Google Cloud

AutoMLOps

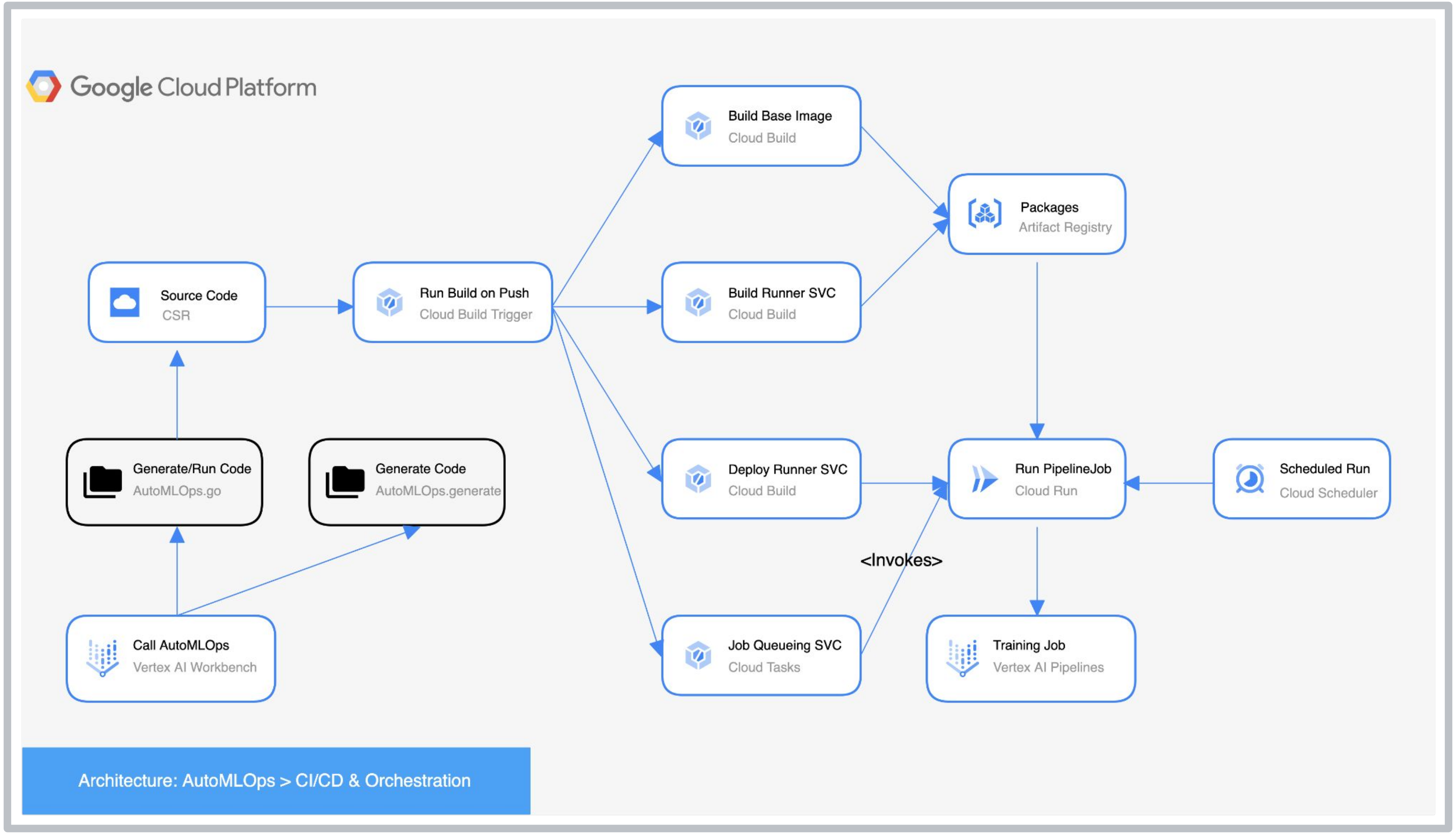
From Notebooks to Pipelines in Minutes

Implementation Guide v1.0.1

srastatter@, tonydiloreto@, allegranoto@

February 2023

Solution Overview



Set Up

Google Cloud

Prerequisites / Assumptions

The prerequisites for use of AutoMLOps are as follows:

- Jupyter (or Jupyter-compatible) notebook environment
- Notebooks API is enabled
- Python version ≥3.0 and ≤3.10
- Google Cloud SDK 407.0.0
- gcloud beta 2022.10.21
- Terraform is installed
- git is installed and logged-in

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

- Application Default Credentials (ADC) are set up, which can be done through the following commands:

```
gcloud auth application-default login
gcloud config set account <account@example.com>
```

Set Up AutoMLOps Package

1. Clone the repo:

```
git clone https://github.com/GoogleCloudPlatform/AutoMLOps.git
```

2. Install the AutoMLOps package:

```
pip install dist/AutoMLOps-1.0.1-py2.py3-none-any.whl
```

3. Open a notebook and import the AutoMLOps package:

```
from AutoMLOps import AutoMLOps
```

4. Decide whether to use Kubeflow definitions or Python definitions

Using Python Components (no Kubeflow)

Google Cloud

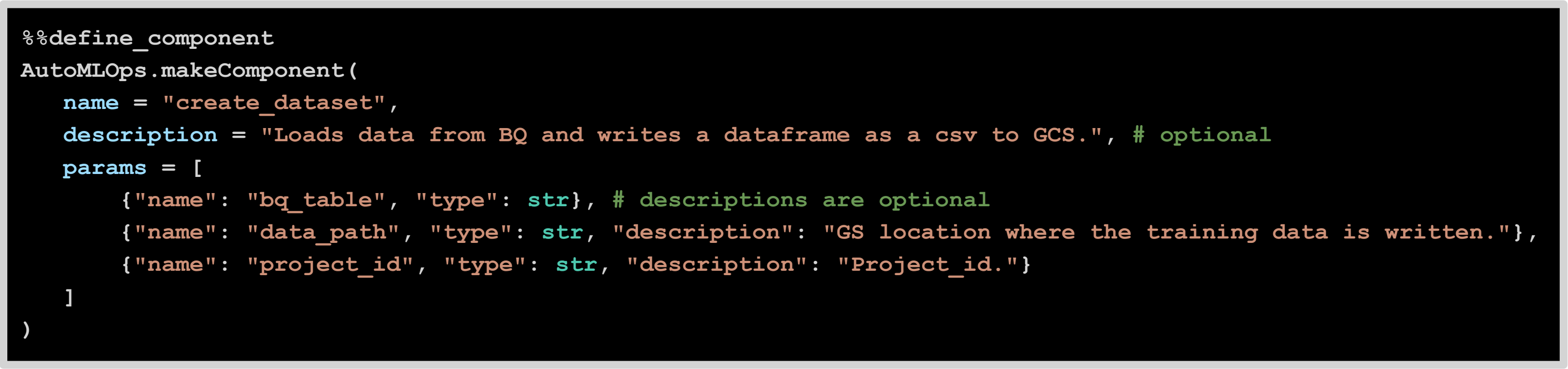
AutoMLOps Python Process

1. Define import code cells

```
%%define_imports
import json
import pandas as pd
from google.cloud import aiplatform
from google.cloud import aiplatform_v1
from google.cloud import bigquery
...
```

