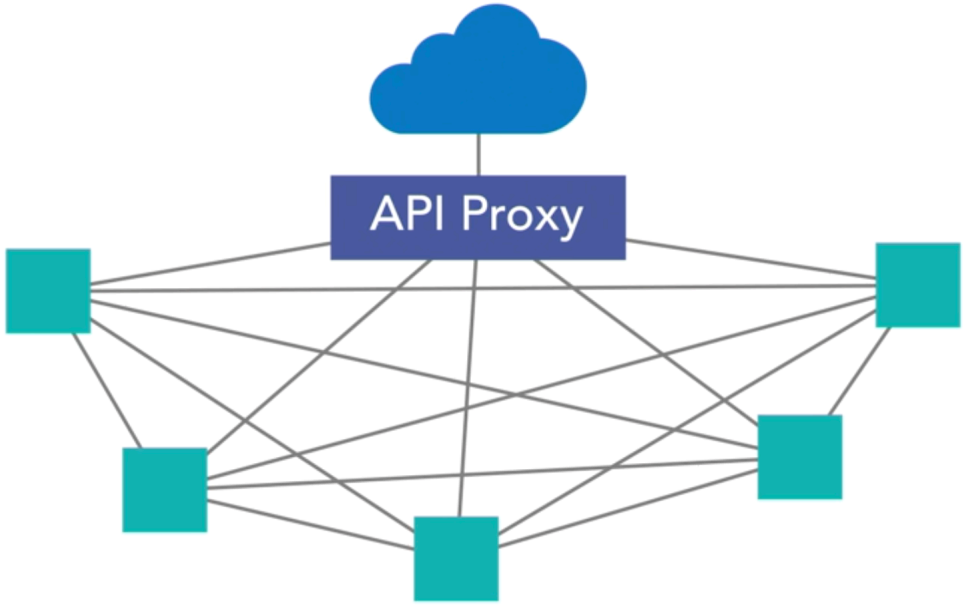
Microservices

Monolithic Application

- Tight-coupling
- Mass resources for every code changes
- Affect the speed to deploy critical features

Microservices

- **Decomposition**: breaking a software problem into smaller pieces that are easier to understand and solve
- All communications over ReST



Each unit of work can be called by any unit of work within the system

Cost

- Complexity: move from few deployment components to a lot
- Distribution tax: dramatic increase in network communication between individual services; latency
- System Reliability

