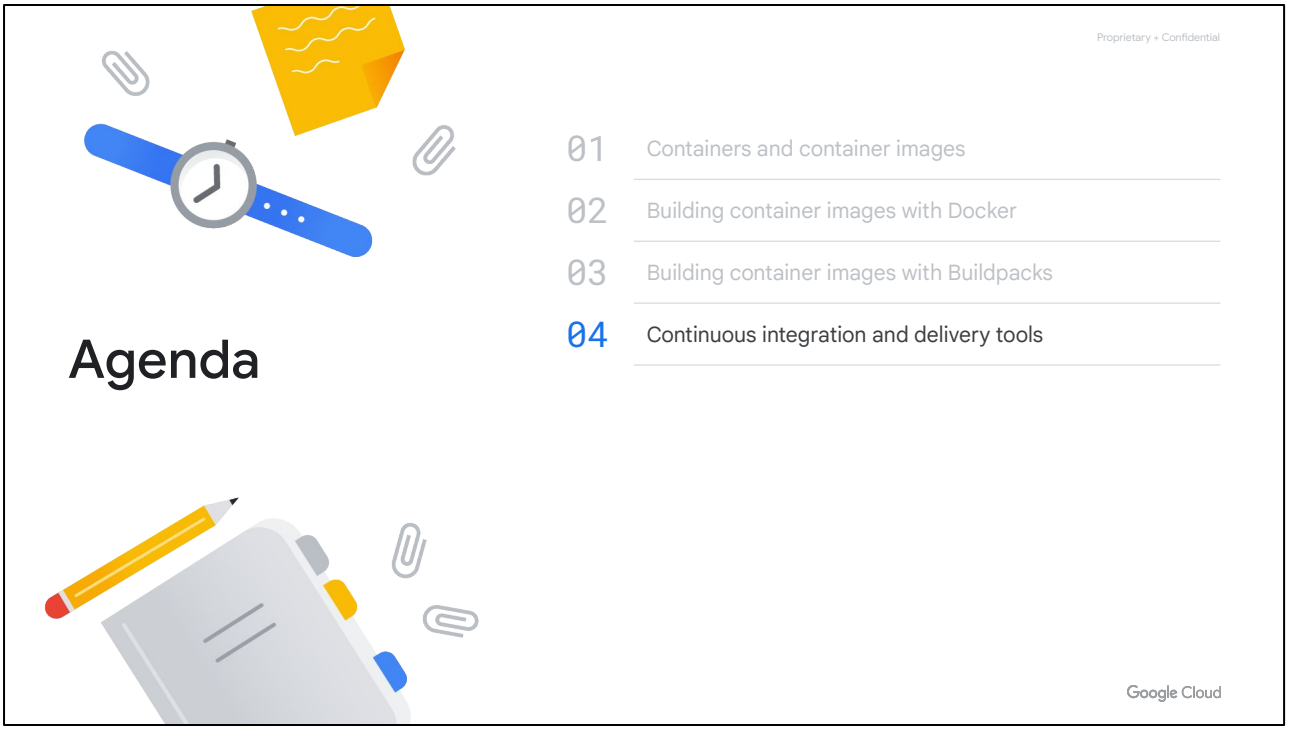


In this lesson, we review some of the tools to implement continuous integration and delivery (CI/CD) for your container images.

