Microsoft

Azure

Serverless in the era of data engineering and data science

Shreya Batra

Program Manager, Azure Functions

Varad Meru

Senior Software Engg. Manager, Azure Functions

Skyler Hartle

Senior Program Manager, Azure App Platform

Agenda

- Who we are
- What does “serverless” mean?
- What we’ve learned about Python and serverless
- Why are people using serverless?
- **Demo**: (Near) Real-time stream processing with Azure Functions
- What’s next for (some of) serverless and Python?

Who we are

Shreya Batra

Program Manager, Azure Functions

Varad Meru

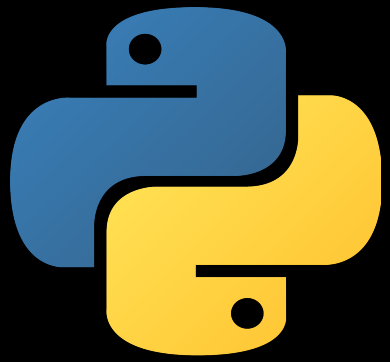
Senior Software Engg. Manager, Azure Functions

Skyler Hartle

Senior Program Manager, Azure Application Platform

Charter: Solve high-value problems for Python customers creating data-first applications on serverless platforms.

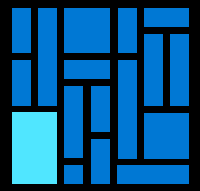
Visual Studio Code



Python

Azure

What does “serverless” mean?



No infrastructure management

Developers can just focus on their code—without needing to worry about provisioning and managing infrastructure

Instant, event-driven scalability

Application components react to events and triggers in near real-time with virtually unlimited scalability

Pay-per-use

Only pay for what you use: billing is typically calculated on the number of function calls, code execution time, and memory used*

Azure Serverless

PLATFORM

API Management

Event Grid

Manage all events that can trigger code or logic

Database

Storage

Functions

Execute your code based on events you specify

Automation

Intelligence

Logic Apps

Design workflows and orchestrate processes

Security

IoT

DEVELOPMENT

IDE support

Integrated DevOps

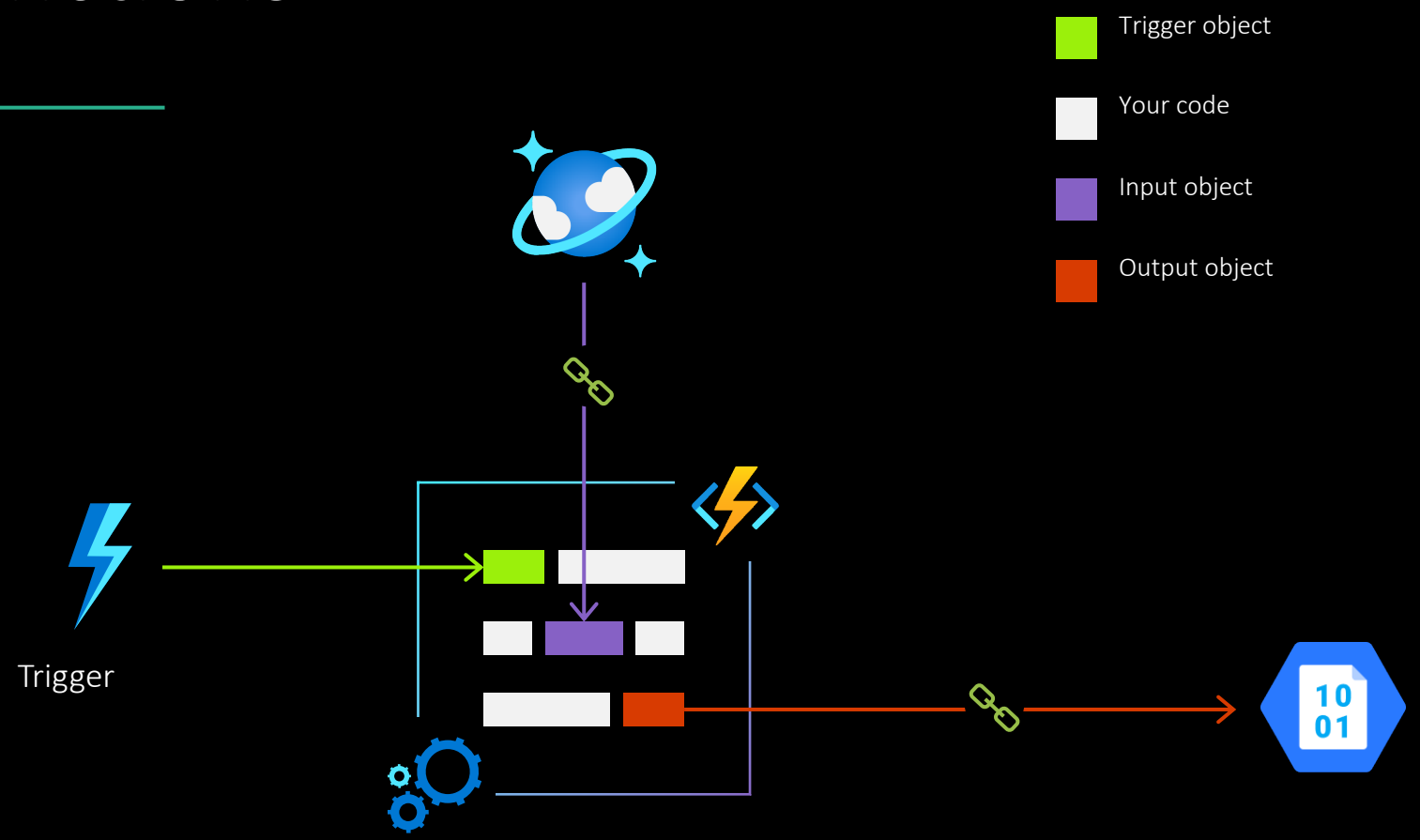
Local development

Monitoring

Visual debug history

Example: Azure Functions

Azure Functions features input/output bindings which provide a means of pulling data or pushing data to other services.



What we’ve learned about Python and serverless

Trend #1

Data analysis is becoming the most common use case for Python.

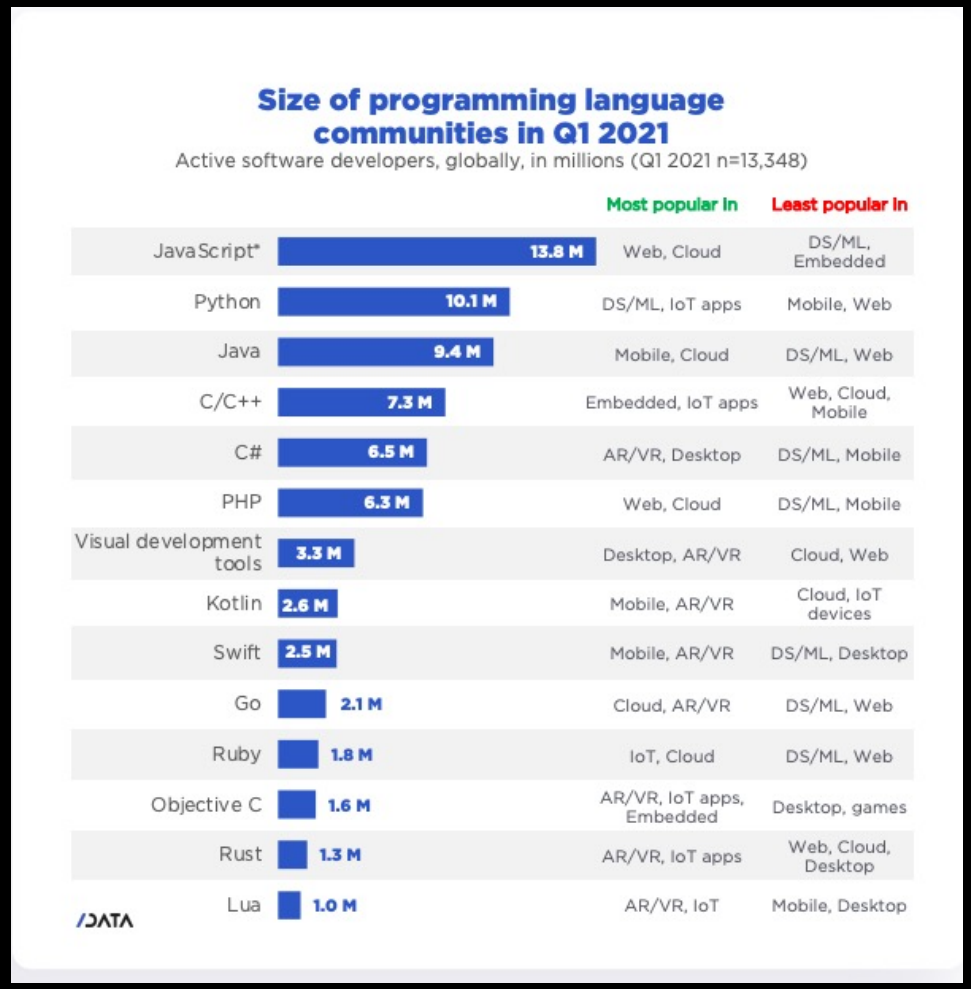
54%

Primary use of Python is Data Analysis (2020)

456%

Growth per year as of 2019

Kaggle Machine Learning & Data Science survey



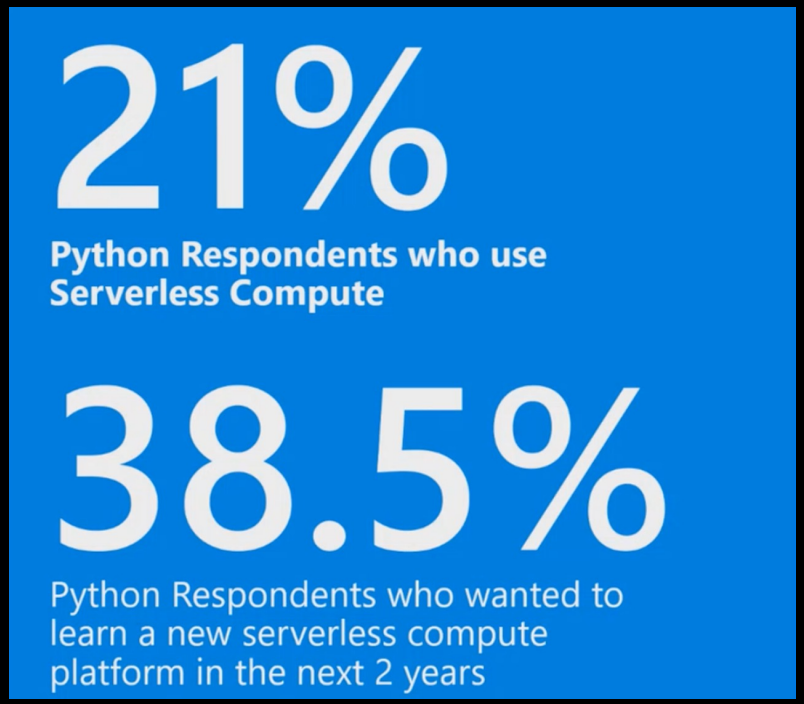
SlashData

2020 Kaggle Machine Learning & Data Science Survey | Kaggle

SlashData: JavaScript and Python boast largest developer communities (developer-tech.com)

Trend #2

There has been an explosion in growth between data related workloads and serverless platforms.

