Demystifying Kubernetes ConfigMaps and Secrets for Smooth Deployments 🚀: Day 35 of 90DaysOfDevOps

Unlocking Kubernetes ConfigMaps and Secrets: Your Guide to Seamless Deployments

Ajit Fawade · Follow

7 min read · Oct 12, 2023

Listen Share More

ConfigMaps and Secrets play a pivotal role in Kubernetes for managing configuration data and safeguarding sensitive information. Let's delve into these critical components, understand how to use them, and explore their real-world significance.

What are ConfigMaps and Secrets?

ConfigMaps 📁

A ConfigMap is a Kubernetes object that stores configuration data as key-value pairs. A ConfigMap allows you to decouple the configuration data from the container image, making it easier to update and reuse. You can use a ConfigMap to store environment variables, configuration files, command-line arguments, or any other data that you want to pass to your containers.

> *Think of ConfigMaps as neatly labeled folders within a file cabinet. Each folder contains instructions or settings that help different parts of your spaceship (Kubernetes cluster) function efficiently.*

To create a ConfigMap for your deployment, you need to define the key-value pairs that you want to store in the ConfigMap. You can do this using a file or the command line.

- To create a ConfigMap using a file, you need to create a YAML or JSON file that contains the key-value pairs. For example, you can create a file named config.yml with the following content:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-config
data:
  MYSQL_HOST: mysql
  MYSQL_USER: root
  MYSQL_DATABASE: mydb
```

- To create a ConfigMap using the command line, you need to use the `kubectl create configmap` command with the `--from-literal` option. For example, you can run the following command:

```
kubectl create configmap mysql-config --from-literal=MYSQL_HOST=mysql --from-lit
```

Secrets 🔐

A Secret is a Kubernetes object that stores sensitive data in an encrypted form. A Secret allows you to protect your data from unauthorized access or exposure. You can use a Secret to store passwords, tokens, keys, certificates, or any other data that you want to keep secret.

> *Imagine Secrets as a highly secured safe where you store your most valuable possessions. They are encrypted and protected, allowing only authorized personnel (containers) with the right clearance to access them.*

To create a Secret for your deployment, you need to encode the data that you want to store in the Secret using base64 encoding. You can do this using a file or the command line.

- To create a Secret using a file, you need to create a YAML or JSON file that contains the encoded data. For example, you can create a file named secret.yml with the following content:

```
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret
type: Opaque
data:
  MYSQL_PASSWORD: YWRtaW4K # This is the base64 encoded value of "admin".
```

- To create a Secret using the command line, you need to use the kubectl create secret generic command with the — from-literal option. For example, you can run the following command:

```
kubectl create secret generic mysql-secret --from-literal=MYSQL_PASSWORD=root
```

Now, let's put these concepts into practice by setting up ConfigMaps and Secrets for a two-tier application running on Flask and MySQL within a Kubernetes cluster.

ConfigMaps and Secrets in a Two-Tier Application

In this example, we have a two-tier application consisting of a Flask backend and a MySQL database. We'll create ConfigMaps and Secrets to manage the configuration and sensitive data for these components.

Creating ConfigMaps for a Two-Tier Application

Step 1: Create a ConfigMap for your Deployment

Let's start by creating a ConfigMap for your Flask deployment. We'll define the configuration in a YAML file, `mysql-config.yaml`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-config
data:
  MYSQL_HOST: mysql
  MYSQL_USER: root
  MYSQL_DATABASE: mydb
```

In this example, we define key-value pairs for the database URL, API key, and debug mode.

Step 2: Update the Flask deployment to include the ConfigMap

To use the ConfigMap in your deployment, you need to update the deployment.yml file to include the `envFrom` field under the `spec.containers` section. The `envFrom` field allows you to inject all the key-value pairs from the ConfigMap as environment variables into your containers.

For example, you can update your deployment.yml file for your Flask web server with the following content:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: two-tier-app
  labels:
    app: two-tier-app
spec:
  replicas: 1
  selector:
```

