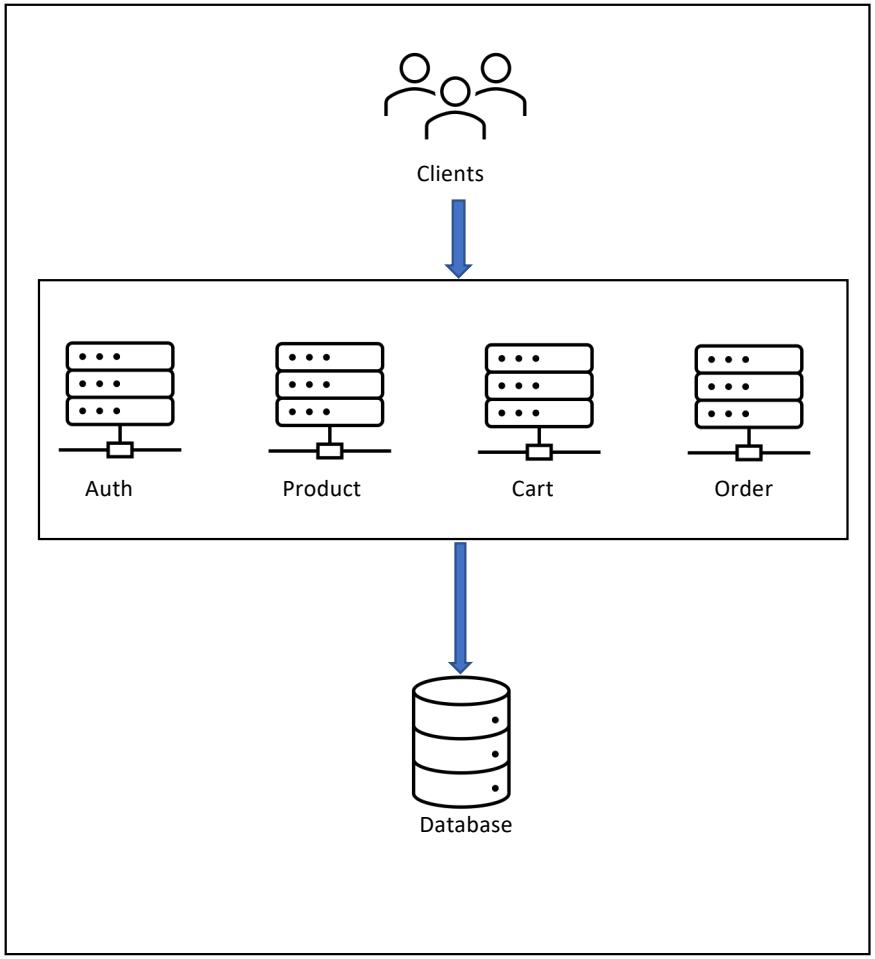
Monolithic Architecture

What is a Monolithic Architecture?

- Traditional and unified approach to software development.
- Application is built as a single, tightly integrated unit.
- It usually involves
  - Client-side
  - Server-side
  - Database layer



Benefits of a Monolithic Architecture

- Single codebase
- Simple to develop
- Easy deployment
- Easier debugging
- Simplified testing
- Single technology stack

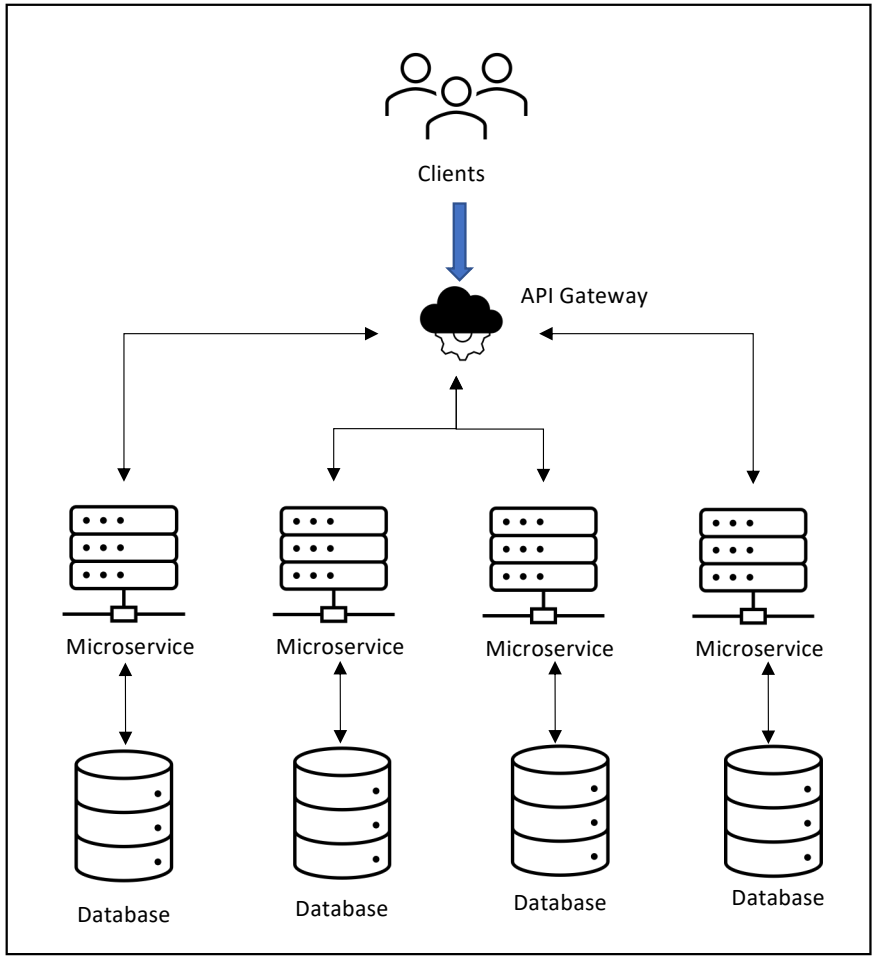
Challenges of a Monolithic Architecture

- Potentially difficult to understand and maintain due to its size and complexity
- Less team autonomy
- Single technology stack
- Potentially slow deployment pipeline
- No possibility of scaling individual subdomains
- If there’s an error in any subdomain, it could affect the entire application.

Microservices Architecture

What is Microservices Architecture?

- Architectural style that structures an application as a collection of small, independently deployable services
- Architectural is oriented mainly to the back-end, although the approach is being used for the front-end
- Services communicate via well-defined APIs
- Communication uses protocols such as HTTP/HTTPS, Websockets, or Advanced Message Queuing Protocol (AMQP)
- API communication could be direct client-to-microservice or API gateway pattern



Benefits of Microservices Architecture?

- **Scalability**: Microservices allow each service to be independently scaled to meet demand for the application feature it supports.
- **Faster Development**: Teams can work on and release individual services independently, leading to faster development cycles.
- **Flexibility in technology Stack**: This allows teams to choose the best tools and programming languages for specific tasks.
- **Fault Isolation**: Microservices are designed to be fault-tolerant. If one service fails, it doesn't necessarily bring down the entire system.

- **Improved Maintainability**: Smaller services are easier to understand, maintain and update.
- **Isolation of Data**: Microservices can have their own data storage solutions, which can lead to improved data isolation and security.
- **Enhanced Monitoring and Observability**: Each service can have its own monitoring and observability tools, making it easier to track performance, troubleshoot issues, and gather insights into the behavior of a system.
- **Easy Deployment**: Microservices enables continuous integration and continuous delivery, making it easier to try out new ideas.

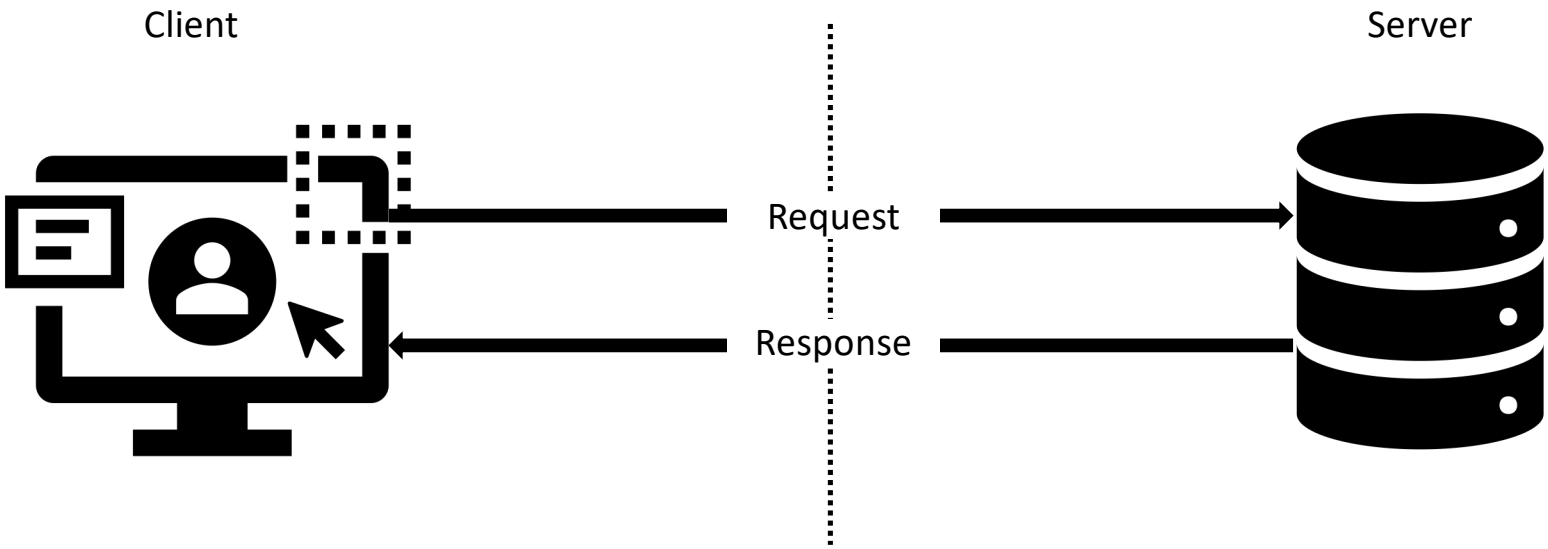
Challenges of Microservices Architecture?

- **Operational Complexity**: Managing a large number of microservices can be operationally complex in terms of infrastructure, tools for deployment, monitoring, etc.
- **Inter-Service Communication**: As microservices rely on network communication, there can be issues with latency, reliability, and data consistency between services.
- **Testing and Debugging**: testing microservices individually and ensuring proper integration can be challenging. Debugging issues that span multiple services can be complex and time-consuming.

