**To automate the deployment of MediaWiki in AWS using Kubernetes with Helm chart, using our own Dockerfiles for the application and database, and incorporating Blue-Green deployment (Using Jenkins) in a CI/CD pipeline, follow the below step-by-step instructions:**

**{Advantage of using this approach - It provides flexibility and allows you to customize the deployment according to your specific requirements.}**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**High level overview of the assignment:**

**1. Set up the AWS infrastructure:**

**- Create an Amazon EKS cluster to host your Kubernetes cluster.**

**- Set up the required AWS resources like VPC, subnets, security groups, and IAM roles.**

**2. Install and configure Jenkins:**

**- Launch an EC2 instance to host Jenkins.**

**- Install Jenkins and set up necessary plugins and configurations.**

**- Configure Jenkins to access your AWS resources.**

**3. Create Docker images:**

**- Write Dockerfiles for the MediaWiki application and database.**

**- Build and tag the Docker images for both components.**

**- Push the Docker images to a container registry (e.g., Amazon ECR).**

**4. Set up Helm chart for MediaWiki:**

**- Create a Helm chart with the necessary configurations for the MediaWiki deployment.**

**- Customize the chart to use your Docker images for both the application and the database.**

**5. Configure Blue-Green deployment in the Helm chart:**

**- Modify the Helm chart to support Blue-Green deployment.**

**- Create two sets of Kubernetes resources: one for the "blue" environment and another for the "green" environment.**

**- Set up Ingress or a Load Balancer to handle traffic routing between environments.**

**6. Set up Jenkins Pipeline:**

**- Create a Jenkins Pipeline in your MediaWiki project repository.**

**- Define stages for the CI/CD pipeline, including building Docker images, deploying to "green" environment, switching traffic.**

**7. Integrate AWS services in the pipeline:**

**- Use AWS CLI within your Jenkins Pipeline to interact with AWS services.**

**8. Trigger the Jenkins Pipeline:**

**- Configure webhook triggers to start the pipeline whenever there are changes to the MediaWiki project repository.**

**9. Execute the CI/CD pipeline:**

**- Save and run the Jenkins Pipeline to initiate the deployment process.**

**- Jenkins will build the Docker images, deploy the "green" environment, run tests, switch traffic, and clean up the "blue" environment automatically.**

**By following these steps, you can automate the deployment of MediaWiki in AWS using Kubernetes with Helm chart, your own Dockerfiles, and the Blue-Green deployment approach in a CI/CD pipeline.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Detailed information on how to implement the project with every step explained in detail along with the required code:**

**1. Set up the AWS infrastructure:**

**- Create an Amazon EKS cluster or use an existing one to host your Kubernetes cluster.**

**- Set up the required AWS resources like VPC, subnets, security groups, and IAM roles.**

To create an Amazon EKS (Elastic Kubernetes Service) cluster in AWS, follow these step-by-step instructions:

1. Sign into the AWS Management Console.

2. Open the Amazon EKS console.

3. Click on "Create cluster" to start the cluster creation process.

4. Select the "Create standard cluster" option.

5. Configure cluster details:

- Cluster name: Provide a unique name for your EKS cluster. **(Mediawki Cluster)**

- Kubernetes version: Select the desired version of Kubernetes. **(Version 1.26)**

- Role ARN: create a new role. This role will grant necessary permissions to EKS for managing resources on your behalf.

6. Configure networking:

- VPC: Select the VPC in which you want to create the EKS cluster.

- Subnets: Choose the subnets where EKS will deploy worker nodes. You need to select at least two subnets in different availability zones.

- Security groups: Select or create a security group to control inbound and outbound traffic for your EKS cluster.

7. Configure logging: - Enable control plane logging according to your preference.

8. Configure tags: - Add tags to your EKS cluster for better organization and identification.

9. Configure endpoint access (optional):

- You can specify whether the cluster's Kubernetes API server should be publicly accessible or accessible only from within the VPC.

10. Click on "Create" to create the EKS cluster.

Once the cluster is created, we can access and manage it using the AWS Management Console, AWS CLI, or other Kubernetes management tools like *kubectl*.

