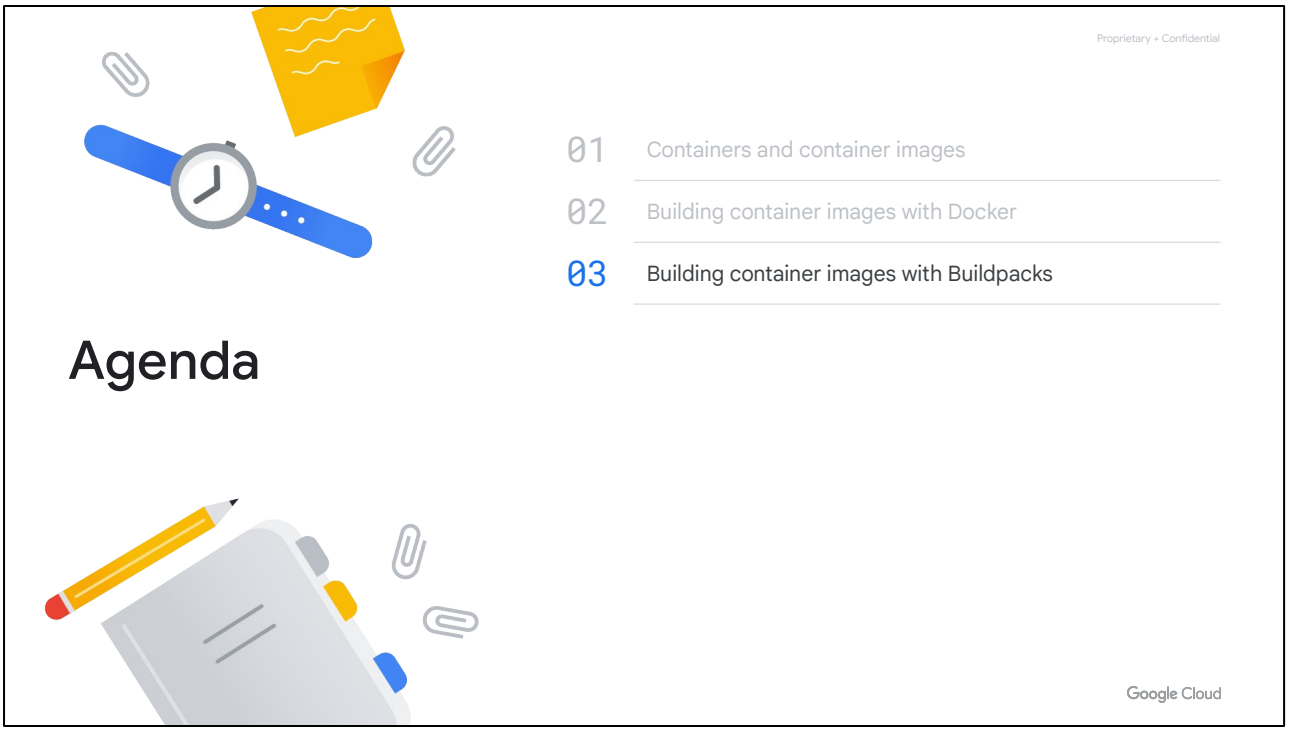
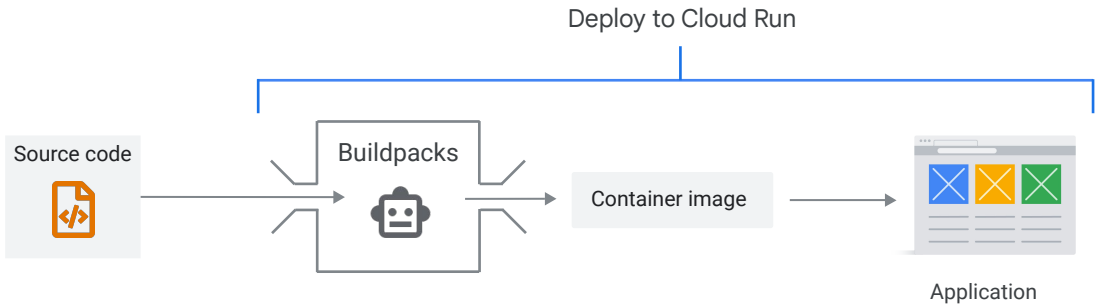


Let’s discuss how you can create container images with Buildpacks.

