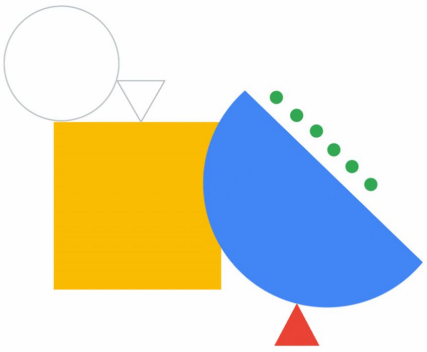
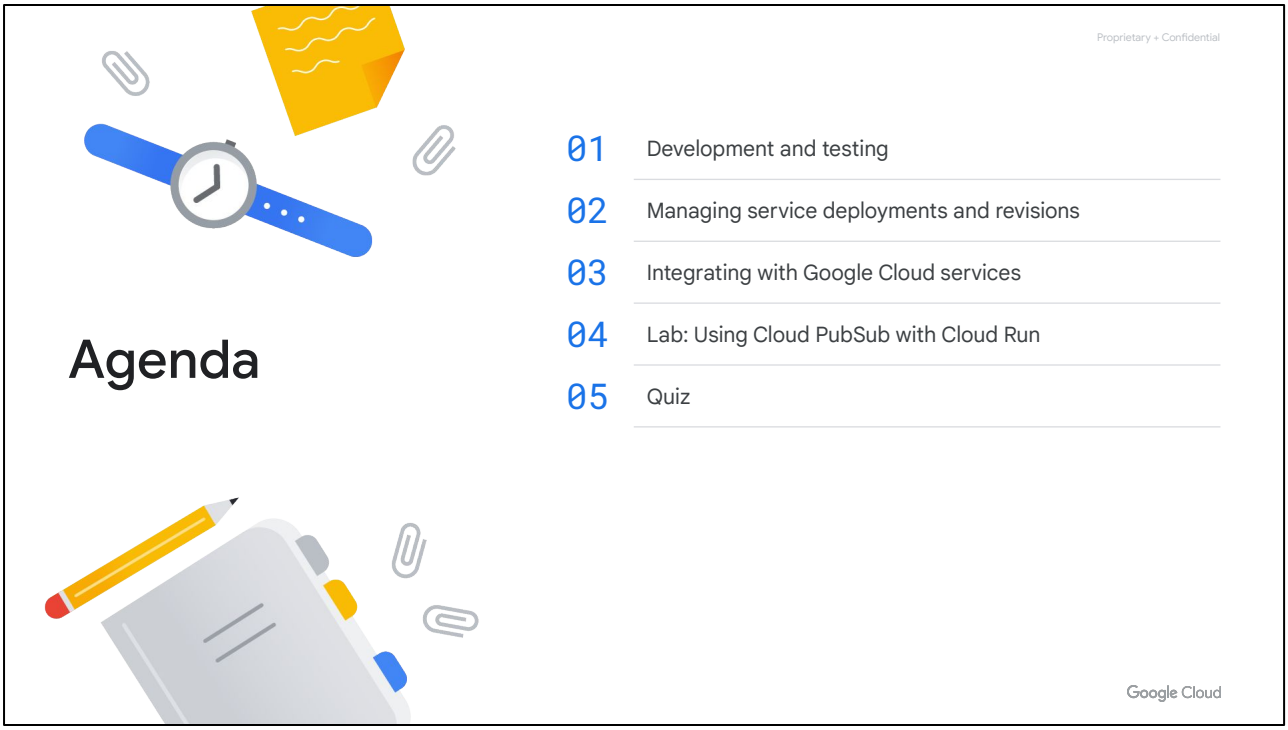
Google Cloud

Application Development, Testing, and Integration



In this module, you learn how to develop and test applications to run on Cloud Run, and manage service deployments and revisions on Cloud Run,

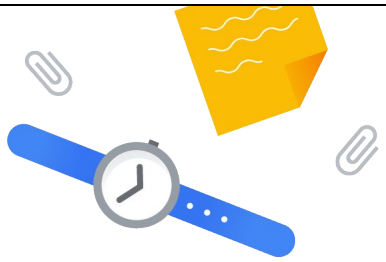


Here is a list of topics that will be discussed in this module.

We'll first discuss the process and tools that you can use to develop and test applications for Cloud Run.

You'll learn how to integrate your Cloud Run service with other Google Cloud services, and complete a lab.

We'll end the module with a short quiz on the topics that were discussed.



01 Development and testing

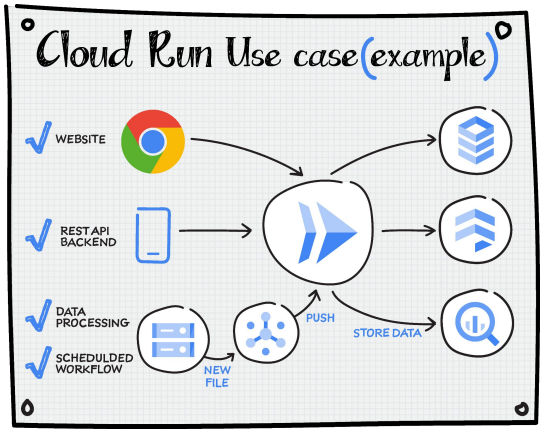
Agenda

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.

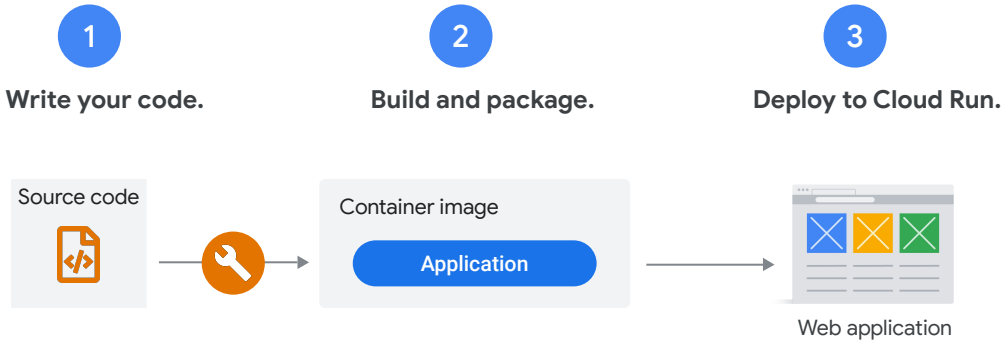


To be a good fit for Cloud Run, your application needs to meet *all* of the following criteria.

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

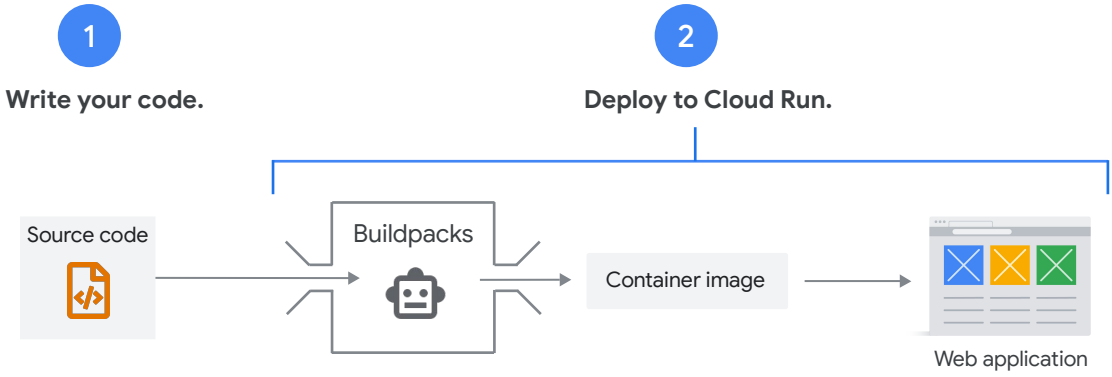
If your app meets those criteria, then it's a good fit for Cloud Run!

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.

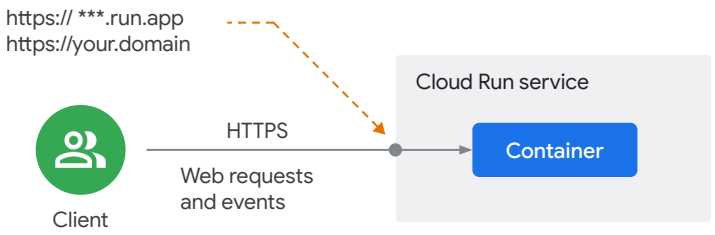
Cloud Run developer workflow



Building container images before deployment 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.

Services on Cloud Run 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, whereas 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 by 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 main requirements for running containers in Cloud Run.

Your application can be written in any programming language and must be containerized by 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

- Is 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 512 MiB of memory.

Second generation

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

Google Cloud

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

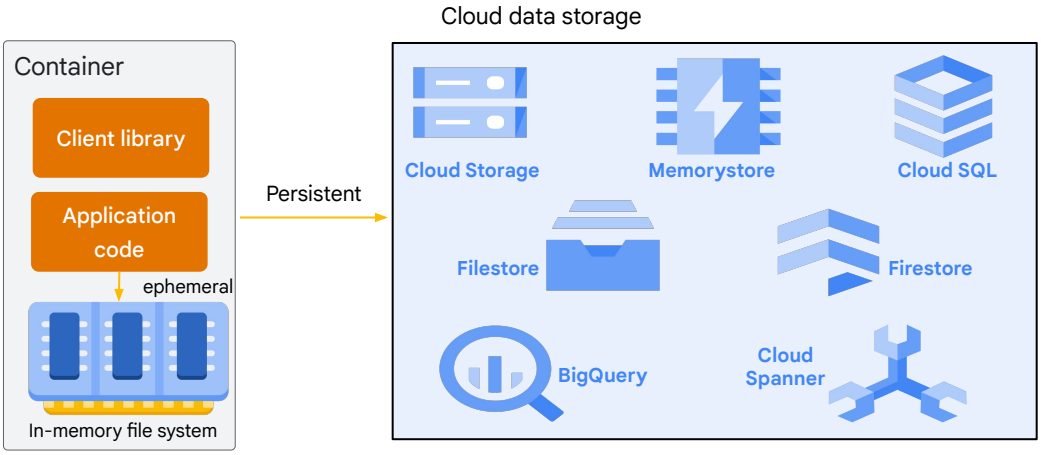
Cloud Run services by default operate within the first generation execution environment, which features shorter cold start times and emulation of most, but not all operating system calls. The second generation execution environment provides full Linux compatibility instead of system call emulation.

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.

