Micro Services

A good microservice designer understands the need for adaptability and endeavors to continually improve the system instead of working to simply produce a solution.

Microservices architectures are not a completely new approach to software engineering, but rather a combination of various successful and proven concepts

Challenges with monolithic architecture

- Developer perspective
- Tester perspective
- Business owner perspective
- Service management perspective
- Sample REST micro services can be found on https://reqres.in/
- https://gorest.co.in/
- https://dummy.restapiexample.com/

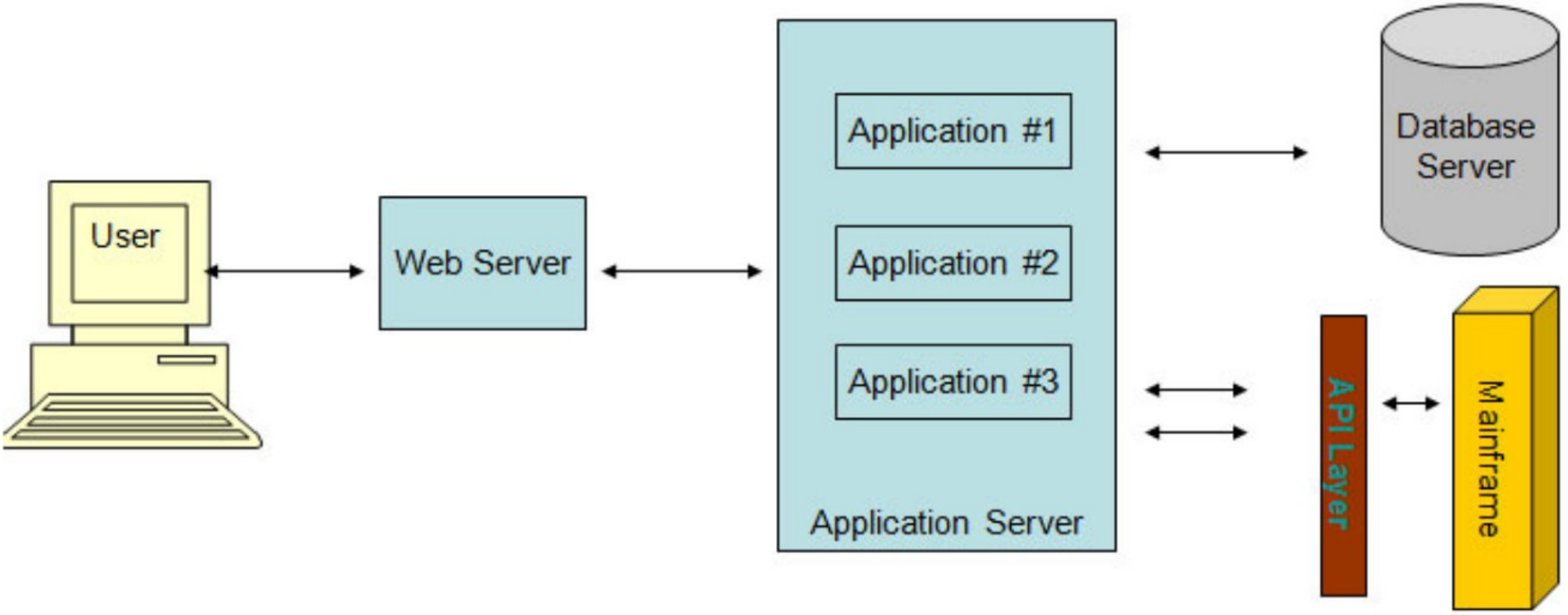
Why Not Monolithic Application

- User interface (UI)layer
- Business layer
- Persistence layer.
- A central idea of a micro services architecture is to split functionalities into cohesive “verticals”—not by technological layers, but by implementing a specific domain.
- WSDL for SOAP service https://www.soapui.org/docs/soap-and-wsdl/working-with-wsdls/
- https://www.learnwebservices.com/
- https://www.crcind.com/csp/samples/SOAP.Demo.cls

What is Monolithic Application?

- A monolithic application is an application where all of the logic runs in a single app server.
- Typical monolithic applications are large and built by multiple teams, requiring careful orchestration of deployment for every change.
- We also consider applications monolithic if, while there are multiple API services providing the business logic, the entire presentation layer is a single large web app.
- In both cases, microservice architecture can provide an alternative.

Monolithic Application



Monolithic vs Microservice Application

Category	Monolithic architecture	Microservices architecture
Code	A single code base for the entire application.	Multiple code bases. Each microservice has its own code base.
Understand-ability	Often confusing and hard to maintain.	Much better readability and much easier to maintain.
Deployment	Complex deployments with maintenance windows and scheduled downtimes.	Simple deployment as each microservice can be deployed individually, with minimal if not zero downtime.
Language	Typically entirely developed in one programming language.	Each microservice can be developed in a different programming language.
Scaling	Requires you to scale the entire application even though bottlenecks are localized.	Enables you to scale bottle-necked services without scaling the entire application

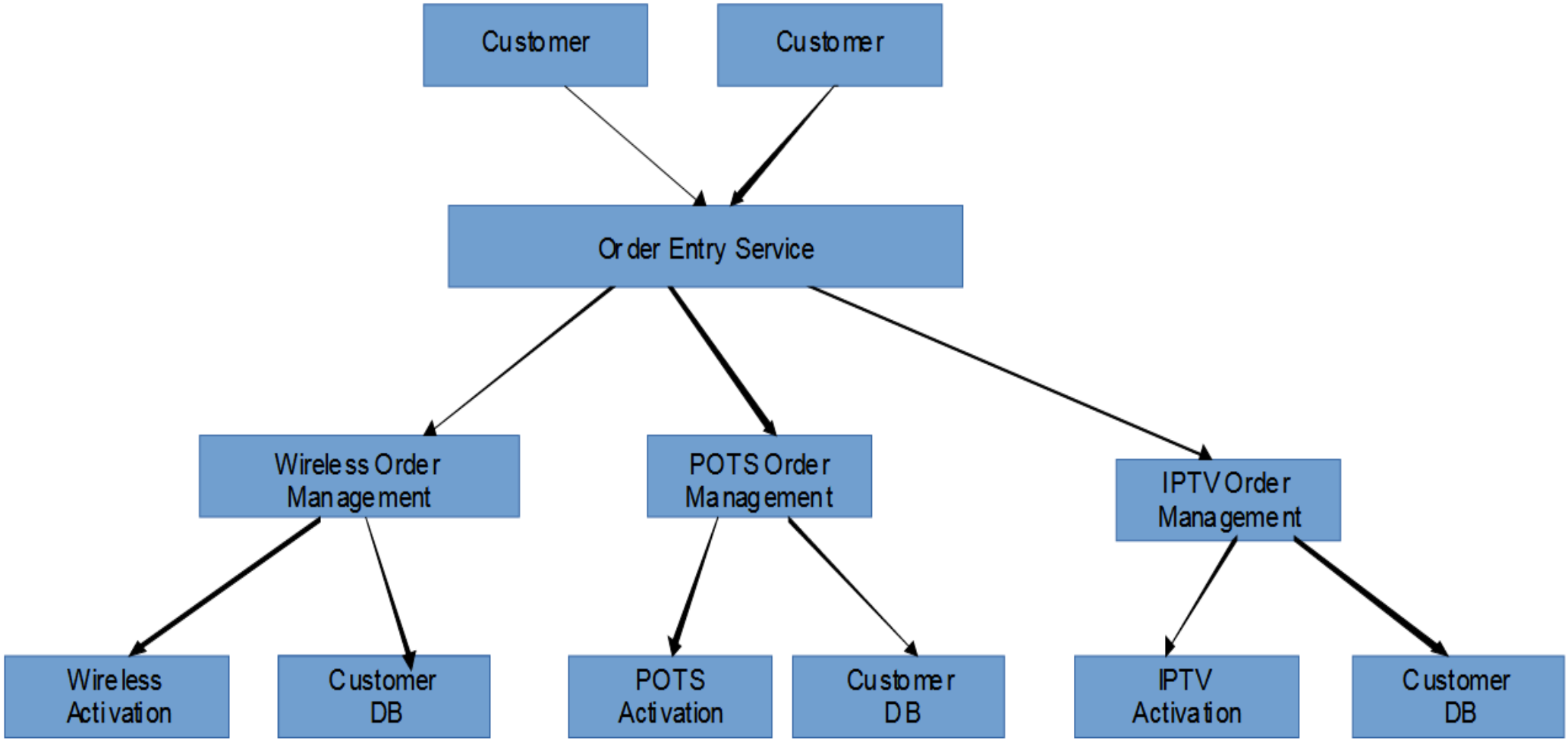
What to avoid with microservices

- Don’t start with microservices
- Don’t even think about microservices without DevOps
- Don’t manage your own infrastructure
- Don’t create too many microservices
- Don’t forget to keep an eye on the potential latency issue

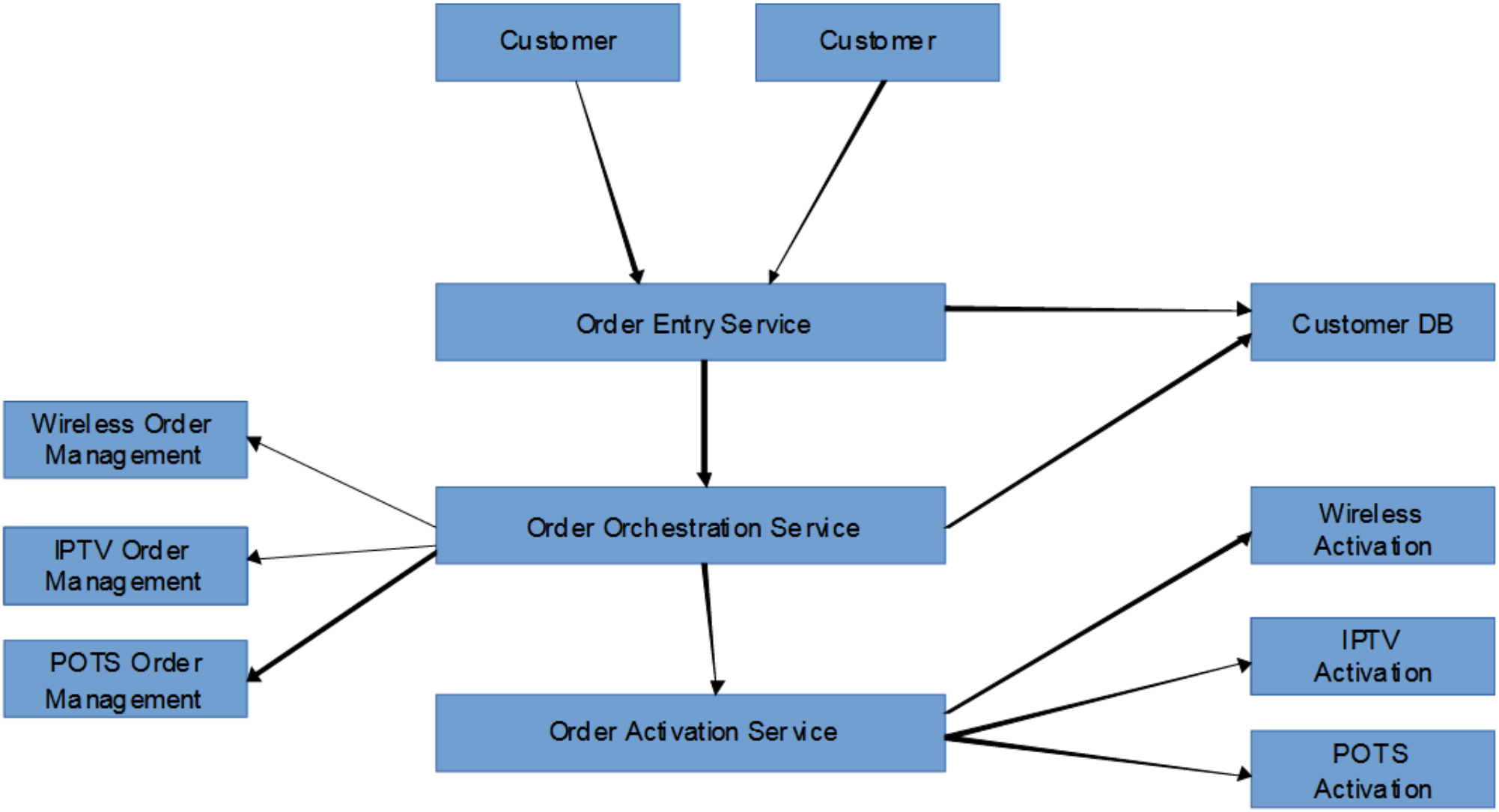
How is this different than service-oriented architecture?

- A set of services and business-aligned functionality that a business wants to provide to their customers, partners, or other areas of an organization.
- An architectural style that requires a service provider, a service requester with a service description and possibly mediation
- A set of architectural principles, patterns, and criteria that address characteristics, such as modularity, encapsulation, loose coupling, separation of concerns, reuse, and composability.
- A programming modelcomplete with standards, tools, and technologies that supports web services, REST services, or other kinds of services.
- A middleware solution optimized for service assembly, orchestration, monitoring, and management.

