83Kubernetes 系列（七十六）揭开 Secrets 的神秘面纱：保护敏感信息

介绍

在容器化和编排领域，Kubernetes 已成为管理和扩展应用程序的事实标准。在 Kubernetes 集群中安全运行应用程序的一个关键方面是管理敏感信息，例如密码、API 密钥和数据库凭据。这就是 Kubernetes Secrets 发挥作用的地方。在这篇博文中，我们将深入探讨 Kubernetes Secret，探索它们的用途、工作原理以及有效管理应用程序中 Secret 的最佳实践。

了解 Kubernetes Secrets

Kubernetes Secrets 提供了一种安全的方式来存储和管理应用程序所需的敏感信息。它们旨在保护 Kubernetes 集群中的静态数据和传输中的数据。应用程序和容器可以将 Secret 用作环境变量或挂载为文件，从而允许安全访问敏感信息。

创建 Secret

要创建 Kubernetes Secret，您可以使用 kubectl 命令行工具直接创建它，也可以在 Kubernetes 清单文件中定义它。Secret 通常从纯文本文件或通过命令行输入创建。然后，Kubernetes 对这些数据进行编码和加密，然后再将其存储在集群中。

Secret 的类型

Kubernetes 提供了两种主要类型的 Secret：

1. **不透明 Secret：** 这些是最常见的 Secret 类型，可以将任意敏感信息存储在键值对中。不透明 Secret 采用 base64 编码，不提供任何内置加密。应用程序负责处理解密过程。

2. **TLS Secret** ：TLS Secret 专门用于存储 SSL/TLS 证书。它们由两部分组成：私钥和相应的公共证书。这些 Secret 会自动挂载到需要 TLS 加密的 Pod 中。

管理 Pod 中的 Secret

要访问 Pod 中的 Secret，您可以使用环境变量或卷挂载。使用环境变量时，Secret 值将直接公开给在 Pod 中运行的应用程序。另一方面，卷挂载允许您将 Secret 挂载为容器文件系统中的文件。此方法通过限制对 Secret 值的直接访问来提供更高的安全性。

管理 Secret 的最佳实践

以下是使用 Kubernetes Secret 时需要考虑的一些最佳实践：

1. 将 Secret 与应用程序代码分开：避免在应用程序代码中硬编码 Secret。相反，将它们安全地存储为 Kubernetes Secrets，将敏感信息与应用程序逻辑分离。

2. **使用 RBAC 进行 Secret 访问控制** ：实施基于角色的访问控制 （RBAC） 来控制对 Secret 的访问。仅向特定用户或服务帐户授予必要的权限。

3. **定期轮换 Secret：** 定期轮换 Secret 以降低潜在的安全风险。这可以通过更新密钥的数据并将更改部署到受影响的 Pod 来完成。

4. **加密 Pod 之间的通信 ：**通过使用传输层安全性 （TLS） 证书启用加密，确保 Pod 之间的安全通信。将证书作为 TLS Secret 安全地存储和管理。

5. **使用安全的 Secret 管理解决方案：** 考虑利用外部 Secret 管理解决方案（如HashiCorp Vault）或Kubernetes原生解决方案（如Sealed Secrets）。这些工具提供额外的加密、访问控制和审核功能层。

下面是一个如何创建和使用 Kubernetes Secret 的示例。

假设您有一个需要数据库连接的应用程序，并且您希望安全地存储数据库凭据。您可以创建一个 Kubernetes Secret 来保存用户名和密码。

1. 创建一个名为 `db-credentials.yaml` 的文件，其中包含以下内容：

```
apiVersion: v1
kind: Secret
metadata:
  name: db-credentials
type: Opaque
data:
  username: <base64-encoded-username>
  password: <base64-encoded-password>
```

将 `<base64-encoded-username>` 和 `<base64-encoded-password>` 替换为实际用户名和密码的 base64 编码值。可以使用以下命令生成 base64 编码的值：

```
$ echo -n "your-username" | base64
$ echo -n "your-password" | base64
```

2. 使用 `kubectl` 命令将 Secret 应用于 Kubernetes 集群：

```
$ kubectl apply -f db-credentials.yaml
```

