Open in app ↗

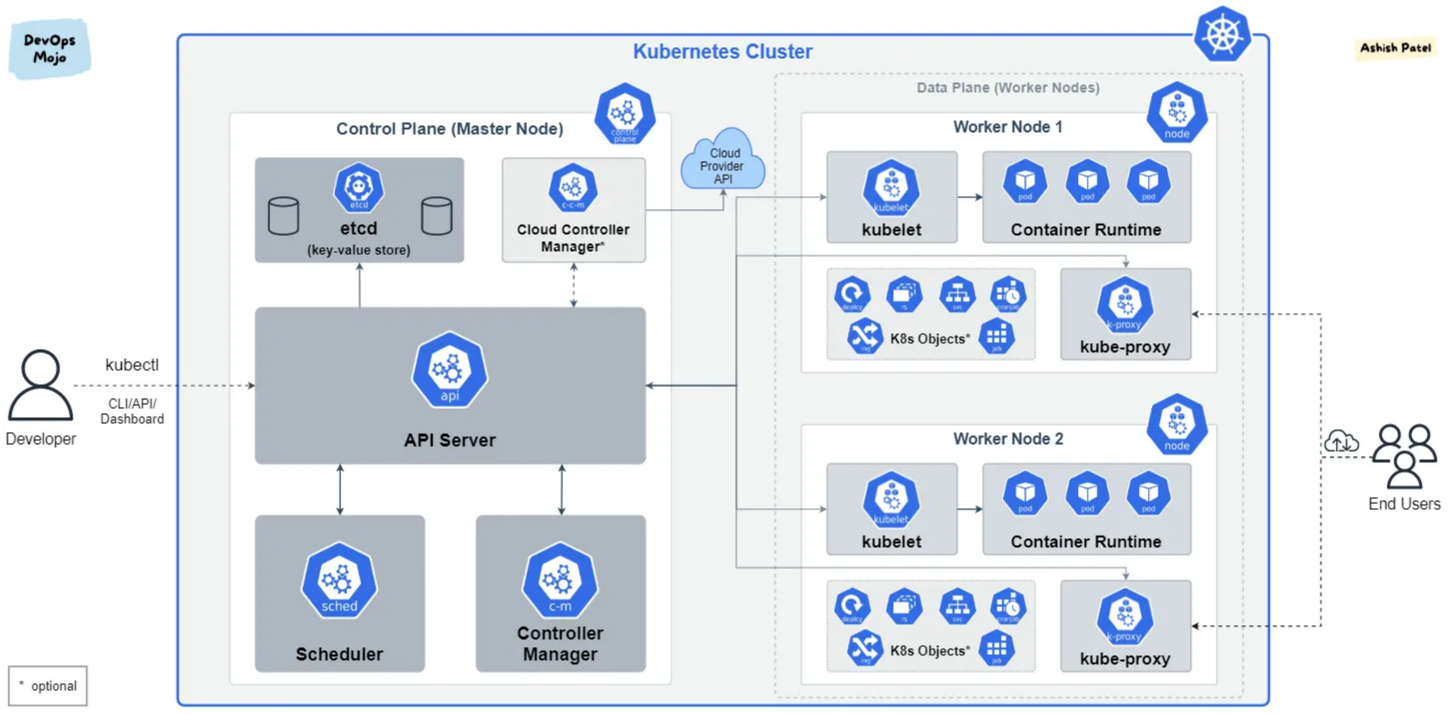
Medium Search 2

Kubernetes — Architecture Overview

Ashish Patel · Follow
Published in DevOps Mojo
5 min read · Aug 12, 2021

Listen Share More

Introduction to Kubernetes Architecture and Understanding K8s Cluster Components.



Kubernetes Architecture and Components

TL;DR

Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications. Kubernetes abstracts away complex container management and provides us with declarative configuration to orchestrate containers in different compute environments.

Kubernetes Architecture

Kubernetes deployment is called a cluster. A Kubernetes cluster consists of at least one main (control) plane, and one or more worker machines, called nodes. Both the control planes and node instances can be physical devices, virtual machines, or instances in the cloud.

