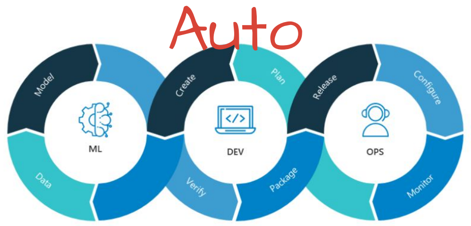
Google Cloud

AutoMLOps

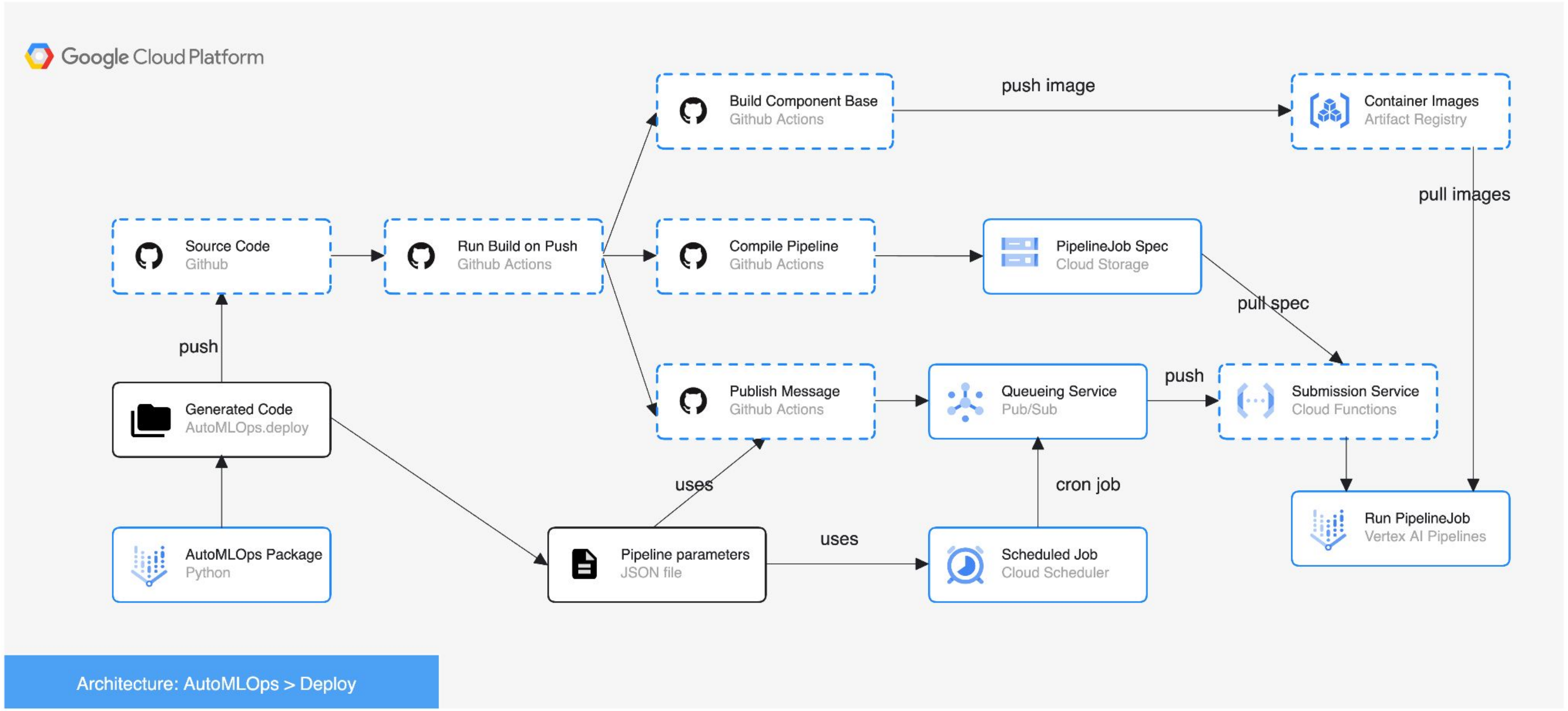
Build MLOps Pipelines in Minutes

User Guide



automlops@

Solution Overview



Prerequisites / Assumptions

Prerequisites / Assumptions

The prerequisites for use of **`AutoMLOps.generate()`** *are as follows:*

- Python version ≥3.7 and ≤3.10

The recommended configuration for use of **`AutoMLOps.provision()`** *with gcloud are as follows:*

- Google Cloud SDK 407.0.0
- gcloud beta 2022.10.21

The recommended configuration for use of **`AutoMLOps.provision()`** *with terraform are as follows:*

- Terraform v1.5.6

The prerequisites for use of **`AutoMLOps.deploy()`** *with use_ci=False are as follows:*

- Local python environment with these packages installed:
    - `kfp<2.0.0`
    - `google-cloud-aiplatform`
    - `google-cloud-pipeline-components`
    - `google-cloud-storage`
    - `pyyaml`

Prerequisites / Assumptions

The prerequisites for use of **`AutoMLOps.monitor()`** *are as follows:*

- Local python environment with these packages installed:
  - \`google-cloud-aiplatform\`
  - \`google-cloud-logging\`
  - \`google-cloud-storage\`
  - \`pyyaml\`

Prerequisites / Assumptions

The prerequisites for use of **`AutoMLOps.deploy()`** *with use_ci=True, as well as* **`AutoMLOps.monitor()`** *are as follows:*

- git is installed and logged-in

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

- Registered and setup your SSH key if you are using Github, Gitlab, or Bitbucket
- Application Default Credentials (ADC) are set up if you are using Cloud Source Repositories. This can be done through the following commands:

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

Supported Tools and Technologies

Support Tools and Technologies (as of v1.2.0)

Artifact Repositories: Stores component docker containers

- Artifact Registry

Deployment Frameworks: Builds component docker containers, compiles pipelines, and submits Pipeline Jobs

- Cloud Build
- Github Actions
- [coming soon] Gitlab CI
- [coming soon] Bitbucket Pipelines
- [coming soon] Jenkins

Orchestration Frameworks: Executes and orchestrates pipelines jobs

- Kubeflow Pipelines (KFP) - Runs on Vertex AI Pipelines
- [coming soon] Tensorflow Extended (TFX) - Runs on Vertex AI Pipelines
- [coming soon] Argo Workflows - Runs on GKE
- [coming soon] Airflow - Runs on Cloud Composer
- [coming soon] Ray - Runs on GKE

Support Tools and Technologies (as of v1.2.0)

Submission Service Compute Environments: RESTful service for submitting pipeline jobs to the orchestrator (e.g. Vertex AI, Cloud Composer, etc.)

- Cloud Functions
- Cloud Run

Provisioning Frameworks: Stands up necessary infra to run MLOps pipelines

- gcloud
- terraform
- [coming soon] pulumi

Source Code Repositories: Repository for versioning generated MLOps code

- Cloud Source Repositories
- Bitbucket
- Github
- Gitlab

Set Up

Google Cloud

Set Up AutoMLOps Package

1. Install the AutoMLOps package:

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

2. Import the AutoMLOps package:

```
from google_cloud_automlops import AutoMLOps
```

3. Decide whether to use AutoMLOps Python Definitions or AutoMLOps Kubeflow definitions

Using AutoMLOps Python Definitions

- This syntax allows for defining pipelines and components using just pure python, and does not require Kubeflow to be installed or knowledge of how to use Kubeflow
- Input parameters are limited to python primitives only (e.g. int, float, str, etc). Use the Kubeflow definitions if you need more complex input parameters (e.g. Output[Metrics])

Google Cloud

AutoMLOps Python Definitions

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 biquery 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
- Optionally use the ***`packages_to_install`*** parameter of ***`@AutoMLOps.component`*** to explicitly specify packages and versions.
- Repeat this process for each component

AutoMLOps Python Definitions

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)
```

Google Cloud

AutoMLOps Python Definitions

3. 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"
}
```

Using AutoMLOps Kubeflow Definitions

Google Cloud

AutoMLOps Kubeflow Definitions

1. Define your components using KFP

- 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 the kfp @dsl.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
- Specify the output_component_file to point to AutoMLOps.OUTPUT_DIR/<component_name>.yaml; this will allow AutoMLOps to find your component definition
- Do not use the base_image parameter, the base_image can be specified during the AutoMLOps.generate() step
- Repeat this process for each component

```
from kfp.v2 import dsl


@dsl.component(
   packages_to_install = [
       "google-cloud-bigquery",
       "pandas",
       "pyarrow",
       "db_dtypes"
   ],
   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 Definitions

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 Definitions

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"
}
```

Callable Functions

AutoMLOps Callable Functions

AutoMLOps provides 6 functions for building and maintaining MLOps pipelines:

- **`AutoMLOps.generate(...)`**: Generates the MLOps codebase. Users can specify the tooling and technologies they would like to use in their MLOps pipeline.
- **`AutoMLOps.provision(...)`**: Runs provisioning scripts to create and maintain necessary infra for MLOps.
- **`AutoMLOps.deprovision(...)`**: Runs deprovisioning scripts to tear down MLOps infra created using AutoMLOps.
- **`AutoMLOps.deploy(...)`**: Builds and pushes the component container, then triggers the pipeline job.
- **`AutoMLOps.launchAll(...)`**: Runs `generate()`, `provision()`, and `deploy()` all in succession.
- **`AutoMLOps.monitor(...)`**: Creates model monitoring jobs on deployed endpoints.

