

Open in app ↗



Search

Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)

Kubernetes: Architecture and Components explained

WHAT IS KUBERNETES?

Himanshu Sangshetti · [Follow](#)

8 min read · Aug 18, 2023



Listen



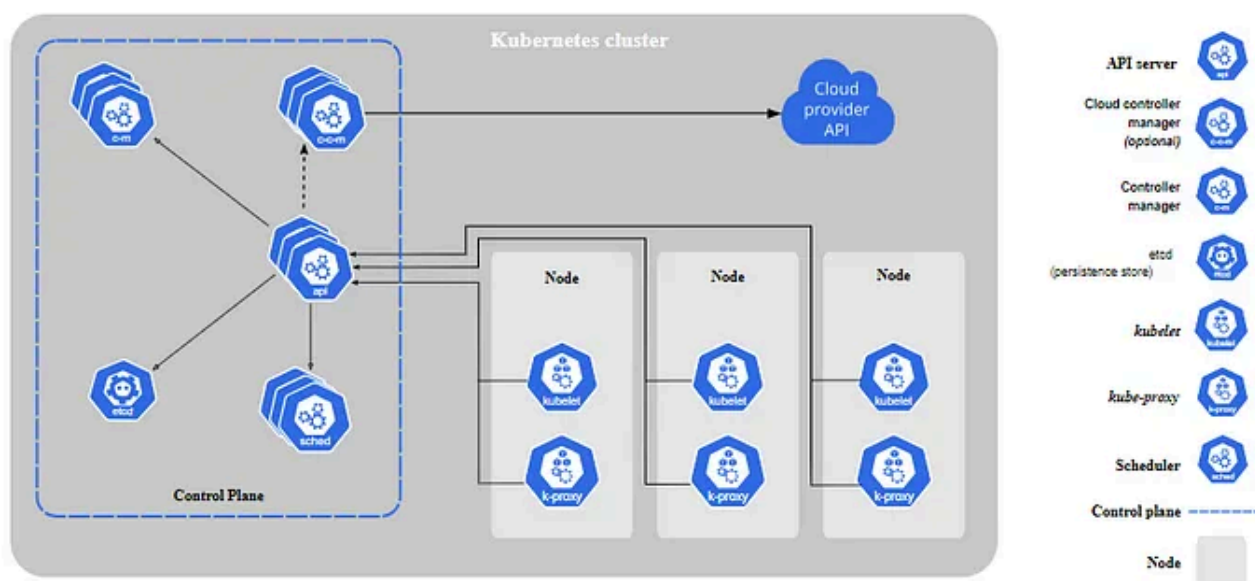
Share



More

Kubernetes, often abbreviated as K8s, is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. It provides a framework to efficiently manage the complexities of deploying and running applications in containers across a cluster of machines.

KUBERNETES ARCHITECTURE AND COMPONENTS:



— CONTROL PLANE:

The master control plane is the central management unit of a Kubernetes cluster. It oversees the entire cluster's operation and coordination, making high-level decisions about how applications should run and ensuring that the actual state of the cluster matches the desired state.

1. **API Server:**The API Server acts as the entry point for interactions with the Kubernetes cluster. It exposes the Kubernetes API, which allows users, administrators, and other components to communicate with the cluster.
2. **etcd:**etcd is a distributed key-value store that serves as Kubernetes' backing store for all cluster data. It holds the configuration data and the state of the entire cluster. This includes information about Pods, Services, replication settings, and more.
3. **Controller Manager:**The Controller Manager is responsible for monitoring the state of various objects in the cluster and taking corrective actions to ensure the desired state is maintained. It includes several built-in controllers, such as the Replication Controller, Deployment Controller, and StatefulSet Controller.
4. **Scheduler:**The Scheduler is responsible for placing Pods onto suitable worker nodes. It takes into account factors like resource availability, constraints, and optimization goals.

Together, these components form the master control plane, which acts as the brain and command center of the Kubernetes cluster.

— WORKER NODES:

Worker nodes, also known as worker machines or worker servers, are the heart of a Kubernetes cluster. They are responsible for running containers and executing the actual workloads of your applications.

1. **Kubelet:** The Kubelet is an agent that runs on each worker node and communicates with the master control plane. Its primary responsibility is to ensure that containers within Pods are running and healthy as per the desired state defined in the cluster's configuration. The Kubelet works closely with the master control plane to start, stop, and manage containers based on Pod specifications.
2. **Kube Proxy:** Kube Proxy sets up routing and load balancing so that applications can seamlessly communicate with each other and external resources.

3. **Container Runtime:** A container runtime is the software responsible for running containers on the worker nodes.
4. **Container Storage Interfaces (CSI):** Worker nodes need to provide storage for persistent data. Kubernetes supports different storage solutions through Container Storage Interfaces (CSI). These interfaces allow different storage providers to integrate with Kubernetes and offer persistent storage volumes for applications.

— PODS:

Pods are fundamental building blocks in Kubernetes that group one or more containers together and provide a shared environment for them to run within the same network and storage context.