Control Plane

- Also known as master node or head node.
- The control plane manages the worker nodes and the Pods in the cluster.
- In production environments, the control plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing fault-tolerance and high availability.
- The control plane receives input from a CLI or UI via an API.
- It is not recommended to run user workloads on master mode.

Node(s)

- Also known as worker node or compute node.
- A virtual or physical machine that contains the services necessary to run containerized applications.
- A Kubernetes cluster needs at least one worker node, but normally have many.
- The worker node(s) host the Pods that are the components of the application workload.
- Pods are scheduled and orchestrated to run on nodes.
- You can scale up and scale down cluster by adding and removing nodes.

Kubernetes Control Plane Components

The control plane's components make global decisions about the cluster, as well as detecting and responding to cluster events. Kubernetes relies on several administrative services running on the control plane. These services manage aspects, such as cluster component communication, workload scheduling, and cluster state persistence.

API Server (kube-apiserver)

- API server exposes the Kubernetes API.
- Entry point for REST/kubectl — It is the front end for the Kubernetes control plane.
- It tracks the state of all cluster components and managing the interaction between them.
- It is designed to scale horizontally.
- It consumes YAML/JSON manifest files.
- It validates and processes the requests made via API.

etcd (key-value store)

- It is a consistent, distributed, and highly-available key value store.
- It is stateful, persistent storage that stores all of Kubernetes cluster data (cluster state and config).
- It is the source of truth for the cluster.
- It can be part of the control plane, or, it can be configured externally.

Scheduler (kube-scheduler)

- It schedules pods to worker nodes.
- It watches api-server for newly created Pods with no assigned node, and selects a

healthy node for them to run on.

- If there are no suitable nodes, the pods are put in a pending state until such a healthy node appears.
- It watches API Server for new work tasks.

Factors taken into account for scheduling decisions include:

- Individual and collective resource requirements.
- Hardware/software/policy constraints.
- Affinity and anti-affinity specifications.
- Data locality.
- Inter-workload interference.
- Deadlines and taints.

Controller Manager (kube-controller-manager)

- It watches the desired state of the objects it manages and watches their current state through the API server.
- It takes corrective steps to make sure that the current state is the same as the desired state.
- It is controller of controllers.
- It runs controller processes. Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process.

Some types of controllers are:

- Node controller: Responsible for noticing and responding when nodes go down.
- Job controller: Watches for Job objects that represent one-off tasks, then creates Pods to run those tasks to completion.

- Endpoints controller: Populates the Endpoints object (that is, joins Services & Pods).
- Service Account & Token controllers: Create default accounts and API access tokens for new namespaces.

Cloud Controller Manager

- The cloud controller manager integrates with the underlying cloud technologies in your cluster when the cluster is running in a cloud environment.
- The cloud-controller-manager only runs controllers that are specific to your cloud provider.
- Cloud controller lets you link your cluster into cloud provider’s API, and separates out the components that interact with that cloud platform from components that only interact with your cluster.

The following controllers can have cloud provider dependencies:

- Node controller: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding.
- Route controller: For setting up routes in the underlying cloud infrastructure.
- Service controller: For creating, updating and deleting cloud provider load balancers.

Kubernetes Node Components

Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.

kubelet

- It is an agent that runs on each node in the cluster.
- It acts as a conduit between the API server and the node.

- It makes sure that containers are running in a Pod and they are healthy.
- It instantiates and executes Pods.
- It watches API Server for work tasks.
- It gets instructions from master and reports back to Masters.

kube-proxy

- It is networking component that plays vital role in networking.
- It manages IP translation and routing.
- It is a network proxy that runs on each node in cluster.
- It maintains network rules on nodes. These network rules allow network communication to Pods from inside or outside of cluster.
- It ensure each Pod gets unique IP address.
- It makes possible that all containers in a pod share a single IP.
- It facilitating Kubernetes networking services and load-balancing across all pods in a service.
- It deals with individual host sub-netting and ensure that the services are available to external parties.