- **Security and Compliance**: Security can be more challenging in a microservices environment, as there are more potential entry points for security breaches.
- **Monitoring and Observability**: Monitoring and tracking the behavior of multiple microservices can be complex.
- **Team Communication and Collaboration**: Effective communication and collaboration between teams are crucial for asuccessful implementation.

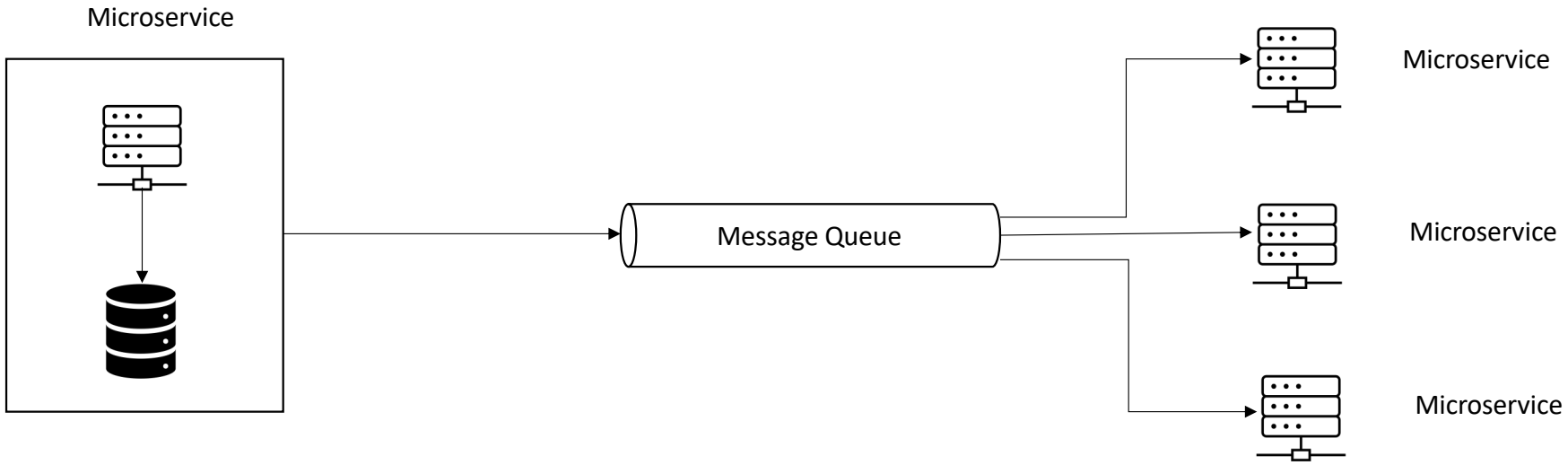
Microservices Communication Types

Synchronous Communication



- The client sends a request and waits for a response from the service.
- **HTTP** uses synchronous communication.
- The communication protocol can be **HTTP** or **HTTPS.**

Asynchronous Communication

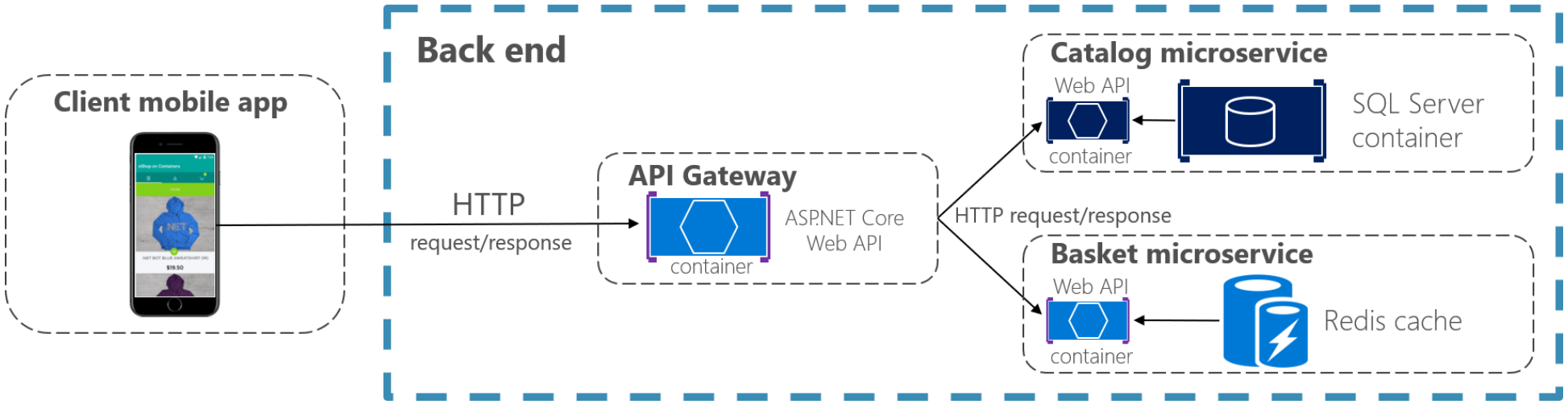


- The client sends a request but doesn't wait for a response from the service(s).
- It uses **AMQP** (Advanced Message Queuing Protocol)with message brokers like **Kafka** and **RabbitMQ**.
- Communication could be **one-to-one** or **one-to-many**.

Microservices Communication Styles

Request/response communication for live queries and updates

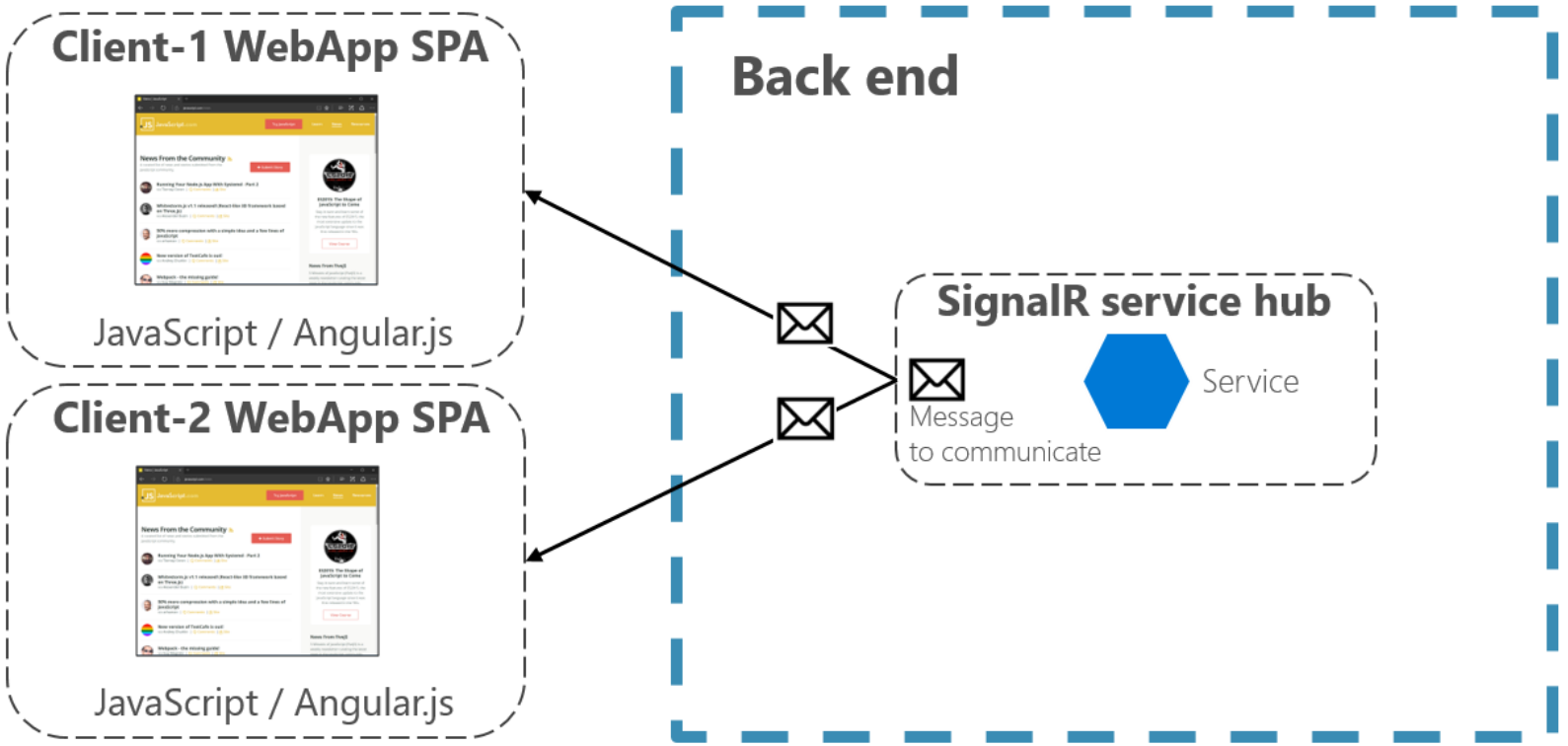
HTTP-based Services



Source: Microsoft Learn

Push and real-time communication based on HTTP

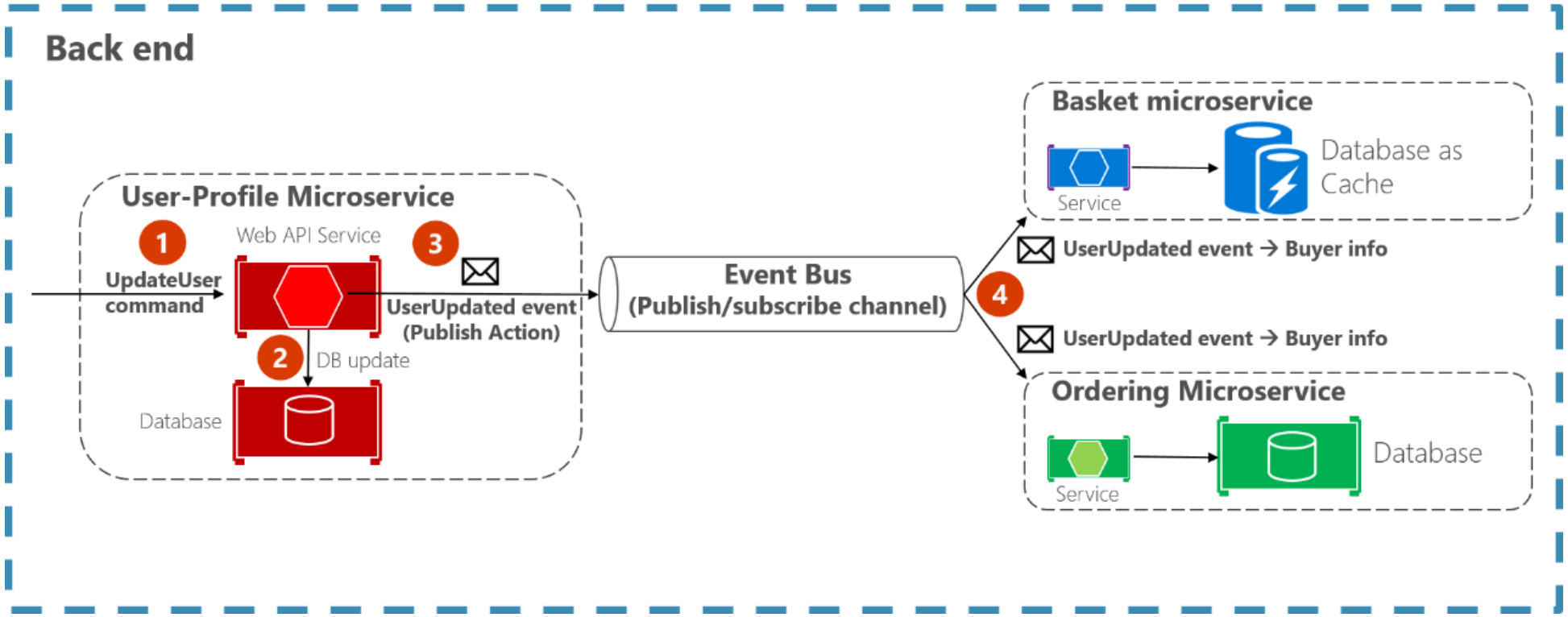
One-to-many communication



Source: Microsoft Learn

Asynchronous event-driven communication

Multiple receivers



Source: Microsoft Learn

Project Description

What are we going to build?

