PubNub

A Comprehensive Guide to the Realtime Technology Stack

A Comprehensive Guide to the Realtime Technology Stack

Apps, products, and businesses are growing more interactive, more data-intensive, and more connected. Everyone, and every product, will be dramatically changed by an “always-on” connection to the Internet, and through it, to every other person and device in the world.

3 billion people now have smartphones, most with virtually unlimited data.

Most people use their phones to read content, watch videos, and send emails. Yet the most exciting use of our phones leverages a blazing fast connection to the Internet.

Think about what has changed in the last decade. People worldwide now use their phones to order taxis, track food delivery, control their lights, and have about 100 apps for chatting with each other. They all have something in common – these apps are all powered by a new layer of technology, realtime technology.

For some novice developers, realtime is more about Internet speed than the design and architecture of the app. If you think of the app as the car and the Internet as the road, it's akin to saying, "give me an empty road and I can race my car like a rocket ship." Achieving top performance in a car is more about the design of the car, not just the road itself.

Challenges of Building a Realtime App

INTERNET IS A BEST EFFORT SERVICE

What's that mean? The underlying protocols of the Internet work in a way that assure that data will reach the other end without any hard guarantees. This is not what you want when it comes to building realtime applications. Messages are valuable, so you need reliable, guaranteed data delivery. What if that message is an ambulance dispatch request or sending a hazard warning in an industrial IoT setting? Every message matters.

MANAGING CONNECTIONS BETWEEN PEERS

Building a realtime application that simply communicates between two users is fairly easy. But expanding the application to connect multiple

users across the globe is challenging. You need to maintain a reliable and secure “always-on” connection between peers.

SCALING IS KEY

Scalability is a challenge you should be thinking about from the beginning. You’ve built an amazing app, but when the userbase grows beyond your expectations, you need to ensure that it can handle an increased load of users and data. How can you best prepare for this?

Thinking in Realtime

A realtime application performs an action against an event, within a fraction of a second. However, the key indicator for realtime performance is the collaboration between apps, such that an event occurring in one app can trigger an action in another app.

USE CASES

Chat

Two users are connected in a mobile chat app. A chat message keyed in by one user is an event which triggers an action to display the message in the other user’s chat app.

Data Dashboard

A user is monitoring the state of a device through a dashboard. A change in the parameter of the device is an event which triggers an update. The value displayed in the data dashboard is updated in realtime.

Geolocation

A bus is sending its current location to a passenger’s mobile device. A change in the location of the bus is an event that triggers a change in the location of the bus icon displayed on the map on the passenger’s mobile device.

CONSIDERATIONS

1. It's About Orchestrating The Communication Between Multiple Apps

A realtime application has multiple app instances that need to sync with each other in a split second to provide the best user experience. So it all boils down to the orchestration of data between the app instances.

2. It's About Manipulating Data In Motion

To achieve peak realtime performance, it is imperative to process the data while in-motion. This means that the data exchanged as part of realtime interactions should not be stored before processing, which can lead to delays.

3. There Is Always A Human Angle

In almost all cases, a realtime interaction has a human angle, because one of the collaborators in any realtime data exchange is always a human. Even in a machine to machine interaction, there is a human sitting at the back, receiving feedback. Hence the human perception of a realtime event is very important.

If you pay attention to the above observations, the common denominator is the time difference between app to app data exchange. To achieve realtime performance this data exchange should happen within a fraction of a second. Although the internet speed and network bandwidth do play a role here, the true essence of a realtime interaction over the internet lies with the application protocol stack that works behind the scenes.

Let's take a dig at building a realtime app with first principles. This approach will help you figure out the challenges along the way. As you overcome those challenges, you will be able to arrive at the perfect combination of the stack components that will form the basis for your upcoming realtime app.