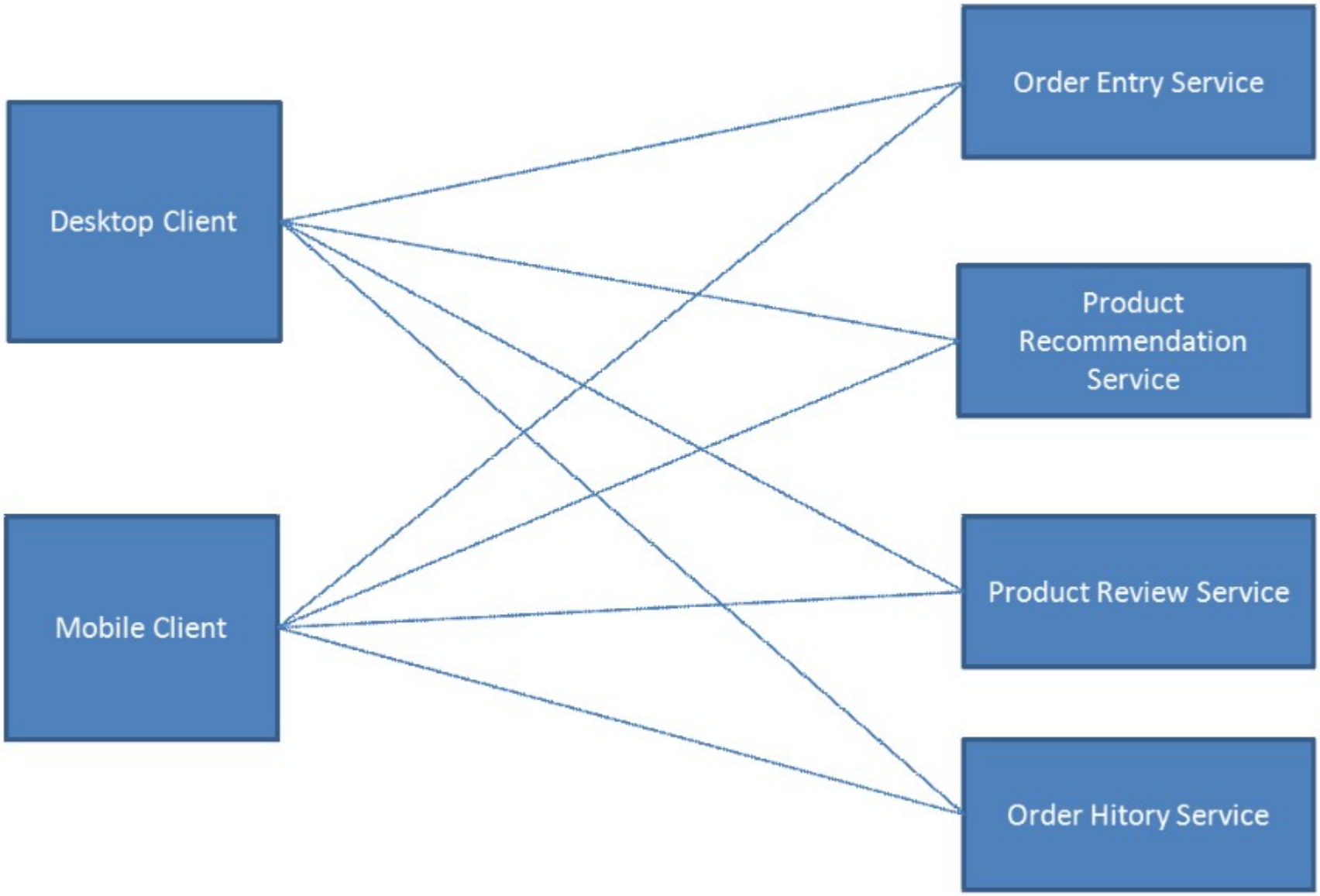
“Old” Style of Service Design



Microservice Architecture Design



Sample Microservice Application



Introduction to Micro Services

- Microservices architecture is a distributed design approach intended to overcome the limitations of traditional monolithic architectures.
- Microservices help to scale applications and organizations while improving cycle times.
- Micro services comes with a couple of challenges that might add additional architectural complexity and operational burden.
- Microservices architectures and minimize architectural and operational complexity.
- How to implement typical patterns,such as service discovery or event sourcing,natively with AWS service

Introduction to Micro Services

- Microservices represent a design pattern in which each microservice is just one small piece of a bigger overall system.
- Each microsystem performs a specific and limited scope task that contributes to the end result. Each task could be as simple as “calculate the standard deviation of the input data set” or “count the number of words in the text.”
- The key behind building microservices is planning the system to identify the distinct subtask, then writing applications that address each subtask.
- As each microservice needs to deliver output data to the next microservice, a microservices architecture often uses a lightweight messaging system for that data handoff.

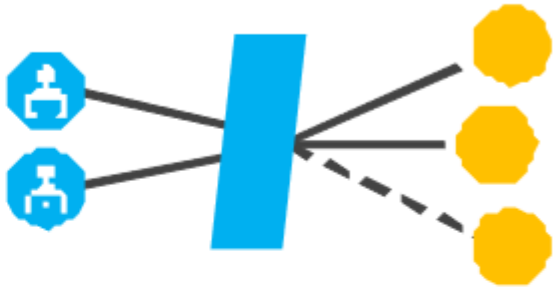
What is Microservice...

Use Cases

Worker Offload

Intensive work offloaded and
distributed amongst worker processes
to be performed asynchronously

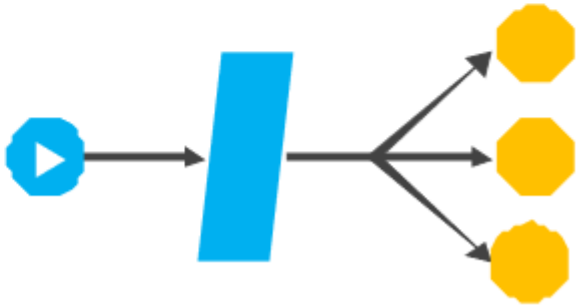
- Processing images or videos
- Performing text analytics



Event Driven

Take one or more actions
when something interesting happens

- E-mail logs and update dashboards when build finishes
- Upload videos once finished transcoding



What is Microservice

- Microservices is an architecture style, in which large complex software applications are composed of one or more services. Microservice can be deployed independently of one another and are loosely coupled.
- Each of these microservices focuses on completing one task only and does that one task really well.
- In all cases, that one task represents a small business capability.

Java Micro service Frameworks

Spring Boot. This is a popular framework for building Java applications, especially microservices, because it simplifies much of the setup and configuration process for getting your applications running.

- **Jersey**. This is a Java framework for simplifying the development of REST web services. This can help with the communications layer between microservices.
- **Swagger**. This is a Java framework for building APIs. This also can help with the communications layer between microservices.

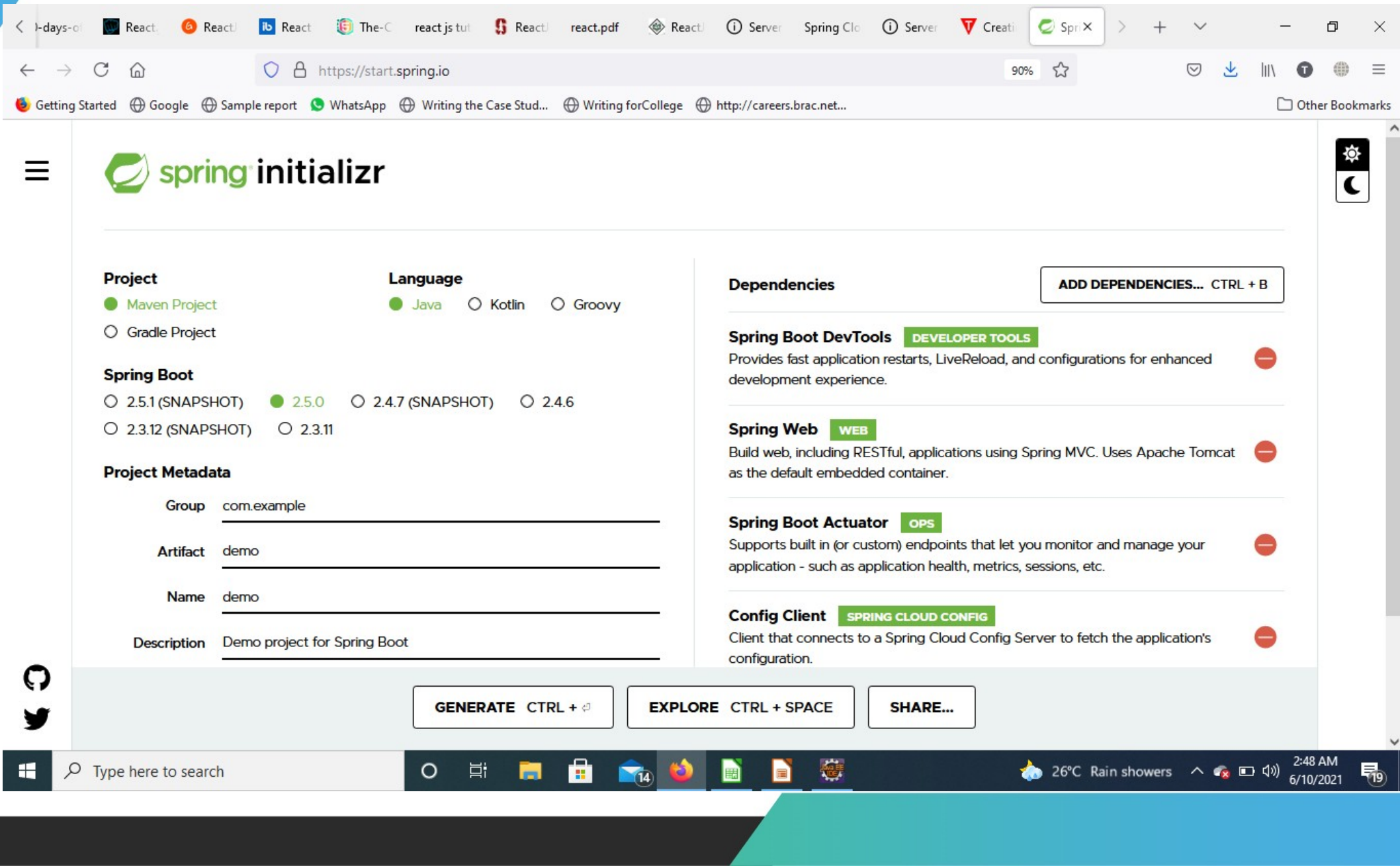
Creating Microservice

- **Step 1:** Create a Maven project using Spring Initializr https://start.spring.io/
- **Step 2:** Choose the Spring Boot version 2.2.0 M6 or higher version. Do not choose the snapshot version.
- **Step 3:** Provide the Group name. In our case *com.tarkeshwar.barua*
- **Step 4:** Provide the Artifact id. We have provided limits-service.
- **Step 5:** Add the following dependencies: *Spring Web, Spring Boot DevTools, Spring Boot Actuator, Config Client.*

Creating Microservice

- **Step 6:** Click on Generate the project button. A zip file will download, extract it into the hard disk.
- **Step 7:** Now, open the eclipse. Import the created maven project. It takes some time to download the required files.
- **Step 8:** Once the project is downloaded, go to *src/main/java*. Open the Demo*Application.*
- **Step 9:** Now run the Demo*Application.java* as Java Application.
- It started the Tomcat on port(s) 8080 (http).

Creating Microservice



application.properties

- spring.application.name = limits-service
- spring.datasource.url = jdbc:mysql://localhost:3306/iiht?useSSL=false&serverTimezone=UTC&useLegacyDatetimeCode=false
- spring.datasource.username=root
- spring.datasource.password=aaaAAA123
- server.port=9090
- spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
- spring.jpa.properties.hibernate.ddl-auto=update| create | create-drop
- logging.level.hibernate.SQL=DEBUG
- lLogging.level.org.hibernate.type=TRACE
- author.name=Tarkeshwar Barua
- spring.datasource.url=jdbc:h2:mem:testdb
- spring.h2.console.enabled=true
- management.endpoints.web.exposure.include=*

The Holistic System

- Microservices are interconnected and a change to one element can have a meaningful and some-times unpredictable impact on other elements.
- The system changes over time and is unpredictable.
- It produces behavior that is greater than the behavior of its individual components.
- It adapts to changing contexts, environments, and stimuli.
- Microservices system is complex and teasing desirable behaviors and outcomes from that system isn’t an easy task.
- Organizations have had enormous success in doing so and we can learn from their examples.

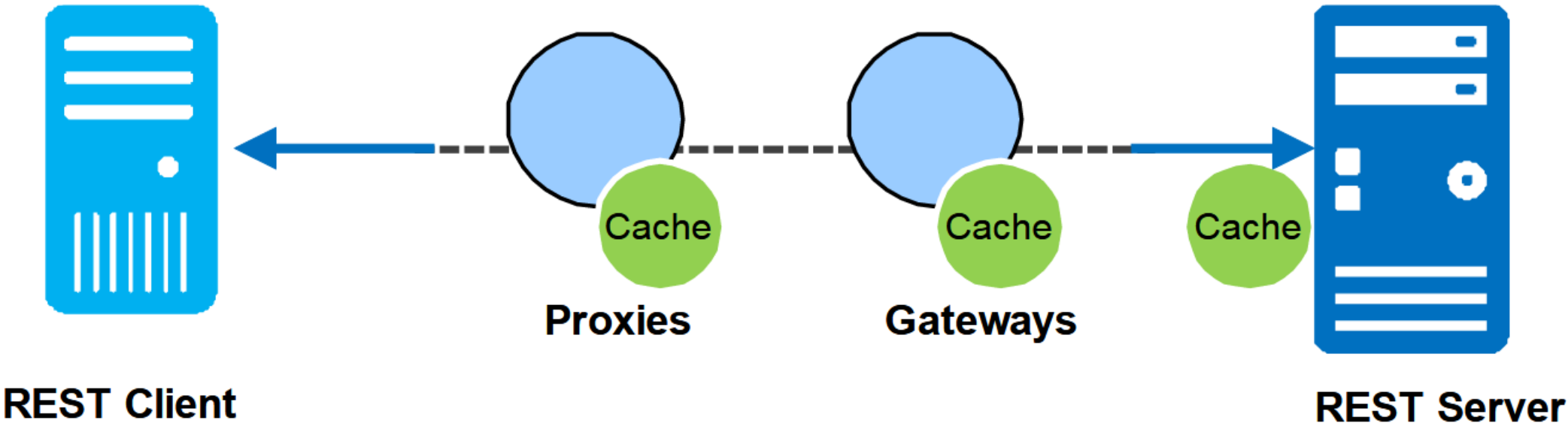
Advantages of Microservices

- Microservices are ideal for size big systems
  - The common theme among the problems that people were facing was related tosize. This is significant because it highlights a particular characteristic of themicroservices style—it is designed to solve problems for systems that are big.
- Microservice architecture is goal-oriented
  - Derive from James' recollection of the day is the focus on agoal rather than just a solution. Microservice architecture isn't about identifying aspecific collection of practices, rather it's an acknowledgment that software pro-fessionals are trying to solve a similar goal using a particular approach.
- Microservices are focused on replaceability
  - The revelation that microservices are really about replaceability is the mostenlightening aspect of the story. This idea that driving toward replacement ofcomponents rather than maintaining existing components get to the very heart ofwhat makes the microservices approach special.

