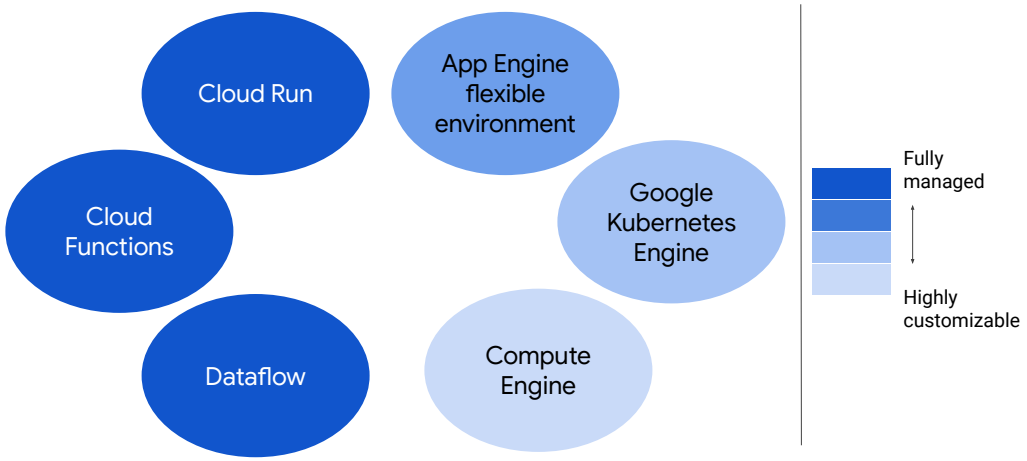
Execution Environments for Your Application

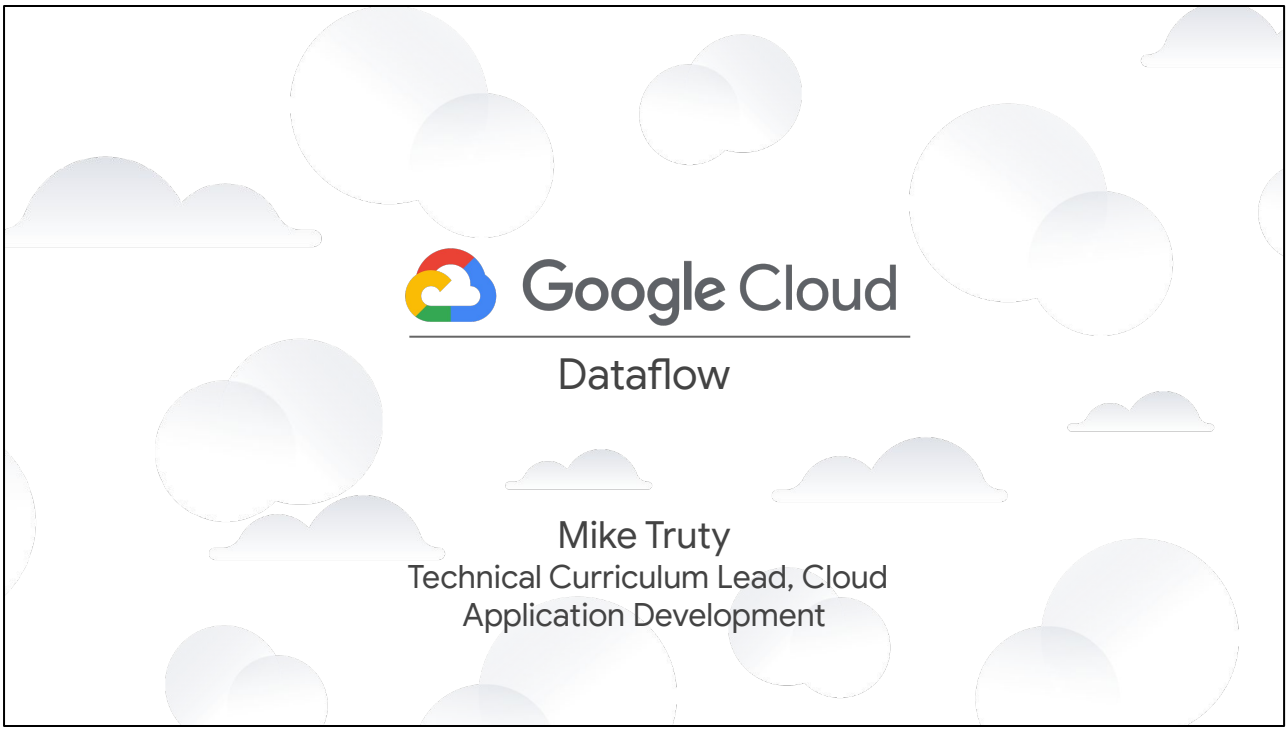
Mike Truty
Technical Curriculum Lead, Cloud Application Development

One of the key themes of this course is that you're not boxed in with respect to where you can run your application. Google Cloud has a range of application execution environments. You can choose an application environment that matches the needs of your application as well as the level of control that you want to have over the infrastructure. Remember, more control over the infrastructure usually implies greater operational burden. If your needs change, it isn’t a problem. As long as you use Cloud Client Libraries in your application to work with Google Cloud services, you can usually move to another execution environment without having to rework your application. In the module execution environments for your application, you'll learn which use cases are most appropriate for a particular execution environment and when you should consider other options. We'll discuss execution environments including Dataflow, Cloud Functions, App Engine, Google Kubernetes Engine (GKE), and Compute Engine.

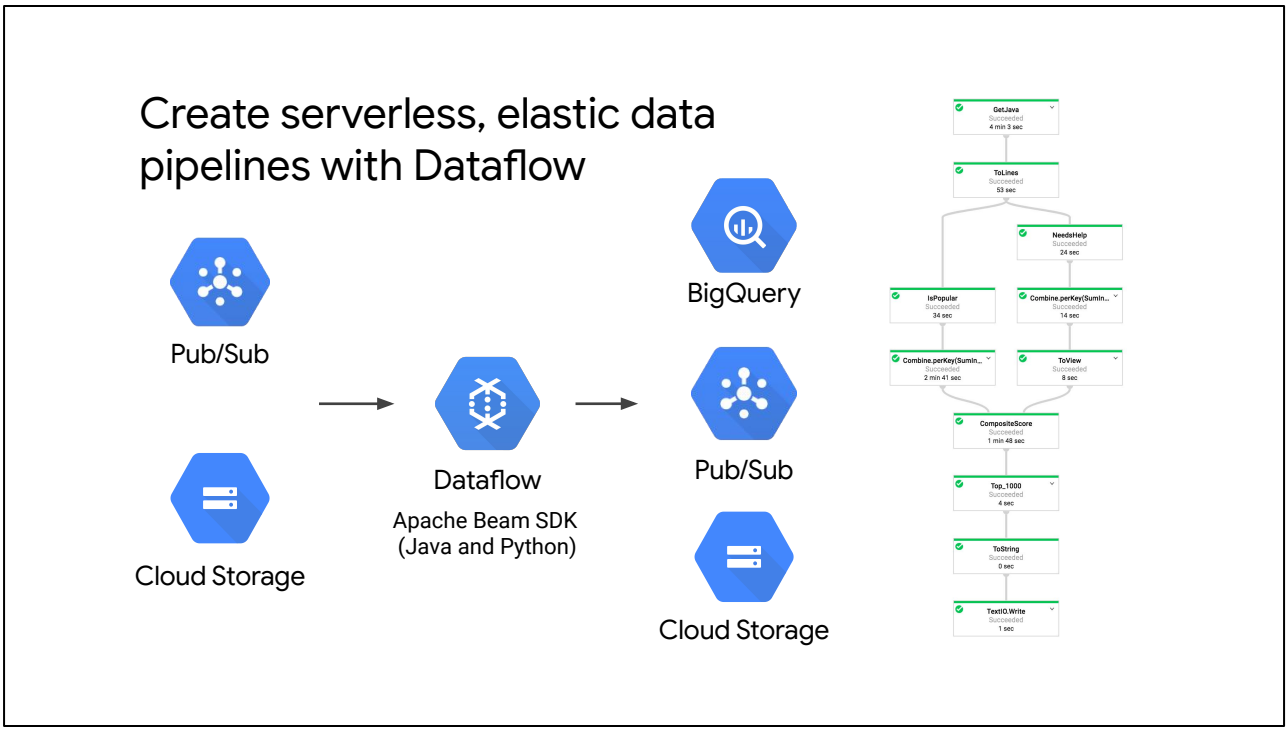
You have a choice of application execution environments



Google Cloud provides a range of options to run your application code. These options range from services like Dataflow and Cloud Functions, which are fully managed, to App Engine, GKE, and Compute Engine, which offer steadily increasing abilities to customize your execution environment. Fully managed compute environments require minimal set up and operations. Highly customizable environments, on the other hand, require greater operational and management effort to keep the application running optimally.



Let's look at Dataflow.



