TEXAS ADVANCED COMPUTING CENTER

WWW.TACC.UTEXAS.EDU

TEXAS
The University of Texas at Austin

🧠 Build a Job Queue API with Flask, Redis, and HotQueue

Putting it all together

PRESENTED BY:

🟦 Today’s Goal

✅ Understand how job queues work
✅ Submit jobs using a Flask route
✅ Track job status with Redis
✅ Run background jobs using HotQueue

🟦 What is a Job Queue?

🧾 A job queue allows you to:

- Submit a task (e.g., analyze traffic data)
- Let it run *later* in the background
- Check its progress or result

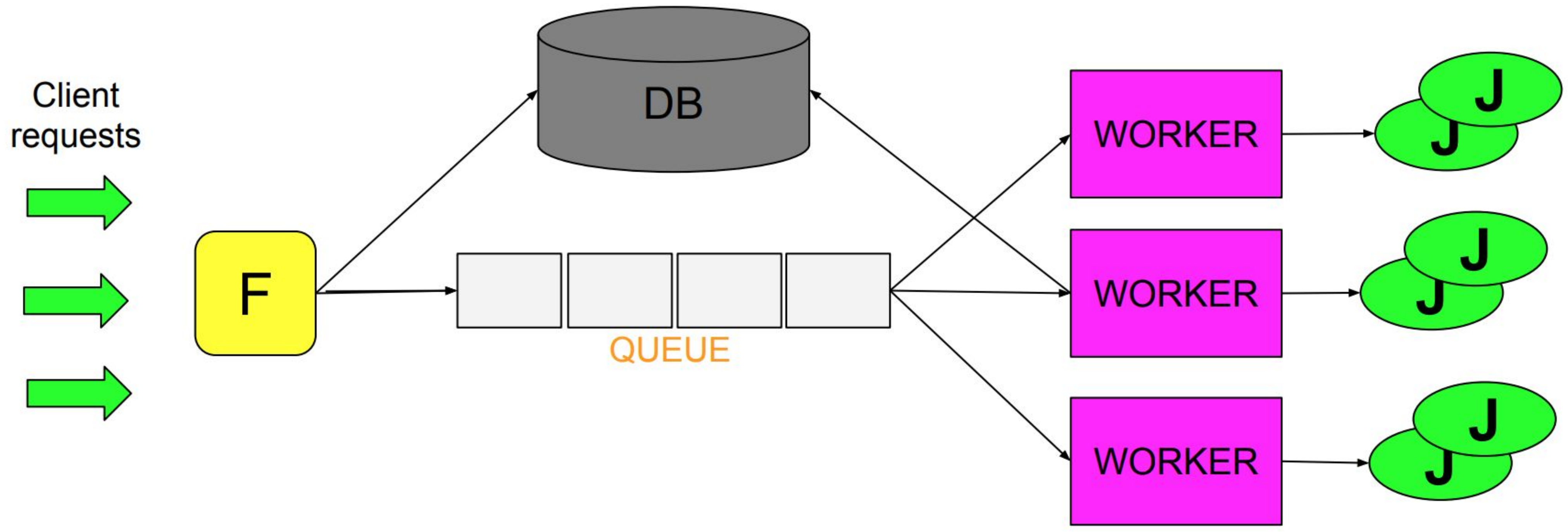
🧠 It separates **task submission** from **task processing**

🟦 Analogy: The Food Truck

1. You place an order (Flask `/submit`)
2. It goes into the order list (Redis Queue)
3. The cook (Worker) makes the food
4. You check if your food is ready (`/status/<job_id>`)

🍔 ➡️ ⏳ ➡️ 👩🏾‍🍳 ➡️ ✅

Software Engineering, Basic Framework



🟦 Key Tools

Tool	Purpose
Flask	Web server & API routes
Redis	In-memory key-value store
HotQueue	Queuing job IDs in Redis
Worker	Python script that processes jobs
`uuid`	Unique IDs for job tracking

🟦 Flask API - Submit Job

```
@app.route("/submit", methods=["POST"])
def submit_job():
    job_id = str(uuid.uuid4())
    job = {"id": job_id, "status": "submitted", "input": request.json}
    rdb.set(job_id, str(job))
    queue.put(job_id)
    return jsonify({"job_id": job_id})
```

🟦 Flask API - Get Job Status

```
@app.route("/status/<job_id>")
def check_job(job_id):
    job = rdb.get(job_id)
    return jsonify(eval(job)) if job else ("Not found", 404)
```

🟦 Worker Script (HotQueue Worker)

```
@queue.worker
def work():
    while True:
        job_id = queue.get()
        job = eval(rdb.get(job_id))
        job["status"] = "in-progress"
        rdb.set(job_id, str(job))

        # Simulate a long job
        result = {"message": f"Processed {job['input']}"}
        time.sleep(2)

        job["status"] = "complete"
        job["result"] = result
        rdb.set(job_id, str(job))
```