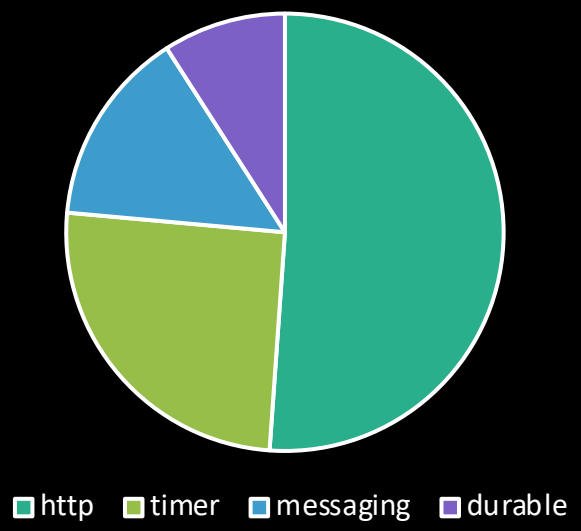


Kaggle Machine Learning & Data Science survey

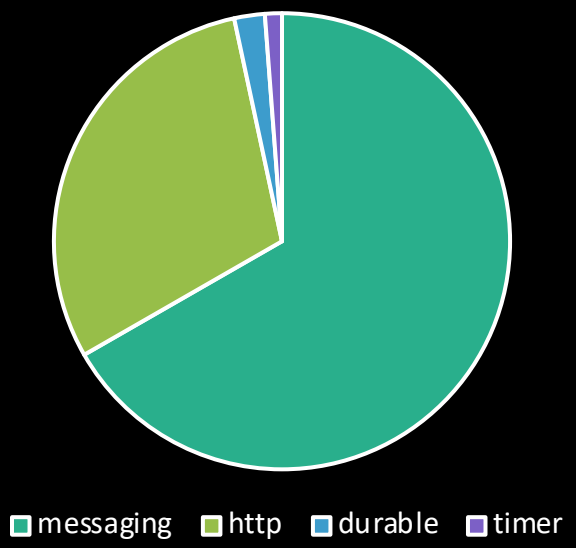
2020 Kaggle Machine Learning & Data Science Survey | Kaggle

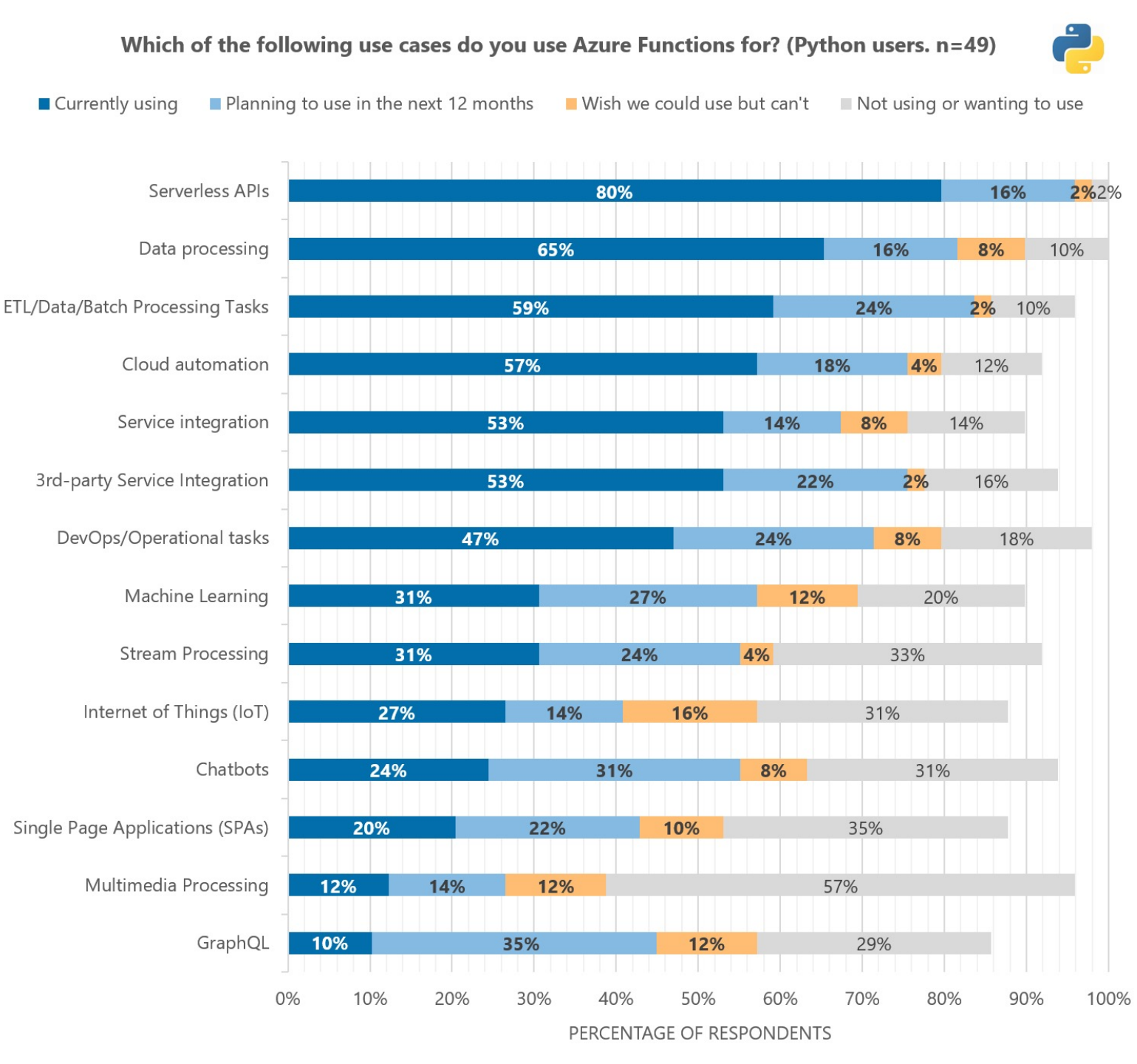
Python customer usage on Azure Functions

Most popular triggers by # of customers



Most popular triggers by # of executions



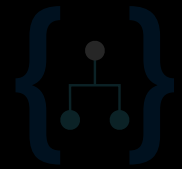

Which of the following use cases do you use Azure Functions for? (Python users. n=49)
Currently using
Planning to use in the next 12 months
Wish we could use but can't
Not using or wanting to use
Serverless APIs
80%
16%
2%
2%
Data processing
65%
16%
8%
10%
ETL/Data/Batch Processing Tasks
59%
24%
2%
10%
Cloud automation
57%
18%
4%
12%
Service integration
53%
14%
8%
14%
3rd-party Service Integration
53%
22%
2%
16%
DevOps/Operational tasks
47%
24%
8%
18%
Machine Learning
31%
27%
12%
20%
Stream Processing
31%
24%
4%
33%
Internet of Things (IoT)
27%
14%
16%
31%
Chatbots
24%
31%
8%
31%
Single Page Applications (SPAs)
20%
22%
10%
35%
Multimedia Processing
12%
14%
12%
57%
GraphQL
10%
35%
12%
29%
0%
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%
PERCENTAGE OF RESPONDENTS

What are customers doing with serverless and data?

