OPEN ACCESS
J. OF INISTA, VOL. 4, NO. 2, PP.001-012, MEI 2022

ISSN : 2622-8106 (ONLINE)
DOI: 10.20895/INISTA.V4I2

Journal of Informatics, Information System, Software Engineering and Applications

Akuisisi Data Prediksi Curah Hujan Secara Periodik Menggunakan Apache Airflow

Erwin Eko Wahyudi[#1], Muhammad Auzan[*2], Andi Dharmawan[*3], Danang Eko Nuryanto[*4], Nanang Susyanto[†5], Guruh Samodra[*6], Danang Sri Hadmoko[*7]

[1,2,3]*Departemen Ilmu Komputer dan Elektronika, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Gadjah Mada*
Gedung C Lantai 4, Sekip Utara, Yogyakarta, 55281, Indonesia
[4]*Pusat Penelitian dan Pengembangan, Badan Meteorologi, Klimatologi, dan Geofisika*
Jl. Angkasa I No 2, Kemayoran, Jakarta Pusat, 10720, Indonesia
[5]*Departemen Matematika, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Gadjah Mada*
Sekip Utara, Yogyakarta, 55281, Indonesia
[6,7]*Departemen Geografi Lingkungan, Fakultas Geografi, Universitas Gadjah Mada*
Sekip Utara BLS 21, Yogyakarta, 55281, Indonesia

[1]erwin.eko.w@ugm.ac.id
[2]muhammadauzan@ugm.ac.id
[3]andi_dharmawan@ugm.ac.id
[4]danang_eko@bmkg.go.id
[5]nanang_susyanto@ugm.ac.id
[6]guruh.samodra@ugm.ac.id
[7]hadmoko@ugm.ac.id

accepted on 17-05-2022

Abstrak

Akuisisi data, bertujuan untuk mengambil data awal, merupakan salah satu tahapan dalam metodologi penambangan data. Data awal akan diproses menjadi data akhir yang digunakan untuk proses pemodelan, seperti pembuatan model untuk memprediksi potensi terjadinya tanah longsor. Data prediksi curah hujan yang disediakan oleh Badan Meteorologi, Klimatologi, dan Geofisika (BMKG) dapat digunakan untuk pemodelan tersebut. Data akan disimpan di komputer lokal dengan menggunakan alat atau aplikasi otomasi yang bernama Apache Airflow. Proses akuisisi data dari *server* BMKG ke komputer lokal dijalankan secara otomatis dalam dua kali sehari, yaitu pada pukul 00.00 dan 12.00. Terdapat dua *task* yang dibuat di *Directed Acyclic Graph* (DAG) untuk proses ini, yaitu *task* pertama sebagai sensor ketersediaan data dan *task* kedua yang melakukan proses utama. Status dari DAG pada Apache Airflow juga dapat diketahui secara cepat, misalnya status telah berhasil, gagal, atau sedang berjalan. Apache Airflow juga menyediakan *log* yang dapat diakses untuk mengetahui alasan kegagalan suatu *task*. Hasil dari penelitian ini adalah terdapat *pipeline* pada aplikasi otomasi Apache Airflow untuk membantu proses akuisisi data secara periodik.

Keywords: Akuisisi data, curah hujan, Apache Airflow, otomasi.

I. PENDAHULUAN

AKUISISI data merupakan salah satu langkah atau tahapan dalam metodologi penambangan data (*data mining*) [1]. Sesuai dengan namanya, proses akuisisi data merupakan proses pengumpulan data awal yang dibutuhkan dalam proyek penambangan data. Data awal tersebut selanjutnya akan diubah ke data akhir setelah melalui beberapa tahapan analisis data dan persiapan data. Data akhir tersebut nantinya akan digunakan dalam proses pemodelan yang akan menghasilkan suatu model, seperti model untuk memprediksi potensi terjadinya tanah longsor di Provinsi Daerah Istimewa Yogyakarta, Indonesia.

Terdapat beberapa faktor yang mempengaruhi terjadinya tanah longsor, seperti curah hujan [2]. Ketika hujan semakin deras, curah hujan akan semakin tinggi. Begitu pun ketika hujannya semakin lama, maka curah hujan akan semakin tinggi. Semakin tinggi curah hujan, maka peluang terjadinya tanah longsor akan semakin besar. Infiltrasi air hujan dapat memicu longsor karena tanah menjadi jenuh dan meningkatkan tekanan air pori dalam tanah [3]. Sebagai contoh, hujan deras sebesar 235.97 mm dalam waktu 37 jam akibat Siklon Tropis Cempaka dapat memicu longsor sebanyak 743 di Kabupaten Pacitan pada tahun 2017 [4]. Dengan demikian, ketika akan dibuat model untuk memprediksi potensi terjadinya tanah longsor, data prediksi curah hujan sebaiknya didapatkan terlebih dahulu.

