TEXAS ADVANCED COMPUTING CENTER

WWW.TACC.UTEXAS.EDU

TEXAS
The University of Texas at Austin

🔷 Intro to Redis & HotQueue: Powering Background Jobs

PRESENTED BY:

Why Background Jobs?

- APIs can’t always process everything instantly
- Some tasks take time: data analysis, ML inference, PDF generation...
- Users don’t want to wait!

What is Redis?

In-memory key-value store

Fast, lightweight, used as:

✅ Cache

✅ Message broker

✅ Pub/sub system

Redis is often described as a "data structure server." We'll use it as a queue backend.

What is HotQueue?

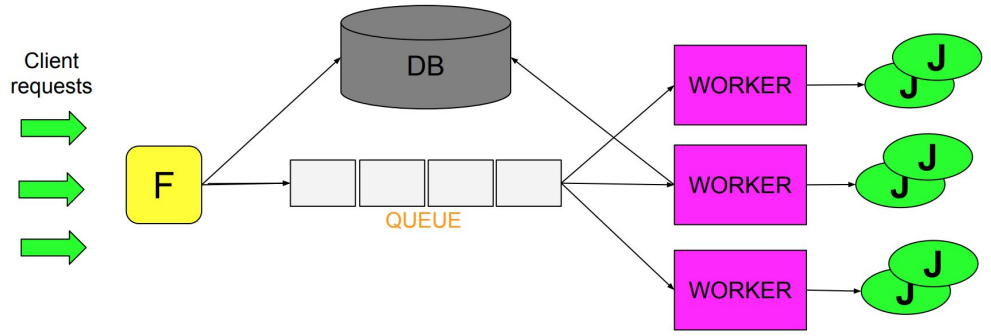
- A simple Python wrapper around Redis
- Lets us queue jobs and process them with worker functions
- Great for teaching and lightweight use cases :)

```
from hotqueue import HotQueue
q = HotQueue("queue", host="localhost", port=6379)
q.put("do_something")
```

TACC

Queue Architecture

1. Client submits job (e.g. via API)
2. Flask app puts job in Redis queue
3. Worker polls Redis for jobs
4. Worker runs task and stores result



This model decouples job *submission* from *execution*. The user moves on while the job runs in the background

Sample Queue Code

```
# in app.py
q.put({"type": "plot", "params": {"date":
"2024-06-01"}})


# in worker.py
for job in q.consume():
    if job["type"] == "plot":
        generate_plot(job["params"])
```

Demo Flow

You'll Build This Today

- `/job/submit` → accepts JSON and queues job
- Redis stores pending jobs
- Worker processes jobs in background
- `/job/result/<id>` → gets result when ready

Common Use Cases in Data Apps

- Heavy data processing (traffic analysis!)
- Image/video manipulation
- ML model inference
- Scheduled jobs (e.g., run every hour)

Getting Started with Redis & HotQueue

```
$>docker pull redis
$>docker run -d --name redis-server -p 8020:6379 redis
$>docker ps
$>pip3 install hotqueue redis
```

***NOTE:**

red - this is *your* port on the server

blue - this is the port *inside* the container

Hit the Queue!

```
from hotqueue import HotQueue
import time

# Connect to Redis and create a queue named 'job_queue'
q = HotQueue("job_queue", host="localhost", port= XXXX)

def producer():
    # Put some sample jobs on the queue
    for i in range(5):
        job_data = {"type": "print", "msg": f"Hello {i}"}
        print(f"Producer: Adding job {job_data}")
        q.put(job_data)
        time.sleep(1)
```

```
def consumer():
    print("Consumer: Waiting for jobs...")
    for job in q.consume():
        print(f"Consumer: Got job: {job}")
        if job["type"] == "print":
            print(f"Message: {job['msg']}")
        else:
            print("Unknown job type.")
```

Hit the Queue!

```
if __name__ == "__main__":
    import threading

    # Run producer and consumer in separate threads for
demo
    t_producer = threading.Thread(target=producer)
    t_consumer = threading.Thread(target=consumer)

    t_consumer.start()
    t_producer.start()

    t_producer.join()
    # Note: consumer runs infinitely waiting for jobs,
    # so we won't join it here to keep demo simple
```