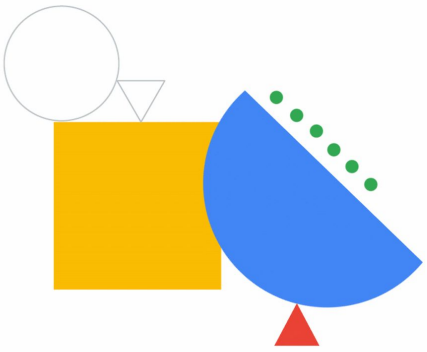
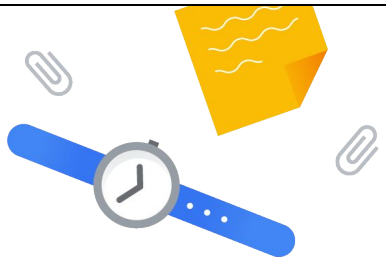
Google Cloud

Application Development, Testing, and Integration





01 Development and testing

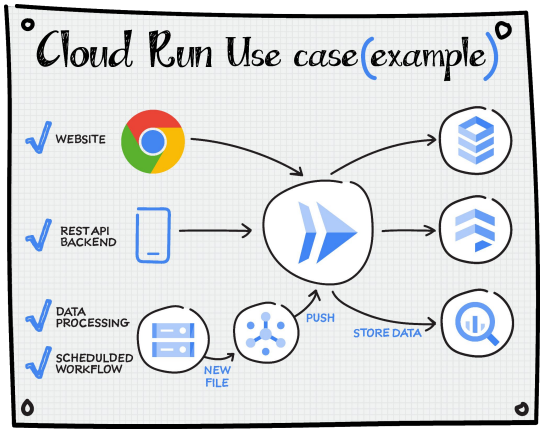
Agenda

In this section, we discuss how you can develop and test your applications locally before deploying them to Cloud Run.

Is your application a good fit for Cloud Run?

Your application is a good fit, if it:

- Serves requests, streams, or events that are delivered over HTTP, HTTP/2, Websockets, or gRPC.
- Does not require a local persistent file system, and works with a local ephemeral or network file system.
- Does not require more that 8 CPU or 32 GiB of memory per instance.
- Is containerized, or written in Go, Java, Node.js, Python, or .NET.



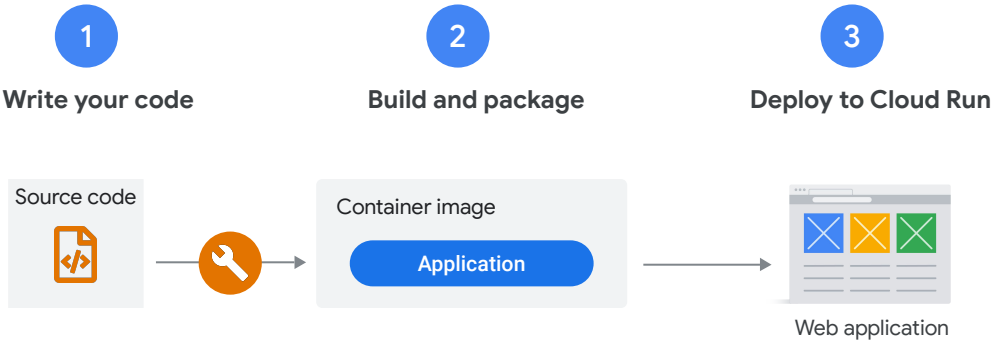
In order to be a good fit for Cloud Run, your application needs to meet *all* of the following criteria.

Your application:

- Serves requests, streams, or events delivered over HTTP, HTTP/2, WebSockets, or gRPC, or executes to completion.
- Does not require a *local persistent* file system, but either a local *ephemeral* file system or a *network* file system.
- Is built to handle multiple instances of the app running simultaneously.
- Does not require more than 8 CPU and 32 GiB of memory per instance.
- Meets one of the following criteria:
  - Is containerized.
  - Is written in Go, Java, Node.js, Python, or .NET.
  - You can otherwise containerize it.

