|  |  |  |
| --- | --- | --- |
| Security Domain | Control & Architectural Suggestions | References |
| Encryption |  |  |
| Encryption of data at-rest | By default, GKE encrypts customer content stored at rest, including Secrets. GKE handles and manages this default encryption for the user without any additional actions. Persistent disks in GKE are already encrypted at the hardware layer by default, however, there is also the option of adding additional encryption where you, the user, can manage the encryption keys.  To encrypt persistent disks in GKE, you must use the GCP Persistent Disk CSI plugin, which lets you protect disks in GKE with a key that you manage in Cloud KMS—by creating a StorageClass referencing a key. This encryption key is used to encrypt the disks created with that StorageClass. If your organization is required to manage its own key material, the CSI plugin provides the same functionality available in traditional CMEK for persistent disks in GKE. For this, you will need to create a Cloud KMS key to use for encryption, then you can create a StorageClass on Kubernetes that specifies the Cloud KMS key *KMS\_KEY\_ID* to use to encrypt the disk.  Eg of storage class manifest: apiVersion: storage.k8s.io/v1beta1  kind: StorageClass  metadata:  name: csi-gce-pd  provisioner: pd.csi.storage.gke.io  parameters:  type: pd-standard  disk-encryption-kms-key: KMS\_KEY\_ID  **Application-layer Secrets Encryption** provides an additional layer of security for sensitive data, such as Secrets, stored in etcd. Using this functionality, you can use a key managed with Cloud KMS to encrypt data at the application layer. This protects against attackers who gain access to an offline copy of etcd.  To use Application-layer Secrets Encryption, you must first create a Cloud KMS key and give the GKE service account access to the key. The key must be in the same location as the cluster to decrease latency and to prevent cases where resources depend on services spread across multiple failure domains. Then, you can enable the feature on a new or existing cluster by specifying the key you would like to use.  Note: Default GKE cluster has encrypted persistent disks and encryption layers for secrets stored in etcd/cluster.  Migrating from a workload without Customer Managed Encryption Keys (CMEK) disks to an environment with CMEK disks will require the creation of new storage classes and Persistent Volume Claims. | Default encryption at rest on GCP:<https://cloud.google.com/security/encryption-at-rest/default-encryption>  Using your own keys to protect data on GKE: <https://cloud.google.com/blog/products/containers-kubernetes/exploring-container-security-use-your-own-keys-to-protect-your-data-on-gke>  Application Layer Secrets Encryption: <https://cloud.google.com/kubernetes-engine/docs/how-to/encrypting-secrets> |
| Encryption of data in-transit | The GKE Ingress Load Balancer is the ideal SSL termination point to ensure that data in transit is encrypted.  There are three ways to provide SSL certificates to an HTTPS load balancer on GKE:   1. **Google-managed certificates**   Google-managed SSL certificates are provisioned, deployed, renewed, and managed for your domains. Managed certificates do not support wildcard domains.   1. **Self-managed certificates shared with Google Cloud**   You can provision your own SSL certificate and create a certificate resource in your Google Cloud project. You can then list the certificate resource in an annotation on an Ingress to create an HTTP(S) load balancer that uses the certificate   1. **Self-managed certificates as Secret resources**   You can provision your own SSL certificate and create a Secret to hold it. You can then refer to the Secret in an Ingress specification to create an HTTP(S) load balancer that uses the certificate.  The 3rd option is generally the most preferred and the most secure option if you incorporate Application Layer Secrets Encryption as mentioned in the previous sub-category.  Note: The GKE LB supports multiple TLS certificates | SSL certs on GCP: <https://cloud.google.com/load-balancing/docs/ssl-certificates>  Using Multiple SSL certs with GKE Ingress: <https://cloud.google.com/kubernetes-engine/docs/how-to/ingress-multi-ssl> |
| Encryption Key Management | If you want to control and manage the encryption on GCP yourself, you can use key encryption keys. Key encryption keys do not directly encrypt your data but are used to encrypt the data encryption keys that encrypt your data.  You have two options for key encryption keys in GCP:   1. Use Cloud Key Management Service to create and manage key encryption keys. 2. Create and manage your own key encryption keys, also known as customer-supplied encryption keys (CSEK).   Key Rotation should occur as often as organization security guidelines dictate.  Key rotations can be automated like so:  gcloud kms keys update **key-name** \  --location **location** \  --keyring **key-ring-name** \  --rotation-period **rotation-period** \  --next-rotation-time **next-rotation-time** | Protecting resources with KMS keys: <https://cloud.google.com/compute/docs/disks/customer-managed-encryption>  Using KMS: <https://cloud.google.com/kms/docs/quickstart>  Configuring automatic key rotation: <https://cloud.google.com/kms/docs/rotating-keys#kms-enable-key-version-cli> |