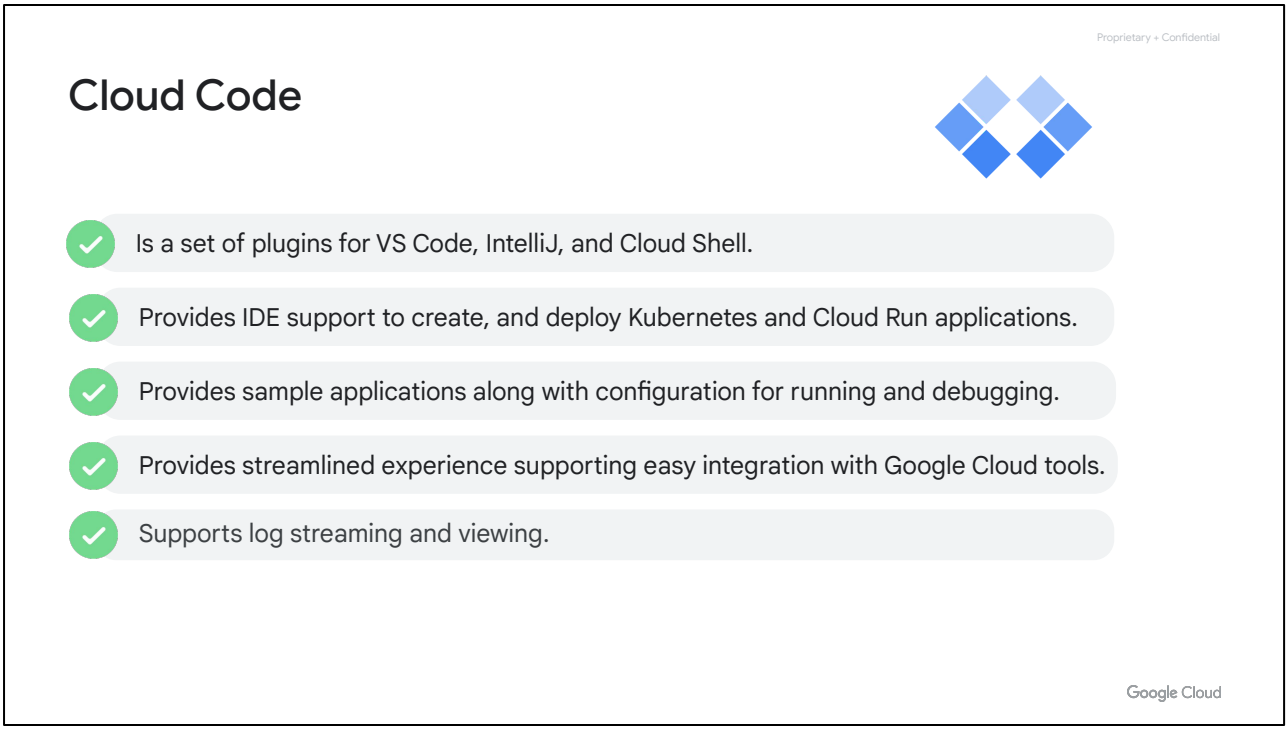
Data written to the file system does not persist when the container instance is stopped. You can use the in-memory file system as a cache to store configuration or disposable per-request data.

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.

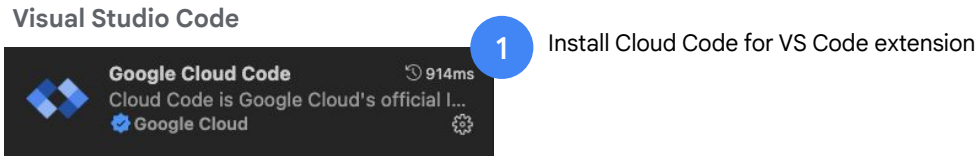


Cloud Code is a set of plugins for popular IDEs that make it easier to create, deploy, and integrate your applications with Google Cloud.

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.

Although 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.

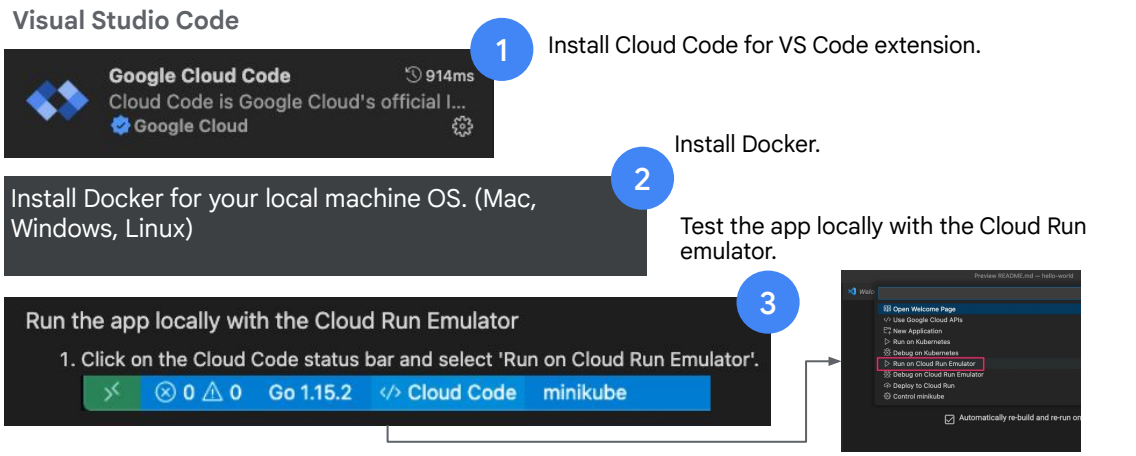
Cloud Code templates



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

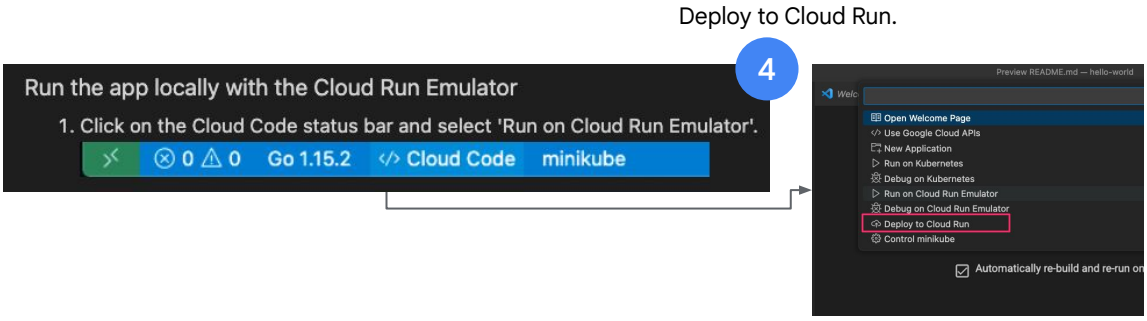


You can develop your app in VS Code and run it locally with the Cloud Run Emulator in VS Code. To build your container image locally with the emulator, install a builder such as Docker on your local machine.

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.

When it's 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.

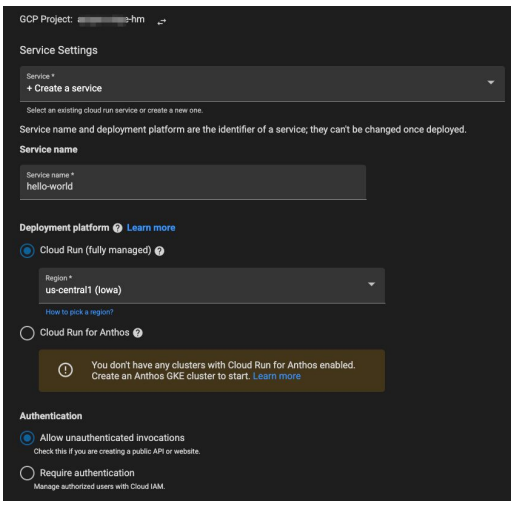
Cloud Code for VS Code - Deploy to Cloud Run



After testing 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 you're 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.

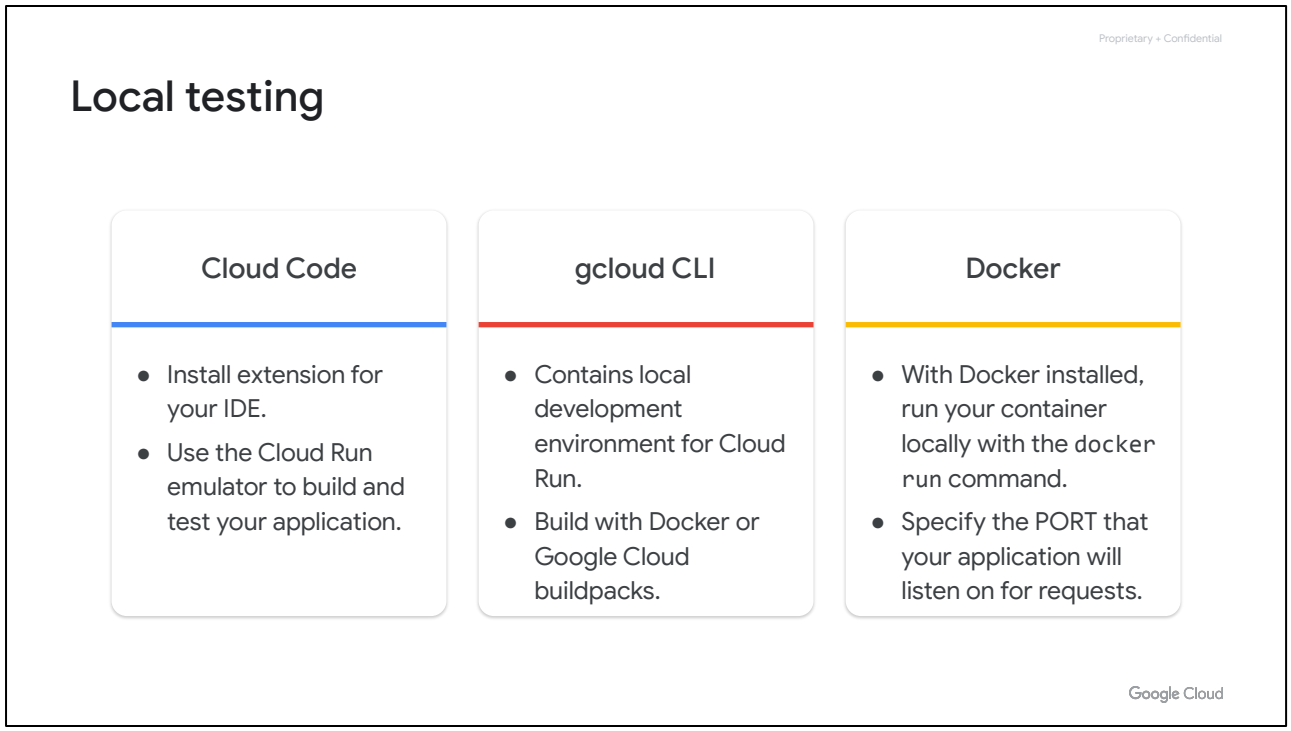
You can also provide additional configuration for your service for authentication, container image URL, and service account.

