Micro-Services

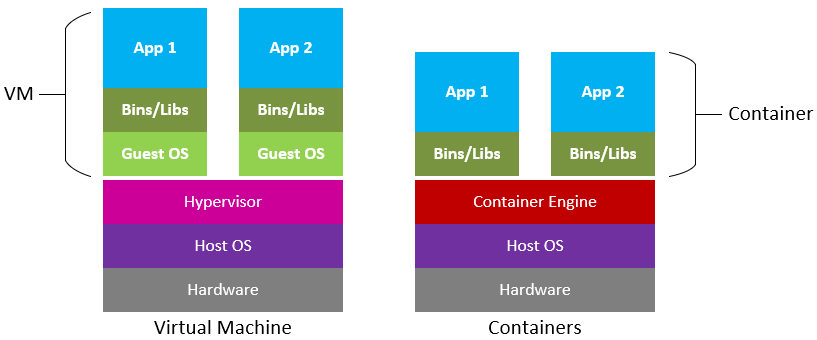
* A newish type of software system architecture
* Micro-services is a break from monolithic applications
  + All your code for running an application is on a single machine
* Break down a large application into small independent services on their own servers.
* They will interact with each
* All services are black box meaning you need only to know how to interface with them not how they work
* Pro
  + Allows you to scale each individual feature
  + A service can fail without the entire application becoming useless
  + Allows developers to work on small individuals projects without needing to know the entirety of the software ecosystem
  + Microservices are blackbox and usually communicate via URLs
  + Technically each microservice could be written in a different programming language
* Cons
  + Microservices can add a lot more overhead and configuration to your application
  + Microservices and their databases tend to be BASE not ACID
    - Basically Availability Soft state Eventually
    - Usually each microservice has its own database which means that your overall application might have inconsistencies

Technologies

* H2 database
  + In memory database for you application
    - There is no persistence if you shut down the server
  + Very helpful development tool
  + Allows you to simulate having a database without actually having to create one
* Eureka (discovery service / registration service)
  + Your microservices to communicate with each other need to register with a Registry service like eureka
  + Think of it like a phone book where each service will list their name and what URL the service can be reached at
* YAML (YAML ain’t markup language)
  + It’s a type of formatting
  + Can be used instead of xml or .properties, or .json
  + Sensitive whitespace
* Feign Client
  + Serves as an interface for us to programmatically create http calls from out Java
  + Rather than having to know the URL endpoints to consume we will configure Feign to access those endpoints for us and return Java Objects
* Hystrix
  + Serves as a circuit breaker and failsafe for our applications
  + If a service fails Hystrix will activate a circuit breaker and use a backup
  + Try-catch vs Hystrix
    - Definitely use try catch for completely internal logic
      * Methods that don’t rely on another service
    - Hystrix circuit breaking can be configured more than a try catch
      * If a method fails you can immediately redirect incoming request to the backup method without the code in the try executing
    - Exponential back-off
      * If a service calls fails you do not try again for 5 seconds
        + Then 10
        + Then 20 seconds
        + Then 40 seconds
* Zuul (Gateway service)
  + Primarily responsible for routing
  + In a micro services architecture you want to have unified domain
    - You do not want to have what we have with 5 different ports
    - You do not want to have 5 different domains
  + One URL/ base API for everyone to work off of

Docker

* Docker is designed to address the old problem of “It works on my machine”
  + Setting up environments for your software to execute and execute reliably across multiple machines is always challenging
    - One of key features of Java in 1995 WORA
* Docker is a containerization software
* Container is a virtual environment for your application to run in
  + This container can have system variables and other software that your application need to run
* Containers vs Virtual Machines
  + Containers more lightweight
    - Less taxing processing and memory
    - Less management than a VM
    - Much easier to deploy and control containers than VMs
  + Complete control over the environment that a VM could



* Hyper visors will distribute the processing power and memory among the multiple VMs
* Container Engine (Docker) will manage your applications threads and processes without the need for a Guest OS
  + Direct interaction with the OS leads to increased efficiency
* Docker is primarily for Linux
  + You can use it with Windows containers though

Docker Terminology

* Image
  + A blue-print for a container
  + Optionally contains a base image with the software/environment necessary to run your application
* Dockerfile
  + A file with instructions on how to make and create an image

Docker commands

* docker ps (lists all running containers on a machine)
* docker build (Build a image off of a Dockerfile)
* docker run (create a container based off an image)

Container Orchestration

* We can scale amount of containers of an application to meet demand
  + Bit more targeted than using entire EC2s to scale the application
* Deploying applications in containers is generally easier to manage and control
* Docker Swarm
* Kubernetes
  + KaaS (Kubernetes as a Service)
* Fargate (Serverless Container Service provided by Amazon)
  + Can deploy your container application without needing to spin up an EC2