Github Repo link: :
https://github.com/abhishek-jha-24/273-project

Architecture

Service A (Port 8080)
- `/health` - Returns service status
- `/echo?msg=...` - Echoes back the provided message

Service B (Port 8081)
- `/health` - Returns service status
- `/call-echo?msg=...` - Calls Service A's `/echo` endpoint and returns the result

Service B includes:
- Timeout handling (1 second timeout on requests to Service A)
- Error logging with HTTP status codes
- Graceful degradation when Service A is unavailable (returns 503)

How to Run Locally

Prerequisites
- Python 3.8+
- pip (Python package manager)

Setup & Run

Terminal 1 - Start Service A:
```bash
cd python-http/service-a
pip install -r requirements.txt
python app.py
```

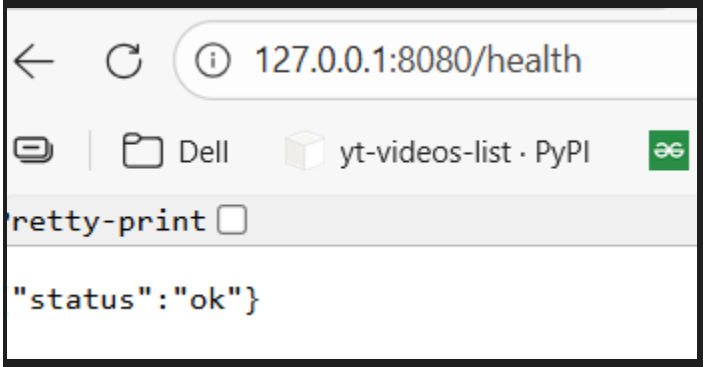
Terminal 2 - Start Service B (in a new terminal):

```bash
cd python-http/service-b
pip install -r requirements.txt
python app.py
```

You should see output indicating both services are running:
- Service A: `Running on http://127.0.0.1:8080`
- Service B: `Running on http://127.0.0.1:8081`

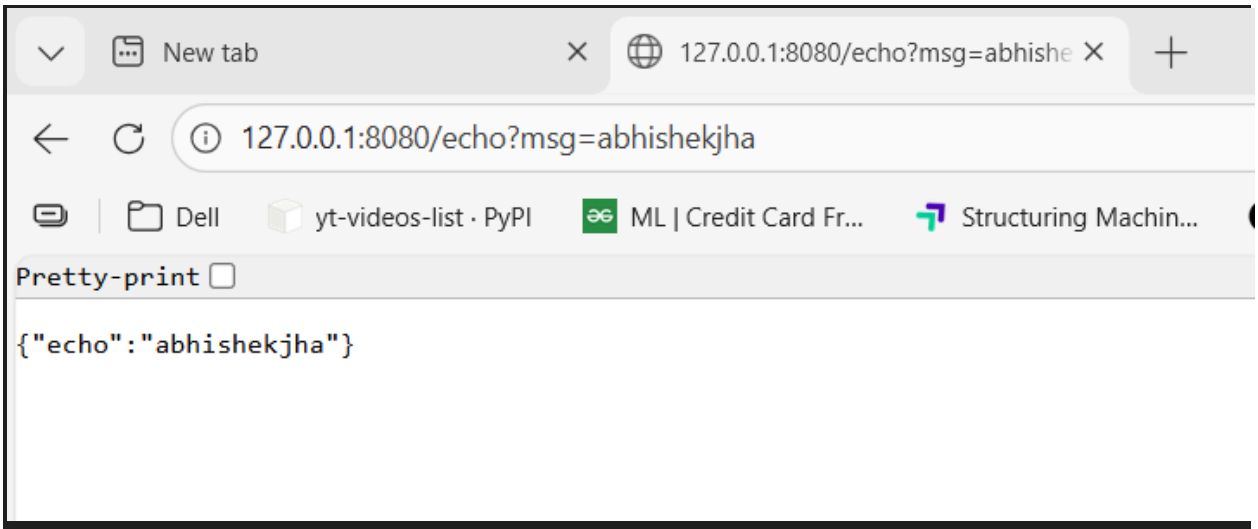
Testing

Service A's endpoints:



Log:

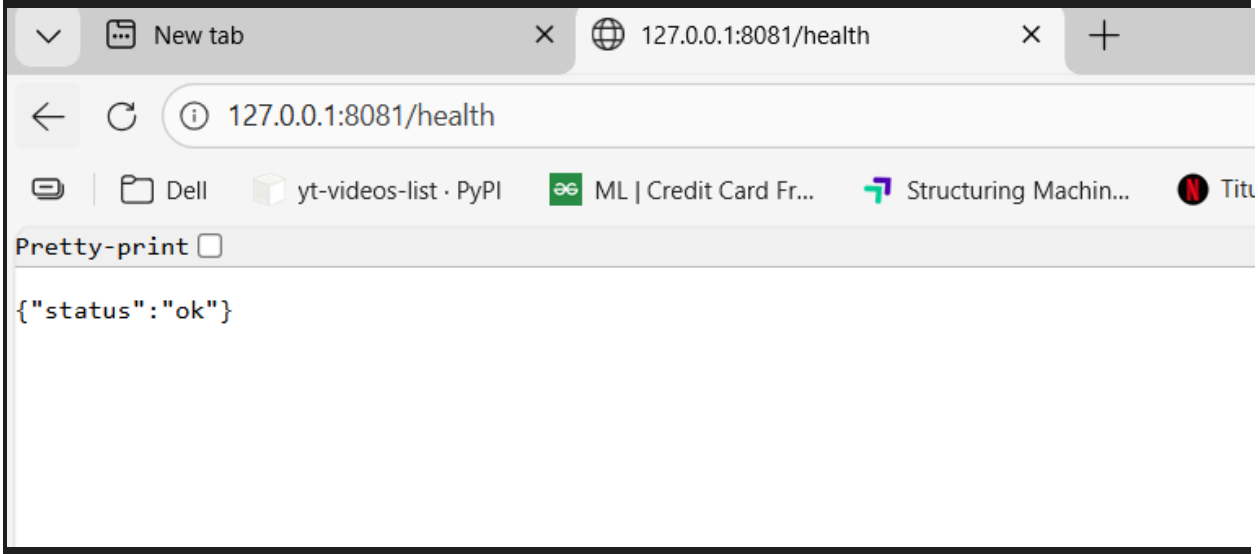
```
2026-02-04 21:13:23,463 Press CTRL+C to quit
2026-02-04 21:13:42,655 127.0.0.1 - - [04/Feb/2026 21:13:42] "GET /health HTTP/1.1" 200 -
```



Log:

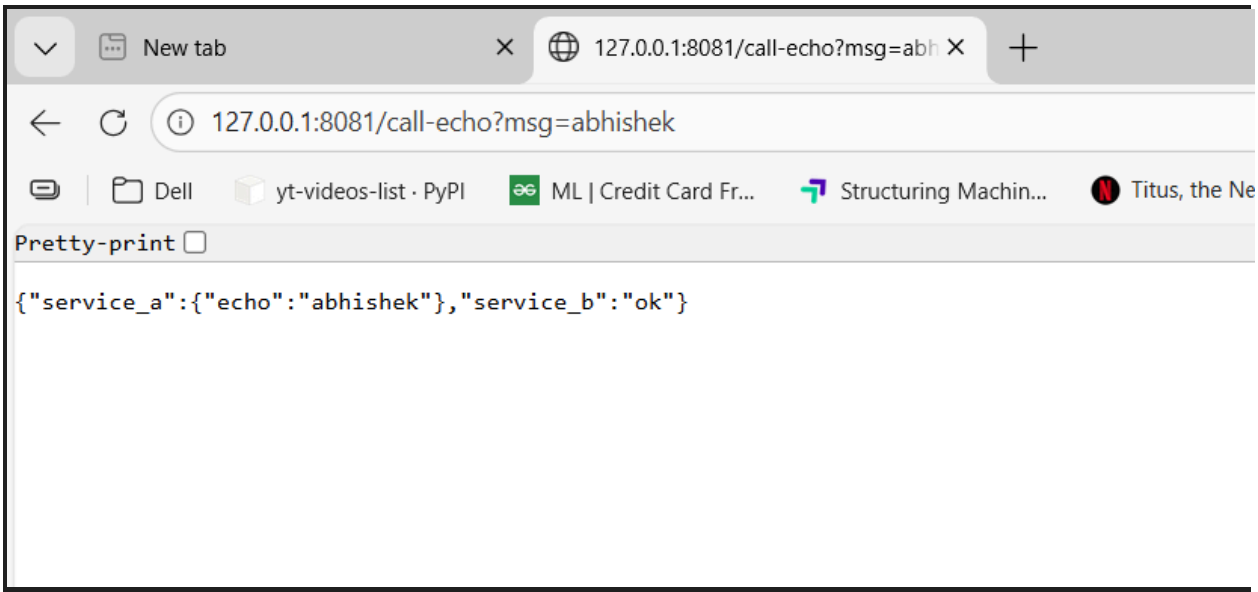
```
2026-02-04 21:14:22,663 service=A endpoint=/echo status=ok latency_ms=0
2026-02-04 21:14:22,665 127.0.0.1 - - [04/Feb/2026 21:14:22] "GET /echo?msg=abhishekjha HTTP/1.1" 200 -
```

