enable self-subscription and convenient ways to learn and test the interface are critical to the success of any API.

In today's multicloud world, it is also critical to be able to administer, catalog, and secure APIs from a single place even if they are exposed on many different cloud endpoints, which further simplifying the consumer's experience.

Fine-grained application integration

In 2.4, “The three aspects of agile integration” on page 21, you explore why microservices concepts are popular in the application space. You see how those principles can be applied to the modernization of an integration architecture.

The centralized deployment of an integration hub or ESB pattern, where all integrations are deployed to a single, highly available (HA) pair of integration servers, has some benefits in terms of consistency, simplicity, and efficiency. However, centralized deployment might introduce a bottleneck for projects. Furthermore, any deployment to the shared servers runs a risk of destabilizing existing critical interfaces because this deployment is on a single software instance, so no individual project can choose to upgrade the version of the integration middleware to gain access to new features without impacting others.

You could break up the enterprise-wide ESB component into smaller more manageable and dedicated pieces. Perhaps in some cases you can even get down to one run time for each interface you expose. These *fine-grained integration deployment* patterns provide specialized, right-sized containers for improved agility, scalability, and resilience, and are different than the centralized ESB patterns.

Application-owned messaging and events

Asynchronous communication is even more relevant in today's geographically separated distributed solutions:

- Messaging, which was originally introduced to enable decoupled communication across disparate platforms, continues in that mission-critical purpose, but it also now has the same role in reliable communication across cloud boundaries.
- Events provide mechanisms to store an event history, which provides an alternative source of information about data changes and enable applications to listen selectively for notifications, and build local data stores suited to their needs.

However, it is no longer acceptable to wait on a highly specialized team to provision an asynchronous communication infrastructure. Teams must be able to self-provision and configure queues and topics for immediate use as part of their event-driven integration projects.

Templated and patternized mechanisms must be introduced to simplify provisioning tasks, and where possible provide them in as *in-place* multi-tenant managed services. The queue and topic configurations must be application-owned so that they can produce prototypes and iterate on solutions rapidly.