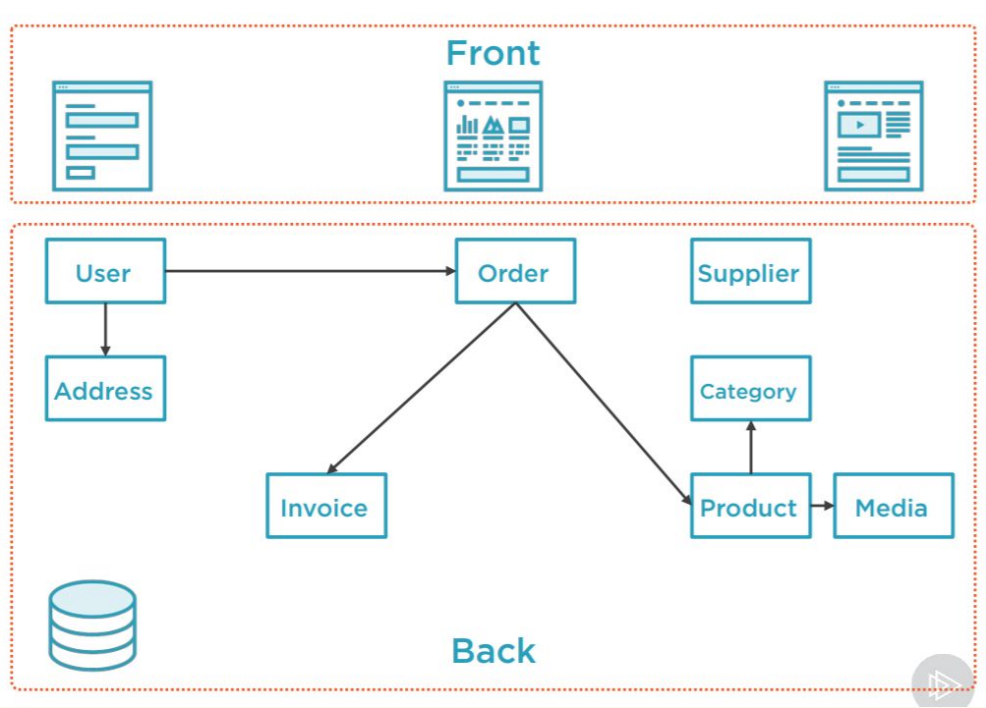
MicroServices

—

Notes

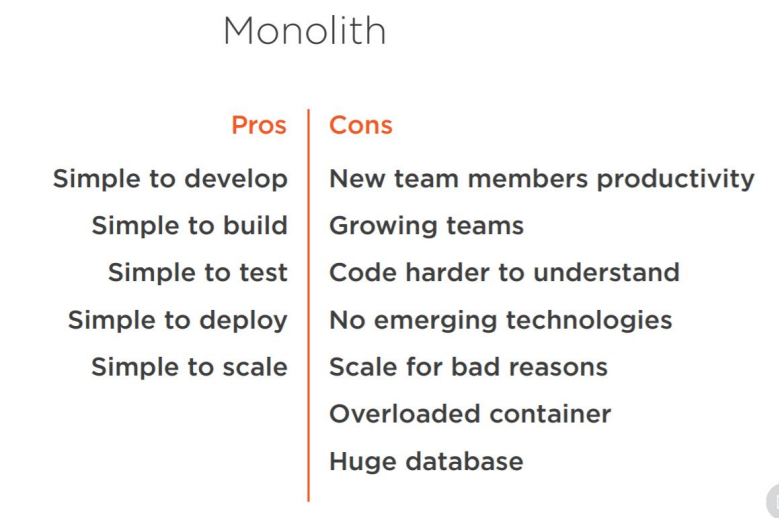


1. Single monolith
2. Single database
3. User interface
4. Expose APIs
5. Multiple instances

Domain Driven Design

Domain: E Commerce

Subdomain: User, Product, Order.



Micro Services?

*"Micro Services are **small, autonomous services** that **work together**"*

- Autonomously developed
- Independently deployable
- "Does one thing"
- Bounded Context
- Sub Domain
- Exchange messages
- Solve systems that are too big.
- Team more agile
- Faster to develop
- **Time to market**