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 that runs on Cloud Run.

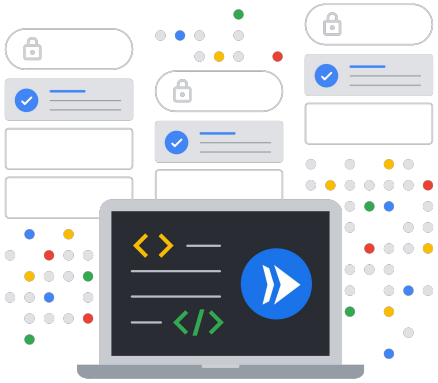
You can configure properties like CPU and memory allocation, specify environment variables, and set Cloud SQL database connections in Cloud Code.

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.

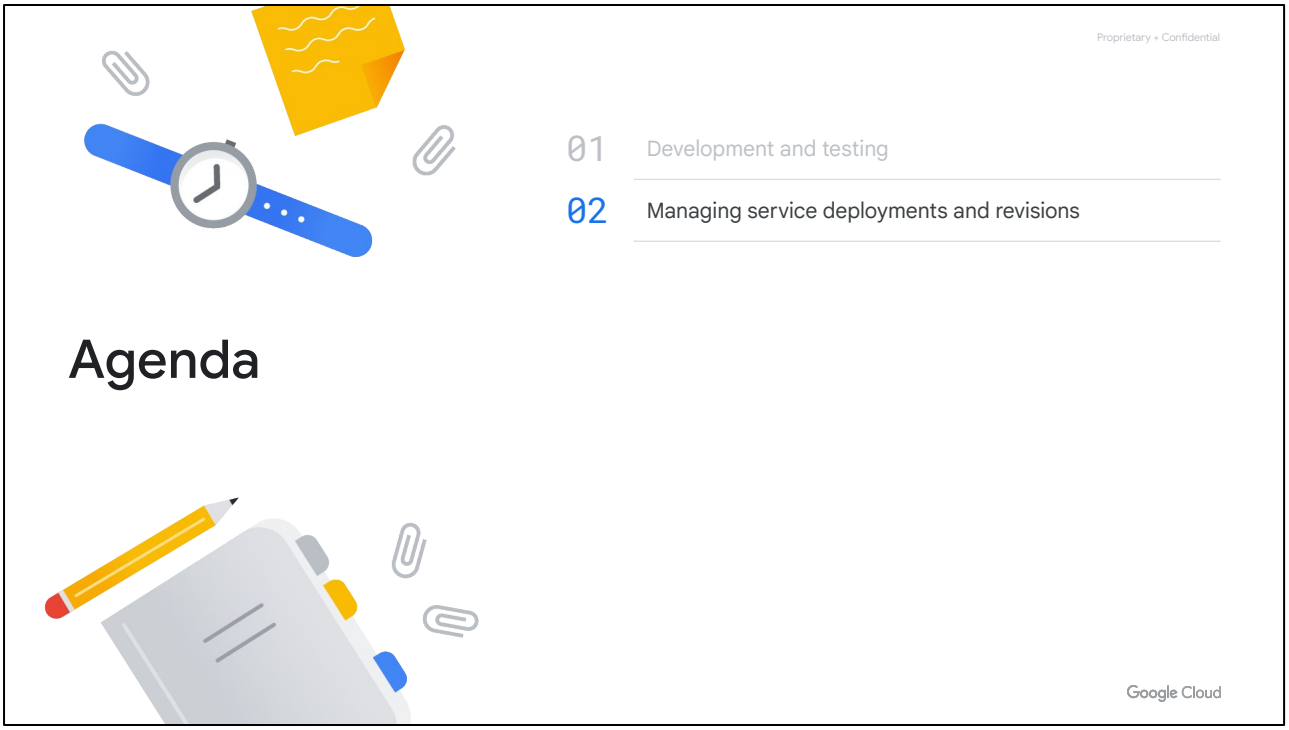
Remember

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.

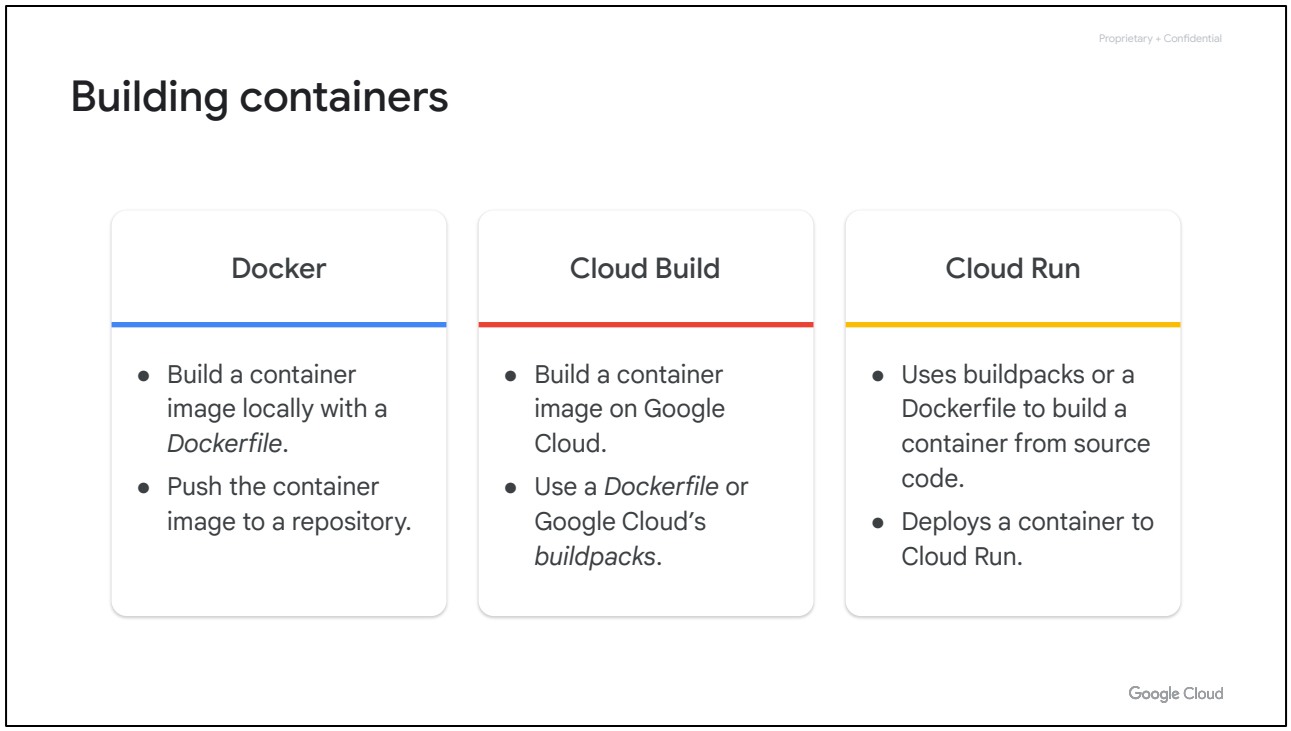


In summary:

- 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.



Let’s now discuss how you can manage your service deployments and revisions on Cloud Run.



Before we discuss Cloud Run deployment, let's talk about how you can build your containerized application.

We discussed how you can build containers with Docker and other CI/CD tools such as Cloud Build in the course on *Deploying Containerized Applications on Google Cloud.*

Here's a brief review:

To build your application into a container image with Docker or with Cloud Build, you can use a Dockerfile. To build a container locally, you can install Docker and use the `docker build` command to build your container image. To push the local container image to an image repository, use the `docker push` command.

You can also build your container image on Google Cloud with Cloud Build. To use Cloud Build, run the `gcloud builds submit` command from the gcloud CLI.

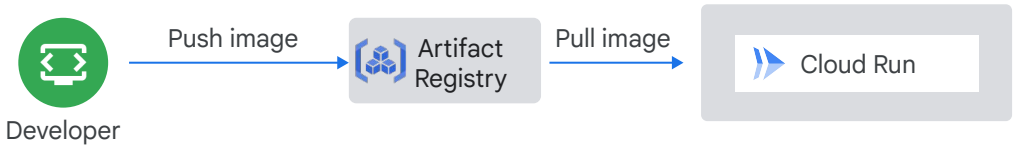
To build your container image from source code without creating a Dockerfile, you can use buildpacks. Google provides a set of CNCF-compatible buildpacks that build source code into container images designed to run on Google Cloud container platforms, including Cloud Run. To build a container image with Google Cloud's buildpacks using Cloud Build, run the `gcloud builds submit` command with the `pack` flag.

Buildpacks are built into Cloud Run to enable a source-based deployment workflow. The gcloud run deploy command with the source flag builds your application source

code with a Dockerfile (if one is present), or with Google Cloud's buildpacks. The resulting container image is also uploaded to an image repository and deployed to Cloud Run.

You can also build your container image locally with Google Cloud's buildpacks by using the pack command.

Deploying containers to Cloud Run



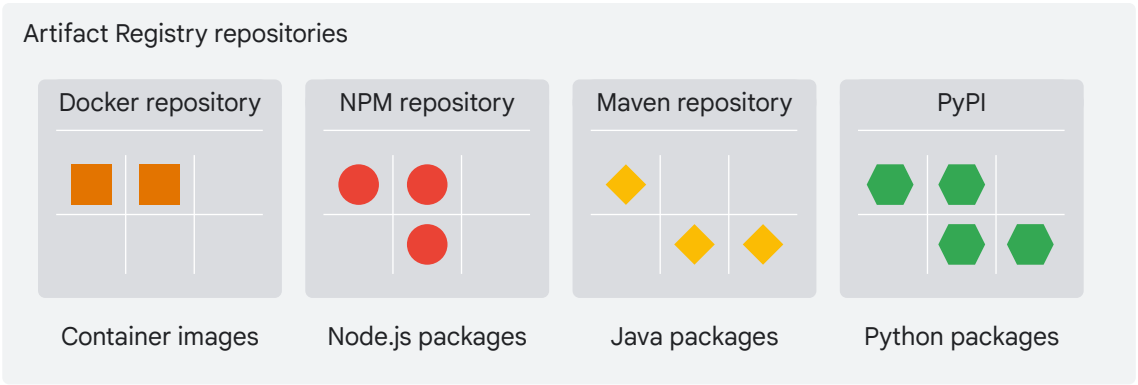
Before a container is deployed to Cloud Run, the container image must be stored in a repository that Cloud Run can access.

