

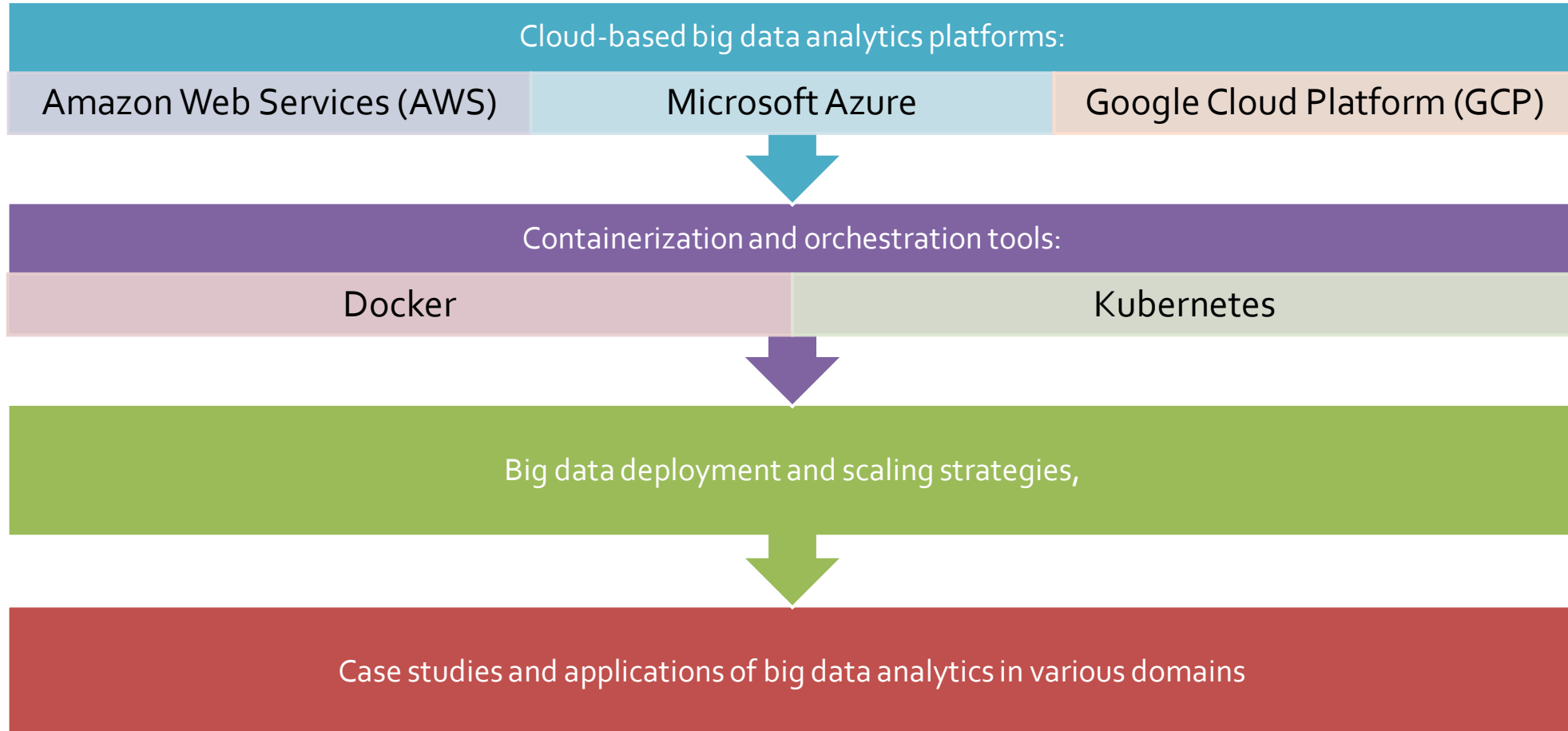


# SCALABLE BIG DATA ANALYTICS PLATFORMS

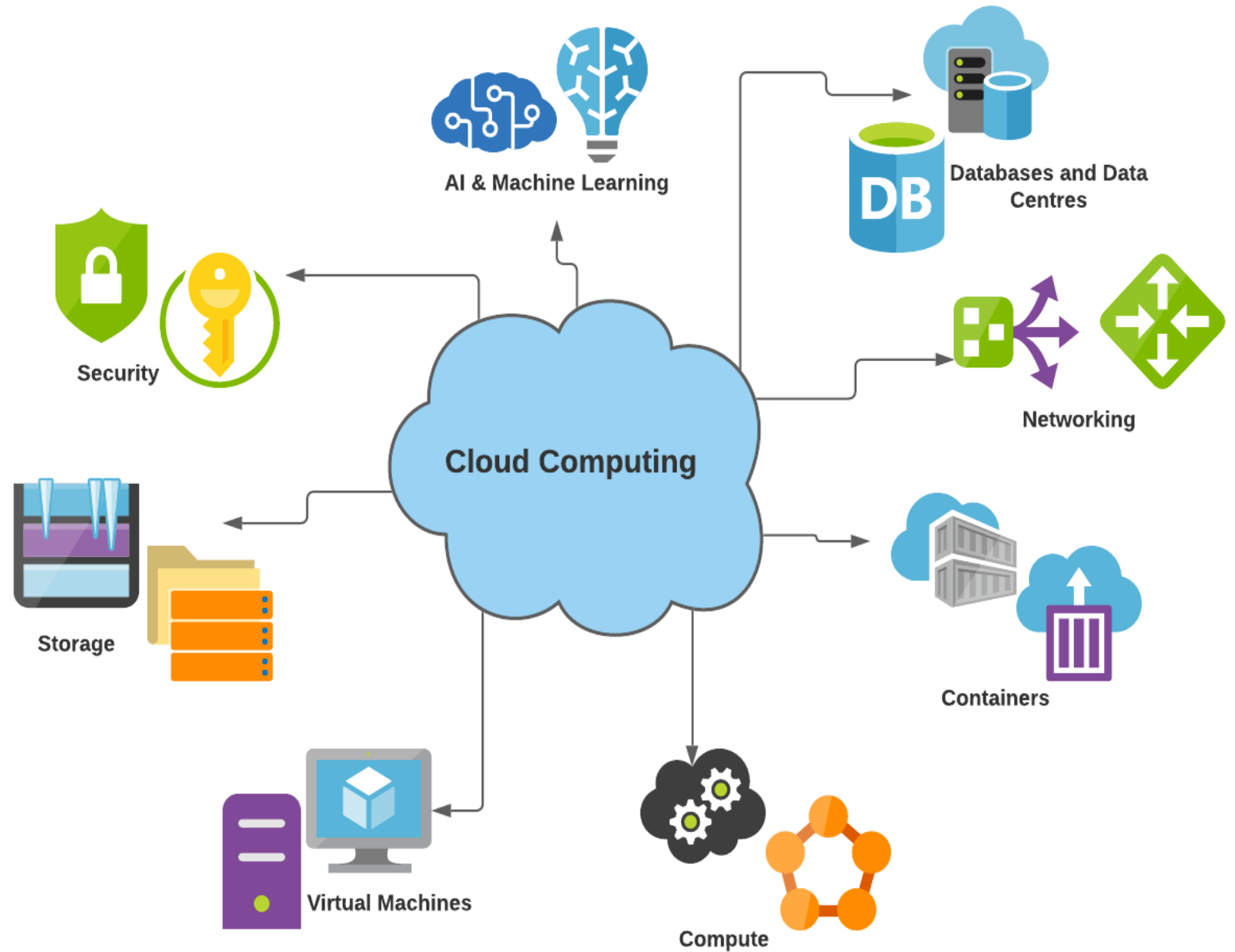
UNIT 5



# CONTENT



# CLOUD COMPUTING



# CLOUD COMPUTING

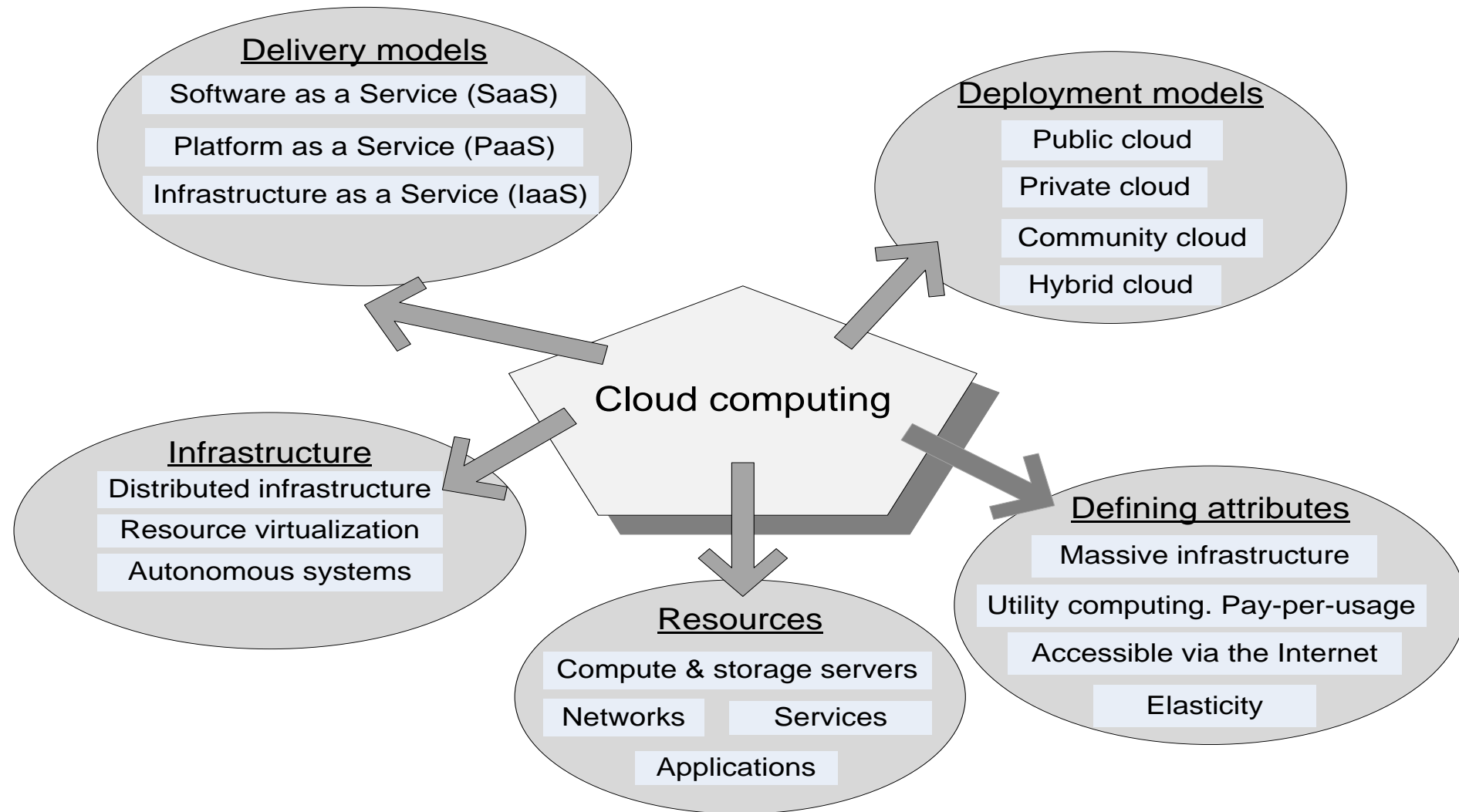


*cloud computing is the delivery of computing services – servers, storage, databases, networking, software, analytics and more – over the Internet (“the cloud”).*



*Companies offering these computing services are called cloud providers and typically charge for cloud computing services based on usage, like how you’re billed for gas or electricity at home.*

# Cloud Computing Models, Resources, Attributes



# Cloud computing - Characteristics



*"Cloud Computing offers on-demand, scalable and elastic computing (and storage services)."*



*The resources used for these services can be metered and users are charged only for the resources used.*

# Cloud computing (cont' d)

## Shared Resources and Resource Management:

- Cloud uses a shared pool of resources
- Uses Internet techn. to offer **scalable** and **elastic** services.
- The term “**elastic computing**” refers to the ability of **dynamically** and **on-demand** acquiring computing resources and supporting a variable workload.
- Resources are metered and users are charged accordingly.
- It is more cost-effective due to **resource-multiplexing**. Lower costs for the cloud service provider are passed to the cloud users.

## Data Storage:

- Data is stored:
  - in the “cloud”, in certain cases closer to the site where it is used.
  - appears to the users as if stored in a location-independent manner.
- The data storage strategy can increase reliability, as well as security, and can lower communication costs.

## Management:

- The maintenance and security are operated by service providers.
- The service providers can operate more efficiently due to specialization and centralization.

aws





# Amazon Web Services (AWS)



**A comprehensive cloud computing platform provided by Amazon, offering a broad range of services for**

Computing

Storage

Networking

Machine learning

Analytics

databases, and more



**Launched in 2006, AWS provides on-demand cloud infrastructure and resources, allowing individuals, businesses, and organizations to access and scale IT infrastructure without the need for physical hardware or data centres.**

# Key Characteristics of AWS



## On-Demand Resources:

AWS offers resources such as servers, databases, storage, and applications on a pay-as-you-go basis, meaning users only pay for what they use.



## Scalability and Elasticity:

AWS can scale resources up or down based on demand, making it suitable for varying workloads, from small-scale startups to large enterprises.



## Global Reach:

AWS has data centers worldwide, allowing users to deploy applications closer to their customers, improving latency and performance.



