* Rest vs Soap
  + Representational State Transfer
  + Richardson Maturity Model
    - The steps towards becoming Restful
    - Level 0 - You have one endpoint that processes every single request in your web service, we are using http
    - Level 1 - We make resources - we have many different endpoints for different individual resources ShoppingCart, Item, User, Order
    - Level 2 - use different HTTP verbs to define different actions for interacting with the resources
      * Some HTTP verbs have different data safety levels
      * Safe - Get, Option - no matter what will never change the data in the system
      * Idempotent - Put, Patch, Delete - Change data, but multiple copies of the same request change data to the same state
      * Dangerous - Post - Every time we send post we make something new
    - Level 3 - Introduce HATEOS - Hypertext as the engine of application state
      * When interacting with a resource, it should also tell you the next endpoints and verbs for doing the process
    - <https://martinfowler.com/articles/richardsonMaturityModel.html>
  + 6 principles of Rest
    - Stateless
      * Why? We need to be able to do horizontal scaling. Once we have multiple copies of the server, the client should be able to be directed to any copy at any time.
      * If the server has state, this means the state must be replicated on every copy, which means pretty soon, the only thing the server has memory for is just state
      * We make the client send in any relevant state with every request
      * JWT
    - Uniform Interface
      * Everyone everywhere using everything interacts with the api in the same way
      * The interface should be internally consistent
      * We should be HATEOS
    - Client-Server
      * Client and server should be able to grow and change independent of each other
      * Client only knows resource endpoints and uses those for everything
    - Layered System
      * When a client sends a request, that request can be processed by many different servers, running different technologies and the client will be none the wiser
    - Cacheable
      * Whenever possible allow resources to be cacheable and declare that they are cacheable
      * It means we can save the data somewhere as the api assures us that the data won’t change frequently
    - (optional) Code on Demand
      * Mostly we return data in a format like JSON
      * We can also return executable code for a client to use to render something
  + What is soap?
    - Simple Object Access Protocol
    - Competitor to REST
    - We could use with any transfer protocol, http, ftp, smtp,
    - We always had to build an XML Envelope that described our request and how it functioned
    - The server made an XML WSDL- Web Service Definition Language
      * Defines every endpoint and what you can do at those endpoints + how to access
    - For a while SOAP was better because soap transmissions could be encrypted
    - But then HTTPs comes around and http can be encrypted as well, and nowadays soap is pretty dead
* Microservices
  + What is this design pattern
    - Idea where we break up our Monolithic Server into smaller discrete servers that each handle a particular part of our service as a whole
  + Benefits
    - Loosely Couple Servers
    - Independent Scaling of Servers - most cost efficient
    - No single point of failure - great for users
    - Easier to try new technologies and new techniques
    - Every service has its own DB - speed up DB access times
  + Downsides
    - No longer have referential integrity across our microservices
    - No longer have acid transactions across our microservices
    - More Complexity
  + What are some of the necessary pieces
    - Gateway
    - ServiceRegistry
    - Intercommunication
    - CircuitBreaking
* Netflix OSS stack
  + They made all of these tools for their internal use, and released those tools and made them open source
  + Nowadays they don’t use this stack a lot
  + Who manages it now?
    - Spring Cloud
  + How do we utilize it
    - In Spring boot, all we need to do is download the dependency, add an annotation and some properties to the application.yaml
* Eureka
  + Annotation and setup
  + Discovery
  + Heartbeat
* Zuul
  + Annotation and setup
  + Path Proxy
* FeignClient
  + Annotations and setup
  + How to send requests to other services
* Hysterix
  + Annotations and setup
  + How does it interact with FeignClient
  + Why is it important to have
* Spring Cloud Config
  + Annotations and setup
  + Why is this useful
* BASE
  + What happens to our database consistency and transactions
  + How should they look now?
  + How to implement that?
* Messaging Queues
  + Amazon SNS
  + Amazon SQS
  + How to use these to achieve E