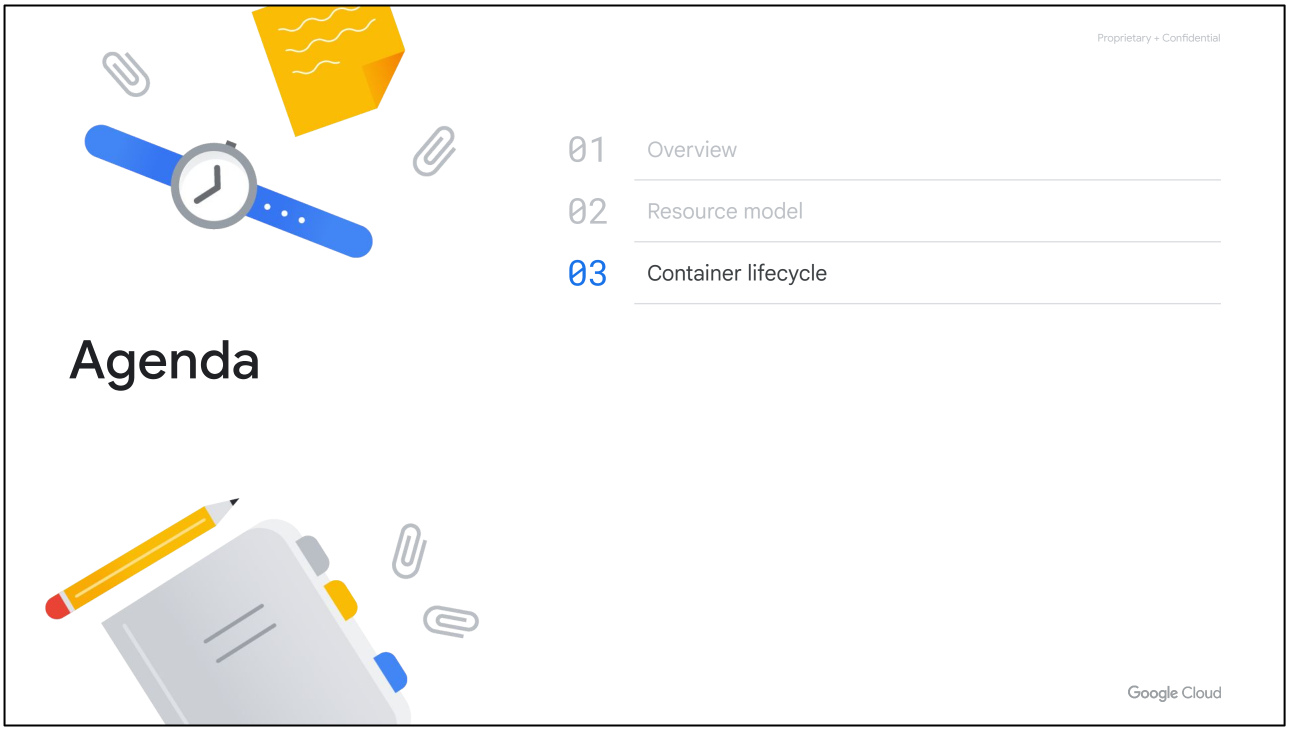
Google Cloud

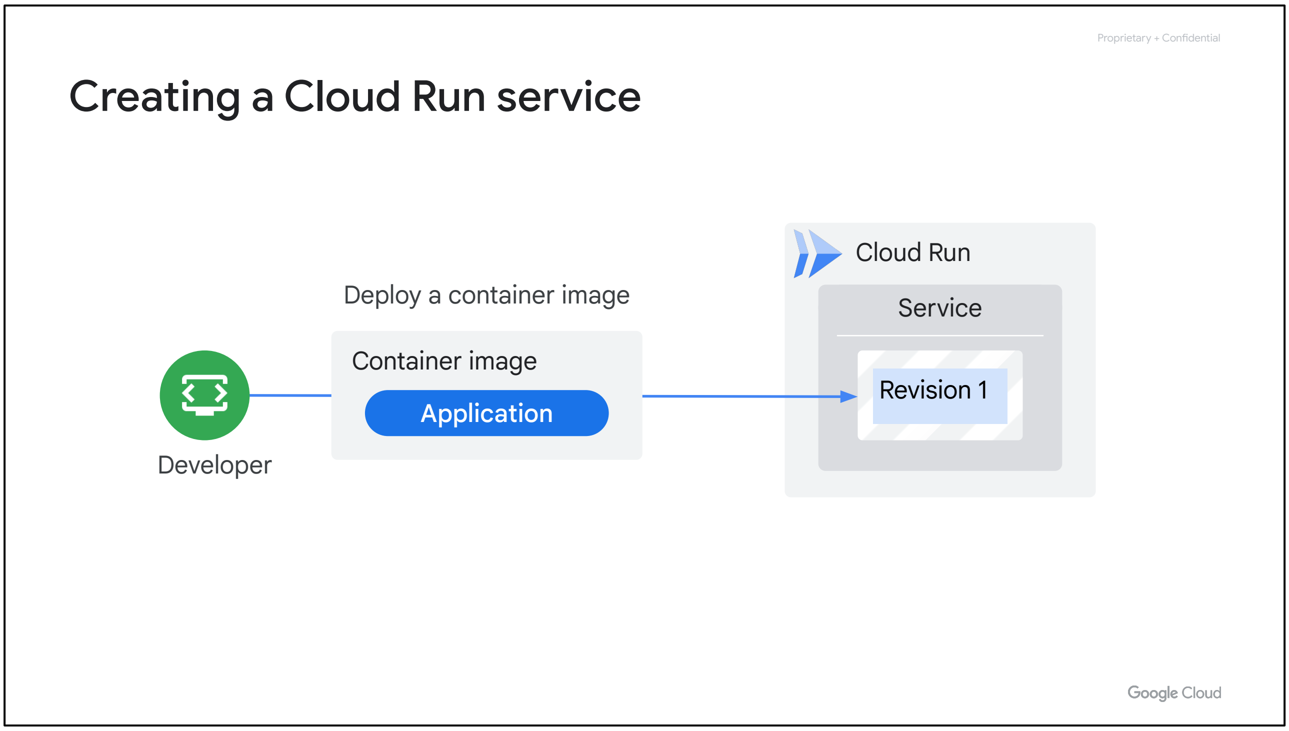
Fundamentals of Cloud Run

In this module, we discuss the fundamentals of developing and running applications on the Cloud Run platform.



Agenda

Let's discuss the lifecycle of a container on Cloud Run.



