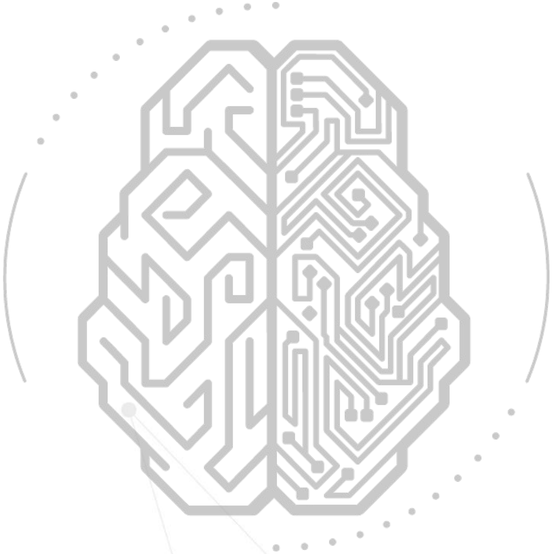
Era of Containerisation

अमृतं तु विद्या



Speaker

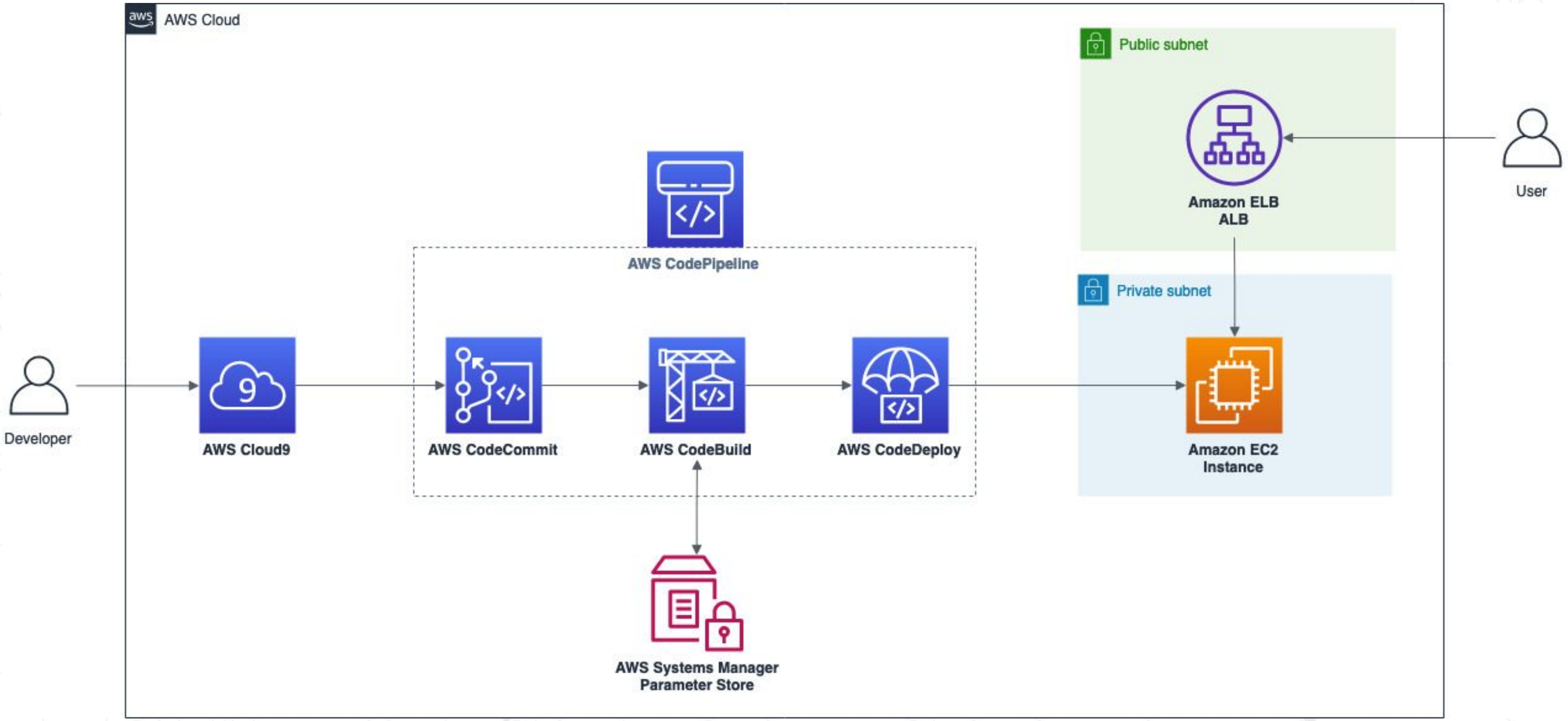
Sanchit Jain

Lead Architect - AWS at Quantiphi
AWS APN Ambassador

FOLLOW ME

Demo - CloudNative CI/CD Pipeline

AWS CI/CD simple Node.js app



Key Pairs

- Open the EC2 console at https://console.aws.amazon.com/ec2
- In the navigation pane choose KeyPair, and create or import a Key Pair named devops-demo