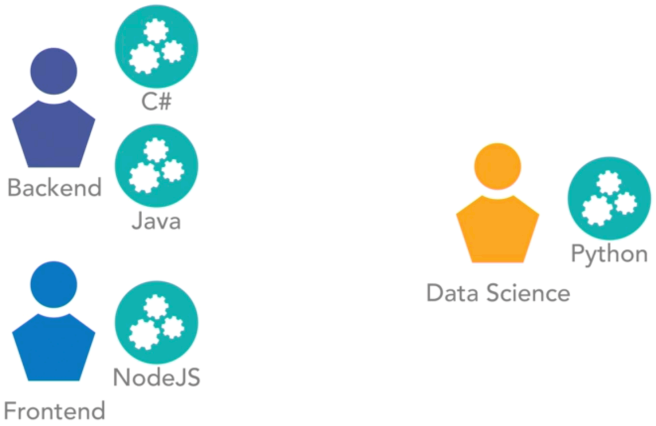
Cloud-Native Microservice-Based

- Single code base
- Completely self-contained
- Zero file system usage

Internal Server Communication

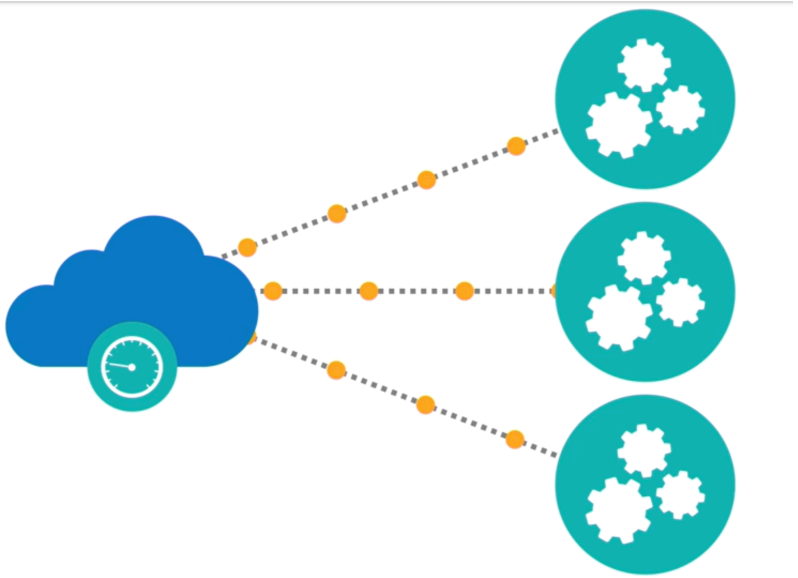
- All communication between individual services in a microservices architecture is over HTTP using ReST based services

- Allows for the use of any coding language or framework that supports ReSTful services



Distribution and Scale

- Allows global distribution (remote network call)
- Can be individually scaled
  - Increase the number instances of customer service when experiencing load



Circular Calls

- Latency can become a problem

Solution

- Circuit Breaker
  - Trigger the circuit to do default behavior when timeout

Domain Driven Design

- Investigate working system
- Determine the domains
- Break services up accordingly
- Reduce latency: analyze the traffic and determine the bounded context to reduce cross-domain calls

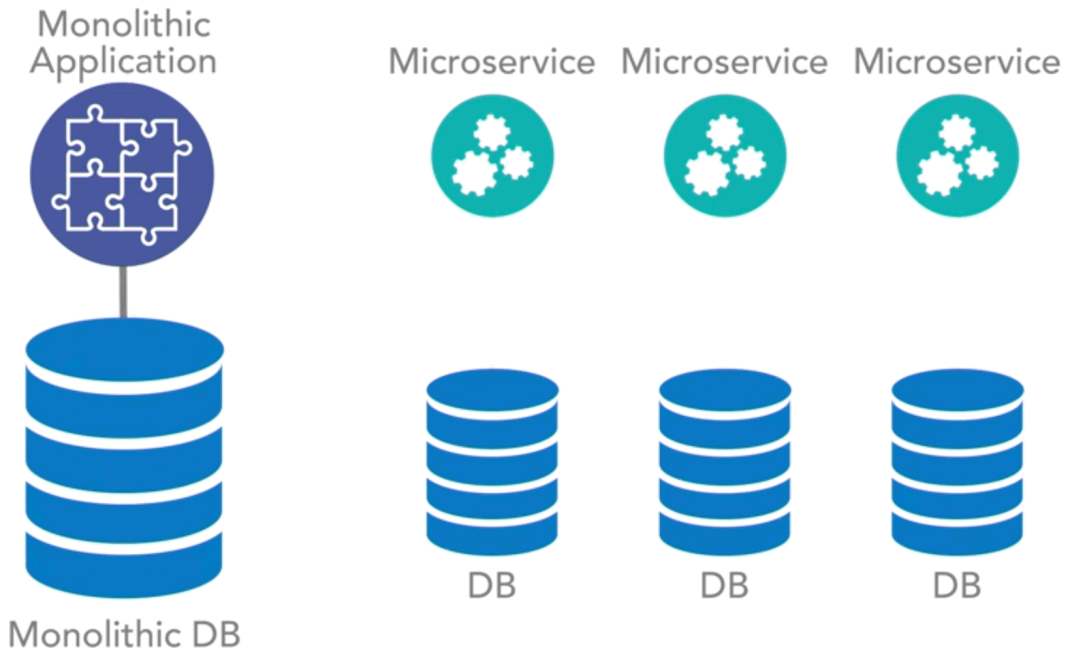
Bounded Context

- If every time user domain is called, it calls the customer domain → we put them in a single bounded context

