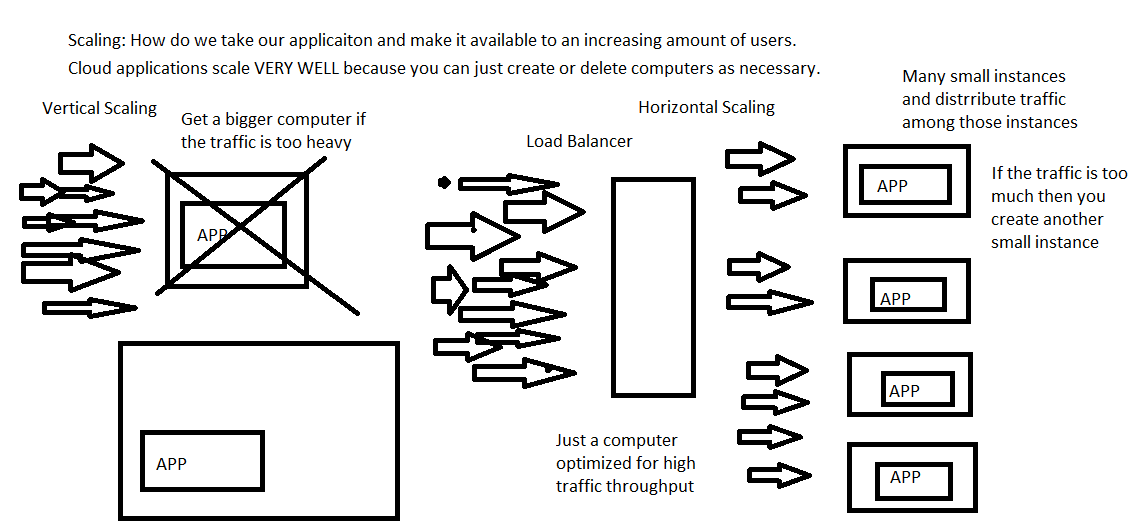
Web Services Review

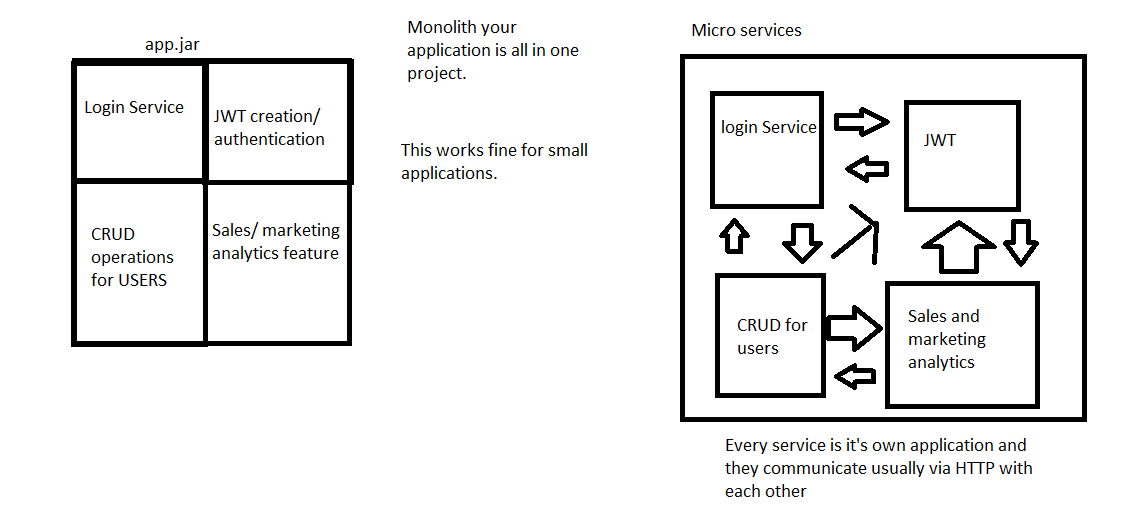
* Scaling
  + Process by which your application increases or decreases computing resources to match demand.
  + Horizontal scaling
    - Load balancer that all traffic is sent to. The balancer then distributes the traffic to many small instances.
      * More common usually preferred type of scaling.
  + Vertical Scaling
    - Creating a larger single computer to meet increased demand.
      * More prone to failure.
      * Does not work well microservices.
  + Cloud resources are supposed to be ephemeral.
    - Created and destroyed easily and essentially interchangeable.