1. **Container Co-location:** Pods allow you to colocate containers that need to work closely together within the same network namespace. This means they can communicate using localhost and share the same IP address and port space.
2. **Shared Storage and Volumes:** Containers within a Pod share the same storage volumes, which allows them to easily exchange data and files. Volumes are attached to the Pod and can be used by any of the containers within it.
3. **Single Unit of Deployment:** Kubernetes schedules Pods as the smallest deployable unit. If you want to scale or manage your application, you work with Pod replicas, not individual containers.
4. **Init Containers:** A Pod can include init containers, which are containers that run before the main application containers.

— CONTROLLER:

In Kubernetes, controllers are crucial components responsible for maintaining the desired state of resources in the cluster. They monitor changes to resources and ensure that the actual state matches the intended state specified in the cluster's configuration. Controllers help automate tasks like scaling, self-healing, and application management.

1. **Replication Controller:** The Replication Controller ensures a specified number of identical replica Pods are running at all times. If a Pod fails or is deleted, the Replication Controller creates a new one to maintain the desired number.

2. **Deployment:** Deployments provide declarative updates to applications. They allow you to define how many replica Pods should be running and how to update them over time. Deployments support rolling updates, which gradually replace old Pods with new ones to minimize service disruption. You can easily roll back to a previous version if issues arise.
3. **StatefulSet:** StatefulSets are used to manage stateful applications that require unique identities and persistent storage. They ensure that each Pod gets a stable, unique hostname and are created and scaled in a predictable order.
4. **DaemonSet:** DaemonSets ensure that a specific Pod runs on every node in the cluster. They're useful for deploying monitoring agents, log collectors, or networking components that require presence on all nodes.
5. **Horizontal Pod Autoscaler (HPA):** The HPA controller automatically scales the number of replica Pods based on CPU usage or custom metrics. It ensures that your application adapts to varying levels of traffic and demand by increasing or decreasing the number of running Pods.
6. **Vertical Pod Autoscaler (VPA):** The VPA controller adjusts resource requests and limits for Pods based on their actual usage. It optimizes resource allocation to improve performance and efficiency.

— SERVICES:

In Kubernetes, Services are a fundamental concept that enables communication and load balancing between different sets of Pods, making your applications easily discoverable and resilient.

1. **Service Types:** There are different types of Services in Kubernetes, each designed for specific use cases:
 - **ClusterIP:** Creates an internal virtual IP that exposes the Service within the cluster. This type is suitable for communication between different parts of your application within the cluster.
 - **NodePort:** Exposes the Service on a static port on each node's IP. This makes the Service accessible from outside the cluster, typically for development or testing purposes.

- **LoadBalancer:** Creates an external load balancer that distributes traffic to the Service across multiple nodes. This type is useful when you need to expose your Service to the internet or an external network.
- **ExternalName:** Maps the Service to a DNS name, allowing you to reference services external to the cluster.

2. **Selectors and Labels:** Services use selectors and labels to identify the Pods they should target. Labels are key-value pairs attached to Pods, and selectors define which Pods the Service should include. For example, a Service with a selector might target all Pods with the label “app=web.”

3. **Load Balancing:** Services provide load balancing across multiple Pods with the same label. When you send traffic to a Service, it distributes the traffic evenly among the available Pods. This ensures that no single Pod gets overwhelmed with requests.

4. **Headless Services:** A Headless Service is a special type of Service that doesn't load balance or provide a stable IP. Instead, it allows you to access individual Pods directly using their IPs or DNS names. This is useful for applications that require direct communication with specific Pods.

— VOLUMES:

Volumes are a way to provide persistent storage to containers within Pods. They enable data to be shared and preserved across container restarts, rescheduling, and even Pod failures. Volumes enhance the flexibility and reliability of containerized applications.

1. **Types of Volumes:** Kubernetes supports various types of volumes to accommodate different storage needs:

- **EmptyDir:** A temporary storage that's created when a Pod is assigned to a node and deleted when the Pod is removed or rescheduled.
- **HostPath:** Mounts a file or directory from the host machine's filesystem into the container. Useful for testing and development, but not recommended for production due to portability and security concerns.
- **PersistentVolumeClaim (PVC):** An abstract way to request and manage a persistent storage volume. PVCs can dynamically provision storage based on a

predefined storage class.

- **ConfigMap and Secret Volumes:** Special volumes that allow you to inject ConfigMap or Secret data as files into containers.
- **NFS:** Network File System volumes allow you to use network-attached storage in your containers.
- **CSI (Container Storage Interface) Volumes:** A plugin framework that allows storage providers to develop volume plugins without having to modify Kubernetes core code.

— CONFIGMAPS AND SECRETS:

1. **ConfigMaps:** ConfigMaps are used to store non-sensitive configuration data as key-value pairs. This data can include settings, environment variables, command-line arguments, configuration files, or any other configuration-related information that your application needs.
2. **Secrets:** Secrets are similar to ConfigMaps but are specifically designed for storing sensitive information, such as passwords, API keys, certificates, and tokens. Secrets provide an additional layer of security by ensuring that sensitive data is not stored in plain text within your container images or configuration files.

— NAMESPACES:

In Kubernetes, namespaces are a way to organize and partition resources within a cluster. They provide a way to create multiple virtual clusters within the same physical cluster, allowing you to separate and manage resources for different teams, projects, or environments.

1. Purpose of Namespaces:

Namespaces serve several purposes:

- **Resource Isolation:** Namespaces create isolated environments, so resources in one namespace are distinct from those in another. This helps prevent resource conflicts and accidental interference.
- **Resource Management:** Namespaces help organize and manage resources more effectively, especially in large and complex deployments.

