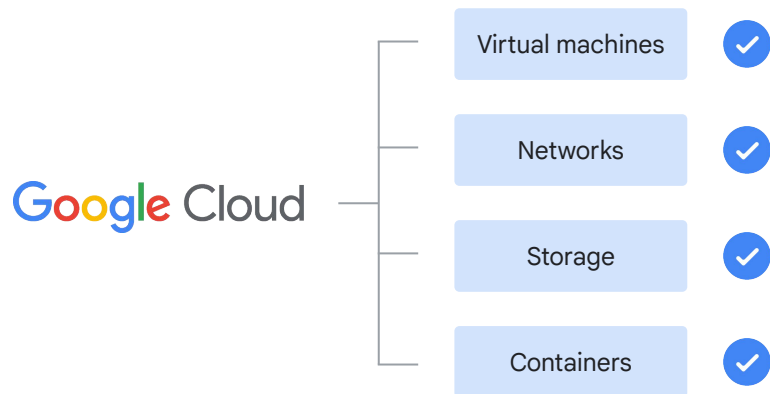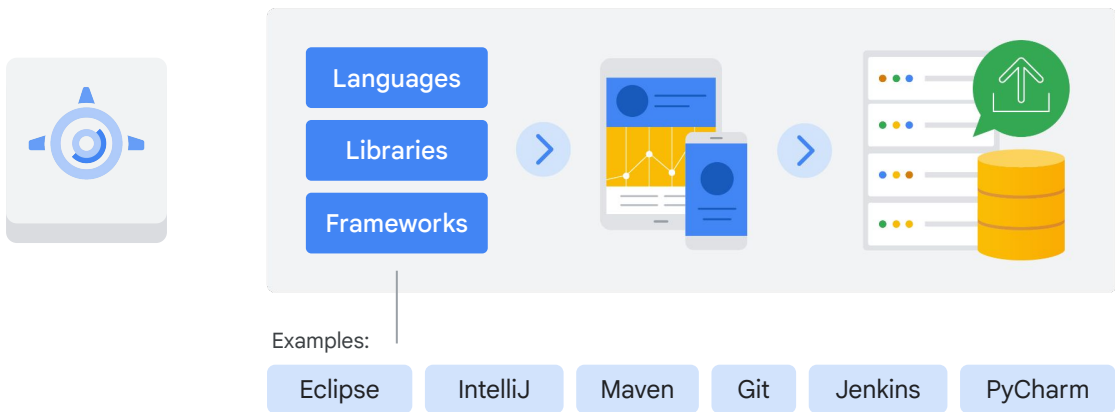Google Cloud

Google Cloud
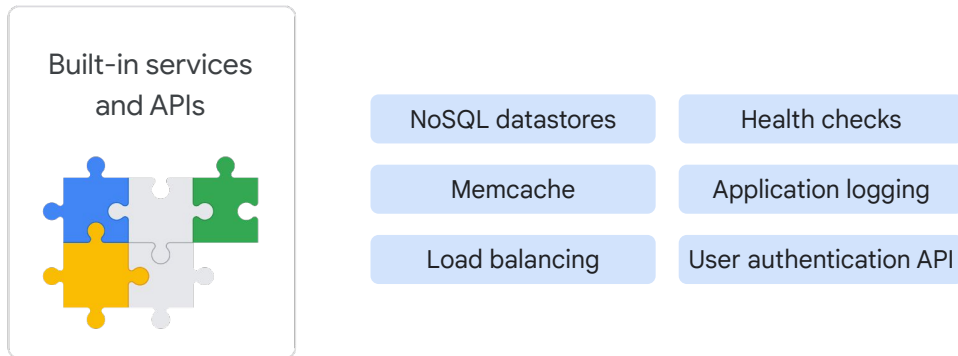# Core Infrastructure
Module 6

On-demand course
March 2022

So far in this course, we've provided an introduction to Google Cloud and explored the options and benefits related to using virtual machines, networks, storage, and containers in the Cloud.

Coding options include Eclipse, IntelliJ, Maven, Git, Jenkins, and PyCharm.

Built-in services and APIs

NoSQL datastores	Health checks
Memcache	Application logging
Load balancing	User authentication API

App Engine provides built-in services and APIs, like NoSQL datastores, Memcache, load balancing, health checks, application logging, and a user authentication API that's common to most applications.
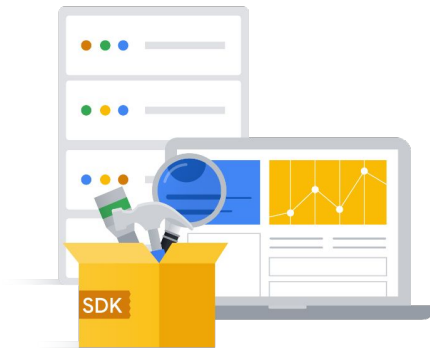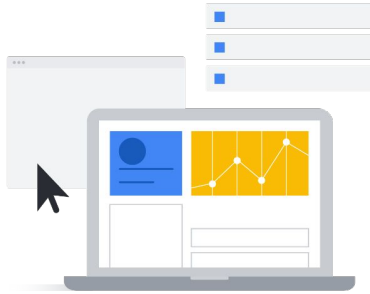
Each SDK includes:

APIs and libraries

Sandbox environment

Deployment tools

Each SDK includes:

- All of the APIs and libraries available to App Engine
- A simulated, secure sandbox environment that emulates all of the App Engine services on your local computer
- And deployment tools to upload your application to the cloud and manage different versions

Google Cloud console's web-based interface

- ✓ Create new applications
- ✓ Configure domain names
- ✓ Change which version is live
- ✓ Examine access and error logs

You can use the Google Cloud console's web-based interface to:

- create new applications,
- configure domain names,
- change which version of your application is live,
- examine access and error logs,
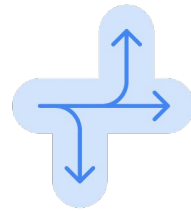
and much more.

Keeps web applications safe
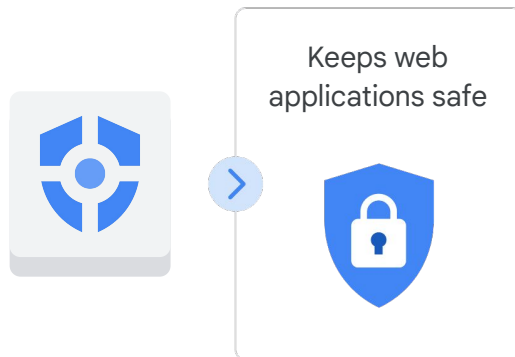
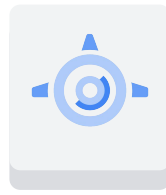From a security perspective, the **Security Command Center**–Google Cloud's security and risk management platform–keeps web applications safe.

