Best practices for GKE networking

AUTOPILOT (/KUBERNETES-ENGINE/DOCS/CONCEPTS/AUTOPILOT-OVERVIEW)

STANDARD (/KUBERNETES-ENGINE/DOCS/CONCEPTS/TYPES-OF-CLUSTERS)

This document outlines the best practices for configuring networking options for Google Kubernetes Engine (GKE) clusters. It is intended to be an architecture planning guide for cloud architects and network engineers with cluster configuration recommendations that are applicable to most GKE clusters. Before you create your GKE clusters, we recommend that you review all the sections in this document to understand the networking options that GKE supports and their implications.

The networking options that you choose impact the architecture of your GKE clusters. Some of these options cannot be changed once configured without recreating the cluster.

This document is not intended to introduce Kubernetes networking concepts or terminology and assumes that you already have some level of general networking concepts and Kubernetes networking knowledge. For more information, see the GKE network overview (/kubernetes-engine/docs/concepts/network-overview).

While reviewing this document consider the following:

- How you plan to expose workloads internally to your Virtual Private Cloud (VPC) network, other workloads in the cluster, other GKE clusters, or externally to the internet.
- How you plan to scale your workloads.
- What types of Google services you want to consume.

For a summarized checklist of all the best practices, see the Checklist summary (#checklist).

VPC design

When designing your VPC networks, follow best practices for VPC design (/solutions/best-practices-vpc-design).

The following section provides some GKE-specific recommendations for VPC network design.

Best practices:

Use VPC-native clusters (#vpc-native-clusters).
Use Shared VPC networks (#sharedvpc_networks).

Use VPC-native clusters

We recommend that you use VPC-native clusters (/kubernetes-engine/docs/how-to/alias-ips). VPC-native clusters use alias IP address ranges (/vpc/docs/alias-ip) on GKE nodes and are required for private GKE clusters and for creating clusters on Shared VPCs (/vpc/docs/shared-vpc), and have many other benefits (/kubernetes-engine/docs/concepts/alias-ips#benefits). For clusters created in the Autopilot (/kubernetes-engine/docs/concepts/autopilot-overview) mode, VPC-native mode is always on and cannot be turned off.

VPC-native clusters scale more easily than routes-based clusters without consuming Google Cloud routes and so are less susceptible to hitting routing limits.

The advantages (/kubernetes-engine/docs/concepts/alias-ips#benefits) to using VPC-native clusters go hand-in-hand with alias IP support (/vpc/docs/alias-ip#key_benefits_of_alias_ip_ranges). For example, network endpoint groups (/load-balancing/docs/negs) (NEGs) can only be used with secondary IP addresses, so they are only supported on VPC-native clusters.

Use Shared VPC networks

GKE clusters require careful IP address planning (/kubernetes-engine/docs/concepts/alias-ips#defaults_limits). Most organizations tend to have a centralized management structure with a network administration team who can allocate IP address space for clusters and a platform administrator for operating the clusters. This type of organization structure works well with Google Cloud's Shared VPC network architecture. In the Shared VPC network architecture, a network administrator can create subnets and share them with VPCs. You can create GKE clusters in a service project (/kubernetes-engine/docs/how-to/cluster-shared-vpc#overview) and use the subnets shared from the Shared VPC on the host project (/kubernetes-engine/docs/how-to/cluster-shared-vpc#overview). The IP address component stays in the host project, and your other cluster components live in the service project.

In general, a Shared VPC network is a frequently used architecture that is suitable for most organizations with a centralized management team. We recommend using Shared VPC networks to create the subnets for your GKE clusters and to avoid IP address conflicts across your organization. You might also want to use Shared VPCs for governance of operational functions. For example, you can have a network team that works only on network components and reliability, and another team that works on GKE.

IP address management strategies

All Kubernetes clusters, including GKE clusters, require a unique IP address for every Pod. To learn more, see the GKE networking model (/kubernetes-engine/docs/concepts/gke-compare-network-models#gke-networking-model). In GKE, all these IP addresses are routable throughout the VPC network. Therefore, IP address planning is necessary because addresses cannot overlap with internal IP address space used on-premises or in other connected environments. The following sections suggest strategies for IP address management with GKE.

Best practices:

Plan the required IP address allotment

Private clusters are recommended and are further discussed in the Network security (#network-security) section. In the context of private clusters, only VPC-native clusters are supported and require the following IP address ranges:

- Control plane IP address range: use a /28 subnet within the IP address private ranges included on the RFC 1918 (https://datatracker.ietf.org/doc/html/rfc1918). You must ensure this subnet doesn't overlap any other classless inter-domain routing (CIDR) in the VPC network.
- Node subnet: the subnet with the primary IP address range that you want to allocate for all the nodes in your cluster. Services with the type `LoadBalancer` that use the `cloud.google.com/load-balancer-type: "Internal"` annotation also use this subnet by default. You can also use a dedicated subnet for internal load balancers (/kubernetes-engine/docs/how-to/internal-load-balancing).
- Pod IP address range: the IP range that you allocate for all Pods in your cluster. GKE provisions this range as an alias of the subnet. For more information, see IP address ranges for VPC-

native clusters (/kubernetes-engine/docs/concepts/alias-ips#cluster_sizing)

- Service IP address range: the IP address range that you allocate for all Services in your cluster. GKE provisions this range as an alias of the subnet.