- **Access Control:** Namespaces enable fine-grained access control. You can grant different teams or users access only to specific namespaces and the resources within them.
- **Tenant Separation:** If you're using Kubernetes to offer services to multiple customers or tenants, namespaces help isolate their resources.

2. Default Namespace: When you create a Kubernetes cluster, there's a default namespace where resources are created if you don't specify a namespace explicitly. It's recommended to use namespaces to keep your cluster organized, even if you're not dealing with multiple tenants or teams.

3. Namespace Scope: Certain resources, like Nodes and PersistentVolumes, are not tied to a specific namespace and are accessible from all namespaces. Other resources, such as Pods, Services, ConfigMaps, and Secrets, belong to a specific namespace.

4. Access Control and Quotas: Namespaces allow you to implement Role-Based Access Control (RBAC) to manage who can access or modify resources within a namespace. Additionally, you can set resource quotas and limits on namespaces to prevent resource overuse.

— INGRESS:

Ingress is a resource that manages external access to services within your cluster. It acts as a way to configure and manage routing rules for incoming traffic, allowing you to expose your services to the outside world, often using HTTP and HTTPS protocols.

1. Purpose of Ingress: The primary purpose of Ingress is to provide a way to manage external access to your services without modifying the application code. It serves as an entry point to your cluster, allowing you to route traffic to the appropriate services based on rules you define.

2. How Ingress Works: Ingress resources define rules that map incoming requests to specific services. These rules are based on hostnames, paths, and other criteria. When a request matches a rule, the Ingress controller routes the traffic to the appropriate service or backend.

3. Ingress Controller: An Ingress resource doesn't actually handle the traffic routing; it's the responsibility of an Ingress controller. The Ingress controller watches for changes to Ingress resources and configures the underlying infrastructure to implement the defined routing rules.

Basic understanding about the Kubernetes components was understood, I hope you found this blog helpful.

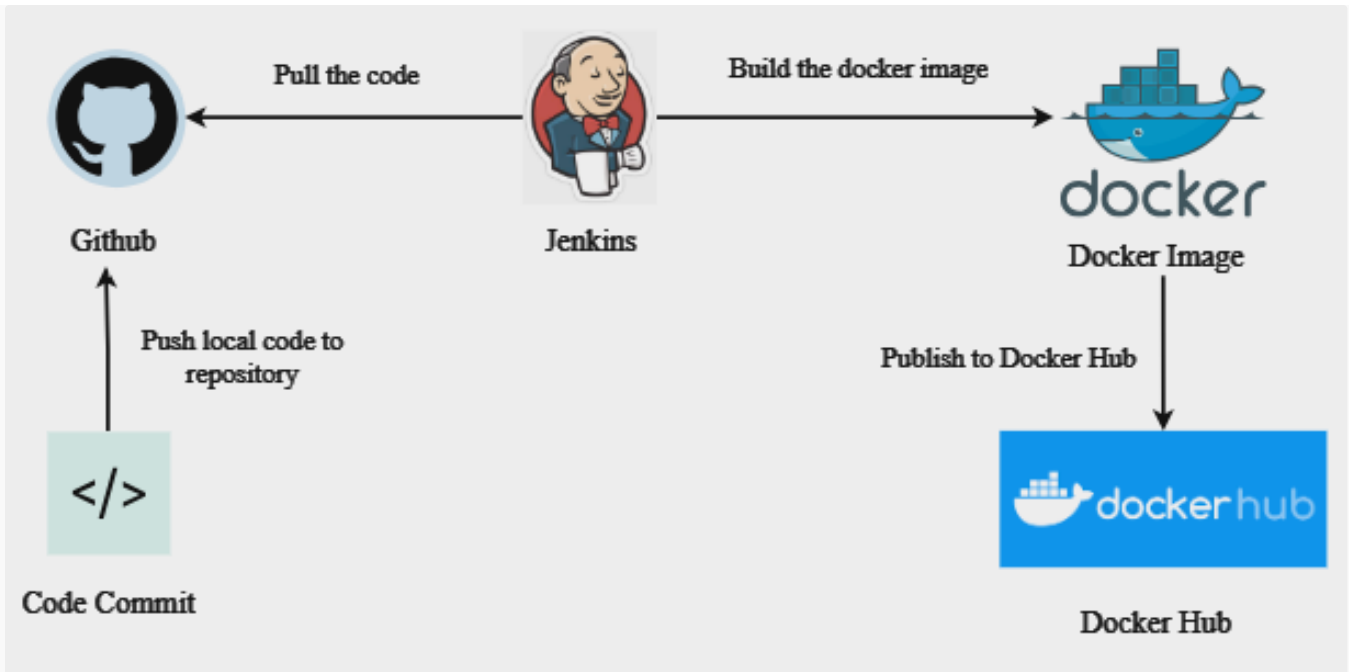
[Kubernetes](#)[Kubernetes Architecture](#)[Kubernetes Components](#)[Follow](#)

