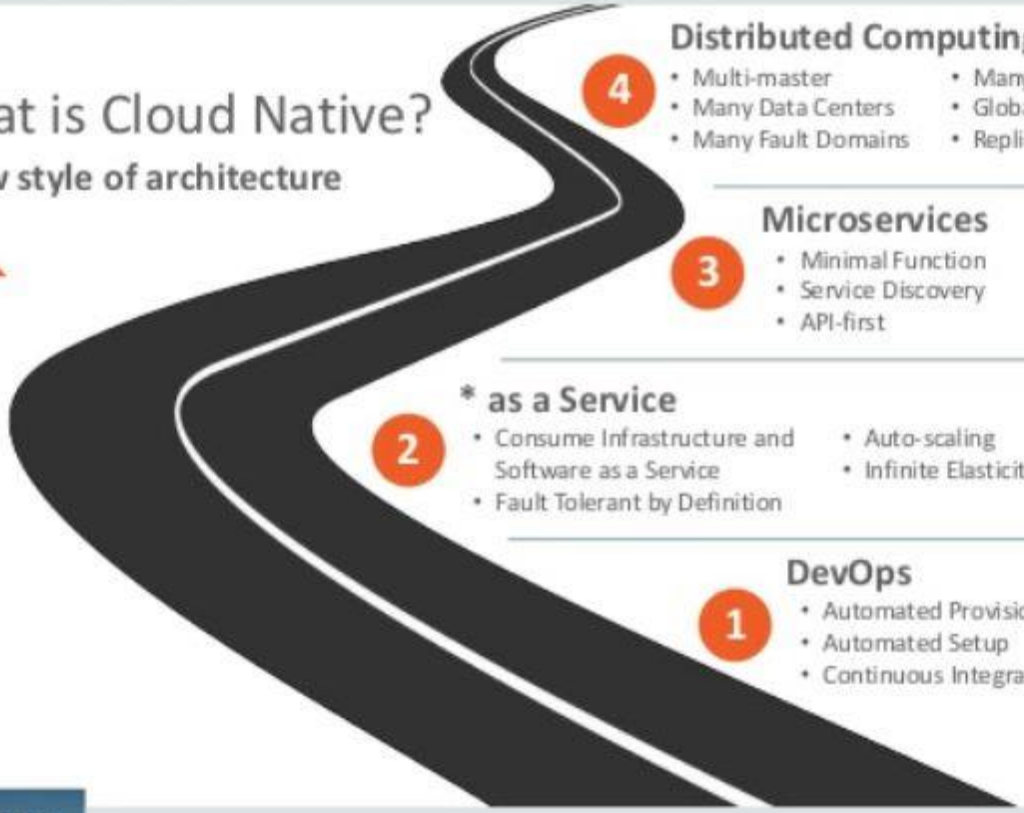
What is Cloud Native?

A new style of architecture



4

Distributed Computing

- Multi-master
- Many Data Centers
- Many Fault Domains
- Many Regions
- Global Server Load Balancing
- Replication

3

Microservices

- Minimal Function
- Service Discovery
- API-first
- Polyglot
- Choreography
- Loose Coupling

2

* as a Service

- Consume Infrastructure and Software as a Service
- Fault Tolerant by Definition
- Auto-scaling
- Infinite Elasticity

1

DevOps

- Automated Provisioning
- Automated Setup
- Continuous Integration
- Continuous Delivery
- Automated Testing
- Agile
- Culture Change