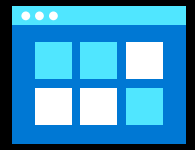
Common Functions scenarios

Machine learning

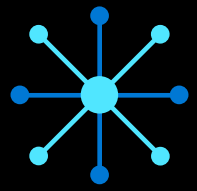
Real-time
stream processing



Workflows
and orchestration

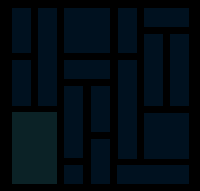


Scheduled
task automation



IoT-connected backends

Web/Mobile application
backends



No infrastructure management

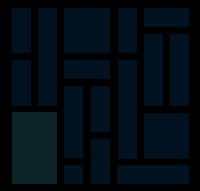
Developers can just focus on their code—without needing to worry about provisioning and managing infrastructure

Instant, event-driven scalability

Application components react to events and triggers in near real-time with virtually unlimited scalability

Pay-per-use

Only pay for what you use: billing is typically calculated on the number of function calls, code execution time, and memory used*



No infrastructure management

Developers can just focus on their code—without needing to worry about provisioning and managing infrastructure

Instant, event-driven scalability

Application components react to events and triggers in near real-time with virtually unlimited scalability

Pay-per-use

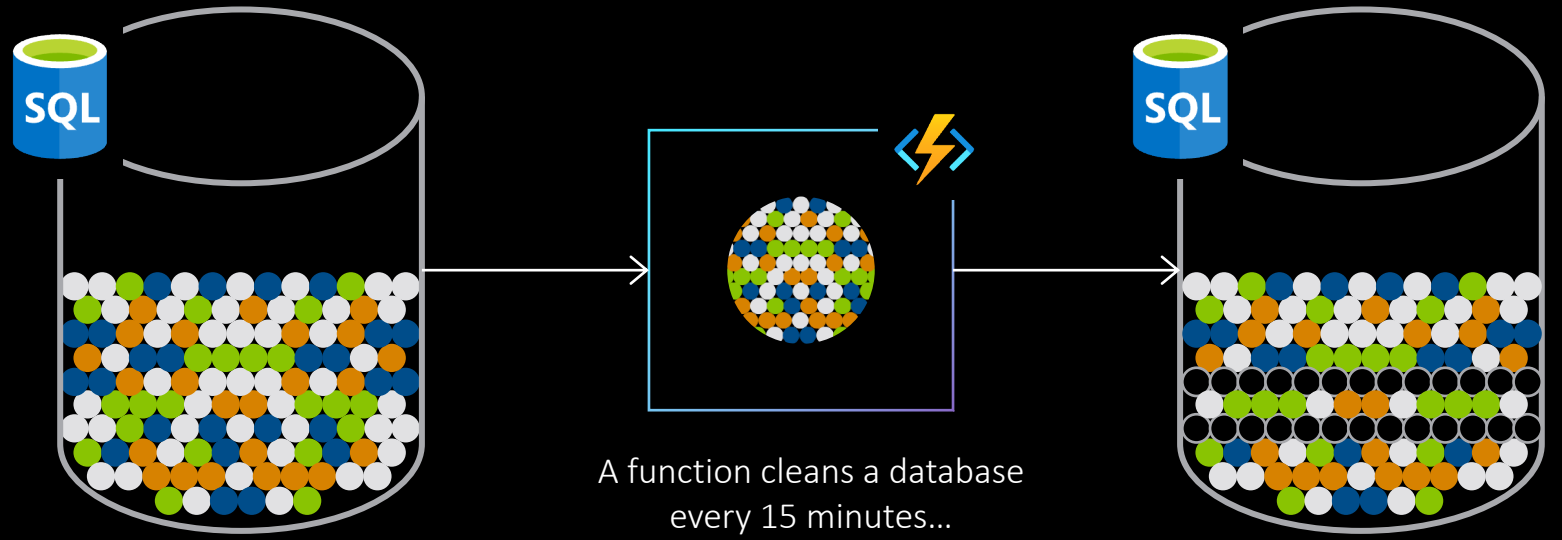
Only pay for what you use: billing is typically calculated on the number of function calls, code execution time, and memory used*

Automation of scheduled tasks

SCENARIO EXAMPLE

Financial services

A customer database is analyzed for duplicate entries every 15 minutes, to avoid multiple communications being sent out to same customers



A function cleans a database every 15 minutes…

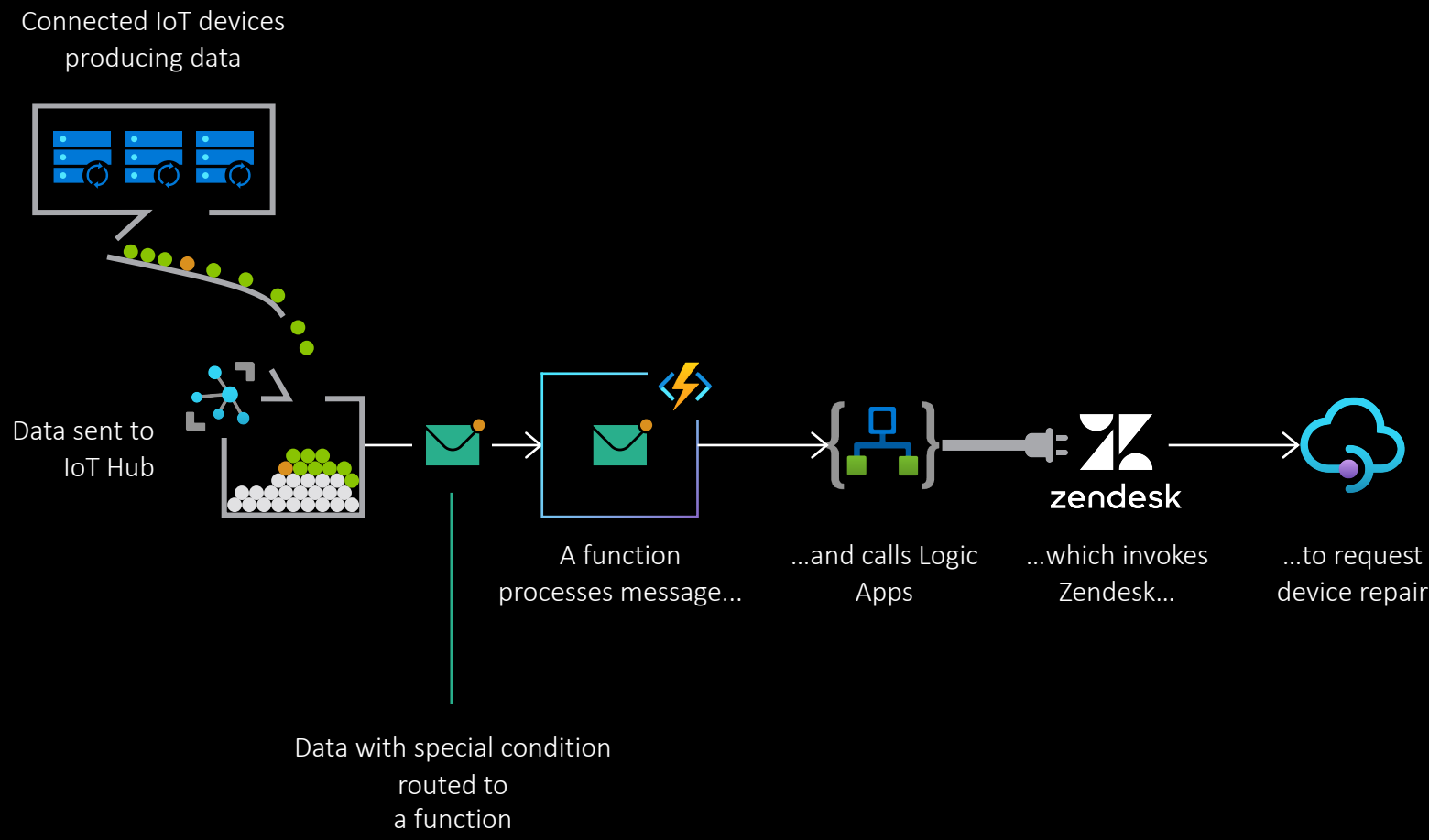
…deduplicating entries based on business logic

IoT-connected backends

SCENARIO EXAMPLE

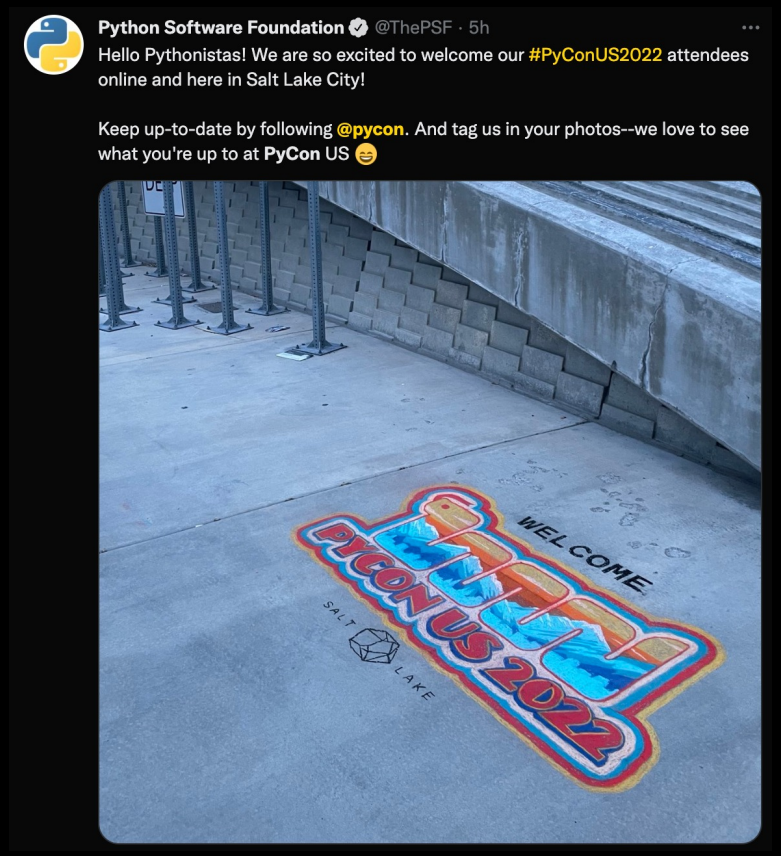
Manufacturing

A manufacturing company uses IoT to monitor its machines. Functions detects anomalous data and triggers a message to Service department when repair is required

