1. (a) & (b)Tools like Kubernetes manage and scale apps by automating deployment and resource allocation. They monitor app health, automatically restart failed components, and scale the number of running instances up or down based on traffic, ensuring high availability and efficient resource use without constant manual oversight.

2. A Pod is the smallest unit, hosting one or more containers. A Deployment manages the desired state for these Pods, like how many copies to run. A Service provides a stable network address for those Pods, enabling other apps to find and communicate with them reliably.

3.A Namespace creates a virtual cluster within a physical one, isolating resources for different teams or projects. A common example is `kube-system`, which houses all the critical system components, keeping them separate from user applications for better organization and security.

4. The Kubelet is an agent running on each node. Its main job is to ensure that containers are running in a Pod as defined by the control plane. You can check the status of all nodes in a cluster using the command `kubectl get nodes`.

5. A ClusterIP service is for internal cluster communication. A NodePort exposes a service on each node's IP at a static port, allowing external access. A LoadBalancer uses a cloud provider's load balancer to automatically route external traffic to the service, offering the most straightforward external access.

1. You scale a Deployment by using the `kubectl scale` command. For example, to set a deployment named 'web-server' to 5 replicas, you would run: `kubectl scale deployment/web-server --replicas=5`. This instructs Kubernetes to adjust the number of running Pod instances to match your request.

7. You update an image without downtime using a rolling update. The command `kubectl set image deployment/my-app app-container=my-app:v2` triggers this. It gradually replaces old Pods with new ones, waiting for new Pods to be healthy before terminating the old ones, ensuring continuous service.

8. You expose a Deployment externally by creating a Service of type `NodePort` or `LoadBalancer`. The command `kubectl expose deployment/my-app --type=LoadBalancer --port=80` creates a cloud load balancer that directs internet traffic to your app's Pods. Alternatively, an Ingress provides more advanced routing rules.

9. The scheduler decides by first filtering out nodes that don't meet the Pod's requirements for resources or labels. It then scores the remaining nodes based on policies like spreading Pods for high availability or packing them for efficiency, finally selecting the node with the highest score.

10. An Ingress acts as a smart HTTP/HTTPS traffic router, directing requests to different services based on the URL path or hostname. A Service is a simpler internal load balancer for network traffic. Ingress handles complex web routing, while Service provides basic connectivity.