Quickstart: Deploy an app to a GKE cluster

Autopilot overview

AUTOPILOT

This page describes the Autopilot mode of operation in Google Kubernetes Engine (GKE) and provides you with resources that you can use to plan, set up, and manage your clusters.

What is Autopilot?

GKE Autopilot is a mode of operation in GKE in which Google manages your cluster configuration, including your nodes, scaling, security, and other preconfigured settings. Autopilot clusters are optimized to run most production workloads, and provision compute resources based on your Kubernetes manifests. The streamlined configuration follows GKE best practices and recommendations for cluster and workload setup, scalability, and security. For a list of built-in settings, refer to the Autopilot and Standard comparison table (/kubernetes-engine/docs/resources/autopilot-standard-feature-comparison).

Pricing

You only pay for the CPU, memory, and storage that your workloads request while running on GKE Autopilot.

You aren't billed for unused capacity on your nodes, because GKE manages the nodes. You also aren't charged for system Pods, operating system costs, or unscheduled workloads. For detailed pricing information, refer to Autopilot pricing (/kubernetes-engine/pricing#autopilot_mode).

Benefits

- **Focus on your apps**: Google manages the infrastructure, so you can focus on building and deploying your applications.
- **Security**: Clusters have a default hardened configuration, with many security settings enabled by default. GKE automatically applies security patches to your nodes when available, adhering to any maintenance schedules you configured.
- **Pricing**: The Autopilot pricing model simplifies billing forecasts and attribution because it's based on resources requested by your Pods.

- **Node management**: Google manages worker nodes, so you don't need to create new nodes to accommodate your workloads or configure automatic upgrades and repairs.
- **Scaling**: When your workloads experience high load and you add more Pods to accommodate the traffic, such as with Kubernetes Horizontal Pod Autoscaling, GKE automatically provisions new nodes for those Pods, and automatically expands the resources in your existing nodes based on need.
- **Scheduling**: Autopilot manages Pod bin-packing for you, so you don't have to think about how many Pods are running on each node. You can further control Pod placement by using Kubernetes mechanisms such as affinity and Pod spread topology.
- **Resource management**: If you deploy workloads without setting resource values such as CPU and memory, Autopilot automatically sets pre-configured default values and modifies your resource requests at the workload level.
- **Networking**: Autopilot enables some networking security features by default, such as ensuring that all Pod network traffic passes through your Virtual Private Cloud firewall rules, even if the traffic is going to other Pods in the cluster.
- **Release management**: All Autopilot clusters are enrolled in a GKE release channel, which ensures that your control plane and nodes run on the latest qualified versions in that channel.
- **Managed flexibility**: If your workloads have specific hardware or resource requirements, such as high CPU or memory, Autopilot offers pre-configured compute classes (/kubernetes-engine/docs/concepts/autopilot-compute-classes) built for those workloads. You request the compute class in your deployment instead of needing to manually create new nodes that are backed by customized machine types and hardware. You can also select GPUs (/kubernetes-engine/docs/how-to/autopilot-gpus) to accelerate workloads like batch or AI/ML applications.
- **Reduced operational complexity**: Autopilot reduces platform administration overhead by removing the need to continuously monitor nodes, scaling, and scheduling operations.

Autopilot comes with a SLA (/kubernetes-engine/sla) that covers both the control plane and the compute capacity used by your Pods.

Plan your Autopilot clusters

Before you create a cluster, plan and design your Google Cloud architecture. In Autopilot, you request hardware in your workload specifications. GKE provisions and manages the corresponding infrastructure to run those workloads. For example, if you run machine learning workloads, you request hardware accelerators. If you develop Android apps, you request Arm CPUs.

Plan and request quota for your Google Cloud project or organization based on the scale of your workloads. GKE can only provision infrastructure for your workloads if your project has enough quota for that hardware.