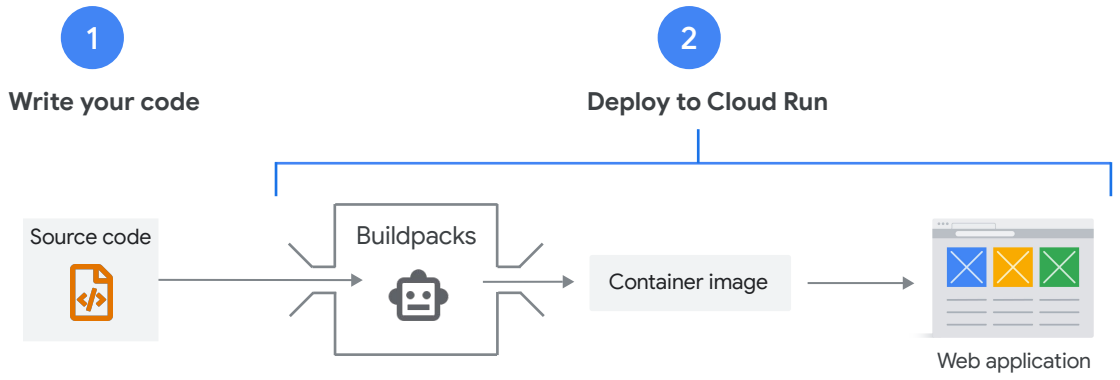
If your app meets those criteria, then it's a good fit for Cloud Run!

Cloud Run developer workflow



You can write code in any programming language and deploy it on Cloud Run if you can build a container image from it.

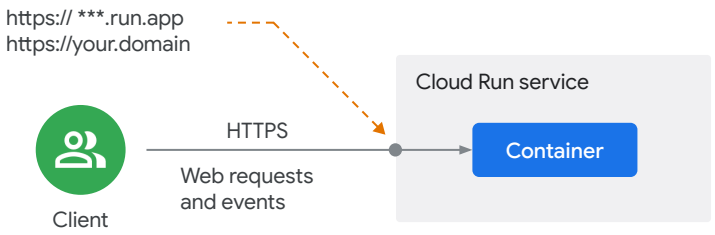
Cloud Run developer workflow



Also, building container images is optional. If you use Go, Node.js, Python, Java, .NET Core, or Ruby, you can use the source-based deployment option that builds the container for you and deploys it on Cloud Run.

If you use the source-based approach, you deploy your source code, instead of a container image. Using Buildpacks, Cloud Run then builds your source, and packages the application along with its dependencies into a container image for you. Buildpacks is discussed in the course on Developing Containerized Applications on Google Cloud.

Application on Cloud Run must handle web requests



Cloud Run supports secure HTTPS requests to your application. On Cloud Run, your application can either run continuously as a service or as a job. Cloud Run services respond to web requests, or events, while jobs perform work and quit when that work is completed.

Cloud Run:

- Provisions a valid TLS certificate, and other configuration to support HTTPS requests.
- Handles incoming requests, decrypts, and forwards them to your application.

Cloud Run expects your container to listen on port 8080 to handle web requests. The port number is a configurable default, so if this port is unavailable to your application, you can change the application's configuration to use a different port. You don't need to provide an HTTPS server, Google's infrastructure handles that for you.

Container runtime contract

- Write your application in any programming language and containerize it using any base image.
- As a service, your container must listen for requests on the correct port.
- Send a response from the container within the configured timeout setting.
- As a job, when successfully completed your container must exit with exit code 0, or if failed, with non-zero exit code.
- Do not implement any transport layer security because HTTPS is transparently handled by Cloud Run.

Here are the key requirements for running containers in Cloud Run.

Your application can be written in any programming language and must be containerized using any base image. Executables in the container image must be compiled for Linux 64-bit.
Cloud Run accepts container images in the Docker Image Manifest V2, Schema 1, Schema 2, and OCI image formats.