A freelance marketplace where sellers can create gigs and buyers can purchase gigs or custom gigs.

Freelance Marketplace Subdomains

Handles sending emails to users

Handles gig CRUD operations

Handles reviews and ratings

Handles users authentication

Handles buyers and sellers messaging

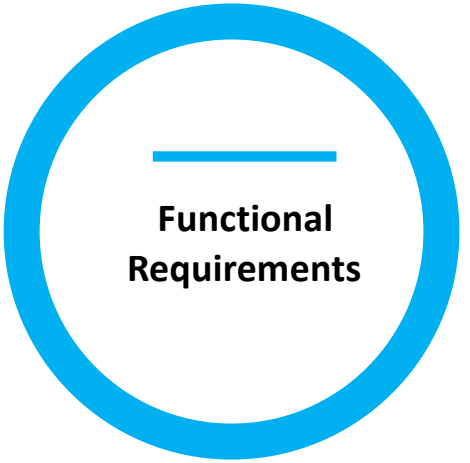
Handles Sellers and Buyers Profiles

Handles orders and payment

Functional Requirements

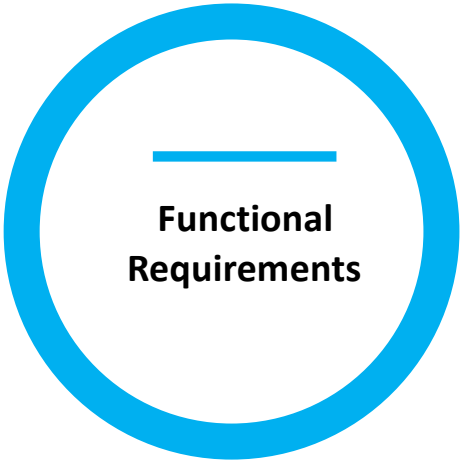
Functional Requirements

- Functional requirements are specific features, behaviors, and capabilities that a system must possess.
- These requirements describe what the system should do.
- They are typically defined in detail to guide the design, development, and testing processes.



Freelance Marketplace Domain Requirements

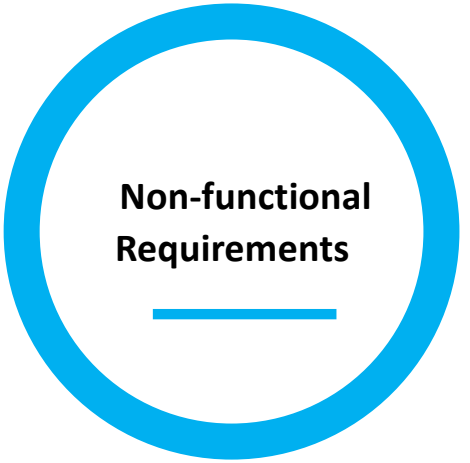
- **User Authentication**
  - Create accounts, login, password reset
- **User Profiles**
  - Create a seller's profile
  - Update profile
- **Search and Filters**
  - Search for gigs
- **Messaging System**
  - Buyers should be able to message sellers
- **Rating and Reviews**
  - Buyers can review sellers
  - Sellers can review buyers
- **Payment Gateway Integration**
- **View Orders**
  - Buyers and sellers can view **active**, **completed** and **cancelled** orders
- **Cancellation of Orders**



Non-functional Requirements

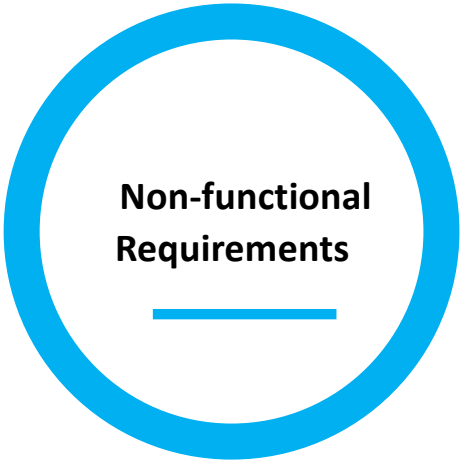
Non-functional Requirements

- Non-functional requirements describes how the system should perform, rather than what it should do.
- NFR are requirements that describes how the system works.
- They focus on the quality attributes and constraints that a system must meet.
- NFR are as equally important as functional requirements because they impact the user experience with the system.



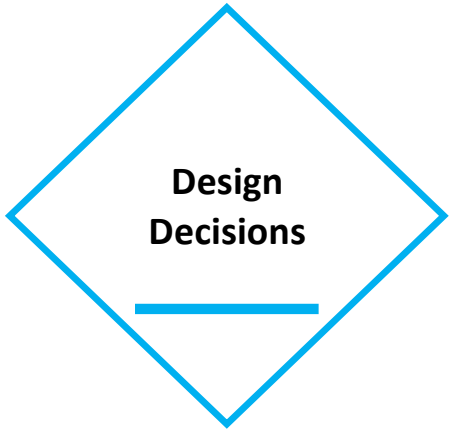
Freelance Marketplace Domain Requirements

- **Scalability**
  - The system should be able to scale to accommodate increased load during peak times.
- **Availability**
  - The system should be available 99.99% of the time
  - A failover mechanism should ensure high availability in case of server failures
- **Reliability**
  - The system should be dependable
- **Maintainability**
  - Code should follow coding standards and be well-documented
  - Regular code reviews and automated testing should be performed
- **Usability**
  - Users should find the system easy to use
- **Security, Performance, Capacity**



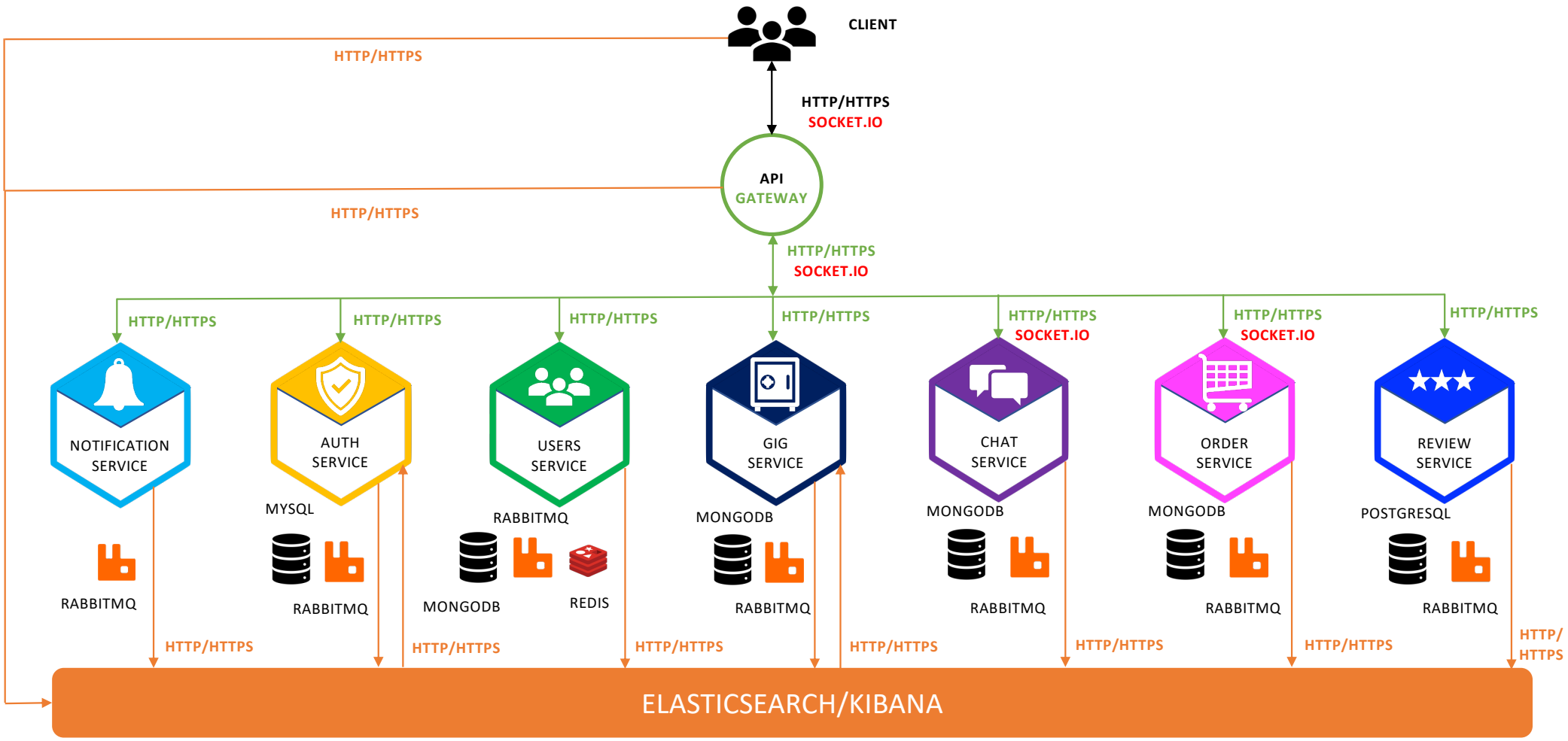
Design Decisions

- No direct client-to-microservice communication.
- All request from client must go through API gateway.
- Communication between API gateway and other microservices will be HTTP based and socket.io.
- Inter-process communication between microservices will only be event-driven. No HTTP request/response.
- Token generation and management will be handled by API gateway.
- All microservices except API gateway will not be accessible from outside.
- Every request from API gateway will include a token.
- Microservices will send client errors to API gateway but other errors will be sent to monitoring and logging system.



