What are Virtual Machines

25 May 2024 13:16

devlopes

How to run a web application on a physical server?

→ highly available →24 shank

→ scale

Constraint! - We can run only one web application on a single physical server. But Why?

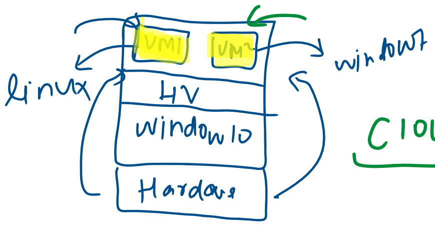
1. If we host multiple web applications on a single server, they will use the same OS and hardware. If one application consumes too much CPU, RAM etc. It will degrade the performance of other application.
2. Compatibility issues - Different web application might require different versions of libraries, runtimes or even OS. This can create conflict.
3. Security issues - If one application is compromised other will be at risk too.

Problem

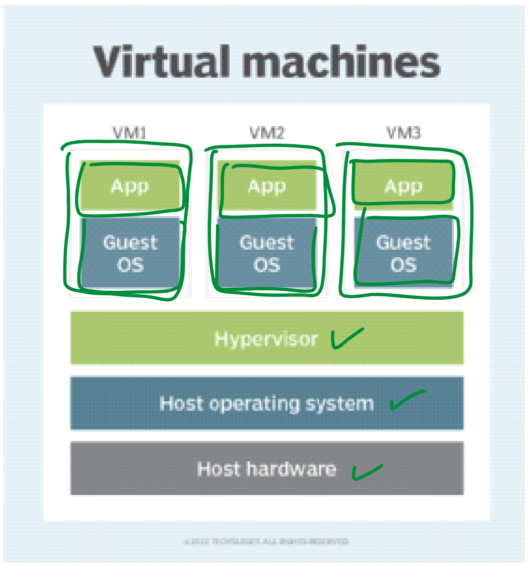
1. Resource wastage
2. High cost

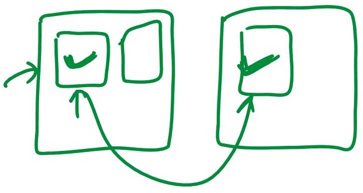
Solution - Virtual Machines

A virtual machine (VM) is a software-based emulation of a physical computer. VMs are created using virtualization software, known as a hypervisor, which allows multiple VMs to run on a single physical hardware system. Each VM operates as an independent and isolated environment with its own operating system (OS), applications, and resources.



cloud computing

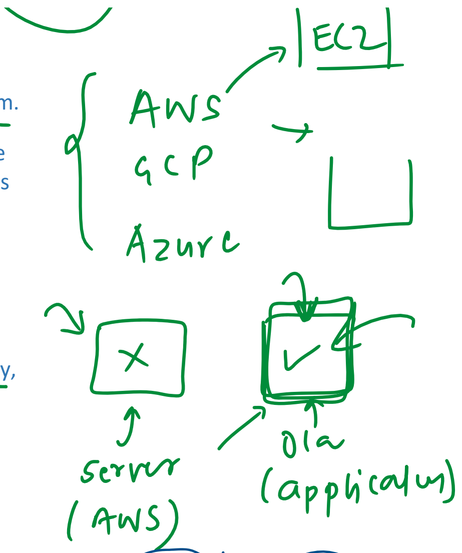




EC2

1. **Isolation**: Each VM is isolated from others, meaning that problems in one VM (such as crashes or security breaches) do not affect other VMs or the host system

1. **Isolation**: Each VM is isolated from others, meaning that problems in one VM (such as crashes or security breaches) do not affect other VMs or the host system.
2. **Independence**: VMs can run different operating systems and applications on the same physical hardware. For example, you can run both Windows and Linux VMs on a single physical server.
3. **Resource Allocation**: VMs share the physical resources (CPU, memory, storage, etc.) of the host system. The hypervisor manages resource allocation, ensuring that each VM gets the necessary resources.
4. **Encapsulation**: VMs are encapsulated into files, making them easy to move, copy, and back up. This encapsulation simplifies VM management and deployment.



Disadvantages of using VM OS

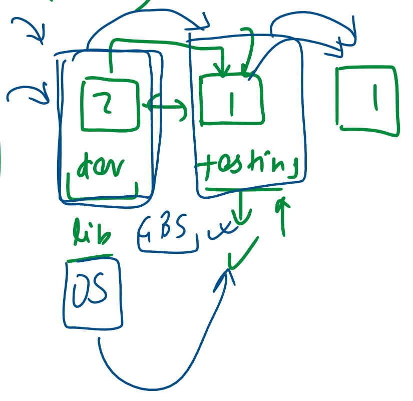
- **Limited ~~Portability~~**: VMs are less portable compared to containers. Migrating VMs between different environments or cloud providers can be complex due to differences in hypervisors and configurations. Size is also a consideration. It makes VM difficult to move between different teams like development, testing and production.

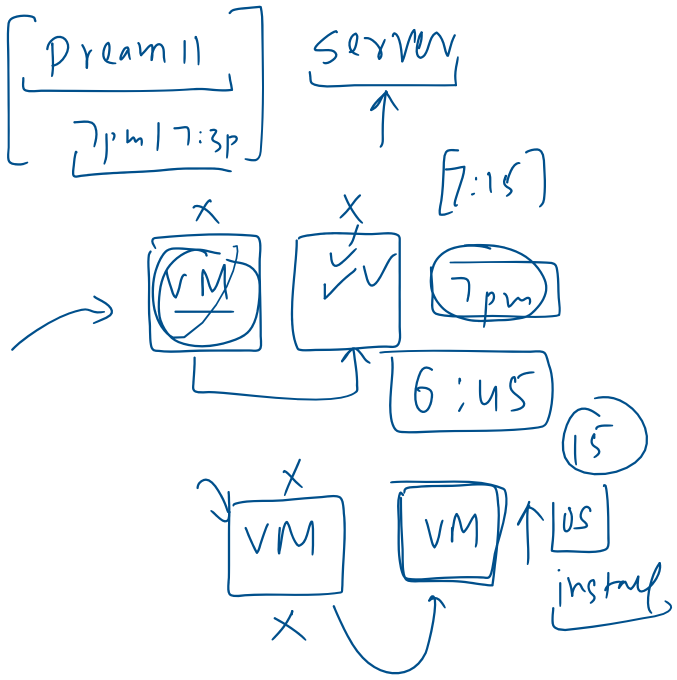
OS

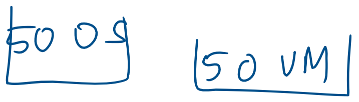
- **Inefficient Scaling**: VMs take longer to boot compared to containers because they need to initialize an entire OS. This can delay the deployment and scaling of applications. In the event of a failure, the time taken to restart VMs can be longer, impacting availability and responsiveness.

