

Addressing Containerization Challenges in Google Kubernetes Engine: The GKE Auth Proxy

Robert Wilkins III

May 13, 2023

Abstract

As many teams begin to explore containerizing their applications, they often run into challenges while running their applications in the cloud. Specifically, while running containerized workloads on GKE, many users find it challenging to create GKE clusters due to security guardrails in place that prevent users from creating public-facing nodes or control planes. Further, non-transitive peering makes it difficult for users to directly communicate with GKE control planes. Nevertheless, users can create a bastion host or a GKE Auth proxy inside the project to address this challenge.

1 Introduction

In the era of cloud computing, more and more teams are exploring the idea of containerizing their applications, moving away from the on-premises model of running their applications. The trend has led teams to cloud providers, specifically Google Cloud Platform (GCP), and their default choice for running containerized workloads is Google Kubernetes Engine (GKE).

2 Challenges

Running workloads on GKE poses unique challenges. Security guardrails prevent users from creating GKE clusters with public-facing nodes or control planes. These must be private entities, possessing only internal IP addresses. This restriction, while necessary for security, has caused confusion and difficulty for many teams.

A deeper issue exists with the structure of GKE itself. When a GKE cluster is spun up, the control plane resides in a Google-managed project that is peered to the user's project. This peering is non-transitive, meaning that direct communication with the control plane is only possible within the peered project. This non-transitive peering

problem prevents users from accessing the control plane from a connected project or on-premises network.

3 Proposed Solutions

To overcome these challenges, a more sustainable solution can be proposed: the creation of a "GKE Auth Proxy". This proxy would reside on a VM in the user's project, acting as a bastion host, and would handle connection requests to the control plane, effectively creating a semblance of transitive peering allowing users to connect to the GKE control plane from a connected project or on-premises network.

4 GKE Auth Proxy Implementation

A shell script named `gke_auth_proxy.sh` was created to set up the GKE Auth Proxy. This script, once executed, prompts the user for the necessary information, such as GKE cluster name, GKE zone, GCP project ID, local and remote port numbers for the secure tunnel, and the username and external IP of the bastion host.

The script also ensures that 'gcloud' and 'kubectl' are installed on the bastion host. If they are not, the script will install them. The 'gcloud' is used to authenticate with GCP, set the project in gcloud, and get the credentials for the GKE cluster. Then, 'kubectl' is used to start a proxy on the bastion host.

A secure SSH tunnel is then set up between the user's local machine and the bastion host. This tunnel allows the user to securely run 'kubectl' commands on their local machine and have them be executed on the GKE cluster.

The script can be found in the following GitHub repository: <https://github.com/genome21/GKE-Auth-Proxy>.

The direct link to the script is: https://raw.githubusercontent.com/genome21/GKE-Auth-Proxy/main/gke_auth_proxy.sh.

Once the GKE Auth Proxy is set up, users can SSH into the bastion VM on a specific port. Upon a successful connection, they are greeted with the GKE Auth Proxy, where they can run kubectl commands to set the desired state of objects in the cluster.

5 Conclusion

While Google may address these issues in future updates, users currently face challenges in using GKE due to restrictions on public-facing nodes, control planes, and the issue of non-transitive peering. The creation of a bastion host or a GKE Auth proxy offers a viable solution to these challenges. With the GKE Auth Proxy solution, users can securely communicate with their GKE control plane and manage their clusters effectively.