When running as a Cloud Run service, your container must listen for requests on the correct port. Your container instance must send a response within the time specified in the request timeout setting (max. 1 hour) after it receives a request, including the container instance startup time. Otherwise, the request is ended and a 504 error is returned.

For Cloud Run jobs, the container must exit with exit code 0 when the job has successfully completed, and exit with a non-zero exit code when the job has failed. Because jobs should not serve requests, the container should not listen on a port or start a web server.

The container should not implement any transport layer security directly because TLS is terminated by Cloud Run for HTTPS and gRPC. Requests are then proxied as HTTP/1 or gRPC to the container. For HTTP/2, your container must handle requests in HTTP/2 cleartext format.

For more detailed information, refer to the container runtime contract documentation.

Cloud Run execution environments

First generation

- Used for services by default, and can be changed.
- Use with services that must scale out quickly.
- Use with services that need short cold start times.
- Use with services with infrequent traffic.
- Use with services that consume less than 512MiB of memory.

Second generation

- Used for jobs by default, and cannot be changed.
- Use when a network file system is required.
- Use with services with steady traffic, and tolerant with slower cold starts.
- Use when CPU-intensive workloads are required.
- Use when unimplemented system calls cause issues in first generation environments.

Cloud Run has two execution environments that run your services and jobs: First generation, and second generation.

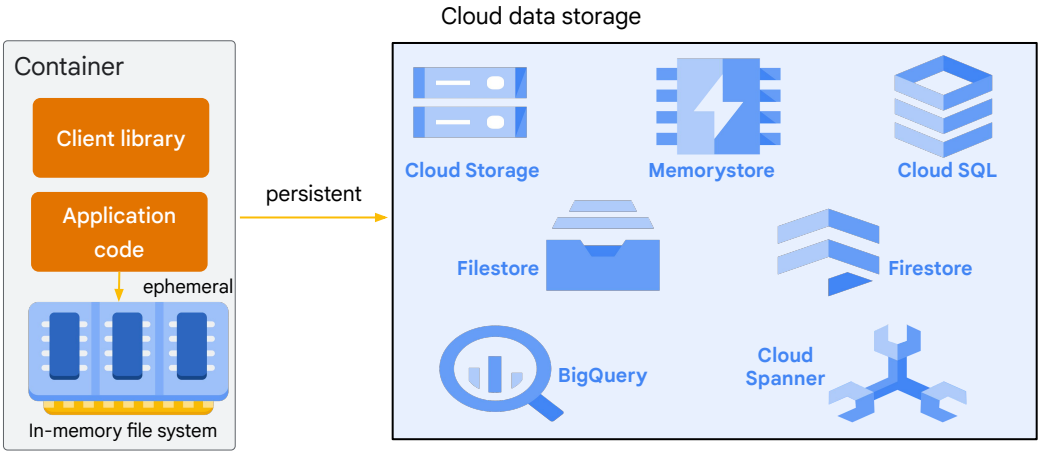
Cloud Run services by default operate within the first generation execution environment, which features shorter cold start times and emulation of most, but not all operating system calls. The second generation execution environment provides full Linux compatibility rather than system call emulation.

You can change the execution environment for services only. Cloud Run jobs automatically use the second generation execution environment, which cannot be changed for jobs. You can choose between the two environments based on the needs of your Cloud Run service.

The second generation execution environment provides:

- Faster CPU performance
- Faster network performance, especially in the presence of packet loss
- Full Linux compatibility, including support for all system calls, namespaces, and cgroups
- Network file system support

File system and data storage access



On Cloud Run, your container has access to a writable in-memory filesystem. Writing to a file from your container uses your container instance's allocated memory.