AutoMLOps Python Process

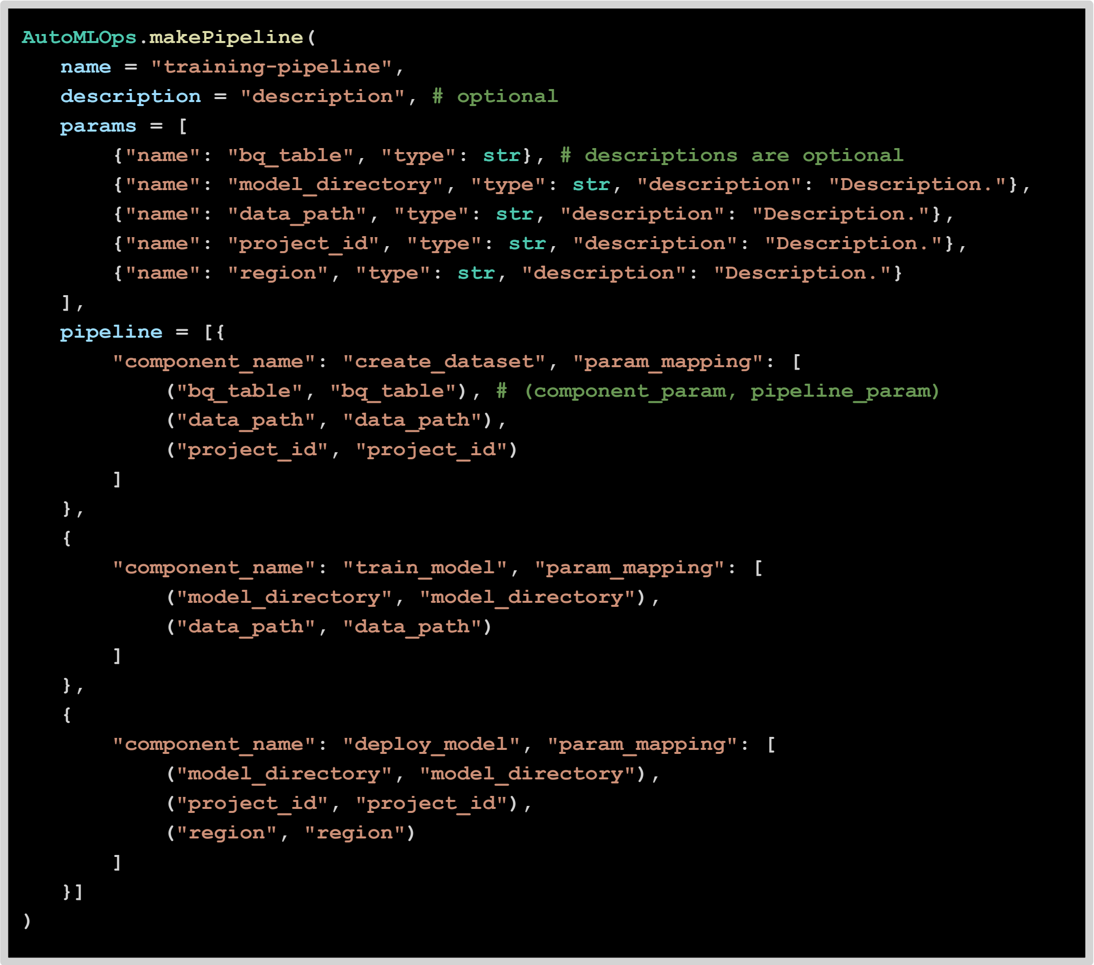
2. Define component code cells



```
%%define_component
AutoMLOps.makeComponent(
   name = "create_dataset",
   description = "Loads data from BQ and writes a dataframe as a csv to GCS.", # optional
   params = [
       {"name": "bq_table", "type": str}, # descriptions are optional
       {"name": "data_path", "type": str, "description": "GS location where the training data is written."},
       {"name": "project_id", "type": str, "description": "Project_id."}
   ]
)
```

