Introduction to CI/CD

Module 1: Modern Computing

Role of Hardware

- Advances in computing are always driven by advances in hardware capability
- For example: until desktop computer hardware became available
  - The only computing was mainframe batch processing
  - The costs of hardware prevented other models of computing from being feasible economically

Distributed Computing

- Local compute hardware enabled remote computing
  - Led to the development of networking hardware, software and protocols
  - Led to implementation of new ways of programming (object oriented for example)
- New kinds of software could now be developed and deployed
  - These often were well developed in theory but the computing infrastructure could not support their implementation

The Rise of AI

- Many of the modern AI developments were well developed theoretically decades ago
  - For example, neural networks were developed theoretically in the 1960s
- However the hardware of the time could not do the necessary computations to make them a reality
  - When the compute power became available in the 2010s, then neural nets became possible from an engineering perspective
  - One of the current concerns is that the computing and energy requirements for LLMs is exceeding the compute hardware capabilities and power supplies available

Evolution of Neural Networks

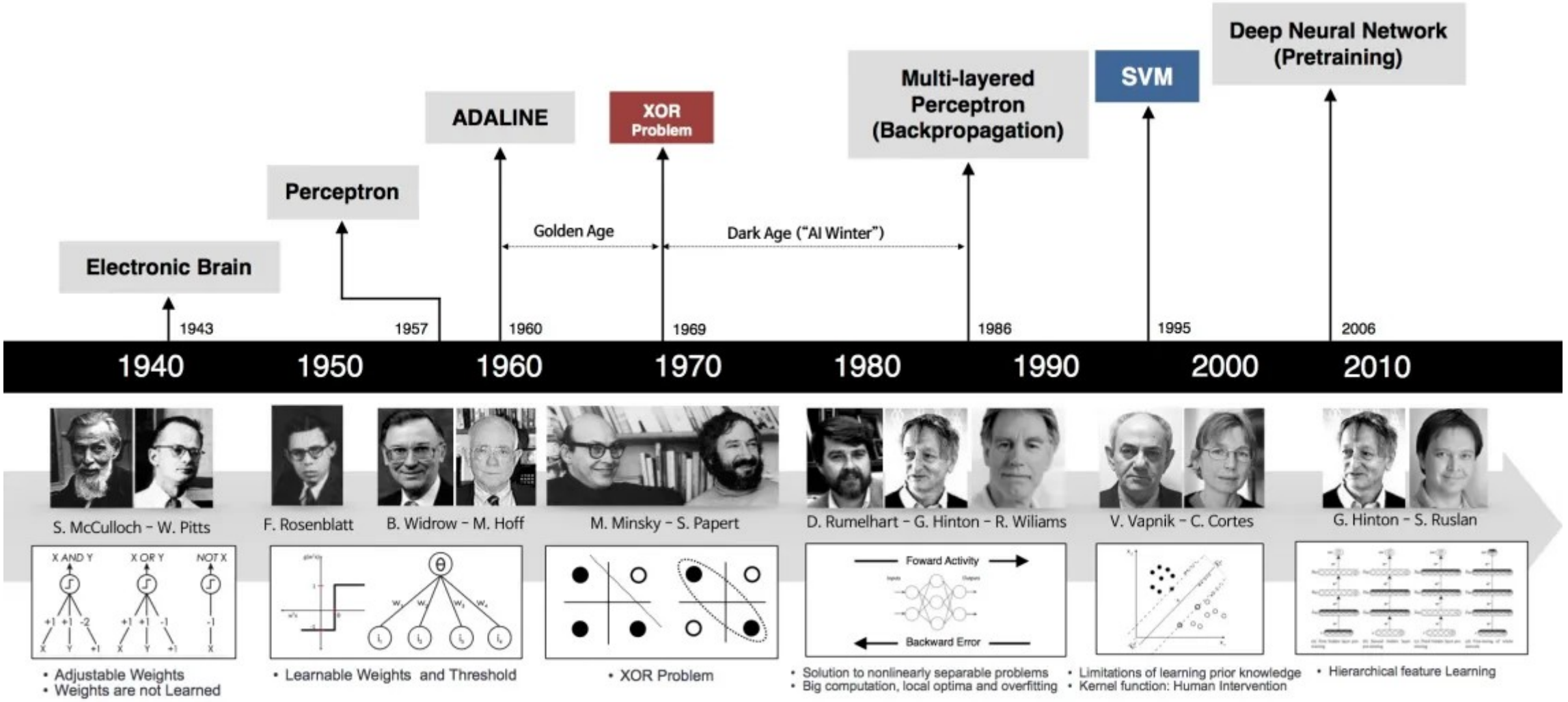


Image Credit: https://sefiks.com/2017/10/14/evolution-of-neural-networks/

Moore’s Law

- Historically, advances in hardware capabilities tend to be rapid while costs drop at the same time
- This is referred to as Moore’s law
  - Originally formulated as a measure of the amount of computation power of a chip in terms of “transistors”
  - The original formulation is no longer valid because of changes in chip and CPU architecture
  - But the term Moore’s law is now generally used as description of increasing compute and storage capabilities coupled with dropping costs