```
      matchLabels:
        app: two-tier-app
    template:
      metadata:
        labels:
          app: two-tier-app
      spec:
        containers:
          - name: two-tier-app
            image: 'trainwithshubham/flaskapp:latest'
            envFrom: null
          - configMapRef:
              name: mysql-config
            ports:
              - containerPort: 5000
            imagePullPolicy: Always
```

By referencing the ConfigMap `mysql-config`, your Flask application can access these configuration values.

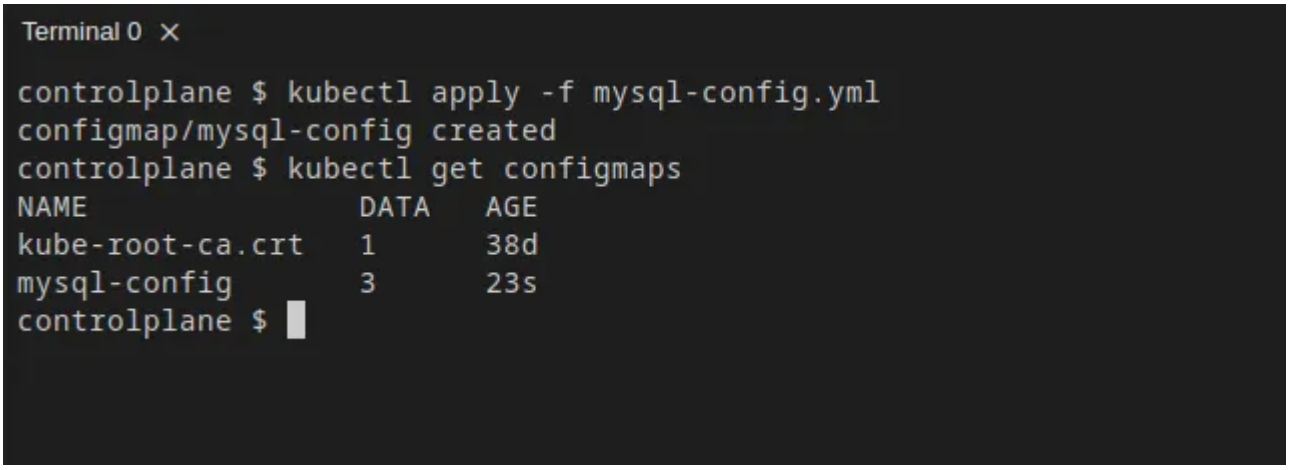
Step 3: Apply the updated deployment

To apply the updated deployment to your cluster, you need to use the `kubectl apply` command with the `-f` option and specify the name of your deployment file. For example, you can run the following command:

```
kubectl apply -f two-tier-app-deployment.yml
```

Step 4: Verify that the ConfigMap has been created

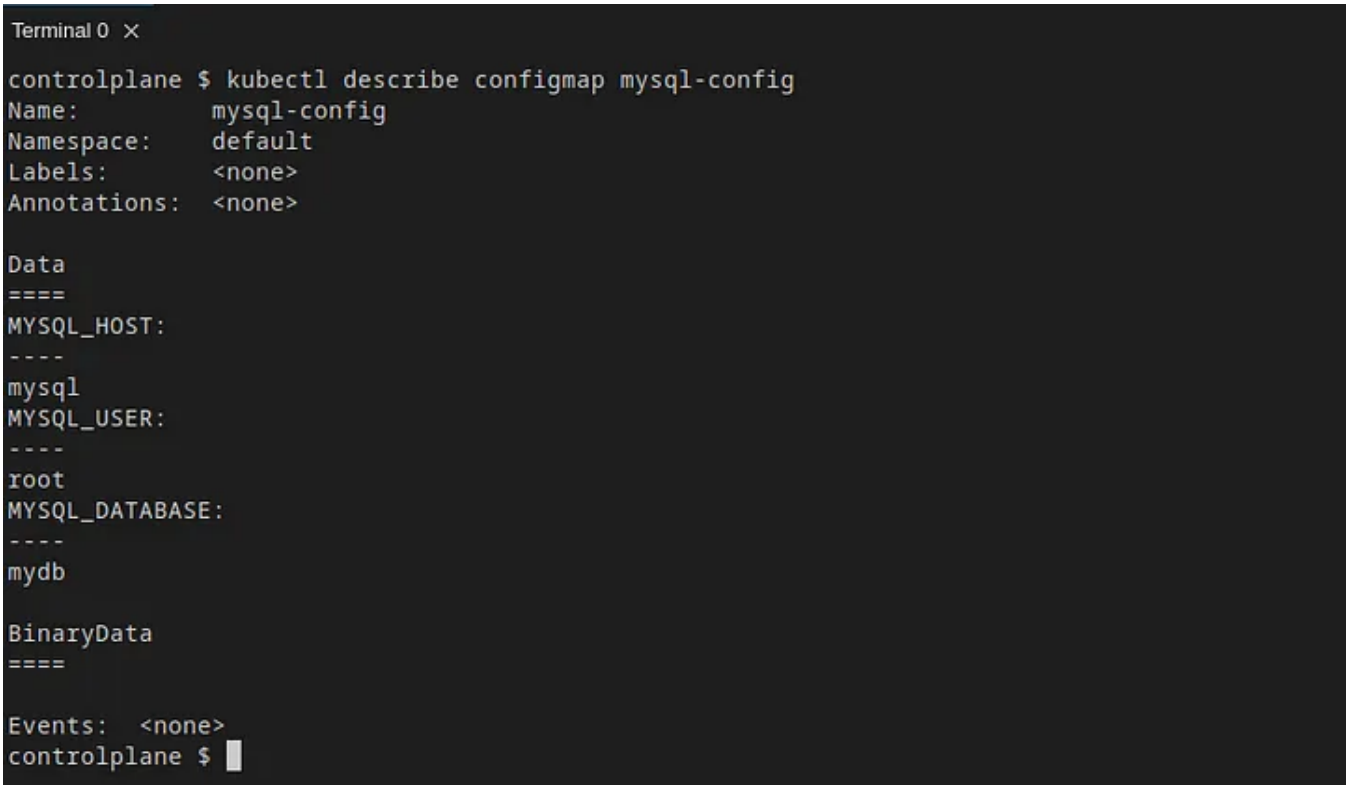
To verify that the ConfigMap has been created, you can use the `kubectl get configmaps` command to list all the ConfigMaps. You should see something like this:



To inspect the details of a specific ConfigMap, you can use the `kubectl describe configmap` command with the name of the ConfigMap. For example, you can run the following command:

```
kubectl describe configmap mysql-config
```

You should see something like this:



Creating Secrets for a Two-Tier Application

Step 1: Create a Secret for your Deployment

Let's create a Secret to securely store the MySQL database credentials. Define the Secret in a YAML file, `mysql-secret.yaml`:

```
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret
type: Opaque
```

```
data:
  MYSQL_PASSWORD: YWRtaW4K # This is the base64 encoded value of "admin".
```

In this example, we encode the password to keep it secure.

Step 2: Update the MySQL deployment to include the Secret

To use the Secret in your deployment, you need to update the deployment.yml file to include the env field under the `spec.containers` section. The env field allows you to inject specific key-value pairs from the Secret as environment variables into your containers.

For example, you can update your deployment.yml file for your Flask web server with the following content:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: two-tier-app
  labels:
    app: two-tier-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: two-tier-app
  template:
    metadata:
      labels:
        app: two-tier-app
    spec:
      containers:
        - name: two-tier-app
          image: trainwithshubham/flaskapp:latest
        envFrom:
        - configMapRef:
            name: mysql-config # This refers to the name of the ConfigMap that
        env:
        - name: MYSQL_ROOT_PASSWORD # This is an additional environment variabl
          valueFrom:
            secretKeyRef:
              name: mysql-secret # This refers to the name of the Secret that w
              key: MYSQL_PASSWORD # This refers to the key of the data that we
```

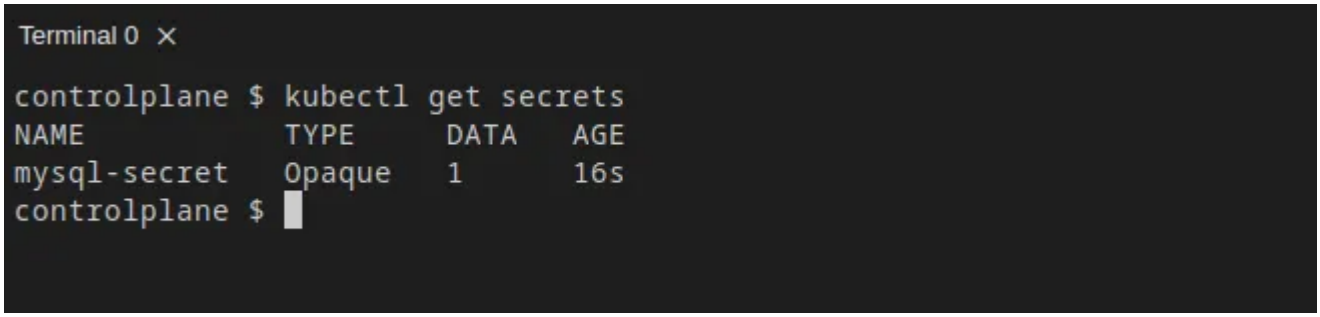
Step 3: Apply the updated deployment

Apply the changes to your Kubernetes cluster:

```
kubectl apply -f two-tier-app-deployment.yml
```

