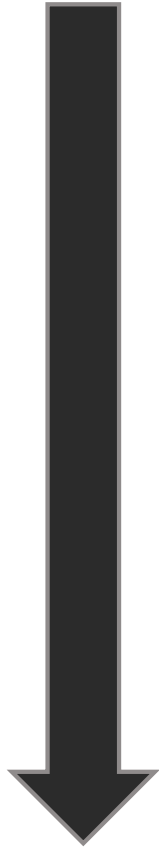


# Application Modernization Workshop

- WW Developer Advocate Team

## IBM Developer

# Evolution of application architectures



IBM Developer

Late 90's	<b>Enterprise Application (EAI) Services and Models</b> Addressed integration and transactional challenges primarily by using message oriented middleware. Mostly proprietary systems needing a proliferation of custom interfaces.
Mid 00's	<b>Service Oriented Architectures</b> Based on open protocols like SOAP and WSDL making integration and adoption easier. Usually deployed on an Enterprise ESB which is hard to manage and scale.
Early 10's	<b>API Platforms and API Management</b> REST and JSON become the defacto standard for consuming backend data. Mobile apps become major consumers of backend data. New Open protocols like OAuth become available further simplifying API development .
2015 and beyond	<b>Microservice Architecture</b> Applications are composed of small, independently deployable processes communicating with each other using language-agnostic APIs and protocols.

# Key tenets of a microservices architecture

1. Large are monoliths are broken down into many small services
  - Each service runs into its own process
  - Generally accepted rule is one service per container
2. Services are optimized for a single function
  - One business function per service
    - The service will have only one reason to change
3. Services are tightly encapsulated behind concrete programming interfaces
  - Have to balance between evolving the interface and maintaining backward compatibility
4. Communication via REST API and/or message brokers
  - Avoids tight coupling and allows for flexibility of synchronous and asynchronous access
5. Per-service continuous integration and continuous deployment (CI/CD)
  - Services can evolve at different rates
6. Per-service HA and clustering
  - Services can be scaled independently at different rates as needed

# Why microservices ?

- **Efficient teams**
  - End to end team ownership of relatively small codebases
    - Teams can innovate faster and fix bugs more quickly
- **Simplified deployment**
  - Each service is individually changed, tested, and deployed without affecting other services
    - Time to market is accelerated.
- **Right tools for the job**
  - Teams can use best of breed technologies, libraries, languages for the job at hand
    - Leads to faster innovation
- **Improved application quality**
  - Services can be tested more thoroughly in isolation
    - Better code coverage
- **Scalability**
  - Services can be scaled independently at different rates as needed
    - Leads to better overall performance at lower cost

# Microservices Dos and Don'ts

## 1. **Don't assume all monolithic apps are microservice candidates**

- Stable monolithic applications that aren't subject to frequent updates will not provide the return on investment. Good candidate applications have multiple upgrades per year, lots of business rules requiring complex regression testing and require extended service outages to implement

## 2. **Don't even think about microservices without DevOps**

- Microservices cause an explosion of moving parts. It is not a good idea to attempt to implement microservices without serious deployment and monitoring automation

## 3. **Don't try to migrate all external dependencies at once**

- Microservices often introduce multiple databases, message brokers, data caches, and similar services that all need to be maintained, clustered, and kept in top shape. It really helps if your first attempt at microservices is free from such concerns

## 4. **Don't create too many microservices**

- Each new microservice uses resources. Cumulative resource usage might outstrip the benefits of the architecture if you exceed the number of microservices that your DevOps organization, process, and tooling can handle

## 5. **Don't forget to keep an eye on the potential latency issue**

- Making services too granular or requiring too many dependencies on other microservices can introduce latency. Care should be taken when introducing additional microservices

## Microservices Dos and Don'ts (cont'd)

### 6. Do manage the dependency matrix







### 7. Do realize you're dealing with an ever changing deployment environment


- Questions like the following may have different answers at any given point in time
  - » “where are my applications running?!”
  - » “where is my data/metrics/logs?!?!?”

