INDIAN INSTITUTE OF TECHNOLOGY MADRAS

सिद्धिर्भवति कर्मजा

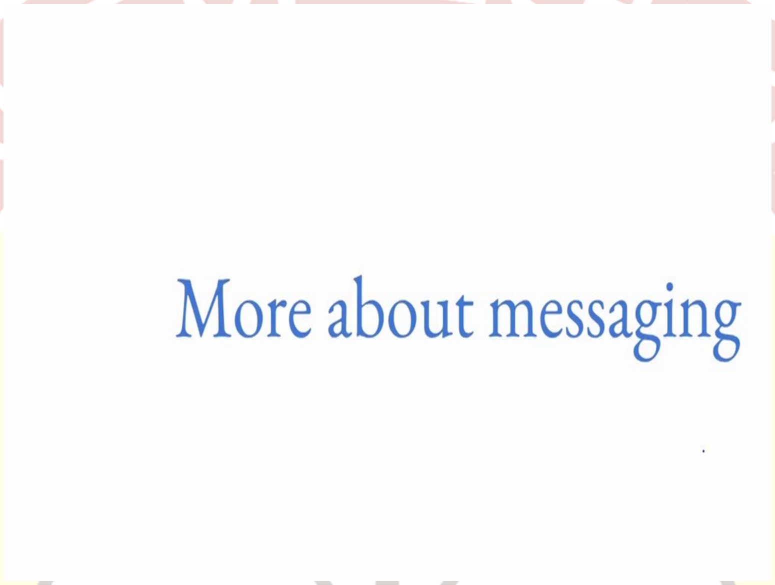
IIT Madras

ONLINE DEGREE

Modern Application Development - II
Professor. Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology, Madras
Inter-Service Messaging

Hello, everyone, welcome to Modern Application Development part 2.

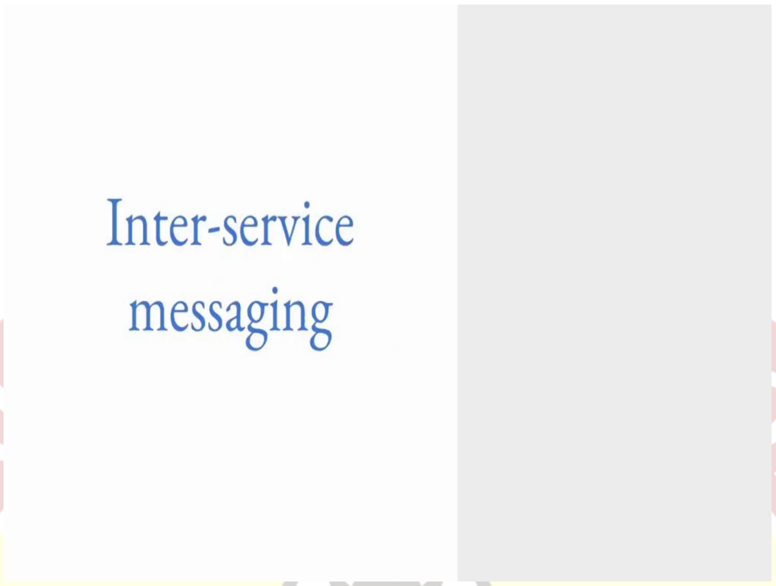
(Refer Slide Time: 00:14)



More about messaging

Hello, everyone. In these lectures, I am going to talk a little bit more about the idea of messaging, and how we use the concepts of messaging in order to communicate between various different processes.

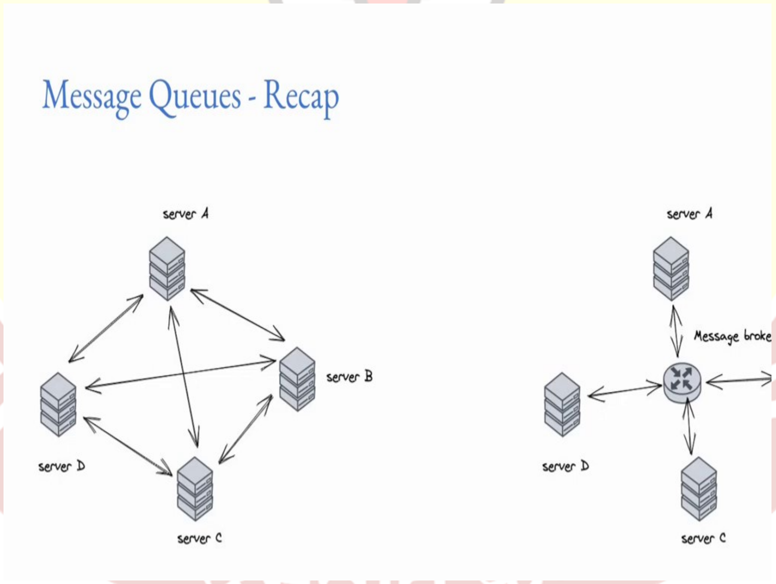
(Refer Slide Time: 00:26)



