Mphasis
The Next Applied

Java Microservices – S1

Manpreet Singh Bindra
Manager

Do’s and Don’t

- Login to GTW session on Time
- Login with your Mphasis Email ID only
- Use the question window for asking any queries

Welcome

1. Skill - Proficiency Introduction
2. About Me - Introduction
3. Walkthrough the Skill on TalentNext
4. About Peer Learning

About Peer Learning Platform

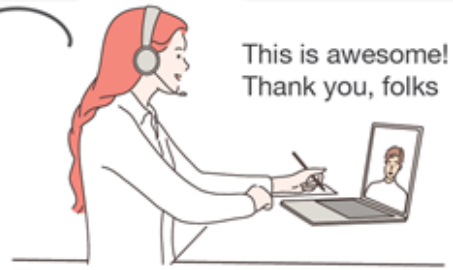
Hi folks! I am preparing for the AWS Certified Developer – Associate Certification. It has been a wonderful experience, so far.

How I wish there was a forum where I could post questions, follow discussions, seek practical insights.

You are in luck, my friend!
Say hello to "Peer Learning" feature on Talent Next! It's a collection of discussion forums on specific skills.

You can follow ongoing discussions, contribute to it by sharing your learnings and create new discussion threads to seek clarification. All these are bound to add to your overall learning experience.

Yes indeed! Peer Learning comes handy to get quick responses to your queries as a wide pool of subject matter experts interact actively



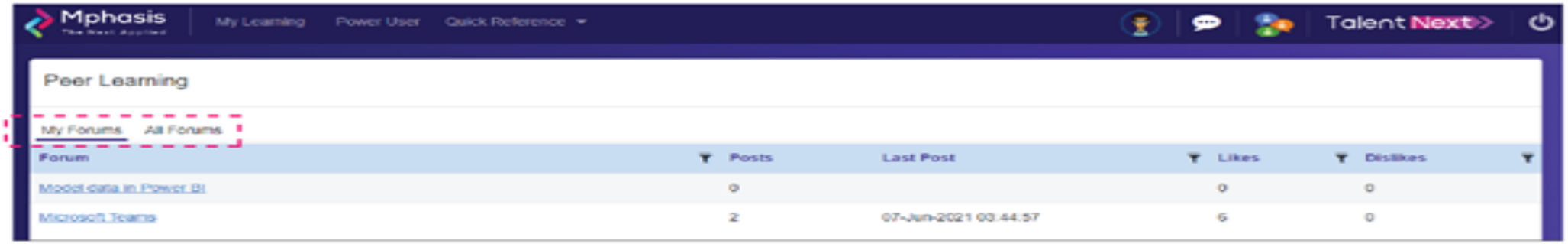
This is awesome!
Thank you, folks

Where to find Peer Learning Platform

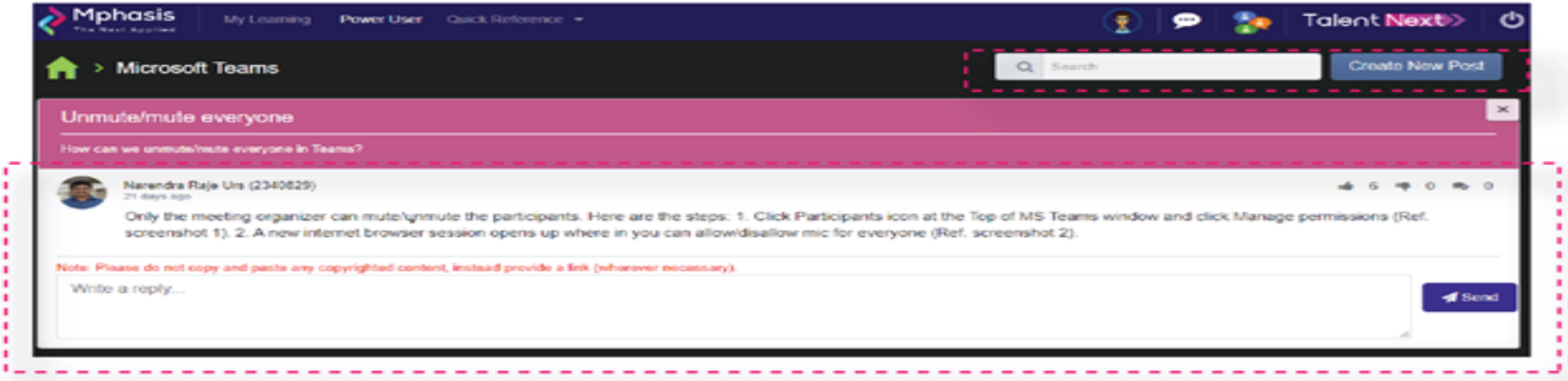
A quick way to get started:

Step 1: Click on the Peer Learning Icon at the top right of the menu bar

Step 2: Select the forum you want to post in from the list



Step 3: Use the "Create New Post" to create a new discussion thread



Overall Agenda

- Use Spring Boot to build standalone web applications and RESTful services
- Understand the Configuration techniques that Spring Boot Provides
- Build Spring boot based Microservices for JSON and XML data exchange
- Monitor services using the Actuator
- Understand the major components of Netflix OSS
- Register services with a Eureka Service
- Implement “client” load balancing with Ribbon to Eureka managed Services
- Isolating from failures with Hystrix
- Filter requests to your Microservices using Zuul
- Define Feign clients to your services
- Scaling Microservices with Spring Cloud

Day - 1

Day – 1 Agenda

- What Is Monolithic Architecture?
- Concerns With the Monolith
- The Microservice architecture
- Characteristics of a Microservice Architecture
- Principles of microservices
- Business demand as a catalyst for Microservices
- Technology as a catalyst for the microservices evolution
- Building microservices with spring boot
- The microservices capability model

Day - 1

Introduction to Microservices

What are Microservices?