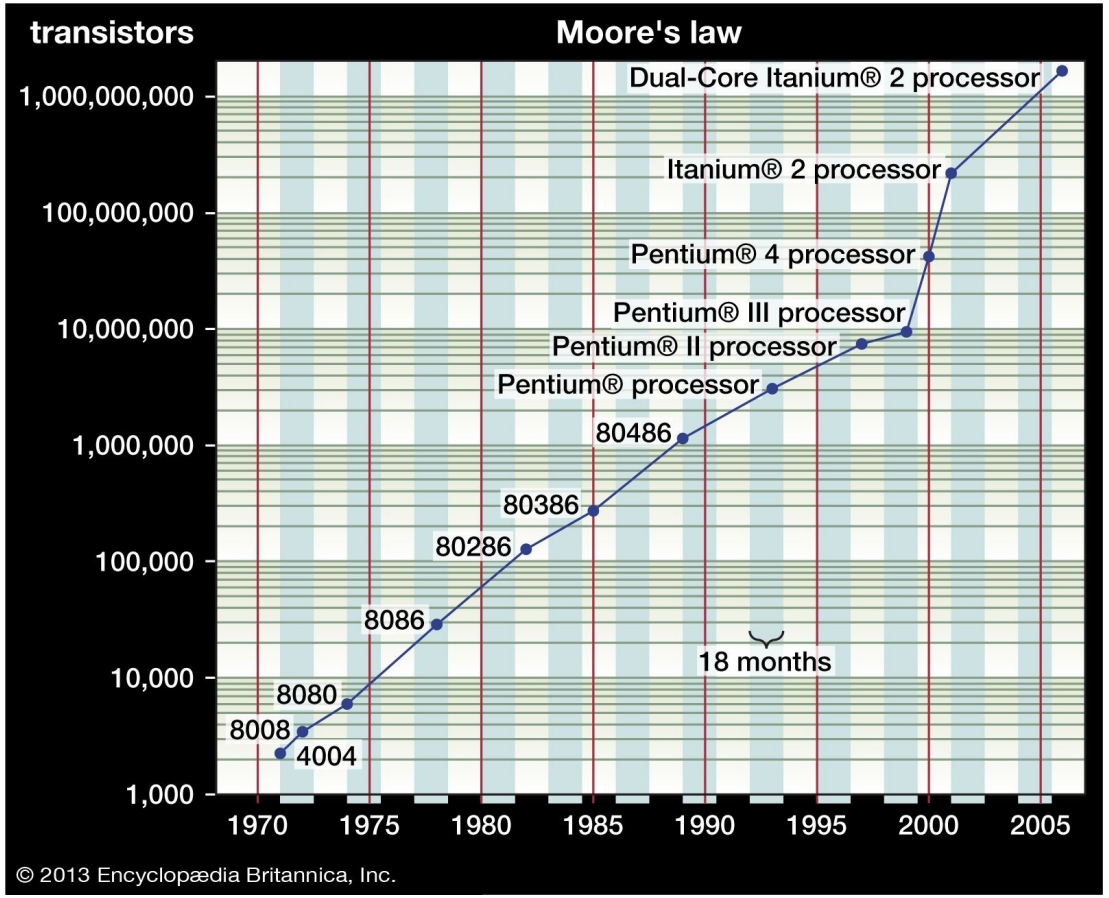


Image Credit: https://www.britannica.com/technology/Moores-law