Written by Himanshu Sangshetti

40 Followers

Cloud enthusiast

More from Himanshu Sangshetti



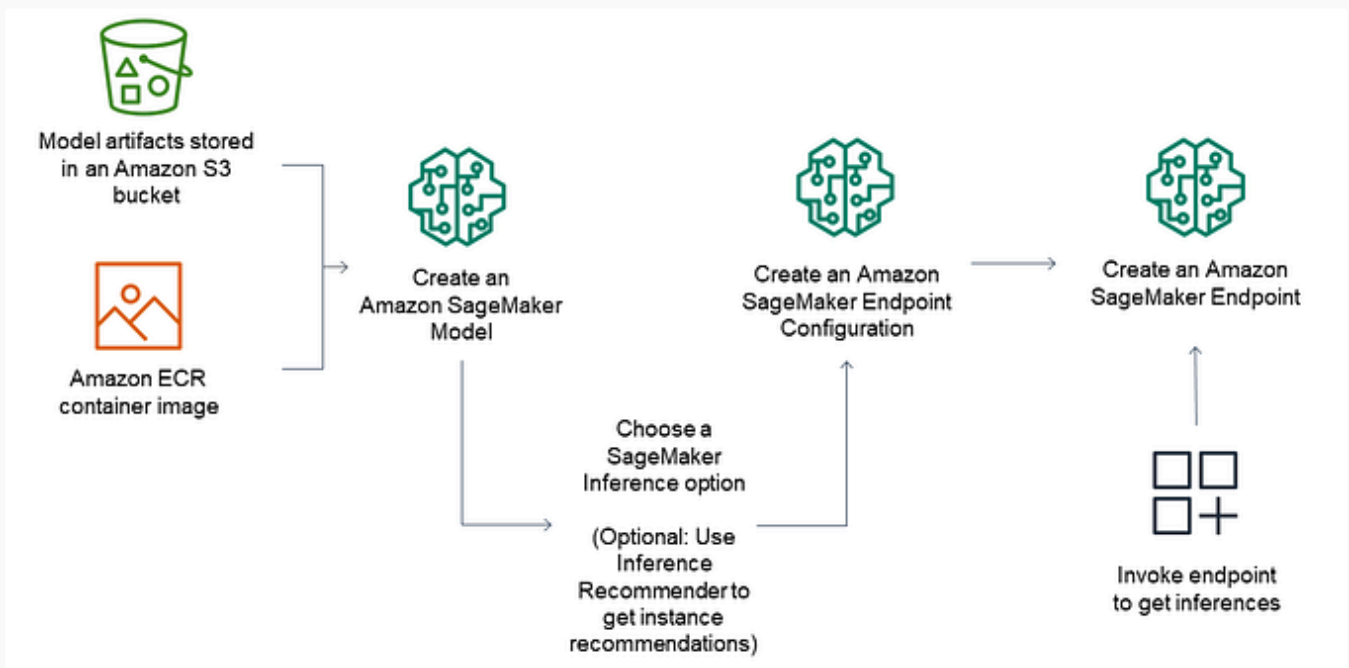
Himanshu Sangshetti

Jenkins CI/CD with Github integration on AWS EC2 instance

Jenkins CI/CD with GitHub integration on an AWS EC2 instance is a powerful combination that automates the build, test, and deployment...

