	Hope Foundation's **Finolex Academy of Management and Technology, Ratnagiri** **Department of Computer Science and Engineering (AIML)**		
Subject name: Software Engineering and Project Management Lab			Subject Code: CSL603
Class	TE CSE	Semester –VI(CBCGS)	Academic year: 2023-24
Name of Student	Patil Anuja Hemant		**QUIZ Score :**
Roll No	41	Experiment No.	01

Title: **To understand DevOps practices which aim to simplify Software Development Life Cycle.**

1. Lab objectives applicable:

LOB1: To understand DevOps practices which aim to simplify Software Development Life Cycle.

2. Lab outcomes applicable:

LOB1: To understand the fundamentals of DevOps engineering and be fully proficient with DevOps terminologies, concepts, benefits, and deployment options to meet our business requirements.

3. Learning Objectives:

1. To know the daily activities of DevOps Engineers.
2. To know the softwares/ automated tools used to do DevOps engineering.

4. Practical applications of the assignment/experiment:

5. Prerequisites:

1. The knowledge of Software Development Life Cycle (SDLC).

6. Minimum Hardware Requirements:

1.

7. Software Requirements:

1.

8. Quiz Questions (if any): (Online Exam will be taken separately batch-wise, attach the certificate/ Marks obtained)

9. Experiment/Assignment Evaluation:

Sr. No.	Parameters	Marks obtained	Out of
1	Technical Understanding (Assessment may be done based on Q & A **or** any other relevant method.) Teacher should mention the other method used -		6
2	Lab Performance		2
3	Punctuality		2
Date of performance (DOP)		**Total marks obtained**	**10**

Signature of Faculty

10. Theory:

Understanding daily activities of devops engineer:A DevOps engineer plays a crucial role in streamlining and enhancing the software development cycle. The daily activities iclude:

1. **Continuous Integration/Continuous Deployment (CI/CD):**
   Manage CI/CD pipelines to automate the software delivery process.
   Monitor and troubleshoot build and deployment failures.
2. **Collaboration and Communication:**
   Work closely with development, operations, and other cross-functional teams.
   Communicate and collaborate with team members to ensure a smooth flow of information and efficient problem-solving.
3. **Infrastructure as Code (IaC):**
   Write and manage infrastructure code using tools like Terraform, Ansible, or CloudFormation.
   Maintain and update infrastructure configurations to meet evolving requirements.
4. **Automation and Scripting:**
   Develop and maintain automation scripts for deployment, configuration management, and other repetitive tasks.
   Implement and optimize continuous integration and continuous delivery (CI/CD) pipelines.
5. **Monitoring and Logging:**
   Set up and maintain monitoring tools to track system performance, application health, and security.
   Analyze logs and metrics to identify issues and optimize system performance.

Knowing software/automated tools used in DevOps Engineering:
DevOps engineers use a variety of software and automated tools to streamline processes, enhance collaboration, and automate various aspects of software development and IT operations.

1. **Version Control (VC):** Git is a widely-used distributed version control system enabling collaborative code management and tracking changes efficiently.
2. **Continuous Integration (CI):** Jenkins automates the building, testing, and integration of code changes to ensure a consistent and reliable software delivery process.
3. **Configuration Management:** Ansible automates configuration management and application deployment, ensuring consistent server states across the infrastructure.
4. **Infrastructure as Code (IaC):** Terraform simplifies infrastructure provisioning and management by using declarative configuration files to define infrastructure components.
5. **Containerization:** Docker provides containerization, packaging applications and dependencies into portable units for efficient deployment across diverse environments.
6. **Monitoring:** Prometheus monitors and alerts on infrastructure and application metrics, offering a powerful open-source solution for observability.
7. **Logging:** ELK Stack (Elasticsearch, Logstash, Kibana) processes and visualizes log data, aiding in centralized log management and analysis.

Apache Airflow in the DevOps Toolchain:

Apache Airflow plays a crucial role in the DevOps toolchain by serving as an orchestrator for workflows and automating the execution of tasks. Its integration with other DevOps tools enhances collaboration, continuous integration, continuous delivery, and overall operational efficiency. Here's how Apache Airflow fits within the broader landscape of DevOps tools:

1. **Workflow Automation:**
   Airflow orchestrates complex workflows, enabling the automation of tasks, dependencies, and scheduling.
   Integration: Works seamlessly with other DevOps tools to coordinate and automate end-to-end processes.

