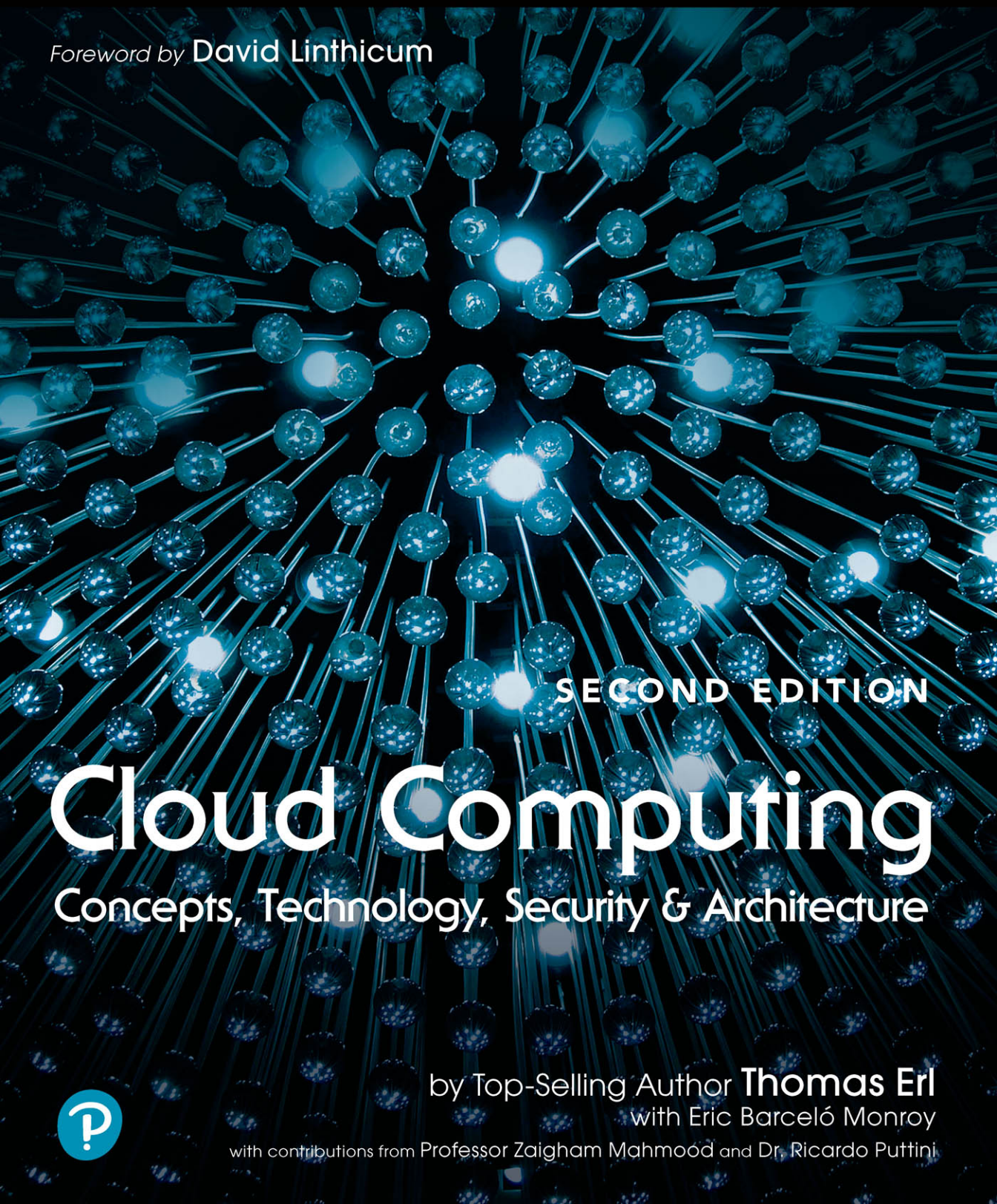
THE PEARSON DIGITAL ENTERPRISE SERIES FROM THOMAS ERL