####### Service B's endpoints: #######



Log:

```
2026-02-04 21:15:50,833 127.0.0.1 - - [04/Feb/2026 21:15:50] "GET /health HTTP/1.1" 200 -
```



Log:

```
2026-02-04 21:16:33,747 service=B endpoint=/call-echo status=ok latency_ms=24
2026-02-04 21:16:33,750 127.0.0.1 - - [04/Feb/2026 21:16:33] "GET /call-echo?msg=abhishek HTTP/1.1" 200 -
```

Testing successful communication:

curl "http://127.0.0.1:8081/call-echo?msg=abhishek"

Response:

```json
{
  "service_b": "ok",
  "service_a": {
    "echo": "abhishek"
  }
}
```

```
PS C:\Users\abhi2\Documents\273\273-project> curl "http://127.0.0.1:8081/call-echo?msg=abhishek"

Security Warning: Script Execution Risk
Invoke-WebRequest parses the content of the web page. Script code in the web page might be run when the page
is parsed.
      RECOMMENDED ACTION:
      Use the -UseBasicParsing switch to avoid script code execution.

      Do you want to continue?

[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"): y


StatusCode        : 200
StatusDescription : OK
Content           : {"service_a":{"echo":"abhishek"},"service_b":"ok"}

RawContent        : HTTP/1.1 200 OK
                    Connection: close
                    Content-Length: 51
                    Content-Type: application/json
                    Date: Thu, 05 Feb 2026 05:08:21 GMT
                    Server: Werkzeug/3.1.5 Python/3.13.9

                    {"service_a":{"echo":"abhishek"},"s...
Forms             : {}
Headers           : {[Connection, close], [Content-Length, 51], [Content-Type, application/json], [Date,
                    Thu, 05 Feb 2026 05:08:21 GMT]...}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 51
```

Test health check:

```bash
curl "http://127.0.0.1:8080/health"
```

```
                                        curl "http://127.0.0.1:8080/health"
>> C:\Users\abhi2\Documents\273\273-project>

  Security Warning: Script Execution Risk
  Invoke-WebRequest parses the content of the web page. Script code in the web page might be run when the page
  is parsed.
        RECOMMENDED ACTION:
        Use the -UseBasicParsing switch to avoid script code execution.

        Do you want to continue?

  [Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"): y


  StatusCode        : 200
  StatusDescription : OK
  Content           : {"status":"ok"}

  RawContent        : HTTP/1.1 200 OK
                      Connection: close
                      Content-Length: 16
                      Content-Type: application/json
                      Date: Thu, 05 Feb 2026 05:10:43 GMT
                      Server: Werkzeug/3.1.5 Python/3.13.9

                      {"status":"ok"}

  Forms             : {}
  Headers           : {[Connection, close], [Content-Length, 16], [Content-Type, application/json], [Date,
                      Thu, 05 Feb 2026 05:10:43 GMT]...}
  Images            : {}
  InputFields       : {}
  Links             : {}
  ParsedHtml        : mshtml.HTMLDocumentClass
  RawContentLength  : 16
```