Clip slide

Clip slide

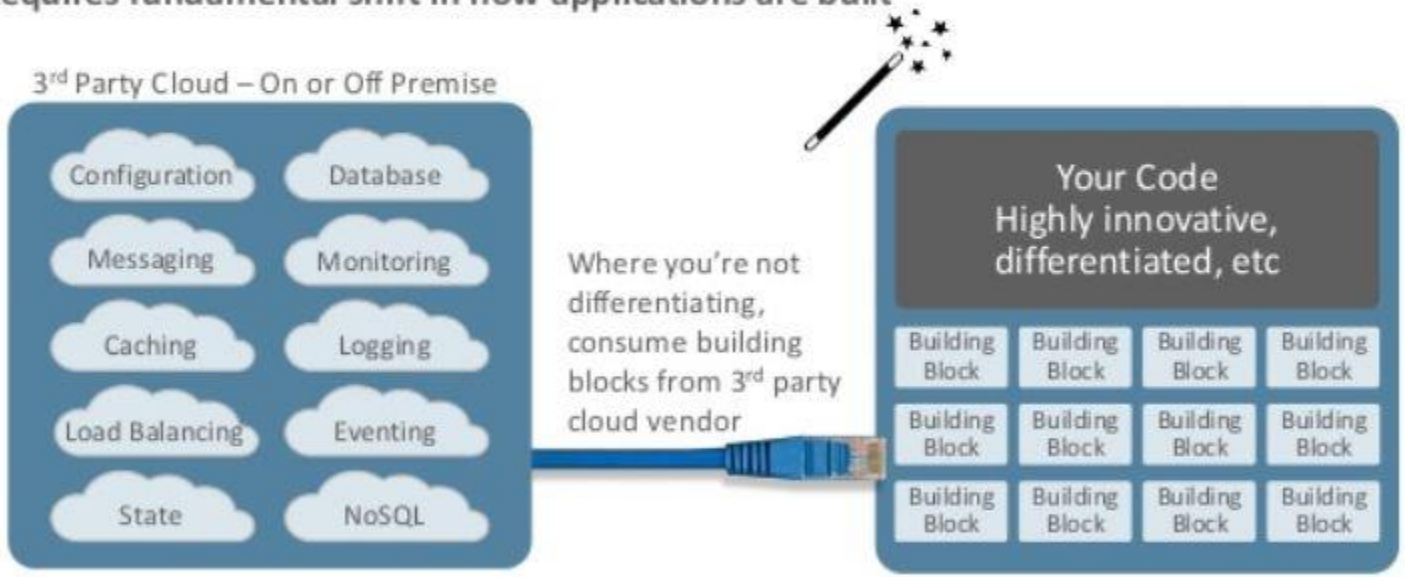
Prerequisite #1 - DevOps

A prerequisite to consuming infrastructure and software as a service

Clip slide

Prerequisite #2 - * as a Service

Requires fundamental shift in how applications are built

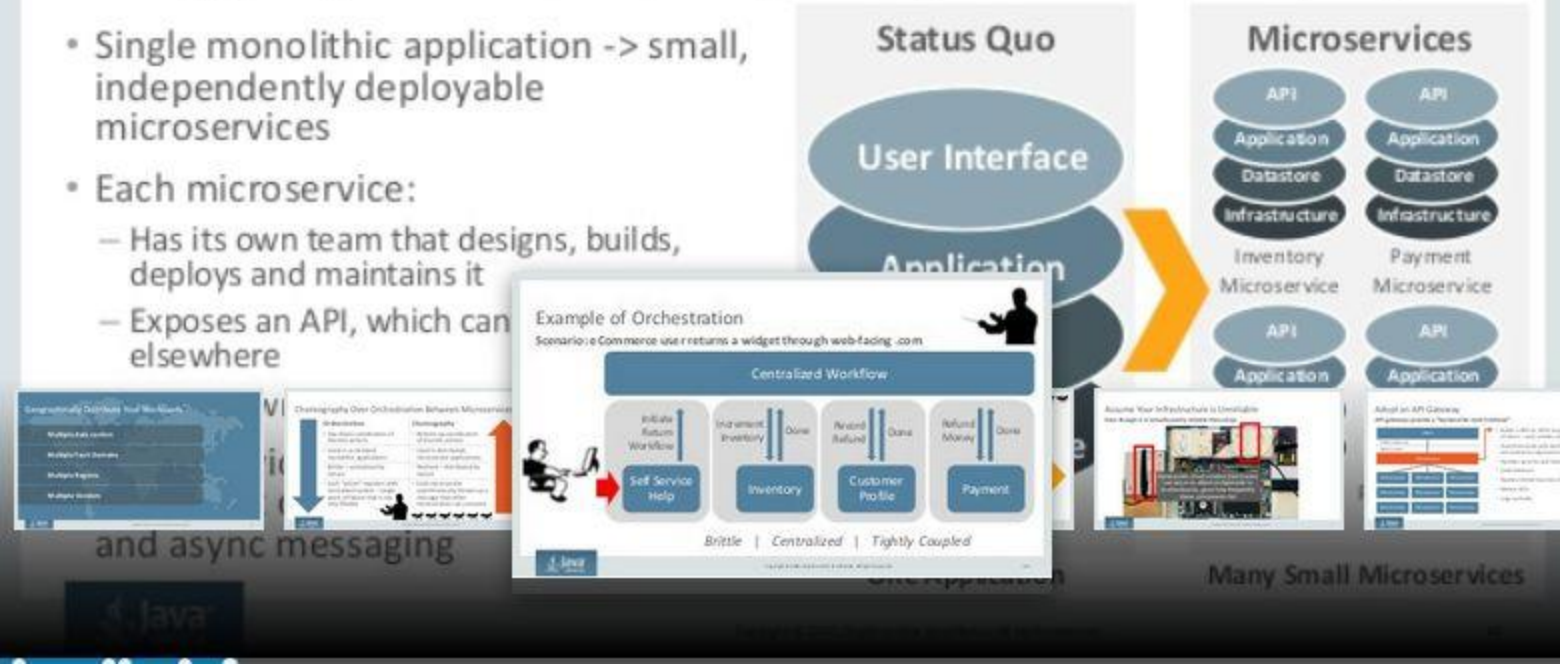


Clip slide

Prerequisite #3 – Microservices

Break up your application into many small pieces to get features to market quickly

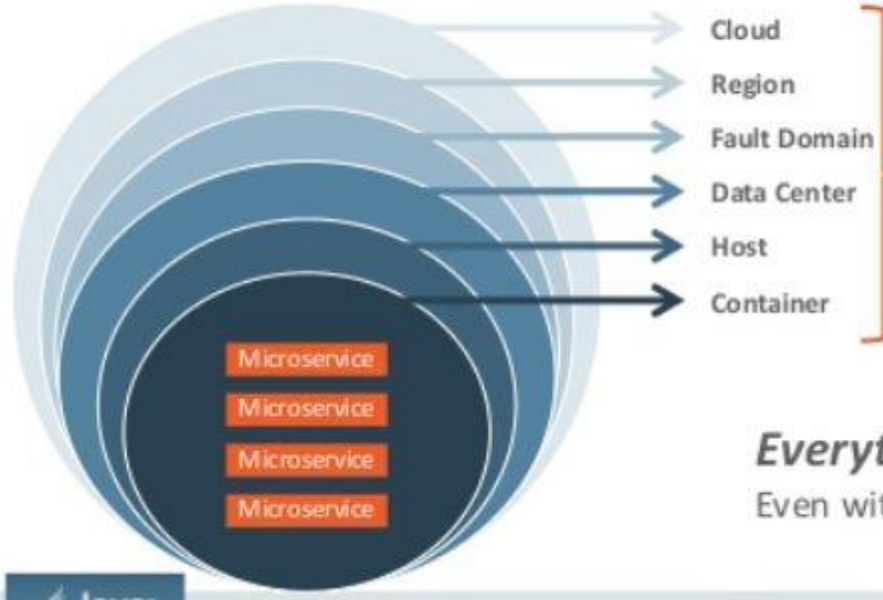
- Single monolithic application -> small, independently deployable microservices
- Each microservice:
  - Has its own team that designs, builds, deploys and maintains it
  - Exposes an API, which can
    elsewhere



Clip slide

Prerequisite #4 – Distributed Computing

Run your applications across multiple data centers, fault domains, regions, etc



***Hundreds* of milliseconds of latency**

Everything is now distributed

Even within the same data center

Clip slide

What's the Value of Cloud Native?

