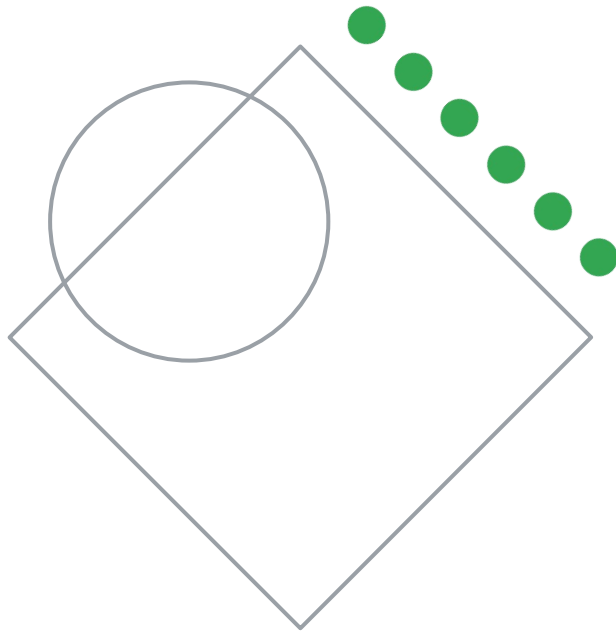
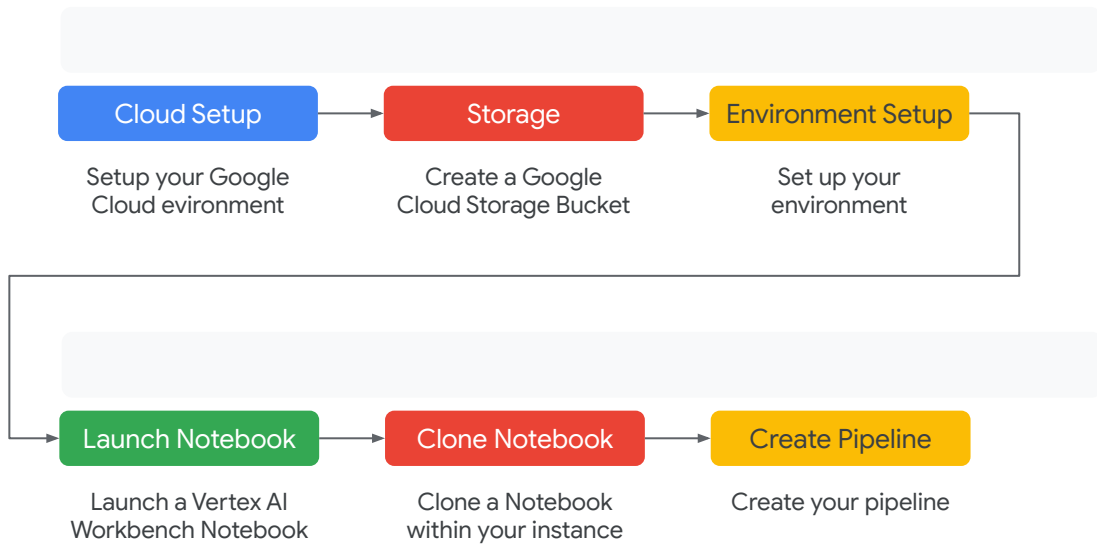


Introduction to Vertex AI Pipelines Lab Walkthrough



Workflow: Introduction to Vertex AI Lab



Setup Steps

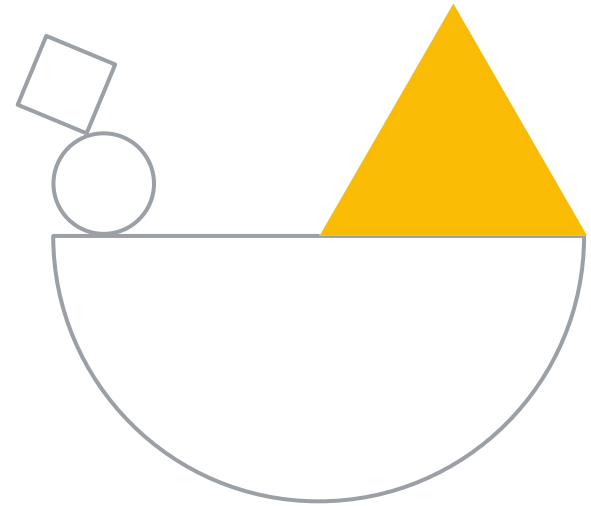
Setup environment/launch notebooks

Pipeline Creation

Create a basic pipeline to understand concepts

Task 1:

Cloud Environment Setup



Activate Cloud Shell!



Find the Google Cloud Project ID

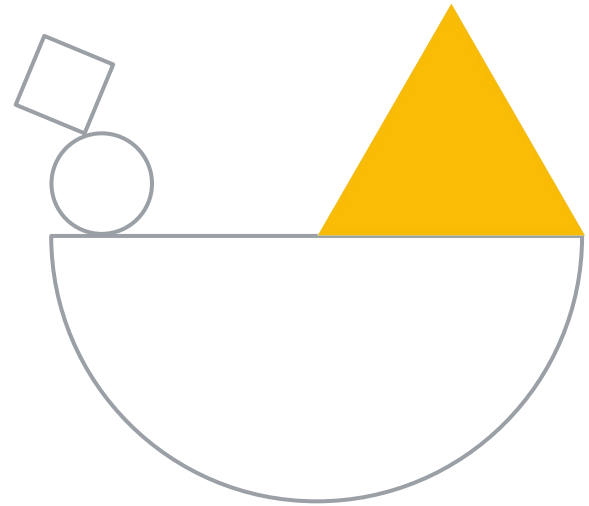
```
Use "gcloud config set project [PROJECT ID]" to change to a different project.  
stripling@cloudshell:~ Your project ID shown here $ echo $GOOGLE_CLOUD_PROJECT
```

Enable the APIs

```
stripling@cloudshell:~ $ gcloud services enable compute.googleapis.com
containerregistry.googleapis.com \
aiplatform.googleapis.com \
cloudbuild.googleapis.com \
cloudfunctions.googleapis.com
Operation "operations/acat.p2-333736501253-babc17e7-e45e-4405-961f-8e2e6c078124" finished successfully.
```

Task 2:

Create a GCS Bucket



Create a Cloud Storage Bucket

```
stripling@cloudshell:~ | $ BUCKET_NAME=gs://$GOOGLE_CLOUD_PROJECT-bucket  
gsutil mb -l us-central1 $BUCKET_NAME  
Creating gs://cloud-training-prod-bucket-bucket/...
```