AutoMLOps Python Process

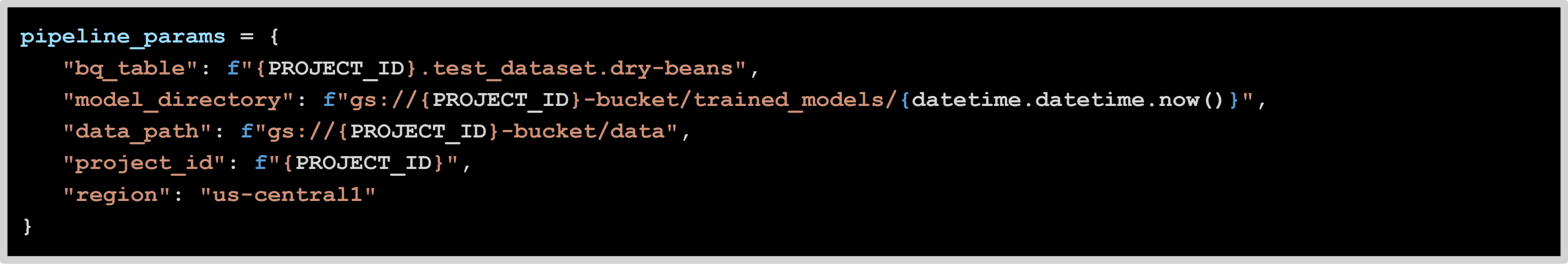
3. Define pipeline code cell



```
AutoMLOps.makePipeline(
    name = "training-pipeline",
    description = "description", # optional
    params = [
        {"name": "bq_table", "type": str}, # descriptions are optional
        {"name": "model_directory", "type": str, "description": "Description."},
        {"name": "data_path", "type": str, "description": "Description."},
        {"name": "project_id", "type": str, "description": "Description."},
        {"name": "region", "type": str, "description": "Description."}
    ],
    pipeline = [{
        "component_name": "create_dataset", "param_mapping": [
            ("bq_table", "bq_table"), # (component_param, pipeline_param)
            ("data_path", "data_path"),
            ("project_id", "project_id")
        ]
    },
    {
        "component_name": "train_model", "param_mapping": [
            ("model_directory", "model_directory"),
            ("data_path", "data_path")
        ]
    },
    {
        "component_name": "deploy_model", "param_mapping": [
            ("model_directory", "model_directory"),
            ("project_id", "project_id"),
            ("region", "region")
        ]
    }]
)
```

AutoMLOps Python Process

4. Define the pipeline parameters dictionary

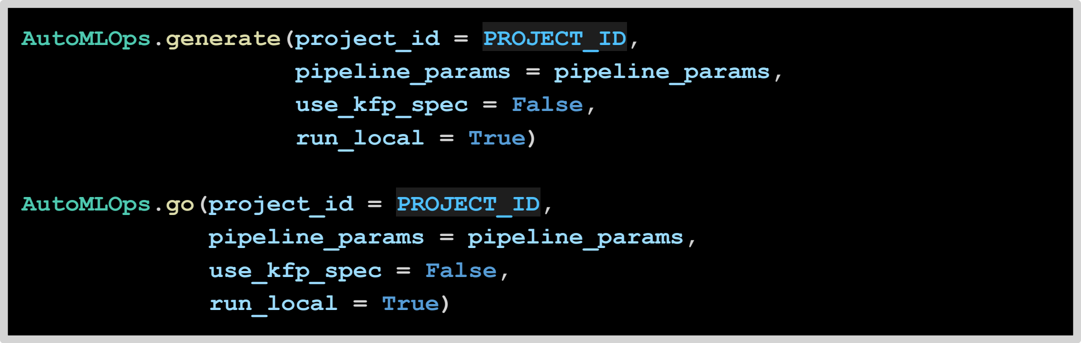


```
pipeline_params = {
   "bq_table": f"{PROJECT_ID}.test_dataset.dry-beans",
   "model_directory": f"gs://{PROJECT_ID}-bucket/trained_models/{datetime.datetime.now()}",
   "data_path": f"gs://{PROJECT_ID}-bucket/data",
   "project_id": f"{PROJECT_ID}",
   "region": "us-central1"
}
```

AutoMLOps Python Process

5. Call ***AutoMLOps.generate()*** to create the resources and repository

   Or ***AutoMLOps.go()*** to call generate in addition to building/submitting the pipeline job



```
AutoMLOps.generate(project_id = PROJECT_ID,
                   pipeline_params = pipeline_params,
                   use_kfp_spec = False,
                   run_local = True)

AutoMLOps.go(project_id = PROJECT_ID,
             pipeline_params = pipeline_params,
             use_kfp_spec = False,
             run_local = True)
```