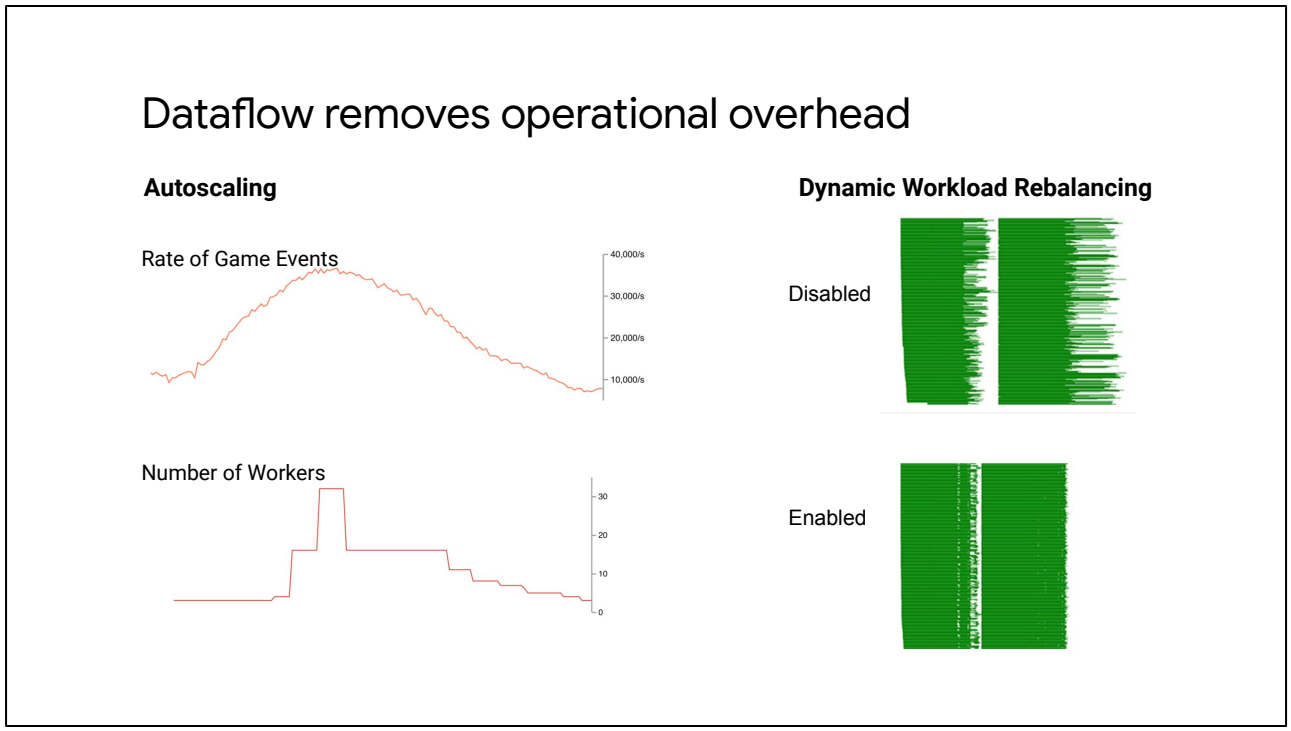
Dataflow is a serverless execution engine, our runner for executing parallel data processing pipelines that are developed using Apache Beam SDKs.

Dataflow supports fast, simplified pipeline development by using expressive Java and Python APIs in the Apache Beam SDK. Dataflow integrates with Google Cloud services for streaming events ingestion using Pub/Sub and data warehousing using BigQuery. It integrates with machine learning APIs and more.

Apache Beam supports Java and Python SDKs to develop pipelines. A pipeline comprises your entire data processing task including reading input data, transforming the data, and writing output data. The Dataflow service uses other managed services such Compute Engine, Cloud Logging, Cloud Storage, BigQuery, Pub/Sub, and the Firestore APIs to execute your pipelines.

Dataflow does a fantastic job of removing operational overhead.

For more information, see:
Dataflow: https://cloud.google.com/dataflow/
Apache Beam: https://beam.apache.org/documentation/



With systems such as Spark and Hadoop, users spend a significant amount of time tuning their jobs. They have to estimate the number of workers to use for their job or cluster. A single configuration might not even be sufficient because resource needs can dynamically change over the lifetime of the job. With Dataflow, you don't need to specify work accounts. Dataflow supports simple configuration and optimizes the work account over time. Dataflow auto scales based on metrics such as CPU utilization, throughput, and the amount of work remaining. It adds workers when CPU utilization and backlog increase and removes them when these metrics decrease.

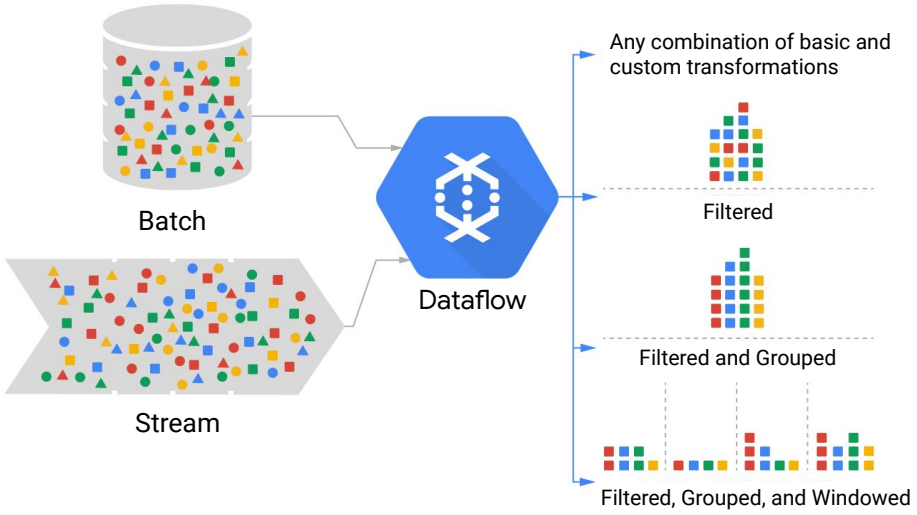
For more information about auto-scaling in Dataflow, see the Downloads and Resources pane.

Dynamic Workload Rebalancing reduces overall completion time and reduces cost by keeping all workers busy. It provides better predictability of overall job completion time. For example, in this image, you see that some workers are still busy, while others have finished. So you don't want stragglers to take up your resources. In this image, the workload is evenly balanced between all the workers. And so all workers finish at approximately the same time.

For more information about autoscaling in Dataflow, see https://cloud.google.com/blog/big-data/2016/03/comparing-cloud-dataflow-autoscaling-to-spark-and-hadoop.

For more information about dynamic workload rebalancing, see https://cloud.google.com/blog/big-data/2016/05/no-shard-left-behind-dynamic-work-rebalancing-in-google-cloud-dataflow.

Use Dataflow for high-volume data processing



Dataflow is ideal for streaming and batch data pipelines. It's great for use cases such as click stream, point of sale, and segmentation analysis in the retail industry, fraud detection in the financial services industry, personalized user experience in the gaming industry, and IoT analytics in manufacturing, healthcare, and logistics industries. Dataflow is ideal for streaming and batch data pipelines.

Consider other compute environments

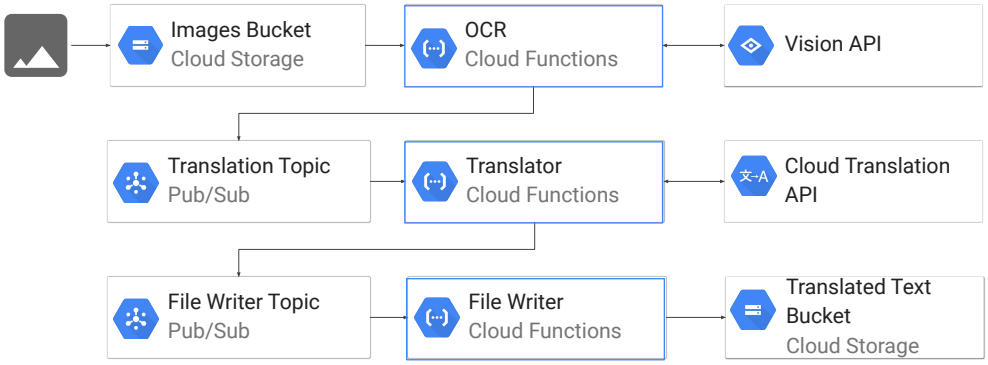
Apache Beam SDK 2.x for Java and Python

So when should you consider other compute environments? The Dataflow SDK 2.x is based on Apache Beam SDK. And remember, this is available in Java and Python programming languages. Consider other compute environments if you need to execute code in other programming languages.

The Dataflow SDK 2.5.0 was the last Dataflow SDK release that was separate from the Apache Beam SDK releases. The Dataflow SDK 2.x for Java and Python are now part of the mainline Apache Beam 2.x SDK.
https://cloud.google.com/dataflow/docs/resources/release-notes-python
https://cloud.google.com/dataflow/docs/resources/release-notes-java-2

Let's talk about Cloud Functions.

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



With Cloud Functions, you can develop an application that is event-driven, serverless, and highly scalable. Each function is a lightweight microservice that enables you to integrate application components and data sources. Cloud Functions are ideal for microservices that require a small piece of code to quickly process data related to an event.