Inter-service messaging

In particular, what I am going to look at is inter service messaging and sort of contrast this with the idea of inter task messaging, which we had already looked at with the idea of task queues.

(Refer Slide Time: 00:39)



Message Queues - Recap

So, we have already had a discussion of the basic idea of message queues and how a message queue can in turn be used to implement something called a task queue. And the idea behind the task queue is simply that it is something where you can give a set of tasks or functions or operations to be executed, they will all get ordered, according to the message queue. The message queue is responsibility is to make sure that the messages are saved until such time as somebody, some worker process is able to actually execute the tasks.

So, we found that this is particularly useful in a scenario where you have multiple servers that are trying to communicate with each other. It could be something like there is a web front end server, there is a database server, there is an email sending server, there is something else that sends SMS messages, that might be something that handles payment gateways, each of those is a different server, that and all of them put together are finally constituting your application.

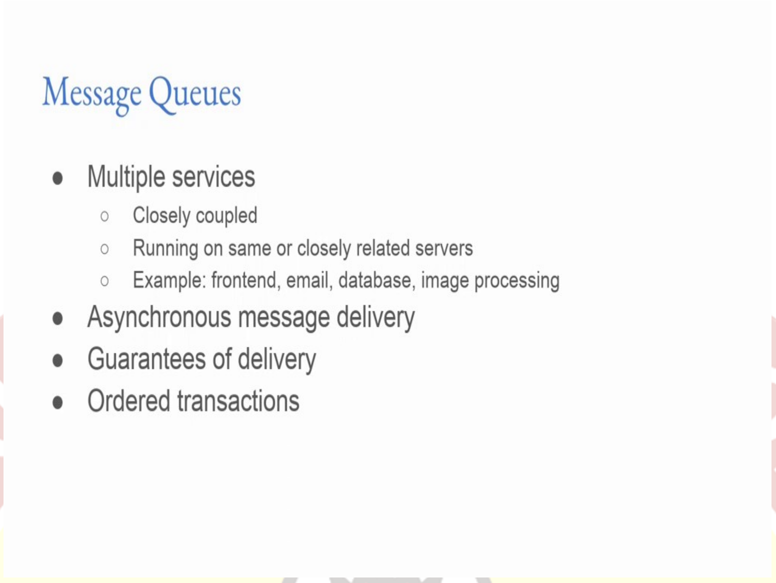
So, they are all quite tightly coupled in the sense that unless all of them are operating together, you do not have a working application. In this scenario, what we said was, rather than having all these point to point connections, have a message broker somewhere in the middle, whose job is to accept messages from various different sources, and make sure that they get delivered to the right destinations.

Which means that from server A, for example, if I wanted to send a message to send out an email, maybe that is handled by server B. Rather than directly communicating to server B, I insert a message into the my task queue or my message queue, saying that I need to send out an email. And I am sort of using the words message queue and task queue interchangeably over here because what we are doing is that we are using the message queue in order to implement the functionality of a task queue.

So, the message queue is what is fundamental that is what actually exists, there is something that allows you to queue up or create a list of messages. On top of that the added layered functionality that we are putting in is the task queue, which says that by looking at the messages I attach meaning to them, which means that when server B looks at the message queue and sees that there is a message there saying send out an email, it knows it has a task to perform.

So, we are constructing task queues on top of message queues. The point over here is that the message Broker sitting in the middle is able to take care of many functionalities such as automatically retrying until the message goes through properly, ensuring that the messages are delivered to the correct recipients sending it to more than one recipient if required, ensuring that the messages go in the order in which they were received a number of different functions, which we have already looked at in a previous video.