## Comprehensive Service Offerings:

AWS provides over 200 fully featured services, covering areas such as computing, storage, databases, machine learning, analytics, IoT, security, and more.



## Security and Compliance:

AWS prioritizes security with features like data encryption, identity and access management (IAM), and compliance with major regulatory standards (e.g., GDPR, HIPAA).

# Amazon EMR

- Easily run and scale Apache Spark, Hive, Presto, and other big data workloads
- <https://aws.amazon.com/emr/>

# AWS, Azure & GCP

S no	Feature/Aspect	AWS	Azure	GCP
1	Provider	Amazon	Microsoft	Google
2	Launch Year	2006	2010	2008
3	Global Market Share (2023)	Leading (around 32%)	2nd largest (around 22%)	3rd largest (around 10%)
4	Compute Services	EC2 (Elastic Compute Cloud)	Virtual Machines (VMs)	Compute Engine
5	Storage	S3 (Simple Storage Service), EBS	Blob Storage, Disk Storage	Cloud Storage, Persistent Disks
6	Networking	VPC (Virtual Private Cloud)	Virtual Network	Virtual Private Cloud (VPC)
7	Database Services	RDS, DynamoDB, Aurora, Redshift	SQL Database, Cosmos DB, MySQL	Cloud SQL, Bigtable, Firestore
8	AI & Machine Learning	SageMaker, Lex, Polly, Deep Learning AMLs	Azure Machine Learning, Cognitive Services	AI Platform, TensorFlow, AutoML
9	Big Data	EMR (Elastic MapReduce), Redshift	HDInsight, Synapse Analytics	BigQuery, Dataproc
10	Serverless	Lambda	Azure Functions	Cloud Functions

# AWS, Azure & GCP

S no	Feature/Aspect	AWS	Azure	GCP
11	DevOps Tools	CodeBuild, CodeDeploy, CodePipeline	Azure DevOps, GitHub Actions	Cloud Build, Cloud Source Repositories
12	Containers	ECS, EKS (Elastic Kubernetes Service)	Azure Kubernetes Service (AKS)	Google Kubernetes Engine (GKE)
13	Hybrid Cloud	Outposts, VMware Cloud on AWS	Azure Arc, Azure Stack	Anthos
14	Pricing Model	Pay-as-you-go, Reserved Instances, Spot Instances	Pay-as-you-go, Reserved Instances, Hybrid Benefit	Pay-as-you-go, Sustained use discounts
15	Compliance & Security	Extensive certifications, IAM, KMS	Extensive certifications, Azure Active Directory	Extensive certifications, IAM, KMS
16	Enterprise Adoption	Widely adopted across industries	Strong presence in enterprises (Microsoft ecosystem)	Strong in AI/ML and data-heavy industries
17	Developer Ecosystem	Large community, extensive documentation	Strong integration with Microsoft tools	Strong community with a focus on data science
18	Free Tier	12-month free tier, always free offers	12-month free tier, always free offers	12-month free tier, always free offers
19	Hybrid Cloud Focus	Limited compared to Azure	Strong focus with Azure Stack	Strong focus with Anthos
20	Customer Support	24/7 support with various plans	24/7 support with various plans	24/7 support with various plans

# Containerization

---



The process of packaging an application and all its dependencies into a single, lightweight, executable unit called a **container**.