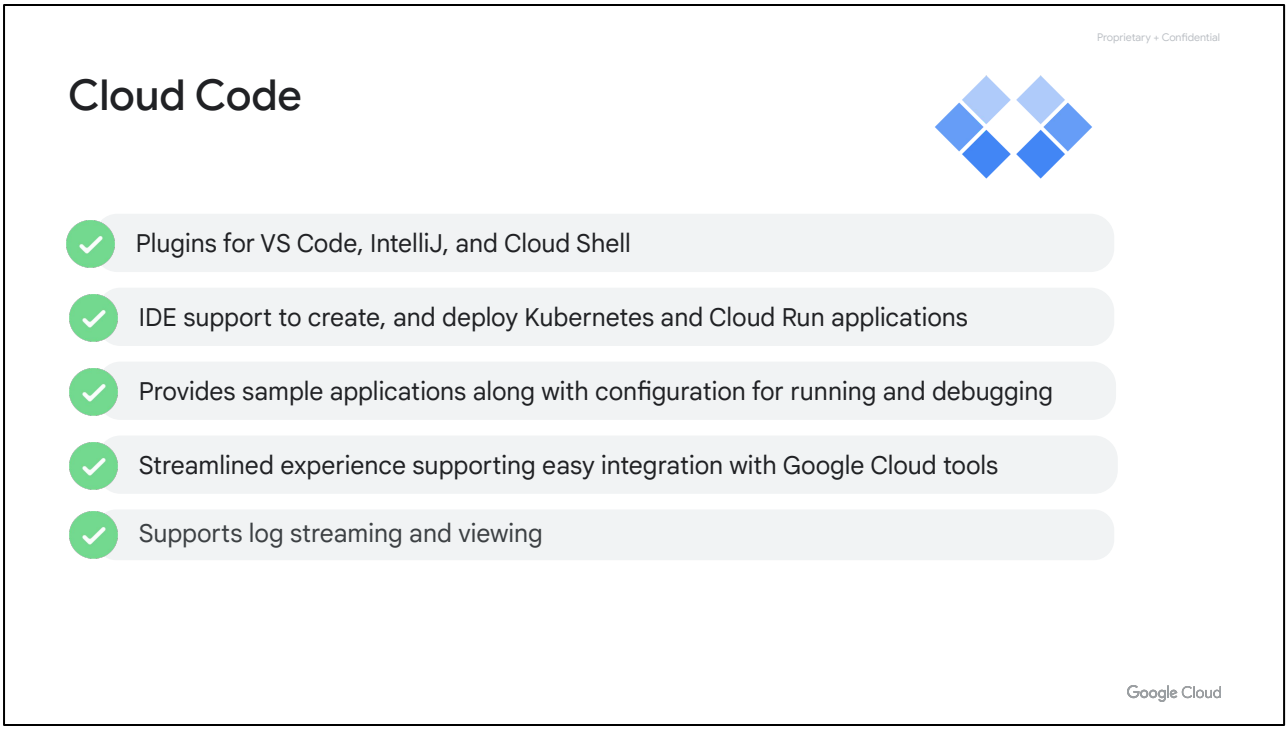
Data written to the file system does not persist when the container instance is stopped. You can use the in-memory file system as a cache, to store disposable per-request data, or configuration.

If you need to persist data beyond a container instance lifetime, and you want to use standard file system semantics, you can use Filestore or other self-managed network file systems with Cloud Run. To use network file systems with Cloud Run, you must specify the second generation execution environment when you deploy your service to Cloud Run.

For more details on setting up a network file system for Cloud Run, refer to the documentation.

To access Cloud Storage as a mounted network file system onto a Cloud Run service, you can use Cloud Storage FUSE.

If you don't need a standard file system, the simplest option is to use cloud data storage client libraries. With these libraries, you can connect your Cloud Run service to Firestore, Cloud SQL, Cloud Spanner, Cloud Storage, Memorystore, and BigQuery storage services on Google Cloud.

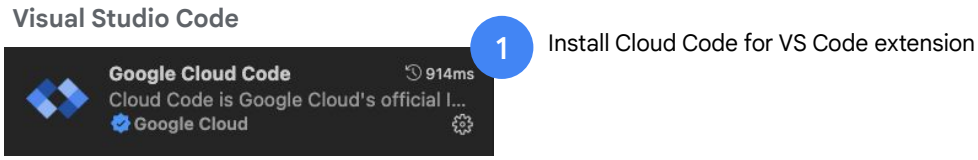


Cloud Code is a set of plugins for popular IDEs that make it easier to create, deploy, and integrate your applications with Google Cloud.