Project Architecture

Project Architecture



Inter-process Communication

Inter-Process Communication

Send	Receive
-	Auth
-	Order
-	Chat
-	-

Send	Receive
Notifi...	-
Users	-
-	-
-	-

Send	Receive
Gig	Auth
-	Order
-	Gig
-	Review

Send	Receive
Users	Users
-	-
-	-
-	-

Send	Receive
Notifi...	-
-	-
-	-
-	-

Send	Receive
Users	Review
Notifi...	-
-	-
-	-

Send	Receive
Users	-
Order	-
-	-
-	-

Send ≈ Producer

Receiver ≈ Consumer

Local Development Tools

Development tools

- VSCode
- Docker and Docker-Compose
- Rabbitmq
- Elasticsearch and Kibana
- Mongodb
- MySQL
- Postgresql
- Redis

Helper/Shared Library

Why do we need a helper library?

To avoid duplicate codes across microservices

Challenges

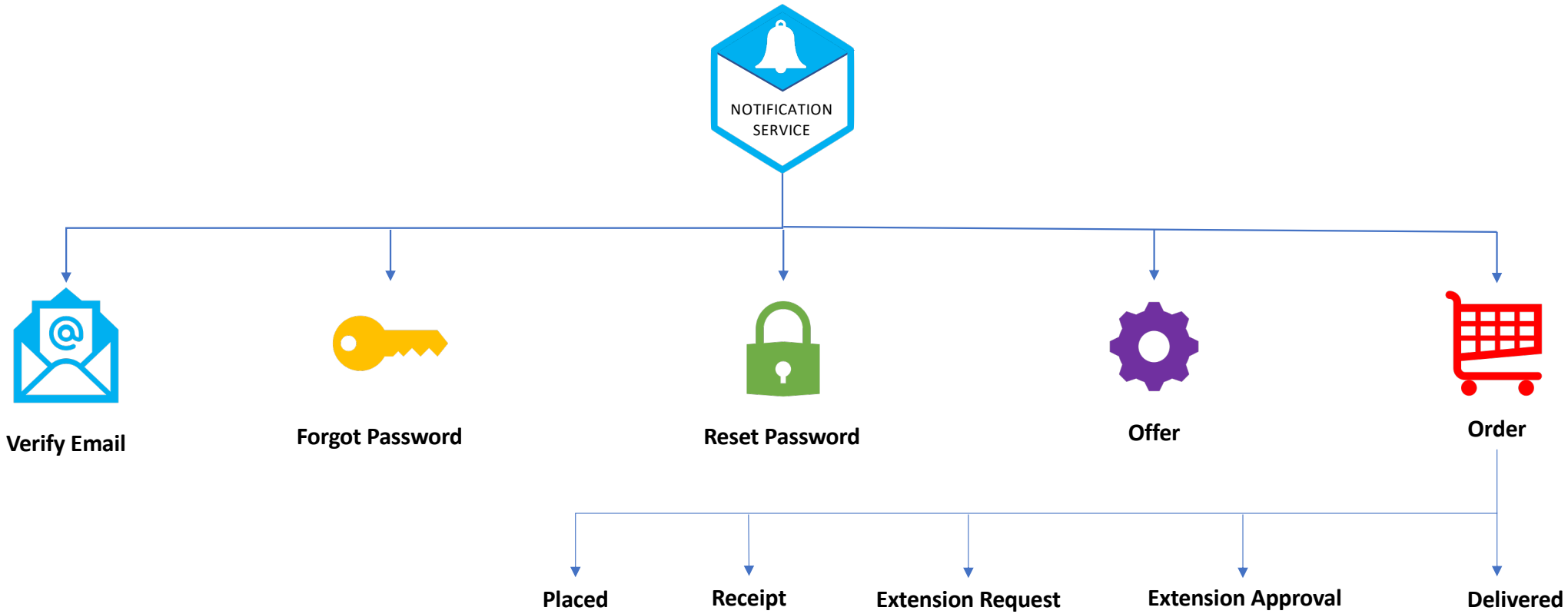
- If you need to update the library, you have to go through the process of updating, pushing and running pipeline.
- Updating all services that requires the new version
  - Best case scenerio is updating only one service
  - Worst case scenerio is updating all services

Helpers

- Interfaces
- Cloudinary upload methods
- Error handlers
- API gateway middleware
- Logger methods
- Helper methods

Notification Service

Notification Emails



Inter-Process Communication

Send	Receive
-	Auth
-	Order
-	Chat
-	-

Send	Receive
Notifi...	-
Users	-
-	-
-	-

Send	Receive
Gig	Auth
-	Order
-	Gig
-	Review

Send	Receive
Users	Users
-	-
-	-
-	-

Send	Receive
Notifi...	-
-	-
-	-
-	-

Send	Receive
Users	Review
Notifi...	-
-	-
-	-

Send	Receive
Users	-
Order	-
-	-
-	-

Send ≈ Producer

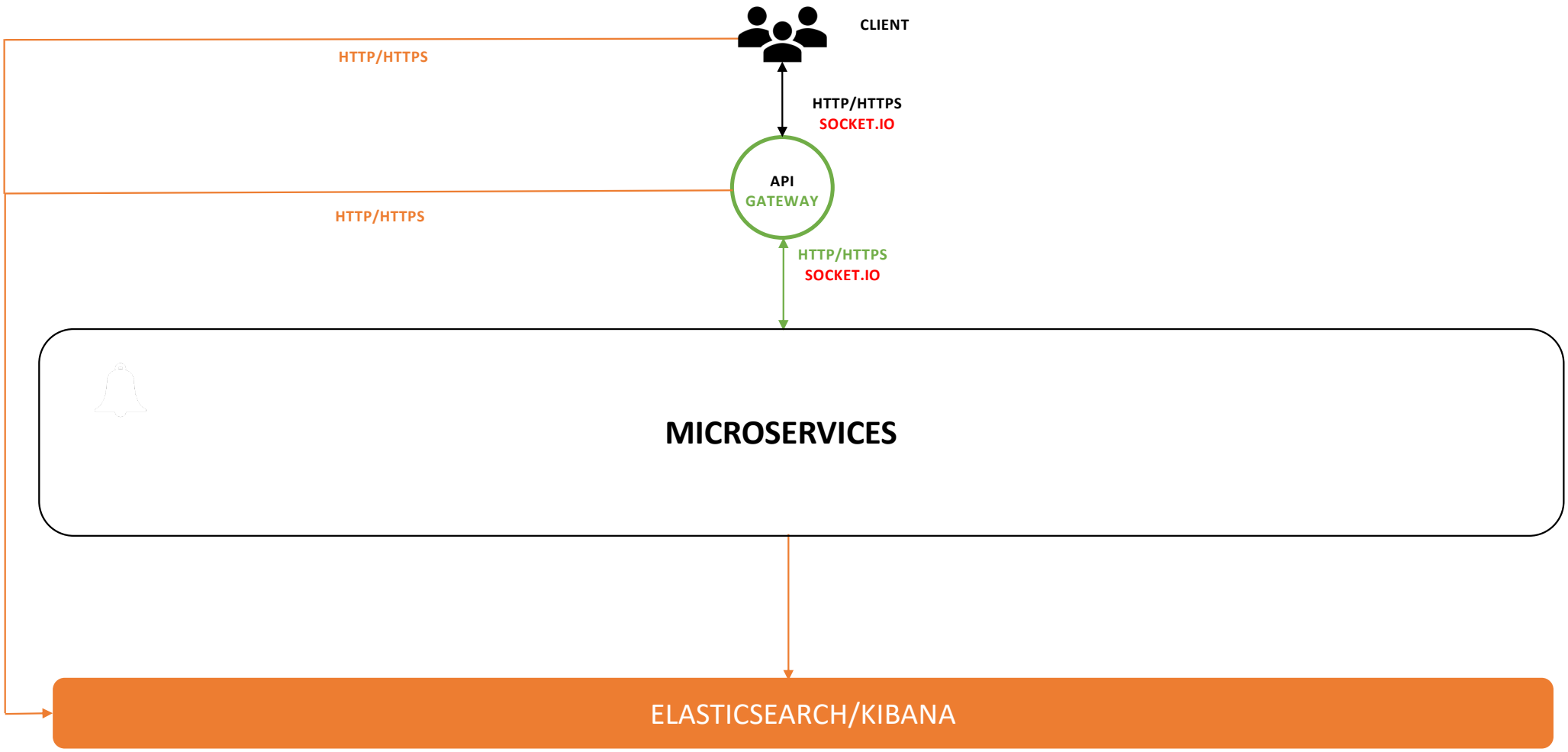
Receiver ≈ Consumer

RabbitMQ Channel Methods

- **channel.publish**
  - Publish a message to an exchange
- **channel.assertExchange**
  - Asserts an exchange into existence
- **channel.assertQueue**
  - Checks if a queue exists
  - If it does not exist, it creates a new queue
- **channel.bindQueue**
  - Assert a routing path from an exchange to a queue
- **channel.consume**
  - Set up a consumer with a callback to be invoked with each message
- **channel.ack**
  - Acknowledge the given message