| Infrastructure |  |  |
| Isolation | **Layers of isolation in Kubernetes:**  Kubernetes has several nested layers, each of which provides some level of isolation and security.  **Container (not specific to Kubernetes):** A container provides basic management of resources, but does not isolate identity or the network, and can suffer from a noisy neighbor on the node for resources that are not isolated by cgroups. It provides some security isolation, but only provides a single layer, compared to our desired double layer.  **Pod**: A pod is a collection of containers. A pod isolates a few more resources than a container, including the network. It does so with micro-segmentation using Kubernetes [Network Policy](https://cloud.google.com/kubernetes-engine/docs/how-to/network-policy), which dictates which pods can speak to one another. At the moment, a pod does not have a unique identity, but the Kubernetes community has made [proposals](https://github.com/kubernetes/community/blob/master/contributors/design-proposals/auth/bound-service-account-tokens.md) to provide this. A pod still suffers from noisy neighbors on the same host.  **Node:** This is a machine, either physical or virtual. A node includes a collection of pods, and has a superset of the privileges of those pods. A node leverages a hypervisor or hardware for isolation, including for its resources. Modern Kubernetes nodes run with distinct identities, and are authorized only to access the resources required by pods that are scheduled to the node. There can still be attacks at this level, such as convincing the scheduler to assign sensitive workloads to the node. You can use [firewall rules](https://cloud.google.com/vpc/docs/firewalls) to restrict network traffic to the node.  **Cluster**: A cluster is a collection of nodes and a control plane. This is a management layer for your containers. Clusters offer stronger network isolation with [per-cluster DNS](https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/).  **Project:** A GCP project is a collection of resources, including Kubernetes Engine clusters. A project provides all of the above, plus some additional controls that are GCP-specific, like [project-level IAM for Kubernetes Engine](https://cloud.google.com/kubernetes-engine/docs/how-to/iam-integration) and [org policies](https://cloud.google.com/resource-manager/docs/organization-policy/overview). Resource names, and other resource metadata, are visible up to this layer. | Security Isolation on GKE:  <https://cloud.google.com/blog/products/gcp/exploring-container-security-isolation-at-different-layers-of-the-kubernetes-stack>  Network Policy for inter pod communication: <https://cloud.google.com/kubernetes-engine/docs/how-to/network-policy> |
| Network Isolation | There are three different flavors of network modes for clusters that can deliver network level protection:   * **Public endpoint access disabled, master authorized networks enabled (recommended):** This is the most secure option as it prevents all internet access to both masters and nodes, except for certain whitelisted IPs and Google Service IP ranges. * **Public endpoint access enabled, master authorized networks enabled:** This option provides restricted access to the master from source IP addresses that you define. This is a good choice if you don't have existing VPN infrastructure or have remote users or branch offices that connect over the public internet instead of the corporate VPN and Cloud Interconnect or Cloud VPN. * **Public endpoint access enabled, master authorized networks disabled (least secure):** This is the default and allows anyone on the internet to make network connections to the control plane.   The most secure option is utilizing private GKE cluster with no public endpoints (on either master or nodes), and whitelisting Kube API server access in the master to only select machines.This ensures the control plane is reachable by:   * The whitelisted CIDRs in master authorized networks (bastion host or other admin machines). * Nodes within your cluster's VPC. * Google's internal production jobs that manage your master.   That corresponds to the following gcloud flags at cluster creation time:   * --enable-ip-alias * --enable-private-nodes * --enable-master-authorized-networks | Restricting Network Access: <https://cloud.google.com/kubernetes-engine/docs/how-to/hardening-your-cluster#restrict_network_access_to_the_control_plane_and_nodes>  Creating a private cluster: <https://cloud.google.com/kubernetes-engine/docs/how-to/private-clusters>  Master Authorized Networks: <https://cloud.google.com/kubernetes-engine/docs/how-to/authorized-networks> |
| GKE Master Network | An important thing to note is that the entire GKE control plane is managed by Google.  Essentially, the master node is managed in a separate Google managed project, in a separate Google managed VPC that is automatically peered with the VPC in which you deploy your cluster upon cluster creation time.  This means that if you need to communicate to the Kube master API in a private cluster, it would require establishing a VPN tunnel and exporting the custom routes. |  |
| IAM |  |  |