Container runtime

- The container runtime is the software that is responsible for running containers (in Pods).
- To run the containers, each worker node has a container runtime engine.
- It pulls images from a container image registry and starts and stops containers.

Kubernetes supports several container runtimes:

1. Docker

2. containerd
3. CRI-O
4. Any implementation of the Kubernetes CRI (Container Runtime Interface).

What else does a Kubernetes cluster need?

Container Registry

The container images that Kubernetes relies on are stored in a container registry. This can be a registry you configure, or a third party registry like

- Docker Hub
- Amazon Elastic Container Registry (*ECR*)
- Azure Container Registry (*ACR*)
- Google Container Registry (*GCR*)

Persistent Storage

Managing the containers that run an application, Kubernetes can also manage application data attached to a cluster. Kubernetes allows users to request storage resources without having to know the details of the underlying storage infrastructure.

Underlying Infrastructure

One of Kubernetes's key advantages is it works on many different kinds of infrastructure. This can be bare metal servers, virtual machines, public cloud providers, private clouds, and hybrid cloud environments.

View more from *DevOps Mojo*

- Kubernetes Ingress Overview
- Kubernetes Service Types Overview
- Kubernetes Storage Overview — PV, PVC and Storage Class
- Kubernetes Services Overview
- Terraform Remote States Overview

Happy Learning!!!

Kubernetes K8s Kubernetes Architecture Kubernete Components Kubernetes Cluster

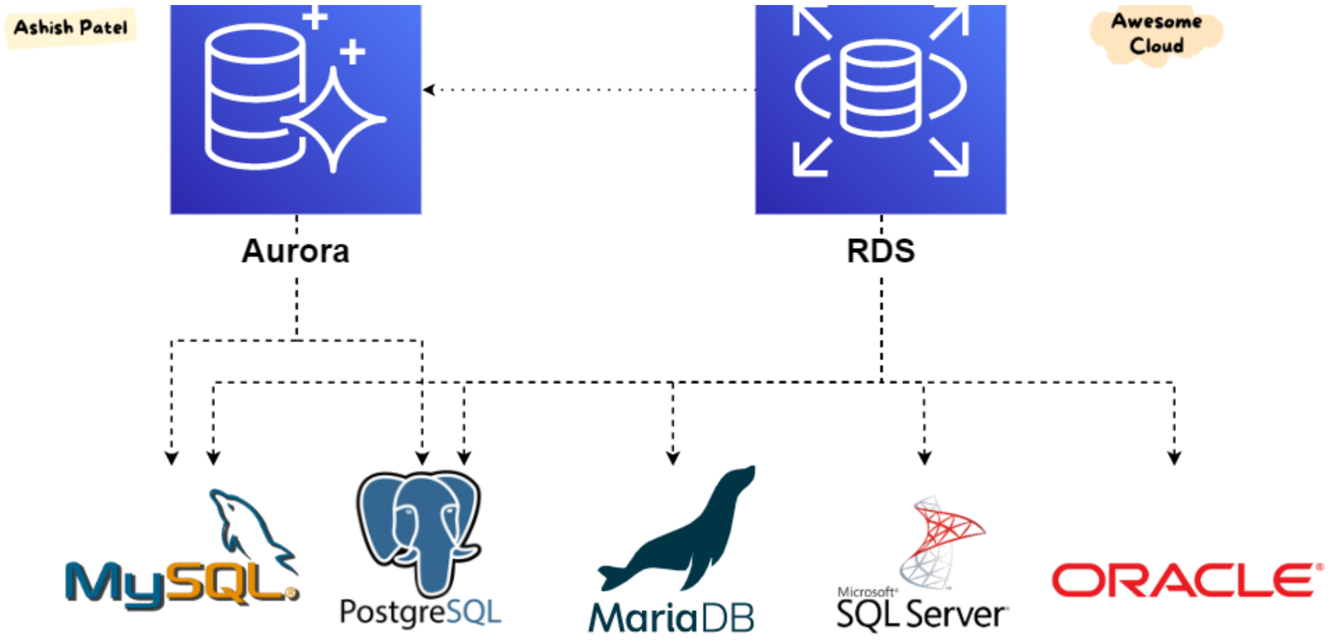
Follow

Written by Ashish Patel

20K Followers · Editor for DevOps Mojo

Cloud Architect • 4x AWS Certified • 6x Azure Certified • 1x Kubernetes Certified • MCP • .NET • Terraform • DevOps • Blogger [https://bit.ly/iamashishpatel]