**2. Install and configure Jenkins:**

**- Launch an EC2 instance to host Jenkins.**

**- Install Jenkins and set up necessary plugins and configurations.**

**- Configure Jenkins to access your AWS resources (credentials, AWS CLI, etc.).**

1. Launch an EC2 instance:

- Launch an EC2 instance using an RHEL 8 AMI.

- Make sure the instance is launched in a subnet with internet connectivity and has a security group allowing inbound access on the necessary ports (e.g., SSH, HTTP, HTTPS).

2. Connect to the EC2 instance:

- Connect to the EC2 instance using SSH.

- Log in as the user = *MediaWiki\_User*

3. Install Java:

- Install Java Development Kit (JDK) on the EC2 instance. Jenkins requires Java to run.

- The exact installation command is:

*sudo yum install java-1.8.0-openjdk*

4. Install Jenkins:

- Add the Jenkins repository to the package manager:

*sudo yum install wget*

*wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat-stable/jenkins.repo # rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key*

- Update the package manager and install Jenkins:

*sudo yum apt update*

*sudo yum apt install Jenkins*

5. Start Jenkins:

- Start the Jenkins service:

*sudo yum systemctl start jenkins*

- Enable the Jenkins service to start on boot:

*sudo yum systemctl enable jenkins*

6. Configure Jenkins:

- Open a web browser and access Jenkins using the public IP address or domain name of your EC2 instance, followed by port 8080 (e.g., http://your-ip-address:8080).

- Retrieve the initial administrator password */var/lib/jenkins/secrets/initialAdminPassword* file.:

- Copy the password and paste it into the Jenkins setup wizard.

- Follow the on-screen instructions to complete the setup, choosing the recommended plugins.

7. Install AWS CLI:

- Install the AWS Command Line Interface (CLI) on the EC2 instance to interact with AWS services.

*sudo apt install awscli*

8. Configure AWS credentials:

- Configure AWS credentials for Jenkins by running the AWS CLI configuration command:

*aws configure*

- Enter your AWS Access Key ID, Secret Access Key, default region name, and default output format when prompted.

9. Grant AWS permissions:

- Create an IAM user in the AWS Management Console with the necessary permissions to access the AWS resources required for your Jenkins pipeline.

- Attach an appropriate IAM policy user, granting permissions to access AWS services and resources.

After completing these steps, you should have Jenkins installed and configured on your EC2 instance with access to AWS resources.

**3. Create Docker images:**

**- Write Dockerfiles for the MediaWiki application and database.**

1. Dockerfile for MediaWiki Application:

*FROM php:7.4-apache* # Base image

*WORKDIR /var/www/html* # Set the working directory

# Copy the MediaWiki source code to the container

*https://github.com/wikimedia/mediawiki-docker*

# Install dependencies and PHP extensions

*apt-get update && apt-get install -y \*

*git \*

*imagemagick \*

*libicu-dev \*

*libjpeg-dev \*

*libldap2-dev \*

*libpng-dev \*

*libzip-dev \*

*unzip \*

*zlib1g-dev \*

*&& docker-php-ext-configure gd --with-jpeg \*

*&& docker-php-ext-install -j$(nproc) \*

*gd \*

*intl \*

*ldap \*

*opcache \*

*zip \*

*&& a2enmod rewrite*

# Expose the port

*EXPOSE 80*

# Start Apache server

*CMD ["apache2-foreground"]*

2. Dockerfile for MediaWiki Database (MySQL):

*FROM mysql:8.0* # Base image

# Environment variables for MySQL configuration

*ENV MYSQL\_DATABASE=mediawiki \*

*MYSQL\_USER=wikiuser \*

*MYSQL\_PASSWORD=secretpassword \*

*MYSQL\_ROOT\_PASSWORD=secretpassword*

# Copy the custom MySQL configuration file

*COPY my.cnf /etc/mysql/conf.d/my.cnf*

# Expose the port

*EXPOSE 3306*

# Start MySQL server

*CMD ["mysqld"]*

**- Build and tag the Docker images for both components.**

1. Navigate to the directory containing the MediaWiki application Dockerfile.

2. Open a terminal and execute the following command to build the MediaWiki application image:

*docker build -t mediawiki-app:latest .*

This command will build the Docker image using the Dockerfile in the current directory (`.`) and tag it as `mediawiki-app:latest`.