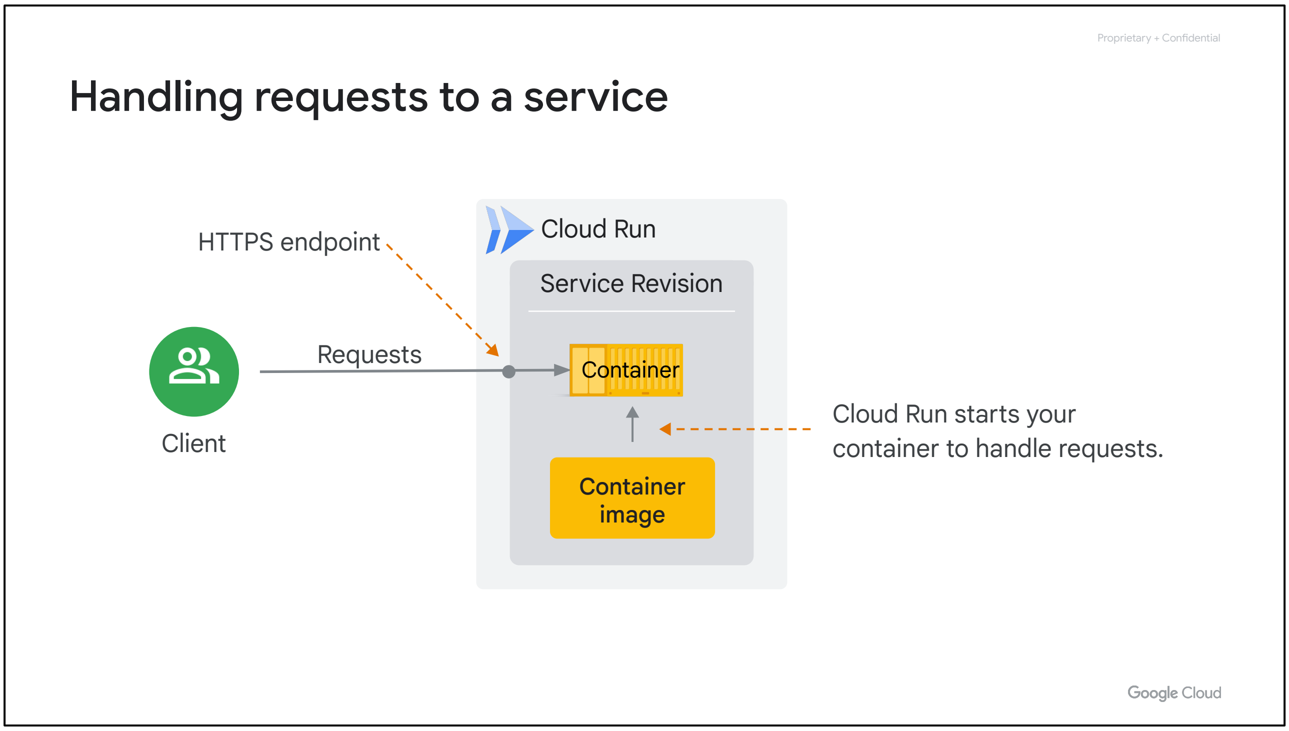
Creating a Cloud Run service

When you deploy a container image for the first time to Cloud Run, a service, and its first revision is created.

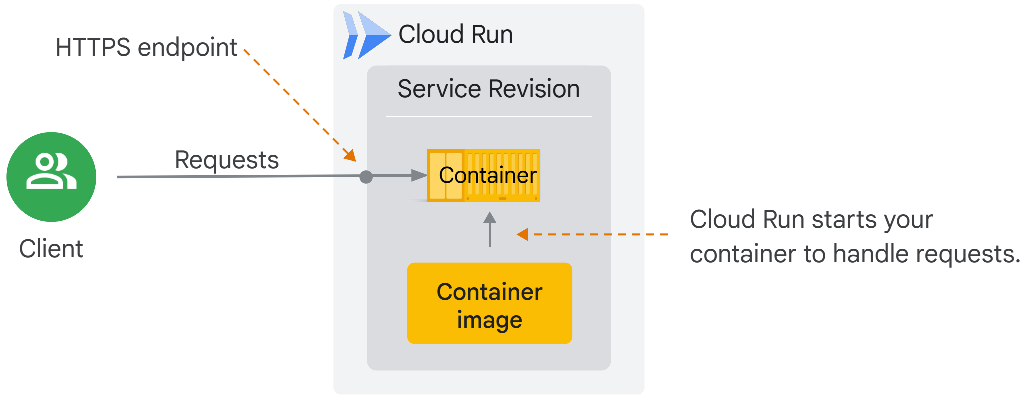
You can deploy a container image from the Google Cloud console, the gcloud CLI, or from a YAML configuration file.

Each service has a unique and permanent HTTPS endpoint URL that does not change over time.

In Cloud Run, you interact primarily with a service resource to perform tasks such as deploying a new container image, rolling back to a previously deployed revision, and changing configuration settings such as environment variables and scaling.

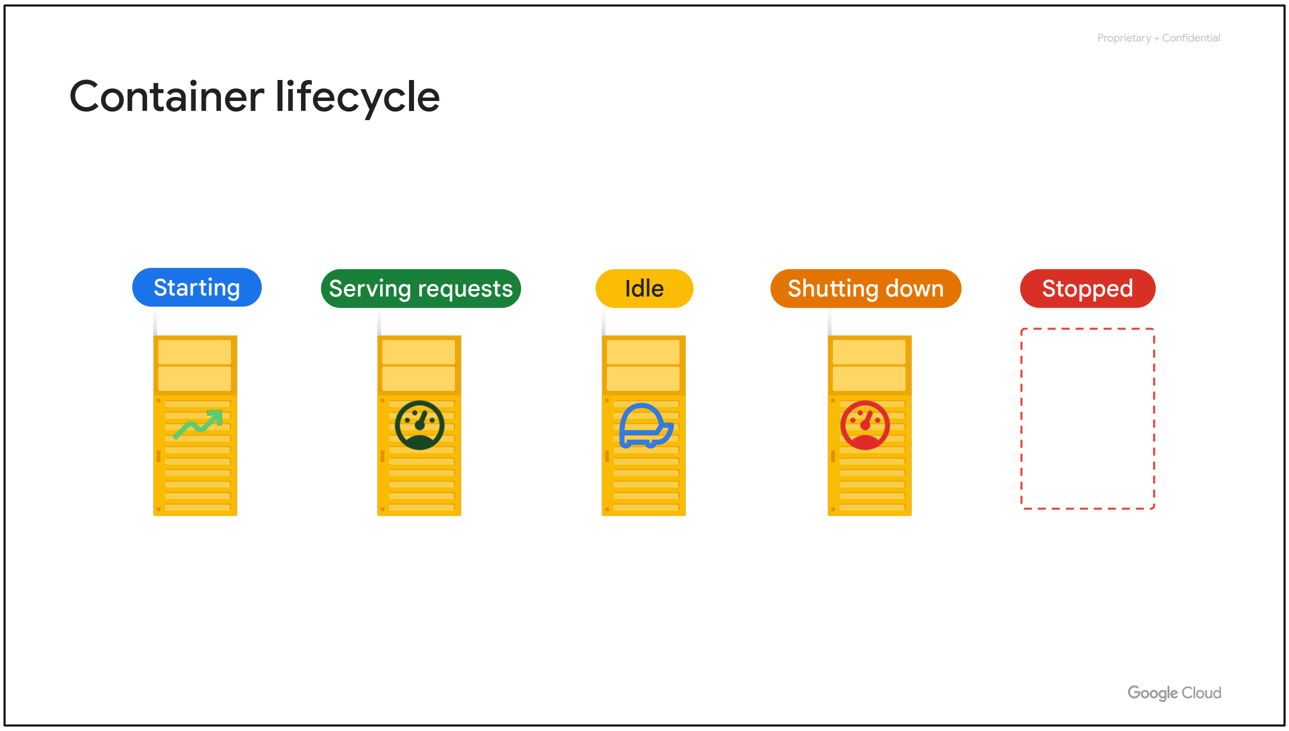


Handling requests to a service



The service HTTPS endpoint is on a subdomain of the run.app domain.

To handle requests that are sent to the service endpoint URL, Cloud Run starts a container and forwards the requests to the container.