Cloudformation

- Open the CloudFormation console at https://console.aws.amazon.com/cloudformation
- On the Welcome page, click on Create stack button
- On the Step 1 - Specify template: Choose Upload a template file, click on Choose file button and select the template.yaml located inside deploy directory
- On the Step 2 - Specify stack details: Enter the Stack name as 'cicd-techtalk'
- On the Step 3 - Configure stack options: Just click on Next button
- On the Step 4 - Review: Enable the checkbox I acknowledge that AWS CloudFormation might create IAM resources with custom names., and click on Create Stack button
- Wait for the stack get into status CREATE_COMPLETE
- Under the Outputs tab, take a note of ELB value

CodeCommit

- Open the CodeCommit console at https://console.aws.amazon.com/codecommit
- Click on Create repository button, enter the Repostiory name as 'cicd-demo and then click on Create button

Grant access for CodeCommit on IAM

- Open the IAM console at https://console.aws.amazon.com/iam
- In the navigation pane select Users, and then click on demo username
- Under Security Credentials tab, click on Generate button of section HTTPS Git credentials for AWS CodeCommit
- Click on Download credentials button and save the generated csv file

Push code into code-commit

- git init
- git add .
- git commit -m "Repo Init"
- git remote add origin https://git-codecommit.ap-south-1.amazonaws.com/v1/repos/cicd-demo
- git push -u origin master

CodeBuild

- Open the CodeBuild console at https://console.aws.amazon.com/codebuild
- Click on Create build project button
- Enter the Project name as 'cicd-demo
- On Source define AWS CodeCommit as the source provider and select cicd-techtalk for repository
- On Environment choose Ubuntu for Operational System, Standard for Runtime and aws/codebuild/standard:4.0 as the Image version. After, select Existing service role and search for CodeBuild-cicd-demo
- Uncheck the checkbox Allow AWS CodeBuild to modify this service role so it can be used with this build project
- Click on Create build project button

CodeDeploy

- Open the CodeDeploy console at https://console.aws.amazon.com/codedeploy
- Click on Create application button
- Enter the Application name as 'cicd-demo, select EC2/On-premises for Compute platform and then click on Create button
- Once your application was created, under Deployment groups tab click on Create deployment group button
- Enter the deployment group name as 'cicd-demo
- Select the Service Role as CodeDeploy-cicd-demo
- On Environment configuration select Amazon EC2 instances, enter for Key 'Name' and for Value 'cicd-demo'
- On Load Balancer configuration select cicd-techtalk-tg
- Click on Create deployment group button

CodePipeline

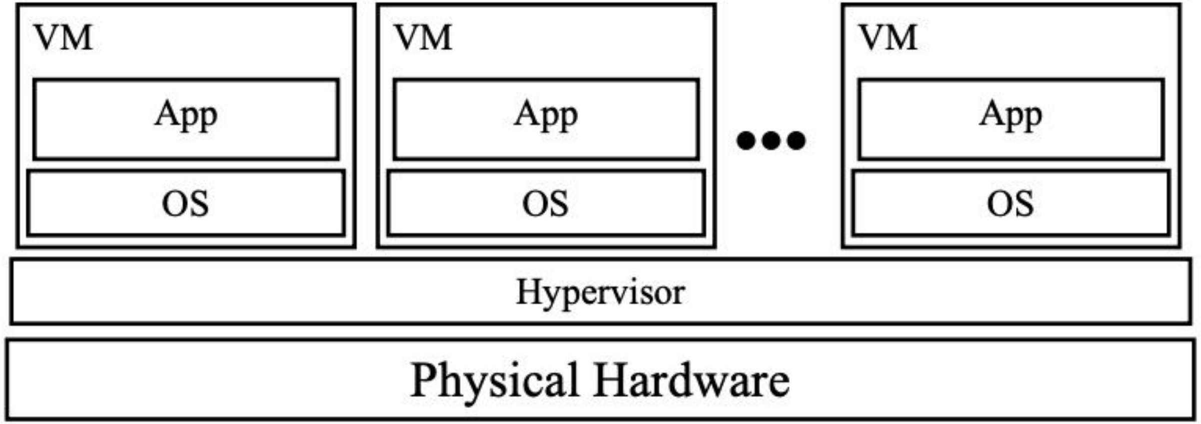
- Open the CodePipeline console at https://console.aws.amazon.com/codepipeline
- Click on Create pipeline button
- On the Step 1 - Choose pipeline setting: Enter the Pipeline name as 'cicd-techtalk'
- On the Step 2 - Add source stage: 3.1 Select AWS CodeCommit for Source provider 3.2 Select cicd-demo for Repository name 3.3 Select master for Branch name
- On the Step 3 - Add build stage: Select AWS Codebuild for Build provider, and 'cicd-demo' for Project name
- On the Step 4 - Add deploy stage: Select AWS CodeDeploy for Deploy provider, cicd-demo for Application and Deployment group
- On the Step 5 - Review: Click on Create pipeline button
- Note: The first execution will fail during the build phase, because the project code is not committed yet.