Test failure scenario:

1. Stopping Service A (press Ctrl+C in Terminal 1)
2. calling Service B:

curl "http://127.0.0.1:8081/call-echo?msg=hello"

Response:

```
json
{
 "service_b": "ok",
 "service_a": "unavailable",
```

```
  "error": "timeout",
  "http_status": 504
}
```

```
PS C:\Users\abhi2\Documents\273\273-project> curl "http://127.0.0.1:8081/call-echo?msg=abhishek"
curl : {"error":"timeout","http_status":504,"service_a":"unavailable","service_b":"ok"}
```

```

# ## What Makes This Distributed?

This system demonstrates core distributed systems principles.

**Independent Processes** - Service A and Service B run as separate Python processes that can be started, stopped, and restarted independently, allowing them to have different lifetimes and failure modes.

**Network Communication** - Services communicate exclusively over HTTP/TCP, not through shared memory or direct function calls, simulating real-world distributed communication across network boundaries.

**Fault Isolation** - Failure of Service A does not crash Service B; instead, Service B gracefully handles the error with a 1-second timeout and returns a 503 status code, allowing clients to understand the degraded state and react accordingly.

**Timeout & Resilience** - Service B implements timeout logic (1 second) to prevent indefinite blocking when Service A is unavailable, a critical resilience pattern in distributed systems that prevents cascading failures.

# ## When happens on Timeout?

**Detection** - The client library (requests) detects that the network operation exceeded the time limit and raises a Timeout exception.

**Graceful Degradation** - Service B catches this exception and doesn't crash. Instead, it logs the failure with HTTP status code 504 (Gateway Timeout).

**Fault Isolation** - The failure in Service A is isolated; Service B continues running and remains available to handle other requests.

**Client Notification** - Service B returns HTTP 503 (Service Unavailable) to inform the client that Service A is unreachable, allowing the client to implement retry logic or fallback strategies.

**Resilience Pattern** - This prevents resource exhaustion and enables system recovery by releasing stuck connections
```

.

What happens if service A is down?

When Service A is down, the distributed system demonstrates critical resilience patterns:
Dependency Failure - Service B cannot establish a connection to Service A, simulating a real-world scenario where a dependent service becomes unavailable due to crashes, network issues, or maintenance.
Timeout Protection - Rather than waiting indefinitely, Service B enforces a 1-second timeout. This prevents resource exhaustion and thread starvation by releasing blocked connections.
Fault Isolation - Service A's failure is contained. Service B doesn't crash; it remains operational and continues accepting requests from clients.
Graceful Degradation - Service B returns HTTP 503 to inform clients that Service A is unavailable, enabling them to implement retry logic or use fallback mechanisms.
Error Logging - The system logs failures with HTTP status codes (504 Gateway Timeout), providing observability for debugging and monitoring.

What The Logs show and how would we debug?

Service A Logs (Success Case):
2026-02-04 14:39:03,020 service=A endpoint=/echo status=ok latency_ms=2

Shows: service name, endpoint, status, and response latency in milliseconds.

Service B Logs (Success Case): 2026-02-04 14:40:25,150 service=B endpoint=/call-echo status=ok latency_ms=45

Service B Logs (Timeout/Failure Case): 2026-02-04 14:41:10,320 #### Failure contacting service A: timeout
2026-02-04 14:41:10,321 service=B endpoint=/call-echo status=error http_status=504 error="timeout" complete_error="ConnectTimeout('max retries exceeded...')" latency_ms=1005

How to Debug?

- **Check Latency** - If latency_ms is ~1000ms, a timeout occurred

- **Monitor Status Codes** - "status=ok" vs "status=error" indicates success/failure
- **HTTP Status Codes** - 504 indicates gateway timeout; check if Service A is running
  Exception Messages - Look for "timeout", "ConnectionRefused", or other errors
  Cross-Service Correlation - Compare timestamps across Service A and B logs to trace request flow
- **Health Checks** - Regularly call /health endpoints to verify service availability