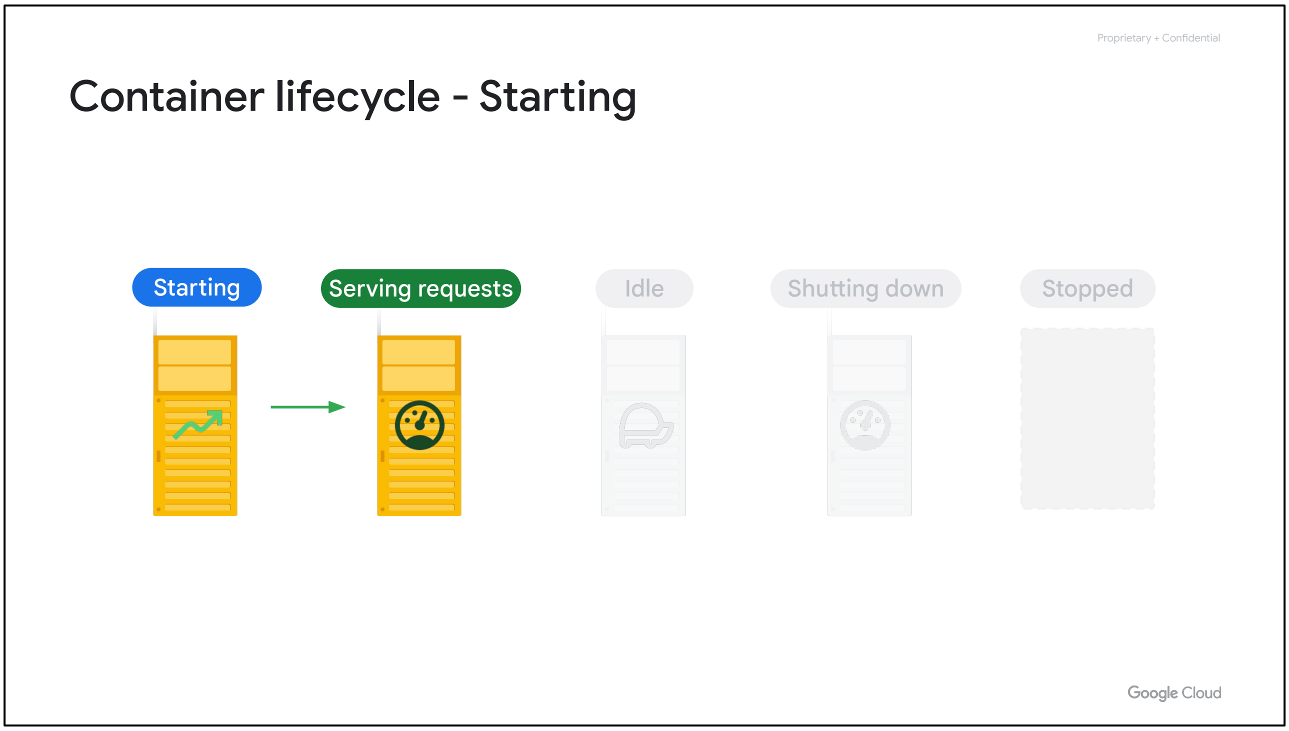
Container lifecycle

The relevant states of a container are:

- **Starting**
  The starting state is when Cloud Run materializes the container image and starts your application.
- **Serving requests**
  This is when your container is handling web requests.
- **Idle**
  The container is idle when it is not handling web requests.
- **Shutting down**
  If you handle the shutdown hook, Cloud Run lets you stop your application gracefully.
- **Stopped**
  The final state in the lifecycle is when the container is stopped.

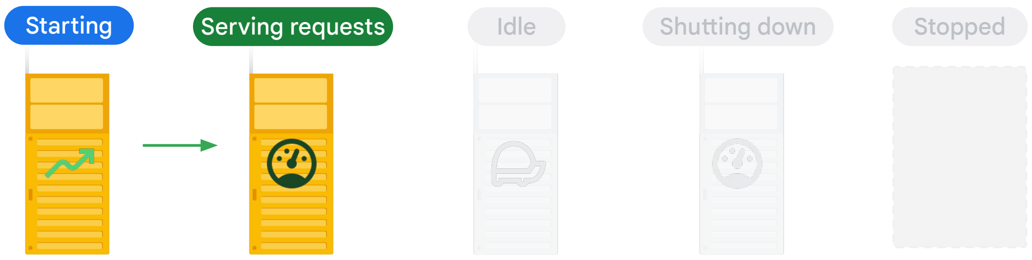
Let's review the states, from starting to stopped and all the transitions between them.

https://cloud.google.com/blog/topics/developers-practitioners/lifecycle-container-cloud-run



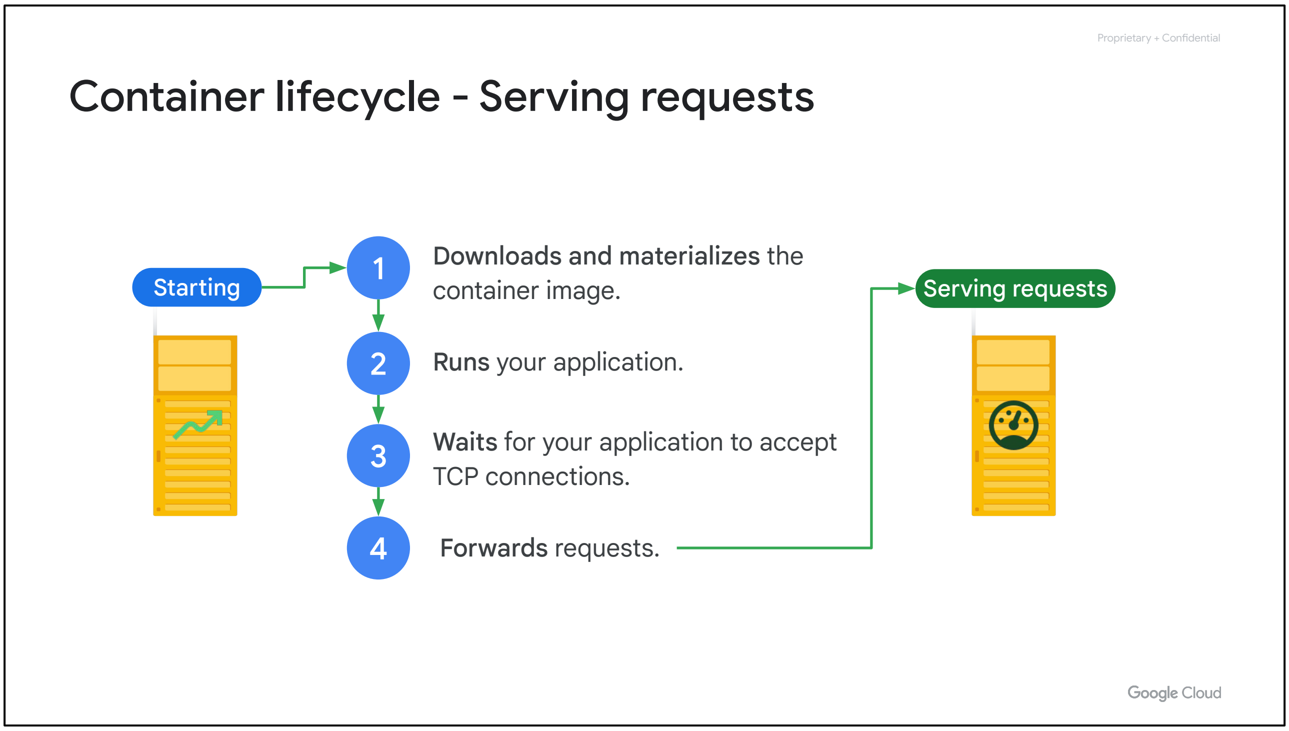
Proprietary + Confidential

Container lifecycle - Starting



Google Cloud

The Starting phase begins when Cloud Run pulls a container image, and ends when the container starts to serve requests.

