1. What is Apache Airflow and why is it used?

Answer:

Apache Airflow is an open-source platform used to programmatically author, schedule and monitor the work flow. Airflow helps you organize and automate a series of task(like running scripts, monitor data) into pipeline

2. What is Cloud Composer? How is it different from open-source Airflow?

Answer:

Cloud Composer is a managed service and orchestration service by Google Cloud built on Apache Airflow. GCP's managed Airflow: no need to manage infrastructure. Auto-scaling, built-in monitoring, IAM integration, versioned Airflow upgrades

3. Airflow operator examples?

Answer:

1.DummyOperator - Used as a placeholder or logical step.

```
from airflow.operators.dummy import DummyOperator
start = DummyOperator(
        task_id='start_pipeline',
        dag=dag
)
```

2.BashOperator - Executes a bash command or shell script.

```
from airflow.operators.bash import BashOperator
task = BashOperator(
        task_id='print_date',
        bash_command='date',
        dag=dag
)
```

3.PythonOperator - Runs Python functions directly in a task.

```
from airflow.operators.python import PythonOperator
def my_function():
   print("Running Python task")
task = PythonOperator(
        task_id='run_python_task',
        python_callable=my_function,
        dag=dag
)
```

4. BranchPythonOperator - Used for conditional branching logic.

```
from airflow.operators.python import BranchPythonOperator
def choose_path():
     return 'task_a' if condition else 'task_b'
branch = BranchPythonOperator(
        task_id='branch_task',
        python_callable=choose_path,
        dag=dag
    )
```

5. TriggerDagRunOperator - Triggers another DAG.

```
from airflow.operators.trigger_dagrun import TriggerDagRunOperator
trigger = TriggerDagRunOperator(
        task_id="trigger_another_dag",
        trigger_dag_id="other_dag_id",
        dag=dag
    )
```

6. GCSCreateBucketOperator - Creates a GCS bucket.

```
from airflow.providers.google.cloud.operators.gcs import GCSCreateBucketOperator
create_bucket = GCSCreateBucketOperator(
                        task_id='create_gcs_bucket',
                         bucket_name='my-new-bucket',
                        location='US',
                        storage_class='STANDARD',
                        dag=dag,
        )
```

7. BigQueryCreateEmptyDatasetOperator - Creates an empty BigQuery dataset**.**

```
from airflow.providers.google.cloud.operators.bigquery import BigQueryCreateEmptyDatasetOperator
create_dataset = BigQueryCreateEmptyDatasetOperator(
                        task_id='create_bq_dataset',
                        dataset_id='my_new_dataset',
                        project_id='my-gcp-project',
                        location='US',
        )
```

8. BigQueryCreateEmptyTableOperator - Creates an empty table with a schema in BigQuery.

```
from airflow.providers.google.cloud.operators.bigquery import BigQueryCreateEmptyTableOperator
create_table = BigQueryCreateEmptyTableOperator(
                        task_id='create_bq_table',
                        dataset_id='my_new_dataset',
                        table_id='my_table',
                        project_id='my-gcp-project',
                        schema_fields=[
                                {'name': 'id', 'type': 'STRING', 'mode': 'REQUIRED'},
                                {'name': 'amount', 'type': 'FLOAT', 'mode': 'NULLABLE'},
                                {'name': 'created_at', 'type': 'TIMESTAMP', 'mode': 'NULLABLE'},
                        ],
                        location='US'
        )
```

9. BigQueryInsertJobOperator- Runs SQL queries or jobs like DDL/DML, load, or export operations.

Load operation:

```
from airflow.providers.google.cloud.operators.bigquery import BigQueryInsertJobOperator
load_data_bucket_to_table=BigQueryInsertJobOperator(
                                task_id='load_data_bucket_to_table',
                                configuration={
                                        "load":{
                                                "destinationTable":{
                                                        "projectId": "project-name",
                                                        "datasetId": "dataset",
                                                        "tableId": "table"
                                                },
                                                "sourceUris":["gs://file_name"],
                                                "sourceFormat": "CSV",
                                                "skipLeadingRows": 1,
                                                "writeDisposition":"WRITE_TRUNCATE"
                                                }
                                        },
                                project_id='project-name'
        )
```

DML Operation:

```
run_query = BigQueryInsertJobOperator(
task_id='run_bq_query',
configuration={
        "query": {
                "query": "SELECT * FROM
                        `my-gcp-project.my_dataset.source_table`
                        WHERE amount > 1000",
                 "useLegacySql": False,
                "writeDisposition": "WRITE_TRUNCATE",
                }
 },
project_id='project-name'
)
```

4. What is a DAG in Airflow?

Answer: DAG (Directed Acyclic Graph): defines the workflow structure — tasks and their execution order.

5. How do you schedule a DAG in Airflow?

Answer:

--Using schedule_interval (e.g., @daily, '0 12 * * *', None for manual).

--Can also define start_date, catchup, and timezone.