AutoMLOps Generate

AutoMLOps Generate

Use the generate function to write the MLOps codebase to the local filesystem. This function will create the relevant directories, pull out the code from the components and pipelines defined in the previous step, and write the code to files. All generated code will be placed under the generated AutoMLOps/ directory created under the current working directory.

The project_id and pipeline_params are required, the rest of the parameters are optional and AutoMLOps will provide defaults. Below is an example generate function call:

```
AutoMLOps.generate(project_id=PROJECT_ID,
       pipeline_params=pipeline_params,
       use_ci=True,
       naming_prefix='dry-beans-dt',
       provisioning_framework='terraform',
       schedule_pattern='59 11 * * 0' # retrain every Sunday at Midnight
)
```

Use naming_prefix to differentiate this pipeline from others you wish to run in your project; naming_prefix will give a prefix to the pipelines infra that is stood up during the provision() step. A list and description of all the available parameters can be found on the next slides.

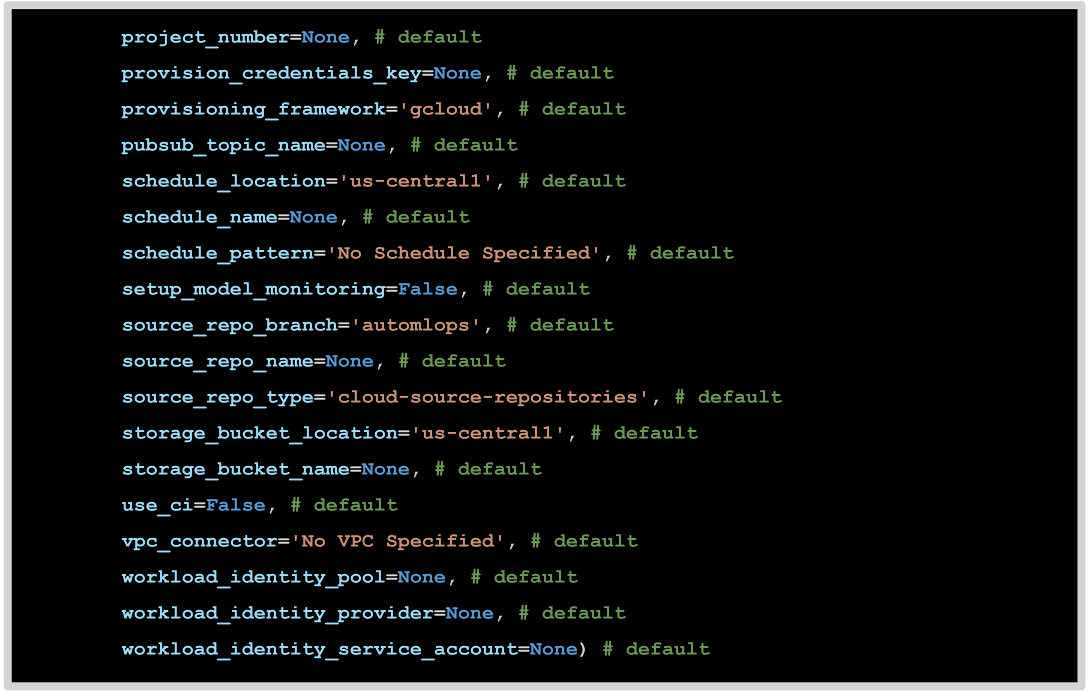
AutoMLOps Generate

There are a number of optional parameters that can be configured when running generate. To the right is a list of the parameters and their defaults:

```
AutoMLOps.generate(project_id=PROJECT_ID, # required
        pipeline_params=pipeline_params, # required
        artifact_repo_location='us-central1', # default
        artifact_repo_name=None, # default
        artifact_repo_type='artifact-registry', # default
        base_image='python:3.9-slim', # default
        build_trigger_location='us-central1', # default
        build_trigger_name=None, # default
        custom_training_job_specs=None, # default
        deployment_framework='cloud-build', # default
        naming_prefix='automlops-default-prefix', # default
        orchestration_framework='kfp', # default
        pipeline_job_runner_service_account=None, # default
        pipeline_job_submission_service_location='us-central1', # default
        pipeline_job_submission_service_name=None, # default
        pipeline_job_submission_service_type='cloud-functions', # default
```

AutoMLOps Generate (cont)

There are a number of optional parameters that can be configured when running generate. To the right is a list of the parameters and their defaults:



```
    project_number=None, # default
    provision_credentials_key=None, # default
    provisioning_framework='gcloud', # default
    pubsub_topic_name=None, # default
    schedule_location='us-central1', # default
    schedule_name=None, # default
    schedule_pattern='No Schedule Specified', # default
    setup_model_monitoring=False, # default
    source_repo_branch='automlops', # default
    source_repo_name=None, # default
    source_repo_type='cloud-source-repositories', # default
    storage_bucket_location='us-central1', # default
    storage_bucket_name=None, # default
    use_ci=False, # default
    vpc_connector='No VPC Specified', # default
    workload_identity_pool=None, # default
    workload_identity_provider=None, # default
    workload_identity_service_account=None) # default
```

Google Cloud

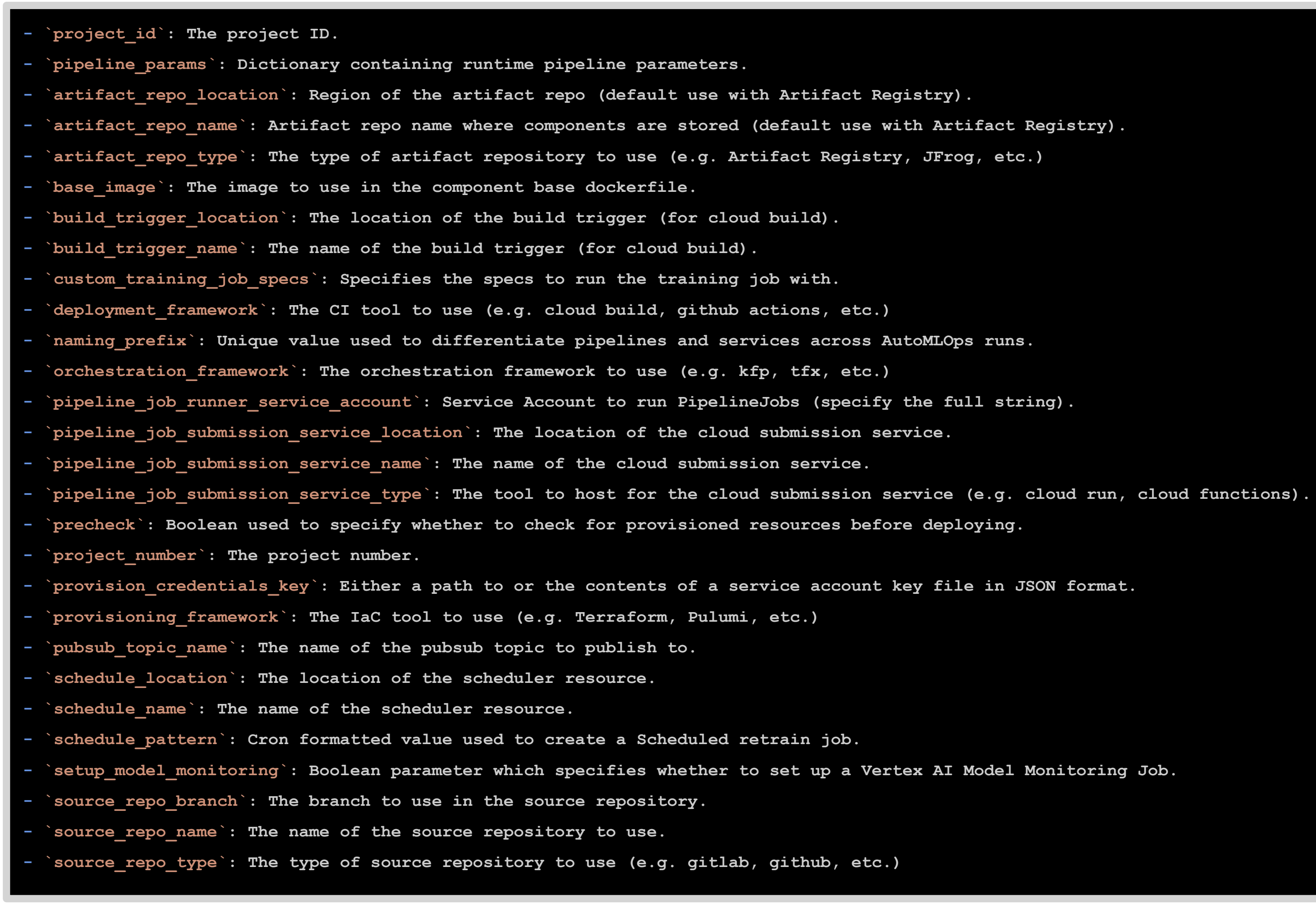
AutoMLOps Generate
Set Tools and Technologies