Consider the following factors during planning:

- Estimated cluster size and scale
- Workload type
- Cluster layout and usage
- Networking layout and configuration
- Security configuration
- Cluster management and maintenance
- Workload deployment and management
- Logging and monitoring

The following sections provide information and useful resources for these considerations.

Networking

When you create an Autopilot cluster with public networking, workloads in the cluster can communicate with each other and with the internet. This is the default networking mode. Google Cloud and Kubernetes provide various additional networking features and capabilities that you can leverage based on your use case, such as clusters with private networking (/kubernetes-engine/docs/how-to/private-clusters).

Networking in Kubernetes and in the cloud is complex. Ensure that you understand the basic concepts of networking before you start changing the defaults that Google Cloud sets for you. The following table provides you with resources to learn more about networking in GKE based on your use case:

Use case	Resources

Understand how networking works in Kubernetes and GKE	• Learn the Kubernetes networking model (https://kubernetes.io/docs/concepts/services-networking/). • Learn the GKE networking model (/architecture/gke-compare-network-models). After you learn the networking model, consider your organization's networking and network security requirements. Choose GKE and Google Cloud networking features that satisfy those criteria.
Plan your GKE networking configuration	We recommend that you understand the networking quotas (/kubernetes-engine/quotas) for GKE, such as endpoints per Service and API request limits. The following resources will help you to plan specific aspects of your networking setup: • To learn about networking options inside and outside the cluster, read the GKE networking overview (/kubernetes-engine/docs/concepts/network-overview). • To learn our recommendations for network design, read the Best practices for GKE networking (/kubernetes-engine/docs/best-practices/networking). • To learn how to optimize your IP address management, read the GKE address management series (/architecture/gke-address-management-overview). • To learn what firewall rules GKE creates based on the Kubernetes resources you create, refer to Automatically created firewall rules (/kubernetes-engine/docs/concepts/firewall-rules).
Expose your workloads	• To expose your apps to the internet, use Services (/kubernetes-engine/docs/concepts/service-networking), which let you expose an app running in a group of Pods as a single network service. • To configure workloads to securely communicate with Google Cloud APIs, use Workload Identity (/kubernetes-engine/docs/how-to/workload-identity#authenticating_to).
Run highly-available connected services in multiple clusters	Use multi-cluster Services (MCS) (/kubernetes-engine/docs/concepts/multi-cluster-services).
Load balance incoming traffic	• To load balance external HTTP(S) traffic to multiple Services based on URIs and paths, for example a complex web application, use Ingress for external Application Load Balancers (/kubernetes-engine/docs/tutorials/http-balancer). • To load balance external traffic to a single Service, such as a Deployment running a public email server, use a LoadBalancer Service (/kubernetes-engine/docs/how-to/exposing-apps#creating_a_service_of_type_loadbalancer) to create an external passthrough Network Load Balancer. • To load balance internal HTTP(S) traffic to multiple Services based on URIs and paths, such as with a web application in your company intranet, use Ingress for

	internal Application Load Balancers (/kubernetes-engine/docs/tutorials/http-balancer). • To load balance internal traffic to a single Service, such as with a corporate email server, use an internal passthrough Network Load Balancer (/kubernetes-engine/docs/how-to/exposing-apps#creating_a_service_of_type_loadbalancer) .
Configure cluster network security	• To control or prevent access to your cluster from the public internet, create private clusters (/kubernetes-engine/docs/how-to/private-clusters#private_cp). • To restrict control plane access to specific IP address ranges, use control plane authorized networks (/kubernetes-engine/docs/how-to/authorized-networks). • To control Pod traffic at the IP address or port level, use network policies (https://kubernetes.io/docs/tasks/administer-cluster/declare-network-policy/). Autopilot clusters use GKE Dataplane V2 (/kubernetes-engine/docs/concepts/dataplane-v2) to route packets with low latency using eBPF.
Observe your Kubernetes network traffic	• To ingest the GKE Dataplane V2 metrics, configure Google Managed Service for Prometheus (/stackdriver/docs/managed-prometheus). By default, GKE Dataplane V2 metrics are exposed in GKE Autopilot. • To access visualizations, Network Policy verdicts, and flow dumps, configure additional troubleshooting tools using GKE Dataplane V2 observability (/kubernetes-engine/docs/concepts/about-dpv2-observability).