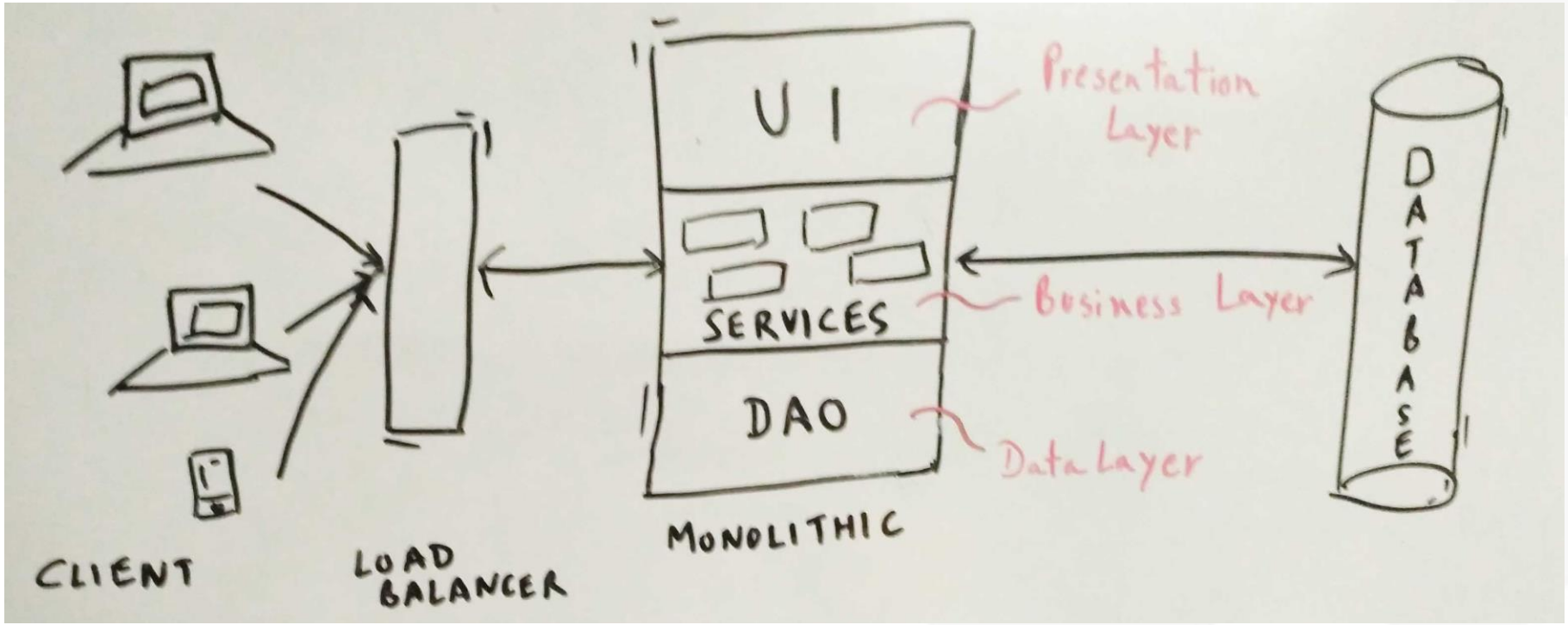
- Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of services that are
  - Highly maintainable and testable
  - Loosely coupled
  - Independently deployable services
  - Organized around business capabilities
  - Owned by a small team

Need for Microservices

- To understand the need for microservices, we need to understand problems with our typical 3-tier monolithic architecture.

What Is Monolithic Architecture?

- Monolithic means composed all in one piece. A monolithic application is one which is self-contained. All components of the application must be present in order for the code to work.



Concerns With the Monolith

- The large monolithic code base
- Overloaded IDE
- Overloaded web container
- Continuous deployment is difficult
- Scaling the application can be difficult
- Obstacle to scaling development
- Requires a long-term commitment to a technology stack

The Microservice architecture

- The microservice architecture enables the rapid, frequent and reliable delivery of large, complex applications. It also enables an organization to evolve its technology stack.
- While there is no precise definition of this architectural style, there are certain common characteristics around organization around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data.

Characteristics of a Microservice Architecture

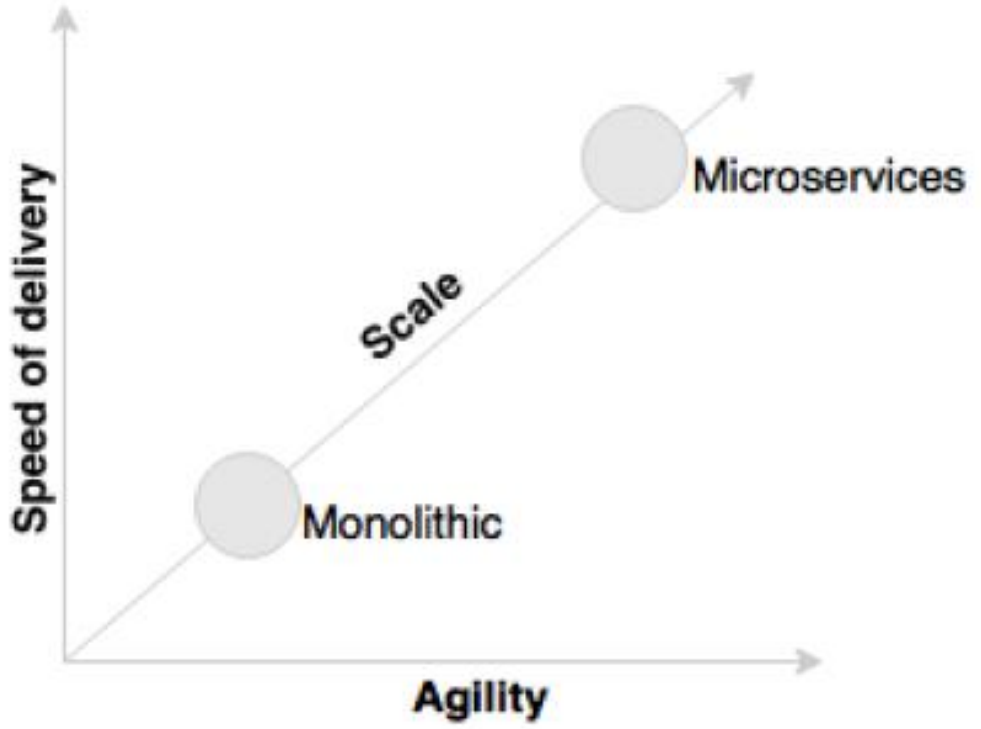
- Componentization via Services
- Organized around Business Capabilities
- Products not Projects (“you build, you run it”)
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure

Business demand as a catalyst for Microservices

- In this era of digital transformation, enterprises increasingly adopt technologies as one of the key enablers for radically increasing their revenue and customer base.
- Enterprises primarily use social media, mobile, cloud, big data, and Internet of Things as vehicles to achieve the disruptive innovations. Using these technologies, enterprises find new ways to quickly penetrate the market, which severely pose challenges to the traditional IT delivery mechanisms.

Business demand as a catalyst for Microservices

- The following graph shows the state of traditional development and microservices against the new enterprise challenges such as agility, speed of delivery, and scale.



Technology as a catalyst for the microservices evolution

- Emerging technologies have also made us rethink the way we build software systems.
- The emergence of HTML 5 and CSS3 and the advancement of mobile applications repositioned user interfaces. Client-side JavaScript frameworks such as Angular, Ember, React, Backbone, and so on are immensely popular due to their client-side rendering and responsive designs.
- With cloud adoptions steamed into the mainstream, **Platform as a Services (PaaS) providers** such as Pivotal CF, AWS, Salesforce.com, IBMs Bluemix, RedHat OpenShift, and so on made us rethink the way we build middleware components.
- The container revolution created by **Docker** radically influenced the infrastructure space. These days, an infrastructure is treated as a commodity service.

Technology as a catalyst for the microservices evolution

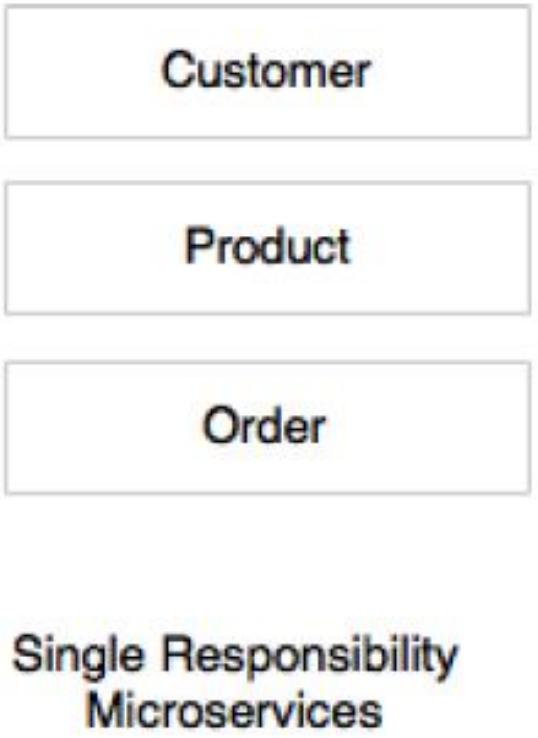
- The integration landscape has also changed with **Integration Platform as a Service (iPaaS)**, which is emerging. Platforms such as Dell Boomi, Informatica, MuleSoft, and so on are examples of iPaaS.
- NoSQLs have revolutionized the databases space. A few years ago, we had only a few popular databases, all based on relational data modeling principles. We have a long list of databases today: Hadoop, Cassandra, CouchDB, and Neo 4j to name a few.

Principles of microservices

- These principles are a "must have" when designing and developing microservices.
  - Single responsibility per service
  - Microservices are autonomous

Single responsibility per service

- The single responsibility principle is one of the principles defined as part of the SOLID design pattern.
- It states that a unit should only have one responsibility.