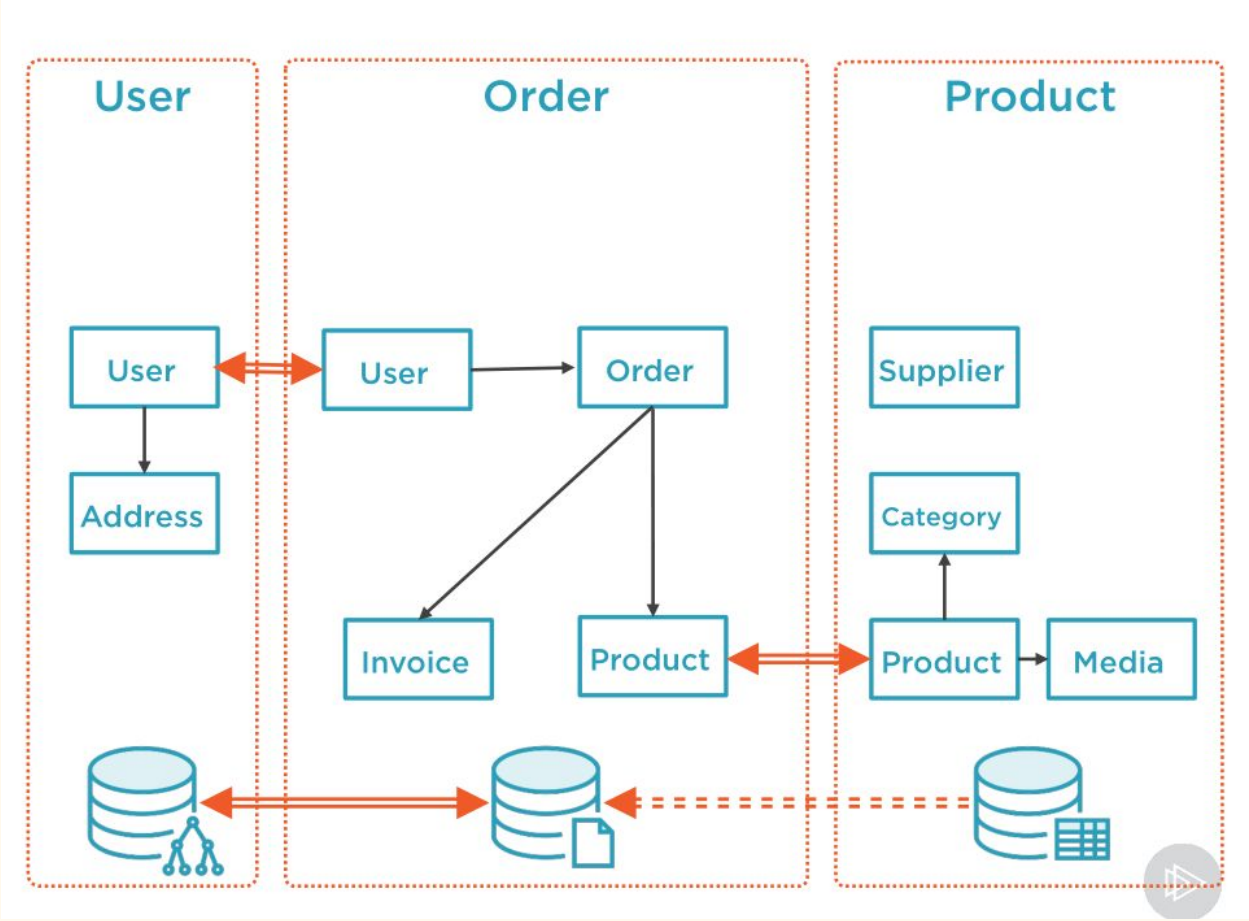
Microservices.

Codebase

কোড বেজ আলাদা আলাদা থাকে, এতে কোড মেইনটেইন করা সহজ হয়। ছোট টিম একটা মাত্র মাইরকো সার্ভিসে কাজ করে সেটা ডকুমেন্টেড করে রাখতে পারে।

User Interface

একটা সিঙ্গেল এপলিকেশনের মত এটাকে দেখাতে হবে। তাই সেটার মত করে UI ডেভলপ করতে হবে। এটার দুইটা টেকনিক আছে, হয় সার্ভার সাইডে মাইক্রোসার্ভিস ডেভলপার পেইজ বানায়ে ফ্রন্টে পাঠাবে। অথবা ফ্রন্টে UI টিম থাকবে যারা সব ডেটা নিয়ে একসাথে বানাবে। মাইক্রোসার্ভিসে UI নিয়ে চিন্তা না করলেও হবে।



Data Storage: Independent Different requirements Relational NoSQL

মাইক্রো সার্ভিসে ডেটা স্টোরেজ থাকে আলাদা আলাদা, এতে বিভিন্নয় রিকোয়ামেন্টে বিভিন্ন ধরনের ডেটাবেজ ইউজ করতে পারি আমরা। এর মাধ্যমে বিভিন্ন ধরনের ডেটাবেজের বিভিন্ন স্ট্রং ফিচারগুলা আমরা ইউটিলাইজ ক্রতে পারি। যেমন ইউজারের জন্য LDAP ইউজ করলাম আর অর্ডারের জন্য NoSQL আবার প্রডাক্টের জন্য রাখলাম SQL ডেটাবেজ।

কিন্তু তাহলে তারা কমিউনিকেট করবে কিভাবে?