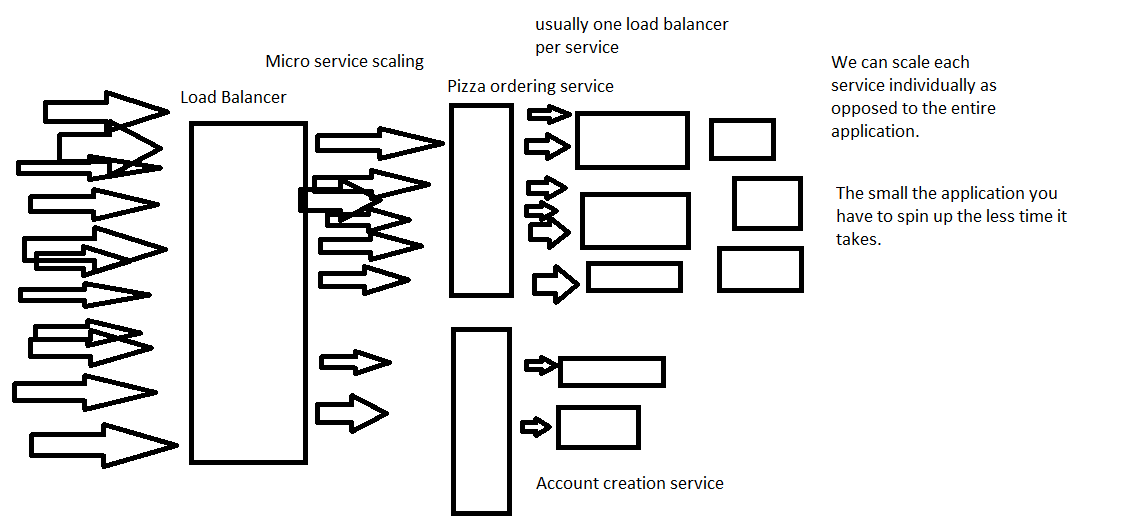


* All cloud services (including GCP) can be broken down into 3 main categories.
  + **Infrastructure as a Service (IaaS)**
    - Hardware that you can buy/provision from the cloud.
    - The compute engine VMs are the classic example.
      * Straight up computers with configuration built into it.
  + **Platform as a service (PaaS)**
    - Foundational building blocks or tools for developers to use.
      * Not completed applications but used to create applications.
    - SQL databases
      * VM’s preconfigured for use as a database.
        + Have a SQL server already installed an optimized for DB operations.
    - GCP storage
      * Buckets that store files.
    - GKE (Google Kubernetes Engine)
      * You could manually spin up several vms and install Kubernetes but GCP already has a service that does that.
  + **Software as a service**
    - Completed application that you can directly use.
      * Gmail, YouTube, Slack

