Build configuration overview

A build config file contains instructions for Cloud Build to perform tasks based on your specifications. For example, your build config file can contain instructions to build, package, and push Docker images.

This page explains the structure of the Cloud Build build configuration file.

Structure of a build config file

Build config files are modeled using the Cloud Build API's `Build` (/cloud-build/docs/api/reference/rest/v1/projects.builds#Build) resource.

You can write the build config file using the YAML or the JSON syntax. If you submit build requests using third-party http tools such as curl, use the JSON syntax.

If you're using VS Code or IntelliJ IDEs, you can use Cloud Code (/code) to author your YAML config files. For m ation, see Cloud Code for VS Code (/code/docs/vscode/yaml-editing) and Cloud Code for IntelliJ e/docs/intellij/yaml-editing).

A build config file has the following structure:

YAMLJSON (#json)

```
steps:
- name: string
  args: [string, string, ...]
  env: [string, string, ...]
  dir: string
  id: string
  waitFor: [string, string, ...]
  entrypoint: string
  secretEnv: string
  volumes: object(Volume)
  timeout: string (Duration format)
- name: string
  ...
```

```
- name: string
  ...
timeout: string (Duration format)
queueTtl: string (Duration format)
logsBucket: string
options:
 env: [string, string, ...]
 secretEnv: string
 volumes: object(Volume)
 sourceProvenanceHash: enum(HashType)
 machineType: enum(MachineType)
 diskSizeGb: string (int64 format)
 substitutionOption: enum(SubstitutionOption)
 logStreamingOption: enum(LogStreamingOption)
 logging: enum(LoggingMode)
substitutions: map (key: string, value: string)
tags: [string, string, ...]
secrets: object(Secret)
images:
- [string, string, ...]
artifacts: object (Artifacts)
```

Each of the sections of the build config file defines a part of the task you want Cloud Build to execute:

Build steps

A **build step** specifies an action that you want Cloud Build to perform. For each build step, Cloud Build executes a docker container as an instance of `docker run`. Build steps are analogous to commands in a script and provide you with the flexibility of executing arbitrary instructions in your build. If you can package a build tool into a container, Cloud Build can execute it as part of your build. By default, Cloud Build executes all steps of a build serially on the same machine. If you have steps that can run concurrently, use the waitFor (#waitfor) option.

You can include one or more build steps in your config file.

Use the `steps` field in the build config file to specify a build step. Here's a snippet of the kind of configuration you might set in the `steps` field:

YAMLJSON (#json)

```
steps:
- name: 'gcr.io/cloud-builders/kubectl'
  args: ['set', 'image', 'deployment/mydepl', 'my-image=gcr.io/my-project/myimage']
  env:
  - 'CLOUDSDK_COMPUTE_ZONE=us-east4-b'
  - 'CLOUDSDK_CONTAINER_CLUSTER=my-cluster'
- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/my-project-id/myimage', '.']
```

name

Use the `name` field of a build step to specify a cloud builder (/cloud-build/docs/cloud-builders), which is a container image running common tools. You use a builder in a build step to execute your tasks.

The following snippet shows build steps calling the `bazel` (https://github.com/GoogleCloudPlatform/cloud-builders/tree/master/bazel), `gcloud` (https://github.com/GoogleCloudPlatform/cloud-builders/tree/master/gcloud), and `docker` (https://github.com/GoogleCloudPlatform/cloud-builders/tree/master/docker) builders:

YAMLJSON (#json)

```
steps:
- name: 'gcr.io/cloud-builders/bazel'
...

- name: 'gcr.io/cloud-builders/gcloud'
...

- name: 'gcr.io/cloud-builders/docker'
...
```

args

The `args` field of a build step takes a list of arguments and passes them to the builder referenced by the `name` field. Arguments passed to the builder are passed to the tool that's running in the builder, which allows you to invoke any command supported by the tool. If the

builder used in the build step has an entrypoint, args will be used as arguments to that entrypoint. If the builder does not define an entrypoint, the first element in args will be used as the entrypoint, and the remainder will be used as arguments.

