☸️ A Hands-On Guide to Kubernetes RBAC With a User Creation 🛠️

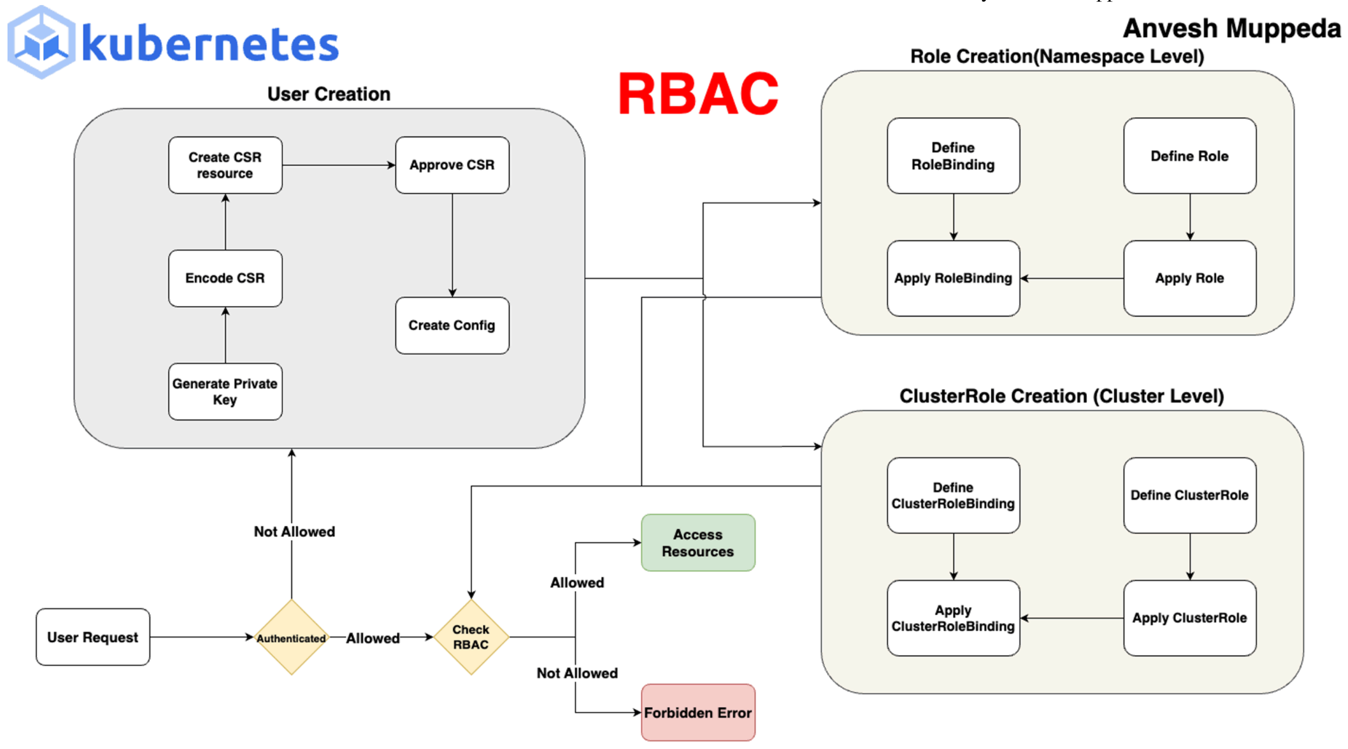
⇢ Understanding RBAC in Kubernetes: A Practical Example

Anvesh Muppeda · Follow
8 min read · May 3, 2024

RBAC

In a Kubernetes cluster, managing access control is crucial for maintaining security and ensuring that only authorized users have access to resources. Kubernetes RBAC (Role-Based Access Control) provides a powerful mechanism for controlling access to Kubernetes resources based on roles and permissions.

In this article, we'll explore Kubernetes RBAC in detail, covering its components, how to define roles and permissions, best practices, and advanced examples.



RBAC Flowchart

Overview of RBAC in Kubernetes

RBAC in Kubernetes allows you to define roles and permissions for users, groups, or service accounts. It consists of four main components:

- **Roles**: Define a set of permissions within a namespace.
- **RoleBindings**: Associate roles with users, groups, or service accounts within a namespace.
- **ClusterRoles**: Similar to roles, but apply across the entire cluster.
- **ClusterRoleBindings:** Associate cluster roles with users, groups, or service accounts across the entire cluster.

Understand Roles, RoleBindings, ClusterRoles, and ClusterRoleBindings

Roles & Role Bindings:

Let's say we have a Kubernetes namespace called "development" and we want to grant permissions to a user named "developer" to manage pods within that namespace.

Role Definition (YAML):

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: development
  name: pod-manager
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list", "create", "delete"]
```

This role (`pod-manager`) allows the `developer` to get, list, create, and delete pods within the `development` namespace.

Role Binding (YAML):

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: pod-manager-binding
  namespace: development
subjects:
- kind: User
  name: developer
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-manager
  apiGroup: rbac.authorization.k8s.io
```

This role binding (`pod-manager-binding`) binds the `pod-manager` role to the `developer` user within the `development` namespace.