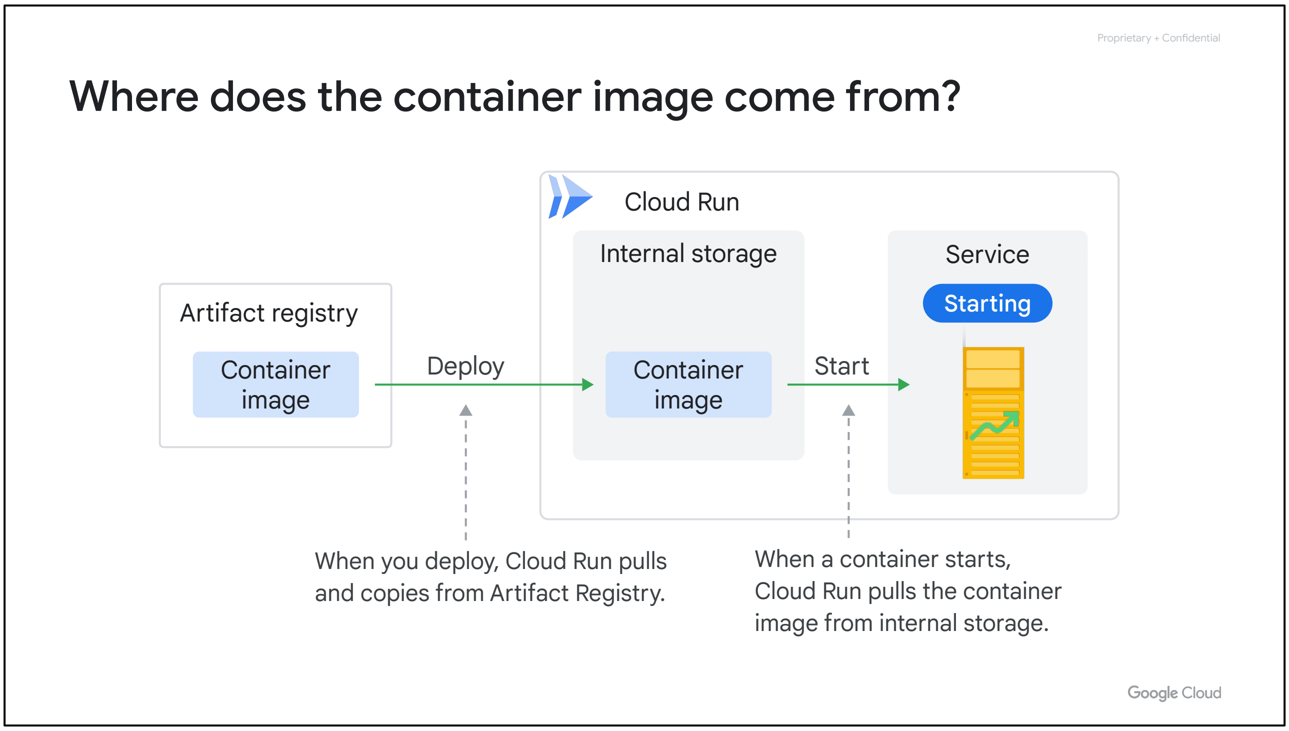


Container lifecycle - Serving requests

Starting
1 Downloads and materializes the container image.
2 Runs your application.
3 Waits for your application to accept TCP connections.
4 Forwards requests.
Serving requests

Starting a container requires four steps:

1. Cloud Run creates the container's root file system by materializing the container image.
2. Once the container file system is ready, Cloud Run runs the entrypoint program of the container (your application).
3. While your application is starting, Cloud Run continuously probes port 8080 to check whether your application is ready. (You can change the port number if you need to.)
    - You can configure HTTP, TCP, and gRPC startup health check and liveness probes for new and existing Cloud Run services using a YAML file.
    - A startup probe can be used to determine when a container has started and is ready to accept traffic.
      For more information on setting up these types of probes, view the documentation on container health checks.
4. Once your application starts accepting TCP connections, Cloud Run forwards incoming web requests to your container. Make sure that your application only opens the port when it's ready to handle requests.


Proprietary + Confidential

Where does the container image come from?

When you deploy, Cloud Run pulls and copies from Artifact Registry.

When a container starts, Cloud Run pulls the container image from internal storage.

Google Cloud

Cloud Run pulls container images from internal storage.

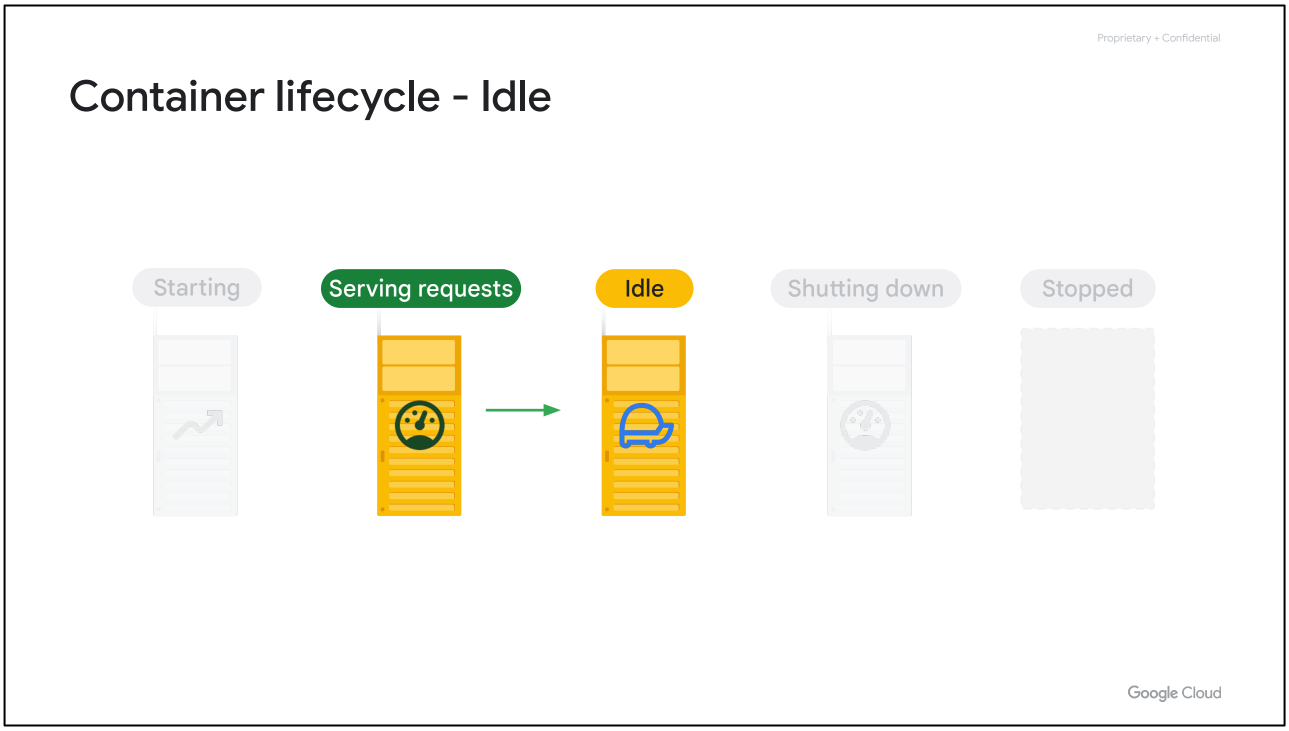
However, there are two distinct events when Cloud Run pulls a container image. And there are two different sources Cloud Run can pull from.