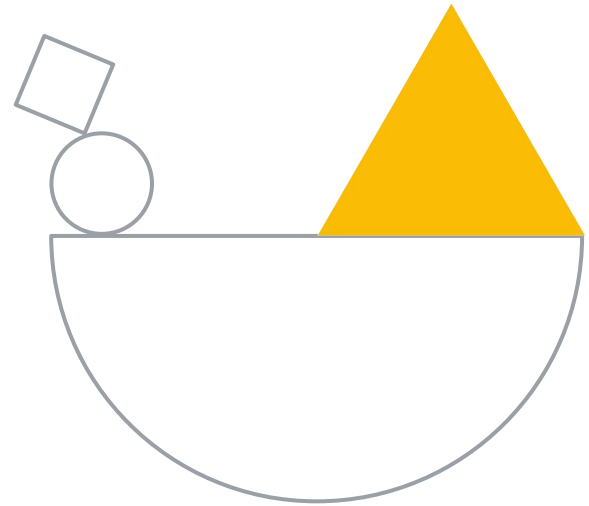

Access to Bucket

```
stripling@cloudshell:~ (c [REDACTED]) gcloud projects describe $GOOGLE_CLOUD_PROJECT > project-info.txt
PROJECT_NUM=$(cat project-info.txt | sed -nre 's:.*projectNumber\: (.*):\1:p')
SVC_ACCOUNT="${PROJECT_NUM//\'/}-compute@developer.gserviceaccount.com"
gcloud projects add-iam-policy-binding $GOOGLE_CLOUD_PROJECT --member serviceAccount:$SVC_ACCOUNT --role roles/storage.objectAdmin
Updated IAM policy for project [cloud-training-prod-bucket].
bindings:
```

Next we give our compute service account access to this bucket. This ensures that Vertex Pipelines has the necessary permissions to write files to this bucket.

Task 3:

Your Environment Setup



Navigate to the [Vertex AI section of your Cloud Console](#) and click Enable Vertex AI API.

Get started with Vertex AI

Vertex AI empowers machine learning developers, data scientists, and data engineers to take their projects from ideation to deployment, quickly and cost-effectively. [Learn more](#)

ENABLE VERTEX AI API

Region

us-central1 (Iowa)



Prepare your training data

Collect and prepare your data, then import it into a dataset to train a model



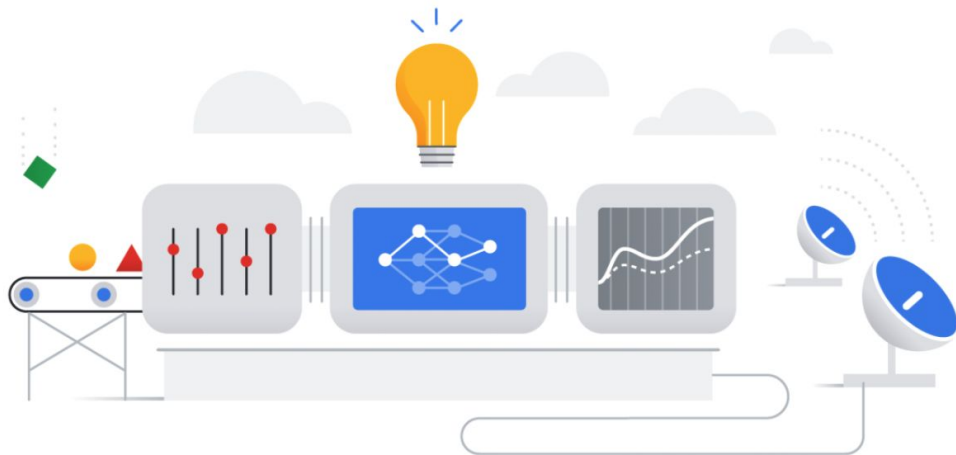
Train your model

Train a best-in-class machine learning model with your dataset. Use **Google's AutoML**, or

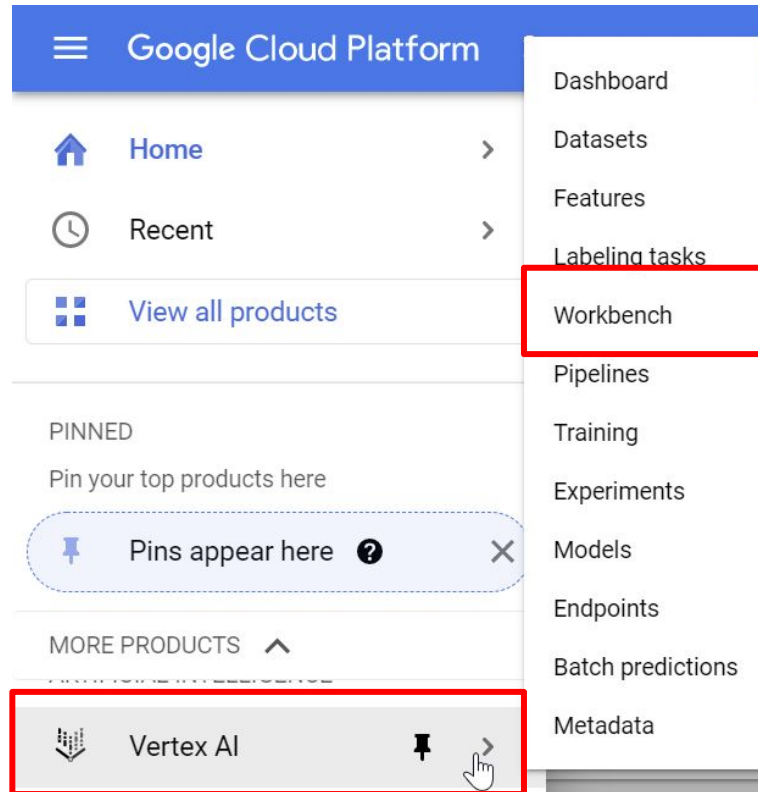


Get predictions

After you train a model, you can use it to get predictions, either online as an endpoint or

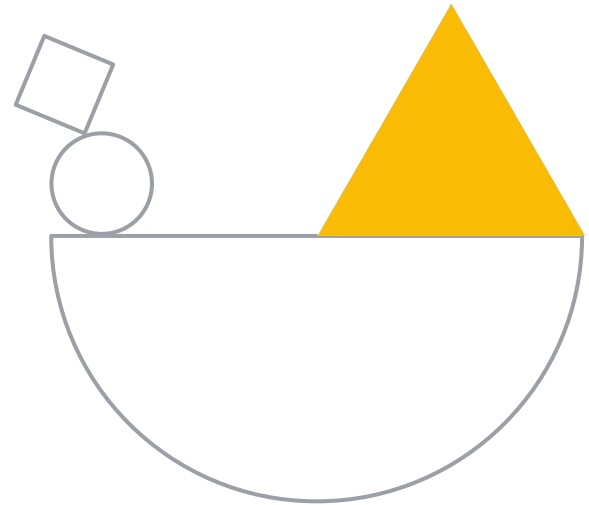


Select Vertex AI, then Workbench.