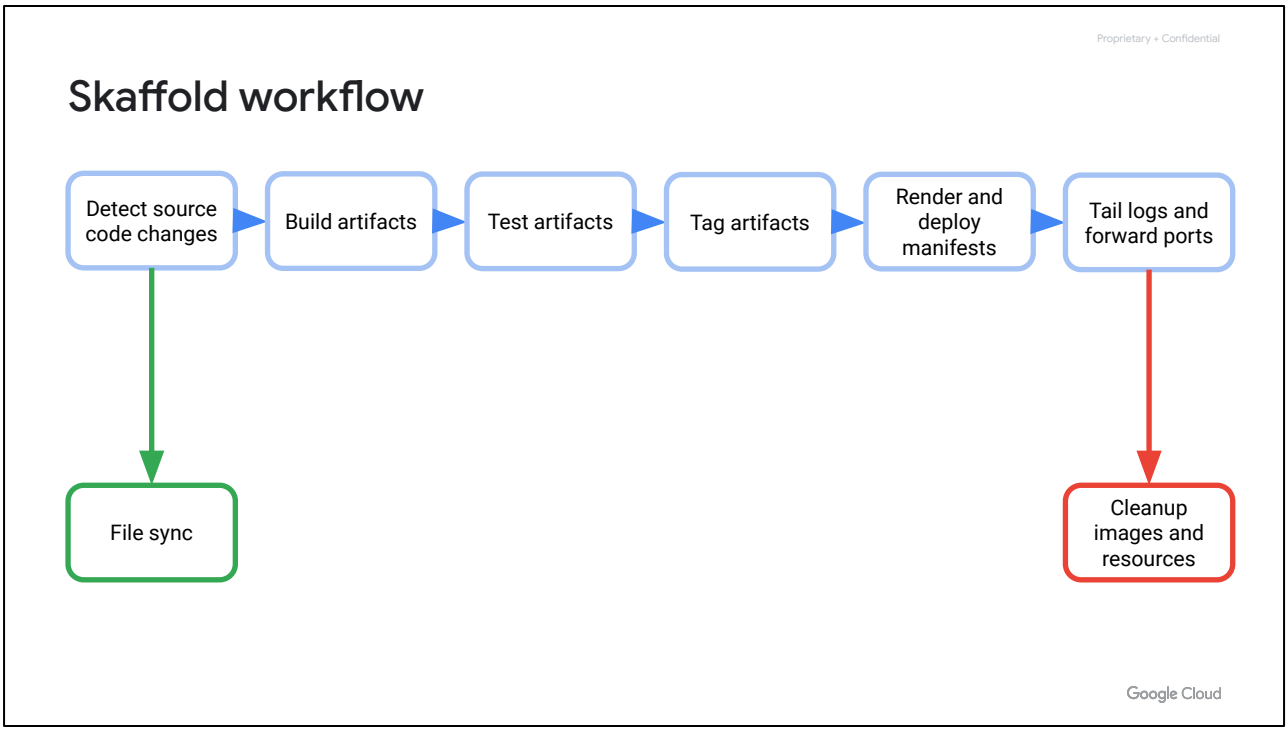
Skaffold

- A tool to orchestrate continuous development, continuous integration (CI), and continuous delivery (CD) of your application code.
- A Google open source project
- Provides building blocks for creating CI/CD pipelines
- Uses portable, declarative configuration

Skaffold is a command-line tool that orchestrates continuous development, continuous integration (CI), and continuous delivery (CD) of container-based and Kubernetes applications.

It's a Google open source project that provides declarative, portable configuration with a pluggable architecture.

Skaffold handles the workflow for building, and deploying your application, and provides building blocks for creating CI/CD pipelines. Skaffold can be used to continuously deploy containers to your local or remote Kubernetes cluster, Docker environment, or Cloud Run project.



Skaffold uses a multi-stage workflow.

When you start Skaffold, it collects source code changes in your project and builds artifacts with the tool of your choice. Skaffold supports various tools like Dockerfiles, Cloud Native Buildpacks and others for building your containers locally, and remotely with Google Cloud Build. For files that don't need to be built, Skaffold supports syncing or copying changed files to a deployed container.

Once the artifacts are successfully built, they are tested, tagged, and pushed to the repository you specify. Skaffold lets you skip stages. For example, if you run Kubernetes locally with Minikube, Skaffold will not push artifacts to a remote repository.

In the test phase, you can run custom commands for unit and integration tests, validation, security scans, and structure tests on the container image. You can also tag your built image with different tag policies like gitCommit, sha256, and others that are supported by Skaffold.