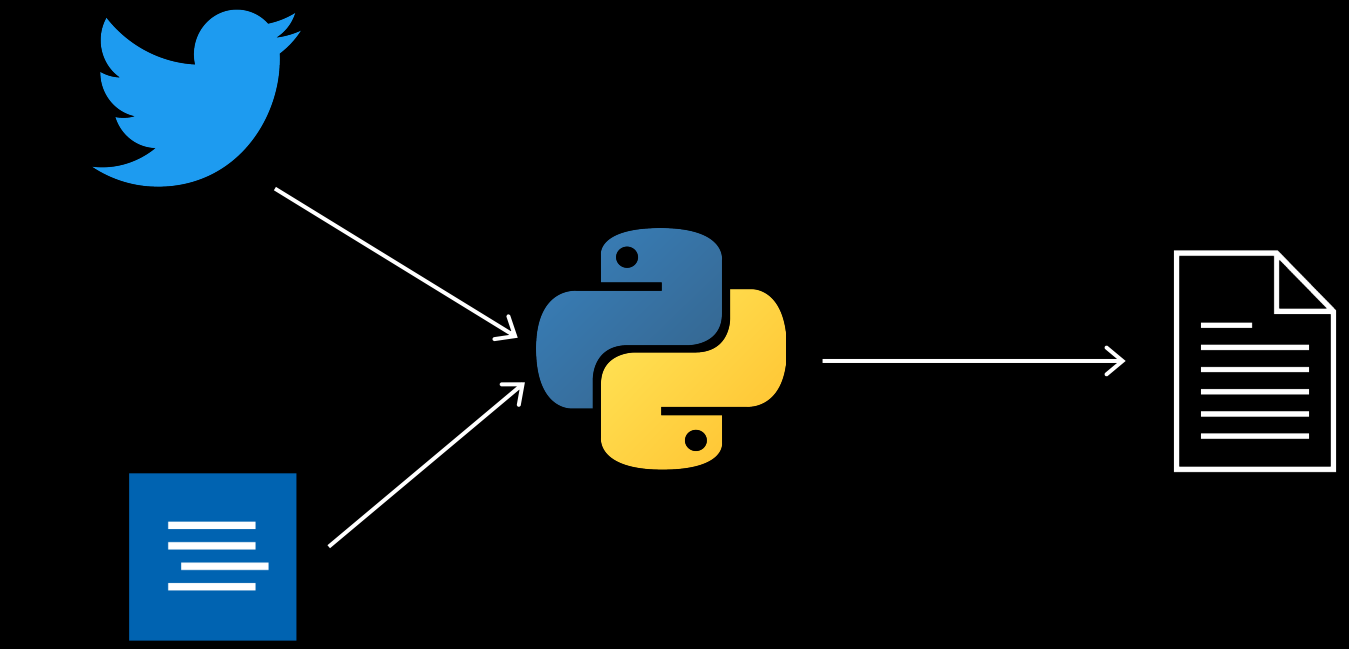


Demo: (near) Real-time stream processing app

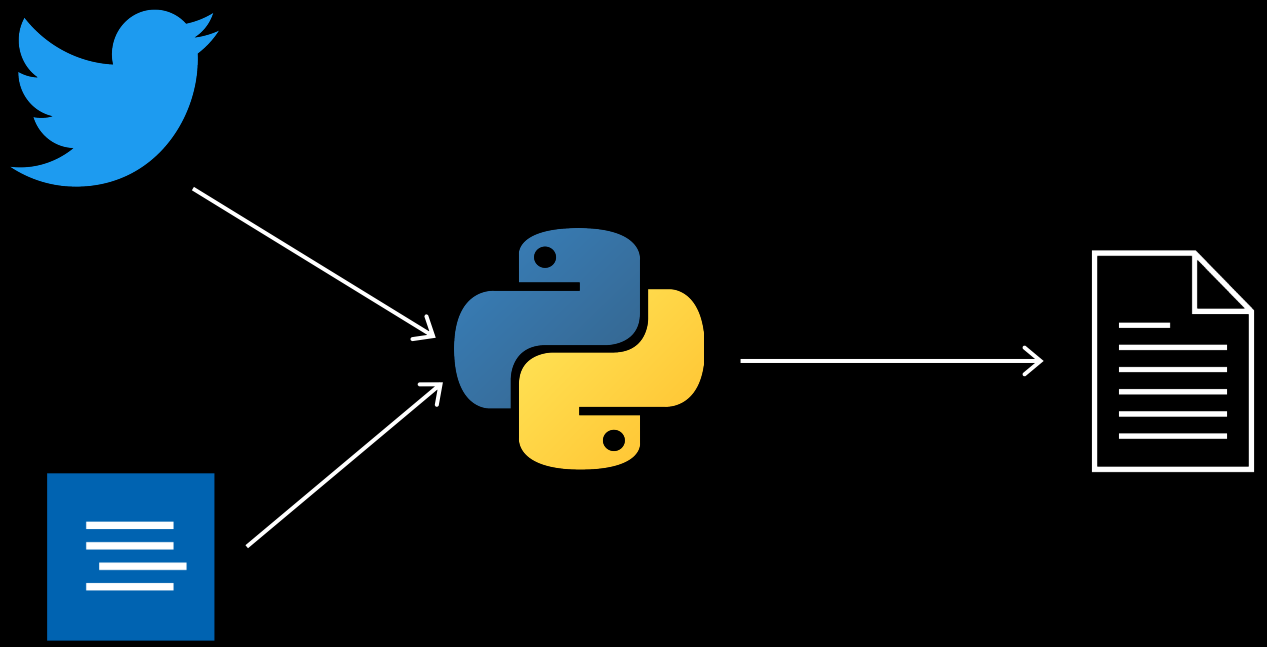
Tweets-Elonyzer



Making Tweets-Elonyzer

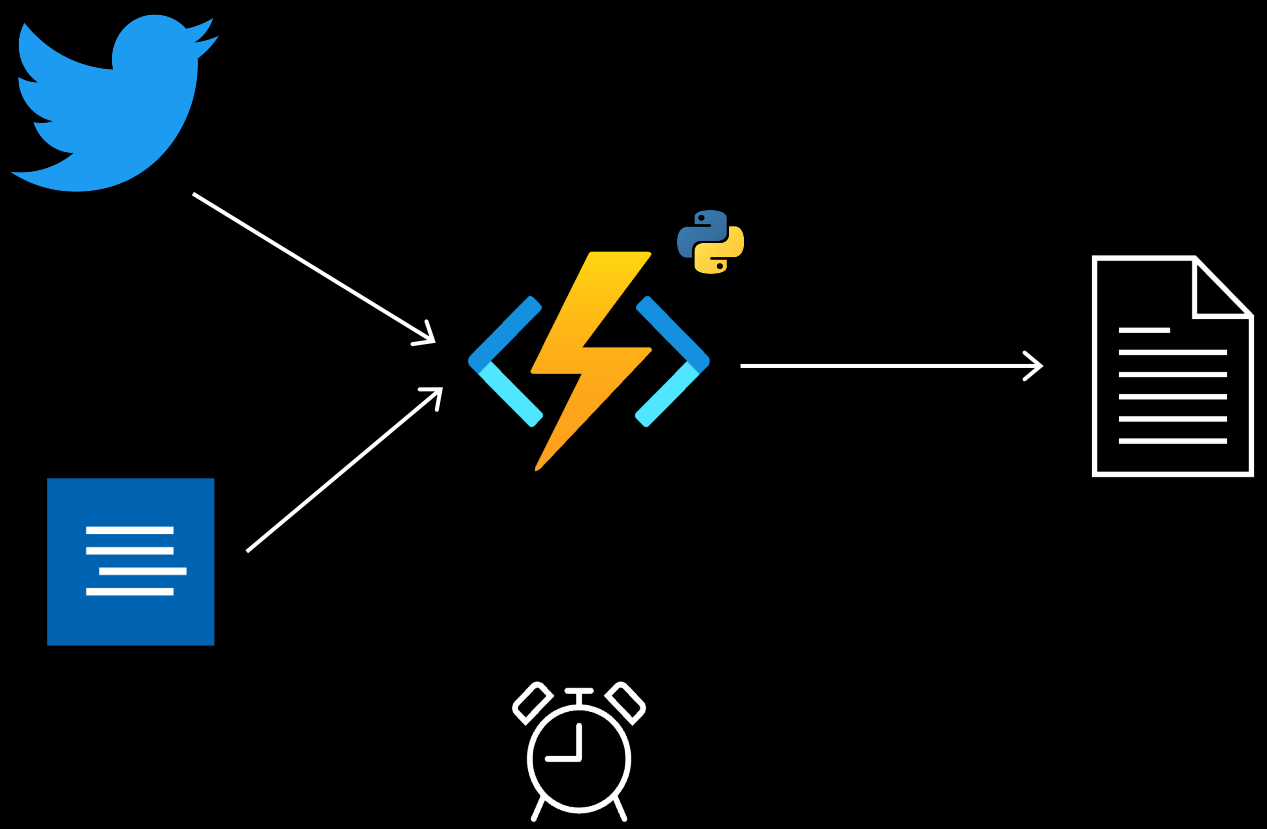


Making Tweets-Elonyzer

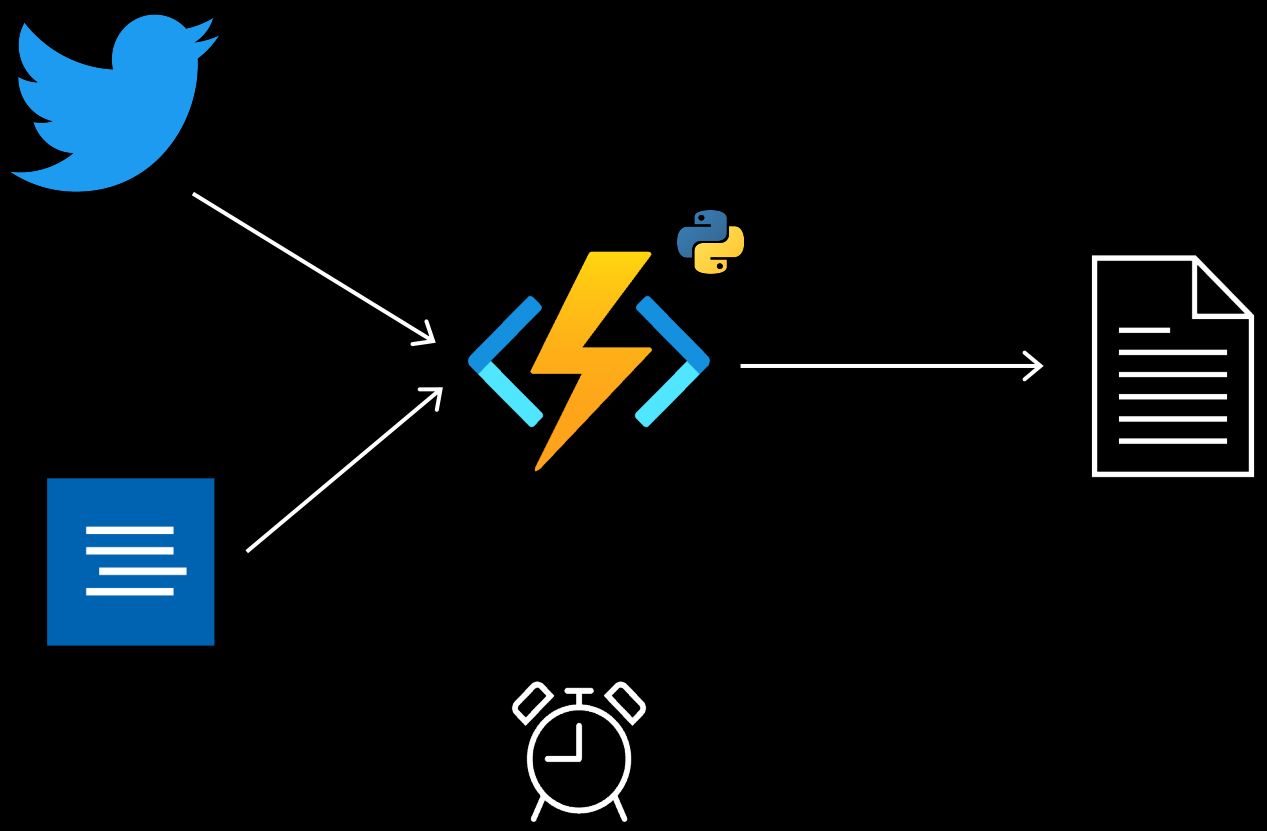