Cloud Code provides IDE support for the full development cycle of Kubernetes and Cloud Run applications, from creating and customizing a new application from sample templates to running your finished application. Cloud Code provides samples, configuration snippets, and a tailored debugging experience — making developing with Kubernetes and Cloud Run a whole lot easier.

While Cloud Code works with any cloud platform, it provides a streamlined experience for easy creation of clusters hosted on Google Cloud and better integration with Google Cloud tools like Cloud Source Repositories, Cloud Storage, and Cloud Client Libraries.

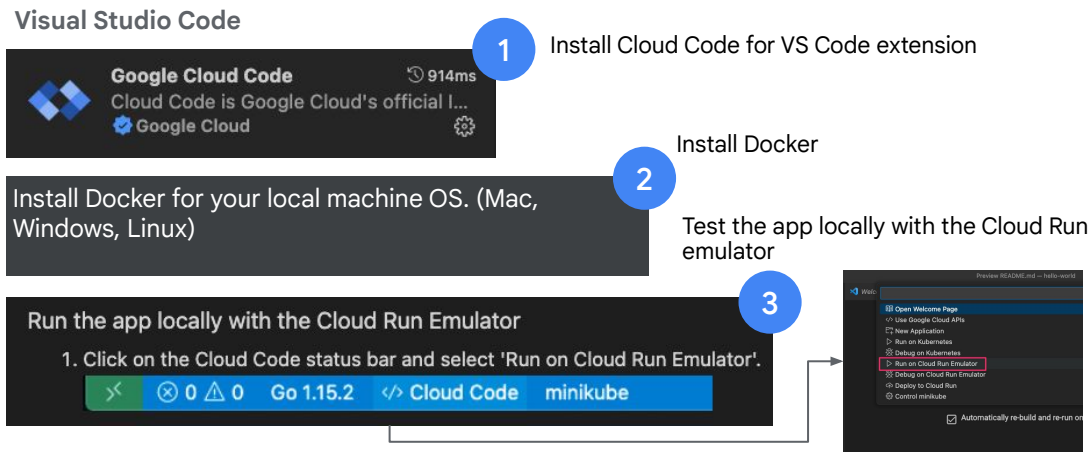
Cloud Code templates



You can create and deploy a Cloud Run service using Intellij, Visual Studio Code (VS Code), or Cloud Shell with a Cloud Code template.

To use Cloud Code, you need to install the required plugins or extensions for your IDE. For example, the Cloud Code for VS Code extension adds support for Google Cloud development in VS Code.

Cloud Code for VS Code

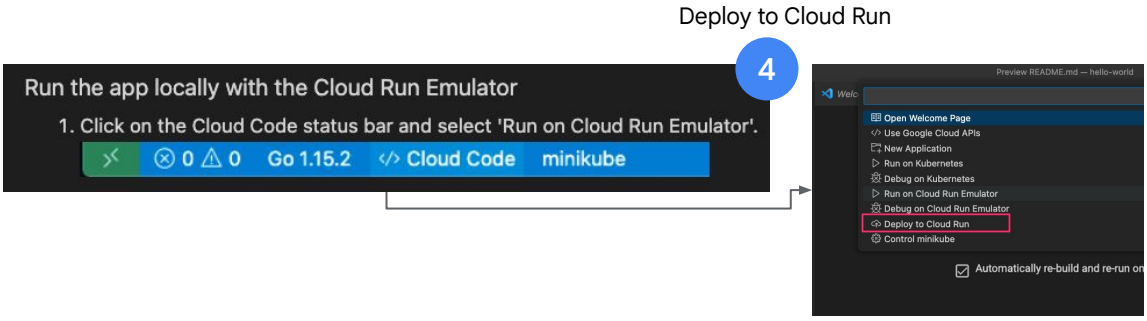


You can develop your app in VS Code and run it locally with the Cloud Run Emulator in VS Code. To build your container image locally with the emulator, install a builder such as Docker on your local machine.