The following snippet invokes the `docker build` command and installs Maven dependencies:

YAMLJSON (#json)

```
steps:
- name: 'gcr.io/cloud-builders/mvn'
  args: ['install']
- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/my-project-id/myimage', '.']
```

env

The `env` field of a build step takes a list of environment variables to be used when running the step. The variables are of the form `KEY=VALUE`.

In the following build config the `env` field of the build step sets the Compute Engine zone and the GKE cluster prior to executing `kubectl`:

YAMLJSON (#json)

```
steps:
- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/myproject/myimage', '.']
- name: 'gcr.io/cloud-builders/kubectl'
  args: ['set', 'image', 'deployment/myimage', 'frontend=gcr.io/myproject/myimage']
  env:
  - 'CLOUDSDK_COMPUTE_ZONE=us-east1-b'
  - 'CLOUDSDK_CONTAINER_CLUSTER=node-example-cluster'
```

dir

Use the `dir` field in a build step to set a working directory to use when running the step. By default, Cloud Build uses a directory named `/workspace` as a working directory. If your config

file has more than one build step, the assets produced by one step can be passed to the next one via the persistence of the `/workspace` directory, which allows you to set up a pipeline of build steps that share assets. If you set the `dir` field in the build step, the working directory is set to `/workspace/<dir>`. If this value is a relative path, it is relative to the build's working directory. If this value is absolute, it may be outside the build's working directory, in which case the contents of the path may not be persisted across build step executions (unless a volume (#volumes) for that path is specified).

The following snippet sets the working directory as `examples/hello_world`:

YAMLJSON (#json)

```
steps:
- name: 'gcr.io/cloud-builders/go'
  args: ['install', '.']
  env: ['PROJECT_ROOT=hello']
  dir: 'examples/hello_world'
```

timeout

Use the `timeout` field in a build step to set a time limit for executing the step. If you don't set this field, the step has no time limit and will be allowed to run until either it completes or the build itself times out. The `timeout` field in a build step must not exceed the `timeout` (#timeout_2) value specified for a build. `timeout` must be specified in seconds with up to nine fractional digits, terminated by 's'. Example: "3.5s"

In the following build config, the `ubuntu` step is timed out after 500 seconds:

YAMLJSON (#json)

```
steps:
- name: 'ubuntu'
  args: ['sleep', '600']
  timeout: 500s
- name: 'ubuntu'
  args: ['echo', 'hello world, after 600s']
```

The timeout field exists for both the build step and the build: the timeout field of a build step specifies the amou
ıe step is allowed to run, and the timeout field of a build
d-build/docs/api/reference/rest/v1/projects.builds#resource-build) specifies the amount of time the build is
d to run.

id

Use the `id` field to set a unique identifier for a build step. `id` is used with the `waitFor` field to configure the order in which build steps should be run. For instructions on using `waitFor` and `id`, see Configuring build step order (/cloud-build/docs/configuring-builds/configure-build-step-order).

waitFor

Use the `waitFor` field in a build step to specify which steps must run before the build step is run. If no values are provided for `waitFor`, the build step waits for all prior build steps in the build request to complete successfully before running. For instructions on using `waitFor` and `id`, see Configuring build step order (/cloud-build/docs/configuring-builds/configure-build-step-order).

entrypoint

Use the `entrypoint` in a build step to specify an entrypoint if you don't want to use the default entrypoint of the builder. If you don't set this field, Cloud Build will use the builder's entrypoint. The following snippet sets the entrypoints for the `npm` build step:

YAMLJSON (#json)

```
steps:
- name: 'gcr.io/cloud-builders/npm'
  entrypoint: 'node'
  args: ['--version']
```

secretEnv

A list of environment variables which are encrypted using a Cloud KMS crypto key. These values must be specified in the build's secrets. For information on using this field see Using the

encrypted variable in build requests (/cloud-build/docs/securing-builds/use-encrypted-secrets-credentials#using_the_encrypted_variable_in_build_requests).

volumes

A **Volume** (/cloud-build/docs/api/reference/rest/v1/projects.builds#volume) is a Docker container volume that is mounted into build steps to persist files across build steps. When Cloud Build runs a build step, it automatically mounts a `workspace` volume into `/workspace`. You can specify additional volumes to be mounted into your build steps' containers using the `volumes` field for your steps.

