

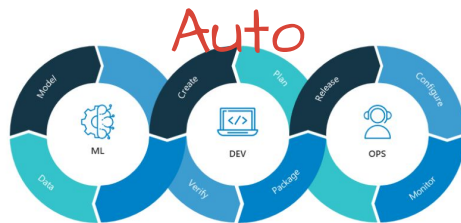


# AutoMLOps

*From Notebooks to Pipelines in Minutes*

## Implementation Guide

automlops@  
April 2023



The diagram illustrates the CI/CD and orchestration architecture for AutoML Ops on Google Cloud Platform. The workflow is as follows:

- Call AutoML Ops** (Vertex AI Workbench) initiates the process by calling **Generate/Run Code** (AutoML Ops.go) and **Generate Code** (AutoML Ops.generate).
- Generate/Run Code** and **Generate Code** both lead to **Source Code** (CSR).
- Source Code** is pushed to **Run Build on Push** (Cloud Build Trigger).
- Run Build on Push** triggers three parallel builds:
  - Build Base Image** (Cloud Build)
  - Build Runner SVC** (Cloud Build)
  - Deploy Runner SVC** (Cloud Build)
- Build Base Image** and **Build Runner SVC** are used to create **Packages** (Artifact Registry).
- Packages** are used to create **Run PipelineJob** (Cloud Run).
- Deploy Runner SVC** also triggers **Run PipelineJob**.
- Scheduled Run** (Cloud Scheduler) can trigger **Run PipelineJob**.
- Run PipelineJob** invokes **Job Queueing SVC** (Cloud Tasks), which then triggers a **Training Job** (Vertex AI Pipelines).

```
graph LR; Call[Call AutoML Ops  
Vertex AI Workbench] --> GenRun[Generate/Run Code  
AutoML Ops.go]; Call --> GenCode[Generate Code  
AutoML Ops.generate]; GenRun --> Source[Source Code  
CSR]; GenCode --> Source; Source --> RunBuild[Run Build on Push  
Cloud Build Trigger]; RunBuild --> BuildBase[Build Base Image  
Cloud Build]; RunBuild --> BuildRunner[Build Runner SVC  
Cloud Build]; RunBuild --> DeployRunner[Deploy Runner SVC  
Cloud Build]; BuildBase --> Packages[Packages  
Artifact Registry]; BuildRunner --> Packages; Packages --> RunPipeline[Run PipelineJob  
Cloud Run]; DeployRunner --> RunPipeline; Scheduled[Scheduled Run  
Cloud Scheduler] --> RunPipeline; RunPipeline -->|<Invokes>| JobQueue[Job Queueing SVC  
Cloud Tasks]; JobQueue --> Training[Training Job  
Vertex AI Pipelines];
```

Architecture: AutoML Ops > CI/CD & Orchestration

Set Up



# Prerequisites / Assumptions

*The prerequisites for use of AutoMLOps are as follows:*

- Python version  $\geq 3.0$  and  $\leq 3.10$
- [Google Cloud SDK 407.0.0](#)
- [gcloud beta 2022.10.21](#)
- 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. Install the AutoMLOps package:

```
pip install google-cloud-automlops
```

2. Import the AutoMLOps package:

```
from AutoMLOps import AutoMLOps
```

3. Decide whether to use Kubeflow definitions or Python definitions

## Using Python Components (no Kubeflow)



# AutoMLOps Python Definition

## 1. Define a component

```
@AutoMLOps.component
def create_dataset(
    bq_table: str,
    data_path: str,
    project_id: str
):
    """Custom component that takes in a BQ table and
        writes it to GCS.

    Args:
        bq_table: The source bigquery table.
        data_path: The gcs location to write the csv.
        project_id: The project ID.
    """
    from google.cloud import bigquery
    import pandas as pd
    ...
```

- Wrap your code into a function
- Provide input parameters and specify their types
- (Optionally) Specify a docstring with parameter descriptions
- Include required imports inside of the function
- Use @AutoMLOps.component decorator to specify a component. This will automatically containerize your code, creating a separate python file, dockerfile, requirements.txt, and a component specification yaml
- Repeat this process for each component

# AutoMLOps Python Definition

2. Define a pipeline
  - Define a function for your pipeline definition
    - Provide pipeline input parameters and specify their types
  - Chain together all components
    - Use `.after(...)` to specify the order of execution for the pipeline
  - Link the pipeline parameters to their matching component parameters
  - (Optionally) Provide a name and description for the pipeline
  - Use `@AutoMLOps.pipeline` decorator to specify the pipeline. This will automatically turn your function into a pipeline

```
@AutoMLOps.pipeline #(name='automlops-pipeline', description='This is an
optional description')
def pipeline(bq_table: str,
             model_directory: str,
             data_path: str,
             project_id: str,
             region: str,
             ):

    create_dataset_task = create_dataset(
        bq_table=bq_table,
        data_path=data_path,
        project_id=project_id)

    train_model_task = train_model(
        model_directory=model_directory,
        data_path=data_path).after(create_dataset_task)

    deploy_model_task = deploy_model(
        model_directory=model_directory,
        project_id=project_id,
        region=region).after(train_model_task)
```



# AutoMLOps Python Definition

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 Definition

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



# AutoMLOps Kubeflow Definition

1. Define your components using KFP

```
from kfp.v2 import dsl

@dsl.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 Definition

2. Define a pipeline
  - Define a function for your pipeline definition
    - Provide pipeline input parameters and specify their types
  - Chain together all components
    - Use `.after(...)` to specify the order of execution for the pipeline
  - Link the pipeline parameters to their matching component parameters
  - (Optionally) Provide a name and description for the pipeline
  - Use `@AutoMLOps.pipeline` decorator to specify the pipeline. This will automatically turn your function into a pipeline