Task 4:

Launch Vertex AI Workbench





Vertex AI



Dashboard



Datasets



Features



Labeling tasks



Workbench



Pipelines



Training



Experiments



Models



Marketplace

Workbench



NEW NOTEBOOK



REFRESH

MANAGED NOTEBOOKS

PREVIEW

USER-MANAGED NOTEBOOKS

EXECUTIONS

PREVIEW



Filter Enter property name or value

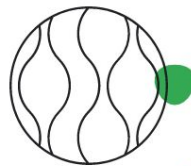


Notebook name ↑

Zone

Auto-upgrade

Envir



You don't have any notebooks in this project yet

CREATE NOTEBOOK



Vertex AI



Dashboard



Datasets



Features



Labeling tasks



Workbench

Workbench

 NEW NOTEBOOK

 REFRESH

MANAGED NOTEBOOKS

PREVIEW


USER-MANAGED NOTEBOOKS

EXECU

Notebooks have JupyterLab 3 pre-installed and are configured with GPU-enabled machine learning frameworks. [Learn more](#)

 Filter Enter property name or value



Notebook name 

Zone

Auto-upgrade

TensorFlow 2.6 – LTS is “Long Term Supported”

The screenshot shows the Google Cloud AI Platform interface. At the top, there is a navigation bar with buttons: **+ NEW NOTEBOOK**, **REFRESH**, **START**, **STOP**, **RESET**, **DELETE**, and a **LEARN** link. Below the navigation bar, on the left, is a sidebar with a 'Customize...' dropdown and a list of notebook templates: Python 3, Python 3 (CUDA Toolkit 11.0), TensorFlow Enterprise, PyTorch 1.10, R 4.1, RAPIDS 0.18 [EXPERIMENTAL], and Kaggle Python [BETA]. On the right, there is a 'SCHEDULES' tab and a 'PREVIEW' button. Below these, a list of TensorFlow Enterprise versions is shown: 1.15 (with LTS), 2.1 (with LTS), 2.3 (with LTS), 2.6 (with LTS), and 2.7. A red box highlights the 'TensorFlow Enterprise 2.6 (with LTS)' option, and a hand cursor points to it. To the right of the version list, a table shows the permissions for each version.

TensorFlow Enterprise 1.15 (with LTS)	Permission	L
TensorFlow Enterprise 2.1 (with LTS)	Service account	F
TensorFlow Enterprise 2.3 (with LTS)		P
TensorFlow Enterprise 2.6 (with LTS)	Without GPUs	
TensorFlow Enterprise 2.7	With 1 NVIDIA Tesla T4	

Here are the parameters of your new Workbench Notebook.

Your region may show “us-central” - do not change your region.

It takes a few minutes to spin up your Notebook.

New notebook

Notebook name *

tensorflow-2-6-20220226-215208

63-char limit with lowercase letters, digits, or '-' only. Must start with a letter. Cannot end with a '-'.

Region *

us-west1 (Oregon) ▼ ?

Zone *

us-west1-b ▼ ?

Notebook properties

Environment ?	TensorFlow Enterprise 2.6 (with LTS and Intel® MKL-DNN/MKL)
Machine type	4 vCPUs, 15 GB RAM
Boot disk	100 GB Standard persistent disk
Data disk	100 GB Standard persistent disk
Subnetwork	<div>default(10.138.0.0/20) ▼</div>
External IP	Ephemeral(Automatic)
Permission	Compute Engine default service account
Estimated cost ?	\$102.70 monthly, \$0.141 hourly

ADVANCED OPTIONS

CANCEL

CREATE

Here is your new Notebook. Select “Open Jupyter Lab.”

Workbench [+ NEW NOTEBOOK](#) [REFRESH](#) [▶ START](#) [■ STOP](#) [⏻ RESET](#) [🗑 DELETE](#) [LEARN](#) [SHOW INFO PANEL](#)

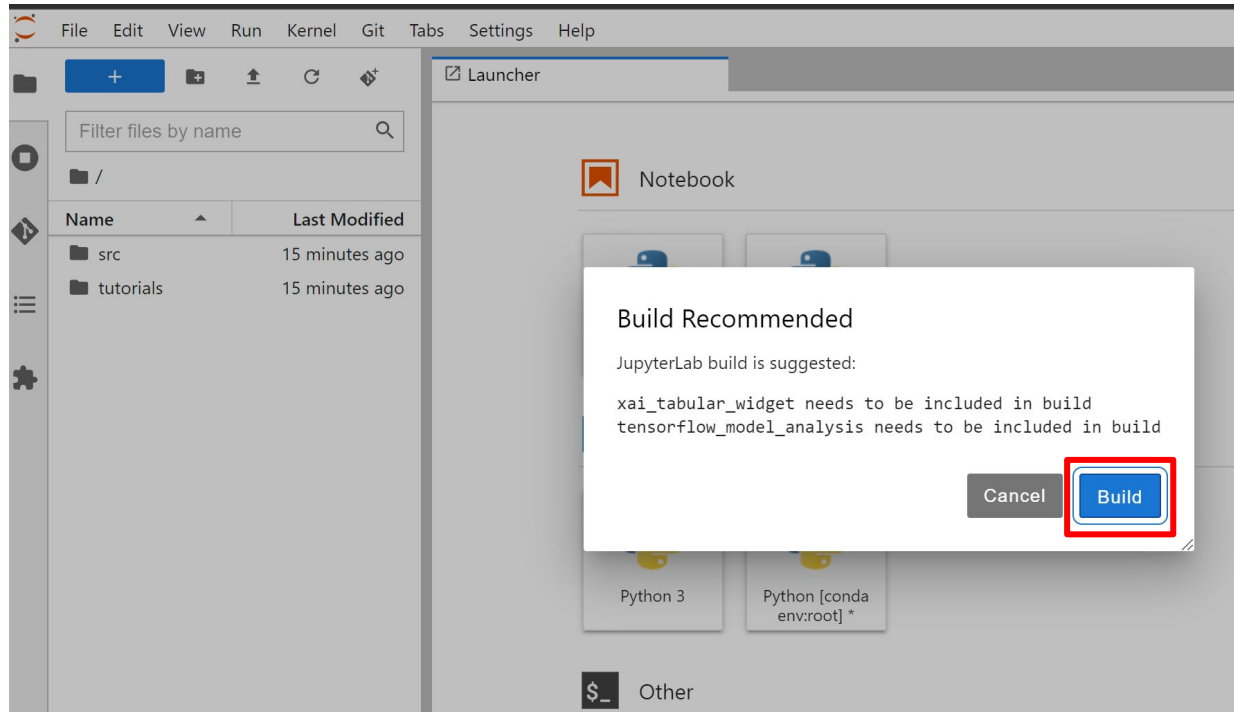
MANAGED NOTEBOOKS [PREVIEW](#) [USER-MANAGED NOTEBOOKS](#) EXECUTIONS [PREVIEW](#) SCHEDULES [PREVIEW](#)

Notebooks have JupyterLab 3 pre-installed and are configured with GPU-enabled machine learning frameworks. [Learn more](#)

[Filter](#) Enter property name or value [?](#) [⌵](#)

<input type="checkbox"/>	<input type="checkbox"/>	Notebook name ↑		Zone	Auto upgrade	Environment	Machine type	GPUs	Permission	Last modified
<input type="checkbox"/>	<input checked="" type="checkbox"/>	tensorflow-2-6-20220226-215208	OPEN JUPYTERLAB	us-central1-a	—	TensorFlow:2.6	4 vCPUs, 15 GB RAM	None	Service account	Feb 26, 2022, 9:59:11 PM
<input type="checkbox"/>	<input checked="" type="checkbox"/>	tensorflow-2-7	OPEN JUPYTERLAB	us-west1-b	—	TensorFlow:2.7	4 vCPUs, 15 GB RAM	None	Service account	Feb 26, 2022, 9:59:11 PM

You will see “Build recommended” pop up, click **Build**. If you see the build failed, ignore it.