3. Execute the following command to build the MediaWiki database image:

*docker build -t mediawiki-db:latest .*

4. After the build process finishes for both images, you can verify the images by executing the following command:

*docker images*

This command lists all the Docker images on your machine, and you should see the `mediawiki-app` and `mediawiki-db` images with the appropriate tags.

Now you have successfully built and tagged the Docker images for both the MediaWiki application and database components.

**- Push the Docker images to a container registry - Amazon ECR.**

To push the Docker images for both the MediaWiki application and database components to Amazon ECR, follow these steps:

1. Log in to the container registry:

- In Amazon ECR, we need to authenticate the Docker CLI with the registry. Run the following command:

*aws ecr get-login-password --region <your-aws-region> | docker login --username AWS --password-stdin <your-aws-account-id>.dkr.ecr.<your-aws-region>.amazonaws.com/<your-ecr-repository>*

This command retrieves an authentication token and logs in to the Amazon ECR registry.

2. Tag the Docker images with the registry information:

- Run the following commands to retag the Docker images with the ECR registry information.

*docker tag mediawiki-app:latest <your-aws-account-id>.dkr.ecr.<your-aws-region>.amazonaws.com/<your-ecr-repository>/mediawiki-app:latest*

*docker tag mediawiki-db:latest <your-aws-account-id>.dkr.ecr.<your-aws-region>.amazonaws.com/<your-ecr-repository>/mediawiki-db:latest*

These commands retag the images with the ECR repository information.

3. Push the Docker images to the container registry:

- Execute the following commands to push the images to the ECR repository:

*docker push <your-aws-account-id>.dkr.ecr.<your-aws-region>.amazonaws.com/<your-ecr-repository>/mediawiki-app:latest*

*docker push <your-aws-account-id>.dkr.ecr.<your-aws-region>.amazonaws.com/<your-ecr-repository>/mediawiki-db:latest*

4. Verify the pushed images:

- To verify that the images are successfully pushed to the container registry, we need to navigate to the ECR console in the AWS Management Console. Select our repository, and we should see the `mediawiki-app` and `mediawiki-db` images listed.

Now the Docker images for the MediaWiki application and database are pushed to the container registry. We can use these images when deploying our application in your AWS environment.

**4. Set up Helm chart for MediaWiki:**

**- Create a Helm chart with the necessary configurations for the MediaWiki deployment.**

**- Customize the chart to use your Docker images for both the application and the database.**

1. Install Helm:

- Ensure that you have Helm installed on your local machine. You can download and install Helm from the official Helm website: https://helm.sh/docs/intro/install/.

2. Initialize a Helm chart:

- Open an AWS terminal and navigate to the directory where you want to create your Helm chart.

- Run the following command to initialize a new Helm chart:

*helm create mediawiki-chart*

This command creates a new directory named `mediawiki-chart` with the basic structure of a Helm chart.

3. Customize the chart:

- Open the `*mediawiki-chart*` directory in a text editor or IDE.

- Update the `*values.yaml*` file with the necessary configurations for MediaWiki deployment. We can specify the image repository, tag, and other deployment-specific values.

4. Use custom Docker images:

- In the `*values.yaml*` file, find the `*image*` section under the `*mediawiki*` and `*mediawiki-database*` sections.

- Modify the `*repository*` field to specify the repository of your custom Docker images.

5. Package the Helm chart:

- In the root directory of your Helm chart (`*mediawiki-chart*`), run the following command to package the chart:

*helm package .*

This command creates a `.tgz` file containing your Helm chart.