```
@AutoMLOps.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

## Customizations





# AutoMLOps Defaults

There are a number of custom parameters that can be configured. To the left is a list of the default parameters:

```
AutoMLOps.go(project_id=PROJECT_ID, # required
               pipeline_params=pipeline_params, # required
               af_registry_location='us-central1', # default
               af_registry_name='vertex-mlops-af', # default
               cb_trigger_location='us-central1', # default
               cb_trigger_name='automlops-trigger', # default
               cloud_run_location='us-central1', # default
               cloud_run_name='run-pipeline', # default
               cloud_tasks_queue_location='us-central1', # default
               cloud_tasks_queue_name='queueing-svc', # default
               csr_branch_name='automlops', # default
               csr_name='AutoMLOps-repo', # default
               custom_training_job_specs=None, # default
               gs_bucket_location='us-central1', # default
               gs_bucket_name=None, # default
               pipeline_runner_sa=None, # default
               run_local=True, # default
               schedule_location='us-central1', # default
               schedule_name='AutoMLOps-schedule', # default
               schedule_pattern='No Schedule Specified', # default
               use_kfp_spec=False, # default
               vpc_connector='No VPC Specified' # default)
```

## AutoMLOps Defaults

- ``project_id``: The project ID.
- ``pipeline_params``: Dictionary containing runtime pipeline parameters.
- ``af_registry_location``: Region of the Artifact Registry.
- ``af_registry_name``: Artifact Registry name where components are stored.
- ``cb_trigger_location``: The location of the cloudbuild trigger.
- ``cb_trigger_name``: The name of the cloudbuild trigger.
- ``cloud_run_location``: The location of the cloud runner service.
- ``cloud_run_name``: The name of the cloud runner service.
- ``cloud_tasks_queue_location``: The location of the cloud tasks queue.
- ``cloud_tasks_queue_name``: The name of the cloud tasks queue.
- ``csr_branch_name``: The name of the csr branch to push to to trigger cb job.
- ``csr_name``: The name of the cloud source repo to use.
- ``custom_training_job_specs``: Specifies the specs to run the training job with.
- ``gs_bucket_location``: Region of the GS bucket.
- ``gs_bucket_name``: GS bucket name where pipeline run metadata is stored.
- ``pipeline_runner_sa``: Service Account to runner PipelineJobs.
- ``run_local``: Flag that determines whether to use Cloud Run CI/CD.
- ``schedule_location``: The location of the scheduler resource.
- ``schedule_name``: The name of the scheduler resource.
- ``schedule_pattern``: Cron formatted value used to create a Scheduled retrain job.
- ``use_kfp_spec``: Flag that specifies whether you are using Kubeflow spec or not.
- ``vpc_connector``: The name of the vpc connector to use.

# AutoMLOps Set Scheduled Run

Use the *schedule\_pattern* parameter to specify a cron job schedule to run the pipeline job on a recurring basis.

The *run\_local* must be set to *False* to make use of this feature.

```
AutoMLOps.generate(project_id = PROJECT_ID,  
                    pipeline_params = pipeline_params,  
                    use_kfp_spec = False,  
                    run_local = False,  
                    schedule_pattern = '0 */12 * * *')
```

The above example will rerun the pipeline every 12 hours.

# AutoMLOps Set Pipeline Compute Resources

Use the `custom_training_job_specs` parameter to specify resources for any custom component in the pipeline. The `component_spec` must match exactly the name of the custom component.

```
AutoMLOps.generate(project_id = PROJECT_ID,  
                    pipeline_params = pipeline_params,  
                    use_kfp_spec = False,  
                    run_local = False,  
                    custom_training_job_specs = [{  
                        'component_spec': 'train_model',  
                        'display_name': 'train-model-accelerated',  
                        'machine_type': 'a2-highgpu-1g',  
                        'accelerator_type': 'NVIDIA_TESLA_A100',  
                        'accelerator_count': '1'  
                    }])
```

The example above uses a GPU for accelerated training. See [Machine types](#) and [GPUs](#) for more info.

The `custom_training_job_specs` parameter takes in any key-value pair available under [google\\_cloud\\_pipeline\\_components.v1.custom\\_job.create\\_custom\\_training\\_job\\_op\\_from\\_component](#)

# AutoMLOps VPC Connector

Use the `vpc_connector` parameter to specify a vpc connector.

```
AutoMLOps.generate(project_id = PROJECT_ID,  
                    pipeline_params = pipeline_params,  
                    use_kfp_spec = False,  
                    run_local = False,  
                    vpc_connector = 'example-vpc')
```

# AutoMLOps Specify package versions

Use the `packages_to_install` parameter of `@AutoMLOps.component` to explicitly specify packages and versions. You wish to use for your component. AutoMLOps will infer these package requirements otherwise.

```
@AutoMLOps.component(  
    packages_to_install=[  
        "google-cloud-bigquery==2.34.4",  
        "pandas",  
        "pyarrow",  
        "db_dtypes"  
    ]  
)  
  
def create_dataset(  
    bq_table: str,  
    data_path: str,  
    project_id: str  
):  
    ...
```

## 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.enqueueur
- roles/cloudscheduler.admin

# Package Dependencies

When using AutoMLOps, the following package versions are used:

1. `docopt==0.6.2`,
2. `docstring-parser==0.15`,
3. `pipreqs==0.4.11`,
4. `PyYAML==5.4.1`,
5. `yarg==0.1.9`