You may see this message - just dismiss it.

Build Failed

Build failed with 524.

If you are experiencing the build failure after installing an extension (or trying to include previously installed extension .

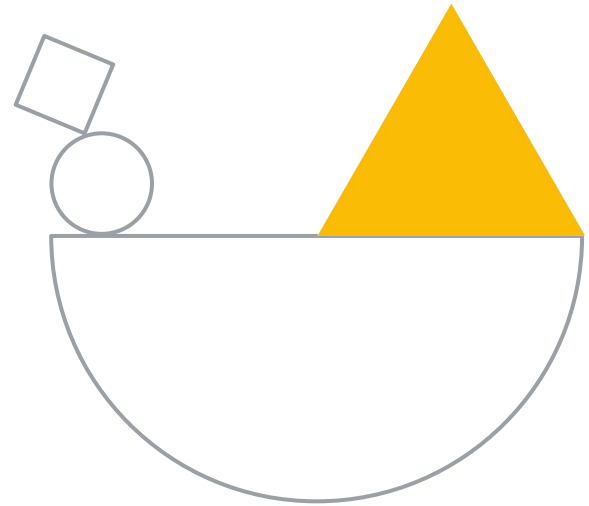
If you specifically intended to install a source extension, please run 'jupyter lab build' on the server for full output.

A blue rectangular button with rounded corners and a thin white border, containing the word "Dismiss" in white text. A white mouse cursor arrow is pointing at the bottom right corner of the button. The button is enclosed within a red rectangular border.

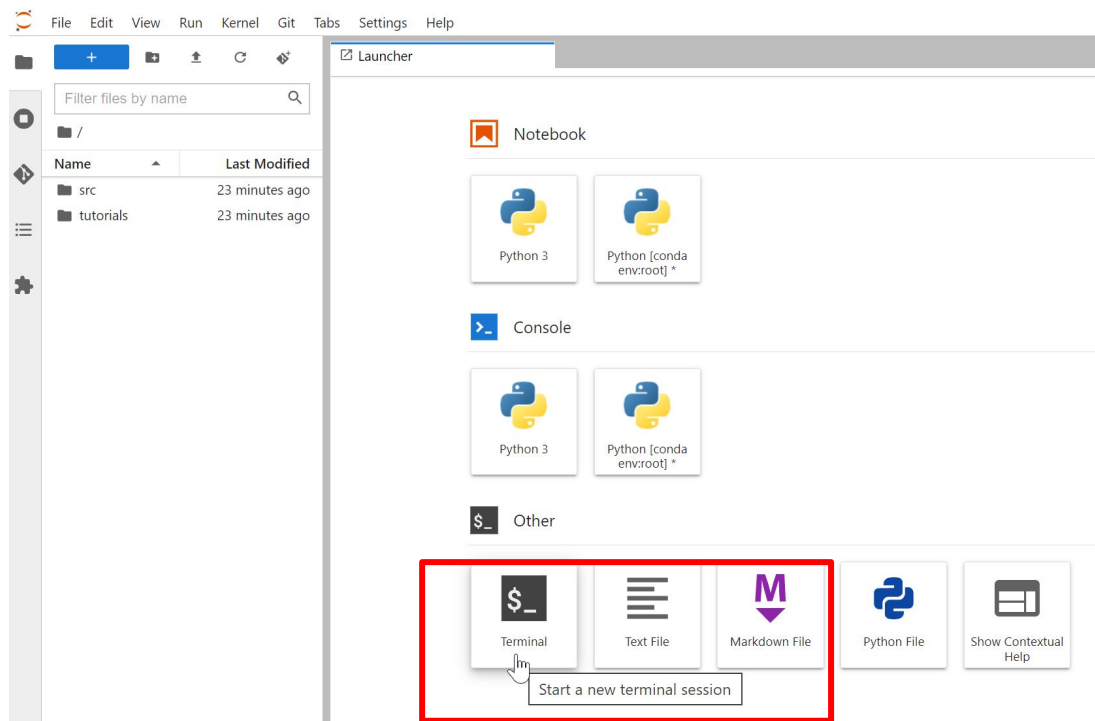
Dismiss

Task 5:

Clone the Notebook



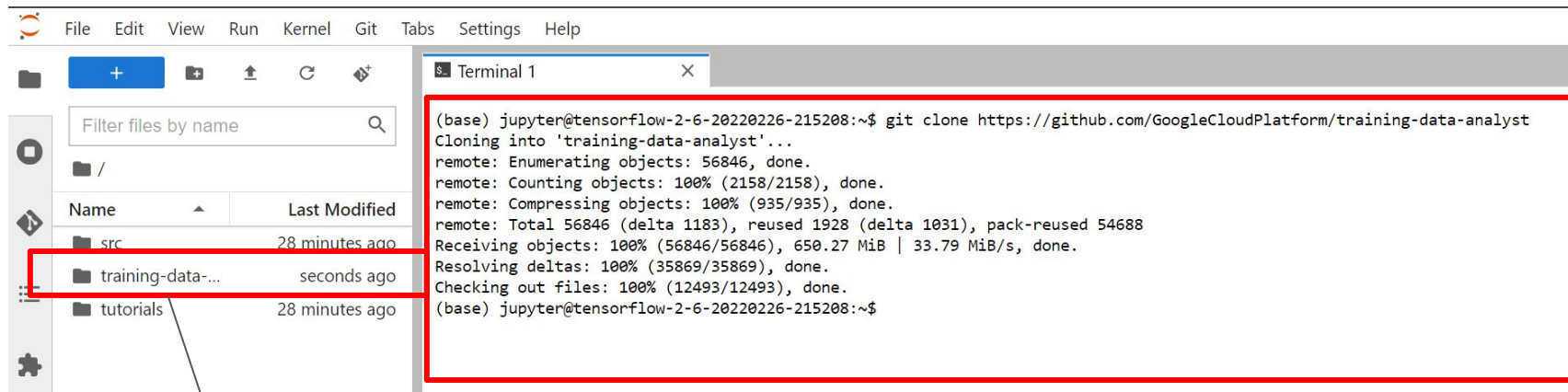
To clone the training-data-analyst notebook in your JupyterLab instance. In JupyterLab, to open a new terminal, click the **Terminal** icon.



At the command-line prompt, run the following command.