5 min read · Aug 3, 2023

7



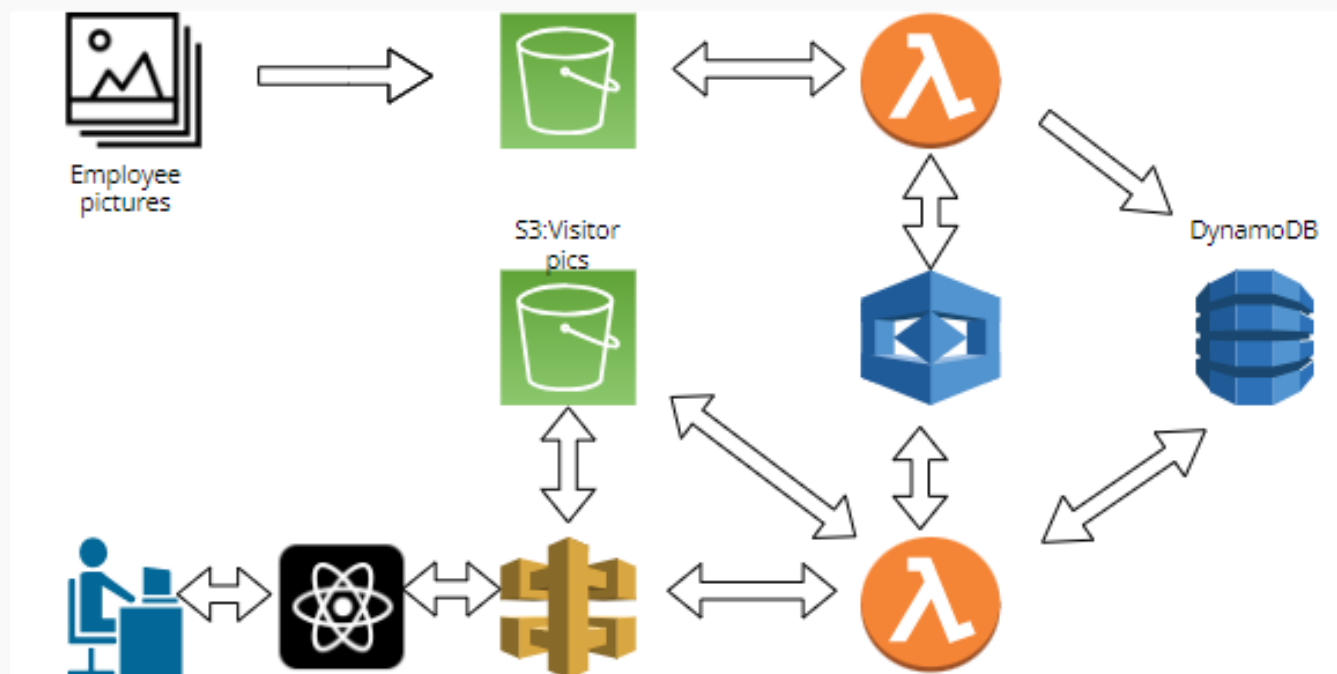
Himanshu Sangshetti

Deploying a ML model on Amazon SageMaker

Amazon SageMaker is a fully managed service provided by Amazon Web Services (AWS) for building, training, and deploying machine learning...

6 min read · Dec 18, 2023

85



Himanshu Sangshetti

Building facial recognition app on AWS using Rekognition, Lambda, DynamoDB, API Gateway, S3

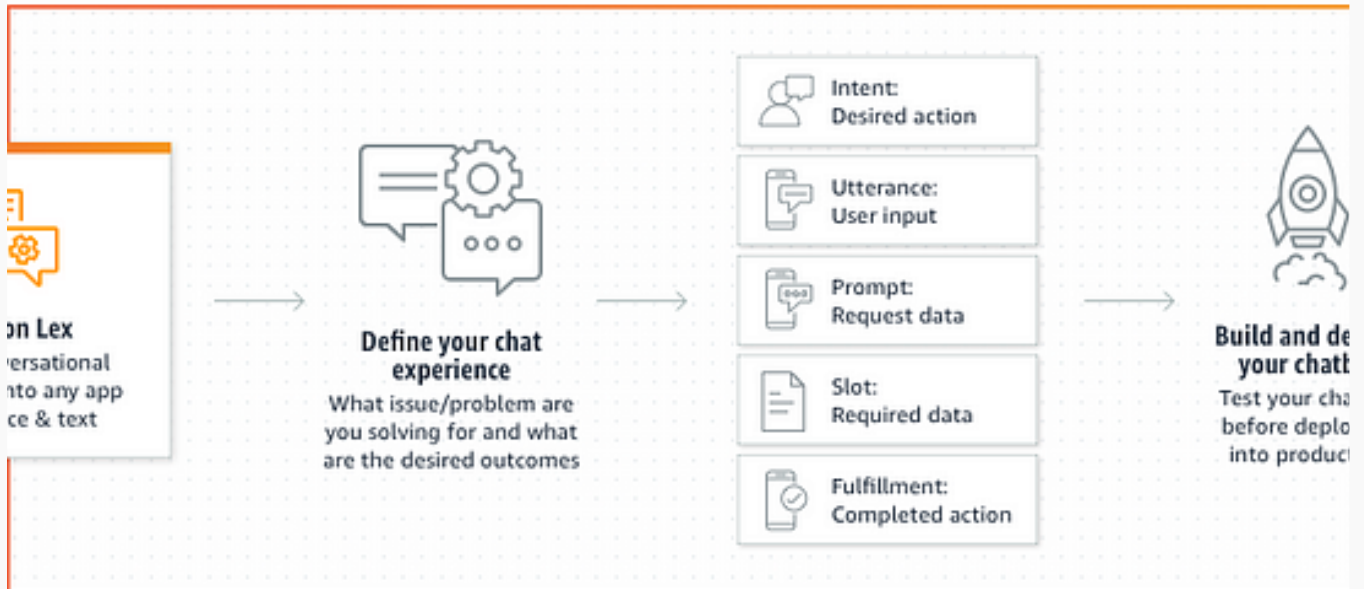
Explore how to build a Facial Recognition App on AWS using services Amazon Rekognition, AWS Lambda, Amazon DynamoDB with a React front end.


6 min read · Jul 17, 2023

27

1





 Himanshu Sangshetti

Custom pizza ordering chatbot using Amazon Lex

Amazon Lex is a fully managed artificial intelligence (AI) service with advanced natural language models to design, build, test, and deploy