What is REST

- REST is an architectural style for networked applications, primarily used to build web services that are lightweight, maintainable, and scalable.
- A service based on REST is called a RESTful service.
- REST is not dependent on any protocol, but almost every RESTful service uses HTTP as its underlying protocol.
- REST is often used for web applications as a way to allow resources to communicate by exchanging information.
- REST enables you to have applications that are loosely coupled, can be scaled and provide functionality across services.
- REST has emerged as a predominant web service design model, and almost every major development language includes frameworks for building RESTful web services.
- The REST APIs use HTTP verbs to act on a resource. The API establishes a mapping between CREATE, READ, UPDATE, and DELETE operations, and the corresponding POST, GET, PUT, and DELETE HTTP actions.

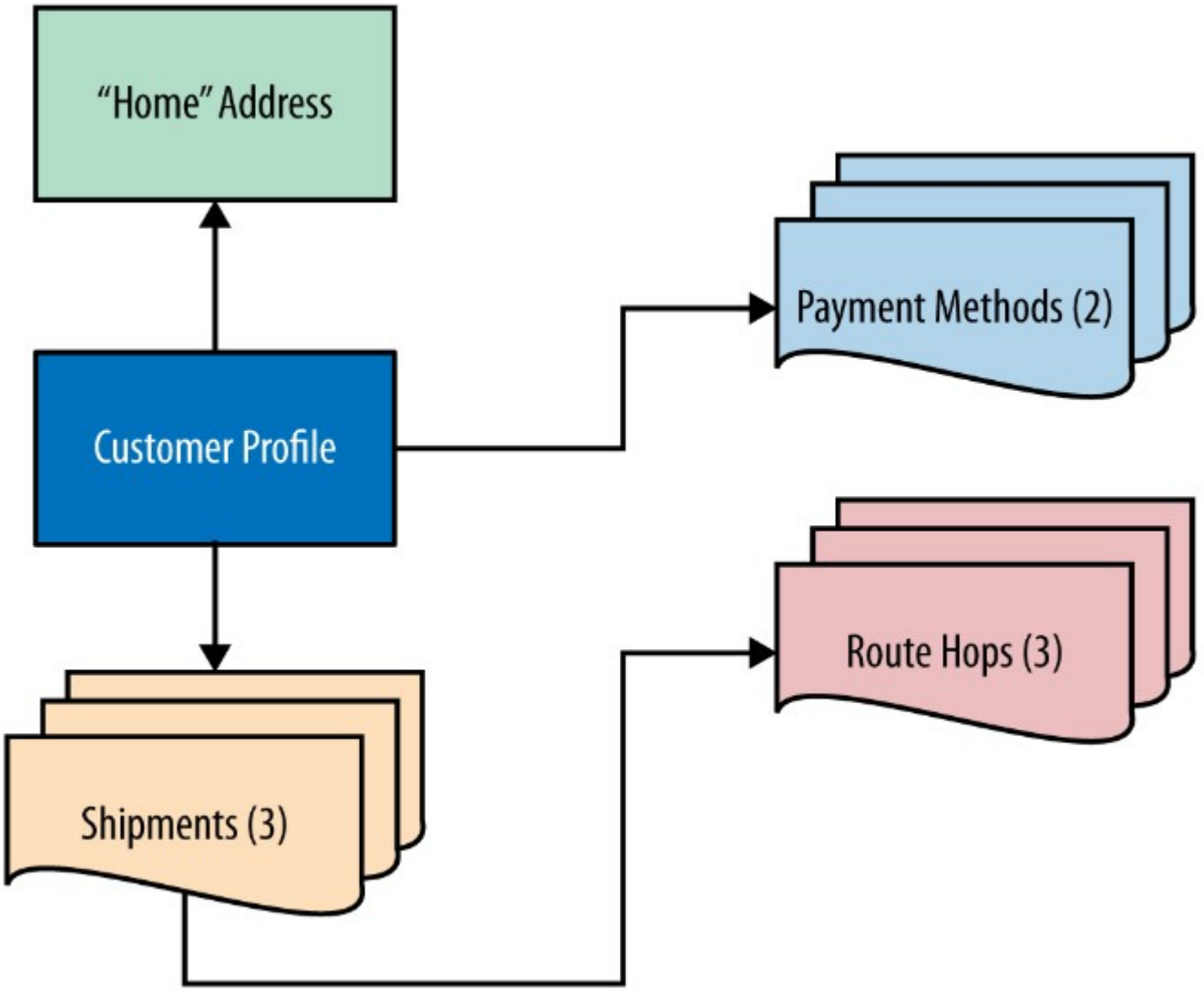
REST architecture with intermediary caches



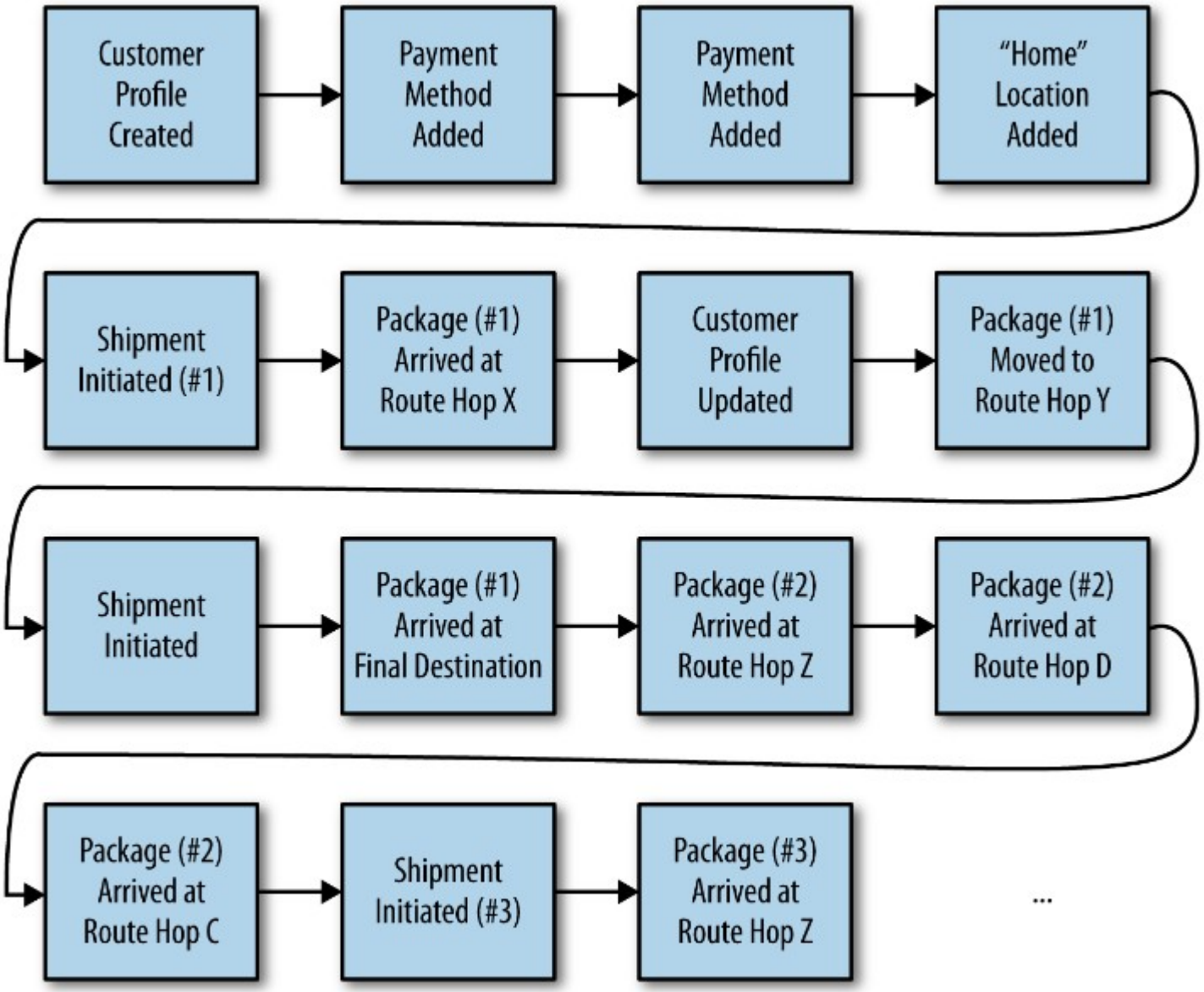
Features of Microservices

- Small in size and focused
- Messaging enabled
- Bounded by contexts
- Autonomously developed
- Independently deployable
- Decentralized
- Built and released with automated processes
- Loosely coupled
- Language-neutral

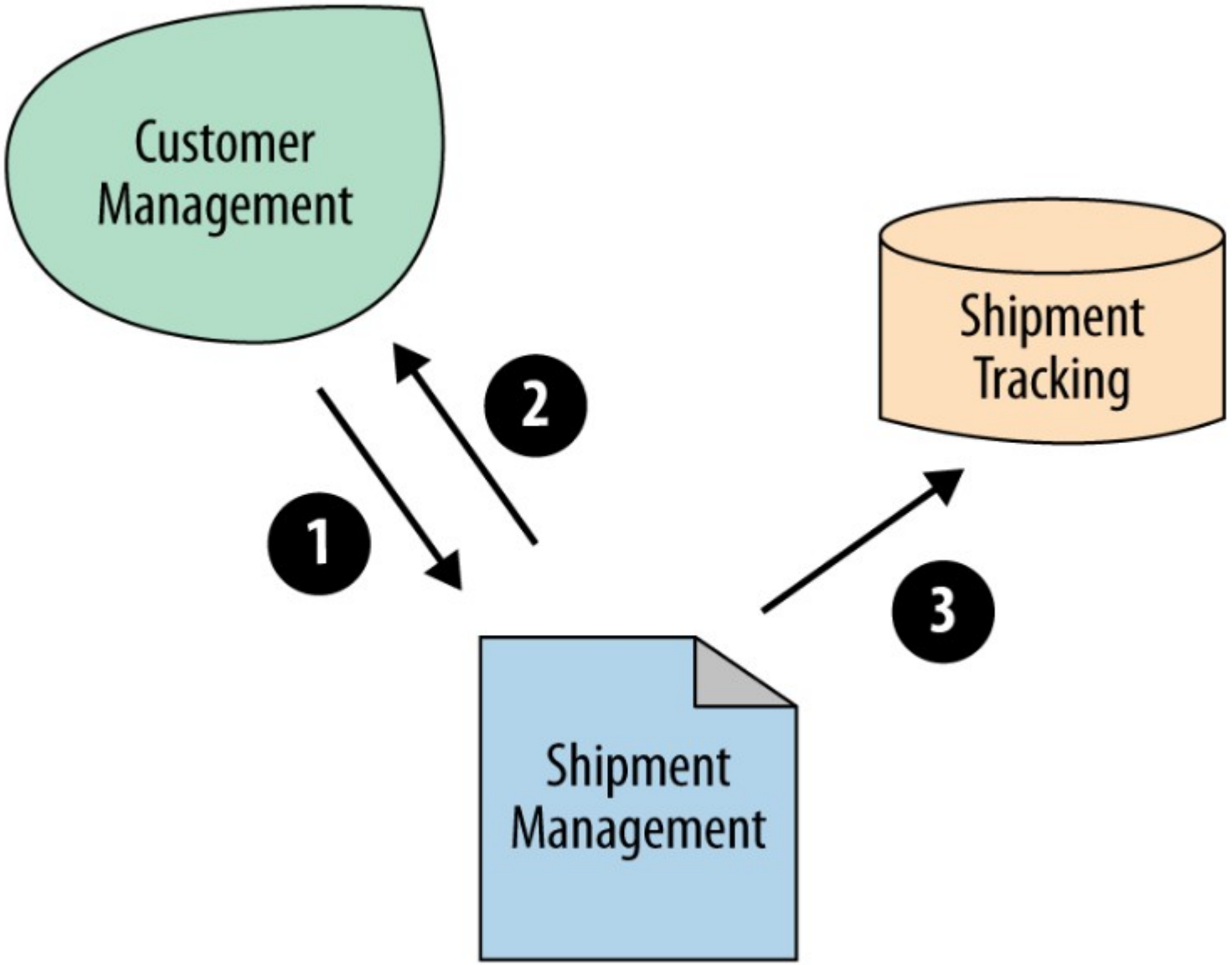
Example of Microservices



Example of Microservices



Example of Microservices



Example of Microservices

1)Customer Management creates, edits, enables/disables customer accounts, and can provide a representation of a customer to any interested context.
2)Shipment Management is responsible for the entire life cycle of a package from drop-off to final delivery.
3)It emits events as the package moves through sorting and forwarding facilities, along the delivery route.
4)Shipment Tracking is a reporting application that allows end users to track their shipments on their mobile device.