Containers are self-contained and run consistently across different environments, making them portable and efficient.



A container includes the application code, runtime, libraries, environment variables, and configuration files necessary to run the application.



Unlike virtual machines (VMs), containers share the host operating system's kernel, making them faster to start and more resource-efficient.

# Benefits of Containerization

---

## Portability:

- Containers run the same way across different environments (developer laptops, on-premises servers, or cloud).

## Consistency:

- Containers ensure that applications run with the same configurations and dependencies, reducing "it works on my machine" issues.

## Efficiency:

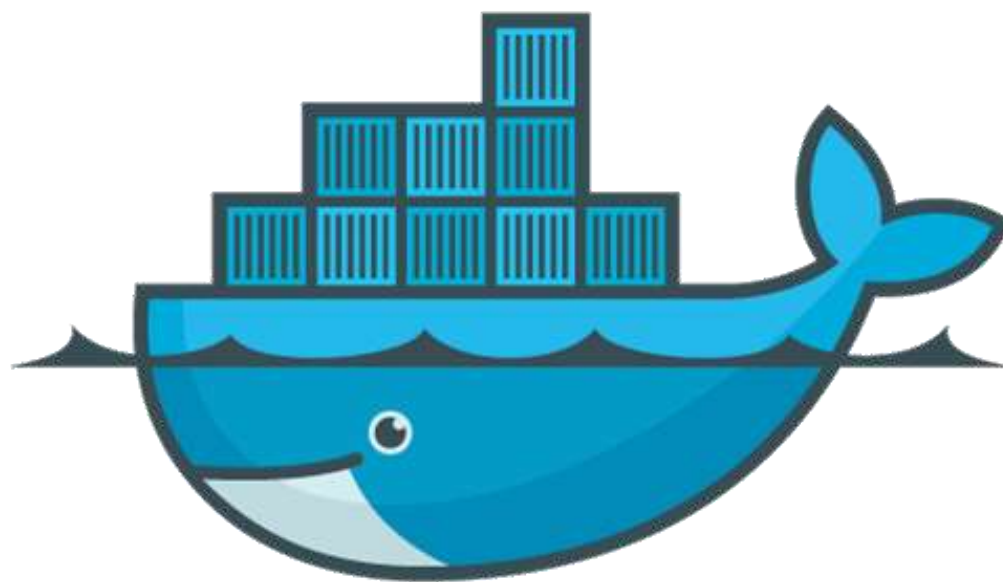
- Containers share resources and start up quickly, making them more lightweight and cost-effective than traditional VMs.

## Isolation:

- Each container operates in its own isolated environment, allowing multiple applications to run on the same machine without conflicts.

## Example

- Docker is a popular tool for containerization, allowing developers to build and deploy containers consistently and efficiently.



**docker**  
**www.docker.io**

<https://pointful.github.io/docker-intro/#/20>



# Since it started in March 2013...

200,000 pulls

7,500 github stars

200 significant contributors

200 projects built on top of docker

- UIs, mini-PaaS, Remote Desktop...

1000's of Dockerized applications

- Memcached, Redis, Node.js, Hadoop...


Integration in Jenkins, Travis, Chef, Puppet, Vagrant and OpenStack


Meetups arranged around the world...

- with organizations like Ebay, Cloudflare, Yandex, and Rackspace presenting on their use of Docker

# The Challenge


Multiplicity of Stacks

 Static website  
nginx 1.5 + modsecurity + openssl + bootstrap 2

 Background workers  
Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs

 User DB  
postgresql + pgv8 + v8

 Web frontend  
Ruby + Rails + sass + Unicorn

 Queue  
Redis + redis-sentinel

 Analytics DB  
hadoop + hive + thrift + OpenJDK

 API endpoint  
Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client

Do services and apps  
interact  
appropriately?

Multiplicity of  
hardware  
environments

 Development VM

 QA server

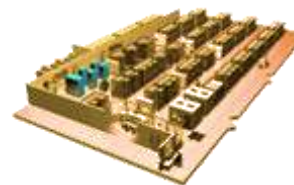
Customer Data Center



  
Public Cloud

Disaster recovery

Production Servers



Production Cluster





Contributor's laptop



Can I migrate  
smoothly and  
quickly?

# The Matrix from Hell

	Static website	?	?	?	?	?	?	?
	Web frontend	?	?	?	?	?	?	?
	Background workers	?	?	?	?	?	?	?
	User DB	?	?	?	?	?	?	?
	Analytics DB	?	?	?	?	?	?	?
	Queue	?	?	?	?	?	?	?
		Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
								

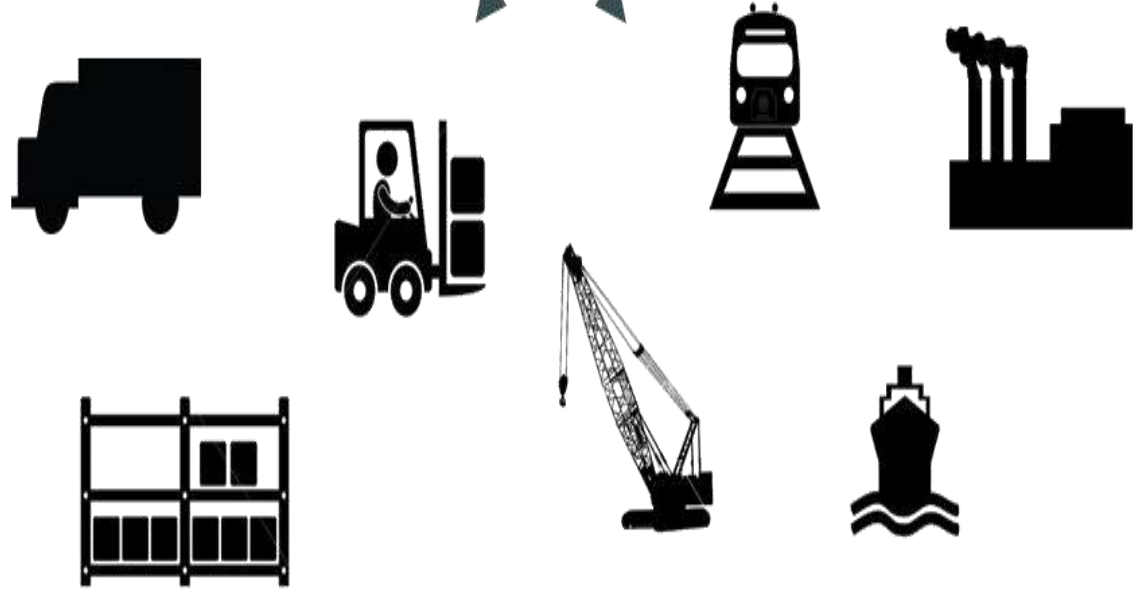
# Let's create and analogy :Cargo Transport Pre- 1960