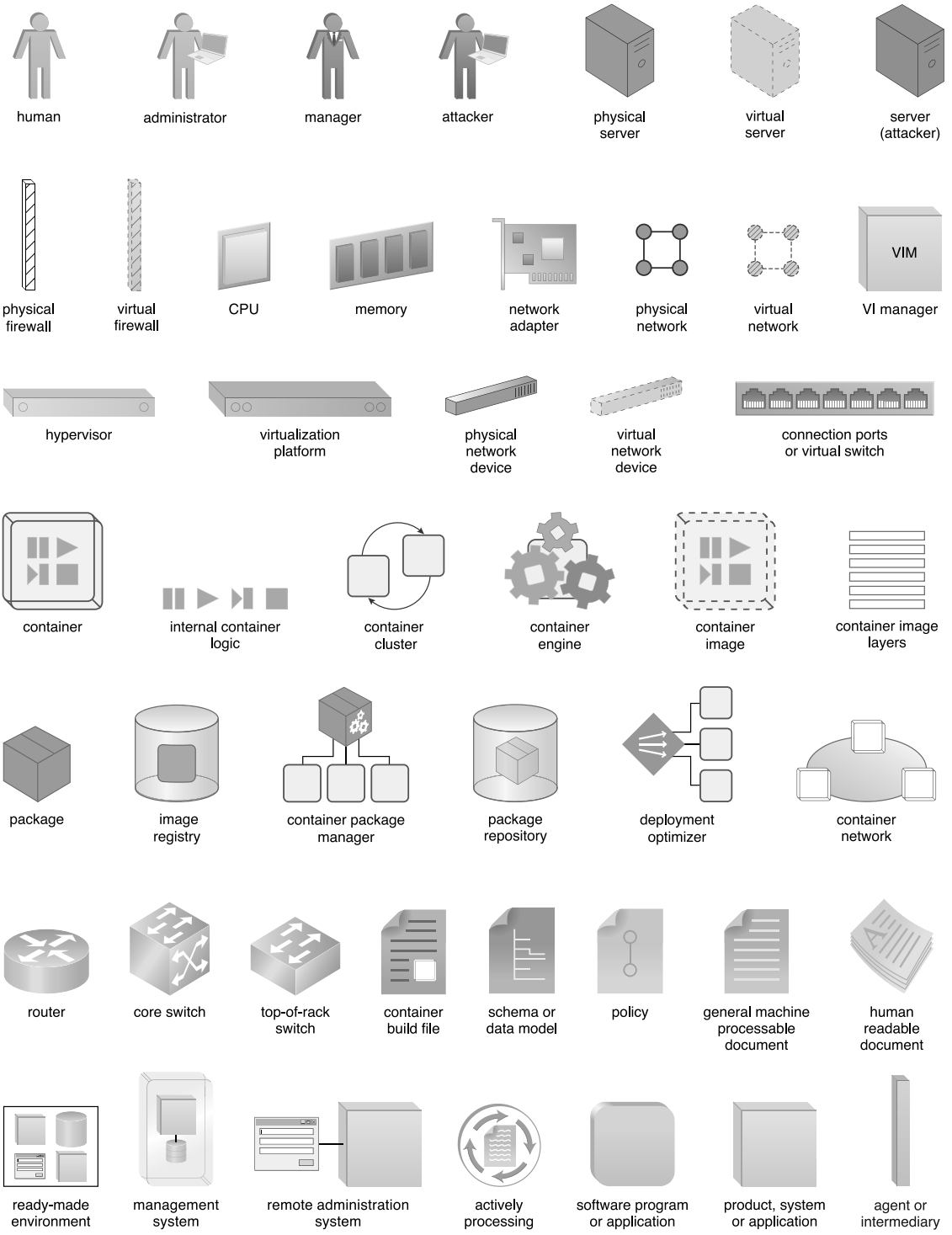
Foreword by David Linthicum

SECOND EDITION

Cloud Computing

Concepts, Technology, Security & Architecture

by Top-Selling Author **Thomas Erl**
with Eric Barceló Monroy
with contributions from Professor Zaigham Mahmood and Dr. Ricardo Puttini


