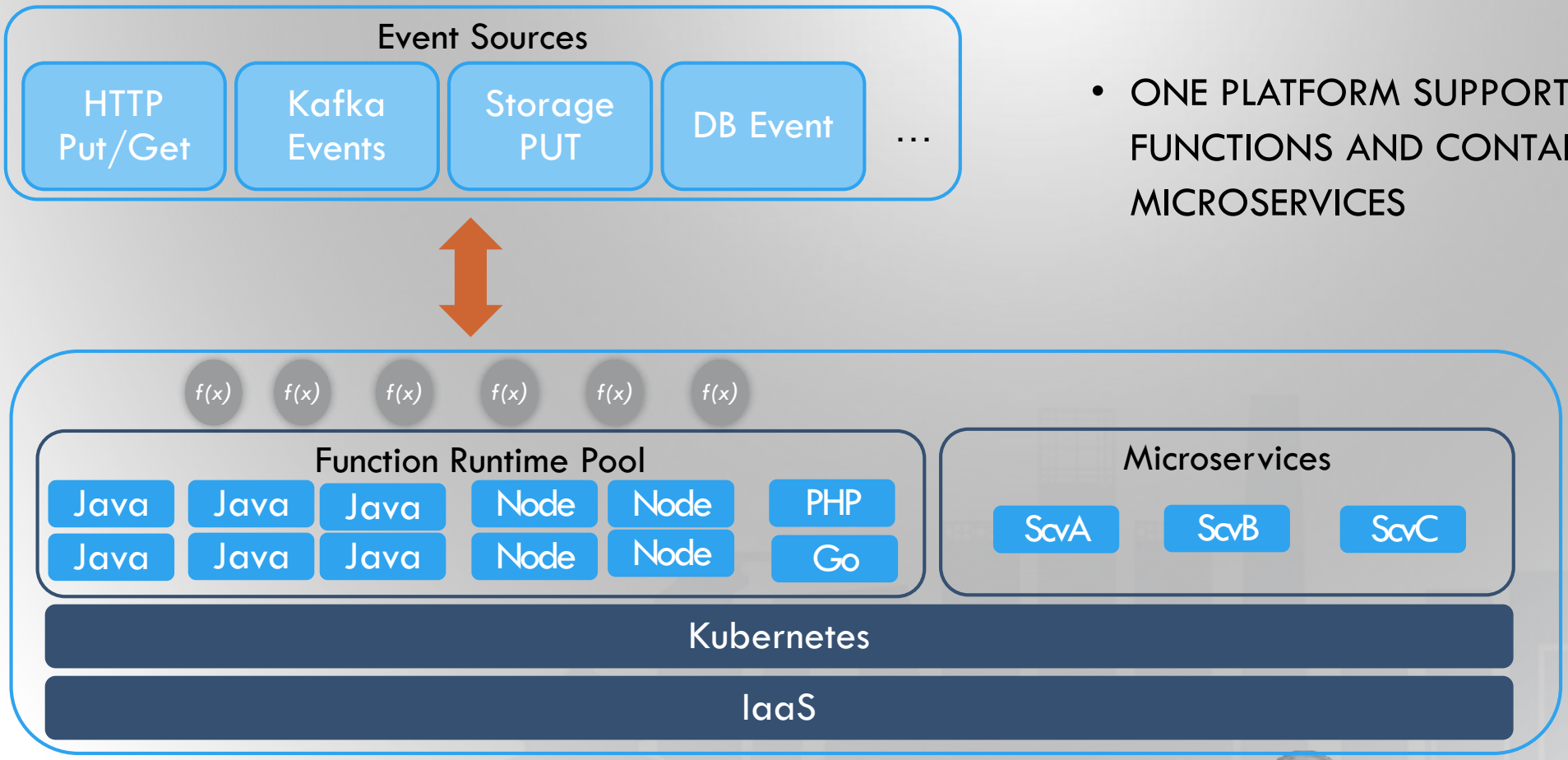
MICROSERVICES AND FUNCTIONS



THE EMERGING EVENT DRIVEN CLOUD



ARCHITECTURE SUPPORTING EVENT DRIVEN CLOUD WITH MICROSERVICES



- ONE PLATFORM SUPPORTING BOTH FUNCTIONS AND CONTAINERIZED MICROSERVICES

HOW CAN WE HELP JAVA DEVELOPERS?