Hardware Capabilities

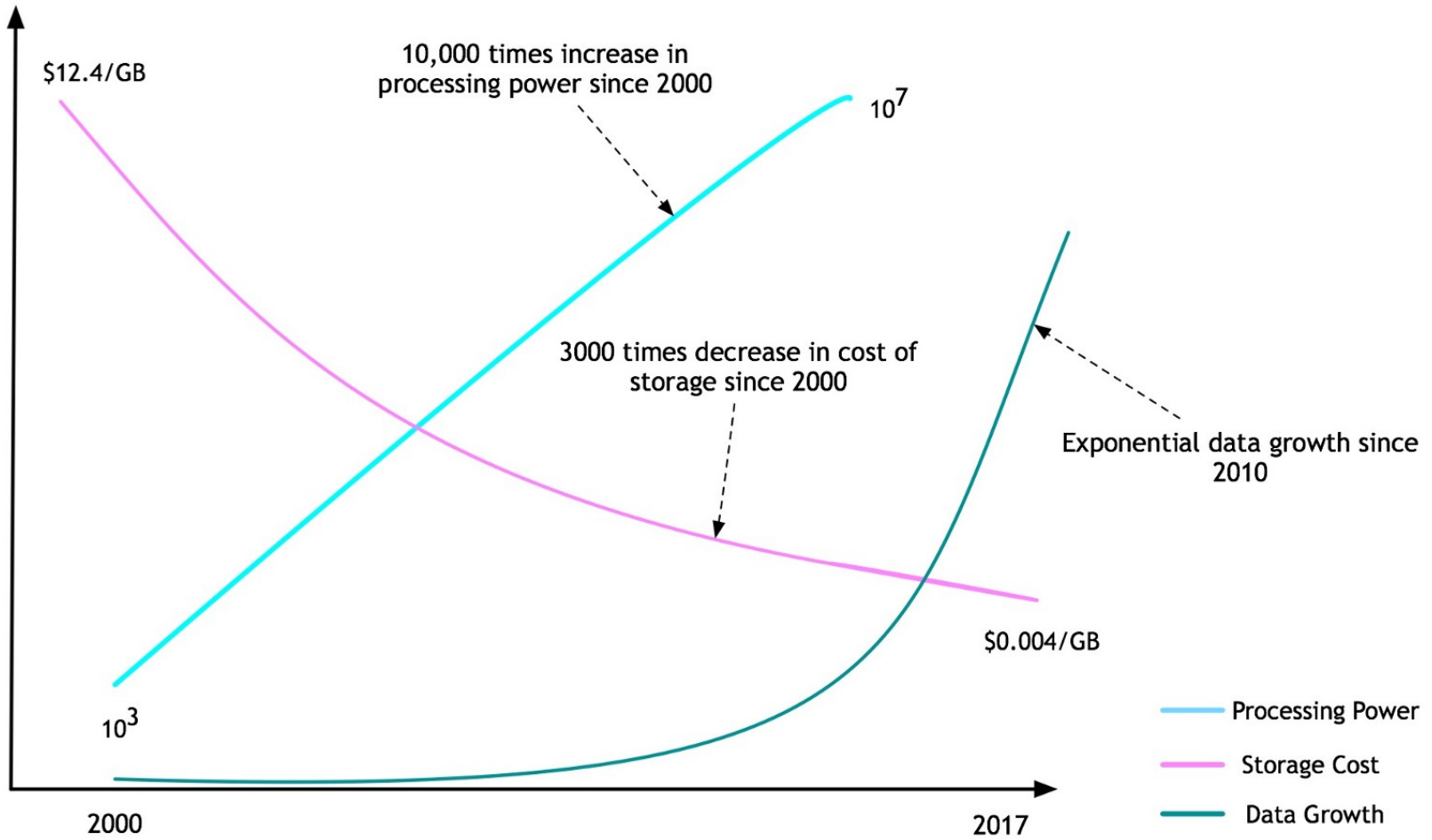
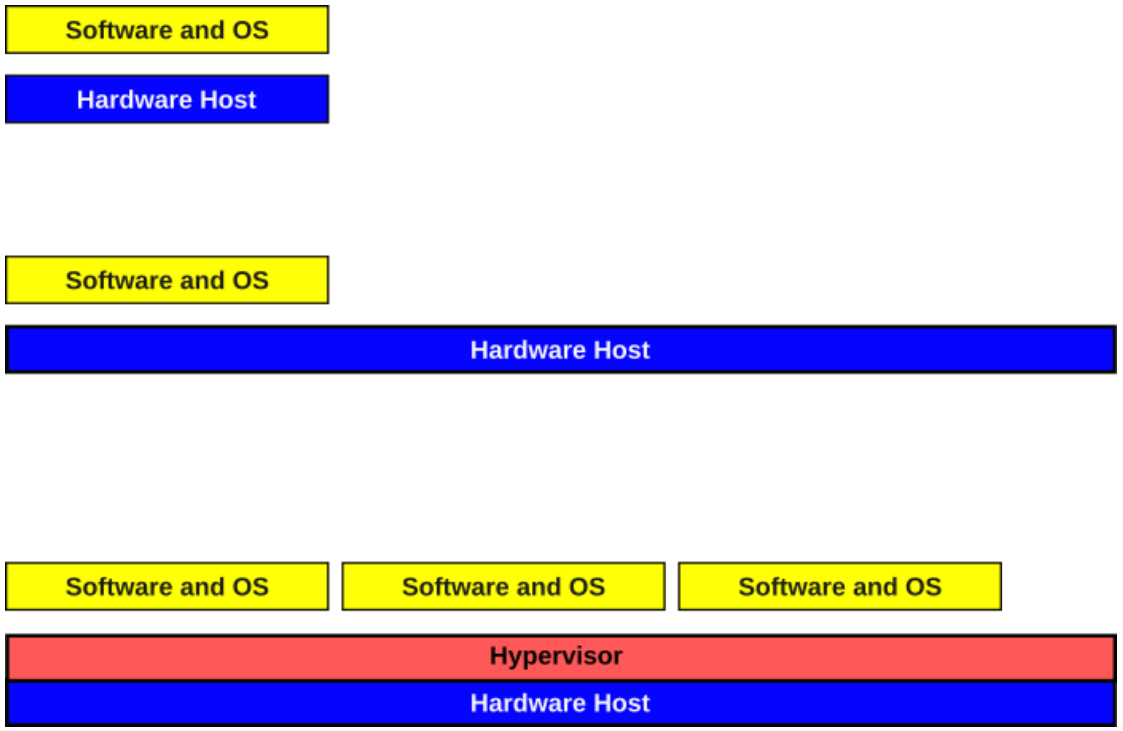


