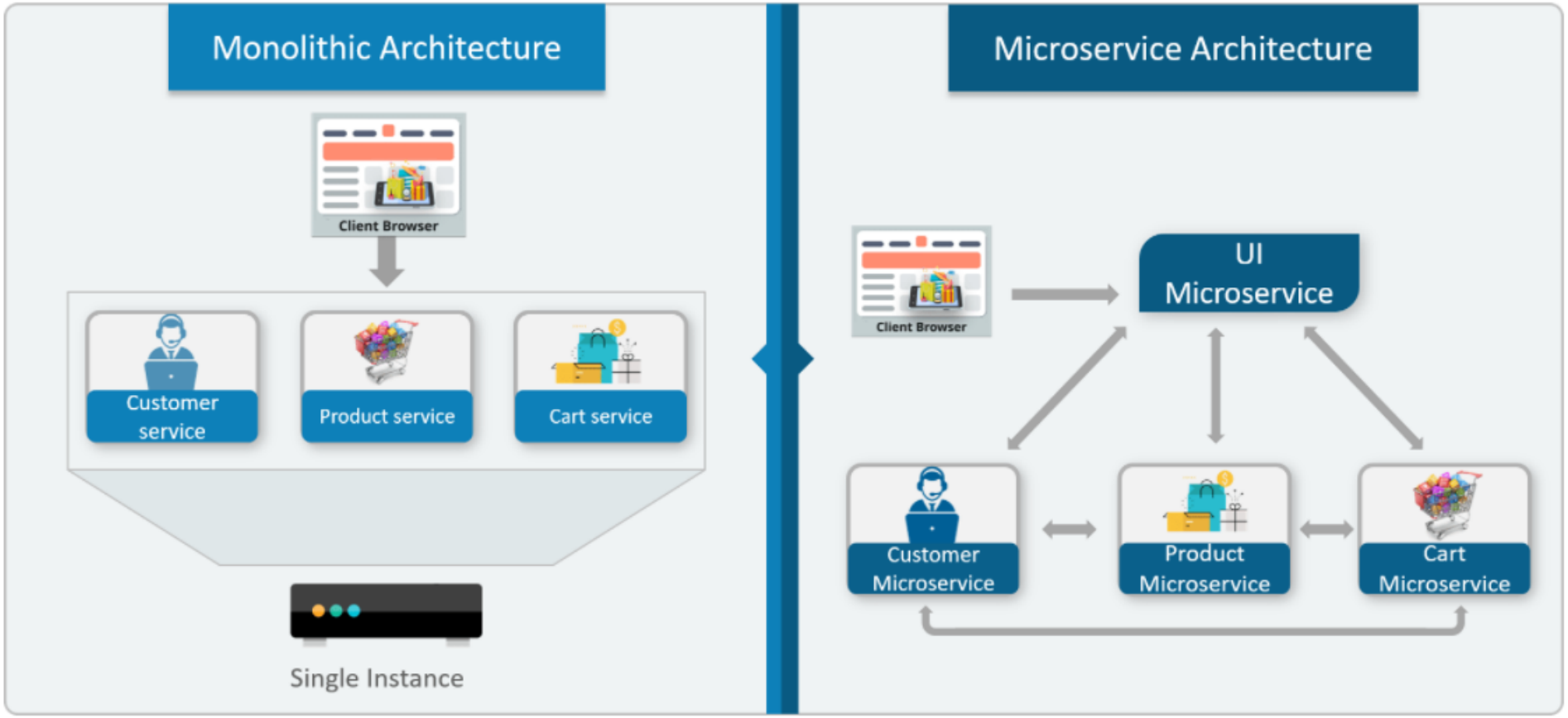
Limitations of Monolithic Architecture

- **Inflexible –** Monolithic applications cannot be built using different technologies
- **Unreliable –** Even if one feature of the system does not work, then the entire system does not work
- **Unscalable –** Applications cannot be scaled easily since each time the application needs to be updated, the complete system has to be rebuilt
- **Blocks Continuous Development –** Many features of the applications cannot be built and deployed at the same time
- **Slow Development –** Development in monolithic applications take lot of time to be built since each and every feature has to be built one after the other
- **Not Fit For Complex Applications –** Features of complex applications have tightly coupled dependencies