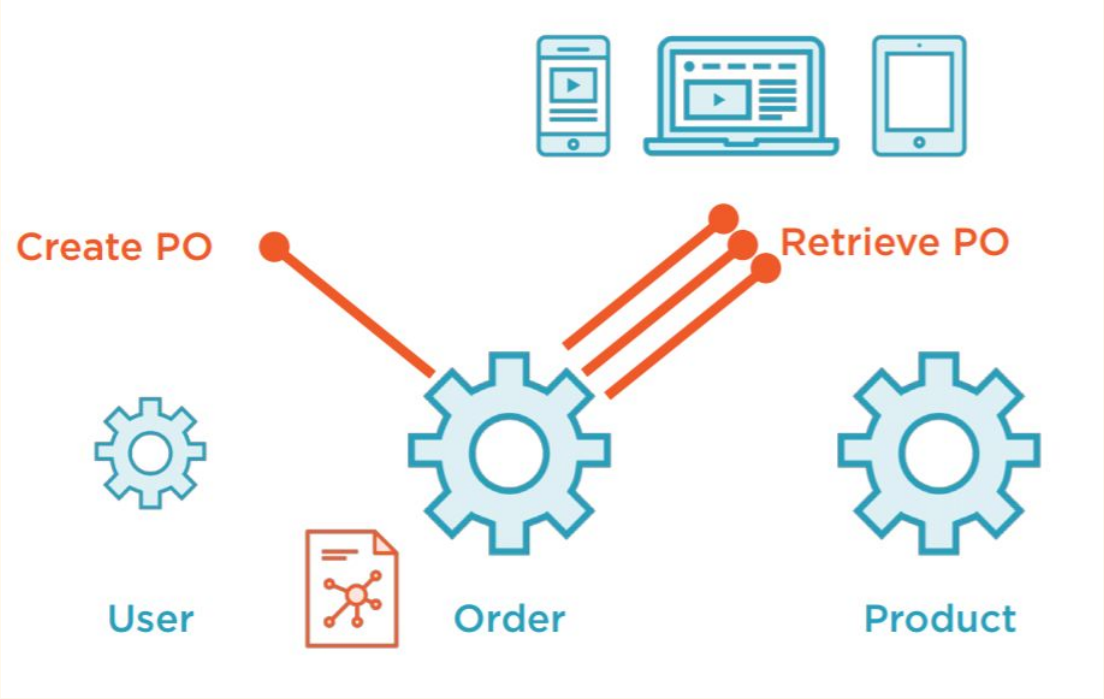
Data Synchronization

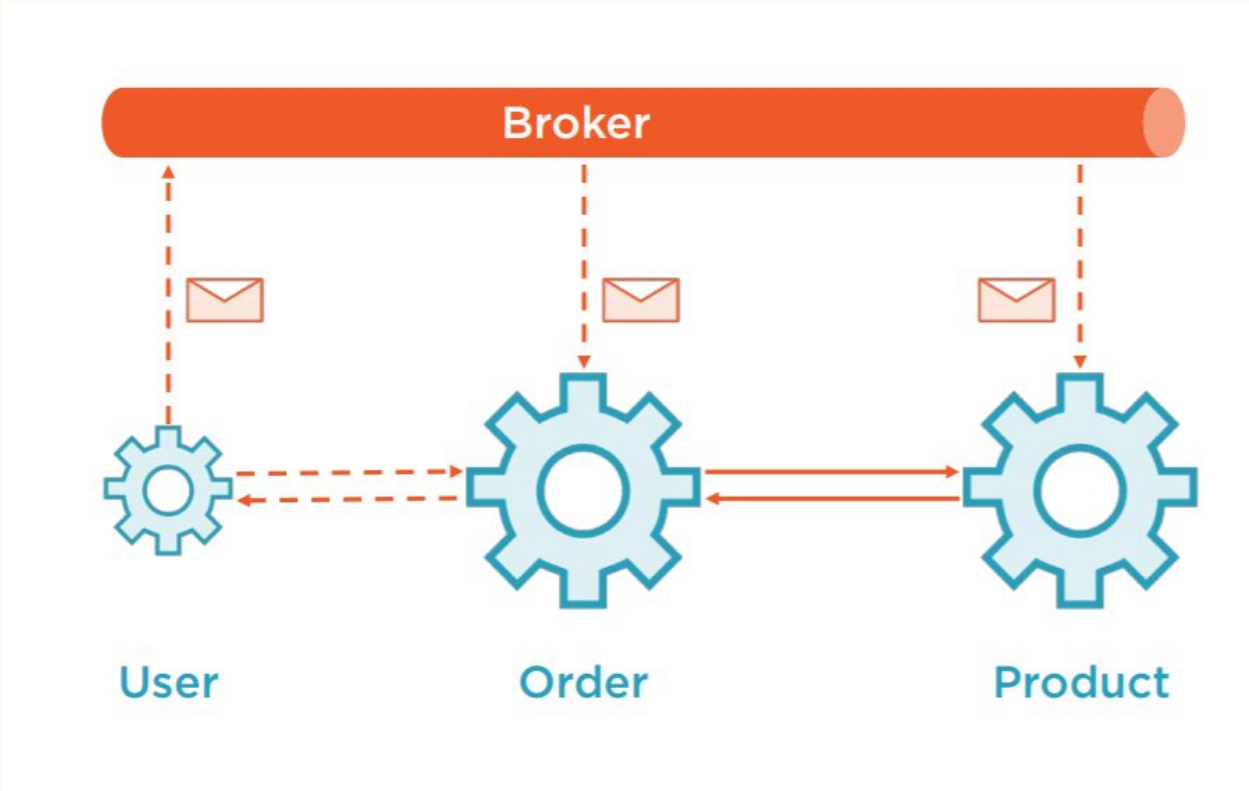
No distributed transaction মানে এক সাথে পুরা ডেটাবেজ আপডেট করে ফেলারে একটা চেষ্টা। সেটা করা যাবে না। এতে স্লো হয়ে যাবে। কিন্তু **Immediately consistent** লাগবে তার জন্য Eventual consistency ইমপ্লিমেন্ট করতে হয় যেটা করা হয় ***Capture data change আর Event sourcing*** এর মাধ্যমে যার জন্য লাইব্রেরী লাগে **Akka, Kafka, Rabbit MQ Debezium**

Services Communication:

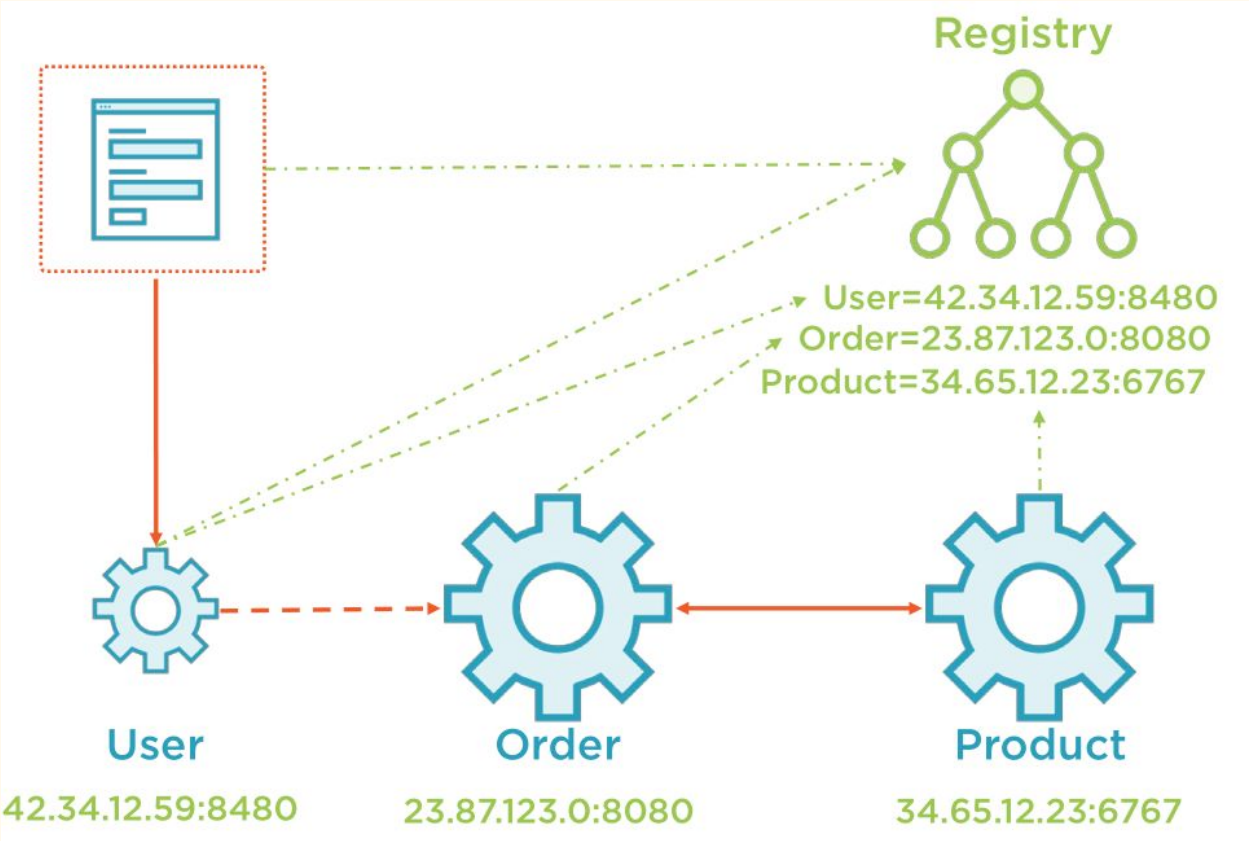
1. **RPC**
2. **Messaging**

- **Remote Procedure Invocation (RPC):** পরস্পরের সাথে **Request/Reply** মেথডে কাজ করে যেমন **REST, SOAP, gRPC** প্রটোকলে।
    - **APIs and Contracts**





- **Messaging/Events: Broker or Channel** এর মাধ্যমে মাইক্রো সার্ভিস ম্যাসেজিং করতে পারে। এখানে চ্যানেলে সার্ভিসগুলা **Subscribe** থাকে, যদি কোন ম্যাসেজ কেউ চ্যানেলে **Publish** দেয় তাহলে অন্যান্য মাইক্রো সার্ভিস সেটা রিসিভ করে, প্রয়োজনে রিপ্লাই দেয়। এটা **Loosely** couple কারন ম্যাসেজ কিউ তে থাকে। **Kafka, RabbitMQ**
    - **Protocol Format Exchange**
    - Text XML, JSON, YAML Human readable Easy implement Binary gRPC More compact

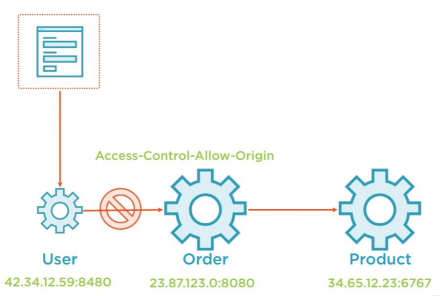


Distributed Services যেহেতু মাল্টিপল ডিভাইসে এই সার্ভিস গুলো থাকে, তাই তাদের ম্যানেজ করাও একটা বিষয়।

Service Registry/Discovery Server কে কোথায় আছে, সেটা একেক সময় একেক IP তে থাকতে পারে, তার জন্য Service Registry মেইনটেইন করতে হয়। Locations change এর Phone book বলা যায় যা Self registration Discovery Invocation এর মাধ্যমে নিজেকে রেজিস্ট্রিতে এন্ট্রি করায় দেয়। এটা মেইন টেইন করার কিছু লাইব্রেরী Eureka, Zookeeper, Consul

Cross-origin Resource Sharing (CORS)

HTTP এর একটা সমস্যা হচ্ছে Same-origin policy তে কাজ করে। মানে নিজের সার্ভারে ফাইল সেয়ারিং এ সমস্যা নাই। কিন্তু অন্য সার্ভারের সাথে কমিউনিকেট করতে গেলে অনেক কাহিনি করা লাগে এই Same protocol, server, port Restrict cross-origin এর কারনে।