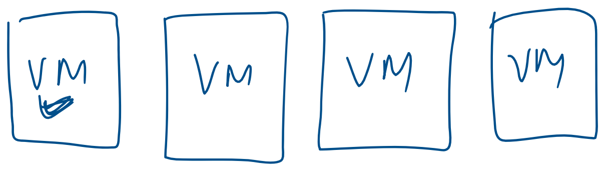
OS

- **OS Licensing**: Each VM typically requires a separate operating system license, which can increase costs, especially in environments with a large number of VMs.









Problem → parallel exc of applic'n

Solu'n – OS

Containerization

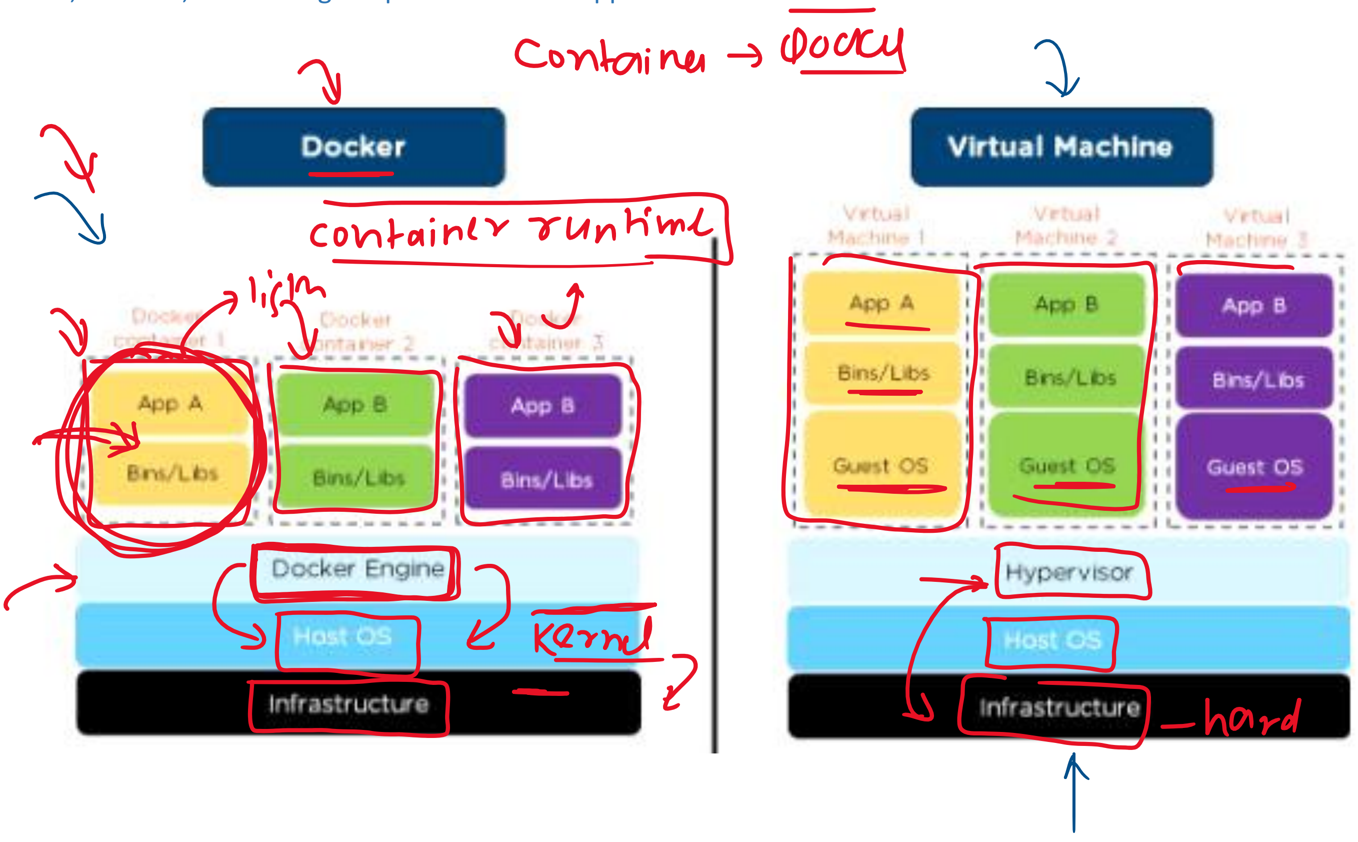
26 May 2024 23:46

dream → python
US

Containerization is a lightweight form of virtualization that allows you to run multiple isolated applications on a single host operating system. This involves packaging an application and it's dependencies into a container.

Containers are standardized units of software that encapsulate the code, runtime, system tools, libraries, and settings required to run an application.

Container → Docker



The **Docker Engine** is the core component of the Docker platform, responsible for creating and running Docker containers.

Namespaces helps separate and isolate different parts of a computer system for different groups of processes. Think of namespaces as creating separate compartments within the same computer. Each compartment can have its own set of resources like files, network connections, and process IDs, making it seem like each compartment is a completely independent mini-computer.

cgroups (control groups) is a feature that helps manage how much of the system's resources (like CPU power, memory, disk usage, and network bandwidth) different groups of processes can use. It keeps track of how much each group is using and can limit their usage to make sure no single group uses too much, ensuring fair resource distribution and system stability.

Deve

How to create docker containers?

docker

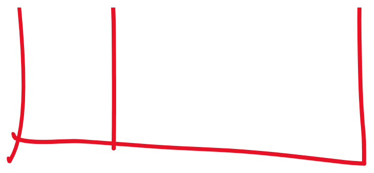
Dockerfile is a text file with instructions on how to build a Docker image