- What if only 1% calls to customer domain from the user domain and at the same time the data in the user domain needs to be secured in a different manner → separate them

Data Domains as a Service Boundary

- Transactional Boundaries
  - Cannot eliminate transactions completely
  - No distributed transactions



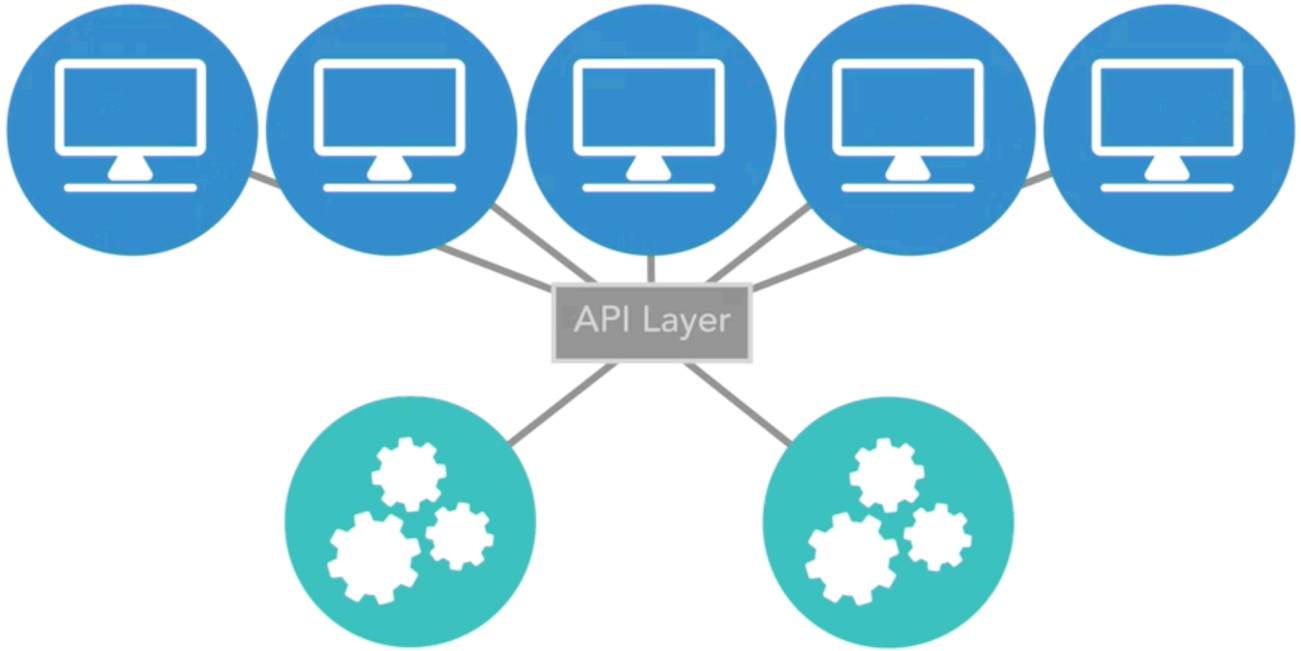
only BASE – eventual consistency

- In a microservice's architecture , you need to identify where you truly need ACID transactions, and wrap service boundary around these operations

- Example:
  - ACID: Bank Transfer Service - credit/debit
  - BASE: Applying for loan