The generate function provides a number of optional parameters that allow you to change your tooling. These parameters can be set by choosing from a list of available strings for each optional parameter. The available options are shown to the right:

```
- `artifact_repo_type=`:
        - 'artifact-registry' (default)
- `deployment_framework=`:
        - 'cloud-build' (default)
        - 'github-actions'
        - [coming soon] 'gitlab-ci'
        - [coming soon] 'bitbucket-pipelines'
        - [coming soon] 'jenkins'
- `orchestration_framework=`:
        - 'kfp' (default)
        - [coming soon] 'tfx'
        - [coming soon] 'argo-workflows'
        - [coming soon] 'airflow'
        - [coming soon] 'ray'
- `pipeline_job_submission_service_type=`:
        - 'cloud-functions' (default)
        - 'cloud-run'
- `provisioning_framework=`:
        - 'gcloud' (default)
        - 'terraform'
        - [coming soon] 'pulumi'
- `source_repo_type=`:
        - 'cloud-source-repositories' (default)
        - 'github'
        - 'gitlab'
        - 'bitbucket'
```

AutoMLOps Parameter Descriptions

AutoMLOps will generate the resources specified by these parameters (e.g. Artifact Registry, Cloud Source Repo, etc.). If use_ci is set to True, the generated AutoMLOps/ will be turned into a git repo and used for the source repo. Additionally, if a cron formatted str is given as an arg for `schedule_pattern` then it will set up a Cloud Schedule to run accordingly.



```
- `project_id`: The project ID.
- `pipeline_params`: Dictionary containing runtime pipeline parameters.
- `artifact_repo_location`: Region of the artifact repo (default use with Artifact Registry).
- `artifact_repo_name`: Artifact repo name where components are stored (default use with Artifact Registry).
- `artifact_repo_type`: The type of artifact repository to use (e.g. Artifact Registry, JFrog, etc.)
- `base_image`: The image to use in the component base dockerfile.
- `build_trigger_location`: The location of the build trigger (for cloud build).
- `build_trigger_name`: The name of the build trigger (for cloud build).
- `custom_training_job_specs`: Specifies the specs to run the training job with.
- `deployment_framework`: The CI tool to use (e.g. cloud build, github actions, etc.)
- `naming_prefix`: Unique value used to differentiate pipelines and services across AutoMLOps runs.
- `orchestration_framework`: The orchestration framework to use (e.g. kfp, tfx, etc.)
- `pipeline_job_runner_service_account`: Service Account to run PipelineJobs (specify the full string).
- `pipeline_job_submission_service_location`: The location of the cloud submission service.
- `pipeline_job_submission_service_name`: The name of the cloud submission service.
- `pipeline_job_submission_service_type`: The tool to host for the cloud submission service (e.g. cloud run, cloud functions).
- `precheck`: Boolean used to specify whether to check for provisioned resources before deploying.
- `project_number`: The project number.
- `provision_credentials_key`: Either a path to or the contents of a service account key file in JSON format.
- `provisioning_framework`: The IaC tool to use (e.g. Terraform, Pulumi, etc.)
- `pubsub_topic_name`: The name of the pubsub topic to publish to.
- `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.
- `setup_model_monitoring`: Boolean parameter which specifies whether to set up a Vertex AI Model Monitoring Job.
- `source_repo_branch`: The branch to use in the source repository.
- `source_repo_name`: The name of the source repository to use.
- `source_repo_type`: The type of source repository to use (e.g. gitlab, github, etc.)
```

AutoMLOps Parameter Descriptions (cont)

```
- `storage_bucket_location`: Region of the GS bucket.
- `storage_bucket_name`: GS bucket name where pipeline run metadata is stored.
- `hide_warnings`: Boolean used to specify whether to show provision/deploy permission warnings
- `use_ci`: Flag that determines whether to use Cloud CI/CD.
- `vpc_connector`: The name of the vpc connector to use.
- `workload_identity_pool`: Pool for workload identity federation.
- `workload_identity_provider`: Provider for workload identity federation.
- `workload_identity_service_account`: Service account for workload identity federation (specify the full string).
```

AutoMLOps Generated Directories/ Files

```
.
├── components                                 : Custom vertex pipeline components.
    ├──component_base                          : Contains all the python files, Dockerfile and requirements.txt
        ├── Dockerfile                         : Dockerfile containing all the python files for the components.
        ├── requirements.txt                   : Package requirements for all the python files for the components.
        ├── src                                : Python source code directory.
            ├──component_a.py                  : Python file containing code for the component.
            ├──...(for each component)
    ├──component_a                             : Components specs generated using AutoMLOps
        ├── component.yaml                     : Component yaml spec, acts as an I/O wrapper around the Docker container.
    ├──...(for each component)
├── configs                                    : Configurations for defining vertex ai pipeline and MLOps infra.
    ├── defaults.yaml                          : Runtime configuration variables.
├── images                                     : Custom container images for training models (optional).
├── pipelines                                  : Vertex ai pipeline definitions.
    ├── pipeline.py                            : Full pipeline definition; compiles pipeline spec and uploads to GCS.
    ├── pipeline_runner.py                     : Sends a PipelineJob to Vertex AI.
    ├── requirements.txt                       : Package requirements for running pipeline.py.
    ├── runtime_parameters                     : Variables to be used in a PipelineJob.
        ├── pipeline_parameter_values.json     : Json containing pipeline parameters.
├── provision                                  : Provision configurations and details.
    ├── provision_resources.sh                 : Provisions the necessary infra to run the MLOps pipeline.
    ├── provisioning_configs                   : (Optional) Relevant terraform/Pulumi config files for provisioning infa.
├── scripts                                    : Scripts for manually triggering the cloud run service.
    ├── build_components.sh                    : Submits a Cloud Build job that builds and pushes the components to the registry.
    ├── build_pipeline_spec.sh                 : Compiles the pipeline specs.
    ├── run_pipeline.sh                        : Submit the PipelineJob to Vertex AI.
    ├── run_all.sh                             : Builds components, compiles pipeline specs, and submits the PipelineJob.
    ├── publish_to_topic.sh                    : Publishes a message to a Pub/Sub topic to invoke the pipeline job submission service.
    ├── create_model_monitoring_job.sh         : Creates or updated a Vertex AI model monitoring job for a given deployed model endpoint.
```

AutoMLOps Generated Directories/ Files (cont)

```
.
├── model_monitoring                                : Code for building and maintaining model monitoring jobs.
   ├── requirements.txt                            : Package requirements for creating and updating model monitoring jobs.
   ├── monitor.py                                  : Creates a ModelDeploymentMonitoringJob and optionally creates a Log Sink for automatic retraining.
├── services                                        : MLOps services related to continuous training.
    ├── submission_service                          : REST API service used to submit pipeline jobs to Vertex AI.
        ├── Dockerfile                              : Dockerfile for running the REST API service.
        ├── requirements.txt                        : Package requirements for the REST API service.
        ├── main.py                                 : Python REST API source code.
├── README.md                                       : Readme markdown file describing the contents of the generated directories.
└── Build config yaml                               : Build configuration file for building custom components.
```

AutoMLOps Provision

Google Cloud

AutoMLOps Provision

Use the provision function to provision the required infrastructure to support the MLOps pipeline. This function will run any IaC code found under the AutoMLOps/provision directory. Below is an example provision function call:

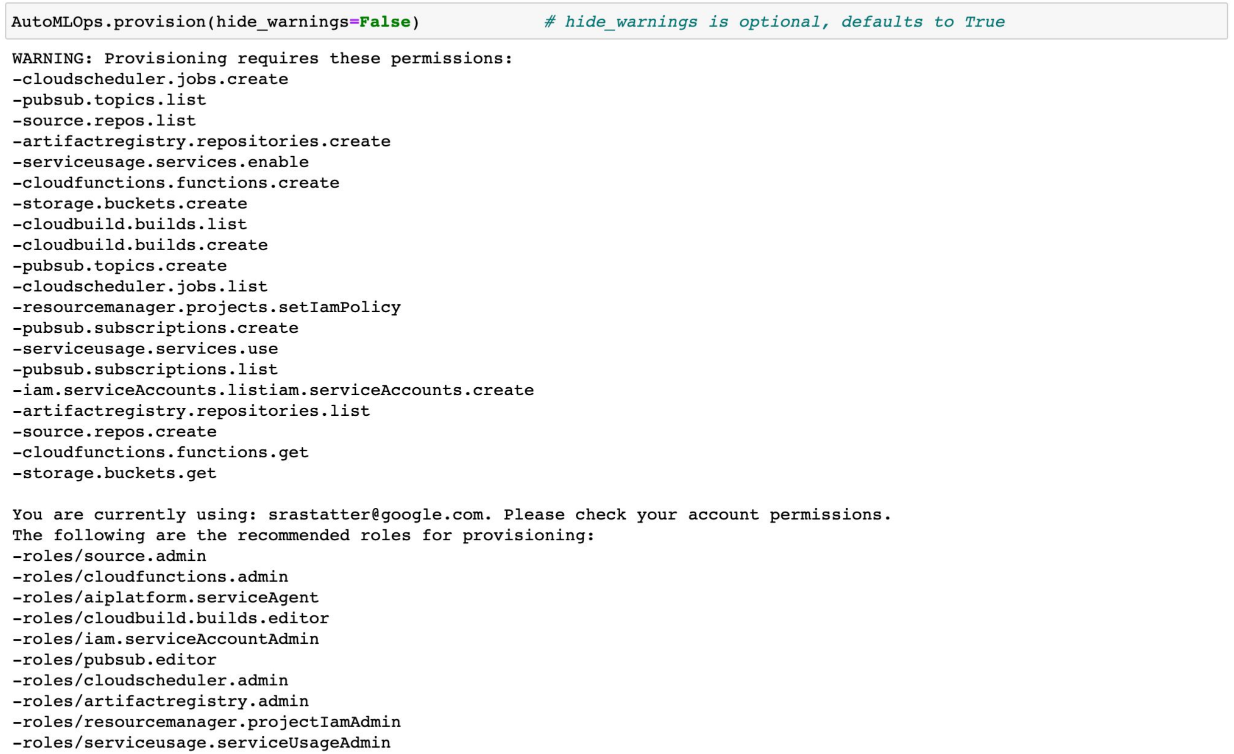
```
AutoMLOps.provision(hide_warnings=False) # hide_warnings is optional, defaults to True
```

Based on your selection for provision_framework during the generate function call (defaults to gcloud), the provision function will use the IaC code under AutoMLOps/provision.

If you select terraform for your provisioning_framework, AutoMLOps will first create a state_bucket in gcs to version your state file, and then use this bucket as the backend for your IaC configuration. This prevents loss of the state file.

AutoMLOps Provision

The hide_warnings parameter specifies whether to show permissions warnings before provisioning. If hide_warnings is set to False, it will show a warning specifying necessary permissions and recommended roles; an example run is shown below:



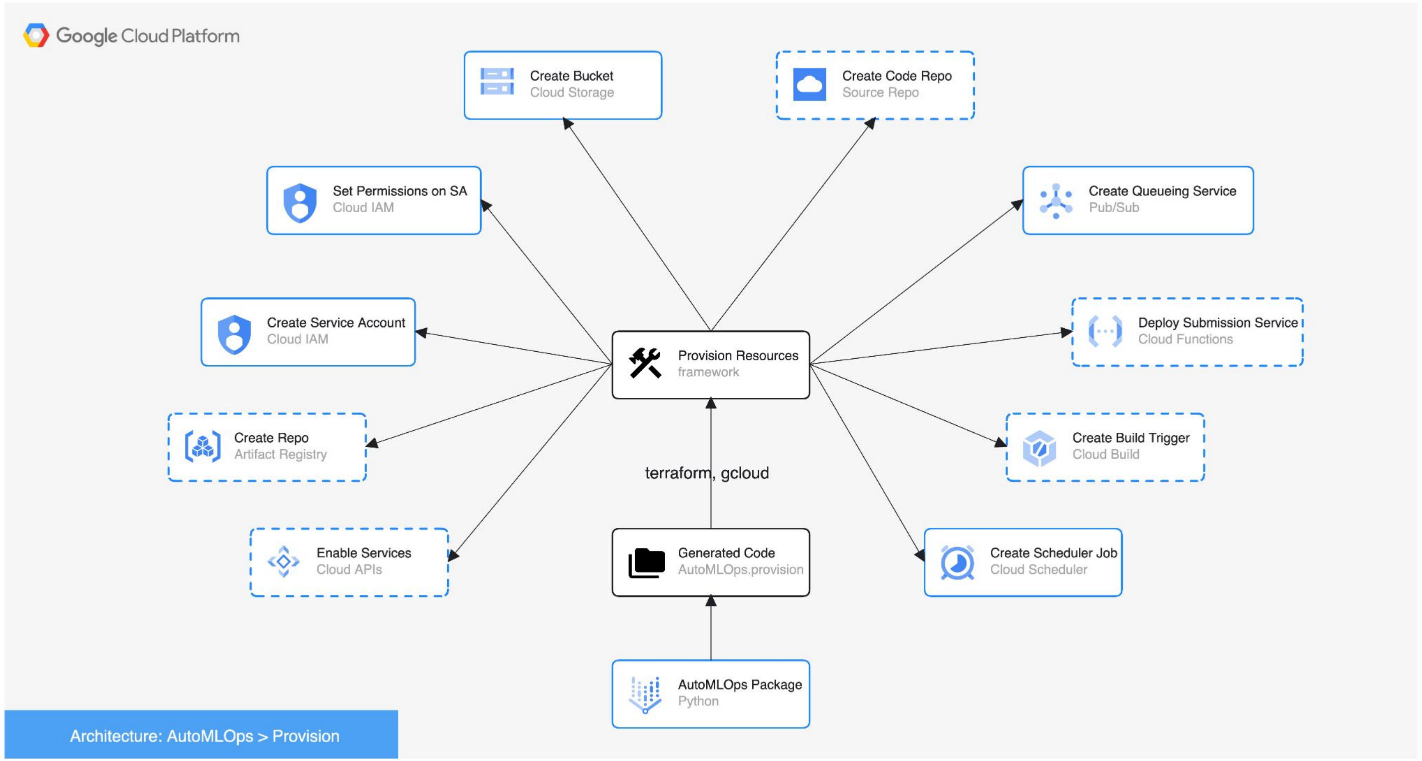
```python
AutoMLOps.provision(hide_warnings=False)            # hide_warnings is optional, defaults to True
```

```
WARNING: Provisioning requires these permissions:
-cloudscheduler.jobs.create
-pubsub.topics.list
-source.repos.list
-artifactregistry.repositories.create
-serviceusage.services.enable
-cloudfunctions.functions.create
-storage.buckets.create
-cloudbuild.builds.list
-cloudbuild.builds.create
-pubsub.topics.create
-cloudscheduler.jobs.list
-resourcemanager.projects.setIamPolicy
-pubsub.subscriptions.create
-serviceusage.services.use
-pubsub.subscriptions.list
-iam.serviceAccounts.listiam.serviceAccounts.create
-artifactregistry.repositories.list
-source.repos.create
-cloudfunctions.functions.get
-storage.buckets.get

You are currently using: srastatter@google.com. Please check your account permissions.
The following are the recommended roles for provisioning:
-roles/source.admin
-roles/cloudfunctions.admin
-roles/aiplatform.serviceAgent
-roles/cloudbuild.builds.editor
-roles/iam.serviceAccountAdmin
-roles/pubsub.editor
-roles/cloudscheduler.admin
-roles/artifactregistry.admin
-roles/resourcemanager.projectIamAdmin
-roles/serviceusage.serviceUsageAdmin
```

AutoMLOps Provision

AutoMLOps currently provides 2 primary options for provisioning infrastructure: `gcloud` and `terraform`. In the diagram below dashed boxes show areas users can select and customize their tooling.



AutoMLOps Deprovision

Google Cloud

AutoMLOps Deprovision

Use the deprovision function to deprovision the infrastructure created during the provision function call. This function will effectively run a terraform destroy operation using the code under AutoMLOps/provision. Below is an example provision function call:

```
AutoMLOps.deprovision()
```

The following provisioning_frameworks are currently supported by the deprovision function:

- terraform

The following provisioning_frameworks are currently not supported by the deprovision function:

- gcloud

AutoMLOps Deploy

AutoMLOps Deploy

Use the deploy function to trigger a PipelineJob. Calling deploy will build and push the component_base image, compile the pipeline, upload the compiled pipeline spec to GCS, and submit a message to the queueing service to execute a PipelineJob. The specifics of the deploy step are dependent on the defaults set during the generate step, particularly:

- **use_ci**: if use_ci is False, the deploy step will use scripts/run_all.sh, which will submit the build job, compile the pipeline, and submit the PipelineJob all from the local machine. If use_ci is True, it will use the CI/CD workflow shown on the next slide.
- **artifact_repo_type**: Determines which type of artifact repo the image is pushed to.
- **deployment_framework**: Determines which build tool to use for building.
- **source_repo_type**: Determines which source repo to use for versioning code and triggering the build.