Image Credit: https://blog.govnet.co.uk/technology/reflection-on-the-maturity-of-bim-and-digital-twins

Hardware Outstrips Software

- Historically, the limit of computing was hardware capabilities
- With the increases in hardware capabilities, the situation reversed
- The model of running and developing a single application in an OS running on a hardware platform didn’t scale well
  - Hardware capabilities became under used
  - Adding virtualization allowed for multiple OS installations (VMs) to use the same hardware



Containers

- An alternative to VMs was development of Linux containers
  - These allowed very lightweight self-contained applications to be independently
  - Like VMs, these are much smaller without the overhead of an OS and libraries
  - Containers only contain what is necessary to run the application
  - Containers don’t require infrastructure beyond the container engine

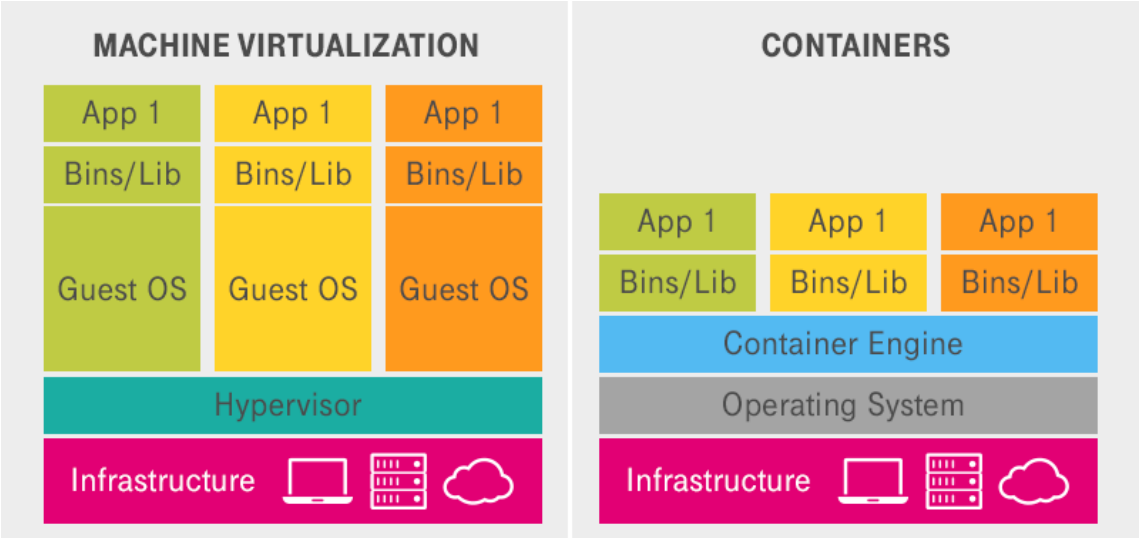


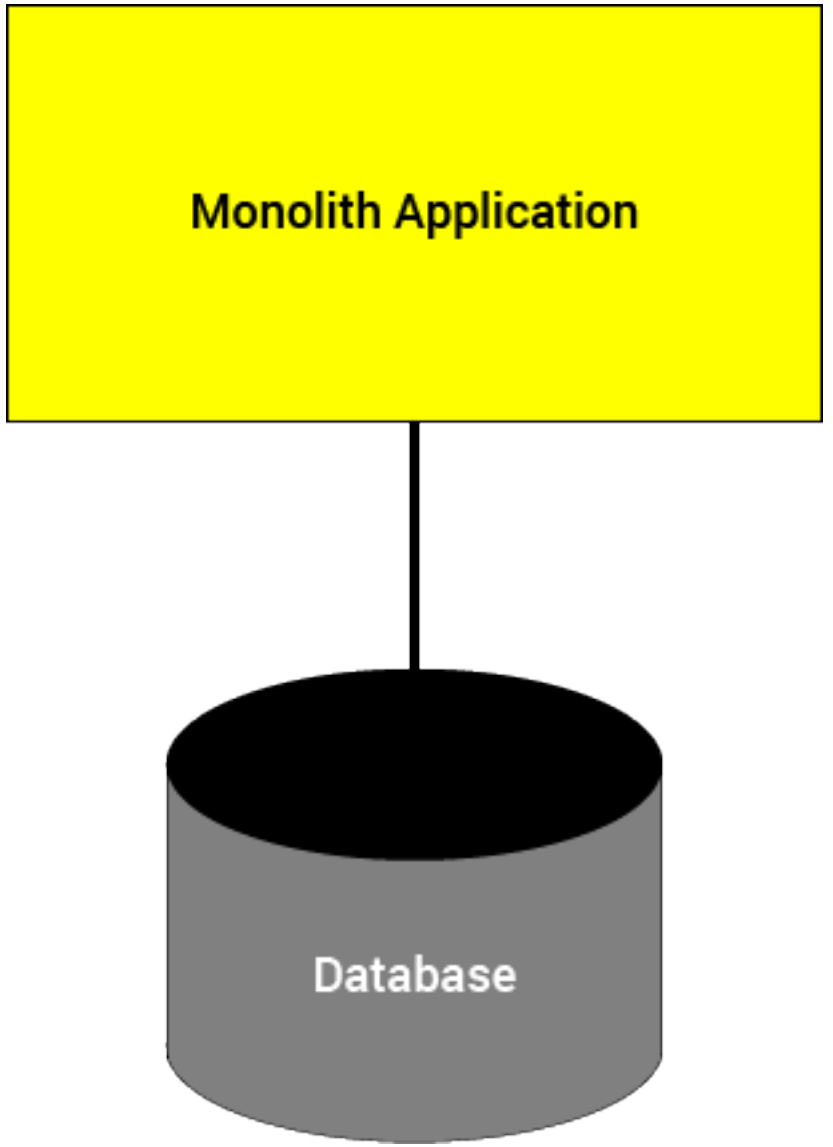
Image Credit: https://www.open-telekom-cloud.com/en/blog/cloud-computing/container-vs-vm

Microservices

- Microservices are a software architectural pattern
  - They effectively solve problems in modern software operations
  - Specifically, issues around scaling in both the development and operations
- Deploying microservices requires
  - Supporting technologies in operations like Kubernetes, Kafka, Docker
  - Analysis and design techniques for building a component based software architecture
  - Code and application design techniques to make code “microservices ready”
  - Production techniques to support the successful deployment of a microservice
- An integral part of microservices development is the use of CI/CD

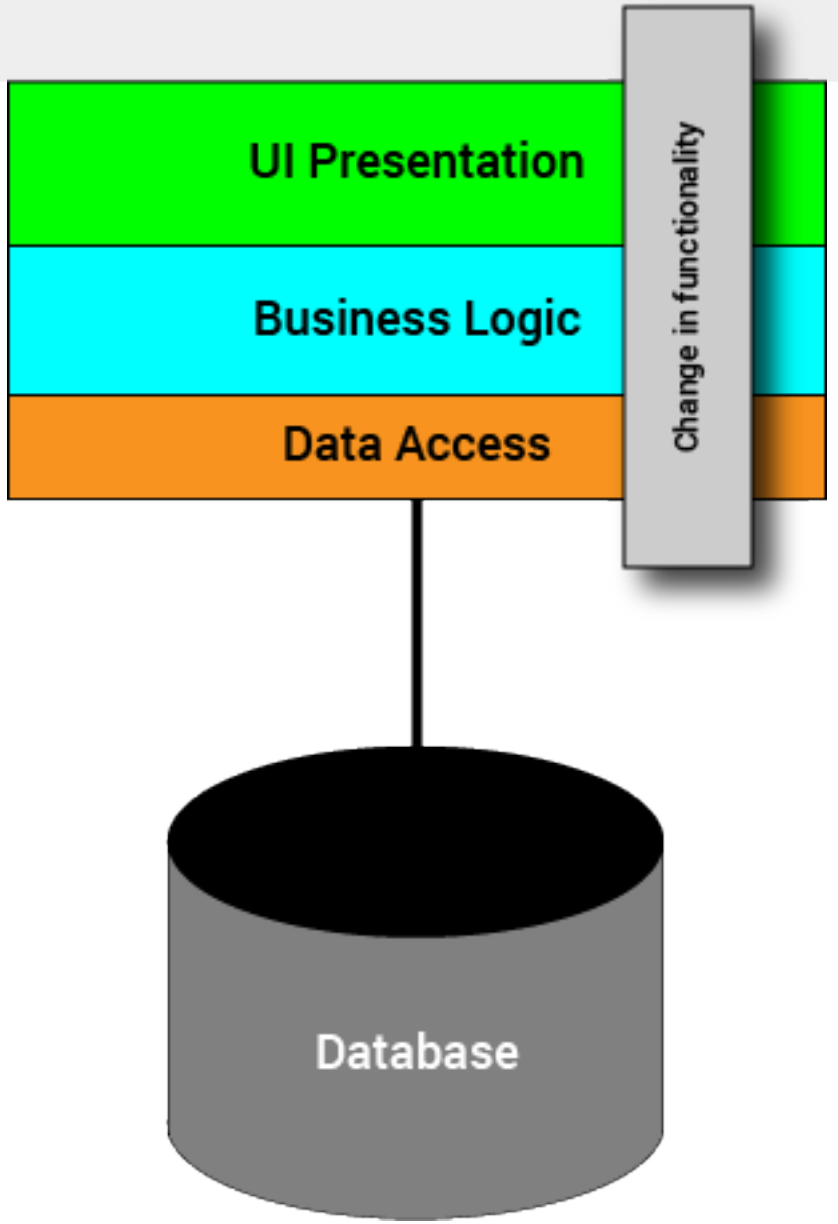
Monoliths

- Characterized by a single code base
  - May be modular at the programming language level
  - Integrated with a single database
  - All code uses a common schema
- “Monolith” means
  - If a change is made to the code base or to the data schema
  - Then the entire application needs to be redeployed