Virtualization & Docker

Advantages of Virtualization

- Minimize hardware costs (CapEx)
- Multiple virtual servers on one physical hardware
- Easily move VMs to other data centers
- Provide disaster recovery. Hardware maintenance.
- Follow the sun (active users) or follow the moon (cheap power)
- Consolidate idle workloads. Usage is bursty and asynchronous.
- Increase device utilization
- Conserve power
- Free up unused physical resources
- Easier automation (Lower OpEx)
- Simplified provisioning/administration of hardware and software
- Scalability and Flexibility: Multiple operating systems

Problems of Virtualization



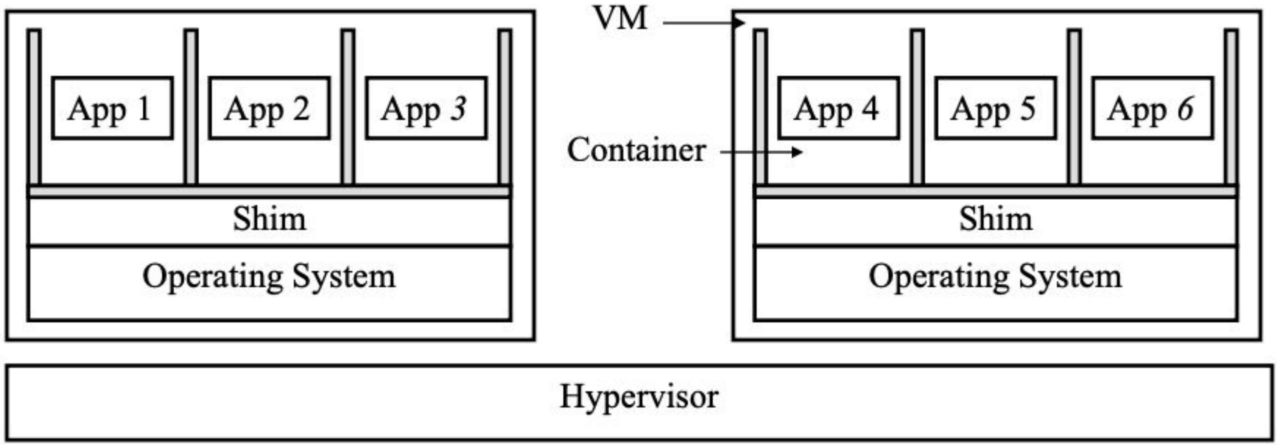
Each VM requires an operating system (OS)

- Each OS requires a license ⇒ CapEx
- Each OS has its own compute and storage overhead
- Needs maintenance, updates ⇒ OpEx
- VM Tax = added CapEx + OpEx

Solution: Containers

- Run many apps in the same virtual machine
- These apps share the OS and its overhead
- But these apps can’t interfere with each other
- Can’t access each other’s resources
- without explicit permission
- Like apartments in a complex ⇒ Containers

Containers

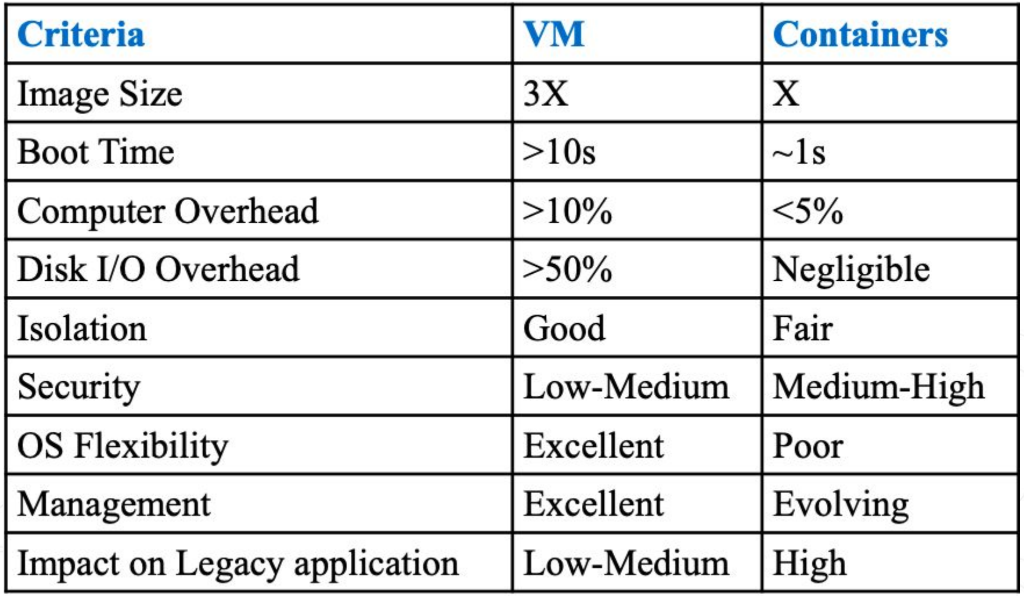


- Multiple containers run on one operating system on a virtual/physical machine
- All containers share the operating system ⇒ CapEx and OpEx
- Containers are isolated ⇒ cannot interfere with each other
  - Own file system/data, own networking ⇒ Portable