🟦 Testing with Python

```
import requests

# Submit a job
job = requests.post("http://localhost:5000/submit", json={"date":
"2023-01-01"}).json()
print("Job ID:", job["job_id"])

# Poll for status
while True:
    status = requests.get(f"http://localhost:5000/status/{job['job_id']}").json()
    if status["status"] == "complete":
        print("Result:", status["result"])
        break
```

🔁 Full Job Flow

🔁 **User hits**:
`POST /submit` (JSON payload sent to Flask)

🧠 **Flask app does**:

- Generates a unique `job_id`
- Saves the full job (input + metadata) in Redis (`db=0`)
- Pushes just the `job_id` into a Redis-backed HotQueue (`db=1`)

⚙️ **Worker.py does**:

- Listens to the HotQueue
- Pulls `job_id` from the queue
- Loads job details from Redis using `job_id`
- Performs the analysis (e.g., filters Austin traffic data)
- Updates job in Redis with `status = complete` + result

📲 **User hits**:
`GET /status/<job_id>` to retrieve results

💡 So What Happens to the Route Logic?

Let’s say you had this Flask route from earlier in the week:

```
@app.route("/incident_count_by_date")

def count_incidents():

    date = request.args.get("date")

    result = traffic_df[traffic_df["date"] == date].shape[0]

    return jsonify({"count": result})
```

💡 So What Happens to the Route Logic?

```
# worker.py

@queue.worker

def process_jobs():

    while True:

        job_id = queue.get().decode("utf-8")

        job = json.loads(rdb.get(job_id))


        job["status"] = "in-progress"

        rdb.set(job_id, json.dumps(job))

        # 💥 This is where you move the analysis logic:

        date = job["input"].get("date")

        count = traffic_df[traffic_df["date"] == date].shape[0]

        job["result"] = {"incident_count": count}


        job["status"] = "complete"

        rdb.set(job_id, json.dumps(job))
```

Suggestion!

Don't delete the route logic right away.
Instead:

1. Keep the route version active for direct API testing.
2. Move the same logic into a **Python function** (e.g., `analyze_date(date)`).
3. Call that function from both the route and the worker.

```
def analyze_date(date):

    return traffic_df[traffic_df["date"] == date].shape[0]
```

Then:

- In Flask:
  `return jsonify({"count": analyze_date(request.args.get("date"))})`
- In Worker:
  `job["result"] = {"incident_count": analyze_date(job["input"]["date"])}`

We need a some way of telling the worker which method needs to run

the worker needs a way to know which method to run when a job comes in. That means:

The job submission should include:

- A job_id (autogenerated)
- A task or operation field that tells the worker which method to run
- Any required input parameters (e.g., date, location, etc.)

✅ Recommended JSON Job Format

```
{
  "operation": "incident_count_by_date",
  "params": {
    "date": "2023-01-01"
  }
}
```

Then the worker would look for `operation`, and dynamically call the correct function.

🧠 Implementation Plan

1. Define Task Functions

```
# analysis.py

def incident_count_by_date(df, date):
    return {"incident_count": df[df["date"] == date].shape[0]}

def top_locations(df, n=10):
    return {"top_locations": df["Location"].value_counts().head(n).to_dict()}

def average_response_time(df):
    return {"avg_response": df["ResponseTime"].mean()}
```

🧠 Implementation Plan

2. Modify Worker to Dispatch Tasks

```
# worker.py
from analysis import *
TASK_MAP = {
    "incident_count_by_date": incident_count_by_date,
    "top_locations": top_locations,
    "average_response_time": average_response_time
}

@queue.worker
def process_jobs():
    while True:
        job_id = queue.get().decode("utf-8")
        job = json.loads(rdb.get(job_id))
```

```
job["status"] = "in-progress"
rdb.set(job_id, json.dumps(job))

op = job["input"].get("operation")
params = job["input"].get("params", {})

try:
    result = TASK_MAP[op](traffic_df, **params)
    job["status"] = "complete"
    job["result"] = result
except Exception as e:
    job["status"] = "failed"
    job["result"] = {"error": str(e)}

rdb.set(job_id, json.dumps(job))
```

🧠 Implementation Plan

3. Modify the `/submit` Route in Flask

```
@app.route("/submit", methods=["POST"])
def submit_job():
    job_id = str(uuid.uuid4())
    data = request.get_json()

    job = {
        "id": job_id,
        "status": "submitted",
        "input": data,
        "result": None
    }

    rdb.set(job_id, json.dumps(job))
    queue.put(job_id)

    return jsonify({"job_id": job_id, "status": "submitted"}), 202
```