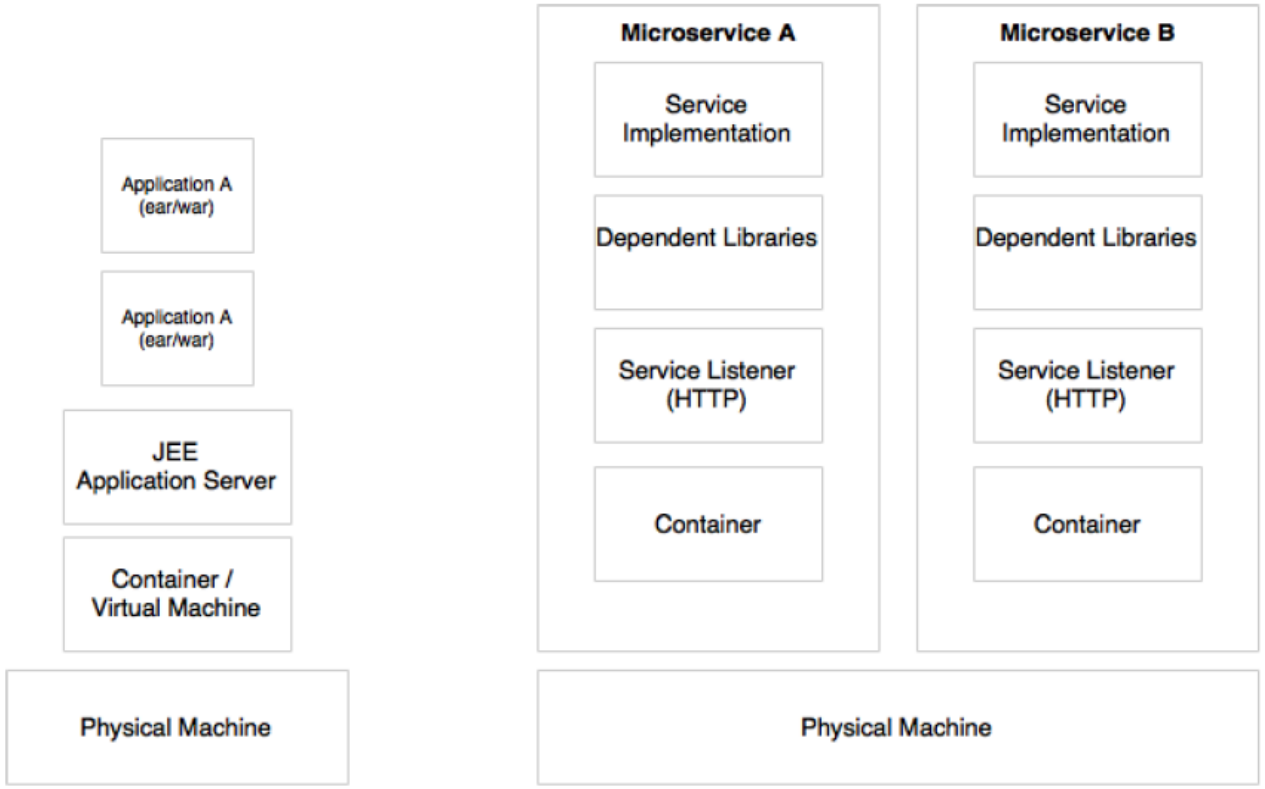


Microservices are autonomous

- Microservices are self-contained, independently deployable, and autonomous services that take full responsibility of a business capability and its execution.
- They bundle all dependencies, including library dependencies, and execution environments such as web servers and containers or virtual machines that abstract physical resources.

Major differences between Microservices and SOA

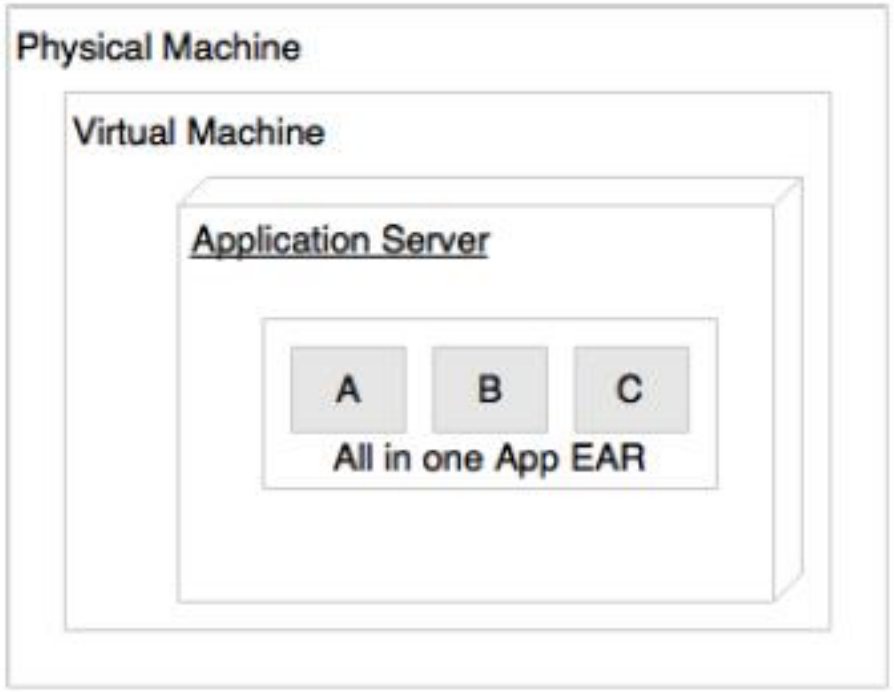
- One of the major differences between microservices and SOA is in their level of autonomy. While most SOA implementations provide service-level abstraction, microservices go further and abstract the realization and execution environment.



