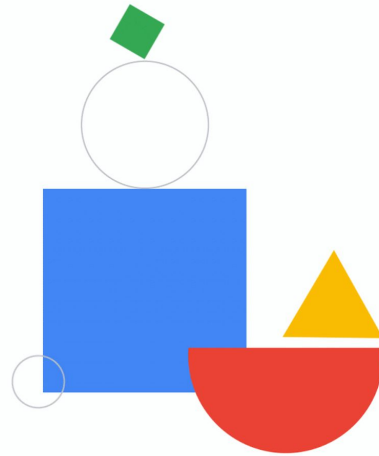


Developing Applications with Google Cloud: Foundations

Module 7: Compute Options for Your Application



Welcome to Developing Applications with Google Cloud: Foundations, module 7: Compute Options for Your Application.

Google Cloud has a range of compute options that you can use to run your applications. You can choose a platform that matches the needs of your application, including the level of control that you need for the infrastructure. Having more control over the infrastructure usually leads to a greater operational burden. If you use Cloud Client Libraries in your application to work with Google Cloud services, you can usually move to another platform without reworking your application.

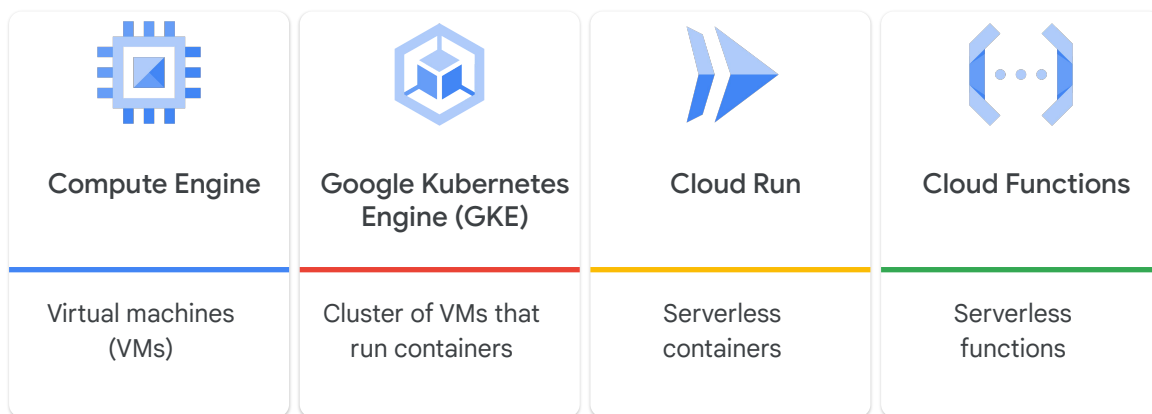
Agenda

01	Compute Engine
02	Google Kubernetes Engine
03	Cloud Run
04	Cloud Functions
05	Comparisons



In this module, you learn about which use cases are most appropriate for each compute option and how to decide between them.

Where should I run my applications?



Google Cloud provides a wide range of platforms on which to run your applications.

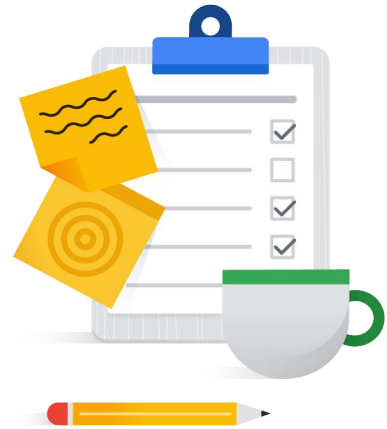
- Compute Engine lets you create virtual machines, or VMs, that mimic the servers you may have used in a traditional data center. Virtual machines are highly flexible: they let you run the same wide range of applications you can run on physical hardware, but now on Google's infrastructure.
- Google Kubernetes Engine, or GKE, is Google Cloud's managed service for running containers and managing the virtual machines used to run them. With GKE, a cluster of virtual machines is created for running your containerized applications. When you deploy applications to the cluster, GKE manages the scaling and security for the cluster and applications, reducing the operational costs of running your applications.
- Cloud Run is a fully managed serverless platform that also runs containerized applications. Unlike with GKE, all infrastructure management for Cloud Run is abstracted away. Cloud Run automatically scales up and down from zero almost instantaneously, depending on traffic. You only pay when your code is running.
- Cloud Functions also lets you run your code in the cloud with zero

- management of servers or containers. Write your event-driven code, and Google Cloud manages everything, scaling from zero to planet-scale.

So, which compute option should you use to run your applications? Well, it depends.

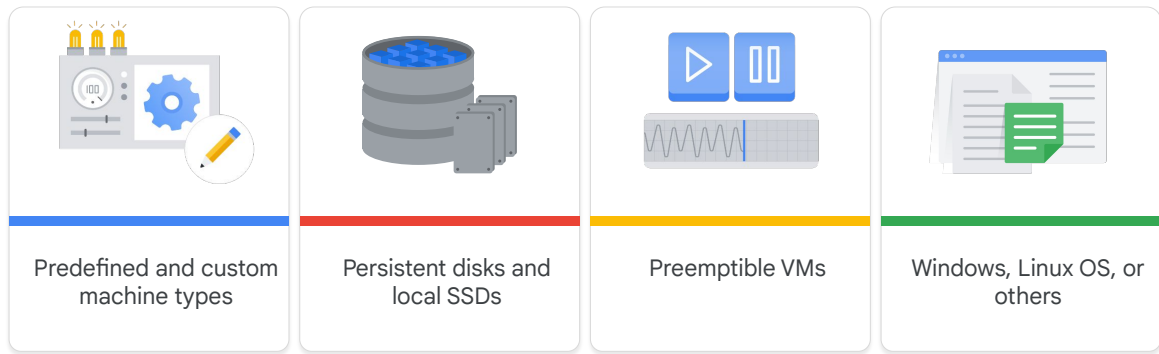
Agenda

01	Compute Engine
02	Google Kubernetes Engine
03	Cloud Run
04	Cloud Functions
05	Comparisons



First, we start with Compute Engine. Compute Engine is the most flexible option for running your applications, but it requires the most operational effort to manage.

Run your application on high-performance VMs with Compute Engine



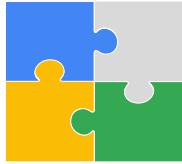
With Compute Engine, you create high-performance virtual machines and then install and run your applications on them.

- Compute Engine supports predefined machine types for popular configurations but also lets you create custom machine types to customize CPU and memory for your VMs.
- Compute Engine lets you create and attach persistent disks and local SSDs. These disks can be accessed like physical disks in a desktop or server. Unlike typical physical disks, Compute Engine disks can be increased in size while they are running. The performance and throughput of persistent disks increase when they increase in size.
- Compute Engine provides preemptible VMs that are ideal for large compute and batch jobs. If capacity must be reclaimed, Google Cloud can terminate preemptible VMs. For applications that can handle these interruptions, preemptible VMs are available at a discount of at least 60% compared to standard VMs.
- You can run your choice of operating system on your VMs, including Debian, CentOS, Ubuntu, and various other flavors of Linux or Windows. You can also

- use a shared image from the Google Cloud community or bring your own operating system.

[Compute Engine: <https://cloud.google.com/compute>]

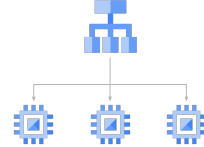
Use Compute Engine for full infrastructure control



Machine type and OS



Software

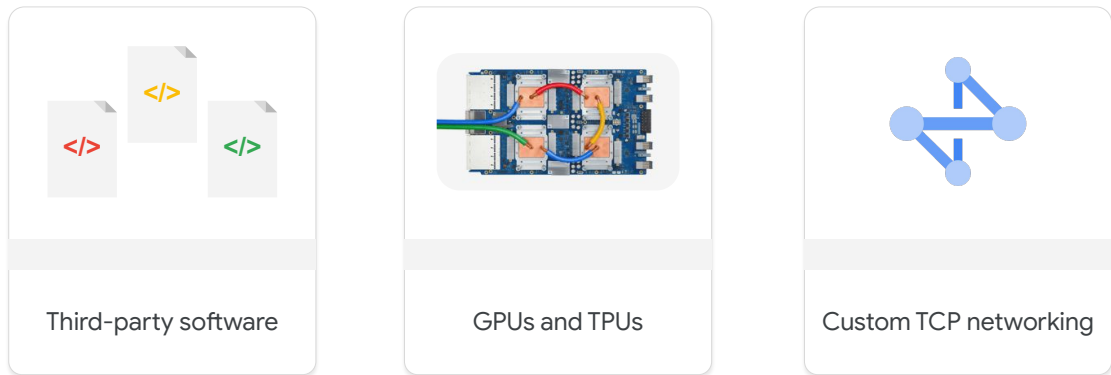


Instance groups with
global load balancing

Use Compute Engine when you want full control of your infrastructure.