3. 创建 Secret 后，您可以在应用程序部署中使用它。下面是一个部署配置示例：

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-app
          image: your-app-image
          env:
            - name: DB_USERNAME
              valueFrom:
                secretKeyRef:
                  name: db-credentials
                  key: username
            - name: DB_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: db-credentials
                  key: password
```

在此示例中，我们在部署配置中定义了两个环境变量 `DB_USERNAME` 和 `DB_PASSWORD` 。这些变量的值是使用 `secretKeyRef` 字段从 `db-credentials` Secret 中检索的。

4. 应用部署配置以创建应用程序部署：

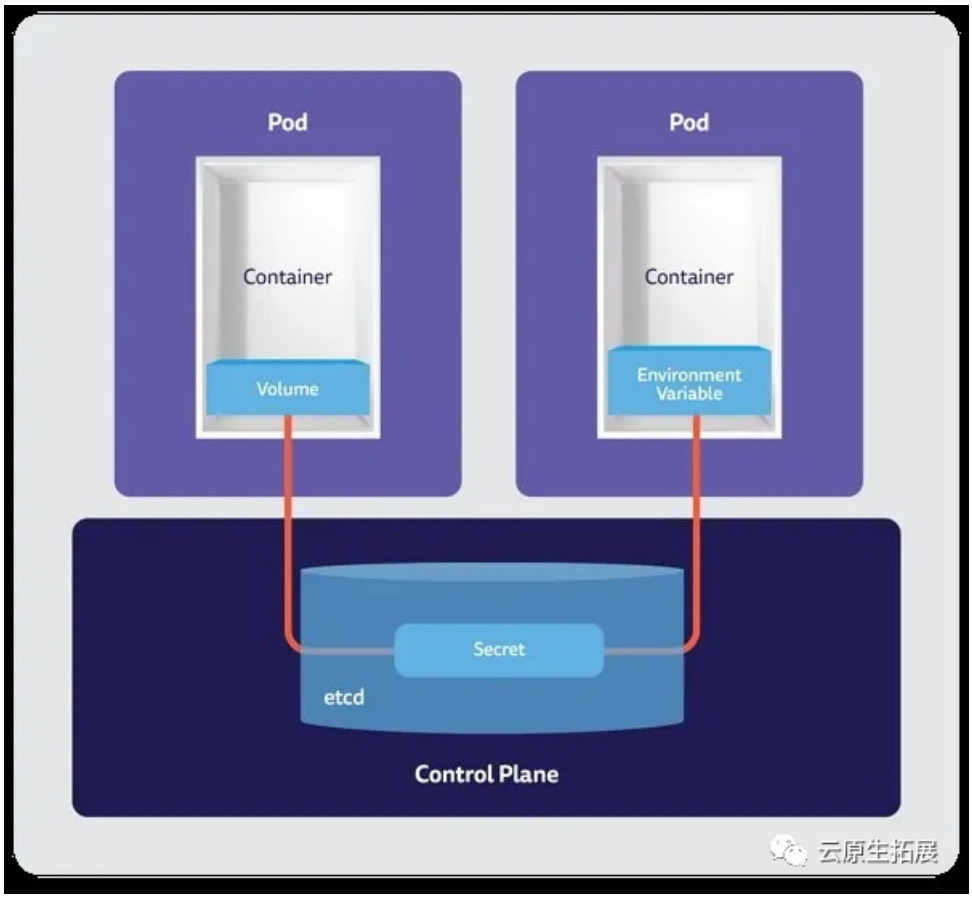