Scaling

Operating a platform effectively at scale requires planning and careful consideration. You must consider the *scalability* of your design, which is the ability of your clusters to grow while remaining within service-level objectives (SLOs). For detailed guidance for both platform administrators and developers, refer to the Guidelines for creating scalable clusters (/kubernetes-engine/docs/best-practices/scalability).

You should also consider the GKE quotas and limits (/kubernetes-engine/quotas#limits_per_cluster), especially if you plan to run large clusters with potentially thousands of Pods.

Scale Autopilot workloads

In Autopilot, GKE automatically scales your nodes based on the number of Pods in your cluster. To automatically scale the number of Pods in your cluster, we recommend that you

use a mechanism such as Kubernetes horizontal Pod autoscaling (/kubernetes-engine/docs/concepts/horizontalpodautoscaler), which can scale Pods based on the built-in CPU and memory metrics, or custom metrics from Cloud Monitoring. To learn how to configure scaling based on various metrics, refer to Optimize Pod autoscaling based on metrics (/kubernetes-engine/docs/tutorials/autoscaling-metrics).

Security

Autopilot clusters enable and apply security best practices and settings by default, including many of the recommendations in Harden your cluster security (/kubernetes-engine/docs/how-to/hardening-your-cluster) and the GKE security overview (/kubernetes-engine/docs/concepts/security-overview).

If you want to learn more about Autopilot hardening measures and how to implement your specific security requirements, refer to Security measures in Autopilot (/kubernetes-engine/docs/concepts/autopilot-security).

Create a cluster

After planning your environment and understanding your requirements, create an Autopilot cluster (/kubernetes-engine/docs/how-to/creating-an-autopilot-cluster). New Autopilot clusters are regional clusters that have a publicly accessible IP address. Each cluster has baseline hardening measures applied, as well as automatic scaling and other features. For a full list of pre-configured features, refer to Compare GKE Autopilot and Standard (/kubernetes-engine/docs/resources/autopilot-standard-feature-comparison).

If you want to create the cluster with no public IP address, create a private cluster (/kubernetes-engine/docs/how-to/private-clusters#gcloud) instead.

Deploy workloads on Autopilot

To deploy a workload to a running Autopilot cluster, write a Kubernetes manifest and apply it to the cluster. By default, Autopilot clusters are optimized to run most production workloads.

For an interactive guide in the Google Cloud console for deploying and exposing an app, click **Guide me**:

Guide me (https://console.cloud.google.com/getting-started?tutorial=kubernetes--autopilot)

Some of your workloads might have specialized hardware requirements, such as ML workloads that need hardware accelerators or mobile app testing that requires the Arm architecture. Autopilot has *compute classes* (/kubernetes-engine/docs/concepts/autopilot-compute-classes) that Google Cloud has configured to run workloads that have special compute requirements. When deploying these workloads, request a compute class in the manifest. Autopilot automatically provisions nodes backed by specialized machines, manages scheduling, and allocates hardware.

The following table shows some common requirements and provides recommendations for what you should do:

Use case	Resources
Run Arm workloads	Request the `Scale-Out` compute class and the `arm64` architecture in your manifest. For instructions, refer to Deploy Autopilot workloads on Arm architecture (/kubernetes-engine/docs/how-to/autopilot-arm-workloads).
Run accelerated AI/ML workloads	Request GPUs in your manifest. For instructions, refer to Deploy GPU workloads in Autopilot (/kubernetes-engine/docs/how-to/autopilot-gpus).
Run workloads that require high compute or memory capacity	Request the `Balanced` compute class. For instructions, refer to Choose compute classes for Autopilot Pods (/kubernetes-engine/docs/how-to/autopilot-compute-classes).
Run workloads that require more efficient horizontal scaling of CPU capacity and single thread-per-core compute	Request the `Scale-Out` compute class. For instructions, refer to Choose compute classes for Autopilot Pods (/kubernetes-engine/docs/how-to/autopilot-compute-classes).
Run fault-tolerant workloads such as batch jobs at lower costs	Specify *Spot Pods* in your manifest. For instructions, refer to Run fault-tolerant workloads at lower costs in Spot Pods (/kubernetes-engine/docs/how-to/autopilot-spot-pods). You can use any compute class or hardware configuration with Spot Pods.
Run workloads that require minimal disruptions, such as game servers or work queues	Specify the `cluster-autoscaler.kubernetes.io/safe-to-evict=false` annotation in the Pod specification. Pods are protected from eviction caused by node auto-upgrades or scale-down events for up to seven days. For instructions, see Extend the run time of Autopilot Pods (/kubernetes-engine/docs/how-to/extended-duration-pods).

Autopilot lets you request CPU, memory, and ephemeral storage resources for your workloads. The allowed ranges depend on whether you want to run your Pods on the

default general-purpose compute platform, or on a *compute class* (/kubernetes-engine/docs/concepts/autopilot-compute-classes).

For information about the default container resource requests and the allowed resource ranges, refer to Resource requests in Autopilot (/kubernetes-engine/docs/concepts/autopilot-resource-requests).

Workload separation

Autopilot clusters support using node selectors and node affinity to configure *workload separation*. Workload separation is useful when you need to tell GKE to place workloads on nodes that meet specific criteria, such as custom node labels. For example, you can tell GKE to schedule game server Pods on nodes with the `game-server` label and avoid scheduling any other Pods on those nodes.

To learn more, refer to Configure workload separation in GKE (/kubernetes-engine/docs/how-to/workload-separation).

Schedule Pods in specific zones using zonal topology

If you need to place Pods in a specific Google Cloud zone, for example to access information on a zonal Compute Engine persistent disk, see Place GKE Pods in specific zones (/kubernetes-engine/docs/how-to/gke-zonal-topology).

Pod affinity and anti-affinity

Use Pod affinity and anti-affinity to co-locate Pods on a single node or to make some Pods avoid other Pods. Pod affinity and anti-affinity tell Kubernetes to make a scheduling decision based on the labels of Pods running on nodes in a specific topology domain, such as a specific region or zone. For example, you could tell GKE to avoid scheduling frontend Pods alongside other frontend Pods on the same nodes to improve availability in case of an outage.

For instructions and more details, refer to Pod affinity and anti-affinity (https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#inter-pod-affinity-and-anti-affinity)
.

In GKE, you can use Pod affinity and anti-affinity with the following labels in `topologyKey`:

- `topology.kubernetes.io/zone`

- `kubernetes.io/hostname`

Pod topology spread constraints

To improve the availability of your workloads as Kubernetes scales the number of Pods up and down, you can set *Pod topology spread constraints*. This controls how Kubernetes spreads your Pods across nodes within a topology domain, such as a region. For example, you could tell Kubernetes to place a specific number of game server session Pods in each of three Google Cloud zones in the `us-central1` region.

For examples, more details, and instructions, refer to Pod Topology Spread Constraints (https://kubernetes.io/docs/concepts/scheduling-eviction/topology-spread-constraints/).

Manage and monitor your Autopilot clusters

In Autopilot, GKE automatically manages cluster upgrades and maintenance for both the control plane and worker nodes. Autopilot clusters also have built-in functionality for you to monitor your clusters and workloads.

GKE version upgrades

All Autopilot clusters are enrolled in a GKE *release channel*. In release channels, GKE manages the Kubernetes version of the cluster, balancing between feature availability and version stability depending on the channel. By default, Autopilot clusters are enrolled in the Regular release channel, but you can select a different channel that meets your stability and functionality needs. For more information about release channels, see About release channels (/kubernetes-engine/docs/concepts/release-channels).

