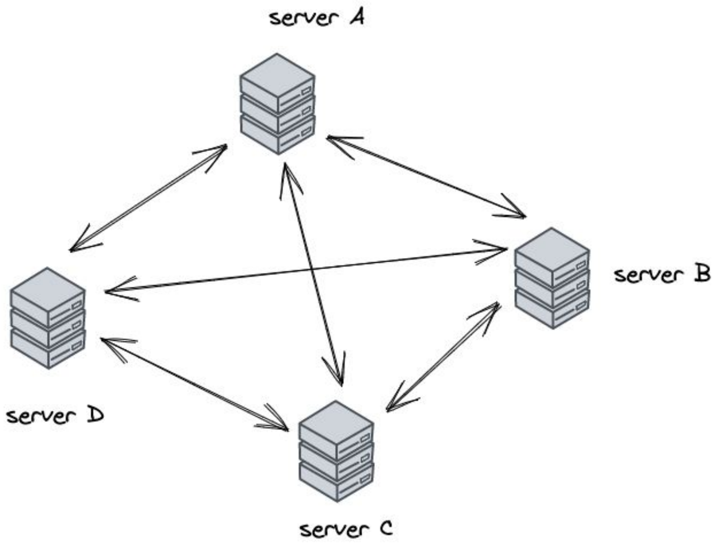
IIT Madras
BSc Degree

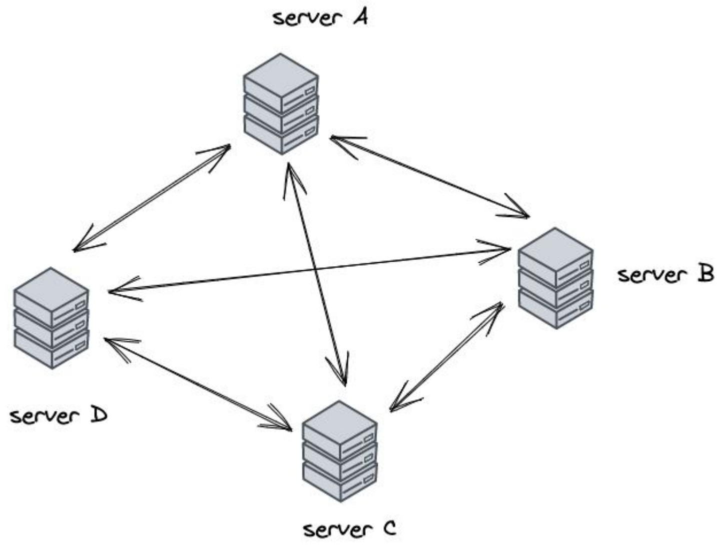
More about messaging

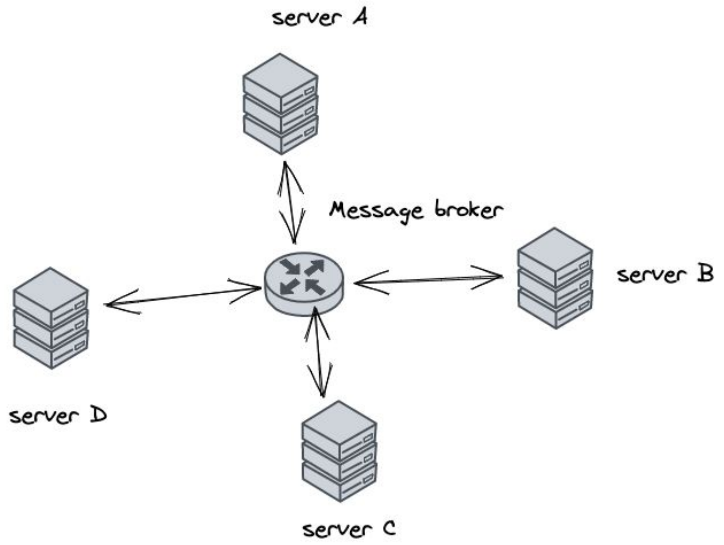
Inter-service messaging

Message Queues - Recap



Message Queues - Recap



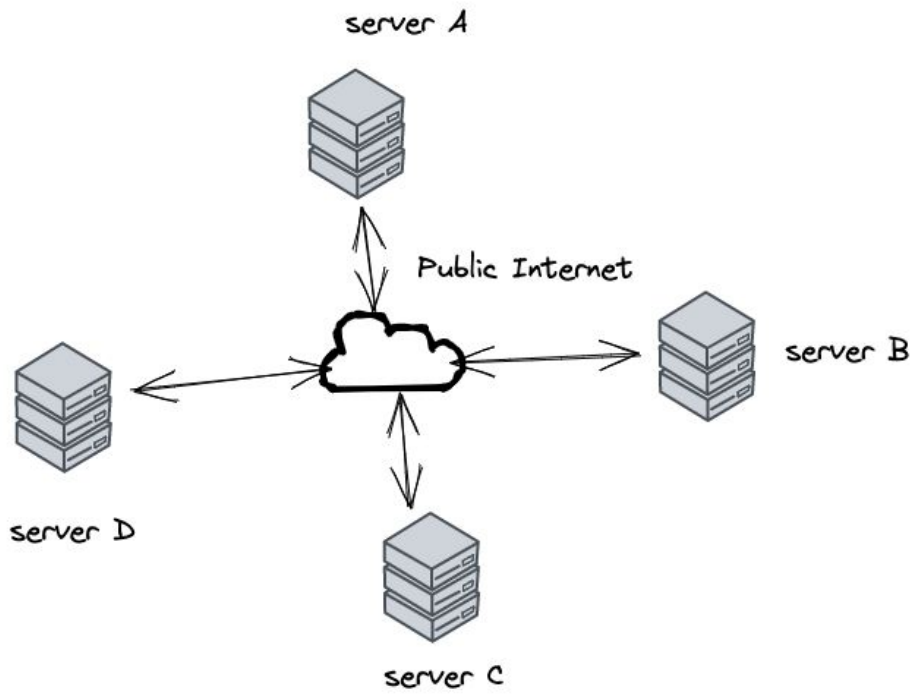


Message Queues

- Multiple services
  - Closely coupled
  - Running on same or closely related servers
  - Example: frontend, email, database, image processing
- Asynchronous message delivery
- Guarantees of delivery
- Ordered transactions

Internet-distributed Services?

- No common message broker
- Servers expose services for public use
- May not need delivery or order guarantees
- Lightweight messaging



Lightweight API calls

- Server exposes certain endpoints
- Meant for others to PUSH messages, not retrieve data
- Request usually through POST, maybe GET
- Data payload may be trivial or even non-existent

Why? - to receive messages from others

Examples

- Every time there is a commit pushed to github, send a message on Google chat room
    - github allows you to register a URL
    - you create a server to receive the request and then push to GChat
- Use Twilio to send several messages
    - Twilio calls you back when done with messages
    - You don’t have to keep checking status from Twilio

Webhooks

- Use the existing web infrastructure to send messages
- Server to server communication
    - Usually… can also be direct client invoking hook on server
- Simpler than message queues

Webhooks

What is a webhook?

A way for an “app” to provide other “apps” with real-time information [1]

- Also called a web callback or HTTP Push API
- Regular HTTP request
  - Uses standard HTTP protocol
  - Usually either POST or GET
- Sometimes called reverse API
  - Similar specifications to regular APIs
  - Usually only to push information, not retrieve data
- Synchronous!
  - No store, retry etc - may trigger other behaviour, but webhook response must be immediate

[1] https://sendgrid.com/blog/whats-webhook/

Example webhook: gitlab

- Setting up a receiver
  - https://requestbin.com/r/enhd2m5q5rmi8
  - https://replit.com/@nchandra/webhooktest#main.py
- Setting up the caller
  - https://gitlab.com/chandrachoodan/webhooktest/-/hooks

Message contents

- Entirely application dependent
- Keep to minimum
  - Not meant for transfer of large amounts of data
  - Only as a message
- Request body

Message response

- Webhooks are “machine called”
  - Invoked by another server, not a human client
- Response should just indicate status
  - 200 for success, 4xx for failures
  - Minimal data returned - mostly will be discarded

Webhooks vs...

- Websockets
  - Real-time 2-way communication vs One way
  - Keep connection open