Through the Google Cloud console, you can use the Security Command Center to automatically scan and detect common web application vulnerabilities.
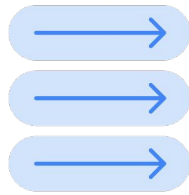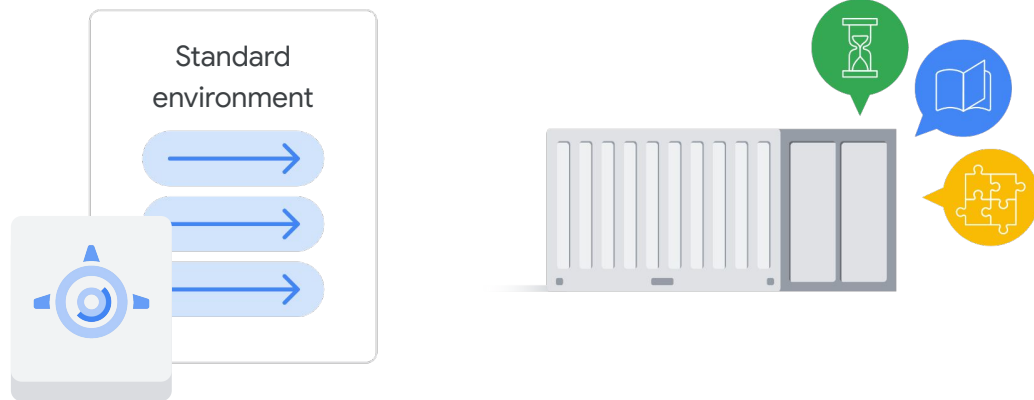
Standard   Flexible

There are two types of App Engine environments: **standard** and **flexible**.

The App Engine **standard environment** is based on container instances running on Google's infrastructure.

Containers are preconfigured with a runtime from a standardized list of supported languages and versions, which includes libraries that support App Engine standard APIs.
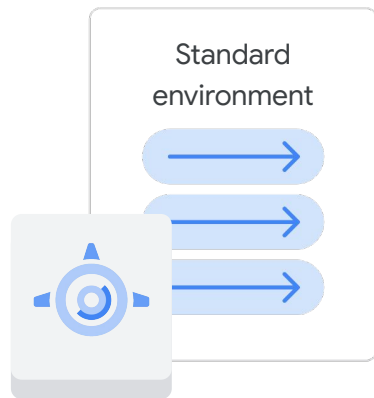
Standard environment	Persistent storage with queries, sorting, and transactions
	Automatic scaling and load balancing
	Asynchronous task queues for performing work outside the scope of a request
	Scheduled tasks for triggering events at specified times or regular intervals
	Integration with other Google Cloud services and APIs

Standard environment features include:

- Persistent storage with queries, sorting, and transactions

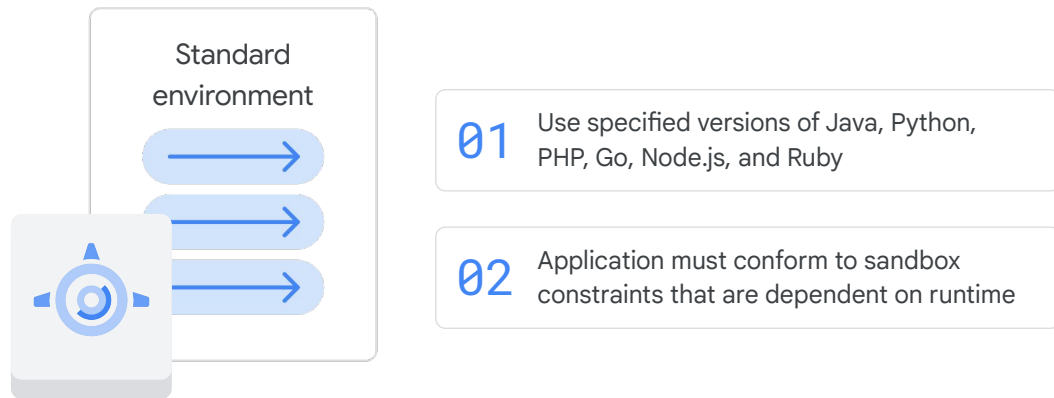- Automatic scaling and load balancing

- Asynchronous task queues for performing work outside the scope of a request

- Scheduled tasks for triggering events at specified times or regular intervals

- And integration with other Google Cloud services and APIs

Standard environment

01   Use specified versions of Java, Python, PHP, Go, Node.js, and Ruby

02   Application must conform to sandbox constraints that are dependent on runtime

There are a couple of requirements for using the standard environment:
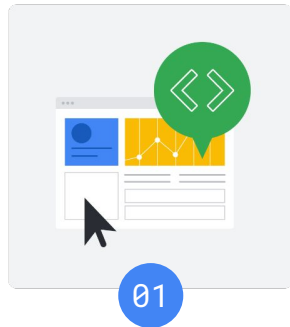
1. You must use specified versions of Java, Python, PHP, Go, Node.js, and Ruby, and

2. Your application must conform to sandbox constraints that are dependent on runtime.
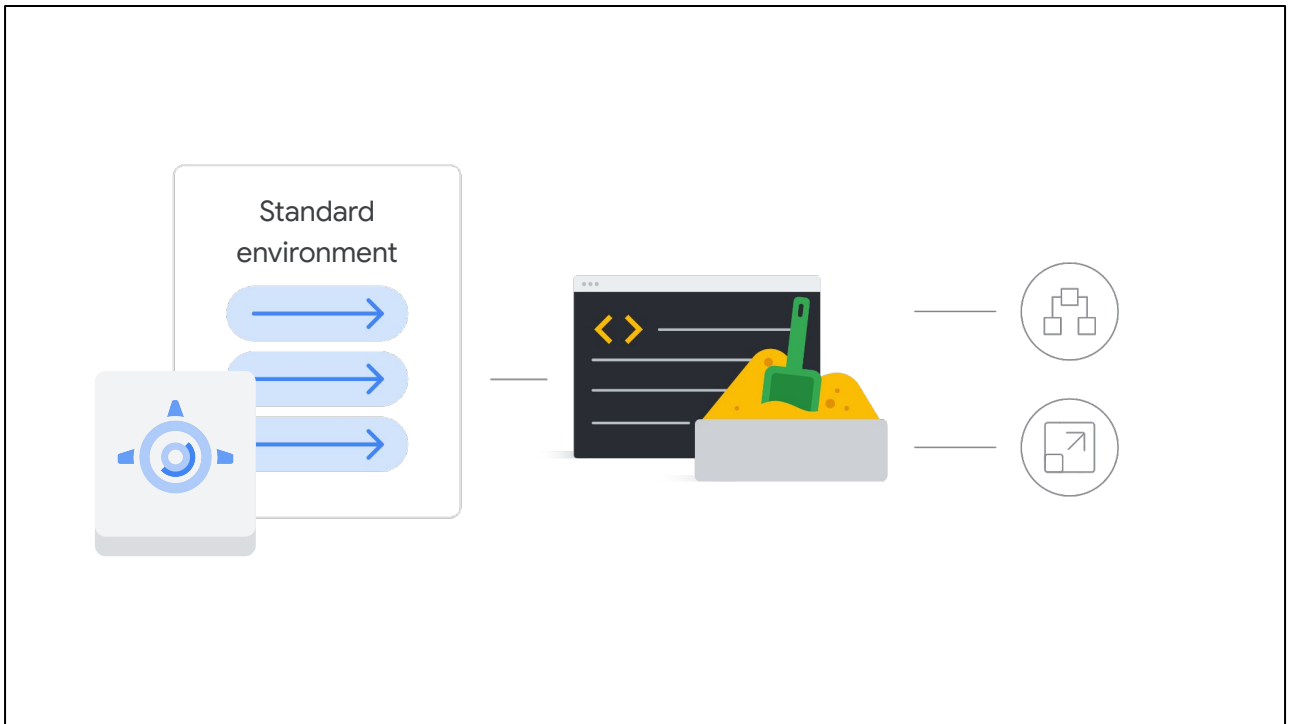
Applications run in a secure, sandboxed environment. This allows the App Engine standard environment to distribute requests across multiple servers and scale servers to meet traffic demands.