More from Ashish Patel and DevOps Mojo

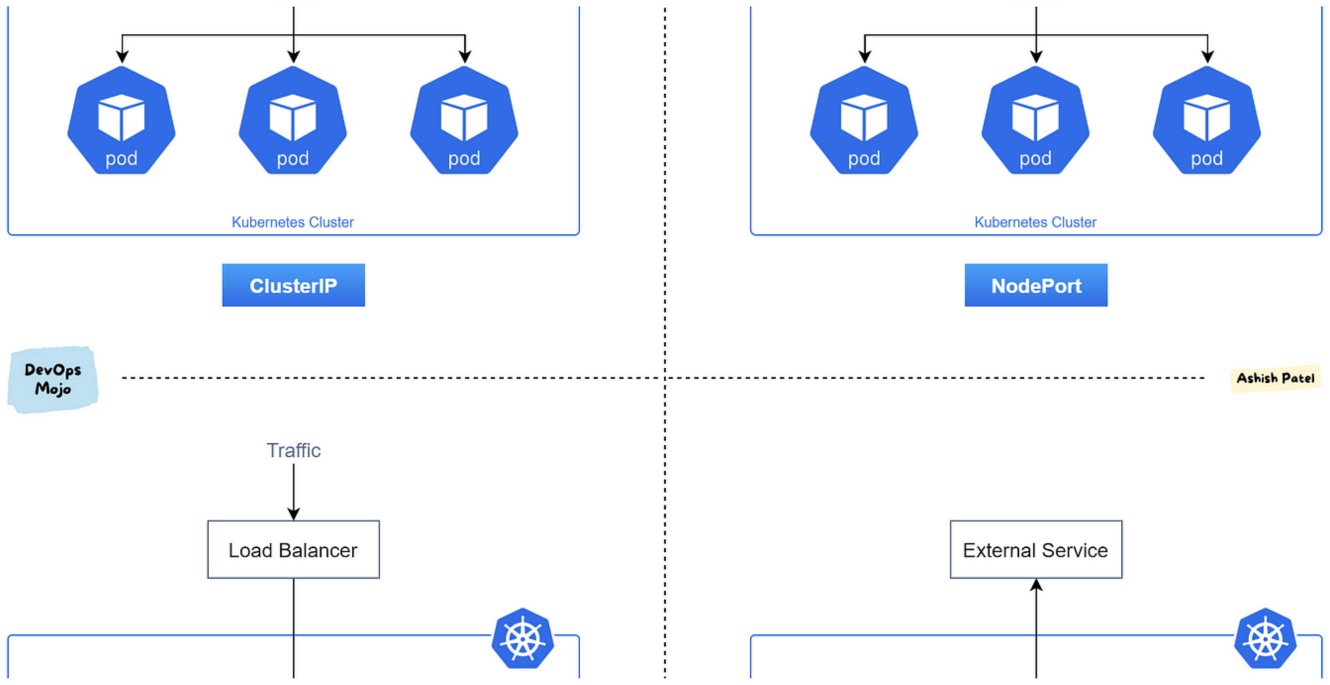


Ashish Patel in Awesome Cloud

AWS—Difference between Amazon Aurora and Amazon RDS

Comparison: Amazon Aurora vs Amazon RDS.