2. **Data Pipelines:**
   Airflow is often utilized for building and managing data pipelines. DevOps engineers leverage its capabilities to design workflows that involve data processing, transformation, and loading (ETL), ensuring efficient and reliable data management.

3. **Integration with Other Tools:**
   Airflow is known for its extensibility and flexibility. It integrates seamlessly with various tools and platforms, making it a valuable asset in a DevOps environment. For instance, it can be integrated with containerization tools like Docker and orchestration tools like Kubernetes.

4. **Dependency Management:**
   In the context of DevOps, where dependencies between different stages of the SDLC are crucial, Airflow excels in managing dependencies within workflows. It ensures that tasks are executed in the correct order and according to specified conditions.

5. **Monitoring and Logging:**
   Apache Airflow provides a user-friendly web interface for monitoring and managing workflows. DevOps teams can easily track the progress of tasks, view logs, and troubleshoot any issues that may arise during the execution of workflows.

6. **Dynamic and Scalable:**
   DevOps often requires solutions that can scale with the needs of the organization. Airflow's dynamic nature allows for the creation of workflows that adapt to changing requirements, making it suitable for a wide range of scenarios.

7. **Community and Support:**
   Being an open-source project, Apache Airflow benefits from a vibrant community. This community-driven aspect ensures continuous development, support, and a wealth of resources for DevOps practitioners.

In Apache Airflow, the combination of Cron expressions and the scheduler is integral to how tasks are scheduled and executed within workflows.

1. **Defining Schedules with Cron Expressions:**In Apache Airflow, when you create a Directed Acyclic Graph (DAG), you can specify a schedule_interval parameter using a Cron expression. This expression determines when the DAG or individual tasks within the DAG should be executed.
2. **The Role of the Scheduler:**The scheduler component in Apache Airflow continually monitors DAGs for changes and new schedules. It uses the specified Cron expressions to determine when a DAG or task should be triggered. When the scheduled time arrives, the scheduler initiates the execution of the associated tasks

3. **Dynamic Scheduling with Cron:**Cron expressions provide flexibility in defining dynamic schedules. Whether you need a task to run every minute, hourly, daily, or on a more complex schedule, Cron expressions allow you to articulate these requirements precisely.

4. **Task Execution and Dependencies:**The scheduler ensures that tasks are executed in the correct order based on dependencies defe the DAG. It uses the Cron expression to trigger the tasks when the specified time conditions are This enables the orchestration of complex workflows within the DevOps pipeline.

5. **Handling Time Zones:**Apache Airflow allows you to set the time zone for your DAGs, ensuring that the Cron expressions align with the desired time zone. This is essential when dealing with distributed teams or systems in different geographical locations

Cron Expression in Airflow and DevOps:

Understanding Cron Expression:

In the context of Apache Airflow and DevOps, a Cron expression is a time-based scheduling format widely used to define the frequency and timing of recurring tasks. The Cron syntax consists of fields representing minute, hour, day of the month, month, and day of the week, allowing for precise scheduling of jobs.

Key Components of a Cron Expression:

- Minute (0-59):Specifies the minute of the hour when the task should run. ● Hour (0-23): Indicates the hour of the day when the task is scheduled ● Day of the Month (1-31): Specifies the day of the month for the task.
- Month (1-12 or JAN-DEC): Indicates the month in which the task is scheduled.

- Day of the Week (0-6 or SUN-SAT): Specifies the day of the week for the task.