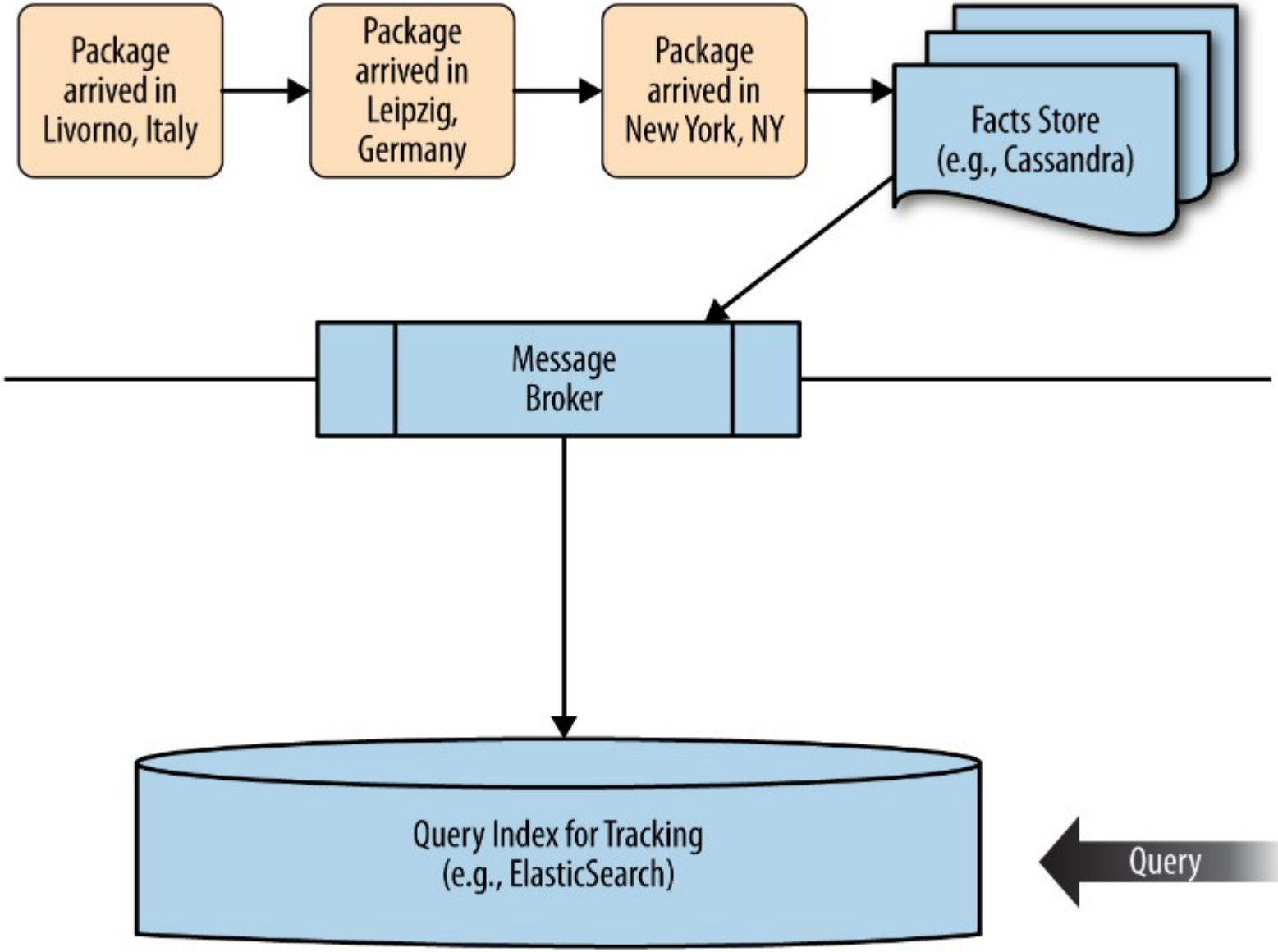
Drawback of Previous Approach

1)If we were to implement a data model of this application using a traditional, structural, CRUD-oriented model we would immediately run into data sharing and tight-coupling problems.
2)The Shipment Management and ShipmentTracking contexts will have to query the same tables, at the very least the ones containing the transitions along the route.
3)The ShipmentManagement bounded context (and its corresponding microservice) can instead record events/commands and issue event notifications for other contexts and thoseother contexts will build their own data indexes (projections), never needing directaccess to any data owned and managed by the Shipment Management microservice.
4)The formal approach to this process is described in a pattern called CQRS.

CQRS

1)**Command query responsibility segregation** is a design pattern that states that we can separate data-update versus data-querying capabilities into separate models.
2)It tracks its ancestry back to a principle called command–query sep-aration (CQS), which was introduced by **Bertrand Meyer** in his book **Object-Oriented Software Construction** (Prentice-Hall, 1997).
3)Meyer argued that data-altering opera-tions should be in different methods, separated from methods performing read-only operations.
4)CQRS takes this concept a large step further, instructing us to use entirely different models for updates versus queries.
5)This seemingly simple statement often turns out to be powerful enough to save the day, especially in the complicated case of the reports centric microservices.

Shipment Management Microservice



implementation stack

- Microservice systems consist of individual services running as separate processes, it stands to reason to expect any competent technology capable of supporting communication protocols, such as HTTP REST, or messaging protocols, such as MQ Telemetry Transport (MQTT) or Advanced Message Queuing Protocol (AMQP), to work.
- Nevertheless, several considerations need to be considering when choosing the implementation stack:
- Classic stacks, such as Java Platform, Enterprise Edition (Java EE), work by synchronous blocking on network requests.
- They must run in separate threads to be able to handle multiple concurrent requests.
- Asynchronous stacks handle requests using an event loop that is often single-threaded, yet can process many more requests when handling them requires downstream input/output (I/O) operations.

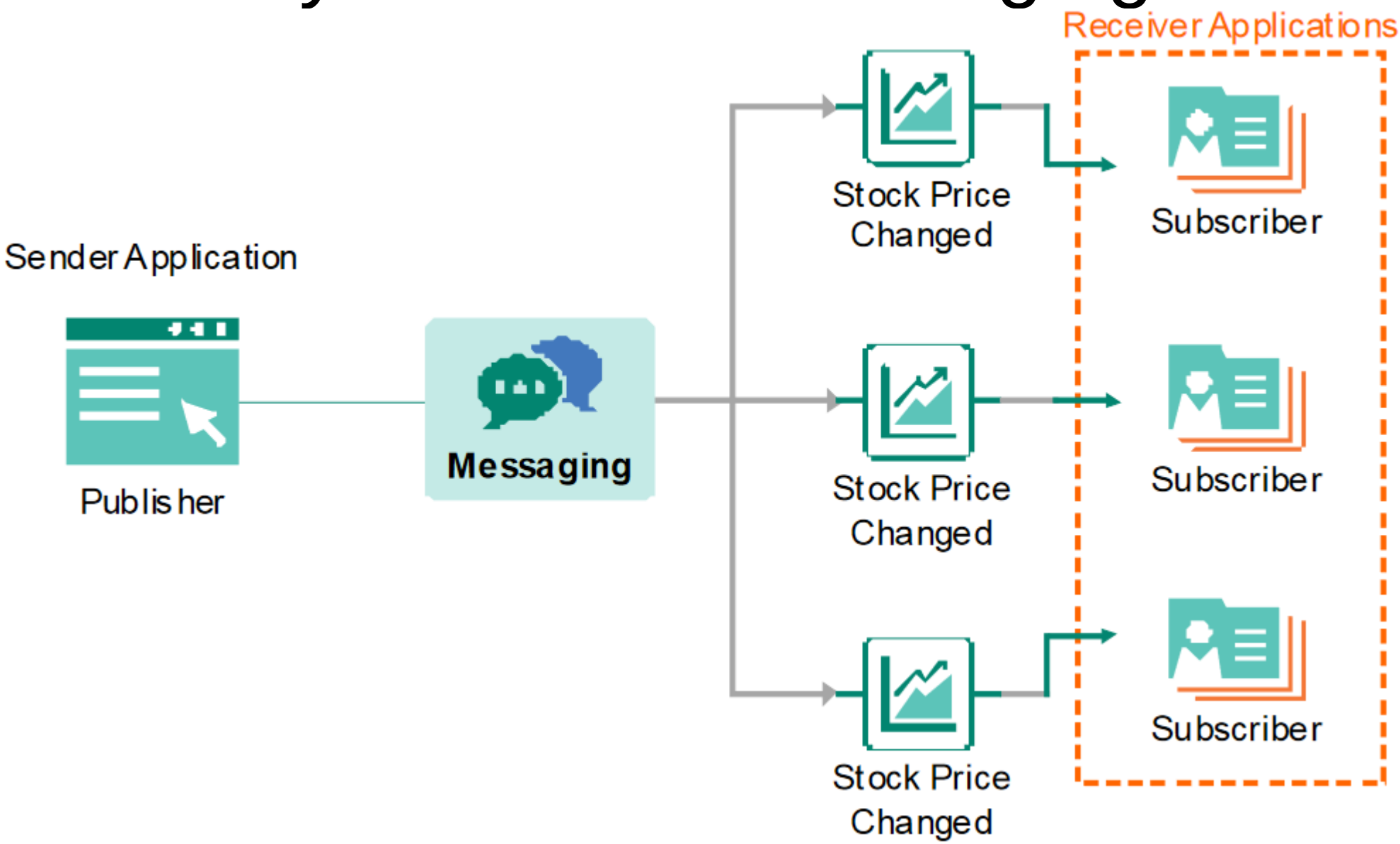
Synchronous HTTP vs asynchronous messaging

- Synchronous HTTP-based mechanisms, such as Representational State Transfer (REST), SOAP, or WebSockets, which enable keeping a communication channel between browser and server open for event-driven request/response
- Asynchronous messaging using a message broker
- Synchronous messaging is a simple and familiar mechanism, which is firewall-friendly.
- It does not support other patterns, such as a publish/subscribe model.
- The publish/subscribe messaging style is more flexible than point-to-point queuing, because it enables multiple receivers to subscribe to the flow of messages.
- This enables more microservices to be easily added on to the application.

Synchronous HTTP vs asynchronous messaging

- It also does not allow queuing of requests, which can act as a kind of shock absorber.
- **Asynchronous messaging decouples the services from each other in time, enabling the sending thread to get on with work while the messaging system takes responsibility for the message, keeping it safely until it can be delivered to the destination.**
- Synchronous applications, both the client and server must be simultaneously available, and the clients always need to know the host and port of the server, which is not always straightforward when services are auto-scaling in a cloud deployment.
- One of the service discovery mechanisms described previously is required. Alternatively, an asynchronous mechanism uses a message broker, which decouples the message consumers and producers.
- The message broker buffers messages until the consumer is able to process them.

Synchronous HTTP vs asynchronous messaging



Synchronous HTTP vs asynchronous messaging

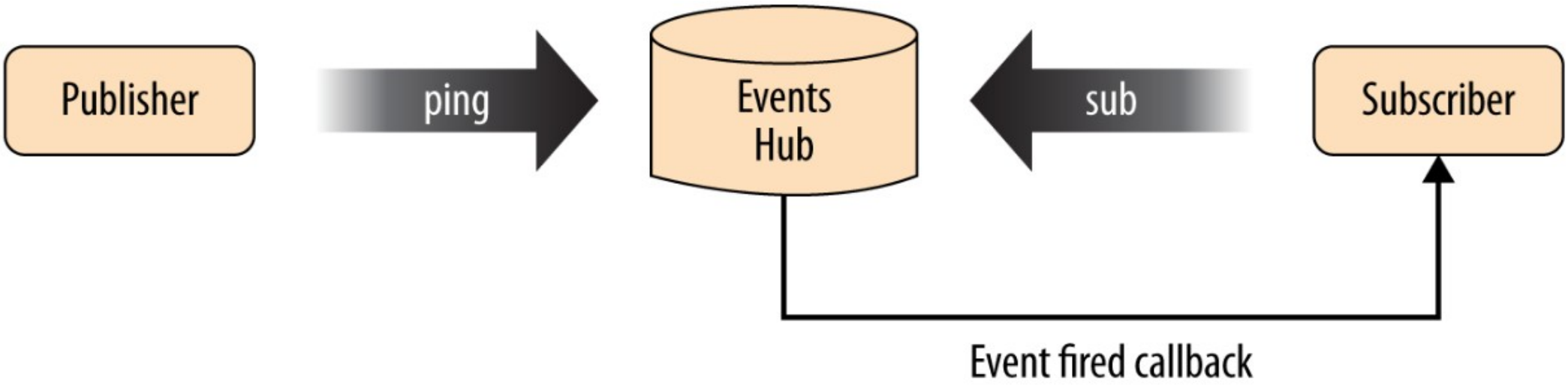
- A further benefit of messaging is that the back-end applications behind the queue can distribute workload among several clients.
- This type of worker offload scenario enables developers to build scalable, responsive applications.
- Implementing one or more instances of a back-end worker application enables the putting application to continue its work without the need to wait for the back-end application to complete.

Asynchronous Message-Passing and Microservices

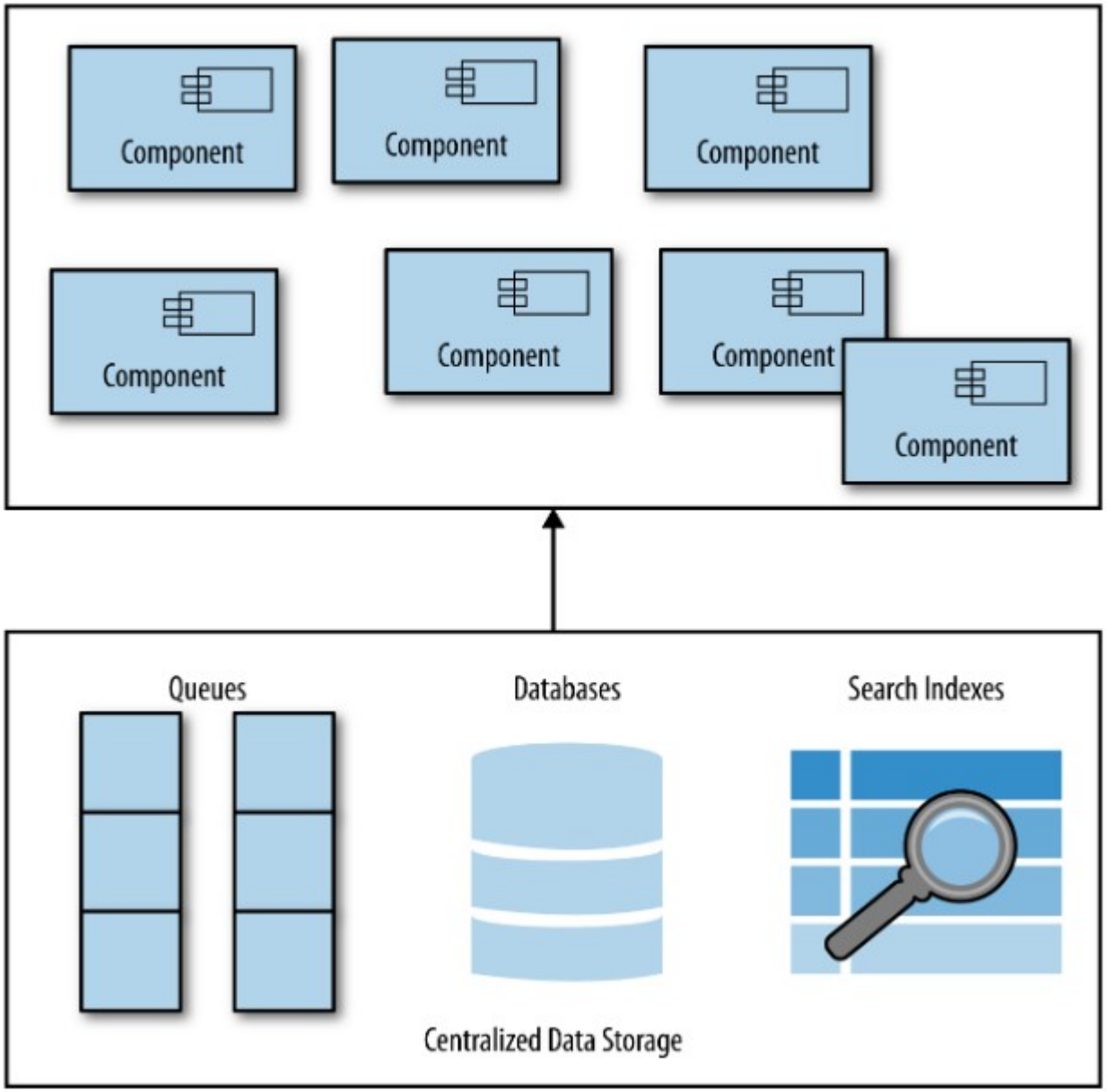
- Asynchronous message-passing plays a significant role in keeping things loosely coupled in a microservice architecture.
- Message broker to deliver event notifications from our Shipment Management microservice to the Shipment Tracking microservice in an asynchronous manner.
- Letting microservices directly interact with message brokers (such as RabbitMQ, etc.) is rarely a good idea.
- If two microservices are directly communicating via a message-queue channel, they are sharing a data space(the channel) and we have already talked, at length, about the evils of two microservi-ces sharing a data space.
- Encapsulate message-passing behind an independent microservice that can provide message-passing capability, in a loosely coupled way, to all interested microservices.