Docker Images: Read-only templates used to create containers

Dockerfile is a text file with instructions on how to build a Docker image

Docker Images: Read-only templates used to create containers

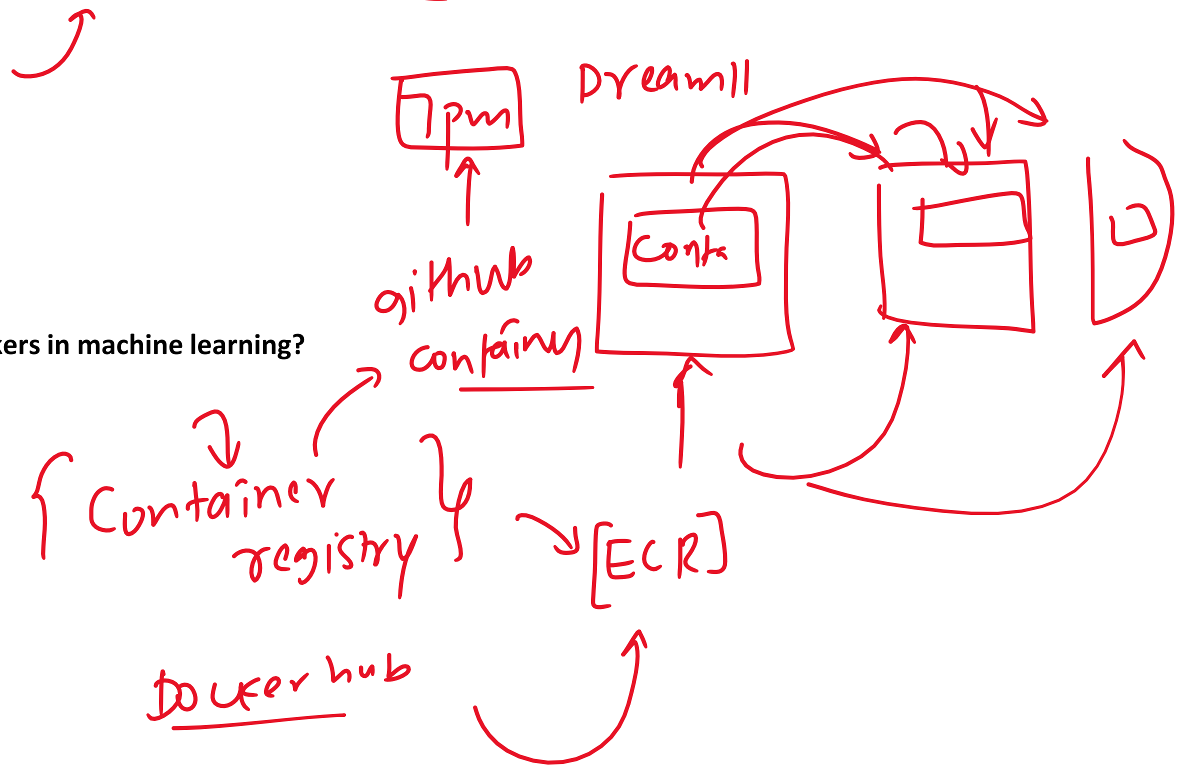
Docker Containers: Runnable instances of Docker images.



Benefits

1. Portability
2. Scalability

How do we use dockers in machine learning?



What is Container Registry

A container registry is a centralized repository or service that stores, manages, and distributes container images. It functions similarly to a source code repository but is specifically designed for container images. Container registries play a crucial role in the container ecosystem, enabling developers to store and share container images easily.

Key Features of a Container Registry:

1. Image Storage: Container registries provide a place to store container images. These images can be public or private, depending on the registry's configuration and permissions.

2. Versioning: Registries support versioning of images, allowing users to tag images with specific versions (e.g., v1.0, latest). This makes it easy to manage and track different versions of an application.

3. Distribution: Registries facilitate the distribution of container images. Users can push images to a registry and pull them from anywhere, making it easy to deploy applications across different environments.

4. Access Control: Registries offer access control mechanisms to restrict who can push and pull images. This is important for security, especially in enterprise environments.

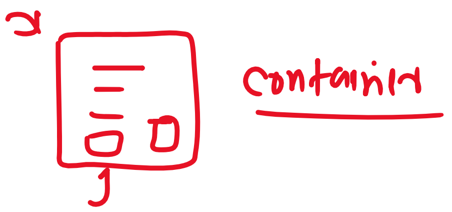
5. Integration with CI/CD: Container registries integrate with continuous integration and continuous deployment (CI/CD) pipelines, automating the process of building, storing, and deploying container images.

Examples -> Docker Hub, ECR and GCR

Examples -> Docker Hub, ECR and GCR

Deploying a container

27 May 2024 11:18



✓ Step 1: Prepare Your Application

Ensure your application code and dependencies are ready. Typically, this involves setting up a project structure and creating a `Dockerfile` to define how the Docker image should be built.

Step 2: Write a Dockerfile

Create a `Dockerfile` in the root of your project directory. Here's an example `Dockerfile` for a simple Node.js application:

2

```
# Use an official Node.js runtime as a parent image
FROM node:14

# Set the working directory
WORKDIR /usr/src/app

# Copy the package.json and package-lock.json files
COPY package*.json ./

# Install the dependencies
RUN npm install

# Copy the rest of the application code
COPY . .

# Expose the port the app runs on
EXPOSE 8080

# Define the command to run the application
CMD ["node", "app.js"]
```

Step 3: Build the Docker Image

Use the Docker CLI to build an image from the Dockerfile. Run this command in the directory containing your `Dockerfile`:

```
docker build -t myapp:latest .
```

Step 4: Test the Docker Image Locally

Before deploying, test the image locally to ensure it works as expected:

```
docker run -d -p 8080:8080 myapp:latest
```

Step 5: Push the Docker Image to a Container Registry

If you're deploying to a remote server or cloud provider, push your Docker image to a container registry. You can use Docker Hub, AWS ECR, Google Container Registry, or another registry.

First, log in to your registry:

```
docker login
```

✓

Then, tag your image with the registry URL and push it:

```
docker tag myapp:latest myregistry.com/myapp:latest
docker push myregistry.com/myapp:latest
```

✓

1. **Launch an $\boxed{\text{EC2 Instance}}$**

   1. **Log in to AWS Management Console**:
      - Go to the AWS Management Console.
   2. **Navigate to EC2 Dashboard**:
      - Select "EC2" under the "Compute" section.

3. **Launch Instance**:
   - Click the "Launch Instance" button.
   - Choose an Amazon Machine Image (AMI), such as "Amazon Linux 2 AMI".
   - Select an instance type, like `t2.micro` (eligible for free tier).
   - Configure instance details as needed.
   - Add storage (8 GB is usually sufficient).
   - Add tags if necessary.
   - Configure security group to allow SSH (port 22) and the port your application will use (e.g., HTTP port 80 or 8080).

- Configure security group to allow SSH (port 22) and the port your application will use (e.g., HTTP port 80 or 8080).
- Review and launch the instance.

4. Download Key Pair:
   - When prompted, create a new key pair or use an existing one.
   - Download the key pair (`.pem` file) and save it securely.