(Refer Slide Time: 03:30)



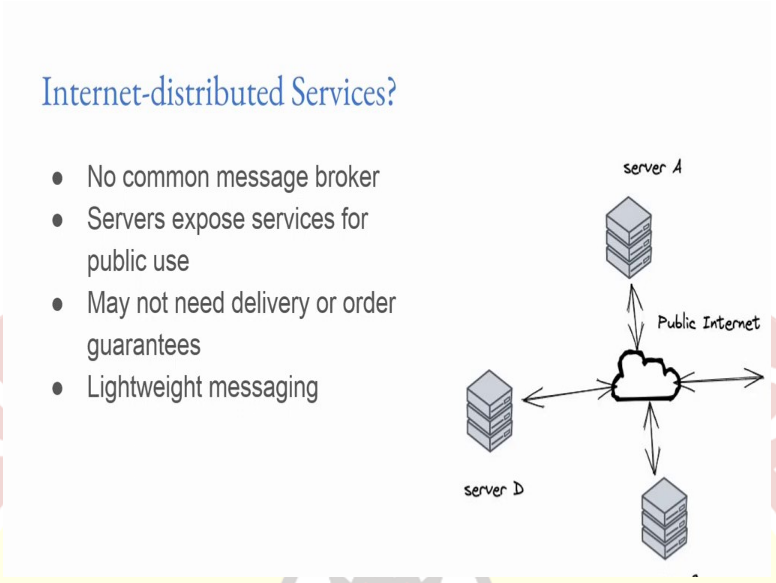
Message Queues

- 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

So, the point out here is just to recap, we have multiple services, they are closely coupled, meaning that I expect all of these services to be functioning properly, in order for my application to be working as expected. Now, some of the functionality that is enabled by using message queues is asynchronous message delivery.

Meaning that I do not need to wait until the message has been accepted by the end user or the email server, before I move on the front end can basically deliver a message and move on. In other words, it is now restricted or the speed at which the front end can move is restricted by how fast the message broker can accept messages, rather than how fast the email server can send out messages. It also provides guarantees of delivery ordering of transactions and so on.

(Refer Slide Time: 04:22)



Internet-distributed Services?

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

But what happens when you move beyond this to so called Internet distributed services. Now, what do we mean by Internet distributed services? From the picture, you can see that it is a very similar kind of scenario I mean, I have just replaced that message broker with something that I am calling the public Internet and the reason why I am putting this over here is I want to consider a scenario where let us say my email sending agent is not an SMTP server that is running on my infrastructure.

Similarly, let us say my database rather than just being something which I run as a MySQL database on another server inside my same data centre might be something which is sitting on Google cloud and it could be that, my, I have something else, which is some image processing workload, which is running on, let us say, the Amazon cloud.

So, what you can see is that there are distinct services running in different data centres, you could potentially use a message broker in order to get data between all of them. But what that means is, you now need to have a publicly exposed message broker, and handle all the authentication and related issues corresponding to that.

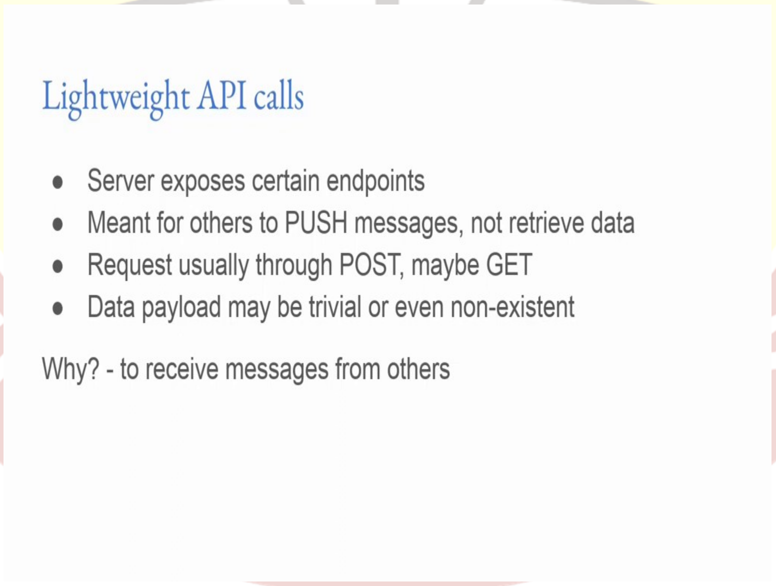
Now, of course, if this is how you are building your application, there is no other way around it. But there are scenarios where you only want something called a lightweight message to be sent. You might not want something which requires like heavy guarantees of certain kind of performance.