Microservices vs Monolith

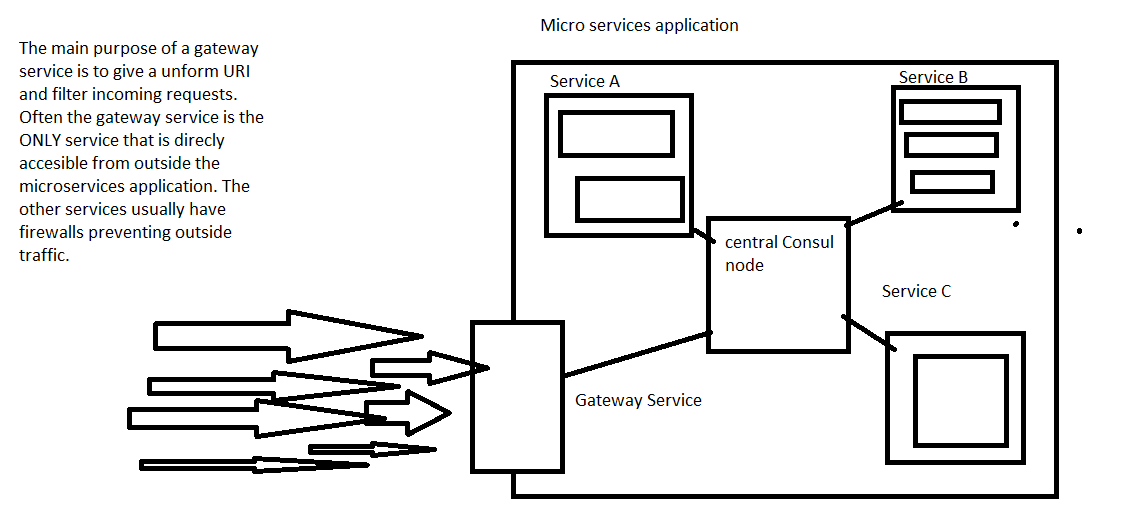
* Monolith
  + Application where all the code is in a single project.
  + All the services are just methods that can call each other.
    - Pros
      * Easy to set up.
      * No lag in between services calling each other.
    - Cons
      * Limits what technologies you use.
        + If it’s a Java app everything is Java.
      * Very large codebase for a single project that can be intimidating and tough to work on.
        + Especially true for developers who are new to the project.
        + A lot code might\* end up being intermingled.
      * A single failure can crash the server making all services useless.
* Micro-services
  + Application is split among many small services.
  + The services communicate with each other over the internet.
    - Usually HTTP and JSON.
    - Technically can be anything.
  + Pros
    - A small micro service can be easier to work on.
    - Smaller more independent module.
    - Micro-services lend themselves to Excellent pin-point precision scaling.
      * Scale an application by the amount a service is used.
    - More resistant to failures.
      * A single service going down does not crash the system.
      * Very quick to spin up another microservice.
    - Micro-services can use different languages and technologies that best suit the developers and the application requirements. (polyglot applications)
  + Cons
    - Microservices are significantly harder to
      * Set up
      * Configure
      * Maintain
    - Add a LOT more latency, unreliability between services.
      * Milliseconds compared nanoseconds.
    - More points of potential failure.