- Compute Engine enables you to create highly customized VMs for specialized applications that have unique compute or operating system requirements.
- You can install and patch software that runs on a VM.
- You can create managed instance groups of VMs based on an instance template and configure global load balancing and autoscaling of the managed instance groups. Compute Engine can perform health checks and replace unhealthy instances in an instance group. Compute Engine can also autoscale the number of instances based on the traffic volume in specific regions.

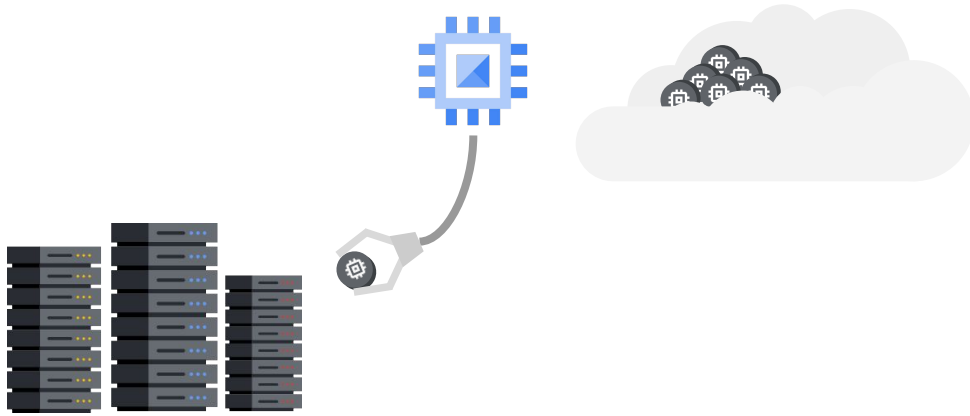
Use Compute Engine for maximum flexibility



Compute Engine offers you the most flexibility to configure your resources for the specific type of application that you need to run.

- You can install and run any third-party licensed software on Compute Engine.
- You can attach graphics processing units (GPUs) and Tensor Processing Units (TPUs) to Compute Engine VMs to accelerate parallel processing and machine learning workloads.
- You can use Compute Engine for applications that require TCP network protocols other than HTTP or HTTPS.

Use Compute Engine for lift-and-shift migrations



Compute Engine is ideal for lift-and-shift migrations. You can move virtual machines from your on-premises data center, or another cloud provider, to Google Cloud without changing your application.

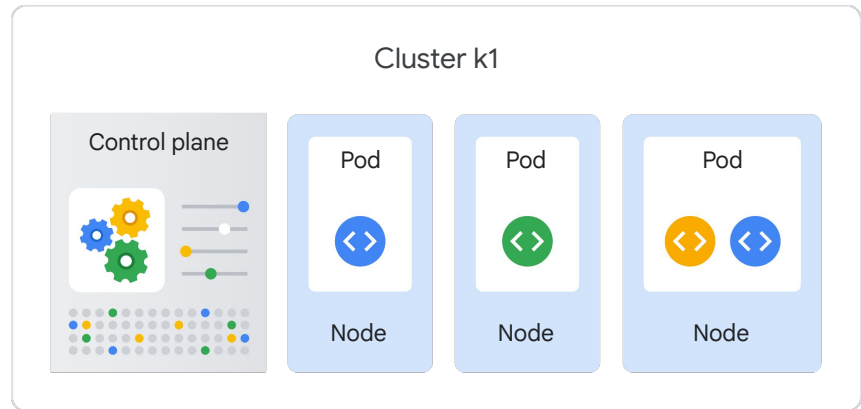
Agenda

01	Compute Engine
02	Google Kubernetes Engine
03	Cloud Run
04	Cloud Functions
05	Comparisons



Next is Google Kubernetes Engine. Kubernetes is a leading open source platform for deploying, scaling, and operating containers. Kubernetes, first developed at Google, is now a Cloud Native Computing Foundation project with a large and active community.

Kubernetes is an open source platform for deploying, scaling, and operating containers

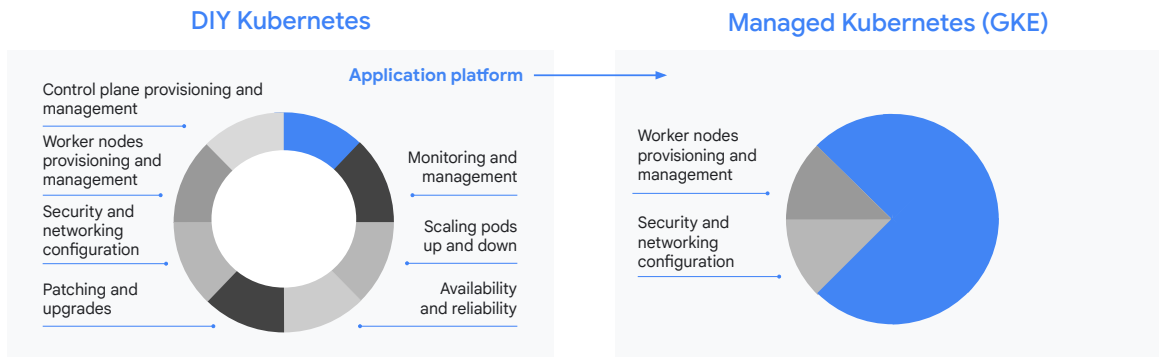


Kubernetes provides you with a framework to run distributed containerized systems resiliently and at scale. It manages many operational tasks, such as scaling application components, providing network abstractions, orchestrating failovers, rolling out deployments, storage orchestration, and management of secrets and configurations.

A Kubernetes cluster contains control plane and worker nodes. The nodes in a cluster are the machines—virtual or physical—that run your applications. The Kubernetes control plane manages the worker nodes and the Pods in the cluster. A Pod is a group of containers that share networking and storage resources on the node.

[Cloud Native Computing Foundation: <https://www.cncf.io/>
Kubernetes documentation: <https://kubernetes.io/docs/home/>]

GKE is a managed service for Kubernetes



Google Kubernetes Engine, or GKE, is a managed Kubernetes service on Google infrastructure. GKE helps you deploy, manage, and scale Kubernetes environments for your containerized applications on Google Cloud.

More specifically, GKE is a component of the Google Cloud compute offerings that facilitates bringing your Kubernetes workloads into the cloud.

For an unmanaged cluster, you need to manage most of the operational aspects of the cluster yourself.

GKE handles much of this operational effort for you automatically by eliminating many of the infrastructure tasks required to create and manage a Kubernetes cluster. With GKE, Google manages most of your cluster tasks. Google manages the control plane, scaling of Pods, node patching and upgrades, and the monitoring, availability, and reliability of the cluster.

By default, you manage the underlying nodes and node pools, including provisioning, maintenance, and lifecycle management. You're also responsible for selecting the security and networking configuration for your cluster. This level of management is the standard mode for GKE.

[Google Kubernetes Engine: <https://cloud.google.com/kubernetes-engine/>

Google Kubernetes Engine Overview:

<https://cloud.google.com/kubernetes-engine/docs/concepts/kubernetes-engine-overview>]

GKE Autopilot manages it all



GKE Autopilot is a mode of operation in which the entire cluster's infrastructure is managed for you, including control plane, node pools, and nodes.

By managing the cluster infrastructure, Autopilot helps reduce operational and maintenance costs while improving resource utilization. Autopilot is a fully managed Kubernetes experience that lets you focus on your workloads instead of the management of the cluster's infrastructure.

Autopilot automatically implements GKE hardening guidelines and security and networking best practices and blocks less safe practices.

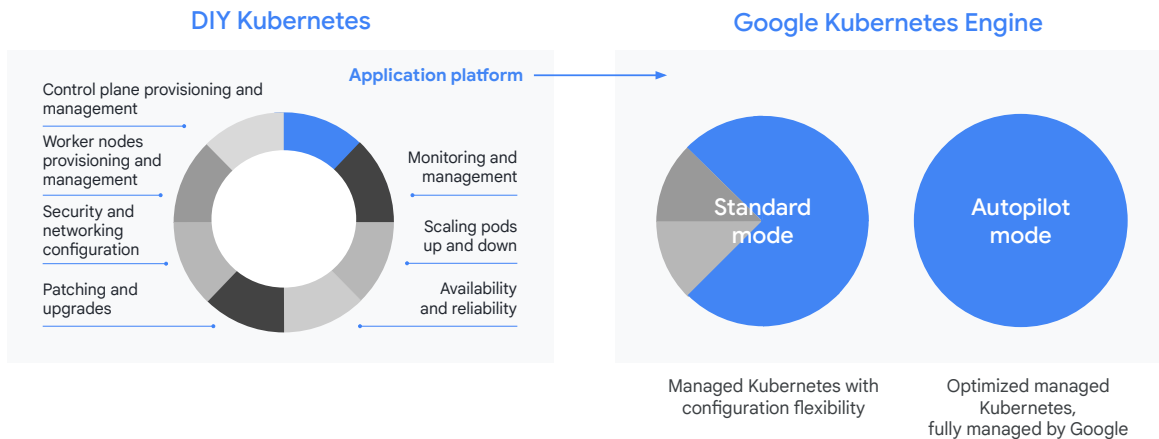
[Introducing GKE Autopilot (blog):

<https://cloud.google.com/blog/products/containers-kubernetes/introducing-gke-autopilot>

Autopilot overview:

<https://cloud.google.com/kubernetes-engine/docs/concepts/autopilot-overview>]

Choose your GKE operation mode



GKE Standard mode provides customers with advanced configuration flexibility over the cluster infrastructure. GKE Autopilot mode lets Google provision and manage the entire cluster and underlying infrastructure.