Cluster Roles & Cluster Role Bindings:

Let’s consider a scenario where we want to grant permissions to view nodes across the entire cluster to a group named “ops-team”.

Cluster Role Definition (YAML):

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: node-viewer
rules:
- apiGroups: [""]
  resources: ["nodes"]
  verbs: ["get", "list"]
```

This cluster role (`node-viewer`) allows the `ops-team` group to get and list nodes across the entire cluster.

Cluster Role Binding (YAML):

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: node-viewer-binding
subjects:
- kind: Group
  name: ops-team
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: node-viewer
  apiGroup: rbac.authorization.k8s.io
```

This cluster role binding (`node-viewer-binding`) binds the `node-viewer` cluster role to the `ops-team` group, allowing them to view nodes across the entire cluster.

In summary, **roles** & **role bindings** are used to manage permissions within namespaces, while **cluster roles** & **cluster role bindings** are used to manage permissions across the entire cluster.

Create a new user in Kubernetes Cluster

To test RBAC we need to a user, so first create a new user. Below is the step-by-step guide to creating a new user named Anvesh in a Kubernetes cluster

Step 1: Generate Certificates for the User

Generate a private key for Anvesh using RSA algorithm (4096 bits):

```
openssl genrsa -out anvesh.pem
```

This command will generate an RSA private key.

Create a Certificate Signing Request (CSR) for Anvesh:

```
openssl req -new -key anvesh.pem -out anvesh.csr -subj "/CN=anvesh"
```

Step 2: Create a Certificate Signing Request (CSR)

1. Obtain the base64-encoded representation of the CSR:

```
cat anvesh.csr | base64 | tr -d "\n"
```

Here we encode the CSR to be used in the **CertificateSigningRequest.**

2. Use the output to create a `CertificateSigningRequest` resource:

```
cat <<EOF | kubectl apply -f -
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: anvesh
spec:
  request: <base64_encoded_csr>
  signerName: kubernetes.io/kube-apiserver-client
  expirationSeconds: 86400  # one day
  usages:
  - digital signature
  - key encipherment
  - client auth
EOF
```

CSR status

```
$ kubectl get csr
NAME     AGE   SIGNERNAME                            REQUESTOR
anvesh   9s    kubernetes.io/kube-apiserver-client   anveshmuppeda@gmail.com
```

Step 3: Sign the Certificate Using the Cluster Certificate Authority

1. Approve the CertificateSigningRequest for Anvesh:

```
$ kubectl certificate approve anvesh
```

```
$ kubectl certificate approve anvesh

certificatesigningrequest.certificates.k8s.io/anvesh approved
```

2. Check the CSR status:

```
$ kubectl describe csr/anvesh
```

```
$ kubectl get csr
NAME     AGE   SIGNERNAME                            REQUESTOR
anvesh   31s   kubernetes.io/kube-apiserver-client   anveshmuppeda@gmail.com
```

Step 4: Create a Configuration Specific to the User

1. **Extract the signed certificate from the CertificateSigningRequest:**

```
kubectl get csr/anvesh -o jsonpath="{.status.certificate}" | base64 -d > anvesh
```

2. Create new user config file:

Use the `kubectl config set-cluster` command to set up the cluster information:

```
kubectl --kubeconfig ~/.kube/config-anvesh config set-cluster preprod --insecur
```

Replace `KUBERNETES-API-ADDRESS` with the actual API server address of your Kubernetes cluster.

3. Set user credentials:

Use the `kubectl config set-credentials` command to set up the user credentials:

```
kubectl --kubeconfig ~/.kube/config-anvesh config set-credentials anvesh --clie
```

Replace `anvesh-user.crt` and `anvesh.pem` with the paths to your user certificate and private key files respectively.

4. Set context information:

Use the `kubectl config set-context` command to set up the context information:

```
kubectl --kubeconfig ~/.kube/config-anvesh config set-context default --cluster
```

5. Use the context:

Finally, use the `kubectl config use-context` command to use the newly created context:

```
kubectl --kubeconfig ~/.kube/config-anvesh config use-context default
```

Now, your `config-anvesh` file is configured with the necessary cluster, user, and context information. You can use it with `kubectl` commands by passing `--kubeconfig ~/.kube/config-anvesh`. Make sure to replace placeholder values with your actual configuration details.

Example:

```
kubectl --kubeconfig ~/.kube/config-anvesh get pods
```

We've successfully created a new user named "anvesh". However, when we try to access the pods using this user, we encounter a **Forbidden** error:

```
$ kubectl --kubeconfig ~/.kube/config-anvesh get pods
Error from server (Forbidden): pods is forbidden: User "anvesh" cannot list res
```

This error occurs because we haven't assigned any permissions to the "**anvesh**" user yet; we've only created the user.

Now, let's proceed to create new roles and role bindings and associate them with the "**anvesh**" user using the following steps.