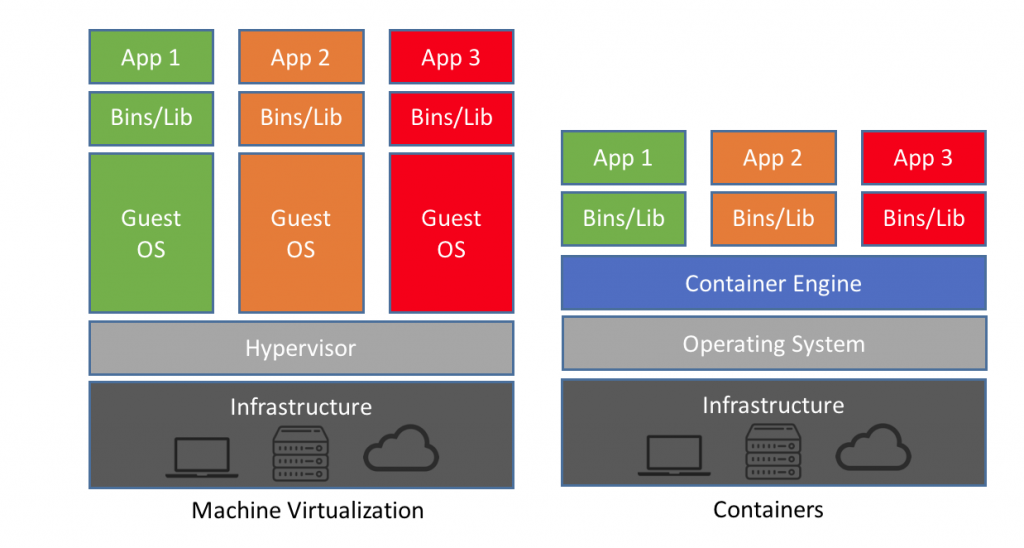
Microservices in Spring

* Consul was the main Microservices software mesh tool that we used.
  + IT IS NOT SPECIFIC TO SPRING.
* Central server/software that provides a few fundamental microservices features for our application.
  + **Discovery**.
    - Consul acts as a registry for a micro-service instance to list and identify itself.
      * Type of service it is.
      * The IP address the instance is located at.
        + IP addresses are constantly changing on the cloud. Trying to call a service just by it’s IP address is doomed to fail.
    - Micro-services discover (find other micro-services) by checking with consul.
  + **Health Checks**
    - Consul will make requests to a specified endpoint on an instance to make sure that it is still working correctly.
  + **Load Balancing**
    - Splitting traffic to a service to its different instances.
    - @LoadBalanced
      * **Ribbon** is the technology under the hood that makes this work.
* Important annotations and code snippets
  + @EnableDiscoveryClient
    - Used to tell the Spring application to register with consul and to discover the other services on there.
  + RestTemplate
    - Allows you to create HTTP requests.
    - You can specify full URLS.
    - You can make requests to services in the URI and if the discovery is configured correctly it will find an IP address that provides that service.
  + @LoadBalanced
    - Put on a RestTemplate bean.
    - Requests made to a service will be split among the many instances.
  + Configuration for most services is done application.yaml.
    - Service name
    - Location of a consul server.
      * If not specified defaults to localhost:8500
  + Gateway Serivce
    - A micro service that forwards and filters requests.
    - Provides a uniform single url for anyone needing to use the microservices.



Docker

* Containerization software.
* Addresses the problem of “it works on my machine”.
  + Deployments are difficult.
  + Applications need specific environments and configurations to run.
* Container vs VM
  + Containers
    - Very lightweight compared to a VM.
    - Containers usually lack an OS.
    - They run as a simple program on a Linux machine not as a full fledged separate machine on a machine.
      * More efficient than VMs
  + VMs
    - Complete virtualization of a computer.
    - Includes CPU and memory.
    - Much bulkier and less efficient.