### 8. Do stick with Java if you're an experienced Java developer

- Learning a new language introduces a new layer of complexity as you're getting familiar with a new development paradigm

The Matrix of Hell

	Static website	?	?	?	?	?	?	?
	Web frontend	?	?	?	?	?	?	?
	Background workers	?	?	?	?	?	?	?
	User DB	?	?	?	?	?	?	?
	Analytics DB	?	?	?	?	?	?	?
	Queue	?	?	?	?	?	?	?
	Development VM	QA Server	Single Prod Server	On-pre Cluster	Public Cloud	Contributor's laptop	Customer Servers	



# Microservices Architecture – Cultural change considerations

- **Smaller teams with broader scope**

- Mini end to end development orgs in each team vs large silos across the entire development team

- **Top down support with bottom up execution**

- Change can't happen effectively w/o executive sponsorship
- Change needs to be executed at the smallest organizational unit to take hold

- **Teams own all metrics related to operations and development**

- Have to minimize downtime + number of bugs while also maximizing the rate at which needed features are added and minimizing the time to market of those new features

- **Trust**

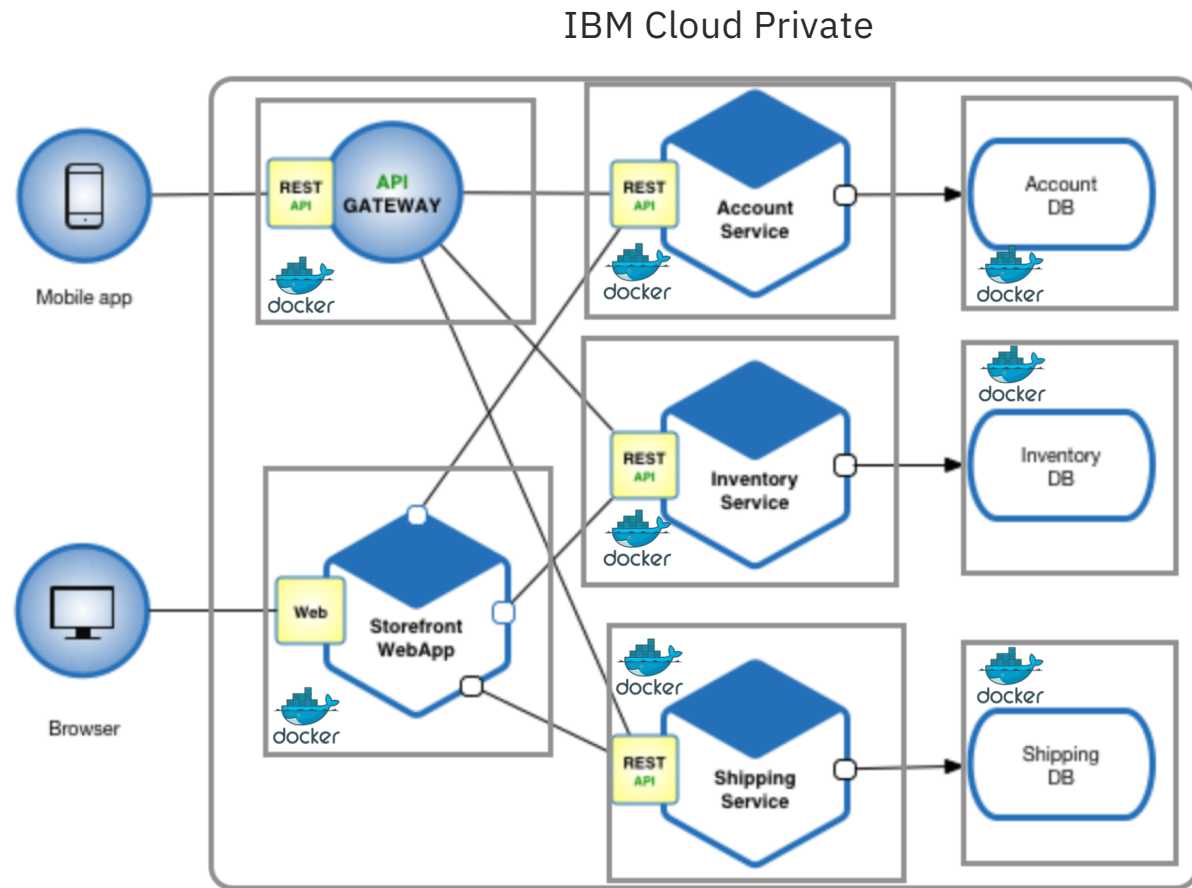
- Teams need to build trust with other teams that they collaborate with rather than relying on one size fits all checklists and rules

- **Reward based on results not compliance**

- Cultures only change when people are measured and rewarded for outcomes consistent with the changes
- Smaller more autonomous teams work better with less central micromanagement and more focus on broad measurable goals

## Technologies for microservice transformation

- Containers (Docker)
- Container orchestration (Kubernetes)
- IBM Cloud Private
- Transformation Advisor
- 12-Factor Best Practices
- CI/CD tools (e.g Jenkins)





# Lab Environment