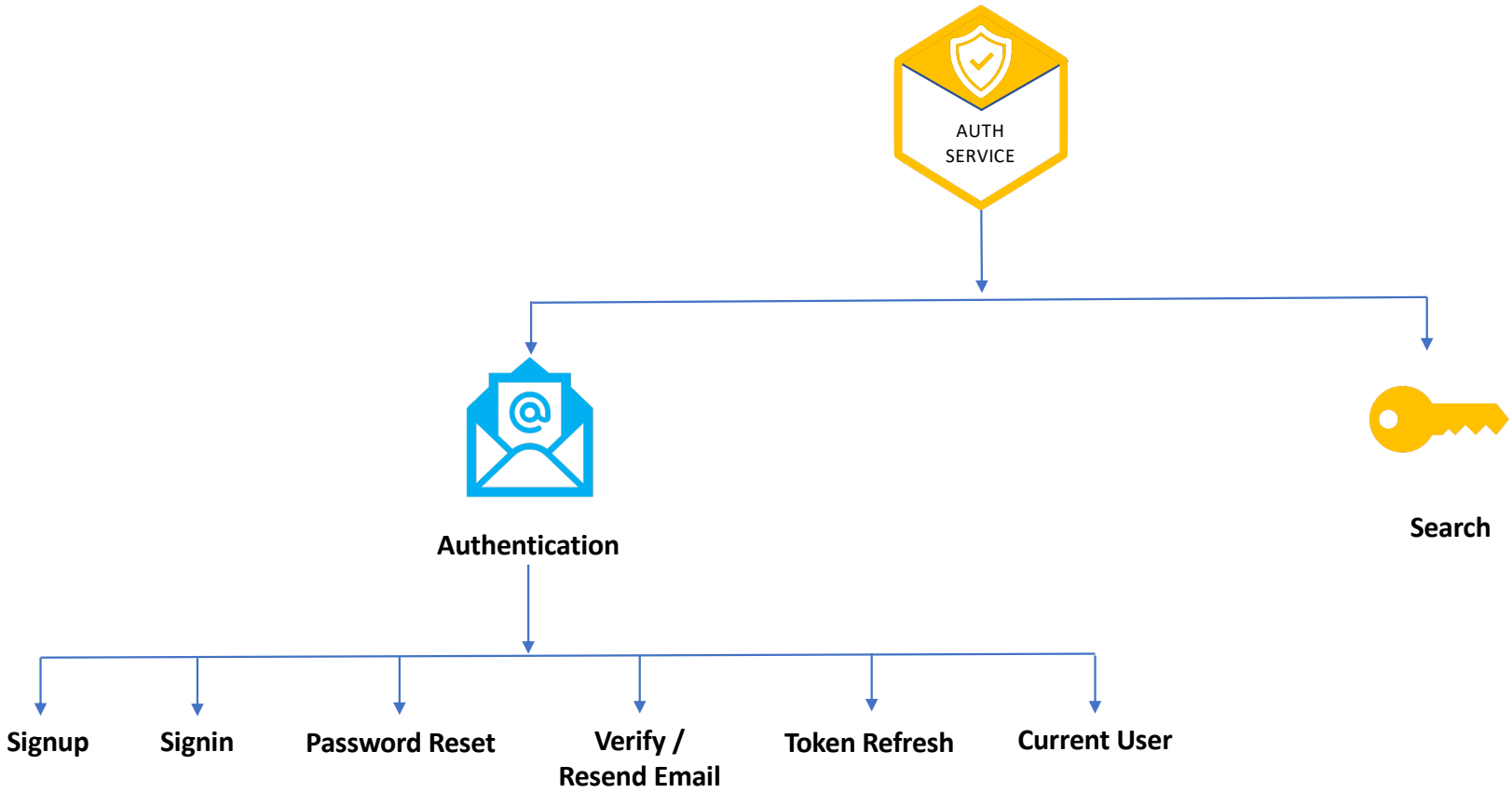
API Gateway Service

API Gateway


CLIENT
HTTP/HTTPS
HTTP/HTTPS
SOCKET.IO
API
GATEWAY
HTTP/HTTPS
HTTP/HTTPS
SOCKET.IO
MICROSERVICES
ELASTICSEARCH/KIBANA

Authentication Service

Auth Service Features



Auth Service Endpoints

Auth Service API Endpoints

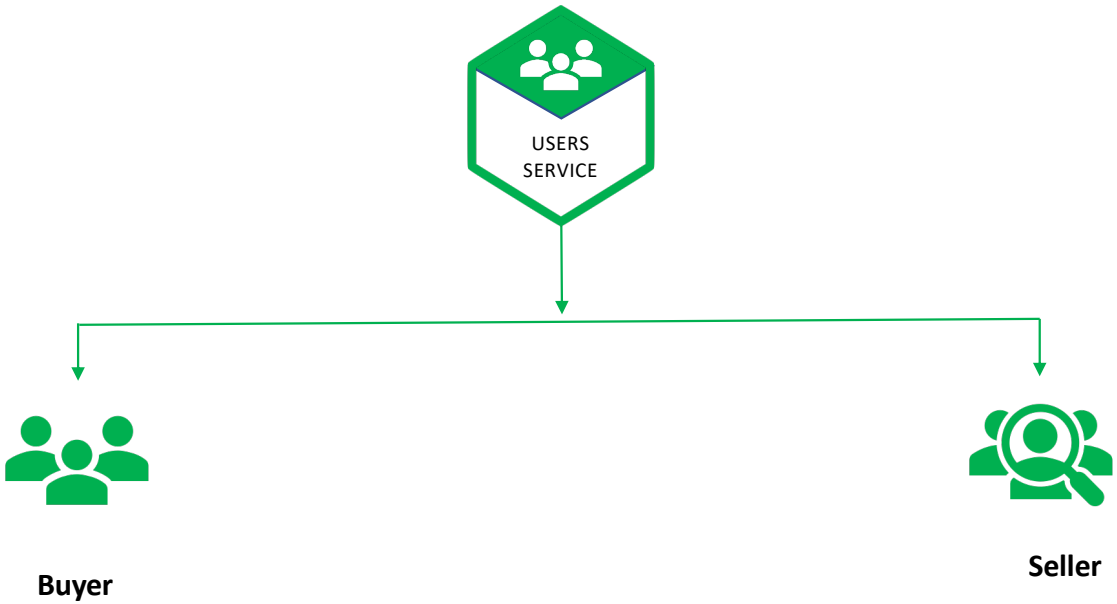
Frontend to API Gateway – http(s)://<host>:<port>/api/v1/gateway/auth

API Gateway to Auth Service – http(s)://<api-gateway-host>:<port>/api/v1/auth

Name	API Endpoint	Verb
Signup	/signup	POST
Signin	/signin	POST
Verify Email	/verify-email	PUT
Forgot Password	/forgot-password	PUT
Reset Password	/reset-password/:token	PUT
Change Password	/change-password	PUT
Current User	/currentuser	GET
Resend Email	/resend-email	POST
Seeding Data	/seed/:count	PUT
Search Gigs	/search/gig/:from/:size/:type	GET
Search Gig	/search/gig/:gigId	GET

Users Service

Users Service Features



Users Service API Endpoints (Buyer)

Frontend to API Gateway – http(s)://<api-gateway-host>:<port>/api/v1/gateway/buyer

API Gateway to Users Service – http(s)://<users-service-host>:<port>/api/v1/buyer

Name	API Endpoint	Verb
Buyer by Email	/email	GET
Buyer by Current Username	/username	GET
Buyer by Username	/:username	GET

Users Service API Endpoints (Seller)

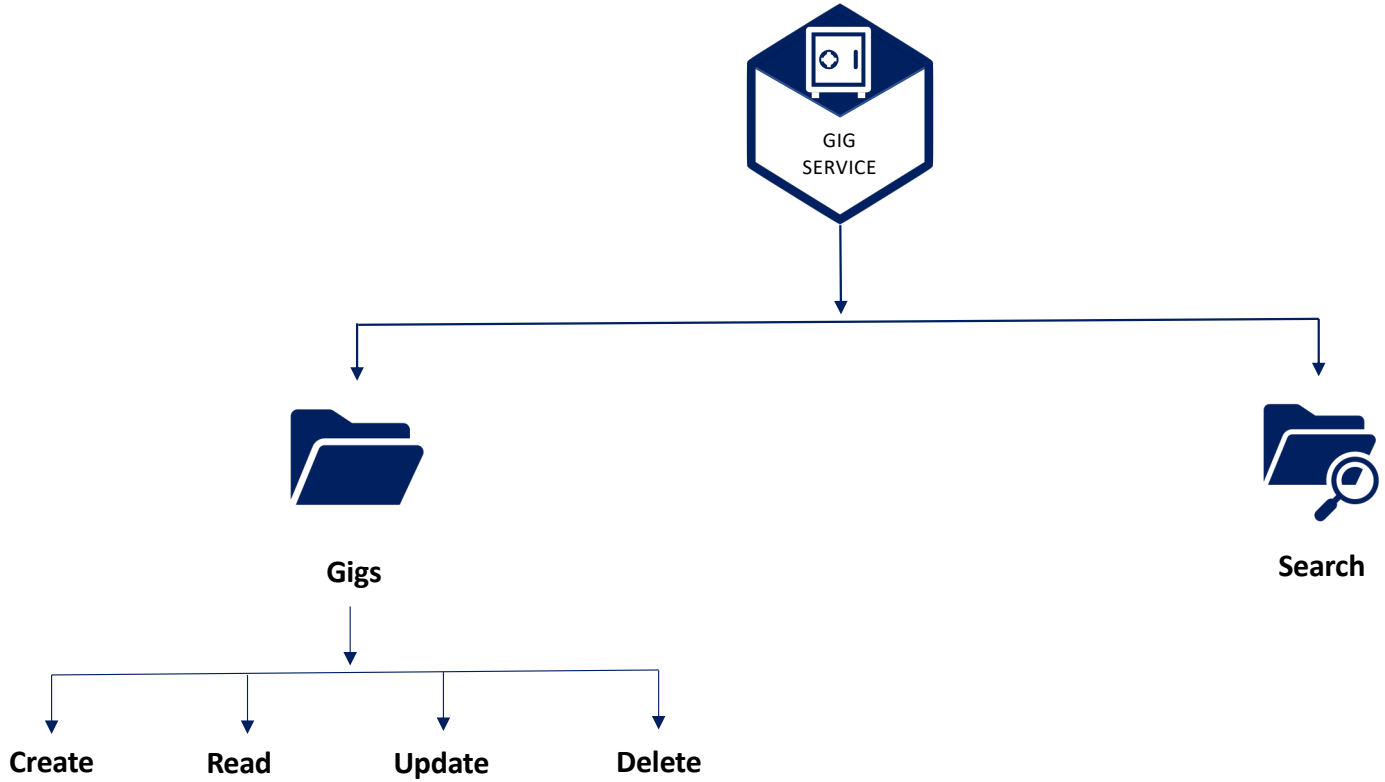
Frontend to API Gateway – http(s)://< api-gateway-host>:<port>/api/v1/gateway/seller

API Gateway to Users Service – http(s)://<users-service-host>:<port>/api/v1/seller

Name	API Endpoint	Verb
Seller by Id	/id/:sellerId	GET
Seller by Username	/username/:username	GET
Random Sellers	/random	GET
Create Seller	/create	POST
Update Seller	/:sellerId	PUT
Seeding Seller	/seed/:count	PUT

Gig Service

Gig Service Features



Gig Service API Endpoints

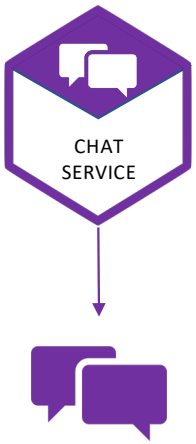
Frontend to API Gateway – http(s)://<api-gateway-host>:<port>/api/v1/gateway/gig