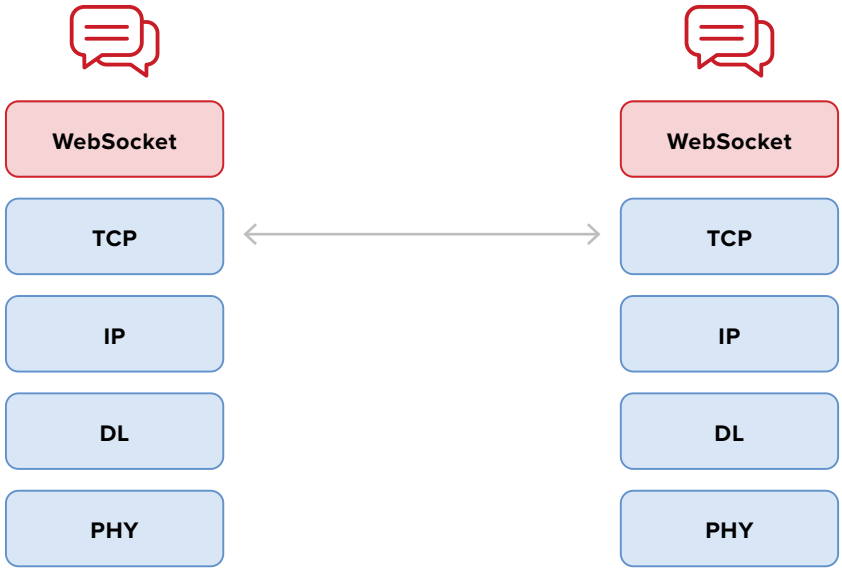
Your First Realtime App

Let's dive into the development process of building a realtime app. A simple in-app chat feature is a great showcase. In its simplest form, two users can chat in realtime via a chat interface.

In this case, you would develop the app on a standard TCP/IP stack and both users will use their individual app instance to communicate over the Internet.

What about the protocol stack for this app?

Since the app instances talk to each other directly, WebSocket is the likely choice here. WebSocket works on the application layer and provides a full duplex communication over TCP.

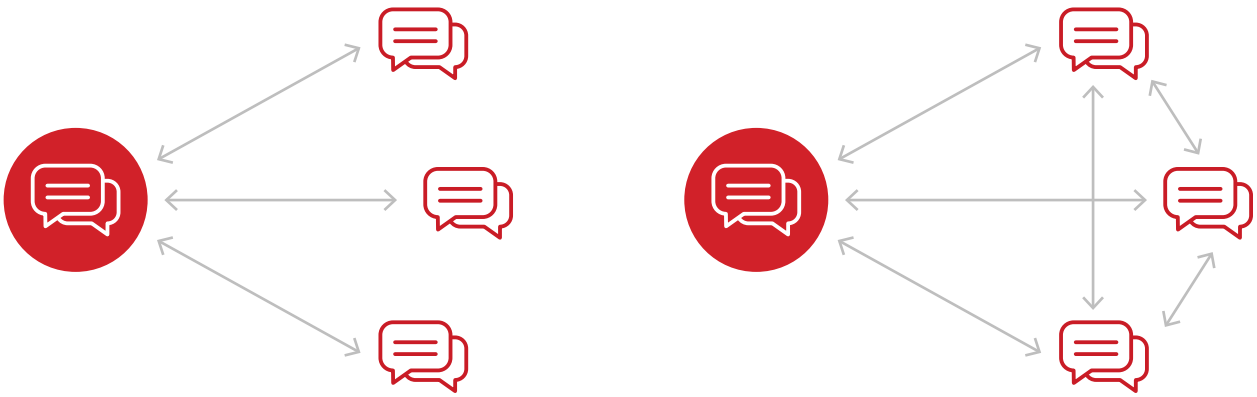


NOTE OF TECH

Socket.IO is one of the most popular libraries for implementing WebSockets. Although all the major browsers now support the WebSocket interface natively, Socket.IO provides a wrapper that provides additional features on top of a WebSocket connection such as fallback and acknowledgment mechanism.

Does the app possess realtime communication capabilities? Perhaps yes, because there is a direct TCP based connection between the app instances. However, if the geographical separation between the two app users is large then the realtime performance will deteriorate.