API Layer

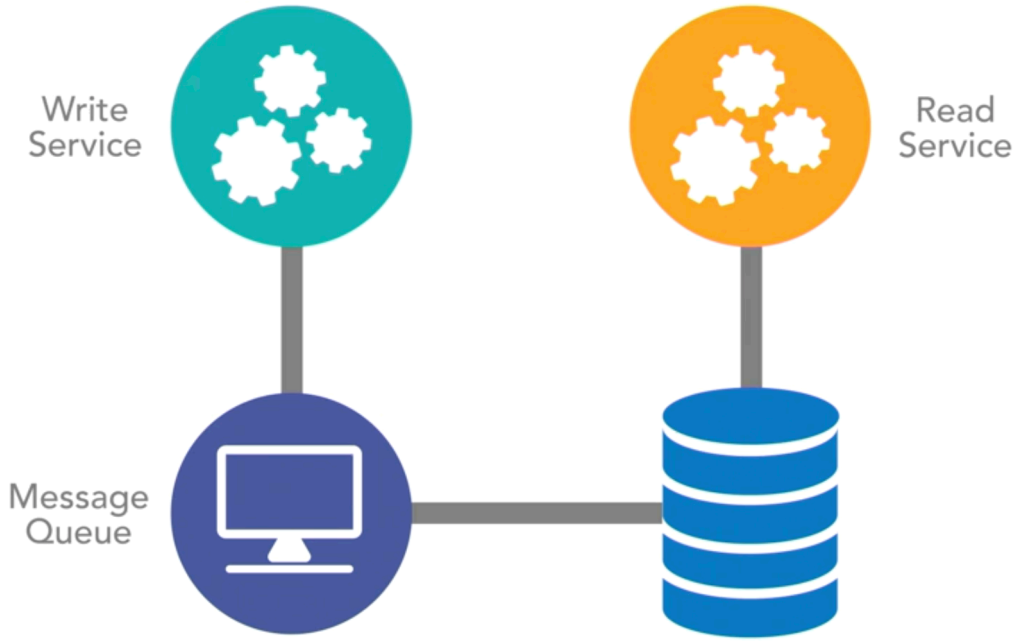
- Aggregated **proxy** of all your services' offerings
- Show the outside world / clients from knowing the structure