API Gateway to Gig Service – http(s)://<gig-service-host>:<port>/api/v1/gig

Name	API Endpoint	Verb
Gig by Id	/:gigId	GET
Seller Gigs	/seller/:sellerId	GET
Seller's Inactive Gigs	/seller/pause/:sellerId	GET
Gigs By Category	/category/:username	GET
Top Rated Gigs	/top/:username	GET
Similar Gigs	/similar/:gigId	GET
Create	/create	POST
Update	/:gigId	PUT
Update Active Gig	/active/:gigId	PUT
Delete Gig	/:gigId/:sellerId	DELETE
Seed	/seed/:count	PUT
Search	/search/:from/:size/:type	GET

Chat Service

Chat Service Features



Buyer/Seller Communication

Chat Service API Endpoints

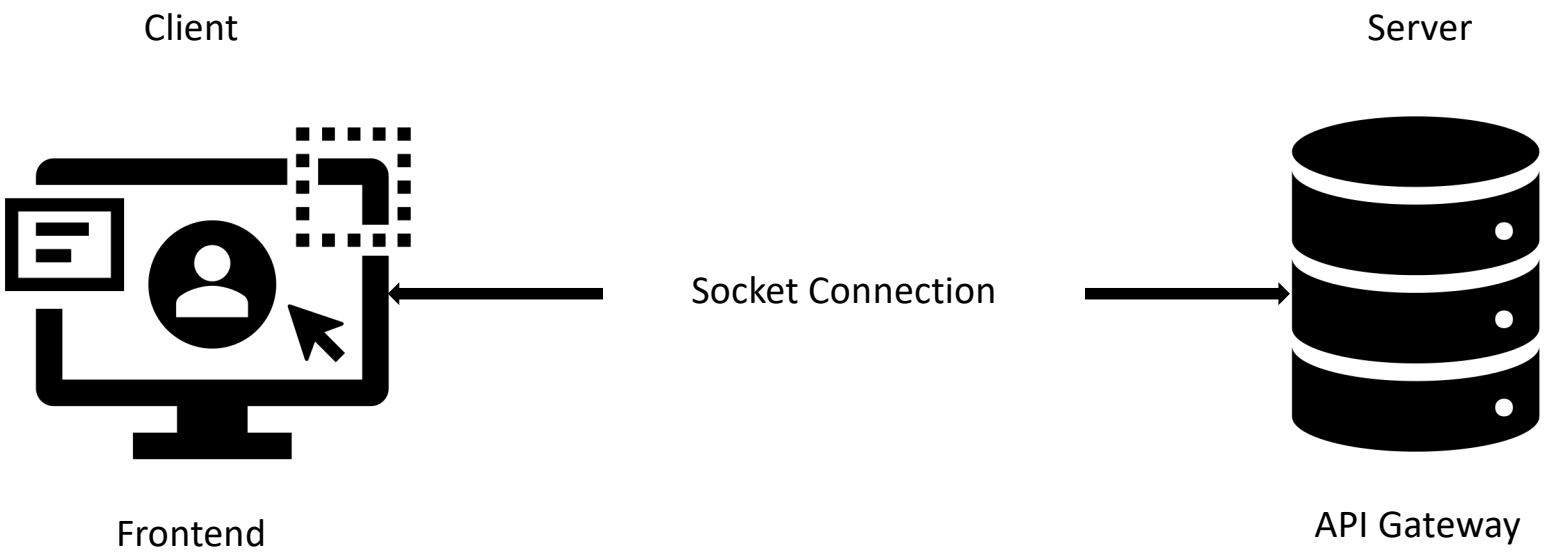
Frontend to API Gateway – http(s)://<api-gateway-host>:<port>/api/v1/gateway/message

API Gateway to Chat Service – http(s)://<chat-service-host>:<port>/api/v1/message

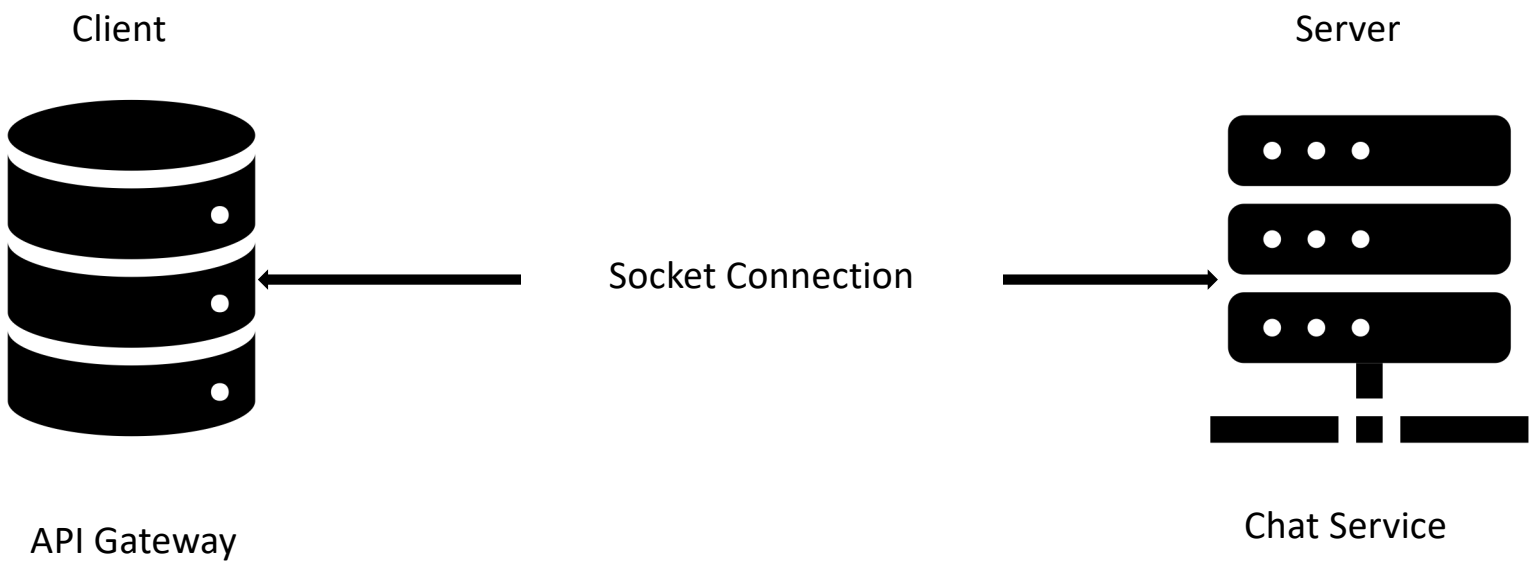
Name	API Endpoint	Verb
Get Conversation	/conversation/:senderUsername/:receiverUsername	GET
Messages by Username	/conversation/:username	GET
Messages by Sender Name and Receiver Name	/:senderUsername/:receiverUsername	GET
Messages by Conversation ID	/:conversationId	GET
Create Message	/	POST
Update Custom Offer	/offer	PUT
Mark Message as Read	/mark-as-read	PUT
Mark Multiple Messages as Read	/mark-multiple-as-read	PUT

Chat Service Socket.io Connection

Frontend to API Gateway Connection

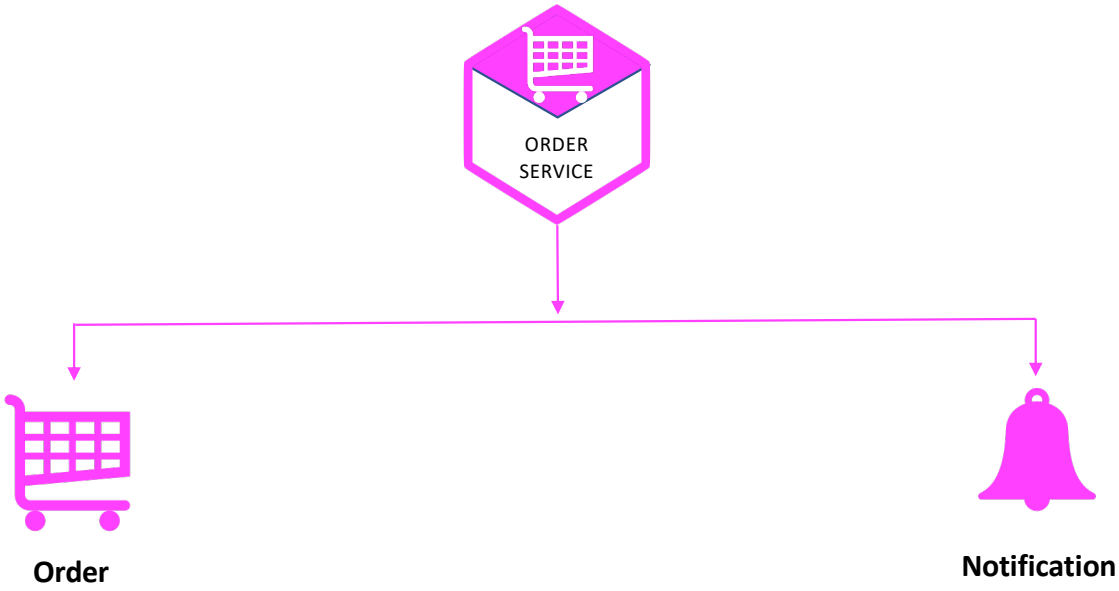


API Gateway to Chat Service Connection



Order Service

Order Service Features



Order Service API Endpoints

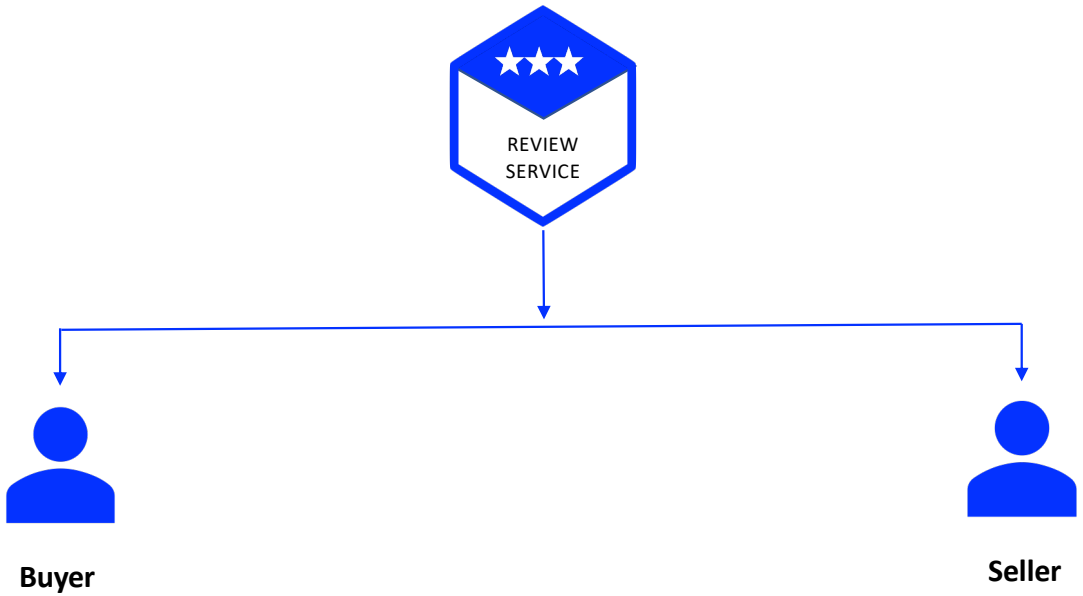
Frontend to API Gateway – http(s)://<api-gateway-host>:<port>/api/v1/gateway/order