Now you have created a Helm chart for deploying MediaWiki, customized it to use your Docker images for both the application and the database, and packaged it into a `.tgz` file. You can use this Helm chart to deploy MediaWiki using your custom Docker images in a Kubernetes cluster with Helm.

**5. Configure Blue-Green deployment in the Helm chart:**

**- Modify the Helm chart to support Blue-Green deployment.**

To configure Blue-Green deployment in the Helm chart, you can follow these steps:

1. Open the `*values.yaml*` file in your Helm chart directory (`*mediawiki-chart*`).

2. Add the following values under the `*mediawiki*` section to define the Blue-Green deployment strategy:

*yaml*

*mediawiki:*

*deploymentStrategy: bluegreen*

*blueReplicaCount: 1*

*greenReplicaCount: 0*

- `*deploymentStrategy*`: Set this value to `bluegreen` to enable Blue-Green deployment strategy.

- `*blueReplicaCount*`: Specify the number of replicas for the "blue" version of the application. In here, it is set to `1`.

- `*greenReplicaCount*`: Specify the number of replicas for the "green" version of the application. In here, it is set to `0`.

3. Update the `*deployment.yaml*` file in the templates directory of your Helm chart.

4. In the `*deployment.yaml*` file, modify the `replicas` field under the `*spec.template.spec*` section as follows:

*yaml*

*spec:*

*replicas: {{ .Values.mediawiki.blueReplicaCount }}*

- This configuration sets the number of replicas for the "blue" version of the application based on the value specified in the `*blueReplicaCount*` field in the `*values.yaml*` file.

5. Add a new block for the "green" version of the application in the `*deployment.yaml*` file, below the existing block for the "blue" version. The "green" version should have the `replicas` field set to *`{{ .Values.mediawiki.greenReplicaCount }}`.*

After configuring Blue-Green deployment in the Helm chart, you can use Helm to deploy and manage the MediaWiki application with the Blue-Green strategy. When deploying a new version, you can gradually shift traffic from the "blue" deployment to the "green" deployment and roll back if needed.

**- Create two sets of Kubernetes resources (deployments, services, etc.): one for the "blue" environment and another for the "green" environment.**

**- Set up Ingress or a LoadBalancer to handle traffic routing between environments.**

1. Create the Kubernetes resources for the "blue" environment:

- Create a deployment for the "blue" environment by applying the following YAML configuration (save it as `*mediawiki-blue-deployment.yaml*`):

*yaml*

*apiVersion: apps/v1*

*kind: Deployment*

*metadata:*

*name: mediawiki-blue-deployment*

*spec:*

*replicas: 1*

*selector:*

*matchLabels:*

*app: mediawiki*

*environment: blue*

*template:*

*metadata:*

*labels:*

*app: mediawiki-app:latest*

*environment: blue*

*spec:*

*containers:*

*- name: mediawiki*

*image: <your-ecr-repository>/mediawiki-app:latest*

*ports:*

*- containerPort: 80*

- Apply the deployment to the Kubernetes cluster by running the following command:

*kubectl apply -f mediawiki-blue-deployment.yaml*

2. Create the Kubernetes resources for the "green" environment:

- Create a deployment for the "green" environment by applying the following YAML configuration (save it as `*mediawiki-green-deployment.yaml*`):

*yaml*

*apiVersion: apps/v1*

*kind: Deployment*

*metadata:*

*name: mediawiki-green-deployment*

*spec:*

*replicas: 1*

*selector:*

*matchLabels:*

*app: mediawiki-app:latest*

*environment: green*

*template:*

*metadata:*

*labels:*

*app: mediawiki*

*environment: green*

*spec:*

*containers:*

*- name: mediawiki*

*image: <your-ecr-repository>/mediawiki-app:latest*

*ports:*

*- containerPort: 80*

- Apply the deployment to the Kubernetes cluster by running the following command:

*kubectl apply -f mediawiki-green-deployment.yaml*

3. Set up Ingress or LoadBalancer for traffic routing:

- If you want to use Ingress, create an Ingress resource with rules for routing traffic to the appropriate environment. Modify the host and paths according to your requirements. Save the following YAML configuration as `*mediawiki-ingress.yaml*`:

*```yaml*

*apiVersion: networking.k8s.io/v1*

*kind: Ingress*

*metadata:*

*name: mediawiki-ingress*

*spec:*

*rules:*

*- host: mediawiki-app:latest*

*http:*

*paths:*

*- path: /blue*

*pathType: Prefix*

*backend:*

*service:*

*name: mediawiki-blue-service*

*port:*

*number: 80*

*- path: /green*

*pathType: Prefix*

*backend:*

*service:*

*name: mediawiki-green-service*

*port:*

*number: 80*

- Apply the Ingress resource to the Kubernetes cluster by running the following command:

*kubectl apply -f mediawiki-ingress.yaml*

- If you want to use a LoadBalancer, create two separate Services, one for the "blue" environment and another for the "green" environment. Modify the service configurations according to your requirements. Save the following YAML configuration as `*mediawiki-loadbalancer.yaml*`:

*yaml*

*apiVersion: v1*

*kind: Service*

*metadata:*

*name: mediawiki-blue-service*

*spec:*

*selector:*

*app: mediawiki-app:latest*

*environment: blue*

*ports:*

*- protocol: TCP*

*port: 80*

*targetPort: 80*

*type: LoadBalancer*

*---*

*apiVersion: v1*

*kind: Service*

*metadata:*

*name: mediawiki-green-service*

*spec:*

*selector:*

*app: mediawiki-app:latest*

*environment: green*

*ports:*

*- protocol: TCP*

*port: 80*

*targetPort: 80*

*type: LoadBalancer*

- Apply the LoadBalancer service configurations to the Kubernetes cluster by running the following command:

*kubectl apply -f mediawiki-loadbalancer.yaml*

4. Once the resources are applied successfully, the "blue" and "green" environments should be deployed, and the Ingress or LoadBalancer will handle the traffic routing between the environments based on the defined rules.

**6. Set up Jenkins Pipeline:**

**- Create a Jenkins Pipeline in your MediaWiki project repository.**

**- Define stages for the CI/CD pipeline, including building Docker images, deploying to "green" environment, switching traffic.**

To create a Jenkins Pipeline for deploying the MediaWiki application with Blue-Green deployment, we can *Jenkinsfile*.

*groovy*

*pipeline {*

*agent any*

*stages {*

*stage('Build') {*

*steps {*

*// Clone your MediaWiki project repository*

*git 'https://github.com/your-username/mediawiki.git'*

*// Build the MediaWiki application Docker image*

*sh 'docker build -t mediawiki-app:latest .'*

*// Push the Docker image to your container registry*

*sh 'docker push your-ecr-repository/mediawiki-app:latest'*

*}*

*}*

*stage('Deploy Blue') {*

*steps {*

*// Deploy the "blue" environment using Helm chart*

*sh 'helm upgrade --install mediawiki-blue mediawiki-chart --set mediawiki.deploymentStrategy=bluegreen,mediawiki.blueReplicaCount=1,mediawiki.greenReplicaCount=0'*

*}*

*}*

*stage('Deploy Green') {*

*steps {*

*// Build and push the new version of the MediaWiki application Docker image*

*sh 'docker build -t mediawiki-app:newversion .'*

*sh 'docker push your-ecr-repository/mediawiki-app:newversion'*

*// Deploy the "green" environment using Helm chart*

*sh 'helm upgrade --install mediawiki-green mediawiki-chart --set mediawiki.deploymentStrategy=bluegreen,mediawiki.blueReplicaCount=0,mediawiki.greenReplicaCount=1'*

*}*

*}*

This Jenkinsfile defines a pipeline with stages for building the Docker image, deploying the "blue" environment, deploying the "green" environment, switching traffic from "blue" to "green".