For example, the following build config file writes a file into a volume in the first step and reads it in the second step. If these steps did not specify `/persistent_volume` path as a persistent volume, the first step would write the file at that path, then that file would be discarded before the second step executes. By specifying the volume with the same name in both steps, the contents of `/persistent_volume` in the first step are persisted to the second step.

YAMLJSON (#json)

```
steps:
- name: 'ubuntu'
  volumes:
  - name: 'vol1'
    path: '/persistent_volume'
  entrypoint: 'bash'
  args:
  - '-c'
  - |
        echo "Hello, world!" > /persistent_volume/file
- name: 'ubuntu'
  volumes:
  - name: 'vol1'
    path: '/persistent_volume'
  args: ['cat', '/persistent_volume/file']
```

timeout

Use the `timeout` field for a build to specify the amount of time that the build must be allowed to run, to second granularity. If this time elapses, work on the build will cease and the build status (/cloud-build/docs/api/reference/rest/v1/projects.builds#status) will be `TIMEOUT`. If `timeout` is not set, a default `timeout` of 10 minutes will apply to the build. The maximum value that can be applied to `timeout` is 24 hours. `timeout` must be specified in seconds with up to nine fractional digits, terminated by 's'. Example: "3.5s"

In the following snippet, `timeout` is set to 660 seconds to avoid the build from timing out because of the sleep:

YAMLJSON (#json)

```
steps:
- name: 'ubuntu'
  args: ['sleep', '600']
timeout: 660s
```

The timeout field exists for both the build step and the build: the **`timeout`** field of a build step d-build/docs/api/reference/rest/v1/projects.builds#buildstep) specifies the amount of time the step is allowe ıd the **`timeout`** field of a build specifies the amount of time the build is allowed to run.

queueTtl

Use the `queueTtl` field to specify the amount of time a build can be queued. If a build is in the queue for longer than the value set in `queueTtl`, the build expires and the build status (/cloud-build/docs/api/reference/rest/v1/projects.builds#status) is set to `EXPIRED`. `queueTtl` starts ticking from `createTime`. `queueTtl` must be specified in seconds with up to nine fractional digits, terminated by 's', for example, "3.5s".

In the following snippet `timeout` is set to "20s" and `queueTtl` is set to "10s". `queueTtl` starts ticking at `createTime`, which is the time the build is requested, and `timeout` starts ticking at `startTime`, which is the time the build starts. Therefore, `queueTtl` will expire at `createTime` + 10s unless the build starts by then.

YAMLJSON (#json)

```
steps:
- name: 'ubuntu'
  args: ['sleep', '5']
timeout: 20s
queueTtl: 10s
```

logsBucket

Set the `logsBucket` field for a build to specify a Cloud Storage bucket where logs must be written. If you don't set this field, Cloud Build will use a default bucket to store your build logs.

The following snippet sets a logs bucket to store the build logs:

YAMLJSON (#json)

```
steps:
- name: 'gcr.io/cloud-builders/go'
  args: ['install', '.']
logsBucket: 'gs://mybucket'
```

options

Use the `options` (/cloud-build/docs/api/reference/rest/v1/projects.builds#buildoptions) field to specify the following optional arguments for your build:

`env` (/cloud-build/docs/api/reference/rest/v1/projects.builds#buildoptions): A list of global environment variable definitions that will exist for all build steps in this build. If a variable is defined in both globally and in a build step, the variable will use the build step value. The elements are of the form `KEY=VALUE` for the environment variable `KEY` being given the value `VALUE`.