Use of Cron Expression in Apache Airflow:

1. **Scheduling Workflows:**
   Apache Airflow uses Cron expressions to schedule the execution of workflows or Directed Acyclic Graphs (DAGs). By defining a Cron schedule in the DAG definition, DevOps engineers can precisely control when specific tasks within the workflow should be triggered.
2. **Automating Repetitive Tasks:**
   Cron expressions are instrumental in automating repetitive tasks within DevOps workflows. This automation ensures that routine processes, such as data backups, system maintenance, or periodic data processing, are executed at specified intervals without manual intervention.
3. **Time-Based Dependencies**:
   Cron expressions are used to define time-based dependencies between tasks. In a workflow, one task might depend on the completion of another task, and Cron scheduling ensures that these tasks are executed in the correct sequence based on time requirements.
4. **Dynamic Scaling:**
   With Cron expressions, DevOps teams can scale workflows dynamically by adjusting the scheduling parameters. This flexibility allows for easy adaptation to changing business requirements and workloads.
5. **Monitoring and Logging:**
   Cron expressions play a crucial role in the monitoring and logging of scheduled tasks. DevOps engineers can track the execution of tasks over time, identify potential issues, and analyse logs to ensure the reliability and efficiency of the scheduled workflows.

11. Installation Steps / Performance Steps and Results –

Apache airflow doesn't work on windows, but on Linux. So, we can use either Linux in virtual box / VMware or use windows subsystem for Linux (wsl). If you have Linux as virtual machine, then you can skip wsl installation process.

wsl is feature of windows that allows developers to run Linux environment without need for separate virtual machine or dual booting. it works in windows command prompt.

```
wsi-upgrade
wsl-install // (install ubuntu by default)
```

After successful installation, enter new username and password and then restart terminal. You can start the Linux shell by either using: ws! command in cmd or open it from start menu using the Linux distribution name you installed (ubuntu by default) python3 will be already installed. You need to install pip. mostly your present working directory

(pwd) will be /home/<username>

sudo apt install python3-pip

Then install virtual environment library

pip3 install virtualenv

Then create venv to work on airflow

virtualenv <name_of_env>

Then activate the environment source

<name_of_env>/bin/activate

Then install Apache airflow (version 2.8.1) pip3 install

apache-airflow pip3 install apache-airflow [gcp,sentry,

statsd) pip install "apache-airflow [celery]==2.8.1"--

constraint

"https://raw.githubusercontent.com/apache/airflow/constraints-2.8.1/constraints-3.10.txt"

pip install apache-airflow[cncf.kubernetes]

To set environment variable for airflow, create folder named airflow, then use following commands to set the variable sudo vim ~/.bash_profile // (create bash file to set environment variable permanently, else

it vanishes after you restart terminal)

Press i to get into edit mode, and then go to end of file using arrow keys. (if there is no such file beforehand, it will create new empty file, so no need to go to end) export APACHE AIRFLOW=~/airflow

Then press esc to come out of edit mode, and type :wq to exit. then activate the file (variable)

source ~/.bash_profile

Then go to the airflow folder

cd airflow

To view airflow version

airflow version

Then create airflow database

airflow db init

Create user profile for airflow (needed to login and manage to workspace) airflowusers create role

Admin -username admin --email <your_email> --firstname

<your_name>--lastname <your_name>--password <your_password>

To see if user is created

airflow users list

Start webserver airflow webserver // (starts at port 8080 by default, if it is busy, you can use

another port:

airflow webserver -p <port >)

Then start another terminal, activate the virtual environment using: source

<name_of_env>/bin/activate

Start scheduler of airflow

airflow scheduler

Then go to browser and open 'localhost:8080' (or any other port you are using), you will need to enter credentials you defined while creating user profile for airflow

There you will see many DAGs in airflow environment. (Directed Acyclic Graph is core concept of airflow, collecting tasks together, organized with dependencies and relationships to say how they should run)

