The main components of Kubernetes are:

* **The API server**: the central management point for the cluster, which exposes the Kubernetes API and handles all read and write requests.
* **etcd**: a distributed key-value store that stores the configuration data of the cluster, including the state of the objects such as pods and services.
* **The controller manager**: a daemon that runs controllers, which are responsible for maintaining the desired state of the cluster.
* **The scheduler**: a daemon that assigns pods to nodes based on their resource requirements and other constraints.
* **The kubelet**: a daemon that runs on each node, responsible for starting and stopping pods, and reporting the status of the node to the API server.
* **The Kube proxy**: a daemon that runs on each node and manages the network communication between pods and services.
* **The pod**: the basic unit of deployment in Kubernetes, which can contain one or more containers.
* **The service**: a logical abstraction for pods, which provides a stable endpoint for external access to the pods.
* **The namespace**: a way to organize and divide resources in a cluster.
* **The volume**: a way to persist data for pods, which can be backed by various storage solutions.

#### **How does Kubernetes handle container scaling?**

Kubernetes is a container orchestration system that can automatically handle the scaling of containers. It does this by monitoring the resource usage of the containers and adjusting the number of replicas (or copies) of a container as needed.

Kubernetes has several methods for scaling containers, including:

* **Manual Scaling**: Administrators can manually increase or decrease the number of replicas for a given deployment or stateful set.
* **Autoscaling**: Kubernetes can automatically scale the number of replicas based on CPU or memory usage. Administrators can set up autoscaling rules and thresholds, and Kubernetes will adjust the replicas as needed.
* **Vertical Pod Autoscaling**: Kubernetes can automatically scale the resources (such as CPU and memory) allocated to a container, rather than the number of replicas.
* **Horizontal Pod Autoscaling**: Kubernetes can automatically scale the number of replicas based on a metric other than CPU or memory usage, such as the number of requests per second to an application.

Overall, Kubernetes makes it easy to scale containers horizontally or vertically, providing an efficient way of scaling the system to handle the load.

#### **How does Kubernetes handle container failover and self-healing?**

Kubernetes handles container failover and self-healing through a combination of features. Pods, which are the basic unit of deployment in Kubernetes, can be configured with a desired number of replicas. This means that if a pod fails, another replica will automatically be created to take its place. Additionally, Kubernetes has built-in health-checking capabilities which can detect when a container is not functioning properly and take action, such as restarting the container or scaling up the number of replicas.

Furthermore, Kubernetes has a feature called “self-healing” which automatically replaces and reschedules containers that fail, or if the node they are running on fails.

#### **How do you troubleshoot a failed deployment in Kubernetes?**

There are several steps you can take to troubleshoot a failed deployment in Kubernetes:

* Check the status of the deployment by running the command ***kubectl get deployments***. This will show the current state of the deployment and any associated events.
* Look at the pod status by running the command ***kubectl get pods***. This will show the current state of the pods and any associated events.
* Check the logs of the pods by running the command ***kubectl logs <pod-name>***. This will show the logs of the container in the pod, which can help to identify any issues.
* Check the environment variables of the pods by running the command ***kubectl exec <pod-name> env***. This will show the environment variables that the container is using.
* Describe the deployment by running the command ***kubectl describe deployment <deployment-name>***. This will show detailed information about the deployment including the events, conditions, replicas, and more.
* Check for resource limits or constraints by running the command ***kubectl describe pod <pod-name>*** to check for any resource limits or constraints that may be causing the deployment to fail.
* Check for any error message on the status of the pod, deployment, and replicaset
* Check for any network connectivity issues by running the command ***kubectl describe pod <pod-name>***
* If none of the above steps solve your issue, you can try rolling back the deployment to a previous version using the command ***kubectl rollout undo deployment/<deployment-name>.***
* If all the above steps failed, you can try debugging the pod by running the command ***kubectl exec -it <pod-name> — /bin/bash*** to access the pod and check the issue manually.

#### **How do you monitor the health and performance of a Kubernetes cluster?**

There are several tools and methods for monitoring the health and performance of a Kubernetes cluster. Some commonly used tools include:

* **Kubernetes Dashboard**: A web-based UI that allows you to view and manage the resources in your cluster, including pods, services, and replica sets.
* **Prometheus**: An open-source monitoring system that can scrape metrics from Kubernetes clusters and alert on potential issues.
* **Grafana**: A visualization tool that can be used in conjunction with Prometheus to display metrics in a graphical format.
* **kubectl**: The command-line tool for interacting with a Kubernetes cluster, which can be used to view the status of pods, services, and other resources.
* **kubeadm**: A command line tool that helps to bootstrap a minimal viable Kubernetes cluster that you can use as a foundation for your cluster.
* **Kubernetes API-server**: Kubernetes API-server is an entry point to the Kubernetes cluster, it exposes various endpoints to access the Kubernetes objects and cluster information.

In addition to these tools, it is also important to monitor the underlying infrastructure on which the Kubernetes cluster is running, such as the network, storage, and compute resources.