For private clusters, you must define a node subnet, a Pod IP address range, and a Service IP address range. If you want to use IP address space more efficiently, see Reduce internal IP address usage in GKE (/kubernetes-engine/docs/concepts/gke-ip-address-mgmt-strategies#reduce-private-ip-address-usage-in-gke).

The control plane IP address range is dedicated to the GKE-managed control plane that resides in a Google-managed tenant project peered with your VPC. This IP address range shouldn't overlap with any IP addresses in your VPC peering group because GKE imports this route into your project. This means that if you have any routes to the same CIDR in your project, you might experience routing issues.

When creating a cluster, the subnet has a primary range for the nodes of the cluster and it should exist prior to cluster creation. The subnet should accommodate the maximum number of nodes that you expect (/kubernetes-engine/docs/concepts/alias-ips#cluster_sizing_primary_range) in the cluster and the internal load balancer IP addresses across the cluster using the subnet. You can use the cluster autoscaler (/kubernetes-engine/docs/concepts/cluster-autoscaler) to limit the maximum number of nodes.

The Pod and service IP address ranges are represented as distinct secondary ranges of your subnet, implemented as alias IP addresses in VPC-native clusters.

Choose wide enough IP address ranges so that you can accommodate all nodes, Pods, and Services for the cluster (/kubernetes-engine/docs/concepts/alias-ips#defaults_limits).

Consider the following limitations:

- You can expand primary IP address ranges (/vpc/docs/create-modify-vpc-networks#expand-subnet) but you cannot shrink them. These IP address ranges cannot be discontiguous.
- You can expand the Pod range (/kubernetes-engine/docs/how-to/multi-pod-cidr) by appending additional Pod ranges to the cluster or creating new node pools with other secondary Pod ranges.
- The secondary IP address range for Services cannot be expanded or changed over the life of the cluster.
- Review the limitations for the secondary IP address range for Pods and Services (/kubernetes-engine/docs/concepts/alias-ips#node_limiters).

Use more than private RFC 1918 IP addresses

For some environments, RFC 1918 space in large contiguous CIDR blocks might already be allocated in an organization. You can use non-RFC 1918 space (/kubernetes-engine/docs/how-to/alias-ips#enable_reserved_ip_ranges) for additional CIDRs for GKE clusters, if they don't overlap with Google-owned public IP addresses. We recommend using the 100.64.0.0/10 part of the RFC (https://tools.ietf.org/html/rfc6598) address space because Class E (https://tools.ietf.org/html/rfc5735) address space can present interoperability issues with on-premises hardware. You can use privately reused public IPs (PUPI (/kubernetes-engine/docs/how-to/alias-ips#enable_pupis)).

When using privately reused public IP addresses (/kubernetes-engine/docs/archive/configuring-privately-used-public-ips-for-GKE), use with caution and consider controlling route advertisements in on-premises networks to the internet when choosing this option.

You shouldn't use source network address translation (SNAT) (https://cloud.google.com/sdk/gcloud/reference/beta/container/clusters/create#--disable-default-snat) in a cluster with Pod-to-Pod and Pod-to-Service traffic. This breaks the Kubernetes networking model (https://kubernetes.io/docs/concepts/services-networking/).

Kubernetes assumes that all non-RFC 1918 IP addresses are privately reused public IP addresses and uses SNAT for all traffic originating from these addresses.

If you are using a non-RFC 1918 IP address for your GKE cluster, for Standard clusters, you will need to either explicitly disable SNAT (/sdk/gcloud/reference/beta/container/clusters/create#--disable-default-snat) or configure the configure the IP masquerade agent (/kubernetes-engine/docs/concepts/ip-masquerade-agent#how_ipmasq_works) agent to exclude your cluster's Pod IP addresses and the secondary IP address ranges for Services from SNAT. For Autopilot clusters, this doesn't require any extra steps.

Use custom subnet mode

When you set up the network, you also select the subnet mode: `auto` (default) or `custom` (recommended). The `auto` mode leaves the subnet allocation up to Google and is a good option to get started without IP address planning. However, we recommend selecting the `custom` mode because this mode lets you choose IP address ranges that won't overlap other ranges in your environment. If you are using a Shared VPC, either an organizational administrator or network administrator can select this mode.

The following example creates a network called `my-net-1` with custom subnet mode:

```
gcloud compute networks create my-net-1 --subnet-mode custom
```

Plan Pod density per node

By default, Standard clusters reserve a /24 range for every node out of the Pod address space in the subnet and allows for up to 110 Pods per node (/kubernetes-engine/docs/concepts/alias-ips#cluster_sizing_secondary_range_pods). However, you can configure a Standard cluster to support up to 256 Pods per node, with a /23 range reserved for every node. Depending on the size of your nodes and the application profile of your Pods, you might run considerably fewer Pods on each node.

If you don't expect to run more than 64 Pods per node, we recommend that you adjust the maximum Pods per node (/kubernetes-engine/docs/how-to/flexible-pod-cidr) to preserve IP address space in your Pod subnet.

If you expect to run more than the default 110 Pods per node, you can increase the maximum Pods per node up to 256, with /23 reserved for every node. With this type of high Pod density configuration, we recommend using instances with 16 or more CPU cores to ensure the scalability and performance of your cluster.

For Autopilot (/kubernetes-engine/docs/concepts/autopilot-overview) clusters, the maximum number of Pods per node is set to 32, reserving a /26 range for every node. This setting is non-configurable in Autopilot clusters.