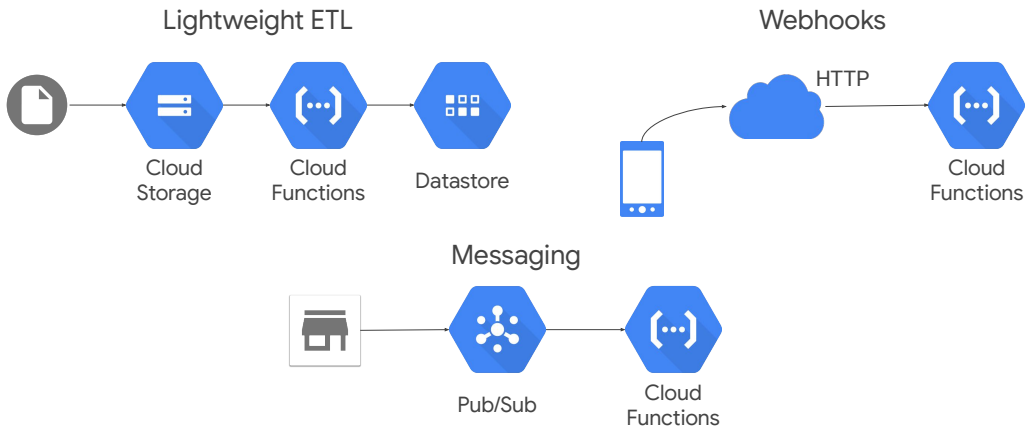
Cloud Functions are priced according to how long your function runs, the number of times it is invoked, and the resources that you provision for the function.

A cloud function can be triggered automatically in response to events like an object being added to a Cloud Storage bucket or a message being published to a Pub/Sub topic.

In this example, all components of the application use fully managed services and APIs such as Cloud Functions, Cloud Storage and Pub/Sub, and the Vision and Cloud Translation APIs. These services scale automatically depending on the volume of incoming requests. This scalability and reliability enables you to focus on your application code.

[For more information about Cloud Functions, see https://cloud.google.com/functions/]

Use Cloud Functions to enable event-driven processing or to develop lightweight microservices



Cloud Functions can be used in a variety of 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, allowing applications or services to make direct HTTP calls to invoke microservices implemented using Cloud Functions.

Any lightweight functionality that is run in response to an event is a candidate for Cloud Functions.

Focus on code: Node.js and 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 allows you to focus on code. Let's look at Node.js as an example. When using 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 will automatically install all dependencies before running your code.

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

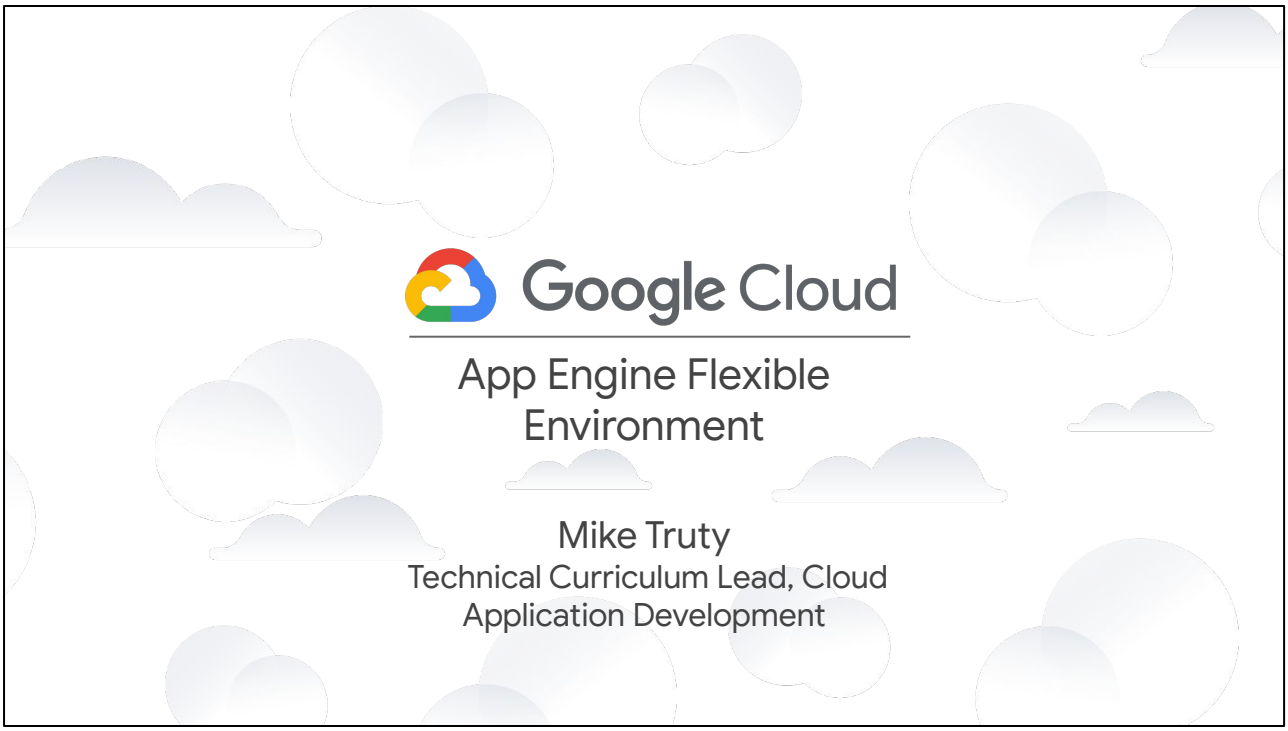
Cloud Functions currently supports four languages: Node.js, Python, Go, and Java.

Consider other compute environments

Large or complex codebase

Languages other than Node.js, Python, Go, or Java

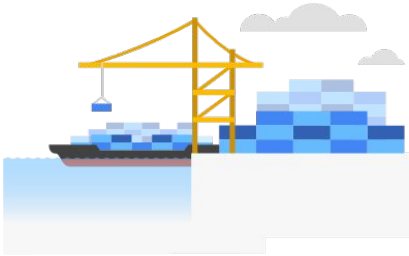
Cloud Functions may not be appropriate for all use cases. You should consider other compute environments if your application or microservice has a large and complex codebase, or if you need to use languages other than Node.js, Python, Go, or Java.



Let's talk about App Engine flexible environment.

Deploy scalable web and mobile backends in any language with App Engine flexible environment

Native runtimes



Custom runtimes

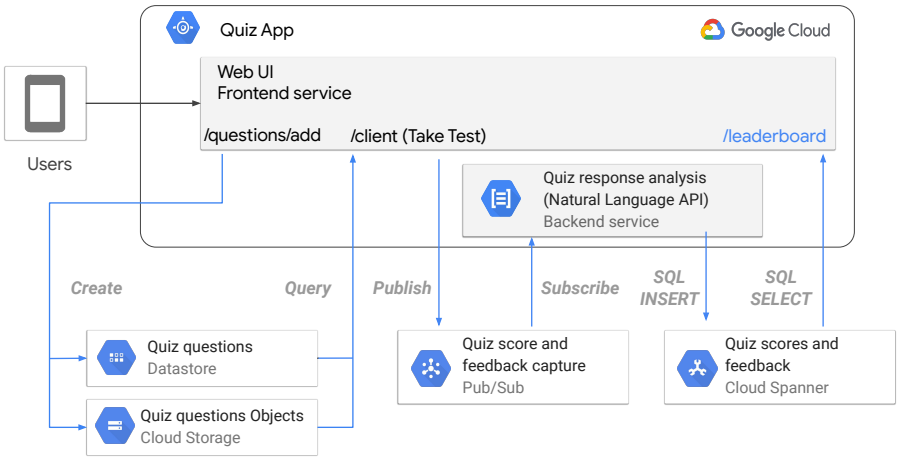
App Engine flexible environment is an excellent option for deploying web applications, backends for mobile applications, HTTP, APIs, and internal business applications.

App Engine flexible environment provides default settings for infrastructure components. You can customize settings such as network, sub-network, port forwarding, and instance tags. SSH access to VM instances in the flexible environment is disabled by default. You can enable route access to the underlying VM instances. App Engine flexible environment runs a Docker image for your application. If needed, you can generate the Docker image of the application and run it in other container-based environments such as GKE.