+ Quick and fast
- Sharability, Scalability

Making Tweets-Elonyzer

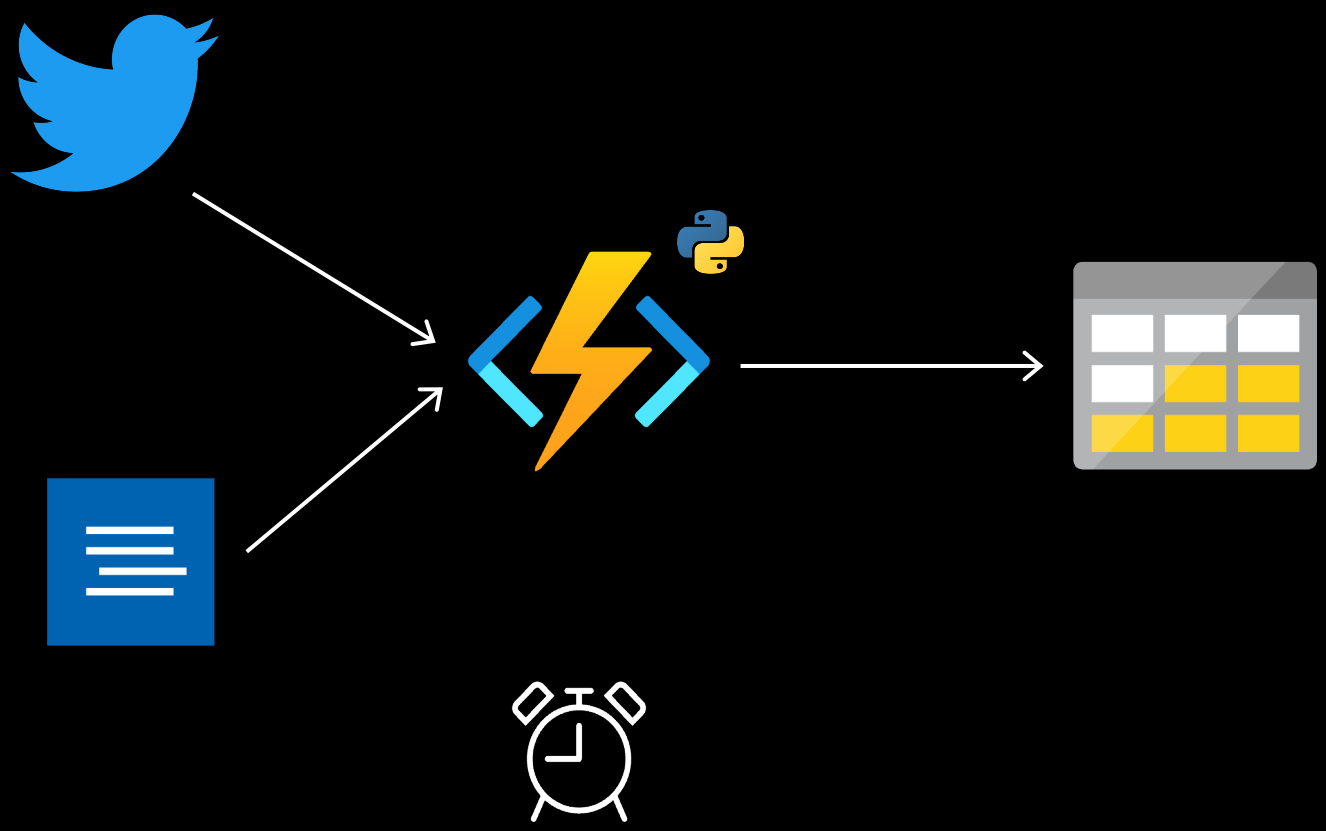


Making Tweets-Elonyzer

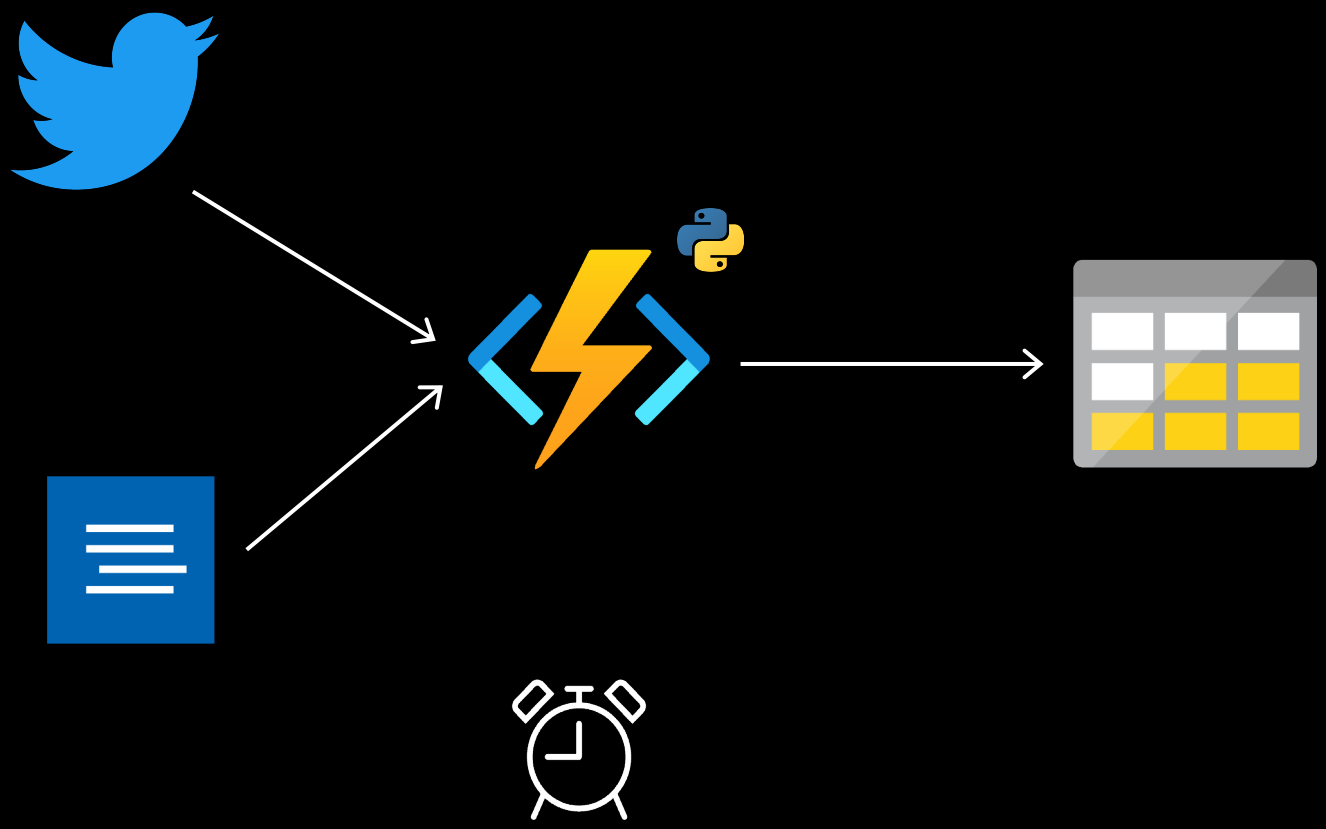


+ Quick-ish and fast
- Queryability, Scalability

Making Tweets-Elonyzer