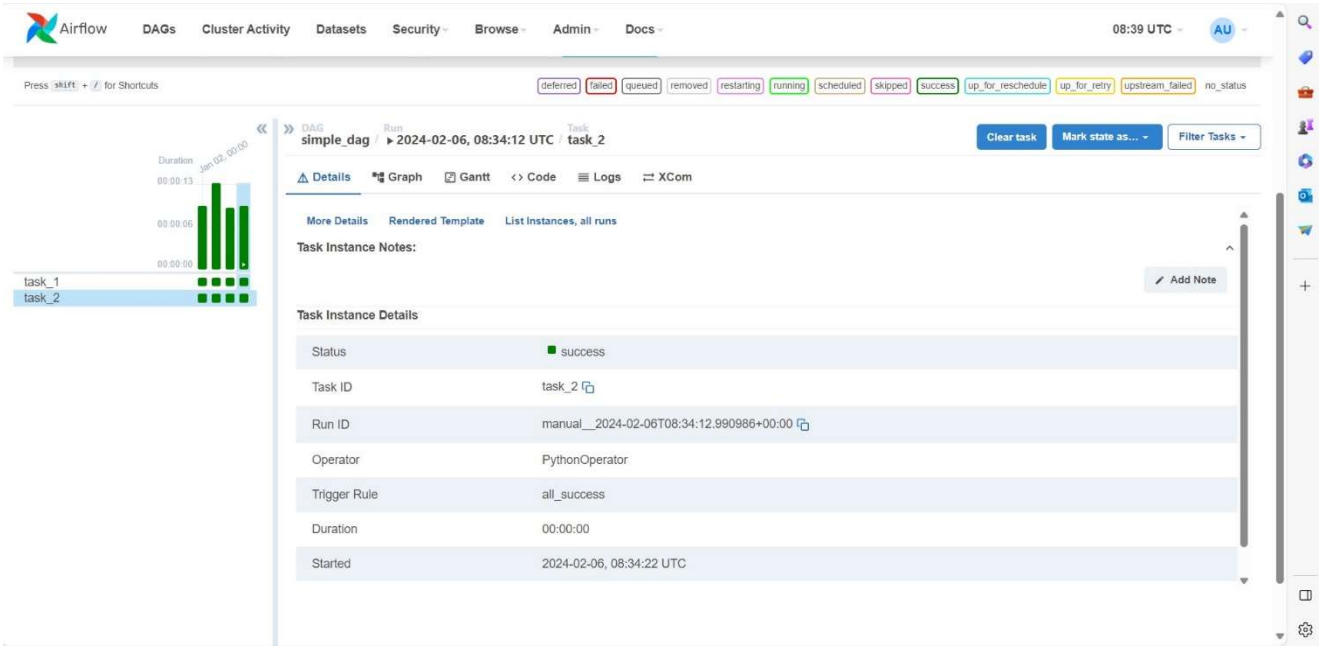
Create folder named 'dags' in airflow folder (mkdir dags) where you will save your own DAGS. to create your own DAG, you need code editor, if you have visual studio/ vs code, much better. Go to dags folder (cd dags) and open vs code from there (code) then create python file hello_world_dag.py' and write the given code and save the file.

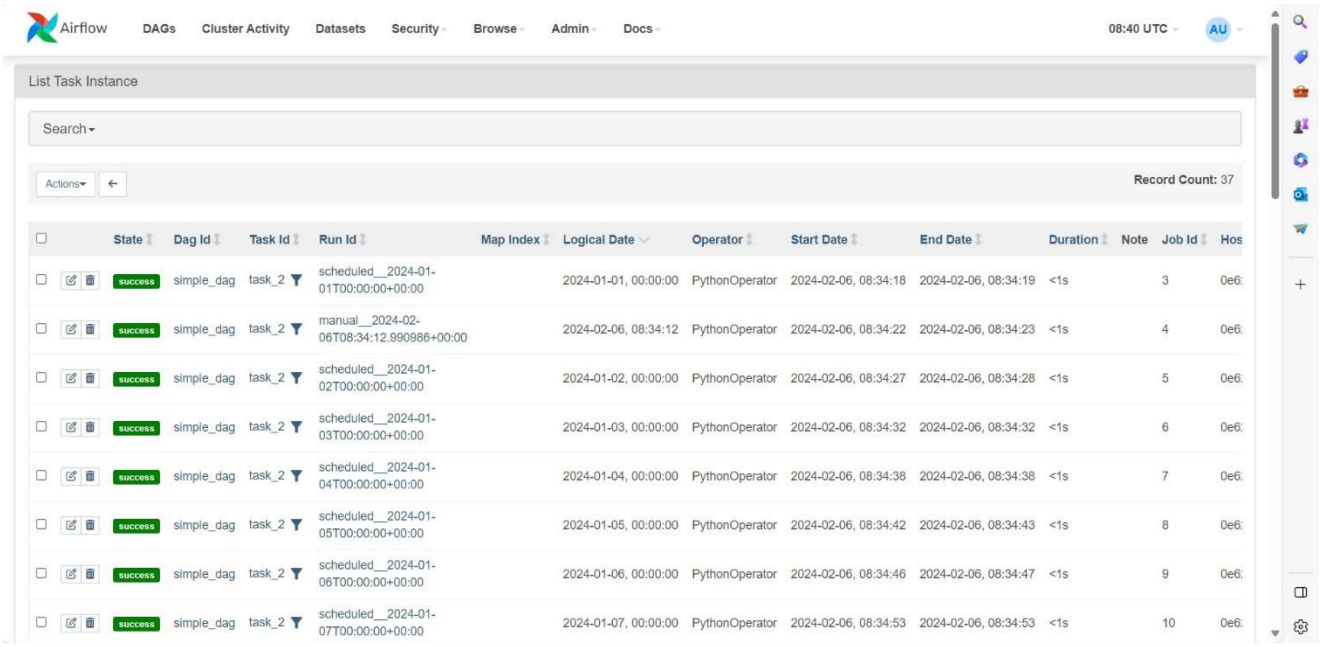
Then reload the webserver page. and you will be able to notice the newly created DAG. Then enable it using toggle button, and then open the DAG by clicking on name. You will see the first run green if successful, the above DAG will trigger at each hour. So, wait for 1 hour and you will see another run of the task.