You can use different modes for different clusters, depending on how much infrastructure control you need.

[Comparing Autopilot and Standard modes:

<https://cloud.google.com/kubernetes-engine/docs/concepts/autopilot-overview#comparison>]

GKE features



- ✓ Fully managed
- ✓ Container-optimized operating system
- ✓ Auto-upgrade and auto-repair
- ✓ Cluster scaling
- ✓ Integrated logging and monitoring
- ✓ Google Cloud integration

- GKE is fully managed, which means that you don't have to provision the underlying resources.
- GKE uses a container-optimized operating system to run your workloads. Google maintains this operating system, which is optimized to scale quickly with a minimal resource footprint.
- When you use GKE, you start by directing the service to instantiate a Kubernetes cluster for you. The GKE auto-upgrade feature, when enabled, ensures that your clusters are always automatically upgraded with the latest stable version of Kubernetes.
The virtual machines that host your containers in a GKE cluster are called nodes. Auto-repair can automatically repair unhealthy nodes for you. It performs periodic health checks on each node of the cluster. If a node is determined to be unhealthy and requires repair, GKE will drain the node, thus allowing workloads to gracefully exit. It will then recreate the node.
- GKE and Kubernetes both support the scaling of workloads within a cluster. GKE also supports scaling of the cluster itself.
- GKE uses Cloud Monitoring and Cloud Logging to help you monitor and

- understand your applications' performance and behavior.
- GKE seamlessly integrates with many parts of Google Cloud. With Cloud Build, you can use private container images that you have securely stored in Artifact Registry to automate the deployment of your workloads. Identity and Access Management lets you control access by using accounts and role permissions. GKE is integrated with Virtual Private Clouds, or VPCs, which lets you use Google Cloud's networking features. And finally, the Google Cloud Console provides insights into GKE clusters and their resources, thus letting you view, inspect, and delete resources in those clusters.

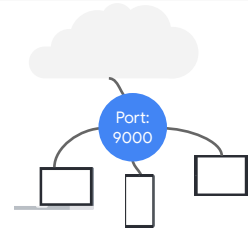
Use GKE for complex, portable applications



Any application runtime packaged as a Docker container image



Hybrid or multi-cloud applications



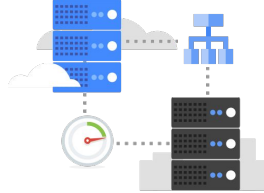
Protocols other than HTTPS

- GKE supports any application runtime that you can package as a Docker image. GKE is ideally suited for containerized applications, including third-party containerized software.
- You can run your container image on Kubernetes in a hybrid or multi cloud environment. This feature is especially helpful when some parts of your application run on-premises and other parts run in the cloud.
- You can use GKE to run containerized applications that use network protocols other than HTTP and HTTPS.

GKE simplifies infrastructure service provisioning for your applications



Google Cloud
persistent disks



Google Cloud network
load balancers



Integration with Google
Cloud's operations suite

Managing the infrastructure for a Kubernetes environment can be complex. GKE simplifies many of the operational tasks associated with provisioning and managing the infrastructure.

- With GKE, Google Cloud persistent disks are automatically provisioned by default when you create Kubernetes persistent volumes to provide storage for stateful applications.
- GKE automatically provisions Google Cloud network load balancers when you deploy Kubernetes network load balancer services, and provisions Google Cloud HTTP and HTTP(S) Load Balancing when you configure Kubernetes Ingress resources. This auto-provisioning feature eliminates the need to configure and manage these resources manually.
- GKE has support for Google Cloud's operations suite, which provides integration with tools for troubleshooting and application and service monitoring.

Use GKE for greater control over how resources are deployed for your applications

```
gcloud container clusters create
```

```
--machine-type=MACHINE_TYPE  
--disk-size=DISK_SIZE  
--num-nodes=NUM_NODES  
...
```

```
gcloud container clusters create
```

```
--num-nodes 30  
--enable-autoscaling  
--min-nodes 15  
--max-nodes 50  
...
```

GKE simplifies cluster deployment and scaling. You can describe the compute, memory, network, and storage resources that you want to make available across all the containers required by your applications. GKE will provision and manage the underlying Google Cloud resources automatically. You can either deploy fixed-size clusters or configure your clusters to automatically scale. Autoscaling adds or removes compute instances in response to changes in the resource requirements of the containers that run inside the cluster.

Use standard Kubernetes tools to deploy applications

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: quiz-frontend
spec:
  replicas: 3
  template:
    spec:
      containers:
        - name: quiz-frontend
          image: us-docker.pkg.dev/...
          ports:
            - containerPort: 8080
```

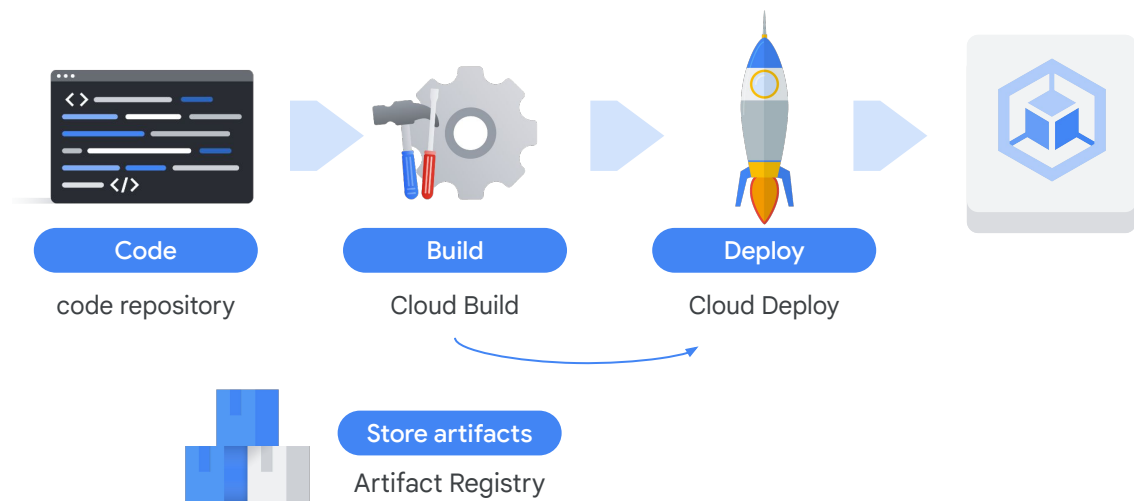
```
$ kubectl create -f ./quiz-frontend.yaml
```

You deploy and manage your containerized application for GKE the same way you would for any other Kubernetes environment. You can use the “kubectl” command to perform most operational tasks.

Although you can deploy ad hoc resources directly by using kubectl commands, the recommended best practice is to use YAML manifest files to define configurations. These files define the properties of the containers that are used for the components in your applications. Manifest files can also define the network services, security policies, and other Kubernetes objects that are used to deliver resilient, scalable, containerized applications.

Applications can be deployed by using Deployments, where Kubernetes continually ensures that a specified number of replicas for a Pod or set of Pods is running. The deployment shown here is for stateless components. You can also use StatefulSets for applications where you need persistent storage. You can also use YAML manifests to define a range of other resource types.

Use GKE as a part of your CI/CD pipeline



As a part of a continuous integration and delivery (CI/CD) pipeline, you can generate a new Docker image for each code commit. The CI/CD pipeline can automatically deploy the image to development, test, and production environments. Cloud Build, Artifact Registry, Cloud Deploy, and GKE can be used to create a strong CI/CD system.

Agenda

01	Compute Engine
02	Google Kubernetes Engine
03	Cloud Run
04	Cloud Functions
05	Comparisons



Next is Cloud Run. Cloud Run is a fully managed compute platform that allows you to run request or event-driven stateless workloads without having to worry about servers.

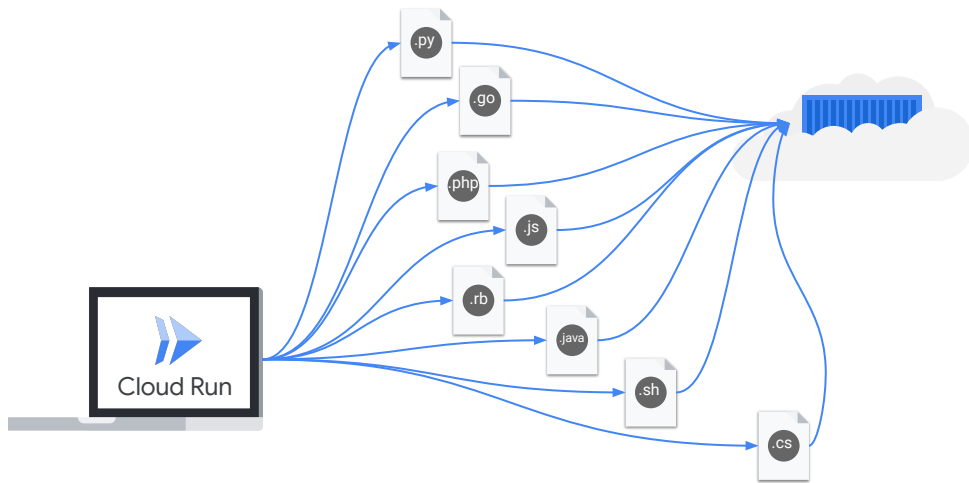
Cloud Run lets you focus on development



It abstracts away all infrastructure management such as provisioning, configuring, and managing servers, so you can focus on writing code. It automatically scales up and down from zero, almost instantaneously, depending on traffic, so you never have to worry about scale configuration. Cloud Run also charges you only for the resources you use, rounded up to the nearest 100 milliseconds, so you never pay for overprovisioned resources.

[Cloud Run: <https://cloud.google.com/run/>]

Cloud Run doesn't restrict the way you code



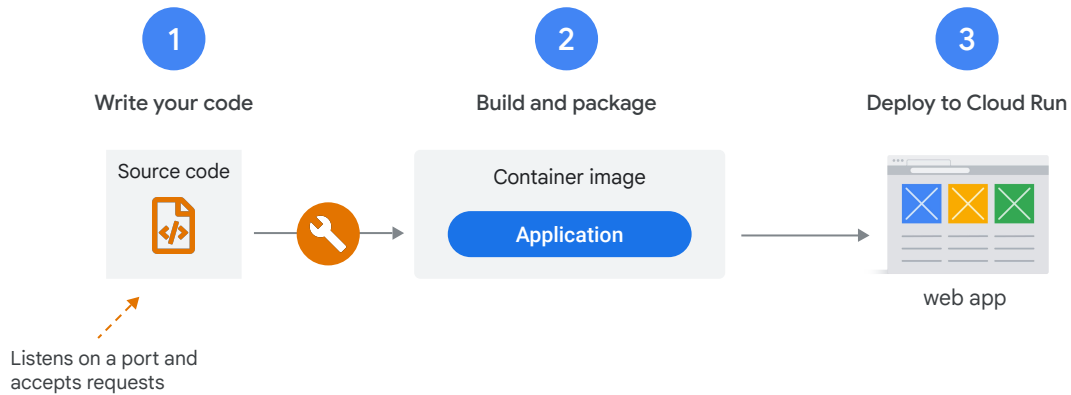
Many serverless platforms add constraints around support for languages and libraries, or even restrict the way you code. Cloud Run enables you to write code your way by letting you easily deploy any stateless containers that listen for requests or events delivered over HTTP. Containers offer flexibility and portability of workloads. With Cloud Run, you can build your applications in any language, using whatever frameworks and tools you want, and deploy them in seconds.

Deploy containerized applications with a single command