`secretEnv` (/cloud-build/docs/api/reference/rest/v1/projects.builds#buildoptions): A list of global environment variables, encrypted using a Cloud Key Management Service crypto key, that will be available to all build steps in this build. These values must be specified in the build's `Secret`.

`volumes` (/cloud-build/docs/api/reference/rest/v1/projects.builds#Build.Volume): A list of volumes to mount globally for ALL build steps. Each volume is created as an empty volume prior to

starting the build process. Upon completion of the build, volumes and their contents are discarded. Global volume names and paths cannot conflict with the volumes defined a build step. Using a global volume in a build with only one step is not valid as it signifies a build request with an incorrect configuration.

`sourceProvenanceHash` (/cloud-build/docs/api/reference/rest/v1/projects.builds#sourceprovenance): Set the `sourceProvenanceHash` option to specify the hash algorithm for source provenance. The following snippet specifies that the hash algorithm is `SHA256`:

YAMLJSON (#json)

```
steps:
- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/myproject/myimage', '.']
options:
    sourceProvenanceHash: ['SHA256']
```

`machineType` (/cloud-build/docs/api/reference/rest/v1/projects.builds#machinetype): Cloud Build provides two high-CPU virtual machine types to run your builds: 8 CPUs and 32 CPUs. The default machine type is 1 CPU. Requesting a high-CPU virtual machine may increase the startup time of your build. Add the `machineType` option to request a virtual machine with a higher CPU:

YAMLJSON (#json)

```
steps:
- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/myproject/myimage', '.']
options:
 machineType: 'N1_HIGHCPU_8'
```

For more information on using the `machineType` option see Speeding up builds (/cloud-build/docs/speeding-up-builds#using_custom_virtual_machine_sizes).

`diskSizeGb` (/cloud-build/docs/api/reference/rest/v1/projects.builds#buildoptions): Use the `diskSizeGb` option to request a custom disk size for your build. The maximum size you can request is 1000 GB.

The following snippet requests a disk size of 200 GB:

YAMLJSON (#json)

```
steps:
- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/myproject/myimage', '.']
options:
 diskSizeGb: 200
```

`logStreamingOption` (/cloud-build/docs/api/reference/rest/v1/projects.builds#logstreamingoption): Use this option to specify if you want to stream build logs to Cloud Storage. By default, Cloud Build collects build logs on build completion; this option specifies if you want to stream build logs in real time through the build process. The following snippet specifies that build logs are streamed to Cloud Storage:

YAMLJSON (#json)

```
steps:
- name: 'gcr.io/cloud-builders/go'
  args: ['install', '.']
options:
 logStreamingOption: STREAM_ON
```

`logging` (/cloud-build/docs/api/reference/rest/v1/projects.builds#loggingmode): Use this option to specify if you want to store logs in Cloud Logging or Cloud Storage. If you do not set this option, Cloud Build stores the logs in both Cloud Logging and Cloud Storage. You can set the `logging` option to `GCS_ONLY` to store the logs only in Cloud Storage. The following snippet specifies that the logs are stored in Cloud Storage:

YAMLJSON (#json)

```
steps:
- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/myproject/myimage', '.']
```

```
options:
 logging: GCS_ONLY
```

substitutionOption (/cloud-build/docs/api/reference/rest/v1/projects.builds#substitutionoption): You'll set this option along with the `substitutions` field below to specify the behavior when there is an error in the substitution checks (/cloud-build/docs/configuring-builds/substitute-variable-values#using_user-defined_substitutions).

substitutions

Use substitutions in your build config file to substitute specific variables at build time. Substitutions are helpful for variables whose value isn't known until build time, or to re-use an existing build request with different variable values. By default, the build returns an error if there's a missing substitution variable or a missing substitution. However, you can use the `ALLOW_LOOSE` option to skip this check.

The following snippet uses substitutions to print "hello world." The `ALLOW_LOOSE` substitution option is set, which means the build will not return an error if there's a missing substitution variable or a missing substitution.