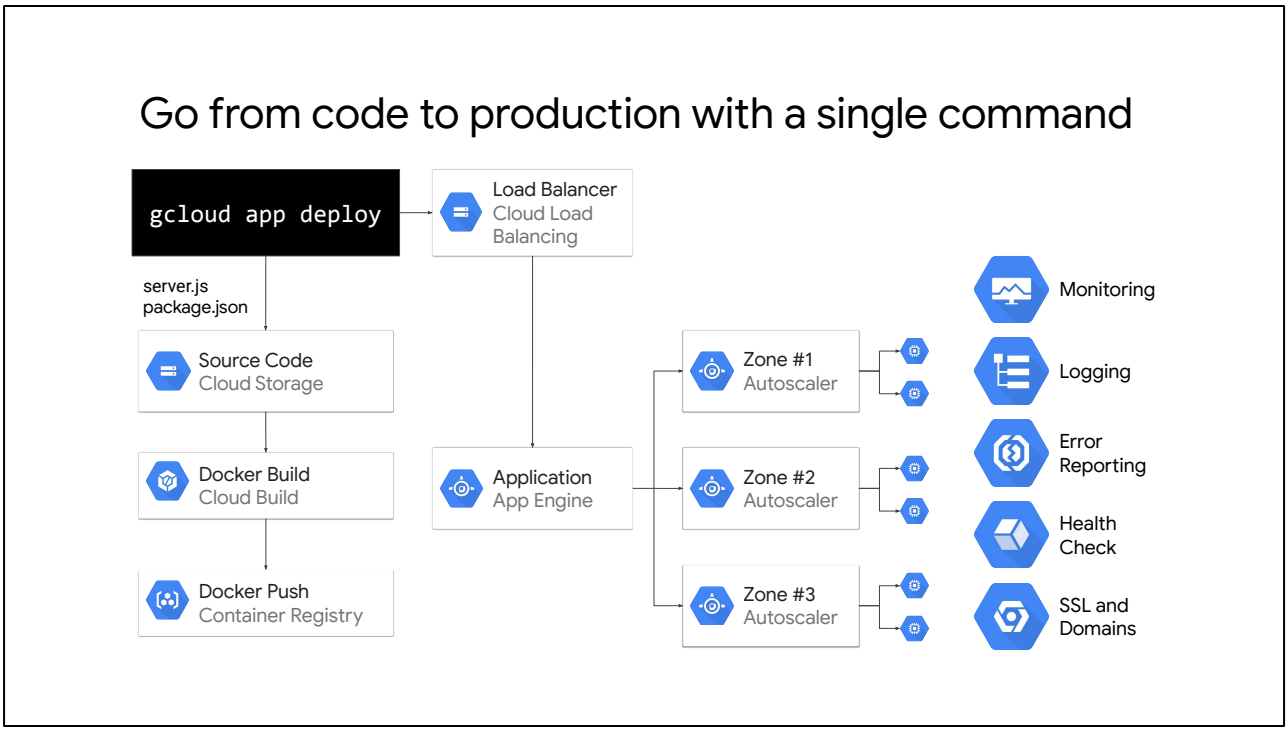
App Engine flexible environment supports Java 8, Python 2.7 and 3.5, Node.js, Ruby, PHP, .NET Core, Go, and other runtimes based on a custom Docker image or open-source stock of file. You can use Cloud Client Libraries to programmatically interact with other Google Cloud services.

Images:
https://cloud.google.com/appengine/
https://cloud.google.com/container-engine/

Develop and deploy microservices



You can use App Engine flexible environment to get started with a microservices architecture for your application. The application shown here deploys a frontend service and a backend service that are loosely coupled through Pub/Sub.



You can go from code to production with a single command. After you develop your application, you can deploy to your test, staging, or production environment with a single command: "`gcloud app deploy`".

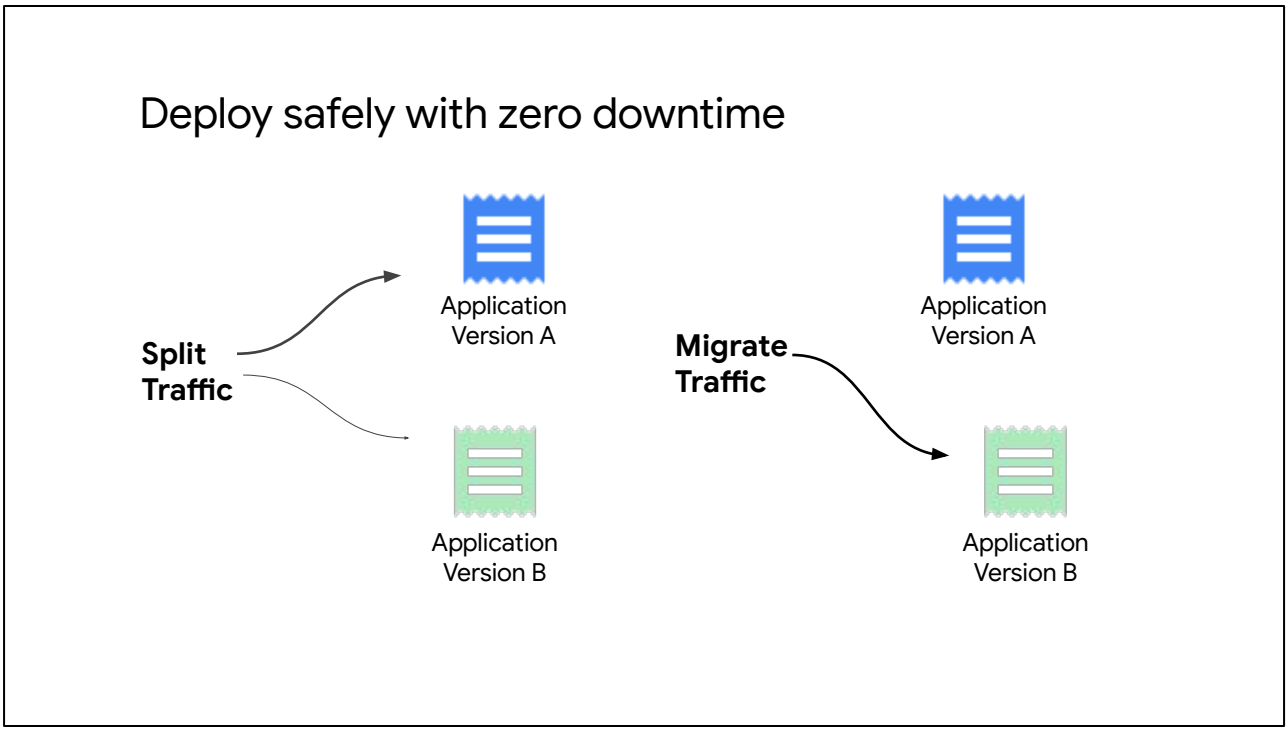
When you run this command, App Engine flexible environment automatically uploads your source code to Cloud Storage, builds a Docker image for your application with the runtime environment, and pushes the image to Container Registry. You don't need to set up any Docker scripts for applications that use one of App Engine's native runtime environments. So what else does App Engine flexible environment do behind the scenes?

App Engine sets up a load balancer and runs your application in three zones to ensure that your application is always up and running. App Engine launches and autoscales Compute Engine instances to run your application and ensure that your application can scale up or down depending on traffic volume.

App Engine also sets up other services that are crucial for application monitoring and management, such as monitoring, logging, error reporting, health checks, and SSL.

App Engine flexible environment enables you to quickly deploy your application without manual infrastructure setup. It automatically applies critical backward

compatible updates and security updates to the underlying operating system. If you need more control, you can directly SSH to the VM instances to work with custom runtimes and more. With App Engine flexible environment, you can deploy safely with zero downtime.



App Engine flexible environment enables you to perform canary testing. You can verify a production release before serving any external traffic.

You can perform AB testing of your application and deploy updates by easily splitting traffic between the current version and the new version. After you verify that the new version works, you can migrate all traffic to the new version without any downtime.

Use App Engine flexible environment for highly scalable web-focused applications

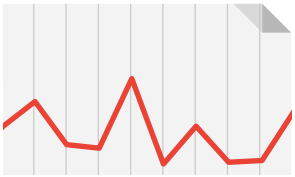
- ✓ Applications that are based on HTTP/s request-responses.
- ✓ Applications that deploy public endpoints.
- ✓ Continuous integration and delivery (CI/CD) pipelines with Jenkins or Spinnaker.

App Engine flexible environment is ideal for highly scalable web-focused applications. You can use App Engine flexible environment for HTTP or HTTPS request responses and applications that deploy public endpoints. You can also implement CI/CD pipelines that use Jenkins or Spinnaker to deploy applications to App Engine.

Consider other compute environments

Protocols other than HTTP/S

Persistent Disks



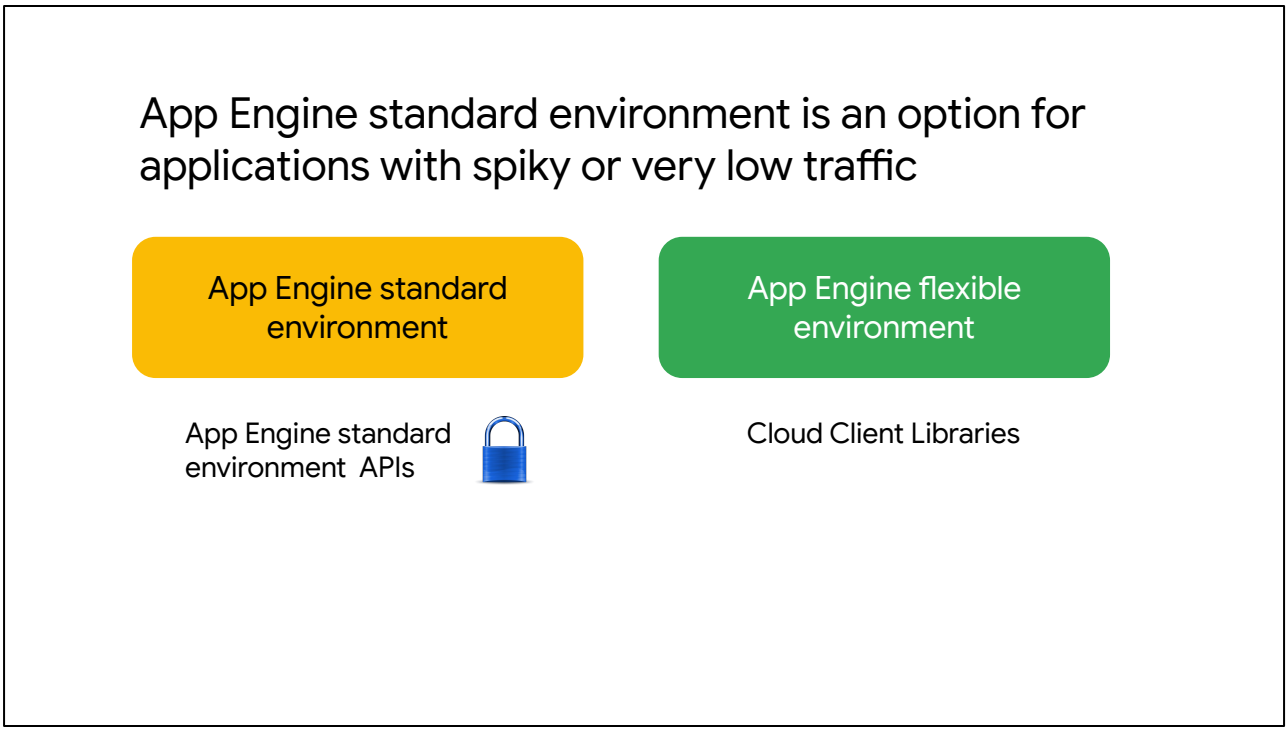
Spiky or very low traffic

When is App Engine flexible environment *not* the right choice for your application? Consider other compute environments such as GKE or Compute Engine if your application needs to support other network protocols besides HTTP or HTTPS.