```
gcloud run deploy  
  --image us-docker.pkg.dev/my-proj/helloworld  
  --platform managed  
  ...
```

You can use a single command to deploy containerized applications. After that, Cloud Run horizontally scales your container image automatically in order to handle received requests, then scales it down when demand decreases. You only pay for the CPU, memory, and networking that are consumed during request handling.

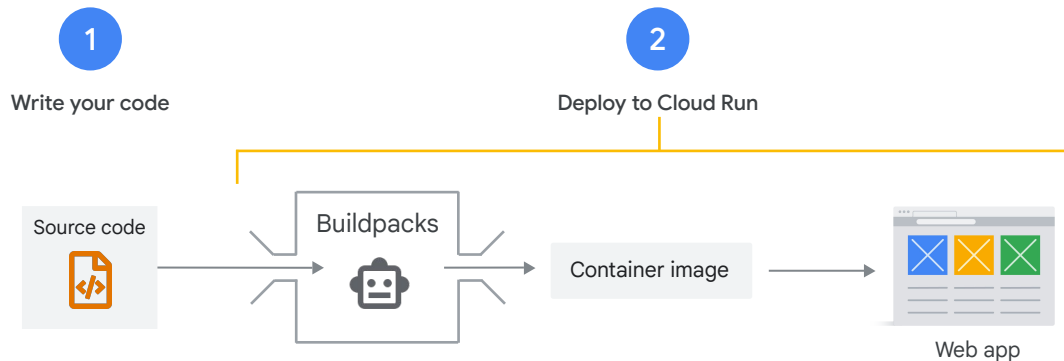
Cloud Run development workflow is a three-step process



The Cloud Run developer workflow is a straightforward three-step process.

- First, use your favorite programming language to write your application. This application should listen for web requests.
- Second, build and package your application into a container image.
- Finally, deploy the container image to Cloud Run.
When you deploy your container image, you get a unique HTTP(S) URL. Cloud Run starts your container on demand to handle requests and dynamically adds and removes containers to ensure capacity to handle all incoming requests.

Cloud Run also has a source-based workflow



When you build a container image, you have full control over how the software is built and how every file is added. Sometimes you need this control.

However, building an application is difficult enough already, and sometimes you just want to turn source code into an HTTPS endpoint. You want to ensure that your container image is secure, well configured, and built in a consistent way.

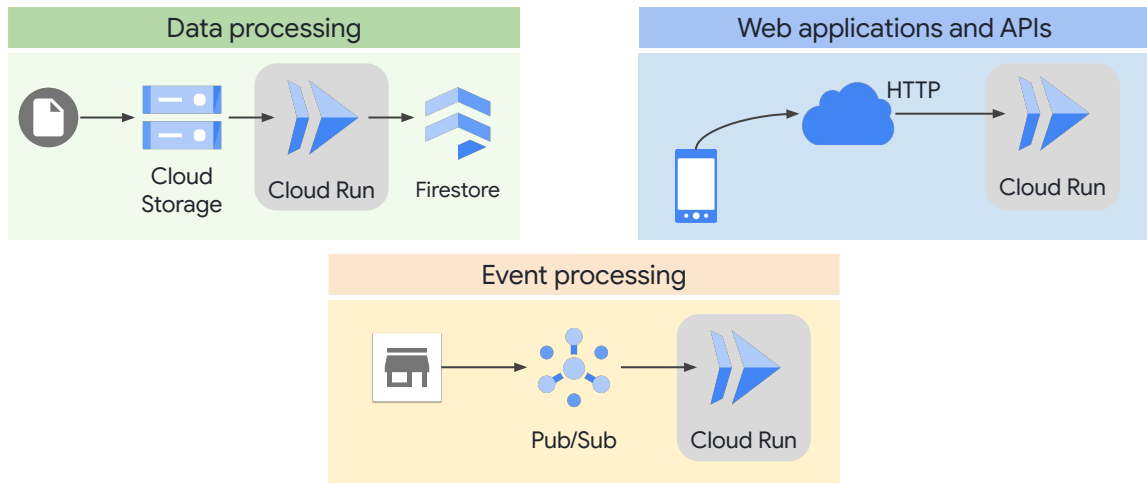
With Cloud Run, you can choose either approach. You can use a container-based workflow or a source-based workflow.

When you use the source-based approach, you deploy your source code instead of a container image. Cloud Run then builds your source and packages the application into a secure container image for you automatically.

Cloud Run uses buildpacks to automatically build container images. Buildpacks transform your application source code into container images that can run on any cloud. Google Cloud buildpacks are also used for the internal build system for Cloud Functions and App Engine.

[Cloud Native Buildpacks: <https://buildpacks.io/>
Google Cloud buildpacks repository:
<https://github.com/GoogleCloudPlatform/buildpacks>]

Build web-and event-based applications

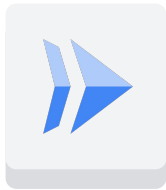


Cloud Run can be used for many use cases.

- Cloud Run is an excellent choice for data processing applications. Because you pay for usage—not provisioned resources—you only pay when data is being processed.
- Cloud Run can also host small or large web applications and web APIs. Cloud Run scales up when necessary and scales down to zero when it's not in use.
- Jobs that must be run in response to Pub/Sub or Eventarc events are good candidates for Cloud Run.

[Can Cloud Run handle these 9 workloads: <https://www.youtube.be/R2b7aZTRf-c>]

Cloud Run job



Cloud Run
job

- ✓ Runs once, in a workflow, or on a schedule
- ✓ Exits when finished
- ✓ Contains one or more independent tasks
- ✓ Runs tasks in parallel and auto-retries failures
- ✓ Uses one container image per task
- ✓ Is fully serverless

Cloud Run jobs work differently from HTTP Cloud Run services.

A Cloud Run job doesn't listen for and serve HTTP requests. There is no need to listen on a port or start a web server. Instead, the job is executed as a one-off task or as part of a workflow. You can also use Cloud Scheduler to run a job on a regular schedule.

When a Cloud Run job finishes, the job exits.

Each job can be composed of a single task or multiple independent tasks.

Because multiple tasks within a job are independent, the tasks can be run in parallel. Each task execution is aware of its task index, which is provided to the task in an environment variable. In addition, tasks that fail can be automatically retried. The maximum number of parallel tasks can be specified so that you don't overwhelm backend services with too many concurrent tasks.

Each task within a job runs a single container image. This container runs to completion.

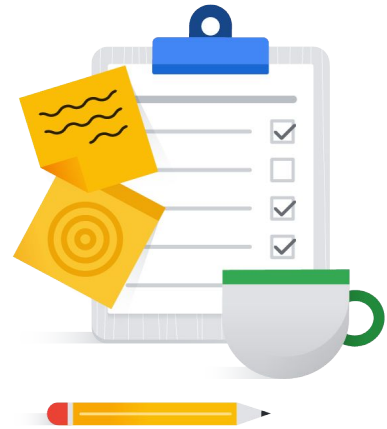
Like Cloud Run services, Cloud Run jobs run on a fully serverless platform. You don't

need to manage any infrastructure to run your jobs.

[Create jobs: <https://cloud.google.com/run/docs/create-jobs>]

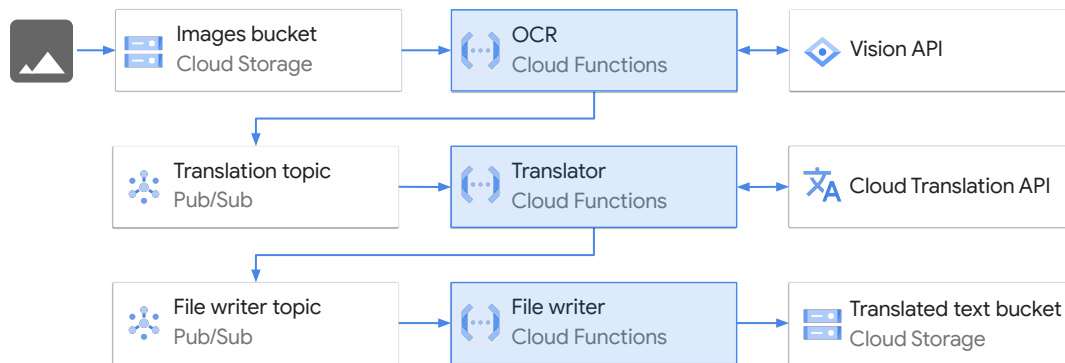
Agenda

01	Compute Engine
02	Google Kubernetes Engine
03	Cloud Run
04	Cloud Functions
05	Comparisons



Next, we look at Cloud Functions. With Cloud Functions, you can develop an application that is event-driven, serverless, and highly scalable.

Develop event-driven, serverless, highly scalable microservices with Cloud Functions



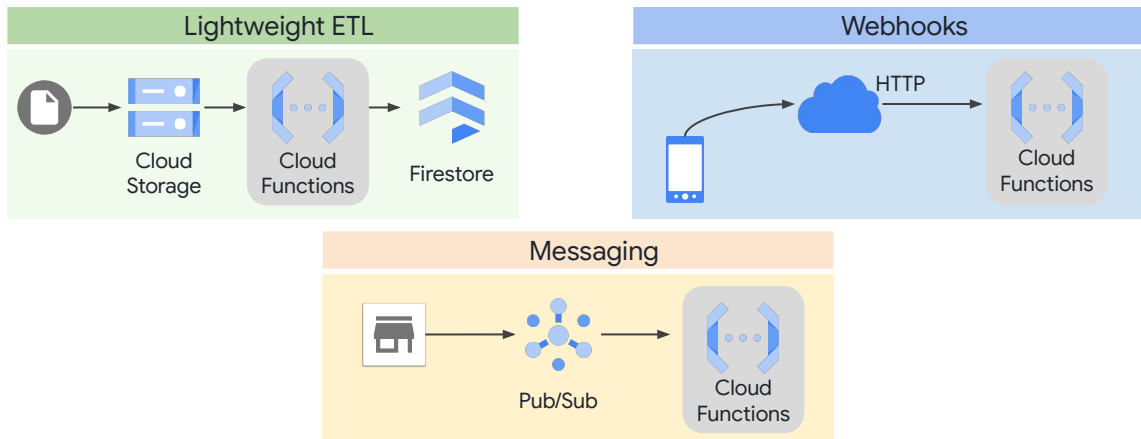
Each function is a lightweight microservice that allows you to integrate application components and data sources. Cloud Functions is ideal for microservices that require a small piece of code to quickly process data in response to an event.

Cloud Functions is priced according to how long your function runs, the number of invocations, and the resources that you provision for the function.

In this example, all components of the application use fully managed services: Cloud Storage, Pub/Sub, Cloud Functions, the Vision API, and the Cloud Translation API. These services scale automatically depending on the volume of incoming data and required compute capacity. This scalability and reliability allows you to focus on your application code.

[Cloud Functions: <https://cloud.google.com/functions/>]

Enable event-driven processing or develop lightweight microservices



Cloud Functions is appropriate for many use cases.

- You can use Cloud Functions for lightweight extract-transform-load, or ETL, operations or for processing messages that are published to a Pub/Sub topic.
- Cloud Functions can also serve as a target for webhooks, which allow applications or services to make direct HTTP calls to invoke microservices.
- Any lightweight functionality that is run in response to an event is a candidate for Cloud Functions.

Focus on code: Cloud Client Libraries

index.js