Step 4: Verify that the Secret has been created

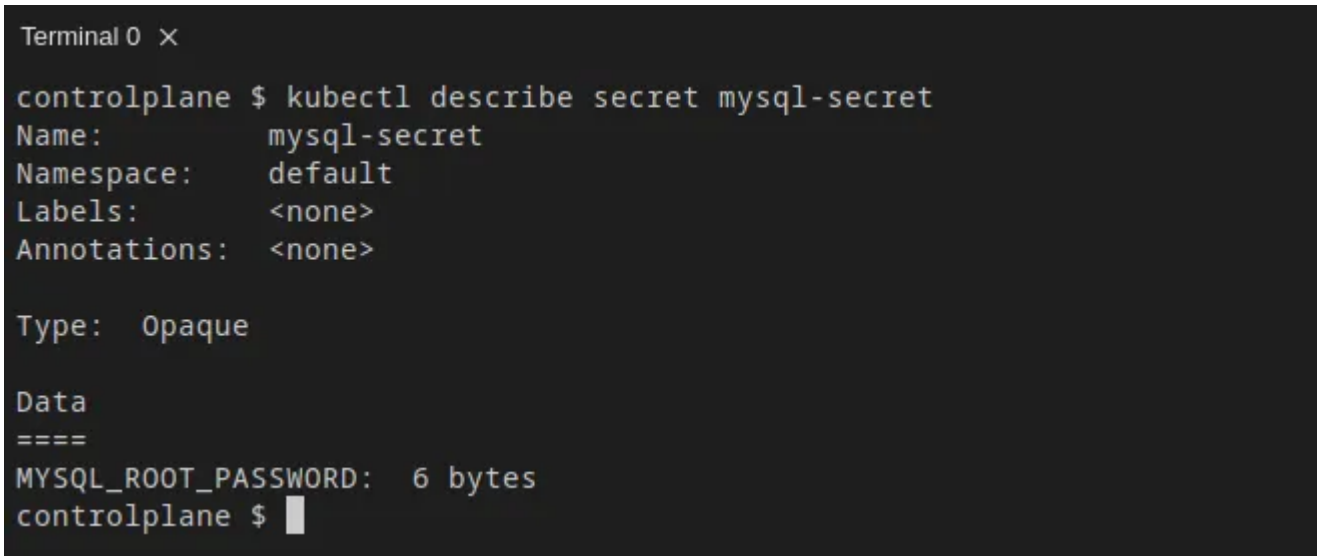
To verify that the Secret has been created, you can use the `kubectl get secrets` command to list all the Secrets. You should see something like this:



To inspect the details of a specific Secret, you can use the `kubectl describe secret` command with the name of the Secret. For example, you can run the following command:

```
kubectl describe secret mysql-secret
```

You should see something like this:



I've presented the two-tier deployment configuration here, but you can find all the configuration files for this project in the Git repository.

https://github.com/ajitfawade/two-tier-flask-app

Exploring Real-World Scenarios

The above steps demonstrate how to set up ConfigMaps and Secrets for a two-tier application. However, in real-world scenarios, you might encounter more complex requirements, such as dynamic updates, multiple namespaces, or external secrets management tools. The flexibility of Kubernetes allows you to adapt to these challenges efficiently.

Conclusion

In this blog post, we learned how to create ConfigMap and Secret for a two-tier application in Kubernetes. We saw how to create, update, and apply ConfigMap and Secret for our deployments using files or the command line. We also learned how to verify that our ConfigMap and Secret have been created and applied correctly by checking the status of our resources.

ConfigMap and Secret are essential tools for managing configuration data and secrets in Kubernetes. They help us keep our deployments consistent, secure, and scalable. By mastering ConfigMap and Secret, we can ensure that our Kubernetes clusters run smoothly and efficiently.

If you want to learn more about ConfigMap and Secret, you can check out these resources:

- Kubernetes Documentation: ConfigMaps

- Kubernetes Documentation: Secrets
- Kubernetes Tutorial: Configure a Pod to Use a ConfigMap
- Kubernetes Tutorial: Distribute Credentials Securely Using Secrets

I hope you enjoyed this blog post on creating ConfigMap and Secret for a two-tier application in Kubernetes. Please leave your feedback or questions in the comments section below.