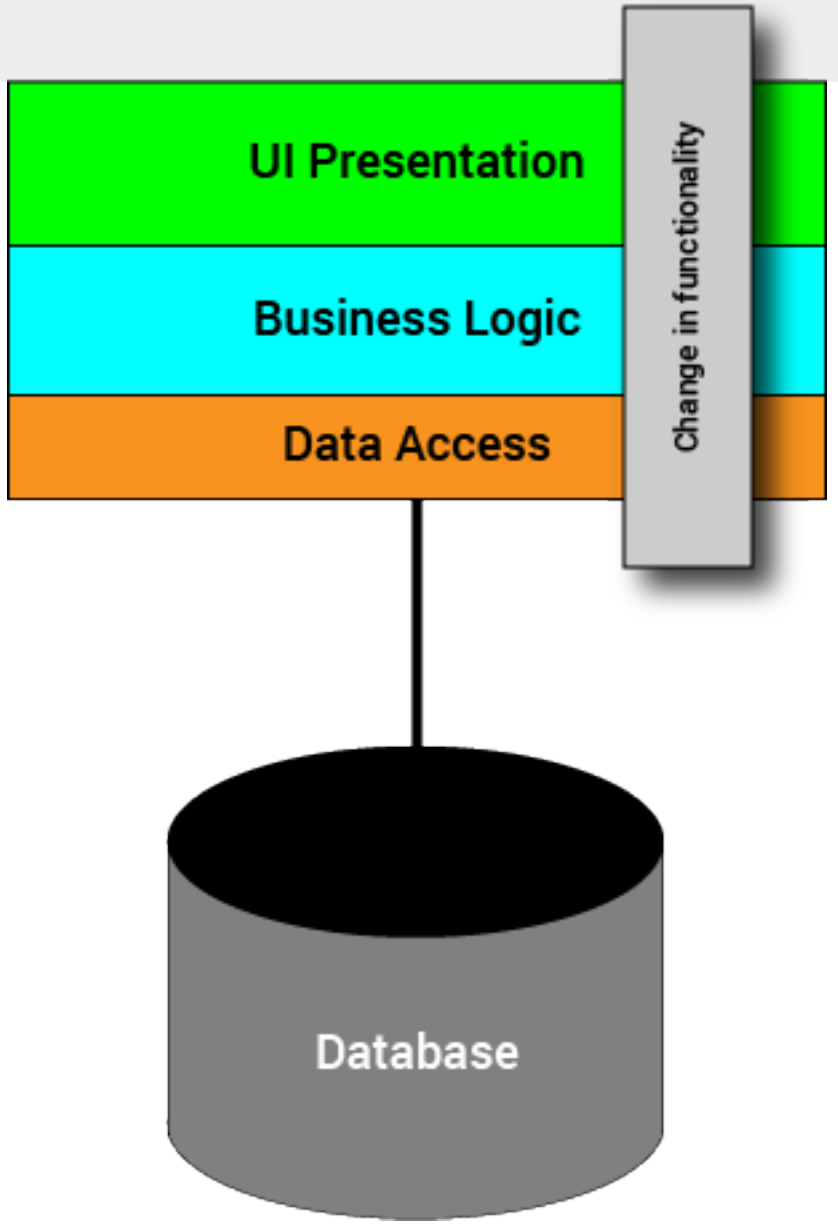
Modular Monolith

- The application is still a monolith
- A change in functionality
  - Affects each layer because related changes have to be made in each layer
- The data model may constrain what changes can be made
  - Changing the data model might break other parts of the app
  - The modules and the database often show high coupling
- This limits the scaling of the application due to computational costs (hardware)



Modular Monolith

- The application is still a monolith
- A change in functionality
  - Affects each layer because related changes have to be made in each layer
- The data model may constrain what changes can be made
  - Changing the data model might break other parts of the app
  - The modules and the database often show high coupling
- This limits the scaling of the application due to computational costs (hardware)



Scaling in Systems

- Scaling is an increase in size or quantity along some dimension
  - Can take place in the development or operations space
- Development scaling
  - Increase in complexity, functionality or volume of code
  - These dimensions are often related
  - Business analogy is a company increasing the range of services and products they offer or expanding into different markets (like a Canadian company expanding into Europe)
- Operational scaling
  - Increase in the amount of activity of a system
  - Throughput, load, transaction time, simultaneous users, etc
  - Traditional monoliths tend to be scalable only to a limited degree
  - There is a certain level of complexity after which they become unmaintainable

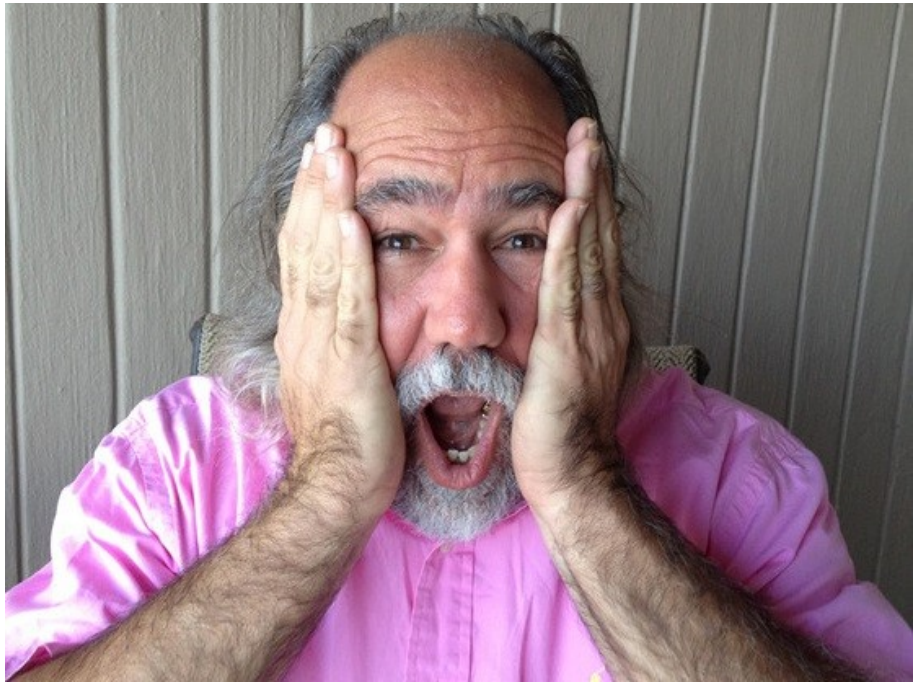
Mission Critical Industrial Strength Software

Mission critical software tends to have a long lifespan, and over time, many users come to depend on their proper functioning. In fact, the organization becomes so dependent on the software that it can no longer function in its absence. At this point, we can say the software has become industrial-strength.