You can use container images that are stored in Artifact Registry or Docker Hub. Google recommends the use of Artifact Registry.

You can use container images that are stored in the same project as the one you are creating the job or service in, or from other Google Cloud projects, if the correct IAM permissions are set. To view more details on how to set these permissions, view the documentation.

If you usually host your container images somewhere else such as an unsupported public or private container registry, remember that with Cloud Run you'll need to push them to Artifact Registry first. You can achieve this with the `docker push` command.

Artifact Registry



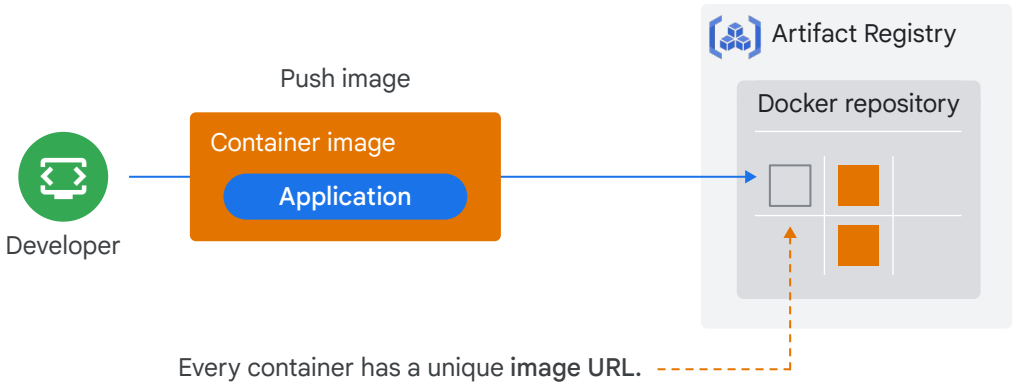
Artifact Registry is a universal package manager service in Google Cloud that is used to store and manage software artifacts in private repositories, including container images, and software packages.

It's the recommended container registry for Google Cloud.

Artifact Registry integrates with Cloud Build to store the packages and container images from your builds.

To host your container images, you create a "Docker repository" that Cloud Run can pull container images from.

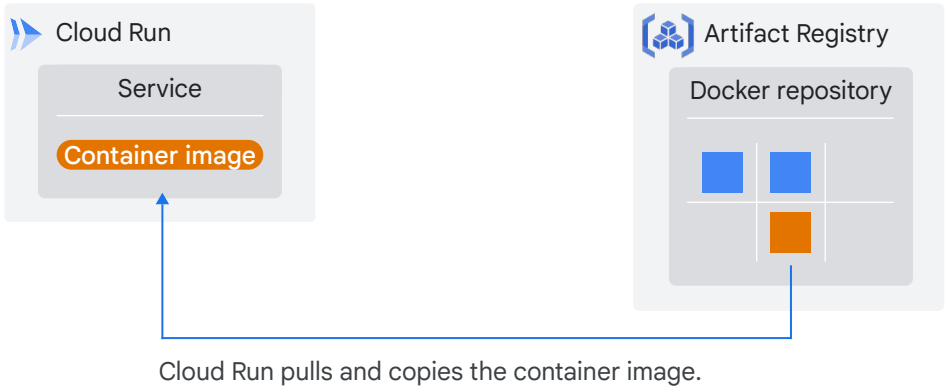
Pushing container images to Artifact Registry



When you're ready to deploy your container image to Cloud Run, you begin by "pushing" (that's a term used for uploading) the image to a Docker repository in Artifact Registry.

Your container image will have a unique URL in the repository, for example, *us-central1-docker.pkg.dev/${PROJECT_ID}/my-repo/my-image,* which you can use when you deploy the image to Cloud Run.

Pulling container images from Artifact Registry

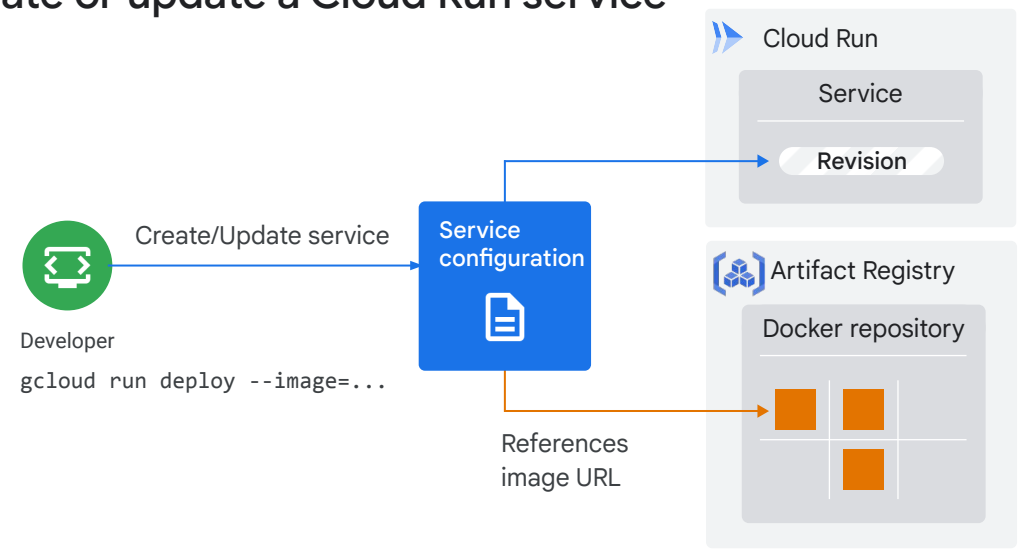


After your container image is pushed into the Docker repository, you can deploy it to a service on Cloud Run.
This means you hand the container image *URL* to Cloud Run, which then pulls the image from Artifact Registry.

To ensure that containers on Cloud Run start reliably and quickly, Cloud Run copies and stores the container image locally. The internal container storage is fast, which ensures that your image size does not impact your container startup time. Large images load as quickly as small ones.

Because Cloud Run copies the image, it won't be an issue if you accidentally delete a deployed container image from Artifact Registry.
The copy ensures that your Cloud Run service will continue to work.

Create or update a Cloud Run service



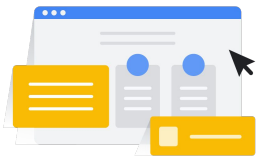
To deploy a container image to Cloud Run, use the Google Cloud console or gcloud CLI to create or update an existing service, and provide the container image URL.

When you deploy a container image for the first time, Cloud Run creates a *service* and its first *revision*. There is only one container image per service.

To be able to deploy, you must have one of Owner, Editor, or both the Cloud Run Admin and Service Account user roles. You can also have a custom role that includes the necessary permissions.

Deploying a new service revision

1. Modify your code.
2. Build and package a container image.
3. Push the image to Artifact Registry.
4. Deploy to the service.
5. Wait for Cloud Run to deploy your changes.



Each revision of a Cloud Run service is immutable. To update your application on Cloud Run, you generally follow these steps:

1. Modify your application source code.
2. Build and package your application into a container image.
3. Push the container image to Artifact Registry.
4. Redeploy the container image to the Cloud Run service.

When you re-deploy your container image to an existing service, a new revision is automatically created.

You can deploy a new revision using the Google Cloud console, the gcloud CLI, a YAML configuration file, or with Terraform.

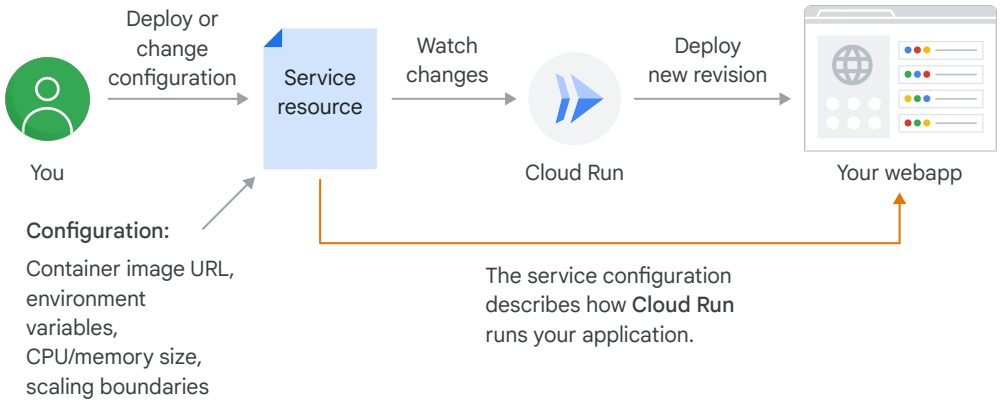
Service configuration

1. Container image URL
2. Container entrypoint and arguments
3. Secrets and Environment variables
4. Request timeout
5. Concurrency
6. CPU/memory limits
7. Scaling boundaries
8. Google Cloud configuration (service account, connectors)

Changing any configuration settings of your Cloud Run service results in the creation of a new revision, even if there is no change to the container image itself.

Subsequent service revisions will also automatically get these configuration settings unless you make explicit updates to change them.

Updating a service



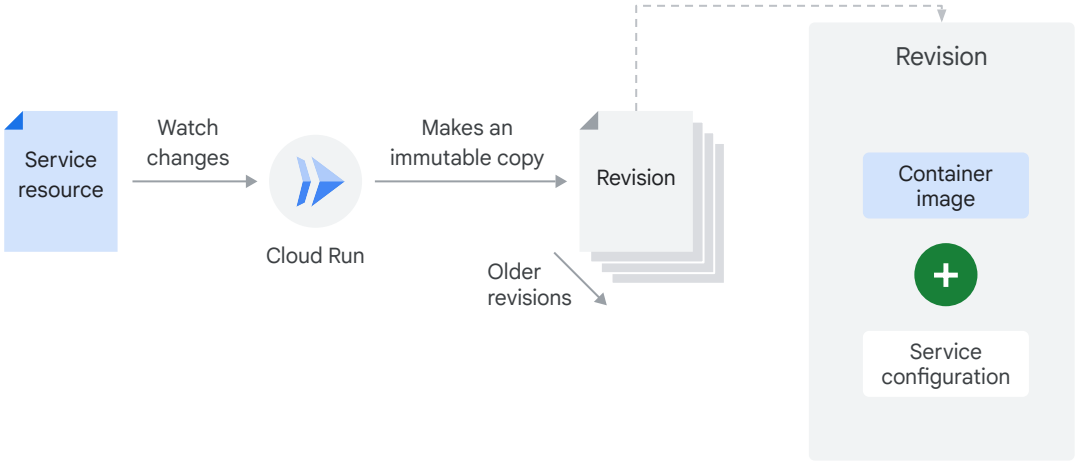
The service resource includes configuration that describes how Cloud Run runs your application.