You can't write data to persistent disks with App Engine. App Engine flexible environment enables you to add volumes to tmpfs. These files are in memory only.

Currently, App Engine flexible environment is not ideal for applications with spiky or very low traffic. At least two instances are running at all times to serve traffic.

Images:
https://pixabay.com/en/success-curve-hand-finger-touch-1093889/
Other images from cloud.google.com



App Engine has another flavor called the App Engine standard environment. If you do have applications that have spiky or very low traffic, App Engine standard environment may be an option for you. Applications that run on App Engine standard environment must use App Engine Standard APIs. App Engine Standard APIs are supported only in the App Engine standard environment. So, if you've built an app using App Engine Standard APIs, you cannot run it on another platform such as App Engine flexible environment, GKE, or Compute Engine.

Applications that are developed using Cloud Client Libraries can be moved to another compute environment such as Cloud Functions, GKE, or Compute Engine if the needs of the application change.

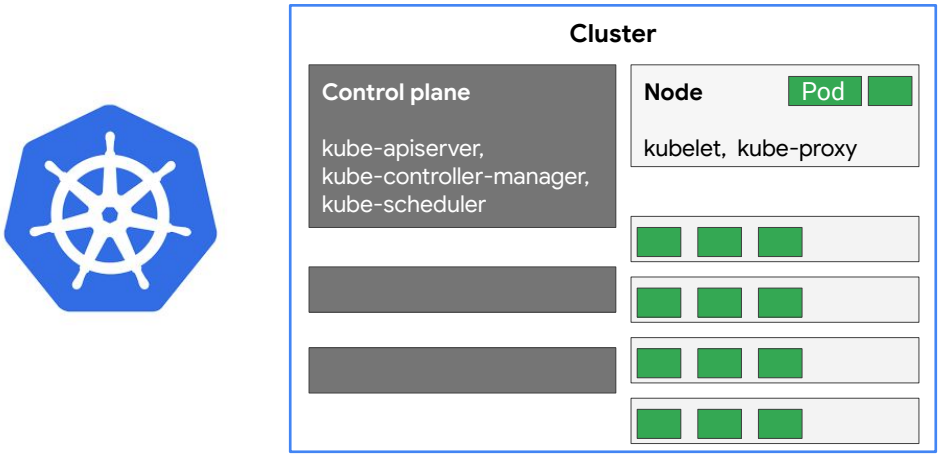
For more information about App Engine standard environment, see the downloads and resources pane.

For more information about App Engine standard environment, see https://cloud.google.com/appengine/docs/standard/.

For more information about App Engine flexible environment, see:
App Engine flexible environment: https://cloud.google.com/appengine/docs/flexible/
You can run that on App Engine?: https://www.youtube.com/watch?v=sATG0OfdP4g

Let's dive into Google Kubernetes Engine.

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



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

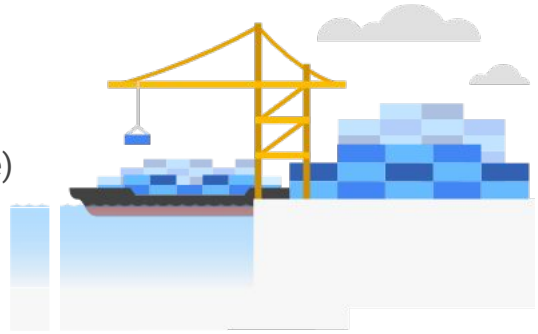
A Kubernetes cluster contains control plane and worker nodes. The nodes in a cluster are the machines, such as virtual machines, physical servers, etc., that run your applications. The Kubernetes control plane manages the worker nodes and the pods in the cluster. A pod is a group of containers that share networking and storage resources on the node.

[For more information, see https://kubernetes.io/docs/home/]

GKE is a managed service for Kubernetes

Google maintains:

- Operating systems
- Nodes (including control plane)
- Monitoring and logging
- Kubernetes upgrades

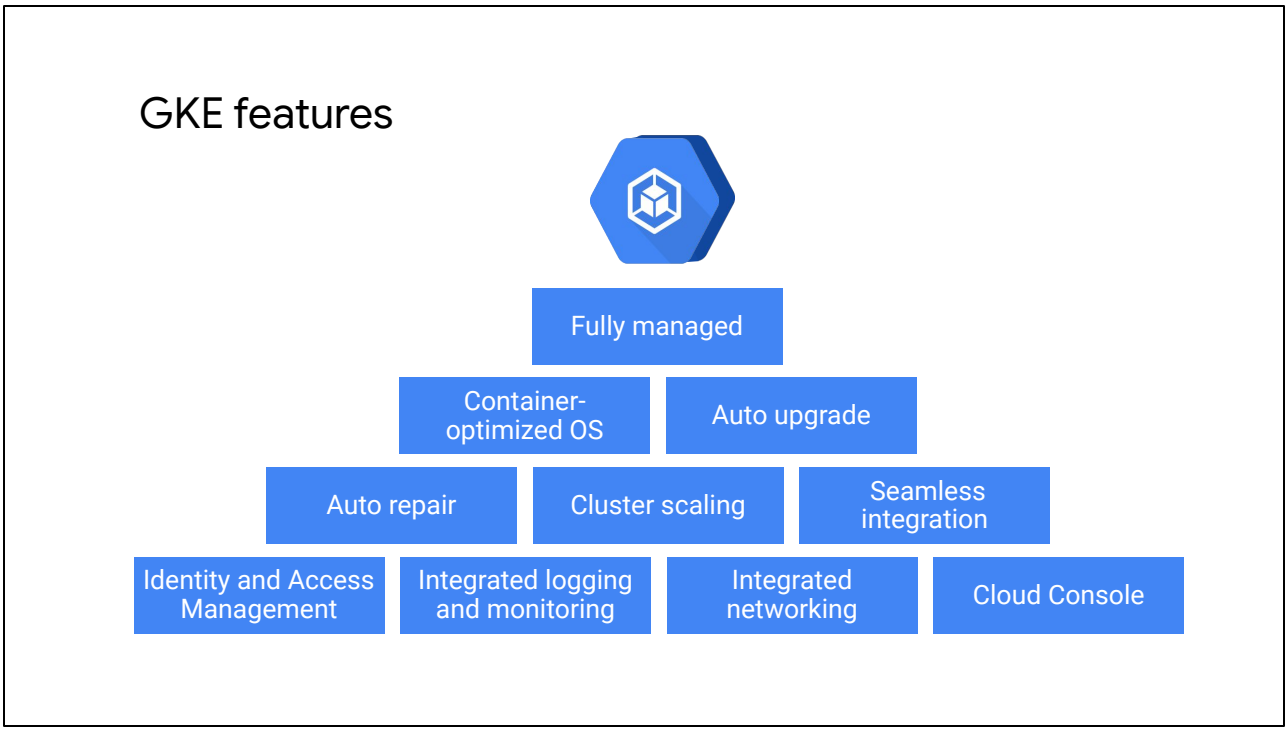


GKE is a managed Kubernetes service on Google infrastructure. GKE helps you to deploy, manage, and scale Kubernetes environments for your containerized applications on Google Cloud.

More specifically, GKE is a component of the Google Cloud compute offerings. It makes it easy to bring your Kubernetes workloads into the cloud.

For more information, see:
Google Kubernetes Engine: https://cloud.google.com/container-engine/
Google Kubernetes Engine Overview:
https://cloud.google.com/container-engine/docs/concepts/container-engine-overview
Upgrading a Container Cluster:
https://cloud.google.com/container-engine/docs/clusters/upgrade

Google Kubernetes Engine - The easiest way to use containers in production (Google Cloud Next '17): https://youtu.be/_yk1tTHYBvg



GKE is fully managed, which means you don't have to provision the underlying resources.

GKE uses a container-optimized operating system to run your workloads. These operating systems are maintained by Google and are optimized to scale quickly with a minimal resource footprint.

When you use GKE, you start by directing the service to instantiate a Kubernetes system for you. This system is called a cluster. GKE's 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 repair unhealthy nodes for you. It makes 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, allowing workloads to gracefully exit. It will then recreate the node. Just as Kubernetes supports scaling workloads, GKE supports scaling of the cluster itself.