```
/**
 * Triggered from a message on a Cloud Pub/Sub topic.
 *
 * @param {!Object} event The Cloud Functions event.
 * @param {!Function} The callback function.
 */
exports.subscribe = function subscribe(event, callback) {
  // The Cloud Pub/Sub Message object.
  const pubsubMessage = event.data;

  // We're just going to log the message to prove that
  // it worked.
  console.log(Buffer.from(pubsubMessage.data, 'base64').toString());

  // Don't forget to call the callback.
  callback();
};
```

package.json

```
{
  "name": "sample-pubsub",
  "version": "0.0.1"
}
```

Cloud Functions lets you focus on code. Let's look at Node.js as an example. When you use the Node.js runtime, your function's source code must be exported in a Node.js module.

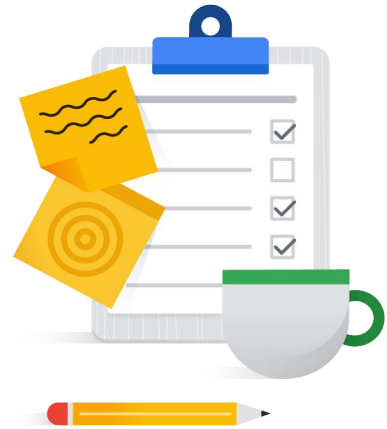
You do not need to upload zip files with packaged dependencies. You can specify any dependencies for your Node.js Cloud Function in a package.json file. The Cloud Functions service automatically installs all dependencies before running your code.

You can use Cloud Client Libraries to programmatically interact with other Google Cloud services.

Cloud Functions currently supports Node.js, Python, Go, Java, .NET, Ruby, and PHP.

Agenda

01	Compute Engine
02	Google Kubernetes Engine
03	Cloud Run
04	Cloud Functions
05	Comparisons



Now we compare the platforms to understand their relative strengths.

Where should I run my applications?



Compute Engine



Google Kubernetes
Engine (GKE)



Cloud Run

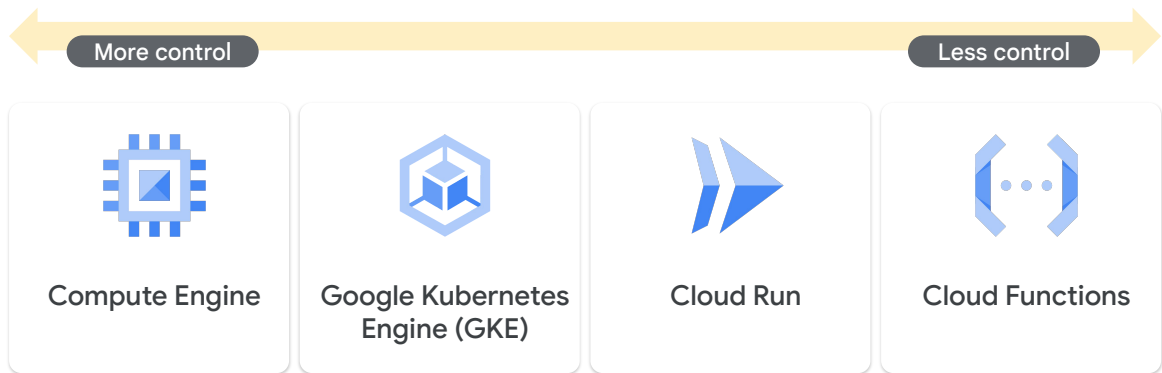


Cloud Functions

After we've talked about those platforms, the question remains: where should I run my applications?

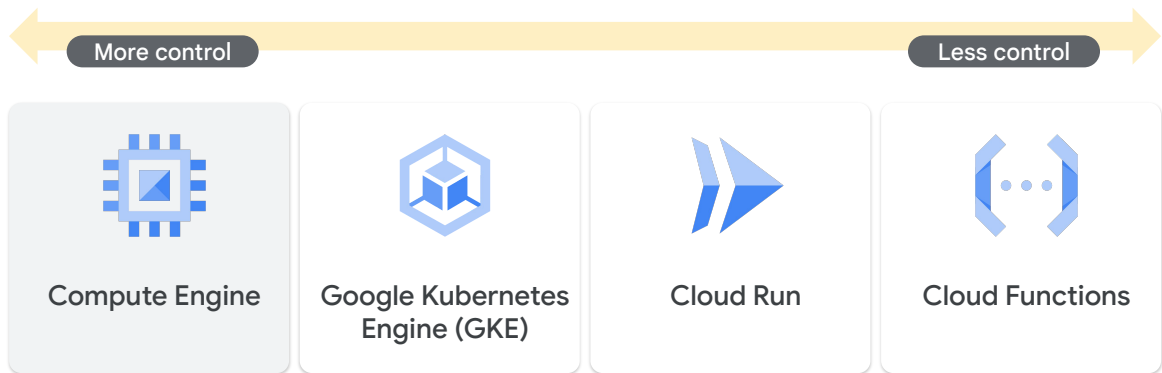
And the answer is still "it depends."

How much infrastructure control do I need?



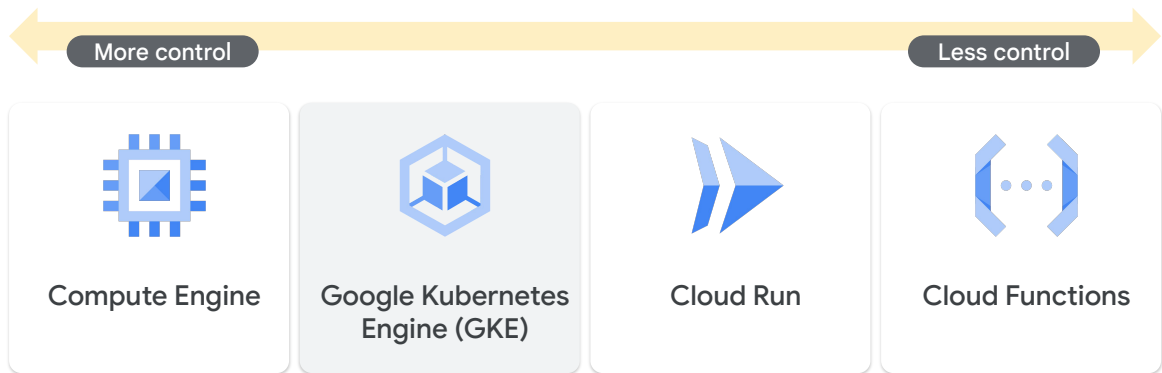
The first question that you might ask is “how much infrastructure control do I need?”

How much infrastructure control do I need?



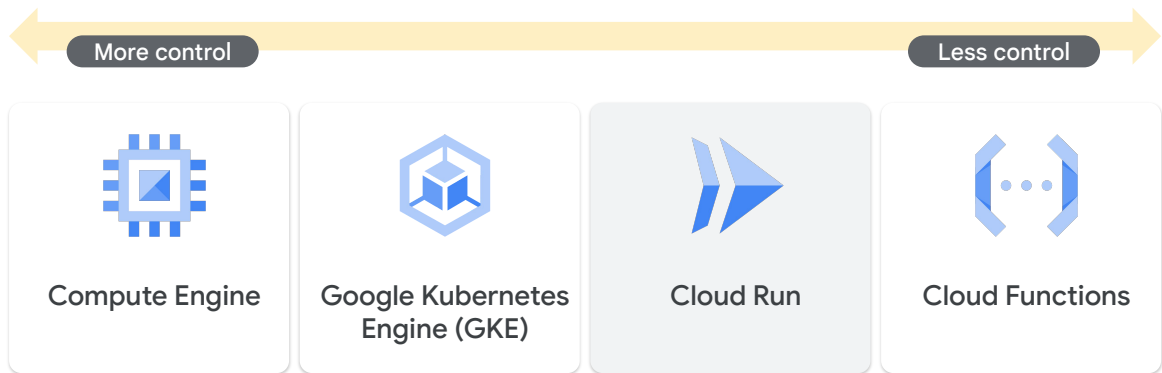
If you want to lift-and-shift legacy systems to the cloud, or you have specific licensing requirements that depend on specific hardware, you might need to use Compute Engine.

How much infrastructure control do I need?



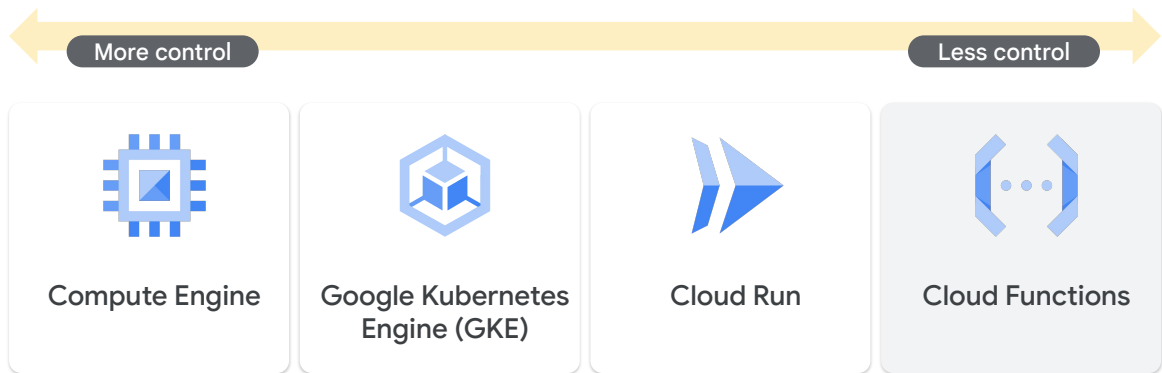
If you can run containers and have hybrid systems on multiple clouds or data centers, or if you have applications that are not HTTP-based, Google Kubernetes Engine might be the right choice.

How much infrastructure control do I need?



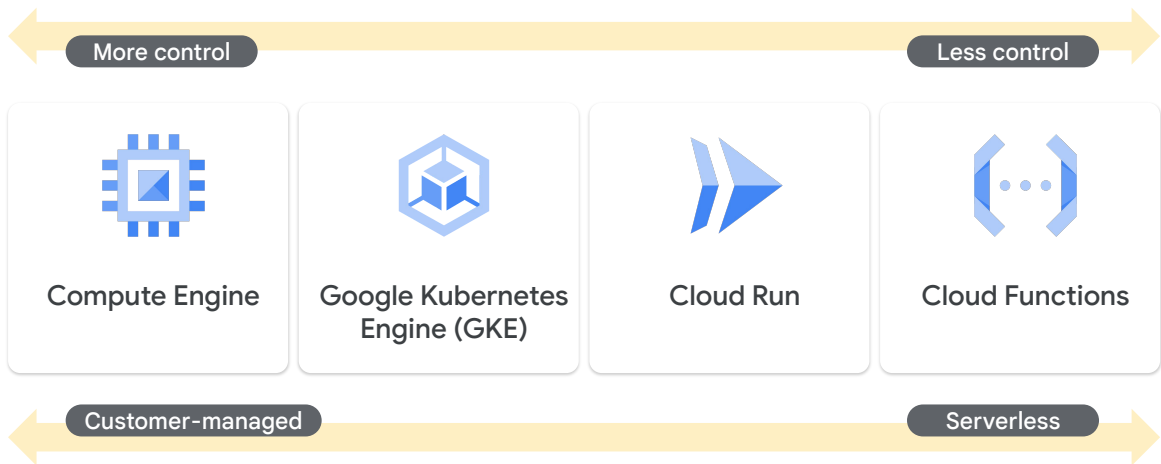
If you want to run stateless containers but not manage the infrastructure at all, Cloud Run might be the best choice.

How much infrastructure control do I need?



And if you just need to write event-driven functions to connect cloud services, Cloud Functions is probably the correct choice.

How much infrastructure control do I need?



Gaining more control over the infrastructure requires more operational effort. When you create a Compute Engine virtual machine, you control the updates for the operating system and software. With GKE, Google manages the virtual machine nodes for your cluster, but you still manage the size of the cluster and decide how to scale each application within the cluster. Cloud Run and Cloud Functions are serverless. For those platforms, you just need to deploy your application, and Google manages the infrastructure and autoscaling.