- The first is when you **deploy** a container image for the first time.
- The second is when Cloud Run **starts a new container.**

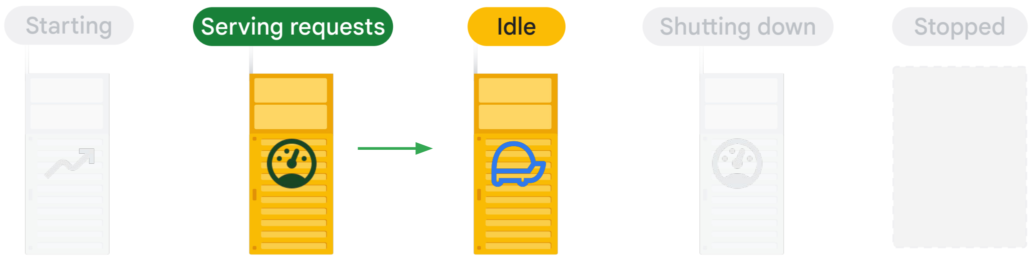
When you **deploy** a new container image, Cloud Run pulls and copies the container image from Artifact Registry. Cloud Run then stores the image in its internal storage, and every time it **starts** a new container, it pulls the container image from there.

This internal storage is optimized to make sure that large container images load just as fast as tiny ones.

Because Cloud Run copies the container images, it also insulates your service from failures in Artifact Registry, or when you accidentally remove a deployed container image from Artifact Registry.

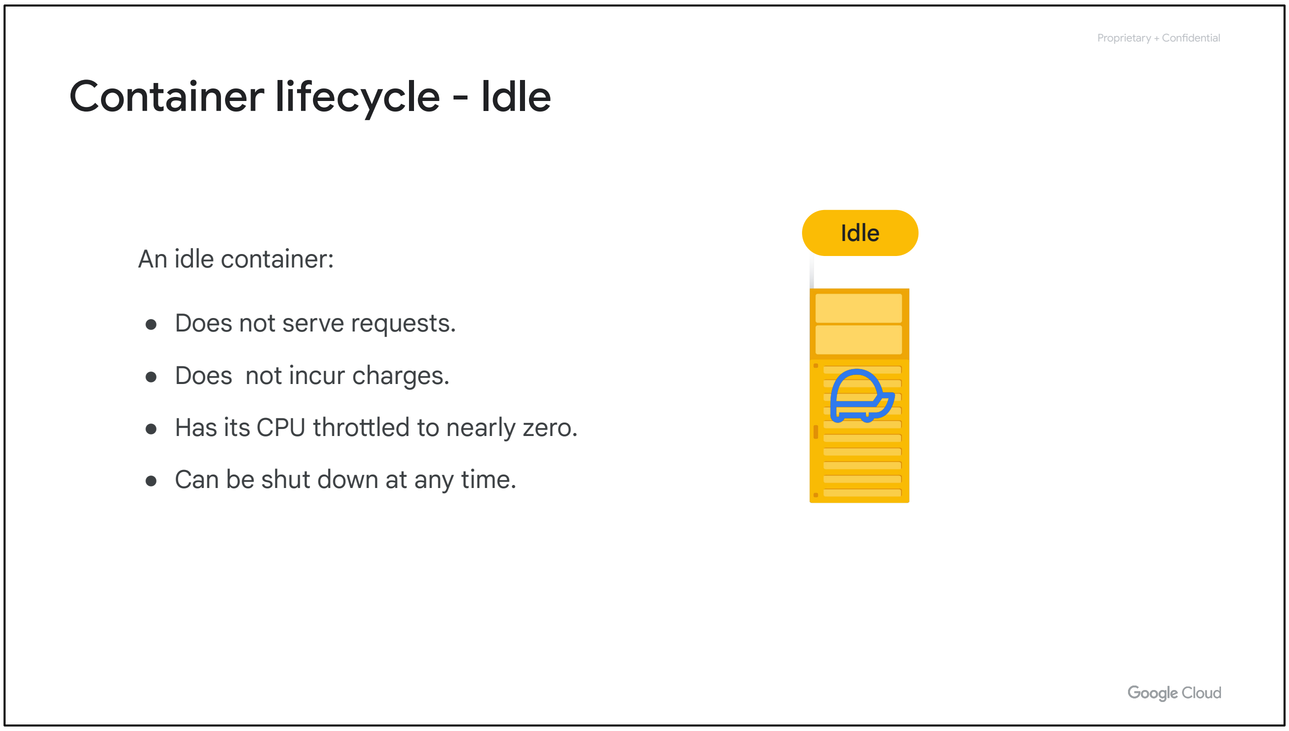


Container lifecycle - Idle



A container is in the serving requests state as long as it handles web requests.

If a container handles no requests for 100 ms, the container transitions into the *idle* state.


Proprietary + Confidential

Container lifecycle - Idle

An idle container:

- Does not serve requests.
- Does not incur charges.
- Has its CPU throttled to nearly zero.
- Can be shut down at any time.

Google Cloud

An idle container does not serve requests. An idle container does not incur costs, which means you're not charged.

Idle containers on Cloud Run have these limitations:

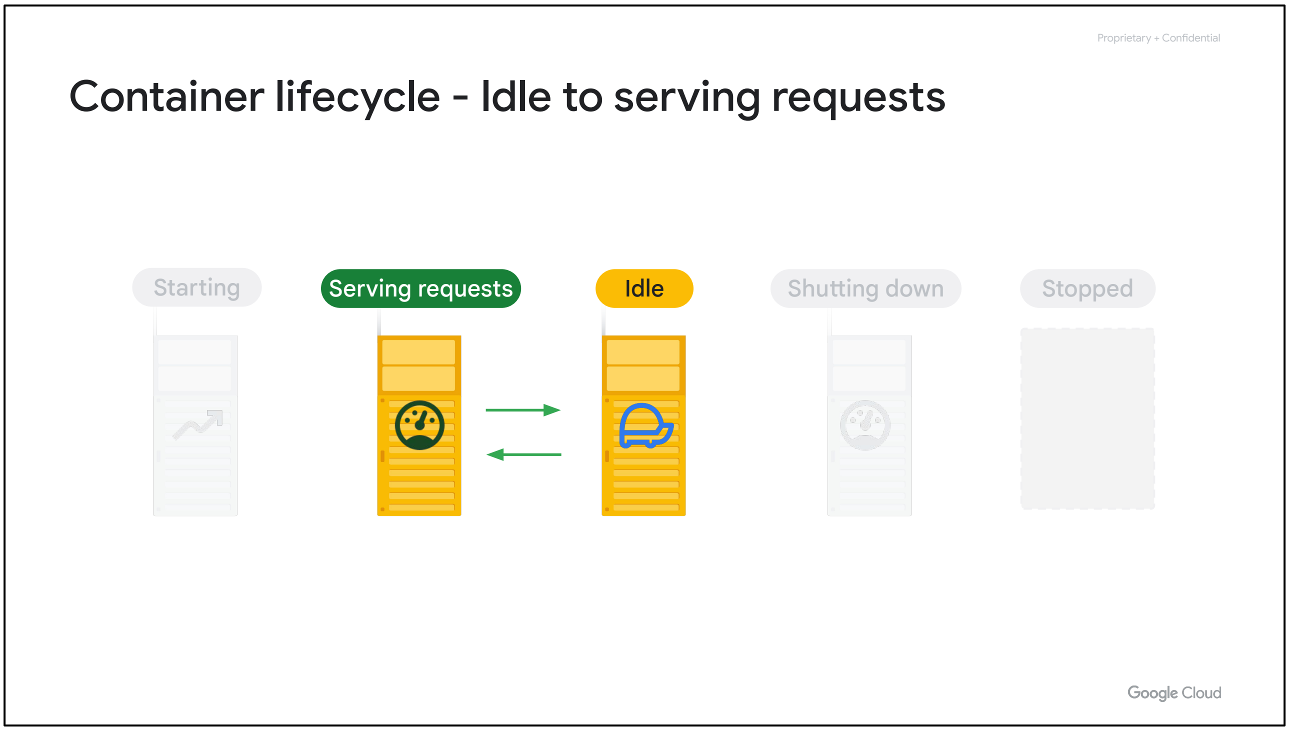
- Cloud Run throttles the CPU of an idle container to nearly zero, which means that your application will run very slowly.

