freecodecamp.org

A brief history of serverless (or, how I learned to stop worrying and start loving the cloud)

freeCodeCamp.org

11-14 minutes

by Himanshu Pant

Gee, I wish we had one of them doomsday machines.

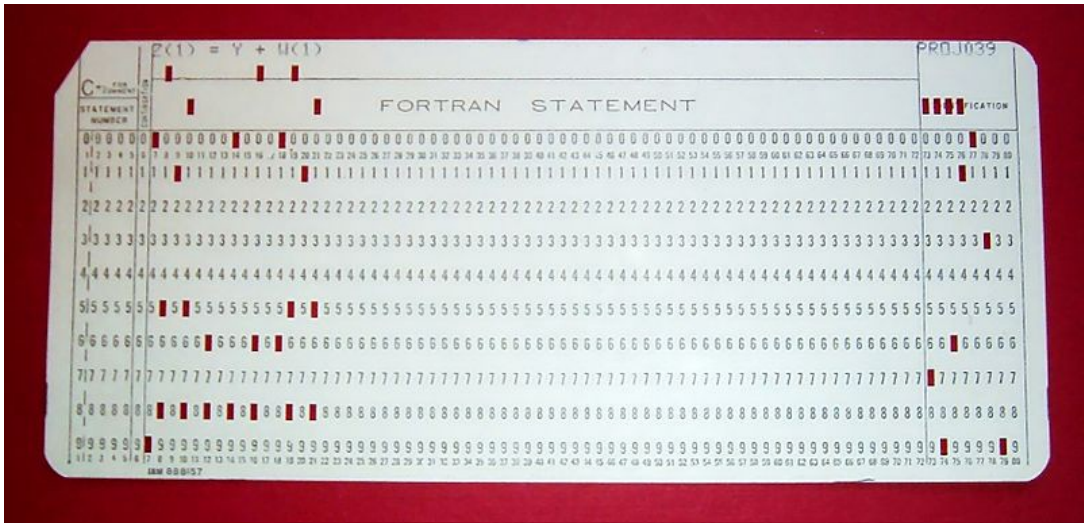
Once upon a time…

The beginning

In the good ol' days of the 1950s when Elvis was ruling the charts with hits like Jailhouse Rock, a computing paradigm came out called mainframes a.k.a Big Iron (no relation to Arnold Schwarzenegger).

IBM was the main mover and shaker in this space, with a sprinkling of other small providers sometimes giving competition to Big Blue.

These early beasts were purely batch processing oriented — Mr John Appleseed could punch in his instructions(read programs) on a punch card (see below), pass it to the operations team, and go off for a coffee, or even leave for the day.



By Arnold Reinhold — I took this picture of an artifact in my possession. The card was created in the late 1960s or early 1970s and has no copyright notice., CC BY-SA 2.5, https://commons.wikimedia.org/w/index.php?curid=775153

The operator would schedule and execute the previously submitted punch cards.

© MITRE Corporation

Fun fact: the above picture is of a HUGE 5Mb of of data on 62,500 punched cards. So just imagine the number of cards required to capture the code base of Win 10 or MacOS if we still used this technology!

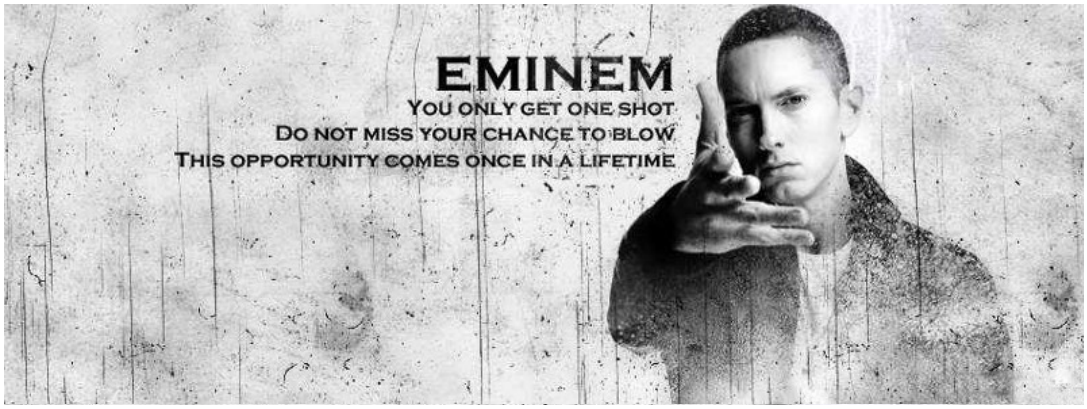
Once the output was generated/printed, the operator would physically collect it and deposit the same to Mr. Appleseed's cubicle or physical mailbox. Online access was primarily reserved for resources requiring it, like, say, staff at an airline's booking counter.