Avoid overlaps with IP addresses used in other environments

You can connect your VPC network to an on-premises environment or other cloud service providers through Cloud VPN (/network-connectivity/docs/vpn) or Cloud Interconnect (/network-connectivity/docs/interconnect). These environments can share routes, making the on-premises IP address management scheme important in IP address planning for GKE. We recommend making sure that the IP addresses don't overlap with the IP addresses used in other environments.

Create a load balancer subnet

Create a separate load balancer subnet (/kubernetes-engine/docs/how-to/internal-load-balancing) to expose services with internal TCP/UDP load balancing. If a separate load balancer subnet is not used, these services are exposed by using an IP address from the node subnet, which can lead to the use of all allocated space in that subnet earlier than expected and can stop you from scaling your GKE cluster to the expected number of nodes.

Using a separate load balancer subnet also means that you can filter traffic to and from the GKE nodes separately to services that are exposed by internal TCP/UDP load balancing, which lets you set stricter security boundaries.

Reserve enough IP address space for cluster autoscaler

Note: For Autopilot (/kubernetes-engine/docs/concepts/autopilot-overview) clusters, Google dynamically provisions resources based on your Pod specification. The recommendations in this section can still be applied to Autopilot clusters.

You can use the cluster autoscaler (/kubernetes-engine/docs/concepts/cluster-autoscaler) to dynamically add and remove nodes in the cluster so that you can control costs and improve utilization. However, when you are using the cluster autoscaler, make sure that your IP address planning accounts for the maximum size of all node pools. Each new node requires its own node IP address as well as its own allocatable set of Pod IP addresses based on the configured Pods per node. The number of Pods per node can be configured differently than what is configured at the cluster level. You cannot change the number of Pods per node after you create the cluster or node pool. You should consider your workload types and assign them to distinct node pools for optimal IP address allocation.

Consider using node auto-provisioning (/kubernetes-engine/docs/how-to/node-auto-provisioning), with the cluster autoscaler, particularly if you're using VPC-native clusters. For more information, see Node limiting ranges (/kubernetes-engine/docs/concepts/alias-ips#node_limiters).

Share IP addresses across clusters

You might need to share IP addresses across clusters if you have a centralized team that is managing the infrastructure for clusters. To share IP addresses across GKE clusters, see Sharing IP address ranges across GKE clusters (/kubernetes-engine/docs/concepts/alias-ips#sharing_ip_address). You can reduce IP exhaustion by creating three ranges, for Pods, Services and nodes, and reusing or sharing them, especially in a Shared VPC model. This setup can also make it easier for network administrators to manage IP addresses by not requiring them to create specific subnets for each cluster.

Consider the following:

- As a best practice, use separate subnets and IP address ranges for all clusters.
- You can share the secondary Pod IP address range, but it is not recommended because one cluster might use all of the IP addresses.

- You can share secondary Service IP address ranges, but this feature does not work with VPC-scope Cloud DNS for GKE (/kubernetes-engine/docs/how-to/cloud-dns#vpc_scope_dns).

If you run out of IP addresses, you can create additional Pod IP address ranges using discontiguous multi-Pod CIDR (/kubernetes-engine/docs/how-to/multi-pod-cidr).

Share IP addresses for internal LoadBalancer Services

You can share a single IP address with up to 50 backends using different ports. This lets you reduce the number of IP addresses you need for internal LoadBalancer Services.

For more information, see Shared IP (/kubernetes-engine/docs/how-to/internal-load-balancing#shared_VIP).

Network security options

A few key recommendations are outlined in this section for cluster isolation. Network security for GKE clusters is a shared responsibility between Google and your cluster administrator(s).

Best practices:

Use GKE Dataplane V2. (#dataplane-v2)
Choose a private cluster type (#private-cluster-type).
Minimize the cluster control plane exposure (#minimize-control-plane-exposure).
Authorize access to the control plane (#authorize-cp-access).
Allow control plane connectivity (#allow-cp-connectivity).
Deploy proxies for control plane access from peered networks (#deploy-proxies).
Restrict cluster traffic using network policies (#restrict-traffic-network-pols).
Enable Google Cloud Armor security policies for Ingress (#enable-security-policies).
Use Identity-Aware Proxy to provide authentication for applications with IAM users (#use-iap).
Use organization policy constraints to further enhance security (#use-org-policy-constraints).

Use GKE Dataplane V2

GKE Dataplane V2 (/kubernetes-engine/docs/how-to/dataplane-v2) is based on eBPF (https://ebpf.io/) and provides an integrated network security and visibility experience. When you create a cluster using GKE Dataplane V2 you don't need to explicitly enable network policies because GKE Dataplane V2 manages service routing, network policy enforcement and logging. Enable the new dataplane with the Google Cloud CLI `--enable-dataplane-v2` option when creating a cluster. After network

policies are configured, a default `NetworkLogging` CRD object can be configured (/kubernetes-engine/docs/how-to/network-policy-logging#configuring_network_policy_logging) to log allowed and denied network connections. We recommend creating clusters with GKE Dataplane V2 to take full advantage of the built-in features such as network policy logging (/kubernetes-engine/docs/how-to/network-policy-logging).

Choose a private cluster type

Public clusters have both private and public IP addresses on nodes and only a public endpoint for control plane nodes. Private clusters provide more isolation by only having internal IP addresses on nodes, and having private or public endpoints for control plane nodes (which can be further isolated and is discussed in the Minimize the cluster control plane exposure (#minimize-control-plane-exposure) section). In private clusters, you can still access Google APIs with Private Google Access (#use-private-google-access). We recommend choosing private clusters.