  You can change this behavior so CPU is always allocated and available even when there are no incoming requests. Setting the CPU to be always allocated can be useful for running short-lived background tasks and other asynchronous processing tasks. If you choose the CPU always allocated setting, you are charged for the entire lifecycle of the container instance. For more information on this setting view the documentation.

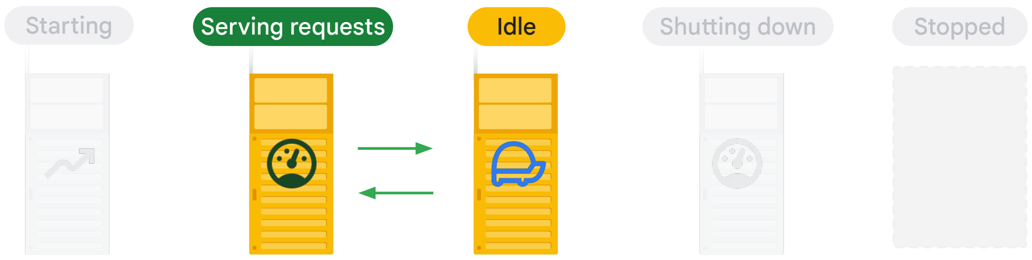
- When your container's CPU is throttled, you can't reliably perform background tasks on your container. To schedule tasks on Cloud Run, you can use Cloud Tasks.

- Network requests to third parties are likely to fail while the container is idle.

- An idle container can be shut down at any time. You do, however, have a lifecycle hook you can use to shut down gracefully.



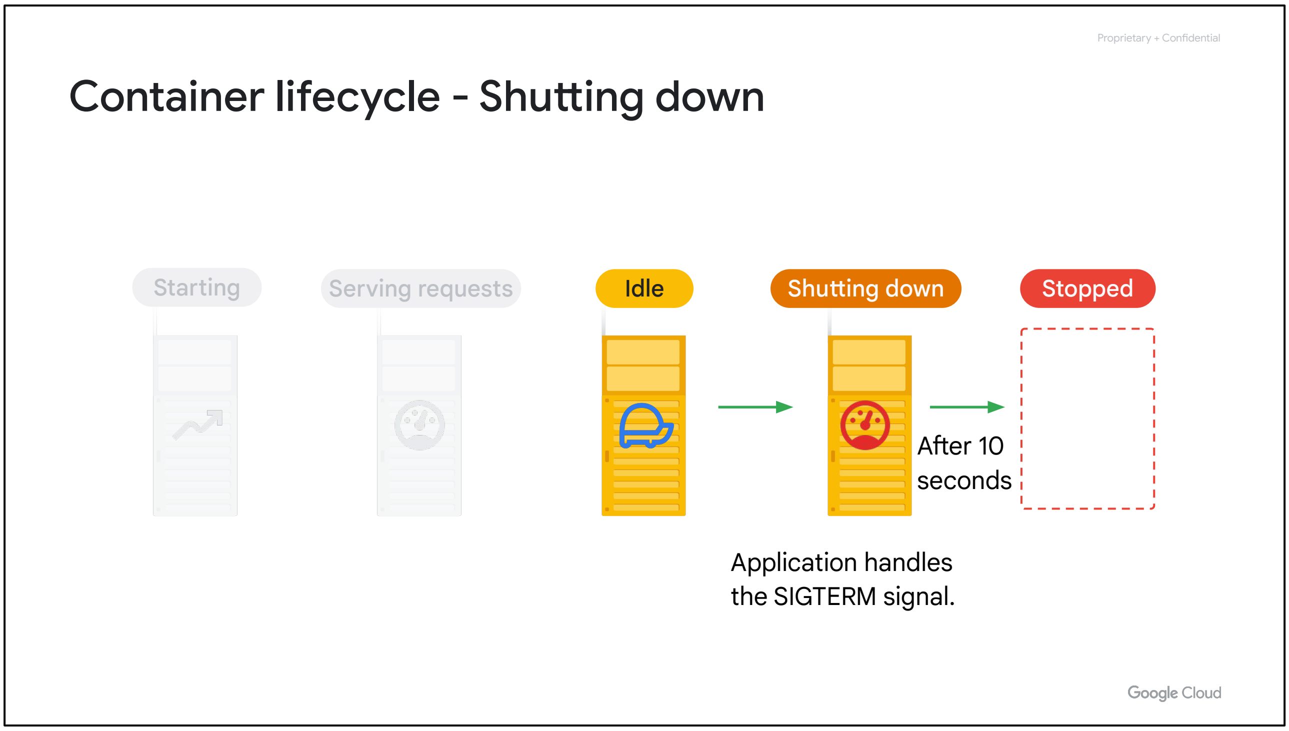
Container lifecycle - Idle to serving requests



Containers can go from idle to serving requests and back again many times.

When a container handles a request after being idle, the container goes from the idle state to serving requests. During this state transition, Cloud Run will unthrottle the container's CPU, and return full access to the container immediately. Your application, and users won't notice any lag.

To handle traffic spikes and minimize cold starts, Cloud Run may keep some instances idle for a maximum of 15 minutes.
The *minimum instances* setting ensures that Cloud Run always keeps a certain number of container instances ready to serve requests.



