ChatGPT

Introduction

Provisioning the Cloud Build *runtime service account* (the **legacy Cloud Build service account** with email `PROJECT_NUMBER@cloudbuild.gserviceaccount.com`) in a deterministic way – **using only REST APIs and no** `gcloud` **CLI** – is now possible. In the past, Google-managed service accounts like Cloud Build's were created at unpredictable times (sometimes when enabling the API, other times on first use), which made automated permission setup difficult [1]. Below we outline a reliable, step-by-step plan to ensure this service account exists and has the correct IAM roles for triggering builds and pushing images, all via REST calls.

Important Context: Starting in mid-2024, new projects *by default* use the Compute Engine default service account (PROJECT_NUMBER-compute@developer.gserviceaccount.com) to run Cloud Build, *not* the legacy Cloud Build account [2]. The legacy Cloud Build service account is disabled by default for new projects unless an organization policy allows it [3] [4]. This plan assumes you **want to use** the legacy Cloud Build service account; ensure your organization or project policy permits its creation (see **Org Policy** note below).

1. Enable the Cloud Build API (if not already)

Make sure the Cloud Build API (`cloudbuild.googleapis.com`) is enabled for your project. This can be done via the Service Usage API's `services.enable` method or in Cloud Console. Enabling the API is a prerequisite to provisioning the service account.

- *Org Policy Note:* If your project was created after June 2024 and is under an organization, Cloud Build will default to the Compute Engine service account unless you **opt out of the new behavior** via org policy [3]. To **use the legacy Cloud Build service account**, set the org policy constraints to allow it: for example, ensure `constraints/cloudbuild.disableCreateDefaultServiceAccount` is **Not Enforced** and `constraints/cloudbuild.useBuildServiceAccount` is **Enforced** (this explicitly opts in to using the Cloud Build service account) [5] [6]. If you cannot change these policies, you will have to use an alternate service account (Compute Engine default or a user-created one) for Cloud Build.

2. Deterministically Create the Cloud Build Service Account via REST

To avoid "lazy" creation (e.g. waiting for the first build or trigger to implicitly create the account), use the **Service Usage API**'s *generateServiceIdentity* method. This is available in the `v1beta1` endpoint and will force the creation of the Google-managed service account for Cloud Build if it doesn't already exist [1] [7]. It returns the service account's email, ensuring you can retrieve it reliably on first-run.

Call: `POST https://serviceusage.googleapis.com/v1beta1/projects/<PROJECT_NUMBER>/services/cloudbuild.googleapis.com:generateServiceIdentity` (the request body is empty). Ensure your OAuth token has scope `https://www.googleapis.com/auth/cloud-platform` or `service.management` [8]. This call will return a long-running Operation; you should poll the Operation

(e.g. via `operations.get`) until it completes. On success, the operation's response contains the **ServiceIdentity** with the service account's email.

- **Example:** Using `curl` (with appropriate authentication) –

```
curl -X POST \
  "https://serviceusage.googleapis.com/v1beta1/projects/PROJECT_NUMBER/services/
cloudbuild.googleapis.com:generateServiceIdentity" \
  -H "Authorization: Bearer $TOKEN"
```

This ensures the Cloud Build service account is created and returns its identifier (email) deterministically [7]. *(Under the hood, the gcloud beta command* `gcloud beta services identity create --service=cloudbuild.googleapis.com` *uses this same API* [9] *.)*

If the generateServiceIdentity call returns 404: Make sure you are using the **v1beta1** endpoint (it's not available in the v1 stable API as of 2025). Also ensure the Cloud Build API is enabled first. In our research, this method is the official way to guarantee creation; it should succeed when the project is allowed to have the Cloud Build service account. (Previous workaround was to trigger a dummy build, but that is not reliable and is no longer needed now that this API exists.)

3. Grant Required Permissions to the Service Account (via IAM REST APIs)

Once the service account exists, you must assign it the IAM roles needed for your use case. By default, if the legacy Cloud Build service account is enabled, Google automatically grants it the **"Cloud Build Service Account" role** (`roles/cloudbuild.builds.builder`) on your project [10] [11]. This role allows it to run builds, write Cloud Logs, etc., but **it does NOT automatically include permissions to push artifacts to Artifact Registry or Cloud Storage buckets**. You'll need to grant those explicitly:

- **Artifact Registry Push Permissions:** Assign the service account the **Artifact Registry Writer** role (`roles/artifactregistry.writer`) on the relevant scope. Granting this role at the project level is simplest if you want it to push to any Artifact Registry repository in the project. This role includes permissions to upload and download artifacts in Artifact Registry [12] [13]. (If you prefer finer scope, you could grant it on specific repositories via the Artifact Registry API's repository IAM policy.)