Simple publish/subscribe workflow

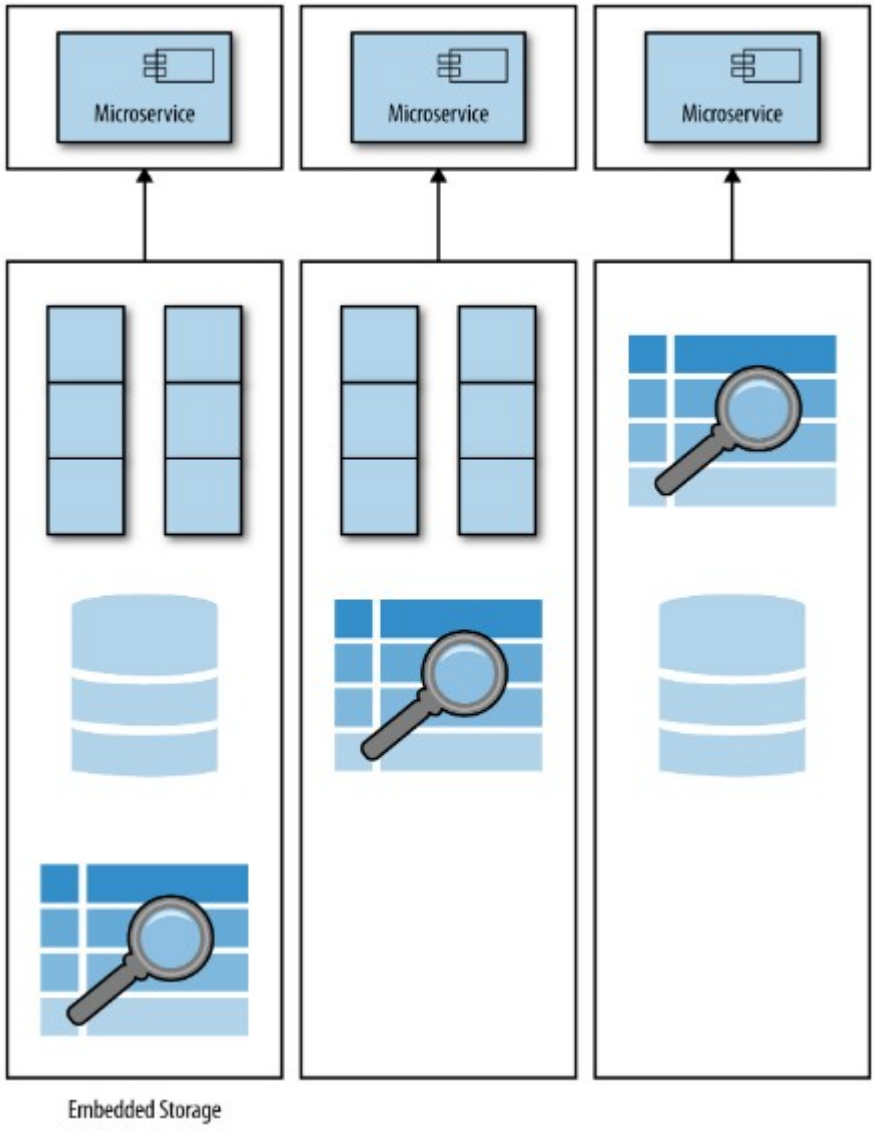
- PubSubHubbub wasn't created for APIs or hypermedia APIs, it was created for RSS and Atom feeds in the blogging context.
- We can adapt it relatively well to serve a hypermediaAPI-enabled workflow. To do so, we need to implement a flow similar to the one



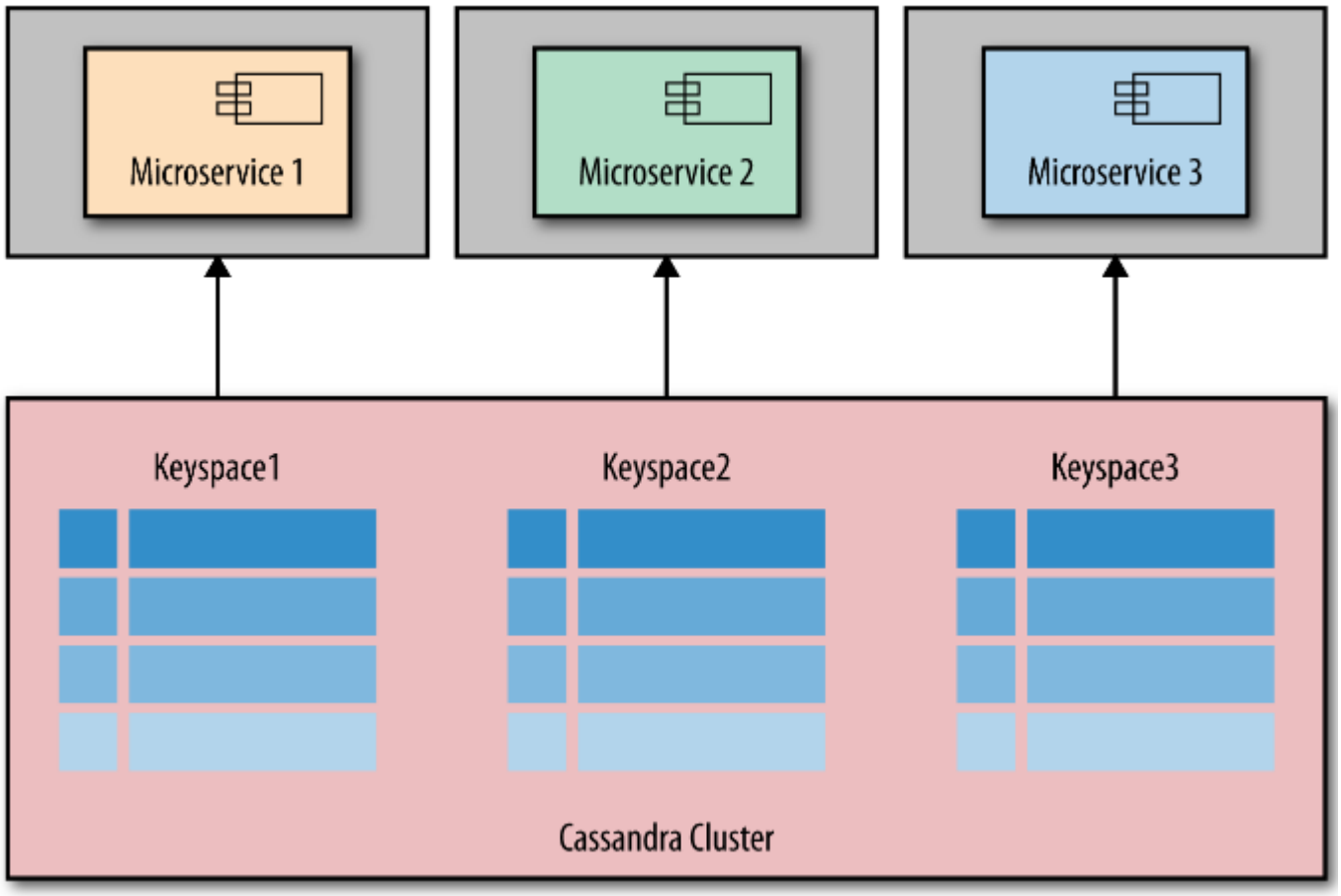
Components using a centralized pool of dependencies



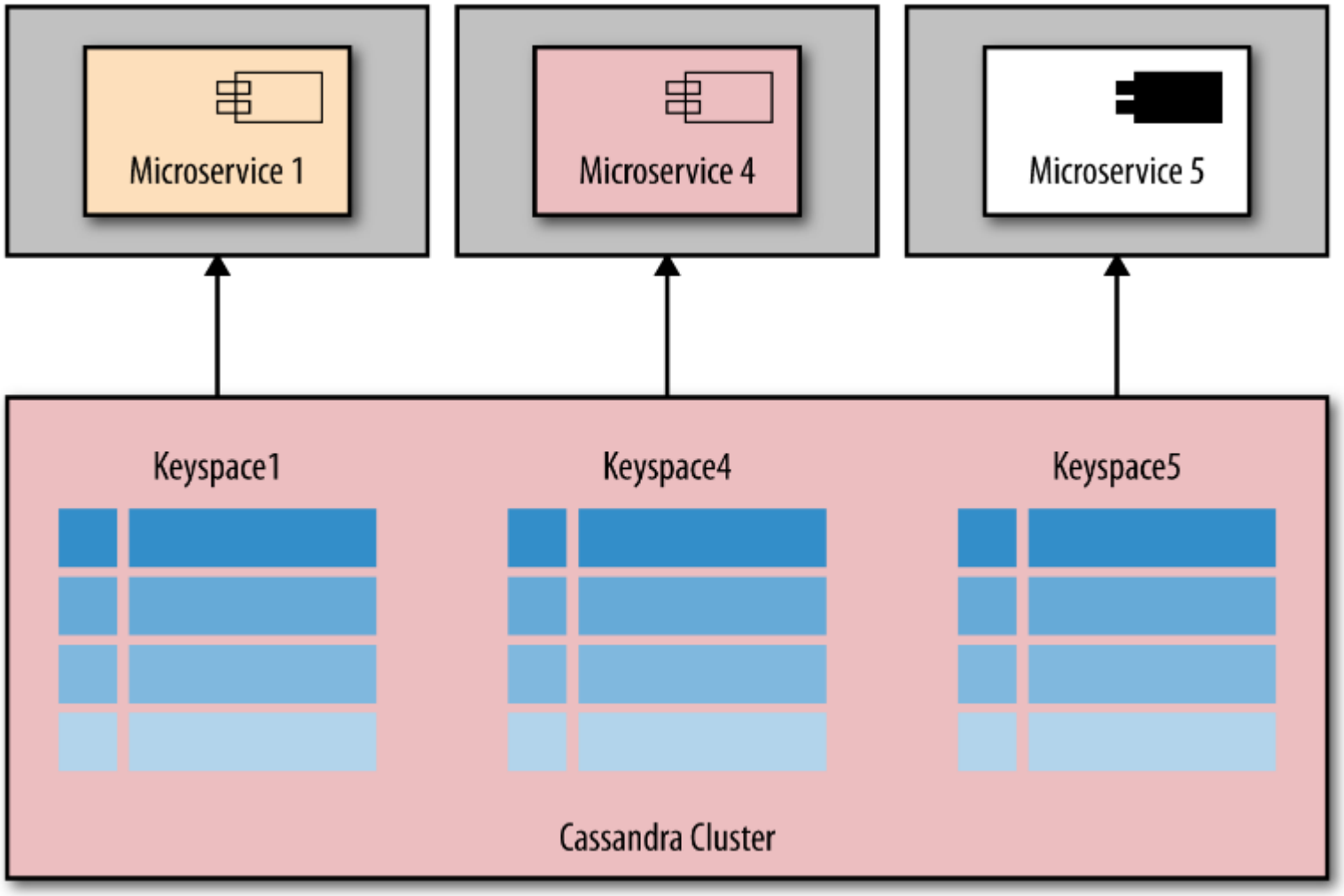
Components using fully embedded, isolated dependencies



Pragmatic approach: Components using a centralized pool of dependencies,without sharing data spaces



Pragmatic approach: Microservice 1's move to different data center made possible without data sharing



Messaging Protocol

- messaging protocol (such as MQTT or AMQP) can be better than REST to allow real-time event updates.
- An application uses the request/response pattern associated with RESTful services, a broken link means that no collaboration is happening.
- What if your message broker fails? Then messages are not delivered. To safeguard against a disruption in message delivery, an HA configuration can be implemented.
- You can also scale your applications so that you have multiple instances available to handle messages if one instance becomes unavailable.