Without a doubt, this is not the best way to build any chat app since a little bit of scale will progressively make the inter-app communication complex. The impending bottleneck is too many connections to handle for each app.



DEVELOPER'S DILEMMA

Building a 1:1 or 1:N chat app is easy if you are a seasoned programmer. However, in the case of 1:N chat app, a significant amount of time has to be spent in managing the connections between the app instances. As a developer, you will experience some frustrating moments when you aren't able to focus on the app and UI/UX features just because the connection logic isn't working right.

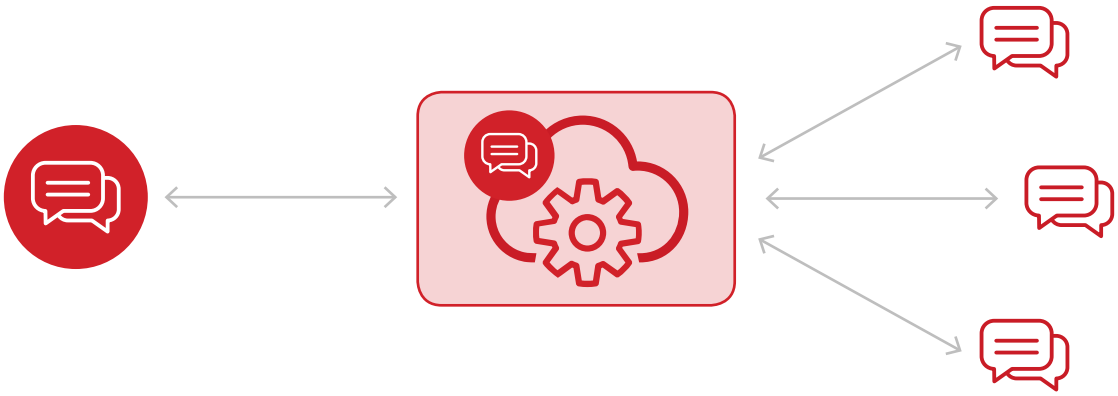
In this case, the app and UI/UX features are what constitute the business logic of the app. The connection management is part of the infrastructure. In terms of the percentage of time spent for developing both these components, there is a significant jump in the later while upgrading the app for 1:N chat.

Building a 1 to 1 Chat App

Building a 1 to N Chat App

Your Second Realtime App: Hosted Services

To handle the challenge of scale, your best option is to have an architecture based on each chat app instance only having to deal with one connection, no matter how many users are involved in the chat session. This can be achieved by defining a new entity, which acts as the centralized chat service component.



With this arrangement, the chat client is simplified. However, the chat service component now has to handle and orchestrate all messages between all the chat app instances.

In terms of the protocol stack, the same WebSocket stack can be retained across all the components including the chat service. However, since the app instances do not have a direct connection anymore, the chat service component has to maintain an active connection with each chat app instance and must move the messages quickly across them to provide realtime performance.

Vertical and Horizontal Scaling

With the revised architecture, it can be expected that each chat service component plays the role of a chat room hosting multiple chat users via their chat app instances.