Writing RBAC Rules

Let's create RBAC rules and apply to the above user:

Example 1: Allow read-only access to pods in specific namespace

1. **Role YAML:**

Create a file named `pod-reader-role.yaml` with the following content:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: kube-system
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list"]
```

Apply the Role using:

```
kubectl apply -f pod-reader-role.yaml
```

This Role allows getting and listing pods in the `kube-system` namespace.

2. RoleBinding YAML:

Create a file named `anvesh-pod-reader-rolebinding.yaml` with the following content:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: anvesh-pod-reader
  namespace: kube-system
subjects:
- kind: User
  name: anvesh
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

Apply the RoleBinding using:

```
kubectl apply -f anvesh-pod-reader-rolebinding.yaml
```

This RoleBinding binds the “ `anvesh` ” user to the “ `pod-reader` ” Role in the `kube-system` namespace.

Now, the “ `anvesh` ” user should be able to list and get pods in the **kube-system** namespace using the provided `kubeconfig` i.e., `~/.kube/config-anvesh` .

We’ve set up a new role and role binding for the user “ `anvesh` ”. Now, let’s try accessing the pods in the `kube-system` namespace with the following command.

```
$ kubectl --kubeconfig ~/.kube/config-anvesh get pods -n kube-system
```

Output:

```
$ kubectl --kubeconfig ~/.kube/config-anvesh get pods -n kube-system
NAME                                 READY   STATUS    RESTARTS       AGE
cilium-operator-66846d4dd8-x2fpv     1/1     Running   2 (2d19h ago)  10d
cilium-whkvz                         1/1     Running   0              11d
```

We’ve successfully listed the pods in the kube-system namespace. However, if you attempt to access other resources and resources outside of the `kube-system` namespace, you’ll receive a Forbidden error.

```
$ kubectl --kubeconfig ~/.kube/config-anvesh get cm -n kube-system
```

```
Error from server (Forbidden): configmaps is forbidden: User "anvesh" cannot li
```

Example 2: Allow read-only access to pods in all namespace

You can define a `ClusterRole` and `ClusterRoleBinding` in YAML format using below steps to allow the “ `anvesh` ” user to list pods across all namespaces.

1. **ClusterRole YAML:**

Create a file named `pod-list-clusterrole.yaml` with the following content:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pod-lister
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list"]
```

Apply the **ClusterRole** using:

```
kubectl apply -f pod-list-clusterrole.yaml
```

This ClusterRole allows getting and listing pods across all namespaces.

2. ClusterRoleBinding YAML:

Create a file named `anvesh-pod-lister-clusterrolebinding.yaml` with the following content:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: anvesh-pod-lister
subjects:
- kind: User
  name: anvesh
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: pod-lister
  apiGroup: rbac.authorization.k8s.io
```

Apply the **ClusterRoleBinding** using:

```
kubectl apply -f anvesh-pod-lister-clusterrolebinding.yaml
```

This ClusterRoleBinding binds the “anvesh” user to the “pod-lister” ClusterRole, allowing the user to list pods across all namespaces.

Now, the “anvesh” user should be able to list pods across all namespaces using the provided `kubeconfig` i.e., `~/.kube/config-anvesh` .

We’ve set up a new **clusterRole** and **clusterRoleBinding** for the user “ `anvesh` ”. Now, let’s try accessing the pods in all namespaces with the following command.

```
kubectl --kubeconfig ~/.kube/config-anvesh get po -A
```

Output:

```
$ kubectl --kubeconfig ~/.kube/config-anvesh get pods -A
NAMESPACE        NAME                                          READY   STATUS
ingress-nginx    ingress-nginx-admission-create-6mbxh          0/1     Completed
ingress-nginx    ingress-nginx-admission-patch-4p82d           0/1     Completed
ingress-nginx    ingress-nginx-controller-7dcdbcff84-kqc2n     1/1     Running
kube-system      cilium-operator-66846d4dd8-x2fpv              1/1     Running
```

```
  kube-system      cilium-whkvz                         1/1     Running
  kube-system      coredns-6857f5494d-lq2c8             1/1     Running
  kube-system      coredns-6857f5494d-xt7rw             1/1     Running
  kube-system      cpc-bridge-proxy-hdzn6               1/1     Running
  kube-system      csi-do-node-v52xs                    2/2     Running
  kube-system      do-node-agent-25w77                  1/1     Running
  kube-system      hubble-relay-778856d87d-hkbjf        1/1     Running
  kube-system      hubble-ui-776986f894-l9qnm           2/2     Running
  kube-system      konnectivity-agent-lhq85             1/1     Running
  kube-system      kube-proxy-4bk9h                     1/1     Running
```

We’ve successfully listed the pods in all namespaces. However, if you attempt to access other resources(other than pods) in any namespace, you’ll receive a Forbidden error.