Skaffold also helps you deploy the artifacts to your Kubernetes cluster using the tools you prefer.

To deploy your containers to Kubernetes, Skaffold supports the rendering of manifests using raw Yaml, or rendering tools *helm*, *kpt*, and *kustomize*. In this phase, Skaffold replaces untagged image names in the manifests with the final tagged image names, and might expand templates for *helm*, or calculate overlays for *kustomize*.

docker to deploy to a Docker runtime. Skaffold also supports deployment to Cloud Run with a valid Cloud Run yaml manifest.

Skaffold has built-in support for tailing logs for containers built and deployed by Skaffold when running in either dev, debug or run mode. There is also support for forwarding ports from exposed Kubernetes resources on your cluster to your local machine.

Skaffold also has cleanup functionality to delete Kubernetes resources, and Docker images.

https://skaffold.dev/docs/pipeline-stages/

Skaffold configuration - skaffold.yaml

```
apiVersion: skaffold/v1beta13
kind: Config
build:
  artifacts:
    - image: skaffold-app
      context: app
      docker:
        dockerfile: Dockerfile
deploy:
  kustomize:
    paths:
      - overlays/dev
```

Build section defines how to build the image.

Deploy section defines how to deploy the image.

Skaffold requires a YAML configuration file (skaffold.yaml) that defines how your project is built and deployed.

By running the *skaffold init* command, you can get started with Skaffold using a wizard that generates the required skaffold.yaml file in the root of your project directory.

The skaffold.yaml defines your build and deploy configuration for your container and resources.

Here's a partial sample skaffold.yaml file. Notice the Kubernetes-like YAML format specification.

The build section contains configuration that defines how the container image should be built. It has an artifacts section that defines the container images that comprise the application, and specifies the Dockerfile to be used to build the container image.

The deploy section contains configuration that defines how the container should be deployed. In this sample, the default deployment uses the Kustomize tool. Kustomize is a tool that is used to generate Kubernetes manifests by combining a set of common YAML files in a base directory with one of more overlays. An overlay typically corresponds to a deployment environment, for example, dev, staging, or production.

Skaffold also supports a *profiles* section in the configuration file (not shown), that

defines build, test, and deployment configurations for different contexts or environments in your deployment pipeline, like staging or production. This enables you to easily manage different manifests for each environment, without repeating common configuration.

For more information on Skaffold and its commands, please refer to the documentation on the Skaffold website.

Artifact Registry

- A Google Cloud service used to store and manage software artifacts like container images.
- Integrates with Cloud Build.

Artifact Registry is a service 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.

Cloud Build

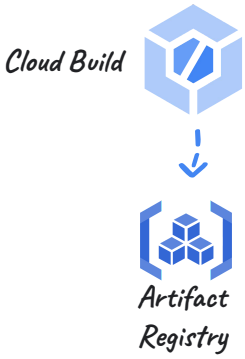
- A service that executes builds on Google Cloud.
- Imports source code from various repositories or cloud storage spaces.
- Produces artifacts such as Docker containers or Java archives.
- Works with a build configuration to:
  - Fetch dependencies.
  - Run unit and integration tests.
  - Perform static analyses.
  - Create artifacts with build tools.

Cloud Build is a service that executes your builds on Google Cloud. With Cloud Build, you can continuously build, test, and deploy your application using a CI/CD pipeline.

Cloud Build can import source code from various repositories or cloud storage spaces, execute a build to your specifications, and produce artifacts such as Docker containers or Java archives.

To provide instructions to Cloud Build, you create a build configuration file that contains a set of tasks. These instructions can configure builds to fetch dependencies, run unit and integration tests, perform static analyses, and create artifacts with build tools (builders) like docker, gradle, maven, and others.

Build process



Cloud Build:

- Automatically builds your code into a container image.
- Pushes the image to Artifact Registry.

Build Process:

- Executes in your project.
- The Cloud Build API must be enabled.
- Provides access to all build logs through Cloud Logging.