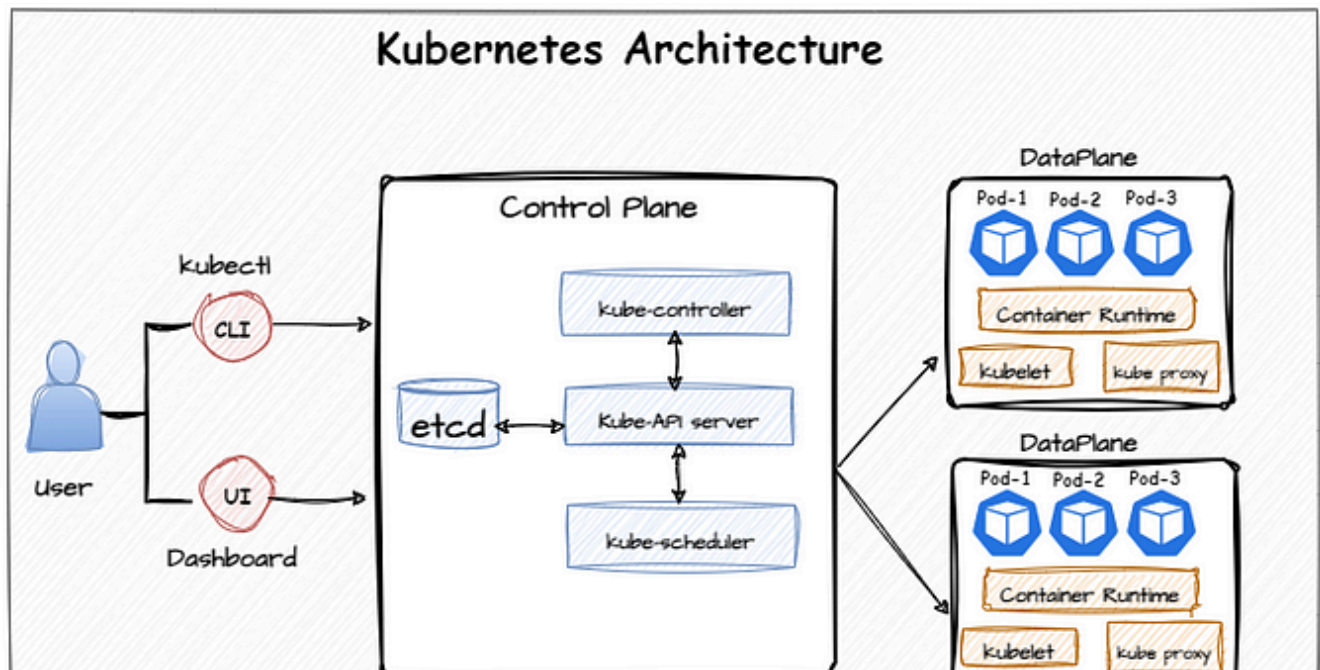
4 min read · Jul 29, 2023

 1  1

See all from Himanshu Sangshetti

Recommended from Medium



 VenKube in FAUN—Developer Community 🍷

Kubernetes Chronicles: (K8s#01)—Introduction to Kubernetes | Architecture.

Kubernetes also known as K8s, is an open-source container orchestration platform that automates the deployment, management, scaling and...

6 min read · Nov 26, 2023



Booking.com



Talha Şahin

High-Level System Architecture of Booking.com

Take an in-depth look at the possible high-level architecture of Booking.com.