GKE seamlessly integrates with Cloud Build and Container Registry. This allows you to automate deployment using private container images that you have securely stored

in Container Registry.

GKE also integrates with Cloud Identity and Access Management, which allows you to control access through the use of accounts and role permissions.

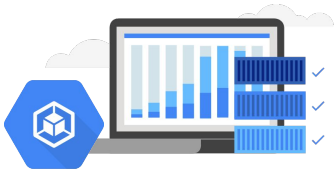
GKE uses Cloud Monitoring and Cloud Logging to help you understand your applications' performance and behavior.

GKE is integrated with Virtual Private Clouds, making use of Google Cloud's networking features.

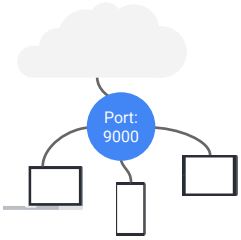
And finally, the Cloud Console provides insights into GKE clusters and their resources and allows you to view, inspect and delete resources in those clusters.

Use GKE for complex, portable applications

Any application runtime packaged as a Docker container image



Hybrid or multi-cloud applications



Protocols other than HTTP/S

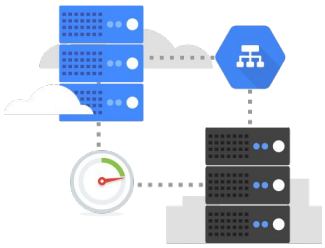
GKE supports any application runtime that you can package as a Docker image. GKE is ideally suited for containerized applications, including third-party containerized software.

You can run your container image on Kubernetes in a hybrid-cloud or multi-cloud environment. This is especially helpful when you have some parts of your application running on-premises and other parts 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 also automatically provisions Google Cloud network load balancers when you deploy Kubernetes Network Load Balancer Services, and Google Cloud HTTP and HTTPS Load Balancers when you configure Kubernetes Ingress resources. This eliminates the need to configure and manage these resources manually.

GKE has native support for Google Cloud's operations suite, providing seamless integration with tools for application and service monitoring, and for troubleshooting.

Use GKE for greater control over how Google Cloud 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 makes it easy to deploy and scale clusters. You can describe the compute, memory, network, and storage resources that you want to make available across all the containers required by your applications, and GKE will provision and manage the underlying Google Cloud resources automatically. You can either deploy fixed sized clusters or you can configure your clusters to automatically scale, adding or removing compute instances in response to the resources required by the containers running inside the cluster.