Di Indonesia, data prediksi curah hujan disediakan oleh Badan Meteorologi, Klimatologi, dan Geofisika (BMKG). Data tersebut dapat diakses secara *online*. Fig. 1 merupakan tampilan teratas ketika halaman *server* BMKG diakses. Sedangkan Fig. 2 merupakan tampilan lanjutan untuk halaman yang sama dengan Fig. 1. Dapat dilihat bahwa pada Fig. 1 dan Fig. 2 merupakan daftar dari data prediksi curah hujan yang tersedia di *server* BMKG. Format dari penamaan data yaitu `<yyyymmddhh>`, misalkan 2021022700, maka ini menunjukkan data prediksi curah hujan dari tanggal 27 Februari 2021 pukul 00.00. Seperti yang dapat dilihat pada Fig. 1 dan Fig. 2, data prediksi curah hujan di *server* BMKG tersebut tidak bersifat *real-time*, namun data akan disimpan setiap 12 jam.

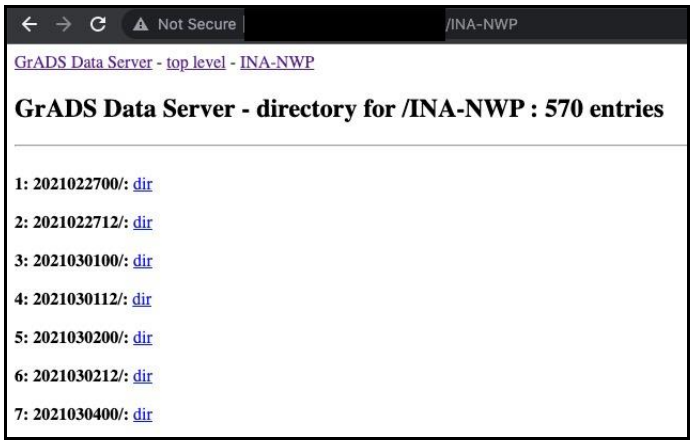

Fig. 1 Tampilan halaman teratas data prediksi curah hujan di *server* BMKG.

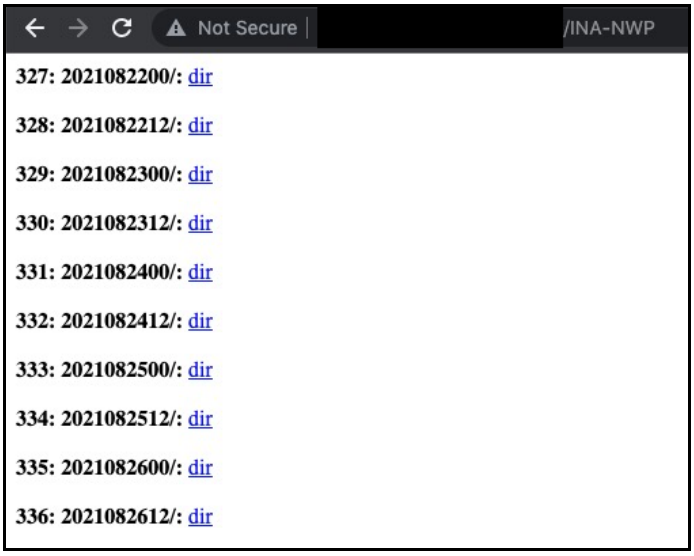

Fig. 2 Tampilan halaman lanjutan data prediksi curah hujan di *server* BMKG.

Jika tautan pada salah satu tanggal diklik, misalnya 2021122912, atau 29 Desember 2021 pukul 12.00, maka akan muncul tampilan seperti pada Fig. 3. Terdapat dua data, d01-asim yang menunjukkan data prediksi curah hujan untuk seluruh wilayah di Indonesia, dan d02-asim yang menunjukkan data prediksi curah hujan untuk seluruh wilayah di Pulau Jawa.

Pemrosesan data prediksi curah hujan yang terdapat di *server* BMKG lebih mudah dilakukan apabila data tersebut telah disimpan ke dalam komputer lokal. Akibatnya, perlu sebuah alat atau aplikasi otomasi untuk mendapatkan data tersebut. Salah satu alat atau aplikasi yang bisa digunakan adalah Apache Airflow. Oleh karena itu, Apache Airflow akan digunakan untuk menyelesaikan permasalahan akuisisi data secara periodik dari server BMKG ke komputer lokal.

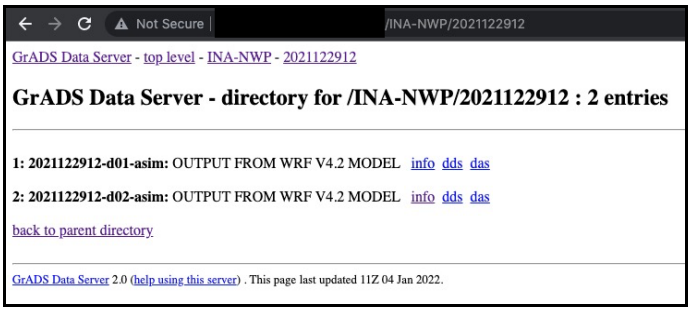

Fig. 3 Tampilan halaman data yang tersedia tanggal 29 Desember 2021 pukul 12.00

Kerangka penulisan makalah ini sebagai berikut. Bab I menjelaskan mengenai latar belakang. Bab II menjelaskan mengenai penelitian terkait dengan topik Apache Airflow. Bab III menjelaskan mengenai metode penelitian, yang berisi tentang penggunaan Apache Airflow untuk sensor ketersediaan data, penjadwalan tugas, serta pengambilan data dari *server* untuk disimpan di lokal. Bab IV menjelaskan mengenai hasil dan pembahasan, dan diakhiri dengan Bab V berisi kesimpulan dan saran.