API Gateway to Order Service – http(s)://<order-service-host>:<port>/api/v1/order

Name	API Endpoint	Verb
Order by Id	/:orderId	GET
Seller Orders	/seller/:sellerId	GET
Buyer Orders	/buyer/:buyerId	GET
Create Order	/	POST
Create Payment Intent	/create-payment-intent	POST
Cancel Order	/cancel/:orderId	PUT
Extension Request	/extension/:orderId	PUT
Extension Approval	/gig/:type/:orderId	PUT
Deliver Order	/deliver-order/:orderId	PUT
Approve Order	/approve-order/:orderId	PUT
Get Notifications	/notification/:userTo	GET
Update Notification	/notification/mark-as-read	PUT

Review Service

Review Service Features



Review Service API Endpoints

Frontend to API Gateway – http(s)://<api-gateway-host>:<port>/api/v1/gateway/review

API Gateway to Review Service – http(s)://<review-service-host>:<port>/api/v1/review

Name	API Endpoint	Verb
Reviews by Gig Id	/gig/:gigId	GET
Reviews By Seller Id	/seller/:sellerId	GET
Create Review	/	POST

Kubernetes Fundamentals

What is Kubernetes?

Kubernetes is a portable, extensible, opensource platform for managing containerized workloads and services.

What can Kubernetes do

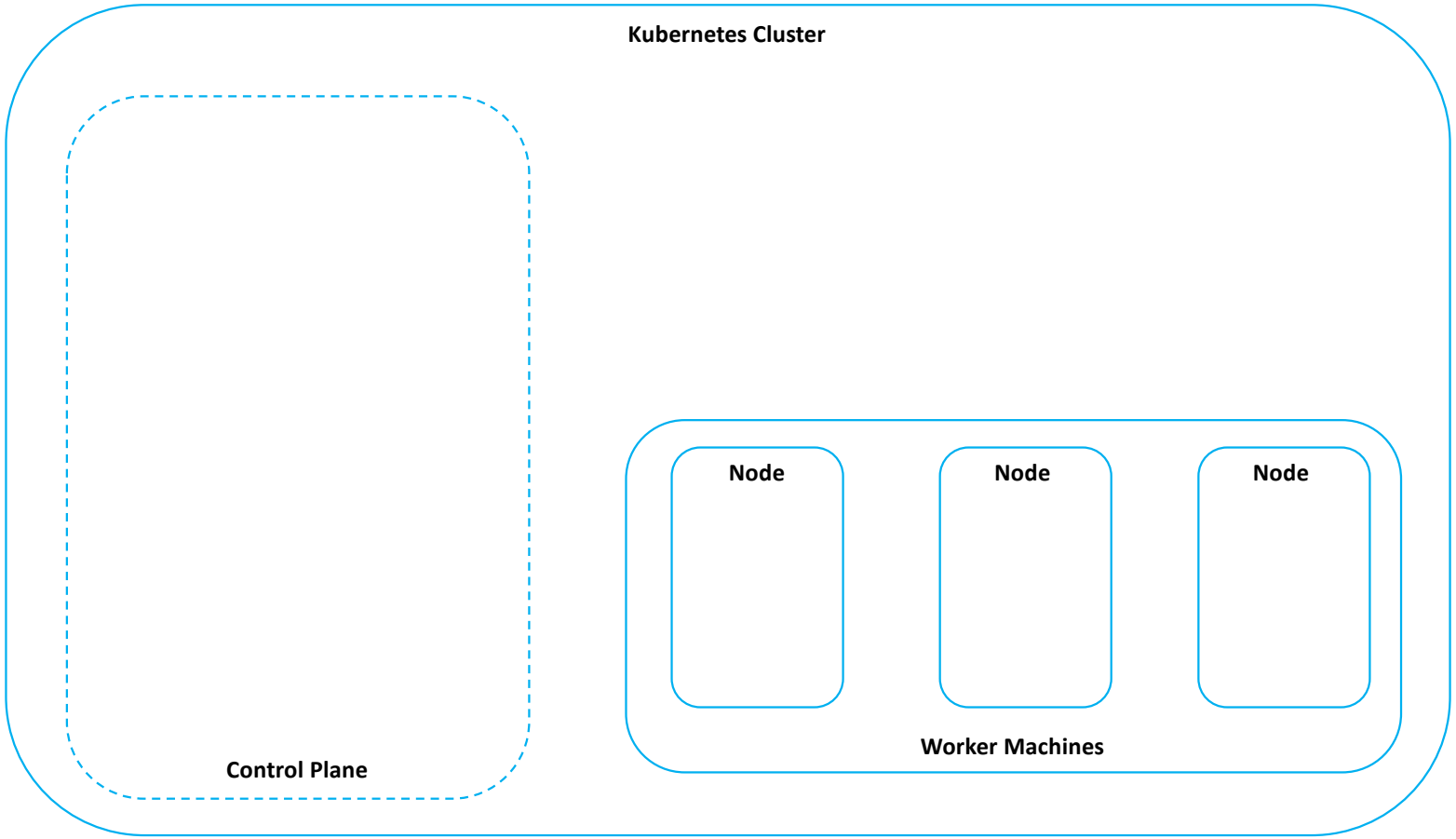
- Service discovery and load balancing
- Storage orchestration
- Automated rollouts and rollbacks
- Automatic bin packing
- Self-healing
- Secret and configuration management
- Horizontal scaling
- IPv4/IPv6 dual-stack

What Kubernetes is not

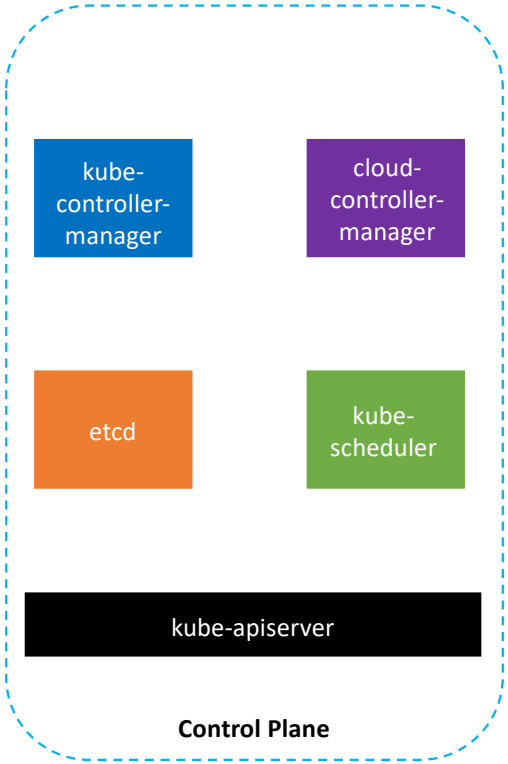
- Kubernestes does not deploy source code and does not build your application. It is not used for CI/CD workflows.
- Kubernetes does not provide application-level services, such as middleware, databases, caches as built-in services.
- Kubernetes does not dictate logging, monitoring, or alerting solutions.
- Kubernetes does not provide nor mandate a configuration language/system. It provides a declarative API.
- Kubernetes does not limit the types of applications supported.

Kubernetes Components

Kubernetes Cluster

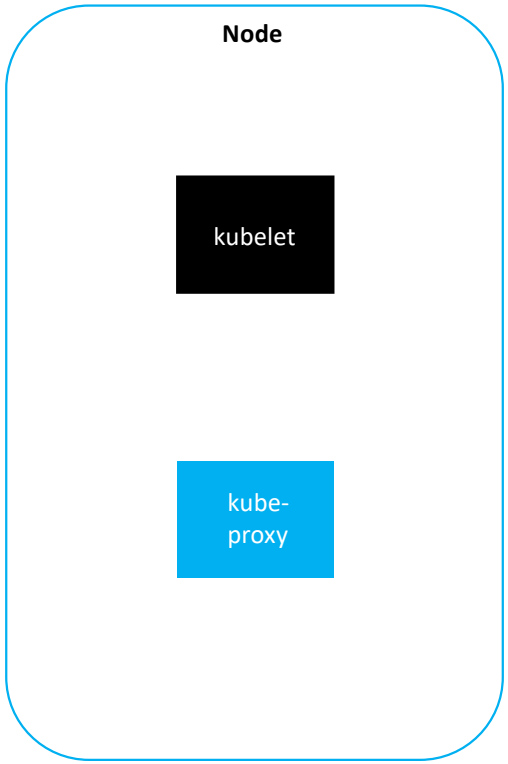


Control Plane Components



- kube-apiserver
  - Exposes the Kubernetes API.
  - It is the front end for the kubernetes control plane.
- etcd
  - Consistent and highly-available key-value store for all cluster data.
- kube-scheduler
  - Watches for newly created pods with no assigned node, and selects a node for them to run on.
- kube-controller-manager
  - Responsible for running multiple controllers.
  - Monitors the current state of the cluster and changes the current state to the desired state.
- cloud-controller-manager
  - Component that embeds cloud-specific control logic.
  - It lets you link your cluster into your cloud provider's API.

Node Components



- kubelet
  - Kubelet is the agent that runs on every node in the cluster.
  - The agent is responsible for making sure that containers are running in a Pod and healthy.
- kube-proxy
  - Kube-proxy is the network proxy that runs on each node in your cluster.
  - They maintain network rules on nodes.
  - These network rules allow network communication to your Pods inside or outside your cluster

Kubernetes Objects

Pod

- Pods are the smallest unit of deployment in Kubernetes.
- They are used to deploy, scale, and manage containerized applications in a cluster

Deployment

- Deployment objects are used to manage the lifecycle of one or more identical Pods.
- A Deployment allows you to declaratively manage the desired state of your application.

ReplicaSets

- In Kubernetes, Deployments don’t manage Pods directly.
- The ReplicaSet ensures that the desired number of replicas (copies) are running at all times by creating or deleting Pods as needed.

StatefulSet

- A StatefulSet is a Kubernetes object that is used to manage stateful applications.
- The StatefulSet ensures that each Pod is uniquely identified by a number, starting at zero.
- This allows for the preservation of state and data across Pod replacements.