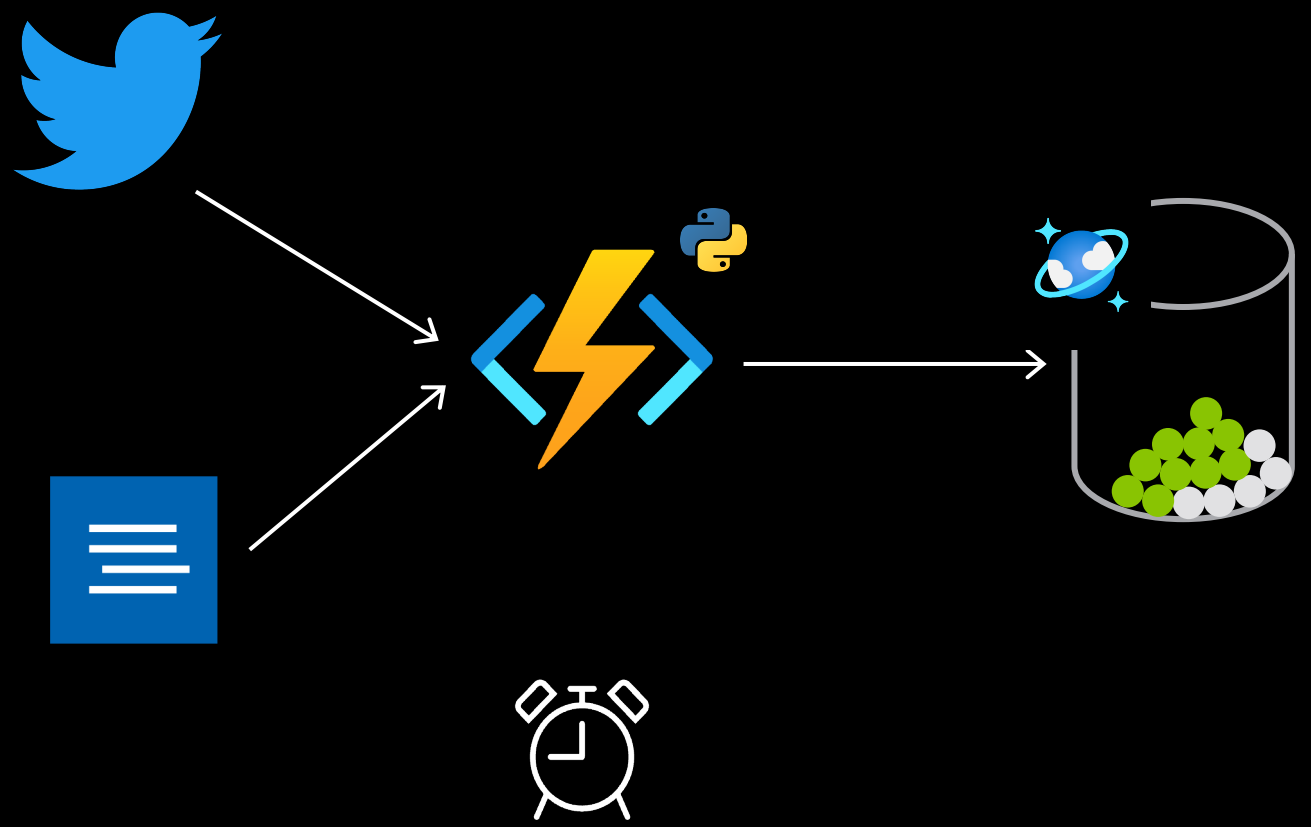


Making Tweets-Elonyzer

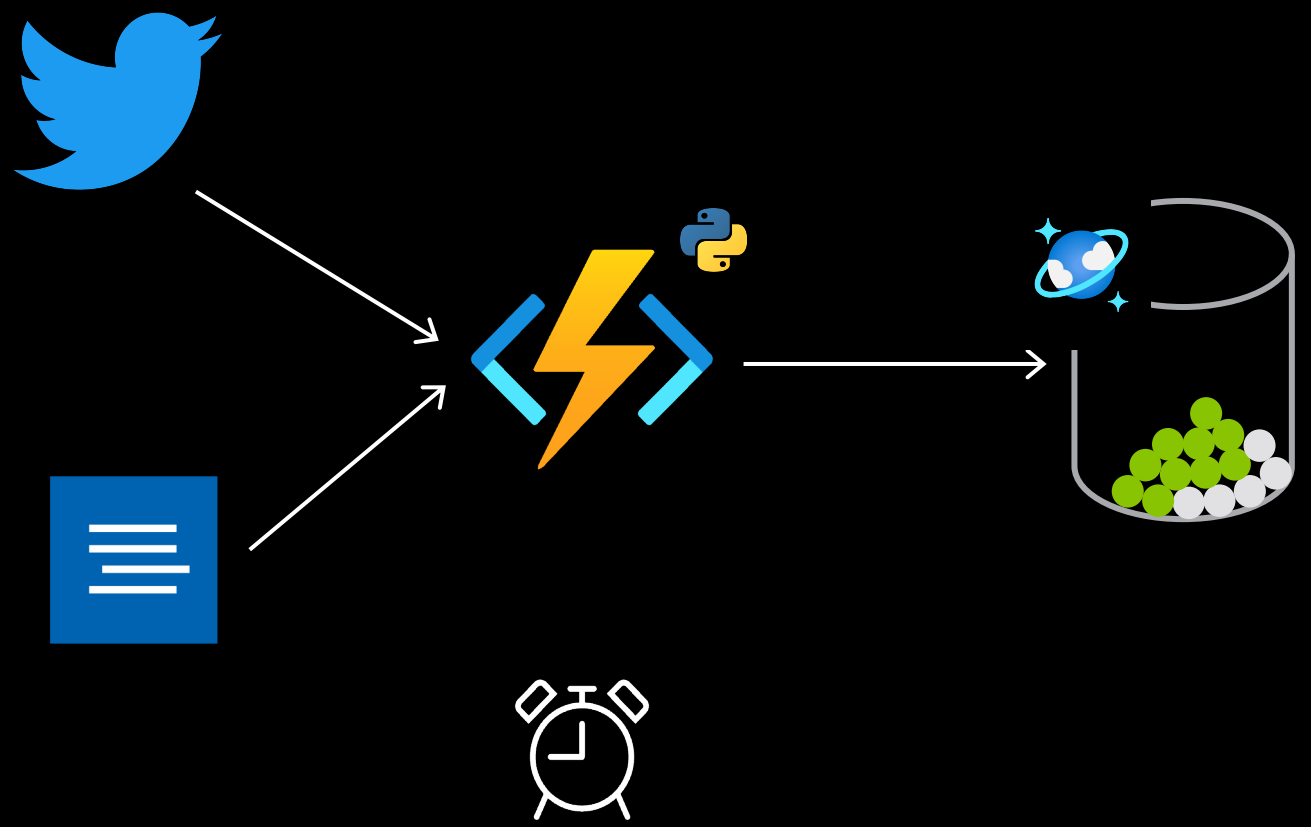


+ Accessible resources
- Queryability

Making Tweets-Elonyzer

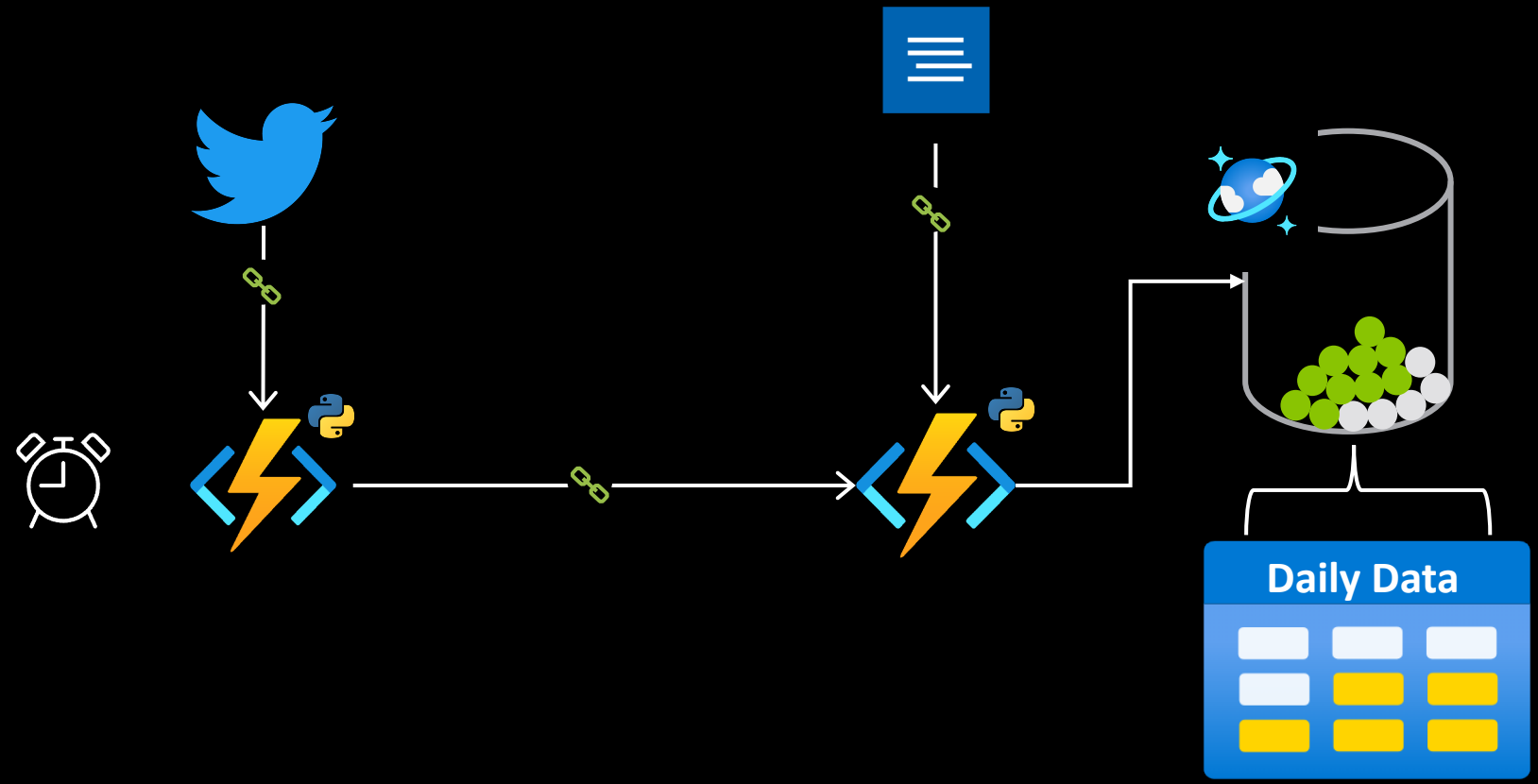


Making Tweets-Elonyzer

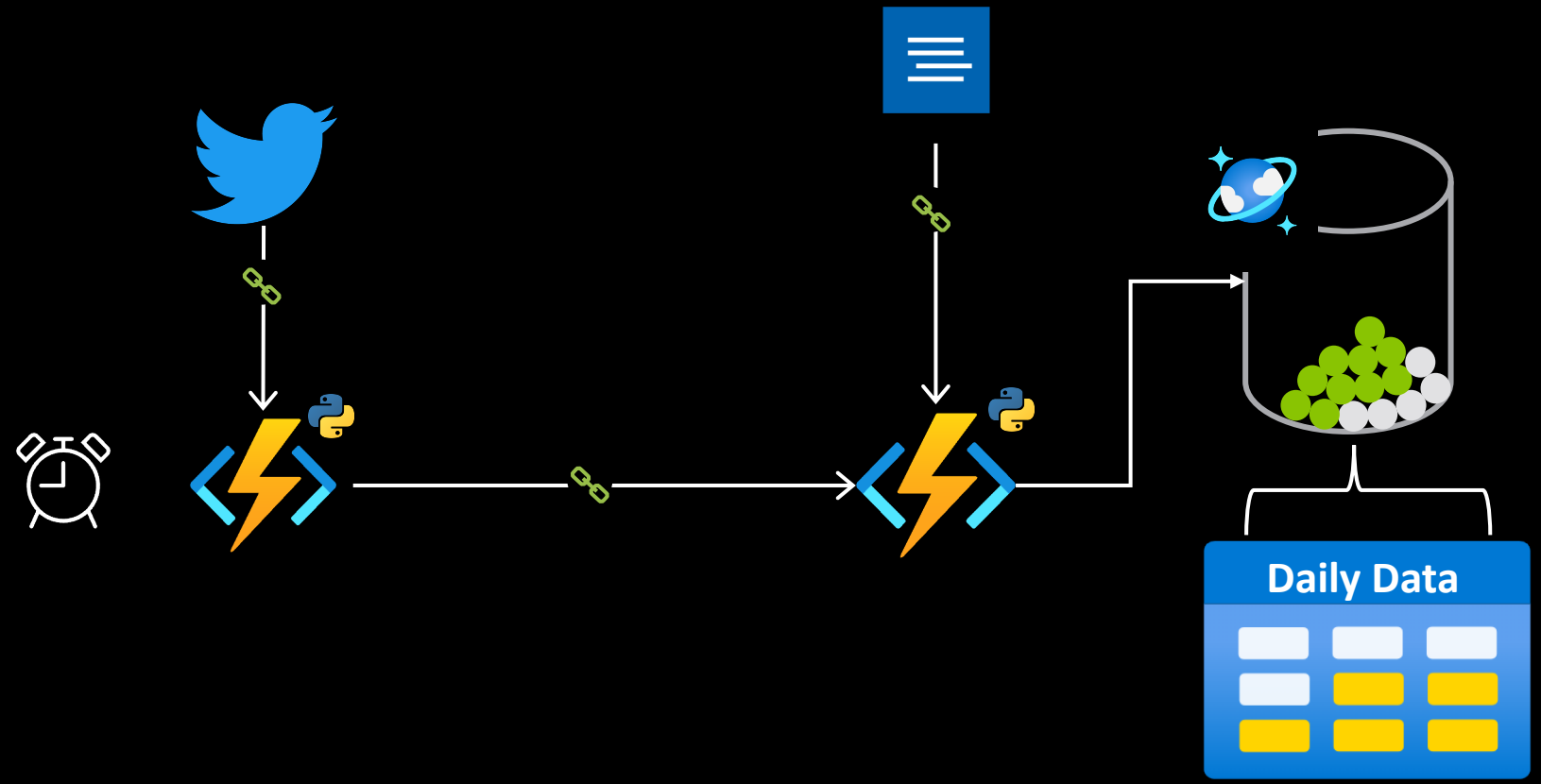


