- The *microservices architectural style* very quickly evolved to become the norm, and it provides valuable principles and practices for designing changing distributed applications. Applying these principles lets you create implementations that are optimized for scale, resiliency, and pace of change, which are common requirements for any modern software today.
- *Containers* were very quickly adopted as the standard way of packaging and running distributed applications. Creating modular, reusable containers that are good cloud-native citizens is another fundamental prerequisite. With a growing number of containers in every organization comes the need to manage them using more effective methods and tools. *Cloud native* is a relatively new term used to describe principles, patterns, and tools to automate containerized microservices at scale. We use *cloud native* interchangeably with *Kubernetes*, which is the most popular open source cloud-native platform available today.

In this book, we are not covering clean code, domain-driven design, or microservices. We are focusing only on the patterns and practices addressing the concerns of the container orchestration. But for these patterns to be effective, your application needs to be designed well from the inside by applying clean code practices, domain-driven design, microservices patterns, and other relevant design techniques.

Distributed Primitives

To explain what we mean by new abstractions and primitives, here we compare them with the well-known object-oriented programming (OOP), and Java specifically. In the OOP universe, we have concepts such as class, object, package, inheritance, encapsulation, and polymorphism. Then the Java runtime provides specific features and guarantees on how it manages the lifecycle of our objects and the application as a whole.

The Java language and the Java Virtual Machine (JVM) provide local, in-process building blocks for creating applications. Kubernetes adds an entirely new dimension to this well-known mindset by offering a new set of distributed primitives and runtime for building distributed systems that spread across multiple nodes and processes. With Kubernetes at hand, we don't rely only on the local primitives to implement the whole application behavior.

We still need to use the object-oriented building blocks to create the components of the distributed application, but we can also use Kubernetes primitives for some of the application behaviors. Table 1-1 shows how various development concepts are realized differently with local and distributed primitives.