2. Connect to Your EC2 Instance

3. Install Docker on the EC2 Instance ✓

4. Pull and Run Docker Image from Docker Registry

```
docker login
```

```
docker pull <your-dockerhub-username>/myapp:latest
```

```
docker run -d -p 80:8080 <your-dockerhub-username>/myapp:latest
```

Make it available for the internet

Container Orchestration

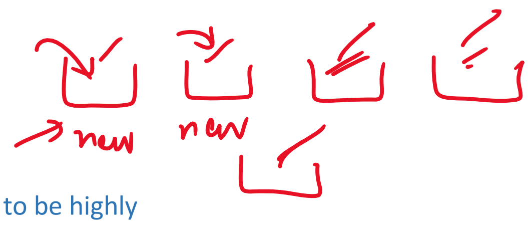
27 May 2024 09:26

Container orchestration is the process of automatically managing, coordinating, and scaling containerized applications. Think of it as a smart manager for your containers that handles tasks like:

- Starting and stopping containers: Ensuring your applications are running smoothly.
- Scaling: Adding or removing containers based on how much load your application is experiencing.
- Networking: Making sure all containers can talk to each other and the outside world.
- Health monitoring: Checking if your containers are working properly and restarting them if they fail.
- Updating applications: Rolling out updates to your applications without causing downtime.

more than 1 containers

But when do we need container orchestration?

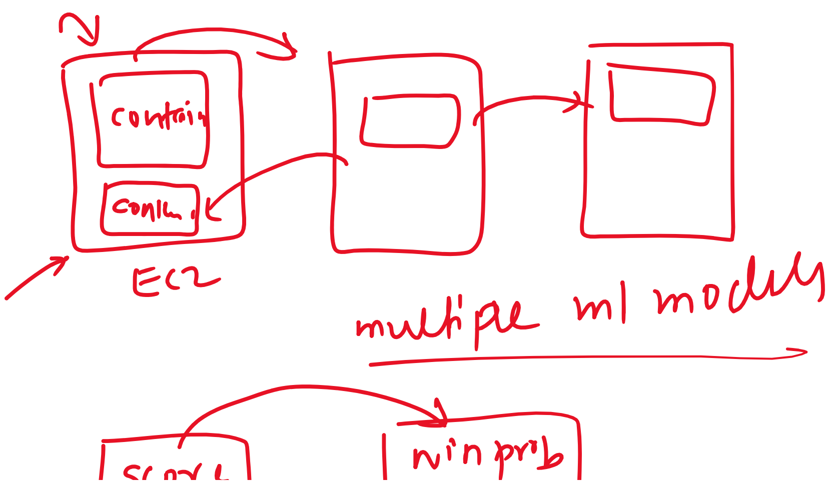


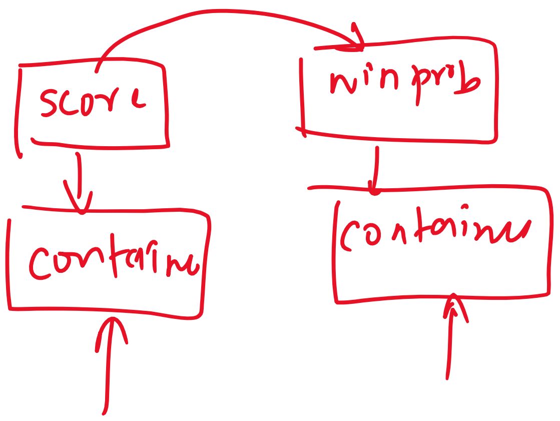
Scenario 1 - Highly Availability & High Scalability requirement for a service

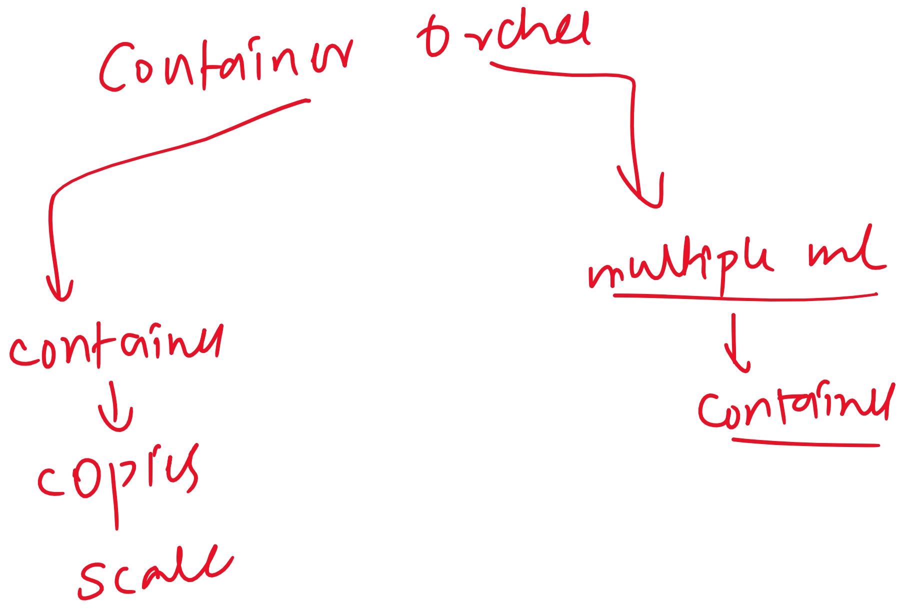
Consider a scenario where you have a single web application service that needs to be highly available and scalable. Here's how container orchestration can help:

1. High Availability: The orchestration platform monitors the health of your web application container and automatically restarts it if it crashes.
2. Auto-scaling: During peak times, such as a marketing campaign or a sale event, the platform can automatically scale the number of instances of your web application container to handle the increased traffic.
3. Rolling Updates: When you release a new version of your web application, the orchestration platform can perform a rolling update, gradually replacing old containers with new ones, ensuring zero downtime.
4. Resource Optimization: The platform allocates resources dynamically based on the current load and resource availability, ensuring efficient use of infrastructure.

