A **Kubernetes Administrator** working with **Azure Kubernetes Service (AKS)** and using **Jenkins**, **Ansible**, and **Terraform** will have a range of responsibilities to ensure smooth cluster operations, continuous integration, and infrastructure management. Here are 20 specific responsibilities:

**Jenkins Responsibilities**

1. **CI/CD Pipeline Integration with Kubernetes**
   * **Responsibility**: Set up and manage CI/CD pipelines in **Jenkins** for deploying applications to AKS. This includes automating the build, test, and deployment processes.
   * **Tools**: Jenkins, Jenkinsfiles, Kubernetes plugin, and Helm charts.
2. **Automating Kubernetes Deployment via Jenkins**
   * **Responsibility**: Automate the deployment of Kubernetes manifests or Helm charts to AKS clusters using Jenkins pipelines.
   * **Tools**: Jenkins pipelines, kubectl, and Helm.
3. **Jenkins Slave Management in Kubernetes**
   * **Responsibility**: Configure Jenkins to dynamically provision Kubernetes worker nodes as build agents using the Kubernetes plugin for Jenkins.
   * **Tools**: Jenkins Kubernetes plugin, AKS, Docker.
4. **Monitoring CI/CD Pipeline Performance**
   * **Responsibility**: Monitor and optimize Jenkins pipelines to ensure high performance, especially for large-scale deployments on AKS.
   * **Tools**: Jenkins monitoring tools, Prometheus, and Grafana.
5. **Containerization and Build Management**
   * **Responsibility**: Use Jenkins to automate the building, testing, and pushing of Docker images for applications that will run on AKS.
   * **Tools**: Jenkins Docker plugin, Docker, Azure Container Registry (ACR).
6. **Automating Rollbacks in Kubernetes**
   * **Responsibility**: Set up Jenkins pipelines to automatically trigger rollbacks if a deployment to AKS fails.
   * **Tools**: Jenkins, Helm, Kubernetes Rollout Strategies.

**Ansible Responsibilities**

1. **Automating AKS Cluster Configuration with Ansible**
   * **Responsibility**: Use Ansible playbooks to automate post-provisioning tasks such as configuring AKS networking, storage, and RBAC roles.
   * **Tools**: Ansible, kubectl, and Azure CLI modules for Ansible.
2. **Application Deployment to AKS using Ansible**
   * **Responsibility**: Automate the deployment of applications and services to AKS using Ansible playbooks.
   * **Tools**: Ansible Kubernetes modules, Kubernetes API, Helm.
3. **Managing Kubernetes Secrets and ConfigMaps**
   * **Responsibility**: Use Ansible to manage Kubernetes secrets, ConfigMaps, and other sensitive data, ensuring secure and consistent configuration of Kubernetes resources.
   * **Tools**: Ansible, kubectl, Kubernetes Secrets.
4. **Infrastructure Management with Ansible**
   * **Responsibility**: Automate infrastructure changes on Azure resources (e.g., VMs, virtual networks, storage) that interact with AKS using Ansible.
   * **Tools**: Ansible Azure modules, Azure CLI, Azure API.
5. **Disaster Recovery and Backup Automation**
   * **Responsibility**: Implement automated backup strategies for Kubernetes objects and persistent volumes using Ansible.
   * **Tools**: Ansible, Velero, Azure Blob Storage.
6. **Kubernetes Cluster Scaling with Ansible**
   * **Responsibility**: Automate the scaling of AKS clusters based on load and resource usage using Ansible playbooks and Kubernetes autoscalers.
   * **Tools**: Ansible, Kubernetes Horizontal Pod Autoscaler (HPA), Azure autoscaling.

**Terraform Responsibilities**

1. **Infrastructure as Code for AKS**
   * **Responsibility**: Use **Terraform** to provision and manage AKS clusters and related resources (e.g., virtual networks, Azure Managed Disks).
   * **Tools**: Terraform, AKS provider, AzureRM provider.