Containers

- Containers have all the good properties of VMs
  - Come complete with all files and data that you need to run
  - Multiple copies can be run on the same machine or different machine ⇒ Scalable
  - Same image can run on a personal machine, in a data center or in a cloud
  - Operating system resources can be restricted or unrestricted as designed at container build time
  - Isolation: For example, “Show Process” (ps on Linux) command in a container will show only the processes in the container
  - Can be stopped. Saved and moved to another machine or for later run

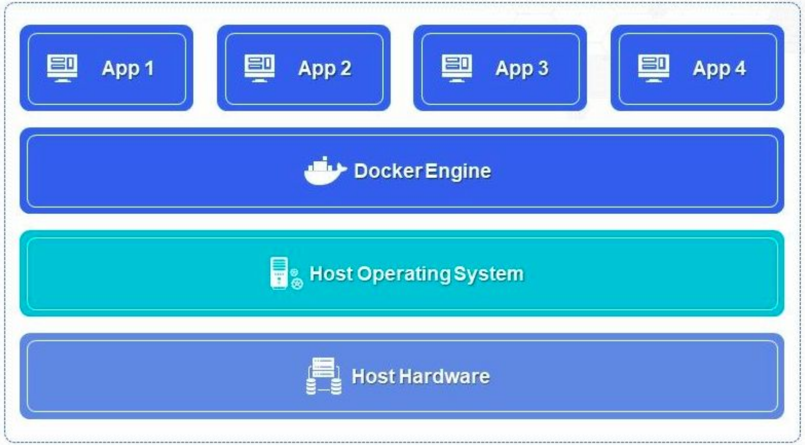
VM vs. Containers



Criteria	VM	Containers
Image Size	3X	X
Boot Time	>10s	~1s
Computer Overhead	>10%	<5%
Disk I/O Overhead	>50%	Negligible
Isolation	Good	Fair
Security	Low-Medium	Medium-High
OS Flexibility	Excellent	Poor
Management	Excellent	Evolving
Impact on Legacy application	Low-Medium	High

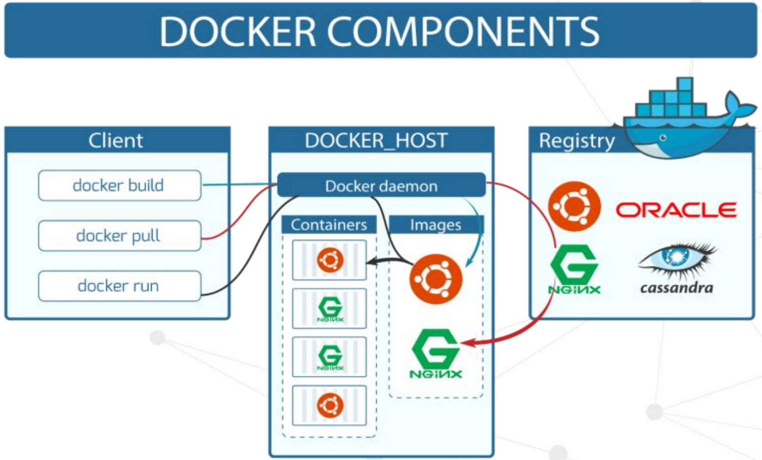
Docker

- Provides the isolation among containers
- Helps them share the OS
- Docker = Dock worker ⇒ Manage containers
- Developed initially by Docker.com
- Downloadable for Linux, Windows, and Mac from Docker.com
- Customizable with replacement modules from others



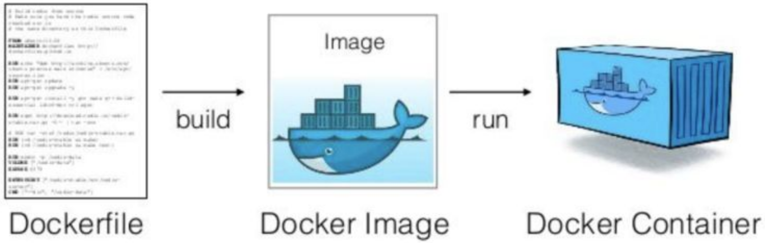
Docker Engine Components

- The Docker client enables users to interact with Docker
- The Docker host provides a complete environment to execute and run applications. It includes Docker daemon, Images, Containers, Networks, and Storage.
- Docker Daemon is a persistent background process that manages Docker images, containers, networks, and storage volumes. The Docker daemon constantly listens for Docker API requests and processes them.
- Docker-images are a read-only binary template used to build containers. Images also contain metadata that describe the container's capabilities and needs.