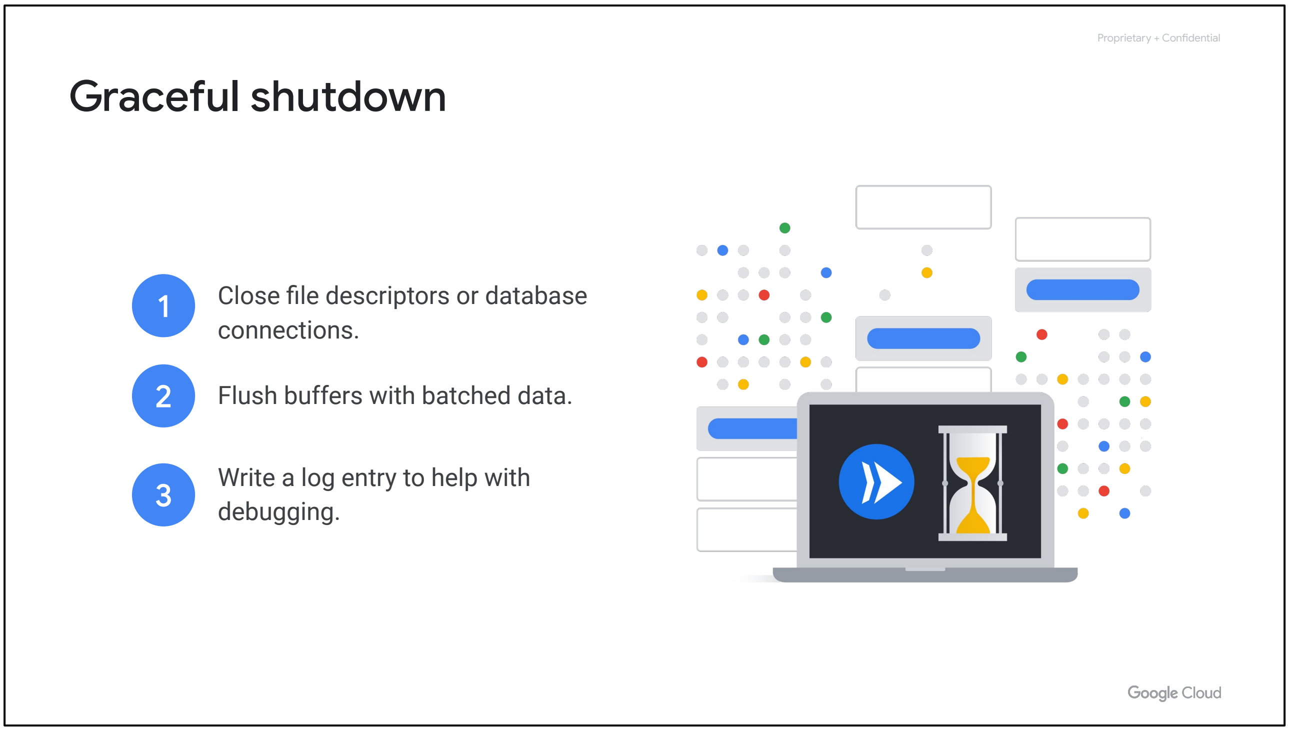
Container lifecycle - Shutting down

Application handles the SIGTERM signal.

If your container is idle, Cloud Run can decide to stop it. By default, a container just disappears when it's shut down.

However, you can build your application to handle a SIGTERM signal. The SIGTERM signal warns your application that shutdown is imminent. That gives the application 10 seconds to clean up things before the container is removed, such as closing database connections or flushing buffers with data.

You can learn how to handle SIGTERMs on Cloud Run so that you can gracefully shut down your application.



Graceful shutdown

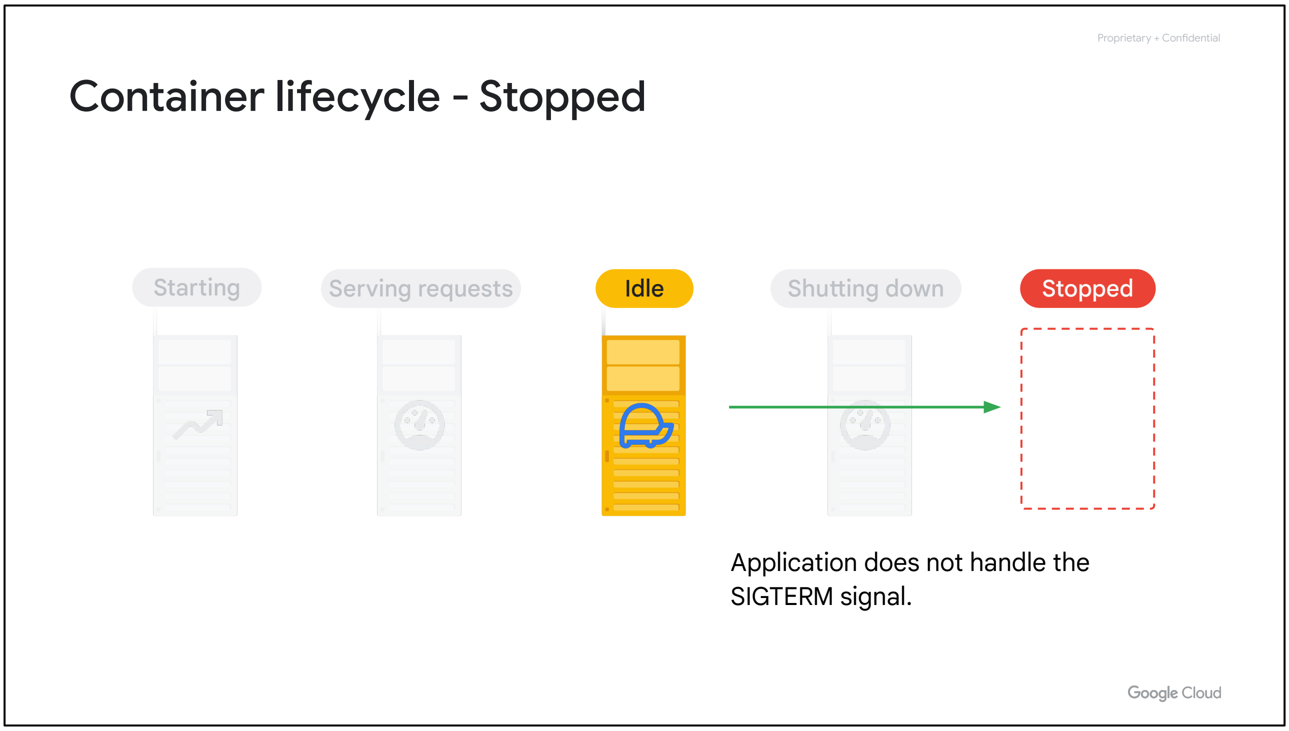
1. Close file descriptors or database connections.
2. Flush buffers with batched data.
3. Write a log entry to help with debugging.

Stopping gracefully allows your application time to clean up. Examples of things your application can do are:

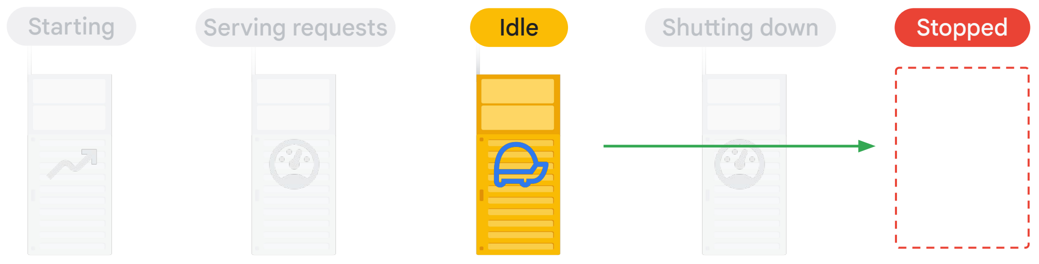
- Close open TCP connections, file descriptors, and database connections. Most downstream systems are slow to time out idling connections, If your application experiences much scaling up and down, you might hit maximum connection errors if you don't close connections.
- Flush any *buffers* that you have with data, such as telemetry data, or any other data that you batch before sending.
- Finally, it's useful to write a log that'll help with debugging later.

Most programming languages provide libraries to trap termination signals like SIGTERM and run routines before your application terminates.

For more information on gracefully shutting down your application on Cloud Run, view this blog post.

