Protecting secrets in deployment

Phạm Quang Bách

Introduction

When an application reaches production phase, be it on a serverless/managed service or a bare-metal solution, the question of managing secrets inevitably comes. Heck, it comes *during this midterm:* as I'm wrapping up the database and API server, I found myself struggling to find an adequate way to illustrate the web application without straight up uploading the data file.

In the software development world, secret comes in many flavors. Your certificate signing key, secret token generation keys, database endpoints and service account credentials - all crucial pieces of information that your entire software stack resides on. They deserve some love, and some serious considerations as to when and how to store, distribute, and load them.

Credentials should not be stored where there is little to no control of who gets access to them. This requirements completely precludes credential files from being able to be committed to version control in any form: .env files, hard-coded strings, etc.

In practice, such controls could be difficult to enforce. To combat this, Git Hooks that automatically scans and detects stray credentials before commits are recommended. The most popular tool in this class is the git-secrets hook from AWS. Crucially, the tool includes an option to scan the entire repository history for credentials; this is of paramount importance, especially when the service still utilizes keys from the early days of operation, when security standards might not be enough.

But storing a credential off-VCS in and of itself does not do very much. In production, the keys and certificates must go somewhere, and that's when encryption comes into play.

When you deploy a toy cluster on Minikube, you may have seen a flash warning that the built-in K8s secret vault is not considered secure. Why? Even though Kubernetes offer elaborate RBAC controls, the secrets volume is not encrypted; if an attacker gets control of the control node, it's K.O.

For this reason, K8s often recommends a secrets manager. These services act as key-value pairs, but they also enforce authentication before secrets can be obtained. Examples include Hashicorp Vault (most common with K8s clusters), AWS Secret Managers (for AWS users). Yes, this also means you have another set of credentials to worry about - but the advantage here is that by funneling a myriad of keys and certificates from all sorts of servers and assign their retrievability by user identity, access control is much easier - you can onboard a new engineer in a few minutes, and revoke their access similarly easily. This also enables a crucial part of DevSecOps, and that is key rotation, enabled through a central plane.