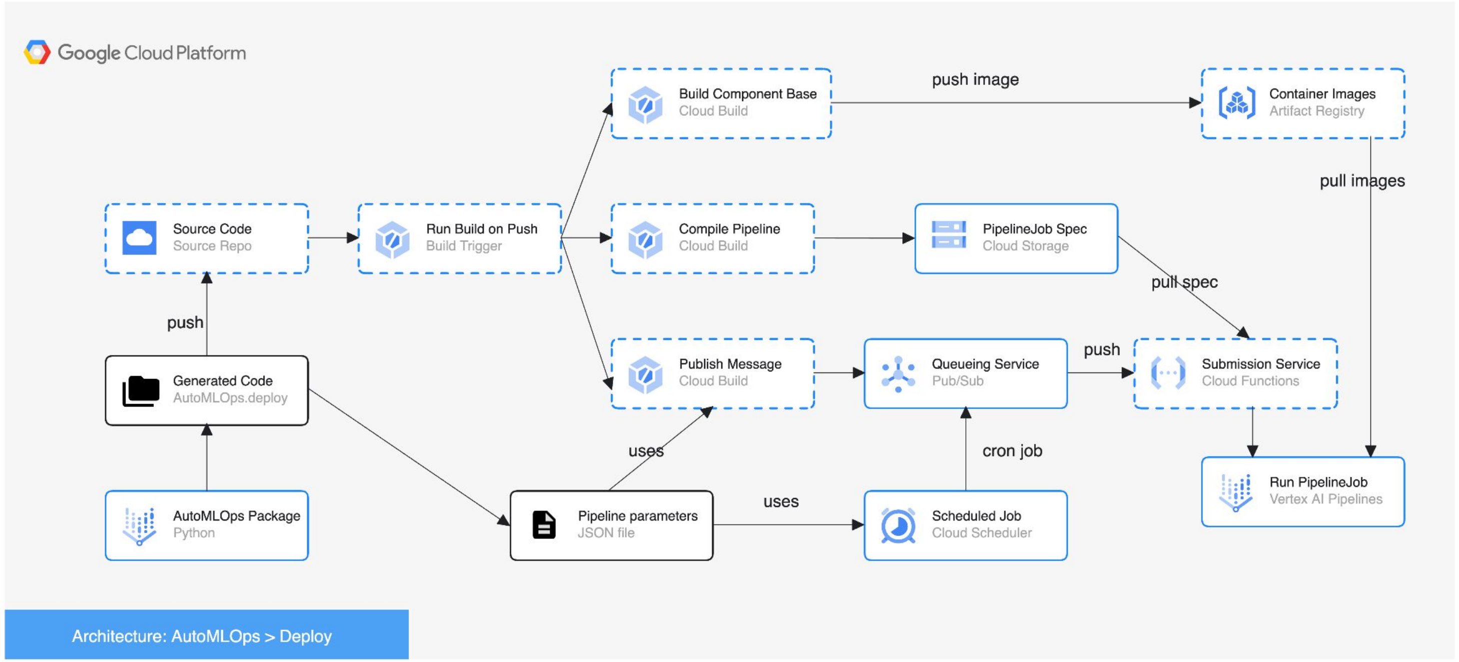
All defaults from the generate step are stored in config/defaults.yaml.

Below is an example deploy function call:

```
AutoMLOps.deploy(precheck=True, # precheck is optional, defaults to True
       hide_warnings=False) # hide_warnings is optional, defaults to True
```

AutoMLOps Deploy

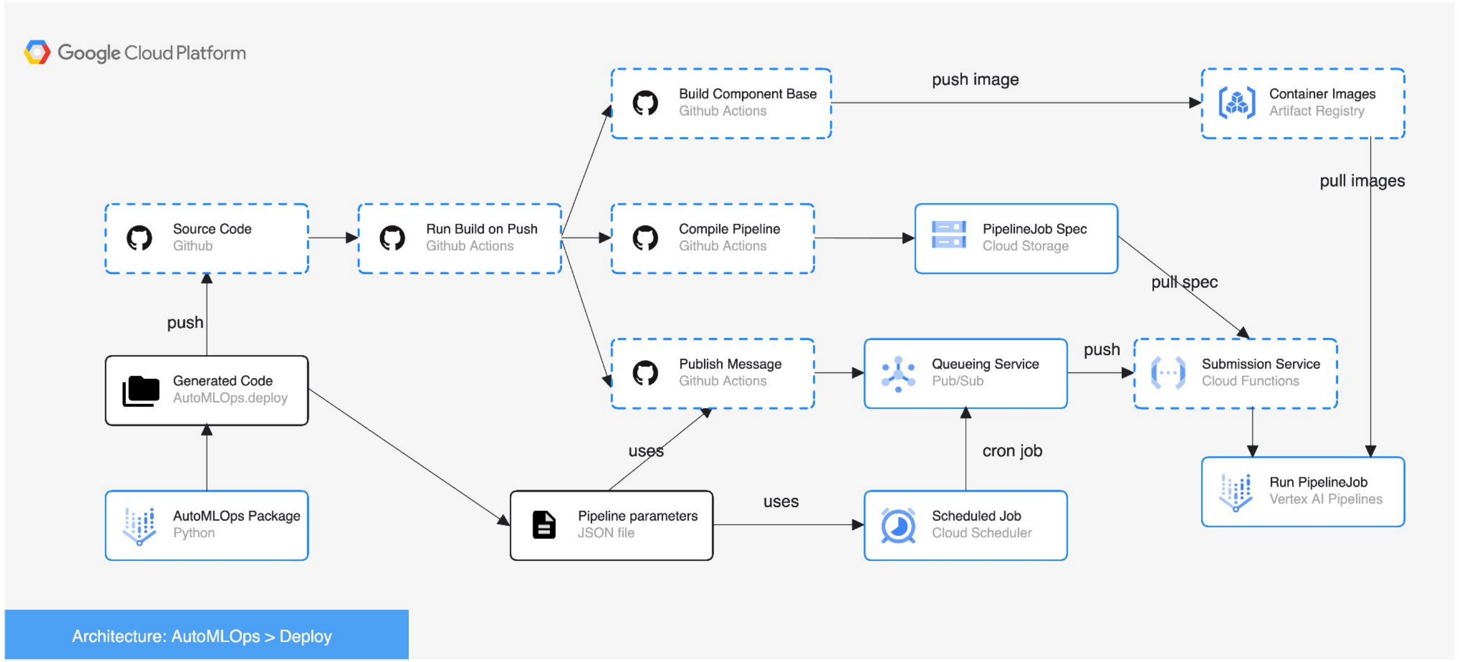
If `use_ci=True`, AutoMLOps will generate and use a fully featured CI/CD environment for the pipeline. Otherwise, it will use the local scripts to build and run the pipeline. In the diagram below dashed boxes show areas users can select and customize their tooling.



deployment_framework=
'cloud-build'