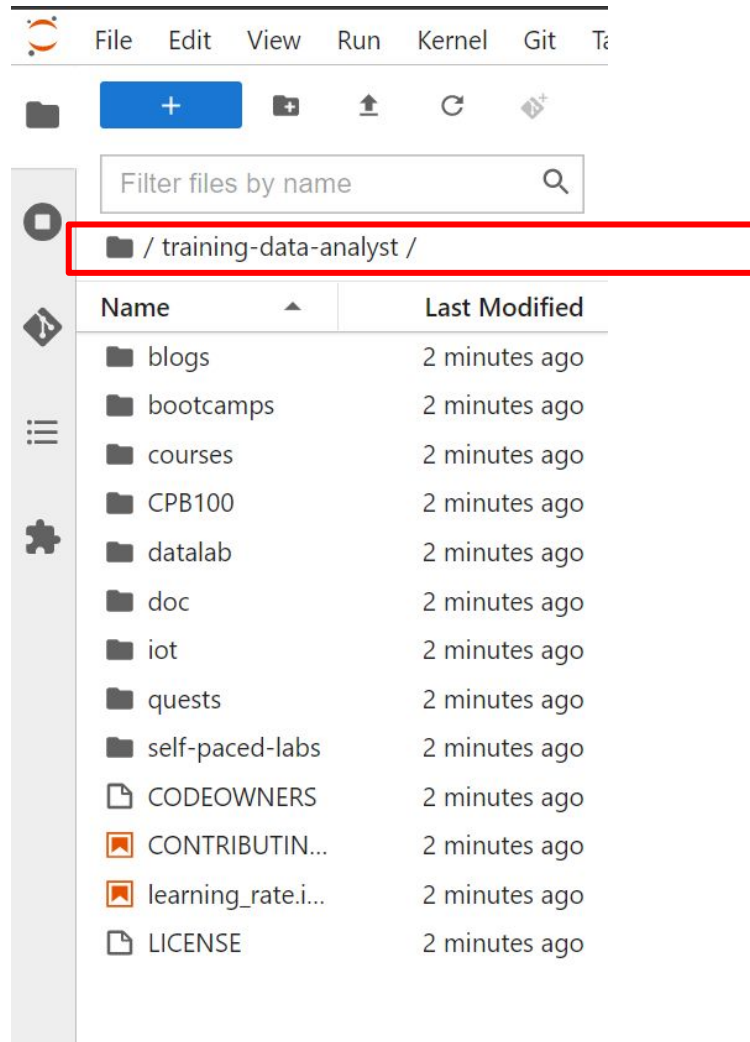


After you run the command, you will see a new folder.



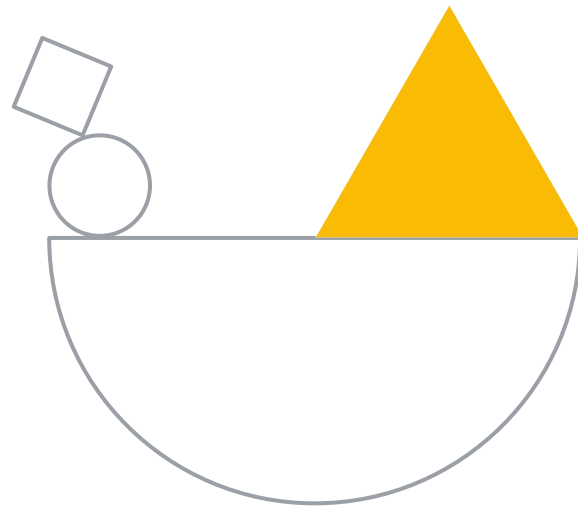
New folder

To confirm that you have cloned the repository, double-click on the training-data-analyst directory and ensure that you can see its contents. The files for all the Jupyter notebook-based labs throughout this course are available in this directory.

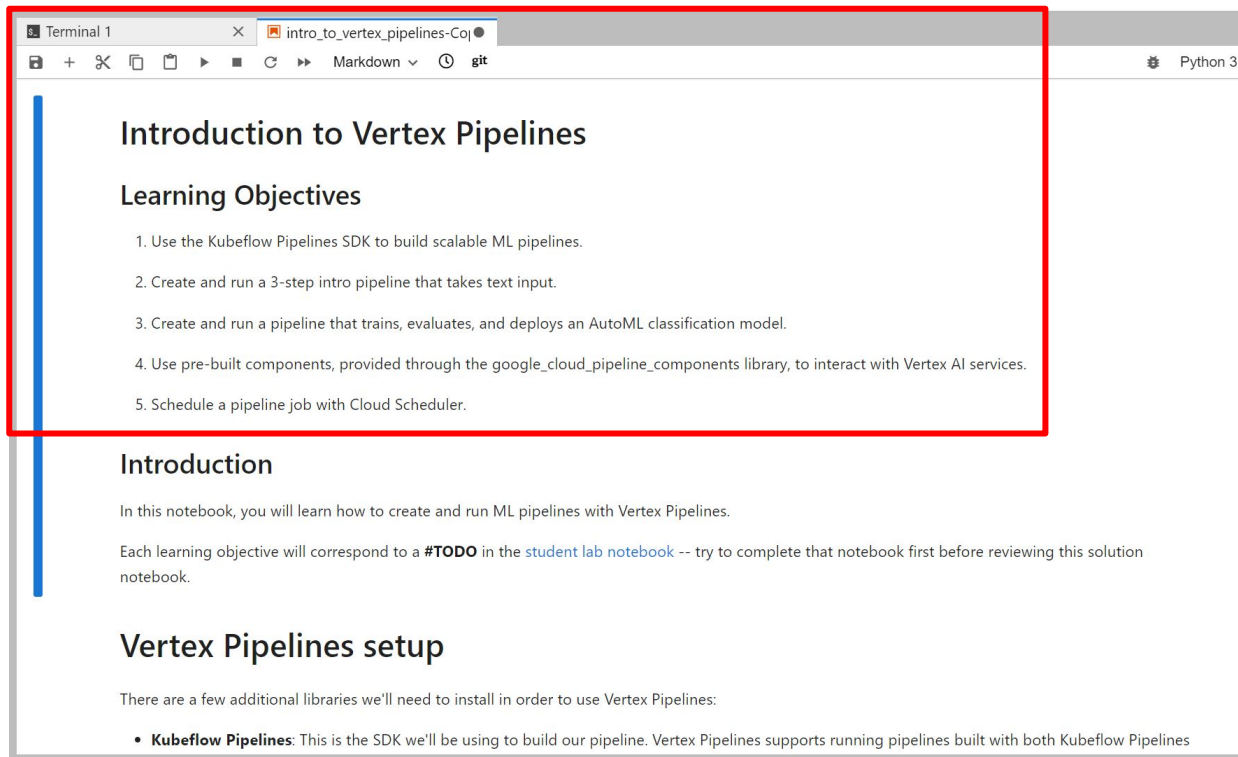


Task 6:

Create the Vertex AI Pipeline



Here is the notebook!