Feb 6, 2022 1.1K 4

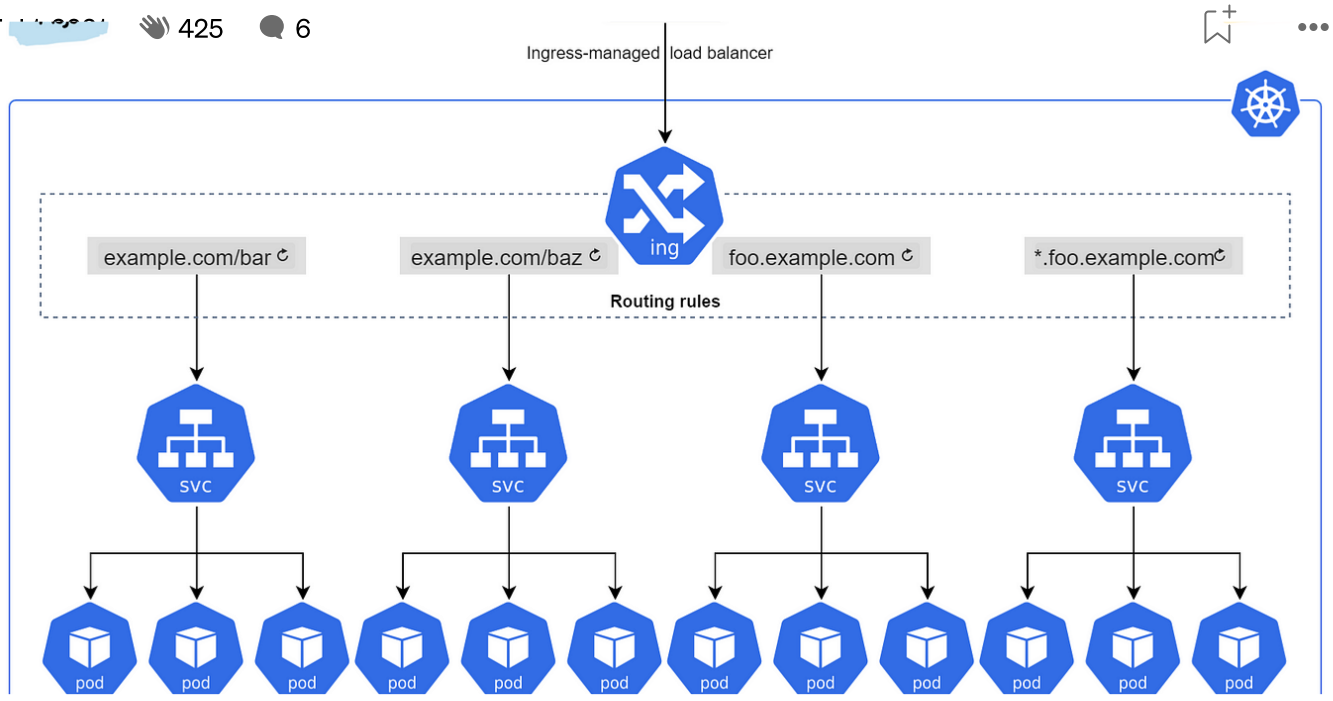


Ashish Patel in DevOps Mojo

Kubernetes—Service Types Overview

Introduction to Service types in K8s — Types of Kubernetes Services.