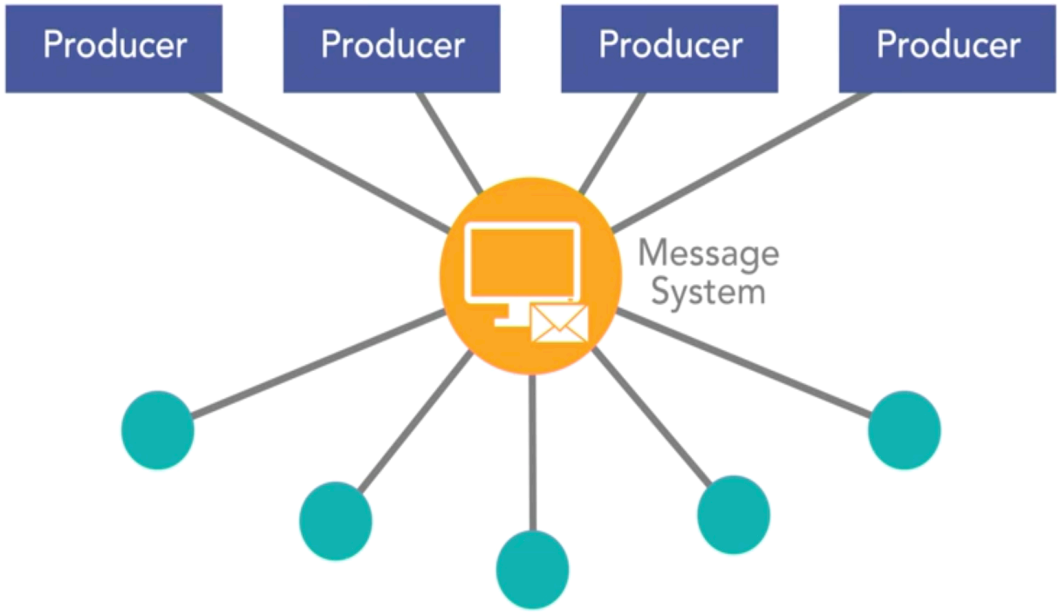


Easier for scaling

Asynchronous Communications

- Benefits: reduce latencies for the remote calls

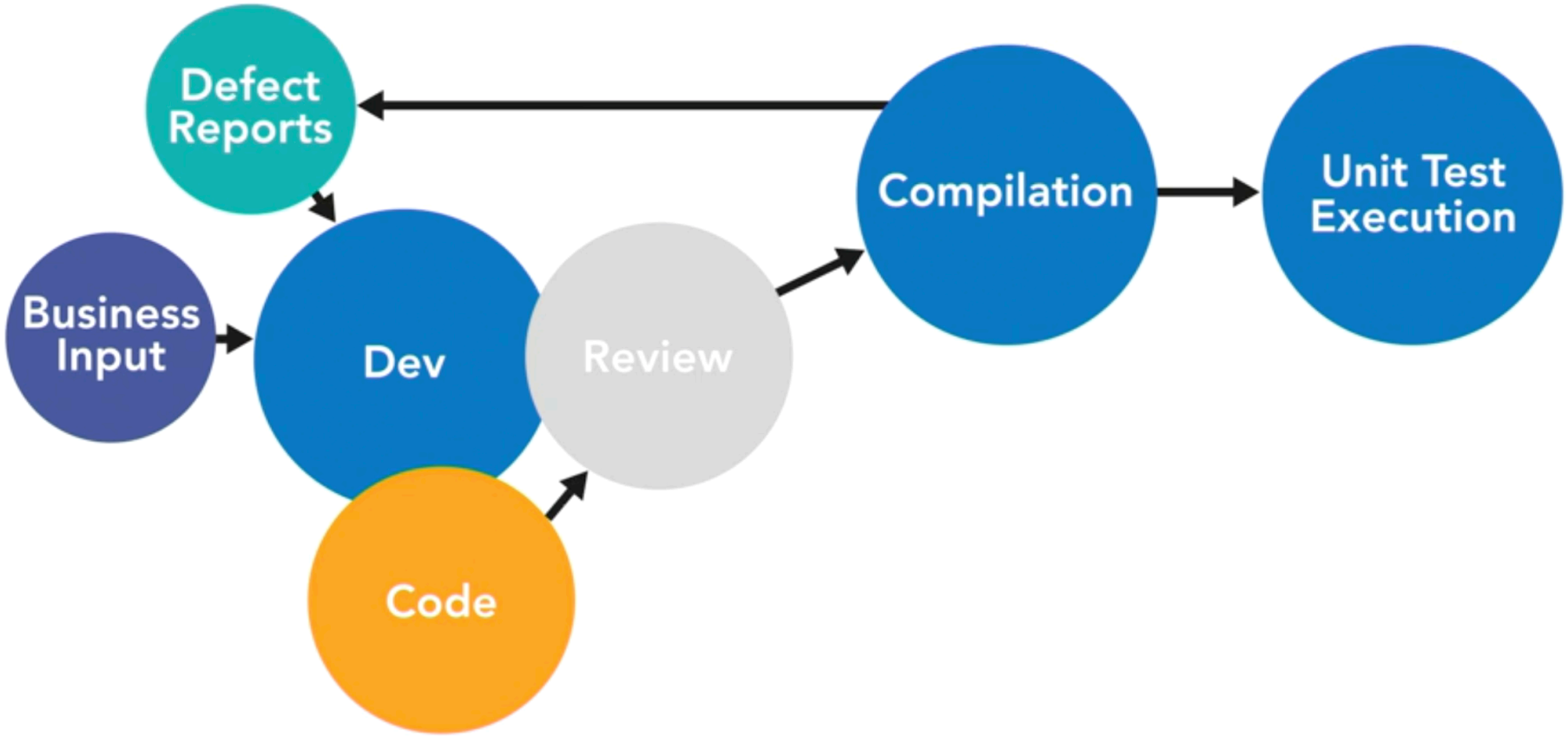