VERTICAL SCALING

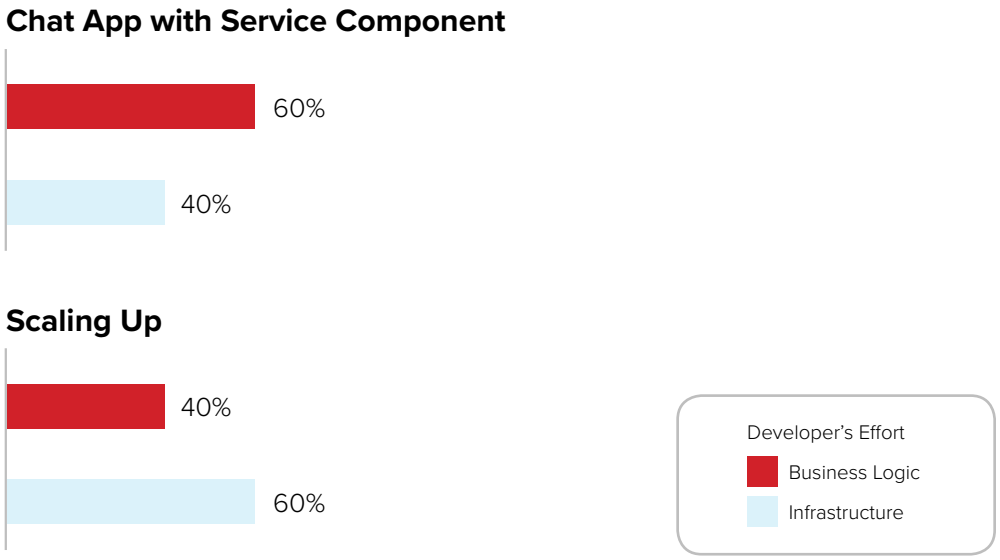
The handling of scale with respect to the number of chat users is now the sole responsibility of the chat service component. Imagine a public chat room where hundreds of users can join or leave in a given moment. This can lead to a significant processing lag to exchange messages via the chat service components.

This leads to vertical scaling of the chat service component, such that more CPU and memory resources are utilized to serve more chat users. Eventually, this has a ceiling beyond which the chat service component can no longer handle anymore uses joining in.

DEVELOPER'S DILEMMA

Even though the connection management at the chat app client has been simplified, there are additional challenges at the chat service component. Coding the service component might seem easy in the beginning but as more and more users signup, managing their connectivity, state and status information becomes a challenge.

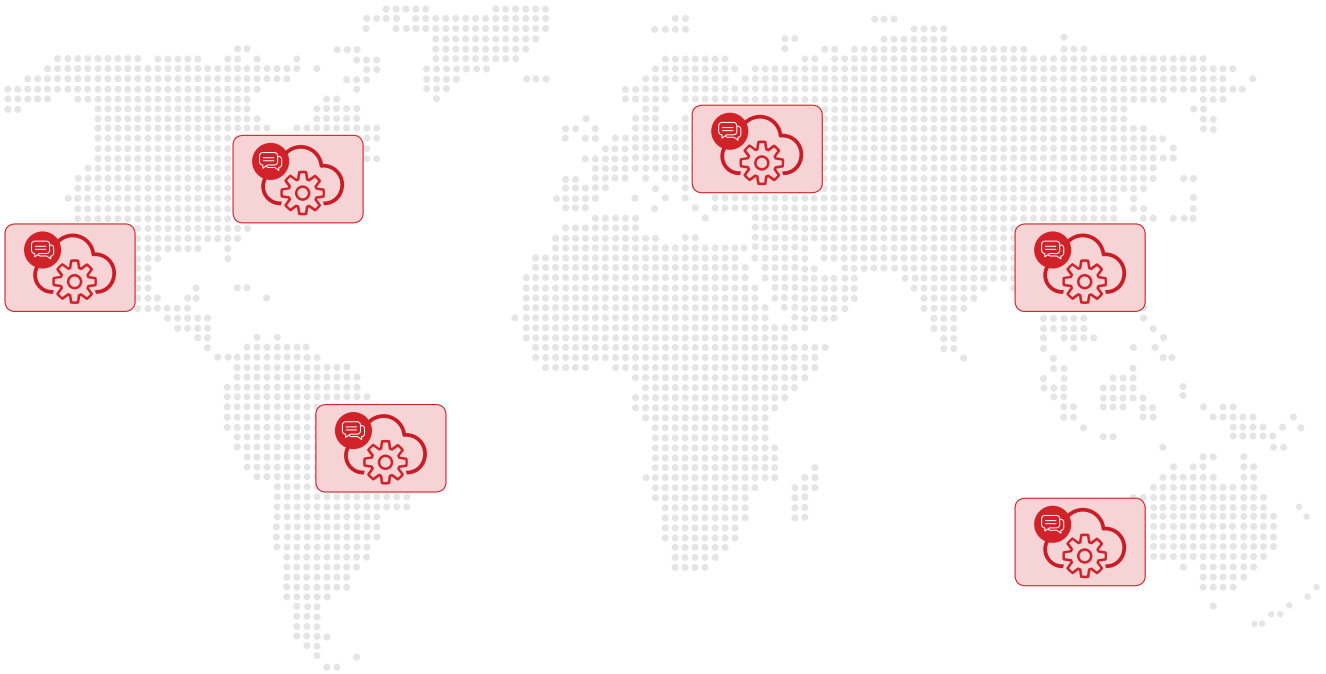
By and by, the effort will be split into building the overall business login and the infrastructure. At scale, the latter will take over. You will find yourself spending more time in ensuring reliability and resilience of infrastructure rather than working on the business logic.