Multiplicity of Goods










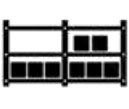





Do I worry about  
how goods interact  
(e.g. coffee beans  
next to spices)

Multiplicity of  
methods for  
transporting/storing



Can I transport quickly  
and smoothly  
(e.g. from boat to train  
to truck)

Also a Matrix  
from Hell

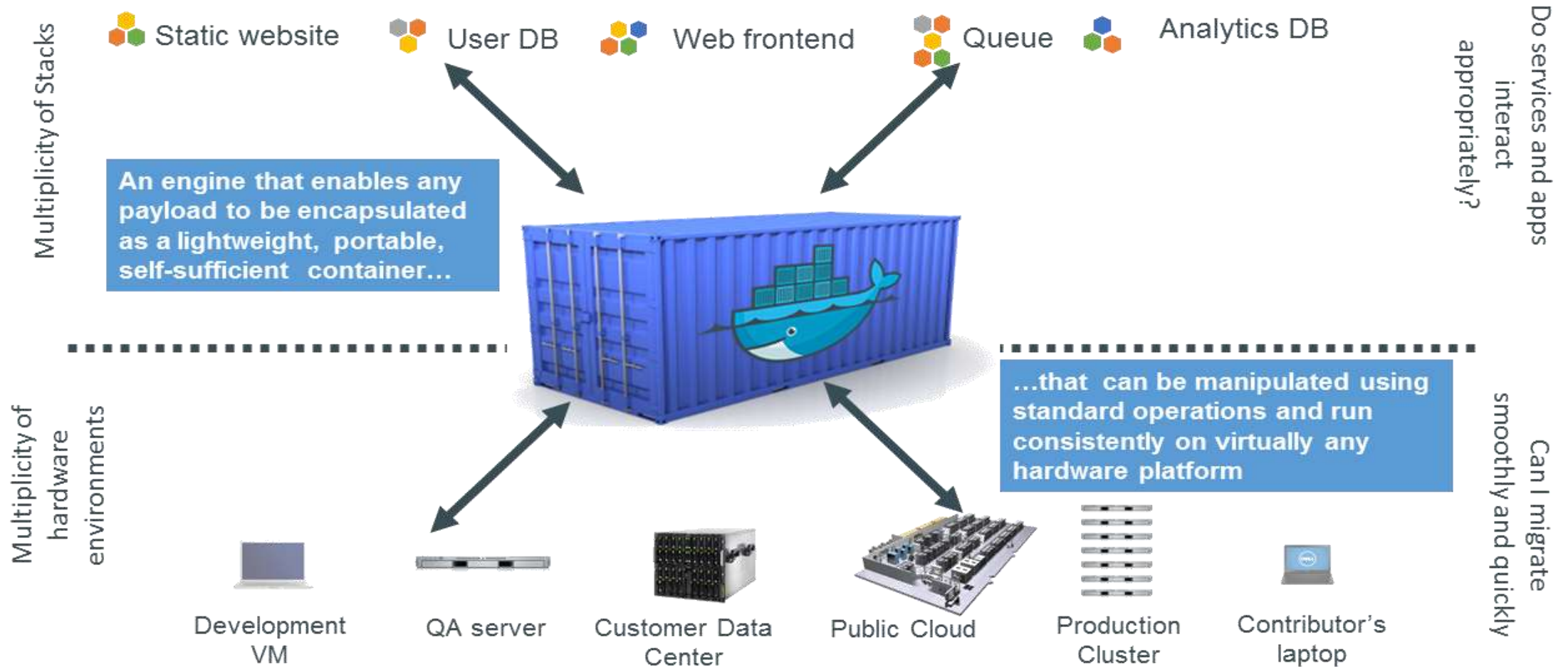
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							



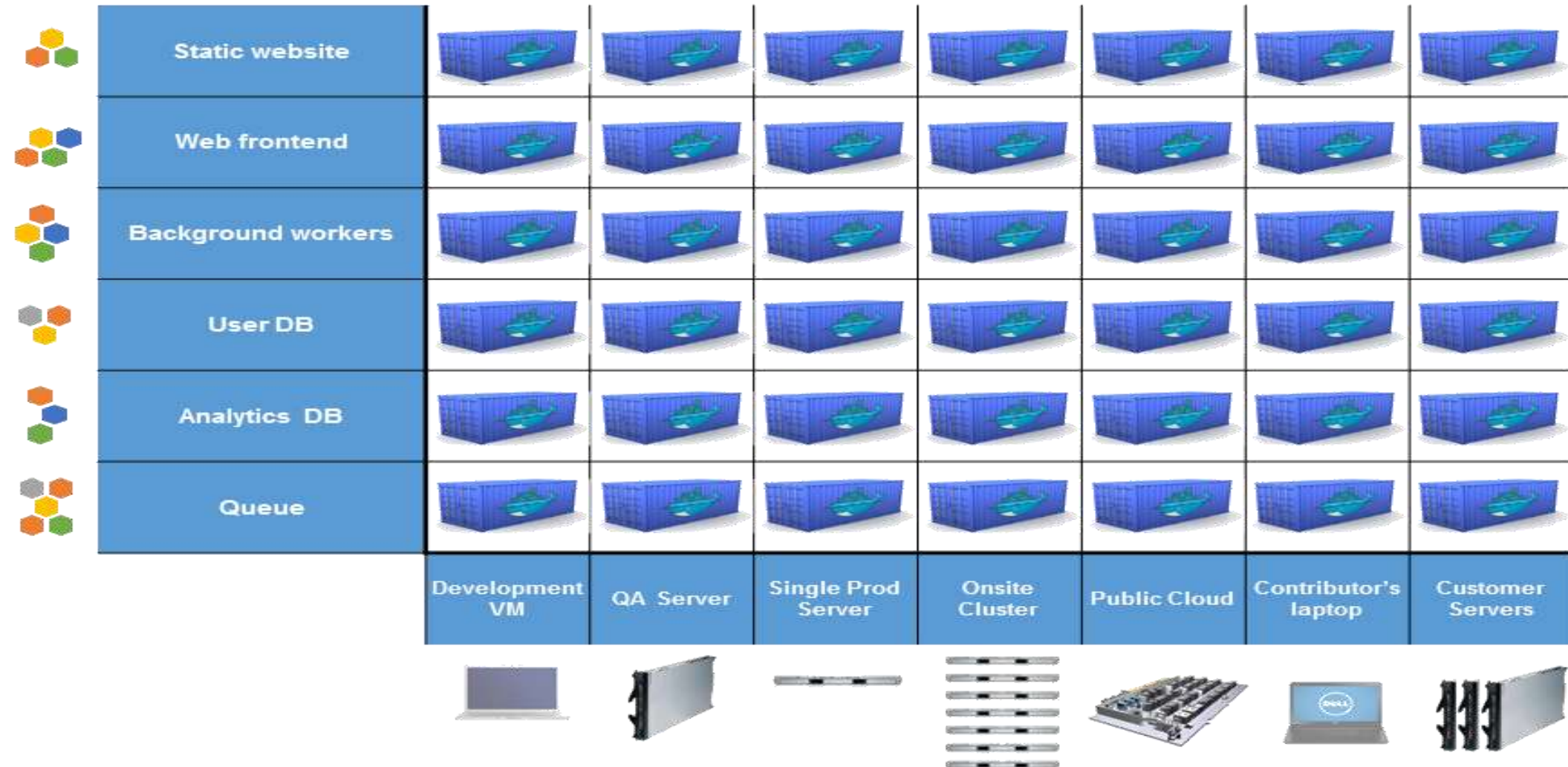
# Solution: Intermodal Shipping Container



# Docker is a Container System for Code



# Docker Eliminates the Matrix from Hell





# Why Developers Care

---

Build once... (finally) run *anywhere*

---

A clean, safe, hygienic, portable runtime environment for your app.