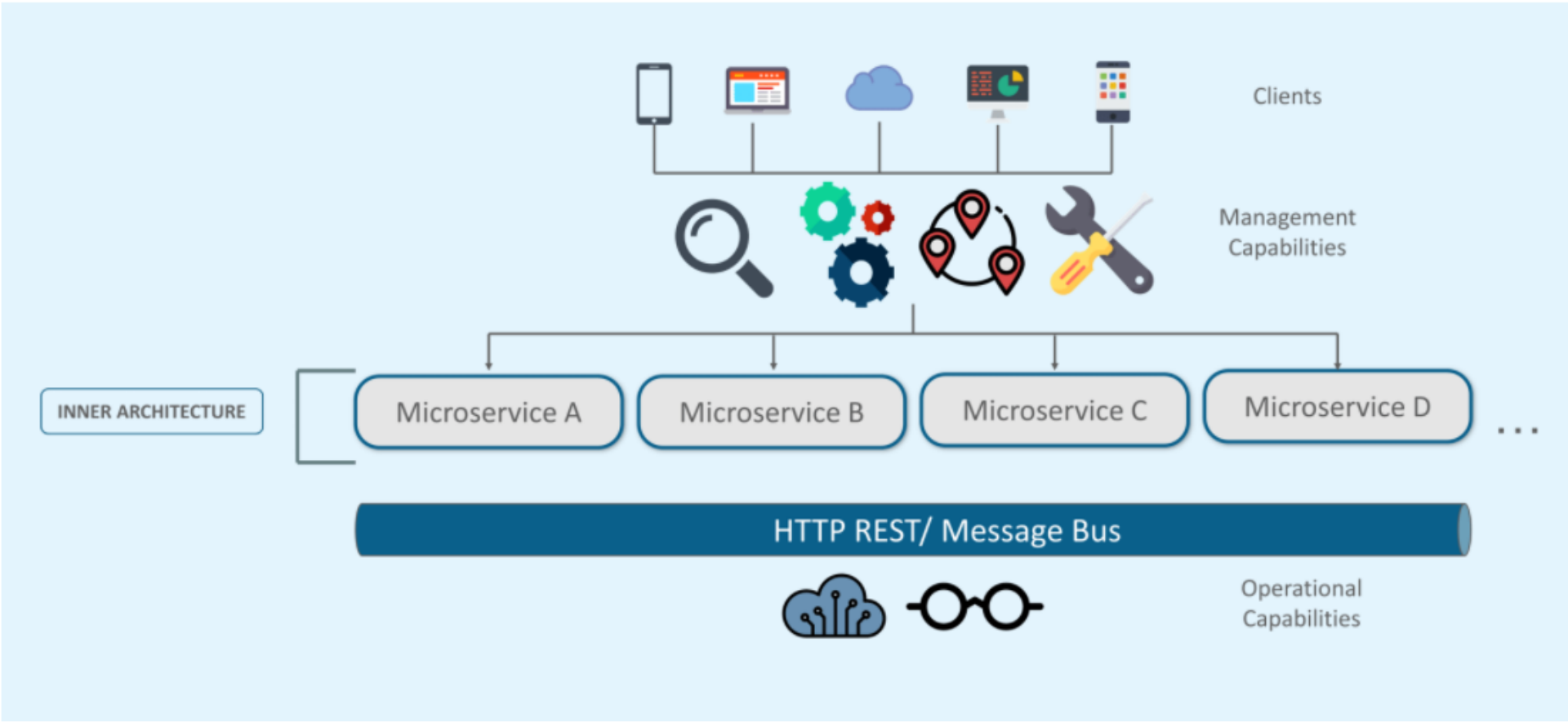
Difference



https://www.edureka.co/blog/what-is-microservices/

https://www.edureka.co/blog/microservices-tutorial-with-example

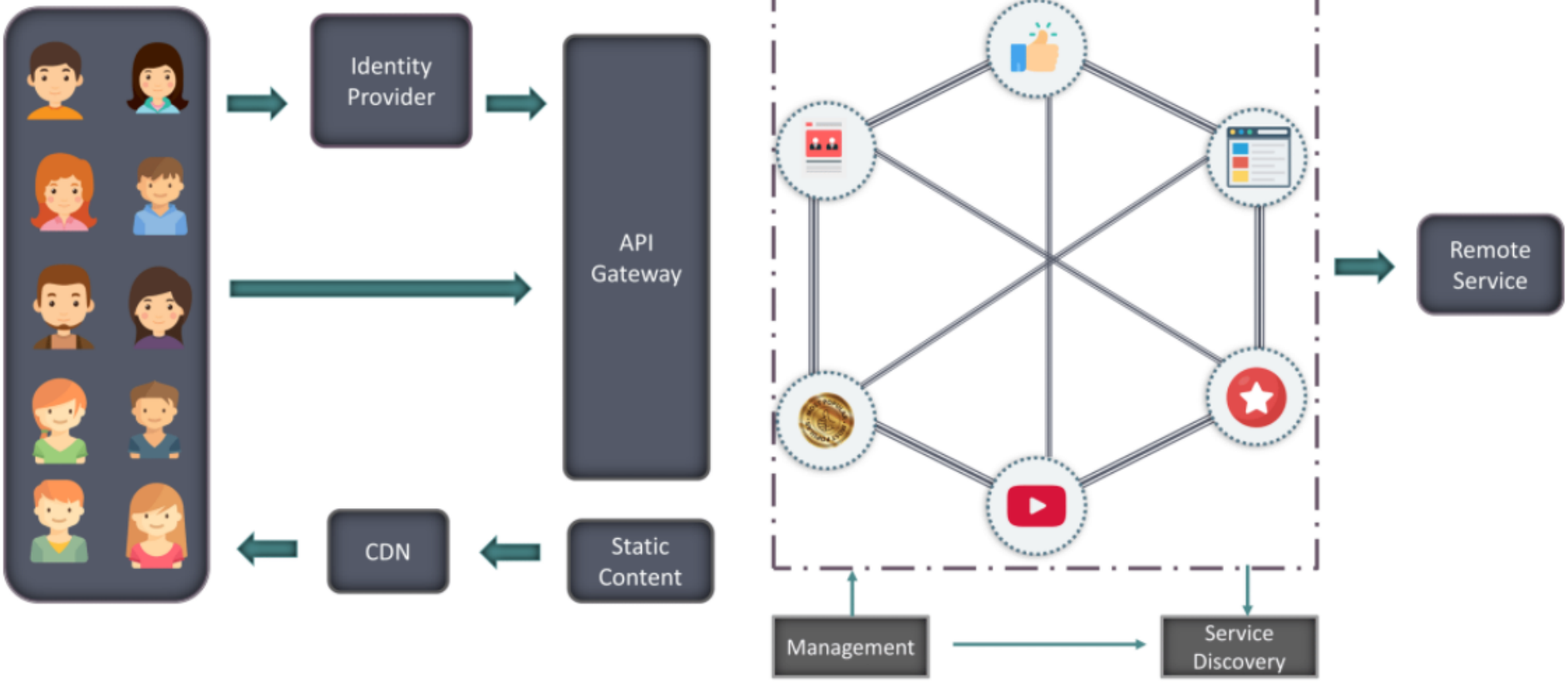
Detail Architecture



Details of Microservice

- Different clients from different devices try to use different services like search, build, configure and other management capabilities
- All the services are separated based on their domains and functionalities and are further allotted to individual microservices
- These microservices have their own **load balancer** and **execution environment** to execute their functionalities & at the same time captures data in their own databases
- All the microservices communicate with each other through a stateless server which is either **REST** or **Message Bus**
- Microservices know their path of communication with the help of **Service Discovery** and perform operational capabilities such as automation, monitoring
- Then all the functionalities performed by microservices are communicated to clients via **API Gateway**
- All the internal points are connected from the API Gateway. So, anybody who connects to the API Gateway automatically gets connected to the complete system