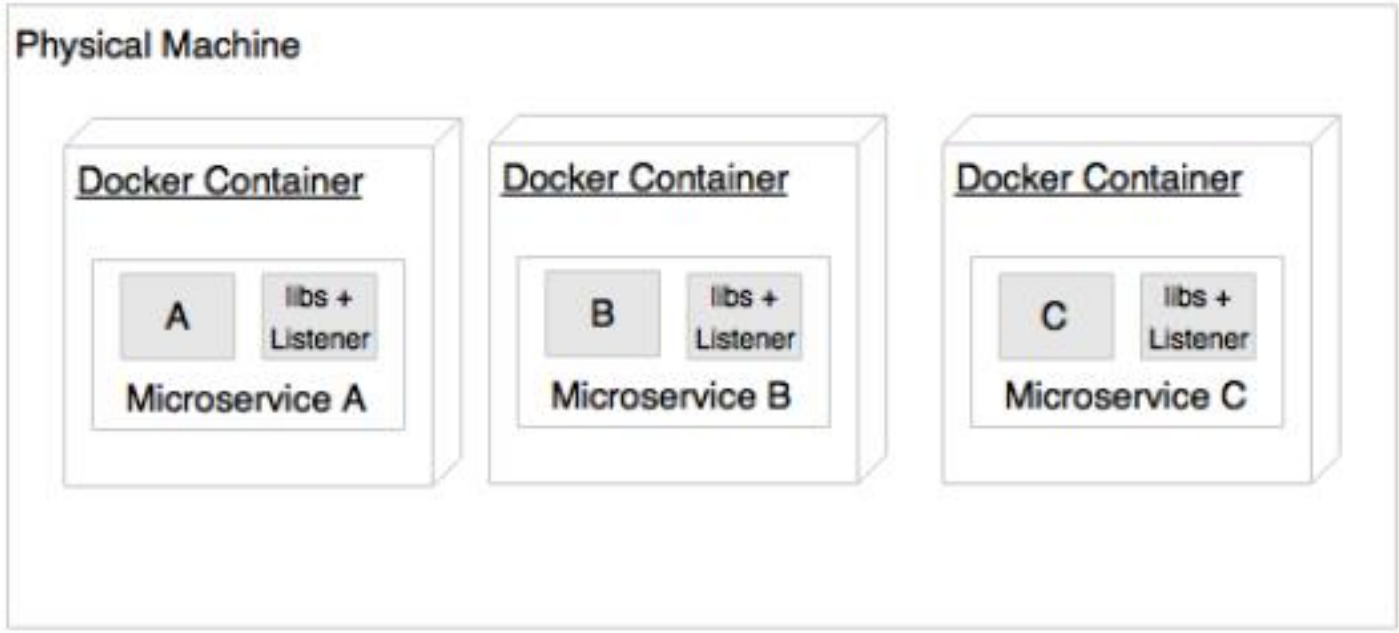
Microservices are lightweight

- Well-designed microservices are aligned to a **single business capability**, so they perform only one function. As a result, one of the common characteristics we see in most of the implementations are microservices with smaller footprints.
- When selecting **supporting technologies**, such as web containers, we will have to ensure that they are also lightweight so that the overall footprint remains manageable. For example, Jetty or Tomcat are better choices as application containers for microservices compared to more complex traditional application servers such as WebLogic or WebSphere.
- **Container technologies such as Docker** also help us keep the infrastructure footprint as minimal as possible compared to hypervisors such as VMWare or Hyper-V.