6. What are the components of Airflow DAG?

Answer:

DAG: defines metadata

Operators: units of work (e.g., BashOperator, PythonOperator, BigQueryOperator)

Tasks: instantiations of operators

Dependencies: task flow defined using >> or set_upstream()

7. How do you set dependency in DAG?

Answer: In Airflow, setting task dependencies defines the order of execution.

Using >> and << Operators (Recommended)-- task1 >> task2 or task2 << task1

Using set_downstream() and set_upstream()--task1.set_downstream(task2) or task2.set_upstream(task1)

Multiple Dependencies -- task1 >> [task2, task3]

8. Which is Orchestration tool have you worked on?

Answer:

"I have worked with Apache Airflow, specifically using Cloud Composer on Google Cloud Platform for orchestration. Airflow allowed me to automate and schedule data pipelines efficiently. I used various operators like BigQueryInsertJobOperator, GCSOperator, and PythonOperator to build end-to-end pipelines.

I also used features like:

- DAG dependencies to define task execution order,
- XComs for passing data between tasks,
- Sensor operators for waiting on external triggers (like file arrival in GCS),
- and Airflow Variables and Connections for configuration management.

9. What are Trigger rules and DAG dependency?

Answer:

DAG Dependencies: DAG dependencies define the **order** in which tasks should be executed.

task1 >> task2 >> task3

This means: task1 runs first >> Then task2 >> Then task3

Trigger Rules: By default, a task only runs if all upstream tasks succeed. But you can change this behaviour using Trigger Rules**.**

Trigger Rule	Description
all_success (default)	Run only if all upstream tasks succeed
all_failed	Run if all upstream tasks **fail**
all_done	Run when all upstream tasks are done, regardless of success or failure
one_success	Run if **any one** upstream task succeeds
one_failed	Run if **any one** upstream task fails
none_failed	Run if **no** upstream task failed
none_skipped	Run if **no** upstream task was skipped

```
from airflow.operators.dummy import DummyOperator
from airflow import DAG
from airflow.utils.trigger_rule import TriggerRule
from datetime import datetime
with DAG(
        'Trigger_rule_example',
         start_date=datetime(2023, 1, 1),
        schedule_interval='@daily',
         catchup=False)
as dag:
        start = DummyOperator(task_id='start')
        task_a = DummyOperator(task_id='task_a')
        task_b = DummyOperator(task_id='task_b')

        final_task = DummyOperator(
                        task_id='final_task',
                         trigger_rule=TriggerRule.ONE_SUCCESS   #Trigger Rule
                )

        start >> [task_a, task_b] >> final_task       # Dag Dependency
```

10. List of tasks performed in Airflow?

Answer:

Typical Tasks Performed in Airflow

1.DAG Design & Configuration

2. Task Dependency Management

3. Operators Used

4. Scheduling & Monitoring

5. Data Pipeline Orchestration

---Ingest data from GCS to BigQuery

---Run SQL transformations in BigQuery

---Chain multi-step pipelines: extract → transform → load (ETL)

6. Communication Between Tasks

7. Observability

8. Testing & Debugging

9.Performance & Scalability

11. What are the default arguments in Airflow Dag?

Answer:

In Apache Airflow, default_args is a Python dictionary that defines default settings for tasks in a DAG

```
default_args = {
        'owner': 'airflow',
        'depends_on_past': False,
        'start_date': datetime(2024, 1, 1),
        'email': ['alert@example.com'],
        'email_on_failure': True,
        'email_on_retry': False,
        'retries': 2,
        'retry_delay': timedelta(minutes=5),
    }
```

Commonly Used Fields in default_args

Parameter	**Description**
Owner	Person/team responsible for the DAG
depends_on_past	If True, task runs only if previous run succeeded
start_date	When the DAG should start scheduling
email	Email address to notify on failures/retries
email_on_failure	Send email when task fails
email_on_retry	Send email when task is retried
retries	Number of times to retry on failure
retry_delay	Time to wait between retries (e.g., timedelta(minutes=5))

12. Bounded and Unbounded collection in Airflow?

Answer:

Bounded Collection : A dataset or data source that has a clearly defined beginning and end.

Example: A batch file in GCS, a static CSV, a daily BigQuery export, or a snapshot table**.**

Unbounded Collection : A continuously growing dataset with no defined end — often related to streaming data.

Example: Pub/Sub messages, Kafka topics, continuously updated logs or real-time APIs.

13. How to Integrate Airflow with BigQuery?

Answer:

Integrating Apache Airflow with BigQuery allows you to automate data pipelines that interact with Google's data warehouse. This is typically done using Airflow's BigQuery Operators and Google Cloud Connection.

14. What is Catch-up and what would be the default value?

Answer:

Catch-up in Apache Airflow controls whether past DAG runs should be executed if the scheduler was paused or off for a period of time.

When catchup=True (default behaviour), Airflow will run all missed DAG runs between start_date and current date based on your schedule interval.