```
$ kubectl --kubeconfig ~/.kube/config-anvesh get cm -A
Error from server (Forbidden): configmaps is forbidden: User "anvesh" cannot li
```

We have successfully created a user in the Kubernetes cluster and granted access at both the namespace and cluster levels using roles, role bindings, cluster roles, and cluster role bindings.

Source Code

You’re invited to explore our GitHub repository, which houses a comprehensive collection of source code for Kubernetes.

GitHub - anveshmuppeda/kubernetes: Kuberntes Complete Notes

Kuberntes Complete Notes. Contribute to anveshmuppeda/kubernetes development by creating an account…

github.com

Also, if we welcome your feedback and suggestions! If you encounter any issues or have ideas for improvements, please open an issue on our GitHub repository. 🚀

Support Me

Connect With Me

If you found this blog insightful and are eager to delve deeper into topics like AWS, cloud strategies, Kubernetes, or anything related, I'm excited to connect with you on LinkedIn. Let's spark meaningful conversations, share insights, and explore the vast realm of cloud computing together.

Feel free to reach out, share your thoughts, or ask any questions. I look forward to connecting and growing together in this dynamic field!

Happy deploying! 🚀

Happy Kubernetings! ☸

Kubernetes | Rbac Access Control | K8s | Kubernetes Administration | Access Control

Follow

Written by Anvesh Muppeda

681 Followers · 9 Following

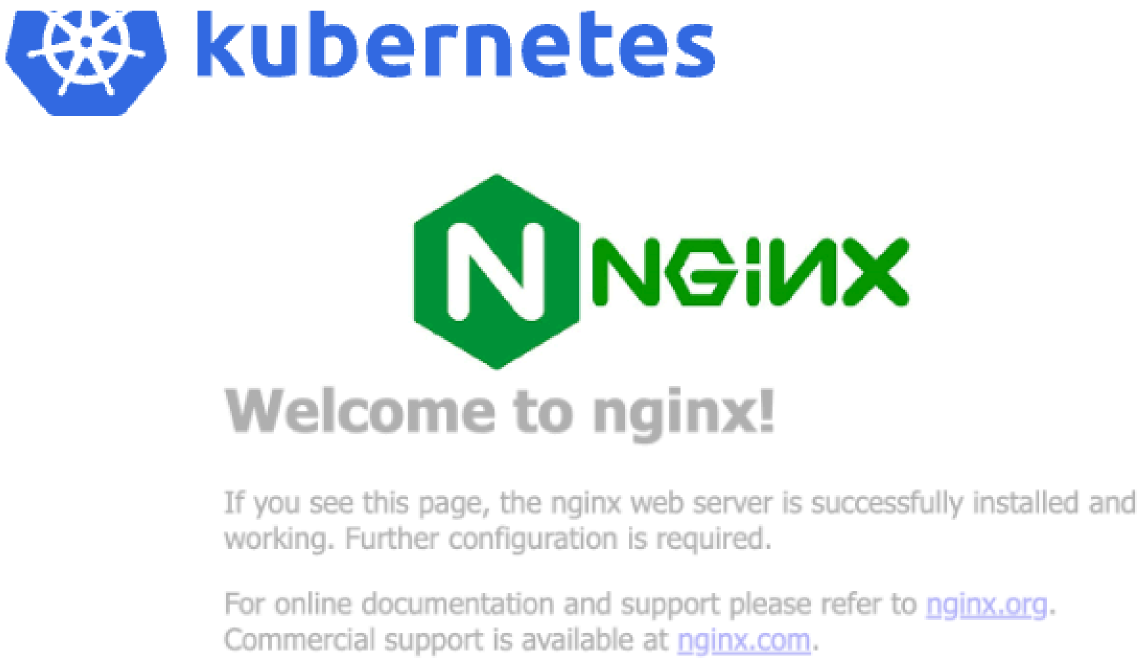
🤝 Cloud Architect & DevOps Engineer || Kubernetes ☸ & Docker 🚢 aficionado || CKA || CKAD || AWS SAA || Connect with me on www.linkedin.com/in/anveshmuppeda

No responses yet

What are your thoughts?

Respond

More from Anvesh Muppeda



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

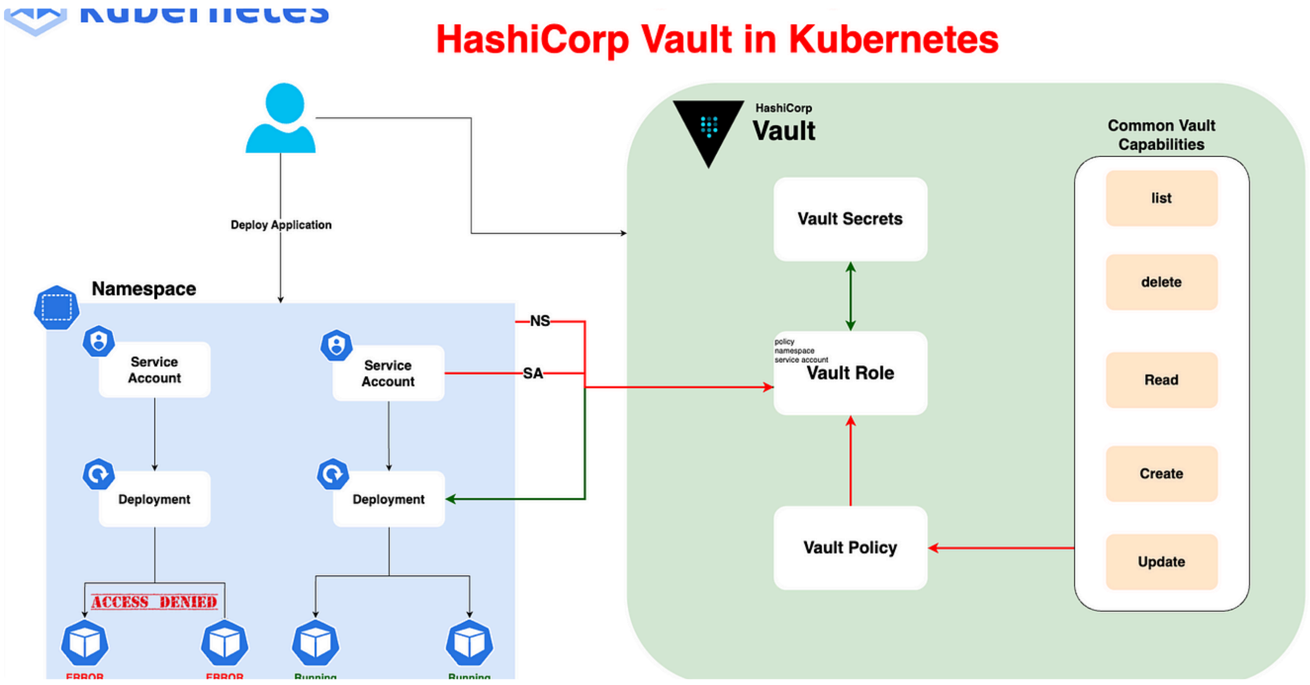
For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Anvesh Muppeda