Set ***use_kfp_spec=False*** when using an AutoMLOps Python defined pipeline

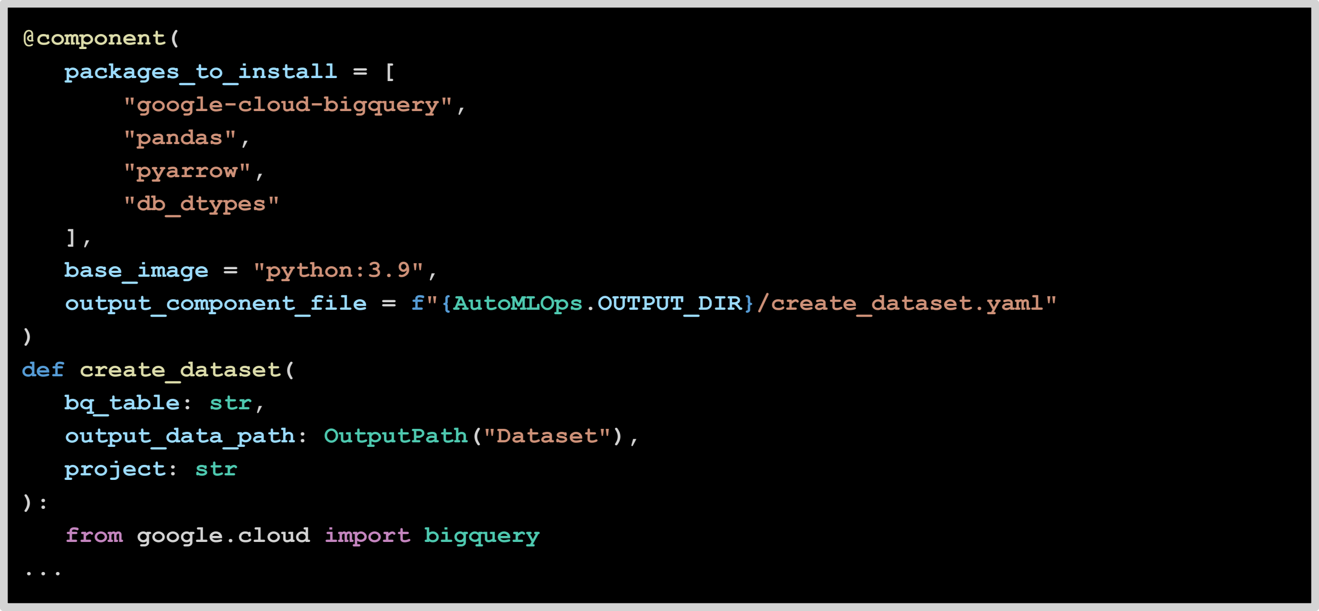
Set ***run_local=False*** if you want to generate and use CI/CD features