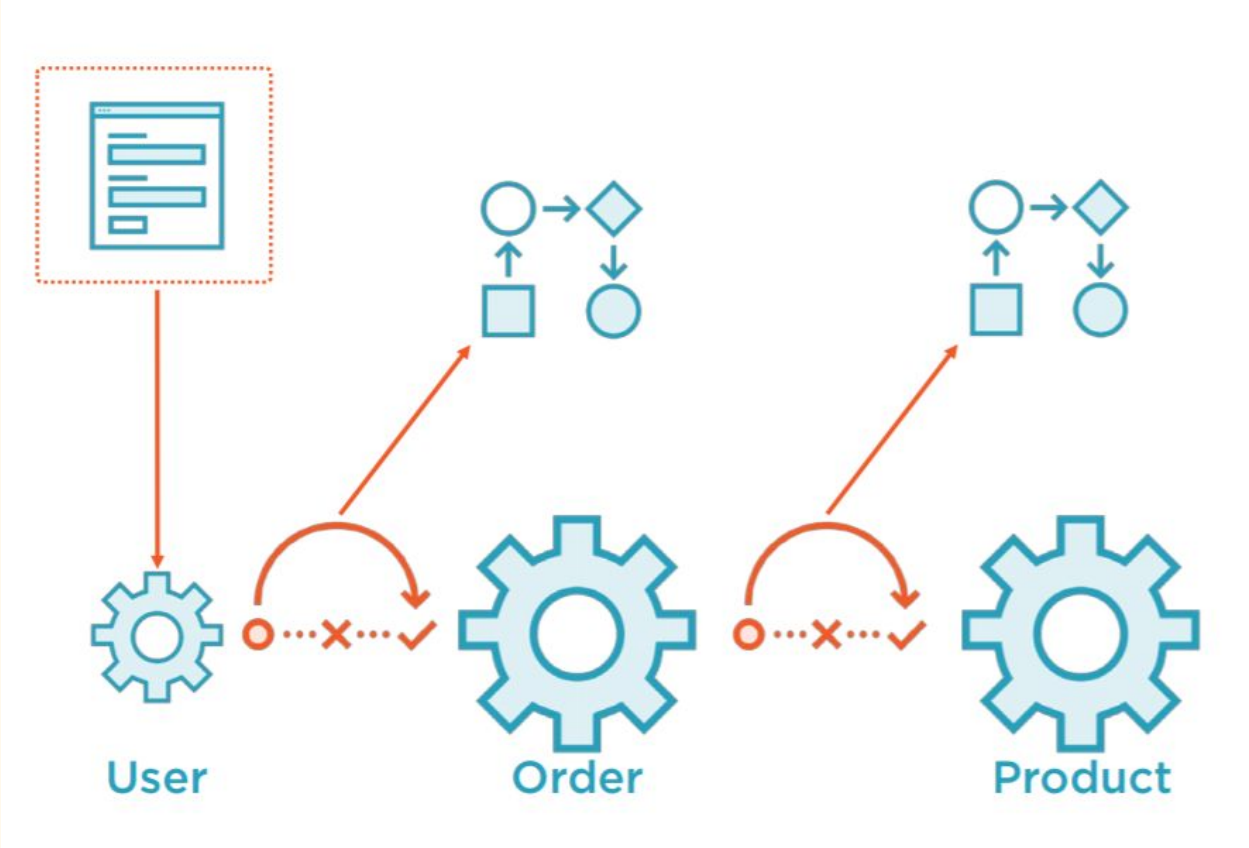
এর কারনে আমরা HTTP headers ইউজ করি। হেডারে বলে রাখি ভাই এইটা দে, তখন দেয়।

Circuit Breaker

১০ টা সার্ভিসের ৯টায় রিকোয়েস্ট দিয়ে ১০নাম্বারটাতে ফেইল করলে কিন্তু ঝামেলা হয়ে যাবে। সব রিকোয়েস্টই ফেইল্ড দেখাবে। এখানে সার্কিট ব্রকার লাগবে।

এটাকে ডমিনো ইফেক্ট বলে। এখানে একটা সার্কিট ব্রেকার ইন্ট্রোডিউস করা লাগে। যেটা বুঝবে যে কল ঠিক মত এন্সার হচ্ছে না। তখন সে কল ডাইভার্ট করে প্রক্সি সার্ভারে দিয়ে দিবে। অথবা আস্তে আস্তে রিকোয়েস্ট পাঠাবে, রিকোয়েস্ট ম্যানেজ করবে ঝামেলা হলে। অর্থাৎ রিকোয়েস্টটা ফেইলল্ড যাতে না হয় সে ব্যবস্থা নিবে।

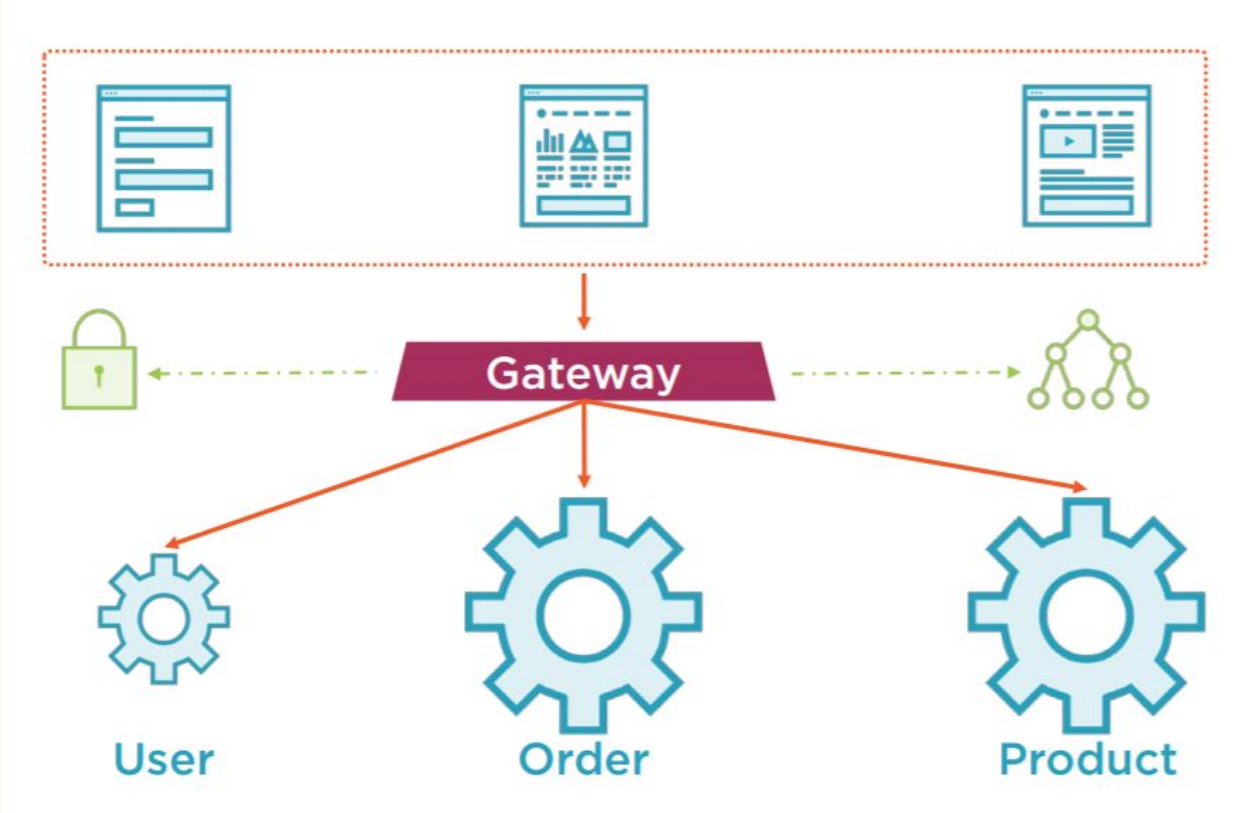
Services available Network failure Heavy load **Domino effect Invoke via proxy Deviate calls Reintroduce traffic Hystrix, JRugged**