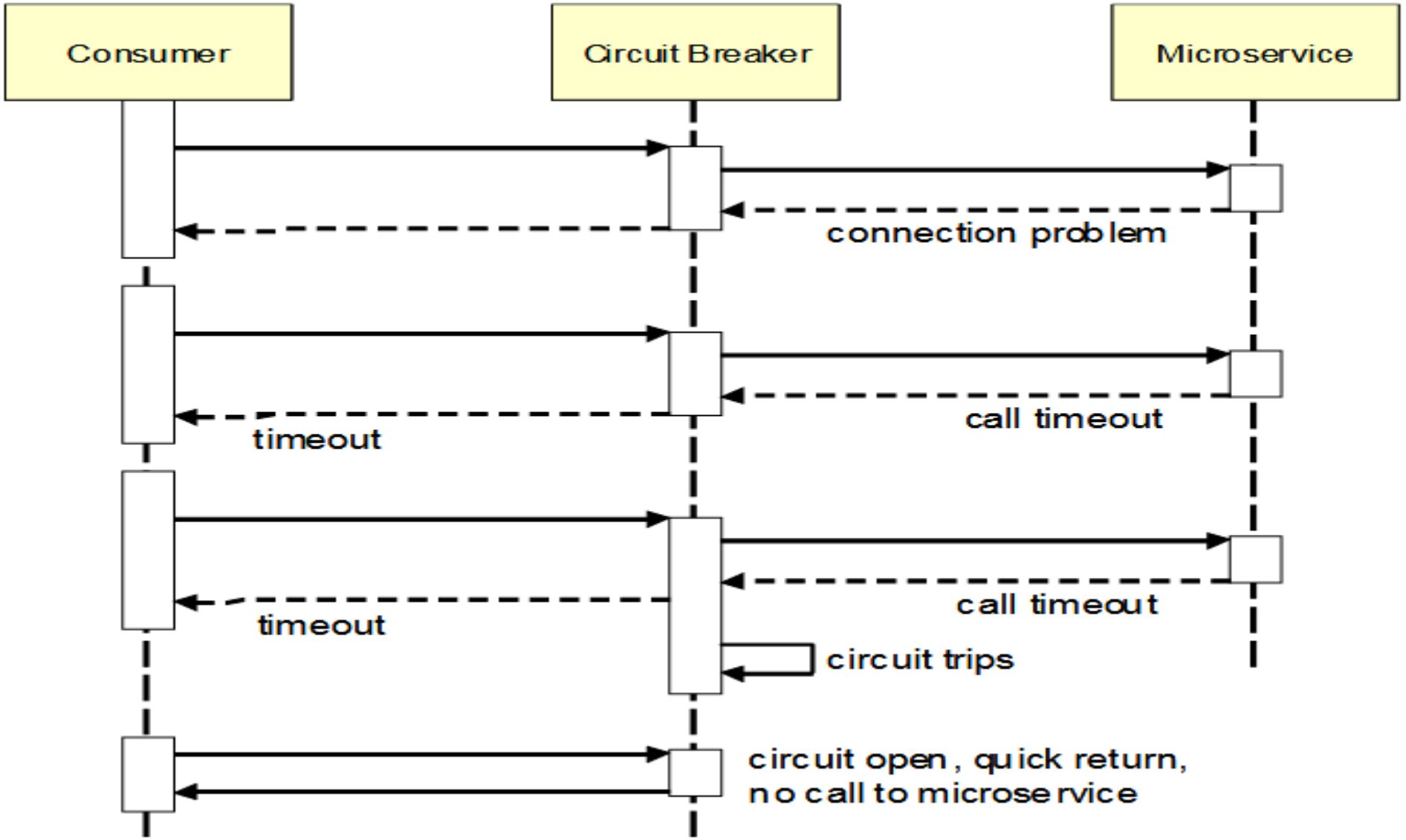
Messaging Protocol

- REST API and add messages for REST endpoints that result in a state change (POST, PUT, PATCH, or DELETE), an HTTP GET request can be done to fetch a resource.
- **If another service issues an update and publishes a message to notify subscribers that the information has changed, a downstream microservice consuming a REST API endpoint can also subscribe for the topic that matches the endpoint syntax.**
- With this technique, you can reduce REST frequent polling and still have the application up to date

Circuit Breaker

- The circuit breaker pattern is commonly used to ensure that when there is failure that the failed service does not adversely affect the entire system.
- This would happen if the volume of calls to the failed service was high, and for each call we’d have to wait for a timeout to occur before moving on.
- Making the call to the failed service and waiting would use resources that would eventually render the overall system unstable.

Circuit Breaker….



Bulkheads

- A ship’s hull is composed of several individual watertight bulkheads.
- The reason for this is that if one of the bulkheads gets damaged, that failure is limited to that bulkhead alone, as opposed to taking down the entire ship.
- This kind of a partitioning approach can be used in software as well, to isolate failure to small portions of the system.
- The service boundary serves as a bulkhead to isolate any failures. Breaking out functionality (as we would also do in an SOA architecture) into separate microservices serves to isolate the effect of failure in one microservice

Microservices Architecture (MSA) and Services-Oriented Architecture (SOA)

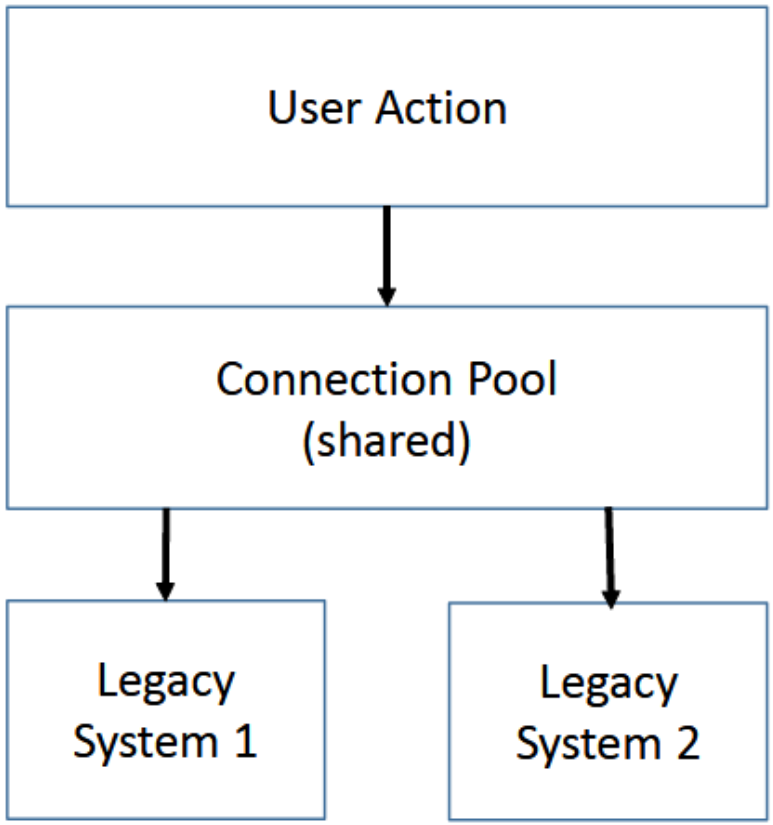
Microservices Architecture	**Services Oriented Architecture**
Microservices uses lightweight protocols such as REST, and HTTP, etc.	SOA supports multi-message protocols.
It focuses on decoupling	It focuses on application service reusability.
It uses a simple messaging system for communication.	It uses Enterprise Service Bus (ESB) for communication.
Microservices follows "share as little as possible" architecture approach.	SOA follows "share as much as possible architecture" approach.
Microservices are much better in fault tolerance in comparison to SOA.	SOA is not better in fault tolerance in comparison to MSA.
Each microservice have an independent database.	SOA services share the whole data storage.

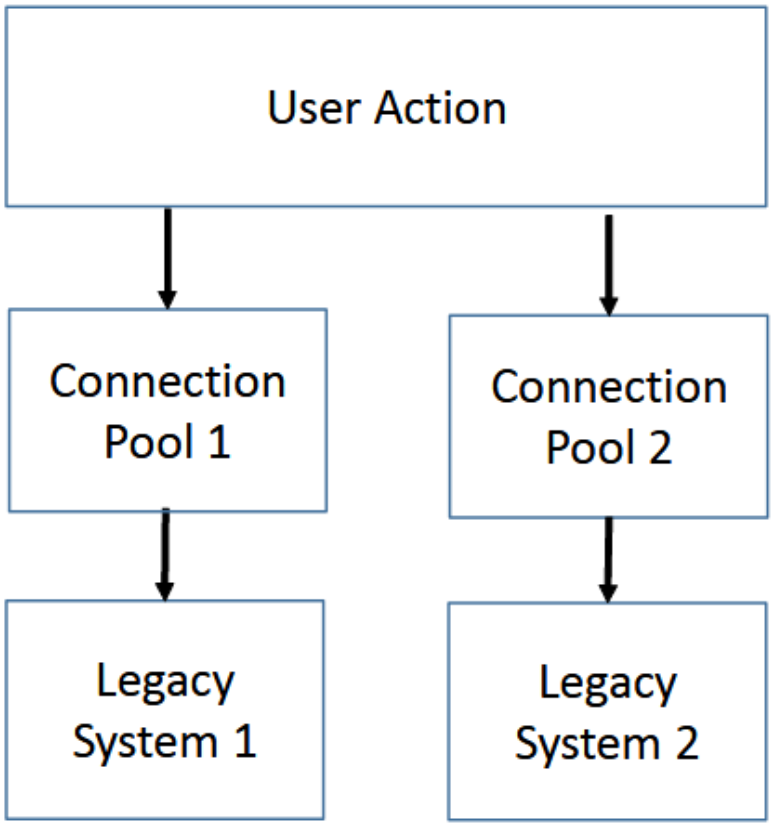
Microservices Architecture (MSA) and Services-Oriented Architecture (SOA)

Microservices Architecture	**Services Oriented Architecture**
MSA used modern relational databases.	SOA used traditional relational databases.
MSA tries to minimize sharing through bounded context (the coupling of components and its data as a single unit with minimal dependencies).	SOA enhances component sharing.
It is better suited for the smaller and well portioned, web-based system.	It is better for a large and complex business application environment.

Bulkheads...

bulkhead – separate connection pools
isolate slowdown in a legacy system





Goals Achieved By Microservices

- **Reduce Cost:** Will this reduce overall cost of designing, implementing, and maintaining IT services?
- **Increase Release Speed:** Will this increase the speed at which my team can get from idea to deployment of services?
- **Improve Resilience:** Will this improve the resilience of our service network?
- **Enable Visibility:** Does this help me better see what is going on in my service network?

Questions

- Why are organizations adopting microservices?
- What are the motivations and challenges?
- How can the leaders of these organizations tell that taking on the challenges of managing a collection of small, loosely coupled, independently deploy-able services is actually paying off for the company?
- What is the measure of success?
- What is the mean of “balancing speed and safety at scale.”

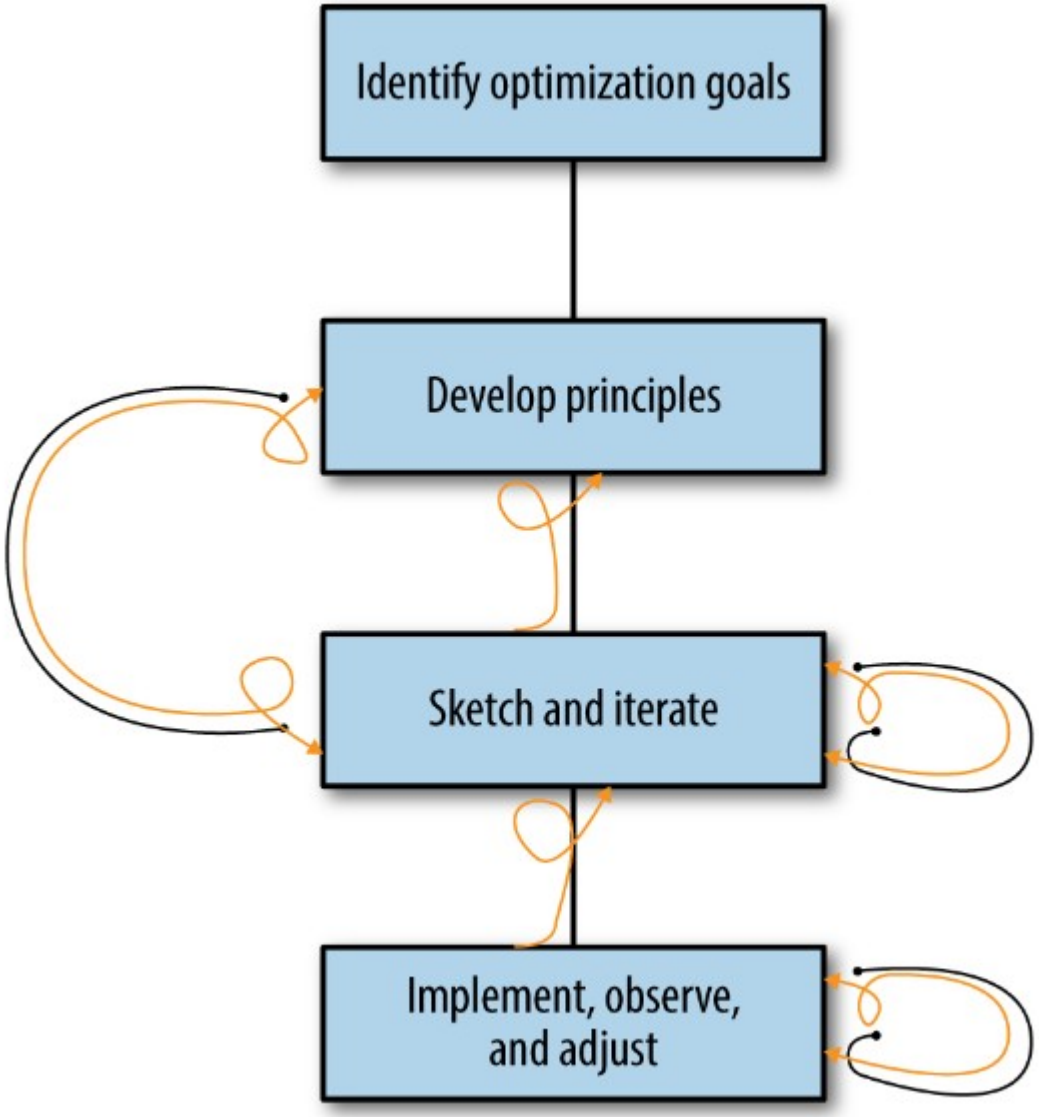
Netflix

- Netflix has been open about their own journey toward creating a successful microservice architecture is Netflix.
- In 2013, Adrian Cockcroft, Netflix's CloudArchitect, presented a day-long workshop on Netflix's cloud architecture and operating principles.