YAMLJSON (#json)

```
steps:
- name: 'ubuntu'
  args: ['echo', 'hello ${_SUB_VALUE}']
substitutions:
    _SUB_VALUE: world
options:
    substitution_option: 'ALLOW_LOOSE'
```

If your build is invoked by a trigger, the **`ALLOW_LOOSE`** option is set by default. In this case, your build will not re
or if there is a missing substitution variable or a missing substitution. You cannot override the **`ALLOW_LOOSE`** o
lds invoked by triggers.

For additional instructions on using `substitutions`, see Substituting variable values (/cloud-build/docs/configuring-builds/substitute-variable-values).

tags

Use the `tags` field to organize your builds into groups and to filter your builds (/cloud-build/docs/view-build-results#filtering_build_results_using_tags). The following config sets two tags named `mytag1` and `mytag2`:

YAML JSON (#json)

```
steps:
- name: 'gcr.io/cloud-builders/docker'
  ...
- name: 'ubuntu'
  ...
tags: ['mytag1', 'mytag2']
```

secrets

Secret pairs a set of secret environment variables containing encrypted values with the Cloud KMS key to use to decrypt the value. Use the `secrets` field to define secrets to decrypt using Cloud KMS. For an example of how to use this field, see Using encrypted secrets and credentials (/cloud-build/docs/securing-builds/use-encrypted-secrets-credentials#example_build_request_using_an_encrypted_variable).

images

The `images` field in the build config file specifies one or more Docker images to be pushed by Cloud Build to Container Registry. You may have a build that performs tasks without producing any Docker images, but if you build images and don't push them to Container Registry, the images are discarded on build completion. If a specified image is not produced during the build, the build will fail. For more information on storing images see Storing Images and Artifacts (/cloud-build/docs/configuring-builds/store-images-artifacts).

The following build config sets the `images` field to store the built image:

```
steps:
- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/myproject/myimage', '.']
images: ['gcr.io/myproject/myimage']
```

artifacts

The `artifacts` field in the build config file specifies one or more non-container artifacts to be stored in Cloud Storage. For more information on storing non-container artifacts see Storing Images and Artifacts (/cloud-build/docs/configuring-builds/store-images-artifacts).

The following build config sets the `artifacts` field to store the built Go package to `gs://mybucket/`:

YAMLJSON (#json)

```
steps:
- name: 'gcr.io/cloud-builders/go'
  args: ['build', 'my-package']
artifacts:
  objects:
    location: 'gs://mybucket/'
    paths: ['my-package']
```

Using Dockerfiles

If you're executing Docker builds in Cloud Build using the `gcloud` tool or build triggers (/cloud-build/docs/running-builds/create-manage-triggers), you can use only a `Dockerfile` (https://docs.docker.com/engine/reference/builder/) to build the image. You don't require a separate build config file. If you wish to make more adjustments to your Docker builds, you can provide a build config file in addition to the Dockerfile. For instructions on how to build a Docker image using a `Dockerfile`, see Quickstart for Docker (/cloud-build/docs/quickstart-docker).

Cloud Build network

When Cloud Build runs each build step, it attaches the step's container to a local Docker network named `cloudbuild`. The `cloudbuild` network hosts Application Default Credentials (/../../docs/authentication/production) (ADC) that Google Cloud services can use to automatically find your credentials. If you're running nested Docker containers and want to expose ADC to an underlying container, use the `--network` flag in your docker `build` step:

YAMLJSON (#json)

```
steps:
- name: gcr.io/cloud-builders/docker
  args: ["build","--network=cloudbuild", "."]
```

What's next

- Learn how to create a basic build config file (/cloud-build/docs/configuring-builds/create-basic-configuration) to configure builds for Cloud Build.
- Read Starting a Build Manually (/cloud-build/docs/running-builds/start-build-manually) for instructions on how to run builds in Cloud Build.

Last updated 2020-06-22 UTC.