USE AND INNOVATE WITH THE BEST OF WHAT HAS EMERGED IN THE LAST 5 YEARS

Leverage Key Parts of Java EE

JAX-RS, CDI, …

+

Learn from/with Open Source

…

+

Leverage Proven Deployment Architectures

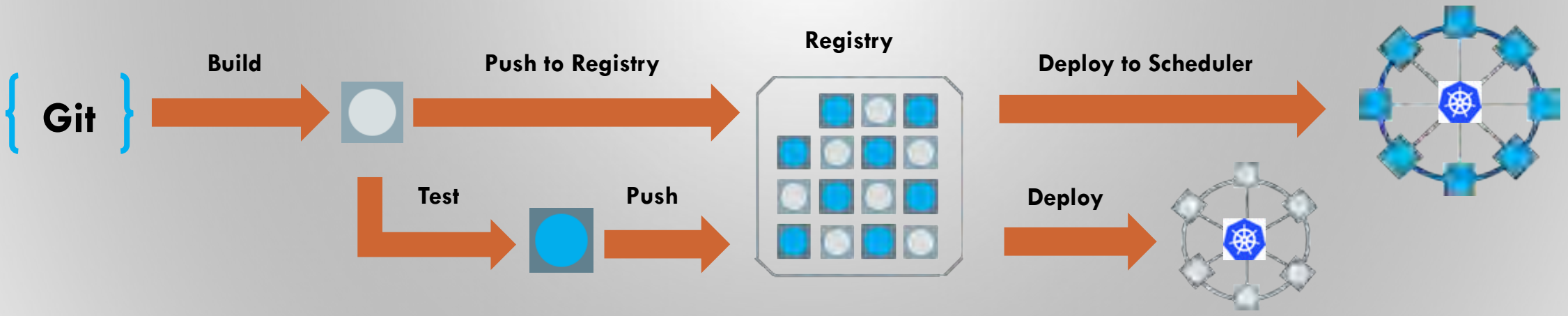
Hysterix,
Archaius
Ribbon
…

SO WHAT PROGRAMMING MODEL DO WE NEED?

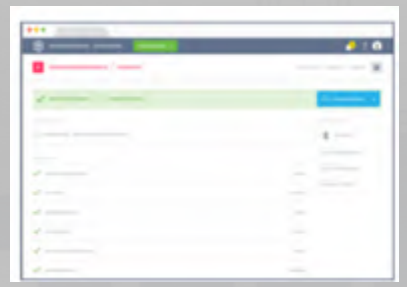
- A PROGRAMMING MODEL THAT WORKS FOR BOTH MICROSERVICES AND SERVERLESS.
  - SUPPORTING ALL PATTERNS NEEDED
  - HELPING WITH DECENTRALIZED DATA
  - HELPING WITH SECURITY
  - SUPPORTING MODERN PARADIGMS, E.G. REACTIVE ETC
- SELF TUNING RUNTIME
  - OPTIMIZE BASED ON RUNTIME BEHAVIOR

CONTINUOUS INTEGRATION AND DELIVERY

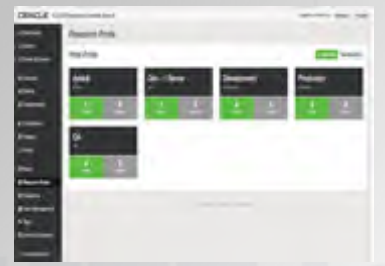
CI/CD SYSTEM THAT WORKS FOR BOTH



OSS CLI



Pipeline/Build Console
Workflow Automation

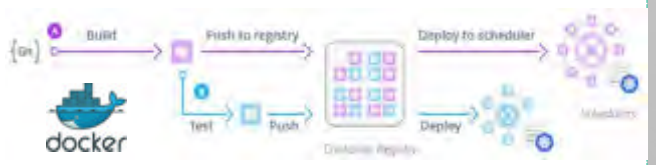


Container Cloud

ORACLE CLOUD DEV PLATFORM

MICROSERVICES, FUNCTIONS AND JAVA

Devops Automation
CICD for Docker



Microservice/Function Framework

spring boot

Java

Dropwizard

NETFLIX OSS

Container Cloud
Container Centric Applications

Functions

Microservices

Kubernetes Container Service

API Management

apiary

Management Cloud
Ops/Diagnostics

Java

Diagnostics

APM

Logs

IT Analytics

SUMMARY

- SERVERLESS IS GAINING MORE AND MORE TRACTION
  - IT'S NOT A ONE SIZE FITS ALL WORLD
- CERTAIN DEVELOPMENT PATTERNS STILL APPLY TO FAAS
- THE NEW HYBRID ARCHITECTURES WILL BE MICROSERVICES WITH FAAS
- EVENT DRIVEN CLOUD WILL BE THE NEW NORMAL
- JAVA IS THE LANGUAGE THAT MAKES IT EASY FOR DEVELOPERS TO BUILD MICROSERVICES, FUNCTIONS OR HYBRID APPLICATIONS

关注QCon微信公众号，
获得更多干货！

Thanks!

INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

主办方 Geekbang. 极客邦科技 InfoQ