  - Custom protocol vs standard HTTP
- Pub/Sub
  - Message queue - but public: Google cloud, Apache Kafka, etc.
  - Asynchronous
  - More oriented to message delivery and processing - may be overkill
- Polling
  - Periodic requests to check status
  - PULL, not PUSH
  - Server can be overloaded when client numbers increase
- APIs
  - Collection of endpoints
  - Any client can send request to API
  - Usually used for retrieving data
  - Webhook uses a form of API more suited for pushing messages

How to… ?

- Consume webhook
    - Create URL endpoint, register with provider (eg. gitlab, twilio, …)
- Debugging
    - requestbin: dummy endpoint to receive data and see content
    - curl or postman: API debugging tools
    - ngrok: public URL endpoint for private code without needing public IP
- Securing webhook
    - Restrict at IP access level? Difficult for public facing
    - Use API key/access token/header: eg. X-Gitlab-Token header

Push to Client

Message targets

- Server to server push
  - Server A requests some messages to be sent
  - Server B calls back webhook after sending all messages
- What about asynchronous push to client?
  - Status updates?
  - Push notifications?

Client-side updates

- Requires persistent connection to/from client
- Pull vs Push
  - Client can pull updates
  - Server can push updates
- Original HTTP spec provided no support for this
  - Connection is stateless
  - Client-side pull easy - always possible with page refresh

Polling

- Client repeatedly sends requests
- Fixed Interval
  - easy to implement at client
  - Server may not have any updates! Unnecessary work
  - Can overwhelm server if too many clients
- Variable interval: long poll
  - Server blocks until it has something to update
  - Keep connection open - data sent only when needed
  - Occupies server resources
- https://replit.com/@nchandra/simplechat#main.py
- https://javascript.info/long-polling#demo-a-chat

Server-sent Events

- Mechanism for server to “push” events
- Requires WebWorker (service worker) on client
- Service worker can continue running in background
- Receive update, push to page

Needs more work on server side, but true push possible

Push notifications

- JS Push API
  - https://developer.mozilla.org/en-US/docs/Web/API/Push_API
- Web Push Protocol
  - https://datatracker.ietf.org/doc/html/draft-ietf-webpush-protocol
- Message urgency, priority possible
- Service worker on client receives and updates

Public push notification providers

- Alternatives
  - Firebase Cloud Messaging (previously Google Cloud Messaging)
  - Apple Push
- Authenticated and registered with app
- Web apps vs native apps
  - Web tech (HTTP) vs custom connections (TCP)

Summary: push messaging important part of user experience