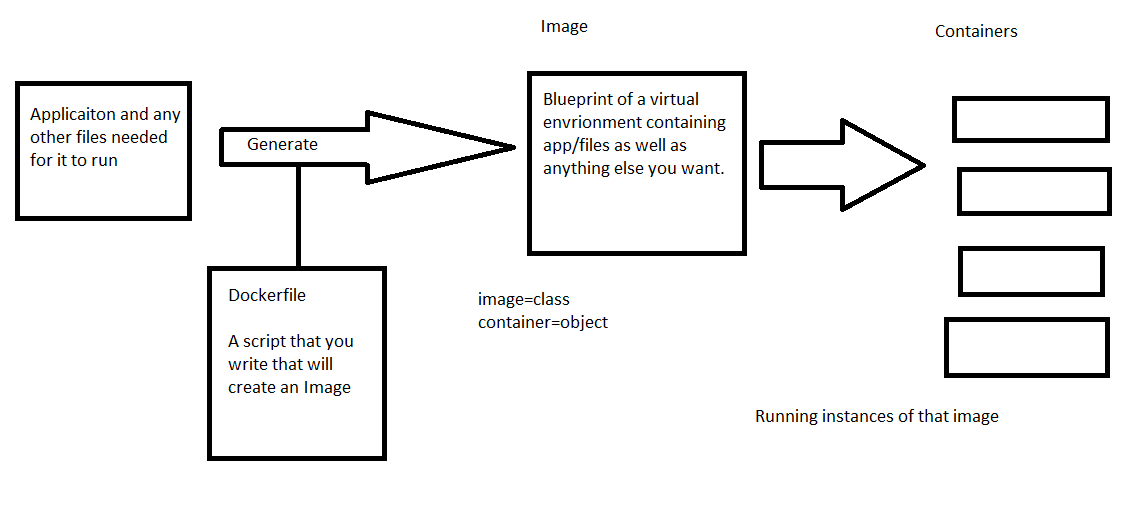
**Key Docker terminology**

* **Image**
  + Blueprint for a container.
  + Snapshot or ‘image’ of a virtual environment.
  + An image is like a class.
* **Container**
  + A *running* instance of an image.
    - You can shut down containers.
  + A container is like an instance of a class.
* **Dockerfile**
  + A script that creates an image.
    - FROM
      * First line of a Dockerfile.
      * Parent image. Usually a base environment for you to build off of.
        + Java
        + Node
        + Python
        + MySQL
    - COPY
      * What files to put into this image
    - RUN
      * Command to be executed in the virtual environment
        + Echo ‘hello’
        + Npm intsall
    - WORKDIR
      * Specify what directory your commands are running in.
    - EXPOSE
      * What port does the application listen to.
        + Needed for port mapping.
    - ENTRYPOINT
      * The command that gets run when you create an instance of the image.
        + Usually this starts the application that is in the image.

npm start

Java -jar myjar.jar

* Docker commands
  + Docker images
    - Lists all images
  + Docker ps
    - Show all running containers
  + Docker build -t myimage <path to dockerfile>
    - Build an image using this script
  + Docker run -p 80:7000 imagename
    - Create an instance of that image and map the port 7000 on that image to an outside port 80
  + Docker kill container id.
    - Destroy the container
* Docker hub
  + The repository for images.
  + Maven Repository is a repository for Java code packages.
  + Docker hub is a repository for docker images

