**CLOUD DEPLOYMENT DOCUMENTATION**

**Configuration: app-config.yaml**

**Overview**

This configuration file defines a Kubernetes ConfigMap object. ConfigMaps are used to store non-confidential data in key-value pairs, which can then be consumed by pods or used to set environment variables and command-line arguments in containerized applications.

**File Details**

* **apiVersion**: **v1**
  + This is the version of the Kubernetes API used for this resource. **v1** indicates it is a stable version.
* **kind**: **ConfigMap**
  + Specifies the kind of resource being defined. In this case, it's a ConfigMap, which is used to hold user-defined configuration data.
* **metadata**:
  + **name**: **enrollments-config**
    - The name of the ConfigMap. This name is used to reference the ConfigMap from other resources within the same namespace.
* **data**:
  + **app\_port**: **"80"**
    - Contains the configuration data in key-value pairs. Here, **app\_port** is set to **"80"**, which is likely used by the Enrollments application to determine the port on which it should listen. This port value can be injected into your application's containers using environment variables.

**Usage**

The **enrollments-config** ConfigMap can be used by pods in the following ways:

* As an environment variable inside a Kubernetes deployment to dynamically set the port for the Enrollments application.
* Can be referenced in pod templates to configure applications without hardcoding values into the application code.

**Configuration: db-credentials.yaml**

**Overview**

This configuration file defines a Kubernetes Secret object. Secrets are used in Kubernetes to store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys. Storing secrets in this way allows for more control over how sensitive information is used and reduces the risk of accidental exposure.

**File Details**

* **apiVersion**: **v1**
  + This is the version of the Kubernetes API used for this resource. **v1** indicates it is a stable version.
* **kind**: **Secret**
  + Specifies the kind of resource being defined. In this case, it's a Secret, which is used to store sensitive data securely.
* **metadata**:
  + **name**: **postgres-secret**
    - The name of the Secret. This name is used to reference the Secret from other resources within the same namespace.
* **type**: **Opaque**
  + Indicates that the Secret holds arbitrary data. This type is the default and is used when the secret does not need to be used for a specific service type.
* **data**:
  + **postgres-password**: **cGFzcw==**
    - Contains the sensitive data in key-value pairs. Here, the key **postgres-password** holds the Base64 encoded password. The value **cGFzcw==** decodes to "pass". This password can be used by your PostgreSQL database container.

**Usage**

The **postgres-secret** can be used by pods in the following ways:

* As an environment variable inside a Kubernetes deployment to securely pass the PostgreSQL password to your database container.
* Can be mounted as a file in a pod that requires access to the password.

**Script Overview: deploy.py**

**Purpose**

This Python script uses the **docker** Python library to automate the setup of a local Docker environment for the university's enrollment application and its associated PostgreSQL database. It ensures that both containers are connected on the same Docker network to facilitate communication between them.

**Functionality**

1. **Import Docker Library**: The script starts by importing the **docker** library, which allows for Docker operations to be performed using Python.
2. **Initialize Docker Client**: The **docker.from\_env()** method initializes a client session using the Docker configuration on the host machine. This client is used to interact with the Docker engine.
3. **Pull Docker Images**:
   * **markvellaum/university:v0.0.1**: The Docker image for the university's enrollment application.
   * **postgres:latest**: The latest Docker image for PostgreSQL.
4. **Manage Docker Network**:
   * Checks for an existing Docker network named **university-net**.
   * If found, the script stops and removes all containers attached to this network to prevent any conflict.
   * Deletes the existing network if it exists to start fresh.
   * Creates a new network named **university-net** using the bridge driver for inter-container communication.
5. **Deploy PostgreSQL Container**:
   * Runs the PostgreSQL container with the name **postgres-db**.
   * Attaches it to the **university-net** network.
   * Sets environment variables like **POSTGRES\_DB**, **POSTGRES\_USER**, and **POSTGRES\_PASSWORD** to configure the database.
   * Maps the default PostgreSQL port **5432** on the container to the same port on the host.
6. **Deploy Enrollments Application Container**:
   * Runs the Enrollments application container with the name **enrollments-app**.
   * Attaches it to the **university-net** network.
   * Configures the application's connection to the PostgreSQL database using environment variables such as **PGHOST**, **PGPORT**, **PGDB**, **PGUSER**, and **PGPASS**.
   * Maps port **80** from the container to the host, enabling web access to the application.
7. **List Running Containers**:
   * Prints the names of all currently running containers, verifying that both the application and database containers are operational.