HORIZONTAL SCALING

The other scalability challenge is about the handling of geographically diverse locations from where the users join in.

Realtime performance of the app is impacted if the geographical expanse is not considered. In order to counter this, multiple chat service components need to be spawned at various geographical locations to aggregate the users for a location. This leads to what we know as horizontal scaling.

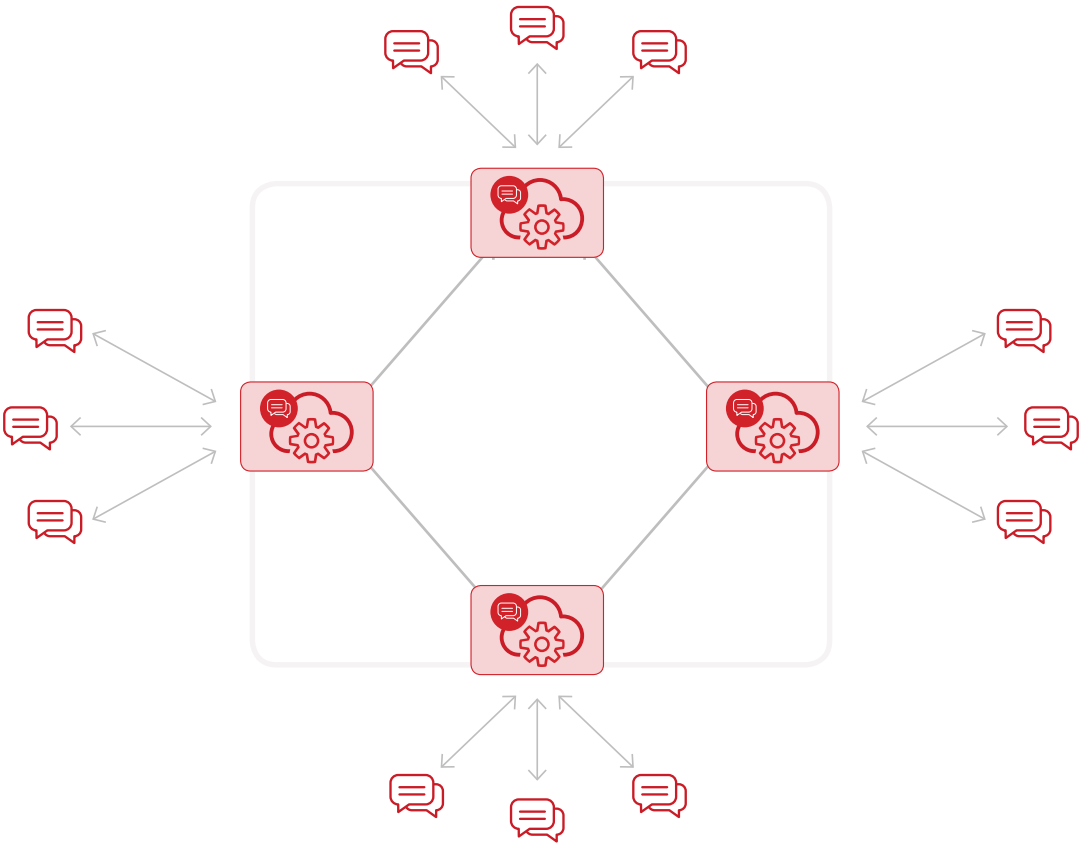


At this point, the architecture of the chat service component has to evolve to handle users from multiple regions, as well as orchestrate with the other chat service components. This is where things become quite complex for you as a developer due to several reasons.

