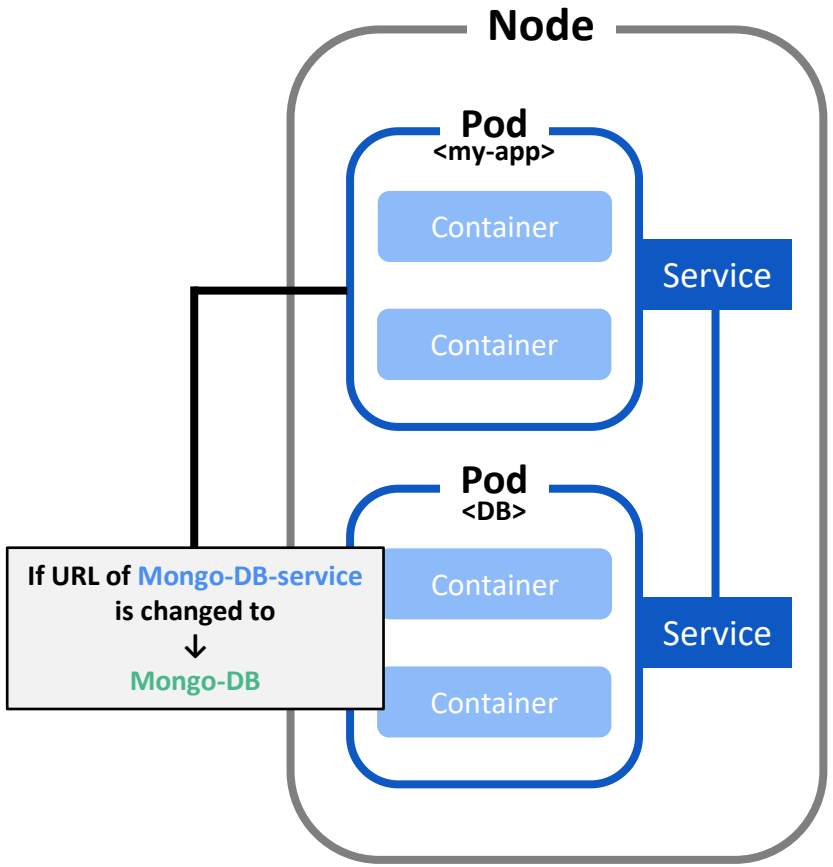
Secret and ConfigMap

DCN Lab

ConfigMap, Secret 의 필요성

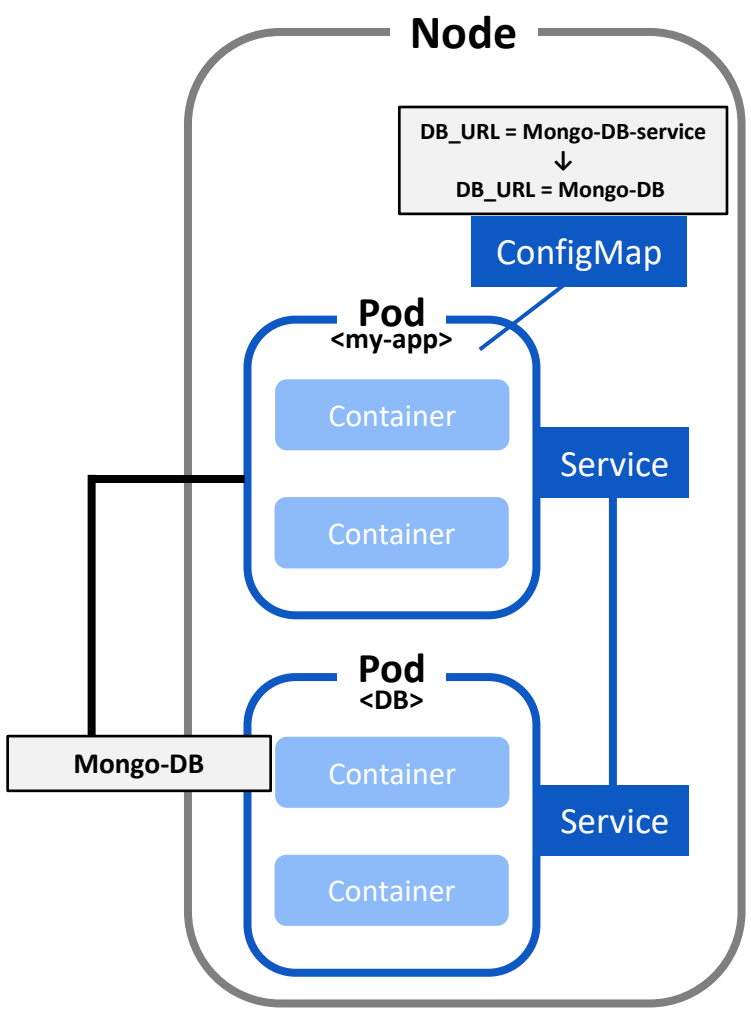


- 만일 database의 위치정보(즉 IP주소 또는 URL) 이 이를 이용하는 응용프로그램(그림상의 my-app) 코드상에 기록하여 사용하게 하면..