How are my teams structured?



Compute Engine



Google Kubernetes
Engine (GKE)



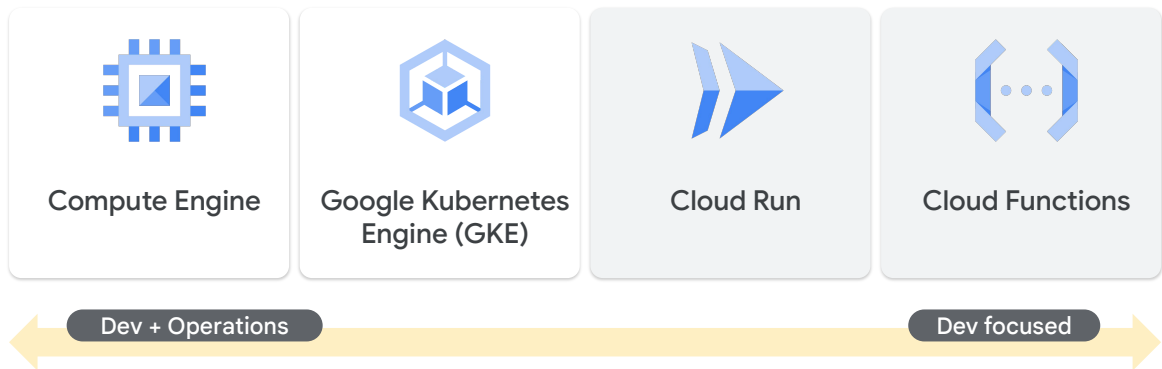
Cloud Run



Cloud Functions

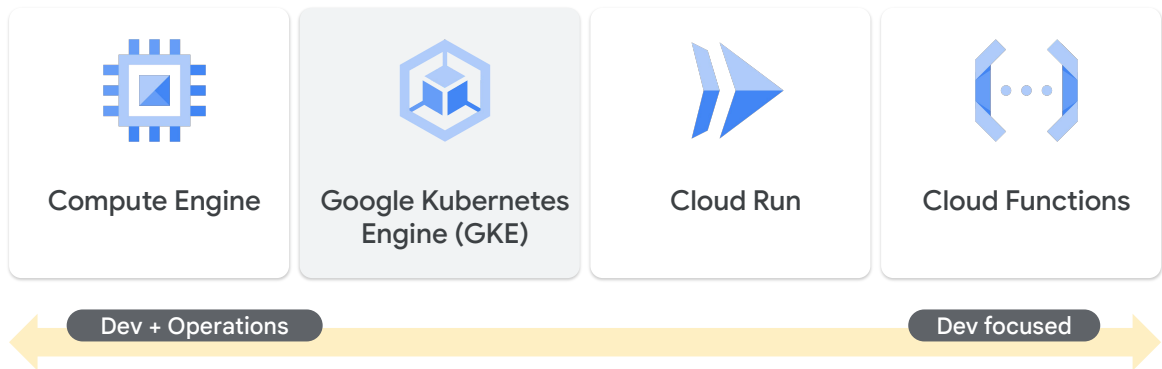
Another question is “how are my teams structured?”

How are my teams structured?



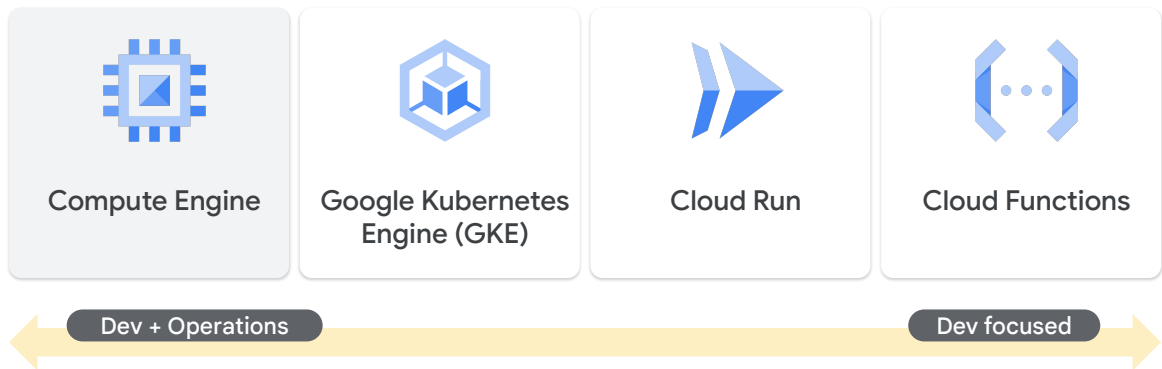
If your teams are mostly developer-focused, Cloud Run and Cloud Functions are probably best for you. If you have both developers and operations teams, you might still use Cloud Run and Cloud Functions when appropriate.

How are my teams structured?



You might also decide to use Google Kubernetes Engine to integrate with your hybrid systems and have more control over your workloads. GKE also lets you use GPUs, TPUs, and non-HTTP network protocols, which are not available for Cloud Run and Cloud Functions.

How are my teams structured?



If you're modernizing your applications over time, you might need to manage Compute Engine VMs that have been migrated from on-premises data centers. Your operations team must be able to manage the health and security of these virtual machines.

What pricing model do I want?



Compute Engine



Google Kubernetes
Engine (GKE)



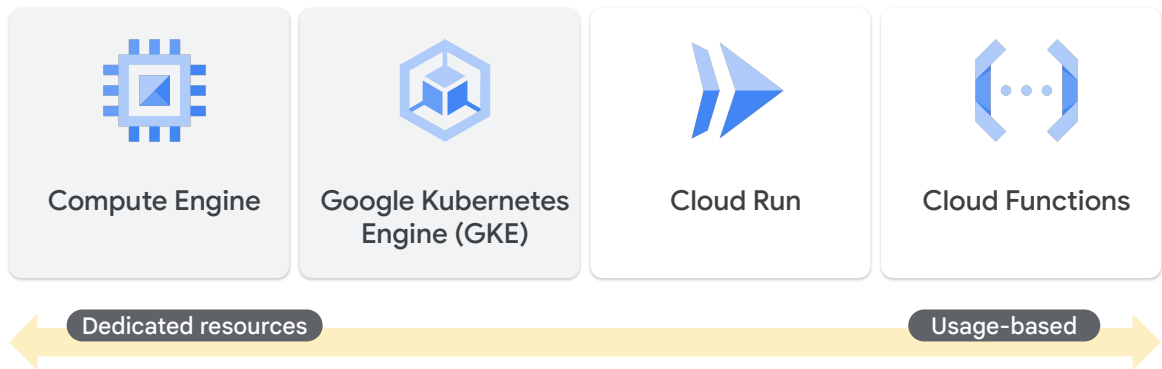
Cloud Run



Cloud Functions

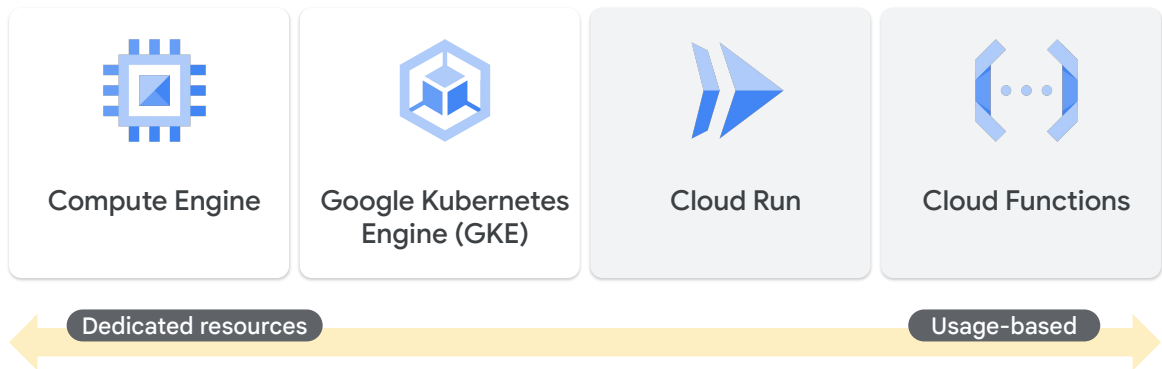
Pricing for the platforms is different, which might affect your choice as well.

What pricing model do I want?

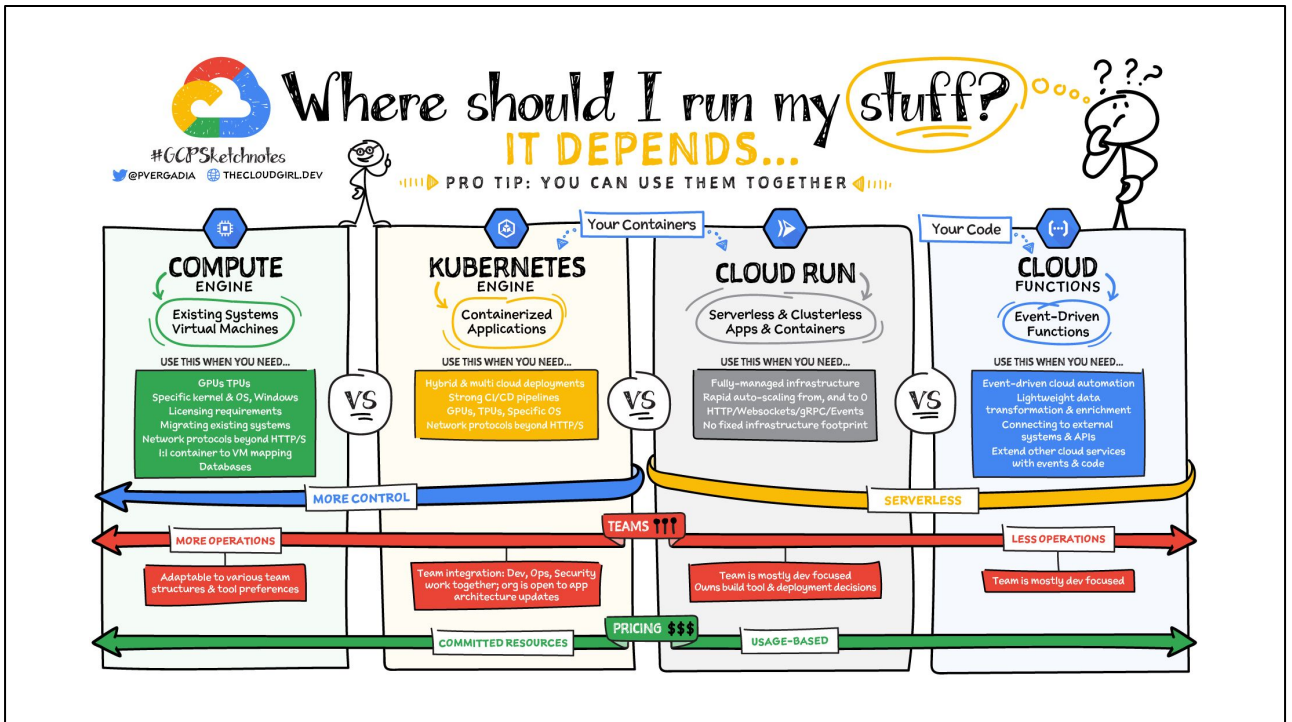


Compute Engine and Google Kubernetes Engine charges are based on the usage of dedicated VMs. The charges are predictable, and these platforms can be ideal when you require consistent capacity for your applications.

What pricing model do I want?



Cloud Run and Cloud Functions are pay per use, which can result in significant savings, especially when your traffic patterns are inconsistent.



This sketch note, titled "Where should I run my stuff," provides a visualization of the relative strengths of the platforms. In general, as you move toward the left, you have more control over your application and infrastructure, but managing that infrastructure requires more operational cost and effort. The two platforms on the right are serverless, which requires much less operational burden and lets you focus on the application instead of the infrastructure. Different applications have different requirements, and you can choose the platform that best meets your requirements.

You don't need to standardize on a single platform, even within a single application. Larger applications might benefit from using multiple platforms that allow them to solve each problem with the correct tool.

Applications that don't have unusual infrastructure requirements can typically be run on multiple platforms. If you use the Cloud Client Libraries in your code, moving code from one platform to another is usually easy. Containerized applications can be run on both Cloud Run and Google Kubernetes Engine. When you don't want to manage the containers, you can deploy your code to Cloud Functions or deploy to Google Kubernetes Engine or Cloud Run by using buildpacks.