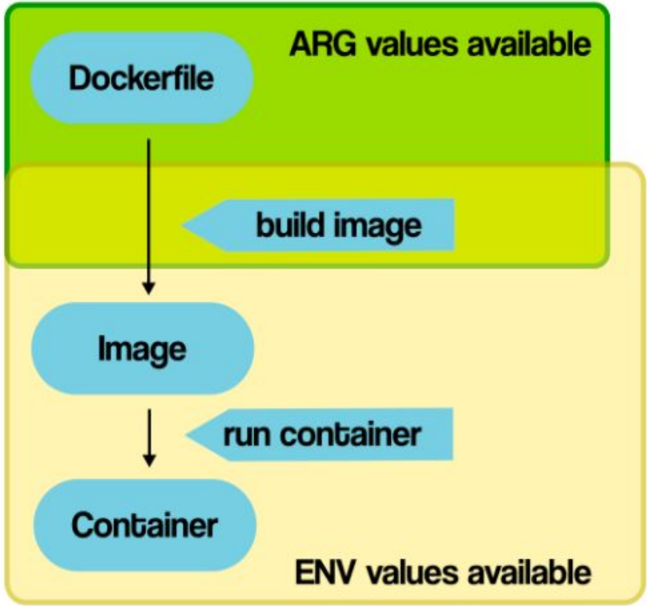


Docker Engine Components

- A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI
- Through the docker networking, we can communicate one container to other containers.
- A container is volatile it means whenever you remove or kill the container then all of its data will be lost from it. If you want to persist the container data use the docker storage concept.
- Docker-registries are services that provide locations from where you can store and download images.



Building Container Images



Dockerfile content:

```
ARG required_var          # expects a value
ARG var_name=def_value # set default ARG value
ENV foo=other_value     # set default ENV value
ENV bar=${var_name}     # set default ENV value from ARG
```

Override Dockerfile ARG values on build:

```
$ docker build --build-arg var_name=value
```

Override Docker image ENV values on run:

```
$ docker run -e "foo=other_value" [...]
$ docker run -e foo [...] # pass from host
$ docker run --env-file=env_file_name [...] # pass from file
```

Docker Commands

- **docker container run**: Run the specified image
- **docker container ls**: list running containers
- **docker container exec**: run a new process inside a container
- **docker container stop**: Stop a container
- **docker container start**: Start a stopped container
- **docker container rm**: Delete a container
- **docker container inspect**: Show information about a container