🚀 Example Submission

```
# Submit a top 5 location job
requests.post("http://localhost:5000/submit", json={
    "operation": "top_locations",
    "params": {"n": 5}
})
```

Sample Code: Jobqueue, Flask Worker

```
### analysis.py


def incident_count_by_date(df, date):

    count = df[df['date'] == date].shape[0]

    return {"incident_count": count}


def top_locations(df, n=10):

    top =
df['Location'].value_counts().head(n).to_dict()

    return {"top_locations": top}


def average_response_time(df):

    if "ResponseTime" in df.columns:

        avg = df["ResponseTime"].mean()

        return {"avg_response": avg}

    return {"error": "ResponseTime column missing"}
```

```
def unique_issues(df):

    return {"unique_issues":
df["Issue_Report"].nunique()}


def incidents_by_road(df, road):

    filtered = df[df["Location"].str.contains(road,
case=False, na=False)]

    return {"count": filtered.shape[0]}
```

Sample Code: Submit Template

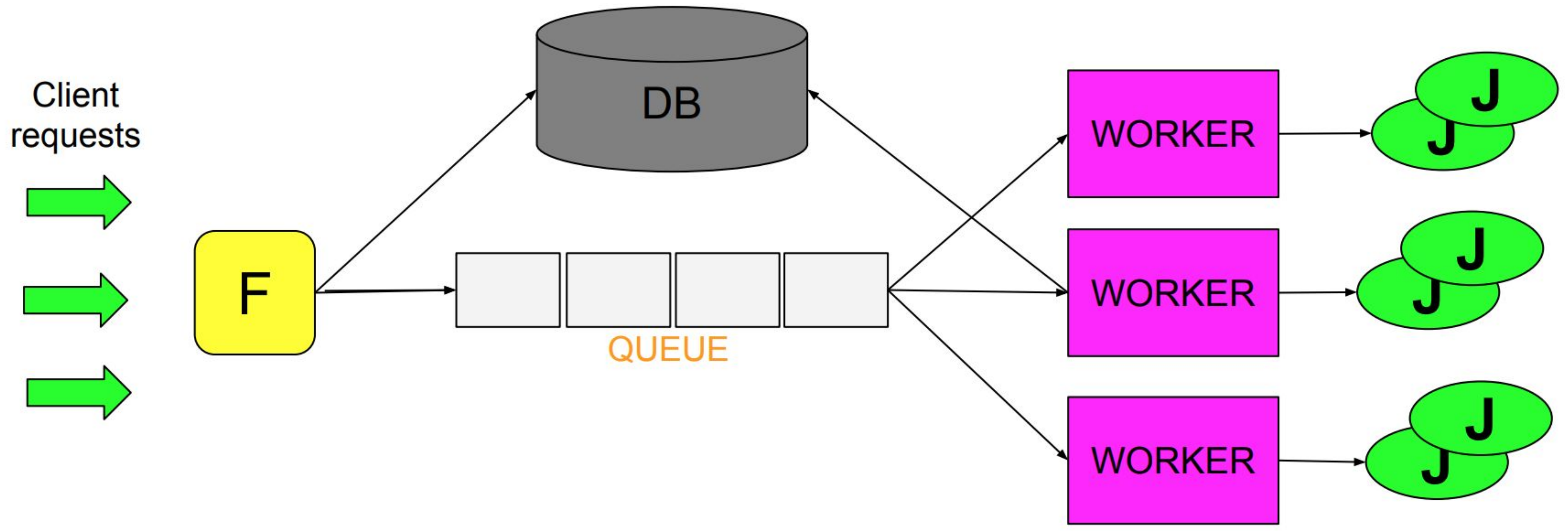
```
// submit_templates.json

{
  "incident_count_by_date": {
    "operation": "incident_count_by_date",
    "params": {
      "date": "2023-01-01"
    }
  },
  "top_locations": {
    "operation": "top_locations",
    "params": {
      "n": 5
    }
  },
  "average_response_time": {
    "operation": "average_response_time",
    "params": {}
  },
  "unique_issues": {
    "operation": "unique_issues",
    "params": {}
  },
  "incidents_by_road": {
    "operation": "incidents_by_road",
    "params": {
      "road": "S CONGRESS"
    }
  }
}
```