In a private cluster, Pods are isolated from inbound and outbound communication (the cluster perimeter). You can control these directional flows by exposing services by using load balancing and Cloud NAT (/nat/docs), discussed in the cluster connectivity (#scaling) section in this document. The following diagram shows this kind of setup:

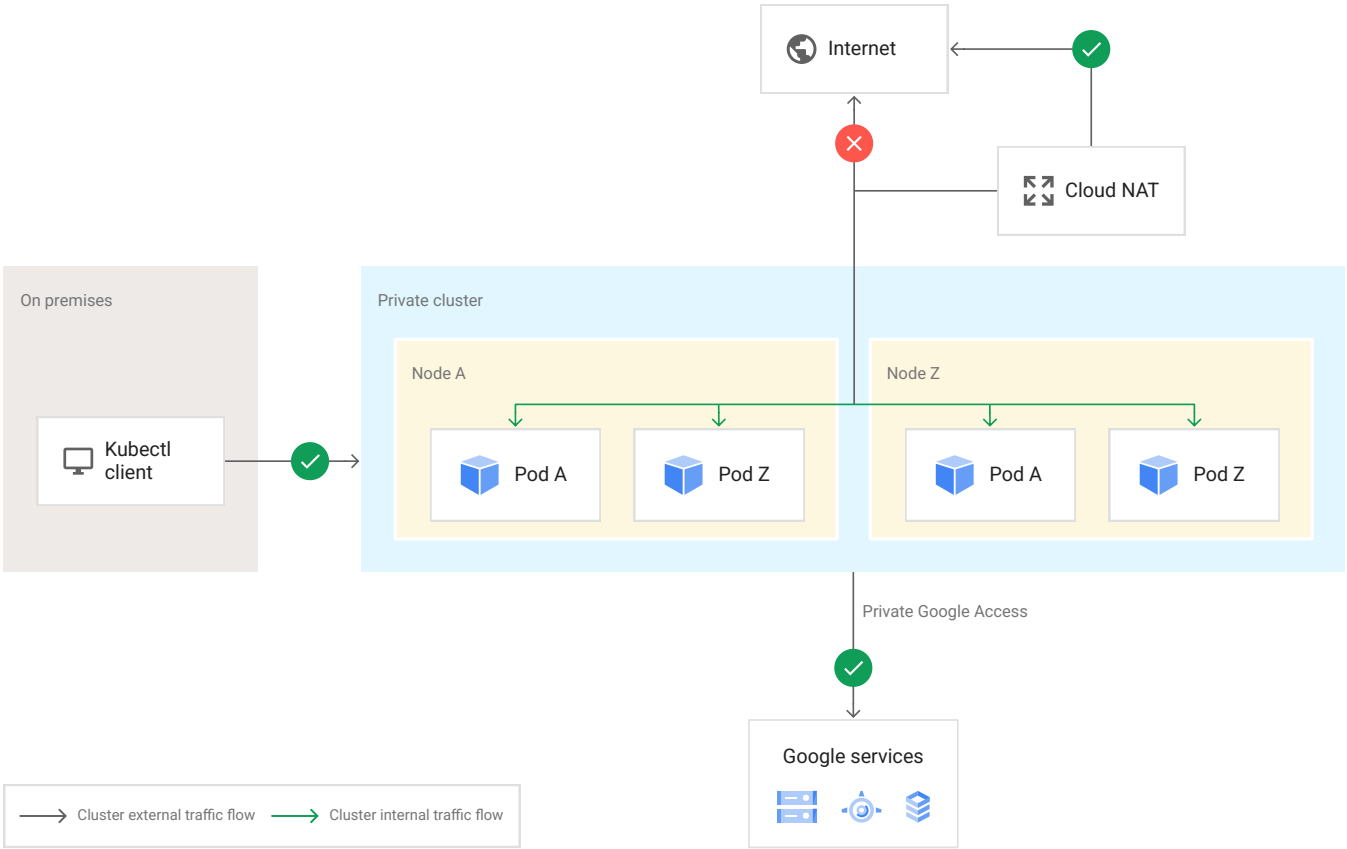

Diagram 1: *Private cluster communication*

This diagram shows how a private cluster can communicate. On-premises clients can connect to the cluster with the kubectl client. Access to Google Services is provided through Private Google Access, and communication to the internet is available only by using Cloud NAT.

For more information, review the requirements, restrictions, and limitations of private clusters (/kubernetes-engine/docs/how-to/private-clusters#req_res_lim).

Minimize the cluster control plane exposure

In a private cluster, the GKE API server can be exposed as a public or a private endpoint. You can decide which endpoint to use when you create the cluster. You can control access with authorized networks (/kubernetes-engine/docs/how-to/authorized-networks), where both the public and private endpoints default to allowing all communication between the Pod and the node IP addresses in the cluster. To enable a private endpoint when you create a cluster, use the `--enable-private-endpoint` (/sdk/gcloud/reference/container/clusters/create#--enable-private-endpoint) flag.

Authorize access to the control plane

Authorized networks can help dictate which IP address subnets are able to access the GKE control plane nodes. After enabling these networks, you can restrict access to specific source IP address ranges. If the public endpoint is disabled, these source IP address ranges should be private. If a public endpoint is enabled, you can allow public or internal IP address ranges. Configure custom route advertisements (/network-connectivity/docs/router/how-to/advertising-overview) to allow the private endpoint of the cluster control plane to be reachable from an on-premises environment. You can make the private GKE API endpoint be globally reachable by using the `--enable-master-global-access` (/kubernetes-engine/docs/how-to/authorized-networks#create_cluster) option when you create a cluster.

