Installing Python dependencies

This page describes how to install Python packages and connect to your Cloud Composer environment from a few common applications.

Dependencies are installed with the existing Python dependencies that are included in the base environment.

If your environment requires a specific package, we recommend that you explicitly install the package to avoid issues due to package changes across Cloud Composer image versions. Do not rely on the pre-installed packages in the Cloud Composer version that is running in your environment.

Python 3 environments (/composer/docs/concepts/python-version) install only Python 3 packages.

Options for managing dependencies

If your Python dependency has no external dependencies and does not conflict with Cloud Composer's dependencies, you can install Python dependencies from the Python Package Index (/composer/docs/how-to/using/installing-python-dependencies#install-package). You can also install a Python dependency from private package repository (/composer/docs/how-to/using/installing-python-dependencies#install-private).

For other requirements, here are a few options.

Option	Use if ...
Local Python library (#install-local)	Your Python dependency can't be found the Python Package Index, and the library does not have any external dependencies, such as dist-packages.
Plugins (/composer/docs/concepts/plugins) feature	You want to use plugin-specific functionality, such as modifying the Airflow web interface.

Option	Use if ...
PythonVirtualenvOperator (https://github.com/apache/incubator-airflow/blob/1.9.0/airflow/operators/python_operator.py#L178)	Your Python dependency can be found on the Python Package Index and has no external dependencies. However, you don't want your Python dependency to be installed for all workers, or the dependency conflicts with dependencies required for Cloud Composer.
KubernetesPodOperator (/composer/docs/how-to/using/using-kubernetes-pod-operator)	You require external dependencies that can't be installed from pip, such as dist-packages, or are on an internal pip server. This option requires more setup and maintenance and should generally be considered if the other options do not work.

Before you begin

- The following permission is required to install Python packages in the Cloud Composer environment: `composer.environments.update`. For more information, see Cloud Composer Access Control (/composer/docs/how-to/access-control).
- If your environment is protected by a VPC Service Controls perimeter, before installing PyPI dependencies you must grant additional user identities (/composer/docs/configuring-vpc-sc) with access to services that the service perimeter protects and enable support for a private PyPI repository.
- Requirements must follow the format specified in PEP-508 (https://www.python.org/dev/peps/pep-0508/#grammar) where each requirement is specified in lowercase and consists of the package name with optional extras and version specifiers.
- When you install custom Python dependencies by using the API, all Cloud Composer processes run with newly-installed PyPI dependencies.
- Custom PyPI dependencies might cause conflicts with dependencies that Airflow requires, causing instability.

- Before deploying to production, we recommend that you test your PyPI packages locally in an Airflow worker container (/composer/docs/how-to/using/testing-dags#checking_for_pypi_package_errors).

Pypi dependency updates generate Docker images in Container Registry (/container-registry). Do not modify o
the images.

Viewing installed Python packages

To see the installed Python packages in your environment:

1. Connect to the GKE cluster (/composer/docs/how-to/managing/deploy-webserver#connect-cluster) for your environment.
2. Connect to a pod. To access pods in the GKE cluster, use namespace-aware kubectl commands. To view all namespaces, use `kubectl get pods --all-namespaces`.
3. Run `pip freeze`.

For example:

```
oud container clusters get-credentials projects/composer-test-1/zones/us-central1-f/
ing cluster endpoint and auth data.
onfig entry generated for us-central1-quickstart-f5da909c-gke.
```

```
composer-test-1)$ kubectl exec -itn composer-1-7-2-airflow-1-9-0-0a9f265b airflow-wo

mposer-test-1)$ pip freeze

py==0.7.1
=1.2.0
rypto==0.24.0
==0.8.0
==19.1.0
ep8==1.4.4
```

Connecting to a private Cloud Composer environment (/composer/docs/concepts/private-ip#cluster) might r
onal setup, depending on whether the Cloud Composer-managed GKE cluster permits external access.

Installing a Python dependency from PyPi

Your Python dependency must not have external dependencies or conflict with Cloud Composer's dependencies to install Python dependencies from the Python Package Index (https://pypi.org).

To add, update, or delete the Python dependencies for your environment:

Consolegcloud (#gcloud)rest (#rest)

Specify the package name and version specifiers as shown:

- `"pi-python-client", "==1.1.post1"`
- `"go-api-python-client", "==1.0.0.dev187"`

For a package without the version specifier, use an empty string for the value, such as `"glob2", ""`.

To access an environment's Python dependencies, navigate to the **PyPi dependencies** page using the following steps:

1. Open the **Environments** page in the Google Cloud Platform Console.

   Open the Environments page (https://console.cloud.google.com/composer/environments)

2. Click the **Name** of the environment you want to install, update, or delete Python dependencies for.
3. Select the **PyPi dependencies** tab.
4. Click the **Edit** button.
5. To add a new dependency:
   a. Click the **Add dependency** button.
   b. Enter the name and version of your library in the **Name** and **Version** fields.
6. To update an existing dependency:

a. Select the **Name** and/or **Version** field of the library you want to update.

b. Enter a new value.