[illegible] 425 6

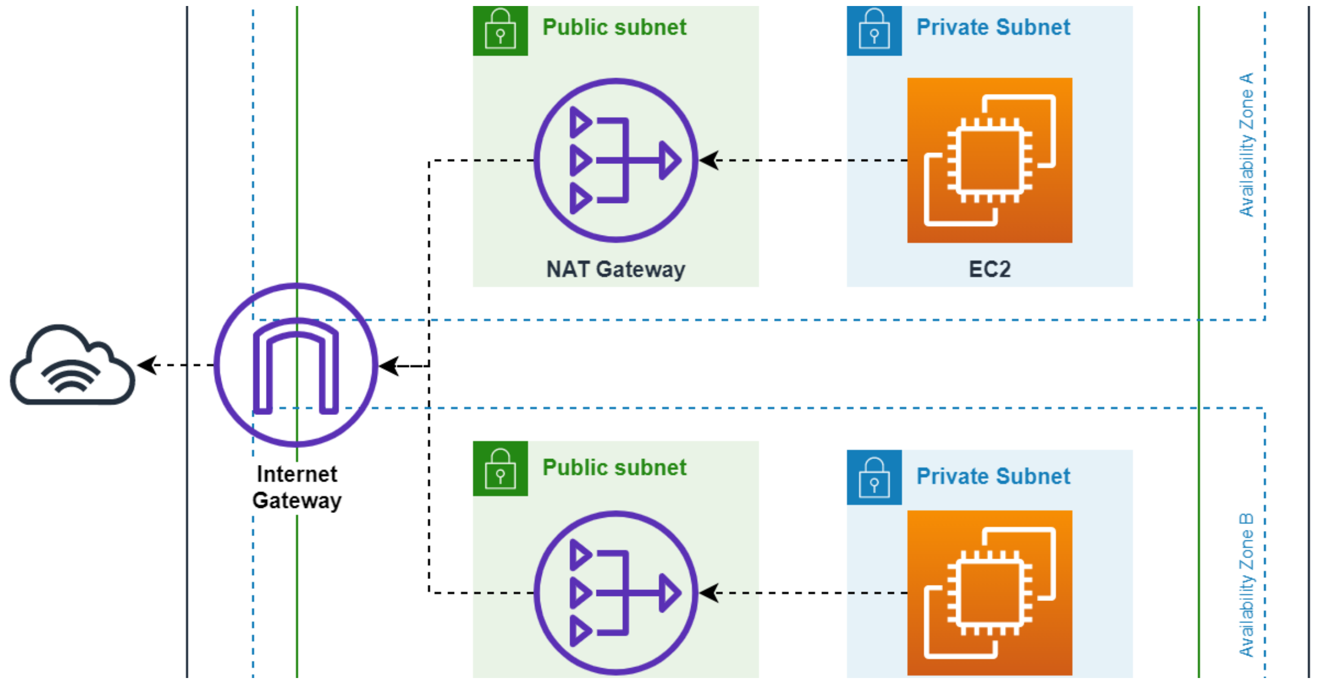


Ashish Patel in DevOps Mojo

Kubernetes—Ingress Overview

What is K8s Ingress?—Introduction to Kubernetes Ingress.

Oct 5, 2021 377



Ashish Patel in Awesome Cloud

AWS—Difference between Internet gateway and NAT gateway

Internet gateway vs NAT gateway in AWS

May 25, 2019 1.2K 12

See all from Ashish Patel

See all from DevOps Mojo

Recommended from Medium

Sheikh Wasiu Al Hasib

Difference Between `matchLabels` and `labels` in Kubernetes

Both `matchLabels` and `labels` play an essential role in managing workloads in Kubernetes, especially in resources like Deployments...

Oct 13

Prince Yadav

How Helm Sub Charts Simplify Enterprise-Level DevOps

In a microservices architecture project, when you are handling deployments as part of your DevOps responsibilities, managing a large number...