GKE automatically starts upgrades, monitors progress, and pauses the operation if problems occur. You can manually control the upgrade process in the following ways:

- To control when GKE *can* perform automatic upgrades, create maintenance windows (/kubernetes-engine/docs/concepts/maintenance-windows-and-exclusions#maintenance_windows). For example, you can set the maintenance window to the night before your multiplayer game's weekly reset, so that players can log in at reset without disruptions.
- To control when GKE *can't* start automatic upgrades during a specific time range, use maintenance exclusions

(/kubernetes-engine/docs/concepts/maintenance-windows-and-exclusions#exclusions). For example, you can set a maintenance exclusion for the duration of your Black Friday and Cyber Monday sales event so that your customers can shop without issues.

- To get a new version before auto-upgrades start, manually upgrade the control plane (/kubernetes-engine/docs/how-to/upgrading-a-cluster#upgrade_cp). GKE reconciles the node version with the control plane version over time.
- To get a patch version that's only available in a newer release channel, see Run patch versions from a newer channel (/kubernetes-engine/docs/concepts/release-channels#newer-patch-versions). For example, you might need a specific patch version to mitigate a recent vulnerability disclosure.

Monitor your Autopilot clusters

Autopilot clusters already have Cloud Logging, Cloud Monitoring, and Google Cloud Managed Service for Prometheus enabled.

Autopilot clusters collect the following types of logs and metrics automatically, adhering to Google's best practices for telemetry collection:

Logs for Cloud Logging

- System logs
- Workload logs
- Admin Activity audit logs
- Data Access audit logs

Metrics for Cloud Monitoring

- System metrics
- Workload metrics (from Managed Service for Prometheus)

No additional configuration is required to enable logging and monitoring. The following table shows you how to interact with the collected telemetry based on your requirements:

Use case	Resources
Understand and access your GKE logs	• To learn about the types of logs that we automatically collect, see What logs are collected (/stackdriver/docs/solutions/gke/managing-logs#what_logs).

	• To access the logs and to use the Cloud Logging user interface in the Google Cloud console, see Viewing your GKE logs (/stackdriver/docs/solutions/gke/using-logs). • For sample queries that you can use to filter Kubernetes system and workload logs, see Kubernetes-related queries (/logging/docs/view/query-library#kubernetes-filters). • For sample queries that you can use to filter Admin Activity and Data Access audit logs, see GKE audit logging information (/kubernetes-engine/docs/how-to/audit-logging#sample_queries). • To configure logs for multi-tenant environments, for example when teams have specific namespaces in a single GKE cluster but each team has its own Google Cloud project, see Multi-tenant logging on GKE (/stackdriver/docs/solutions/gke/multi-tenant-logging).
Observe the performance of your GKE clusters	Effective monitoring of your cluster performance can help you to optimize the operating costs of your clusters and workloads. • Use the GKE dashboard in Monitoring to visualize the status of your clusters. To learn more, see Observing your GKE clusters (/stackdriver/docs/solutions/gke/observing). • GKE also provides an **Observability** dashboard in the Google Cloud console. For details, see View observability metrics (/kubernetes-engine/docs/how-to/view-observability-metrics).
Monitor the security posture of your clusters	Use the security posture dashboard to audit your running workloads against GKE best practices, scan for vulnerabilities in your container operating systems and language packages, and get actionable mitigation recommendations. To learn more, see About the security posture dashboard (/kubernetes-engine/docs/concepts/about-security-posture-dashboard).

Troubleshooting

For troubleshooting steps, refer to Troubleshooting Autopilot clusters (/kubernetes-engine/docs/troubleshooting/troubleshooting-autopilot-clusters).

What's next

- Learn more about Autopilot architecture (/kubernetes-engine/docs/concepts/autopilot-architecture).
- Create an Autopilot cluster (/kubernetes-engine/docs/how-to/creating-an-autopilot-cluster).

Last updated 2023-11-20 UTC.