**Story 1: Setup Terraform for Azure Resource Management**

* **As a** Kubernetes administrator, **I want** to set up Terraform to manage Azure resources, **so that** I can automate the provisioning of AKS clusters and related infrastructure.
* **Acceptance Criteria**:
  + Terraform configuration files for Azure resources are created.
  + Terraform initializes successfully with the correct backend.
  + The configuration can deploy resource groups and other basic infrastructure.

**Story 2: Provision an Azure Kubernetes Service (AKS) Cluster with Terraform**

* **As a** DevOps engineer, **I want** to provision an AKS cluster using Terraform, **so that** the Kubernetes cluster is automatically created and managed through code.
* **Acceptance Criteria**:
  + AKS cluster is provisioned using Terraform.
  + Node pools are configured with specified VM sizes.
  + The configuration supports scaling of nodes and network settings.

**Story 3: Configure Virtual Networks and Subnets for AKS Cluster**

* **As a** network engineer, **I want** to configure virtual networks and subnets for the AKS cluster using Terraform, **so that** the cluster can communicate securely within a controlled network environment.
* **Acceptance Criteria**:
  + A virtual network and subnet are created and associated with the AKS cluster.
  + CIDR ranges are correctly configured for the AKS cluster.
  + Terraform successfully deploys the network infrastructure.

**Story 4: Automate AKS Deployment Using Jenkins Pipeline**

* **As a** DevOps engineer, **I want** to automate AKS cluster deployment using a Jenkins pipeline, **so that** the cluster is deployed consistently with every change in the Terraform configuration.
* **Acceptance Criteria**:
  + Jenkins pipeline triggers AKS provisioning using Terraform.
  + Pipeline stages include initialization, planning, and applying the Terraform configuration.
  + Notifications are sent on success or failure of the deployment.

**Story 5: Automate Kubernetes Resource Deployment Using Ansible**

* **As a** Kubernetes administrator, **I want** to use Ansible to automate the deployment of Kubernetes resources (e.g., namespaces, ConfigMaps, secrets), **so that** I can easily manage the configuration of Kubernetes components.
* **Acceptance Criteria**:
  + Ansible playbooks are created for deploying Kubernetes resources.
  + Playbooks can manage namespaces, deployments, and services.
  + Playbooks are integrated with Jenkins for automated execution.

**Story 6: Implement Kubernetes Secrets Management with Ansible**

* **As a** security engineer, **I want** to automate the management of Kubernetes secrets using Ansible, **so that** sensitive data is securely handled across deployments.
* **Acceptance Criteria**:
  + Ansible playbook is created to manage Kubernetes secrets.
  + Secrets are encrypted using ansible-vault or Kubernetes secret management.
  + Playbook is tested and verified with successful deployments to the AKS cluster.

**Story 7: Set Up AKS Cluster Monitoring Using Azure Monitor and Prometheus**

* **As a** Kubernetes administrator, **I want** to configure Azure Monitor and Prometheus for monitoring the AKS cluster, **so that** I can track the health and performance of the cluster and applications.
* **Acceptance Criteria**:
  + Azure Monitor for Containers is set up to monitor AKS metrics.
  + Prometheus is integrated to scrape AKS metrics for advanced monitoring.
  + Dashboards are created for visualizing cluster and pod metrics.

**Story 8: Implement Horizontal Pod Autoscaling (HPA) in AKS**

* **As a** Kubernetes administrator, **I want** to implement horizontal pod autoscaling (HPA) on the AKS cluster, **so that** application workloads automatically scale based on demand.
* **Acceptance Criteria**:
  + HPA is configured on a Kubernetes deployment.
  + Autoscaling triggers based on CPU or memory utilization.
  + Ansible playbook is created to automate HPA setup across multiple services.

**Story 9: Setup RBAC and Secure Access to AKS Cluster**

* **As a** security engineer, **I want** to implement Role-Based Access Control (RBAC) for the AKS cluster, **so that** access to cluster resources is controlled based on user roles.
* **Acceptance Criteria**:
  + RBAC roles and role bindings are configured using Ansible.
  + Azure AD integration is set up for user authentication.
  + Permissions are verified to ensure users have appropriate access levels.