Step 1: Install necessary libraries

To install both services we'll be using in this lab, first set the user flag in a notebook cell:

```
: USER_FLAG = "--user"
```

Then run the following from your notebook:

```
: # Install necessary libraries
!pip3 install {USER_FLAG} google-cloud-aiplatform==1.0.0 --upgrade
!pip3 install {USER_FLAG} kfp google-cloud-pipeline-components==0.1.1 --upgrade
```

Collecting google-cloud-aiplatform==1.0.0

Downloading google_cloud_aiplatform-1.0.0-py2.py3-none-any.whl (1.8 MB)
1.8/1.8 MB 25.4 MB/s eta 0:00:0000:0100:01

Requirement already satisfied: packaging>=14.3 in /opt/conda/lib/python3.7/site-packages (from google-cloud-aiplatform==1.0.0) (21.3)

Requirement already satisfied: google-cloud-bigquery<3.0.0dev,>=1.15.0 in /opt/conda/lib/python3.7/site-packages (from google-cloud-aiplatform==1.0.0) (2.32.0)

Collecting google-api-core[grpc]<2.0.0dev,>=1.22.2

Downloading google_api_core-1.31.5-py2.py3-none-any.whl (93 kB)
93.3/93.3 KB 13.6 MB/s eta 0:00:00

Collecting google-cloud-storage<2.0.0dev,>=1.32.0

Downloading google_cloud_storage-1.44.0-py2.py3-none-any.whl (106 kB)
106.8/106.8 KB 14.4 MB/s eta 0:00:00

Requirement already satisfied: proto-plus>=1.10.1 in /opt/conda/lib/python3.7/site-packages (from google-cloud-aiplatform==1.0.0) (1.19.9)

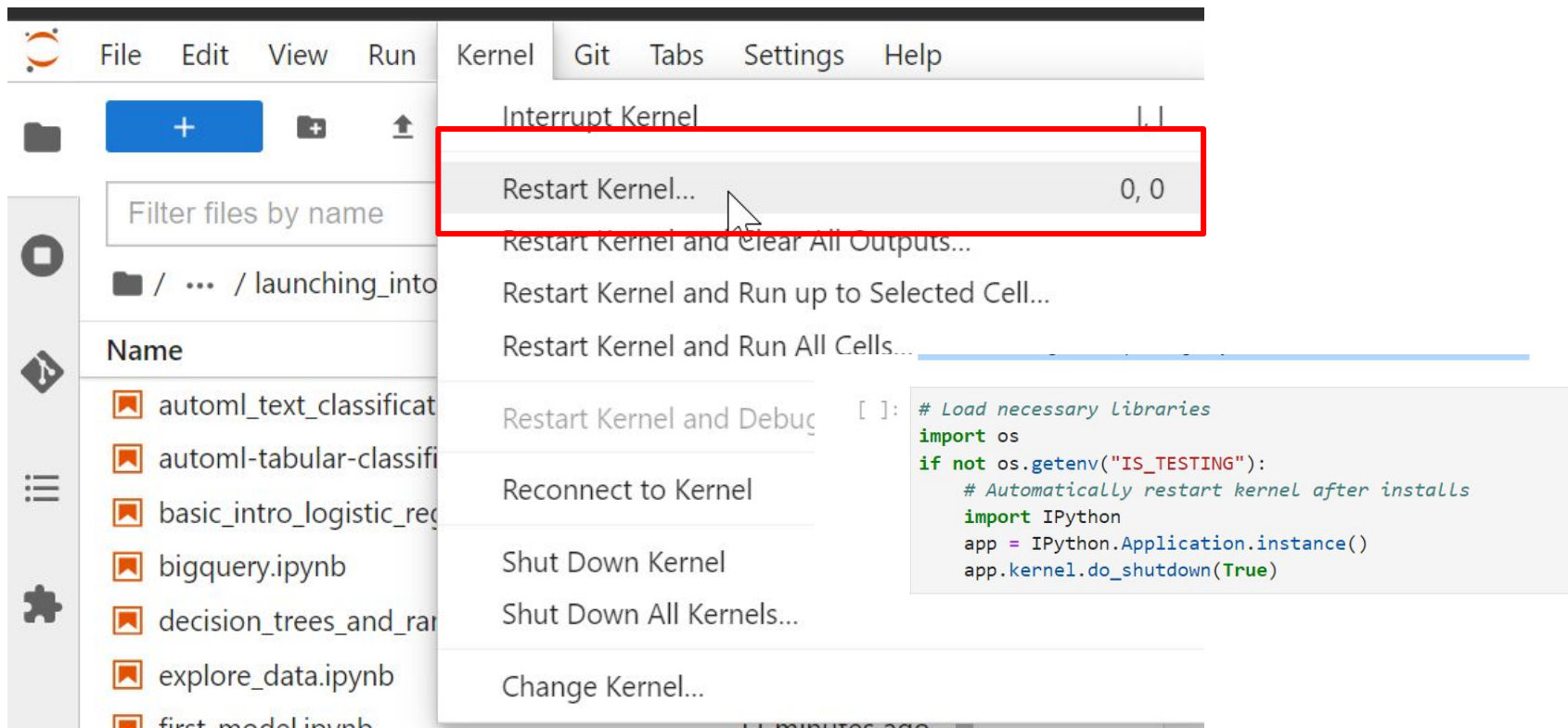
Requirement already satisfied: protobuf>=3.12.0 in /opt/conda/lib/python3.7/site-packages (from google-api-core[grpc]<2.0.0dev,>=1.22.2->google-cloud-aiplatform==1.0.0) (3.19.4)

Requirement already satisfied: setuptools>=40.3.0 in /opt/conda/lib/python3.7/site-packages (from google-api-core[grpc]<2.0.0dev,>=1.22.2->google-cloud-aiplatform==1.0.0) (59.8.0)

Requirement already satisfied: googleapis-common-protos<2.0dev,>=1.6.0 in /opt/conda/lib/python3.7/site-packages (from google-api-core[grpc]<2.0.0dev,>=1.22.2->google-cloud-aiplatform==1.0.0) (1.54.0)

Requirement already satisfied: pytz in /opt/conda/lib/python3.7/site-packages (from google-api-core[grpc]<2.0.0dev,>=1.22.2->google-cloud-

After installing these packages, you'll need to restart the kernel



Restart the Kernel

After installing these packages you'll need to restart the kernel:

```
[1]: # Load necessary Libraries
import os
if not os.getenv("IS_TESTING"):
    # Automatically restart kernel after installs
    import IPython
    app = IPython.Application.instance()
    app.kernel.do_shutdown(True)
```

Finally, check that you have correctly installed the packages. **The KFP SDK version should be ≥ 1.6 :**

```
[1]: # print KFP SDK version
!python3 -c "import kfp; print('KFP SDK version: {}'.format(kfp.__version__))"
!python3 -c "import google_cloud_pipeline_components; print('google_cloud_pipeline_components version: {}'.format(google_cloud_pipeline_cor
```

```
KFP SDK version: 1.8.11
google_cloud_pipeline_components version: 0.1.1
```

Check that you have installed the packages

After installing these packages you'll need to restart the kernel:

```
[1]: # Load necessary libraries
import os
if not os.getenv("IS_TESTING"):
    # Automatically restart kernel after installs
    import IPython
    app = IPython.Application.instance()
    app.kernel.do_shutdown(True)
```

Finally, check that you have correctly installed the packages. **The KFP SDK version should be ≥ 1.6 :**

```
[1]: # print KFP SDK version
!python3 -c "import kfp; print('KFP SDK version: {}'.format(kfp.__version__))"
!python3 -c "import google_cloud_pipeline_components; print('google_cloud_pipeline_components version: {}'.format(google_cloud_pipeline_cor

KFP SDK version: 1.8.11
google_cloud_pipeline_components version: 0.1.1
```


Step 2: Set your project ID and bucket

Throughout this lab you'll reference your Cloud project ID and the bucket you created earlier. Next we'll create variables for each of those.