---

No worries about missing dependencies, packages and other pain points during subsequent deployments.

---

Run each app in its own isolated container, so you can run various versions of libraries and other dependencies for each app without worrying.

---

Automate testing, integration, packaging...anything you can script.

---

Reduce/eliminate concerns about compatibility on different platforms, either your own or your customers.

---

Cheap, zero-penalty containers to deploy services. A VM without the overhead of a VM. Instant replay and reset of image snapshots.

# Why Administrators Care

---

Configure once... run anything

---

Make the entire lifecycle more efficient, consistent, and repeatable

---

Increase the quality of code produced by developers.

---

Eliminate inconsistencies between development, test, production, and customer environments.

---

Support segregation of duties.

---

Significantly improves the speed and reliability of continuous deployment and continuous integration systems.

---

Because the containers are so lightweight, address significant performance, costs, deployment, and portability issues normally associated with VMs.

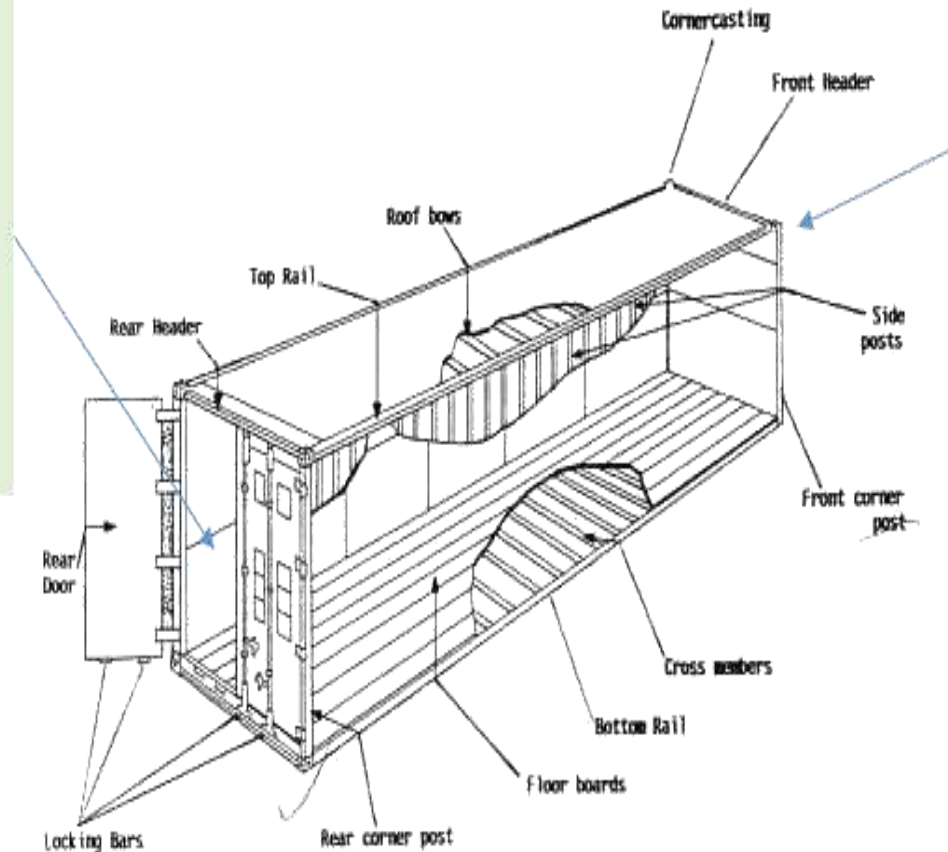
# Why it Works: Separation of Concerns

- **Dan the Developer**

- Worries about what's "inside" the container
  - His code
  - His Libraries
  - His Package Manager
  - His Apps
  - His Data
- All Linux servers look the same

- **Oscar the Ops Guy**

- Worries about what's "outside" the container
  - Logging
  - Remote access
  - Monitoring
  - Network config
- All containers start, stop, copy, attach, migrate, etc. the same way



Major components of the container:

# More Technical Details

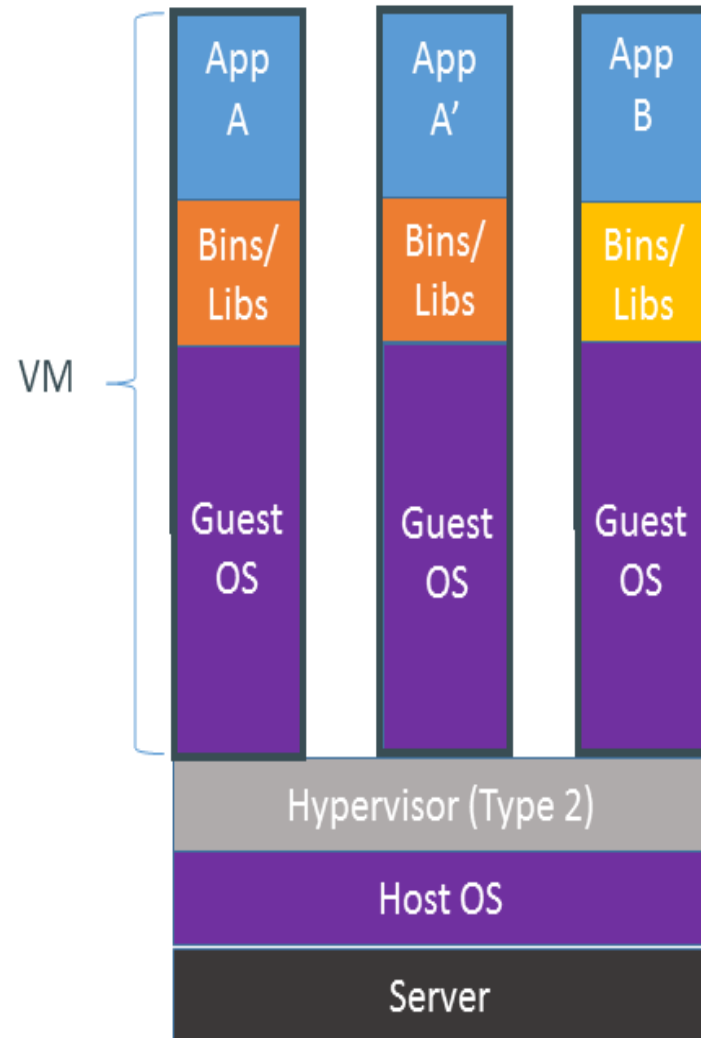
## Why

- Run everywhere
- Regardless of kernel version
- Regardless of host distro
- Physical or virtual, cloud or not
- Container and host architecture must match...
- Run anything If it can run on the host, it can run in the container
- If it can on a Linux kernel, it can run

## What

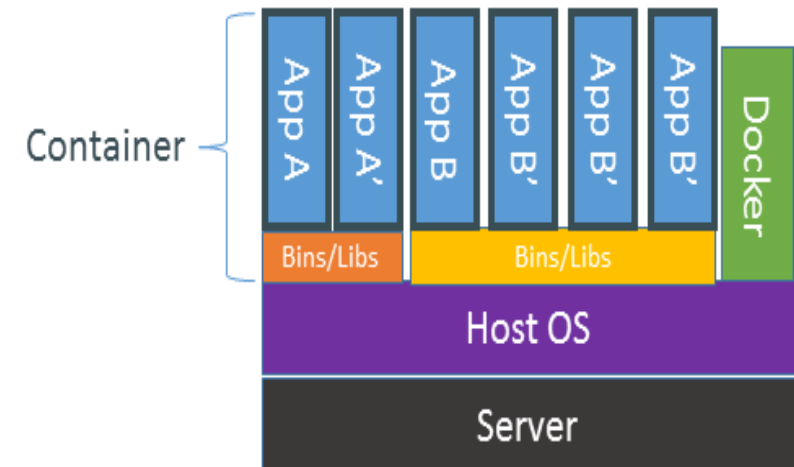
- High level: a lightweight VM Own process space
- Own network interface
- Can run stuff as root
- Can have its own /sbin/init (different from host)
- <<machine container>>
- Low level: chroot on steroids Can also *not* have its own /sbin/init
- Container = isolated processes
- Share kernel with host
- <="" li=""><<application container>>