**Story 10: Configure Persistent Storage for AKS Using Managed Disks**

* **As a** Kubernetes administrator, **I want** to configure persistent storage for applications running on AKS using Azure Managed Disks, **so that** applications can store and retrieve data across restarts.
* **Acceptance Criteria**:
  + Persistent Volume (PV) and Persistent Volume Claim (PVC) resources are created.
  + Storage classes are configured for dynamic provisioning of Azure Managed Disks.
  + Ansible playbooks manage storage resources and are integrated with Jenkins for automation.

1. **Automating AKS Cluster Deployment**
   * **Responsibility**: Automate the creation and management of AKS clusters using Terraform modules, ensuring consistency and repeatability across environments.
   * **Tools**: Terraform, AKS, AzureRM provider.

**Story 1: Set Up Azure Authentication for Terraform**

* **As a** DevOps engineer, **I want** to configure Azure authentication for Terraform, **so that** I can securely interact with Azure services to provision AKS clusters.
* **Acceptance Criteria**:
  + Azure service principal is created.
  + Terraform is configured with the correct Azure credentials and access.
  + The connection to the Azure environment is validated.

**Story 2: Create Terraform Configuration for AKS Cluster**

* **As a** Kubernetes administrator, **I want** to create Terraform configuration files for deploying an AKS cluster, **so that** the AKS infrastructure can be managed as code.
* **Acceptance Criteria**:
  + Terraform files (main.tf, variables.tf, outputs.tf) are created.
  + AKS cluster is defined with node pools, networking, and resource group.
  + Terraform configuration is stored in version control (Git repository).

**Story 3: Automate AKS Cluster Deployment Using Jenkins Pipeline**

* **As a** DevOps engineer, **I want** to automate AKS cluster deployment using a Jenkins pipeline, **so that** the cluster is automatically deployed and managed via CI/CD.
* **Acceptance Criteria**:
  + Jenkins pipeline is configured to trigger on changes in the Terraform configuration.
  + Pipeline stages include Terraform initialization, plan, and apply.
  + Successful execution of the pipeline provisions an AKS cluster in Azure.

**Story 4: Configure Virtual Network and Subnets in Terraform**

* **As a** network engineer, **I want** to create and manage virtual networks and subnets using Terraform, **so that** the AKS cluster can communicate securely within an isolated network.
* **Acceptance Criteria**:
  + Virtual network and subnet configurations are defined in Terraform.
  + Subnets are associated with the AKS cluster for pod communication.
  + Terraform applies the network changes successfully.

**Story 5: Implement Auto-Scaling for AKS Nodes in Terraform**

* **As a** Kubernetes administrator, **I want** to implement node pool auto-scaling in the Terraform configuration, **so that** the AKS cluster can scale automatically based on demand.
* **Acceptance Criteria**:
  + Node pool auto-scaling is configured in Terraform with minimum and maximum node counts.
  + Auto-scaling works as expected based on cluster usage.
  + Terraform applies the configuration without issues.

**Story 6: Set Up Role-Based Access Control (RBAC) for AKS Cluster**

* **As a** security engineer, **I want** to configure RBAC for the AKS cluster using Terraform, **so that** access to the Kubernetes cluster is restricted based on user roles.
* **Acceptance Criteria**:
  + RBAC roles and role bindings are defined for users and groups.
  + Azure AD integration for AKS authentication is enabled.
  + Terraform provisions RBAC roles and applies the configuration.

**Story 7: Deploy Kubernetes Resources Using Ansible Playbooks**

* **As a** Kubernetes administrator, **I want** to deploy Kubernetes resources (pods, services, ConfigMaps) using Ansible playbooks, **so that** application configuration is managed and deployed efficiently.
* **Acceptance Criteria**:
  + Ansible playbooks are created for managing Kubernetes resources.
  + Playbooks are tested and successfully deploy resources to the AKS cluster.
  + Playbooks are integrated into Jenkins for automated execution.