01 — **Develop** web app and test locally
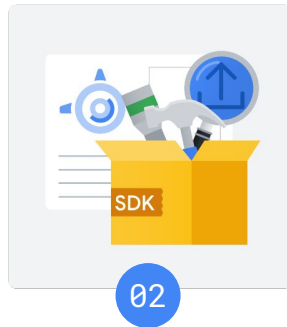
02 — **Deploy** to App Engine with SDK
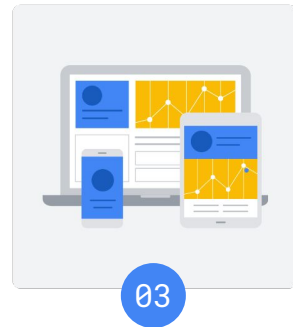
03 — App Engine **scales** and services the app

A standard environment workflow typically follows these three steps:

- **First**, a web application is developed and tested locally.

- **Second**, the SDK is used to deploy the application to App Engine.

- And **third**, App Engine scales and services the application.
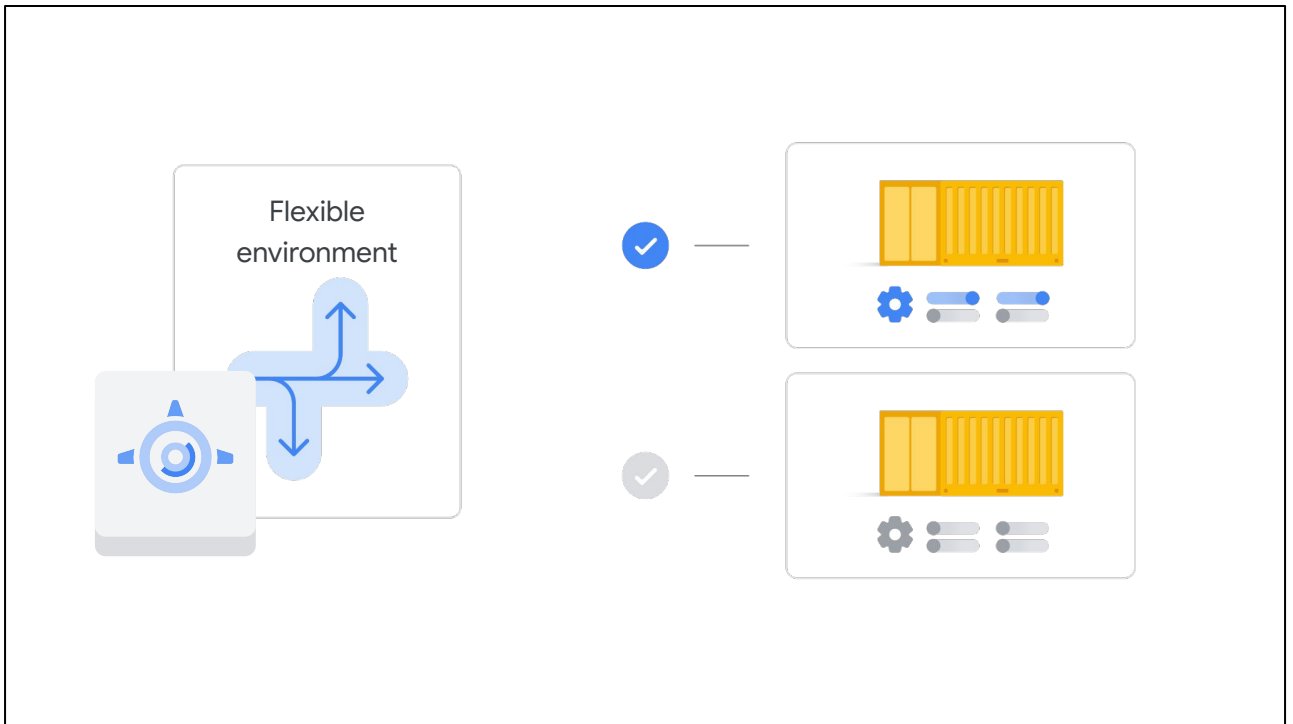
App Engine also offers a **flexible environment**.

If the standard environment's sandbox model is too restrictive for you, the flexible environment can let you specify the type of container your web application will run in.

Docker containers   Compute Engine VM

This option lets an application run inside Docker containers on Google Cloud's Compute Engine virtual machines.

Flexible environment

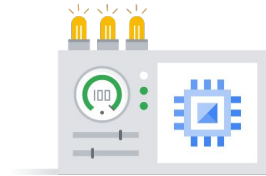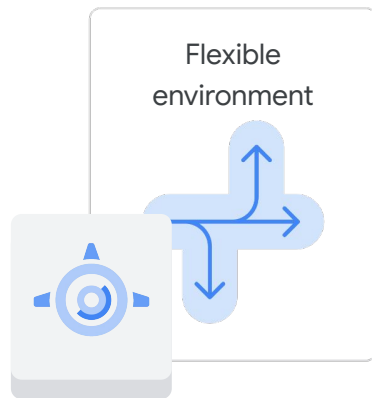Instances are health-checked, healed, and co-located

Critical, backward-compatible updates are automatically applied to the underlying operating system

VM instances are automatically located by geographical region according to the settings in your project

VM instances are restarted on a weekly basis

This means that:

- Instances are health-checked, healed as necessary, and co-located with other module instances within the project.

- Critical, backward-compatible updates are automatically applied to the underlying operating system.

- VM instances are automatically located by geographical region according to the settings in your project. Google's management services ensure that all of a project's VM instances are co-located for optimal performance.
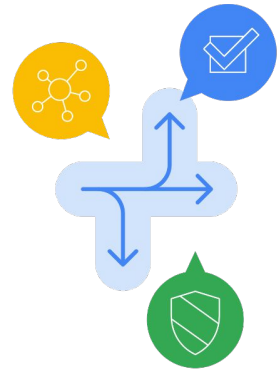
- And VM instances are restarted on a weekly basis. During restarts, Google's management services will apply any necessary operating system and security updates.

- ✓ Microservices
- ✓ Authorization
- ✓ SQL & NoSQL databases
- ✓ Traffic splitting
- ✓ Logging
- ✓ Search
- ✓ Versioning
- ✓ Security scanning
- ✓ Memcache
- ✓ Content delivery networks

The flexible environment supports microservices, authorization, SQL and NoSQL databases, traffic splitting, logging, search, versioning, security scanning, Memcache, and content delivery networks.
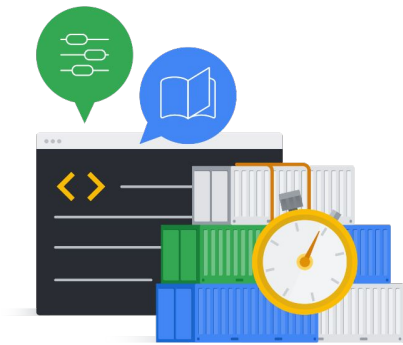
01 Benefit from custom configurations and libraries, while focusing on writing code

02 Customize the runtime and the operating system of your virtual machine. Standard runtimes include Python, Java, Go, Node.js, PHP, .NET, and Ruby

03 Customize or provide runtimes by supplying a custom Docker image or Dockerfile

App Engine Flexible allows users to also benefit from custom configurations and libraries while still keeping their main focus on what they do best – writing code.