7. To delete a dependency:

a. Hover over the name of the dependency to delete.

b. Click the trash can icon that appears.

Installing a Python dependency from a private repository

You can install packages hosted in private package repositories available on the public internet. The packages must be properly configured packages that the default pip (https://pip.pypa.io/en/stable/user_guide/#config-file) tool can install.

Cloud Composer does not support pip customization or resolve package dependencies and conflicts outside c
tion mechanisms that the default pip tool provides.

To install from a private package repository with a public address:

1. Create a pip.conf (https://pip.pypa.io/en/stable/user_guide/#config-file) file and include the following information in the file if applicable:

   - Access credentials for the repository
   - Non-default pip installation options

   Example:

```
[global]
extra-index-url=https://my-example-private-repo.com/
```

2. Upload the pip.conf file to your environment's Cloud Storage bucket (/composer/docs/concepts/cloud-storage) and place it in the folder `/config/pip/`, for example: *gs://us-central1-b1-6efannnn-bucket/config/pip/pip.conf*

Installing a Python dependency to a private IP environment

A private IP environment restricts access to the public internet, so installing Python dependencies may require additional steps.

When installing dependencies from a public PyPI repository, no special configuration is required. You can follow the normal process described above (#install-package). You can also request packages from a private repository with a public address (#install-private).

Alternatively, you can host a private PyPI repository in your VPC network. When installing dependencies, Cloud Composer will run the operation within the private IP GKE cluster hosting your environment, without accessing any public IP address through Cloud Build.

To install packages from a private repository hosted in your VPC network:

1. If the service account for your Cloud Composer environment does not have the `project.editor` role, grant it the `iam.serviceAccountUser` role.
2. Specify the private IP address of the repository in the `pip.conf` file uploaded to the `/config/pip/` folder in the Cloud Storage bucket.

Installing a Python dependency to a private IP environment in a VPC Service Controls perimeter

Protecting your project with a VPC Service Controls perimeter (/composer/docs/concepts/features#vpc-service-controls) results in further security restrictions. In particular, Cloud Build cannot be used for package installation, preventing direct access to repositories on the public internet.

To install Python dependencies for a private IP Composer environment inside a perimeter, you have some options:

1. Use a private PyPI repository hosted in your VPC network (as described in the section above (#install-private-ip)).
2. Use a proxy server (https://pip.pypa.io/en/stable/user_guide/#using-a-proxy-server%20VM) in your VPC network to connect to a PyPI repository on the public internet. Specify the proxy address in the `/config/pip/pip.conf` file in the Cloud Storage bucket.

3. If your security policy permits access to your VPC network from external IP addresses, you can enable this by configuring Cloud NAT (/nat/docs/overview).
4. Vendor the Python dependencies into the `dags` folder in the Cloud Storage bucket to install them as local libraries (#install-local). This may not be a good option if the dependency tree is large.

Installing a local Python library

To install an in-house or local Python library:

1. Place the dependencies within a subdirectory in the `dags/` folder. To import a module from a subdirectory, each subdirectory in the module's path must contain a `__init__.py` package marker file.

   In this example, the dependency is `coin_module.py`:

```
dags/
  use_local_deps.py  # A DAG file.
  dependencies/
    __init__.py
    coin_module.py
```

2. Import the dependency from the DAG definition file.

   For example:

s-samples/blob/c5635d1146fc2c0ff284c41d4b2d1132b25ae270/composer/workflows/use_local_deps.py)

```
from dependencies import coin_module
```

An import error occurs if an `__init__.py` file is missing. Directory and file names must be valid Python identif

Using Python packages that depend on shared object libraries

Certain PyPI packages depend on system-level libraries. While Cloud Composer does not support system libraries, you can use the following options:

1. Use the KubernetesPodOperator (/composer/docs/how-to/using/using-kubernetes-pod-operator). Set the Operator image to a custom build image. If you experience packages that fail during installation due to an unmet system dependency, use this option.
2. Upload the shared object libraries to your environment's Cloud Storage bucket.
   a. Manually find the shared object libraries for the PyPI dependency (an .so file).
   b. Upload the shared object libraries to `/home/airflow/gcs/plugins`.
   c. Set the following Cloud Composer environment variable: `LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/airflow/gcs/plugins`

   If your PyPI packages have installed successfully but fail at runtime, this is an option.

Files in the **`plugins/`** folders are synced to the local file system (/composer/docs/concepts/cloud-storage).
ding large .so files can affect the performance of your environment and the Airflow web server
poser/docs/how-to/accessing/airflow-web-interface).