To start the build, from the Cloud Code status bar in VS Code, select Run on Cloud Run Emulator. You can view the progress of the build in the output window in VSCode.

Once successfully completed, a URL to your application is generated and displayed in the output tab in VS Code. To test the application, navigate to the URL in a browser window.

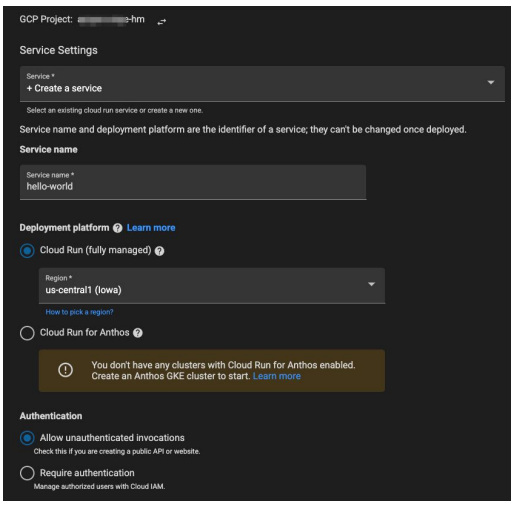
Cloud Code for VS Code - Deploy to Cloud Run



Once you've tested your application locally, you can deploy it to Cloud Run.

To deploy your application to Cloud Run, in the Cloud Code status bar in VS Code, select Deploy to Cloud Run.

Cloud Code for VS Code - Deploy to Cloud Run



To deploy your application to Cloud Run using Cloud Code in VSCode:

- Set your Google Cloud project.
- Provide a name for the service.
- Select Cloud Run (fully-managed), and select a region.
- Configure Kubernetes cluster information (if using Cloud Run for Anthos).
- Provide authentication, container image URL, and service account.
- Build your image locally or remotely with Cloud Build.

In the Deploy to Cloud Run tab, you can set your Google Cloud project, provide a name for your Cloud Run service, and select a Cloud Run region where your service will run. You can also deploy your application to the Cloud Run for Anthos platform, in which case you would need to configure Kubernetes cluster information.

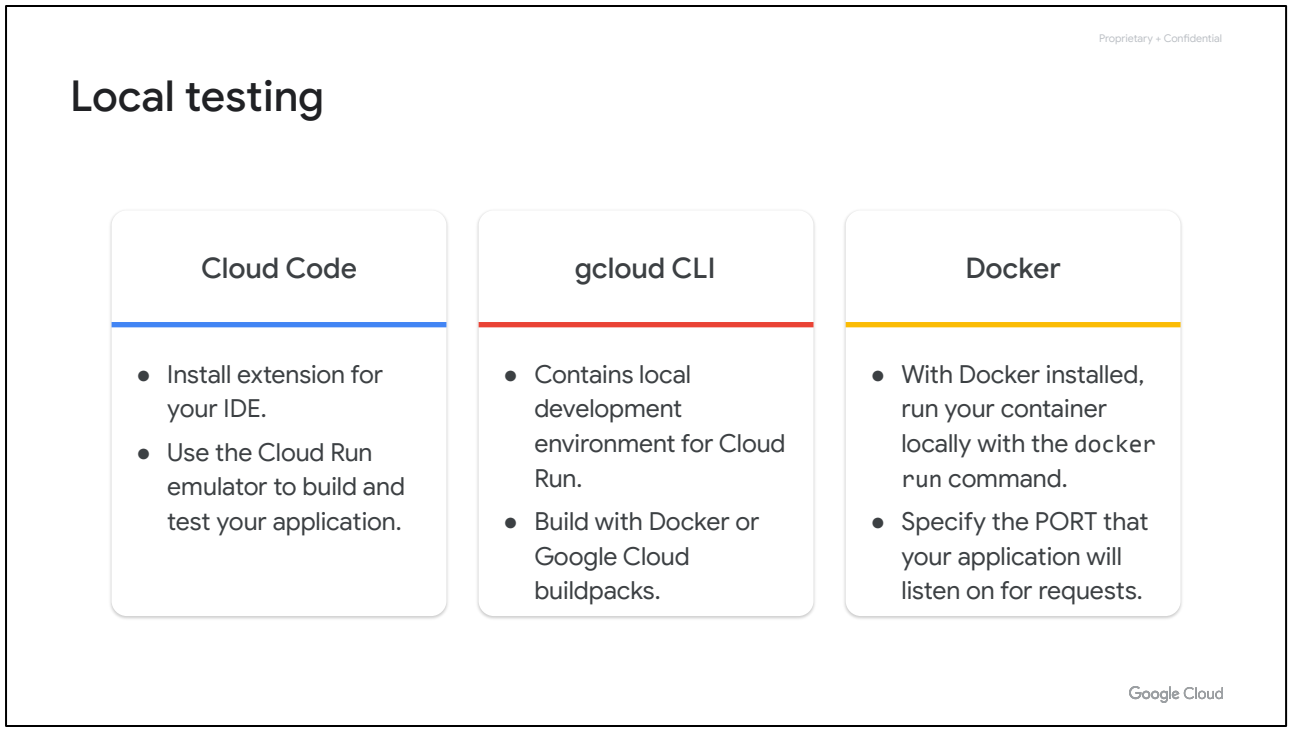
You can also provide additional configuration for your service for authentication, container image URL, and service account.