II. TINJAUAN PUSTAKA

Pesatnya perkembangan ilmu data membuat perangkat *pipeline/workflow manager* semakin banyak digunakan. Apache Airflow adalah aplikasi *Workflow Management System* (WMS) berbasis tugas yang ditulis menggunakan bahasa Python [5], [6]. Proyek Airflow dimulai pada Oktober 2014 oleh Maxime Beauchemin

di Airbnb. Proyek Airflow bergabung dengan program Apache Software Foundation's Incubator pada Maret 2016 dan selanjutnya Apache Airflow sebagai Proyek Tingkat Tinggi pada Januari 2019. Dari awal, Apache Airflow merupakan proyek *open-source*. Dokumentasi dari macam-macam fitur yang tersedia serta langkah-langkah instalasi Apache Airflow juga dijelaskan secara lengkap dan runut di [7].

Apache Airflow telah digunakan dalam berbagai penelitian ilmu data [8]–[13]. Dalam [9], Apache Airflow berhasil menyelaraskan data dan menjadwalkan secara tepat untuk satu proses yang dijalankan sekali sehari. Apache Airflow juga dapat bersinergi dengan aplikasi lain seperti pustaka manajemen beban kerja PanDA untuk menghasilkan manajemen tugas terdistribusi [10].

Penelitian sebelumnya menunjukkan bahwa Apache Airflow mudah digunakan, efisien, terukur, dan memiliki fitur pemulihan kesalahan [11]. Dalam kasus *Big Data*, Apache Airflow dapat membantu proses *Extract-Transform-Load* (ETL) pada penyimpanan yang berbeda dengan total 30 juta dataset [12]. Penelitian di [13] membuktikan bahwa alur kerja Apache adalah WMS multiguna.

Apache Airflow juga digunakan dalam industri, seperti Tokopedia [14] dan Apple [15]. Tokopedia memanfaatkan Apache Airflow untuk melakukan migrasi data dari database, membuat laporan harian, menggabungkan beberapa tabel, menghitung beberapa metrik pengukuran, dan menuliskan data ke tempat penyimpanan eksternal. Terdapat lebih dari 1000 *pipeline* dan 9000 *task* yang dijalankan setiap harinya. Sedangkan Apple memanfaatkan Apache Airflow untuk membuat *pipeline* yang digunakan oleh beberapa tim dan membuat *operator* sendiri yang sesuai dengan kebutuhan.

III. METODE PENELITIAN

A. *Sensor Ketersediaan Data*

Seperti yang terlihat pada Fig. 2, data tidak selalu ada setiap 12 jam. Sebagai contoh, data untuk tanggal 29 Desember 2021 pukul 12.00 tersedia, terbukti dengan adanya tautan 2021122912. Sedangkan data untuk tanggal 29 Desember 2021 pukul 00.00 tidak tersedia. Jika tautan 2021122912 (data yang tersedia) diklik, maka akan muncul tampilan seperti pada Fig. 3. Namun, jika tautan tersebut diubah untuk menuju ke 2021122900 (data yang tidak tersedia), maka akan muncul tampilan seperti pada Fig. 4.

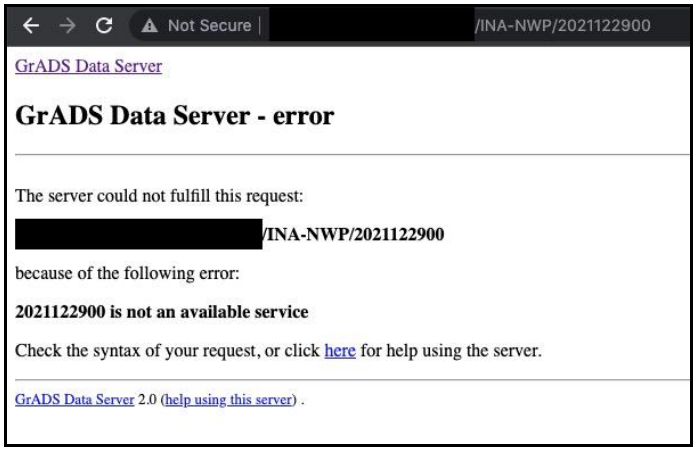

Fig. 4 Tampilan halaman data yang tidak tersedia tanggal 29 Desember 2021 pukul 00.00

Seperti yang terlihat pada Fig. 3 dan Fig. 4, kedua tautan dapat diakses meskipun data untuk tanggal 29 Desember 2021 pukul 00.00 tidak tersedia. Salah satu yang membedakan adalah dalam halaman yang ditampilkan, terdapat kalimat "*not an available service*". Perbedaan inilah yang akan diaplikasikan ke dalam sensor ketersediaan data.