+ Close to complete solution
- Reader/Writer problems

Making Tweets-Elonyzer

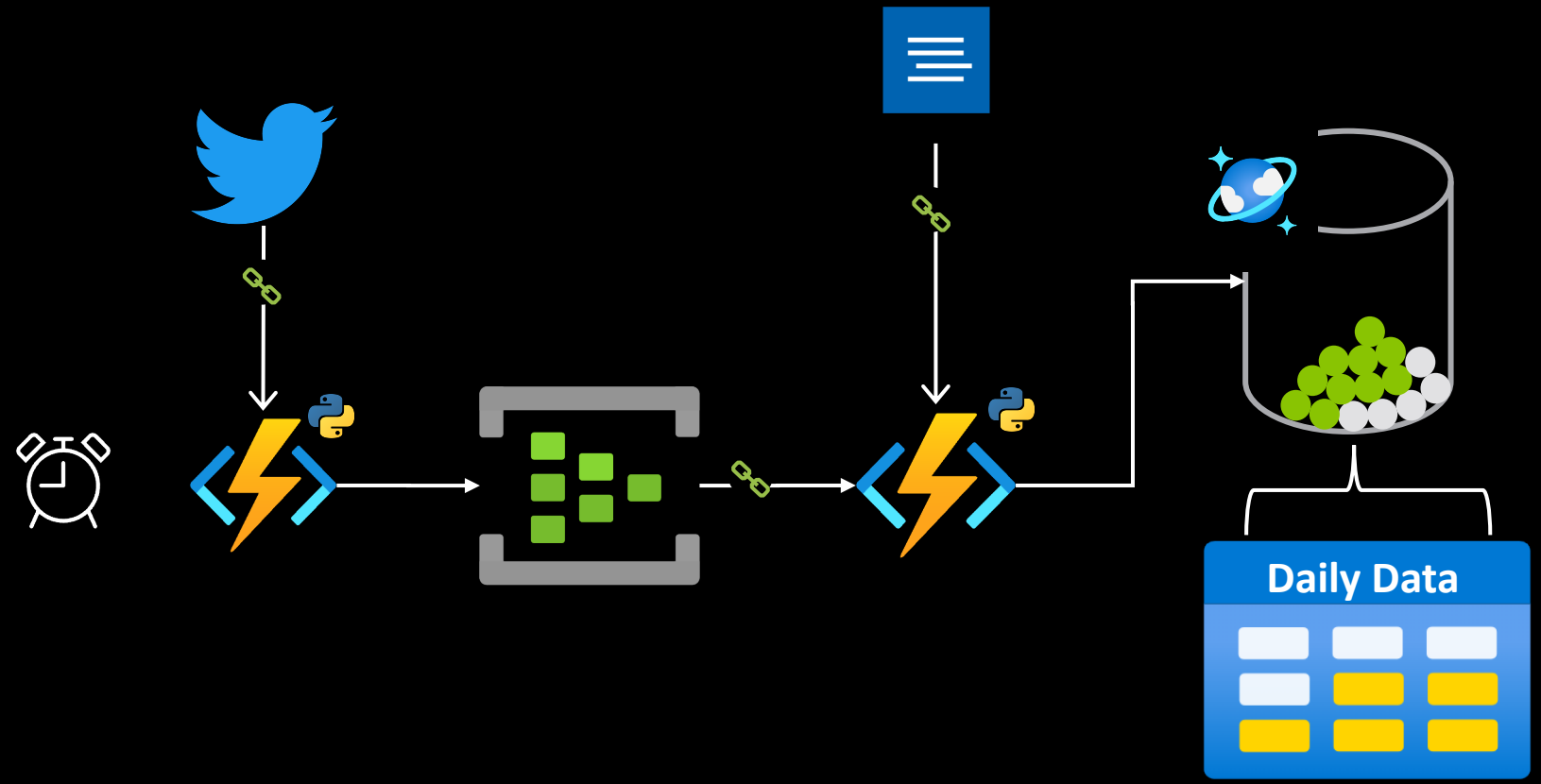


Making Tweets-Elonyzer

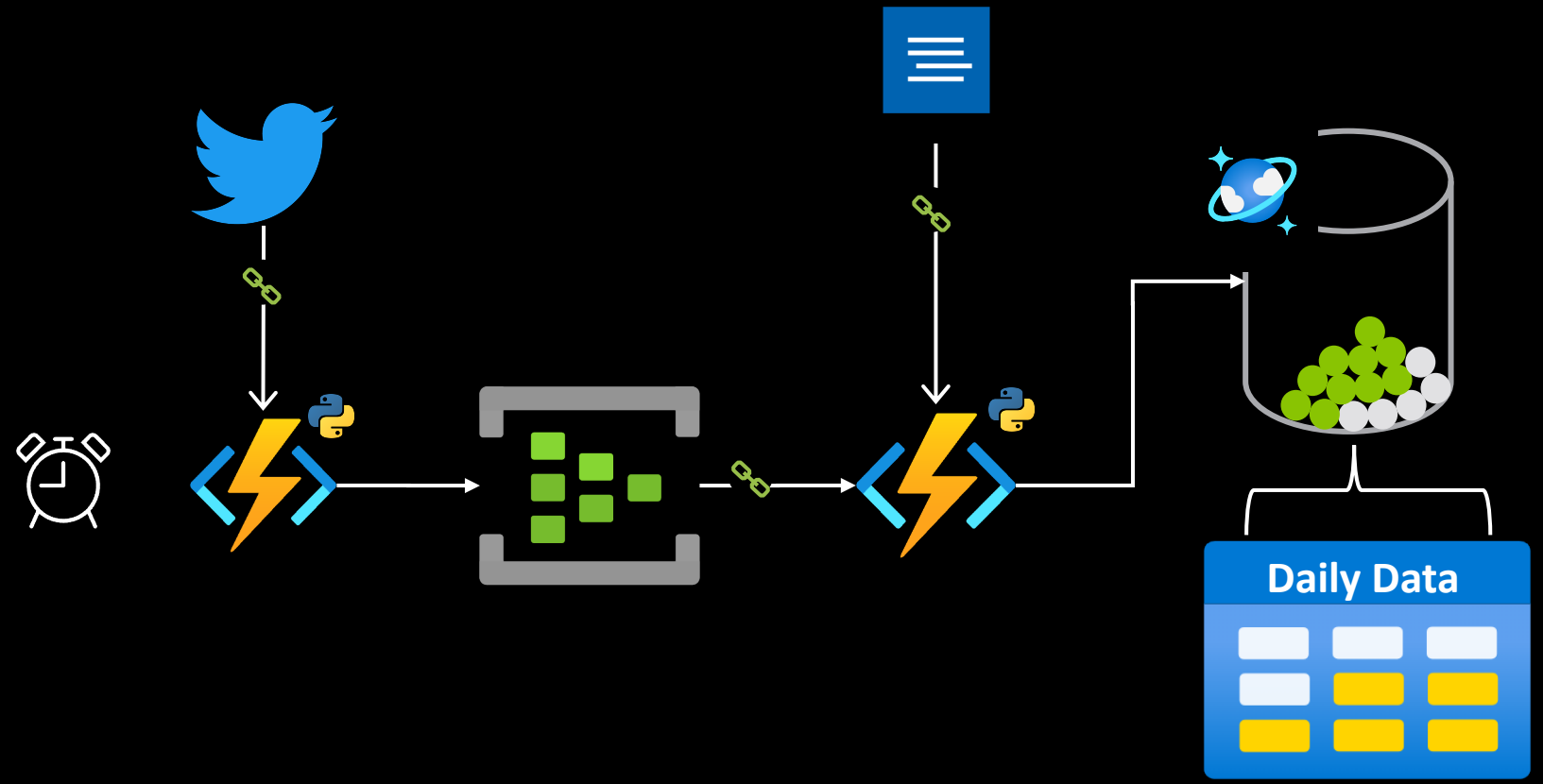


+ Close to complete solution
- Inter-function communication problems, at high scale