Microservices are lightweight



Traditonal Deployment

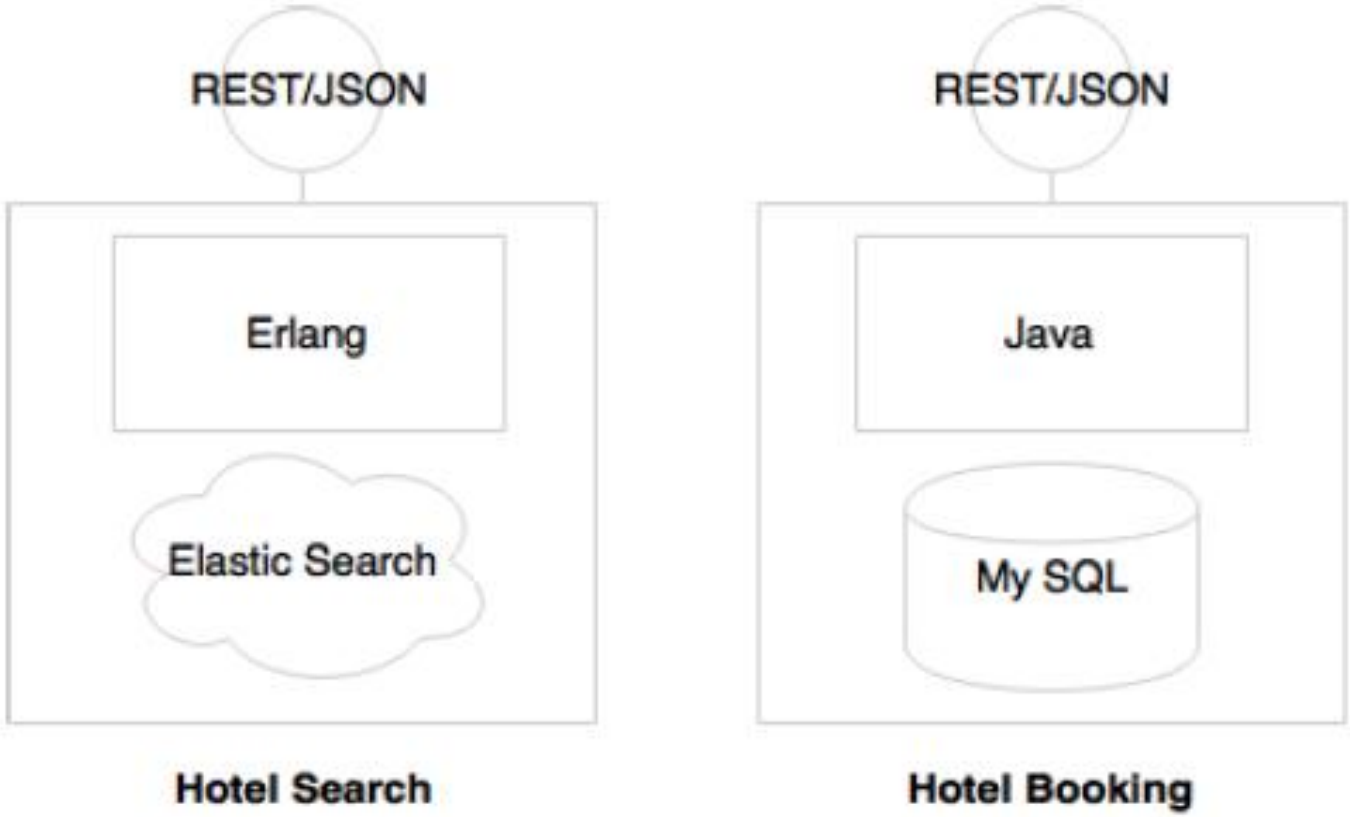


Microservices Deployment

Microservices with polyglot architecture

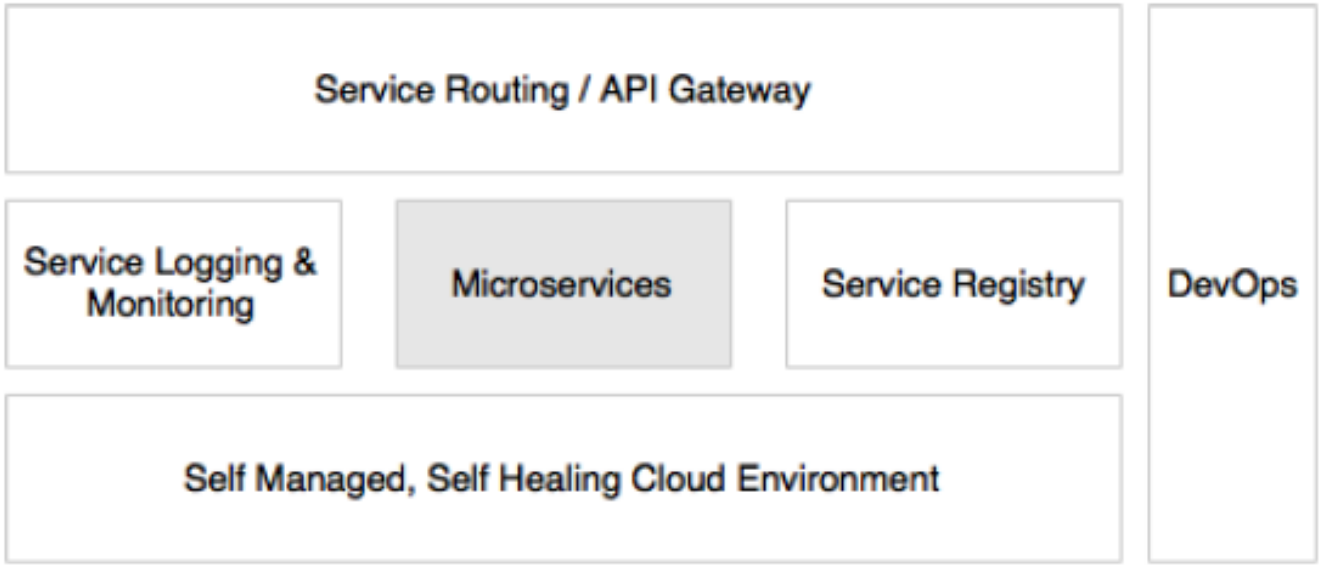
- As microservices are autonomous and abstract everything behind service APIs, it is possible to have different architectures for different microservices.
- A few common characteristics that we see in microservices implementations are:
  - Different services use **different versions** of the same technologies. One microservice may be written on Java 1.7, and another one could be on Java 1.8.
  - **Different languages** are used to develop different microservices, such as one microservice is developed in Java and another one in Scala.
  - **Different architectures** are used, such as one microservice using the Redis cache to serve data, while another microservice could use MySQL as a persistent data store.

Microservices with polyglot architecture



Microservices with a supporting ecosystem

- Most of the large-scale microservices implementations have a supporting ecosystem in place.
- The ecosystem capabilities include
  - DevOps processes
  - Centralized log management
  - Service Registry
  - API Gateways
  - Extensive Monitoring
  - Service Routing
  - Flow control mechanisms



Day - 1

Building Microservices with Spring Boot

Building microservices with spring boot

- There is no "one size fits all" approach when implementing microservices.
- Examining the simple microservices version of this application, we can immediately note a few things in this architecture:
  - Each subsystem has now become an independent system by itself, a microservice.
  - Each service encapsulates its own database as well as its own HTTP listener.
  - Each microservice exposes a REST service to manipulate the resources/entity that belong to this service.

Microservices Benefits

- Supports Polygot Architecture
- Enabling experimentation and innovation
- Elastically and selectively scalable
- Allowing substitution
- Enabling to build organic systems
- Helping reducing technology debt
- Allowing the coexistence of different versions
- Supporting the building of self-organizing systems
- Supporting event-driven architecture
- Enabling DevOps

Relationship with other architecture styles

- We will explore the relationship of microservices with other closely related architecture styles such as SOA and Twelve-Factor Apps

Relations with SOA

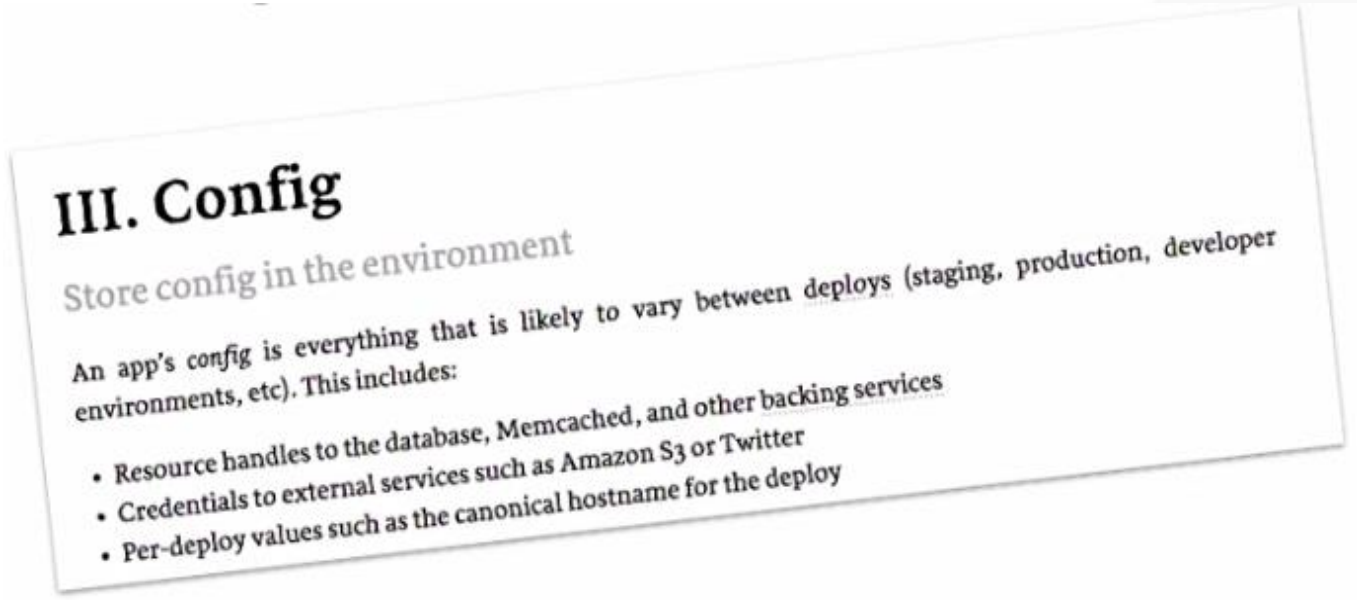
- The definition of SOA from The Open Group consortium is as follows:

  "Service-Oriented Architecture (SOA) is an architectural style that supports service orientation. Service orientation is a way of thinking in terms of services and service-based development and the outcomes of services.