**Execution Instructions**

To run this script:

1. Ensure you have Docker installed and running on your system.
2. Install the **docker** Python library if not already installed using **pip install docker**.
3. Execute the script using Python by running **python deploy.py** in your terminal or command prompt.

**Configuration: enrollments-deployment.yaml**

**Overview**

This Kubernetes deployment configuration file specifies how the Enrollments application container should be deployed within a Kubernetes cluster. The configuration ensures the application is properly linked to the necessary PostgreSQL database service and securely manages sensitive credentials using Kubernetes secrets.

**File Details**

* **apiVersion**: **apps/v1**
  + Specifies the API version for the deployment configuration, which is stable and widely used for defining deployments in Kubernetes.
* **kind**: **Deployment**
  + Defines the type of resource being configured. In this case, it's a Deployment, which manages the deployment of a replicated application on a Kubernetes cluster.
* **metadata**:
  + **name**: **enrollments-app**
    - The name of the deployment. This name is used to reference the deployment within the Kubernetes cluster.

**Specification Details**

* **replicas**: **1**
  + Specifies the number of instances of the application that should be running. In this case, it is set to 1, meaning only one pod of the Enrollments application will be deployed.
* **selector**:
  + **matchLabels**:
    - **app**: **enrollments**
      * Selector that determines which pods belong to the deployment. Only pods with labels matching these selectors will be managed by this deployment.
* **template**:
  + Metadata and specifications for the pods that will be created as part of this deployment.
  + **metadata**:
    - **labels**:
      * **app**: **enrollments**
        + Labels applied to pods, used for organizing and selecting subsets of pods.
  + **spec**:
    - **containers**:
      * **name**: **enrollments**
        + The name of the container within the pod.
      * **image**: **markvellaum/university:v0.0.1**
        + The Docker image to use for the container, sourced from a Docker registry.
      * **ports**:
        + **containerPort**: **80**

The port on which the container will accept incoming connections.

* + - * **env**: Environment variables necessary for the container to correctly connect to the PostgreSQL database.
        + **PGHOST**: Points to the fully qualified domain name (FQDN) of the PostgreSQL service within the cluster.
        + **PGPORT**: The port on which the PostgreSQL service is exposed (5432).
        + **PGUSER**: Default PostgreSQL user.
        + **PGPASS**: References the password from a secret (**postgres-secret**) to secure database access.
        + **PGDB**: The specific database name to connect to within the PostgreSQL server.

**Usage**

This deployment configuration ensures that the Enrollments application is deployed with the correct environment settings, using secrets for sensitive data, and is scalable and maintainable within a Kubernetes environment. It is crucial for deployments where security and data integrity are a priority.

**Example of Using the Configuration**

To deploy this configuration:

1. Ensure that you have a Kubernetes cluster running.
2. Deploy the **postgres-secret** first to ensure the password is available.
3. Apply this deployment configuration by running:

bash

kubectl apply -f enrollments-deployment.yaml

**Configuration: enrollments-service.yaml**

**Overview**

This Kubernetes service configuration file defines how to expose the Enrollments application running within the Kubernetes cluster to external traffic or to other services within the cluster. It specifies the use of a NodePort service type, which makes the application accessible on a static port on each node of the cluster.

**File Details**

* **apiVersion**: **v1**
  + Indicates the version of the Kubernetes API used to create this resource. **v1** is the most stable and widely supported version for defining Kubernetes services.
* **kind**: **Service**
  + Specifies the kind of Kubernetes resource being defined. In this case, it's a Service, which is used to expose applications running on a set of Pods as a network service.
* **metadata**:
  + **name**: **enrollments-app**
    - The name of the service. This name is used within the Kubernetes cluster to reference this particular service.