After you make changes to your application or its service configuration, and deploy it, Cloud Run creates a new revision of your service.

You can control sending all traffic to the new revision immediately when the new revision is healthy.

You can also perform a gradual rollout by controlling the percentage of requests sent to the new revision.

A service revision is immutable



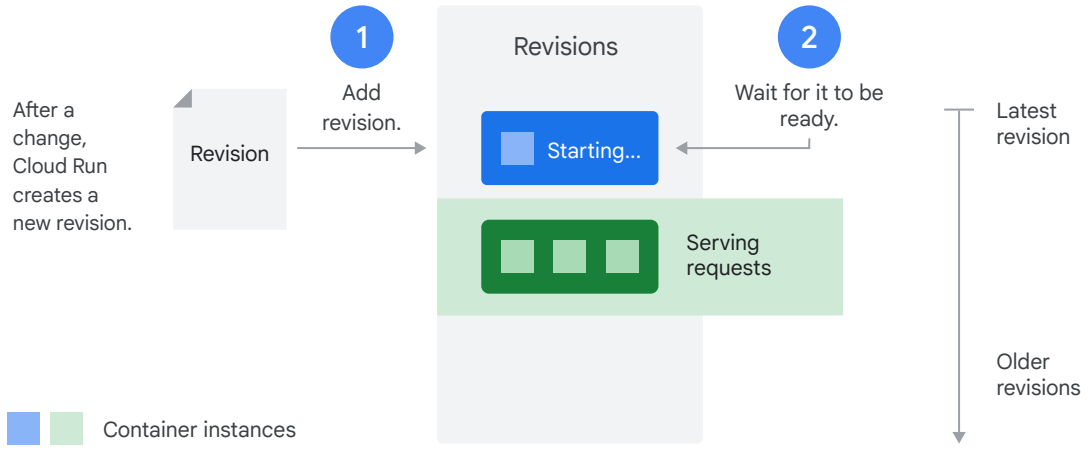
Cloud Run deploys your application after every change that you make to the service resource.

At the same time, it also makes an immutable copy of the service resource, called a revision. “Immutable” means that you can’t make changes to a revision.

You can only add new revisions to make further updates.

A revision is an immutable copy of your container image and service configuration.

Serving traffic

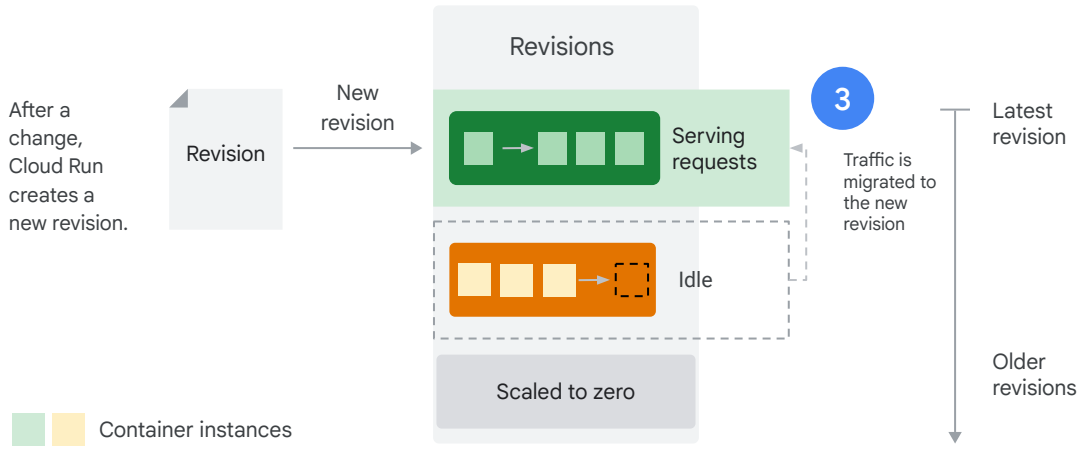


After a new service revision is created, Cloud Run will first scale up the new revision to match the capacity of the current revision.

Cloud Run waits for the container instances in that revision to finish starting up.

While this is happening, the container instances in the current revision still serves any request traffic to the service.

Serving traffic



After the container instances in the new service revision are ready, Cloud Run routes traffic to the new revision.

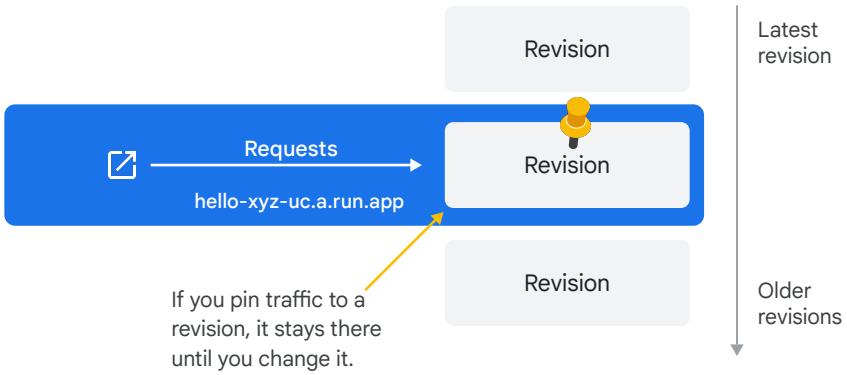
Both revisions (the “previous” revision and the “new” revision) autoscale independently. The containers in the previously active revision will eventually stop serving requests and become idle.

The new revision might have to add more containers to handle demand, and the previous revision will eventually remove the idle containers and scale to zero.

As mentioned previously, you can gradually migrate traffic to the new revision by setting the percentage of requests sent to the new revision.

To perform a gradual rollout of changes to an application, a new service revision can be configured to receive no traffic initially, when deployed with the `–no-traffic` option. To gradually increase the amount of traffic received by the new service revision, you can then update the service to specify an incremental percentage value.

Pinning traffic



You can also pin request traffic to a specific service revision rather than the latest revision, decoupling the deployment of a new revision from the migration of traffic.

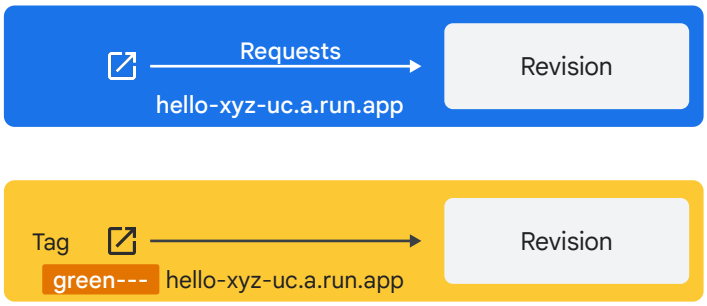
This also means that if you add a new revision, Cloud Run will not automatically send traffic to that new revision.

Pinning to a revision is useful if you want to roll back to previous revision, or if you first want to test your new revision before migrating all request traffic to it.

This is achieved by setting the percentage of request traffic to the revision to 100.

You can set this in the Google Cloud console, with the gcloud CLI, a YAML configuration file, or with Terraform.

Tagging service revisions



Test URL: https://green---hello-xyz-uc.a.run.app

When you deploy a service, you can assign a tag to the new revision that lets you access the revision at a specific URL without serving traffic.

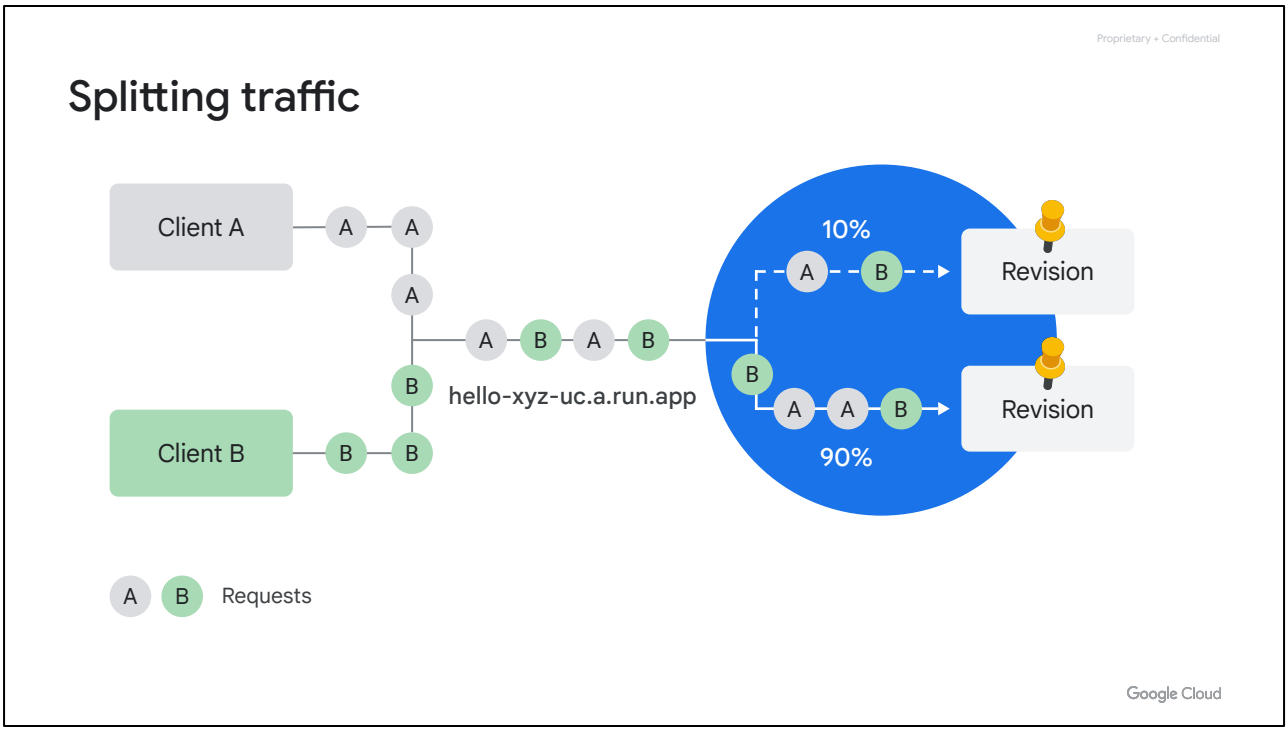
You can then use that tag to gradually migrate traffic to the tagged revision, or to rollback a tagged revision.