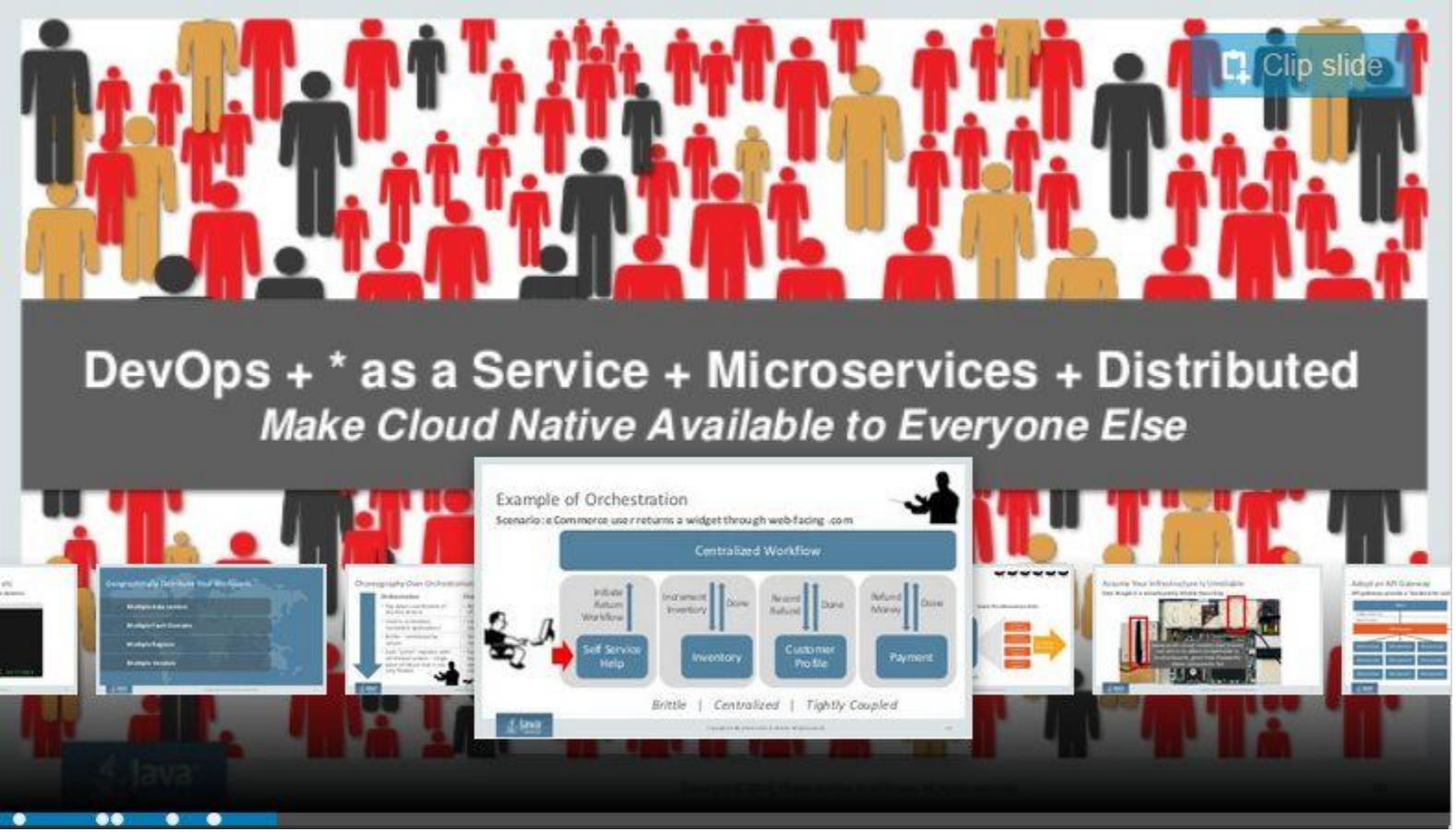
#1 Value – Quickly Capitalize on Business Opportunities