Scenario 2 - Microservices

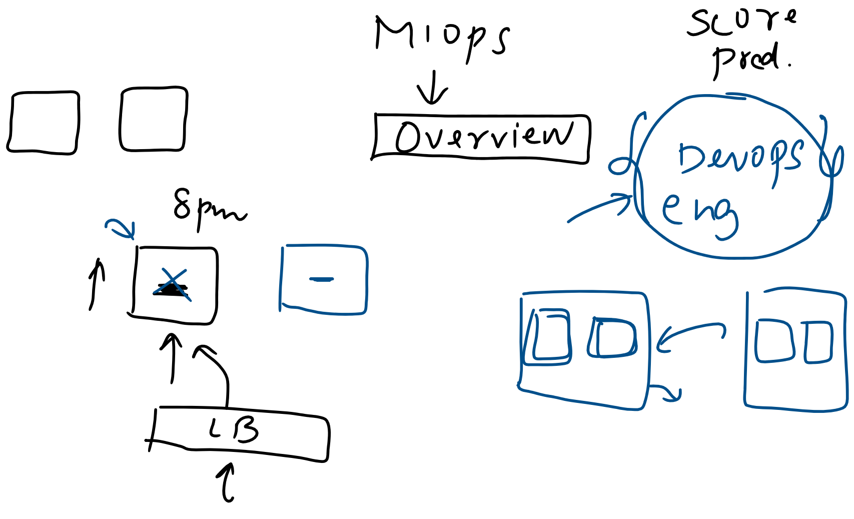



score
win prob
container
container


Container orchel
container
copies
scale
multiple ml
container

Recap

31 May 2024 08:14

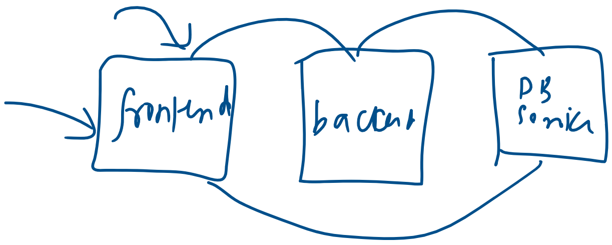


- What are containers
- Why do we need containers
- Container Orchestration
- Role of a container orchestration service

1. Scaling
2. Load balancing
3. Communication between containers
4. Self-healing
5. Resource management
6. Security
7. Monitoring
8. DevOps Integration (with CI/CD pipelines, support for different types of deployment like blue green deployment)

Scenarios where we use multiple containers

1. One container scaled to multiple containers
2. Multiple connected services running on different containers
3. Microservices architecture



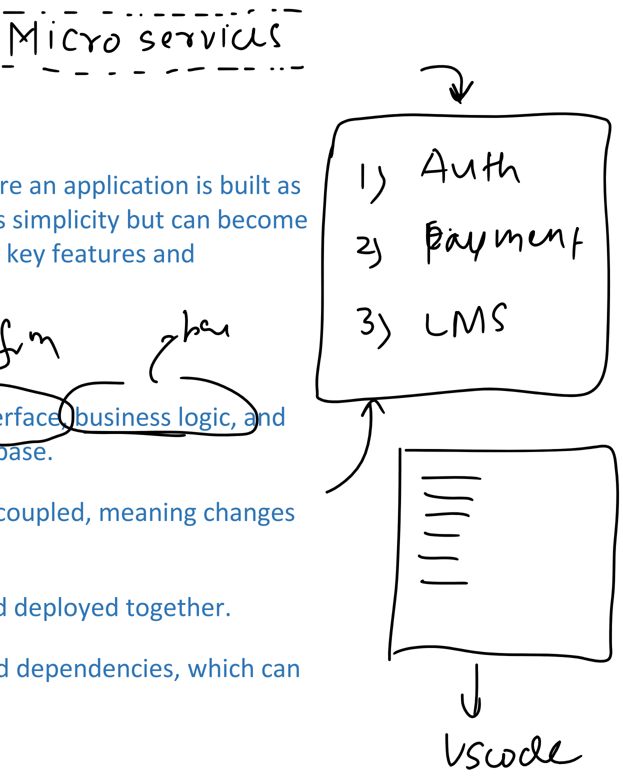
Monolithic Architecture

27 May 2024 11:15

Monolithic architecture is a traditional software design model where an application is built as a single, unified unit. This type of architecture is characterized by its simplicity but can become difficult to manage and scale as the application grows. Here are the key features and characteristics of monolithic architecture:

Key Characteristics:

1. Single Codebase: The entire application, including the user interface, business logic, and data access layers, is developed and deployed as a single codebase.
2. Tight Coupling: Components within the application are tightly coupled, meaning changes in one part of the application can impact other parts.
3. Single Deployment Unit: The entire application is packaged and deployed together.
4. Same tech stack: All components share the same resources and dependencies, which can lead to conflicts and challenges in dependency management.

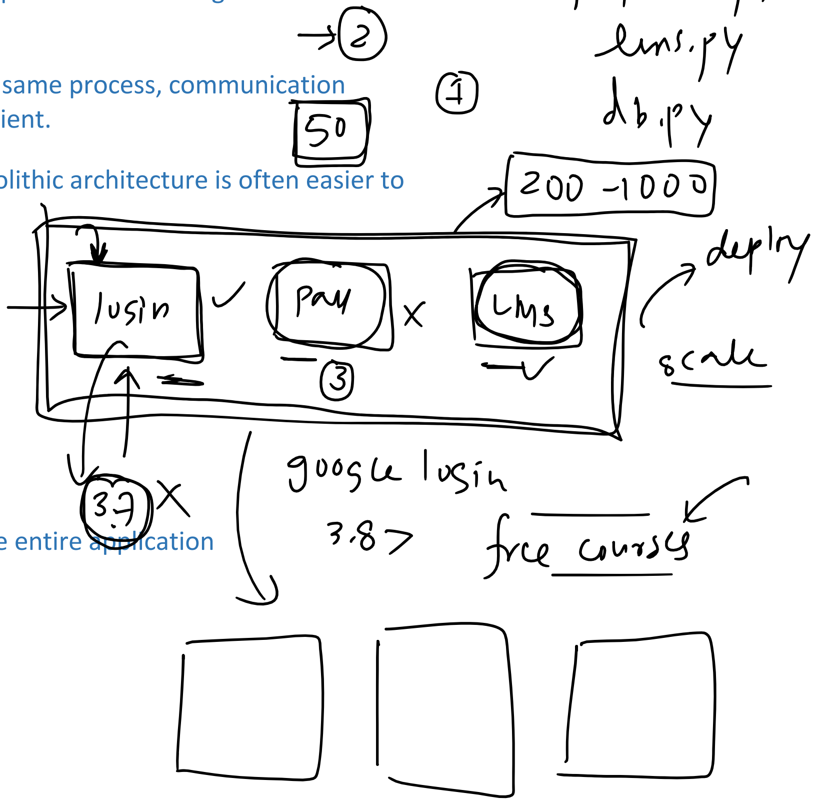


Advantages:

1. Simplicity: Development, testing, and deployment processes are straightforward due to the single codebase and deployment unit.
2. Performance: Since all components run within the same process, communication between components can be faster and more efficient.
3. Ease of Development: For small applications, monolithic architecture is often easier to develop and manage.