So to all those crying in spite of having GUI-based IDEs to do their software development, if you think your job is hard — think of these

pioneers.

Needless to say, the above process was not just slow but also extremely expensive. So much so that system usage was measured and charged in seconds towards the client's account.

Code re-writes were extremely cumbersome and time-consuming. In fact, come to think of it, this may be the reason for the exceptionally good coding skills of the developers of that era, because as one Mr. Real slim shady has rightly said:



The PC Era

Next came the era of the PC, which was the democratization of technology in the form of the personal computer.

The processing model started moving from mainframe-based, centralized systems to the client-server-based model. Infrastructure planning, procurement, and maintenance became a vital part of software development, as the pace of business started to increase exponentially.

So it was not unusual to see software deployment on a client-server-based system accompanied by projections for future volume growth. Provisions had to be made for that. As it was, the typical engineer's job was tough — but this also required him to be a soothsayer.

So now, our poor Mr. Appleseed had a lot of things to figure out. He not only had to figure out what the business teams were saying in the form of their requirements, but he also had to learn all the coding tools, standards, best practices, and innovations to follow.

He also had to take into account parameters like what the usage patterns of the software might be, how fast or slow it may grow in future, and what the optimum hardware may be so as to strike the right balance between performance and cost efficiency.

The bureaucratic hurdles in any given organization ensured that the hapless engineer was always stuck between the devil and the deep blue sea.

In addition to having sleepless nights over the ever-changing requirements and shifting deadlines, he also had to make do with the vagaries of infrastructure procurement. In the mainframe world,

the situation was a bit better, as hardware procurement was not that commonly done. But resource allocation was still a chore, which led to a lot of heartburn.

The Cloud Era

The mid-2000s saw the advent of a new paradigm in computing: “the cloud.” It changed the nature of computing as it was known up to that point.

Before we delve deeper into it, let’s take a minute to refresh the definition of “the cloud” as given by National Institute of Standards and Technology (NIST):

> *Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. Five major attributes define a cloud based system viz-*
>
> *a) On Demand self service*
>
> *b) Broad network access*
>
> *c) Resource pooling*
>
> *d) Rapid elasticity*
>
> *e) Measurement of service*
>
> *The key enablers for this paradigm are:-*
>
> *a) Fast WANs*
>
> *b) powerful commodity servers*
>
> *c) high performance visualization for commodity hardware*

Our beloved Mr. Appleseed was now happy — after all, he was no longer at the mercy of his infrastructure engineer for getting servers

or storage. He could, at the push of a button, get compute power, storage, a queue, and any other such service.

All was well in the computing world. But Mr. Appleseed, true to his human nature and being an enterprising species, started pondering how he could make his life easier.

Moreover, the world was becoming digital at a great pace with unheard-of scales of enterprises. This was making the practice of pre-allocating hardware a self defeating purpose. All the interactions with the digital world were becoming majorly event-driven.

Enter Serverless architecture

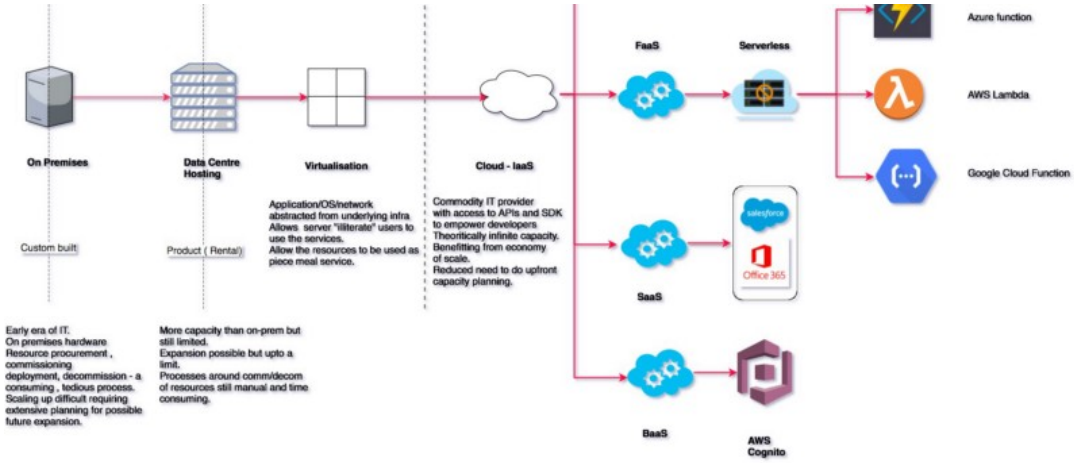
2015 (or some say 2012) was the time when this computing paradigm came into being. There have been a number of interpretations proposed for this term (serverless) and its implications.

One school of thought attributes it to Backend as a Service (BaaS). For example, authentication services offered by third party providers like Google or Facebook.