DaemonSet

- A DaemonSet ensures that a copy of a Pod is running across all, or a subset of nodes in a Kubernetes cluster.
- They are useful for running system-level services, such as logging or monitoring agents.

Service

- A Kubernetes Service is a method for exposing a network application that is running as one or more Pods in your cluster.
- Types of Service
  - ClusterIP
  - NodePort
  - LoadBalancer
  - ExternalName
    - Maps a service to a DNS name
- Ingress
  - Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster.

PersistentVolume

- PersistentVolume represents a piece of storage you can attach to a Pod.

Namespaces

- A Kubernetes namespace is a way to isolate groups of resources in a single cluster.

ConfigMaps & Secrets

- They both allow for the configuration of apps that run in your Pods.
- **ConfigMaps** are used to store non-sensitive data in key-value pairs.
- **Secrets** are meant to hold sensitive data.

Service Account

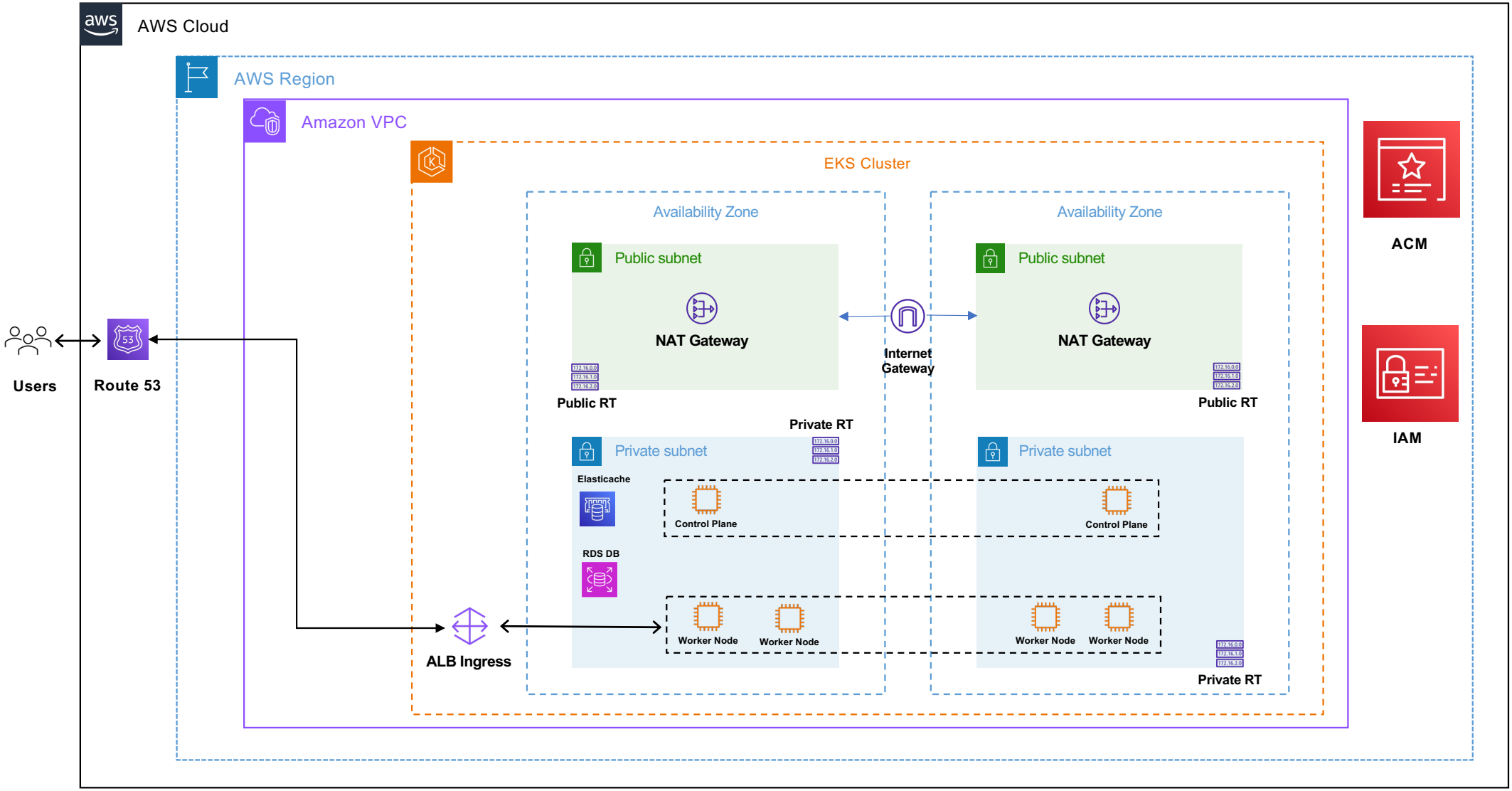
- A serivce account in Kubernetes provides a distinct identity in a Kubernetes cluster.

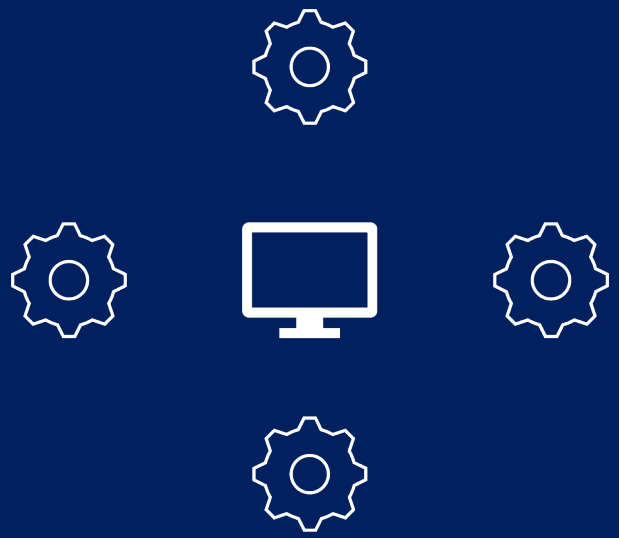
Minikube vs. Cloud Resources

	Minikube	AWS
Redis	Managed	Cloud
RabbitMQ	Managed	Managed
MySQL	Managed	Cloud
Postgresql	Managed	Cloud
MongoDB	Managed	Cloud
Elasticsearch	Managed	Managed & Cloud
Kibana	Managed	Managed & Cloud

Microservices Architecture on Amazon EKS

Microservices Architecture on Amazon EKS





Microservices

with

NodeJS, React, Typescript and Kubernetes

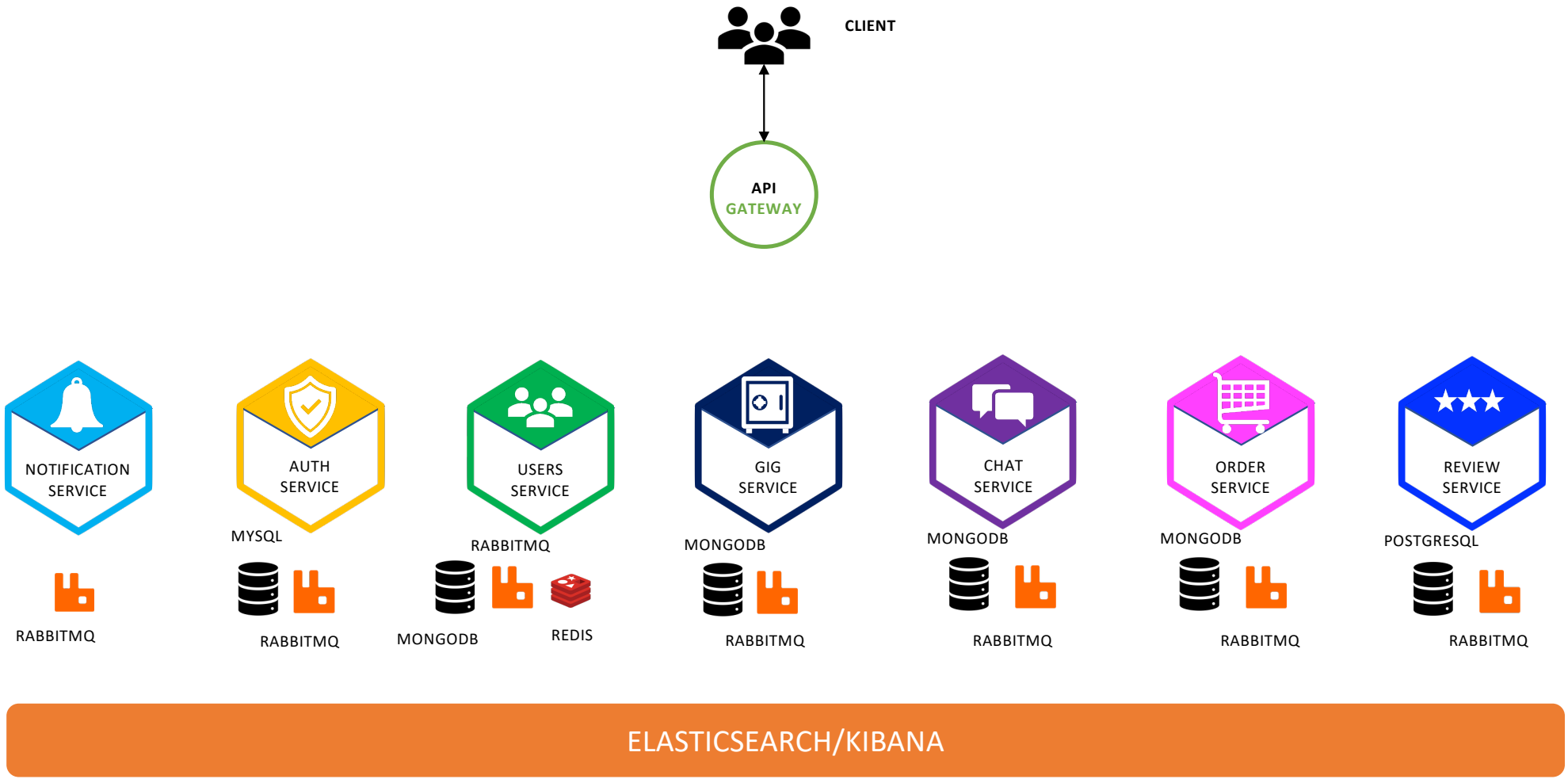
What are you going to build?

A fullstack e-commerce freelance marketplace application

At the end of the course

- Understand the basics of microservices architecture
- Build event-driven microservices using NodeJS, Express and Typescript
- Use Database-per-service pattern
- Setup communication styles with both Request/Response pattern and Event-driven pattern
- Setup single node kubernetes cluster with Minikube and multi-node kubernetes cluster with EKS
- Use Docker and Kubernetes to deploy multiple microservices either locally with Minikube or to the cloud with AWS EKS

Project Architecture



Hope you take this course.