The distinguishing characteristic of industrial strength software is that it is intensely difficult, if not impossible, for the individual developer to comprehend all of the subtleties of its design.

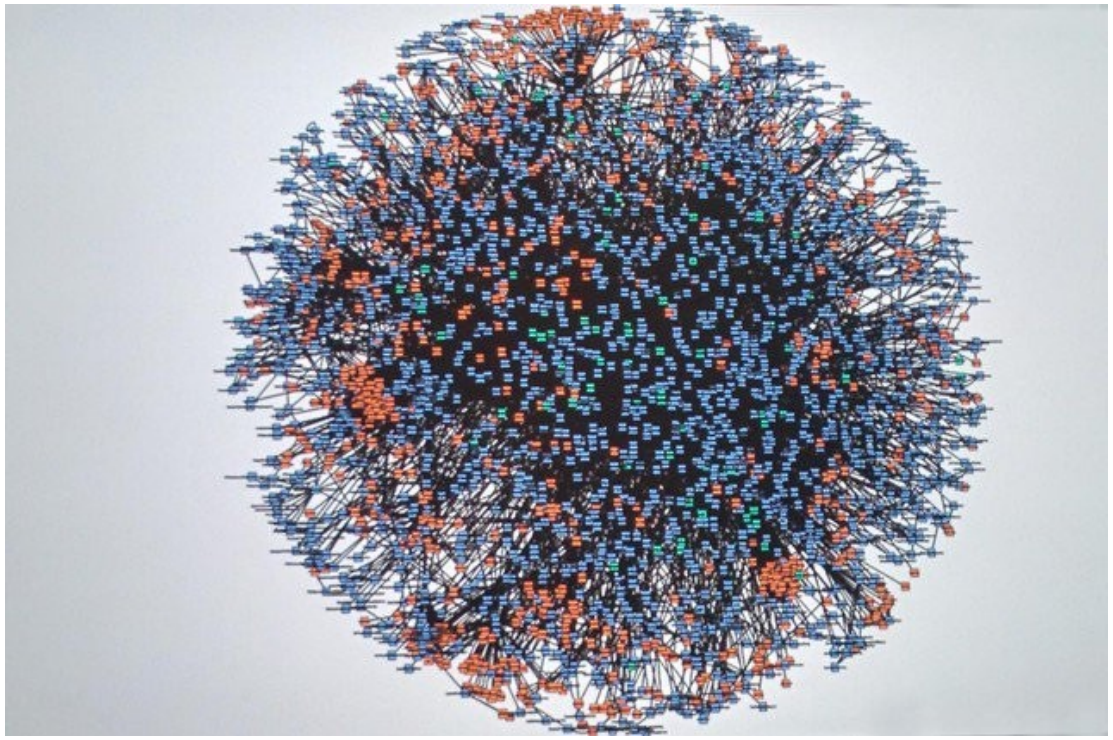
Stated in blunt terms, the complexity of such systems exceeds the human intellectual capacity. Alas, this complexity we speak of seems to be an essential property of all large software systems. By 'essential' we mean that we may master this complexity, but we can never make it go away.

Grady Booch



Operational Complexity

- Modern applications have to deal with
  - Petabytes of streaming data
  - Billions of transaction
  - Mission critical fault tolerance
- Non-functional requirements
  - Throughput, response time, loading, stress
  - Disaster recovery, transaction time
- Results in a “death star” architecture
  - Image: Amazon in 2008
  - The complexity of the operational architecture is now industrial strength



IT Failures and Complexity

The United States is losing almost as much money per year to IT failure as it did to the [2008] financial meltdown. However the financial meltdown was presumably a onetime affair. The cost of IT failure is paid year after year, with no end in sight. These numbers are bad enough, but the news gets worse. According to the 2009 US Budget [02], the failure rate is increasing at the rate of around 15% per year.

Is there a primary cause of these IT failures? If so, what is it?.... The almost certain culprit is complexity.... Complexity seems to track nicely to system failure.

Once we understand how complex some of our systems are, we understand why they have such high failure rates.

We are not good at designing highly complex systems. That is the bad news. But we are very good at architecting simple systems. So all we need is a process for making the systems simple in the first place.

Roger Sessions

Microservices

"The Microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API."