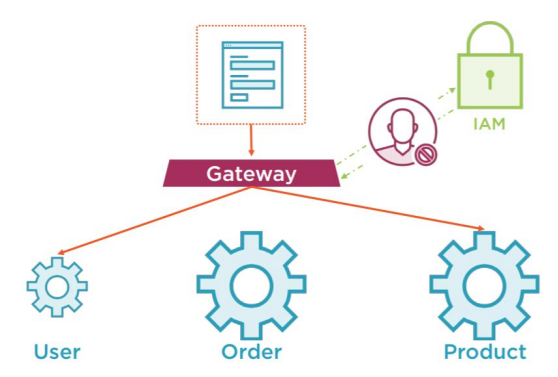
Gateway

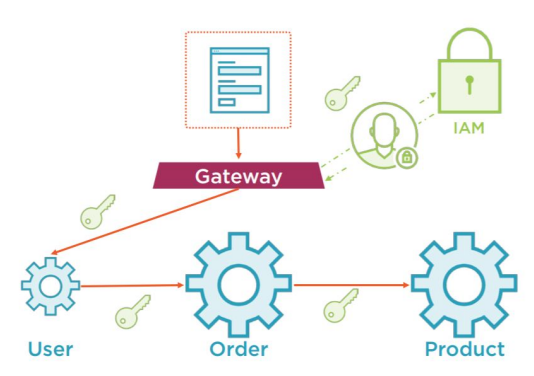
UI এর জন্য Single entry point লাগে যা Unified interface বানাবে, বা Cross-cutting concerns লাগে যেমন সিকিউরিটি। কোন কোন রিকোয়েস্ট ডিরেক্ট চলে যাবে, কোন কোন রিকোয়েস্ট সিকিউরিটির মধ্য দিয়ে যাবে। আবার বিভিন্ন ডিভাইসের জন্য বিভিন্ন API এখানে দিয়ে রাখতে পারি। Single entry point

Unified interface-Cross-cutting concerns-API translation-Security: Zuul, Netty, Finagle

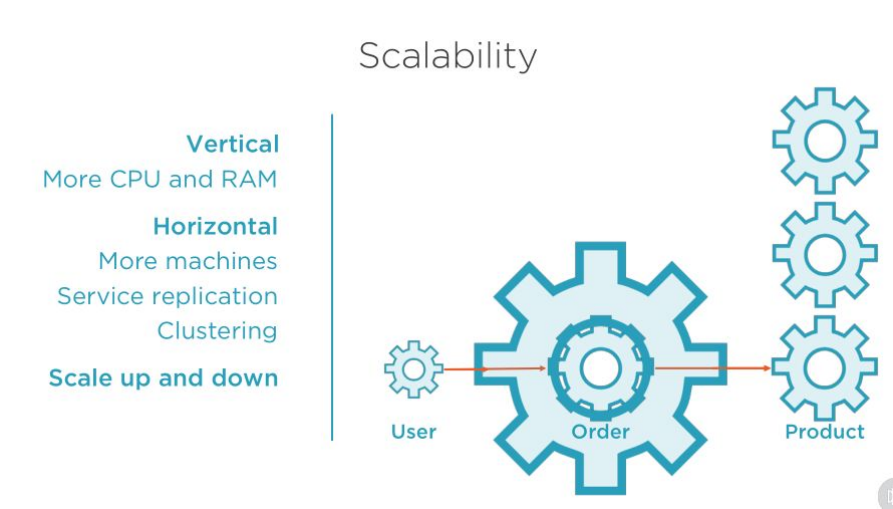


Single Sign-on Kerberos, OpenID, OAuth 2.0, SAML Okta, Keycloak, Shiro Access





TokenStores information about user Exchanged between services JSON Web Token Cookie



Scalability and Availability

Scalability:

- **Vertical**

  More CPU and RAM

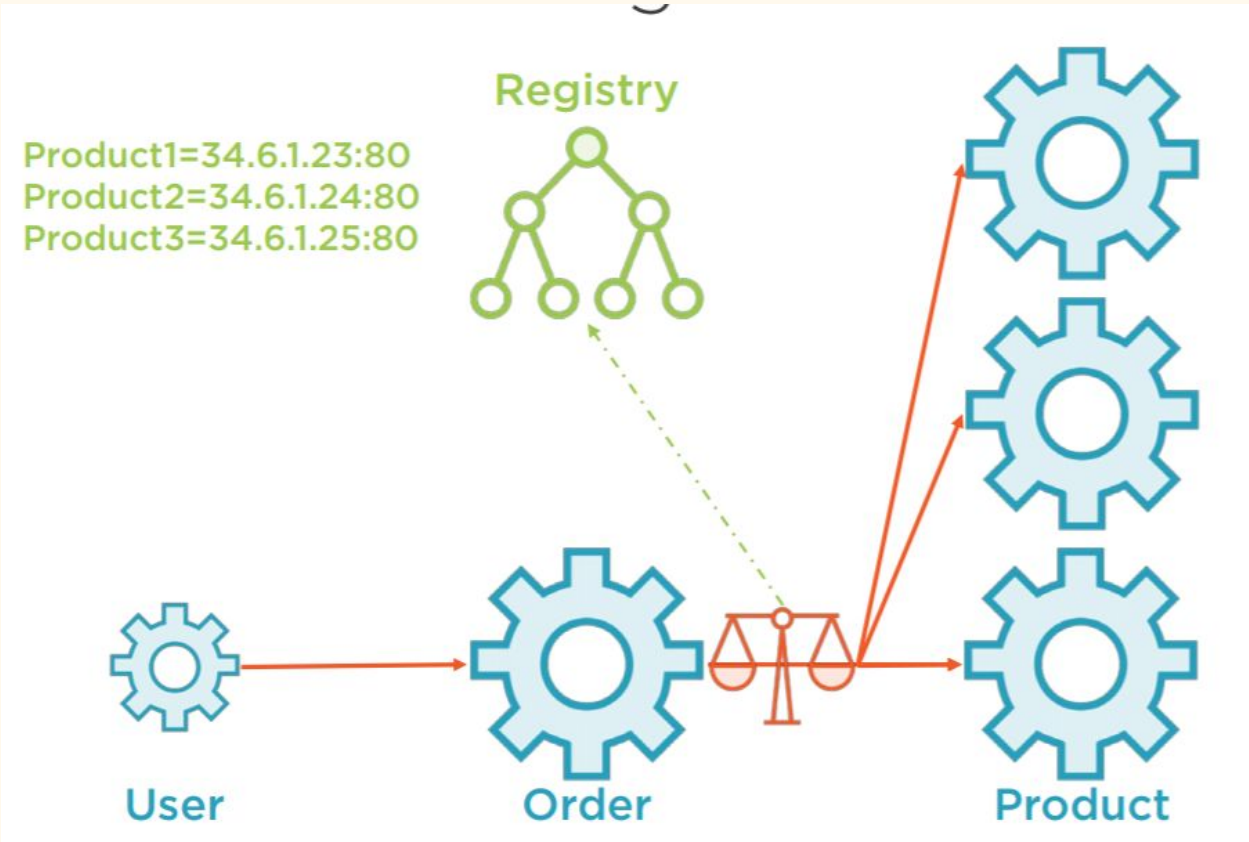
- **Horizontal**
  More machines
  Service replication
  Clustering
  Scale up and down

Availability: Down time in a year.

Client Load Balancing: Which instance to choose Registry Round-robin, weight, capacity Ribbon, Meraki

Single Point of Failure কপি করে রাখতে হবে। হরাইজন্টাল স্কেলিং আরকি।

যেমনঃ Gateway Broker Registry IAM



Monitoring

Monitoring and Dashboard: Kibana, Grafana, Splunk

Health Check: Service running ? Incapable handling requests ?

Log Aggregation: to Understand behavior Write logs for each service and then Aggregate logs using: LogStash, Splunk, PaperTrail

Exception Tracking: Error, Exception, Investigation etc

Metrics: System slowing down Performance issues Gather statistics Aggregate metrics DropWizard, Actuator, Prometheus

Auditing: Behavior of users Login Logout Visited pages Browsed products Record user activity

Rate Limiting: Third-party access Control API usage Defend DoS attacks Limit traffic In a period of time Monetize our APIs

Alerting: Tons of information How to be proactive Fix error when occurs Configure threshold Trigger alerts

Distributed Tracing: Requests span services Logs Trace entire request Correlation id Chain of calls Dapper, HTrace, Zipkin