-> 해당 DB의 service IP주소가 변경되거나 URL 이 변경되면 아래의 과정이 필요

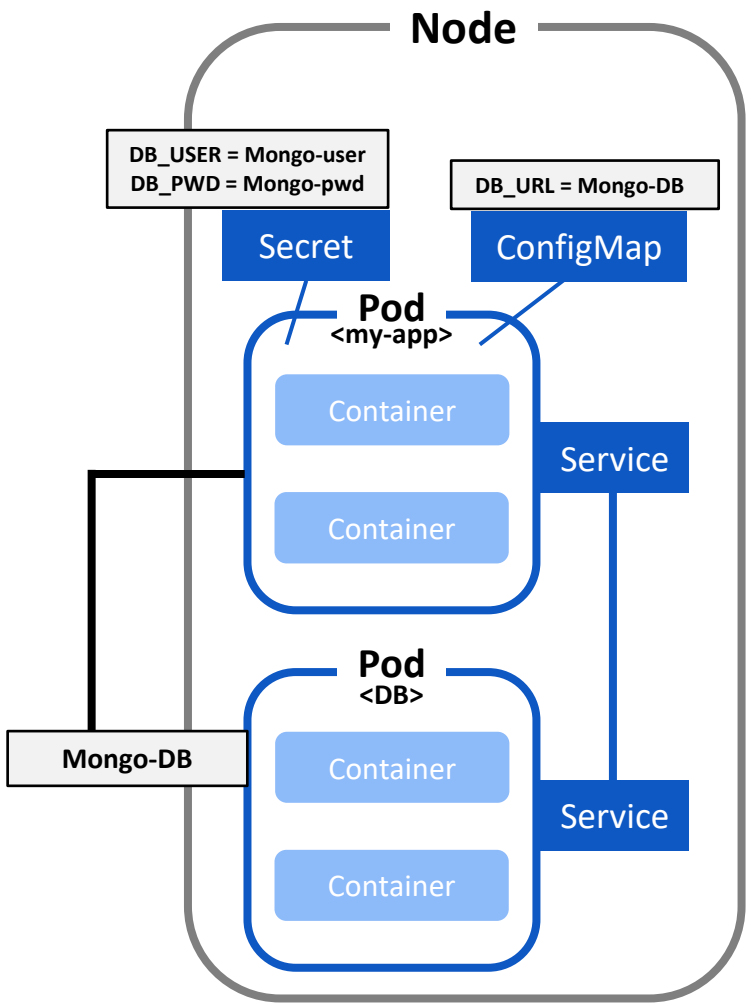
- 응용프로그램 코드상의 URL 등 수정
- 그후 **rebuild** the application with a new version.
- **push it to the repository.**
- **pull new image in Pod and restart**.

ConfigMap and Secret



- **ConfigMap:**
  - **External** configuration of application
  - **Contains configuration data (e.g.,URLs of database)**
  - Connect to Pods so that Pods get data contained in the ConfigMap.
- **If the endpoint of the Service or Service name changed, we just adjust the ConfigMap**

ConfigMap and Secret

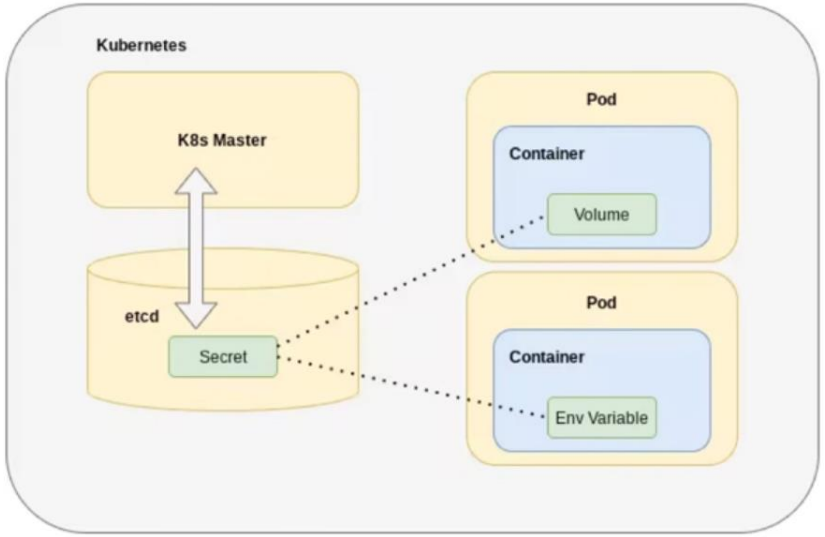


- **Secret:**
  - Used to store secret data
  - Base64 encoded
  - Contains credentials, passwords, certificates, etc.
  - **Attach the Secret to the Pod so that the Pod can see its data and read from it**
- ejecting the **data from** ConfigMap or Secret **to application pod** using environmental variables or as a properties file

Secret

- A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key.