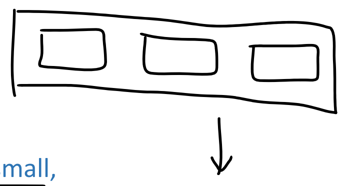
Disadvantages

1. Redeployment
2. Handling scalability
3. Handling different dependencies
4. Complexity grows -> collaboration issues
5. Restriction to use the same tech stack
6. A bug in one part of the system can bring down the entire application

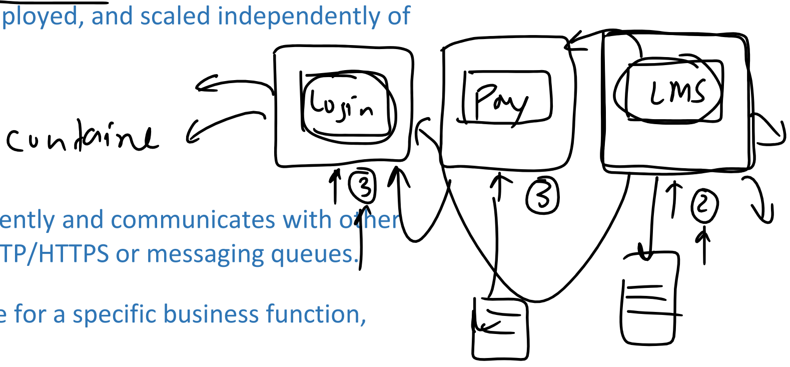


Microservices

27 May 2024 11:15

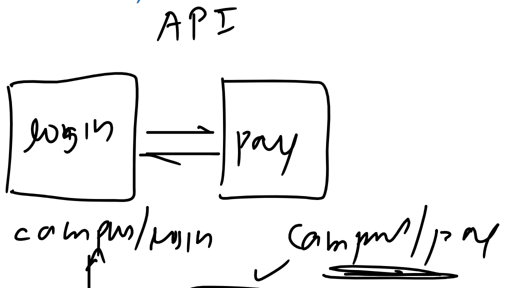


Microservices architecture is a design approach where an application is composed of small, independent services that work together to form a complete system. Each service is focused on a specific business capability and can be developed, deployed, and scaled independently of the other services.



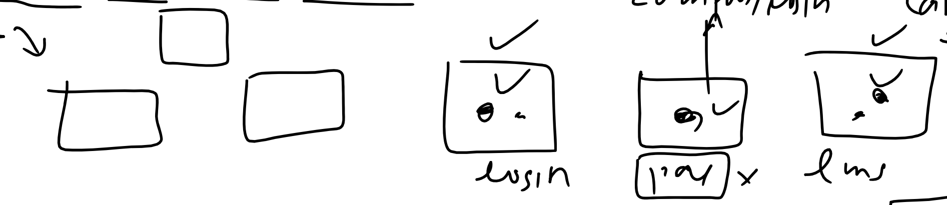
Key Characteristics of Microservices Architecture:

1. Independence: Each microservice operates independently and communicates with other services through well-defined APIs, typically using HTTP/HTTPS or messaging queues.
2. Single Responsibility: Each microservice is responsible for a specific business function, adhering to the Single Responsibility Principle.
3. Decentralized Data Management: Each microservice can have its own database or data storage solution, tailored to its specific needs.
4. Scalability: Services can be scaled independently, allowing for more efficient use of resources and the ability to handle different loads on different parts of the application.
5. Technology Diversity: Different microservices can be developed using different programming languages, frameworks, or databases, allowing teams to choose the best tools for each specific task.
6. Autonomous Teams: Development teams can work independently on different services, speeding up development and deployment cycles.

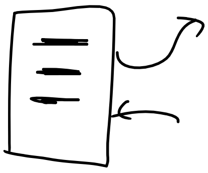


How does microservices communicate with each other?

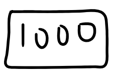
1. APIs
2. Message Brokers [RabbitMQ, Redis, AWS SQS, Apache Kafka]
3. Service Mesh [Istio]
4. Web sockets

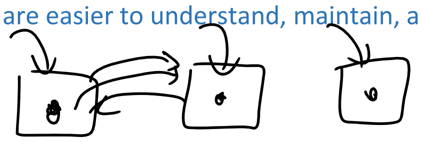


Advantages:



1. Scalability: You can scale individual microservices based on their specific demand, rather than scaling the entire application.
2. Flexibility: Teams can use different technologies and frameworks for different services, allowing for more innovation and optimization.
3. Resilience: A failure in one microservice doesn't necessarily bring down the entire application, improving overall system reliability.
4. Faster Time to Market: Independent development and deployment of services can lead to faster release cycles.
5. Easier Maintenance: Smaller codebases are easier to understand, maintain, and update





Disadvantages:

1. Complexity: Managing multiple services and their interactions can be complex and requires robust DevOps practices.
2. Distributed Systems Challenges: Issues like network latency, security, and data

1. Complexity: Managing multiple services and their interactions can be complex and requires robust DevOps practices.

2. Distributed Systems Challenges: Issues like network latency, security, and data consistency need careful handling.

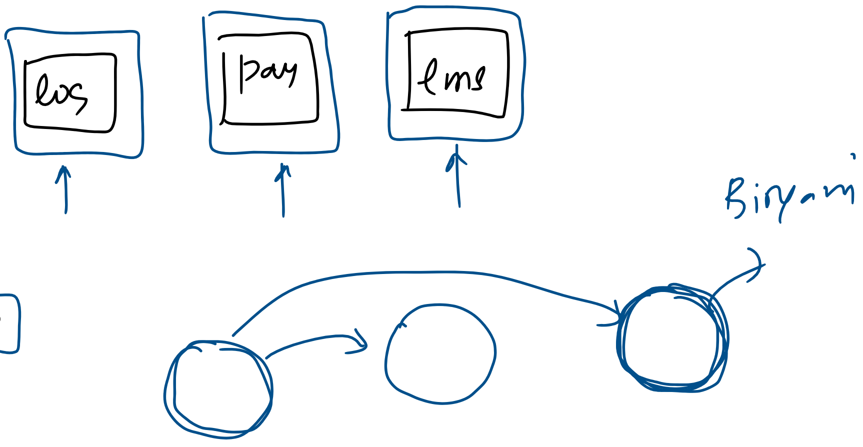
3. Increased Resource Usage: Running multiple microservices can lead to higher resource consumption compared to a monolithic application.

4. Monitoring and Debugging: Tracing and debugging issues across multiple services can be challenging and requires specialized tools.