If you don't know your project ID you may be able to get it by running the following:

```
] : import os
PROJECT_ID = ""
# Get your Google Cloud project ID from gcloud
if not os.getenv("IS_TESTING"):
    shell_output=!gcloud config list --format 'value(core.project)' 2>/dev/null
    PROJECT_ID = shell_output[0]
    print("Project ID: ", PROJECT_ID)
```

Project ID: qwiklabs-gcp-04-75f7f02eedf5

Otherwise, set it here:

```
] : if PROJECT_ID == "" or PROJECT_ID is None:
    PROJECT_ID = "your-project-id" # @param {type:"string"}
```

Then create a variable to store your bucket name. If you created it in this lab, the following will work. Otherwise, you'll need to set this manually:

```
] : BUCKET_NAME="gs://" + PROJECT_ID + "-bucket"
```


Step 3: Import libraries

Add the following to import the *libraries* we'll be using throughout this lab:

```
] : # Load necessary Libraries
from typing import NamedTuple
import kfp
from kfp import dsl
from kfp.v2 import compiler
from kfp.v2.dsl import (Artifact, Dataset, Input, InputPath, Model, Output,
                        OutputPath, ClassificationMetrics, Metrics, component)
from kfp.v2.google.client import AIPlatformClient
from google.cloud import aiplatform
from google_cloud_pipeline_components import aiplatform as gcc_aip
```

<https://arxiv.org/ftp/arxiv/papers/1602/1602.07637.pdf>

<https://kubeflow-pipelines.readthedocs.io/en/latest/source/kfp.dsl.html>

Step 4: Define constants

The last thing we need to do before building our pipeline is define some constant variables. PIPELINE_ROOT is the Cloud Storage path where the artifacts created by our pipeline will be written. We're using us-central1 as the region here, but if you used a different region when you created your bucket, update the REGION variable in the code below:

```
[ ]: PATH=%env PATH
%env PATH={PATH}:/home/jupyter/.local/bin
REGION="us-central1"
PIPELINE_ROOT = f"{BUCKET_NAME}/pipeline_root/"
PIPELINE_ROOT
```

```
env: PATH=/usr/local/nvidia/bin:/usr/local/cuda/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/tools/node/bin:/tools/google-cloud-sdk/bin:/opt/bin:/home/jupyter/.local/bin
```

```
[6]: 'gs://qwiklabs-gcp-04-75f7f02eedf5-bucket/pipeline_root/'
```

After running the code above, you should see the root directory for your pipeline printed. This is the Cloud Storage location where the artifacts from your pipeline will be written. It will be in the format of gs://Your_Project_ID/pipeline_root/

Creating your first pipeline

To get familiar with how Vertex Pipelines works, we'll first create a short pipeline using the KFP SDK. This pipeline doesn't do anything ML related (don't worry, we'll get there!), we're using it to teach you:

- How to create custom components in the KFP SDK
- How to run and monitor a pipeline in Vertex Pipelines

We'll create a pipeline that prints out a sentence using two outputs: a product name and an emoji description. This pipeline will consist of three components:

- **product_name:** This component will take a product name (or any noun you want really) as input, and return that string as output
- **emoji:** This component will take the text description of an emoji and convert it to an emoji. For example, the text code for 🌟 is "sparkles". This component uses an emoji library to show you how to manage external dependencies in your pipeline
- **build_sentence:** This final component will consume the output of the previous two to build a sentence that uses the emoji. For example, the resulting output might be "Vertex Pipelines is 🌟".

Let's start coding!

Step 1: Create a Python function based component

Using the KFP SDK, we can create components based on Python functions. We'll use that for the 3 components in our first pipeline. We'll first build the **product_name** component, which simply takes a string as input and returns that string. Add the following to your notebook:

```
[ ]: @component(base_image="python:3.9", output_component_file="first-component.yaml")
def product_name(text: str) -> str:
    return text
```

Let's take a closer look at the syntax here:

- The @component decorator compiles this function to a component when the pipeline is run. You'll use this anytime you write a custom component.
- The base_image parameter specifies the container image this component will use.
- The output_component_file parameter is optional, and specifies the yaml file to write the compiled component to. After running the cell you should see that file written to your notebook instance. If you wanted to share this component with someone, you could send them the generated yaml file and have them load it with the following:

```
[ ]: # TODO
product_name_component = kfp.components.load_component_from_file('./first-component.yaml')
```

- The -> str after the function definition specifies the output type for this component.

Step 2: Create two additional components

To complete our pipeline, we'll create two more components. The first one we'll define takes a string as input, and converts this string to its corresponding emoji if there is one. It returns a tuple with the input text passed, and the resulting emoji:

```
[ ]: @component(packages_to_install=["emoji"])
def emoji(
    text: str,
) -> NamedTuple(
    "Outputs",
    [
        ("emoji_text", str), # Return parameters
        ("emoji", str),
    ],
):
    import emoji
    emoji_text = text
    emoji_str = emoji.emojize(':' + emoji_text + ':', use_aliases=True)
    print("output one: {}; output_two: {}".format(emoji_text, emoji_str))
    return (emoji_text, emoji_str)
```

This component is a bit more complex than our previous one. Let's break down what's new:

- The `packages_to_install` parameter tells the component any external library dependencies for this container. In this case, we're using a library called `emoji`.
- This component returns a `NamedTuple` called `Outputs`. Notice that each of the strings in this tuple have keys: `emoji_text` and `emoji`. We'll use these in our next component to access the output.

The final component in this pipeline will consume the output of the first two and combine them to return a string:

The final component in this pipeline will consume the output of the first two and combine them to return a string:

```
[ ]: @component
def build_sentence(
    product: str,
    emoji: str,
    emotext: str
) -> str:
    print("We completed the pipeline, hooray!")
    end_str = product + " is "
    if len(emoji) > 0:
        end_str += emoji
    else:
        end_str += emotext
    return(end_str)
```

You might be wondering: how does this component know to use the output from the previous steps you defined? Good question! We will tie it all together in the next step.

Step 3: Putting the components together into a pipeline

The component definitions we defined above created factory functions that can be used in a pipeline definition to create steps. To set up a pipeline, use the `@dsl.pipeline` decorator, give the pipeline a name and description, and provide the root path where your pipeline's artifacts should be written. By artifacts, we mean any output files generated by your pipeline. This intro pipeline doesn't generate any, but our next pipeline will.

In the next block of code we define an `intro_pipeline` function. This is where we specify the inputs to our initial pipeline steps, and how steps connect to each other:

- `product_task` takes a product name as input. Here we're passing "Vertex Pipelines" but you can change this to whatever you'd like.
- `emoji_task` takes the text code for an emoji as input. You can also change this to whatever you'd like. For example, "party_face" refers to the 🥳 emoji. Note that since both this and the `product_task` component don't have any steps that feed input into them, we manually specify the input for these when we define our pipeline.
- The last step in our pipeline - `consumer_task` has three input parameters:
- The output of `product_task`. Since this step only produces one output, we can reference it via `product_task.output`.
- The emoji output of our `emoji_task` step. See the emoji component defined above where we named the output parameters.
- Similarly, the `emoji_text` named output from the emoji component. In case our pipeline is passed text that doesn't correspond with an emoji, it'll use this text to construct a sentence.