Buildpacks



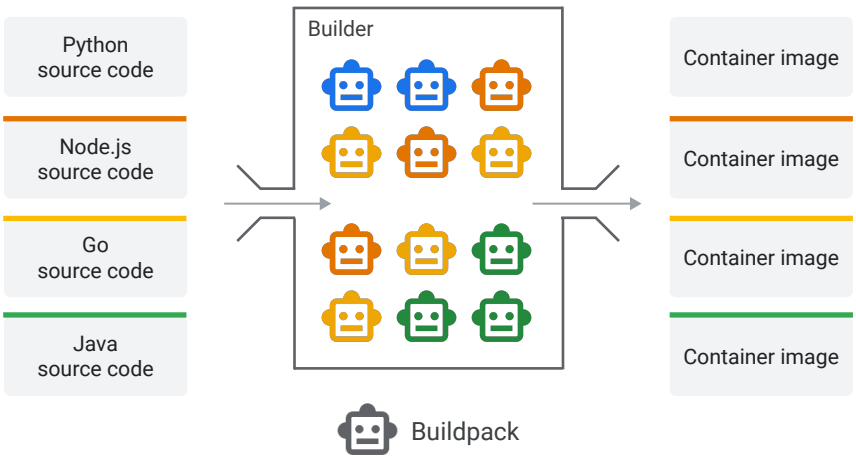
Buildpacks are a way to turn source code into a container image without writing a Dockerfile.

Buildpacks provide developers with a convenient way to work with container images, without thinking about the complexities that come with building them.

You can create your own Buildpacks, or use those provided by multiple vendors.

Buildpacks are built into Cloud Run to enable a source-based deployment workflow.

Builders



Buildpacks are distributed and executed in OCI images called builders. Each builder can have one or more buildpacks.

OCI stands for the **Open Container Initiative**, a Linux Foundation project that was started in 2015 to design open standards for operating-system-level virtualization of Linux containers.

A builder turns your source code into a container image. The buildpacks do the actual work to build and package the container image.

Builders can support source code written in multiple languages. In this example, the builder can build and package a Python, Node.js, Go, and a Java application into a container image.

If a builder starts to process a source directory, it executes two phases of a buildpack:

1. The **detect** phase

   The detect phase runs against your source code to determine if a buildpack is applicable or not. Once a buildpack is detected to be applicable, the builder proceeds to the build phase. If detection fails, the build phase for a specific buildpack is skipped.

   For example, to pass the detect phase:
   a. A Python buildpack may look for a requirements.txt or a setup.py file.
   b. A Node buildpack may look for a package-lock.json file.

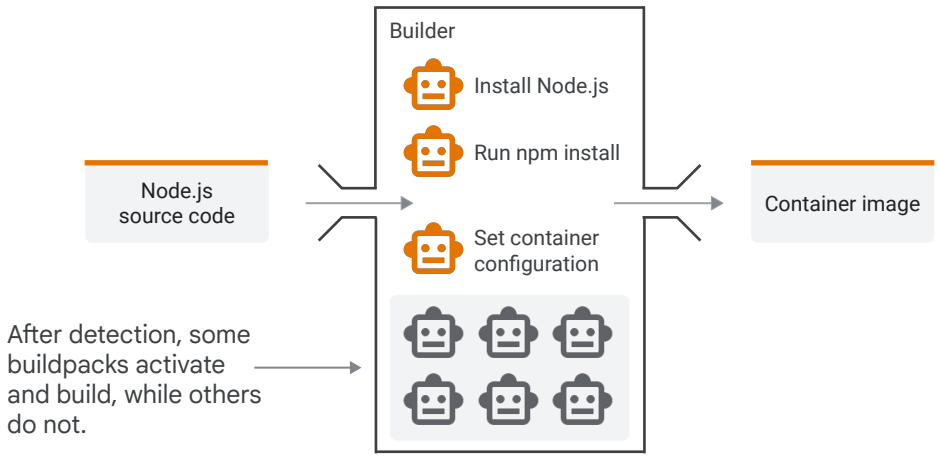
1. The **build** phase:

   The build phase runs against your source code to set up the build-time and run-time environment, download dependencies and compile your source code (if needed), and set appropriate entry point and startup scripts.

   For example:

   a. A Python buildpack may run *pip install -r requirements.txt* if it detected a requirements.txt file.
   b. A Node buildpack may run *npm install* if it detected a package-lock.json file.

Builders



Let's look at an example.

After the builder runs the detect phase, suitable buildpacks activate and perform a build, while others do not.

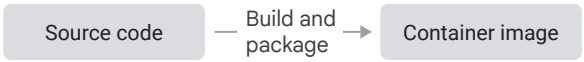
In this example, three buildpacks activate to build a directory with a Node.js project.

1. One buildpack installs the Node.js runtime.
2. Another buildpack runs npm install.
3. The final buildpack configures the resulting image to start the Node.js runtime.