AutoMLOps Deploy

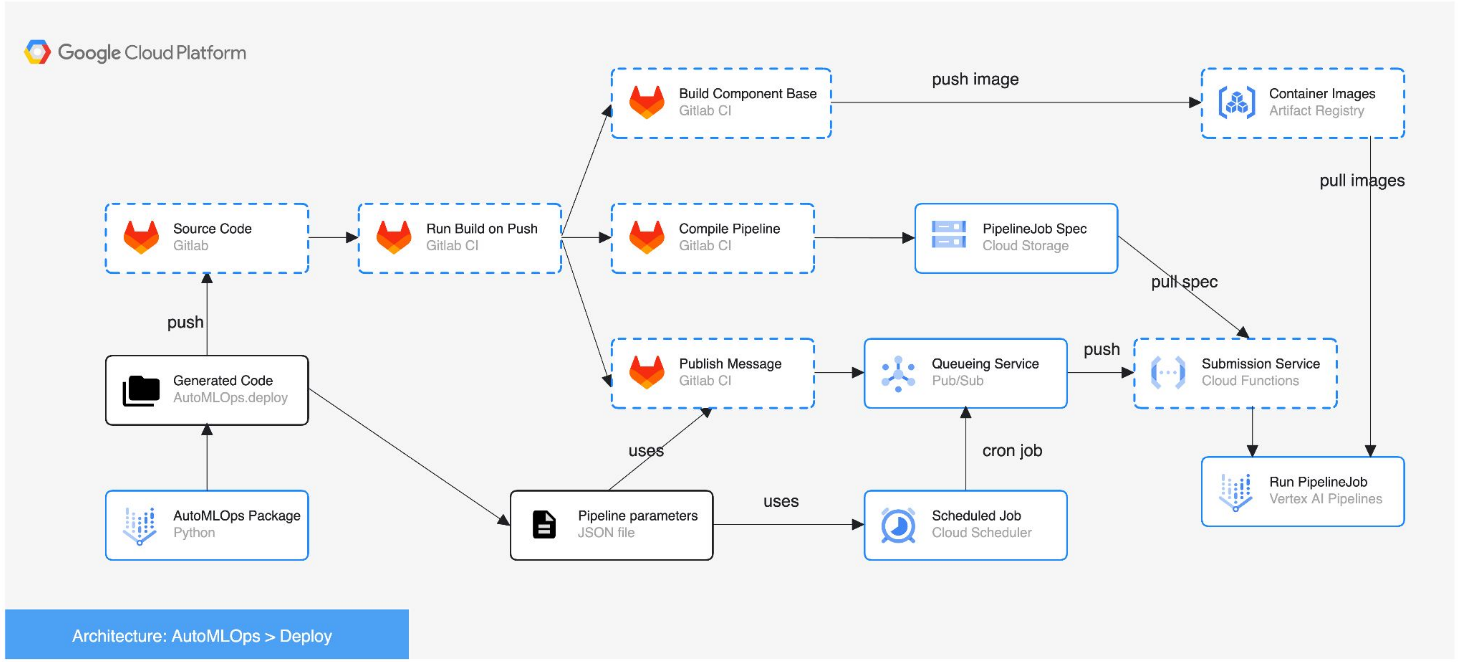
If `use_ci=True`, AutoMLOps will generate and use a fully featured CI/CD environment for the pipeline. Otherwise, it will use the local scripts to build and run the pipeline. In the diagram below dashed boxes show areas users can select and customize their tooling.



deployment_framework=
'github-actions'

AutoMLOps Deploy

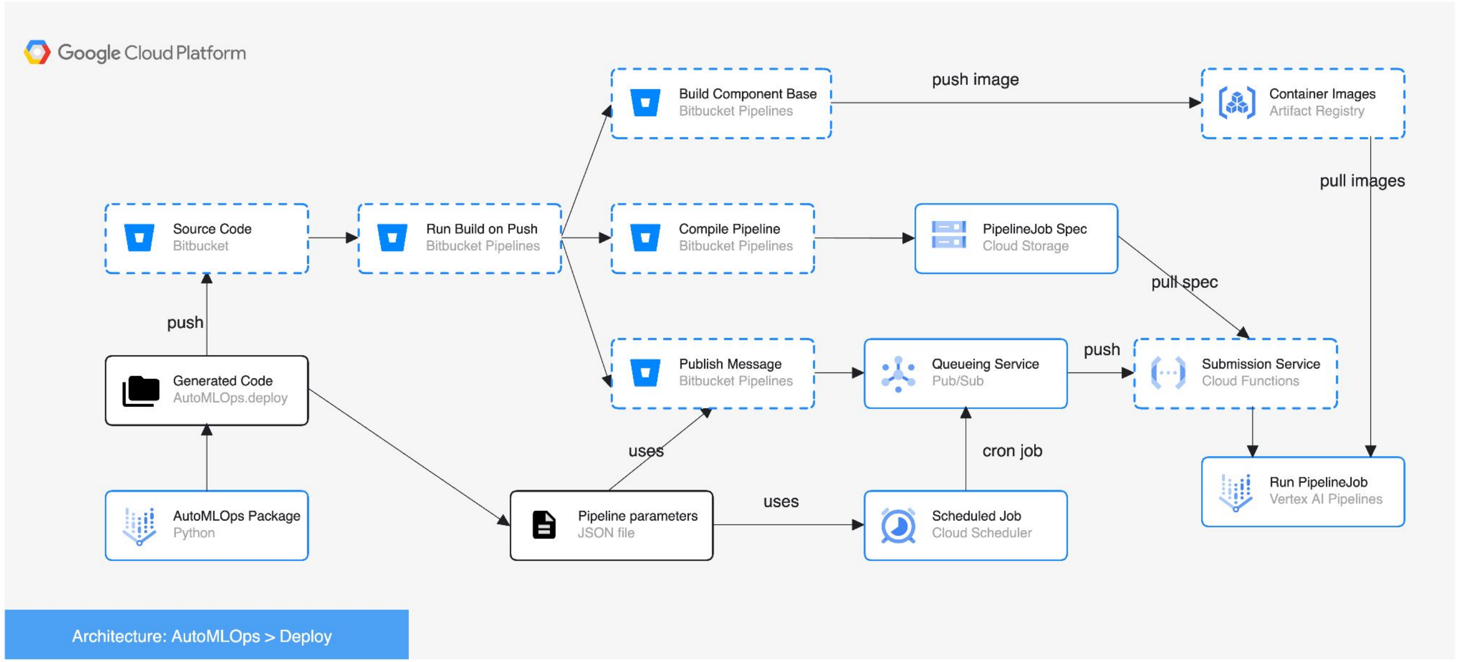
If `use_ci=True`, AutoMLOps will generate and use a fully featured CI/CD environment for the pipeline. Otherwise, it will use the local scripts to build and run the pipeline. In the diagram below dashed boxes show areas users can select and customize their tooling.



deployment_framework=
'gitlab-ci'
[coming soon]

AutoMLOps Deploy

If `use_ci=True`, AutoMLOps will generate and use a fully featured CI/CD environment for the pipeline. Otherwise, it will use the local scripts to build and run the pipeline. In the diagram below dashed boxes show areas users can select and customize their tooling.



deployment_framework=
'bitbucket-pipelines'
[coming soon]

AutoMLOps Deploy Warnings

The hide_warnings parameter specifies whether to show permissions warnings before deploying. If hide_warnings is set to False, it will show a warning specifying necessary permissions and recommended roles; an example run is shown below:

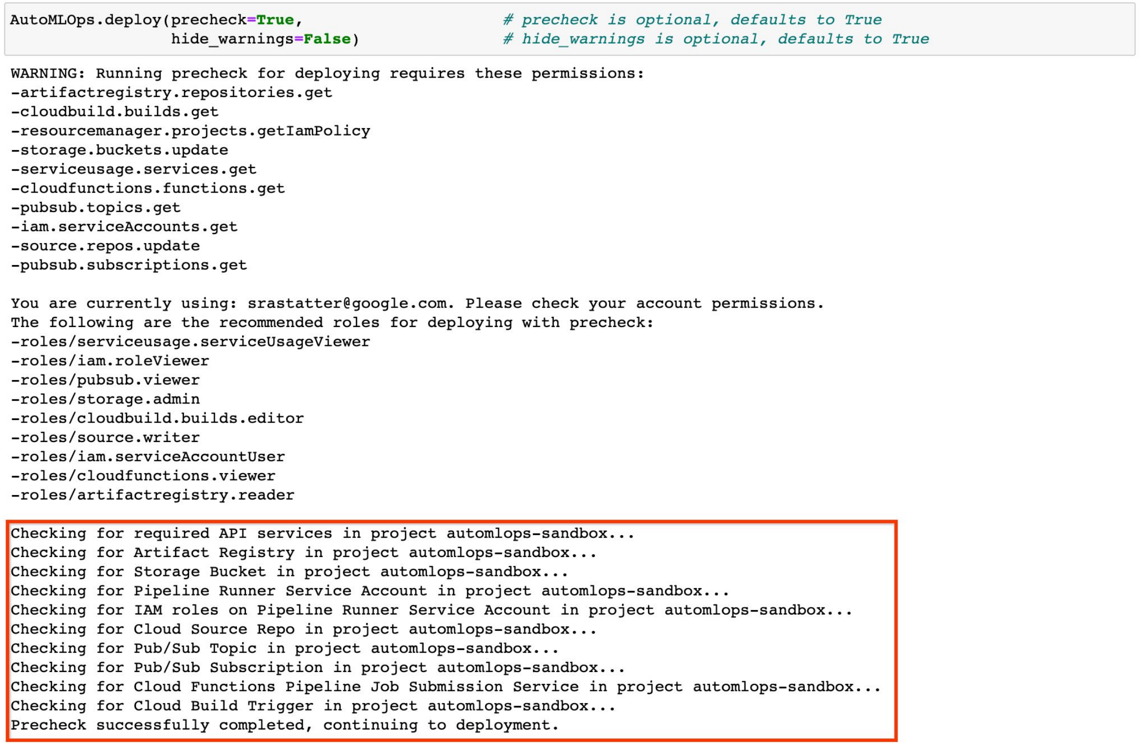
```
AutoMLOps.deploy(precheck=True,                   # precheck is optional, defaults to True
                 hide_warnings=False)             # hide_warnings is optional, defaults to True
```

```
WARNING: Running precheck for deploying requires these permissions:
-artifactregistry.repositories.get
-cloudbuild.builds.get
-resourcemanager.projects.getIamPolicy
-storage.buckets.update
-serviceusage.services.get
-cloudfunctions.functions.get
-pubsub.topics.get
-iam.serviceAccounts.get
-source.repos.update
-pubsub.subscriptions.get

You are currently using: srastatter@google.com. Please check your account permissions.
The following are the recommended roles for deploying with precheck:
-roles/serviceusage.serviceUsageViewer
-roles/iam.roleViewer
-roles/pubsub.viewer
-roles/storage.admin
-roles/cloudbuild.builds.editor
-roles/source.writer
-roles/iam.serviceAccountUser
-roles/cloudfunctions.viewer
-roles/artifactregistry.reader
```