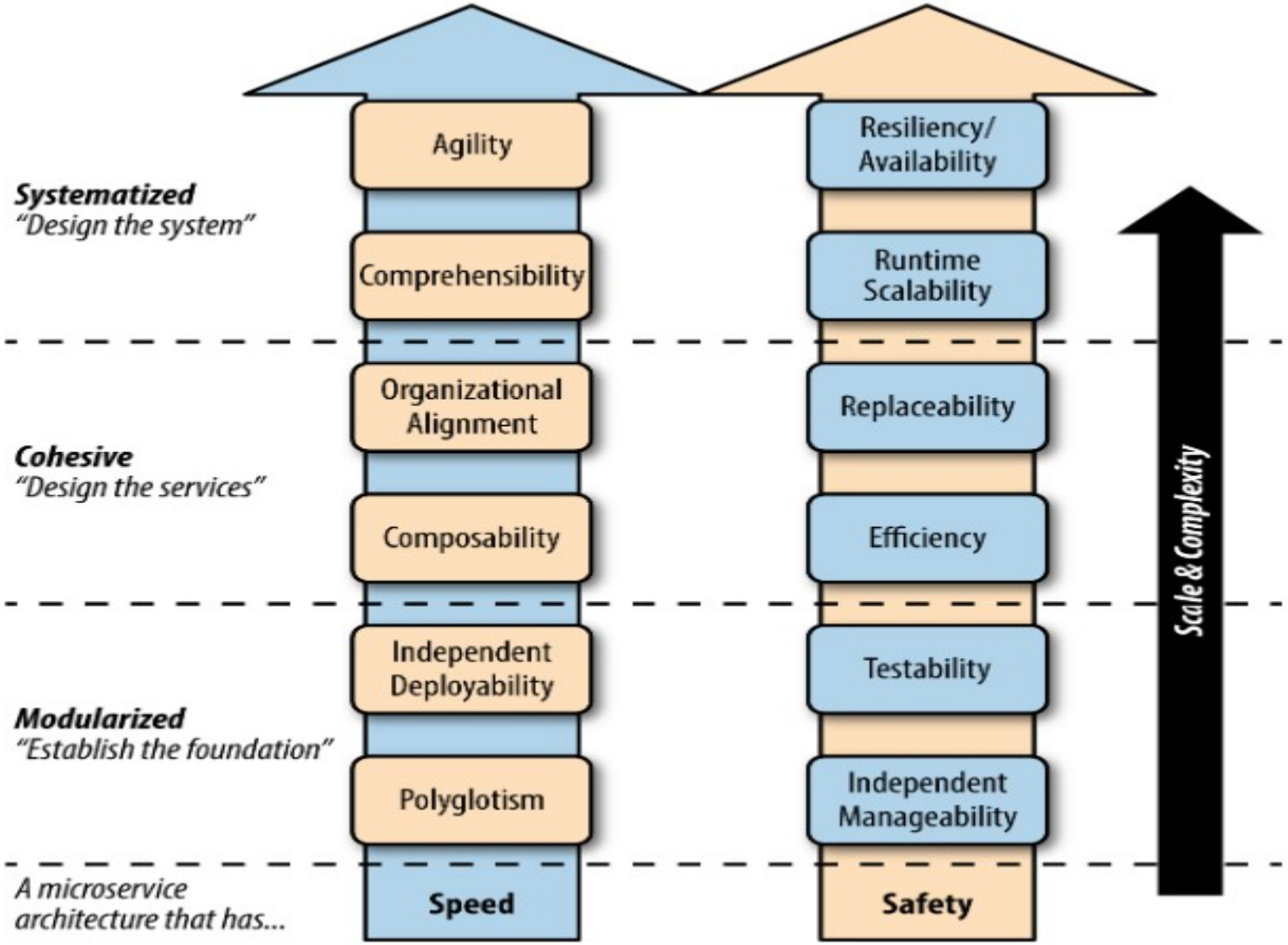
Working Principle of Netflix

- Antifragility
  - The point of antifragility is that you always want a bit ofstress in your system to make it stronger.” There are several things Netflix does topromote this, including their “Simian Army” set of tools, which “enforce archi-tectural principles, induce various kinds of failures, and test our ability to survivethem”. Software has bugs, operators make mistakes, and hardware fails. By creat-ing failures in production under controlled conditions, developers are incentiv-ized to learn to build more robust systems. Error reporting and recovery systemsare regularly tested, and real failures are handled with minimal drama and customer impact
- Immutability
  - auto-scaled groups of service instances are stateless and identical, which enables Netflix’s system to “scale horizontally.” The Chaos Monkey, a member of the SimianArmy, removes instances regularly to enforce the immutable stateless serviceprinciple. Another related technique is the use of “Red/Black pushes”. Althougheach released component is immutable, a new version of the service is introducedalongside the old version, on new instances, then traffic is redirected from old tonew. After waiting to be sure all is well, the old instances are terminated.
- Separation of Concerns
  - Each team owns a group of services.They own building, operating, and evolving those services, and present a stableagreed interface and service level agreement to the consumers of those services.Invoking Conway’s law, an organization structured with independent self-contained cells of engineers will naturally build what is now called a microservicearchitecture.

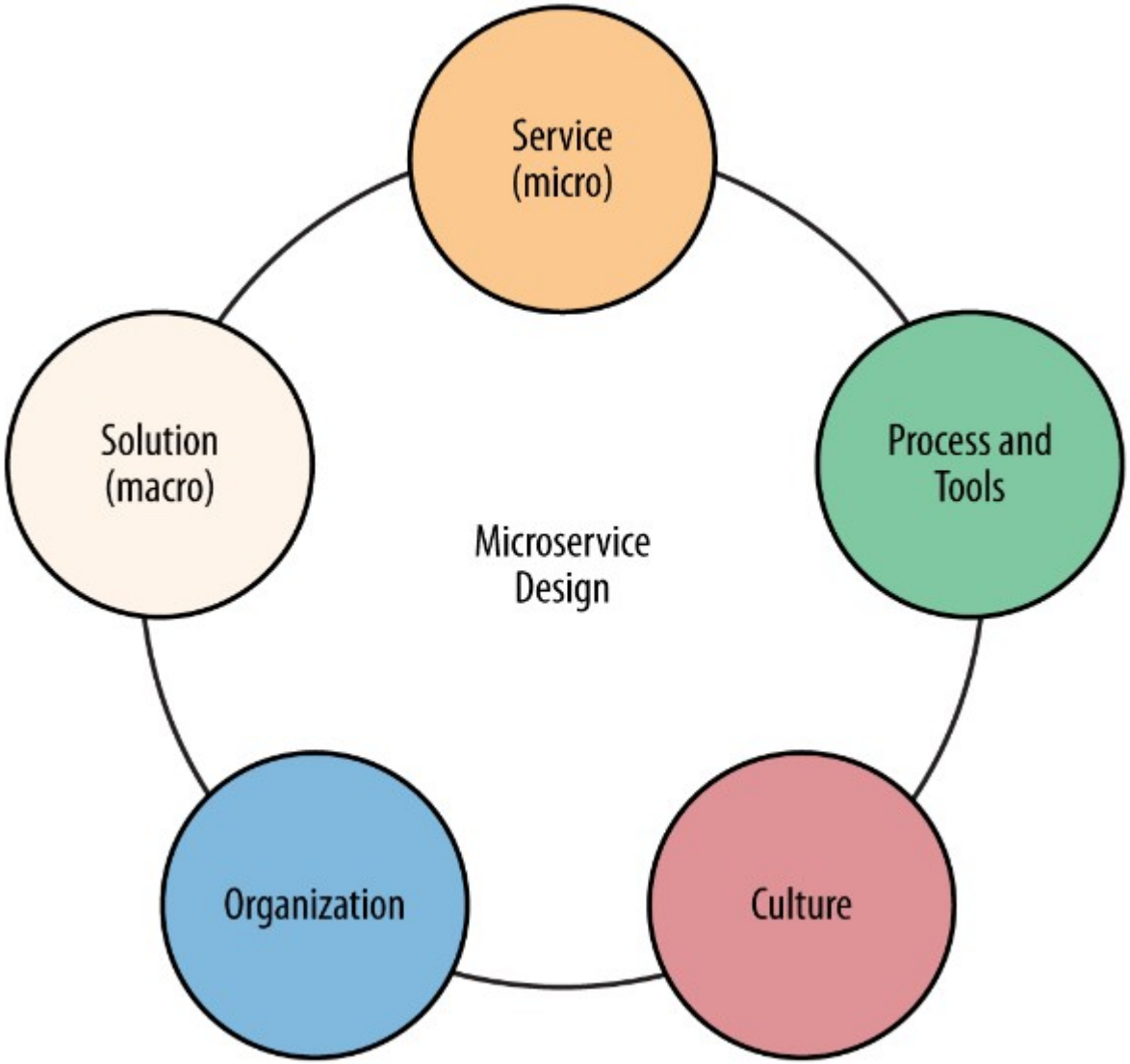
Design Process



Maturity Model for Microservice Architecture Goals and Benefits



The Microservice System Design Model



Service

- Implementing well-designed microservices and APIs are essential to a microservice system.
- The services form the atomic building blocks from which the entire organism is built.
- If you can get the design, scope, and granularity of your service just right you'll be able to induce complex behavior from a set of components that are deceptively simple.

Solution

- A solution architecture is distinct from the individual service design elements because it represents a macro view of our solution.
- When designing a particular microservice your decisions are bounded by the need to produce a single output—theservice itself.
- When designing a solution architecture your decisions are bounded by the need to coordinate all the inputs and outputs of multiple services.
- The macro-level view of the system allows the designer to induce more desirable system behavior.
- For example, a solution architecture that provides discovery, safety, androuting features can reduce the complexity of individual services

Process and Tools

- Your microservice system is not just a byproduct of the service components that handle messages at run-time.
- The system behavior is also a result of the processes and tools that workers in the system use to do their job.
- Usually includes tooling and processes related to software development, code deployment, maintenance, and product management.
- Choosing the right processes and tools is an important factor in producing good microservice system behavior
- For example, adopting standardized processes like DevOps and Agile or tools like Docker containers can increase the changeability of your system.

Organization

- How we work is often a product of who we work with and how we communicate.
- The microservice system perspective, organizational design includes the structure,direction of authority, granularity, and composition of teams.
- Many of the companies that have had success with microservice architecture point to their organizational design as a key ingredient.
- Organizational design is incredibly context-sensitive and you may find yourself in a terrible situation if you try to model your 500+employee enterprise structure after a 10-person startup (and vice versa).
- A good microservice system designer understands the implications of changing these organizational properties and knows that good service design is a byproduct of good organizational design.

Culture

- culture is a context-sensitive feature of your system.
- What works in Japan may not work in the United States and what works in a large insurance firm may not work at an e-commerce company.
- So, you'll need to be cau-tious when attempting to emulate the practices that work in a company whose cultureyou admire. There is no recipe or playbook that will guarantee you the same results.

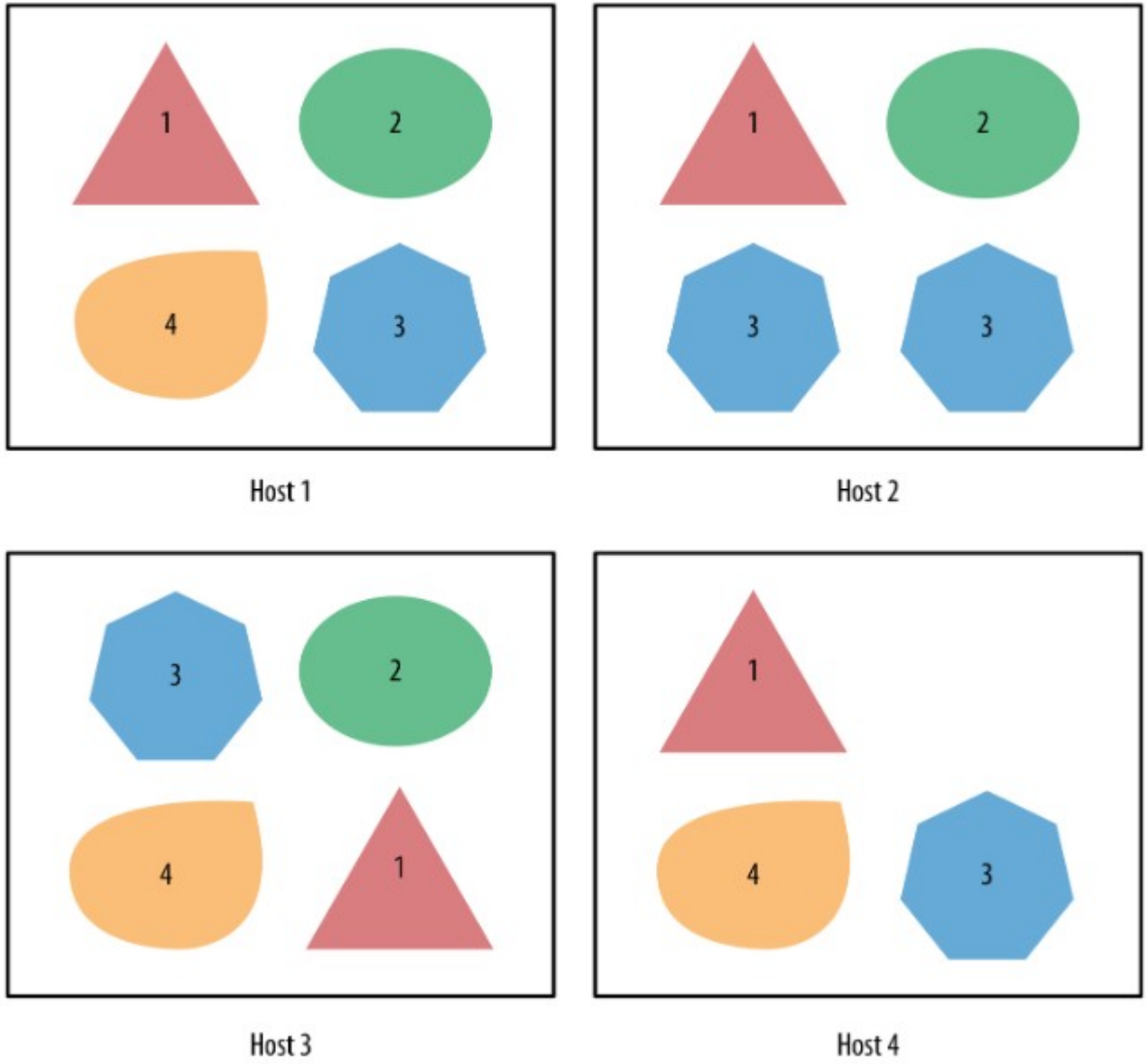
Cloud Watch

- CloudTrail and CloudWatch Events are important building blocks to track and respond to infrastructure changes across microservices, AWS Config rules allow a company to define security policies with specific rules to automatically detect, track,and alert you to policy violations
- Detect, inform, and automatically react to non-compliant configuration changes within your microservices architecture.
- A member of the development team has made a change to the API Gateway for a microservice to allow the endpoint to accept inbound HTTP traffic,rather than only allowing HTTPS requests.