A common use case for this feature is to use it for testing and vetting of a new service revision before it serves any traffic.

A tagged revision has its own unique URL, which is the URL of the Cloud Run service and the name of the tag added as a prefix. For example, if you used the tag name green on the service hello, you would test the tagged revision at the URL: `https://green---hello-xyz-uc.a.run.app`.

For example, you might want to tag a revision with the ID of the commit in version control that was used to create the revision.

After confirming that the new revision works properly, you can start migrating traffic to it by using the Google Cloud console, the gcloud command line, Terraform, or a YAML file.



Cloud Run lets you specify which service revisions should receive traffic and traffic percentages that are received by a revision.

This feature lets you roll back to a previous revision, gradually roll out a revision, and split traffic between multiple revisions.

To split request traffic between multiple service revisions, you specify a percentage value. This value indicates the percentage of requests that are routed to each revision. You can configure this percentage in the Google Cloud console, with the gcloud CLI, a YAML configuration file, or with Terraform.

Traffic routing adjustments are not instantaneous. When you change traffic splitting configuration for revisions, all requests currently being processed continue to completion. Requests that are in-flight will not be dropped and may be directed to either a new revision or a previous revision during the transition period.

By default, requests from the same client can be handled by different container instances in a service revision. To change this behavior, you can enable session affinity for a revision in Cloud Run. Session affinity is a best effort made by Cloud Run to route requests from the same client to the same revision container instance.

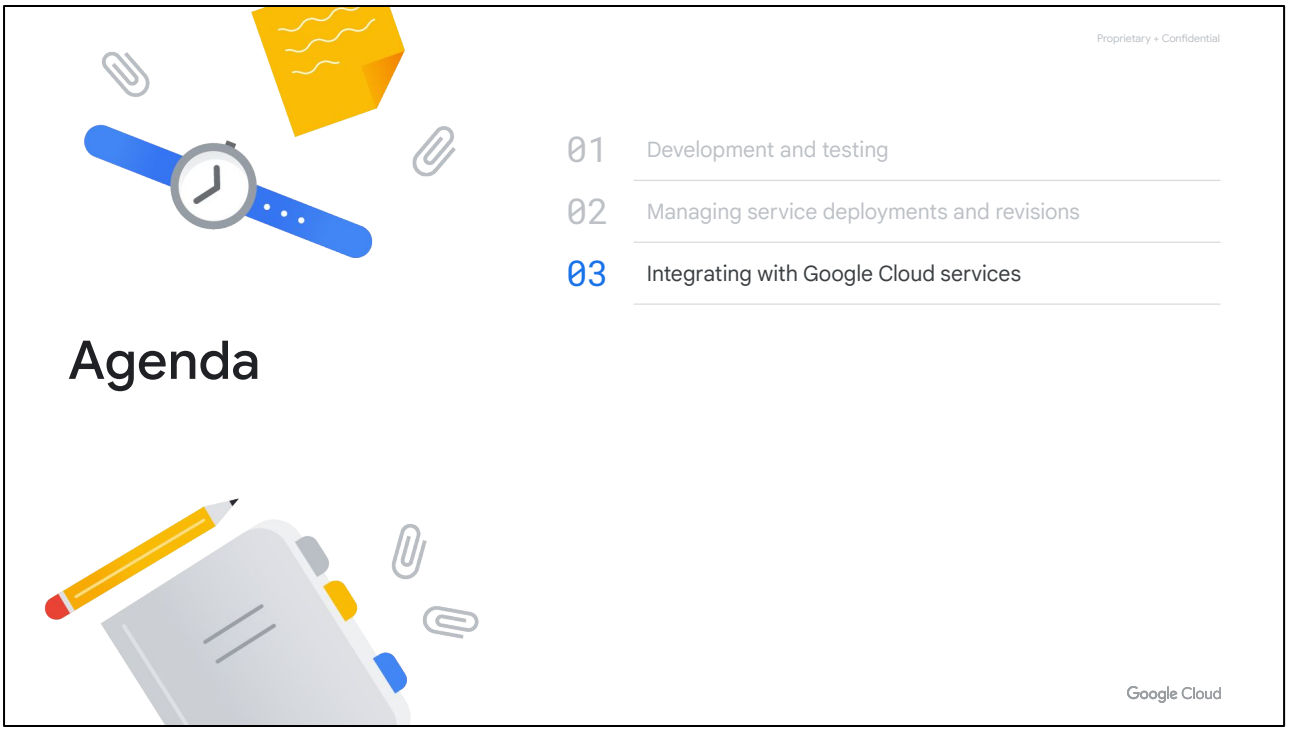
If you are splitting traffic between multiple revisions with session affinity enabled, see session affinity and traffic splitting for details on the effect of session affinity on traffic splitting.

Remember

1. To deploy your application on Cloud Run, store your container image in Artifact Registry.
2. Cloud Run copies and stores the container image locally.
3. A service revision on Cloud Run is immutable.
4. Changing any service configuration settings results in the creation of a new revision.
5. You can split traffic between service revisions based on percentage of requests.

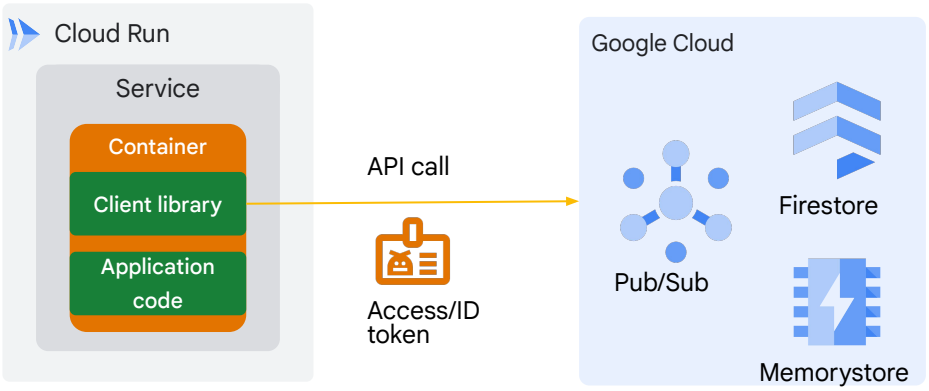
In summary:

- When deploying to Cloud Run, you can use container images that are stored in Artifact Registry, Container Registry, or Docker Hub. Google recommends the use of Artifact Registry.
- To ensure that containers on Cloud Run start reliably and quickly, Cloud Run copies and stores the container image locally.
- When you deploy a container image for the first time, Cloud Run creates a *service* and its first *revision*. There is only one container image per service.
- Changing any configuration settings of your Cloud Run service results in the creation of a new revision.
- Cloud Run lets you split traffic between service revisions based on the percentage of requests received by the service.



In this topic, we briefly discuss how you can connect and integrate your Cloud Run service with other services in Google Cloud.

Connect to Google Cloud services



You can use Cloud Run to connect to supported Google Cloud services from your application by using the client libraries that those services provide. The client libraries use the built-in service account to transparently authenticate with Google Cloud services. This service account has the *Project > Editor* role, which means it's able to call all Google Cloud APIs and have read and write access on all resources in your Google Cloud project.

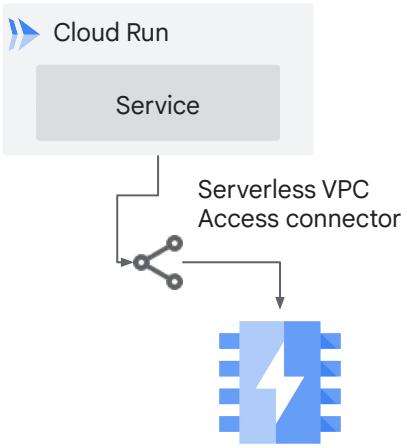
As mentioned previously, you should use a per-service identity to restrict the APIs and resources that your Cloud Run service can access. You can do this by assigning a service account with a minimal set of permissions to your Cloud Run service. For example, if your Cloud Run service is only reading data from Firestore, you should assign it a service account that only has the *Firestore User* IAM role.

For a full list of Google Cloud services with which you can integrate your Cloud Run service, refer to the documentation.

Connect Cloud Run to Memorystore

To access Memorystore from Cloud Run:

1. Determine your Redis instance's authorized VPC network.
2. Create a Serverless VPC Access connector.
3. Attach the connector to the Redis instance's authorized VPC network.



Let's review how you can connect to a Memorystore for Redis instance from your Cloud Run service.

Memorystore is a Google Cloud service that provides a highly available, scalable, and secure in-memory cache solution for Redis and Memcached.

To connect to a Memorystore for Redis instance from your Cloud Run service, you use Serverless VPC Access.

To connect to your Redis instance, your Cloud Run service needs access to the Redis instance's authorized VPC network. To enable this access, you need a Serverless VPC Access connector.

After you determine your Redis instance's authorized VPC network, create a Serverless VPC Access connector in the same region as your Cloud Run service. Then, attach the connector to the Redis instance's authorized VPC network.

Read the documentation to learn about creating a Serverless VPC Access connector.

Connect Cloud Run to Memorystore

Deploy the service:

- Specify the name of the connector.
- Set environment variables for the service application code to connect to the Redis instance host and port.

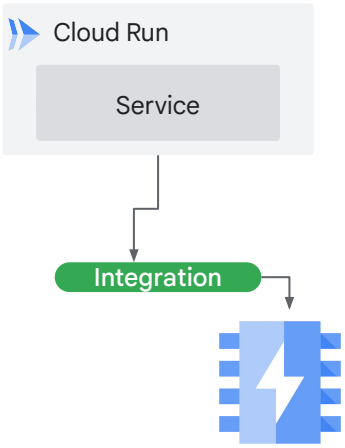
```
gcloud run deploy \
--image my-container-image \
--platform managed \
--region us-central1 \
--vpc-connector my-connector \
--set-env-vars \
REDISHOST=[REDIS_IP],REDISPORT=[REDIS_PO
RT]
```


Deploy the service to Cloud Run and specify the name of the connector, and environment variables for the Redis instance's host IP address and port.

Your application code can then use these environment variables to instantiate a client that connects to the Redis instance.

Cloud Run integrations

1. Create an integration to connect a Cloud Run service to a Memorystore for Redis cache.
2. A fully configured Redis cache is automatically created with a configurable memory size.
3. A new Cloud Run service revision is created for the service.
4. Networking and environment variables are also configured for the service to access the Redis cache.