Oct 5 3

Lists

Natural Language Processing

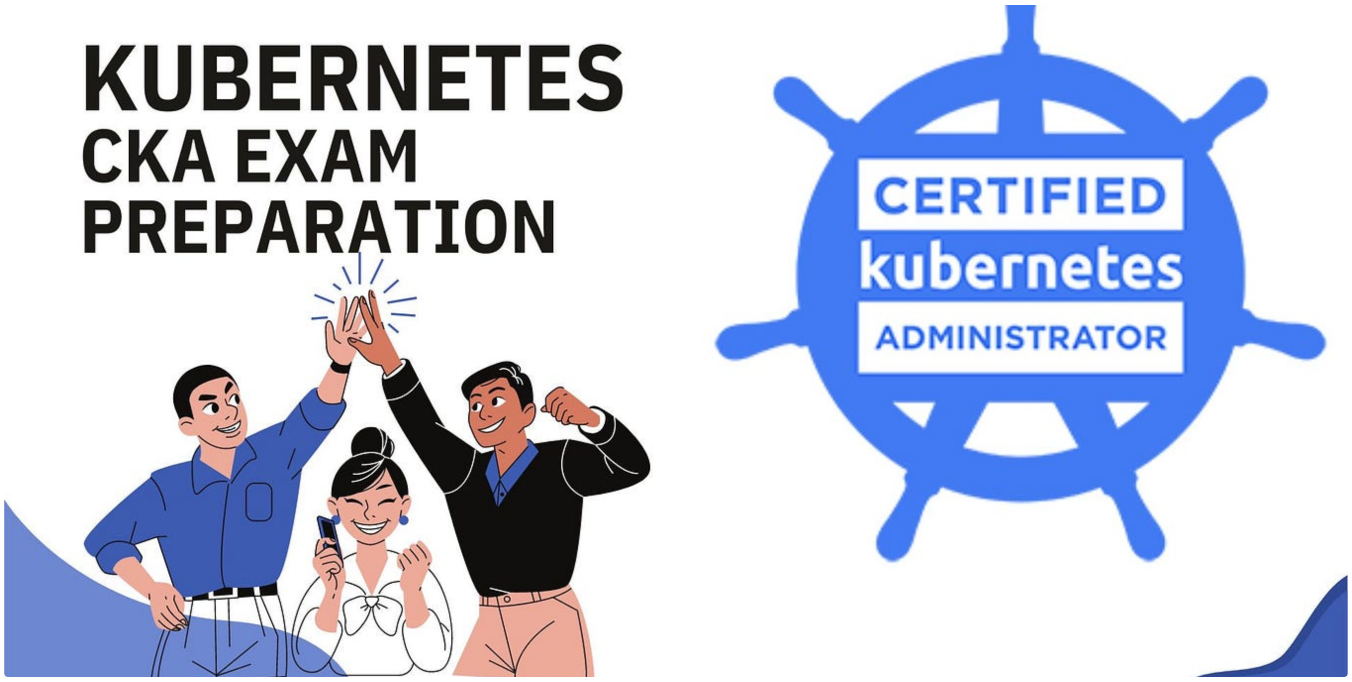
1789 stories · 1400 saves

The Devops Girl in AWS in Plain English

How to Answer Container Troubleshooting Questions in a DevOps Interview

In a DevOps interview, you're likely to be asked about troubleshooting container issues. These questions are designed to test your...