# VMs vs Containers



Containers are isolated, but share OS and, where appropriate, bins/libraries

...result is significantly faster deployment, much less overhead, easier migration, faster restart



# Why are Docker Containers Lightweight?

## VMs



## VMs

Every app, every copy of an app, and every slight modification of the app requires a new virtual server

## Containers



**Original App**  
(No OS to take up space, resources, or require restart)

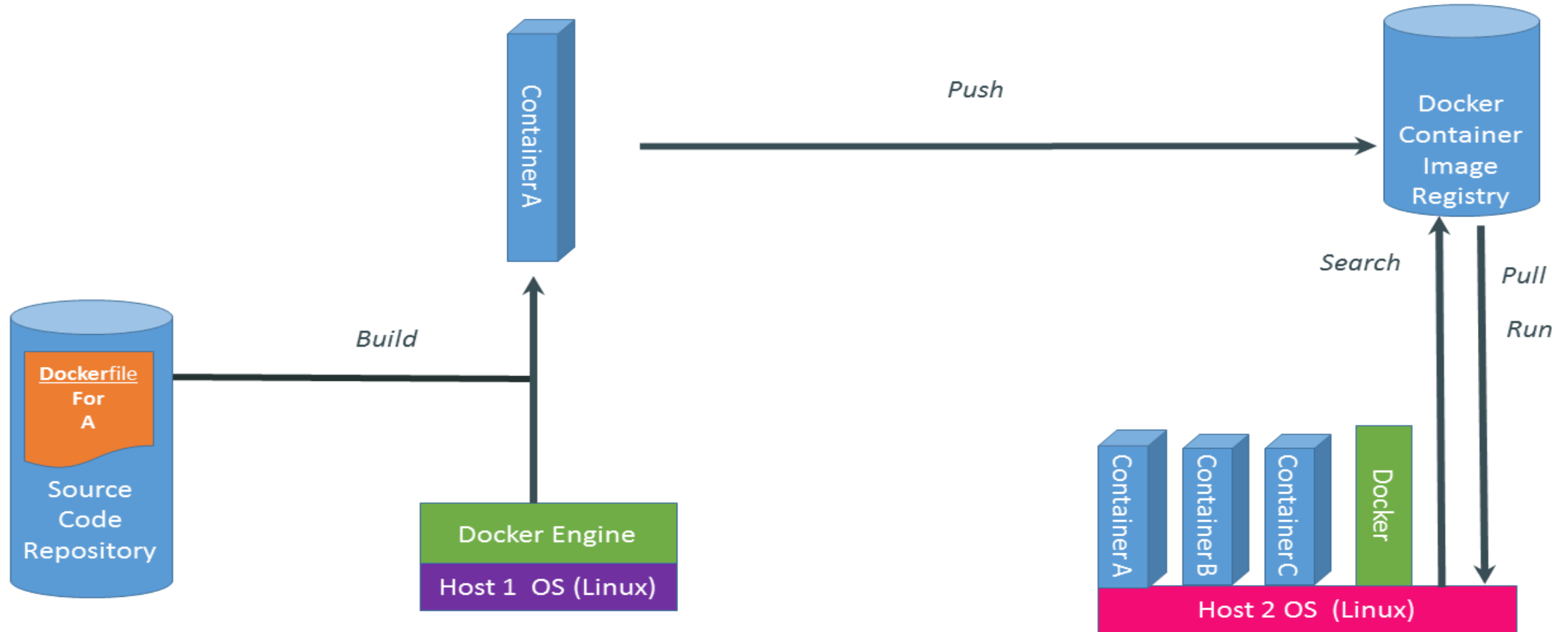


**Copy of App**  
No OS. Can Share bins/libs

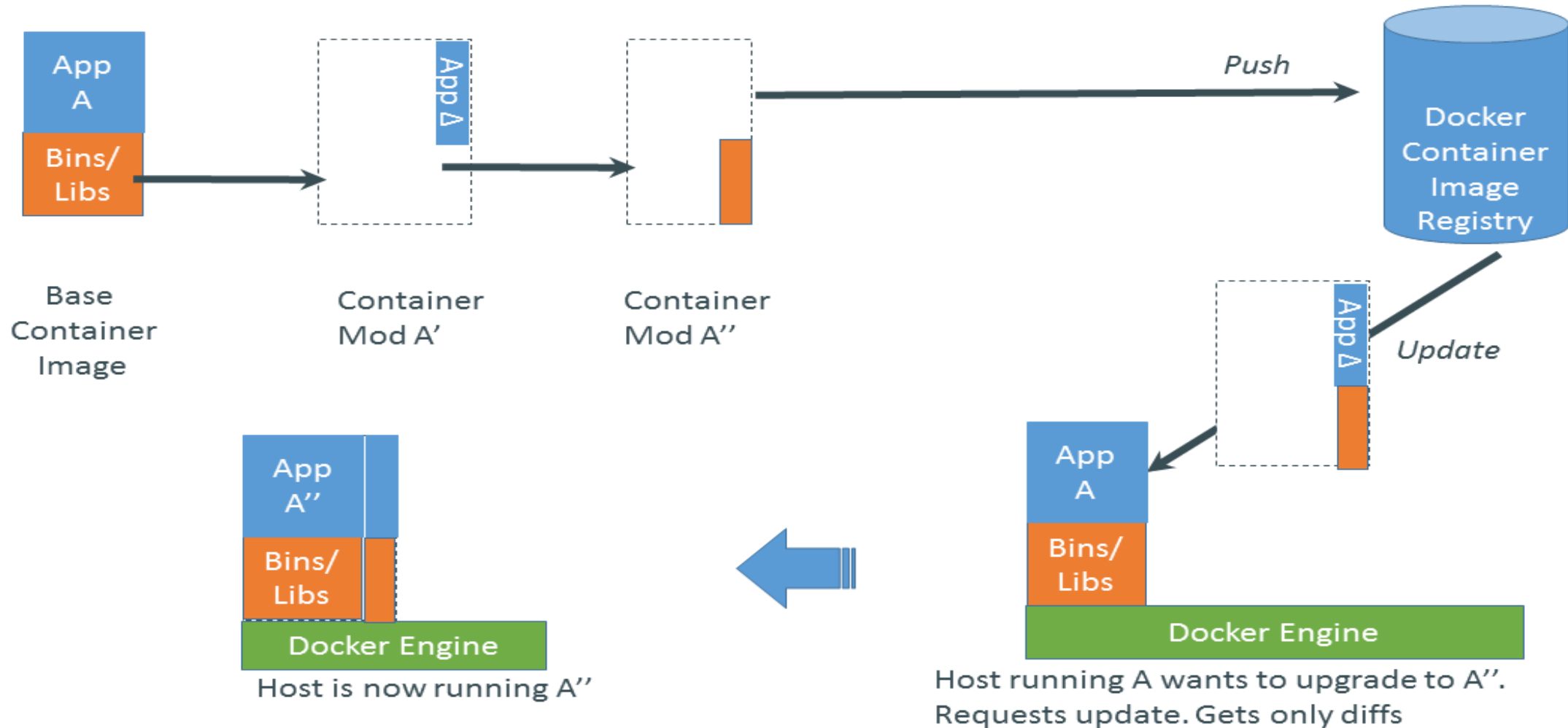


**Modified App**  
Copy on write capabilities allow us to only save the diffs Between container A and container A'

# What are the Basics of a Docker System?



# Changes and Updates





# Ecosystem Support

## Operating systems

- Virtually any distribution with a 2.6.32+ kernel
- Red Hat/Docker collaboration to make work across RHEL 6.4+, Fedora, and other members of the family (2.6.32 +)
- CoreOS—Small core OS purpose built with Docker