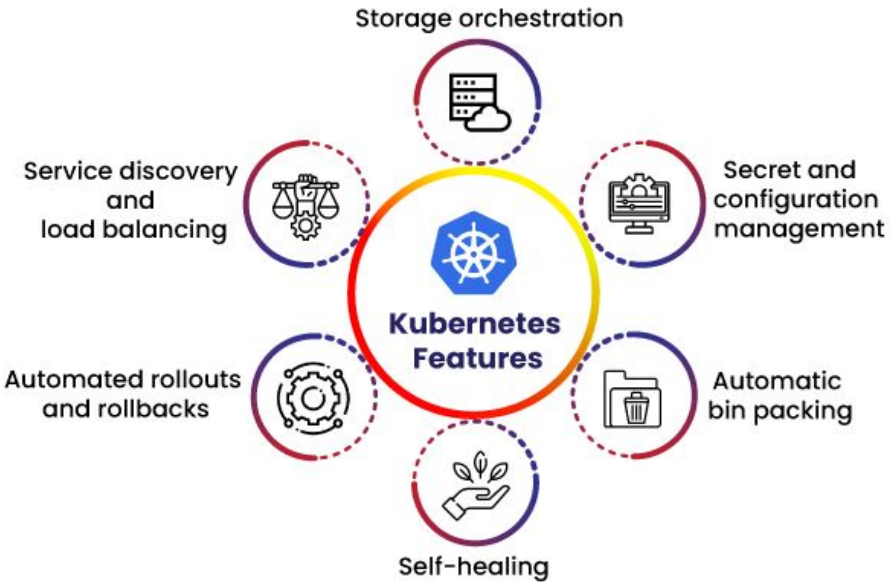
Demo - Docker

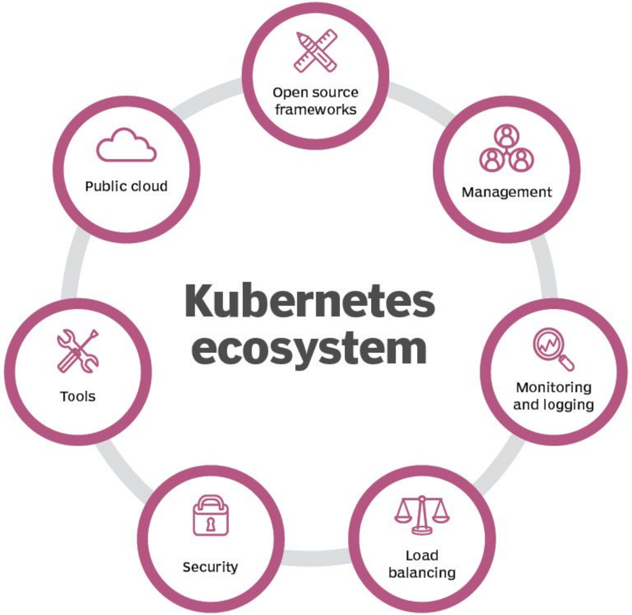
- Setup Docker - link
- Download Github repo - link
- Point to the downloaded folder, and run below commands
  - Build it: docker build -t linux_tweet .
  - Run it: docker container run --detach -p 80:80 linux_tweet

Kubernetes

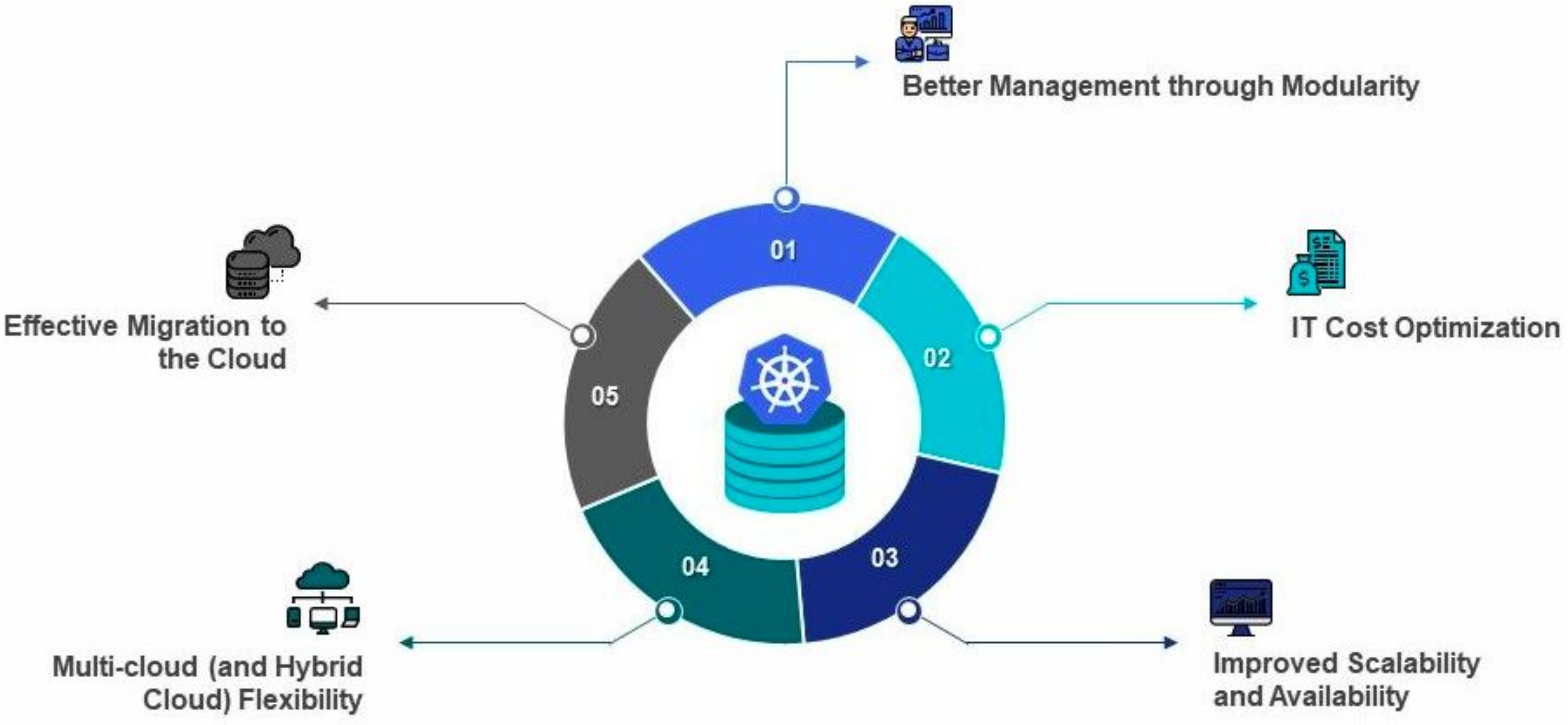
What is Kubernetes

- Kubernetes, also referred to as K8s, is an open source platform used to manage Linux Containers across private, public and hybrid cloud environments.
- Businesses also can use Kubernetes to manage microservice architectures. Containers and Kubernetes are deployable on most cloud providers.


Storage orchestration
Service discovery and load balancing
Secret and configuration management
Kubernetes Features
Automated rollouts and rollbacks
Automatic bin packing
Self-healing



Why Kubernetes?