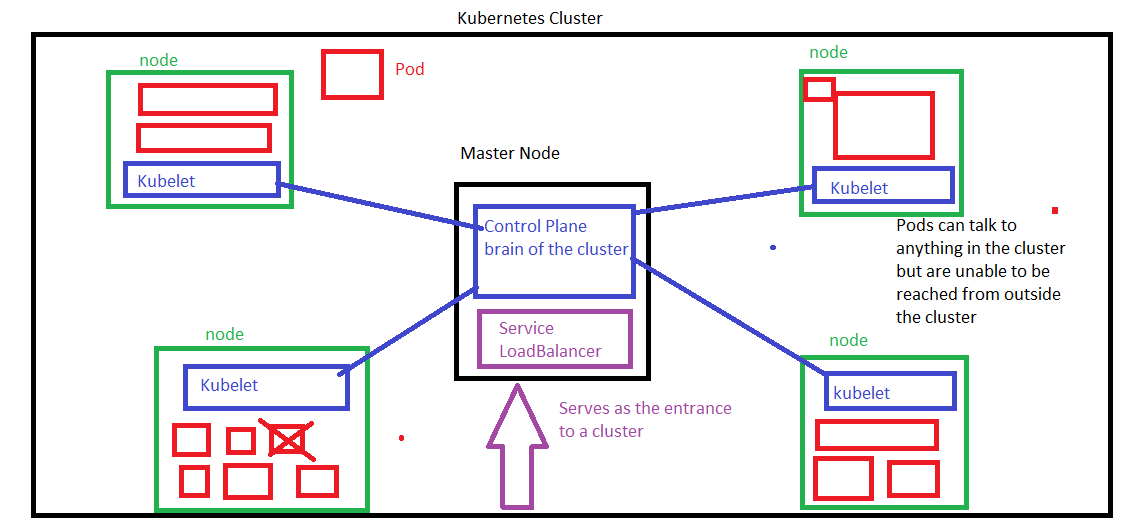


**Kubernetes**

* Container orchestration software.
* It will manage a whole bunch of containers and ensure that they are always operating correctly.
  + There are always the correct instances up and running.
    - If one shuts down it will spin up another one.
    - Configured to perform auto-scaling for you.
    - Balance traffic between containers.
* Key terms
  + Node
    - Literally a computer (often it is a VM if Kubernetes is used in the cloud)
  + Master Node
    - The main node that host the Kubernetes control plane.
  + Control Plane
    - RESTful server that acts as the “brain” of a Kubernetes cluster.
  + Kubernetes Cluster
    - A master node and all the nodes that are controlled by the master node.
  + Kubelet
    - Software on a node that allows it to be connected and controlled by the master node.
      * I have seen people just called these nodes kubelets
  + Pod
    - Atomic unit of software that you can put on a cluster.
      * A pod contains one or many containers.
  + Deployment
    - Configuration/formation of pods
      * Specify things about the deployment like
        + ReplicaSet

How pods should there always be.

* + - * + When do you create new pods.
        + When do you delete pods.
  + Service
    - How we expose a deployment and pods on the cluster to outside requests.
      * Everything cluster is closed off to the outside world EXCEPT for services.
    - LoadBalancer
      * Creates a loadbalacner on the cluster that has an external IP address and will forward requests to appropriate pods.
  + How do you configure kubernetes?
    - Declarative by design.
    - The idea of writing scripts is discouraged.
      * Scripts are imperative you say step by step what to do.
        + SQL works this way
    - You define what you want in a file.
      * You ask kubernetes to make it happen.
        + Magic.



OWASP top 10

Open Web Application Security Project

https://owasp.org/www-project-top-ten/

Each team will pick 2 security flaws. You will make a cheat sheet about that security flaw.

should unclude the following parts

- what is it?

- how does the vulneratbility work?

- why is it dangerous? (see if you can find a real world example)

- how to protect against it

- If possible write a piece of code or demo that shows off the vulneratbility\*\*\*\*

Each part should only be 2 or threee sentences.

Include a diagram if it is helpful.

Each vulnerabliity presentaion should only take about 3-5 minutes.

1. injection

3

https://docs.google.com/document/d/12O2LG\_wfsIY8QtYkiFDYla6w1VaTShn01dEskgGa7es/edit?usp=sharing

2. Broken Authentication

4

https://drive.google.com/file/d/1BENTvJ3IBRKRp8U4vKJBTaNxCLASpJ51/view?usp=sharing

3. Sensitive Data Exposure

2

https://docs.google.com/document/d/19wkk-gNmnhxYjEONjam8B5YGPYJI7jeTZxrLV9UliUk/edit?usp=sharing

4. XML external entities

1

5. Broken Access Controls

5

https://docs.google.com/document/d/1MG4wUTjEQwzsM0-wA9rXV5RGQKOXmGe6aQWv33USysM/edit?usp=sharing

6. Security Miconfiguration

2

https://docs.google.com/document/d/19wkk-gNmnhxYjEONjam8B5YGPYJI7jeTZxrLV9UliUk/edit?usp=sharing

7. Cross site scripting

1

8. Insecure deserialization

4

https://drive.google.com/file/d/1BENTvJ3IBRKRp8U4vKJBTaNxCLASpJ51/view?usp=sharing

9. Using components with known vulnerabilities

3

https://docs.google.com/document/d/12O2LG\_wfsIY8QtYkiFDYla6w1VaTShn01dEskgGa7es/edit?usp=sharing

10. Insufficient logging and monitoring

5

https://docs.google.com/document/d/1MG4wUTjEQwzsM0-wA9rXV5RGQKOXmGe6aQWv33USysM/edit?usp=sharing