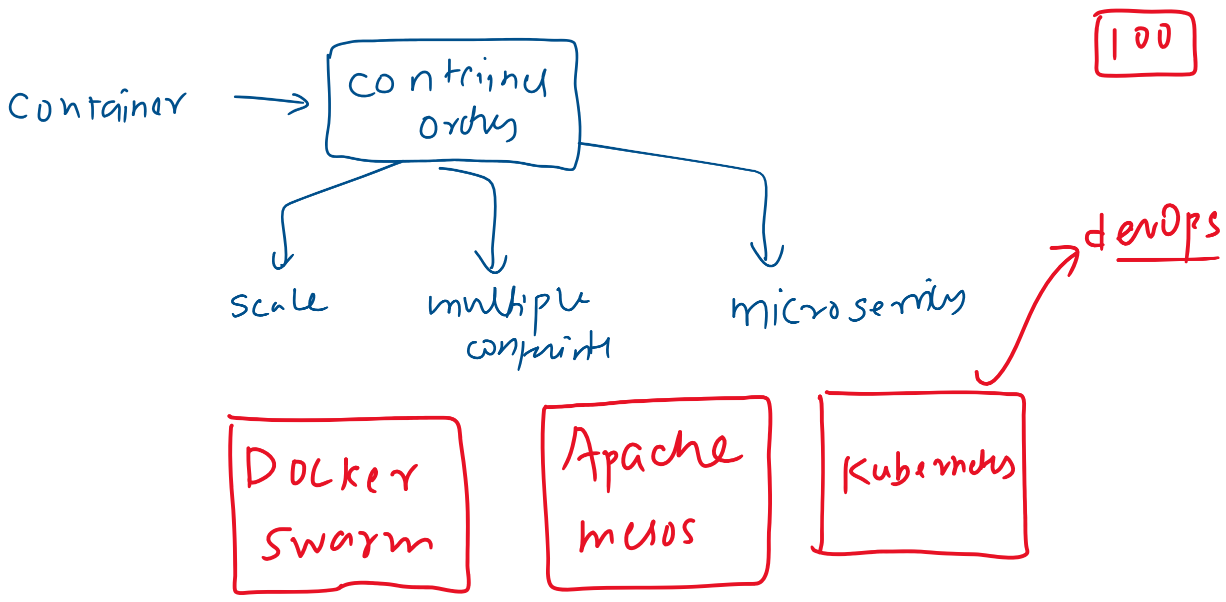
How does containers fit into the microservices architecture?

In a microservices architecture, each microservice is packaged into its container. This strategy has the following benefits:

1. Isolation
2. Portability
3. Scalability
4. Ease of deployment
5. Fault tolerance



How are microservices used in the ML universe?



Kubernetes

30 May 2024 00:32

Kubernetes, often abbreviated as K8s, is an open-source container orchestration platform designed to automate deploying, scaling, and managing containerized applications.

google
↓
internal

Early Development

1. Origins at Google: Kubernetes' roots can be traced back to an internal Google project called Borg, which was developed in the early 2000s. Borg was a cluster management system used to deploy and manage thousands of applications at Google, providing a foundation for container orchestration at scale. Following Borg, Google developed Omega, an even more flexible and scalable cluster management system. Omega incorporated lessons learned from Borg and introduced new features and architectural improvements.

Open Source and Initial Release

1. 2014 - Project Launch: Google initiated the Kubernetes project in mid-2014, releasing it as an open-source project to bring container orchestration technology to the broader developer community.
2. July 2015 - Version 1.0: Kubernetes reached version 1.0, marking its readiness for production use. Around the same time, the Cloud Native Computing Foundation (CNCF) was established, with Kubernetes as its first project.

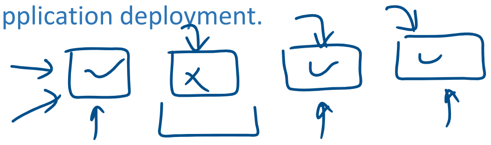
Growth and Adoption

EKS

1. 2016-2017 - Rapid Adoption: Kubernetes quickly gained traction among developers and enterprises, becoming the dominant container orchestration platform. Major cloud providers began offering managed Kubernetes services.

Maturity and Ubiquity

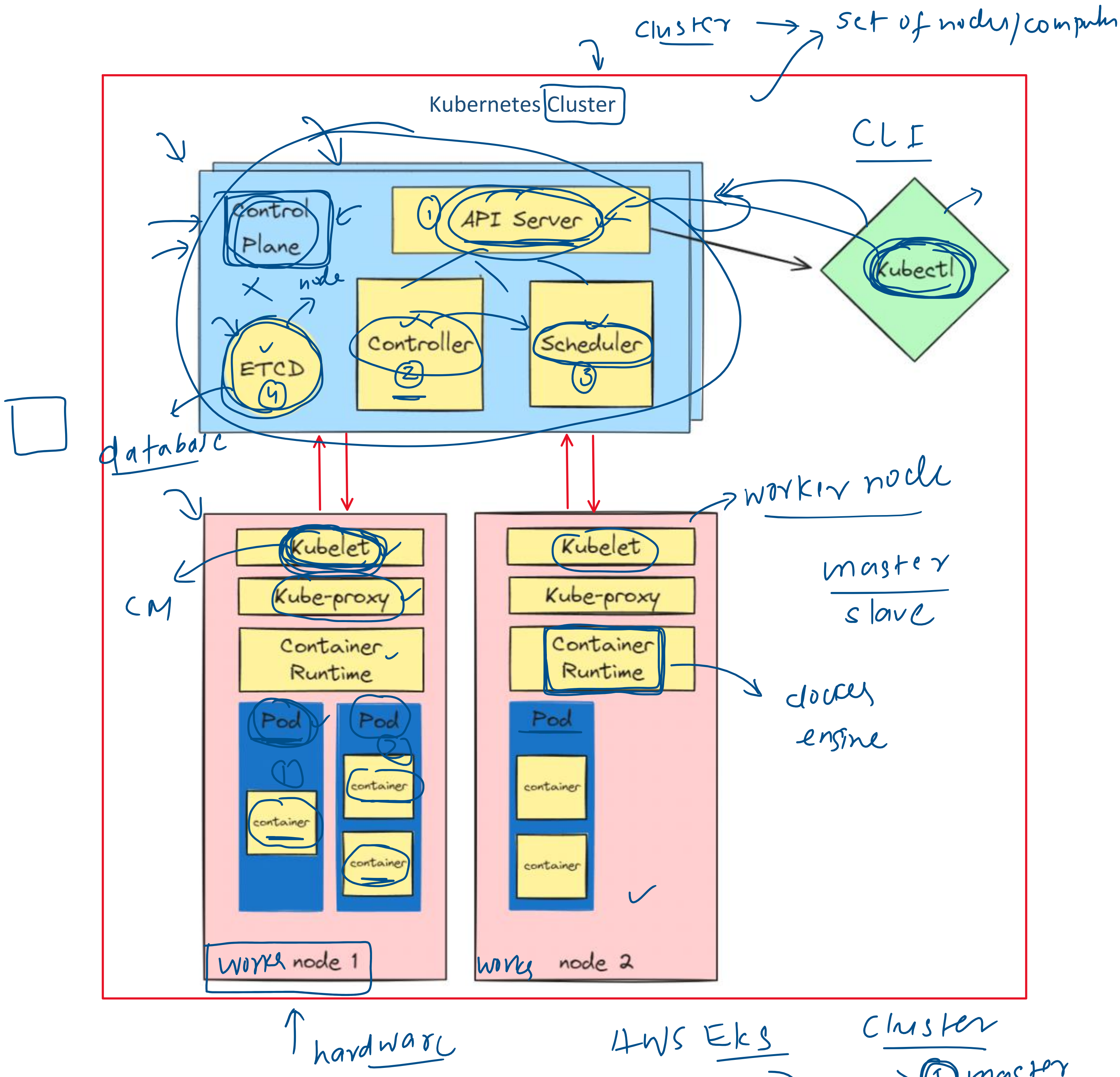
1. 2018-Present - Continued Evolution: Kubernetes has continued to evolve with regular releases, adding new features and improving stability. It is now a fundamental technology for cloud-native architectures and modern application deployment.