Better Management through Modularity

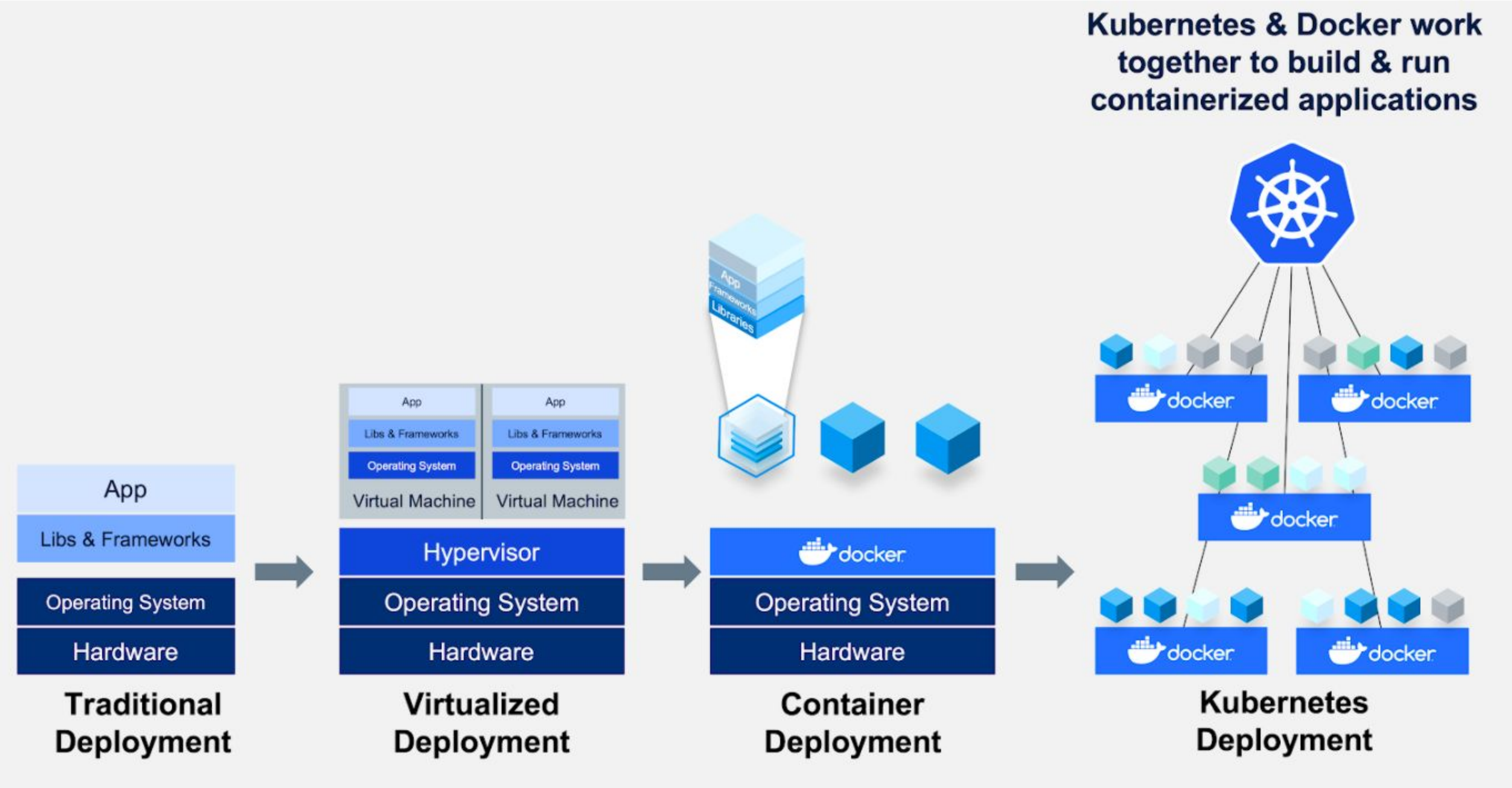
IT Cost Optimization

Improved Scalability and Availability

Multi-cloud (and Hybrid Cloud) Flexibility

Effective Migration to the Cloud

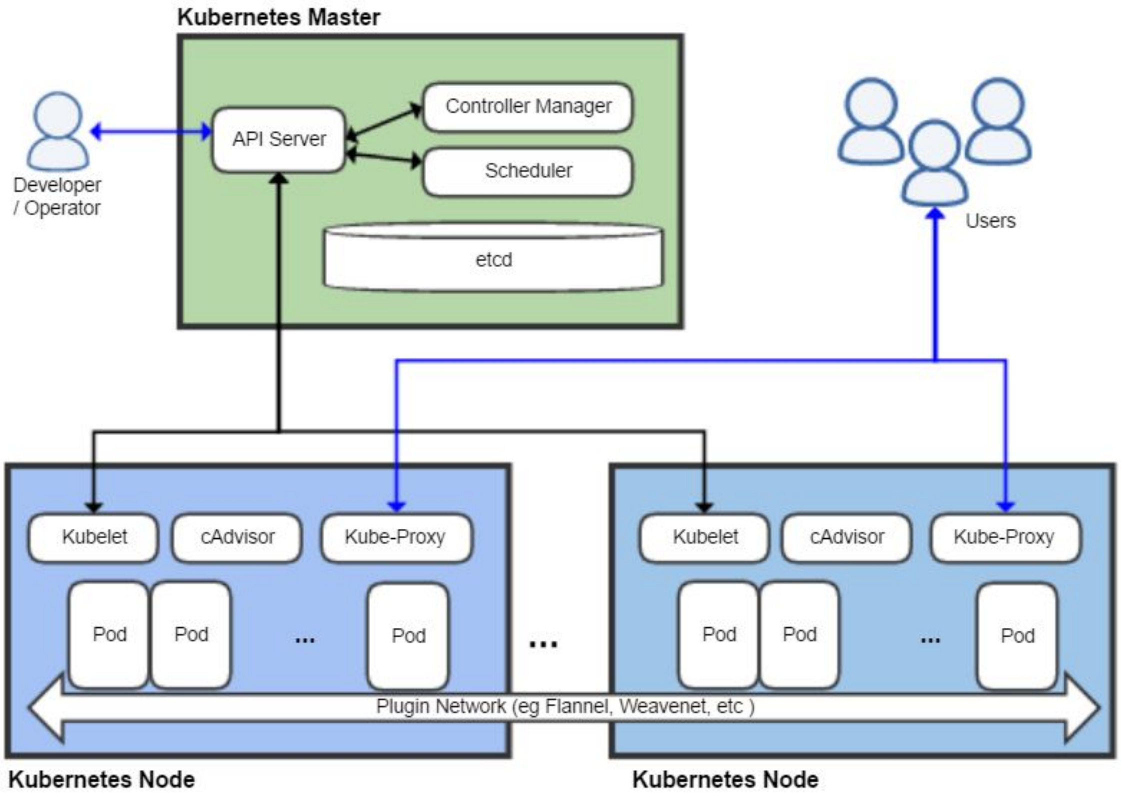
Docker vs Kubernetes



Docker Swarm vs Kubernetes

Docker Swarm	Vs	Kubernetes
No Auto Scaling	01	Auto Scaling
Good Community	02	Great Active Community
Easy to Start a Cluster	03	Difficult to Start a Cluster
Limited to the Docker Api's Capabilities	04	Can Overcome Constraints of Docker and Docker API
Does not have as Much Experience with Production Deployments at Scale	05	Deployed at Scale more often among Organisations

Kubernetes Architecture



Kubernetes Architecture

- A Kubernetes instance is made up of a cluster of hosts, it can be a small as just one host, minikube is an example of a minimal install. At the other extreme, 1000 node clusters are possible.
- Kubernetes consists of 2 main components:
  - Master(s) – At least one is required, use multiple for resilience
  - Node(s) – Can be run on the Master host, typically these are separate hosts
- Master: The Kubernetes Master is the controller of the cluster and it consists of multiple services:
  - API server: The API server is the service through which Kubernetes commands are issued and Kubernetes specific information is queried.
  - Controller: The controller watches the state of the cluster and initiates changes in an attempt to comply with the desired state (desired state is defined to K8s by the K8s administrator and is typically stored in YAML files).
  - Scheduler: The scheduler is responsible for deciding where to run the Pods (containers) based on the resources available on the Nodes.
  - etcd: Configuration data and container state information is held in etcd.