Deploying Nginx on Kubernetes: Exploring Various Methods

Mastering Nginx Deployment in Kubernetes: A Comprehensive Guide to Pods, Deployments, and Beyond

Jan 13, 2024 152 2



Anvesh Muppeda

☸ A Hands-On Guide to Vault in Kubernetes ⚙️

⇢ Manage k8s Secrets Using HashiCorp Vault: With Practical Examples

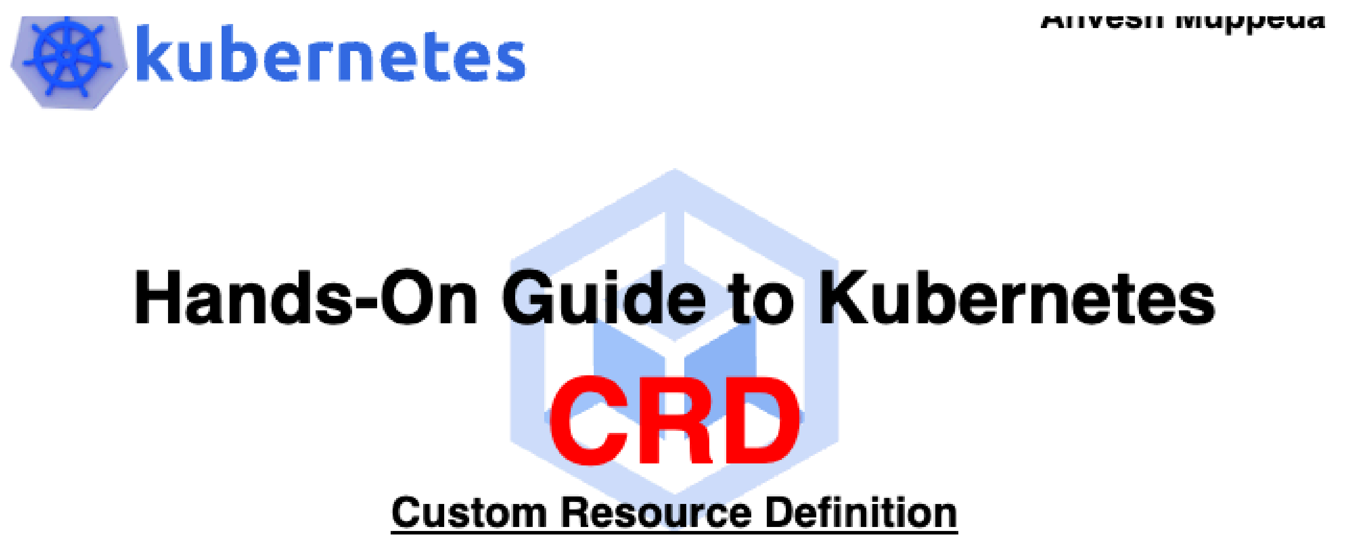
May 22, 2024 30 3

Anvesh Muppeda

🐤 Implementing Canary Deployment in Kubernetes ☸

⇢ Understanding Canary Deployment in Kubernetes with a Practical Example — PART-6

Apr 25, 2024 👏 42 💬 1



Anvesh Muppeda

☸ A Hands-On Guide to Kubernetes Custom Resource Definitions (CRDs) With a Practical Example 🛠

⇢ Understanding Custom Resource Definitions (CRDs) in Kubernetes: A Practical Example

Apr 23, 2024 👏 94 💬 1

See all from Anvesh Muppeda

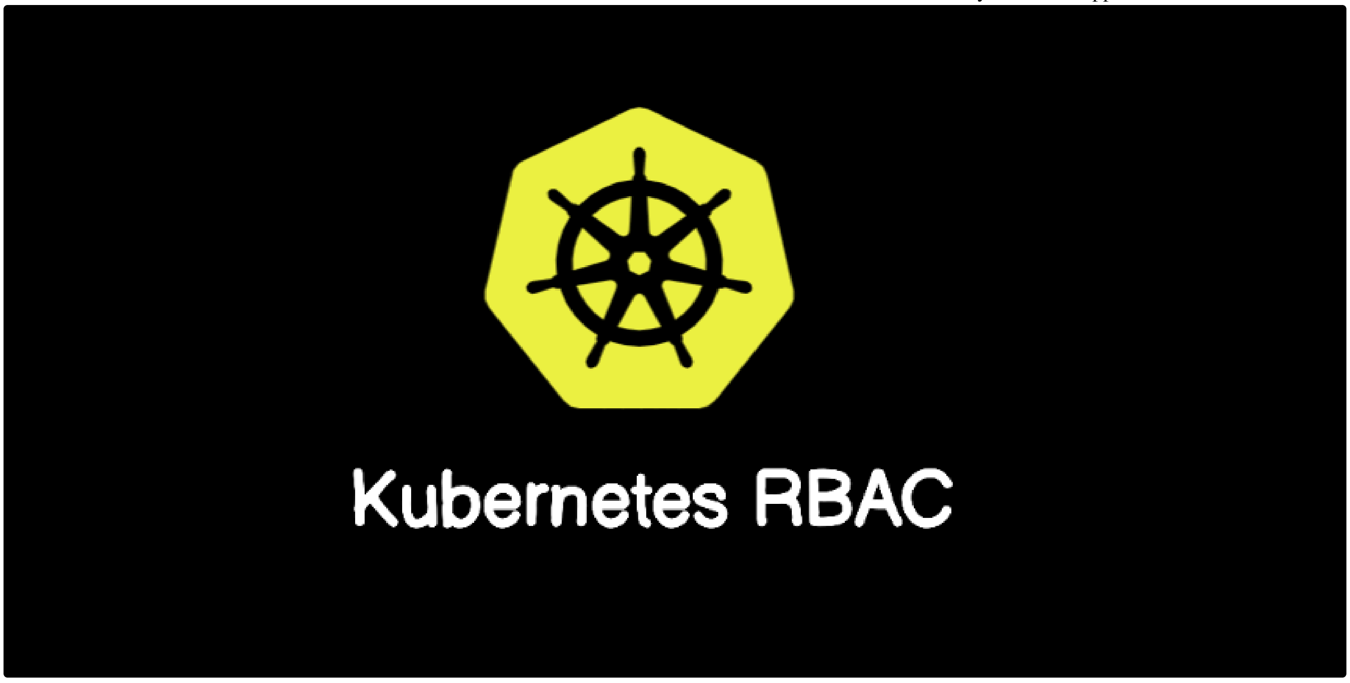
Recommended from Medium

The Devops Girl