An example of that might be let us say that you are using a public service like Twilio, whose purpose is essentially to send out messages. Now, when you use Twilio, it allows you to send SMS messages, WhatsApp messages you can sign up for various different contracts with them and you get the corresponding API keys, which enable you to send a certain number of messages, and you are charged accordingly.

Now, of course, you might want to have a proper message broker to ensure that your messages are sent out. But you might also want to have something simpler, which basically says okay, just send out a message. More importantly, you might also want something which once Twilio is done with sending out a message, maybe they you want them to alert you saying that yes, the task is completed.

Of course, a message queue will solve these problems. The question is there a simpler way of doing? Do you really need a message queue in all of these instances, because as we know, a message queue or a message broker is one more piece of infrastructure that needs to be installed and maintained.

(Refer Slide Time: 06:57)



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

So, the question is, can we get some kind of a lightweight API call and what I mean by lightweight is that you typically do not send a huge amount of data to these endpoints. So, basically, what we are thinking of is a server could expose certain endpoints, either to the public internet or via some kind of authentication mechanism.

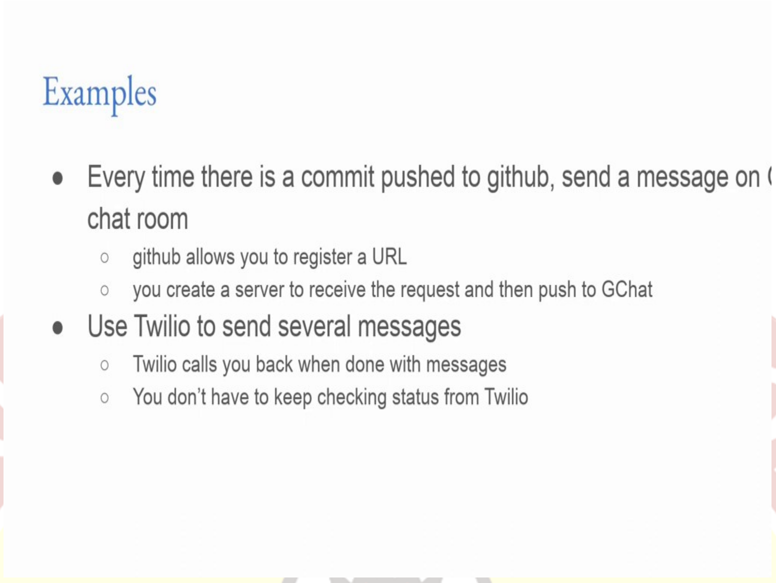
And those endpoints are, whereas a usual rest API, for example, is meant for retrieving data from the server. Usually, what I would do is, I would access a rest API, for example the Wikipedia rest API, one of the main purposes over there is to get articles from Wikipedia. Similarly, when I access the GitHub API, I am usually looking to get information about a user or their repositories or who they are following, and so on.

Whereas over here, what we are doing is the opposite. We want an API that allows people to push messages on to the server that is running the API. So, the process has slightly been reversed. Rather than trying to get information from a server, we are trying to push information onto the server.

Now, this request is usually done through the POST method, rather than occasionally you might use GET, but we are not really going to look at defining a new verb. So, in other words, what I am saying is that we do not really have a push verb in HTTP or if we do, that is not really what we are talking about using over here. We basically want to use HTTP POST requests. But the purpose is to send some kind of a message to the server rather than to retrieve information from there.

Now, why would a server want to do this, it basically enables that server to receive messages from others. So, it exposes an endpoint and you are telling the world that if you connect to that endpoint, and you post some information there, I will get the message. You do not need a message queue, you do not need some other way of ensuring that the message gets through to me. This is a way by which you can directly send a message to whatever service I am running.

(Refer Slide Time: 09:08)



Examples

- Every time there is a commit pushed to github, send a message on (
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

So, let us say, you know, let us consider a couple of examples and these are examples that are actually used in practice. One of them which is commonly used in, especially in large company deployments, where there are a large number of users who are working on a given project. What happens is usually that there are multiple people responsible for different parts of a common repository.

So, maybe the repository is on GitHub and what you have is there is also some kind of a chat room, sometimes mailing lists, but these days chat rooms are more popular and in fact, one of the most popular chat rooms would be slack. But you could also have Google chat, assuming that you are all on the entire Google service. You could have Microsoft Teams. There are a few other chat systems that are available out there.