Important: Even if you disable access to the public endpoint, Google can use the control plane's public endpoint for cluster management purposes, such as scheduled maintenance (/kubernetes-engine/docs/scheduled-maintenance) and automatic upgrades (/kubernetes-engine/versioning-and-upgrades#automatic_cp_upgrades).

The following diagram shows typical control plane connectivity using authorized networks:

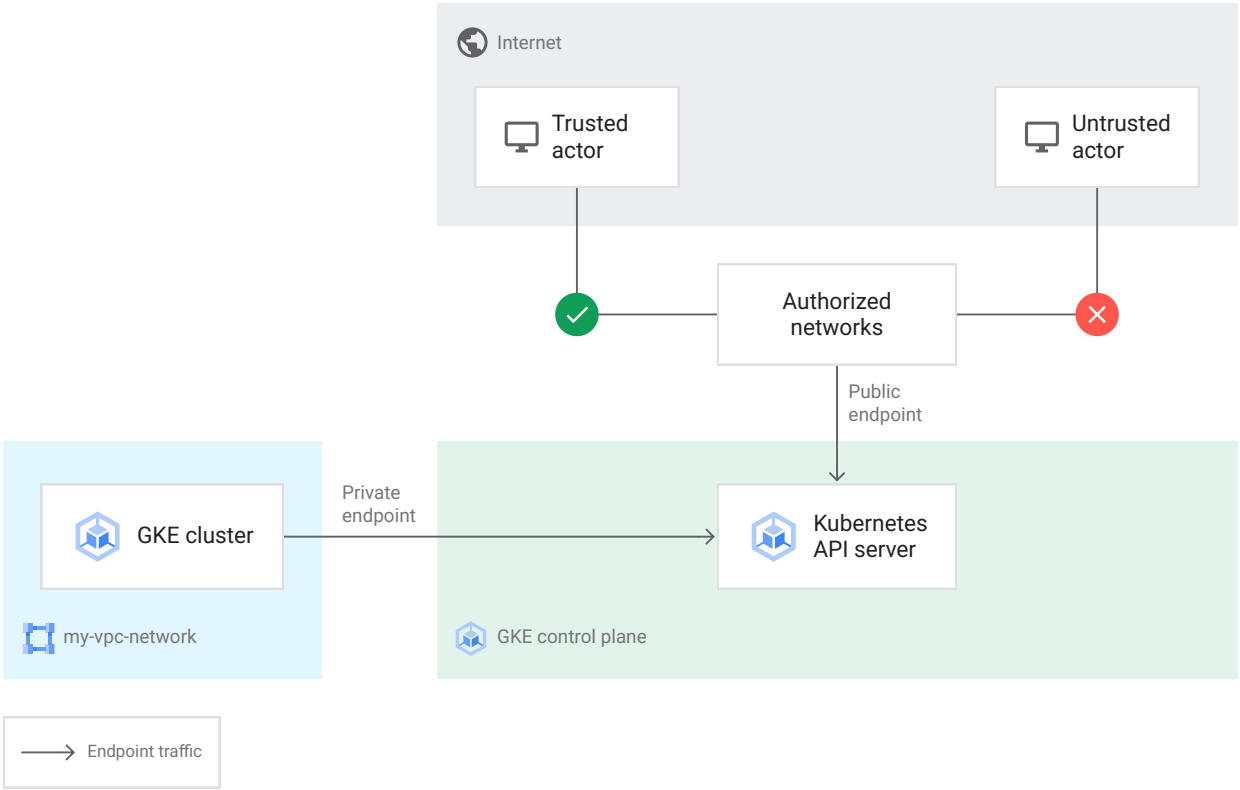

Diagram 2: *Control plane connectivity using authorized networks*

This diagram shows trusted users being able to communicate with the GKE control plane through the public endpoint as they are part of authorized networks, while access from untrusted actors is blocked. Communication to and from the GKE cluster happens through the private endpoint of the control plane.

Allow control plane connectivity

Certain system Pods on every worker node will need to reach services such as the Kubernetes API server (`kube-apiserver`), Google APIs, or the metadata server. The `kube-apiserver`also needs to communicate with some system Pods, such as `event-exporter` specifically. This communication is allowed by default. If you deploy VPC firewall rules within the projects (more details in the Restrict cluster traffic section (#restrict-traffic-network-pols)), ensure those Pods can keep communicating to the `kube-apiserver` as well as to Google APIs.

Deploy proxies for control plane access from peered networks

Access to the control plane for private GKE clusters is through VPC Network Peering (/vpc/docs/vpc-peering). VPC Network Peering is non-transitive, therefore you cannot access the cluster's control plane from another peered network.

If you want direct access from another peered network or from on-premises when using a hub-and-spoke architecture (/solutions/architecture-centralized-network-appliances-on-google-cloud), deploy proxies for control plane traffic.

Restrict cluster traffic using network policies

Multiple levels of network security are possible for cluster workloads that can be combined: VPC firewall rules, Hierarchical firewall policies, and Kubernetes network policies. VPC firewall rules and Hierarchical firewall policies apply at the virtual machine (VM) level, that is the worker nodes on which the Pods of the GKE cluster reside. Kubernetes network policies apply at the Pod-level to enforce Pod to Pod traffic paths.