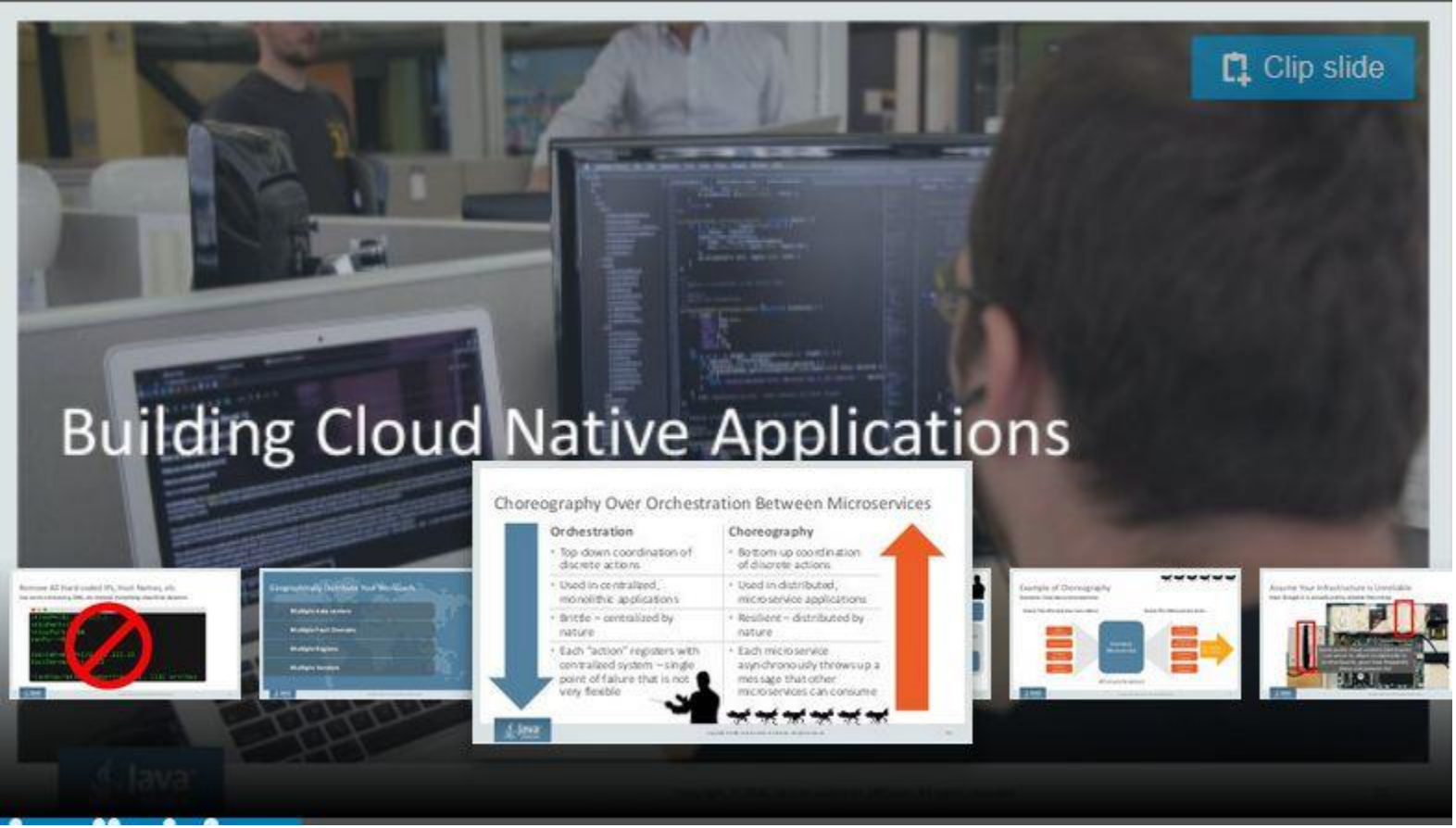
SPEED

Quickly move changes

RES

Y


Clip slide
DevOps + * as a Service + Microservices + Distributed
Make Cloud Native Available to Everyone Else
Example of Orchestration
Scenario: e Commerce user returns a widget through web-facing .com
Centralized Workflow
Initiate Return Workflow
Instrument Inventory
Done
Record Refund
Done
Refund Money
Done
Self Service Help
Inventory
Customer Profile
Payment
Brittle | Centralized | Tightly Coupled
Java


Clip slide
Building Cloud Native Applications
Choreography Over Orchestration Between Microservices
Orchestration
Top-down coordination of discrete actions
Used in centralized, monolithic applications
Brittle – centralized by nature
Each "action" registers with centralized system – single point of failure that is not very flexible
Choreography
Bottom-up coordination of discrete actions
Used in distributed, microservice applications
Resilient – distributed by nature
Each microservice asynchronously throws up a message that other microservices can consume
Java

Cloud Native Takes Real Organizational Commitment

The legacy style has inertia

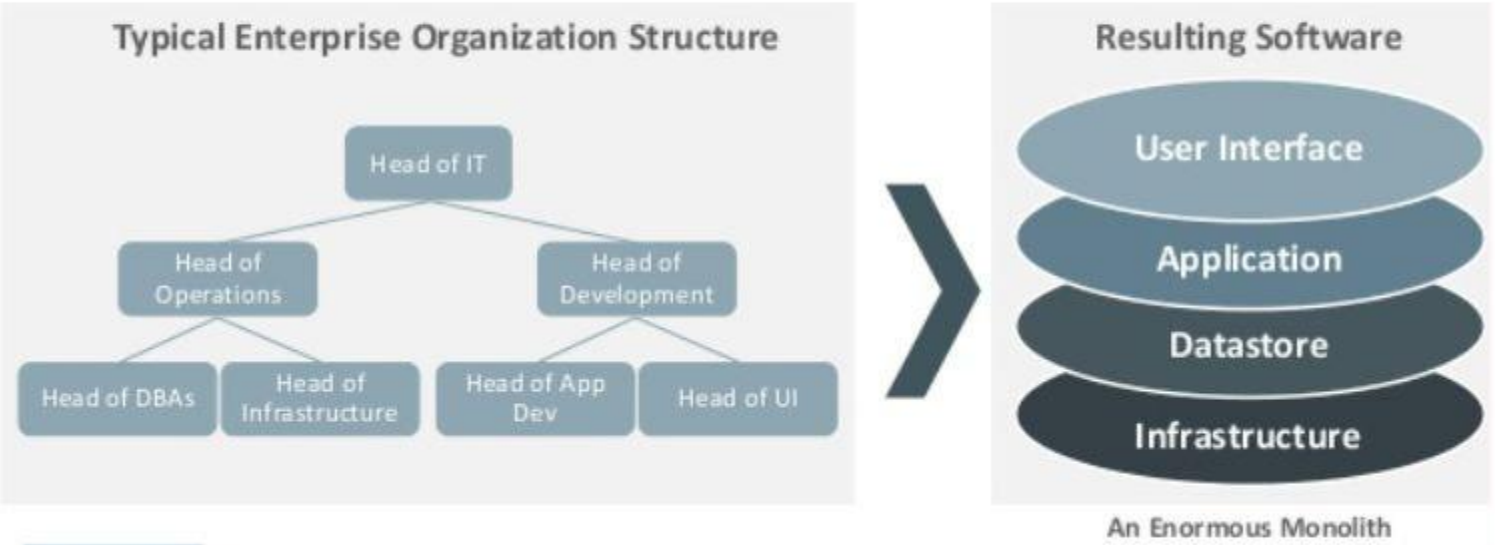
- LARGE HORIZONTAL LAYERS FOCUSED ON PROJECTS-> SMALL VERTICAL TEAMS FOCUSED ON PRODUCTS
- CHANGE HOW PEOPLE ARE REWARDED
- CHANGE PEOPLE'S JOBS