How to Update Your Amazon EKS Cluster from 1.27 to 1.28: A Step-by-Step Guide

If you're managing an Amazon EKS (Elastic Kubernetes Service) cluster, keeping your Kubernetes version up to date is crucial for...

Aug 19, 2024 👏 110



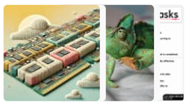
Subham Pradhan

Configuring Kubernetes RBAC: A Comprehensive Guide

Introduction:

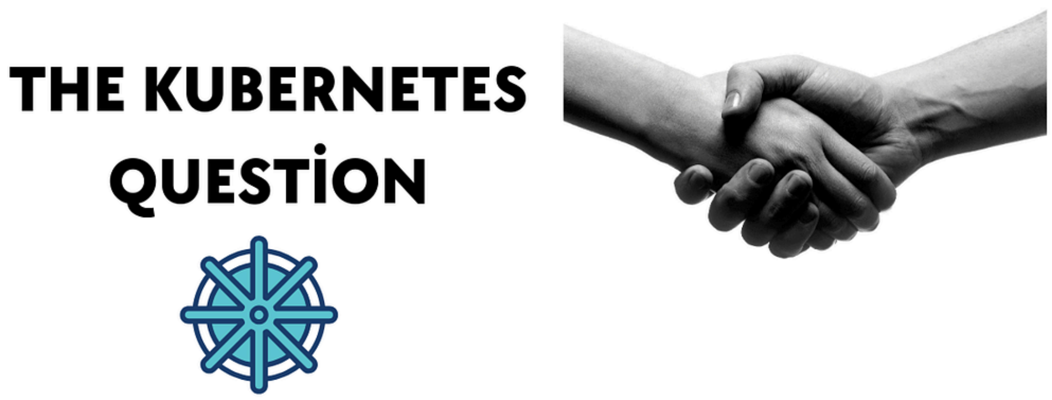
Aug 10, 2024 3 1

Lists



Natural Language Processing

1893 stories · 1553 saves

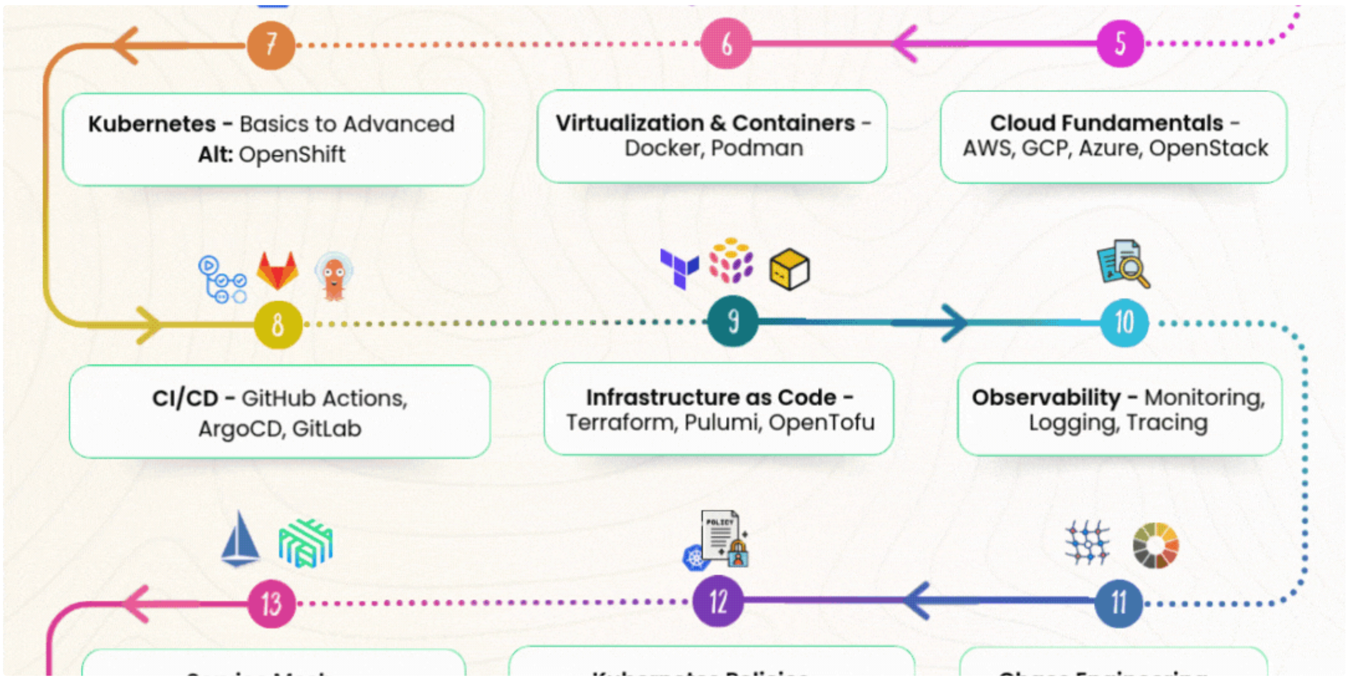


In Level Up Coding by Rahul Sharma

I Asked This Kubernetes Question in Every Interview — And Here's the Catch

When I interview candidates, I prefer a real-world problem that demonstrates the candidate's practical expertise with Kubernetes to be...

Jan 15 315 11

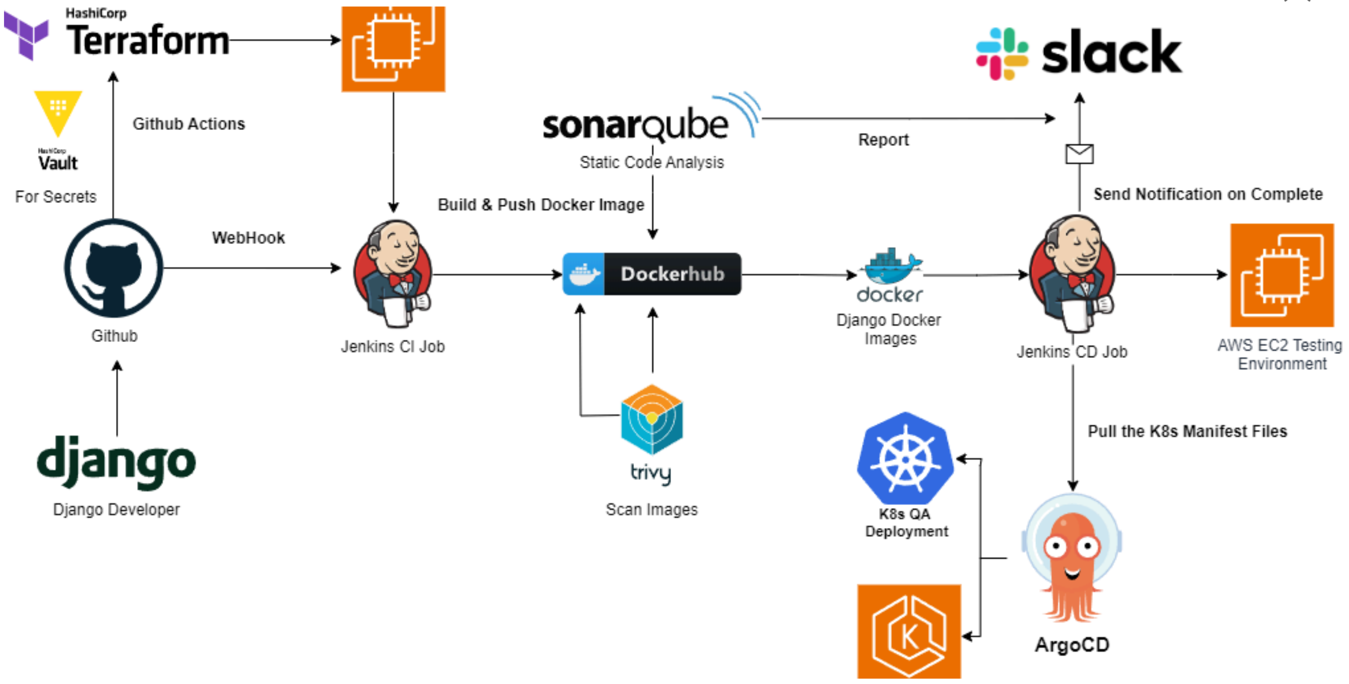

7
6
5
Kubernetes - Basics to Advanced
Alt: OpenShift
Virtualization & Containers - Docker, Podman
Cloud Fundamentals - AWS, GCP, Azure, OpenStack