The result is a container image that you can deploy to Cloud Run or run with Docker locally.

Pack

Pack is a command line tool. It needs a builder to turn a source directory into a container image.



Shell

```
$: pack build
   --builder gcr.io/buildpacks/builder:v1
   --path ./source-dir
   sample-app
```


As a developer, it's easy to use Buildpacks.

With the command line tool "pack", you can use a builder to turn source code into a container image.

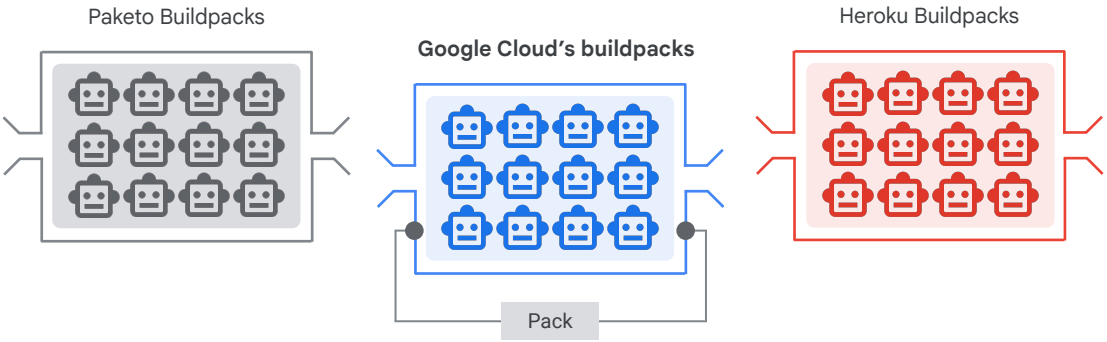
Pack is a tool that is maintained by the Cloud Native Buildpacks project to support the use of buildpacks.

The example shows how to use *pack* to build a source directory using the Google Cloud's buildpacks builder that's built into Cloud Run.

By running this command on your local machine, you can reproduce the container image in a similar manner as Cloud Run does in Google Cloud.

For more information on using Google Cloud's buildpacks, view the documentation.

Choice of builders



The command-line tool pack can work with all of them.

There are many projects that use the buildpacks standard to create their own builders. You can choose to use a builder from any of these projects.

Some examples are:

Paketo Buildpacks: This is a Cloud Foundry Foundation project, which is dedicated to maintaining vendor-neutral governance.

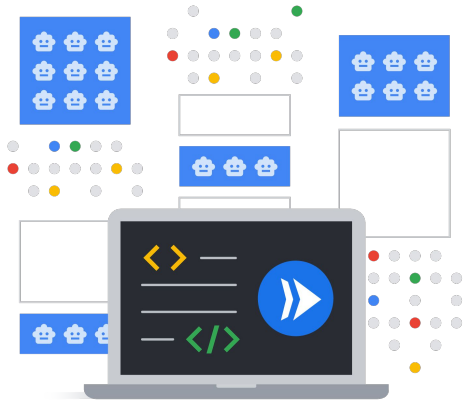
Heroku Buildpacks: Heroku is a well known platform as a service product that makes it easy to build cloud native applications.

Regardless of what buildpack you use to build your container image - Cloud Run can run it.

Google Cloud's buildpacks are built into Cloud Run.

Google Cloud's buildpacks

- The builder is **used internally** by App Engine, Cloud Functions, and Cloud Run.
- The builder supports **Go, Java, Node.js, Python,** and **.NET Core.**
- The project is **open source.**
- Source code deployed to **Cloud Run** is built with Google Cloud's buildpacks.



Google Cloud's buildpacks are used internally by App Engine, Cloud Functions, and Cloud Run.

The Google Cloud's buildpacks builder supports applications written in Go, Java, Node.js, Python, and .NET Core.

You can deploy source code, and container images to Cloud Run. Cloud Run builds source code with Google Cloud's buildpacks.

Google Cloud's buildpacks are optimized for security, speed, and reusability.

github.com/GoogleCloudPlatform/buildpacks

Remember

1. Buildpacks are a way to create a container image from source code without writing a Dockerfile.
2. Buildpacks are distributed and executed in OCI images called builders. Each builder can have one or more buildpacks.
3. Builders can support source code written in multiple languages.
4. Use the command line tool *pack* with a builder to turn source code into a container image.

Here's what's important to remember about building container images with Buildpacks:

- Buildpacks are a way to turn source code into a container image without writing a Dockerfile.
- Buildpacks are distributed and executed in OCI images called builders. Each builder can have one or more buildpacks.
- Builders can support source code written in multiple languages.
- With the command line tool "pack", you can use a builder to turn source code into a container image.