**Phase 5: Deployment — Detailed Overview**

**1. Multiple Servers and Load Balancing (5-multiple-servers-and-load-balancing)**

**Purpose:**  
When your application grows beyond a single server’s capacity, distributing incoming traffic across multiple servers is essential to maintain performance and availability.

**Key Concepts:**

* **Setting up multiple server instances:**  
  Deploying your application on multiple physical or virtual servers ensures that workload is shared. This could involve spinning up several cloud VM instances, containers, or serverless functions running the same app version.
* **Configuring load balancers:**  
  A load balancer acts as the “traffic cop,” distributing incoming user requests evenly or based on defined rules across the available server instances. Popular load balancers include AWS ELB, NGINX, HAProxy, and Google Cloud Load Balancer.
  + **Load balancing algorithms:** Round Robin, Least Connections, IP Hash, etc.
  + **SSL termination:** Offloading SSL/TLS processing at the load balancer.
  + **Session persistence:** Sticky sessions when needed for stateful apps.
* **Ensuring high availability and fault tolerance:**
  + Servers should be deployed across multiple availability zones or data centers to prevent a single point of failure.
  + Health checks regularly monitor server status; unhealthy instances are removed from the pool.
  + Automatic failover mechanisms reroute traffic if an instance fails.

**2. Production Configuration (13-production-configuration)**

**Purpose:**  
Transitioning from development or staging to production demands careful configuration to ensure the application is secure, stable, and maintainable.

**Key Elements:**

* **Environment variable management:**  
  Sensitive information like API keys, database credentials, and config flags should be stored in environment variables or secret management tools rather than hard-coded.
* **Security best practices:**
  + **HTTPS:** Use TLS to encrypt data in transit, often enforced via load balancers or web servers.
  + **Content Security Policy (CSP):** Mitigate cross-site scripting (XSS) and other injection attacks by restricting sources of executable scripts and resources.
  + **Other security headers:** HSTS, X-Frame-Options, X-Content-Type-Options, etc.
  + **Authentication and authorization:** Ensure production-grade identity management and access controls.
* **Monitoring and logging setups:**
  + **Centralized logging:** Aggregate logs from all servers/services for easier troubleshooting and analysis using tools like ELK Stack, Splunk, or CloudWatch Logs.
  + **Application performance monitoring (APM):** Track metrics like response times, error rates, throughput using tools like New Relic, Datadog, or Prometheus.
  + **Alerts and notifications:** Set up automated alerts for anomalies or failures to enable rapid response.

**3. Container Orchestration (17-container-orchestration)**

**Purpose:**  
As containerized applications grow, managing multiple containers becomes complex. Orchestration tools automate deployment, scaling, and management of containers.

**Core Topics:**

* **Defining services and deployments using Docker Compose:**  
  Docker Compose simplifies running multi-container Docker applications by defining services, networks, and volumes in a single YAML file. It’s ideal for local development or small-scale deployments.
* **Scaling containers based on demand:**  
  Orchestrators allow you to increase or decrease container replicas dynamically according to CPU/memory usage or request load, ensuring efficient resource utilization.
* **Health checks and self-healing:**  
  Containers can be configured with liveness and readiness probes. If a container becomes unresponsive or unhealthy, the orchestrator automatically restarts or replaces it to maintain service reliability.

**4. Scaling with Kubernetes (18-scaling-with-kubernetes)**

**Purpose:**  
Kubernetes (K8s) is a powerful open-source platform designed for managing containerized applications at scale, supporting complex deployment patterns and automation.

**Important Areas:**

* **Setting up Kubernetes clusters:**  
  A cluster consists of one or more master nodes managing the cluster state and multiple worker nodes running containerized workloads. Clusters can be provisioned on-premises or using cloud services (e.g., GKE, EKS, AKS).
* **Deploying applications using manifests:**  
  Kubernetes resources are defined declaratively in YAML manifests (Deployments, Services, ConfigMaps, etc.). This allows version-controlled, reproducible deployments and infrastructure-as-code practices.
* **Implementing auto-scaling and rolling updates:**
  + **Horizontal Pod Autoscaler (HPA):** Automatically adjusts the number of pod replicas based on observed metrics like CPU usage or custom metrics.
  + **Rolling updates:** Kubernetes can update the application to a new version incrementally without downtime, gradually replacing old pods with new ones.
  + **Self-healing:** Kubernetes continuously monitors pods and restarts or reschedules them if they fail or crash.