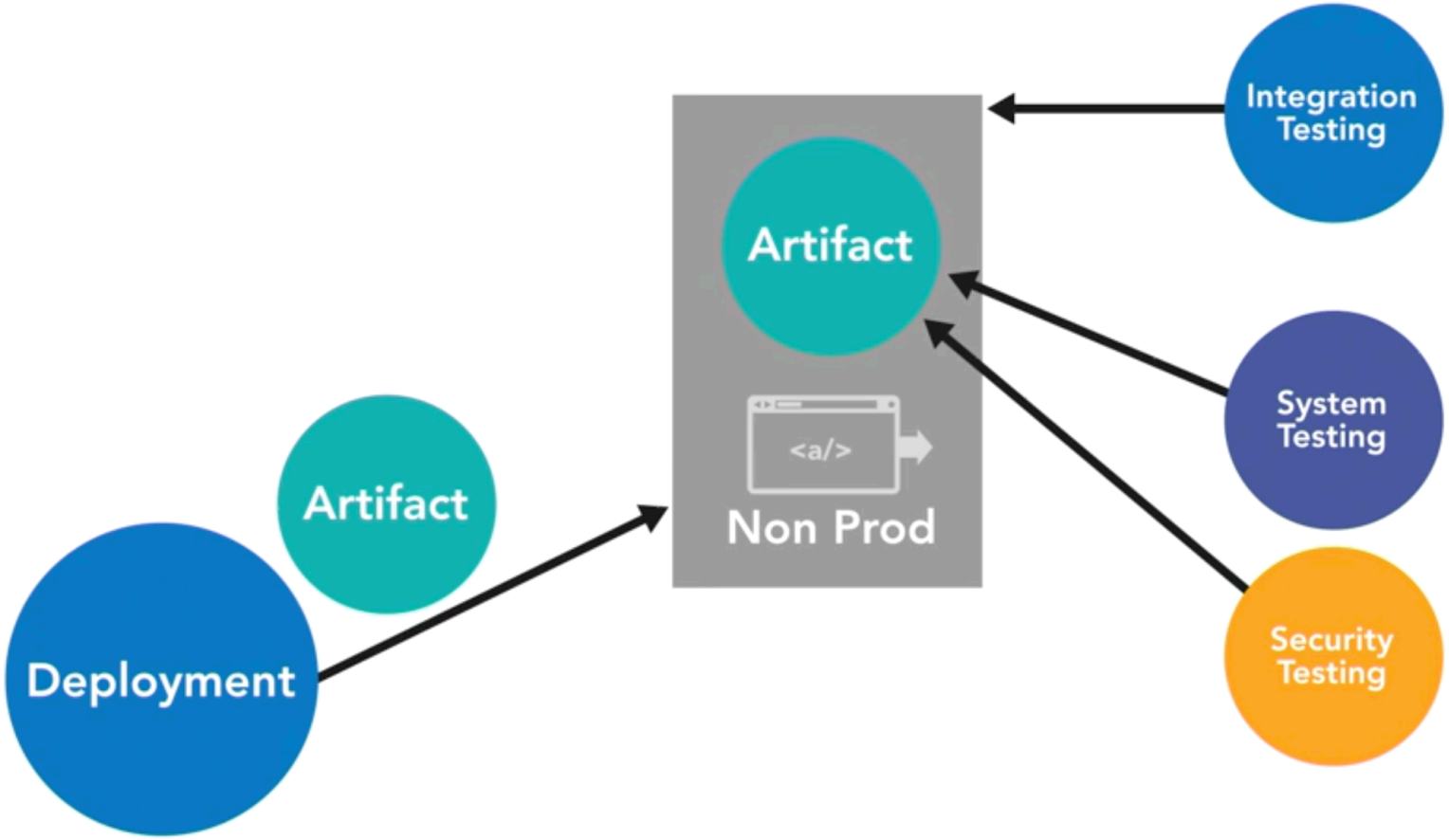
Logging and Tracing

- Unified logging
- Tracing: determine the actual flow

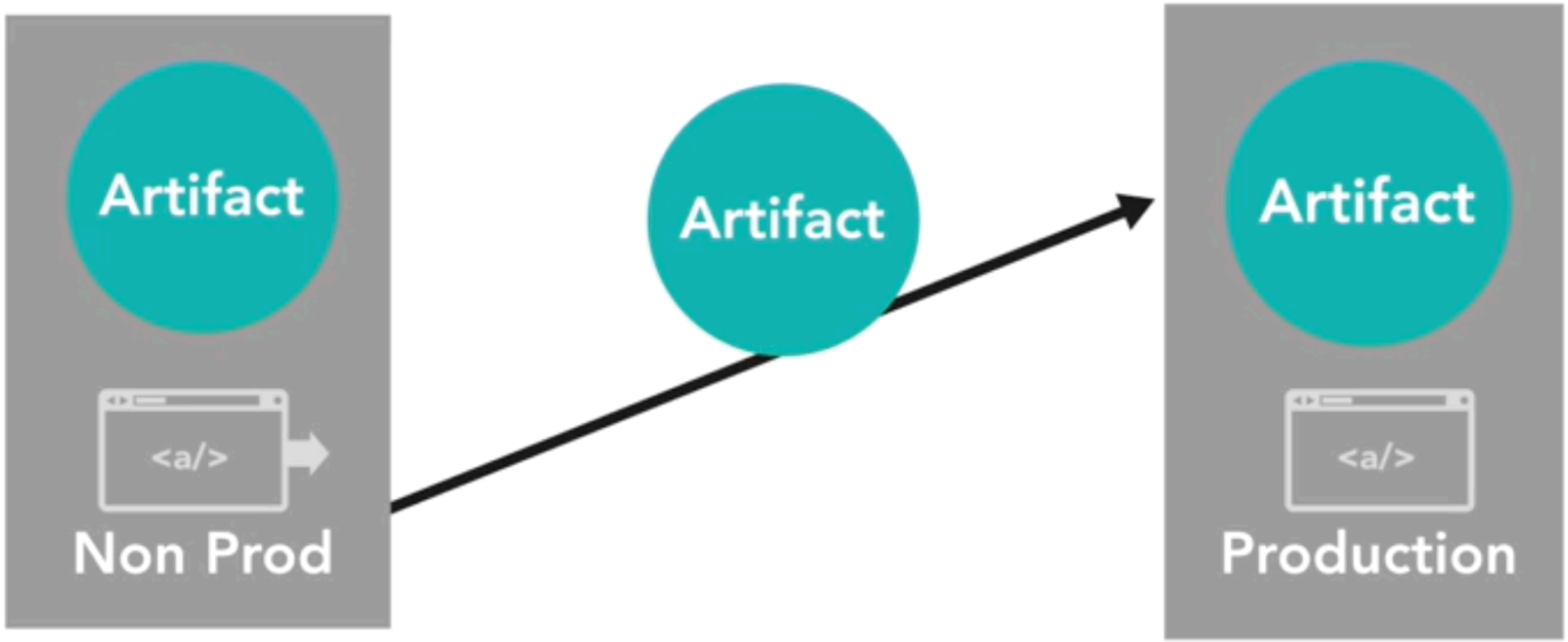
CI/CD – build step



Automated Deployment to the Non-production environment