Happy sailing on your Kubernetes spaceship! 🌟

DevOps Kubernetes Configmap 90daysofdevops Beginners Guide

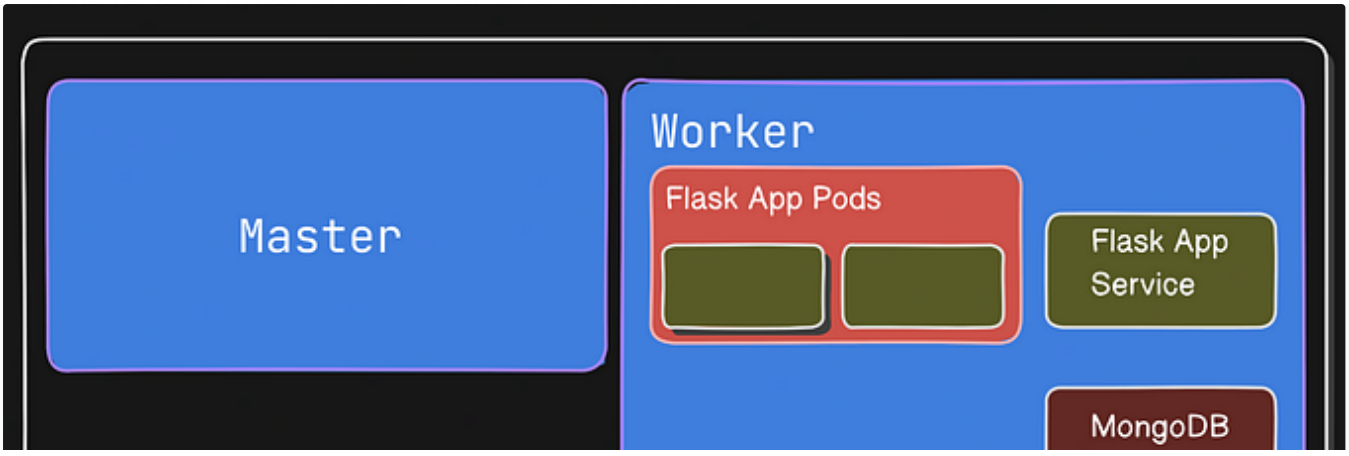
Follow

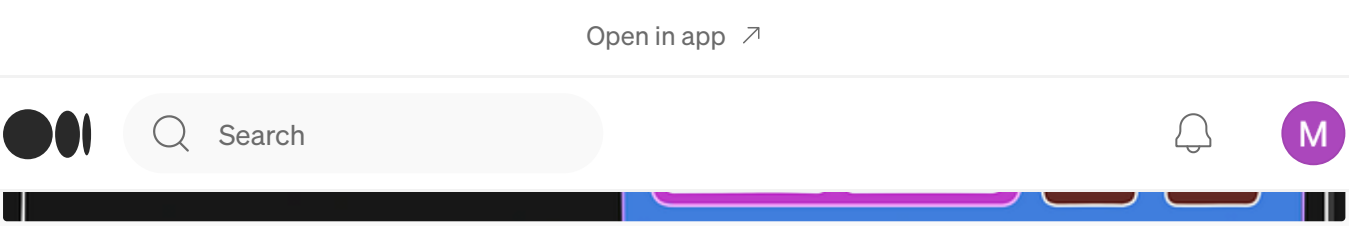
Written by Ajit Fawade

196 Followers

DevOps Enthusiast

More from Ajit Fawade





Ajit Fawade

How to Deploy a Microservices Application on AWS EC2 using Kubernetes: A Step-by-Step Guide

Learn how to deploy a microservices application on AWS EC2 using Kubernetes.

