The heavily centralized ESB pattern can be broken up in this way, and so can the older hub-and-spoke pattern. This change makes each individual integration easier to change independently, and improves agility, scaling, and resilience.

Note: With fine-grained integration deployment, you can change an individual integration with complete confidence that you will not introduce any instability into the environment.

With this approach, you can change to an individual integration with complete confidence that you will not introduce any instability into the environment on which the other integrations are running. You can choose to use a different version of the integration run time, perhaps to take advantage of new features, without forcing a risky upgrade to all other integrations. You can scale up one integration independently of the others, making efficient use of the infrastructure, especially when using cloud-based models.

There are considerations with this approach, such as the increased complexity. Also, although this result can be achieved by using VM technology, it the long-term benefits are greater if you use containers such as Docker and orchestration mechanisms such as K8s. Introducing new technologies to the integration team adds a learning curve. However, these challenges are the same ones that an enterprise already faces if they are exploring microservices architecture in other areas so that expertise might exist within the organization.

We typically call this pattern *fine-grained integration deployment* (a key aspect of agile integration) to differentiate it from other microservices application architectures. We also want to distinguish it from ESB, which is associated with the centralized integration architecture.

Conclusion on fine-grained integration deployment

With fine-grained deployment, you can reap some of the benefits of microservices architecture in your integration layer by enabling greater agility because of infrastructural decoupled components, elastic scaling of individual integrations, and an inherent improvement in resilience from greater isolation. For more information, see 3.2, “Capability perspective: Application integration” on page 54.

2.6.3 Application-owned messaging and events

Synchronous interaction patterns over HTTP are ubiquitous both within the enterprise and across the internet. From an application architecture point of view, assume that the infrastructure for them is present. However, this situation is still not true for asynchronous patterns that use events and messaging.

Asynchronous communication typically requires the storage of state for a period, and someone must own the storage of that state. For these reasons, asynchronous communication is not as commonly available as stateless HTTP communications. Asynchronous patterns must be supported by explicitly installed and managed components, which has been an inhibitor to agility in the past.

It was a relatively expensive and specialist task to configure and administer these asynchronous communication infrastructures. They tended to be centrally owned, and even the provisioning of a new queue or topic might involve specialists. This situation does not work efficiently with a decentralized and agile development where teams want to be self-sufficient.