Dalam mengimplementasikan sensor ketersediaan data, Apache Airflow telah menyediakan satu *operator* yang bernama `BaseSensorOperator`. Namun, *operator* ini tidak bisa langsung digunakan, karena terdapat *method* `poke` yang masih kosong, dan harus diimplementasikan sendiri dengan cara *override* untuk membuat

sensor yang dibutuhkan. Akibatnya, *class* baru perlu dibuat untuk mengimplementasikan sensor ketersediaan data. *Class* baru tersebut akan dinamakan `DataSensor` dan merupakan turunan dari *class* `BaseSensorOperator`. Implementasi dari pembuatan *class* `DataSensor` dapat dilihat pada Fig. 5.

```
import requests

from airflow.sensors.base import BaseSensorOperator
from airflow.utils.decorators import apply_defaults

class DataSensor(BaseSensorOperator):
  template_fields = ('host', 'ts',)

  @apply_defaults
  def __init__(self, host, ts, *args, **kwargs):
    self.host = host
    self.ts = ts
    kwargs.setdefault('poke_interval', 15 * 60)
    super(DataSensor, self).__init__(*args, **kwargs)

  def poke(self, _context):
    temp = self.ts.split('T')
    url = f'{self.host}/{temp[0]}{temp[1][:2]}'
    print('poking url', url)
    r = requests.get(url)
    c = r.text.find('not an available service') == -1
    return c
```

Fig. 5 Implementasi *class* `DataSensor`

Operator sensor tersebut akan terus berjalan selama *method* `poke` mengembalikan nilai *false*. Terdapat parameter `poke_interval` dalam detik yang digunakan untuk mengatur berapa lama jarak *method* `poke` akan dijalankan ulang. *Method* `poke` berisikan kode untuk membuat tautan yang akan dituju terlebih dahulu, kemudian mengambil seluruh halaman dari tautan tersebut, dan mencari apakah kalimat "*not an available service*" ditemukan. Jika kalimat tersebut ditemukan, maka *method* `poke` akan mengembalikan nilai *false*. Namun, jika tidak ditemukan, maka *method* `poke` akan mengembalikan nilai *true*, dan *operator* sensor tersebut akan berubah status menjadi berhasil.

B. Penjadwalan Tugas

Setelah membuat sensor ketersediaan data, tugas-tugas yang dibutuhkan untuk mendapatkan data curah hujan akan diimplementasikan. Dalam Apache Airflow, tugas disebut sebagai *task*, dan rangkaian tugas-tugas disebut sebagai *pipeline*. *Pipeline* di Apache Airflow disebut dengan istilah *Directed Acyclic Graph* (DAG), yaitu graf berarah yang tidak memiliki *cycle*.

Pipeline atau DAG akan dibuat agar dapat beroperasi selama 2 kali sehari, yaitu pada pukul 00.00 dan 12.00, mengikuti jadwal data di *server* BMKG. Fig. 6 adalah implementasi untuk inisialisasi DAG.

```
from airflow import DAG
from airflow.utils.dates import days_ago

dag = DAG(
  dag_id='get_rainfall_data_dag',
  schedule_interval='0 0,12 * * *',
  start_date=days_ago(7),
  max_active_runs=1
)
```

Fig. 6 Implementasi inisialisasi DAG

Terdapat beberapa parameter yang digunakan untuk inisialisasi DAG pada Fig. 6. Parameter `dag_id` berisikan *id* dari DAG atau *pipeline* yang dibuat, dan setiap DAG haruslah memiliki *id* yang berbeda-beda. DAG pada Fig. 6 memiliki *id* `get_rainfall_data_dag`. Kemudian, parameter `schedule_interval` berisikan "`0, 0,12 * * *`" dengan jenis *cron* [16], karena DAG diharapkan untuk beroperasi 2 kali sehari

pada pukul 00.00 dan 12.00. Parameter `start_date` menunjukkan kapan pipeline akan mengambil waktu pertama untuk jalan. Karena parameter `start_date` diisi `days_ago(7)`, maka DAG akan berjalan dari 7 hari sebelum DAG dijalankan. Terakhir, parameter `max_active_runs` mendefinisikan berapa DAG yang dapat dijalankan bersama, dan diberi nilai 1 agar tidak ada dua DAG `get_rainfall_data_dag` dengan jadwal berbeda yang jalan pada waktu yang sama.

DAG yang sudah dibuat akan digunakan untuk mendefinisikan *task*. Terdapat dua *task* yang akan dibuat, yaitu `wait_data` dan `get_data`. *Task* `wait_data` digunakan sebagai sensor untuk mengecek ketersediaan data. *Task* tersebut akan berjalan selama data belum tersedia, dan akan berhenti dengan status berhasil jika datanya ada, atau berhenti dengan status gagal ketika waktu eksekusi sudah habis. *Task* `get_data` digunakan untuk mengambil data. Tentunya *task* `wait_data` harus berhasil terlebih dahulu sebelum *task* `get_data` jalan. Jika *task* `wait_data` gagal, maka *task* `get_data` tidak akan bisa dijalankan. Implementasi dari pendefinisian kedua *task* ini dapat dilihat pada Fig. 7, yang diakhiri dengan *statement* `wait_data >> get_data`, yang menunjukkan urutan *task* yang akan dikerjakan.