RETHINK **YOUR ORGANIZATION**

Clip slide

Horizontally Tiered Enterprises == Horizontally Tiered Apps

Conway's Law: Software reflects the structure of the organization that produced it



Clip slide

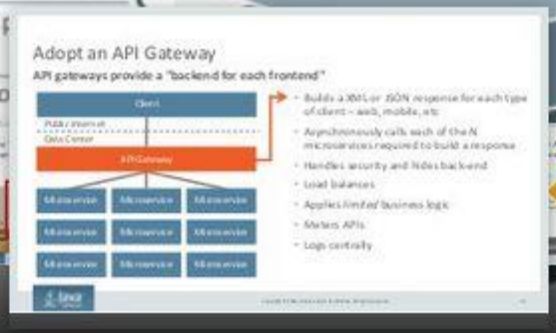
Even Simple Changes Are Hard to Implement With Monoliths

Organizational boundaries introduce the need to extensively coordinate

New requirement: Add a birthdate property to the customer's profile. How does this get implemented?

3. Application developer tickets UI team to have them add that property to the profile screens
2. Application developer adds the new p… the application-level code
1. Application developer tickets DBAs to …

Different Teams
Different Timelines
Different Priorities



Clip slide

Characteristics of Different Organization Types

Centralized Organizations
Focused on Technology Layers

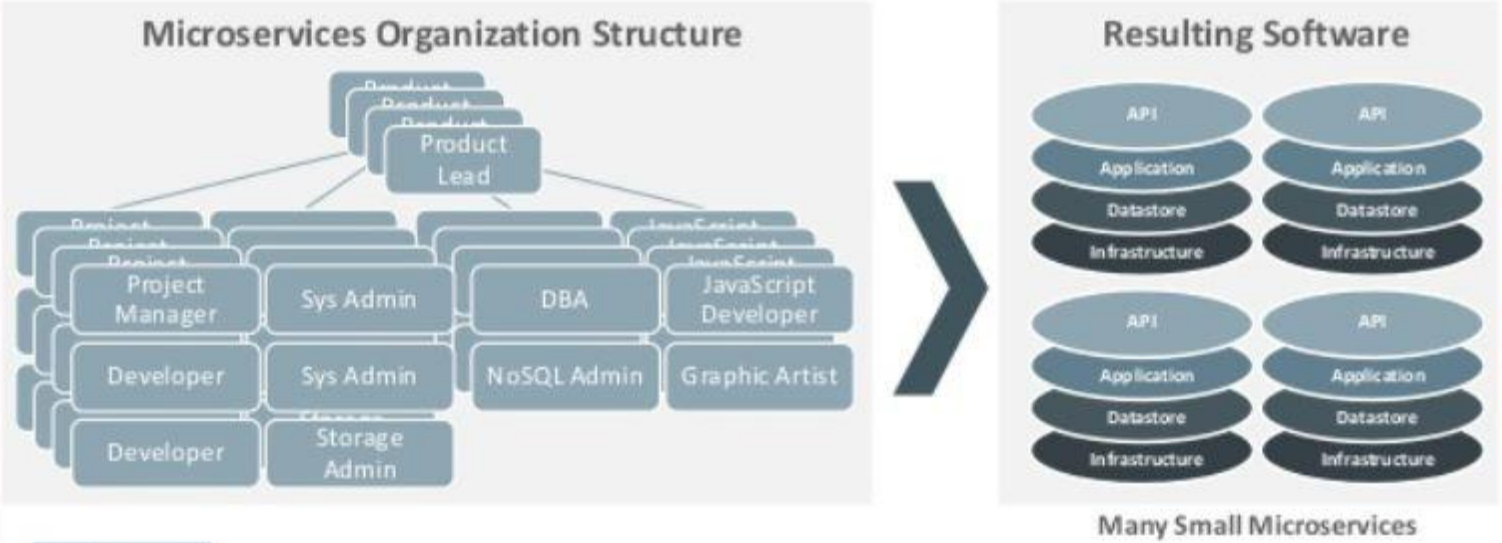
- Produce monoliths
- Simple change requires extensive coordination across all of the different layers
- Business logic is spread everywhere because it's easier to bury it in the layer you own
- No real ownership

Distributed Organizations
Focused on Products

- Produce microservices
- A team can make any change to their microservice
- Architecture more clean because it's not done by central committee
- **A lot of freedom and responsibilities**
- **True ownership**

Re-structure Your Organization – Put Conway's Law to Work

Build small product-focused teams – *strict one team to one microservice mapping*



Small Teams = Much Better Communication

Low Latency/High Bandwidth Communication

Legacy

services

Java

Clip slide

Adopt DevOps

Requires changes throughout your entire organization

Culture

Respect	• Dev respect for ops • Ops respect for dev
Discuss	• Ops should be in dev discussions • Dev should be in ops discussions • Shared runbooks
Avoid Blaming	• No fingerpointing! • Everyone should have some culpability
"Done" Means Released	• Dev's responsibility shouldn't ever end – production support required • "Throwing it over the wall" is dead