The Role of Service Discovery

- Simple Docker Compose configuration to orchestrate multiple microservi-ces (and their containers) into a coherent application.
- As long as you are on a single host (server) this configuration will allow multiple microservices to “discover” and communicate with each other. This approach is commonly used in local development and for quick prototyping.

Microservice deployment topology with nonuniform service distribution



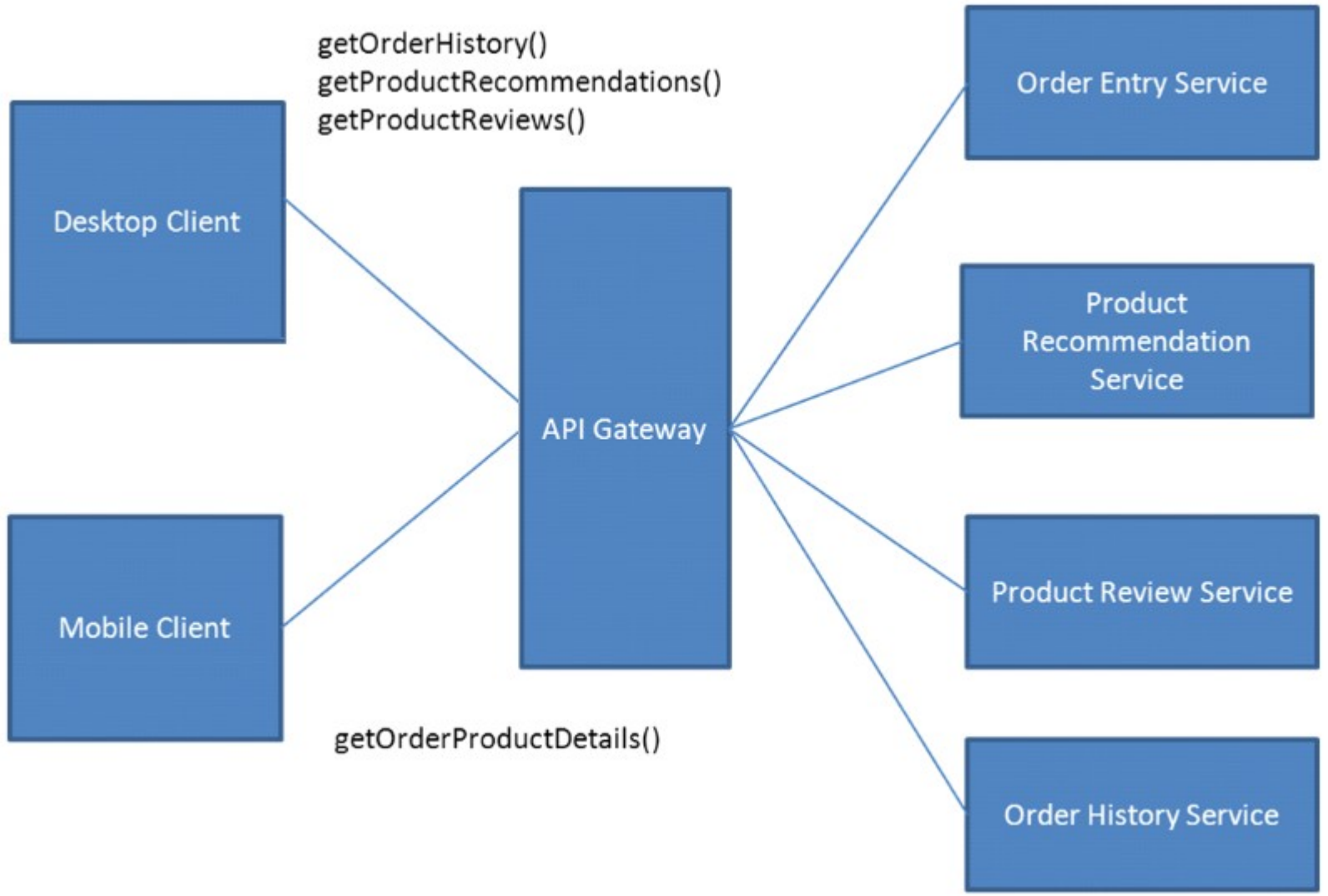
API Gateway

- A common pattern observed in virtually all microservice implementations is teams securing API endpoints, provided by microservices, with an API gateway.
- Modern API gateways provide an additional, critical feature required by microservices: trans-formation and orchestration.
- API gateways cooperate with service discovery tools to route requests from the clients of microservices.
- API gateway features and clarify their role in the overall architecture of the operations layer for microservices.

API Gateway….

- Some clients can be “chatty”; others can require many calls to get the necessary data.
- The services would probably scale differently, there are other challenges to be addressed such as handling of partial failures.
- A better approach is for the clients to make a few requests, perhaps only one, using a front-end such as an API Gateway

API Gateway….


getOrderHistory()
getProductRecommendations()
getProductReviews()
Desktop Client
Mobile Client
API Gateway
Order Entry Service
Product Recommendation Service
Product Review Service
Order History Service
getOrderProductDetails()

API Gateway

- The API Gateway sits between the clients and microservices, and provides APIs that are tailored for the clients.
- The API Gateway is principally about hiding technological complexity versus interface complexity.
- API Gateway is a facade. However, a facade provides a uniform view of complex internals to external clients, where an API Gateway provides a uniform view of external resources to the internals of an application.

API Gateway

- API Gateway provides a simplified interface to the clients, making the services easier to use, understand, and test.
- It can provide different levels of granularity to desktop and browser clients.
- API Gateway might provide coarse-grained APIs to mobile clients, and fine-grained APIs to desktop clients that might use a high performance network.

API Security

- Microservice architecture is an architecture with a significantly high degree of freedom.
- There are a lot more moving parts than in a monolithic application.
- In mature microservices organizations where the architecture is implemented for complex enterprise applications, it is common to have hundreds of microservices deployed.
- Things can go horribly wrong security-wise when there are many moving parts.
- We certainly need some law and order to keep everything in control and safe. The API gateway is used to do so.

Transformation and Orchestration

- Microservices are typically designed to provide a single capability.
- They are the Web's version of embracing the Unix philosophy of "**do one thing, and do it well**."
- As any Unix developer will tell you, the single responsibility approach only works because Unix facilitates advanced orchestration of its highly specialized utilities, through universal piping of inputs and outputs.
- Using pipes, you can easily combine and chain Unix utilities to solve nontrivial problems involving sophisticated process workflows.
- A critical need for a similar solution exists in the space of APIs and microservices as well.
- To make microservices use-ful, we need an orchestration framework like Unix piping, but one geared to webAPIs.

Routing

- In order to properly discover microservices we need to use a service discovery system like Consul and etcd, will monitor your microservice instances and track metadata about what IPs and ports each one of your microservices is available at, at any given time.
- Directly providing tuples of the IP/port combinations to route an API client is not an adequate solution.
- A proper solution needs to abstract implementation details from the client.
- An API client still expects to retrieve an API at a specific URI, regardless of whether there's a microservice architecture behind it and independent of how manyservers, Docker containers, or anything else is serving the request.

Monitoring and Alerting

- Microservice architecture delivers significant benefits, it is also a system with a lot more moving parts than the alternative monolith.
- When implementing a microservice architecture, it becomes very important to have extensive system-wide monitoring and to avoid cascading failures.
- service discovery can also provide powerful monitoring and failover capabilities. Consul as an example.
- How many active containers exist for a specific service, marking a service broken if that number is zero, but Consul also allows us to deploy customized health-check monitors for any service.
- This can be very useful. Indeed, just because a container instance for a microservice is up and running doesn’t always mean the microservice itself is healthy. We may want to additionally check that the microser-vice is responding on a specific port or a specific URL, possibly even checking that the health ping returns predetermined response data.

Assessing Your Organization

- How are responsibilities divided?
- Are responsibilities aligned to business or technology?
- Do you practice DevOps, or Dev and Ops?
- How big are the teams?
- What kinds of skills do they have?
- What are the dependencies for cross-team communication?
- What does the power distribution look like between teams?

User Interface Layer

- Modern web applications often use JavaScript and frameworks to implement a single-page application that communicates with a Representational State Transfer (REST) or RESTful API.
- Static web content can be served using Ordinary Application Server, Amazon Simple Storage Service (Amazon S32)and Amazon CloudFront3.Since clients of a micro service are served from the closest edge location and get responses either from a cache or a proxy server with optimized connections to the origin, latency can be significantly reduced.
- Micro services running close to each other don’t benefit from a CDN(Content Delivery Network). In some cases, this approach might actually add additional latency.
- The best practice is to implement other caching mechanisms to reduce chattiness and minimize latency.

What is Microservice

- APIs are the front door of microservices.
- APIs serve as the entry point for applications logic behind a set of programmatic interfaces, typically a RESTful web services API.
- API accepts and processes calls from clients and might implement functionality such as traffic management, request filtering, routing, caching, authentication,and authorization

Implementation of Microservice

- Apache Tomcat
- JBoss
- AWS Lambda
- Docker containers with AWS Fargate.
- Amazon Elastic Container Service (Amazon ECS8)
- Amazon Elastic Kubernetes Service (Amazon EKS9) eliminate the need to install

Elastic Container Service

- Amazon ECS supports container placement strategies and constraints to customize how Amazon ECS places and terminates tasks.
- A task placement constraint is a rule that is considered during task placement.
- You can associate attributes, essentially key-value pairs, to your container instances and then use a constraint to place tasks based on these attributes.
- For example, you can use constraints to place certain microservices based on instance type or instance capability, such as GPU-powered instances.

Elastic Kubernetes Service

- Amazon EKS runs up-to-date versions of the open-source Kubernetes software, so you can use all the existing plugins and tooling from the Kubernetes community.
- Applications running on Amazon EKS are fully compatible with applications running on any standard Kubernetes environment, whether running in on-premises data centers or public clouds.
- Amazon EKS integrates IAM with Kubernetes, enabling you to register IAM entities with the native authentication system in Kubernetes.
- There is no need to manually set up credentials for authenticating with the Kubernetes masters. The IAM integration allows you to use IAM to directly authenticate with the master itself as provide fine granular access to the public endpoint of your Kubernetes masters.

AWS Fargate

- AWS Fargate is a container management service that allows you to run serverless containers so you don’t have worry about provisioning, configuring, and scaling clusters of virtual machines to run containers.
- No longer have to worry about provisioning enough compute resources for your container applications.
- Fargate can launch tens of thousands of containers and easily scale to run your most mission-critical applications

Docker with AWS

- Docker images used in Amazon ECS and Amazon EKS can be stored in Amazon Elastic Container Registry (Amazon ECR.)
- Amazon ECR eliminates the need to operate and scale the infrastructure required to power your container registry

Methodologies

- Agile software development
- Service-oriented architectures
- API-first design
- Continuous Integration/Continuous Delivery (CI/CD)

Service-oriented Architectures

- Agile software development
- Service-oriented architectures
- API-first design
- Continuous Integration/Continuous Delivery (CI/CD)

API-first Design

- Agile software development
- Service-oriented architectures
- API-first design
- Continuous Integration/Continuous Delivery (CI/CD)

Continuous Integration/Continuous Delivery

- Continuous integration and continuous delivery (CI/CD) is a best practice and a vital part of a DevOps initiative that enables rapid software changes while maintaining system stability and security.
- This is out of the scope of this course, more information can be found in the“Practicing Continuous Integration and Continuous Delivery on AWS”

Twelve Factor App

- Agile software development
- Service-oriented architectures
- API-first design
- Continuous Integration/Continuous Delivery (CI/CD)

Serverless

- No infrastructure to provision or manage
- Automatically scaling by unit of consumption
- “Pay for value” billing model
- Built-in availability and fault tolerance

Monitoring and Auditing

- No infrastructure to provision or manage
- Automatically scaling by unit of consumption
- “Pay for value” billing model
- Built-in availability and fault tolerance

Docker Container

- Docker-enabled applications, query the complete state of your cluster
- Access many familiar features like security groups, Load Balancing, Amazon Elastic Block Store (Amazon EBS)volumes, and AWS Identity and Access Management (IAM) roles

Private Link

- highly available, scalable technology that enables you to privately connect your VPC to supported AWS services, services hosted by other AWS accounts (VPC endpoint services), and supported AWS Marketplace partner services.
- Do not require an internet gateway, NAT device, public IP address, AWS Direct Connect connection, or VPN connection to communicate with the service.
- Traffic between your VPC and the service does not leave the Amazon network.

Private Link…..

- Private links are a great way to increase the isolation of microservices architectures,
  - e.g.,it is possible to create hundreds of VPCs, each hosting and providing a single microservice.
- Companies can now create services and offer them for sale to other AWS customers, for access via a private connection.
- They create a service that accepts TCP traffic, host it behind a Network Load Balancer, and then make the service available, either directly or in AWS Marketplace.
- They will be notified of new subscription requests and can choose to accept or reject each one.
- The power of AWS PrivateLink has merits in any number of scenarios, it’s of particular interest to SaaS organizations.
- SaaS providers see new and creative opportunities to use this networking construct to enhance and expand the architectural and business models of their solutions