- **Container Registry/Storage Permissions:** If you use Container Registry (or need to push to GCS buckets), grant the **Storage Object Admin** role (`roles/storage.objectAdmin`) on the appropriate Cloud Storage bucket or at the project level. For example, to allow pushing Container Registry images (which are stored in a bucket like `artifacts.<PROJECT>.appspot.com`), you can give this role on the project (covering all buckets in the project) or on that specific bucket. This role lets the Cloud Build service account read/write objects in Cloud Storage. *(Cloud Build's builder role does include basic storage permissions like* `storage.objects.create` [14] *, but to avoid any access issues with artifact storage buckets, granting Storage Object Admin ensures broad access.)*

- **Grant via REST:** Use the Cloud Resource Manager **setIamPolicy** or **patchIamPolicy** on the project (or the target resource) to add these role bindings. For example, to add Artifact Registry Writer at the project level, call:

`POST https://cloudresourcemanager.googleapis.com/v1/projects/<PROJECT_ID>:setIamPolicy` with a request body that includes the new binding:

```
{
  "policy": {
    "bindings": [
      {
        "role": "roles/artifactregistry.writer",
        "members":
["serviceAccount:PROJECT_NUMBER@cloudbuild.gserviceaccount.com"]
      },
      {
        "role": "roles/storage.objectAdmin",
        "members":
["serviceAccount:PROJECT_NUMBER@cloudbuild.gserviceaccount.com"]
      }
    ],
    "etag": "<existing-policy-etag>"
  }
}
```

Include the current policy's etag to do a safe update (you can get the current IAM policy via `projects.getIamPolicy`). This will grant the Cloud Build service account the needed write access to Artifact Registry and Storage.

Verification: After these calls, you can list the IAM policy to confirm the service account appears with the new roles. Also, any subsequent Cloud Build triggers or builds that push an image should succeed in publishing artifacts (no permission denied errors for Artifact Registry or GCS).

4. (Optional) Impersonation by an Admin Service Account

You mentioned allowing a predefined admin service account to impersonate the Cloud Build runtime account. **However, due to Cloud Build's design, this is generally not possible for the legacy service account.** The Cloud Build service account is **Google-owned** (not user-managed), meaning you **cannot modify its own IAM policy** to grant others impersonation rights [11]. In fact, GCP explicitly disallows adding any policy bindings on the legacy Cloud Build service account, so you cannot grant another principal the role `roles/iam.serviceAccountTokenCreator` on it [15]. In practical terms, no service account (or user) can impersonate the Cloud Build **legacy** account to act on its behalf – attempts to do so will fail because you can't assign the necessary *"Service Account User"* or token creator role on that principal.

Workaround: If impersonation or "acting as" the build service account is a requirement (for example, your automation needs to trigger builds as if coming from that account), the recommended approach is to **use a user-managed service account to run Cloud Build**. You can create your own service account, grant it Cloud Build permissions (e.g., Cloud Build Service Account role, plus artifact registry/storage roles), and configure Cloud Build to use that as a **substitute** for the default. Since this new account is user-managed, you *can* grant your admin service account the `roles/iam.serviceAccountUser` (or Token Creator) on it to allow impersonation. This aligns with Google's best practice of using a user-specified service account for Cloud Build [16]. Otherwise, if you must use the Google-managed one, you'll have to run operations as yourself or as a principal with the appropriate Cloud Build IAM roles, rather than impersonating the Cloud Build service account.

5. Triggering Builds and Using the Service Account

With the Cloud Build service account created and granted the necessary roles, you can confidently use it in Cloud Build triggers or API calls:

- When creating Cloud Build triggers via the Cloud Build REST API (or Terraform, etc.), you typically must specify which service account will execute the builds. In projects where the **legacy Cloud Build account is enabled and default**, you can actually leave the trigger's serviceAccount field empty to use it by default [17] [18]. If you explicitly specify it, use the email `PROJECT_NUMBER@cloudbuild.gserviceaccount.com`.

- **Permissions for trigger creators:** If your automation (the "pre-defined admin service account") is setting up triggers, ensure it has the ability to assign that service account to the trigger. Normally, using a **user-specified** service account in a trigger requires the caller to have `iam.serviceAccounts.actAs` on that service account [19]. However, if the legacy account is the default for the project, Cloud Build allows Cloud Build Editors to create triggers without needing actAs on the legacy account [18]. Since your admin service account likely has high-level permissions (perhaps Owner or Cloud Build Admin), this shouldn't be an issue. Just be aware of this requirement if using a non-legacy account.