Technology

Infra as Code	• Don't build envs by hand • Scripts checked in and managed as src
Shared Version Control	• Single system • Ship trunk • Enable features through flags
One Step Build/Deploy	• One button build/deploy • If verification fails, stop and alert or take action
Don't Fix Anything	• If something breaks, re-deploy. Don't fix • Fix environment setup scripts

Clip slide

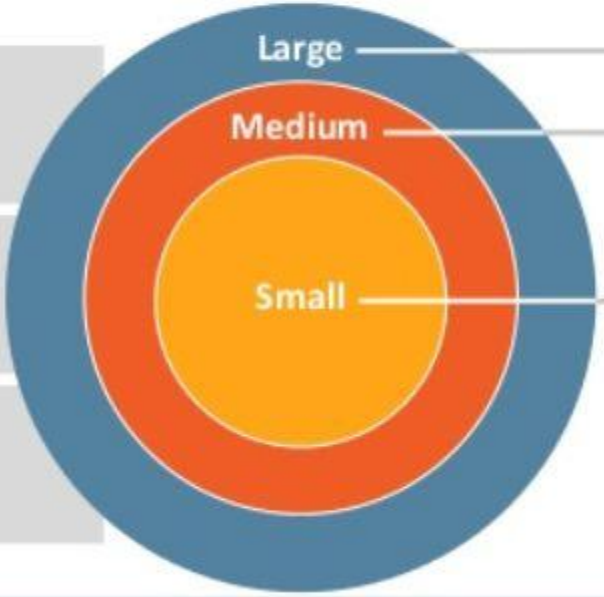
Start Managing Small, Vertical Teams

Can have hundreds of microservices for a larger application

Do One Thing and Do It Well

Avoid Inter-dependencies

Focus on Business Capabilities



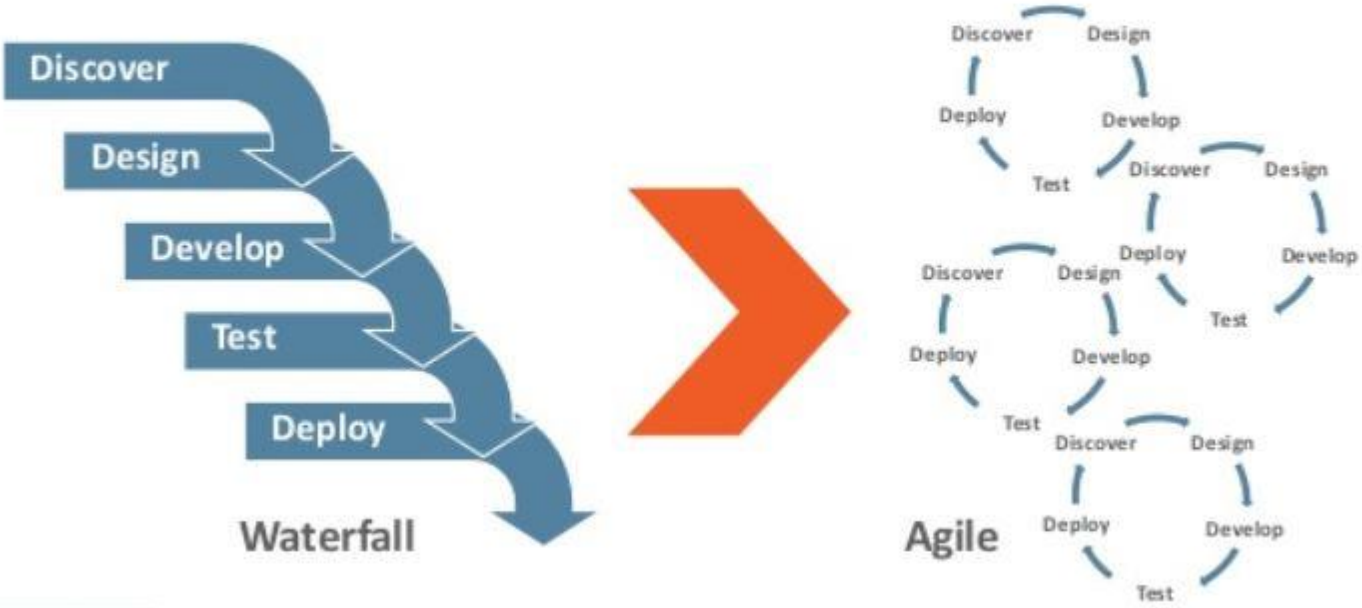
11-15 People
Example: Order Microservice