AutoMLOps Deploy Precheck

The precheck parameter specifies whether to check if the necessary infrastructure exists before deploying. If precheck is set to True, it will use the discovery service to determine if the infra pieces exist, it will error out if they do not exist; an example run is shown below:



```
AutoMLOps.deploy(precheck=True,                    # precheck is optional, defaults to True
                 hide_warnings=False)              # hide_warnings is optional, defaults to True
```

```
WARNING: Running precheck for deploying requires these permissions:
-artifactregistry.repositories.get
-cloudbuild.builds.get
-resourcemanager.projects.getIamPolicy
-storage.buckets.update
-serviceusage.services.get
-cloudfunctions.functions.get
-pubsub.topics.get
-iam.serviceAccounts.get
-source.repos.update
-pubsub.subscriptions.get

You are currently using: srastatter@google.com. Please check your account permissions.
The following are the recommended roles for deploying with precheck:
-roles/serviceusage.serviceUsageViewer
-roles/iam.roleViewer
-roles/pubsub.viewer
-roles/storage.admin
-roles/cloudbuild.builds.editor
-roles/source.writer
-roles/iam.serviceAccountUser
-roles/cloudfunctions.viewer
-roles/artifactregistry.reader

Checking for required API services in project automlops-sandbox...
Checking for Artifact Registry in project automlops-sandbox...
Checking for Storage Bucket in project automlops-sandbox...
Checking for Pipeline Runner Service Account in project automlops-sandbox...
Checking for IAM roles on Pipeline Runner Service Account in project automlops-sandbox...
Checking for Cloud Source Repo in project automlops-sandbox...
Checking for Pub/Sub Topic in project automlops-sandbox...
Checking for Pub/Sub Subscription in project automlops-sandbox...
Checking for Cloud Functions Pipeline Job Submission Service in project automlops-sandbox...
Checking for Cloud Build Trigger in project automlops-sandbox...
Precheck successfully completed, continuing to deployment.
```

AutoMLOps LaunchAll

Google Cloud

AutoMLOps LaunchAll

Use the launchAll function to call generate, provision, and deploy all at once. It will run each of these operations one after the other. This function is useful for quickly standing up and running a pipeline for the first time. Below is an example use of this function:

```
AutoMLOps.launchAll(project_id=PROJECT_ID,
       pipeline_params=pipeline_params,
       use_ci=True,
       naming_prefix='dry-beans-dt',
       provisioning_framework='terraform',
       schedule_pattern='59 11 * * 0' # retrain every Sunday at Midnight
)
```

AutoMLOps Monitor

Google Cloud

AutoMLOps Monitor

Use the monitor function to set up and create model monitoring jobs. To use this function, the **setup_model_monitoring** parameter must be set to `True` during the ***AutoMLOps.generate*** step. This will create the necessary files under AutoMLOps/scripts and AutoMLOps/model_monitoring. From there, configurations can be set using ***AutoMLOps.monitor***. This operation requires a functioning Vertex AI model endpoint, and the name of the endpoint's prediction column. Monitoring can be set to check for anomalies in 2 ways:

1. **Data Drift**: This compares incoming feature distributions to past feature distributions that the model endpoint has seen, within a given window of time.
2. **Data Skew**: This compares incoming feature distributions to feature distributions within the training dataset to check for training/serving skew.

At least one of these monitoring types must be used. To specify drift and skew, provide a dictionary of feature names and their threshold values to the `**drift_thresholds**` and `**skew_thresholds**` parameters. When an anomaly is detected, actions can be taken in 2 ways:

1. **Generate email alerts**: This will send an email to the specified email(s) informing the user of a detected data drift or skew.
2. **Automatically retrain the model**: This will send anomaly alerts to the Pub/Sub Queueing Service and trigger a retraining of the model, with the parameters specified, when a data drift or skew is detected.

These can be configured using the `**alert_emails**` and `**auto_retraining_params**` parameters, which are left null by default. If `**auto_retraining_params**` are specified, ***AutoMLOps.monitor*** will set up a Log Sink to connect the Vertex AI Model Monitoring Job to the Pub/Sub Queueing Service, and filter on only ModelMonitoringJob anomalies.

AutoMLOps Monitor

There are a number of optional parameters that can be configured when running monitor. Below is a list of the parameters and their defaults

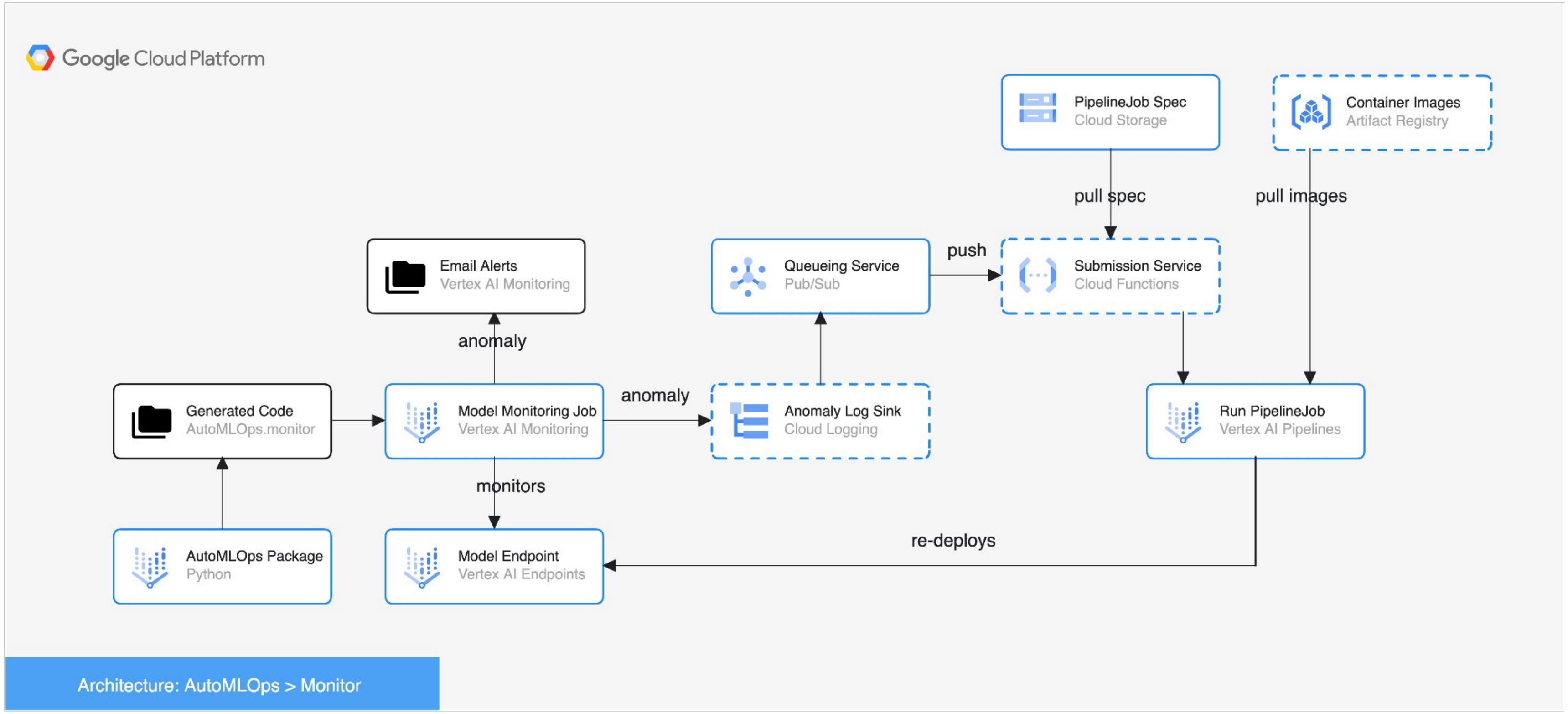
Note: Before running ***AutoMLOps.monitor***, be sure to install the package dependencies using the following command: ***pip3 install -r AutoMLOps/model_monitoring/requirements.txt --user***

```
AutoMLOps.monitor(
       target_field: str, # required
       model_endpoint: str, # required
       alert_emails: Optional[list] = None, # default
       auto_retraining_params: Optional[dict] = None, # default
       drift_thresholds: Optional[dict] = None, # default
       hide_warnings: Optional[bool] = True, # default
       job_display_name: Optional[str] = f'{naming_prefix}-model-monitoring-job', # default
       monitoring_interval: Optional[int] = 1, # default
       monitoring_location: Optional[str] = 'us-central1', # default
       sample_rate: Optional[float] = 0.8, # default
       skew_thresholds: Optional[dict] = None, # default
       training_dataset: Optional[str] = None # default
```