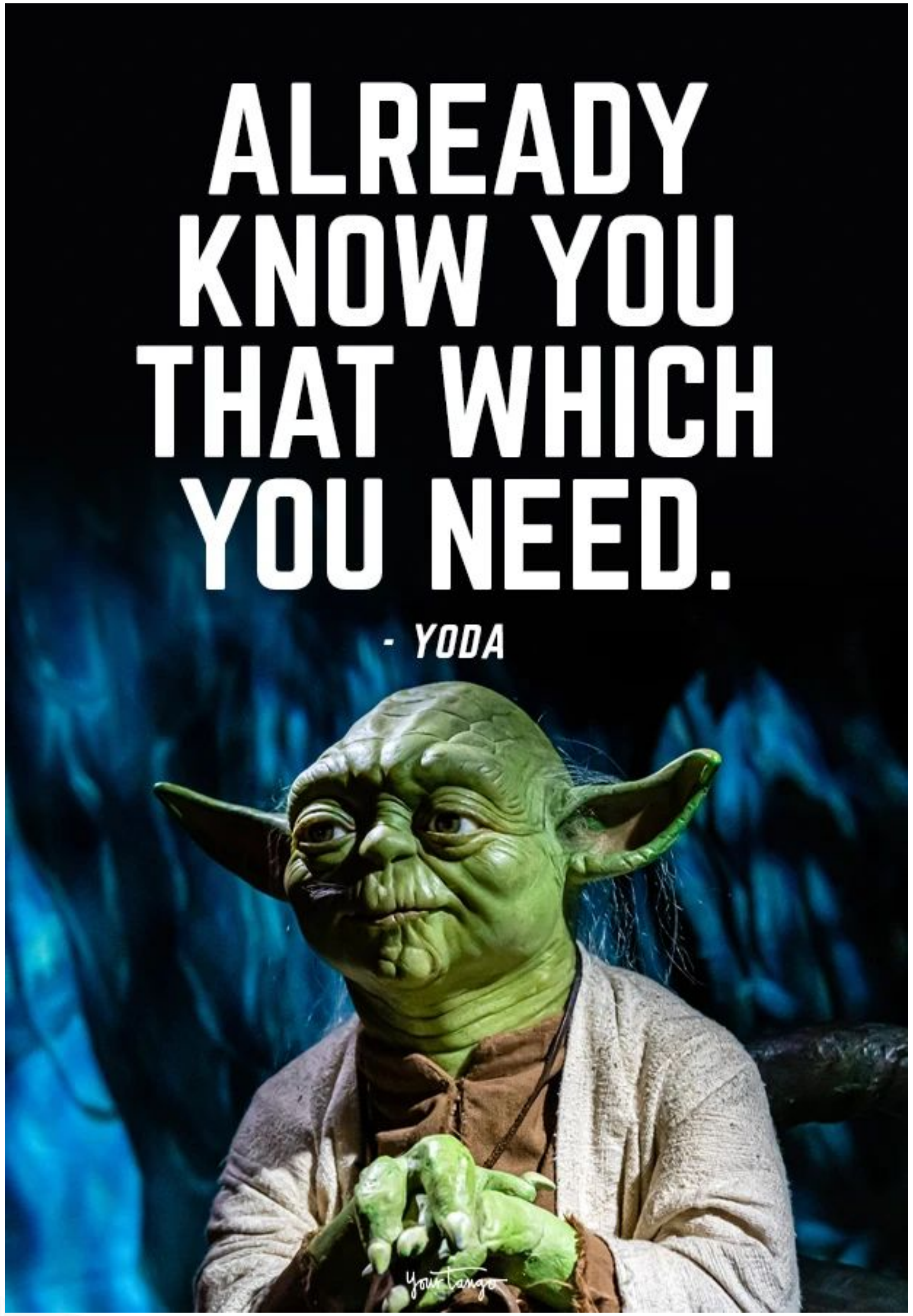
Sample Code: Worker Dispatch

```
# worker.py
from hotqueue import HotQueue
import redis, json, time
from analysis import *
import pandas as pd

# Load the data once globally
traffic_df = pd.read_csv("data/atx_traffic.csv")

rdb = redis.Redis(host="redis", port=6379, db=0)
queue = HotQueue("job-queue", host="redis", port=6379,
db=1)

TASK_MAP = {
    "incident_count_by_date": incident_count_by_date,
    "top_locations": top_locations,
    "average_response_time": average_response_time,
    "unique_issues": unique_issues,
    "incidents_by_road": incidents_by_road
```

```
@queue.worker
def process_jobs():
    while True:
        job_id = queue.get().decode("utf-8")
        job = json.loads(rdb.get(job_id))
        job["status"] = "in-progress"
        rdb.set(job_id, json.dumps(job))

        op = job["input"].get("operation")
        params = job["input"].get("params", {})
        try:
            result = TASK_MAP[op](traffic_df, **params)
            job["status"] = "complete"
            job["result"] = result
        except Exception as e:
            job["status"] = "failed"
            job["result"] = {"error": str(e)}

        rdb.set(job_id, json.dumps(job))
```

Software Engineering, Basic Framework




ALREADY KNOW YOU THAT WHICH YOU NEED.
- YODA
YourTango