Challenges of Scaling Realtime Apps

1. Managing Users And Orchestrating Between Other Chat Service Components

The chat service component is now distributed across multiple locations such that each chat message is replicated at every location. This adds an additional partition in the topology of the application deployment, such that there is an inner partition which handles the orchestration between chat service components and an outer partition that manages the connectivity with chat app instances.



This brings upon additional complexity because the chat service components have to deal with multifarious messages ranging from users' chat messages to state synchronization between all chat service component instances.

NOTE OF TECH

Handling diverse messages at a scale requires an effective data processing pipeline which provides a queuing mechanism to process messages. There are several message queue libraries available to handle this scenario. Kafka is one of the popular platforms used for building a realtime data pipeline for consuming messages.

When deployed in the above scenario, a bunch of Kafka brokers would sit between the chat service components and provide the data path between the chat service components.

2. Managing The States And World Views Across All Service Components

The problem of orchestrating between the multiple instances of chat service components brings us to the second problem of maintaining a uniform state across all instances. In this application, the main state information is the online status of every chat user. When there was a single chat service component, it maintained the online status for all users.

Now, with multiple components, the online status of all users have to be synchronized across all chat service components. This is a humongous task to achieve as chat service component scales horizontally.

3. Managing The Dynamic Allocation Of Chat Service Components Based On Regional Volumes

For building a scalable system, it is important to consider the load on the service components. If there are too many users logging in from a given location, then a single chat service components alone may not be able to handle that scale due to the limits of vertical scaling. Moreover, a single service component is also not desirable due to a single point of failure.

To mitigate these challenges, a load balancer is required at each location, so that the chat traffic can be spread across multiple chat service components. Above all, a set of protocols need to be defined such that these chat service components can be spawned dynamically based on traffic surge. Additionally, they should sync up with the existing components and be part of the data synchronization.

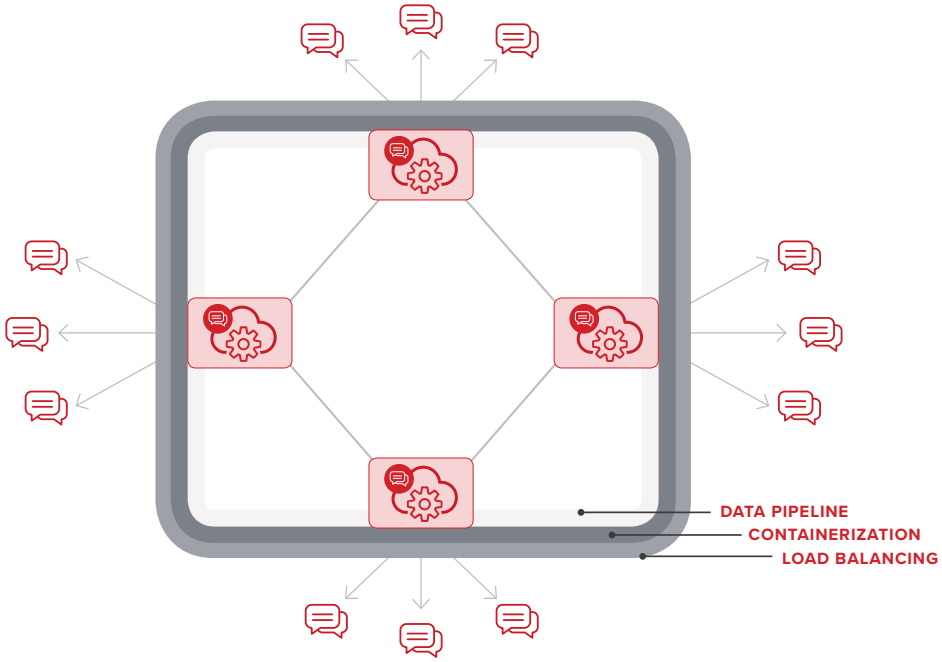
NOTE OF TECH

When deploying an application for scale across multiple geolocations, it is more about elasticity rather than scalability. Based on the user volume, the application should be able to scale up or down to handle the traffic with the help of the requisite amount of computing resources. That's where containerization technology provides a lot of flexibility. A container can be deployed as a standalone execution context within a host operating system/server and has its own isolated file system and resources.