human
administrator
manager
attacker
physical server
virtual server
server (attacker)
physical firewall
virtual firewall
CPU
memory
network adapter
physical network
virtual network
VIM
VI manager
hypervisor
virtualization platform
physical network device
virtual network device
connection ports or virtual switch
container
internal container logic
container cluster
container engine
container image
container image layers
package
image registry
container package manager
package repository
deployment optimizer
container network
router
core switch
top-of-rack switch
container build file
schema or data model
policy
general machine processable document
human readable document
ready-made environment
management system
remote administration system
actively processing
software program or application
product, system or application
agent or intermediary

Container Instances and Clusters

Multiple *instances* of the same container with the same software program can be generated (Figure 6.30). This is usually required when concurrent usage of the hosted software program by multiple consumer programs is necessary. Instances of containers are commonly referred to as *replicas*.

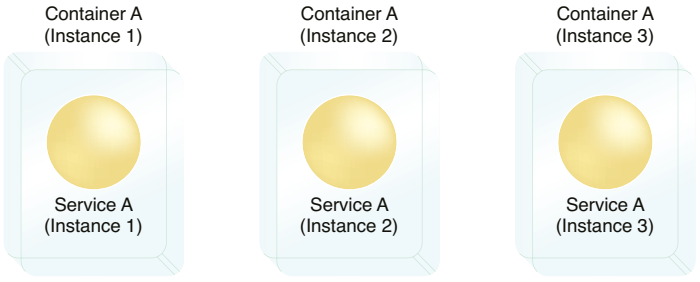


Figure 6.30
Three instances of Container A and its hosted Service A are generated. This allows each instance of Service A to interact with a different consumer program.

Container clusters (Figure 6.31) are pools of container instances that are instantiated in advance of their actual usage. Container clusters can be manually created or automatically generated. They are loaded into memory, where they sit idle, waiting to be invoked. They can be scheduled so that they only reside in memory during predetermined time periods, such as anticipated peak usage times.

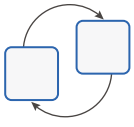


Figure 6.31
The symbol used to represent a container cluster.

Container clusters are primarily created in support of high-performance requirements, often for service-based solutions, to ensure that the containerized service instances can be rapidly provisioned in response to usage demands. Container cluster enviornments can provide auto-scaling capabilities, enabling them to dynamically adjust the size of a cluster based on demand.

Container Package Management

Container package management refers to the process of managing software packages and dependencies within containerized applications. It enables an application and its dependencies to be grouped into a single portable unit called a *package*, which can be deployed on any system that supports the containerization technology.

A *container package manager* is a tool that makes containerized application packaging and distribution easier. It allows container images and their dependencies to be grouped into a single, distributable package that can be deployed and managed across multiple container orchestrators (a mechanism described in the following section).

Figure 6.32
The symbol used to represent a package.

Container package managers typically include a set of command-line tools for creating, tagging, and submitting container images to a container registry, as well as for creating and managing container images and their dependencies. They frequently allow the use of templates or configuration files to define the contents of the package and its dependencies, as well as a method to version and manage the package over time.

Figure 6.33
The symbol used to represent a package repository.

A container package manager is used to coordinate the initial deployment of containers based on predefined workflow logic. The deployment workflow logic is defined in a *package*, also known as a *container deployment file* (Figure 6.32). Typically, host clusters are required to provide a pool of hosts in support of the deployment requirements.

The container deployment file is retrieved from a *package repository* (Figure 6.33).

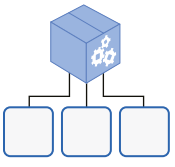


Figure 6.34
The symbol used to represent a container package manager.