Connecting to the Flower web interface

Flower (http://flower.readthedocs.io/en/latest/) is a web-based tool for working with Celery clusters. Flower is pre-installed in your environment. You can use its web UI to monitor the Apache Airflow workers for your environment.

To access Flower:

1. To determine the Kubernetes Engine cluster, view your environment:

```
gcloud composer environments describe ENVIRONMENT-NAME /
    --location LOCATION
```

The cluster is listed as the `gkeCluster`. The zone where the cluster is deployed is listed as the `location`.

For example:

```
gcloud composer environments describe environment-name --location us-cent
config:
  airflowUri: https://uNNNNe0aNNbcd3fff-tp.appspot.com
  dagGcsPrefix: gs://us-central1-may18-test-00a47695-bucket/dags
  gkeCluster: projects/example-project/zones/us-central1-a/clusters/us-ce
  nodeConfig:
    diskSizeGb: 100
    location: projects/example-project/zones/us-central1-a
```

In the example, the cluster is `us-central1-environment-name-00a47695-gke`, and the zone is `us-central1-a`. This information is also available on the Environment details (/composer/docs/how-to/managing/updating#details) page in the Cloud Console.

2. Connect to the Kubernetes Engine cluster:

```
gcloud container clusters get-credentials CLUSTER_NAME /
    --zone CLUSTER_ZONE
```

For example:

```
gcloud container clusters get-credentials us-central1-environment-name-00a47695

Fetching cluster endpoint and auth data.
kubeconfig entry generated for us-central1-environment-name-00a47695-gke.
```

3. View the worker pods and select the pod to run Flower on:

```
kubectl get pods --all-namespaces | grep worker
```

For example:

```
kubectl get pods --all-namespaces | grep worker

airflow-worker-89696c45f-49rkb      2/2       Running   1          29d
airflow-worker-89696c45f-gccmm      2/2       Running   1          29d
airflow-worker-89696c45f-llnnx      2/2       Running   0          29d
```

The pod names match the regex `"airflow-(worker|scheduler)-[-a-f0-9]+"`.

4. Run Flower on the worker pod:

```
kubectl exec -n NAMESPACE -it POD_NAME -c airflow-worker -- airflow flower
```

★ **Note:** The parameters for `airflow flower` are read automatically from your environment's Airflow configuration. For parameter details, see the Airflow documentation (https://airflow.apache.org/docs/stable/cli-ref#flower).

For example:

```
kubectl exec -n composer-1-6-0-airflow-1-10-1-9670c487 -it airflow-worker-89696
    -c airflow-worker -- airflow flower

[I 180601 20:35:55 command:139] Visit me at http://0.0.0.0:5555
[I 180601 20:35:55 command:144] Broker: redis://airflow-redis-service.default.s
```

5. In a separate terminal session, use `kubectl` to forward a port on your local machine to the pod running the Flower UI:

```
kubectl -n NAMESPACE port-forward POD_NAME 5555
```

For example:

```
kubectl -n composer-1-6-0-airflow-1-10-1-9670c487 port-forward airflow-worker-c
```

```
Forwarding from 127.0.0.1:5555 -> 5555
```

6. To access the web UI, go to `http://localhost:5555` in your local browser.

Installing SQLAlchemy to access the Airflow database

SQLAlchemy (https://www.sqlalchemy.org/) is a Python SQL toolkit and Object Relational Mapper (ORM). You can install SQLAlchemy and use it to access the Cloud SQL instance for Cloud Composer. During installation, Cloud Composer configures the Airflow environment variable `AIRFLOW__CORE__SQL_ALCHEMY_CONN`.

To install SQL Alchemy:

1. Install `sqlalchemy` in your environment.

```
gcloud composer environments update ENVIRONMENT-NAME /
    --location LOCATION /
    --update-pypi-package "sqlalchemy"
```

2. To determine the Kubernetes Engine cluster, view your environment:

```
gcloud composer environments describe ENVIRONMENT-NAME /
    --location LOCATION
```

3. Connect to the Kubernetes Engine cluster:

```
gcloud container clusters get-credentials CLUSTER_NAME /
    --zone CLUSTER_LOCATION
```

4. View the worker pods and select the pod to connect to:

```
kubectl get pods --all-namespaces | grep worker
```

5. SSH to the worker pod:

   ```
   kubectl -n NAMESPACE exec -it POD_NAME -c airflow-worker -- /bin/bash
   ```

   For example:

   ```
   kubectl -n composer-1-6-0-airflow-1-10-1-9670c487 /
       exec -it airflow-worker-54c6b57789-66pnr -c airflow-worker -- /bin/bash
   airflow@airflow-worker-54c6b57789-66pnr:~$
   ```

6. Use the `sqlalchemy` library to interact with the Airflow database:

   ```
   python
   import airflow.configuration as config
   config.conf.get('core', 'sql_alchemy_conn')
   ```