**Specification Details**

* **type**: **NodePort**
  + Defines the type of service. **NodePort** exposes the service on a static port on each node’s IP address. NodePort services are accessible from outside the Kubernetes cluster by connecting to **<NodeIP>:<NodePort>**.
* **ports**:
  + A list defining the ports that the service should expose.
    - **port**: **80**
      * The port that the service will be accessed on within the cluster.
    - **targetPort**: **80**
      * The port on the container that the service routes to. In this case, it's routing to port 80 on the Enrollments application container.
    - **protocol**: **TCP**
      * The network protocol used by the service. TCP is typically used for applications requiring reliable communication.
* **selector**:
  + **app**: **enrollments**
    - A selector to identify the pods that this service should route traffic to. It matches pods labeled with **app: enrollments**, which should be the label set on the Enrollments application pods.

**Usage**

This service configuration is crucial for making the Enrollments application accessible to other parts of the cluster and potentially from outside the cluster. By using a NodePort, the service can be accessed from any node in the cluster at a specific port, making it flexible for development and testing environments.

**Example of Using the Configuration**

To apply this service configuration:

1. Ensure that the Enrollments application deployment is already applied and running in your Kubernetes cluster.
2. Deploy this service configuration by running:

kubectl apply -f enrollments-service.yaml

This will expose the Enrollments application on each node at a specific port, which you can access using any node's IP address and the NodePort assigned by Kubernetes.

**Configuration: postgres-pv.yaml**

**Overview**

This configuration file defines a Persistent Volume (PV) in Kubernetes, which provides an abstraction for storage resources. The PV specified in this file is intended for use by a PostgreSQL database, ensuring that data persists across pod restarts and deployments.

**File Details**

* **apiVersion**: **v1**
  + The version of the Kubernetes API used for this resource. **v1** is stable and widely used for defining persistent volumes.
* **kind**: **PersistentVolume**
  + Indicates the type of Kubernetes resource being defined, which in this case is a Persistent Volume.
* **metadata**:
  + **name**: **postgres-pv**
    - The name of the persistent volume, used to reference this volume in the cluster.

**Specification Details**

* **capacity**:
  + **storage**: **5Gi**
    - Specifies the size of the storage available in the volume, set to 5 gigabytes in this instance.
* **accessModes**:
  + **- ReadWriteOnce**
    - Defines the access mode of the volume, allowing it to be mounted as read-write by a single node.
* **persistentVolumeReclaimPolicy**: **Retain**
  + Determines what happens to the data after the persistent volume claim is released. **Retain** means that the volume and its data remain intact after it is released.
* **storageClassName**: **standard**
  + The StorageClass used for the volume. Storage classes determine policies for volume provisioning and management.
* **hostPath**:
  + **path**: **"/mnt/data"**
    - Specifies a path on the host node's filesystem that will be used by the PV. This is typically used for development and testing, not recommended for production environments.

**Usage**

This PV is crucial for database applications that require persistent data storage, like PostgreSQL. It ensures that the database can retain data even if the pod crashes or is rescheduled to another node.

**Configuration: postgres-pvc.yaml**

**Overview**

This configuration file defines a Persistent Volume Claim (PVC) in Kubernetes. A PVC is a request for storage by a user and consumes Persistent Volume resources.

**File Details**

* **apiVersion**: **v1**
  + Specifies the version of the Kubernetes API used for this resource, which is **v1**.
* **kind**: **PersistentVolumeClaim**
  + Defines the resource type as a Persistent Volume Claim, which is used to request storage resources defined by a Persistent Volume.
* **metadata**:
  + **name**: **postgres-pvc**
    - The name of the PVC, which is used to bind it to the corresponding PV.

**Specification Details**

* **accessModes**:
  + **- ReadWriteOnce**
    - Allows the volume to be mounted as read-write by a single node, matching the access mode of the PV.
* **resources**:
  + **requests**:
    - **storage**: **5Gi**
      * Requests 5 gigabytes of storage, matching the capacity provided by the PV.
* **storageClassName**: **standard**
  + Specifies the StorageClass associated with the PVC, which should match that of the PV to ensure they are compatible and can be bound correctly.

**Usage**

This PVC is designed to be used by your PostgreSQL pod to ensure that it has access to persistent storage. By specifying a matching StorageClass and access modes, it will bind to the **postgres-pv**, providing reliable and persistent data storage capabilities.