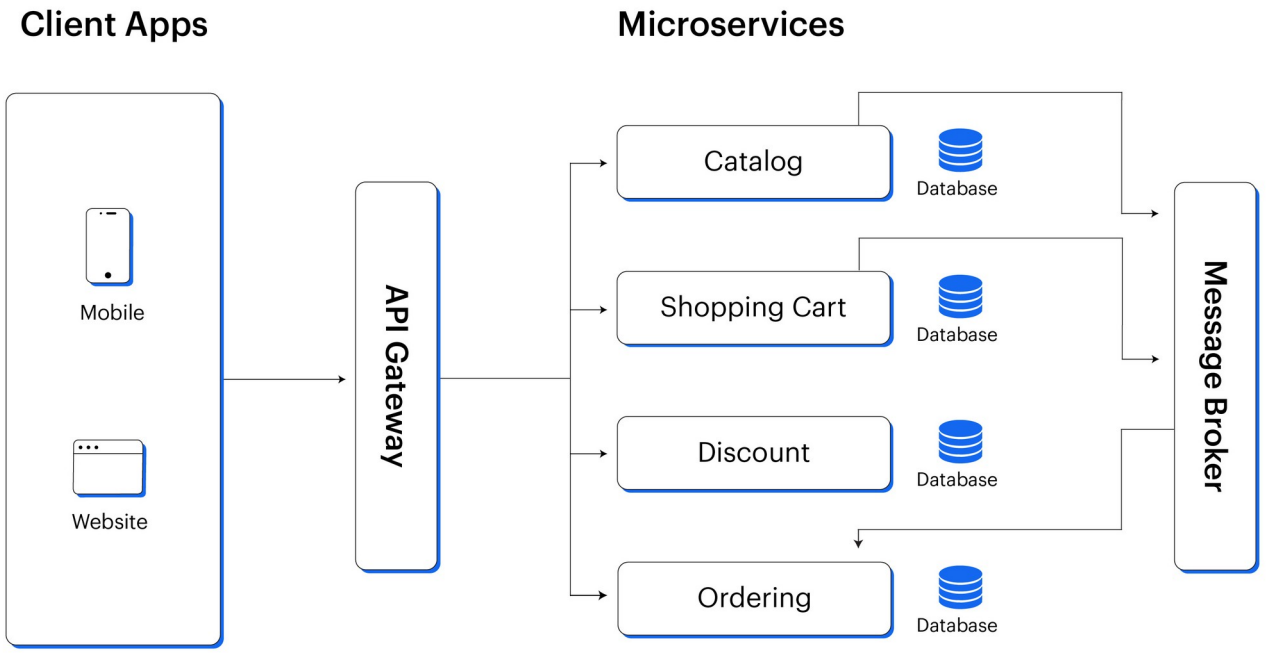


Image Credit: https://webandcrafts.com/blog/what-is-microservices-architecture

Microservices Drivers

Modeling Microservices

- A core principle of microservices is that they model a business domain or a domain of activity
- These domains are often complex and wide ranging in scope
- Very often, they have evolved a domain structure that is
  - A hierarchy of sub-domain layers
  - The structure is recursive
  - Each layer is made up of modular components
  - The components work together by sending well-defined messages
  - Messages are sent through interfaces
- Components are cohesive and loosely coupled
  - This is a sort of idealized version of services “in the wild”
  - There are many factors that can make this organization dysfunctional

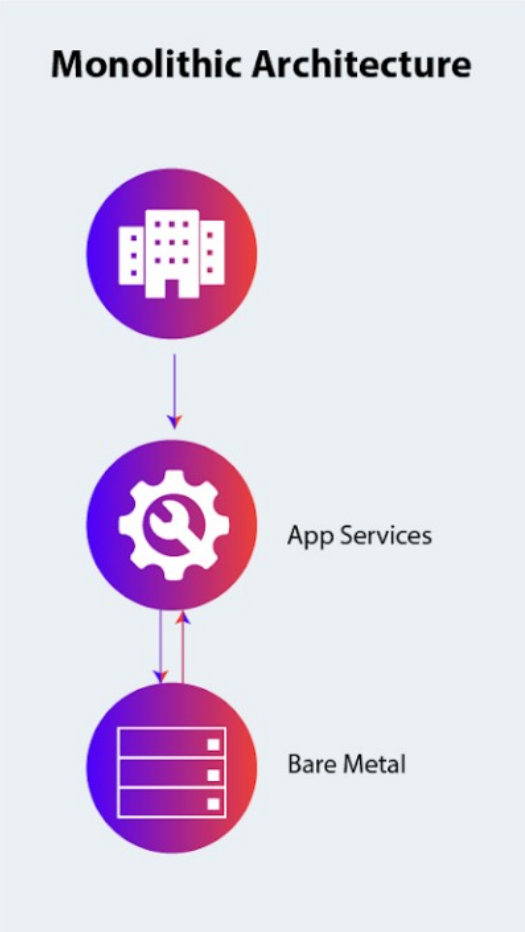
Modeling Microservices

- A core principle of microservices is that they model a business domain or a domain of activity
- These domains are often complex and wide ranging in scope
- Very often, they have evolved a domain structure that is
  - A hierarchy of sub-domain layers
  - The structure is recursive
  - Each layer is made up of modular components
  - The components work together by sending well-defined messages
  - Messages are sent through interfaces
- Components are cohesive and loosely coupled
  - This is a sort of idealized version of services “in the wild”
  - There are many factors that can make this organization dysfunctional

Modeling Microservices

- A core principle of microservices is that they model a business domain or a domain of activity
- These domains are often complex and wide ranging in scope
- Very often, they have evolved a domain structure that is
  - A hierarchy of sub-domain layers
  - The structure is recursive
  - Each layer is made up of modular components
  - The components work together by sending well-defined messages
  - Messages are sent through interfaces
- Components are cohesive and loosely coupled
  - This is a sort of idealized version of services “in the wild”
  - There are many factors that can make this organization dysfunctional