4-10 People
Example: Inventory Microservice

1-3 People
Example: Order Status Microservice

Clip slide

Change to a Form of Agile Project Management

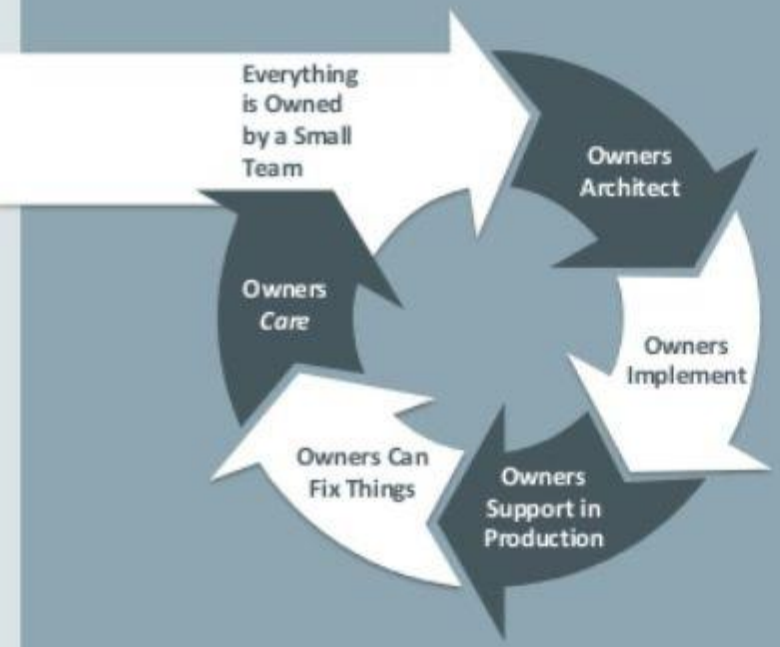


Clip slide

Hold Each Person on Each Team Accountable

Much easier with microservices-style architecture

Clip slide



Ownership is Key to the Success of Cloud Native

In traditional enterprises, any one individual has very low ownership of anything. It's classic tragedy of the commons

Clip slide

Consume Application Building Blocks as Software

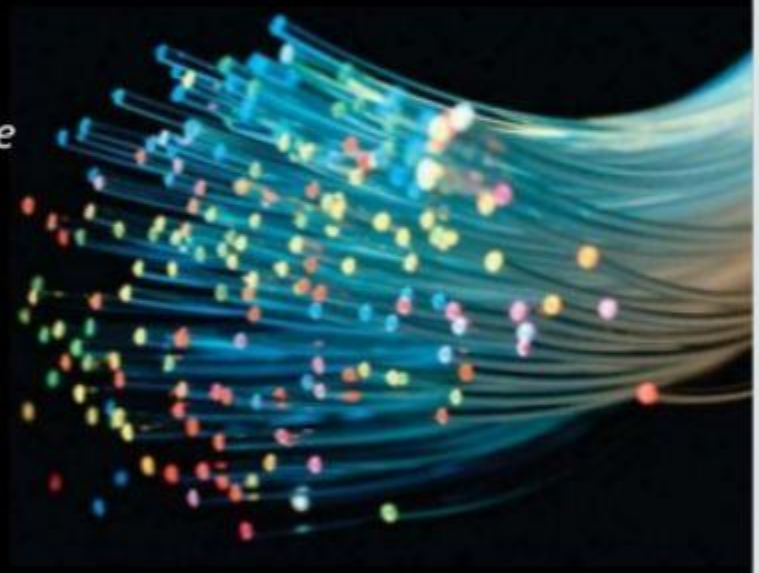
Cloud (*-as-a-Service) is key to innovation

Start consuming resources as a *service*

Building Blocks
Database, NoSQL, Messaging, etc

Platforms
Java EE, Java SE, Node, etc

Infrastructure
Compute, Networking, Storage



Clip slide

Empower Developers to Make Procurement Decisions

- #1 focus of cloud native: time to market. Long-term maintenance should not be a big consideration
- Let developers who are innovating pick the absolute best technology for their own use
- Each small team supports their own microservice in perpetuity. No need to have large maintenance teams
- Standardization is best for system of record-style applications

Clip slide

Each Microservice Can Now Use Its Own Database/Datastore

RDBMS is great but not necessary for all use cases

Relational Database	NoSQL	Object Grids
ACID-compliant, suitable for a wide range of workloads. Trusted, reliable, wide client support, easy to use	Non-relational organization of data, including key/value, graph, document, tabular, etc. Always BASE and sometimes ACID compliant	Store objects in and move business logic into the server-side grid.
ORACLE, MySQL, SQL Server, PostgreSQL	ORACLE BERKELEY DB, mongoDB, Couchbase, Cassandra	hazelcast, ORACLE Coherence, GEMFIRE, Infinispan

Java

Clip slide

What Are Microservices?

Minimal function services that are deployed separately but can interact together to achieve a broader use-case

Monolithic Applications

- Single, Monolithic App
- Must Deploy Entire App
- One Database for Entire App
- Organized Around Technology Layers
- State In Each Runtime Instance
- One Technology Stack for Entire App
- In-process Calls Locally, SOAP Externally

Microservices

- Many, Smaller Minimal Function Microservices
- Can Deploy Each Microservice Independently
- Each Microservice Has Its Own Datastore
- Organized Around Business Capabilities
- State is Externalized
- Choice of Technology for Each Microservice
- REST Calls Over HTTP, Messaging, or Binary

Clip slide

Three Key Rules to Microservices

Can **build** a microservice independently

Can **release** each microservice independently

Everything Is Now Distributed. Get Used To It

Applications

Applications are now broken up into small microservices, with separate frontends and backends

Infrastructure

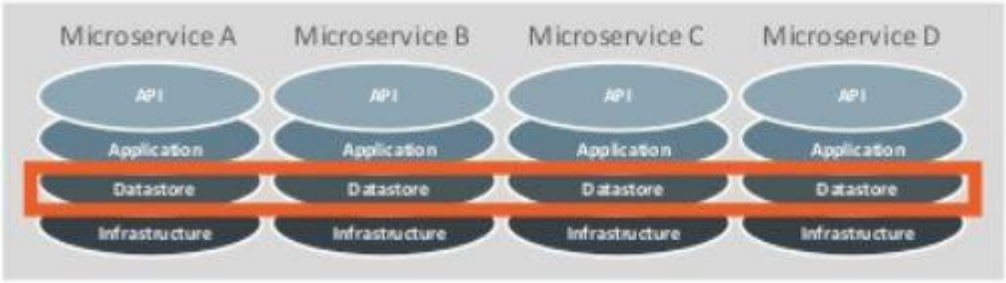
Different data centers, fault domains, regions, etc. Even within the same data center, latency may be unacceptably high

Teams

Many small teams, each responsible for their own microservice. Each team is often geographically distributed