Kubernetes Features

- ✓ Scalability: Kubernetes can scale applications automatically based on demand. It monitors the resource usage of containers and can dynamically allocate resources as needed.
- ✓ Self-Healing: Kubernetes can detect when a container or node fails and automatically restart or replace it to maintain the desired state of the application.
- ✓ Container Orchestration: Kubernetes helps manage and coordinate containers across a cluster of machines. This includes scheduling containers to run on specific nodes, scaling applications up or down by adding or removing containers, and managing the lifecycle of containers.
  ×
- ✓ Deployment Automation: Kubernetes provides mechanisms to automate the deployment of applications. This includes rolling updates and rollbacks, ensuring that the application is always in a desired state or highly available.
- Service Discovery and Load Balancing: Kubernetes provides built-in service discovery and load balancing. It assigns IP addresses to containers and can distribute network traffic among them to ensure even load distribution.

Kubernetes Architecture



Cluster - A Kubernetes cluster is a set of nodes (machines) that run containerized applications managed by Kubernetes. It consists of a master node and several worker node.

200

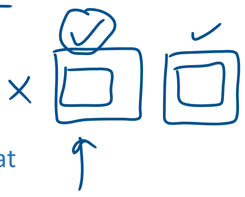
Cluster
→ 1 master node
control plane
→ multiple worker nodes
→ multiple pods

Control Plane - The control plane in Kubernetes is the central management entity that regulates and maintains the desired state of the Kubernetes cluster. It consists of several components that work together to ensure that the cluster operates correctly, managing the lifecycle of containers, scheduling workloads, and handling all the administrative tasks.

Worker Node - In a Kubernetes cluster, worker nodes are the machines (either physical or virtual) where the actual containerized applications run. Each worker node contains several key components that allow it to interact with the control plane and manage the lifecycle of the containers.

node contains several key components that allow it to interact with the control plane and manage the lifecycle of the containers.

Kube-apiserver - The API server (kube-apiserver) is the core component of the Kubernetes control plane, providing the primary interface for all interactions with the cluster. It handles requests, manages authentication and authorization, validates data, and serves as the communication hub for the entire cluster.



→ can have multiple containers

Controller - In the Kubernetes control plane, controllers are processes that continuously monitor the state of the cluster and make changes to ensure that the cluster's actual state matches the desired state defined by the user. Controllers automate the operational tasks in the cluster, such as managing replicas, ensuring nodes are healthy, and handling various resource lifecycles.

- Desired State: Users define the desired state of the cluster by creating and modifying Kubernetes objects (e.g., deployments, services, jobs) through the API server.
- Monitoring: Controllers continuously watch the current state of these objects by querying the API server. They look for any discrepancies between the desired state and the current state.
- Reconciliation Loop: Each controller runs a reconciliation loop, a process that repeatedly checks the current state of the cluster against the desired state. If the current state does not match the desired state, the controller takes corrective actions.
- Corrective Actions: Controllers make the necessary changes to bring the cluster back to the desired state. For example, if a pod fails, the replication controller will create a new pod to maintain the specified number of replicas.

Kube-scheduler - The Kubernetes scheduler, known as kube-scheduler, is a crucial component of the Kubernetes control plane. Its primary role is to assign pods to nodes within the cluster. The scheduler ensures that pods are placed on appropriate nodes based on various factors such as resource requirements, constraints, and policies.

Etcd - etcd is a distributed, reliable key-value store that is used to store all the configuration data and state information for a Kubernetes cluster. It is a crucial component of the Kubernetes control plane, ensuring that all cluster data is consistently replicated across the control plane nodes.

Kubectl - kubectl is a command-line tool that allows you to interact with and manage Kubernetes clusters. It provides a powerful interface for performing various administrative tasks, deploying applications, inspecting and managing cluster resources, and troubleshooting issues.

Pods - Pods are the smallest and most basic deployable units in Kubernetes. A pod represents a single instance of a running process in your cluster and can contain one or more tightly coupled containers. Pods provide a level of abstraction for grouping containers that share resources such as networking and storage.

Kubelet - The kubelet is a vital component of the Kubernetes architecture that runs on every worker node. It manages the lifecycle of pods, ensures containers are running and healthy, and communicates with the Kubernetes API server to report node status and receive instructions. By managing resources, health checks, and storage volumes, the kubelet ensures that the node operates correctly within the Kubernetes cluster.

Kube-proxy - kube-proxy is an essential component of the Kubernetes networking model. It runs on every node and ensures that traffic is correctly routed to the appropriate pods based on the service specifications. By managing network rules and implementing load balancing, kube-proxy helps maintain reliable and efficient network communication within the Kubernetes cluster.

appropriate pods based on the service specifications. By managing network rules and implementing load balancing, kube-proxy helps maintain reliable and efficient network communication within the Kubernetes cluster.

Container Orchestration Services

31 May 2024 08:15

①

kubernetes

✓

docker SWARM

Apache MESOS

AWS ECS

Azure Kubernetes Service (AKS)

Amazon EKS

Google Kubernetes Engine

OpenShift

non-kube

HashiCorp Nomad

non-kube

✓

Why do managed Kubernetes services exist?

1. Simplified Management ✓
2. Higher security
3. Integration of cloud services
4. Technical support

$\boxed{\text{EKS}}$

Kubernetes

$\boxed{\text{EKS}}$