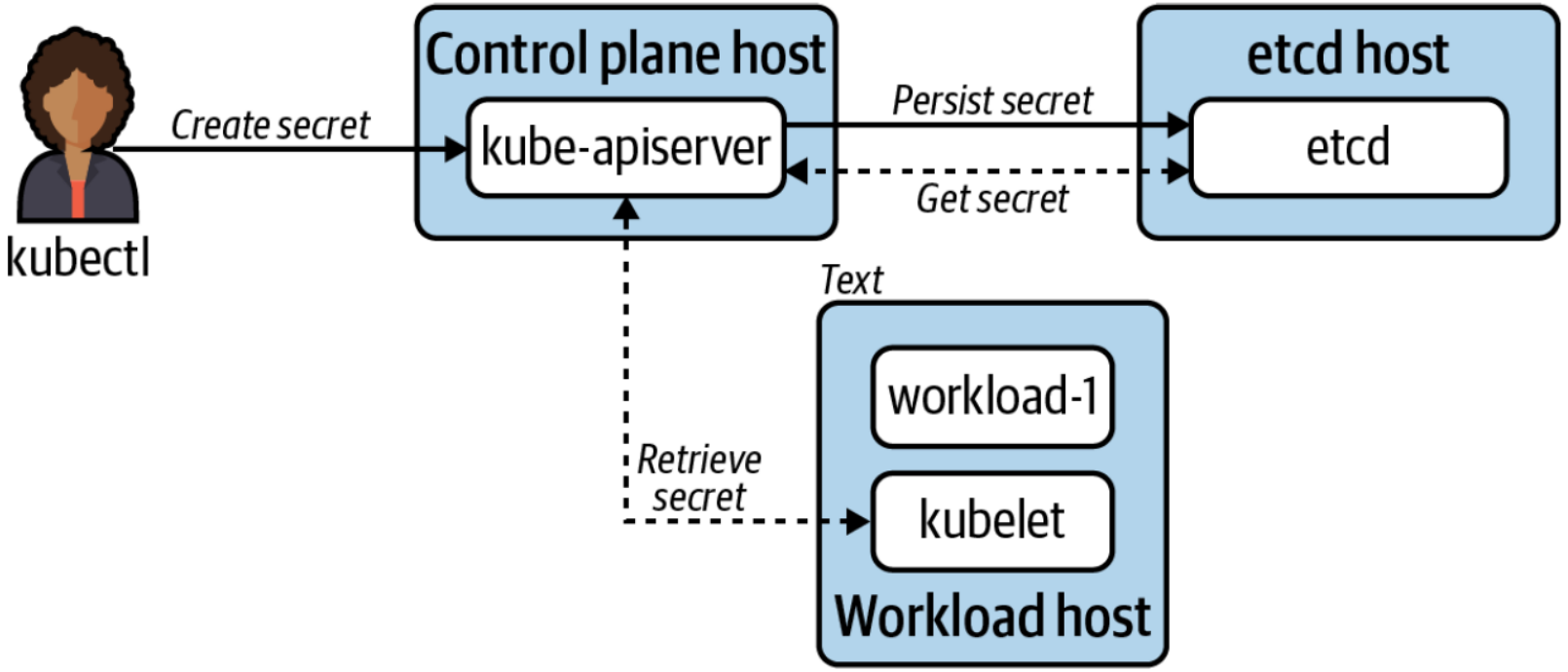
  - Such information should not be put in a Pod specification or in a container image



Kubernetes Secret

- By default, Secrets are stored unencrypted in the API server's underlying data store (etcd)
- So,
  1. Enable Other Encryption Methods for Secrets
  2. Enable or configure API access rules that restrict reading and writing the Secret

Secret Creation and Retrieve



Secret Consumption Models

- Environment variables
- Volumes
- Client API consumption

a) Environment variables

- Secret data may be injected into environment variables.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    env:
      - name: USER ❶
        valueFrom:
          secretKeyRef:
            name: mysecret ❷
            key: dbuser ❸
      - name: PASS
        valueFrom:
          secretKeyRef:
            name: mysecret
            key: dbkey
```

1. The **environment variable key** that will be available **in the application**
2. **The name of the Secret object** in Kubernetes
3. The **key in the Secret object that should be injected into the USER variable**

- **Limitations**
  - A change in a Secret object will not be reflected until the Pod is re-created.

b) Volumes

- Alternatively, secret objects may be injected via volumes.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - name: creds ❷
      readOnly: true
      mountPath: "/etc/credentials" ❸
  volumes: ❶
  - name: creds
    secret:
      secretName: mysecret
```

1. Pod-level **volumes** available **for mounting**. Name specified must be referenced in the mount
2. The volume object to mount into the container filesystem
3. **Where in the container filesystem** the mount is made available

- **Benefits**
  - **Secrets may be up-dated dynamically, without the Pod restarting**.
  - Inside app, or side-car proxy required to get the updated secrets

  (The application must watch the configuration files on disk and apply new configuration when the files change.