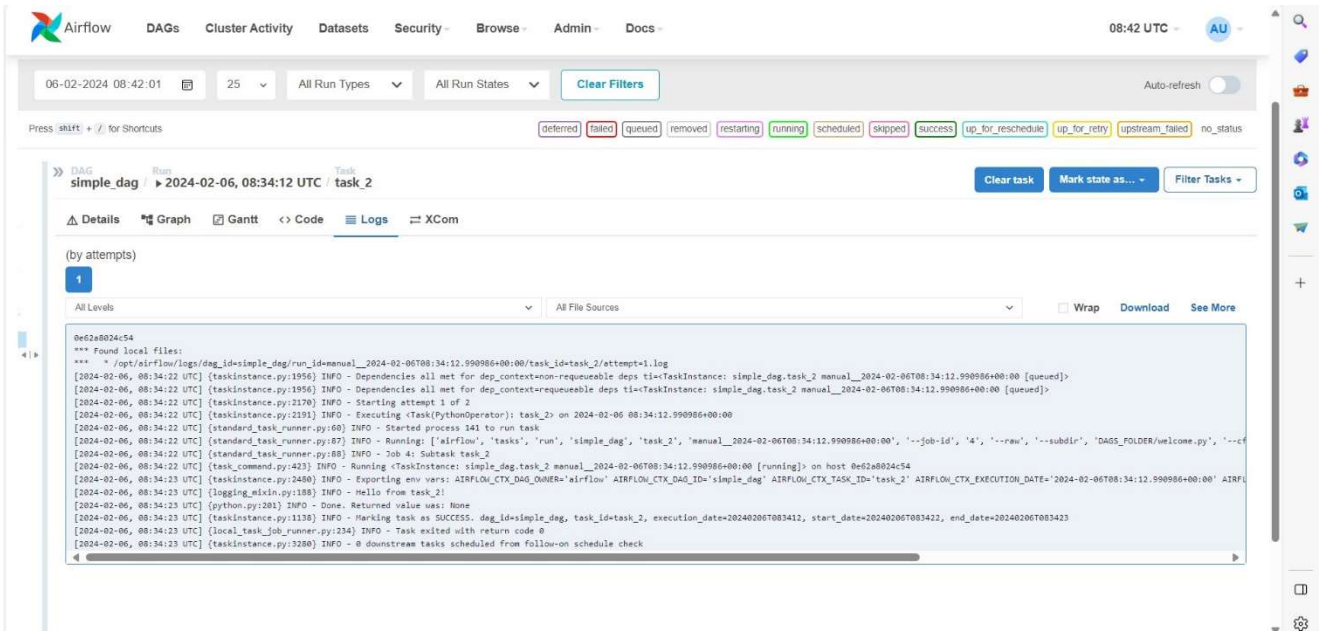
Source code:

```
from airflow import DAG from
airflow.operators.python import PythonOperator from
datetime import datetime def helloWorld():
        print("Hello World")

with DAG(dag_id="hello_world_dag",
        start_date=datetime(2021,1,1),
        schedule_interval="@hourly",
        catchup=False) as dag:

        task1 = PythonOperator( task_id="hello_world", python_callable-helloWorld)

task1
```