If you implement VPC firewalls, it can break the default, required control plane communication—for example the kubelet communication with the control plane. GKE creates required firewall rules (/kubernetes-engine/docs/concepts/firewall-rules) by default, but they can be overwritten. Some deployments might require the control plane to reach the cluster on the service. You can use VPC firewalls to configure an ingress policy that makes the service accessible.

GKE network policies are configured through the Kubernetes Network Policy API to enforce a cluster's Pod communication. You can enable network policies when you create a cluster by using the `gcloud container clusters create` option `--enable-network-policy`. To restrict traffic using network policies, you can follow the Anthos restricting traffic blueprint implementation guide (https://github.com/GoogleCloudPlatform/anthos-security-blueprints/tree/master/restricting-traffic).

Enable Google Cloud Armor security policies for Ingress

Using Google Cloud Armor security policies (/armor/docs/security-policy-overview), you can protect applications that are using external Application Load Balancers (/load-balancing/docs/https) from DDoS attacks and other web-based attacks by blocking such traffic at the network edge. In GKE, enable Google Cloud Armor security policies for applications by using Ingress for external Application Load Balancers (/kubernetes-engine/docs/concepts/ingress-xlb) and adding a security policy to the BackendConfig (/kubernetes-engine/docs/how-to/ingress-configuration#cloud_armor) attached to the Ingress object.

Use Identity-Aware Proxy to provide authentication for applications with IAM us

If you want to deploy services to be accessed only by users within the organization, but without the need of being on the corporate network, you can use Identity-Aware Proxy (/iap/docs/concepts-overview) to create an authentication layer for these applications. To enable Identity-Aware Proxy for GKE, follow the configuration steps (/iap/docs/enabling-kubernetes-howto) to

add Identity-Aware Proxy as part of the BackendConfig for your service Ingress. Identity-Aware Proxy can be combined with Google Cloud Armor.

Use organization policy constraints to further enhance security

Using organizational policy constraints (/resource-manager/docs/organization-policy/org-policy-constraints), you can set policies to further enhance your security posture. For example, you can use constraints to restrict Load Balancer creation to certain types (/load-balancing/docs/org-policy-constraints), such as internal load balancers only, or restricting external IP address usage (/compute/docs/ip-addresses/reserve-static-external-ip-address#disableexternalip).

Scaling cluster connectivity

This section covers scalable options for DNS and outbound connectivity from your clusters towards the internet and Google services.

Best practices:

Use Cloud DNS for GKE (#cloud-dns).
Enable NodeLocal DNSCache (#enable-nodelocal-dnscache).
Use Cloud NAT for internet access from private clusters (#use-cloudnat).
Use Private Google Access for access to Google services (#use-private-google-access).

Use Cloud DNS for GKE

You can use Cloud DNS for GKE (/kubernetes-engine/docs/how-to/cloud-dns) to provide Pod and Service DNS resolution with managed DNS without a cluster-hosted DNS provider. Cloud DNS removes the overhead of managing a cluster-hosted DNS server and requires no scaling, monitoring, or managing of DNS instances because it is a hosted Google service.

Enable NodeLocal DNSCache

GKE uses `kube-dns` in order to provide the cluster's local DNS service as a default cluster add-on. `kube-dns` is replicated across the cluster as a function of the total number of cores and nodes in the cluster.

You can improve DNS performance with NodeLocal DNSCache (/kubernetes-engine/docs/how-to/nodelocal-dns-cache). NodeLocal DNSCache is an add-on that is

deployed as a DaemonSet, and doesn't require any Pod configuration changes. DNS lookups to the local Pod service don't create open connections that need to be tracked on the node which allows for greater scale. External hostname lookups are forwarded to Cloud DNS whereas all other DNS queries go to kube-dns.

Enable NodeLocal DNSCache (/kubernetes-engine/docs/how-to/nodelocal-dns-cache) for more consistent DNS query lookup times and improved cluster scale. For Autopilot clusters, NodeLocal DNSCache is enabled by default and cannot be overridden.

The following Google Cloud CLI option enables NodeLocal DNSCache when you create a cluster: `--addons NodeLocalDNS.`

If you have control over the name that applications are looking to resolve, there are ways to improve DNS scaling. For example, use an FQDN (end the hostname with a period) or disable search path expansion through the `Pod.dnsConfig` manifest option.

Use Cloud NAT for internet access from private clusters

By default, private clusters don't have internet access. In order to allow Pods to reach the internet, enable Cloud NAT (/nat/docs) for each region. At a minimum, enable Cloud NAT for the primary and secondary ranges in the GKE subnet. Make sure that you allocate enough IP addresses for Cloud NAT and ports per VM (/nat/docs/ports-and-addresses#ports).

Use the following Cloud NAT Gateway configuration best practices while using Cloud NAT for private clusters:

- When you create your Cloud NAT gateway, enable it only for the subnet ranges used by your clusters. By counting all the nodes in all the clusters, you can determine how many NAT consumer VMs you have in the project.
- Use dynamic port allocation (/nat/docs/ports-and-addresses#dynamic-port) to allocate different numbers of ports per VM, based on the VM's usage. Start with minimum ports of 64 and maximum ports of 2048.
- If you need to manage many simultaneous connections to the same destination 3-tuple, lower the TCP `TIME_WAIT` timeout from its default value of `120s` to `5s`. For more information, see Specify different timeouts for NAT (/nat/docs/set-up-manage-network-address-translation#specify_different_timeouts_for_nat).
- Enable Cloud NAT error logging (/nat/docs/monitoring#configuring_logging) to check related logs.
- Check the Cloud NAT Gateway logs after configuring the gateway. To decrease allocation status dropped problems, you might need to increase the maximum number of ports per VM.