In addition, the App Engine flexible environment allows you to customize the runtime and the operating system of your virtual machine by using Dockerfiles. As in App Engine Standard, supported runtimes include Python, Java, Go, Node.js, PHP, and Ruby.

However, in App Engine Flexible, developers can also use different versions of these runtimes or provide their own custom runtime by supplying a custom Docker image or using a Dockerfile from the open source community.

	Standard environment	Flexible environment
Instance startup	Seconds	Minutes
SSH access	No	Yes (although not by default)
Write to local disk	No (some runtimes have read and write access to the /tmp directory)	Yes, ephemeral (disk initialized on each VM startup)
Support for 3rd-party binaries	For certain languages	Yes
Network access	Via App Engine services	Yes
Pricing model	After free tier usage, pay per instance class, with automatic shutdown	Pay for resource allocation per hour; no automatic shutdown

So, how do these two environments compare to each other?

Let's start with the **standard environment**, which is fast. It starts up instances of your application in seconds, but you have less access to the infrastructure in which your application runs.

With the standard environment, you can't use SSH to connect to the virtual machines on which your application runs, and you can't write to a local disk.

The standard environment does support third-party binaries for certain languages, and you can use App Engine to make calls to the network.

Finally, in terms of pricing, after a free tier usage, you pay per instance class with automatic shutdown.

The **flexible environment** takes minutes to start up, instead of seconds.

But it lets you use SSH to connect to the virtual machines on which your application runs, it lets you use local disk for scratch space, it lets you install third-party software, and it lets your application make calls to the network without going through App Engine.

In terms of pricing, with the flexible environment, you pay for resource allocation per hour with no automatic shutdown.
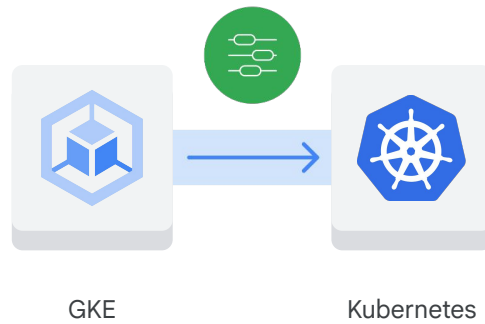
GKE   Kubernetes

Google Kubernetes Engine, however, gives the application *owner* the full flexibility of Kubernetes.

Cloud Endpoints — API — API Management
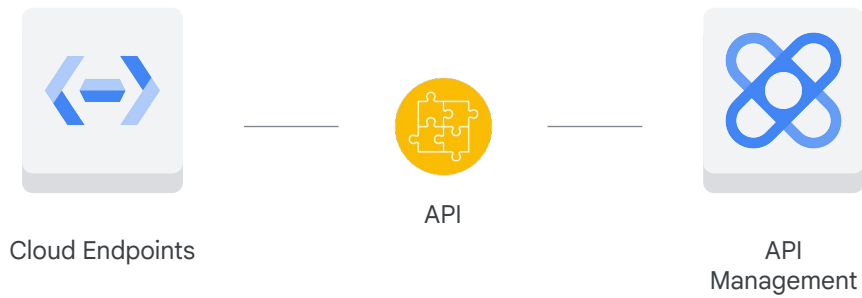
Now that you've had a thorough overview of App Engine, let's transition to Cloud Endpoints and API Management. These are management tools for Google Cloud's application programming interface, or API.

So what exactly is an API?
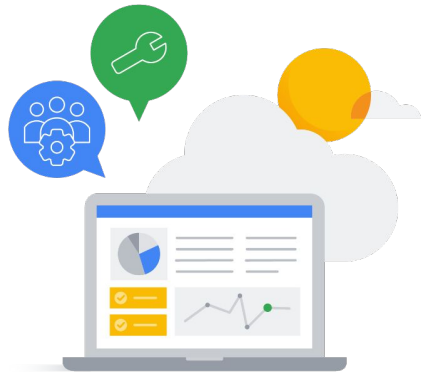
Software service

Implementation
- ✓ Complex
- ✓ Changeable

Manually coded
- ✓ Brittle
- ✓ Error-prone

A software service's implementation can be complex and changeable.

If other software services had to be explicitly coded in detail in order to use that service, the result would be brittle and error-prone.

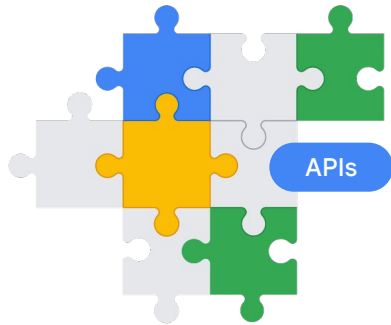- A clean, well-defined interface
- Underlying implementation can change
- Changes to the API are made with versions

So instead, application developers structure the software they write so that it presents a clean, well-defined interface that hides unnecessary detail, and then they document that interface. That's an application programming interface.

The underlying implementation can change, as long as the interface doesn't, and other pieces of software that use the API don't have to know or care.

Sometimes you do have to change an API, perhaps to add or deprecate a feature. To cleanly make this kind of change to the API, developers create versions. For example, version 2 of an API might contain calls that version 1 does not. This means that programs that consume the API can specify the API version they want to use in their calls.
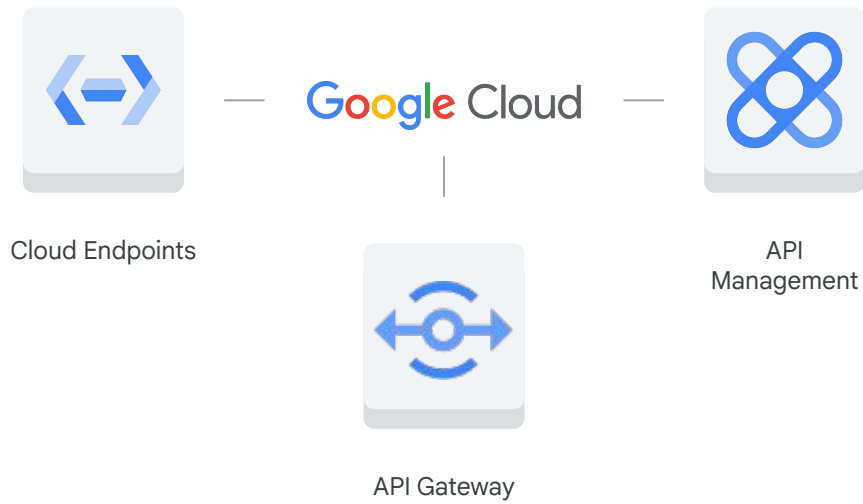
Supporting an API is a very important task, and Google Cloud provides **three** API management tools: **Cloud Endpoints, API Gateway,** and **Apigee API Management.**

Cloud Endpoints

Distributed API management system

Provides an API console, hosting, logging, monitoring, and other features

Use with any APIs that support the OpenAPI Specification

Supports applications running in App Engine, Google Kubernetes Engine, and Compute Engine

Clients include Android, iOS, and Javascript

- Cloud Endpoints is a distributed API management system that uses a distributed Extensible Service Proxy, which is a service proxy that runs in its own Docker container. The goal is to help you create and maintain even the most demanding APIs with low latency and high performance.