8 min read · Jan 10, 2024

👏 4.8K 💬 39

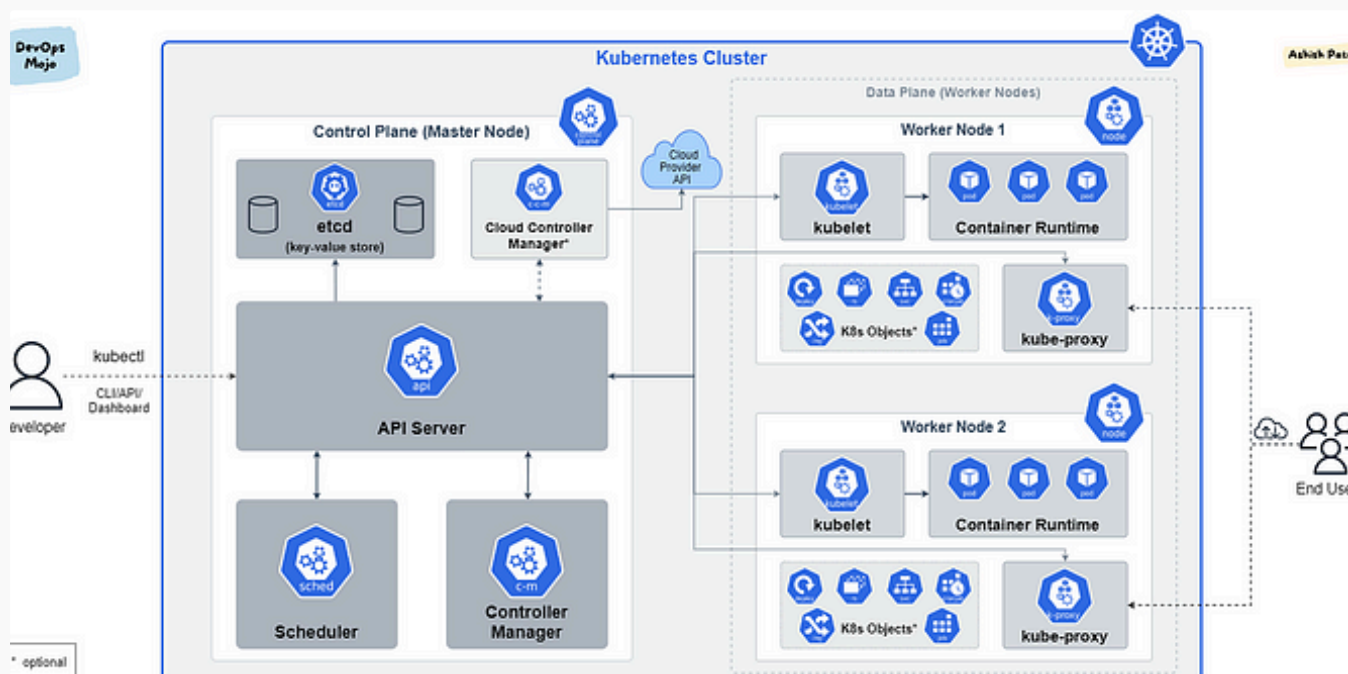


Lists



Natural Language Processing

1471 stories · 983 saves



Kavishka Fernando

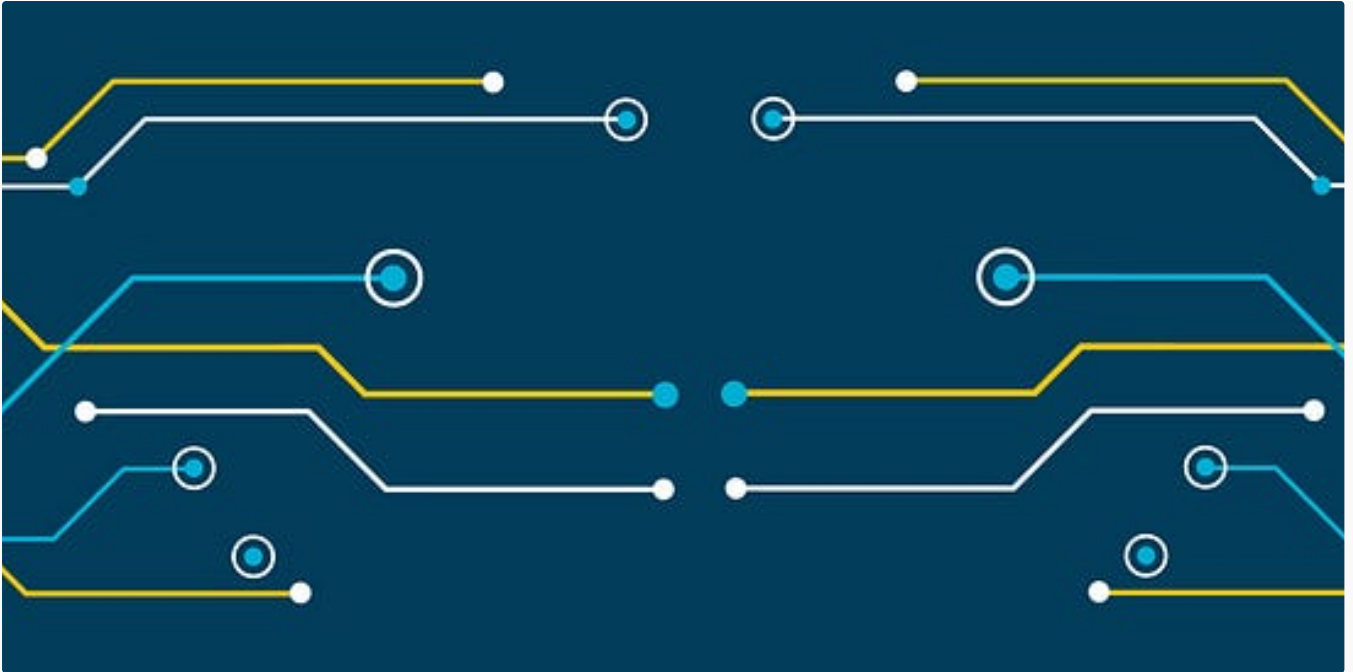
Exploring the Kubernetes Architecture: A Foundation for Modern Application Deployment

In the modern era of cloud computing, Kubernetes has emerged as a crucial tool for orchestrating containerized applications, providing a...

3 min read · Feb 2, 2024

👏 17 💬





Capital One Tech in Capital One Tech

10 microservices design patterns for better architecture

Consider using these popular design patterns in your next microservices app and make organization more manageable.