The other school of thought links it to a concept wherein applications with business (read: server side) logic are run over stateless containers, managed by a third party provider in its entirety, which is known as Function as a Service (Faas).

This article focuses on the second definition of the concept, as it has interesting implications for the way web applications are being architected.

The brief evolution of the cloud computing paradigm



At its core, FaaS is a simple concept which means that:

- The development team should not get worried over aspects like the backend server, its maintenance, procurement, or scaling (well, to an extent). All the team has to worry about is application logic.
- Processing is carried out on compute containers which are stateless. So in other words, after one logical unit of processing, there is no storage of the processing attributes by the system
- Instead of having a server with long running processes, say cron, here the processing is only initiated once the qualifying "**event"** occurs and is terminated when the processing is complete or the set time has elapsed (whichever happens earlier).

  It does not mean that there are no servers anywhere in the whole scheme of things. All it means is that the servers and their maintenance is now hidden from the developer.

  Also, since the physical unit of the compute is now a container, there is no need to have long running servers with event listeners running on them to carry out any processing. Qualified event sources could be plugged into the system and the service would take over.

What Serverless is NOT

> *"What's in a name? That which we call a rose by any other name*

> *would smell as sweet". — Juliet, Romeo and Juliet*

As it is with any new innovation, more often than not glitzy names are used by vendors to re-package old wine in a new bottle, so to speak, to gain more eyeballs.

In case of serverless, this confusion is more wide-spread. This is because the concept has developed in a very short period of time and in very close proximity to the other concepts with which it can be confused.

So here I take a shot at trying to clarify what serverless usually is (or can be) confused with. I'll also share my two cents.

It's NOT a Container

> *"A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings. Available for both Linux and Windows based apps, containerized software will always run the same, regardless of the environment. Containers isolate software from its surroundings, for example differences between development and staging environments and help reduce conflicts between teams running different software on the same infrastructure."* — Docker.com

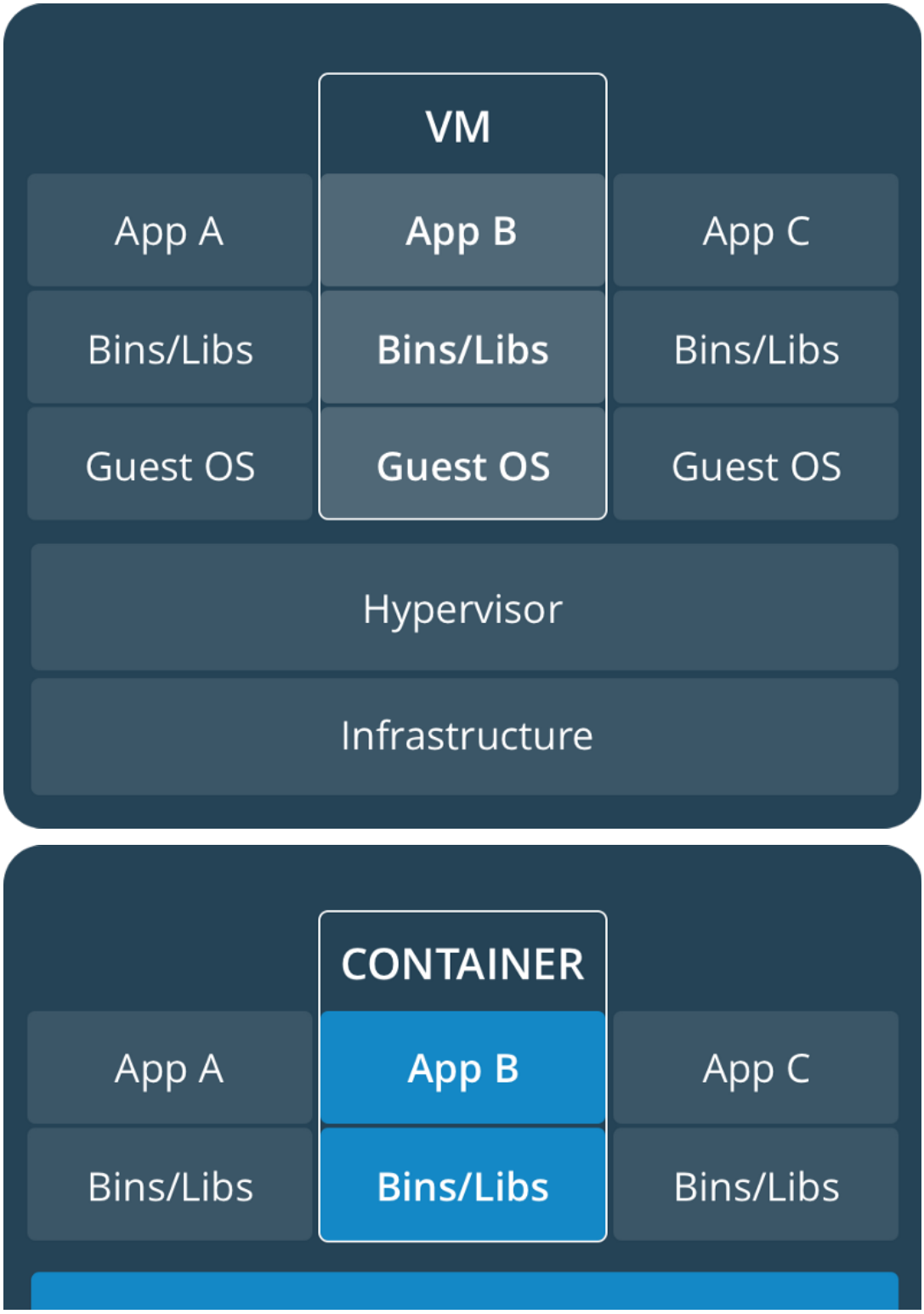
The container is a recent innovation in computing. Google popularized it by running Gmail on it. Containers are useful for maintaining the reliability and homogeneity of software running across various computing environments.