Task `wait_data` pada Fig. 7 menggunakan *operator* `DataSensor` yang telah diimplementasikan pada Fig. 5. Terdapat beberapa parameter yang digunakan dalam mendefinisikan *task* `wait_data`. Parameter `dag` berisikan *object* `dag` yang telah didefinisikan pada Fig. 6, yang menunjukkan bahwa *task* `wait_data` akan diletakkan dalam DAG tersebut. Parameter `task_id` berisikan dari *id* untuk *task* yang terdapat dalam suatu DAG. Setiap *task* di dalam suatu DAG harus memiliki nilai `task_id` yang berbeda. Parameter `execution_timeout` merupakan batas waktu yang diperbolehkan bagi suatu *task* untuk berjalan. Dalam Fig. 7. digunakan nilai 13 jam, yang artinya jika *task* `wait_data` berjalan lebih dari 13 jam, maka *task* tersebut akan berubah status menjadi gagal. Parameter `host` dan `ts` merupakan parameter tambahan bagi `DataSensor`, yang mana nilai parameter `host` berisikan alamat *host* dari *server* BMKG dan parameter `ts` akan berisikan waktu sesuai penjadwalan ketika DAG berjalan.

Task `get_data` pada Fig. 7 menggunakan *operator* `BashOperator`. *Operator* dengan tipe tersebut akan menjalankan perintah layaknya di *terminal* atau *cmd*. Terdapat beberapa parameter yang digunakan dalam mendefinisikan *task* `get_data`. Parameter `dag` dan `task_id` memiliki makna yang serupa dengan yang terdapat dalam *task* `wait_data`, hanya berbeda di nilai dari parameter `task_id`. Parameter `bash_command` menunjukkan *command* atau perintah yang akan dijalankan. Perintah akan dimulai dengan membuat direktori atau *folder* yang digunakan untuk menyimpan data prediksi curah hujan di lokal, dan dilanjutkan dengan menjalankan kode `get_data.py` untuk pengambilan data dari *server*. Parameter `retries` merupakan berapa kali maksimal *task* tersebut akan diulang sampai berhasil. Jika *task* sudah diulangi sejumlah `retries` dan masih gagal, maka *task* tersebut akan berubah status menjadi gagal.

```
from airflow.operators.bash_operator
import BashOperator

project_dir = 'Users/erwinekowahyudi/airflow'
host = 'http://xxx.xxx.xxx.xxxx:xxxx/xxxx/INA-NWP'
ts = '{{ts_nodash}}'

wait_data = DataSensor(
  dag=dag,
  task_id='wait_data',
  execution_timeout=timedelta(hours=13),
  host=host,
  ts=ts
)

get_data = BashOperator(
  dag=dag,
  task_id='get_data',
  bash_command=f'mkdir -p {project_dir}/rainfall_data' \
    f' && python' \
    f' {project_dir}/get_data.py' \
    f' --host {host}' \
    f' --ts {ts}' \
    f' --result-dir {project_dir}/rainfall_data',
  retries=4
)

wait_data >> get_data
```

Fig. 7 Implementasi *task* `wait_data` dan `get_data` serta urutannya

Ketiga implementasi, yaitu Fig. 5, Fig. 6, dan Fig. 7, merupakan implementasi yang berkaitan dengan *pipeline*. Ketiganya akan disimpan ke dalam satu *file* dengan nama `get_rainfall_data_dag.py`.

C. *Pengambilan Data*

Seperti yang terlihat di Fig. 7, *file* `get_data.py` akan menerima 3 argumen, yaitu `host`, `ts`, dan `result-dir`. Argumen `host` berisi alamat *host* dari *server* BMKG dan argumen `ts` akan berisikan waktu sesuai penjadwalan ketika DAG berjalan, dengan argumen `ts` memiliki format `<yyyymmdd>T<hhmmss>`, dan argumen `result-dir` merupakan direktori atau *folder* tempat data akan disimpan di lokal. Untuk memudahkan pembacaan argumen program, digunakan pustaka argparse yang merupakan pustaka bawaan dari bahasa Python. Tautan untuk data yang ada di *server* BMKG dibentuk dari argumen `host` dan `ts`, dengan format `<host>/<yyyymmddhh>/<yyyymmddhh>-d02-asim`. Fig. 8 adalah implementasi fungsi pembacaan argumen beserta pembuatan tautan untuk data.

```
import argparse

def parse_arguments():
  parser = argparse.ArgumentParser()
  parser.add_argument('--host')
  parser.add_argument('--ts')
  parser.add_argument('--result-dir')

  return parser.parse_args()


def make_url(host, ts):
  temp = ts.split('T')
  ts = temp[0] + temp[1][:2]

  return f'{host}/{ts}/{ts}-d02-asim'  ts
```

Fig. 8 Implementasi pembacaan argumen dan pembuatan tautan untuk data

Setelah memperoleh tautan dari data yang akan diakses, data akan dibaca dengan menggunakan pustaka xarray. Pustaka xarray merupakan proyek *open-source* yang membuat bekerja dengan array multidimensi berlabel menjadi sederhana dan efisien, dikarenakan pustaka xarray memperkenalkan label dalam bentuk

dimensi, koordinat, dan atribut di atas *array* mentah seperti NumPy [17]. Pustaka tersebut bukan merupakan pustaka bawaan dari bahasa Python, sehingga pustaka tersebut harus dipasang terlebih dahulu. Lebih lanjut, data dibaca dengan pustaka xarray dengan format NetCDF4 [18].

Karena data dari *server* BMKG merupakan data untuk seluruh wilayah di Pulau Jawa, maka perlu dilakukan proses seleksi untuk membatasi data hanya untuk daerah Provinsi Daerah Istimewa Yogyakarta. Sehingga, hanya dipilih nilai *latitude* dan *longitude* yang membentuk persegi panjang yang menutupi wilayah Provinsi Daerah Istimewa Yogyakarta.