Using Kubeflow Components

Google Cloud

AutoMLOps Kubeflow Process

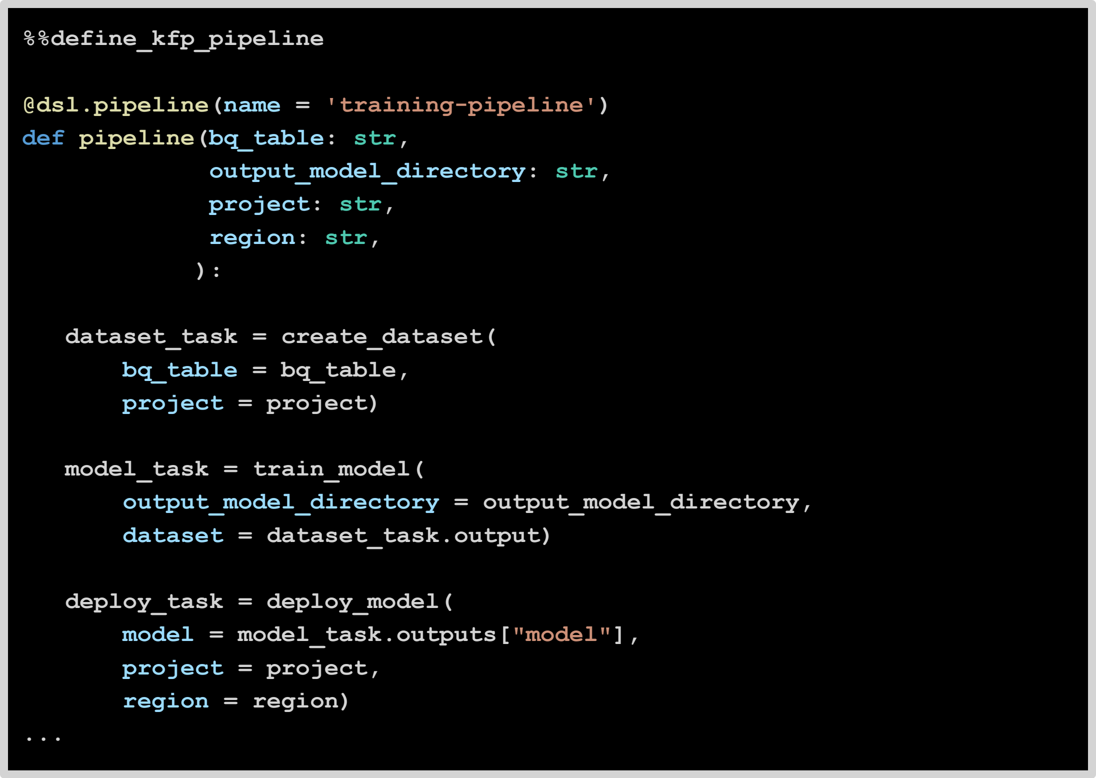
1. Define your components using KFP



```
@component(
   packages_to_install = [
       "google-cloud-bigquery",
       "pandas",
       "pyarrow",
       "db_dtypes"
   ],
   base_image = "python:3.9",
   output_component_file = f"{AutoMLOps.OUTPUT_DIR}/create_dataset.yaml"
)
def create_dataset(
   bq_table: str,
   output_data_path: OutputPath("Dataset"),
   project: str
):
   from google.cloud import bigquery
...
```

AutoMLOps Kubeflow Process

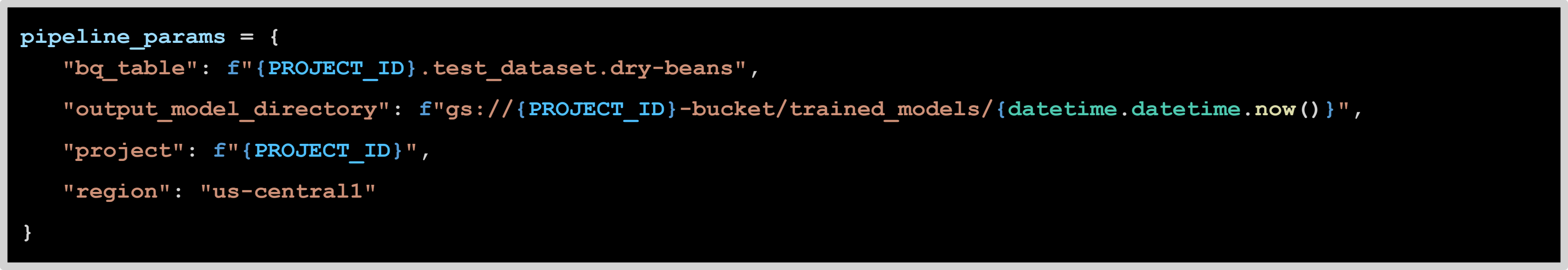
2. If using Kubeflow defs, define your pipeline using KFP and AutoMLOps:



```
%%define_kfp_pipeline

@dsl.pipeline(name = 'training-pipeline')
def pipeline(bq_table: str,
             output_model_directory: str,
             project: str,
             region: str,
            ):

   dataset_task = create_dataset(
       bq_table = bq_table,
       project = project)

   model_task = train_model(
       output_model_directory = output_model_directory,
       dataset = dataset_task.output)

   deploy_task = deploy_model(
       model = model_task.outputs["model"],
       project = project,
       region = region)
...
```