Deploy applications using standard Kubernetes tools

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: quiz-frontend
spec:
  replicas: 3
  template:
    spec:
      containers:
      - name: quiz-frontend
        image: gcr.io/.../quizapp
        ports:
        - name: http-server
          containerPort: 8080
```

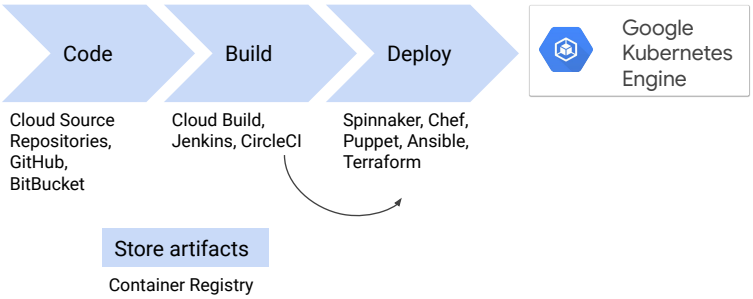
```
$ kubectl create -f
./frontend-deployment.yaml
```

Inside your GKE clusters you deploy and manage your containerized application in exactly the same way you would with any other Kubernetes environment. Most operational tasks are carried out using the “kubectl” command.

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

Applications can be deployed using Deployments, where Kubernetes will ensure that a specified number of replicas for a pod, or set of pods, is running at all times. The Deployment shown here is for stateless components. You can also use StatefulSets for applications where you need persistent storage for your application. There are a range of other resource types that you can define using YAML manifests and deploy and manage using the kubectl command.

Use GKE as a part of your CI/CD pipeline

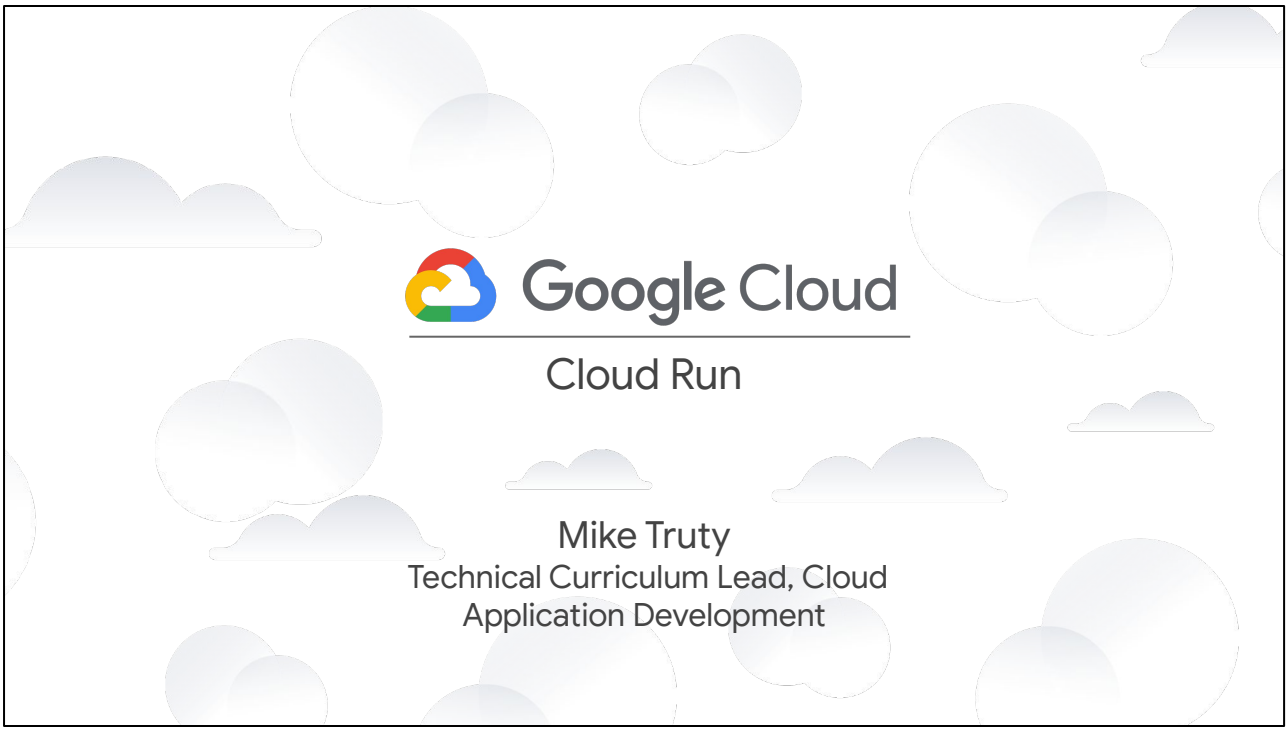


As a part of a continuous integration and delivery (CI/CD) pipeline, you can generate a new Docker image for each code commit and automatically deploy the image to development, test, and production environments. Cloud Build, Container Registry, and GKE can be used to create a strong CI/CD system.

Consider other compute environments

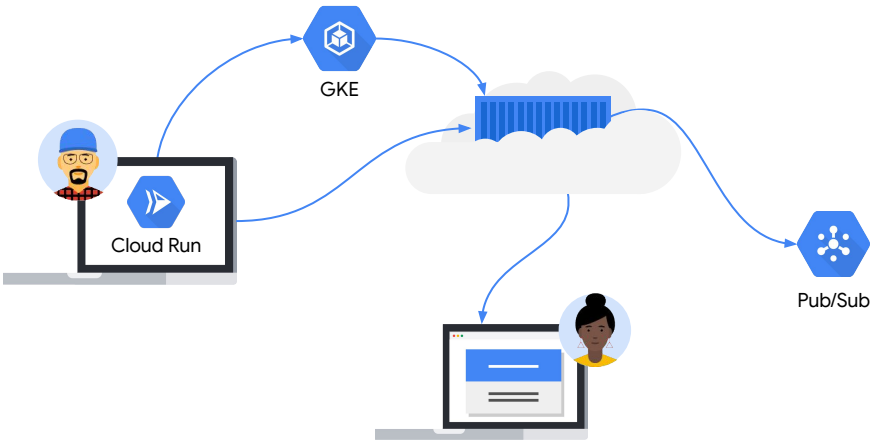
If your application cannot be packaged as a container.

GKE requires that your application be packaged and deployed as one or more container images. If your application cannot be containerized, you should consider other compute environments.

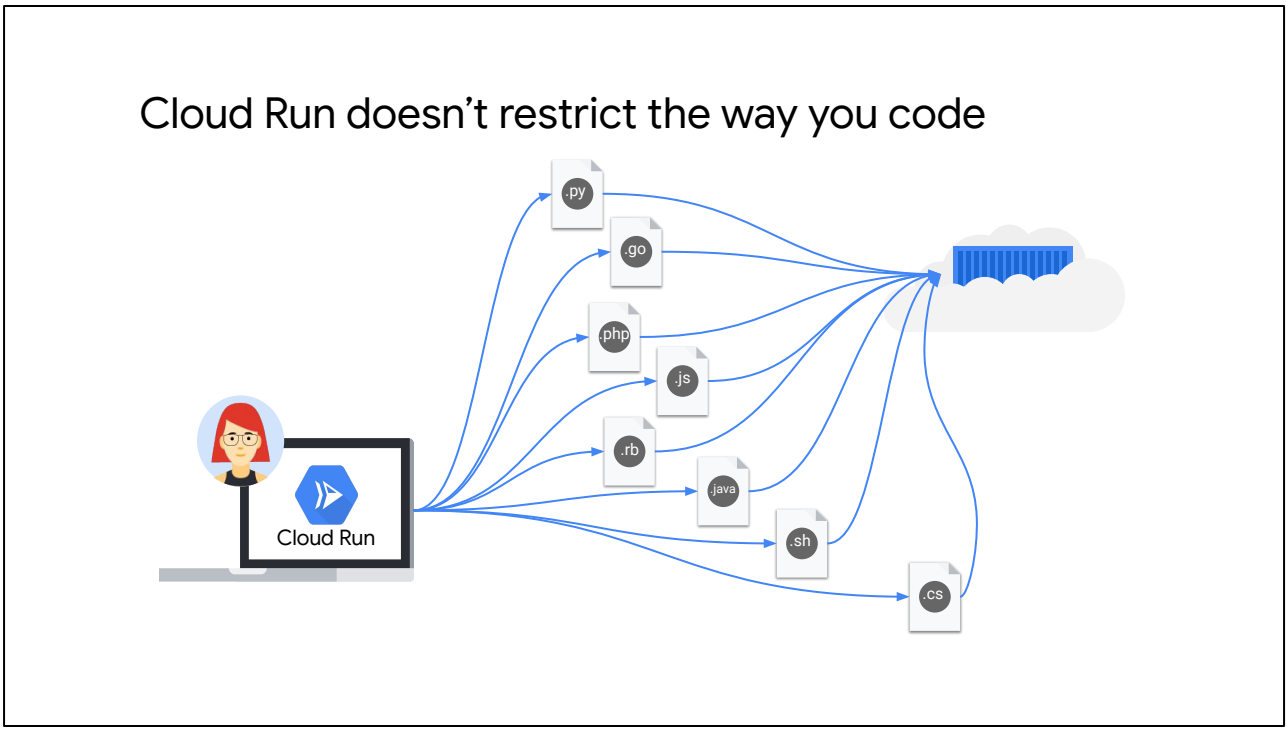


Let's dive into Cloud Run.

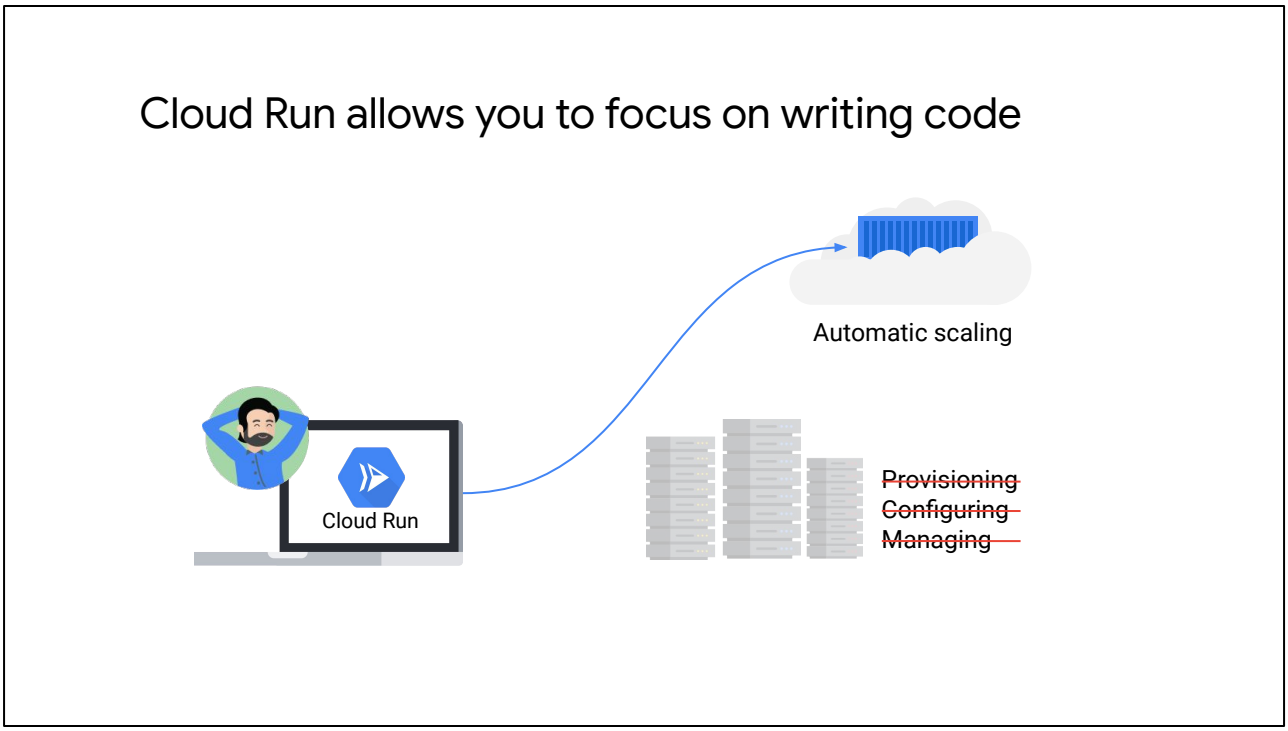
Cloud Run lets you focus on development



Cloud Run is a managed compute platform that enables you to run stateless containers that are invocable via web requests or Pub/Sub events. Cloud Run is serverless: it abstracts away all infrastructure management, so you can focus on developing applications. It is built on Knative, and gives you the choice of running your containers either fully managed with Cloud Run, or in your own GKE cluster with Cloud Run on GKE.



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 allowing you to easily deploy any stateless containers that listen for requests or events delivered via 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 wish, and deploy them in seconds.



Cloud Run enables you to run request or event-driven stateless workloads without having to worry about servers. 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 (calculated down to the nearest 100 milliseconds), so you will never have to pay for your over-provisioned resources.

With Cloud Run you can choose to deploy your stateless containers with a consistent developer experience to a fully managed environment or to your own GKE cluster. This common experience is enabled by Knative, an open API and runtime environment built on Kubernetes that gives you freedom to move your workloads across different environments and platforms. It is fully managed on Google Cloud, on GKE, or anywhere Knative runs.

Deploy automatically scaled containerized applications with a single command