The integrations feature provides a simple Google Cloud console UI and gcloud CLI command that create and configure the resources and services needed for the specific integration, eliminating the complicated steps that would otherwise be required.

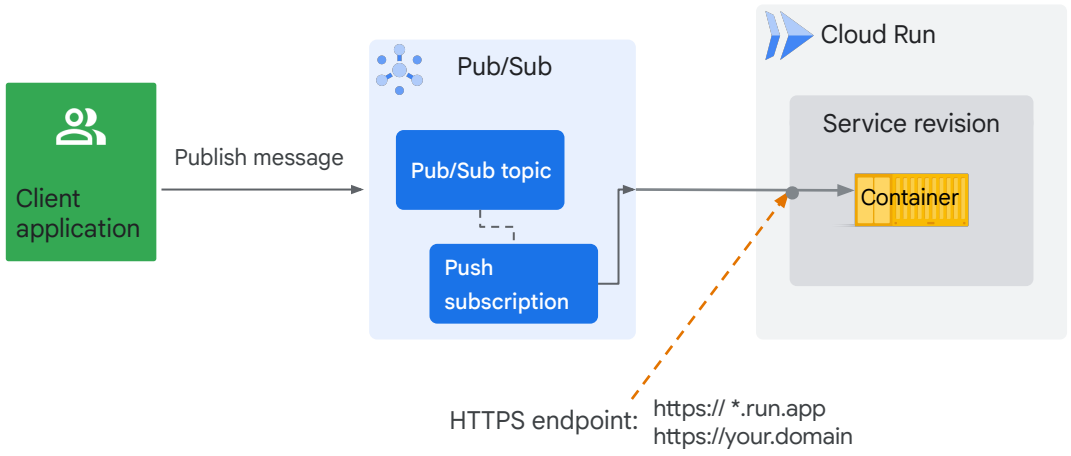
Integrations currently let you:

- Map custom domains to Cloud Run services.
- Connect a Cloud Run service to a Memorystore for Redis instance.

More integrations will be supported in the future.

For more information, refer to the documentation on connecting to a Memorystore instance from Cloud Run using integrations.

Trigger from Pub/Sub

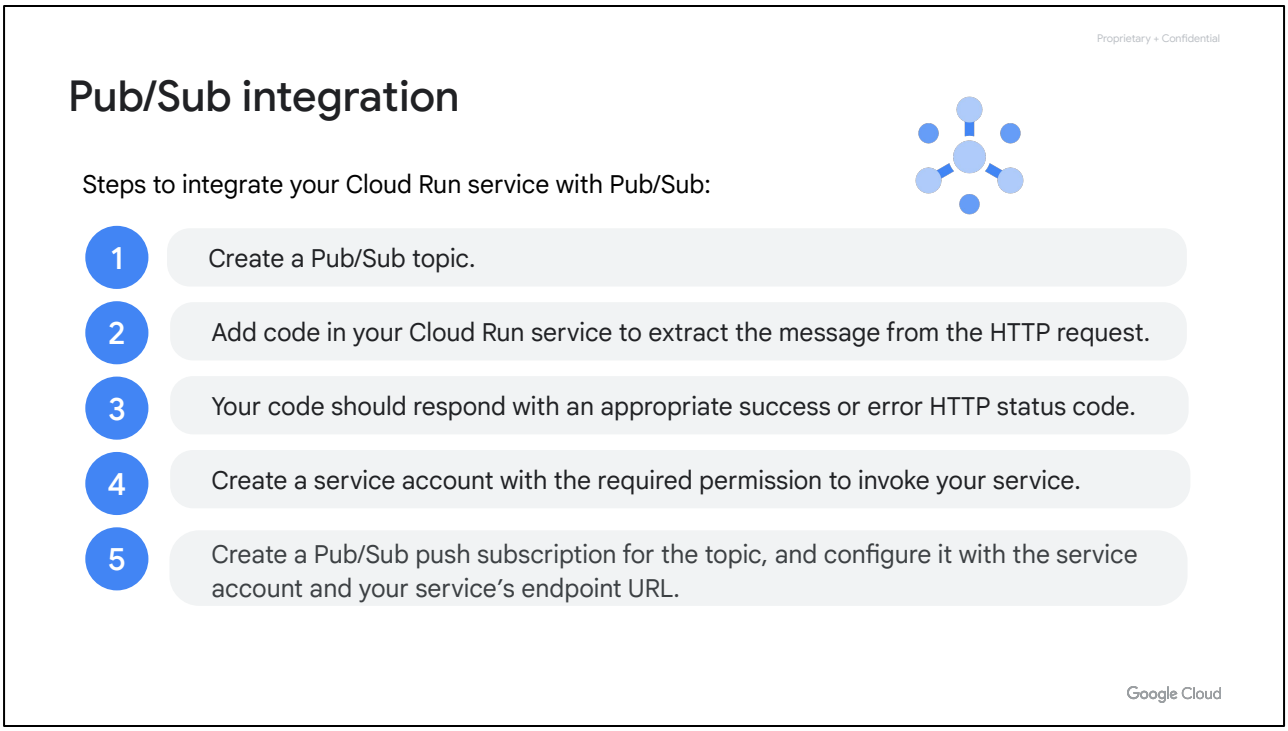


We discussed how you can publish a message to a Pub/Sub topic from a Cloud Run service in the previous module.

Let's talk about how you can trigger a Cloud Run service by a message from a topic in Pub/Sub.

You can use Pub/Sub to *push* messages to the endpoint of your Cloud Run service, where the messages are subsequently delivered to containers as HTTP requests. The endpoint can be protected using IAM and doesn't need to be public.

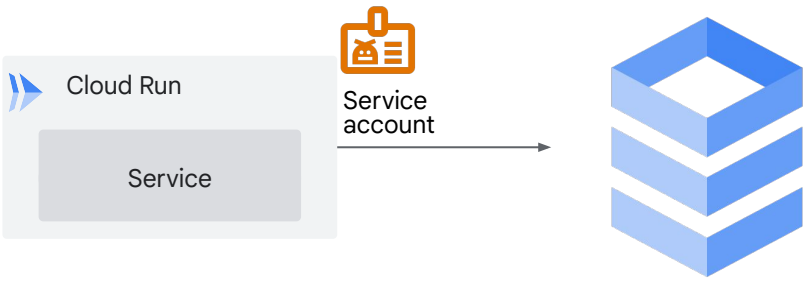
Your Cloud Run service must acknowledge the Pub/Sub message by returning a response within 600 seconds (maximum acknowledgement deadline), otherwise Pub/Sub redelivers the message, which causes that your Cloud Run service is triggered again.



To integrate your service with Pub/Sub:

- Create a Pub/Sub topic.
- Add code in your Cloud Run service to respond to the Pub/Sub messages sent to the topic that you created.
    a. Your service must extract the message from the request and return an expected HTTP status response code.
    b. Success codes, such as HTTP 200 or 204, acknowledge complete processing of the Pub/Sub message.
    c. Error codes, such as HTTP 400 or 500, indicate that the message will be retried.
- Create a service account with the required permission (role: Cloud Run Invoker) to invoke your Cloud Run service.
- Create a Pub/Sub push subscription for the topic that you created, and associate it with the service account. Provide your service's URL as the endpoint URL. This subscription will send any message that is published to the topic to your service.

Connect Cloud Run to Cloud SQL




```
gcloud run services update my-service
--add-cloudsql-instances=my-sql-instance-connection
```

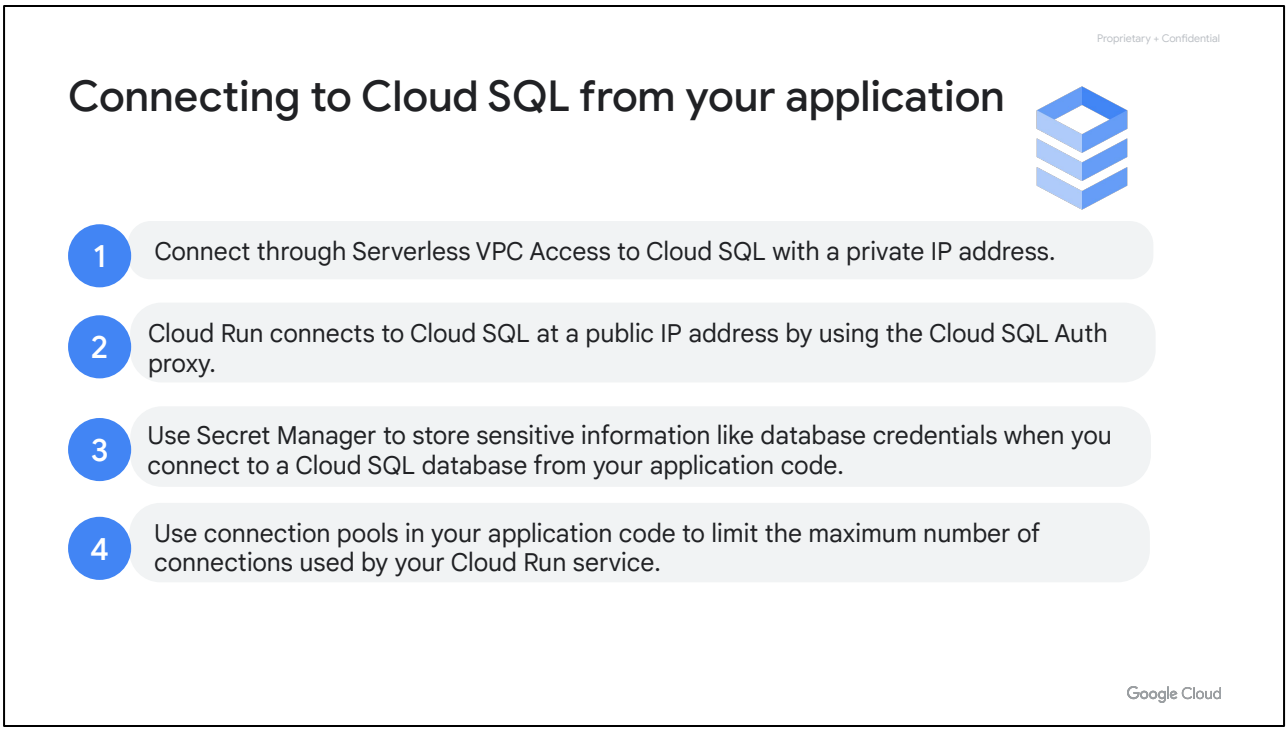

Let's now review how you can connect to a Cloud SQL instance from a service in Cloud Run.

Cloud SQL is a fully managed database service for MySQL, PostgreSQL, and SQL Server that lets you set up, manage, and administer relational databases in Google Cloud.