- A service:
  - Is a logical representation of a repeatable business activity that has a specified outcome
  - It is self-contained.
  - It may be composed of other services.
  - It is a "black box" to consumers of the service."

Relations with Twelve-Factor apps

- Twelve-Factor App, forwarded by Heroku, is a methodology describing the characteristics expected from modern cloud-ready applications.
- Twelve-Factor App is equally applicable for microservices as well. Hence, it is important to understand Twelve-Factor App.
- See 12factor.net/config



III. Config

Store config in the environment

An app's *config* is everything that is likely to vary between deploys (staging, production, developer environments, etc). This includes:

- Resource handles to the database, Memcached, and other backing services
- Credentials to external services such as Amazon S3 or Twitter
- Per-deploy values such as the canonical hostname for the deploy

Microservices early adopters

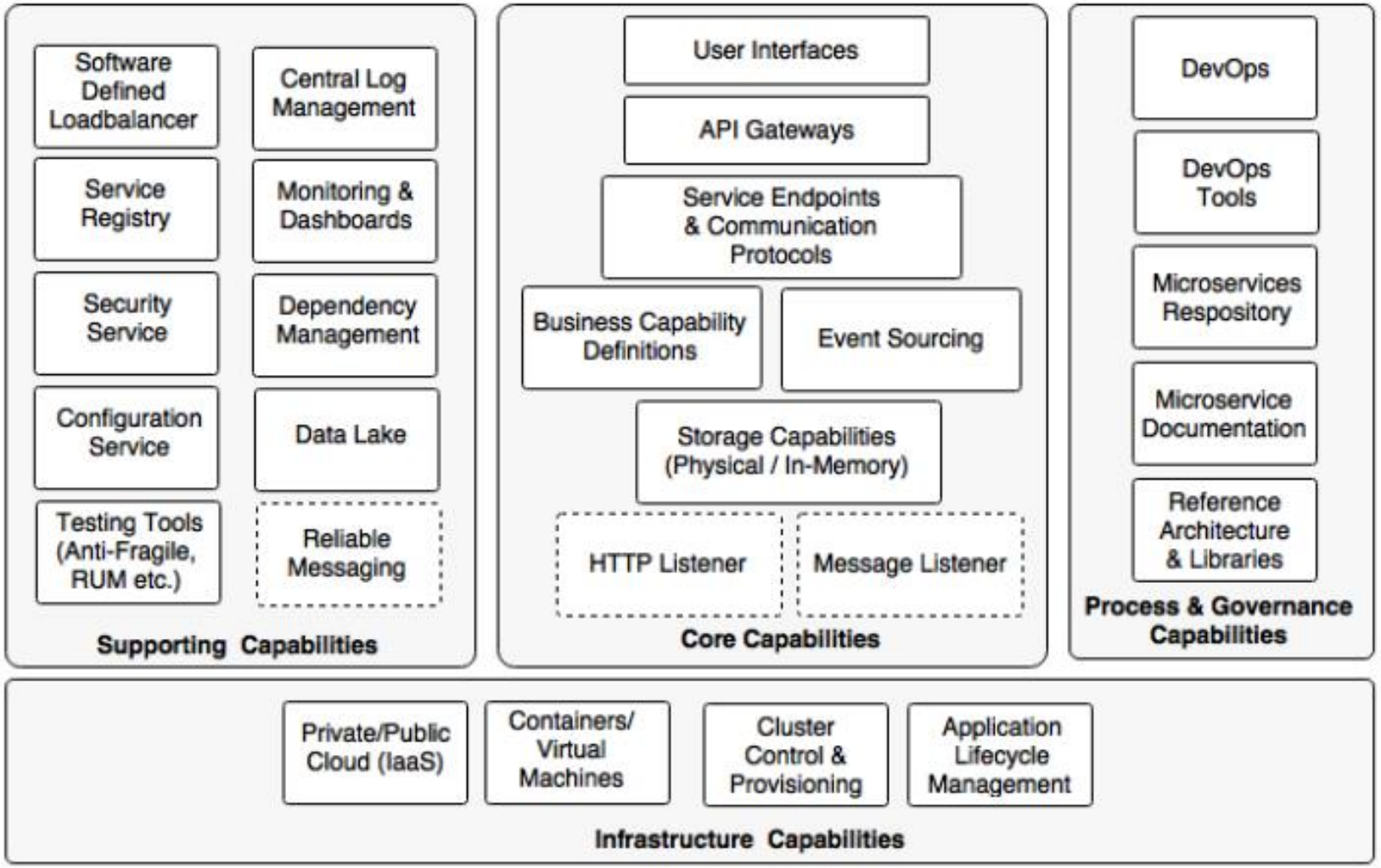
- Netflix
- Uber
- Airbnb
- Orbitz
- eBay
- Amazon
- Gilt
- Twitter
- Nike

Day - 1

The Microservices Capability Model

The microservices capability model

- The following diagram depicts the microservices capability model:



The microservices capability model

The capability model is broadly classified in to four areas:

- **Core capabilities**: These are part of the microservices themselves
- **Supporting capabilities**: These are software solutions supporting core microservice implementations
- **Infrastructure capabilities**: These are infrastructure level expectations for a successful microservices implementation
- **Governance capabilities**: These are more of process, people, and reference information

Core capabilities

The core capabilities are explained as follows:

- Service listeners (HTTP/messaging)
- Storage capability
- Business capability definition
- Event sourcing
- Service endpoints and communication protocols
- API gateway
- User interfaces

Infrastructure capabilities

The Infrastructure capabilities are explained as follows:

- Cloud
- Containers or virtual machines
- Cluster control and provisioning
- Application lifecycle management