Making Tweets-Elonyzer

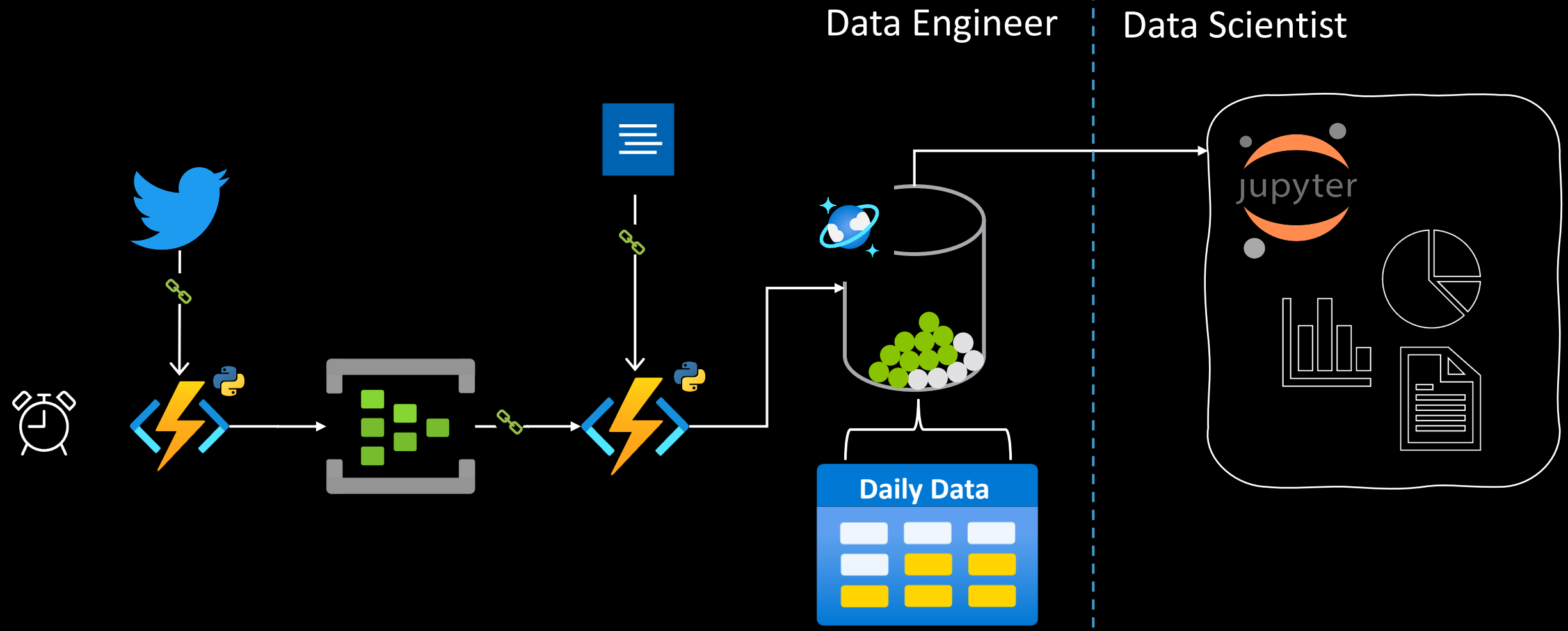


Making Tweets-Elonyzer



+ Close to complete solution
- Most-likely nothing. 🤞🏻

Tweets-Elonyzer: Overview



Tweets-analyzer – FetchTweets

```
recent_tweets = [
    TWITTER_CLIENT.search_recent_tweets(term, max_results=FETCH_TWEETS_COUNT).data
    for term in SEARCH_TERMS
]

flattened_recent_tweets = functools.reduce(operator.iconcat, recent_tweets, [])

logging.info(
    f"Fetched {len(flattened_recent_tweets)} tweets from twitter for {SEARCH_TERMS} terms"
)

return [json.dumps(tweet) for tweet in map(dictify_tweet, flattened_recent_tweets)]
```

Tweets-analyzer – AnalyzeLoadTweets

```python
def main(events: List[func.EventHubEvent], tweetdoc: func.Out[func.Document]):
    logging.info(f"Got {len(events)} events from EventHub")

    tweets_json = []
    for event in events:
        tweet_dict = json.loads(event.get_body().decode("utf-8"))
        response = AZURE_AI_CLIENT.analyze_sentiment(
            documents=[tweet_dict["text"]], show_opinion_mining=True
        )[0]
        add_text_analysis_to_tweet(tweet_dict=tweet_dict, response=response)
        tweets_json.append(tweet_dict)

    logging.info(f"{len(tweets_json)} records being sent to CosmosDB")
    for tweet in tweets_json:
        tweetdoc.set(func.Document.from_json(json.dumps(tweet)))
```

Data Visualization with Jupyer Notebooks

Consuming the data

```
{
    "data": {
        "id": "1518841131819143169",
        "text": "#pycon If you are attending @pycon and are interested in building cloud-agnostic cloud-native microservices come talk to me
    },
    "id": "1518841131819143169",
    "text": "#pycon If you are attending @pycon and are interested in building cloud-agnostic cloud-native microservices come talk to me abou
    "attachments": null,
    "author_id": null,
    "context_annotations": null,
    "conversation_id": null,
    "created_at": null,
    "entities": null,
    "geo": null,
    "in_reply_to_user_id": null,
    "lang": null,
    "non_public_metrics": null,
    "organic_metrics": null,
    "possibly_sensitive": null,
    "promoted_metrics": null,
    "public_metrics": null,
    "referenced_tweets": null,
    "reply_settings": null,
    "source": null,
    "withheld": null,
    "sentiment": "neutral",
    "confidence_scores": {
        "positive": 0.05,
        "neutral": 0.93,
        "negative": 0.02
    }
}
```

What are we working on?

Sneak Peak: Azure Functions Python programming model

New Features

- Decorator based approach
- Reduce configuration files
- Manager code easier

Example: FetchTweets

```
@app.function_name("FetchTweets")
@app.schedule(schedule="0 0 * * * *", arg_name="timer", run_on_startup=True)
@app.write_event_hub_message(
    arg_name="$return", event_hub_name="tweets_queue", connection="TWEETS_EH_CONNECTION"
)
def main(timer: func.TimerRequest) -> str:
    recent_tweets = [
        TWITTER_CLIENT.search_recent_tweets(term, max_results=FETCH_TWEETS_COUNT).data
        for term in SEARCH_TERMS
    ]

    flattened_recent_tweets = functools.reduce(operator.iconcat, recent_tweets, [])

    logging.info(
        f"Fetched {len(flattened_recent_tweets)} tweets from twitter for {SEARCH_TERMS} terms"
    )

    return [json.dumps(tweet) for tweet in map(dictify_tweet, flattened_recent_tweets)]
```

Example: AnalyzeLoadTweets

```python
@app.function_name(name="AnalyzeLoadTweets")
@app.on_event_hub_message(
    arg_name="events",
    event_hub_name="tweets_queue",
    connection="TWEETS_EH_CONNECTION",
    cardinality="many",
    consumer_group="$Default",
)
@app.write_cosmos_db_documents(
    arg_name="tweetdoc",
    database_name="tweetsdatabase",
    collection_name="tweetscollection",
    connection_string_setting="tweetspycon_DOCUMENTDB",
)
def send_to_cosmos_db(
    events: List[func.EventHubEvent], tweetdoc: func.Out[func.Document]
):
    logging.info(f"Got {len(events)} events from EventHub")

    tweets_json = []
    for event in events:
        tweet_dict = json.loads(event.get_body().decode("utf-8"))
        response = AZURE_AI_CLIENT.analyze_sentiment(
            documents=[tweet_dict["text"]], show_opinion_mining=True
        )[0]
        add_text_analysis_to_tweet(tweet_dict=tweet_dict, response=response)
        tweets_json.append(tweet_dict)

    logging.info(f"{len(tweets_json)} records being sent to CosmosDB")
    for tweet in tweets_json:
        tweetdoc.set(func.Document.from_json(json.dumps(tweet)))
```

Our major takeaways

1. Data analysis, data science, and data engineering are becoming the most popular workloads for Python (at least from our research!).

2. More and more Python developers are turning to serverless as a potential solution for handling data at scale, due to its elasticity and event-driven model.

3. Based on customer data, event-driven solutions are one of the fastest and easiest ways to handle this complexity.

Thank you!

☺

Microsoft @PyCon

Take our survey to help us learn about your Python needs and earn swag!

aka.ms/pyconus22survey

Attend Anthony Shaw's Talk on Saturday!

Write Faster Python! Common performance anti-patterns.

Saturday, April 30 - 5-5:45 pm Room 355DEF

Resources

GitHub repo for Elonyzer: aka.ms/pycon-azure-demo