AutoMLOps Kubeflow Process

3. Define the pipeline parameters dictionary

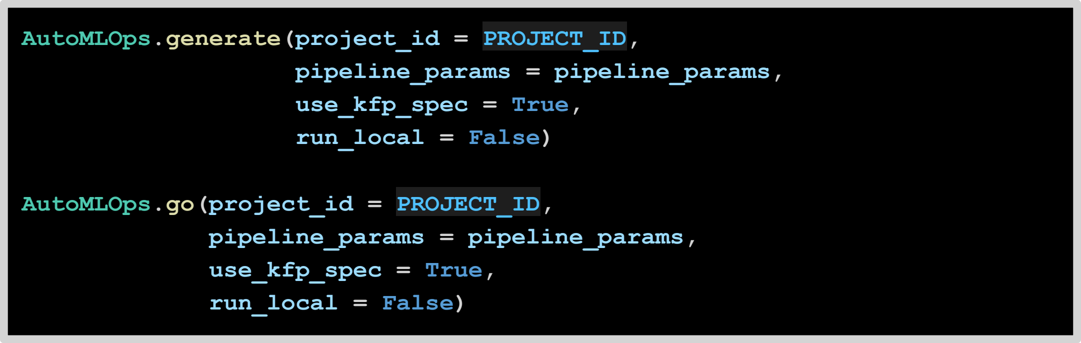


```
pipeline_params = {
   "bq_table": f"{PROJECT_ID}.test_dataset.dry-beans",
   "output_model_directory": f"gs://{PROJECT_ID}-bucket/trained_models/{datetime.datetime.now()}",
   "project": f"{PROJECT_ID}",
   "region": "us-central1"
}
```

AutoMLOps Kubeflow Process

4. Call ***`AutoMLOps.generate()`*** to create the resources and repository

   Or ***`AutoMLOps.go()`*** to call generate in addition to building/submitting the pipeline job



```
AutoMLOps.generate(project_id = PROJECT_ID,
                   pipeline_params = pipeline_params,
                   use_kfp_spec = True,
                   run_local = False)

AutoMLOps.go(project_id = PROJECT_ID,
             pipeline_params = pipeline_params,
             use_kfp_spec = True,
             run_local = False)
```

Set ***`use_kfp_spec=True`*** when using a Kubeflow defined pipeline

Set ***`run_local=False`*** if you want to generate and use CI/CD features

Behind the Scenes

Cloud Resources

When the pipeline is run, the following resources are created to complete the MLOps pipeline:

1. AutoMLOps codebase
2. Artifact Registry
3. GS Bucket
4. Pipeline Runner Service Account
5. Cloud Source Repository (turns the notebooks working directory into a CSR)
6. Cloud Build Trigger
7. Cloud Runner Service
8. Cloud Scheduler
9. Cloud Tasks queue

APIs

When the pipeline is run, the following APIs are enabled:

1. cloudresourcemanager.googleapis.com
2. aiplatform.googleapis.com
3. artifactregistry.googleapis.com
4. cloudbuild.googleapis.com
5. cloudscheduler.googleapis.com
6. cloudtasks.googleapis.com
7. compute.googleapis.com
8. iam.googleapis.com
9. iamcredentials.googleapis.com
10. ml.googleapis.com
11. run.googleapis.com
12. storage.googleapis.com
13. sourcerepo.googleapis.com

IAM Access

When the pipeline is run, the following IAM access roles are updated:

1. **Pipeline Runner Service Account** (created if it does exist, defaults to: *vertex-pipelines@<PROJECT_ID>.iam.gserviceaccount.com*).

   Roles added:

   - roles/aiplatform.user
   - roles/artifactregistry.reader
   - roles/bigquery.user
   - roles/bigquery.dataEditor
   - roles/iam.serviceAccountUser
   - roles/storage.admin
   - roles/run.admin

2. **Cloudbuild Default Service Account** (*<PROJECT_NUMBER>@cloudbuild.gserviceaccount.com*).

   Roles added:

   - roles/run.admin
   - roles/iam.serviceAccountUser
   - roles/cloudtasks.enqueuer
   - roles/cloudscheduler.admin

Package Dependencies

When using AutoMLOps, the following package versions are used:

1. autoflake==2.0.0,
2. docopt==0.6.2,
3. ipython==7.34.0,
4. pipreqs==0.4.11,
5. pyflakes==3.0.1,
6. PyYAML==5.4.1,
7. yarg==0.1.9