Oct 11 54 1

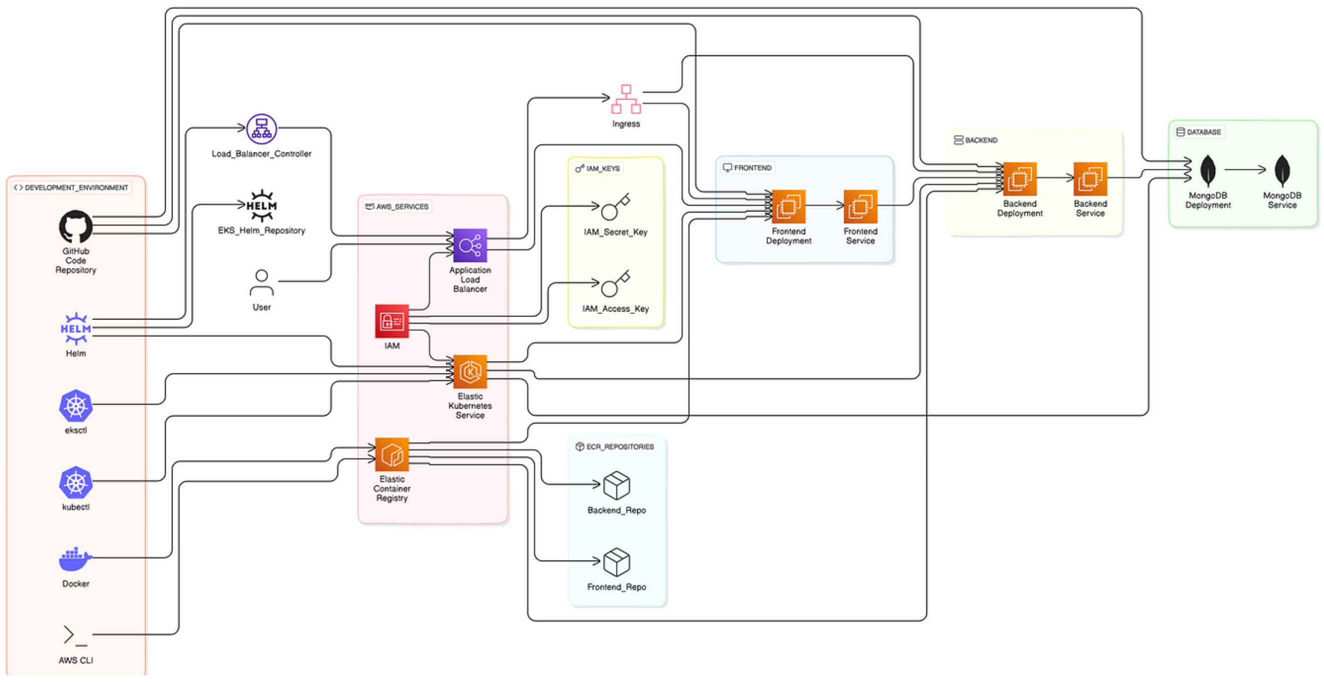


Nidhi Ashtikar

Certified Kubernetes Administrator- Exam Preparation

CKA Exam Preparation

Jun 4 3 1

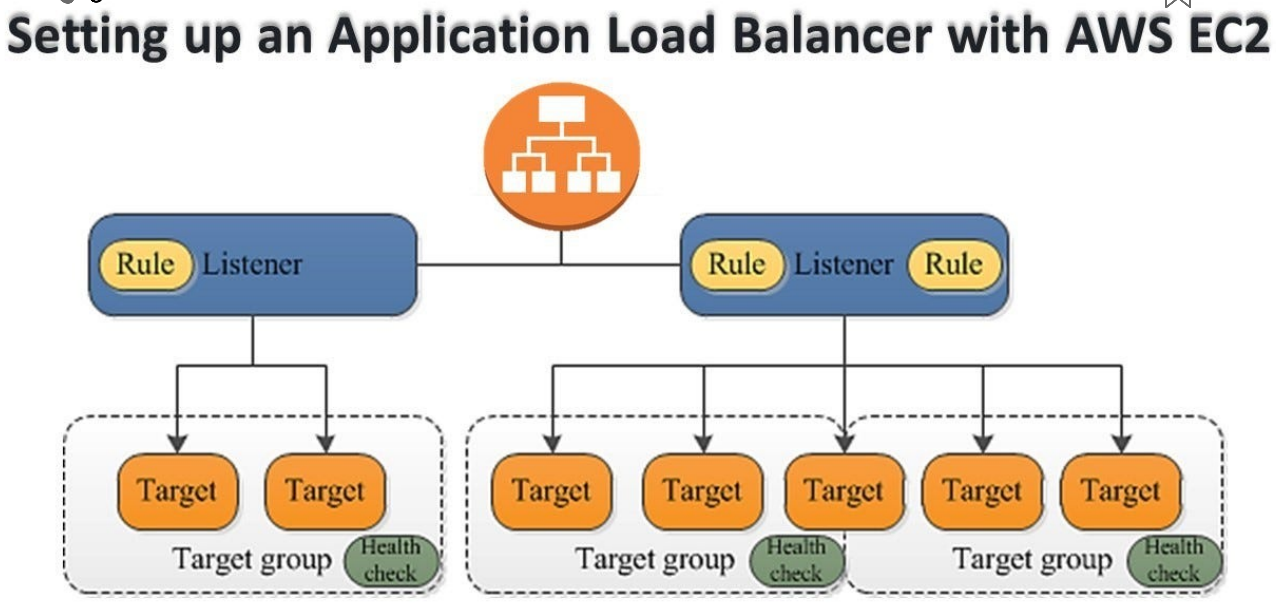


Subham Pradhan

Deployment of a Three-Tier Application on Kubernetes

In this article, I will guide you through the deployment of a three-tier application on Kubernetes. A three-tier architecture separates...

3



@Harsh

Deploying Applications with AWS Application Load Balancer (ALB): A Practical Guide

Introduction

Jun 21 3

See more recommendations