- After triggers are created, when they fire or when you invoke builds via the API, the Cloud Build service will use the provisioned service account to execute the build. It will now have the necessary permissions to **pull/push images** (Artifact Registry write) and **upload objects** to GCS as needed.

6. Summary of the REST-Only Procedure

a. *Enable Cloud Build API* – Ensure the API is enabled and org policy allows the legacy Cloud Build service account if you intend to use it [3] [4].

b. *Create the Cloud Build service account deterministically* – Call the Service Usage API's `generateServiceIdentity` for `cloudbuild.googleapis.com` [7]. This will synchronously trigger creation of `PROJECT_NUMBER@cloudbuild.gserviceaccount.com` (if it isn't already present) and return its email.

c. *Grant roles via IAM API* – Use Cloud Resource Manager's IAM policy methods to bind the required roles: - `roles/artifactregistry.writer` for artifact push access. - `roles/storage.objectAdmin` for GCS/ bucket access (for container images or artifacts). These are in addition to the Cloud Build Service Account role that Google automatically granted to the service account for build execution.

d. *(Optional)* If using a **custom build service account** instead, create it and grant it the above roles, plus `roles/cloudbuild.builds.builder`. Then grant your admin identity `roles/iam.serviceAccountUser` on it for impersonation. This avoids the limitations of the Google-managed account.

e. Use the service account in Cloud Build triggers or API calls. Your admin automation can create triggers pointing to it (with appropriate IAM permissions as noted). The setup is one-time and the behavior will be reliable on the first build – no "missing service account" or permission errors.

Caveats and Conclusion

- **Impersonation Limitation:** As noted, you **cannot impersonate the Google-managed Cloud Build service account** due to it being non-user-managed [11]. This is a design constraint, not a tooling limitation. The `gcloud` CLI cannot override this either – it doesn't use any hidden API to enable impersonation. It's simply disallowed by Google's IAM system for that account. The solution is to use a different (user-managed) service account if impersonation is required.

- **No Undocumented APIs Required:** The above process uses official REST endpoints. The `gcloud` CLI itself uses these under the hood (for example, `gcloud beta services identity create` calls the Service Usage API as described, and IAM bindings use the standard Cloud Resource Manager calls). There are no secret APIs in play – just ensure you use the **v1beta1** ServiceUsage method for creation. If you were relying on `gcloud` before, this replicates its steps in pure REST calls.

- **First-Build Success:** By proactively creating the service account and assigning roles, you avoid race conditions where the first Cloud Build might fail due to missing permissions or identity. This method guarantees that when your CI/CD pipeline runs for the first time, the Cloud Build runtime service account is ready to go with the correct privileges.

Following this plan, you can provision the Cloud Build runtime service account in an automated, deterministic way without ever invoking `gcloud`. This approach is ideal for Terraform, custom bash scripts, or other IaC pipelines running in constrained environments. It leverages Google's documented APIs [7] and avoids any unpredictable behavior, ensuring your Cloud Build can trigger builds and push images successfully on the first try.

Sources:

- Google Cloud documentation – *Cloud Build service account updates (2024 change)* [2] [4]
- Google Cloud documentation – *Default Cloud Build service account (permissions and restrictions)* [11] [15]

- Christoph Petersen, *“Deterministically creating service identities for APIs in Google Cloud”* – on using `generateServiceIdentity` for Cloud Build [1] [7].

---

[1] [7] [9] Deterministically creating service identities for APIs in Google Cloud | Order to Management
https://www.peterschen.de/deterministically_creating_service_identities_for_apis_in_google_cloud/

[2] [3] [4] [5] [6] Cloud Build default service account change | Cloud Build Documentation | Google Cloud
https://cloud.google.com/build/docs/cloud-build-service-account-updates

[8] Method: services.generateServiceIdentity | Service Usage | Google Cloud
https://cloud.google.com/service-usage/docs/reference/rest/v1beta1/services/generateServiceIdentity

[10] Cannot find cloud build service account for GCP - Stack Overflow
https://stackoverflow.com/questions/79174237/cannot-find-cloud-build-service-account-for-gcp

[11] [12] [13] [14] [15] [16] [17] [18] [19] Default Cloud Build service account | Cloud Build Documentation | Google Cloud
https://cloud.google.com/build/docs/cloud-build-service-account