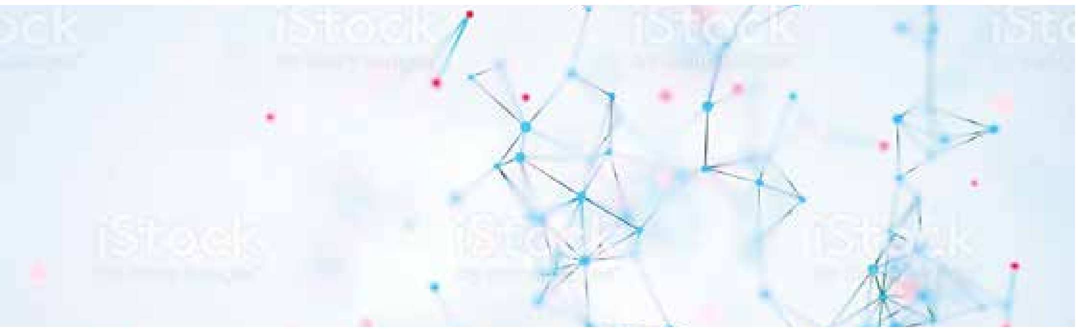

Transforming Instant Messaging Using **Microservices**

To enable instant messaging over a lightweight message bus by implementing the microservices architecture.

BACKGROUND

Instant messaging is the crux of any product/project, and it's essential for any product to have it as an independent entity. But in most cases, it remains cohesive and tightly coupled with the other features of the product. However over the years, there is a seemingly growing demand for every entity, not just instant messaging alone, to be loosely coupled and not have any dependencies.

CHALLENGES

- Although instant messaging was an independent entity, for undergoing any enhancement or testing, the whole product had to undergo similar changes as it belonged to the entire system. This resulted in three things: 1. Scalability wasn't a viable option, 2. testing the entire product wasn't cost effective, 3. any changes made to 'messaging' would lead to changes in the entire product.

- The user experience wasn't particularly exciting because instant messaging platform shared the regular UI that the entire product had. In short, it wasn't as flexible or delightful as Slack, WhatsApp etc.

- There were a couple of deployment complexities because of the existing model. For instance, the lack of velocity, automation, and well-planned release pipelines was a hiccup for quick deployment.

The existing model stayed in the game for a long time now, however, it's not equipped for meeting the current market demand. These hindrances and limitations were some reasons that made us re-architect to modern approaches.

SOLUTIONS

We have already realized that retooling ourselves wouldn't be enough for transforming the way instant messaging worked. It was crucial for us to adopt an evolving architecture that significantly reduced the dependency and brought in velocity.

After understanding the requirements and goals, we chose Microservices, which is an architecture that allows even the largest and complex applications to be composed into two or three services. With microservices, everything can be deployed independently and work as a single entity that focusses on just one task at a time.

WHAT WE DID

- We treated microservices as a product or an application rather than as a service. This mindset alone helped us to isolate the enterprise dependencies while improving the business capabilities. This emerged as a solution for loosely coupling every entity in the product.
- By nature, microservices embraces DevOps which makes continuous delivery and continuous integrations happen. This helped us in creating a release pipeline.
- We created an exclusive roadmap for our MVP, which presented us with the power of controlling the product.
- We applied Spring Cloud, an Open Source tech stack for implementing the microservice space.
- In the next phase of development, we needed to install entities that connect data within the product, but it was also required to resolve enterprise data dependencies. And hence, we used data grids which provided us with the much-needed resilience.
- For instant messaging, we implemented cloud-based data storage. Subsequently which we implemented an event sourcing strategy to publish the domain level events to rest of the enterprise. In this manner, the domain level events are published independently without affecting the entire product.

BENEFITS

Faster and frequent delivery: In the past, we weren't able to speed up the product to market due to which it was time-consuming for receiving market feedback. But with microservices, we were able to reshape it. The continuous and quick delivery fetched us the user feedback, and this seamless access to how users reacted to the product helped us in continuously integrating the enhancements. This was great as we eliminated bottlenecks and gave the users exactly what they want.

Independent release plans: The current environment empowers the product and its delivery in myriad ways, and the release pipeline happens to be just one of the few. The ability to know what must be delivered in the MVP first acts as a catalyst of defining the features right from day 1. So, every feature is released independently and of course, all features are released continuously as the product is always in a ready-to-deploy state.

TAKEAWAY

For one thing, incorporating microservice architecture has kindled a clarity of thought from the initial stages itself. As the developers deal with smaller code bases, it gives them a focussed functionality and a better perspective.

CONCLUSION

After sowing the microservice architecture, we're already reaping the benefits from 'Instant Messaging' which now is versatile, quick, and more adaptable to future scalability.