Supporting capabilities

The Supporting capabilities are explained as follows:

- Software defined Load Balancer
- Central log management
- Service registry
- Security service
- Service configuration
- Testing tools (anti-fragile, RUM and so on)
- Monitoring and dashboards
- Dependency and CI management
- Reliable Messaging

Process and governance capabilities

The Process and governance capabilities are explained as follows:

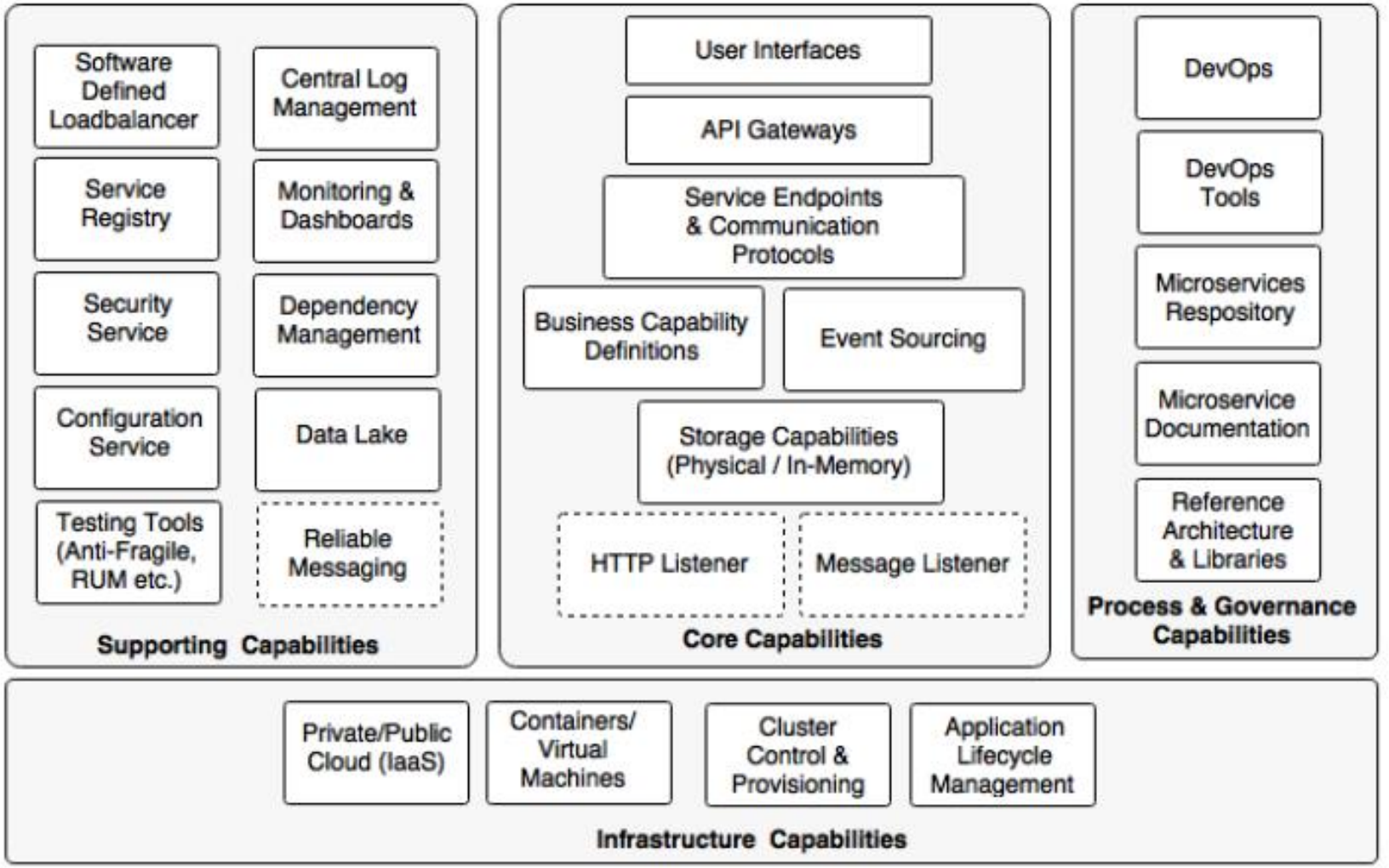
- DevOps
- DevOps tools
- Microservices repository
- Microservices documentation
- Reference architecture and libraries

Reviewing microservices capabilities

We will explore the following microservices capabilities from the microservices capability model:

- Software Defined Load Balancer
- Service Registry
- Configuration Service
- Reliable Cloud Messaging
- API Gateways

Reviewing microservices capabilities



Recap of Day – 1

- What Is Monolithic Architecture?
- Concerns With the Monolith
- The Microservice architecture
- Characteristics of a Microservice Architecture
- Principles of microservices
- Business demand as a catalyst for Microservices
- Technology as a catalyst for the microservices evolution
- Building microservices with spring boot
- The microservices capability model

Additional Recommended Reading

- Spring Microservices in Action
- Spring Boot Intermediate Microservices
- Enterprise Java Microservices

Mphasis
The Next Applied

THANK YOU

About Mphasis

Mphasis (BSE: 526299; NSE: MPHASIS) applies next-generation technology to help enterprises transform businesses globally. Customer centricity is foundational to Mphasis and is reflected in the Mphasis' Front2Back™ Transformation approach. Front2Back™ uses the exponential power of cloud and cognitive to provide hyper-personalized ($C=X2C^{2}_{TM}=1$) digital experience to clients and their end customers. Mphasis' Service Transformation approach helps 'shrink the core' through the application of digital technologies across legacy environments within an enterprise, enabling businesses to stay ahead in a changing world. Mphasis' core reference architectures and tools, speed and innovation with domain expertise and specialization are key to building strong relationships with marquee clients. Click here to know

Important Confidentiality Notice

Any Questions?

Manpreet.Bindra@mphasis.com

FOLLOW US IN THE LINK BELOW

@Mphasis