Internal Communication in Microservices

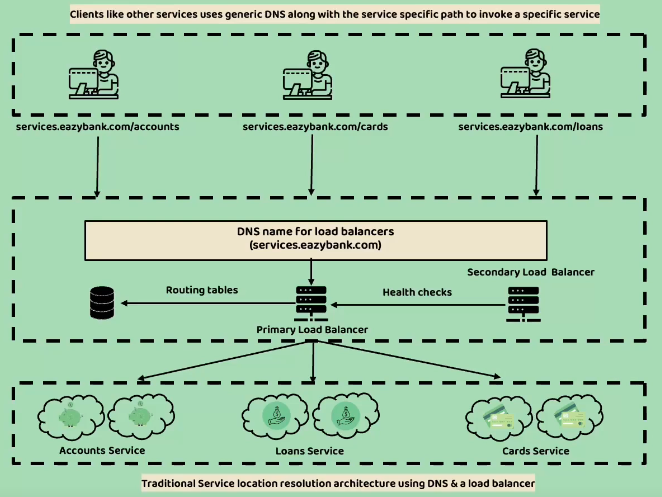
# Introduction

* In the microservices architecture, we will have a microservices network inside which the microservices talk to each other
* External traffic to the application will enter this network through an API gateway which can be responsible for all sorts of tasks like authentication, auditing and logging
* In this document we will be focusing on how the service talk to each other
* In order for this internal communication to work, services need to discover other services and also register themselves in the network
* In this internal communication set-up, we will face 3 challenges:
* **Service Discovery**: as the run-time services are instances of a docker image for example, they can have different Ips for example and they may have been destroyed at the moment another service tries to call them. So, services need to be able to handle this dynamic nature of the running services
* **Service Registration**: The new instances should introduce themselves to the network so other services can use them
* **Load Balancing**: As there are multiple instances of each service the traffic should be balanced between these constantly changing instances

# What Happens if We Use Traditional Approaches for Internal Communication in a Microservices Architecture

## The Traditional Approach

* Let’s say we have a single instance of an upstream service that is reached by the client and it uses a downstream service that is also a one-instance service to process the requests
* In this scenario we can hardcode the IP or DNS address of the downstream service in the upstream service which has some level of flexibility given the fact that we can use a DNS address and map different IPs to this domain without the need to change anything on the upstream service
* But as we have multiple instances of a services so we need to map the IPs of the instances to the domain name and use algorithms like the **round-Robin scheduling** which will schedule the requests to a domain name to be distributed between different IPs equally in a circular order
* But the instances are constantly changing. They can have a short life-span. they can be added or disposed based on the network traffic. So, maintaining a constantly changing mapping between domain names and IP addresses will soon become a challenge and inefficient.
* Bellow, is the diagram of this approach and 98% sure that in this approach clients and other services are treated the same:



* While this works well with monolithic applications where you have a set of static IPs

It will introduce some problems in case of microservices

* **Limited scalability**: this guy says that you have to update the routing tables and IP addresses manually and this will be a problem for the ephemeral nature of microservices instances. So, this approach is not particularly container-friendly
* **Cost**: this load balancers are often costly.
* **Single Point of Failure**: although we have a backup, secondary load balancer that comes to play if the primary fails, we will still have just to of these that the entire system is relying on so the application will fail in case that these two load balancers fail.

# How to Move on to a New Approach to Overcome These Challenges

* In cloud-native applications we will have a Service Registry where service will subscribe/unsubscribe to or added/removed from upon creation/termination
* A service discovery is performed using this centralized service registry
* **Client-Side Service Discovery:** the services will register themselves in the registry, if a client wants to use a service it calls the registry and receives a list of available IPs and makes a load balanced request to one of them. The services should send their heartbeat as a health signal in intervals and if they fail to do so the registry will remove them. In case of a shut-down they need to deregister themselves from the registry.
* **Server-Side Service Discovery:** in this approach, services will be added to the registry by a third-party registration mean and when a client wants use a service it sends a request to the registry and the registry makes a load-balanced request to the proper instance of the queried service. So, the client won’t do anything regarding load balancing/registration

# Client-Side Service Registration and Load Balancing