The container deployment file is then provided to the container package manager (Figure 6.34).

Before the container package manager carries out the deployment workflow, a special *deployment optimizer* program (Figure 6.35) studies the contents of the package and then assesses available hosts in the cluster to determine the optimal destination for the containers to be deployed.

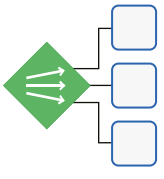


Figure 6.35
The symbol used to represent a deployment optimizer.

Besides the processing capacity of a candidate host, some of the other factors that the deployment optimizer may consider include:

- hardware and software policy limitations
- affinity and anti-affinity specifications
- data locality
- inter-workload interference

Once it has chosen a suitable destination host, the deployment optimizer instructs the container package manager as to where the containers should go. A deployment optimizer can further monitor already deployed containers to ensure their current hosts remain suitable.

NOTE

Within the context of containerization, deployment optimization is often referred to as "scheduling." Furthermore, container package manager and deployment optimizer programs are often limited to deploying containers residing in pods.

Typically, a package represents the containers that comprise an entire solution. Container package managers are therefore created for a set of related containers. In this sense, the package repository can provide a means of application version management.

Examples of what is defined in a package include:

- which host a given container will be deployed on
- which pod a given container will be deployed in
- what sequence a set of containers are deployed in

The administrator authors a package, stores it in the package repository, and then assigns it to the container package manager when it is time for the containers to be deployed (Figure 6.36).

After the deployment, packages are still usually kept in the package repository, as they are often reusable. For example, if a set of containers should need to be ported to a new host, the same container deployment file could be revised with the new host information and then reused.

Docker Compose and Helm are some popular container package managers. These tools make it easier for developers to deploy and manage containerized applications on a variety of container orchestrators (described in the following section) by simplifying the packaging and distribution of containerized applications.