14 min read · Oct 9, 2023

64 1

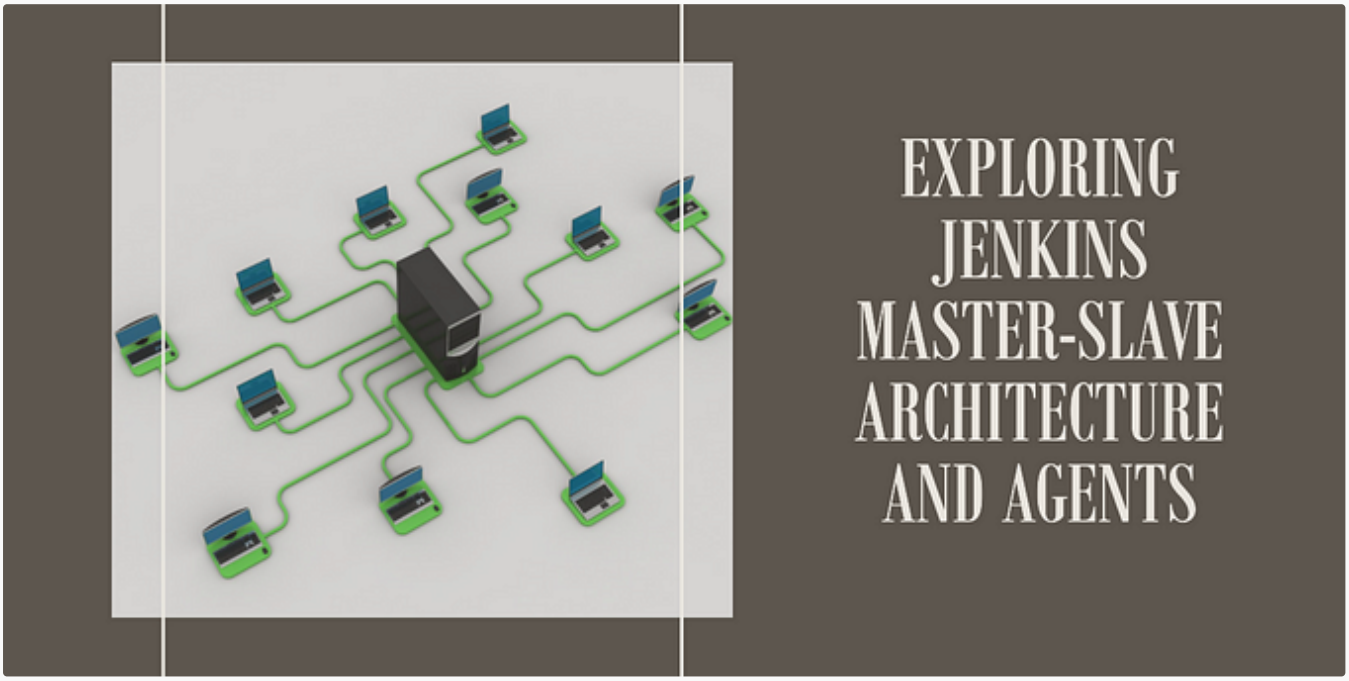
Ajit Fawade

Jenkins Interview Questions and Answers | Day 29 of 90 Days of DevOps

Learn some of the important interview questions and answers related to Jenkins, one of the most popular tools for CI/CD

11 min read · Sep 3, 2023

31



Ajit Fawade

Jenkins Master-Slave Architecture and Agents | Day 28 of 90 Days of DevOps

Learn how to set up Jenkins master-slave architecture and agents on AWS EC2 machines

8 min read · Sep 2, 2023

29

Ajit Fawade

How to Use Docker in Jenkins Declarative Pipeline | Day 27 of 90 Days of DevOps

Learn how to use Docker commands and syntax in Jenkins declarative pipeline to build and run Docker images and containers.