## OpenStack

- Docker integration into NOVA (& compatibility with Glance, Horizon, etc.) accepted for Havana release

## Private PaaS

- OpenShift, Solum (Rackspace, OpenStack), Other TBA

## Public PaaS

- Deis, Voxoz, Cocaine (Yandex), Baidu PaaS

# Ecosystem Support

## Public IaaS

- Native support in Rackspace, Digital Ocean,+++
- AMI (or equivalent) available for AWS & other

## DevOps Tools

- Integrations with Chef, Puppet, Jenkins, Travis, Salt, Ansible  
+++

## Orchestration tools

- Mesos, Heat, ++
- Shipyard & others purpose built for Docker

## Applications

- 1000's of Dockerized applications available at [index.docker.io](https://index.docker.io)

# Want to Learn More?



[www.docker.io](http://www.docker.io)

Documentation

Getting started (tutorial,  
installation, guide, etc)

Introductory [whitepaper](#)



Github: [dotcloud/docker](https://github.com/dotcloud/docker)



IRC: freenode  
#docker



Google  
Group: [docker-user](#)



Twitter: [@docker](#)



Meetups: [www.docker.io/meetups](http://www.docker.io/meetups)



kubernetes

# Introduction to kubernetes



Kubernetes, also known as k8s or kube, is an open-source container orchestration platform for scheduling and automating the deployment, management and scaling of containerized applications.



Today, Kubernetes and the broader ecosystem of container-related technologies have merged to form the building blocks of modern cloud infrastructure.



This ecosystem enables organizations to deliver a highly productive hybrid multicloud computing environment to perform complex tasks surrounding infrastructure and operations.



It also supports cloud-native development by enabling a build-once-and-deploy-anywhere approach to building applications.



The word Kubernetes originates from Greek, meaning *helmsman* or *pilot*, hence the helm in the Kubernetes logo (link resides outside of ibm.com).

# Difference between docker and kubernetes

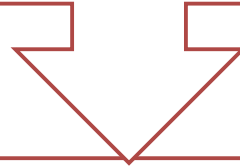
Aspect	Docker	Kubernetes
Purpose	A platform for building, packaging, and distributing containerized applications.	A container orchestration tool for managing, deploying, and scaling containerized applications.
Functionality	Provides tools to create, run, and manage individual containers.	Manages clusters of containers, ensuring high availability, load balancing, and scaling.
Focus	Focuses on containerization: packaging applications with all their dependencies.	Focuses on orchestration: scheduling, scaling, and managing containers in a distributed environment.
Core Component	Docker Engine (runtime) enables running containers.	Uses Docker (or other runtimes) to deploy and manage containers.
Deployment	Primarily for single-container applications or lightweight solutions.	Designed for managing multi-container applications and complex systems (e.g., microservices).
Scaling	Manual scaling of containers.	Automated scaling based on demand using predefined policies.
Networking	Limited networking for linking containers.	Advanced networking with service discovery, load balancing, and inter-container communication.

# Difference between docker and kubernetes

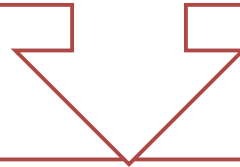
Aspect	Docker	Kubernetes
Self-Healing	Does not provide automated container recovery.	Automatically restarts, replaces, or reschedules containers in case of failures.
Updates	Supports building and running updated containers manually.	Provides rolling updates and rollbacks without downtime.
Portability	Ensures applications run consistently across environments (local, cloud, etc.).	Extends portability to orchestrated clusters across hybrid and multi-cloud environments.
Ease of Use	Simple and developer-friendly.	Complex, requires expertise to configure and manage effectively.
Use Case	Suitable for small-scale containerized applications and local development.	Ideal for large-scale deployments, distributed systems, and production environments.
Relationship	Docker containers are used to package applications.	Kubernetes uses Docker (or other runtimes) to orchestrate and manage containers.
Can Be Used Independently	Yes, Docker can run containers without Kubernetes.	Yes, Kubernetes can work with other container runtimes (e.g., containerd, CRI-O).

# What is Container Orchestration?

**Container orchestration refers to the automated management of containerized workloads and services.**



**It addresses the challenges that arise when organizations deploy and manage hundreds or even thousands of containers.**



**It includes tasks like:**

Scheduling container deployment.

Managing networking between containers.

Ensuring scalability, fault tolerance, and high availability.

Monitoring and maintaining the health of containerized applications.

Kubernetes has emerged as the industry standard for container orchestration.



# History and Development of Kubernetes

---



**Origins:**  
Kubernetes is based on Borg, Google's internal container orchestration platform, which was developed to manage the company's massive infrastructure.



**Open Source Launch (2014):** Google released Kubernetes as an open-source project, recognizing the need for an external, community-driven solution.



**CNCF Partnership (2015):** Google donated Kubernetes to the Cloud Native Computing Foundation (CNCF), a neutral body fostering cloud-native technologies.



**Growth and Popularity:**

Became the CNCF's **first hosted project** in March 2016.

Graduated as CNCF's **first graduate project** in 2018, a testament to its maturity and widespread adoption.

Currently, Kubernetes is the **primary container orchestration tool** for **71% of Fortune 100 companies**.



**Community Contributions:**

The Kubernetes repository on GitHub has over **8,000 contributors** and **123,000 commits**.

It is the **second-largest open-source project globally**, after Linux.

# Why Kubernetes Became Dominant

---

- **Vendor-Neutral and Open Source:**
  - Supported by major tech companies like Microsoft, IBM, Red Hat, and Google.
  - Not tied to any single vendor, fostering broad adoption.
- **Scalability and Performance:**
  - Handles large-scale deployments with thousands of containers.
- **Ecosystem Support:**
  - Seamlessly integrates with other tools and platforms (e.g., Helm, Prometheus).
  - Backed by cloud providers like AWS, Azure, and Google Cloud.
- **Flexibility:**
  - Supports hybrid and multi-cloud environments.
  - Can orchestrate various container runtimes beyond Docker, such as **containerd** and **CRI-O**.
- **Active Community:**
  - Fast-paced development with frequent updates and robust support.
- **Feature-Rich:**
  - Automates rolling updates, rollbacks, self-healing, and scaling.

## Kubernetes vs. Other Orchestration Tools

Feature	Kubernetes	Docker Swarm	Apache Mesos
Scalability	Highly scalable for large clusters.	Suitable for smaller clusters.	Designed for massive scale but complex to manage.
Community Support	Strong global community and ecosystem.	Smaller community.	Limited adoption in modern container ecosystems.
Ease of Use	Complex setup but feature-rich.	Simpler to set up and use.	High learning curve.
Adoption	Widely adopted across industries.	Niche use cases.	Primarily used for legacy systems.