Monolith to Microservice



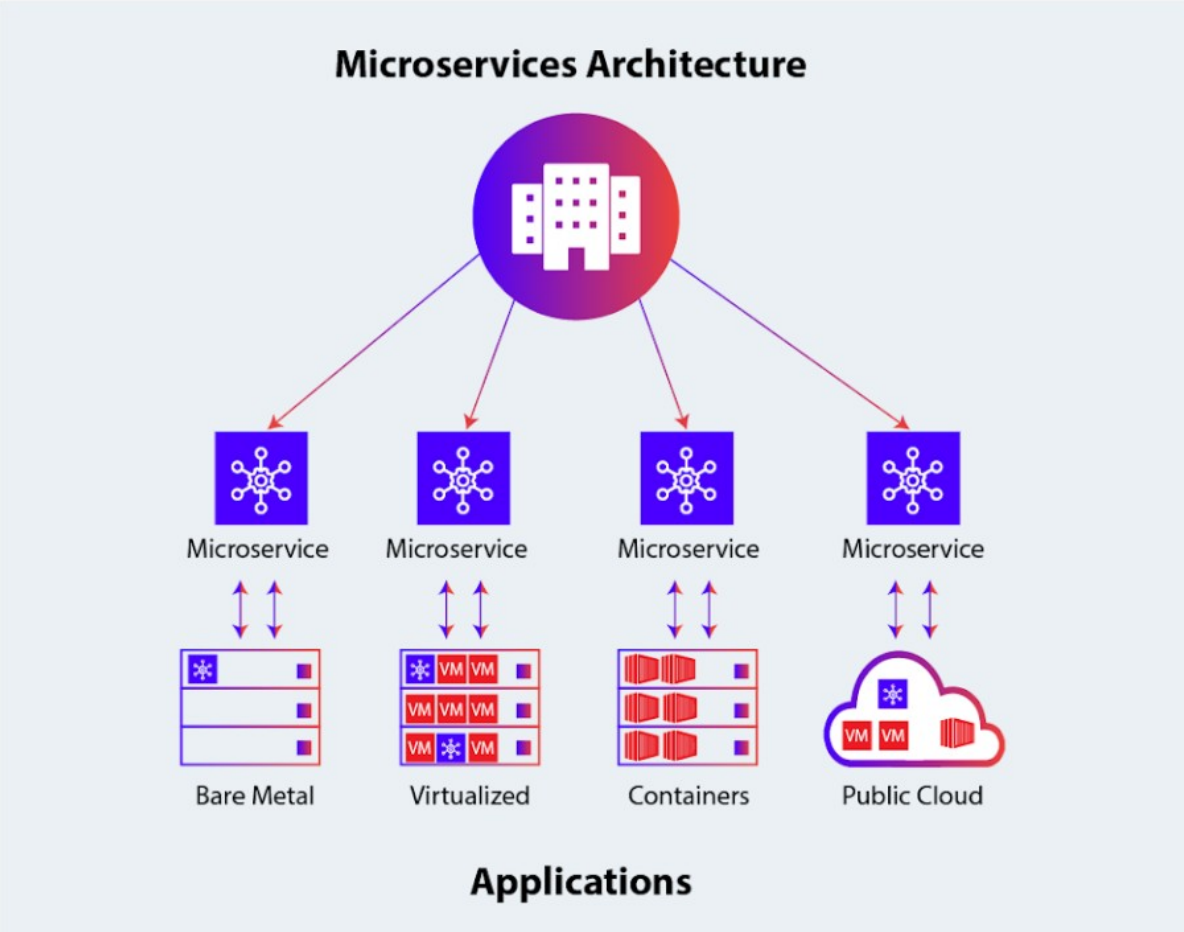


Image Credit: https://www.appviewx.com/education-center/microservices/

The 12 Factor App Methodology

- Methodology for building micro-services apps (software as a service)
  - Drafted by developers at Heroku
  - First presented by Adam Wiggins circa 2011
  - Motivated by the problem of scaling the codebase and the operational environment
- Intended to apply to building applications that:
  - Use declarative formats for setup automation, like containers or VMs
  - Have a clean contract with the underlying operating system
  - Are suitable for deployment on modern cloud platforms
  - Minimize divergence between development and production (DevOps)
  - Scale up without significant changes to tooling, architecture, or development practices
-

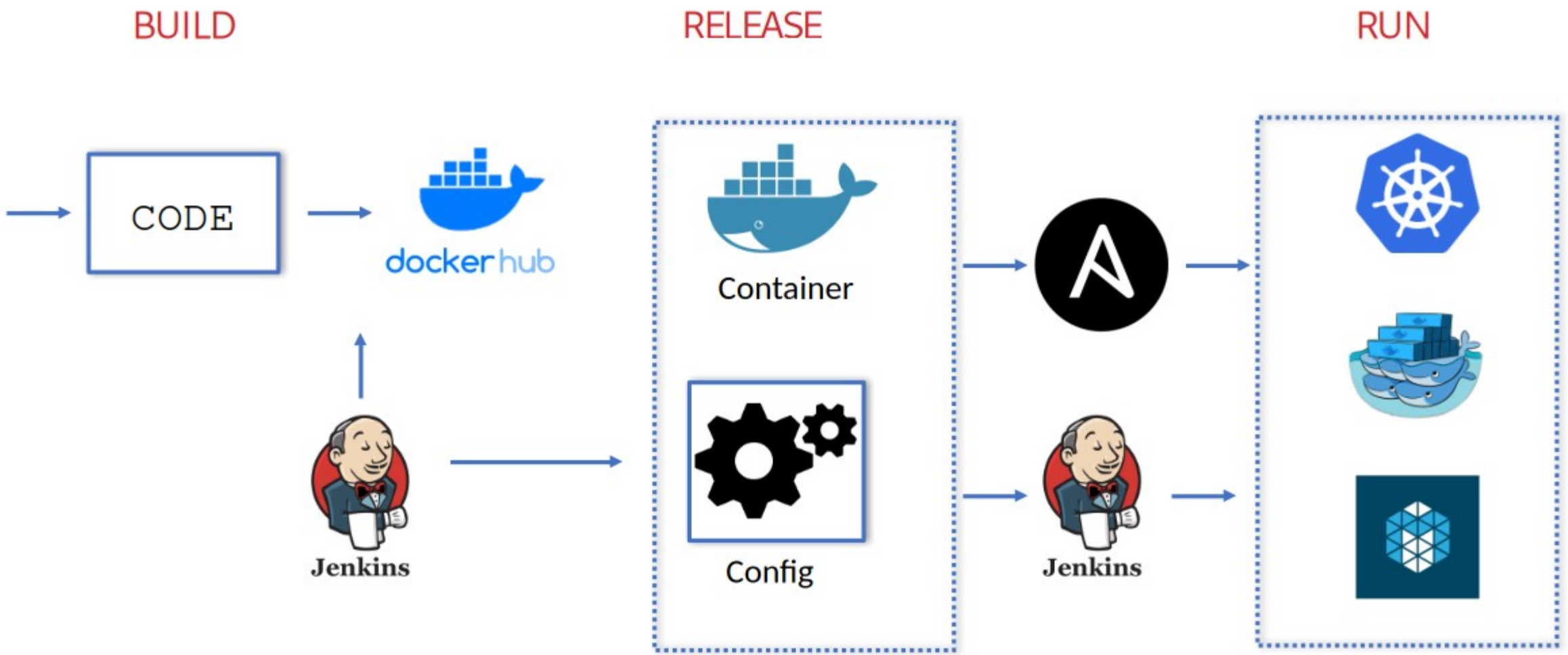
12 Factor Overview

Factor	Description
I. Codebase	One codebase tracked in revision control, many deploys
II. Dependencies	Explicitly declare and isolate dependencies
III. Config	Store config in the environment
IV. Backing services	Treat backing services as attached resources
V. Build, release, run	Strictly separate build and run stages
VI. Processes	Execute the app as one or more stateless processes
VII. Port binding	Export services via port binding
VIII. Concurrency	Scale out via the process model
IX. Disposability	Maximize robustness with fast startup and graceful shutdown
X. Dev/prod parity	Keep development, staging, and production as similar as possible
XI. Logs	Treat logs as event streams
XII. Admin processes	Run admin/management tasks as one-off processes

V. Build, Release and Run

- A codebase is transformed into a deploy through three stages:
  - Build: converts a code repo into an executable bundle known as a build
  - Release: combines a build with the deploy's current config so that the release is ready for immediate execution
  - Run: runs the app in the execution environment
- These stages are strictly separated and distinct
- These stages are the CI/CD pipeline

V. Build, Release and Run



The Importance of CI/CD

- Modern software architectures can be massive and complex
  - Enabled by the rise in computational power and scaling requirements
  - Traditional software development cannot meet these requirements
- CI/CD, Agile and DevOps meet this modern environment
  - Each provides a part of the solution
  - CI/CD and automation are the glue that holds it all togethers

Questions