You should avoid double SNAT for Pods traffic (SNAT first at the GKE node and then again with Cloud NAT). Unless you require SNAT to hide the Pod IP addresses towards on-premises networks connected by Cloud VPN or Cloud Interconnect, `disable-default-snat` (/sdk/gcloud/reference/container/clusters/create#--disable-default-snat) and offload the SNAT tracking to Cloud NAT for scalability. This solution works for all primary and secondary subnet IP ranges. Use network policies to restrict external traffic after enabling Cloud NAT. Cloud NAT is not required to access Google services.

Use Private Google Access for access to Google services

In private clusters, Pods don't have public IP addresses to reach out to public services, including Google APIs and services. Private Google Access (/vpc/docs/private-google-access) lets private Google Cloud resources reach Google services.

This option is off by default and needs to be enabled on the subnet associated with the cluster during subnet creation time.

The `--enable-private-ip-google-access` Google Cloud CLI option enables Private Google Access when you create the subnet.

Serving applications

When creating applications that are reachable either externally or internal to your organization, make sure you use the right load balancer type and options. This section gives some recommendations on exposing and scaling applications with Cloud Load Balancing.

Best practices:

Use container-native load balancing (#use-container-native-lb).
Choose the correct GKE resource to expose your application (#choose-correct-resource).
Create health checks based on BackendConfig (#create-health-checks).
Use local traffic policy to preserve original IP addresses (#use-local-traffic-policy).
Use Private Service Connect (#use-psc).

Use container-native load balancing

Use container-native load balancing (/kubernetes-engine/docs/how-to/container-native-load-balancing) when exposing services by using HTTP(S) externally. Container-native load balancing allows for

fewer network hops, lower latency, and more exact traffic distribution. It also increases visibility in round-trip time and lets you use load-balancing features such as Google Cloud Armor.

Choose the correct GKE resource to expose your application

Depending on the scope of your clients (internal, external, or even cluster-internal), the regionality of your application, and the protocols that you use, there are different GKE resources that you can choose to use to expose your application. The Service networking overview (/kubernetes-engine/docs/concepts/service-networking) explains these options and can help you choose the best resource to expose each part of your application by using Google Cloud load balancing options.

Create health checks based on BackendConfig

If you use an Ingress to expose services, use a health check configuration in a BackendConfig CRD (/kubernetes-engine/docs/how-to/ingress-configuration#direct_health) to use the health check functionality of the external Application Load Balancer. You can direct the health check to the appropriate endpoint and set your own thresholds. Without a BackendConfig CRD, health checks are inferred from readiness probe parameters or use default parameters.

Use local traffic policy to preserve original IP addresses

When you use an internal passthrough Network Load Balancer with GKE (/kubernetes-engine/docs/how-to/internal-load-balancing), set the `externalTrafficPolicy` (/kubernetes-engine/docs/how-to/service-parameters#externalTrafficPolicy) option to `Local` to preserve the source IP address of the requests. Use this option if your application requires the original source IP address. However, the `externalTrafficPolicy` `local` option can lead to less optimal load spreading, so only use this feature when required. For HTTP(S) services, you can use Ingress controllers and get the original IP address by reading the `X-Forwarded-For` (/load-balancing/docs/https#target-proxies) header in the HTTP request.

Use Private Service Connect

You can use Private Service Connect (/vpc/docs/private-service-connect) to share internal passthrough Network Load Balancer Services across other VPC networks. This is useful for Services that are hosted on GKE clusters but are serving customers that are running in different projects and different VPCs.

You can use Private Service Connect to reduce IP address consumption by providing connectivity between VPCs with overlapping IP addresses.

Operations and administration

Best practices:

Use IAM for GKE permissions to control policies in Shared VPC networks (#use-iam-to-control-sharedvpc-networks).
Use regional clusters and distribute your workloads for high availability (#use-regional-clusters-distribute-workloads).
Use Cloud Logging and Cloud Monitoring and enable network policy logging (#logging-monitoring).

The following sections contain operational best practices which help you ensure granular authorization options for your workloads. To avoid creating manual firewall rules, follow the operational best practices in this section. It also includes recommendations for distributing your workloads and for monitoring and logging in GKE.

Use IAM for GKE permissions to control policies in Shared VPC networks

When using Shared VPC networks (/vpc/docs/shared-vpc), firewall rules for load balancers are automatically created in the host project.

To avoid having to manually create firewall rules, assign a least-privilege custom role to the GKE service account in the host project named `service-`*`HOST_PROJECT_NUMBER`*`@container-engine-robot.iam.gserviceaccount.com`.

Replace *`HOST_PROJECT_NUMBER`* with the project number of the host project for the Shared VPC.

The custom role that you create should have the following permissions:

- `compute.firewalls.create`
- `compute.firewalls.get`
- `compute.firewalls.list`
- `compute.firewalls.delete`

In addition, firewall rules created by GKE always have the default priority of 1000, so you can disallow specific traffic from flowing by creating firewall rules at a higher priority.

If you want to restrict creation of certain load balancer types, use organizational policies to restrict load balancer creation (/load-balancing/docs/org-policy-constraints).