The process of building a container image is entirely automatic and requires no direct input from you.

The Cloud Build API must be enabled for your project.

All the resources used in the build process execute in your own user project, and you have access to all the build logs through Cloud Logging.

Features of Cloud Build

- Write code in any programming language.
- Use a build configuration file.
- Integrate with different source code repositories.
- Automatically build, test, and deploy your code.
- Store build artifacts in different registries and storage systems.
- Deploy to popular platforms.

With Cloud Build, you can build your applications written in Java, Go, Python, Node.js, or any programming language of your choice.

Cloud Build integrates with different source code repositories such as GitHub, Bitbucket, and GitLab. You can store your application source code in any of these repositories and use Cloud Build to automate building, testing, and deploying your code.

You can use Artifact Registry with Cloud Build to store build artifacts. You can also store artifacts in other storage systems such as Cloud Storage.

Cloud Build supports the deployment of your application code to popular deployment platforms such as Cloud Run, Google Kubernetes Engine, Cloud Functions, Anthos, and Firebase.

Cloud Build steps

```
steps:
- name: 'gcr.io/cloud-builders/docker'
  args: ['build',
  'us-central1-docker.pkg.dev/${PROJECT_ID}/my-repo/my-image', '.']
- name: 'gcr.io/cloud-builders/docker'
  args: ['push',
  'us-central1-docker.pkg.dev/${PROJECT_ID}/my-repo/my-image']
- name: 'gcr.io/google.com/cloudsdktool/cloud-sdk'
  entrypoint: 'gcloud'
  args: ['compute', 'instances', 'create-with-container', 'my-vm-name',
  '--container-image',
  'us-central1-docker.pkg.dev/${PROJECT_ID}/my-repo/my-image']
  env:
  - 'CLOUDSDK_COMPUTE_REGION=us-central1'
  - 'CLOUDSDK_COMPUTE_ZONE=us-central1-a'
```


Here's a basic build configuration file. The build configuration file is named cloudbuild.yaml and can be written in YAML or JSON format.

Instructions are written as a set of steps. Each step must contain a *name* field that specifies a cloud builder, which is a container image that runs common tools. In this sample, we have a build step with a docker builder, which is an image running Docker.

The *args* field of a step takes a list of arguments and passes them to the builder. The values in the args list are used to access the builder's entrypoint. If the builder does not have an entrypoint, the first element in the args list is used as the entrypoint.

In the example shown:

1. The first build step builds a container image using the docker builder from source code located in the current directory.
2. The second build step uses the docker push command to push the image that was built in the previous step to Artifact Registry.
3. The third step uses the Cloud SDK with the gcloud entrypoint to create a Compute Engine instance from the container image.

There are additional fields that you can use to configure the build. For a complete list of fields, view the Cloud Build documentation.

Running builds with Cloud Build

`gcloud builds submit` command:

1. Uploads your application code, and other files in the current directory to Cloud Storage.
2. Initiates a build in the specified region.
3. Tags the image with the specified name.
4. Pushes the built image to Artifact Registry.

Build with Dockerfile

```
gcloud builds submit \
--region=us-central1 --tag \
$REPO/my-image .
```

Build with configuration file

```
gcloud builds submit \
--region=us-central1 \
--config=cloudbuild.yaml .
```


With Cloud Build, you can run builds manually or use build triggers.

To start a build manually, you use the Google Cloud gcloud CLI or Cloud Build API.

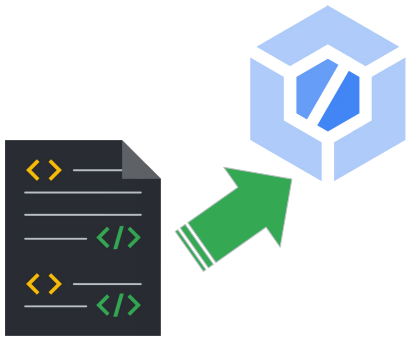
You can use a Dockerfile or a Cloud Build configuration file with the *gcloud builds submit* command.