**Story 8: Set Up Persistent Storage for AKS Using Terraform**

* **As a** Kubernetes administrator, **I want** to configure persistent storage using Azure Managed Disks and integrate it with AKS, **so that** stateful applications can store and retrieve data.
* **Acceptance Criteria**:
  + Persistent Volume (PV) and Persistent Volume Claim (PVC) are configured in Terraform.
  + Storage classes for dynamic provisioning of Azure Disks are defined.
  + Applications using PVCs are successfully deployed in AKS.

**Story 9: Implement CI/CD Pipeline for Kubernetes Application Deployment**

* **As a** DevOps engineer, **I want** to create a Jenkins pipeline for the continuous integration and deployment (CI/CD) of applications to the AKS cluster, **so that** application updates are automatically deployed.
* **Acceptance Criteria**:
  + Jenkins pipeline is configured to deploy Docker images to AKS.
  + Pipeline includes stages for build, test, and deploy.
  + Successful deployment of the application to AKS is confirmed.

**Story 10: Monitor AKS Cluster with Azure Monitor and Prometheus**

* **As a** Kubernetes administrator, **I want** to set up monitoring for the AKS cluster using Azure Monitor and Prometheus, **so that** I can track the health and performance of the cluster and workloads.
* **Acceptance Criteria**:
  + Azure Monitor for containers is enabled for AKS.
  + Prometheus is deployed in the AKS cluster to monitor custom metrics.
  + Dashboards are created for monitoring cluster and application performance.

1. **Manage Infrastructure Lifecycle with Terraform**
   * **Responsibility**: Use Terraform to manage the entire lifecycle of infrastructure, including creating, updating, and destroying AKS resources.
   * **Tools**: Terraform, Azure DevOps, AzureRM.

**Story 1: Set Up Terraform Backend for Remote State Management**

* **As a** DevOps engineer, **I want** to configure remote state management in Terraform using Azure Blob Storage, **so that** the Terraform state is stored centrally and shared across the team.
* **Acceptance Criteria**:
  + Terraform backend is configured to store the state file in Azure Blob Storage.
  + State locking is enabled to prevent simultaneous updates.
  + Terraform state file can be accessed by multiple team members.

**Story 2: Create Terraform Configuration for AKS Cluster Lifecycle Management**

* **As a** Kubernetes administrator, **I want** to create Terraform configurations that define the entire lifecycle of an AKS cluster, **so that** the cluster can be created, updated, and destroyed as needed.
* **Acceptance Criteria**:
  + Terraform configuration files for AKS provisioning are created.
  + Variables and modules are used for reusable configuration.
  + The Terraform configuration can provision and destroy the AKS cluster successfully.

**Story 3: Automate Resource Group Creation and Deletion Using Terraform**

* **As a** cloud engineer, **I want** to automate the creation and deletion of resource groups in Azure using Terraform, **so that** the infrastructure lifecycle is fully managed through code.
* **Acceptance Criteria**:
  + Terraform can provision resource groups with defined tags and naming conventions.
  + Resource groups are successfully destroyed when no longer needed.
  + Outputs are used to provide relevant information (e.g., resource group names).

**Story 4: Implement Infrastructure Scaling for AKS Nodes Using Terraform**

* **As a** Kubernetes administrator, **I want** to manage the scaling of AKS nodes (manually or via auto-scaling) using Terraform, **so that** node pools can be scaled up or down based on demand.
* **Acceptance Criteria**:
  + Node pool auto-scaling is configured in Terraform.
  + Terraform can update the node count based on resource utilization.
  + Scaling operations are successfully managed by Terraform plans.

**Story 5: Configure Virtual Network Lifecycle Management Using Terraform**

* **As a** network engineer, **I want** to manage the lifecycle of virtual networks and subnets using Terraform, **so that** network configurations are consistent and version-controlled.
* **Acceptance Criteria**:
  + Terraform provisions virtual networks and subnets with appropriate CIDR ranges.
  + Terraform applies changes to existing networks without downtime.
  + Virtual networks can be destroyed and recreated using Terraform without manual intervention.

**Story 6: Manage Terraform Module for Reusable Network and Cluster Provisioning**

* **As a** DevOps engineer, **I want** to create and manage a Terraform module for reusable components like networks and AKS clusters, **so that** infrastructure components can be easily reused across projects.
* **Acceptance Criteria**:
  + A reusable Terraform module for AKS cluster provisioning is created.
  + The module accepts parameters for cluster size, networking, and scaling.
  + The module is version-controlled and used in multiple environments.