- Cloud Endpoints provides an API console, hosting, logging, monitoring, and other features to help you create, share, maintain, and secure your APIs.

- You can use Cloud Endpoints with any APIs that support the OpenAPI Specification.

- Cloud Endpoints supports applications running in App Engine, Google Kubernetes Engine, and Compute Engine.

- Clients include Android, iOS, and Javascript.

	Backend implementations can vary for a single service provider
API Gateway	Provide secure access to your backend services through a well-defined REST API
	Clients consume your REST APIS to implement standalone apps

API Gateway is another API management tool.

- Web-based services today provide a huge variety of functionality, meaning everything from map, weather, and image services, to games, auctions, and many other service types. Service providers have many options for how to implement, deploy, and manage their services. For example, one service might be developed in Java or .NET, while another uses Node.js.

  Backend implementations can also vary for a single service provider. A service provider might have legacy services implemented using one architecture, and new services implemented using a completely different architecture.

- API Gateway enables you to provide secure access to your backend services through a well-defined REST API that is consistent across all of your services, regardless of the service implementation.

- Clients consume your REST APIS to implement standalone apps for a mobile device or tablet, through apps running in a browser, or through any other type of app that can make a request to an HTTP endpoint.

Specific focus on business problems, like rate limiting, quotas, and analytics

Many Apigee API Management users provide a software service to other companies

Backend services for Apigee API Management don't need to be in Google Cloud

Apigee API Management

Another Google Cloud platform available for developing and managing API proxies is Apigee API Management.

- Unlike Cloud Endpoints, Apigee API Management has a specific focus on business problems, like rate limiting, quotas, and analytics.

- In fact, many Apigee API Management users provide a software service to *other* companies.

- Backend services for Apigee API Management don't have to be in Google Cloud, and as a result, engineers also often use it to take apart legacy applications. So, instead of replacing a large, important application in one move, they can use Apigee API Management to peel off its services individually instead. This allows them to stand up microservices to implement each in turn until the legacy application can finally be retired.

A managed compute platform that can run stateless containers

Serverless, removing the need for infrastructure management

Built on Knative, an open API and runtime environment built on Kubernetes

Can automatically scale up and down from zero almost instantaneously, charging only for the resources used
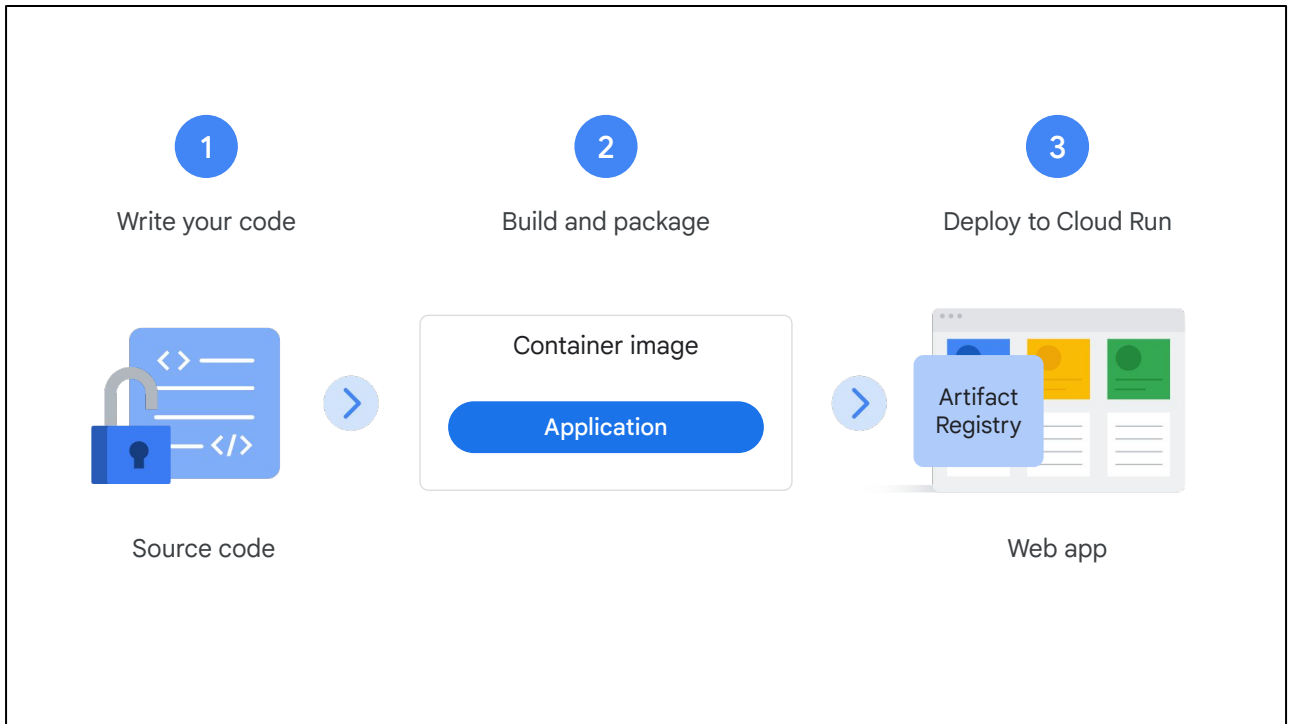
Cloud Run

The final application platform we'll explore in this section of the course is Cloud Run,

- A managed compute platform that lets you run stateless containers via web requests or Pub/Sub events.

- Cloud Run is serverless. That means it removes all infrastructure management tasks so you can focus on developing applications.

- It's built on Knative, an open API and runtime environment built on Kubernetes that gives you freedom to move your workloads across different environments and platforms. It can be fully managed on Google Cloud, on Google Kubernetes Engine, or anywhere Knative runs.