The ease of running the same application on different platforms means that you don't have to worry too much about your decision. For example, you might initially decide to

run your application on Cloud Run, with Google managing the infrastructure and scaling. Later, when your application starts getting more traffic, you might decide to run the same application on Google Kubernetes Engine for more control.

There are similar sketch notes for many products and product comparisons, and you may find them helpful for understanding the benefits and usage of different Google Cloud products. You can see more sketch notes at thecloudgirl.dev.

[Where should I run my stuff? (blog):

<https://cloud.google.com/blog/topics/developers-practitioners/where-should-i-run-my-stuff-choosing-google-cloud-compute-option>

Sketch notes: <https://thecloudgirl.dev>]

What about App Engine?

	App Engine (standard environment)	App Engine (flexible environment)
Code in any language?	Supported versions of supported languages	Yes
Required to use a container?	No	Yes
Scaling time	Seconds	Minutes
Scales to zero?	Yes (after 15 minutes)	No
Pricing model	Pay for computing instances (15 minutes after the last request)	Pay for computing instances (granularity of 1 minute)

We've talked about four platforms, but haven't yet talked about App Engine.

App Engine is a fully managed serverless compute option used to build and deploy low latency, highly scalable web applications.

App Engine supports two environments: standard and flexible.

What about App Engine?

	App Engine (standard environment)	App Engine (flexible environment)
Code in any language?	Supported versions of supported languages	Yes
Required to use a container?	No	Yes
Scaling time	Seconds	Minutes
Scales to zero?	Yes (after 15 minutes)	No
Pricing model	Pay for computing instances (15 minutes after the last request)	Pay for computing instances (granularity of 1 minute)

The App Engine standard environment runs your code in a sandbox, and doesn't require you to build containers. However, your applications must be written with specific versions of supported languages.

The standard environment responds well to spikes in traffic by scaling up within seconds and scaling down to zero after 15 minutes of inactivity.

You pay based on the computing instances that are running your application, but do not need to manage those instances. You pay nothing when your application scales down to zero.

The standard environment is a reasonable choice for non-containerized applications with spikes in traffic.

What about App Engine?

	App Engine (standard environment)	App Engine (flexible environment)
Code in any language?	Supported versions of supported languages	Yes
Required to use a container?	No	Yes
Scaling time	Seconds	Minutes
Scales to zero?	Yes (after 15 minutes)	No
Pricing model	Pay for computing instances (15 minutes after the last request)	Pay for computing instances (granularity of 1 minute)

The App Engine flexible environment requires you to create a container for your application, but you gain flexibility by doing so.

Your code can be written by using any languages and libraries.

The flexible environment is better for applications that have sustained traffic, because it scales up and down much slower than the standard environment does, and cannot scale to zero.