**Story 7: Handle Infrastructure Changes Using Terraform Plan and Apply**

* **As a** DevOps engineer, **I want** to ensure that infrastructure changes are applied safely using terraform plan and terraform apply, **so that** changes are reviewed before being implemented.
* **Acceptance Criteria**:
  + Terraform plan output is reviewed and verified before applying changes.
  + Changes to infrastructure (e.g., scaling, network updates) are applied successfully.
  + Any failed changes are rolled back, and a plan to resolve issues is created.

**Story 8: Set Up CI/CD Pipeline for Terraform-Based Infrastructure Updates**

* **As a** DevOps engineer, **I want** to integrate Terraform with Jenkins (or another CI/CD tool) to automate the provisioning and updating of infrastructure based on changes in the code, **so that** infrastructure updates are part of the CI/CD process.
* **Acceptance Criteria**:
  + Jenkins pipeline triggers when Terraform configuration is updated.
  + Pipeline stages include terraform init, plan, and apply.
  + Notifications are sent on success or failure of the pipeline execution.

**Story 9: Manage Kubernetes Configuration Changes with Terraform**

* **As a** Kubernetes administrator, **I want** to manage the lifecycle of Kubernetes-related resources (e.g., ConfigMaps, secrets, ingress) using Terraform, **so that** cluster resources are consistently deployed and updated.
* **Acceptance Criteria**:
  + Terraform manages Kubernetes resources via the Kubernetes provider.
  + ConfigMap, secret, and ingress changes are handled using terraform plan and apply.
  + Changes to Kubernetes resources are applied without disrupting existing services.

**Story 10: Implement Infrastructure Decommissioning with Terraform**

* **As a** DevOps engineer, **I want** to manage the decommissioning of infrastructure using Terraform, **so that** resources are cleanly removed when no longer needed.
* **Acceptance Criteria**:
  + Terraform destroy successfully removes AKS clusters, resource groups, and networking components.
  + Dependencies are handled correctly to avoid orphaned resources.
  + Decommissioning includes logging and reporting on deleted resources.

1. **Version Control for Kubernetes Infrastructure**
   * **Responsibility**: Ensure all infrastructure configurations for AKS are versioned in Git repositories, enabling easy tracking of changes.
   * **Tools**: Terraform, Git, GitHub/GitLab.
2. **Networking and Security Configuration Using Terraform**
   * **Responsibility**: Manage networking (e.g., Azure VNet, subnets, NSGs) and security (e.g., IAM roles, RBAC) configurations for AKS using Terraform.
   * **Tools**: Terraform, AzureRM, Kubernetes RBAC.
3. **Cluster Scaling and Auto-scaling with Terraform**
   * **Responsibility**: Use Terraform to configure and automate the scaling of node pools and other Kubernetes resources in response to load.
   * **Tools**: Terraform, Kubernetes Autoscaler, AKS.
4. **Terraform State Management**
   * **Responsibility**: Ensure Terraform state is managed securely, using remote state storage in Azure Blob Storage, and set up state locking to avoid conflicts.
   * **Tools**: Terraform, Azure Blob Storage, Terraform backend configuration.
5. **Continuous Delivery of Infrastructure Changes**
   * **Responsibility**: Set up **CI/CD pipelines** in Jenkins or other tools to automatically apply Terraform infrastructure changes when configurations are modified.
   * **Tools**: Terraform, Jenkins, GitHub Actions, Azure DevOps.

**Summary of Tools and Technologies:**

* **Jenkins**: Used for automating the CI/CD pipeline, deploying to AKS, and managing Kubernetes resources.
* **Ansible**: Used for automating configuration management, deployment tasks, and operational tasks in AKS.
* **Terraform**: Used for Infrastructure as Code (IaC) to provision, scale, and manage AKS clusters and related Azure resources.
* **Azure Kubernetes Service (AKS)**: The primary platform for managing containerized applications in the Azure cloud.