Deployment

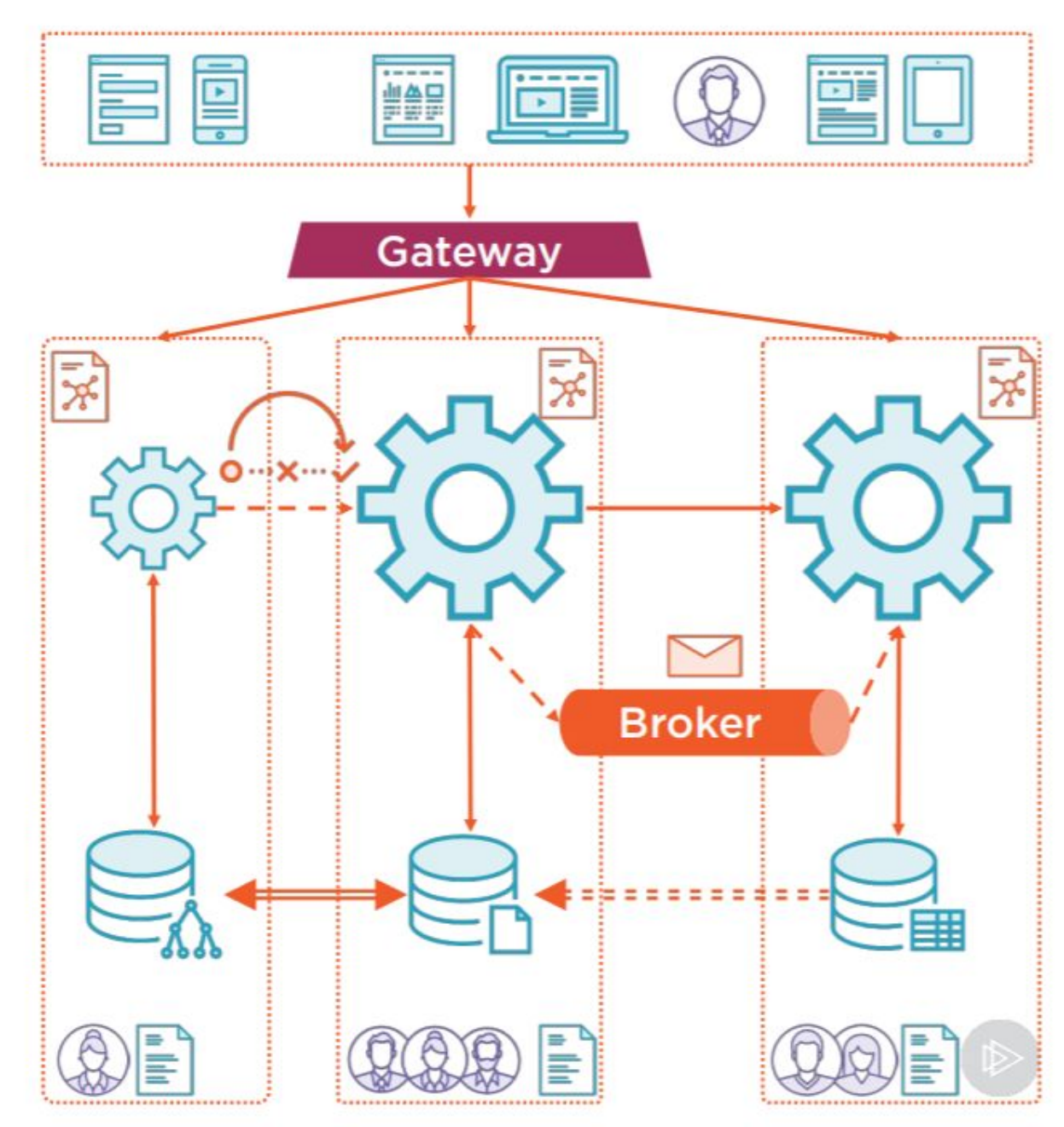
Containers - Packaging microservice With dependencies Container image Easy to move from environment Scale up and down Docker, rkt

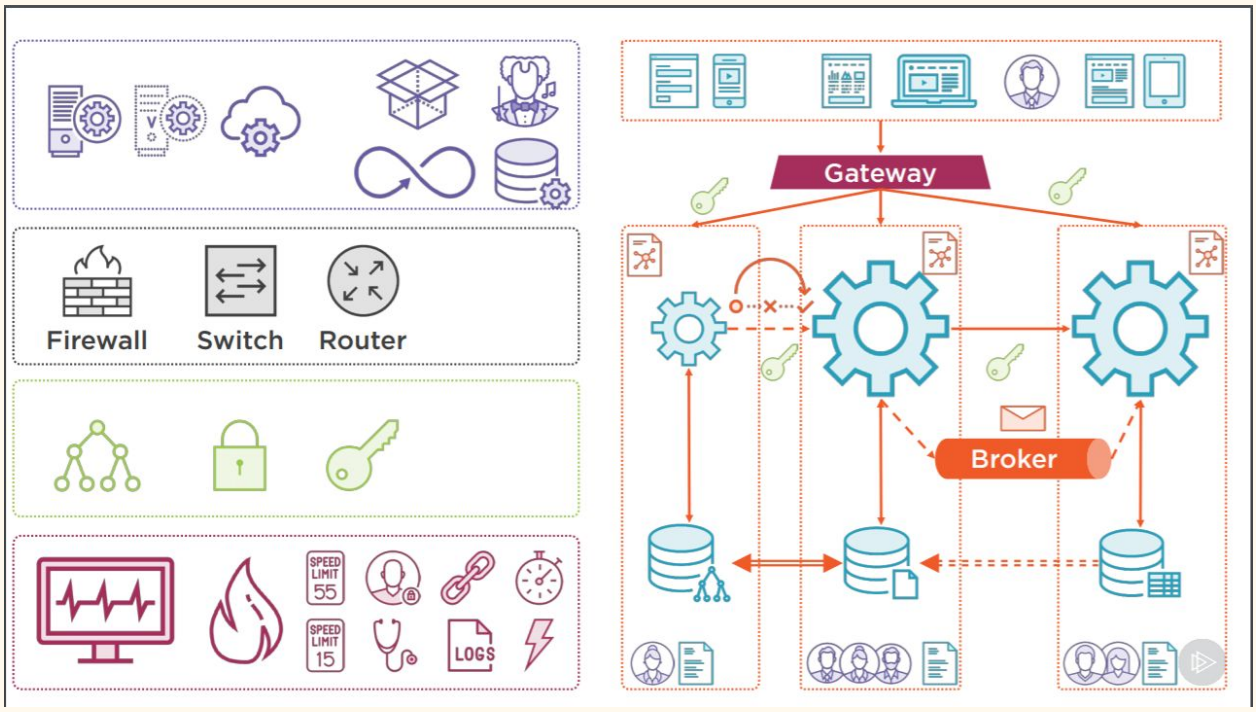
Orchestrator - Multiple containers Multiple machines Start at the right time Failed containers **Kubernetes, Mesos, Docker Swarm**

Continuous Delivery - Automate deployment Cost-effective Quick Reliable Build, test, deploy **Jenkins, Asgard, Aminator**

Environments - Production Dev, test, QA, staging Integration Versionning

External Configuration - Different environments Different configuration **Activate functionality Externalize configuration Archaius, Consul, Decider**


Gateway
Broker



Spring Cloud Components

- **Configuration.**
- **Service Discovery.**
- **Circuit Breakers.**
- **Routing and Messaging.**
- **API Gateway.**
- **Tracing.**
- **CI Pipeline and Testing.**