Kubernetes Architecture

- Nodes: also known as Workers or Minions, are responsible for running multiple Kubernetes services. These services consist of:
  - Kubelet: The Kubelet service is responsible for starting and stopping the containers and provides a heartbeat back to the Master.
  - Kube-proxy: The Kube Proxy service is a network proxy and load balancer that is responsible for routing traffic to the appropriate container based on the incoming request IP and port number.
  - Pod: The Pod is the basic unit of work, it runs the container image; think of it as a wrapper around the container. Pods can also run multiple containers and it wraps these together along with storage into one manageable unit.
  - cAdvisor: cAdvisor is an agent (built into the Kubelet) that collects resource utilisation information which is used by the Kubernetes Master.

Kubernetes terms

- Pods: Kubernetes deploys and schedules containers in groups called pods. Containers in a pod run on the same node and share resources such as filesystems, kernel namespaces, and an IP address.
- Deployments: These building blocks can be used to create and manage a group of pods. Deployments can be used with a service tier for scaling horizontally or ensuring availability.
- Services: Services are endpoints that can be addressed by name and can be connected to pods using label selectors. The service will automatically round-robin requests between pods.
- Labels: Labels are key-value pairs attached to objects and can be used to search and update multiple objects as a single set.
- Selector. A matching system used for finding and classifying specific resources.
- Ingress. An application program interface (API) object that controls external access to services in a cluster usually HTTP.

THANK YOU

aws