Microservices Forces Move To Distributed Computing

Introduces enormous complexity – monoliths don't suffer from this



- Distributed computing is a natural consequence of microservices because each microservice has its own datastore
- Sharing datastores across microservices introduces coupling – very bad!
- There will always be latency between microservices
- Latency = eventual consistency

- All data exchange between microservices must be through API layer or messaging – no accessing datastores cross-microservices
- Must implement high-speed messaging for *synchronous* calls between microservices. REST + HTTP probably isn't fast enough
- May end up duplicating data across datastores – e.g. a customer's profile

Clip slide

Choreography Over Orchestration Between Microservices

Orchestration

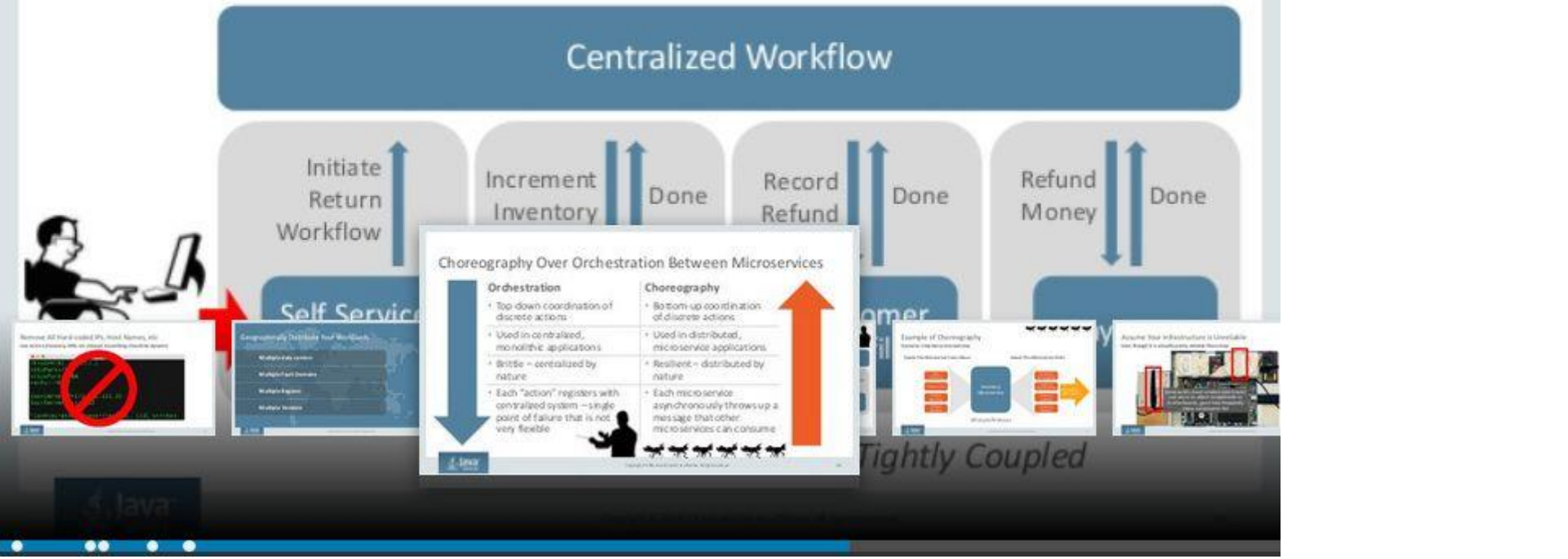
- Top-down coordination of discrete actions
- Used in centralized, monolithic applications
- Brittle – centralized by nature
- Each "action" registers with centralized system – single point of failure that is not very flexible

Choreography

- Bottom-up coordination of discrete actions
- Used in distributed, microservice applications
- Resilient – distributed by nature
- Each microservice asynchronously throws up a message that other microservices can consume

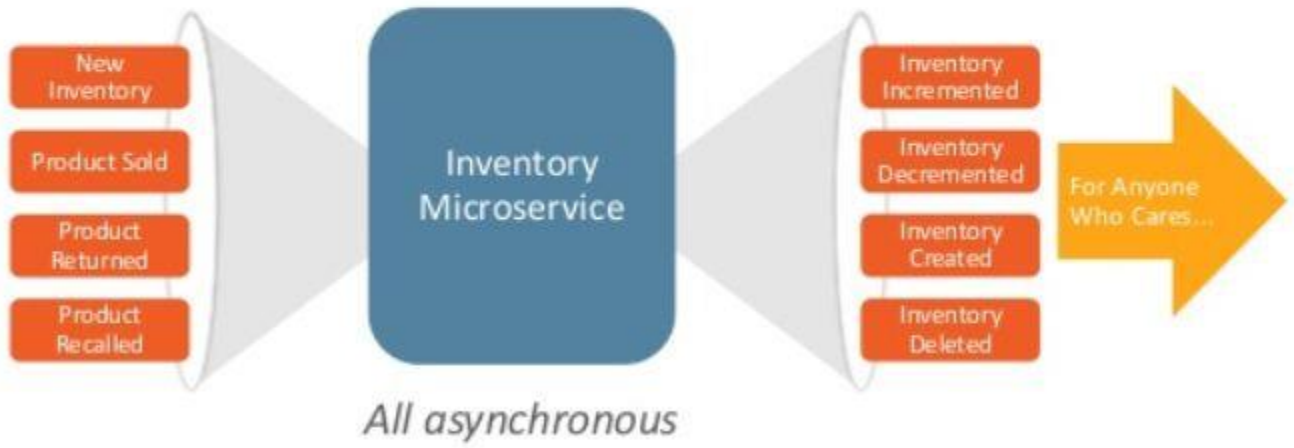
Example of Orchestration

Scenario: eCommerce user returns a widget through web-facing .com



Example of Choreography

Scenario: Inventory microservice



All asynchronous