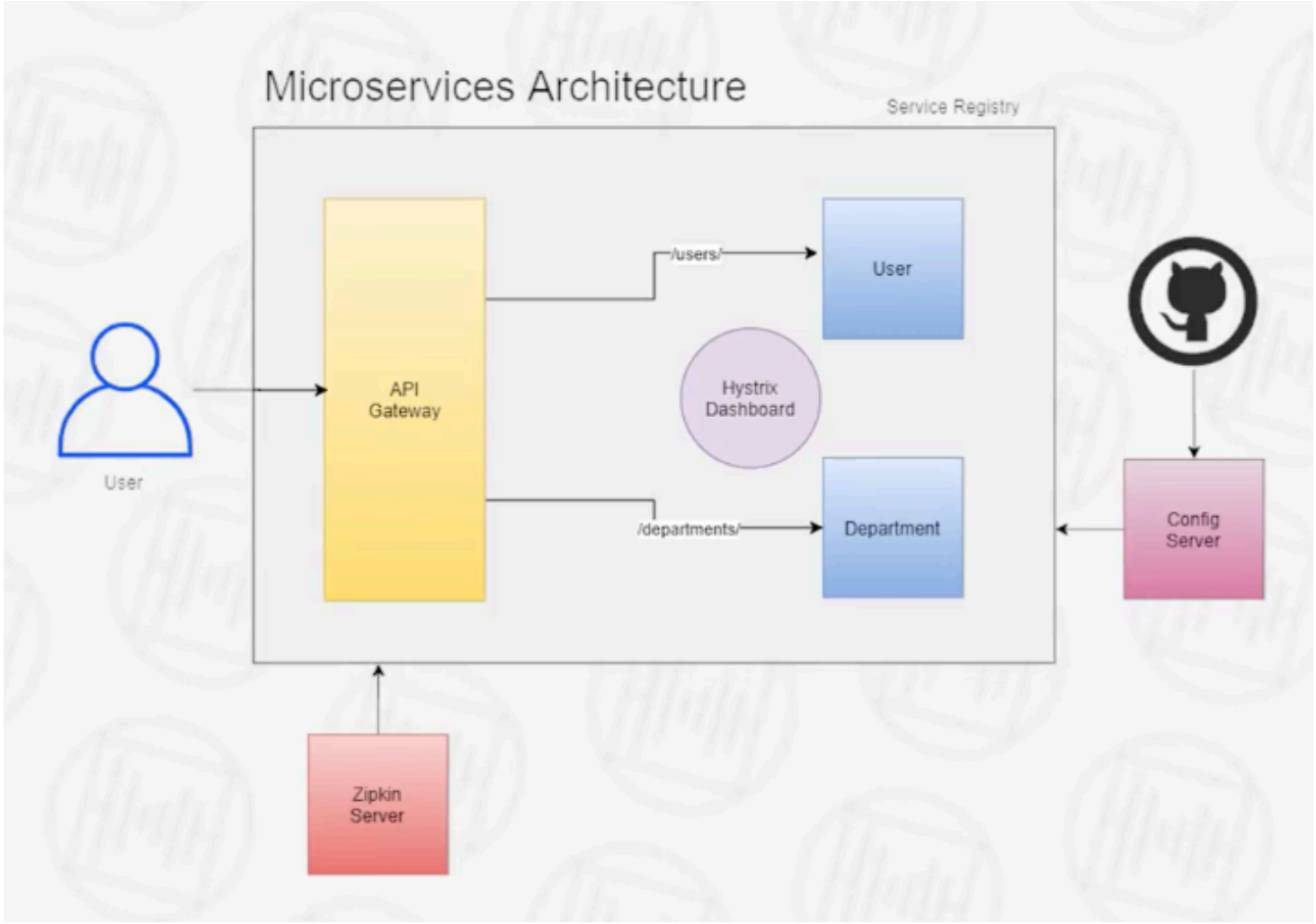
Deploy to Production



Design Considerations

- CI/CD pipelines
- Logging and tracing
- DDD: Service Boundaries
- Consider asynchronous calls first

Example: department service



Zipkin server: distributed logging

Reference

- https://www.linkedin.com/learning/microservices-foundations
- https://www.youtube.com/watch?v=BnknNTN8icw