Proses seleksi selanjutnya adalah pemilihan kolom. Data dari *server* BKMG terdiri atas beberapa kolom. Yang akan digunakan adalah kolom "rainc", "rainnc", dan "rainsh", untuk mendapatkan prediksi curah hujan, berdasarkan lokasi *latitude* dan *longitude* yang telah dibatasi pada proses sebelumnya.

Setelah dilakukan proses seleksi wilayah dan seleksi kolom, data hasil seleksi akan dibentuk dan disimpan di komputer lokal dalam format NetCDF4. Penamaan file di lokal akan menggunakan format `<yyyymmddhh>.nc` sesuai dengan penamaan yang ada pada *server* BMKG. Fig. 9 merupakan implementasi dari pembacaan data dari *server* BMKG, kemudian proses seleksi wilayah dan dilanjutkan dengan proses seleksi kolom, kemudian diakhiri dengan menyimpan data hasil seleksi ke komputer lokal.

```
import xarray as xr

def main():
  args = parse_arguments()

  url, ts = make_url(args.host, args.ts)

  xarray_file = xr.open_dataset(url, engine='netcdf4')

  time_data = xarray_file['time'].to_numpy()
  lat_data = xarray_file['lat'][105:129].to_numpy()
  lon_data = xarray_file['lon'][310:340].to_numpy()

  rainc_data =
xarray_file['rainc'][:,0,105:129,310:340].to_numpy()
  rainnc_data =
xarray_file['rainnc'][:,0,105:129,310:340].to_numpy()
  rainsh_data =
xarray_file['rainsh'][:,0,105:129,310:340].to_numpy()

  data = xr.Dataset(
    data_vars=dict(
      rainc=(['time', 'lat', 'lon'], rainc_data),
      rainsh=(['time', 'lat', 'lon'], rainsh_data),
      rainnc=(['time', 'lat', 'lon'], rainnc_data),
    ),
    coords=dict(
      time=(['time'], time_data),
      lat=(['lat'], lat_data),
      lon=(['lon'], lon_data)
    ),
  )

  result_path = f'{args.result_dir}/{ts}.nc'
```

Fig. 9 Implementasi pembacaan data, proses seleksi wilayah dan kolom, serta penulisan data hasil seleksi ke komputer lokal

Kedua implementasi, yaitu Fig. 8 dan Fig. 9, merupakan implementasi yang berkaitan dengan proses utama pengambilan data. Keduanya akan disimpan ke dalam satu *file* dengan nama `get_data.py`.

IV. HASIL DAN PEMBAHASAN

Ketika DAG `get_rainfall_data_dag` dijalankan untuk pertama kali pada 30 Desember 2021, jadwal pertama yang dijalankan oleh DAG tersebut adalah 23 Desember 2021 pukul 00.00. Fig. 10 merupakan tampilan dari *history* status DAG tersebut pada Apache Airflow.

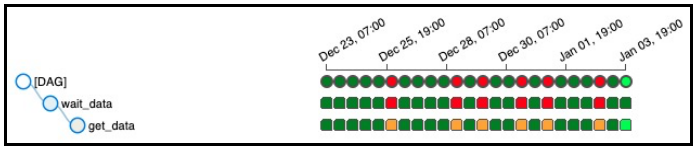

Fig. 10 Status DAG dari pertama kali berjalan

Pada Fig. 10 terlihat bahwa untuk setiap penjadwalan DAG, terdapat tanda lingkaran yang menyatakan status dari keseluruhan DAG dan tanda persegi yang menyatakan status dari masing-masing *task*. Terdapat beberapa macam status untuk DAG, yaitu warna hijau tua yang menandakan status DAG telah selesai dan berhasil, warna merah yang menandakan status DAG telah selesai namun gagal, dan warna hijau muda yang menandakan status DAG tersebut sedang berjalan. Jika salah satu lingkaran berwarna hijau tua (menandakan DAG berhasil) diklik dan menuju ke *graph view*, maka akan muncul tampilan seperti pada Fig. 11. Namun, jika salah satu lingkaran berwarna merah (menandakan DAG gagal) diklik, maka akan muncul tampilan seperti pada Fig. 12. Jika lingkaran berwarna hijau muda (menandakan DAG sedang berjalan) diklik, maka akan muncul tampilan seperti pada Fig. 13.