Deployment Architecture



Description of components

- **Clients –** Different users from various devices send requests.
- **Identity Providers –** Authenticates user or clients identities and issues security tokens.
- **API Gateway –** Handles client requests.
- **Static Content –** Houses all the content of the system.
- **Management –** Balances services on nodes and identifies failures.
- **Service Discovery –** A guide to find the route of communication between microservices.
- **Content Delivery Networks –** Distributed network of proxy servers and their data centers.
- **Remote Service –** Enables the remote access information that resides on a network of IT devices.

Microservice Features

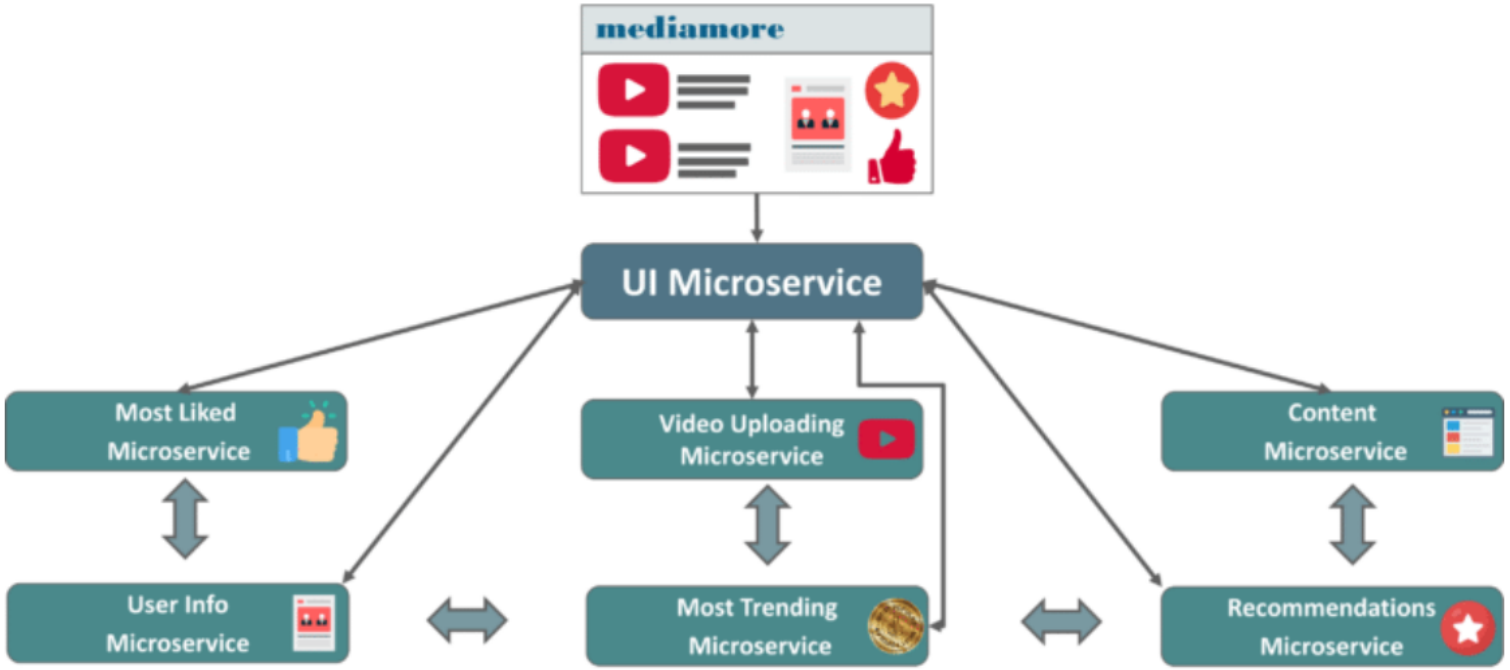
- **Decoupling** – Services within a system are largely decoupled. So the application as a whole can be easily built, altered, and scaled
- **Componentization** – Microservices are treated as independent components that can be easily replaced and upgraded
- **Business Capabilities** – Microservices are very simple and focus on a single capability
- **Autonomy** – Developers and teams can work independently of each other, thus increasing speed
- **Continous Delivery** – Allows frequent releases of software, through systematic automation of software creation, testing, and approval
- **Responsibility** – Microservices do not focus on applications as projects. Instead, they treat applications as products for which they are responsible
- **Decentralized Governance** – The focus is on using the right tool for the right job. That means there is no standardized pattern or any technology pattern. Developers have the freedom to choose the best useful tools to solve their problems
- **Agility** – Any new feature can be quickly developed and discarded again