  (If not possible, introduce a sidecar container that watches the config files and signals the main process (with a SIGHUP, for example) when new configuration is available.)

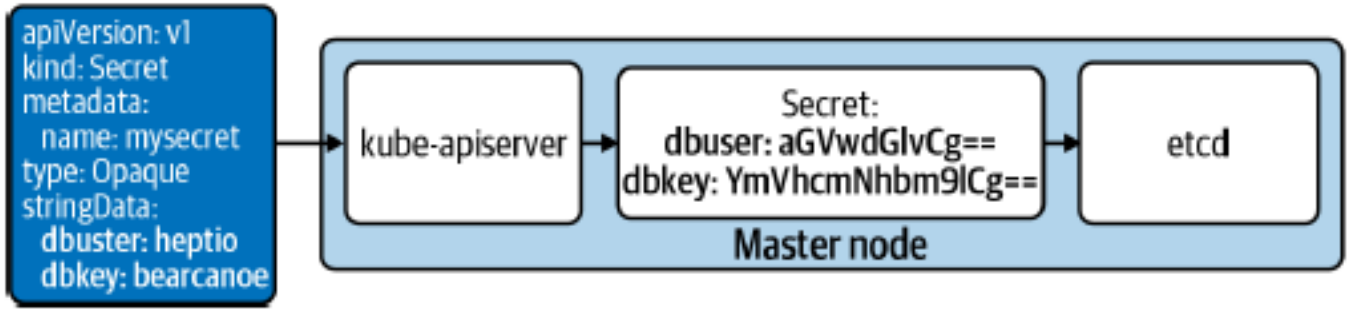
c) Client API Consumption—추천하는 방법은 아님!!!

- application to communicate with the kube-apiserver to retrieve Secret(s) and inject them into the application.
- Ex: For Java, Spring's Spring Cloud Kubernetes brings this functionality to Spring applications.

- **Limitations**
  - Special approach and not recommended.
  - Need to connect and establish a watch on the API server by separately.

Secret Data in etcd

- Like most Kubernetes objects, Secrets are stored in etcd. By default, **no encryption** is done at the Kubernetes layer before persisting Secrets to etcd.

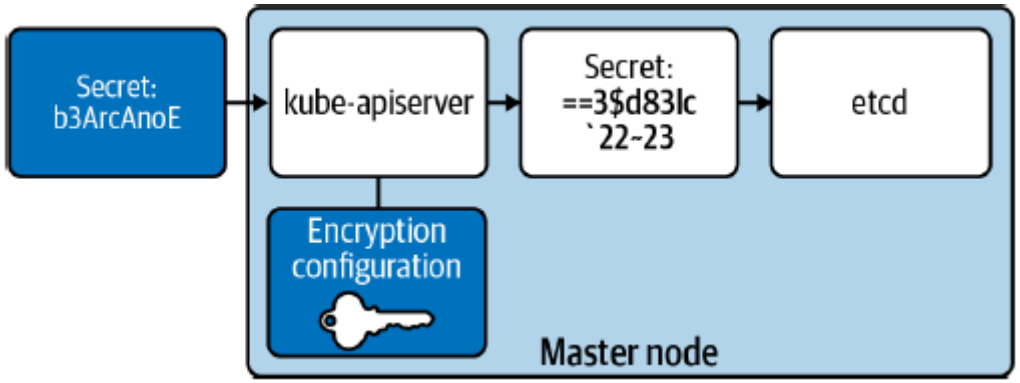


Default secret data flow in Kubernetes

- Threat Model
  - Obtaining root access to the etcd node, find the data location, and then read the secrets from the etcd database.
  - Getting root access to the API server, locating the API server and etcd certs, then impersonating the API server in communicating with etcd to read secrets.

Encryption required : ex) Static-Key Encryption

- providing the Kubernetes API server with an encryption key, which it will use to encrypt all secret objects before persisting them to etcd.



Encrypt secrets before storing them in etcd

- Even though attacker get access to etcd, they would see the encrypted data within, meaning the Secret data is not compromised.

External Provider for Secrets, … Ex) Vault

- Vault is an open source project by HashiCorp
- Secure, store and tightly control access to tokens, passwords, certificates, encryption keys for protecting secrets and other sensitive data using a UI, CLI, or HTTP API