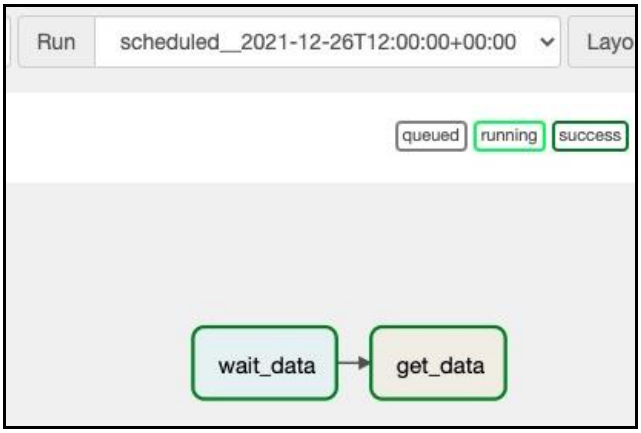

Fig. 11 DAG selesai berjalan dengan status berhasil

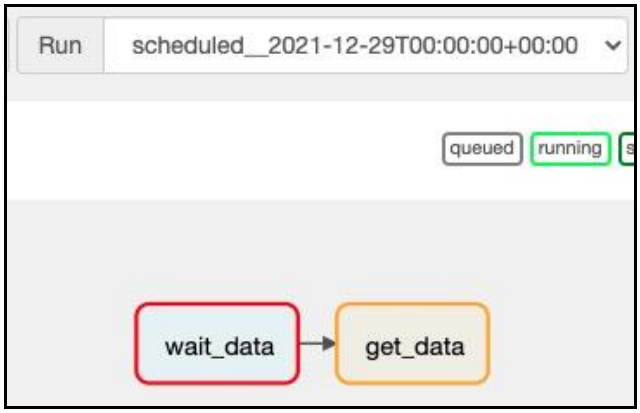

Fig. 12 DAG selesai berjalan dengan status gagal

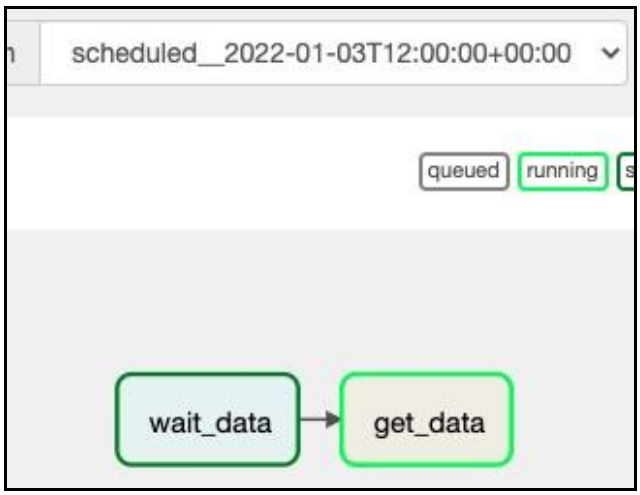

Fig. 13 DAG sedang berjalan

Kedua *task* pada Fig. 11 berwarna hijau tua, menandakan kedua *task* berhasil. *Task* `wait_data` mempunyai status berhasil, artinya data di *server* BMKG untuk tanggal tertentu (26 Desember 2021 pukul 00.00 pada Fig. 11) benar tersedia. *Task* `get_data` juga mempunyai status berhasil, artinya data yang ada di *server* BMKG sudah berhasil disimpan di komputer lokal.

Pada Fig. 12, *task* `wait_data` berwarna merah, artinya *task* tersebut mempunyai status gagal, dan *task* `get_data` berwarna oranye, yang artinya *task* tersebut tidak akan dijalankan karena *task* sebelumnya (*task* `wait_data`) mempunyai status gagal. Lebih lanjut, jika *task* `wait_data` diklik dan melihat *log*, maka akan muncul tulisan "`AirflowTaskTimeout`" (Fig. 14), yang artinya *task* `wait_data` sudah melebihi batas waktu maksimal yang ditentukan (13 jam).

```
  File "/Users/erwinekowahyudi/miniconda3/envs/airflow/lib/
    raise AirflowTaskTimeout(self.error_message)
airflow.exceptions.AirflowTaskTimeout: Timeout, PID: 47938
```

Fig. 14 *Log* menampilkan `AirflowTaskTimeout`

Pada Fig. 13, *task* `wait_data` berwarna hijau tua, yang artinya *task* tersebut berhasil dijalankan dan data di *server* BMKG untuk tanggal tertentu (3 Januari 2021 pukul 12.00 pada Fig. 13) benar tersedia. *Task* `get_data` berwarna hijau muda yang menandakan *task* tersebut sedang berjalan, artinya data yang ada di *server* BMKG sedang dalam proses disimpan di komputer lokal.

Data yang sudah berhasil disimpan di komputer lokal, akan tersimpan pada *folder* rainfall_data, seperti yang telah didefinisikan pada perintah atau *command* pada Fig. 7. Fig. 15 menunjukkan daftar *file* yang sudah berhasil disimpan pada komputer lokal.

```
(airflow) → rainfall_data git:(master) ✗ ls
2021122300.nc 2021122500.nc 2021122712.nc 2021123100.nc 2022010300.nc
2021122312.nc 2021122600.nc 2021122812.nc 2022010100.nc 2022010312.nc
2021122400.nc 2021122612.nc 2021122912.nc 2022010112.nc
2021122412.nc 2021122700.nc 2021123000.nc 2022010200.nc
```

Fig. 15 Data prediksi curah hujan disimpan di komputer lokal

V. KESIMPULAN

Data prediksi curah hujan yang telah disediakan oleh BMKG dapat disimpan ke komputer lokal secara otomatis dengan menggunakan alat atau aplikasi otomasi bernama Apache Airflow. Terdapat dua *task* yang dibuat dalam DAG, yaitu *task* pertama sebagai sensor ketersediaan data dan *task* kedua yang menjalankan

proses pengambilan data dari server BMKG dan penyimpanan ke komputer lokal. Status dari DAG juga dapat diketahui secara cepat, seperti DAG telah berhasil, gagal, atau sedang berjalan. Apache Airflow juga menyediakan *log* yang dapat digunakan untuk mengetahui penyebab kegagalan suatu *task*.