8
9
10
CI/CD - GitHub Actions, ArgoCD, GitLab
Infrastructure as Code - Terraform, Pulumi, OpenTofu
Observability - Monitoring, Logging, Tracing
13
12
11

Rohit Ghumare

DevOps Roadmap 2025

In today's rapidly evolving tech landscape, DevOps has become more than just a buzzword—it's a crucial methodology that bridges the gap...

Jan 15 175 4

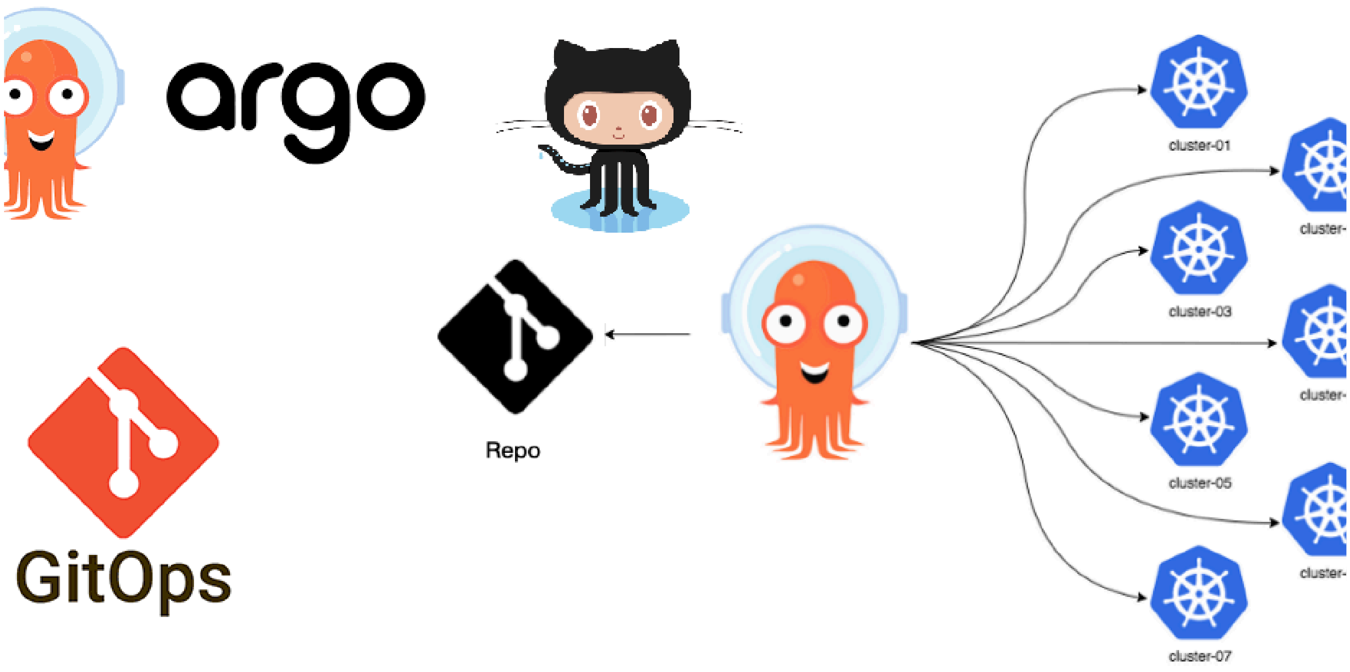


In Django Unleashed by Joel Wembo

Technical Guide: End-to-End CI/CD DevOps with Jenkins, Docker, Kubernetes, ArgoCD, Github Actions ...

Building an end-to-end CI/CD pipeline for Django applications using Jenkins, Docker, Kubernetes, ArgoCD, AWS EKS, AWS EC2

Apr 12, 2024 1K 21



Mr.PlanB

How to Structure GitOps Repositories for ArgoCD and Kubernetes Applications

In the world of GitOps, managing Kubernetes resources and organizing application deployments effectively can make all the difference in…

Oct 12, 2024 34

See more recommendations