Use regional clusters and distribute your workloads for high availability

Regional clusters (/kubernetes-engine/docs/concepts/types-of-clusters#regional_clusters) can increase the availability of applications in a cluster because the cluster control plane and nodes are spread across multiple zones.

However, to have the best possible user experience in case of a zone failure, use the cluster autoscaler (/kubernetes-engine/docs/concepts/cluster-autoscaler) to make sure that your cluster can handle the required load at any time. Also use Pod anti-affinity (https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#node-affinity) to ensure that Pods of a given service are scheduled in multiple zones. For more information about how to configure these settings for high availability and cost optimizations, see the Best practices for highly-available GKE clusters (/blog/products/containers-kubernetes/best-practices-for-creating-a-highly-available-gke-cluster).

Note: There are costs involved for cross-zone data transfers (/vpc/network-pricing#egress-within-gcp).

Use Cloud Logging and Cloud Monitoring and enable network policy logging

While each organization has different requirements for visibility and auditing, we recommend enabling network policy logging (/kubernetes-engine/docs/how-to/network-policy-logging). This feature is only available with GKE Dataplane V2 (/kubernetes-engine/docs/how-to/dataplane-v2). Network policy logging provides visibility into policy enforcement and Pod traffic patterns. Be aware that there are costs involved for network policy logging (/kubernetes-engine/docs/how-to/network-policy-logging#pricing).

For GKE clusters using version 1.14 or later, Logging and Monitoring (/stackdriver/docs/solutions/gke) are both enabled by default. Monitoring provides a dashboard for your GKE clusters. Logging also enables GKE annotations for VPC Flow Logs (/vpc/docs/using-flow-logs). By default, Logging collects logs for all workloads deployed to the cluster but a system-only logs option (/stackdriver/docs/solutions/gke/installing#controlling_the_collection_of_application_logs) also exists. Use the GKE dashboard (/stackdriver/docs/solutions/gke/observing) to observe and set alerts. Be aware that there are costs involved for the Google Cloud's operations suite (/stackdriver/pricing). For clusters created in the Autopilot mode, monitoring and logging are automatically enabled and not configurable.

Checklist summary

Area	Practice
VPC design (#vpc-design)	☐ Use VPC-native clusters (#vpc-native-clusters) ☐ Use Shared VPC networks (#sharedvpc_networks)
IP address management strategies (#ip-address-mgmt)	☐ Plan the required IP address allotment (#plan-ip-allotment) ☐ Use non-RFC 1918 space if needed (#use-non-rfc1918) ☐ Use custom subnet mode (#custom-subnet-mode) ☐ Plan Pod density per node (#pod-density-per-node) ☐ Avoid overlaps with IP addresses used in other environments (#avoid-ip-overlaps) ☐ Create a load balancer subnet (#create-lb-subnet) ☐ Reserve enough IP address space for cluster autoscaler (#reserve-ip-space) ☐ Share IP addresses across clusters (#share-ip-clusters) ☐ Share IP addresses for internal LoadBalancer Services (#share-ip-services)
Network security options (#network-security)	☐ Use GKE Dataplane V2 (#dataplane-v2) ☐ Choose a private cluster type (#private-cluster-type) ☐ Minimize the cluster control plane exposure (#minimize-control-plane-exposure) ☐ Authorize access to the control plane (#authorize-cp-access) ☐ Allow control plane connectivity (#allow-cp-connectivity) ☐ Deploy proxies for control plane access from peered networks (#deploy-proxies) ☐ Restrict cluster traffic using network policies (#restrict-traffic-network-pols) ☐ Enable Google Cloud Armor security policies for Ingress (#enable-security-policies) ☐ Use Identity-Aware Proxy to provide authentication for applications with IAM users (#use-iap) ☐ Use organization policy constraints to further enhance security (#use-org-policy-constraints)
Scaling (#scaling)	☐ Use Cloud DNS for GKE (#cloud-dns)

	☐ Enable NodeLocal DNSCache (#enable-nodelocal-dnscache) ☐ Use Cloud NAT for internet access from private clusters (#use-cloudnat) ☐ Use Private Google Access for access to Google services (#use-private-google-access)
Serving applications (#serving-apps)	☐ Use container-native load balancing (#use-container-native-lb) ☐ Choose the correct GKE resource to expose your application (#choose-correct-resource) ☐ Create health checks based on BackendConfig (#create-health-checks) ☐ Use local traffic policy to preserve original IP addresses (#use-local-traffic-policy) ☐ Use Private Service Connect (#use-psc)
Operations and administration (#ops-and-admin)	☐ Use IAM for GKE permissions to control policies in Shared VPC networks (#use-iam-to-control-sharedvpc-networks) ☐ Use regional clusters and distribute your workloads for high availability (#use-regional-clusters-distribute-workloads) ☐ Use Cloud Logging and Cloud Monitoring and enable network policy logging (#logging-monitoring)

What's next

- GKE best practices insights (/network-intelligence-center/docs/%0Anetwork-analyzer/insights/kubernetes-engine/gke-best-practices)
- Best practices for enterprise multi-tenancy (/kubernetes-engine/docs/best-practices/enterprise-multitenancy)
- Exposing GKE applications through Ingress and Services (/blog/products/containers-kubernetes/exposing-services-on-gke)
- Best practices and reference architectures for VPC design (/solutions/best-practices-vpc-design)

Last updated 2024-02-01 UTC.