The important point is, once you have a chat room set up, what you would like to have is, anytime somebody pushes a commit onto GitHub, just drop an alert into the chatroom so that everybody who's working on the project knows that there has been some progress. Now, how do I automate this? I do not want the person who committed into GitHub to have to go and also drop a message in the chat room.

Instead, I could just have it such that whenever I push a commit to GitHub, GitHub automatically takes some information about that particular commit, and posts it into the chat room. How does this happen? GitHub does not know whether you are using Google Chat or WhatsApp, or Slack or any other messaging mechanism.

Instead, what it provides you is that it allows you to register a single URL or multiple URLs. And what you can do with that is that as soon as you push a message onto GitHub, GitHub also makes an HTTP request to that URL. What does that do? It basically triggers a URL call and if you have a server at the back end that is listening for those URL requests, it can do anything at once with the what it has received from the URL.

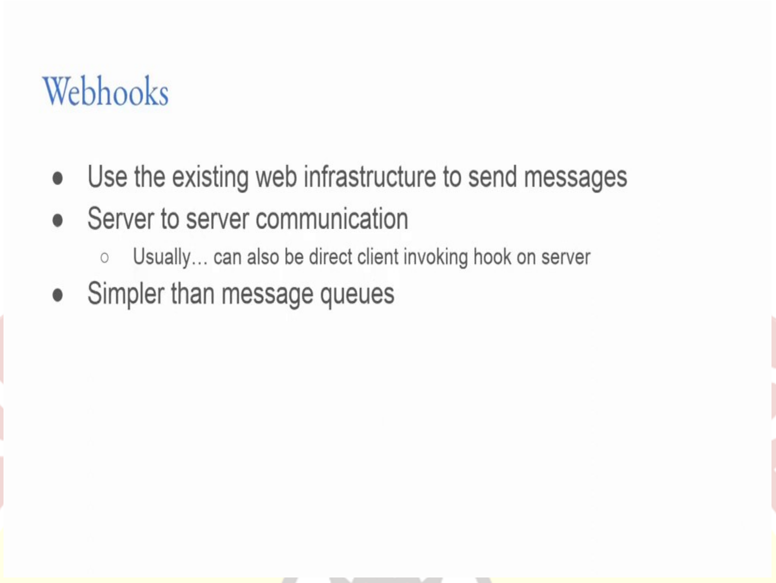
So, which means that you need to create another server, which is again publicly exposed or at least reachable from GitHub, in order to actually receive the incoming request and it is your function, maybe a Python programme or a programme written in some other language that actually takes care of executing the functionality that is needed that is pushing the message to G chat.

So, GitHub, in other words, does not care or know how to directly send a message to Google Chat. But it provides you with some mechanism whereby you can call your own function that can then send a message. Now, similarly, the Twilio messaging system, it allows you to send messages, you could have something where you have bulk messaging being performed. And one simple way of doing this is that you deliver a set of messages to be sent out.

Now, how do you handle this? Should you wait until Twilio acknowledges each and every message? Or can you just dump all the messages on Twilio and wait for information when Twilio is done, sending out all the messages. One simple way of doing this is that once again, Twilio accepts all your messages, says that it will send them out takes its own time, because after all, they have to spread it out. If they try sending too many messages in bulk, they might get blocked by other services.

And once they have done all that they can make a call back. How do you make a call back? Once again, you register a URL with Twilio and Twilio just invokes that URL, it does not care what is actually happening with that URL, it just makes a call to the URL with some information about what task you had given to do. And by looking at the payload, whatever is being sent by Twilio to that URL, you can actually figure out what needs to be done or rather what was done whether the messages were sent successfully, or there was a problem at some point.

(Refer Slide Time: 13:11)



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

Now, all of this read, the scenario that I have described is essentially something called a webhook. So, webhooks in other words, they use existing web infrastructure in order to send lightweight messages. It is primarily for server to server communication. You could potentially also directly have a client invoking a hook on a server.

When I say client over here, I am specifically talking about a user front end that is a bit less common or on the other hand, you might in a more interesting example, have a sort of lightweight server being run on your client. And the server in turn, actually invokes a hook back onto your client.

Now, that is a bit harder to set up because your client in general does not have a public IP address. So, how do you actually hook back onto the client? But potentially, you could do that. The point is, you can use this to actually just send simple messages without having to set up the entire infrastructure for a message queue.