- Cloud Run is fast. It can automatically scale up and down from zero almost instantaneously, and it charges you only for the resources you use, calculated down to the nearest 100 milliseconds, so you'll never pay for your over-provisioned resources.
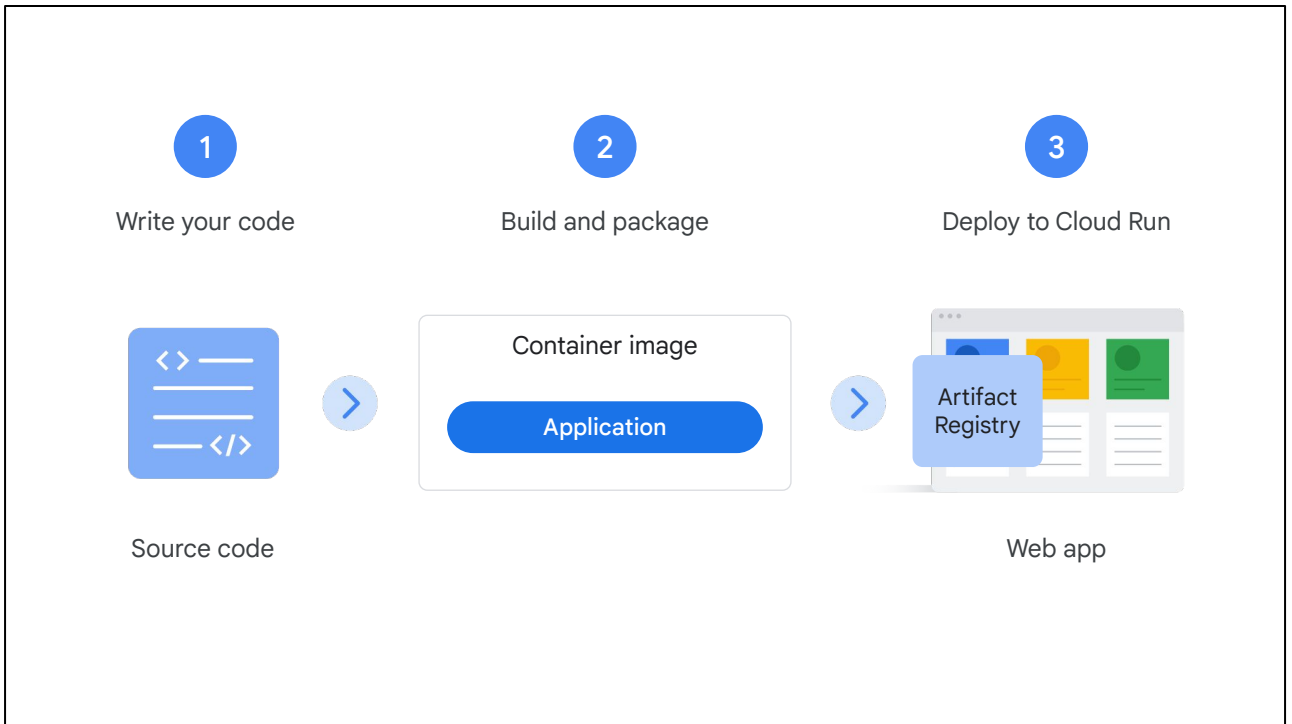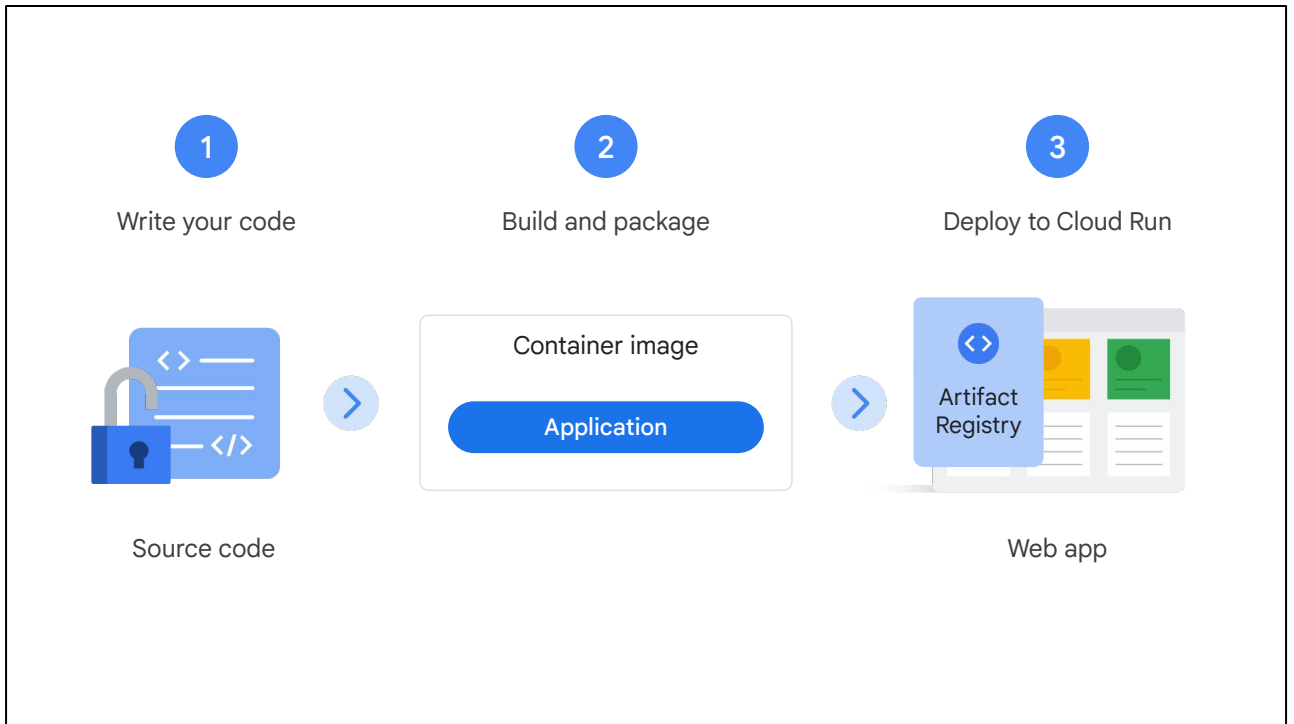
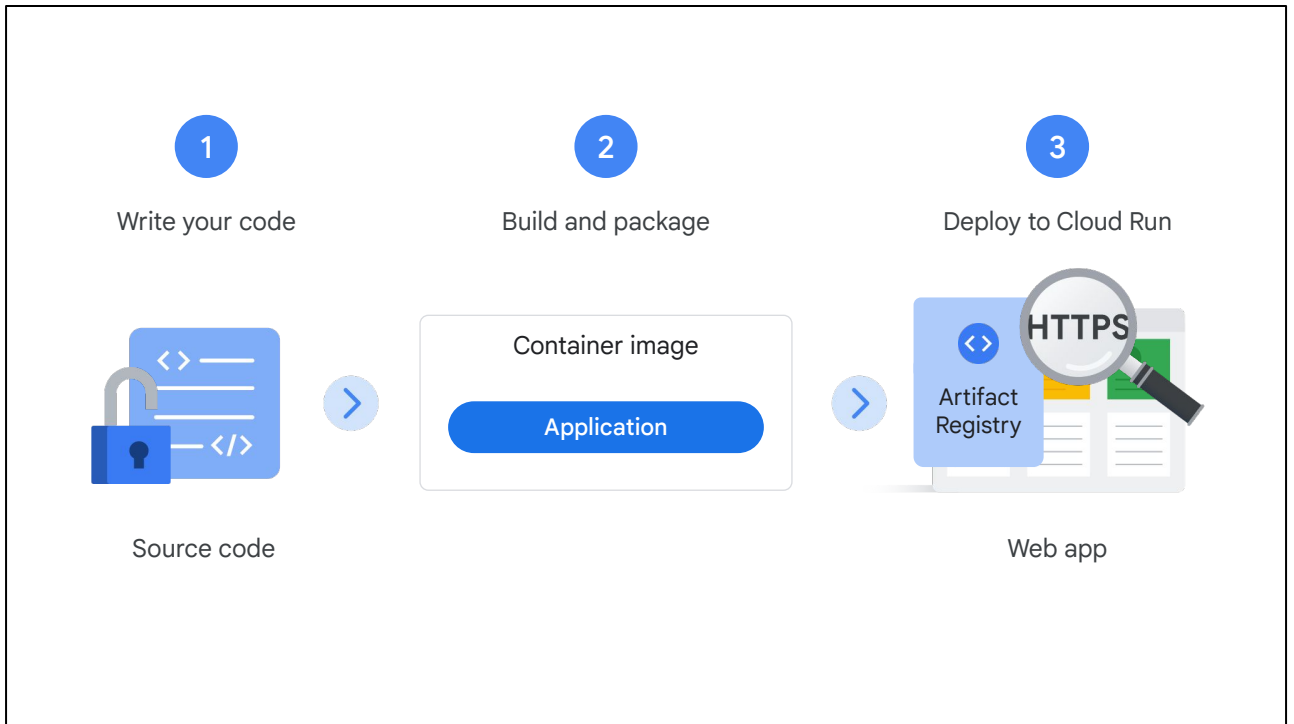The Cloud Run developer workflow is a straightforward three-step process:

- **First**, you write your application using your favorite programming language. This application should start a server that listens for web requests.