**7. Integrate AWS services in the pipeline:**

**- Use AWS CLI within your Jenkins Pipeline to interact with AWS services.**

To integrate AWS services into your Jenkins pipeline, you can utilize the AWS CLI within your pipeline stages. Steps to integrate AWS services into the pipeline:

1. Configure AWS CLI in the Jenkins pipeline:

- Add the following stage to your Jenkinsfile to configure the AWS CLI with the appropriate AWS credentials:

*groovy*

*stage('Configure AWS CLI') {*

*steps {*

*withCredentials([*

*[*

*$class: 'AmazonWebServicesCredentialsBinding',*

*credentialsId: 'aws-credentials',*

*accessKeyVariable: 'AWS\_ACCESS\_KEY\_ID',*

*secretKeyVariable: 'AWS\_SECRET\_ACCESS\_KEY'*

*]*

*]) {*

*sh 'aws configure set aws\_access\_key\_id $AWS\_ACCESS\_KEY\_ID'*

*sh 'aws configure set aws\_secret\_access\_key $AWS\_SECRET\_ACCESS\_KEY'*

*sh 'aws configure set default.region <your-aws-region>'*

*}*

*}*

*}*

2. Integrate AWS services in the desired pipeline stages:

- For example, if you want to deploy your MediaWiki application to Amazon EKS, you can use the AWS CLI to update the EKS cluster with the new version of your application. Add the following stage to your Jenkinsfile:

*groovy*

*stage('Deploy to EKS') {*

*steps {*

*sh 'aws eks update-kubeconfig --name <your-eks-cluster-name> --region <your-aws-region>'*

*sh 'kubectl apply -f mediawiki-deployment.yaml'*

*}*

*}*

- Modify the `*mediawiki-deployment.yaml*` file as per your requirements and deployment configuration.

3. Add error handling and cleanup:

- It's recommended to add error handling and cleanup steps within your Jenkins pipeline to handle failures and clean up any temporary resources created during the pipeline execution.

- We need to add a stage at the end of the pipeline to delete any temporary resources.

*groovy*

*stage('Cleanup') {*

*steps {*

*// Delete any temporary resources or perform cleanup tasks*

*}*

*}*

**8. Trigger the Jenkins Pipeline:**

**- Configure webhook triggers to start the pipeline whenever there are changes to the MediaWiki project repository.**

1. Set up a Jenkins job for your MediaWiki pipeline:

- Open your Jenkins dashboard.

- Click on "New Item" to create a new Jenkins job.

- Enter a name for the job and select the "Pipeline" option.

- Click "OK" to proceed.

2. Configure the pipeline to use your repository:

- In the pipeline configuration, under the "Pipeline" section, select the "Pipeline script from SCM" option.

- Choose your source code management system and provide the repository URL of your MediaWiki project.

3. Configure webhook triggers:

- Enable webhook triggers for your repository.

- In your repository settings, navigate to the webhook settings or integrations section.

- Create a new webhook and provide the Jenkins webhook URL.

- Ensure that the webhook is configured to trigger on push events or repository changes.

4. Save and test the configuration.

**9. Execute the CI/CD pipeline:**

**- Save and run the Jenkins Pipeline to initiate the deployment process.**

**- Jenkins will build the Docker images, deploy the "green" environment, run tests, switch traffic, and clean up the "blue" environment automatically.**

To save and run the Jenkins Pipeline, follow these steps:

1. Open your Jenkins dashboard.

2. Locate the Jenkins job that corresponds to your MediaWiki pipeline.

3. Click on the job to navigate to the job details page.

4. On the left side of the page, you should see a "Build Now" or "Build with Parameters" option. Click on it.

5. If you have configured any parameters for your pipeline, you will be prompted to enter the parameter values. Provide the required values and click "Build" to proceed.

6. Jenkins will start the pipeline execution, and you can monitor the progress in real-time through the Jenkins interface.

7. Once the pipeline execution is complete, you will see the final status (success or failure) of the pipeline.

**By following these steps, you can automate the deployment of MediaWiki in AWS using Kubernetes with Helm chart, your own Dockerfiles, and the Blue-Green deployment approach in a CI/CD pipeline. Adjust the instructions as per your specific AWS infrastructure, Jenkins setup, and deployment requirements.**