Output:


Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
08:39 UTC
AU
Press shift + / for Shortcuts
deferred
failed
queued
removed
restarting
running
scheduled
skipped
success
up_for_reschedule
up_for_retry
upstream_failed
no_status
Duration
00:00:13
00:00:06
00:00:00
task_1
task_2
DAG
simple_dag
Run
2024-02-06, 08:34:12 UTC
Task
task_2
Clear task
Mark state as...
Filter Tasks
Details
Graph
Gantt
Code
Logs
XCom
More Details
Rendered Template
List Instances, all runs
Task Instance Notes:
Add Note
Task Instance Details
Status
success
Task ID
task_2
Run ID
manual__2024-02-06T08:34:12.990986+00:00
Operator
PythonOperator
Trigger Rule
all_success
Duration
00:00:00
Started
2024-02-06, 08:34:22 UTC


Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
08:40 UTC
AU
List Task Instance
Search
Actions
Record Count: 37
State
Dag Id
Task Id
Run Id
Map Index
Logical Date
Operator
Start Date
End Date
Duration
Note
Job Id
Hos
success simple_dag task_2 scheduled__2024-01-01T00:00:00+00:00 2024-01-01, 00:00:00 PythonOperator 2024-02-06, 08:34:18 2024-02-06, 08:34:19 <1s 3 0e6
success simple_dag task_2 manual__2024-02-06T08:34:12.990986+00:00 2024-02-06, 08:34:12 PythonOperator 2024-02-06, 08:34:22 2024-02-06, 08:34:23 <1s 4 0e6
success simple_dag task_2 scheduled__2024-01-02T00:00:00+00:00 2024-01-02, 00:00:00 PythonOperator 2024-02-06, 08:34:27 2024-02-06, 08:34:28 <1s 5 0e6
success simple_dag task_2 scheduled__2024-01-03T00:00:00+00:00 2024-01-03, 00:00:00 PythonOperator 2024-02-06, 08:34:32 2024-02-06, 08:34:32 <1s 6 0e6
success simple_dag task_2 scheduled__2024-01-04T00:00:00+00:00 2024-01-04, 00:00:00 PythonOperator 2024-02-06, 08:34:38 2024-02-06, 08:34:38 <1s 7 0e6
success simple_dag task_2 scheduled__2024-01-05T00:00:00+00:00 2024-01-05, 00:00:00 PythonOperator 2024-02-06, 08:34:42 2024-02-06, 08:34:43 <1s 8 0e6
success simple_dag task_2 scheduled__2024-01-06T00:00:00+00:00 2024-01-06, 00:00:00 PythonOperator 2024-02-06, 08:34:46 2024-02-06, 08:34:47 <1s 9 0e6
success simple_dag task_2 scheduled__2024-01-07T00:00:00+00:00 2024-01-07, 00:00:00 PythonOperator 2024-02-06, 08:34:53 2024-02-06, 08:34:53 <1s 10 0e6



12. Learning Outcomes Achieved

1. Students are able to schedule script using Airflow software.

13. Conclusion:

1. **Applications of the studied technique in industry** To schedule scripts.
   Enables the fast delivery of project.
2. **Engineering Relevance**
   DevOps engineer acts as a bridge between development and operational team for speedy delivery of project.
3. **Skills Developed**
   Learned cron expression, Airflow scheduler.

14. References:

1. https://www.freecodecamp.org/news/install-apache-airflow-on-windows-without-docker/
2. https://docs.oracle.com/cd/E12058_01/doc/doc.1014/e12030/cron_expressions.htm