```
gcloud beta run deploy
--image gcr.io/PROJECT-ID/helloworld
--platform managed
...
```

Once you have deployed your application, Cloud Run horizontally scales your container image automatically in order to handle received requests, then scales down when demand decreases. You only pay for the CPU, memory, and networking consumed during request handling.

Consider other compute environments

If your application cannot be packaged as a container.

If your application is not stateless, or must respond to requests or events delivered using protocols other than HTTP.

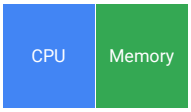
Cloud Run requires that your application be packaged and deployed as a container image. If your application is not stateless, or must respond to requests or events delivered using protocols other than HTTP, then Cloud Run will not be suitable.

If you need access to VPC resources, or need to deploy your containers to custom machine types, consider Cloud Run on GKE.

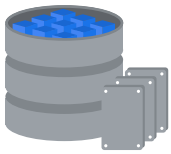
And now, on to Compute Engine.

Compute Engine gives you the greatest amount of control over your infrastructure compared to the other execution environments we've looked at so far.

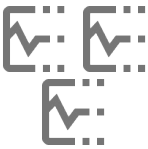
Run your application on high-performance, scalable VMs with Compute Engine



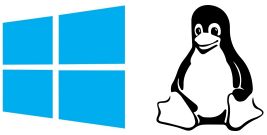
Predefined and custom machine types



Persistent Disks and Local SSDs



Pre-emptible VMs



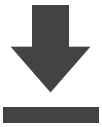
Windows, Linux OS, or Bring Your Own

Compute Engine supports predefined machine types. You can also create custom machine types to create VMs with the optimal amount of CPU and memory for your workloads. Compute Engine supports persistent disks and local SSDs. You can also launch pre-emptible VMs for large compute and batch jobs.

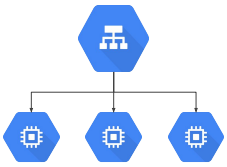
You can run your choice of operating system including Debian, CentOS, and various other flavors of Linux and Windows. You can also use a shared image from the Google Cloud community or bring your own operating system.

Use Compute Engine for full control of infrastructure

Machine type and OS



Software



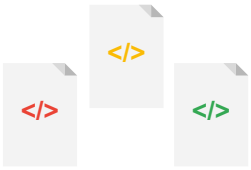
Instance groups with global load balancing

Use Compute Engine for 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.

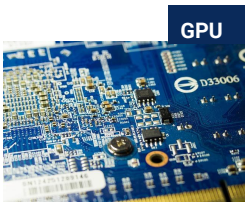
You can install and patch software running on the VM.

You can create managed instance groups based on an instance template. You can configure global load balancing and auto scaling of the managed instance groups. Compute Engine can perform health checks and replace unhealthy instances in an instance group. It auto-scales instances based on the traffic volume in specific regions.

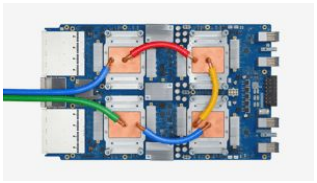
Use Compute Engine for maximum flexibility



Third-party software



Graphics Processing Unit



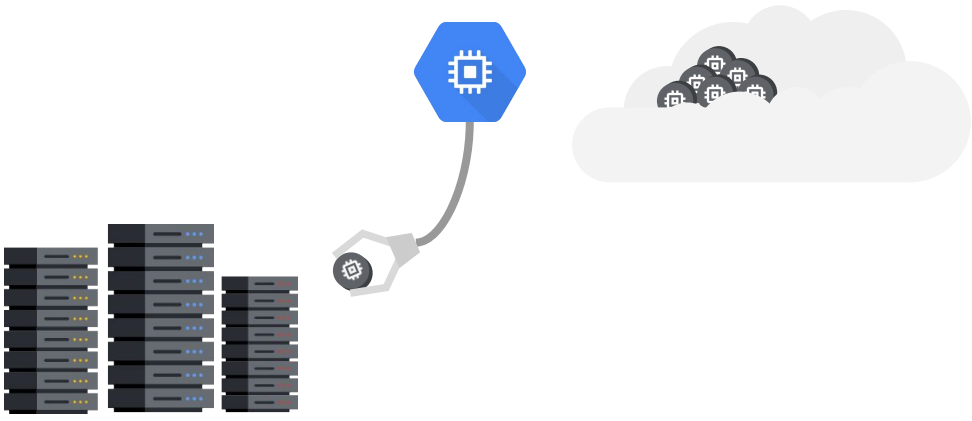
TensorFlow Processing Unit (TPU)

Compute engine offers you the most flexibility to configure your resources for the specific type of application that you need to run. You can run any third party license software on Compute Engine.

You can attach GPUs to Compute Engine VMs to speed up machine learning and data processing workloads.

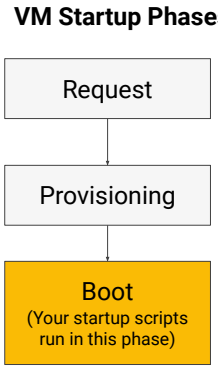
You can use Compute Engine for applications that require network protocols other than HTTP or HTTPS.

Use Compute Engine for lift-and-shift migration



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

Consider startup time

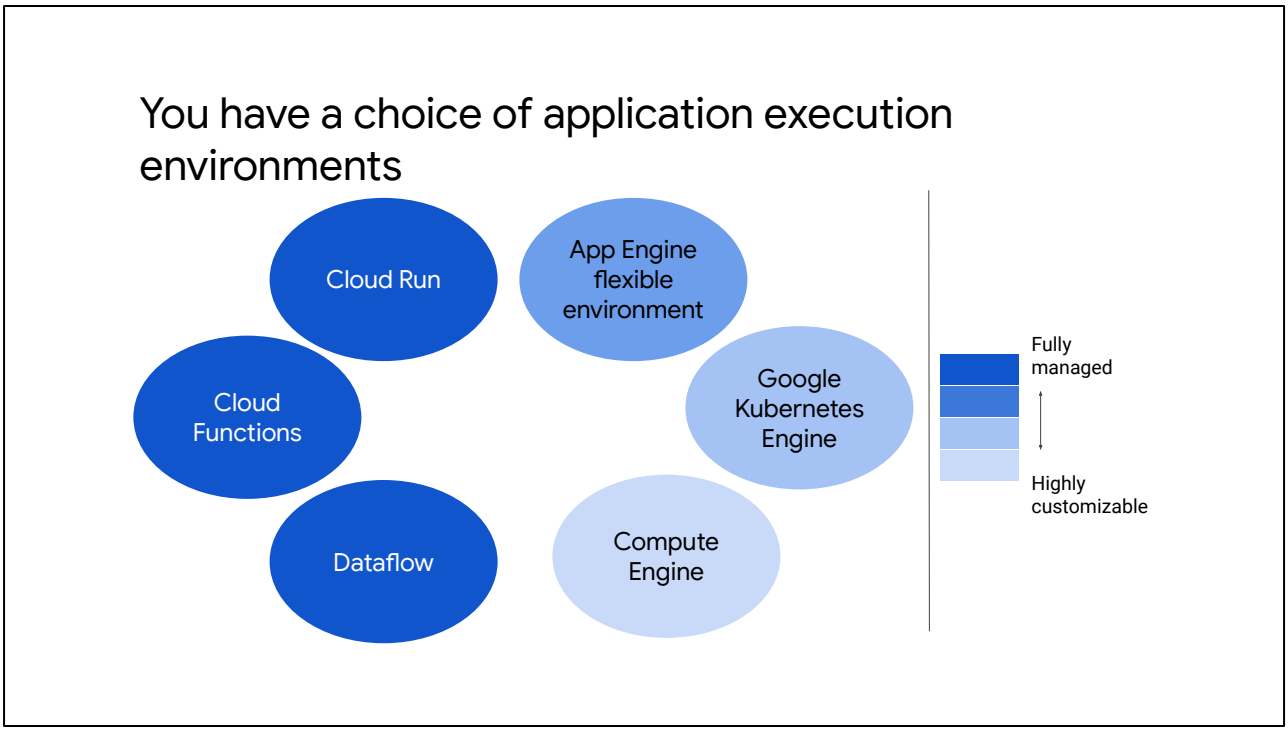


- Profile startup scripts
- Consider custom image
- Set appropriate target usage levels in autoscaling policy

Consider startup time when you use Compute Engine instances. A virtual machine can take about 60 seconds to spin up and become available. You can, however, launch hundreds of VMs within a few minutes.

To ensure that VM instances launch quickly, profile your startup scripts to identify and correct steps that take a long time to complete. If you're downloading and installing a lot of software, consider creating a custom image with all the software pre-installed. Plan ahead for bursts of traffic by setting appropriate target usage levels in your auto-scaling policy.

For more information, see https://cloudplatform.googleblog.com/2017/07/three-steps-to-Compute-Engine-startup-time-bliss-Google-Cloud-Performance-Atlas.html.



To summarize, Google Cloud provides a range of options to run your application code. We looked at Dataflow, Cloud Functions, and Cloud Run, which are fully managed. We also looked at App Engine, GKE, and Compute Engine which offer steadily increasing abilities to customize your execution environment. Fully managed compute environments require minimal set up and operations. On the other hand, highly customizable environments require greater operational and management effort to keep the application running optimally.

Now that you know the ideal use cases for each execution environment, you can choose the compute environment that matches the needs of your application and your team's ability to perform ongoing operations related tasks. For more information about execution environments, see the downloads in resources pane.

For more information about choosing a compute option, see:

- Deciding between Compute Engine, Container Engine, App Engine and more (Google Cloud Next '17): https://youtu.be/g0dN8Hkh5H8
- Choosing the right compute option in Google Cloud: a decision tree: https://cloudplatform.googleblog.com/2017/07/choosing-the-right-compute-option-in-GCP-a-decision-tree.html
- Choosing a Computing Option: https://cloud.google.com/docs/choosing-a-compute-option

Deploying the Application into App Engine Flexible Environment

Duration: 60 minutes

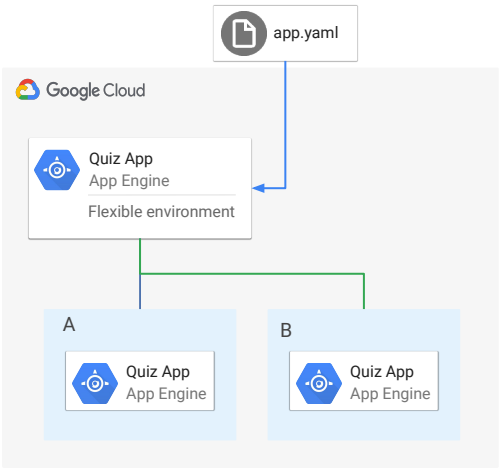
In this lab, you will deploy an application into App Engine flexible environment.

Lab objectives

Create an app.yaml file to describe the App Engine flexible environment requirements for an application.

Deploy the Quiz application into App Engine flexible environment.

Employ versions and traffic splitting to perform A/B testing of an application feature.



In the lab, you will create an app.yaml file to describe the App Engine flexible environment requirements for the Quiz application.

You'll deploy the Quiz application into App Engine flexible environment.

You'll then use versions and traffic splitting to perform A/B testing of an application feature.

Summary

Mike Truty
Technical Curriculum Lead, Cloud Application Development

Google Cloud offers a range of execution environments depending on the needs of your application and the level of operational control you desire. Cloud Client Libraries are the recommended way to programmatically interact with Google Cloud services. With this approach, you are not boxed into one execution environment.

For example, let's say you have an application that uses HTTP-based protocols and you initially deploy it to App Engine flexible environment because of App Engine's no-ops benefits. If your application evolves in the future and needs to use non-HTTP based protocols, you can deploy the application to Container Engine without reworking the code.

You are not boxed in!