- The last step in our pipeline - `consumer_task` has three input parameters:
- The output of `product_task`. Since this step only produces one output, we can reference it via `product_task.output`.
- The emoji output of our `emoji_task` step. See the emoji component defined above where we named the output parameters.
- Similarly, the `emoji_text` named output from the emoji component. In case our pipeline is passed text that doesn't correspond with an emoji, it'll use this text to construct a sentence.

```
[ ]: @dsl.pipeline(  
    name="hello-world",  
    description="An intro pipeline",  
    pipeline_root=PIPELINE_ROOT,  
)  
  
# You can change the `text` and `emoji_str` parameters here to update the pipeline output  
def intro_pipeline(text: str = "Vertex Pipelines", emoji_str: str = "sparkles"):  
    product_task = product_name(text)  
    emoji_task = emoji(emoji_str)  
    consumer_task = build_sentence(  
        product_task.output,  
        emoji_task.outputs["emoji"],  
        emoji_task.outputs["emoji_text"],  
    )
```


Step 4: Compile and run the pipeline

With your pipeline defined, you're ready to compile it. The following will generate a JSON file that you'll use to run the pipeline:

```
[ ]: compiler.Compiler().compile(  
    pipeline_func=intro_pipeline, package_path="intro_pipeline_job.json"  
)
```

Next, instantiate an API client:

```
[ ]: api_client = AIPlatformClient(  
    project_id=PROJECT_ID,  
    region=REGION,  
)
```

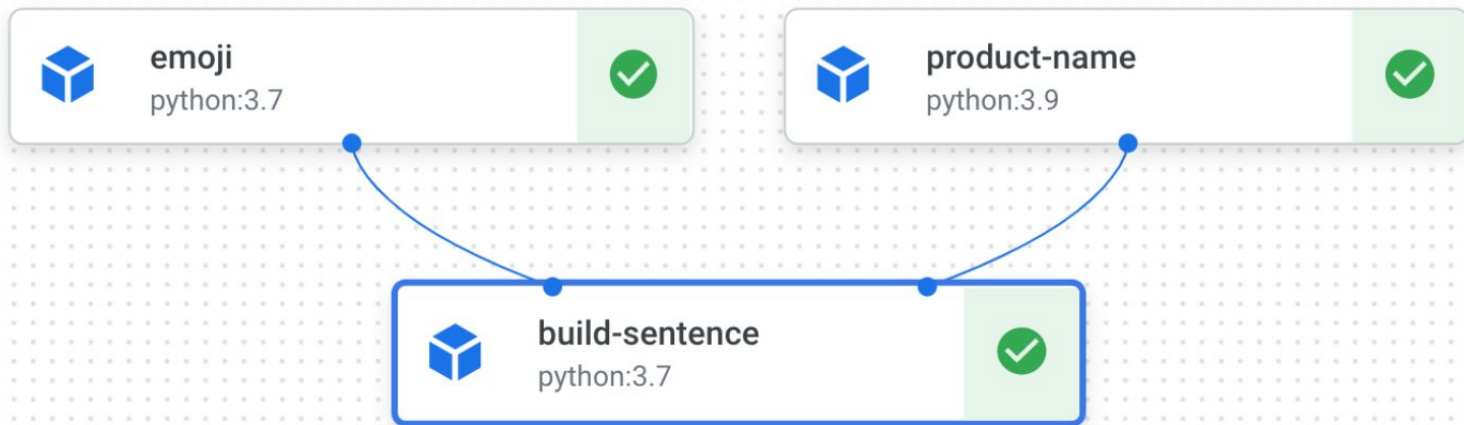
Finally, run the pipeline:

```
[ ]: # TODO  
response = api_client.create_run_from_job_spec(  
    job_spec_path="intro_pipeline_job.json",  
    # pipeline_root=PIPELINE_ROOT # this argument is necessary if you did not specify PIPELINE_ROOT as part of the pipeline definition.  
)
```

```
[17]: # TODO
response = api_client.create_run_from_job_spec(
    job_spec_path="intro_pipeline_job.json",
    # pipeline_root=PIPELINE_ROOT # this argument is necessary if you did not specify PIPELINE_ROOT as part of the pipeline definition.
)
```

See the Pipeline job [here](#).

Running the pipeline should generate a link to view the pipeline run in your console. It should look like this when complete:





hello-world-20220227080911



CLONE



STOP



DELETE

Runtime Graph

3/3 steps completed



Expand Artifacts

100%



Pipeline run analysis

SUMMARY

NODE INFO

Execution Info

Completed

[VIEW JOB](#)

[VIEW LOGS](#)

Display name	build-sentence
Name	build-sentence
Type	system.ContainerExecution
Duration	1 min 42 sec
Started	Feb 27, 2022, 12:11:11 AM
Completed	Feb 27, 2022, 12:12:53 AM

Input Parameters

Parameter	Type	Value
emoji	string	🌟
emojitext	string	sparkles
product	string	Vertex Pipelines

Output Parameters

Parameter	Type	Value
Output	string	Vertex Pipelines is 🌟

[Show debug panel](#)

emoji
python:3.7

product-name
python:3.9

build-sentence
python:3.7

Selecting one of the nodes
shows information here

Logs



Pipeline run proto

This is the JSON for the current run. Click anywhere in the text area to copy the response to clipboard.

```
{
  "name": "projects/333736501253/locations/us-central1/pipelineJobs/hello-world-20220227080911",
  "displayName": "hello-world-20220227080911",
  "createTime": "2022-02-27T08:09:12.157988Z",
  "startTime": "2022-02-27T08:09:17.442447Z",
  "updateTime": "2022-02-27T08:09:17.812351Z",
  "pipelineSpec": {
    "deploymentConfig": {
      "@type": "type.googleapis.com/ml_pipelines.PipelineDeploymentConfig",
      "executors": {
        "exec-build-sentence": {
          "container": {
            "image": "python:3.7",
            "command": [
              "sh",
              "-c",
              "\nif ! [ -x \"$(command -v pip)\" ]; then\n    python3 -m ensurepip || python3 -m ensu",
              "sh",
              "-ec",
              "program_path=$(mktemp -d)\nprintf \"%s\\n\" \"$0\" > \"$program_path/ephemeral_component.",
              "\nimport kfp\nfrom kfp.v2 import dsl\nfrom kfp.v2.dsl import *\nfrom typing import *\n",
              ],
            "args": [

```

☐ Line wrapping

You can also see the JSON for the entire run when you select the run from the Pipeline Interface.

CLOSE

run analysis

NODE INFO

1 min 49 sec

Feb 27, 2022, 12:09:17 AM

hello-world-20220227080911

hello-world

Environment Serverless

us-central1

ount 333736501253-

compute@developer.gserviceaccount.com

[View pipeline proto](#)

eters

meter values used for this run

Type

Value

string

sparkles

string

Vertex Pipelines