Saran pengembangan selanjutnya adalah untuk lebih banyak memanfaatkan fitur-fitur yang telah disediakan oleh Apache Airflow. Salah satu fitur yang berguna adalah Apache Airflow dapat memberikan notifikasi ke pengguna jika ada *task* yang berhasil atau gagal.

UCAPAN TERIMA KASIH

Ucapan terima kasih disampaikan kepada Badan Meteorologi, Klimatologi, dan Geofisika (BMKG) yang telah menyediakan data prediksi curah hujan dan dapat diakses secara *online*.

DAFTAR PUSTAKA

[1] S. Huber, H. Wiemer, D. Schneider, and S. Ihlenfeldt, "DMME: Data mining methodology for engineering applications – a holistic extension to the CRISP-DM model," *Procedia CIRP*, vol. 79, pp. 403–408, 2019, doi: https://doi.org/10.1016/j.procir.2019.02.106.

[2] S. T. McColl, "Chapter 2 - Landslide Causes and Triggers," in *Landslide Hazards, Risks, and Disasters*, J. F. Shroder and T. Davies, Eds. Boston: Academic Press, 2015, pp. 17–42.

[3] R. M. Iverson, "Landslide triggering by rain infiltration," *Water Resour. Res.*, vol. 36, no. 7, pp. 1897–1910, 2000, doi: https://doi.org/10.1029/2000WR900090.

[4] G. Samodra, N. Ngadisih, M. Malawani, D. Mardiatno, A. Cahyadi, and F. S. Nugroho, "Frequency–magnitude of landslides affected by the 27–29 November 2017 Tropical Cyclone Cempaka in Pacitan, East Java," *J. Mt. Sci.*, vol. 17, pp. 773–786, 2020, doi: 10.1007/s11629-019-5734-y.

[5] M. Kotliar, A. V Kartashov, and A. Barski, "CWL-Airflow: a lightweight pipeline manager supporting Common Workflow Language," *Gigascience*, vol. 8, no. 7, 2019, doi: 10.1093/gigascience/giz084.

[6] B. P. Harenslak and J. de Ruiter, *Data Pipelines with Apache Airflow*. Manning, 2021.

[7] "Apache Airflow." https://airflow.apache.org/ (accessed Jan. 01, 2022).

[8] T. Koivisto, "Efficient Data Analysis Pipeline," in *Data Science for Natural Sciences Seminar*, 2019, pp. 1–4.

[9] P. Chirupphapa, H. Esaki, and H. Ochiai, "INTAP: Integrated Network Traffic Analysis Pipeline for LAN Monitoring System," in *2021 7th International Conference on Information Management (ICIM)*, 2021, pp. 92–96, doi: 10.1109/ICIM52229.2021.9417147.

[10] D. I. Gavrilov, A. A. Iachmenev, I. A. Matveev, D. A. Oleynik, and A. S. Petrosyan, "Usage of The JINR SSO Authentication and Authorization System with Distributed Data Processing Services," in *9th International Conference "Distributed Computing and Grid Technologies in Science and Education" (GRID'2021)*, 2021, pp. 536–540.

[11] B. Ramanan, L. Drabeck, T. Woo, T. Cauble, and A. Rana, "~PB amp;J~ - Easy Automation of Data Science/Machine Learning Workflows," in *2020 IEEE International Conference on Big Data (Big Data)*, 2020, pp. 361–371, doi: 10.1109/BigData50022.2020.9378128.

[12] A. Suleykin and P. Panfilov, "Implementing big data processing workflows using open source technologies," in *30th DAAAM International Symposium on Intelligent Manufacturing and Automation*, 2019, pp. 394–404, doi: 10.2507/30th.daaam.proceedings.054.

[13] L. Finnigan and E. Toner, "Building and Maintaining Metadata Aggregation Workflows Using Apache Airflow." 2021, [Online]. Available: http://hdl.handle.net/20.500.12613/6955.

[14] S. Suganda, "How Does Tokopedia Take Airflow to the Next Level?," 2020. https://medium.com/tokopedia-data/how-does-tokopedia-take-airflow-to-the-next-level-fa7dbda3be2b (accessed Jan. 01, 2022).

[15] R. Santamaria and H. Wang, "Apache Airflow at Apple - Multi-tenant Airflow and Custom Operators," 2021. https://airflowsummit.org/sessions/2021/apache-airflow-at-apple/ (accessed Jan. 01, 2022).

[16] "Crontab," *The Open Group Base Specifications Issue 7 - IEEE Std 1003.1, 2018 edition*, 2018. https://pubs.opengroup.org/onlinepubs/9699919799/utilities/crontab.html (accessed Jan. 02, 2022).

[17] S. Hoyer and J. Hamman, "xarray: N-D labeled arrays and datasets in Python," *J. Open Res. Softw.*, vol. 5, no. 1, 2017, doi: 10.5334/jors.148.

[18] Unidata, "Network Common Data Form (NetCDF)." 2015, [Online]. Available: http://doi.org/10.5065/D6H70CW6.