The command first uploads your application source code and other files from the specified directory to Cloud Storage. It then builds the container image containing your application from the instructions specified in the Dockerfile or build configuration file, tags the image with the specified image name and pushes the image to the appropriate registry.

You can also use this command with Cloud Native Buildpacks to build a container image without a Dockerfile or build configuration file.

To use the Cloud Build API, you create a JSON request file containing the source code location, build steps, and other information. Then, submit a POST request to the /builds endpoint using Curl or other HTTP client and with the appropriate authentication credentials.

Cloud Build triggers



- Automatically run builds with Cloud Build triggers.
- Changes can be made to source code in:
  - Cloud Source Repositories
  - GitHub
  - Bitbucket
- A Dockerfile or Cloud Build configuration file is required.

You can run builds automatically with Cloud Build triggers.

A Cloud Build **trigger** automatically starts a build whenever you make any changes to your source code in a Google Cloud Source repository, GitHub, or Bitbucket repository. Build instructions must be supplied in a Dockerfile or Cloud Build configuration file.

You must first connect Cloud Build to your source repository before building the code in that repository. Repositories in Cloud Source Repositories are connected to Cloud Build by default. To connect an external repository hosted in GitHub or Bitbucket, follow the steps that are outlined in the documentation.

Creating a Cloud Build trigger

Create a trigger for GitHub

```
gcloud beta builds triggers
create github \
--name=my-trigger \
--region=us-central1 \
--repo-name=repos \
--branch-pattern=".*" \
--build-config=cloudbuild.yaml \
--service-account=sa_email
```

To create a Cloud Build trigger, provide:

- The trigger name, description, and cloud region
- The trigger event
- Source repository and branch
- Build configuration
- Service account email

Cloud Build triggers are created in the Google Cloud console or with the gcloud CLI.

To create a build trigger, you provide:

- The name, description, and cloud region for the trigger.
- The trigger event in the remote origin of your source code repository, which can be:
  - A commit or push to a particular branch
  - A commit or push that contains a specific tag
  - A commit to a pull request in GitHub.
- The source repository that contains your source code.
- A regular expression that identifies the source branch or tag.
- The location and type of your build configuration, which can be:
  - A Cloud Build configuration file in YAML or JSON format
  - A Dockerfile
  - Buildpacks
- The email associated with your service account, or the default Cloud Build service account.

For a complete list of build trigger configuration items, refer to the documentation.

The example shows how you can use the gcloud CLI to create a build trigger to

automatically build your source code that resides in a GitHub repository, when a push event occurs on any branch in the repository.

When using GitHub or Bitbucket repositories, you must first connect Cloud Build to your repository before building code in that repository. For more information on how to connect Cloud Build to source repositories, refer to the documentation.

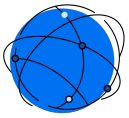
Other types of Cloud Build triggers

Manual triggers

Invoke builds manually and deploy code to other environments.

Pub/Sub triggers

Execute builds in response to Pub/Sub events.



Webhook triggers

Connect external source code management systems to Cloud Build.

You can also create triggers to invoke builds manually. Manual triggers enable you to create a pipeline to automatically fetch code from a hosted repository with a specific branch or tag, and then manually build and deploy that code to other environments.

Cloud Build Pub/Sub triggers enable you to execute builds in response to Pub/Sub events. For example, you can create a Pub/Sub trigger to automate builds in response to events on Artifact Registry, and Cloud Storage. Use cases for these types of events include pushing, tagging, or deleting images in the registry.

Cloud Build enables you to define webhook triggers which authenticate and process incoming webhook events to a custom URL. With this capability, you can connect external source code management systems (for example GitLab, Bitbucket.com) to Cloud Build and use a build configuration file to automate your build.

View the documentation for more information on using Cloud Build Pub/Sub triggers and webhook triggers.