By default, Cloud SQL assigns a public IP address when you create a new instance. To connect to the instance from your Cloud Run service:

- The service account used by the service must have the appropriate Cloud SQL roles and permissions (one of Cloud SQL Client, Cloud SQL Admin).
- You must deploy or update your Cloud Run service with the instance connection name of your Cloud SQL instance. You can do this in the Google Cloud console, with the gcloud CLI, or with Terraform.

You also have the option to assign a private IP address to your Cloud SQL instance. With a private IP address, you can route all egress traffic from your Cloud Run service to the Cloud SQL instance using a Serverless VPC Access connector. To do this, you configure your Cloud Run service to use the connector when you deploy or update the service.



As mentioned previously, when connecting to a Cloud SQL instance with a private IP address, your application will connect directly through Serverless VPC Access.

For the public IP path, Cloud Run provides encryption and connects using the Cloud SQL Auth proxy using network sockets or a Cloud SQL connector.

Cloud SQL connectors are language specific libraries that provide encryption and IAM-based authorization when connecting to a Cloud SQL instance.

Your application code will need access to the Cloud SQL instance connection name, database name, and credentials. It's recommended to use Secret Manager to store the sensitive database credentials, and pass this information to your application as environment variables or mounted as a volume in Cloud Run.

You should also use a client library that supports connection pools that automatically reconnect broken client connections to the Cloud SQL database.

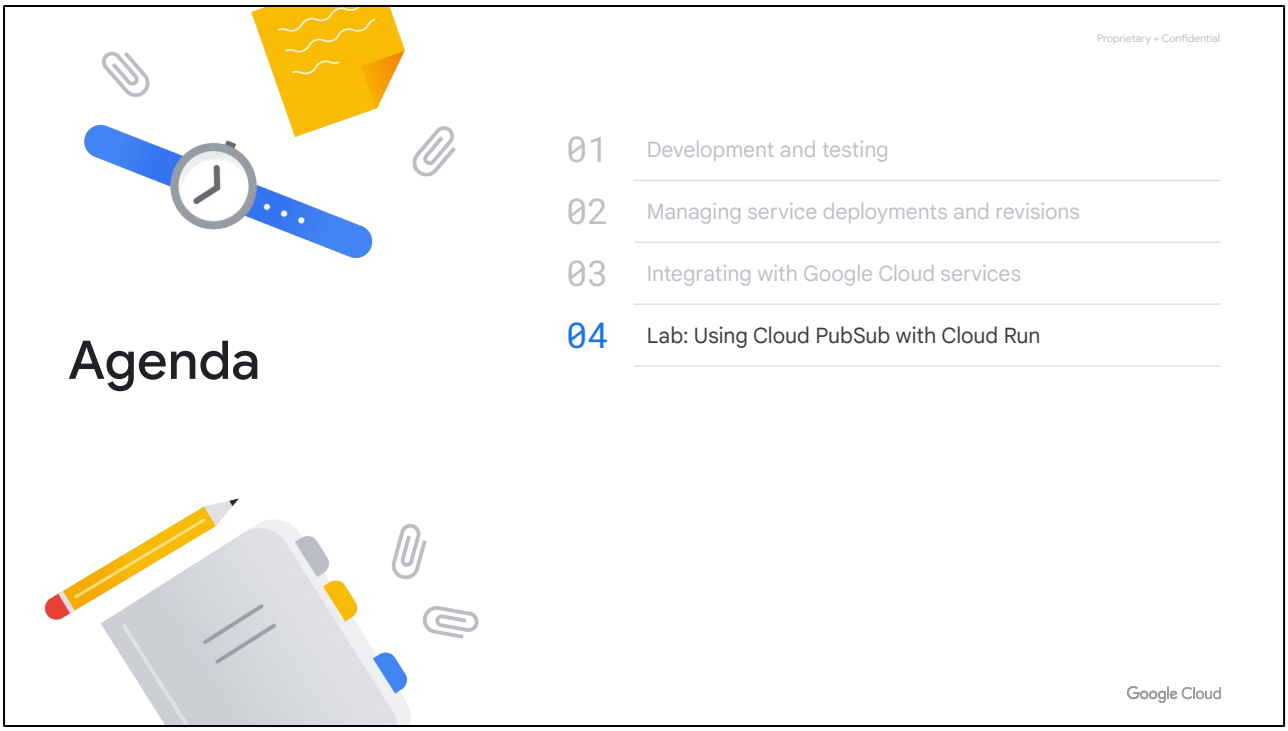
By using a connection pool, you can also limit the maximum number of connections used by the service. Cloud Run services are limited to 100 connections per service to a Cloud SQL database. There are also quotas and limits imposed by Cloud SQL. For more information, refer to the documentation.

Remember

1. Connect to supported Google Cloud services from your Cloud Run application using client libraries.
2. Use a per-service identity to restrict the APIs and resources that your Cloud Run service can access.
3. Connect to a Memorystore for Redis instance from your Cloud Run service with Serverless VPC Access.
4. Pub/Sub messages are delivered to your container in Cloud Run with HTTP requests.
5. Use Secret Manager to store sensitive database credentials, and pass them to your service as environment variables or mounted as a volume in Cloud Run.

In summary:

- Use Cloud Run to connect to supported Google Cloud services from your application by using the client libraries that those services provide.
- Use a per-service identity (service account) to restrict the APIs and resources that your Cloud Run service can access.
- Connect to a Memorystore for Redis instance from your Cloud Run service with Serverless VPC Access.
- Use Pub/Sub to *push* messages to the endpoint of your Cloud Run service, where the messages are then delivered to containers as HTTP requests.
- Use Secret Manager to store sensitive database credentials, and pass this information to your service as environment variables or mounted as a volume in Cloud Run.



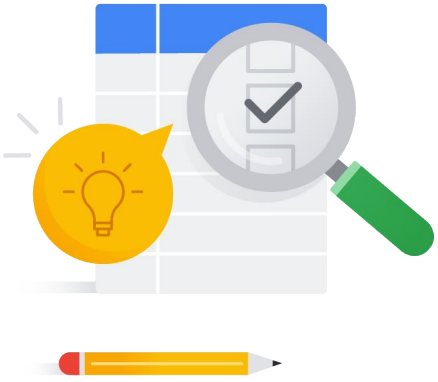
You now complete a lab to use Pub/Sub with Cloud Run.

Lab

45 min | Individual

Using Cloud PubSub with Cloud Run

Learn how to use Pub/Sub with Cloud Run.



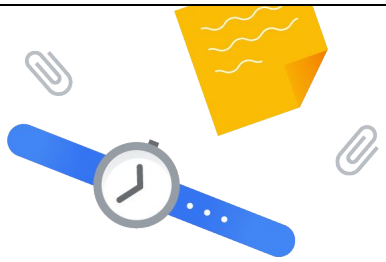
In this lab, you learn how to use Pub/Sub with Cloud Run.

Lab instructions

45 min | Individual | Tasks

- Deploy services to Cloud Run that produce and consume Pub/Sub messages.
- Configure a topic, subscription, and service account for use with Pub/Sub.
- Test the application.

1. Deploy producer and consumer services to Cloud Run.
2. Configure Pub/Sub.
3. Test the application.



Agenda

01 Development and testing

02 Managing service deployments and revisions

03 Integrating with Google Cloud services

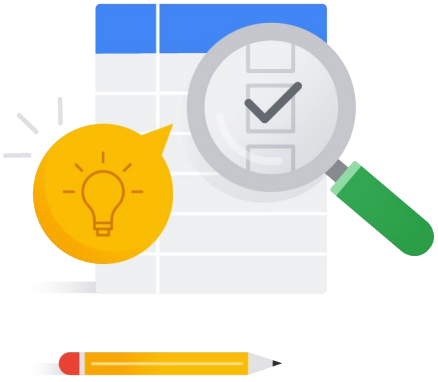
04 Lab: Using Cloud PubSub with Cloud Run

05 Quiz

Quiz

5 min | Group

Application development, testing, and integration



Let’s do a short quiz on the topics that were discussed in this module.

Quiz | Question 1

Question

What are some characteristics of applications that are a good fit for Cloud Run? Select three.

A. The application is containerized.

B. The application depends on a local persistent file system.

C. When run as a service, the application responds to a request within a specified time.

D. When run as a service, the application listens for HTTP requests on a specified port.

Quiz | Question 2

Question

What are some products or services that you can use to build containers? Select three.

A. Cloud Run
B. Cloud Build
C. Docker
D. Artifact Registry

Quiz | Question 3

Question

Which of these statements regarding service revisions in Cloud Run are correct? Select two.

A. A revision is created when you deploy a container image to a service in Cloud Run.

B. A revision is created when you update the configuration of a service in Cloud Run.

C. You cannot control the amount of request traffic received by a service revision.

D. In Cloud Run, only the newest or latest service revision can receive and process requests to the service.

Quiz | Question 4

Question

To integrate a Cloud Run service with other Google Cloud APIs and resources, what are some steps you should take? Select three.

A. Use the default runtime service account to access Google Cloud APIs and resources from a Cloud Run service.

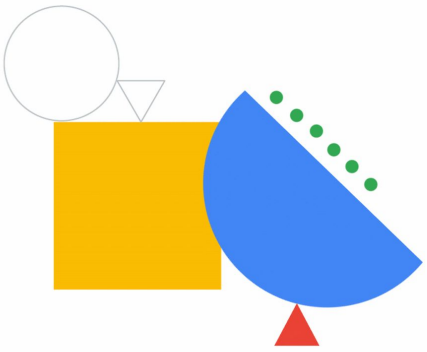
B. Connect to Google Cloud services from your Cloud Run service using client libraries.

C. To connect to internal cloud resources from your Cloud Run service, use Serverless VPC Access.

D. Use Secret Manager to store credentials required by downstream database services.

Google Cloud

Review: Application development, testing, and integration



Let’s review the topics that were discussed in this module.

In this module, you learned about:

01 Developing and testing your Cloud Run application

02 Managing service deployments and revisions

03 Integrating with Google Cloud services

In this module, we discussed what makes an application a good fit for Cloud Run.

You learned that you can use Cloud Run to run a containerized application, which can be written in any programming language.

We discussed using Cloud Code with popular IDEs to easily create, deploy, and integrate your application with Google Cloud, and how you can test your application with Cloud Code, the gcloud CLI, or with Docker locally.

You learned how Cloud Run services are deployed and how you can split request traffic between multiple service revisions.

Finally we discussed how you can integrate your application on Cloud Run with other Google Cloud services, such as Memorystore, Pub/Sub, and Cloud SQL.