# Kubernetes architecture and components

## KUBERNETES ARCHITECTURE

User Interface



kubectl

Kubernetes Master

API Server

Scheduler

Controller-Manager

etcd

Worker Node 1

Pod 1

Container 1  
Container 2  
Container 3

Pod 2

Container 1

Pod 3

Container 1  
Container 2

DOCKER

kubelet

Kube-proxy

Worker Node 2

Pod 1

Container 1  
Container 2

Pod 2

Container 1  
Container 2  
Container 3

Pod 3

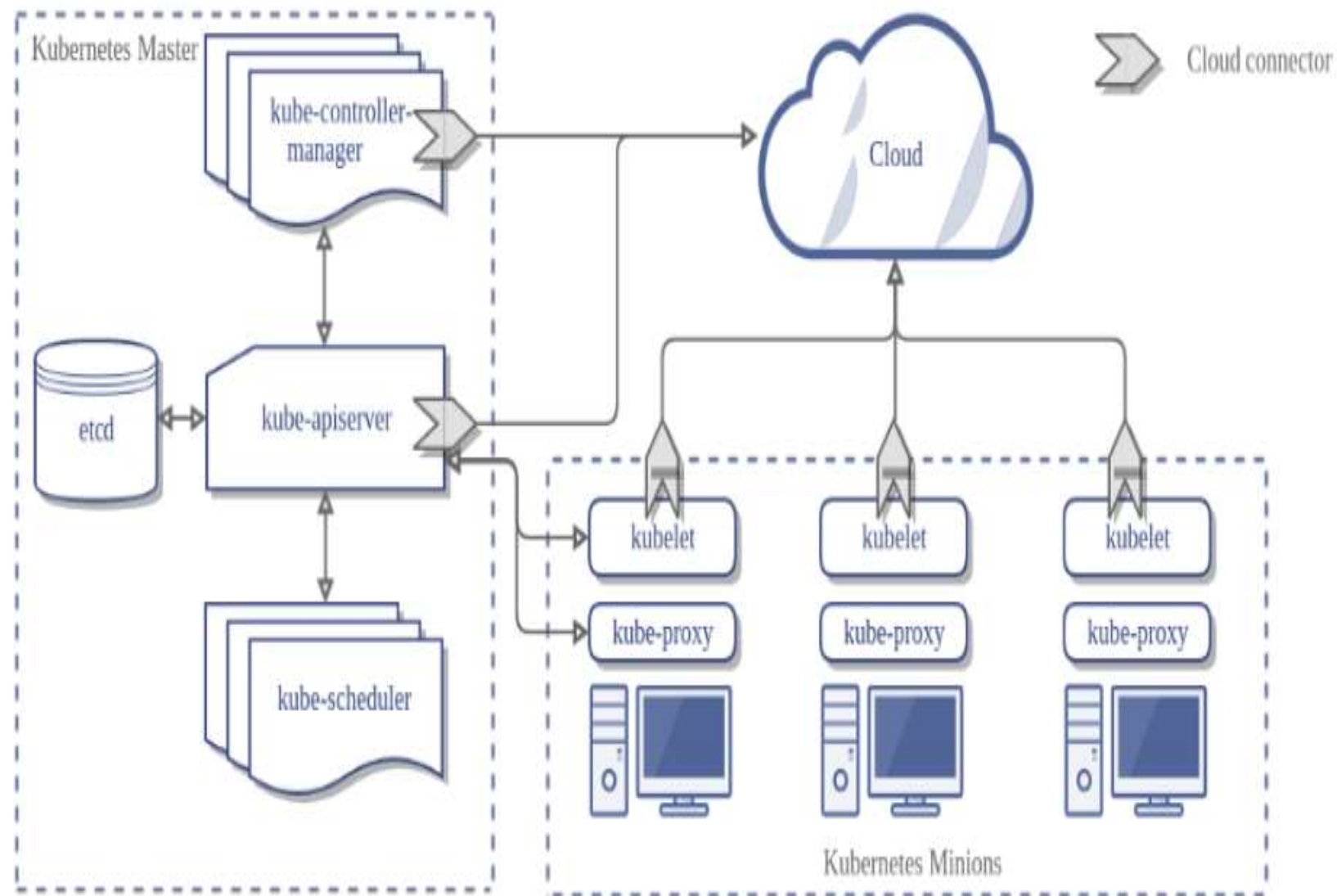
Container 1

DOCKER

kubelet

Kube-proxy

# Kubernetes components



# Cluster Overview

**Cluster:** The fundamental Kubernetes architecture unit, consisting of:

**Master Node:** Manages the cluster's control plane.

**Worker Nodes:** Execute the workloads by running containerized applications.



**Pods:** The smallest deployable unit, consisting of one or more containers that share resources like networking and storage.

Pods are the **unit of scalability**, replicated as needed across nodes.

# Control Plane Components (Master Node)

*The control plane is the "brain" of Kubernetes, responsible for orchestrating the cluster.*

Component	Function
API Server (kube-apiserver)	Exposes the Kubernetes API as an entry point for commands and queries, enabling communication between users, control plane components, and the cluster.
etcd	A distributed key-value store that manages cluster state, configuration data, and metadata, ensuring consistency across the cluster.
Scheduler	Assigns newly created pods to nodes by evaluating resource availability, hardware/software requirements, and constraints.
Controller Manager	Manages control loops to maintain desired states for resources like pods, service endpoints, and replica sets by interacting with the API server.
Cloud Controller Manager	Interfaces with the cloud provider to manage resources like load balancers, nodes, and storage in cloud environments.

# Node Components (Worker Nodes)

*Worker nodes are responsible for **running workloads** and managing pods.*

Component	Function
Kubelet	A node-level agent that ensures containers in pods are running as specified by the control plane.
Kube-proxy	Manages networking by maintaining network rules and load balancing between services and pods.



# Other Kubernetes Concepts

---

Concept	Description
ReplicaSet	Ensures a specified number of pod replicas are running at any given time.
Deployment	Manages the lifecycle of applications, ensuring desired states and handling scaling and updates.
Kubectl	Command-line tool for managing clusters via the Kubernetes API.
DaemonSets	Ensures a specific pod is present on every node in the cluster.
Add-ons	Extend Kubernetes functionality, such as DNS for service discovery, web UI dashboards, and monitoring.
Service	Provides a stable network endpoint for accessing pods, enabling load balancing and service discovery.

# Kubernetes Deployment Process

---

- **Developer** creates a **deployment** using **kubectl** or another tool.
- Deployment specifies the desired number of pods and configurations.
- **API Server** processes the deployment and schedules pods via the **Scheduler**.
- **etcd** stores the cluster's state.
- **Controller Manager** ensures pods meet the desired state.
- **Kubelet** runs the pods on the nodes, monitored by **Kube-proxy** for networking.

# Kubernetes use cases

## Microservices architecture or cloud-native development

- Cloud native is a software development approach for building, deploying and managing cloud-based applications.
- The major benefit of cloud-native is that it allows DevOps and other teams to code once and deploy on any cloud infrastructure from any cloud service provider.
- This modern development process relies on microservices, an approach where a single application is composed of many loosely coupled and independently deployable smaller components or services, which are deployed in containers managed by Kubernetes.
- Kubernetes helps ensure that each microservice has the resources it needs to run effectively while also minimizing the operational overhead associated with manually managing multiple containers.

## Hybrid multicloud environments

- Hybrid cloud combines and unifies public cloud, private cloud and on-premises data center infrastructure to create a single, flexible, cost-optimal IT infrastructure.
- Today, hybrid cloud has merged with multicloud, public cloud services from more than one cloud vendor, to create a hybrid multicloud environment.
- A hybrid multicloud approach creates greater flexibility and reduces an organization's dependency on one vendor, preventing vendor lock-in. Since Kubernetes creates the foundation for cloud-native development, it's key to hybrid multicloud adoption.

# Kubernetes use cases

---

## Applications at scale

- Kubernetes supports large-scale cloud app deployment with autoscaling. This process allows applications to scale up or down, adjusting to demand changes automatically, with speed, efficiency and minimal downtime.
- The elastic scalability of Kubernetes deployment means that resources can be added or removed based on changes in user traffic like flash sales on retail websites.

## Application modernization

- Kubernetes provides the modern cloud platform needed to support application modernization, migrating and transforming monolithic legacy applications into cloud applications built on microservices architecture.

## DevOps practices

- Automation is at the core of DevOps, which speeds the delivery of higher-quality software by combining and automating the work of software development and IT operations teams.
- Kubernetes helps DevOps teams build and update apps rapidly by automating the configuration and deployment of applications.

## Artificial intelligence (AI) and machine learning (ML)

- The ML models and large language models (LLM) that support AI include components that would be difficult and time-consuming to manage separately.
- By automating configuration, deployment and scalability across cloud environments, Kubernetes helps provide the agility and flexibility needed to train, test and deploy these complex models.
- Kubernetes tutorials