If today is **May 17, 2025**, and the DAG's start_date is **Jan 1, 2023**:

- **If catchup=True**: Airflow will try to run **every daily task** from Jan 1 to May 17 (over 800 runs).
- **If catchup=False**: Only the most recent **current** scheduled DAG run will execute (no backlog).

Set catchup=False for real-time or event-driven pipelines, and only use catchup=True for batch/historical reprocessing.

15. Airflow issues and what are the pre-requests for airflow?

Answer:

Issue	Description	Resolution
Scheduler not triggering DAGs	DAG file syntax error, DAG not in the dags_folder, or paused DAG.	Check logs, validate DAG syntax, and unpause the DAG.
Task stuck in scheduled or queued	Executor misconfiguration or insufficient resources.	Verify executor setup and available system resources.
Import errors in DAG	Missing Python packages or syntax errors.	Install missing libraries using pip and validate DAG script.
XCom too large	Pushing large objects (e.g., DataFrames) into XCom.	Only push light, serializable data (like IDs or file paths).
Too many DAGS runs or tasks in the queue	Due to catchup=True with an old start_date.	Set catchup=False to avoid unnecessary backlog.
Broken webserver UI or crashing	Caused by malformed DAG or excessive XCom entries.	Clean XComs, remove broken DAGs, restart webserver.
Timezone issues	Scheduled time not aligning with expected run time.	Set correct timezone in airflow.cfg or DAG.
Task retry not working	Retry policy misconfigured.	Set retries, retry_delay, and ensure tasks are not marked as success.

16. Write a Dag for GCSToBigQueryOperator?

Answer:

```
load_gcs_to_bq = GCSToBigQueryOperator(
                    task_id='load_gcs_to_bigquery',
                    bucket='your-gcs-bucket-name',
                    source_objects=['path/to/your/file.csv'],  # can be a list
                    destination_project_dataset_table='your-project.your_dataset.your_table',
                    schema_fields=[
                        {'name': 'id', 'type': 'INTEGER', 'mode': 'REQUIRED'},
                        {'name': 'name', 'type': 'STRING', 'mode': 'NULLABLE'},
                        {'name': 'created_at', 'type': 'TIMESTAMP', 'mode': 'NULLABLE'},
                         ],
                    source_format='CSV',
                    skip_leading_rows=1,
                    write_disposition='WRITE_TRUNCATE',  # or WRITE_APPEND
                    create_disposition='CREATE_IF_NEEDED',
                    autodetect=False,
                )
```

17. How job meta data handling in airflow?

Answer:

In Apache Airflow, job metadata (i.e., information about DAG runs, task status, execution time, logs, etc.) is handled by its metadata database, which is typically backed by a relational database like PostgreSQL or MySQL.

18. Explain workflow in airflow?

Answer:

Airflow workflow = **DAG definition → Scheduler Triggers the DAG→ Executor Runs the Tasks→ Metadata is Stored→ Logs and Monitoring**

Airflow vs Dataflow (Comparison Table)

Feature / Purpose	**Airflow**	**Dataflow**
Type	Orchestration Tool	Data Processing Tool (based on Apache Beam)
Use Case	Managing and scheduling workflows (ETL orchestration)	Processing large-scale data (batch or stream)
Execution Model	DAG-based task scheduling	Pipeline-based data computation
Batch / Streaming	Mostly batch (with support for retries, triggers)	Supports both batch and real-time streaming
Language	Python	Java / Python (via Apache Beam SDK)
Parallelism	Limited to task concurrency	Scales automatically with parallel workers
Stateful Processing	No	Yes (especially for stream processing)
Data-Aware	No – Just orchestrates steps	Yes – Processes and transforms data
Integration with GCP	Works with GCS, BigQuery, Dataflow, etc.	Deeply integrated with Pub/Sub, BigQuery, GCS
Managed Service	Cloud Composer (GCP's managed Airflow)	Fully managed (Google Cloud Dataflow)
Use With	Great for coordinating jobs across tools	Ideal for heavy transformations and streaming
Retries, SLA, Alerting	Built-in	Must be manually implemented in Beam code

Composer vs Airflow (Comparison Table)

Feature / Category	**Apache Airflow**	**Cloud Composer (GCP)**
Type	Open-source tool	Managed service for Airflow on GCP
Deployment	Self-hosted (manual setup on VM, Kubernetes, etc.)	Fully managed by Google (auto-scaling, updates)
Environment Setup	Manual: install Airflow, set up DB, scheduler, etc.	GCP handles infra setup and config
Integration	Generic (plugins needed for cloud services)	Built-in integration with GCP services (GCS, BQ, Pub/Sub)
Scaling	You must manage scalability	Composer auto-scales workers and components
Monitoring & Logs	Basic UI + custom setup for observability	GCP-native monitoring with Cloud Logging
Upgrades	Manual upgrades	GCP handles version upgrades (you choose version)
Security & IAM	Self-managed authentication	Integrated with Google IAM, VPC, secrets, etc.
Costs	You pay for infra (can be cheaper, more effort)	Pay-as-you-go for Composer environment (more expensive)