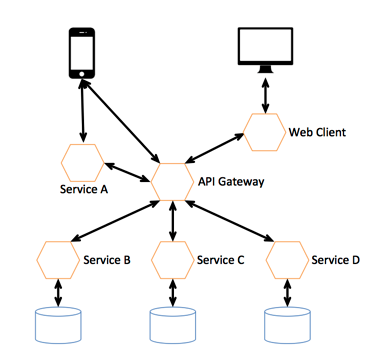
What does an orchestration system need to do? Among other things, it must:

* Handle a large volume of containers and users, simultaneously. An application may have thousands of containers and users interacting with each other at the same time; managing and keeping track of these interactions requires a comprehensive overall system designed specifically for that purpose.
* Manage service discovery and communication between containers and users. How does a user find a container and stay in contact with it? Providing each microservice with its own, built-in functions for service discovery would be repetitive and highly inefficient at best; in practice, it would be likely to lead to intolerable slowdowns (or gridlock), at scale.
* Balance loads efficiently. In an ad-hoc, un-orchestrated environment, loads at the container level are likely to be based largely on user requirements at the moment, resulting in highly imbalanced loads at the server level, along with logjams resulting from the inefficient allocation and resulting limited availability of containers and system resources. Load-balancing replaces this semi-chaos with order and efficient resource allocation.
* Authentication and security. An orchestration system such as Kubernetes makes it easy to handle authentication and security at the infrastructure (rather than the application) level, and to apply consistent policies across all platforms.
* Multi-platform deployment. Orchestration manages the otherwise very complex task of coordinating container operation, microservice availability, and synchronization in a multi-platform, multi-cloud environment.

An orchestration system serves as a dynamic, comprehensive infrastructure for a container-based application, allowing it to operate in a protected, highly organized environment, while managing its interactions with the external world.

Kubernetes is well-suited to the task and is one of the reasons it has become so popular



Microservices