5 min read · Aug 30, 2023

22

See all from Ajit Fawade

Recommended from Medium

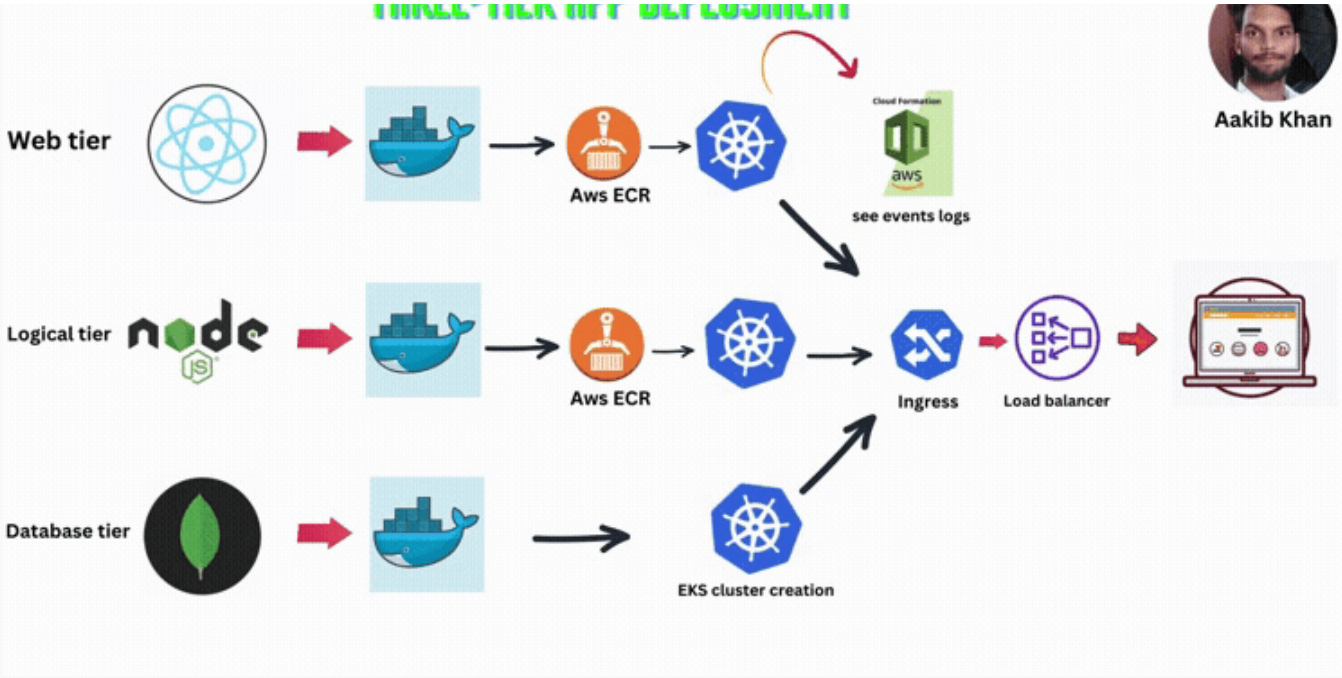
Ajit Fawade

How to Use Docker in Jenkins Declarative Pipeline | Day 27 of 90 Days of DevOps

Learn how to use Docker commands and syntax in Jenkins declarative pipeline to build and run Docker images and containers.

5 min read · Aug 30, 2023

22



Aakib

Project 8 → Three tier application deployment on Kubernetes

what do you mean by three tier ?

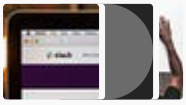
12 min read · Jan 26

131 1

Lists

Practical Guides to Machine Learning
10 stories · 1016 saves

General Coding Knowledge
20 stories · 885 saves



Productivity
237 stories · 319 saves

Natural Language Processing
1162 stories · 639 saves

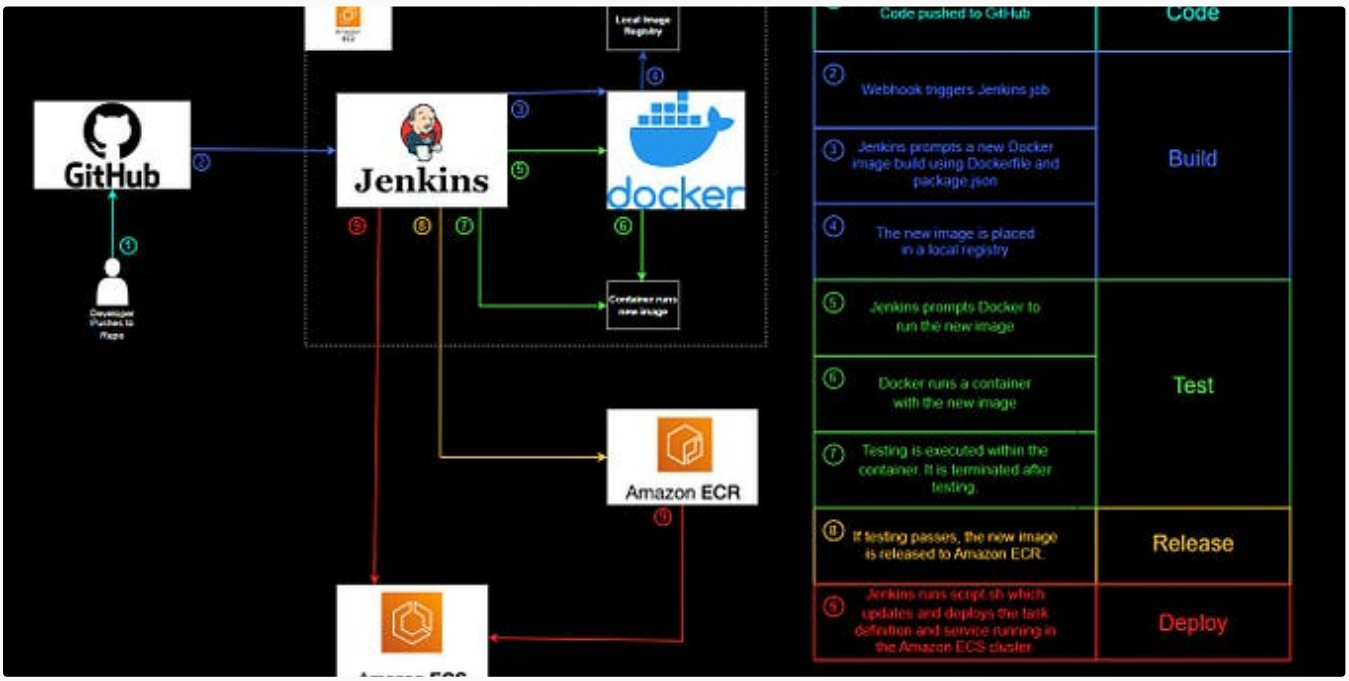
Ekant Mate (AWS APN Ambassador) in Technology Hits