- **Second**, you build and package your application into a container image.

- **Third**, the container image is pushed to Artifact Registry, where Cloud Run will deploy it. Note that Cloud Run can only deploy images that are stored in Artifact Registry. You can build, push and deploy your own code from your local source

| 1 | 2 | 3 |
| Write your code | Build and package | Deploy to Cloud Run |

Source code

Container image

Application

Artifact Registry

Web app

if you have the required permissions. You can also deploy an image

| 1 | 2 | 3 |
| Write your code | Build and package | Deploy to Cloud Run |

Source code

Container image

Application

Artifact Registry

Web app

that already exists in Artifact Registry.

1 Write your code

2 Build and package

3 Deploy to Cloud Run

Source code

Container image

Application

Artifact Registry

HTTPS

Web app

Once you've deployed your container image, you'll get a unique HTTPS URL back.
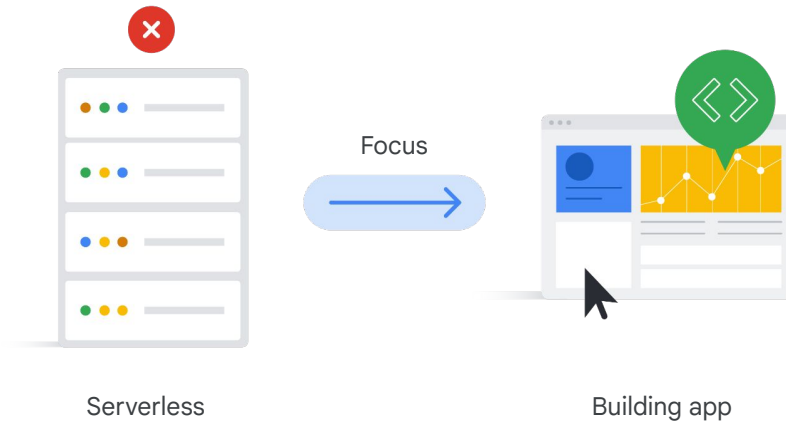
Because Cloud Run is serverless, it means that you, as a developer, can focus on building your application and not on building and maintaining the infrastructure that powers it.
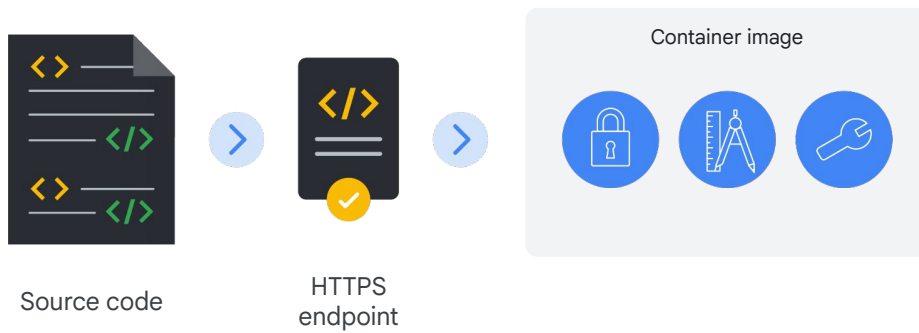
Container-based workflow

✔ Transparency

✔ Flexibility

For some use cases, a container-based workflow is great, because it gives you a great amount of transparency and flexibility.

If **you** build the container image **you** have the power to decide exactly what file ends up in your container image, and how it gets there.
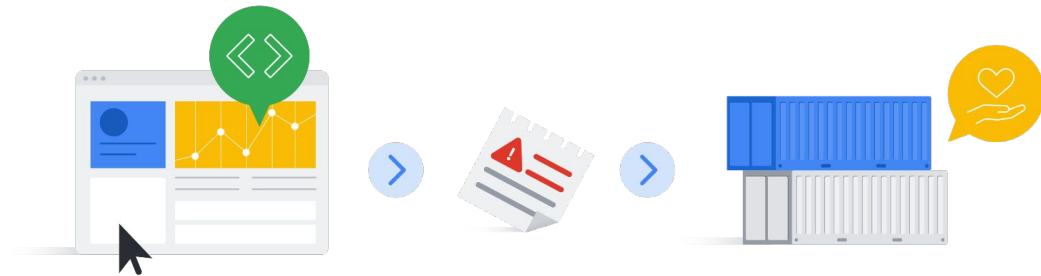
However, building an application is hard enough already, let alone having to think about **containerization** and the responsibilities that come with that.

Sometimes, you're just looking for a way to turn source code into an HTTPS endpoint, and you want your vendor to make sure your container image is secure, well-configured, and built in a consistent way.

With **Cloud Run** you can use a container-based workflow as well as a source-based workflow

