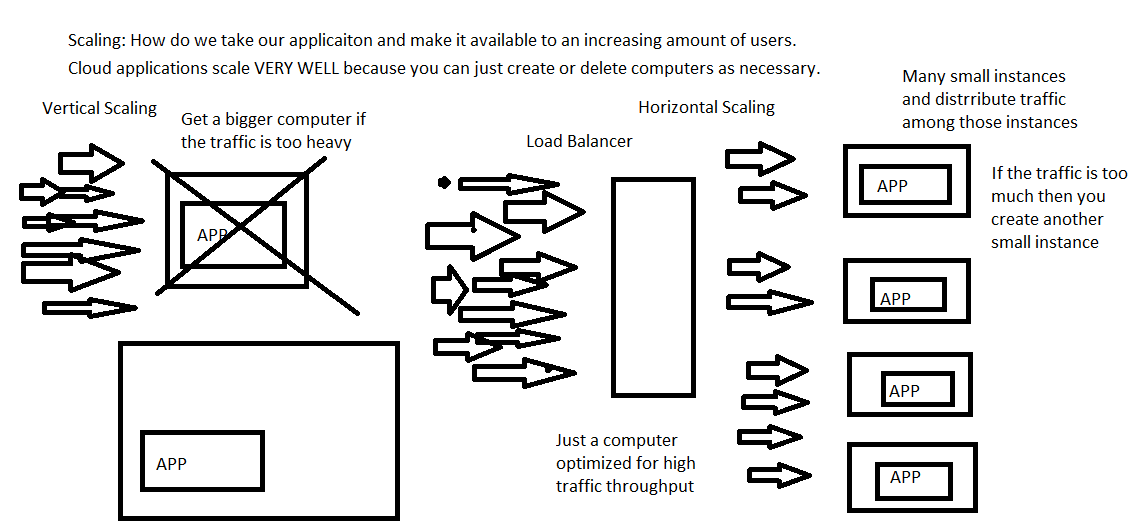
Enterprise Software Architecture



* Benefits of Horizontal scaling.
  + If an instance crashes it’s no big deal.
    - There are other instances handling most of the traffic so only a small portion of users will be immediately impacted.
    - The load balancer can detect when an instance fails and just not route traffic to it.
    - Load balancer can be programmed to recreate instances if an instance fails. Because the instances are small this usually takes minutes.
  + There is no downtime like in vertical scaling.
    - You do not have to shut down the current app to to start up a new larger version.
  + It is easier to target exactly the amount of computing resources you need to meet traffic.
    - Easily spin up another instance to meet high demand.
    - Shut down an instance during a lull.
      * SAVES MONEY.

MicroServices

* For a long time applications were monolithic.
  + The applications that you are currently making are monoliths.
* In a micro services architecture the application is split into multiple separate small features.(Micro Services).
* Micro services will communicate with each other to make a gestalt application.
* Pros to making an application a micro-services architecture.
  + Small modular of units of functionality that can be easier to work on and acquire less technical debt due to size.
    - Being asked to create a JWT service creation/validation and you do not have to worry about anything else.
  + The small modular units allow developers to pick the technologies that are best suited for that service.
    - Analytics team is a bunch of Python developers who know this great Python machine learning library for sales analysis.
    - Since all services communicate via HTTP (and probably JSON) you can write services in any language.
  + Allows you to incorporate legacy programs that work but might not be the language the team currently uses or knows how to use.
  + Micro services can give you pin point precision in scaling an application to meet demand.
* Cons
  + Micro services are a lot harder to set up.
    - Docker and Kubernetes have made it ”easier”.
    - A huge problem in Microservices is “Discovery”
      * How do microservices know what IP address the other services are located at.
      * Every time you create or restart an instance in the cloud the IP address changes.
  + When services call each other over the web via HTTP the lag is in milliseconds versus nano seconds when they are in the same program.
    - HTTP and internet connections have more security issues and more prone to disruption then a monolith.

Consul

* Not specific to spring.
  + Spring consul is just the Spring package for using consul.
  + Other frameworks and programming languages will have their own packages.
* **Discovery**
  + Discovery is the process by which services can find each other.
    - In the real world services are constantly spun up and deleted and that means the IP addresses are always changing.
  + Central Consul server (central node)
    - Any service that is spun will register itself with this server.
      * “Hey look I am a service of name ‘something’ you can find me at this IP address.
    - Any time you need to find a service consul can point you to the correct IP addresses and redirect traffic accordingly.
    - Think of the Consul discovery like a phone book.
      * Look for the name of something and it give you the IP address.
    - A service registers with consul.
      * Consul does not register services.
  + **Health Checking**
    - Consul will make periodic requests to an endpoint to see if the server is still up an running correctly.
      * Defaults to every 10 seconds.
  + **Load Balancing**
    - Consul will take requests to a service and split the requests to the instances of that services.

Gateway Service

* It provides a uniform url and entry point for requests.
  + Act as firewall
  + Add or check for specific headers