Kubernetes Interview Questions: The Basics 🌟

🚀 Interview Preparation: Mastering Kubernetes Fundamentals 🌟

✨ · 23 min read · Jan 21

366 6



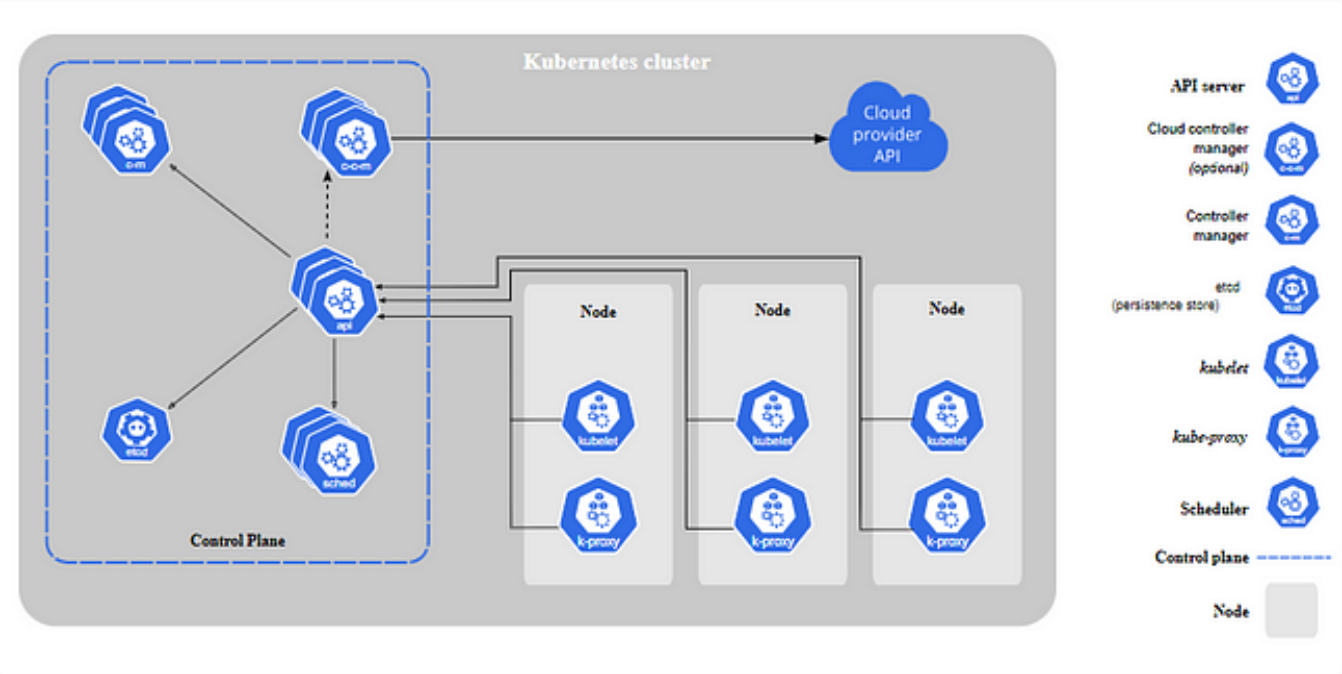
Anna Beagle

How to Create a CI/CD Pipeline for Amazon ECS with Jenkins and Docker

This is a detailed tutorial for creating an end-to-end CI/CD pipeline. An EC2 instance is used as a Jenkins server in order to...

14 min read · Aug 30, 2023

42



Himanshu Sangshetti

Kubernetes: Architecture and Components explained

WHAT IS KUBERNETES?

8 min read · Aug 18, 2023

55

Mesut Oezdil

Preparation for the K8s ⎈ Interviews

100 Kuberbetes Interview Questions

46 min read · Aug 21, 2023

231

See more recommendations