Choose to build your container image locally, or use Cloud Build to build your image remotely.

Cloud Code for VS Code builds your image, pushes it to the registry, and deploys your service to Cloud Run.

The live URL to your service is displayed in the output tab in VSCode which you can access to test your application.

You can extend Cloud Code with custom templates. For more information on this topic, read the blog post.



During development, you can run and test your container locally, prior to deploying it to Cloud Run. To run and test locally, you can use Cloud Code or Docker that you can install on your machine.

As discussed previously, the Cloud Code plugin for supported IDEs lets you locally run and debug your container image in a Cloud Run emulator within your IDE. The emulator lets you configure an environment that is representative of your service running on Cloud Run.

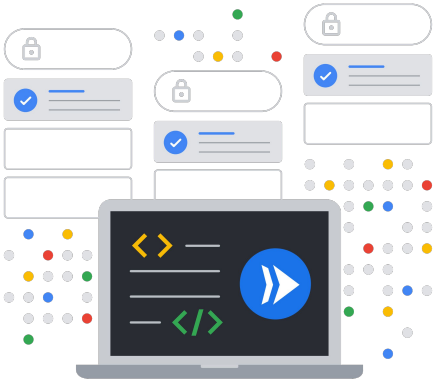
You can configure properties like CPU and memory allocation, specify environment variables, and set Cloud SQL database connections in Cloud Code.

Google Cloud CLI contains a local development environment for emulating Cloud Run that can build a container from source, run the container on your local machine, and automatically rebuild the container upon source code changes. If a Dockerfile is present in the local directory, it's used to build the container. If no Dockerfile is present, the container is built with Google Cloud's buildpacks. To test your service locally, visit http://localhost:8080/ in your browser.

To test your container image locally using Docker, use the docker run command, providing the container image URL, and the port that your application will listen on for HTTP(S) requests. To test your service locally, visit http://localhost:port/ in your browser.

Remember

1. Use Cloud Run to run containerized applications.
2. Build your source code with Buildpacks and deploy to Cloud Run.
3. Your service must listen for requests on the configured port, and return a response within a specified time.
4. Use Cloud Code with popular IDEs to easily create, deploy, and integrate applications with Google Cloud.
5. Test your application locally with Cloud Code, the gcloud CLI, or with Docker before deploying to Cloud Run.



In summary:

- Use Cloud Run to run containerized applications that are written in any programming language.
- With the source-based approach, build your source code with Buildpacks and deploy to Cloud Run.
- As a service running on Cloud Run, your container must listen for requests on the configured port, and return a response within a specified time.
- Use Cloud Code with popular IDEs to easily create, deploy, and integrate your applications with Google Cloud.
- Test your application locally with Cloud Code, the gcloud CLI, or with Docker before deploying to Cloud Run.