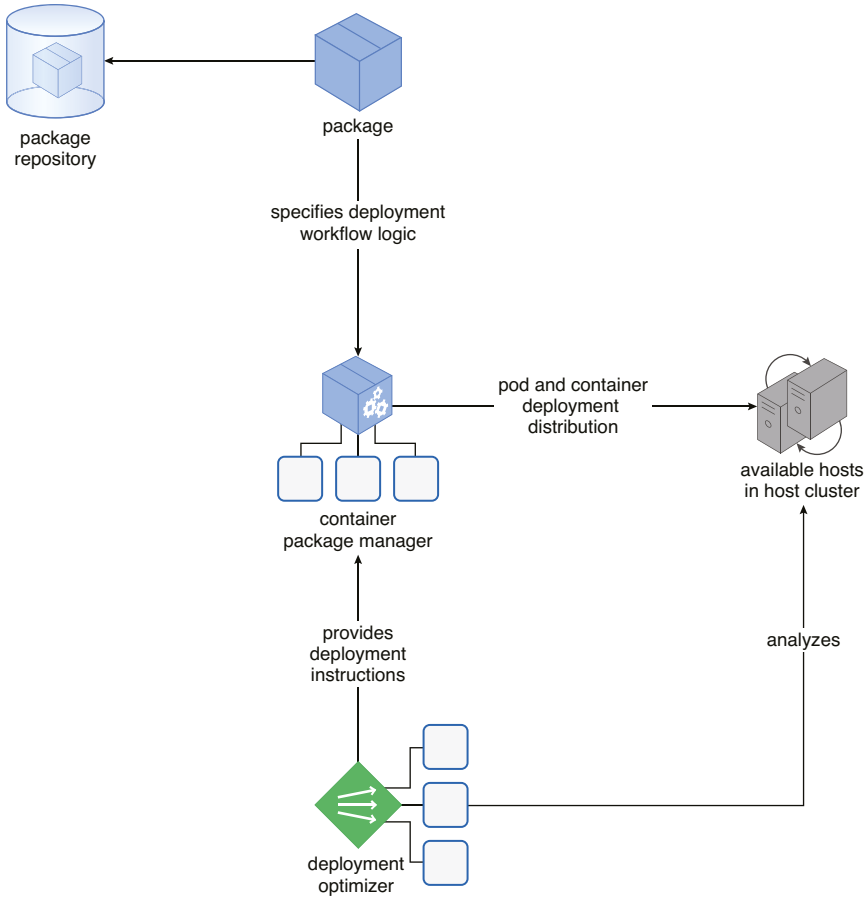


Figure 6.36

The container package manager coordinates the deployment of containers, as per the deployment workflow logic provided in the package and the host deployment instructions it receives from the deployment optimizer.

Container Orchestration

The process of automating the deployment, scaling, and management of containerized applications in a distributed computing environment is known as *container orchestration*. It entails the use of a *container orchestrator*, also referred to as a *container orchestration tool* or *container orchestration platform*.

A container orchestrator performs a wide range of operations in a distributed computing environment. These are some of the key operations performed by a container orchestrator:

- *Container Deployment* – A container orchestrator deploys containers across multiple nodes in a cluster, ensuring that the containers are properly configured and networked.
- *Load Balancing* – The orchestrator distributes traffic across multiple containers running the same application, helping to ensure high availability and scalability.
- *Scaling* – The orchestrator automatically scales up, down, in, or out the number of containers running an application based on demand, helping to ensure optimal resource utilization and cost efficiency.
- *Health Monitoring* – The orchestrator monitors the health of containers and can automatically restart failed containers or replace them with healthy ones.
- *Service Discovery* – The orchestrator maintains a service registry, allowing applications to discover and communicate with each other across the network.
- *Storage Orchestration* – The orchestrator manages the persistent storage needs of containers, ensuring that data is stored and retrieved correctly.
- *Network Orchestration* – The orchestrator manages the networking needs of containers, providing each container with a unique IP address and routing network traffic between containers.
- *Configuration Management* – The orchestrator manages the configuration of containers and can automatically apply changes to running containers.

A container orchestrator typically consists of several components that work together. Some of the key components of a container orchestrator are:

- *Container Runtime* – Responsible for running and managing containers on each node in the cluster.
- *API Server* – Provides a central interface for interacting with the orchestrator. It accepts API requests from clients and communicates with the other components of the orchestrator to perform the requested actions.
- *Scheduler* – Responsible for deciding which node in the cluster to deploy a new container to, based on factors such as resource availability and workload balancing.

- *Controller Manager* – Responsible for managing various controllers that automate different aspects of the containerized application lifecycle, such as scaling, replication, and health monitoring.
- *Distributed Key-Value Store* – Used by the orchestrator to store configuration data, service discovery information, and other metadata.
- *Networking* – A component that provides the necessary network infrastructure to allow containers to communicate with each other across the cluster, including routing and load balancing.
- *Storage* – A component that manages the persistent storage needs of containers, including providing access to shared storage resources and ensuring data integrity.

The basic steps involved in container orchestration are:

1. *Create a Container Image* – Developers create a container image that includes their application code and all of its dependencies.
2. *Push the Image to a Container Registry* – The container image is pushed to a container registry, which is a central remote repository of container images.
3. *Define the Application Deployment* – Using a container orchestrator, developers define how the containerized application should be deployed, including the number of replicas, the network configuration, and any storage requirements.
4. *Deploy the Application* – The container orchestrator deploys the application across multiple nodes in a cluster, ensuring that the desired number of replicas are running and that the application is accessible to users.
5. *Monitor and Manage the Application* – The container orchestrator monitors the health of the application, automatically scaling it up or down as needed, and rolling out updates and patches without causing downtime. It also provides logging and monitoring capabilities to identify and troubleshoot any issues that arise.
6. *Manage Multiple Applications* – The container orchestrator can manage multiple containerized applications simultaneously, ensuring that they are deployed, scaled, and managed according to their individual requirements.