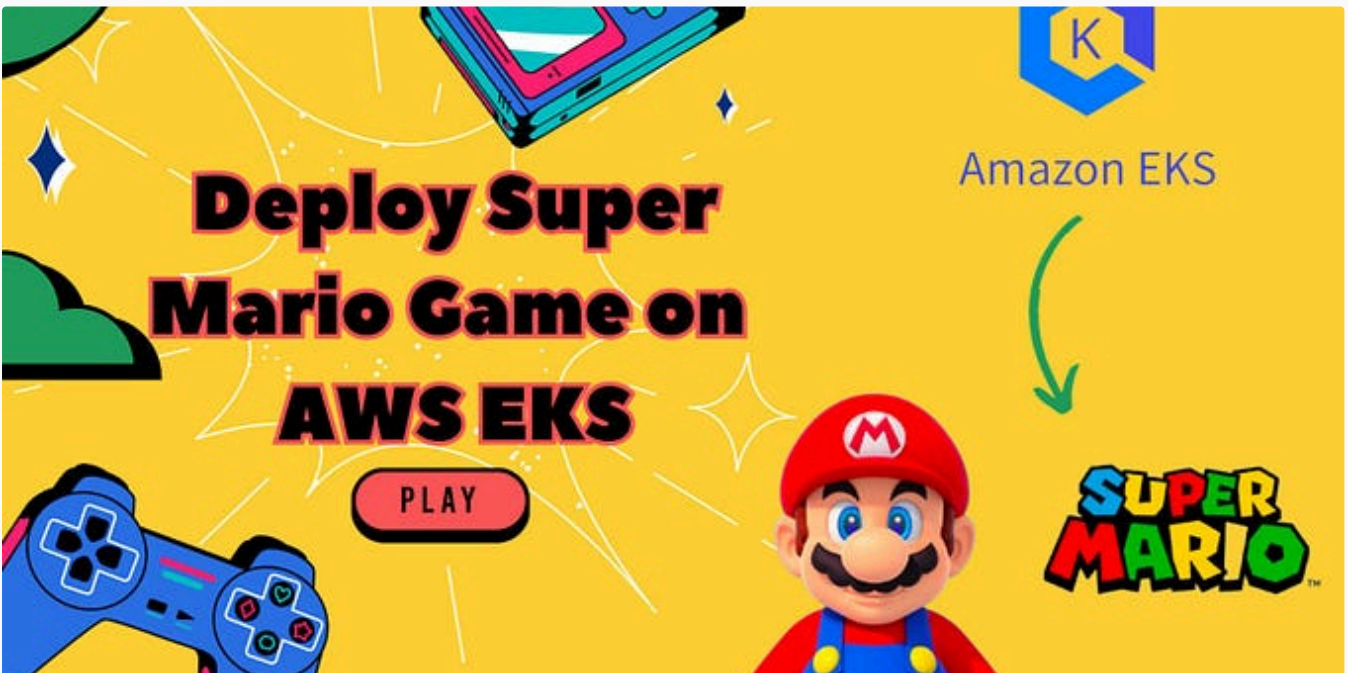
11 min read · Jan 10, 2024



2.6K



16



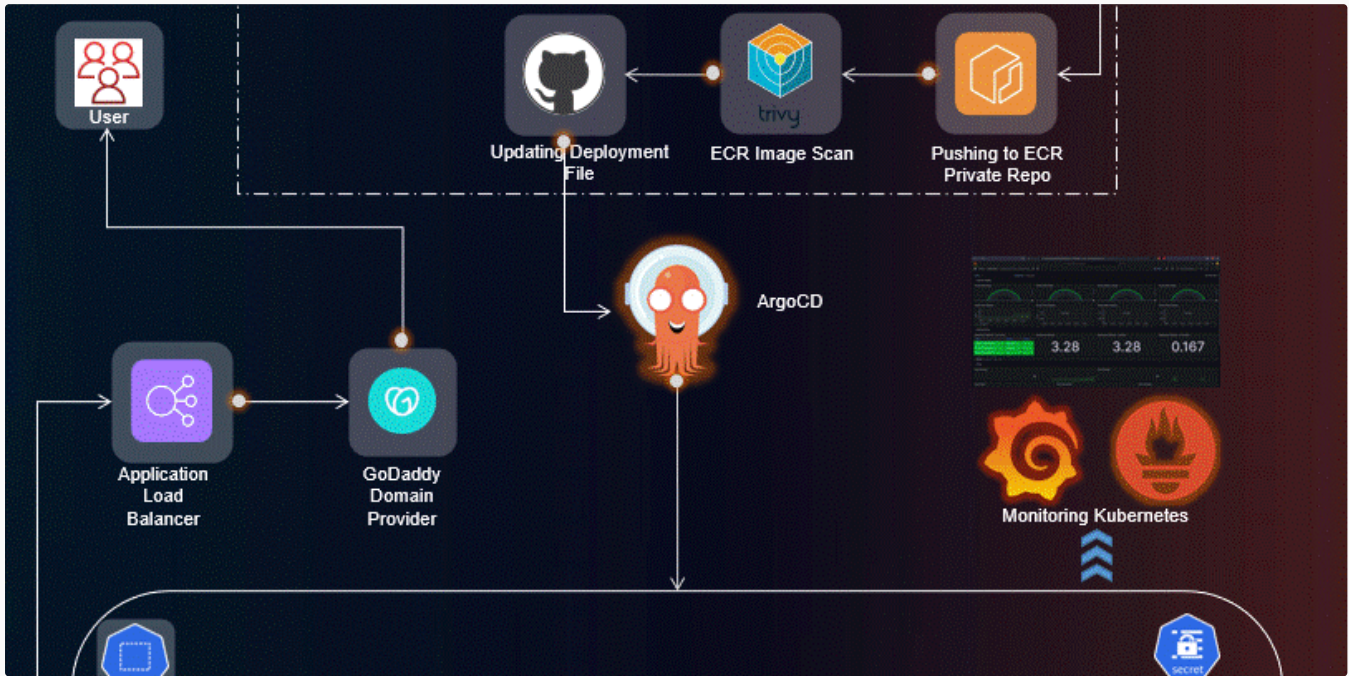
Anurag Kadu

Deploying Super Mario on Kubernetes

Hey folks, remember the thrill of 90's gaming? Let's step back in time and relive those exciting moments! With the game deployed on...

6 min read · Jan 17, 2024

👏 2 💬 1



Aman Pathak in Stackademic

Advanced End-to-End DevSecOps Kubernetes Three-Tier Project using AWS EKS, ArgoCD, Prometheus...

Project Introduction:

24 min read · Jan 18, 2024

👏 1.7K 💬 19



See more recommendations