Docker is the most popular container technology available today. A Docker container can be pre-built as an image and executed within a host OS as an independent system with its pre-allocated computing and memory resources.

For managing Docker-based elastic application deployment, there is also a need for a container management system that provides an application-wide view of all the containers. Kubernetes is one such tool that is used for this purpose. Together with Docker, Kubernetes provides a complete suite of tools for management and administration of Docker containers.

With the additional complexity around horizontal scalability, there is a need for additional layers of architectural components to manage the service components over and above the application stack.



4. Security And Access Control

At a global scale, the issues of security and access control become paramount. Forbidding unauthorized users, denying access to hackers or even avoiding DDoS attacks by rogue users is a must. Security is a separate subject itself and for such a vast network, a specialized layer of security protocols needs to reside at the outer edge of the network to thwart any possible breach. All of it adds to the complexity of the inner partition and the chat service components.

DEVELOPER'S DILEMMA

If your application has come this far, then, needless to say, it is popular and is being used by many users from all across the world. Just like the saying goes in software engineering: 80% code is written to check and handle errors, while only 20% code does the real job.

The same logic applies to handling super scalability as well.

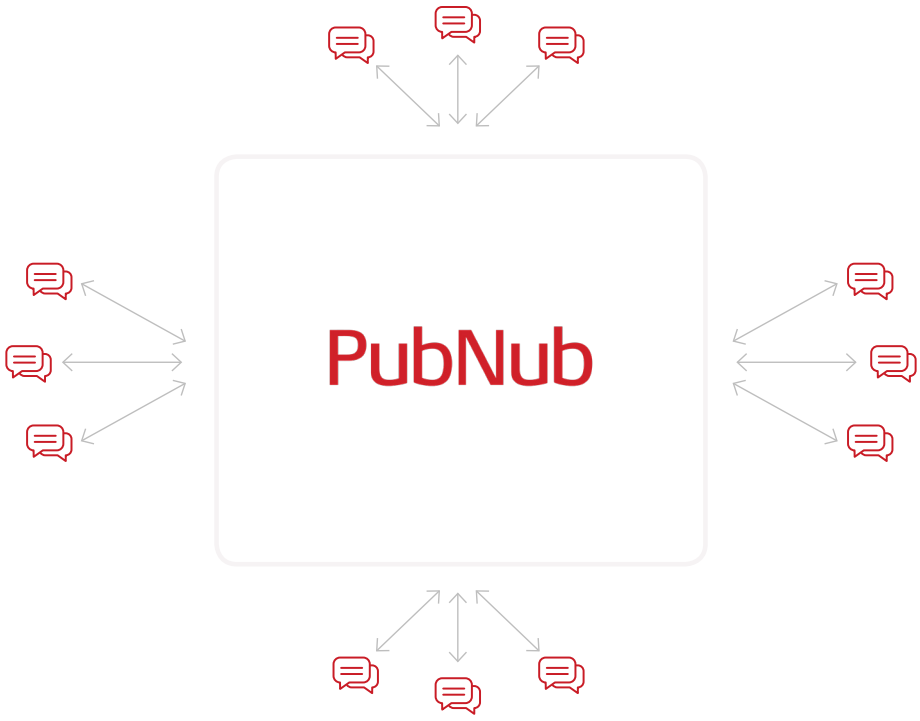
Horizontal Scale Up for Volume & Geo-Distribution

Realtime Infrastructure-as-a-Service

Now it's time to talk about cost – for both setting up and maintaining the infrastructure at scale.