Advantages

- **Independent Development –** All microservices can be easily developed based on their individual functionality
- **Independent Deployment –** Based on their services, they can be individually deployed in any application
- **Fault Isolation –** Even if one service of the application does not work, the system still continues to function
- **Mixed Technology Stack –** Different languages and technologies can be used to build different services of the same application
- **Granular Scaling –** Individual components can scale as per need, there is no need to scale all components together

Example Scenario

- Alice is an avid user of mediamore. She uses mediamore regularly to watch her favorite series online. She recently missed watching an episode of her favorite TV show.
- When Alice logs in to the application, she sees the most recommended content on her home page. After some searching, she finally finds her TV Show.
- What if Alice wants to get her TV Show with a single click.



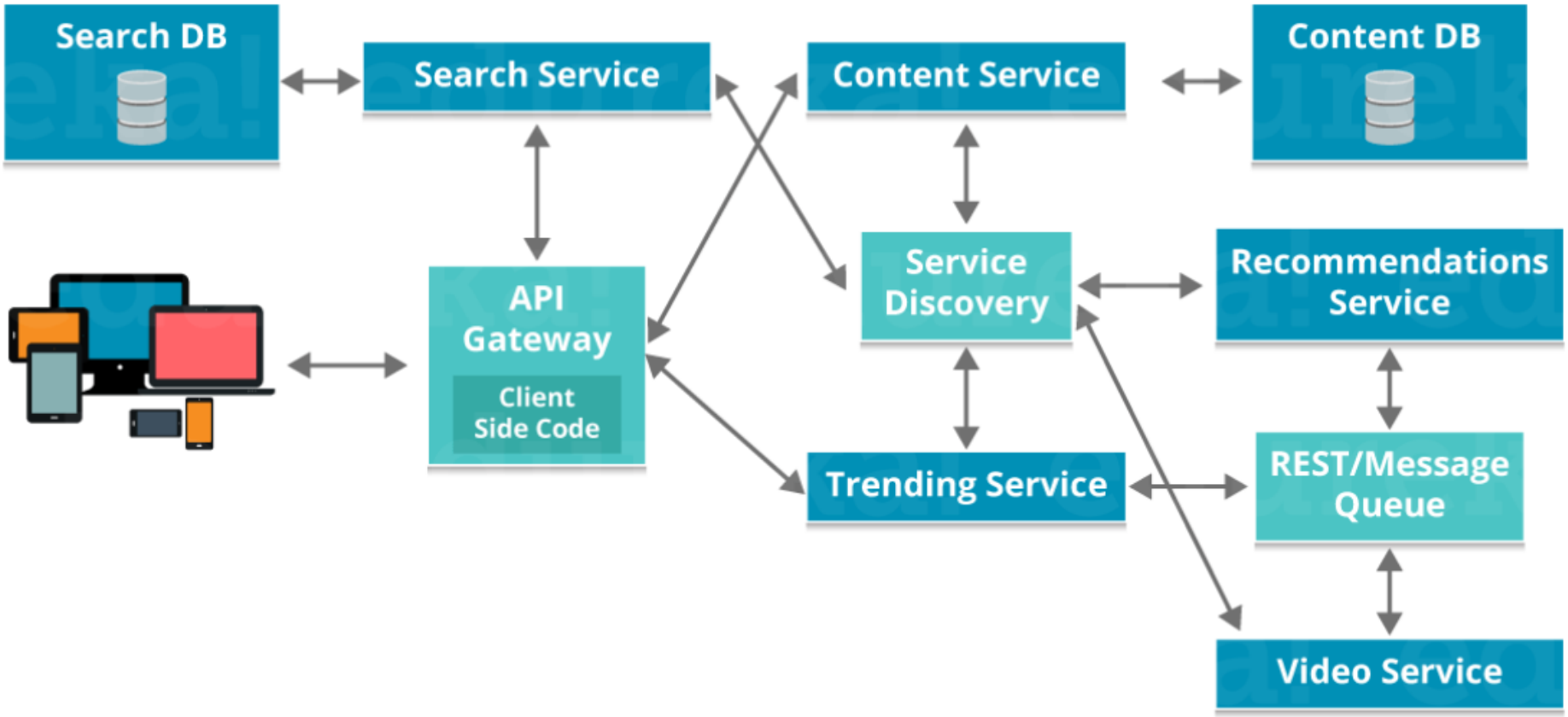
How will the developers work together to fulfill Alice's request?