AutoMLOps Monitor Parameter Descriptions

```
- `target_field`: Prediction target column name in training dataset.
- `model_endpoint`: Endpoint resource name of the deployed model to monitoring.
      Format: projects/{project}/locations/{location}/endpoints/{endpoint}
- `alert_emails`: Optional list of emails to send monitoring alerts. Email alerts not used if this value is set to None.
- `auto_retraining_params`: Pipeline parameter values to use when retraining the model. Defaults to None;
      if left None, the model will not be retrained if an alert is generated.
- `drift_thresholds`: Compares incoming data to data previously seen to check for drift.
- `hide_warnings`: Boolean that specifies whether to show permissions warnings before monitoring.
- `job_display_name`: Display name of the ModelDeploymentMonitoringJob. The name can be up to 128 characters long
      and can be consist of any UTF-8 characters.
- `monitoring_interval`: Configures model monitoring job scheduling interval in hours. This defines how often the
      monitoring jobs are triggered.
- `monitoring_location`: Location to retrieve ModelDeploymentMonitoringJob from.
- `sample_rate`: Used for drift detection, specifies what percent of requests to the endpoint are randomly sampled for
      drift detection analysis. This value most range between (0, 1].
- `skew_thresholds`: Compares incoming data to the training dataset to check for skew.
- `training_dataset`: Training dataset used to train the deployed model. This field is required if using skew detection.
```

AutoMLOps Monitor Diagram

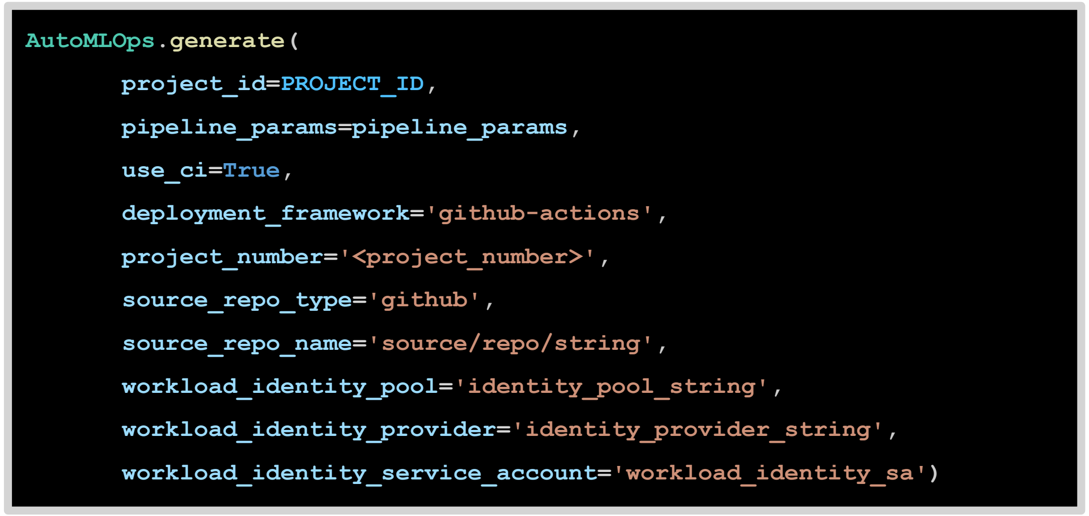


Other Customizations

Google Cloud

AutoMLOps Github Actions Integration

To use Github Actions integration, you must first have a Workload Identity Federation set up properly. You must use a pre-existing Github repo, and you must also have already set up and registered your ssh keys with your Github Repo. If you meet all of these prerequisites, you can use github actions as follows:



```
AutoMLOps.generate(
        project_id=PROJECT_ID,
        pipeline_params=pipeline_params,
        use_ci=True,
        deployment_framework='github-actions',
        project_number='<project_number>',
        source_repo_type='github',
        source_repo_name='source/repo/string',
        workload_identity_pool='identity_pool_string',
        workload_identity_provider='identity_provider_string',
        workload_identity_service_account='workload_identity_sa')
```

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 ***use_ci*** must be set to ***True*** to make use of this feature.

```
AutoMLOps.generate(project_id = PROJECT_ID,
                   pipeline_params = pipeline_params,
                   use_ci = True,
                   schedule_pattern = '0 */12 * * *')
```

The above example will rerun the pipeline every 12 hours.

AutoMLOps Use Vertex AI Experiments

To use Vertex AI Experiments, include key-value pair for **`vertex_experiment_tracking_name`** in your pipeline parameters dictionary. An experiment will be created if one does not already exist with the specified name.

```
pipeline_params = {
       'project_id': PROJECT_ID,
       'region': 'us-central1',
       'vertex_experiment_tracking_name': 'my-experiment-name'
}
AutoMLOps.generate(project_id = PROJECT_ID,
                   pipeline_params = pipeline_params)
```

AutoMLOps Set Pipeline Compute Resources

Use the ***base_image*** and ***custom_training_job_specs*** parameters 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_ci = True,
                   base_image = 'us-docker.pkg.dev/vertex-ai/training/tf-gpu.2-11.py310:latest',
                   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_ci = True,
                   vpc_connector = 'example-vpc-connector')
```

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
):
...
```

Google Cloud

Behind the Scenes

Google Cloud

Importing AutoMLOps

Importing the AutoMLOps package will create a cache subdirectory within the same directory as the file where the import statement is called:

```python
from google_cloud_automlops import AutoMLOps
```

```
├─ my_notebook.ipynb
├─ .AutoMLOps-cache/
```

Defining an AutoMLOps Component

Defining an AutoMLOps component will create a corresponding temporary file within the cache subdirectory:

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

```
├─ my_notebook.ipynb
├─ .AutoMLOps-cache/
        ├─ create_dataset.yaml
```

Defining an AutoMLOps Pipeline

Defining an AutoMLOps pipeline will create a corresponding temporary file within the cache subdirectory:

```python
@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"],
```

```
├─ my_notebook.ipynb
├─ .AutoMLOps-cache/
       ├─ create_dataset.yaml
       ├─ pipeline_scaffold.py
```

Google Cloud

Clearing the cache

Calling clear_cache will remove all previously defined components and pipelines within the directory. Use this function if you have components or pipelines that you no longer need:

```
AutoMLOps.clear_cache()
```

```
├─ my_notebook.ipynb
├─ .AutoMLOps-cache/
      ├─ create_dataset.yaml
      ├─ pipeline_scaffold.py
```

Running AutoMLOps

When AutoMLOps.generate() or AutoMLOps.go() is called, these cached component and pipeline files are "consumed" and turned into the production ready pipeline codebase:

```
AutoMLOps.generate(...)
```

```
├─ my_notebook.ipynb
├─ .AutoMLOps-cache/
       ├─ create_dataset.yaml
       ├─ pipeline_scaffold.py
```

```
├─ my_notebook.ipynb
├─ .AutoMLOps-cache/
       ├─ create_dataset.yaml
       ├─ pipeline_scaffold.py
├─ AutoMLOps/
       ├── services
       ├── components
              ├──component_base
              ├──create_dataset
       ├── provision
       ├── pipelines
              ├── pipeline.py
       ├── configs
       ├── scripts
       └── cloudbuild.yaml
```

APIs

Based on user's tooling selection, AutoMLOps will enable up to the following APIs during the provision step:

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

IAM Access

AutoMLOps will create the following service account and update IAM permissions during the provision step:

1. **Pipeline Runner Service Account** (created if it does not 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/cloudfunctions.admin

Package Dependencies

When using AutoMLOps, the following package versions are used:

1. docopt==0.6.2
2. docstring-parser==0.15
3. google-api-python-client==2.97.0
4. google-auth==2.22.0
5. importlib-resources==6.0.1
6. Jinja2==3.1.2
7. packaging==23.1
8. pipreqs==0.4.13
9. pydantic==2.3.0
10. PyYAML==6.0.1
11. yarg==0.1.9