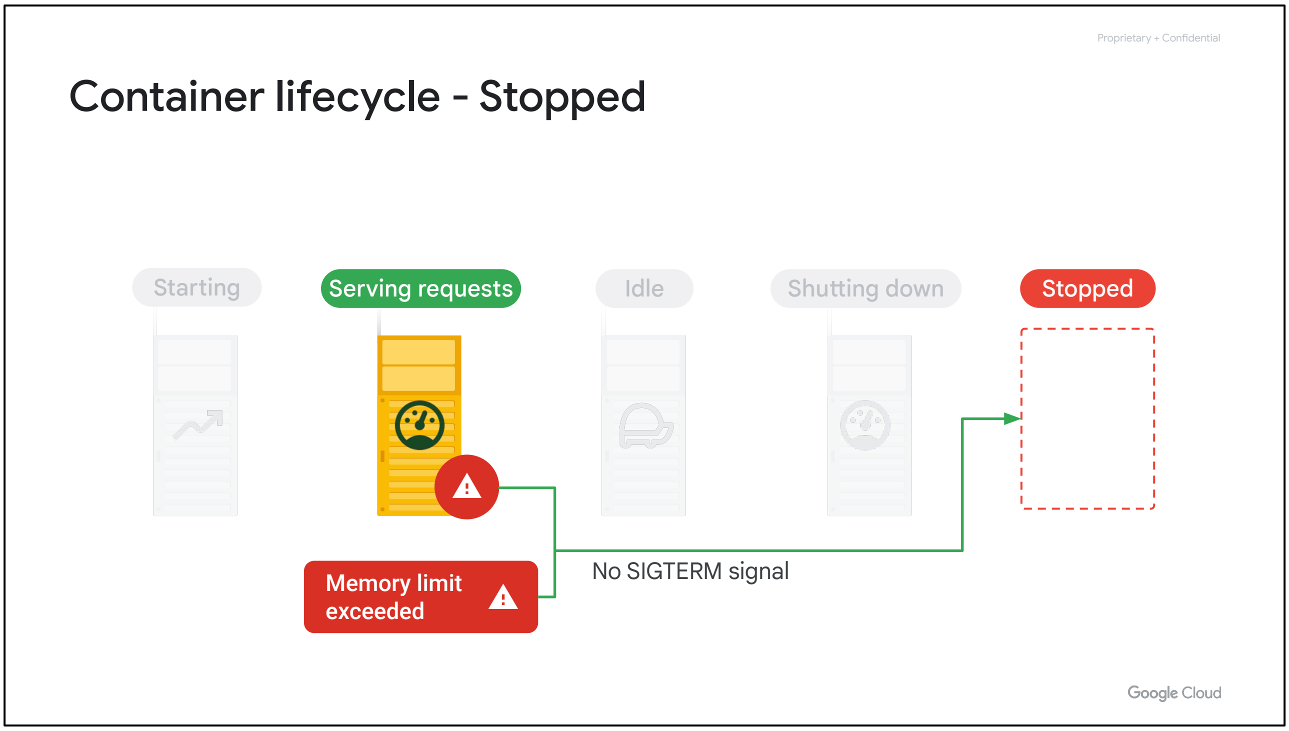


Container lifecycle - Stopped



Application does not handle the SIGTERM signal.

If your application does not handle the SIGTERM signal, Cloud Run immediately stops the container. In this case, the process just stops and disappears.



Container lifecycle - Stopped

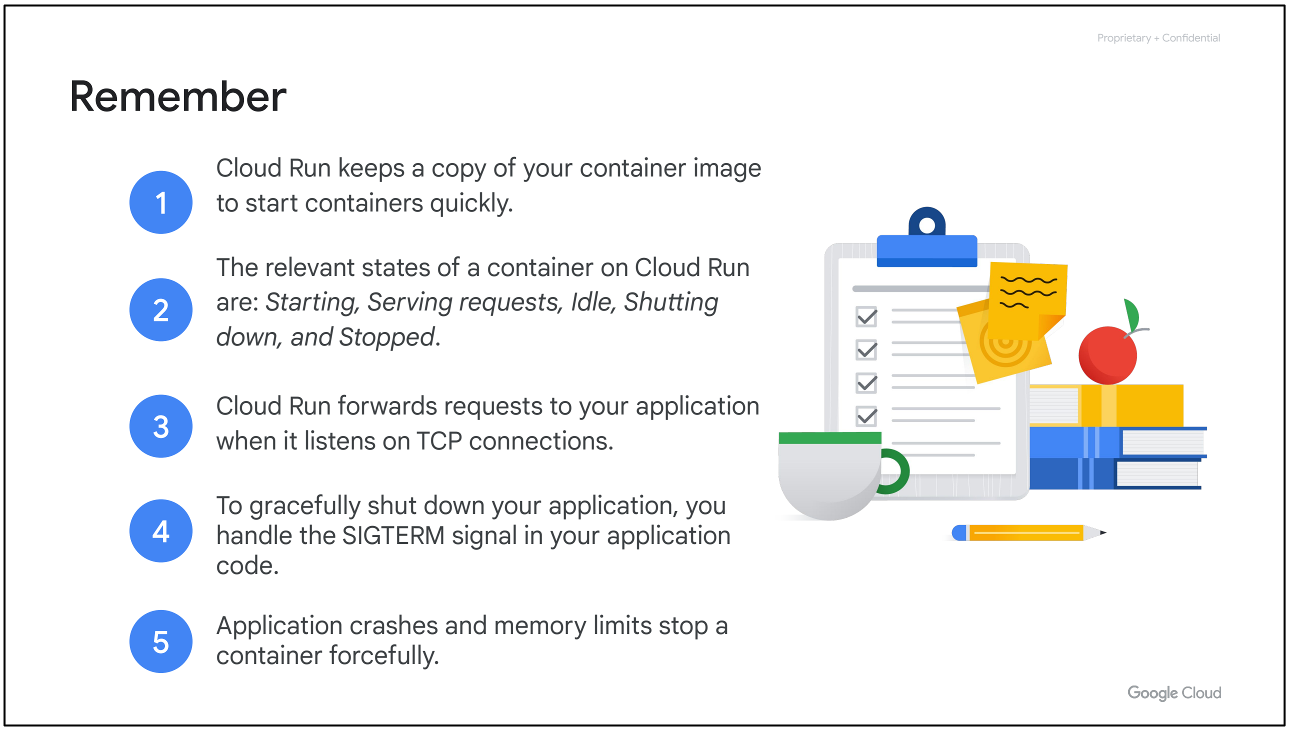
Cloud Run never stops a container while it serves requests under normal circumstances.

However, a container can stop suddenly:

- If your application exits (for instance due to an error in your application code), or
- If the container exceeds the memory limit. By default, the memory allocated to each container instance of a revision or job is 512 MiB.

If a container stops while it's handling requests, all in-flight requests are terminated and fail with an error. While Cloud Run starts a replacement container, new requests might have to wait.

To avoid running out of memory, you can configure memory limits. By default, a container is allocated 512 MiB of memory on Cloud Run, but you can increase the allocation to 32 GiB.



Remember

1. Cloud Run keeps a copy of your container image to start containers quickly.
2. The relevant states of a container on Cloud Run are: *Starting, Serving requests, Idle, Shutting down, and Stopped*.
3. Cloud Run forwards requests to your application when it listens on TCP connections.
4. To gracefully shut down your application, you handle the SIGTERM signal in your application code.
5. Application crashes and memory limits stop a container forcefully.

In summary:

Cloud Run keeps a copy of your container image in its internal storage to start containers fast. This way, large container images load just as quickly as tiny container images.

When your application listens on TCP connections, Cloud Run decides it is ready to handle web requests, and starts to forward them to the container.

There's no way to control *when* a container stops, but in order to shut down gracefully, your application should catch the SIGTERM signal. When it does, it gets 10 seconds to clean up.

Finally, Cloud Run never stops a container that is handling requests, but application crashes and memory limits can stop a container that is serving requests.