```
$ kubectl apply -f deployment.yaml
```

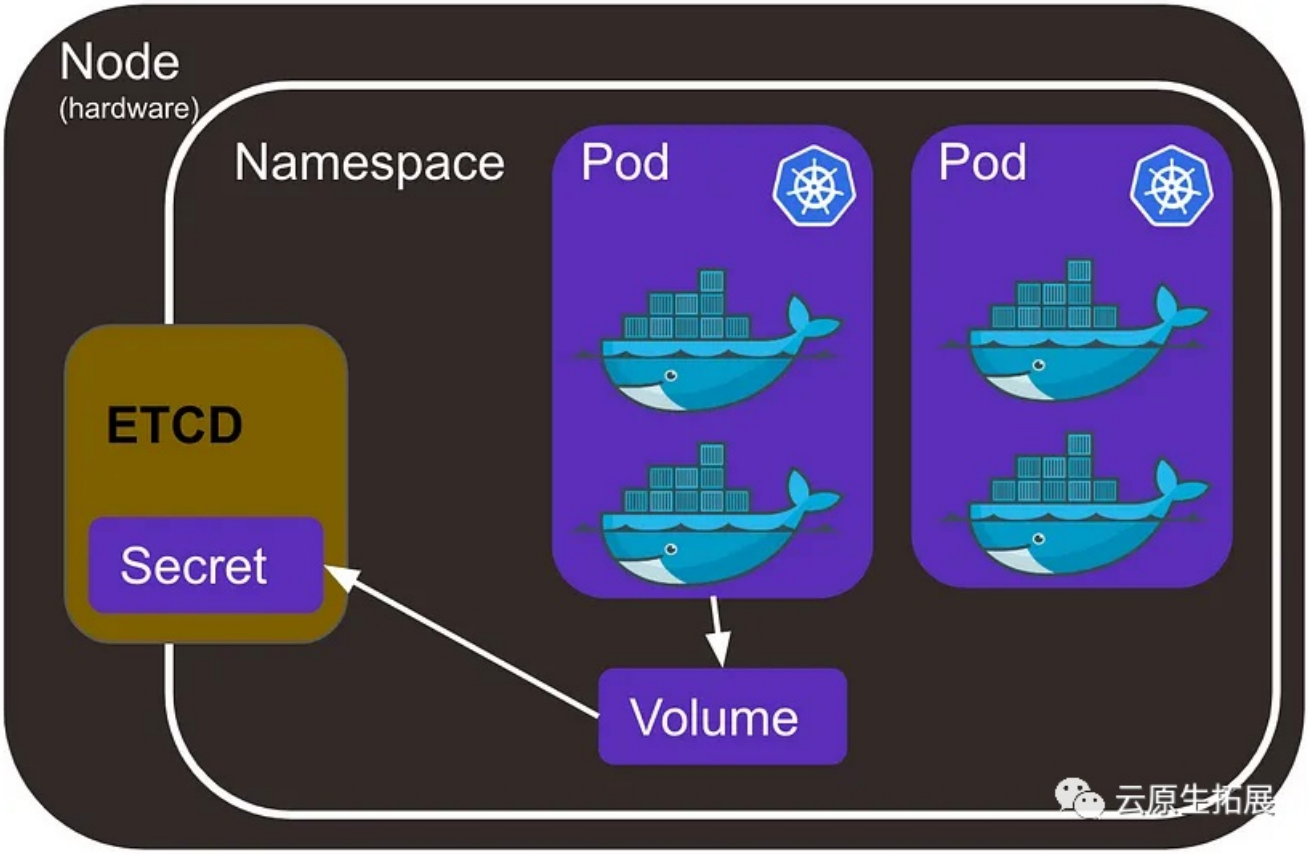
现在，您的应用程序可以通过环境变量 `DB_USERNAME` 和 `DB_PASSWORD` 安全地访问数据库凭据。实际值在运行时从 Kubernetes Secret db-credentials 中检索。

通过使用 Kubernetes Secrets，您可以安全地管理 Kubernetes 集群中的敏感信息，确保您的应用程序可以访问所需的凭据，而无需直接公开它们。

总结

Kubernetes Secrets 是保护 Kubernetes 集群中敏感信息不可或缺的一部分。通过遵循最佳实践并利用 Secret 的强大功能，您可以确保您的应用程序可以访问所需的敏感数据，而不会影响安全性。随着容器化的发展势头不断增强，理解和有效管理 Secret 对于在现代构建强大且安全的容器化应用程序至关重要。


Pod
Container
Volume
Pod
Container
Environment Variable
Secret
etcd
Control Plane
云原生拓展


Node
(hardware)
Namespace
Pod
Pod
ETCD
Secret
Volume
云原生拓展