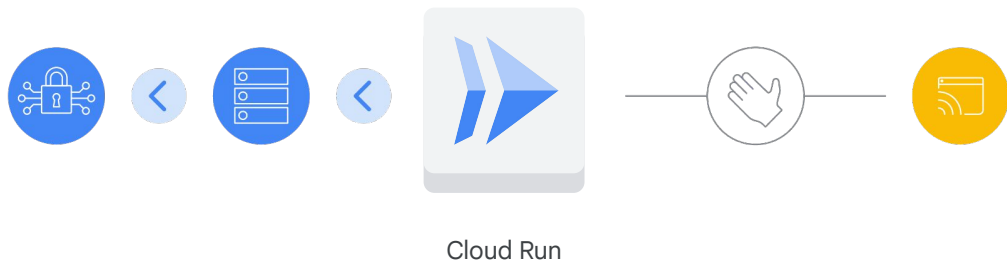With Cloud Run, you can do both. You can use a container-based workflow, as well as a source-based workflow.
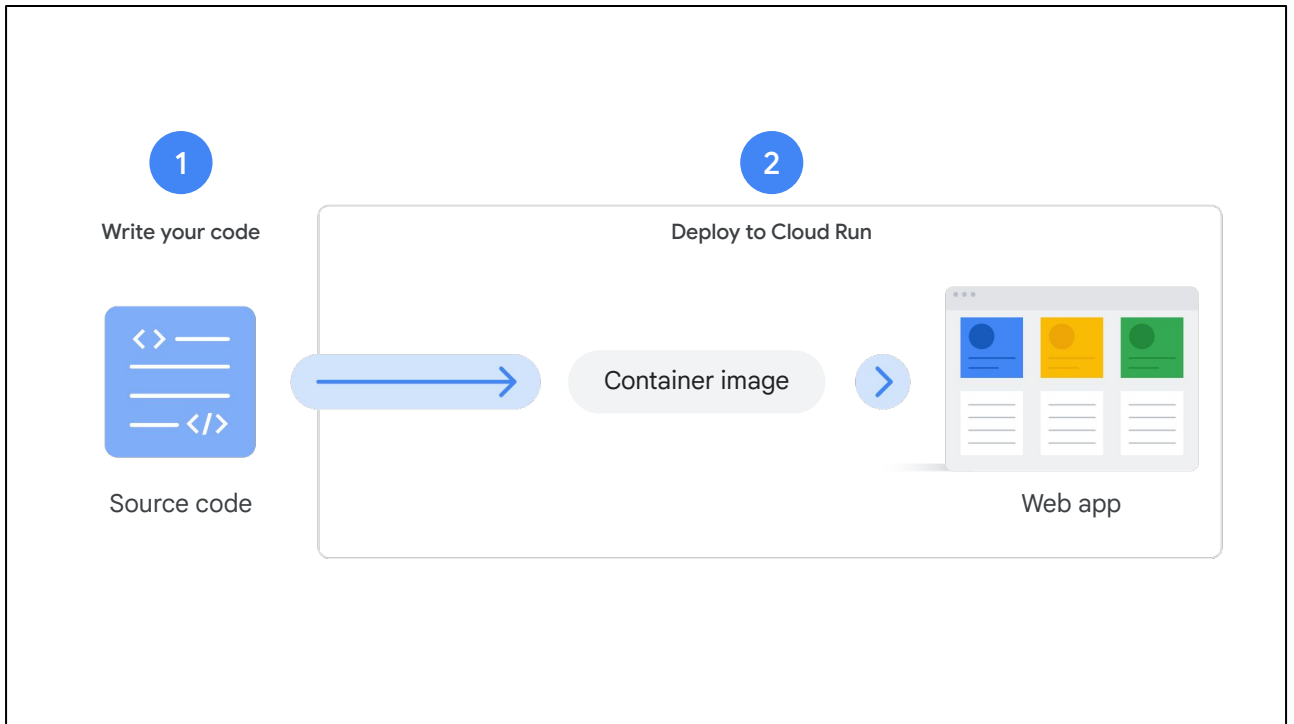
If you use the source-based approach, you'll deploy your source code, instead of a container image. Cloud Run then builds your source and packages the application into a container image for you.
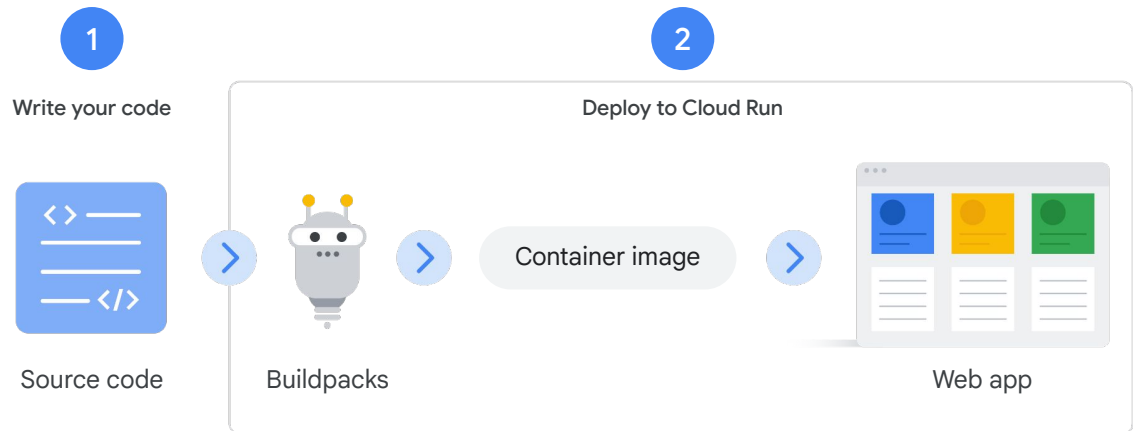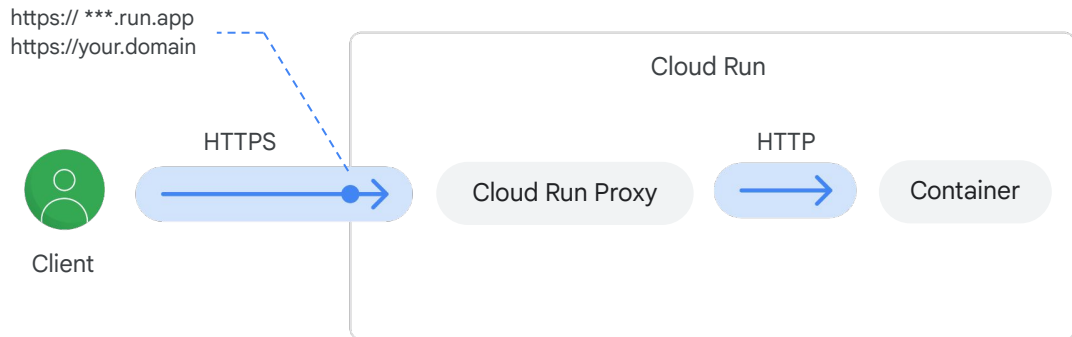
Cloud Run does this using Buildpacks - an open source project.

Cloud Run

Cloud Run handles HTTPS serving for you.

That means you only have to worry about handling web requests, and you can let Cloud Run take care of adding the encryption.
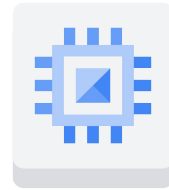
By default, your application is exposed on a unique subdomain of the global run.app domain. You can also use your own, custom domain.

Cloud Run manages everything else:
- Generating a valid SSL certificate
- Configuring SSL termination correctly with secure settings
- And handling incoming requests, decrypting them, and forwarding them to your application
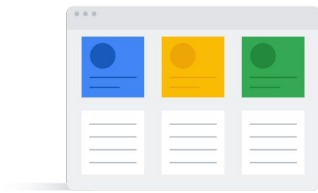
Most of the other compute products (such as Compute Engine), charge for servers as long as they're running, even if you're not using them.

That means you're often paying for idle server capacity.
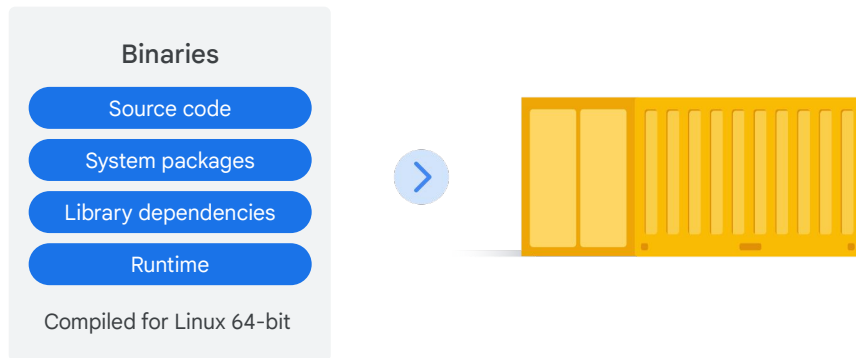
You can use Cloud Run to run any binary, as long as it's compiled for Linux sixty-four bit.

Acceptable programming languages:

✓ Java   ✓ PHP

✓ Python   ✓ Go

✓ Node.js   ✓ C++

Web app

Now, this means you can use Cloud Run to run web applications written using popular languages, such as:

- Java
- Python
- Node.js
- PHP
- Go
- C++