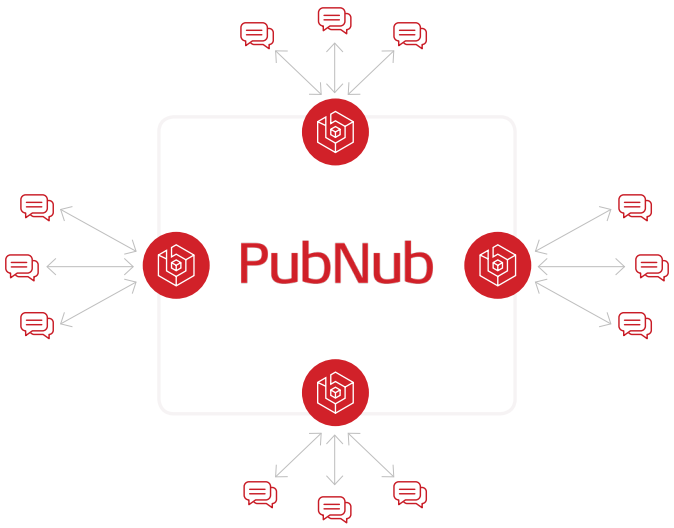
If you include the costs and the developer's efforts in managing a realtime infrastructure for your app, then the situation can get out of hand rather quickly. All this, at the expense of the user experience of your app since the developers would be busy tuning the infrastructure for realtime performance. That's where a specialized cloud service can be a boon which handles the challenges of realtime scaling and expansion.

PubNub's realtime data streaming service solves every problem faced by developers and IT teams in scaling up a realtime application, both in terms of time and cost. With PubNub your app's realtime infrastructure is as simple as this.



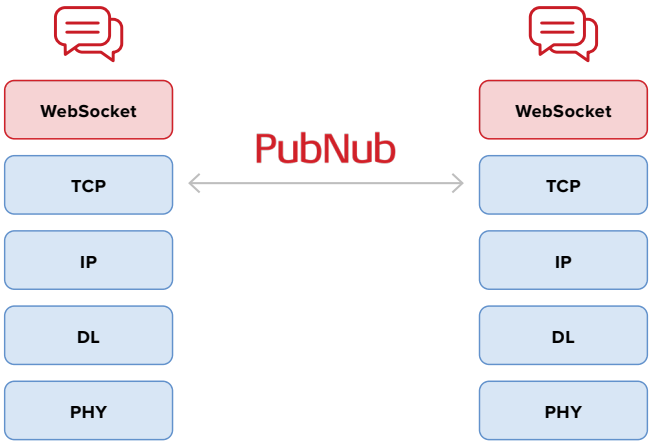
Apart from handling all the heavy lifting associated with handling scale, PubNub guarantees 99.999% of data transmission reach the recipient within a quarter of a second.

The most noteworthy feature of PubNub is that it is not entirely a black box system. PubNub's infrastructure is like a semi-porous white box which allows you to plug in realtime message handling components. Also known as PubNub Functions, these components allow developers to build their custom realtime data processing pipelines on the go. Consequently, PubNub is capable of processing data in motion, rather than at rest, which has significant upside for realtime data transfer.



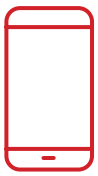
Backed by a globally distributed data streaming network, PubNub handles all your chores related to infrastructure, from scaling to security, access control and more.

And now if you ask, what about the protocol stack?



Yes, PubNub now becomes the de-facto application layer protocol responsible for transmitting your data packets from one app instance to the other. Now, you do not worry about the infrastructure. You only worry about building that all enticing UX for your end users and just plug in the relevant PubNub SDK at your client code to take care of all things in realtime.

DEVELOPER'S DELIGHT



When you use a service like PubNub, which obviates any need to manage the infrastructure, then you focus all your energies in making your application more and more feature rich. The more sparkling features you add to the app, the more kudos you receive from your end users.

And the best part is that this percentage split in the effort will remain more or less constant. No matter how many users you have, PubNub will work silently behind the scenes to deliver your messages in realtime, all the time.

PubNub

www.pubnub.com

(415) 223-7552

@pubnub

Contact Us