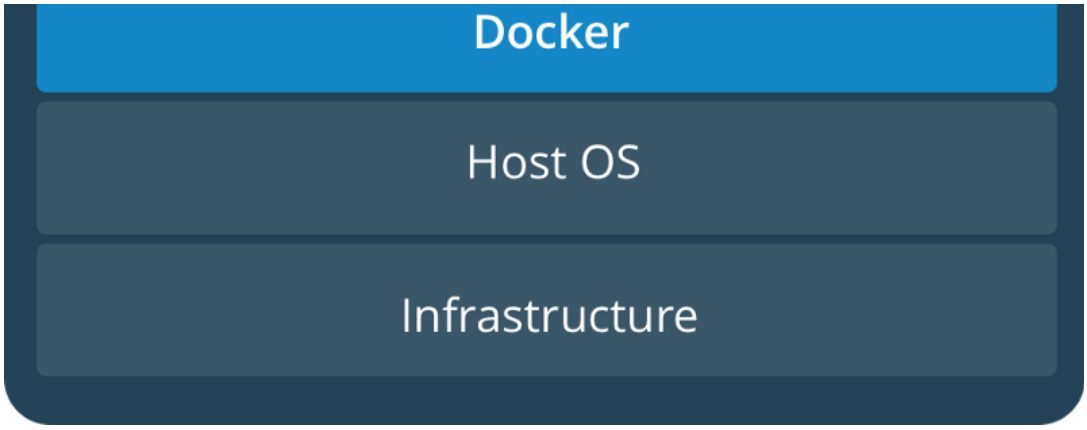
For example, the code may have been developed in Java 8, but production may be on Java 9 — so the code which was running fine in dev may start throwing weird errors in production.

Roughly speaking, the container is the erstwhile VM (virtual machine) on steroids. It has none of the excess flab of individual

OS versions. Unlike VMs, where there used to be a separate copy of the guest OS (thereby making the VM resource heavy), containers share the underlying OS kernel. This allows multiple containers to be run on that particular machine compared to VMs.

VM vs Container





VM vs Container

Containers remove one part of the problem: they provide a homogenous runtime production environment, albeit at the cost of increased maintenance effort.

However, to be fair, containers are typically better-placed for different kinds of work-load (like those that are inherently more complex). So they find favor in enterprise IT landscapes where there is already a monolith up and running and the organization may be wanting to port it quickly to the cloud.

Horizontal scaling is a vector wherein serverless steals the show over containers. Modern cloud vendors **"theoretically"** provide unlimited scaling capability for their serverless offerings. And best of all, this scaling is totally transparent to the user.

Serverless computing is better aligned to event-driven, asynchronous operations, while containers appear to be better aligned to the synchronous REQ/RESP workloads.

But given the pace at which things are changing, the differences can get diluted to a great extent over a short period of time.

It's NOT PaaS

> *"Platform as a service" (PaaS) is a cloud computing model in which a thirdparty provider delivers hardware and software tools — usually those needed for application development — to users over*

> *the internet. A PaaS provider hosts the hardware and software on its own infrastructure. As a result, PaaS frees users from having to install in-house hardware and software to develop or run a new application.”* — Techtarget

Just like containers , PaaS also differs mainly concerning the vector of scaling. However hands off the PaaS vendors may claim their offering is, it still requires some admin maintenance effort, unlike serverless.

With PaaS, there is always a minimum running footprint in the system which will be up and incurring costs. However with serverless, it can be brought down to absolute zero.

PaaS may still be a good choice given its more advanced ecosystem and tooling and language support. However this should not be a show-stopper given the high pace of advancements in the field of serverless.