- Alice's request is passed on to the **Identity Provider**. Identity provider thus authenticates Alice's request by identifying her as a regular user on mediamore.
- These requests are passed to the **API Gateway** which acts as an entry point for Alice to forward her requests to the appropriate microservices.
- Each feature has its own working microservice, handling their own data. These microservices also have their own **load balancers** and **execution environments** to function properly.
- Each microservice is handled by a small agile team such as content team, video uploading team, most trending team, search team etc.
  - The content team consists of millions of TV Shows that the application provides.
  - The video uploading team have the responsibility to upload all the content into the application
  - The most trending team houses the most trending shows according to the geographical location of users and so on.

Cont.

- These small teams of developers relate each and every piece of content with the metadata that describes the searched content.
- Then, metadata is fed into another microservice i.e. the search function which ensures Alice's search results are captures into the content catalog.
- Then, the third microservice i.e. most trending microservice captures the trending content among all the mediamore users according to their geographical locations.
- The content from this microservice is what Alice sees when she first logs into mediamore.
- These individually deployable microservices are put in specific **containers** to join the application. Containers are used to deliver the code to the sector where deployment is required.
- But before they join the application to work together, they have to find each other to fulfil Alice's request.

How do these microservices find each other?



These microservices communicate with each other using an **Application Program Interface(API)**. After the Microservices communicate within themselves, they deploy the **static content** to a cloud-based storage service that can deliver them directly to the clients via **Content Delivery Networks (CDNs)**.

- So, when Alice searches for her TV Show, the search microservice communicates with the content catalog service in API about what is Alice searching for and then these microservices compare the typed words with the metadata they already have.
- Once the teams of developers capture the most typed words by Alice, the analytics team update the code in recommendations microservice and compare Alice's most viewed content and preferences to popular content among other users in the same geographical region.
- This means that the next time Alice logs on to the application, she not only sees the most popular content but also finds a personalized playlist which contains the shows she has previously viewed.
- In this way, Alice's request is fulfilled by the development team in a quick manner as they did not have to build the complete application again and just had to update the code to deploy this new functionality.
- See an example: https://www.youtube.com/watch?v=gfWr2_H39N0