Your application will always need to run in at least one instance.

Like the standard environment, you pay for all computing instances that are used by the flexible environment.

What about App Engine?

	App Engine (standard environment)	App Engine (flexible environment)	Cloud Run
Code in any language?	Supported versions of supported languages	Yes	Yes
Required to use a container?	No	Yes	Optional (can use buildpacks)
Scaling time	Seconds	Minutes	Almost immediately
Scales to zero?	Yes (after 15 minutes)	No	Yes (almost immediately)
Pricing model	Pay for computing instances (15 minutes after the last request)	Pay for computing instances (granularity of 1 minute)	Pay when requests are being processed (granularity of 0.10 seconds)





Cloud Run provides many of the best features of both App Engine environments.

Cloud Run applications are required to run in containers. If you do not need custom containers, buildpacks can automatically create the containers for you.

Cloud Run can scale up and down almost immediately in response to traffic spikes. Unlike App Engine, you only pay for Cloud Run when you are processing requests, rounded up to the nearest tenth of a second. This pricing model can result in significant cost savings for your application.

App Engine is fully supported, and it works well for creating web applications and web APIs, but Cloud Run is often a better choice for those use cases.

Where should I run my applications?

			
Compute Engine	Google Kubernetes Engine (GKE)	Cloud Run	Cloud Functions
Virtual machines (VMs)	Cluster of VMs that run containers	Serverless containers	Serverless functions

Returning to the question "Where should I run my applications?", the best answer is that you should run each application on the platform that best fits its requirements.

Most applications written with the Cloud Client Libraries can be easily moved from platform to platform, so you can change your decision later. If you do not have complex infrastructure requirements, start with a serverless platform that lets you focus on the application instead of the infrastructure. If you later want more control over the infrastructure, you can move your application to a platform that requires more operational effort but provides the needed control or flexibility.

[Application Hosting Options: <https://cloud.google.com/hosting-options>]

In this module, you learned ...



Google Cloud provides compute options for different application needs and operational control requirements.



We discussed and compared the benefits of **Compute Engine**, **GKE**, **Cloud Run**, and **Cloud Functions**.



Cloud Client Libraries are the recommended way to programmatically interact with Google Cloud services.

Google Cloud offers a range of compute options depending on the needs of your application and the level of operational control that you require.

We compared the benefits of **Compute Engine**, **GKE**, **Cloud Run**, and **Cloud Functions**.

Cloud Client Libraries are the recommended way to programmatically interact with Google Cloud services. With this approach, you can move your application to a different platform when your needs change.